THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

On the Robustness of Statistical Models: Entropy-based Regularisation and Sensitivity of Boolean Deep Neural Networks

Olof Zetterqvist

CHALMERS

Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2023

On the Robustness of Statistical Models: Entropy-based Regularisation and Sensitivity of Boolean Deep Neural Networks
Olof Zetterqvist
Gothenburg 2023
ISBN 978-91-7905-897-5

Doktorshavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5363
ISSN 0346-718X

Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 (0)31 772 1000

Cover:
Illustration of how sensitive a deep neural network is towards mislabelled training data. Additional details may be found on page 20.

Typeset with LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

On the Robustness of Statistical Models: Entropy-based Regularisation and Sensitivity of Boolean Deep Neural Networks

Olof Zetterqvist

Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Models like deep neural networks are known to be sensitive towards many different aspects of noise. Unfortunately, due to the black-box nature of these models, it is in general not known why this is the case. Here, we analyse and attack these problems from three different perspectives. The first one (Paper I) is when noise is present in training labels. Here we introduce a regularisation scheme that accurately identifies wrongly annotated labels and sometimes trains the model as if the noise were not present. The second perspective (Paper II) studies the effect of regularisation in order to reduce variance in the estimation. Due to the bias-variance trade-off, it is a hard task to find the appropriate regularisation penalty and strength. Here we introduce a methodology to reduce bias from a general regularisation penalty to make the estimation closer to the true value. In the final perspective (Paper III), we study the sensitivity that deep neural networks tend to have with respect to noise in their inputs, in particular, how these behaviours depend on the model architecture. These behaviours are studied within the framework of noise sensitivity and noise stability of Boolean functions.

Keywords: Deep neural networks, Regularisation, Noisy labels, Boolean functions, Noise sensitivity, Noise stability

List of publications

This thesis is based on the work represented by the following papers:

I. **Zetterqvist, O.**, Jörnsten, R., Jonasson, J. Regularisation via observation weighting for robust classification in the presence of noisy labels. *Submitted*

II. **Zetterqvist, O.**, Jonasson, J. Entropy weighted regularisation, a general way to debias regularisation penalties. *Manuscript*

III. Jonasson, J., Steif, J, **Zetterqvist, O**. Noise Sensitivity and Stability of Deep Neural Networks for Binary Classification. *Submitted*

Author contributions

I, II. Responsible for simulations and development of the methodology. Also contributed to the manuscript and some of the theory.

III. Contributed to the development of the theory and the manuscript.

Acknowledgements

First and foremost, I would like to express my deepest appreciation to my supervisor Johan Jonasson for all the good discussions, excellent supervision, and for always keeping my curiosity alive! I would also like to thank my co-supervisor Rebecka Jörnsten for helpful inputs and ideas. Also, thanks to Jeffrey Steif for great collaboration and the many discussions from which I have learned very much!

I would also like to show my appreciation to all my colleagues and friends at Mathematical Sciences in Gothenburg: Anna, Carl-Joar, Clemens, David, Edvin, Erik, Felix, Gabrijela, Helga, Henrik, Jimmy, Johan, Juan, Linnea, Mattias, Mikael, Oscar, Oskar, Selma, Tobias, Vincent and many more. Without you, my time at the department would not have been as bright and fun! I would also like to thank Annika, Serik and Petter for helping me when things were extra hard and Aila, Elisabeth, Marie and Marija for all the help when I was confused and didn't know how things should be done at the department.

I would also like to thank the Wallenberg AI and Autonomous Systems and Software Program (WASP) for creating such a great community where I meet fellow friends and extended my knowledge in AI.

Finally, I would like to thank my family and friends for their endless support and for always being there in all the ups and downs.

Contents

Abstract **iii**

List of publications **v**

Acknowledgements **vii**

Contents **ix**

1 Introduction **1**

2 Background **3**

2.1 Neural network models 4

2.2 Regularisation penalties and the bias-variance tradeoff 13

2.3 The effect of label noise and how to train with it 18

2.4 Important concepts for Boolean functions 22

3 Summary of papers **27**

3.1 Paper 1 27

3.2 Paper 2 30

3.3 Paper 3 32

4 Discussion **39**

Bibliography **43**

1 Introduction

We, as individuals and organisations, take decisions based on data all the time, and the need for automatic methods increases by each day. Therefore much research is put on methodologies to set up a model, let it learn from data and then apply its knowledge to new unseen examples. This can, for example, be learning to separate between malignant and nonmalignant cancer cells or detecting spam mail in your inbox. Depending on the task at hand, there is an extensive set of models and methodologies that can be used. In recent years, one of the most discussed model architectures is the deep neural network (DNN), mostly due to the many success stories. The model especially shines in image processing, natural language processing and signal processing but has been showing impressive results in other areas as well.

The first ideas behind the neural network were first introduced in (McCulloch and Pitts, 1943), and the model is structure-wise often compared to the architecture of the brain. Even if the idea has been known for a long time, it was not until much later that it became as popular as today. The main reason for the delay was the need for huge computational power to handle the size of the model and data. However, the model has become much more available for practical use with today's exceptional computation units, such as the graphical processing unit (GPU) and tensor processing unit (TPU), which is optimised for fast matrix multiplications, something that is frequent during training and evaluation of DNNs.

Even if DNNs have shown to be very powerful tools, they have their weaknesses. They require some expertise to master both during training and during inference of data not seen in the training process. The reasons for this are many. It could be that there are errors in the training data that are unknown to the developer, which has a considerable impact on the model. There would also be errors in the data not used for training which make the model not recognise it as a typical input that it is trained to recognise.

In general, it is not known why DNNs work as well as the do. One reason is their large flexibility and ability to adapt to complex patterns. Still, this does not answer the question of why the model tends to generalise well especially not when considering that the large flexibility leads to the model can be very sensitive towards errors in training and evaluation data. This is why, even if we know the internal computations, the model is considered a black-box model.

Fortunately, there are ways to make DNNs and similar models more robust. Here the focus is on three different aspects of model robustness. The first one is robustness towards errors in training data, in particular errors in labels which can be devastating in a classification setting. This is discussed in Section 2.3, which shows how noisy labels affect the model and discusses what can be done to reduce the effect of adverse data points. This section lays out the background of Paper 1, which introduces a new, easily implemented methodology to identify unreliable training examples and adapt the training thereafter. Another way to make training more robust is to limit the search space during training by introducing a regularisation penalty. By doing so, the hope is to get a model that generalises better to new observations. This is discussed in more detail in Section 2.2, which summarises the properties of regularisation penalties and what alternative versions there are, which is the main background of Paper 2. Finally, we study robustness regarding the stability towards errors in the model input, as small errors can lead to devastating output results. Paper 3 investigates typical DNN architectures from the perspective of Boolean functions and shows how the properties of the model depend on the model architecture. The relevant theory and concepts of Boolean functions are discussed in Section 2.4. Before going into these areas, we need to discuss the models at hand, which is done in Section 2.1. Finally, in Chapter 3, there is a short summary of the papers followed by a discussion in Chapter 4.

2 Background

Let $(\mathcal{X}, \mathcal{Y})$ be the data domain of the task at hand where $\mathcal{X}$ is the input domain, and $\mathcal{Y}$ is the output domain. For example, one can think of $\mathcal{X}$ as a set of all possible images and $\mathcal{Y}$ as the set of labels these images can be associated with. Another example is to think of $\mathcal{X}$ as the set of all stock market values in the latest months and $\mathcal{Y}$ as the set of all possible values of the stock market tomorrow. The assumption is that there is some unknown function f that maps each element in $\mathcal{X}$ to $\mathcal{Y}$. The goal is, based on observations $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, $i = 1, \dots, n$, to make inferences about the spaces $\mathcal{X}$ and $\mathcal{Y}$. Here each observation x_i is assumed to be sampled from some distribution over $\mathcal{X}$ and y_i sampled from

$$y_i = f(x_i) + \epsilon_i \tag{2.1}$$

where ϵ_i is some noise, possibly dependent on x_i. The inference is made by setting up a model space $\mathcal{F}$ with models $\hat{f}(x; \theta)$ and, based on the observations, find the parameters $\theta \in \Theta$ that make $\hat{f}$ best approximate f. Here Θ is the space of all possible parameters that could describe $\hat{f}$, and $\mathcal{F}$ is induced by this variation in θ. There are naturally a few questions that one needs to consider. First, how does one measure the similarity between $\hat{f}$ and f when f is unknown? The best thing we can do is to base the similarity on the difference between observed data and predictions done by the model $\hat{f}$. Secondly, to represent f fully, we need samples from the whole space $\mathcal{X} \times \mathcal{Y}$, which is not possible in practice. Is there some family of functions $\mathcal{F}$ that makes the generalisation better for new observations making it easier to get away with fewer observations? Also, given the function family $\mathcal{F}$, finding the optimal θ is not trivial, and the methods used depend heavily on the properties of data and what family of functions one optimises over. In the following sections, we will review two common function families $\mathcal{F}$, what type of data they tend to handle well, some problems these models come with, and what can be done to make these problems less noticeable.

2.1 Neural network models

One commonly used family of models are the deep neural networks (DNN). The first ideas behind artificial neural networks were introduced by McCulloch and Pitts (1943) and have since then become a popular tool in many data science applications. The reason for this is the great flexibility of these models and the increment of computational power, making it feasible to train them. However, even if they show very impressive results, their flexibility leads to some serious challenges in the form of overfitting. Here, we consider three main models, the fully connected neural network (FCNN), the linear model which is a special case of the FCNN and the convolutional neural network (CNN). Here follows a short overview of these models.

2.1.1 Linear models and the fully connected neural network

One of the most common neural network architectures is the fully connected neural network (FCNN). The structure of the FCNN is highly inspired and compared with the architecture of the brain, which is built by billions of neurons. Due to this analogy, the components in the FCNN are usually called neurons as well. In the model, these neurons can be seen as small computational building blocks used to build larger models. Even if one specific neuron cannot solve complex problems, their union can achieve remarkable things. The simplest possible case of the FCNN is the linear model which corresponds to a single neuron. Given an input vector $x_i = (x_{i,1}, \ldots, x_{i,d})$ the output from the linear model is given by

$$f(x_i;\theta) = \sigma\left(\sum_{j=1}^{d} \theta_j x_{i,j} + \theta_0\right) \tag{2.2}$$

where $\theta = (\theta_0, \ldots, \theta_d) \in \Theta$ are fixed parameters determining the behaviour of the model. Here Θ is considered the space of all feasible parameters. In the neural network literature, the function σ is called the activation function and transforms the input to the desired output domain $\mathcal{Y}$. In linear models, one can simply take $\sigma(x) = x$ giving a linear regression model typically used if $\mathcal{Y} = \mathcal{R}$, or $\sigma(x) = \frac{1}{1+e^{-x}}$ giving a logistic regression model which typically is used if $\mathcal{Y} = \{0, 1\}$. A logistic regression model is typically used in a classification setting where one wants to determine the separation between two classes in data. Graphically the linear model or a neuron can be described as a directed

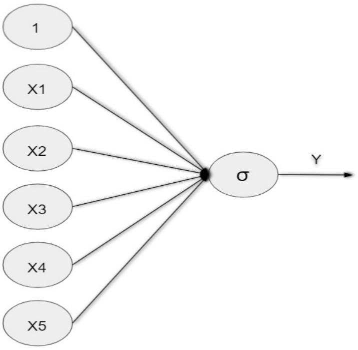


Figure 2.1: An illustration of the mathematical modulation of a neuron represented in Equation 2.2. The neuron takes in several inputs $x_{i,j}$, and processes these by multiplying them with the parameters θ_j and summing them up. The final output is then given by the function σ of the weighted sum.

acyclic graph (DAG) where each neuron corresponds to one local calculation using (2.2). The corresponding DAG can be seen in Figure 2.1.

