UNIVERSITÀ
DI SIENA 1240

Backpropation experiments on MNIST

Danial Moafi

Machine Learning Course

Professor Marco Gori

2024-2025

Title of the assignment

Based on the Python script on Backpropagation that for experiments MNIST character recognition, extend it properly to address the following requirements:

1. Improve the script by a modular organization and save the weights to a file; Design a module for carrying out the test on the given MNIST csv data on the basis of saved weight files;
2. Discuss appropriate choices of the mini-batches -
3. Discuss the initialization of the weights and the network architecture;
4. Discuss the role of the stopping criterion
5. Design a module for plotting the results

1- Introduction

I implemented the model from scratch. The hidden layers use the **ReLU activation function**, while the **final layer uses Softmax**. The loss function is **CrossEntropy**. The model is a **fully connected neural network**, and I experimented with different architectures, weight initialization methods, and other aspects, which I discuss below. Also, I use the **MNIST dataset** to train models.

2- Results

2.1- Weight Initilization

The first step is weight and bias initialization. Initially, I used random values between 0 and 1. However, this approach caused **overflow** and **underflow** issues due to the exponential nature of the Softmax and CrossEntropy functions. To address this, I explored different weight initialization methods. I tested four approaches: **Zero Initialization, Random (0,1), He Initialization, and Xavier Initialization.** I ran these methods on a simple architecture to compare their performance.

$$\text{HE METHOD} = \frac{2}{\sqrt{\text{input size}}} \qquad \text{XAVIER METHOD} = \frac{2}{\sqrt{\text{input size} + \text{output size}}}$$

NETWORK ARCHITECTURE
OUTPUT_SIZE = 10
NUM_HIDDEN_LAYERS = 2
HIDDEN_SIZE = [100, 100]

ACTIVATION_FUNCTION = 'RELU'
LEARNING_RATE = 0.001
EPOCHS = 100
BATCH_SIZE = 16

Accuracy in different weight Initialization

Method	Ex 1	Ex 2	Ex 3	Ex 4	Ex 5	Avg
zero	OverFlow	OverFlow	OverFlow	OverFlow	OverFlow	-
random	OverFlow	OverFlow	OverFlow	OverFlow	OverFlow	-
He	0.967	0.965	0.967	0.965	0.966	0.966 ± 0.0008
Xavier	0.967	0.967	0.969	0.967	0.967	0.967 ± 0.0007

The He and Xavier methods perform well; however, in our case, they result in minimal differences because adding 10 to the denominator (output size) has little impact when the input size is 784.

*For the remaining experiments, I used the Xavier method.

2.2- Batch Size

In this experiment, I analyzed the **effect of batch size on accuracy using a fixed model** with a simple architecture.

Accuracy in different batch sizes

batch size	Ex 1	Ex 2	Ex 3	Ex 4	Ex 5	Avg
4	overflow	overflow	overflow	overflow	overflow	overflow
8	0.9737	0.9736	0.9737	0.974	0.972	0.973 ± 0.0007
16	0.9669	0.9655	0.9667	0.9666	0.9688	0.967 ± 0.001
32	0.954	0.9558	0.9548	0.9527	0.9548	0.954 ± 0.001
64	0.9367	0.937	0.936	0.935	0.933	0.935 ± 0.0015
128	0.915	0.918	0.917	0.915	0.917	0.917 ± 0.0009

OUTPUT_SIZE = 10
NUM_HIDDEN_LAYERS = 2
HIDDEN_SIZE = [100, 100]
ACTIVATION_FUNCTION = 'RELU'

LEARNING_RATE = 0.001
EPOCHS = 100
WEIGHTS_METHOD = 'XAVIER'

