BinaryConnect: Training Deep Neural Networks with binary weights during propagations

E4040.2017Fall.PSYY.report
Peiqi Sun ps2998, Yong Yang yy2779
Columbia University

Abstract

This project is to repeat the result of paper "BinaryConnect: Training Deep Neural Networks with binary weights during propagations", and examine the conclusion of this paper, which is that BinaryConnect is one way of regularization. Thus the goal is to implement and train the BinaryConnect networks. The final result shows that BinaryConnect networks is truly better than no binary networks.

1. Introduction

The original paper [2] proposed that, though GPUs enabled deeper and larger neural networks, the need of stronger computation ability still exists. So this paper proposed to train a DNN with binary weight, +1 and -1, which can replace multiply-accumulate operations by simple accumulations, during forward, backward propagations and parameter update. Besides, binary weights is able to maintain the precision of gradients during the backpropagation with the real value weights, which is quite important for SGD method. Paper [3] shows that the low resolution for weights will lead to great loss in performance, especially for SGD algorithm. And also, the paper believes that BinaryConnect network is able to provide regularization like Dropout technique. Thus the BinaryConnect networks are able to reach higher accuracy.

The BinaryConnect method sounds like Drop Connect [7] method, while they indeed have some similar attribution. Drop Connect method randomly drop the weights during propagations, while BinaryConnect binary the weights during propagations. Besides, the stochastic BinaryConnect will introduce randomness to the network, which can improve the performance of the networks, just as methods variational weight noise [4] and Dropout [5, 6].

The goal of our project is to implement the BinaryConnect networks on MNIST, CIFAR-10 and SVHN datasets, find out if BinaryConnect network can achieve better or not.

The difficulty of this project is that we need to implement the forward, backward propagation and parameter update by ourselves, which has not been achieved by tensorflow. Also, we need to train the network until we get the state-of-art result. So we implement the propagation and parameter update with tf.gradients and tf.assign functions. And we achieve very good results at last.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

The original paper provides the method of BinaryConnect for forward propagation, backward propagation and parameter update.

For forward propagation, the weights are converted from the real valued weights into two value +1 and -1. There are two ways to do the binarization operation.
Deterministic,

$$w_b = \begin{cases} +1 & if\ w > 0 \\ -1 & otherwise \end{cases}$$

Stochastic,

$$w_b = \begin{cases} +1 & with\ propability\ p = \sigma(w) \\ -1 & with\ probability\ 1 - p \end{cases}$$

where σ is the hard sigmoid function.

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

The reason for hard sigmoid rather than the soft version is that it is much easier for computer to calculate. Hard sigmoid is similar to hard tanh as introduced by [8].

For backward propagation, the BinaryConnect is similar to normal network. But for parameter update, we should noticed that the BinaryConnect network update the real value weights with gradients computed by binary weights. This is to keep good precision weights during the updates.

Algorithm SGD training with BinaryConnect.

Forward propagation:

$w_b \leftarrow binarize(w_{t-1})$

For k = 1 to L, compute a_k with a_k, w_b, b_{t-1}

Backward propagation:

Initialize output layer's activations gradient $\frac{\partial\ Loss}{\partial a_L}$

For $k = L$ to 2, compute $\frac{\partial\ Loss}{\partial a_k}$ with chain rule

Parameter update:

Compute $\frac{\partial\ Loss}{\partial w_b}$ and $\frac{\partial\ Loss}{\partial b_{t-1}}$

$w_t \leftarrow clip(w_{t-1} - \alpha \frac{\partial\ Loss}{\partial w_b})$

$b_t \leftarrow b_{t-1} - \alpha \frac{\partial\ Loss}{\partial b_{t-1}}$

where L represents the number of layers, a_k represents the output of k^{th} layer.

2.2 Key Results of the Original Paper

Below are the results achieved by the original paper,

Method	MNIST	CIFAR-10	SVHN
No regularization	1.30±0.04%	10.64%	2.44%
BinaryConnect(det.)	1.29±0.08%	9.90%	2.30%
BinaryConnect(stoch.)	1.18±0.04%	8.27%	2.15%
50% Dropout	1.01±0.04%		

Table 1: the error rate of ANNs achieved by original paper.

Besides, the BinaryConnect Network should have their weights like this.

