CS6910 Assignment - 1

Name: Balakrishnan A

Roll No: CS20B012

WandB Report Link: Report (check report here as rendering in pdf is quite bad)

GitHub Repository Link: Repo

CS6910 - Assignment 1

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

Balakrishnan A

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.

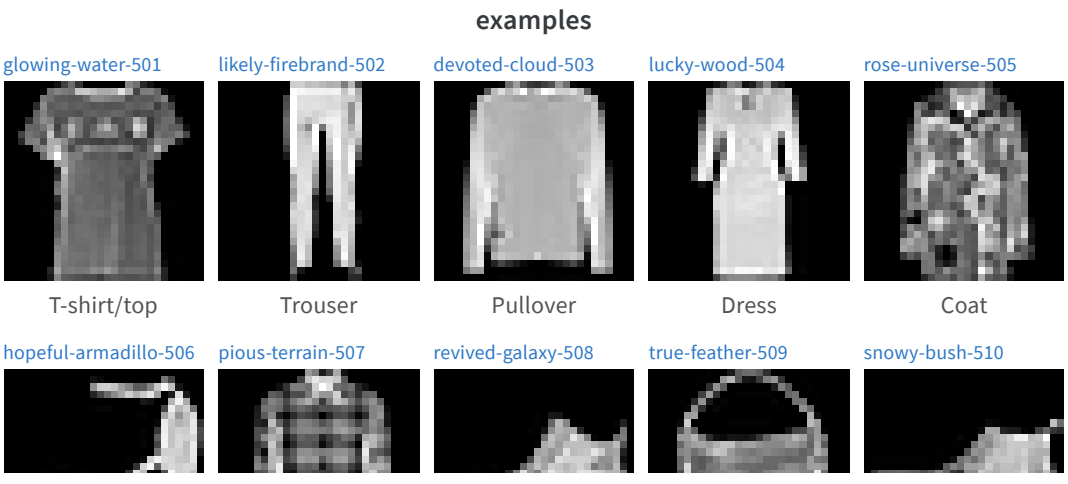
Problem Statement

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.

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.

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.

Random search strategy was chosen, since grid search would try all available combinations, taking too long to run until completion. The following hyperparameter choices were used:

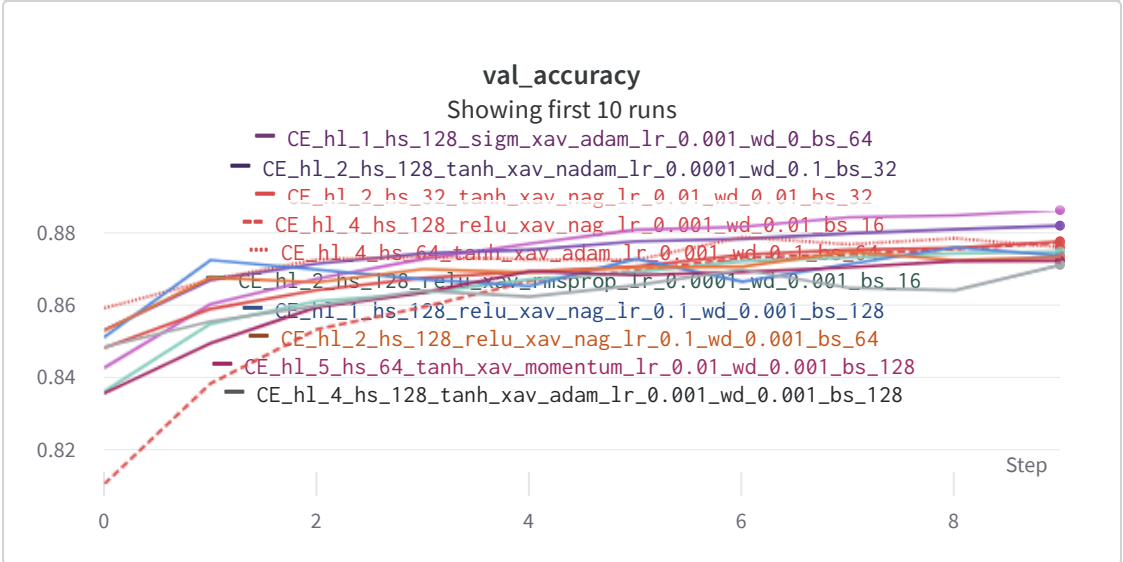
- number of hidden layers: 1, 2, 3, 4, 5
- size of every hidden layer: 16, 32, 64, 128
- activation functions: sigmoid, tanh, ReLU
- weight initialization: random, Xavier
- optimizer: sgd, momentum, nag, rmsprop, adam, nadam
- learning rate: 1e-1, 1e-2, 1e-3, 1e-4
- weight decay (L2 regularization): 0, 0.1, 0.01, 0.001

