HadaNets: Flexible Quantization Strategies for Neural Networks

Yash Akhauri
Birla Institute of Technology and Science, Pilani
India, RJ 333031
f2016142@pilani.bits-pilani.ac.in

Abstract

On-board processing elements on UAVs are currently inadequate for training and inference of Deep Neural Networks. This is largely due to the energy consumption of memory accesses in such a network. HadaNets introduce a flexible train-from-scratch tensor quantization scheme by pairing a full precision tensor to a binary tensor in the form of a Hadamard product. Unlike wider reduced precision neural network models, we preserve the train-time parameter count, thus out-performing XNOR-Nets without a train-time memory penalty. Such training routines could see great utility in semi-supervised online learning tasks. Our method also offers advantages in model compression, as we reduce the model size of ResNet-18 by $7.43\times$ *with respect to a full precision model without utilizing any other compression techniques. We also demonstrate a 'Hadamard Binary Matrix Multiply' kernel, which delivers a 10-fold increase in performance over full precision matrix multiplication with a similarly optimized kernel.*

1. Introduction

Convolutional neural networks (CNNs) have achieved state-of-the-art results on several computer vision tasks [13, 25] and even demonstrate superhuman classification in some test-cases [17]. Such models are often the result of ensemble learning [27] and behemoth neural networks [13]. At the other end, there is a huge demand for offline inference on embedded devices. Aligning with progress in this field, low power inference and training is an intriguing domain, drawing attention from corporations and researchers.

State-of-the-art Convolutional Neural Networks typically require large amounts of memory and computation. This is not entirely feasible for edge devices, with ResNet-152 [13] holding 60M parameters and 11 GFLOPs for a single forward pass on a single element batch. It is unrealistic to use such classifiers on current mobile devices.

Current attempts at reducing the memory footprint of neural networks focus primarily on compression techniques [12, 6, 5, 8] or bit-width reduction [7, 11, 9, 26, 28, 19]. Several of these techniques are not completely realizable on current hardware.

XNOR-Nets [23] have an exceptionally low memory footprint and energy requirement. Most multiply-accumulate operations are replaced by simple bit-manipulations [7]. Aside from sequential bit-packing for subsequent bit-operators (XNOR operation), binary neural networks are fully realizable on current hardware. XcelRAM [1] is a modified Von Neumann machine which enables binary convolutions, providing a $6.1\times$ energy saving for XNOR-Net inference. The scope for such hardware optimization further incentivizes the search for better quantization strategies for binary neural networks.
This paper makes the following contributions:

- We introduce a quantized neural network training strategy with flexible memory and energy requirements. HadaNets yield a higher accuracy than XNOR-Nets without increasing filter map counts, as opposed to WRPN Nets [22].
- HadaNets prove the be a more effective training strategy for quantized neural networks as both XNOR-Nets and HadaNets require us to maintain full-precision parameters for gradient updates. Our training methodology performs at par with post-training quantization methodologies like ABC-Nets [21] ($\pm 0.5\%$ top-1 accuracy).
- We introduce Hadamard-Binary-Weight-Networks (HBWNs) (indicated by $\beta_a = 1$ in our tests). HBWNs outperform Binary-Weight-Networks [23] on the ResNet-18 topology by 1.5% in top-1 accuracy.
- We develop Hadamard binary matrix multiply CPU kernel which demonstrates a 10 fold increase in performance over its full precision counter-part.

2. Related Work

Aiming to reduce the computation and memory costs of current architectures, bit quantization and compression

techniques have been thoroughly explored in recent literature. We briefly review these works in this section.

2.1. Quantizing pre-trained models

DeepCompression [12] prunes unimportant connections, after which the remaining weights are quantized, followed by Huffman coding to compress the weight values. Incremental Network Quantization [3] involves weight partitioning, followed by group-wise quantization and re-training. ABC Networks [21] deliver a performance identical to its full precision counterparts given sufficient activation and weight bases. Methods such as Network Sketching [10] use full precision activations for forward passes, they do not exploit the massive memory and energy saving that is offered by bit-packing binary matrix elements.

2.2. Train-from-scratch quantization

The current standard (32 bit weights and activations) has proven to be unnecessary for high accuracy in computer vision. [11] and [9] demonstrate that 16 bit precision is sufficient to train most neural networks. Binary Neural Networks [7] have been proven trainable for small data-sets, with XNOR-Nets [23] converging for the ImageNet dataset. Mishra *et al*. [22] demonstrate that the accuracy loss by binarization can be prevented by increasing the filter map count in each layer, this increases the parameter count quadratically [4]. This raises questions about the efficiency of this approach. Our research differs from these as we do not increase the train-time parameter count or the width of the convolutional filters, thus making our method less resource intensive to train.

3. HadaNets

HadaNets extend the binarization scheme used in XNOR-Nets [23] by converting an input tensor to the hadamard product of a full precision tensor and a binary tensor. The binary tensor is simply the $Sign$ of the input tensor. The full precision tensor holds the mean of the absolute value of segments of the input tensor. The segment size is decided by β. This is depicted in Figure 1.
We refer to the input tensor at the l^{th} layer in an L-layer CNN architecture with $A_{l(l=1,..,L)}$. We refer to the k^{th} weight filter in the l^{th} layer as $W_{lk(k=1,...,K^l)}$. K^l is the number of weight filters in the l^{th} layer of the CNN. $A_l \in \mathbb{R}^{c\times w_{in}\times h_{in}}$ where (c, w_{in}, h_{in}) represents *channels, width* and *height* respectively. $W_l \in \mathbb{R}^{k\times c\times w\times h}$ $s.t.$ $w \leq w_{in}, h \leq h_{in}$. The 'hadamard binarized' tensors as calculated by Eqn (2) and Eqn (4) in Section 3.1.1 and 3.1.2 for W_l and A_l respectively will be referenced as $\widetilde{W_l}$ and $\widetilde{A_l}$.

