Binary Neural Networks

Ramteja Tadishetti, Daniel Sam Pete Thiyagu
College of Information and Computer Sciences
University of Massachusetts Amherst
{rtadishetti, dthiyagu}@cs.umass.edu

Abstract— **Network quantization compresses the original neural network network by reducing the number of bits required to represent each weight. This has the benefit of lowering memory constraints, reducing training time(since search space for weights is less). In this project we implement Binary Neural Networks in which the weights, activations are constrained to less number of bits when compared to general neural networks. So this project involves, designing a binary neural network, with an in-depth analysis of the networks and comparing it with other standard neural net architectures and evaluate their performance on the task of Image Classification.**

I. INTRODUCTION

Most of current neural network architectures designed for image related tasks like image classification, face recognition have multiple convolution layers, fully connected layers. The total number of parameters in the architectures designed for these tasks could be in order of millions.

As we continue to increase the number of layers for different specific tasks, the total number of parameters increase and finally increasing the memory foot print of the architecture. Also this becomes severe bottleneck for any real time applications designed based on computation, storage heavy architectures. Currently most of the applications developed on top of these tasks work in client-server model i.e requires endpoint to transfer the entire data to server and server completes the inference and gives back the result to client. This model has problem of wastage of network bandwidth and user information privacy. In order to avoid these problems endpoint should be able do the inference. Generally endpoints (mobile devices) have very limited compute power and memory resources. In order to fit the entire model on a mobile device, the memory foot print of the neural network has to be low. So we intend to solve this problem by reducing the memory consumption and computation of neural networks by using Binary Neural Networks.

II. RELATED WORK

As mentioned above most of the current DNN architectures are targeted in increasing the accuracy or performance. Efficient processing of Neural networks can be categorized into two; one trying to reduce the the precision of weight, operands, activations and other category which tries to reduce the number of operations by decreasing the layers by pruning or changing the architecture.

Apart from Binary Neural Networks which we explain in the next section, XNOR netsRastegari et al. [2016] was proposed for efficient approximation of standard convolution neural networks. DoReFa-NetZhou et al. [2016] is another method proposed for CNN's less number of bits are used for weights, activations and also for parameter gradients. HWGQ-NetCai et al. [2017] uses hardware efficient approximation methods during the forward propagation and backward propagation . All the above mentioned methods quantizes the weights uniformly. Apart from uniform spacing logarithmic quantization, methods have been proposed in which arithmetic operations are much simpler to execute due to logarithmic quantization.

The second category of neural networks target changing the network architectures to get comparable performance with decrease in overall computation. One of such method involves reducing the storage and computation required by neural networks by an order of magnitude without affecting their accuracy by learning only the important connectionsHan et al. [2015]. Apart from network pruning, some of the methods tried to exploit the sparsity in activations of layers for faster training and inference on hardware with acceptable decrease in accuracy Parashar et al. [2017].

III. PROBLEM STATEMENT

For the purpose of this project, we implement Binary Neural networks for Image Classification and make a comparative analysis of how much the accuracy varies, when the network is binarized. We are looking at different architectures on MNIST& CIFAR-10. We compare the same networks with and without binarization. As proposed in Hubara et al. [2016] we do a similar analysis to the network with and without binarization. Our evaluation metric is going to be Accuracy on the Test Set. This problem statement is an interesting area of research as we move into the future where devices with low memory can also use pre-trained quantized models for prediction since the memory requirements are reduced by a significant factor.

IV. TECHNICAL APPROACH AND MODEL

Binary Connect was proposed by Matthieu CourbariauxCourbariaux et al. [2015] in which weights are constrained to two values 1 & -1. By restricting the weights to values to 2 values , we are reducing the memory required by a factor of 16x. Since the weights are constrained to 1 , -1 the vector product of weights, activations is nothing but addition and subtraction of vectors which are computationally faster than multiplication.

