Training Discrete-Valued Neural Networks with Sign Activations Using Weight Distributions

Wolfgang Roth[1(✉)], Günther Schindler[2], Holger Fröning[2], and Franz Pernkopf[1]

[1] Signal Processing and Speech Communication Laboratory,
Graz University of Technology, Graz, Austria
{roth,pernkopf}@tugraz.at
[2] Institute of Computer Engineering, Ruprecht Karls University,
Heidelberg, Germany
{guenther.schindler,holger.froening}@ziti.uni-heidelberg.de

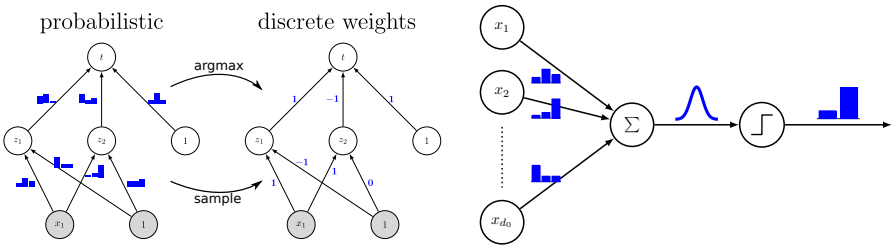
Abstract. Since resource-constrained devices hardly benefit from the trend towards ever-increasing neural network (NN) structures, there is growing interest in designing more hardware-friendly NNs. In this paper, we consider the training of NNs with discrete-valued weights and sign activation functions that can be implemented more efficiently in terms of inference speed, memory requirements, and power consumption. We build on the framework of probabilistic forward propagations using the local reparameterization trick, where instead of training a single set of NN weights we rather train a distribution over these weights. Using this approach, we can perform gradient-based learning by optimizing the continuous distribution parameters over discrete weights while at the same time perform backpropagation through the sign activation. In our experiments, we investigate the influence of the number of weights on the classification performance on several benchmark datasets, and we show that our method achieves state-of-the-art performance.

Keywords: resource-efficiency · deep learning · weight distributions.

1 Introduction

In recent years, deep neural networks (NNs) achieved unprecedented results in many applications such as computer vision [17], speech recognition [10], and machine translation [32], among others. These improved results, however, can be largely attributed to the growing amount of available data and to increasing hardware-capabilities as NNs are essentially known for decades. On the opposite side, there is also a growing number of hardware-constrained embedded devices that barely benefit from this trend in machine learning. Consequently, over the past years an own research field has emerged that is concerned with developing NN architectures that allow for fast and energy-efficient inference and require little memory for the weights.

In this paper, we consider NNs with discrete-valued (ternary, quaternary, quinary) weights and sign activation functions. While such weight representations offer an obvious reduction in memory requirements compared to the commonly used 32-bit floating-point representation, discrete weights and activations



(a) Overview (b) Central Limit Approximation

Fig. 1. (a) Overview of our method. We train distributions over discrete weights (left). After training, a discrete-valued NN can be obtained by selecting the most probable weights or sampling from these distributions. (b) The expectation in Equation (2) is approximated by invoking a central limit approximation at the neurons and propagating the resulting Gaussians through the sign activations. This results in a loss function that is differentiable with respect to the weight probabilities.

can also be exploited to speed up inference. For instance, when using ternary weights $\{-1, 0, 1\}$, we can effectively get rid of multiplications.

However, there is one fundamental problem when learning discrete-valued NNs. Real-valued NNs are commonly trained with gradient-based learning using the backpropagation algorithm which is not feasible for discrete weights and/or piecewise constant activation functions. Most of the research concerned with the training of low-bit NNs can be divided into two categories. (i) Methods that quantize the weights of a pre-trained real-valued NN in a more or less heuristic post-processing step. (ii) Methods that perform "quantization aware" training by employing the so called straight-through gradient estimator [1]. Such methods maintain a set of auxiliary real-valued weights w_r that are quantized during forward propagation using some zero-gradient quantization function to obtain w_q. During backpropagation the gradient of the zero-gradient quantization function is assumed to be non-zero and gradient updates are subsequently applied to the auxiliary weights w_r. Analogously, the same technique can be applied for the sign activation by assuming that its derivative is non-zero. At test time, the real-valued weights w_r are ignored and only the quantized weights w_q are kept. Although these methods achieve impressive results in practice, they are theoretically not well understood. Therefore, it is desired to develop methods that are not based on quantization heuristics.

To this end, we adopt a probabilistic view of NNs by considering a distribution $q(\mathbf{W}|\nu)$ over the discrete weights [31, 28, 23]. We can then introduce an expectation of the zero-gradient loss function over q to obtain a *new* loss function that is differentiable with respect to the distribution parameters ν. This allows us to perform gradient-based learning over the distribution parameters ν. After training has finished, a discrete-valued resource-efficient NN is obtained by either sampling or taking the most probable weights from q, respectively. This is illustrated in Figure 1(a).

Compared to the most relevant previous work in [28, 23] which used binary and ternary weights, we consider general discrete weights. These methods use a parameterization for q that is tailored to binary and ternary weights and does not easily generalize to more than three weights. Similarly, their initialization method for q that is crucial to achieve competitive results does also not easily generalize to more than three weights. We introduce a simpler parameterization and initialization method that generalizes to arbitrary discrete weights. We introduce a variance-aware approximation for max-pooling as opposed to the method in [23] which effectively ignores the variance. In contrast to several other works that require real-valued weights in the input and/or output layers [24, 36, 28], we employ discrete weights in *all* layers. Our method achieves state-of-the-art performance on several benchmark datasets. In our experiments, we show that more weights typically result in better performance. We found a two-stage procedure, where we first train a NN using discrete weights only, and subsequently also train with the sign activation function to mostly improve results compared to training with discrete weights and the sign activation immediately.

