Accelerating Neural Networks with Block-Circulant Matrices and Fast Fourier Transform

Hasnain Abdur Rehman[*]
Boston University
Boston, MA 02115
hasnain@bu.edu

Dasha Smolina [†]
Boston University
Boston, MA 02115
dsmolina@bu.edu

Abstract

Neural networks are computationally expensive and have intensive memory requirements. As the size and prominence of NN's continues to grow, it is critical to improve their efficiency and performance while maintaining accuracy and scalability. Improvement techniques such as hardware acceleration and model compression are limited by their high power consumption, irregular network structure, increased complexity, and/or lack of precision control. As an effort to explore overcoming these drawbacks, we evaluated the CirNN approach, an approach that represents weight matrices as block-circulant matrices for improving storage complexity and utilizes the Fast Fourier Transform (FFT) for reducing computational complexity with negligible accuracy loss. CirNN is unique because it retains efficiency without compressing the data. We will compare the performance of the CirNN techniques in a fully connected neural network as compared to a multi-layered perceptron neural network. The circulant approach with FFT computations should facilitate high performance and energy efficiency, however, it requires intensive assumptions about the data's ability to be represented as block-circulant matrices without conversion.

1 Background

1.1 Neural Networks

Neural networks are experiencing a resurgence fueled largely by the prevalence of big data and increased processing power of graphics chips. Modeled loosely on the human brain, a neural network consists of thousands or even millions of simple processing nodes that are densely interconnected and often organized into layers. Neural networks are utilized for tremendously challenging problems because of their ability to efficiently perform some task by analyzing training examples, particularly impressive because once trained, they continue to learn on their own. The real world applications of neural networks vary from facial recognition, to hedge fund analytics, to autonomous vehicle progression, and so much more.

The success of deep learning with neural networks relies on large models consisting of multiple interconnected layers and over millions of parameters for models. Larger neural networks allow for more complex features and improve the overall accuracy. However, this high performance comes with a high cost in terms of computational capability and memory requirements. To reduce these limitations and accelerate neural networks, various approaches have been explored, some of which have sustained high levels of interest.

[*] Implementation of the CirNN, lead the collaborative implementation of the multi-layered perceptron neural network, collaborative documentation and project reporting

[†] Collaborative implementation of the multi-layered perceptron neural network, led the collaborative documentation and project reporting

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

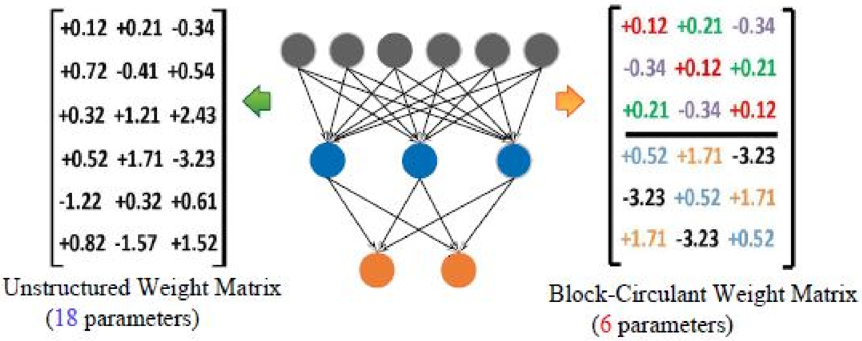


Figure 1: Block-circulant Matrices for weight representation

1.2 Alternative Approaches to Improving Neural Networks

One trend for improving neural network efficiency is hardware acceleration. There are various neural network hardware accelerator platforms each with their own advantages and disadvantages, such as graphics processing units (GPUs), application specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs). GPUs are excellent in parallel processing, but they consume a huge amount of power and typically the latency will be high, so they are not always suitable. ASIC-based acceleration overcomes the limitations of general-purpose computing approaches but they have less I/O bandwidth, memory, and other computing resources and are very costly to develop. Alternatively FPGA-based accelerators attain high performance through extensive parallelism and programmability, but they are complex and difficult to program.

Another trend aimed to tackle the energy efficiency problem caused by the large model sizes of neural networks is model compression such as pruning, weight quantization, and low rank approximation. However, this compression leads to minor accuracy degradation, irregular network structure, increased training complexity, and a lack of rigorous guarantee of compression ratio and inference accuracy.