- batch size: 16, 32, 64, 128

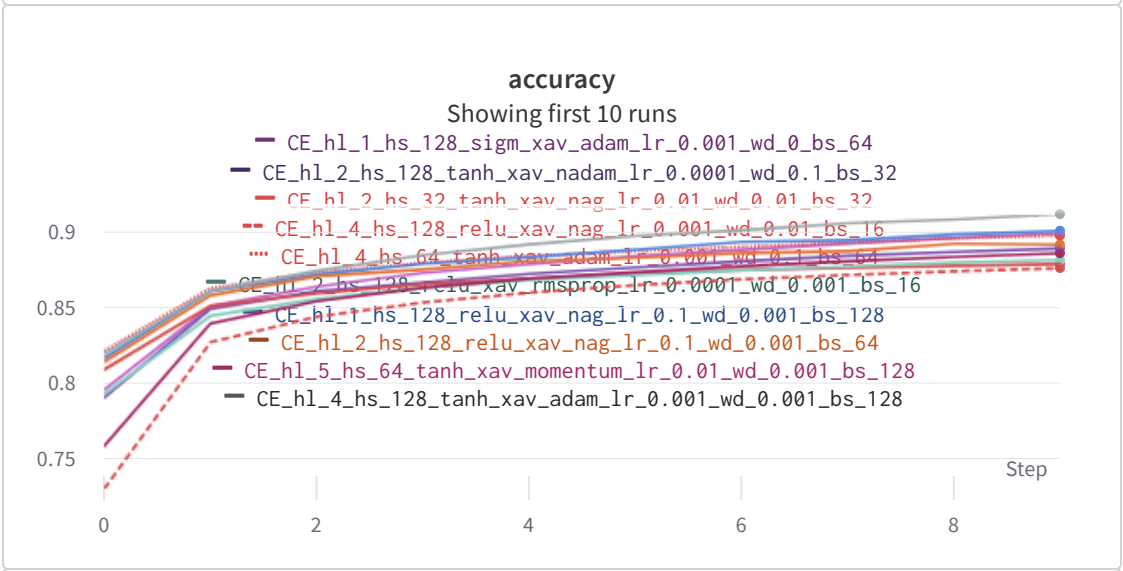
wandb will automatically generate the following plots. Paste these plots below using the "Add Panel to Report" feature. Make sure you use meaningful names for each sweep (e.g. hl_3_bs_16_ac_tanh to indicate that there were 3 hidden layers, batch size was 16 and activation function was ReLU) instead of using the default names (whole-sweep, kind-sweep) given by wandb.