The remainder of the paper is structured as follows. Section 2 reviews the most relevant work. In Section 3 we introduce our probabilistic neural network framework. Section 4 presents an efficient approximation to the intractable expected loss function followed by model details in Section 5. We show results of our model in Section 6 before we conclude in Section 7. Code related to the paper is available online at `https://github.com/wroth8/nn-discrete`.

2 Related Work

In our literature review, we focus on work that is most related to this paper, namely work that quantizes weights and/or activations. Most recent works concerned with the training of low-precision NNs rely on the straight-through gradient estimator and introduce different quantizers [5, 11, 37, 24, 19, 2]. Soudry et al. [31] used a Bayesian approach based on expectation propagation to obtain distributions over discrete weights. The work of Shayer et al. [28] is closely related to our work, but they only consider binary and ternary NNs with full-precision ReLU activations. Most related to our work is the work of Peters and Welling [23]. They consider binary and ternary weights using sign activations. We extend the work of [28, 23] to general discrete weights and introduce a variance-aware approximation for max-pooling.

Beyond work focusing on low-bit NNs, there exist several orthogonal directions that facilitate resource-efficient inference. Weight pruning methods, i.e., setting a large portion of the weights to zero, can be utilized to reduce the memory footprint and to improve inference speed [8, 21]. The work in [3, 33, 26] introduces weight sharing to reduce the memory footprint. More global strategies are concerned with special matrix structures that facilitate efficient inference [6, 7, 4]. There also exists work regarding efficient training of neural networks that is not within the scope of this paper [36, 35].

3 Neural Networks and Weight Distributions

A NN with L layers and weights $\mathbf{W} = (\mathbf{W}^1, \ldots, \mathbf{W}^L)$ defines a function $f(\mathbf{x}^0; \mathbf{W})$ by repeatedly computing a linear transformation $\mathbf{a}^l = \mathbf{W}^l \mathbf{x}^{l-1}$ followed by a non-linear activation function $\mathbf{x}^l = \phi^l(\mathbf{a}^l)$. The linear function is either a general matrix-vector multiplication or a convolution operation.[1] For layers $l = 1, \ldots, L-1$, typical choices for the non-linear activation function $\phi^l(a)$ are the ReLU activation $\max(0, a)$, $\tanh(a)$, or, in the context of resource-efficient NNs, $\text{sign}(a) = \mathbb{I}(a \geq 0) - \mathbb{I}(a < 0)$, where $\mathbb{I}$ is the indicator function. In this paper we consider classification problems where the task is to assign an input $\mathbf{x}^0$ to one of C classes. For classification, the activation function ϕ^L at the output is the softmax function $x_i^L = \exp(a_i^L) / \sum_{j=1}^{C} \exp(a_j^L)$. An input $\mathbf{x}^0$ is classified according to $c = \arg\max_j x_j^L$. Note that the computationally expensive softmax does not change the maximum of its inputs, and, therefore, computing the softmax is not required to determine the predicted class label.

Let $\mathcal{D} = \{(\mathbf{x}_1^0, t_1), \ldots, (\mathbf{x}_N^0, t_N)\}$ be a dataset of N input-target pairs and let $y_n = \mathbf{x}_n^L = f(\mathbf{x}_n^0; \mathbf{W})$ be the NN output for the n^{th} sample. The weights $\mathbf{W}$ are typically obtained by performing gradient descent on a loss function

$$\mathcal{L}(\mathbf{W}; \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} l(f(\mathbf{x}_n^0; \mathbf{W}), t_n) + \lambda r(\mathbf{W}), \tag{1}$$

where $l(y_n, t_n)$ penalizes the weights $\mathbf{W}$ if the n^{th} sample is misclassified, r is a regularizer that favors simpler models to enforce generalization, and $\lambda > 0$ is a tunable hyperparameter.

However, when considering weights from a discrete set, gradient-based learning cannot be applied. Moreover, the sign activation function results in a gradient that is zero almost everywhere, rendering backpropagation not usable. In this paper, we employ weight distributions to solve both of these problems at the same time. Instead of a single set of NN weights $\mathbf{W}$, we consider a *distribution* $q(\mathbf{W}|\nu)$ over the discrete weights, where ν are the parameters governing the distribution q. By redefining (1) using an expectation with respect to q, i.e.,

$$\mathcal{L}_{prob}(\nu; \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}_{q(\mathbf{W}|\nu)} \left[l(f(\mathbf{x}_n^0; \mathbf{W}), t_n) \right] + \lambda r(\nu), \tag{2}$$

we obtain a differentiable function with respect to the parameters ν of q. In principle, this allows us to perform gradient-based learning over the distribution parameters ν, and subsequently to determine the discrete-valued weights by either sampling or selecting the most probable weights from q, respectively.[2] However, the expectation in (2) is essentially a sum over exponentially many terms which is generally intractable. In Section 4 we show how the gradient of (2) can be approximated.

[1] A convolution can be cast as a matrix-vector multiplication.

[2] We only consider distributions q where sampling and maximization is easy.

3.1 Discrete Neural Networks

Let $\mathbb{D}^D = \{w_1, \ldots, w_D\}$ be a discrete set of weight values with $w_1 < \ldots < w_D$. In this paper we consider discrete weights with $D \in \{3, 4, 5\}$, i.e., ternary, quaternary, and quinary weights. The choice of the particular weights w_d is arbitrary and we restrict ourselves to equidistributed weights with constant $\delta_w = w_{d+1} - w_d$. In particular, we have $\mathbb{D}^3 = \{-1, 0, 1\}$, $\mathbb{D}^4 = \{-1, -\frac{1}{3}, \frac{1}{3}, 1\}$, and $\mathbb{D}^5 = \{-1, -\frac{1}{2}, 0, \frac{1}{2}, 1\}$. We use the sign activation function which implies that the scale of the discrete weight set becomes irrelevant as either the sign stays unaffected or batch normalization [12] compensates for the change in scale.