Algorithm 1 Training a L-Layer HadaNet

Input: (Input, Target): (I, T), Cost function: $C(T, \widehat{T})$, Current LR: η^t, set $\beta_{a,l}$ and $\beta_{w,l}$.
Output: Updated weight W^{t+1} and learning rate η^{t+1}

for l = 1 to L **do**
  S_l = **Shape**(W_l)
  X_l = **Reshape**(W_l, shape=$(S_l[0], -1)$) // *Reshape with -1 infers the second dimension from the length of the array and remaining dimensions.*
  $\widetilde{X_l}$ = **Binarize**(X_l, β_w) // *Binarize as Eqn (2)*
  $\widetilde{W_l}$ = **Reshape**(X_l, shape= S_l)
$\widehat{T}$ = **HadaForward**(I, $\widetilde{W}$) // *Standard forward propagation with activations binarized as Eqn (4)*
$\frac{\delta C}{\delta \widetilde{W}}$ = **HadaBackward**($\frac{\delta C}{\delta \widetilde{W}}$, $\widetilde{W}$) // *Standard backward propagation, gradients are calculated using* $\widetilde{W}$.
W^{t+1} = **UpdateParameters**(W^t, $\frac{\delta C}{\delta W}$, η_t)
η_{t+1} = **UpdateLearningRate**(η^t, t)

3.1. Hadamard binarization of tensors

We introduce two hyper-parameters per layer in a neural network. We refer to these as $\beta_{w,l}$ and $\beta_{a,l}$ where l is the l^{th} layer in a L-Layer CNN and suggest $\beta_{w,l\in\{0,L\}} = \beta_{a,l\in\{0,L\}} = 1$ for standard architectures. Figure 1 depicts the process of binarization and the role of $\beta_{a,l}$ and $\beta_{w,l}$. β is also referred to as the binarization aggression in this paper. In all our experiments, $\beta_{w,l\in\{0,L\}} = \beta_{a,l\in\{0,L\}} = 1$. If $'l'$ is not sub-scripted for β_w or β_a, we take $\beta_{w,l\in[1,L-1]} = \beta_w$ and $\beta_{a,l\in[1,L-1]} = \beta_a$ in a L-Layer CNN.
$\odot$ denotes Hadamard product of two tensors.
$\delta \in \mathbb{R}^{layer\times channel\times column\times row}$, and is indexed as $\delta_{l,x,y,z}$.

3.1.1 Weights

We compute $S = shape(W_l)$ and we refer to **Reshape**(W_l, shape=$(S[0], -1)$) as W^f.

$$\delta_{l,x,y,z} = \frac{1}{\beta_w}\sum_{r=0}^{\beta_w}\left|W^f_{(x,r+\beta_w*\lfloor\frac{(w-1)z+y}{\beta_w}\rfloor)}\right| \tag{1}$$

$$\widetilde{W_l} = \delta_l \odot Sign(W_l) \tag{2}$$

In (1), $x \in (0, c), y \in (0, w), z \in (0, h)$.
In most CNNs, filter sizes for convolution rarely exceed 11×11. Row-major Hadamard Binarization of the actual filter kernels will not benefit with $\beta_w > w_{kernel}$, preventing us from achieving maximum memory saving. Hence we reshape our convolution weight filters as $W^f = Reshape(W_l, shape = (shape(W_l[0], -1))$ for the binarization process.