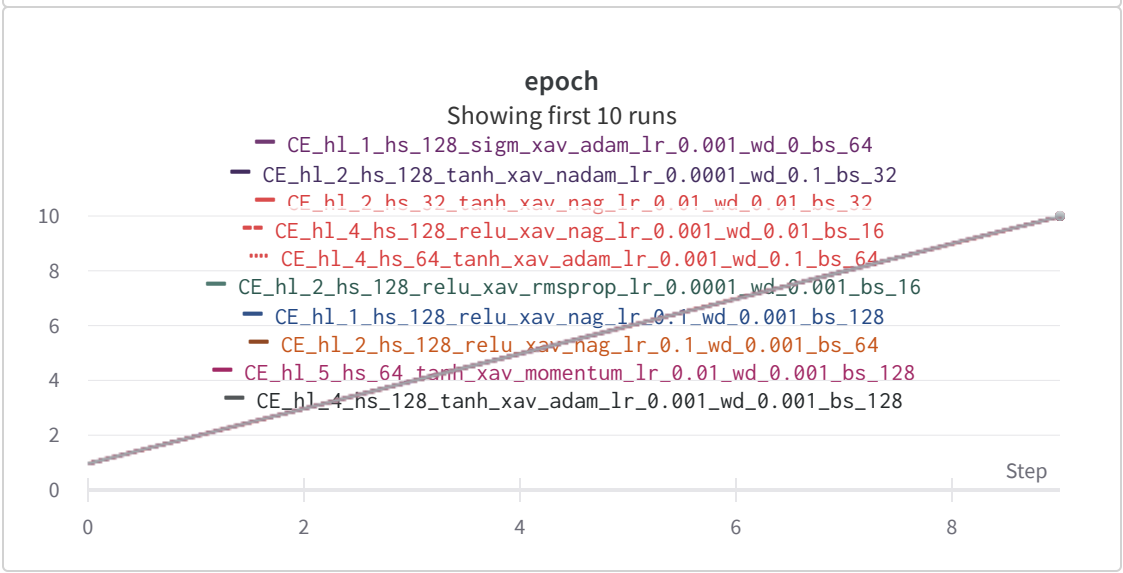
Naming convention: Each run is named as CE_hl_{number of hidden layers}_hs_{size of every hidden layer}_{activation function (first 4 letters)}_{weight initialization (first 3 letters)}_{optimizer}_lr_{learning rate}_wd_{weight decay}_bs_{batch size}

The top 10 runs (ranked by val_accuracy) are shown below.




loss
Showing first 10 runs
CE_hl_1_hs_128_sigm_xav_adam_lr_0.001_wd_0_bs_64
CE_hl_2_hs_128_tanh_xav_nadam_lr_0.0001_wd_0.1_bs_32
CE_hl_2_hs_32_tanh_xav_nag_lr_0.01_wd_0.01_bs_32
CE_hl_4_hs_128_relu_xav_nag_lr_0.001_wd_0.01_bs_16
CE_hl_4_hs_64_tanh_xav_adam_lr_0.001_wd_0.1_bs_64
CE_hl_2_hs_128_relu_xav_rmsprop_lr_0.0001_wd_0.001_bs_16
CE_hl_1_hs_128_relu_xav_nag_lr_0.1_wd_0.001_bs_128
CE_hl_2_hs_128_relu_xav_nag_lr_0.1_wd_0.001_bs_64
CE_hl_5_hs_64_tanh_xav_momentum_lr_0.01_wd_0.001_bs_128
CE_hl_4_hs_128_tanh_xav_adam_lr_0.001_wd_0.001_bs_128
0.6
0.4
0.2
0
Step
0
2
4
6
8


accuracy
Showing first 10 runs
CE_hl_1_hs_128_sigm_xav_adam_lr_0.001_wd_0_bs_64
CE_hl_2_hs_128_tanh_xav_nadam_lr_0.0001_wd_0.1_bs_32
CE_hl_2_hs_32_tanh_xav_nag_lr_0.01_wd_0.01_bs_32
CE_hl_4_hs_128_relu_xav_nag_lr_0.001_wd_0.01_bs_16
CE_hl_4_hs_64_tanh_xav_adam_lr_0.001_wd_0.1_bs_64
CE_hl_2_hs_128_relu_xav_rmsprop_lr_0.0001_wd_0.001_bs_16
CE_hl_1_hs_128_relu_xav_nag_lr_0.1_wd_0.001_bs_128
CE_hl_2_hs_128_relu_xav_nag_lr_0.1_wd_0.001_bs_64
CE_hl_5_hs_64_tanh_xav_momentum_lr_0.01_wd_0.001_bs_128
CE_hl_4_hs_128_tanh_xav_adam_lr_0.001_wd_0.001_bs_128
0.9
0.85
0.8
0.75
Step
0
2
4
6
8