For the weight distribution q, we assume independence among the weights which is commonly referred to as the mean-field assumption in the variational inference framework. This implies that q factorizes into a product of factors $q(w|\nu_w)$ for each weight $w \in \mathbf{W}$. Each of these factors is a probability mass function (pmf) over D values. We elaborate more on the parameterization of the pmf over discrete weights in Section 5.2.

3.2 Relation to Variational Inference

The presented work is closely related to the Bayesian variational inference framework. For a Bayesian treatment of NNs, we assume a prior distribution $p(\mathbf{W})$ over the weights and interpret the NN output after the softmax as likelihood $p(\mathcal{D}|\mathbf{W})$ to obtain a posterior $p(\mathbf{W}|\mathcal{D}) \propto p(\mathcal{D}|\mathbf{W})p(\mathbf{W})$ over the weights. As the induced posterior $p(\mathbf{W}|\mathcal{D})$ is generally intractable, the aim of variational inference is to approximate it by a simpler distribution $q(\mathbf{W}|\nu)$ by minimizing the negative evidence lower bound

$$\mathcal{L}_{elbo}(\nu; \mathcal{D}) = \sum_{n=1}^{N} \mathbb{E}_{q(\mathbf{W}|\nu)}[-\log p(t_n|\mathbf{x}_n, \mathbf{W})] + \mathrm{KL}(q(\mathbf{W}|\nu)||p(\mathbf{W})). \quad (3)$$

Equation (3) is proportional to (2) for $l(y_n, t_n)$ being the cross-entropy loss, $r(\nu)$ being the KL-divergence, and $\lambda = 1/N$. The main difference to variational inference is our motivation to use distributions in order to obtain a gradient-based learning scheme for discrete-valued NNs with discrete activation functions. Variational inference is typically used to approximate expectations over the posterior $p(\mathbf{W}|\mathcal{D})$ and to obtain well calibrated uncertainty estimates for NN predictions.

4 Approximation of the Expected Loss

The expected loss in (2) is given by

$$\mathbb{E}_{q(\mathbf{W}|\nu)}\left[l(f(\mathbf{x}_n^0; \mathbf{W}), t_n)\right] = \sum_{\mathbf{W}^1} \cdots \sum_{\mathbf{W}^L} q(\mathbf{W}|\nu)\ l(f(\mathbf{x}_n^0; \mathbf{W}), t_n). \quad (4)$$

Equation (4) contains a sum over exponentially many terms and is generally intractable. Nevertheless, we adopt a practical approximation based on the central limit theorem that has been widely used in the literature [31, 9, 25, 28, 23].

As each neuron computes a sum over many random variables, we can apply the central limit theorem and approximate the neuron distribution by a Gaussian $\mathcal{N}(a_i^1|\mu_{a_i^1}, \sigma^2_{a_i^1})$ where $\mu_{a_i^1} = \sum_j \mathbb{E}[w_{i,j}^1]\ x_j^0$ and $\sigma^2_{a_i^1} = \sum_j \mathbb{V}[w_{i,j}^1]\ (x_j^0)^2$. The binary distribution after the sign function is obtained by $q(x_i^1 = 1) = \Phi(\mu_{a_i^1}/\sigma_{a_i^1})$ where Φ denotes the cumulative distribution function (cdf) of a zero-mean unit-variance Gaussian.[3] This is illustrated in Figure 1(b). This approach transfers the weight distributions $q(\mathbf{W}^1)$ to distributions over the inputs of the next layer $q(\mathbf{x}_n^1)$, i.e.,

$$\mathbb{E}_{q(\mathbf{W}|\nu)}\left[l(f(\mathbf{x}_n^0;\mathbf{W}), t_n)\right] \approx \sum_{\mathbf{W}^2} \cdots \sum_{\mathbf{W}^L} \sum_{\mathbf{x}_n^1} q(\mathbf{W}^{>1}|\nu)\ q(\mathbf{x}_n^1)\ l(f(\mathbf{x}_n^1;\mathbf{W}^{>1}), t_n), \tag{5}$$

where $\mathbf{W}^{>1} = (\mathbf{W}^2, \ldots, \mathbf{W}^L)$. In principle, we can iterate this procedure up to the output layer where it remains to compute the expected loss function $l(y_n, t_n)$ with respect to a Gaussian. However, there are two disadvantages with this approach. (i) For the following layers, the inputs $\mathbf{x}$ are random variables rather than fixed values which requires, assuming independence, to compute $\sigma^2_{a_i^l} = \sum_j \mathbb{V}[w_{i,j}^l]\ \mathbb{E}[x_j^{l-1}]^2 + \mathbb{E}[w_{i,j}^l]^2\ \mathbb{V}[x_j^{l-1}] + \mathbb{V}[w_{i,j}^l]\ \mathbb{V}[x_j^{l-1}]$. This boils down to computing three linear transformations for the variances $\sigma^2_{a_i^1}$ rather than just one as for the first layer which is impractical. (ii) Since $\mathbf{x}^1$ is not observed, the neurons in the next layer $\mathbf{x}^2$ are not independent and, thus, we are effectively introducing an unreasonable independence assumption.

To avoid these problems, we adopt the local reparameterization trick [15, 28, 23]. Since the reparameterized distribution is discrete, we apply the Gumbel softmax approximation [13, 20]. The reparameterization trick transforms the distribution into an observed value that eliminates the before mentioned problems at the cost of introducing a small bias due to the Gumbel softmax approximation. We iterate this scheme up to the output layer where we approximate the expectation of the loss function again by applying the local reparameterization trick at the output activations. Note that due to the zero derivative of the sign activation, the reparameterization trick cannot be applied before the sign activation function. This implies that we have to propagate distributions through max-pooling and batch normalization [12] which is not straightforward and could otherwise be circumvented by simply reparameterizing before these operations.

