
Some Representation Learning Tasks and the Inspection of Their Models

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Lukas Pfahler

Dortmund

2022

Tag der mündlichen Prüfung: 28.11.2022

Dekan: Prof. Dr.-Ing. Gernot Fink

Gutachter/Gutachterinnen:

- Prof. Dr. Katharina Morik
- Prof. Dr. Andreas Hotho

Contents

I	Introduction	7
1	Motivation	9
1.1	Contributions	10
1.2	Outline	12
2	A Brief Introduction to Machine Learning	15
2.1	Deep Learning	15
2.1.1	Models	16
2.1.2	Objective Functions	21
2.1.3	Optimizers	23
2.2	Tree-Based Machine Learning Models	27
2.2.1	Decision Trees	27
2.2.2	Random Forest Models	28
II	Designing Machine Learning Tasks	31
3	Related Work on Machine Learning Tasks	33
3.1	Supervised Learning	33
3.1.1	Supervised Learning under Class Imbalance	33
3.1.2	Noisy Label Learning	34
3.1.3	Multi-Task Learning	35
3.2	Representation Learning	37
3.2.1	Auto-Encoders	37
3.2.2	Contextual Embeddings	38
3.2.3	Representation Learning based on Similarity	39
3.2.4	Other Self-Supervised Approaches	43
4	Learning Representations with Unsupervised Tasks	45
4.1	Introduction	45
4.2	Math Search and KDD	46
4.2.1	Representation	46
4.2.2	Similarity Measure	47
4.2.3	Retrieval	47

4.3	The Dataset	48
4.4	Embedding using Convolutional Networks on Bitmap Representations	49
4.4.1	Data Representation	50
4.4.2	The Equation-Encoder Model	50
4.4.3	Empirical Evaluation	51
4.4.4	Conclusion	54
4.5	Embedding using Transformers on Sequential Representations . . .	55
4.5.1	Data Representation	55
4.5.2	Training Transformer Models	57
4.5.3	Retrieval of Equalities and Inequalities	57
4.6	Embedding using Graph Convolutional Networks on Tree Representations	60
4.6.1	Data-Representation	60
4.6.2	Training Graph-Neural-Network for Embedding Formulas .	61
4.6.3	Statistical Analysis	64
4.6.4	Experimental Results	66
4.7	Conclusion and Outlook	71
5	Learning Representation with Multitask Learning and Noisy Labels	73
5.1	A Supervised Multi-Task Representation Learning Approach	75
5.1.1	Supervision by Simulated Data, Data Representation and Pre-Processing	76
5.1.2	Multi-Task Deep Neural Network	77
5.1.3	Experiments	79
5.1.4	Discussion	84
5.2	Supervision based on Noisy Labels with Partially-Known Class-Conditional Label Noise	84
5.2.1	The Significance of Detection as Noisy Label Learning with Partially Known Label Noise	85
5.2.2	Experimental Results on Imbalanced Data Sets	91
5.2.3	Gamma-Hadron Classification with Noisy Labels	95
5.2.4	Deep Gamma-Hadron Classification with Noisy Labels . . .	99
5.3	Summary	102
III	Inspecting Machine Learning Models	103
6	Motivation	105
6.1	Related Work	105
6.1.1	Explanations	105
6.1.2	Robustness	108
6.2	Structure of this Part	110

7	How Does the Model Compute?	113
7.1	Introduction	113
7.2	Method	114
7.2.1	Graph Neural Networks for Trees	114
7.2.2	Forward-Visualization	114
7.2.3	Backward-Visualization	115
7.3	Qualitative Study: Interpretable Search for Formulas	115
7.4	Conclusion and Outlook	119
8	Why Did the Model Learn a Representation?	121
8.1	Related Work	122
8.2	Method	123
8.2.1	Explaining Pre-Activations by the Training History	124
8.2.2	Computing Most Influential Examples	125
8.2.3	Visualizing Contributions in Ridge Plots	126
8.3	Batch-Wise Tracking of Contributions	126
8.3.1	Ghost-Batchnormalization Layers	127
8.3.2	Implementation for pyTorch	129
8.3.3	Experimental Evaluation	130
8.3.4	Explanations for Graph Classification	131
8.4	Example-Wise Tracking of Contributions	137
8.4.1	Method	137
8.4.2	Implementation Details	139
8.4.3	Experimental Evaluation	139
8.5	Conclusion, Discussion and Future Research	144
9	How Robust is a Model?	147
9.1	Introduction	147
9.2	Related Work	148
9.3	Preliminaries	149
9.3.1	Decision Tree Distillation	150
9.3.2	Decision Tree Verification	152
9.4	Random Forest Verification Through Knowledge Distillation	153
9.4.1	Bounding the Output Range of Random Forests	153
9.4.2	Simultaneous Distillation and Verification	154
9.4.3	Some Comments on Asymptotic Runtime	157
9.4.4	Implementation Details	157
9.5	Experimental Evaluation	158
9.5.1	Setup	158
9.5.2	Results	159
9.6	Conclusion	160

IV Conclusion	161
10 Conclusion and Outlook	163
11 Bibliographical Remarks	167
V Appendix	169
12 Open Source Software	171
12.1 malocher	171
12.1.1 Installation	172
12.1.2 Setting up the Malocher-Workers	172
12.1.3 Sample	172
12.1.4 Pitfalls	173
12.2 meticulous-ml	173
12.2.1 Best Practices for Tracking Machine Learning Experiments .	173
12.2.2 Example Python Snippet for meticulous-ml	175
12.3 arxiv-library	178
12.3.1 Installation	178
12.3.2 Pipeline	178
12.3.3 Internal Python Functions	179
12.4 xai-tracking	181
13 Additional Tables and Figures	183
13.1 Mathematical Retrieval Results	183
13.2 Noisy Label Learning	185

Part I

Introduction

Chapter 1

Motivation

MACHINE LEARNING in its cosmos of tasks, models, and methods is as diverse as ever. While traditionally, we have distinguished only supervised learning tasks like classification or regression, and unsupervised learning tasks like clustering or association rule mining [100], today, the field of machine learning knows a wide and diverse range of tasks for a wide range of data modalities. Supervision comes in different varieties, with many sources of supervision in multi-task learning, low-quality supervision in noisy-label learning or even self-supervision, where we algorithmically derive supervision signals from the training examples themselves for unsupervised training without annotations. Classically, data is represented using a fixed set of manually engineered features derived from neat database tables and choosing the best representation was treated separately from training models, through approaches like feature selection or dimensionality reduction. Now, *deep learning* further shifts problem-solving towards the data and tackles problems through increasing data quantities. We now incorporate representation learning into the learning task and trust that a model, equipped with sufficient amounts of training data, learn a suitable representation automatically.

In this data-driven approach, collecting and curating training data becomes more and more important [229, 270, 271]. For instance, the successes of machine learning methods in computer vision tasks like object detection [137] are fueled by the availability of large, labeled data sets, most prominently ImageNet. ImageNet was started by Fei-Fei Li [54], who collected millions of images from the web and used crowd-sourced annotators to group them into well-defined classes derived from the WordNet lexical database [55]. Since then, the use of even larger quantities of cheaper, unlabeled data in conjunction with modern representation learning tasks that do not require labels for choosing useful representations have sparked the development of models with even higher object detection accuracies [295]. A similar trend can be observed in natural language processing. Language models with millions of parameters are trained with large collections of texts obtained from print media as well as the web and achieve remarkable results on various downstream tasks [31, 58, 217]. With unsupervised training, it is crucial to have downstream evaluation tasks for measuring the representational

power of a trained model [154]. For natural language, the GLUE benchmarks for English language understanding [277, 278], a diverse compilation of curated evaluation tasks that require models to exhibit forms of language understanding, have been used to quantify the evolution of models.

Despite the importance of training *and* evaluation data for current machine learning tasks and models [203, 270, 271], Sambasivan et al. note that “*paradoxically, data is the most under-valued and de-glamorised aspect of AI.*” [229]. The decisions taken in the process of collecting and curating data sets for machine learning have significant influence on the resulting machine learning models and their impact, both positive and negative [203]. Bender et al. demonstrate the dangers of training large language models on web crawls with little quality control [14], including how small diversity in the data can lead to discrimination, inequality, and bias in the learned representations. Similar problems have been reported for image classifiers [19, 131]. Pauladda et al. [203] identify another issue with the way we treat data for machine learning: Oftentimes, following the mantra that more data is better, data is reused outside of the context it was collected in. This blind use can lead to models that exploit *spurious cues* or correlations [203] or shortcuts [82] to make decisions.

This highlights the importance of methods for human inspection of the resulting models. To establish trust the decisions of a machine learning model, it is not enough to rely on empirical validation scores like accuracy. In the data-driven approach to representation learning, we also need to establish trust that the model learns to make decisions based on relevant aspects of the data, without exploiting spurious correlations [234] and without amplifying biases [165]. The umbrella term *trustworthiness* spans many research directions taken to establish this trust. Explainable artificial intelligence aims to provide human-understandable explanations for decisions. It is through the inspection of these explanation that we gain understanding of and ultimately trust in our models. Complimentarily, computing adversarial perturbations show us how to induce prediction errors in our models and robustness verification provides formal guarantees that these errors will not happen.

We see that there are problems to address along the full pipeline of machine learning projects, from data collection over prediction task design to the inspection of the trained model.

1.1 Contributions

This work places itself amidst the diverse research landscape of machine learning tasks with all the challenges this entails:

- We design machine learning tasks for applications where we do not immediately have access to neatly-labeled training data.
- We contribute a dataset for learning representations as well as a number of suitable downstream evaluation tasks.
- We develop methods to inspect models and gain trust into their decisions.

More specifically, we first look into the problem of learning representations for the retrieval of related mathematical expressions. To this end, we propose unsupervised or self-supervised machine learning tasks. We train convolutional neural networks, transformer models and graph neural networks to embed formulas from scientific articles into a real-valued vector space. We collect a dataset of over 28.9 million formulas from over 900,000 papers from arXiv.org and represent the formulas in different input formats — images, sequences or trees — depending on the embedding model. We compile an evaluation dataset with annotated search queries from several different disciplines and showcase the usefulness of our approach for deploying a search engine for mathematical expressions. Our embedding approach is resource-efficient: The collection of formulas has to be processed by our neural models only once to obtain embeddings. At retrieval time, we use nearest-neighbor search to obtain search results. The search problem can be solved efficiently using existing index data-structures.

Furthermore, we investigate machine learning tasks in astrophysics. Models are traditionally trained on simulated data, with hand-crafted features and using multiple single-task models. In contrast, we build a single multi-task neural model that works directly on telescope images and uses convolution layers to learn suitable feature representations automatically. Our model has multiple outputs, one for each learning task, which are based on the same hidden representation. This way, we learn representations that are not only useful for a single task but all tasks. We design loss functions for each task and, more importantly, propose a novel way to combine the different loss functions to account for their different scales and behaviors.

We explore another form of supervision that does not rely on simulated training data, but learns from actual telescope recordings. Through the framework of noisy label learning, we propose an approach for learning gamma hadron classifiers that can outperform existing classifiers trained on simulated data. Our approach reduces the dependency on complicated numerical simulations for generating training data, taking the data-driven approach to a next level.

Orthogonally, we contribute methods for the inspection of machine learning models, particularly large, non-linear models that can no longer be understood in their entirety. We investigate three approaches for establishing trust in models.

We propose a method to highlight influential input nodes for similarity computations performed by graph neural networks. We test this approach with our embedding models for retrieval of related formulas and show that it can help understand the output of models.

Next we investigate explanation methods that provide explanations based on the training process that produced the model. This way we can provide explanations that are not merely an approximation of the computation of the prediction function, but actually an investigation into why the model learned to produce an output grounded in the actual data. More specifically, we propose two different forms of explanations, first the output of the most influential training examples for an individual prediction, and

second an aggregated plot that shows the influence of all trainings-examples grouped by class. We propose two different methods for tracking the training process and show how they can be easily implemented within existing deep learning frameworks. Experimental evaluations illustrate the usefulness of our approach, e.g. for image classification.

Finally, we contribute a method to verify the adversarial robustness of random forest classifiers. Our method is based on knowledge distillation into a decision tree model. We bound the approximation error of using the decision tree as a proxy model to the given random forest model and use these bounds to provide guarantees on the adversarial robustness of the random forest. Our robustness guarantees are approximate, but we can provably control the quality of our results using a hyperparameter. We demonstrate the usefulness of our approach in experiments with common tabular datasets and large random forest models.

1.2 Outline

This thesis is structured in four parts. This part, Part 1, proceeds to introduce important basic concepts from machine learning in Chapter 2. These concepts will be used in the remainder of this work and will be familiar to anyone with a scholarly interest in machine learning. While we focus on the building blocks of deep learning in Section 2.1, we also cover other machine learning models in Section 2.2.

The second part of this thesis is concerned with designing machine learning tasks. We begin with a literature review on the machine learning tasks relevant for this thesis in Chapter 3. Starting with tasks that have access to forms of ground truth annotations, we look into classification tasks with high class imbalances, e.g. detection of instances of rare classes, tasks that combine multiple sources of supervision in multi-task learning, and tasks that only have access to noisy versions of the ground-truth learning. Then we proceed to look into unsupervised representation learning tasks that do not have access to annotated training data. For this learning paradigm, we cover embedding learning tasks based on contextual similarity as well as other contrastive learning approaches, but also self-supervised tasks like masking.

In Chapter 4, we develop machine learning tasks for learning to retrieve related mathematical expressions. We base our studies on related work for unsupervised and self-supervised representation learning. We show how we collect a dataset of mathematical expressions in Section 4.3 and proceed to investigate two different data representations, bitmap representations (Section 4.4) and XML-representations, either sequentially (Section 4.5) or tree-shaped (Section 4.6). We investigate different pretraining tasks and propose a number of different evaluation tasks to measure the usefulness of our learned representation models.

Chapter 5 investigates different sources of supervision for machine learning in gamma ray astronomy. We begin by reviewing the three prediction tasks that are cur-

rently tackled in the FACT telescope [6]. In Section 5.1, we propose to use multitask deep network architecture to tackle these machine learning tasks simultaneously in a shared model. Then, in Section 5.2, we investigate the use of noisy label learning for separating gamma particles from hadronic particles, a binary classification problem central in the data analysis pipeline of the FACT telescope. We demonstrate that the noisy learning approach works in conjunction with the deep-learning pipeline developed in the first section.

Finally, Part 3 will approach the topic of trustworthy machine learning from the perspective of the data scientist who is interested in inspecting their trained models, rather than an outsider who is affected by the model’s decisions. Chapter 6 introduces related work on the subfields of trustworthy machine learning most relevant to this part of the thesis: explainable artificial intelligence and robustness.

In Chapter 7, we revisit the idea of searching related mathematical formulas and investigate the use of visual XAI approaches, including a saliency-map approach, to explain nearest neighbor queries of embeddings computed with graph neural networks.

Then, in Chapter 8 we investigate methods that track the training history in order to provide explanations that tackle the problem of explaining “why” a model learned to predict a class. We present two approaches: The first one presented in Section 8.3 can work with any optimizer, but requires training with customized random minibatches. The second approach presented in Section 8.4 can work with standard minibatches, but requires adapting the implementation of the optimizer, which we illustrate for the standard SGD optimizer.

Finally, in Chapter 9, we investigate another aspect of trust, namely robustness. In particular, we investigate adversarial perturbations, i.e. small changes to the input data that flip the prediction of models, in our case random forest models. Analyzing these perturbations may give insights into what causes models to fail and produce wrong predictions. We present an approach based on knowledge distillation from a random forest into a less complex decision tree model. We present the distillation method in Section 9.3.1, then we present existing methods on the robustness verification of trees in Section 9.3.2. We then combine the two methods into a robustness verification method for random forest models in Section 9.4.

We end this thesis in Part 4 that concludes in Chapter 10 with a discussion of the contribution and future research directions. In Chapter 11, we find bibliographical remarks, including discussions of the collaborations that lead to the results presented in this thesis.

Chapter 2

A Brief Introduction to Machine Learning

Deep learning is constructing networks of parameterized functional modules and training them from examples using gradient-based optimization. That's it.

Yann LeCun

MACHINE LEARNING is defined by Tom Mitchell as “the study of computer algorithms that improve automatically through experience” [172]. In this chapter, we will introduce the foundations in the area of machine learning that we will build the rest of this thesis on. We will take the rather unusual road and begin by introducing deep learning, which allows us to introduce many of the terms and definitions used by machine learning in general. Then we widen our scope and also introduce concepts from the remaining areas of machine learning.

2.1 Deep Learning

What started out as a term for training neural networks with more than 3 layers [108], is now an umbrella term for machine learning techniques centered around non-linear models that are composed from differentiable building blocks with differentiable parameters trained using variants of stochastic gradient descent optimization with learning tasks that rely on training data to estimate statistical quantities: *Deep Learning*. In this chapter, we will introduce the most important building blocks of deep learning along with the necessary notation. This chapter is not intended to replace a thorough introduction to deep learning like the excellent textbook by Goodfellow et al. [92].

As hinted above, deep learning requires three components: A model, an objective function and an optimizer. First we review their roles and how these components

interact, then we will cover the fundamentals of each component individually.

Let f_θ denote a function parameterized by a real-valued set of parameters θ . We will call f_θ our *model* and θ are its *weights*. Training the model involves minimizing an *objective function* \mathcal{L}

$$\min_{\theta} \mathcal{L}(f_\theta) \tag{2.1}$$

that often involves a given set of training data. For instance in the classical empirical risk minimization (ERM) for *supervised learning*, we are given a dataset

$$\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

of pairs, where $x \in \mathcal{X}$ is an example and $y \in \mathcal{Y}$ is a desired output of our model called *label*. When \mathcal{Y} is continuous, we speak of *regression* tasks, for countable discrete sets $\mathcal{Y} = \{1, \dots, M\}$ we speak of *classification* tasks. Tasks with $M = 2$ are called *binary classification*, while $M > 2$ is called *multiclass classification*. The ERM objective is

$$\mathcal{L}_{\text{ERM}}(f_\theta) = \frac{1}{n} \sum_{i=1}^n \ell(f_\theta(x_i), y_i)$$

for a *loss function* $\ell \mapsto \mathbb{R}$ that judges the quality of the prediction such that 'better' predictions obtain lower losses. In the next chapters we will see different objectives for different kinds of supervision. These objectives, however, will share a common property: They estimate an expected value by computing an empirical mean; hence they can be decomposed into sums. The optimizers we use to minimize the objective and train the model will exploit this decomposable structure. When f is differentiable with respect to θ and L is differentiable with respect to the outputs of f , we can use gradient-based optimization techniques, most prominently stochastic gradient descent [186]. Intuitively, the gradient $\nabla_{\theta} \mathcal{L}$ of the objective function points in the direction of the steepest ascent of the function in the parameter space. Modifying θ by taking a small enough step in the negative direction of the gradient reduces the objective value. This is the fundamental idea of gradient-based optimization that we will outline in Section 2.1.3.

2.1.1 Models

Out of the many possible ways to introduce the family of deep networks, this work chooses to approach them via their computation graph. This representations allows us to elegantly formalize the mathematics of deep network optimization for a very general set of possible models, but is also very close to the way modern deep learning software libraries implement deep networks.

The Computation Graph

Deep networks are composed from simple building blocks. In its simplest form, the model applies different operations $f_1 \dots f_l$ in a sequence

$$f(x) = (f_l \circ f_{l-1} \circ \dots \circ f_2 \circ f_1)(x)$$

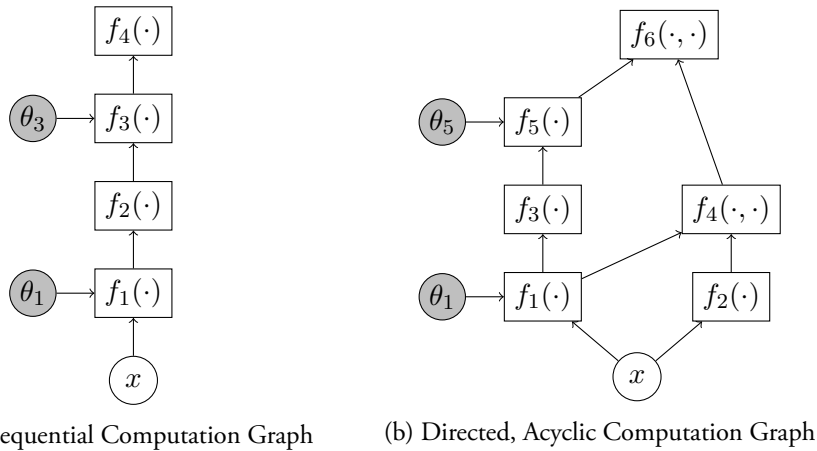


Figure 2.1: Different types of computation graphs

to compute its output. We call these models *feed-forward* models or sequential models. Each operation takes one input and computes one output. More complex structures are possible. First, the function f_i could be parameterized by a set of weights θ_i . Second, instead of the sequential structure, we can also design deep networks represented as directed, acyclic graphs, so-called *computation graphs*. Now the building blocks are arranged as nodes in graph that can have more than one output, which branches out the computation graph, or can take more than one input, which allows to merge different branches of the computation. Nodes with no incoming edges are inputs to the function, either the actual inputs x or the weights that parameterize the deep network. Nodes without outgoing edges compute the outputs of our model. There is at least one output, but more than one outputs are possible. This is illustrated in Figure 2.1.

The order in which the composed function of a deep network is evaluated is determined by the topological order of the graph. In deep learning, this is referred to as the *forward pass*: The forward pass of the whole network executes the forward passes of all blocks in topological order.

When we optimize an objective function that depends on the outputs of our model, we extend the computation graph by another node that computes the real-valued objective based on the one or more output nodes. Now instead of f , the graph expresses $(\mathcal{L} \circ f)$. As we have noted, optimization for deep learning requires the computation of gradients. How these gradients are computed is also described by the computation graph.

We need to compute the gradient of \mathcal{L} , the final objective-node in our extended computation graph, with respect to the inputs $x^{(i)}$ for all nodes in the graph. We consider a single node. Let $f(x)$ be its function with inputs $x^{(1)}, \dots, x^{(k)}$ and K outputs where $f^{(j)}(x)$ is the j -th output. The chain rule of differentiation states

$$\nabla_{x^{(i)}} \mathcal{L} = \sum_{j=1}^K \left(\frac{\partial f^{(j)}(x)}{\partial x^{(i)}} \right)^T \nabla_{f^{(j)}(x)} \mathcal{L} \quad (2.2)$$

where $\left(\frac{\partial f^{(j)}(x)}{\partial x^{(i)}}\right)$ is the Jacobian matrix of $f^{(j)}$ w.r.t. $x^{(i)}$. This equality expresses the dynamic programming approach for obtaining all gradients in the computation graph famously known as back-propagation [227], the computation (2.2) for a single block is called the *backward function*. Starting with the objective node, we can propagate the vector $\frac{\partial \mathcal{L}}{\partial x^{(i)}}$ through the graph in reversed topological order. For all other nodes, if we know the derivative with respect to their outputs, i.e. the inputs of their successors, we can compute the derivative with respect to their inputs. In particular we can compute the gradients for all weights of our model, which are represented as input nodes in the graph. This is called the *backward pass*.

For instance for the ERM objective, we can compute the gradient of the objective with respect to the weights from the sum of the gradients of the loss function ℓ that is differentiable in its first argument as

$$\nabla_{\theta} \mathcal{L}_{\text{ERM}}(f_{\theta}) = \frac{1}{n} \sum_{i=1}^n \left(\frac{\partial f_{\theta}(x_i)}{\partial \theta} \right)^T \nabla_{f_{\theta}(x_i)} \ell(f_{\theta}(x_i), y_i).$$

To summarize, computation graphs simultaneously specify how to compute two aspects of the model: How to compute the output of a model given its inputs and weights (forward pass) and how to compute the gradient of a loss function with respect to the inputs and weights (backward pass).

Basic Layers

We continue the introduction of models for deep learning with an overview over the most important and popular building blocks or layers of deep networks.

Linear Layers Given an input $x \in \mathbb{R}^d$, the linear layer computes its output

$$y = Wx + b$$

with weights $W \in \mathbb{R}^{d' \times d}$ and bias $b \in \mathbb{R}^{d'}$. Linear layers are often called *fully-connected* layers, as each component of the output y_k potentially depends on all components of the input, since $y_k = W_k^T x$. In traditional neural network literature, these components are called *neurons*. The backward pass of a linear layer is given as

$$\begin{aligned} \nabla_x \mathcal{L} &= W^T \nabla_y \mathcal{L} \\ \nabla_W \mathcal{L} &= (\nabla_y \mathcal{L}) x^T \\ \nabla_b \mathcal{L} &= \nabla_y \mathcal{L}. \end{aligned}$$

For high dimensional data, these linear layers have a very high number of trainable weights. For data with spatial or temporal neighborhoods *convolution* layers are an alternative with sparse and shared weights [145]. We denote the convolution with a filter W as

$$y = W * x$$

and note that it can be expressed as a matrix multiplication with a large, sparse weight matrix with duplicate entries. Convolution operations for 1d, 2d or 3d tensors, are efficiently implemented in all common deep learning toolkits.

Non-Linear Activation Functions Composing functions from only linear functions will always result in a linear function. To gain expressivity, we introduce non-linear activation functions. There are hundreds of alternatives, we will just introduce four of them. We begin with activations that are applied elementwise to a vectorial input. Traditional neural networks use the sigmoid activation

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

which outputs values in the range $(0, 1)$ which is convenient for modelling probabilities. Modern deep networks often use the ReLU activation [80]

$$[a]_+ = \max(0, a)$$

which sets all negative values to zero, thereby inducing sparse representations. While the sigmoid unit reduces the magnitudes of gradients which becomes a problem in deep networks, the ReLU activation either preserves the gradient or sets it to zero. This has been shown to reduce the *vanishing gradient* problem [90]. Many variants of the ReLU function have been proposed, mostly in order to keep gradient information for negative inputs. One popular choice is the Gaussian Error Linear Unit (GELU) defined by Hendrycks and Gimpel [105] as

$$g(a) = \frac{a}{2} \left[1 + \operatorname{erf}(a/\sqrt{2}) \right].$$

Finally we introduce the softmax activation that is not an element-wise activation, but transforms a vectorial input $a \in \mathbb{R}^d$ and returns a categorical probability distribution

$$s(a)_i = \frac{\exp(a_i)}{\sum_{j=1}^d \exp(a_j)} \text{ for } i = 1, \dots, d.$$

This activation is especially useful for transforming the output of the last layer of deep networks to compute class probabilities.

Batch-Normalization Batch normalization [115] is one of the success stories of deep learning. A seemingly simple operation that speeds up training, reduces the burden of hyperparameter tuning regarding learning rates and weight initializations and often even helps generalization.

The forward pass of batch normalization receives a minibatch of inputs x_1, \dots, x_m and computes its outputs y_i in the following steps:

$$\mu = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2$$

$$y_i = (x_i - \mu) \cdot (\sigma^2 + \epsilon)^{-\frac{1}{2}} \text{ for } i = 1, \dots, m$$

where μ is the sample mean, σ^2 is the sample variance and ϵ is a small constant for numerical stability that ensures that we never divide by zero.

Batch normalization relies on minibatches of data, sufficiently large as to allow the estimation of the mean and variance of the feature vector x . At inference time, we also need to make predictions for single data points, which does not allow the computation of variances and renders the definition of means void. To solve this problem, during training estimates of mean and variances are updated. Usually we use exponential smoothing to compute running means

$$\mu^{(t)} = (1 - \eta)\mu^{(t-1)} + \eta\mu \quad (2.3)$$

$$(\sigma^2)^{(t)} = (1 - \eta)(\sigma^2)^{(t-1)} + \eta\sigma^2 \quad (2.4)$$

for a momentum parameter $\eta \in [0, 1)$. At inference time, we use the running statistics $\mu^{(t)}$, $(\sigma^2)^{(t)}$ instead of computing batch statistics.

It is important to note that during training, we do a full backpropagation pass through this forward computation, also backpropagating the loss gradient through μ and σ^2 , hence gradients with respect to x are not just scaled and shifted (see e.g. [136] for a derivation of the backward pass).

Self-Attention Self-attention, a special form of the attention mechanism [273], is a network layer that learns how to aggregate inputs structured in a sequence.

We assume that our input is structured like a sequence, e.g. s.t. $x = [x_1, \dots, x_k] \in \mathbb{R}^{d \times k}$ for a fixed length k . We compute three vectorial representations of the input, the key k , the value v and the query q , via

$$k = Kx \in \mathbb{R}^{d' \times k} \quad (2.5)$$

$$v = Vx \in \mathbb{R}^{d' \times k} \quad (2.6)$$

$$q = Qx \in \mathbb{R}^{d' \times k} \quad (2.7)$$

with weights $K, V, X \in \mathbb{R}^{d' \times d}$. Now we compute the so-called attention matrix

$$A = \text{softmax}(k^T q) \in \mathbb{R}^{k \times k} \text{ s.t. } \sum_i A_{.i} = 1 \quad (2.8)$$

and use the attention weights that sum to 1 to compute the final output

$$y = vA \text{ or see below for componentwise} \quad (2.9)$$

$$y_i = \sum_{j=1}^k A_{ji} v_j \quad (2.10)$$

The attention matrix A is the source of non-linearity of the self-attention layer. It controls how the v_i values are aggregated at each step of the sequence. Multi-head self-attention computes a specified number of self-attention layers in parallel, concatenating the parallel outputs into a single output.

Graph Convolutions Graph CNNs have been applied in many contexts, for instance for classifying molecules [67] or classification and segmentation of point-clouds [281]. Like classic convolutional neural networks for image processing, graph convolutions compute feature maps based on local neighborhoods and thus can exploit relations between nodes in a graph. While in CNNs, we have features associated with each pixel in the pixel grid and neighborhoods are defined by this grid, in graph CNNs we have features associated with each node of the graph and neighborhoods are defined by the edges in the graph. We define graph structures $x = (V, E)$ as a tuple of node-features V and edges E . Let $|x|$ denote the number of nodes in x . We assume that $V \in \mathbb{R}^{|x| \times d}$ where v_i are the features of the i -th node. A graph CNN maps an input graph to an output with transformed feature vectors in a d' -dimensional output space but with identical edge structure. Let $\mathcal{N}(i)$ denote the set of neighboring nodes of the i -th node.

Borrowing the notation of Morris et al. [179], an abstract graph network layer can be described by its output

$$v'_i = \psi \left(v_i, \square_{j \in \mathcal{N}(i)} \phi(v_i, v_j, e_{ij}) \right)$$

where ϕ, ψ are differentiable buildingblocks such as linear transformations or neural networks, \square denotes a differentiable, permutation invariant function like sum, mean or max and $\mathcal{N}(i)$ denotes the set of all neighboring nodes of i in the tree with edges E . Note that ϕ might use information about the edges in the form of vectorial edge-features e_{ij} . An example of a simple graph convolutional layer is

$$v'_i = \sigma \left(W \sum_{j \in \mathcal{N}(i) \cup i} v_j \right)$$

which linearly transforms all nodes using a weight matrix W , aggregates by computing the sum of all neighborhoods and applying a component-wise activation function, in this case the sigmoid function σ .

We can use the average-pooling to aggregate all node-features of a graph into a single feature vector for a graph.

2.1.2 Objective Functions

We begin by discussing the objective function at heart of most supervised machine learning, the empirical risk minimization objective introduced earlier.

Empirical Risk Minimization

Empirical risk minimization is the predominant objective function for supervised learning tasks. It is centered around the concept of the *risk* of a model. Let $\ell : \hat{\mathcal{Y}} \times \mathcal{Y} \rightarrow \mathbb{R}$ be a loss function that takes an output of a model $\hat{y} \in \hat{\mathcal{Y}}$ and a desired output $y \in \mathcal{Y}$ and computes a score of how wrong the prediction is. A simple example is the 0/1 loss that takes a vector of class probabilities \hat{y} and the true class y and checks if we assign the largest probability to the correct class

$$\ell_{0/1}(\hat{y}, y) = \begin{cases} 0 & \text{if } \arg \max_i \hat{y}_i = y \\ 1 & \text{otherwise.} \end{cases}$$

The risk of a model f_θ is defined as the expected loss over the full data distribution

$$\mathcal{R}(f_\theta) = \mathbb{E}_{(x,y)} \ell(f_\theta(x), y) \quad (2.11)$$

When we do not have access to the data distribution, but only to a finite sample of datapoints that are independent and identically distributed (iid), we consider the empirical risk function as a mean to estimate the stochastic quantity using finite amounts of data

$$R(f_\theta) = \frac{1}{n} \sum_{i=1}^n \ell(f_\theta(x_i), y_i) \quad (2.12)$$

We hope that through minimizing the empirical risk

$$\arg \min_{\theta} R(f_\theta) = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \ell(f_\theta(x_i), y_i), \quad (2.13)$$

we obtain a model that has a small true risk. We employ validation techniques to verify this: The most common validation protocol in deep learning literature is the use of a train-test split. We split our dataset into two disjunct parts. Then we train on the training set and evaluate the risk on the test set. It is important that the two sets are independent. Another popular albeit more computationally expensive choice is cross-validation, where we divide the dataset into k disjunct chunks and use each chunk as test data once, training on the remaining chunks. This way we obtain k estimates of the risk [100].

When the predictions $\{f_\theta(x_i), i = 1, \dots, n\}$ are independent, we can bound the difference $|\mathcal{R} - R|$ using concentration inequalities like Hoeffding's bound or Bernstein's bound [24]. This is the case when we evaluate the risk on a sample of iid. holdout data, e.g. the test-data or in cross-validation. When we want to bound the difference on the training data and select f_θ by running a learning algorithm on the data, eg. by empirical risk minimization

$$f_\theta = \arg \min_{\theta} R(f_\theta),$$

f depends on all x_i and consequently, the $f(x_i)$ are no longer independent, even if the x_i are sampled independently. Statistical learning theory [176, 240, 272] attempts to bound this difference when functions f_θ are chosen from a family of functions.

When the model performs substantially better on our training data than on validation data, we speak of *overfitting*. Traditionally the term is used, when we find a generalization gap, i.e. the difference $|\mathcal{R} - R| > 0$. Lately, the meaning of the term overfitting has changed in deep learning: Deep networks with many parameters or overparameterized models, i.e. models with more parameters than input data, can easily fit any dataset with near-optimal training loss and we find that during training the validation loss still decreases, the so-called double-descent [296, 297]. We speak of overfitting when during training, the loss on a holdout set stops decreasing and starts to increase again.

Popular Loss Functions

For classification tasks, the most popular loss function is the *cross entropy loss*, which is the negative log-likelihood of the predicted probability of the true class.

$$\ell_{\text{CE}} = -\log \hat{y}_y$$

where \hat{y} is a probability vector $\hat{y} \in [0, 1]^d$ with $\|\hat{y}\|_1 = 1$ and $y \in \{1, \dots, d\}$. In deep learning we usually use ℓ_{CE} on the outputs of a softmax activation function.

Another important loss is the *hinge loss* known from support vector machines

$$\ell_{\text{hinge}}(\hat{y}, y) = \max\left(0, 1 - \left[\hat{y}_y - \max_{i \neq y} \hat{y}_i\right]\right)$$

that is zero when the model's output for the true class exceeds the outputs for all the other classes by a margin of 1 or more. This notion of a margin is useful only when we control the magnitudes of the model's outputs. In support vector machines this is done through l_2 -regularization. For deep networks we can either use activations that limit the range of the outputs, e.g. sigmoid or softmax, or apply explicit normalization [62] to the output by scaling with the inverse of its norm. For the case of binarized neural networks, the maximum output is known, and we can change the margin constant from 1 to a more appropriate value [33].

2.1.3 Optimizers

We will cover the basics of optimization methods used for deep learning in this section.

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is an optimization algorithm for minimization of expectations over distributions that we can sample from. The most relevant problem that fits this description in machine learning is the risk of a classifier

$$\min_{\theta} \mathcal{R}(f_\theta) = \min_{\theta} \mathbb{E}_{(x,y)} \ell(f_\theta(x), y).$$

Starting with an initial $\theta^{(0)}$, we iteratively refine θ by sampling a data point $(x^{(i)}, y^{(i)})$ and evaluating its stochastic gradient

$$\nabla_{\theta} \ell(f_{\theta}^{(i-1)}(x^{(i)}), y^{(i)}).$$

Then we take a small step in the negative direction of the gradient, the direction of steepest descent for $\ell(f_{\theta}^{(i-1)}(x^{(i)}), y^{(i)})$ in order to optimize the risk. We arrive at

$$\theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} \ell(f_{\theta}^{(i-1)}(x^{(i)}), y^{(i)})$$

where $\eta > 0$ is a small *step size*, also referred to as *learning rate* in the context of machine learning. In practice, however, rather than sampling the examples during the optimization we instead work on a finite dataset and iterate over the data in random order.

The use of stochastic gradients in contrast of using the full gradient computed on all available data greatly reduces the runtime cost for large datasets. Because gradients are linear, the stochastic gradients are unbiased estimators of the full gradient, i.e.

$$\mathbb{E}_{(x,y)} \nabla_{\theta} \ell(f_{\theta}(x), y) = \nabla_{\theta} \mathcal{R}(f_{\theta})$$

however the variance of this estimator will often be high. Hence instead of estimating the gradient based on a single data point, in practice we use small *minibatches* of data to estimate the gradient. Let $B^{(i)} = \{(x_k^{(i)}, y_k^{(i)}) \mid k = 1, \dots, m\}$ be the minibatch sampled at the i -th iteration. The size of these minibatches m , the *batch-size*, is an important hyper-parameter. We update θ based on the stochastic gradient estimated on $B^{(i)}$

$$\theta^{(i)} = \theta^{(i-1)} - \frac{\eta}{m} \sum_{k=1}^m \nabla_{\theta} \ell(f_{\theta}^{(i-1)}(x_k^{(i)}), y_k^{(i)})$$

In practice, we usually do not have the possibility of sampling the distribution, but only have access to a fixed dataset for training. We can still apply SGD by iterating over the data set a number of times. We call this number of iterations the number of *epoches* where an *epoch* is one iteration over the full dataset.

Momentum and Adam

Stochastic gradient descent is a so-called *first-order* method, i.e. it only uses the first derivative for optimization. Access to higher order derivatives gives an optimization algorithm additional information about the curvature of the objective function. For instance, Newton method uses the second derivative, the Hessian matrix. Unfortunately, computing the quadratic Hessian matrix is intractable for large deep networks. Thus we focus on first-order methods in this section, but consider additional tricks that help convergence with noisy stochastic gradients.

The first trick, *momentum*, does not use the raw stochastic gradients, but keeps a running average of the gradients and uses this more stable estimate for updating the

weights. The influence of the old estimate is controlled using a parameter $\beta \in (0, 1]$ referred to as *momentum*. For notational simplicity we show the computation without minibatches:

$$\begin{aligned}v^{(i)} &= \beta v^{(i-1)} + \eta \nabla_{\theta} \ell(f_{\theta}^{(i-1)}(x^{(i)}), y^{(i)}) \\ \theta^{(i)} &= \theta^{(i-1)} - v^{(i)}\end{aligned}$$

Common choices for β are 0.9 or 0.99.

Probably the most widely-used SGD-variant for deep learning is Adaptive Moment Estimation (Adam) [134]. It aims to choose learning rates for each parameter individually, such that weights that frequently receive substantial updates have smaller learning rates than weights with infrequent updates. To this end, Adam tracks the magnitudes of the gradients and the element-wise-squared gradients in running averages. With $v_1^{(0)} = v_2^{(0)} = 0$, we update

$$\begin{aligned}v_1^{(i)} &= \beta_1 v_1^{(i-1)} + (1 - \beta_1) \nabla_{\theta} \ell(f_{\theta}^{(i-1)}(x^{(i)}), y^{(i)}) \\ v_2^{(i)} &= \beta_2 v_2^{(i-1)} + (1 - \beta_2) \left(\nabla_{\theta} \ell(f_{\theta}^{(i-1)}(x^{(i)}), y^{(i)}) \right)^2 \\ \theta^{(i)} &= \theta^{(i-1)} - \eta \cdot \left(\sqrt{v_2^{(i)} / (1 - \beta_2^i)} + \varepsilon \right)^{-1} \cdot (1 - \beta_1^i)^{-1} v_1^{(i)}.\end{aligned}$$

More Design Choices

Typical deep learning pipelines include a large number of techniques and tricks, most coming with their own set of hyperparameters to tune. We discuss some common design choices.

Initialization There are many ways of randomly initializing the weights in deep networks. The most common initialization scheme is Glorot or Xavier initialization [89] where weights are drawn from a normal distribution with zero mean and variance inversely proportional to the size of the layer. More formally, when a linear layer maps from d to d' dimensions, we chose

$$\sigma = \sqrt{\frac{2}{d + d'}}$$

and initialize the weights by sampling a normal distribution $w_{ij} \sim \mathcal{N}(0, \sigma)$. Alternatively, we can sample from a uniform distribution $w_{ij} \sim \mathcal{U}(-\sqrt{3}\sigma, \sqrt{3}\sigma)$.

Regularization We often want to encode additional desired properties of our model into the objective function or into the training process. These properties often relate to the ability of our model to generalize to unseen data: By penalizing the proxy measures for the complexity of the model, we wish to obtain 'simpler' models that generalize better.

A classic example of regularization is l_2 -regularization, where we additionally penalize the l_2 of the (vectorized) weights of a network and include an additional $+\lambda\|\theta\|_2^2$ in the objective that has a hyperparameter λ that controls the strength of regularization. In the context of deep learning this is often called *weight decay* and is implemented in the optimizer rather than in the objective function: After each gradient step, the magnitude of the weights is reduced by a factor of λ , which, mathematically is the same as adding the term to the objective. We can use other norms for regularization, l_1 is a popular choice for inducing sparsity [260] and the nuclear matrix norm is a popular choice for inducing low-rank transformations in linear layers [164].

Dropout regularization [247] randomly sets the values of neurons to 0 during training, while during testing the network uses all of its neurons, appropriately scaled. Intuitively, this forces the network to learn redundant representations that still work under this stochastic noise, thereby reducing the capacity of the network. Indeed, dropout is related to nuclear norm, low-rank regularization [41].

Early Stopping, i.e. stopping the training before convergence, e.g. after a predefined number of epochs or as soon as the models performance measured on a separate development set stops improving, also biases the training process towards simpler models.

Learning Rate Schedules Tuning the learning rate or step-size for training deep learning remains a challenge. For good convergence to optimal objective values, we often adapt the learning rate during training, so-called *learning rate scheduling*. Instead of using a constant learning rate, decreasing it during training has shown benefits. A simple schedule is step-wise decrease, where at pre-specified epochs we decrease the learning rate by a specified factor. A common choice is decreasing the learning rate by $1/2$ twice during training, e.g. after 30 and 60 epochs for 100 epochs of training.

Data Augmentation When we want to reduce the generalization gap between training and test performance, conventional wisdom dictates that we find more training data. However, obtaining more training data is expensive. Instead, we can apply cheap randomized heuristics to obtain artificial training data by making small modifications to examples in our existing data. We call this process *data augmentation*. A simple example is adding Gaussian noise with small variance to the original data points. Depending on the data modality, we can apply domain-specific augmentations that incorporate background knowledge about invariances. For instance for images, we can apply a number of image transformations like cropping, scaling, rotating, changing the colors, brightness or contrast, etc. A popular approach is RandAugment [305], where we randomly choose a number of these transformations and randomly choose the intensity of the transformations. We may also obtain new training example by interpolating between two or more existing examples. Examples of this approach include SMOTE [20] and Mixup [298].

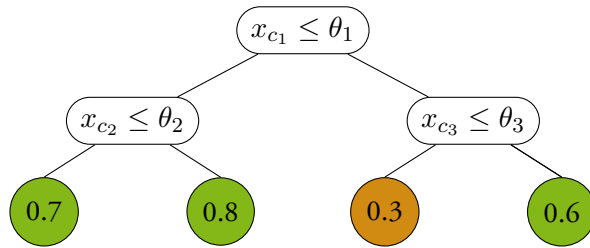


Figure 2.2: Example decision tree of depth 2 for binary classification

2.2 Tree-Based Machine Learning Models

After discussing deep learning and some of its many design choices, we now cover tree-based machine learning models. We begin by discussing decision trees, then we cover random forest models, i.e. ensembles of decision trees.

2.2.1 Decision Trees

Decision trees are a family of models that follow a different structure than deep networks. While deep networks are, for the most part, a composition of linear and non-linear transformations, decision trees work by recursively partitioning the input domain by axis-aligned cuts. We will first formally define the family of decision tree models, then we will introduce the standard greedy decision tree induction algorithm for learning decision trees from data.

Model Structure A decision tree $f(x)$ for classification tasks can be represented by a binary tree that describes how a given example x is classified. We start the classification at the root node. Each internal node v_i , identified by index i , computes a comparison $x_{c_i} \leq \theta_i$ on a feature c_i using a split threshold θ_i . Depending on the outcome of the comparison, the classification recursively proceeds in the left or in the right child node. Each leaf node stores an output probability vector in the multiclass setting or a single probability in the binary classification setting. These probabilities f^{v_i} are output when we reach a leaf node during inference. In conclusion, each node v of the tree is a tuple of feature c , threshold θ , left and right child denoted by $\text{l_child}(v)$ and $\text{r_child}(v)$, and output probability f^v , where the output probability is only used if the node has no children. The set of leaf nodes of the decision tree f is denoted by $V(f)$. We see a small example in Figure 2.2.

Greedy Decision Tree Induction The de-facto standard method for training decision trees given a set of labeled training data (x_i, y_i) is top-down greedy decision tree induction [30, 214]. We recursively split the full dataset X using the split (c, θ) that locally optimizes a so-called *split criterion* until a termination criterion is fulfilled. More

specifically, we chose

$$c, \theta = \arg \min_{c, \theta} F(\underbrace{\{(x, y) \in X \mid x_c \leq \theta\}}_{=: \text{left}(X, f, \theta)}, \underbrace{\{(x, y) \in X \mid x_c > \theta\}}_{=: \text{right}(X, f, \theta)})$$

and recursively continue on $\text{left}(X, c, \theta)$ and $\text{right}(X, c, \theta)$.

Common choices for the split criterium F are the Gini score that computes the weighted sum of Gini impurities on the left and right side of the split value

$$F_{\text{gini}}(X) = |\text{left}(X, c, \theta)| \cdot G(\text{left}(X, c, \theta)) \quad (2.14)$$

$$+ |\text{right}(X, c, \theta)| \cdot G(\text{right}(X, c, \theta)) \text{ where} \quad (2.15)$$

$$G(X) = 1 - |X|^{-2} \sum_{j=1}^C \left(\sum_{i=1}^{|X|} \mathbb{1}[y_i = j] \right)^2 \quad (2.16)$$

and the entropy measure, weighted analogously

$$F_{\text{ent}}(X) = |\text{left}(X, c, \theta)| \cdot H(\text{left}(X, c, \theta)) \quad (2.17)$$

$$+ |\text{right}(X, c, \theta)| \cdot H(\text{right}(X, c, \theta)) \text{ where} \quad (2.18)$$

$$H(X) = - \sum_{j=1}^C \left(|X|^{-1} \sum_{i=1}^{|X|} \mathbb{1}[y_i = j] \right) \log \left(|X|^{-1} \sum_{i=1}^{|X|} \mathbb{1}[y_i = j] \right) \quad (2.19)$$

When we contrast the entropy measure with the measure with no new split, we can interpret the difference as *information gain* and use this as a termination criterion.

Different termination criteria are possible. Obviously, stopping when $|X| = 1$ is the ultimate stopping criterion. Other choices are building trees up to a user-specified maximum tree depth. In general, smaller trees generalize better whereas deeper trees have higher capacity to fit complex decision boundaries, yielding the traditional trade-off seen in statistical learning. We can also stop once the split criterion computes an impurity score below a user-specified threshold or once the number of training data points in a node is below a threshold. Each of these termination criteria is represented by tunable hyperparameters in common machine learning frameworks.

2.2.2 Random Forest Models

To improve the classification performance of a single decision tree, we can use a collection of decision trees and use their average prediction to classify examples. This technique is called *ensembling*. Obviously, we need to obtain a set of different decision trees. To this end, Breiman has proposed bagging [28], where we draw bootstrap samples, i.e. samples of the same size as the original dataset drawn with replacement, and train classifiers on these bootstrap samples. The random forest classifier, proposed by Breiman [29], goes one step further and also uses a random subset of features, drawn without replacement, usually of size \sqrt{d} where d is the original dimensionality of the data. Using these bootstrap samples of reduced dimensionality, we train decision trees

using the greedy top-down induction algorithm discussed above. The resulting random forest classifiers have been shown to be among the most successful classifiers on a wide range of datasets [75].

Recent results illustrate that random forests work well in the interpolating regime, i.e. in situations where the individual decision trees are trained to classify their respective training sets perfectly [34]. Traditional theoretic frameworks for investigating generalization of random forests, like the bias-variance-decomposition [36] or the C-bound [83] for majority vote random forest, still fall short of fully explaining the success of random forests.

Part II

Designing Machine Learning Tasks

Chapter 3

Related Work on Machine Learning Tasks

How do we turn data into actionable knowledge or models that guide decisions? Machine learning provides the answer in the form of machine learning tasks that specify desiderata for models, and in the form of machine learning algorithms that train models on the specified tasks. In the last section we have seen the objectives of empirical risk minimization for supervised learning, e.g. for classification we can use a model that minimizes the cross-entropy-loss of its output with respect to the ground-truth labels. In this chapter we review a wider range of objective functions, covering variants of supervised learning and unsupervised representation learning approaches.

3.1 Supervised Learning

We present related work on machine learning tasks that work with variants of full supervision that go beyond the traditional empirical risk minimization objective covered in the last chapter.

3.1.1 Supervised Learning under Class Imbalance

In classification problems with strong class imbalance, i.e. situations where some classes are much more frequent than other classes, plain empirical risk minimization may yield undesirable models. Solutions include weighting the training examples [127] or, over- or undersampling the dataset to obtain a balanced dataset [20].

Alternatively, for binary classification problems, formulating the training objective as a pairwise ranking task has been proposed. Intuitively, for any pair of data points $(x, y), (x', y')$, whenever $y \neq y'$ we want the model to output a higher probability for class +1 for the example with positive label [51]. This yields the risk functional

$$\mathbb{E}_{(x,y),(x',y')} \mathbb{1}[y = +1 \wedge y' = -1] \cdot \mathbb{1}[f(x) > f(x')] \quad (3.1)$$

In order to apply numerical optimization methods, we substitute the indicator function with a convex loss function like the hinge loss and obtain the following training objective for a finite data set

$$\mathcal{L}(f) = \frac{1}{K} \sum_{(x,+1) \in D} \sum_{(x',-1) \in D} \max(0, 1 - (f(x) - f(x'))) \text{ with} \quad (3.2)$$

$$K = |\{y \mid (x, y) \in D \wedge y = +1\}| \cdot |\{y \mid (x, y) \in D \wedge y = -1\}| \quad (3.3)$$

It has been shown that this objective function directly optimized the area under the ROC-curve, an important quality measure in retrieval as well as imbalanced classification [162, 274]. For linear models $f(x) = \langle w, x \rangle$ it holds that $f(x) - f(x') = \langle w, x - x' \rangle$, so we can generate new training data comprised of the differences of pairs of examples. Training using all combinations of data points quickly becomes infeasible, so randomly generating training batches is a more efficient algorithmic option [48].

We include pairwise ranking in our discussion as it illustrates that objective functions may compute losses over more than one output of the model; pairwise uses the models twice on a pair of examples from the training data. We will see more examples of this pattern in the following sections, the formulation is particularly similar to similarity learning approaches.

3.1.2 Noisy Label Learning

For binary classification problems, we traditionally assume access to ground-truth labels. In contrast, in binary classifications problems with noisy labels, we only have access to labels that are noisy, i.e. they may be flipped according to a random process [84, 183, 235]. We consider so-called *class-conditional label noise*: Here labels are flipped according to probabilities that depend only on the true class [183]. Let $\hat{y} = \pm 1$ denote a noisy label and $y = \pm 1$ be its corresponding clean label. Then the noise process is defined by two probabilities p_+, p_- . We define $p_+ := P(\hat{y} = -1 \mid y = +1)$ as the probability of flipping a clean positive, and $p_- := P(\hat{y} = +1 \mid y = -1)$ for the probability of flipping a clean negative label. The process is summarized in Figure 3.1.

Learning under this noise model is feasible when noisy labels are correct on average [167, 183], i.e.

$$p_+ + p_- < 1. \quad (3.4)$$

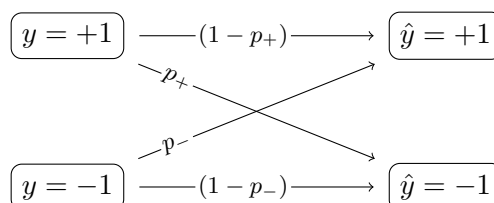


Figure 3.1: The Flip-Probabilities for Class-Conditional Label Noise

Equivalently, we can say that learning is feasible if it is more likely to observe a noisy positive given a true positive than given a true negative

$$P(\hat{y} = +1 | y = +1) > P(\hat{y} = +1 | y = -1).$$

According to Scott et al. [235, Proposition 1], there is a one-to-one correspondence between the optimal noisy thresholds $\theta \in \mathbb{R}$ and the optimal clean thresholds $\lambda \in \mathbb{R}$. Namely, $\forall x \in \mathcal{X} \forall \lambda \geq 0 \exists \theta \geq 0$ such that

$$\frac{\mathbb{P}(x | Y = +1)}{\mathbb{P}(x | Y = -1)} > \lambda \Leftrightarrow \frac{\mathbb{P}(x | \hat{Y} = +1)}{\mathbb{P}(x | \hat{Y} = -1)} > \theta.$$

Indeed, for a Bayes-optimal classifier for the noisy labels, $f(x) = P(\hat{y} = +1 | x)$ we can use the threshold $\theta^* = (1 - p_+ + p_-)/2$ and output +1 only if the $f(x) \geq \theta^*$.

Another direct consequence of this result is that the area under the ROC curve is immune to label noise [167]. For other measures such as accuracy, we can obtain good classifiers by training on noise data and then finetuning the decision threshold [167]. If we have access to a small sample with clean labels, this sample can be used to estimate the threshold [21]. However, in many situations we do not have access. If both p_+ and p_- are known, we can set the threshold as described above. Alternatively, we can apply label-dependent weights in the loss function during training [84, 183].