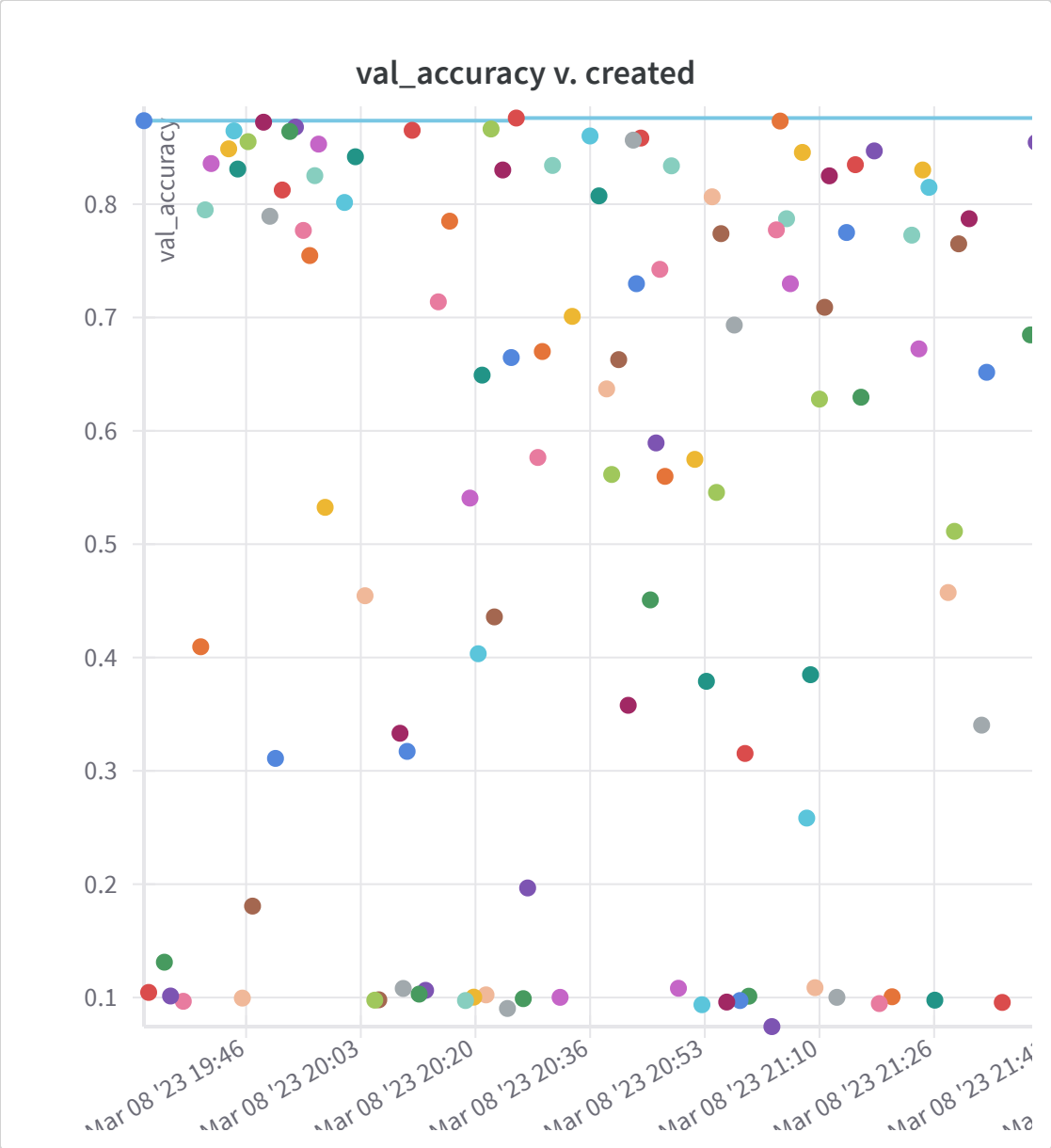

epoch
Showing first 10 runs
CE_hl_1_hs_128_sigm_xav_adam_lr_0.001_wd_0_bs_64
CE_hl_2_hs_128_tanh_xav_nadam_lr_0.0001_wd_0.1_bs_32
CE_hl_2_hs_32_tanh_xav_nag_lr_0.01_wd_0.01_bs_32
CE_hl_4_hs_128_relu_xav_nag_lr_0.001_wd_0.01_bs_16
CE_hl_4_hs_64_tanh_xav_adam_lr_0.001_wd_0.1_bs_64
CE_hl_2_hs_128_relu_xav_rmsprop_lr_0.0001_wd_0.001_bs_16
CE_hl_1_hs_128_relu_xav_nag_lr_0.1_wd_0.001_bs_128
CE_hl_2_hs_128_relu_xav_nag_lr_0.1_wd_0.001_bs_64
CE_hl_5_hs_64_tanh_xav_momentum_lr_0.01_wd_0.001_bs_128
CE_hl_4_hs_128_tanh_xav_adam_lr_0.001_wd_0.001_bs_128
10
8
6
4
2
0
Step
0
2
4
6
8

