BinaryConnect: Training Deep Neural Networks with binary weights during propagations

Zanxu Wang zw2864
Columbia University

Abstract

With large training sets and large models, Deep Neural Networks (DNN) have achieved remarkable results in wide range of tasks. These breakthroughs were due to the faster computational speed of GPUs, and it is likely that faster computation for both training and testing of the models will be crucial for future progressions as well as for consumer applications on low-power devices. Therefore, there is significant interest in research and development of deep learning in dedicated hardware. An algorithm called BinaryConnect was introduced in the original paper [1] which constrains weights to only two possible values during forward and backward propagation phase of deep learning while retaining precisions of the stored weight in which the gradients are accumulated. The algorithm intends to reduce the training time by replacing many multiply-accumulate operations into summations, as multipliers are the computationally exhaustive components of neural networks. In this project, BinaryConnect algorithm was extended with MLP and CNN models and implemented both deterministically and stochastically on the three datasets the original paper has used, permutation-invariant MNIST, CIFAR-10 and SVHN. The objective is to see if near state-of -the -art results can be achieved using BinaryConnect as described in the original paper. Test error of rates of 1.80%, 13.18%, 5.12% were achieved for deterministic binarization on the three datasets respectively. For stochastic binarization, the test error rates on the three datasets were 3.55%, 34.84% and 8.92%. Although there is a discrepancy of model accuracy between model in the project and the paper, it has been shown that training speed can be greatly reduced while maintaining reasonable accuracy using BinaryConnect.

1. Introduction

The background of BinaryConnect is rooted in addressing the challenges posed by the conventional training of DNNs, which requires substantial computational resources. Traditional DNNs use floating-point arithmetic for weight computations, which can be computationally expensive and memory-intensive. This poses a significant barrier when deploying DNNs on low-power devices like mobile phones or embedded systems.

BinaryConnect addresses these challenges by using binary weights instead of floating-point weights. In this approach, weights are constrained to +1 or -1, which significantly simplifies the computations during training and inference. This binary constraint reduces the memory footprint and computational complexity, as the multiplications in the network can be replaced with simple sign changes.

The biggest challenge of BinaryConnect is the limited representational capacity due to the binary nature of the weights. In standard neural networks, weights are real valued, allowing for a more nuanced representation of learned patterns. Binary weights, however, restrict this capacity, potentially leading to a loss of information and expressiveness in the model.

To mitigate this, BinaryConnect employs a technique where the real-valued weights are retained during the training phase, but only the sign of these weights is used during forward and backward propagations. [1] This allows the network to learn and adjust the real-valued weights, even though the actual operations during training are performed with binary weights. Additionally, stochastic methods are used for binarization, which introduces noise and acts as a regularizer, potentially improving the generalization of the model.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

The original paper on BinaryConnect focuses on training Deep Neural Networks with binary weights during forward and backward propagations. The methodology includes:

2.1.1 Deterministic and Stochastic Binarization:

BinaryConnect uses both deterministic (based on the sign of the weight) and stochastic methods (probabilistic binarization based on the hard sigmoid function) for converting real-valued weights to binary values.

$$w_b = \begin{cases} +1 & \text{if } w \geq 0 \\ -1 & \text{otherwise} \end{cases} \quad (1)$$

Eqn.1 shows the straightforward deterministic binarization operation which based on the sign function, where w_b is the binarized weight and w is the real-valued weight.

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

where σ is the 'hard sigmoid' function:

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

The hard sigmoid function was applied because it's far less computationally expensive while yielded excellent results in the experiments on the three datasets.

2.1.2 Preserving Full-Precision Weight Accumulators: While the forward and backward propagations use binary weights, the actual weight updates are performed with high precision, maintaining the real-valued accumulators.

A few tricks were introduced in the paper to optimize the results including clipping the weights between [-1,1] right after weight updates, batch normalization, scaling the weights learning rate respectively with the weight initialization coefficients.

After introducing methods of training a DNN with on-the-fly weight binarization, the paper also applied different methods on test-time inference of new examples.
It uses the resulting binarized weights for deterministic BinaryConnect and real-valued weights for stochastic BinaryConnect.

Weight Binarization was incorporated into MLP models as well as CNN models. MLP was used to train the permutation-invariant MNIST dataset and BinaryConnect CNN models were used to train the CIFAR-10 and SVHN datasets.

