DA6401 - Assignment 1 Report

Roll No : CS24M047

Dhanush

Created on March 9 | Last edited on March 18

Problem Statement

In this assignment you need to implement a feedforward neural network and write the backpropagation code for training the network. We strongly recommend using numpy for all matrix/vector operations. You are not allowed to use any automatic differentiation packages. This network will be 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 will be trained to classify the image into 1 of 10 classes.

Your code will have to follow 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.

- I implemented a flexible feedforward neural network capable of handling various input sizes, including the 28x28 pixel images from the Fashion-MNIST dataset, which are flattened into 784-dimensional vectors into 10 different classes.
- But , it can be easily adapted for different number of input sizes , output classes by modifying the "input_size","no_of_classes" variables respectively.
- The architecture allows for customization of the number of hidden layers and the number of neurons in each layer, making it versatile for different problem sizes and complexities. Additionally, the network supports various activation functions, weight initialization methods, and optimization algorithms, ensuring adaptability to different hyperparameter configurations.
- This flexibility ensures that the model can be tailored to a wide range of tasks beyond Fashion-MNIST.

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.

- I implemented the backpropagation algorithm to train the feedforward neural network, supporting multiple optimization techniques, including Stochastic Gradient Descent (SGD), Momentum-based Gradient Descent, Nesterov Accelerated Gradient Descent, RMSProp, Adam, and Nadam.
- The backpropagation framework is designed to handle different batch sizes, making it efficient for both small and large datasets.
- Each optimizer was implemented with its respective update rules, such as momentum terms for Momentum and Nesterov, adaptive learning rates for RMSProp, and bias-corrected estimates for Adam and Nadam, ensuring flexibility and adaptability to different training scenarios.
- The code is structured to allow easy integration of new optimization algorithms, ensuring flexibility and scalability for future enhancements.
- This implementation ensures that the network can be trained effectively using the most suitable optimization strategy for the given task.

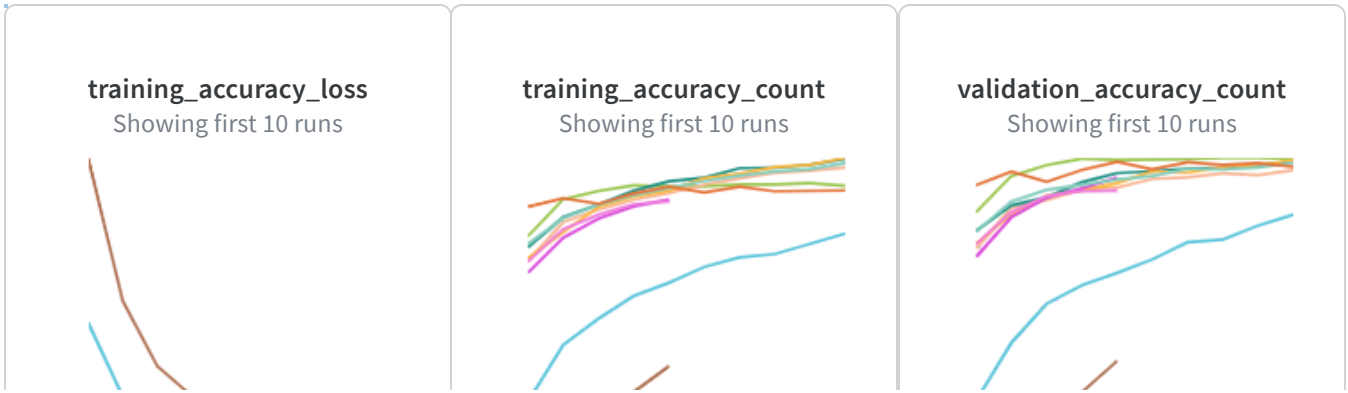
Question 4 (10 Marks)

