Share Comment Star

DA6401 - Assignment 1

Feedforward Neural Network implemented using NumPy

Karan Agrawal cs24m021

Created on March 13 | Last edited on March 16

Problem Statement

In this assignment, I have implemented a feedforward neural network and written the backpropagation code for training the network. I have used NumPy for all matrix/vector operations, as recommended, and have not used any automatic differentiation packages. The network is trained and tested using the Fashion-MNIST dataset. Specifically, given an input image (28 x 28 = 784 pixels) from the Fashion-MNIST dataset, the network is trained to classify the image into one of 10 classes. My code follows the format specified in the Code Specifications section.

Question 1 (2 Marks)

Download the fashion-MNIST dataset and plot 1 sample image for each class as shown in the grid below. Use `from keras.datasets import fashion_mnist` for getting the fashion mnist dataset.

Question 2 (10 Marks)

Implement a feedforward neural network which takes images from the fashion-mnist data as input and outputs a probability distribution over the 10 classes.

Your code should be flexible such that it is easy to change the number of hidden layers and the number of neurons in each hidden layer.

Code is on Github, click here.

Question 3 (24 Marks)

Implement the backpropagation algorithm with support for the following optimisation functions

- sgd
- momentum based gradient descent
- nesterov accelerated gradient descent

- rmsprop
- adam
- nadam

(12 marks for the backpropagation framework and 2 marks for each of the optimisation algorithms above)

We will check the code for implementation and ease of use (e.g., how easy it is to add a new optimisation algorithm such as Eve). Note that the code should be flexible enough to work with different batch sizes.

Code is on Github, click here.

Question 4 (10 Marks)

Use the sweep functionality provided by wandb to find the best values for the hyperparameters listed below. Use the standard train/test split of fashion_mnist (use `(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()`). Keep 10% of the training data aside as validation data for this hyperparameter search. Here are some suggestions for different values to try for hyperparameters. As you can quickly see that this leads to an exponential number of combinations. You will have to think about strategies to do this hyperparameter search efficiently. Check out the options provided by wandb.sweep and write down what strategy you chose and why.

- number of epochs: 5, 10
- number of hidden layers: 3, 4, 5
- size of every hidden layer: 32, 64, 128
- weight decay (L2 regularisation): 0, 0.0005, 0.5
- learning rate: 1e-3, 1 e-4
- optimizer: sgd, momentum, nesterov, rmsprop, adam, nadam
- batch size: 16, 32, 64
- weight initialisation: random, Xavier

- activation functions: sigmoid, tanh, ReLU

wandb will automatically generate the following plots. Paste these plots below using the "Add Panel to Report" feature. Make sure you use meaningful names for each sweep (e.g. hl_3_bs_16_ac_tanh to indicate that there were 3 hidden layers, batch size was 16 and activation function was ReLU) instead of using the default names (whole-sweep, kind-sweep) given by wandb.

Here are the plots:

Question 5 (5 marks)

We would like to see the best accuracy on the validation set across all the models that you train.

wandb automatically generates this plot which summarises the test accuracy of all the models that you tested. Please paste this plot below using the "Add Panel to Report" feature

Question 6 (20 Marks)

Based on the different experiments that you have run we want you to make some inferences about which configurations worked and which did not.

Here again, wandb automatically generates a "Parallel co-ordinates plot" and a "correlation summary" as shown below. Learn about a

"Parallel co-ordinates plot" and how to read it.

By looking at the plots that you get, write down some interesting observations (simple bullet points but should be insightful). You can also refer to the plot in Question 5 while writing these insights. For example, in the above sample plot there are many configurations which give less than 65% accuracy. I would like to zoom into those and see what is happening.

I would also like to see a recommendation for what configuration to use to get close to 95% accuracy.

Observations from the Data:

1. Activation Functions:

• ReLU and tanh give the best accuracy (85–88%), while sigmoid performs poorly (<80%).

• Poor initialization with sigmoid can drop accuracy to 10% due to vanishing gradients.

2. Optimizers:

• Nadam, Adam, and RMSprop work best (up to 87.96% accuracy).

• SGD and Momentum often fail to converge (as low as 10% accuracy).

3. Batch Size:

• Smaller batch sizes (16) perform better (87.96% accuracy) than larger ones (~85% for batch size 64).

4. Initialization:

• Xavier initialization improves stability.

• Random initialization lowers accuracy by 10–20% (87.14% → 65.6%).

