# Report: Assignment 1

Report on training a FNN using gradient descent with backpropagation and keeping track of experiments using wandb.ai

Rigved Sah cs20m053

In this assignment we implemented and trained a FNN using Gradient Descent and its variants with backpropagation. This network was 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 was trained to classify the image into 1 of 10 classes.

Below panel contains sample image for each class along with label present in the Fashion-MNIST dataset.

We implemented a FNN which takes images from the fashion-mnist data as input and outputs a probability distribution over the 10 classes. The code was written from scratch without using any libraries such as Keras, PyTorch or Tensorflow. For matrix and vector operations NumPy was used.

We used **Softmax** as output function and **Cross-Entropy** as loss function.

The code is written in a very flexible manner that allows changing any hyperparameters such as optimizers, activation functions, number & size of hidden layers, batch size, learning rate etc. at just one place.

## Optimization Functions Implemented

We implemented the backpropagation algorithm with support for the following optimization functions,

- Stochastic Gradient Descent (SGD)
- Momentum Gradient Descent (MGD)
- Nesterov Accelerated Gradient Descent (NAG)
- RMSPROP
- ADAM
- NADAM

Addition of new algorithm is very easy to integrate into the code.

# Hyperparameters Tuning

We used the standard train/test split of fashion-mnist dataset and kept 10% of training data aside as validation dataset for hyperparameter tuning.

We shuffled the training data to randomly picked validation dataset and also made sure the distribution of each image in validation dataset is uniform so as to prevent bias towards any particular class.

The summary for different datasets is as:

| | |
|---|---|
| Training Data: | 54,000 images of size 28 x 28 pixels |
| Validation Data: | 6,000 images of size 28 x 28 pixels (600 Images of each class) |
| Test Data: | 10,000 images of size 28 x 28 pixels |

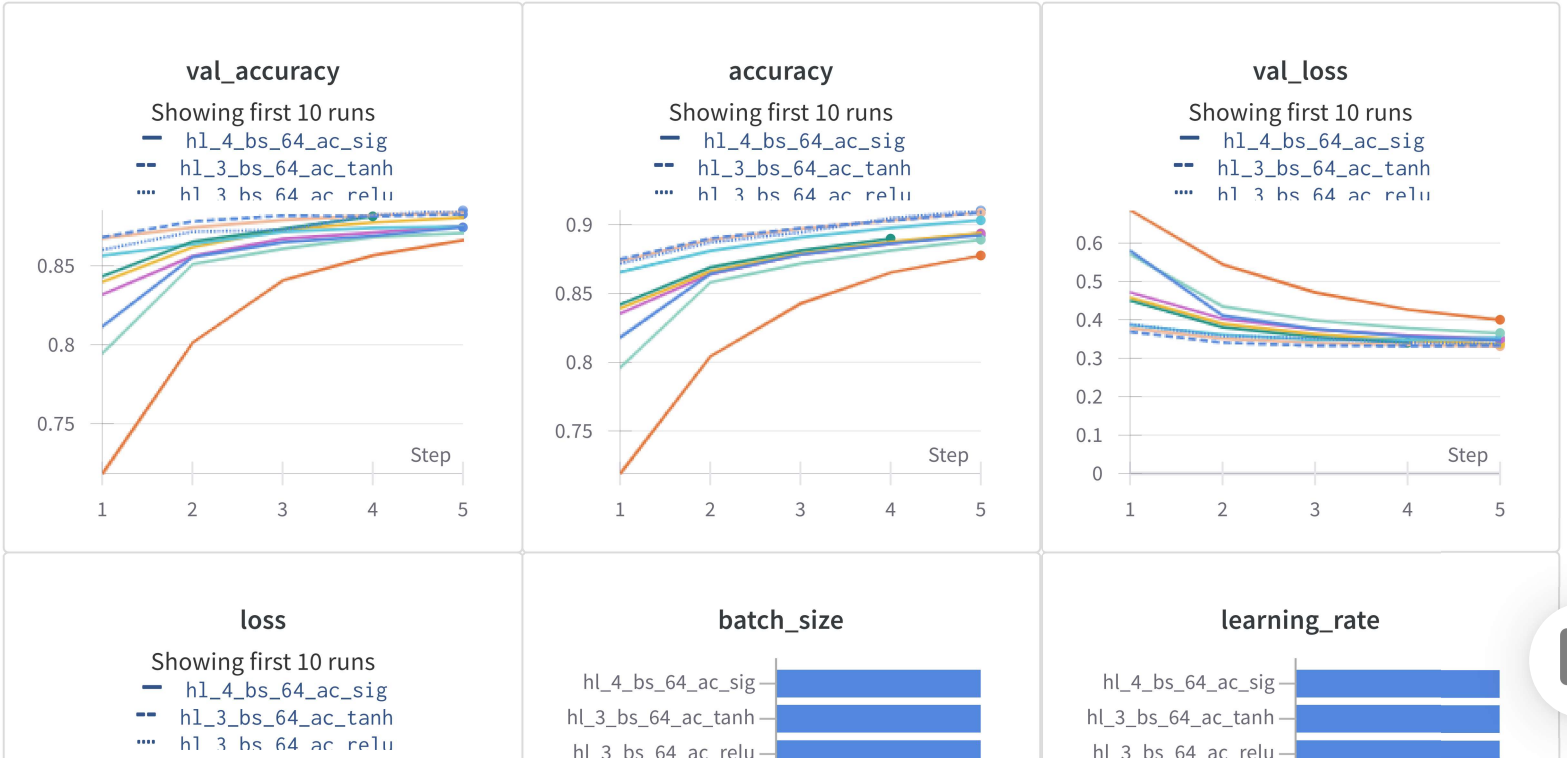
The hyperparameters considered for sweep as as:

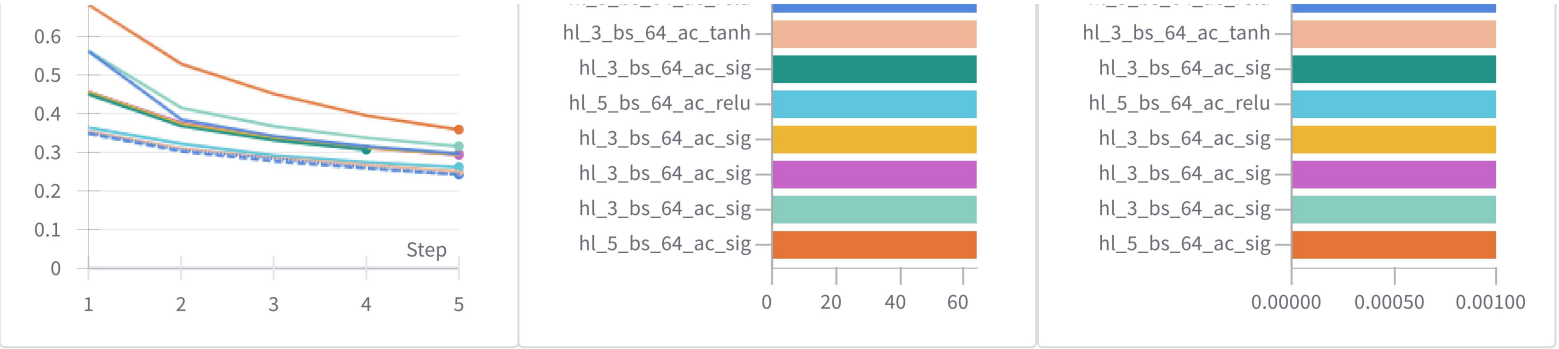
- Number of Epochs: **5**
- Number of Hidden Layers: **3, 4 , 5**
- Size of Hidden Layers: **64, 128**
- Weight Decay (L2 Regularization) ( $\alpha$ ): **0, 0.0005, 0.5**
- Learning Rate ( $\eta$ ): **0.001, 0.0001**
- Batch Size: **16, 32, 64**
- Weight Initialization Strategy: **Random, Xavier**