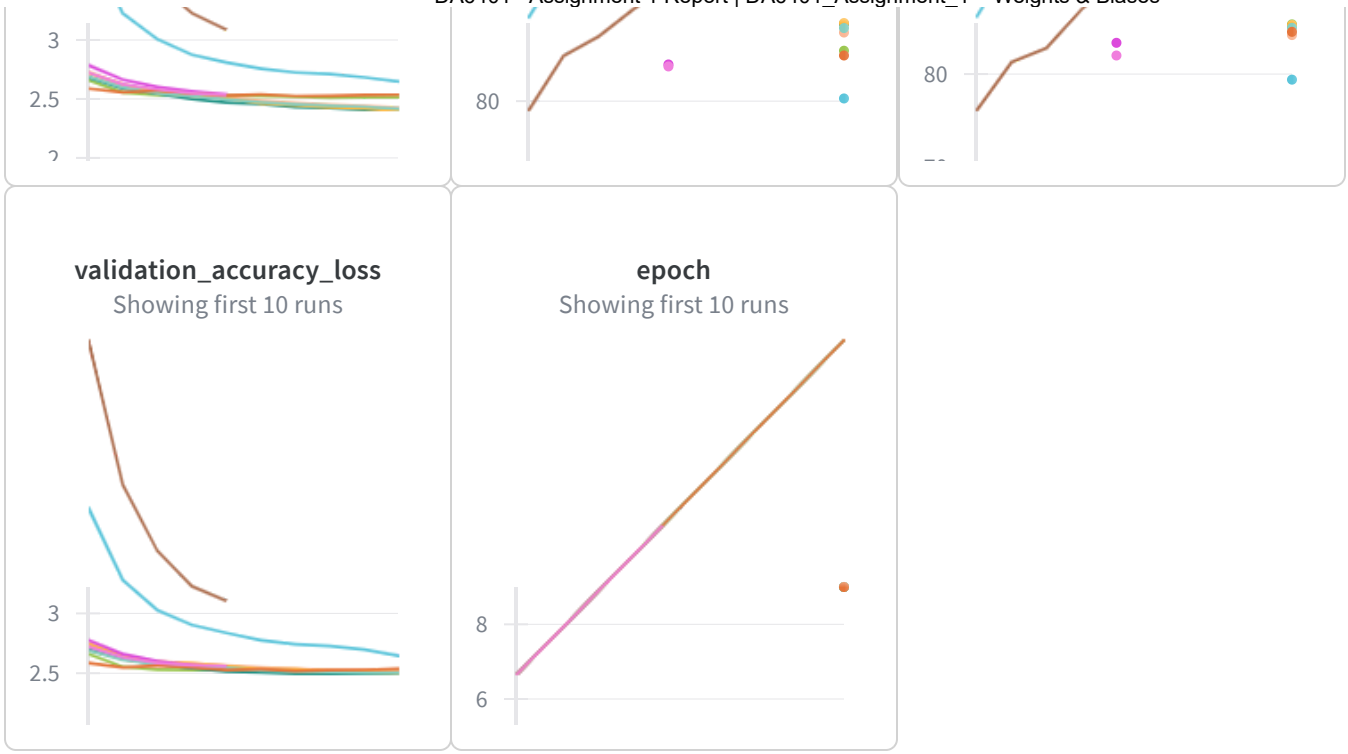
Use the sweep functionality provided by wandb to find the best values for the hyperparameters listed below. Use the standard train/test split of fashion_mnist (use `(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()`). Keep 10% of the training data aside as validation data for this hyperparameter 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.