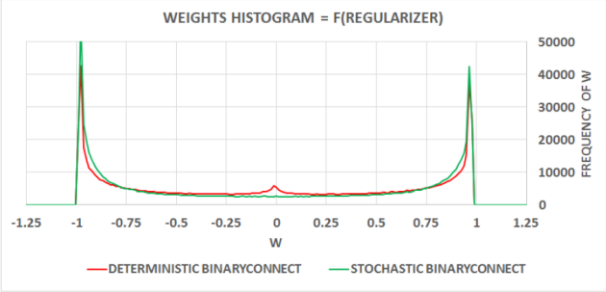

Figure 1: Histograms of weights of the first layer of MLP trained on MNIST

3. Methodology

For this section, we present the problem we encountered and the solution we find. Mostly, the problems are about how to update gradients within BinaryConnect networks.

3.1. Objectives and Technical Challenges

Our objective is to repeat the results of the Original paper. So there are three steps, implement the no regularization network (also dropout network for MNIST dataset), the deterministic BinaryConnect network, and finally the stochastic BinaryConnect.

1. Mostly, the challenge is how to implement the BinaryConnect parameter update. We are not able to use the optimizer provided by tensorflow, because the gradients is calculated with binarized weights, while they are used to update the real value weights. Thus we implement the SGD and Adam method with tf.gradient function and tf.assign function. Mostly, we are following the formula below.

SGD,

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$

Adam,

$$t = t + 1$$
$$m_t \leftarrow \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$$
$$v_t \leftarrow \beta_2 * v_{t-1} + (1 - \beta_2) * {g_t}^2$$
$$\hat{\alpha} \leftarrow \alpha * \sqrt{1 - {\beta_2}^t} / (1 - {\beta_1}^t)$$
$$\theta_t \leftarrow \theta_{t-1} - \hat{\alpha} * m_t / (\sqrt{v_t} + \epsilon)$$

Besides, we use tf.cond to avoid learning the validation and test dateset. We achieved the parameter update operation inside the network function. Thus the update operation is executed when validate the network. So we use tf.cond function to identify whether the network is training or not.

3.2. Problem Formulation and Design

Since our goal is to reproduce the results of the original paper, we need to implement the three networks used. First step is to build the same networks with the original paper, and train the networks without regularization. Then add BinaryConnect to the networks and train again. There is two BinaryConnect networks we need to achieve, one deterministic, and another stochastic. After attain all the results, we compare them with the original paper to judge the conclusion.

4. Implementation

This section is about how we implement the networks similar to the original paper, also some details about parameter update and preprocessing.

4.1. Deep Learning Network

The original paper implement 3 ANN in total, which are used for training on 3 datasets individually, the MNIST, CIFAR-10 and SVHN. The ANN for MNIST is a MLP with 3 hidden layer, no convolution layer, and use SGD for parameter update. The ANNs for CIFAR-10 and SVHN have similar architecture, and both use Adam optimizer [9] to update parameters.

Besides, these 3 ANNs all use Glorot's method [11] to initialize the parameters, and they use batch normalization [10] and ReLU [12] everywhere.

1. MNIST dataset

MNIST dataset contains a training set of 60K and a test set of 10K 28×28 gray-scale images representing digits from 0 to 9. Last 10K samples of the training set is used to be the validation set. And there is no data-augmentation, preprocessing or unsupervised pre-training. The network uses L2-SVM as output layer, which has better performance than Softmax output layer [13].

The architecture of the network show as below.

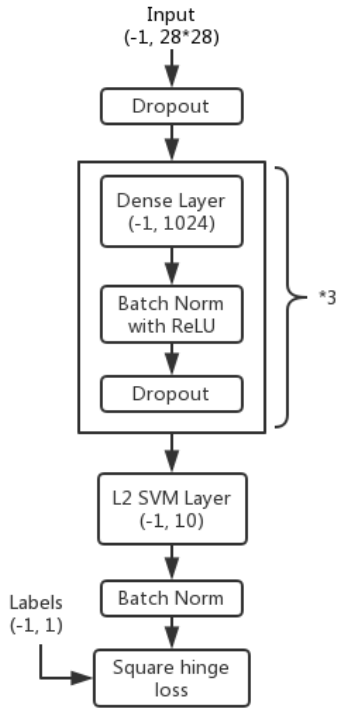

Figure 2: The architecture of the MLP trained on MNIST

And we are using the same parameters with the original paper, except the learning rate and the number of epochs. Also, we are using different weight initializer. We set weights to be initialized from uniform distribution of -1 and +1.