- Activation Function: **Sigmoid, Tanh, ReLU**
- Optimizer: **SGD, MGD, NAG, RMSPROP, ADAM, NADAM**

As we can see above, we will have exponential number of hyperparameters combinations therefore, to perform hyperparameter tuning efficiently we used,

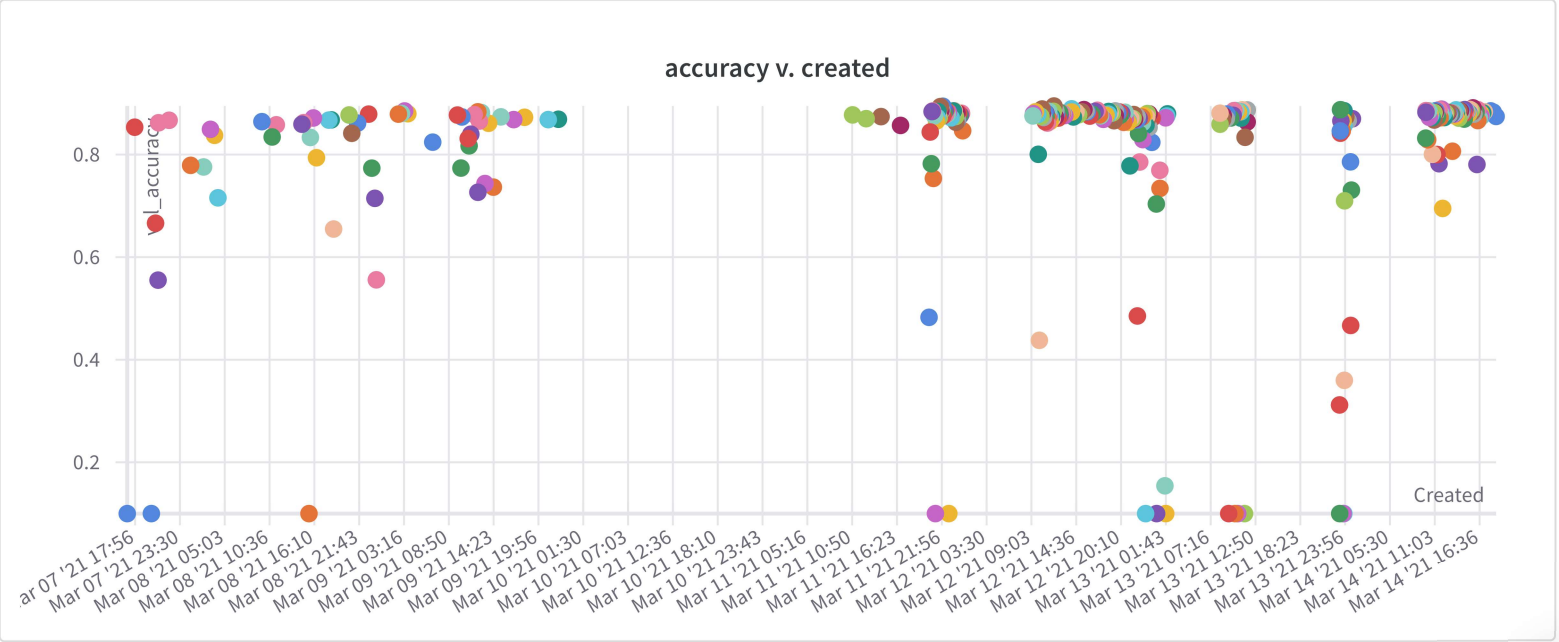
1. Wandb's **Bayes Optimization** search strategy which uses gaussian function to model the function and then chooses parameter to optimize probability of improvement. Here we are optimizing (maximizing) the **validation accuracy.**
2. Wandb's **Hyperband** stopping criteria. It speeds up hyperparameter search by stopping poorly performing runs