When both noise rates are unknown, we can try to estimate them from noisy data [167, 235]. This requires additional assumptions. For instance, Menon et al. [167] assume that there are areas of the feature space where the labels are clean and the other clean class has zero probability. Under class imbalance, this becomes more challenging [173].

Other two-step procedures that estimate the transition matrix between the noisy- and true-classes have been proposed. A common assumption is the existence of so-called anchor points where we can assume that the examples are correctly labeled. For instance Patrini et al. [202] use the most confidently predicted examples for each class as anchor points and estimate the transition matrix based on these few points. Then the transition matrix is applied at inference time to compute the estimated true-class probabilities from the predictions for the noisy class. Building on this so-called forward-correction procedure, Xia et al. [286] and Yao et al. [288] propose other approaches for estimating the transition matrix from noisily-labeled data. In the context of deep learning, methods for training the prediction model and estimating the transition matrix simultaneously during training have been proposed, e.g. by Li et al. [153].

3.1.3 Multi-Task Learning

In multi-task learning [39, 40] we want to train a model that can simultaneously solve more than one prediction tasks given the same input data. Since the outputs required for the different tasks may vary, usually models are constructed to have multiple outputs where the outputs share significant parts of the computation.

By exploiting similarities between related tasks we can use information in the training signals of other tasks and obtain better generalization [40]. Intuitively, multi-task

learning increases the amount of training data, as we also benefit from the training data of the related tasks. Additionally, the space of possible models is constrained to the intersection of task-specific models, reducing the statistical complexity of learning [40, Sec. 3.2.4]. Another benefit is the resource efficiency: When we can share parts of the model for a number of tasks that need to be solved together frequently, we save computation time, memory and, more generally, energy.

In the case of deep networks, the models may learn a shared hidden representation and apply separate *prediction heads* on top of that shared representation. Formally, we have $f(x) = [f_1(x), \dots, f_k(x)]^T$ with $f_i(x) = h_i(g(x))$ with a shared feature encoder $g(x)$ and prediction heads $h_i(z)$. Each of these prediction tasks is associated with its own objective and loss function.

During *simultaneous training*, all task objectives are combined and jointly trained. A naive way to achieve this is to add all loss values and optimize their sum. This works well if for all examples we have labels for all tasks and if the loss values fall into a similar numeric range. Indeed, Kurin et al. demonstrate that in fair evaluations, this approach outperforms more complicated methods in a number of benchmark tasks [141]. When loss ranges differ significantly, we may need to reweight the objectives in a weighted sum, as to not put too much emphasis on individual objectives. How these weights are chosen has a strong effect on the resulting behavior of the model [132]. The task weights are additional hyperparameters that need resource-intensive tuning. Kendall et al. [132] propose to automatically select weights based on the homoscedastic uncertainty of the tasks. By explicitly modeling the variance of continuous outputs and the temperature of discrete outputs using tunable parameters that are part of the overall objective, we obtain a fully differentiable objective function that includes task weights, where naively tuning weights would converge to a zero weight vector solutions [132]. When our training data is labeled incompletely, i.e. each example only has labels on a subset of tasks, we can optimize the objectives in an alternating fashion, where we optimize a different task at each weight update in an SGD-like optimization algorithm [49].

Ruder distinguished between hard parameter sharing and soft parameter sharing methods [224], where hard parameter sharing forces the multitask models to use exactly the same model parameters up to the individual prediction heads, whereas in soft parameter sharing the task-specific models are only regularized to be similar, e.g. by penalizing the pairwise squared Euclidean distances of the corresponding layer weight matrices.

With the availability of large collections of pre-trained model, a second multi-task paradigm has been introduced: Sequential Training. Here we take a model trained on a set of tasks and train it on a novel task. Hence the user training the new model may not have access to the training data used for the earlier model. This introduces the problem of catastrophic forgetting [79], where after training the problem on the novel task, the performance of the previous tasks degrades.

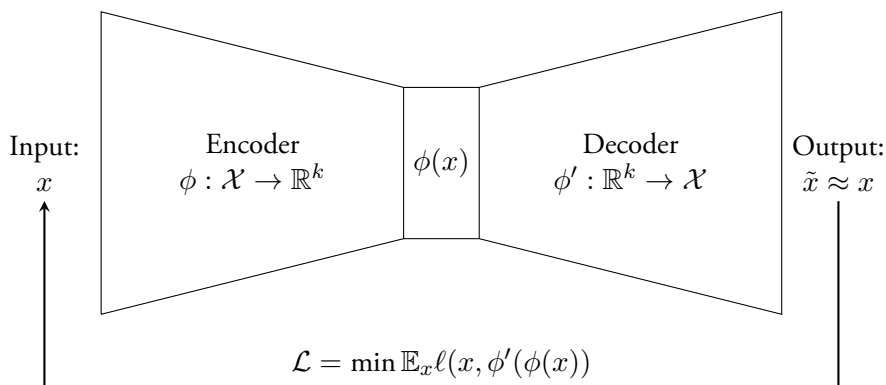


Figure 3.2: The Auto-Encoder architecture with its components. The training objective is minimizing the expected reconstruction error $\min \mathbb{E}_x \ell(x, \phi'(\phi(x)))$ and we train using an unlabeled dataset.

3.2 Representation Learning

In this Section we will cover approaches that work without full supervision. Instead we learn representations based on other sources of supervision. Particularly for high-dimensional data, learning a more compact representation that captures the latent structures of the high-dimensional space is an important topic [15]. Learning a good representation that captures useful information for our data makes it easier to solve downstream tasks like similarity search [209], transfer learning [73], or few-shot classification [251]. We will cover different ways to obtain supervision signals in this section.

3.2.1 Auto-Encoders

The first source of supervision signals we consider in this chapter is reconstruction. We want to learn a representation that can be approximately inverted, such that we can reconstruct the input given the representation.

Principal component analysis (PCA) [100] is a linear representation learning algorithm that finds the orthogonal, linear map into a lower-dimensional subspace that has minimal reconstruction error measured in squared Euclidean norm. Let d be the original dimensionality of the data. We can choose the dimensionality of our representation, $k \leq d$. The weight matrix of our PCA representation are the k leading eigenvectors of the covariance matrix of our training data.

Auto-Encoders [26, 109] use the same training objective idea, i.e. minimizing the reconstruction error between input and reconstruction as measured by a loss function, but use deep networks rather than linear functions to compute the hidden representation (encoder) and the reconstruction (decoder) [15]. This is depicted in Fig 3.2. For real-valued inputs like images or timeseries, we often use the Euclidean distance as loss function. For textual data that is represented as bag-of-words, we can use the negative log-likelihood of predicting the original tokens given the token probabilities

computed by the decoder [52, 207].

3.2.2 Contextual Embeddings

One popular class of unsupervised pretraining algorithm is built around the distributional hypothesis. Formulated eg. by Harris in 1954 [99], it postulates that objects that occur in similar contexts are similar. In the context of natural language, this means that words are similar when the words that frequently surround it (the context) are similar. Put differently: The semantic meaning or grammatical function of a word can be derived by its context. We can exploit these 'distributional regularities' for designing representation learning algorithms. This is the second source of supervision signals covered in this chapter: Contexts.

The idea of contextual embeddings for natural language processing was made popular by Mikolov et al. [168–170], who proposed several approaches for learning vectorial representations of words. We introduce their so-called *Skip-Gram* approach: Starting from a large text corpus \mathcal{T} , let $w_1 w_2 \dots w_k$ be its tokenized sequence of words. Let \mathcal{V} denote the vocabulary, i.e. the set of words that we consider from the corpus. The vocabulary might contain a special $\langle \text{unk} \rangle$ token for handling out-of-vocabulary tokens. For simplicity, we assume single large text rather than a collection of texts. We define contexts of words using a positive integer k and say that w_j occurs in the contexts of w_i if $|i - j| \leq k$ with $i \neq j$. We denote the set of all words in the context of w_i as $\mathcal{C}_k(w_i)$. When handling a collection of texts, we have to define the contexts accordingly, such that contexts do not cross over the boundaries of the individual texts.

We want to learn a model $f_\theta : \mathcal{V} \rightarrow \mathbb{R}^d$ that maps a given word into a vector space such that we can predict which other words frequently appear in its context. We can extend the model to (f_θ, W) with $W \in \mathbb{R}^{|\mathcal{V}| \times d}$ such that $s(W f_\theta(w))$ computes word-probabilities using the softmax function $s(\cdot)$. Now we define the objective as minimizing the negative log likelihood of the predicted word probabilities in the contexts of tokens in the data

$$\mathcal{L}(\theta, W) = \mathbb{E}_{w \sim \mathcal{T}, w' \in \mathcal{C}_k(w)} - \log s_{w'}(W f_\theta(w)) \quad (3.5)$$

where words from the context are selected with uniform probability. For large vocabularies, computing the softmax operation is computationally expensive. One alternative proposed by Mikolov et al. is using the hierarchical softmax operation [168, 178] that decomposes the computation of the output probabilities along a binary tree which enables the computation of output probabilities in $\mathcal{O}(\log |\mathcal{V}|)$ for balanced trees. Instead of using balanced trees, in practice we often use Huffman trees. This results in smaller tree-depth for frequent words at the cost of larger depth for the infrequent tokens.

Another alternative to eliminate the softmax operation is *negative sampling*. Instead of estimating word probabilities, we turn the problem into a binary classification problem: Given a pair of words, decide whether they appear in the same context or not. The original formulation uses separate models θ, θ' for embedding the words and

the contexts, though there are works that just rely on the same model (e.g. [121]).

$$\mathcal{L}(\theta, \theta') = \mathbb{E}_{\substack{w \sim \mathcal{T} \\ w_+ \in \mathcal{C}_k(w)}} \left[-\log \sigma(\langle f_\theta(w), f_{\theta'}(w_+) \rangle) \right] \quad (3.6)$$

$$+ K \mathbb{E}_{w_- \notin \mathcal{C}_k(w)} \left[-\log(1 - \sigma(\langle f_\theta(w), f_{\theta'}(w_-) \rangle)) \right] \quad (3.7)$$

Algorithmically, we randomly sample K negative examples and apply stochastic gradient descent on each of these samples. Sampling the negative samples can be done uniformly at random, but Mikolov et al. [169] find that choosing them with probability proportional to frequency of a token raised to the power $2/3$ yields better results.

Levy and Goldberg show that the objective function can be expressed as a matrix factorization approach of the shifted pointwise mutual information matrix [149], which connects the approach to other matrix factorization approaches like SVD.

We note that Mikolov et al. also proposed another variant for learning word embeddings, the *CBOW* model. In this formulation, we use the full context as input and predict the original token. This idea has later been called *masking* and is at the core of training modern, large-scale language models. We will discuss it further later in this Chapter.

The use of contextual embeddings is not limited to word embeddings. For instance, Sermanet et al. [238] use it to embed individual image frames from a video in a shared vector space. Here the context is defined by the time difference between two frames.

3.2.3 Representation Learning based on Similarity

Next we consider a class of training objectives originally proposed for similarity learning where ground-truth similarities are known. However, we can also apply weaker notions of similarity such as the contextual similarity introduced in the previous section or the augmentation approaches introduced in the next section. This was proposed already by Becker and Hinton [12], who train representations by maximizing the similarity of representations of separate but related parts, e.g. adjacent patches of the same input image or different modalities of the same input (e.g. vision and audio of a video). Note that our introductory example of contextual embeddings also follows the principle of maximizing similarities. Let us first consider the loss functions used.

Margin Approaches for Pairs and Triplets

Mobahi et al. [174] propose the *Siamese loss*: Given a desired margin $\Delta \geq 0$, the dot product of the representations should be larger than Δ for a similar pair and less than $-\Delta$ otherwise. We use $y = \pm 1$ to encode the true similarity of unit-length, vectorial representations x_1 and x_2 and define

$$\ell_{\text{siam}}(x_1, x_2, y) = \max(0, \Delta - y\langle x_1, x_2 \rangle). \quad (3.8)$$

The model f is used twice to calculate the loss, motivating the term *Siamese network*, because we can also view this as two models that share all their weights (twins) and that are connected by the loss function.

The second loss works on triplets, so that the model is used three times per training example: Given an example x , the so-called anchor, a similar example x_+ and a dissimilar example x_- , we want the model to judge the similar pair as more similar than the dissimilar pair. Balntas et al. propose to formulate this as a margin loss as well [10]. Furthermore they propose to swap the roles of anchor example and the positive example if this yields a larger loss value to obtain a more strict loss:

$$\ell_{\text{tri}}(x, x_+, x_-) = \max \begin{cases} 0 \\ \Delta - \langle f(x), f(x_+) \rangle + \langle f(x), f(x_-) \rangle \\ \Delta - \langle f(x_+), f(x) \rangle + \langle f(x_+), f(x_-) \rangle \end{cases} \quad (3.9)$$

Following Janocha and Czarnecki [122] we can use squared hinge losses ℓ_{siam}^2 and ℓ_{tri}^2 instead of hinge losses for faster convergence.

These losses can be used to derive a risk function. For the Siamese network formulation, we assume that a distribution of labeled similarity pairs is available

$$\mathcal{L}(\theta) = \mathbb{E}_{(x_1, x_2, y)} \ell_{\text{siam}}(f_\theta(x_1), f_\theta(x_2), y) \quad (3.10)$$

and if p_+ and p_- denote the probabilities of sampling a positive pair or a negative pair, we can decompose

$$\begin{aligned} \mathcal{L}(\theta) &= p_+ \mathbb{E}_{(x_1, x_2) | y=+1} \ell_{\text{siam}}(f_\theta(x_1), f_\theta(x_2), +1) \\ &\quad + p_- \mathbb{E}_{(x_1, x_2) | y=-1} \ell_{\text{siam}}(f_\theta(x_1), f_\theta(x_2), -1) \end{aligned}$$

In situations where we only have access to a distribution of positive pairs, we can again apply negative sampling: A positive pair should be more similar than a random pair. Formally, this amounts to substituting the distribution of negative pairs in the second expectation with an artificial negative-distribution.

For the triplet loss, we can also assume a distribution over triplets (x, x_+, x_-) . Alternatively, we need access to a distribution of anchor examples $x \sim \mathcal{P}$, as well as positive and negative distributions $\mathcal{P}_+(x)$ and $\mathcal{P}_-(x)$ that are conditioned on the anchor example x . Then we define the risk as

$$\mathcal{L}(\theta) = \mathbb{E}_{x \sim \mathcal{P}} \mathbb{E}_{x_+ \sim \mathcal{P}_+(x)} \mathbb{E}_{x_- \sim \mathcal{P}_-(x)} \ell_{\text{tri}}(f_\theta(x), f_\theta(x_+), f_\theta(x_-)).$$

We demand that all inputs to the loss functions are unit-length. Without controlling the norm of the inputs, we can trivially minimize the loss by scaling all vectors as soon as the similarity of the positive pair is larger than for the negative pair. Controlling the magnitude can alternatively be achieved through regularization of the model.

Minibatch Approaches

Instead of using only two or three examples to compute a loss of a predicted similarity between two representations, we can also design losses that work on a larger batch of data points.

Ustinova and Lempitsky [269] have proposed the so-called *histogram loss*. We have a minibatch of m positive pairs X^+ and a batch of negative pairs X^- with respect to m anchor examples X . For now, we leave the question of where the labels for these pairs come from untouched. We collect all cosine similarities between positive pairs in a vector $s^+ = (\langle f(x_i), f(x_i^+) \rangle)_{i=1, \dots, m}$ and of all negative pairs in s^- . We divide the interval $[-1, 1]$ into $R - 1$ equally-sized bins with boundaries $-1 = t_1, t_2, \dots, t_R = 1$ and width $\Delta = 2/(R - 1)$ and build histograms for the positive similarities and the negative similarities. Now we demand that the positive histogram leans more toward the $+1$ similarity than the negative histogram. We formalize this intuition as

$$\ell_h(s^+, s^-) = \frac{1}{m^2} \sum_{r=1}^R \sum_{r'=1}^r \left(\sum_{i=1}^m \delta_r[s_i^-] \right) \left(\sum_{i=1}^m \delta_{r'}[s_i^+] \right) \quad (3.11)$$

where instead of hard assignments, we use the triangular kernel

$$\delta_r[s] = \begin{cases} (s - t_{r-1})/\Delta & \text{if } s \in [t_{r-1}, t_r] \\ (t_{r-1} - s)/\Delta & \text{if } s \in [t_r, t_{r+1}] \\ 0 & \text{otherwise} \end{cases}$$

to put similarities into bins. This way we obtain a differentiable loss function. When negative pairs are chosen randomly, instead of using a vector of dissimilar dot product values, we can also use all combinations of dot products, i.e. all entries in the matrix $S_{ij}^- = \langle f(x_i), f(x_j^-) \rangle$, to compute the histogram of the negative similarities.

Arguably the most popular loss function is the Info-NCE loss [198]. It treats the similarity learning task with a minibatch of size m as a m -class classification problem: Given a minibatch of x_i with positive partners x_i^+ , the model should output the i -th partner for the i -th example. This is achieved by applying the softmax and crossentropy-loss function to the matrix of cosine similarities. Hence implicitly, all other examples in a mini batch are treated as negative samples.

$$\ell(X, X^+) = m^{-1} \sum_{i=1}^m \log \frac{\exp\langle f(x_i), f(x_i^+) \rangle}{\sum_{j \neq i} \exp\langle f(x_i), f(x_j^+) \rangle} \quad (3.12)$$

Usually, it is beneficial to choose the batch size as large as possible. When this is not computationally feasible, we can also use so-called memory banks of older negative examples [285]. Instead of using a square matrix of similarities where all dot products depend on the current batch, we then have a rectangular matrix where only some of the entries depend on the current batch and part of the current computation graph.

Contrastive Learning

We now present another source of supervision: In contrastive learning, we rely on random data augmentations. Here both negative and positive examples are artificially generated. For generating positive pairs, we generate two different views of the same

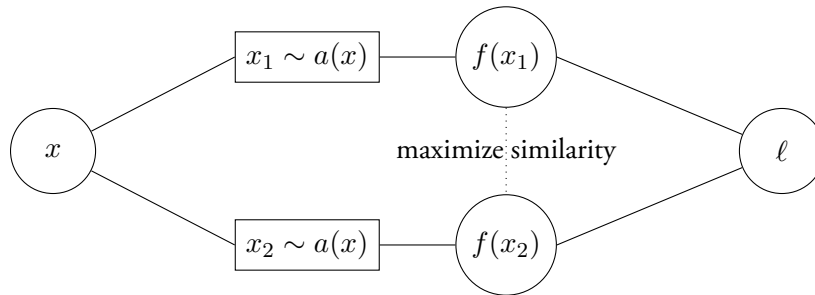


Figure 3.3: Contrastive Learning with Data Augmentations

data point. This can be done using randomly applied data augmentations. When these augmentations preserve the semantic information in the data point and produce examples that are non-trivially different, we can train models using this artificial data [259].

Minibatch-based losses, most commonly the Info-NCE loss, are used for contrastive learning. This approach is summarized in Figure 3.3: Given a data point x , two views x_1, x_2 are randomly generated. Then the same model f is used to compute a vectorial representation. Minimizing a loss function promotes high similarity of the representations.

We can also imagine generating views of different modalities, e.g. separating audio and visual from a video. We can apply the same idea then, but need two different models f_1 and f_2 to embed the different data formats into a shared vector space where the dot product similarity can be computed.

For computer vision tasks, Chen et al, proposed the SimCLR approach for contrastive learning [46]. They propose a set of random image augmentations that has become the baseline for self-supervised learning with images. These augmentations include operations like Gaussian blur, addition of random noise, cutting, scaling, flipping or rotating the image, manipulation of brightness or contrast, or other manipulations of the color channels. Building on this set of augmentations, Grill et al. propose BYOL - bootstrap your own latents - that maintains a running average model that is used for the second view rather than using the same model. This running average model is updated using exponentially smoothed averaging and is detached from the gradient-based optimization. The objective function minimizes the distance between the latent representations of the model and the average model. This way the authors claim that the method works without explicit negative examples. However, further investigation reveals that batch normalization in the model acts a form of implicit negative examples, as embeddings in a batch are spread into the range of a unit Gaussian distribution, thereby prohibiting the collapse of all representations to a constant vector, so-called *mode collapse*, which is the trivial global minimizer of the BYOL objective [76]. Other measures to prevent mode collapse are possible; Richemond et al. find that a combination of group normalization and weight standartization also successfully prevents mode collapse [222].

Zbontar et al. [293] propose an alternative to contrastive losses: Instead of working on the Gram matrix of cosine similarities, they work on the empirical correlation matrix of features between the two views in the representation state. This correlation matrix is computed on batches of data and is optimized towards the identity matrix. Ablation studies show that the normalization term in the computation of the correlation prevents mode collapse in this training objective [293].

3.2.4 Other Self-Supervised Approaches

Finally, we consider other self-supervised approaches that do not fit the above categories.

Autoregressive Tasks

We review tasks based on predicting the next input in a sequence of inputs. For time-series data this is typically called timeseries forecasting. However, we can also use this task to learn representations. Particularly in the natural language processing domain, autoregressive tasks have shown great success. Language models are tasked with predicting the next word given the previous words in a paragraph and trained on large collections of texts. Sometimes this is also called *causal language modeling*. Large transformer-based models like GPT-2 [217] or GPT-3 [31] have demonstrated that they learn useful representations of text for a wide variety of downstream tasks, including few-shot learning [251].

Masking Tasks

Masking tasks exploit the compositional structure of inputs: When an input is composed of parts that are related, we can hide parts of the input and ask the model to predict the hidden parts given the remaining parts. Mikolov et al. proposed masking as a way to learn word embeddings, as discussed above. Devlin et al. made this approach popular for learning representations for text using large transformer models [58]. Given a tokenized sequence of text, we can randomly hide a fraction of the tokens and train the model to reconstruct the original sequence. In this particular case, the reconstruction of each token is a classification task where each possible token is a possible output. The overall loss is given as the sum of all prediction losses over all masked tokens. Instead of asking the model to predict the masked token, Clark et al. propose to randomly exchange some tokens with random tokens and ask the model to predict which tokens are fake [47], turning the pretraining problem into a discriminative task, namely a binary classification problem.

Masking is also popular in computer vision, for instance Lee et al. randomly hide square patches in images and ask the model to pick the original patch from a set of candidates [147]. They show that this discrete treatment of the masking problem as a classification task is superior to predicting the actual image pixels in a regression approach. It has the additional advantage of only requiring a model for encoding

images, but no decoder architecture for outputting images given a latent representation. Hence training is more resource-efficient.

Other Tasks

Finally, we review some more exotic representation learning tasks for learning image representations. Gidaris et al. [85] propose to randomly rotate images in one of 4 ways and task the model with predicting the rotation. This 4-way classification task requires leaning about the objects depicted in the images and how they are usually oriented. Zhang et al. [299] propose to feed grayscale images into a model that has to reconstruct color images. This regression task also requires learning about the objects in the image and what colors are usually present in the detected shapes. Noroozi and Favaro [192] propose a task that is based on the global compositional structure of images: They cut the images in so-called JigSaw puzzles that the model should solve, i.e. they split the image into tiles, and task the model with predicting the In another approach, Noroozi et al. [193] use a task based on counting. Here the sum of counts in 4 distinct parts of the input image should equal the count in the full image. The model is trained without access to ground-truth counts, but has to figure out some feature in the data that fulfils this sum-of-counts property.

All of the above tasks have been demonstrated to be beneficial for downstream tasks like object detection.

Chapter 4

Learning Representations with Unsupervised Tasks

Only mathematics and mathematical logic can say as little as the physicist means to say.

Bertrand Russell

MATHEMATICS as a discipline relies on abstract reasoning and applying transformations to expressions. In this chapter we investigate how to apply machine learning techniques for representation learning to the problem of finding representations for mathematical expressions.

4.1 Introduction

Machine learning has contributed to many success stories of search engines. Unfortunately, the search itself is most often based on words or text. Technical terms in different disciplines, however, may have different meanings or the same meaning may be referred to by different terms. For instance, various usages of the Bayes' law occur in different scientific fields and can be found under different titles. For instance in astrophysics, it is known as *information field theory* [71]. Without knowing physics and even if the name *Bayes* were not mentioned, it is easily recognized by the formula $P(d|s) = P(d, s)/P(s)$ in the paper. Another example is Ising's paper in a physics journal from 1925 under the title *Ferromagnetismus*. Today, the Ising model is also popular in machine learning, but is referred to first as *Hopfield network* and later as *Boltzmann machine*. This illustrates the aspect of time: words for particular topics change over time. The language of Ising's paper is German, the paper introducing Jensen's inequality in 1906 is written in French. Again, the inequality $f((a + b)/2) \leq f(a)/2 + f(b)/2$ can be easily understood, anyhow. We conclude that the most compact and comprehensive way to transport the main ideas of scientific

manuscripts in disciplines like computer science or physics are the formulas used. Thus it should also be the way we formulate our search queries when searching for scientific manuscripts. In order to judge the relevance of mathematical expressions for a search query, a system has to generalize between different notations and match the parts of equations, that describe the same concepts, even if they appear in a different form. A human reader resorts to domain knowledge acquired over years of training in his field to judge the relevance. We wonder how machine learning models with access to vast amounts of mathematical content can help automatize this process.

In this chapter, we propose to use machine learning models to learn a representation of mathematical expressions that captures semantic relatedness. To this end, we design unsupervised learning tasks based on the ideas in Section 3.2, most notably based on contextual similarity and based on self-supervised learning. We curate a dataset of over 28.9 million equations from over 900,000 papers from arXiv.org and explore different ways to represent the data to train our machine learning models.

4.2 Math Search and KDD

Mining and indexing mathematical expressions in document collections is a challenging task, mostly tackled in the information retrieval community [96, 302]. We outline how the problem of math search is treated with the tools from knowledge discovery and data mining and present related work on the machine learning methods we chose for our approach.

4.2.1 Representation

The first question we have to consider is how to represent mathematical expression. Choices can be divided into two categories: those for visually representing and those for semantically representing math. The former category is focused on the layout of an expression. The most prominent choices are LaTeX, a Turing-complete language used in the publications on arXiv.org, as well as Presentation MathML¹, an XML dialect for displaying math on the web that we chose in this work. The latter category includes Content MathML and OpenMath, two similar XML dialects that focus on semantics rather than layout, but also domain-specific languages for symbolic math solvers like Mathematica, that also allow to manipulate and transform formulas. To the best of our knowledge, no large, diverse, and public collection of semantic math expressions exists and, unfortunately, converting math from a display-representation, where data is available in large quantities, to a semantic representation which seems more appropriate for searching, is a non-trivial task. Available solutions either use rules and heuristics, e.g. the converter ml2om that translates LaTeX to OpenMath [261], or also apply machine learning [279]. For the studies in this chapter, we chose to apply machine learning methods directly on visual representations, which are more readily available.

¹<https://www.w3.org/TR/MathML3/>

4.2.2 Similarity Measure

The second question is how we compute similarity between formulas. Zanibbi et al. distinguish text-based, tree-based and spectral approaches [292]. Text-based approaches transform tree-structured math into a sequence, for instance by pre-order traversal, and then estimate the similarity using methods for sequences like cosine similarities of bags-of-words or the length of the largest common substring. Tree-based approaches focus on matching trees or subtrees in hierarchical representations of expressions like XML formats. Typically computing similarities involves solving dynamic-programming problems. Spectral approaches work on paths or partial subtrees in the trees. An example is [302], that indexes root-leaf paths of operator trees. From matches of the root-leaf paths, they compute the largest common subexpression to score the similarity of two equations. To convert math from LaTeX to the semantic representation of operator trees, the authors use ca. 100 handwritten grammar rules.

A new trend is to use machine learning to learn a similarity measure. A machine learning model maps an equation to a dense, low-dimensional vector. The similarity between these so-called embeddings can be computed via their inner product, which enables fast indexing using a variety of index structures, including faiss and annoy, designed for efficiently handling millions of these dense, low-dimensional vectors. Mansouri et al. [161] propose to embed equations using fastText, a method originally designed for computing word embeddings. In this Chapter we explore other machine learning tasks based on ideas presented in Section 3.2 and other models like convolutional neural networks (Section 4.4), transformer models (Section 4.5) and graph convolutional neural networks (Section 4.6).

4.2.3 Retrieval

Approaches developed for the application of information retrieval need to actually allow efficient retrieval. We do not want to compute similarity between a query and all of millions of formulas in a database, as runtime and energy costs are impractical for search engine applications. For similarity measures working on matching sub-structures in expressions, this means that we need to implement index structures like inverted indices [9] for these tasks (see e.g. [302]). For machine learning based approaches, we have to deal with vectorial data. If a model computes a complex relationship between a query formula and a candidate formula simultaneously, designing an index structure becomes difficult [258]. Hence the approaches considered in this chapter reduce retrieval to nearest neighbor queries in vector spaces by using machine learning models to embed the query and candidate expressions separately. Hence we can rely on existing index structures for nearest neighbor queries to implement retrieval. Indexing solutions include annoy, which indexes data using an ensemble of random halfspace trees [16] as well as other approaches based on small-world graphs [160].

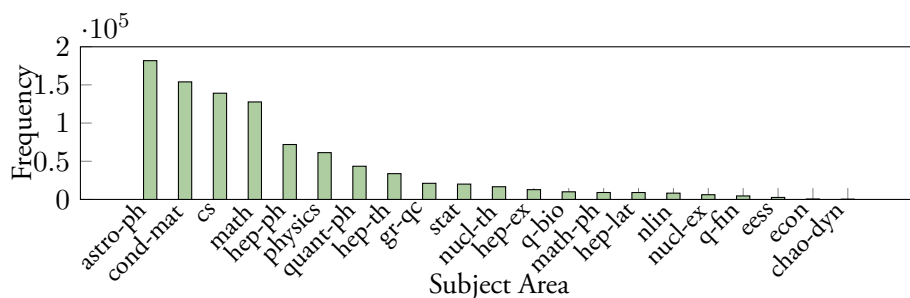


Figure 4.1: Number of Papers per Subject Areas in our Sample

4.3 The Dataset

Data is the fuel that powers machine learning models [271]. In the introduction of this thesis, we have seen examples how large datasets have sparked innovation and performance increases in different domains. We collect and curate a training dataset for the domain of mathematical expressions. In the following, we describe which steps we take.

We are working on data obtained from arxiv.org, a service where scientists can upload their manuscripts or pre-prints without reviewing process. We have downloaded all the LaTeX sources of publications up to April 2019 from the official bulk data repositories². This way we have obtained 934,287 papers. As we can see in Figure 4.1, the large majority of these papers are from disciplines where mathematical expressions are an important part of publications. The most prominent subject areas are astrophysics, condensed-matter physics, computer science, mathematics, and high energy physics. We merge the sources with meta data like paper title, authors, abstract, data of publication, etc. using the arXiv API.

The goal of pre-processing the raw dataset is to provide a list of mathematical expressions for each paper in a way that allows compilation to other display formats. Recall that LaTeX is a Turing-complete programming language rather than a markup language for mathematics. As such, it may not be the most suitable choice for representation due to its complexity; in the course of this chapter we shall investigate different representations that can be obtained through compilation of the LaTeX sources.

Starting with the folder of source files, including at least one LaTeX source as well as images or figures, for a single paper, we begin by identifying the main document. We proceed to resolve all the imports of sub-files to obtain a single source file with all contents in the right order. We divide this single document into parts according to the section structure to provide fine-grained contexts to formulas. From all publications, for each section of the document we extract mathematical expressions by using regular expressions for the most common math-environments like '\$\$', 'equation', 'align', etc. We do not use inline math snippets but focus on expressions that stand on their own, as they tend to describe more important concepts. Many of the formulas

²https://arxiv.org/help/bulk_data_s3

found this way need special LaTeX packages that provide additional LaTeX commands. We automatically detect which packages are used and provide this list to enable compilation. Some packages commonly break compilation, so we have assembled a list of packages to exclude. Similarly, some LaTeX commands commonly break compilation, so we override them with harmless alternatives. Furthermore we extract user-defined commands and macros, which may be prerequisites for compiling the formulas and also include these when compiling the formulae.

Furthermore we have used regular expressions to find arXiv-ids in the bibliographies of the paper to build a citation graph. In total, 540,892 papers have an outgoing edge, with a total number of edges of 4,553,297. Since we only detect those references that use an arXiv-id, for instance in an url, our citation graph is only a subgraph of the true citation graph.

To ensure reproducibility we provide the scripts used for processing the public arXiv data dump, extracting the mathematical expressions and converting them to MathML as well as collecting meta-data and citations at https://github.com/Whadup/arxiv_library. This library is further discussed in the appendix. We also share our citation graph, which might be interesting in other applications.

4.4 Embedding using Convolutional Networks on Bitmap Representations

We first explore the idea of representing formulas as fixed-size bitmap images. While at first this seems like an unsuitable choice, the multitude of machine-learning or computer-vision approaches that successfully transform images of typeset [56] or hand-written [5, 159] math back to tree-based representations suggests that bitmap representations preserve all required information of tree-based approaches. Furthermore, this representation is invariant to the authors' LaTeX programming style, use of macros, etc. There are many ways to write the same formula in LaTeX, however, compiling the different input strings to a bitmap gives the same image. Additionally the LaTeX sources of a scientific publication are often unavailable, particularly on platforms other than arXiv, but images of equations may be identified and cropped from digital documents automatically.

To the best of our knowledge we propose the first system that represents mathematical expressions as bitmap images and uses machine learning to detect similarities. This way we can build on a rich set of works on similarity learning between images. The current boom of deep networks, and in particular convolutional neural networks, has heavily impacted computer vision. One important task is detecting similar images or similar objects in images or videos and a popular architecture for learning these similarities are Siamese networks, i.e. convolutional neural networks that are used to encode two images into a shared vector space where the similarity is measured. Examples of this application include identifying products in different product photos on e-commerce websites [242], person re-identification in photos [62, 300], detection of models of vehicles [155]

$$\min_w \frac{1}{n} \sum_{i=1}^n \ell(y_i, w^T x_i) + \lambda \|w\|^2 \quad \boxed{\min_w \frac{1}{n} \sum_{i=1}^n \ell(y_i, w^T x_i) + \lambda \|w\|^2} \quad (1)$$

(a) Original Formula (b) Bitmap Representation from [211]

Figure 4.2: Illustration of the Bitmap Representation

4.4.1 Data Representation

For this study, we only work on a small subset of publications, a combination of two publicly available crawls of arXiv-ids at Kaggle.com³ and at Andrej Karpathy’s page⁴. The publications in these crawls are mainly from machine-learning related subdomains of computer science. In total, this machine learning subset contains more than 44k papers.

We chose to represent equations as fixed-size bitmaps as illustrated in Figure 4.2. The images we created are 531×106 pixels in size, however for more efficient learning we only use the center rectangle of size 333 × 32 pixels for training. This decrease in height does not affect single-line equations, the decrease in width does not affect equations that fit into a single column in a two-column layout, longer equations will miss beginning and end. We have applied only rudimentary data-cleaning on the images. Most notably, we omit images that are mostly white background and have only few black foreground pixels. The majority of these images are artifacts of failed LaTeX compilation.

This dataset of bitmap images is published at <https://whadup.github.io/EquationLearning/> and we invite the machine learning community to use it.

4.4.2 The Equation-Encoder Model

Our embedding model, the *equation-encoder*, is a standard convolutional neural network. We try two different networks, a small and a large one. The small model is comprised of three convolution layers with 32 channels, each followed by a MaxPooling layer, followed by a final fully connected linear layer, as depicted in Figure 4.3. The larger model uses 6 convolution layers with 64 channels each, every other layer followed by a MaxPooling layer. Then the embedding is computed by two fully connected layers. We apply batch normalization between the fully connected layers for more stable optimization, but freeze that layer in later epochs of training. Both models map into a space of dimensionality $d = 64$. All non-linearities are rectifying linear units (ReLU). In total, the small model has 45,504 parameters, while the number of parameters in the large model is 255,040. As noted before, margin losses require that the magnitude of embeddings is controlled either by applying regularization or normalization. Otherwise arbitrary absolute margins can be satisfied by scaling the

³<https://www.kaggle.com/neelshah18/arxivdataset>

⁴https://cs.stanford.edu/people/karpathy/arxiv_10K_export.zip

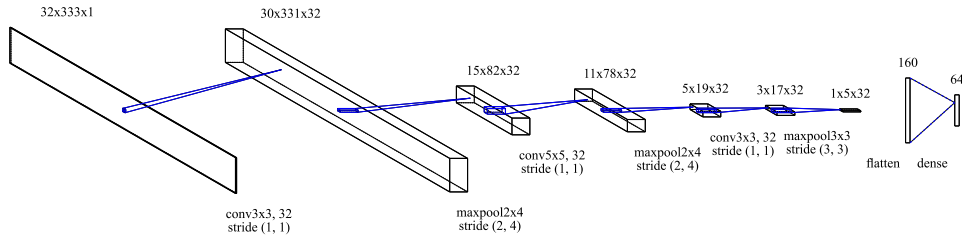


Figure 4.3: Equation Encoder Architecture (small)

output by a large enough values. We choose to normalize our embeddings to unit length $\bar{f}(x) = f(x)/(\|f(x)\|_2 + \epsilon)$ where a small, constant $\epsilon > 0$ is added for numerical stability. This choice avoids the introduction of yet another hyperparameter that controls regularization. We set $\epsilon = 10^{-5}$ in all our experiments.

We train these models using two different contextual embedding tasks. Specifically, we compare using the Siamese loss (3.8) that works on pairs of formulas labeled as either similar or dissimilar, and the triplet loss (3.9) that works on a triplet of formulas including one similar and one dissimilar partner. We have discussed these losses in Section 3.2.3. For our representation learning task, we claim that two formulas are similar if they appear in the same paper. To this end, we select an equation from our dataset uniformly at random. Using this anchor point, we can randomly select other equations from either the same paper or a different paper to obtain the training pairs and triplets.

We train our models using the Adam optimizer and initialize the learning rate at 0.00025 and reduce it by a factor of 0.1 every 10 epochs of training. We train using minibatches of size 100 and run training for 30 epochs.

4.4.3 Empirical Evaluation

In this section we experimentally evaluate the models we proposed in the previous section. First we try to establish quantitative metrics for measuring the quality of our embeddings. To this end, we curate a small evaluation dataset with hand-labeled equations. Second, we do a qualitative evaluation and inspect a few queries and discuss the results.

Gold-Label Evaluation Data

Quantitative evaluation is difficult whenever we cannot compare to a ground truth. This problem arises in many unsupervised learning applications, including clustering [275] and embedding learning [88, 170, 233]. Word Embedding models are often evaluated by computing measures using a small, manually labeled set of word pairs. Following their experimental protocol, we hand-labeled about 100 equations into 13 categories of equations that we have identified as prominent in machine learning literature. These include equations describing the empirical risk minimization principle, the k-means objective, the dual formulation of the kernelized support vector machine,

(q)	$c_i^t = v^T \tanh(W_{hh}h_i + W_{hs}s_t + b_{\text{atten}} + W_{cov}v_t)$	$\mathcal{B}[p_{\sigma}] = \frac{T(\sigma, w_1^n) + 1}{n + \Sigma }$
(1)	$h_0 = f(W_{hh}h_{-1} + W_{hc}c_0 + W_{hs}s + W_{hv}v)$	$\mathcal{B}[p_{\alpha}] = \frac{T(\sigma, w_1^n) + \alpha m_{\sigma}}{n + \alpha \Sigma }$
(2)	$h_0 = f(W_{hh}h_{-1} + W_{hc}c_0 + W_{hs}s)$	$A_d(\kappa) = \frac{I_{d/2}(\kappa)}{I_{d/2-1}(\kappa)}$
(3)	$y_t = \text{softmax}(W_{of}(W_{dec}s_t + V_{dec}d_t))$	$C'_M(\kappa) = -\mathcal{C}_M(\kappa) \cdot \frac{I_{s+1}(\kappa)}{I_s(\kappa)}$
(4)	$\beta_{tj} = v^T \tanh(W_{st-1} + Ux_j + Vh_j)$	$R(\pi) = \frac{ G(\pi) \cap S_{\text{pair}} }{ G(\pi) }$
(5)	$h_0 = f(W_{hh}h_{-1} + W_{hc}c_0 + W_{hv}v)$	$Q(k, \alpha) = C_k \cdot \prod_{i=1}^{\alpha} \frac{C_{i\alpha_i k}}{C_k}$
	(a)	(b)

Figure 4.5: Sample Queries and Results: The first line shows queries, the remaining lines show the top-5 results. Shows center rectangle only.

(q)	$V^{\pi}(s) = \sum_{s' \in \mathcal{S}} \mu(\pi(s), s, s') (r(\pi(s), s, s') + V^{\pi}(s'))$	$\mathcal{L}_r = \frac{1}{T} \sum \log p(Y, z^{(t)}) - D[q(z X) p(z)]$
(1)	$\hat{Q}(s, a) \leftarrow \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \gamma \mathcal{P}(s, a, s') \otimes_{a'} \hat{Q}(s', a')$	$(\theta_x^*, \theta_y^*) = \arg \max_{(\theta_x, \theta_y)} \text{corr}(f_x(X; \theta_x), f_y(Y; \theta_y))$
(2)	$b_{t+1}(\theta^t) = \eta \text{Pr}(s', r s, a, \theta^t) \int_{\mathcal{S}, \Theta} \text{Pr}(s', \theta' s, a, \theta) b_t(\theta) ds d\theta$	$\hat{w}(Y, \theta) \approx p(Y \theta) = \int p(X, Y \theta) dX$
(3)	$\frac{1}{q(\bar{a} s)} (\bar{Q}_{\bar{w}}^{\pi}(s, a) - \bar{r} - \gamma \bar{Q}_{\bar{w}}^{\pi}(s', \mu(\bar{s}'))^2$	$p(z_t^k; X_t, L_{y_t^k}) \sim \mathcal{N}(L_{z_t^k} \ominus X_t, R)$
(4)	$g_x(s' s) = \sum_{a^1, a^2} \pi^1(a^1 s) \pi^2(a^2 s) p(s' s, a^1, a^2)$	$\theta = \arg \min_n \mathbb{E}_{P_{\text{reg}}(z)P(y x)} [-\log \hat{P}(Y X)] + \lambda \ \theta\ _2^2$
(5)	$Q^A(s, a) = Q^A(s, a) + \alpha_t(s, a)(y - Q^A(s, a))$ $Q^B(s, a) = Q^B(s, a) + \alpha_t(s, a)(y - Q^B(s, a))$	$\frac{\partial}{\partial \omega} \log p(y \hat{X}(\omega), \theta)$
	(a)	(b)

Figure 4.6: More Sample Queries and Results: The first line shows queries, the remaining lines show the top-5 results. Shows center rectangle only. The results for these queries are more diverse.

We visually inspect the performance of the large embedding-model in a hierarchical clustering of our labeled evaluation data. The clustering is done using agglomerative clustering with the complete-linkage strategy and dot-product similarity. In Figure 4.4 we see that on the lower levels of the hierarchy many clusters belong to a unique category of equations and that the computed similarities are high. On the right side of the plot, we see that categories mix, but do so in a plausible manner: the connection from sgd (stochastic gradient descent) and convex (equations describing properties like convexity or Lipschitz continuity) is apparent, as articles analyzing the convergence of optimization algorithms rely heavily on these definitions. The connection between Rademacher complexities and concentration inequalities cannot be denied as well.

A Qualitative Analysis of Example Queries

In Figure 4.5 we want to show some example queries with their respective top results⁵. In the first column we query an equation that describes a modified variant of an LSTM recurrent neural network. The system returns equations that seem to describe other recurrent network architectures or parts of recurrent units like the output part in row (3). In the second column the system does not work as well: The first result is relevant, however it is from the same publication as the query. That is a) not very useful for the user, and b) not difficult for the system to identify, as the query-answer-pairs were used during training. The remaining answers seem to have no semantic similarity. However they are at least visually similar: Short equation where an uppercase character function equals a term with a fraction. This difference in quality between the two columns is also expressed in the computed similarities. While the most relevant equation on the left has a similarity of 0.99, the most relevant equation on the right that is not from the same paper, has a similarity of only 0.91.

In Figure 4.6 we see more examples: On the left we query a definition of the value function, an expression related to reinforcement learning. And indeed all results are related to reinforcement learning. Even (2), which looks the most dissimilar, is taken from a survey on reinforcement learning⁶. On the right, the equations appear more diverse, but all involve probabilistic modeling. The computed similarities range between 0.93 and 0.90, which again is substantially lower than on the left.

Overall the quality of the results is convincing, the results contain useful and related equations in most cases. It seems that our system has learned to group equations that belong to a subfield of machine learning together in a latent space. Using a visual representation helps comparing the coarse structure of expressions.

4.4.4 Conclusion

In this Section we have designed embedding models for mathematical expressions based on convolutional neural networks. We have represented data as fixed-size bitmap images and demonstrated that a system trained in an unsupervised manner is useful for identifying related formulas. Our machine learning tasks are using the contextual similarity paradigm and we have found in a comparison of two loss functions for similarity learning that the triplet loss yields better models.

While we have stated valid arguments for using an image representation, we admit that it does not feel like the most natural choice of representation. In the next sections, we will look into alternative representations that treat mathematical expressions as textual data.

⁵In the appendix you can find a table of the corresponding works

⁶Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, Aviv Tamar. *Bayesian Reinforcement Learning: A Survey*, 2016.

4.5 Embedding using Transformers on Sequential Representations

Large natural language models based on transformers [273], i.e. neural networks with self-attention, pretrained on large corpora of textual data have enabled breakthroughs in natural language processing [31, 58, 217]. In this section, we will take inspiration from the textual data domain and design machine learning tasks and models for mathematical formulas represented as sequences of tokens.

Transformer models have been used for information retrieval [220] and also mathematical retrieval, for instance in the ARQ-Math competition [291]. The participants Novotny et al. propose a mathematical retrieval system for question answering that also utilizes a BERT architecture named CompuBERT [197]. The authors finetune a pretrained natural language model on a task that matches questions and answers containing formulas on stackexchange. They use the triplet loss to learn embeddings for questions and answers that capture their connection first proposed in SentenceBERT [220]. The authors also experiment with different representations of math, including working on pure LaTeX and on MathML trees, serialized via pre-order iteration as in this work. However, as they rely on pretrained NLP BERT models, they do not carefully design a vocabulary to express the mathematical formulas within the natural text more appropriately. Ultimately, they use the CompuBERT architecture in an ensemble, where the strength of the CompuBERT seems to lie in grasping the semantics of the textual parts of question and answer, and the other two systems, including Formula2Vec, an approach based on FastText document embeddings [124], increase the quality in retrieval of formulas. Similar to Novotny et al., Rohatgi et al. use a RoBERTa model trained on natural language posts containing LaTeX formulas to solve the question-answering task of the ARQ-Math competition [223].

In this section, we explore transformer models for the retrieval of related mathematical expressions. Unlike the works discussed above, we train a transformer model solely on the mathematical formulae of our dataset. We explore a representation based on a markup language for mathematical expressions, MathML and feed a sequentialized variant of these XML-documents to a transformer model to embed formulas in a vector space.

We examine if our BERT model learns useful representations for advanced mathematical tasks by finetuning pre-trained and not pre-trained models for a discriminative tasks of identifying equalities or in-equalities.

4.5.1 Data Representation

While in the last section, we have represented mathematical expressions using bitmap images, in this section we explore an alternative: representing formulas as text-like sequences. To convert the formulas from the multi-purpose LaTeX language to a more normalized form, we rely on the XML-based MathML format for rendering mathematical expressions on webpages. Using the library `Katex`⁷ we compile the raw LaTeX

⁷<http://katex.org>

formulas to MathML. The MathML standard defines around 30 different XML-tags like `<mi>` for math identifiers or `<mo>` for math operators. Some of these tags use attributes, for instance to change font or spacing. Leaf nodes contain text like numbers, parenthesis or letters (greek, latin, etc...). We see an example of an equation encoded as MathML in Figure 4.7. Out of all papers downloaded, 760,041 papers contain at least one equation that we were able to convert to MathML. In total we have a dataset of 28,973,591 MathML equations.

In order to feed the MathML to a transformer model that works on sequential data, we flatten the tree-based XML using in-order tree traversal, where opening and closing tags are placed around any child nodes. We one-hot-encode the resulting sequence of tags and tokens, but use the same encoding for opening- and closing tags to reduce the number of distinct tokens. We truncate the vocabulary to a total size of 512, which captures 98.8% of token occurrences in the dataset. This is a notably smaller vocabulary than commonly found in natural language models, where vocabulary sizes above 30,000 are common [58, 217].

```

<math xmlns="http://www.w3.org/...">
  <semantics>
    <mrow>
      <mfrac>
        <mn>1</mn><mi>n</mi>
      </mfrac>
      <msubsup>
        <mo>∑</mo>
        <mrow>
          <mi>i</mi><mo>=</mo><mn>1</mn>
        </mrow>
        <mi>n</mi>
      </msubsup>
      <mi mathvariant="normal">ℓ</mi>
      <mo stretchy="false">(</mo>
      <mi>f</mi>
      <mo stretchy="false">(</mo>
      <msub>
        <mi>x</mi><mi>i</mi>
      </msub>
      <mo stretchy="false">)</mo>
      <mo separator="true">,</mo>
      <msub>
        <mi>y</mi><mi>i</mi>
      </msub>
      <mo stretchy="false">)</mo>
      <mo separator="true">,</mo>
    </mrow>
  </semantics>
</math>

```

Figure 4.7: Example MathML for $\frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i)$.

4.5.2 Training Transformer Models

The original transformer model — as proposed by Vaswani et al. [273] — is slightly modified in BERT [58], which is only using encoder layers. In our work we use the same transformer model architecture as BERT - including the same encoder layers, activation functions, optimization algorithms and learning rate schedules. The transformer architecture introduces the multi-head-attention layers as key mechanism for learning the relations between each pair of tokens in the input sequence. This is applicable on mathematical formulas too, because understanding the relations between the symbols of a mathematical formula is crucial for understanding the meaning of the formula. We also extend the vocabulary by the special classification, separation, masking and unknown token – as did in [58] – in order to perform prediction of masked tokens and thereby integrate in the model the ability to correct mathematical expressions.

We explore three differently sized variants of the BERT architectures for embedding mathematical expressions: While BERT has a hidden-size-dependent number of attention heads, we keep them constant. We set the number of different multi-head-attention heads D to 4. By doing so the hidden size H has the largest impact on the performance of the multihead attention. As for the intermediate projection size we kept this always bigger than the hidden size so that we can have a linear projection on a higher space. The resulting models are summarized in Table 4.2.

Table 4.2: Math-BERT model configurations

Model	L	H	I	D	Params	GFLOP
SMALL	4	128	768	4	1.2m	0.7
BASE	8	256	768	4	6.0m	3.6
LARGE	12	512	768	4	25.0m	15

We train these models using the original masking task proposed by Devlin et al., we have discussed masking in Chapter 3.2 and will further discuss this task in more detail in the next section. The original BERT model also includes a classification task where the model has to guess if two sequences fed into the model simultaneously are consecutive sentences in a shared document. We exchange this task with the contextual task used in the last Section and classify, if two formulas are from the same paper instead. We use the ADAM optimizer used in the original BERT paper.

4.5.3 Retrieval of Equalities and Inequalities

We have extracted equalities and inequalities in the test-set of our data using regular expressions. Using a simple heuristic, we filter the resulting (in-)equalities, such that left-hand-side (LHS) and right-hand-side (RHS) do not differ in length dramatically, thereby eliminating formulas like definitions, where the LHS is a only single symbol. We derive three different data sets, one with only equalities (LHS and

RHS split at "="), one with inequalities (split at $<$ and \leq) and one with mixed relations (split at $=<>\leq$ and \geq). This data allows us to use the LHS of the (in-)equalities as query in hopes of retrieving RHS. We make our finetuning-data available at https://whadup.github.io/arxiv_learning/ as well.

Following other machine-learning-based approaches for mathematical retrieval [161, 209, 211], we use our models to encode formulas into a dense vector space and retrieve results using approximate nearest neighbor search [16] in this vector space. In the case of our BERT models, we use output embedding of the CLS token as the representation for the whole formula and finetune the model to output meaningful embeddings for this first token. We finetune our models on half of the available data and test on the remaining half.

Finetuning Task We propose to use contrastive learning to learn to identify the RHS given the LHS. The learning task in contrastive learning is identifying the right partner for each input in a minibatch of datapoints. Hence the representation learning problem is formulated as a classification problem. Let $X^l, X^r \in \mathbb{R}^{m \times d}$ contain the output embeddings of a minibatch of LHSs and RHSs. We normalize each embedding to unit-length and denote the normalized embeddings by \bar{X}^l and \bar{X}^r . We use the InfoNCE loss [198] presented in (3.12) in Chapter 3.2, i.e. the negative log likelihood of softmax probabilities parameterized by the pairwise cosine similarities between the LHs and RHSs:

$$\ell_\tau(\bar{X}^l, \bar{X}^r) = m^{-1} \sum_{i=1}^m \log \frac{\exp(\langle \bar{X}_i^l, \bar{X}_i^r \rangle / \tau)}{\sum_{j \neq i} \exp(\langle \bar{X}_i^l, \bar{X}_j^r \rangle / \tau)} \quad (4.1)$$

where $\tau > 0$ is a hyperparameter that controls the temperature of the output probability distribution, which we set to 10^{-2} . The contrastive learning task is more difficult for larger batchsizes m , as there are more candidate RHSs to choose from and thus the underlying classification problem becomes more difficult. But it has been shown that the utility of the model increases for larger batchsizes [46, 171], which we also investigate in our application.

Baseline-Models In addition to our models we include several baseline models:

- First, we test a simple *bag-of-words* (BoW) model that is trained on a bag of MathML tree nodes. This model does not use a pre-training phase, but is only tuned on the finetuning data. The BoW model maps the sparse BoW representation to a d -dimensional vectorial embedding through a single matrix multiplication. In comparison to our BERT models, we do not restrict the vocabulary size of the inputs. The representation is trained using the same contrastive learning task with InfoNCE loss. We test $d \in \{64, 128, 256\}$ and report the best result after varying learning rates and number of training epochs in a grid search.
- Second we evaluate a pretraining approach based on the BoW-model. The word-embedding based approach *fastText* by Joulin et al. [124] is trained by predicting

which tokens appear in the contexts together. Mansoury et al. use it for learning embeddings of formulae by serializing a MathML layout tree similar to the one we are using in this work [161], hence we include it in our comparison. We finetune these embeddings by learning a linear mapping into a d -dimensional vector space, $d \in \{64, 128, 256\}$ using the same contrastive learning task.

We begin by training our models and the baseline-models with a minibatch-size of 1024. Then we also investigate the effect of varying the batch-size.