The question arises whether we can expect the *most probable* discrete weights to perform well if we perform well in expectation? In our probabilistic forward propagation, the loss function essentially only depends on the means $\mathbb{E}_q[w]$ and the variances $\mathbb{V}_q[w]$. Using discrete weights with $d_1 = -1$ and $d_D = 1$, we can represent any mean in the interval $[-1, 1]$. However, we can only achieve low variance if the expectation is close to a weight in $\mathbb{D}$. Therefore, our approach can be seen as a way of parameterizing expectations and constrained variances,

[3] Given finite integer-valued summands, the activation distribution could also be computed exactly in sub-exponential time by convolving the probabilities. However, this would be impractical for gradient-based learning.

respectively. As we require small variances to obtain a small expected loss – in fact a point mass would be optimal – optimization favors expectations that are closer to values in $\mathbb{D}$. Consequently, also the most probable weights in q are expected to perform well.

However, there is one caveat when applying this scheme to *convolutional* layers that was not mentioned in the works of [28, 23]. As weights in our framework are not observed and a single weight in convolutional layers is used in the computation of many activations, these activations actually become dependent. However, when applying the local reparameterization trick, we effectively assume independence among the activations which would be equivalent to sampling different weights for each activation. This could be avoided by sampling the weights directly using the Gumbel softmax approximation at the possible cost of increased variance [15]. Note that this problem does not arise in fully-connected layers as weights are not shared among neurons.

4.1 Approximation of the Maximum Function

Many CNN architectures involve a max-pooling operation where feature maps are downscaled by only passing the maximum of several spatially neighboring activations to the next layer. To this end, we approximate the maximum of two Gaussians by another Gaussian by moment-matching. Let μ_1, μ_2 and σ_1^2, σ_2^2 be the means and the variances of two *independent* Gaussians. Then the mean μ_{max} and the variance σ_{max}^2 of the maximum of these Gaussians is given by [30]

$$\mu_{max} = \mu_1\Phi(\beta) + \mu_2\Phi(-\beta) + \alpha\phi(\beta) \qquad \text{and} \tag{6}$$

$$\sigma_{max}^2 = (\sigma_1^2 + \mu_1^2)\Phi(\beta) + (\sigma_2^2 + \mu_2^2)\Phi(-\beta) + (\mu_1 + \mu_2)\alpha\phi(\beta) - \mu_{max}^2, \tag{7}$$

where ϕ and Φ are the pdf and the cdf of a zero-mean unit-variance Gaussian and

$$\alpha = \sqrt{\sigma_1^2 + \sigma_2^2} \quad \text{and} \quad \beta = \frac{\mu_1 - \mu_2}{\alpha}. \tag{8}$$

This scheme can be iteratively applied to approximate the maximum of several Gaussians. As long as the number of Gaussians is relatively small – CNNs typically involve 2×2 max-pooling – this scheme results in a fairly efficient approximation for max-pooling. In particular, we first approximate the maximum of the two upper and the two lower activations by a Gaussian, respectively, and then we approximate the maximum of these two Gaussians by another Gaussian. This is in contrast to the method proposed in [23] where max-pooling is approximated by selecting the mean and the variance of the activation whose mean is maximal, which effectively ignores the variance in the process.

5 Model Details

A basic convolutional block is depicted in Figure 2. We typically start with dropout, followed by a convolution layer. Motivated by [24], we perform the

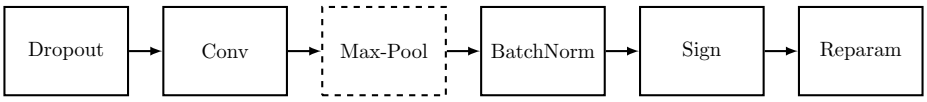


Fig. 2. The convolutional block used in this paper. Max-pooling is not always present.

pooling operation (if present) right after the convolutional layer to avoid information loss. Afterwards we perform batch normalization as described in Section 5.1, followed by computing the pmf after the sign function, and finally performing the local reparameterization trick using the Gumbel softmax approximation.

We do not perform batch normalization in the final layer. Instead we introduce a real-valued bias and divide the output activations by the square root of the number of incoming neurons. This normalization step is crucial for training as otherwise the output softmax would be saturated in most cases as the output activations are typically large due to the discreteness of the weights and inputs from the previous layer, respectively. Moreover, we found dropout in our experiments to be particularly helpful as it improved performance considerably. Dropout was performed by randomly setting both the neuron's mean and its variance to zero in order to completely remove its influence.

5.1 Batch Normalization

As briefly mentioned in Section 4, we are required to generalize batch normalization to distributions. Batch normalization is particularly important when using sign activations to avoid excessive information loss [24]. We use the method proposed in [23] to normalize distributions to approximately having zero-mean and unit-variance. The mini-batch statistics for N_B samples are computed as

$$\mu_{i,bn} = \frac{1}{N_B}\sum_{n=1}^{N_B} \mu_{a_{n,i}} \quad \text{and} \quad \sigma^2_{i,bn} = \frac{1}{N_B - 1}\sum_{n=1}^{N_B} \sigma^2_{a_{n,i}} + (\mu_{a_{n,i}} - \mu_{i,bn})^2. \quad (9)$$

Subsequently, batch normalization is computed as

$$\mu_{a_i} \leftarrow \frac{\mu_{a_i} - \mu_{i,bn}}{\sigma_{i,bn}}\gamma_i + \beta_i \quad \text{and} \quad \sigma^2_{a_i} \leftarrow \frac{\sigma^2_{a_i}}{\sigma^2_{i,bn}}\gamma_i^2, \quad (10)$$

where β_i and γ_i are the learnable batch normalization parameters. For predictions, it is important to compute the batch statistics *using the discrete NN* as the batch statistics computed during training might differ significantly. Using batch statistics computed during training resulted in severe fluctuations in the validation errors. However, this implies that estimating the training set statistics using exponential moving averages[4] *during* training, as is commonly done in practice, is not applicable anymore, and we have to compute a separate forward pass using the discrete NN to obtain these statistics. We estimate the training

[4] $\mu^{new}_{i,tr} \leftarrow \xi_{bn}\mu_{i,bn} + (1 - \xi_{bn})\mu^{old}_{i,tr}$ for $\xi_{bn} \in (0, 1)$, and similarly for $\sigma^2_{i,tr}$.

set statistics using an exponential moving average over the whole training set *after* each epoch and right before computing the validation error. Note that batch normalization, although introducing real-valued variables, requires only a marginal computational overhead at test time [34].