Algorithm 1 SGD training with BinaryConnect. C is the cost function for minibatch and the functions binarize(w) and clip(w) specify how to binarize and clip weights.Lis the number of layers.

```
1: Require: a minibatch of inputs and targets, previous parameters weights $w_{t-1}$ and biases $b_{t-1}$ and learning rate $\eta$.
2: Ensure: updated parameters $w_t$ and $b_t$
3: 1. Forward Prop:
       $w_b \leftarrow binarize(w_{t-1})$
       For $k$ = 1 to $L$, compute $a_k$ using $a_{k-1}, w_b, b_{t-1}$
4: 2. Backward Prop:
       Initialize output layer's activation gradient $\frac{\partial C}{\partial a_L}$
       For k = $L$ to 2, compute $\frac{\partial C}{\partial a_{k-1}}$ using $\frac{\partial C}{\partial a_k}$, $w_b$
5: 3. Parameter Update:
       Compute $\frac{\partial C}{\partial w_b}$ and $\frac{\partial C}{\partial b_{t-1}}$ using $\frac{\partial C}{\partial a_k}$ and $a_{k-1}$
       $w_t \leftarrow clip(w_{t-1} - \eta \frac{\partial C}{\partial w_b})$
       $b_t \leftarrow b_{t-1} - \eta \frac{\partial C}{\partial b_{t-1}}$
```

As mentioned above binarization function maps real values to 1 , -1. There are two ways of performing this.

Deterministic Binarization:

$$x^b = sign(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

Stochastic Binarization:

$$x^b = sign(x) = \begin{cases} +1, & \text{with probability p } =\sigma(x) \\ -1, & \text{with probability 1-p} \end{cases} \tag{1}$$

$$\sigma(x) = clip(\frac{x+1}{2}, 0, 1) = max(0, min(1, \frac{x+1}{2}))$$

where x^b is the binarized variable of x, a real-valued variable. x could be either weight or activation. Also $\sigma(x)$ is called the hard sigmoid. Over here only the weights are restricted to (-1, 1) but still the activations and gradients are still real values. In parameter update instead of applying the update incrementally, we give the update and restrict the weights between the limits 1, -1 by clipping them.

Apart from restricting weights, we could also restrict activations which gets passed to the next layers as inputs. This is required for training on edge devices which have very low memory. This was proposed by Matthieu Courbariaux as an extension to Binary connect[add_ref]. The way we binarize the inputs still remains same as earlier apart from binarizing weights we also binarize the activations.

Since we are binarizing the weights, activations based on sign, gradient becomes zero incase of sign function.This makes it incompatible with backpropagation, as the exact gradient of the cost with respect to the quantities before the discretization (preactivations or weights) would be zero. Consider C as the cost function for minibatch and the sign function quantization $q = sign(r)$ and we assume that an estimator g_q of the gradient $\frac{\partial C}{\partial q}$ is obtained with the straight-through estimator as described in Algorithm 1 and referenced

Algorithm 2 Training a BNN. C is the cost function for minibatch, λ, the learning rate decay factor, and L, the number of layers. $\circ$ stands for element-wise multiplication.

```
1: Require: a minibatch of inputs and targets (a0, a*), previous weights W, previous BatchNorm parameters $\theta$, weight initialization coefficients from Glorot and Bengio [2010] $\gamma$,and previous learning rate $\eta$.
2: Ensure: updated weights $W^{t+1}$, updated BatchNorm parameters $\theta^{t+1}$ and updated learning rate $\eta^{t+1}$
3: 1.Computing Parameter Gradients:
4: 1.a Forward Prop:
5: for K = 1 to L do
6:     $W_k^b \leftarrow Binarize(W_k)$
7:     $s_k \leftarrow a_{k-1}^b W_k^b$
8:     $a_k \leftarrow BatchNorm(s_k, \theta_k)$
9:     if $k < L$ then
10:        $a_k^b \leftarrow Binarize(a_k)$
11: 1.b Backward Prop:
12: Compute$g_{a_L} = \frac{\partial C}{\partial a_L}$ knowing $a_L$and $a*$
13: Note that the gradients are not binary
14: for K = L to 1 do
15:     if $k < L$ then
16:        $g_{a_k} \leftarrow g_{a_k}^b \circ 1_{|a_k| \leq 1}$
17:     $(g_{s_k}, g_{\theta_k}) \leftarrow BackBatchNorm(g_{a_k}, s_k, \theta_k)$
18:     $g_{a_{k-1}}^b \leftarrow g_{s_k} W_k^b$
19:     $g_{W_b^k} \leftarrow g_{s_k}^T a_{k-1}^b$
20: 2. Accumulating Parameter Gradients:
21: for K = 1 to L do
22:     $\theta_k^{t+1} \leftarrow Update(\theta_k, \eta, g_{\theta_k})$
23:     $W_k^{t+1} \leftarrow Clip(Update(W_k, \gamma_k \eta, g_{W_k^b}), -1, 1)$
24:     $\eta^{t+1} \leftarrow \lambda \eta$
```