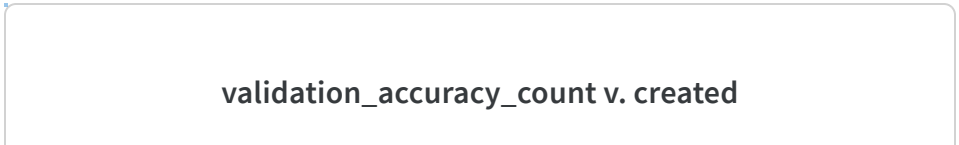


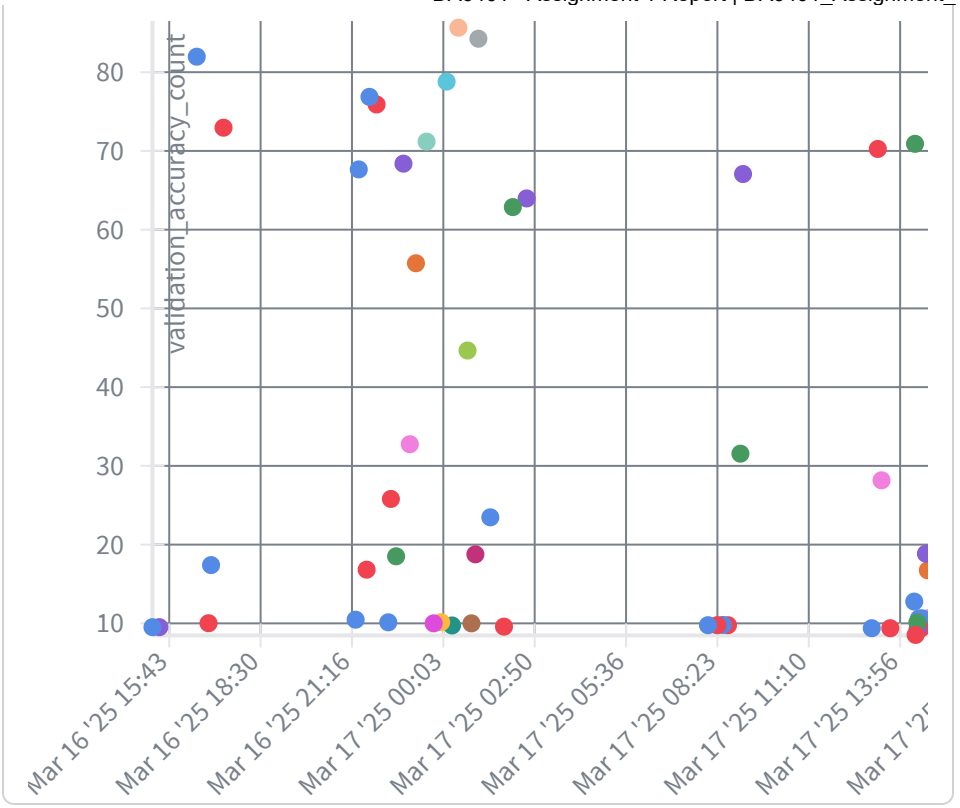


Question 5 (5 marks)

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.

Answer :

Multiple Low-Accuracy Runs:

- High or Very Low Learning Rates leading to unstable or underfit models.
- Inappropriate Optimizer Choices (e.g., plain SGD without momentum) or too few epochs.

Optimizer Importance:

- The correlation summary suggests that using Adam or Nadam has a positive correlation with higher accuracy. These optimizers help converge faster and handle noisy gradients better than basic SGD.

Learning Rate and Weight Decay.

- A moderate learning rate (e.g., 0.001) is consistently linked to better performance.
