CS6910 - Assignment 1

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

Arun Muthukkumaran

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.
Neural network was created which takes in choice for the number of hidden layers, neurons per layer, etc. Codes have been uploaded to Github

Question 3 (18 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.
Neural network was constructed with all the above mentioned hyperparameter. Hyperparameter tuning and visualization were performed using wandb.ai. Codes have been uploaded to Github. Visualizations have been depicted below.

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.
Hyperparameter tuning was performed using wandb.ai. Wandb provides 3 main strategies for hyperparameter tuning viz. Grid search, Random search and Bayesian Optimization.

Grid search: Grid search involves defining the feasible set of hyperparameters and the the model is tested across all the hyperparameters. Though this method is comprehensive, it is time consuming and not feasible all the time.

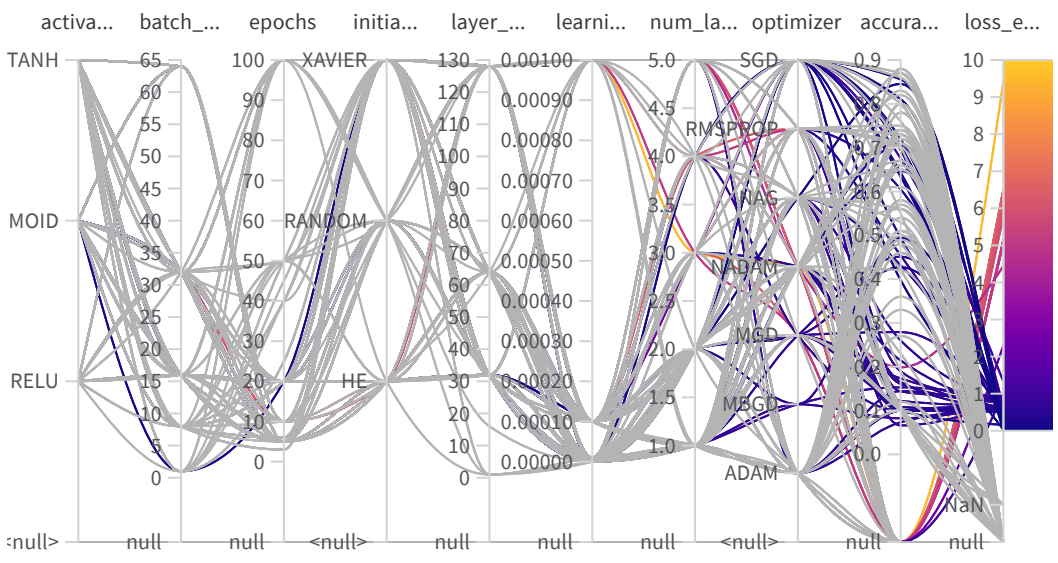
Random search: In the random search method, a set of hyperparameters are chosen at random and the neural network is trained using the chosen hyperparameters. The number of sets of hyperparameters to be chosen are provided by the user.

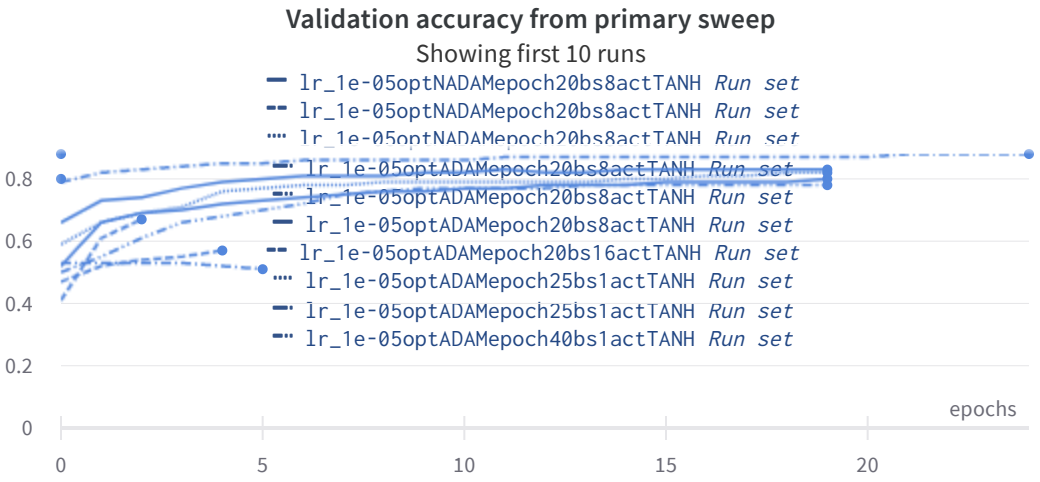
Bayesian optimization: In Bayesian optimization, the optimizer learns from the previous set of hyperparameter trials and finds the best possible hyperparameters in the subsequent trials.

