PROBABILISTIC BINARY NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Low bit-width weights and activations are an effective way of combating the increasing need for both memory and compute power of Deep Neural Networks. In this work, we present a probabilistic training method for Neural Network with both binary weights and activations, called PBNet. By embracing stochasticity during training, we circumvent the need to approximate the gradient of functions for which the derivative is zero almost always, such as $\text{sign}(\cdot)$, while still obtaining a fully Binary Neural Network at test time. Moreover, it allows for anytime ensemble predictions for improved performance and uncertainty estimates by sampling from the weight distribution. Since all operations in a layer of the PBNet operate on random variables, we introduce stochastic versions of Batch Normalization and max pooling, which transfer well to a deterministic network at test time. We evaluate two related training methods for the PBNet: one in which activation distributions are propagated throughout the network, and one in which binary activations are sampled in each layer. Our experiments indicate that sampling the binary activations is an important element for stochastic training of binary Neural Networks.

1 INTRODUCTION

Deep Neural Networks are notorious for having vast memory and computation requirements, both during training and test/prediction time. As such, Deep Neural Networks may be unfeasible in various environments such as battery powered devices, embedded devices (because of memory requirement), on body devices (due to heat dissipation), or environments in which constrains may be imposed by a limited economical budget. Hence, there is a clear need for Neural Networks that can operate in these resource limited environments.

One method for reducing the memory and computational requirements for Neural Networks is to reduce the bit-width of the parameters and activations of the Neural Network. This can be achieved either during training (e.g., Ullrich et al. (2017); Achterhold et al. (2018)) or using post-training mechanisms (e.g., Louizos et al. (2017), Han et al. (2015)). By taking the reduction of the bit-width for weights and activations to the extreme, i.e., a single bit, one obtains a Binary Neural Network. Binary Neural Networks have several advantageous properties, i.e., a $32\times$ reduction in memory requirements and the forward pass can be implemented using XNOR operations and bit-counting, which results in a $58\times$ speedup on CPU (Rastegari et al., 2016). Moreover, Binary Neural Networks are more robust to adversarial examples (Galloway et al., 2018).

Shayer et al. (2018) introduced a probabilistic training method for Neural Networks with binary weights, but allow for full precision activations. In this paper, we propose a probabilistic training method for Neural Networks with both binary weights and binary activations, which are even more memory and computation efficient. In short, we train a stochastic Binary Neural Network by leveraging the local reparametrization trick (Kingma et al., 2015). Subsequently, we propagate the obtained (pre-)activation distribution throughout the network, which allows us to back-propagate through function like sign. We call networks trained using this method Probabilistic Binary Neural Networks (PBNet). We introduce two variants: one in which the activation distributions are propagated throughout the Neural Network, and one in which the binary activation is sampled in each layer using the Concrete distribution (Maddison et al., 2016; Jang et al., 2016). At test time, we obtain a single deterministic Binary Neural Network, an ensemble of Binary Neural Networks by sampling from the parameter distribution, or a Ternary Neural Network based on the Binary weight distribution. An advantage of our method is that we can take samples from the parameter distribution

indefinitely—without retraining. Hence, this method allows for anytime ensemble predictions and uncertainty estimates. Note that while in this work we only consider the binary case, our method supports any discrete distribution over weights and activations.

In the proposed method, binary activations are either sampled as the very last operation in each layer, or propagated throughout the network. As such, any other operation that is normally applied to the pre-activation must be applied to random variables. One of the contributions of this paper is the definition of batch-normalization and max-pooling for random variables. Our experiments show that these operations transfer well to a non-stochastic operation in a deterministic network—after re-estimation of the batch norm statistics.

2 Probabilistic Binary Neural Network

We introduce a training method for binary Neural Networks that have both binary weights and activations that allows for optimization using gradient descent. We call a nework trained using this method a Probabilistic Binary Neural Network (PBNet). In order to deal with the inherit discrete nature of a binary neural network, we make use of the variational optimization framework (Staines & Barber, 2012). Specifically, we pose a binary distribution $q_{\boldsymbol{\theta}}(\mathbf{B})$ over the binary weights $\mathbf{B}$ of the neural network in order to obtain an upper bound on the training objective $\mathcal{L}(\cdot)$:

$$\min_{\mathbf{B}} \mathcal{L}(\mathbf{B}) \leq \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{B})}[\mathcal{L}(\mathbf{B})] \tag{1}$$

This allows us to use a stochastic neural network at training time, and sample a deterministic binary neural network at test time. Moreover, in order to deal with the binarization of activations, we propagate (pre-)activation distributions through the network such that the activation (or binarization) function is performed on random variables. This allows for backpropagation through functions that would normally zero out gradients (e.g., sign$(\cdot)$).