1.3 CirCNN

The ideal technique for improving neural network efficiency should maintain the network structure, reduce the complexity of inference and training, and retain the mathematical foundations of computations for accuracy. By making use of the Fast Fourier Transform (FFT) and block-circulant matrices, the CirNN approach obtains these benchmarks.

1.4 Block Matrices

Organizing the weights as a block-circulant matrix allows us to reduce the storage complexity from O(n^2) → O(n). The concept of the block-circulant matrix compared to the ordinary weight matrix is shown in Fig. 1. In a square circulant matrix, each row or column vector is the circulant reformat of the other vectors. A non-square matrix could be represented by a set of square circulant submatrices (blocks). In Fig. 1, the unstructured 6×3 weight matrix (on the left) holds 18 parameters. Suppose we can represent the weights using two 3×3 circulant matrices (on the right), we just need to store 6 parameters, easily leading to storage size reduction. The reduction ratio is determined by the block size: larger block size leads to high compression ratio. It is important to note that utilization of these block-circulant matrices involves no conversion. Instead, we assume the weights can be represented by this style of matrix.

1.5 FFT

Block-circulant representation of the weights allows for a computational complexity reduction (O(n^2) → O(nlogn)) by utilizing the Fast Fourier Transform (FFT). FFT is implemented as the basic computing block. According to the circulant convolution theorem, the calculation of the weights multiplied by the inputs to pass through the layers of our neural network can be performed using FFT computations.

