Back Propagation Neural Network for Image Classification

Sarah Simionescu
2G03 Final Write Up, McMaster University
Hamilton, ON, Canada
Email: simiones@mcmaster.ca

I. Introduction

Machine learning algorithms have significantly improved our ability to mathematically model complex real-world problems. These models allow us to automatically perform complex tasks such as image classification [1], object detection [2] and natural language processing [3]. According to MIT professor Aleksandy Madry, "Machine learning is changing, or will change, every industry, and leaders need to understand the basic principles, the potential, and the limitations" [4]. With this project, I aim to dive into the technical details and implement a machine-learning algorithm to classify images of hand-written digits. There exist many different types of machine learning algorithms to accomplish classification, such as K-nearest neighbours [5], decision trees [6] and artificial Neural Networks [7]. For my specific task, I created a standard back propagation Neural Network, the foundation for many popular networks used today.

II. Methods

I created a model with two hidden layers and two respective activation layers. See Fig. 1 for a visualization of the model and Section II-B for a mathematical description.

A. Dataset

I trained and evaluated my model on the standard MNIST dataset [8], which is composed of 28 by 28 pixel images of hand written digits. There were 60000 images in the training set and 10000 images in the evaluation set.

B. Model

Let N be a function $\mathbb{R}^{n\times n} \to \{x \in \mathbb{R} | 0 \le x \le 1\}^m$, Input $I \in \mathbb{R}^{n\times n}$ is the pixel values of an $n \times n$ black and white image.

$$I = \begin{bmatrix} i_{11} & i_{12} & \dots & i_{1n} \\ i_{21} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ i_{n1} & \dots & \dots & i_{nn} \end{bmatrix}$$

Output $A_2 \in \mathbb{R}^m$ is a column vector where m is the number of labels in the training data. Each label has a corresponding index from $[0, m]$. The greater the element at this index in A_2, the stronger the "likelihood" that label m is the correct label.

$$A_2 = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_m \end{bmatrix}$$

N is a composition of many functions. I refer to the output of each of these functions as "layers". As visually demonstrated in Fig. 1, N takes and produces 5 layers:

Input Layer $I \in \mathbb{R}^{n\times n}$ which is simply the input image,

Flattening Layer $F \in \mathbb{R}^{n^2}$ which is the columns of I rearranged into one long column vector,

Hidden Layers $H_1, H_2 \in \mathbb{R}^m$ which compresses the previous layer into a feature via weights $W_1, W_2 \in \mathbb{R}^{n^2\times m}$ and biases $B_1, B_2 \in \mathbb{R}^m$,

Activation Layers $A_1, A_2 \in \mathbb{R}^m$ which applies a non-linear function to the elements of H_1, H_2, allowing us to model more complex functions and analyze our output.

1) Forwards Propagation: To compute a predication given a particular image $N(I) = A_2$ we compute the following:

1) $F = flatten(I)$ where $flatten$ lists all the columns of I in one n^2 column vector
2) $H_1 = W_1 \cdot F + B_1$
3) $A_1 = ReLU(H_1)$ where
$$ReLU(X) = \begin{cases} x_i \text{ if } x_i > 0 \\ 0 \text{ if } x_i \le 0 \end{cases}$$
4) $H_2 = W_2 \cdot A_1 + B_1$
5) $A_2 = softmax(H_2)$ where
$$softmax(X) = \frac{e_i^x}{\sum_{j=1}^K e_j^x}$$

W_1, W_2 are referred to as the weights and B_1, B_2 are referred to as the biases. These are trainable parameters that adapt through back propagation to produce increasingly accurate predictions.

2) Back Propagation: To adjust the weights and biases to obtain increasingly more accurate predictions, we compute the following:

1) Take a batch of k training images $I \in \mathbb{R}^{n\times n\times k}$ and their corresponding hot-encoded training labels $L \in \{1, 0\}^{m\times k}$ where $hot - encode(x)$ is a function $\mathbb{I} \to$

