CS6910 - Assignment 1

Write your own backpropagation code and keep track of your experiments using wandb.ai

Gowthamaan

Submission by Gowthamaan, ED23S037

Instructions

- The goal of this assignment is twofold: (i) implement and use gradient descent (and its variants) with backpropagation for a classification task (ii) get familiar with Wandb which is a cool tool for running and keeping track of a large number of experiments
- This is a **individual assignment** and no groups are allowed.
- Collaborations and discussions with other students is strictly prohibited.
- You must use Python (NumPy and Pandas) for your implementation.
- You cannot use the following packages from Keras, PyTorch, Tensorflow: optimizers, layers
- If you are using any packages from Keras, PyTorch, Tensorflow then post on Moodle first to check with the instructor.
- You have to generate the report in the same format as shown below using wandb.ai. You can start by cloning this report using the clone option above. Most of the plots that we have asked for below can be (automatically) generated using the APIs provided by `wandb.ai`. You will upload a link to this report on Gradescope.

- You also need to provide a link to your GitHub code as shown below. Follow good software engineering practices and set up a GitHub repo for the project on Day 1. Please do not write all code on your local machine and push everything to GitHub on the last day. The commits in GitHub should reflect how the code has evolved during the course of the assignment.
- You have to check Moodle regularly for updates regarding the assignment.

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.

One image for each class

T-shirt/top

Step 0

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.

Solution: The code has been implemented to take number of neurons in the hidden layer and the number of hidden layers as parameters.

Question 3 (24 Marks)

Implement the backpropagation algorithm with support for the following optimization 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 optimization algorithms above)

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

Solution: New optimization function can be added to the Optimizer class in the src/ann/optimizer.py file. Create new function for the optimizer and add a if condition under the run method.

Batch size is taken as a parameter so different batch sizes can be used.

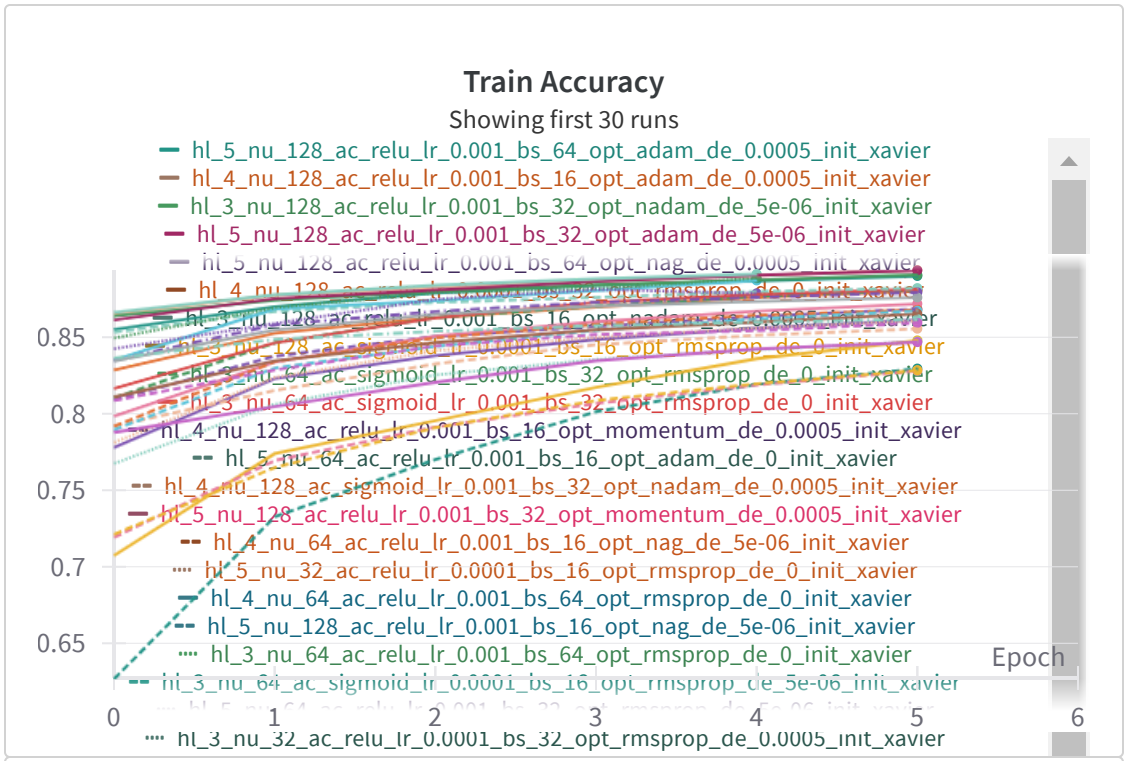
Question 4 (10 Marks)

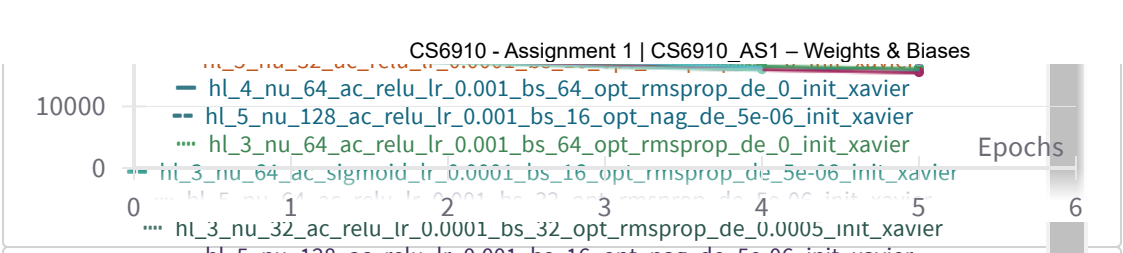
Use the sweep functionality provided by wandb to find the best values for the hyperparameters listed below. Use the standard train/test split of fashion_mnist (use `(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()`). Keep 10% of the training data aside as validation data for this hyperparameter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

