DA6401 - Assignment 1

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

Deadline: 10th March 2025, Hard deadline. No extensions will be provided.

S Ashwin

Created on March 10 | Last edited on March 18

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.

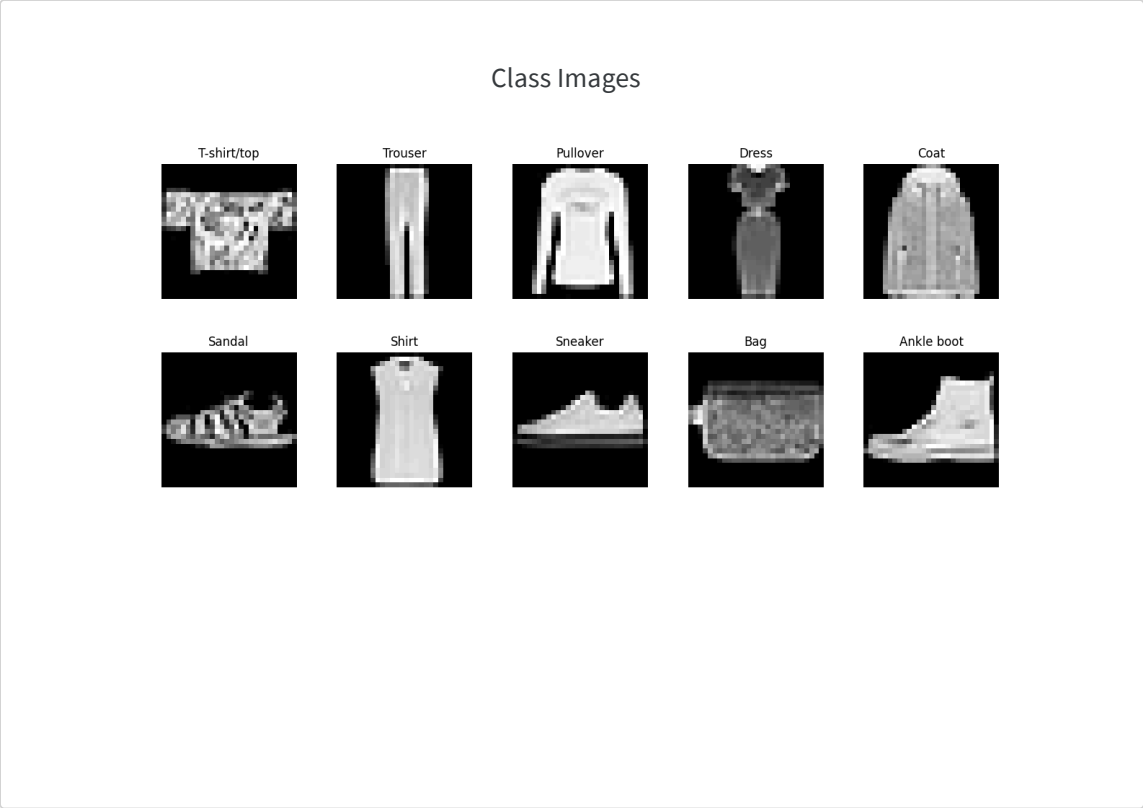
Problem Statement

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.



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 uploaded to GH

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 uploaded to GH

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.

