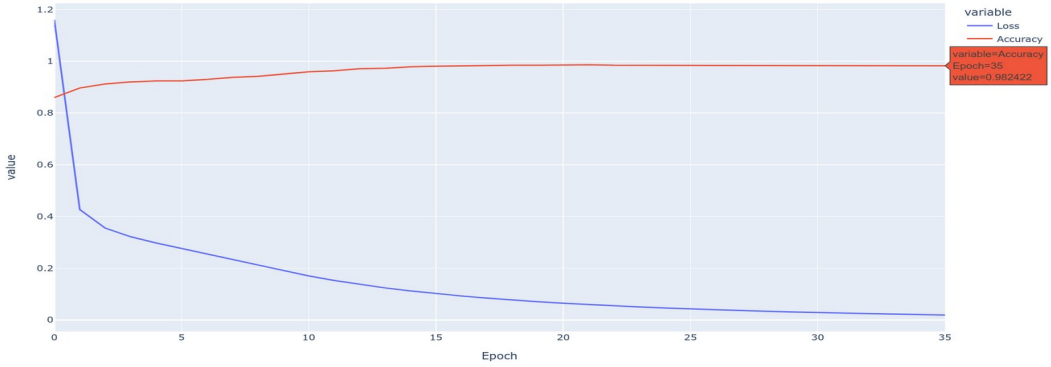
STUDENT: TSWLUN002
ASSIGMENT: 1 ML

Layers

- for number of layers, keep optimizer = SGD, learning rate =0.02, batch size =512.
- In the hidden layers has PReLu() and output layer have LogSoftMax(dim=1)
- factor of early stop =15 ,
  - early stop – means if validate with 15 epochs but there is no change then stop , store current accuracy as best accuracy for model
  - It is also constant factor
- So actual investigation will be the depth(number of layers) of neural network and width(number of nodes in each layer) of the neural network
  - investigation on Depth and width of neural network
  - Input Layer - It has 784 inputs because model uses pictures of 28 X 28
  - Hidden layer

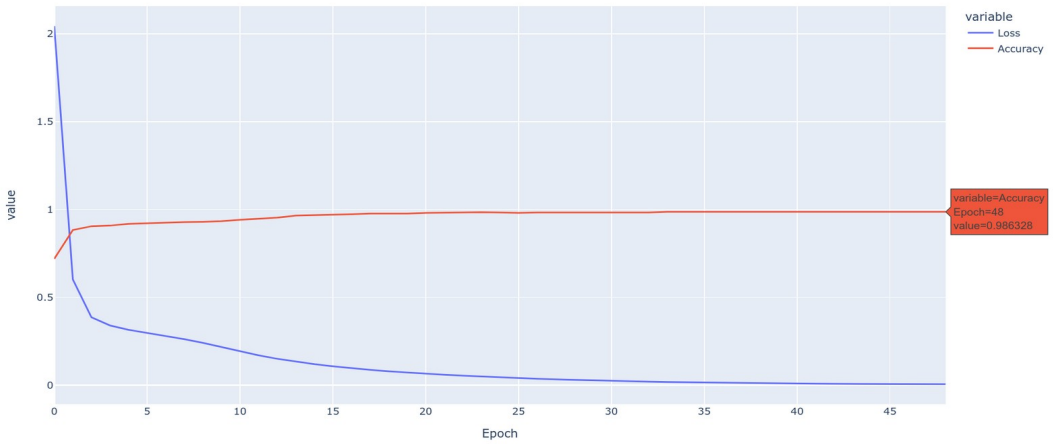
- **one hidden layer network**

```
NeuralNetwork(
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): PReLU(num_parameters=1)
    (2): Linear(in_features=512, out_features=10, bias=True)
    (3): LogSoftmax(dim=1)
  )
)
```



