

Bachelor Project



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

Training of Binary-Ternary Neural Networks

Viktor Nezveda

**Supervisor: doc. Ing. Tomáš Pevný, Ph.D.
May 2024**

I. Personal and study details

Student's name: **Nezveda Viktor** Personal ID number: **507345**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Cybernetics**
Study program: **Open Informatics**
Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Training of Binary-Ternary Neural Networks

Bachelor's thesis title in Czech:

U ení binárních-ternárních neuronových sítí

Guidelines:

Neural networks with binary activations and ternary weights can be represented as a large set of logical rules, which makes them appealing for explanation of neural networks and understanding their inner workings. Since their training is difficult due to absence of informative gradient, there is not a single established method. The work investigates different strategies to training and compares them.

Instructions:

1. Learn about training of normal neural networks.
2. Read the prior art about training of binary neural networks.
3. Implemented methods based on straight-through gradient, sampling, and regularization.
4. Compare the convergence of implemented methods on few problems.

Bibliography / sources:

- [1] Courbariaux, Matthieu, et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).
- [2] Alizadeh, Milad, et al. "An empirical study of binary neural networks' optimisation." International conference on learning representations. 2018.
- [3] Deng, Lei, et al. "GXNOR-Net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework." Neural Networks 100 (2018): 49-58.

Name and workplace of bachelor's thesis supervisor:

doc. Ing. Tomáš Pevný, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **16.02.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

doc. Ing. Tomáš Pevný, Ph.D.
Supervisor's signature

prof. Dr. Ing. Jan Kybic
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgements

I would like to thank my supervisor, doc. Ing. Tomáš Pevný, Ph.D., for his exceptional guidance and support throughout my bachelor's thesis journey. His expertise, consistency, and patience were crucial in shaping both my research and my personal growth. He will remain an inspiration for my future endeavors, both academically and personally.

Declaration

I declare that the presented work was developed independently and that I have listed all the sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 24, 2024

Viktor Nezveda

Abstract

Neural networks are powerful tools for solving complex problems, but their inner workings often remain incomprehensible to humans. In this thesis, we formulate a novel type of neural network called "binary-ternary" neural network, characterized by ternary weights and binary activation function. This model offers distinct advantages over traditional neural networks, such as the ability to be fully represented as a series of logical rules, which ultimately makes the output of the network more understandable. However, the process of training binary-ternary networks brings forth a new set of challenges such as non-informativeness of gradients and discrete weight space, uncommon for conventional deep nets. In this thesis, we describe and compare the results of three distinct methods that try to circumvent these problems, thereby enabling an effective training.

Keywords: machine learning, neural networks, optimization, ternary weights, binary activation function

Supervisor: doc. Ing. Tomáš Pevný,
Ph.D.
Resslova 307/9
120 00 Praha 2

Abstrakt

Neuronové sítě jsou užitečné nástroje pro efektivní řešení rozličných problémů. Jednou z jejich hlavních nevýhod je však to, že jsou pro člověka často nepřehledné a jejich závěry neodůvodněné. V této práci formulujeme specifický tip neuronových sítí, tzv. binární-ternární sítě, které se vyznačují ternárními vahami a binárními aktivačními funkcemi. Tento model nabízí několik výhod oproti tradičním neuronovým sítím, například schopnost být plně reprezentován jako soubor logických pravidel, což zpřehledňuje jeho výstupy a umožňuje tak lidem hlubší pochopení. Proces trénování binárních-ternárních sítí však přináší problémy netipické pro tradiční hluboké sítě, jako neinformativnost gradientů a diskrétní váhový prostor. V této práci popisujeme a porovnáváme tři rozdílné metody, které se snaží těmto problémům předcházet a umožňují tak efektivní trénování binárních-ternárních sítí.

Klíčová slova: strojové učení, neuronové sítě, optimalizace, ternární váhy, binární aktivační funkce

Překlad názvu: Učení
binárních-ternárních neuronových sítí

Contents

1 Introduction	1
1.1 Definition of Binary-Ternary Neural Network	2
2 Background	3
2.1 The Binary Sign Function	3
2.2 The Hyperbolic Tangent Function	3
2.3 Batch Normalization	5
3 Converting to Logical Expressions	7
3.1 Single Neuron to Logic	7
3.2 Generalization for the whole Network	8
3.3 Data Quantization	9
4 Training and its Challenges	11
5 Deterministic Straight Through Estimator	13
5.1 Quantizing the Weights and the Output	13
5.2 Computing Gradients	14
6 Stochastic Straight Through Estimator	17
6.1 Bernoulli Distribution	17
6.2 Obtaining the Ternary Weights .	17
6.3 Binarizing the Output	18
7 Regularization	21
7.1 The Objective Function	22
7.2 The Regularizer Functions	22
7.3 The Nearest Solution Distance (NSD)	22
7.4 Weight Regularizer	24
7.5 Activation Regularizer	24
7.6 Regularizer Strength Schedule ..	24
7.7 Implementation of a Schedule ..	25
7.8 Conversion to Discrete	26
8 Experimentation Section	29
8.1 Datasets	29
8.2 Training Details	29
8.3 Comparing the Results	30
9 Conclusion	31
Bibliography	33

Figures

<p>2.1 Possible activation functions, $H(x)$ represents the Heaviside step function. 4</p> <p>2.2 Graph of the $\tanh(x)$ and its derivative. Note that for larger (or lower) x the gradient approaches 0, making it non-informative. This phenomenon is called the "vanishing gradient problem". 4</p> <p>2.3 Graph of the $\tanh(\alpha x)$ for different alphas and the sign function. 5</p> <p>2.4 The graph shows the immense impact of the ordering of the batch normalization and the activation functions (denoted as bn and σ respectively). Even if we were to disregard the constraint of binary-ternary model and apply both normalization on top of the signb as the activation, we would still get suboptimal results. The activation function in this case is signb, and a is the preactivation of the neuron, $a = \sum_{i=0}^n w_i x_i + b$. 6</p> <p>4.1 This figure shows the enormous drop in accuracy that makes the trivial approach of training binary-ternary networks inapplicable. 11</p> <p>5.1 Quantizer function with $t_1 = -0.5$ and $t_2 = 0.5$. 13</p> <p>5.2 Quantizer functions and the derivatives that we assigned to them. The quantizer function has $t_1 = -0.9$ and $t_2 = 0.5$. 15</p> <p>6.1 The graph of F and the logistic sigmoid function. 19</p> <p>7.1 Plots of the NSD function for different target sets K. 23</p> <p>7.2 Plots of the NSD function raised to a power p for different target sets K. 23</p>	<p>7.3 Example of an implementation of the schedule during training. Note that we can artificially set the minimum and maximum values for each strength, thus manipulating what the minimization algorithm should focus on. We can also choose whether we will update the λs at all. 25</p> <p>7.4 Results of threshold optimization, where $t_1 = -t_2$. Note that the threshold values were originally set to ± 0.5, thus this maximization had nonnegligible impact of 2 percent increase in accuracy. 27</p>
--	---

Tables

8.1 Results of the proposed methods applied on the three datasets. . . .	30
---	----



Chapter 1

Introduction

Over the past decade, neural networks have demonstrated their effectiveness across a wide range of problems and their usage has been becoming ever more popular, however, they still do suffer from some major disadvantages.