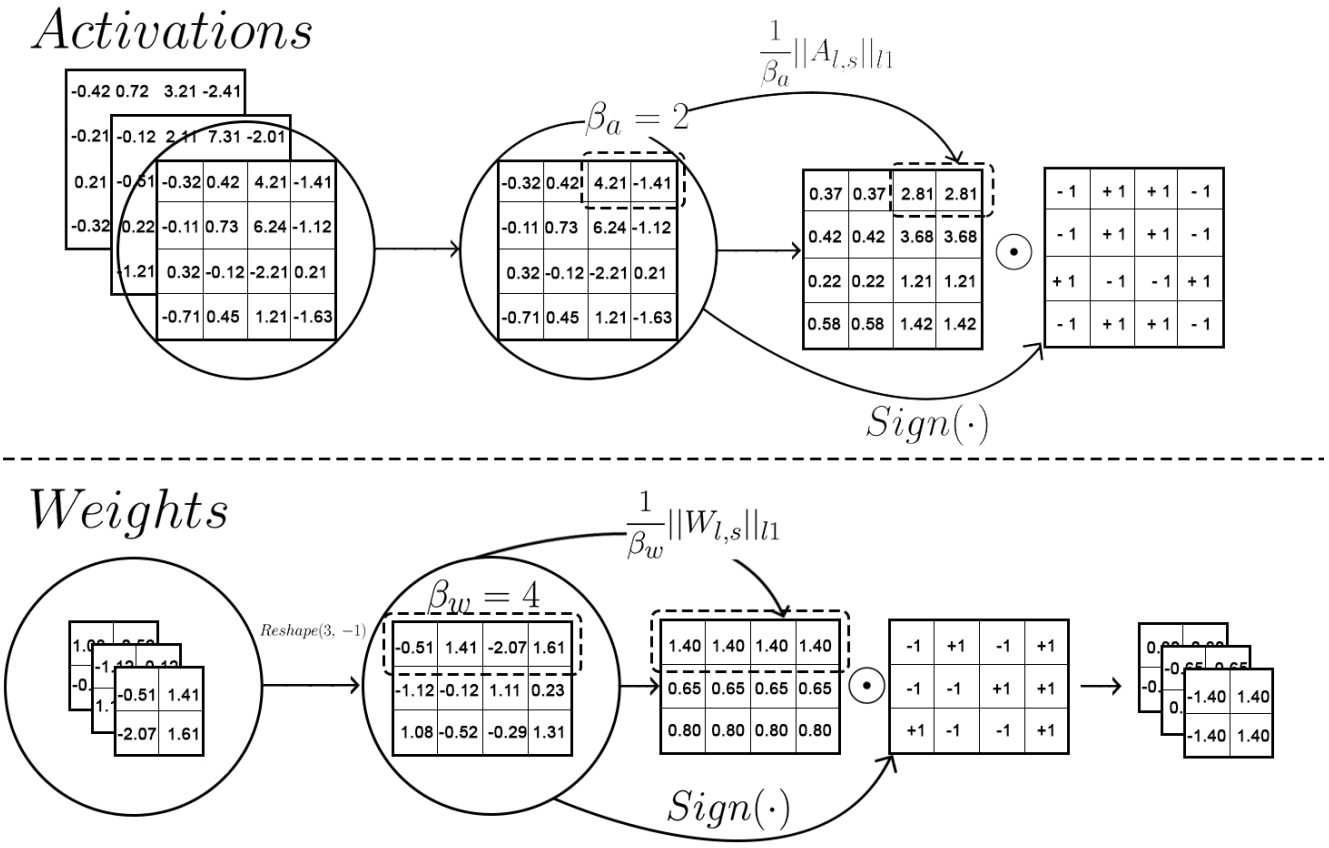

Figure 1. The figure depicts the 'hadamard' binarization of tensors in the CNN. We do not store repeated full precision values and compress the binary matrix by bit-packing it to available data-types. $A_{l,s}$ and $W_{l,s}$ refer to sub-matrices of size $(\beta \times 1)$ extracted from the full precision matrix of layer 'l' from an L-layer CNN. In our figure, we only show the Hadamard Binarization of one channel for the activations.

3.1.2 Activations

$$\delta_{l,x,y,z} = \frac{1}{\beta_a} \sum_{r=0}^{\beta_a} \left| A_{l,(x,r+\beta_a * \lfloor \frac{y}{\beta_a} \rfloor, z)} \right| \tag{3}$$

$$\widetilde{A}_l = \delta_l \odot Sign(A_l) \tag{4}$$

In Eqn (3), $x \in (0, c), y \in (0, w_{in}), z \in (0, h_{in})$ and $\beta_a \leq w_{in}$. We utilize the binary weight estimation derived in [23]. The optimal binary estimation of a weight filter is simply $\frac{1}{n}||W||_{l1} \odot Sign(W)$. By segmenting W with windows of size $(1 \times \beta_w)$ row-wise, a better estimate of a tensor can be developed. Alexander and Cory [2] empirically demonstrate the Angle Preservation Property of XNOR type tensor binarization. Upon studying the angle (α) between a random vector (from a standard normal distribution) and its hadamard binarized version for different β, we observe that as β decreased, α diminishes. This is demonstrated in Figure 3.

3.1.3 Propagating gradients

The derivative of the $Sign$ function is zero almost everywhere. To preserve the gradient information and cancel the gradient when the input (x) is too large, we take a straight-through estimator to obtain the derivative of the $Sign$ function (5) [7].

$$\frac{\partial Sign(x)}{\partial x} = \begin{cases} 1_{|x| \leq 1} \\ 0_{|x|>1} \end{cases} \tag{5}$$

To propagate the gradients through the hadamard binarization function, we assume that the incoming gradient from the $(l+1)^{th}$ layer from a L-layer neural network is $\frac{\partial C}{\partial \widetilde{W}_l}$. We require $\frac{\partial C}{\partial W_l}$, as given in (6). For simplicity, we take $W \in \mathbb{R}^n$. Note that $\delta_{l,i}$ references the absolute value of the hadamard binarized weight at the i^{th} index of the l^{th} layer.

