COMP ENG 4SL4: Machine learning

Assignment 4 - Neural Networks

Instructor: Dr. Sorina Dumitrescu

TA: Zewei Zhang

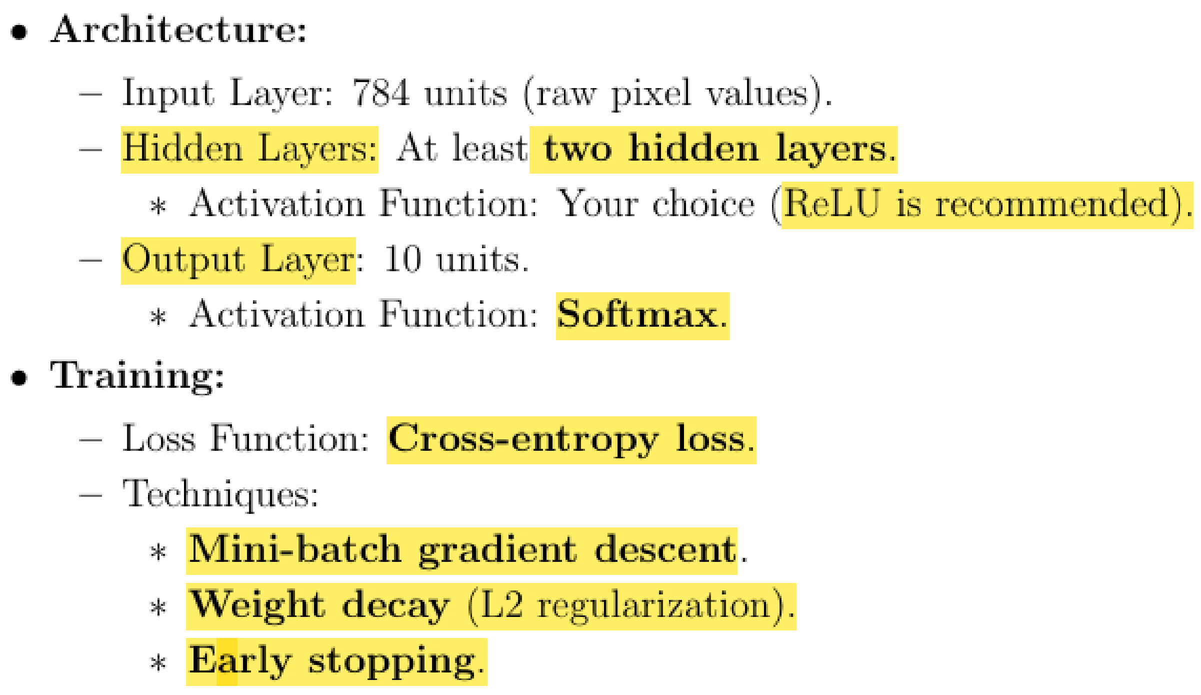
Richard Qiu – qiur12 – 400318681

Introduction

The objective of this lab is to implement a custom neural network to solve a multi-class classification problem and also use pytorch library to implement the similar neural network.

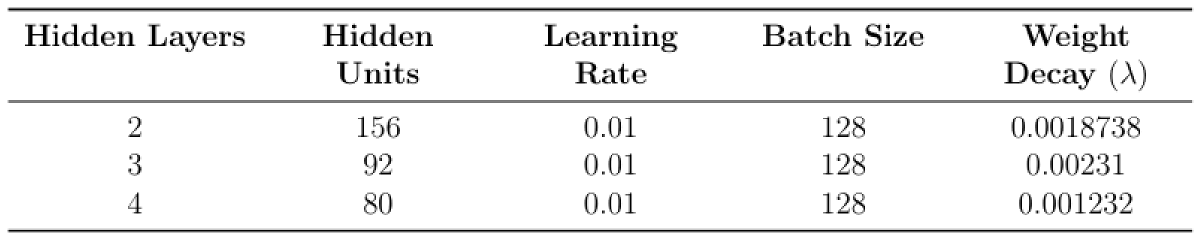
Methodology

The custom neural network implementation contains below features where there will be 3 hidden layers, 1 input layer and 1 output layer.