A fundamental problem with traditional networks is their inability to clarify any of the results that they forward. Such a network is essentially a black box since its inner parameter configuration is very hard for humans to grasp. This characteristic makes it virtually impossible for a neural network to be used in areas where humans cannot depend on the vague arguments of machines and an explicit explanation is needed. Fields where human accountability for decisions is high, such as in legal proceedings or medical diagnoses, demand a level of transparency that traditional neural networks struggle to provide.

For that reason, we propose a solution in the form of a binary-ternary network that offers interesting edges over traditional approaches. These networks are constrained to have ternary weights and binary activation functions, which allows them to be fully rewritten as a series of straightforward True or False propositions. Consequently, they can be compressed into a set of logical rules that are much more tangible and comprehensible to human understanding.

Another drawback of traditional networks is that they do require a lot of memory and computational power, since one model can have as many as tens of thousands of neurons. The complexity of such networks makes them quite demanding to be used on resource-limited hardware such as cameras or phones. The binary-ternary networks grant a resolution to this issue as they radically reduce memory usage since they are replacing 32-bit floating point parameters with ternary ones. They also allow a $58\times$ improvement in compute-time by replacing full-precision operations with cheap binary ones [1].

■ 1.1 Definition of Binary-Ternary Neural Network

A "binary-ternary neural network" is a network whose layers have these following properties:

- all weights are ternary;

$$\forall i, j \in \mathbb{N}; i \leq n, j \leq m : w_{ij} \in \{-1; 0; 1\}$$

- the activation function has a binary range;

$$\sigma : \mathbb{R} \rightarrow \{-1; 1\}$$

Note that this definition does not constraint the bias term, therefore it can be any real number.

Chapter 2

Background

2.1 The Binary Sign Function

Since the second condition of the binary-ternary layer requires the activation function to output only binary values, we can define it as:

$$\sigma(x) = \begin{cases} 1 & x \in \mathcal{I} \\ -1 & x \in \mathbb{R} \setminus \mathcal{I} \end{cases} \quad (2.1)$$

where $\mathcal{I} \subseteq \mathbb{R}$. To make the training process more efficient, we will restrict \mathcal{I} to be an interval $[s; \infty)$, where s is a parameter, that quantifies the shift of the activation function.

$$\sigma(x) = \sigma\left(\left(\sum_{i=0}^n w_i x_i + b\right) - s\right) = \sigma\left(\sum_{i=0}^n w_i x_i + (b - s)\right)$$

Note that both the bias term of the neuron b and s fulfill the same role in the output equation, which effectively makes them equivalent. Therefore, to simplify the model, we can set s to zero without loss of generality.

This modification of the interval \mathcal{I} creates a new activation function that greatly resembles the sign function. We will define it as the "binary sign" and denote it as "signb".

$$\text{signb}(x) = \begin{cases} 1 & x \in [0; \infty) \\ -1 & x \in (-\infty; 0) \end{cases} \quad (2.2)$$

Instead of being exclusively $\{-1; 1\}$, the binary range of the activation function could also be $\{0; 1\}$, thus allowing for the usage of the Heaviside step function instead of sign. However, the Heaviside step function does not offer any advantages over the sign, and during implementation seems to be an unpromising choice.

2.2 The Hyperbolic Tangent Function

In this section, we'll highlight the benefits of using the tanh function. In our case, the hyperbolic tangent is a crucial component for all the methods

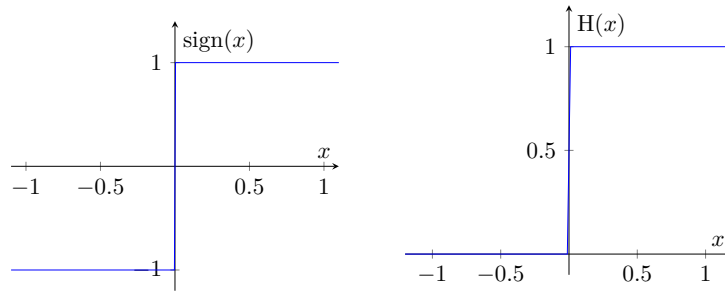


Figure 2.1: Possible activation functions, $H(x)$ represents the Heaviside step function.

discussed in this thesis, as it can be used as a non-linear activation function or a regularizer (more on this in the paragraph focused on regularization). Its desired attributes are differentiability on the whole domain and informativeness of its gradient. The hyperbolic tangent does suffer from the vanishing

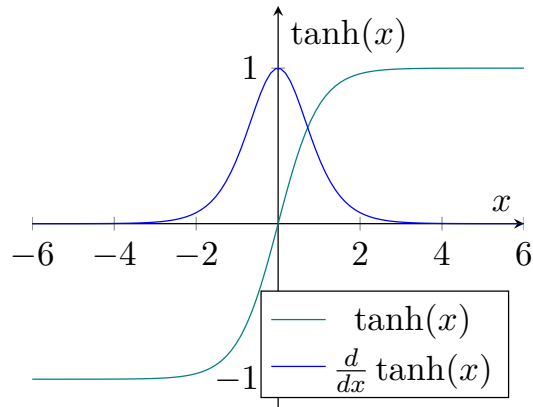


Figure 2.2: Graph of the $\tanh(x)$ and its derivative. Note that for larger (or lower) x the gradient approaches 0, making it non-informative. This phenomenon is called the "vanishing gradient problem".

gradient issue, characterized by the fact that the derivative of the function approaches 0 as $|x|$ increases. However, this characteristic is not crucial for our task, since we are not going to input as large values for x . This is due the fact that we will change the order in which we apply the activation and batch normalization, thus lowering the magnitude of the input values for \tanh (more on this in the section about batch normalization).