The binary weights are modelled using a reparametrization of a scaled and translated Bernoulli distribution, which we refer to as the Binary distribution. For $\theta \in [-1, 1]$, it is given by:

$$a \sim \text{Binary}(\theta) \iff \frac{a+1}{2} \sim \text{Bernoulli}(\frac{\theta+1}{2}). \tag{2}$$

The support of the Binary distribution is $\{-1, +1\}$, such that $p(a = +1) = \frac{1}{2}(\theta + 1)$ and $p(a = -1) = \frac{1}{2}(1 - \theta)$. The mean $\mathbb{E}[a] = \theta$ and variance $\mathbb{V}[a] = 1 - \theta^2$ are easily computed from θ. Moreover, let $b \sim \text{Binary}(\phi)$, then $ab \sim \text{Binary}(\theta\phi)$. Hence, given weights $\mathbf{w} \sim \text{Binary}(\boldsymbol{\theta})$ and activations $\mathbf{h} \sim \text{Binary}(\boldsymbol{\phi})$, the inner-product between the weights and activations is distributed according to a translated and scaled Poisson binomial distribution:

$$\frac{\mathbf{w} \cdot \mathbf{h} + D}{2} \sim \text{PoiBin}(2[\boldsymbol{\theta} \odot \boldsymbol{\phi}] - 1). \tag{3}$$

Where D is the dimensionality of $\mathbf{h}$ and $\mathbf{w}$ and $\odot$ denotes element-wise multiplication. See the *picket fence* on the top in Figure 1 for an illustration of the PMF of a Poisson binomial distribution. Although the scaled and translated Poisson binomial

Algorithm 1: Pseudo code for forward pass of single layer in PBNet(-S). $\mathbf{a}_{l-1}$ denotes the activation of the previous layer, $\mathbf{B}$ the random binary weight matrix, τ is the temperature used for the concrete distribution, $f(\cdot,\cdot)$ the linear transformation used in the layer, $\epsilon > 0$ a small constant for numerical stability, D the dimensionality of the inner product in f, and γ & β are the parameters for batch normalization.

Input: $\mathbf{a}_{l-1}$, $\mathbf{B} \sim p(\mathbf{B})$, τ, $f(\cdot,\cdot)$, ϵ, γ, β
Result: Binary activation $\mathbf{a}_l$

```
// CLT approximation
```
if $\mathbf{a}_{l-1}$ *is a binary random variable* **then**
  $\boldsymbol{\mu} = f(\mathbb{E}[\mathbf{B}], \mathbb{E}[\mathbf{a}_{l-1}])$;
  $\boldsymbol{\sigma}^2 = D - f((\mathbb{E}[\mathbf{B}])^2, (\mathbb{E}[\mathbf{a}_{l-1}])^2)$;
else
  $\boldsymbol{\mu} = f(\mathbb{E}[\mathbf{B}], \mathbf{a}_{l-1})$;
  $\boldsymbol{\sigma}^2 = f(\mathbb{V}[\mathbf{B}], \mathbf{a}_{l-1}^2)$;
end

```
// Batch normalization
```
$\mathbf{m} = \text{channel-wise-mean}(\boldsymbol{\mu})$;
$\mathbf{v} = \text{channel-wise-variance}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2, \mathbf{m})$;
$\boldsymbol{\mu} = \gamma(\boldsymbol{\mu} - \mathbf{m})/\sqrt{\mathbf{v} + \epsilon} + \beta$;
$\boldsymbol{\sigma}^2 = \gamma^2\boldsymbol{\sigma}^2/(\mathbf{v} + \epsilon)$;

```
// Max pooling
```
if *max pooling required* **then**
  $\mathbf{n} \sim \mathcal{N}(0, \mathbf{I})$;
  $\mathbf{s} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \mathbf{n}$;
  $\boldsymbol{\iota} = \text{max-pooling-indices}(\mathbf{s})$;
  $\boldsymbol{\mu}, \boldsymbol{\sigma}^2 = \text{select-at-indices}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2, \boldsymbol{\iota})$;
end

```
// Binarization and sampling
```
$\mathbf{p} = 1 - \Phi(0|\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$;
if *sample activation* **then**
  $\mathbf{a}_l \sim \text{BinaryConcrete}(\mathbf{p}, \tau)$;
  return $\mathbf{a}_l$;
else
  return $\text{Binary}(\mathbf{p})$
end

distribution is the exact solution for the inner product between the weight and activation random variables, it is hard to work with in subsequent layers. For this reason, and the fact that the Poisson binomial distribution is well approximated by a Normal distribution (Wang & Manning, 2013), we use a Normal approximation to the Poisson binomial distribution, which allows for easier manipulations. Using the properties of the Binary distribution and the Poisson binomial distribution, the approximation for the pre-activation a is given by:

$$a = \mathbf{w} \cdot \mathbf{h} \quad \dot{\sim} \quad \mathcal{N}(\sum_{d=1}^{D} \theta_d \phi_d, D - \sum_{d=1}^{D} \theta_d^2 \phi_d^2). \tag{4}$$

This is exactly the approximation one would obtain by using the Lyapunov Central Limit Theorem (CLT). The CLT approximation was also used by Shayer et al. (2018) to train a neural network with binary weights by sampling from this normal distribution using the reparametrization trick (Kingma & Welling, 2014). When combined, this is known as the *local* reparametrization trick (Kingma et al., 2015).

Equation 4 gives a Normal approximation to the pre-activation given that both the weights and activations of the previous layer are distributed according to a Binary distribution as defined in Equation 2. In Section 2.1, the application of the binarization function is discussed. Since we propagate random variables (or distributions) throughout the network, any other layer-type must also be applied on the level of random variables. For this reason, we introduce an interpretation of Batch Normalization (Ioffe & Szegedy, 2015) and max pooling that can be trained in a stochastic setting and applied in a deterministic setting. Pseudo code for the full forward pass of a single layer, including batch normalization and max pooling, is given in Algorithm 1.