2. CIFAR-10 dataset

CIFAR-10 dataset contains a training set of 50K, and test set of 10K 32×32 color images representing 10 different objects. The last 5K samples of training set are used as validation set. The dataset is preprocessed with global contrast normalization and ZCA whitening to reduce the correlation among different images. Also, there is no data-augmentation. This network architecture comes from [14].

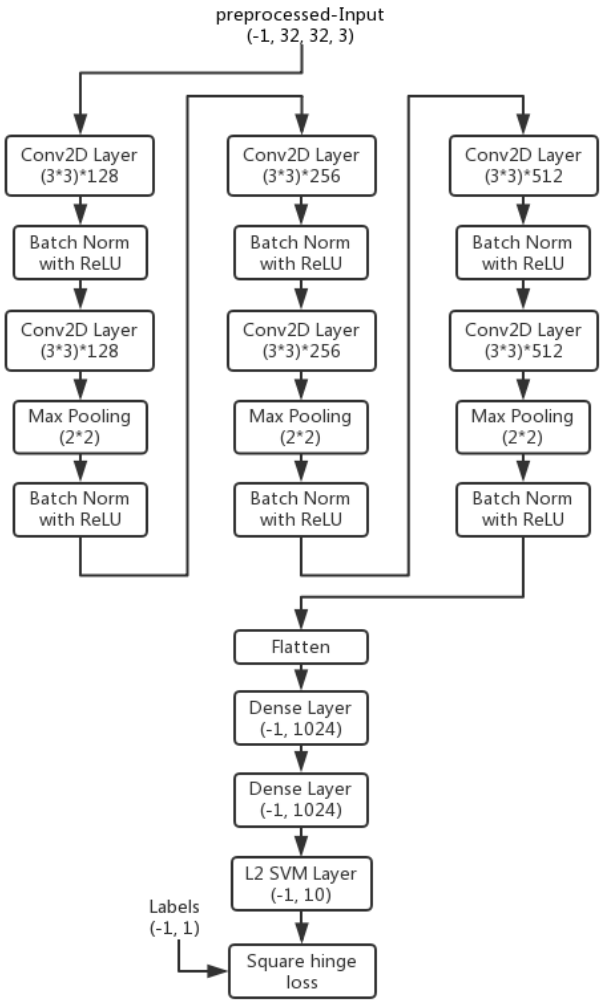

Figure 3: The architecture of the ANN train on CIFAT-10 and SVHN. The ANN for SVHN will have only half hidden neuron in the convolution layer.

3. SVHN dataset

SVHN dataset contains a training set of 604K and a test set of 26K 32×32 color images representing digits from 0 to 9. The validation set is generated based on paper [Convolutional Neural Networks Applied to House Numbers Digit Classification], which is a set of 6K images.

The network for SVHN dataset is similar to CIFAR-10, but extract half the features within the convolution layers. Also, we use Adam optimizer for parameters update. As for preprocessing, the SVHN dataset is too large to do preprocessing. Computer always break-down on this section, so we didn't do the preprocessing for SVHN. But we still generated good results.

4.2. Software Design

Below is the flowchart of our program.

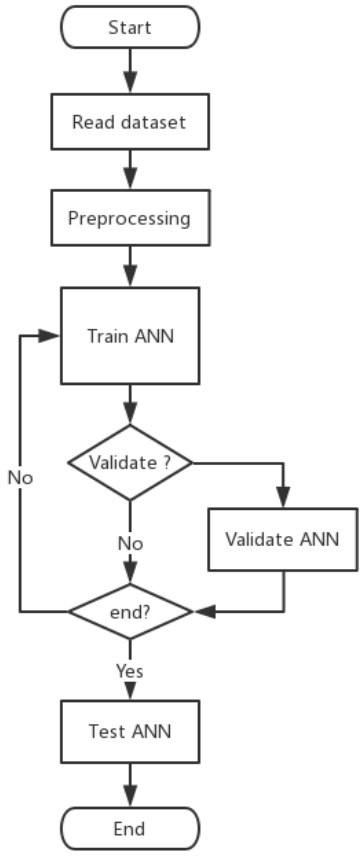

Figure 4: The flowchart of our ANNs, similar to regular ANNs.