The MLP trained on MNIST consists in 3 hidden layers of 1024 Rectifier Linear Units (ReLU) and a L2-SVM output layer. The square hinge loss is minimized with SGD without momentum. An exponentially decaying learning rate was applied. The model uses batch normalization with a minibatch of size 200 to speed up the training. The experiment was repeated 6 times with different initializations and the test error rate associated with the best validation error rate after 1000 epochs was reported.
The architecture of the CNN used in CIFAR-10 and SVHN is the same:

$$(2 \times 128C3) - MP2 - (2 \times 256C3) - MP2 \\ - (2 \times 512C3) - MP2 \\ - (2 \times 1024FC) - 10SVM$$

Where C3 is a 3 × 3 ReLU convolution layer, MP2 is a 2 × 2 max-pooling layer, F C a fully connected layer, and SVM a L2-SVM output layer.[1] The square hinge loss is minimized with Adam. Batch Normalization with minibatch of size 50 was used to speed up the training and test error rate associated with the best validation error rate after 500 epochs was reported.

Same procedure with the same model was used to train SVHN except that the hidden units in every layer was reduced by half and training epochs were reduced to 200.

2.2 Key Results of the Original Paper

Method	MNIST	CIFAR-10	SVHN
No regularizer	$1.30 \pm 0.04\%$	10.64%	2.44%
BinaryConnect (det.)	$1.29 \pm 0.08\%$	9.90%	2.30%
BinaryConnect (stoch.)	$1.18 \pm 0.04\%$	**8.27%**	2.15%
50% Dropout	$1.01 \pm 0.04\%$		
Maxout Networks [29]	0.94%	11.68%	2.47%
Deep L2-SVM [30]	**0.87%**		
Network in Network [31]		10.41%	2.35%
DropConnect [21]			1.94%
Deeply-Supervised Nets [32]		9.78%	**1.92%**

Table 1: Test error rates of DNNs trained on MNIST, CIFAR-10 and SVHN, depending on the method.

Table 1 shows that test error rates of DNNs trained on MNIST, CIFAR-10, and SVHN that using BinaryConnect, which employs just a single bit per weight during propagation, does not degrade performance. In fact, results are comparable or even better than traditional DNNs without regularizers. Particularly, the stochastic version of BinaryConnect enhances performance, indicating its effectiveness as a regularizer.

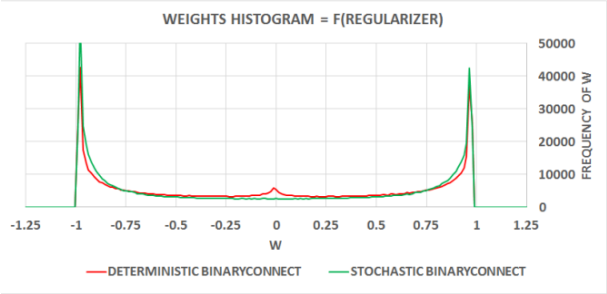

Figure 1: Histogram of Histogram of the weights of the first layer of an MLP trained on MNIST depending on the regularizer.

From Figure 1, in both deterministic and stochastic binarization, it seems that weights are trying to become deterministic to reduce the training error. It also shows that weights of deterministic BinaryConnect stucks around 0 and hesitating between -1 and 1.[1]

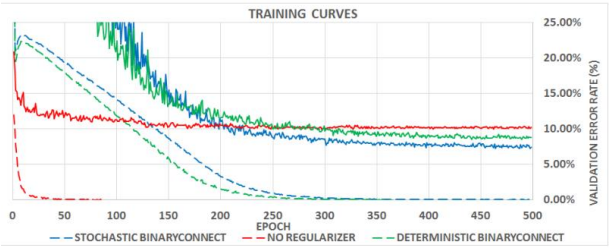

Figure 2: Training curves of a CNN on CIFAR-10 depending on the regularizer. The dotted lines represent the training costs (square hinge losses) and the continuous lines the corresponding validation error rates.

Figure 2 shows that both versions of BinaryConnect augment training cost significantly, slow down the training and lower the validation error rate. It has a similar effect of applying a Dropout scheme.

The paper introduces a new scheme for binarizing weights in Deep Neural Networks during forward and backward propagations. It demonstrates that DNNs can be effectively trained using BinaryConnect on datasets like permutation invariant MNIST, CIFAR-10, and SVHN, achieving near state-of-the-art results. BinaryConnect significantly reduces computational demands, potentially tripling training speed by cutting two-thirds of multiplications and allowing to speed-up by a factor of 3 at training time. In its deterministic form, it also greatly reduces test-time computational load and memory requirements. Future research is encouraged to apply this method to more models and datasets and to eliminate multiplications in weight update computations during training.

3. Methodology (of the Students' Project)

This project mainly focusing on implementing both deterministic and stochastic binarization on the three datasets using the same algorithm introduced in the original paper. Traditional DNNs with no regularizer and 50% dropout were not trained and compared with the BinaryConnect DNNs in this project.

The CIFAR-10 and SVHN datasets were pre-processed using global mean normalization and ZCA whitening as illustrated by the paper. Data splitting as well as hyperparameters like batch size and training epochs were set to be identical as the paper. However, to reduce training time and allow more rounds of result optimizing, early stopping scheme was applied throughout the project.