2.1 Stochastic Binary Activation

Since the output of a linear operation using binary inputs is not restricted to be binary, it is required to apply a binarization operation to the pre-activation in order to obtain binary activations. Various works – e.g., Hubara et al. (2016) and Rastegari et al. (2016) – use either deterministic or stochastic binarization functions, i.e.,

$$b_{\text{det}}(a) = \begin{cases} +1 & \text{if } a \geq 0 \\ -1 & \text{otherwise} \end{cases} \quad b_{\text{stoch}}(a) = \begin{cases} +1 & \text{with probability } p = \text{sigmoid}(a) \\ -1 & \text{with probability } 1 - p \end{cases}. \tag{5}$$

In our case the pre-activations are random variables. Hence, applying a deterministic binarization function to a random pre-activations results in a stochastic binary activation. Specifically, let $a_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ be a random pre-ctivation obtained using the normal approximation, as introduced in the previous section, then the activation (after binarization) is again given as a Binary random variable". Interestingly, the Binary probability can be computed in closed form by evaluating the probability density that lies above the binarization threshold:

$$h_i = b_{\text{det}}(a_i) \sim \text{Binary}(q_i), \quad q_i = 1 - \Phi(0|\mu_i, \sigma_i^2), \tag{6}$$

where $\Phi(\cdot|\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ denotes the CDF of $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$. Applying the binarization function to a random pre-activation has two advantages. First, the derivatives $\partial q_i / \partial \mu_i$ and $\partial q_i / \partial \sigma_i$ are not zero almost everywhere, in contrast to the derivatives of b_{det} and b_{stoch} when applied to a deterministic input. Second, the distribution over h_i reflects the true uncertainty about the sign of the activation, given the stochastic weights, whereas b_{stoch} uses the magnitude of the pre-activation as a substitute. For example, a pre-activation with a high positive magnitude and high variance will be deterministically mapped to 1 by b_{stoch}. In contrast, our method takes the variance into account and correctly assigns some probability mass to -1. See Figure 1 for a graphical depiction of the stochastic binary activation.

2.1.1 Sampling the Binary Activations

So far, we have discussed propagating distributions throughout the network. Alternatively, the binary activations can be sampled using the Concrete distribution (Maddison et al., 2016) during training. specifically, we use the *hard* sample method as discussed by Jang et al. (2016). By sampling the activations, the input for subsequent layers will match the input that is observed at test time more closely.

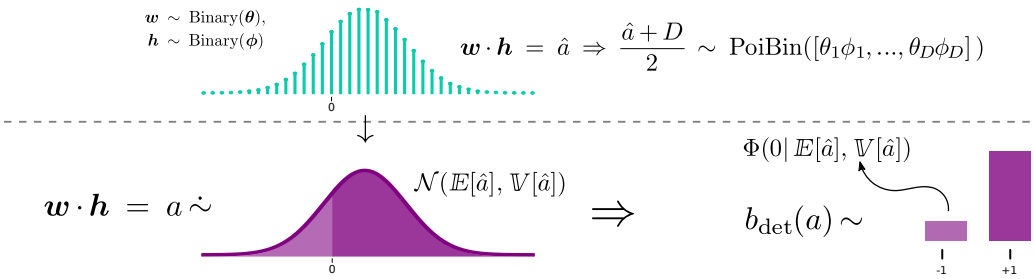


Figure 1: The discrete Poisson binomial distribution (in green) is approximated by a continuous Gaussian distribution (in purple). By applying b_{det} to a random pre-activation we obtain a binary activation distribution.

As a consequence of sampling the activation, the input to a layer is no longer a distribution but a $\mathbf{h} \in \{-1, +1\}^D$ vector instead. As such, the normal approximation to the pre-activation is computed slightly different. From the Lyapunov CLT it follows that the approximation to the distribution of the pre-activation is given by:

$$a = \mathbf{w} \cdot \mathbf{h} \quad \dot{\sim} \quad \mathcal{N}(\sum_{d=1}^{D} \theta_d h_d, \sum_{d=1}^{D} \theta_d^2 h_d^2), \tag{7}$$

where $\mathbf{w} \sim \text{Binary}(\boldsymbol{\theta})$ is a random weight. Similarly, the pre-activation of the input layer is also computed using this approximation—given a real-valued input vector. We will refer to a PBNet that uses activation sampling as PBNet-S.

2.2 Normalization and Pooling

Other than a linear operation and an (non-linear) activation function, Batch Normalization (Ioffe & Szegedy, 2015) and pooling are two popular building blocks for Convolutional Neural Networks. For Binary Neural Networks, applying Batch Normalization to a binarized activation will result in a non-binary result. Moreover, the application of max pooling on a binary activation will result in a feature map containing mostly $+1$s. Hence, both operations must be applied before binarization. However, in the PBNet, the binarization operation is applied before sampling. As a consequence, the Batch Normalization and pooling operations can only be applied on random pre-activations. For this reason, we define these methods for random variables. Although there are various ways to define these operation in a stochastic fashion, our guiding principle is to only leverage stochasticity during training, i.e., at test time, the stochastic operations are replaced by their conventional implementations and parameters learned in the stochastic setting must be transferred to their deterministic counterparts.