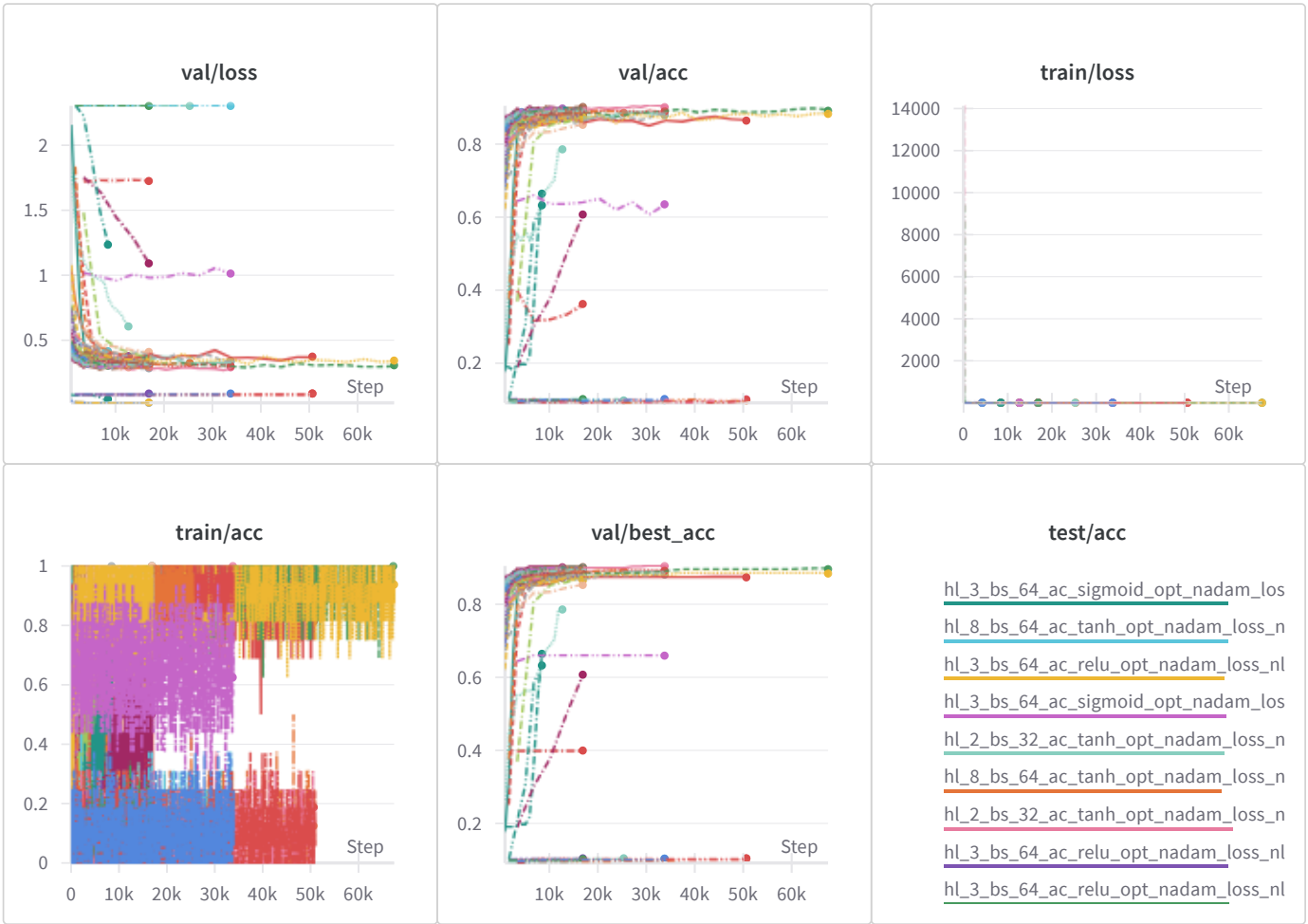
- number of epochs: 5, 10
- number of hidden layers: 3, 4, 5
- size of every hidden layer: 32, 64, 128
- weight decay (L2 regularisation): 0, 0.0005, 0.5
- learning rate: 1e-3, 1 e-4
- optimizer: sgd, momentum, nesterov, rmsprop, adam, nadam
- batch size: 16, 32, 64

- weight initialisation: He, 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.

Sweep Strategy used:

Bayesian optimization is superior to random and grid search because it intelligently models the objective function using a Gaussian Processes to guide the search. Unlike grid search, which exhaustively evaluates all parameter combinations and is computationally expensive, Bayesian optimization efficiently explores promising regions. Compared to random search, which samples parameters blindly, Bayesian optimization updates its belief about the function and selects the next set of hyperparameters to evaluate in a way that balances exploration and exploitation. This makes it significantly more sample-efficient, leading to better hyperparameters with fewer evaluations, especially for expensive models.