Even though ordinary regression and logistic regression models perform well in many situations, they have many limitations. The most significant one is of course that they only can model linear behaviours in data. To make the model more flexible and able to handle more complex data, one has to extend the model space. The idea of the FCNN is to combine many neurons creating a larger DAG, giving the model a larger model space and flexibility. The typical way is to have many neurons grouped in many layers. Within each layer, each neuron works in parallel with the same input as the other neurons in that layer, but different neurons have different θ. The output produced is then sent to the next layer which considers the outputs of the previous layer as input. This creates a recursive structure with a DAG illustrated in Figure 2.2. Typically this is iterated by a fixed number of layers T and with a fixed number of neurons $d^{(t)}$ at each layer $t \in \{0, \dots, T\}$. $d^{(0)}$ is considered the input dimension of the model. Let us now define a FCNN formally.

Let $\mathbb{X}^{(t)}$ be a column vector with elements corresponding to the output values from neurons at layer t. Each element in $\mathbb{X}^{(t)}$ corresponds to the output from one specific neuron in contrast to the linear model case where we only needed one vector $\theta \in \mathcal{R}^{d+1}$ to calculate the output. The function that takes us from $\mathbb{X}^{(t-1)}$ to $\mathbb{X}^{(t)}$, is now parameterised by a matrix $\theta^{(t)} \in \mathcal{R}^{d^{(t)},d^{(t-1)}}$ and vector $b^{(t)} \in \mathcal{R}^{d^{(t)}}$. Given $\theta^{(t)}$, $b^{(t)}$ and $\mathbb{X}^{(t-1)}$, $\mathbb{X}^{(t)}$ is given by

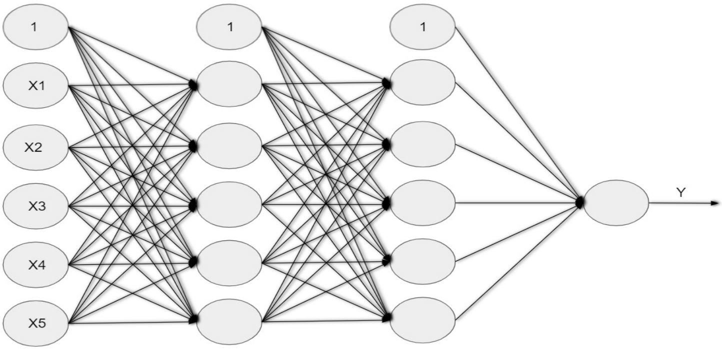


Figure 2.2: An illustration of a neural network. Each circle represents one neuron shown in Figure 2.1. By stacking these together creating a neural network, we get a more flexible model that can approximate more complex functions.

$$\mathbb{X}^{(t)} = \sigma\left(\theta^{(t)}\mathbb{X}^{(t-1)} + b^{(t)}\right). \tag{2.3}$$

This process is now iterated for each layer until the final layer T. The total model is thus parameterised by the sequence of matrices $\theta = (\theta^{(1)}, b^{(1)}, \ldots, \theta^{(T)}, b^{(T)})$. The layer at $t = 0$ is often called the input layer, and $\mathbb{X}^{(0)}$ corresponds to the model's input. The layers at $t = 1, \ldots, T-1$ are called the hidden layers and layer T is the output layer. As in the linear model, the chosen σ at the final layer is chosen to correspond to the output domain $\mathcal{Y}$. If a multidimensional output is desired one can add additional neurons in the final layer. One example is if one has a multi-class classification problem with C classes. Then typically $d^{(T)} = C$ where each output neuron models the probability of belonging to the corresponding class. Here the most common activation function at the final layer is the softmax function

$$\sigma(x) = \frac{e^{-x}}{\sum_j e^{-x_j}}.$$

For the hidden layers, σ can be chosen more arbitrarily. Common choices are the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$, tangens hyperbolicus function $\sigma(x) = (e^x - e^{-x})/(e^x + e^{-x})$ or the ReLU function $\sigma(x) = \max(0, x)$.

2.1.2 The convolutional model

An alternative model to the FCNN is the convolutional neural network (CNN) first introduced in (LeCun et al., 1998). One disadvantage of the FCNN model is that it is not very good at handling translations in input data. Since each weight in θ corresponds precisely to its location in the model, a shift in the input domain leads to significantly different output results. The model is not translation invariant. There are many situations where it is desirable to have a translation invariant model. For example, when working with image data, the object location in the image should not affect the output result. Here the CNN model is of great use.

The CNN is very similar to the FCNN in that it is constructed by concatenating layers to create the full deep model. However, while each layer in the FCNN consists of a matrix product, summation and an activation function, each layer in the CNN instead consists of a convolution between the input and some predetermined matrices, so called filters. These filters correspond to the parameters θ that determine the model behaviour. We will start to consider the most simple case of a CNN where the input is a vector and where there is only one filter between each layer. The one-dimensional model is also the version with the most theoretical focus in this thesis. After that, we will generalise the concept to more filters and complex inputs.

Let $\mathbb{X}^{(0)} \in \mathcal{R}^{d^{(0)}}$ be the input to the CNN, $\theta^{(1)} \in \mathcal{R}^k$ a filter of size k and $b^{(1)} \in \mathcal{R}$ the bias. The output of the convolutional layer is given by

$$\mathbb{X}^{(1)} = \sigma(Z^{(1)} + b^{(1)})$$

where $Z^{(1)}$ is the convolution between $\mathbb{X}^{(0)}$ and $\theta^{(1)}$. The idea behind the convolution operator is to construct $Z^{(1)}$ index by index where a fixed index in $Z^{(1)}$ is determined by a weighted sum of k spatially nearby values in $\mathbb{X}^{(0)}$. The weights in the sum are given by $\theta^{(1)}$. Typically is k much smaller than the length of $\mathbb{X}^{(0)}$. The next element in $Z^{(1)}$ is determined by a new weighted sum with the same weights $\theta^{(1)}$ but where the scope of input locations in $\mathbb{X}^{(0)}$ has shifted some distance s, the so called stride of the convolution. This process is repeated until the filter has travelled through the whole input. The advantage of the convolution is that it naturally captures the local information in the input such that correlations between inputs spatially close to each other are more easily captured. In Figure 2.3, there is an illustration of the one-dimensional convolutional operator. Notice that the size of the output Z is determined by the input size $d^{(0)}$, the filter size k and the stride s and is given by $(d^{(0)} - k)/s + 1$.

The activation function σ is typically chosen in the same way as for the FCNN.

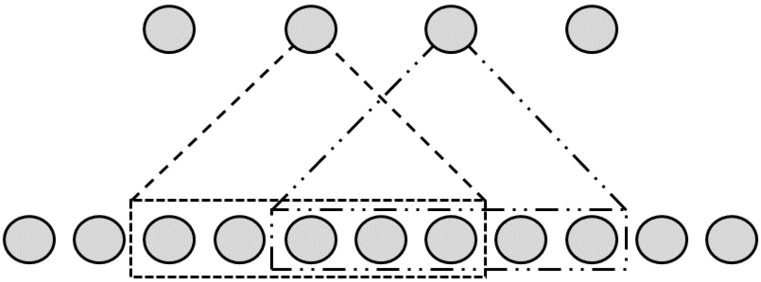


Figure 2.3: An illustration of the convolution operator with a one-dimensional input string. Each circle in the bottom row represent one input value while each circle represent one value in Z. The corresponding rectangles correspond to the scope of values that are considered to determine the following value in Z when the filter size is 5 and stride is 2.

This process is then iterated until a desired depth T resulting in the model output. In many cases, one often adds fully connected layers as well after the convolutional layers.

The convolutional layer can easily be extended to data structured as a matrix or a higher dimensional tensor. For example, consider a coloured image $\mathbb{X}^{(0)}$ as input, represented as a tensor with width W, height H and C colour channels. Consider also a filter $\theta^{(1)}$ as a tensor with width w, height h and C colour channels. Then the convolution between $\mathbb{X}^{(0)}$ and $\theta^{(1)}$ is done similarly to the one-dimensional case, but where the filter slide over both the image height and width. At each position, the weighted majority gives the output value over all values covered by the filter. Notice that all colour channels are always considered for each output value. This results in an output Z with height $(H-h)/s_h+1$ and width $(W-w)/s_w+1$ where s_h and s_w is the stride in the corresponding directions. The number of colour channels in the output is one. Since each filter outputs one output channel, one uses more filters between each layer to get an output with multiple channels. This way, given a specific layer one can construct outputs with an arbitrary number of colour channels by chancing how many filters one use at that specific layer. The total model is then iterated similarly to the one-dimensional case. In Paper 3, theoretical aspects of the one-dimensional CNN are studied in more detail.

2.1.3 Loss functions and training

Let (X, Y) be random variables on $\mathcal{X} \times \mathcal{Y}$. The goal is to find the optimal setting θ^* such that $\hat{f}(x;\theta)$ approximates $f(x)$ as closely as possible. To say that $\hat{f}(x;\theta)$ approximates $f(x)$ does, of course, depend on what metric is used to measure the similarity. To measure the similarity between $\hat{f}$ and f, we need some sort of metric $D(\hat{f}(X;\theta), f(X))$ that captures the task at hand. In a regression setting, the most commonly used metric is the expected value of the norm distance squared

$$D(\hat{f}(X;\theta), f(X)) = \frac{1}{2}\mathrm{E}_X\left[||f(X) - \hat{f}(X;\theta)||_2^2\right]$$

where $||\cdot||_2$ is the L2 norm. The estimate $\hat{\theta}$ is now given by the argument that minimises this metric, hence

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{2}\mathrm{E}_X\left[||f(X) - \hat{f}(X;\theta)||_2^2\right].$$

In a classification setting, this metric is not very suitable. In that case, a more commonly used comparison between $\hat{f}$ and f is the cross entropy (which technically is not a metric)

$$D\left(\hat{f}(X;\theta), f(X)\right) = \mathrm{E}_X\left[-f(X)\log(\hat{f}(X;\theta))\right]$$

giving rise to the minimisation objective

$$\hat{\theta} = \arg\min_{\theta} \mathrm{E}_X\left[-f(X)\log(\hat{f}(X;\theta))\right]$$

If the model space is limited to linear models, both these estimates are unambiguous. However, this is not true for deep neural networks where the same function can be represented by many different settings of θ. A simple way to see this is by perturbing the neurons within one layer and switching the weights accordingly. Then the weights θ would be different, but the corresponding function would be the same.

One problem with the above minimisation tasks is that the distributions of X and Y and the function f are generally unknown. Therefore, one has to estimate the expected values based on observed data $(\mathbb{X}, \mathbb{Y}) = (x_1, y_1), \ldots, (x_n, y_n)$

sampled from their corresponding random variables (X, Y). This means that we use approximations for the corresponding minimisation objectives. Since both are expected values, this is done with an average overall observation. In the regression situation, this gives the objective

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{2n} \sum_{i=1}^{n} ||y_i - \hat{f}(x_i; \theta)||_2^2 \tag{2.4}$$

and in the classification setting

$$\hat{\theta} = \arg\min_{\theta} -\frac{1}{n} \sum_{i=1}^{n} \left(y_i \log \left(\hat{f}(x_i; \theta) \right) + (1 - y_i) \log \left(1 - \hat{f}(x_i; \theta) \right) \right). \tag{2.5}$$

Since the estimates are now based on samples from $X \times Y$, it is not reasonable to assume that $\hat{f}$ will be the same as f everywhere, even if the model space contains the true data generating model, f. This is especially true if the noise levels ϵ are large. A common phenomenon is that the model learns the noise and becomes more complex than the true data structure. This phenomenon is called overfitting, which will be discussed in more detail in the next section.

Notice that in both the regression and classification problems, the minimisation problem is of the form

$$\hat{\theta} = \arg\min_{\theta} L(\hat{f}(\mathbb{X}; \theta), \mathbb{Y}) = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(\hat{f}(x_i; \theta), y_i). \tag{2.6}$$

where $L(\hat{f}(\mathbb{X}; \theta), \mathbb{Y})$ is typically refereed to as the loss function and, $\ell(\hat{f}(x_i; \theta), y_i)$ represent the loss contribution from data point (x_i, y_i). This form will be of importance, as we will see later.

