Draft autosaved just now Save

This report contains Markdown blocks. Would you like to convert them into WYSIWYG? Convert Dismiss

CS6910 - Assignment 1

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

Viswanathan S

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.

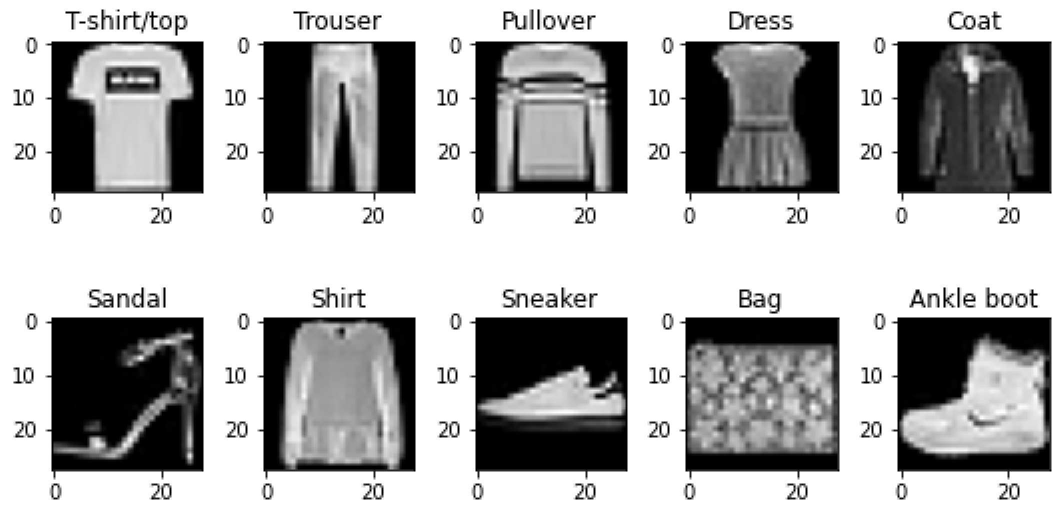
Problem Statement

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.



Fashion MNIST classes

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.

Implemented in code.

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 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.

Implemented in code.

Question 4 (10 Marks)

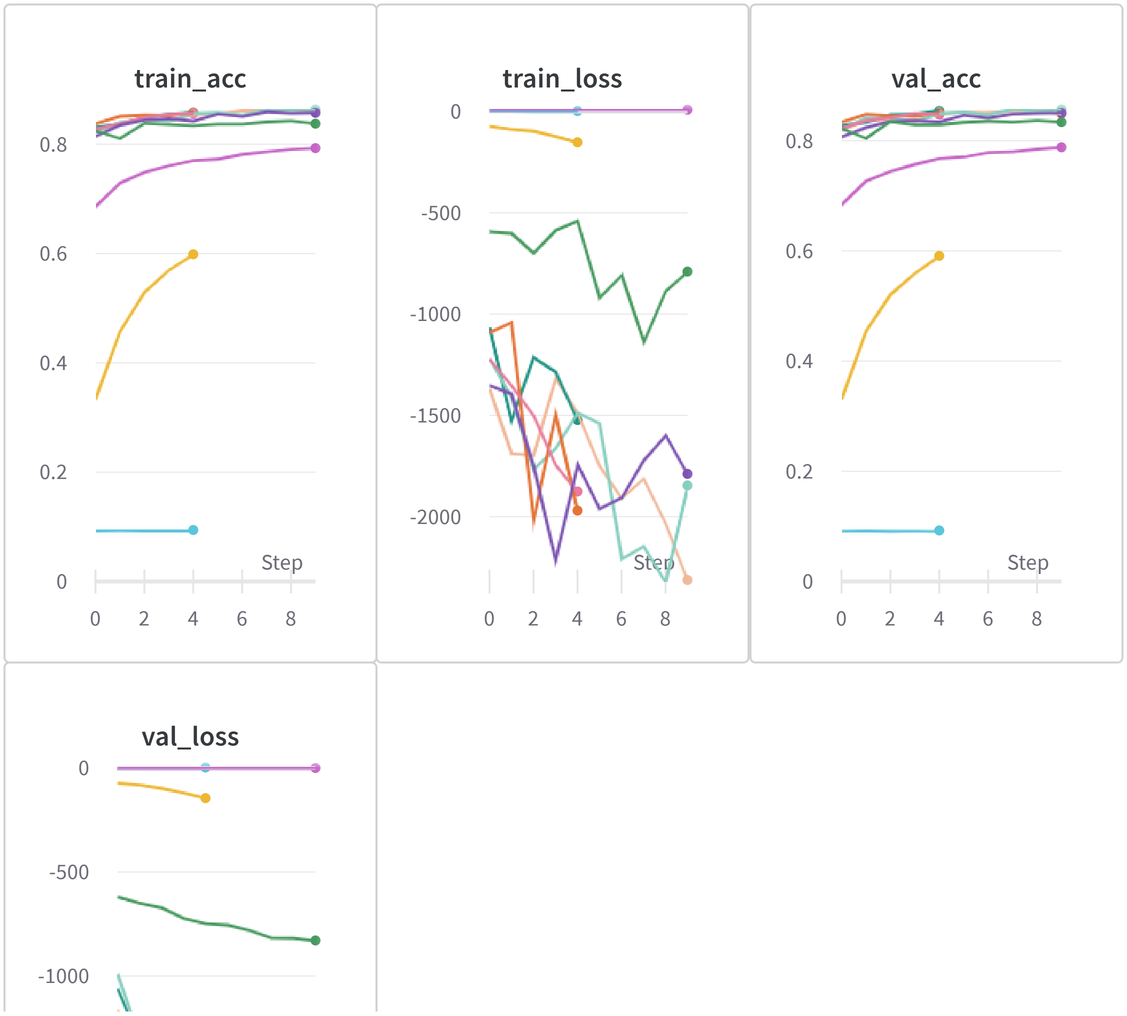
Use the sweep functionality provided by wandb to find the best values for the hyperparameters listed below. Use the standard train/test split of fashion_mnist (use (X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()). Keep 10% of the training data aside as validation data for this hyperparameter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