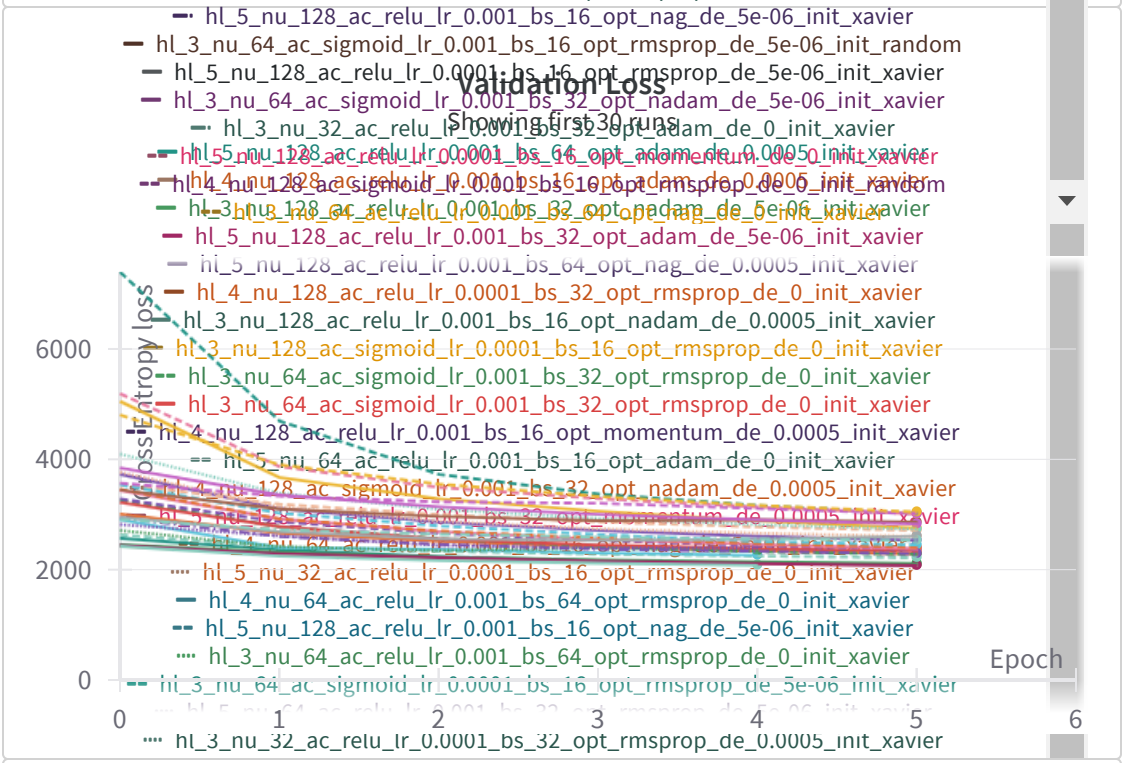
- number of epochs: 5, 10
- number of hidden layers: 3, 4, 5
- size of every hidden layer: 32, 64, 128
- weight decay (L2 regularisation): 0, 0.0005, 0.5
- learning rate: 1e-3, 1 e-4
- optimizer: sgd, momentum, nesterov, rmsprop, adam, nadam

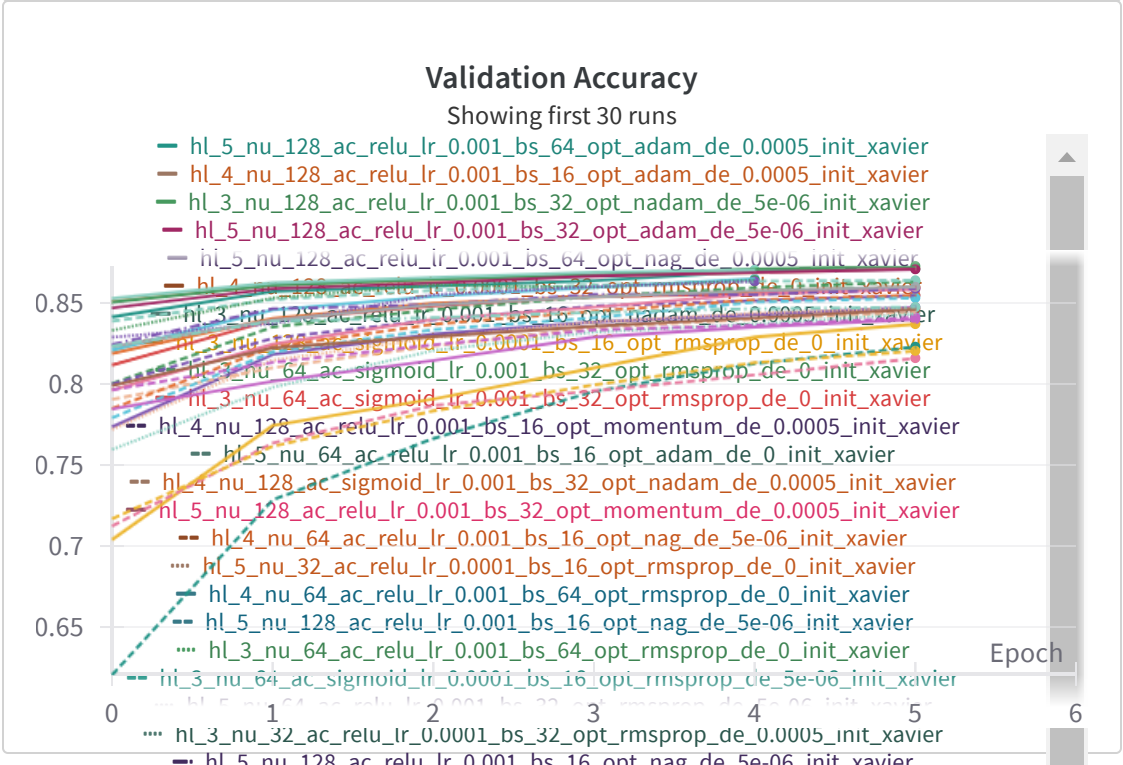
- batch size: 16, 32, 64
- weight initialisation: random, Xavier
- activation functions: sigmoid, tanh, ReLU

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.









