Training Binarized Neural Networks the Easy Way

Alasdair Paren[1]
https://alasdair-p.github.io/Alasdair-P/
Rudra P. K. Poudel[2]
https://www.rudrapoudel.com

[1] Department of Engineering Science
University of Oxford
Oxford, UK

[2] Cambridge Research Laboratory,
Toshiba Europe Ltd,
Cambridge, UK.

Abstract

In this work we present a simple but effective method for training Binarized Neural Networks (BNNs). Specifically, models where the majority of both weights and activations are constrained to the set $\{-1,1\}$. These models offer significant improvements in memory efficiency, energy usage and inference speed over their floating point counterparts. Our approach to training BNN splits the task into two phases. In the first phase a model with binary activations and floating point weights is trained. In the second, a concave regulariser is added to encourage the weights to become binary. This work is orthogonal to improvements of BNN architectures, and offers an alternative optimisation scheme for these models. Our method doesn't require an auxiliary set of weights during training and can be easily applied to any existing architectures. Finally, we achieve a new state of the art training a BNN on the ImageNet data set.

1 Introduction

During the last decade there has been a steady trend towards the use of increasingly larger neural networks. Exceptionally large neural networks have led to some of the most exciting breakthroughs in machine learning. For example, very large transformer architectures have redefined what is possible with language modelling [2, 8, 26]. However, large models with billions of parameters have inherent limitations due to their computational costs. These models require large, expensive, and power hungry hardware. For example, the largest version of GPT-3 required a staggering 10^{23} floating-point operations during training [4]. Befitting hardware is required at inference time to run large models at reasonable speeds. This dependence on specialised hardware presents two major drawbacks for using large models. First, it restricts use cases preventing their use in power, size, or compute limited settings such as mobile devices and remote sensing applications. In some use cases it may be possible to offload the heavy computation to the cloud. However, this depends on the bandwidth and stability of the connection, and importantly the criticality of the application, and thus is not always possible. Second, even in applications where dedicated hardware is not a limiting factor, the energy usage of very large over-parameterised models can still be significant, and

this conflicts with a push to a more sustainable future. These limitations have led to significant work to reduce the memory and computational costs of neural networks while retaining strong generalisation performance. Many different methods have been proposed to this end, including distillation, efficient architectures, network pruning, and various compression and quantisation techniques. At the moment these are all active areas of research and it remains unclear which provides the best results [22]. In this work we focus on network quantisation.

Binarized Neural Networks (BNN) are an extreme form of network quantisation where the majority of weights and activations are constrained to the set $\{-1,1\}$. This allows almost all of the floating point arithmetic operations to be replaced with faster bit-wise alternatives that can be performed on far cheaper hardware. Due to the discrete nature of binary parameters, without modification existing continuous optimisation techniques such as stochastic gradient descent and its variants are unsuitable for training these models.

To address this issue Courbariaux et al. [6] proposed a "trick" that is now known as the Straight Through Estimator (STE) method. The STE method relies on using auxiliary parameters to accumulate successive updates allowing several small gradients to produce a sign change. Helwegen et al. [11] instead use the auxiliary parameters to keep track of a moving average of the gradient and change sign if the value passes a threshold. More recently Mirror Descent View for Neural Network Quantisation [1] introduced a closely related method derived as a numerically stable implementation of Mirror Descent. This method helps provide more of a theoretical justification for the STE method and solid results when well tuned.

We take inspiration from the work of Bai et al. [2] who provide a more principled method of training binary weight neural networks. We relax the constraint on the binary parameters to its convex hull and use a simple concave regulariser to force the solution to become binary. Hence, we present a continuation method with a clear objective that is being minimised at each step. Thus, we name our approach Binary Network the Easy Way or BNEW .

2 Preliminaries

Model Parameters and Quantisation scheme. We use $\boldsymbol{w}$ to denote the vector containing the d parameters within a quantised neural network. We use $\boldsymbol{w}^b$ to represent the subset of parameters that we want to take discrete values. For clarity, note that $\boldsymbol{w}^b$ does not always encode the final quantised values themselves. Hence, during an intermediate stage of training, some of the values of $\boldsymbol{w}^b$ may refer to a real valued vector. Many quantisation schemes have been proposed in the literature offering different benefits. We refer the interested reader to [9] for a good review of this area. However, the selection of a scheme is not the focus of this work and thus we choose to stick to a canonical binary quantisation scheme. We aim to obtain a model with $\boldsymbol{w}^b \in \{-1,1\}^p$. As is common practice, we let some parameters retain floating point values. We denote the vector containing these unconstrained parameters with $\boldsymbol{w}^r \in \mathbb{R}^{d-p}$. This vector typically includes the weights in the first and last layers of the network, all biases, bottleneck layer and parameters of batch norm layers. In our notation $\boldsymbol{w}$ is simply the concatenation of $\boldsymbol{w}^b$ and $\boldsymbol{w}^r$. Finally, we define the feasible set $\Phi \triangleq \{-1,1\}^p \cup \mathbb{R}^{d-p}$ which represents the constraints on $\boldsymbol{w}$.