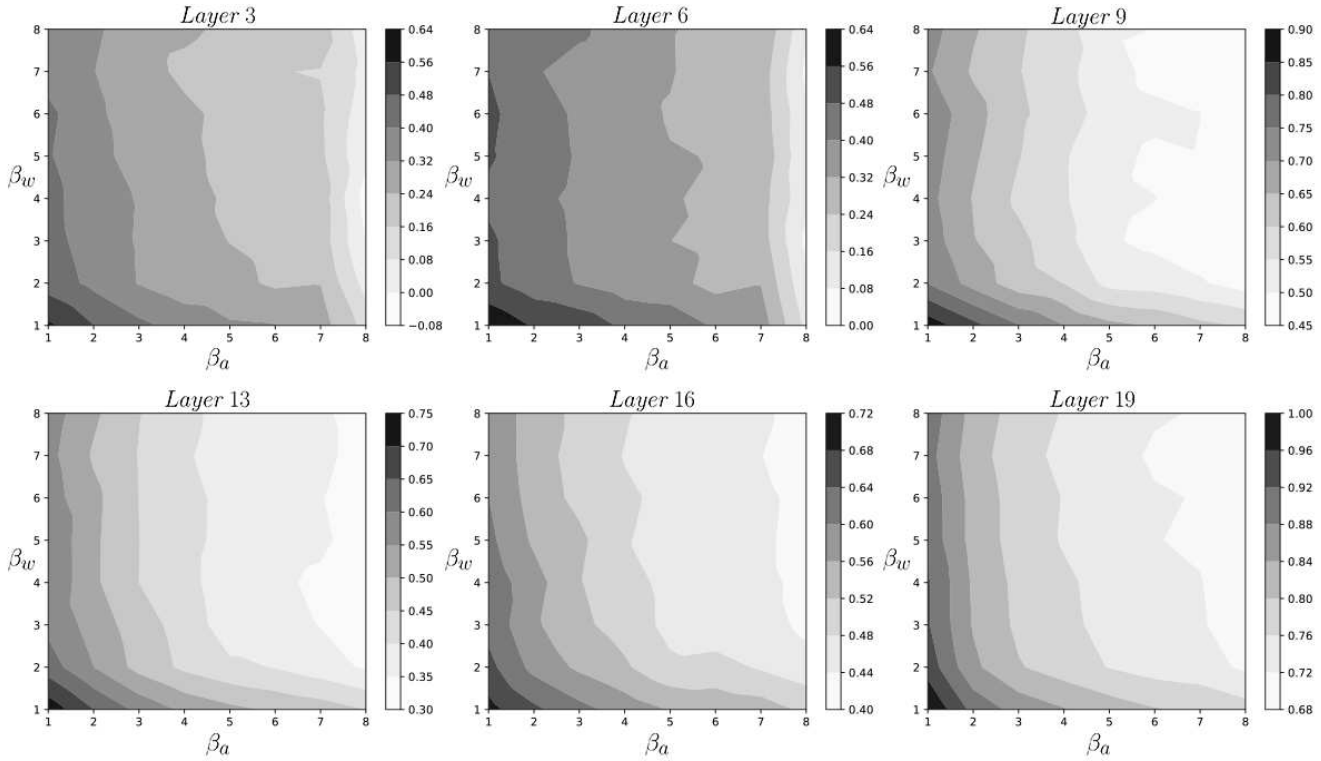

Figure 2. We study the Dot-Product Preservation property for the AlexNet-inspired architecture trained on the CIFAR-10 data-set. The filled contour maps reveal the Pearson's correlation coefficient between the activations in the HadaNet as β_w and β_a vary through different layers of the network. $\beta_w, \beta_a \in \mathbb{Z}^+$

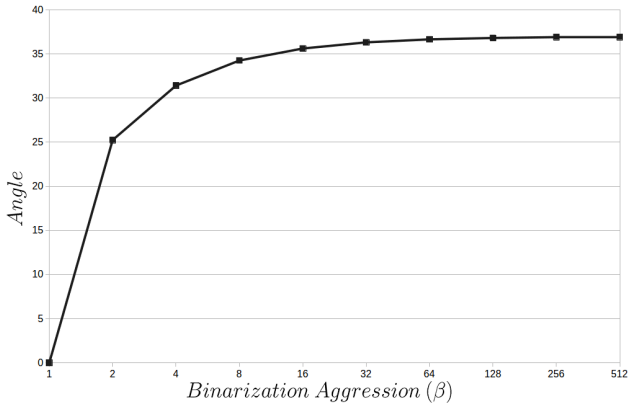

Figure 3. The angle between a random vector and its hadamard binarized vector with respect to the binarization aggression (β).

We refer to $\beta_{w,l}$ as β_w in Eqn (6).

$$\frac{\partial C}{\partial W_{l,i}} = \frac{1}{\beta_w} Sign(W_{l,i}) \sum_{j=\beta_w \lfloor \frac{i}{\beta_w} \rfloor}^{\beta_w(\lfloor \frac{i}{\beta_w} \rfloor + 1)} \frac{\partial C}{\partial \widetilde{W}_{l,j}} (Sign(W_{l,j})) + \delta_{l,i} \frac{\partial C}{\partial \widetilde{W}_{l,i}} \frac{\partial (Sign(W_{l,i}))}{\partial W_{l,i}} \tag{6}$$

3.2. Binarization aggression

Deciding the values for $\beta_{w,l}$ and $\beta_{a,l}$ in HadaNets is of great importance while making architectural decisions. Extending the Dot-Product preservation study done by Alexander *et al*. [2], we discover that the activations for HadaNet with varying ($\beta_{w,l}$, $\beta_{a,l}$) and its Full Precision variant for the same architecture are highly correlated.

In Figure 2 we train a neural network (AlexNet-inspired architecture) with $\beta_a = 8$ and $\beta_w = 8$ over the CIFAR-10 data-set. We use this network to find the Pearson's correlation coefficient between the activations for the Network Variant ($\beta_a = 1, \beta_w = 1$) and activations for Network Variants with β_a and β_w independently varying from 1 to 8. We restrict our β to 8 because the activation maps inside this neural network topology have a minimal activation matrix size of 8×8. It is evident from Figure 2 that changing β_w for a given β_a has little effect on the Pearson's correlation coefficient. Hence we can binarize the weights more aggressively than we binarize the activations. This makes the model more compressible, as the Hadamard binarization of model weights reduce the size of the model $\sim \frac{\beta_w X}{\beta_w + X}$ times. Here X refers to the bit-width of the available data type to bit-pack the elements of the 'binary' tensor (± 1) to.

