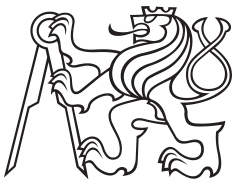
Bachelor Project



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Relaxed Quantization and Binarization for Neural Networks

Martin Mráz

Supervisor: Mgr. Oleksandr Shekhovtsov, Ph.D.
Field of study: Cybernetics and Robotics
May 2022

Acknowledgements

I thank Mgr. Alexander Shekovtsov, Ph.D. for his expert guidance and his great contribution to this thesis.

I also thank my cat Peggy who's cuteness helped me mentally when the days were rough and the horizon was full of clouds.

At last I thank CTU for being a great *Alma mater* of which I am proud of.

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 the university thesis.

In Prague, May 20, 2022

Martin Mráz

Abstract

Quantized neural networks (QNNs) help implement large-scale models on devices with limited hardware resources. This thesis aims to compare and improve methods for training QNNs, so the gap between quantized and full-precision models closes. Discretization is generally a non-differentiable procedure and, therefore, unsuitable for gradient-based backpropagation. The introduction of stochasticity to the network solves this issue for the price of a more complex training process.

Two families of methods were studied to train QNNs. Both introduce stochastic quantization into the standard NNs. One family samples stochastic quantized values in the forward pass. The other family propagates moments of probability distributions.

We propose simplifications to sampling-based methods and suggest that probabilistic propagation can be used for pre-training. Pre-training enables the otherwise slow learning binary NNs to be used on a broader range of deep NN architectures. Experiments validate the functionality of both approaches on the MNIST and CIFAR datasets.

Keywords: quantized neural networks, stochastic neural networks, relaxed quantization, probabilistic learning, binary neural networks

Supervisor: Mgr. Oleksandr Shekhovtsov, Ph.D.

Abstrakt

Kvantované neuronové sítě (QNNs) pomáhají implementovat rozsáhlé modely na zařízeních s omezenými hardwarovými zdroji. Cílem této práce je porovnat a zlepšit metody pro trénování QNNs, aby se zmenšil rozdíl mezi kvantizovanými a 'full-precision' modely. Diskretizace je obecně nediferencovatelný proces, a proto je nevhodná pro optimalizaci zpětným šířením gradientu chyby. Zavedení stochastičnosti do neuronových sítí tento problém řeší za cenu složitějšího tréninkového procesu.

Byly studovány dvě rodiny metod pro trénování QNNs. Obě zavádějí do standardních neuronových sítí stochastickou kvantizaci. Jedna rodina vzorkuje stochastické kvantizované hodnoty během dopředné propagace. Druhá rodina propaguje momenty pravděpodobnostních rozdělení.

Navrhujeme zjednodušení metod založených na vzorkování a navrhujeme, že by pravděpodobnostní propagace mohla být použita pro předtrénování sítí. Předtrénování umožňuje využít jinak pomalu se učící binární neuronové sítě na širší škále architektur hlubokých neuronových sítí. Experimenty ověřují funkčnost obou přístupů na datasetech MNIST a CIFAR.

Klíčová slova: kvantizované neuronové sítě, stochastické neuronové sítě, uvolněná kvantizace, pravděpodobnostní učení, binární neuronové sítě

Překlad názvu: Uvolněná kvantizace a binarizace neuronových sítí

Contents

Figures

Tables

Chapter 1

Introduction

Neural networks are exploited in various cases with full-precision weights and activations. However, larger models demand a lot of computational power and a large memory to store all their parameters/weights. This requirement renders them unusable for resource-constrained devices or limited-budget applications. Therefore some compression is needed to limit the memory and CPU requirements. Among multiple different approaches, several stand out. One such method is *pruning*, which focuses on removing redundant parameters from already functional NNs [AK13]. Another method, and the main focus of this work, is the quantization of weights or activations or both. Last to point out are the methods focusing on custom hardware optimization for different structures of different hidden layers. For example, specialized FPGAs can be used instead of GPUs [CBM+20], which may improve the performance and even lower financial costs. All the methods mentioned above might as well be used together to maximize the savings.

Quantization aims to utilize an already existing neural network and reduce its precision. Therefore scalars can be represented with less than the standard 32 or even 64 bits. While working with smaller precision, much fewer computational operations need to be executed, and thus the CPU power requirements are lowered. Quantization also results in memory savings since all weights are in lower precision. The extreme case of quantization is the Binary Neural Network (BNN) which works with only 1-bit weights and activations. BNNs achieve a 32× (or even 64×) reduction in memory requirements. Rastegari showed in his work [RORF16] that in BNNs, all multiply-accumulate operations (MACs) can be replaced with XNOR operations and bit-counting, which results in up to 58× speedup on CPUs (rather than GPU).

The quantization process can be introduced into a NN before or after training. Quantization after training is called *post quantization*. Popular DL frameworks like **PyTorch** and **TensorFlow** include the post quantization function in their libraries. Such a lowering of precision works well for most DNNs until 8 bits, regardless, without the loss of accuracy [ZZS22]. Zhang demonstrated in his paper [ZZS22] that even at 4-bit precision, a post-training quantized NN for the ImageNet can achieve comparable results to a NN trained with full-precision. For precision lower than 4 bits, the *quantization aware training* needs to be introduced.

The general problem with quantization is that discretization by rounding is discontinuous, and thus its derivative is zero-almost-everywhere. Therefore, the generally used gradient-based optimization is inapplicable. For that reason, multiple approaches were introduced to work around the absence of gradients or gain the gradients from more sophisticated discretization approximation methods. The complexity of these approaches substantially differs, and it fundamentally represents a trade-off between the computational complexity and the accuracy of a network for different levels of precision. The commonly used Straight Through Estimator (STE) is the simplest workaround [YLZ+19]. However, it is mainly used as a heuristic: it is ambiguous with respect to the shape and scale of the surrogate gradients, different implementations involve different clamping rules, etc. Relaxed Quantization by Louizos [LRB+18] allows for more theoretically sound approach. It introduces stochastic relaxation by injecting noises into the quantization. The expected loss is a smooth function of latent weights, and the gradient is well defined. However, it uses discrete categorical random variables and a complicated Gumbel-SoftMax estimator, which is computationally demanding. A fast deterministic function for test-time replaces the exploited quantization process, showing significantly better results. Another popular quantization approach, stochastic rounding by Gupta [GAGN15], can also be explained as quantization with injected uniform noise. It is a simpler method, but it is unclear why it underperforms. We compare relaxed quantization by Loiuzos at al. with the stochastic STE. We find that using uniform noise and the stochastic STE is much simpler and computationally more efficient and achieves the same accuracy.

The second part of this work focuses on probabilistic propagation in Quantized Neural Networks. Instead of sampling the stochastically quantized weights and activations, we compute the mean and variance of the respective discrete variables and propagate these statistics through the network using analytical approximations. Similar approaches were used in the literature for different purposes: in [SF19] for the analytic dropout, in [SCH19] for variational Bayesian learning of QNNs, in [PW18] for training binary neural networks, in [RSFP19] for training networks with quantized weights and binary activations. In these works, quantized weights are represented by general categorical distributions.

We propose developing and applying probabilistic propagation for the noisy quantization model of Louizos et al., which is easier to parameterize. We propose the same treatment for weights and activations. We want to explore the possibility of using this method as pre-training. Its approximate loss function is smooth and non-stochastic, which should enable easier optimization in the beginning. However, we do not expect probabilistic learning to achieve the best performance as a sole method due to the analytic approximations.

Chapter 2

Quantization and Relaxed Quantization

All quantization methods have several common elements, which determine how the method will translate a full precision value to lower bit precision such as 4, 2, or even 1 bit. These elements are *the merit of precision b, a quantization function* $q(\cdot)$ and *an output vocabulary.* The merit of precision is a constant value b, which represents the number of bits available to represent a quantized value. A general discretization function $q(\cdot)$ is defined as a surjective function that returns a value from an output vocabulary of K= 2^b items. The vocabulary is a countable set of values that can be distinguished with b bits. The structure of the output vocabulary is predominantly given as a grid of ordered scalars.

Grid of Quantized Values

The grid of scalars can be generally arbitrary. In this work, an equidistant grid is presumed, for now, with step size 1. While assuming a fixed point quantization, the grid G constructed with b bits is set as:

$$G = [g_0, ..., g_{K-1}] = [0, 1, \ldots, 2^b - 1] \tag{2.1}$$

Since the input values in most cases will not tally with the universally set grid points, additional parameters are needed for scale α and offset β of the

grid. These parameters can be either learnable or set by hand. The resulting grid is:

$$\tilde{G} = \alpha G + \beta \tag{2.2}$$

Note that now the grid is still equidistant, but the step size is now set to α.

Quantization Function $q(\cdot)$

With the set grid $\tilde{G}$, a problem arises regarding how to classify continuous values to the given grid points. As it has been already outlined, simple rounding to the closest grid point

$$\hat{x} \in \tilde{G} = \alpha \cdot \lfloor \frac{1}{2} + \frac{x}{\alpha} \rfloor \tag{2.3}$$

is a non-invertible function. Figure 2.1 shows that in backpropagation, the gradient of such quantization function is zero almost everywhere, and therefore, the SGD optimization is not applicable.

The following sections will introduce multiple heuristics which solve the missing gradient problem by either replacing the gradient in backpropagation or either by introducing a differentiable replacement for the rounding function 2.3.

2.1 Post-Quantization

The post-training quantization method's biggest strength comes from skipping overall issues regarding the backpropagation through the quantization function. At the beginning is a DNN obtained through standard full precision training. Zhang showed in his latest paper [ZZS22] that with a large enough dataset supporting large batch sizes, for example, 1024 samples, he can achieve the near-original results as the state-of-the-art full-precision model network with only 4 bits.

Zhang's work claims that his good accuracy can be algorithmically replicated with guaranteed results. Even though his results are impressive, this work aims to reduce the precision down to the extreme of one or two bits, which the Post-Quantization approach does not deliver. Another issue of Post-Quantization is that it achieves good results almost exclusively in classification tasks [DNL+17].

2.2 Deterministic Quantization with STE

The vanilla version of the Straight Through Estimator (STE) estimates the gradient of the discretization function as the gradient of the identity function $f(x) = x$. The identity function is plotted in the figure 2.1. The outcome of this replacement in its vanilla form is that the backpropagation gradient descent process ignores the presence of the quantization function, so it does not influence the parameter updates in SGD.

The STE method has other modifications, where the gradient approximation imitates some common activation functions such as standard, leaky, or clipped ReLu. It was shown by Shinya [GI22] and Penghang [YLZ+19] that these variant methods converge even in cases when the identity STE fails. The better performance is presumably achieved by retrieving the information about the maximum and minimum values set in the quantization grid $\tilde{G}$.

STE is a relatively simple approach to implement in code. However, the hypothetical information of how are the propagated values influenced by the quantization is lost. Regardless of this rough approximation, especially the clipped ReLu version of STE is a method that shows impressive results even for BNNs [HCS+16]. However, for some low precision cases, the quantization might not be able to reach optimal values as it was described by Penghang [YLZ+19] as the *instability phenomena*. Another issue can arise by the commutation of the quantization errors and can lead to stagnation of the output [Hig20].

Overall, the ReLu STE methods are the simplest and fastest quantization-aware training approaches. They are also deterministic, which makes them easier to implement on hardware [HCS+16]. The results of training a NN with STE are relatively good. Especially for non-extreme cases, STE methods seem like a good trade-off between complexity and accuracy, at least shown on ImageNet, CIFAR10, and MNIST datasets by [YLZ+19], [GI22], [HCS+16].

2.3 Stochastic Quantization

The previous section explained that the only possible way of using deterministic quantization is to exploit some approximation for the gradient of the quantization function. Even though these methods performed well on the

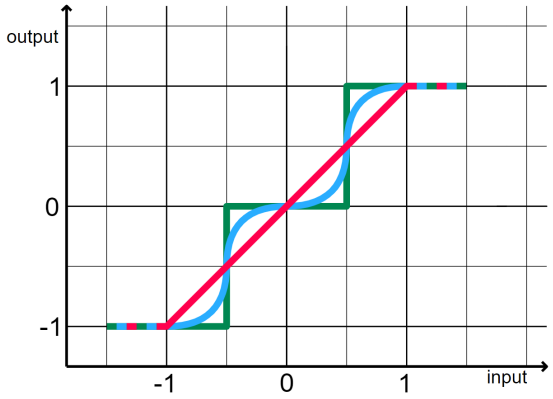


Figure 2.1: The functionality of STE and RQ for the ternary grid. The green line represents the process of deterministic rounding to the closest grid point. The red line represents the approximation of the rounding process using STE within the first and last grid point. The blue line represents the average values of quantization using RQ.

MNIST and CIFAR10 datasets, they are sensitive to their settings and have not proven to be very robust [YLZ+19]. Therefore it is relevant to gain more quantization methods, ideally without the rough gradient approximations.

The new approach started focusing on the injection of stochasticity into the process of quantization. Methods exploiting stochasticity for quantization can be universally called *Stochastic Quantization* (SQ) methods. The idea behind SQ methods have been already explored by several papers ([DNL+17], [LRB+18], [PW18], [GAGN15]) resulting into methods that managed to achieve good results. Y. Dong showed in his paper [DNL+17] that SQ methods are capable of compensating the quantization error and, therefore, can achieve better accuracy than the deterministic methods. Moreover, SQ applies to a wider range of data sets and DNN structures [DNL+17].

The SQ methods can be generally divided into two categories. The first category follows the direction of the deterministic methods and only introduces a quantization function exploited as additional processing of weights and activations. This category does not make major changes in the original network structure, focusing mainly on the sampling of a selected distribution. The other category is based on Probabilistic Neural Networks and the relevant propagation of distributions through a network. This approach requires more significant interventions to the original network's structure as all layers must be rebuilt in this matter. On the other hand, sampling is significantly limited and can even be completely eliminated.

Stochastic Attributes of SQ

A stochastic quantizer can generally be gained by assuming an input noise ϵ. This noise will determine the probability of a specific value of the input signal x to be quantized as each of the grid points from $\tilde{G}$. In other words, artificial noise is added to the input, which will make the previously non-differentiable quantizer **on average** a smooth function. An example of the on average smooth function is depicted in the figure 2.1. This addition of noise creates another variable based on the input x as

$$\tilde{x} = x + \epsilon \tag{2.4}$$

However, only the independent offset value ϵ is obtained for a singular feedforward pass, and the gradient of the 'on average' smooth function cannot be computed. Nevertheless, when assumed that ϵ has zero mean and is continuously distributed, the expected value of deterministically rounded random variable $\tilde{x}$ will generally be closer to the original input value x. Therefore the expected error of the quantization procedure is smaller; thus, SQ methods are capable of describing their input with significantly better accuracy than the deterministic methods while working with the same precision [Hig20]. The following section will describe how the probability of selecting a grid point is computed based on a cumulative distribution function (CDF).