Another critical feature of \tanh is the fact, that it is strictly monotonous and bounded, hence there exists a finite limit on both ends.

$$\lim_{x \rightarrow \pm\infty} \tanh(x) = \pm 1$$

Using this property of \tanh , we can derive that it approaches sign when a parameter that multiplies the argument increases. This is expressed in the following lemma:

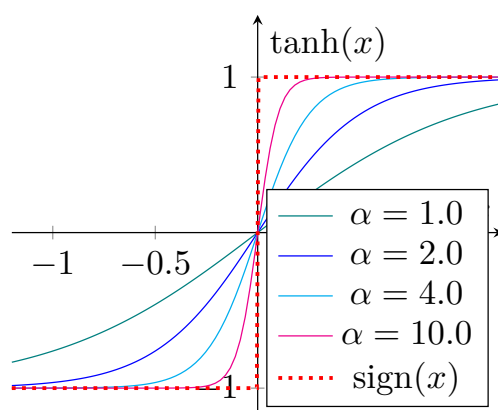


Figure 2.3: Graph of the $\tanh(\alpha x)$ for different alphas and the sign function.

$$\forall x \in \mathbb{R} : \lim_{\alpha \rightarrow \infty} \tanh(\alpha x) = \text{sign}(x)$$

The fact that this identity holds, essentially makes the hyperbolic tangent a relaxed version of the sign function, which positions it to be the perfect candidate when trying to regularize sign.

2.3 Batch Normalization

The batch normalization is a well known technique that greatly improves the performance and stability of neural networks. It achieves that by normalizing the output of a neuron with respect to the expectation and variance obtained from a mini-batch. It then scales the output by applying some trainable parameters.

The conventional approach is to apply the batch normalization function after the neuron's activation function. However, this will not work for any binary-ternary layer, since this would render the output of the neuron continuous rather than binary, thereby violating the given constraint.

Therefore, we will strictly apply batch normalization to the preactivation, rather than after it.

$$f(x) = \sigma \circ \text{bn} \left(\sum_{i=0}^n w_i x_i + b \right)$$

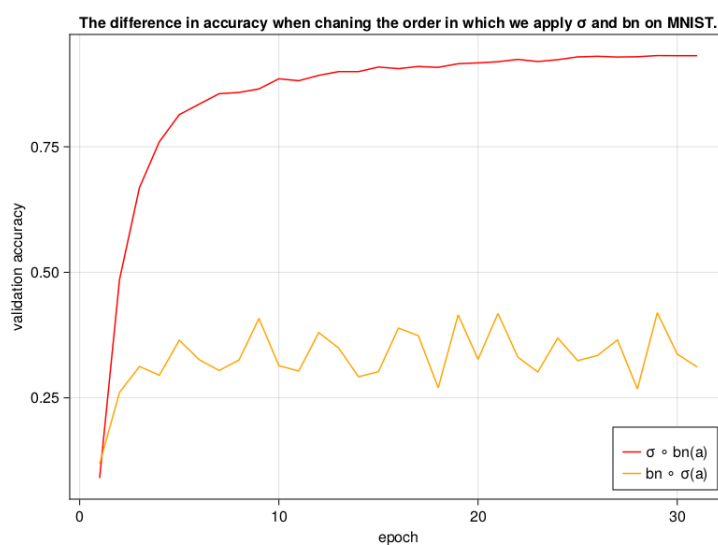


Figure 2.4: The graph shows the immense impact of the ordering of the batch normalization and the activation functions (denoted as bn and σ respectively). Even if we were to disregard the constraint of binary-ternary model and apply batch normalization on top of the signb as the activation, we would still get suboptimal results. The activation function in this case is signb, and a is the preactivation of the neuron, $a = \sum_{i=0}^n w_i x_i + b$.

Chapter 3

Converting to Logical Expressions

In this section, we will demonstrate that the proposed binary-ternary neural network can indeed be fully transformed into an equivalent set of logical expressions. We begin by showing that it is possible for a single neuron, and then we will generalize it for the entire network.

3.1 Single Neuron to Logic

The output function of the binary-ternary neuron is defined as:

$$f(x) = \sigma\left(\sum_{i=0}^n w_i x_i + b\right)$$

where σ is the binary activation function, w is a vector of ternary weights, and b is the bias. For the procedure to work, we also demand the input x to be binary.

In the context of this thesis, we will solely focus on using the sign function as the activation.

$$f(x) = \text{sign}\left(\sum_{i=0}^n w_i x_i + b\right)$$

Since the sign function returns 1 for positive inputs and -1 for negative ones, we can split the output function into two intervals:

$$f(x) = \begin{cases} 1 & \sum_{i=0}^n w_i x_i + b \geq 0 \\ -1 & \sum_{i=0}^n w_i x_i + b < 0 \end{cases} \quad (3.1)$$

This adjustment allows us to determine the output of the neuron by only knowing whether the following inequality holds.

$$\sum_{i=0}^n w_i x_i \geq -b$$

Notice that if a weight w_i is equal to 0, it does not affect the total of the sum nor the truthfulness of the inequality. Therefore, we do not have to consider any zero-valued weights.

Since both the input x_i and any non-zero weight w_i are binary (either 1 or -1), their product must be also binary. Specifically, their product will be 1 if both factors have the same sign and -1 if they do not. We can rewrite the sum by taking advantage of the Iverson bracket function denoted as $\llbracket \cdot \rrbracket$, that outputs 1 if the condition inside is true and 0 otherwise.

$$\sum_{i=0}^n (\llbracket w_i = x_i \rrbracket - \llbracket w_i \neq x_i \rrbracket) \geq -b$$

$$\sum_{i=0}^n \llbracket w_i = x_i \rrbracket - \sum_i \llbracket w_i \neq x_i \rrbracket \geq -b$$

Afterwards, we will denote N as the sum of all non-zero weights and M as the sum of weights that are equal to their respective inputs.

$$N = \sum_{i=0}^n \llbracket w_i \neq 0 \rrbracket$$

$$M = \sum_{i=0}^n \llbracket w_i = x_i \rrbracket$$

Subsequently, by substituting N and M (using the fact that $\sum_i \llbracket w_i \neq x_i \rrbracket = N - M$) to the previous inequality, we obtain the final solution.

$$M - (N - M) \geq -b$$

$$M \geq \frac{N - b}{2}$$

Since the product of the non-zero weight w_i and the input x_i can be interpreted as the logical equivalence ($w_i \iff x_i$), and the output of the neuron solely depends on how many of these expressions are true, we can represent the neuron as a set of these expressions, where it activates (outputs 1) when a certain number of the equivalences hold. Specifically, at least $\frac{N-b}{2}$ of the total N of them must be true. This "m of n" formulation of the problem can be then represented as a disjunction of conjunctions, thus finally transforming the neuron to a logical single term.

For the purposes of this thesis, it is sufficient that a solution indeed exists, but we will not delve in to the problematics of the transformation. This topic is described in more detail in [3].