We can also draw from Figure 3 that the primary benefit of

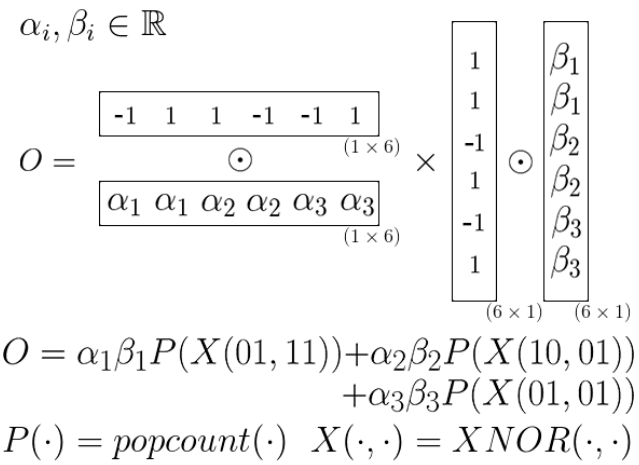

Figure 4. The figure above demonstrates the hadamard multiplication (xHBNN) of two vectors. Here $\beta_a = \beta_w = 2$

Hadamard Binarization can be observed if $\beta \leq 8$, as the angle (α) is seen to saturate to approximately 37 degrees for $\beta > 8$. This could also explain why the Binary-Weight-Network [23] performs significantly better than XNOR-Nets on the ImageNet data-set (A Binary-Weight-Network for the ResNet-18 architecture achieved 60.8% top-1 accuracy, whereas an XNOR-Network had a top-1 accuracy of 51.2%).

We also observe that for our AlexNet-inspired architecture, changing the β_w for the 3^{rd} and 4^{th} convolutional layers causes negligible difference ($\pm 0.5\%$) in CIFAR-10 accuracy. We tested this for $\beta_{w,l\in\{3,4\}} = \{16, 32, 64\}$. We observe that we could binarize the 2^{nd} convolution with $\beta_w = 32$ for Le-Net architecture for the MNIST data-set with no degradation in accuracy ($\pm 0.1\%$). With the above, we reason that we can yield even greater memory savings for very large scale image recognition models by aggressive layer-specific binarization with little or no loss in accuracy. We discuss the different β configurations we tested over standard data-sets in Section 5 of this paper.

4. Efficiency analysis

A majority of current hardware implementations are variants of Von Neumann machines [1]. In such machines, the memory and computation blocks are separate. This is a bottleneck as CNNs require constant data transfer between the memory and computational blocks. A single 32 bit DRAM memory access takes 640pJ in 45nm CMOS technology, whereas a floating point add consumes 0.9pJ [12]. In this regard, binary neural networks are very memory friendly. It is possible, and 128× more power efficient to store some models in the SRAM cache of the device, where a 32 bit SRAM memory access takes 5pJ. Courbariaux *et al.* [7] states that due to binary convolution kernel repetitions, the dedicated hardware time complexity is reduced by 60%. While this is true, kernel repetitions restrict the model accuracy, in the same way increasing kernel depth in a Binary Weight Neural network might be futile because of the limited number of filter kernels possible ($2^{n\times m}$) for a filter of size $(n \times m)$. HadaNets are less susceptible to learning redundant filters as we have greater degree of freedom (A full precision tensor instead of a scalar tied to a binary tensor).

Network	Memory Saving
AlexNet	6.51×
ResNet-18	7.43×

Table 1. Reduction in model size for $\beta_w = 16$ with respect to the full precision model.

The energy spent to move data from SRAM to register or register to register increases with the length of the word size. In our networks, the smaller the β, the higher the network accuracy. It can therefore be beneficial to bit-pack the binary matrix resulting from Hadamard Binarization to smaller data-types (8-bits). As an example, from Figure 4 we can see that it is possible to bit-pack the row vector $'-1\ 1\ 1\ -1\ -1\ 1'$ as 011001 to a 6-bit data-type. To compute O, we can get to the relevant part of the binary matrix by doing bit-shifts. Packing the binary matrices to a smaller data-type not only reduces the energy consumption in moving the data, but also reduces the amount of bit-shifts required to obtain the relevant part of the binary matrix for bit-manipulations.

A quadratic increase in parameters for ABC-Nets [21] makes maintaining full-precision parameters during training costly. HadaNets side step the issue of a quadratic rise in parameter count, and successfully reduce the train time and cost. The issue of creating too many activation/weight bases with high correlation in the ABC Nets is dealt with by l_2 regularization. This inherent problem with ABC-Nets does not exist in HadaNets.

In all our experiments and tests reported, we do row-wise binarization for all β_w and β_a variants. We assume that all tensors are stored in a row-major order. We avoid segmenting tensors into tensor blocks as this will increase the memory access times significantly irrespective of row/column major storage.

4.1. Memory

As discussed earlier, the memory required for a HadaNet model is $\sim \frac{\beta_w X}{\beta_w + X}$ times lesser than its full precision counterpart. For a FP32 model, we shall assume that the binary elements are bit-packed to an unsigned long int, making $X = 32$. Memory accesses cost significantly more energy than arithmetic operations in a neural network. Table 1 de-

tails how aggressive the memory saving is for ResNet-18 and AlexNet. For both models, we shall take $\beta_w = 16$, and keep $\beta_{w,1} = \beta_{w,L} = 1$. Rastegari *et al*. [23] found that the scaling factor for weights is more important than the scaling factor for the activations. We suspect that this observation for an XNOR-Net is valid because the weight tensors compensate for the absence of the activation scaling factor at train-time. A Binary-Weight-Network, where no binarization is done on the activations outperform XNOR networks. We also found that β_a played a more important role in accuracy than β_w, with $\beta_a = 2$ and $\beta_w = 16$ ($\beta_w \geq \beta_a$) outperforming models where $\beta_w \leq \beta_a$ on the CIFAR-10 data-set. We also conducted tests on the ResNet-18 keeping $\beta_a = 1$ and $\beta_w = 4$ and observe that the network outperformed the Binary-Weight-Network by 2.5% in top-1 accuracy. These observations pave way for greater binarization aggression for weights, which further reduce the memory of trained networks. It has been observed that at train-time, storing the activation maps for mini-batches creates a larger memory footprint than the weights do. This has not been effectively resolved by HadaNets.