Categorical Distribution Function for SQ

The CDF of the noised input $P(\tilde{x})$ is directly dependent on the CDF of the injected noise ϵ. To evaluate the probability of selecting each grid point from the grid $\tilde{G}$, it is necessary to specify the properties of the distribution. In theory, almost any distribution with zero mean can be chosen for the noise ϵ. The chosen distribution for ϵ is then transferred with its parameters θ onto the input variable:

$$\epsilon \sim P(\mu = 0, \theta) \quad \longrightarrow \quad \tilde{x} \sim P(\mu = x, \theta) \tag{2.5}$$

The probability of selecting a particular grid point is equivalent to the likelihood of $\tilde{x}$ to be inside the interval $\langle -\alpha/2; \alpha/2 \rangle$ around the specific grid point. This estimation can be viewed in the figure 2.3. The probability $p(\tilde{x} = g_i)$ is evaluated based on the original input value x and the parameters of the selected distribution θ:

$$p(\tilde{x} = g_i | x, \theta) = P(\tilde{x} \leq (g_i + \alpha/2)) - P(\tilde{x} < (g_i - \alpha/2)) \tag{2.6}$$

The General SQ Learning Problem

The previous sections have defined how to compute the probability of selecting a grid point as the output of an SQ function. This will serve as the shared basis for SQ methods. Therefore it is suitable to describe the general problem of learning NN with SQ.

As was previously outlined, stochastic quantization works with random variables instead of the point estimates exploited in the deterministic rounding. Let's assume a loss function l. The arguments of the loss function are latent weights θ, the injected noises over the whole network ξ, and the network's input x. The standard aim of any learning process is to minimize the loss function. In the case of NNs, the minimization is obtained via gradient-based optimization. Because of the involvement of the random noises ξ, the gradient for an optimalization step is computed for the expected value over the injected noise ξ:

$$\frac{1}{\partial\theta}\mathbb{E}_{\xi}\left[l(x,\theta,\xi)\right] \tag{2.7}$$

There are multiple approaches to solving the equation 2.7. As outlined in this section's introduction, the methods can be divided into two separate categories. The two following sections will focus on the first category and introduce the well-known Stochastic Rounding (section 2.3.1) and the more general Relaxed Quantization (section 2.3.2). These two methods exploit sampling of the input noise, and the reparameterization trick [KSW15] for learning. On the contrary, the approach of the latter category will avoid sampling and create a Probabilistic Neural Network (PNN), where the general distribution's parameters represent all values.

2.3.1 Stochastic Rounding

One of the pioneer's works on SQ was the Gupta's Stochastic Rounding (SR) [GAGN15]. The SR method considers only the closest higher and lowers grid points with regard to the input value. The elementary approach of SR is to round up or down with an equal probability of one-half. A more promising method determines the probability of selecting a grid point based on the distance between the input value and the closest grid points.

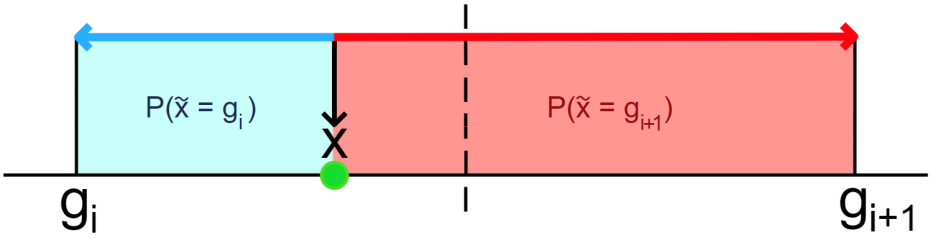


Figure 2.2: Stochastic Rounding principle. The red and blue lines represent the distance between the input x and the closest higher and lower gird point. The ratio between the length of the colored lines corresponds with the ratio of the probability of each grid point to represent the quantized value of x.

The most common case of SR determines the output grid point by sampling from a uniform distribution. In other words, the injected noise from the equation 2.4 is uniformly distributed. Let g_i be the closest lower grid point, and g_{i+1} be the closest higher grid point to the input value x. The start and end point of the selected uniform CDF is set to be g_i and g_{i+1}. The distribution for the formula 2.6 is then:

$$\tilde{x} \sim U(\lfloor x \rfloor_{\tilde{G}}, \lceil x \rceil_{\tilde{G}}) = U(x - \alpha/2, x + \alpha/2) \tag{2.8}$$

Stochastic rounding will round up to g_{i+1} with the probability of $(x - g_i)/(g_{i+1} - g_i)$ and correspondingly round down to g_i with the probability of $(x - g_{i+1})/(g_{i+1} - g_i)$. This process is visualized in the figure 2.2.

After quantization, the input is represented only by the closest higher or lower grid point. This observation is essential for the future distinction from Relaxed Quantization. The average value of the chosen non-extreme grid points (over many quantizations of the same input) will be equal to the value of the original input x. If the value of x is higher/lower than the highest/lowest grid point g_1/g_K, the average value will be cropped in regards to this limitation.

Effects of Stochastic Rounding

Deterministic rounding to the nearest creates rounding errors which are in some cases correlated to each other [Hig20]. This causes a systematic error growth or sum stagnation, which was already observed in 1949 by Huskey and Hartree. The rounding errors generated with stochastic rounding are mean indented. In other words, their average value has no error [Hig20]. Therefore, it can be stated that the expected value of inner (dot) products of large vectors is equal to the exact inner product without quantization. Such

operations are a significant aspect of networks working with convolutional and linear layers, making SR more advantageous than the STE methods.

2.3.2 Relaxed Quantization

This section serves as an introduction to relaxed quantization (RQ). The RQ approach was introduced under this name in a white paper *Relaxed Quantization for Discretized neural Networks* by Louizos [LRB+18] on which are segments of the following text partly based on. In his paper, Louizos elegantly explains the concept of using probabilities and cumulative distribution functions (CDFs) for the stochastic quantization process. Relaxed Quantization (RQ) evades the problem of the non-existing gradient by so-called 'smoothing'. This process is outlined in the figure 2.1. After obtaining the smooth quantizer $q(\cdot)$, the fundamental SGD process can be exploited for optimization.

Let's assume the noised input $\tilde{x}$ from the equation 2.4. For its mathematical attributes, the zero mean logistic distribution with the standard deviation of σ will be considered as the distribution of the injected noise ϵ. This resolves in the context of the formula 2.6 into:

$$\begin{aligned} p(\tilde{x} = g_i|x, \theta) = L(\tilde{x} \leq (g_i + \alpha/2)) - L(\tilde{x} < (g_i - \alpha/2)) = \\ Sigmoid((g_i + \alpha/2 - x)/\sigma) - Sigmoid((g_i - \alpha/2 - x)/\sigma) \end{aligned} \tag{2.9}$$

The CDF of the logistic distribution is the softmax function meaning that no integration is necessary for its evaluation. The sigmoid CDF is fast to evaluate, and it can also be efficiently used for backpropagation. Even though Gaussians are generally used for such approximations, it has been experimentally proven to be less effective to use the normal distribution instead of the logistic distribution [LRB+18].

Now that it has been established how to determine the likelihood of each grid point to be selected as the quantized value of the input, random samples can be drawn. The drawn samples will, on average, create an adhesion curve to the step function visualized in the figure 2.1.

At this point, it is clear that the whole process depends on the scale/deviation of the noise ϵ. When the deviation σ of the selected, in this case logistic, distribution is too large, it prevents the model from fitting the data, which results in the loss of accuracy. When it is too small, the CDF starts

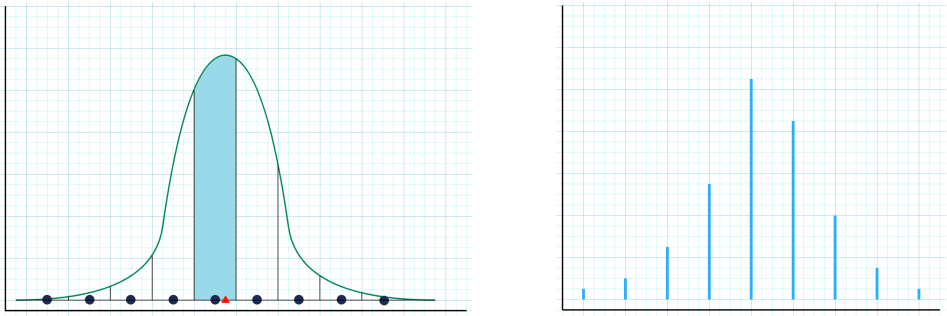


Figure 2.3: On the left is depicted the quantization process with a given non specific distribution $p(\tilde{x})$ (green curve). The real line is separated into K intervals with the width of α, and the center of each interval corresponds to a grid point g_i. The colored area corresponds to the probability of $\tilde{x}$ being assigned to the grid point g_i. **On the right** is a CDF evaluated for the interval around each grid point from $\tilde{G}$. The values correspond to the probability of $\tilde{x}$ being assigned to the appropriate grid point.

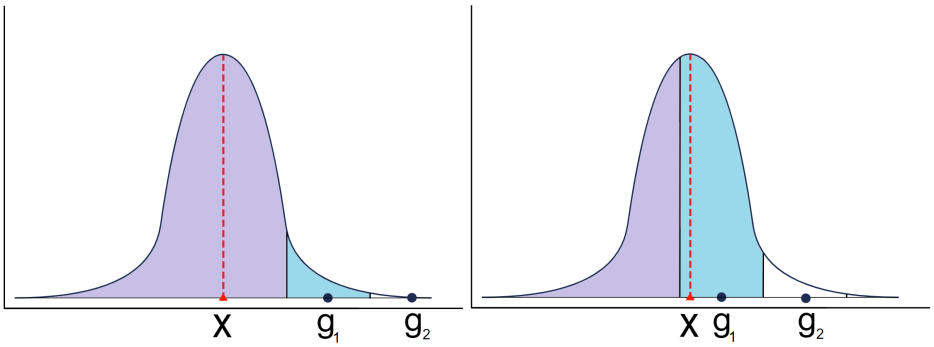


Figure 2.4: Two extreme examples of evaluation of the probability to select the fist grid point g_1. The color purple represents the overflowing probability mass, blue color shows the actual probability within the interval of the first grid point.

to converge to the step function, and the gradient flow will be interrupted [LRB+18]. Therefore the noise deviation needs to be initialized reasonably, not causing underfitting or insufficient gradient flow. The deviation parameter can also be optimized with gradient descent to minimize the loss function.

Categorical Distribution over Extreme Grid Points

The interval around the first and the last grid point generally does not cover all extreme cases of $p(\tilde{x})$. That is especially relevant when x is close to or beyond the border grid points. This can be observed in the figure 2.4. Unlike in SR, automatically assigning the closest grid point when exceeding the grid

range is not applicable since all grid points are possible results of quantization. Therefore it is needed to decide how to deal with the redundant probability mass overflowing the grid range. One option is to remove one of the interval limitations and assign the excess probability to the corresponding grid point. Another option is to remove the relevance from the culprit probability; this option was used by Louizos [LRB+18].

The latter option is de facto a truncation of the distribution so it is restricted to the interval $(g_1 - \alpha/2, g_K + \alpha/2\rangle$. The probability of $\tilde{x}$ being smaller than some constant value C can be then gain as the normalized difference between the probability of the input being smaller than the value C and the probability of the input being smaller than the low end of the interval around the first grid point:

$$P(\tilde{x} \leq C | \tilde{x} \in (g_0 - \alpha/2, g_{K-1} + \alpha/2\rangle, C \in \mathbb{R}) = \frac{P(\tilde{x} \leq C) - P(\tilde{x} < (g_0 - \alpha/2))}{P(\tilde{x} \leq (g_{K-1} + \alpha/2)) - P(\tilde{x} < (g_0 - \alpha/2))} \tag{2.10}$$

The influence of the truncation is negligible when the over/under flown probability has small values. That occurs for well-scaled and offset grids $\tilde{G}$, for non-extreme precision cases, or small variance distributions. On top of that, the re-normalisation of the CDF only introduces more computation to the training which can be saved by exploiting the other method. Moreover, it is unclear if the normalized CDF would perform significantly better solely by decreasing the dominance of the extreme grid points shown in both cases in the figure 2.4. Therefore from now on, the limit removal method will be considered the primary one while keeping the latter option used by Louizos [LRB+18] for possible future consideration for implementation.

Non-Differentiability of the Categorical Distribution

In the previous steps was gained a categorical distribution over the grid $\tilde{G}$, which is still non-differentiable. Once again, the gradient could be approximated. However, this is undesirable towards the ongoing push for directly differentiable quantization. Louizos [LRB+18], proposed replacing the categorical distribution with a concrete distribution, securing a continuous gradient flow. He named this transformation the *relaxation* procedure. It introduces a smooth categorical distribution in a process similar to the renowned softmax function. Softmax is a function that turns a vector of K real values into a vector of K non-negative real values that sum to 1. Let π_i be the categorical probability of sampling the grid point g_i. The shift to the

concrete distribution is obtained as [LRB+18]:

$$u_i \sim \text{Gumbel}(0, 1) \tag{2.11}$$

$$z_i = \frac{\exp((\log \pi_i + u_i)/\lambda)}{\sum_j \exp((\log \pi_j + u_j)/\lambda)} \tag{2.12}$$

The variable z_i represents a random sample from the concrete distribution. The parameter λ is a temperature of approximation, as λ is closer to zero, the result will still be a non-continuous categorical distribution [LRB+18]. The final quantized value $\hat{x}$ of input x can be obtained as [LRB+18]:

$$\hat{x} = \sum_{i=1}^{K} z_i g_i \tag{2.13}$$

Note that the value of $\hat{x}$ **is not** necessarily equal to any grid point from $\tilde{G}$. It is only desirable to deviate from the grid point values because it enables a smooth gradient flow for the SGD optimalization.

RQ Process in Use

Working with $\hat{x}$ allows for usage of the standard gradient-based optimization. However, the process is computationally demanding, and therefore it does not make sense to use the same method of quantization during the testing time when gradients are not computed. Therefore during test time, the quantization can stay stochastic with RQ, or it can be substituted with a basic deterministic one just like in the section 2.2. For this switch between methods to work correctly, it is crucial to transfer the offset and bias of each grid $\tilde{G}$ in the whole DPP.

Even though the explanation of RQ is much more complicated than for the deterministic approach, it can be simplified into four steps.

1. obtain the intervals of each grid point
2. calculate the categorical distribution from the selected CDF
3. calculate the shift to concrete distribution
4. evaluate the output $\hat{x}$ as in equation 2.13

Note that the RQ computational complexity explodes for higher precision. The complexity explosion is caused by the computation of the probability of **each** grid point and the subsequent transformation into the concrete distribution. Another reason is that to compute the equation 2.12, multiple random numbers need to be drawn. To keep the complexity at bay, the *quantization-on-a-local-grid* modification from the following section can be exploited. Another option is to not apply RQ for precision over 4 bits, where other previously introduced fast approaches can be used.

Quantization on a Local Grid

Relaxed Quantization over a grid with a larger amount of grid points is computationally expensive since it requires drawing 2^b random numbers for all weights and activations which are currently being quantized in this layer [LRB+18]. This resolves into a substantially increased number of operations used in forward-pass throughout the network and increased memory demands to keep more parameter values for the backward gradient propagation. Therefore the training time and memory requirements are gradually extended for girds with the precision of over 2 bits.

Louizos [LRB+18] introduced a solution called the 'localized grid'. It is based on the same approach of truncating the extremes of the distribution $p(\tilde{x})$ in the equation 2.6. Let us assume an input x, the closest grid point to it is x_m. Simultaneously, only the grid points within the standard deviation range from x_m are considered. The standard deviation can be derived from the parameter σ, which is used to introduce noise ϵ to the input x.