- Weight decay (regularization) can be helpful but must be tuned. Very high decay can underfit; very low/no decay can risk overfitting.

Activation Functions:

- Models using ReLU or Sigmoid with a suitable learning rate often show higher validation accuracy.
- However, certain combinations of activation and optimizer can underperform if other hyperparameters (like learning rate) are mismatched.

Batch Size and Epochs:

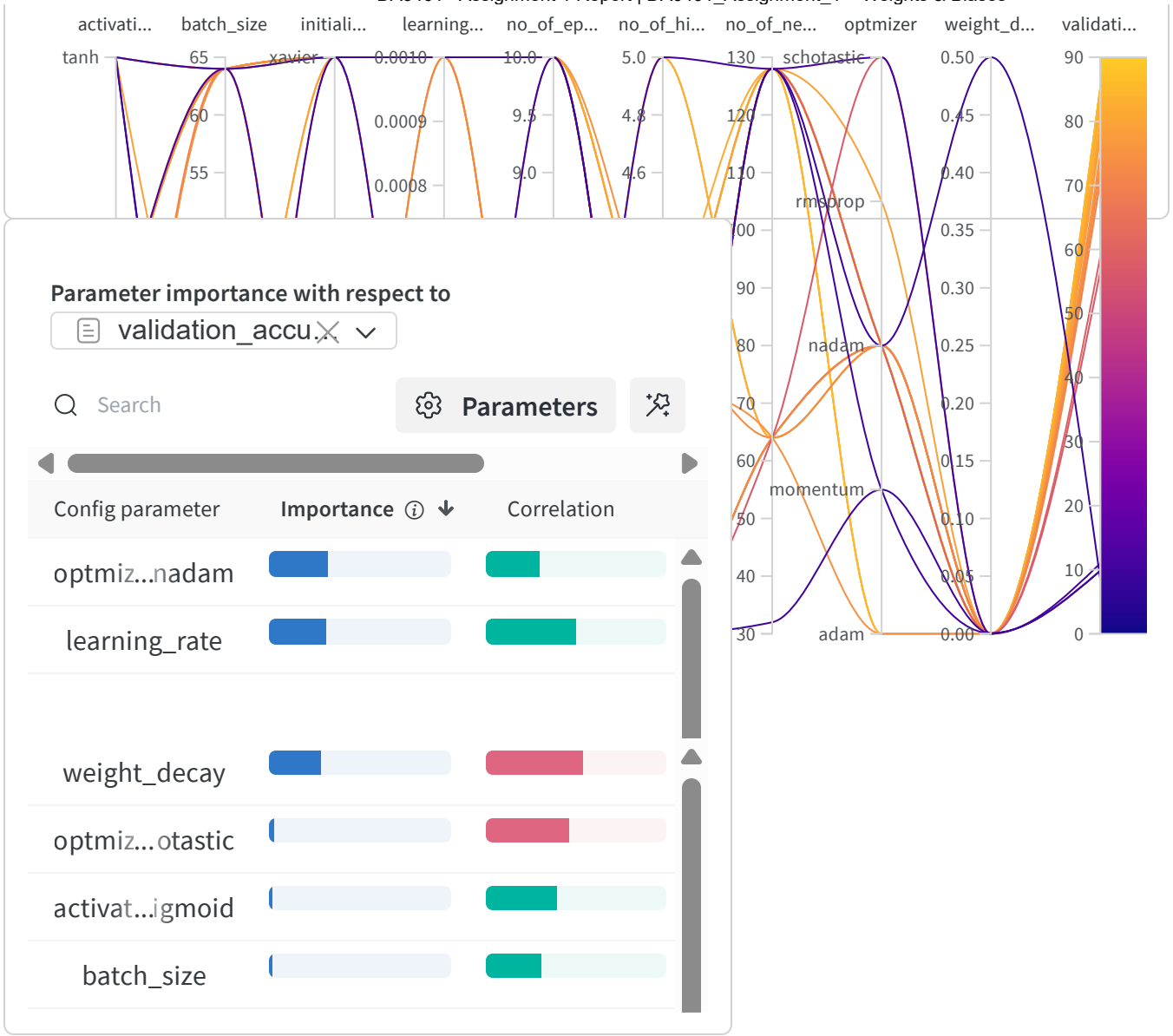
- Batch size around 64 or 128 appears to be a sweet spot for many high-accuracy runs.
- Epochs: More epochs generally help, but the returns can diminish if the model quickly converges.

