Final Project Report of Group 44

BinaryConnect: Training Deep Neural Networks with binary weights during propagations

Yi Luo
Electrical Engineering
Columbia University
yl3364@columbia.edu

Fan Yang
Electrical Engineering
Columbia University
fy2207@columbia.edu

Xiaowen Zhang
Electrical Engineering
Columbia University
xz2461@columbia.edu

Jingyi Yuan
Electrical Engineering
Columbia University
jy2736@columbia.edu

April 2016

1 Introduction

In this project, we implemented BinaryConnect and reproduced some experiments in it. This report includes our summarization of the paper, our understanding of BinaryConnect, results of experiments and our discussions about them.

1.1 Definition and Benefits of BinaryConnect

BinaryConnect is a method which consists in training a DNN with binary weights, in this paper with 1 and -1, during the forward and backward propagations, while retaining precision of the stored weights in which gradients are accumulated[1].

Today, new deep learning algorithms and applications, often with multiple layers and parameters, are often limited by computational capability, especially when deep learning systems are put on low-power devices, e.g. smart phones and watches. This is greatly increasing the interest in research and techniques to compress the size of the parameters. By using binary weights, we can eliminate the need for the float-point multiplications, while decrease the size of the parameters (since int type is much smaller than float type). The change of weights from float to int would greatly increases the computational capability of deep learning system, as multipliers are the most space and power-hungry components of the digital implementation of neural networks[1].

1.2 Connection between BinaryConnect and Dropout

DropConnect[3], which is a generalization of Hinton's Dropout[6], is for regularizing large fully connected layers within neural networks, is closest to BinaryConnect. DropConnect randomly sets half of weights within the network to zero, while Dropout sets a randomly selected subset of activations to zero within each layer[3].

Dropout enables the system to be more robust. That is, under the condition that half of weights are set to be zero, the system can still work and achieve state-of-the-art results. When training neural networks, the presence of nonlinear hidden layers makes deep networks very expressive models which are therefore prone to severe overfitting[4]. Besides, Large neural networks are hard to train and slow to use at test time. The paper uses dropout also to address both these concerns.

1.3 Algorithm: SGD Training with BinaryConnect

Intuitively, BinaryConnect discretizes weights to be binary in forward and backward propagations to speed up computation process and save memory, and update parameters using real-valued ones to keep accuracy.

Before diving into algorithm details, we first consider binary values and discretization approaches. By forcing weights to be $+1$ or -1, the multiply-accumulate operation in deep learning is simply adding or subtracting without expensive multiplying, and thus save energy and memory. To obtain binarized weights, both deterministic and stochastic binarization are effective. Naturally, sign function can constraints weights to be binary values[1]:

$$w_b = \begin{cases} +1 \text{ if } w \geq 0, \\ -1 \text{ otherwise.} \end{cases} \tag{1}$$

Or alternatively[1]:

$$w_b = \begin{cases} +1 \text{ with probability } p = \sigma(w), \\ -1 \text{ with probability } 1 - p. \end{cases} \tag{2}$$

Where σ is the hard sigmoid function[1]:

$$\sigma(x) = \text{clip}(\frac{x+1}{2}) = \max(0, \min(1, \frac{x+1}{2})) \tag{3}$$

and w_b is the binarized weight and w is the real-valued weight in both cases. The advantages of hard sigmoid over soft version described in [7].

The training process of BinaryConnect with Stochastic Gradient Descent (SGD) consist of three parts: forward propagation, backward propagation and parameter updates.

Forward propagation follows the order of layers from input layer to output, and computes activations, weights and biases each layer based on results from previous one. Backward propagation computes activations gradients in reversed order from output layer to input. In these two steps, the weights we use from last iteration is pre-binarized, i.e. w_b.

Then we use the gradients obtained from backward propagation with respect to binarized weights to update real-valued weights. Note that the update uses real-value weight to keep good precision for SGD to work, since the tiny changes may cause significant change in direction:

$$w_t \leftarrow \text{clip}(w_{t-1} - \eta \frac{\partial C}{\partial w_b}) \tag{4}$$

Where C is cost function, η is learning rate and clip() is clipping function. Otherwise, if using w_b instead of w_{t-1}, SGD may end up crashed in high dimensional space. Thus we keep weights accumulated in real-valued version.

Since the magnitude of weights do not have impact on binarization during propagation, the clipping function constraints weights to $[-1, 1]$ interval for regularization, Or the real-valued weight w may grow rapidly without effecting binarized weights w_b.