This results into a new categorical probability established as:

$$P(\tilde{x} \leq C | \tilde{x} \in (\lfloor x \rceil - \delta\sigma, \lfloor x \rceil + \delta\sigma)) = \frac{P(\tilde{x} \leq C) - P(\tilde{x} < \lfloor x \rceil - \delta\sigma)}{P(\tilde{x} \leq \lfloor x \rceil + \delta\sigma) - P(\tilde{x} < \lfloor x \rceil - \delta\sigma)} \tag{2.14}$$

This entry only serves to complete the topic of the RQ. For lower precision quantization, i. e. 2 bits, is such computation unnecessary since the CPU and memory savings are not significant. For larger precision models where the more additional process complexity would actually improve the training, other less demanding methods of discretization can be used.

2.3.3 Comparison between RQ, SR and STE methods

From the theory is clear that the implementation of the logistic RQ algorithm is more computationally complex than the SR and STE methods. It requires probability evaluation over the grid $\tilde{G}$ and also utilizes sampling from the Gumbel distribution for each of its grid points. Moreover, additional computations are needed to acquire the propagating value. The advantage of this stochastic quantization process is its full differentiability which allows for the gradient propagation without any reparametrization tricks. In comparison, stochastic rounding requires only one sampling from the uniform distribution. The only other computation in implementation is the deterministic rounding to the closest grid point. Finally, STE does not require sampling, making it the fastest method to evaluate. However, the complexity of stochastic rounding is not much higher. This difference in evaluation complexity will later notably manifest during experiments.

Rounding to the closest grid point can substitute the sampling-based quantization methods for the testing time. Therefore, the same speed as for the deterministic method is achieved during test time for both RQ and SR. In general better generalization/robustness and the overall accuracy is to be expected for networks using stochastic quantization (RQ, SR) over the deterministic approach (STE). Corresponding results to this expectation were shown in [GAGN15] or in [XAHK21]. Stochastic rounding and RQ both avoid stagnation which was observed for the methods using STE [YLZ+19].

In the work by Louizos et al. was introduced a hybrid function between RQ and SR. This method samples from the logistic distribution and propagates the sampled value via the reparametrization trick. During backpropagation, it is assumed that the value was obtained from the evaluated concrete distribution, and thus, the continuous gradient flow is preserved. The difference between this method and the original SR is the different gradient propagation, the different noise distribution, and the fact that this method can propagate more distant grid points. The probability of propagating a non-neighboring grid point is based on the size of the noise deviation parameter. Sampling over more grid points should, in theory, allow for faster optimization.

It is significant to mention that RQ requires the hardware to be able to support stochastic operations, i. e. draw random numbers with the given parameters [HCS+16]. This requirement can be limited to the training time since for the test time is exploited the deterministic quantization method. Nevertheless, the sampling capability requirement might pose a hindrance to implementing onboard training on small CPUs or chips For such cases, STE is a simple solution, and it can be sufficient for many.

Chapter 3

Probabilistic Learning

Probabilistic learning of neural networks generally includes some kind of stochasticity such as a dropout, various input noise, weights as random variables, etc. The introduced uncertainty is then consciously propagated through the network as a random variable distribution. A great example of this approach is the well-known Bayesian Neural Networks which can generalize better and show confidence in their results thanks to the use of stochastic parameters [SF19]. Multiple approaches have been introduced, i.e. *Sampling-Free Learning of Bayesian Quantized Neural Networks* [SCH19], *Probabilistic Binary Neural Networks* [PW18], *Feed-Forward Propagation in PNNs* [SF19].

Where a standard neural network (SNN) propagates only point estimates and loses the concept of uncertainty, a probabilistic neural network (PNN) keeps its knowledge of uncertainty and propagates it to the following layers [SCH19]. A significant advantage of PNNs also arises in the field of quantization, where optimization of the distribution parameters does not face the trouble of zero gradients in the rounding function [SF19].

A PNN is based on working with probability distributions (PDs). Every value in PNNs is perceived as a PD, which can be to a great extent characterized only by its mean and variance. In cases when no sampling is performed, no other information about a PD is needed, which keeps the propagation process as general as possible. For cases when detailed identification of a PD is needed, the standard normal distribution was chosen in this work since it is widely exploited and has an apparent use of the parameters, which is advantageous. The assumption of the normal distribution is also based on the

renowned Central Limit theorem, which takes place during the multiplication of random variables within linear and convolutional layers [SLF17].

PNNs are the second category of approaches to the SQ learning problem stated in the section 2.3. Unlike the sampling methods, it is entirely stochastic, so it generalizes better and learns faster [SF19]. Thus the PNN learning for quantization poses a good method for fast model initialization and provides the model with better robustness. However, during the derivation of some layers, some dependencies between random variables occur, and approximations must be made. Together with the general tendency of stochastic learning to underfit the model for the price of better robustness, the PNN cannot converge to the best accuracy results. Therefore, when the stochasticity enhanced by the influence of the exploited approximation starts to slow down the learning, the PNN method can be replaced by a more precise sampling method.

This chapter introduces a process of taking a state-of-the-art architecture designed for SNNs and rebuilding it into a PNN. Ideally, this process should be universal to apply to the same stochastic relaxation model as the SR and RQ methods. Then the targeted transfer between these two approaches can be smoothly achieved during the training phase.

Inputs & Outputs of a Layer in PNNs

Generally an input to any layer in NNs is a vector $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$. In a PNN is every element of $\mathbf{x}$ represented by its mean $\mu_i = \mathbb{E}[x_i] = \phi(\mathbf{h})$ and variance $\sigma_i^2 = \mathbb{V}[x_i] = \theta(\mathbf{h})$. The vector $\mathbf{h}$ represents the input to the NN. Functions ϕ and θ are given by the previous layers, such as a Linear layer or a Quantization Layer, which influence the propagating parameters.

Since point estimates are no longer used, it is necessary to derive how PDs and their means and variances will change when various functions are used. Thus, in the following section, the propagation of distribution parameters of all elementary layers in a NN will be derived. Notably, a derivation is needed for the already introduced relaxed quantization layer 2.3.2 which will keep the memory and computational demands low during the training time.

3.1 Mean and Variance Propagation

This section focuses on working with means and variances of stochastic weights and activations. Unlike the standard approach, gradient-based learning is not updating the trained value itself, but it changes the properties of each value's probability distribution. Backpropagation of means and variances requires an analysis of calculations used in elementary layers and derivations of how the layer affects the propagating PD properties.

In the ongoing sections, the below-listed formulas for probabilistic mean $\mathbb{E}$ and variance $\mathbb{V}$ will be repeatedly exploited. Since quantization focuses strictly on the discrete representation of values, only the formulas in the 'sum form' are necessary. When $P(x = a)$ represents the probability of the event 'a' to occur, then:

$$\mathbb{E}[X] = \sum_{a \in \tilde{G}} a \cdot P(X = a) \tag{3.1}$$

$$\mathbb{V}[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}\left[X^2\right] - (\mathbb{E}[X])^2 \tag{3.2}$$

In the equation 3.2 it is assumed that all terms are finite. From now on, notations $\mu_X = \mathbb{E}[X]$ and $\sigma_X^2 = \mathbb{V}[X]$ will be used. Similarly when random vectors are computed with, notations $\mu_i = \mathbb{E}[X_i]$ and $\sigma_i^2 = \mathbb{V}[X_i]$ are used.

Further calculations will be based on the properties of the mean and variance. The properties show the consequence of scaling and offsetting the random variable X. The rest of the properties show the reaction of variance to the addition and multiplication of two random variables.

$$\mathbb{E}[c \cdot X + b] = c \cdot \mathbb{E}[X] + b = c \cdot \mu_X + b \tag{3.3}$$

$$\mathbb{V}[c \cdot X + b] = c^2 \cdot \mathbb{V}[X] = c^2 \cdot \sigma_X^2 \tag{3.4}$$

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y] = \mu_X + \mu_Y \tag{3.5}$$

$$\mathbb{V}[X + Y] = \mathbb{V}[X] + \mathbb{V}[Y] + 2 \cdot \mathrm{Cov}(X, Y) = \sigma_X^2 + \sigma_Y^2 + 2 \cdot \mathrm{Cov}(X, Y) \tag{3.6}$$

$$\mathrm{Cov}(X + Y, Z) = \mathrm{Cov}(X, Z) + \mathrm{Cov}(Y, Z) \tag{3.7}$$

$$\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y] = \mu_X \cdot \mu_Y \tag{3.8}$$

$$\mathbb{V}[X \cdot Y] = \mathbb{E}\left[X^2 Y^2\right] - (\mathbb{E}[XY])^2 = (\sigma_X^2 + \mu_X^2)(\sigma_Y^2 + \mu_Y^2) - \mu_X^2 \mu_Y^2 = \\ = \sigma_X^2 \cdot \sigma_Y^2 + \sigma_X^2 \cdot \mu_Y^2 + \mu_X^2 \cdot \sigma_Y^2 \tag{3.9}$$

The last two properties **assume independence** between the random variables X and Y.

3.1.1 Fully Connected (Linear) and Affine Layer

The general calculation for the Fully Connected and Affine layers is a sum of products of n weights and n input values which is then offset by an external bias b:

$$a_j = \sum_{i=0}^{n-1} x_i w_i + b \tag{3.10}$$

For the linear layer the weights and are considered random due to the stochastic relaxation of the quantized weights, for the affine layer the weights and the bias are considered deterministic scalars.

Fully Connected

The mean for the linear layer $\mathbb{E}[a_{lin}]$ is derived gradually. First the mean of the multiplication of two random variables (Eq. 3.8) x_i and w_i is computed and later the sum (Eq. 3.5) over all the semi-activations is performed. At last is added the mean of the bias variable:

$$\mathbb{E}[x_i \cdot w_i] = \mathbb{E}[x_i] \cdot \mathbb{E}[w_i] = \mu_{x_i} \cdot \mu_{w_i} \tag{3.11}$$

$$\mathbb{E}\left[\sum_{i=0}^{n-1} x_i \cdot w_i\right] = \sum_{i=0}^{n-1} \mathbb{E}[x_i \cdot w_i] = \sum_{i=0}^{n-1} \mu_{x_i} \cdot \mu_{w_i} \tag{3.12}$$

$$\mu_{lin} = \sum_{i=0}^{n-1} \mu_{x_i} \cdot \mu_{w_i} + \mathbb{E}[b] = \sum_{i=0}^{n-1} \mu_{x_i} \cdot \mu_{w_i} + \mu_b \tag{3.13}$$

The variance for the linear layer is computed in a similar matter using the equations 3.9 and 3.6:

$$\mathbb{V}[x_i \cdot w_i] = \sigma_{w_i}^2 \cdot \sigma_{x_i}^2 + \sigma_{w_i}^2 \cdot \mu_{x_i}^2 + \mu_{w_i}^2 \cdot \sigma_{x_i}^2 \tag{3.14}$$

$$\mathbb{V}\left[\sum_{i=0}^{n-1} x_i \cdot w_i\right] = \sum_{i=0}^{n-1} \mathbb{V}(\sigma_{w_i}^2 \cdot \sigma_{x_i}^2) + \text{covariance term} \approx \sum_{i=0}^{n-1} \mathbb{V}(\sigma_{w_i}^2 \cdot \sigma_{x_i}^2) \tag{3.15}$$

$$\sigma_{lin}^2 = \mathbb{V}\left[\sum_{i=0}^{n-1} x_i w_i\right] + \mathbb{V}[b] = \sigma_b^2 + \sum_{i=0}^{n-1} (\sigma_{w_i}^2 \cdot \sigma_{x_i}^2 + \sigma_{w_i}^2 \cdot \mu_{x_i}^2 + \mu_{w_i}^2 \cdot \sigma_{x_i}^2) \tag{3.16}$$

In the equations 3.14 and 3.15 an approximation was made based on the independence between the random values of the 'part-preactivations' $x_i \cdot w_i$.

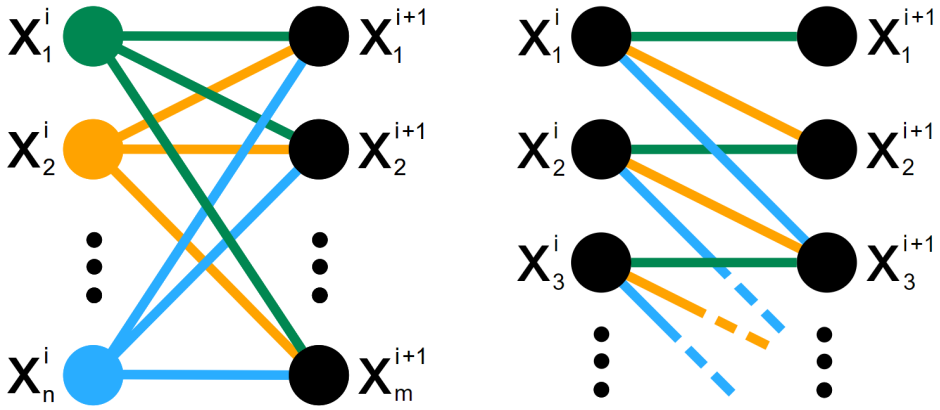


Figure 3.1: **On the left** is a diagram of influence of the parent units on the output activations in a linear layer. **On the right** is a diagram of the influence of the shared weights on the output values of a convolutional layer. Convolutional and Linear layers make their outputs dependent, which corrodes the otherwise precisely computed propagation of the statistical moments, which assumes complete independence across the network.

The 'covariance term' is a sum of covariance evaluations (based on the formula 3.7) between the gradually added random values a_i, which are assumed to be independent. Therefore the covariance term is equal to zero. Even though some influence is be shared between the individual NN units within a linear layer, it is still a fair approximation [RSFP19]. The mentioned dependence is based on the fact that all variables a_i inherit properties from the same parent units, as is shown in the figure 3.1. Since the bias b is completely independent of the previous layers, the covariance between b and the pre-activation a_i is zero without any approximation.

Affine

For the Affine Layer, the results from the Fully Connected layer can be used after substituting the variance of the weights and bias with zero and after substituting the appropriate means by their true value.

$$\mu_{aff} = \mathbb{E}\left[\sum_{i=0}^{n-1} x_i \cdot w_i\right] + b = \sum_{i=0}^{n-1} \mu_{x_i} \cdot w_i + b \tag{3.17}$$

$$\sigma^2_{aff} = \sum_{i=0}^{n-1} w_i^2 \cdot \sigma^2_{x_i} \tag{3.18}$$

The same formulas can be derived from equations 3.3 and 3.4.

3.1.2 Batch Normalization Layer

The general reason to use the Batch Normalization layer (BN) is to speed up the training and make it more stable by normalizing activation vectors in DNNs. The normalization is applied to a batch of input data. The input of a BN layer consists of m activations. The activations are then normalized, scaled, and offset. This process is expressed in the following equations where i represents the input index within a batch:

$$\mu_{BN} = \frac{1}{m}\sum_{i=1}^{m} a_i \tag{3.19}$$

$$\sigma_{BN}^2 = \frac{1}{m}\sum_{i=1}^{m}(a_i - \mu_{BN})^2 \tag{3.20}$$

$$a_{i_{norm}} = \frac{a_i - \mu_{BN}}{\sqrt{\sigma_{BN}^2 + \varsigma}} \qquad \text{normalization} \tag{3.21}$$

$$\tilde{a} = \alpha \cdot a_{i_{norm}} + \beta \qquad \text{learnable scale and offset} \tag{3.22}$$

The parameter ς is present to assure numerical stability for extreme cases. The parameters for scaling α and offset β are deterministic scalars, which can be optimized during training.

