Share Comment Star

Sudhanva Satish DA24M023 | DA6401 - Assignment 1

Report for DA6401 Intro to DL Assignment 1

Sudhanva Satish da24m023

Created on March 8 | Last edited on March 9

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.

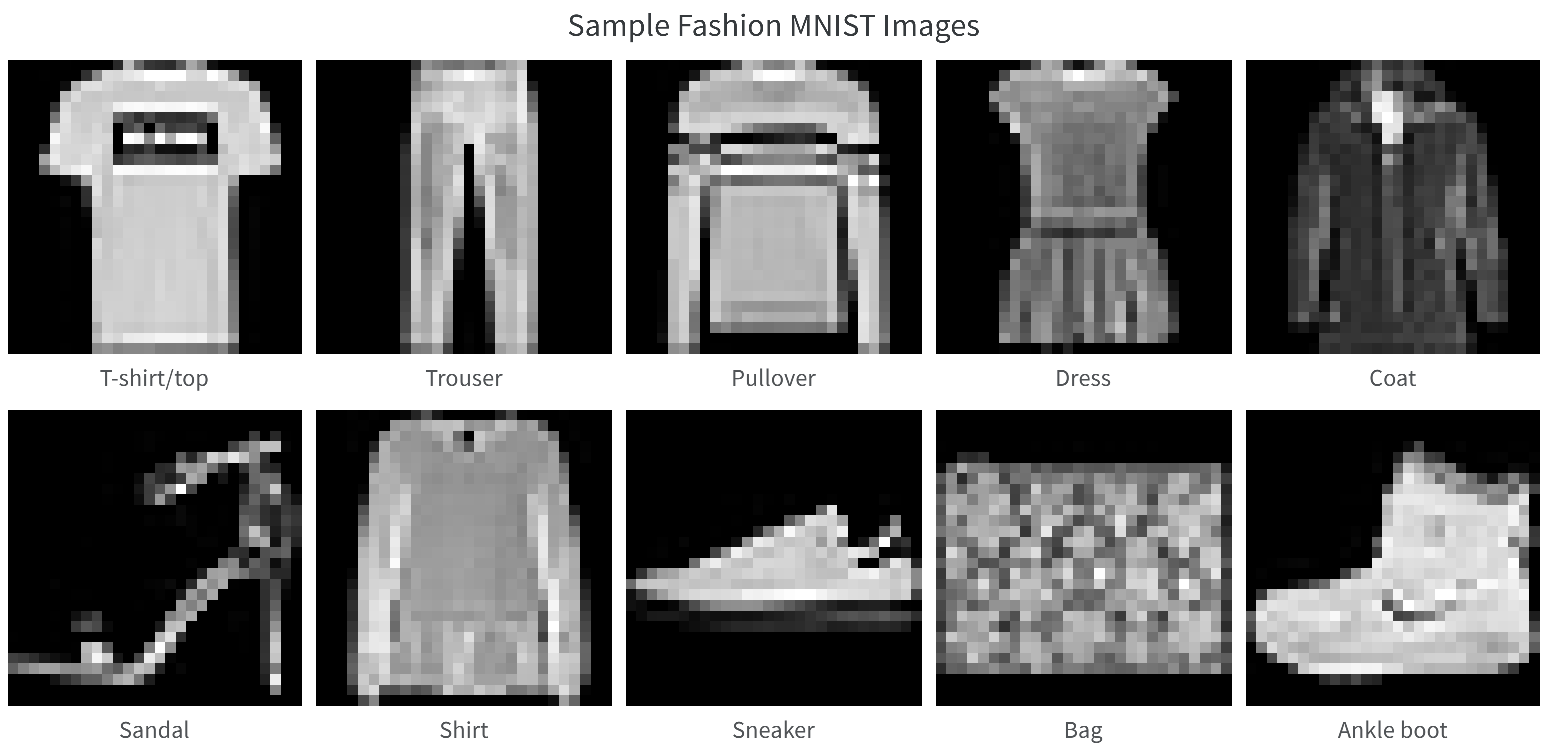
Your code will have to follow the format specified in the `Code Specifications` section.

Question 1 (2 Marks)