in Bengio et al. [2013]. Our straight-through estimator of $\frac{\partial C}{\partial r}$ is simply $g_r = g_q 1_{|r| \leq 1}$

With the above straight through estimator, notice that it removes gradient propagation for higher values of r else it preserves the gradient and propagates it.

The reasoning behind using the straight through estimator is as follows, not cancelling gradient for high values of r reduces performance. Consider the stochastic binarization equation in equation 1.

$\sigma(r) = \frac{HT(r)+1}{2}$

where HT(r) is the hard tanh function.

$$HT(r) = \begin{cases} +1, & r > 1, \\ r, & r \in [-1, 1] \\ -1, & r < 1 \end{cases} \tag{2}$$

So the input to next layer has the form,

$$W_b h_b(r) = W_b HT(r) + n(r)$$

where we use the fact that $HT(r)$ is the expectation over $h_b(x)$ using equations 1 and 2. n(r) is defined as binarization noise with mean equal to zero. Binarization noise component is ignored during the backprop since we expect the deterministic mean term HT(x) to dominate. Therefore, we replace $\frac{\partial h_b(r)}{\partial r}$ with

$$\frac{\partial HT(r)}{\partial x} = \begin{cases} 0, & r > 1, \\ 1, & r \in [-1, 1] \\ 0, & r < 1 \end{cases} \quad (3)$$

V. Experiments

We have implemented binary connect and binary neural networks in Pytorch[1]

A. Datasets

For the task of Image Classification we are using the dataset of MNIST, CIFAR 10. MNIST has images for 10 classes of numbers from 0-9. CIFAR10 has 10 classes of 'plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'. The dataset is split into 50k Training Samples and 10k test samples.

B. Preprocessing

The images are cropped to a 32*32*3 and then they are flipped randomly and also it is normalized.

C. Evaluation Metrics:

The evaluation metrics we use for our two classification tasks on the MNIST and CIFAR-10 are accuracy over time. Our baselines are the networks without binary weights or activations.

MNIST: **Architecture I**
We tried out two architectures, one with a simple fully connected layer, The input image is of size 28*28*1 which is unrolled into a single vector and then it is passed into a Fully connected layer of dimensions (784*10). We used loss as negative log likelihood loss. We tested it out with deterministic and stochastic binarization and at the end of 50 epochs, the results were as follows. The training set was 60000, and the test set was 10000 images. As you can see in the plot of training and test accuracies in Figures 1 and 2, the stochastic binarization scheme is much more smooth and resistant to changes. As seen in table I the drop is accuracy for the deterministic case is 14% and the stochastic was 12%.