To propagate through BN, it is first needed to evaluate the mean values for the batch normalization's mean and variance from the equations 3.19 and 3.20 since the value a_i is a random variable.

$$\mathbb{E}\left[\mu_{BN}\right] = \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} a_i\right] = \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}\left[a_i\right] = \frac{1}{m}\sum_{i=1}^{m}\mu_i \tag{3.23}$$

Before calculating the expected value of the BN variance, it is important to realize that the BN mean is also a random variable. Therefore the result from the equation 3.23 is used in the following computation:

$$\mathbb{E}\left[\sigma_{BN}^2\right] = \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m}(a_i - \mathbb{E}\left[\mu_{BN}\right])^2\right] = \frac{1}{m}\sum_{i=1}^{m}\sigma_i^2 + \frac{1}{m}\sum_{i=1}^{m}\left(\mu_i - \mathbb{E}\left[\mu_{BN}\right]\right)^2 \tag{3.24}$$

The concrete derivation of 3.24 is in more detail in the Appendix A.1.

Now all terms in the equation 3.22 are represented as deterministic scalars but a_i. However, its mean is known as one of the inputs to the layer -

$\mu_i = \mathbb{E}[a_i]$. Therefore the propagating mean of the BN layer is:

$$\mathbb{E}[\bar{a}] = \mu_{out} = \frac{\mu_i - \mathbb{E}[\mu_{BN}]}{\sqrt{\mathbb{E}[\sigma^2_{BN}] + \varsigma}} \cdot \alpha + \beta \tag{3.25}$$

Lastly the variance can be computed. It can be computed from the equation's 3.2 not simplified version. The detailed derivation is in the appendix A.2, below is the resulting formula:

$$\sigma^2_{out} = \alpha^2 \cdot \frac{\sigma^2_i}{\mathbb{E}[\sigma^2_{BN}] + \varsigma} \tag{3.26}$$

As for now, this derivation of Batch Normalisation is only for one channel propagating the values according to the equations 3.25 and 3.26. For standard usage, at least the second dimension is needed. This can be done in a similar way as for standard BN, where all the dimensions within one layer have their own BN processes with separate means, variances, and BN parameters $\alpha^{(k)}$, $\beta^{(k)}$.

3.1.3 Quantization Layer

This section will take into account two approaches. One method will derive the propagation of mean and variance for rounding to the closest grid point without any external noise. This method simulates the deterministic quantization of stochastic inputs. The other section will derive the propagation for a quantization layer with injected external noise ϵ in a form based on the RQ layer 2.3.2. This can be viewed as a deterministic and stochastic quantization procedure in PNNs.

Propagation of Statistical Moments without Noise Injection

Let's assume an input vector $\mathbf{x} = \{x_0, x_1, \ldots, x_{n-1}\}$ of the quantization layer without regards to the Probabilistic Learning (PL). The general formula for the quantization process the output of this layer is

$$\tilde{x_j} = \lfloor x_j \rceil_{\tilde{G}} \tag{3.27}$$

where $\tilde{x_j}$ is the quantized value of x_i on the discrete grid $\tilde{G}$ containing K grid points. In PNNs is x_j a random variable, and it is represented by its

statistical moments. Until now, no other assumptions about the propagating distribution have been made. However, for quantization to be exploited, more information about the density function of the PD is required in order to obtain the probability of the discrete grid points. Therefore an assumption is made about the form of the density function belonging to the propagating PD. The most logical assumption is that the CLT will occur somewhere in the network to some extent.

Shayer [SLF17] showed in his paper, that when a linear or convolutional layer has stochastic independent weights $w_{m,n}$, then the pre-activations $z_m = \sum_n w_{m,n} \cdot x_n$ are approximately Gaussian. This is caused by a long addition of random variables, which resolves into an appropriate number of convolutions of probability distributions. When a large enough number of densities convolves together, then the resulting distribution will be very close to a Gaussian distribution, and therefore it can be approximated by it.

Even if the CLT does not take place (or has not taken place yet), a Gaussian is still a valid approximation since it universally gradually gives more probability to the closest grid points without any skew. Another valid option for the distribution assumption would be the Logistic distribution, which resembles the normal distribution, but its cumulative distribution function is much easier to compute.

The probability of a scalar random variable x being quantized to each grid point g_i from $\tilde{G}$ with the step size α is then given by:

$$P(\tilde{x_j} = g_i) = F_j(g_i + \alpha/2)) - F_j(g_i - \alpha/2)) \tag{3.28}$$

The function $F_j(\cdot)$ represents the CDF of the normal distribution $\mathcal{N} \sim (\mu_{in,j}, \sigma^2_{in,j})$, parameters μ_{in_j} and $\sigma^2_{in_j}$ are the inputs of this quantization layer in related to the random variable x_j. The formula for F_j is then computed as:

$$F_j(x) = \frac{1}{2} + \text{erf}(\frac{x - \mu_{in,j}}{\sqrt{2} \cdot \sigma_{in,j}}) \tag{3.29}$$

$$\text{erf}z = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \tag{3.30}$$

From now on, will the notation omit the j index since the focus will be on the evaluation of a particular input random variable x without regard to its position in the input structure.

For extreme cases, when g_j is either the highest or lowest grid point, the appropriate item in the equation is replaced with $F(\pm\infty)$. The process of quantization is the same as in RQ and can be viewed in the Fig.2.3. After computing the probability for every grid point, the mean and variance are calculated by a simple substitution in the equations 3.1 and 3.2:

$$\mu_{\tilde{x}} = \mathbb{E}[\tilde{x}] = \sum_{i=0}^{K-1} g_i \cdot P(\tilde{x} = g_i) = \sum_{i=0}^{K-1} g_i \cdot (F(g_i + \alpha/2) - F(g_i - \alpha/2)) \tag{3.31}$$

$$\sigma_{\tilde{x}}^2 = \mathbb{V}[\tilde{x}] = \mathbb{E}\left[(\tilde{x} - \mathbb{E}[\tilde{x}])^2\right] = \mathbb{E}\left[\tilde{x}^2\right] - (\mathbb{E}[\tilde{x}])^2 =$$
$$= \sum_{i=0}^{K-1} g_i^2 \cdot P(\tilde{x} = g_i) - \mu_{\tilde{x}}^2 = \sum_{i=0}^{K-1} g_i^2 \cdot (F(g_i + \alpha/2)) - F(g_i - \alpha/2)) - \mu_{\tilde{x}}^2 \tag{3.32}$$

The formulas 3.31 and 3.32 can be further analysed to achieve additional computational simplification. The simplification is based on the fact that two neighboring grid points share the same CDF value, which is in the current equations computed redundantly twice. The process is evaluated in detail in the appendix A.3. The resulting formulas are:

$$\mu_{\tilde{x}} = \alpha \cdot (K-1) + g_0 - \alpha \cdot \sum_{i=1}^{K-2} F(\alpha \cdot i + g_0 + \alpha/2) \tag{3.33}$$

$$\sigma_{\tilde{x}}^2 = (\alpha(K-1) + g_0)^2 - \sum_{i=1}^{K-2} (\alpha^2 + 2\alpha^2 i + 2\alpha \cdot g_0) F(\alpha \cdot i + g_0 + \alpha/2) - \mu_{\tilde{x}}^2 \tag{3.34}$$

The parameter α represents the distance between two neighbouring grid points, g_0 is the offset of the grid $\tilde{G}$, and i is the index of the i-th grid point. A substitution $(\alpha \cdot i + g_0) = g_i$ can be made in both equations.

Propagation of Statistical Moments with Noise Injection

The whole process of simulating a non-deterministic quantization is analogous to the deterministic version. The only difference is caused by the injected noise ϵ, which could alternate the input mean and variance. The output of this modified quantization is:

$$\tilde{x} = \lfloor x + \epsilon \rceil_{\tilde{G}} \tag{3.35}$$

In the case when x is deterministic, the mean and variance considered for the quantization process are purely dependent on the distribution of the input noise. When x is a random variable, the distribution of the input x is set following the idea from the previous part. Due to the addition of random variables throughout the network, the CLT takes place, and a Gaussian approximates x. The addition of the noise ϵ now represents only another convolution resulting in a distribution even closer to a Gaussian. Thus the only necessary information about the noise ϵ for stochastic input x is its mean and variance with no regard to the type of its distribution.

When the noise ϵ is virtually added to x, it influences the mean and variance of the random variable x. The changes follow the properties 3.5 and 3.6. Since ϵ is added externally, it is completely independent of the input properties. The new mean and variance are then:

$$\mu_{noised} = \mu_{in} + \mu_{\epsilon} \tag{3.36}$$

$$\sigma^2_{noised} = \sigma^2_{in} + \sigma^2_{\epsilon} \tag{3.37}$$

The new parameters are used to define the CDF of the assumed propagating PD. The probabilities of selection of each grid point are calculated, and at last, the output mean and variance is obtained in the same matter as in the deterministic quantization method (3.31, 3.32). The only difference between the two methods is the offset of mean and variance in the CDF evaluation $P(x = g_i)$.

3.1.4 Average Pooling

Average pooling is de facto an affine layer with all weights set to one and the bias set to zero. The resulting mean and variance are then normalized by the number of input values. For example, if average pooling is supposed to be done over $k = 4$ random pre-activations $\{a_1, a_2, a_3, a_4\}$ then a small affine layer is used instead with only one output a_{out}.

$$\mathbb{E}\left[a_{out}\right] = \mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k} a_i \cdot 1\right] = \frac{1}{k} \cdot \sum_{i=1}^{k} \mathbb{E}[a_i] \tag{3.38}$$

$$\mathbb{V}\left[a_{out}\right] = \mathbb{V}\left[\frac{1}{k}\sum_{i=1}^{k} a_i \cdot 1\right] \approx \frac{1}{k^2} \cdot \sum_{i=1}^{k} \mathbb{V}[a_i] \tag{3.39}$$

3.1.5 Max Pooling Layer

The max layer is commonly used in various NNs. The problem arises when the input set of compared values has different means and variances. Even if a random value has a slightly different mean than another, if they have high variances, then their sampled values can significantly vary. This can be viewed in the figure 3.2. Therefore the problem of selecting one highest value from a set of means and variances is unsolvable without exploiting some heuristic. There are several possible heuristics applicable to exploiting different approaches with different computational complexity. Some of max-pooling heuristics were described in [NK08a], [PW18], or in [SHYS20]. In the following sections, multiple methods will be shown.

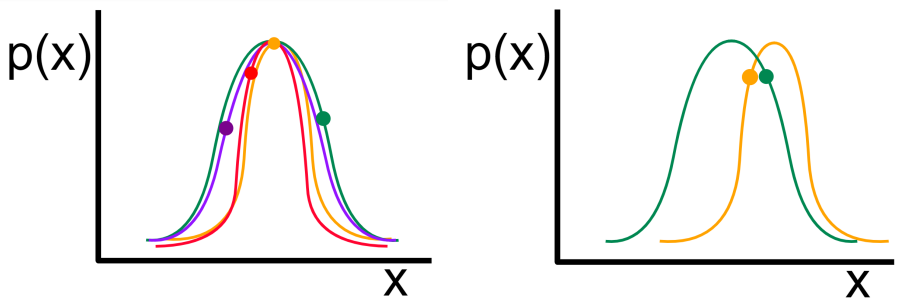


Figure 3.2: On the left: Samples from multiple random variables with almost identical mean values can still varry significantly. **On the right:**Even when the mean value of one random variable is unequivocally larger, its sample can be still smaller since the other random variable has large variation.

Simplification to Basic Pooling

The simplest method ignores the max function and only performs pooling by subsampling to downsize the input dimensions for output propagation. This can be achieved by always selecting the same index position as the maximum value or by selecting a random position for each forward pass. This approach is not ideal, but for NNs where the main idea of the Max Layer was to downsize the propagating values, it is still applicable. It can also be used to quickly substitute other more complicated pooling layers.

Selecting the Highest Mean

This heuristic is self-explanatory. The variance of a random variable is ignored altogether, and the forward propagation is based on the vanilla MaxPool method for full precision deterministic inputs. In the case of a large pooling, this method can be modified by selecting the top m highest means and then selecting one of them randomly [SHYS20]. When a mean of an input random variable is selected for propagation, the variance of the same variable will be propagated.

Substitution for Average Pooling

This option is rather a different architecture choice than a method of max pooling. The more problematic max-pooling layer is substituted with average pooling from the section 3.1.4. It proposes a more sophisticated method of ignoring the max function while keeping the pooling. In backpropagation, all the involved inputs of the max layer will be updated, which distinguishes this method from the rest. An example of such changing the architecture can be the toy LeNet5 architecture which is commonly implemented both with average and max-pooling layers.

Sampling in Evaluation of Max Layer

Another solution is to assume a normal distribution for the input values and sample from them. The maximum is selected from the sampled values. Then the properties of the selected distribution are passed forward either unchanged or with the variance set to zero. The latter option is based on the fact that the sampled value is fixed for future computations and thus has no variance.

This method is relatively straightforward without any significant complications. Peters & Welling also introduced in their work [PW18] another MaxPool approximation method based on sampling from a general distribution. However, Peters & Welling also concluded that the simple argmax propagation from the sampled values is more efficient and has satisfactory results. The elementary sampling process is shown in the figure 3.3.

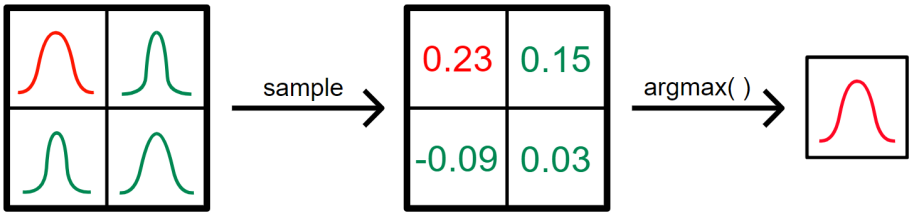


Figure 3.3: The sample&argmax process of selecting a PD which will be propagated through one max layer patch with the kernel size 2×2. First all distributions are assumed Gaussian and scalar values are sampled from them. After selecting the maximal sampled value, its parent distribution sharing the same index in the MaxPool kernel is propagated.

Repeated Approximate Maximum of Two Random Variables

Nadarajah & Kotz introduced in their paper [NK08b] a process of gaining the variance and mean of the max operation over two random variables which are both normally distributed. With the notation $s = (\sigma_X^2 + \sigma_Y^2 + 2\mathrm{Cov}(X,Y))^{\frac{1}{2}}$ and $a = (\mu_X - \mu_Y)/s$, the formulas for mean and variance are:

$$\mu_{out} = \mu_X \cdot \psi(a) + \mu_Y \cdot \psi(-a) + s \cdot \phi(a) \tag{3.40}$$

$$\sigma_{out}^2 = (\sigma_X^2 + \mu_X^2) \cdot \psi(a) + (\sigma_Y^2 + \mu_Y^2) \cdot \psi(-a) + (\mu_Y \cdot \mu_X) \cdot \phi(a) \cdot s - \mu_{out}^2 \tag{3.41}$$

Since this works neglects minor dependencies caused by the linear and convolutional layer, the input random variables to the MaxPool layer are all assumed independent and therefore the covariance term in the formula for s is always zero.

The max evaluation for 2 values can be then arbitrarily chained. For example for MaxPool2d with kernel (2×2) and stride 2, every other column computes the max with its neighbouring column to the right, then every other row computes the max with its downwards neighbouring row. This process is also shown in the figure 3.4.

Figure 3.4: Elementary case of the process of the chained max function of two variables.