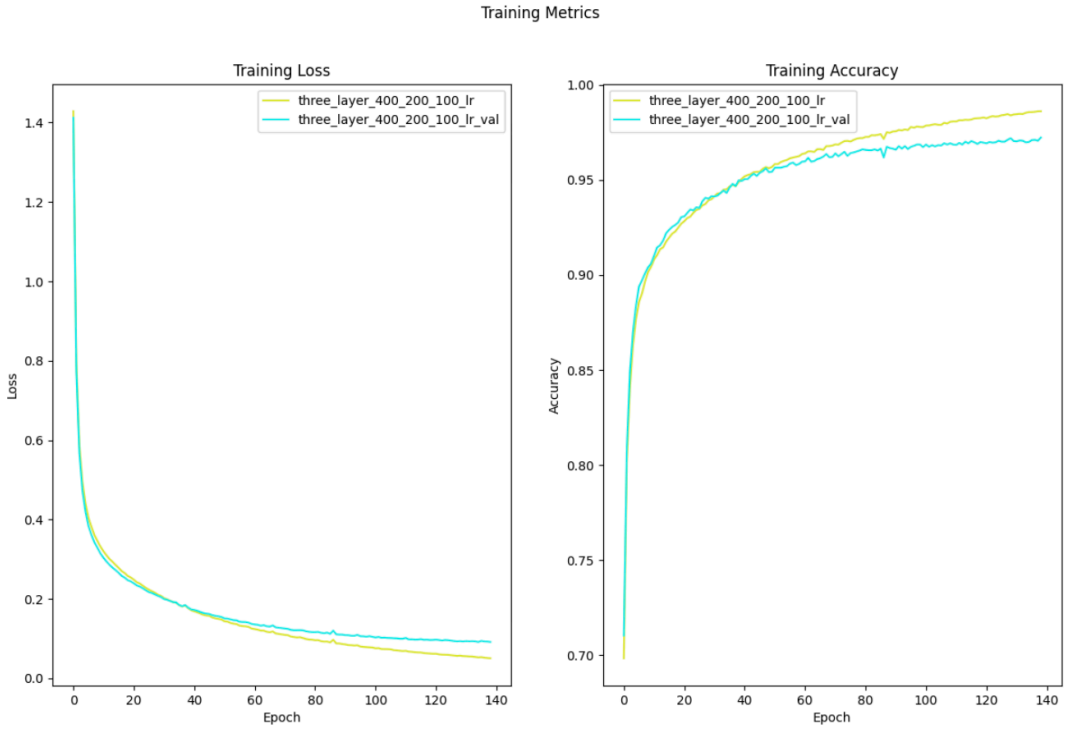
As observed, increasing the batch size in this architecture leads to a decrease in accuracy.

2.3- Network Architecture

I tested three approaches:

1. Using two hidden layers of equal size and increasing their size.
2. Increasing the number of layers while keeping the size of each hidden layer fixed.

3. Assigning different sizes to each layer, starting with larger sizes and decreasing progressively.



BATCH_SIZE = 8
OUTPUT_SIZE = 10
ACTIVATION_FUNCTION = 'RELU'
LEARNING_RATE = 0.001

EPOCHS = 100
WEIGHTS_METHOD = 'XAVIER'
NUM_EXPERIMENTS = 5

Accuracy in different architecture

model_name	ex1	ex2	ex3	ex4	ex5	Avg
md_100	0.970	0.970	0.969	0.969	0.971	0.9698 ± 0.0008
md_100_100	0.974	0.974	0.973	0.972	0.974	0.9736 ± 0.0008
md_100_100_100	overFlow	overFlow	overFlow	overFlow	overFlow	overFlow
md_100_100_100_100	overFlow	overFlow	overFlow	overFlow	overFlow	overFlow
md_200_200	0.975	0.978	0.975	0.977	0.977	0.9768 ± 0.0013
md_400_400	0.977	0.978	0.978	0.979	0.979	0.9784 ± 0.0005
md_800_800	0.979	0.979	0.979	0.979	0.979	0.9791 ± 0.0008
md_400_100_50	overFlow	overFlow	overFlow	overFlow	overFlow	overFlow
md_800_200_100	overFlow	overFlow	overFlow	overFlow	overFlow	overFlow
md_1600_400_200	overFlow	overFlow	overFlow	overFlow	overFlow	overFlow

As observed, I encountered significant overflow issues during this step. To address this, I modified the Softmax function by applying a threshold and also increased the batch size for the more complex architecture.