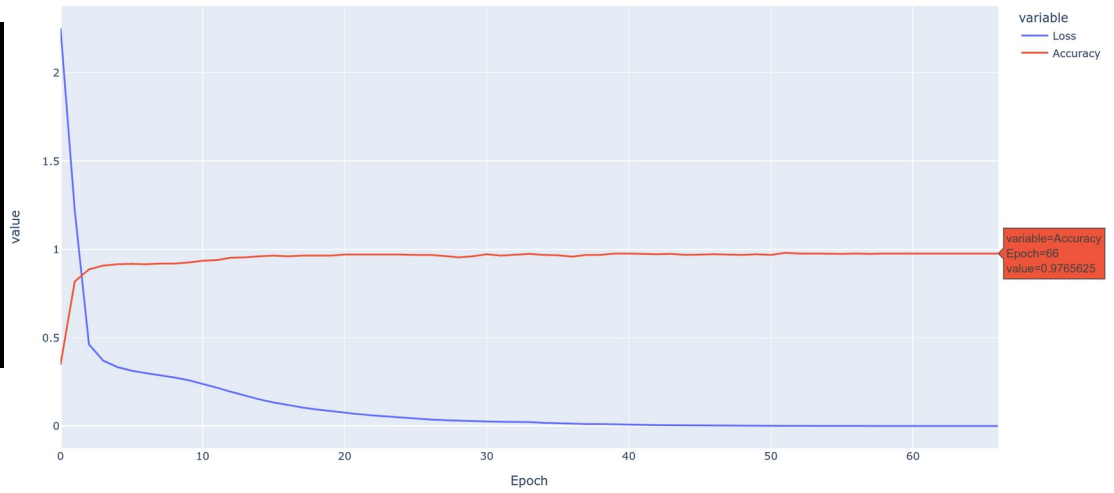
- **2 hidden layers**

```
NeuralNetwork(
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): PReLU(num_parameters=1)
    (2): Linear(in_features=512, out_features=64, bias=True)
    (3): PReLU(num_parameters=1)
    (4): Linear(in_features=64, out_features=10, bias=True)
    (5): PReLU(num_parameters=1)
    (6): LogSoftmax(dim=1)
  )
)
```



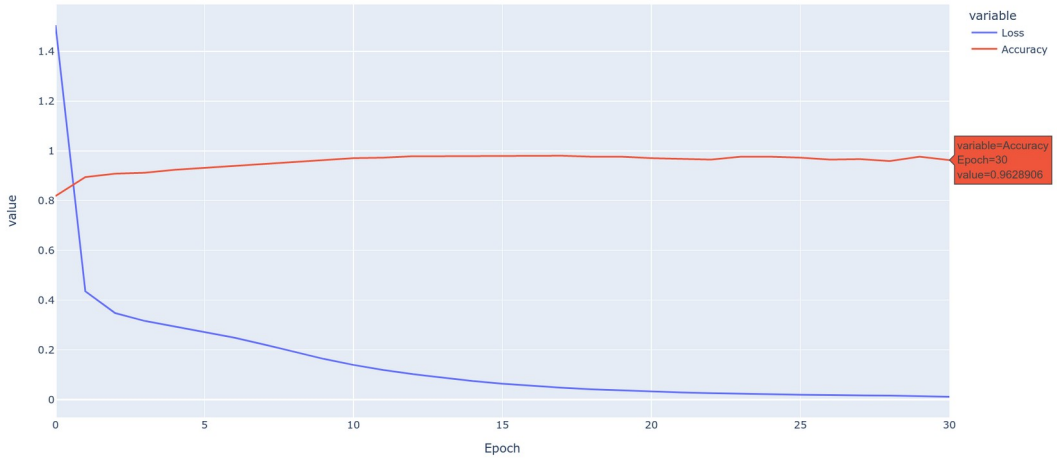
- **3 hidden layers**

```
NeuralNetwork(
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): PReLU(num_parameters=1)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): PReLU(num_parameters=1)
    (4): Linear(in_features=256, out_features=64, bias=True)
    (5): PReLU(num_parameters=1)
    (6): Linear(in_features=64, out_features=10, bias=True)
    (7): PReLU(num_parameters=1)
    (8): LogSoftmax(dim=1)
  )
)
```



- **4 Hidden Layers**

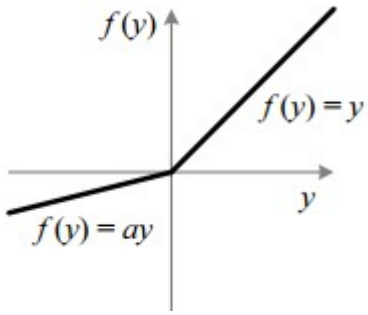
```
NeuralNetwork(
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): PReLU(num_parameters=1)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): PReLU(num_parameters=1)
    (4): Linear(in_features=256, out_features=128, bias=True)
    (5): PReLU(num_parameters=1)
    (6): Linear(in_features=128, out_features=64, bias=True)
    (7): PReLU(num_parameters=1)
    (8): Linear(in_features=64, out_features=10, bias=True)
    (9): PReLU(num_parameters=1)
    (10): LogSoftmax(dim=1)
  )
)
```