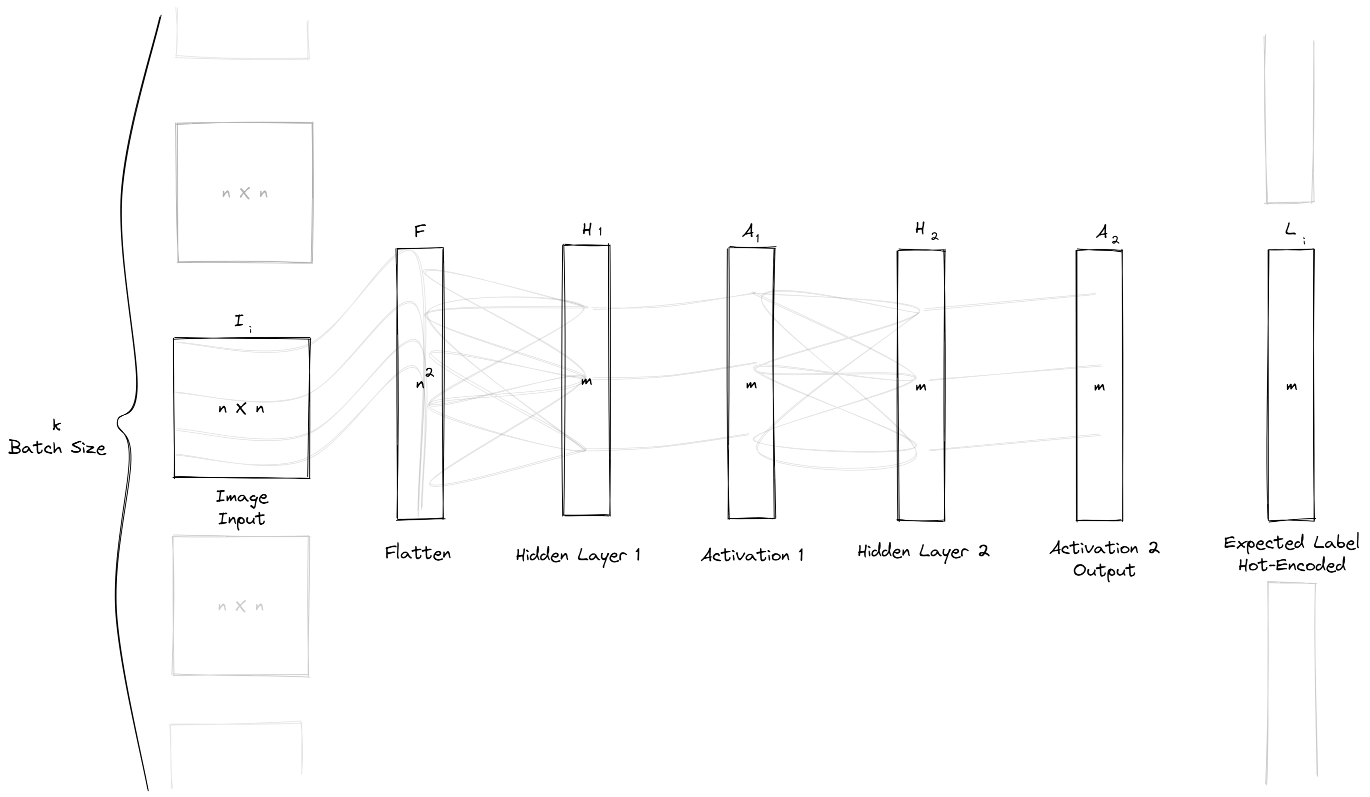


Fig. 1. A visualization of my model. See Section II-B for more details.

$\{1, 0\}^m$ which returns a column vector with all of its elements set to 0, except for the element at index x, which is set to 1

2) Through Forward Propagation, obtain the output for each $i = \{1, 2, \ldots, k\}$ training image $N(I_i)$ and save the intermediate outputs of the layers to obtain $F, H_1, A_1, H_2, A_2 \in \mathbb{R}^{m \times k}$

$$A_2 = \begin{bmatrix} N(I_0) & N(I_1) & \ldots & N(I_k) \end{bmatrix}$$

3) Calculate the error of H_2

$$\delta H_2 = A_2 - L$$

4) Find the derivative of the loss function with the respect to the weights W_2 and biases B_2

$$\delta W_2 = \frac{1}{k} \delta H_2 A_1^T$$

$$\delta B_2 = \frac{1}{k} \sum_{i=1}^{k} \delta H_{2i}$$

5) Calculate the error of H_1

$$\delta H_1 = W_2^T \delta H_2 * ReLU'(F)$$

6) Find the derivative of the loss function with respect to the weights W_1 and biases B_1

$$\delta W_1 = \frac{1}{k} \delta H_1 F^T$$

$$\delta B_1 = \frac{1}{k} \sum_{i=1}^{k} \delta H_{1i}$$

7) Update our parameters

$$W_1 := W_1 - \alpha \delta W_1$$

$$B_1 := B_1 - \alpha \delta B_1$$

$$W_2 := W_2 - \alpha \delta W_2$$

$$B_2 := B_2 - \alpha \delta B_2$$

where α is a hyper parameter called the learning rate

Through this process, we minimize our error δH_2, forcing our model to create a prediction closer to the hot-encoded expected result. After repeating this process over many training batches, our model produces increasingly accurate predictions.