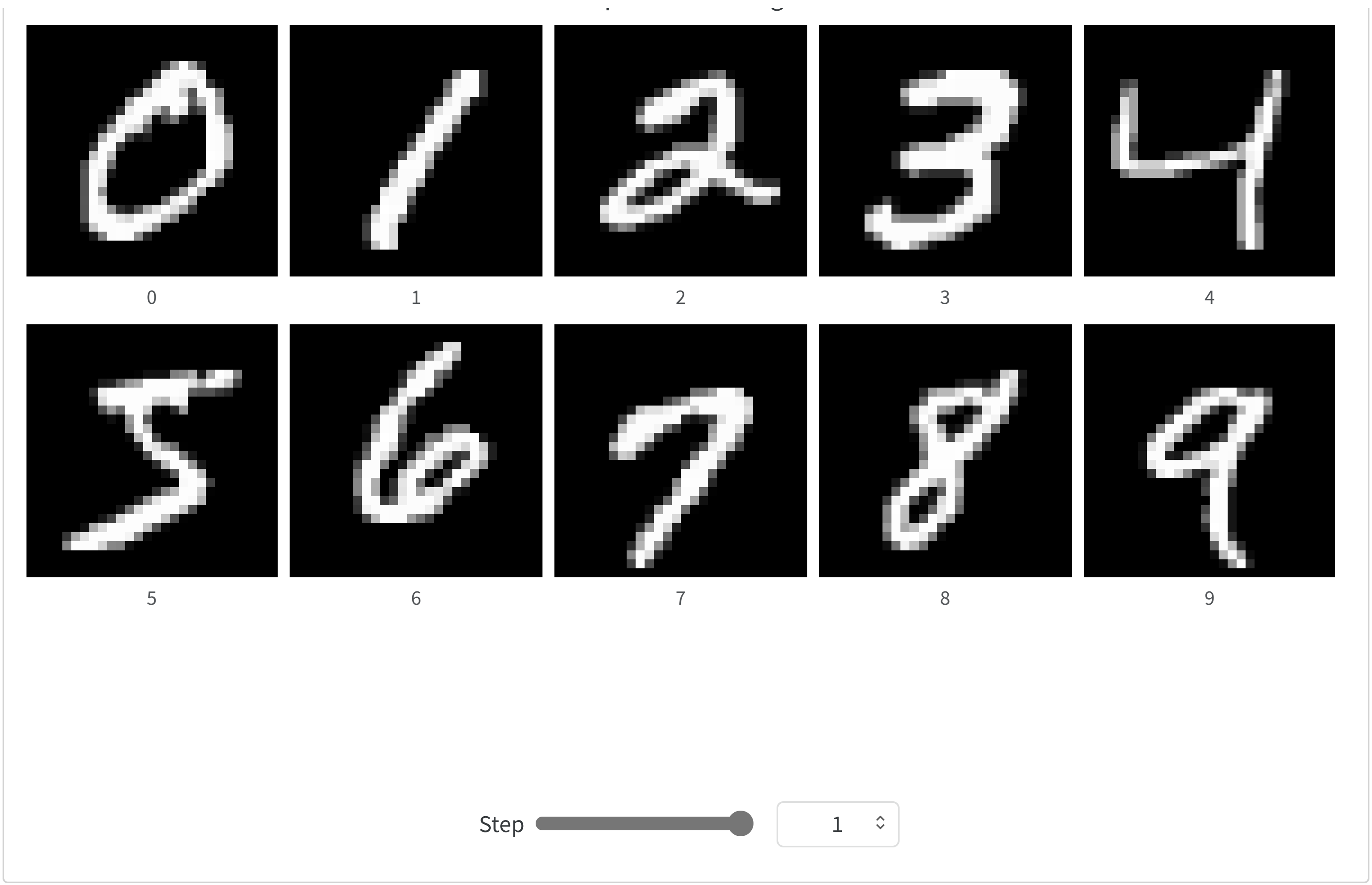
Download the fashion-MNIST dataset and plot 1 sample image for each class as shown in the grid below. Use `from keras.datasets import fashion_mnist` for getting the fashion mnist dataset.

Answer

The first image of each class in the dataset along with it's class label has been plotted below. Same has been done for the MNIST dataset as well.



Sample MNIST Images



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.

Answer

- The required neural network has been implemented in the code submitted.
- To change the number of hidden layers, give the command line argument "--num_layers "
- To change the number of neurons in the hidden layers, give the command line argument "--hidden_size "
- Note - All the hidden layers will have same number of neurons as specified in the CLI table given below

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.

Answer

- The Backpropogation algorithm has been implemented as a method of the Neural Network class.

```
NeuralNetwork.backward() method
* Inputs: Takes current data batch x and true labels y as parameters.
Uses the stored activations and pre-activations of each layer to compute the error at the fin
and backpropagate it through the network.
* Output: dW, db - gradients of weights and biases
```

- To add a new optimizer, in the optimizers.py file create a new optimizer class containing a method update_parameters(). This method should take the NeuralNetwork object, and gradient vectors dW and db as input. If any parameters other than the already existing ones are needed, create them as instance variables of the NeuralNetwork class and use them according to the optimizer's logic. The final return values should be dW and db.
- In the get_optimizer() helper function, add the new optimizer in the optimizers dictionary as a 'name': Class() pair.
- The code is also compatible for different batch sizes that can be given via the CLI argument --batch_size

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.