The preprocessing contains two steps. Global contrast normalization and ZCA whitening

And the Train ANN step should be divided into 3 parts.

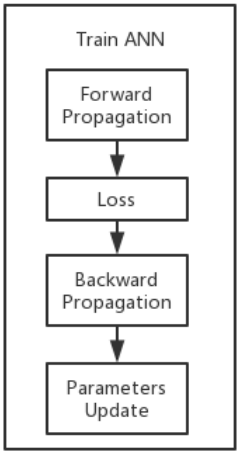

Figure 5: The detail of train step. Due to BinaryConnect method, we implement them by ourselves.

The parameters update is different from the normal network. We need to store the real value of weights, then the forward and backward propagation is processed with the binarized weights, while the parameter update is to update the real value weights with binarized weights' gradients.

The pseudo code show as below.

Algorithm training with BinaryConnect.

Forward propagation:

$w_{binary} \leftarrow binarize(w_{real})$

For k = 1 to L, $output_k = w_{binary} * output_{k-1} + b$

Backward propagation:

For $k = L$ to 2, compute $\frac{\partial Loss}{\partial output_k}$ with chain rule

Parameter update:

Compute $\frac{\partial Loss}{\partial w_{binary}}$ and $\frac{\partial Loss}{\partial b}$

$w_{real} \leftarrow clip(w_{real} - \alpha \frac{\partial Loss}{\partial w_{binary}})$

$b \leftarrow b - \alpha \frac{\partial Loss}{\partial b}$

5. Results

5.1. Project Results

1. MNIST dataset

The ANN is trained for four different method. And below are the results.

	Train acc	Val acc	Test acc
No regular.	100%	98.69%	98.64%
Binary det.	99.99%	98.55%	98.29%
Binary stoc.	99.868%	98.56%	98.5%
Dropout	99.87%	98.76%	98.71%

Table 2: The results of the ANN trained on MNIST dateset

As we can see from the table above, the maximum difference of test accuracy is only 0.42%, existing between the dropout version and the deterministic binary version. Although this result cannot confirm the regularization effect of BinaryConnect method, it is safe to say that BinaryConnect ANNs can have a not bad accuracy while reducing most multiplications in the learning process.

Here is the graph of training loss, which might have some problem with the first epoch.

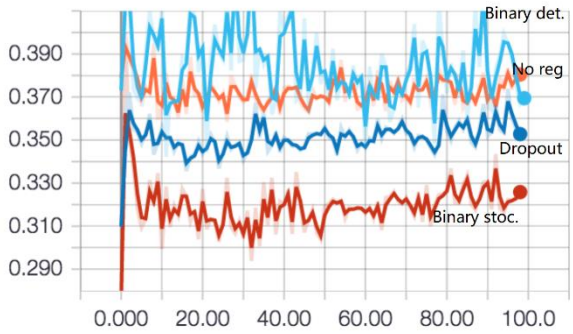

Figure 6: The loss of the ANN trained on MNIST

Also the histograms of weights. We only show the weights of first layer.