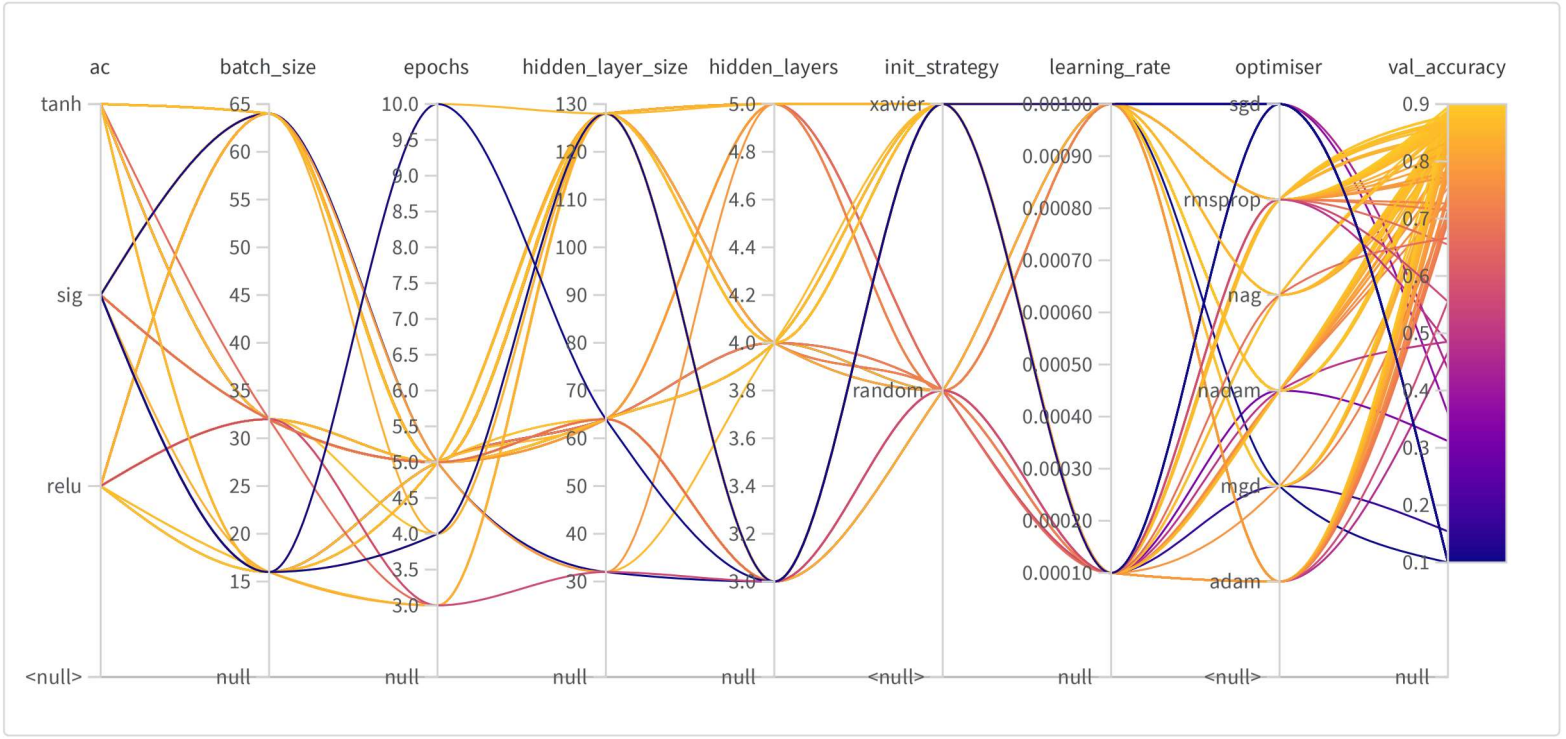



# Validation Accuracy across Model

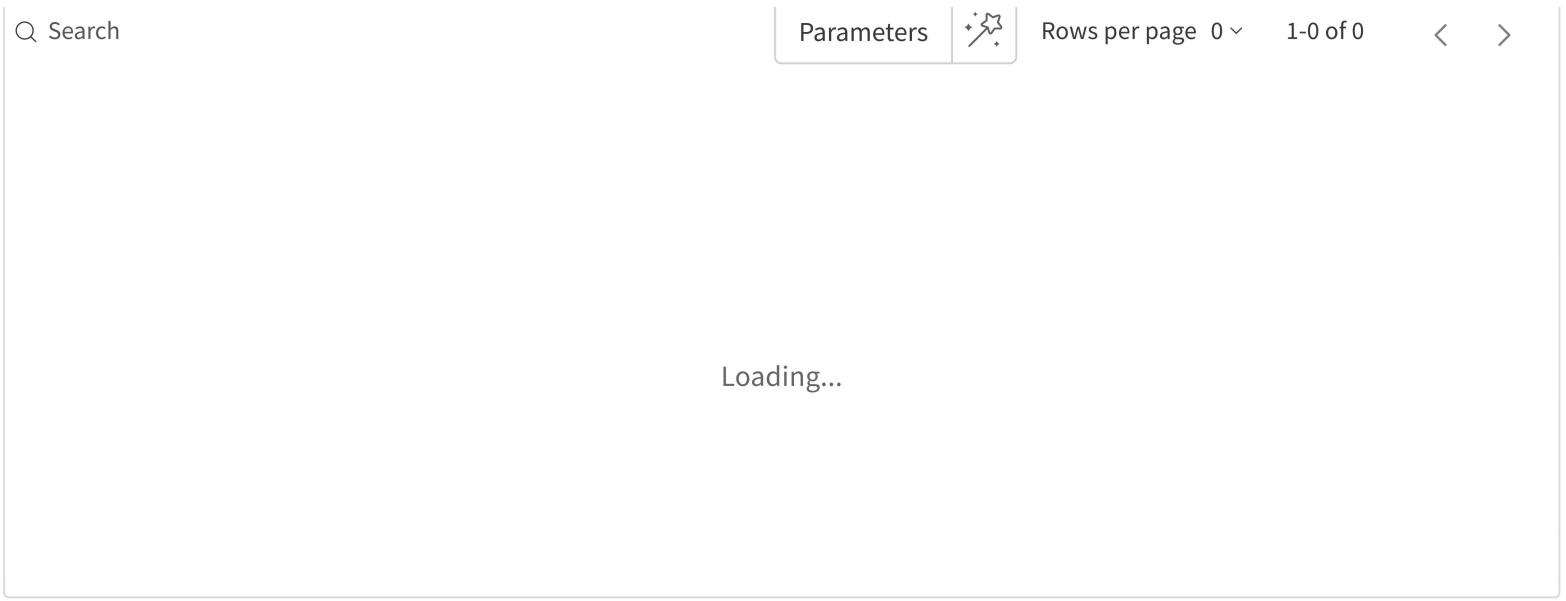
The validation accuracy for different models is as:



# Parallel co-ordinates plot & Inferences




Parameter importance with respect to val_accuracy




## Configurations:

There are many configurations that works well, one such configurations is,

| | |
|---|---|
| Val Accuracy: | **0.891** |
| Optimizer: | **NADAM** |
| Number of Epochs: | **5** |
| Number of Hidden Layers: | **3** |
| Size of Hidden Layers: | **128** |
| Learning Rate ( $\eta$ ): | **0.001** |
| Batch Size: | 64 |
| Weight Initialization Strategy: | **Xavier** |
| Activation Function: | **ReLU** |

One of the configuration that did not worked well is

| | |
|---|---|
| Val Accuracy: | 0.3598 |
| Optimizer: | **SGD** |
| Number of Epochs: | **5** |
| Number of Hidden Layers: | **3** |
| Size of Hidden Layers: | **64** |
| Learning Rate ( $\eta$ ): | **0.001** |
| Batch Size: | 6**4** |
| Weight Initialization Strategy: | **Random** |
| Activation Function: | **Tanh** |

## Observations & Inferences:

- SGD has lowest while NADAM has highest validation accuracy than other optimizers. It may be because SGD is greedy in updating weights and biases, thus requiring large number of epochs ( > 10 ) to train properly.
- Learning rate 0.001 seems to work well than 0.0001 for most configurations & optimizers.
- Tanh works better than Sig (Logistic) for most configurations & optimizers.
- Xavier works better than Random for most configurations and optimizers. It may be because large random weights saturates tanh and sigmoid making gradient zero (Vanishing Gradient).
- Most of the configurations give validation accuracy in the range [0.75, 0.89].