Binary Activations. In this paper we focus on BNN or binary neural networks with both discrete weights and activations. In these models dot products can be implemented using a

bitwise XNOR operation followed by a bit count operation. This is in contrast to conventional floating point models where a dot product requires multiple floating point multiplications and accumulation operations. Hence BNNs offer a drastic speed up, in inference speed and latency. This is especially true for architectures with less parallelism such as CPUs [25]. In order to ensure that the activations of a BNN are binary the sign function is applied to the inputs of every convolutional or linear layer. The gradient of the sign function is zero almost everywhere. In order to back-propagate gradients through the model it is necessary to approximate the sign function's derivative with something continuous. A common choice for this approximation is the Straight Through Estimator, which approximates the derivative of $\text{sign}(x)$ in the backwards pass with:

$$\frac{\partial \text{sign}(x)}{\partial x} = \begin{array}{ll} 1, & \text{for } |x| \leq 1, \\ 0, & \text{for } |x| > 1. \end{array} \tag{1}$$

Alternatively, [19, 20] suggest the use of the following approximation, which we adopt in this paper, however, empirically we find that its benefits are limited.

$$\frac{\partial \text{sign}(x)}{\partial x} = \begin{array}{lll} 2x+2, & \text{for} & -1 \leq x \leq 0, \\ 2-2x, & \text{for} & 0 \leq x \leq 1, \\ 0, & \text{for} & 1 \leq |x|. \end{array} \tag{2}$$

Training Resources. Typically a BNN is desired to efficiently perform some inference task on some lightweight hardware, for example controlling the steering and navigation in a self driving car. It is commonly assumed that the model performing this task does not need to be trained in-place on the lightweight hardware. Instead, extra computational resources are available and can be used. Revisiting our example, a high performance computing cluster could be used by the company developing the navigation system. Once trained, the final BNN is downloaded onto the many lightweight devices executing the task, the fleet of cars in our example. Hence, it is normally assumed when training BNN extra resources can be used and thus most works use floating point parameters during this phase. Additionally, as inference or test performance is the focus the training, cost and time are not the number one concern when considering BNNs. While this is the dominant paradigm in the literature, there are of course many interesting exceptions.

Loss Function. As is standard for supervised learning, we assume the objective function f can be expressed as an expectation over $z \in \mathcal{Z}$, where z is a random variable indexing the samples of the training set $\mathcal{Z}$. We define the loss function associated with the z^{th} sample as ℓ_z, and hence:

$$f(\boldsymbol{w}) \triangleq \mathbb{E}_{z \in \mathcal{Z}}[\ell_z(\boldsymbol{w})], \tag{3}$$

Learning Task. The task of training a BNN can be expressed as finding an optimal vector $\boldsymbol{w}_\star \in \Omega$ that minimises f:

$$\boldsymbol{w}_\star \in \underset{\boldsymbol{w} \in \Phi}{\text{argmin}}\, f(\boldsymbol{w}). \tag{$\mathcal{B}$}$$

This is a highly non-linear non-convex mixed integer program and thus is NP hard in general, and does not permit an efficient solution. Using projected sub-gradient descent to find a local optimum would require an unreasonably large learning rate to produce a sign change in $\boldsymbol{w}^b$. Next, we will discuss how notable previous works have tackled optimising $\mathcal{B}$.