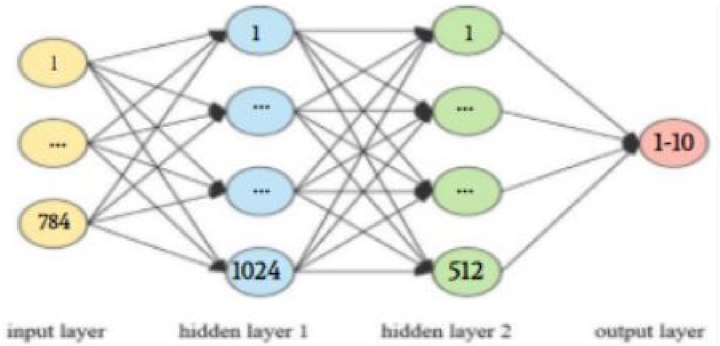


Figure 2: Simplified NN Architecture Diagram

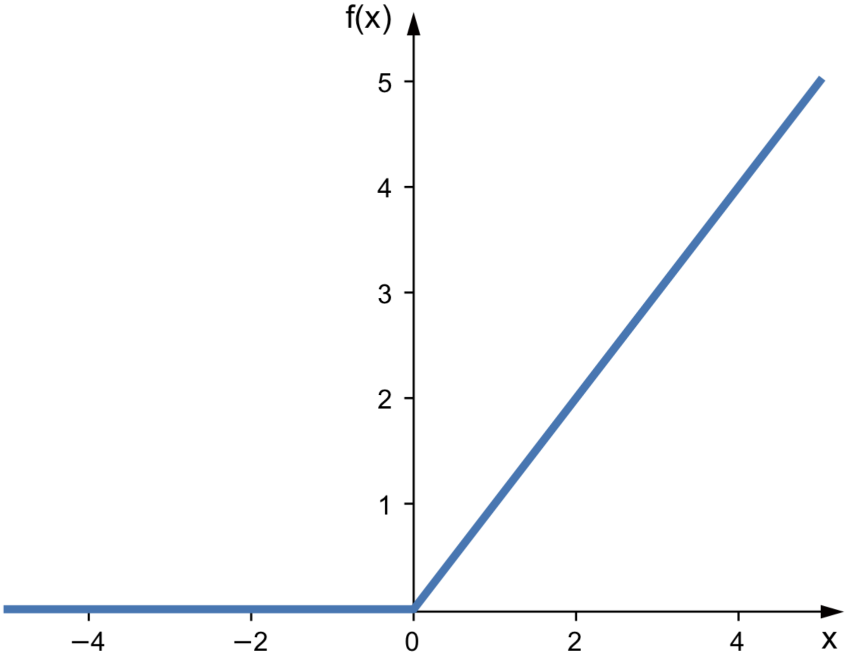


Figure 3: ReLU

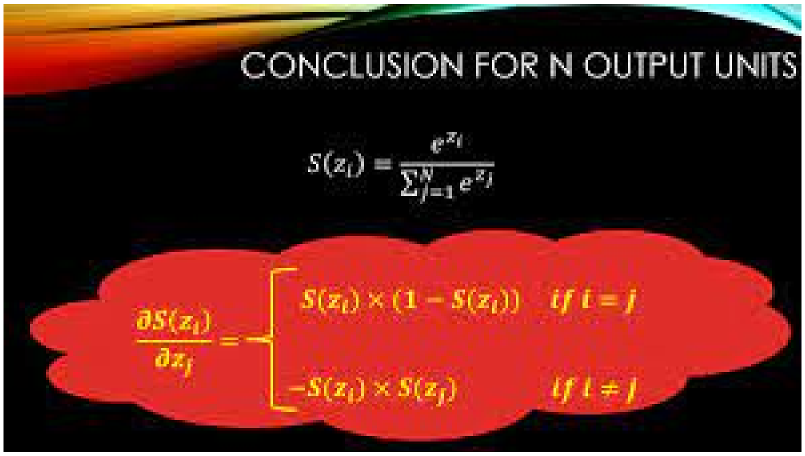
2 Multi-Layered Perceptron Neural Network Implementation

To ensure that we had a basis of comparison, we implemented a regular multi-layered perceptron neural network in MATLAB from scratch. Our multi-layered perceptron neural network consists of input and output layers, and two hidden layers with many neurons stacked together. Each neuron receives a multiplied version of inputs and random weights which is then summed with a bias value. Then, this is then passed to an activation function which decides the final value to be given out of the neuron. Once the output is generated from the output layer, the loss function is calculated and backpropagation adjusts the weights to make the loss minimum. Optimizing the weight values is what determines the success of the neural network.

2.1 Neural Network Details

2.1.1 Arichitecture and Forward Propagation

The simplified architecture of the neural network is shown in Fig. 2. Our neural network consists of an input layer with 784 neurons. These values are multiplied by the weights and passed through the ReLU activation function to the first hidden layer, which consists of 1024 neurons. The rectified



CONCLUSION FOR N OUTPUT UNITS

$$S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

$$\frac{\partial S(z_i)}{\partial z_j} = \begin{cases} S(z_i) \times (1 - S(z_i)) & \text{if } i = j \\ -S(z_i) \times S(z_j) & \text{if } i \neq j \end{cases}$$

Figure 4: Softmax Derivative for Gradient Descent

linear unit activation function or ReLU for short (shown in graph form in Fig. 3) is a piecewise linear function that outputs the input directly if it is positive, otherwise, it will output zero. This process is repeated to obtain the 512 neuron values of the second hidden layer. Finally, these values are multiplied by the next weight values and passed to the softmax activation function to obtain the ten output layer values. The softmax activation function is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are relatively proportional to the other values in the vector. The details and an example of this equation are shown in Fig. 4. As shown, the values before the softmax function can be positive, negative, or zero and they are converted to proportions that sum to one. The derivative of the softmax function is shown in Fig. 4 as well and explicates the dependencies between the elements inside the Softmax function and how to compute the gradient.

2.1.2 Backwards Propagation

Backpropagation is the algorithm that allows the multi-layer perceptron to efficiently and iteratively adjust the weights in the network to minimize the cost function. We used gradient descent as our optimization function for our neural network. In each iteration, after the weighted sums are forwarded through the four layers, the gradient of the mean squared error, our loss function, is computed across all input and output pairs. Then, to propagate it back, the weights of the hidden layers are updated with the value of the gradient. We updated the weights with momentum which incorporates the concept that previous changes in the weights should influence the current direction of movement in weight space. Momentum helps our neural network roll past local minima. We also decrease our learning rate with each batch which helps us fine tune our neural network weights more so as the learning progresses.

3 CirCNN

Our CirNN utilizes the same neuron scheme (784 → 1024 → 512 → 10) as our multi-layered perceptron neural network. However, we padded the first input layer with a column of zeros to make the layer size divisible by five, which allows us to use a block size of five for our block-circulant matrices. For the other layers, we padded them with normal neurons and bias neurons to make them also multiples of five to have the block size of five. We also utilized the same activation functions, ReLU for the two hidden layers and softmax for the final output layer. Also, the weights were updated using momentum and a decreasing learning rate. The difference in performance of the CirNN model stems from the changes made in the fully-connected, or FC, layer algorithm changes.

