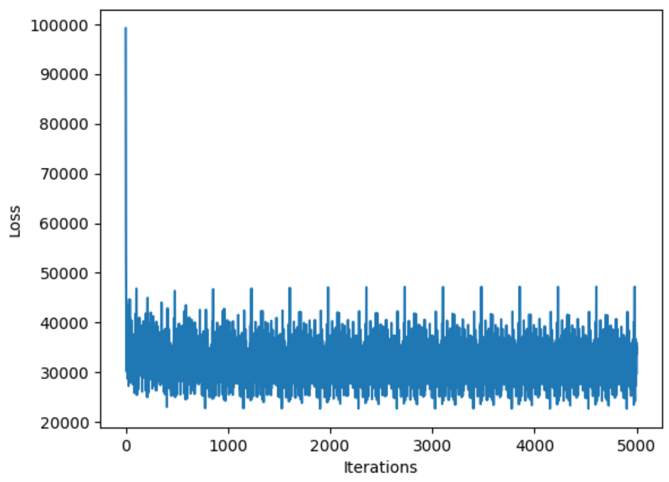
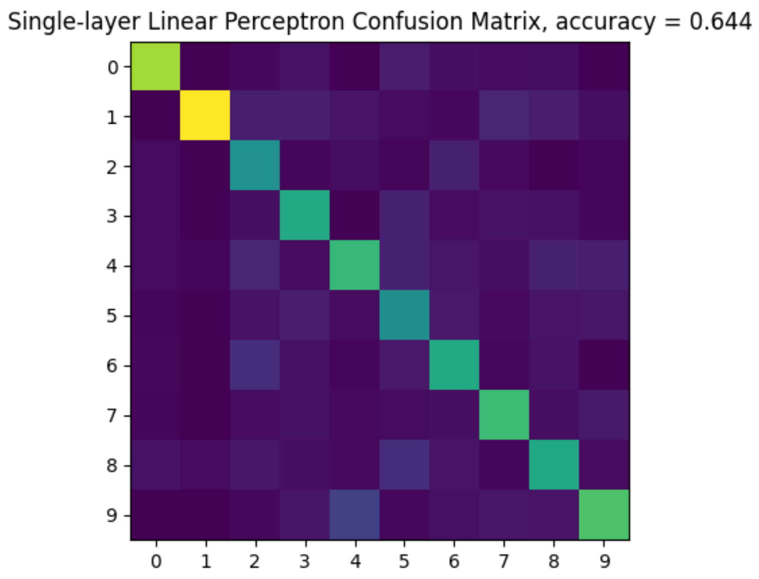
Asal Shavandi **Computer Vision HW4** **UMN S2022**

1. Single-Layer Linear Perceptron

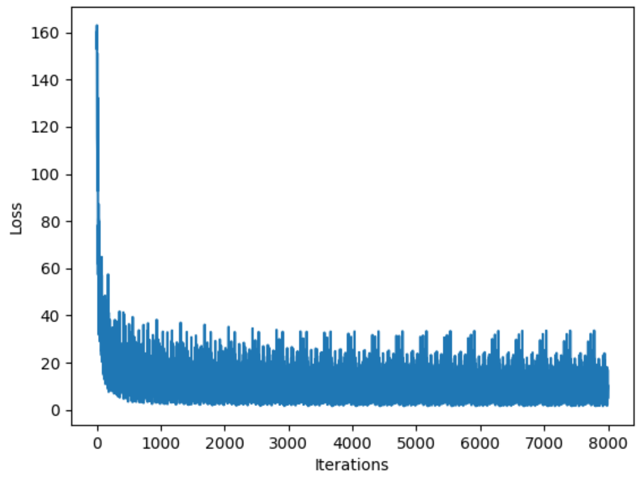
In this single layer perceptron, there are weights and bias and output will be y=w*x+b. There will be no activation function, that is why it is called linear perceptron.
Euclidean Loss is used to compute loss and Stochastic gradient descent as an optimize.
My hyper parameters are: learning_rate=0.01, decay_rate=0.9 and number of iterations is 5000.
I could get an accuracy of 64% which is higher than expected accuracy 30%.