3 Related Work

3.1 The Straight Thorough Estimator Method

What has now become known as The Straight Through Estimator (STE) method was first introduced in [5] as a method to train neural networks with binary weights and real valued activations. A year later [12] showed that this method could be used to train neural networks with both binary weights and activations. The STE method relies on introducing a second set of auxiliary parameters $\tilde{\boldsymbol{w}}^b \in \mathbb{R}^p$ one for each of the binary parameters. Binary parameters are calculated before every forward pass using $\boldsymbol{w}_t^b = \text{sign}(\tilde{\boldsymbol{w}}_t^b)$. The gradient is then evaluated on the model with the binary parameters $\boldsymbol{w}_t^b$. However, this gradient is used to update the auxiliary parameters in the backwards pass. At time step t this optimisation scheme can be succinctly described as follows:

$$\tilde{\boldsymbol{w}}_{t+1}^b = \Pi_{[-1,1]}(\tilde{\boldsymbol{w}}_t^b - \eta_t \nabla \ell_{z_t}(\boldsymbol{w}_t^b)), \tag{4}$$

$$\boldsymbol{w}_{t+1}^r = \boldsymbol{w}_t^r - \eta_t \nabla \ell_{z_t}(\boldsymbol{w}_t^r), \tag{5}$$

where η_t is the learning rate and $\Pi_{[-1,1]}$ is projection onto the interval $[-1,1]$. In practice the Adam optimiser [14] update is typically used. The STE method has been used as the work horse in training many interesting extensions since the pioneering work of [5] and [12]. These works focus on adding extra scalars and parameters [17, 25], different quantisation schemes[16, 28, 30], and architectures designed for quantised weights [18] to name a few. More recently, the STE method has been used to good effect by works such as [20] which have by clever architectural design pushed the performance of BNNs, and offer state of the art levels. We refer the interested reader to [9] for a more thorough review of the area. Of special note are the works that removed the need for real valued weights during training [7, 29]. These works present methods suited to training on low compute devices. However, in this work we choose to focus on the more standard assumption that extra resources are available at training time, and the goal is to develop a lightweight model for inference.

3.2 Mirror Descent

Recently Ajanthan et al. [1] introduced a new method of training BNNs using Mirror Descent. Mirror Descent is a well studied first-order optimisation method for constrained convex problems [23]. However, before the work of Ajanthan et al. its application to training BNN had not been explored. In particular, this work introduced "MD-tanh-S" a numerically stable implementation of Mirror Descent that shares a lot of similarity with the STE method. For convenience we will refer to this method as Binary Mirror Descent (BMD). As mentioned, BMD is operationally similar to the STE method also introducing an auxiliary set of parameters $\tilde{\boldsymbol{w}}^b$. However, in BMD both sets of parameters $\tilde{\boldsymbol{w}}^b$ and $\boldsymbol{w}^b$ take real values throughout training. Similar to the STE method, before every forward pass $\tilde{\boldsymbol{w}}^b$ is mapped to $\boldsymbol{w}^b$ by the following equation:

$$\boldsymbol{w}_t^b = \tanh(\beta_t \tilde{\boldsymbol{w}}_t^b), \tag{6}$$

where β_t is a temperature parameter. The gradient is evaluated at $\boldsymbol{w}_t^b$ and then equations (4) and (5) are used to update $\tilde{\boldsymbol{w}}_t^b$. It is worth noting as $\beta_t \to \infty$, this mapping becomes equivalent to the sign function and in this case BMD is identical to the STE method. In practice, β_t starts at a low value $\beta_0 \approx 1$ and is increased throughout training according to the following scheme $\beta_t = \lambda_{rate}^t$. Mirror descent is a well studied algorithm, and is more principled than the STE method, which lacks strong justification from theory.

3.3 Binary Optimiser

Helwegen et al. [11] have also proposed a bespoke optimiser for the training of BNNs which they call Binary Optimiser (BOP). BOP aims to redefine the latent weights as an inertia rather than an accumulation of the gradients. Hence, they calculate an exponential moving average of the gradients for each binary weight, denoted m_t^b, using the following update:

$$\boldsymbol{m}_{t+1}^b = (1-\eta_t)\boldsymbol{m}_t^b - \eta_t \nabla \ell_{z_t}(\boldsymbol{w}_t^b). \tag{7}$$

After each update of equation (7), a binary parameter w_t^b has its sign flipped if m_t^b and w_t^b have the same sign and m_t^b has greater magnitude than a threshold hyperparameter τ. BOP can also be viewed as a modified version of the STE method where binarization in the forward pass is completed via $\boldsymbol{w}_{t+1}^b = \text{sign}(\tilde{\boldsymbol{w}}_t^b + \tau \text{sign}(\boldsymbol{w}_t^b))$ and the update (4) is modified to $\tilde{\boldsymbol{w}}_{t+1}^b = (1-\eta_t)\tilde{\boldsymbol{w}}_t^b - \eta_t \nabla \ell_{z_t}(\boldsymbol{w}_t^b)$. The authors of bop suggest that the floating point parameters $\boldsymbol{w}_t^r$ are updated using Adam with a different learning rate η_t^r. We find BOP performs well in practice, however, it lacks theoretical guarantees. More recently, [27] have proposed a modified version of BOP that keeps track of a vector containing a moving average of the second moment of the gradients which they label $\boldsymbol{v}_t^b$. At each time step, the first moment of each gradient m_{t+1}^b is then re-scaled by the $\frac{1}{\sqrt{v_t^b}}$ before comparing to the threshold τ.