1.4 Clipping

As mentioned above, the paper uses clipping while updating the parameters. Clipping is used to eliminate gradient explosion. If the parameters grow very large during iteration, divergence loss may occur. Clipping prevents this problem by limiting the weights in an appropriate range. In this paper, right after updating weights, we clip the real-valued weights within the $[-1, 1]$ interval in case that the real-valued weights grow very large and have no obvious impact on the binary weights any more.

The paper uses a hard sigmoid, which is far less computationally expensive (both in software and specialized hardware implementations) than soft sigmoid[1] and produced excellent results in the experiments.

1.5 Definition and Benefits of Batch Normalization

Batch normalization is an effective way to address problem of variation in inputs during training[2]. Due to inputs changes of each layer and each mini-batch, network needs to keep adapting to the new distributions, and requires lower learning rate and careful initialization.

Batch normalization transform input to follow a standard Gaussian distribution, by normalizing over a mini-batch using estimators of means and variances of each activation

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \tag{5}$$

where $E[x]$ and $Var[x]$ are estimator of mean and variance over mini-batch.

The distributions of normalized values have zero-mean and unit-variance, while ϵ is a small positive number for numerical stability. For each layer, batch normalization avoids internal covariance shift after adjustments of parameters. Batch normalization addresses the issues of higher learning rate, be less careful about initialization, and prevent gradient exploding[5]. It also acts as a regularizer, in some cases eliminating the need of dropout.[2]

2 Experiments

We've done experiments on two datasets: MNIST and CIFAR-10. The SVHN dataset is too large for us so that we only implemented the code for training on that set, but we did not run it.

The network structure for MNIST is exactly the same as the one mentioned in the paper. We set the initial learning rate to be 3e-4, the final learning rate to be 3e-6, and used an exponential decay. The number of epochs was set to 1000.

For CIFAR-10, we halved the number of hidden units in the convolutional layers, since even using a 64G memory Titan X server, it needs more than 40 hours (estimated) to finish 500 epochs using the original structure of the network. Here, we set the number of epochs to 250, and set the initial learning rate to be 3e-3, and the final learning rate to be 2e-6.

Method	MNIST	CIFAR-10
No regularizer	1.37%	14.91%
BinaryConnect(det.)	1.62%	17.57%
BinaryConnect(stoch.)	1.91%	17.21%
Dropout	1.26%	-

Table 1: Test error rates of DNNs trained on the MNIST and CIFAR-10

Method	MNIST	CIFAR-10
No regularizer	73.2	708.64
BinaryConnect(det.)	77.8	723.24
BinaryConnect(stoch.)	72.5	731.71
Dropout	67.2	-

Table 2: Training Time for Models (minutes)

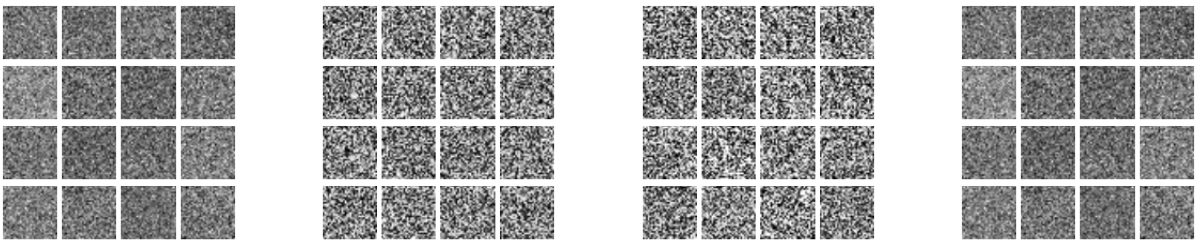

Figure 1: Features of the first layer of an MLP trained on MNIST depending on the regularizer. From left to right: no regularizer, deterministic BinaryConnect, stochastic BinaryConnect and Dropout.