TABLE I: MNIST - Arch I

	Train	Test
No binarization	93%	92%
Deterministic binarization of weights	79%	80%
Stochastic binarization of weights	81%	81%

Architecture II This is a three layered architecture, The input image is of size 28*28*1 which is unrolled into a single vector and then it is passed into a Fully connected layer of dimensions (N*784*4096). Then there is a batchnorm layer followed by an activation, Then it is passed into (L, BN, ACT) * 3. L is a Fully Connected layer with weights (N*4096*4096) and BN is batchnorm, ACT is an activation

[1]https://github.com/ramtejatadishetti/bnn

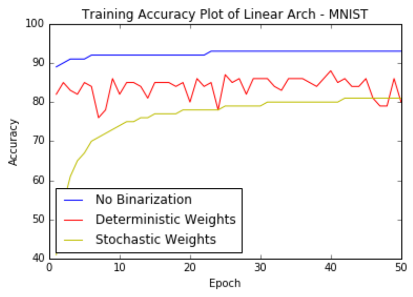

Fig. 1: MNIST Training accuracy - Arch I

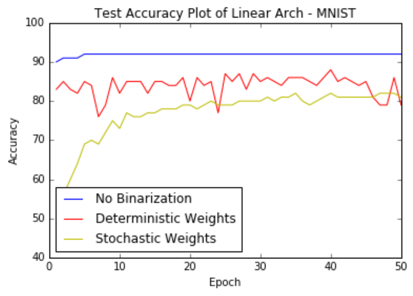

Fig. 2: MNIST Test accuracy - Arch I

function, we used the Tanh non linearity. This is the architecture used by [Courbariaux et al., 2014] for reporting their results. We used loss as negative log likelihood loss. We tested it out with deterministic and stochastic binarization as well as deterministic activations and at the end of 50 epochs, the results were as follows. The training set was 60000, and the test set was 10000 images. As you can see in the plot of training and test accuracies in Figures 3 and 4, the stochastic binarization scheme doesn't work well for this architecture, even though we tried hyperparameter tuning. As seen in table II the drop is accuracy for the deterministic case is as reported in the paper, We had a drop in 1% accuracies. This result was promising. We experimented with binary activations as well. We binarized the activations to be +1 or -1, while constraining the updates of the activations based on real values. This process is similar to the binarization of weights and is explained in Algorithm 2. This resulted in a drop of 8% accuracy, but the model does fairly well.

TABLE II: MNIST - Arch II

	Train	Test
No binarization	100%	98%
Deterministic binarization of weights	99%	96%
Stochastic binarization of weights	11%	11%
Deterministic binarization of weights and activations	92%	92%

CIFAR-10: VGG11:
The architecture is shown in figure 5 So the input to the architecture is a normalized and cropped 32*32*3 image and each block labelled 3*3 conv in the figure 5 is a 3*3 convolutional layer with a padding of 1 followed by

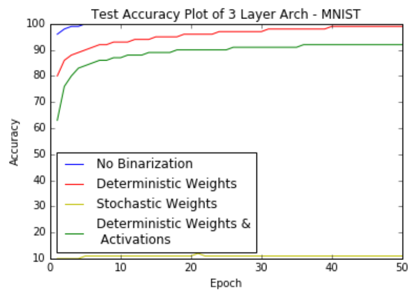

Fig. 3: MNIST Training accuracy - Arch II

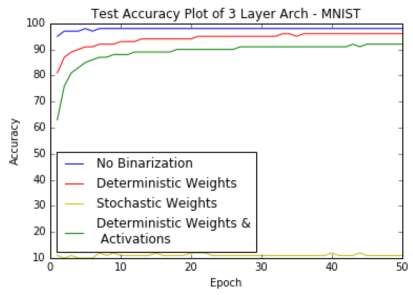

Fig. 4: MNIST Test accuracy - Arch II

a BatchNorm layer and followed by a RELU layer. Each Max Pool layer in the figure 5 is a Max Pool Layer of size 2 and stride 2. The Avg Pool Layer is of size 1 and stride 1. The last block is a linear classifier that takes in 512 and classifies it as one of the 10 classes. As described earlier the loss function used was Cross Entropy loss and the optimizer used was SGD.