3.4 ProxQuant

Bai et al. [2] introduced ProxQuant, a surprisingly simple and effective algorithm for training neural networks with binary weights. ProxQuant is notably different from the other methods. The ProxQuant algorithm does not require an additional auxiliary set of parameters, instead it acts on a single set of real valued weights that are initialised to a pretrained floating point network. Throughout the training procedure $\boldsymbol{w}^b$ is slowly encouraged to become binary by use of a regularisation function $R(\boldsymbol{w})$. The regularisation functions suggested by [2] penalise either the ℓ_1 or ℓ_2 norm of the distance between $\boldsymbol{w}^b$ and its nearest quantised value. In the case where a binary quantization scheme is used, these can be detailed as follows:

$$R_{\ell_2}(\boldsymbol{w}) = \|\boldsymbol{w}^b - \text{sign}(\boldsymbol{w}^b)\|^2, \qquad R_{\ell_1}(\boldsymbol{w}) = |\boldsymbol{w}^b - \text{sign}(\boldsymbol{w}^b)|. \tag{8}$$

From either of these regularisation functions the following proximal operator is derived:

$$\text{prox}_{\lambda R}(\boldsymbol{w}_t) = \underset{\boldsymbol{w}}{\text{argmin}} \left\{ \lambda R(\boldsymbol{w}) + \|\boldsymbol{w} - \boldsymbol{w}_t\|_2^2 \right\}. \tag{9}$$

With the proximal operator defined, the ProxQuant algorithm at time t can be summerised as:

$$\tilde{\boldsymbol{w}}_{t+1}^b = \text{prox}_{\eta_t \lambda_t R}\left(\tilde{\boldsymbol{w}}_t^b - \eta_t \nabla \ell_{z_t}(\boldsymbol{w}_t^b) \right). \tag{10}$$

Again, in practice the Adam optimiser update is used within the proximal operator. λ_t is increased throughout training according to a linear scheme. Finally, to ensure all the weights $\boldsymbol{w}^b$ are binary for the final part of training $\boldsymbol{w}^b$ is projected onto the set $\{-1,1\}^p$ and then $\boldsymbol{w}^r$ is fine-tuned to the these final values. ProxQuant only provided results for small models trained on the CIFAR data sets [15]. Our work is inspired by ProxQuant. However, critically we show, i) how their approach can be extended to models with both binary weights and activations; ii) the effectiveness when doing so; iii) that this approach scales to large binary models and data sets, and finally; iv) some additional improvements to boost performance.

4 Algorithm

4.1 Problem Formulation

We do not try to optimise ($\mathcal{B}$) directly and instead relax the constraint on $\boldsymbol{w}^b$ from $\boldsymbol{w}^b \in \{-1,1\}^p$ to $\boldsymbol{w}^b \in [-1,1]^p$. In order to ensure quantised solutions we introduce a concave regularisation function $R_{BNEW}(\boldsymbol{w})$ resulting in the formulation:

$$\boldsymbol{w}_\star \in \operatorname*{argmin}_{\boldsymbol{w}\in\Omega} F_\lambda \triangleq f(\boldsymbol{w}) + \lambda R_{BNEW}(\boldsymbol{w}), \tag{$\mathcal{P}$}$$

where $\Omega \triangleq [-1,1]^p \cup \mathbb{R}^{d-p}$ and $R_{BNEW}(\boldsymbol{w}) \triangleq -\|\boldsymbol{w}^b\|^2 + p$. Where p ensures the non negativity of the objective, however, we will suppress this term for clarity. It is clear that as $\lambda_t \to \infty$ any solution to the above problem must include $\boldsymbol{w}^b \in \{-1,1\}^p$, as all other solutions would incur infinite cost. With ($\mathcal{P}$) in this form we can make use of a Projected Stochastic Gradient Descent (PSGD) like update, which we detail in Section 4.2.