Depending on the data and the model at hand, the minimisation problem can be hard to solve analytically, meaning we need to turn to numerical methods. The only situation in this thesis where we can find an analytical solution is when we have a linear regression model. Otherwise the most common method is to use variants of gradient descent where the gradients are obtained through some backpropagation procedure (Hecht-Nielsen, 1992).

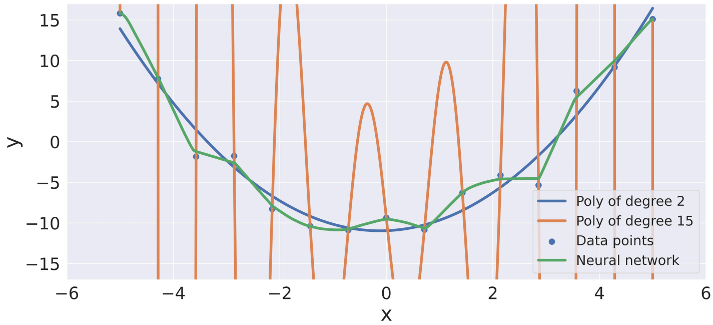


Figure 2.4: An illustration of the generalisation properties of neural network. Here we see data point generated by $y_i = x_i^2 - 10 + 3\epsilon_i$ where ϵ_i is iid standard Gaussian. Based on these, three models are fitted, a second-degree polynomial, a 15-degree polynomial and a deep neural network consisting of 101 301 parameters. The optimal polynomial is found analytically, while the neural network is trained using gradient descent.

2.1.4 Overfitting and why neural network tends to generalise well.

Overfitting, which loosely speaking is when the model $\hat{f}(x;\theta)$ is far too complex in contrast to the amount of data given, leads to bad generalisation on new observations. Figure 2.4 shows an example of this, where three models have been trained on 15 data points generated by the function $y_i = x_i^2 - 10 + 3\epsilon_i$ where ϵ_i is iid standard Gaussian. Two of these models are polynomials, one of second order and one with an order of 15. The higher-order model goes through every data point perfectly, but it does not capture the overall structure of the data. It has overfitted, making it generalise poorly.

Conversely, the second-order polynomial is not that flexible and can not express such complicated functions. In this case, limiting the model's flexibility by not using too many tunable parameters seems to be a good idea. However, in the figure, one can also see a DNN consisting of many more tunable parameters than both polynomial models. Still, that model does not overfit as much, even if it still fits almost every point perfectly.

The property that DNNs tend not to overfit is one of many reasons behind the many success stories of neural networks. The DNNs models have been showing that they can handle many different kinds of data and are, in many fields, state of the art. In general, it is not well known why deep neural networks perform as well as they do, but one reason is the enormous flexibility of the model. In fact, (Cybenko, 1989) show that neural networks are universal approximators, i.e.

for any arbitrary function, there is an architecture of a neural network such that the model approximates the function arbitrarily close. This property gives rise to a natural question; If neural networks have such extensive flexibility, why do they not tend to overfit, even if they are highly over-parameterised? As it turns out, as shown in (Belkin et al., 2019), the generalisation property changes behaviour as the number of tunable parameters increases. First, models tend to overfit with data if the number of parameters is very large. However, this is only true until a certain point. If one adds even more parameters, the generalisation property of the model tends to overfit less. One can wonder why this is the case. One explanation is given in (Pérez et al., 2018). Here the authors study the parameter function map $\theta \to \hat{f}(x;\theta)$ of deep neural networks and show that by randomising θ, which typically is done at the start of training, this map is heavily biased towards simple functions. So even if the neural network can represent very complex functions describing the most strange behaviours, they tend to start from a position where the model is not that complex. However, these results only hold for the initialisation of the model. The next step is to ask oneself what happens during training. Will the model move away from simple functions to more complex ones? In (Mingard et al., 2021), the authors look at this question. They show that during training, the parameters θ tend not to move that far away from their initial state, still representing a "simple" function. A similar result to this is shown in (Jacot et al., 2018; Zou et al., 2020; Du et al., 2018), where they show that for a very deep DNN, the hidden layers can be approximated as a kernel, similar to a kernel method for linear models, and during training θ tend to not move that far from its initial state. This preserves the model's simplicity.

These results sound promising as long as observed data follows the distribution that the model is set to approximate. However, several things can make the model perform poorly anyway. For example, noise or errors can make the data unnecessarily complex. Examples are if the noise term ϵ_i in 2.1 is very large so it overshadows f or if samples from a completely different distribution, called outliers, have been added to the observations. This can force the parameters θ to move further during training, potentially leading to a region where the function is unnecessarily complex. This, in combination with the fact that DNN is so flexible, makes them extra vulnerable. To prevent this from happening, a common approach is to limit the search space of the training algorithm and stop it from reaching complex functions. The following sections will discuss how such limitations can be implemented.

2.2 Regularisation penalties and the bias-variance tradeoff

In this thesis, we consider two main ideas to prevent the model $\hat{f}(x;\theta)$ from reaching too complex functions during training. One of them is to, in some way, turn off noxious data points that we believe are outliers or have been annotated incorrectly. This can for example be if some data has been mislabelled in a classification setting. These methods will be discussed in more detail in section 2.3. The other methodology discussed here is to limit the search space during training such that the parameters θ can not express complex models. A common way to do this is by adding a penalty to the loss objective 2.6 that penalises complex models. What follows is a description of how this is done in the setting of linear models. However, many of the concepts immediately generalise to neural networks.

Let θ^* be the true best parameters θ that makes $\hat{f}(x;\theta^*)$ approximate f as closely as possible. For the rest of this section, assume that $\hat{f}(x;\theta)$ is a linear model and that the amount of training data is larger than the number of unknown parameters. This ensures that 2.6 is convex and, more importantly, θ^* is unique. Assume also that $\mathcal{Y} = \mathcal{R}$. This means that each data point $x_i \in \mathcal{R}^p$ can be seen as a row vector and $\theta = (\theta_1, \dots, \theta_p) \in \mathcal{R}^p$ as a column vector, where p is the number of covariates. However, many concepts presented here immediately extend to DNNs, higher dimensional outputs and more unknown parameters. Let $\hat{\theta}$ be the estimated parameters θ based on the observations $(x_1, y_1), \dots, (x_n, y_n)$ that we have at hand. Since each data point is a random variable, $\hat{\theta}$ is also a random variable. Optimally one would have an estimator $\hat{\theta}$ such that

$$\mathrm{E}_\theta\left[(\hat{\theta} - \theta^*)^2\right] = \left(\mathrm{E}_\theta[\hat{\theta} - \theta^*]\right)^2 + \mathrm{E}_\theta\left[(\hat{\theta} - E_\theta[\hat{\theta}])^2\right] \tag{2.7}$$

is as small as possible. Both estimates 2.4 and 2.5 are such that 2.7 converges to zero as n increases. However, in practice, n is fixed and sometimes small compared to the dimension of θ, making the estimate not necessarily optimal due to overfitting. Fortunately, a few things can be done. As seen in Equation 2.7, the error depends on two terms, the first one $(\mathrm{E}_\theta[\hat{\theta} - \theta^*])^2$ corresponds to the bias squared and measures how close the model comes to the true optimal model on average. The second term $\mathrm{E}_\theta[(\hat{\theta} - E_\theta[\hat{\theta}])^2]$ corresponds to the variance of the estimated model. A common technique to reduce the variance is to add a penalty to the optimisation objective. This is typically done by adding a regularisation term $\lambda g(\theta;\phi)$ to the loss function where λ determines the strength of the regularisation and ϕ is the set of hyperparameters determining the shape of g. Expanding the estimator 2.6, the regularised estimation of θ is thus given by

$$\hat{\theta} = \arg\min_{\theta} \left[\frac{1}{n} \sum_{i=1}^{n} \ell(\hat{f}(x_i;\theta), y_i) + \lambda g(\theta;\phi) \right]. \tag{2.8}$$

The regularisation term $g(\theta;\phi)$ should only have one unique local minimum, located at $\theta = \mathbf{0}$, and be quasi-convex. This makes the objective more biased towards smaller parameters θ, limiting the solution space and reducing the variance. To see this, one can show that as long as $g(\theta;\phi)$ is quasi-convex, then for all $\lambda > 0$ there is a $t > 0$ such that the objective in 2.8 can be formulated as,

$$\begin{aligned} \hat{\theta} = \arg\min_{\theta} \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \hat{f}(x_i;\theta) \right)^2 &= \arg\min_{\theta} \frac{1}{2n} \sum_{i=1}^{n} (y_i - x_i\theta)^2 \\ \textit{subject to } g(\theta;\phi) \leq t. \end{aligned} \tag{2.9}$$

By setting a small t, corresponding to a large λ, one can force θ to be within lower level curves of $g(\theta;\phi)$, which reduces the search room and hence also the variance.

The most well-known choices of g are the Ridge penalty

$$g(\theta) = \frac{1}{2}||\theta||_2^2 = \frac{1}{2}\sum_j \theta_j^2$$

and the Lasso penalty (Tibshirani, 1996),

$$g(\theta) = ||\theta||_1 = \sum_j |\theta_j|$$

which both give convex constraints in 2.9.

So the technique of adding a regularisation penalty works very well at reducing the variance of the estimate. However, by limiting the search space, we also create a bias in the estimates. If θ^* is not in the set $g(\theta;\phi) \leq t$, θ^* is not a possible estimate, and we have created a bias. Thus, the bias will increase with a smaller t or larger λ. Since both terms in 2.7 depend on λ, one has to tune λ such that the model generalises as well as possible and hence minimises 2.7. This is called the bias-variance tradeoff. Figure 2.5 illustrates this tradeoff in

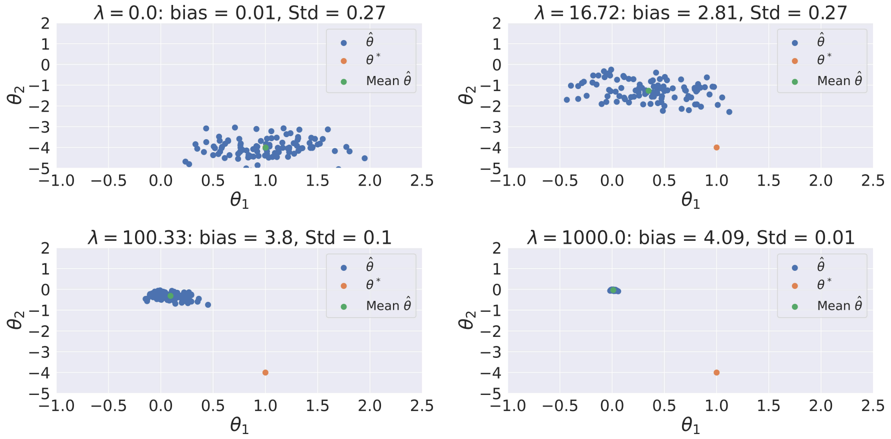


Figure 2.5: An illustration of the bias-variance tradeoff for the ridge regularisation method on a two-dimensional linear regression setting. Data is sampled from the model $Y_i = x_i\theta^* + \epsilon_i$ there $x_{i,1}, x_{i,2}, \epsilon_i \sim N(0,1)$. The true parameters are $\theta^* = (1,-4)^T$. In each λ setting, 100 datasets are sampled and for each one, θ is estimated according to $\hat{\theta} = (\mathbb{X}^T\mathbb{X} + \lambda \boldsymbol{I})^{-1}\mathbb{X}^T\mathbb{Y}$. Based on each estimate $\hat{\theta}$, the mean estimate is also calculated. The main takeaway is that when $\lambda = 0$, there is a large variance for estimates θ, but the mean estimate is very close to the true parameter θ^*. However, as we increase λ, the variance decreases, but each estimate is moved closer to the origin, increasing the bias.

a two-dimensional setting using ridge regularisation in a regression setting. The illustration shows that when the regularisation strength λ is small, the bias is small but has a large variance. However, by increasing λ, the estimations $\hat{\theta}$ shrink towards the origin, reducing the variance and increasing the bias of the estimates. The behaviour of the tradeoff between bias and variance depends on the regularisation term $g(\theta; \phi)$. By choosing different $g(\theta; \phi)$, one could get a more suitable tradeoff making the choice of regularisation important.