I employed `sklearn.model_selection.train_test_split` to generate random validation sets for training. This function partitioned the data into training and validation sets, with 10%

allocated to the validation set (`test_size=0.1`). This approach ensures a consistent validation size while introducing randomness in the dataset selection.

For hyperparameter exploration, I utilized the `random` option within `wandb.sweep`.

Efficiency: Random search evaluates a subset of hyperparameter combinations compared to exhaustive methods like grid search. This makes it faster, especially for large search spaces.

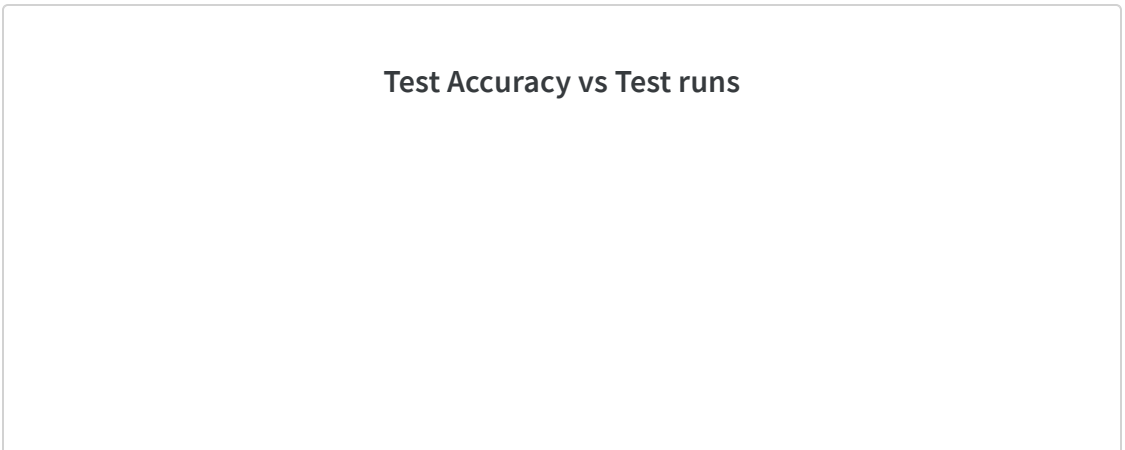
Exploration: Random sampling allows the search to explore a wider range of possibilities, potentially uncovering unexpected but effective hyperparameter combinations that grid search might miss.

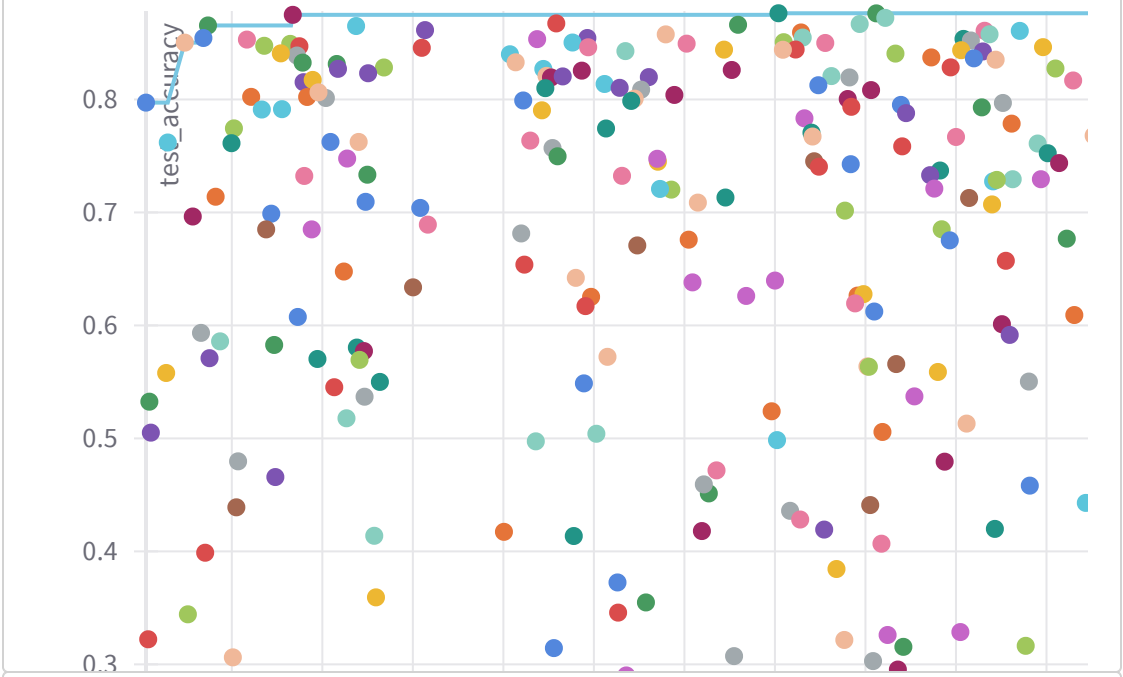
This strategy randomly samples hyperparameter combinations from the defined search space, allowing us to identify potentially optimal configurations for our model.