Initially, BinaryConnect MLP and CNN models with the same architecture, using same optimizer as well as output layer, as mentioned by the paper were initially trained and tested. However, many hyper parameters including learning rate, decays, details of implement Batch Normalization and the way to cope with gradients from binarizing the weights were not revealed by the paper.

Due to the initial unsatisfactory results, ithe final version of BinaryConnect CNN model in the project was changed to different hidden units in each layer, dropout scheme, l2 regularization as well as different optimizer and hyperparameters were applied to optimize the results. Customized training function as well as customized gradient function were used in order to evaluate the model in the same way the paper did. The paper uses real-valued weights for evaluating stochastic binarization and binarized weights for deterministic binarization, which is not included in the built-in training function of Keras.

Same techniques like clipping the weight within the interval of -1 and 1 right after updating the real-valued weights was applied. However, scaling the weights learning rates respectively with the weights initialization coefficients was not applied throughout the model, which might lead to potential discrepancies of model accuracies between the project and the paper.

Experiment on MNIST were repeated six times for every round of training. However, every time the initialization was set to be the same, which is different from the paper did. Data augmentation techniques could greatly improves the model's ability to generalize to unseen data especially in CIFAR-10. However, it is not applied in the project since the paper specifically mentioned not to do so. Detailed results and revised model architecture will be introduced in more detail in 4.2.

3.1. Objectives and Technical Challenges

The main objective of the project is to reproduce the model used in the paper on the same datasets and see if the state-of-the-art results could be achieved using BinaryConnect and possibly see if the stochastic version of BinaryConnect has an effect of regularizing the model.

There is a series of technical challenges:

1. Although the concept of BinaryConnect is relatively straightforward compared to other papers in the list, to train the model with the same batch size and number of epochs(1000 for MNIST, 500 for CIFAR-10, 200 for SVHN), a conservative estimation is over 50 hours will be taken to train the three datasets both deterministically and stochastically using RTX4090 mobile GPU, which is a relatively advanced GPU. Despite the time spent on building and debugging code, ensure it correctly binarize the weights and use binarized or real-valued weights for test and inference, it is computationally expansive to train and optimize model results with different architectures and hyper parameters considering the timeline of the project.

2. The models were mainly trained locally throughout the project and tested on GCP in the end. Initially, due to the CUDA and cuDNN library as well as TensorFlow not setting up correctly, CPU was used for training, taking significant amounts of time. Ultimately, CUDA v11.2, cuDNN v8.9.7 and Tensorflow 2.10 were used to ensure GPU training in the project.

3. Although the paper introduces the techniques and model architecture it used, many hyper parameters as well as details of implementation is not mentioned in the paper, to reduce the discrepancy between the project results and paper results, large amounts of hyper parameter tuning as well as modifying model architecture will be needed.

3.2.Problem Formulation and Design Description

The paper spilts the training phase into three sections: forward propagation, backward propagation and parameter update.

1. Forward Propagation: Given the DNN input, compute the unit activations layer by layer, leading to the top layer which is the output of the DNN, given its input.

2. Backward Propagation: Given the DNN target, compute the training objective's gradient w.r.t. each layer's activations, starting from the top layer and going down layer by layer until the first hidden layer.

3. Parameter Update: Compute the gradient w.r.t. each layer's parameters and then update the parameters using their computed gradients and their previous values.
With the three phases defined properly, the formulation of the BinaryConnect algorithm can be introduced here[1]:

C is the cost function for minibatch and the functions binarize (w) and clip (w) specify how to binarize and clip weights. L is the number of layers.

Require: a minibatch of (inputs, targets), previous parameters w_{t-1} (weights) and b_{t-1} (biases) and learning rate η.

Ensure: updated parameters w_t and b_t.

1 Forward propagation:
$w_b \leftarrow$ binarize (w_{t-1})
For $k = 1$ to L, compute a_k knowing a_{k-1}, w_b and b_{t-1}

2 Backward propagation:
Initialize output layer's activations gradient $\frac{\partial C}{\partial a_L}$
For $k = L$ to 2, compute $\frac{\partial C}{\partial a_{k-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and w_b

3 Parameter update:
Compute $\frac{\partial C}{\partial w_b}$ and $\frac{\partial C}{db_{t-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and a_{k-1}

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

$$b_t \leftarrow b_{t-1} - \eta \frac{\partial C}{\partial b_{t-1}}$$

Flow charts of implementing the algorithm, model architecture as well as vital functions used in the project will be presented in Section 4.

4. Implementation

This section introduces the three datasets used, the details of DNNs in the project as well as the software design.

4.1 Data

There are three datasets used in the paper and project.