4.2. CMMA vs xHBNN

Figure 5 benchmarks the Classical Matrix Multiply Algorithm with our Hadamard Binary Matrix Multiplication kernel. We keep the $\beta = 16$ while multiplying the matrices. We used the Intel® Xeon® Gold 6128 processor clocked at 3.40 GHz, a 19.25 MB cache, and 24 cores with two-way Intel® Hyper-Threading Technology in our benchmark. We observe an approximate $10\times$ speed up when using the xHBNN kernel for matrix multiplication.

As discussed in [7], we can concatenate groups of β binary variables into available data types, and replace β multiplication and $\beta - 1$ sum operations with two multiplication and simple bit-wise operations.

To compute the product of two vectors W and $A \in \mathbb{R}^K$ where W_f, $A_f \in \mathbb{R}^{\lceil \frac{K}{\beta} \rceil}$ and $W_b, A_b \in \{-1, +1\}^{\lceil \frac{K}{\beta} \rceil}$, the following operations shall be necessary:

$$O = \sum_{j=0}^{\lceil \frac{K}{\beta} \rceil} W_{f,j} \times A_{f,j} \times popcount(xnor(W_{b,j}, A_{b,j})) \tag{7}$$

It is important to note that Eqn (7) refers to W_b, W_f, A_b, A_f after the 'paired' full precision and binary vector approximations of W and A have been compressed. The binary vector is bit-packed to available data types, and the full precision vector can be compressed β times. For brevity, we assume that there is a data-type with bit-width $= \beta$ available, thus compressing the binary vector to a size of $\left\lceil \frac{K}{\beta} \right\rceil$. This method of multiplication has been demonstrated in Figure 4.

It is evident that if $\beta_a \leq \beta_w$, the inference speed of the network will be decided by β_a. This places a potential bottleneck in computation time. We reason that since a parallel reduction approach computes the summation or multiplication of N-1 numbers in $log_2(N)$ steps; changing β_a from 4 to 16 would not give significant speedups and will cause accuracy deterioration. For a 45nm CMOS technology a 64-bit memory access from an 8K cache takes about 10pJ [14], whereas a 32-bit FMUL operation takes 3.7pJ. Typical SRAM access latency is around 2-3 ns, whereas DRAM access latency is approximately 20-35 ns. We reason that it is a better pursuit to reduce the size of the model instead of the number of arithmetic operations. The primary speed up will be realized by placing model weights in the cache. As the batch-size increases, the memory footprint of activations increase. It was found in [22] that about 96.5% of the memory footprint for the ResNet-101 at inference with a batch-size of 1 was due to weights. Thus the memory saving during inference for most real-time scenarios will primarily come from keeping β_w low.

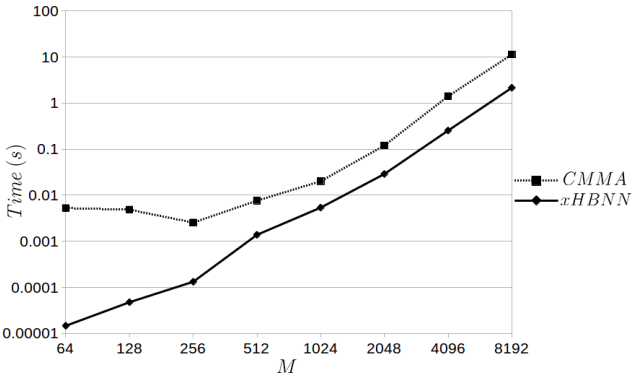

Figure 5. Bench-marking CMMA (Classical matrix multiply algorithm) with the Hadamard Binary Matrix Multiplication (xHBNN) kernel. The M denotes the size of a square matrix $(M \times M)$. In this figure, $\beta = 16$.

5. Experiments

In this section we evaluate the performance of HadaNet variants with respect to full precision neural networks, XNOR-Nets [23] and ABC Nets [21]. We train our models on the MNIST [16], CIFAR-10 [15] and the ImageNet (ILSVRC2012) [24] data-sets. XNOR-Net [23] is a neural network where both the filters and input to convolution layers are binary, with a full precision scalar scaling factor for the activations and weight tensors. HadaNet with $\beta_{a,l} = nElement(A_l)$ and $\beta_{w,l} = nElement(W_l)$ (where $nElement$ stands for the number of elements in the weight or activation tensor) is essentially the XNOR-Net. ABC Nets approximate full-precision weights as a linear combination of multiple binary weight bases. We report results for ABC Nets with a 5-weight base, 1-activation base. HadaNet

Data-set		MNIST	CIFAR-10	
Network Variant	Memory	LeNet	NIN	AlexNet-inspired
Full-Precision	$1\times$	99.39%	89.59%	89.36%
HBWN ($\beta_w = 4; \beta_a = 1$)	$\sim .28\times$	99.41%	89.33%	89.24%
HadaNet ($\beta_w = 4; \beta_a = 4$)	$\sim .28\times$	99.43%	87.33%	88.64%
ABC Net ($b_w = 5; b_a = 1$)	$\sim .16\times$	99.41%	-	88.69%
HBWN ($\beta_w = 8; \beta_a = 1$)	$\sim .16\times$	99.41%	89.11%	89.04%
HadaNet ($\beta_w = 16; \beta_a = 2$)	$\sim .09\times$	99.40%	88.74%	89.02%
XNOR-Net	$\sim .03\times$	99.23%	86.28%	88.60%

Table 2. Classification test accuracy of CNNs trained on MNIST and CIFAR-10 with different network topologies.