Results For testing, we compute embeddings for all LHSs and RHSs in the test-data and store them in an index structure. We use annoy [16], an indexing method for approximate nearest-neighbor search based on an ensemble of random projection trees. We use an ensemble of 16 trees with default hyperparameters, but we found that the results were very insensitive to our particular parameter choices.

Then we query the k -nearest neighbors, $k \in \{1, 10, 100\}$ for each formula from the test-set and check if the corresponding other side of the (in-)equality is in the result set. This way we can compute recall values to measure the quality of our embeddings.

We summarize our findings in Table 4.3. Our BERT-approach substantially outperforms both the BoW approaches, without (BOW) and with pretraining (FAST-TEXT). This suggests that our model is capable of matching formulas based on characteristics that go beyond merely counting the number of matching tokens.

Throughout all our models we observe the benefit of pretraining, as models that were trained from scratch – with exception of the small model – perform worse than their pretrained counterparts.

Overall, the recall at 10 is already pretty high, which indicates that our approach is useful in search engine applications where users generally want to inspect only a small number of results. In summary, we recommend using the BASE pre-trained model.

Table 4.3: Results of the Mathematical Retrieval Experiment. We report recall@K for $K \in \{1, 10, 100\}$.

Model	Equalities (36864)			Relations (40960)			Inequalities (13312)		
	R@1	R@10	R@100	R@1	R@10	R@100	R@1	R@10	R@100
SMALL-PRE	0.379	0.577	0.714	0.432	0.626	0.749	0.397	0.661	0.795
SMALL-NO-PRE	0.400	0.584	0.700	0.405	0.632	0.769	0.371	0.632	0.769
BASE-PRE	0.511	0.71	0.805	0.503	0.697	0.791	0.484	0.765	0.860
BASE-NO-PRE	0.434	0.623	0.729	0.446	0.63	0.734	0.409	0.682	0.797
LARGE-PRE	0.507	0.704	0.799	0.496	0.683	0.777	0.489	0.765	0.864
LARGE-NO-PRE	0.452	0.637	0.737	0.46	0.640	0.736	0.427	0.703	0.817
BOW	0.483	0.653	0.739	0.491	0.658	0.743	0.503	0.738	0.821
FASTTEXT [124]	0.480	0.650	0.739	0.488	0.651	0.742	0.488	0.713	0.810

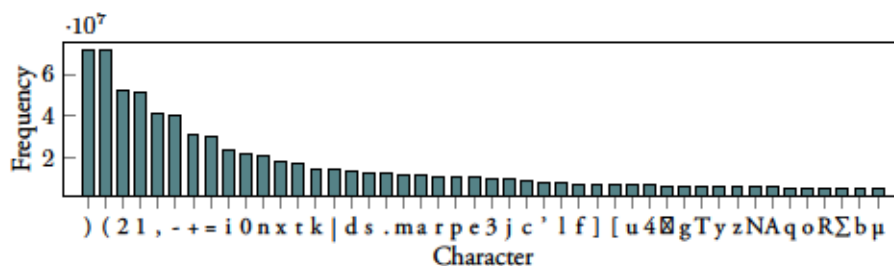


Figure 4.8: The 50 most frequent characters in math environments.

4.6 Embedding using Graph Convolutional Networks on Tree Representations

We reviewed representation choices for mathematical retrieval in the beginning of this chapter. We found that the bottom line of the representation question is that math is expressed in trees, either XML or other parse trees. Consequently, in this Section, we investigate a family of machine learning models that can work directly on tree-structured data. Specifically, we design a graph neural network for obtaining vectorial representations of mathematical expressions.

4.6.1 Data-Representation

We want to use a tree-based representation derived from the MathML representation we introduced in the last Chapter. In order to feed the MathML to a graph convolutional neural network, we have to convert it to a graph with vectorial node features.

We view the XML-structure as a tree and use its nodes and edges and derive features based on tags, attributes and text. For each node we use one-hot encoded feature vectors of dimensionality 256. The first 32 features are used to encode the type of the XML-tag. The next 32 features are used to encode optional attributes, most commonly changes of the font to bold or calligraphy fonts. The remaining 192 dimensions are used to encode the most frequent characters used in leaf-nodes. In Figure 4.8 we see that the most frequent characters are opening and closing parenthesis, followed by a variety of numbers, latin or greek letters and mathematical operators. For both attribute and character features, we introduce special unknown symbols for all rare attributes/characters. We distinguish three groups of features: tag, attribute and content. We encode this data using a total of 256 features where we use 32 dimensions to encode the tags, 32 dimensions to encode the most frequent attributes (including one special symbol for unknown attributes) and the remaining 192 dimensions to encode the most frequent characters of the content (also including one unknown symbol). In addition to the one-hot encoded features, we store the position of the node amongst its sibling nodes.

4.6.2 Training Graph-Neural-Network for Embedding Formulas

We define a graph convolutional neural network for the task of embedding mathematical expressions into a low-dimensional vector space. The raw MathML is converted to graphs with vectorial features as described in Section 4.6.1. We propose to use a special first layer that combines the one-hot encoded information at a node with the decimal position attribute. Following Vaswani et al. [273], we encode the position of the i -th node $p_i \in \mathbb{N}$ using positional embeddings. We use fixed sinusoid embeddings [273] denoted by $E(p_j)$, but to still allow the model to control the influence of the positional embeddings, we introduce a learnable scaling coefficient α initialized to 1.

$$x_i^{(1)} = \max \left(\vec{0}, \sum_{j \in \mathcal{N}(i) \cup i} W^{(1)} x_j + \alpha E(p_j) + b^{(1)} \right)$$

The first layer is followed by 3 fully-connected graph convolution layers of width 512, where the l -th layer is defined by

$$x_i^{(l)} = \max \left(\vec{0}, \sum_{j \in \mathcal{N}(i) \cup i} W^{(l)} x_j^{(l-1)} + b^{(l)} \right)$$

which linearly transforms all nodes using a weight matrix $W^{(l)}$, adds a bias term $b^{(l)}$, aggregates by computing the sum over all neighborhoods and applies the ReLU activation component-wise. All graph convolution layers output feature maps with 512 dimensions. We apply batch-normalization before the first and third graph convolution layer. For the remainder of this paper, let $\phi(x) \in \mathbb{R}^{|x| \times 512}$ denote the output of the last graph convolution layer given the input x . To obtain a single embedding for an input graph, we compute the mean of all node features. This mean is transformed in another linear layer to reduce the dimensionality to 64. For the remainder of this paper, let $\bar{\phi}(x) \in \mathbb{R}^{64}$ denote this embedding of x .

When scoring similarities between embeddings with margin losses, we need to control the norm of the embeddings, otherwise the notion of adherence to a margin becomes meaningless. Among others, Ding et al. [62] propose to normalize all embeddings to unit length. We propose a softer normalization inspired by batch normalization [115] that also allows to obtain embeddings with norms smaller than 1. For every training batch of graphs, we compute the mean of the norm as well as its standard deviation. Then we inversely scale each embedding by the mean plus the standard deviation. This way, most embeddings have norm smaller than 1. We keep a running average of the means and standard deviations. At inference time, we use these running averages for scaling.

We propose to train our embeddings using two self-supervised learning tasks simultaneously by adding their respective losses.

Contextual Similarity

For learning relations between equations, we rely on the established contextual similarity task that we introduced in Section 3.2. The central idea of the learning task is that objects that frequently appear in shared contexts are related. We define the context of mathematical expressions as the paper containing the equation and conjecture that two equations are related if they appear in the same paper, as in Section 4.4. We extend this approach and further define two equations as related if one paper references the other using a citation graph. This way we hope to connect equations that describe the same context but use different notation. In addition, we discriminate between sampling expressions from the same paper and from the same section. We hope that within sections, equations are more related to each other. For obtaining positive examples of related equations, we

1. sample a paper uniformly at random and select an expression from this paper uniformly at random.
2. randomly select whether we sample from the same section, same paper or along a citation,
3. sample a positive example using that method. When we cannot find a positive example using that method, we jump back to (1).

For learning similarities we also require negative examples. To obtain these, we sample a paper uniformly at random and select an expression from this paper uniformly at random. The random process that generates these weak labels for similarity learning introduces a lot of noise, as many equations we claim are related are unrelated and some of the pairs we say are unrelated are related. We leave the investigation of more advanced sampling schemes to future work.

Using the sampled equations x with positive x^+ and negative partners x^- , we apply similarity learning loss functions to design an objective. We investigate two different losses: Triplet and Histogram. The triplet loss (3.9) proposed by Balntas et al. [10] we have previously used in Section 4.4, contrasts the similarity between a positive pair of examples and a negative pair of examples. For this study we also investigate the histogram loss (3.11) as proposed by Ustinova and Lempitsky [269]. It does not work on a single triplets of equations, but on mini-batches of size m positive pairs X^+ and a batch of negative pairs X^- with respect to anchor examples X . We hope that histogram loss is more robust with regard to the massive noise in our labels as each positive example is contrasted with all negative examples.

Masking Task We propose to extend the contextual similarity task by another tasks and optimize the sum of both tasks for training our embedding models. The main idea of our second task is, that the symbols in mathematical expressions do not appear independent from each other, but have strong dependencies. Thus if we hide a fraction of the symbols in an equation, we should be able to approximately reconstruct the hidden symbols from the remaining symbols. This task is reminiscent of masked

$$\min_{\blacksquare} \frac{1}{n} \sum_{i=1}^n \ell(\langle w, \blacksquare_i \rangle, y_i)$$

$$P(\blacksquare = ?) = \begin{pmatrix} w : 0.81 \\ \beta : 0.04 \\ \vdots \end{pmatrix} \quad P(\blacksquare = ?) = \begin{pmatrix} x : 0.73 \\ y : 0.02 \\ \vdots \end{pmatrix}$$

Figure 4.9: Example of the Masking Task with Fictional Values

language modeling tasks made popular by BERT [58] seen in the last Section. In order to successfully solve this task, a model has to learn about the frequencies of symbols and their dependencies from the data, as is illustrated in Figure 4.9.

More formally, we proceed as follows: For each input graph x with features X , we randomly set the feature vector of 15% of the nodes to all zero obtaining the graph x_{\blacksquare} . Then we compute $\phi(x_{\blacksquare}) \in \mathbb{R}^{|x| \times 512}$. Now for each masked node, we want to solve three separate classification tasks: Given $\phi_i(x_{\blacksquare})$, predict the right XML-tag, predict the right XML-attributes (or no-attribute if no attributes were used) and predict the right character (or no-character if no character was used). We solve these tasks using a single linear layer of dimensionality 32,32+1 and 192+1 respectively with soft-max activation and compute the cross-entropy loss ℓ on all tasks:

$$\begin{aligned} \ell_{\text{tag},i} &= \ell(\text{softmax}(W^{(\text{tag})} \phi_i(x_{\blacksquare}) + b^{(\text{tag})}), X_{i,1:32}) \\ \ell_{\text{attr},i} &= \ell(\text{softmax}(W^{(\text{attr})} \phi_i(x_{\blacksquare}) + b^{(\text{attr})}), X_{i,33:64}) \\ \ell_{\text{char},i} &= \ell(\text{softmax}(W^{(\text{char})} \phi_i(x_{\blacksquare}) + b^{(\text{char})}), X_{i,65:256}) \end{aligned}$$

The overall loss of the masking task is defined as the mean of all three classification losses $\ell_{\text{mask},i} = \frac{1}{3}(\ell_{\text{tag},i} + \ell_{\text{attr},i} + \ell_{\text{char},i})$. The loss is only evaluated for the masked tokens and we compute the mean over all masked tokens to obtain a loss value for x_{\blacksquare} .

Adding this task to the contextual similarity task has the additional advantage that we now learn a representation that not only captures context information, but also preserves information about the raw input symbols.

Data-Augmentation

Data augmentation eases the generalization of machine learning models and is particularly popular for image classification tasks where we can augment images by randomly rotating, scaling, padding, etc. For mathematical expressions, we propose the following random data augmentation: Since we know that a renaming of symbols in equations

rarely changes the semantic, we propose to randomly permute the character features of all nodes that correspond to a math identifier, encoded in $\langle \text{mi} \rangle$ tags according to the MathML standard. For each equation we process, we sample a number of flips from a Poisson distribution with expected value 32. Then starting with the identity permutation that does not change the order of our 192 features, we construct a permutation with the desired number of flips by incrementally exchanging two random characters.

4.6.3 Statistical Analysis

Before we present empirical results on our embedding approach, we want to discuss their statistical significance using concentration inequalities. We begin by discussing the assumptions we use for our discussion that go beyond the usual iid. assumptions. Then we talk about testing of embedding models on hold-out data.

Assumptions

In usual classification settings [272], as well as metric learning settings [48, 163], we assume that we have access to independent and identically distributed training data. In our case of embedding learning for mathematical expressions, we would require to a sample of independent equations, or to be more precise, independent pairs of equations with (weak) similarity labels. However, this assumption surely does not hold, as two equations that appear in the same paper do not appear independently from each other. An example illustrates this: Let X_1 and X_2 denote two equations. If it is revealed to you that X_1 shows Heisenberg’s uncertainty principle $X_1 = \{\sigma_x \sigma_p \geq \hbar/2\}$, and that X_2 appears in the same paper, the probability of X_2 being related to quantum mechanics increases. This illustrates that $P(X_2 = x_2 \mid X_1 = x_1) \neq P(X_2 = x_2)$ and thus X_1 and X_2 are not independent. For the purpose of our analysis, we assume that

(A1) we have an iid. sample of N papers where the i -th paper contains n_i equations

(A2) the number of equations is $n = \sum_{i=1}^N n_i$

(A3) equations in the i -th paper are independent of equations in all other papers.

Surely papers do not appear independently of one another, but this we will ignore in our statistical analysis.

Confidence Intervals for Hold-Out Data

Now we carry out our statistical analysis for measuring scores on hold-out data where the model $\bar{\phi}$ is fixed and independent of the hold-out data. In the common iid setting, we can apply concentration inequalities like Hoeffding’s inequality to bound the gap between performance evaluations like accuracy or loss measured on a finite hold-out sample and the true expected value.

Since we do not have iid. data, we have to use a refined approach. We use U -statistics [111] s to analyze our models. In our case, they are defined as expectations over functions of triples of equations $V(x_1, x_2, x_3)$ called kernel.

$$s = \mathbb{E}_{x_1, x_2, x_3} V(x_1, x_2, x_3).$$

For instance we can define the ranking score, i.e. the fraction of triplets where the positive pair has a higher similarity than the negative pair. For convenience, we define $P(x)$ as the set of possible positive examples for an expression x . Then we can write the ranking score as

$$V_{\text{ranking}}(x_1, x_2, x_3) = \left[\mathbb{1}(x_2 \in P(x_1) \wedge x_3 \notin P(x_1)) \cdot \right. \quad (4.2)$$

$$\left. \mathbb{1}(\langle \bar{\phi}(x_1), \bar{\phi}(x_2) \rangle > \langle \bar{\phi}(x_1), \bar{\phi}(x_3) \rangle) \right] \quad (4.3)$$

The histogram loss can also be expressed as a U -statistic:

$$V_{\text{hist}}(x_1, x_2, x_3) = \left[\mathbb{1}(x_2 \in P(x_1) \wedge x_3 \notin P(x_1)) \cdot \right. \quad (4.4)$$

$$\left. \sum_{r=1}^R \sum_{r'=1}^r \delta_{r'}(\langle \bar{\phi}(x_1), \bar{\phi}(x_2) \rangle) \delta_r(\langle \bar{\phi}(x_1), \bar{\phi}(x_3) \rangle) \right]$$

Note that whenever (x_1, x_2, x_3) is not a triple with a positive and a negative example, the triple contributes zero to the expectations. To simplify the analysis, we first consider so-called complete U -statistics, i.e. we estimate the true score using all possible triplets from a finite set of n expressions $\{x_i \mid i = 1, \dots, n\}$

$$\hat{s} = \binom{n}{3}^{-1} \sum_{i,j,k} V(x_i, x_j, x_k). \quad (4.5)$$

Now the goal is to bound the difference between s and \hat{s} in high probability.

Theorem 1. *If $V(x_1, x_2, x_3) \in [0, 1]$, then for $\delta \in (0, 1)$ with probability at least $1 - \delta$ we have*

$$s \leq \hat{s} + 3\sqrt{\frac{\ln(1/\delta)}{2N}}$$

Proof. We prove this using Janson's concentration inequality for sums of partly dependent variables [123]. In Equation (4.5), we compute a sum over triplets, where the triplets are constructed from dependent samples. Thus the summands have dependencies. We construct a dependency graph, where each node corresponds to a triplet and two nodes have an edge if they are dependent. In our case two triplets are connected if one of the equations in the first triplet is from the same paper as any equation in the second triplet. In order to apply [123], Theorem 2.1, we have to bound the chromatic number χ^* of this dependency graph. To this end, we consider an arbitrary node v in

the graph with equations from paper i , j and k . Its degree is bounded by the number of equations in the papers $\deg(v) \leq (n_i + n_j + n_k) \binom{n-1}{2}$, hence

$$\chi^* \leq 3 \binom{n-1}{2} \max_i n_i. \quad (4.6)$$

Then for $t > 0$ we have

$$\mathbb{P}(s - \hat{s} \geq t) \stackrel{[123]}{\leq} \exp\left(\frac{-2t^2 \binom{n}{3}}{\chi^*}\right) \quad (4.7)$$

$$\stackrel{(4.6)}{\leq} \exp\left(\frac{-2t^2 n}{9 \max_i n_i}\right) \leq \exp\left(\frac{-2t^2 N}{9}\right) \quad (4.8)$$

Solving for t yields the desired confidence interval. \square

For incomplete U -statistics, where we estimate the score by a subset of triplets D sampled independently with replacement [48] as

$$\hat{s}_C = \frac{1}{|B|} \cdot \sum_{(x_i, x_j, x_k) \in D} V(x_i, x_j, x_k),$$

the bound (4.6) no longer holds. But we can compute an empirical bound $\hat{\chi}$ using any greedy graph coloring algorithm. Then Janson's concentration inequality implies that with probability at least $1 - \delta$

$$s \leq \hat{s}_C + \sqrt{\frac{\ln(1/\delta) \hat{\chi}}{2|D|}}. \quad (4.9)$$

4.6.4 Experimental Results

In this section we perform an experimental evaluation of our embedding model. In particular, we focus on the use-case of a search engine for mathematical expressions. We begin by investigating the effects of the individual components of our model on a small, closed subset of the data. Then we investigate the effectiveness of our method on all 29,9 million equations.

Analysis on the Machine-Learning-Subset

We begin our analysis only on arXiv publications where the primary subject classification is machine learning (cs.LG). This is a natural choice, as we have some expertise to judge the quality of our results, a task which we are in no way equipped for across all subject fields.

Of these 9,936 publications, we sample two subsets, train and test of size 7,949 and 1,987 respectively with a total number of equations of 237,335 and 54,767 respectively. In Section 5 we have seen that scores evaluated on hold-out data converge to true empirical scores in $\mathcal{O}(1/\sqrt{N})$ where N is the number of papers in the test set.

Table 4.4: Ablation Study

Influence Factor	Ranking Hold-Out	Ranking Eval	Accuracy Eval
Full Model	76.5 (± 0.0)	57.7 (± 0.0)	60.6 (± 0.0)
No Histogram Loss	72.5 (-4.0)	49.6 (-8.1)	30.9 (-29.7)
No Masking	75.2 (-1.3)	54.3 (-3.4)	50.0 (-10.6)
No Augmentation	75.3 (-1.2)	53.6 (-4.1)	50.0 (-10.6)
Bitmap CNN from Section 4.4	76.2 (-0.3)	71.9 ($+14.2$)	68.3 ($+7.7$)
Bitmap CNN retrained	70.0 (-6.5)	50.0 (-7.7)	52.9 (-7.7)

In this regard, our test set is appropriately sized. We use the train-set for building our embedding models and use the test-set to investigate generalization properties.

For training, we sample 1 million triplets (x, x^+, x^-) . Of these triplets, 45.9% have a positive pair from the same section, 42.2% from the same paper and 13.9% along an edge in the citation graph. We sample 100k triplets for testing with similarly distributed positive examples.

We perform an ablation study on our proposed embedding model and compare it to prior work. This section investigates the influence of our design choices. We decided (a) to use the Histogram loss instead of the triplet loss, (b) to also add an masking task, (c) to data augmentation.

We measure Ranking score, i.e. the fraction of all triples in the training data where same-class pairs of equations have higher similarities than across-class pairs. As we see in Table 4.4, our evaluations indicate that all of our design choices contribute favorably to the overall performance on hold-out data, as deactivating any component decreases the score. We note that the biggest gain is achieved by switching from triplet-loss to histogram-loss. We believe that this is due to the massive noise in our labels.

We also compare with the our previous model from Section 4.4 and see that we beat it by a small margin. However this comparison is not entirely fair, as their model was trained on a larger dataset of around 25k papers, probably including some of the papers in our test set. We use their code to re-train on our subset of equations and yield a substantial margin of 6.5 percentage points.

We also use our previous evaluation data (Sec. 4.4). It consists of 103 equations labeled into 13 categories related to machine learning including k-means, LSTMs, empirical risk minimization, etc. Since only bitmaps are available, we transcribe the equations manually. There are three issues with this evaluation set: First, it is too small to produce significant numbers. Second, some equations in the dataset appear in the training data. This is not only the case for our subset, but also for the training data used in Section 4.4. Third, many equations within a category are obviously from the same paper, hence we have seen some of the pairs in our training data. Nevertheless we use the evaluation data. Indeed in our use-case of search engines, the crawled equations will always be in the training data and only the user queries will be unseen equations.

Table 4.5: Eval Scores

Dataset	Ranking Eval	Accuracy Eval
1mio ML-Subset Triplets	57.7	60.6
5mio Full ArXiv Triplets	76.2	80.9
20mio Full ArXiv Triplets	75.3	84.0
Bitmap CNN (Sec. 4.4)	71.9	68.3

In a way, we simulate this with the eval data.

Following the original experimental protocol, we measure the 1-nearest-neighbor accuracy obtained in leave-one-out validation (named Accuracy) as well as the above Ranking score. In Table 4.4, we again see that our model is only surpassed by the pre-trained model that uses a larger training dataset. This motivates the use of a much larger dataset.

Large-Scale Experiments

For training on all the papers in our dataset, we sample two different sets of training triplets, one with 5 million triplets and one with 20 million triplets. We train our models on a Nvidia GTX1080 GPU with 8GB memory, which allows us to process mini-batches of 128 triplets or 384 equations. During training, we process around 1,300 triplets per second, not counting the time for reading data from hard disk. In total, one of the 20 epochs of training on 20mio triplets takes 6:30h on our system. We use annoy to construct an index for approximate nearest neighbor retrieval. In total, our index uses 13GB of hard disk storage to manage all mathematical expressions in our dataset.

Before we evaluate our models in a search engine study, we again check the performance on the aforementioned evaluation data. The results in Table 4.5 indicate the power of using large amounts of training data, although it is unclear if using 20mio training triplets is an advantage over using only 5mio. Our large-scale models beat all the models trained on smaller amounts of data. Even though the smaller models were trained on only machine-learning related data, we obtain better scores on the machine learning evaluation data by training on all disciplines.

Let us now inspect two example search queries. In Figures 4.10 and 4.11 we see two examples from the introduction, Bayes law and Ising models, and their respective nearest neighbors under our model trained on 5mio triplets. We see that we can find other definitions of Bayes' law as well as the related law of total probability. When we perform a query for the Ising Model and look at the first 20 results, we find papers where the model is called Boltzmann machine as well as papers that refer to the Ising model. This illustrates the power of querying for mathematical expressions instead of using keywords.

Table 4.6: Results of the Mathematical Retrieval Experiment. We report recall@K for $K \in \{1, 10, 100\}$.

Model	Equalities (36864)			Relations (40960)			Inequalities (13312)		
	R@1	R@10	R@100	R@1	R@10	R@100	R@1	R@10	R@100
Transformer Sec. 4.5	0.511	0.71	0.805	0.503	0.697	0.791	0.484	0.765	0.86
BoW	0.483	0.653	0.739	0.491	0.658	0.743	0.503	0.738	0.821
FastText [124]	0.480	0.650	0.739	0.488	0.651	0.742	0.488	0.713	0.810
Graph-CNN	0.507	0.833	0.884	0.512	0.834	0.883	0.504	0.870	0.922

Identifying Equalities and Inequalities

We repeat the experiment of retrieving the right-hand-sides of (in-)equalities given the left-hand-sides of Section 4.5. We again use the contrastive-learning task to fine-tune our graph-convolutional neural network that was pretrained using the masking and contextual similarity task.

As we can see in Table 4.6, our graph-cnn network substantially outperforms both baseline methods – bag-of-words and FastText – but also the transformer model proposed in Section 4.5. With one exception, we can report the best recall scores on all 3 subsets of formulas, often by quite substantial margins.

Search Engine Study

Finally we want to study the usefulness of our embedding approach for a search engine application more systematically. Traditionally, validating search engines using measures like precision or recall, requires relevance scores for each result for each evaluation query. We see that this requires a lot of manual annotation work since we have to manually identify each relevant equation for each query. Unfortunately, we were not able to find available evaluation data. The best fit is the NTCIR-12 task evaluation data [292] consisting of 37 annotated queries. However is not appropriate for our approach, as most queries are a combination of math as well as keywords. When we ignore the keywords, the remaining query become very generic, for instance $x + y$, which makes it very unlikely that we accurately find the articles labeled as relevant. In addition, the overall focus of the NTCIR-12 task is recovery of exact matches, whereas

$$\begin{array}{l}
 \text{Query:} \quad P(d | s) = \frac{P(d,s)}{P(s)} \\
 \text{1st Result:} \quad P(s | d) = \frac{P(d|s)P(s)}{P(d)} \\
 \text{4th Result:} \quad P(d) = \int P(d | s)P(s)ds
 \end{array}$$

Figure 4.10: Example: Bayes’ law. We report the first result and the first result that does not show Bayes’ law, but, in this case, the related law of total probability. The first result is from: R. H. Leike, T. A. Enßlin, *Charting nearby dust clouds using Gaia data only*, 2019.

Query:	$\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i$
'Boltzmann' Result:	$E = -\sum_i b_i s_i - \sum_{i<j} w_{ij} s_i s_j.$
'Ising' Result:	$\mathcal{H} = -\sum_{i<j} C_{ij} J_{ij} \sigma_i \sigma_j - \sum_i h_i \sigma_i$

Figure 4.11: Example: Ising Model. We find equations related to both Ising Models and Boltzmann Machines. First result is from: Weinstein, *Learning the Einstein-Podolsky-Rosen correlations on a Restricted Boltzmann Machine*, 2017. Second result is from: Ferrari et al., *Finite size corrections to disordered systems on Erdős-Rényi random graphs*, 2013.

our focus is on retrieving *related* expressions.

Consequently, we curate and publish our own evaluation data. To reduce the manual annotation labour, we want to apply a heuristic for the relevance judgement. To this end, we have asked our colleagues, many from disciplines other than computer science and data science, to provide us with equations that we should query. For each equation, they provide a set of keywords or key-phrases that should appear in the section around the result. If one of the keywords is present, we count the result as correct. This way we can evaluate our search result without manually checking result lists. If a keyword has more than 10 characters, we also count it, if we find a substring that has a Levenshtein distance less than 2. In total, we have 53 evaluation queries publicly available and editable online⁸.

We inspect two different information retrieval metrics that do not require to know the number of relevant documents in advance: Precision@ k and unnormalized Mean Average Precision. Precision@ k is defined as the fraction of relevant documents within the first k results. We report it for lists of 10, 100 and 1000 results and compute its mean over our evaluation queries.

Unnormalized Mean Average Precision is derived of the standard mean average precision metric. Since we do not know the number of relevant documents in advance, we omit this term, limit the search to a maximum of 1000 results and obtain the following definition

$$\text{uMAP} = \sum_{k=1}^{1000} P(k) \Delta_k$$

where $P(k)$ is Precision@ k and Δ_k specifies if the k -th result is relevant. Again we compute the mean over all evaluation queries. In comparison to Precision@ k , uMAP considers the order of the search results and rewards relevant results early in the result lists.

For reference, we include retrieval based on a bag-of-words representation. To this end, we use our data representation as in Section 4.6.1, but compute the sum over all nodes in the graph to obtain a single 256-dimensional vector of the whole tree. We retrieve the nearest neighbors using cosine similarity.

⁸Crowd-sourced evaluation data can be accessed and edited here: <https://www.overleaf.com/8721648589nrjxgwmtzfvn>.

Table 4.7: Search Engine Performance

	P@10	P@100	P@1000	uMAP
BoW	0.4567	0.3170	0.2083	106.17
5Mio	0.5038	0.3817	0.2984	165.04
20Mio	0.4547	0.3709	0.2897	156.51

In Table 4.7, we see that our approach beats the bag-of-words margin, in particular for larger values of k . We see for Precision@10, the performance between BoW and our embedding model is very similar. This is because for many queries the top-10 results are mostly near-perfect matches which are easily identified even without machine learning. However when we look at more results, we are able to find almost 50% more relevant equations.

4.7 Conclusion and Outlook

Finding relevant literature even across disciplines is essential for scientific investigations. The search results should entail stimulating, relevant papers. Very often, a look at the formula in a paper gives a compact description of the problems and solutions discussed in the paper. Hence, the goal is to offer related papers based on the mathematical expressions. In this chapter we have demonstrated how unsupervised representation learning can train useful representations for mathematical expressions that allow us to find related expressions in large collections of formulae. We have investigated three different ways to represent the training data and have proposed models for each data format. This way, we have progressed from a convolutional neural network to a graph-convolutional neural network and found that the tree-based representation was indeed superior to both bitmap and sequential representations.

We have curated a huge dataset with over 29 million mathematical expressions based on over 900,000 papers hosted on arXiv.org and designed a number of evaluation tasks to measure the quality of our models. Particularly, on one of these evaluation tasks, we have shown the benefit of our search system using a new dataset of annotated math queries. Our study suggests that future search engines for scientific literature could benefit from considering mathematical expressions as a means to formulate queries and from using machine learning models for mathematical formulas for retrieval.


Viewed from a broader perspective, we see that machine learning is increasingly used for problems in mathematics, e.g. in symbolic mathematic software or computer algebra systems. For instance, Arabshahi et al. [7] use a synthetic data generator to generate equalities. Then they train tree-recurrent neural networks to decide if two formulas are equal. Wanklerl et al. [282] solve issues with the data generator that allowed models to exploit shortcuts to make decisions and extend the scope of the work by also including other relations between formulas like derivatives. Transformer models have been shown to solve differential equations or integrals [143, 191].

Since publishing our work [209] we have seen many works on pretraining large language models on scientific mathematical expressions crawled on arXiv.org for various use-cases. Similar to our study in Section 4.5, Peng et al. [204] also investigate the use of transformer models for mathematical retrieval. But applications are more diverse than that. Researchers at Google demonstrate the usefulness of these models for mathematical reasoning tasks [216]. By using arXiv data and MATH [104], a dataset of mathematical problems with step-by-step solutions, Lewkowycz et al. [150] train Minerva, a transformer model that achieves state-of-the-art performance on several quantitative reasoning datasets. These datasets include high-school to undergraduate level mathematical problems as well as other multi-step reasoning problems.

Bridging the gap between applying models to the highly standardized quantitative reasoning problems or integral and differential equation problems and to the complex, diverse, ambiguous, and often underspecified mathematical expressions found in scientific literature remains an open problem. However, we have demonstrated that a machine learning model enables retrieval that goes beyond merely comparing symbol frequencies. Combining more data sources, as pioneered by Lewkowycz et al. [150] might be one ingredient for further improvements. The other ingredients that need further research are the family of models used for computing embeddings and the training tasks used. Whether or not the tasks investigated in this Chapter can be solved with near-perfect precision using only large datasets and neural network models without a “symbolic” reasoning component is still a debate in the artificial intelligence community [146, Sec. 8].

Chapter 5

Learning Representation with Multitask Learning and Noisy Labels

STRONOMY, literally translated, is the science that studies the laws of the stars. Modern astronomy relies on machine learning models for data analysis, and we explore training these models based on multitask learning and noisy label learning in this chapter. To study stars, other celestial objects or – more broadly put – our universe, modern astronomers observe high energy particles emitted from celestial objects. Analyzing these energy beams allows to draw conclusions on the key characteristics of the sources. For example, different types of supernovae can be found by observing high energy particles [38]. Large international collaborations deploy a wide variety of detector hardware including telescopes [6, 133, 205] to observe different ranges of electromagnetic beams.

In the FACT experiment [6] that detects highly energetic gamma rays using a Cherenkov telescope, we have to solve three central machine learning problems in the detector. The first is the distinction between gamma rays which may indicate a celestial object and background noise which is mostly produced by cosmic rays from hadrons that do not allow to conclude on a particular source – the *gamma-hadron separation problem*. Secondly, we want to estimate the energy of the recorded particle. And third we need to reconstruct its origin, i.e. the position of the detected source in the view pane of the telescope camera, which is particularly challenging in the mono-telescope setting of the FACT experiment.

Machine learning has been established as an effective tool for data analysis in modern high energy particle experiments and for solving the above-mentioned prediction problems. Current approaches use a basic hardware trigger, i.e., they begin the recording of an event when sufficient energy hits the detector. From the telescope recording we can obtain the number of photos recorded by each pixel in the telescope camera, giving us an image of the event, as illustrated in Figure 5.1. Then, a complex analysis pipeline calibrates the data and extracts pre-defined features which are used for classi-

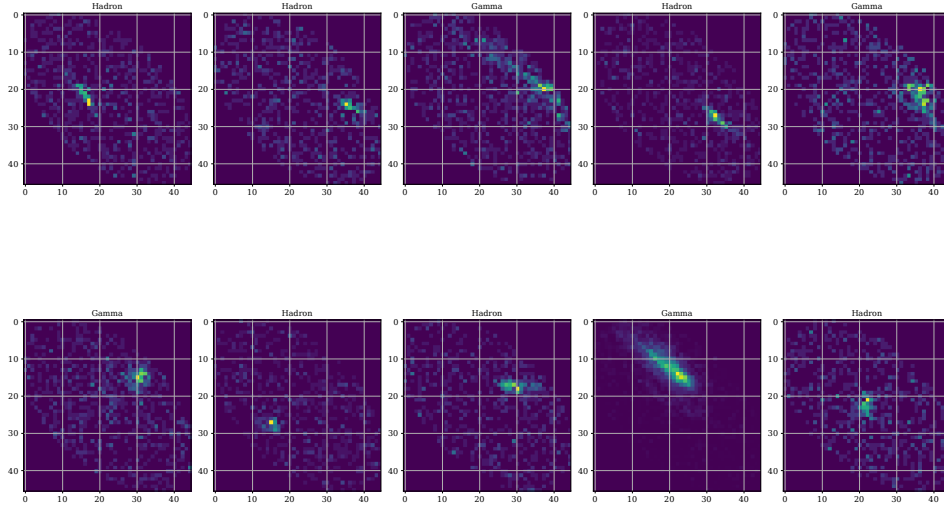


Figure 5.1: Simulated air showers of both classes – gamma and hadron – as captured by the FACT telescope.

ifying the stored event as hadron or gamma rays. The current models rely on Random Forest classifiers on these hand-crafted features.

These hand-crafted features include simple aggregate statistics like the number of pixels that recorded photons or the sum of overall photons. It also includes more advanced statistics based on image processing, like the number of islands, i.e. the number of connected blobs. One noteworthy feature set are the so-called Hillas parameters [107]. The Hillas parameters describe an ellipsis that best fits the shower image through its width, length and orientation. The Hillas parameters are informative for gamma hadron separation, but even more for estimating the source position of a particle. From domain knowledge we know that the origin position lies on the major axis of the Hillas ellipse that fits the shower image in the camera, at a certain distance from the image center of gravity [64, 77]. Using this insight, the task of origin estimation can be reduced to estimating this distance (which can be negative), a one-dimensional regression task. Noethe proposes to decompose this task into a regression task of the absolute distance and a classification task of its sign [194] and solve these tasks with a random forest regressor and classifier, respectively.

All of the above machine learning problems are traditionally tackled as supervised learning tasks where annotated training data is produced in sophisticated simulations [22, 195]. In this chapter, we investigate two more variants of supervision for

machine learning: Supervised training with *multiple supervision sources* and supervised training with *noisy labels*. Currently, three separate models are trained and deployed. But, since all tasks depend on the same telescope measurements, we view the problem as a multi-task learning problem and train a shared model based on multiple sources of supervision, one source for each task. We outline how we can replace the manual feature engineering currently applied with a learned representation trained with traditional full supervision. Our approach will train a shared representation that can solve all three prediction tasks with specialized prediction networks build on top of the shared representation. This parameter sharing increases the inference efficiency of our models. Furthermore, tasks may benefit from shared properties of the prediction tasks. We design loss functions for each task and, more importantly, propose a novel way to combine the different loss functions in a way that accounts for their different scales and convergence behaviors.

Furthermore, we look into an alternative source of supervision that reduces the burden of simulating training data by using real telescope recordings. We rely on the concept of noisy labels and introduce a novel method for learning under label noise where only one noise rate is known. We show how gamma-hadron separation can be framed in this setting and illustrate that the method allows us to train accurate classifiers. For this result, we first use the hand-crafted features of the current pipeline in conjunction with random forest classifiers. In a next step we proceed to train the deep networks presented in the first chapter using our novel noisy-label learning approach.

5.1 A Supervised Multi-Task Representation Learning Approach

The current machine learning pipeline in the FACT experiment heavily relies on the manual features designed by a domain experts. We wonder, whether we could replace the long pipeline by a deep learning process. Deep learning is supposed to decrease the burden of feature engineering, as the network already learns a suitable feature representation. Is this true for gamma astronomy? Works in other telescope projects suggest the benefits of using deep learning [118, 190, 243]. However, deep learning is widely known to be resource hungry requiring not only vast amounts of training data, but also GPU hours to train and apply models. Future monitoring facilities will be installed around the world, often in remote positions, for a round-about view of the sky [226]. They need to detect an interesting gamma event fast, so that they can notify the other telescopes, which then turn in order record the event from their angle. In modern multi-messenger astrophysics, even different types of detectors inform each other so that the same event can be verified by different measurements. So we have to not only consider the quality of our model predictions, but also take into account the resource constraints of executing the models. These constraints include the inference time for enabling fast reaction times, but also energy demands when powering high performance computers in remote locations is not feasible.

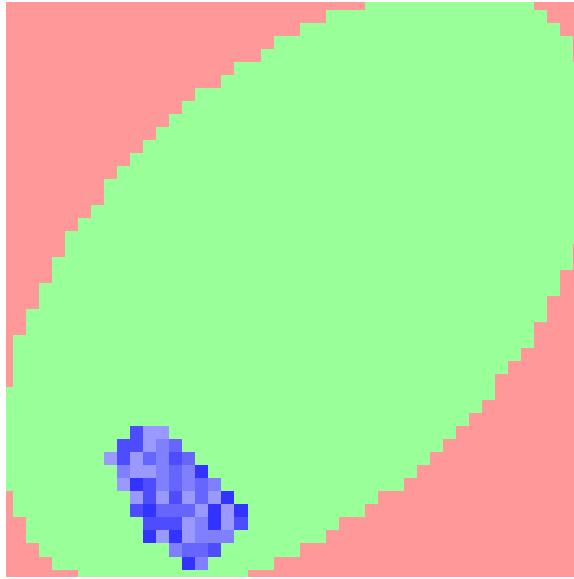


Figure 5.2: Hexagonal data transformed to a quadratic grid. The transformation is bijective in the green (and blue) area and does not change the raw values, however it only approximates the neighborhood relationships of the original hexagonal image. Red areas are padded with zeros. Figure taken from [35].

5.1.1 Supervision by Simulated Data, Data Representation and Pre-Processing

When applying machine learning in astrophysics, it is difficult to obtain labeled data. A common approach to solve this problem is to combine Monte Carlo simulations with a careful training of the classifier. Astrophysics has a profound understanding of particle interactions in the atmosphere: Given the energy and direction of some parent particle (gamma, proton, etc.), its interaction can be described by a probabilistic model which gives a probability for particle collisions, possibly resulting in secondary particles, which again may interact with each other. This results in a cascade of levels of interactions that form the air shower, which can be simulated by software like the CORSIKA simulator [102]. The output is a simulated air shower, which needs to be run through a simulation of the telescope and camera device to produce realistic raw data mimicking a shower that would have been recorded using the telescope. We can simulate interesting particles (e.g. gamma) and uninteresting particles (e.g. proton) and label the resulting raw-data accordingly. Furthermore we can simulate particles of different energy levels and control the source position of the simulated gamma rays.

For our study, the simulation is run in the so-called diffuse mode that simulates gamma particles originating from positions distributed uniformly in the camera pane.

In this study, we want to apply a multi-task deep network in the data analysis pipeline of the FACT telescope. In our training data, each event is represented as an image where each of 1440 pixels contains a photon count represented as a positive integer.

This photon count representation is based on the single-photon extraction for FACTs SiPM sensors proposed by Mueller et al. [181] and is implemented in the `photon-stream` software library [185]. The `photon-stream` library allows to extract the arrival times of individual photons on a per-pixel level, hence we could work on this event sequence of timestamped photon arrivals. To reduce the complexity, we just work on the number of photons per pixel for each event.

In the camera, these pixels are arranged in a hexagonal grid. However, in contrast to [1], we chose to embed these pixels in a regular, quadratic grid of size 45×45 where the additional pixels are padded zero values, as depicted in Figure 5.2. This allows us to use regular convolutional neural network rather than networks specialized for hex-grids. Given the image data, the three prediction tasks, gamma-hadron separation, energy estimation and origin estimation, that we will introduce in detail in the next section, should be tackled, generating annotated events for further downstream analysis. We try to solve them by a single deep neural network without the need for manual feature engineering.

5.1.2 Multi-Task Deep Neural Network

We design a model that solves all three machine learning tasks simultaneously as also proposed by Jacquemont et al. [119, 120]. It utilizes a shared encoder architecture that extracts features from the raw photon count images. These features are passed into three separate prediction heads. The whole network is trained jointly on all tasks. We begin by discussing each task individually and present the corresponding prediction heads as well as loss functions. Then we present our approach for combining the losses to train with three prediction heads.

Gamma-Hadron Separation is a binary classification task. Early approaches based on manually-designed decision rules were already quite successful, also simple machine learning rules based on histogram features already achieve good classification performance. The random forest approaches outlined by Nöthe [195] achieve excellent classification results that are outperformed on simulation data by deep learning approaches like the one proposed by Buschjaeger et al. [35]. For application on real telescope recordings, this performance advantage on simulation data does not increase source detection performance, hinting at overfitting to peculiarities of the simulation [35], an issue that may be addressed through domain adaptation techniques [65].

For the classification task we rely on the standard cross-entropy loss function. The classification head outputs a probability \hat{y} that the recorded event is a gamma event. We minimize the negative log-likelihood of the ground-truth label

$$\ell(\hat{y}, y) = -y \log \hat{y} + (1 - y) \log(1 - \hat{y}). \quad (5.1)$$

To ensure that the outputs are indeed valid probabilities, we apply the sigmoid activation $\sigma(a) = 1/(1 + \exp(-x))$ after the final affine layer.

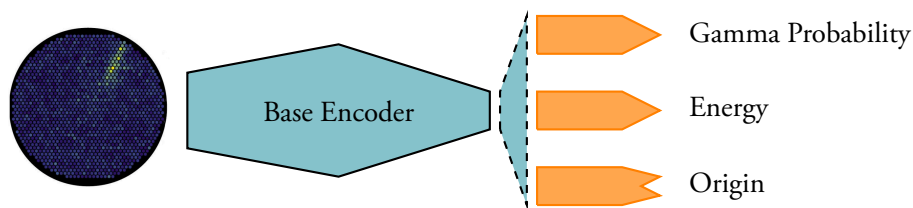


Figure 5.3: Schematic of our Multi-Head Architecture

Energy Estimation is a univariate regression task. We use an affine function to output the prediction given the last hidden representation. Instead of using an absolute error, we prefer the relative error that normalizes the difference between target and prediction by the true energy level. We use the loss function

$$\ell(\hat{y}, y) = \frac{|y - \hat{y}|}{y}. \quad (5.2)$$

and apply the exponential function as activation function after the last layer to account for the power-law distribution of energies. This presents an alternative to minimizing the squared difference between the log-target and the prediction, another approach to dealing with the power-law proposed by Nöthe [195].

Origin Estimation is a two-dimensional regression task. We have already discussed an approach based on the disp-method that utilized domain knowledge: it separates the task into a 1d regression task and a classification task by assuming the event originates on the main axis of the reconstructed shower axis [195]. However, since we no longer estimate Hillas parameters, but want to rely on a fully-learned analysis pipeline parameterized by a deep network, we instead tackle the problem as a 2d-regression task and try to predict the position of the source on the camera-pane. This approach uses less domain knowledge, which might be a disadvantage, but we might eliminate errors where the reconstruction of the shower axis is flawed.

Our prediction head uses an affine map to compute the coordinates given the last hidden representation. For downstream tasks like source detection, it is important that the angle θ between the predicted source position and the actual position is as small as possible. We compute this angle θ accounting for the geometry of the telescope and use it as the loss function that is minimized during training.

Multi-Head Architecture Our multitask network is structured as in Figure 5.3: We utilize a shared base encoder, i.e. a convolutional neural network that takes the input images and maps them into a latent vector space. Then this vectorial representation is fed into three separate prediction heads, i.e. multilayer perceptrons that take the representation and compute a one- or two-dimensional output for each task.

For this study, we rely on different EfficientNet architectures [257] for our base encoder. The prediction heads alternate fully-connected layers of a fixed width with

batch normalization layers and ReLU activations for a specified number of layers and compute output using a single affine layer with the desired number of outputs, as described above.

Multi-Head Loss Combining the three loss functions outlined above is not a trivial task, because they have very different loss value ranges. This results in different magnitudes of the gradients used during the neural network training. Consequently, tasks with large gradients dominate the training. An ad-hoc solution is figuring out a set of weights and optimize a weighted sum of the losses. This however turned out to be a tedious task, particularly when experimenting with different losses or transformations of the targets.

We propose to use a normalization layer inspired by batch normalization that estimates a normalizing constant for each loss. This constant is supposed to correspond to the inverse of a loss value that is easily obtained after a short amount of training, a kind-of default loss. Hence we measure the loss relative to this default loss.

Let $l^t = (l_1^t, \dots, l_k^t)$ be the vector of individual losses observed at the t -th weight update for $t \in \mathbb{N}_0$. We compute the following aggregates loss \bar{l}^t

$$w^t \leftarrow (1 - t^{-\alpha})w^{t-1} + t^{-\alpha}l^t \quad (5.3)$$

$$\bar{l}^t \leftarrow \frac{1}{k} \sum_{i=1}^k \frac{l_i^t}{w_i^t} \quad (5.4)$$

where $\alpha \geq 1$ controls the speed of convergence of w to the default loss. We use $\alpha = 1.2$ in all of our experiments, but preliminary studies found only little effects of varying this hyperparameter. To further increase the stability of our multi-head loss, we clip individual loss values at a maximum value of $2w^t$. This prevents the undesired effect that high losses incurred on rare, very-high energy particles create a normalization constant that is too large: so large that training of the head comes to a halt and so large that it never recovers a useful value during the remaining training process.

5.1.3 Experiments

Experiments are designed to answer two key question: First, can CNNs replace hand-crafted features with reasonable accuracy on simulated data? Second, will these models generalize well enough to be used for real data?

Models Architectures and Hyperparameters For our study, we investigate deep networks based on the EfficientNet architecture which we train from scratch for our application. This family of deep networks contains 7 different sized models. Instead of the traditional classification head, we use our multi-head architecture. We use heads of depth 3 with a fixed width of 512 neurons. In Table 5.1, we see the different resource requirements for the architectures considered in this study. The run-times measured for fitting the deep network are measured on a Nvidia DGX-A100 using a single GPU.

Table 5.1: Comparison of the different model architectures used for our study.

Model	Number of Parameters	Fit Time in Seconds
efficientnet-b0	13,540,617	4,542
efficientnet-b1	16,046,253	6,694
efficientnet-b2	18,062,479	7,892
efficientnet-b3	21,877,725	10,367
efficientnet-b4	30,345,765	14,079
efficientnet-b5	42,721,037	18,886
efficientnet-b6	56,666,077	24,860
efficientnet-b7	81,234,685	33,882

We repeat each experiment 10 times and report means and standard deviations of all metrics.

Simulation Studies In Table 5.2, we see that the performance on holdout-simulation data is very good across all architectures. The classification performance of the gamma-hadron-separation task is traditionally measured via area-under-the ROC curve and all models score roughly the same. Similarly, the origin estimation does not depend on the architecture. The energy error, however, is larger for the 3 smallest architectures. In addition, one training run for `efficientnet-b7` failed to produce a good model for the energy estimation, resulting in the outlier mean and standard deviation score. The performance of origin estimation is measured via the average angle θ between the true and the estimated origin. A more detailed analysis in Figure 5.4 (left) reveals that the deep-network analysis outperforms the random-forest-based analysis substantially for lower-energy, while it is not competitive for higher energies. This so-called angular resolution plot shows the 68% quantile of the angles between true and estimated origin for different energy levels. A similar effect can be observed for the estimation of the energy: We see in Figure 5.4 (right) that for lower-energy particles our deep network approach has a lower bias than the random forest analysis and that this effect reverses for higher energies. Here bias is defined as the median of the relative L1-error and resolution is the difference between the 84.1% quantile and the 15.9% quantile of the relative L1-error, or intuitively the width of a confidence interval around the median error [195, Eq. 6.5].

Two orthogonal approaches to further improve our performance come to mind: Using weighted sampling, we can increase the importance of the rarer higher-energy particles during training, hopefully improving our results in the tail. Additionally, the combination of a random forest model with our deep network looks promising for getting a good performance across all energy levels.

Looking at the resource consumption of our models, we note that the network using the `efficientnet-b0` architecture can still be executed efficiently and fast. On an Apple Macbook Pro 2020, using a single Intel i7 CPU core, it takes $4.48237 \pm$

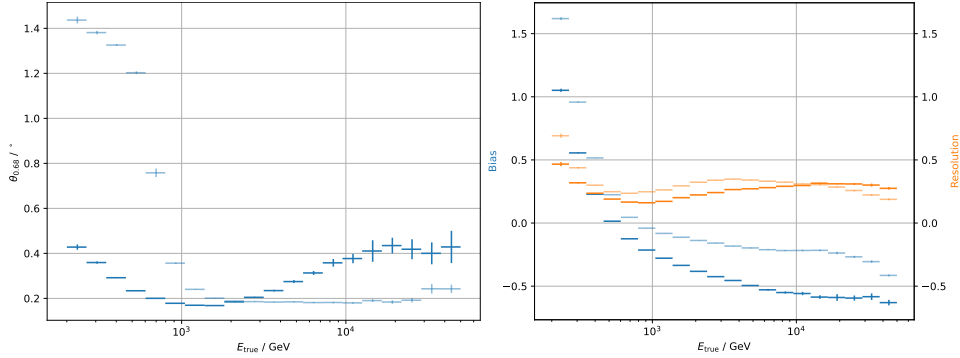


Figure 5.4: Angular resolution of the origin estimator for different energy levels (left) and bias and resolution of the energy estimator. Transparent markers show the performance of the random-forest-based estimators.

Table 5.2: Performance of the three prediction heads on simulation data of the different multi-task model architectures.

Model	AUC-ROC \uparrow	Direction θ \downarrow	Energy Error \downarrow
efficientnet-b0	0.8939 ± 0.0020	0.3253 ± 0.0035	0.3646 ± 0.0856
efficientnet-b1	0.8972 ± 0.0026	0.3194 ± 0.0040	0.3540 ± 0.0767
efficientnet-b2	0.8968 ± 0.0038	0.3236 ± 0.0079	0.3407 ± 0.0603
efficientnet-b3	0.8953 ± 0.0040	0.3226 ± 0.0061	0.3273 ± 0.0515
efficientnet-b4	0.8898 ± 0.0066	0.3477 ± 0.0606	0.3248 ± 0.0543
efficientnet-b5	0.8883 ± 0.0083	0.3263 ± 0.0100	0.3383 ± 0.0530
efficientnet-b6	0.8929 ± 0.0035	0.3190 ± 0.0020	0.3294 ± 0.0345
efficientnet-b7	0.8897 ± 0.0041	0.3237 ± 0.0059	0.3936 ± 0.1476
efficientnet-b8	0.8876 ± 0.0087	0.3281 ± 0.0130	0.3217 ± 0.0103
SOTA Random Forest	0.7720 ± 0.0004	0.6127 ± 0.0006	0.3275 ± 0.0006

0.3082 ms to tag a single event using `onnx-runtime` [57], which is sufficiently fast for the typical arrival times in the FACT experiment without the need for specialized hardware.

OpenCrab Source Detection Now we evaluate our trained models from the last section on real-world data collected by the FACT telescope at the Observatorio del Roque de los Muchachos (La Palma, Canary Islands, Spain). The telescope has been directed once towards a known gamma source, the Crab Nebula. A significantly higher number of gamma ray candidates originates from the direction of the known gamma ray source than from other directions. On these recorded data, we run the full source detection pipeline of the FACT experiment and investigate the influence of the gamma-hadron separation models on the overall quality of the source detection.

The evaluation proceeds in the following steps: We take the publicly available Crab

Nebula observation data [6, 17], which consist of 17.7 hours or 3,972,043 recorded events. Our multitask model is applied to the events to estimate the probability of an event being a gamma particle and to estimate its direction of origin.

From the direction, we compute the angle between the trajectory of any incoming ray and the known direction of the Crab Nebula. The number of gamma rays can be regarded as a distribution that depends on the angle. High counts are to be expected for small angles. The distribution with respect to the direction of the Crab Nebula is called an *on-distribution* [77]. Contrasting the distribution of counts with distributions for five different positions with no known gamma sources yields the *off-distributions*. A uniform distribution of counts over angles is expected, where the majority of counted rays can be attributed to misclassified hadronic rays. We state the null-hypothesis that on-distribution and off-distribution follow the same distribution, or, intuitively, that there is no gamma source at the direction of Crab Nebula. The margin by which a significance test rejects this null-hypothesis gives us a significance of detection $S_{Li\&Ma}$, reported by the number of standard deviations σ [151].

