Introduction to Reinforcement Learning & Deep Learning

CSI 5340

Implementing Backward Propagation & Neural Networks for Classification of MNIST data

Assignment 2

Authors
Alexandra Sklokin (300010511)

Professor
Runzhi Tian

October 10, 2022

Contents

1 Question 1

1.1 Introduction

In this report I will discuss the implementation of Backward Propagation for the proposed forward feedforward neural network structure. Parameter gradients, obtained by Backward Propagation, were compared to those calculated using the *torch* library. Gradient Descent (GD) was used to train the parameters of the network, matrices A, B, and C.

1.2 Methodology

For question 1 of this assignment I used *jupyter notebooks* and Excel spreadsheets. Please find all of the code for this assignment in my Git repository[1].

$Question1.ipynb$ contains all of the code as required.

To use this notebook, begin by setting the parameters of the notebook:

- K: length of x, and dimension of A, B, and C.
- *min_range*: min value for x and matrix random entries.
- *max_range*: max value for x and matrix random entries.
- N: number of data points for Gradient Descent algorithm.
- *epochs*: number of iterations of Gradient Descent algorithm.

I have created helper methods for matrix-vector multiplication, vector addition, matrix addition, matrix-scalar multiplication, sigmoid, 2norm calculations, etc.

1.3 Implementation

1.3.1 Forward Propagation

Forward propagation was simple to implement, by using the prescribed equations. These values will be used to calculate neuron outputs for Backward Propagation ($a_j^{(l)}$ and $h_j^{(l)}$).

y, u, v, z, and w are vectors of length K, and L describes the loss of the network.

[1] https://github.com/alexandrasklokin/CSI5340/tree/main/CSI5340_A2

$$\begin{aligned}
y &:= Ax \\
u &:= \sigma(y) \\
v &:= Bx \\
z &:= (u+v) \\
w &:= Cz \\
\mathcal{L} &:= \|w\|^2
\end{aligned}$$

1.3.2 Backward Propagation

I had difficulty implementing the Back Propagation algorithm. I was able to compute the $a_j^{(l)}$ and $h_j^{(l)}$ values for all layers, and attempted to match my gradient results to the calculations from *torch*. I used the following equation to obtain the Back Propagation gradients:

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} = (\, g'(a_j^{(l)}) \sum_k \frac{\partial \mathcal{L}}{\partial a_k^{(l+1)}} w_{kj}\,)\, h_i^{(l-1)}$$

1.3.3 Backward Propagation (Torch)

To confirm the results from my implementation of Backward Propagation, I used the *torch* library's method *tensor.backward*() and computed the gradient of L with respect to our matrices A, B, and C.

1.3.4 Gradient Descent

For model parameter (A, B, and C) training, I used GD. I began by initializing the matrices to random values, and updating them by subtracting the learning rate times the gradients with respect to L, obtained through Back Propagation.

1.4 Experimental Results

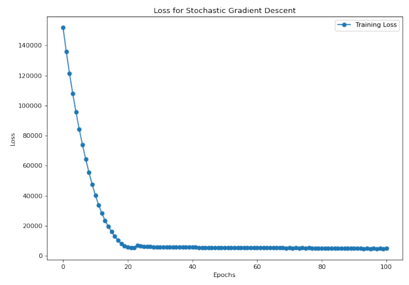


Figure 1: Graph of Training Loss for Gradient Descent.

1.5 Discussion

Figure [1] shows the training loss of my GD implementation, over the epochs (with learning rate $\lambda = 0.1$). From the decreasing trend of the training loss, we can observe how Gradient Descent is able to minimize the loss for this network model.

It is observed that the model has significant improvement in performance up until 20 epochs, after which there is a plateau.

1.6 Conclusion

In conclusion, I was able to understand a general approach to implementing Back Propagation, and its importance for model parameter training in the Gradient Descent algorithm. Within GD, I was able to achieve a smooth decrease in training loss throughout epochs.