5.2 Parameterization and Initialization of q

Shayer et al. [28] introduced a parameterization for ternary distributions based on two probabilities, $q(w = 0)$ and $q(w = 1|w \neq 0)$, which is not easily generalizable to distributions over more than three weights. In this work, we parameterize distributions over D values using unconstrained unnormalized log-probabilities (logits) ν_w^d for $d \in \{1, \ldots, D\}$. The normalized probabilities $q(w|\nu_w)$ can be recovered by applying the softmax function to the logits ν_w. This straightforward parameterization allows to select the d^{th} weight by setting $\nu_w^d > \nu_w^{d'}$ for $d' \neq d$. Due to the sum-to-one constraint of probabilities, we can reduce the number of parameters in ν by fixing an arbitrary logit, e.g., $\nu_w^1 = 0$. However, we refrain to do so as it is more natural to increase a probability explicitly by increasing its corresponding logit rather than indirectly by reducing all other logits.

Moreover, Shayer et al. [28] introduced an initialization method for the distribution parameters ν by matching the expectation $\mathbb{E}_q[w]$ to the real weights $\tilde{w}$ of a pre-trained network. In our experiments, we found such an initialization scheme to be crucial as starting from randomly initialized logits one usually gets stuck in a bad local minimum. However, their initialization method also does not generalize easily to more than three weights, especially since matching the expectation $\mathbb{E}_q[w] = \tilde{w}$ is already an underconstrained problem for $D = 3$. Hence, we propose to use the following initialization scheme to approximately match the expectations which we found to be at least as effective as Shayer et al.'s approach for ternary weights. Let $w_1 < \ldots < w_D$ be the set of discrete weights. Furthermore, let q_{min} be a minimal probability that is required to avoid zero probabilities. The maximum probability is then given by $q_{max} = 1-(D-1)q_{min}$ and we define $\delta_q = q_{max} - q_{min}$. Given a real-valued weight $\tilde{w}$, we initialize q as

$$q(w = w_j) = \begin{cases} q_{min} + \delta_q \ \frac{\tilde{w} - w_{j-1}}{w_j - w_{j-1}} & w_{j-1} < \tilde{w} \leq w_j \\ q_{min} + \delta_q \ \frac{w_{j+1} - \tilde{w}}{w_{j+1} - w_j} & w_j < \tilde{w} \leq w_{j+1} \\ q_{max} & (j = 1 \wedge \tilde{w} < w_1) \vee (j = D \wedge \tilde{w} > w_D) \\ q_{min} & \text{otherwise.} \end{cases} \tag{11}$$

This scheme is illustrated in Figure 3(a). However, weight magnitudes might differ across layers which Shayer et al. [28] addressed by dividing the weights in each layer by their standard deviation before applying (11). We propose the following scheme which distributes probabilities more uniformly across the discrete weights in order to benefit from the increased expressiveness when using a larger D. Let $\Phi_e^l(w) = 1/|\mathbf{W}^l| \sum_{\tilde{w} \in \mathbf{W}^l} \mathbb{I}[\tilde{w} \leq w]$ be the empirical cdf of the weights in layer l. We compute $\tilde{w}^l \leftarrow \Phi_e^l(\tilde{w}^l)$ such that the weights cover the unit interval

with equal spacing while keeping the relative order of the weights, essentially removing the scale. Then we shift and scale the weights $\tilde{w}$ to cover the interval $[w_1 - \delta_w/2, w_D + \delta_w/2]$, followed by assigning the probabilities to q according to (11). This ensures that each discrete weight is initially selected equally often as the most likely weight in q. We propose to use this scheme for the positive and the negative weights separately such that the signs of the weights are preserved.

6 Experiments

We performed classification experiments on several datasets that are described in Section 6.1. We optimized (2) using ADAM [14], and we report the test classification error of the epoch resulting in the best validation classification error. All results for discrete-valued NNs are reported using the most probable model from q. We selected $l(y_n, t_n)$ to be the cross-entropy loss, $r(\nu)$ to be the squared ℓ^2-norm over the logits [28], and $\lambda = 10^{-10}$. Penalizing large logits can be seen as enforcing a uniform pmf and therefore increasing entropy and variance. As stated in [28], this rather helps to obtain better Gaussian approximations using the central limit theorem rather than to reduce overfitting. After each gradient update we clip the logits to the range $[-5, 5]$. We set the initial step size to 10^{-2} for the logits and to 10^{-3} for all other parameters (batch normalization, bias in the final layer). We use the following plateau learning rate reduction scheme: The learning rate is kept for at least τ_1 epochs and after τ_1 epochs we multiply the learning rate by $1/2$ if the validation error has not improved within the last τ_2 epochs. The parameters τ_1 and τ_2, as well as some other dataset-depending settings can be found in Section 6.1. We selected $q_{max} = 0.95$, the Gumbel softmax temperature $\tau_g = 1$, and $\xi_{bn} = 0.1$.