- **Architecture:**
  - Input Layer: 784 units (raw pixel values).
  - Hidden Layers: At least **two hidden layers**.
    * Activation Function: Your choice (ReLU is recommended).
  - Output Layer: 10 units.
    * Activation Function: **Softmax**.
- **Training:**
  - Loss Function: **Cross-entropy loss**.
  - Techniques:
    * **Mini-batch gradient descent**.
    * **Weight decay** (L2 regularization).
    * **Early stopping**.

Hyperparameters use to train the model is selected as shown below.



Hidden Layers	**Hidden Units**	**Learning Rate**	**Batch Size**	**Weight Decay** (λ)
2	156	0.01	128	0.0018738
3	92	0.01	128	0.00231
4	80	0.01	128	0.001232

The backpropagation method is used for the training process. For the forward pass, ReLU is chosen as the activation function for each hidden layer and on the output layer, the Softmax activation function is used to calculate the final output.

The gradient of the cross-entropy loss with respect to the output layer is simply the difference between the output of the forward loss and the actual label value. And after finding out the dl/dz4, we can then use chain rule to find dl/dw4 by using below formula from lecture slides where the only difference is in our model, bias will be added separately.

$$\nabla_{\mathbf{W}^{(3)}} J = \frac{\partial J}{\partial z^{(3)}} \left(1 \quad \mathbf{h}^{(2)^T}\right), \qquad (7)$$

And below the code is the gradient implementation on the last layer.

```
d_loss_z4 = y_pred - y_true  # Derivative of cross-entropy loss w.r.t softmax output
# Gradients for the output layer
d_loss_w4 = np.dot(self.h3.T, d_loss_z4)/ batch_size + weight_decay_4 * self.w_4
d_loss_b4 = np.sum(d_loss_z4, axis=0, keepdims=True)
```

After found out w4, we can then apply the another formula below to calculate the gradient before the activation function. In this formula, we would also need to find out the derivative of ReLu which for any numbers greater than 0 would be 1 and else to be 0. The weights of the layer after this hidden layer(w4) and the z4 would perform elementwise multiplication to the ReLu derivative. Also because we are using the batch gradient descent, the batch size will be divided to average the gradient.

$$\nabla_{\mathbf{z}^{(2)}} J = \begin{pmatrix} g'(z_1^{(2)}) \\ g'(z_2^{(2)}) \end{pmatrix} \odot \left\{ \bar{\mathbf{W}}^{(3)^T} \cdot \frac{\partial J}{\partial z^{(3)}} \right\} \qquad (4)$$

Below is the gradient implementation on hidden layer 3.

```
# Gradients for third hidden layer
# test_relu = self.relu_derivative(self.z3)
d_loss_z3 = self.relu_derivative(self.z3) * np.dot(d_loss_z4,self.w_4.T)
d_loss_w3 = np.dot(self.h2.T, d_loss_z3)/ batch_size + weight_decay_3 * self.w_3
d_loss_b3 = np.sum(d_loss_z3, axis=0, keepdims=True)
```

And by following the same method, I find the gradients of each layer till the first hidden layer.

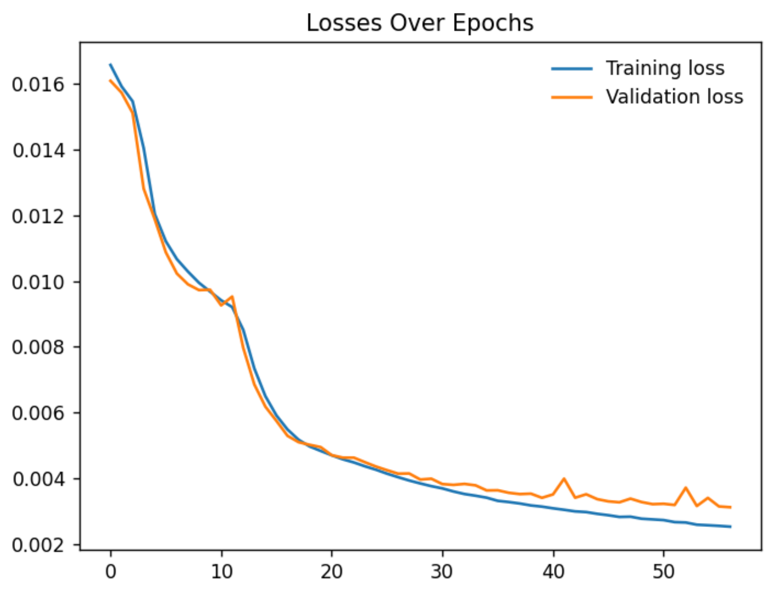
After found out gradients of each layer, then I can update the parameters correspondingly by following below formula.