Compare the architectures using a batch size of 64 and 200 epochs.

model_name	ex1	ex2	ex3	ex4	ex5	Avg
md_400_100_50	0.967	0.965	0.966	0.966	0.966	0.9664 ± 0.0006
md_800_200_100	0.969	0.969	0.972	0.970	0.971	0.9702 ± 0.0011
md_1600_400_200	0.974	0.974	0.973	0.974	0.975	0.9743 ± 0.0006

2.4- Stopping Criterion

In the **StopCriterion class**, I defined **two** methods:

• **Early Stopping**: In this method, I **compare the training loss with the validation loss**. If the training loss is lower than the validation loss for **five consecutive steps,** the training stops.

• **Plateau Method:** Here, I monitor the **standard deviation of the training loss.** If the standard deviation decreases by **0.001 of the mean,** it indicates that **the loss function has plateaued**, and the training is stopped.

* The validation size is set to 10% of the dataset.

3- Visualization

I designed a module to display the loss and accuracy during the training process. By using this function, I can compare and monitor the performance of models side by side.

4- Fine Tuning

In the next step, I selected the best model and tested it with different batch sizes to validate the results.

md_800_800 (Training in 200 epoches with stopping criteria)

batch size : 8 -> Accuracy : 0.970 (Early stopping at the 44th epoch)

batch size : 32 -> Accuracy : 0.9753 (Early stopping at the 153rd epoch)

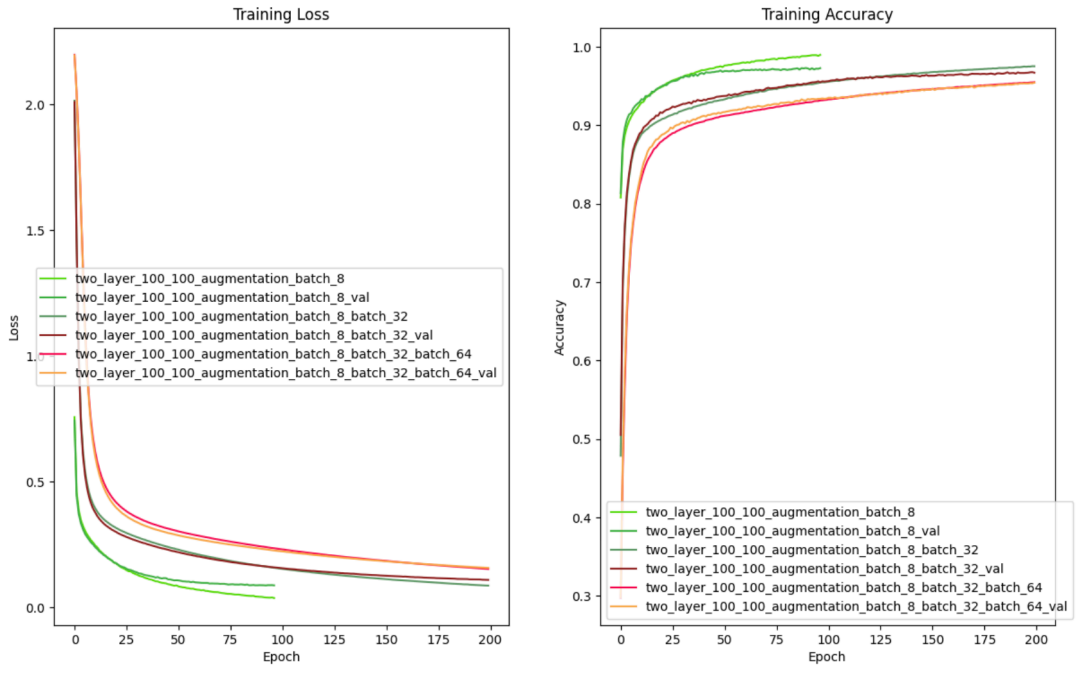
batch size : 64 -> Accuracy : 0.9697

md_100_100

batch size : 8 -> Accuracy : 0.9740 (Early stopping at the 98th epoch)