C. Implementation

I implemented my model N in C++ very closely to the mathematical model described in Section II-B with only one key difference. Instead of using a $flatten(I)$ function on the input image, I initially load the images as flattened vectors. This avoids a significant amount of additional computation when training and testing my model.

My implementation is composed of 6 C++ classes.

Dataset: Responsible for loading and storing the MNIST dataset.

Layer: The generic layer class which initializes a layer given an input size and output size.

DenseLayer: Inherits the Layer class and initializes and stores matrices of weights and biases. Its "forward" function (used in Forwards Propagation) computes a weighted sum on the input vector.

ReLULayer: Inherits the Layer class and contains the $ReLU$ function and its derivative. It's "forward" function (used in Forwards Propagation) applies the ReLU function on each individual element in the input vector. It's "backward" function (used in Backwards Propagation) applies the derivative of the ReLU function on each individual element of the given input vector.

SoftMaxLayer: Inherits the Layer class and contains the $SoftMax$ function. It's "forward" function (used in Forwards Propagation) applies the SoftMax function on each individual element in the input vector. During backwards propagation, this output is used to calculate the error relative to our hot-encoded label.

Network: Generates a Neural Network using four Layer objects: A DenseLayer $h1$, ReLULayer $a1$, a second DenseLayer $h2$ and a SoftMaxLayer $a2$. During forward propagation, an input goes through a composition of "forwards" functions from each layer as described in section II-B1. During training, the Network is given a dataset and several training parameters. Given these parameters, the Network performs the calculations described in Section II-B2 and then updates the weights in $h1$ and $h2$ accordingly.

D. Network Parameters

We refer to the output size of $h1$ as the **hidden layer size**, which is a variable parameter when generating the model. The learning rate, as described in Section II-B2, is a constant multiplied by the change in our weights and biases. The batch size is what we referred to as variable k in Section 1, and represents the number of training examples considered per update to the Network. We assume this is a divisor of the total number of examples in our training data, i.e. 60000 [8]. Epochs refer to the number of times we repeat over the entire dataset.

E. Network Accuracy and Overfitting

In addition to the dataset and training parameters, the Network also takes to array pointers which it will fill with the following data at the start of each batch:

- **Accuracy on Training Batch:** What was the model's accuracy over the current training batch? This is data the Network has already been exposed to after the first epoch.
- **Accuracy on Validation Data:** What was the model's accuracy over the validation data? This is data the Network will never be exposed to. It allows us to see if our Network is learning a pattern that can be accurately applied to new data outside of its training data.

We take and compare both of these measurements to check for overfitting. Overfitting occurs when a model's accuracy over its training data is much better than that of its validation data. Intuitively, the model essentially "memorizes" the correct labels to its training data rather than picking up on the general pattern. This is unideal since, in real work applications, we want our model to be robust and make accurate predictions on new data we have never seen before.

III. Tests and Results

I conducted 6 tests to explore how changing training parameters affected the accuracy of my model over its training data and validation data. The parameters I chose for each test are displayed in Table I. Figure III-D graphs the accuracy over each update to the model. Note the total number of updates will be

$$(60000/\text{batch size}) \times \text{epochs}$$

In other words, we split our training data of 60000 [8] images into batches of a given size, and for each epoch, we repeat over those batches.

Test	Hidden Layer Size	Batch Size	Learning Rate	Epochs	Final Accuracy on Training Batch	Final Accuracy on Validation Data
1	10	10000	1	10	76.30%	74.13%
2	100	10000	1	10	87.58%	86.69%
3	50	10000	1	10	86.83%	85.63%
4	10	10000	1	100	92.84%	91.55%
5	10	60000	1	10	49.98%	8.39%
6	10	30000	1	10	44.83%	15.07%

TABLE I
Tests Conducted. Darker blue cells indicate a relatively larger parameter. Green cells indicate a relatively better accuracy while red cells indicate a relatively lower accuracy.

A. Test 1: Default Parameters

I chose a hidden layer of size 10, a batch size of 10000 with 10 epochs as my default parameters since I could quickly train my model to a decent accuracy.