3.2 Generalization for the whole Network

We have successfully demonstrated that a neuron indeed can be rewritten as a logical expression. The next step is to determine whether this representation generalizes for the entire model. If we assume that the input of the whole network is binary, we can deduce that the first layer can be transformed to logic using the previously described approach. From there, we can cascade

this procedure through subsequent layers, ultimately applying it on the whole network.

This shows that a binary-ternary model, with binary input data can be successfully converted to logical expressions.

■ 3.3 Data Quantization

As we established in the previous section, for us to be able to successfully convert a binary-ternary network to a set of logical expressions, we need all the input data to be binary. We can achieve that by applying an embedding function on the datasets beforehand, thus transforming the data to the desired form.

The quantization function that we chose firstly creates k number of bins and randomly initialized vector $b \in \mathbb{R}^k$. For each value of each feature $x_r \in \mathbb{R}$ of the given data, it then evaluates the following:

$$x_q = \text{signb} \left(\begin{bmatrix} x_r - b_1 \\ x_r - b_2 \\ \vdots \\ x_r - b_k \end{bmatrix} \right)$$

where signb is applied element wise, and $x_q \in \{-1; 1\}^k$ is the desired quantized feature.

Note that this embedding increases the dimension of the dataset by the factor of k , however it does convey less information than the original features.

Chapter 4

Training and its Challenges

The initial approach when attempting to create and train a binary-ternary neural network is to treat it as a traditional (continuous) network without any constraints and train it accordingly. Then, after the training is complete, simply convert it to the desired discrete form by rounding the weights to their nearest ternary values and replacing the activations with the sign function. However, this trivial inapplicable, since there is no inherit connection between the continuous and discrete models, and the conversion between the two is therefore quite drastic, leading to an immense drop in accuracy.

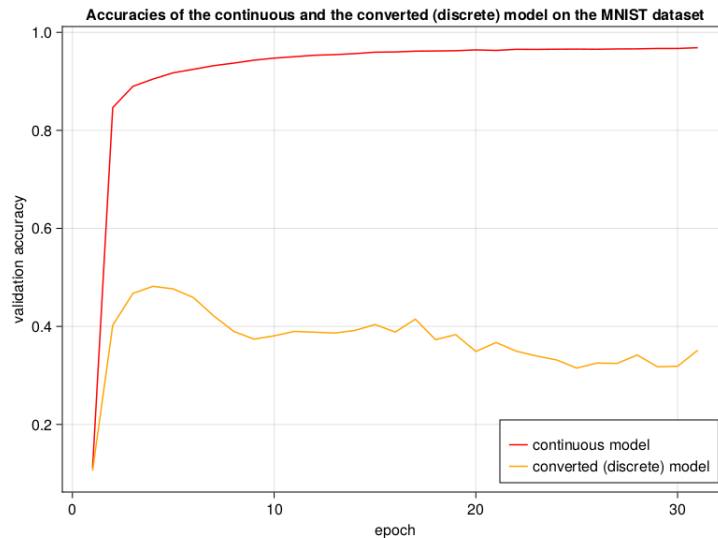


Figure 4.1: This figure shows the enormous drop in accuracy that makes the trivial approach of training binary-ternary networks inapplicable.

Since the trivial approach does not work, we need to develop more sophisticated techniques for training binary-ternary networks. In this thesis, we compare three distinct approaches, each designed to address the challenges of training binary-ternary networks in a unique manner.

The first issue, which all of our methods must combat, arises from the discrete nature of the weight space. Since the parameters of the network are not continuous, the loss function with respect to the weights is non-

differentiable, which makes the use of gradient-based optimization methods impossible.

The subsequent problem we must face is the fact that the activation function must be the sign function, which has certain undesirable properties. Although it is differentiable on almost the whole domain (besides $x = 0$), the gradient that we obtain is consistently 0 and thus non-informative. Therefore, any loss minimization method such as gradient descent will inevitably fail as it does not possess any information about where to descend.

These problems pose a grave obstacle to the training process, and all of our methods must address them both.

Chapter 5

Deterministic Straight Through Estimator

This method consists of two main components, each addressing specific problem of training binary-ternary neural networks.

Firstly, the straight-through approach addresses the discrete weight space issue by introducing a set of full-precision (continuous) parameters, from which are the actual ternary weights then derived. These real parameters work as proxies for the ternary weights during training, thus enabling the use of gradient-based optimization methods.

To deal with the non-informativeness of the gradients, the deterministic straight through estimator (DSTE) method substitutes the derivatives of the problematic functions with artificially fabricated, more informative ones. Although this replacement is not mathematically sound, it does enhance training.

5.1 Quantizing the Weights and the Output

The first step to successfully implementing the DSTE method is to create a function that will convert the continuous weights to ternary ones. We will refer to this function as "quantizer" and denote it as quant :

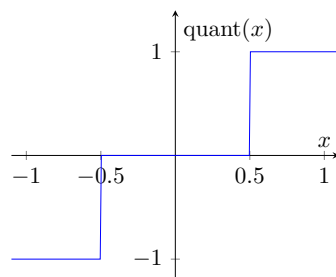


Figure 5.1: Quantizer function with $t_1 = -0.5$ and $t_2 = 0.5$.

$$\text{quant}(x) = \begin{cases} -1 & x < t_1 \\ 0 & t_1 \leq x \leq t_2 \\ 1 & x > t_2 \end{cases} \quad (5.1)$$

where t_1 and t_2 ($t_1 \leq t_2$) are parameters that work as thresholds and can be learned during training or can be set to certain values beforehand. As long as we set the thresholds symmetrically, the exact value should not matter, since the network will adapt to it. We chose the thresholds to be $t_{1,2} = \pm 0.5$.

Notice that if we set both t_1 and t_2 to 0 we obtain the sign function (or shifted $\text{sign}(x - s)$ if we swap 0 for some $s \in \mathbb{R}$). If we were to use the sign function as the quantizer, we would lose the ability to have 0 as a possible weight, thus leading to lower accuracy.

We can then create a set of full-precision parameters denoted as θ , and use the quantizer as a converter between continuous and discrete weights:

$$W = \text{quant}(\theta)$$

where W are the ternary weights used during evaluation, and θ represents the continuous weights with respect to which we minimize the loss function during training. Note that the quant function must be applied element wise, so that the output W is a matrix of the same dimensions as θ .

The definition of binary-ternary networks also requires the output of all neurons to be binary. The DSTE method addresses this constraint by quantizing the output by applying the signb function on top of the activation function. If we were to replace the activation function with the signb entirely, we would remove all non-linearity from the network, thus radically reducing its potential and lowering its accuracy.