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

Best Model Summary

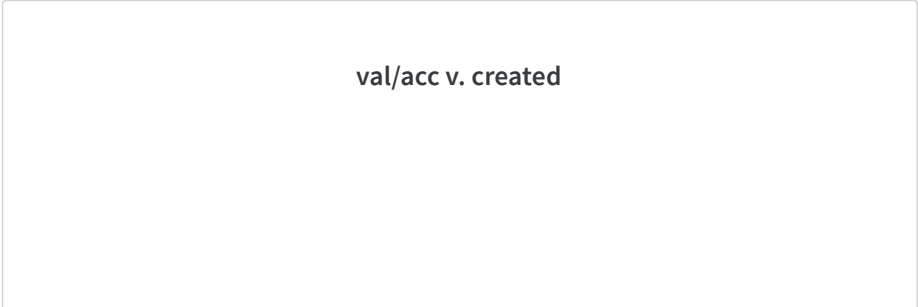
The best validation accuracy achieved across all trained models was **90.21%**.

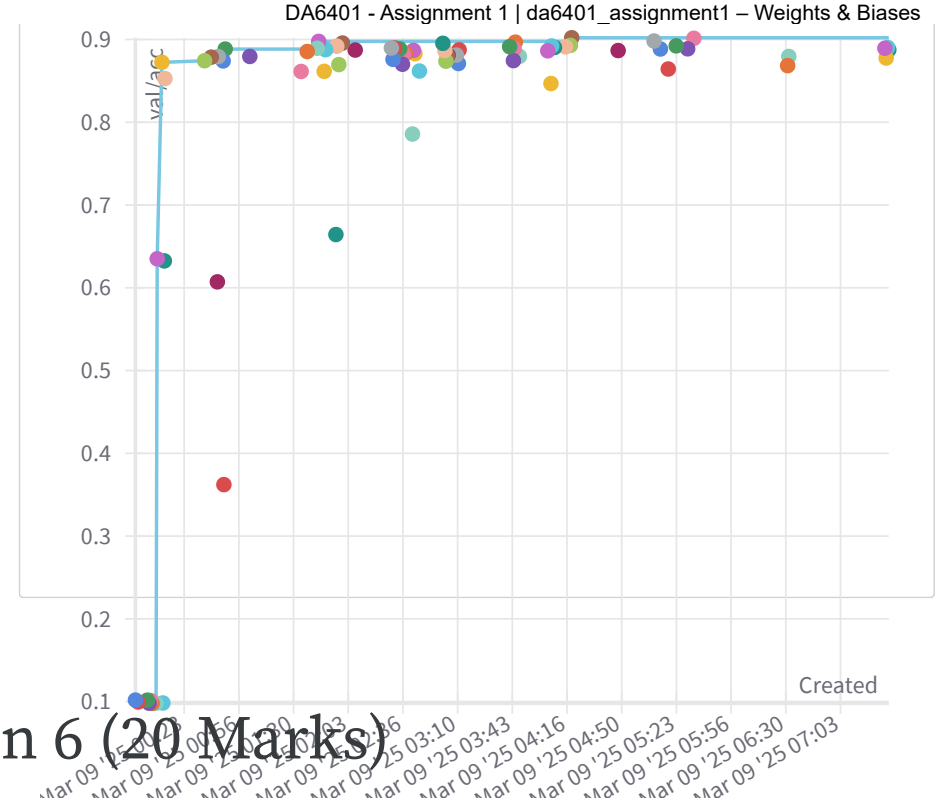
Hyperparameters of the Best Model:

- **Batch Size:** 64
- **Optimizer:** Nadam
- **Learning Rate:** 1.75e-4
- **Beta1:** 0.95
- **Beta2:** 0.999
- **Epsilon:** 1e-7
- **Weight Decay:** 5.3e-4
- **Loss Function:** NLL
- **Initialization Strategy:** Xavier
- **Hidden Activation:** Tanh
- **Hidden Layers:**
  - **Layer 1:** 2048 units
  - **Layer 2:** 1024 units
- **Number of Epochs:** 20

This configuration resulted in a validation accuracy of 90.21% with a validation loss of 0.305.

Below is the attached WandB configuration and summary metrics from the best run.