For this problem I have chosen a combination of Grid search and Random search. This choice was made so as to arrive at the optimal hyperparameters at the shortest time possible. The Random search strategy was employed first to understand the model performance with respect to all the available hyperparameters. The results of various random hyperparameter sweeps have been shown in the *Primary sweep* parallel plot. From the primary sweep the initial set of hyperparameters were fixed.

Initialization was fixed to XAVIER algorithm, optimization algorithm chosen was ADAM, Activation function chosen was TANH, batch size was fixed at 32 and number of epochs was fixed at 100. The results of this analysis are shown in the secondary sweeps plot. From the secondary sweeps plot a best configuration was chosen to train the neural network and obtain the best accuracy.

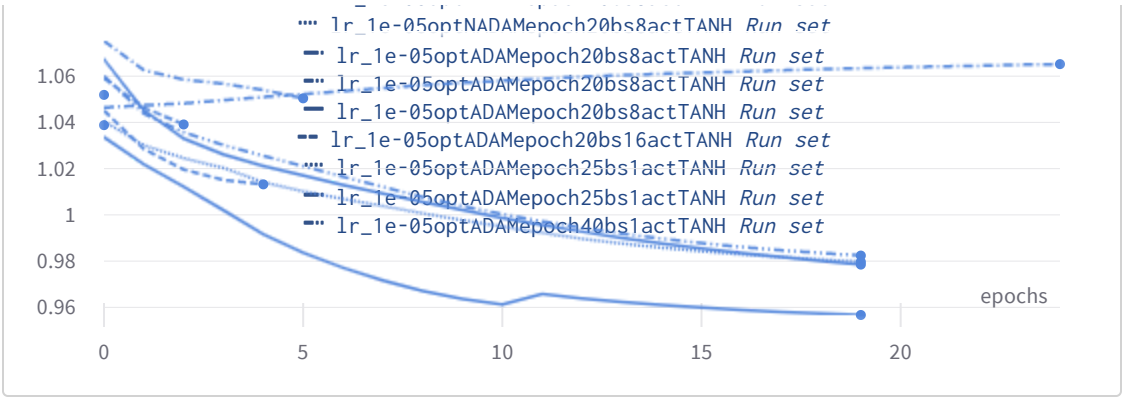
PRIMARY SWEEPS TO NARROW DOWN BEST HYPERPARAMETERS


activa...
batch_...
epochs
initia...
layer_...
learni...
num_la...
optimizer
accura...
loss_e...
TANH
MOID
RELU
<null>
XAVIER
RANDOM
HE
SGD
RMSPROP
NAG
NADAM
MGD
MBGD
ADAM
NaN