2 Question 2

2.1 Introduction

In this report I will discuss the results obtained from training three neural network models on the MNIST dataset. The three models are Softmax Regression (SMR), Multi-Layer Perceptron (MLP), and Convolutional Neural Network (CNN), ordered by model complexity. Model performance is compared when training with/without Dropout and Batch Normalization.

2.2 Methodology

For question 2 of this assignment I used *jupyter notebooks* and Excel spreadsheets. Please find all of the code for this assignment in my Git repository[2].

$Question2\_SMR.ipynb$, $Question2\_MLP.ipynb$, and $Question2\_CNN.ipynb$ contain all of the code as required.

2.3 MNIST Dataset

The Modified National Institute of Standards and Technology (MNIST) dataset used in this assignment contains handwritten digits, and can be used to train neural networks for the purpose of image recognition. This dataset contained 60,000 training points, and 10,000 testing points.

I expect to achieve very high accuracies on the MNIST dataset, as it does not well replicate a real-world digit classification problem. This dataset has been constructed with virtually no noise (ex. no distortions or foreign shapes, etc.). Additionally, only specific variations of digits are used, which may not be representative of all the possible ways a human will write such a digit. For example, most/all 1's in MNIST are a straight line, and have no curve at the top. If I were to write down my own 1 digit, with a curve, then any neural network trained on this dataset would likely output a 7. Nevertheless, we can use this dataset to learn about NN implementation and performance.

2.4 Models

2.4.1 Softmax Regression

The Softmax Regression (SMR) model is an extension of Logistic Regression for the multi-classification problem. In our problem, we are trying to classify hand-written digits $\{0, 1, 2, 3, ..., 9\}$. Training is done using Gradient Descent. The SMR model works well when data is linear-separable. This is a 'simple'

[2]https://github.com/alexandrasklokin/CSI5340/tree/main/CSI5340_A2

model, meaning that we could observe lower training and testing accuracies (underfitting), but it will likely not be prone to overfitting.

2.4.2 Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) model is a fully connected Neural Network (NN) where training is conducted using Back Propagation. This model can contain any number of hidden layers, an input, and an output layer. The MLP may be better suited for non-linearly separable datasets, since we can utilize non-linear activation functions between layers of the NN.

2.4.3 Convolution Neural Network

The Convolutional Neural Network (CNN) model is a type of NN commonly used for image recognition problems. Similarly to MLPs, CNN will contain hidden layers (including convolutional and pooling layers), an input and an output layer. To minimize overfitting, CNNs are not fully connected, but rather utilize kernels to learn hierarchical structure in data. Convolutional layers work by sliding the kernel along the input data and performing the convolution between the kernel and the input data.

2.4.4 'Best' Model

In the attempt to achieve best results, I trained $M = 50$ SMR models, $M = 50$ MLP models, and $M = 10$ CNN models. For each model type, I selected the model (from the M repetitions) with the highest testing accuracy. I repeated this with Batch Normalization, with Dropout, and with both regularization methods.

I collected the accuracies, F1-scores, and training times (seconds) of these 'best' models. In this report, I will be commenting on the results of these 'best' models.

A possible extension of this assignment would have been to comment on the 'mean' model, or the 'mode' model, by using ensembles of the M neural networks. Ensembles can be used to greatly improve generalization results.

2.5 Regularization

2.5.1 Dropout

Dropout is a NN regularization method to reduce overfitting and increase generalization. In this method, we 'drop' neurons from our model during training, in order to train many different network architectures (at low computational cost).

Dropout can be used in 1, some, or all layers of the NN. It is usually characterized by a parameter p which identifies the probability of a node being dropped. This parameter can be different for each layer in which dropout is used. If dropout is too high, we can lose important information from the input data, so it must be chosen sufficiently small (p=0.1 as a common choice).