4.1.1 Permutation-invariant MNIST

MNIST is a benchmark image classification dataset [33]. It consists in a training set of 60000 and a test set of 10000 28×28 gray-scale images representing digits ranging from 0 to 9. [1] Permutation-invariant MNIST is an adaptation of the classic MNIST dataset, this version involves a consistent shuffle of pixel order across all images. While the MNIST dataset features simple, well-aligned grayscale images of digits, the permutation-invariant variant challenges models to recognize digits independent of pixel alignment, thus testing the model's ability to understand complex, non-spatial patterns and structures in the data. The project and paper both use the last 10000 samples from the training set as a validation set for early stopping and model selection.

4.1.2 CIFAR-10

This dataset is more complex than MNIST, consisting of small color images (32x32 pixels) in 10 different classes, including animals and vehicles. Each class contains 6,000 images, split into 50,000 for training and 10,000 for testing. The variety and color complexity of CIFAR-10 make it a standard benchmark for image classification algorithms, especially in contexts where understanding and distinguishing finer details and color patterns are crucial. Last 5000 samples from the training set were used in the project and paper as a validation set.

4.1.3 SVHN

The Street View House Number (SVHN) dataset is a collection of digit images derived from Google Street View. Unlike the simplicity of MNIST, SVHN provides a real-world scenario with a large set of digit images against varied and colorful backgrounds. It consists in a training set of 604K and a test set of 26K color images representing digits ranging from 0 to 9.[1] All digits have been resized to a fixed resolution of 32-by-32 pixels. The original character bounding boxes are extended in the appropriate dimension to become square windows, so that resizing them to 32-by-32 pixels does not introduce aspect ratio distortions. [2] This dataset challenges models to recognize and differentiate numbers in a more natural and less controlled environment, dealing with a range of styles, scales, and color contrasts. It's particularly useful for testing algorithms under real-world image conditions. Following the same procedure as CIFAR-10, project and paper use the last 10 percent of training data as a validation set.

4.2 Deep Learning Network

The architecture of MLP for MNIST is described in Figure 3, the number of layers and hidden units were the same as the MLP in paper except that the paper used l2-SVM output

layer instead of traditional SoftMax layer. Hinge square loss is minimized using SGD without momentum and an decaying learning rate in the paper. In the project, however, Adam optimizer was used across the three datasets. The batch size is 200, number of epochs is 1000 with early stopping patience set to be 80 to reduce training time, meaning the training will be interrupted if the validation loss is not improved over 80 epochs.

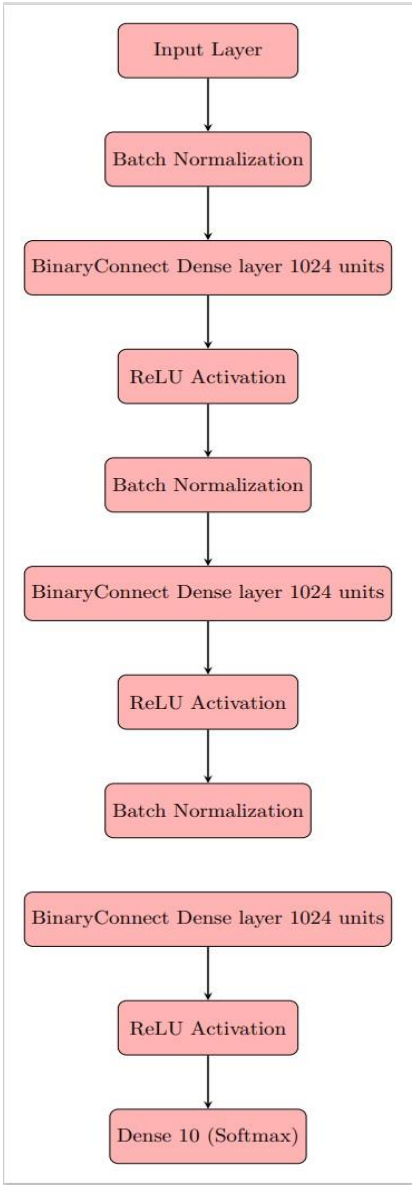

Figure 3: Architectural block diagram of BinaryConnect MLP used in MNIST

As mentioned in the paper, it uses the same procedure and same model for CIFAR-10 and SVHN except that number of epochs is reduced from 500 to 200 since SVHN is quite a large dataset.

The architecture of BinaryConnect CNN follows:

$$(2 \times 256C3 - BN) - MP2 - D0.1 - (2 \times 512C3 - BN) - MP2 - D0.1 - (2 \times 1024C3 - BN) - MP2 - D0.2 - (2048FC - BN - D0.3) - (2048FC - BN - D0.3) - 10Softmax$$

- 2×256C3: Two layers of BinaryConnect Convolution with 256 filters and a kernel size of 3.
- BN: Batch Normalization layer.
- MP2: MaxPooling layer with pool size (2, 2).
- D0.1, D0.2, D0.3: Dropout layers with dropout rates of 0.1, 0.2, and 0.3 respectively.
- 2048FC: Fully Connected (Dense) layer with 2048 units.
- 10Softmax: Output Dense layer with 10 units and softmax activation (for classification into 10 classes).