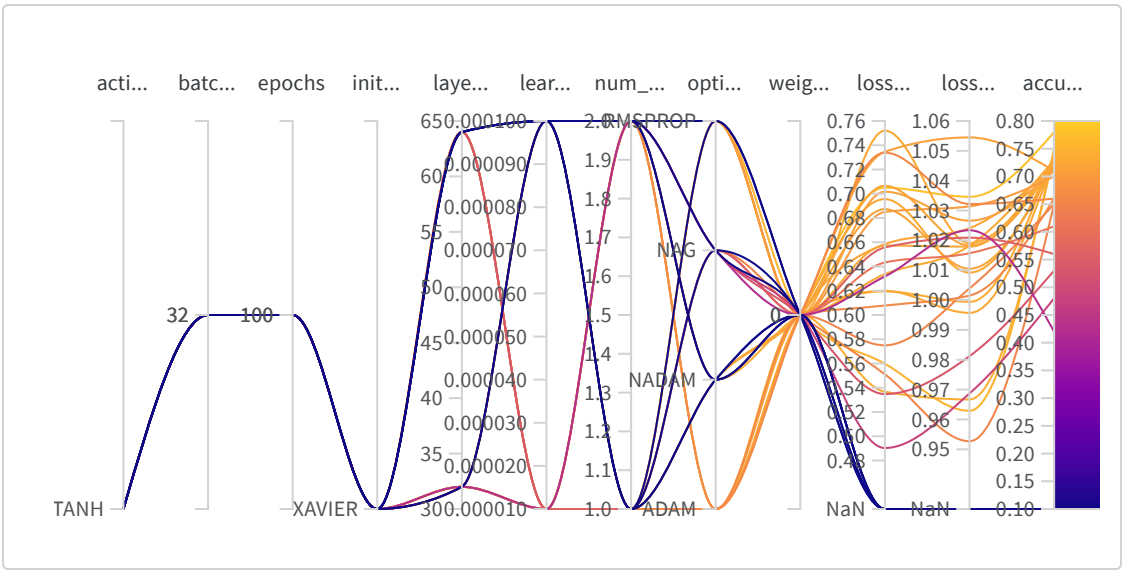

Validation accuracy from primary sweep
Showing first 10 runs
lr_1e-05optNADAMepoch20bs8actTANH Run set
lr_1e-05optNADAMepoch20bs8actTANH Run set
lr_1e-05optADAMepoch20bs8actTANH Run set
lr_1e-05optADAMepoch20bs8actTANH Run set
lr_1e-05optADAMepoch20bs16actTANH Run set
lr_1e-05optADAMepoch25bs1actTANH Run set
lr_1e-05optADAMepoch25bs1actTANH Run set
lr_1e-05optADAMepoch40bs1actTANH Run set
epochs


loss_train
Showing first 10 runs
lr_1e-05optNADAMepoch20bs8actTANH Run set
lr_1e-05optNADAMepoch20bs8actTANH Run set
lr_1e-05optADAMepoch20bs8actTANH Run set
lr_1e-05optADAMepoch20bs8actTANH Run set
lr_1e-05optADAMepoch20bs16actTANH Run set
lr_1e-05optADAMepoch25bs1actTANH Run set
lr_1e-05optADAMepoch25bs1actTANH Run set
lr_1e-05optADAMepoch40bs1actTANH Run set
epochs


loss_val
Showing first 10 runs
lr_1e-05optNADAMepoch20bs8actTANH Run set



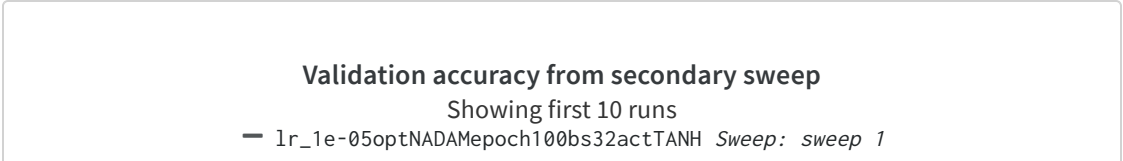
SECONDARY SWEEPS FOR CHOOSING BEST HYPERPARAMETERS