6.1 Datasets

MNIST The MNIST dataset [18] contains grayscale images of size 28×28 pixels showing handwritten digits from 0-9. The training set contains 60,000 images and the test set contains 10,000 images. We split the training set into a training set of 50,000 training images and 10,000 validation images. We normalize the pixels to be in the range $[-1, 1]$. We considered two scenarios for the MNIST dataset: (i) A permutation-invariant (PI) setting where each pixel is treated as independent feature without taking pixel locality into account, i.e., we do not use a CNN. For this setting we use the fully-connected architecture

$$\text{FC1200} - \text{FC1200} - \text{FC10},$$

where FC1200 denotes a fully-connected layer with 1200 output neurons. We refer to this setting as MNIST (PI). (ii) We keep the image structure and use CNNs with the architecture

$$\text{32C5} - \text{P2} - \text{64C5} - \text{P2} - \text{FC512} - \text{FC10},$$

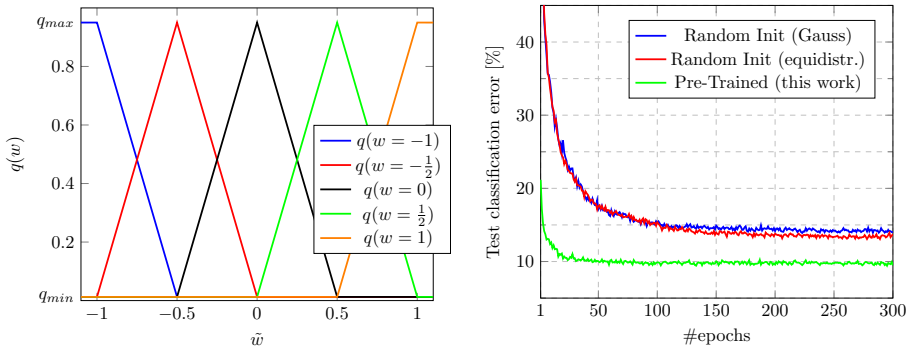


Fig. 3. (a) Our initialization method for quinary weight distributions based on pre-trained real-valued weights $\tilde{w}$. (b) Test classification error [%] over number of epochs on Cifar-10 for ternary weights using different initialization methods for q. For randomly initialized probabilities, we either sampled real-valued weights $\tilde{w} \sim \mathcal{N}(0,1)$ (Gauss) or randomly assigned equidistributed values in the interval $[-1.5, 1.5]$ randomly to the weights $\tilde{w}$ (equidistr.) before applying the method described in (a).

where 32C5 means that 5×5 filter kernels are applied and 32 output feature maps are generated, and P2 means that 2×2 max-pooling is applied. We trained both architectures for 500 epochs using mini-batches of 100 samples with $\tau_1 = 50$ and $\tau_2 = 10$. We used dropout probabilities (0.1, 0.2, 0.3) for MNIST (PI) and (0, 0.2, 0.3, 0) for the CNN setting, respectively, where the first entry corresponds to the input layer and the following entries correspond to the subsequent layers.

Cifar-10 and Cifar-100 The Cifar-10 dataset [16] contains 32×32 pixel RGB images showing objects from ten different categories. The dataset is split into 50,000 training images and 10,000 test images. We split the training set into 45,000 training images and 5,000 validation images. The pixels are again normalized to be in the range $[-1, 1]$. For training, we perform data augmentation by shifting the images randomly by up to four pixels in each direction, and we randomly flip images along the vertical axis similar as in [28]. Cifar-100 is similar to Cifar-10 except that the task is to assign an image to one of 100 object categories. As the image sizes and the training and test set sizes are equal, we perform the same preprocessing steps as described above. For both datasets, we use the VGG-inspired [29] CNN architecture

$$2\times128\text{C3} - \text{P2} - 2\times256\text{C3} - \text{P2} - 2\times512\text{C3} - \text{P2} - \text{FC1024} - \text{FC10/100},$$

where 2×128C3 denotes two consecutive 128C3 blocks. We trained for 300 epochs using mini-batches of 100 samples with $\tau_1 = 30$ and $\tau_2 = 10$. We used dropout probabilities (0, 0.2, 0.2, 0.3, 0.3, 0.3, 0.4, 0) for both datasets.

Table 1. Classification errors [%] of various NN models. Real+Tanh is the baseline that was used to initialize the discrete NNs. For discrete NNs, we conducted each experiment five times and report the means and standard deviations, respectively.

Dataset	Real+Tanh	Ternary+Sign	Quaternary+Sign	Quinary+Sign
MNIST (PI)	1.030	1.350 ± 0.058	1.326 ± 0.012	1.334 ± 0.027
MNIST	0.560	0.712 ± 0.040	0.652 ± 0.020	0.654 ± 0.040
Cifar-10	7.620	9.508 ± 0.289	9.494 ± 0.210	9.078 ± 0.218
Cifar-100	30.150	33.550 ± 0.161	33.534 ± 0.400	33.026 ± 0.231
SVHN	2.259	2.618 ± 0.047	2.631 ± 0.051	2.605 ± 0.045

SVHN The SVHN dataset [22] contains 32×32 pixel RGB images showing parts of pictures containing house numbers that need to be classified to the digits 0-9. The dataset is split into 604,388 training images and 26,032 test images. We follow the procedure of [27] to split the training set into 598,388 training images and 6,000 validation images. Once again, we normalize pixels to be in the range $[-1, 1]$. Since the dataset is quite large, we do not perform data augmentation. We use the same CNN architecture as for the Cifar datasets except that we only use half the number of feature maps in the convolutional layers, i.e.,

$$2\times64\text{C3} - \text{P2} - 2\times128\text{C3} - \text{P2} - 2\times256\text{C3} - \text{P2} - \text{FC1024} - \text{FC10}.$$

Since SVHN is quite large, we performed only 100 epochs of training using mini-batches of 250 samples with $\tau_1 = 15$ and $\tau_2 = 5$. We used the same dropout probabilities as for the Cifar datasets.

6.2 Classification Results

In the first experiment, we used pre-trained real-valued NNs with tanh activation to initialize the discrete NNs with sign activation function as shown in Section 5.2. The results are shown in Table 1. The performance gap compared to the real-valued NN tends to become smaller as more weights are used. There is a consistent improvement of quinary weights over ternary weights. Quaternary weights improve on four datasets compared to ternary weights whereas they achieve worse performance than quinary weights on the more challenging Cifar and SVHN datasets. We attribute the mixed behavior of quaternary weights to the missing zero-weight that might be important.