It can been seen that double amount of hidden units were used compared to the CNN used in the paper in order to capture more complexities of the datasets and faster convergence of training. To reduce the discrepancy between training accuracy and validation accuracy due to overfitting, a l2-regularization of 0.005 was applied. To reduce the noise and randomness caused by higher amount of hidden units and stochastically binarizing the weights, a drop out scheme was also applied. Specific number of hidden units, regularization and drop out rate was determined after several rounds of testing with different combinations.

4.3 Software Design

There are a few core components to implement integrate BinaryConnect to traditional MLP and CNN models. The overall flow of the implementation for the project is depicted in Figure 4.

First, it is necessary to properly handle the gradient calculation, in the project, gradients for binary weights are calculated using custom functions instead of Keras built-in function for gradients.

The reason is that regular gradient functions assume continuous, real-valued weights. However, BinaryConnect imposes binary constraints (+1 or -1) on weights, necessitating a different approach to gradient computation to accommodate this binarization while still enabling effective learning through backpropagation. These customized functions apply a condition during the gradient computation:

- For deterministic binarization, the gradient is passed through unchanged for weights within the range [-1, 1]. For weights outside this range, the gradient is set to zero. This approach respects the binary constraints while allowing the optimization algorithm to function.
- For stochastic binarization, the gradient is similarly modified based on the binary thresholds, but the binarization process is probabilistic, depending on the output of a hard sigmoid function.

By modifying gradients in this way, BinaryConnect allows the backpropagation algorithm to work effectively with binary weights, enabling efficient training of deep neural networks.

Next objective is to define and built MLP and CNN layer classes which extend on the Tensorflow keras layer. Both classes are equipped with mechanisms for deterministic and stochastic weight binarization. It involves defining custom weights and biases and overriding the call method to use binarized weights during forward propagation. Therefore, both forward propagation and backward propagation can implement weight binarization as illustrated in Figure 5.

Model classes were also built separately for MLP and CNN, as illustrated in Figure 3 and section 4.2, the model class integrates the layer class and has the functionality of using kernel constraint, regularizer as well as functions of choosing weather to use binarized weights or real-valued weights, which will be called in the customized training function.

Customized training function was also used in this project, since the built-in Keras training function cannot differentiate between the binarized weights or real-valued weights, it will use the most up-to-date weight for model evaluation by default. This is the main reason to define the customized training function. Figure 6 describes the flow of the training function includes shuffling data at the beginning of each epoch to avoid the effect of data ordering on the model, store and update real-valued weights during training, choose whether to use binarized or real-valued weights to evaluate model during validation phase as well as store the relevant metrics which can be used for plotting and monitoring training progress.

Github Link: https://github.com/ecbme4040/e4040-2023fall-Project-ZXYF-zw2864.git

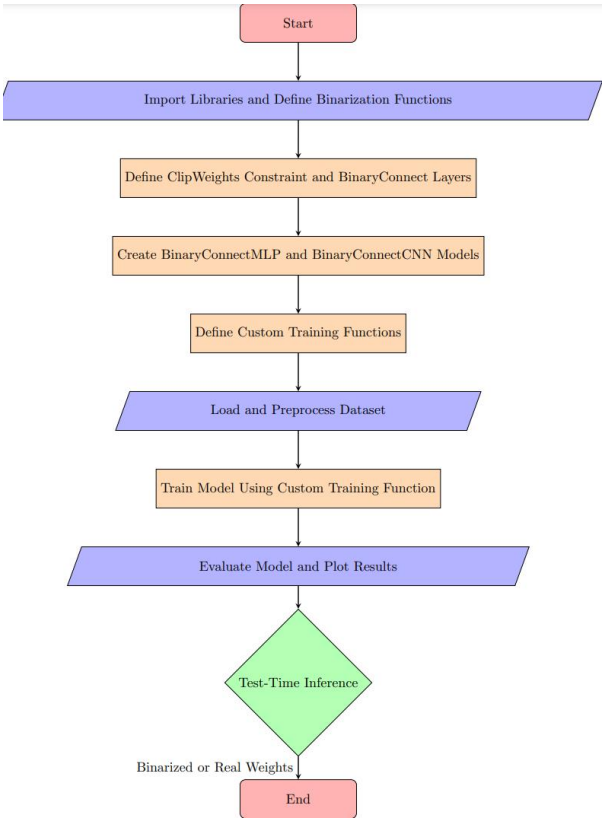

Figure 4: Flowchart of code implementation for the project from building models, pre-processing data, training models and test-time inference