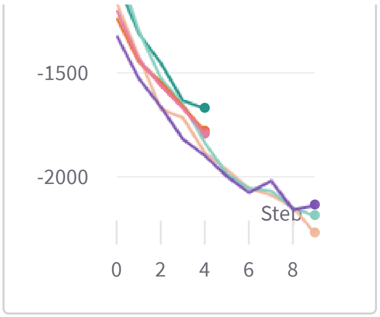
- number of epochs: 5,10
- number of hidden layers: 3,4,5
- size of every hidden layer: 32,64,128
- weight decay: 0,5e-4,0.5
- learning rate: 1e-3,1e-4
- optimizer: sgd, momentum, nesterov, rmsprop, adam, nadam
- batch size: 16, 32, 64
- weight initialization: random, xavier
- activation functions: sigmoid, 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 have used the Bayesian strategy to sweep the hyperparameters. It is a method that explores the search space more efficiently than a grid search or random search, by using the past evaluation results of the hyperparameters to guide the search towards regions of the search space that are more likely to yield better results.

In Bayesian hyperparameter search, the search space is defined by a set of hyperparameters, and for each combination of hyperparameters, the corresponding model is trained and evaluated on a validation set. The evaluation metric is then used to update the probability distribution over the hyperparameters, which guides the search towards the more promising regions of the search space. This process is repeated for a number of iterations, and the search terminates when the specified budget (e.g., number of trials) is exhausted.