- From the above diagrams we observe that from 1 hidden layer to 2 hidden layers the Accuracy increased from 98.2422% to 98.6328% , this was because the notion of more layers , accuracy of network increases predict
- From 2 hidden layer to 4 hidden layer we decrease in accuracy, this cause by over learning of training which called Over-fitting. “Over-fitting is Neural networks are flexible enough that they can adjust their parameters to fit the training data so precisely that they no longer generalize to data from outside the training set (for example, the test data)”[1]
- Also from the figure as increased depth(number of layer) , width(number of units in each layer) were decrease by factor 2
- As number of units decreases , accuracy was decreasing too so with large width, accuracy of network increases.
- number because of the early stop used when there is no long significant learning , we can see from loss graph of 3 and 4 hidden layer , it become constant value very fast hence less epochs
- So in my topology i have **2 hidden layers** with 512 neuron in first hidden layer and 64 neuron second hidden layer.

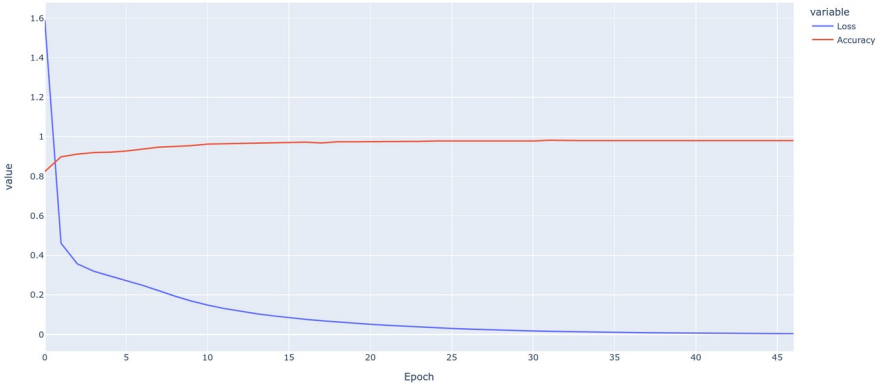
Activation functions

- In my final topology , i used PReLu for hidden layer ,LogSoftMax (dim=1) for output layer
- Why PReLu ?
  - PReLu function
  - **f(y) = { y, y>=0**
    αy , y<0



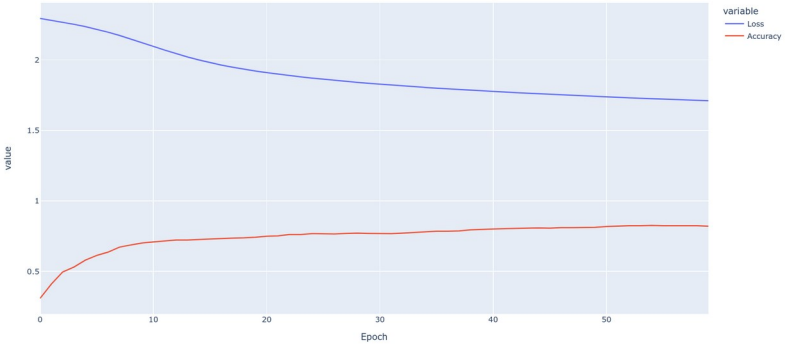
  - so PReLu unlike ReLu it able to normalise data around zero which means its able to use negative weight and positive weight .
    - To note when **α = 0** , PReLu became ReLu
    - When α close to zero like 0.01 , it become leaky ReLu
    - So PReLu both advantages of both ReLu and leaky ReLu
  - PReLu does not vanish during back propagation of many layered network unlike Tanh and Sigmoid functions
    - This because derivative PReLu ranges [-1 ,1] while derivative Tanh [0, 1] and Sigmoid ranges [0, 0.25]
    - Also PReLu has **O(n)** while Tanh and Sigmoid **O(e^x)**
  - In figure 1 & 2 show why PReLu is better , we can loss function figure1 change fast than the figure2 loss function. Figure slow change of loss function is caused by vanishing of gradient for Tanh and Sigmoid function
  - **figure 1**

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): PReLU(num_parameters=1)
    (2): Linear(in_features=512, out_features=64, bias=True)
    (3): PReLU(num_parameters=1)
    (4): Linear(in_features=64, out_features=10, bias=True)
    (5): LogSoftmax(dim=1)
  )
)
```



  - **Figure 2**

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): Tanh()
    (2): Linear(in_features=512, out_features=64, bias=True)
    (3): Tanh()
    (4): Linear(in_features=64, out_features=10, bias=True)
    (5): Sigmoid()
  )
)
```