B. Test 2 and 3: Increased Size of Hidden Layer

I tested the default training parameters on two different models with more weights and biases. I chose a hidden layer of size 100 in Test 1 and a hidden layer of size 500 in Test 2. Increasing the hidden layers' size improved my Network's final accuracy without overfitting. We can also see in III-D(Test 2)(Test 3) that my accuracy improved more steadily over time.

C. Test 4: Increased Epochs

I maintained all default training parameters except increased the number of epochs to 100. This improved the accuracy of my model without overfitting. However, as seen in III-D(Test 4), the accuracy did not improve much after about epoch 33. So the additional training and computation time may have been unnecessary.

D. Test 5 and 6: Increased Training Batch Size

I maintained all default training parameters except increased the batch size to 60000 in Test 5 (the size of the entire dataset) and 30000 in Test 6. As can be seen in Table I, the final accuracy was much smaller. Figure III-D(Test 5)(Test 6) displays a significant difference in training batch accuracy and validation data accuracy, indicating that our model has an overfitting issue as described in Section II-E. Intuitively we can reason that since each update to the model is based on the entire dataset, the changes made to the weights and biases are tailored to the dataset's specific details and not to the overall pattern, which may be causing our overfitting issue.

IV. Conclusion

Neural Networks are powerful mathematical tools used to analyze large complex data. We have designed, implemented and tested a standard back propagation Neural Network to classify MNIST dataset images. Adjusting the training parameters and the model's design can help improve the accuracy and prevent overfitting. Therefore, to create effective and robust models, it is essential to run many tests and observe which strategies work best over the given data. New models and techniques are continuously being developed to solve new problems, and it is exciting to see what machine learning will be able to accomplish in the future.

References

[1] T. Guo, J. Dong, H. Li, and Y. Gao, "Simple convolutional neural network on image classification," in *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, 2017, pp. 721–724.

[2] W. Wang, J. Dai, Z. Chen, Z. Huang, Z. Li, X. Zhu, X. Hu, T. Lu, L. Lu, H. Li, X. Wang, and Y. Qiao, "Internimage: Exploring large-scale vision foundation models with deformable convolutions," 2022. [Online]. Available: https://arxiv.org/abs/2211.05778

[3] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: an introduction," *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 544–551, 09 2011. [Online]. Available: https://doi.org/10.1136/amiajnl-2011-000464

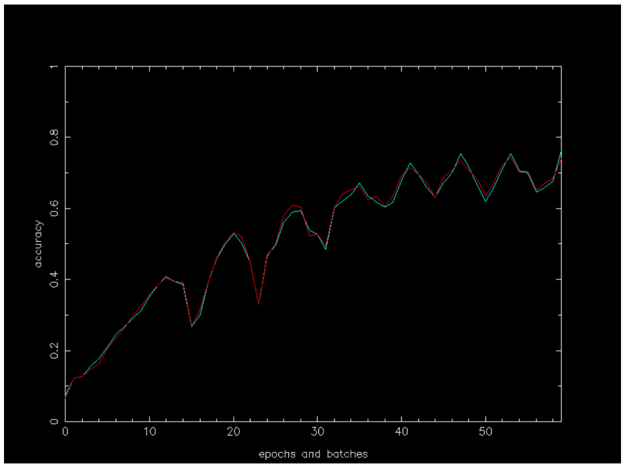
[4] S. Brown, "Machine learning, explained," Apr 2021. [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

[5] "What is the k-nearest neighbors algorithm?" [Online]. Available: https://www.ibm.com/topics/knn

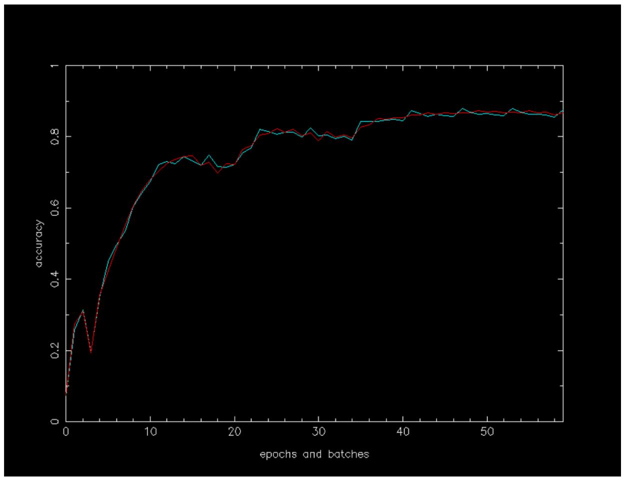
[6] L. Rokach and O. Maimon, *Decision Trees*. Boston, MA: Springer US, 2005, pp. 165–192. [Online]. Available: https://doi.org/10.1007/0-387-25465-X$_9$

[7] S.-H. Han, K. Kim, S. Kim, and Y. C. Youn, "Artificial neural network: Understanding the basic concepts without mathematics," *Dementia and Neurocognitive Disorders*, vol. 17, p. 83, 09 2018.