Dropout is a suitable method for large and complex networks, which are prone to overfitting. Thus, dropout may not be appropriate for the SMR and MLP model. This method is also useful for smaller and harder to fit datasets, where overfitting is a common problem with NN models.

2.5.2 Batch Normalization

Batch Normalization is a NN regularization technique also aimed to reduce overfitting and increase model generalization. In this method, data is normalized (subtract mean and divide by square root of variance, from a 'representative' batch of the input), scaled and shifted.

This can also be applied to any number of layers of the NN, usually hidden. Batch size, scaling and shifting parameters can be chosen or tuned.

Batch Normalization is suitable to speed up and stabilize model training.

2.6 Experimental Results

2.6.1 Accuracies, F1-Score, and Time

Model	Dropout	Batch Norm.	Accuracy	F1-Score	Training Time (sec)
SMR	1	1	0.9396	0.9395	32.04
	0	1	0.9370	0.9370	26.61
	1	0	0.9107	0.9104	30.08
	0	0	0.8987	0.8984	21.13
MLP	1	1	0.9812	0.9812	16.16
	0	1	0.9820	0.9820	15.86
	1	0	0.9574	0.9574	13.02
	0	0	0.9712	0.9714	13.02
CNN	1	1	0.9937	0.9936	1522.77
	0	1	0.9923	0.9923	1468.21
	1	0	0.9910	0.9910	1410.67
	0	0	0.9873	0.9872	1364.82

Table 1: Experimental Training time, Testing Accuracy and F1-Score for Neural Network Models

2.6.2 Training Loss

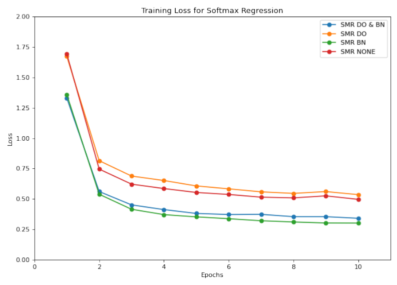


Figure 2: Graph of Training Loss for Softmax Regression Model.

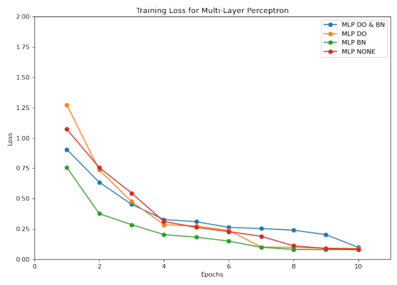


Figure 3: Graph of Training Loss for Multi-Layer Perceptron.

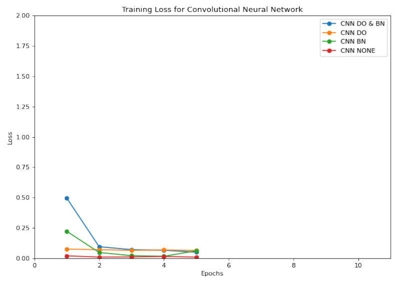
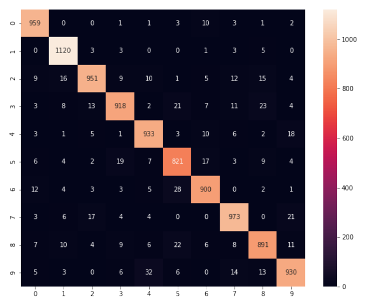
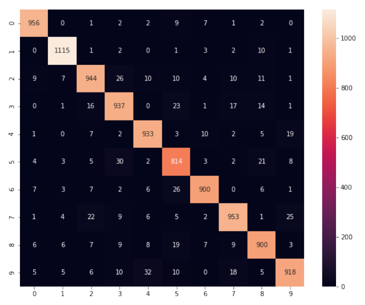
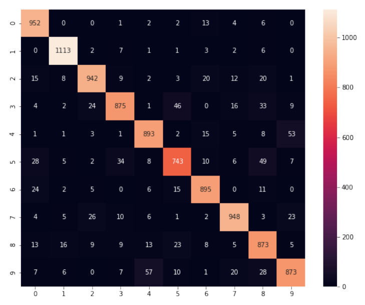


Figure 4: Graph of Training Loss for Convolution Neural Network.