2.2.1 Stochastic Batch Normalization

Batch Normalization (BN) (Ioffe & Szegedy, 2015) — including an affine transformation — is defined as follows:

$$\hat{\mathbf{a}}_i = \frac{\mathbf{a}_i - \mathbf{m}}{\sqrt{\mathbf{v} + \epsilon}} \gamma + \beta, \tag{8}$$

where $\mathbf{a}_i$ denotes the pre-activation before BN, $\hat{\mathbf{a}}$ the pre-activation after BN, and $\mathbf{m}$ & $\mathbf{v}$ denote the sample mean and variance of $\{\mathbf{a}_i\}_{i=1}^{M}$, for an M-dimensional pre-activation. In essence, BN translates and scales the pre-activations such that they have approximately zero mean and unit variance, followed by an affine transformation. Hence, in the stochastic case, our aim is that samples from the pre-activation distribution after BN also have approximately zero mean and unit variance—to ensure that the stochastic batch normalization can be transfered to a deterministic binary neural network. This is achieved by subtracting the population mean from each pre-activation random variable and by dividing by the population variance. However, since $\mathbf{a}_i$ is a random variable in the PBNet, simply using the population mean and variance equations will result in non-standardized output. Instead, to ensure a standardized distribution over activations, we compute the expected

population mean and variance under the pre-activation distribution:

$$\mathbb{E}_{p(\mathbf{a}|\mathbf{B},\mathbf{h})}[\mathbf{m}] = \mathbb{E}\left[\frac{1}{M}\sum_{i=1}^{M}\mathbf{a}_i\right] = \frac{1}{M}\sum_{i=1}^{M}\mathbb{E}\left[\mathbf{a}_i\right] = \frac{1}{M}\sum_{i=1}^{M}\boldsymbol{\mu}_i \tag{9}$$

$$\mathbb{E}_{p(\mathbf{a}|\mathbf{B},\mathbf{h})}[\mathbf{v}] = \mathbb{E}\left[\frac{1}{M-1}\sum_{i=1}^{M}(\mathbf{a}_i - \mathbb{E}[\mathbf{m}])^2\right] = \frac{1}{M-1}\left\{\sum_{i=1}^{K}\boldsymbol{\sigma}_i^2 + \sum_{i=1}^{M}\left(\boldsymbol{\mu}_i - \mathbb{E}[\mathbf{m}]\right)^2\right\}, \tag{10}$$

where M is the total number of activations and $\mathbf{a}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i)$ are the random pre-activations. By substituting $\mathbf{m}$ and $\mathbf{v}$ in Equation 8 by Equation 9 and 10, we obtain the following batch normalized Gaussian distributions for the pre-activations:

$$\hat{\mathbf{a}}_i = \frac{\mathbf{a}_i - \mathbb{E}[\mathbf{m}]}{\sqrt{\mathbb{E}[\mathbf{v}] + \epsilon}}\gamma + \beta \quad \Rightarrow \quad \hat{\mathbf{a}}_i \sim \mathcal{N}\left(\frac{\boldsymbol{\mu}_i - \mathbb{E}[\mathbf{m}]}{\sqrt{\mathbb{E}[\mathbf{v}] + \epsilon}}\gamma + \beta, \frac{\gamma^2}{\mathbb{E}[\mathbf{v}] + \epsilon}\boldsymbol{\sigma}_i^2\right). \tag{11}$$

Note that this assumes a single channel, but is easily extended to 2d batch norm in a similar fashion as conventional Batch Normalization. At test time, Batch Normalization in a Binary Neural Network can be reduced to an addition and sign flip of the activation, see Appendix A for more details.

2.2.2 Stochastic Max Pooling

In general, pooling applies an aggregation operation to a set of (spatially oriented) pre-activations. Here we discuss max pooling for stochastic pre-activations, however, similar considerations apply for other types of aggregation functions.

In the case of max-pooling, given a spatial region containing stochastic pre-activations $\mathbf{a}_1, \ldots, \mathbf{a}_K$, we aim to stochastically select one of the $\mathbf{a}_i$. Note that, although the distribution of $\max(\mathbf{a}_1, \ldots, \mathbf{a}_K)$ is well-defined (Nadarajah & Kotz, 2008), its distribution is not Gaussian and thus does not match one of the input distributions. Instead, we sample one of the input random variables in every spatial region according to the probability of that variable being greater than all other variables, i.e., $\rho_i = p(\mathbf{a}_i > \mathbf{z}_{\setminus i})$, where $\mathbf{z}_{\setminus i} = \max(\{\mathbf{a}_j\}_{j \neq i})$. ρ_i could be obtained by evaluating the CDF of $(\mathbf{z}_{\setminus i} - \mathbf{a}_i)$ at 0, but to our knowledge this has no analytical form. Alternatively, we can use Monte-Carlo integration to obtain $\boldsymbol{\rho}$:

$$\boldsymbol{\rho} \approx \frac{1}{L}\sum_{l=1}^{L}\text{one-hot}(\arg\max \mathbf{s}^{(l)}), \quad \mathbf{s}^{(l)} \sim p(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_K) = \prod_{i=1}^{K}\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2) \tag{12}$$

where one-hot(i) returns a K-dimensional one-hot vector with the ith elements set to one. The pooling index $\boldsymbol{\iota}$ is then sampled from Cat$(\boldsymbol{\rho})$. However, more efficiently, we can sample $\mathbf{s} \sim p(\mathbf{a}_1, \ldots, \mathbf{a}_K)$ and select the index of the maximum in $\mathbf{s}$, which is equivalent sampling from Cat$(\boldsymbol{\rho})$. Hence, for a given max pooling region, it is sufficient to obtain a single sample from each normal distribution associated with each pre-activation and *keep* the random variable for which this sample is maximum. A graphical overview of this is given in Figure 2.

Other forms of stochastic or probabilistic max pooling were introduced by Lee et al. (2009) and Zeiler & Fergus (2013), however, in both cases a single activation is sampled based on the magnitude of the activations. In contrast, in our procedure we stochastically propagate one of the input distributions over activations.

2.3 Weight Initialization

For the PBNet the parameters $\boldsymbol{\theta}$ for $q_{\boldsymbol{\theta}}(\mathbf{B})$ are initialized from a uniform $U(-1, 1)$ distribution. Although the final parameter distribution more closely follows a Beta(α, α) distribution, for $\alpha < 1$, we did not observe any significant impact choosing another initialization method for the PBNet.

In the case of the PBNet-S, we observed a significant improvement in training speed and performance by initializing the parameters based on the parameters of a pre-trained full precission Neural Network. This initializes the convolutional filters with more structure than a random initialization. This is desirable as in order to flip the value of a weight, the parameter governing the weight has to pass through a high variance regime, which can slow down convergence considerably.

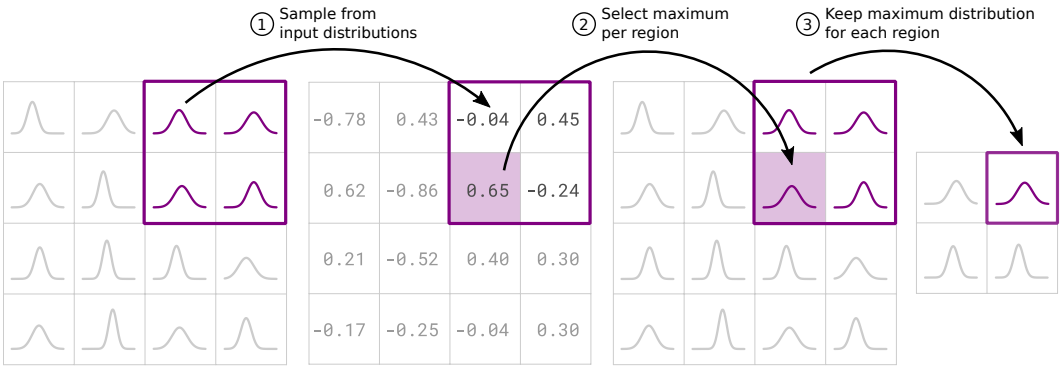


Figure 2: Max pooling for random variables is performed by taking a single sample from each of the input distributions. The output random variable for each pooling region is the random variable that is associated with the maximum sample.

For the PBNet-S, We use the weight transfer method introduced by Shayer et al. (2018) in which the parameters of the weight distribution for each layer are initialized such that the expected value of the random weights equals the full precision weight divided by the standard deviation of the weights in the given layer. Since not all rescaled weights lay in the $[-1, 1]$ range, all binary weight parameters are clipped between $[-0.9, 0.9]$. This transfer method transfers the structure present in the filters of the full precision network and ensures that a significant part of the parameter distributions is initialized with low variance.

2.4 Deterministic Binary Neural Network

In our training procedure, a stochastic neural network is trained. However, at test time (or on hardware) we want to leverage all the advantages of a full binary Neural Network. Therefore, we obtain a deterministic binary Neural Network from the parameter distribution $q_{\boldsymbol{\theta}}(\mathbf{B})$ at test time. We consider three approaches for obtaining a deterministic network: a deterministic network based on the mode of $q_{\boldsymbol{\theta}}(\mathbf{B})$ called PBNet-MAP, an ensemble of binary Neural Networks sampled from $q_{\boldsymbol{\theta}}(\mathbf{B})$ named PBNet-x, and a ternary Neural Network (PBNet-Ternary), in which a single parameter W_i may be set to zero based on $q_{\boldsymbol{\theta}}$, i.e.:

$$W_i = \begin{cases} +1 & \text{if } q_{\boldsymbol{\theta}}(B_i = +1) \geq 3/4 \\ -1 & \text{if } q_{\boldsymbol{\theta}}(B_i = -1) \geq 3/4 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

The ternary network can also be viewed as a sparse PBNet, however, sparse memory look-ups may slow down inference.

Note that, even when using multiple binary neural networks in an ensemble, the ensemble is still more efficient in terms of computation and memory when compared to a full precision alternative. Moreover, it allows for anytime ensemble predictions for improved performance and uncertainty estimates by sampling from the weight distribution.