5. Hidden Layers:

   • 128-unit layers outperform smaller ones (32 or 64 units).

   • Using 4–5 layers with ReLU/tanh and Xavier initialization gives the best results.

6. Epochs:

   • 10 epochs work better than 5, but more may be needed for further improvement.

How to Reach ~95% Accuracy:

1. Use RMSprop or Nadam (best optimizers).
2. Choose ReLU or tanh (avoid sigmoid).
3. Stick with Xavier initialization for stability.
4. Use batch size of 16 for better generalization.
5. Set learning rate to 0.0001, then fine-tune it.
6. Train with 4–5 hidden layers of 128 units each.
7. Increase epochs to 20–30 for better learning.

But, the highest accuracy recorded is 87.96%. To reach 95%, these changes might not work, we may need major architectural changes.

Plots are attached below:

Question 7 (10 Marks)

For the best model identified above, report the accuracy on the test set of fashion_mnist and plot the confusion matrix as shown below. More marks for creativity (less marks for producing the plot shown below as it is)

I got the best accuracy on the Fashion MNIST dataset for the following configuration:

- Optimizer: RMSprop
- Number of hidden layers: 4
- Size of each hidden layer: 128
- Batch size: 16
- Number of epochs: 10
- Activation function: ReLU
- Learning rate (eta): 0.0001
- Weight Decay: 0.0005
- Beta1: 0.9
- Beta2: 0.999
- Epsilon: 1e-8
- Weight initialization: Xavier

Test accuracy was **87.96%** and Validation accuracy was **89.25%**.

Here is the confusion matrix for the same:

Question 8 (5 Marks)

In all the models above you would have used cross entropy loss. Now compare the cross entropy loss with the squared error loss. I would again like to see some automatically generated plots or your own plots to convince me whether one is better than the other.

I did similar experiments with Mean Squared Error (MSE) loss, and the best test accuracy I achieved was 86.26% with the following configuration:

- Optimizer: RMSprop
- Number of hidden layers: 4
- Size of each hidden layer: 128
- Batch size: 16
- Number of epochs: 10
- Activation function: ReLU
- Learning rate (eta): 0.0001
- Weight initialization: Xavier
- Dataset: Fashion-MNIST
- Loss function: Mean Squared Error (MSE)

From the below attached plots, it is evident that Cross-Entropy Loss performs slightly better than Mean Squared Error (MSE) Loss for the classification task. The test accuracy achieved with Cross-Entropy is higher, indicating that it is more effective in distinguishing between different classes. The training and validation accuracy curves further reinforce this, as the Cross-Entropy model shows steady improvement over epochs, whereas the MSE model fluctuates and struggles to reach similar accuracy levels. Additionally, the loss curves highlight that Cross-Entropy Loss leads to smoother and more stable convergence, while MSE Loss exhibits instability, flattening early and even increasing slightly at times.

Also, it can be compared from the two below plots that, models trained with Cross Entropy tend to reach and maintain higher accuracy levels more consistently. The MSE-based models exhibit more fluctuation, with a noticeable portion of lower accuracy results. This suggests that Cross Entropy might be a better loss function choice for optimizing test accuracy in this scenario.

Question 9 (10 Marks)

Paste a link to your github code for this assignment

GitHub Repository:
https://github.com/karan757527/DA6401_Assignment1/tree/main

Wandb Report Link: https://wandb.ai/cs24m021-iit-madras/DA6401_DL_Assignment1/reports/DA6401-Assignment-1--VmlldzoxMTc4Mzg1Mw

DA6401 - Assignm...
Feedforward Neural ...

Question 10 (10 Marks)

Based on your learnings above, give me 3 recommendations for what would work for the MNIST dataset (not Fashion-MNIST). Just to be clear, I am asking you to take your learnings based on extensive experimentation with one dataset and see if these learnings help on another dataset. If I give you a budget of running only 3 hyperparameter configurations as opposed to the large number of experiments you have run above then which 3 would you use and why. Report the accuracies that you obtain using these 3 configurations.

I first tried out the best configuration I got for Fashion-MNIST, i.e. :

- Optimizer: RMSprop
- Number of hidden layers: 4

- Size of each hidden layer: 128
- Batch size: 16
- Number of epochs: 10
- Activation function: ReLU
- Learning rate (eta): 0.0001
- Weight Decay: 0.0005
- Beta1: 0.9
- Beta2: 0.999
- Epsilon: 1e-8
- Weight initialization: Xavier

and I found Test accuracy to be 97.44%.