4.2 Parameter Update

In order to find a solution to ($\mathcal{P}$) at time step t we solve the following proximal problem:

$$\boldsymbol{w}_{t+1} = \operatorname*{argmin}_{\boldsymbol{w}\in\Omega} \Big\{ \frac{1}{2\eta_t}\|\boldsymbol{w}-\boldsymbol{w}_t\|^2 + \ell_{z_t}(\boldsymbol{w}_t) + \nabla\ell_{z_t}(\boldsymbol{w}_t)^\top(\boldsymbol{w}-\boldsymbol{w}_t) - \lambda_t\|\boldsymbol{w}^b\|^2 \Big\}. \tag{11}$$

As (11) is smooth and Ω is a convex set we can simply solve for $\boldsymbol{w}_{t+1}$ to get the following update for $\boldsymbol{w}^b$:

$$\boldsymbol{w}^b_{t+1} = \Pi_\Omega\left(\frac{1}{1-2\lambda_t\eta_t}(\boldsymbol{w}^b_t - \eta_t\nabla\ell_{z_t}(\boldsymbol{w}^b_t))\right) \approx \Pi_\Omega\left((1+2\lambda_t\eta_t)(\boldsymbol{w}^b_t - \eta_t\nabla\ell_{z_t}(\boldsymbol{w}^b_t))\right), \tag{12}$$

Where Π_Ω is the euclidean projection onto $[-1,1]^p$, and the second equality is approximately equal for small λ_t. The real value parameters $\boldsymbol{w}^r$ are updated according to (5). Note, when $\lambda_t = 0$, (12) is identical to PSGD. In practice the Adam update [14] is used over the vanilla SGD update inside of (5) and (12). In order to find a good solution to ($\mathcal{P}$) we use the following training process.

4.3 Training Procedure

Our training procedure is split into three phases. In the first phase we train a neural network with binary activations and real valued weights. In the second phase of training we slowly encourage the weights to become binary. Finally, we project $\boldsymbol{w}^b$ onto the set $\{-1,1\}^p$, and fine-tune the real valued parameters. At a high level, our training process is similar to that of ProxQuant and enjoys the same asymptotic convergence rate, see supplementary materials.

Pre-Training Phase. Similar to the work of [19, 20] we first train a neural network with binary activations and real valued weights. This is achieved by the addition of sign activations before each convolution layer. This results in the input to intermediate convolutional layers being binary. Our pre-training phase is different from that of ProxQuant in the following three ways. First, as we are concerned with training models with binary activations, the

model trained during the pre-training phase has binary activations. Second, we use the Adam optimiser rather than SGD during the pre-training phase. Third, we use PSGD or (12) with $\lambda_t = 0$, in other words we project $\boldsymbol{w}^b$ to the set $[-1,1]^p$ after every iteration. These last two changes address an issue with ProxQuant where the accuracy drops a lot during the first few iterations of the quantisation phase. This reduction is caused as a lot of the useful structure in the network is destroyed by $\boldsymbol{w}^b$ being suddenly clipped to $[-1,1]^p$ and a jump to a much higher effective learning rate, as a result of switching from SGD to Adam.

Quantisation Phase. Once the first phase of training is complete we proceed with the quantisation phase. In this phase we slowly move from a model with both real valued weights and binary activations to a model with binary weights and activations. We start from the best model from pretraining and increase the weight of the regularization function λ_t throughout this phase, according to a linear rate $\lambda_t = t \cdot \lambda_{rate}$. In general, completing the quantisation phase over a longer time frame with a lower rate λ_{rate} produces better results.

Fine Tuning. Before the last few epochs of training we project $\boldsymbol{w}^b$ onto the set of quantised values $\{-1,1\}^p$. We then set $\nabla \ell_{z_t}(\boldsymbol{w}_t^b) = 0$. This process is done in order to ensure all weights are binary and to fine-tune the real value parameters $\boldsymbol{w}^r$ to the final values of $\boldsymbol{w}^b$.

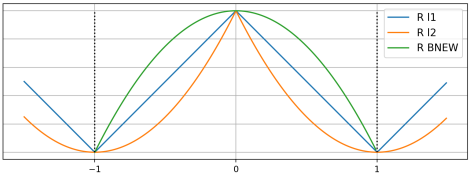

Figure 1: Different choices for regularisation function penalising non-binary weights.