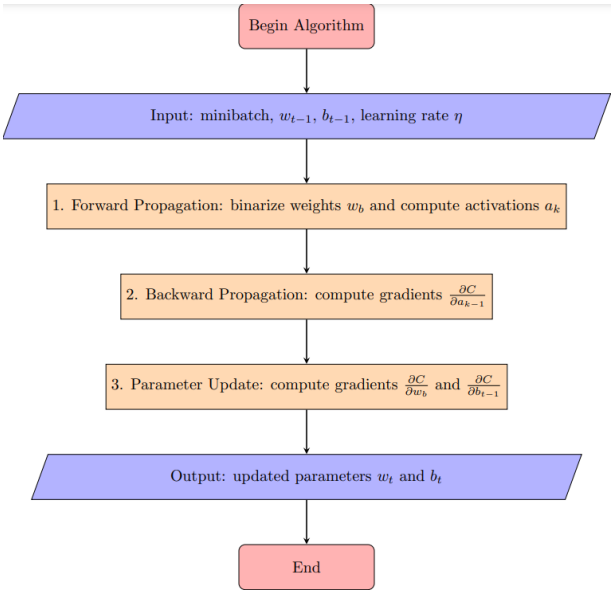

Figure 5: Flowchart of implementing the BinaryConnect algorithm step by step

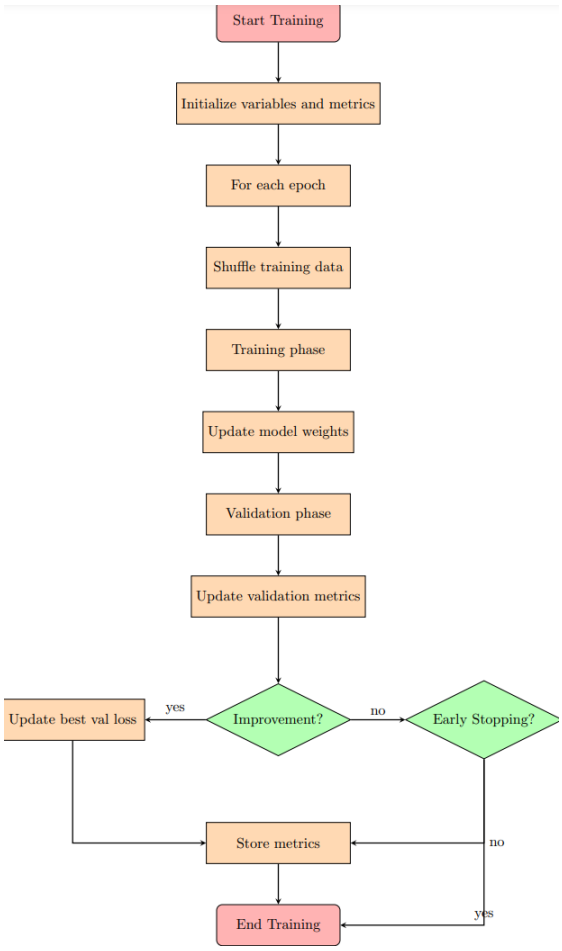

Figure 6: Flowchart of using customized training function to train the model, data shuffling, early stopping, real-valued weights were stored in the function, the model accuracy was evaluated using binarized or real-valued weights depending on the version of BinaryConnect.

5. Results

5.1 Project Results

Table 2 presents test error rates for two versions of the BinaryConnect method across three datasets: MNIST, CIFAR-10, and SVHN. The deterministic approach appears to yield better results in this comparison. The error rates are presented with a margin of error, indicating variability in the results since the experiment of MNIST was repeated for 6 times. For CIFAR-10 and SVHN, the error rate was reported with the best validation accuracy throughout the number of epochs trained as the paper did.

Method	MNIST	CIFAR-10	SVHN
BinaryConnect (det.)	1.80% ± 0.27%	13.18%	5.12%
BinaryConnect (stoch.)	3.52% ± 0.48%	34.84%	8.92%

Table 2: The test error rates for the model used in project for the three datasets.

Figure 7 shows the histogram of the real-value weights extracted from the first BinarcyConnect MLP after training MNIST for 1000 epochs. There are two pronounced spikes at the extremes (-1 and +1), which would typically indicate a deterministic binarization where weights are forced to these two values while the spread in the middle might represent the stochastic weights before binarization.

Given the nature of stochastic binarization, which introduces randomness in the weight assignment, one would expect a more uniform or varied distribution across the weight spectrum. The deterministic process, on the other hand, should produce a clear binary distribution with weights clustered at -1 and +1. However, it is clearly labelled that the behaviour of the weights is the other way around, with the deterministic binarized weights spread around the centre and the stochastic weights peaks at -1 and 1. In a typical training scenario without binarization, such sharp peaks wouldn't occur, and the weight values would likely form a more Gaussian-like distribution around zero. Since this histogram is post-training, the sharp peaks could potentially indicate that the training process is already pushing the weights towards -1 and 1. There are a few reasons for the spread of weights around the centre. After 1000 epochs, some weights may not have fully converged to the binary values due to the learning rate, weight decay, or other hyperparameters affecting training dynamics. The spread in the middle might be due to the fact that `glorot_uniform` is used. It initializes weights with a distribution that has a mean of zero and a variance that considers the number of input and output neurons, aiming to keep the scale of gradients roughly the same in all layers.