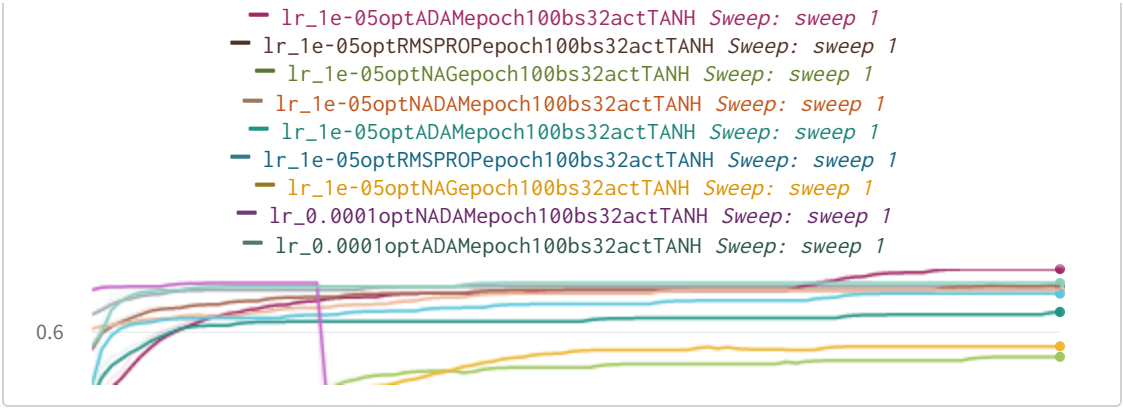


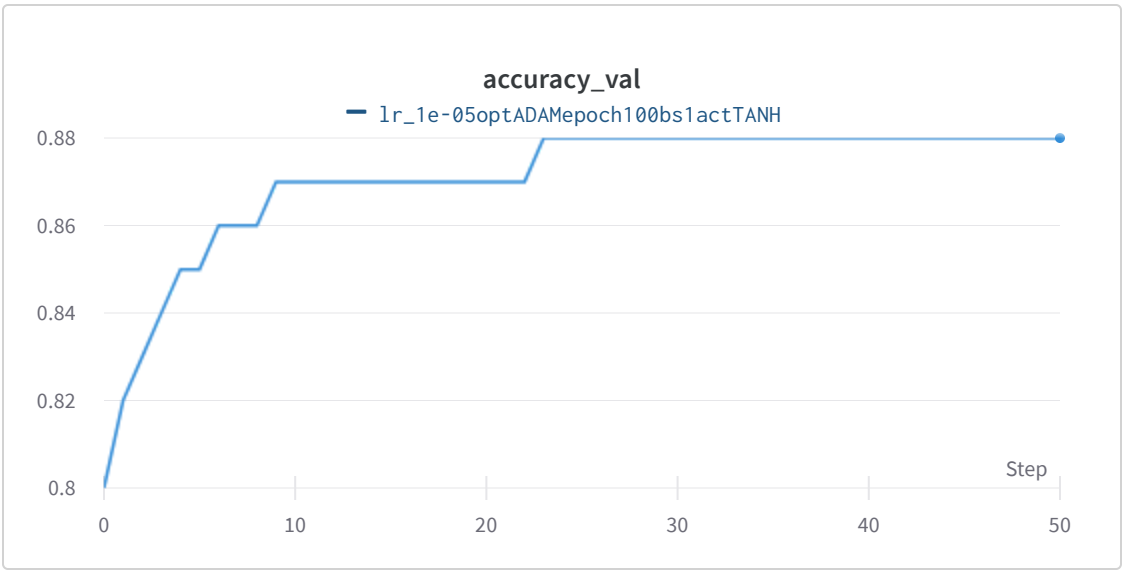
Question 5 (5 marks)

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







From the Primary and Secondary sweep analysis it was observed that Stochastic version of Adam optimization algorithm gave the best accuracy. The ADAM algorithm was used for training the Neural Network with 2 hidden layers containing 64 neurons each. Tanh activation function was used and the accuracies obtained during the training process have been shown in the plot above. After 50 epochs an accuracy of 0.88 was obtained on the validation set.

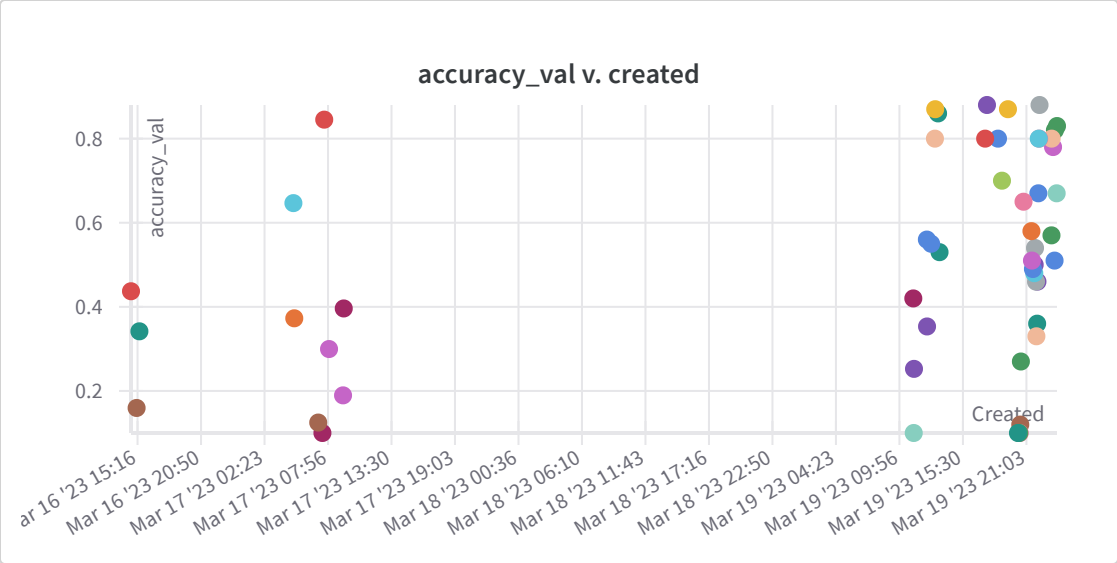
Question 6 (20 Marks)

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.



From the primary and secondary sweeps it was learnt that the stochastic version of the optimization algorithms showed better performance compared to the batched versions. Performance of simple SGD was close to other sophisticated algorithms such as ADAM, NADAM and RMSPROP.

It was observed that the RELU activation function required very low learning rate. Since the performance of the optimization algorithms was not good with RELU activation function it was dropped from subsequent analysis.

It could be noted that choosing an appropriate value for learning rate was paramount. A larger learning rate often led to numerical instabilities and a smaller learning rate lead to slower training. It was crucial to arrive at a logical value for the learning rate.

As mentioned earlier, the stochastic variants of the different optimization algorithms gave better results compared to batched versions. This might be because the stochastic variants update the parameters more often leading to better optimization and reduction of loss.