ImageNet						
Network Topology	AlexNet			ResNet-18		
Network variant	Top-1	Top-5	Memory	Top-1	Top-5	Memory
Full-Precision	56.6%	80.2%	$1\times$	69.3%	89.2%	$1\times$
HBWN ($\beta_w = 4; \beta_a = 1$)	56.7%	80.1%	$\sim .32\times$	62.3%	84.4%	$\sim .31\times$
HadaNet ($\beta_w = 4; \beta_a = 4$)	46.3%	71.2%	$\sim .32\times$	53.3%	77.3%	$\sim .31\times$
ABC Net ($b_w = 5; b_a = 1$)	-	-	$\sim .19\times$	54.1%	78.1%	$\sim .17\times$
ABC Net ($b_w = 5; b_a = 3$)	-	-	$\sim .19\times$	62.5%	84.2%	$\sim .17\times$
HadaNet ($\beta_w = 16; \beta_a = 2$)	47.3%	73.3%	$\sim .13\times$	53.8%	77.2%	$\sim .12\times$
ABC Net ($b_w = 3; b_a = 1$)	-	-	$\sim .12\times$	49.1%	73.8%	$\sim .10\times$
BWN	56.8%	79.4%	$\sim .05\times$	60.8%	83.0%	$\sim .05\times$
XNOR-Net	44.2%	69.2%	$\sim .05\times$	51.2%	73.2%	$\sim .05\times$
BNN	27.9%	50.4%	$\sim .04\times$	42.2%	67.1%	$\sim .04\times$

Table 3. Classification test accuracy of CNNs trained on the ImageNet data-set with different network topologies. Note that the memory column only estimates model sizes, and does not describe the run-time memory overhead.

with $\beta_a = 2; \beta_w = 16$ has a comparable memory footprint. In all our experiments, the first and the last layer have $\beta_w = \beta_a = 1$. In all trials where $\beta_{w,l}, \beta_{a,l} \neq 1$, we use the following placement of layers: Batch Normalization - Binarization - Convolution - Activation - Pooling, as suggested by Rastegari *et al*. [23], the Convolution - Batch Normalization - Binarization - Pooling placement of layers reduced the top-1 accuracy of the XNOR-Network by approximately 14% [23].

5.1. MNIST

We train on the MNIST [16] data-set for 60 epochs and decay the learning rate by 0.1 every 15 epochs with an initial learning rate of 0.005. We minimize the Cross-Entropy Loss with the Adam optimizer. We train the LeNet topology and use Batch Normalization with a mini-batch of size 128. No data-augmentation is done.

5.2. CIFAR-10

We train two network topologies on the CIFAR-10 [15] data-set. The first is the Network-In-Network architecture [20]. The second is an AlexNet-inspired network with the following architecture:
$(2 \times 128C3) - MP2 - (2 \times 256C3) - MP2 - (2 \times 512C3) - MP2 - (2 \times 1024FC) - 10FC$
C3 is a batch normalized 3×3 ReLU convolution layer. $\beta_a = \beta_w = 1$ for the first and the last layer. We minimize the Cross-Entropy Loss with the Adam optimizer and an exponentially decaying learning rate. We train our neural network with a batch size of 128 for 60 epochs. We trained the ABC Net on the CIFAR-10 dataset as it was not reported in their paper. We do not use any data-augmentation techniques. We simply Normalize the data-set.

5.3. ImageNet

ImageNet (ILSVRC2012) is a benchmark image classification dataset [24] which consists of 1.2 million training images. There are 1000 categories.

AlexNet

The AlexNet network has 5 convolutional layers and 2 fully-connected layers. This network has 61 million parameters. We resize our ImageNet data-set to a size of 256×256 and take random-crops of size 227×227. We augment our data-set with random horizontal flips. We minimize the Cross-Entropy Loss with the ADAM optimizer and set our initial learning rate as 0.001. We train our network for 32 epochs, decaying the learning rate by 0.1 after

every 8 epochs.

ResNet-18

We evaluate the ResNet-18 [13] as done in [23]. We train for 60 epochs with a batch size of 256 and initial learning rate of 0.01, we decay our LR by 0.1 at epoch 30 and 40. We augment our data-set as done in the AlexNet training procedure but crop the input to 224×224.

6. Discussion

In most of our trials, we keep $\beta_a \leq \beta_w$. Increasing β_a significantly degrades the network accuracy. The primary issue with quantization of activations is that we need to introduce an approximation for the non-differentiable $Sign$ operator. The derivative is zero almost everywhere for the $Sign$ function. We use a linear approximation for the $Sign$ function as detailed in Eqn (5). This does not solve the gradient mismatch problem. We keep $\beta_a \in [1, 8]$ in our experiments. The benefits of Hadamard Binarization become less pronounced for $\beta_a > 4$, but is relatively flexible to changes in β_w.

Drawing from the findings in ABC Nets [21], we tested Average Pooling instead of Max Pooling over the CIFAR-10 data-set. This was because in their experiments, max-pooling returned a tensor with most elements equal to $+1$. We also used the PReLU activation function and tested the AlexNet-inspired network on the CIFAR-10 data-set. The PReLU activation function is initialized with $a_i = 0.25$. These tests did not yield any noticeable gain in performance for HadaNets.