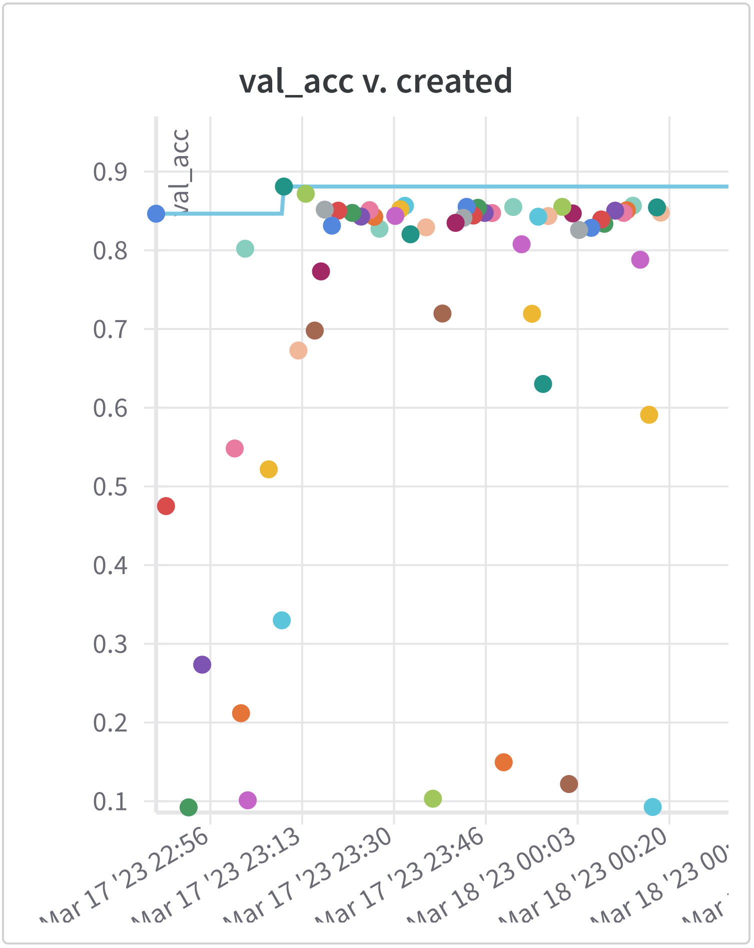
Import panel Add panel

Run set 60

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 summarizes the test accuracy of all the models that you tested. Please paste this plot below using the "Add Panel to Report" feature.



Import panel

Add panel

Run set 60

Question 6 (20 Marks)

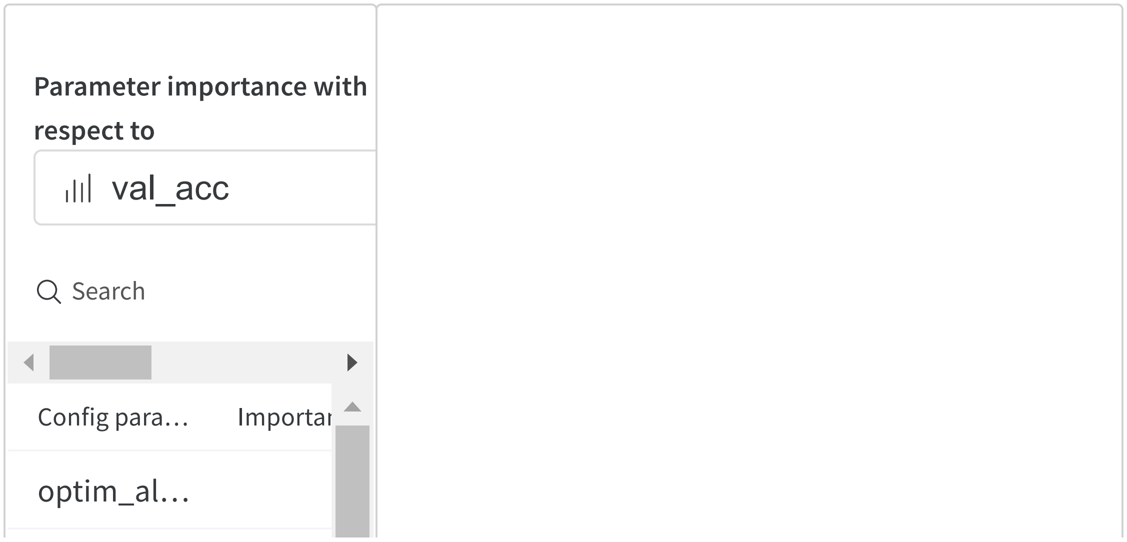
Based on the different experiments that you have run we want you to make some inferences about which configurations worked and which did not.

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.

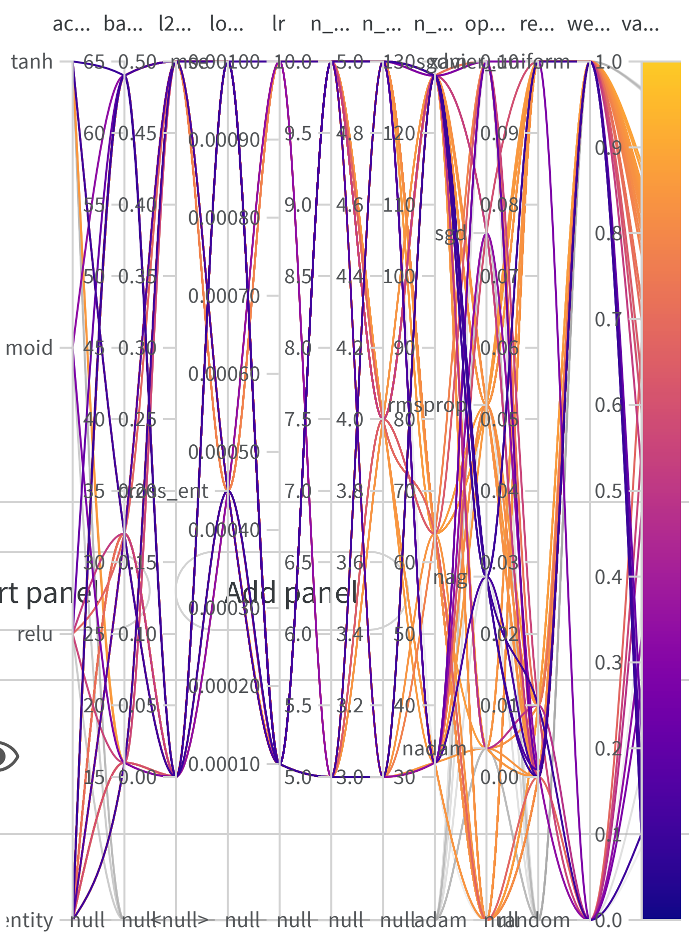
- The first observation that we can make is that the mse loss function is consistently out-performing the cross-entropy function. This may be because the dataset is balanced *i.e.,* all classes have same number of training examples.
- From parameter importance plot, we can see that the choice of optimizer, weight initialization strategy and activation function are the three most important parameters which affect validation accuracy. Only optimizing these we can reduce search space when we want to train the network in similar tasks like MNIST.
- The weight decay parameter has the least importance which signifies that even when the network gets over-trained it does not affect the validation accuracy much. It may be because the samples in the training dataset are not diverse enough.
- The way the hyperparameter search is done as can be seen from question 5 is that its not a brute force search for hyperparameter. The search is tunes in the neighborhood of the current best performance.
- Choosing the number of epochs more than 10 can take the validation accuracy closer to 95%. The best configuration (88.1% validation accuracy) is given below.
- *activation: tanh, batch_size:16, weight_decay:0, loss: 'mse', lr:1e-4, n_epochs:10, n_hidden:5, n_hidden_units:64, optimizer: rmsprop, weight_initialization: xavier.*