Since the trained weight distribution is not fully deterministic, the sampling of individual weight instantiations will result in a shift of the batch statistics. As a consequence, the learned batch norm statistics no longer closely match the true statistics. This is alleviated by re-estimating the batch norm statistics based on (a subset of) the training set after weight sampling using a moving mean and variance estimator. We observed competitive results using as little as 20 batches from the training set.

3 Related Work

Binary and low precision neural networks have received significant interest in recent years. Most similar to our work, in terms of the final neural network, is the work on Binarized Neural Networks by Hubara et al. (2016). in this work a real-valued shadow weight is used and binary weights are

obtained by binarizing the shadow weights. Similarly the pre-activations are binarized using the same binarization function. In order to back-propagate through the binarization operation the straight-through estimator (Hinton, 2012) is used. Several extensions to Binarized Neural Networks have been proposed which — more or less — qualify as binary neural networks: XNOR-net (Rastegari et al., 2016) in which the real-valued parameter tensor and activation tensor is approximated by a binary tensor and a scaling factor per channel. ABC-nets Lin et al. (2017) take this approach one step further and approximate the weight tensor by a linear combination of binary tensors. Both of these approaches perform the linear operations in the forward pass using binary weights and/or binary activations, followed by a scaling or linear combination of the pre-activations. In McDonnell (2018), similar methods to Hubara et al. (2016) are used to binarize a wide resnet (Zagoruyko & Komodakis, 2016) to obtain results on ImageNet very close to the full precision performance. Another method for training binary neural networks is Expectation Backpropagation (Soudry et al., 2014) in which the central limit theorem and online expectation propagation is used to find an approximate posterior. This method is similar in spirit to ours, but the training method is completely different. Most related to our work is the work by Shayer et al. (2018) which use the local reparametrization trick to train a Neural Network with binary weights and the work by Baldassi et al. (2018) which also discuss a binary Neural Network in which the activation distribution are propagated through the network. Moreover, in (Wang & Manning, 2013) the CLT was used to approximate dropout noise during training in order to speed up training, however, there is no aim to learn binary (or discrete) weights or use binary activations in this work.

4 Experiments

We evaluate the PBNet on the MNIST and CIFAR-10 benchmarks and compare the results to Binarized Neural Networks (Hubara et al., 2016), since the architectures of the deterministic networks obtained by training the PBNet are equivalent.

4.1 Experimental Details

The PBNets are trained using either a cross-entropy (CE) loss or a binary cross entropy for each class (BCE). For the CE loss there is no binarization step in the final layer, instead the mean of the Gaussian approximation is used as the input to a softmax layer. For BCE, there is a binarization step, and we treat the probability of the ith output being $+1$ as the probability of the input belonging to the ith class. Specifically, for an output vector $\mathbf{p} \in [0,1]^C$ for C classes and the true class y, the BCE loss for a single sample is defined as

$$\mathcal{L}_{\text{BCE}}(\mathbf{p}, y) = -\sum_{c=1}^{C} [c = y] \log p_c + [c \neq y] \log(1 - p_c). \tag{14}$$

The weights for the PBNet-S networks are initialized using the transfer method described in Section 2.3 and the PBNets are initialized using a uniform initialization scheme. All models are optimized using Adam (Kingma & Ba, 2014) and a validation loss plateau learning rate decay scheme. We keep the temperature for the binary concrete distribution static at 1.0 during training. For all settings, we optimize model parameters until convergence, after which the best model is selected based on a validation set. Our code is implemented using PyTorch (Paszke et al., 2017).

For Binarized Neural Networks we use the training procedure described by Hubara et al. (2016), i.e., a squared hinge loss and layer specific learning rates that are determined based on the Glorot initialization method (Glorot & Bengio, 2010).

Experimental details specific to datasets are given in Appendix B and the results are presented in Table 1. We report both test set accuracy obtained after binarizing the network as well as the the test set accuracy obtained by the stochastic network during training (i.e., by propagating activation distributions).

4.2 Ensemble Based Uncertainty Estimation

As presented in Table 1 the accuracy improves when using an ensemble. Moreover, the predictions of the ensemble members can be used to obtain an estimate of the certainty of the ensemble as a whole.

Table 1: Test accuracy on MNIST and CIFAR-10 for Binarized NN (Hubara et al., 2016), PBNet, and a full precission network (FPNet). PBNet-map refers to a deterministic PBNet using the map estimate, PBNet-Ternary is a ternary deterministic network obtained from $q_{\boldsymbol{\theta}}$, and PBNet-X refers to an ensemble of X networks, each sampled from the same weight distribution. For the ensemble results both mean and standard deviation are presented. The propagate column contains results obtained using the stochastic network whereas results in the binarized column are obtained using a deterministic binary Neural Network.