Definition 2 (Li & Ma statistic). Let $N_{\text{on}}, N_{\text{off}}$ be the number of events in the on- and off-positions. Let $\alpha = \frac{1}{k}$ where k is the number of considered off-regions. Then we define

$$S_{Li\&Ma}(N_{\text{on}}, N_{\text{off}}) = \left[2N_{\text{on}} \cdot \ln \left(\frac{1 + \alpha}{\alpha} \cdot \frac{N_{\text{on}}}{N_{\text{on}} + N_{\text{off}}} \right) + 2N_{\text{off}} \cdot \ln \left((1 + \alpha) \cdot \frac{N_{\text{off}}}{N_{\text{on}} + N_{\text{off}}} \right) \right]^{1/2}.$$

This $S_{Li\&Ma}$ is the performance metric for gamma source detection, where larger numbers are better. A more detailed discussion of this statistic is delayed until next section.

The trained classifiers output probabilities that an event is a gamma ray and we can control the classification behavior by varying the threshold for actually predicting gamma. A large threshold yields less events and also less misclassified events, because the classifier is more certain. If we set the threshold too large, we get too few total events which results in a small statistical significance. In contrast, if we decrease the threshold, we obtain more events, but also more misclassifications. If we set the value too small, we count too many noise events, the difference between on- and off-distribution shrinks, which also yields a low significance of detection. Following common practice in gamma-ray astronomy, we chose the threshold that maximizes the significance of detection.

As we can see in Table 5.3, the significance of detection is large for all models we tested. Particularly the smaller models up to `efficientnet-b3` obtain large mean significances of detection that seem to outperform the current state-of-the-art random forest models. However we must note the large variance between the different runs of the examples. The standard deviation is one order of magnitude larger than for the random forest models. Unfortunately, we often observe this pattern for deep networks [27, 213]. Another drawback is that we do not see that the performance measured on

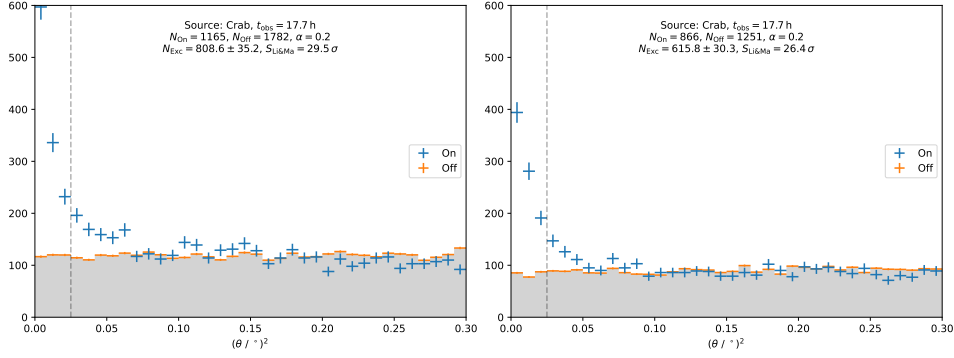


Figure 5.5: Histogram of the frequencies of gamma predictions as a function of the squared angular distance between the trajectory of any incoming ray and a position in the sky for `efficientnet-b0` (left) and the standard analysis using random forest (right). On-events show the frequency with respect to the position of the Crab Nebula, while Off-events are w.r.t. positions with no known sources. The significance of detection test only considers angles smaller than 0.025 (left of the dashed vertical line).

Table 5.3: Significance of Detection (Def. 2) on Crab nebula data of the different model architectures.

Model	Mean \pm Std-Dev	Min	Max
<code>efficientnet-b0</code>	26.85 \pm 0.57	26.07	29.51
<code>efficientnet-b1</code>	27.28 \pm 1.38	23.71	29.26
<code>efficientnet-b2</code>	27.02 \pm 1.17	25.03	28.77
<code>efficientnet-b3</code>	26.73 \pm 1.10	25.49	29.24
<code>efficientnet-b4</code>	26.80 \pm 1.04	24.87	28.44
<code>efficientnet-b5</code>	26.49 \pm 1.16	24.40	27.75
<code>efficientnet-b6</code>	25.64 \pm 0.90	24.33	26.85
<code>efficientnet-b7</code>	25.64 \pm 1.21	23.01	27.12
<code>efficientnet-b8</code>	25.66 \pm 0.41	25.18	26.10
SOTA Random Forest	26.25 \pm 0.15	-	-

simulated data (Table 5.2) can predict the detection scores. For practitioners, there are not only disadvantages: If we have the budget to train a number of candidate models, we can pick one that works really well. Looking at the maximum reported detection scores, we see that they exceed 29.0σ , which marks a substantial improvement upon the state-of-the-art.

We conclude that our multitask model not only works on simulation data, but also provides meaningful results on real-world source detection tasks. In Figure 5.5 we see, how the `efficientnet-b0` model achieves a higher significance of detection than the current random-forest-based approach: Comparing the number of on-events and off-events we see that it predicts more gamma events in the on-region, yielding a stronger signal of the source.

5.1.4 Discussion

Machine Learning is one of the basic building blocks of modern high-energy astroparticle experiments. The FACT telescope measures Cherenkov light emitted in earth's atmosphere when hit by gamma beams. The telescope measures at a rate of 60 events per second, better telescopes even measure orders of magnitudes faster, motivating the demand for fast processing pipelines. This is even more crucial for large telescope arrays in one location or a global distribution of sites that are possibly being deployed in resource constraint and remote locations. Current approaches solve the event-tagging problem for FACT by using long processing pipelines that extract hand-crafted features which are then used by RFs. In this section, we successfully replaced this entire prediction pipeline with a single multitask CNN. We have seen that the performance exceeds the state-of-the-art on both simulated data and in a real-world task, the significant detection of a known Gamma source.

5.2 Supervision based on Noisy Labels with Partially-Known Class-Conditional Label Noise

In this section, we explore the potential of weak supervision for *gamma hadron classification* [22, 35], one of the three central machine learning tasks in gamma ray astronomy. By pointing the telescope at a known gamma source, we record unlabeled real-world data and use the estimated origin of recorded particles as a source of supervision. Formally, we treat the origin of gamma and hadron particles as a noisy weak label.

Discussing our approach within the theoretical framework of noisy labels even reveals a more general contribution of our work: unlike existing approaches in noisy label learning [84, 183, 235], we do not require knowledge about both the class-wise noise rates, but only one. We design a learning criterion based on the significance of detection and show how it tackles a noisy label learning problem for binary classification where exactly one of the noise probabilities is known. We call this setting partially known class-conditional label noise. We address the partial knowledge of

class-conditional label noise through an objective function that a learning algorithm can optimize.

Recall from Chapter 3 that $\hat{y} = \pm 1$ denotes a noisy label and $y = \pm 1$ is its corresponding clean label. Furthermore, the noise process is defined by two probabilities p_+, p_- . We define $p_+ := P(\hat{y} = -1 \mid y = +1)$ as the probability of flipping a clean positive, and $p_- := P(\hat{y} = +1 \mid y = -1)$ for the probability of flipping a clean negative label. Learning under this noise model is feasible iff. $p_+ + p_- < 1$. In Chapter 3 we have discussed related work on noisy label learning with either fully unknown noise rates or fully known noise rates. We present the first method for settings where only one noise-rate is unknown.

For the rest of this chapter, we use subscripts p_y with $y \in \{+1, -1\}$ or $N_{\hat{y}}$ with $\hat{y} \in \{+1, -1\}$ to compactly represent $p_+, p_-, N_+,$ and N_- .

5.2.1 The Significance of Detection as Noisy Label Learning with Partially Known Label Noise

We propose a novel method for learning under class conditional label noise with one known noise rate. The method is inspired by the gamma hadron separation problem that arises during source detection, a situation where we will see that indeed exactly one noise rate is known beforehand. Our method is based on the Li&Ma significance of detection score of Definition 2 that we will repurpose for general binary classification tasks with partially known label noise. In particular, we assume, without loss of generality, that the class-conditional noise probability p_- is known. We begin by first redefining the score more generally in terms of noisy labels rather than on- and off-regions, although the correspondence is one-to-one.

Definition 3 (Li & Ma Detection Score). Let $N_{\hat{y}}$ be the number of true positives with a noisy label $\hat{y} \in \{+, -\}$ and let $\alpha = \frac{p_-}{1-p_-}$. We assume that $N = N_+ + N_-$ follows a Poisson distribution. The Li&Ma detection score is defined as

$$f_\alpha(N_+, N_-) = \left[2N_+ \cdot \ln \left(\frac{1 + \alpha}{\alpha} \cdot \frac{N_+}{N_+ + N_-} \right) + 2N_- \cdot \ln \left((1 + \alpha) \cdot \frac{N_-}{N_+ + N_-} \right) \right]^{1/2}.$$

Remark 1. We assume that the number of true positives follows a Poisson distribution. This is plausible in a lot of situations, including gamma ray astronomy: Recall that Poisson distributions models counts of events that are recorded in a fixed, continuous interval of time if these events occur with a constant mean rate and independently of the time since the last event. The same data collection process can be found in many detection tasks in machine learning. Even in situations where N is not naturally described using a Poisson distribution, the test statistic can still be useful: For instance, when could assume that N follows a Binomial distribution, i.e. a distribution that models the number of events in a discrete interval. Then for large N , the Binomial distribution approaches a Poisson distribution.

Now we will show that we can use the Li&Ma statistic of Definition 3 to test the foundational assumption of noisy label learning with class-conditional label noise, i.e. that $p_+ + p_- < 1$. To do so, we formulate the null-hypothesis

$$h_0 : p_+ + p_- = 1. \quad (5.5)$$

We do not need to consider the case $p_+ + p_- > 1$ as flipping the roles of the noisy labels fulfils the assumption with $(1 - p_+) + (1 - p_-) = 2 - (p_+ + p_-) < 1$.

By showing how this null-hypothesis can be reformulated as a hypothesis that compares two Poisson distributions, we can use known results that tell us when we can reject this reformulated null-hypothesis.

To do so, we first prove a technical lemma that establishes that the number of true positives with a particular noisy label is Poisson-distributed if the number of true positives is Poisson-distributed.

Lemma 1. *Let λ be the rate of the Poisson distribution $\mathcal{P}(\cdot | \lambda)$ that models the number of true positive examples N . Then the number of true positives with a noisy label $\hat{y} = +1$ and $\hat{y} = -1$ are Poisson distributed with rates $(1 - p_+)\lambda$ and $p_+\lambda$, respectively.*

Proof. We prove the statement for $\hat{y} = +1$, the proof for $\hat{y} = -1$ is analogous. Let $\hat{y}_1, \dots, \hat{y}_N$ be random variables that contain the noisy labels of the true positive examples y_1, \dots, y_N . Then the probability

$$\begin{aligned} \mathbb{P}\left(\sum_{k=1}^N \mathbb{1}[\hat{y}_k = +1] = N_+\right) &= \sum_{l=N_+}^{\infty} \mathbb{P}\left((N = l) \cap \left(\sum_{k=1}^l \mathbb{1}[\hat{y}_k = +1] = N_+\right)\right) \\ &= \sum_{l=N_+}^{\infty} \mathcal{P}(N = l | \lambda) \mathbb{P}\left(\sum_{k=1}^l \mathbb{1}[\hat{y}_k = +1] = N_+\right) \\ &= \sum_{l=N_+}^{\infty} e^{-\lambda} \frac{\lambda^l}{l!} \binom{l}{N_+} (1 - p_+)^{N_+} p_+^{l-N_+} \\ &= \frac{e^{-\lambda} (1 - p_+)^{N_+} \lambda^{N_+}}{N_+!} \sum_{l=N_+}^{\infty} \frac{\lambda^{l-N_+} p_+^{l-N_+}}{(l - N_+)!} \\ &= \frac{e^{-\lambda(1-p_+)} (\lambda(1-p_+))^{N_+}}{N_+!} \\ &= \mathcal{P}(N_+ | (1 - p_+)\lambda) \end{aligned}$$

which is exactly the density function of a Poisson random variable with rate $(1 - p_+)\lambda$. \square

Now we prove that our initial null-hypothesis can be reformulated as a comparison of Poisson rates.

Lemma 2. *Let λ be the rate of the Poisson distribution that models the number of true positive examples N . Let $\alpha = \frac{p_-}{1-p_-}$. Let $\lambda_+ = (1 - p_+)\lambda$ and $\lambda_- = p_+\lambda$.*

Then

$$p_+ + p_- = 1 \quad \Leftrightarrow \quad \lambda_+ - \alpha\lambda_- = 0.$$

Proof. Algebraic massaging yields the desired result:

$$\begin{aligned} p_+ + p_- &= 1 \\ \Leftrightarrow p_+p_- &= 1 - p_- - p_+ + p_+p_- \\ &= (1 - p_-)(1 - p_+) \\ \Leftrightarrow p_+\frac{p_-}{1 - p_-} &= 1 - p_+ \\ \Leftrightarrow \alpha p_+\lambda &= (1 - p_+)\lambda \\ \Leftrightarrow \alpha\lambda_- &= \lambda_+ \\ \Leftrightarrow \lambda_+ - \alpha\lambda_- &= 0. \end{aligned}$$

□

Lemmas 1 and 2 yield that our initial hypothesis can be rearranged into a hypothesis that contrasts two Poisson distributions with rates λ_+ and λ_- . Now we show that the Li&Ma statistic is the statistic of a hypothesis test that rejects the null hypothesis that noisy label learning with class conditional label noise is infeasible.

Theorem 4 (Li&Ma Hypothesis Test). *Let $N_{\hat{y}}$ be the number of true positives with a noisy label $\hat{y} \in \{+, -\}$ and let $\alpha = \frac{p_-}{1 - p_-}$. We assume that $N = N_+ + N_-$ follows a Poisson distribution. Let $\mathcal{N} : \mathbb{R} \rightarrow [0, 1]$ be the cumulative density function of the standard normal distribution. The p value for rejecting $h_0 : p_+ + p_- \geq 1$ is given by*

$$p = 1 - \mathcal{N}(f_\alpha(N_+, N_-)) \quad (5.6)$$

where $f_\alpha(N_+, N_-)$ is the Li&Ma statistic stated in Definition 3.

Proof. Using Lemma 2, we can reformulate the null-hypothesis as $\lambda_+ - \alpha\lambda_- = 0$. Recall from Lemma 1 that N_+ and N_- follow a Poisson distribution. Based on their respective Poisson rates λ_- and λ_+ , we employ the likelihood ratio test as first proposed in Li and Ma [151, Eq. 17]. We present their proof in the our setting of noisy label learning here.

The data used as evidence in the statistical hypothesis test are the counts N_+ and N_- and we employ two parameters β and γ with $\lambda_+ = \beta + \gamma$ and $\lambda_- = \gamma/\alpha$ for modeling the likelihood of the measured data

$$\mathcal{P}(N_+, N_- | \beta, \gamma) = \mathcal{P}(N_+ | \beta + \gamma) \times \mathcal{P}(N_- | \gamma/\alpha).$$

Hence β models the excess counts $N_+ - \alpha N_-$ and γ models αN_- , consequently we can express the null hypothesis as $h_0 : \beta = 0$. In order to apply the likelihood ratio test, we have to compute the maximum likelihood estimates of β and γ given (N_+, N_-) . Starting with the log likelihood

$$L(N_+, N_- | \beta, \gamma) = N_+ \ln(\beta + \gamma) - (\beta + \gamma) - \ln N_+!$$

$$+ N_-(\ln \gamma - \ln \alpha) - \gamma/\alpha - \ln N_-!$$

we find the maximum likelihood estimates at the root of the partial derivatives

$$\begin{aligned}\frac{\partial}{\partial \beta} L(N_+, N_- | \beta, \gamma) &= \frac{N_+}{\beta + \gamma} - 1 = 0 \\ \frac{\partial}{\partial \gamma} L(N_+, N_- | \beta, \gamma) &= \frac{N_+}{\beta + \gamma} - 1 + \frac{N_-}{\gamma} - \frac{1}{\alpha} = 0\end{aligned}$$

which is at $\hat{\beta} = N_+ - \alpha N_-$ and $\hat{\gamma} = \alpha N_-$.

Furthermore we need a maximum likelihood estimate of γ conditioned on the null hypothesis $\beta = 0$. We again find the root of the partial derivative of the log likelihood

$$\begin{aligned}\frac{\partial}{\partial \gamma} L(N_+, N_- | \beta = 0, \gamma) &= \frac{N_+ + N_-}{\gamma} - (1 + 1/\alpha) = 0 \\ \Rightarrow \gamma^{(0)} &= \frac{N_+ + N_-}{1 + 1/\alpha} = \frac{\alpha}{1 + \alpha}(N_+ + N_-).\end{aligned}$$

Now the maximum likelihood ratio

$$\begin{aligned}\lambda &= \frac{L(N_+, N_- | \beta = 0, \gamma = \gamma^{(0)})}{L(N_+, N_- | \beta = \hat{\beta}, \gamma = \hat{\gamma})} \\ &= \left[\frac{\alpha}{1 + \alpha} \frac{N_+ + N_-}{N_+} \right]^{N_+} \left[\frac{1}{1 + \alpha} \frac{N_+ + N_-}{N_-} \right]^{N_-}\end{aligned}$$

can be used to reject the null hypothesis in a likelihood ratio test.

Standard results on likelihood ratio tests (eg. [244, Theorem 7.2.2]) dictate, that

$$\sqrt{-2 \ln \lambda} \sim \chi(1)$$

if the null hypothesis is true. We can take

$$S = \sqrt{-2 \ln \lambda} = f_\alpha(N_+, N_-)$$

as the significance value with corresponding p -value $1 - \mathcal{N}(S | \mu = 0, \sigma = 1)$ for rejecting the null hypothesis. \square

Remark 2. A similar hypothesis test can be derived if we assume that N follows a Binomial distribution. Then analogously, N_+ and N_- also follow Binomial distributions. Unfortunately, the resulting test statistic does not have a nice, compact form. Still it has a closed-form that we can compute and our experiments found that the Poisson variant is more useful but that the difference is slim.

In the next sections we will use this hypothesis test to find classifiers by selecting the models that reject the null hypothesis as significantly as possible.

Decision Threshold Optimization

So far, the counts $N_{\hat{y}}$ from Definition 3 describe the numbers of *true* positives with a noisy label \hat{y} . Thus, they require a set of data that is both ground-truth labeled and noisily labeled. To lose this unrealistic requirement, we heuristically replace the $N_{\hat{y}}$ counts with counts of *predicted* positives. This replacement allows us to tune the decision threshold which leads to the predicted positives, so that the tuned threshold maximally aligns with the foundational assumption $p_+ + p_- < 1$.

More specifically, we compute the function f_α from Definition 3 over $N_{\hat{y}}^\theta$, the numbers of predicted positives according to a decision threshold θ . We choose θ such that f_α becomes maximal, i.e. for noisy labels $\hat{y} \in \{+1, -1\}$, a soft classifier $h : \mathcal{X} \rightarrow \mathbb{R}$, and a noisily labeled data set $\{(\vec{x}_i, \hat{y}_i) : 1 \leq i \leq N\}$, we choose

$$\theta^* = \arg \max_{\theta} f_\alpha(N_+^\theta, N_-^\theta), \quad (5.7)$$

where $N_{\hat{y}}^\theta = \sum_{i=1}^N \mathbb{1}_{\hat{y}h(\vec{x}_i) > \hat{y}\theta}$,

so that f_α becomes the objective function with which we optimize the threshold θ .

This optimization does not require any ground-truth labels; we only need to count the numbers of predicted positives in both noisy classes. Moreover, we require knowledge about the rate p_- , so that we can compute $\alpha = \frac{p_-}{1-p_-}$, but we do not need to know the p_+ rate. The maximization works with any soft classifier h , like SVMs, decision trees, deep neural networks, and many more.

We heuristically replace true positives $N_{\hat{y}}$ with predicted positives $N_{\hat{y}}^\theta$ in the hypothesis test. This maximization of f_α can lead to rejections that are overly optimistic and learning with label noise can falsely appear feasible. On the other hand, if the null hypothesis cannot be rejected despite the maximization of f_α , i.e. if not even the optimized predictions of the model fulfill the foundational noisy learning assumption, we may conclude that learning is infeasible given only noisy labels.

We summarize our threshold optimization technique in Algorithm 1. If we cannot establish learnability with the given noisy labels, we can throw an error.

Algorithm 1 Li&Ma decision threshold tuning.

Input: A scoring function $h : \mathcal{X} \rightarrow \mathbb{R}$, a desired p value $p > 0$, a noise rate $0 < p_- < 1$, and N noisily labeled instances $\{(\vec{x}_i, \hat{y}_i) : 1 \leq i \leq N\}$.

Output: A decision threshold $\theta \in \mathbb{R}$.

- 1: Let $\alpha = \frac{p_-}{(1-p_-)}$
 - 2: Optimize θ^* according to Eq. 5.7
 - 3: **if** $p > 1 - \mathcal{N}(f_\alpha(N_+^{\theta^*}, N_-^{\theta^*}))$ **then**
 - 4: **return** θ^*
 - 5: **else**
 - 6: **return failure:** could not establish that learning is feasible
 - 7: **end if**
-

Decision Tree Induction

The threshold tuning from Algorithm 1 is particularly advantageous if some soft classifier h already exists. If not, we can alternatively maximize f_α over the entire model, not only over the decision threshold θ .

To this end, we propose a decision tree induction algorithm which learns by maximizing f_α directly, i.e. in every split. Like classic decision trees, our algorithm recursively partitions the (noisy) training set in a greedy fashion: at each node, we split the data by thresholding a single feature. To find the best split, evaluate f_α for both sides of the partition. We stop at a user-defined maximum tree depth or when no split can further improve f_α . Namely, each iteration of our tree induction algorithm selects a feature j and a feature threshold t , such that

$$\begin{aligned}
 j^*, t^* &= \arg \max_{j,t} \max \left\{ f_\alpha^{X_j \leq t}, f_\alpha^{X_j > t} \right\}, \\
 \text{where } f_\alpha^{X_j \leq t} &= f_\alpha \left(N_+^{X_j \leq t}, N_-^{X_j \leq t} \right), \\
 f_\alpha^{X_j > t} &= f_\alpha \left(N_+^{X_j > t}, N_-^{X_j > t} \right), \\
 N_{\hat{y}}^{X_j \leq t} &= \sum_{i=1}^N \mathbb{1}_{\vec{x}_{ij} \leq t \wedge \hat{y}_i = \hat{y}}, \\
 N_{\hat{y}}^{X_j > t} &= \sum_{i=1}^N \mathbb{1}_{\vec{x}_{ij} > t \wedge \hat{y}_i = \hat{y}},
 \end{aligned} \tag{5.8}$$

The best split according to Eq. 5.8 can be evaluated efficiently by sorting the data according to each of the features. Considering only the maximum of $f_\alpha^{X_j \leq t}$ and $f_\alpha^{X_j > t}$ amounts to the fact that a greedy maximization of f_α does not require balanced splits: if the other side of a split results in a low sigma value, we can either discard this side without harming the overall f_α or we can split this side further in a later step. As f_α increases with larger counts, the greedy approach is not tempted to split off individual examples. We have summarized our method in Algorithm 2.

Algorithm 2 Li&Ma decision tree induction.

Input: A maximum depth $d \geq 0$, a noise rate $0 < p_- < 1$, and N noisily labeled instances $\{(\vec{x}_i, \hat{y}_i) : 1 \leq i \leq N\}$.

Output: A trained decision tree.

- 1: Let $\alpha = \frac{p_-}{1-p_-}$.
 - 2: **if** $d > 0$ **then**
 - 3: Find j^*, t^* according to Eq. 5.8.
 - 4: Split $L = \{\vec{x}_i : \vec{x}_{ij^*} \leq t^*\}$, $R = \{\vec{x}_i : \vec{x}_{ij^*} > t^*\}$
 - 5: Construct sub-trees on L and R with depth $d - 1$.
 - 6: **return** a tree with a split j^*, t^* and both sub-trees.
 - 7: **else**
 - 8: **return** a leaf node with the output $\frac{1}{N} \sum_{i=1}^N \mathbb{1}_{\hat{y}_i = +1}$.
 - 9: **end if**
-

Ensemble Methods Tree models often benefit from ensembling approaches, so we also consider them in this context. As with classic random forests [29], we choose a bagging approach [28] to build ensembles from Algorithm 2. Namely, we use a bootstrapped sample of the noisy instances for each tree and draw a random feature subset of size $\lfloor \sqrt{d} \rfloor$ for each split. We combine the predictions of all trees by averaging their classification outputs. Using the bagging approach, we can tune the final decision threshold of the ensemble with Algorithm 1 on out-of-bag, noisily labeled data. This avoids the use of a separate dataset for decision threshold tuning.

5.2.2 Experimental Results on Imbalanced Data Sets

We begin our experimental evaluation using data sets outside of gamma ray astronomy. Specifically, we begin with a synthetic example, before investigating our approach on imbalanced datasets with random forest classifiers.

An Illustrative Example

We begin our experimental evaluation with a synthetic classification problem with one-dimensional $\mathcal{X} = [0, 1]$. We summarize the problem in Figure 5.6: The conditional distributions given the clean label are summarized in Fig. 5.6a. For this illustrative example we have chosen the prior probability $P(Y = +1) = 1/4$. Hence we know the Bayes-optimal classifier that is shown in Fig 5.6c.

When we induce class-conditional label noise with $p_+ = 0.2$ and $p_- = 0.4$, we now observe the conditional distributions $P(X | \hat{Y})$ shown in Fig 5.6b. We immediately see that the classification problem will be more difficult to solve, because in contrast to the clean conditionals we observe significant overlaps of the peaks of the distributions for both noisy classes. This is also apparent in the Bayes-optimal classifier that outputs probability estimates closer to $1/2$. When we use this classifier to predict the clean labels Y using the decision threshold $1/2$, we produce wrong outputs in the red regions highlighted in Fig 5.6d. The probability of sampling x in this red region is 0.0224. But we can recover the perfect decision boundary by setting the decision threshold to $\theta^* = (1 - p_+ + p_-)/2 = 0.6$.

Without access to both p_+ and p_- , we use our Li&Ma threshold tuning procedure to estimate $\hat{\theta}$. Hence we need counts N_+ and N_- for a given threshold θ . For this simulation we use their expected values for an imaginative sample of size N

$$N_+ = N \cdot \mathbb{E}_{(x, \hat{y})} (\mathbb{1}[\hat{y} = +1] \mathbb{1}[P(\hat{y} = +1 | x) \geq \theta]) \quad (5.9)$$

$$N_- = N \cdot \mathbb{E}_{(x, \hat{y})} (\mathbb{1}[\hat{y} = -1] \mathbb{1}[P(\hat{y} = +1 | x) \geq \theta]) \quad (5.10)$$

which we can compute exactly given the conditional, prior and noise probabilities of our synthetic example. As we can see in Figure 5.6e, we find the best $\hat{\theta} = 0.5935$, which is very close to the optimal value of 0.6. Indeed, using this threshold yields a substantially smaller region with wrong classification with regard to the clean labels, as indicated by the very slim red region in Fig 5.6f. The probability for sampling x in

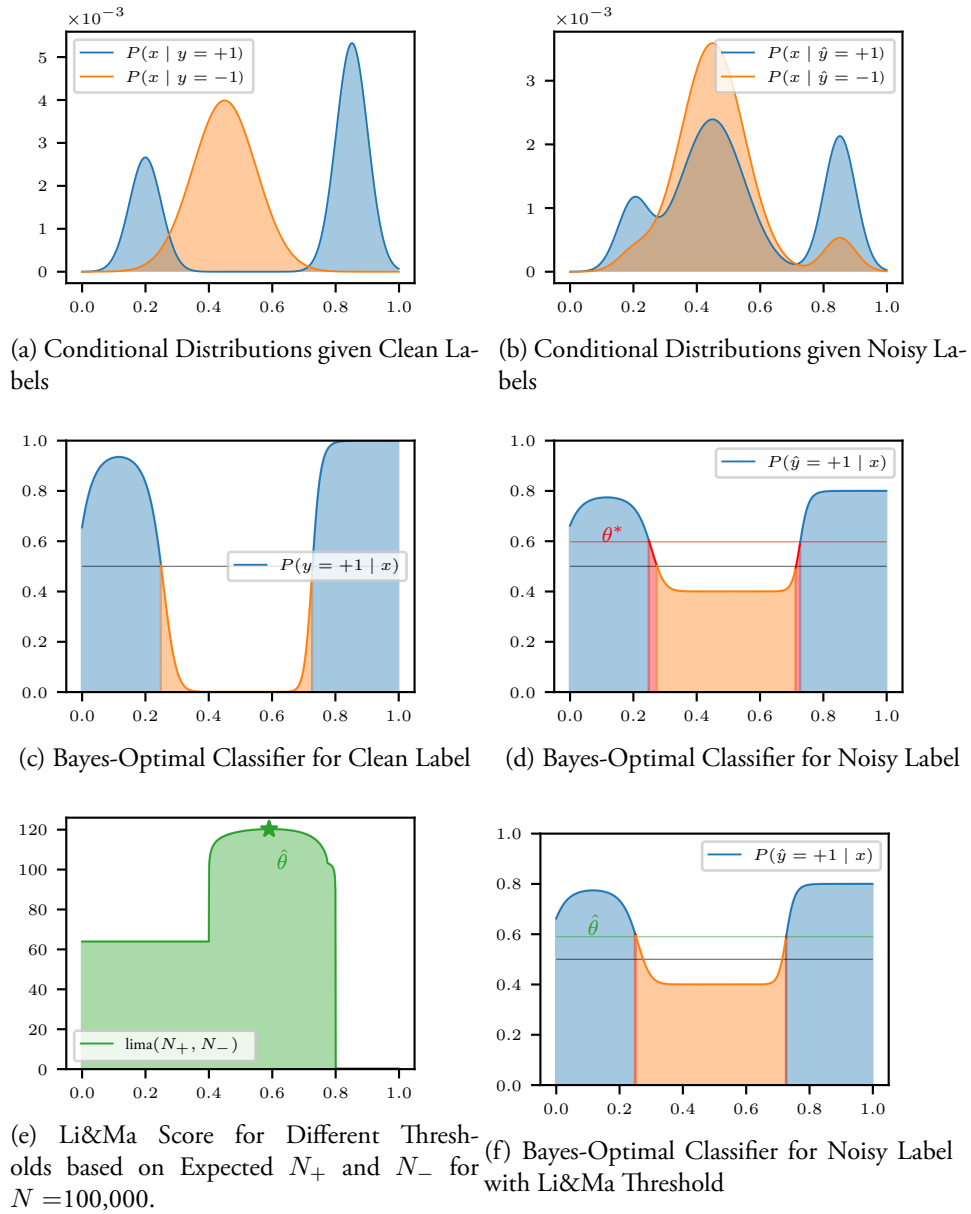


Figure 5.6: An Illustrative Example of Li&Ma Decision Threshold Tuning with $p_- = 0.4$, $p_+ = 0.2$ and $P(y = +1) = 0.25$. See Main-Text for a Detailed Review.

Table 5.4: Results of the Simulation Study for Different Noise Configurations. We observe that our estimated thresholds are always close to the optimal thresholds. In configurations where $p_+ \neq p_-$ this yields substantial reductions of the error probabilities. In configurations where $p_+ = p_-$, the optimal threshold is the naive threshold of 0.5, which cannot be improved by our method.

NOISE CONFIGURATION	OPTIMAL THRESHOLD	LI&MA THRESHOLD	NAIVE ERROR	LI&MA ERROR
$p_- = 0.5, p_+ = 0.1$	0.7000	0.7094	0.7454	0.0021
$p_- = 0.5, p_+ = 0.25$	0.6250	0.6197	0.7454	0.0018
$p_- = 0.2, p_+ = 0.2$	0.5000	0.4682	0.0000	0.0046
$p_- = 0.4, p_+ = 0.4$	0.5000	0.4917	0.0000	0.0036
$p_- = 0.4, p_+ = 0.2$	0.6000	0.5737	0.0224	0.0014
$p_- = 0.1, p_+ = 0.3$	0.4000	0.3324	0.0163	0.0096
$p_- = 0.3, p_+ = 0.1$	0.6000	0.6005	0.0144	0.0001

the red region is reduced to 0.0014. We can only speculate why our method cannot recover the perfect decision boundary. A likely explanation is the violated assumption that the number of true positives follows a Poisson distribution, as in our example it follows a Binomial distribution with $p = 1/4$.

We can repeat this experiment for different noise configurations. In Table 5.4 we summarize the results obtained. For symmetric noise configurations $p_+ = p_-$, the optimal decision thresholds $\theta^* = 0.5$ is the naive threshold, so any other threshold obtained by a tuning procedure cannot be better. Still our approach can recover it almost perfectly. For asymmetric noise configurations we can report great decreases of the probability of sampling x in a region where the Bayes-optimal classifier for the clean labels disagrees with the output of the thresholded classifier for the noisy labels.

Results on Imbalanced Datasets

Our next evaluation is based on imbalanced data sets¹ from the *imbalanced-learn* library [148], where label noise is particularly harmful. We artificially inject different levels of class-conditional label noise, which are listed in Fig. 5.7 and in Table 5.5. The first two noise levels are designed by us and the remaining four were proposed by Natarajan et al. [183]. We limit our evaluation to those 17 data sets that have at least 100 instances of the minority class.

We estimate the performance of each method in terms of the F1 score, a metric that is well suited for imbalanced data. This score is averaged over 20 repetitions of a 10-fold stratified cross-validation, where we observe quite small standard deviations among all repetitions. We do not validate in terms of the area under the ROC curve

¹<https://imbalanced-learn.org/stable/datasets/>

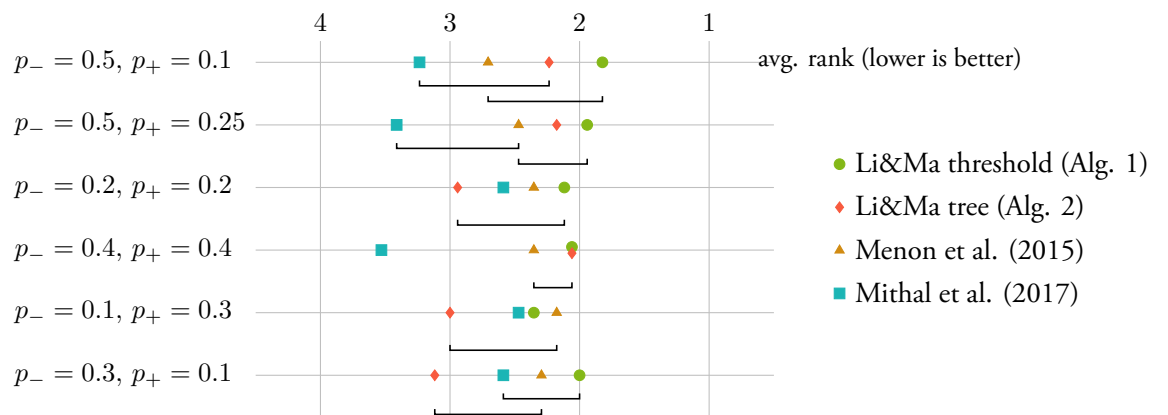


Figure 5.7: Each row displays one critical difference diagram [53] which corresponds to one noise configuration. This overview summarizes a total of 81,600 random forest classifiers, which consist of 4 binary learning methods for noisy labels \times 6 noise configurations \times 17 data sets \times 10 cross validation folds \times 20 repetitions with random initialization. Methods are connected with horizontal brackets if and only if a Holm-corrected Wilcoxon signed-rank test cannot significantly distinguish their pairwise performances, in terms of their cross-validated F1 score, at a confidence level of 0.90.

Table 5.5: Average F1 scores (higher is better) over 17 data sets and 20 repetitions of a 10-fold cross-validation with different label noise configurations. A full version is in the appendix.

NOISE CONFIGURATION	LI&MA THRESHOLD	LI&MA TREE	MENON THRESHOLD	MITHAL THRESHOLD
$p_- = 0.5, p_+ = 0.1$	0.5436	0.5475	0.4795	0.4578
$p_- = 0.5, p_+ = 0.25$	0.4302	0.4372	0.3817	0.2842
$p_- = 0.2, p_+ = 0.2$	0.6006	0.5737	0.5838	0.5778
$p_- = 0.4, p_+ = 0.4$	0.3661	0.3884	0.2961	0.2500
$p_- = 0.1, p_+ = 0.3$	0.6164	0.5899	0.6140	0.6053
$p_- = 0.3, p_+ = 0.1$	0.5970	0.5611	0.5775	0.5795
OVERALL AVERAGE	0.5256	0.5163	0.4888	0.4591

because this metric is anyway optimal under class-conditional label noise [167] and therefore not informative. We do also not validate in terms of accuracy because this metric is usually meaningless under severe class imbalance.

Figure 5.7 compares the performances of the noisy learning methods in Critical Difference Diagrams [53]. These diagrams plot the average ranks of all methods across the 17 data sets and connect those methods that a Holm-corrected Wilcoxon signed-rank test cannot significantly distinguish. The average ranks (lower is better) tell which method frequently wins against the others, while all *missing* connections indicate significant differences between the competitors.

Additionally, Table 5.5 reports the average F1 scores across all data sets and repetitions for each of the methods. The F1 scores for every individual data set, noise configuration and method can be found in the appendix, see Table 13.1 and Table 13.2.

Combining the views from Figure 5.7 and Table 5.5, we find that our thresholding approach from Algorithm 1 has the best average rank across all noise configurations, as well as the best average F1 score. Furthermore, it is always located in the best performing group of methods, according to the Wilcoxon signed-rank tests. Accounting for the partial knowledge of the noise rates thus gives an advantage over existing methods that have to estimate both rates. Our decision tree approach from Algorithm 2 is in the best group of most noise configurations, but generally performs worse. Therefore, we generally recommend Algorithm 1 for imbalanced, partially-known class-conditional label noise problems.

5.2.3 Gamma-Hadron Classification with Noisy Labels

Now we benchmark our approach on the gamma hadron classification task that motivated the method. We begin by reviewing the source detection task already presented in Section 5.1.3.

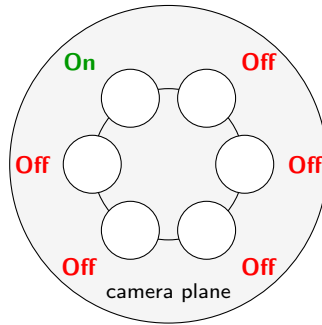


Figure 5.8: An Illustration of the Wobble mode for capturing on- and off-regions simultaneously. Figure taken from [206].

In astrophysics, gamma ray sources are detected by measuring rays emitted from the direction of the suspected source. This signal is contrasted with background measurements obtained by recording from a direction where no source is suspected. A detection is successful, if the signal significantly exceeds the background, which e.g. motivated Definition 2. Li and Ma [151] have originally defined the scaling factor α as the fraction $t_{\text{on}}/t_{\text{off}}$ of observation times. Here, t_{on} measures the time used to observe the source and t_{off} measures the time used to observe the background in an area where no source is assumed. These times are known by design; astronomers can freely choose how much time they assign to the observation of different positions in the sky.

In the Wobble observation mode, the region of interest and 5 background regions are recorded simultaneously for the same observation time. We illustrate this operation mode in Figure 5.8. The region of interest, the *on-region*, is placed on a ring around the camera center. It measures the rate of gamma rays that the suspected source emits. The additional *off-regions* on the same ring simultaneously measure the background rate, provided that no other gamma ray source falls into any of them. The FACT telescope, for instance, places six regions on a 0.6° ring inside its 4.5° field of view. Every 20 minutes, another region on the camera pane is used as on-region, which causes the telescope to *wobble* from region to region while observing a fixed position in the sky.

We want to tackle the gamma hadron problem using the estimated origin of particles as weak labels. We have $\hat{y} = +1$ if the particle was estimated to originate from the on-region and $\hat{y} = -1$ if it falls into one of the off-regions. Intuitively, we expect more gamma particles with $y = +1$ in the on-regions, so learning should be feasible. Astronomers can make an additional assumption: hadronic particles are expected to occur uniformly, i.e. at a constant rate, across all sky positions. Consequently p_- , the probability of recording a hadronic particle in the on-region, must exclusively depend on the observation times t_{on} and t_{off} . More specifically, it is known to be $p_- = \frac{t_{\text{on}}}{t_{\text{on}} + t_{\text{off}}}$. In the FACT experiment, operating in Wobble mode as described above, the background is recorded 5 times as long, which yields $p_- = 1/6$ and $\alpha = p_- / (1 - p_-) = 0.2$.

So we are in the setting of noisy label learning with one known noise rate.

Dataset and Classification Models We begin our analysis using decision tree and random forest classifiers on the hand-crafted features described at the beginning of this chapter.

We again use the open data sample of the FACT collaboration [196]. This data contains five nights of observations in wobble mode, totalling 757,993 recorded events, where the telescope places the Crab nebula, a bright supernova remnant, in the on-region. Each of these events is characterized by 22 features. We expect that the vast majority of events are hadronic particles and that only around 0.1% to 1% are the gamma events we are interested in. Of all events, 4,322 fall into the on-region $\hat{y} = +1$ and 17,229 fall into off-regions with $\hat{y} = -1$. The estimation of origin that these noisy labels are based on is performed using the current SOTA random forest approach and are part of the available dataset.

We again compare our methods, Algorithm 1 and Algorithm 2 to the methods of Menon [167] and Mithal [173]. We use the same models for all decision threshold tuning approaches and use the same hyperparameters for inducing decision trees with Algorithm 2 as we use in the construction of the standard classification trees.

We test a number of different hyperparameter settings per model and repeat each setup ten times to report average values and standard deviations.

Performance We use a cross-validation protocol to estimate the significance of detection using the noisy-label approaches: We train and predict on disjoint splits of the data, as usual. However, we store the predictions of each holdout set, rather than computing a score in each fold. This way, we obtain predictions for all instances in the data, but each prediction is based on a model that has not seen the instance during training. We then report the values of the significance of detection (Def. 2) that we obtain with the full data set. For reliable results, the training and hold-out sets of the data should be as independent as possible. Since recording conditions, like the brightness of the night sky, change during the course of recording, we group the data based on an attribute that corresponds to the time of the day. We found that this grouping yields the lowest values of significance, giving us the most cautious estimates of model performances. Additionally, we compare to the state-of-the-art supervised models that are trained on simulated data with the same 22 features. By the design of the cross-validation protocol, we ensure that the measurements of fully-supervised and noisy-label models are comparable, as the significance grows with larger numbers of examples, which is now the same.

The average values and standard deviations across all repetitions are displayed in Table 5.6. We see that all of the approaches for class-conditional label noise, if they employ random forests, perform as good as the traditional approach that is trained with costly, simulated data. Some of the methods even slightly outperform this strongly supervised baseline. Apparently, the potential disadvantage of noisy labels being weaker than simulated clean labels is compensated by the fact that the noisily labeled data, coming from the actual telescope, is more realistic than the simulated state-of-the-art

Table 5.6: f_α outcomes (see Definition 2, higher is better) for the open Crab Nebula data of the FACT telescope.

METHOD	DECISION TREE	RANDOM FOREST
LI&MA TREE (ALG. 2)		
MAX_DEPTH=4	24.059±0.00	23.302±0.332
MAX_DEPTH=8	20.623±0.00	24.737±0.240
LI&MA THRESHOLD (ALG. 1)		
MAX_DEPTH=4	24.290±0.00	25.300±0.108
MAX_DEPTH=8	20.920±0.00	26.555±0.249
MENON THRESHOLD [167]		
MAX_DEPTH=4	24.290±0.00	25.131±0.218
MAX_DEPTH=7	18.232±0.00	26.197±0.289
MITHAL THRESHOLD [173]		
MAX_DEPTH=4	24.582±0.00	25.279±0.223
MAX_DEPTH=8	20.534±0.00	26.389±0.180
SIMULATED CLEAN LABELS (SOTA)		
DT / RF	24.650±0.000	26.254±0.151

training data. This remarkable result could shape the way in which future telescope projects approach their analysis pipelines: it demonstrates that we can benefit from real recordings for training, not only as background events, as it has been done before [2], but also for learning about the gamma class. Gamma hadron classification could become even more data driven than today and less sensitive to imperfections and biases of the simulation.

Once we decided to use noisy labels, the choice of the particular method for tuning the decision threshold is only a secondary concern: All methods considered yield high significance scores, although our approach using domain knowledge on the noise rate p_- slightly outperforms the other contestants.

Interpretability To validate the claim that Definition 3 is interpretable as a hypothesis test about learnability under class-conditional label noise, we conduct another experiment in which we artificially remove all on-labeled instances from the data set. We then incorrectly label one of the off-regions as being the on-region. These fake re-assignments are meant to break the correspondence between clean and noisy labels and should therefore render learning infeasible. As desired, Table 5.7 demonstrates that our proposed algorithm methods indeed produce f_α values close to zero, making our algorithm output an error message. Practitioners can tell from these values that no strong classifier can be learned from the given noisy labels. The fact that our algorithms intend to maximize f_α does not result in a false outcome of the hypothesis test.

Table 5.7: f_α outcomes (see Def. 2) for the FACT data with artificially substituted on-instances.

method	decision tree	random forest
Li&Ma tree (Alg. 2)	0.0000 ± 0.0000	0.0435 ± 0.0746
Li&Ma threshold (Alg. 1)	0.0000 ± 0.0000	0.1788 ± 0.2909

Table 5.8: Significance of Detection when our noisy learning models are transferred to different sources.

Target Source	Observation Time	Alg. 1 max_depth=8, $T=100$	Super- vised
Crab Nebula	Oct '13	30.8710 ± 0.3611	30.381
Mrk 421	Oct '13 – Feb '14	23.2386 ± 0.3068	23.278
Mrk 501	Jul '13 – Dec '14	27.3412 ± 0.1490	26.330

Transfer to Different Sources Next we test if the models trained on the available Crab nebula sample are useful classifiers in the detection of other sources. To this end, we use three other datasets of recorded events: Two where the telescope was pointed at other known sources Mrk421 and Mrk501, and one where the Crab nebula was recorded on different days. These samples are used exclusively for testing, training was performed on the dataset described above. This way we test if there is overfitting to the particular source in the training data. As we can see in Table 5.8, the performance of our models trained with noisy labels derived from the estimated origin of particles is on-par with the performance of the current state-of-the art classifier trained on simulation data. This is first evidence that the classifiers trained are indeed useful for the analysis of gamma ray sources and even the detection of new gamma sources.

5.2.4 Deep Gamma-Hadron Classification with Noisy Labels

We combine the last two Sections and directly train Gamma-Hadron separators from raw image data using noisy labels and repeat the experiments done with random forest classifiers on handcrafted features for deep networks trained on the photon count image data.

We consider two approaches, deriving a differentiable Li&Ma loss function and using a standard loss plus the threshold tuning algorithm, and quickly settle for the latter.

Soft-Li&Ma As commonly done in deep learning literature, we replace the indicator function in the original Li&Ma formulation with sigmoid functions $\sigma(\cdot)$. This way we obtain the differentiable loss function that computes the loss given a vector of noisy

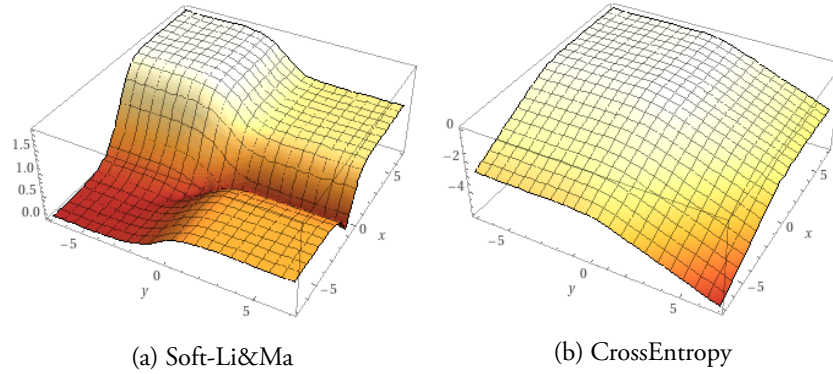


Figure 5.9: The loss surface for the Soft-Li&Ma and crossentropy objective for a mini-batch of size 2, where x, y are the model outputs for two examples with noisy label $+1$ and -1 respectively. We see that there is a bad local maximum that is separated from the global maxima by a deep valley. In contrast, the cross-entropy loss has only one global maxima that is easily accessible.

labels \hat{Y} and a vector of predictions $f(X)$ and computes

$$\ell_{\text{soft-lima}}(\hat{Y}, f(X)) = f_{\alpha}(S_+, S_-) \quad (5.11)$$

$$S_+ = \sum_{i=1}^N \mathbb{1}[\hat{y}_i = +1] \sigma(f(x_i)) \quad (5.12)$$

$$S_- = \sum_{i=1}^N \mathbb{1}[\hat{y}_i = -1] \sigma(-f(x_i)) \quad (5.13)$$

Unfortunately, learning models that maximize the Soft-Li&Ma objective using stochastic gradient optimization methods turns out to be a challenging task. We can see why in Figure 5.9a: The objective function has spurious local maxima that are separated by valleys around the axis. Crossing these valleys requires taking a step in the gradient direction while SGD takes optimization steps in the opposite gradient direction. We quickly abandon this idea in favor of the two-step procedure proposed earlier.

Noisy Classification + Threshold-Tuning We use standard cross entropy loss, see Fig 5.9b, and fit our model to the noisy labels. Then we apply the Li&Ma threshold tuning approach of Alg. 1. In the two-dimensional example, we see no spurious local extrema in the loss surface. As the telescope records the off-region 5 times longer, we weight the loss function inversely to reduce the models bias towards predicting the negative class.

Deep Learning requires large amounts of data for fitting large numbers of parameters. However, only a limited amount of training data is available. While the telescope records data in large volumes, the number of events that originate in the on- and off-regions is still rather small. For instance, using the `efficientnet-b0` model from

Table 5.9: Significance of Detection Scores (Cross-Validated) for `efficientnet-b0` networks. We see that the noisy label approach that retrains the classification head with noisy labels and the one that finetunes the classification head outperform the previous classification head based on full supervision with simulation data.

Training Paradigm	Li&Ma Significance
Full	20.03 ± 1.72
Head	28.13 ± 0.73
Finetune	28.50 ± 0.32
Simulation-Based	26.85 ± 0.57

the first Section yields a subset of only about 25,000 examples that can be used for training in the noisy label approach. To account for this, we propose three different training paradigms:

- **Full Training:** We train a deep network with randomly initialized weights that tackles just the gamma-hadron separation task. This model would have to be used in parallel with the model from the first section that predicts the origin and energy of observations.
- **Head Training:** We take the model of the first section trained on simulation data and only re-initialize the parameters of the prediction head responsible for gamma-hadron separation. We freeze all weights in the shared encoder and train only the prediction head using the noisy labels.
- **Finetune Training:** We also take the model trained on simulation data and freeze the weights of the shared encoder. However, we do not reinitialize the weights of the gamma-hadron prediction head. Instead we continue to train its weights using the noisily-labeled data.

Empirical Results

We limit our empirical study to the smallest `efficientnet-b0` architecture. We run a hyperparameter search and optimize the hyperparameters for each training paradigm separately. As we can see in Table 5.9, we can outperform the models trained with full supervision on simulated data. Indeed the models where the prediction head was retrained or finetuned outperform the supervised models substantially. Only the model trained from scratch is not competitive. This is most likely due to the limited amount of training data available. Recall that the model has 13.5 million parameters while the data set used has only about 25,000 examples.

The hyperparameter search revealed that the best results for the finetuning approach are obtained using only a single epoch of finetuning with a small learning rate. In contrast, tuning the head usually needs 10-20 epochs of training to obtain the best

results. This suggests that we do not need to change the model a lot to adapt from the simulation data to the noisily-labeled, real-world data.

5.3 Summary

In this Chapter, we have tackled the event-tagging problem that arises in modern gamma-ray astronomy through different variants of supervision. The problem involves three machine learning tasks: gamma-hadron separation, energy estimation and origin estimation. We have advanced the current state-of-the-art in two separate ways: First, using a multitask deep network, we were able to replace the current approach based on manual feature engineering and random forest models with a shared deep network that works on raw photon-count images. The training approach belongs to the family of supervised methods and we have combined three sources of full supervision to train a single, multi-task model. Our model outperforms the current models on both simulation data and on a real-world source detection task.

Second, we showed how the gamma-hadron task can benefit from noisy label learning, another form of supervision. To this end, we designed novel noisy-label learning algorithms inspired by the source-detection task. In contrast with existing approaches for class-conditional label noise, our approach assumes that exactly one noise rate is known, as it is the case in source detection for gamma ray astronomy. Empirical results show that both the current detection pipeline based on random forests, as well as the deep network pipeline proposed in this chapter benefit from this novel training paradigm, as we report increases in the significance of detection.

Part III

Inspecting Machine Learning Models

Chapter 6

Motivation



ONCE a model has been designed and trained through an appropriate choice of model family, learning task and data processing, and once it has been established that we expect good empirical performance through an appropriate choice of validation protocol, we have to decide if we trust the model enough to deploy it in its intended application.

Under the umbrella term *trustworthy machine learning*, many aspects of trust and methods of establishing trust have been investigated. In this work we focus on methods aimed at helping the data scientist establish trust in their models before deployment, rather than considering laypersons who interact with a deployed model. Hence we do not have to account for common misconceptions on AI systems [4].

Particularly for models with many parameters, like deep networks or random forests, we cannot just inspect the final, learned decision function and expect to understand its inner workings. But understanding these inner workings is a good pathway to building trust [11, 139]. We consider two different approaches for inspecting machine learning models: Providing explanations and checking the robustness of models.

6.1 Related Work

We review related work that targets model inspection, in particular approaches related to explanations and to robustness verification.

6.1.1 Explanations

One way of inspecting a model is inspecting its decisions. By investigating why a model makes decisions, we gain insights - and ultimately trust - in the model and its behavior. The explainable artificial intelligence (XAI) community has proposed a plethora of methods for providing explanations, i.e. answering why a decision was made [13, 225]. We provide a short overview of related work in that area. While work on *interpretable machine learning* focusses on learning models that can be understood as a whole, we focus on works that explain models that are too complex to be understood in their entirety. These methods can be divided into global and local methods, where

the former provide explanations to the whole model whereas the latter methods explain individual decisions.

Interpretable Proxy Models Interpretable models are useful for general explanation: We can train an interpretable model that approximates the output of a model that we want to explain to obtain a global explanation of our model (see e.g. [177, Chapter 8.6] or [228]). However, the discrepancy between the model and its interpretable surrogate can lead to unfaithful explanations. Furthermore, characterizing or measuring this discrepancy is difficult on finite datasets. In the last Chapter we see a case where a global surrogate model is used and the discrepancy is measured faithfully.