Parameter importance with respect to

val_acc

Search

Config para…	Importar
optim_al…	



weights_…

act_func.…

act_func.…

optim_al…

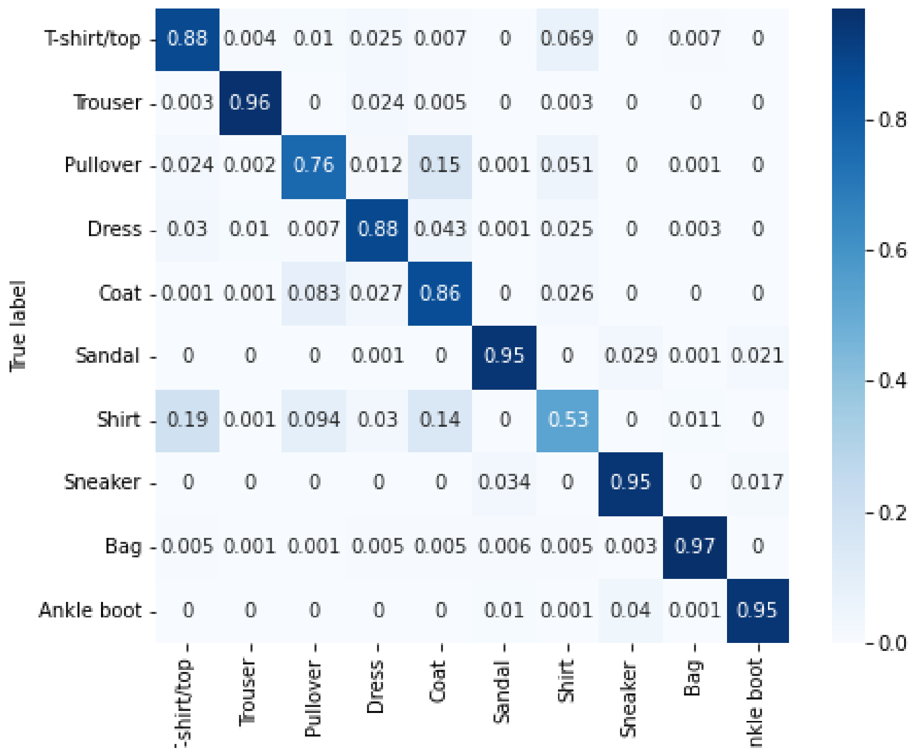
Import panel

Add panel

Run set 59

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 on test data for the best model in the list of sweeps

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.

The panel grids above are arranged in descending order with respect to validation loss and the choice of loss function is included as a hyper-parameter to sweep for the Wandb agent. It can be seen that, the first five entries are optimized with mean squared error as the objective function. To understand this, we have to look at the gradient of the objective functions. Assuming θ is the model parameters (weights and biases),