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 validation accuracy: 0.8863



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:

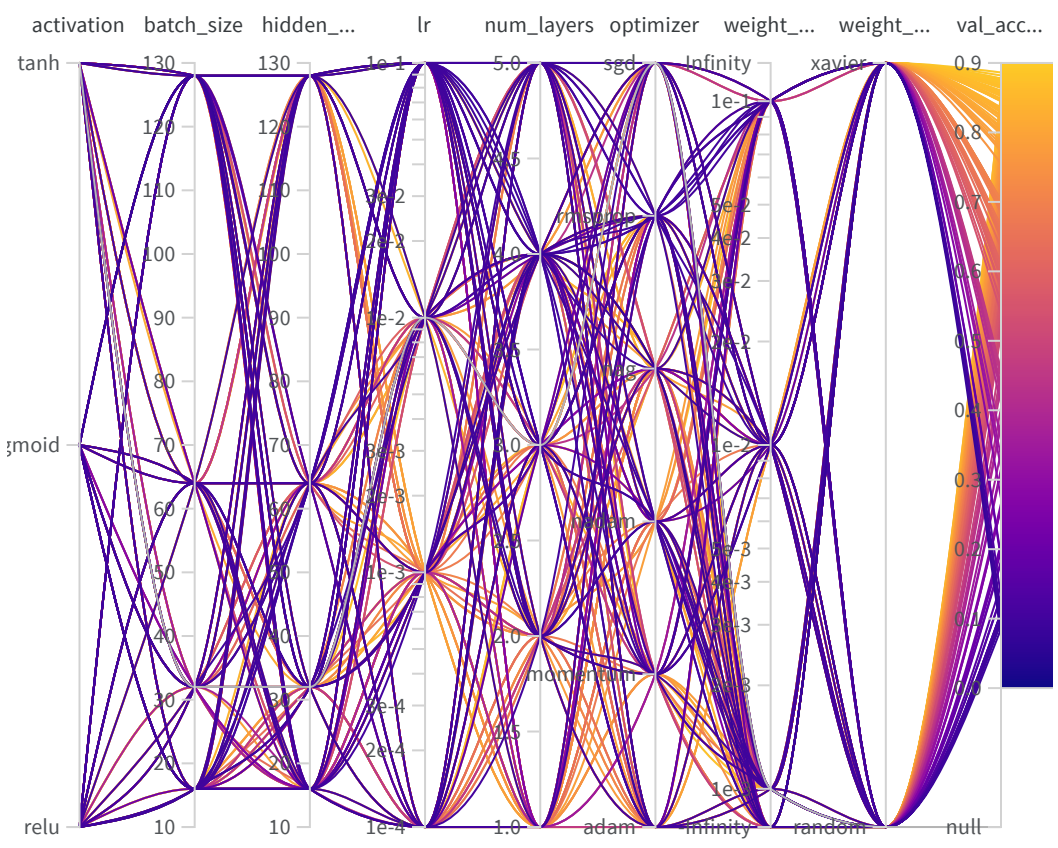
- Xavier initialization significantly outperforms random, reaching higher accuracies easier and more frequently, but also struggling to leave the 0.1 region more than random does. Random tends to fill the entire spectrum, whereas Xavier populates the low and high regions quite densely
- Lower learning rates tend to end up learning something, i.e., learning rates 1e-3, 1e-4 are quite sparse around 0.1 val_accuracy whereas 1e-2, 1e-1 (especially the latter) have quite a few values in that region. Perhaps those learning rates are too high to produce any actual learning and instead just move to different points with the same loss. Learning rate 1e-4 is a little less and tends to not reach "good" accuracies of > 0.8 as much, making 1e-3 just right
- SGD typically learns something but lacks the momentum to learn enough to achieve high scores. Momentum, NAdam and NAG

perform better, with NAG especially gaining an edge and performing really well, filling the high region quite densely. RMSProp performs in an all-or-nothing fashion for the most part, either performing poorly or performing quite well. Adam performs in the decent to high region, easily getting scores of > 0.5, and typically > 0.7. Overall, Adam and Momentum achieve best performance.