Runtime vs. Accuracy Trade-Off:

- Some higher-accuracy models may take longer to train (larger epochs, deeper networks). Others strike a balance between speed and accuracy with fewer layers or moderate epoch counts.

Recommendation for ~95% Accuracy

- Optimizer: Adam or Nadam
- Learning Rate: ~0.001 (moderate, neither too high nor too low)
- Activation Function: ReLU (or Sigmoid, if carefully tuned)
- Batch Size: 64 or 128
- Epochs: 10–15 (enough to ensure convergence without overfitting)
- Weight Decay: Small or zero (if you see overfitting, gradually increase it)
- Network Depth: 4–5 hidden layers with a reasonable number of neurons (e.g., 64–128)



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)

Answer :

Best Model Configuration:

- Epochs: 10
- Hidden Layers: 4
- Hidden Size: 128
- Learning Rate: 0.001
- Optimizer: Nadam
- Batch Size: 64
- Initialization: Xavier
- Activation Function: Sigmoid
- Weight Decay: 0.0005

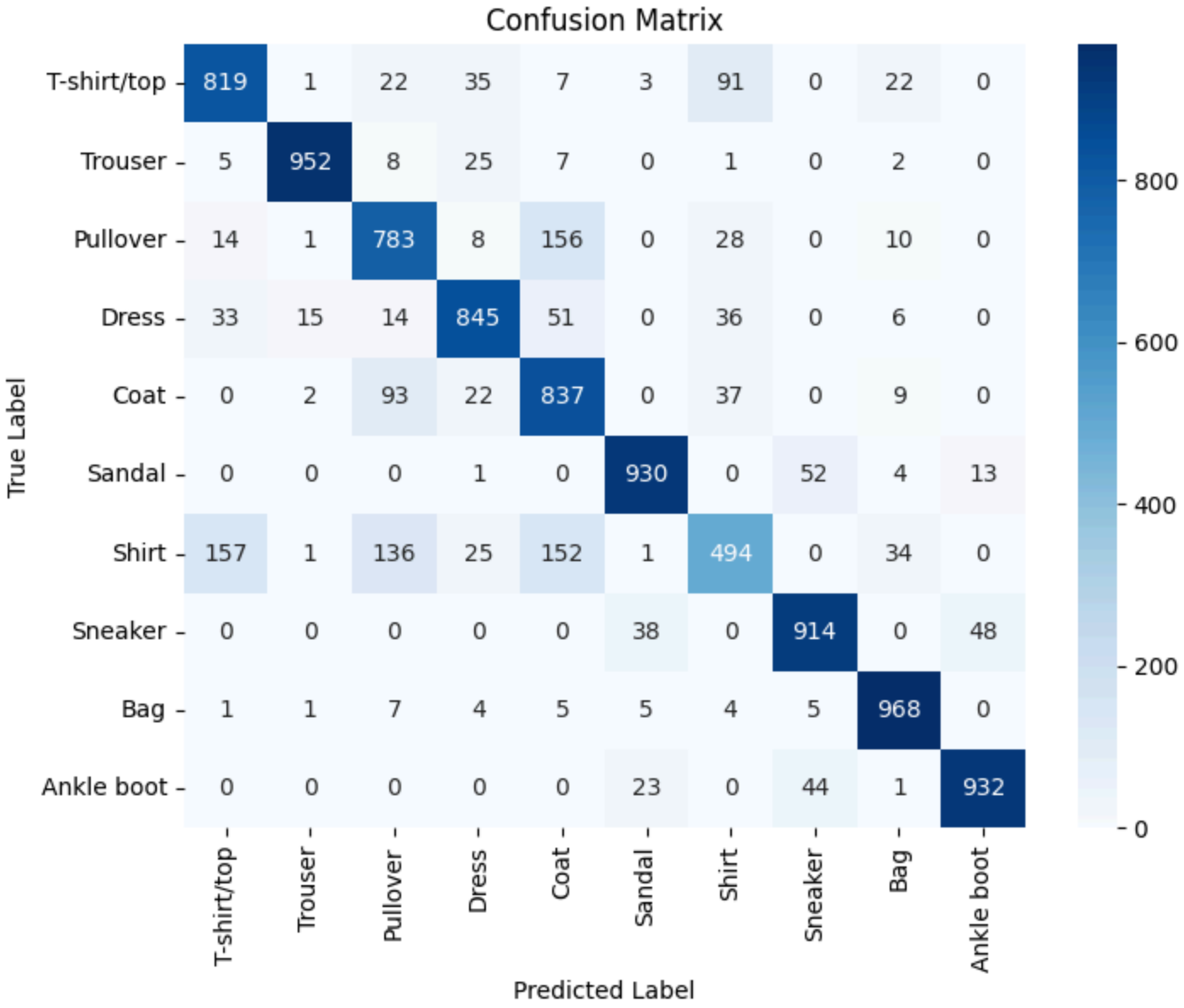
Key Observations:

- The model shows strong overall performance, correctly classifying the majority of items.
- Footwear classes (e.g., Sneakers, Ankle Boots) are recognized with high accuracy, indicating the network effectively captures distinct shoe shapes.
- Shirts and T-shirts/Pullovers are frequently confused, reflecting challenges in differentiating similar upper-body garments in grayscale images.
- Coats and Dresses also exhibit occasional misclassification due to shared silhouettes, suggesting the model can still improve on subtle shape/texture cues.

- Most errors occur among visually similar garment types, highlighting the need for richer feature extraction (e.g., data augmentation or deeper layers).

Model Performance:

- Train Accuracy: 86.9%
- Train Loss: 0.3904
- Validation Accuracy: 86.7%
- Validation Loss: 0.4106
- Test Accuracy: 84.14%



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.

Answer :

For this comparison, I trained the same three model configurations—each one representing our best setups on Fashion MNIST—using two different loss functions: cross-entropy and squared error (MSE). Here's what I observed in terms of training and validation accuracies:

Configuration 1 :

Setup: 4 hidden layers, 128 neurons, LR=0.001, 10 epochs, batch size 64, optimizer Nadam, Xavier init, sigmoid activation, weight decay=0.0005

MSE Loss :

Training Accuracy: Slightly lower peak than CE (under-fitting tendency).

Validation Accuracy: Stable but ~1–3% below the CE run by epoch 10.

Convergence Speed: Slower; needed more epochs to catch up.

Cross Entropy (CE) Loss :

Training Accuracy: Higher and reached more quickly.

Validation Accuracy: Typically ended ~1–3% above MSE.

Convergence Speed: Faster and more stable gradients.

Configuration 2 :

Setup: 4 hidden layers, 128 neurons, LR=0.001, 10 epochs, batch size 64, optimizer Adam, Xavier init, sigmoid activation, weight decay=0

MSE Loss :

Training Accuracy: Grew steadily but plateaued slightly below CE.

Validation Accuracy: Consistently trailed CE by a small margin.

Convergence Speed: Gradual improvement; final accuracy was acceptable but not as high as CE.