By applying these two quantizers we obtain new neuron output function with binary output and ternary weights, thus satisfying the constraints of binary-ternary networks:

$$f(x) = \text{signb} \circ \sigma \left(\sum_{i=0}^n \text{quant}(\theta)_i x_i + b \right)$$

5.2 Computing Gradients

The other problem that this approach must confront is the non-informative gradient of the signb and quantizer functions. To address this, the method opts to change these functions during the backpropagation phase to ones that do possess an informative gradient. In our case, we will substitute these functions with identity, whose derivative is always equal to $\mathbf{1}$.

$$\frac{d}{dx} \text{signb}(x) := \mathbf{1}$$

$$\frac{d}{dx} \text{quant}(x) := \mathbf{1}$$

Using these substitutions and the fact, that the ternary weights are obtained by applying the quantizer, we can derive that the partial derivative of W with respect to θ is equal to $\mathbf{1}$:

$$\frac{\partial W}{\partial \theta} = \mathbf{1}$$

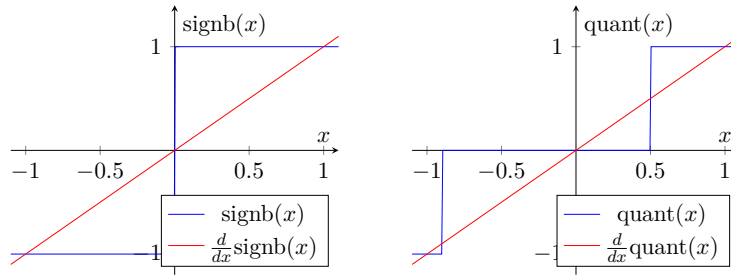


Figure 5.2: Quantizer functions and the derivatives that we assigned to them. The quantizer function has $t_1 = -0.9$ and $t_2 = 0.5$

This can be then used to compute the relationship between the gradient of the loss function with respect to the continuous and the ternary weights:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial W} \mathbf{1}$$

This newly obtained relation enables us to train the network as we would normally do, utilizing the standard optimization techniques.

Chapter 6

Stochastic Straight Through Estimator

This approach combats the training problems posed by the formulation of binary-ternary neural networks in a manner similar to its deterministic counterpart. It likewise tries to render the weight space continuous to allow the use of gradient-based optimization during training. However, the stochastic straight through estimator (SSTE) differentiates itself in the method it uses to derive the ternary weights and the binary output of the neuron. Rather than being exact, it opts to acquire the discrete values by sampling from a probability distribution whose parameters are based on the full-precision weights of the network. This adjustment allows for the usage of standard optimization techniques, since the parameter space with respect to which we minimize the loss function is continuous rather than discrete.

We will use the results from [6] where a similar method was introduced, with the exception that they were using binary weights rather than ternary ones.

6.1 Bernoulli Distribution

In this thesis, we chose to use the Bernoulli distribution as the sampler, since it is the perfect candidate for deriving discrete values from continuous ones. This distribution is characterized by a parameter $p \in [0; 1]$, where the random variable sampled from it, $x \sim \text{Bernoulli}(p)$, takes the value 1 with probability p and 0 with probability $1 - p$.

6.2 Obtaining the Ternary Weights

Since the context of binary-ternary neural networks allows us to use ternary weights, but the possible output of a Bernoulli distribution is only binary (either 1 or 0), we need to modify it to fully utilize the available weight space. We also want the output of the distribution to maintain consistency with the sign of the input, meaning that for a positive input, it will never output -1 as the value for the ternary weight or the other way around.

To successfully sample the ternary weights, we first must compress the full-precision weight space to the interval $[-1; 1]$ by applying the hyperbolic

tangent function to it, obtaining the bounded weights ω .

$$\omega = \tanh(\theta)$$

We will then let the absolute value of ω to function as the probability p in the context of the Bernoulli distribution. However, such a setup would only sample values 0 or 1, entirely neglecting -1 as an option. Therefore, we also have to multiply the result by $\text{sign}(\omega)$ to allow for the output of -1. These adjustments lead us to the following formula:

$$w \sim \text{Bernoulli}(|\omega|)\text{sign}(\omega)$$

where w is the desired ternary weight.

During backpropagation we will set the gradient of the Bernoulli distribution to an artificial value as we did in the DSTE method.

$$\frac{\partial \mathcal{L}}{\partial \omega} := \frac{\partial \mathcal{L}}{\partial w}$$

6.3 Binarizing the Output

Since the Bernoulli distribution is inherently binary, the process of sampling binary values from it is simpler than trying to obtain ternary ones, as we did in the previous section. In this thesis, we solely focus on activation functions whose range is $\{-1;1\}$, thus we need to map the output of the Bernoulli distribution $\{0;1\}$ to the desired ± 1 range. We can achieve that by applying these simple operations:

$$b \in \{-1;1\} \sim 2\text{Bernoulli}(p) - 1$$

where b is some binary output, and $p \in [0;1]$ is the parameter of the distribution.

For us to successfully sample binary output values using the Bernoulli distribution, we first need to derive the probability p from the preactivation a (the output of a neuron before the application of the activation function $a(x) = \sum_{i=0}^n w_i x_i + b$, $a : \mathbb{R}^n \rightarrow \mathbb{R}$). This can be achieved by applying some function F that compresses the unbounded preactivation space to the interval $[0;1]$. In this thesis, we chose F as:

$$F(a) = \frac{\tanh(a) + 1}{2}$$

Note that the F function can be looked upon as the cumulative distribution function (cdf) of the output probability distribution. This is described in detail in [6], where they opt to use the logistic sigmoid function as it is the mathematically sound cdf function when dealing with logistic noise. However, for the purposes of this thesis, it is sufficient to use F .

Combining the previously described mapping and the cdf function F , we can derive the final Bernoulli distribution to sample the desired binary output.

$$o \sim 2\text{Bernoulli}(F(a)) - 1$$

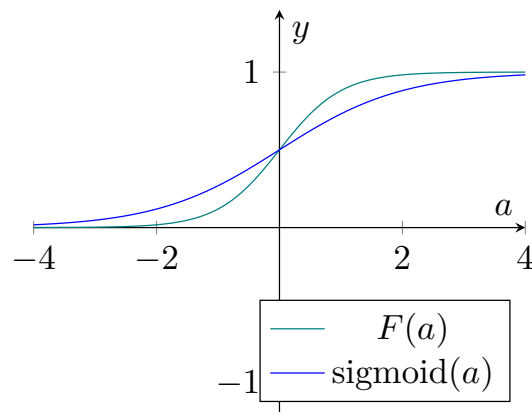



Figure 6.1: The graph of F and the logistic sigmoid function.

where $F(a)$ functions as the parameter p of the distribution.