2.2.1 Properties of good regularisation penalties and ways to achieve them.

This section lays out the main background for Paper 2. The Ridge and Lasso penalties are good at limiting the expressfullness of the model. However, they come with a few undesired properties in the form of large additional bias. For g to be considered "good", the community has identified several characteristics that it is good if g fulfils. In Fan and Li (2001), they list a few properties.

The first one is **unbiasedness**, meaning that g should not introduce any additional bias to the estimate $\hat{\theta}$. Since the estimates of θ are unbiased when there is no regularisation at all, setting $g(\theta; \phi) = 0$ would give us unbiasedness. In general, unbiasedness is hard to achieve for all θ if a regularisation penalty is used. However, a penalty should give as small bias as possible, especially for large θ. As a rule, we will say that the estimator introduces a small bias if for all j, the bias of $\hat{\theta}_j$ tends to zero as $|\theta_j^*|$ increases.

The second property is **sparsity**, meaning that g is a thresholding rule i.e. that if θ_j is very small, it should be estimated as zero. This gives a sparse estimate of θ and thus reduces model complexity.

The final property listed is **continuity**, meaning that the estimates θ should be continuous with respect to changes in the training input x_i and y_i. If this were not the case, it could be that the resulting estimator $\hat{\theta}$ is not able to reach the correct value θ, introducing a bias in the estimates.

In addition to these common goals, some additional ones concern what happens as the number of observations grows to infinity. Two common goals here are:

The estimator should be **consistent**, meaning that the estimated parameters should converge in probability to the true parameters as n goes to infinity. Note that this includes cases when λ depends on n. Typically $\lambda \to 0$ as n increases.

Finally, the estimator should be **sign consist**, meaning that the sign of the estimated θ should converge to the sign of the correct parameters as the number of observations goes to infinity. Here the sign is defined to be zero if the input is zero.

While it may sound hard to fulfil all these properties, several well-known methods achieve some or most of them. One family of regularisation penalties is the Bridge penalty

$$g(\theta; \alpha) = \sum_j |\theta_j|^\alpha \tag{2.10}$$

where $\alpha > 0$ is a hyperparameter determining the shape of the penalty (Frank and Friedman, 1993). Special cases are when $\alpha = 2$ when we get the Ridge penalty and $\alpha = 1$ when we get the Lasso penalty. The Lasso estimator is well known to give sparse and continuous estimates. By allowing λ to decrease with n, (Knight and Fu, 2000) show that the lasso estimator is consistent under mild conditions. This relies on λ decreases at a sufficient rate. Additionally, (Zhao and Yu, 2006) shows that the Lasso can be sign consistent with the correct choice of λ. However, for a fixed λ, Lasso is well known to be biased with a bias that increases linearly with λ. The Ridge estimator provides neither sparsity nor unbiased estimates. The more general Bridge estimator can be consistent (Knight and Fu, 2000), again with a λ depending on n, and is continuous if $\alpha > 1$. However, it is only sparse and signs consistent if $\alpha \leq 1$ Huang et al. (2008), and it is no longer continuous if $\alpha < 1$. Summarising, in the Bridge family, the Lasso estimator is the regularisation penalty that fulfils most of the properties listed above.

In many cases, when the number of covariates is large, the Lasso estimator is a good choice. However, to reduce the estimator's bias, alternative versions with a smaller bias have evolved. The most common ones are the Adaptive Lasso estimator (Zou, 2006), and the Smoothly Clipped Absolute Deviation estimator (SCAD) (Fan and Li, 2001). The Adaptive Lasso estimator is based on the idea of using a weighted lasso objective

$$\hat{\theta} = \arg\min_{\theta} \left[\frac{1}{2n} \sum_i (y_i - x_i\theta)^2 + \lambda \sum_j \omega_j |\theta_j| \right] \tag{2.11}$$

where ω_j is the weight corresponding to covariate θ_j. This is not a unique idea and has been investigated in other ways as well (Jung, 2011; Bergersen et al., 2011; Zhang, 2011; Javanmard et al., 2018; Bellec and Zhang, 2019). In Adaptive Lasso, the estimate $\hat{\theta}$ is found in a two-step procedure. First, one estimates θ without any regularisation. Let us call this estimate $\tilde{\theta}$. Then based on $\tilde{\theta}$ one calculate the weights ω_j as $\omega_j = 1/|\tilde{\theta}_j|^{\alpha}$ where α is a hyper parameter. The estimate $\hat{\theta}$ is then given by minimising 2.11.

The idea behind the SCAD estimator is to work similarly to the Lasso estimator for small θ but with a reduced penalty for larger θ. This is done with the penalty $g(\theta, a) = \frac{1}{\lambda} \sum_j \tilde{g}_{\lambda}(\theta_j, a)$ where a is a hyperparameter larger than 2 and

$$\tilde{g}_{\lambda}\left(\theta_j; a\right) = \begin{cases} \lambda|\theta_j| & \textit{if}\, 0 \leq |\theta_j| \leq \lambda \\ -\frac{|\theta_j|^2 - 2a\lambda|\theta_j| + \lambda^2}{2(a-1)} & \textit{if}\, \lambda \leq \theta_j \leq a\lambda \\ \frac{(a+1)\lambda^2}{2} & \textit{if}\, |\theta_j| \geq a\lambda. \end{cases} \tag{2.12}$$

Since both Adaptive Lasso and SCAD build on the idea behind the Lasso estimator, they inherit some properties of the Lasso estimator, such as consistency and sign consistency, for the right choice of $\lambda = \lambda_n$. They are also both continuous. In addition, they are both unbiased in the limit as $|\theta^*_j|$ increases,

	Unbiased	**Consistency**	**Continuity**	**Sparsity**	**Sign consistency**
OLS	Yes	Yes	Yes	No	No
Lasso	No	Yes	Yes	Yes	Yes
Ridge	No	Yes	Yes	No	No
Bridge	When $\gamma < 1$	Yes	When $\gamma \geq 1$	When $\gamma \leq 1$	When $\gamma \leq 1$
SCAD	Yes	Yes	Yes	Yes	Yes
Adaptive Lasso	Yes	Yes	Yes	Yes	Yes

Table 2.1: A table showing which properties the discussed methods fulfil.

which the Lasso estimator is not. Table 2.1 summarises the methods and their properties. Notably, SCAD and Adaptive Lasso are the only methods with all desired properties. However, SCAD and adaptive Lasso have other issues to consider. The most obvious for Adaptive Lasso is the need to estimate θ twice, once for $\tilde{\theta}$ and once for $\hat{\theta}$. This could slow the minimisation procedure if the number of covariates or data points is very large.

The SCAD algorithm does not need an additional training procedure. However, experiments in Paper 2 indicate that SCAD can be sensitive to small changes in hyperparameter settings. In Figure 2.6, we can see how different estimators behave when only one covariate and data point are present. I.e. the solution to

$$\hat{\theta} = \arg\min_{\theta} \left[\frac{1}{2}(y - \theta)^2 + \lambda g(\theta; \phi) \right]$$

for different y. What to notice here is that the Lasso, SCAD and Adaptive Lasso methods all have a plateau at $\theta = 0$ close to $y = 0$, giving rise to the thresholding rule. In addition, SCAD and Adaptive Lasso both converge to the line $\theta = y$ for large $|y|$ giving rise to less biased estimates for large parameters. This is not true for Lasso or Ridge.

2.3 The effect of label noise and how to train with it

In practice, acquiring a classification dataset requires lots of time and resources. For example, this can be that someone has to look at each data point one at a time and label them accordingly. Now, if someone would do this for thousands of data points, there will naturally be some errors, especially if data is hard to classify. By assumption, data distributions corresponding to different classes are somehow separated, meaning that an erroneously annotated label leads to the data point being very much out of distribution. Since the user assumes that each label is correct, this encourages the estimator to choose a $\hat{\theta}$ corresponding to a very complex model, which can be far from the true function f. This follows since the model is flexible enough to memorise specific data examples. In Figure 2.7, we see an illustration of how label noise can impact the training of

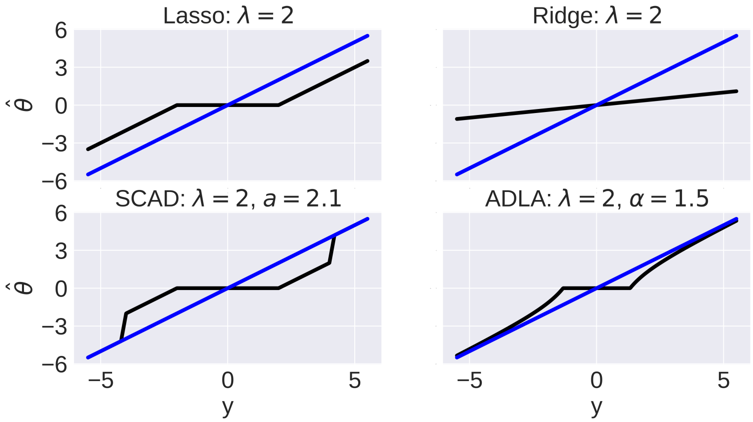
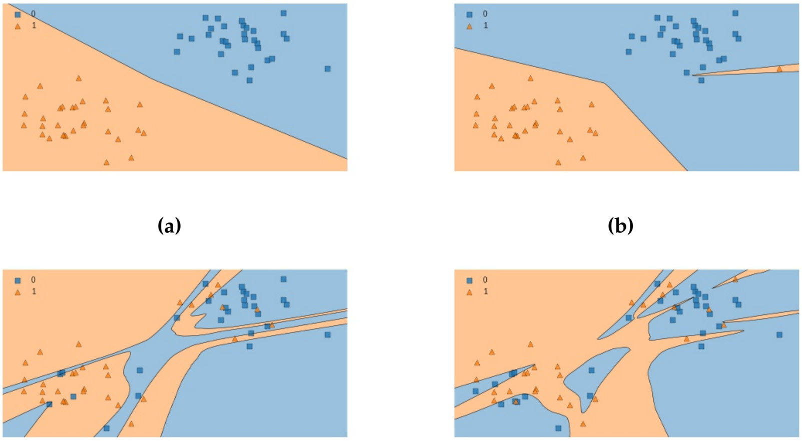


Figure 2.6: An illustration of the introduced biases produced by different methods. For a given y, $\hat{\theta}$ is given by the argument that minimise the objective $L(\theta; \phi) = \frac{1}{2}(y - \theta)^2 + \lambda g(\theta; \phi)$. This is done for the Lasso, Ridge, Adaptive Lasso and SCAD estimators.

a model. Here an FFNN with $201,801$ tunable parameters is trained on a data set with four different label noise settings; when no noise is present, one label is wrong, 20% of labels are incorrect and when 30% are incorrect. Optimally the model would behave almost the same for all noise settings, but clearly, this is not the case. This illustrates that the model's extreme capacity makes it by no means caption the true relationship between the two classes when label noise is present. It is clear that the model does not capture the relation between the classes well when label noise is present.

There is much research that shows that label noise is indeed harmful when training a DNN. (Zhang et al., 2016, 2021) show that even though DNNs tend to generalise well when data is without mislabels, they can easily fit data with random labels. They also argue that this is true under explicit regularisation and can happen even if the input data is pure noise. So by training sufficiently long, the expressed function will be so complex that it can fit pure noise. This makes the model generalise poorly to new data. Fortunately, there are many things one can do. In (Arpit et al., 2017), they show empirically that DNNs tend to learn simple patterns first, which typically is the pattern we would like the model to learn, and then memorise noise in data. This property creates a window of opportunity to stop training when it has learned the simple pattern but before it starts adapting to noise. This is called early stopping, which is widely used in many situations within deep learning today. In the context of label noise, (Li et al., 2020) show that with some assumptions on the data, like how different classes are clustered and how well separated the



(a) **(b)**

(c) **(d)**

Figure 2.7: An illustration of what could happen when one training data point is misclassified. Each image shows the classification boundaries from a neural network with 201 801 parameters trained on data with different label noise settings. In **a)** there is no label noise, in **b)** one point is mislabeled, in **c)** 20% of the labels is incorrect and in **d)** 30% of labels is incorrect.

classes are, training with gradient descent with early stopping is robust to label noise. This goes hand in hand with the discussion above in Section 2.1.4, where we noticed that the models tend to represent a simple function at initialisation. However, with errors in data, the further you train, the more the parameters θ tend to move to a more complex domain. There are, however, a few problems with early stopping. One is that the loss function you train on is never minimised, which means we lose the theoretical justification of our loss function. In addition, one needs an extra dataset that one can use to determine when to stop training.