Answer

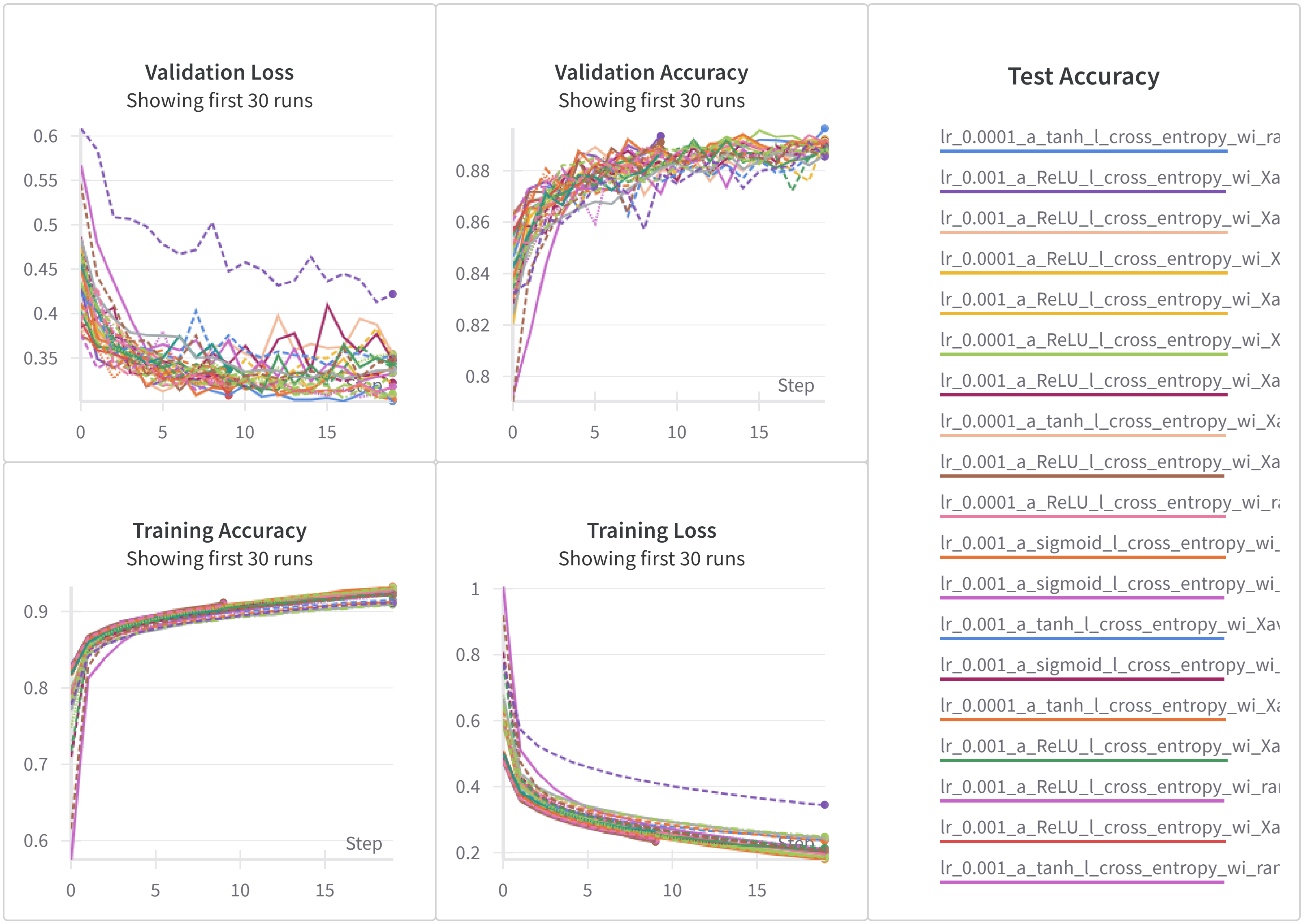
- The sweep functionality has been implemented to find the best set of hyperparameters for the model.
- The sweep has a huge hyperparameter search space so it is highly time-consuming. However wandb provides some methods to optimize the sweeping process:-

    1. **Grid sweep**: This is the usual method of creating a grid representing all possible hyperparameter combinations. It is the most thorough but also the most computationally demanding and time consuming method
    2. **Random sweep**: Random hyperparameter values are chosen from the given list of values for each of them and the model is evaluated accordingly.
    3. **Bayes sweep:** This method uses Bayesian probability to efficiently choose the next hyperparameter combination based on our knowledge of which combinations previously gave good performance. In this way it navigates through the search space finding better combinations to test with after each evaluation.

- I have chosen to use Bayes method with the parameter goal for val_acc as 'maximize' for sweeping as it is more efficient than the other 2. It maximizes the chance that better parameter configurations will be chosen as the sweep progresses. This is seen in the graph in the next question.
- Naming convention for each run in the sweep: "lr*{}_a*{}*l*{}*wi*{}*o*{}*b*{}*wd*{}*e*{}*nhl*{}*sz*{}*d*{}" where each abbreviation denotes a parameter as per the coding specifications mentioned

in the table given at the end of the report.

- Ex: lr_0.001_act_tanh means learning rate=0.001 and activation function = tanh



Question 5 (5 marks)

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

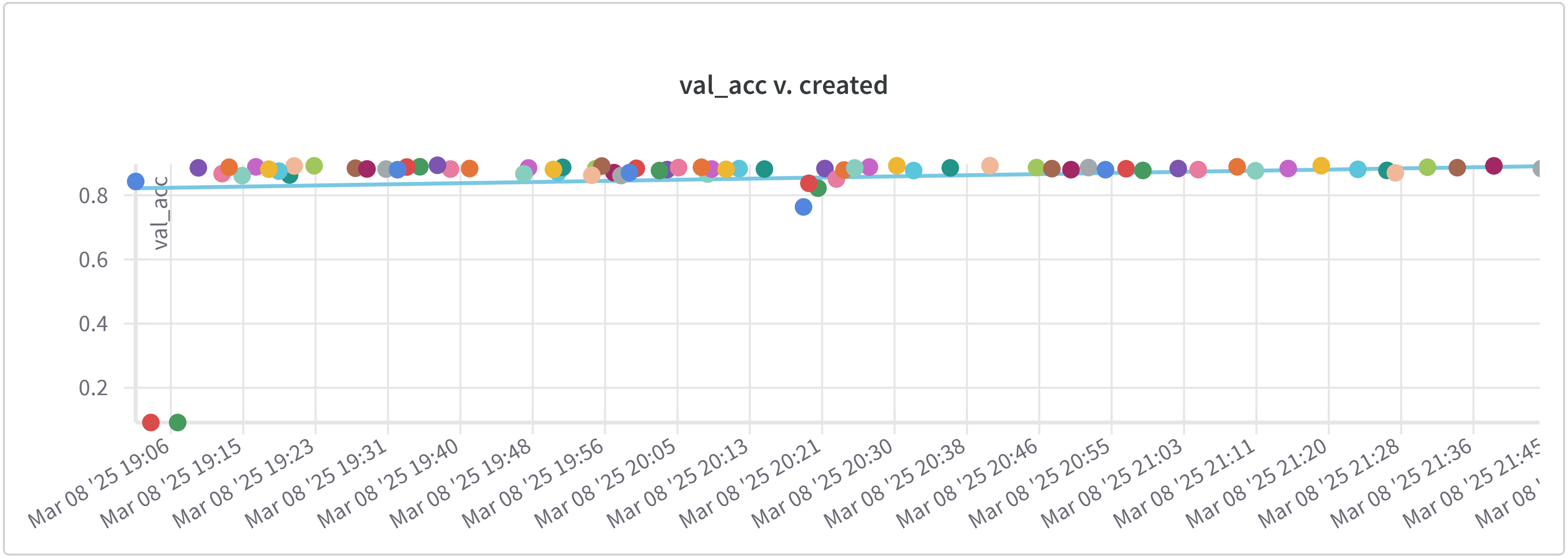
wandb automatically generates this plot which summarises the test accuracy of all the models that you tested. Please paste this plot below using the "Add Panel to Report" feature