$$\nabla_\theta loss_{mse} = 2 \times (y_{true} - y_{pred})$$

$$\nabla_\theta loss_{cross-entropy} = y_{true} - y_{pred}$$

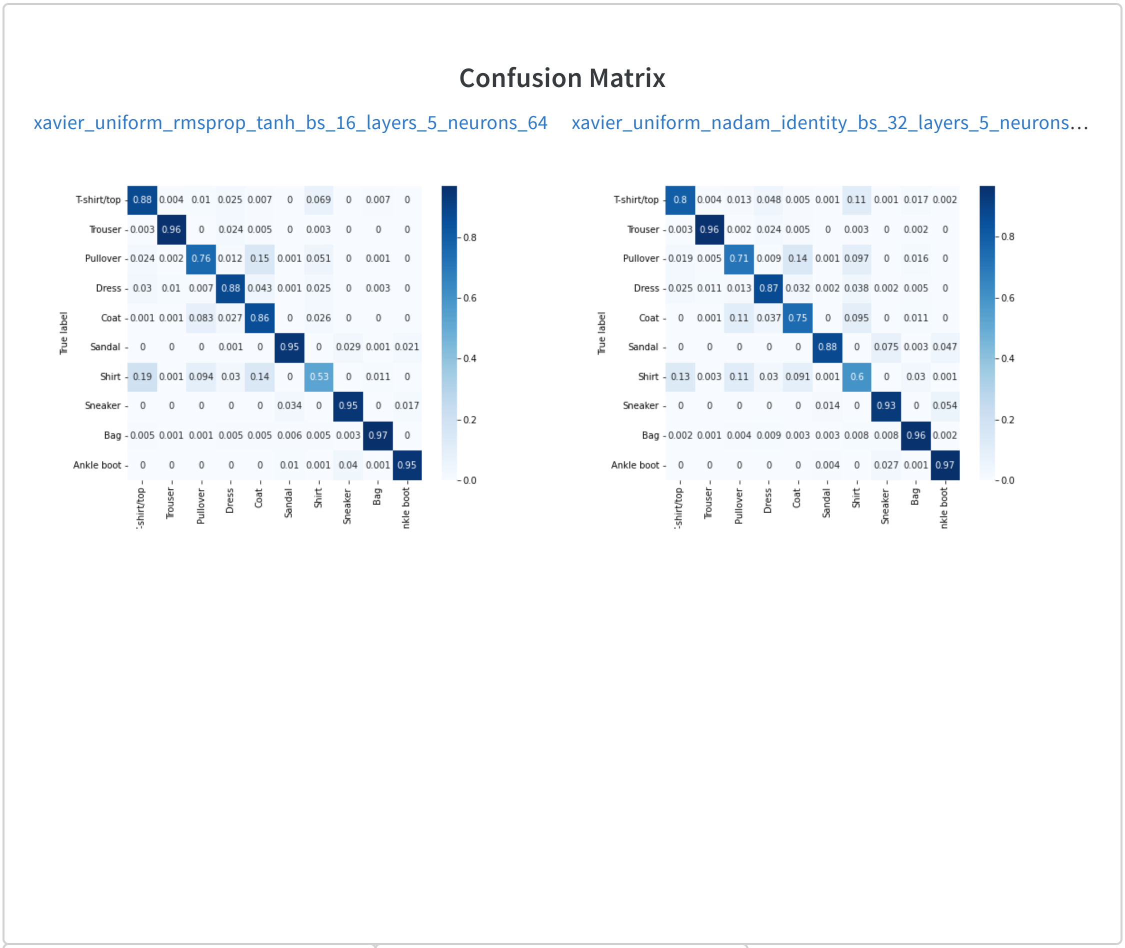
We can see that $\nabla_\theta loss_{mse} = 2 \times \nabla_\theta loss_{cross-entropy}$ which means that the difference in optimizing the two objectives is the gradients are scaled.

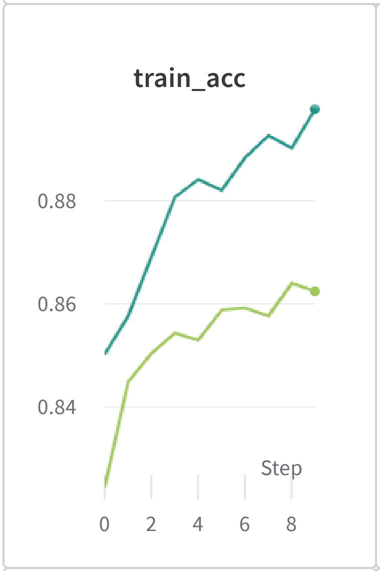
Two reasons can be deduced for why *mean squared error is working better or on-par with cross entropy*