There is an extensive set of other methods that each tries to reduce the impact of mislabelled data and somehow construct critical thinking into model training, Frénay and Verleysen (2014); Song et al. (2020), which are too many for a complete overview here. According to the latter of these papers, these methods can be categorised into four umbrella categories as follows.

The first one is the category of **Robust architectures**. This can, for example, be a noise adaption layer whose purpose is to mimic the errors done in sampling. More concretely, if the model tries to learn the probability $\mathrm{P}(y_i = k|x_i; \theta)$, the extra layer extends the modelling to

$$\mathrm{P}(\tilde{y}_i = k|x_i;\theta) = \sum_j \mathrm{P}(\tilde{y}_i = k|y_i = j;\theta)\mathrm{P}(y_i = j|x_i;\theta)$$

where $\tilde{y}_i$ is the given label, which may be wrong. One example is (Bootkrajang and Kabán, 2012), where they use this on linear logistic regression or (Goldberger and Ben-Reuven, 2017; Hendrycks et al., 2018; Sukhbaatar et al., 2015) where it is implemented on DNNs.

The next category is to use **Robust regularisation** that, in different ways, prevent the expressfullness of the model. This could mean methods such as a Lasso or Ridge penalty, or dropout, which is commonly used in deep learning. One can also imagine using label smoothing and training on convex combinations of data points. One example of this is mixup (Zhang et al., 2017) where instead of training on data points (x_i, y_i) one first does a convex combination between two data points

$$(\hat{x}_i, \hat{y}_i) = (\lambda x_i + (1-\lambda)x_j, \lambda y_i + (1-\lambda)y_j)$$

where λ is chosen randomly from a symmetric beta distribution. This encourages the model to behave linearly between data points, and errors will be smoothed out during training.

The third group is to use a **Robust loss design** which designs a loss function that is more robust to errors in labels. This is the category that the method presented in Paper 1 would be categorised into. There are several robust loss functions. For linear models, some examples are the support vector machines (Cortes and Vapnik, 1995) or the many different M-estimators for linear regression such as the Huber loss or Welsch's exponential loss (Hampel et al., 1986). There are many methods for DNN as well. One way is to extend the loss function (2.6) to a weighted sum. This leads to a new loss

$$\hat{\theta} = \arg\min_\theta \sum_{i=1}^n \omega_i \ell(\hat{f}(x_i;\theta), y_i)$$

where ω_i are the individual weights for the different data points. There are several different methods to determine ω_i. In (Gao and Fang, 2016; Gao and Feng, 2018), they extend the minimisation problem in addition to minimising over θ; they also minimise over ω_i. They use an optimisation constraint on positive weights to stop them from becoming negative. They also introduce an additional regularisation term that penalises them from going too far away from one. The minimisation problem is solved by iterating between minimising over θ and ω.

The final category uses a **Sample selection** technique, making it more probable to sample correctly labelled data for training. This can be done in many ways. One way is to sample data based on the loss values of the data and interpret data with a low loss to be more likely to be correct. One way to use this is demonstrated in (Jiang et al., 2018), where they have two networks, a student and a mentor network. The idea is that the mentor determines which data points the student should train on based on the loss of data that the mentor produces.

Of course, there are other methods as well that do not fit any of these categories. One example is (Koh and Liang, 2017), where they represent a methodology to estimate how much each data point impacted model training. Typically a data point which is mislabelled or any other outlier affects the training much more than other points. This way, one can find hazardous data points and remove them manually.

2.4 Important concepts for Boolean functions

This section will cover a small subset of the otherwise extensive theory of Boolean functions. The purpose is to give insight into the concepts used in Paper 3. First, the important concepts for general Boolean functions are introduced and in Subsection 2.4.1 there will be a discussion about the connection to previous sections. The literature on Boolean functions is quite extensive; however, the go-to sources in the area are (Garban and Steif, 2014; O'Donnell, 2014).

A Boolean function is a function f taking an input ω_n from the hypercube $\Omega_n = \{\pm 1\}^n$ and producing an output $f(\omega_n) \in \{\pm 1\}$. In the literature, a few functions are frequently occurring and of interest when concerning neural networks. Examples of Boolean functions that will be of interest are

i) The majority function ($\mathbf{maj}_n$) returning the most common value within its inputs

$$\mathrm{maj}_n(\omega_n) = \mathrm{sign}\left(\sum_{i=1}^{n} (\omega_n)_i\right).$$

ii) The weighted majority function ($\mathbf{maj}_{\theta,n}$) described as

$$\text{maj}_{\theta,n}(\omega_n) = \text{sign}\left(\sum_{i=1}^{n}(\omega_n)_i\theta_i\right).$$

iii) As a special case of the weighted majority function, we have the dictator function ($\mathbf{dict}_{n,k}$), which only returns the value of the k'th bit. Hence

$$\text{dict}_{n,k}(\omega_n) = (\omega_n)_k.$$

iv) The parity function (par_n) defined as

$$\text{par}_n(\omega_n) = \prod_{i=1}^{n}(\omega_n)_i$$

v) the iterated 3-majority (**iter** 3-$\mathbf{maj}_n$) is defined as an iteration of majority functions with three inputs. If $n = 3^k$ for some integer k, the function can be described as a tree graph $G_n = (V, E)$, where

$$\begin{aligned} V &= \{v_{0,0}, v_{1,0}, v_{1,1}, v_{1,2}, \ldots, v_{k-1,0}, \ldots, v_{k-1,3^{k-1}}, v_{k,0}, \ldots, v_{k,3^k}\} \\ E &= \{(v_{k',i}, v_{k'-1,j}) : k' = 1, \ldots, k, i = 3j, 3j+1, 3j+2.\} \end{aligned}$$

Each vertex $v_{k',i}$ is assigned a Boolean value which is iteratively calculated as the majority of the nodes $v_{k'+1,3j}, v_{k'+1,3j+1}$ and $v_{k'+1,3j+2}$. The function inputs give the value of the leaves, and the function output is given by the value of $v_{0,0}$. Similarly, we define the iterated k-majority function where each iteration is done with a majority function with k inputs.

Two important concepts are noise sensitivity and noise stability, first introduced in (Benjamini et al., 1999). To understand these concepts, assume ω_n is sampled uniformly on Ω_n i.e. the bits are i.i.d. fair coin flips. Now for any fixed $\epsilon \in [0, 1]$ also define ω_n^ϵ as for each i, $(\omega_n^\epsilon)_i$ equals $(\omega_n)_i$ with probability $1-\epsilon$ and $-(\omega_n)_i$ with probability ϵ where each bit change happens independently of each other. In (Benjamini et al., 1999), noise sensitivity is defined as follows.

Definition 2.4.1. The sequence $\{f_n\}$ is **noise sensitive** if for every $0 < \epsilon \leq 1/2$,

$$\lim_{n\to\infty} \text{Cov}\left(f_n(\omega), f_n(\omega^\epsilon)\right) = 0.$$

Notice that this concept tells us nothing about a specific Boolean function but rather a sequence of Boolean functions. In summary, as n grows, the sequence $\{f_n\}$ is noise sensitive if it is such that small initial noise in the input leads to an output that is uncorrelated with the original output, removing all information in the initial input. Among the examples of noise sensitive functions are the parity function and the iterated k-majority function noise-sensitive for each odd k (Garban and Steif (2014) exercise 7 p. 23). Often one says that a function f_n is noise sensitive, implicitly talking about the sequence $\{f_n\}$ as n increases.

In contrast to noise sensitivity, (Benjamini et al., 1999) also defines noise stability as follows

Definition 2.4.2. The sequence $\{f_n\}$ is **noise stable** if

$$\lim_{\epsilon \to 0} \sup_n \mathrm{P}(f_n(\omega) \neq f_n(\omega^\epsilon)) = 0.$$

Noise stability is often considered the opposite of noise sensitivity, even if this is strictly not true. It could be that a sequence of Boolean functions $\{f_n\}$ is neither noise stable nor sensitive, and it could also be both. However, these examples are often rather constructed. In words, noise stability can be described as follows: as n grows, small changes in the initial input will likely lead to no changes in the output domain. Among the functions defined above the weighted majority functions and hence also the majority and the dictator function are noise stable (Peres, 2021).

2.4.1 Connection to neural networks

There are several different versions of Boolean neural networks. One example is the Hopfield model (Hopfield, 1982; Juang, 1999), which has shown interesting results lately (Ramsauer et al., 2020). One can also base the Boolean model on structures from an ordinary feed-forward model or convolutional model. There are many similarities between neural networks and familiar Boolean functions (Anthony, 2005; Steinbach and Kohut, 2002). Here follows a short overview of the similarities and differences. The most obvious way to study a neural network from a Boolean perspective is to start from a typical neural network and make it to a Boolean function. The simplest way is to use an activation function in the final layer that forces the output to be Boolean. For example, one can use the sign function $\sigma(x) = \mathrm{sign}(x)$. Comparing the sign function with the otherwise commonly used function hyperbolic tangent function (tanh) $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ the latter can be seen as a real-valued version of the former. We compare the sign and tan hyperbole functions in Figure 2.8.

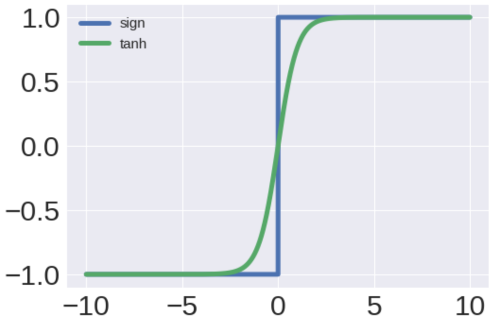


Figure 2.8: An comparison between the sign and tanh activation functions.

Using the sign function as an activation in each neuron, see Figure 2.1, the computation in each unit becomes the weighted majority function where the weights θ corresponds to the model parameters representing the input to that unit. Hence each output in Equation 2.3 corresponds to a weighted majority. The entire model is created by stacking many of these layers as in the regular network structure. Unfortunately, since the sign function is not differentiable, one aspect that is lost by translating it to a Boolean network is the ability to calculate gradients which makes training harder. However, in this thesis, we are only interested in the theoretical properties of these models and no focus is put on how to train them.

It is well-known that real-valued neural networks can be sensitive to input noise and that small perturbations in the input space can lead to significant changes in the output domain. In (Goodfellow et al., 2014) they show a typical example where the classifier model sees an image of a panda, but after some cleverly chosen noise is added to the image, the model is fooled into believing that the image contains a gibbon while a human can not see any difference between the original and the noisy image. Many of these kinds of sensitivity results have been shown in the literature (Szegedy et al., 2013; Moosavi-Dezfooli et al., 2016). This is potentially a huge problem concerning the trustworthiness of the model, and much research has been done on how to train neural nets to make them more robust (Ren et al., 2020; Zheng et al., 2016). While sensitivity holds for typical neural networks, (Peluchetti et al., 2020) shows that a neural network can theoretically be made stable by letting the width of each layer grow to infinity.

This type of noise sensitivity to input perturbations is similar to noise sensitivity for Boolean functions. The similarity is that both consider small noise perturbations in the input domain and study how it affects the output results. It is clear though that they are different since the Boolean version only considers i.i.d. flips with random i.i.d inputs, while the concepts for typical neural networks consider structured data with structured noise components. However, it still raises the interesting question if a Boolean version of a neural network is noise sensitive or noise stable. This question is the inspiration for Paper 3, where we provide theoretical results answering this question for several network settings. There has been some other work done in this aspect. One example is (Peixoto and Drossel, 2009), where they empirically investigate the effect of input noise in Boolean networks.