ABC Nets [21] use pre-trained models to convert an optimization problem to a linear regression problem. The activation maps generated by ABC Nets occupy a larger memory footprint than HadaNets during inference. Utilizing 5 binary activation maps to represent a full precision activation map along with 5 binary weight maps to approximate a full precision weight map would generate a memory footprint close to that of its full-precision counterpart at inference. Unlike ABC Nets, HadaNets are networks that are trained from scratch. If a variant of ABC Net is trained from scratch, the memory overhead generated would be far greater than HadaNets. Training a full-precision model and converting the model to ABC Nets would also require more energy and time than a HadaNet.

Our current training methods utilize different forward and backward approximations, which gives rise to the gradient mismatch problem. Drawing from [18], it is possible to formulate our HadaNets training as a discretely constrained optimization problem and decouple the continuous parameters from the discrete constraints. Leng *et al*. [18] solve this problem using extra-gradient and iterative quantization algorithms. While HadaNets are compatible with this framework, implementing this was beyond the scope of this paper.

7. Conclusion

We introduce HadaNets, which utilize a new weight and activation binarization scheme. This method of binarization does not increase the parameter count of the neural network, and works with the hyper-parameters $\beta_{w,l}$ and $\beta_{a,l}$ that can be tuned to match hardware with a range of compute capabilities and memory constraints. We empirically justify the importance of β_a in accuracy and reinforce our claim by demonstrating the dot product preservation property and the Angle Preservation Property for HadaNets. We train several HadaNet variants and reported our results over the MNIST, CIFAR-10 and ImageNet data-set. We outperform the XNOR-Net and ABC Net over the MNIST and CIFAR-10 data-set. Our AlexNet network variant ($\beta_w = 16; \beta_a = 2$) out-performs the XNOR Net and give accuracy that is at par with full precision neural networks. Our Hadamard-Weight-Binary-Network outperforms Binary-Weight-Networks by 1.5% in top-1 accuracy for ResNet-18. We also demonstrate a Hadamard Binary Matrix Multiplication CPU kernel (xHBNN) which delivered a $10\times$ speed up over a similarly optimized Classical Matrix Multiplication CPU kernel (CMMA). Developing highly efficient quantized neural networks requires novel solutions to quantize weights, activations and gradients. Future work could involve using pre-trained networks to initialize HadaNets, testing more network variants on state-of-the-art network topologies. Gradient quantization methods for more efficient distributed training of neural networks should also be studied in greater detail.

8. Acknowledgements

The author would like to thank Dr. Surekha Bhanot and Sebastian Bodenstein for helpful discussions. The author would also like to thank Intel for providing access to Intel® Xeon® Scalable processors and Wolfram Research for their support.

References

[1] A. Agrawal, A. Jaiswal, B. Han, G. Srinivasan, and K. Roy. Xcel-ram: Accelerating binary neural networks in high-throughput sram compute arrays. *CoRR*, abs/1807.00343, 2018.

[2] A. G. Anderson and C. P. Berg. The high-dimensional geometry of binary neural networks. *CoRR*, abs/1705.07199, 2017.

[3] Y. G. L. X. Y. C. Aojun Zhou, Anbang Yao. Incremental network quantization: Towards lossless cnns with low-precision weights. In *International Conference on Learning Representations,ICLR2017*, 2017.

[4] R. Banner, I. Hubara, E. Hoffer, and D. Soudry. Scalable methods for 8-bit training of neural networks. *CoRR*, abs/1805.11046, 2018.

[5] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. *CoRR*, abs/1710.09282, 2017.

[6] Y. Choi, M. El-Khamy, and J. Lee. Universal deep neural network compression. *CoRR*, abs/1802.02271, 2018.

[7] M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[8] B. Dai, C. Zhu, and D. P. Wipf. Compressing neural networks using the variational information bottleneck. *CoRR*, abs/1802.10399, 2018.

[9] D. Das, N. Mellempudi, D. Mudigere, D. D. Kalamkar, S. Avancha, K. Banerjee, S. Sridharan, K. Vaidyanathan, B. Kaul, E. Georganas, A. Heinecke, P. Dubey, J. Corbal, N. Shustrov, R. Dubtsov, E. Fomenko, and V. O. Pirogov. Mixed precision training of convolutional neural networks using integer operations. *CoRR*, abs/1802.00930, 2018.

[10] Y. Guo, A. Yao, H. Zhao, and Y. Chen. Network sketching: Exploiting binary structure in deep cnns. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4040–4048, 2017.

[11] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1737–1746. JMLR.org, 2015.

[12] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[14] M. Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, Feb 2014.

[15] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research).

[16] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[17] K. Lee, J. Zung, P. Li, V. Jain, and H. S. Seung. Superhuman accuracy on the SNEMI3D connectomics challenge. *CoRR*, abs/1706.00120, 2017.

[18] C. Leng, H. Li, S. Zhu, and R. Jin. Extremely low bit neural network: Squeeze the last bit out with admm. *CoRR*, abs/1707.09870, 2018.

[19] F. Li and B. Liu. Ternary weight networks. *CoRR*, abs/1605.04711, 2016.

[20] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[21] X. Lin, C. Zhao, and W. Pan. Towards accurate binary convolutional neural network. In *NIPS*, 2017.

[22] A. K. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr. WRPN: wide reduced-precision networks. *CoRR*, abs/1709.01134, 2017.

[23] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks, 10 2016.

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[26] G. Venkatesh, E. Nurvitadhi, and D. Marr. Accelerating deep convolutional networks using low-precision and sparsity. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2861–2865, 2017.

[27] Z.-H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1):239 – 263, 2002.

[28] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *CoRR*, abs/1612.01064, 2016.