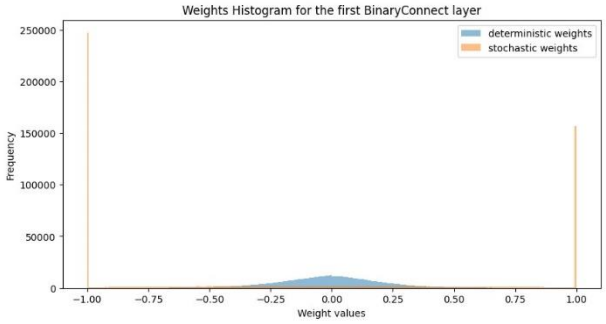

Figure 7: For MNIST, the real-valued weights for the first layer from the MLP model were extracted and plotted in histogram. Both versions of BinaryConnect were trained for 1000 epochs.

Figure 8 and 9 shows the training process of the same Binarized CNN model on CIFAR-10 and SVHN. In both cases, the deterministic approach converges much faster than the stochastic approach and much short number of number is used to achieve a higher accuracy. In both cases, early stopping with patience of 40 is applied. For the stochastic approach, the validation accuracy is much higher on SVHN than CIFAR-10. This is due to the difference

nature of the two datasets. CIFAR-10 dataset has a higher variability and SVHN has a similar nature as MNIST. Even though SVHN is more complex, one would typically expect close accuracy between training set and validation set for SVHN. If the data augmentation is applied, the model generalization ability will greatly improved and overfitting can be reduced on CIFAR-10.

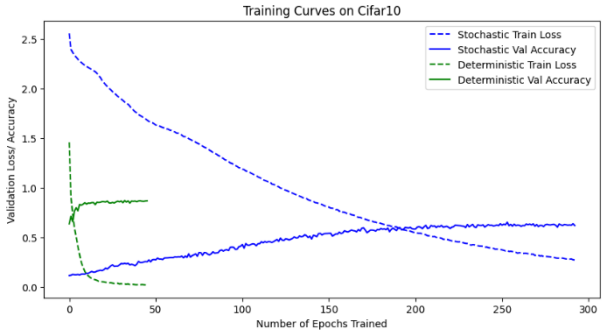

Figure 8: The training loss and validation accuracy of both versions of BinaryConnect on CIFAR-10 were recorded and plotted against the number of epochs trained.

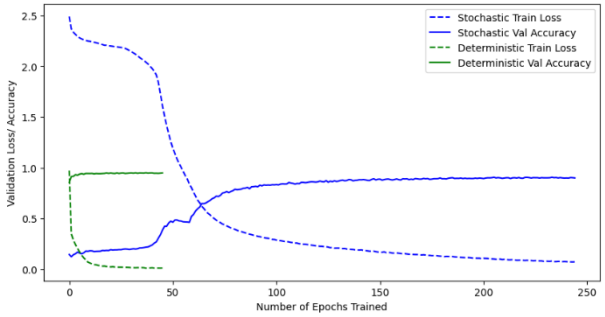

Figure 9: The training loss and validation accuracy of both versions of BinaryConnect on SVHN were recorded and plotted against the number of epochs trained.

5.2 Comparison of the Results Between the Original Paper and Students' Project

The two tables show a discrepancy of model performances. In the project, the deterministic approach outperforms the stochastic approach across the three datasets while it the opposite case for the paper. The enhanced performance by stochastic binarization from Table 1 indicates the effectiveness of using stochastic binarization as a regularizer due to its variability while this cannot be concluded according to the results in Table 2.

Figure 7 shows distinct peaks at -1 and +1, indicating that after training, weights are heavily concentrated at these two values. This is consistent with the binarization process where weights are pushed to the extremes.

The histogram from the paper in Figure 1 shows a similar trend for stochastically binarized weights but with a smoother curve leading up to the peaks. For both stochastic and deterministic weights, the spread across weight values is more pronounced in the paper's histogram, indicating a higher variability in weight distribution.

The paper's histogram suggests a more uniform distribution of stochastic weights across the range, while histogram in the project indicates a more pronounced concentration at the extremes for both deterministic and stochastic weights. The sharp spikes at the extremes in Figure 7 imply that even the stochastic weights have converged closely to binary values, which could be an artifact of the customized gradient calculation function used.