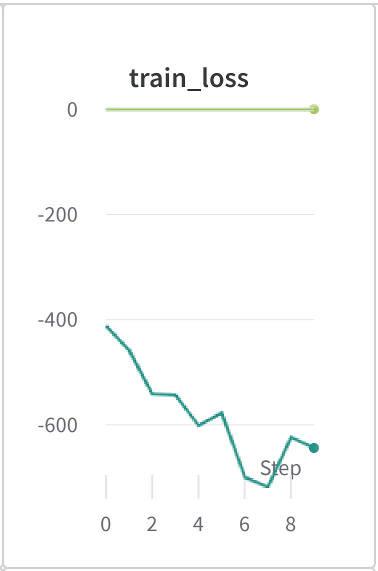
- Output Encoding: MSE loss is commonly used for regression problems, where the output variable is a continuous value. In the case of Fashion-MNIST, each image corresponds to a specific class label. If we encode these labels as continuous values (e.g., 0.0 for "T-shirt/top," 1.0 for "Trouser," etc.), then MSE loss can be used. This encoding is different from the one-hot encoding used with cross-entropy loss, where each label is represented as a vector of zeros with a single one at the index of the corresponding class.
- Class Imbalance: Fashion-MNIST has a relatively balanced distribution of classes, with each class having roughly the same number of examples. In such a case, cross-entropy loss can work well since it penalizes the model more for misclassifying rare classes. However, if the dataset had a significant class imbalance, where some classes have much fewer examples than others, then MSE loss can work better because it treats all errors equally, regardless of the class.

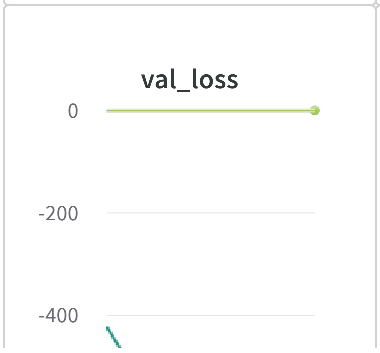
Overall, the choice of loss function depends on the problem at hand and the characteristics of the dataset. While cross-entropy loss is commonly used for classification tasks, MSE loss can be a valid alternative in certain cases, such as when the output encoding is continuous or when the dataset has a balanced distribution of classes.

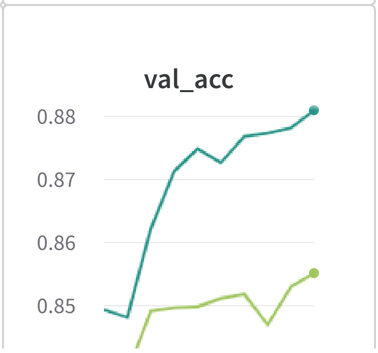
The confusion matrix for best run using mse and best run using cross-entropy is shown below