Then, for the second configuration, I decided to make adjustments to further optimize for the MNIST dataset. Since MNIST is a slightly simpler dataset compared to Fashion-MNIST, I aimed to test whether reducing model complexity and switching to adam optimizer (as adam adapts learning rates, it can speed up convergence while maintaining stability) would yield similar or better results. Configuration was:

- Optimizer: Adam
- Number of hidden layers: 3
- Size of each hidden layer: 128
- Batch size: 32
- Number of epochs: 10
- Activation function: ReLU
- Learning rate (eta): 0.001
- Weight Decay: 0.0005

- Beta1: 0.9
- Beta2: 0.999
- Epsilon: 1e-8
- Weight initialization: Xavier

and the Test accuracy for above configuration was 97.52%.

Then, for the third and final configuration, I decided to experiment with SGD with Momentum. Given that MNIST is relatively simple, I wanted to test whether a more traditional optimizer with momentum could match or even outperform adaptive optimizers like RMSprop and Adam. In Fashion-MNIST I noticed that SGD with Momentum often generalizes well, avoids excessive parameter updates, and can lead to better long-term performance. It's configuration was:

- Optimizer: SGD with Momentum
- Number of hidden layers: 3
- Size of each hidden layer: 64
- Batch size: 32
- Number of epochs: 10
- Activation function: Tanh
- Learning rate (eta): 0.01 (higher learning rate since momentum helps stabilize updates)
- Momentum: 0.9
- Weight Decay: 0.0005
- Weight initialization: Xavier

For this configuration, Test accuracy was 97.20%. The slightly lower accuracy could be due to SGD converging slower than adaptive optimizers like Adam and RMSprop, which might improve by increasing number of epochs.

Code Specifications

Please ensure you add all the code used to run your experiments in the GitHub repository.

You must also provide a python script called `train.py` in the root directory of your GitHub repository that accepts the following command line arguments with the specified values -

We will check your code for implementation and ease of use. We will also verify your code works by running the following command and checking wandb logs generated -

```
python train.py --wandb_entity myname --wandb_project myprojectname
```

Arguments to be supported

Name	Default Value	Description
`-wp`, `--wandb_project`	myprojectname	Project name used to track experiments in Weights & Biases dashboard
`-we`, `--wandb_entity`	myname	Wandb Entity used to track experiments in the Weights & Biases dashboard.
`-d`, `--dataset`	fashion_mnist	choices: ["mnist", "fashion_mnist"]
`-e`, `--epochs`	1	Number of epochs to train neural network.
`-b`, `--batch_size`	4	Batch size used to train neural network.
`-l`, `--loss`	cross_entropy	choices: ["mean_squared_error", "cross_entropy"]
`-o`, `--optimizer`	sgd	choices: ["sgd", "momentum", "nag", "rmsprop", "adam", "nadam"]
`-lr`, `--learning_rate`	0.1	Learning rate used to optimize model parameters
`-m`, `--`		Momentum used by momentum and nag

`momentum`	0.5	optimizers.
`-beta`, `--beta`	0.5	Beta used by rmsprop optimizer
`-beta1`, `--beta1`	0.5	Beta1 used by adam and nadam optimizers.
`-beta2`, `--beta2`	0.5	Beta2 used by adam and nadam optimizers.
`-eps`, `--epsilon`	0.000001	Epsilon used by optimizers.
`-w_d`, `--weight_decay`	.0	Weight decay used by optimizers.
`-w_i`, `--weight_init`	random	choices: ["random", "Xavier"]
`-nhl`, `--num_layers`	1	Number of hidden layers used in feedforward neural network.
`-sz`, `--hidden_size`	4	Number of hidden neurons in a feedforward layer.
`-a`, `--activation`	sigmoid	choices: ["identity", "sigmoid", "tanh", "ReLU"]

<br>

Please set the default hyperparameters to the values that give you your best validation accuracy. (Hint: Refer to the Wandb sweeps conducted.)

You may also add additional arguments with appropriate default values.

Uploaded on GitHub, click here.

Self Declaration

I, **Karan Agrawal**, swear on my honour that I have written the code and the report by myself and have not copied it from the internet or other students.

https://wandb.ai/cs24m021-iit-madras/DA6401_DL_Assignment1/reports/DA6401-Assignment-1--VmlldzoxMTc4Mzg1Mw