	MNIST		CIFAR-10	
	PROPAGATE	BINARIZED	PROPAGATE	BINARIZED
BINARIZED NN	–	99.17	–	88.17
PBNET-MAP (BCE)	99.35	99.13	88.24	79.98
PBNET-MAP (CE)	99.24	98.64	86.73	75.05
PBNET-S-MAP (BCE)	99.26	99.22	89.58	89.10
PBNET-S-MAP (CE)	99.14	99.05	88.67	88.54
PBNET-S-TERNARY (BCE)	99.26		89.70	
PBNET-S-2 (BCE)	99.25 ± 0.047		89.75 ± 0.205	
PBNET-S-5 (BCE)	99.29 ± 0.036		90.75 ± 0.202	
PBNET-S-16 (BCE)	99.30 ± 0.025		91.28 ± 0.112	
FPNET	99.48		92.45	

To evaluate this, we plot an error-coverage curve (Geifman & El-Yaniv, 2017) in Figure 3a. This curve is obtained by sorting the samples according to a statistic and computing the error percentage in the top $x\%$ of the samples – according to the statistic. For the Binarized Neural Network and PBNet-MAP the highest softmax score is used, whereas for the ensembles the variance in the prediction of the top class is used. The figure suggests that the ensemble variance is a better estimator of network certainty, and moreover, the estimation improves as the ensemble sizes increases.

4.3 EFFECT OF BATCH STATISTICS RE-ESTIMATION

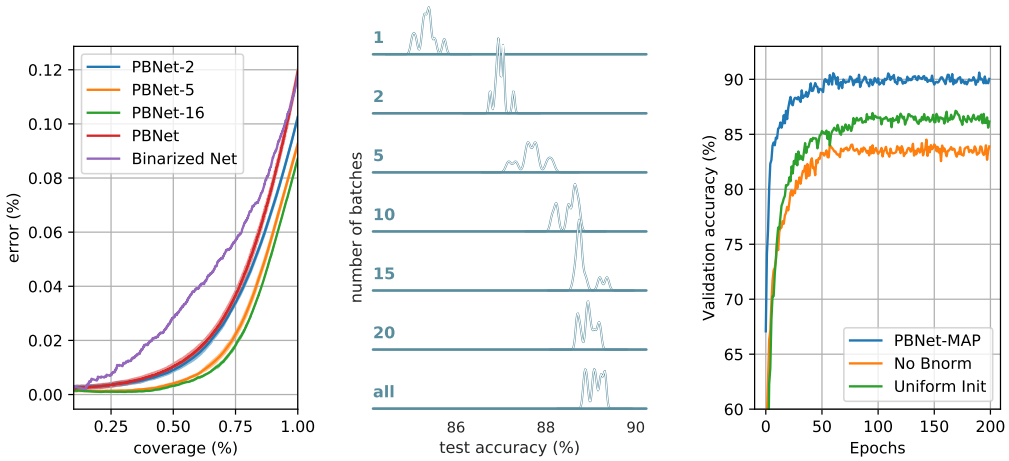
As discussed in Section 2.4, after sampling the parameters of a deterministic network the batch statistics used by Batch Normalization must be re-estimated. Figure 3b shows the results obtained using a various number of batches from the training set to re-estimate the statistics. This shows that even a small number of samples is sufficient to estimate the statistics.

4.4 ABLATION STUDIES

We perform an ablation study on both the use of (stochastic) Batch Normalization and the use of weight transfer for the PBNet-S on CIFAR-10. For Batch Normalization, we removed all batch normalization layers from the PBNet-S and retrained the model on CIFAR-10. This resulted in a test set accuracy of 79.21%. For the weight initialization experiment, the PBNet-S weights are initialized using a uniform initialization scheme and is trained on CIFAR-10, resulting in a test set accuracy of 83.61%. Moreover, the accuracy on the validation set during training is presented in Figure 3c. Note that these numbers are obtained *without* sampling a binarized network from the weight distribution, i.e., local reparametrization and binary activation samples are used. The PBNet-S that uses both weight transfer and stochastic Batch Normalization results in a significant performance improvement, indicating that both stochastic Batch Normalization and weight transfer are necessary components for the PBNet-S.

4.5 SAMPLING OF BINARY ACTIVATIONS

The results of our experiments show that, following our training procedure, sampling of the binary activations is a necessary component. Although the stochastic PBNet generalizes well to unseen data, there is a significant drop in test accuracy when a binary Neural Network is obtained from



(a) Error coverage for CIFAR-10.

(b) Test set performance with increasing number of batches used to re-estimate the batch statistics on CIFAR-10.

(c) Accuracy on validation set during training, i.e., using stochastic weights, local reparametrization and binary activation sampling.

Figure 3: Error coverage curve, batch statistic re-estimation results and ablation study results for CIFAR-10.

the stochastic PBNet. In contrast, this performance drop is not observed for PBNet-S. A potential explanation of this phenomenon is that by sampling the binary activation during training, the network is forced to become more robust to the inherent binarization noise that is present at test time of the binarized Neural Network. If this is the case, then sampling the binary activation can be thought of as a regularization strategy that *prepares* the network for a more noisy binary setting. However, other regularization strategies may also exist.

5 Conclusion

We have presented a stochastic method for training Binary Neural Networks. The method is evaluated on multiple standardized benchmarks and reached competitive results. The PBNet has various advantageous properties as a result of the training method. The weight distribution allows one to generate ensembles online which results in improved accuracy and better uncertainty estimations. Moreover, the Bayesian formulation of the PBNet allows for further pruning of the network, which we leave as future work.