4.4 Choice of Regularisation Function

Bai et al. [2] present two different choices for regularization functions, detailed in equation (8). While these are natural choices, and produce reasonable results, both functions have their largest gradients close to zero. Hence, as λ_t is increased, weights close to zero are quickly pushed in either the positive or negative direction. Moreover, once $\mathbb{E}_t[\nabla \ell_{z_t}(w_t^b)] \leq \mathbb{E}_t[\lambda_t \nabla R(w_t^b)]$ it becomes difficult for w^b to change sign for the rest of training. Figure 1 depicts our choice of regularisation function $R_{BNEW}(\boldsymbol{w}) \triangleq -\|\boldsymbol{w}^b\|^2 + p$ and those introduced in [2]. We suggest R_{BNEW} is a more appropriate choice for the following reasons. First, R_{BNEW} has small gradient close to zero. This means parameters close to zero do not experience as large an update towards -1 or $+1$ as parameters already close to these values. Hence, in general the parameters close to their quantised values are more incentivized to take binary values first, followed by those with smaller absolute values. The large gradient of R_{BNEW} close to -1 or $+1$ is designed to help prevent parameters oscillating around these values. Finally, and most importantly, we find that our R_{BNEW} produces better results in practice.

5 Experiments

Our experiments are split into two sections. In the first we investigate the performance of BNEW using a small BNN on CIFAR-10 and CIFAR-100 [15]. In the second section we show that BNEW scales to large BNN and data sets by training a ReActNet [20] on the ImageNet data set [6].

5.1 Small Scale Experiments

Setting. Much of the previous work on BNNs only present results for large over-parameterised models [1, 19, 20]. While these models may seem appealing due to the smaller accuracy degradation from their real valued counterparts, smaller BNN architectures are preferable to run on embedded devices. In this section we provide results produced from training a small ResNet20 [10], on the CIFAR-10 and CIFAR-100 data sets. These data sets [15] contain 60,000 32x32 pixel RGB images split over 10 and 100 classes, respectively. We modify a ResNet20 to include a similar block structure to that of ReActNet, detailed in [20]. This results in a small effective BNN with $d = 0.28$M. We let the weights in the first and last layers of the network, all biases, parameters of batch norm layers, and bottleneck layers retain floating point values. This results in a model which is 95% binary and requires 12.5 times less memory to store compared to its floating point counterpart. The exact speed of this model would depend on the exact hardware used at inference time. Rastegari et al. [25] suggest up to a 50 times speed up is possible for a BNN on the CPU. For each data set we provide results with and without distillation. When using distillation the CE loss is replaced with loss measuring the Kullback–Leibler (KL) divergence loss between the student and teacher.

Method: We compare BNEW against the STE method, BMD and BOP as described in Sections 3.1, 3.3, 3.2, respectively. We select these methods as they are optimisation algorithms that can be used to train any BNN. For a fair comparison we exclude any methods that require modification of the network architecture, or that produce non-binary quantization schemes such as ALQ [24]. We use the following hyperparameters and strategies for all methods. We use a fixed batch size of 128 and epoch budget of and 1000. We use Adam with a linearly decaying learning rate schedule with $\eta_0 \in \{0.01, 0, 001\}$. For BOP we considered $\eta_0^r \in \{10^{-2}, 10^{-3}\}$, $\tau \in \{10^{-7}, 10^{-8}, 10^{-9}\}, \eta_0 = 10^{-4}$. We use a linear schedule for η^{Adam} and η_0. We find $\eta_0^{Adam} = 10^{-2}$ and $\tau = 10^{-8}$ are best in practice. For BMD we cross-validate $\lambda_{rate} \in \{1.003, 1.01, 1.03, 1.1, 1.3\}$. For BNEW we use $\lambda_{rate} = 0.01$ for all experiments. To ensure the $\boldsymbol{w}^b$ has binary values for BMD and BNEW , the real valued weights are fine tuned for the final 20 epochs, as described in Section 4.3. We include results for a baseline ResNet20 with real valued weights and activations for use as a reference, trained according to the scheme described in [10]. These models are additionally used as the teachers when performing distillation. We use the following hyperparameters and strategies for all methods All results are computed over five runs with different random seeds. We use the same pre-trained models with binary activations and real valued parameters as the starting point for all methods as described in Section 4.3. While the authors of BMD and BOP do not suggest this for their methods, we find it produces superior results. All images are centred and normalised per channel and are subject to random flips and crops during training. In the supplementary materials we include an ablation study, results for BNEW using the regularisation functions detailed in (8), and results for all methods using a shorter epoch budget.

Table 1: Accuracies on CIFAR-10 and CIFAR-100 data sets.