It could be noted that ADAM performed better than other algorithms. This shows the power of adaptive learning rates which improves the speed and stability of the learning algorithm. The scaling of the learning rate helps to normalize the gradients and prevent large changes in the weight values during optimization and the bias correction step helps to improve the accuracy of the estimates of the first and second moments of the gradients.

It was also observed that using a deeper neural network gave better results compared to flatter neural networks. However, it also increased training time so a reasonable trade-off is required.

RECOMMENDATIONS FOR IMPORVEMENT:

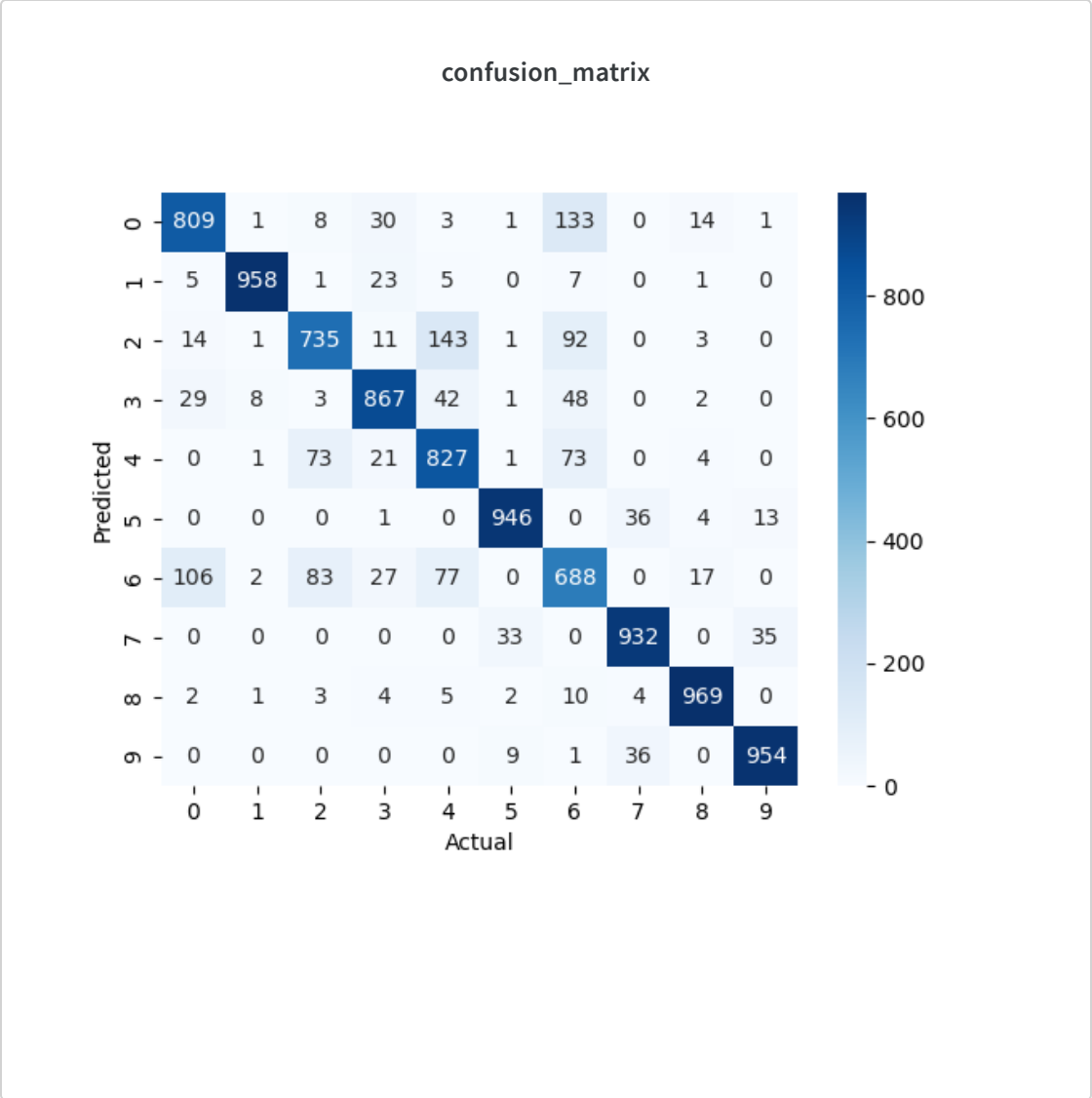
During training it was observed that the validation loss was higher than the training loss. This could be because of the network overfitting to the training data. Though L2 regularization was applied, it was not sufficient to push the accuracy to more than 90% within reasonable training time. Several techniques are available in Deep learning to prevent over fitting viz. Data Augmentation, Dropout, batch normalization, Early stopping, etc. Some of these methods could be tried out to reduce overfitting and obtain higher accuracies.