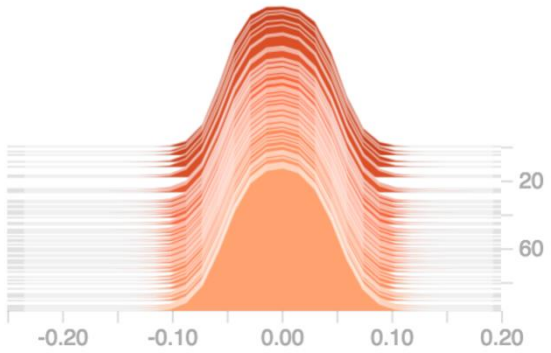

Figure 7: Weights for no regularization network

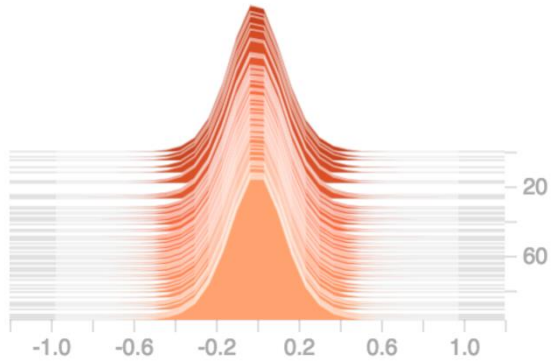

Figure 8: Weights for Dropout network

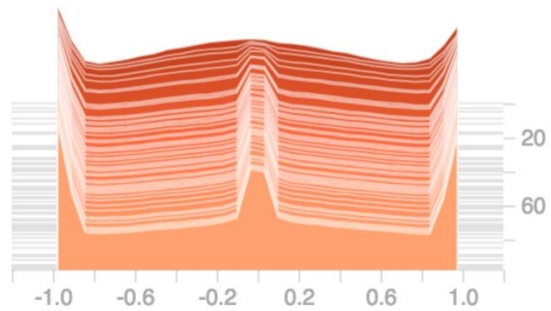

Figure 9: Weights for deterministic BinaryConnect network

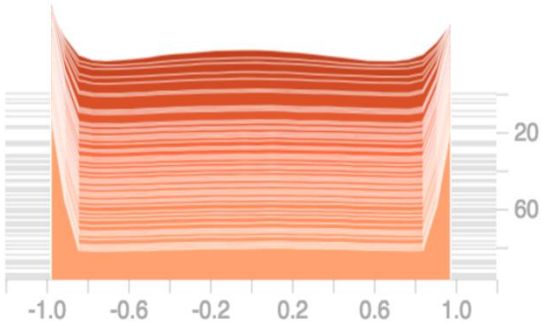

Figure 10: Weights for stochastic BinaryConnect network

The last two weights are similar to the weight histogram of the original paper, which means we successfully implement the BinaryConnect network on MNIST dataset.

2. CIFAR-10 dataset

It takes more time to train the networks of CIFAR-10 dataset than MNIST dataset's, because of the more complicated network architecture and colored input images. As a result, we chose a smaller number of training epochs in this section. Specifically, the no regularization network and the deterministic BinaryConnect network are trained for 20 epochs, and the stochastic BinaryConnect network is trained for 50 epochs.

	Train acc	Val acc	Test acc
No regular.	99.53%	82.96%	83.20%
Binary det.	99.57%	84.94%	84.95%
Binary stoc.	98.24%	85.62%	84.98%

Table 3: The results of the ANN trained on CIFAR dateset

As we can see from the table above, both of deterministic BinaryConnect network and stochastic BinaryConnect network perform better than the ordinary network, implying BinaryConnect can act as regularization. Besides, stochastic BinaryConnect network outperforms its deterministic counterpart. In addition to the tiny advantage among test accuracy, the stochastic BinaryConnect network obtained 85.62% validation accuracy with 98.24% training accuracy, while the deterministic BinaryConnect network has 84.95% validation accuracy with 99.57% training accuracy. However, the stochastic BinaryConnect network requires more epochs to train.

Besides, here is the graph of training loss for three networks.

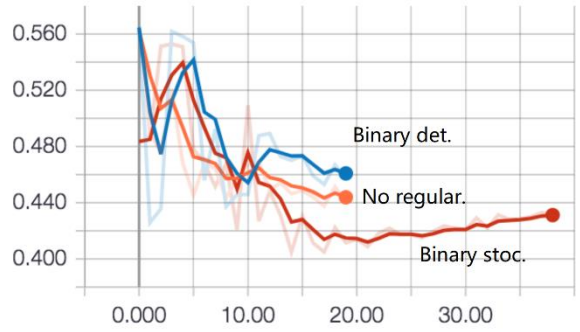

Figure 11: The loss of the ANN trained on CIFAR-10

And the weights of the first layer.