Answer

Upon analyzing the results of the sweeps run, the best performing model which gave a validation accuracy of 89.65%, training accuracy of 92.65% and testing accuracy of 88.58% had hyperparameters as follows:

```
* activation: "tanh"
* batch_size: 16
* epochs: 20
* hidden_layers: 3
* hidden_size: 128
* learning_rate: 0.0001
* loss_fn: "cross_entropy"
* momentum: 0.9
* beta1: 0.9
* beta2: 0.999
* optimizer: "adam"
* weight_init: "random"
```

- Another interesting thing to note is that when a regression line (light blue line in the graph) is drawn to fit the trend of val_acc vs timestamp of the runs, we get an upward trend which verifies that Bayes sweeping, as claimed, is choosing better parameter combinations for every subsequent 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.

Answer

- The best configuration was:

```
activation:"tanh"
batch_size:16
epochs:20
hidden_layers:3
hidden_size:128
learning_rate:0.0001
momentum:0.9
beta1: 0.9
beta2: 0.999
optimizer:"adam"
weight_decay:0.0005
weight_init:"random"
```

- The worst configuration was:

```
activation:"tanh"
batch_size:64
epochs:20
```

```
hidden_layers:5
hidden_size:128
learning_rate:0.0001
momentum:0.9
beta1: 0.9
beta2: 0.999
optimizer:"rmsprop"
weight_decay:0.5
weight_init:"random"
```

- Based on the plots analyzed, the following observations can be made:

    1. Higher number of epochs generally gives better performance as expected, but starts plataeuing around 15 epochs
    2. Most of the higher accuracy runs used Xavier weight initialization, whereas random initialization resulted in worse performance. This is visible in the parallel coordinates plot, however the highest performance model uses random init which according to me is an outlier because of the random factor of random initialization
    3. Tanh and ReLU activations work better than the other activations. For ReLU this is likely because of the constant derivative value which reduces impact of vanishing gradients.
    4. Weight decay factor in L2 regularization of 0.5 harshly reduced the performance as it causes rapid reduction of weight values and weight decay of 0.0005 worked much better
    5. Learning rate value of 0.0001 gave better results than 0.001 or 0.01. This is likely because higher values of learning rate tend to overshoot the global minimum of loss function

6. RMSPROP, ADAM and NADAM usually converged faster while SGD took longer, as expected
7. The runs with lesser accuracy are likely running with a combination of the lesser performing parameter values mentioned above.

- Tips to get better accuracy:

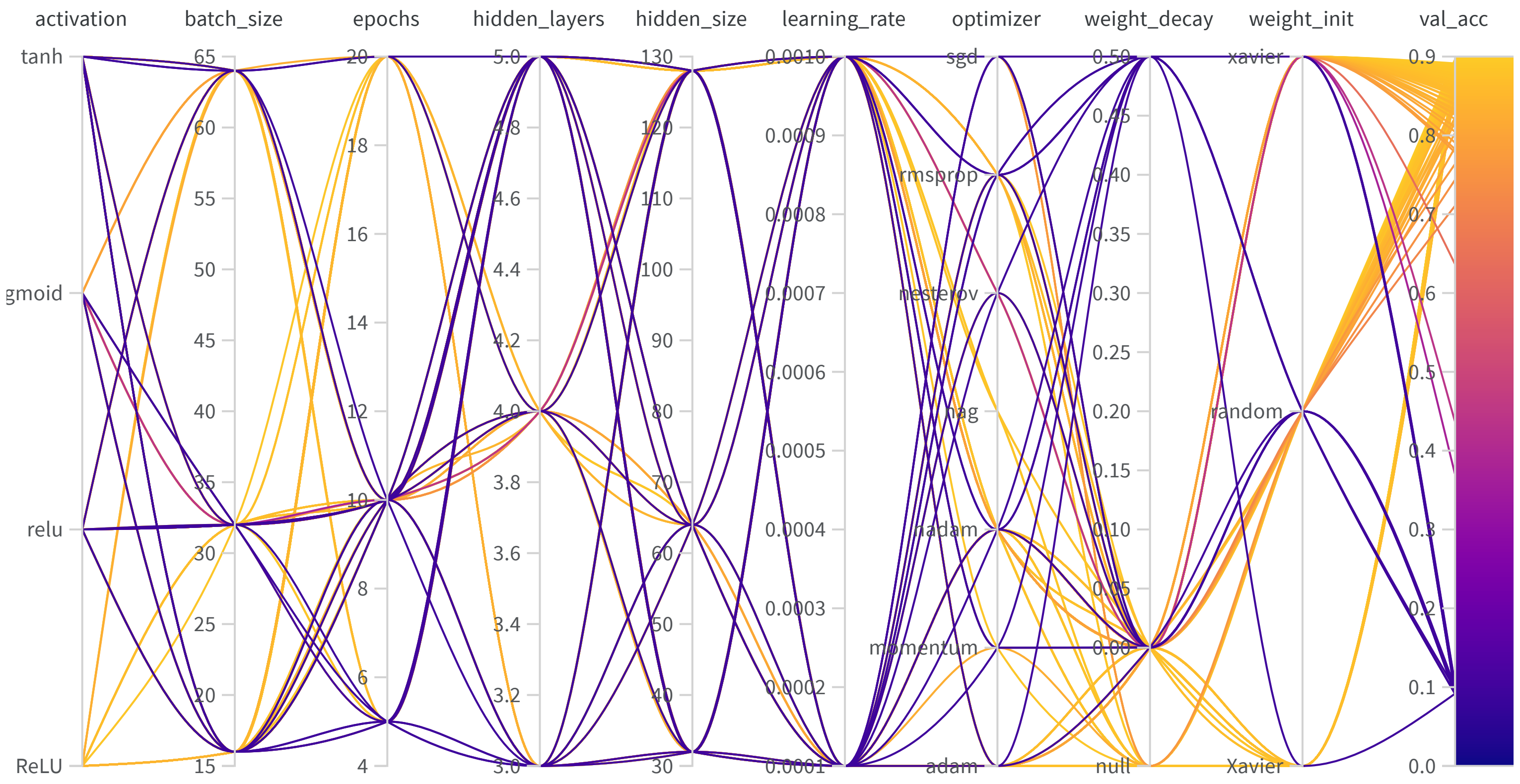
    1. Train the model for more epochs as it generally give better performance
    2. Tanh and ReLU activation give better performance than the other activations
    3. It was noted that increasing the number of hidden layers upto 3 gave better performance, but increasing further started degrading the performance
    4. I also observed during the overall course of the project that using varying number of neurons in the layers, for example 256 in h1, 128 in h2, 64 in h3 to create a sort of funnel shaped network generally gave good performance
    5. We can get several good performing models from the sweeps. Ensembling such models will give much better final performance
    6. Since this is an image dataset, using convolution layers would help.

- Further, from the hyperparameter-performance correlation chart we can also see that factors like using random weight init, sgd optimizer, momentum optimizer etc have high negative correlation whereas using more hidden layers and more neurons in the layer upto a limit have positive correlation.
- Using this info I recommend the following config to get close to 95% accuracy:

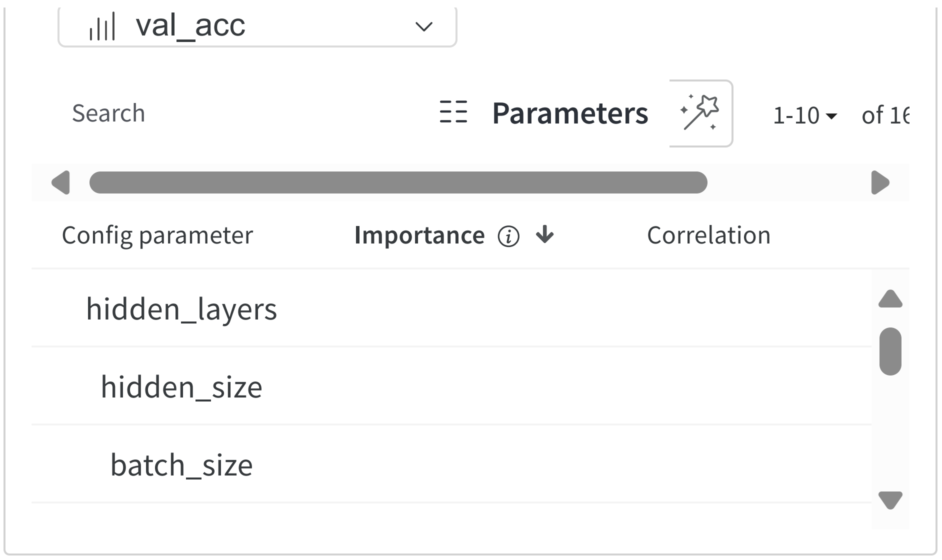
```
activation:"tanh"
batch_size:64 (bigger batches)
epochs:30 (more epochs)
```

```
hidden_layers:4 (more layers)
hidden_size:256 (and more neurons per layer for more trainable params)
learning_rate:0.0001
momentum:0.9
optimizer:"adam"
weight_decay:0.0005
weight_init:"Xavier"
```