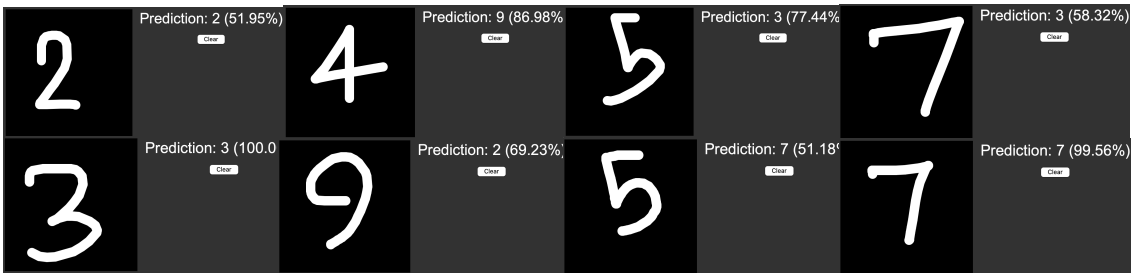
batch size : 32 -> Accuracy : 0.9672

batch size : 64 -> Accuracy : 0.9541



5- Real-Word Test

The result is acceptable, and the next step was to test the model in real-world scenarios. I wrote another module to create an interactive window where users can draw a number using a mouse pad, and the model displays the prediction. I used the best model, named md_100_100, for the prediction. However, the accuracy in this real-world scenario was not as high.



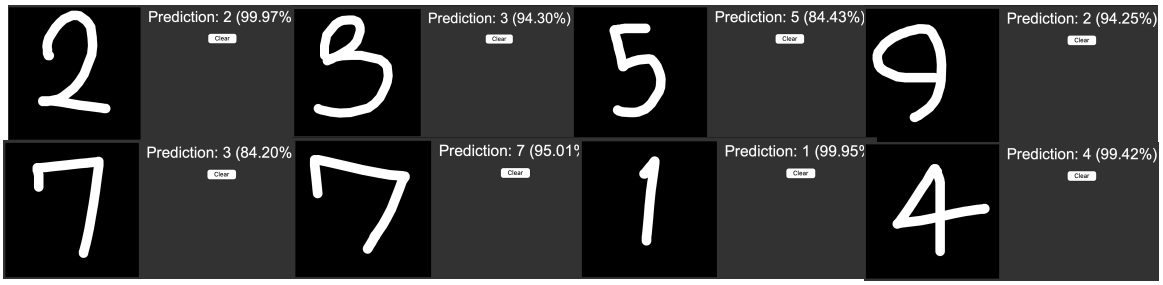
After checking the data and analyzing it further, one reason could be that when I test the model in real-world scenarios, the numbers may be lower or higher, or there may be some rotation. The next step to make the model more robust is to add data augmentation.

6- Data Agumentation

Since the dataset is relatively small, I applied 4 different augmentations to each image and included them in the dataset. These augmentations include zooming in and out, rotation (left and right), blurring ,and adding some black square pixels to the model.

Overall performance did not change with augmentation; however, the model reached the early stopping criterion more quickly.

Accuracy -> 0.9748 (by 36 epoch)



As we can see, the results are slightly better, but this is not an ideal method for comparing models.

7- Conclusion

In this assignment, I wrote a neural network from scratch and compared different augmentations and architectures. The **best performance** was achieved by the model w**ith two fully connected layers, each with 800 neurons**, and with all augmentations added. The final **accuracy** on unseen data was around **0.975.** To further increase performance, it would be better to use **dropout in the network** or **implement a more complex architectur**e, such as a **convolutional neural network.**

Refrenes

1- https://yann.lecun.com/exdb/mnist/

2- https://www.kaggle.com/code/cdeotte/25-million-images-0-99757-mnist

3- https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78

4- https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/

5- https://arxiv.org/abs/2202.03493