We tested it out with deterministic and stochastic binarization as well as deterministic activations and at the end of 50 epochs, the results were as follows. The training set was 60000, and the test set was 10000 images. As you can see in the plot of training and test accuracies in Figures 6 and 7, the stochastic binarization scheme doesn't work well for this architecture, but the deterministic binarization of weights as well as deterministic binarization of activation, seems promising. As seen in table III the drop is accuracy for the deterministic case is 16%. We tested out deterministic binary activations as well, this resulted in a drop of 28% accuracy. Stochastic Binarization did not work well in this setting, even though this was similar to the one proposed in the paper with an adaptive learning rate form 0.001 to 0.0000003. It was unable to learn and it's results were random(10%)

TABLE III: VGG11 - CIFAR10

	Train	Test
No binarization	92%	86%
Deterministic binarization of weights	75.7%	75.9%
Stochastic binarization of weights	10.5%	10%
Deterministic binarization of weights and activations	63.5%	59.5%

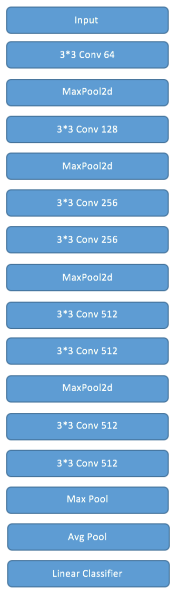

Fig. 5: VGG11 arch

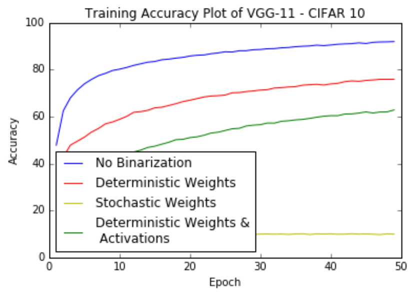

Fig. 6: CIFAR10 - Vgg11 Training accuracy

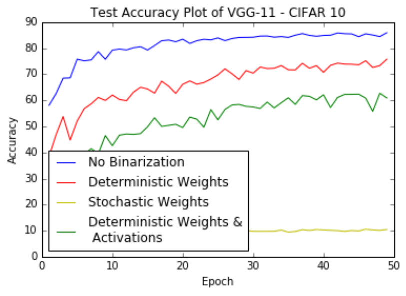

Fig. 7: CIFAR10 - Vgg11 Test accuracy

Apart from two quantization levels {-1,1} we also tried experimenting 4 quantization levels {-2, -1, 1, 2} on MNIST dataset.

VI. Conclusion:

The experiments conducted by us, revealed that it does take more epochs to train the binarized model compared to the one without binarization. But it could be offset by the fact that we need to perform ADD and SUB operations instead of MUL operations. We would like to visualize the difference in the hardware time for each epoch optimized for binary networks which we assume to be faster.

Stochastic Binarization takes a lot more time and is difficult to get it to work and it seems to be dependent on the architecture of the model. Our training time was relatively lower due to limited compute power, but we weren't able to get exact drop in accuracies as expressed in the paper and its probably because they trained it for a longer time.

In the future we would like to try out and evaluate different quantization levels instead of 2 and 1. We found the results to be indicative of success.

References

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. *CoRR*, abs/1702.00953, 2017. URL http://arxiv.org/abs/1702.00953.

Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 269–284, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2305-5. doi: 10.1145/2541940.2541967. URL http://doi.acm.org/10.1145/2541940.2541967.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Low precision arithmetic for deep learning. *CoRR*, abs/1412.7024, 2014. URL http://arxiv.org/abs/1412.7024.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *CoRR*, abs/1511.00363, 2015. URL http://arxiv.org/abs/1511.00363.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir D. Bourdev. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014. URL http://arxiv.org/abs/1412.6115.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015. URL http://arxiv.org/abs/1506.02626.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061, 2016. URL http://arxiv.org/abs/1609.07061.

Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel S. Emer, Stephen W. Keckler, and William J. Dally. SCNN: an accelerator for compressed-sparse convolutional neural networks. *CoRR*, abs/1708.04485, 2017. URL http://arxiv.org/abs/1708.04485.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016. URL http://arxiv.org/abs/1603.05279.

Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016. URL http://arxiv.org/abs/1606.06160.