When trying to take the gradient of this distribution during backpropagation, we will use the results of [6], where they artificially set the derivative of the loss function with respect to o .

$$\frac{\partial \mathcal{L}}{\partial a} := 2 \text{diag}(F'(a)) \frac{\partial \mathcal{L}}{\partial o}$$



Chapter 7

Regularization

The regularization approach circumvents the issues listed above, such as the non-informative gradient and the discrete weight space, by addressing the task in a continuous space rather than in the assigned discrete space. It then penalizes the network for deviating from the discontinuous (ternary and binary) solutions. This adjustment creates a new, relaxed (less constrained) task that is solvable with standard loss minimization techniques.

In our scenario, we'll loosen the two constraints provided: allowing the weight to be real rather than ternary, and substituting the binary activation function in each layer with the hyperbolic tangent. This adjustment makes both aspects continuous, thus allowing for the utilization of gradient descent optimization. In this context, we will refer to the network obtained from the original task as "discrete" and to the one acquired from the continuous surrogate as "smooth," since it resembles continuousness.

As shown in a previous chapter, the conversion between the smooth and the discrete models is quite drastic and leads to an immense drop in accuracy. However, if we train the model by rewarding it for getting closer to a discrete solution, we can significantly reduce the loss in precision post-conversion. In the first section of this chapter, we will cover how to adjust the objective function of the network for us to be able to manipulate the training process. We will then introduce "regularizer" functions that help navigate the network towards a solution that better satisfies the given constraints and thus smoothens the conversion.

Afterwards, we will address how to adjust the learning tendencies using a regularizer strength schedule, that radically modifies the regularizer strengths over time in the hope of finding the best possible local minima.

Finally, we will describe how to set up the conversion process of the smooth model to the discrete one, by optimizing the thresholds of the quantization function, that is used for the transformation. This last part is a crucial component of the regularization method, and has a great potential to increase the final test accuracy.

7.1 The Objective Function

The original problem can be defined as:

$$\begin{aligned} \min_W \quad & \mathcal{L}(f(X, W), Y) \\ \text{s.t.} \quad & \rho_i(W) = 0, i \in \{1, 2, \dots, n\} \end{aligned} \quad (7.1)$$

where \mathcal{L} is the original loss function, n is the number of constraints, and ρ_i is a function that measures how well fulfilled is the corresponding i -th constraint, in the context of this article, we will call these functions "regularizers".

This novel task can be looked upon as a constrained optimization problem and, as such, can be solved by applying the method of Lagrange multipliers. The solution can be then obtained by solving the following problem:

$$\min_W \max_{\lambda} \quad \mathcal{L}(f(X, W), Y) + \sum_{i=1}^n \lambda_i \rho_i(W) \quad (7.2)$$

where λ is the vector of Lagrange multipliers, where each element $\lambda_i \geq 0$ corresponds to the respective constraint. For the purposes of this project, we will refer to these lambdas as the "strengths" of the regularizers.

Since we Now we can set this obtained Lagrangian function as the new "loss" function (also referred to as the objective function) of the network.

$$\mathcal{J} = \mathcal{L}(f(X, W), Y) + \sum_{i=1}^n \lambda_i \rho_i(W)$$

This set-up allows the network not only to optimize the loss but also to move towards satisfying the given constraints, ultimately enhancing accuracy post-conversion.

7.2 The Regularizer Functions

We will define a "regularizer" as a function ρ , which measures how "well satisfied" a given constraint is. The inputs of the regularizer function encompass any parameters of the neural network that we want to control, in our case either f , W , X , and/or Y . We also want this function to have some reasonable properties such as non-negativity for all inputs and for it to be equal to zero when the condition is fully fulfilled. In our scenario, a regularizer will be the mean of the "nearest solution distance" function applied on the weights or on the activation function.

7.3 The Nearest Solution Distance (NSD)

For the regularizers to be able to successfully navigate the weight space, we require a function that quantifies the distance between the input and the

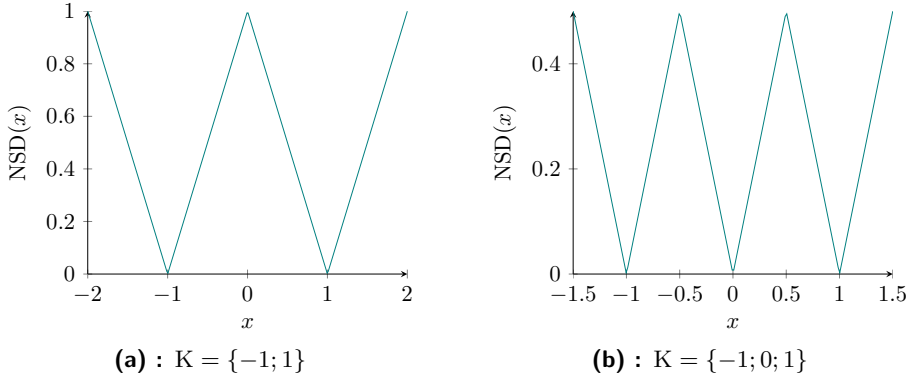


Figure 7.1: Plots of the NSD function for different target sets K .

closest desired solution (in our case, either binary or ternary). We will refer to it as the "nearest solution distance" function, denoted as $\text{NSD} : \mathbb{R} \rightarrow \mathbb{R}$:

$$\text{NSD}(x) = \min_{k \in K} |x - k|$$

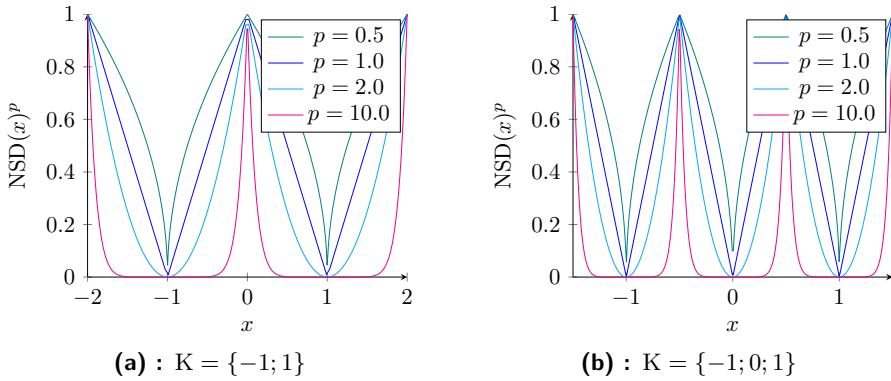


Figure 7.2: Plots of the NSD function raised to a power p for different target sets K .

$$\text{NSD}(x) = \begin{cases} \min_{k \in K} |x - k| & K = \{-1; 1\} \\ 2 \min_{k \in K} |x - k| & K = \{-1; 0; 1\} \end{cases} \quad (7.3)$$

where K is a set of all the solutions, in our case either $\{-1; 1\}$ or $\{-1; 0; 1\}$. Notice that when taking the distance from the ternary solution, the function values are scaled down by 2. Hence, we will normalize NSD when $K = \{-1; 0; 1\}$ like so:

This modification ensures that all the peaks ($x = 0$ or $x = \pm 0.5$) have function value of 1. Without this adjustment, one distance would output larger values, giving it higher priority during training.