3.1 Forward and Backwards Propagation

For the forward propagation process in the FC layer of the CirNN, by partitioning the weight matrix into 2D blocks of sub-matrices, we are able to utilize FFT for our computations. An illustration of

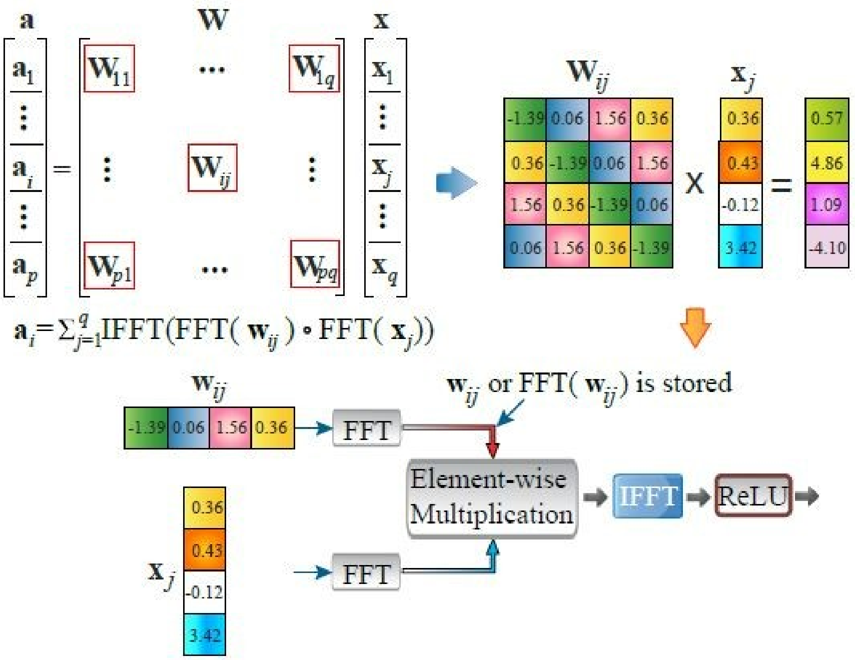


Figure 5: Illustration of the calculation of weights in the FC layer

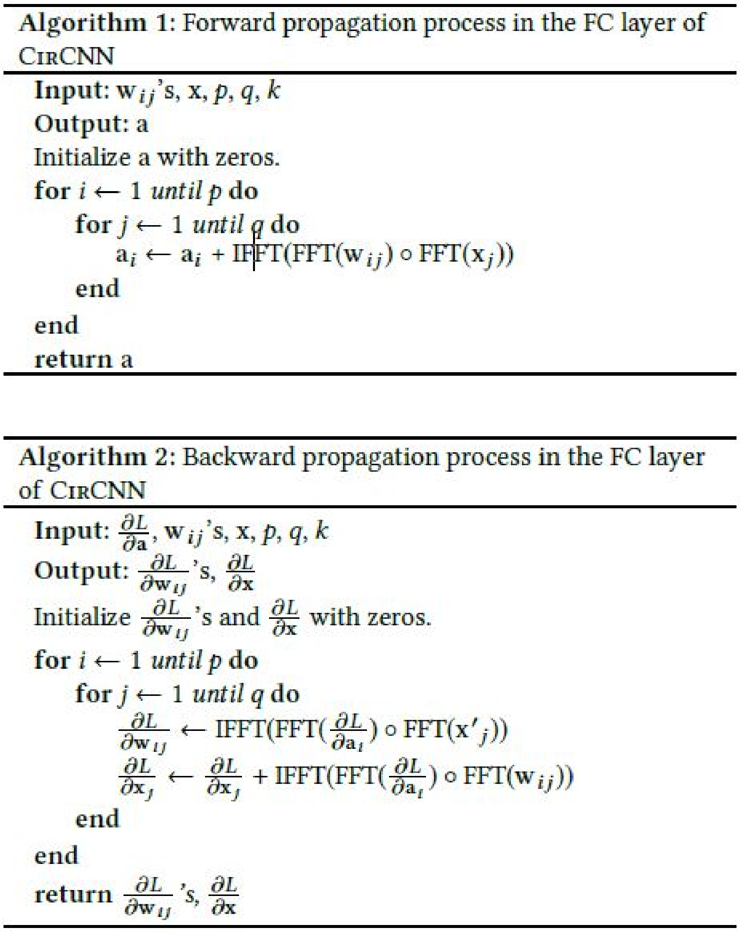
the calculation of the weights can be seen in Fig. 5. The vector of weight values in the circulant matrix are passed through FFT and the inputs are also passed through the FFT. With these values in the frequency domain, we do an element-wise multiplication of the vectors and then perform the inverse FFT to return the values to the original domain. We then proceed to our activation function as usual. The way this process was implemented can be seen in the algorithm 1 details in Fig. 6.