References

Jan Achterhold, Jan Mathias Koehler, Anke Schmeink, and Tim Genewein. Variational network quantization. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=ry-TW-WAb`.

Carlo Baldassi, Federica Gerace, Hilbert J Kappen, Carlo Lucibello, Luca Saglietti, Enzo Tartaglione, and Riccardo Zecchina. Role of synaptic stochasticity in training low-precision neural networks. *Physical review letters*, 120(26):268103, 2018.

Angus Galloway, Graham W. Taylor, and Medhat Moussa. Attacking binarized neural networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=HkTEFfZRb`.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Advances in neural information processing systems*, pp. 4885–4894, 2017.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Geoffrey Hinton. Neural networks for machine learning. 2012.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pp. 4107–4115, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014.

Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pp. 609–616. ACM, 2009.

Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pp. 344–352, 2017.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pp. 3290–3300, 2017.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Mark D. McDonnell. Training wide residual networks for deployment using a single bit for each weight. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rytNfI1AZ`.

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(2):210–212, 2008.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

Oran Shayer, Dan Levi, and Ethan Fetaya. Learning discrete weights using the local reparameterization trick. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BySRH6CpW`.

Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems*, pp. 963–971, 2014.

Joe Staines and David Barber. Variational optimization. *arXiv preprint arXiv:1212.4507*, 2012.

Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

Sida Wang and Christopher Manning. Fast dropout training. In *international conference on machine learning*, pp. 118–126, 2013.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Matthew D Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

A BATCH NORMALIZATION IN A BINARY NEURAL NETWORK

During training the PBNet is trained using stochastic Batch Normalization. At test time, the parameters learned using stochastic Batch Normalization can be transferred to a *conventional* Batch Normalization implementation. Alternatively, Batch Normalization can be reduced to an (integer) addition and multiplication by ± 1 after applying the sign activation function. Given a pre-activation a, the application of Batch Normalization followed by a sign binarization function can be rewritten as:

$$\mathrm{sign}\left(\frac{a-m}{\sqrt{v+\epsilon}}\gamma+\beta\right) = \mathrm{sign}\left(\frac{a}{\sqrt{v+\epsilon}}\gamma-\frac{m}{\sqrt{v+\epsilon}}\gamma+\beta\right) \tag{15}$$

$$= \mathrm{sign}\left(a\gamma - m\gamma + \beta\sqrt{v+\epsilon}\right) \tag{16}$$

$$= \mathrm{sign}(\gamma)\,\mathrm{sign}\left(a - m + \frac{\beta}{\gamma}\sqrt{v+\epsilon}\right) \tag{17}$$

when $a \in \mathbb{Z}$, which is the case for all but the first layer

$$= \mathrm{sign}(\gamma)\,\mathrm{sign}\left(a + \underbrace{\left\lfloor \frac{\beta}{\gamma}\sqrt{v+\epsilon} - m \right\rfloor}_{b}\right) \tag{18}$$

$$= \mathrm{sign}(\gamma)\,\mathrm{sign}\left(a+b\right) \tag{19}$$

Note that we have used $\mathrm{sign}(0) = b_{\text{det}}(0) = +1$ here, as we have used everywhere in order to use sign as a binarization function.

B EXPERIMENTAL DETAILS

B.1 MNIST

The MNIST dataset consists of of 60K training and 10K test 28×28 grayscale handwritten digit images, divided over 10 classes. The images are pre-processed by subtracting the global pixel mean and dividing by the global pixel standard deviation. No other form of pre-processing or data augmentation is used. For MNIST, we use the following architecture:

$$32\text{C}3 \;-\; \text{MP}2 \;-\; 64\text{C}3 \;-\; \text{MP}2 \;-\; 512\text{FC} \;-\; \text{SM}10$$

where XC3 denotes a binary convolutional layer using 3×3 filters and X output channels, YFC denotes a fully connected layer with Y output neurons, SM10 denotes a softmax layer with 10 outputs, and MP2 denotes 2×2 (stochastic) max pooling with stride 2. Note that if a convolutional layer is followed by a max pooling layer, the binarization is only performed after max pooling. All layers are followed by (stochastic) batch normalization and binarization of the activations. We use a batchsize of 128 and an initial learning rate of $10^{-}2$ Results are reported in Table 1.

B.2 CIFAR-10

The CIFAR-10 (Krizhevsky & Hinton, 2009) dataset consists of 50K training and 10K test 32×32 RGB images divided over 10 classes. The last 5,000 images from the training set are used as validation set. Tthe images are only pre-processed by subtracting the channel-wise mean and dividing by the standard deviation. We use the following architecture for our CIFAR-10 experiment: as Shayer et al. (2018) is used, i.e.,

$$2 \times 128\text{C3} \; - \; \text{MP2} \; - \; 2 \times 256\text{C3} \; - \; \text{MP2} \; - \; 2 \times 512\text{C3} \; - \; \text{MP2} \; - \; 1024\text{FC} \; - \; \text{SM10}$$

where we use the same notation as in the previous section. The Binarized Neural Network baseline uses the same architecture, except for one extra 1024 neuron fully connected layer. During training, the training set is augmented using random 0px to 4px translations and random horizontal fl Results are reported in Table 1.