3.1.6 Convolution Layer

The convolutional layer is the cornerstone of CNNs. It contains multiple kernels with learnable parameters for each dimension it operates on. These kernels perform the convolution operation ($y = w \star x$) on the input data, predominantly multidimensional large matrices. The convolutional function sums over multiplications of items that share the same index in the kernel and in the actual window scope over the input matrix. For one dimensional convolution with weights/kernel of length m, the formula for the output with index i is:

$$(w \star x)(i) = \sum_{j=1}^{m} w_j \cdot x_{(i+j)} \tag{3.42}$$

This makes the propagation of mean and variance somewhat similar to the linear layer, with the difference that the weights are shared over many connections as the kernel moves around the input matrix. The dependence on shared weights can be viewed in the figure 3.1. The actual propagated parameters for a general N-dimensional convolution take shape:

$$\mu_{out} = \mathbb{E}[y] = \mathbb{E}[w] \star \mathbb{E}[x] \tag{3.43}$$

$$\sigma^2_{out} = \mathbb{V}[y] = \mathbb{V}[w] \star \mathbb{V}[x] + \mathbb{V}[w] \star \mathbb{E}[x]^2 + \mathbb{E}[w]^2 \star \mathbb{V}[x] \tag{3.44}$$

Both of the derived formulas are, in general, identical to the formulas for the linear layer (section 3.1.1). As it was previously foreshadowed, the only difference is the special indexation of the summed terms defined by the convolution function, which depends also on other parameters like stride or padding.

Independence of Units after the Linear and Conv Layers

The linear layer introduces only minor dependencies between the NN's units since the activations were predominantly dependent on the layer's weights, and therefore the dependence can be effectively ignored with impunity. A convolution layer and its shared weights introduce stronger dependencies because the kernel works with a small number of weights, which are shared across a large amount of input data. Moreover, this dependency weakens as the convolution layer is used repeatedly in the network. In the properties 3.8 and 3.9 is assumed absolute independence between the two multiplication

participants. Since these two properties are essential for all derivations for the propagation of the statistical moments in PNN, not fulfilling the independence requirement introduces some inaccuracy to the network. A possible solution is proposed in a white paper by Roth [RSFP19], and Kingma [KSW15]. However, their method exploits sampling, which this work tries to limit as much as possible.

It is not in the scope and aim of this work to further tackle this issue. The PNN approach is primarily used to speed-up the initial training, so it can be later substituted with a more accurate sampling-based method like the relaxed quantization 2.3.2. The issue of higher variance introduced by the dependencies is thus irrelevant for further steps in this work. Therefore the notable influence of the convolutional layer on the NN's unit independence is neglected.

3.1.7 LogSoftmax Function

The LogSoftmax is the more numerically stable version of 'log of Softmax'. The Softmax function is used as the activation function in the last layer of NNs, which require their outputs to be represented as probabilities. This form of results is especially useful for classification tasks with N possible class outputs. The LogSoftmax normalizes the output of a NN, so the resulting probabilities lie on a logarithmic scale instead of the softmax's interval $\langle 0, 1 \rangle$.

The formula for LogSoftmax with n output scalars is:

$$\text{LogSoftmax}(x_i) = \log\left(\frac{\exp(x_i)}{\sum_j^n \exp(x_j)}\right) \tag{3.45}$$

When trying to derive the mean and variance propagation through this layer, an issue arises. Since the fraction's numerator and denominator are dependent. Therefore no properties from the section 3.1 can be applied. Since this layer is present only once in the whole network, a computationally inefficient and otherwise avoided approach of gaining the mean value through sampling is applied.

Let's assume the fully propagated PD's mean μ_{in} and variance σ_{in}^2. The sampled values s_i will be assumed to be from a normal distribution $\mathcal{N} \sim (\mu_{in}, \sigma_{in}^2)$. Each of sampled values can be decomposed following the idea of

the *reparametrization trick* [KSW15]:

$$s_i = \mu_{in,i} + \sigma^2_{in,i} \cdot \xi \qquad \xi \sim \mathcal{N}(0,1) \tag{3.46}$$

When the reparametrization trick is used, the gradient flow of the SGD process is clear. The samples for each output are drawn T times (i.e. $T = 10$). The number T is selected, so the estimation represents the probabilistic moments well, but the number of samples is not excessive. The mean value then can be computed as the arithmetic mean:

$$\begin{aligned}
\mu_{out,i} = \mathbb{E}\left[\log\left(\frac{e^{x_i}}{\sum_j^n e^{x_j}}\right)\right] &\approx \frac{1}{T}\sum_{t=1}^{T}\left(\log\left(\frac{e^{s_i(t)}}{\sum_j e^{s_j(t)}}\right)\right) \approx \\
&\approx \frac{1}{T}\sum_{t=1}^{T}\log\left(e^{s_i(t)}\right) - \frac{1}{T}\sum_{t=1}^{T}\log\left(\sum_{j=1}^{n} e^{s_j(t)}\right) \approx \\
\approx \frac{1}{T}\sum_{t=1}^{T} s_i(t) - \frac{1}{T}\sum_{t=1}^{T}\log\left(\sum_{j=1}^{n} e^{s_{j=1}(t)}\right) &\approx \mu_{in,i} - \frac{1}{T}\sum_{t=1}^{T}\log\left(\sum_{j=1}^{n} e^{s_{j=1}(t)}\right)
\end{aligned} \tag{3.47}$$

The variance of the LogSoftmax layer in most cases does not need to be calculated since LogSofMax is usualy only exploited in the last layer of a PNN and the loss function does not generally take variance into account. However, for completeness of the method, variance is still derived. The intermediate calculations cannot be simplified and hence it needs to be calculated by brute force from the variance property 3.2:

$$\begin{aligned}
\sigma^2_{out,i} = \mathbb{E}\left[\left(\log\left(\frac{e^{x_i}}{\sum_j^n e^{x_j}}\right) - \mu_{out,i}\right)^2\right] &= \mathbb{E}\left[\log^2\left(\frac{e^{x_i}}{\sum_j^n e^{x_j}}\right)\right] - \mu^2_{out,i} \approx \\
&\approx \frac{1}{T}\sum_{t}^{T}\left(\log\frac{e^{s_i(t)}}{\sum_j^n e^{s_j(t)}}\right)^2 - \mu^2_{out,i} \approx \\
&\approx \frac{1}{T}\sum_{t}^{T}\left(s_i(t) - \log\left(\sum_{j}^{n} e^{s_j(t)}\right)\right)^2 - \mu^2_{out,i}
\end{aligned} \tag{3.48}$$

Cross Entropy Loss

The cross entropy loss is commonly exploited in classification tasks. The formula for this layer is:

$$L_{CE} = -\sum_{i=1}^{n} t_i \log(p_i) \tag{3.49}$$

The parameter n is corresponds to the number of output classes, $t = [t_1, \ldots, t_n]$ is the one-hot encoded ground truth of the classification task and p_i is the softmax probability of the class i. The formula considers as inputs only the input mean values, therefore it is unnecessary to compute the variance of the LogSoftMax function. The propagated value of the CE loss is the identity of minus log probability of the ground true class.

3.1.8 ReLu and Leaky ReLu

These activation functions are unchanged in PNNs in comparison to their standard functionality. For the most part, they have no influence on the propagating PD. When the mean value is smaller than zero, the mean value will be clipped to zero for ReLu or significantly scaled for leaky Relu. Luther showed in his paper [LS19] that after a specific weight initialization, variation in deep MLP networks with ReLu activations decays. Therefore variance will be approximated 'by the worst case' - it will not be affected when the clipping or scaling takes place.

A more precise approach was shown by Schekhovtsov & Flach in [SF19], where the propagated mean and variance is derived from a formula for propagation of mean and variance for maximum of two random variables $Z = \max(X, Y)$, this formula is described in detail in the section 3.1.5. Then for ReLu applies $Y = \max(0, X)$, for LeakyReLu applies $Y = \max(\alpha X, X)$ and the propagated parameters can be evaluated.

3.1.9 Combining the Elementary Layers

The above-derived layers can be freely combined into various DNNs. All the elementary layers are capable to propagate a PD by updating its statistical moments instead of the propagation of point estimates used in standard NNs. In some cases, a more complicated layer can be constructed based on the elementary ones. An example is multidimensional Batch Normalisation or a Convolution layer, which can be built from the elementary layers.

Chapter 4

Implementation and Experiments

4.1 Implementation

The two chapters 2 and 3 theoretically described multiple approaches to quantization. The aim of the implementation section is to introduce the general idea of how to structure the code framework, so it supports multiple methods of quantization for a singular network architecture setup. The transfer between all the methods within the framework should be achieved only by changing the network's settings via an input to the network. Such implementation requires a complex set of classes based on the original **PyTorch** framework. When a smooth substitution between the training methods, pre-training for some more complex tasks can be easily exploited.

E-Blocks

Since the main building structure of QNNs is the same as for SNN, most attributes of the original class **nn.Module** are shared. However, some propagation methods and other attributes need to be modified or entirely replaced. Therefore all NN structure classes inherit from a new Module class:

```
class EModule(nn.Module):
    def super_forward(self, *args) -> Tensor:
        return super(type(self), self).forward(*args)
```

Later in the implementation, each artificial layer will inherit this as their parent class. This will later on serve as a sign of a layer that requires additional inputs specifying the quantization approach. The same change is also done for the other NN structure block **nn.Sequential**:

```
class ESequential(EModule, nn.Sequential):
    def forward(self, x, method: Method, **kwargs):
        return method.forward_Sequential(self, x, **kwargs)
```

Q-Layers, E-Layers

The Q-Layer class is exploited for layers, in which weights are quantized before application within the layer. This is the case for the linear and convolutional layers, which all share the same formula for propagation from the section 3.1.1. The only difference between such layers is the process of selecting the inputs and weights for the equations from 3.1.1. Therefore **QAnyLinear** layer is set up, which solves the propagation of the weighted sum function and quantizes the input weight. The class **QAnyLinear** is then inherited as a parent class by the before mentioned layers.

Functions that evaluate the quantization methods are then united within the QReLu class. QReLu at first scales and offset its input (weight or activation) which is then squashed within the range of the quantization grid G. Then is carried out, the selected quantization function and the result is scaled and offset again, resulting in a value from the grid $\tilde{G}$. The same is then done for the activation value of the given layer. This process is graphically described in the figure 4.1.

During the testing of a QNN, the complex quantization process of weights and activations can be replaced by a computationally more advantageous process depicted in the figure 4.2.

An addition to the Q-classes is the E-Layer class which is not affected by quantization in propagation, however, its forward passes needs to be restructured for the probabilistic learning.

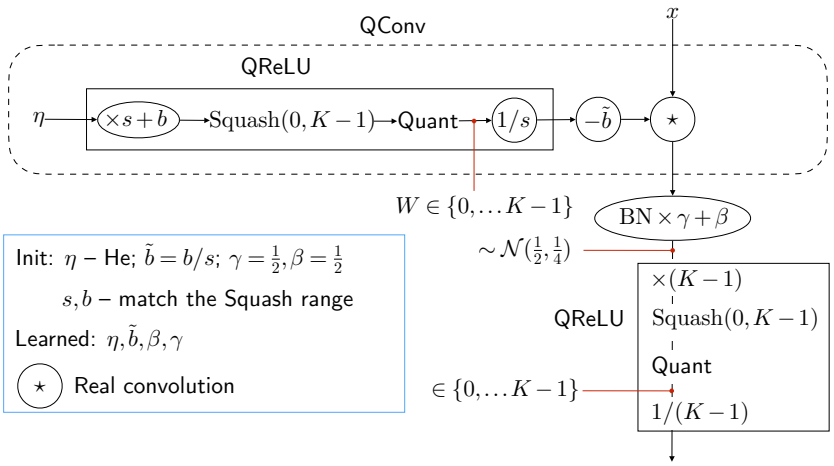


Figure 4.1: Diagram of the implementation of a Q-Layer, in this case the QConv layer. It also shows the process of quantization of activations.

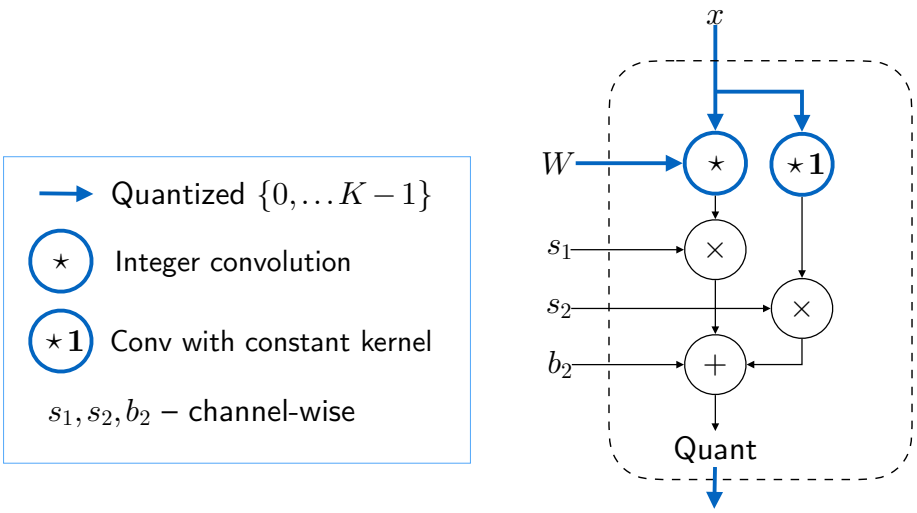


Figure 4.2: Diagram depicting the implementation of a QConv layer during test time. The quantization of weights and activations is not exploited, however the scaling and offset initially introduced for quantization still need to take place.

All the artificial layer classes inherit from either the **EModule** or the **ESequantial** classe, and from their standard full precision counterparts. For the deterministic, sampling-based, and full precision methods are the layers, which do not exploit quantization, unchanged in their forward pass, and therefore they can exploit the original **PyTorch** implementation, for example:

```
class EMaxPool2d(EModule, nn.MaxPool2d):
    def forward(self, x, method: Method, **kwargs):
```

```
        return method.forward_MaxPool2d(self, x, **kwargs)
```

When the forward propagation of a layer is not identical to the original PyTorch implementation, the new forward function is set after calling the selected quantization method class. A particular case of this redefinition is the quantization layer, which is by default initialized as the STE method. Therefore all other methods, including the full precision one, must re-define it.

Quantization Parameters - class dotdict(dot)

Detailed information about the quantization process, such as the quantization method, the number of available bits, or the noise distribution, is stored in one dot.notation dictionary. During a NN's initialization, this dictionary is shared to layers requiring quantization, i.e., Linear, Convolution layers. The layers make a deep copy so that they can train some of the dictionary's components, such as the noise distribution parameters.

The dictionary can also hold other general information about the training process and conduct. Some other information it can hold is the number of epochs that need to be completed before switching from the probabilistic pre-training to the sampling-based methods. Therefore a NN network based on this implementation can be set entirely for various training exploiting multiple approaches only by a short setting of the dictionary.

Method

All methods will share a parent class **Method**, which manages the propagation process between layers. It contains definitions propagation through elementary layers, including the quantization layer, which will then later on be changed depending on the specific quantization approach.