For the back propagation of the CirNN, we continue to use gradient descent, momentum, and a decreasing learning rate. However, the computation of the gradient descent is performed by using the FFT because the matrices are in block-circulant form. The way we implemented this is shown in the algorithm 2 details of Fig. 6.

4 Evaluation

For our evaluation of the neural networks, we utilized the MNIST dataset of handwritten digits. The dataset consisted of 50,000 training samples, 10,000 validation samples, and 10,000 testing samples. Each sample had 784 features that coincide with the 784 neurons we have in our input layer. The output for each sample was the max value of the output layer (a number one through nine) and this output represents the MNIST digit that we are classifying the sample as.

The multi-layered perceptron neural network achieved a test accuracy of 96% and the batch and test loss changes as the number of iterations increased is shown in Fig. 7. For the CirNN, we used a similar architecture and methedology for solving the MNIST dataset problem. However, the CirNN's strict requirements for the data's structure and ability to be represented as block-circulant matrices impeded our neural network's ability to learn with this algorithm. The weight initialization of the CirCNN is critical and makes this method very difficult to train as a neural network. We implemented the CirNN algorithm correctly and we suspect that the specific structure and data of the MNIST dataset problem we are tackling makes this type of network very difficult to train. The paper by Ding et. al. does not specify any parameters or network structre for solving this type of problem and we suspect that the structure and weight initialization approach we chose to pursue makes CirNN very difficult to train with a high accuracy. There are multiple factors that may be possible to tune to improve the CirNN, however it is unclear if training the CirNN on this problem can possibly achieve high accuracy levels and there is no code implementation of the CirNN approach available online



Algorithm 1: Forward propagation process in the FC layer of CirCNN

```
Input: $\mathbf{w}_{ij}$'s, $\mathbf{x}$, $p$, $q$, $k$
Output: $\mathbf{a}$
Initialize $\mathbf{a}$ with zeros.
for $i \leftarrow 1$ until $p$ do
    for $j \leftarrow 1$ until $q$ do
        $\mathbf{a}_i \leftarrow \mathbf{a}_i + \text{IFFT}(\text{FFT}(\mathbf{w}_{ij}) \circ \text{FFT}(\mathbf{x}_j))$
    end
end
return $\mathbf{a}$
```

Algorithm 2: Backward propagation process in the FC layer of CirCNN

```
Input: $\frac{\partial L}{\partial \mathbf{a}}$, $\mathbf{w}_{ij}$'s, $\mathbf{x}$, $p$, $q$, $k$
Output: $\frac{\partial L}{\partial \mathbf{w}_{ij}}$'s, $\frac{\partial L}{\partial \mathbf{x}}$
Initialize $\frac{\partial L}{\partial \mathbf{w}_{ij}}$'s and $\frac{\partial L}{\partial \mathbf{x}}$ with zeros.
for $i \leftarrow 1$ until $p$ do
    for $j \leftarrow 1$ until $q$ do
        $\frac{\partial L}{\partial \mathbf{w}_{ij}} \leftarrow \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}_i}) \circ \text{FFT}(\mathbf{x'}_j))$
        $\frac{\partial L}{\partial \mathbf{x}_j} \leftarrow \frac{\partial L}{\partial \mathbf{x}_j} + \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}_i}) \circ \text{FFT}(\mathbf{w}_{ij}))$
    end
end
return $\frac{\partial L}{\partial \mathbf{w}_{ij}}$'s, $\frac{\partial L}{\partial \mathbf{x}}$
```

Figure 6: Algorithm Details for FC Layer

to compare our results to. The weight initialization is most critical factor for the success of this algorithm and it is very difficult to initiliaze the weights to achieve a high level of accuracy.

5 Conclusion

The paper we implemented proposed CirCNN, which is an approach towards implementing a neural network using block-circulant matrices and the Fast Fourier Transform in order to reduce the computational complexity and storage requirements of convolutional neural networks. The original results of the paper demonstrated a very high energy efficiency and performance. Training the CirCNN for the MNIST dataset problem with normal weight initialization procedures and the network structure that is successful for a multi-layered perceptron does not achieve the same level of accuracy. This is because the CirCNN network saturates and experiences exploding gradients. These factors makes the network very difficult to train on such a problem with high accuracy. Our CirNN is implemented correctly and trains properly but does not achieve extremely high levels of accuracy as our multi-layer perceptron neural network does with the MNIST dataset.