In the next experiment, we performed an intermediate step where we first only discretized the weights and kept the tanh activation. In a next step, we used this NN as initial model to train a NN with discrete weights *and* sign activation. The results of these experiments are shown in Table 2. When only the weights are discretized and tanh is kept, the performance gap compared to real-valued NNs in Table 1 is less severe than when discretizing both the weights and the activations. The only exception is on MNIST (PI) where the gap is similar to the gap when in addition the sign activation is used. Interestingly, the performance on Cifar-100 improves compared to real-valued NNs, indicating a regularizing

Table 2. Classification errors [%] of various NN models. The first three models were initialized with Real+Tanh from Table 1. The last three models were initialized with the corresponding discrete-weight model with tanh activation. For discrete NNs with sign activation, we conducted each experiment five times and report the means and standard deviations, respectively.

Dataset	Ternary+Tanh	Quaternary+Tanh	Quinary+Tanh	Ternary+Sign	Quaternary+Sign	Quinary+Sign
MNIST (PI)	1.310	1.300	1.280	1.352 ± 0.053	1.292 ± 0.031	1.298 ± 0.040
MNIST	0.570	0.560	0.620	0.736 ± 0.037	0.678 ± 0.042	0.736 ± 0.039
Cifar-10	7.770	7.810	8.030	9.174 ± 0.139	9.246 ± 0.251	9.080 ± 0.246
Cifar-100	29.770	29.840	29.120	33.608 ± 0.199	33.236 ± 0.265	32.910 ± 0.196
SVHN	2.328	2.324	2.362	2.574 ± 0.086	2.591 ± 0.081	2.532 ± 0.056

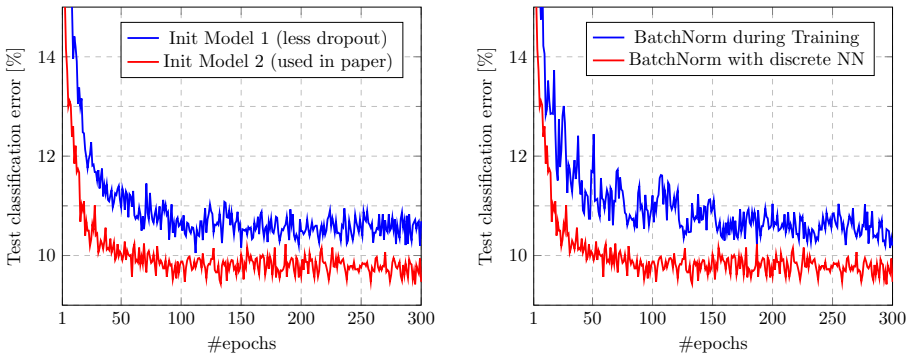


Fig. 4. (a) Test classification error [%] on Cifar-10 obtained by using two different real-valued NNs for the initial parameters of q. Init Model 1 uses less dropout and achieves 7.05% test error whereas Init Model 2 achieves 7.62% test error. However, Init Model 2 results in a better discrete-valued NN. (b) Test classification error [%] on Cifar-10 by estimating the training set batch statistics using an exponential moving average once during training and once using the discrete-valued NN.

effect similar as in [37]. These findings are in line with other papers that have shown little performance degradation when the real-valued activation function is kept and only the weights are discretized [37, 28].

Next, we compare the corresponding values of discrete-valued NNs with sign activation from Table 1 and 2. Except on MNIST where results do not improve, the performance using a pre-trained discrete-valued NN with tanh activation for initialization improves in nine out of twelve cases on the other datasets, showing that a two stage training procedure is mostly beneficial.

We also compare our model with [11, 24, 36, 23] as their quantization is similar to ours. Hubara et al. [11] use binary weights and sign activations, albeit using larger architectures. They report two results and achieve on average 1.18±0.22% on MNIST (PI), 10.775±0.625% on Cifar-10, and 2.66±0.135 on SVHN. XNOR-

Net [24] uses real-valued *data-dependent* scale factors to perform a binary convolution. Using the same structure as [11], they achieve 10.17% on Cifar-10. DoReFa-Net [36] achieves 2.9% on SVHN using binary weights and binary 0-1 activations. The work in [23] is closest to ours and achieves 0.74% on MNIST and 10.30% on Cifar-10 using ternary weights and sign activations.

6.3 Ablation Study

In this Section, we investigate the influence of the initialization of q, dropout, and batch normalization on Cifar-10. Figure 3(b) compares our initialization method for q described in Section 5.2 to random initialization strategies. Our method converges faster than the random strategies and achieves almost 4% less absolute classification error than the random strategies which seem to get stuck in bad local minima. This highlights the importance of a proper initialization strategy for the training of weight distributions as the loss surface being optimized seems to be substantially more delicate than that of a conventional real-valued NN.

This raises the question if it might pay off to put more effort into the training of the real-valued NN serving for initialization. To answer this question, we optimized several dropout rates for the initial real-valued NN with tanh activation, keeping all the other hyperparameters the same. This resulted in dropout rates (0, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1. 0.1) achieving a test error of 7.05% – an almost 0.6% absolute improvement compared to the result in Table 1. However, when using this model to initialize q, we achieved inferior performance for the discrete-valued NN with sign activation as can be seen in Figure 4(a).

As mentioned in Section 5.1, computing exponential moving averages *during* training to estimate the training set statistics required at test time could lead to severely different statistics as those obtained using the discrete-valued NN. To verify this, we performed two runs that only differ in the estimation of the training set statistics. This is shown in Figure 4(b). The performance deteriorates heavily and especially in the beginning there are substantial fluctuations.