Additionally, learning rate scheduling or cyclical learning rates could be used to make sure that the optimization algorithms are not stuck in at a local minima.

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)

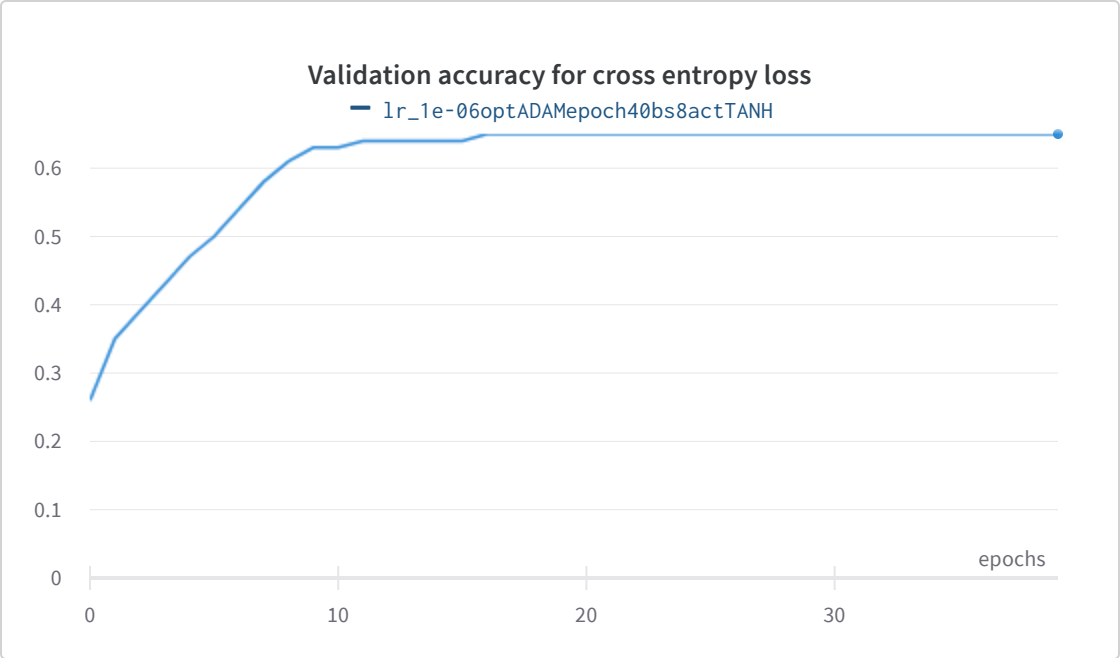
Confusion matrix created on the test set

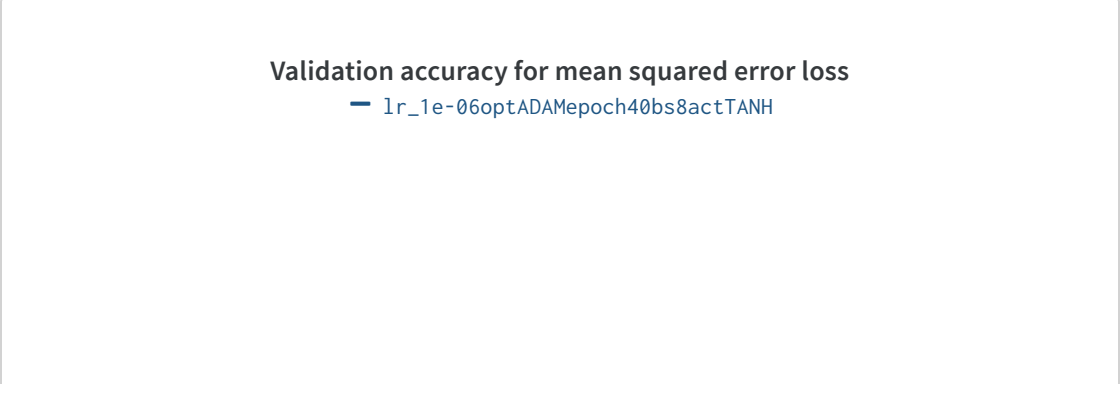


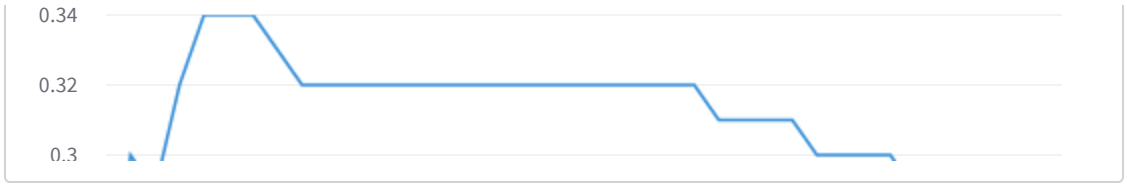
Question 8 (5 Marks)

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.