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

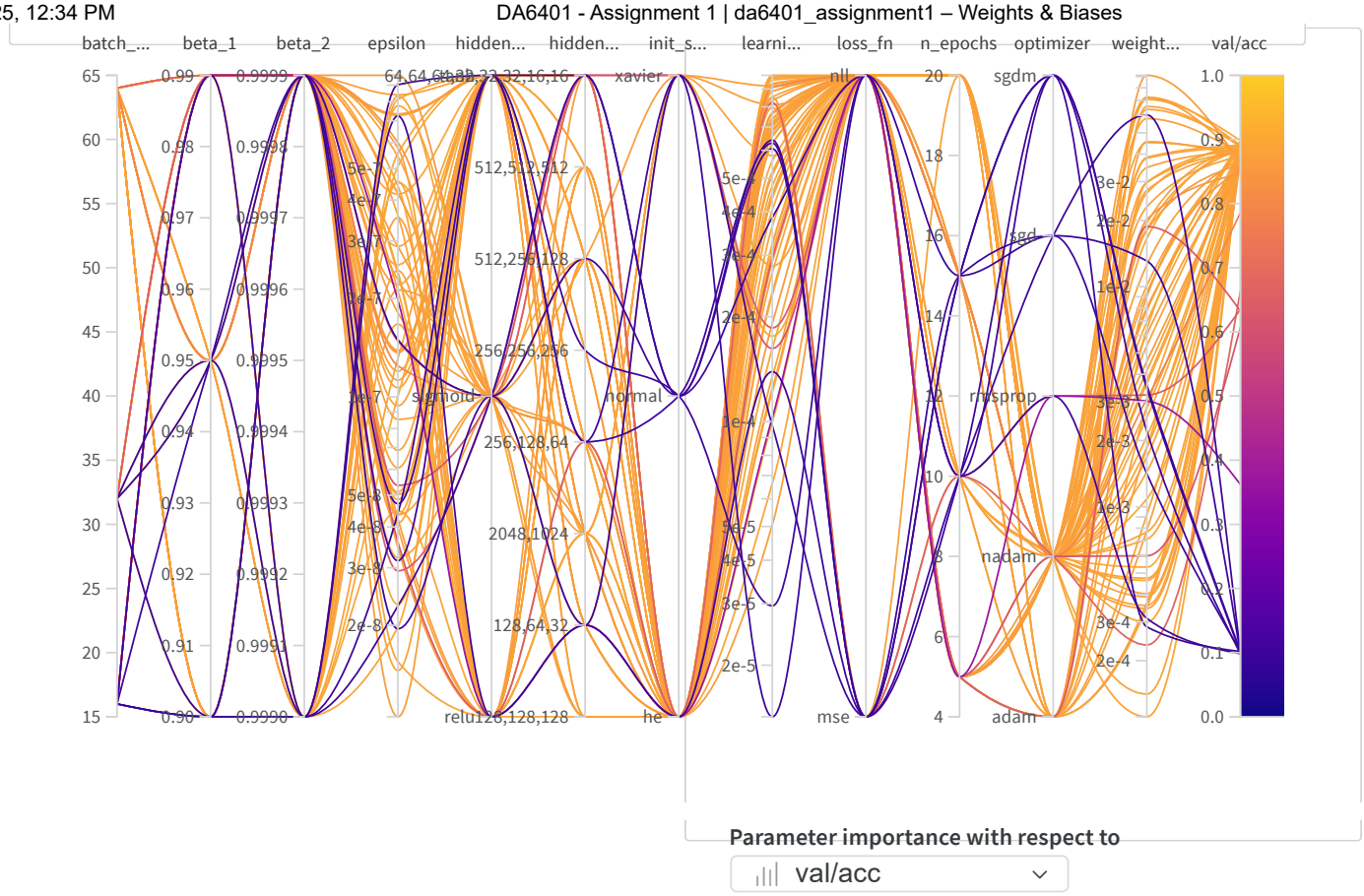
- **Batch Size: Higher batch sizes correlate with better validation accuracy**, suggesting that larger batch sizes lead to more stable training and improved generalization.
- **Learning Rate:** One of the most critical parameter affecting validation accuracy. Optimal values lie between **1e-3 and 5e-5**. Extremely high or low values lead to poor performance.
- **Optimizer Choice:** Models using **NAdam and Adam** generally achieve higher validation accuracy, while **SGD, SGDM, NAG, and RMSprop** are associated with lower accuracy (< 60%).
- **Loss Function:** Models achieving **>80% validation accuracy** predominantly use **NLL Loss**, while MSE Loss is more common among lower-performing runs.