2.6.3 Confusion Matrices


(a) DO & BN


(b) BN


(c) DO

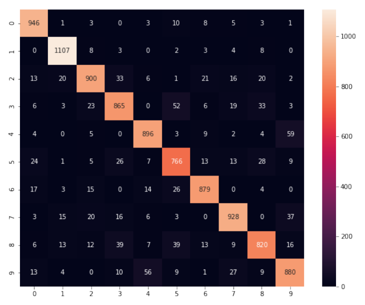
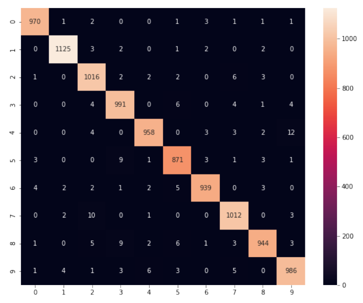
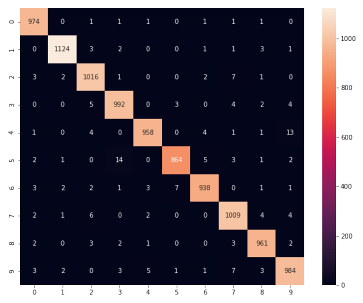
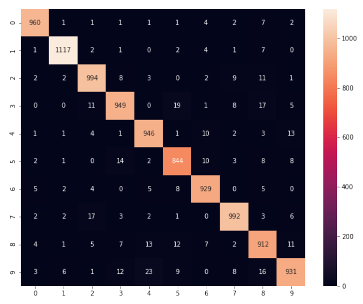

(d) None

Figure 5: Confusion Matrices for Softmax Regression Model on Testing Data


(a) DO & BN


(b) BN


(c) DO

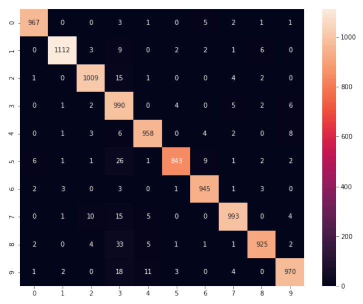
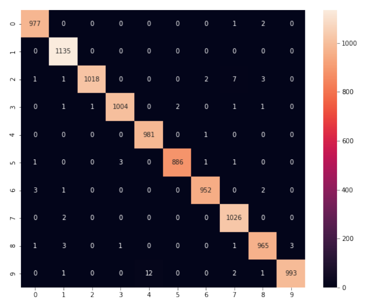
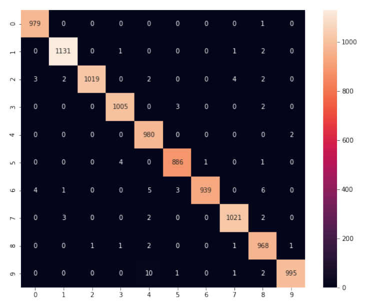

(d) None

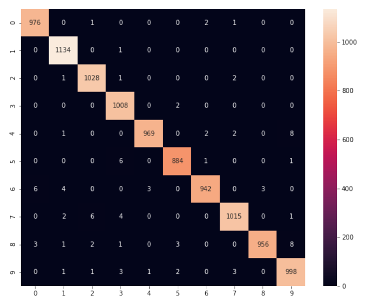
Figure 6: Confusion Matrices for Multi-Layer Perceptron Model on Testing Data



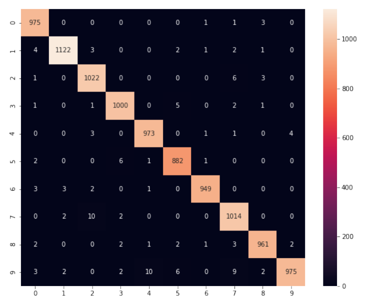
(a) DO & BN



(b) BN



(c) DO



(d) None

Figure 7: Confusion Matrices for Convolution Neural Network Model on Testing Data

2.7 Discussion

2.7.1 SMR vs. MLP vs. CNN

Table [1] shows that all three models achieved great performance, as expected. The CNN models achieved the highest accuracies and F1-scores, across no regularization methods, batch-normalization and dropout. As model complexity grew (SMR to MLP to CNN), testing accuracy and F1-score increased.