From the plots shown below it can be observed that the cross entropy loss in better suited for multi-class classification problems when compared to mean squared error loss.







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

https://github.com/arunm917/CS6910_Assignment1

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 myprojectname
```

Arguments to be supported

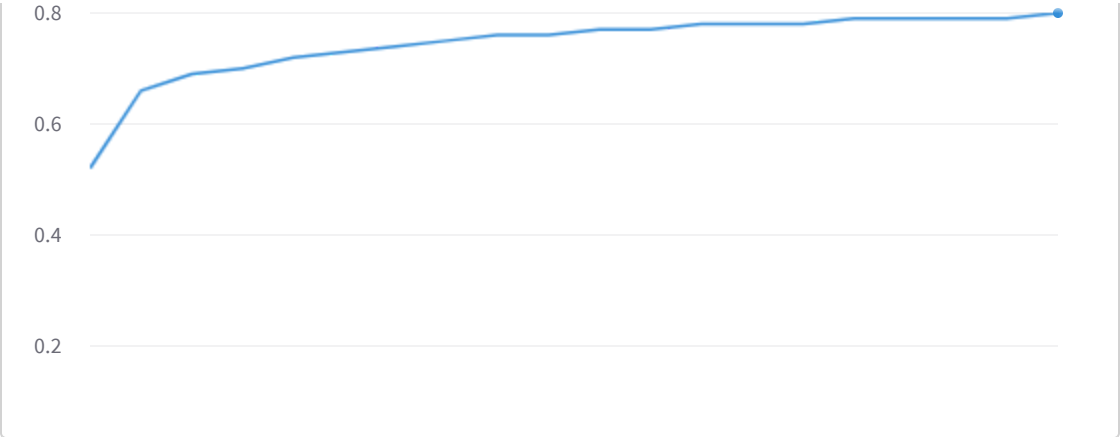
Name	Default Value	Description
`-wp`, `--wandb_project`	myprojectname	Project name used to track experiments in Weights & Biases dashboard
`-we`, `--wandb_entity`	myname	Wandb Entity used to track experiments in the Weights & Biases dashboard.
`-d`, `--dataset`	fashion_mnist	choices: ["mnist", "fashion_mnist"]
`-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"]

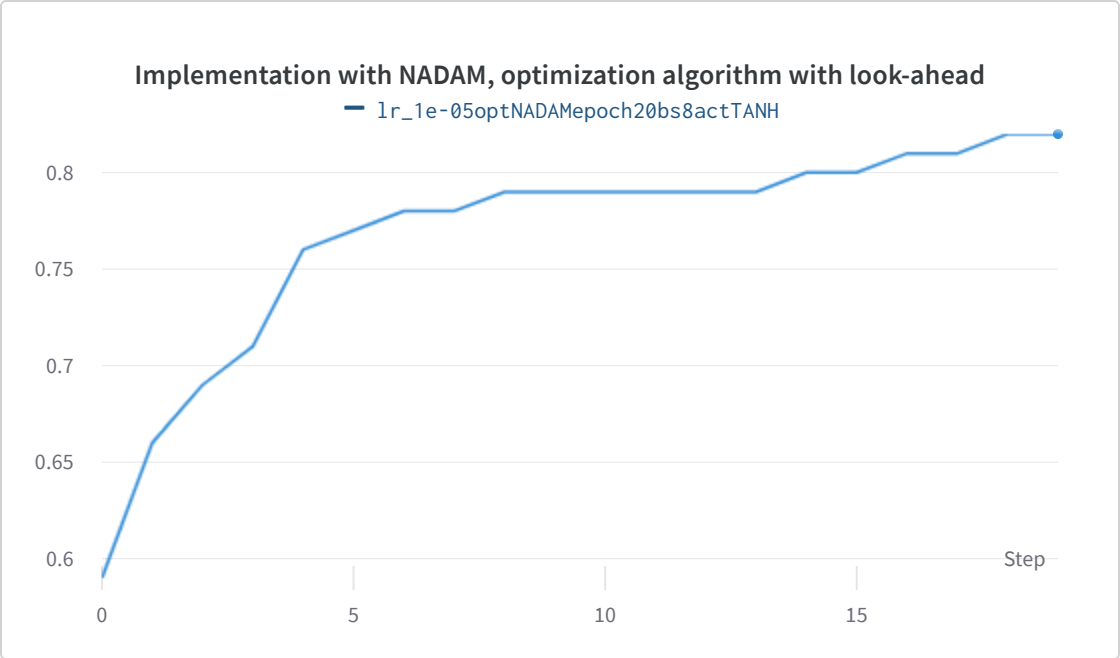
Name	Default Value	Description
`-lr`, `--learning_rate`	0.1	Learning rate used to optimize model parameters