Every quantization-aware training method will then inherit this class and re-define propagation functions for layers, that are affected by the approach. A simple example is the **MethodReal** class, which keeps all original layer definitions and only evades the quantization layer:

```
class MethodReal(Method):
    def forward_quant(self, quant: Quant, x: Tensor):
        """ no quantization"""
        return x
```

4.1.1 Deterministic Quantization Method

This implementation is based on the theory from the section on STE 2.2. The method assumes the gird G with grid points $g_i \in [0, 1, \dots 2^b - 1]$, where b is the number of available bits. The input x is then rounded to the closest integer, and the resulting integer value is then clamped in regard to the first and the last grid point value. For the forward propagation is exploited a notation trick that ensures uninterrupted gradient flow:

```
x = x_quant + (x - x.detach())
```

4.1.2 Relaxed Quantization Method

This implementation is based on the theory and formulas from the section on relaxed quantization 2.3.2. In theory, were multiple different approaches introduced within the RQ method. The CDF evaluation over extreme grid points in the section 2.3.2 will exploit the version without truncation, and Quantization on a Local Grid from the section 2.3.2 is not implemented. Then the output value is evaluated in 4 steps, where σ^2 is the noise variation, $\epsilon = 1e - 10$ is a non-negative small constant to keep numerical stability:

1. interval points evaluation: $thresholds = [G - 0.5, g_{K-1} + 0.5]$
2. CDF evaluation: $p = \text{Sigmoid}((thresholds - x)/\sigma)$
3. concrete distribution: $y = \text{F.gumbel\_softmax}(\log(p))$
4. ouput expectation evaluation: $out = \sum_{i=0}^{K-1} y_i \cdot g_i$

4.1.3 Probabilistic Learning Method

This approach required a re-definition of the forward propagation for all elementary layers. The derivations of each re-definitions were introduced one by one in the section 3.1.

All the forward passes de facto only execute the derived formulas. For the MaxPool layer was selected the process of repeated maximums of two random variables. Its implementation follows the algorithm shown in the figure 3.4.

For a more robust, but slower evaluation of the quantization layer, the basic formulas 3.31, 3.32 can be exploited. Other option is to implement the faster method with formulas 3.33 and 3.34, which can be simplified since the grid is scaled and offset after the quantization. Thus $\alpha = 1$ and $g_0 = 0$. The assumed normal distribution is symmetrical around its mean and thus the final equations for the layer are A.22 and A.32:

$$\mu_{\tilde{x}} = \sum_{i=1}^{K-1} F(-i-0.5) \tag{4.1}$$

$$\sigma_{\tilde{x}}^2 = (K-1)^2 - \sum_{i=1}^{K-1} (1+2i)F(i+0.5) - \mu_{\tilde{x}}^2 \tag{4.2}$$

Random Variable

While working with PNN, the network needs to process and forward two parameters, the mean and variance of the propagating distribution. This requires the implementation of a new class, which will inherit the attributes of the original **torch.Tensor** class. This class is called **RandomVar** and contains two Tensors with the same shape as in SNN. One of the tensors is for the propagating mean, the other for the variance. The **RandomVar** class also has additional attributes supporting several mathematical operations over random variables. These attributes are based on the formulas from the section 3.1. The **RandomVar** class also supports sampling from the Normal distribution with regard to its current mean and variance.

4.1.4 Other Quantization Methods

Multiple other methods can be implemented based on the theory from this work. However, in the majority, they are only a mix of the three already introduced method classes. Their implementation is therefore based only on minor changes in the forwarding functions for each layer. For example, the stochastic rounding method only changes the quantization forward function by implementing a different noise and exploits the notation trick from STE to achieve uninterrupted gradient flow.

4.1.5 Pre-Training

One of the goals of the thesis was to implement selected quantization layers in a universal framework so that different networks could be exploited during the training of one model network. Since all non-probabilistic methods do not significantly change the network's structure and at most introduce some extra parameters for learning, the only step required is to change the forward function for the layer affected by the change.

The transfer from a probabilistic to a non-probabilistic method is needed to evaluate the point estimates of all weights. One option is to sample each parameter's distribution n-times and set the mean value as the initial weight for the standard NN. The other, much faster option is to simply set all weights as the corresponding distribution's mean. For transfer from a non-probabilistic to a probabilistic method, weight distribution means are set as the previous weight values, and the variance is set to any meaningful value, i.e., 0 or 1/3.

Statement of Contribution

Two different implementations were written with the structure explained in this section to test the introduced methods in the theoretical part of the theses. Both implementations were then exploited for experiments.

In the first implementation, only a rough code structure was provided, and I implemented all the methods, training procedures, and support functions myself. An exception to this was the QReLu function (depicted in 4.2) which was implemented by the thesis supervisor Mgr. Oleksandr Shekovtsov, PhD.

In the second implementation were, all methods apart from the **MethoPP** class implemented by Dr. Oleksandr Shekovtsov. Therefore any inaccuracies in the code would not affect the results of the more complex experiments. The theoretical structure behind the implemented methods was following the equations, which were described in the chapter 2.3.2. My contribution to the second implementation was independent derivation and implementation of the probabilistic learning method within the **MethoPP** class. The class was then added to the more precise framework for experimenting.

Chapter 5

Experiments

The outlined implementation structure from the previous section was adopted twice. The first implementation focused on the sampling-based and STE methods of quantization. The second implementation was more thorough, introduced more learnable parameters, improved the training/validation evaluation, and primarily introduced the extension for probabilistic learning.

5.1 First Implementation

The structure of the code in the first implementation was naive, it did not initialize all the feasible parameters, and the overall initialization of the network structure was not optimal. Nevertheless, the goal target structure for training a network with STE, RQ, SR was achieved. With the naive implementation was build a toy LeNet-5 network that trained on the MNIST dataset. Multiple tests were run on this model with the focus on the general performance with united settings for all methods, including the full precision. The training with quantization was set to quantize both activations and weights with the same precision.

During the training of the networks was tested how computationally/time expensive are they to run. The results are shown in the table 5.1. In comparison to the values in the table, the original full precision architecture took, on average, 18.5 seconds per epoch. The data from the table shows that the additional computations and sampling in RQ resulted in a significant

slow down of the training. Even at the 1-bit precision, evaluating a single epoch takes double the time. The results also show that stochastic rounding is, in this matter, close to the fast STE method as expected.

bit precision	average epoch time [seconds]		
	STE	**SR**	**RQ**
1	20.93	22.35	42.25
2	20.81	22.37	43.59
4	20.48	22.31	115.13
6	20.76	22.41	265.2
8	20.73	22.42	420.36

Table 5.1: Time taken to complete one learning epoch on LeNet5 exploiting the STE, SR, or RQ method for quantization with different precision. The listed values are the average time scores over 40 epochs carried out on a personal PC with limited computational resources.

naive imp.	RQ 1 bit	STE 1 bit	SR 1 bit	RQ 2 bit	STE 2 bit	SR 2 bit	Real
accuracy [%]	96.02	98.43	98.52	97.92	99.12	99.19	99.40

Table 5.2: Validation accuracy after 60 epochs of training the LeNet5 network on the MNIST dataset with multiple quantization and full precision methods in the naive environment implementation.

The validation accuracy results with the naive method setup after 60 epochs are shown in the table 5.2. The experiment settings are listed in the appendix B.1. STE and SR both reached the validation accuracy of over 99 % with 2-bit quantization. Gupta's stochastic rounding even achieved accuracy comparable with the original full precision network. With the same precision, RQ achieved an accuracy of 97.88 %. RQ also trained significantly slower than the STE and SR. The results are shown in the figure 5.1. Since the SR method is implemented as STE with uniform noise, the training progress is fairly similar. While the results for STE and SR are satisfactory, the training for RQ implementation was not successful.

The outcome of this set of experiments was that the naive quantization framework was capable of successfully training a simple NN with non-binary weights and activation. The RQ method proved to be computationally more expensive, and the accuracy results from its elementary form underperformed the SR and STE methods. Since the RQ's performance on 2 bits precision was underwhelming in comparison with the experiments conducted by Louizos [LRB+18], it was concluded that a better structured and initialized implementation is needed for RQ to work correctly.

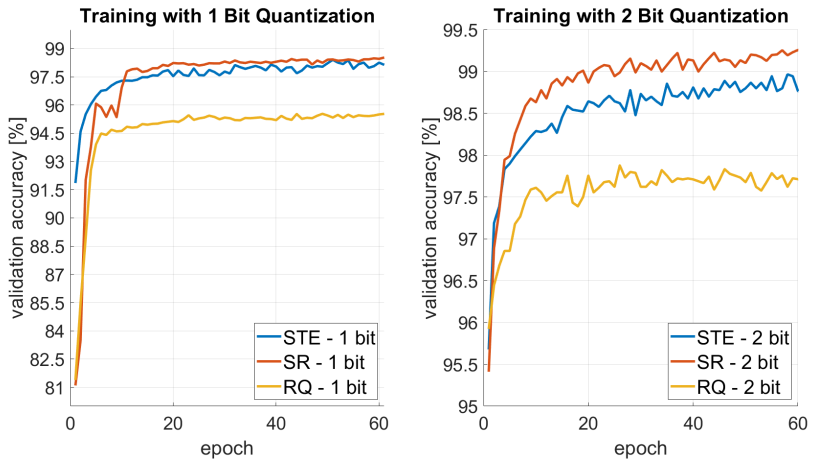


Figure 5.1: The validation accuracy results of training LeNet5 NN with 1 and 2 bit precision quantization of weights and activations. Naive implementation.

5.2 Second Implementation

The second implementation achieved a better structure, initialization of parameters was improved, and some quantization-related functions were updated. The training loop was upgraded, so it supported more training settings, and additional optimization methods were added. Most notably was added the class **MethodPP** for the probabilistic propagation.

At first, was tested the performance of the non-probabilistic learning methods so a comparison between the two implementations could be made. Therefore the experiment setting still follows appendix B.1. The resulting accuracy for 1 and 2-bit precision is in the table 5.3. In general, the methods training with the second implementation achieved better accuracy results. The stochastic rounding method proved to be fast in training and its accuracy, even at 1-bit precision, was over 99 %. In addition to the first implementation was also trained a NN with probabilistic learning. As was expected from the theoretical analyses, it learned fast, but it could not reach the best accuracy.

2nd imp.	RQ 1 bit	STE 1 bit	SR 1 bit	PNN 1 bit	RQ 2 bit	STE 2 bit	SR 2 bit	PNN 2 bit
acc. [%]	98.96	99.10	99.23	98.85	99.19	99.25	99.29	99.23

Table 5.3: Validation accuracy after 60 epochs of training the LeNet5 network on the MNIST dataset in the secondary implementation framework for 1 and 2 bit precision.

The next goal was to pre-train a NN with the probabilistic learning, for which it was expected that the initial optimization would be faster. The experiment was set so that SR or STE substituted the probabilistic learning method after 60 epochs.

The original STE and the pre-trained STE training were run in parallel, so the expected faster training for PNN could be observed. For any optimization speed difference to be noticeable, the more complicated Fashion-MNIST dataset was trained. During the pre-training period, the NNs with probabilistic learning quickly reached accuracy of over 90 %. After the switch to the STE method, the evaluated training loss immediately rises approximately to the values observed for the pure STE training after the same amount of optimization epochs. This can be seen in the figure 5.3. In the figure 5.2 is plotted the accuracy during training. After the switch from probabilistic learning, the accuracy for single sampling validation dropped even below the current accuracy obtained by the pure STE method. Such behavior was unexpected and remained even during further experiments on the FMNIST dataset with various training settings.

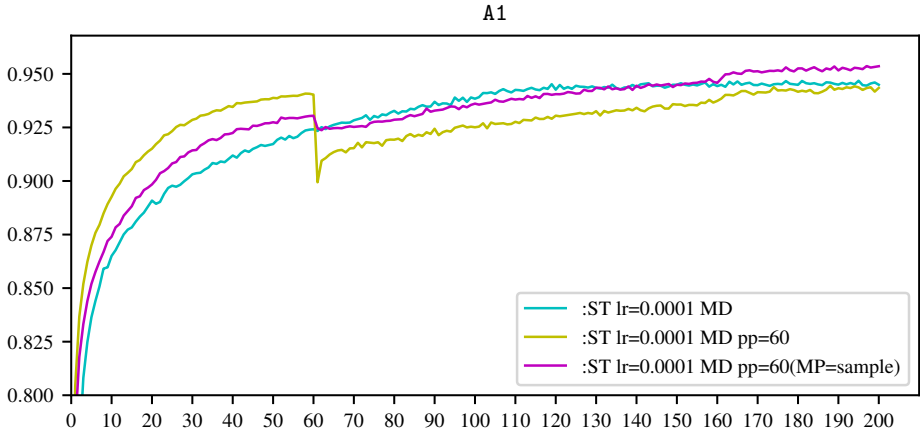


Figure 5.2: Training accuracy evaluation of basic STE and pre-trained STE quantization methods with Mirror Descent policy optimalization for the FMNIST dataset.

At last, was experimented with the CIFAR-10 dataset with the allCNN architecture described in the appendix B.2. In this case, the pre-trained STE method did initially train faster. After the switch, its train accuracy dropped (figure 5.5), and the loss evaluation rose above the results of network training without pre-training.

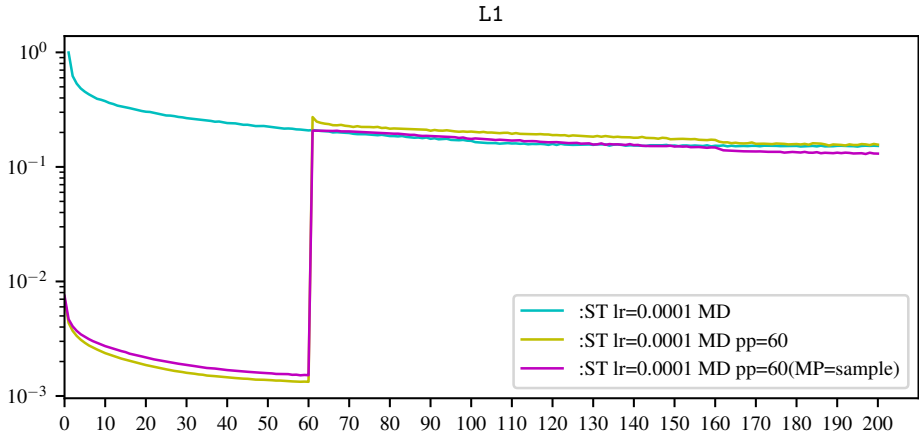


Figure 5.3: Training loss evaluation of basic and pre-trained STE quantization methods with Mirror Descent policy for optimalization for the FMNIST dataset.

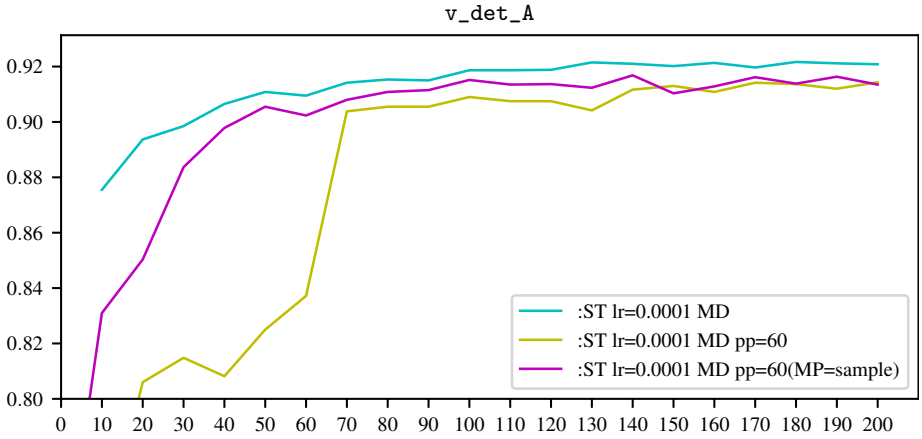


Figure 5.4: Validation accuracy evaluation of basic STE and pre-trained STE quantization methods with Mirror Descent policy optimalization for the FMNIST dataset.