- Sigmoid activation appears to perform in an all-or-nothing fashion, scoring either well or poorly, with sparsity in between, in the "decent" region. Tanh activation usually performs well, reaching high accuracies very frequently. ReLU performs quite well, usually reaching pretty good accuracies
- Performance appears to grow with hidden size, with wider networks performing better, especially visible in 128 hidden size. An interplay with the number of layers also must occur as deep networks that are also wide should overfit to the data
- This interplay becomes apparent here as higher the number of layers, the more unstable learning seems to be. 1, 2, 3 layers perform quite well, usually learning something and leaving the random guessing (~0.1) region, whereas 4, 5 layers may remain in the random region more, but perform quite well, reaching higher accuracies easier when leaving the low region
- Weight decay of 0 performs the best, perhaps the data's high dimensionality provides implicit regularization, eliminating the need for explicit regularization. Weight decay of 0.1 performs quite poorly, struggling to reach 0.8 accuracy as it is regularized quite heavily. Weight decays of 0.01 and 0.001 are decent, not very far behind 0 decay
- Train and validation accuracies agree well for the most part, furthering the idea that the dataset is well regularized on its own

Ways to approach 95% accuracy:

- The best configuration obtained by random search obtains 88.63% accuracy on validation data and has not saturated yet, providing a possibility of growing to at least 90%
- Extra regularization such as dataset augmentation and noise injection could give it the boost required to attain close to 95%



Parameter importance with respect to val_accuracy

Search | Parameters | 1-10 of 16

Config parameter	Importance	Correlation
num_layers		
lr		
optimizer.value_sgd		
optimizer.value_nag		
activation.value_sig…		
weight_decay		
hidden_size		
optimizer.value_nadam		

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)

Test accuracy: 0.8795

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.

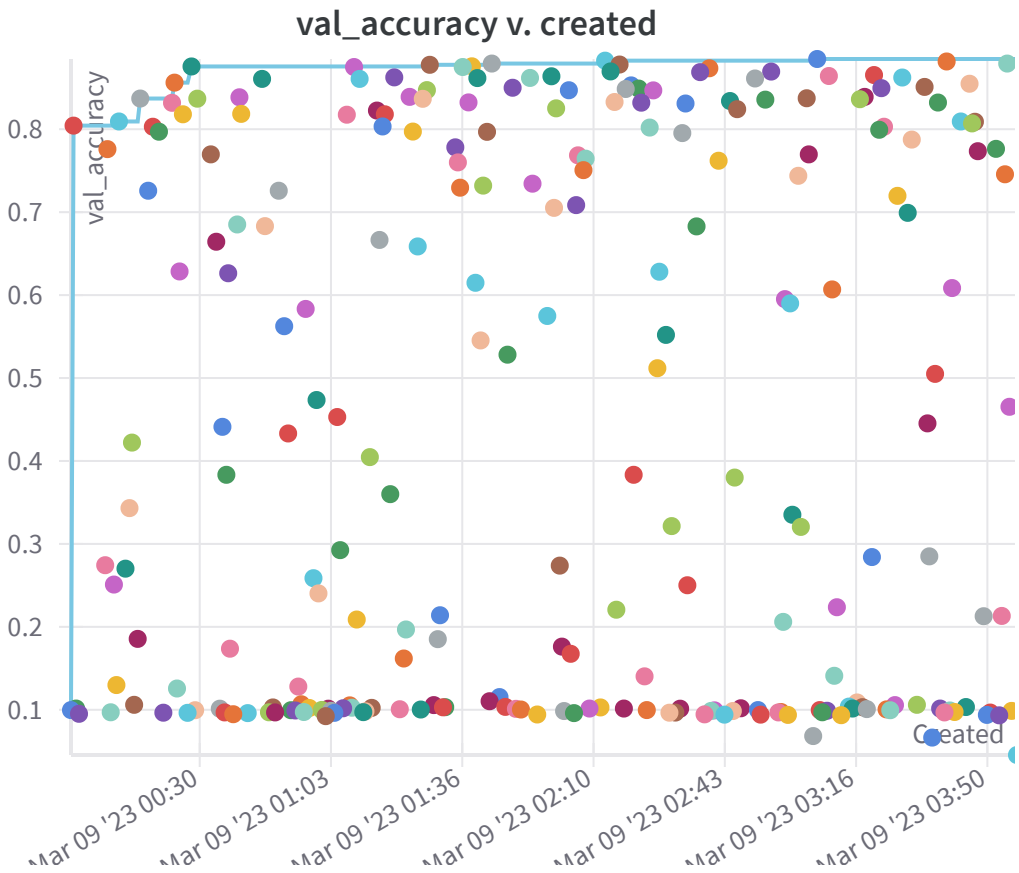
Insights:

- The most significant difference between the losses is the gradient of the loss wrt the produced outputs. For cross-entropy, it can grow unboundedly large when the prediction is poor and is never less than 1, even when the prediction is amazing (for the true class). Whereas for MSE, the partial derivatives can never exceed 1, no matter how poor the prediction is. This leads to a more acute difficulty in leaving low accuracies towards lower losses, as its gradients do not allow it to change significantly, leading to larger density in the low accuracy region
- Performances in sigmoid and tanh are distributed similar to distribution in cross entropy, but it performs surprisingly poorly on ReLU activation. Perhaps the combination of small loss derivatives along with ReLU's 0 gradient possibilities when not "activated" allow for poor learning, as it struggles to leave the 0.1 region
- Mean squared error performs especially poorly for deeper networks, as it populates the low region very densely
- It is also highly sensitive to weight decay, with the effect being very drastic here leading to weight decays of 0 and 0.001 performing a lot better than 0.01 and 0.1
- Cross entropy loss's specialization for probability distributions gives it an edge over mean squared error, filling the high region denser than the decent region, as opposed to mean squared error which does the reverse

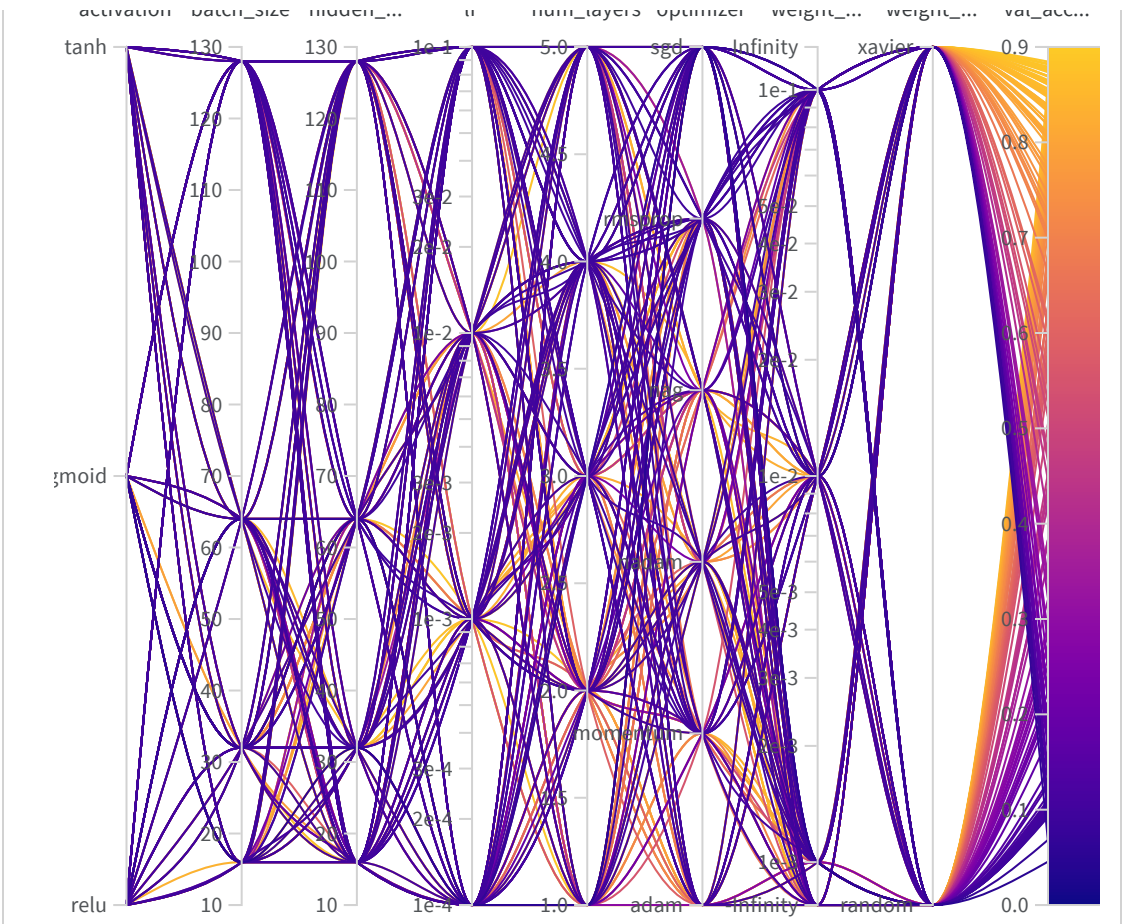
Parameter importance with respect to val_accuracy