7.4 Weight Regularizer

For a neural network to be expressible by logical rules, it needs to have ternary weights. To achieve this goal, we will create a regularizer function that will award the network for getting closer to having ternary weights.

According to [2], the full-precision (continuous) weight space is too vast to find an appropriate ternary solution, hence, they suggest compressing it by applying the tanh function to each weight. This adjustment simplifies the process of finding a ternary solution, as the weights now fall within the range of $[-1, 1]$. We can denote the full-precision parameters of the network as θ :

$$W = \tanh(\theta)$$

Then we can set the regularizer function ρ_1 to the mean over all the weight's nearest distances from the ternary solution $K = \{-1; 0; 1\}$:

$$\rho_1(W) = \rho_1(\tanh(\theta)) = \frac{1}{L} \sum_{i=1}^L \frac{1}{N_i M_i} \sum_{j=1}^{N_i} \sum_{k=1}^{M_i} \text{NSD}(\tanh(\theta))^p$$

where L is the number of layers, N_i is the number of neurons in the i -th layer, M_i is the size of each neuron in the i -th layer, W are the weights of the network, θ are the compressed weights, NSD is the "nearest solution distance" function, and p is a hyperparameter.

7.5 Activation Regularizer

The constraints of this task require the output of each layer to be binary, which effectively means that the activation of each layer must be the sign function. Our regularization method substitutes tanh as the activation for each layer, taking advantage of the properties of hyperbolic tangent showed above. We set the regularizer as the average of the sum of the distances between the output of each layer and the nearest binary solution (NSD).

$$\rho_2(F, X) = \frac{1}{NL} \sum_{x \in X} \text{NSD}(F_1(x))^p + \dots + \text{NSD}(F_L \circ \dots \circ F_2 \circ F_1(x))^p$$

where N is the size of the dataset X , L is the number of layers, and $F = (F_1, F_2, \dots, F_L)$ is the ordered list of all the layer output functions, where F_i corresponds to the output function of i -th layer, NSD is the nearest solution distance from the binary solution ($K = \{-1; 1\}$), and p is a hyperparameter.

7.6 Regularizer Strength Schedule

By redefining the loss function of the neural network, we gain the ability to regulate the regularizer strengths during the training phase. Through this manipulation, we can assign how important each goal is at any given moment

and, thus, on what the minimization algorithm should focus. This process is called "regularizer strength scheduling", and it is a crucial tool that makes training neural networks more efficient.

In this project we used the regularizer strength schedule suggested in [4]. This schedule aims to dynamically adjust the regularizer strength during training using a cosine function. Each cycle begins with a high regularizer strength, which is then gradually reduced to its minimum value, with cycles possibly having different limits and lengths. The purpose of this schedule is

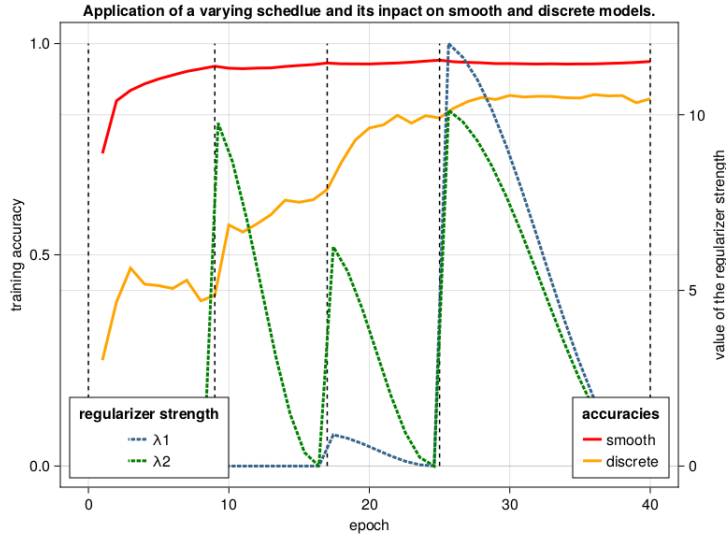


Figure 7.3: Example of an implementation of the schedule during training. Note that we can artificially set the minimum and maximum values for each strength, thus manipulating what the minimization algorithm should focus on. We can also choose whether we will update the λ s at all.

to enable the minimization algorithm to escape getting stuck in local minima, by applying drastic changes to the regularizer strength.

The formula for obtaining the regularizer strength in the T_{cur} epoch of the i -th cycle as described in [4] goes as follows:

$$\lambda_t = \lambda_{min} + \frac{1}{2}(\lambda_{max}^i - \lambda_{min}^i)(1 + \cos(\frac{T_{cur}}{T_i}\pi))$$

where λ_{max}^i and λ_{min}^i are the limits for λ of the i -th cycle, T_i is the length of the i -th cycle. They also suggest to initially start with low T_i and increase it by a factor of T_{mult} at every restart.

7.7 Implementation of a Schedule

Since we are dealing with two different regularizers, we need a strength value for each of them. This makes the use of schedules more difficult since, for example, the constraints might be contrary to some extent, and thus the

minimization algorithm might have a hard time satisfying both of them simultaneously. In this section we will share a few techniques that can help us overcome any potential issues that arise during implementation of the schedule.

We can begin the training process with both regularizer strengths set to zero, thus allowing the network to first learn the task without any constraints. It is also not necessary to update both regularizers at the same time, we can set one to zero and focus solely on the other.

Another technique is to normalize the limits of each λ by setting the minimum to 0, and the maximum to the loss of the network divided by the value of the regularizer.

$$\lambda_{max} = \frac{\mathcal{L}(f(X, W), Y)}{\rho(W)}$$

$$\lambda_{min} = 0$$

This sets the maximum value of the regularizer strength to the loss of the network. We can then multiply the λ_{max} by some factor to prioritize it proportionally to the loss.

7.8 Conversion to Discrete

After the training of the continuous model is finished, it must be converted to its discrete counterpart by applying the quant function on the full-precision weights, and the signb function on the output of each neuron (the signb and quant functions are both introduced in the DSTE chapter).

Since the continuous network is always going to be more precise, this process always lowers the final accuracy. However, the conversion procedure depends on the parameters $t_{1,2}$ of the quant function, and by their optimization we can boost the ultimate accuracy by a few percentage points.