We can also generate explanations for individual decisions by generating local proxy models. Here the model is queried in the neighborhood of the point we want to explain. The resulting set of examples labeled by our model of interest is used to train an interpretable proxy model. This proxy model describes the local decision boundary around the point of interest and we can use it to provide an explanation for our model. Different methods that apply this idea have been proposed; they differ in the way a local neighborhood is generated and in the interpretable model used. The most prominent approach is LIME proposed by Ribeiro et al. [221]. They sample synthetic datapoints in the neighborhood and train linear models to obtain interpretable models. The linear coefficients can be used as explanations. In contrast, LORE [98] applies rule learning to generate local models and selects the neighborhood using a genetic algorithm.

Using interpretable proxy models is only feasible if the complexity of the decision boundary, locally or globally, is small enough for the class of interpretable models. For instance, this approach likely does not yield useful results for deep networks for high resolution image classification.

Feature Attribution Methods Attribution based methods try to identify the most important input dimensions for the model's decisions. Hence they provide explanations that superficially describe how the model computed its output: The complex decision function is reduced to a list of features that are important for the decision. How exactly these features influence the decision is excluded from the explanation. A simple algorithm iterates all features and drops each feature from the training data. We can measure the drop in prediction performance for the model without access to the particular feature and rank all features based on these performance differences: The features that cannot be omitted without a substantial decrease in performance are the most important features for a classifier. The disadvantage of this easy algorithm is that we have to train one model per input dimension. Instead, Breiman [29] proposes to measure the feature importance at test time on a holdout-set. Originally proposed for random forests, for each input feature we randomly permute all the feature values in the test set and measure the resulting drop in classification performance on the corrupted data. This randomization breaks the correspondence between the feature and the target variable. Larger drops in performance indicate that a feature is important for the decision. In the case of random forests or other bagging ensembles, we do

not need the separate test set, but can work with out-of-bag estimates of the performance [29]. Both of these attribution approaches provide global explanations to the model. Domain experts can use these feature importances to judge if it is plausible to make decisions based on the set of important features.

Other approaches estimate local feature importances for individual model decisions. The most prominent method is probably SHAP [156], which estimates feature importances for individual decisions based on the Shapley values (see e.g. [177]). For each feature, SHAP values measure the change in the expected model prediction when fixing that feature. For linear models, these contributions are a function of the weights [248]. For other models, the SHAP values have to be computed approximately [156].

Saliency Map Approaches Feature importance approaches do not easily scale to high-dimensional data like images, text or time series. Here individual features are often meaningless, e.g. a single pixel cannot explain the decision in an object detection task., but explanations must focus on a particular combination of features. For convolutional neural networks, this includes methods like saliency maps [245]. Building onto the deconvolution technique [245,294], Springenberg et al. propose a kind of guided backpropagation where negative gradients are set to zero in the backward step to focus solely on input features which increase the activation of a higher layer unit [246]. Another approach by Stylianou and Pless visualizes the output of a convolutional neural network using just the activations of the last convolutional layer to identify the regions of interest for the image classification [249]. Saliency map approaches have been useful for identifying models that reach decisions by exploiting spurious correlations or shortcuts [82], e.g. detecting a ship when there is water in the image [18, 152]. Saliency methods allow us to detect these shortcuts or so-called Clever-Hans behaviour [276] in image classifiers, in a second step we correct the model [234].

Counterfactual Explanations Counterfactual explanations are primarily tailored for generating explanations meant for the human affected by the current decision. Explanations are provided in form of the closest inputs that achieve the desired output [97]. This input can either be from the training data or a synthetic input. In the case of synthetic inputs, these have to be constrained to plausible values of the input space [138]. Taking plausibility one step further, we can also look into algorithmic recourse. Here, the *change* of the input has to be plausible, for instance something that a human can achieve through taking a feasible action [128].

Example-Based Explanations Another common technique is to use the training data for providing explanations. This is often named *example-based explanation*. In the simplest form, nearest-neighbor based classifiers provide explainable decision through the nearest neighbor examples [177]. Support vector machines have a similar property: Looking at the support vectors – all of them or only the most influential ones for a particular decision – provides interpretable models or explainable decisions [228].

Permutation based approaches are an easy way to measure the importance of individual training instances to the final model by dropping them from the training data and monitoring the change of the model. However they do not scale to large-scale deep learning problems. In contrast to feature attribution methods, this method works on examples rather than features. Influence Functions [135] provide an alternative. They investigate how the model changes when individual examples are reweighted in the empirical loss function. This can be approximated by investigating the Hessian matrix of the loss function around the final model. For interesting models, however, this Hessian matrix can only be approximated [135].

Evaluating Explanations Finally we should note that evaluating explanations is a non-trivial task on its own [23,184,303]. Particularly visual explanations and example-based explanations [187] are often subjective and defining quantitative metrics is difficult. Datasets where expert explanations are provided as an additional layer of annotation may allow quantitative evaluations. However, it is not always apparent that there is a single, unique explanation. Synthetic data can ensure this. For instance, Sanchez-Lengeling [230] provide a set of synthetic graph-classification tasks where labeled sets of nodes are responsible for the target variable. Tritscher et al. generate datasets of synthetic categorical features and use randomly generated boolean functions as labeling functions with 3 to 10 important features [267]. In both cases, XAI method should identify only truly relevant parts of the input as relevant. More recently, Tritscher et al. provide a more realistic dataset for fraud detection where domain experts have manually annotated the “problematic” features of simulated fraud cases [125]. Mücke and Pfahler use the known, deterministic rules for checking for check mate in chess games to derive a dataset with ground-truth explanations [180].

6.1.2 Robustness

Robustness measures describe how stable a model’s predictions are under small changes to the input. Intuitively, we want predictions to remain unchanged when we change the input insignificantly. Szegedy et al. demonstrated that deep networks fail catastrophically in this regard: Seemingly random noise added to input images, far from challenging to the human eye, can flip the prediction of a deep network [254]. Since this initial shock, the topic of robustness was addressed throughout the machine learning community.

Assessing a model’s robustness can increase trust in its behaviour at inference time [166]. For instance we can establish that the model behaves as desired not only at validation points, but also in their immediate neighborhoods. Further, by inspecting perturbations that do change the output, we can gain insights into the shape of the decision boundary.

Overall, related work that investigate robustness can be divided into three categories: attacks, defenses and verification.

Attacks Szegedy et al show that we can find perturbations with small l_2 -norm to a given test example x using simple first-order optimization starting at x [254]. They formalize the problem as searching the smallest possible perturbation, but settle for an approximation to find any locally-optimal small perturbation, due to the non-convex nature of the optimization problem. Few-pixel [175] and one-pixel attacks [250] constrain the adversarial perturbations using l_0 norm, i.e. the number of features affected. The authors show that we can fool deep image recognition models by modifying few or only one pixels for many network architectures, including the popular VGG convolution networks. Approaches that follow similar paradigms have been summarized under the umbrella-term *adversarial attacks* [114]. It has been shown that adversarial attacks on image classifiers are quite robust and even work after printing and photographing the targeted images [140]. Fawkes is a tool to modify photos such that deep network-based facial recognition software cannot recognize faces [241]. It is designed to be used whenever photos are uploaded onto social media to prevent third parties from training facial recognition systems on social media data. Here the constraints are that changes are imperceptible to the human eye. The authors note that this approach is in an "arms-race" with commercial facial recognition software.

Inspecting adversarial examples may provide insights into the behavior of the model.

Defenses We can try to train models that are robust to adversarial attacks, this is commonly known as *adversarial training*. The challenge is both increasing the adversarial robustness while maintaining the accuracy of the model. Goodfellow et al. propose to include a regularization term into the training objective that considers the loss of not only the original training example, but also of an adversarially perturbed example. They obtain this adversarial example using their fast gradient sign method, that simply takes a small step in the direction of the sign of the gradient of the loss with respect to the original example [93]. More generally, methods for training robust models solve a min-max optimization problem: We want to minimize the loss of the 'hardest' adversarial example. Obtaining the hard adversarial example is an inner loss-maximization problem, training or model to classify this adversarial example correctly is the outer minimization problem. Algorithmically, Madry et al. propose to alternate between obtaining a hard adversarial example through a number of stochastic optimization steps in the input space and an optimization step for the classification loss on this adversarial example [158]. Obviously, this algorithm introduces computational overhead in comparison to the standard non-robust training. Shahafi et al. [239] propose to simultaneously train the classifier and improve the adversarial examples instead of alternating the optimization. This way they achieve a substantial speed-up.

Croce et al. discover that the adversarial robustness of a ReLU network is related to the size of the linear regions and propose a regularizer that promotes large linear regions [50].

Verification For establishing trust in a model, robustness verification methods provide formal, provable guarantees that there are no adversarial perturbations within a neighborhood with certain properties.

Verification, in general, proves that a program, in our case a machine learning model, fulfils a certain property [66]. These properties could eg. be safety properties relating to autonomous systems [86]. Properties may either be global, i.e. they hold across the full data space, or local, i.e. they should hold around a specified data points. Girard-Satabin et al. point out that in order to prove global properties, the input space must be low-dimensional and the semantics of the features and their interaction must be well-understood. In problems that arise in perception or computer vision this will not be the case, which is why they employ a simulator [86]. Robustness-verification on the other hand is a local property: Given a constrained set around a focal point x , e.g. defined by an l_2 -ball, verify that there are no adversarial perturbations that change the model output within the set. Alternatively we can phrase this property check as an optimization problem: Find the largest set such there is provably no adversarial perturbation. Hence it is equal to counterfactual explanation, where we find the smallest perturbation as a form of explanation, as detailed above. In summary, while adversarial attacks prove the existence of adversarial perturbations by finding an example, verification proves their non-existence through formal methods. A multitude of different methods exist, tailored to specific model classes. An overview for decision tree and random forest verification is given in Chapter 10. As tree-based models with their if-else-structure closer resemble the input programs traditionally considered in formal verification in software engineering, many of the approaches are inspired from that research domain. In contrast, deep networks with their models that sequentially apply linear algebra operations and non-linearities, without the need for control-flow, constitute very different programs [87]. Still, approaches try to apply standard software verification tools to obtain formal guarantees [68, 86]. Other approaches work by exploiting properties of the function computed by deep network. For instance, networks that apply only ReLU activation compute piecewise-affine function; this structure can be exploited to derive robustness properties, as shown by Croce et al. [50] who exploit this insight to train more robust models through regularization. Abstract interpretation techniques, eg. Müller et al. and references therein [182], can handle a more general family of neural networks and are the current state-of-the-art for deep network verification: They approximate how a deep network transforms a given convex set, rather than a single point. If the set of inputs where we want the network to have robust outputs, intersects the the decision boundary, then the model is not robust. The difficulty lies in handling the non-linearities, which have to be approximated via increasingly tight constraints.

6.2 Structure of this Part

The remaining chapters of this part cover three different approaches for inspecting models with many parameters. We begin by discussing a saliency-map approach for graph-convolutional neural networks in Chapter 7. The method can be used to explain similarity computations based on representations computed by deep networks. We showcase this approach in a mathematical retrieval application.

Next, we present a method for providing explanations for deep networks in Chapter 8. In contrast to other approaches, it aims to answer why a model has learned a representation. Our approach works by tracking all weight updates during the model training and we propose two different ways to integrate this into existing training pipelines. We evaluate our approach for standard image classification tasks.

Finally, we present a method for verifying the robustness of random forest models and show how it can be used to gain insights into the decision boundary of large models.

Chapter 7

How Does the Model Compute?

LARGE machine learning models guide decisions in many high-stakes environments. Their decisions, however, are not understandable by inspecting the weights of the final model anymore. Instead, we try to understand individual decisions. One way to approach this explanation problem is by asking the question “How did the model compute the decision?”. Answers may include descriptions of what features or parts of the input were most important to the decision, as seen in the last Chapter. In this Chapter we investigate this question for graph-neural networks that compute the similarity between two tree inputs.

7.1 Introduction

Advances in artificial intelligence, particularly in deep learning, have had a huge impact on information retrieval and has changed the way we process, index and retrieve data. Artificial neural networks transform multimedia content like text, images, audio or video and compute latent representations which we can use to detect semantic similarities.

In this chapter we revisit the problem of retrieving related mathematical expressions. We investigate the embedding models powered by graph neural networks presented in Chapter 4.6.1. The model is designed to help scientists find related research based on math-queries. In a formula search engine, the users are interacting with a search interface that accepts queries presented as LaTeX mathematical expressions and are presented with results. These results are based on our neural network computations and it is often unclear how they came about. Traditional keyword queries are easily interpreted: We can highlight where the query-keywords or expressions related to the keywords appear in the results, thereby justifying the systems output. Because our system does not rely on exact matching of sub-terms, but rates overall semantic relatedness, the outputs are harder to interpret and harder to visualize, as we cannot necessarily highlight overlapping parts of result and query. To overcome this issue, we propose to apply so-called salience map for visualizing the similarity scores predicted by a graph convolutional neural network [230]. These visualizations help the

machine learning engineer who deploys the models in understanding how the model rates similarities, but may also be integrated in a user interface to support end-users.

The rest of this chapter is structured as follows: We present our two methods for visualizing embeddings of trees in Section 2. In Section 3 we discuss our results in the context of a search engine for mathematical content. We discuss our dataset and data preparation and perform a qualitative study of our visualizations. We conclude this paper with an outlook in Section 4.

7.2 Method

We begin to describe our method by reviewing the definition of graph convolutional neural networks. Then we propose two different methods for obtaining visualizations of nearest neighbor queries: one based on forward information and one based on backward information.

7.2.1 Graph Neural Networks for Trees

We define tree structures $x = (X, E)$ as a tuple of node-features X and edges E . Let $|x|$ denote the number of nodes in x . We assume that $X \in \mathbb{R}^{|x| \times d}$. A graph neural network maps a given tree to an output tree with transformed feature vectors in a d' -dimensional output space but with identical edge structure. We denote the neural network by $\phi(x) = (\phi(X, E), E)$. Let $\phi(x)_i \in \mathbb{R}^{d'}$ denote the output of the i -th node.

To store the fixed-size embedding of the whole tree in a vector database, we compute the mean of all nodes denoted by $\bar{\phi}(x) = |x|^{-1} \sum_i \phi(x)_i$. Then we can compute the cosine similarity or the dot product

$$\text{sim}(x, x') := \langle \bar{\phi}(x), \bar{\phi}(x') \rangle$$

in order to obtain the similarity of two trees.

7.2.2 Forward-Visualization

As the embedding of a tree is the mean of all its nodes, we can see that each pair of nodes contributes to the overall similarity. The similarity i.e. dot-product can be decomposed as

$$\langle \bar{\phi}(x), \bar{\phi}(x') \rangle = (|x| \cdot |x'|)^{-1} \sum_{i=1}^{|x|} \sum_{j=1}^{|x'|} \langle \phi(x)_i, \phi(x')_j \rangle.$$

Following the ideas of Stylianou and Pless [249], we visualize the similarity by coloring the nodes according to their contributions to the overall similarity. Hence given the query, we color the i -th node proportional to

$$c_i \sim \langle \phi_i(x), \bar{\phi}(q) \rangle.$$

This approach is computationally cheap, we just have to run one forward pass for every search result we want to visualize.

7.2.3 Backward-Visualization

Following the ideas of saliency maps [245], we color nodes in the tree according to the gradient of the similarity. Looking at a first order Taylor approximation of the similarity, we can write

$$\text{sim}(\vec{x} + \Delta, \vec{q}) \approx \text{sim}(\vec{x}, \vec{q}) + \langle \Delta, \nabla_{\vec{x}} \text{sim}(\vec{x}, \vec{q}) \rangle$$

to approximate how the similarity changes based on perturbations Δ to the input \vec{x} . If we decrease the value of x in the i -th coordinate, the corresponding coordinate in the gradient $\nabla_{\vec{x}}$ indicates whether the first-order approximation of the similarity decreases or increases. If the gradient is positive, then reducing the feature reduces the similarity. Thus the feature is important for the overall similarity. Hence we use the values in the gradient $\nabla_{\vec{x}}$ to visualize similarity.

In classical convolutional neural networks for image processing, the input is either a greyscale or an RGB-image. Thus at every position, there is either a 1-dimensional or 3-dimensional feature vector likewise gradient vector. If there is only one dimension, we do not need to aggregate the gradient, in the case of three dimensions we often aggregate by using the maximum gradient over all color channels.

In our case, we have higher-dimensional feature vectors in \mathbb{R}^d . Hence we have to aggregate the gradient

$$c_i \sim f(\nabla_{x_i} \text{sim}(x, q)).$$

where f is an aggregation such as max, mean or the sum of all non-negative gradients.

We are particularly interested in the case where x is a one-hot encoding of XML-data. Hence x is sparse and binary. We propose to use only gradient information of components that are set to 1 in the input to answer the question how the similarity changes if we flip inputs from one to zero. The component-wise multiplication of the input and the gradient eliminates all gradient mass at input components that are zero. We aggregate only the remaining components

$$c_i \sim f(x_i \odot \nabla_{x_i} \text{sim}(\vec{x}, \vec{q}))$$

and color the nodes proportional to this aggregation.

7.3 Qualitative Study: Interpretable Search for Formulas

We showcase our approach in a semantic search engine for mathematical equations which we have crawled from arxiv.org. First we give details on our dataset and machine learning model. Then we perform a number of example queries and discuss the quality of our visualizations.

Data We are again working on data obtained from arxiv.org, but filtered all publications that use the subject code `cs.LG` which covers computer science publications on machine learning. Of these 9,936 publications, we sample two subsets, index and query of size 7,949 and 1,987 respectively. We use the index-set for building our index using our graph neural network and the query set for search queries.

Embedding Network We use a graph convolutional network to map the 256-dimensional tree to a 64-dimensional tree and compute a fixed-size embedding by averaging over all nodes. We use a network architecture with 5 layers that use a hidden dimensionality of 256. Each layer computes a linear transformation of the inputs, computes the mean over all neighborhoods ($\square = \text{mean}$) and applies the ReLU activation to the outputs. After each layer, we use batch normalization. The network is trained as described in Section 4.6.1.

In this section we present a qualitative study of our proposed method in the context of math retrieval.

Rendering and Color Palettes First we want to discuss some peculiarities of MathML trees. The advantage of these trees is that the underlying XML-format of our representation is a mark-up language. Hence there is a native way to visualize them by rendering the equation with a corresponding markup processor like Katex or MathJax¹. We can add additional style information to the original markup to color the nodes according to our computed visualization. This way we obtain beautiful visualizations without worrying about issues like tree layouting etc. One problem with this approach is that we can only visualize some of the nodes. Particularly, with some exceptions like root or fraction bars, we will color only leaf-nodes, as most inner nodes belong to XML-tags that do not directly output visible symbols. For instance in Figure 4.7, the `<msubsup>`-tag corresponds to a node that does not output any symbols on its own, but only encapsulates the symbols of its children.

One way to mitigate this issue is to set the color intensity of the i -th node to the sum of all colors intensities on the path from root to i -th node. However we have seen that this puts too much emphasis on nodes that are deeper in the tree. Hence we currently omit all hidden nodes from the visualization.

We use max-aggregation of the gradients and compute colors by linear interpolation where black font indicates the most significant parts and grey parts are less significant.

Results We find semantically related equations by applying annoy to create an index structure for nearest neighbor retrieval with dot-product similarity. The retrieved equations will not be equivalent or strictly equal, but depicting similar concepts. We analyze the visualizations for the nearest neighbor as output by the index.

We compare the visualization based on the forward pass with visualization based on backward. In Examples 1-6 we present pairs of query (first row) and nearest neighbor

¹<http://mathjax.org>

$$\begin{aligned}
\mathbf{f}(\mathbf{x}) &= \mathbf{f}(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^T \nabla \mathbf{f}(\mathbf{x}_0) \\
g(\mathbf{x}) &= \mathbf{f}(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^T \nabla \mathbf{f}(\mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \mathbf{H} (\mathbf{x} - \mathbf{x}_0) \\
g(\mathbf{x}) &= \mathbf{f}(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^T \nabla \mathbf{f}(\mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \mathbf{H} (\mathbf{x} - \mathbf{x}_0)
\end{aligned}$$

Example 7.1: Example Query (Taylor Approximation)

$$\begin{aligned}
&\frac{1}{n} \sum_{i=1}^n l(\mathbf{f}(\mathbf{x}_i), \mathbf{y}_i) \\
\widehat{L}(\mathbf{f}) &= \frac{1}{n} \sum_{i=1}^n l(\mathbf{f}, \mathbf{x}_i, \mathbf{y}_i) \\
\widehat{L}(\mathbf{f}) &= \frac{1}{n} \sum_{i=1}^n l(\mathbf{f}, \mathbf{x}_i, \mathbf{y}_i)
\end{aligned}$$

Example 7.2: Example Query (Empirical Risk Minimization)

$$\begin{aligned}
\mathbf{h}^{l+1} &:= \sigma^l(\mathbf{W}^l \mathbf{h}^l + \mathbf{b}^l) \\
\mathbf{z}^l &= \mathbf{f}(\mathbf{W}^l \mathbf{z}^{l-1} + \mathbf{b}^l) \\
\mathbf{z}^l &= \mathbf{f}(\mathbf{W}^l \mathbf{z}^{l-1} + \mathbf{b}^l)
\end{aligned}$$

Example 7.3: Example Query (Multi-Layer Perceptron)

$$\begin{aligned}
\Delta_f(\mathbf{e}|\mathcal{S}) &= \mathbf{f}(\mathcal{S} \cup \{\mathbf{e}\}) - \mathbf{f}(\mathcal{S}) \\
\Delta_f(\mathbf{r}|\mathcal{S}) &\triangleq \mathbf{f}(\mathcal{S} \cup \{\mathbf{r}\}) - \mathbf{f}(\mathcal{S}), \\
\Delta_f(\mathbf{r}|\mathcal{S}) &\triangleq \mathbf{f}(\mathcal{S} \cup \{\mathbf{r}\}) - \mathbf{f}(\mathcal{S}),
\end{aligned}$$

Example 7.4: Example Query (Submodular Functions)

$$\begin{aligned}
&\mathbb{E}_\sigma \sup_f \frac{1}{n} \sum_{i=1}^n \sigma_i f(\mathbf{x}_i) \\
\mathcal{R}_n(\mathcal{F}) &= \mathbb{E} \sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \varepsilon_i f(\mathbf{x}_i) \\
\mathcal{R}_n(\mathcal{F}) &= \mathbb{E} \sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \varepsilon_i f(\mathbf{x}_i)
\end{aligned}$$

Example 7.5: Example Query (Rademacher Complexity)

and color the result according to the forward pass (second row) and backward pass (third row). All examples are from the context of machine learning, which allows us to judge the results based on our background-knowledge of the research area. We now discuss each example in detail:

In Example 7.1, we have queried the definition of the first-order Taylor approximation. The embedding model retrieved the second-order Taylor approximation as

$$\begin{aligned}
P(\bar{X} - E[\bar{X}] \geq t) &\leq e^{-2nt^2} \\
\Pr(\bar{X} - E[\bar{X}] \geq t) &\leq e^{-2nt^2} \Pr(\bar{X} - E[\bar{X}] \leq t) \leq e^{-2nt^2} \\
\Pr(\bar{X} - E[\bar{X}] \geq t) &\leq e^{-2nt^2} \Pr(\bar{X} - E[\bar{X}] \leq t) \leq e^{-2nt^2}
\end{aligned}$$

Example 7.6: Example Query (Hoeffding’s Bound)

nearest neighbor. Both visualization put the emphasis on the ∇ -symbol, the symbol commonly used for gradients. Interestingly, the backward visualization indicates that the H symbol, commonly used for the Hessian matrix i.e. the second order derivative, also contributes to the overall similarity score. Apparently the model has learned that both symbols appear frequently together, e.g. in the context of differentiable functions.

The query in Example 7.2 is the empirical risk functional commonly used in machine learning. Both approaches highlight that we have a sum over objects subscripted by i . In addition, the backward approach highlights the $\hat{\cdot}$ -symbol commonly used to indicate that a random quantity is estimated using a finite sample. The model seems to have learned that it frequently co-occurs with estimators of the form $\frac{1}{n} \sum_{i=1}^N z_i$.

Example 7.3 shows a query for the definition of a fully connected neural network layer. The forward-visualization focusses on the variables W, B for weight matrix and bias vector, while the backward variant focusses on the superscripted index indicating the number of the layer l .

In Example 7.4, we see the definition of gain used in the context of submodular function maximization, where both visualizations highlight the \cup symbol. There is almost no difference between the two variants, most notable is the stronger emphasis on the opening curly-brace in the forward-visualization.

For Example 7.5, we have queried the definition of the empirical Rademacher complexity, a measure used e.g. in statistical learning theory. Both approaches highlight the sup operation, which is indeed essential for the definition. Without the sup, the expected value would trivially amount to 0. The forward visualization seems to highlight the combination of expectation and supremum, whereas the backward variant highlights mostly supremum and ε_i . This indicates that the model has correctly learned that both σ and ε are frequently used for Rademacher averages.

The last Example 7.6 shows concentration inequality, more precisely Hoeffding’s bound that bounds the deviation from a sample mean to the true expected value of a random variable X . Both visualizations highlight the $\bar{\cdot}$ symbol that is frequently used to indicate sample means and hence is related to the $\hat{\cdot}$ symbol for empirically estimated quantities used by the query. Additionally both visualization highlight the X that is generally used for random variables in statistics literature. It is interesting to note that the forward vizualization also places emphasis on the \leq operator, while the backward visualization mostly highlights the \geq .

Overall we see that the forward visualizations spread the emphasis more evenly over the tokens than the backward pass. We think that with each layer of the encoder network the information in the nodes becomes less local and more global, as the feature vector at each node is a function of all its neighbors on the previous layer. This mixing

makes it more difficult to highlight where the important information is originally coming from. The backward-based approach does not have this issue. Both approaches put a lot of emphasis on math operators like equal or plus-signs.


7.4 Conclusion and Outlook

In this Chapter we have proposed two different approaches for visualizing similarity computations between embeddings computed by graph convolutional networks. We have applied our method in the domain of mathematical retrieval. Both methods have allowed us to gain insights into the way our model scores similarities. The visualizations indicate that our model was able to learn conventions and notations to infer the context in which equations appear.

We are confident that the methods proposed in this work are also useful in other application domains. For instance when we work with graph representations of molecules [67] and use geometric deep learning to solve classification or metric learning tasks, we can color two-dimensional or three-dimensional illustrations of the molecules using our methods.

Chapter 8

Why Did the Model Learn a Representation?

 CONTEMPORARY approaches for explainability in machine learning focus either on explaining individual predictions or on explaining the full model [144, 237]. Explanation methods should answer the question “Why did the model compute this output?”. Many methods approach this question by providing a description on what the model does or how it computes its outputs. Selbst and Baricas observe that

“so far, the majority of discourse around understanding machine learning models has seen the proper task as opening the black box and explaining what is inside.” [237]

For instance for convolutional neural networks, saliency methods [245] provide explanations in the form of heatmap overlays for the input images highlighting which region was most important to the convolutional neural network. In the last Chapter we have seen a saliency-map approach for graph-neural networks that was applied to interpreting retrieval results in a mathematical search engine. However, it is left to the user to infer or interpret why that region is important for the decision. Furthermore, these methods often show little dependence on the model they try to explain [3].

We would like the explanation to include evidence that the decision is correct and that the model behaves sensibly. We argue that, in the case of machine learning models, the leading question for designing explanation methods should be extended to “Why did the model *learn* to compute this output?” The key to answering that question lies in the training process of the model. For instance a decision tree can provide evidence for its decision by reporting the class frequencies that fall into nodes along its decision path. In this Chapter we aim to provide similar evidence for the decisions of deep networks.

We propose a method for tracking the training process of deep networks. By archiving all weight updates of a deep network, we can attribute the representations computed at each layer to individual training examples. Ultimately we can support

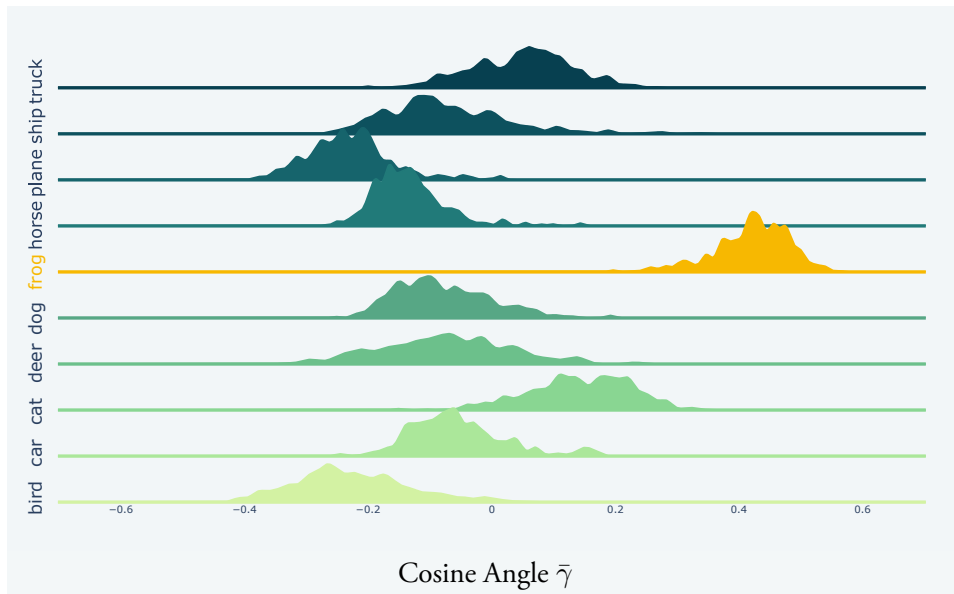


Figure 8.1: Example Explanation Histogram: “*The model predicts **frog** as training examples of that class had the highest influence on the computed intermediate representation in layer 14*”.

the explanation of a model’s decision with data points in the training data. To this end, we propose two visual explanations: First, we can identify and show the most-influential training examples, second we can visualize how all training examples of a class contributed to the computed representation in ridge plots like in Figure 8.1.

8.1 Related Work

We have already reviewed related work on explainable artificial intelligence. In addition, perhaps most relevant to this work are methods that also trace the training process of gradient-based methods: Garima et al. [81] decompose the losses of training instances to sums over the losses of all intermediate models during training. They employ a first-order Taylor approximation to compute the contribution to the loss as a function of the gradient. This allows them to analyze how much a training example contributed to the loss of the final model given a test example. As in our approach, this requires storing parameters in all training steps: While we store the parameter changes, the authors employ an approximation step and store only the intermediate models. Garima et al. justify this decision with the storage and IO demands that render tracking of all updates infeasible, whereas we find that it is tractable to store all weight updates on our hardware. As with our approach, this exact formulation only works in the stochastic setting without minibatches, but minibatches are supported by applying another approximation step. Ultimately, the method they propose judges the influence of a given training example by the dot products between the weight gradient

of the loss given the training example x_i and the gradient given the test example x , summed over all T checkpoints $\theta^{(t)}$

$$I(x_i, x) = \sum_{t=1}^T \left\langle \nabla_{\theta} \ell(f_{\theta^{(t)}}(x_i)), \nabla_{\theta} \ell(f_{\theta^{(t)}}(x)) \right\rangle \quad (8.1)$$

Follow-up work by Zhang et al. [301] extends their work for natural language classifiers which allows finer-grained contribution of the influence to spans of training texts, rather than the full text. Swayamdipta [253] analyze the training process, but their goal is to identify easy and hard training instances in the training data to help the machine learning engineer assess the data quality. Tracking the training process is also interesting for establishing generalization results for stochastic optimizers [110, 256].

8.2 Method

Stochastic gradient descent (SGD) and its variant are the most popular and most frequently used learning algorithms for a diverse range of models, most notable deep networks, but also for kernel methods [208] or matrix factorization methods [106]. For supervised learning, SGD trains a model f_{θ} parameterized by real-valued weights θ by minimizing the empirical risk

$$\arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \ell(f_{\theta}(x_i), y_i).$$

Optimization is performed in T iterations, where, at the t -th iteration, we sample an example (x_t, y_t) from the training data and update the weights in the negative direction of the gradient of the loss

$$\theta^{(t+1)} = \theta^{(t)} - \eta^{(t)} \nabla_{\theta} \ell^{(t)}(f_{\theta^{(t)}}(x_t), y_t) \quad (8.2)$$

with a small step-size $\eta^{(t)}$. Hence the output of the SGD training algorithm is a vector of weights that is the sum of all weight updates performed during training

$$\theta^* = \sum_{t=1}^T \underbrace{-\eta^{(t)} \nabla_{\theta} \ell^{(t)}(f_{\theta^{(t)}}(x_t), y_t)}_{=: \Delta^{(t)}}. \quad (8.3)$$

For the purposes of this paper, we can view this as a sum of vectors $\Delta^{(t)}$. Hence variants of SGD like SGD with momentum [252], Adam [134] or SAM [78] also fit within this framework. These variants do not use the plain gradients, but update the weights based on a function of the gradient and an internal state of the optimizer. In this case, we denote by $\Delta^{(t)}$ the difference of the weight vector before and after the t -th training step.

We propose methods for explaining deep learning representations based on tracking the $\Delta^{(t)}$ values during the training stage. We show how it can be used to identify the most influential training examples, which can be used as a form of explanation of

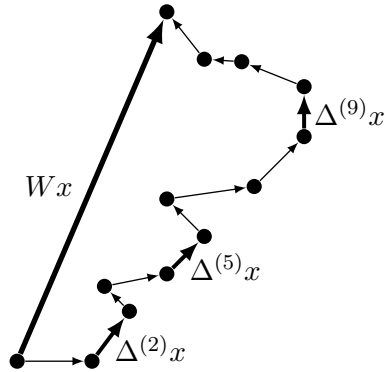


Figure 8.2: Illustration of the decomposition of preactivations into individual gradient updates. The marked updates are the most substantial individual contributions to Wx .

model decisions. Next we propose a visual explanation that summarizes the training process of a learned representation.

8.2.1 Explaining Pre-Activations by the Training History

Deep networks compute outputs in a sequence of layers. In this work we tackle the problem of providing explanations for the intermediate representations learned in any of these layers. We begin our analysis for the case of single-instance stochastic training without mini-batching as in (8.2). In the next sections we develop methods for training with mini-batches.

For notational simplicity we define a deep network as a composition of layers. We note that our method can also be applied to more complicated networks e.g. networks with residual connections, etc. For additional simplicity we limit our analysis to linear layers without bias terms, which includes fully connected layers as well as convolutional layers. We define the *output* of the l -th layer of a deep network

$$f_l(x) = \sigma_l(W_l f_{l-1}(x)) \quad (8.4)$$

where σ_l is an activation function and W_l is a real-valued weight matrix. We define $f_0(x) := x$. We focus on the *pre-activations* $W_l f_{l-1}(x)$ of the l -th layer.

In a slight abuse of notation, following (8.3), we define the decomposition of the weights into the contributions at every training step

$$W_l = \sum_{t=1}^T \Delta_l^{(t)} \quad (8.5)$$

and consequently we can decompose the preactivations as

$$W_l f_{l-1}(x) = \sum_{t=1}^T \Delta_l^{(t)} f_{l-1}(x) \quad (8.6)$$

as is depicted in Figure 8.2. To measure the most substantial individual contributions to the preactivation, we compute the projection of the contributions onto the overall representation in the l -th layer

$$\gamma_l^{(t)} := \left\langle W_l f_{l-1}(x), \Delta_l^{(t)} f_{l-1}(x) \right\rangle \quad (8.7)$$

and we use large γ -values to construct explanations. In addition to the dot product, we also consider the cosine similarity

$$\bar{\gamma}_l^{(t)} := \gamma_l^{(t)} \cdot \left(\|W_l f_{l-1}(x)\|_2 \cdot \|\Delta_l^{(t)} f_{l-1}(x)\|_2 \right)^{-1} \quad (8.8)$$

that describes the angle between the contribution and the final representation.

One special layer is the last layer, as it outputs the logits of the predictions. Hence we can contribute the predicted class probabilities to individual training steps by inspecting the decomposed preactivations there.

8.2.2 Computing Most Influential Examples

We can use the γ values to find the most influential training examples for a decision of interest. Each $\gamma^{(t)}$ is based on exactly one $\Delta^{(t)}$ which, in turn, is based on the loss computed on exactly one training example for stochastic training without minibatching. During training over multiple epochs, each example is used for training multiple times. In this case, we aggregate all γ values for each training example by summing them up to get a total contribution over all training epochs. This approach is outlined in Algorithm 3.

Note that for learning a classifier that discriminates one class vs. all the other classes, influential examples must not necessarily belong to the positive class. Perhaps even the more interesting examples belong to the other class but were misclassified during training, leading to weight updates to correct the behavior.

Algorithm 3 Computing Influential Examples

```

function COMPUTE-INFLUENCE( $f, x, \Delta_l^{(\cdot)}$ )
   $\Gamma_i = 0$  for  $i = 1, \dots, n$                                 ▷ Initialize Influences
  Compute  $f_{(l-1)}(x)$  and  $W_l f_{(l-1)}(x)$ 
  for  $t = 1, \dots, T$  do                                    ▷ Iterate Training History
    Identify example  $x_i$  used for  $\Delta^{(t)}$ 
    Compute  $\gamma_l^{(t)}$  with Eq. (8.7)
     $\Gamma_i = \Gamma_i + \gamma_l^{(t)}$                                 ▷ Sum up example importances
  end for
  return top- $k$  indices in  $\Gamma$ 
end function

```

8.2.3 Visualizing Contributions in Ridge Plots

Privacy Requirements may prevent us from explaining decisions to users by showing the labeled training data as evidence, e.g. in situations where training data contains personal information like for automated credit scoring or in medical applications. In many situations we can still provide explanations based on aggregated information, e.g. when the number of training examples is sufficiently large that aggregation does not leak personal information. Displaying aggregated information has the additional advantage that it shows a more complete picture of the training process, whereas showing individual training instances as explanations shows only a very small part of the process and the interpretation of the instances is very subjective.

For visualizing aggregated statistics, we propose to show ridge plots to aggregate all $\gamma_l^{(t)}$ values grouped by layer and training labels. These plots for a single layer are structured as follows: We use one subplot for each class of interest. Its x -axis shows the cosine similarities $\bar{\gamma}_l^{(t)}$ between the individual contributions and the total preactivations according to (8.8), on the y -axis we show a density estimate of how frequently this cosine similarity appeared during training. Instead of showing raw counts, we weight each of them by the magnitude of the update $\|\Delta_l^{(t)} f_{l-1}(x)\|_2$. We decided against the use of histograms and used kernel density estimates instead. While this is mostly for aesthetic reasons, it also reduces the visual complexity of the plots by smoothing out the noise, making them more digestible for the viewer.

By controlling the kernel bandwidth parameter we can control the amount of smoothing to the plotted density. Throughout our analysis we use a bandwidth of 0.1 which gives a good tradeoff between accuracy of the density estimates and readability. This approach is outlined in Algorithm 4. For visual guidance we highlight the predicted class and only show the classes with the top-10 predicted probabilities. You can find an example ridge plot in Figure 8.1: A high influence of a certain class is indicated by large amount of probability mass at high similarities, as is the case in the example for the target class "frog". Note that inspecting the output layer in these plots goes beyond obtaining a class probability at prediction time: They allow to understand how these class probabilities came about.

8.3 Batch-Wise Tracking of Contributions

Mini-batch training can significantly decrease the duration of training by reducing the overall number of parameter updates to a model. Modern deep networks are almost always trained with mini-batching to allow the efficient processing of massive datasets exploiting parallel computations on GPUs. The prominent deep learning frameworks are build with mini-batching in mind and gradients of network parameters are accumulated over the batch during the backward pass. In our case, however, it causes a serious problem: When we track updates on the granularity of minibatches, we can no longer attribute updates to individual training instances or their respective class labels, as the gradients are already averaged across instances. In this Section, we propose an

Algorithm 4 Computing Ridge Plots

```
function VISUALIZE-INFLUENCE( $f, x, \Delta_l^{(\cdot)}$ )  
  for each class  $y$  do ▷ Initialize statistics  
     $\Gamma_y = \emptyset$   
  end for  
  Compute  $f_{(l-1)}(x)$  and  $W_l f_{(l-1)}(x)$   
  for  $t = 1, \dots, T$  do ▷ Iterate training history  
    Identify class  $y_i$  used for  $\Delta^{(t)}$   
    Compute  $\bar{\gamma}_l^{(t)}$  with Eq. (8.8)  
    ▷ Store cos-similarity and weight for class  $y_i$   
     $\Gamma_{y_i} = \Gamma_{y_i} \cup \{(\bar{\gamma}_l^{(t)}, \|\Delta_l^{(t)} f_{l-1}(x)\|_2)\}$   
  end for  
  plots =  $\emptyset$   
  for each class  $y$  do  
    ▷ Plot cosine similarities and weights from  $\Gamma_y$   
    plots = plots  $\cup \{\text{kdePlot}(\Gamma_y, \text{kernel-bw} = 0.1)\}$   
  end for  
  return plots  
end function
```

alternative for mini-batching where we train with mini-batches of training instances that all belong to the same class. This way we can at least attribute weight updates to classes rather than individual examples.

More specifically, we first compute a random permutation of our dataset and then group our data set by class. We generate mini-batches by first sampling a class with probability proportional to the prior probability we find in the data set. Then we generate a random batch by yielding the desired number of examples that belong to the class. We stop when we do not have enough data points for a full minibatch available for all classes. The next epoch of training proceeds with a new random permutation of the full dataset, giving rise to new mini-batches. This way we obtain random mini-batches of a single class.

8.3.1 Ghost-Batchnormalization Layers

Using mini-batches of a single class causes new problems for many modern deep network architectures, as it limits the usefulness of batch normalization layers. The appealing property of batch normalization is that it normalizes features in a batch, such that their values lie in a standardized range, which allows efficient training without numerical instabilities, but also differ, which allows the model to discriminate between the classes [115]. But when we train with batches of a single class, the batch normalization layer forces feature values to differ that should be similar, as illustrated in the Figure 8.3b. In particular, a following batch of datapoints from a different class would

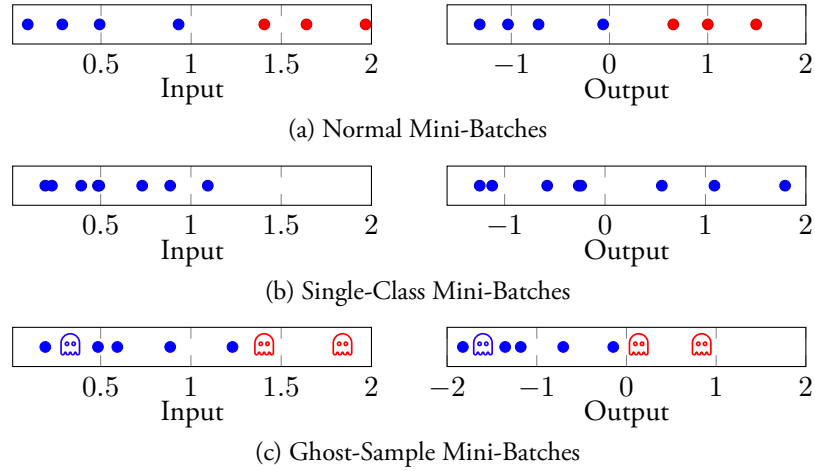


Figure 8.3: Illustration of the different batch-normalization variants, color indicates class labels. (b) When we construct batches with a single class, the outputs span the whole range. An imaginary next batch of the other class would do the same, hindering training. (c) Ghost samples solve this problem, as mean and variance are also computed on the ghost batch.

consequently span the same range of feature values.

We introduce *ghost samples* to the mini batch to prevent this problem: These ghost samples are training instances sampled from all classes according to the class prior probabilities. We compute the batch normalization statistics on the ghost samples and the normal batch separately. Then we combine the statistics by downweighting the examples in the mini batch to only have a total weight corresponding to one example in the ghostbatch as outlined in Algorithm 5. This is done using the standard formulas for weighted merging of two means and variances [42]:

$$\mu = \mu_{\text{ghost}} + (\mu_{\text{batch}} - \mu_{\text{ghost}})/(1 + |x_{\text{ghost}}|) \quad (8.9)$$

$$c = |x_{\text{ghost}}|/(1 + |x_{\text{ghost}}|) \quad (8.10)$$

$$\sigma^2 = c \cdot \sigma_{\text{ghost}}^2 + \sigma_{\text{sample}}^2 + c \cdot (\mu_{\text{batch}} - \mu_{\text{ghost}})^2/(1 + |x_{\text{ghost}}|) \quad (8.11)$$

This way all examples contribute to the mean and variance estimates, but the same-class batch has only little influence, as illustrated in the third row of Figure 8.3. At inference time, we use running estimates of mean and variance, as it is the standard for batch normalization layers.

When evaluating the loss of our model, however, we only use the samples in the original minibatch without the ghost samples. This way normalization is based on all classes, while loss is based only on a single class. Now the model's gradients and consequently the parameter updates are mostly based on the single-class batch. This allows us to still attribute individual training steps to a set of examples of a single class. Instead of presenting the examples that were most influential, we can now present

Algorithm 5 Batch-Normalization with Ghost Samples

```
1: function GHOSTSAMPLE-BATCHNORM( $x, \varepsilon = 10^{-5}$ )
2:   ▷ Split ghost-batch examples and compute statistics
3:   [ $x_{\text{batch}}, x_{\text{ghost}}$ ] = split( $x$ )
4:    $\mu_{\text{ghost}}, \mu_{\text{batch}}$  = mean( $x_{\text{ghost}}$ ), mean( $x_{\text{batch}}$ )
5:    $\sigma_{\text{ghost}}^2, \sigma_{\text{batch}}^2$  = var( $x_{\text{ghost}}$ ), var( $x_{\text{batch}}$ )
6:   ▷ Combine batch and ghost statistics
7:    $\mu$  =  $\mu_{\text{ghost}} + (\mu_{\text{batch}} - \mu_{\text{ghost}})/(1 + |x_{\text{ghost}}|)$ 
8:    $c$  =  $|x_{\text{ghost}}|/(1 + |x_{\text{ghost}}|)$ 
9:    $\sigma^2$  =  $c \cdot \sigma_{\text{ghost}}^2 + \sigma_{\text{sample}}^2 + c \cdot (\mu_{\text{ghost}} - \mu_{\text{batch}})^2/(1 + |x_{\text{ghost}}|)$ 
10:  Update batch-norm running statistics with  $\mu$  and  $\sigma^2$ 
11:  return  $(x - \mu)/(\sigma + \varepsilon)$ 
12: end function
```

the most influential minibatches. When computing the overall contribution of an individual example, the contributions of a minibatch are attributed to all its training examples equally. This way we can still aggregate the contributions by summation. While we could also aggregate contributions in a normal mini-batch, in practice this yields unusable values. With one-class minibatches, however, the results are better. We believe this is due to the lower variance in the batch. In the next section we present an alternative visual explanation that aggregates class-wise rather than example-wise.

8.3.2 Implementation for pyTorch

We have implemented batch-wise tracking of gradients as a pytorch library. This library allows users to easily integrate our proposed approach into existing deep learning codebases. We have illustrated this in a small example in Fig 8.4 for training a VGG16 network on Cifar-10 data. In order to integrate our approach, we have to make the following changes:

- Provide unique identifiers for each training example. This can be achieved conveniently by wrapping an existing `torch.utils.data.Dataset` object using our wrapper class `WrappedDataset`, that automatically assigns ids. For instance for a dataset (x_i, y_i) it instead returns tuples (x_i, y_i, i) .
- Replace standard batch normalization layers with our ghost-batchnormalization layer. This can either be done manually or using our `patch_batchnorm` convenience function.
- Replace the default batch sampler in the `torch.utils.data.DataLoader` to generate single-class batches with ghost-samples. We have implemented this in the `ClassSampler` class.
- Wrap the `torch.optim.Optimizer` used for training to enable the weight-update tracking functionality. We obtain a anonymous wrapped optimizer class

by invoking `WrappedOptimizer()`. Then we can create an optimizer by calling its constructor with the hyperparameters that we want to use for optimizing. Its `step()` method requires additional arguments: The indices and optionally class labels of the training batch. Furthermore, the optimizer has a `done()` method that gracefully terminates the logging.

8.3.3 Experimental Evaluation

In our experimental evaluation, we investigate the behaviour of the batchwise-tracking of weight updates and the usefulness for generating explanations for model decisions.

Cifar-10 Image Classification We investigate our method for a popular image classification benchmark, the Cifar-10 dataset. We train a convolutional neural network that uses the VGG-16 architecture - 13 convolution layers followed by 3 fully connected layers - and train it with our mini-batch training algorithm with a batch size of 128 examples, of which 40 examples are the ghost sample containing 4 images of each class. Note that the classes are balanced in the train- as well as test-data, so the ghost sample also should also have this property. We train using vanilla stochastic gradient descent for 35 epochs with an initial learning rate of 0.01 which is decreased by a factor of 0.1 every 10 epochs. Our model achieves a test accuracy of 80%, which is not state-of-the-art, but sufficiently high to demonstrate the usefulness of our explanation approach. We have noted that training with same-class minibatches leads to slower minimization of the training loss, indicating that it is harder to learn models in this setup. This may have an impact also on inference performance of the models.

We investigate both types of visual explanations proposed in this Section, the most influential training examples, see e.g. Figure 8.5c and ridge plots of all contributions, e.g. in Figure 8.6. Both forms of visualization show changing behavior along the depth of the network: In the ridge plots, we see that later layers are more specific to classes and hence seem more important for explanations, whereas earlier layers do not differ substantially. For instance in the 13th linear layer, Figure 8.5a, we see a clear peak of high similarities i.e. high influence for the true class "truck", whereas in the earlier 4th convolution layer, Figure 8.5c, the classes behave more homogeneous. When looking at the most-influential training images, the later layers show only images of the predicted class, as in Figure 8.5c. Again earlier layers show very mixed images, as e.g. the 1st layer in Figure 8.7. Further analysis reveals that these outputs are triggered by weight updates very early in the training process when the model was still untrained and the optimization performed large weight updates to learn. Other layers, like the 4th layer in Figure 8.5a, again only show images of the target class, but we do note a higher variety in the images, for instance in the perspective of the images, in comparison to the later layers. All in all, this changing behavior along the depth of the network is in line with previous results, that illustrate how convolution filters match more complex patterns corresponding to the classes in later layers, whereas the earlier layers match basic concepts like edges and textures [72].

Furthermore we can see that related classes influence each others representation, e.g. truck and car or plane and bird are often both influential on middle layers. This phenomenon is present in the ridge plots (see eg. Figure 8.6b) as well as the most influential examples (see e.g. Figure 8.5b).

8.3.4 Explanations for Graph Classification

In this section, we investigate explanations for a graph neural network. We work with the ogbg-ppa dataset that contains undirected protein association neighborhoods from 37 different taxonomic groups which is the prediction target [113, 255, 304]. Each node represents a protein and each edge represents an interaction and is described by 7 features. We use the official train-test split, where the training data contains x graphs and the test data contains y graphs. Unlike in the previous section, the classes are not perfectly balanced. We use a graph convolutional neural network based on the model proposed by Xu et al. [287] that uses 5 layers of graph convolutions and a hidden dimensionality of 300. The edge features are transformed to 300-dimensional incoming node features [113].

The graph convolution transforms the features of a node v by

$$f_l(x_v) = \text{MLP}_l \left(\sum_{(u,v) \in \mathcal{E}} \max[x_u + W_l \phi(u, v), 0] \right) \quad (8.12)$$

where $\phi(u, v)$ denotes the edge features of the edge from u to v . We use the edge transformation $W_l \phi(u, v)$ as well as all linear layer in MLP_l to compute explanations.

We use SGD training with Nesterov momentum of 0.9 and an initial learning rate of 0.001 that is decreased by factor 0.1 after 10 and 40 epochs. Our model trained with single-class minibatches of size 32 achieves an accuracy of 71.5% on the test data, which places is comparable to performances of the baselines on the current public leaderbord for the ogbg-ppa dataset.

We only inspect the ridge plots in this application, as images of graphs with more than 100 nodes and 6 numerical edge features are very hard to interpret, particularly without domain knowledge. The visualizations for this graph neural network reveal, that all but the last two layers have almost uniform behavior throughout the classes, as e.g. shown in Figure 8.8c. This indicates that the network is not benefitting from its depth, as the intermediate representations are not discriminative for the target class. This highlights another possible use-case of our method for machine learning engineers who can use the approach for debugging machine learning models.

Computational Overhead Biggest roadblock for implementing this system seems to be the additional overhead during training. In this section, we report the impact on runtime and memory or storage. All reported runtimes are measured on a NVIDIA DGX A100 machine using a single A100 GPU with 400GB memory.

As we can see in Table 8.2, tracking all weight updates requires 1.5 and 1.9 terabytes of disk space. While this seems like a lot, it also is small enough to fit on modern

Table 8.1: Computational Overhead Statistics for the Previous Experiments

Metric	cifar10	ogp-ppa
Model Size	129MB	7.1MB
Number of Weight Updates	11,685	278,500
Runtime per Epoche	72sec	264s
Relative Overhead Runtime	+453%	+54%
Disk Storage Demand	1.5TB	1.9TB
Explanation Time	10min	66min

```

import torch
from torch.optim import SGD
from xai_tracking.xai import WrappedOptimizer
from xai_tracking.xai import WrappedDataset
from xai_tracking.xai import ClassSampler
from xai_tracking.xai import patch_batchnorm

# train_set = CIFAR10()
train_set = WrappedDataset(CIFAR10())

GHOST_SAMPLES=64
sampler = ClassSampler(train_set, ghost_samples=GHOST_SAMPLES)
# train_loader = torch.utils.data.DataLoader(train_set, \
#                                             batch_size=512)
train_loader = torch.utils.data.DataLoader(train_set, \
                                           batch_size=512, \
                                           sampler=sampler)

model = VGG16()

# replace batch normalization layers with ghost normalization
model = patch_batchnorm(model, ghost_samples=GHOST_SAMPLES)

# wrap optimizer class to enable tracking
SGD = WrappedOptimizer(SGD, history_file="history")
optimizer = SGD(model.parameters(), lr=0.1)

# [...]

for batch in train_loader:
    # inputs, labels = data
    inputs, labels, indices = data
    optimizer.zero_grad()
    predictions = model(inputs)[:GHOST_SAMPLES]
    loss = loss_function(predictions, labels[:GHOST_SAMPLES])
    loss.backward()
    # optimizer.step()
    optimizer.step(ids=indices, labels=labels)

optimizer.done() #Wait for all Processes to finish, close open files

```

Figure 8.4: Sample Code for Capturing the Training Process of a VGG16 model trained with SGD on the CIFAR-10 dataset. Required changes are highlighted, we see that we need to include indices in our training data, use our class-wise samples, substitute batch normalization for our ghost normalization layers and use a wrapped variant of the optimizer.