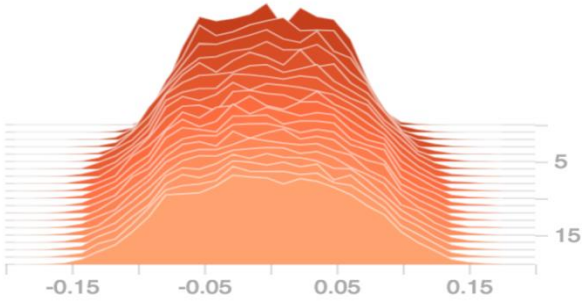

Figure 12: Weights for no regularization network

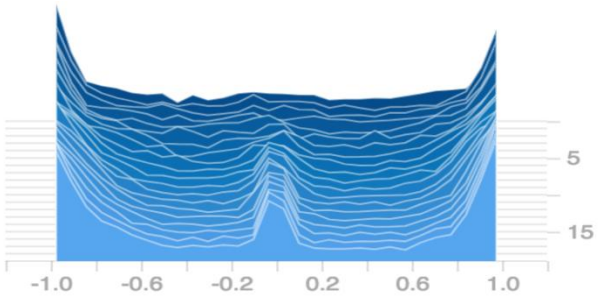

Figure 13: Weights for deterministic BinaryConnect network

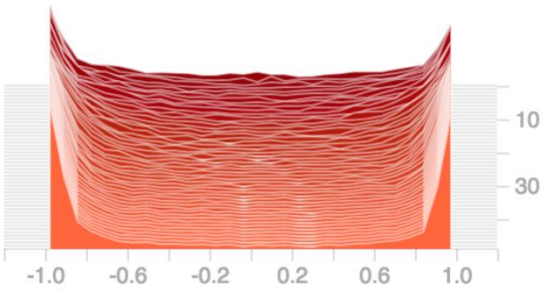

Figure 14: Weights for stochastic BinaryConnect network

The last two weights are similar to the weight histogram of the original paper, which means we successfully implement the BinaryConnect network on CIFAR-10 dataset.

3. SVHN dataset

Since these dataset is really large, it's slow to train the network, also hard to adjust the learning rate for better results. Below is the test accuracy we obtained. The no regularization network is trained for 20 epoch, need 6 hours to train, the deterministic BinaryConnect network is trained for 30 epoch, need 10 hours to train, and the stochastic BinaryConnect network is trained also 30 epoch, need 10 hours to train.

	Train acc	Val acc	Test acc
No regular.	99.4508%	97.1833%	97.2534%
Binary det.	98.8491%	96.8167%	96.9691%
Binary stoc.	97.8096%	92.0333%	90.4233%

Table 4: The results of the ANN trained on SVHN dateset

As we can see from the table above, the BinaryConnect networks are slower than the network without regularization, even though they are trained with larger learning rate, and more epoch. Mostly, the deterministic BinaryConnect network reached the same accuracy with the normal network, but didn't perform any better. While the stochastic BinaryConnect perform worst, only reach 90% accuracy. However, as we have learned from the above two dataset, this might not be the shortcoming of stochastic BinaryConnect network, just because not appropriate trained. Since the dataset is too slow to train, we don't enough time for further improvement.

Besides, here is the loss of training for three network.

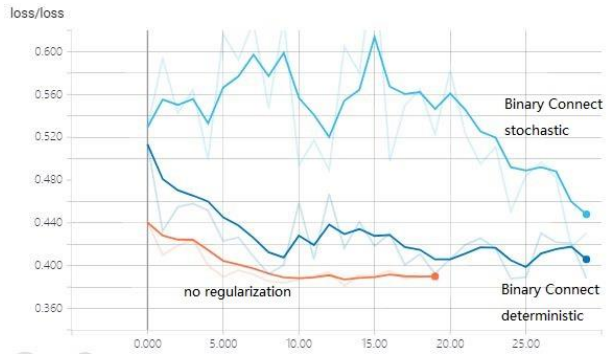

Figure 15: The loss of the ANN trained on SVHN

More obvious here, the BinaryConnect perform not so well for the start stage, but does learn to reduce the loss. Besides, stochastic BinaryConnect network is still learning when it is stopped.

Also, here are the histograms of weights, only the first layers.