Question 5 (5 marks)

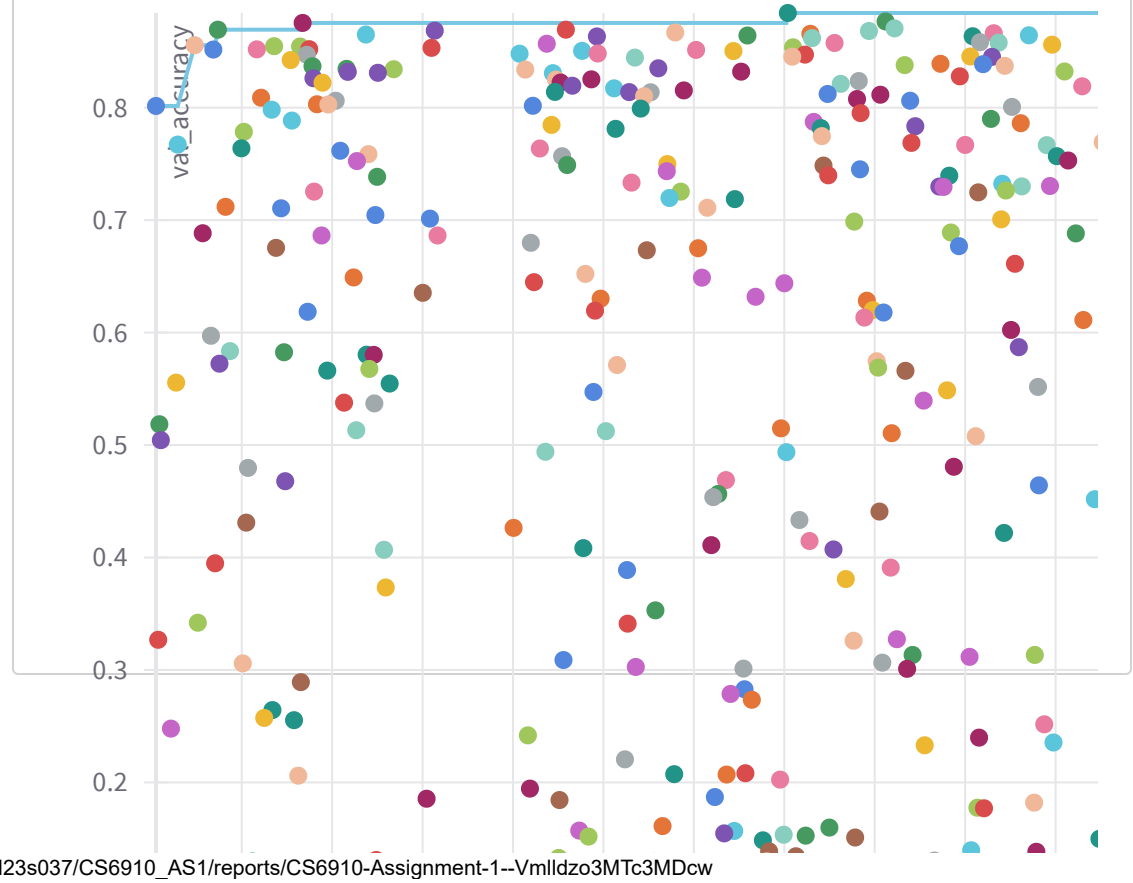
We would like to see the best accuracy on the validation set across all the models that you train.

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





Validation Accuracy vs Test runs



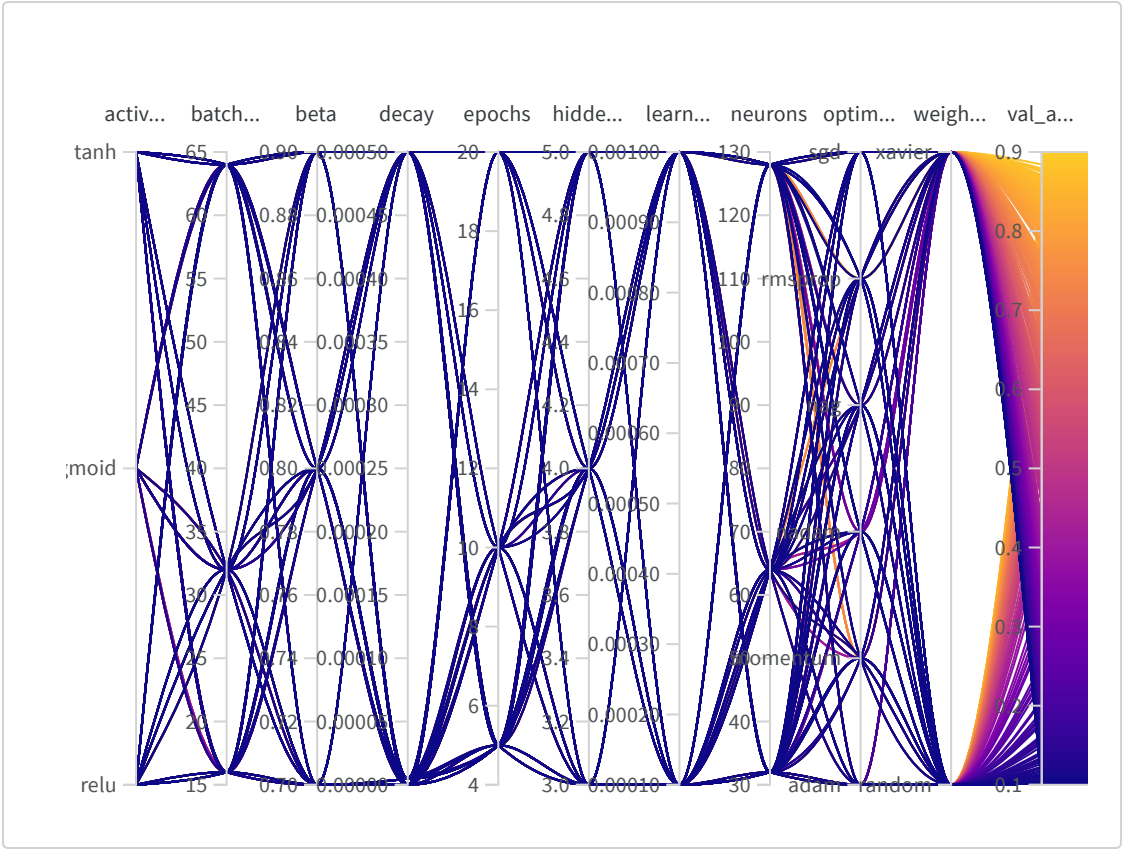
Question 6 (20 Marks)