`-m`, `--momentum`	0.5	Momentum used by momentum and nag optimizers.
`-beta`, `--beta`	0.5	Beta used by rmsprop optimizer
`-beta1`, `--beta1`	0.5	Beta1 used by adam and nadam optimizers.
`-beta2`, `--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"]

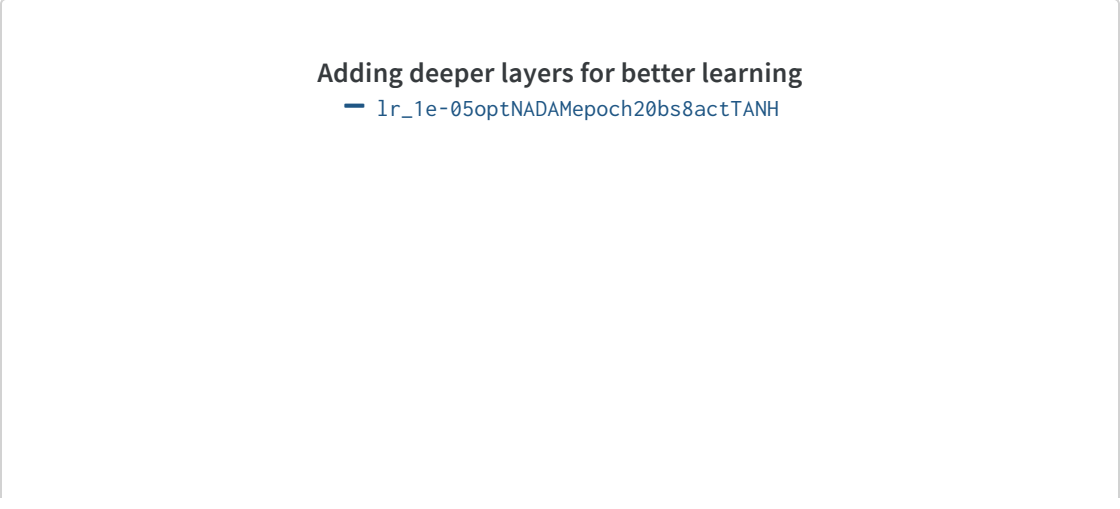
Please set the default hyperparameters to the values that give you your best validation accuracy. (Hint: Refer to the Wandb sweeps conducted.)

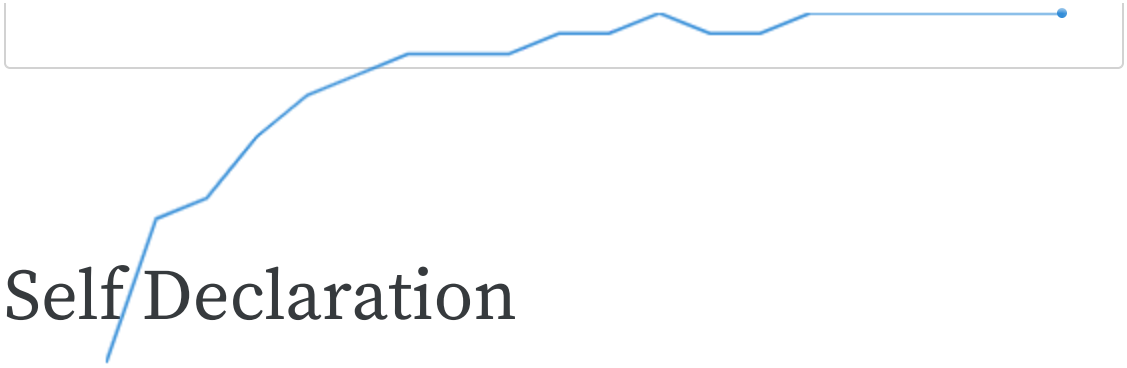
You may also add additional arguments with appropriate default values.

Implementation with higher weight decay to prevent overfitting
— `lr_1e-05optADAMepoch20bs8actTANH`


0.8
0.6
0.4
0.2


Implementation with NADAM, optimization algorithm with look-ahead
lr_1e-05optNADAMepoch20bs8actTANH
0.8
0.75
0.7
0.65
0.6
Step
0
5
10
15


Adding deeper layers for better learning
lr_1e-05optNADAMepoch20bs8actTANH



Self Declaration

I, Arun M., 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.