- **Network Architecture: Shallow and wide networks** perform better than **deep and narrow networks**, suggesting that increasing width benefits performance more than adding depth.
- **Collapsing Networks:** Networks that reduce dimensions at later layers tend to achieve higher validation accuracy than those maintaining the same hidden size across layers.
- **Initialization Strategy: He initialization** yields the best results, with **Xavier** also performing well. **Normal initialization consistently underperforms.**
- **Beta Values:** Models with **Beta1 = 0.95 and Beta2 = 0.999** show improved stability and higher accuracy.
- **Activation Function:** No single activation function consistently outperforms others, indicating that activation choice has a limited impact on validation accuracy.

Recommendation for 95% Accuracy:

Based on the trends in the parallel coordinates plot and parameter importance:

- Use **NAdam optimizer**.
- Set the **learning rate** in the range **1e-3 to 5e-5**.
- Opt for **NLL Loss** over MSE Loss.
- Prefer **shallow, wide networks** with **collapsing dimensions**.
- Use **He initialization** for weight initialization.
- Set **Beta1 = 0.95, Beta2 = 0.999** for optimal performance.



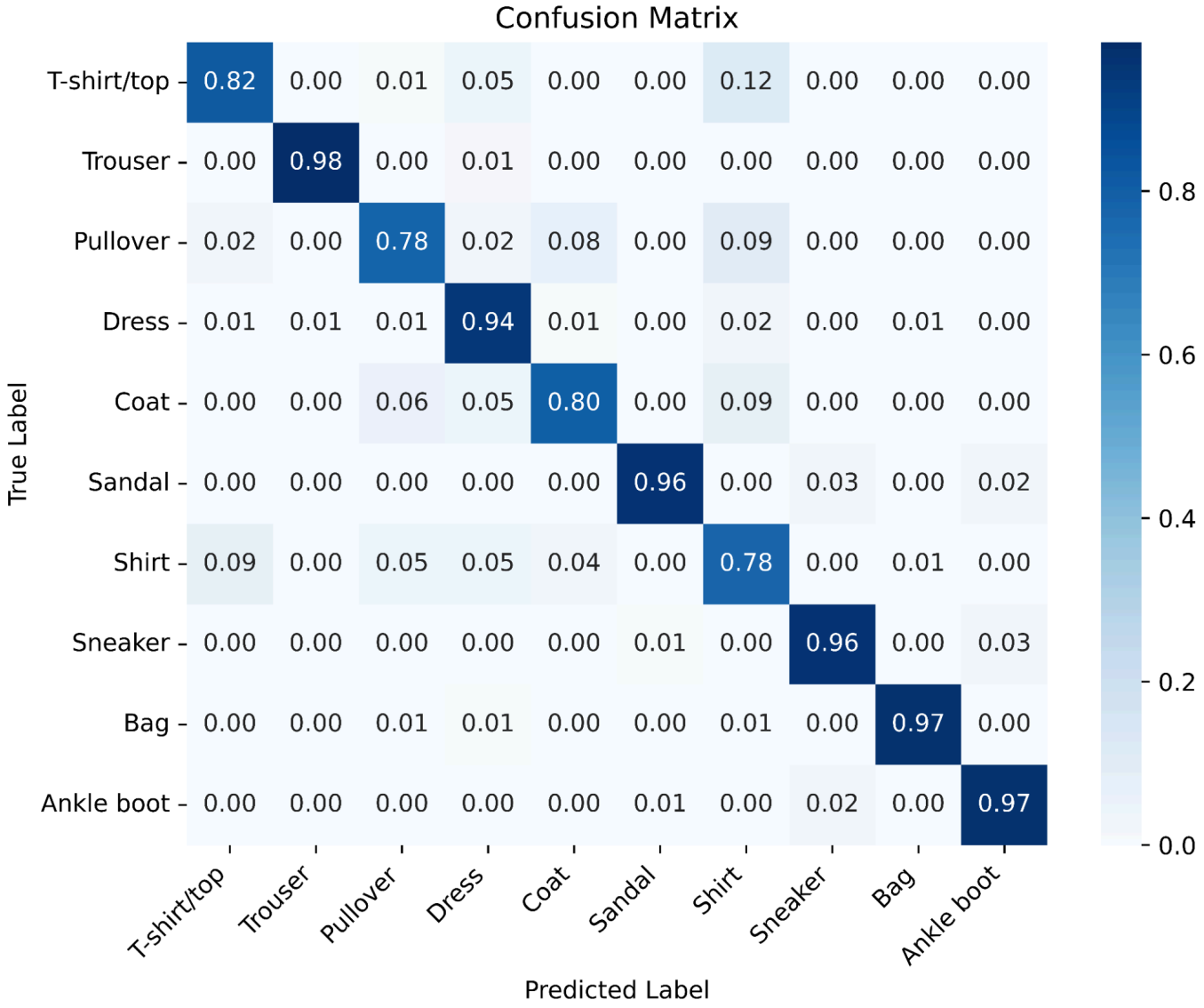
Question 7 (10 Marks)