Parallel Co-ordinates plot



Parameter importance with respect to


val_acc

Search | Parameters | 1-10 of 1€

Config parameter	Importance	Correlation
hidden_layers		
hidden_size		
batch_size		

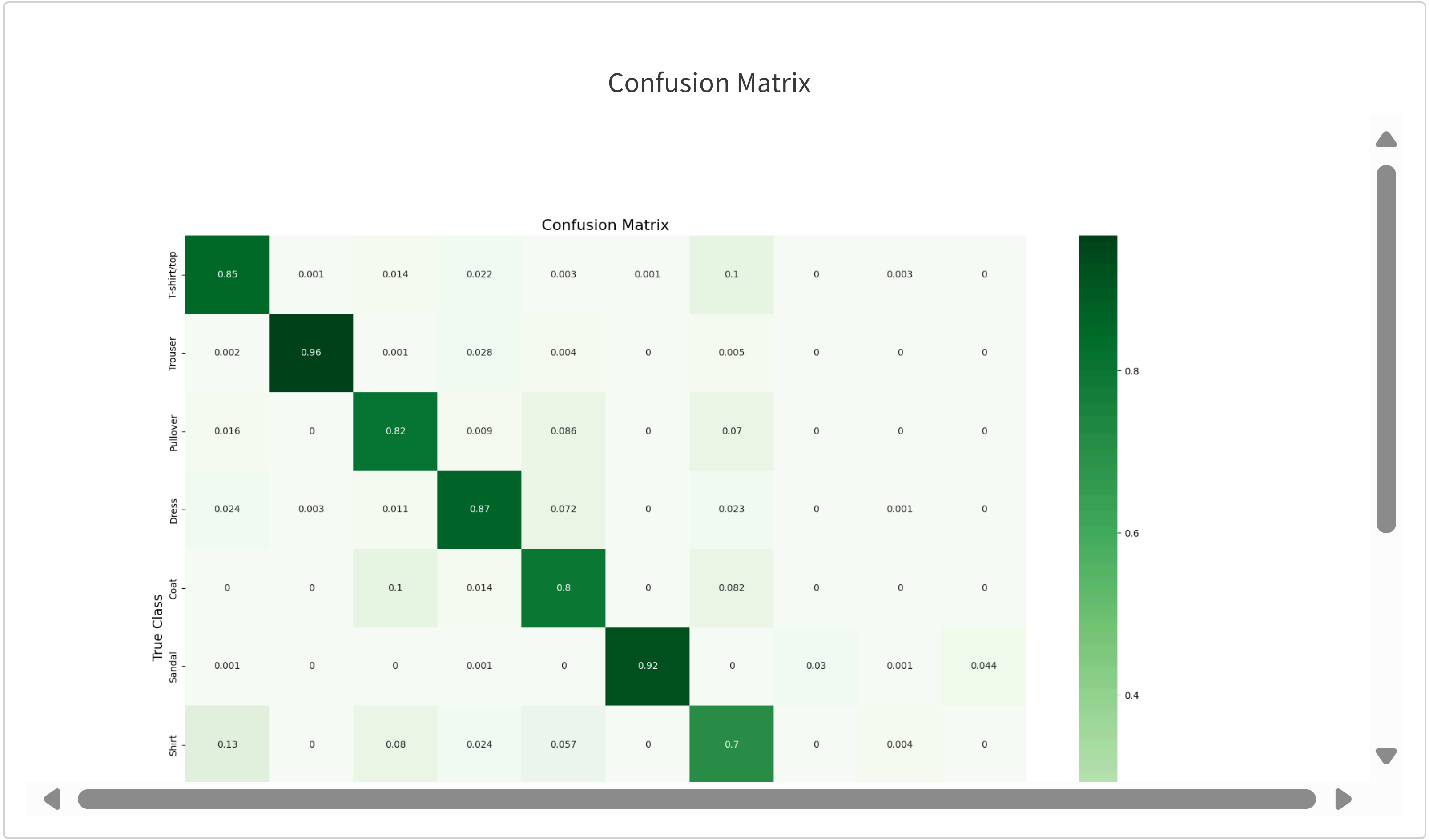
Question 7 (10 Marks)

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

- The best parameter combination model was run multiple times and test accuracy obtained was ~88.5%
- In the confusion matrix below, each cell shows the percentage of images that actually belong to the class in the corresponding row and were predicted as the class in the corresponding column

- From the confusion matrix we can see that the model predicts many classes accurately with >95% accuracy. However the classes like Shirt, Coat, Pullover etc which are visually very similar to each other seem to be getting misclassifed more often.



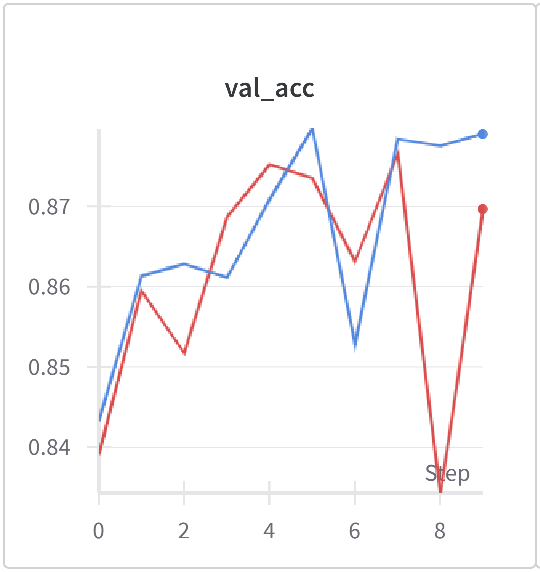
T-shirt/top	0.85	0.001	0.014	0.022	0.003	0.001	0.1	0	0.003	0
Trouser	0.002	0.96	0.001	0.028	0.004	0	0.005	0	0	0
Pullover	0.016	0	0.82	0.009	0.086	0	0.07	0	0	0
Dress	0.024	0.003	0.011	0.87	0.072	0	0.023	0	0.001	0
Coat	0	0	0.1	0.014	0.8	0	0.082	0	0	0
Sandal	0.001	0	0	0.001	0	0.92	0	0.03	0.001	0.044
Shirt	0.13	0	0.08	0.024	0.057	0	0.7	0	0.004	0