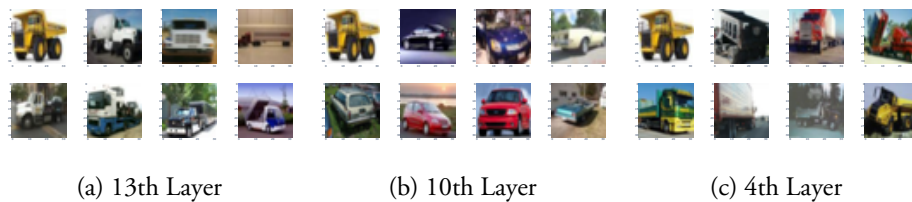
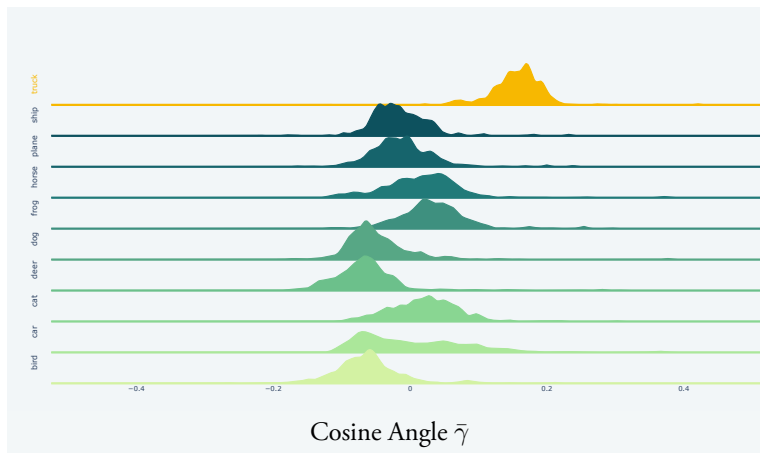
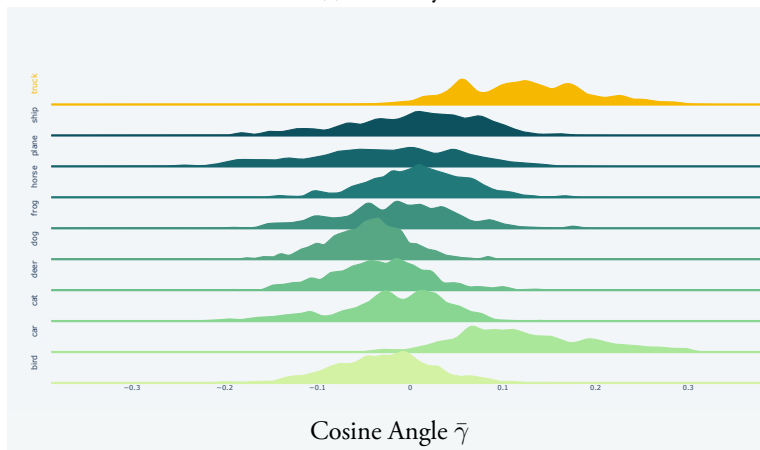


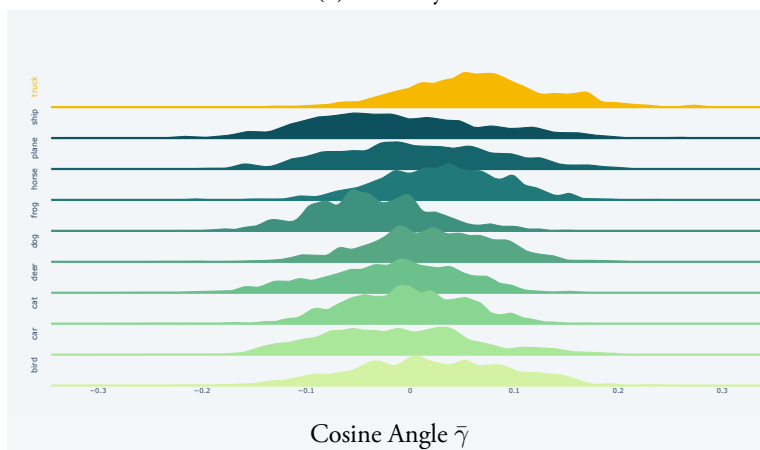
Figure 8.5: The 7 most influential training examples for the 3 layers of the VGG-16 network. Top left image is the input image for reference. We see a high influence of the class "car" on the representation in the middle layers, here e.g. in the 10th layer. The later layers, eg. the 13th layer, is influenced mostly by very similar images, i.e. trucks photographed from an angle, whereas the earlier convolutional layers, here the 4th layer, is influenced by images of trucks shot from a variety of angles.



(a) 13th Layer



(b) 10th Layer



(c) 4th Layer

Figure 8.6: Ridge Plot for the 3 layers of the VGG-16 network. We see a high influence of the class "car" on the representation in the middle layers, here e.g. in the 10th layer. The earlier layers are not as descriptive of individual classes, eg. the 4th layer.

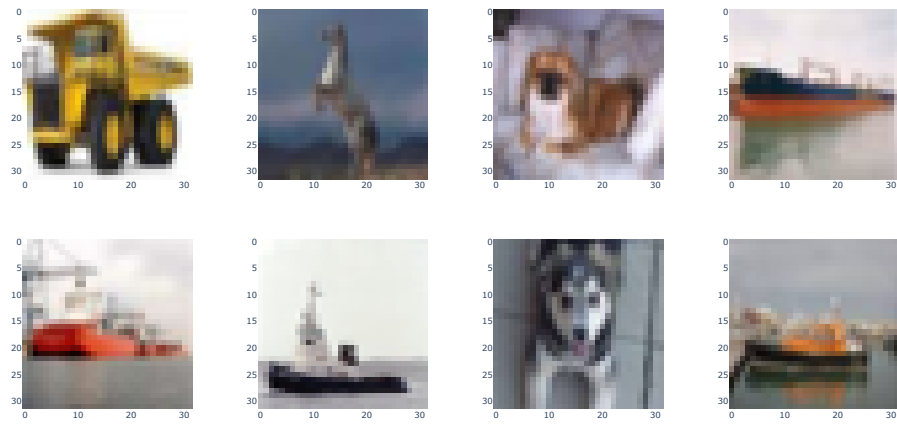


Figure 8.7: Highly influential images for the first layer from a variety of classes leave the practitioner confused. Input image in top left position for reference.

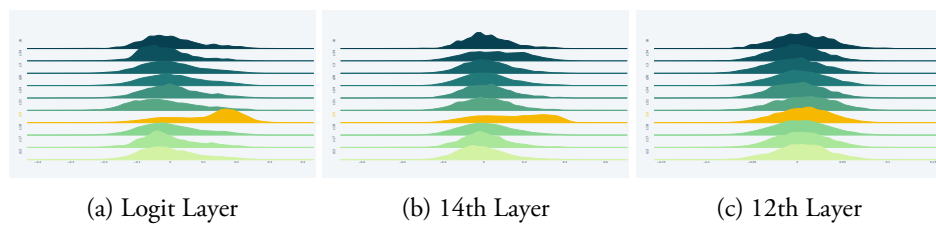


Figure 8.8: Ridge Plot for the output layer, the 14th layer and the 12th layer. We see that while the representation at the 14th layer can be attributed to examples of the correct class, the representations in the 12th layer and in all prior layers look unspecific to a class. This indicates a poor choice of network architecture that does not utilize all layers for classification.

compute architecture. But, generating explanations is expensive, as we have to load these terabytes of parameters efficiently from disk onto GPU for one-time use. These I/O operations are the current computational bottleneck, explaining the long running times of generating explanations. Note that the time per explanation decreases when we generate a batch of explanations, as we only need to load the weight changes once per batch rather than once per example.

At training time, the overhead of storing all weight updates to disk on the training time depends on the model size: For the large VGG-16 model the impact is very high, mostly due to the large linear layers with over 4mio parameters each, whereas for the small graph neural network it is not severe. At the moment we naively store all weights updates uncompressed to disk, which is probably very wasteful and makes explanation rather slow, because massive amounts of weights have to be transferred from disk to GPU.

Different options for making this more efficient come to mind: We can for instance approximate the weight updates of large linear layers with low-rank approximations. For standard SGD-training, these updates are indeed low-rank as the weight update is computed as an outer product of input and gradient and thus its rank is at most the batch-size. We explore this further in the next section. An alternative to low-rank approximations are random projections as e.g. Garima et al. [81] propose.

8.4 Example-Wise Tracking of Contributions

In the last section we have developed an explanation approach that works by tracking the weight updates during training. While it can be wrapped around any optimization procedure that incrementally updates weights based on examples, it prohibited the use of general mini-batches. We can either use batch-size of 1 or mitigated this by using minibatches of the same class. Consequently, it required modifying the training protocol, which includes finding new hyperparameters that work well with same-class mini-batches. In this Section we present another approach that allows the use of general mini-batches, but needs to use modified optimizers implementations: We sacrifice the flexibility of wrapping any optimizer for the ability to handle general minibatches. We demonstrate our approach with an implementation for vanilla SGD optimization, but note that it can be adapted to other gradient-based optimizers like SGD with Momentum or Adam. This way, once we have implemented our modified optimizer of choice, we can use the same training hyperparameters for training with and without tracking.

8.4.1 Method

We look specifically into tracking vanilla stochastic gradient descent training with mini-batches. Consequently, the t -th update is of the form

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta^{(t)}}{k} \sum_{i=1}^k \nabla_{\theta} \ell(f_{\theta^{(t)}}(x_{s_i^{(t)}}), y_{s_i^{(t)}}) \quad (8.13)$$

where $s_1^{(t)}, \dots, s_k^{(t)}$ are the indices of random samples selected from the training data that are used in the t -th mini-batch.

We want to explain deep representations by decomposing their linear parts into sums where the summands correspond to individual training examples. The gradient of the mini-batch is just the average gradient of the individual examples in the mini-batch. Consequently, in a fully-connected layer $h^{(l)} = W^{(l)}h^{(l-1)}$ the gradient $\nabla_{W^{(l)}}\ell$ has low-rank structure

$$\nabla_{W^{(l)}}\ell = \sum_{i=1}^k \left(\nabla_{h_i^{(l)}}\ell \right) \left(h_i^{(l-1)} \right)^T \quad (8.14)$$

as it is the sum of k rank-1 matrices obtained in the outer product of the layer's input and the gradient with respect to its output.

A similar, parameter-efficient parameterization exists for convolutional layers $h^{(l)} = W^{(l)} * h^{(l-1)}$. Its gradient $\nabla_{W^{(l)}}\ell$ can be written as

$$\nabla_{W^{(l)}}\ell = \sum_{i=1}^k \left(\nabla_{h_i^{(l)}}\ell \right) * \left(h_i^{(l-1)} \right). \quad (8.15)$$

In both cases, the gradient can be decomposed into a sum, where each summand corresponds to exactly one training example. We will use these summands to build explanations. In particular, since we can now attribute weight updates precisely to training examples, we can aggregate all weight updates grouped by training example. This way we can decompose the final weights into N components, where N is the number of training examples.

We have seen in the last section, that the largest computational bottleneck is the disk write speed. Equations (8.14) and (8.15) allow us to choose between two ways to store the per-example contributions and pick the one that requires less storage and hence less disk writes: We can either materialize the full weight updates or store the two factors and materialize the full weight updates later, either in a post-processing step or during the generation of explanations.

Deep learning frameworks like pytorch do not compute the per-example contributions, since they only need the batchwise aggregated gradient information. Consequently, tracking and logging these gradients requires additional work.

Remark 3. We have derived the method for vanilla stochastic gradient descent optimization. This is the simplest case, as the SGD optimizer is state-less, i.e. it does not maintain information about the variables it optimizes. Note, however, that we can handle optimizers that maintain information in the following manner: We first choose a linear function based on the state and the aggregated gradient, and then use this linear function to update the weights. Linearity allows us to decompose the weight update into per-example contributions. E.g. vanilla stochastic gradient descent always chooses the linear function

$$g(G) = -\eta G$$

while stochastic gradient descent with momentum also includes the running average $v^{(i-1)}$ that is stored in the optimizer’s state and uses

$$g(G) = -\beta v^{(i-1)} - \eta G.$$

For Adam, the linear function includes one multiplicative weight per entry in G . The multiplicative weights are based on the full aggregated gradient and are maintained in the optimizer’s state. In a way, this view is an over-simplification: The true update depends not only on the current batch of examples in G , but on all previous examples that informed the choice of $g(\cdot)$. We consider our approach an approximation that makes tracking the influence feasible.

8.4.2 Implementation Details

We have also implemented this method in the same pytorch library. We have summarized the necessary code changes in Figure 8.9. In contrast to the last Section, it requires the use of our specialized SGD optimizer. Additionally, we require an additional setup step that initializes routines that store the required forward and backward factors of equations (8.14) and (8.15) during the forward- and backward-computations. Internally, this is done using pytorch’s hook API. The `hookup` method installs the appropriate callbacks for logging the required tensors. Also it decides whether to store the weight updates in the factorized form or in the fully-materialized in order to minimize the disk write latency.

8.4.3 Experimental Evaluation

Cifar-10 Image Classification We again investigate our method for a popular image classification dataset *Cifar-10*. We train a convolutional neural network that uses the VGG-16 architecture. In contrast to the last Section, we employ a slimmed-down variant of the VGG-16 architecture that does not employ three large linear layers. Instead, the last convolution layer is directly mapped to 10 class logits. We train the model for 200 epochs employing standard data augmentations, randomized padded crops and rotations, and reduce the learning rate using the popular cosine-annealing schedule. This way our model achieves a test accuracy of 92%, which is in line with the best published results for VGG-16 networks trained only on *cifar-10* data.

We include an example classification in Figure 8.10. As before, we observe the same changing behaviour across layers, where the later layers have received very class-specific updates that are most responsible for the final representation and earlier layers see very mixed class influences. In this instance, we observe that representations of semantically related classes, e.g. *car* and *truck*, are on opposing ends of the cosine similarity diagram for later layers. This means that the later layers are responsible for separating the semantically similar classes and that during training, large updates were necessary to obtain this separation.

Comparison to other Methods We now compare the most-influential examples we compute to the most influential examples according to *TracIn*, the method proposed by

```

import torch
#from torch.optim import SGD
from xai_tracking.xai import WrappedSGD as SGD
from xai_tracking.xai import WrappedDataset

# train_set = CIFAR10()
train_set = WrappedDataset(CIFAR10())

train_loader = torch.utils.data.DataLoader(train_set, batch_size=512)

model = VGG16()

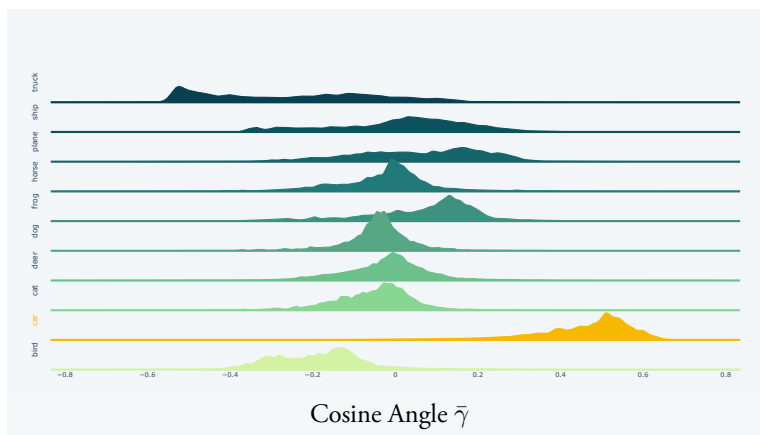
# optimizer = SGD(model.parameters(), lr=0.1)
optimizer = SGD(model.parameters(), lr=0.1, history_file="history/")
optimizer.hookup(model)
# [...]

for batch in train_loader:
    # inputs, labels = data
    inputs, labels, indices = data
    optimizer.zero_grad()
    loss = loss_function(model(inputs), labels)
    loss.backward()
    # optimizer.step()
    optimizer.step(ids=ind, labels=labels)

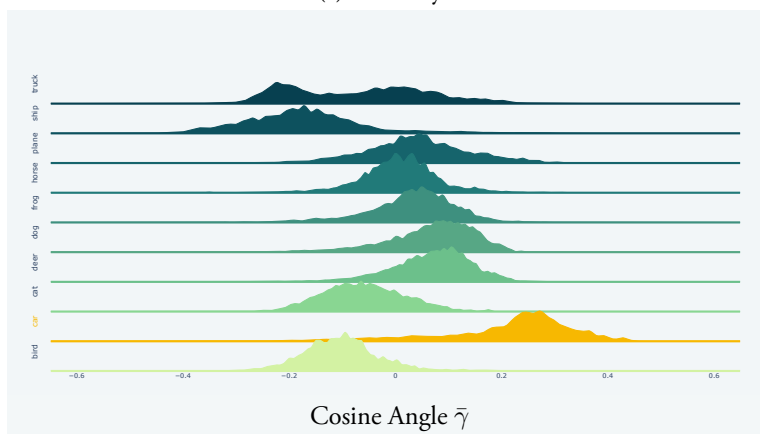
optimizer.done() #Wait for all Processes to finish, close open files.

```

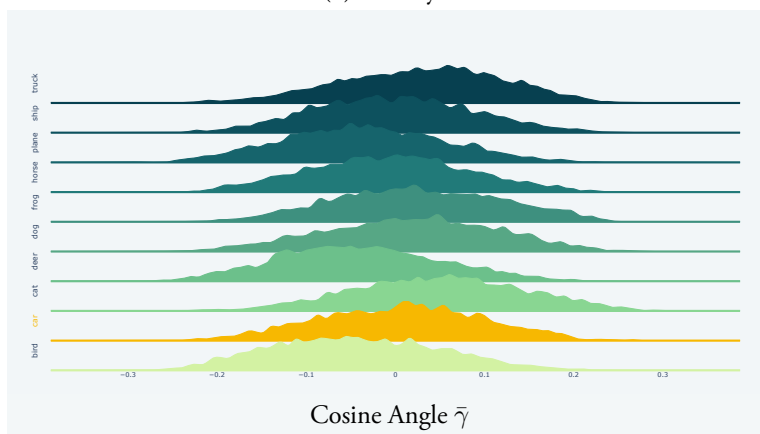
Figure 8.9: Sample Code for Capturing the Training Process of a VGG16 model trained with SGD on the CIFAR-10 dataset. Required changes are highlighted and are minimal. Indeed we do not need to change anything about the model, nor the way data is sampled.



(a) 12th Layer



(b) 6th Layer



(c) 1st Layer

Figure 8.10: Ridge Plot for the 3 layers of the VGG-16 network as computed using the examplewise gradient contributions aggregated across examples. The earlier layers are not as descriptive of individual classes, eg. the 1st layer.

Table 8.2: Computational Overhead Statistics for the Previous Experiments

Metric	examplewise	batchwise
Model Size	57MB	57MB
Number of Weight Updates	78,000	78,000
Runtime per Epoche	56sec	36s
Relative Overhead Runtime	+460%	+260%
Disk Storage Demand	11.0TB	6.0TB
Explanation Time	10min	66min

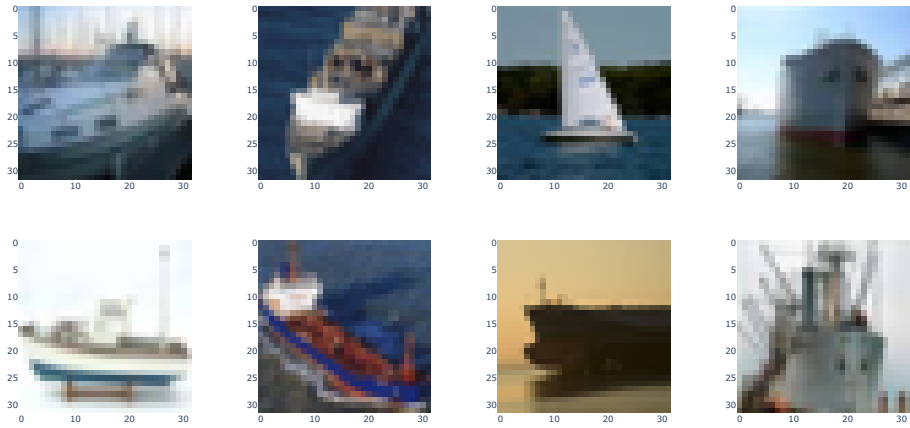
Garima et al. [81]. Both methods work by tracking the model throughout the training process, however Garima et al. only use the checkpoints to estimate the influence of individual examples on the final loss. We use the TracIn variant that only analyzes the final layer of the model and compare to the output that our method produces for the last layer. For our experiment, we use the same Cifar-10 model as described above, hence we have 200 checkpoints, one for every epoch. TracIn has to run inference on the full training dataset for each checkpoint, hence the runtime cost for generating explanations is higher than in our approach.

In Figure 8.11 we show an exemplary comparison of the two methods. This example is characteristic for the outputs we see and highlights an important difference between the approaches. TracIn tends to identify a set of visually similar examples as most influential, whereas we output a diverse set. Often, our outputs are examples that may be mistaken for another class, sometimes even mislabeled training examples, for instance two women labeled truck. This difference is explained by the approach they take. As we see in Eq. 8.1, the similarity is computed as the similarity of gradients of the model given the training input and the test input. These will be similar if the inputs are similar, as has been demonstrated by Charpiat et al. [43], who exploit this property to propose a similarity measure based on gradient dot-products on the final, trained model.

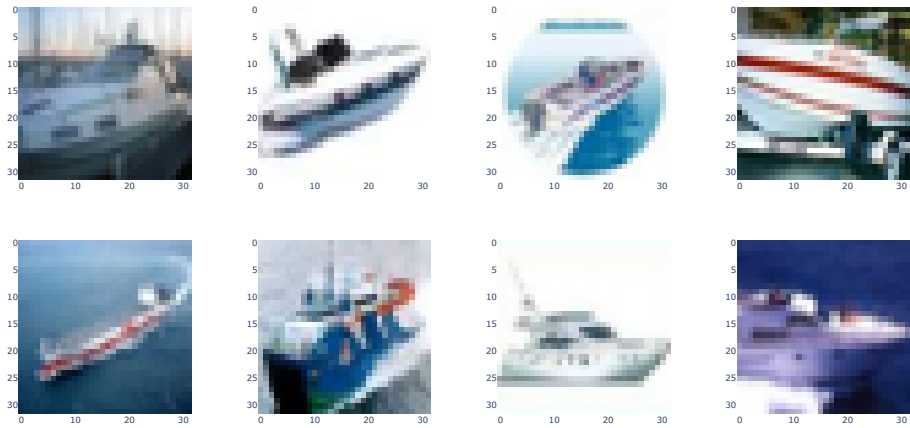
Similarly, another simple explanation method that outputs the training examples with the most similar latent representation produces visually appealing explanations that highlight what the model has learned, but not how and why the model learned. We want to find the examples, that were most influential for the final representation during training, not the examples that are most similar after training.

Another competitor are influence functions. Influence functions model the influence of training examples around the final model and cannot account for the training process that actually led to this model. We can view them as an idealization of training. Hence they too answer a different question.

Computational Overhead We repeat the batch-wise experiment from the last chapter using the same number of 200 epochs for training with a batch-size of 128. Furthermore we use the same, smaller VGG16 model taylorred to the cifar dataset. This



(a) Our Approach



(b) TracIn

Figure 8.11: Example Outputs for our Approach and TracIn for an instance of the class "Ship". Noteworthy, the two sets of top 7 most influential examples are distinct. We observe that TracIn produces more visually similar images, whereas our set captures a more diverse set of ship types.

way we obtain faster tracking of the weight updates for the batchwise variant, as the large fully-connected linear layers are no longer present. We can now fairly compare the computational overhead of both approaches: examplewise and batchwise tracking.

We see that tracking the examples on a per-example basis increases the storage demand from 6.0TB to 11.0TB. Using the low-rank factorization that naturally arises for the parameter updates in stochastic gradient optimization helped reduce this increase in comparison to the naive approach presented in the last Section. We have to track a number of weight updates that increased by a factor of batch-size, in this case 128. With that in mind, we see that the low-rank approach is very effective, getting the factor down from 128 to below 2.

The most important factor for reducing the explanation time, however, is the post-processing step that aggregates all weight updates by the training example that was involved. This reduces the number of sets of weights from 78,000, the number of weight updates, to 50,000, the number of training examples. Without this aggregation, we have to iterate about 1 million individual contribution as each example is used 200 times during training.

The additional callbacks employed in both forward- and backward-computation increase the runtime overhead in comparison to batchwise tracking of weight updates.

Keeping in mind that the modeling process involves training lots of models for hyperparameter tuning, etc. and we only need to track the training for few promising candidate models, this illustrates that applying the approach is actually feasible from a computational cost standpoint: Training some models longer as before is a plausible extra effort for obtaining insights into the model’s decisions, particularly for models we want to deploy.

8.5 Conclusion, Discussion and Future Research

We have proposed an addition to the pool of explanation methods. While many other methods explain, how the model computed a decision, our method asks why the model learned to classify this way. Hence it is meant to complement existing methods to provide evidence from the training set that decisions are sound. Our approach is extremely *simple*: We track all weight updates during training which allows us to decompose the hidden activations of neural networks into a sum over training steps. This way we can attribute the representation to individual training examples. We have proposed two types of *visual explanations*: One based on most-influential individual training instance, the other based on aggregated statistics over all training steps.

Our explanations do not attempt to “open the black box”, instead they find evidence for the decision in the training process. We can provide insights which training examples actually shaped the intermediate representations for a given example the most (Algorithm 3), and which classes influenced the intermediate representations most (Algorithm. 4).

Our method is *general*, it can either be wrapped around any optimization algorithm by switching the training protocol to our single-class mini-batching with ghost-

samples, or it can replace vanilla stochastic gradient training using standard mini-batching and standard models. This allows practitioners to integrate our approach into existing machine learning pipelines.

Using our approach, we can now answer the question “Why did the model learn to predict y ?” with an explanation like in Figure 8.1, optionally — depending on the target audience — equipped with a brief description of neural networks and their weight-decomposition (8.5). Unlike other approaches, the explanations are directly tied to the training *process*, not only to the final model [135], or the training data [289] or even nothing but the model architecture [3].

Our approach has one serious draw-back: It only works for multiclass-classification with a sufficiently large number of classes. Indeed, when we consider binary targets, any update that benefits the representation for classifying one class simultaneously benefits the other class. Hence, attributing the update to either class is pointless. More generally, each weight update is made to improve at the classification problem one vs. the rest. For problems with more classes, we this symmetry becomes weaker and weaker, for our example problems with 10 classes we find that the attribution of weight updates to classes is sufficiently meaningful.

Currently our approach only analyzes the linear maps in a feed forward network. While this includes many popular machine learning models like CNNs or graph neural networks, it does exclude two popular architecture choices: gated recurrent networks and transformer architectures with their self-attention layers. How to extend our approach to these models remains an open question.

We have investigated the use of low-rank representations for efficient storage of all weight updates. Further avenues for increasing the efficiency remain: Sketching and sampling are two promising, orthogonal approaches. While sketching reduces the dimensionality of the (vectorized) weight updates, for instance through multiplication with random matrices [268], sampling reduces the number of updates that we actually store. Coresets [74, 157] may provide a way to cleverly select which updates to write to disk in a streaming fashion, but the number of weight updates it needs to retain in a buffer may exceed the available memory for large models.

We think the combination of our approach with saliency approaches may be an interesting future endeavour. Since we can decompose pre-activations into a sum over groups, e.g. examples or classes, we may highlight the features in the input image that correspond to the group using these deconvolution techniques.

Chapter 9

How Robust is a Model?

MACHINE LEARNING models are under increasing scrutiny. In the last chapters, we have investigated how decisions of models emerge. We looked into what parts of the input influenced a decision to answer, how the decision was computed, and we looked into what parts of the training data influenced a decision to understand, why a model learned to decide the way it does. In this chapter, we investigate another aspect of model predictions: How can we change inputs to provoke errors? And how can we guarantee, that these attacks will not happen?

9.1 Introduction

We tackle the problem of adversarial robustness: Informally, small changes to the input of a model should not revise its decision. Our focus is on random forest models [29], arguably the most popular choice for classification tasks with tabular data [75].

Problem Statement Given a random forest f and an example x , the adversarial robustness is defined as the magnitude of the smallest perturbation to x that changes the predicted class:

$$\min_{\Delta} \|\Delta\|_p \text{ s.t. } f(x) \neq f(x + \Delta). \quad (9.1)$$

Solving this problem is NP-complete for random forests with more than one trees with depth larger than two [126]. Decision trees, however, can be verified with regard to adversarial robustness in linear time w.r.t the number of tree nodes. In this work we hence start by distilling random forest models into a single decision tree which allows efficient verification of random forests. We note that every random forest can be represented as a deeper decision tree, as both model families compute piecewise constant functions. However, a naive construction yields an exponentially deep tree, which is in line with the computational complexity of the problem. We suspect that real-world random forests do not make use of their exponential, worst-case complexity.

Thus we propose to distill the random forest into a deep – albeit not obsessively deep – decision tree using a greedy procedure in order to compute lower bounds.

We highlight the contributions of this Section. We

1. design a knowledge-distillation algorithm that approximates the output of a random forest model in a single decision tree,
2. derive a robustness verification algorithm for random forests that internally uses the distillation approach to compute robustness guarantees for the random forest based on guarantees of a distilled tree,
3. prove that our algorithm computes lower bounds on the robustness that hold on a user-specifiable fraction of the input space in Theorem 5 and Theorem 6, and
4. illustrate the usefulness of our new algorithm in an empirical evaluation.

The rest of this Section is structured as follows: We begin by reviewing related work in Section 10.2. In Section 10.3, we introduce the preliminaries necessary to understand our approach. Besides the mandatory introduction of notation, this includes our greedy, knowledge distillation algorithm for random forests and Chen et al.’s [45] adversarial robustness algorithm for decision trees. Then we are equipped to approach the main contribution of this paper in Section 10.4 and develop a robustness-verification algorithm based on distillation, which we put to an empirical test in Section 10.5. We conclude this paper in Section 10.6.

9.2 Related Work

Since the initial shock over the poor robustness of deep networks [254], a lot of methods for testing and for improving robustness have been proposed (see e.g. [114] for a survey). Our work is focussed on computing adversarial robustness for tree-based models.

Researchers have used logical solvers [25, 59, 69, 117, 231], SMT solvers [188], property checking [264, 265], abstract interpretation [218, 219], as well as mixed-integer linear programming solvers [126] and have rephrased the random forest verification problem in their respective input formats. Most of these approaches are limited to small-scale models and struggle with realistic numbers of trees and tree-depths. Interestingly, similar approaches have recently been used for deep learning as well, e.g. through SMT solvers [129] or mixed integer linear programming [32, 262]. Perhaps most relevant to our approach are the decision tree verification approaches of Chen et al. [45] and Wang et al. [280], so we outline them in adequate detail in Section 3. The most promising work for robustness verification of random forests is VERITAS proposed by Devos et al. [60]. It computes adversarial robustness based on A* search in a cleverly designed search space inspired by prior work of Chen et al. [45]. Like the approach presented in this work, they too present an anytime-algorithm that can output lower bounds at any time.

Another research direction is aimed at making decision trees and random forests more robust through modified training procedures [37,44]. Estimating the robustness and identifying adversarial examples is an important building block of these methods.

Knowledge distillation is the process of using a large model to train a smaller model in an effort to combine the benefits of an expressive, large model with the efficiency of a small model [94]. Our approach is tailored for the piecewise-constant functions computed by random forests. Thus it does not need access to data to perform distillation, unlike many approaches for deep networks; though there are approaches for deep networks that generate synthetic data points for distillation [290]. Many authors suggest using distillation to increase the robustness of small deep networks, e.g. [91]. From the theoretical perspective, distillation has also been used to derive generalization bounds [8, 112], where we can give tighter guarantees to the smaller models.

9.3 Preliminaries

We begin by introducing some notation. Let \mathcal{X} denote the input space of our examples $x \in \mathcal{X}$ and $f(x) \in [0, 1]$ denote the probability output of a random forest. In slight abuse of notation, $f(x) \neq y$ checks if the most likely class label is not $y \in \{0, 1\}$. We denote nodes in a decision tree with the symbols v or w . A node v in a tree f_i has children $\text{l_child}(v)$ and $\text{r_child}(v)$, an output f_i^v , and computes a split on a feature c with a threshold θ by checking if $x_c \leq \theta$. The set of leaf nodes of the decision tree f_i is denoted by $V(f_i)$. Note that a decision tree recursively partitions the data space into axis-aligned hyper-rectangles. We denote these partitions by $\Theta \subseteq [0, 1]^d$, in particular we denote by Θ^v the partition corresponding to a tree-node v .

Let $\mu(\cdot)$ denote the Lebesgue measure, hence $\mu(\Theta)$ computes the volume of the hyper-rectangle Θ .

We make a few assumptions. The first assumption is only technical:

Assumption 1. Our data domain \mathcal{X} is the d -dimensional hypercube $[0, 1]^d$.

We can always meet this assumption through appropriate normalization at pre-processing. In our experimental section we will use quantile-transformations.

Assumption 2. For a given $p \geq 1$, the l_p norm is a meaningful way to measure the magnitude of adversarial perturbations.

This assumption entails that the magnitude of perturbations can be compared across different features. This may be achieved through appropriate feature normalization, but usually requires domain knowledge.

Remark 4. Considering adversarial robustness without considering uncertainty estimation or probability calibration is an ill-posed endeavor. Obviously, every model has a decision boundary and examples closer to the decision boundary are likely less robust to adversarial perturbations. In the most extreme situation, we are evaluating the robustness using an example *on* the decision boundary, where an infinitesimal perturbation can change the output. Consequently, we cannot expect a model to be robust

for every output. We instead want our forests to be robust when the output exceeds a certain minimum certainty, which induces a margin around the decision boundary where we do not expect robustness. For this margin to be meaningful and interpretable, the model output should correspond to error probabilities, i.e. the model should be calibrated.

9.3.1 Decision Tree Distillation

First we show how we can distill a random forest classifier into a single decision tree. More formally, we solve the following problem:

Problem Statement Given a random forest for binary classification tasks, $f(x) = \sum_i h_i(x)$, approximate this ensemble of trees with a single, deep regression tree, such that the mean-squared error

$$\min_{\tilde{f}} \int_{\mathcal{X}} |f(x) - \tilde{f}(x)|_2^2 d\mu \quad (9.2)$$

of the decision tree \tilde{f} is minimized.

A decision tree computes a piecewise-constant function. So let us first consider the loss of a constant prediction $h \in \mathbb{R}$ on a partition Θ

$$\ell(\Theta) = \min_h \int_{x \in \Theta} |f(x) - h|^2 d\mu \quad (9.3)$$

$$= \min_h \underbrace{\int_{\Theta} f^2 d\mu}_{=: \text{var}(\Theta)} - h \cdot \underbrace{2 \sum_i \int_{\Theta} f_i(x) d\mu}_{=: \text{bias}(\Theta)} + h^2 \underbrace{\int_{\Theta} 1 d\mu}_{=: \text{vol}(\Theta)} \quad (9.4)$$

$$= \text{var}(\Theta) - \frac{\text{bias}(\Theta)^2}{4\text{vol}(\Theta)} \quad (9.5)$$

with minimizer $h^* = \frac{\text{bias}(\Theta)}{2\text{vol}(\Theta)}$.

We now inspect the quantities vol , bias and var individually. The quantity $\text{vol}(\Theta)$ is just the volume of the partition Θ that is efficiently computed because Θ is a hyper-rectangle with known lower and upper boundaries

$$\text{vol}(\Theta_i) = \prod_{i=1}^d |\sup \Theta_i - \inf \Theta_i|. \quad (9.6)$$

The computation of $\text{bias}(\Theta)$ decomposes over the trees in the random forest. Hence we only need to identify the nodes in the forest that fall into Θ and compute a weighted sum of their outputs.

$$\text{bias}(\Theta) := 2 \sum_i \int_{\Theta} f_i(x) d\mu \quad (9.7)$$

$$= 2 \sum_i \sum_{v \in V(f_i)} f_i^v \mu(\Theta^v \cap \Theta) \quad (9.8)$$

Finally, we consider $\text{var}(\Theta)$. In decision trees, this is easily computed: it is simply the weighted sum of all squared leaf-nodes. For the random forest model f , however, we have to do more work as we need pairwise intersections of the partitions induced by the individual trees which yields a worst-case quadratic explosion of the number of partition cells:

$$\text{var}(\Theta) := \int_{\Theta} f^2 = \sum_i \sum_{v \in V(f_i)} f_i^v \int_{x \in \Theta^v \cap \Theta} \sum_j f_j(x) \, d\mu \quad (9.9)$$

$$= \sum_i \sum_j \sum_{v \in V(f_i)} \sum_{w \in V(f_j)} f_i^v f_j^w(x) \mu(\Theta^v \cap \Theta^w \cap \Theta) \quad (9.10)$$

Now we show how to split a leaf node to improve the approximation. When evaluating splitting a node v based on feature c and split value θ , we consider the resulting sets of that split $\Theta_{x_c \leq \theta}^v$ and $\Theta_{x_c > \theta}^v$, and choose θ and c , such that the sum of the losses is optimal:

$$\arg \min_{\theta, c} \ell(\Theta_{x_c \leq \theta}^v) + \ell(\Theta_{x_c > \theta}^v) \quad (9.11)$$

$$= \arg \min_{\theta, c} \text{var}(\Theta^v) - \left(\frac{\text{bias}(\Theta_{\leq}^v)^2}{4\text{vol}(\Theta_{\leq}^v)} + \frac{\text{bias}(\Theta_{>}^v)^2}{4\text{vol}(\Theta_{>}^v)} \right) \quad (9.12)$$

$$= \arg \max_{\theta, c} \frac{\text{bias}(\Theta_{\leq}^v)^2}{4\text{vol}(\Theta_{\leq}^v)} + \frac{\text{bias}(\Theta_{>}^v)^2}{4\text{vol}(\Theta_{>}^v)}. \quad (9.13)$$

When maximizing over the decision threshold θ , we only have to consider the split values used in the random forest f . When varying θ , the volumes $|\Theta^v \cap \Theta|$ change only in a single dimension k . So this quantity can be efficiently computed. Furthermore we see that we do not need the costly computation of the variance to compute the best split.

Putting everything together, we can design a knowledge-distillation algorithm that approximates the decision boundary of a random forest using a decision tree. We use the standard, greedy decision tree induction algorithm [215], but instead of splitting according to a finite set of training points, we use the split criterion (9.13). If we build the distilled decision tree in a breadth-first traversal, as the number of its nodes approaches infinity, the output converges to the output of the original random forest model. A formal proof follows standard consistency proofs for decision trees [61]. As we have seen in the introduction, the random forest model f can be represented as an exponentially-deep decision tree. Hence at least one zero-loss solution is attainable. Furthermore, each new split does not increase the loss. In fact the loss is either decreased or the volume of the resulting leaf nodes is decreased. Infinite sequences of stagnating loss are only possible when zero loss is reached. This ensures convergence.

In practice, we can stop at a leaf node once the loss falls below a desired level or once the number of nodes exceeds a tolerable level.

Remark 5. We limit the derivation of this approach to binary classification where the output of the random forest is univariate. However, we note that all of the above derivations can be carried out in the multi-class case where the output of the random-forest is a categorical probability distribution and we use the sum of squared distances over all classes to measure the goodness-of-fit if the distilled model.

9.3.2 Decision Tree Verification

Chen et al. [45] show that we can compute the l_∞ adversarial robustness of decision trees in $\mathcal{O}(v)$ operations, where v is the number of nodes in the tree. Wang et al. [280] present a generalization of their algorithm to analyze the robustness perturbation with respect to any l_p norm. The idea is simple: For a given example and every leaf node of the tree, we compute the smallest perturbation that lands the example in the leaf node recursively in a top-down fashion: At each split, we know the smallest perturbation to the split feature that puts the example on the wrong side of the split. When we processed all the leaf-nodes, we find the leaf node that gives a wrong output with the smallest perturbation, which gives us our exact robustness. We can evaluate this robustness for a whole set of examples in parallel, giving us either a mean-robustness or a minimum robustness. This approach is summarized in Algorithm 6.

Algorithm 6 Decision Tree Verification [45]

```

1: Input: tree  $h$ , example  $x$ 
2:  $A_v = \vec{0}$  for all  $v \in \text{nodes}(h)$  ▷ initialize perturbations
3:  $Q = [\text{root}(h)]$  ▷ depth-first-search
4: while  $Q \neq \emptyset$  do
5:    $v = Q.\text{pop}()$  ▷  $v$  with split-feature  $c$ , threshold  $\theta$ 
6:   if  $\neg \text{is\_leaf}(v)$  then
7:      $A_{\text{l\_child}(v)} = A_{\text{r\_child}(v)} = A_v$  ▷ clone
8:     if  $x_c \leq \theta$  then
9:        $A_{\text{r\_child}(v),c} = \arg \max |y|, y \in \{\theta - x_c, A_{v,c}\}$ 
10:    else
11:       $A_{\text{l\_child}(v),c} = \arg \max |y|, y \in \{\theta - x_c, A_{v,c}\}$ 
12:    end if
13:     $Q = [\text{l\_child}(v), \text{r\_child}(v)] + Q$ 
14:  end if
15: end while
16: return  $\min\{\|A_v\|_p \mid v \in V(h) \text{ s.t. } h^v \neq h(x)\}$ 

```

9.4 Random Forest Verification Through Knowledge Distillation

We have seen that both the distillation and the verification proceed by traversing the tree: the former by splitting nodes to improve the fit, the latter by refining the adversarial perturbations. We can now design an algorithm that combines the two in order to compute adversarial robustness of random forests. In contrast to decision tree verification, now we have to consider the quality of our tree-approximation: Areas with poor approximation may still contain adversarial perturbations, even though the approximated output looks correct.

We proceed in two steps: First, we show how we can determine whether a leaf node in the distilled tree approximates the random forest sufficiently well. Second, based on that, we devise a strategy for selecting a leaf node to split further to improve the approximation. Putting everything together, we devise an anytime algorithm that approximates the adversarial robustness of a random forest and prove guarantees.

9.4.1 Bounding the Output Range of Random Forests

We need to bound the range of function values that the random forest model outputs in a given partition Θ^v . Solving this exactly is equally expensive as bounding its adversarial robustness [60]. Unlike VERITAS [60], we only compute measure-theoretic bounds using exactly computed moments, but we can control the confidence of these bounds using a user-parameter. Chebyshev's inequality states that

$$\mu(\{x \in \Theta \mid f(x) > t\}) \leq \frac{1}{t^k} \int_{\Theta} |f(x)|^k d\mu. \quad (9.14)$$

Hence we can bound the deviation around our approximated output using the second moment of the absolute difference, i.e. the loss

$$\begin{aligned} \mu(\{x \in \Theta^v \mid |f(x) - h_v| > t\}) &\leq \frac{1}{t^2} \int_{\Theta^v} |f(x) - h_v|^2 d\mu \\ &= \frac{\ell(\Theta)}{t^2} \end{aligned} \quad (9.15)$$

or equivalently

$$\mu\left(\left\{x \in \Theta^v \mid |f(x) - h_v| > \sqrt{\frac{\ell(\Theta)}{p}}\right\}\right) \leq p. \quad (9.16)$$

Consequently, we can bound the volume of the input space where the random forest deviates by more than a tolerated amount. As the loss gets lower, the bounds get tighter. Particularly, we are interested in deviations larger than $|h_v - 0.5|$ which lead to inverted classifications. Our approach will allow the user to control the measure of space where function values can fall out of the estimated range.

Remark 6. We have seen above that computing the second moment of f , the variance $\text{var}(\Theta)$, requires runtime that is quadratic in the number of random forest leaf

nodes in Θ . It is possible to use Chebyshev inequalities of higher, even orders or other concentration inequalities that use higher moments like [101] in order to get tighter confidence intervals. Note, however, that computing the k -th moment has runtime exponential in k . For the 4th moment, this is already infeasible for large random forests.

9.4.2 Simultaneous Distillation and Verification

Starting with an empty tree, we incrementally distill the random forest into a decision tree for verification. Let \tilde{f} denote our current, in-progress decision tree. For each leaf node $v \in V(\tilde{f})$, we maintain the smallest perturbation that puts the given example x in it (as in Algorithm 6). Depending on the estimated range of outputs of the original random forest f , we consider them adversarial or benign. This marks the difference to plain decision tree verification, as now we do not use the leaf outputs directly, but have to account for their uncertainty. If we approximate the forest perfectly, then we compute exact lower bounds as Algorithm 6. When we have to assume that the leaf node with the smallest perturbation contains adversarial permutations judging by the confidence interval, but really the node is benign, we only compute a lower bound on the true adversarial robustness. In our concentration inequality approach, the confidence intervals do not hold strictly, but only on the majority of space. Hence it is possible that the bounds we compute do not hold. We bound the effect of these failures in Theorems 5 and 6.

Our algorithm can now choose a leaf node to further split according to (9.13). This will result in two new leaf nodes, where one of the leaf nodes inherits the parent’s perturbation and the other potentially has a larger perturbation depending on the evaluation of the arg max operation in lines 9 or 11 of Alg. 6. This illustrates a way to increase the lower bound estimates of the adversarial robustness: We reduce the loss, such that our confidence interval of random forest outputs (9.16) becomes sufficiently small to become benign. Meanwhile, the other permutation of the other child increases in magnitude.

We maintain an ordered queue of all the leaf nodes who are potential adversarial perturbations, ordered by their norm, and always split the node with the smallest norm.

Now we consider how to select the confidence intervals used for deciding if a perturbation is adversarial or benign: Let $\delta \in (0, 1)$ denote a user-specified tolerance parameter we refer to as *budget*. It controls what fraction of the input space is left unverified during the search for adversarial perturbations. We propose two variants:

Average Variant In this variant, each leaf node can use up an amount of the budget proportional to its own volume. Consequently, at node v we estimate the confidence interval

$$f(x) \in \left(h_v - \sqrt{\frac{\ell(\Theta^v)}{\delta \cdot \text{vol}(v)}}, h_v + \sqrt{\frac{\ell(\Theta^v)}{\delta \cdot \text{vol}(v)}} \right) \quad (9.17)$$

which, by (9.16) is potentially violated only on a subset of the input space with measure bounded by $\delta \cdot \text{vol}(v)$. Taking the union over the violated areas of all leaf nodes, we obtain that our confidence intervals fail on a subset bounded in measure by δ .

Greedy Variant The smallest adversarial perturbations for an example x will be, by definition, in its close proximity. Consequently, large parts of the input space are irrelevant for the verification – spending our budget in those regions is wasteful. Instead we propose to spend the budget greedily on the most promising candidate leaves v for adversarial perturbations. Using all of the budget equals the confidence interval

$$f(x) \in \left(h_v - \sqrt{\frac{\ell(\Theta^v)}{\delta}}, h_v + \sqrt{\frac{\ell(\Theta^v)}{\delta}} \right) \quad (9.18)$$

Since we can only use the budget once, we have to reduce the remaining amount. Let $m = |h_v - 1/2|$ be the margin of h_v : the largest deviation we need to worry about. From (9.15) we can bound the measure of the space where the deviation is larger than m . If the budget permits this, we can verify v and proceed with reduced budget

$$\delta' = \delta - \frac{\ell(\Theta^v)}{m^2}. \quad (9.19)$$

A potential drawback of the greedy approach is that we can miss small perturbations entirely if their corresponding leaf nodes have less volume than the remaining budget. Future work should look into overhauling the budget parameter to be relative rather than our proposed absolute budget.

In the following, we prove the main property of our random forest verification algorithm. For simplicity, we begin by proving the average variant.

Theorem 5. *Let f be a given random forest model and x be an example. Let $\delta \in (0, 1)$. Then the average variant of Algorithm 7 that uses the confidence intervals (9.17), computes a lower bound for the l_p -adversarial-robustness that holds on a subset with at least $(1 - \delta)$ measure: Let r be the output of Algorithm 7. Then there exists $\tilde{\mathcal{X}} \subseteq \mathcal{X}$ with $\mu(\tilde{\mathcal{X}}) \geq (1 - \delta)$ such that*

$$r \leq \min_{\Delta} \|\Delta\|_p \text{ s.t. } x + \Delta \in \tilde{\mathcal{X}} \text{ and } f(x) \neq f(x + \Delta).$$

Proof. Let $C \in \mathbb{N}$. Let \tilde{f} denote the decision tree at the end of Algorithm 7 after running with parameters f, x, δ, C . Let \tilde{r} denote the output of Algorithm 7. We proceed in two steps: First we construct $\tilde{\mathcal{X}}$, then we prove that we compute a lower bound for the adversarial robustness for all adversarial examples in $\tilde{\mathcal{X}}$.

Consider a leaf $v \in V(\tilde{f})$. Let

$$\tilde{\mathcal{X}}^v := \left\{ x \in \Theta^v \mid |f(x) - \tilde{f}^v| \leq \sqrt{\frac{\ell(\Theta^v)}{\delta \text{vol}(\Theta^v)}} \right\}$$

denote the set of all points in Θ^v where the random forest output falls inside our assumed confidence interval. By Chebyshev inequality (9.15), we have

$$\mu(\Theta^v) \geq \text{vol} - \delta \text{vol}(\Theta^v) = \text{vol}(1 - \delta).$$

Algorithm 7 Random Forest Verification (**this chapter**)

```

1: Input: forest  $f$ , example  $x$ , budget  $\delta$ , max. size  $C$ 
2: root = root node of new empty tree  $\tilde{f}$ 
3:  $A_{\text{root}} = \vec{0}$  ▷ initialize perturbation
4:  $Q = [(0, \text{root})]$  ▷ priority queue
5: while  $|\tilde{f}| < C$  and  $Q \neq \emptyset$  do
6:    $v = Q.\text{pop}()$  ▷ leaf with smallest adv. perturbation
7:   if greedy and  $1/2 \notin \delta$ -interval  $c_\delta$  then
8:      $\delta = \delta - \frac{\ell(\Theta_v)}{|1/2 - \tilde{f}_v|^2}$  ▷ (9.19)
9:     continue
10:  end if
11:   $c, \theta =$  best split of  $v$  according to (9.13)
12:  create children  $l, r$  of  $v$  according to  $c$  and  $\theta$ .
13:   $c_l, c_r =$  confidence intervals (9.17) or (9.18) for  $l$  and  $r$ 
14:   $A_l = A_v, A_r = A_v$  ▷ clone permutations
15:  if  $x_f \leq \theta$  then
16:     $A_{l,f} = \arg \max |y|, y \in \{\theta - x_f, A_{v,f}\}$ 
17:  else
18:     $A_{r,f} = \arg \max |y|, y \in \{\theta - x_f, A_{v,f}\}$ 
19:  end if
20:  if  $1/2 \in c_l$  or  $f(x) \neq \tilde{f}^l$  then
21:     $Q.\text{insert}(\|A_{l,f}\|_p, l)$ 
22:  end if
23:  if  $1/2 \in c_r$  or  $f(x) \neq \tilde{f}^r$  then
24:     $Q.\text{insert}(\|A_{r,f}\|_p, r)$ 
25:  end if
26: end while
27: return  $\min\{p \mid (p, q) \in Q\}$  ▷ smallest adversarial leaf

```

The leafs partition $\mathcal{X} = \cup_{v \in V(\tilde{f})} \Theta^v$ with $\Theta^v \cap \Theta^w = \emptyset$ for $v \neq w$. Hence

$$\mu(\cup_{v \in V(\tilde{f})} \tilde{\mathcal{X}}^v) \geq (1 - \delta) \sum_{v \in V(\tilde{f})} \text{vol}(v) = 1 - \delta.$$

Now we proceed to show that we compute lower bounds constrained to the set $\tilde{\mathcal{X}}$. We assume there is an adversarial permutation Δ with $x + \Delta \in \tilde{\mathcal{X}}$ with $\|\Delta\|_p < r$ and $f(x + \Delta) \neq f(x)$. Consider the leaf node v with $x + \Delta \in \Theta^v$. We know that

$$f(x + \Delta) \in \left(h_v - \sqrt{\frac{\ell(\Theta^v)}{\delta \cdot \text{vol}(v)}}, h_v + \sqrt{\frac{\ell(\Theta^v)}{\delta \cdot \text{vol}(v)}} \right)$$

because $x + \Delta \in \tilde{\mathcal{X}}$. Since v was not in the priority Q , it holds that $1/2 \notin c_v$ and $f(x) = \tilde{f}^v$. Since $1/2 \notin c_v$, we know that $\tilde{f}^v = f(x + \Delta)$. This contradicts the

assumption that $f(x + \Delta) \not\approx f(x)$. We can conclude that no such Δ can exist, all adversarial permutations in \mathcal{X} must have robustness greater or equal to our output r . \square

Theorem 6. *Under the assumptions of Theorem 5, the greedy variant of Algorithm 2 provides the same guarantees as the average variant in Theorem 5.*

Proof-Sketch. The proof follows the proof of Theorem 5. However, in constructing the set $\tilde{\mathcal{X}}$, we only consider the nodes v with permutations smaller than the final output. For all the other nodes, we take $\tilde{\mathcal{X}}^v = \Theta^v$ because range errors in nodes with larger permutation have no effect. By the greedy construction, the sum of measures is larger than $1 - \delta$ in this case as well. The second part of the proof applies without modification. \square

9.4.3 Some Comments on Asymptotic Runtime

Let us investigate the computational cost of running the proposed method. The first important variable is the number of desired nodes C in the distilled model, which determines the number of times a node is split.

This node-splitting is the most computationally expensive. Each node in the distilled tree is associated with a set V of leaf-nodes of the original forest. Computing the distillation loss has complexity $\mathcal{O}(|V|^2)$ and computing the best split has complexity $\mathcal{O}(|V|d)$. However, estimating a total runtime of $\mathcal{O}(C(|V_{\text{root}}|^2 + |V_{\text{root}}|d))$ using all leaf nodes of the random forest is a massive exaggeration, as the number of leaf nodes is reduced with each split, in the best case by a factor of two. In practice, we see that the first splits are by far the most expensive ones; further down the distilled tree the sets of leaf nodes decrease in size rapidly. We cannot estimate how balanced the splits are beforehand, which makes an asymptotic analysis difficult without further assumptions. If we, for illustration purposes, assume perfectly balanced splits, this yields a complexity of

$$\mathcal{O}(|V_{\text{root}}|^2 + |V_{\text{root}}|d \log C)$$

using standard divide-and-conquer runtime analysis. Again, this is *not* a formal analysis as we do not know how balanced the splits are.