Data Set	CIFAR-10		CIFAR-100	
Distillation	No	Yes	No	Yes
Real Valued	91.4σ0.3	-	67.1σ0.7	-
STE	84.3σ0.3	85.1σ0.4	55.0σ0.5	56.8σ0.3
BMD	84.0σ0.4	84.9σ0.3	54.8σ0.5	56.8σ0.5
BOP	84.5σ0.2	85.2σ0.4	55.3σ0.4	57.7σ0.4
BNEW	84.5σ0.3	85.1σ0.4	55.0σ0.3	57.5σ0.3

Results: The results of the CIFAR experiments are shown in Table 1. In general, the standard deviation values are fairly high at just under half a percentage. However, this is typical when training small binary models which are prone to bad local minima. The similarity in performance between optimisers is typically less than the standard deviation, which makes it hard to say anything conclusive about which method is best. All techniques result in an accuracy degradation of roughly 6% on the CIFAR-10 and between 9.5-10.5% on CIFAR-100. The largest difference in performance between optimisers is on the CIFAR-100 data set in combination with distillation, where BOP and BNEW perform significantly better. For all methods a shorter quantisation phase and lack of distillation produces a decrease in the quality of results. Hence, when the computation budget allows, we recommend training for longer and using distillation to produce a stronger binary model, regardless of the optimiser used. Figure 2 shows training curves generated by BNEW during the quantisation phase. The full results of the ablation study are presented in the supplementary material. However, here we note that the results are mostly in line with the findings of [3], and while many of the developments of ReActNet are useful we find the use of Equation (2) over (1) and addition of channel-wise scalars have no positive effect. The second result is not surprising given the inclusion of batch-norm layers [13].

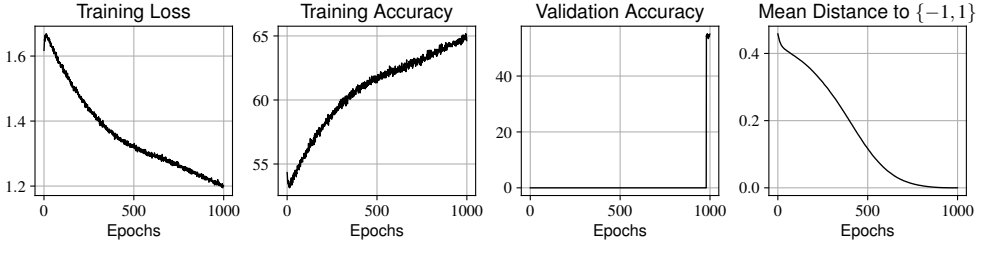

Figure 2: Training curves produced by BNEW on the CIFAR-100 data set. Note, the validation accuracy is only evaluated for the fine training phase.

5.2 ImageNet Experiments

Setting and Method. The ImageNet data set [6] contains 1.2M large RGB images of various sizes split over 1000 classes. On this data set we train a ReActNet-A [20], and

Table 2: Accuracies on ImageNet data set.

ARCHITECTURE	DISTILLATION	BACKBONE	OPTIMISER	ACCURACY
XNORNET[25]	NO	RESNET18	STE	51.2
BIREALNET[19]	NO	RESNET18	STE	56.4
BIREALNET	NO	RESNET18	BOP[11]	56.6
BIREALNET	NO	RESNET18	2^{nd} ORDER BOP [27]	57.2
BIREALNET	YES	RESNET18	BMD[1]	62.8
REAL-TO-BIN[21]	YES	RESNET18	STE	65.4
REACTNET[20]	YES	RESNET18	STE	65.5
REACTNET[20]	YES	MOBLIENET	STE	69.4
REACTNET	YES	MOBLIENET	BNEW	69.7

use their data augmentation scheme. The ground truth labels for ImageNet are not freely available and hence we report validation scores instead. We follow the training methodology of [20], however, in phase two of training we use BNEW rather then the STE to convert the parameters within $\boldsymbol{w}^b$ from real values to binary values. We use $\lambda_{rate} = 0.01$ $\eta_0 = 10^{-3}, \text{epochs} = 500, \text{epoch}_{freeze} = 400$. We restart the linear decay of the η for the fine tuning phase. We do not change any other hyperparameters except the batch size which we reduce to 220 due to hardware constraints. We compare against other BNN models training on ImageNet with comparable compute budgets [20]

Results. BNEW achieves an accuracy of 69.7% which is 0.3% higher than the state of the art performance of [20]. This is a promising result, given the lack of tuning for BNEW, and the fact that their network was designed to maximise the result obtained training with the STE method. This shows our simple approach for training BNN easily scales to large models and produces strong results in this setting.