3 Discussion and Insights

3.1 Discussion of Results

Figure 1 shows the features learnt by the first layer of MLP. It's an reproduction of Figure 1 in the original paper. We can see that binarization makes the weight matrix more sparse, forcing the entried to be closer to

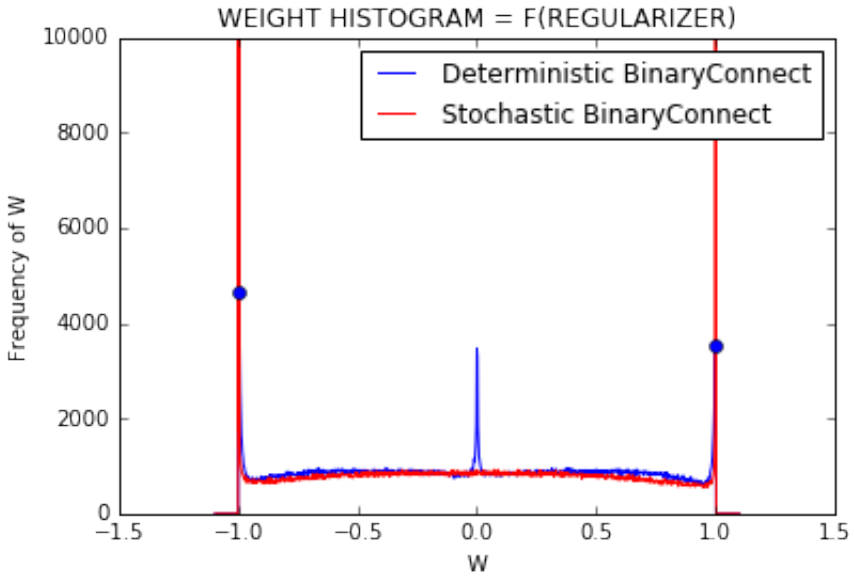

Figure 2: Histogram of the weights of the first layer of MLP trained on MNIST depending on the regularizer (zooming in).

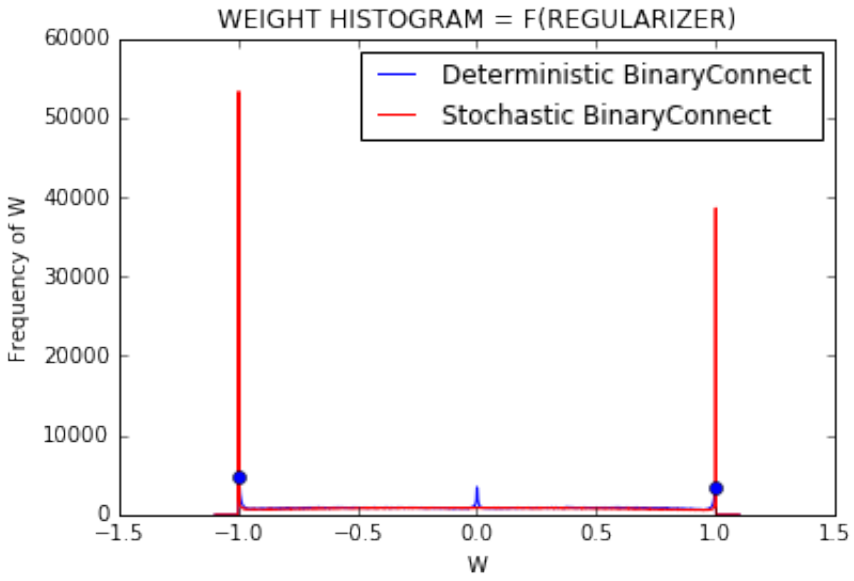

Figure 3: Histogram of the weights of the first layer of MLP trained on MNIST depending on the regularizer (original).

1 and -1. We haven't found continuous regions in the features, which could be found in the BinaryConnect paper. This might be a learning problem, or it could be the case that we haven't found the best-looking features from 764 features.

Figure 2 and Figure 3 reproduce Figure 2 in the original paper. The difference is that in our experiments, the number of entries around 1 and -1 in the deterministic case is smaller that those in the original paper. We are sure that this is due to training problem and random initialization. The overall layout are the same - binarization forces the weights to achieve 1 or -1 to decrease the cost.

We may notice that the performance of BinaryConnect is worse than what we expected. There might be several reasons for that: in the original implementation, the authors used Glorot initialization and a parameter W_LR_scale that haven't been mentioned in the paper. We guess that's why the performance is worse.