7 Conclusion

In this paper, we have generalized previous work on discrete weight distributions to arbitrary discrete weights. To this end, we introduced simpler schemes for weight parameterization and weight initialization, respectively. We introduced a Gaussian approximation for max-pooling that takes the variance into account. Our method achieves state-of-the-art performance on several image classification datasets using discrete weights in *all* layers. We found weight initialization using a pre-trained real-valued NN crucial in order to obtain reasonable performances. However, it remains unclear what properties of a pre-trained NN make it a good choice for an initial model since we observed that a better performing real-valued NN does not necessarily result in a better performing discrete-valued NN.

Acknowledgements. This work was supported by the Austrian Science Fund (FWF) under the project number I2706-N31.

References

1. Bengio, Y., Léonard, N., Courville, A.C.: Estimating or propagating gradients through stochastic neurons for conditional computation. CoRR **abs/1308.3432** (2013)
2. Cai, Z., He, X., Sun, J., Vasconcelos, N.: Deep learning with low precision by half-wave Gaussian quantization. In: Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5406–5414 (2017)
3. Chen, W., Wilson, J.T., Tyree, S., Weinberger, K.Q., Chen, Y.: Compressing neural networks with the hashing trick. In: International Conference on Machine Learning (ICML). pp. 2285–2294 (2015)
4. Cheng, Y., Yu, F.X., Feris, R.S., Kumar, S., Choudhary, A.N., Chang, S.: An exploration of parameter redundancy in deep networks with circulant projections. In: International Conference on Computer Vision (ICCV). pp. 2857–2865 (2015)
5. Courbariaux, M., Bengio, Y., David, J.P.: BinaryConnect: Training deep neural networks with binary weights during propagations. In: Advances in Neural Information Processing Systems (NIPS). pp. 3123–3131 (2015)
6. Denil, M., Shakibi, B., Dinh, L., Ranzato, M., de Freitas, N.: Predicting parameters in deep learning. In: Advances in Neural Information Processing Systems (NIPS). pp. 2148–2156 (2013)
7. Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: Neural Information Processing Systems. pp. 1269–1277 (2014)
8. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. In: International Conference on Learning Representations (ICLR) (2016)
9. Hernandez-Lobato, J.M., Adams, R.: Probabilistic backpropagation for scalable learning of Bayesian neural networks. In: International Conference on Machine Learning (ICML). pp. 1861–1869 (2015)
10. Hinton, G., Deng, L., Yu, D., Dahl, G.E., r. Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., Kingsbury, B.: Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine **29**(6), 82–97 (2012)
11. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: Advances in Neural Information Processing Systems (NIPS). pp. 4107–4115 (2016)
12. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning (ICML). pp. 448–456 (2015)
13. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with Gumbel-softmax. In: International Conference on Learning Representations (ICLR) (2017)
14. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (ICLR) (2015), arXiv: 1412.6980
15. Kingma, D.P., Salimans, T., Welling, M.: Variational dropout and the local reparameterization trick. In: Advances in Neural Information Processing Systems (NIPS). pp. 2575–2583 (2015)
16. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009)
17. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems (NIPS). pp. 1106–1114 (2012)

18. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
19. Lin, X., Zhao, C., Pan, W.: Towards accurate binary convolutional neural network. In: Neural Information Processing Systems. pp. 344–352 (2017)
20. Maddison, C.J., Mnih, A., Teh, Y.W.: The concrete distribution: A continuous relaxation of discrete random variables. In: International Conference on Learning Representations (ICLR) (2017)
21. Molchanov, D., Ashukha, A., Vetrov, D.P.: Variational dropout sparsifies deep neural networks. In: International Conference on Machine Learning (ICML). pp. 2498–2507 (2017)
22. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Y Ng, A.: Reading digits in natural images with unsupervised feature learning. In: Deep Learning and Unsupervised Feature Learning Workshop @ NIPS (2011)
23. Peters, J.W.T., Welling, M.: Probabilistic binary neural networks. CoRR **abs/1809.03368** (2018)
24. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: Imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision (ECCV). pp. 525–542 (2016)
25. Roth, W., Pernkopf, F.: Variational inference in neural networks using an approximate closed-form objective. In: Bayesian Deep Learning Workshop @ NIPS (2016)
26. Roth, W., Pernkopf, F.: Bayesian neural networks with weight sharing using Dirichlet processes. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) accepted (2018)
27. Sermanet, P., Chintala, S., LeCun, Y.: Convolutional neural networks applied to house numbers digit classification. In: International Conference on Pattern Recognition (ICPR). pp. 3288–3291 (2012)
28. Shayer, O., Levi, D., Fetaya, E.: Learning discrete weights using the local reparameterization trick. In: International Conference on Learning Representations (ICLR) (2018)
29. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (ICLR) (2015)
30. Sinha, D., Zhou, H., Shenoy, N.V.: Advances in computation of the maximum of a set of Gaussian random variables. IEEE Trans. on CAD of Integrated Circuits and Systems **26**(8), 1522–1533 (2007)
31. Soudry, D., Hubara, I., Meir, R.: Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In: Advances in Neural Information Processing Systems (NIPS). pp. 963–971 (2014)
32. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems (NIPS). pp. 3104–3112 (2014)
33. Ullrich, K., Meeds, E., Welling, M.: Soft weight-sharing for neural network compression. In: International Conference on Learning Representations (ICLR) (2017)
34. Umuroglu, Y., Fraser, N.J., Gambardella, G., Blott, M., Leong, P.H.W., Jahre, M., Vissers, K.A.: FINN: A framework for fast, scalable binarized neural network inference. In: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (ISFPGA). pp. 65–74 (2017)
35. Wu, S., Li, G., Chen, F., Shi, L.: Training and inference with integers in deep neural networks. In: International Conference on Learning Representations (ICLR) (2018)

36. Zhou, S., Ni, Z., Zhou, X., Wen, H., Wu, Y., Zou, Y.: DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. CoRR **abs/1606.06160** (2016)
37. Zhu, C., Han, S., Mao, H., Dally, W.J.: Trained ternary quantization. In: International Conference on Learning Representations (ICLR) (2017)