9.4.4 Implementation Details

We outline a few optional implementation tricks that we used during the development. For computational efficiency, we do not verify individual examples, but build a joint distilled decision tree for all examples of the dataset we wish to verify. For simplicity, we keep priority queues for all examples separately and use a round-robin strategy for selecting the next split node.

Since the algorithm does not require us to start with an empty tree, to speed up the construction, we can initialize it with a decision tree fitted to the random forest output on a set of examples. Then we can put all the leaf nodes into the priority queue and

Table 9.1: Comparison of the Adversarial Robustness Values Across Datasets and Methods

METHOD	ADULT			MAGIC			HIGGS			MOZILLA		
	hist	min	mean	hist	min	mean	hist	min	mean	hist	min	mean
MILP		0.0493	0.4814		0.0030	0.0965		0.0157	0.0972		0.0012	0.2036
VERITAS		0.0491	0.4811		0.0029	0.0955		0.0154	0.0952		0.0010	0.2036
Distill-AVG		0.0492	0.3441		0.0017	0.0965		0.0061	0.0883		0.0008	0.2089
Distill-GRD		0.0492	0.3356		0.0040	0.0845		0.0008	0.0687		0.0011	0.2172

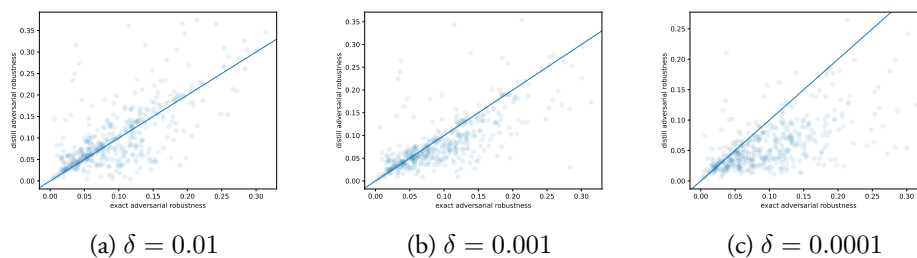


Figure 9.1: Our Lower Bounds (DISTILLGRD) vs. the Ground Truth. Points over the diagonal are outputs that are not actually lower bounds. We see that with lower budget values δ , the number of these failures decreases.

proceed as before. We can determine that we have fully processed an example when the smallest permutation indeed changes the outcome of the original model. Refining the distilled model any further is unnecessary and we can terminate early.

We make our Python implementation available at Github: https://github.com/Whadup/random_forest_robustness. Future work should implement our approach in a compiled language to achieve lower runtimes.

9.5 Experimental Evaluation

We empirically verify the usefulness of our approach in this experimental section.

9.5.1 Setup

We begin our experimental section with a discussion of the used datasets, models as well as methods and their respective hyperparameters.

Datasets and Models We limit our analysis to four standard, tabular datasets, which is the strong-suit of tree-based models. We analyze standard random forest models with 100 trees and a depth limited to 8 layers. We found these models to be competitive on all datasets considered. Particularly, increasing the number of ensemble members or the depth has shown diminishing returns on the classification performance. The models are trained on all but 500 examples of the data sets, we evaluate the robustness

on the holdout data. As outlined in Remark 1, we only verify examples where the models outputs confidences larger than 0.6. We use `sklearn` for fitting the random forests.

Methods We analyze the two variants of our approach: Either we spend the budget evenly across the input space (DISTILL-AVG) or we spend it greedily (DISTILL-GRD). We compare our method with the output of an exact solver based on Mixed-Integer linear programming [126] (MILP) and with the VERITAS algorithm that produces lower bounds. We exclude MERGE [45], that like VERITAS produces lower bounds, but has been shown to consistently produce weaker bounds in more time compared to VERITAS [60]. We compute adversarial robustness with respect to the infimum-norm $\|\cdot\|_\infty$ as it is the standard in robustness literature, but note again that our approach supports any l_p norm.

Hyperparameters Unless noted differently, for our experiments in this section we set the budget value δ to 0.01 for DISTILL-AVG and to 0.001 for DISTILL-GRD. We limit the number of nodes in our distilled trees to 100,000.

9.5.2 Results

We compare the computed robustness scores of the different methods on our datasets. The results are summarized in Table 9.1. Looking at the aggregated statistics, we see that our approach captures the real robustness properties of the models quite well. Particularly the sparklines in Table 1 indicate that our approach captures the distribution of robustness scores well. This highlights the usefulness for the particular use-case of model selection where we want to select a model from a set of candidates that fulfills certain robustness requirements.

We see that sometimes we compute a higher mean robustness than the exact method. Obviously that is not a desired outcome. But we have explicitly designed the algorithm such that it can fail to verify parts of the input space by using up the given budget δ . Next we analyze these failures in more detail. In Figure 9.1 we compare the exact robustness values with the bounds computed by our approach for DISTILLGRD. All budget values show a high correlation between our result and the exact value. As expected, for higher budget values we output invalid lower bounds more often. These violations get less frequent with higher budgets, but at the cost of looser lower bounds. We quantify these effects in Table 9.2 and note that the percentage of invalid lower bound drops from 65% to 18% when we limit the budget. A similar effect happens to the magnitude of violated bounds which drops from 0.029 to 0.017. A possible remedy for improving the quality of the lower bounds for large δ values is the increasing the size of the distilled tree.

We note that our python implementation runs orders of magnitude faster than the exact MILP approach that uses the Gurobi library for solving the mixed-integer linear programs. The runtime of VERITAS and DISTILL are roughly the same, but a fair comparison is not possible: While VERITAS is implemented in a compiled language, but

Table 9.2: Quantitive Analysis of the Quality of our Bounds for the MAGIC dataset. Let r denote the exact robustness and \tilde{r} denote our estimate of DISTILLGRD.

Metric	$\delta = 0.01$	$\delta = 0.001$	$\delta = 0.001$
$\#\tilde{r} > r$ in %	64.81%	38.53 %	17.59 %
mean $ \tilde{r} - r $	0.0329	0.0285	0.0451
mean $ \tilde{r} - r , \tilde{r} > r$	0.0285	0.0214	0.0172
mean $ \tilde{r} - r , \tilde{r} \leq r$	0.0410	0.0329	0.0510
$ \tilde{r} - r $			
mean \tilde{r}/r	1.2736	1.0622	0.7548

runs single threaded, our approach uses multiprocessing to verify the set of examples in parallel, but is implemented in Python. We work toward a fairer comparison by reimplementing our approach in a compiled language as well.

9.6 Conclusion

In this work, we have introduced a novel algorithm for robustness-verification of random forest models. It applies a novel algorithmic paradigm: verification through distillation, that gives an approximate guarantee. Unlike existing approaches that use combinatorial techniques to estimate the range of outputs of a random forest, our approach relies on numerical techniques, particularly the exact estimation of moments and the Chebyshev inequality. We have designed a distillation algorithm that approximates random forests in a single decision tree. Our approach does not need access to a dataset to perform the distillation, but relies on the piecewise-constant output of the random forest which allows exact computation of the loss over the full range of inputs. Then we proposed a verification algorithm that constructs and uses this distilled tree internally. We formally prove that our approach succeeds in verifying a large fraction of the input space and that the tightness of this bound can be controlled by the user. Our empirical evaluation shows that our approach computes useful bounds and that the budget parameter fulfills its intended purpose. We have proposed two simple ways to manage this budget, but future work could look into more refined ways to use that budget during the verification.

Part IV

Conclusion

Chapter 10

Conclusion and Outlook

After climbing a great hill, one only finds that there are many more hills to climb.

Nelson Mandela

This thesis has tackled problems along the full machine learning pipeline. We have investigated how to design task when data does not come neatly labeled and ready-to-use. Then we looked into how to establish trust in our models through inspection. We now conclude by summarizing again the contributions and giving an outlook into future research directions. We structure this conclusion into two parts corresponding to the Parts II and III of this thesis.

Designing Machine Learning Tasks

At the core of this part of the thesis, we have designed machine learning tasks for data-driven training of prediction- and embedding-models.

For the first time, we have designed large embedding models for mathematical expressions and applied them for the application of retrieving related mathematical expressions. We have compiled a large dataset of mathematical expressions for self-supervised pretraining of models based on scientific articles obtained at arxiv.org. Additionally, we have designed a number of evaluation tasks to measure the quality of our models. We investigated three different ways to represent the input data: as bitmap images, as sequences and as trees and designed models for each data modality. Our best results are reported for a graph neural network model that is trained on the self-supervised masking task in conjunction with a contextual similarity task that exploits the structure of papers. We have used data augmentation to encode invariance against symbol substitution or renaming into our training tasks.

Since we published our work, pretraining deep networks for mathematics using datasets derived from arXiv.org [150, 204, 216] or stackexchange.org [150] has took off and shown merit in many tasks relating to mathematical expressions. Other works

use synthetic formula generators to automatically generate training data for a machine learning system that solves particular mathematical tasks like solving differential equations, integrals or equalities [143, 200, 232, 282]. Future work should further bridge the gap between exact symbolic manipulation of formulas required for these tasks and neural similarity estimates for the non-standardized formulae found in scientific publications we investigate. To this end, symbolic mathematic software could act as a source of supervision signals where we generate pairs of formulas through symbolic manipulation of inputs [282] or parts of the input. Difficulties for applying symbolic manipulation to formulas found in scientific literature include inferring the type of symbols (scalar, vector, function, etc.) and non-standardized notations for operations (e.g. different product symbols for dot products). We may also use symbolic solvers as parts of neural models, as research on backpropagating through discrete units progresses [189].

Furthermore, we have developed a method for training multitask networks in the domain of astrophysics. We have replaced the prediction pipeline based on manual feature engineering and separate prediction models with a multitask network that uses convolution layers to automatically learn features from camera inputs. These features are shared between all prediction tasks. Our models outperform existing models in both prediction performance on simulation data, as well as source detection in real world data. We assume that deep learning models will be applied in the next generation of gamma ray telescopes, and our work is another piece of evidence for the superiority in prediction performance for event tagging in Cherenkov telescopes [118–120, 190, 243].

Finally, we reframed the gamma hadron classification problem as a noisy label learning problem and have demonstrated how this allows us to use real telescope recordings rather than simulation data to train the classifier. In the process, we have designed a more-general learning algorithm for training models in the presence of class-conditional label noise. In contrast to existing work, we exploit knowledge of one of the two label noise probabilities, a situation that we found in the gamma-hadron separation use-case. Finally we demonstrate that a multitask deep network trained with simulation data first and with noisy-label, real-world data second further improves the source detection performance. In summary, we believe that using noisy labels based on real telescope recordings can pave the way to making gamma ray astrophysics even more data-driven. Future research should explore the use of real-world data for the remaining prediction tasks: origin estimation and energy estimation. We have seen that deep networks trained with our noisy label, real-world recordings currently require pre-training on simulation data to beat the models trained only on simulation data. We attributed this to the small number of available real-world data points. Nonetheless future work should investigate how to train representations only on real-world data, possibly through the use of unsupervised representation learning approaches.

Inspecting Machine Learning Models

Additionally, we have designed methods to manually inspect machine learning models.

We have derived a heatmap approach for explaining and visualizing similarity computations between two trees embedded by a graph neural network. We have proposed two variants: one based on the forward pass, and the second one based on the backward pass of the graph neural network. We have qualitatively evaluated this approach in a search engine setting and seen that it helps to understand which symbols in a tree are responsible for the computed similarity score.

Existing XAI approaches, including our approach for graph neural networks, often present descriptions of how the prediction was computed, and, depending on the method used, these descriptions can be more or less faithful and easier or harder to understand. Instead, we propose methods to provide explanations that describe how the network has learned to predict. These methods work by tracking the full training process, that means storing all weight updates, in order to analyze which examples or which classes of examples have had the highest influence on the final decisions. Our method works on the level of hidden representations that are learned, as we decompose each linear projection in a neural network into summands that correspond to training examples.

Our methods present visual explanations to the practitioners. These may not yet be as easily digestible as we want. Future work should focus on providing explanations that are easier to grasp. For instance, we currently visualize each layer in the network individually. While this fine-grained analysis may be helpful, looking at many layers in deep networks can be overwhelming. We can look into ways to aggregate the information over different layers or automatically select the most informative visualizations. Another interesting avenue is structuring the explanations according to the training process: We can track how the prediction for a given example changes over the course of the training process. We can identify the training steps where the predicted class changes and provide explanations that provide information on the influential examples that provoked these changes. We already store the full training process, so all required information is available.

Finally, we have proposed a novel method for robustness verification for large random forest models. We present a new idea how to combine knowledge distillation and robustness verification that allows us to provide approximate robustness guarantees with a user-controllable tradeoff between runtime complexity and tightness of the bounds. Future work can look into using the distillation approach for verification of other model classes. Going from the class of piecewise-constant functions of random forests to piecewise-linear functions of deep ReLU-networks seems like a promising research direction to establish verification through distillation in deep learning.

Chapter 11

Bibliographical Remarks

Chapter 4 is based on number of works. Parts of the chapter are published work in [211], where standard convolutional networks were used to embed images of formulas. The first iteration of code for the data pre-processing was written by a group of students as part of a undergraduate project. Jonathan Schill supported this work by helping with the experiments with convolutional networks. After that, the convolutional networks were replaced with graph convolutional networks that used a MathML representation, parts of that section were published in [209]. In joint work with Stefan Todorinski transformer models for sequential MathML representations were considered. His master thesis provided the code for training these models [263].

The work on multitask supervision in Chapter 5 is based on preliminary results with co-authors Sebastian Buschjäger, Jens Buß, Wolfgang Rohde and Katharina Morik [35]. Sebastian Buschjäger and I implemented most of the code for the experiments. He was responsible for studying the inference times on different hardware platforms, including FPGA, a research direction that was not part of this section. I ran the experiments, optimized hyperparameters and did the real-world analysis on significance of detection of Crab nebula. Jens Buss and Maximilian Nöthe advised on astroparticle physics in general and on the details of the FACT telescope in particular, and helped us with the data processing as well as the evaluation. Katharina Morik and Wolfgang Rhode helped writing the paper. The section published in this thesis shows significant progress with respect to the aforementioned results [35]. Instead of using VGG-style convolutional networks to solve only the gamma-hadron classification problem, I have extended the model to the multitask model based on EfficientNet models with its joint objective that we have seen in this section. Most of the code has been rewritten, data-preprocessing was adapted and all experiments were overhauled.

The results on noisy label learning in Chapter 5 are based on joint work with Mirko Bunse that is currently under review and available as a preprint [206]. The research is based on my initial idea of using the Li&Ma significance as a learning criterion for training machine learning models, that was empirically shown to work well. In joint discussions we established that the on- and off-position features serve as noisy labels in the approach, more specifically labels with class-conditional noise. Mirko Bunse

established baselines using other noisy-label-learning approaches that work by tuning the decision threshold of models and reframed the Li&Ma significance as a method for tuning the decision threshold. I designed and implemented the experiments on source detection with deep networks and noisy supervision.

Chapter 7 was originally published as a workshop paper co-authored with Jan Richter [210]. The methods evaluated in this chapter, namely the forward- and backward-visualization, were proposed by me and implemented by Jan Richter in his function as student assistant.

Chapter 12.2 is based on a tech report written with Alina Timmermann [212], who helped with the literature review on reproducibility.

Part V

Appendix

Chapter 12

Open Source Software

THIS chapter gives an overview over software developed or contributed-to in the course of this thesis. The first three libraries presented handle machine learning computing infrastructure as well as reproducible execution of machine learning experiments, an important aspect when scientifically approaching artificial intelligence and data science. After that two libraries for working with formula on arxiv are presented: one for preprocessing and extracting the data and another for machine learning experiments using the preprocessed data. Finally, we present a library for kernel learning on GPUs with stochastic gradient descent and a library for explaining deep networks.

12.1 malocher

»<https://github.com/Whadup/malocher>«

“*Malocher*” is a German colloquialism from the Ruhr-Area for “worker”, particularly used for miners and steel workers.

Malocher is a lightweight python library for running jobs on a cluster where nodes are accessed via SSH and share a common network storage like traditional NFS or mountable cloud storage, as was the case at the Artificial Intelligence group at TU Dortmund university. It

- uses SSH and `paramiko` for communication between workers,
- relies on `dill` for serializing python code and data to a shared filesystem and
- assumes that all python libraries and interpreters are available on all nodes, like when they are also in the NFS.

This way we do not need to use large cluster computing libraries, e.g. from the Apache universe, or bug-infested, hardly-documented libraries from the Python universe like `ray` or `dask`.

12.1.1 Installation

The simplest way to install malocher is to use pip

```
pip install git+https://github.com/Whadup/malocher
```

12.1.2 Setting up the Malocher-Workers

Setting up worker nodes for malocher is very easy. Each node has to fulfil the following requirements:

- Make sure you can access each malocher node from the supervising node using the same SSH key `ssh_private_key`.
- Make sure each malocher-worker, including the supervisor, has access to a shared directory `malocher_dir`.
- Make sure every malocher-worker, including the supervisor, has the same python environment, e.g. put it into a shared directory.

12.1.3 Sample

The following sample shows how we can use malocher to train random forest classifiers of different depths on 4 SSH machines and collect the results in the main process.

```
import os
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
import malocher

def fake_experiment(model, data_path=None):
    train = pd.read_csv(os.path.join(data_path, "train.csv"))
    y_train = train["class"]
    X_train = train.drop(columns="class")
    test = pd.read_csv(os.path.join(data_path, "test.csv"))
    y_test = test["class"]
    X_test = test.drop(columns="class")
    model.fit(X_train, y_train)
    return dict(accuracy = model.score(X_test, y_test))

if __name__ == "__main__":
    print("running")
    CONFIGS = {}
    for D in range(1,10):
        MODEL = RandomForestClassifier(max_depth=D)
        # Store our Configuration under the Job's ID
        JOB = malocher.submit(
            fake_experiment,
```

```

        MODEL,
        data_path="/home/share/datensaetze/pamono"
    )
    CONFIGS[JOB] = D

RESULTS = malocher.process_all(
    jobs=CONFIGS.values(),
    ssh_machines=["ls8ws020", "ls8ws021", "ls8ws022", "ls8ws023"],
    ssh_port=22,
    ssh_username="dummy",
    ssh_private_key="malocher_id_rsa"
)
# Retrieve the config by the result's ID
for JOB, RESULT in RESULTS:
    print(CONFIGS[JOB], RESULT)

```

12.1.4 Pitfalls

Malocher starts each job in a new python interpreter. When libraries have changed during the experiment, these changes will be reflected in the outputs of the jobs. Try to avoid updating libraries and avoid installing libraries you are currently working on with `pip install -e`.

Arguments and globals will be stored on disk per job. Avoid loading large amounts of data in the main process to pass to the workers, as this will result in large files and long IO waits. It is often better to load the data on the workers. If pre-processing is necessary in the main process, you can serialize it to disk once and manually reload it in the workers.

12.2 meticulous-ml

»<https://github.com/Whadup/meticulous-ml>«

Originally developed by Ashwin Paranjape, `meticulous` is a library for tracking the execution and results of machine learning experiments in order to ensure reproducibility. We extended its tracking capabilities and added functionality to track multiple experiments in a single script.

12.2.1 Best Practices for Tracking Machine Learning Experiments

For ensuring proper reproducibility of results of machine learning experiments, there will be suggestions for some best-practices in the following.

Version control of Source Code All the source code written to execute the experiments should be under version control. The de-facto standard tool for version control is Git [266], although alternatives exist. Code should run from clean Git repositories

only, i.e. repositories with no uncommitted changes. This allows to associate the exact status of the code with the experiment run by its unique commit identifier that is provided by the version control software. Consequently, to replicate an experimental result, we can revert the code to the exact version used.

Version Control of all Dependencies Most machine learning experiments rely heavily on software libraries such as Tensorflow or Scikit-learn. These libraries are rapidly evolving and new versions are released frequently. Hence it is important to keep track of which version of libraries was used in an experiment. It is generally recommended to use one environment for each project using tools like Venv or Anaconda for python environments, or Docker [63] on the operating system level, and specify which libraries should be installed using the respective config-files. However, we should not rely on these config-files for ensuring reproducibility: They often allow vague version constraints (e.g. “newer than v1.2”), and a user can update a library without also editing the environment config-file. Hence we should also capture the software versions at runtime.

Version Control of Data There are less established software solutions for version control of data, though research data management is becoming increasingly important in science and more protocols are established in institutions [284]. For instance, open science data is often published and associated with a unique digital object identifier (DOI). Locally, all scripts that access, filter or preprocess the data should also be under version control. Whenever possible, we should include a script for obtaining the original data, alternatively we can maintain a description of how and where to obtain it. Intermediate data files, e.g. preprocessed data, should have meta data that contains, among others, a timestamp, a reference to the git commit of the preprocessing utility script that has been used, as well as a reference to all the input files that were used to generate the output. Many file formats support attaching meta data. For instance, in .json files or .hdf5 files a new field for meta data can be introduced and .csv files can begin with a comment section. If this is not possible, a metadata file should be stored next to the data file and copied around with the data.

Tracking of all Hyperparameters Machine learning methods, particularly deep learning approaches, have many hyperparameters and much time is spent tuning these parameters to maximize performance, either manually or automatically. Thus, we need to track all these hyperparameters to replicate the experiment later. When we are interested in reproducing the results of randomized algorithms, it is important to also track the random seeds used for the random generator.

Tracking of all Results We want to archive all the outcomes of an experiment. That includes any metric, including runtime, that we evaluate to judge the quality of a machine learning model, but also other outputs like the model itself or any other result files. It is important that each result or output can be associated with the correspond-

ing experiment. Often it is also useful to capture all console outputs as well as error messages in text files.

Reproducibility of Findings For maximum reproducibility of findings, experiments should be carried out multiple times with various initialization and different environments. These practices result in claims with sufficient statistical significance [103]. For training data, unbiased data in large quantities should be used. Negative outcomes in an experimental setup should also be published [236]. These measures potentially highlight pros and cons of a model and enhance the understanding of how or when it operates superior.

Use Experiment Tracking Software All the above information needs to be tracked, linked and archived. For this purpose, software solutions such as Sacred [95], MLflow [130], Tensorboard, Wandb [283], Theano [142] or Gym [199] exist and should be used. They provide convenient solutions to reproducibility that do not require changing large amounts of code. When we do machine learning experiments on cloud platforms, they often provide their own tools for reproducibility [116]. In the next section, we will present another software solution, `meticulous-ml`, in greater detail.

12.2.2 Example Python Snippet for `meticulous-ml`

In this section, we present the python library `meticulous-ml` [201], originally written by Ashwin Paranjape, that supports machine learning researchers by handling many of the requirements for reproducible research established above, while requiring only minimal code changes for existing experiment scripts. Perhaps most importantly, it is, as Hady Elsahar puts it, “*suitable for the messy, clueless nature of research*” [70]. We see an example for tracking a simple machine learning experiment, training and evaluating a random forest classifier, in Listing 1 and continue to discuss the highlighted, crucial changes to incorporate `meticulous`.

```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (accuracy_score, f1_score)
import argparse
import pickle
from meticulous import Experiment

parser = argparse.ArgumentParser ()
parser.add_argument('--max-depth', type=int , default=8)
parser.add_argument('--dataset', type=str , default="mozilla4")
# Adds the "meticulous" argument group to your script
Experiment.add_argument_group(parser)
# Creates experiment object and extracts all hyperparameters
experiment = Experiment.from_parser(parser)
```

```

args = parser.parse_args()

#load and split training data
X, y = fetch_openml(args.dataset, return_X_y=True)
y = LabelEncoder().fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y)

# train and validate the model
model = RandomForestClassifier(max_depth=args.max_depth)
model.fit(X_train, y_train)

# Log metrics in experiment directory
accuracy = accuracy_score(model.predict(X_test), y_test)
f1 = f1_score(model.predict(X_test), y_test)
print(f"Accuracy: {accuracy:.4f}, F1-Score: {f1:.4f}")
experiment.summary(accuracy=accuracy, f1=f1)

# Writes model file to the experiment directory
with experiment.open('model.pickle', 'wb') as f:
    pickle.dump(model, f)

```

First, to install `meticulous-ml`, we recommend using `pip`:

```

pip install \
    "git+https://github.com/Whadup/meticulous-ml"

```

but manual installation is also possible.

To enable tracking of an experiment, we have to instantiate an `Experiment` object. One convenient way is to derive it from the argument parser, simultaneously capturing all the hyperparameters provided by the user. Alternatively, we can manually provide all arguments that we want to capture. When the experiment is initialized, a folder is created on the local filesystem. It contains all information related to this experiment run in human-readable format: All standard output and errors will be captured, the hyperparameters provided through the argument parser will be stored and the meta data discussed in the previous section, including version control information, used software libraries, execution time, etc., will be captured.

When we want to capture metrics like accuracy or f1-score, we can use the `experiment.summary()` method. Now these scores are stored in a standardized way and can easily be retrieved and compared, for instance using the CLI tool described below.

When we want to store and archive files, replace `open()` with `experiment.open()` to obtain a file handle in the folder associated to the experiment run. Listing 2 summarizes the folder structure automatically created to archive the experiment results by `meticulous-ml`.

To ease browsing the results, we recommend using the CLI tool `meticulous` to select, filter, group and sort the runs and export the resulting tabular data into a variety of formats. In Listing 3 we see an example for analyzing the experimental results of the performance of the random forest classifier in our example. We have run this

Listing 12.1: Automatically generated file hierarchy of the `meticulous-ml` experiment directory (left) with description of each file or sub-directory (right).

```
$ find ./experiments_dir

./experiments_dir/ ← directory for all records
 1/                ← first experiment run
 [...]
 2/                ← second experiment run
  args.json        ← all cmd-line arguments
 some_output.txt  ← we can save custom files, too
 metadata.json    ← all meta-data
 STATUS           ← success, failure or still running?
 stdout           ← captured std-output
 stderr           ← captured std-error
 summary.json     ← all user summaries
```

Listing 12.2: Example for using the CLI tool `meticulous` to aggregate metrics and summarize experimental results. `Meticulous` automatically computes mean and standard deviation for numerical summaries if we use the `---groupby` argument.

```
$ meticulous experiment_dir
  --filter 'args_dataset=="magictelelescope"' \
  --groupby 'args_max_depth' \
  --sort 'summary_accuracy_mean' \
  --columns 'args_max_depth,summary_acc_mean,summary_acc_std' \
  --export 'results.tex'
```

	args_max_depth	summary_acc_mean	summary_acc_std
7	10	0.8706	0.0041
6	9	0.8681	0.0045
5	8	0.8610	0.0056
4	7	0.8515	0.0048
3	6	0.8486	0.0048
2	5	0.8379	0.0062
1	4	0.8288	0.0060
0	3	0.8038	0.0080

experiment for different values of `max_depth`, using multiple runs per configuration. We filter the results to only show runs relating to the `magictelelescope` dataset and compute means and standard deviations over the repeated runs. We sort the results to show the runs with the best average accuracy first and select only a subset of the table columns. Exporting this table in LaTeX format allows us to easily include the results in a scientific article.

This concludes the quick introduction into the `meticulous-ml` library: We have seen that minimal code changes already result in large increases in reproducibility of results. Furthermore, it provides a convenient commandline tool for browsing, summarizing and exporting the results.

12.3 arxiv-library

»https://github.com/Whadup/arxiv_library«

We have developed `arxiv-library` to solve the problem of extracting all mathematical expressions from the raw arxiv data that is available through the official arXiv.org bulk data website at https://arxiv.org/help/bulk_data_s3.

12.3.1 Installation

First, to install `arxiv-library`, we recommend using pip:

```
pip install \
  "git+https://github.com/Whadup/arxiv-library"
```

but manual installation is also possible.

Additionally the library requires `node.js` and the node module `katex` to be installed. You can download `node.js` at <https://nodejs.org/en/download/> and choose the respective installer for your os. The command

```
npm install -g yargs katex
```

installs the `katex` module we use for compiling LaTeX formulas to MathML.

12.3.2 Pipeline

Our commandline tool `equation-extractor` implements the full preprocessing pipeline used for the studies in this thesis. Starting with the compressed data obtained from arxiv, we extract and compile all formulas to a folder of json files, one per paper using

```
equation-extractor \
  </path/to/tar-archives> \
  </path/to/output_dir> \
  [--fulltext]
```

The first parameter points to a folder containing the tar-archives with the raw data. The tar-archives are organized as follows: One tar archive (e.g. “arXiv_src_1503_007.tar”) contains the sources for multiple papers from March 2015. One member of the tar-archive represents one publication and can be of three different types: A single gzip compressed tex-file, a .tar.gz archive containing multiple source files, or a gzip compressed pdf. The last case is impossible to handle for our library, these papers will be ignored during processing.

The pipeline extracts the archives and continues to process the retrieved papers. At the end of the pipeline we get a dictionary for each paper that contains the formulas that could be extracted (in LaTeX and in MathML format), and some meta data like author, arXiv-category etc. The extracted formulas are organized section wise.

The paper-dicts are stored in json-files in the folder specified in the second parameter.

The parameter `--fulltext` is an optional flag. If this flag is set, the paper’s content will be stored in the paper dict with key ”paper”.

12.3.3 Internal Python Functions

We also provide the internal functions of the pipeline for usage in other software. Below we showcase how to use some of the most important functionalities.

Extract tar You may retrieve all files for a tar archive with all papers from one month like this:

```
import arxiv_library.io_pkg.targz as io_targz

for compressed_file_dict in io_targz.process_tar("path/to/tars"):
    file_dict = io_targz.process_gz(compressed_file_dict)
    # do something with the file dict
```

Extract from LaTeX With the tex-extraction process we want extract the formulas from a paper. The paper is provided as a filedict.

During this process we perform the following steps:

1. Remove all comments in all tex-files.
2. Resolve all imports, so that we get one big tex-files that contains all paper content.
3. Extract the preamble of the big tex-file.
4. Split the paper content (i.e. all text wrapped in between `\begin{document}` and `\end{document}`) into sections.
5. Search for math-environments within the sections and extract them to the dictionary

6. Extract citation information from the paper.

You may implement this pipeline with something like this:

```
import arxiv_library.extraction.comments as comments
import arxiv_library.extraction.imports as imports
import arxiv_library.extraction.preamble as preamble
import arxiv_library.extraction.sections as sections
import arxiv_library.extraction.equations as equations
import arxiv_library.extraction.citations as citations

# Initially you need a filedict from the tar-extraction
file_dict = {"...": "..."}
file_dict = comments.remove_comments(file_dict)
paper_dict = imports.resolve_imports(file_dict)
paper_dict = preamble.extract_preamble(paper_dict)
paper_dict = sections.extract_sections(paper_dict)
paper_dict = equations.extract_equations(paper_dict)
paper_dict = citations.extract_citations(paper_dict)
# do something with the paper_dict
```

Compile to MathML To compile the extracted LaTeX-formulas to MathML you may use this code:

```
import arxiv_library.compilation.mathml as mathml
# Initially you will need the paper dict
# with the LaTeX-formulas that you want to compile.
paper_dict = {"...": "..."}
paper_dict = mathml.compile_paper(
    paper_dict,
    paper_dict["arxiv_id"]
)
# do something with the paper_dict
```

Retrieve meta data To retrieve the meta data for the publications by querying the arXiv-API you may use this code:

```
import arxiv_library.io_pkg.metadata as metadata
# Initially you will need a list with all paper dicts
# for which you want to retrieve meta data
paper_dicts = [...]
paper_dicts = metadata.receive_meta_data(paper_dicts)
# do something with the paper_dicts
```

These snippets illustrate that our library has use-cases outside of our mathematical expression search study.

12.4 xai-tracking

»<https://github.com/Whadup/xai-tracking>«

We have implemented the explanation methods based on tracking the training process of a deep network (Chapter 8) in a small python library. It can be obtained using git

```
git clone https://github.com/Whadup/xai-tracking
```

It supports the deep learning framework pytorch and includes classes to wrap around the optimizers used during training to enable the tracking of all weight updates.

Usage We have already seen two code snippets of our library in use for the tracking of the training process in Chapter 8 in Figures 8.4 and 8.9. To generate explanations from the tracked weight updates, we provide the `explain()` function that takes a model, an input example as well as a link to the tracked weight updates and computes all contribution values for all layers of the model.

From these contributions, we can either infer the most influential training examples, or compute the ridge plots seen in Chapter 8. We provide code snippets for those plots as well.

Chapter 13

Additional Tables and Figures

13.1 Mathematical Retrieval Results

Source of the Equation in Figure 4.5 (a)

- (q) Rajarshree Mitra. *A Generative Approach to Question Answering*, 2017
 - (1) Desmond Elliott, Stella Frank, Eva Hasler. *Multilingual Image Description with Neural Sequence Models*, 2015.
 - (2) Desmond Elliott, Stella Frank, Eva Hasler. *Multilingual Image Description with Neural Sequence Models*, 2015.
 - (3) Preksha Nema, Mitesh Khapra, Anirban Laha, Balaraman Ravindran. *Diversity driven Attention Model for Query-based Abstractive Summarization*, 2017.
 - (4) Hongyuan Mei, Mohit Bansal, Matthew R. Walter. *Listen, Attend, and Walk: Neural Mapping of Navigational Instructions to Action Sequences*, 2015.
 - (5) Hongyuan Mei, Mohit Bansal, Matthew R. Walter. *Listen, Attend, and Walk: Neural Mapping of Navigational Instructions to Action Sequences*, 2015.
-

Source of the Equation in Figure 4.5 (b)

- (q) Ted Dunning. *Finding Structure in Text, Genome and Other Symbolic Sequences*, 2012
 - (1) Ted Dunning. *Finding Structure in Text, Genome and Other Symbolic Sequences*, 2012
 - (2) Parthan Kasarapu, Lloyd Allison. *Minimum message length estimation of mixtures of multivariate Gaussian and von Mises-Fisher distributions*, 2015.
 - (3) Shiliang Zhang, Hui Jiang. *Hybrid Orthogonal Projection and Estimation (HOPE): A New Framework to Probe and Learn Neural Networks*, 2015.
 - (4) Meng Qu, Xiang Ren, Yu Zhang, Jiawei Han. *Weakly-supervised Relation Extraction by Pattern-enhanced Embedding Learning*, 2017.
 - (5) Yu Zhang, Srikanta Tirthapura, Graham Cormode. *Learning Graphical Models from a Distributed Stream*, 2017.
-

Source of the Equation in Figure 4.6 (a)

- (q) Han Cai, Kan Ren, Weinan Zhang, Kleanthis Malialis, Jun Wang, Yong Yu, Defeng Guo. *Real-Time Bidding by Reinforcement Learning in Display Advertising*, 2017.
 - (1) Kavosh Asadi, Michael L. Littman. *An Alternative Softmax Operator for Reinforcement Learning*, 2016.
 - (2) Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, Aviv Tamar. *Bayesian Reinforcement Learning: A Survey*, 2016.
 - (3) Ofir Nachum, Mohammad Norouzi, George Tucker, Dale Schuurmans. *Smoothed Action Value Functions for Learning Gaussian Policies*, 2018.
 - (4) Xiaomin Lin, Peter A. Beling, Randy Cogill. *Multi-agent Inverse Reinforcement Learning for Zero-sum Games*, 2014.
 - (5) Scott Fujimoto, Herke van Hoof, Dave Meger. *Addressing Function Approximation Error in Actor-Critic Methods*, 2018.
-

Source of the Equation in Figure 4.6 (b)

- (q) Yida Wang, Weihong Deng. *Generative Model with Coordinate Metric Learning for Object Recognition Based on 3D Models*, 2017.
 - (1) Yingming Li, Ming Yang, Zhongfei Zhang. *A Survey of Multi-View Representation Learning*, 2016.
 - (2) Tom Rainforth, Tuan Anh Le, Jan-Willem van de Meent, Michael A. Osborne, Frank Wood. *Bayesian Optimization for Probabilistic Programs*, 2017.
 - (3) Beipeng Mu, Shih-Yuan Liu, Liam Paull, John Leonard, Jonathan How. *SLAM with Objects using a Nonparametric Pose Graph*, 2017.
 - (4) Anqi Liu, Rizal Fathony, Brian D. Ziebart. *Kernel Robust Bias-Aware Prediction under Covariate Shift*, 2017.
 - (5) Roger Frigola, Carl Edward Rasmussen. *Integrated Pre-Processing for Bayesian Nonlinear System Identification with Gaussian Processes*, 2013.
-

13.2 Noisy Label Learning

Table 13.1: Average F1 scores (higher is better) per data set.

	$p-$	$p+$	LI&MA TREE	LI&MA THRESHOLD	MENON THRESHOLD	MITHAL THRESHOLD
YEAST-ML8	0.5	0.1	0.141±0.004	0.133±0.004	0.149±0.004	0.138±0.002
	0.5	0.25	0.131±0.006	0.136±0.004	0.137±0.001	0.118±0.013
	0.2	0.2	0.138±0.004	0.133±0.007	0.137±0.002	0.143±0.003
	0.4	0.4	0.135±0.003	0.131±0.006	0.009±0.025	0.092±0.024
	0.3	0.1	0.138±0.004	0.147±0.006	0.144±0.003	0.142±0.003
	0.1	0.3	0.153±0.005	0.156±0.008	0.138±0.001	0.140±0.001
WINE-QUALITY	0.5	0.1	0.086±0.014	0.192±0.005	0.125±0.039	0.214±0.008
	0.5	0.25	0.076±0.002	0.126±0.006	0.082±0.005	0.121±0.023
	0.2	0.2	0.116±0.010	0.244±0.007	0.255±0.016	0.092±0.004
	0.4	0.4	0.073±0.001	0.114±0.004	0.081±0.008	0.159±0.022
	0.3	0.1	0.115±0.022	0.310±0.010	0.277±0.018	0.092±0.006
	0.1	0.3	0.168±0.062	0.372±0.010	0.357±0.014	0.165±0.024
WEBPAGE	0.5	0.1	0.668±0.009	0.666±0.007	0.652±0.004	0.621±0.009
	0.5	0.25	0.306±0.171	0.561±0.007	0.564±0.016	0.544±0.017
	0.2	0.2	0.733±0.004	0.677±0.004	0.686±0.005	0.505±0.031
	0.4	0.4	0.195±0.130	0.401±0.009	0.405±0.042	0.418±0.021
	0.3	0.1	0.671±0.007	0.684±0.005	0.676±0.009	0.602±0.009
	0.1	0.3	0.679±0.009	0.678±0.007	0.679±0.004	0.499±0.010
US-CRIME	0.5	0.1	0.219±0.027	0.401±0.008	0.299±0.035	0.429±0.016
	0.5	0.25	0.157±0.008	0.349±0.011	0.227±0.014	0.215±0.015
	0.2	0.2	0.477±0.083	0.505±0.014	0.482±0.021	0.458±0.010
	0.4	0.4	0.147±0.006	0.204±0.011	0.261±0.028	0.264±0.026
	0.3	0.1	0.362±0.108	0.505±0.012	0.400±0.022	0.335±0.029
	0.1	0.3	0.501±0.044	0.502±0.014	0.428±0.030	0.506±0.018
THYROID-SICK	0.5	0.1	0.554±0.174	0.623±0.015	0.766±0.017	0.713±0.009
	0.5	0.25	0.195±0.031	0.526±0.014	0.623±0.039	0.668±0.042
	0.2	0.2	0.780±0.008	0.835±0.005	0.824±0.007	0.802±0.008
	0.4	0.4	0.166±0.021	0.291±0.007	0.229±0.014	0.644±0.016
	0.3	0.1	0.793±0.075	0.827±0.005	0.813±0.007	0.801±0.009
	0.1	0.3	0.809±0.007	0.836±0.007	0.851±0.009	0.796±0.012
SICK-EUTHYROID	0.5	0.1	0.610±0.159	0.658±0.010	0.802±0.007	0.822±0.006
	0.5	0.25	0.214±0.023	0.575±0.008	0.575±0.033	0.582±0.017
	0.2	0.2	0.731±0.161	0.856±0.004	0.852±0.007	0.847±0.006
	0.4	0.4	0.376±0.080	0.489±0.010	0.416±0.040	0.381±0.053
	0.3	0.1	0.842±0.007	0.808±0.005	0.850±0.005	0.818±0.011
	0.1	0.3	0.863±0.005	0.853±0.005	0.846±0.008	0.864±0.005
SCENE	0.5	0.1	0.144±0.005	0.181±0.005	0.138±0.001	0.151±0.006
	0.5	0.25	0.139±0.004	0.146±0.007	0.146±0.005	0.128±0.008
	0.2	0.2	0.143±0.008	0.221±0.009	0.180±0.010	0.207±0.011
	0.4	0.4	0.139±0.004	0.157±0.006	0.150±0.006	0.092±0.024
	0.3	0.1	0.151±0.007	0.206±0.006	0.167±0.008	0.156±0.010
	0.1	0.3	0.165±0.014	0.277±0.009	0.154±0.005	0.204±0.009
SATIMAGE	0.5	0.1	0.536±0.064	0.564±0.005	0.584±0.011	0.520±0.011
	0.5	0.25	0.438±0.080	0.424±0.004	0.520±0.023	0.539±0.013
	0.2	0.2	0.599±0.007	0.635±0.003	0.605±0.012	0.570±0.009
	0.4	0.4	0.259±0.044	0.321±0.005	0.503±0.013	0.527±0.010
	0.3	0.1	0.626±0.006	0.632±0.005	0.613±0.011	0.588±0.008
	0.1	0.3	0.625±0.007	0.642±0.003	0.611±0.008	0.608±0.005
PROTEIN-HOMO	0.5	0.1	0.784±0.003	0.453±0.004	0.797±0.002	0.791±0.003
	0.5	0.25	0.672±0.242	0.359±0.008	0.776±0.003	0.777±0.003
	0.2	0.2	0.806±0.002	0.504±0.005	0.799±0.003	0.797±0.002
	0.4	0.4	0.689±0.201	0.113±0.004	0.775±0.005	0.717±0.008
	0.3	0.1	0.799±0.002	0.472±0.005	0.786±0.002	0.795±0.002
	0.1	0.3	0.808±0.002	0.544±0.004	0.813±0.002	0.817±0.003

Table 13.2: Average F1 scores continued.

	$p-$	$p+$	LI&MA TREE	LI&MA THRESHOLD	MENON THRESHOLD	MITHAL THRESHOLD
PEN-DIGITS	0.5	0.1	0.936±0.003	0.945±0.002	0.961±0.002	0.952±0.003
	0.5	0.25	0.893±0.007	0.869±0.003	0.785±0.008	0.882±0.004
	0.2	0.2	0.971±0.002	0.956±0.002	0.974±0.002	0.961±0.003
	0.4	0.4	0.698±0.127	0.725±0.005	0.802±0.009	0.823±0.010
	0.3	0.1	0.967±0.002	0.967±0.002	0.974±0.002	0.977±0.002
OPTICAL-DIGITS	0.1	0.3	0.973±0.002	0.956±0.002	0.981±0.001	0.963±0.002
	0.5	0.1	0.842±0.008	0.815±0.007	0.847±0.010	0.805±0.006
	0.5	0.25	0.549±0.066	0.428±0.009	0.712±0.016	0.665±0.020
	0.2	0.2	0.890±0.005	0.906±0.006	0.907±0.004	0.851±0.006
	0.4	0.4	0.357±0.075	0.466±0.007	0.697±0.018	0.660±0.026
MAMMOGRAPHY	0.3	0.1	0.900±0.006	0.891±0.004	0.893±0.007	0.832±0.006
	0.1	0.3	0.888±0.006	0.875±0.005	0.908±0.006	0.841±0.007
	0.5	0.1	0.136±0.046	0.484±0.007	0.583±0.017	0.582±0.011
	0.5	0.25	0.081±0.018	0.308±0.008	0.287±0.062	0.486±0.045
	0.2	0.2	0.622±0.011	0.622±0.008	0.615±0.010	0.636±0.007
LETTER-ING	0.4	0.4	0.070±0.012	0.202±0.005	0.301±0.046	0.239±0.071
	0.3	0.1	0.648±0.009	0.580±0.010	0.593±0.013	0.621±0.012
	0.1	0.3	0.647±0.008	0.664±0.006	0.652±0.008	0.611±0.010
	0.5	0.1	0.780±0.176	0.818±0.004	0.834±0.006	0.810±0.010
	0.5	0.25	0.183±0.053	0.734±0.004	0.648±0.010	0.588±0.019
ISOLET	0.2	0.2	0.880±0.004	0.849±0.002	0.869±0.006	0.830±0.007
	0.4	0.4	0.290±0.120	0.515±0.009	0.571±0.032	0.634±0.020
	0.3	0.1	0.889±0.003	0.861±0.002	0.877±0.004	0.842±0.006
	0.1	0.3	0.892±0.004	0.859±0.003	0.890±0.005	0.869±0.005
	0.5	0.1	0.699±0.015	0.465±0.007	0.729±0.008	0.665±0.009
COIL-2000	0.5	0.25	0.318±0.072	0.298±0.005	0.478±0.044	0.455±0.042
	0.2	0.2	0.819±0.007	0.780±0.006	0.810±0.007	0.776±0.006
	0.4	0.4	0.198±0.029	0.284±0.004	0.370±0.038	0.508±0.035
	0.3	0.1	0.786±0.008	0.763±0.005	0.819±0.007	0.757±0.006
	0.1	0.3	0.802±0.005	0.814±0.008	0.802±0.005	0.783±0.006
CAR-EVAL-34	0.5	0.1	0.117±0.004	0.135±0.004	0.121±0.002	0.135±0.007
	0.5	0.25	0.110±0.004	0.147±0.002	0.123±0.003	0.076±0.016
	0.2	0.2	0.135±0.010	0.207±0.005	0.130±0.004	0.117±0.002
	0.4	0.4	0.120±0.002	0.134±0.003	0.127±0.007	0.116±0.002
	0.3	0.1	0.133±0.006	0.193±0.003	0.153±0.013	0.128±0.004
ABALONE	0.1	0.3	0.169±0.020	0.218±0.004	0.138±0.006	0.166±0.008
	0.5	0.1	0.308±0.077	0.301±0.008	0.487±0.052	0.557±0.026
	0.5	0.25	0.180±0.017	0.238±0.003	0.340±0.019	0.287±0.028
	0.2	0.2	0.686±0.066	0.639±0.011	0.727±0.017	0.792±0.015
	0.4	0.4	0.146±0.006	0.265±0.007	0.337±0.027	0.149±0.003
	0.3	0.1	0.686±0.099	0.601±0.007	0.733±0.021	0.666±0.018
	0.1	0.3	0.757±0.017	0.795±0.013	0.832±0.018	0.826±0.015
	0.5	0.1	0.223±0.015	0.317±0.006	0.366±0.006	0.404±0.012
	0.5	0.25	0.186±0.010	0.265±0.004	0.290±0.012	0.300±0.017
	0.2	0.2	0.298±0.018	0.355±0.004	0.358±0.013	0.369±0.008
	0.4	0.4	0.194±0.010	0.223±0.004	0.192±0.005	0.179±0.003
	0.3	0.1	0.344±0.031	0.368±0.006	0.380±0.008	0.386±0.007
	0.1	0.3	0.392±0.007	0.397±0.004	0.400±0.006	0.371±0.007

List of Figures

2.1	Different types of computation graphs	17
2.2	Example decision tree of depth 2 for binary classification	27
3.1	The Flip-Probabilities for Class-Conditional Label Noise	34
3.2	The Auto-Encoder architecture with its components. The training objective is minimizing the expected reconstruction error $\min \mathbb{E}_x \ell(x, \phi'(\phi(x)))$ and we train using an unlabeled dataset.	37
3.3	Contrastive Learning with Data Augmentations	42
4.1	Number of Papers per Subject Areas in our Sample	48
4.2	Illustration of the Bitmap Representation	50
4.3	Equation Encoder Architecture (small)	51
4.4	A dendrogram showing a hierarchical clustering of our evaluation dataset according to our embedding model, colored according to hand-annotated labels, gray for non-pure clusters.	52
4.5	Sample Queries and Results: The first line shows queries, the remaining lines show the top-5 results. Shows center rectangle only.	53
4.6	More Sample Queries and Results: The first line shows queries, the remaining lines show the top-5 results. Shows center rectangle only. The results for these queries are more diverse.	53
4.7	Example MathML for $\frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i)$	56
4.8	The 50 most frequent characters in math environments.	60
4.9	Example of the Masking Task with Fictional Values	63
4.10	Example: Bayes' law. We report the first result and the first result that does not show Bayes' law, but, in this case, the related law of total probability. The first result is from: R. H. Leike, T. A. Enßlin, <i>Charting nearby dust clouds using Gaia data only</i> , 2019.	69
4.11	Example: Ising Model. We find equations related to both Ising Models and Boltzmann Machines. First result is from: Weinstein, <i>Learning the Einstein-Podolsky-Rosen correlations on a Restricted Boltzmann Machine</i> , 2017. Second result is from: Ferrari et al., <i>Finite size corrections to disordered systems on Erdős-Rényi random graphs</i> , 2013.	70

5.1	Simulated air showers of both classes – gamma and hadron – as captured by the FACT telescope.	74
5.2	Hexagonal data transformed to a quadratic grid. The transformation is bijective in the green (and blue) area and does not change the raw values, however it only approximates the neighborhood relationships of the original hexagonal image. Red areas are padded with zeros. Figure taken from [35].	76
5.3	Schematic of our Multi-Head Architecture	78
5.4	Angular resolution of the origin estimator for different energy levels (left) and bias and resolution of the energy estimator. Transparent markers show the performance of the random-forest-based estimators.	81
5.5	Histogram of the frequencies of gamma predictions as a function of the squared angular distance between the trajectory of any incoming ray and a position in the sky for <code>efficientnet-b0</code> (left) and the standard analysis using random forest (right). On-events show the frequency with respect to the position of the Crab Nebula, while Off-events are w.r.t. positions with no known sources. The significance of detection test only considers angles smaller than 0.025 (left of the dashed vertical line).	83
5.6	An Illustrative Example of Li&Ma Decision Threshold Tuning with $p_- = 0.4$, $p_+ = 0.2$ and $P(y = +1) = 0.25$. See Main-Text for a Detailed Review.	92
5.7	Each row displays one critical difference diagram [53] which corresponds to one noise configuration. This overview summarizes a total of 81,600 random forest classifiers, which consist of 4 binary learning methods for noisy labels \times 6 noise configurations \times 17 data sets \times 10 cross validation folds \times 20 repetitions with random initialization. Methods are connected with horizontal brackets if and only if a Holm-corrected Wilcoxon signed-rank test cannot significantly distinguish their pairwise performances, in terms of their cross-validated F1 score, at a confidence level of 0.90.	94
5.8	An Illustration of the Wobble mode for capturing on- and off-regions simultaneously. Figure taken from [206].	96
5.9	The loss surface for the Soft-Li&Ma and crossentropy objective for a minibatch of size 2, where x, y are the model outputs for two examples with noisy label +1 and -1 respectively. We see that there is a bad local maximum that is separated from the global maxima by a deep valley. In contrast, the cross-entropy loss has only one global maxima that is easily accessible.	100
7.1	Example Query (Taylor Approximation)	117
7.2	Example Query (Empirical Risk Minimization)	117
7.3	Example Query (Multi-Layer Perceptron)	117
7.4	Example Query (Submodular Functions)	117

7.5	Example Query (Rademacher Complexity)	117
7.6	Example Query (Hoeffding's Bound)	118
8.1	Example Explanation Histogram: " <i>The model predicts frog as training examples of that class had the highest influence on the computed intermediate representation in layer 14</i> ".	122
8.2	Illustration of the decomposition of preactivations into individual gradient updates. The marked updates are the most substantial individual contributions to Wx	124
8.3	Illustration of the different batch-normalization variants, color indicates class labels. (b) When we construct batches with a single class, the outputs span the whole range. An imaginary next batch of the other class would do the same, hindering training. (c) Ghost samples solve this problem, as mean and variance are also computed on the ghost batch.	128
8.4	Sample Code for Capturing the Training Process of a VGG16 model trained with SGD on the CIFAR-10 dataset. Required changes are highlighted, we see that we need to include indices in our training data, use our class-wise samples, substitute batch normalization for our ghost normalization layers and use a wrapped variant of the optimizer.	133
8.5	The 7 most influential training examples for the 3 layers of the VGG-16 network. Top left image is the input image for reference. We see a high influence of the class "car" on the representation in the middle layers, here e.g. in the 10th layer. The later layers, eg. the 13th layer, is influenced mostly by very similar images, i.e. trucks photographed from an angle, whereas the earlier convolutional layers, here the 4th layer, is influenced by images of trucks shot from a variety of angles.	134
8.6	Ridge Plot for the 3 layers of the VGG-16 network. We see a high influence of the class "car" on the representation in the middle layers, here e.g. in the 10th layer. The earlier layers are not as descriptive of individual classes, eg. the 4th layer.	135
8.7	Highly influential images for the first layer from a variety of classes leave the practitioner confused. Input image in top left position for reference.	136
8.8	Ridge Plot for the output layer, the 14th layer and the 12th layer. We see that while the representation at the 14th layer can be attributed to examples of the correct class, the representations in the 12th layer and in all prior layers look unspecific to a class. This indicates a poor choice of network architecture that does not utilize all layers for classification.	136
8.9	Sample Code for Capturing the Training Process of a VGG16 model trained with SGD on the CIFAR-10 dataset. Required changes are highlighted and are minimal. Indeed we do not need to change anything about the model, nor the way data is sampled.	140

8.10	Ridge Plot for the 3 layers of the VGG-16 network as computed using the examplewise gradient contributions aggregated across examples. The earlier layers are not as descriptive of individual classes, eg. the 1st layer.	141
8.11	Example Outputs for our Approach and TracIn for an instance of the class "Ship". Noteworthy, the two sets of top 7 most influential examples are distinct. We observe that TracIn produces more visually similar images, whereas our set captures a more diverse set of ship types.	143
9.1	Our Lower Bounds (DISTILLGRD) vs. the Ground Truth. Points over the diagonal are outputs that are not actually lower bounds. We see that with lower budget values δ , the number of these failures decreases.	158