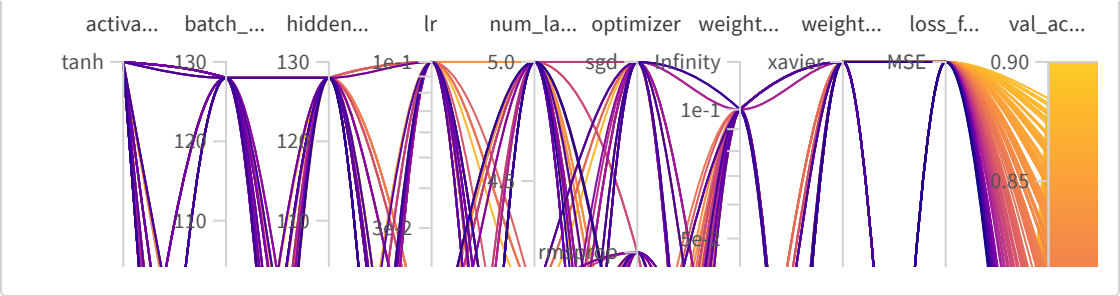
Search | Parameters | 1-10 of 16

Config parameter	Importance	Correlation
weight_decay		
lr		
optimizer.value_sgd		
num_layers		
optimizer.value_mo…		
optimizer.value_nag		
batch_size		



activation | batch_size | hidden | lr | num_layers | optimizer | weight | weight | val_acc


activation
batch_size
hidden_...
lr
num_layers
optimizer
weight_...
weight_...
val_acc...
tanh
sigmoid
relu
sgd
rmsprop
adam
momentum
Infinity
xavier
random



Question 9 (10 Marks)

Paste a link to your github code for this assignment

https://github.com/Bala-A87/CS6910-A1

- 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.

Configurations tried:

Fixed hyperparameters:

- Weight initialization: Xavier
- Batch size: 64
- Epochs: 10
- Learning rate: 1e-3

Remaining parameters (some of the best configurations from random search):

1. Hidden layers: 1, hidden size: 128, activation: Sigmoid, optimizer: Adam, weight decay: 0, **accuracies: 0.9836 (train), 0.9760 (val)**
2. Hidden layers: 4, hidden size: 128, activation: ReLU, optimizer: NAG, weight decay: 1e-2, **accuracies: 0.9404 (train), 0.9420 (val)**
3. Hidden layers: 4, hidden size: 64, activation: Tanh, optimizer: Adam, weight decay: 0.1, **accuracies: 0.9807 (train), 0.9641 (val)**

These were some of the top 5 configurations determined by random search. The 2nd best and 3rd best were removed as one involved NAdam with large weight decay, which was not promising, and another was underparametrized as it only used 32 units per layer.

Test accuracy on the 1st configuration (with best val acc): **0.9736**

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.

Name	Default Value	Description
`-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, Balakrishnan A, 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/cs20b012/CS6910-A1/reports/CS6910-Assignment-1--VmlldzozNTQ0ODA0?accessToken=oeeeab3buqq86mnpxp9ygm56t2x0f9zbx6ne6o11virm3wouk36zdvz7bhrtc7ea