3 Summary of papers

3.1 Paper 1 (Regularisation via observation weighting for robust classification in the presence of noisy labels)

This paper aims to attack the problem of label noise in the classification setting discussed in section 2.3. The goal is to introduce a robust method that both identifies noise examples and adapts the training such that less focus is placed on suspected errors. All this comes with an easy-to-implement methodology.

Following the categorisation of methods discussed by Song et al. (2020), the approach would be categorised into the robust loss design methodology and follows a similar approach of Gao and Fang (2016); Gao and Feng (2018). The approach is to extend the loss function in the minimisation problem 2.6 and introduce weights $\omega = (\omega_1, \dots, \omega_n)$ that distribute the importance between data points in some sense optimally. As in Gao and Fang (2016); Gao and Feng (2018), these weights are found by extending the minimisation objective to minimise over the weights. This leads to the minimisation objective

$$\hat{\theta}, \hat{\omega} = \underset{\theta, \omega \geq 0}{\arg\min} \left[\sum_{i=1}^{n} \omega_i \ell(\hat{f}(x_i; \theta), y_i) + \alpha g(\omega) \right]$$

where g is a regularisation function with a minimum at $\omega_i = 1$ and α is a hyper parameter determining how much ω can fluctuate from one. In Gao and Fang (2016); Gao and Feng (2018) g is set to $g(\omega) = \sum_i (\omega_i - 1)^2$. The algorithm they use to find an optimal θ and ω is by iterating between the objectives

$$\begin{aligned}\hat{\theta} &= \arg\min_{\theta}\left[\sum_{i=1}^{n}\omega_i\ell(\hat{f}(x_i;\theta),y_i)+\alpha g(\omega)\right]\\ \hat{\omega} &= \arg\min_{\omega\geq 0}\left[\sum_{i=1}^{n}\omega_i\ell(\hat{f}(x_i;\theta),y_i)+\alpha g(\omega)\right].\end{aligned}$$

Our paper extends this approach to solve three main problems. Firstly, iterating between two minimisation objectives can be costly, and it could be that one does not converge to a local minimum since one is solving the problem in a block coordinate descent methodology. Secondly, with no further constraints on the weights ω, it could be that different classes are prioritised differently, creating the risk that all ω_i within one class become very small, making the model disregard that class entirely. Finally, since the optimisation problem is extended to include more parameters, the dimensionality significantly increases the optimisation's complexity.

To address these problems, we add constraints on ω such that the mean weight within each class should be constant, typically 1, and use a cleverly chosen regularisation function g. Here we suggest using the regularisation $g(\omega) = \sum_i \omega_i \log\omega_i - \omega_i + 1$. The reason is that this regularisation function has several benefits. First, it is convex and has a global minimum at $\omega = \mathbf{1}$, which encourages each data point to be treated equally. Secondly, since g is not defined for negative ω, one does not need to consider special measurements for hindering ω to become negative. Finally, and maybe most importantly, it leads to an easy solution for ω, significantly reducing the optimisation complexity.

All in all, for a setting with K classes, where $C_k = \{i : y_i = k\}$ the minimisation problem becomes

$$\hat{\theta},\hat{\omega} = \arg\min_{\theta,\omega}\left[\sum_{i}^{n}\omega_i\ell(\hat{f}(x_i;\theta),y_i)+\alpha\sum_{i}^{n}(\omega_i\log(\omega_i)-\omega_i+1)\right] \tag{3.1}$$

with the constraints $\sum_{i\in C_k}\omega_i = |C_k|$ for all $k = 1,\ldots,K$. These constraints force each class to be treated equally and hinder entire classes from being disregarded while still allowing weights to differ within the class.

It turns out, in the minimisation problem 3.1, one can solve for ω analytically.

With the constraints on ω the minimum is found at

$$\hat{\omega}_i = |C_k| \frac{e^{-\ell(\hat{f}(x_i;\theta),y_i)/\alpha}}{\sum_{j \in C_k} e^{-\ell(\hat{f}(x_j;\theta),y_j)/\alpha}}$$

where $k = y_i$. This further results in that an optimal θ can be found by solving

$$\hat{\theta} = \arg\min_{\theta} -\alpha \sum_{k=1}^{K} |C_k| \log \left(\sum_{i \in C_k} e^{-\ell(\hat{f}(x_i;\theta),y_i)/\alpha} \right). \tag{3.2}$$

Summarising, even though it initially seems that this methodology introduces many extra parameters one needs to solve for, further increasing the risk of overfitting, the number of parameters can actually be reduced to the original amount again but still have the effect of weighted observations. The cost is one additional hyper parameter. We call this methodology regularisation via observation weighting (ROW).

While this sound promising, the effects of observation weights come with a problem in the form of introducing non-convexity to the minimisation objective due to the $\omega_i \ell(\hat{f}(x_i;\theta), y_i)$ term. This means that even if the original objective is convex with a unique minimum, the methodology could make the new objective non-convex and introduce non optimal local minima, making the optimisation harder.

We investigate the minimisation objective 3.2 both analytically and empirically and show that it indeed is a loss function that could reduce the impact of noisy labels and potentially, with the correct α get a result as if there were no noise in labels at all. The theoretical results focus on a linear logistic regression model used on a two-class classification problem. Here we show that with Gaussian covariates and independent label flips in both classes, there is an α that gives estimates of θ as if there were no label noise present in the data.

The experiments are done both on DNNs and on linear models, both on real and synthetic data. In both cases, the experiments show that the methodology indeed is able to identify mislabeled data and disregard them during training. In addition, since some data points are turned off, this leads to less overfitting making it less important to use early stopping. This means that we could potentially train until convergence and that we actually have reached an optimum of the model. This is unlike when using early stopping, where one aims for an optimum but on purpose stops long before it is reached, making interpretability hard. These results go hand in hand with the discussion in section 2.1.4. By turning off protrusive data points, the training will not make the model parameters converge too far away from its initial setting, creating a similar effect as early stopping. Since the model tends to represent simple functions before training, ROW will make the model more likely to be simple after training as well.

3.2 Paper 2 (Entropy weighted regularisation, a general way to debias regularisation penalties)

Inspired by the work in Paper 1, this paper uses the idea of weights to attack the problem of the bias introduced by different regularisation terms. A problem that was discussed in Section 2.2. This is done by adding the weights to the regularisation terms instead of the observation terms in the loss function.

For a linear regression model, a general regularisation objective is formulated as

$$\hat{\theta} = \arg\min_{\theta} \left[\frac{1}{2} \sum_i (y_i - x_i\theta)^2 + \lambda g(\theta;\phi) \right] = \arg\min_{\theta} \left[\frac{1}{2} ||\mathbb{Y} - \mathbb{X}\theta||_2^2 + \lambda g(\theta;\phi) \right]$$

where again ϕ is parameters determining the shape of g and $(\mathbb{X}, \mathbb{Y}) = (x_1, y_1), \dots, (x_n, y_n)$. Now, regularisation such as Lasso or Ridge introduces undesirable bias. Assuming that $g(\theta;\phi)$ can be formulated as $g(\theta;\phi) = \sum_j \hat{g}(\theta_j;\phi)$ we expand the minimisation to be

$$\hat{\theta}, \hat{\omega} = \arg\min_{\theta,\omega} \left[\frac{1}{2} ||\mathbb{Y} - \mathbb{X}\theta||_2^2 + \lambda \sum_j \omega_j \hat{g}(\theta_j;\phi) + \gamma \tilde{g}(\omega) \right]$$

with the goal of reducing the bias introduced by regularisation. Here $\tilde{g}(\omega)$ corresponds to an extra regularisation penalty that hinders ω from becoming negative or too small. The function that is chosen is $\tilde{g}(\omega) = \sum_j (\omega_j \log(\omega_j) - \omega_j + 1)$, giving the total minimisation formulation

$$\hat{\theta}, \hat{\omega} = \quad \arg\min_{\theta,\omega} \left[\frac{1}{2} ||\mathbb{Y} - \mathbb{X}\theta||_2^2 + \sum_j \left(\lambda \omega_j \hat{g}(\theta_j;\phi) + \gamma(\omega_j \log(\omega_j) - \omega_j + 1) \right) \right].$$

It may look like these new weights double the number of parameters we need to optimise, leading to an unnecessary overparameterisation. However, due to the structure of the problem, one can solve for ω analytically as in Paper

1. Doing this, one gets that $\omega_j = e^{-\frac{\lambda}{\gamma}\hat{g}(\theta_j;\phi)}$ which further leads to that the estimate of θ is given by

$$\hat{\theta} = \arg\min_{\theta} \left[\frac{1}{2}||\mathbb{Y} - \mathbb{X}\theta||_2^2 + \gamma \sum_j (1 - e^{\frac{\lambda}{\gamma}\hat{g}(\theta_j;\phi)}) \right].$$

In the article, this is done with two different settings of $\hat{g}(\theta_j;\phi)$, when $\hat{g}(\theta_j;\phi)$ is the Lasso penalty $\hat{g}(\theta_j) = |\theta_j|$ and the ridge penalty $\hat{g}(\theta_j) = (1/2)\theta_j^2$. This gives rise to the new methods Entropy weighted Lasso (EWL) and Entropy weighted Ridge (EWR).

For both methods, it turns out that they inherit many properties from the original regularisation penalty but with a decreased bias. However, one thing that has been lost is the guarantee of convexity. This could lead to non-continuous estimates with respect to changes in data. Fortunately, in both cases, λ and γ can be chosen such that we are guaranteed convexity. For EWL, a sufficient condition is that $\gamma > \frac{\lambda^2}{s_1^2}$ where s_1 is the smallest eigenvalue of $\mathbb{X}$. For EWR, a sufficient condition is that $\lambda < \frac{s_1^2 e^{3/2}}{4}$. This requires that $\mathbb{X}$ has full rank; hence the number of unknown parameters in θ is smaller than the number of data points. In Figure 3.1, one can see the estimates $\hat{\theta}$ in a one dimensional case with one datapoint. This is the same setting as in Figure 2.6 but now also with EWR and EWL. Here there are many aspects to notice. First, locally close to $y = 0$ EWR and EWL both perform very similarly to the Ridge and Lasso methods. For the EWL this makes it so that the method is sign consistent and gives sparse estimates. Secondly, for both methods, the bias decreases for larger θ, which is one of the criteria listed in (Fan and Li, 2001). Notice as well that when $\gamma < \lambda^2$ for the EWL method, the estimate is no longer continuous with respect to y. This comes from the fact that the loss is no longer convex. In table 3.1, one can see a summary of the properties of the regularisation methods that have been discussed. Compared to adaptive Lasso and SCAD, EWL behaves quite similarly. There are, however, a few differences. First of all, adaptive lasso requires two estimations of θ to get the final result, making it potentially take longer to estimate. Both SCAD and EWL just require one estimation step, which can be done using coordinate gradient descent. Experiments show that SCAD is more sensitive to hyperparameter selection which makes adaptive lasso and EWL easier to tune. However, EWL is not guaranteed to be convex and hence requires extra information about the problem formulation in the form of s_1^2. Which method to use boils down to a consideration between these pros and cons that best fit the problem setting.

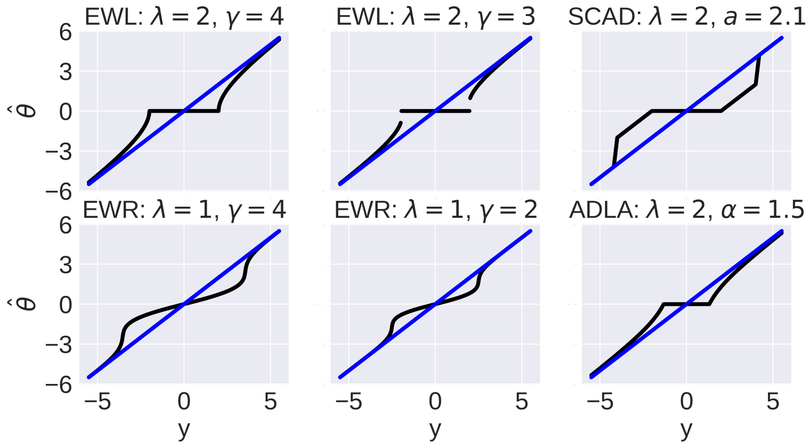


Figure 3.1: How $\hat{\theta}$ depends on y with EWL, EWR, SCAD and Adaptive Lasso for a single data point y and a single parameter θ. In this case $s_1^2 = 1$ and EWL is convex whenever $\gamma > \lambda^2$ and EWR is convex whenever $\lambda < e^{3/2}/4 \approx 1.12$.