Such results were not expected based on the theoretical work in the chapter 3. The validation results show that the pre-training does not reach better accuracy or loss even during the initial phase, when its training accuracy is higher (figures 5.6, 5.4). The validation accuracy for the probabilistic learning method was estimated with deterministic quantization, which showed better results when multiple samples were averaged to gain the deterministic weight representation.

From the loss evaluation in figures, 5.3 is clear that the STE and probabilistic learning methods have different training targets. Therefore their loss evaluation is significantly different, whereas the accuracy results are not distant. After the switch from pre-training, the training target also changes. The current network structure cannot adapt to the change of targets and merely restarts the optimization from a worse position.

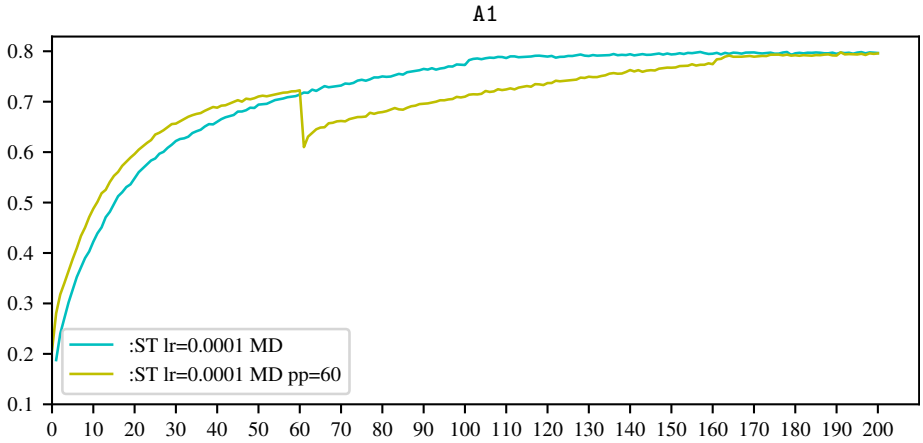


Figure 5.5: Training accuracy evaluation of basic STE and pre-trained STE quantization methods with Mirror Descent policy optimalization for the CIFAR dataset.

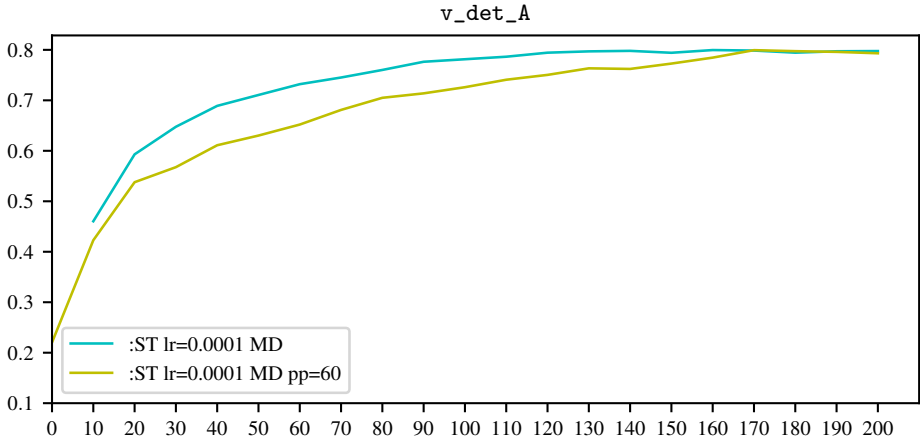


Figure 5.6: Validation accuracy evaluation of basic STE and pre-trained STE quantization methods with Mirror Descent policy optimalization for the CIFAR dataset.

Chapter 6

Discussion

Low resource neural networks have received significant interest in the last few years. In the past, most approaches focused on the post-training quantization and the pruning of redundant connections in DNNs. Such methods lead to significant improvements. However, the power-hungry CNN networks require further computational and memory savings to run on power-constrained devices. Therefore the first methods started to focus on quantization aware training, which would be able to solve the problem of backpropagation through the discontinuous discretization function. These methods could be roughly divided into gradient approximations with STE, sampling-based quantization, and probabilistic training.

This work theoretically covered quantization with probabilistic learning, stochastic rounding, and the STE gradient estimation method. All the methods were implemented and tested in experiments. The stochastic rounding and the relaxed quantization introduced sampling-based propagation. Relaxed quantization and probabilistic learning enabled fully continuous gradient flow without reparametrization tricks. Unlike the stochastic quantization methods, probabilistic learning does not require any sampling outside the cross-entropy loss function, making it faster to evaluate and applicable on hardware without support for sampling. This approach to quantization is related to the approach for full precision training, and analytic dropout in [SF19] or for training networks with quantized weights and binary distributions in [RSFP19].

Other approaches for probabilistic learning of QNNs ([MBK20], [GSS21], [JSS21]) exploit Bayesian learning chain model which also achieving good results. All the probabilistic learning approaches contain a certain level of

approximation, however, unlike SR or STE, they do not build upon any heuristics, and their results are therefore better theoretically justifiable. The significant advantage of probabilistic propagation is that it can train with the speed of a real-valued network. It does not suffer from low signal-to-noise gradients, typical for the STE method. Therefore, it can be, in theory, exploited in training complex problems for which are the basic quantization methods inapplicable.

The initialization training with the **MethodPP** can also be advantageous for networks where the initial gradient for STE and SR methods is too small. Therefore, the network does not learn. Since the probabilistic learning has a better signal-to-noise ratio and generally trains faster during the initialization phase, it can overcome the starting low gradients with relative ease and prepare the network for more accurate learning with SR/STE after gaining training the weights into a less complicated optimization space.

However, the expected good results from pretraining of sampling-based methods with the derived probabilistic learning did not occur. The experiments were otherwise successful in training QNNS for the MNIST and FMNIST datasets with 1 and 2 bits precision only while maintaining accuracy over 99 %. In theory, the pretraining approach, which was unsuccessful for the relatively simple datasets, could improve when implemented on more challenging tasks like the ImageNet, where QNNs do not reach good accuracy without pretraining. This should be tested in further work on this topic.

6.1 Conclusion

This work introduced several popular quantization-aware approaches and stated the general problem of stochastic QNNs. The problem was then solved by different methods of stochastic rounding, relaxed quantization, and probabilistic learning, which were all theoretically described and implemented for experiments. To establish a quantized probabilistic neural network, formulas for propagation of the first two statistical moments were derived. All the introduced methods were included in an environment for quantization aware training in **PyTorch**. The framework allows for the shared implementation of a network architecture, which can be trained with different quantization methods only by parameter setting.

In experiments were QNNs with the extreme case of BNNs trained on the MNIST dataset. The resulting accuracy for MNIST was nearly identical to the accuracy acquired from a full precision network of the same architecture. The stochastic rounding method achieved satisfactory results, even though its structure was less complex than for the relaxed quantization. Relaxed quantization and probabilistic learning for 1-bit precision achieved slightly worse results than the other methods. For the 2-bit precision, all methods were more or less identical. More experiments were performed with the CIFAR-10 and FMNIST datasets with the aim of accelerating the training period by pre-training with the probabilistic method. However, with the current implementation and experiment settings, no improvements to the training speed were achieved. All methods were otherwise able to achieve validation accuracy of over 80 % with basic training settings.

Future work on this topic should extend the benchmark results to other models and expand the environment to utilize more complex neural network architectures. It should reintroduce and improve the current pre-training approach and show its results on quantization tasks that are unsolvable by the standard sampling-based quantization due to the low signal-to-noise gradient.

Appendix A

Derivations for BN Mean and Variance Propagation

A.1 Mean of BN variance

The derivation of the mean of a BN variance σ^2_{BN}:

$$\sigma^2_{BN} = \frac{1}{n}\sum_{i=1}^{n}(a_i - \mu_{BN})^2 \tag{A.1}$$

$$\mathbb{E}\left[\sigma^2_{BN}\right] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(a_i - \mathbb{E}\left[\mu_{BN}\right])^2\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[(a_i - \mathbb{E}\left[\mu_{BN}\right])^2\right] =$$

$$= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[a_i^2 - 2\cdot a_i \cdot \mathbb{E}\left[\mu_{BN}\right] + \mathbb{E}\left[\mu_{BN}\right]^2\right] = \tag{A.2}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\mathbb{E}\left[a_i^2\right] + \mathbb{E}\left[-2\cdot a_i \cdot \mathbb{E}\left[\mu_{BN}\right]\right] + \mathbb{E}\left[\mathbb{E}\left[\mu_{BN}\right]^2\right]\right)$$

For clear process, the terms inside the sum in the Eq. A.2 are processed one by one. Let's start with the first term:

$$\mathbb{E}\left[a_i^2\right] = \mathbb{V}\left[a_i\right] + \mathbb{E}\left[a_i\right]^2 = \sigma_i^2 + \mu_i^2 \tag{A.3}$$

Since the $\mathbb{E}\left[\mu_{BN}\right]$ is already a mean value, another mean function will not change its value:

$$\mathbb{E}\left[\mathbb{E}\left[\mu_{BN}\right]^2\right] = \mathbb{E}\left[\mathbb{E}\left[\mu_{BN}\right]\right] \cdot \mathbb{E}\left[\mathbb{E}\left[\mu_{BN}\right]\right] = \mathbb{E}\left[\mu_{BN}\right]^2 \tag{A.4}$$

$$\mathbb{E}\left[-2 \cdot a_i \cdot \mathbb{E}\left[\mu_{BN}\right]\right] = -2 \cdot \mathbb{E}\left[a_i\right] \cdot \mathbb{E}\left[\mathbb{E}\left[\mu_{BN}\right]\right] = -2 \cdot \mu_i \cdot \mathbb{E}\left[\mu_{BN}\right] \tag{A.5}$$

When the equations A.3, A.4 and A.5 are substituted back to A.2, the resulting mean of BN variance is:

$$\begin{aligned}\mathbb{E}\left[\sigma_{BN}^2\right] = \frac{1}{n}\sum_{i=1}^{n}\left(\sigma_i^2 + \mu_i^2 - 2 \cdot \mu_i \cdot \mathbb{E}\left[\mu_{BN}\right] + \mathbb{E}\left[\mu_{BN}\right]^2\right) = \\ = \frac{1}{n}\sum_{i=1}^{n}\sigma_i^2 + \frac{1}{n}\sum_{i=1}^{n}\left(\mu_i - \mathbb{E}\left[\mu_{BN}\right]\right)^2\end{aligned} \tag{A.6}$$

A.2 Variance propagation of BN

Some of the following steps are based on the fact that $\mu_{\bar{a}}$ is a scalar and not a random variable. Hence propagation of $\mathbb{E}$ is allowed.

There are two possible approaches to deriving the propagated variance through the BN layer. The first one is the expected derivation process of evaluating the mean of squared BN activation and then subtracting the squared mean of the BN activation. However, there has to be an approximation made in the last step assuming independence between a_i and μ_{BN}, which does not hold since a_i is used to calculate μ_{BN}.

Therefore another derivation can be applied based on knowing the expected value of the BN mean $\mathbb{E}[\mu_{BN}]$ and the expected value of the BN variance $\mathbb{E}[\sigma_{BN}^2]$ from equations 3.23 and 3.24. Given these two values, the rest of the BN layer can be seen as an affine layer.

$$\begin{aligned}\frac{a_i - \mu_{BN}}{\sqrt{\sigma_{BN}^2 + \varsigma}} \cdot \alpha + \beta \approx \frac{a_i - \mathbb{E}\left[\mu_{BN}\right]}{\sqrt{\mathbb{E}\left[\sigma_{BN}^2\right] + \varsigma}} \cdot \alpha + \beta = \\ = a_i \cdot \frac{\alpha}{\sqrt{\mathbb{E}\left[\sigma_{BN}^2\right] + \varsigma}} + \left(\frac{\mathbb{E}[\mu_{BN}]}{\sqrt{\mathbb{E}\left[\sigma_{BN}^2\right] + \varsigma}} \cdot \alpha + \beta\right) = a_i \cdot \gamma + \delta\end{aligned} \tag{A.7}$$

The equation A.7 can be seen as a simple affine function with the scalar scale γ and scalar offset β. Propagation of the mean and variance for affine layers

has been already derived in the section 3.1.1, more precisely in the equations 3.17 and 3.18. Thus the rest of the evaluation is unambiguous:

$$\begin{aligned}\mathbb{E}\left[\tilde{a}\right] = \mu_i \cdot \gamma + \delta = a_i \cdot \frac{\alpha}{\sqrt{\mathbb{E}\left[\sigma_{BN}^2\right] + \varsigma}} + \left(\frac{\mathbb{E}[\mu_{BN}]}{\sqrt{\mathbb{E}\left[\sigma_{BN}^2\right] + \varsigma}} \cdot \alpha + \beta\right) = \\ = \frac{\mu_i - \mathbb{E}\left[\mu_{BN}\right]}{\sqrt{\mathbb{E}\left[\sigma_{BN}^2\right] + \varsigma}} \cdot \alpha + \beta\end{aligned} \tag{A.8}$$

$$\mathbb{V}\left[\tilde{a}\right] = \gamma^2 \cdot \sigma_i^2 = \frac{\alpha^2}{\mathbb{E}[\sigma_{BN}^2] + \varsigma} \cdot \sigma_i^2 \tag{A.9}$$

Side note: The result of the mentioned classical derivation approach has reached the same results despite the approximated dependence between terms. However, the classical derivation process is significantly more complicated, rendering it sub-optimal. The demonstrated approach to exploiting the attributes of the affine layer can also serve as an example of how can more complicated layers be in the future constructed by exploiting the properties of the already derived elementary layers.

A.3 Mean and Variance of Quantization Layer

A.3.1 Mean

Let's parameterize the grid points of the grid in the following matter:

$$g_i = \alpha \cdot i + g_0 \tag{A.10}$$

This parametrization emphasizes the equidistant spacing between the grid points. If the grid was selected another way, the following derivation process would be inapplicable. The scale of the grid, or distance between grid points, is α. The offset of the grid is the value of the first grid point.