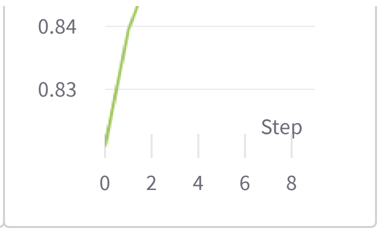




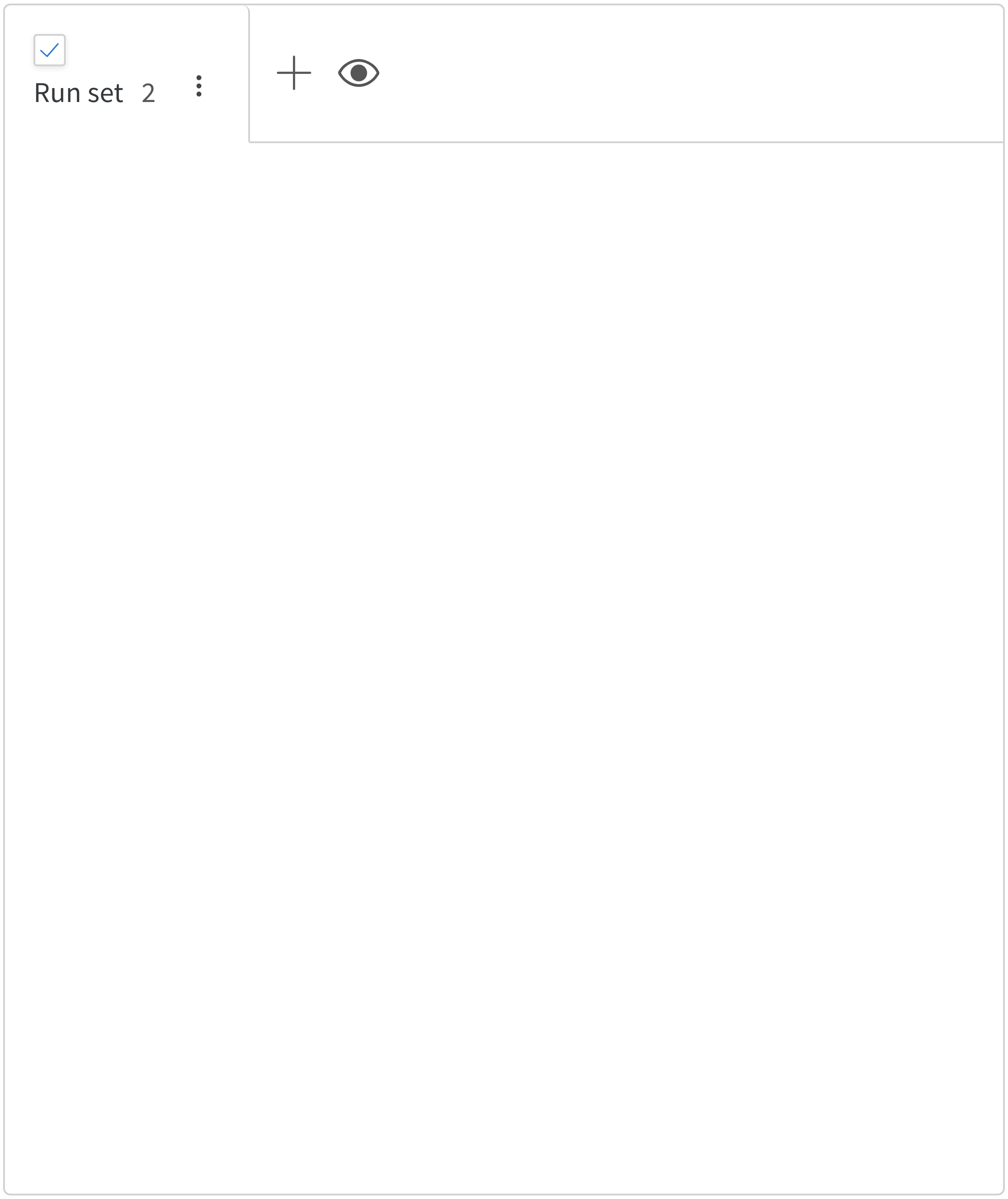








Import panel   Add panel



Question 9 (10 Marks)

Paste a link to your github code for this assignment

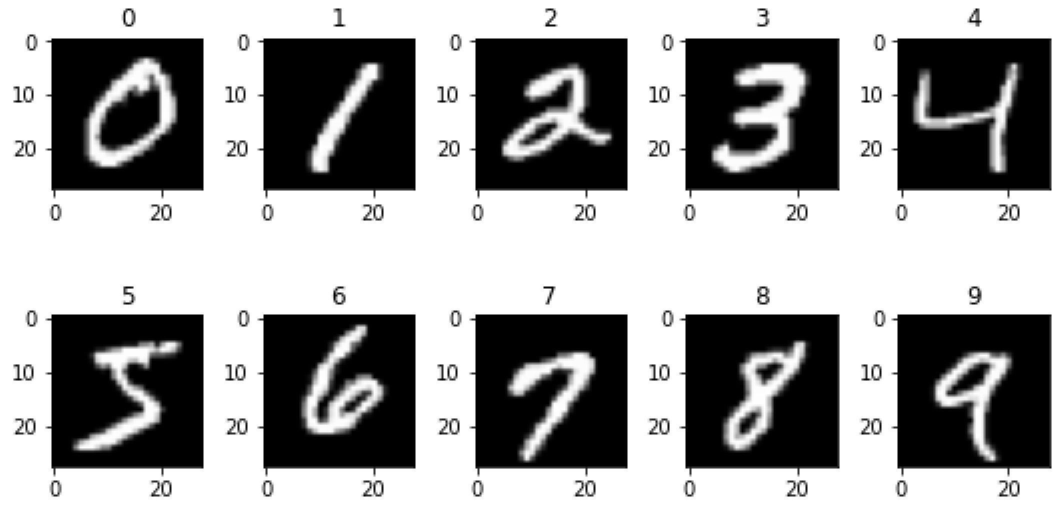
Link to code:
https://github.com/Vishwanath1999/cs6901_assignment1

- Paste a link to your github code for this assignment
- 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