We propose to search for the optimal values $t_{1,2}$ that maximize the accuracy of the discrete model on the training dataset. We can do this by using a grid or random search. Note that at this point, the optimal thresholds might not be symmetrical, however their absolute values should be close to each others. We can simplify this task by searching solely for the symmetrical thresholds, i.e. $t_1 = -t_2$.

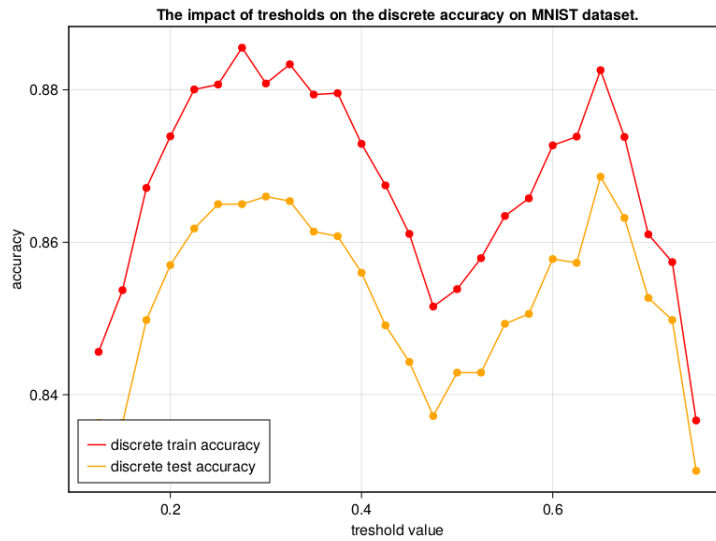


Figure 7.4: Results of threshold optimization, where $t_1 = -t_2$. Note that the threshold values were originally set to ± 0.5 , thus this maximization had nonnegligible impact of 2 percent increase in accuracy.

Chapter 8

Experimentation Section

8.1 Datasets

In this thesis, we opted to choose 3 distinct datasets, MNIST, Wine, and Flower to evaluate the performance of the proposed methods.

The MNIST dataset contains handwritten digits, each 28x28 pixels, labeled from 0 to 9. For the purposes of converting the network to a set of logical expressions, we are not interested in using convolutional networks, thus we can flatten the input image to a vector of 784 elements. For the procedure to work, we need to convert the dataset to a binary format. Since the values of MNIST are in range $[0;1]$, we can round them to the nearest integer.

The Wine dataset consists of 178 samples of Italian wine, each characterized by 13 features and classified into 3 categories. The Flower dataset, introduced in [5], is a toy dataset where the input comprises points arranged in a symmetrical, non-linear shape resembling a flower. The objective is to classify each "leaf" into one of 8 classes.

Since the range of features of both Wine and Flower is not bounded to $[0;1]$, we cannot use simple rounding to make them binary. We need to apply the feature quantizer function introduced in earlier chapter. We used total of 10 bins for each dataset.

Final step of data augmentation is to use the one-hot encoding for the labels, since we are solving a classification problem.

8.2 Training Details

When testing the proposed methods, we used the AdaBelief optimizer with learning rate of 0.001, and cross-entropy as the loss function. We trained the networks for 40 epochs on MNIST and for 400 epochs on Wine and Flower, since they contain fewer samples.

Both the DSTE and SSTE are implemented as described earlier. The regularization model has both powers p of the NSD functions set to 1, and the threshold conversion optimization was applied after the training to enhance the network's performance.

problem	MNIST		Wine		Flower	
	Train	Test	Train	Test	Train	Test
DSTE	.964	.941	.930	.757	.992	.989
SSTE	.850	.854	.597	.636	.853	.853
Regularization	.840	.841	.712	.737	.966	.970
Dense	.994	.975	.869	.895	.954	.960

Table 8.1: Results of the proposed methods applied on the three datasets.

Dense is classical neural network from the library Flux, without any modifications, with the hyperbolic tangent as the activation function.

8.3 Comparing the Results

After evaluating each method on all the datasets, we can conclude that the DSTE method emerges as the top performer among the methods described in this thesis. Remarkably, it surpasses the reference model at the Wine and Flower dataset (this might be due the fact that that we applied feature quantizer on the data beforehand).

We believe that the regularization approach has the greatest potential among all the proposed methods. However, since it has a lot of hyperparameters, optimizing it demands significant time and resources. Perhaps, with different configuration, this method could perform much better. It also seems to be doing significantly better on the Flower dataset where it is comparable to the reference model.

Out of all the methods, our implementation of the SSTE seems to be the least promising. Other stochastic methods may yield better results, for example the PSA method introduced in [6] has potential for improved results.



Chapter 9

Conclusion

In this thesis, we have addressed a fundamental limitation of traditional neural networks: their lack of interpretability. We describe a possible solution to this issue, binary-ternary neural network.

The binary-ternary neural networks characteristics allow the entire network to be represented as a set of logical propositions. This transformation renders the network's decision-making process transparent and understandable for humans, thus making it suitable for applications where accountability and clear explanations are needed.

However, the constraints of binary-ternary neural networks pose grave problems for the utilization of traditional learning algorithms. To overcome these challenges, this thesis describes and compares three unique methods that aim on circumventing the posed issues.

We evaluated these approaches on three diverse datasets, and we have concluded that the deterministic straight through estimator method performs the best on all the datasets, for certain tasks even outperforming an unconstrained model.

We believe that binary-ternary neural networks have great potential in many fields of computer science and beyond. Although their accuracy is lower than that of traditional models, we are certain that their ability to provide clear and interpretable explanations compensates for this limitation.



Bibliography

- [1] Milad Alizadeh, Javier Fernández-Marqués, Nicholas D Lane, and Yarin Gal. An empirical study of binary neural networks' optimisation. In *International conference on learning representations*, 2018.
- [2] Xiang Deng and Zhongfei Zhang. An embarrassingly simple approach to training ternary weight networks. *arXiv preprint arXiv:2011.00580*, 2020.
- [3] Armin Hadžić. Extracting logic rules from neural networks with discrete weights, 2024.
- [4] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.
- [5] Tomas Pevny, Vasek Smidl, Martin Trapp, Ondrej Polacek, and Tomas Oberhuber. Sum-product-transform networks: Exploiting symmetries using invertible transformations, 2020.
- [6] Alexander Shekhovtsov and Viktor Yanush. Reintroducing straight-through estimators as principled methods for stochastic binary networks. In Christian Bauckhage, Juergen Gall, and Alexander Schwing, editors, *Pattern Recognition*, pages 111–126, Cham, 2021. Springer International Publishing.