In this assignment we have used weight decay for only VGD, SGD, MGD & NAG. To achieve close to **95%** accuracy we can use some regularization techniques such as **L2 Regularization** or **Dropout** in optimizers such as RMSPROP, ADAM, NADAM to prevent overfitting and performing well on validation as well as test dataset.

# Confusion Matrix

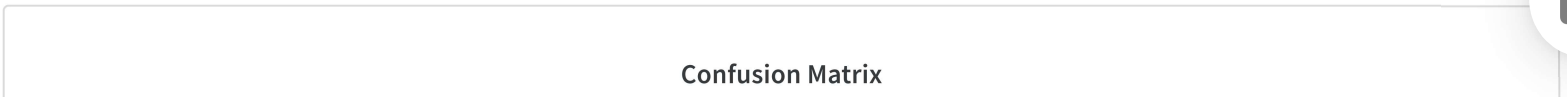
We trained model on the following configuration,

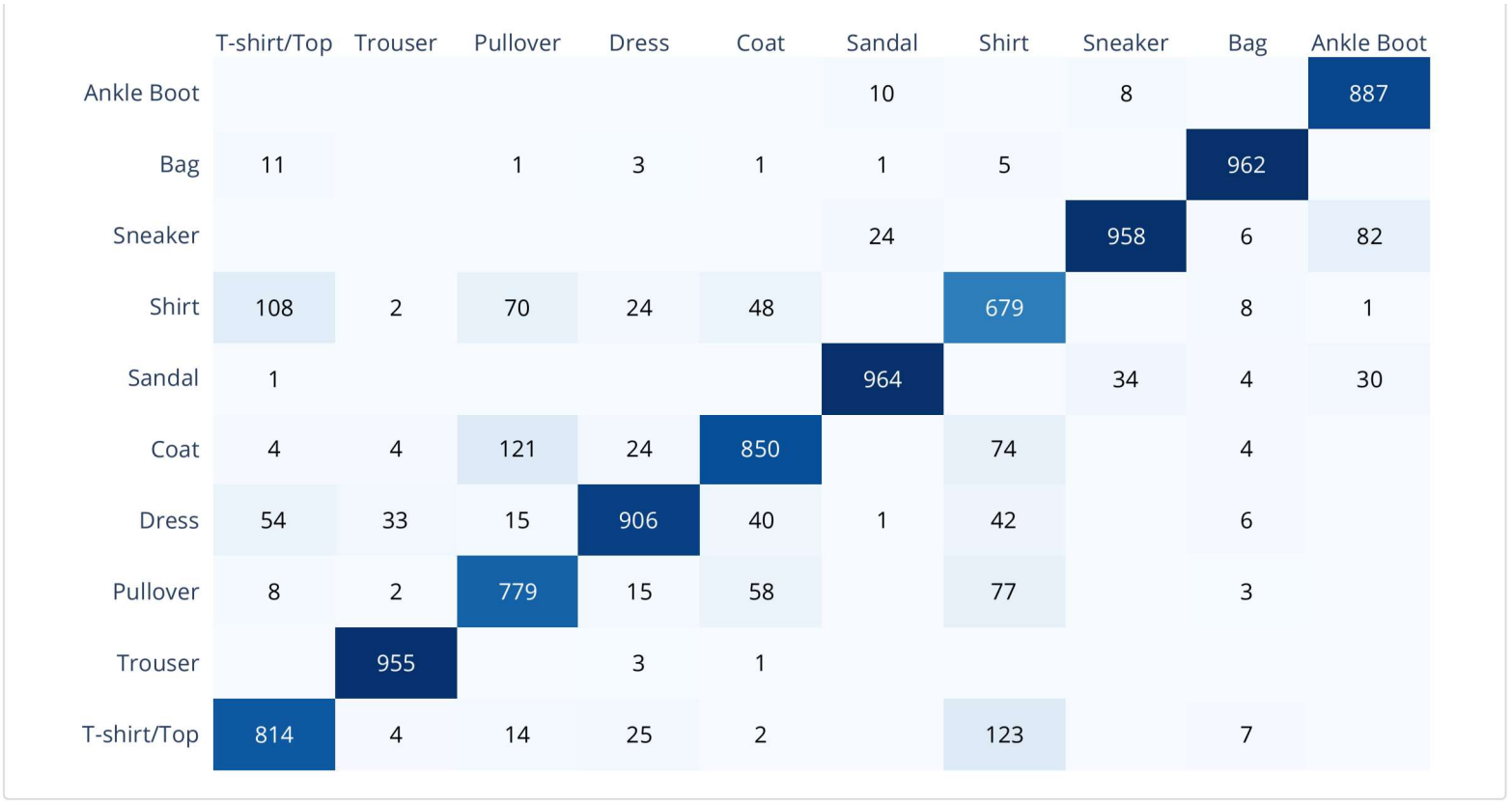
| | |
|---|---|
| Optimizer: | **NADAM** |
| Number of Epochs: | **5** |
| Number of Hidden Layers: | **3** |
| Size of Hidden Layers: | **128** |
| Learning Rate ( $\eta$ ): | **0.001** |
| Batch Size: | 6**4** |
| Weight Initialization Strategy: | **Xavier** |
| Activation Function: | **ReLU** |