However, this did come at the expense of time. CNN took significantly more time to train than SMR and MLP. If accuracy is our priority, then we shall prefer the CNN model. If we would like to save time (ex. real-time machine learning), we may prefer to use the MLP, which was usually without $\pm1\%$ of the CNN accuracies and F1-scores. Finally, we can also prioritize model simplicity and choose the SMR model, which fortunately made correct predictions more than 89% of the time. With the emerging field of Explainable Artificial Intelligence (XAI) and model visualization, simple models have become more desirable by their human users. MLP and CNN models are black-boxes which take some input and give some output- too complex for model visualization.

Thus, machine learning problems ought not to be considered in isolation from the environment in which they will be used. In our academic environment, we call CNN the winner!

Figures [2], [3], and [4] show the graphs of training loss over the epochs. Clearly CNN outperforms the other models, as it begins lower, and trains faster than either SMR or MLP.

Figures [5], [6], and [7] show the confusion matrices for model predictions against the true classification values. Important: MNIST is not uniformly distributed across the ten digits. For all three network types, the '1' digit is predicted the most since it is the largest MNIST group; '5' the least and it is the smallest group. This doesn't mean that the models predicted '1' best and '5' worse, but is rather a result of the distribution of the dataset. These confusion matrices can be used to find the recall and precision of our models.

The weighted F1-score is the harmonic mean of the model recall and precision, weighted based on the distribution of the data. It is used to help choose a 'best' model by finding a compromise for the Recall-Precision Trade-off. By using the weighted statistic, we are also able to adjust this metric according to the distribution of the data. Again, Table [1] shows that the CNN model significantly outperforms SMR and MLP.

2.7.2 Dropout

Table [1] shows that, in comparison to using no regularization strategies, Dropout results in slightly better performance for the SMR and CNN models, and worse for the MLP model. There does not seem to be any significant proof to say that Dropout improves model performance on the MNIST dataset, as implemented in this assignment.

Figures [2], [3], and [4] also do not show any significant evidence of improvement in model training. In fact, SMR training speed (over epochs) seems to have been negatively impacted by Dropout.

2.7.3 Batch Normalization

Table [1] shows that, in comparison to using no regularization strategies, Batch Normalization results in slightly better performance for all models. Therefore, Batch Normalization seems like a suitable regularization technique to be used for training NNs on the MNIST dataset.

Figures [2], [3], and [4] also shows that Batch Normalization results in lower initial training loss and faster training, for the two simpler SMR and MLP models.

2.7.4 Dropout and Batch Normalization

For this assignment, I have also investigated the use of Dropout and Batch Normalization together for NN training. Table [1] seems to show that using both regularization techniques results in the highest accuracies and F1-scores. Additionally, the SMR and MLP training loss curves are lower when using both.

When choosing between Dropout and Batch Normalization, we would prefer to use the later, as it clearly results in better model training and testing performance. However, ideally we would like to use both regularization techniques.

2.8 Conclusion

In conclusion, I was able to train and compare three Neural Network models on the MNIST dataset. I used the Softmax Regression, Multi-Layer Perceptron, and Convolution Neural Network to classify images of digits to the target label. I determined that the CNN model- the most complex model- achieved the highest performance. Additionally, I investigated the effects of Dropout and Batch Normalization on my models, and determined that using both methods in tandem results in the best improvement.