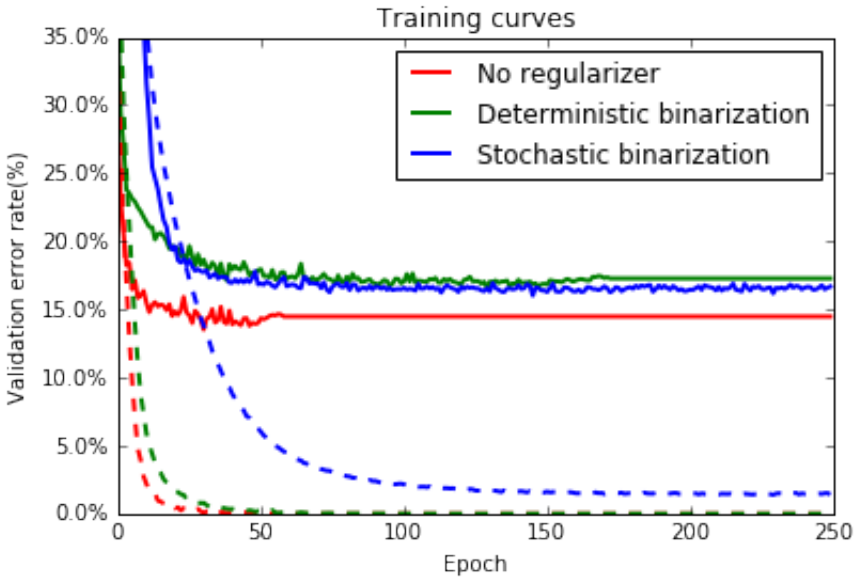

Figure 4: Training curves and test error rate of a CNN on CIFAR-10 depending on the regularizer.

Moreover, the initialization of the network and configurations are different from the original implementation, and that might also be the reason. Moreover, the performance of deterministic and stochastic binarization were worse than Dropout and even the one without regularizer. We guess that's also because we didn't use the *W_LR_scale* parameter, and the initialization and configuration of the networks are different. What we think was strange is that the performance of stochastic binarization is even worse than deterministic binarization. We don't know why.

Figure 3 shows the training cost and test error rate of three experiments on CIFAR-10. The trends are very similar to the ones in the original figure. Here, we use a smaller network so that the downgrade in the final performance is predictable. We was lucky enough to have the access to a server that has two Titan X GPU and 64G RAM, without which we may not finish training the network on time. In this case we can see that the performance of stochastic binarization is better than deterministic binarization, even though they are still worse than the one without any regularization. Again, we think the reason is the same as what we mentioned above.

In order to improve the performance, there are several ways that we can do. First, applying the *W_LR_scale* parameter may help, even though we don't quite understand why they designed that parameter. Glorot initialization may also be a good point to start training the network. Since we applied batch normalization in the network, a larger learning rate is crucial to make sure the network works, from my personal experience. Hence, to find a better learning rate range is also important. In the case of CIFAR-10, using the original structure of the network may do great help to the overall performance.

3.2 Insights

We've learnt several things from the implementation. First of all, a powerful server is very important for deep learning. Lucky enough that we have the access to a Titan X server, unless we could not even do the experiment on CIFAR-10. Second, initialization of the network is extremely important. A good initialization may help a lot on achieving a much better classification result. Moreover, the configuration of the network needs to be carefully designed in order to make sure the network to work. For example, when applying batch normalization in the network, a relatively large learning rate is crucial to make sure the network would not stuck in local minima; when applying dropout, we could not set the dropout rate too large. Last but not least, we are also curious about if binarization could also be useful in recurrent layers, such as LSTM or GRU. It could be really interesting if recurrent layers could also benefit from binarization, although it seems to be more tricky.

4 Contributions of Each Team Member

Yi Luo and Fan Yang wrote code for BinaryConnect, commited to Bitbucket tested and trained the network on CIFAR-10. Jingyi Yuan and Xiaowen Zhang debugged, trained and tested the network on MNIST and summarized the paper on Sharelatex. Details can be seen by taking a look at the commit history of sharelatex and Bitbucket.

References

[1] Courbariaux M, Bengio Y, David J P. BinaryConnect: Training Deep Neural Networks with binary weights during propagations[C]//Advances in Neural Information Processing Systems. 2015: 3105-3113.

[2] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.

[3] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In ICML'2013, 2013.

[4] S. Nitish. Improving Neural Networks with Dropout. PhD thesis, University of Toronto, Toronto, Canada, 2013.

[5] Pascanu R, Mikolov T, Bengio Y. On the difficulty of training recurrent neural networks[J]. arXiv preprint arXiv:1211.5063, 2012.

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15:1929–1958, 2014.

[7] Romo R, Brody C D, Hernández A, et al. Neuronal correlates of parametric working memory in the prefrontal cortex[J]. Nature, 1999, 399(6735): 470-473.