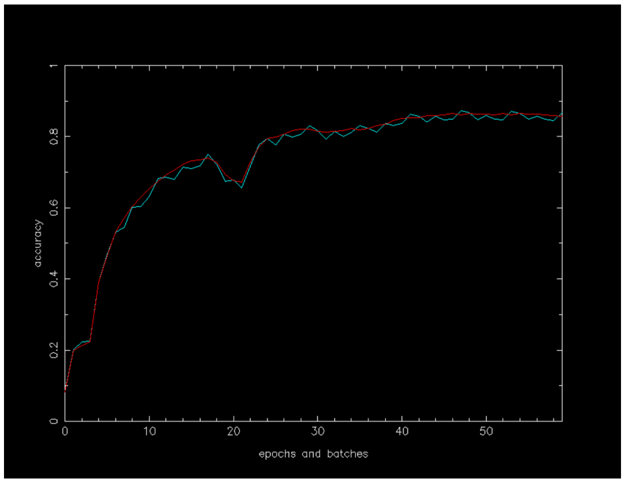
[8] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.



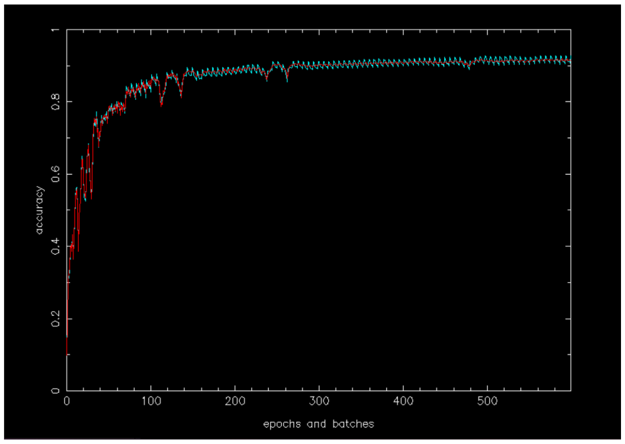
Test 1



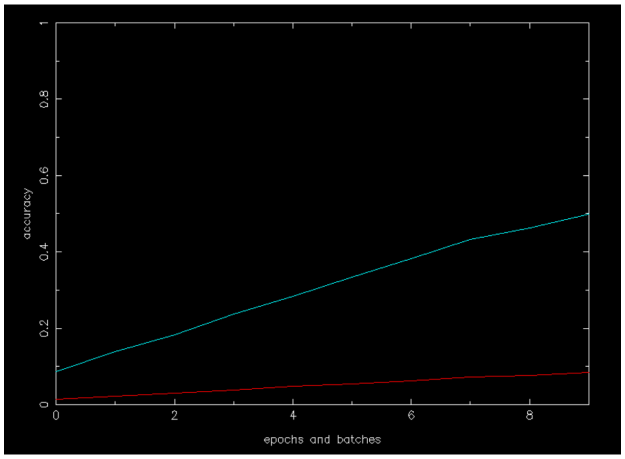
Test 2



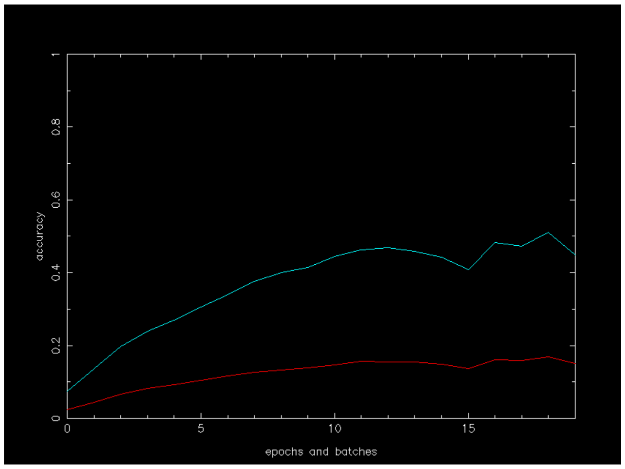
Test 3



Test 4



Test 5



Test 6

Fig. 2. Accuracy results generated from each respective test described in I using PGPLOT. The blue line represents the accuracy of the Network over the training batch and the red line represents the accuracy of the Network over the validation data.