Statistics for model are as,

| Training Accuracy | Validation Accuracy | Test Accuracy (Cross Entropy) |
|---|---|---|

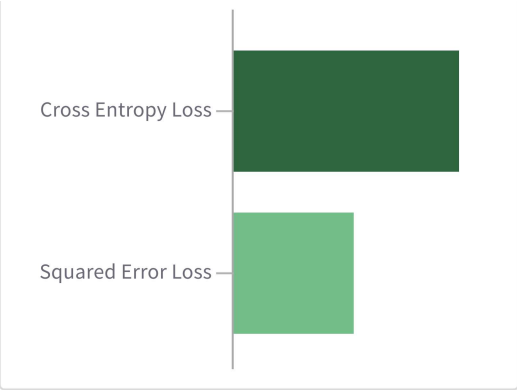

Here **x-axis** corresponds to **True Labels** and **y-axis** corresponds to **Predicted Labels.**



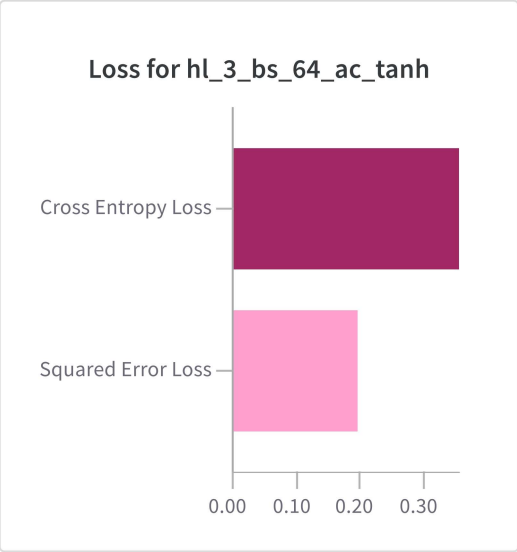



# Cross Entropy vs Squared Error

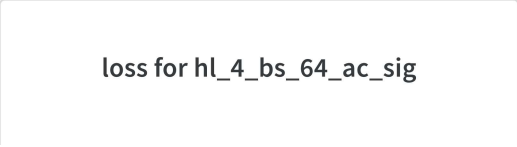
**Loss for hl_3_bs_64_ac_relu**

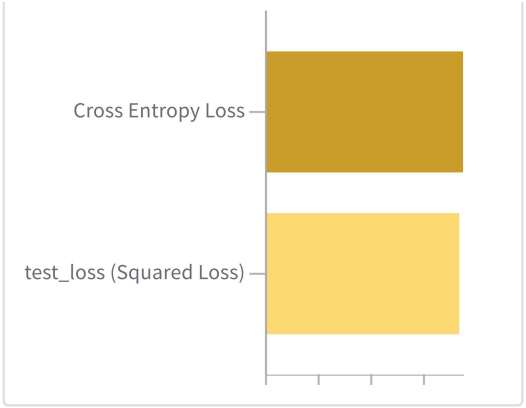



## Error Comparison for **hl_3_bs_64_ac_tanh (ADAM Optimizer)**




## Error Comparison for **hl_4_bs_64_ac_sig (RMSPROP)**






Cross Entropy Loss is better than Squared Error Loss when output of classification is a probability distribution as cross entropy loss tells how much the predicted class differs from the true class whereas squared error tells how much each predicted class differs from other corresponding true class. Therefore cross entropy is much more refined than squared error loss.

Also CE loss is more  robust than squared error loss as it decreases only when predicted probability of true class comes close to 1 whereas in SE loss also depends on probabilities of other classes.

## Recommendations for MNIST Dataset

The 3 hyperparameter configurations that can be used in MNIST dataset along with their training, validation & testing accuracies are,

| | |
|---|---|
| Optimizer: | **NADAM** |
| Number of Epochs: | **5** |
| Number of Hidden Layers: | **3** |
| Size of Hidden Layers: | **128** |

| | |
|---|---|
| Learning Rate ( $\eta$ ): | **0.001** |
| Batch Size: | 64 |
| Weight Initialization Strategy: | **Xavier** |
| Activation Function: | **ReLU** |
| | |
| Training Accuracy: | **0.9105** |