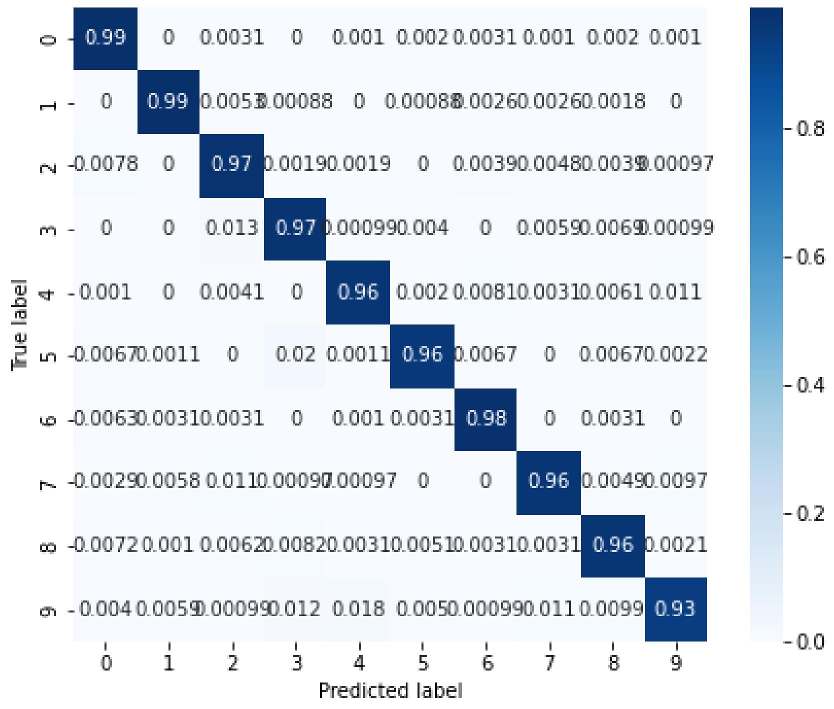
Question 10 (10 Marks)

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.

From the correlation plot above we can infer that activation function, optimizer and weight initialization are the three most important parameters. Running hyper-parameter sweep on those three parameters yielded the following results. The highest validation accuracy is obtained with XAVIER initialization, NADAM optimization and tanh activation function. All the other parameters are kept constant as the best run obtained from fashion MNIST.



MNIST classes

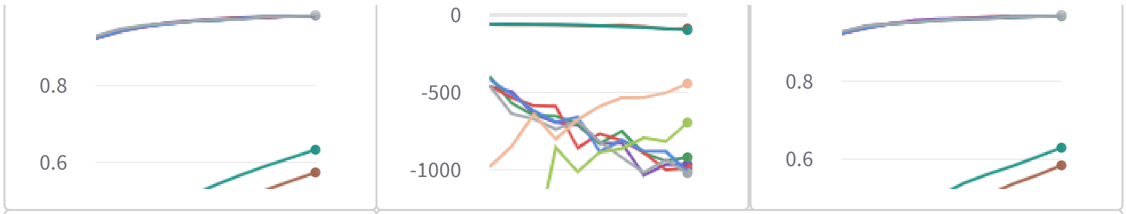


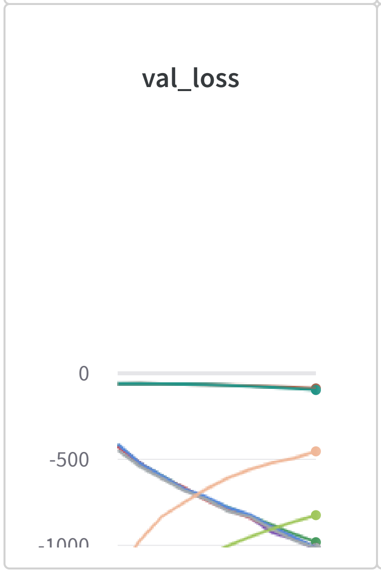
MNIST Confusion matrix for best run

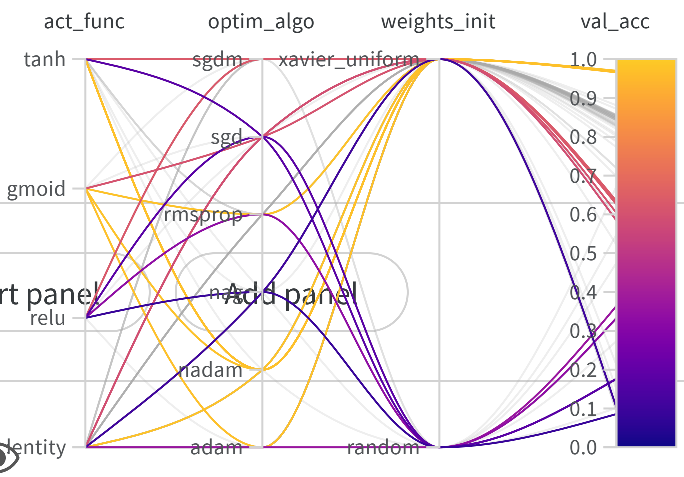
train_acc

train_loss

val_acc







Import panel

Add panel

Run set 20

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"]

Name	Default Value	Description
`-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.)

Self Declaration

I, S Viswanathan, swear on my honor that I have written the code and the report by myself and have not copied it from the internet or other students.