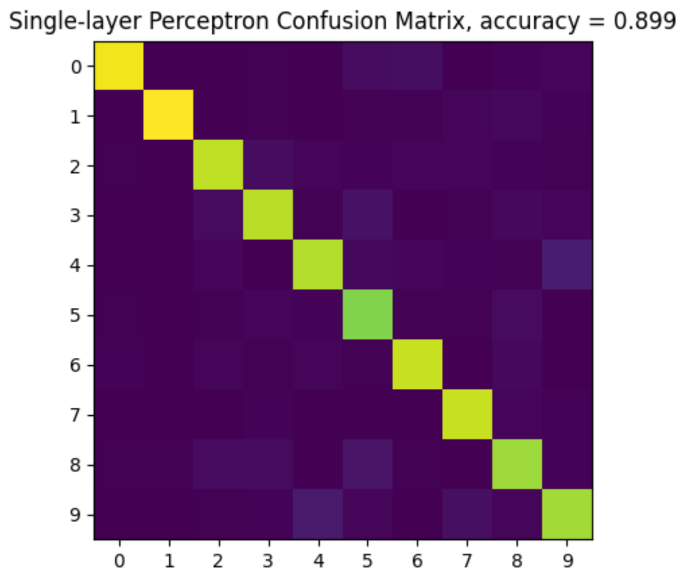




2. Single-Layer Perceptron

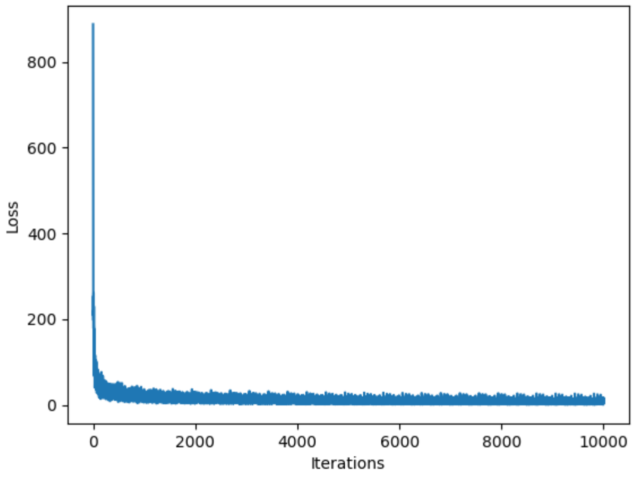
This network can be identical to single-layer linear perceptron except that activation function is applied to the output to propose some non-linearity into the training model.
Also Cross Entropy Loss is used instead of Euclidean loss and same stochastic gradient descent as an optimizer.
My hyper parameters are: learning_rate=0.7, decay_rate=0.9 and number of iterations is 8000.
I could get an accuracy of 89%. The expected accuracy was near 90%.

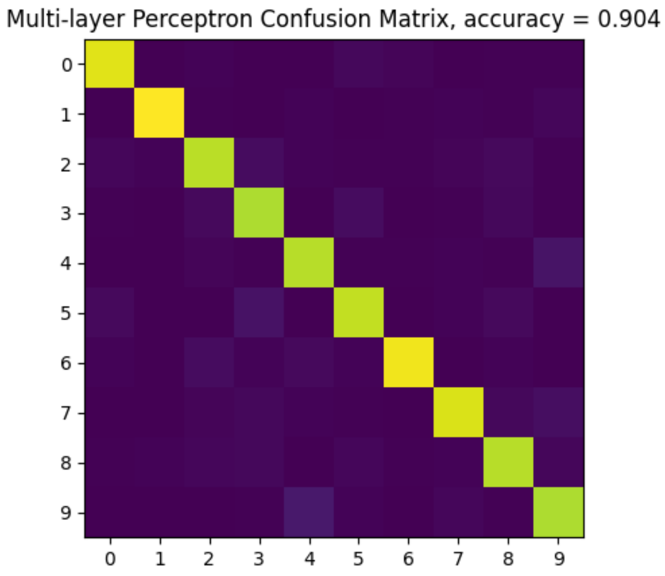




3. Multi-Layer Perceptron

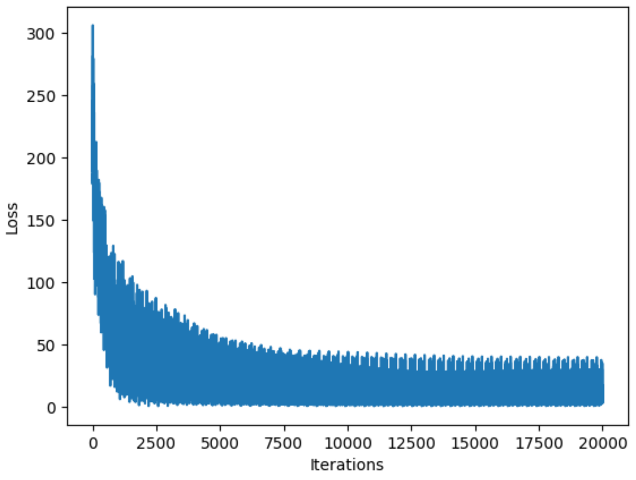
In this network we will have a hidden layer with 30 nodes, that is why it's called multi-layer perceptron. ReLU activation function will be applied on the first layer output and at the end will use Softmax to classify out multi-class data.
Cross Entropy for loss and stochastic gradient descent as an optimizer are used.
My hyper parameters are: learning_rate=0.2, decay_rate=0.9 and number of iterations is 10000.
I could get an accuracy of 90.4%. The expected accuracy was more than 90%.

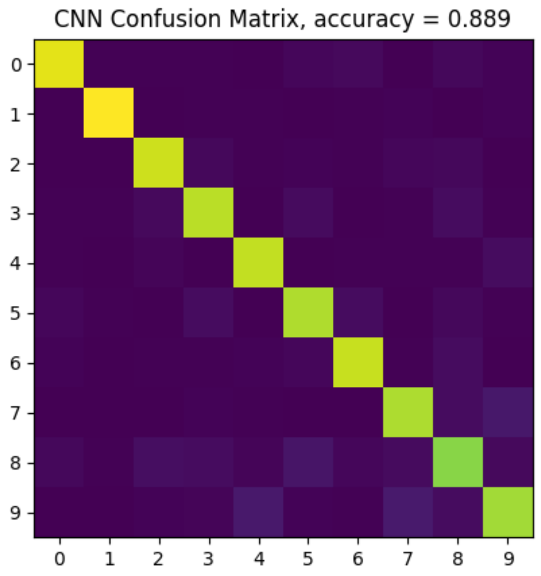




4. Convolutional Neural Network

The order of training a CNN model for MNIST dataset is as below:
Single channel input -> convolved with 3x3 filter -> output convolved image in 3 channels -> ReLU layer -> Max pooling with size 2x2 that reduces the image size to half -> flattening layer -> fully-connected layer -> finally Cross Entropy for loss and Softmax for classification.
My hyper parameters are: learning_rate=0.038, decay_rate=0.95 and number of iterations is 20000.
I could get an accuracy of almost 89%.


Loss
Iterations


CNN Confusion Matrix, accuracy = 0.889