6 Conclusion

In this work we have introduced BNEW, an effective and simple method for training neural networks with both binary weights and activations. To our knowledge, we are the first to show that a continuation method is effective in this setting. BNEW can easily be applied to a wide range of models given its simplicity. Additionally, BNEW has the following advantageous qualities i) strong empirical results; ii) the gradient is evaluated at the current iterate and hence does not require auxiliary parameters during training; and iii) a strong theoretical justification. BNEW also has two main weaknesses. First, the full training procedure must be completed up to the fine tuning phase before one has any indication of the final performance. Second, BNEW's effectiveness has a dependence on the hyperparamater λ_{rate} which needs to be selected to ensure $\boldsymbol{w}^b$ becomes mostly binary before the fine tuning phase. However, this issue can be circumvented by continuing training until this is so. In this work we have only shown results for binary quantisation, however, it would be trivial to extend BNEW to other quantisation schemes. This would be achieved by modifying the regularisation function R_{BNEW}, however, we leave this to future work. We hope this paper provides an alternative direction for BNN optimisation research alongside the STE and its derivatives.

References

[1] Thalaiyasingam Ajanthan, Kartik Gupta, Philip Torr, Richad Hartley, and Puneet Dokania. Mirror descent view for neural network quantization. *International Conference on Artificial Intelligence and Statistics*, 2021.

[2] Yu Bai, Yu-Xiang Wang, and Edo Liberty. Proxquant: Quantized neural networks via proximal operators. *International Conference on Learning Representations*, 2019.

[3] Joseph Bethge, Haojin Yang, Marvin Bornstein, and Christoph Meinel. Back to simplicity: How to train accurate bnns from scratch? *arXiv preprint arXiv:1906.08637*, 2019.

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Neural Information Processing Systems*, 2020.

[5] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Neural Information Processing Systems*, 2015.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *Conference on Computer Vision and Pattern Recognition*, 2009.

[7] Lei Deng, Peng Jiao, Jing Pei, Zhenzhi Wu, and Guoqi Li. Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. *Conference on Computer Vision and Pattern Recognition*, 2018.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

[9] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *Conference on Computer Vision and Pattern Recognition*, 2021.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Conference on Computer Vision and Pattern Recognition*, 2016.

[11] Koen Helwegen, James Widdicombe, Lukas Geiger, Zechun Liu, Kwang-Ting Cheng, and Roeland Nusselder. Latent weights do not exist: Rethinking binarized neural network optimization. *Neural Information Processing Systems*, 2019.

[12] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Neural Information Processing Systems*, 2016.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*, 2015.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[15] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Technical Report*, 2009.

[16] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *Conference on Computer Vision and Pattern Recognition*, 2016.

[17] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. *Neural Information Processing Systems*, 2017.

[18] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. *European Conference on Computer Vision*, 2018.

[19] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. *European Conference on Computer Vision*, 2018.

[20] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. *European Conference on Computer Vision*, 2020.

[21] Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. *International Conference on Learning Representations*, 2020.

[22] James O' Neill. An overview of neural network compression. *arXiv preprint arXiv:2006.03669*, 2020.

[23] A.S. Nemirovsky, D.B. Yudin, and E.R. Dawson. *Problem Complexity and Method Efficiency in Optimization*. 1983.

[24] Zhongnan Qu, Zimu Zhou, Yun Cheng, and Lothar Thiele. Adaptive loss-aware quantization for multi-bit networks. *Conference on Computer Vision and Pattern Recognition*, 2020.

[25] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *European Conference on Computer Vision*, 2016.

[26] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint*, 2019.

[27] Cuauhtemoc Daniel Suarez-Ramirez, Miguel Gonzalez-Mendoza, Leonardo Chang, Gilberto Ochoa-Ruiz, and Mario Alberto Duran-Vega. A bop and beyond: a second order optimizer for binarized neural networks. *LatinX in CV Research Workshop at CVPR*, 2021.

[28] Diwen Wan, Fumin Shen, Li Liu, Fan Zhu, Jie Qin, Ling Shao, and Heng Tao Shen. Tbn: Convolutional neural network with ternary inputs and binary weights. *European Conference on Computer Vision*, 2018.

[29] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[30] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *International Conference on Learning Representations*, 2017.