	Unbiased	Consistency	Continuity	Sparsity	Sign consistency
OLS	Yes	Yes	Yes	No	No
Lasso	No	Yes	Yes	Yes	Yes
Ridge	No	Yes	Yes	No	No
Bridge	When $\gamma < 1$	Yes	When $\gamma \geq 1$	When $\gamma \leq 1$	When $\gamma \leq 1$
SCAD	Yes	Yes	Yes	Yes	Yes
Adaptive Lasso	Yes	Yes	Yes	Yes	Yes
EWL	Yes	Yes	When $\gamma \geq \lambda^2/s_1^2$	Yes	Yes
EWR	Yes	Yes	When $\lambda \leq s_1^2 e^{\frac{3}{2}}/4$	No	No

Table 3.1: A table of which approaches fulfil the different requested properties. This is an extended version of table 2.1 where EWL and EWR are also included.

3.3 Paper 3 (Noise Sensitivity and Stability of Deep Neural Networks for Binary Classification)

This paper's main objective is to more in depth study the sensitivity/stability properties discussed in Section 2.4 of classical neural networks models. Using the sign function at each activation σ ensures that the resulting model is a Boolean function if we enforce a Boolean input string.

Properties of the neural network do, of course, depend on the weights θ, and talking about noise sensitivity/stability does not make much sense if θ is not specified. In practice, θ is not unambiguous due to randomness in initialisation and randomness in the training algorithm. This raises the question, "Is the distribution π_n such that if I sample θ from π_n, will the corresponding Boolean neural network be noise sensitive or stable?"

To attack this question, we extend the noise sensitivity definition 2.4.1 to a more general version that covers that the noise level ϵ may depend on n. This property is called quantitatively noise sensitivity and is defined as follows.

Definition 3.3.1. Let $\epsilon_n \leq 1/2$ be non-increasing in n and $\epsilon_n \to 0$. The sequence $\{f_n\}$ is **quantitatively noise sensitive (QNS) at level** $\{\epsilon_n\}$ if

$$\lim_{n\to\infty} \mathrm{Cov}\left(f_n(\omega_n), f_n(\omega^{\epsilon_n})\right) = 0.$$

Considering random θ, we extend the definitions to cover randomness in the Boolean function. Given a probability measure π_n defined over all Boolean functions, we define both annealed and quenched noise stability and sensitivity versions of noise stability and sensitivity. These are defined as follows

Definition 3.3.2. π_n is **quenched QNS at level** $\{\epsilon_n\}$ if for every $\delta > 0$ and $\epsilon_n \leq 1/2$, there is an N such that for all $n \geq N$

$$\pi_n\{f_n \;:\; \mathrm{Cov}_{\omega,\omega^\epsilon}(f_n(\omega), f_n(\omega^{\epsilon_n})) \leq \delta\} \geq 1-\delta.$$

Definition 3.3.3. π_n is **annealed QNS at level** $\{\epsilon_n\}$ if for every $0 < \epsilon_n \leq 1/2$

$$\lim_{n\to\infty} \mathrm{Cov}_{f_n,\omega,\omega^{\epsilon_n}}(f_n(\omega), f_n(\omega^{\epsilon_n})) = 0.$$

Definition 3.3.4. π_n is **quenched noise stable** if for every δ there is an $\epsilon > 0$ such that for all n,

$$\pi_n\{f_n : \mathrm{P}_{\omega,\omega^\epsilon}(f_n(\omega) \neq f_n(\omega^\epsilon)) < \delta\} \geq 1-\delta$$

Definition 3.3.5. π_n is **annealed noise stable** if

$$\lim_{\epsilon\to 0} \sup_n \mathrm{P}_{f_n,\omega,\omega^\epsilon}(f_n(\omega) \neq f_n(\omega^\epsilon)) = 0.$$

The annealed versions can be seen as a direct extension of the previous definitions where the probability measures have been extended to include the Boolean functions. On the other hand, the quenched versions study if a typical model is noise stable or sensitive.

As it turns out, there are clear relationships between the annealed and quenched versions, summarised as follows

Theorem 3.3.1. *Let $\mathcal{F}_n$ be the set of all Boolean functions on $\{-1,1\}^n \to \{-1,1\}$*

and let π_n be a probability measure on $\mathcal{F}_n$. Then the following are true

(i) $\{\pi_n\}$ *is annealed QNS at level* $\{a_n\}$ *iff* $\{\pi_n\}$ *is quenched QNS at level* $\{a_n\}$ *and* $\mathrm{Var}_{f_n}(\mathrm{E}_\omega[f_n(\omega)]) \to 0$ *as* $n \to \infty$.

(ii) $\{\pi_n\}$ *is annealed noise stable iff* $\{\pi_n\}$ *is quenched noise stable.*

The paper studies two Boolean versions of neural network architectures, the fully connected neural networks and convolutional neural networks, and examines how different assumptions on θ and architecture lead to different model properties. The Boolean version of the fully connected model is defined recursively for $t \in \{1, \ldots, T_n\}$, as $\omega^{(t)} = \mathrm{sign}(\theta^{(t)}\omega^{(t-1)})$ where $\omega^{(0)}$ is the input bits and $\theta^{(t)} \in \mathcal{R}^{n\times n}$, i.e. a typical fully connected neural network with sign as activation and with no bias terms. The final output is $h(\omega^{(T_n)})$ where h is some fixed Boolean function. To make the first step towards understanding the model's sensitivity, the weights $\theta^{(t)}$ are considered jointly Gaussian independent between t and columns. The weights within one column of $\theta^{(t)}$ are correlated with correlation $\rho_n \geq 0$. One can think of $\rho_n = 0$ as that the model has been initiated, but the model has not seen any data, and no training has been done. Setting $\rho_n > 0$ can (a little far-fetched) be thought of as that some training has been done on data that gives rise to correlation.

One concept that turns out to be important in this setting is the concept of a sharp threshold which is defined as follows

Definition 3.3.6. A sequence of Boolean functions $\{h_n\}$ has a **sharp threshold at** $1/2$ if $\{h_n\}$ is such that for ω being i.i.d. Bernoulli(p_n) and if $\{p_n - 1/2\}$ is bounded away from 0, then $\lim_{n\to\infty} \mathrm{P}(h_n(\omega) = \mathrm{sign}(p_n - 1/2)) = 1$.

The most important results for the fully connected neural network can be summarised as follows.

- Let $\rho_n = 0, 1/2 \geq \epsilon_n \downarrow 0$ be such that $n\epsilon_n \to \infty$ and let $\lim_{n\to\infty} K_n/log(1/\epsilon_n) = \infty$. Then
  (i) if $\lim_{n\to\infty} b_n = 0$, $\lim_{n\to\infty} nb_n = \infty$ and $T_n \in [K_n, e^{nb_n}]$, then for any Boolean functions $\{h_n\}$, the resulting $\{f_{n,T_n}\}$ is annealed QNS at level $\{\epsilon_n\}$ with respect to $\{\pi_n\}$,
  (ii) if the h_n´s are odd and $T_n \geq K_n$, then $\{f_{n,T_n}\}$ is annealed QNS at level $\{\epsilon_n\}$,
  (iii) if $T_n \geq K_n$ then for any $\{h_n\}$, $\{f_{n,T_n}\}$ is quenched QNS at level $\{\epsilon_n\}$,

(iv) there are Boolean functions $\{h_n\}$ such that for T_n growing sufficiently fast with n, $\{f_{n,T_n}\}$ is not annealed noise sensitive.

(v) if $\{h_n\}$ is noise stable and T_n is bounded, then $\{f_{n,T_n}\}$ is annealed noise stable.

- Let $\delta < \rho_n < 1 - \frac{4(\log\log n)^3}{\log n}$ for some fix δ, $1/2 \geq \epsilon_n \downarrow 0$ be such that $n\epsilon_n \to \infty$ and $T_n \geq e^{4(\log\log n)^2}$. Then

  (i) if $\{h_n\}$ is odd and has a sharp threshold $1/2$, then $\{f_{n,T_n}\}$ is annealed QNS at level $\{\epsilon_n\}$.

  (ii) if $\{h_n\}$ is odd and T_n grows sufficiently fast with n, then $\{f_{n,T_n}\}$ is annealed QNS at level $\{\epsilon_n\}$.

- If $\rho_n > 1 - \frac{\log\log n}{18\log n}$. Then the following holds.

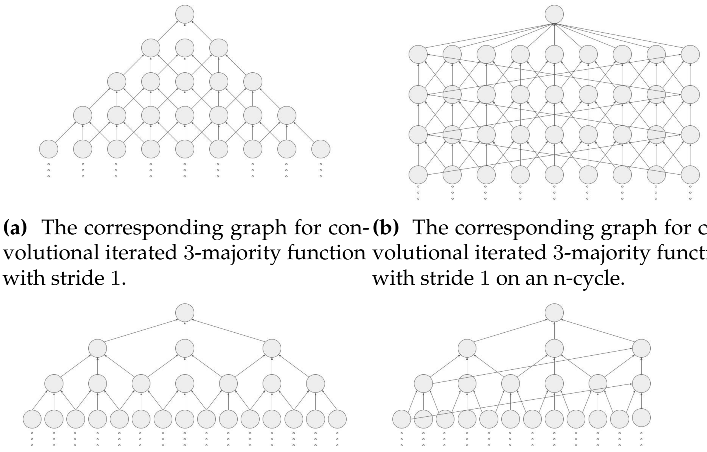
  (i) $\{f_{n,T_n}\}$ is annealed and quenched noise stable if h_n has a sharp threshold at $1/2$.

  (ii) $\{f_{n,T_n}\}$ is annealed and quenched noise stable if $T_n \geq (\log n)^{1/4}$.

Some of these results go hand in hand with the empirical results shown previously. Here typically $\rho_n = 0$. In (Peluchetti et al., 2020), they study the stability of infinitely wide (non-binarized) neural networks and show that taking the width to infinity leads to stable behaviours. Additionally, (Peixoto and Drossel, 2009) show empirically that deep Boolean networks tend to have stable behaviours. Both these results are when the depth is bounded.

The second model studied in our paper is the CNN, specifically a CNN with one-dimensional inputs and very simplified filters, which are mainly considered to be of size three. The final model is given by stacking many of these layers after each other. When analysing the full model, it can help to translate it into a directed graph where each node can be seen as a placeholder of one input bit from one specific layer in the model. In Figure 3.2, one can see the graphs that are considered in the paper, and it can be a helpful reference to have in mind when discussing the properties of the graphs.

Consider first the input string of bits $\omega^{(0)}$. This string is represented by a set of nodes $L^{(0)}$, the nodes at the bottom in the recursion in Figure 3.2. Considering these nodes as an input to a convolutional layer with filter size three, with stride s and no padding, gives rise to a new set of nodes $L^{(1)}$. These nodes are directly above those in $L^{(0)}$ in Figure 3.2. The number of edges between nodes in the two layers corresponds to the filter size, and the amount of overlap between the edges depends on the convolution stride. Each edge in the graph represents one value in the convolutional filter $\theta^{(0)}$, and due to the



(a) The corresponding graph for convolutional iterated 3-majority function with stride 1.

(b) The corresponding graph for convolutional iterated 3-majority function with stride 1 on an n-cycle.

(c) The corresponding graph for convolutional iterated 3-majority function with stride 2.

(d) The corresponding graph for convolutional iterated 3-majority function with stride 2 on an n-cycle.

Figure 3.2

convolutional structure, these values are shared between many edges. This procedure is iterated either until we have a given depth or until only one node is left as output, corresponding to the function's output value. This gives rise to the layers $L^{(t)}$ and filters $\theta^{(t)}$, $t = 0, \ldots, T$ where $\theta^{(t)}$ are independent over t. In the paper, both a regular convolution on an n-cycle is considered.