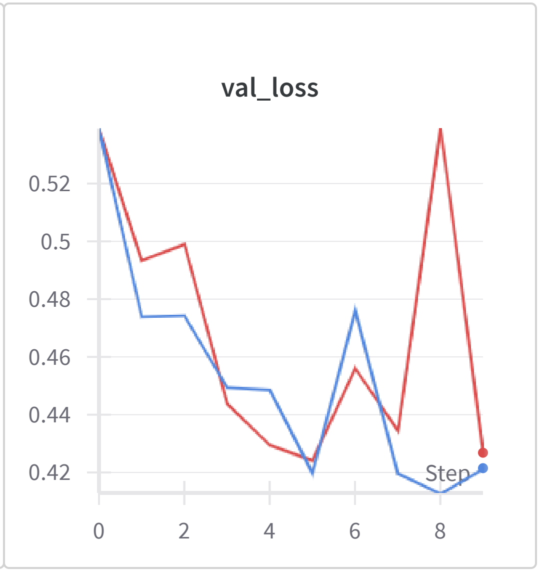
Question 8 (5 Marks)

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

- Mean squared error is a loss function that is typically used in regression problems whereas Cross Entropy loss is used in multi-class classification problems such as the fashion MNIST dataset classification.
- For this assignment, Cross Entropy would be a better option. This is because it produces larger gradients when the output is wrong thus resulting in faster convergence which is seen in the faster rise in accuracy for Cross Entropy. It also peaks sooner than MSE.
- We can see the difference in performance between MSE and CE loss functions in the graph below. Cross Entropy performs better than MSE.





Question 9 (10 Marks)

Paste a link to your github code for this assignment

Example: https://github.com/suddu21/Deep-Learning-Assignmnent-1-DA24M023;

- 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

Answer

Github repo link: https://github.com/suddu21/Deep-Learning-Assignmnent-1-DA24M023;

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

- Since the MNIST dataset is very similar dimensionally to the Fashion MNIST dataset, we can apply our learnings on it and expect good results. Using similar config as the best model on the Fashion MNIST dataset should work very well for the MNIST dataset too.
- I would suggest the configurations below because it is necessary to experiment with different number of hidden layers and sizes of the hidden layers to see which suits the features of the images in MNIST dataset well. We have already seen some values of parameters like tanh and ReLU for activation working well, so we will stick to those.
- The 3 configurations I suggest are:

1. Config 1:

```
activation:"ReLU"
batch_size:128
beta:0.5
beta1:0.9
beta2:0.999
dataset:"mnist"
epochs:10
epsilon:0.00000001
hidden_layers:5
hidden_size:256
learning_rate:0.001
loss_fn:"cross_entropy"
momentum:0.9
optimizer:"nadam"
weight_decay:0
weight_init:"Xavier"
```

- Training Accuracy: 99.04%
- Validation Accuracy: 98.04%
- Test Accuracy: 97.4%

2. Config 2:

```
activation:"ReLU"
batch_size:128
beta:0.9
beta1:0.9
beta2:0.999
dataset:"mnist"
epochs:10
```

```
epsilon:0.00000001
hidden_layers:3
hidden_size:128
learning_rate:0.001
loss_fn:"cross_entropy"
momentum:0.9
optimizer:"adam"
weight_decay:0
weight_init:"random"
```

- Training Accuracy: 99.17%
- Validation Accuracy: 98.02%
- Test Accuracy: 97.42%

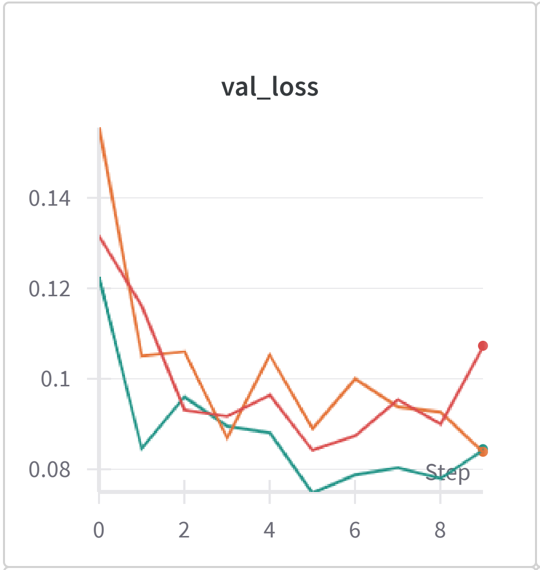
3. Config 3:

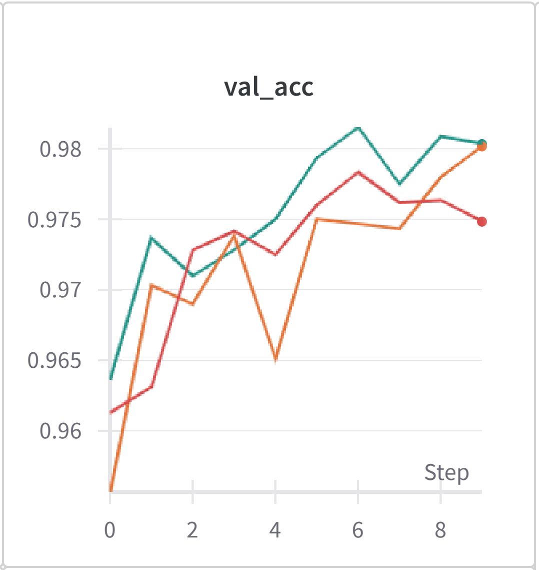
```
activation:"ReLU"
batch_size:64
beta:0.5
beta1:0.5
beta2:0.999
dataset:"mnist"
epochs:10
epsilon:0.00000001
hidden_layers:5
hidden_size:128
learning_rate:0.001
loss_fn:"cross_entropy"
momentum:0.9
optimizer:"adam"
```

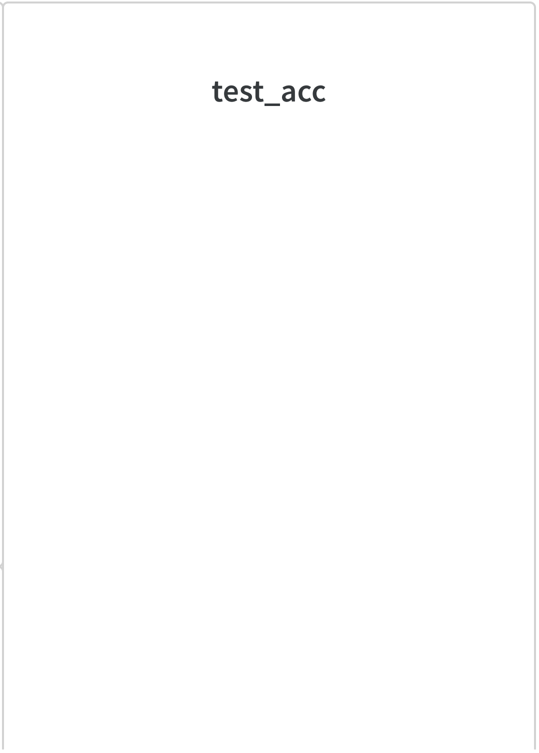
```
weight_decay:0
weight_init:"random"
```

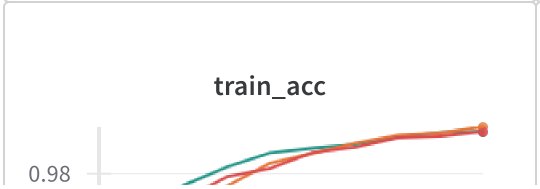
- Training Accuracy: 99.04%
- Validation Accuracy: 97.48%
- Test Accuracy: 97.46%

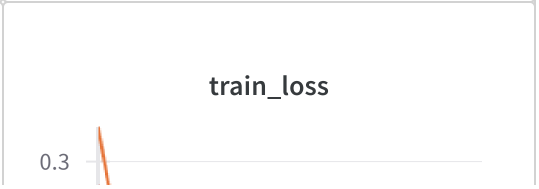
- As we can see, these results are even better than the results we got on the Fashion MNIST dataset, which is a testament to the fact that the 2 datasets are highly similar in nature and the model can generalise and learn from the image features well.
- Getting greater than 97% testing accuracy shows that the neural network is robust and efficient at classification tasks.

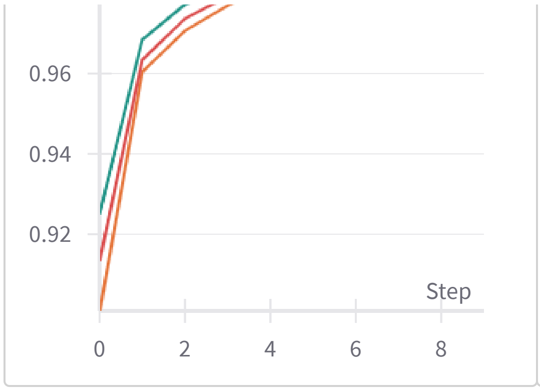


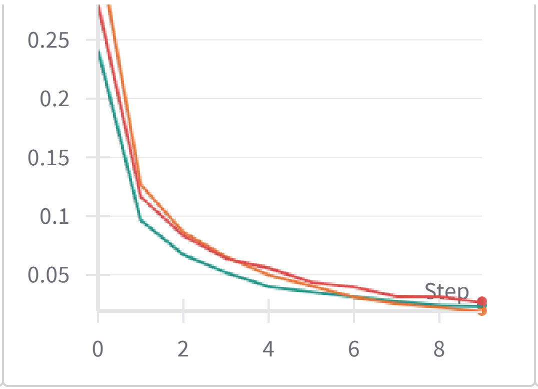












lr_0.001_a_ReLU_l_cross_entropy_wi_Xa
lr_0.001_a_ReLU_l_cross_entropy_wi_ra
lr_0.001_a_ReLU_l_cross_entropy_wi_ra
0.0
0.5

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

Name	Default Value	Description
`-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, Sudhanva Satish DA24M023, 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/da24m023-indian-institute-of-technology-madras/fashion-mnist-nn-sweep/reports/Sudhanva-Satish-DA24M023-DA6401-Assignment-1--VmlldzoxMTY5ODY1OA