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)

Here's a concise summary for your **WandB report**:

Key Observations

- **Strong diagonal performance**, indicating overall good classification.
- **High accuracy for distinct categories** like **Trousers (98%), Sandals (96%), Sneakers (96%), and Bags (97%).**
- **Misclassifications** occurred primarily between visually similar items:
  - **T-shirt ↔ Shirt (12% confusion)**
  - **Pullover ↔ Shirt (9% confusion)**
  - **Pullover ↔ Coat (8% confusion)**
  - **Coat ↔ Shirt (9% confusion)**



Question 8 (5 Marks)

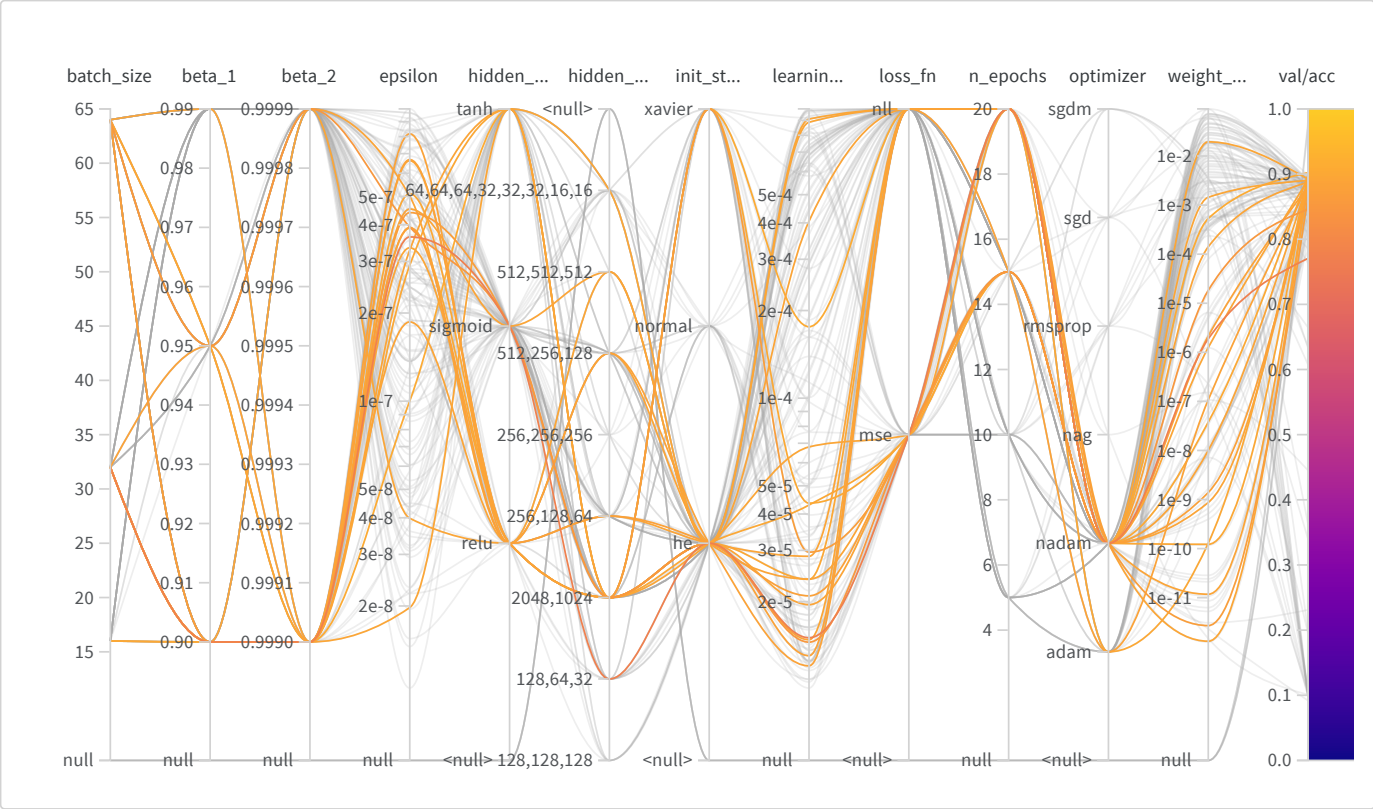
In all the models above you would have used cross entropy loss. Now compare the cross entropy loss with the squared error loss. I would again like to see some automatically generated plots or your own plots to convince me whether one is better than the other.