- OUTPUT LAYER ACTIVATION FUNCTION:
  - I used LogSoftMax(dim=1) for output layer because
    - LogSoftMax is faster than Softmax Sigmoid
      - LogSoftMax is O(log(x))
        - equation = $\mathbf{x_i - \sum x_k}$
      - Softmax O(n)
        - $\mathbf{e^{xi} / \sum e^{xk}}$
      - Sigmoid O(e^x)
        - $\mathbf{1/ 1+ e^{-x}}$
        - Also sigmoid we can be use when we predict binary output and in our case we have more than two units in the output layer
    - So in the above equation we see LogSoftMax is log of Softmax function so if for example given probabilities of Softmax are [0.2, 0.3, 0.1, 0.1, ... n] , these probabilities in LogSoftmax [-0.7, -0.5, -1 ... n) so we LogSoftMax has heavy punishment for wrong prediction than Softmax itself hence LogSoftmax is good classifier than Softmax

- Also LogSoftmax logs probabilities of Softmax and probabilities of Softmax never get small , might range between [0,1] so LogSoftmax has numeric stability
- In the given equation we see LogSoftmax is log graph so is faster
- graphs below show that by means loss function, loss function change very fast and classification are very high in accuracy in **figure 2 of LogSoftmax**

figure 1 Softmax ,Accuracy =96.09%

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): PReLU(num_parameters=1)
    (2): Linear(in_features=512, out_features=64, bias=True)
    (3): PReLU(num_parameters=1)
    (4): Linear(in_features=64, out_features=10, bias=True)
    (5): Softmax(dim=1)
  )
)
```

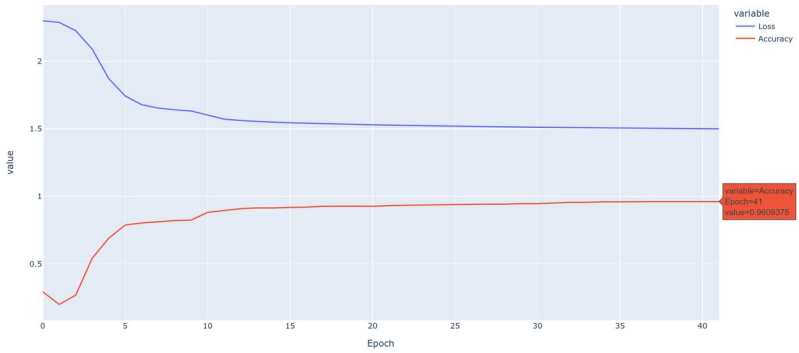
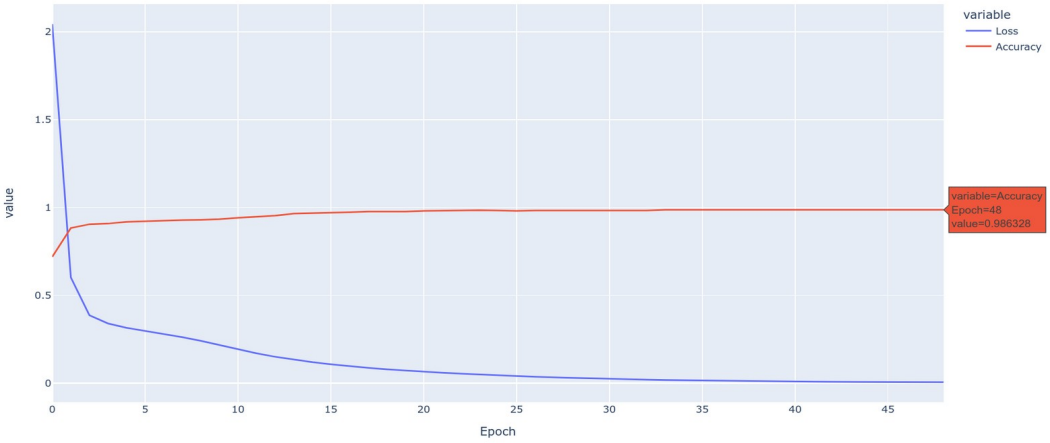


Figure 2 LogSoftmax accuracy =98.63%

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): PReLU(num_parameters=1)
    (2): Linear(in_features=512, out_features=64, bias=True)
    (3): PReLU(num_parameters=1)
    (4): Linear(in_features=64, out_features=10, bias=True)
    (5): LogSoftmax(dim=1)
  )
)
```