| Validation Accuracy: | **0.8852** |
| Testing Accuracy: | **0.8754** |

| | |
|---|---|
| Optimizer: | **ADAM** |
| Number of Epochs: | **5** |
| Number of Hidden Layers: | **3** |
| Size of Hidden Layers: | **128** |
| Learning Rate ( $\eta$ ): | **0.001** |
| Batch Size: | 64 |
| Weight Initialization Strategy: | **Xavier** |
| Activation Function: | **Tanh** |
| | |
| Training Accuracy: | **0.9088** |
| Validation Accuracy: | **0.8827** |
| Testing Accuracy: | **0.8721** |

| | |
|---|---|
| Optimizer: | **RMSPROP** |
| Number of Epochs: | **5** |
| Number of Hidden Layers: | **4** |
| Size of Hidden Layers: | **128** |
| Learning Rate ( $\eta$ ): | **0.001** |
| Batch Size: | 64 |

| | |
|---|---|
| Weight Initialization Strategy: | **Xavier** |
| Activation Function: | **Sig (Logistic)** |
| | |
| Training Accuracy: | **0.8924** |
| Validation Accuracy: | **0.8743** |
| Testing Accuracy: | **0.8645** |

# Logistics

**Link to Github Repository**

Github repository contains a README.md file with clear instructions on training and evaluating the model.

**Members:**

- Rigved Sah, CS20M053
- Sumit Negi, CS20M067

Number of Late Day(s): 1

# Self Declaration

Contributions of the two team members are as,

**CS20M053 (50% Contribution)**

- Implemented SGD, ADAM & NADAM
- Set up sweep in wandb
- Plotted the Confusion Matrix

**CS20M067 (50% Contribution)**

- Implemented MGD, NAG & RMSPROP
- Set up sweep in wandb
- Implemented Squared Error Loss

We, **RIGVED SAH, CS20M053** and **SUMIT NEGI, CS20M067**, swear on our honor that the above declaration is correct.

Created with ♡ on Weights & Biases.

https://wandb.ai/rigvedsah/Assignment%201/reports/Report-Assignment-1--Vmlldzo1MTYxMTU