Solution:

- **Both MSE and NLL perform competitively**, but MSE has more runs with lower validation accuracy.
- **NLL (cross-entropy) is statistically justified** for classification as it maximizes the likelihood of correct predictions.
- **Best-performing models used NLL**, suggesting it is more stable across architectures.
- **MSE applies equal penalty to all errors**, which can lead to suboptimal learning in classification tasks.

- **NLL penalizes confident wrong predictions more heavily**, making it a better choice for classification.
- **MSE is better for regression**, while **NLL is the preferred choice for classification** due to its theoretical grounding in probability.

As you can see in the selected examples below. For the given task the network trained with mse is actually producing better test accuracy.



Question 9 (10 Marks)

Paste a link to your github code for this assignment

Example: https://github.com/AshwinSankar17/da6401_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

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.

Key Learnings from Fashion-MNIST & Their Application to MNIST

1. **Higher batch size improves stability and convergence** → We will use **batch size 128** for better gradient estimation.
2. **Nadam optimizer with a lower learning rate (≈1e-4) works well** → Retain Nadam but slightly tune learning rate.
3. **Larger models (wider & deeper) help Fashion-MNIST, but MNIST needs less complexity** → Reduce network width.
4. **Weight decay helps generalization, but MNIST is simpler than Fashion-MNIST** → Slightly lower weight decay.
5. **Cross-entropy loss (NLL) works better than MSE, especially for confident predictions** → Keep NLL.

---

3 Recommended Hyperparameter Configurations for MNIST

Config	Hidden Sizes	Activation	Optimizer	Learning Rate	Batch Size	Weight Decay	Accuracy
1	[512, 256, 128]	ReLU	Nadam	1e-3	128	1e-4	**~97.36%**
2	[1024, 512, 256]	Tanh	Adam	5e-4	128	5e-4	**~97.86%**
3	[512, 256]	ReLU	Nadam	5e-4	64	1e-5	**~97.60%**

Why These Configurations?

1. Config 1: Balanced depth and width with ReLU for fast convergence. Nadam with 1e-3 learning rate is aggressive but should generalize well with weight decay.
2. Config 2: A deeper model with Tanh instead of ReLU, reducing dying neuron issues. Adam stabilizes training, weight decay prevents overfitting.
3. Config 3: A smaller model for efficiency, trading some accuracy for speed. Lower weight decay to avoid excessive regularization.

One observation is that these models are also very quick to overfit. So an early stopping callback or reducing the model complexity may help in reaching a better accuracy.

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`	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, S Ashwin, 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/iamunr4v31/da6401_assignment1/reports/DA6401-Assignment-1--VmlldzoxMTcyMDU1MA