DATA PREPROCESSING

DATASET : our data set for training is given on Vula, we download it from there to current directory of source files(python scripts). dataset is in two sets , training data and testing dataset

LIBRARIES(FRAMEWORK) :

- Numpy for calculations operations,
- pandas – for plotly backend , it helped to plot figures on internet instead of the the computer
- pytorch for get ML algorithms and data structure for storing and manipulating data
- PIL for image processing , i used to read JPGs images and transform them to tensor for testing purposes

DATA HANDLING : data is fed to data-loader to model by batches of size 512

DATA ENCODING : tensor[channels, batch size , binary data of the image]

LOSS FUNCTION

- Use Cross entropy as cost/loss function
  - I choose cross entropy because in assignment we classifying more than two objects and output does not use normal distribution (data is not normalised around zero) . Even data has local minimum like data we use MNIST , cross entropy is able to find convex solution
  - Equation for cross entropy. It log term which enable heavy penalty for wrong prediction

$$-\sum_{x} p(x) log q(x) \; + (1 - p(x)) log(1 - q(x))$$

image by [3]

  - I did not Mean square Error(MSE) because we can't use linear regression to classifier our data and our data has local minimum that might cause non-convex(when you cannot converge) problems when we use MSE

OPTIMIZER

- FINAL model which classifier_6 , i used stochastic gradient descent (SGD) with learning rate of 0.01, Momentum 0.9
  - Equation for SGD :

$$v_i = \gamma v_i + \alpha \frac{\partial L}{\partial \theta_i}$$, γ is the momentum ..

  - So here the momentum helps to speed training process . "Momentum accelerates the training process but adds an additional hyperparameter"[5]
  - So With the Momentum the SGD has equal speed that make converge fast and close to Adam optimizer
  - Adam optimizer is fast than SGD but Adam has a very less training time , which ultimately decrease generalization ability of of data
  - [5] argues that SGD is conceptually stable for convex and continuous optimization. First, it argues that minimizing training time has the benefit of decreasing generalization error. This is because the model will not see the same data several times, and the model wouldn't be able to simply memorize the data without generalization ability.
  - I have **figure 1** shows Adam converges fast, is unstable with both accuracy and loss at epochs [10 ,20] and accuracy is low than the SGD. **figure 2** SGD is very smooth converging and accuracy is uniform increasing and then constant, this support that SGD generalize better hence better for classification hence i choose it over Adam

Topology used for investigation, everything is constant for figure1 and 2 except added Momentum for SGD

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): PReLU(num_parameters=1)
    (2): Linear(in_features=512, out_features=64, bias=True)
    (3): PReLU(num_parameters=1)
    (4): Linear(in_features=64, out_features=10, bias=True)
    (5): LogSoftmax(dim=1)
  )
)
```

Figure 1

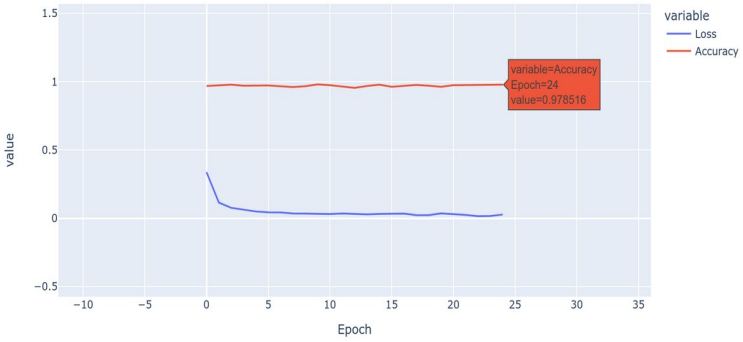
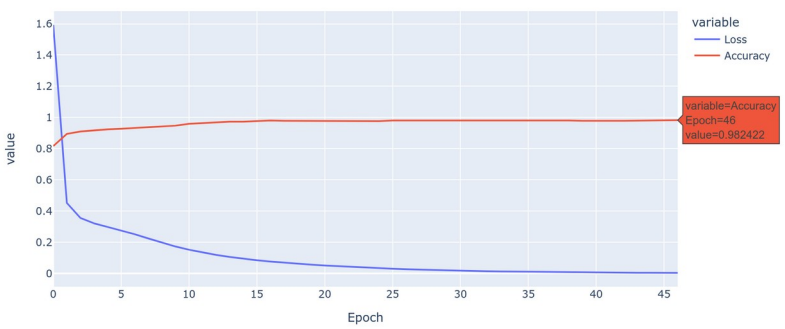


Figure 2

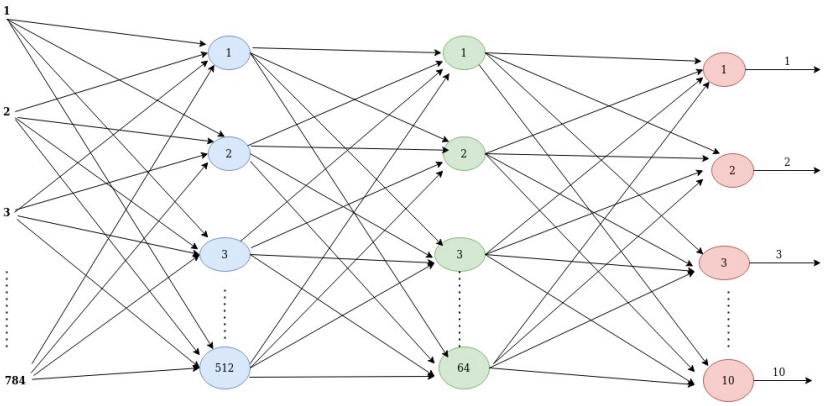


TRAINING AND VALIDATION

- Use early stopping learning
  - [6] *Early stopping* is that every certain number of learning epochs, we compute an evaluation measure (e.g., classification accuracy) on the validation set. If there is an improvement over the previous time, we store the current values of the neural network weights, and if there is no improvement for several times, we stop learning and restore the weights of the last best model. We prepare a helper function to compute the classification *accuracy*. In each row of logits we select the column number with the largest value, compare it with the expected value, and average the values so obtained (0 or 1) over the examples to obtain the classification accuracy.

Final ANN

Topology



Activation function