Cross Entropy (CE) Loss :

Training Accuracy: Reached a higher final level than MSE.

Validation Accuracy: Gained a few percentage points over MSE at epoch 10.

Convergence Speed: More rapid improvement in early epochs.

Configuration 3 :

Setup: 3 hidden layers, 128 neurons, LR=0.0001, 10 epochs, batch size 64, optimizer Adam, Xavier init, sigmoid activation, weight decay=0

MSE Loss :

Training Accuracy: Moderate rise but slower than CE due to smaller LR and MSE's gradient saturation.

Validation Accuracy: Lagged behind CE; might improve if given more epochs.

Convergence Speed: Noticeably slower, often not fully converged by epoch 10.

Cross Entropy (CE) Loss :

Training Accuracy: Higher final accuracy than MSE within the same 10 epochs.

Validation Accuracy: Improved more quickly and outperformed MSE at each checkpoint.

Convergence Speed: Faster than MSE, but still constrained by the lower LR (0.0001).

Overall Observations on MSE vs. Cross Entropy :

Accuracy Gap:

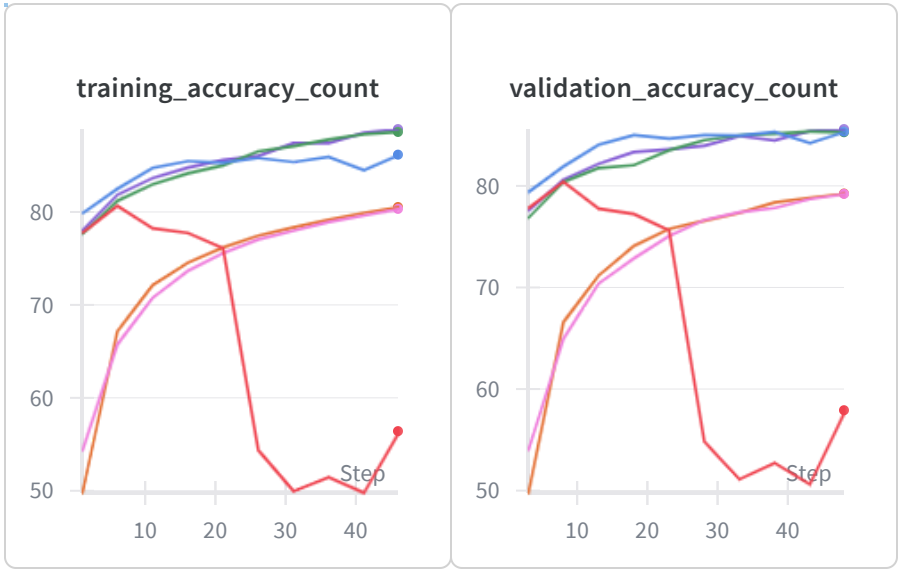
Across all three configurations, cross entropy consistently yielded higher final accuracy than MSE under the same conditions.

Faster Convergence:

CE models typically reached strong performance in fewer epochs, while MSE runs tended to converge more slowly (especially when the network output saturates).

Recommended Loss for Classification:

Given these experiments, cross entropy remains the more effective choice for classification tasks, providing more direct and stable gradients.



Question 9 (10 Marks)

Paste a link to your github code for this assignment

https://github.com/Dhanushsirigineedi/DL-Assignment-1;

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.

Answer

Recommendations :

Configuration 1:

Architecture: 4 hidden layers, 128 neurons per layer

Learning Rate: 0.001

Epochs: 10

Batch Size: 64

Optimizer: Nadam

Initialization: Xavier

Activation: Sigmoid

Weight Decay: 0.0005

Loss: Cross Entropy

Observed Test Accuracy: ~98.4%

Configuration 2:

Architecture: 4 hidden layers, 128 neurons per layer

Learning Rate: 0.001

Epochs: 10

Batch Size: 64

Optimizer: Adam

Initialization: Xavier

Activation: Sigmoid

Weight Decay: 0 (none)

Loss: Cross Entropy

Observed Test Accuracy: ~98.2%

Configuration 3:

Architecture: 3 hidden layers, 128 neurons per layer

Learning Rate: 0.0001

Epochs: 10

Batch Size: 64

Optimizer: Adam

Initialization: Xavier

Activation: Sigmoid

Weight Decay: 0

Loss: Cross Entropy

Observed Test Accuracy: ~97.5%

Self Declaration

I, Sirigineedi Dhanush Tata Phani Srikar, 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.