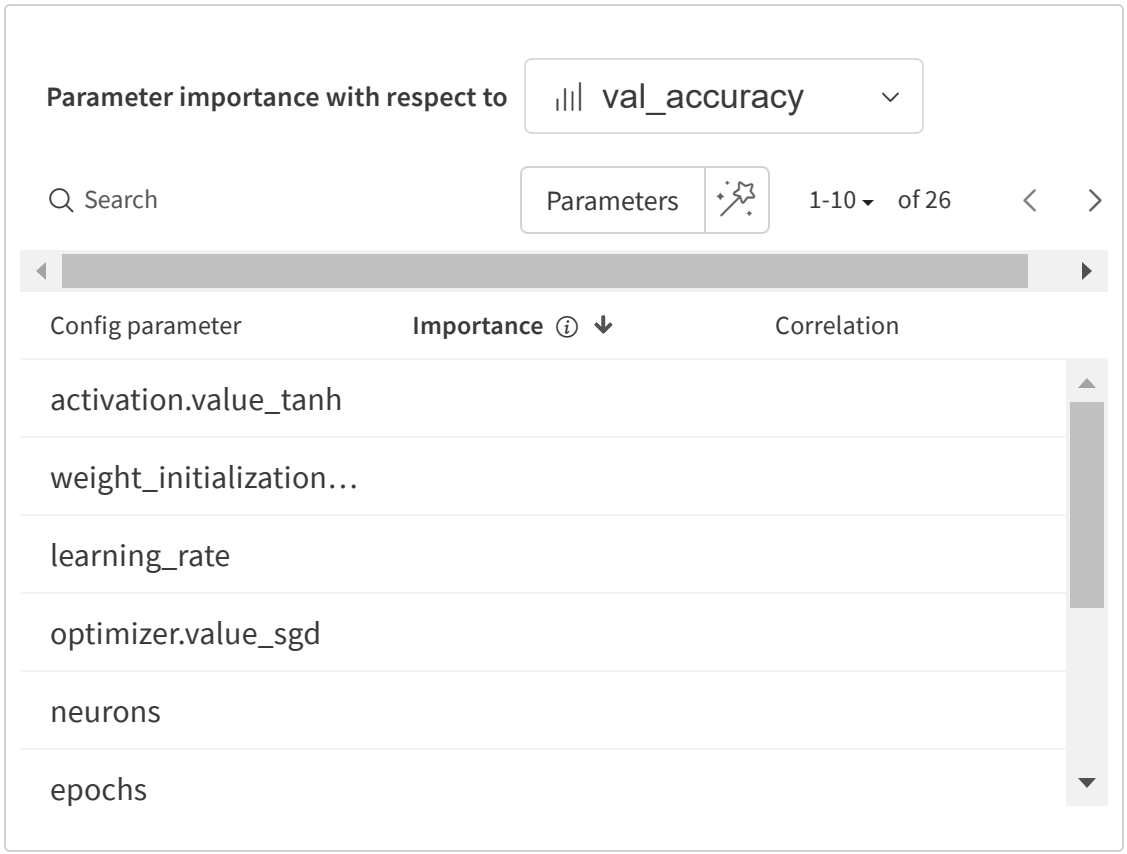
Based on the different experiments that you have run we want you to make some inferences about which configurations worked and which did not.

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.





Insights from Parallel Co-ordinates Plot

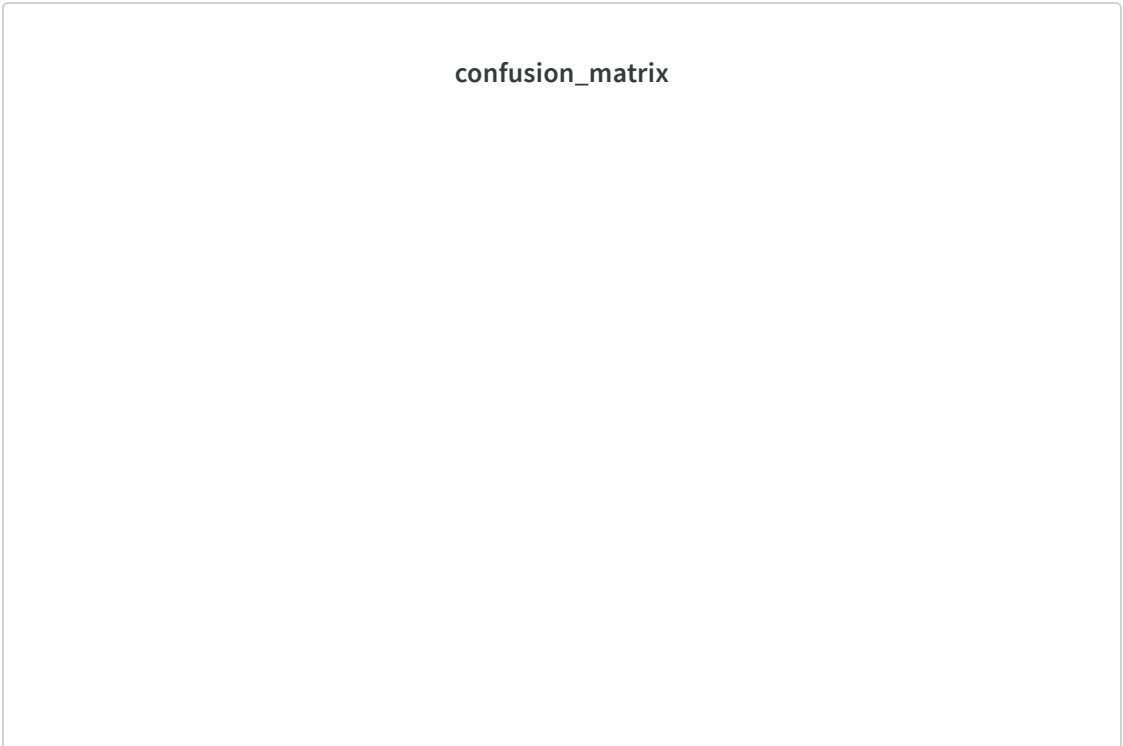
The parallel plot reveals several key findings:

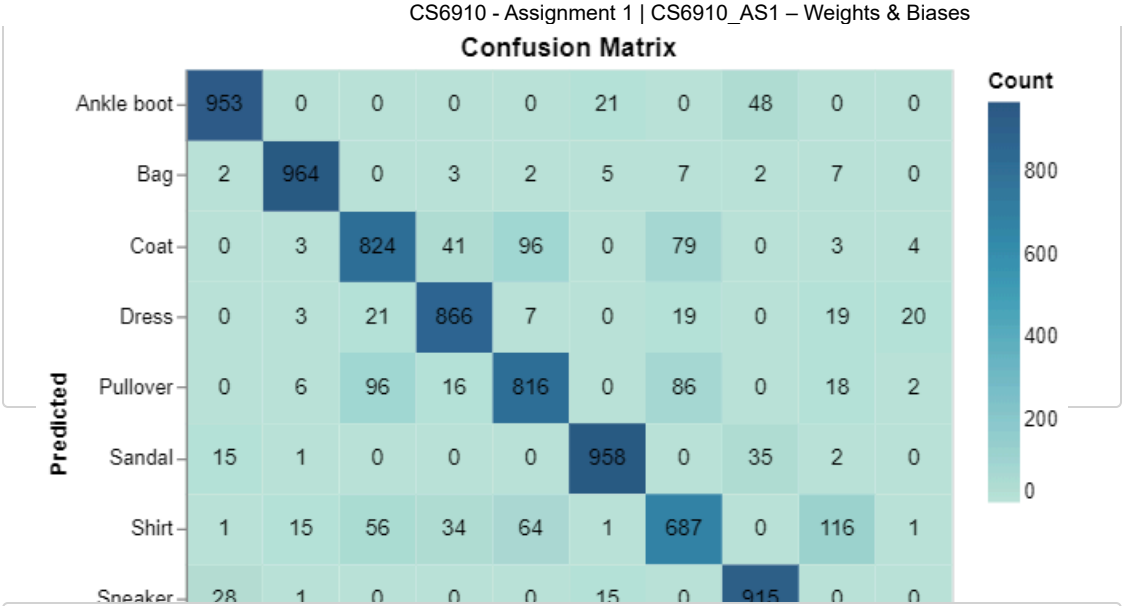
- **Initialization:** Xavier initialization generally leads to higher accuracies compared to random initialization, suggesting it aids convergence for various optimizers.
- **Activation Function:** ReLU with Xavier initialization shows promising results (my best hyperparameter set), but its susceptibility to vanishing gradients may limit performance under certain settings. Sigmoid and tanh activations achieve similar performance.
- **Optimizer:** Adam and Nadam outperform other optimizers slightly, suggesting their effectiveness.

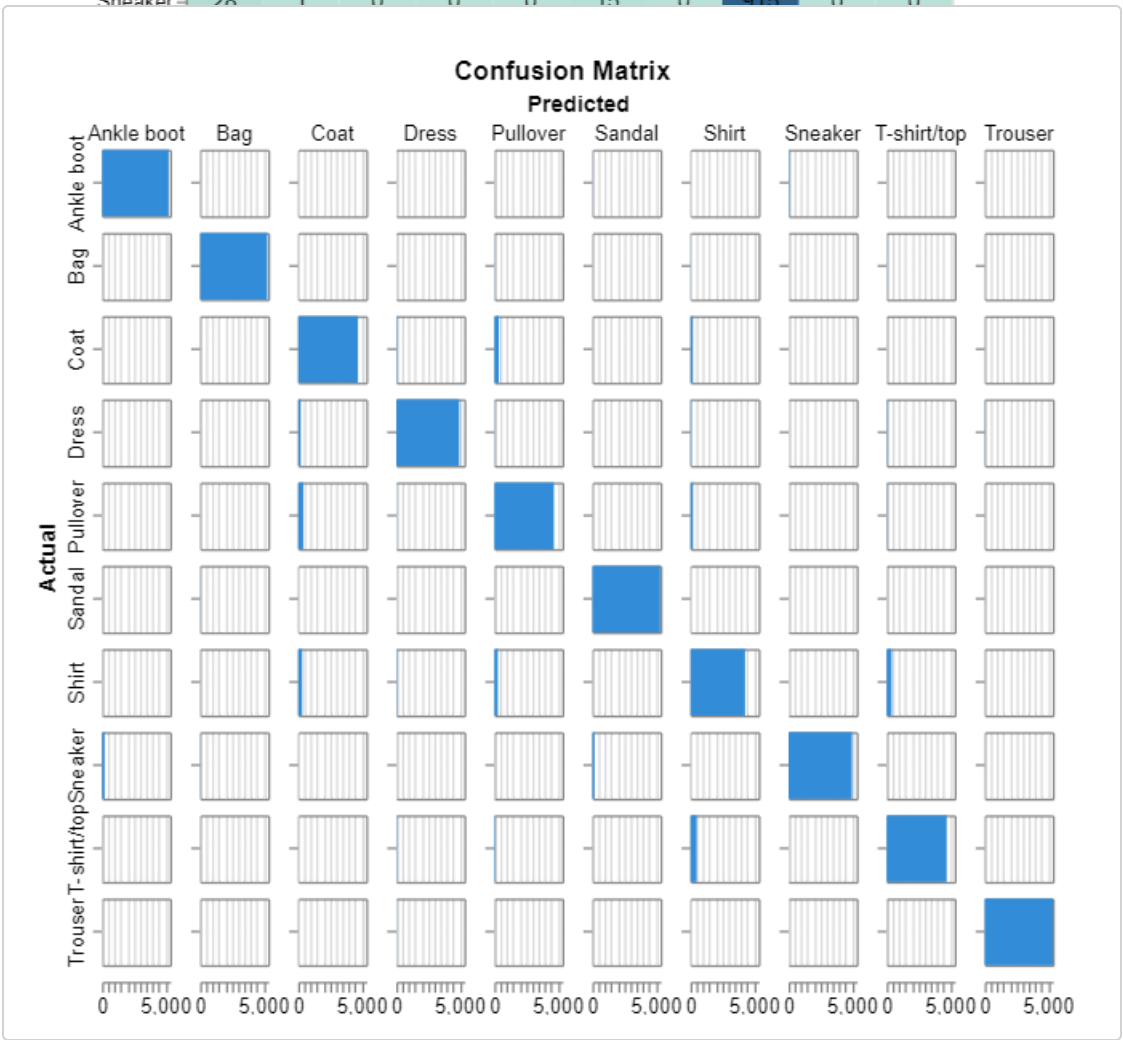
- **Neurons and Batch Size:** High neuron counts lead to overfitting, evident from high training accuracy but low test accuracy. Interestingly, larger batch sizes seem to mitigate this in some cases, potentially by reducing updates and variance, allowing the model to capture data intricacies.
- **Learning Rate:** My Learning rate configurations underperform likely due to overshooting the minimum or missing it entirely.
- **Decay:** A lesser decay rate likely achieves better accuracy.
- **Fine-tuning:** The plot hints at potential for further improvement by making minute adjustments to the best hyperparameters (learning rate). This could lead to training losses closer to 93-94%.