```
NeuralNetwork(
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): PReLU(num_parameters=1)
    (2): Linear(in_features=512, out_features=64, bias=True)
    (3): PReLU(num_parameters=1)
    (4): Linear(in_features=64, out_features=10, bias=True)
    (5): LogSoftmax(dim=1)
  )
)
```

- To get to this ANN ,
  - I first investigated number layers and number units per layer from page 1 to page 2 .With graph figures showing in page 1 to page 2 that topology with 2 hidden layer , 1st hidden layer with 512 units and second hidden layer having 64 units produces highest accuracy, fast to converge
  - Secondly , i investigate activation to use in my topology from page 3 to page 4. With graphs figures and proofs from equation , i concluded that PReLu is best activation function for hidden layer as provide both function of ReLu and leaky ReLu depending on the value of **α**

    f(y) = { y, y>=0

    αy , y<0

    PReLu is very fast , it does now vanish on back propagation and normalise data around zero
  - With proof in the graphs which suggest that LogSoftmax highest accuracy of 98.63% and equations analysis – suggest that it fast than because of loss function decreases significant in first few epoch than any other classifier activation function like Sigmoid and Softmax and also has heavy penalty for wrong prediction and provide numerical stability, page 4 says.
  - Thirdly , cross entropy as loss function . In page 5 , explained that in the assignment we use data with local minimum and we classifying multiple object so with that in mind cross entropy is best loss function to find optimum convergence solutions and provide heavy penalty outputs with wrong prediction
  - Fourth, Investigated best optimization function and SGD with learning rate 0f 0.01 and momentum=0.9 best optimizer due to its accuracy and most important its best generalization of data and stability than Adam optimization . In Page 5 proof how we investigated is provided

Reference :

1. Tyler Elliot Bettily , 8 Aug 2018, How to classify MNIST digits with different neural network architectures
2. Bikal Basnet , 27 February , 2020, Log Softmax Vs Softmax, https://deepdatascience.wordpress.com/2020/02/27/log-softmax-vs-softmax/
3. Anjali Bhardwa , Nov 3, 2020, What is Cross Entropy?, https://towardsdatascience.com/what-is-cross-entropy-3bdb04c13616
4. Jason Brownlee , August 25, 2020, Deep Learning Performance, https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/
5. Hardt, M., Recht, B., & Singer, Y. (2016, June). Train faster, generalize better: Stability of stochastic gradient descent. In *International Conference on Machine Learning* (pp. 1225–1234). PMLR.
6. Learning with early stopping, https://colab.research.google.com/drive/1jrKpcF6AVCh1M6_2aW9j-QpWnzOZh_mh?usp=sharing#scrollTo=vDjSqSeQyJYk