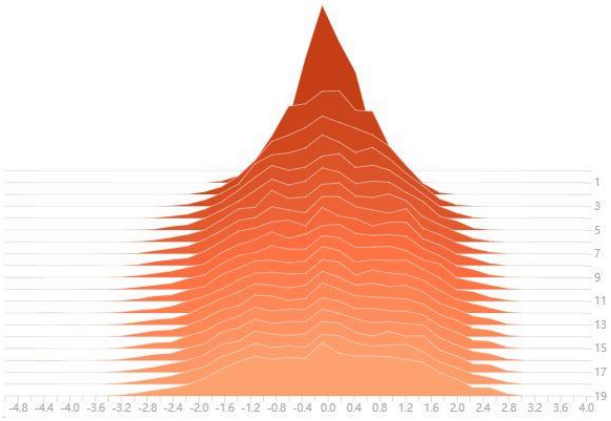

Figure 16: Weights for no regularization network

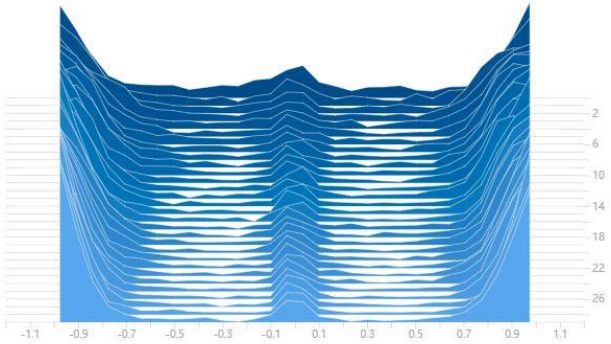

Figure 18: Weights for deterministic BinaryConnect network

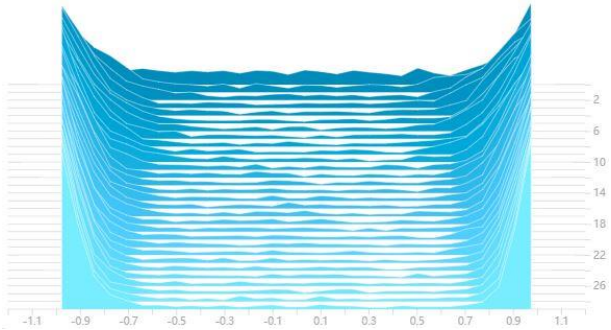

Figure 19: Weights for stochastic BinaryConnect network

Thus the last two weights are similar to the weight histogram of the original paper, which means we successfully implement the BinaryConnect network on SVHN dataset.

5.2. Comparison of Results

Here is the results of our project.

Method	MNIST	CIFAR-10	SVHN
No regularization	98.64%	83.20%	97.2534%
BinaryConnect (det.)	98.29%	84.95%	96.9691%
BinaryConnect (stoch.)	98.5%	84.98%	90.4233%
50% Dropout	98.71%	/	/

Table 5: Final results of our project

And here is the results of the original paper.

Method	MNIST	CIFAR-10	SVHN
No regularization	98.70	89.36%	97.56%
BinaryConnect(det.)	98.71	90.10%	97.70%
BinaryConnect(stoch.)	98.72	91.73%	97.85%
50% Dropout	98.99	/	/

Table 6: The results of original paper

Also, here is the comparison of different parameter we used for training.

		lr_start	lr_end	epoch
MNIST	Original	0.001	0.000003	1000
	No regu.	10	1	100
	determ.	10000	5000	100
	stochas.	10000	1000	100
CIFAR	Original	0.003	0.000002	500
	No regu.	0.001	0.0001	20
	determ.	0.1	0.001	20
	stochas.	0.1	0.0001	50
SVHN	No regu.	0.01	0.000003	200
	determ.	0.005	0.001	20
	stochas.	0.01	0.002	30
	No regu.	0.01	0.001	30

Table 7: Comparison of parameters for training.

We are using the same ANN architecture and number of neurons.

Thus, we have succeed in implement the BinaryConnect method trained on 3 different datasets. And for the MNIST dataset, we almost reached the results of the original paper. But we didn't make the results of BinaryConnect better than no regularization network. And for CIFAR-10 and SVHN datasets, they are much slower to train and also to optimize the results. We didn't generate similar results for CIFAR-10 dataset and the stochastic BinaryConnect on SVHN.

We have tried several ways to improve the results. First is to change the weight initializer. We've tried uniform distribution of [-1, 1] or some other interval, also normal distribution. But the Glorot's initializer always perform better. Besides, we have tried to change the architecture of the ANNs. Also failed to generate better. Besides, the architecture of ANNs from the original paper already have been proved to be able to generate satisfying accuracy. So we didn't change them at last.

Now let's go back to the conclusion of the original paper: the BinaryConnect method can provided regularization to generate better results, and improve the training speed.

For the first conclusion, as we can see from the tables 5, which are the results of our project, the BinaryConnect is able to achieve the similar results as no regularization network. But there is no guarantee that BinaryConnect can perform as regularization method since we didn't always generate better results with it. Only the ANN trained on CIFAR-10 proved this conclusion. And also the results of ANN trained on MNIST shows that Dropout can generate better result. But anyway, it does success to learn and perform not bad.