Figure 8 Shows training and validation loss and accuracy over a smaller number of epochs, around 300. The training loss decreases rapidly, which is common in early training. However, there's a notable level of fluctuation in the validation accuracy, especially for the stochastic version, which could be due to the variability introduced by stochastic binarization.

Paper's plot in Figure 2 extends over 500 epochs, showing a more stabilized and smooth convergence, particularly for the deterministic version. The validation error rate appears to decrease steadily and then plateau, indicating a point of diminishing returns where additional training doesn't significantly improve performance.

The variability in the stochastic curve in Figure 8 suggests that the model might benefit from hyperparameter tuning, possibly adjusting the learning rate or the stochastic binarization process to reduce volatility in the validation metrics. The smoother curves in the paper's plot imply a more stable training process, possibly due to a more gradual learning rate decay. Since the project used Adam optimizer with categorial cross entropy loss while the paper uses SGD with hinge square loss.

5.3 Discussion / Insights Gained

1. Through the project, it became evident that BinaryConnect offers a trade-off between computational efficiency and model accuracy. The project reveals that BinaryConnect can significantly reduce computational demands and training time. Despite accuracy discrepancies with the original paper, the approach demonstrates efficiency, especially in deterministic form, showing potential for low-power devices.

2. The implementation difficulties, particularly around stochastic binarization, highlighted the sensitivity of neural networks to initialization and binarization methods. Since the main implementation difference between the project and the paper is the choice of optimizer, loss function and

hyper parameters like learning rate and decays. It was insightful to observe the impact of the weight distribution's deviation from the binary extremes on the model's performance.

3. From my experience particularly on the Cifar-10 dataset, before implementing the dropout scheme, significant overfitting issues persistent and the validation accuracy plateaued around low values, indicating the effectiveness of applying dropout as a form a regularizer. Dropout randomly "drops" or deactivates a subset of neurons, which forces the network to learn more robust features that are not reliant on any specific set of neurons. This improves the generalization of the model to new, unseen data.

4. Increasing model complexity can sometimes accelerate convergence during training because a more complex model has a greater capacity to learn from data. For the CIFAR-10 dataset, before doubling the hidden units used in each CNN layer, the training phase slowly converges, and training accuracy stuck around 88 percent. The convergence speed improves as well as training accuracy. However, this is not a guarantee, as overly complex models can also overfit to the training data, learning noise rather than the underlying pattern, which can actually harm performance on unseen data. Therefore, while complexity might aid convergence, it needs to be balanced with techniques to ensure generalization, such as regularization or dropout to monitor for overfitting.

6. Future Work

1. In this project, traditional DNNs with similar architectures were not trained and evaluated to compare with the BinaryConnect model and see if the stochastic version of BinaryConnect could act as a form of regularizer.

2. Data augmentation of CIFAR-10 can be done and then trained on the same model used in the project to compare the model's performance.

3. Some techniques mentioned in the paper includes weights coefficient rescaling are not implemented in the project, which might cause the performance discrepancies.

4. The discrepancy of the weight's histogram is particularly worth further investigation to check if the weight binarization is implemented correctly in current code.

5. Experimenting with other datasets and network architectures may further validate BinaryConnect's utility.

6. Hardware implementation of binary networks could be investigated to fully realize their computational benefits.

7. Conclusion

This project mainly focused on integrating deterministic and stochastic BinaryConnect algorithms to MLP and CNN models and trained these models on public datasets, including permutation-invariant MNIST, CIFAR-10 and SVHN. The model performance of both versions was recorded and compared with the model performance on the paper. Most parts of the implementation were used in the same manner with the paper, except the choice of optimizer as well as hyper parameters like learning rates, decay and number of epochs trained. Despite the discrepancy of the model performance, the project unscored the potential of using BinaryConnect to maintain the performance of traditional DNNs while saving much computing resources, laid groundwork for further optimization.

8. Acknowledgement

Special thanks to my friend, Fei Li, for constantly providing me with emotional support throughout the project. It's been quite challenging to complete the project on my own, even though it's based on a relatively simple paper to replicate and the results have discrepancies with the original paper. There were moments when I almost gave up: when the model's accuracy plateaued at around 10 percent, when the histogram for extracted weights followed a Gaussian distribution—which is the opposite of what the paper indicated—when my laptop frequently crashed, or when I accidentally shut down the kernel after a long training session. However, ultimately, it's been a rewarding experience to reflect upon while writing this report.

9. References

[1] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'15)*, Montreal, Canada, 2015, pp. 3123–3131.

10. Appendix

10.1 Individual Student Contributions in Fractions

	zw2864
Last Name	Wang
Fraction of (useful) total contribution	1/1

What I did 1	Implement algorithm and replicate model from scratch
What I did 2	Train and test the model, tune model architecture and hyperparameters to optimize results
What I did 3	Report Writing