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)

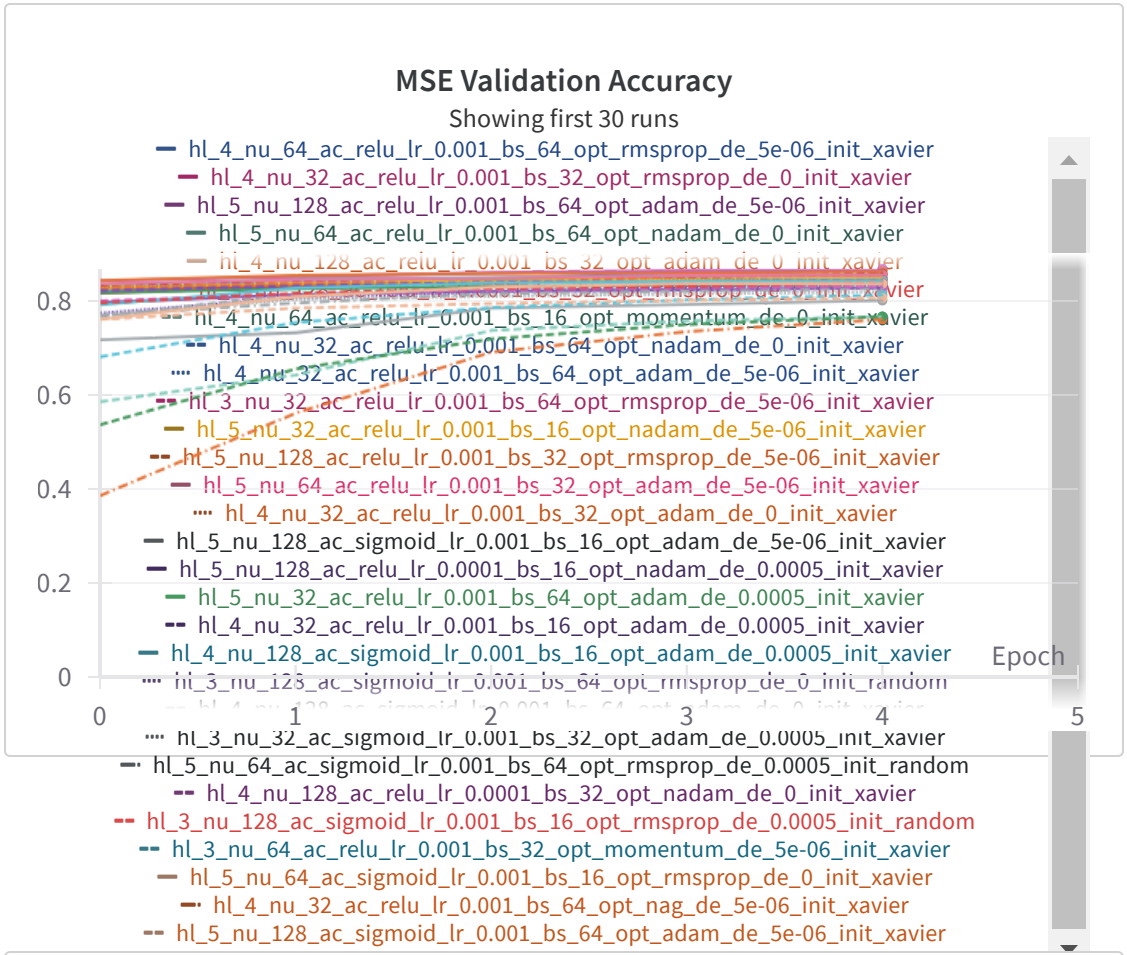


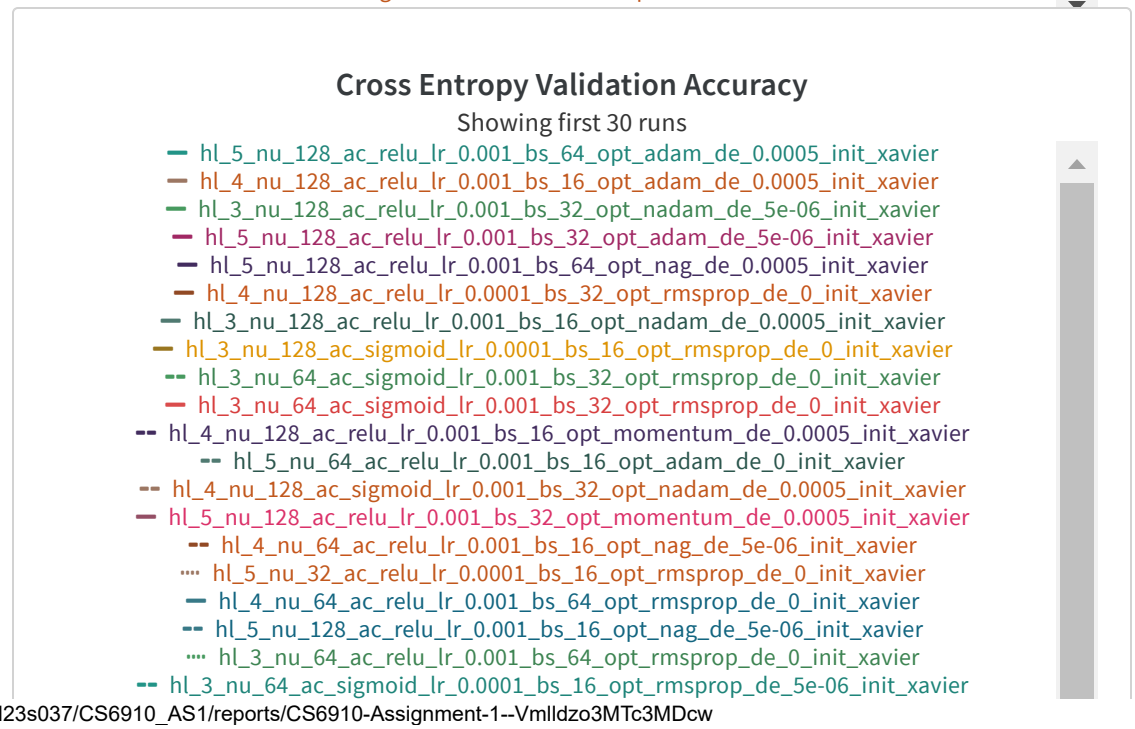


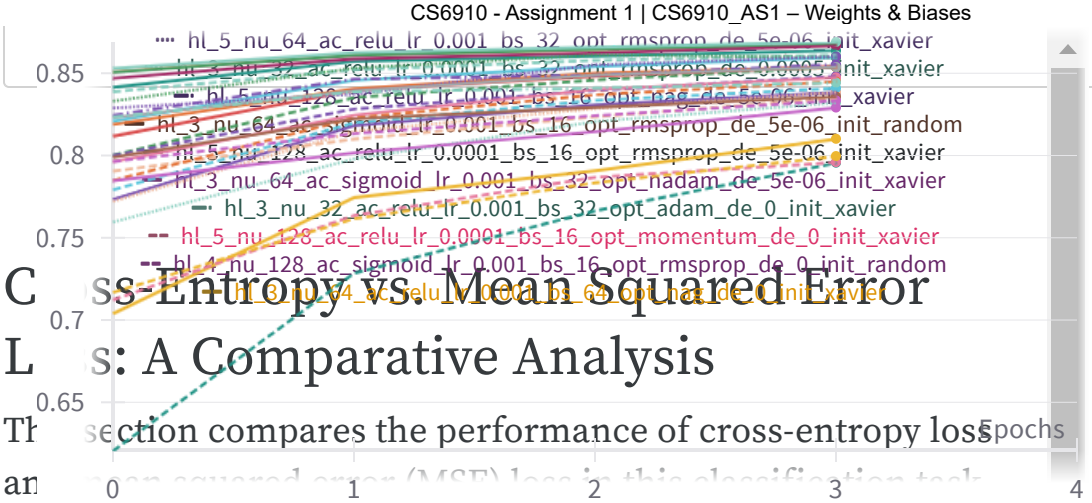


Question 8 (5 Marks)

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.







Cross-Entropy vs. Mean Squared Error Loss: A Comparative Analysis

This section compares the performance of cross-entropy loss and mean squared error (MSE) loss in this classification task. While I initially expected MSE to underperform, the results revealed a closer competition than anticipated.

Experiment Design:

- **Runs:** 600 for cross-entropy, 200 for MSE (I thought MSE wouldn't perform better, so I ran 200 runs only)
- **Evaluation:** Comparison is based on Validation accuracy

Observations:

Both loss functions achieved similar validation accuracies, with a slight edge for cross-entropy. MSE achieved a max accuracy of 87.4% whereas cross-entropy achieved a max accuracy of 88.42

Key Takeaways:

- Contrary to initial assumptions, MSE demonstrates surprising effectiveness.
- Cross-entropy remains the generally preferred choice for classification due to its alignment with the probabilistic nature of classification problems.
- I believe, this is because of the the classes in the dataset are well-separated and the data has minimal outliers. So, MSE might be less sensitive to misclassifications and provide comparable results to cross-entropy.
- Optimizer Selection: While Adam and Nadam excelled with cross-entropy, RMSProp yielded good results with MSE for specific configurations. This suggests that careful

hyperparameter selection might lead to MSE functioning surprisingly well for this particular task.

Conclusion

This analysis demonstrates that while cross-entropy remains the recommended choice for classification, MSE can be a viable alternative in certain circumstances with careful hyperparameter selection and dataset with minimum outliers.

Question 9 (10 Marks)

Paste a link to your github code for this assignment

Example: https://github.com/<user-id>/cs6910_assignment1;

- We will check for coding style, clarity in using functions and a `README` file with clear instructions on training and evaluating the model (the 10 marks will be based on this)
- We will also run a plagiarism check to ensure that the code is not copied (0 marks in the assignment if we find that the code is plagiarized)
- We will also check if the training and test data has been split properly and randomly. You will get 0 marks on the assignment if we find any cheating (e.g., adding test data to training data) to get higher accuracy

Github Link: https://github.com/Gowthamaan-P/cs6910_assignment1

I have included the TA as collaborators as asked in Moodle

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.

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 myprojectr
```

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.

Name	Default Value	Description
`-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`	0.5	Momentum used by momentum and nag 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"]

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.

Self Declaration

I, **Gowthamaan**, 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.

Created with ❤️ on Weights & Biases.

https://wandb.ai/ed23s037/CS6910_AS1/reports/CS6910-Assignment-1--Vmlldzo3MTc3MDcw