Substitute the reparametrized value of g_i to the the formula for the mean value 3.31:

$$\mu_{\tilde{x}} = \sum_{i=0}^{K-1} (\alpha \cdot i + g_0) \cdot (F(\alpha \cdot i + g_0 + \alpha/2) - F(\alpha \cdot i + g_0 - \alpha/2)) \tag{A.11}$$

Then for the grid points with indexes $i \in [0, 1, \ldots, K-1]$ applies:

$$0: \quad (\alpha \cdot 0 + g_0)\Big(F(\alpha \cdot 0 + g_0 + \alpha/2) - F(\alpha \cdot 0 + g_0 - \alpha/2)\Big) \tag{A.12}$$

$$1: \quad (\alpha \cdot 1 + g_0)\Big(F(\alpha \cdot 1 + g_0 + \alpha/2) - F(\alpha \cdot 1 + g_0 - \alpha/2)\Big) \tag{A.13}$$

$$\vdots$$

$$i: \quad (\alpha \cdot i + g_0)\Big(F(\alpha \cdot i + g_0 + \alpha/2) - F(\alpha \cdot i + g_0 - \alpha/2)\Big) \tag{A.14}$$

$$\vdots$$

$$K-1: \quad (\alpha \cdot (K-1) + g_0)\Big(F(\alpha \cdot (K-1) + g_0 + \alpha/2) - F(\alpha \cdot (K-1) + g_0 - \alpha/2)\Big) \tag{A.15}$$

An equation other than the first one, let's notate it $i+1$, can be simplified into:

$$i+1: \quad (\alpha \cdot i + \alpha + g_0)\Big(F(\alpha \cdot (i+1) + g_0 + \alpha/2) - F(\alpha \cdot i + g_0 + \alpha/2)\Big) \tag{A.16}$$

When the probabilities of two consecutive CDF evaluations are added according to the formula A.11, some terms cancel each other. The result of such addition is notated Δ_i:

$$\begin{aligned}\Delta_i = (\alpha \cdot i + g_0)F(\alpha \cdot i + g_0 - \alpha/2) + \\ + (\alpha(i+1) + g_0)\,F(\alpha(i+1) + g_0 + \alpha/2)) - \alpha \cdot F(\alpha \cdot i + g_0 + \alpha/2)\end{aligned} \tag{A.17}$$

Now it is clear, that for non extreme grid points g_i, the only term that prevails is:

$$-\alpha \cdot F(\alpha \cdot i + g_0 + \alpha/2)$$

Therefore a simplified form of A.11 is evaluated:

$$\begin{aligned}\sum_{i=0}^{K-1} g_i \cdot P(\tilde{x} = g_i) = g_0 \cdot F(g_0 - \alpha/2) - \sum_{i=1}^{K-1} \alpha \cdot F(\alpha \cdot i + g_0 + \alpha/2) + \\ + (\alpha \cdot (K-1) + g_0) \cdot F(\alpha \cdot (K-1) + g_0 + \alpha/2)\end{aligned} \tag{A.18}$$

Following the process from relaxed quantization, the first and last CDF evaluation are approximated so

$$F(g_0 - \alpha/2) = F(-\infty) = 0 \tag{A.19}$$

$$F(\alpha(K-1) + g_0 + \alpha/2) = F(\infty) = 1 \tag{A.20}$$

Then the final formula, which is to be exploited in code, is formed:

$$\begin{aligned} \mu_{\tilde{x}} &= \alpha \cdot (K-1) + g_0 - \alpha \cdot \sum_{i=1}^{K-1} F(\alpha \cdot i + g_0 + \alpha/2) = \\ &= \alpha \cdot (K-1) + g_0 - \alpha \cdot \sum_{i=1}^{K-1} F(g_i + \alpha/2) \end{aligned} \tag{A.21}$$

For symmetrical PDFs applies $F(x) = (1 - F(-x))$. Thus a further simplification can be made:

$$\begin{aligned} \mu_{\tilde{x}} &= \alpha \cdot (K-1) + g_0 + \alpha \cdot \sum_{i=1}^{K-1} \Big(F(-g_i - \alpha/2) - 1\Big) = \\ &= \alpha \cdot (K-1) + g_0 - \alpha \cdot (K-1) + \alpha \sum_{i=1}^{K-1} F(-g_i - \alpha/2) = \\ &= g_0 + \sum_{i=1}^{K-1} F(-g_i - \alpha/2) = g_0 + \sum_{i=1}^{K-1} F(-g_i - \alpha/2) \end{aligned} \tag{A.22}$$

A.3.2 Variance

The derivation process for variance is almost identical. Since the squared mean can be assumed from the previous derivation, only the first term from the formula 3.32 needs to be evaluated. The second term's evaluation for each grid point is:

$$0: \qquad g_0^2\Big(F(\alpha \cdot 0 + g_0 + \alpha/2) - F(\alpha \cdot 0 + g_0 - \alpha/2)\Big) \tag{A.23}$$

$$1: \qquad (\alpha \cdot 1 + g_0)^2\Big(F(\alpha \cdot 1 + g_0 + \alpha/2) - F(\alpha \cdot 1 + g_0 - \alpha/2)\Big) \tag{A.24}$$

$$\vdots$$

$$i: \qquad (\alpha \cdot i + g_0)^2\Big(F(\alpha \cdot i + g_0 + \alpha/2) - F(\alpha \cdot i + g_0 - \alpha/2)\Big) \tag{A.25}$$

$$\vdots$$

$$\begin{aligned} K-1: \qquad (\alpha \cdot (K-1) + g_0)^2\Big(&F(\alpha \cdot (K-1) + g_0 + \alpha/2) - \\ &- F(\alpha \cdot (K-1) + g_0 - \alpha/2)\Big) \end{aligned} \tag{A.26}$$

In a similar matter as for the expected value, the evaluation for the grid point g_{i+1} is:

$$(\alpha^2 \cdot i^2 + 2\alpha \cdot i \cdot g_0 + g_0^2 + \alpha^2 + 2\alpha^2 \cdot i + 2\alpha \cdot g_0)\Big(F(\alpha \cdot i + g_0 + \alpha/2) - \\ -F(\alpha \cdot i + g_0 - \alpha/2)\Big) \tag{A.27}$$

For $\Delta_i = i + (i+1)$ then applies:

$$\begin{aligned} \Delta_i = \quad & (\alpha \cdot i + g_0)^2 F(\alpha \cdot i + g_0 - \alpha/2) + \\ & + (\alpha(i+1) + g_0)^2 F(\alpha(i+1) + g_0 + \alpha/2)) - \\ & - (\alpha^2 + 2\alpha^2 \cdot i + 2\alpha \cdot g_0) F(\alpha \cdot i + g_0 + \alpha/2) \end{aligned} \tag{A.28}$$

For the non extreme grid points g_i, the only term that prevails is:

$$-(\alpha^2 + 2\alpha^2 \cdot i + 2\alpha \cdot g_0) F(\alpha \cdot i + g_0 + \alpha/2) \tag{A.29}$$

Therefore a simplified form of 3.32 is evaluated:

$$\begin{aligned} & \sum_{i=0}^{K-1} g_i^2 \cdot P(\tilde{x} = g_i) = \\ = g_0^2 \cdot F(g_0 - \alpha/2) - & \sum_{i=1}^{K-1} (\alpha^2 + 2\alpha^2 \cdot i + 2\alpha \cdot g_0) F(\alpha \cdot i + g_0 + \alpha/2) + \\ & + (\alpha \cdot (K-1) + g_0)^2 \cdot F(\alpha \cdot (K-1) + g_0 + \alpha/2) \end{aligned} \tag{A.30}$$

After the substitution of values from equations A.19 and A.20, the final form of the second term of 3.32 is:

$$\begin{aligned} & \sum_{i=0}^{K-1} g_i^2 \cdot P(\tilde{x} = g_i) = \\ = (\alpha \cdot (K-1) + g_0)^2 - & \sum_{i=1}^{K-1} (\alpha^2 + 2\alpha^2 \cdot i + 2\alpha \cdot g_0) F(\alpha \cdot i + g_0 + \alpha/2) \end{aligned} \tag{A.31}$$

Then the formula for the propagated variance of the quantization layer is:

$$\sigma_{\tilde{x}}^2 = (\alpha \cdot (K-1) + g_0)^2 - \sum_{i=1}^{K-1} (\alpha^2 + 2\alpha^2 \cdot i + 2\alpha \cdot g_0) F(g_i + \alpha/2) - \mu_{\tilde{x}}^2 \tag{A.32}$$

Appendix B

Experimental Details

B.1 MNIST Datasets and LeNet5 Architecture

The MNIST dataset contains 60k training and 10k test handwritten digit images with 28×28 resolution for classification to 10 classes. The images were not pre-processed in any way.

The Fashon MNIST dataset contains 70k images of individual articles of clothing at 28×28 resolution. The dataset uses 60k images for training and 10k images for testing.

The exploited LeNet5 architecture structure was

$$32\text{C5} - \text{AP2} - 64\text{C5} - \text{AP2} - 1024\text{FC512} - 512\text{FC10} - \text{Softmax}$$

With the notation $X\text{C}Y$ denoting a convolutional layer using $Y \times Y$ kernel filter with X output channels. $Y\text{FC}Z$ denotes a fully connected layer with Y in features and Z out features. $\text{AP}Y$ denotes the standard average pooling with kernel $Y \times Y$ with the stride Y. All layers in the network are followed by a batch normalization layer.

For all methods was exploited learning rate 1e-3 and batch size 128. For loss function was selected the cross entropy. Parameters for the batch normalization layer were preset to [-0.5; -0.5.]

B.2 CIFAR-10 Dataset and ALL-CNN Architecture

The CIFAR-10 dataset contains 50k training and 10k test images of 10 different objects with 32×32 resolution. The images were not pre-processed in any way.

The Fashon MNIST dataset contains 70k images of individual articles of clothing at 28×28 resolution. The dataset uses 60k images for training and 10k images for testing.

The exploited architecture structure was constructed mainly from CNN blocks without bias followed by batch normalization.

→Conv2d(3,96, 3, 1, 1,) → Conv2d(96, 96, 3, 1, 1) → Conv2d(96, 96, 3, 2, 1)) → Conv2d(96, 192, 3, 1, 1) → Conv2d(192, 192, 3, 1, 1)→ Conv2d(192, 192, 3, 2, 1) → Conv2d(192, 192, 3, 1, 1) → Conv2d(192, 192, 1, 1,)→Conv2d(192, 10, 1, 1)→AdaptiveAvgPool2d()→Flatten()→

where the convolutional layers have the following parameter notation: Conv2d(I, J, K, L, M) with I for the number of input channels, J for the number of output channels, K for the convolution kernel size, L for the stride, and M for padding. The learning rate was set to 1e-4 and the batch size to 128.

Appendix C

Bibliography

[AK13] M. Augasta and T. Kathirvalavakumar, *Pruning algorithms of neural networks — a comparative study*, Open Computer Science (2013), 105–115.

[CBM+20] Maurizio Capra, Beatrice Bussolino, Alberto Marchisio, Muhammad Shafique, Guido Masera, and Maurizio Martina, *An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks*, Future Internet **12** (2020), no. 7.

[DNL+17] Yinpeng Dong, Renkun Ni, Jianguo Li, Yurong Chen, Jun Zhu, and Hang Su, *Learning accurate low-bit deep neural networks with stochastic quantization*, 2017.

[GAGN15] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan, *Deep learning with limited numerical precision*, 2015.

[GI22] Shinya Gongyo and Kohta Ishikawa, *Proper straight-through estimator: Breaking symmetry promotes convergence to true minimum*, 2022.

[GSS21] Prateek Garg, Lakshya Singhal, and Ashish Sardana, *[re:] training binary neural networks using the bayesian learning rule*, ML Reproducibility Challenge 2020, 2021.

[HCS+16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio, *Quantized neural networks: Training neural networks with low precision weights and activations*, 2016.

[Hig20] Nick Higham, *"what is stochastic rounding?"*.

[JSS21] Hyeryung Jang, Nicolas Skatchkovsky, and Osvaldo Simeone, *Bisnn: Training spiking neural networks with binary weights via bayesian learning*, 2021 IEEE Data Science and Learning Workshop (DSLW), 2021, pp. 1–6.

[KSW15] Diederik P. Kingma, Tim Salimans, and Max Welling, *Variational dropout and the local reparameterization trick*, 2015.

[LRB+18] Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling, *Relaxed quantization for discretized neural networks*, 2018.

[LS19] Kyle Luther and H. Sebastian Seung, *Sample variance decay in randomly initialized relu networks*, 2019.

[MBK20] Xiangming Meng, Roman Bachmann, and Mohammad Emtiyaz Khan, *Training binary neural networks using the bayesian learning rule*, 2020.

[NK08a] Saralees Nadarajah and Samuel Kotz, *Exact distribution of the max/min of two gaussian random variables*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems **16** (2008), no. 2, 210–212.

[NK08b] ———, *Exact distribution of the max/min of two gaussian random variables*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems **16** (2008), no. 2, 210–212.

[PW18] Jorn W. T. Peters and Max Welling, *Probabilistic binary neural networks*, 2018.

[RORF16] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi, *Xnor-net: Imagenet classification using binary convolutional neural networks*.

[RSFP19] Wolfgang Roth, Günther Schindler, Holger Fröning, and Franz Pernkopf, *Training discrete-valued neural networks with sign activations using weight distributions*, Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part II (Berlin, Heidelberg), Springer-Verlag, 2019, p. 382–398.

[SCH19] Jiahao Su, Milan Cvitkovic, and Furong Huang, *Sampling-free learning of bayesian quantized neural networks*, 2019.

[SF19] Alexander Shekhovtsov and Boris Flach, *Feed-forward propagation in probabilistic neural networks with categorical and max layers*.

[SHYS20] Shuai Sun, Bin Hu, Zhou Yu, and Xiaona Song, *A stochastic max pooling strategy for convolutional neural network trained by noisy samples*, International Journal of Computers Communications & Control **15** (2020).

[SLF17] Oran Shayer, Dan Levi, and Ethan Fetaya, *Learning discrete weights using the local reparameterization trick*, CoRR **abs/1710.07739** (2017).

[XAHK21] Lu Xia, Martijn Anthonissen, Michiel Hochstenbach, and Barry Koren, *A simple and efficient stochastic rounding method for training neural networks in low precision*, 2021.

[YLZ+19] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin, *Understanding straight-through estimator in training activation quantized neural nets*, 2019.

[ZZS22] Jinjie Zhang, Yixuan Zhou, and Rayan Saab, *Post-training quantization for neural networks with provable guarantees*, 2022.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Mráz Martin** Personal ID number: **491862**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Relaxed Quantization and Binarization for Neural Networks

Bachelor’s thesis title in Czech:

Uvolněná kvantizace a binarizace neuronových sítí

Guidelines:

Common neural network models have significant redundancy in their size and precision of weights and representations during their operation. One very promising approach to speed them up is by quantizing all weights and activations to a lower precision, possibly down to 1 bit per weight/activation. Accurate quantization requires specialized training techniques. We explore techniques based on stochastic relaxation [1,2].
1. Develop a software framework, in which different quantization methods can be applied and tested on several standard vision benchmark problems (MNIST, CIFAR) for quantization in the range 1-4 bits. Compare several existing quantization methods and associated gradient estimators, including recent methods [2,3] and new simple alternatives proposed by the advisor.
2. Develop a method to obtain a single deterministic model from a stochastic model, leveraging ideas from [4].
3. Apply the selected quantization methods to realistic problems in robot vision or on-device surveillance, where the data, a reference real-valued network and the performance criterion already exist. Try to achieve the lowest level of quantization without loss of accuracy.

Bibliography / sources:

[1] A. Shekhovtsov, V. Yanush: Reintroducing Straight-Through Estimators as Principled Methods for Stochastic Binary Networks, DAGM (2021)
[2] C. Louizos et al.: Relaxed Quantization for Discretized Neural Networks, ICLR (2019)
[3] M. Paulus et al.: Rao-Blackwellizing the Straight-Through Gumbel-Softmax Gradient Estimator, ICLR (2021)
[4] M. Nagel, eta al.: Up or Down? Adaptive Rounding for Post-Training Quantization, ArXiv preprint (2020)

Name and workplace of bachelor’s thesis supervisor:

Mgr. Oleksandr Shekhovtsov, Ph.D. Visual Recognition Group FEE

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

Date of bachelor’s thesis assignment: **20.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Mgr. Oleksandr Shekhovtsov, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	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