The results for these models can be summarised as follows

(i) The corresponding CNN with stride one and filter sizes three is annealed, and hence quenched, noise stable for any distribution on $\theta^{(t)}$. This is true for both the noncyclic and the cyclic versions.

(ii) The corresponding CNN with stride two and filter sizes three is annealed, and hence quenched, noise sensitive if the distribution of $\theta^{(t)}$ is such that the probability of representing a majority function is bounded away from zero. This is true both for the cyclic and the noncyclic version.

(iii) The corresponding CNN with stride one and filters with elements all one and of size $2k + 1$ for all $k \in \mathcal{Z}^+$ is annealed and hence quenched, noise stable.

(iv) The corresponding CNN with stride $s > 1$ and with filter sizes, $2k + 1$

for all $k \in \mathcal{Z}^+$ is annealed, and hence quenched, noise sensitive if the distribution of $\theta^{(t)}$ is such that the probability of representing a majority function is bounded away from zero. This is true both for the cyclic and the noncyclic version.

4 Discussion

This thesis contributes to different aspects of the robustness of deep neural networks and linear models, towards label noise in a classification setting, towards better regularisation penalties giving a more favourable bias-variance trade-off and towards knowledge about fundamental robustness properties of deep neural network designs. The work here contributes to a better understanding of the problems at hand and, in some settings, gives methodologies that make the results better. However, there are many aspects to look into to get a complete understanding and avoid potential traps. Here there will be a short discussion about these questions for the different papers.

As we saw in section 2.3, there is an extensive set of methodologies one can use to handle noise labels in a classification setting, and in Paper 1, we present the ROW methodology, a way to reduce the impact of treacherous data examples in an easily implementable way. The drawback is the need for an extra hyperparameter α, which needs to be optimised using a separate subset of the data. This is not uncommon for these types of regularisation technics. However, for the method to be practically sufficient, it is important to have a good sense of what the parameter should be in order to minimise the search space. The paper provides evidence that the sensitivity with respect to α is quite low, but there is still room for more research in this area. Excitingly there is potential for other improvements as well for the ROW methodology. One example is to improve the constraints of the weights $\sum_{j \in C_k} \omega_j = |C_k|$. One suggestion is to add an extra parameter ρ_k for each class and instead use the constraints $\sum_{j \in C_k} \omega_j = \rho_k |C_k|$. This way, ρ_k could be tuned to capture known class imbalances, for example, due to prior knowledge about the label noise distribution.

Training with noisy labels is a large field in machine learning, and many methods have been suggested to improve training. However, one aspect of the problem that is often overlooked is what these methodologies do with the discrimination properties of the model. There is a large family of approaches that rely on identifying data that is out of distribution and puts less trust in those examples. Either via observation weights or modelling of noise. These show promising results regarding identification of untrustworthy training examples. However, if there is a data domain where data points are less frequent and deviant from other data but still labelled correctly, the model could misinterpret these as outliers, putting less enthuses on that domain and disregarding them as training examples. Hence the algorithm has potentially increased discrimination within the model. To the best of our knowledge, this aspect of training with noisy labels is not much investigated and is something that needs further investigation in order to be able to rely fully on an algorithm.

The methodology in Paper 2 is very similar to the one used in Paper 1. Therefore it comes with many of the same perks, such as an easy implementation and no extra parameters to solve for, even if the task at hand initially requires double the amount. However, it requires an extra hyperparameter γ, which one needs to tune. In some cases, one can choose γ to the smallest value such that the problem is guaranteed to be convex. However, this requires that the number of observations is larger than the number of parameters and that we have enough information about the data such that the smallest singular value of $\mathbb{X}$ is known. Even so, one must tune λ to find the perfect model. This is, of course, true for other methods as well, such as Lasso. However, some algorithms calculate the estimates for each λ as fast as one Lasso estimate (Hastie et al. (2009), p 73-77). This algorithm uses the fact that the map $\lambda \to \theta(\lambda)$ is piecewise linear. For EWL, the corresponding map is not piecewise linear, meaning there is no similar methodology for finding the optimal hyperparameters. An interesting question for future research would be if such an algorithm could be found for more complex methods such as EWL or SCAD, potentially leading to a much faster hyper parameter tuning.

The work in Paper 3 can be seen as the first step of a full understanding of the sensitivity and stability properties of the DNN from the perspective of Boolean functions. Even though it is well-known that regular DNNs tend to be sensitive towards input perturbations (Goodfellow et al., 2014; Szegedy et al., 2013; Moosavi-Dezfooli et al., 2016) and there are similar results for Boolean functions (Peixoto and Drossel, 2009), to the best of our knowledge we are first to show theoretical stability/sensitivity results from a Boolean function perspective. It is clear that the results are only a first step towards a full understanding of the properties of Boolean neural networks. First, the results only hold for untrained models whose weights are independent of the data. The next natural step would be to see what would happen if the model were actually trained on data. How do the stability/sensitivity properties depend on the data or the training algorithm? One can imagine many ways to attack these questions. Since it is known that during training, θ does not move that far from its original position (Jacot et al., 2018; Zou et al., 2020; Du et al., 2018), one could ask the question: given an initialisation of the parameters θ, can the model be made stable/sensitive by shifting θ a small distance δ? More concretely, if one allows θ to move some distance δ, how probable is it that the initialisation of θ is such that one can reach a stable function?

Bibliography

Anthony, M. (2005). Connections between neural networks and boolean functions. *Boolean Methods and Models*, 20.

Arpit, D., Jastrzębski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y., et al. (2017). A closer look at memorization in deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 233–242. JMLR. org.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.

Bellec, P. C. and Zhang, C.-H. (2019). De-biasing the lasso with degrees-of-freedom adjustment. *arXiv preprint arXiv:1902.08885*.

Benjamini, I., Kalai, G., and Schramm, O. (1999). Noise sensitivity of boolean functions and applications to percolation. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 90(1):5–43.

Bergersen, L. C., Glad, I. K., and Lyng, H. (2011). Weighted lasso with data integration. *Statistical applications in genetics and molecular biology*, 10(1).

Bootkrajang, J. and Kabán, A. (2012). Label-noise robust logistic regression and its applications. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 143–158. Springer.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. (2018). Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*.

Fan, J. and Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American statistical Association*, 96(456):1348–1360.

Frank, L. E. and Friedman, J. H. (1993). A statistical view of some chemometrics regression tools. *Technometrics*, 35(2):109–135.

Frénay, B. and Verleysen, M. (2014). Classification in the presence of label noise: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25:846–869.

Gao, X. and Fang, Y. (2016). Penalized weighted least squares for outlier detection and robust regression.

Gao, X. and Feng, Y. (2018). Penalized weighted least absolute deviation regression. *Statistics and Its Interface*, 11:79–89.

Garban, C. and Steif, J. E. (2014). *Noise sensitivity of Boolean functions and percolation*, volume 5. Cambridge University Press.

Goldberger, J. and Ben-Reuven, E. (2017). Training deep neural-networks using a noise adaptation layer. In *ICLR*.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Hampel, F. R., Ronchetti, E. M., Rousseeuw, P. J., and Stahel, W. A. (1986). *Robust statistics*. Wiley Series in Probability and Mathematical Statistics: Probability and Mathematical Statistics, New York: John Wiley Sons, Inc., ISBN 0-471-82921-8, MR 0829458.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*, volume 2. Springer.

Hecht-Nielsen, R. (1992). Iii.3 - theory of the backpropagation neural network**based on "nonindent" by robert hecht-nielsen, which appeared in proceedings of the international joint conference on neural networks 1, 593–611, june 1989. © 1989 ieee. In Wechsler, H., editor, *Neural Networks for Perception*, pages 65–93. Academic Press.

Hendrycks, D., Mazeika, M., Wilson, D., and Gimpel, K. (2018). Using trusted data to train deep networks on labels corrupted by severe noise. In *Advances in neural information processing systems*, pages 10456–10465.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.

Huang, J., Horowitz, J. L., and Ma, S. (2008). Asymptotic properties of bridge estimators in sparse high-dimensional regression models. *The Annals of Statistics*, 36(2):587–613.

Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Javanmard, A., Montanari, A., et al. (2018). Debiasing the lasso: Optimal sample size for gaussian designs. *Annals of Statistics*, 46(6A):2593–2622.

Jiang, L., Zhou, Z., Leung, T., Li, L.-J., and Fei-Fei, L. (2018). MentorNet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2304–2313. PMLR.

Juang, J.-C. (1999). Stability analysis of hopfield-type neural networks. *IEEE Transactions on Neural Networks*, 10(6):1366–1374.

Jung, K. M. (2011). Weighted least absolute deviation lasso estimator. *Communications for Statistical Applications and Methods*, 18(6):733–739.

Knight, K. and Fu, W. (2000). Asymptotics for lasso-type estimators. *Annals of statistics*, pages 1356–1378.

Koh, P. W. and Liang, P. (2017). Understanding black-box predictions via influence functions. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1885–1894, International Convention Centre, Sydney, Australia. PMLR.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, M., Soltanolkotabi, M., and Oymak, S. (2020). Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks. In *International conference on artificial intelligence and statistics*, pages 4313–4324. PMLR.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Mingard, C., Valle-Pérez, G., Skalse, J., and Louis, A. A. (2021). Is sgd a bayesian sampler? well, almost. *Journal of Machine Learning Research*, 22.

Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016). Deepfool: A simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

O'Donnell, R. (2014). *Analysis of Boolean Functions*. Cambridge University Press.

Peixoto, T. P. and Drossel, B. (2009). Noise in random boolean networks. *Physical Review E*, 79(3):036108.

Peluchetti, S., Favaro, S., and Fortini, S. (2020). Stable behaviour of infinitely wide deep neural networks. In Chiappa, S. and Calandra, R., editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 1137–1146. PMLR.

Peres, Y. (2021). Noise stability of weighted majority. In *In and Out of Equilibrium 3: Celebrating Vladas Sidoravicius*, pages 677–682. Springer.

Pérez, G. V., Camargo, C. Q., and Louis, A. A. (2018). Deep learning generalizes because the parameter-function map is biased towards simple functions. *stat*, 1050:23.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Adler, T., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., et al. (2020). Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*.

Ren, K., Zheng, T., Qin, Z., and Liu, X. (2020). Adversarial attacks and defenses in deep learning. *Engineering*, 6(3):346–360.

Song, H., Kim, M., Park, D., and Lee, J.-G. (2020). Learning from noisy labels with deep neural networks: A survey. *arXiv preprint arXiv:2007.08199*.

Steinbach, B. and Kohut, R. (2002). Neural networks–a model of boolean functions. In *Boolean Problems, Proceedings of the 5th International Workshop on Boolean Problems*, pages 223–240. Citeseer.

Sukhbaatar, S., Estrach, J. B., Paluri, M., Bourdev, L., and Fergus, R. (2015). Training convolutional networks with noisy labels. In *3rd International Conference on Learning Representations, ICLR 2015*.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Tibshirani, R. (1996). Regression selection and shrinkage via the lasso. *Journal of the Royal Statistical Society Series B*, 58(1):267–288.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2021). Understanding deep learning (still) requires rethinking generalization. *Commun. ACM*, 64(3):107–115.

Zhang, C.-H. (2011). Statistical inference for high-dimensional data. *Mathematisches Forschungsinstitut Oberwolfach: Very High Dimensional Semiparametric Models, Report*, 48:28–31.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. (2017). mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*.

Zhao, P. and Yu, B. (2006). On model selection consistency of lasso. *The Journal of Machine Learning Research*, 7:2541–2563.

Zheng, S., Song, Y., Leung, T., and Goodfellow, I. (2016). Improving the robustness of deep neural networks via stability training. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 4480–4488.

Zou, D., Cao, Y., Zhou, D., and Gu, Q. (2020). Gradient descent optimizes over-parameterized deep relu networks. *Machine Learning*.

Zou, H. (2006). The adaptive lasso and its oracle properties. *Journal of the American statistical association*, 101(476):1418–1429.