List of Tables

4.1	Performance Metrics for the small and large Models for two different embedding learning objective functions.	52
4.2	Math-BERT model configurations	57
4.3	Results of the Mathematical Retrieval Experiment. We report recall@K for $K \in \{1, 10, 100\}$	59
4.4	Ablation Study	67
4.5	Eval Scores	68
4.6	Results of the Mathematical Retrieval Experiment. We report recall@K for $K \in \{1, 10, 100\}$	69
4.7	Search Engine Performance	71
5.1	Comparison of the different model architectures used for our study. .	80
5.2	Performance of the three prediction heads on simulation data of the different multi-task model architectures.	81
5.3	Significance of Detection (Def. 2) on Crab nebula data of the different model architectures.	83
5.4	Results of the Simulation Study for Different Noise Configurations. We observe that our estimated thresholds are always close to the optimal thresholds. In configurations where $p_+ \neq p_-$ this yields substantial reductions of the error probabilities. In configurations where $p_+ = p_-$, the optimal threshold is the naive threshold of 0.5, which cannot be improved by our method.	93
5.5	Average F1 scores (higher is better) over 17 data sets and 20 repetitions of a 10-fold cross-validation with different label noise configurations. A full version is in the appendix.	95
5.6	f_α outcomes (see Definition 2, higher is better) for the open Crab Nebula data of the FACT telescope.	98
5.7	f_α outcomes (see Def. 2) for the FACT data with artificially substituted on-instances.	99
5.8	Significance of Detection when our noisy learning models are transferred to different sources.	99

5.9	Significance of Detection Scores (Cross-Validated) for <code>efficientnet-b0</code> networks. We see that the noisy label approach that retraines the classification head with noisy labels and the one that finetunes the classification head outperform the previous classification head based on full supervision with simulation data.	101
8.1	Computational Overhead Statistics for the Previous Experiments . . .	132
8.2	Computational Overhead Statistics for the Previous Experiments . . .	142
9.1	Comparison of the Adversarial Robustness Values Across Datasets and Methods	158
9.2	Quantitive Analysis of the Quality of our Bounds for the MAGIC dataset. Let r denote the exact robustness and \tilde{r} denote our estimate of DISTILLGRD.	160
13.1	Average F1 scores (higher is better) per data set.	186
13.2	Average F1 scores continued.	187

Bibliography

- [1] R. Abbasi, M. Ackermann, J. Adams, J. A. Aguilar, M. Ahlers, M. Ahrens, and And Others. A Convolutional Neural Network based Cascade Reconstruction for the IceCube Neutrino Observatory. Technical report, 2021. 77
- [2] V. A. Acciari and Others. MAGIC observations of the diffuse emission in the vicinity of the Galactic center. *A and A*, 642, 2020. 98
- [3] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity Checks for Saliency Maps. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. 121, 145
- [4] Fatemeh Alizadeh, Gunnar Stevens, and Margarita Esau. I Don't Know, Is AI Also Used in Airbags? An Empirical Study of Folk Concepts and People's Expectations of Current and Future Artificial Intelligence. *i-com*, 20(1):3–17, 2021. 105
- [5] Francisco Álvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. Recognition of On-line Handwritten Mathematical Expressions Using 2D Stochastic Context-Free Grammars and Hidden Markov Models. *Pattern Recognition Letters*, (35):58–67, 2014. 49
- [6] H. Anderhub, M. Backes, A. Biland, V. Boccone, I. Braun, T. Bretz, J. Buß, F. Cadoux, V. Commichau, L. Djambazov, D. Dorner, S. Einecke, D. Eisenacher, A. Gendotti, O. Grimm, H. Von Gunten, C. Haller, D. Hildebrand, U. Horisberger, B. Huber, K. S. Kim, M. L. Knoetig, J. H. Köhne, T. Krähenbühl, B. Krumm, M. Lee, E. Lorenz, W. Lustermaan, E. Lyard, K. Mannheim, M. Meharga, K. Meier, T. Montaruli, D. Neise, F. Nessi-Tedaldi, A. K. Overkemping, A. Paravac, F. Pauss, D. Renker, W. Rhode, M. Ribordy, U. Röser, J. P. Stucki, J. Schneider, T. Steinbring, F. Temme, J. Thaele, S. Tobler, G. Viertel, P. Vogler, R. Walter, K. Warda, Q. Weitzel, and M. Zänglein. Design and operation of FACT-the first G-APD Cherenkov telescope. *Journal of Instrumentation*, 8(6), 2013. 13, 73, 82

- [7] Forough Arabshahi, Zhichu Lu, Pranay Mundra, Sameer Singh, and Animashree Anandkumar. Compositional Generalization with Tree Stack Memory Units. Technical report, 2020. 71
- [8] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *Proceedings of the 35th International Conference on Machine Learning*, pages 254–263. PMLR, 2018. 149
- [9] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*, volume 463. ACM press New York, 1999. 47
- [10] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 119.1–119.11. BMVA Press, 2016. 40, 62
- [11] Gagan Bansal, Besmira Nushi, Ece Kamar, Walter S. Lasecki, Daniel S. Weld, and Eric Horvitz. Beyond Accuracy: The Role of Mental Models in Human-AI Team Performance. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, 2019. 105
- [12] Suzanna Becker and Geoffrey E. Hinton. Self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355(6356):161–163, 1992. 39
- [13] Katharina Beckh, Sebastian Müller, Matthias Jakobs, Vanessa Toborek, Hanxiao Tao, Raphael Fischer, Pascal Welke, Sebastian Houben, and Laura von Rueden. Explainable Machine Learning with Prior Knowledge: An Overview. Technical report, 2021. 105
- [14] Emily M Bender and Angelina Mcmillan-major. *On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?*, volume 1. Association for Computing Machinery, 2021. 10
- [15] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1993):1–30, 2013. 37
- [16] Erik Bernhardsson. *Annoy: Approximate Nearest Neighbors in C++/Python*, 2018. 47, 58, 59
- [17] A. Biland, T. Bretz, J. Buß, V. Commichau, L. Djambazov, D. Dorner, S. Eicke, D. Eisenacher, J. Freiwald, O. Grimm, H. Von Gunten, C. Haller, C. Hempfling, D. Hildebrand, G. Hughes, U. Horisberger, M. L. Knoetig, T. Krähenbühl, W. Lustermann, E. Lyard, K. Mannheim, K. Meier, S. Mueller,

- D. Neise, A. K. Overkemping, A. Paravac, F. Pauss, W. Rhode, U. Röser, J. P. Stucki, T. Steinbring, F. Temme, J. Thaele, P. Vogler, R. Walter, and Q. Weitzel. Calibration and performance of the photon sensor response of FACT - The first G-APD Cherenkov telescope. *Journal of Instrumentation*, 9(10), 2014. 82
- [18] Alexander Binder, Wojciech Samek, Grégoire Montavon, Sebastian Bach, and Klaus-Robert Müller. Analyzing and validating neural networks predictions. In *Proceedings of the ICML 2016 Workshop on Visualization for Deep Learning*, 2016. 107
- [19] Abeba Birhane and Vinay Uday Prabhu. Large image datasets: A pyrrhic win for computer vision? In *WACV 2021*, pages 1537–1547, 2021. 10
- [20] Rok Blagus and Lara Lusa. SMOTE for high-dimensional class-imbalanced data. *BMC Bioinformatics*, 14(1):106, 2013. 26, 33
- [21] Avrim Blum and Tom M. Mitchell. Combining Labeled and Unlabeled Data with Co-Training. In *Conference on Computational Learning Theory*. ACM, 1998. 35
- [22] Christian Bockermann, Kai Brügge, Jens Buß, Alexey Egorov, Katharina Morik, Wolfgang Rhode, and Tim Ruhe. Online Analysis of High-Volume Data Streams in Astroparticle Physics. In Albert Bifet, Michael May, Bianca Zadrozny, Ricard Gavaldà, Dino Pedreschi, Francesco Bonchi, Jaime S. Cardoso, and Myra Spiliopoulou, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part III*, volume 9286 of *Lecture Notes in Computer Science*, pages 100–115. Springer, 2015. 74, 84
- [23] Francesco Bodria, Fosca Giannotti, Riccardo Guidotti, Francesca Naretto, Dino Pedreschi, and Salvatore Rinzivillo. A Survey Of Methods For Explaining Black-Box Models. *ACM Computing Surveys*, 51(5):1–33, 2021. 108
- [24] Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. OUP Oxford, 2013. 22
- [25] Ryma Boumazouza, Fahima Cheikh-Alili, Bertrand Mazure, and Karim Tabia. *ASTERYX: A Model-Agnostic SaT-BasEd AppRoach for SYmbolic and Score-Based EXplanations*, pages 120–129. Association for Computing Machinery, New York, NY, USA, 2021. 148
- [26] H. Boursard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4):291–294, 1988. 37
- [27] Xavier Bouthillier, Pierre Delaunay, Mirko Bronzi, Assya Trofimov, Brennan Nichyporuk, Justin Szeto, Naz Sepah, Edward Raff, Kanika Madan, Vikram Voleti, Samira Ebrahimi Kahou, Vincent Michalski, Dmitriy Serdyuk, Tal Arbel, Chris Pal, Gaël Varoquaux, and Pascal Vincent. Accounting for Variance in

Machine Learning Benchmarks. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, 2021. 82

- [28] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2), 1996. 28, 91
- [29] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. 28, 91, 106, 107, 147
- [30] Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984. 27
- [31] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. 9, 43, 55
- [32] Rudy Bunel, Philip H S Torr, M Pawan Kumar, and Jingyue Lu. Branch and Bound for Piecewise Linear Neural Network Verification. *Journal of Machine Learning Research*, 21:1–39, 2020. 148
- [33] Sebastian Buschjäger, Jian-Jia Chen, Kuan-Hsun Chen, Mario Günzel, Christian Hakert, Katharina Morik, Rodion Novkin, Lukas Pfahler, and Mikail Yayla. Margin-Maximization in Binarized Neural Networks for Optimizing Bit Error Tolerance. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 673–678, 2021. 23
- [34] Sebastian Buschjäger and Katharina Morik. There is no Double-Descent in Random Forests. Technical Report 2019, 2021. 29
- [35] Sebastian Buschjäger, Lukas Pfahler, Jens Buß, Katharina Morik, and Wolfgang Rhode. On-Site Gamma-Hadron Separation with Deep Learning on FPGAs. In Yuxiao Dong, Dunja Mladenic, and Craig Saunders, editors, *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part IV*, volume 12460 of *Lecture Notes in Computer Science*, pages 478–493. Springer, 2020. 76, 77, 84, 167, 190
- [36] Sebastian Buschjäger, Lukas Pfahler, and Katharina Morik. Generalized Negative Correlation Learning for Deep Ensembling. Technical report. 29

- [37] Stefano Calzavara, Claudio Lucchese, and Gabriele Tolomei. Adversarial Training of Gradient-Boosted Decision Trees. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM '19*, pages 2429–2432, New York, NY, USA, 2019. Association for Computing Machinery. 149
- [38] Bradley W. Carroll and Dale A. Ostlie. *An introduction to modern astrophysics*. Cambridge University Press, 2017. 73
- [39] Rich Caruana. *Multitask Learning*. PhD thesis, Carnegie Mellon University, 1997. 35
- [40] Rich Caruana. Multitask Learning. *Machine Learning*, 28:41–75, 1997. 35, 36
- [41] Jacopo Cavazza, Pietro Morerio, Benjamin Haeffele, Connor Lane, Vittorio Murino, and René Vidal. Dropout as a Low-Rank Regularizer for Matrix Factorization. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, number 2. JMLR, 2018. 26
- [42] Tony F. Chan, Gene H. Golub, and Randall J. LeVeque. Updating formulae and a pairwise algorithm for computing sample variances. In *COMPSTAT 1982 5th Symposium held at Toulouse 1982*, pages 30–41. Springer, 1982. 128
- [43] Guillaume Charpiat, Nicolas Girard, Loris Felardos, and Yuliya Tarabalka. Input Similarity from the Neural Network Perspective. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 1–10, Vancouver, 2019. Curran Associates, Inc. 142
- [44] Hongge Chen, Huan Zhang, Duane Boning, and Cho-Jui Hsieh. Robust decision trees against adversarial examples. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, pages 1122–1131. PMLR, 2019. 149
- [45] Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane Boning, and Cho-Jui Hsieh. Robustness Verification of Tree-based Models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 1–12. Curran Associates, Inc., 2019. 148, 152, 159
- [46] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, Vienna, 2020. PMLR. 42, 58

- [47] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *International Conference on Learning Representations*, 2020. 43
- [48] Stephan Cl  men  on, Igor Colin, and Aur  lien Bellet. Scaling-up Empirical Risk Minimization: Optimization of Incomplete U-statistics. *Journal of Machine Learning Research*, 17:1–36, 2016. 34, 64, 66
- [49] Ronan Collobert and Jason Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167, New York, NY, USA, 2008. Association for Computing Machinery. 36
- [50] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable Robustness of ReLU networks via Maximization of Linear Regions. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019. 109, 110
- [51] Ricardo Cruz, Kelwin Fernandes, Jaime S. Cardoso, and Joaquim F. Pinto Costa. Tackling Class Imbalance with Ranking. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2182–2187. IEEE, 2016. 33
- [52] Andrew M. Dai, Christopher Olah, Quoc Viet Le, and Greg S. Corrado. Document Embedding with Paragraph Vectors. In *Neurips Workshop on deep neural networks and representation learning*, pages 1–9, 2014. 38
- [53] Janez Dem  sar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, dec 2006. 94, 95, 190
- [54] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 9
- [55] Jia Deng, Olga Russakovsky, Jonathan Krause, Michael Bernstein, Alexander C Berg, and Li Fei-Fei. Scalable Multi-Label Annotation. In *ACM Conference on Human Factors in Computing Systems (CHI)*, 2014. 9
- [56] Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M. Rush. Image-to-Markup Generation with Coarse-to-Fine Attention. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, pages 980–989. JMLR, 2017. 49
- [57] ONNX Runtime Developers and Microsoft. ONNX Runtime, 2021. 81
- [58] Jacob Devlin, Ming-Wei Chang, Lee Kenton, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT 2019*, pages 4171–4186. Association for Computational Linguistics, 2019. 9, 43, 55, 56, 57, 63

- [59] Laurens Devos, Wannes Meert, and Jesse Davis. Verifying tree ensembles by reasoning about potential instances. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 450–458. SIAM, 2021. 148
- [60] Laurens Devos, Wannes Meert, and Jesse Davis. Versatile Verification of Tree Ensembles. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139. PMLR, 2021. 148, 153, 159
- [61] Luc Devroye, László Györfi, and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, 1996. 151
- [62] Shengyong Ding, Liang Lin, Guangrun Wang, and Hongyang Chao. Deep feature learning with relative distance comparison for person re-identification. *Pattern Recognition*, 48(10):2993–3003, 2015. 23, 49, 61
- [63] Inc. Docker. Docker Website, 2021. 174
- [64] E. Domingo-Santamaría, J. Flix, V. Scalzotto, W. Wittek, and J. Rico. The DISP analysis method for point-like or extended γ source searches/studies with the MAGIC Telescope. In *The Magic Project – MAGIC Detector and Analysis Details*. 2005. 74
- [65] Robin Dylan Drew. Deep Unsupervised Domain Adaptation for Gamma-Hadron Separation. Master’s thesis, TU Dortmund, 2021. 77
- [66] Vijay D’Silva, Daniel Kroening, and Georg Weissenbacher. A Survey of Automated Techniques for Formal Software Verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1165–1178, 2008. 110
- [67] David K. Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances In Neural Information Processing Systems*, volume 28, pages 2224–2232. Curran Associates, Inc, 2015. 21, 119
- [68] Rüdiger Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In Deepak D’Souza and K Narayan Kumar, editors, *Automated Technology for Verification and Analysis*, pages 269–286, Cham, 2017. Springer International Publishing. 110
- [69] Gil Einziger, Maayan Goldstein, Yaniv Sa, Itai Segall, and Bell Labs. Verifying Robustness of Gradient Boosted Models. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, 2018. 148

- [70] Hady Elsahar. How do you manage your Machine Learning Experiments?, aug 2019. 175
- [71] Torsten A. Enßlin, Mona Frommert, and Francisco S. Kitaura. Information field theory for cosmological perturbation reconstruction and nonlinear signal analysis. *Phys. Rev. D*, 80(10):105005, nov 2009. 45
- [72] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing Higher-Layer Features of a Deep Network Visualizing Higher-Layer Features of a Deep Network Introduction. Technical Report 2014, 2009. 130
- [73] Linus Ericsson, Henry Gouk, and Timothy M. Hospedales. How Well Do Self-Supervised Models Transfer? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5414–5423, jun 2021. 37
- [74] Dan Feldman, Sedat Ozer, and Daniela Rus. Coresets for Vector Summarization with Applications to Network Graphs. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 2017. 145
- [75] Manuel Fernández-Delgado, Eva Cernadas, Senßen Barro, and Diano Amirim. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014. 29, 147
- [76] Abe Fetterman and Josh Albrecht. Understanding selfsupervised and contrastive learning with” bootstrap your own latent”(byol). 2020. 42
- [77] V. P. Fomin, A. A. Stepanian, R. C. Lamb, D. A. Lewis, M. Punch, and T. C. Weekes. New methods of atmospheric Cherenkov imaging for gamma-ray astronomy. I. The False Source method. *Astroparticle Physics*, 6505(94), 1994. 74, 82
- [78] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware Minimization for Efficiently Improving Generalization. In *International Conference on Learning Representations*, 2021. 123
- [79] Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999. 36
- [80] Kunihiro Fukushima. Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969. 19
- [81] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating Training Data Influence by Tracing Gradient Descent. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances In Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020. 122, 137, 142

- [82] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020. 10, 107
- [83] Pascal Germain, Alexandre Lacasse, Francois Laviolette, Mario March, and Jean-Francois Roy. Risk Bounds for the Majority Vote: From a PAC-Bayesian Analysis to a Learning Algorithm. *Journal of Machine Learning Research*, 16(26):787–860, 2015. 29
- [84] Aritra Ghosh, Naresh Manwani, and P. S. Sastry. Making risk minimization tolerant to label noise. *Neurocomputing*, 160, 2015. 34, 35, 84
- [85] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised Representation Learning by Predicting Image Rotations. In *International Conference on Learning Representations*, pages 1–16, 2018. 44
- [86] Julien Girard-Satabin, Guillaume Charpiat, Zakaria Hichem Chihani, and Marc Schoenauer. CAMUS: A Framework to Build Formal Specifications for Deep Perception Systems Using Simulators. In *ECAI 2020*, pages 1–19, 2020. 110
- [87] Julien Girard-Satabin, Aymeric Varasse, Marc Schoenauer, Guillaume Charpiat, and Zakaria Chihani. COnvex polytopes for neural network verification. Technical report. 110
- [88] Anna Gladkova and Aleksandr Drozd. Intrinsic Evaluations of Word Embeddings: What Can We Do Better? In *Proceedings of the 1st Workshop on Evaluating Vector Space Representations for NLP*, pages 36–42, 2016. 51
- [89] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. 9:249–256, 2010. 25
- [90] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15, pages 315–323. JMLR, 2011. 19
- [91] Micah Goldblum, Liam Fowl, Soheil Feizi, and Tom Goldstein. Adversarially Robust Distillation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*. AAAI Press, 2020. 149
- [92] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. 15
- [93] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*, 2015. 109

- [94] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021. 149
- [95] Klaus Greff, Aaron Klein, Martin Chovanec, and Frank Hutter. The Sacred Infrastructure for Computational Research. In *Proceedings of the 15th Python in Science Conference (SCIPY 2017)*, pages 49–56, 2017. 175
- [96] Ferruccio Guidi and Claudio Sacerdoti Coen. A survey on retrieval of mathematical knowledge. *Mathematics in Computer Science*, 10(4):409–427, 2016. 46
- [97] Riccardo Guidotti, Anna Monreale, Fosca Giannotti, Dino Pedreschi, Salvatore Ruggieri, and Franco Turini. Factual and Counterfactual Explanations for Black Box Decision Making. *IEEE Intelligent Systems*, 34(6):14–23, 2019. 107
- [98] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, and Dino Pedreschi. Local Rule-Based Explanations of Black Box Decision Systems. Technical Report May, 2018. 106
- [99] Zellig S. Harris. Distributional Structure. *WORD*, 10(2-3):146–162, 1954. 38
- [100] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*, volume 2. Springer, 2009. 9, 22, 37
- [101] Simai He, Jiawei Zhang, and Shuzhong Zhang. Bounding Probability of Small Deviation: A Fourth Moment Approach. *Mathematics of Operations Research*, 35(1), 2010. 154
- [102] D. Heck, J. Knapp, J. N. Capdevielle, G. Schatz, and T. Thouw. *CORSIKA: a Monte Carlo code to simulate extensive air showers*. Forschungszentrum Karlsruhe GmbH, Karlsruhe (Germany), feb 1998. 76
- [103] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning that Matters, 2019. 175
- [104] Dan Hendrycks, Collin Burns, Saurav Kadavath, Dawn Song, Akul Arora, Eric Tang, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan, editors, *Advances In Neural Information Processing Systems*, volume 34, pages 1–22. Curran Associates, Inc., 2021. 72
- [105] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). Technical report, 2016. 19
- [106] Sibylle Heß. *A Mathematical Theory of Making Hard Decisions : Model Selection and Robustness of Matrix Factorization with Binary Constraints*. PhD thesis, TU Dortmund University, 2018. 123

- [107] A. M. Hillas. Cerenkov Light Images of EAS Produced by Primary Gamma Rays and by Nuclei. In *19th International Cosmic Ray Conference (ICRC19), Volume 3*, volume 3 of *International Cosmic Ray Conference*, page 445, aug 1985. 74
- [108] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006. 15
- [109] Geoffrey E. Hinton and Richard S. Zemel. Autoencoders, Minimum Description Length and Helmholtz Free Energy. In J. Cowan, G. Tesauro, and J. Al-spector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 3–10. Morgan-Kaufmann, 1993. 37
- [110] Liam Hodgkinson, Umut Şimşekli, Rajiv Khanna, and Michael W. Mahoney. Generalization Properties of Stochastic Optimizers via Trajectory Analysis. Technical report. 123
- [111] Wassily Hoeffding. A class of statistics with asymptotically normal distribution. *Annals of Statistics*, 1948. 65
- [112] Daniel Hsu, Ziwei Ji, Matus Telgarsky, and Lan Wang. Generalization bounds via distillation. In *International Conference on Learning Representations*, pages 1–25, 2021. 149
- [113] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, Jure Leskovec, Regina Barzilay, Peter Battaglia, Yoshua Bengio, Michael Bronstein, Stephan Günnemann, Will Hamilton, Tommi Jaakkola, Stefanie Jegelka, Maximilian Nickel, Chris Re, Le Song, Jian Tang, Max Welling, and Rich Zemel. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances In Neural Information Processing Systems*, volume 33, pages 1–34. Curran Associates, Inc., 2020. 131
- [114] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinpeng Yi. A Survey of Safety and Trustworthiness of Deep Neural Networks: Verification, Testing, Adversarial Attack and Defence, and Interpretability. Technical report, 2020. 109, 148
- [115] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 448–456, Lille, France, 2015. PMLR. 19, 61, 127
- [116] Richard Isdahl and Odd Erik Gundersen. Out-of-the-box Reproducibility: A Survey of Machine Learning Platforms. In *15th International Conference on eScience*, pages 86–95. IEEE, 2019. 175

- [117] Yacine Izza and Joao Marques-Silva. On Explaining Random Forests with SAT. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2584–2591. International Joint Conferences on Artificial Intelligence Organization, 2021. 148
- [118] Mikael Jacquemont, Thomas Vuillaume, Alexandre Benoit, Cilles Maurin, Patrick Lambert, and Giovanni Lamanna. First Full-Event Reconstruction from Imaging Atmospheric Cherenkov Telescope Real Data with Deep Learning. Technical report, 2021. 75, 164
- [119] Mikaël Jacquemont, Thomas Vuillaume, Alexandre Benoit, Gilles Maurin, and Patrick Lambert. Single Imaging Atmospheric Cherenkov Telescope Full-Event Reconstruction with a Deep Multi-Task Learning Architecture. Technical report, 2020. 77, 164
- [120] Mikaël Jacquemont, Thomas Vuillaume, Alexandre Benoit, Gilles Maurin, and Patrick Lambert. Multi-Task architecture with attention for imaging atmospheric cherenkov telescope data analysis. In *VISIGRAPP 2021 - Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, volume 5, pages 534–544, 2021. 77, 164
- [121] Ariel Jaffe, Yuval Kluger, Ofir Lindenbaum, Jonathan Patsenker, Erez Peterfreund, and Stefan Steinerberger. The Spectral Underpinning of word2vec. *Frontiers in Applied Mathematics and Statistics*, 6(December):1–10, 2020. 39
- [122] Katarzyna Janocha and Wojciech Marian Czarnecki. On Loss Functions for Deep Neural Networks in Classification. *Schedae Informaticae*, 25:1–10, 2017. 40
- [123] Svante Janson. Large Deviations for Sums of Partly Dependent Random Variables. *Random Structures & Algorithms*, 24(3):234–248, 2003. 65, 66
- [124] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1–55, 2017. 55, 58, 59, 69
- [125] Andreas Hotho Julian Tritscher, Fabian Gwinner, Daniel Schlör, Anna Krause. Open ERP System Data For Occupational Fraud Detection Julian. Technical report. 108
- [126] Alex Kantchelian, J.D. Tygar, and Anthony D. Joseph. Evasion and Hardening of Tree Ensemble Classifiers. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning*, volume 48. JMLR, 2016. 147, 148, 159

- [127] Grigoris Karakoulas and John Shawe-Taylor. Optimizing Classifiers for Imbalanced Training Sets. In M Kearns, S Solla, and D Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11. MIT Press, 1998. 33
- [128] Amir-Hossein Karimi, Bernhard Schölkopf, and Isabel Valera. Algorithmic recourse: from counterfactual explanations to interventions. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 353–362, 2021. 107
- [129] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017. 148
- [130] Harutaka Kawamura, Arjun DCunha, Andrew Chen, Richard Zang, and Others. MLflow. 175
- [131] Nicolas Kayser-Bril. Google apologizes after its Vision AI produced racist results, 2020. 10
- [132] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. In *CVPR 2018*, pages 7482–7491. IEEE. 36
- [133] D. B. Kieda, VERITAS Collaboration, and Others. Status of the VERITAS ground based GeV/TeV Gamma-Ray Observatory. In *Bulletin of the American Astronomical Society*, volume 36, page 910, 2004. 73
- [134] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In Yoshua Bengio and Yann LeCun, editors, *International Conference on Learning Representations*, 2015. 25, 123
- [135] Pang Wei Koh and Percy Liang. Understanding Black-box Predictions via Influence Functions. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 2017. 108, 145
- [136] Frederik Kratzert. Understanding the backward pass through Batch Normalization Layer, 2016. 20
- [137] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 9
- [138] Ulrike Kuhl, André Artelt, and Barbara Hammer. Keep Your Friends Close and Your Counterfactuals Closer: Improved Learning From Closests Rather Than Plausible Counterfactual Explanations in an Abstract Setting. In *ACM Conference on Fairness, Accountability, and Transparency*, pages 1–17, Seoul, 2022. Association for Computing Machinery. 107

- [139] Todd Kulesza, Simone Stumpf, Margaret Burnett, Sherry Yang, Irwin Kwan, and Weng-keen Wong. Too Much, Too Little, or Just Right? Ways Explanations Impact End Users' Mental Models. In *IEEE Symposium on Visual Languages and Human Centric Computing*, pages 3–10. IEEE, 2013. 105
- [140] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *International Conference on Learning Representations Workshops*, number c, pages 1–14, 2017. 109
- [141] Vitaly Kurin, Alessandro De Palma, Ilya Kostrikov, Shimon Whiteson, and M Pawan Kumar. In Defense of the Unitary Scalarization for Deep Multi-Task Learning. Technical report, 2022. 36
- [142] LISA lab Revision. Theano Documentation, 2021. 175
- [143] Guillaume Lample and François Charton. Deep Learning for Symbolic Mathematics. In *International Conference on Learning Representations*, pages 1–24, 2020. 71, 164
- [144] Markus Langer, Daniel Oster, Lena Kästner, Timo Speith, Kevin Baum, Holger Hermanns, Eva Schmidt, and Andreas Sesing. What Do We Want From Explainable Artificial Intelligence (XAI)? *Artificial Intelligence*, pages 0–57, 2021. 121
- [145] Yan LeCun, O. Matan, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jacket, and H. S. Baird. Handwritten zip code recognition with multilayer networks. In *Proceedings. 10th International Conference on Pattern Recognition*, volume ii, pages 35–40 vol.2, 1990. 18
- [146] Yann Lecun. A Path Towards Autonomous Machine Intelligence. Technical report, 2022. 72
- [147] Jason D. Lee, Qi Lei, Nikunj Saunshi, and Jiacheng Zhuo. Predicting What You Already Know Helps: Provable Self-Supervised Learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances In Neural Information Processing Systems*, volume 34. Curran Associates, Inc, 2021. 43
- [148] Guillaume Lemaitre, Fernando Nogueira, and Christos K Aridas. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *J. Mach. Learn. Res.*, 18, 2017. 93
- [149] Omer Levy and Yoav Goldberg. Neural Word Embedding as Implicit Matrix Factorization. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances In Neural Information Processing Systems*, volume 27, 2014. 39

- [150] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, and Henryk Michalewski. Solving Quantitative Reasoning Problems with Language Models. Technical report. 72, 163
- [151] Ti-Pei Li and Yu-Qian Ma. Analysis methods for results in gamma-ray astronomy. *The Astrophysical Journal*, 272, 1983. 82, 87, 96
- [152] Xiao-Hui Li, Yuhan Shi, Haoyang Li, Wei Bai, Yuanwei Song, Caleb Chen Cao, and Lei Chen. Quantitative Evaluations on Saliency Methods: An Experimental Study. In *Proceedings of ACM Conference (Conference'17)*, volume 1. Association for Computing Machinery. 107
- [153] Xuefeng Li, Tongliang Liu, Bo Han, Gang Niu, and Masashi Sugiyama. Provably End-to-end Label-noise Learning without Anchor Points. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 2021. 35
- [154] Thomas I. Liao, Rohan Taori, Inioluwa Deborah Raji, and Ludwig Schmidt. Are We Learning Yet? A Meta-Review of Evaluation Failures Across Machine Learning. In *35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks*, number NeurIPS, 2021. 10
- [155] Hongye Liu, Yonghong Tian, Yaowei Wang, Lu Pang, and Tiejun Huang. Deep Relative Distance Learning: Tell the Difference Between Similar Vehicles. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2167–2175, 2016. 49
- [156] Scott M Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 107
- [157] Alaa Maalouf, Ibrahim Jubran, Murad Tukan, and Dan Feldman. Coresets for the Average Case Error for Finite Query Sets. *Sensors*, 21(6689), 2021. 145
- [158] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu, and Computer Science. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*, number 2017, pages 1–23, 2018. 109
- [159] Mahshad Mahdavi, Richard Zanibbi, Harold Mouch, and Christian Viardgaudin. ICDAR 2019 CROHME + TFD : Competition on Recognition of Handwritten Mathematical Expressions and Typeset Formula Detection. In *15th IAPR International Conference on Document Analysis and Recognition (ICDAR 2019)*, 2019. 49

- [160] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014. 47
- [161] Behrooz Mansouri, Douglas W. Oard, C. Lee Giles, and Richard Zanibbi. Tangent-CFT : An Embedding Model for Mathematical Formulas. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*. Association for Computing Machinery, 2019. 47, 58, 59
- [162] C. Marrocco, R. P. W. Duin, and F. Tortorella. Maximizing the area under the ROC curve by pairwise feature combination. *Pattern Recognition*, 41(6):1961–1974, 2008. 34
- [163] Andreas Maurer. Learning similarity with operator-valued large-margin classifiers. *Journal of Machine Learning Research*, 9:1049–1082, 2008. 64
- [164] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral Regularization Algorithms for Learning Large Incomplete Matrices. *Journal of Machine Learning Research*, 11:2287–2322, 2010. 26
- [165] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A Survey on Bias and Fairness in Machine Learning. *ACM Computing Surveys*, 54(6), jul 2021. 10
- [166] Dennis Y Menn and Hung-yi Lee. Searching for the Essence of Adversarial Perturbations. Technical report, 2022. 108
- [167] Aditya Krishna Menon, Brendan van Rooyen, Cheng Soon Ong, and Robert C Williamson. Learning from Corrupted Binary Labels via Class-Probability Estimation. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37. PMLR, 2015. 34, 35, 95, 97, 98
- [168] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, Kai Chen, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *International Conference on Learning Representations*, 2013. 38
- [169] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26, pages 1–9, 2013. 38, 39
- [170] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 746–751, 2013. 38, 51

- [171] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6707–6717, 2020. 58
- [172] Tom M Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. 15
- [173] Varun Mithal, Guruprasad Nayak, Ankush Khandelwal, Vipin Kumar, Nikunj C. Oza, and Ramakrishna R. Nemani. RAPT: Rare Class Prediction in Absence of True Labels. *IEEE Transactions on Knowledge and Data Engineering*, 29(11), 2017. 35, 97, 98
- [174] Hossein Mobahi, Ronan Collobert, and Jason Weston. Deep Learning from Temporal Coherence in Video. In *Proceedings of the 26th International Conference on Machine Learning*, pages 737–744. PMLR, 2009. 39
- [175] Apostolos Modas, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. SparseFool: A Few Pixels Make a Big Difference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, jun 2019. 109
- [176] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, Cambridge, MA, USA, 2012. 23
- [177] Christoph Molnar. *Interpretable Machine Learning*. Independently published, 2019. 106, 107
- [178] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 246–252, 2005. 38
- [179] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019. 21
- [180] Sascha Mücke and Lukas Pfahler. Check Mate: A Sanity Check for Trustworthy AI. Technical report, 2022. 108
- [181] S A Mueller and Others. Single photon extraction for FACT’s SiPMs allows for novel IACT event representation. In *Proceedings of Science*, 2017. 77
- [182] Mark Niklas Müller, Gleb Mararchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. PRIMA : General and Precise Neural Network Certification via Scalable Convex Hull Approximations. In *Proceedings of the ACM on Programming Languages*, volume 6, pages 1–33. Association for Computing Machinery, 2022. 110
- [183] Nagarajan Natarajan, Inderjit S. Dhillon, Pradeep K. Ravikumar, and Ambuj Tewari. Learning with Noisy Labels. In C. J. C. Burges, L. Bottou, M. Welling,

- Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. 34, 35, 84, 93
- [184] Meike Nauta, Jan Trienes, Shreyasi Pathak, Elisa Nguyen, Michelle Peters, Yasmin Schmitt, Jörg Schlötterer, Maurice van Keulen, and Christin Seifert. From Anecdotal Evidence to Quantitative Evaluation Methods: A Systematic Review on Evaluating Explainable AI. Technical Report 1, 2022. 108
- [185] Domonik Neise, Laurits Tani, Maximilian Nöthe, Mirko Bunse, Kai Brügge, and Jens Buß. *photon_stream*, 2021. 77
- [186] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Stochastic Approximation approach to Stochastic Programming. *SIAM Journal on Optimization*, 19:1574–1609, 2009. 16
- [187] An-phi Nguyen and María Rodríguez Martínez. On quantitative aspects of model interpretability. Technical report, 2020. 108
- [188] Chaoqun Nie, Jianqi Shi, and Yanhong Huang. VARE: Verifying and Analyzing Robustness of Random Forests. In Shang-Wei Lin, Zhe Hou, and Brendan Mahony, editors, *Formal Methods and Software Engineering*, pages 163–178, Cham, 2020. Springer International Publishing. 148
- [189] Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit MLE: Backpropagating Through Discrete Exponential Family Distributions. *Advances in Neural Information Processing Systems*, 18:14567–14579, 2021. 164
- [190] D Nieto, A Brill, Q Feng, M Jacquemont, B Kim, T Miener, and T Vuillaume. Studying deep convolutional neural networks with hexagonal lattices for imaging atmospheric Cherenkov telescope event reconstruction. *Proceedings of Science*, pages 0–7, 2019. 75, 164
- [191] Kimia Noorbakhsh, Modar Sulaiman, Mahdi Sharifi, Kallol Roy, and Pooyan Jamshidi. Pretrained Language Models are Symbolic Mathematics Solvers too! Technical report, 2019. 71
- [192] Mehdi Noroozi and Paolo Favaro. Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 69–84, Cham, 2016. Springer International Publishing. 44
- [193] Mehdi Noroozi, Hamed Pirsiavash, and Paolo Favaro. Representation Learning by Learning to Count. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5898–5906. IEEE, 2017. 44
- [194] Maximilian Nöthe. Improving the Angular Resolution of FACT Using Machine Learning. Technical report, Tu Dortmund University, Dortmund, 2017. 74

- [195] Maximilian Nöthe. *Monitoring the High Energy Universe*. PhD thesis, TU Dortmund University, 2020. 74, 77, 78, 80
- [196] Maximilian Nöthe. FACT Open Crab Sample Analysis. https://github.com/fact-project/open_crab_sample_analysis, 2021. 97
- [197] Vít Novotný, Petr Sojka, Michal Štefánik, and Dávid Lupták. Three is Better than One Ensembling Math Information Retrieval Systems. (September 2020):22–25. 55
- [198] Aaron Van Den Oord, Yazhe Li, and Oriol Vinyals. Representation Learning with Contrastive Predictive Coding. Technical report. 41, 58
- [199] OpenAI. Gym Website, 2021. 175
- [200] Alexander G Ororbia, Dan Kifer, and C Lee Giles. Recognizing and Verifying Mathematical Equations using Multiplicative Differential Neural Units. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021. 164
- [201] Ashwin Paranjape and Lukas Pfahler. meticulous-ml. 175
- [202] Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making Deep Neural Networks Robust to Label Noise: a Loss Correction Approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, 2017. 35
- [203] Amandalynne Paullada, Inioluwa Deborah Raji, Emily M. Bender, Emily Denton, and Alex Hanna. Data and its (dis)contents: A survey of dataset development and use in machine learning research. *Patterns*, 2(11):100336, 2021. 10
- [204] Shuai Peng, Ke Yuan, Liangcai Gao, and Zhi Tang. MathBERT: A Pre-Trained Model for Mathematical Formula Understanding. Technical report. 72, 163
- [205] D. Petry. The MAGIC Telescope-prospects for GRB research. *Astronomy and Astrophysics Supplement Series*, 138(3):601–602, 1999. 73
- [206] Lukas Pfahler, Mirko Bunse, and Katharina Morik. Noisy Labels for Weakly Supervised Gamma Hadron Classification. Technical report, TU Dortmund University, 2021. 96, 167, 190
- [207] Lukas Pfahler and Katharina Morik. Learning Low-Rank Document Embeddings with Weighted Nuclear Norm Regularization. In *IEEE International Conference on Data Science and Advanced Analytics*, page 16. IEEE, 2017. 38
- [208] Lukas Pfahler and Katharina Morik. Nyström-SGD : Rapidly Learning Kernel-Classifiers with Conditioned Stochastic Gradient Descent. In *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2018*, Dublin, 2018. Springer. 123

- [209] Lukas Pfahler and Katharina Morik. Semantic Search in Millions of Equations. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020. 37, 58, 72, 167
- [210] Lukas Pfahler and Jan Richter. Interpretable Nearest Neighbor Queries for Tree-Structured Data in Vector Databases of Graph-Neural Network Embeddings. In *Workshop Proceedings of the EDBT/ICDT 2020 Joint Conference*, pages 2–6. 168
- [211] Lukas Pfahler, Jonathan Schill, and Katharina Morik. The Search for Equations - Learning to Identify Similarities between Mathematical Expressions. In *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2019*, 2019. 50, 58, 167
- [212] Lukas Pfahler, Alina Timmermann, and Katharina Morik. ML2R Coding Nuggets: Reproducible Machine Learning Experiments ML2R Coding Nuggets Reproducible Machine Learning Experiments. Technical Report January, ML2R, 2022. 168
- [213] Hung Viet Pham, Jiannan Wang, Thibaud Lutellier, Jonathan Rosenthal, and Lin Tan. Problems and Opportunities in Training Deep Learning Software Systems: An Analysis of Variance. In *ASE '20*. 82
- [214] J. Ross Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986. 27
- [215] J. Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 1993. 151
- [216] Markus N. Rabe, Christian Szegedy, Kshitij Bansal, and Dennis Lee. Mathematical Reasoning Via Self-Supervised Skip-Tree Training. In *International Conference on Learning Representations*, pages 1–21, 2021. 72, 163
- [217] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and Others. Language models are unsupervised multitask learners. 2019. 9, 43, 55, 56
- [218] Francesco Ranzato and Marco Zanella. Robustness Verification of Decision Tree Ensembles. In *Proceedings of the 1st Workshop on Artificial Intelligence and Formal Verification, Logics, Automata and Synthesis (OVERLAY)*, pages 8–13, 2019. 148
- [219] Francesco Ranzato and Marco Zanella. Abstract Interpretation of Decision Tree Ensemble Classifiers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5478–5486, 2020. 148
- [220] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference*

- on *Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, nov 2019. Association for Computational Linguistics. 55
- [221] Marco Tulio Ribeiro, Carlos Guestrin, Sameer Singh, and Carlos Guestrin. ”Why Should I Trust You?”: Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’16*, pages 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. 106
- [222] Pierre H. Richemond, Jean-Bastien Grill, Florent Alché, Corentin Tallec, Florian Strub, Andrew Brock, Samuel Smith, Soham De Razvan, Bilal Piot, and Michal Valko. BYOL works even without batch statistics. In *NeurIPS 2020 Workshop: Self-Supervised Learning-Theory and Practice*, pages 1–6. 42
- [223] Shaurya Rohatgi, Jian Wu, and C. Lee Giles. PSU at CLEF-2020 ARQMath Track : Unsupervised Re-ranking using Pretraining. pages 22–25, 2020. 55
- [224] Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. Technical Report May, 2017. 36
- [225] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, and Lesia Semenova. Interpretable Machine Learning : Fundamental Principles and 10 Grand Challenges. *Statistics Surveys*, pages 1–80, 2021. 105
- [226] Tim Ruhe, Dominik Elsässer, Wolfgang Rhode, Maximilian Nöthe, and Kai Brügge. Cherenkov Telescope Ring-An Idea for World Wide Monitoring of the VHE Sky. *EPJ Web Conference*, 207, 2019. 75
- [227] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Internal Representations By Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*. MIT Press, 1985. 18
- [228] Stefan Rüping. *Learning interpretable models*. PhD thesis, TU Dortmund University, 2006. 106, 107
- [229] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Kumar Paritosh, and Lora Mois Aroyo. ”Everyone wants to do the model work, not the data work”: Data Cascades in High-Stakes AI. In *SIGCHI*. ACM, 2021. 9, 10
- [230] Benjamin Sanchez-Lengeling, Jennifer Wei, Brian Lee, Emily Reif, Peter Y Wang, Wei Qian, Kevin Mccloskey, Lucy Colwell, and Alexander Wiltschko. Evaluating Attribution for Graph Neural Networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances In Neural Information Processing Systems*, volume 33, pages 1–13. Curran Associates, Inc., 2020. 108, 113

- [231] Naoto Sato, Hironobu Kuruma, Yuichiro Nakagawa, and Hideto Ogawa. Formal Verification of a Decision-Tree Ensemble Model and Detection of Its Violation Ranges. *IEICE Transactions on Information and Systems*, E103.D(2):363–378, 2020. 148
- [232] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing Mathematical Reasoning Abilities of Neural Models. pages 1–17, 2019. 164
- [233] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, (September):298–307, 2015. 51
- [234] Patrick Schramowski, Wolfgang Stammer, Stefano Teso, Anna Brugger, Franziska Herbert, Xiaoting Shao, Hans-georg Luigs, Anne-Katrin Mahlein, and Kristian Kersting. Making deep neural networks right for the right scientific reasons by interacting with their explanations. *Nature Machine Intelligence*, (Xil):1–18, 2020. 10, 107
- [235] Clayton Scott, Gilles Blanchard, and Gregory Handy. Classification with Asymmetric Label Noise: Consistency and Maximal Denoising. In *Conference on Computational Learning Theory*, volume 30 of *JMLR Workshop and Conf. Proc.*, 2013. 34, 35, 84
- [236] D. Sculley, Jasper Snoek, Alex Wiltschko, and Ali Rahimi. Winner’s Curse? On Pace, Progress, and Empirical Rigor, 2018. 175
- [237] Andrew D. Selbst and Solon Barocas. The Intuitive Appeal of Explainable Machines. *Fordham Law Review*, 2018. 121
- [238] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1134–1141. IEEE, 2018. 39
- [239] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In H Wallach, H Larochelle, A Beygelzimer, F d'Alché-Buc, E Fox, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 109
- [240] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning : From Theory to Algorithms*. Cambridge University Press, 2014. 23
- [241] Shawn Shan, Emily Wenger, Jiayun Zhang, Huiying Li, Haitao Zheng, and Ben Y Zhao. Fawkes: Protecting Privacy against Unauthorized Deep Learning Models. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security 2020)*, 2020. 109

- [242] Devashish Shankar, Sujay Narumanchi, H. A. Ananya, Pramod Kompalli, and Krishnendu Chaudhury. Deep Learning based Large Scale Visual Recommendation and Search for E-Commerce, 2017. 49
- [243] I. Shilon, M. Kraus, M. Büchele, K. Egberts, T. Fischer, T.L. Holch, T. Lohse, U. Schwanke, C. Steppa, and S. Funk. Application of Deep Learning methods to analysis of Imaging Atmospheric Cherenkov Telescopes data. 2018. 75, 164
- [244] S. D. Silvey. *Statistical Inference*, volume 7. CRC Press, 1975. 88
- [245] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *CoRR*, dec 2013. 107, 115, 121
- [246] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. Technical report, 2014. 107
- [247] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. 26
- [248] Erik Strumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41:647–665, 2013. 107
- [249] Abby Stylianou and Robert Pless. Visualizing Deep Similarity Networks. Technical report, 2019. 107, 114
- [250] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019. 109
- [251] Jong-Chyi Su, Subhransu Maji, and Bharath Hariharan. When Does Self-supervision Improve Few-Shot Learning? In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 645–666, Cham, 2020. Springer International Publishing. 37, 43
- [252] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28, Atlanta, Georgia, USA, 2013. PMLR. 123
- [253] Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hananeh Hajishirzi, Noah A Smith, and Yejin Choi. Dataset Cartography: Mapping and Diagnosing Datasets with Training Dynamics. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 9275–9293. Association for Computational Linguistics, 2020. 123

- [254] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 108, 109, 148
- [255] Damian Szklarczyk, Annika L. Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T. Doncheva, John H. Morris, Peer Bork, Lars J. Jensen, and Christian von Mering. STRING v11: protein-protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research*, 47(D1):D607–D613, jan 2019. 131
- [256] Chengli Tan, Jianshe Zhang, and Junmin Liu. Trajectory-dependent Generalization Bounds for Deep Neural Networks via Fractional Brownian Motion. Technical report. 123
- [257] Mingxing Tan and Quoc Viet Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *Proceedings of the 36th International Conference on Machine Learning*, 2019. 78
- [258] Yi Tay, Vinh Q Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, William W Cohen, and Donald Metzler. Transformer Memory as a Differentiable Search Index. Technical report, Google Research, 2022. 47
- [259] Yonglong Tian, Chen Sun, Cordelia Schmid, Ben Poole, and Phillip Isola. What Makes for Good Views for Contrastive Learning? In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances In Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020. 42
- [260] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005. 26
- [261] Richard M Timoney. OpenMath LaTeX to OpenMath Converter. Technical report, 1999. 46
- [262] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations*, pages 1–21, 2019. 148
- [263] Stefan Todorinski. Erkennung von Ähnlichkeiten zwischen mathematischen Ausdrücken mittels Bidirectional Encoder Representations from Transformers. Master’s thesis, TU Dortmund, 2021. 167
- [264] John Törnblom and Simin Nadjm-Tehrani. An Abstraction-Refinement Approach to Formal Verification of Tree Ensembles. In Alexander Romanovsky, Elena Troubitsyna, Ilir Gashi, Erwin Schoitsch, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 301–313, Cham, 2019. Springer International Publishing. 148

- [265] John Törnblom and Simin Nadjm-Tehrani. Formal verification of input-output mappings of tree ensembles. *Science of Computer Programming*, 194:102450, 2020. 148
- [266] Linus Torvalds, Junio Hamano, and Others. Git - fast, scalable, distributed revision control system, 2005. 173
- [267] Julian Tritscher, Markus Ring, Daniel Schlr, Lena Hettinger, and Andreas Hotho. Evaluation of Post-hoc XAI Approaches Through Synthetic Tabular Data. In Denis Helic, Gerhard Leitner, Martin Stettinger, Alexander Felfernig, and Zbigniew W Raś, editors, *Foundations of Intelligent Systems*, pages 422–430, Cham, 2020. Springer International Publishing. 108
- [268] Joel A. Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Practical sketching algorithms for low-rank matrix approximation. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1454–1485, 2017. 145
- [269] Evgeniya Ustinova and Victor Lempitsky. Learning Deep Embeddings with Histogram Loss. In *Advances In Neural Information Processing Systems 2016*, 2016. 41, 62
- [270] Joaquin Vanschoren and Serena Yeung. Announcing the NeurIPS 2021 Datasets and Benchmarks Track. 9, 10
- [271] Joaquin Vanschoren and Serena Yeung, editors. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. 2021. 9, 10, 48
- [272] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, second edition, 2000. 23, 64
- [273] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in Neural Information Processing Systems*, 2017. 20, 55, 57, 61
- [274] Robin Vogel, Aurélien Bellet, and Stéphan Cléménçon. A Probabilistic Theory of Supervised Similarity Learning for Pointwise ROC Curve Optimization. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018. 34
- [275] Ulrike von Luxburg, Robert C. Williamson, and Isabelle Guyon. Clustering: Science or Art? In *JMLR Workshop and Conference Proceedings 27*, pages 65–79, 2012. 51
- [276] David Wallis and Irène Buvat. Clever Hans effect found in a widely used brain tumour MRI dataset. *Medical Image Analysis*, 77:102368, 2022. 107

- [277] Alex Wang, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. SuperGLUE : A Stickier Benchmark for General-Purpose Language Understanding Systems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances In Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 10
- [278] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *International Conference on Learning Representations*, pages 1–20, 2019. 10
- [279] Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. Translating Math Word Problem to Expression Tree. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1064–1069, 2018. 46
- [280] Yihan Wang, Huan Zhang, Hongge Chen, and Duane Boning. On ℓ_p -norm Robustness of Ensemble Decision Stumps and Trees. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*. PLMR, 2020. 148, 152
- [281] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019. 21
- [282] Sebastian Wankerl, Andrzej Dulny, Gerhard Götz, and Andreas Hotho. Learning Mathematical Relations Using Deep Tree Models. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1681–1687, 2021. 71, 164
- [283] Weights & Biases. wandb, 2021. 175
- [284] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A. C. ’t Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1):160018, 2016. 174

- [285] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3733–3742, 2018. 41
- [286] Xiaobo Xia, Tongliang Liu, Nannan Wang, Bo Han, Chen Gong, Gang Niu, and Masashi Sugiyama. Are Anchor Points Really Indispensable in Label-Noise Learning? In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 35
- [287] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful Are Graph Neural Networks? In *International Conference on Learning Representations*, pages 1–17, 2019. 131
- [288] Yu Yao, Tongliang Liu, Bo Han, Mingming Gong, Jiankang Deng, Gang Niu, and Masashi Sugiyama. Dual T: Reducing Estimation Error for Transition Matrix in Label-noise Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 7260–7271. Curran Associates, Inc., 2020. 35
- [289] Chih-kuan Yeh, Joon Sik Kium, Ian E. H. Yen, and Pradeep Ravikumar. Representer Point Selection for Explaining Deep Neural Networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances In Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. 145
- [290] Jaemin Yoo and Taebum Kim. Knowledge Extraction with No Observable Data. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances In Neural Information Processing Systems*, number 32. Curran Associates, Inc., 2019. 149
- [291] Richard Zanibbi, Douglas W. Oard, Anurag Agarwal, and Behrooz Mansouri. Overview of ARQMath 2020 (Updated Working Notes Version): CLEF Lab on Answer Retrieval for Questions on Math. 2020(September):22–25, 2020. 55
- [292] Richard Zanibbi, Goran Topi, Michael Kohlhase, and Kenny Davila. NTCIR-12 MathIR Task Overview. In *Proceedings of the 12th NTCIR Conference on Evaluation of Information Access Technologies*, Tokyo, Japan, 2016. 47, 69
- [293] Jure Zbontar, Li Jing, Ishan Misra, and Yann LeCun. Barlow Twins: Self-Supervised Learning via Redundancy Reduction. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 2021. 43
- [294] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne

- Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833. Springer International Publishing, 2014. 107
- [295] Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words - Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, 2021. 9
- [296] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017. 23
- [297] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding Deep Learning (Still) Requires Rethinking Generalization. *Communications of the ACM*, 64(3):107–115, feb 2021. 23
- [298] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond Empirical Risk Minimization. In *International Conference on Learning Representations*, pages 1–13, 2018. 26
- [299] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful Image Colorization. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 649–666, Cham, 2016. Springer International Publishing. 44
- [300] Ruimao Zhang, Liang Lin, Rui Zhang, Wangmeng Zuo, and Lei Zhang. Bit-Scalable Deep Hashing With Regularized Similarity Learning for Image Retrieval and Person Re-Identification. *IEEE Transactions on Image Processing*, 24(12):4766–4779, 2015. 49
- [301] Wei Zhang, Ziming Huang, Yada Zhu, Guangnan Ye, Xiaodong Cui, and Fan Zhang. On Sample Based Explanation Methods for NLP: Faithfulness, Efficiency and Semantic Evaluation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5399–5411, Online, aug 2021. Association for Computational Linguistics. 123
- [302] Wei Zhong and Richard Zanibbi. Structural Similarity Search for Formulas using Leaf-Root Paths in Operator Subtrees. In *European Conference on Information Retrieval*, pages 116–129. Springer, 2019. 46, 47
- [303] Jianlong Zhou, Amir H Gandomi, Fang Chen, and Andreas Holzinger. Evaluating the Quality of Machine Learning Explanations: A Survey on Methods and Metrics. *Electronics*, 10(5):1–19, 2021. 108
- [304] Marinka Zitnik, Rok Sosič, Marcus W. Feldman, and Jure Leskovec. Evolution of resilience in protein interactomes across the tree of life. *Proceedings of the National Academy of Sciences*, 116(10):4426–4433, 2019. 131

- [305] Barret Zoph, Jonathon Shlens, and Quoc Viet Le. RandAugment: Practical automated data augmentation with a reduced search space. Technical report. 26