Besides, the BinaryConnect networks learns slower, even with larger learning rate. It needs more epoch to achieve 100% on training accuracy. And for the stochastic BinaryConnect network, we can't use too small learning rate due to its stochastic attribution. If we use too small learning rate, the weights are not able to form the histograms as we talk above, gather on +1 and -1.

But anyway, if the BinaryConnect method can speed the training speed, it is still a good method.

We tested whether BinaryConnect method is able to speedup training on computers by comparing the execution time of training 10 epochs on CIFAR-10 and SVHN. The result is as following.

CIFAR	No regularization	1.88min/epoch
	Binary deterministic	1.89min/epoch
	Binary stochastic	1.87min/epoch
SVHN	No regularization	16.7min/epoch
	Binary deterministic	17.1min/epoch
	Binary stochastic	17.6min/epoch

Table 8: Time consumption of ANNs

Actually this is reasonable for not improving the speed. Because we are using normal tensorflow ways to training, rather than implement the multiply-accumulation with simple accumulation.

The author emphasized that BinaryConnect can reduce about 2/3 of the multiplications on specialized hardware implementations of deep networks and thus potentially allowing to speed-up by a factor of 3 at training time.

5.3. Discussion of Insights Gained

The method of BinaryConnect is very straightforward. Use +1 and -1 as weights to build the neural networks, which will turn all the multiply-accumulation operations into simple accumulation operations. The network will be much faster, reducing the requirement of high performance GPUs or CPUs. So the only problem here is that whether the BinaryConnect network can achieve similar or better results than normal network. And we has already know the answer is yes. So the BinaryConnect is a very good example for us to learn: we should always think about how to simplify our problem and how to solve the problem with less or simpler variables. Also, we should know that there are many different ways to optimize the neural networks.

6. Conclusion

Our project completed 3 ANNs with 3 different dataset and reproduced the result of the original paper, showed that BinaryConnect can achieve the similar results with normal ANN without regularization. Although we didn't generate the same result of the paper, which can generate better results when BinaryConnect networks are implemented, we are now quite sure about the importance of BinaryConnect, which can accelerate the ANN while obtain similar results.

We learn from this project that we can turn comprehensive operations to simple operations to speed up the neural network, and we should always keep this in heart.

During the training of 3 ANNs, we didn't feel the acceleration of the BinaryConnect method. This is because that we implement the ANN with normal tensorflow function, which still do the multiply-accumulation operations. If we can implement our project with the fundamental operation of tensorflow, it will definitely run faster.

7. References

[1] Bitbucket link: https://yy2779@bitbucket.org/ecbm4040/2017_assignment2_yy2779.git

[2] 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: 3123-3131.

[3] Muller L K, Indiveri G. Rounding methods for neural networks with low resolution synaptic weights[J]. arXiv preprint arXiv:1504.05767, 2015.

[4] Graves A. Practical variational inference for neural networks[C]//Advances in Neural Information Processing Systems. 2011: 2348-2356.

[5] Srivastava N. Improving neural networks with dropout[J]. University of Toronto, 2013, 182.

[6] Srivastava N, Hinton G E, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. Journal of machine learning research, 2014, 15(1): 1929-1958.

[7] Wan L, Zeiler M, Zhang S, et al. Regularization of neural networks using dropconnect[C]//International Conference on Machine Learning. 2013: 1058-1066.

[8] Collobert R. Large Scale Machine Learning[J]. 2004.

[9] Kingma D, Ba J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.
[10] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C]//International Conference on Machine Learning. 2015: 448-456.
[11] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks[C]//Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. 2010: 249-256.
[12] Nair V, Hinton G E. Rectified linear units improve restricted boltzmann machines[C]//Proceedings of the 27th international conference on machine learning (ICML-10). 2010: 807-814.
[13] Tang Y. Deep learning using linear support vector machines[J]. arXiv preprint arXiv:1306.0239, 2013.
[14] Very deep convolutional networks for large-scale image recognition.

8. Appendix

8.1 Individual student contributions - table

	CUID1	CUID2
Last Name	Yang	Sun
Fraction of (useful) total contribution	1/2	1/2
What I did 1	Write the code of 3 ANN's architecture	Write the preprocessing code
What I did 2	Write the parameter update for SGD and Adam	Train the MNIST and CIFAR ANNs
What I did 3	Train the SVHN ANN	Optimize ANNs
What I did 4	Write the report	Add the results to report