Acknowledgments and Disclosure of Funding

Thank you Professor Orabona for the thought-provoking and challenging assignment.

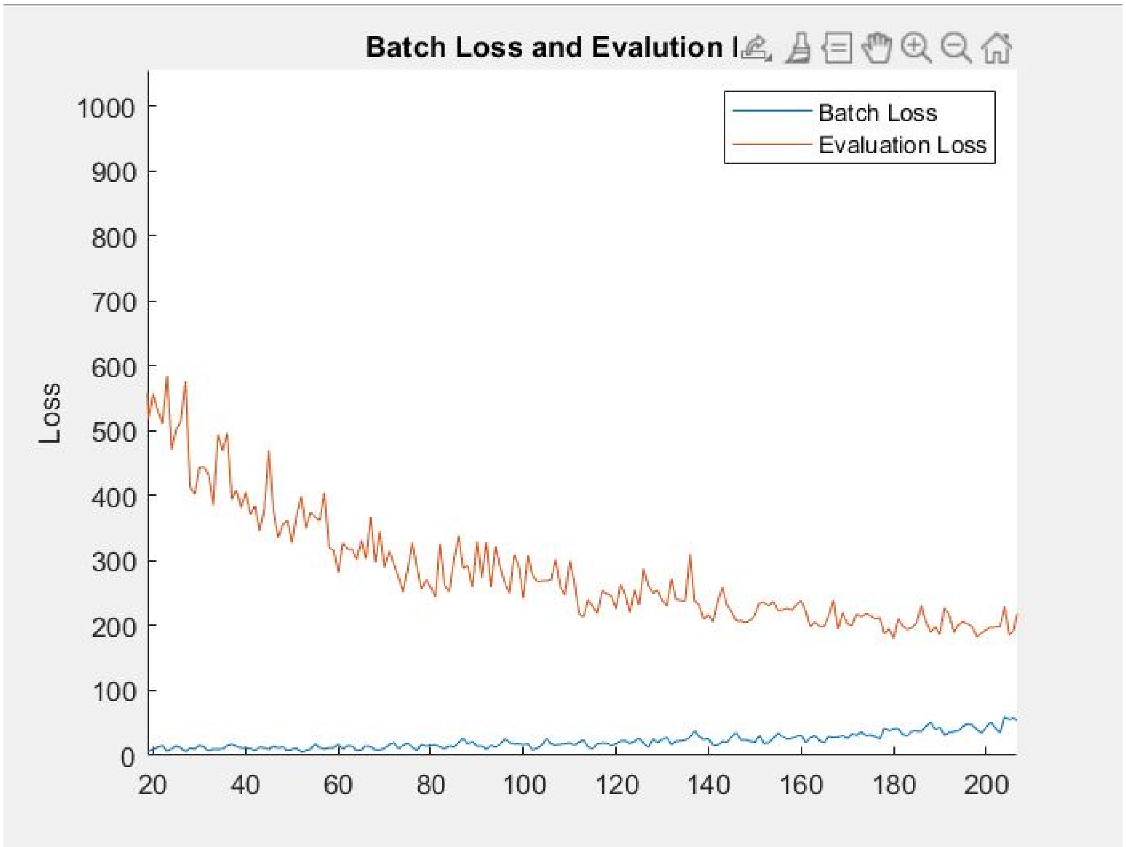


Figure 7: Batch loss and evaluation loss vs. iterations for the multi-layered perceptron neural network

References

[1] Bazargani, Mehran. “The derivative of the Softmax function w.r.t its inputs”.ML Dawn, 27 November 2021, `https://www.mldawn.com/the-derivative-of-softmaxz-function-w-r-t-z/`

[2] Brownlee, Jason. “A Gentle Introduction to the Rectified Linear Unit (ReLU)”. Machine Learning Mastery, 9 January 2019, `https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/`

[3] Ding, Caiwen, et. al. 2017. "CirCNN: Accelerating and Compressing Deep Neural Networks Using Block-Circulant Weight Matrices".In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 ’17). Association for Computing Machinery, New York, NY, USA, 395–408. `https://doi.org/10.1145/3123939.3124552`

[4]Kurbiel, Thomas. “Derivative of the Softmax Function and the Categorical Cross-Entropy Loss”. Towards Data Science, 22 April 2021, `https://towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1`

A Appendix

`https://github.com/hassnain360/EC503_Circulant_NN`