$$\mathbf{W}^{(j)(new)} = \mathbf{W}^{(j)(old)} - \alpha \frac{\nabla_{\mathbf{W}^{(j)}} J}{M}$$

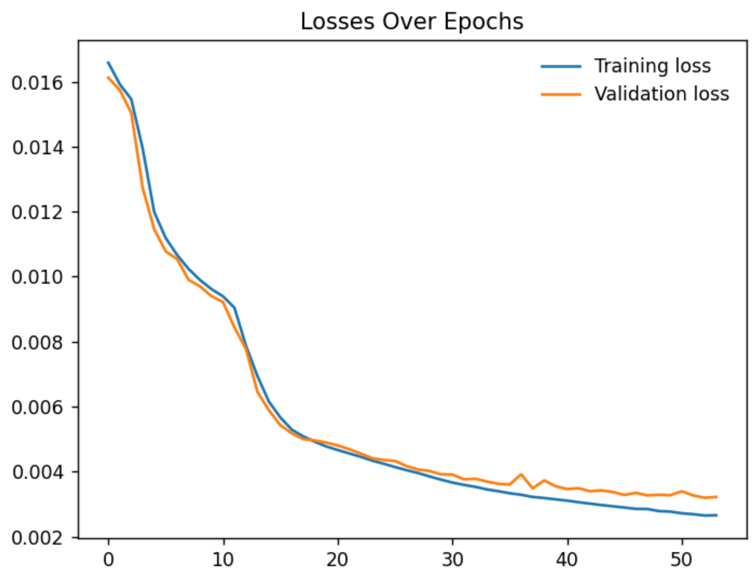
Results

Custom Neural Network

Seed = 8681. Takes 57 Epochs to reach 14% misclassification rate on training.



Seed = 4003. Takes 54 Epochs to reach 14% misclassification rate on training.

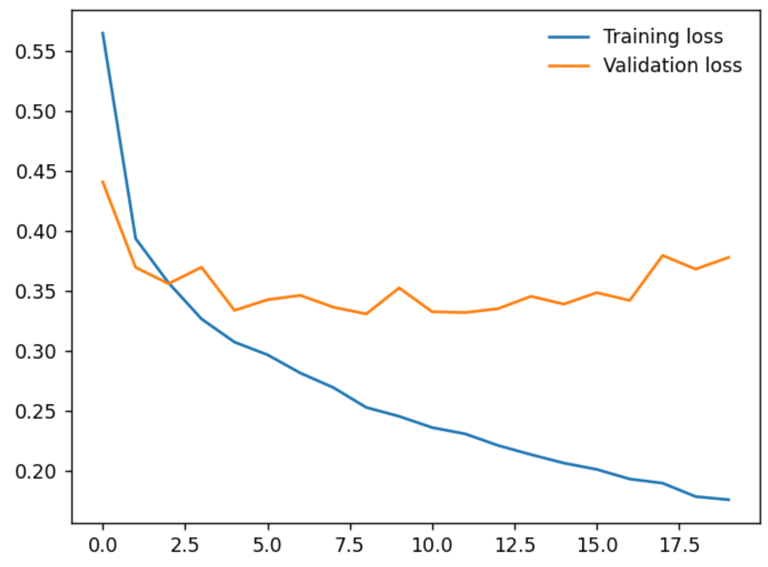


Summarize of the misclassification errors on custom neural network.

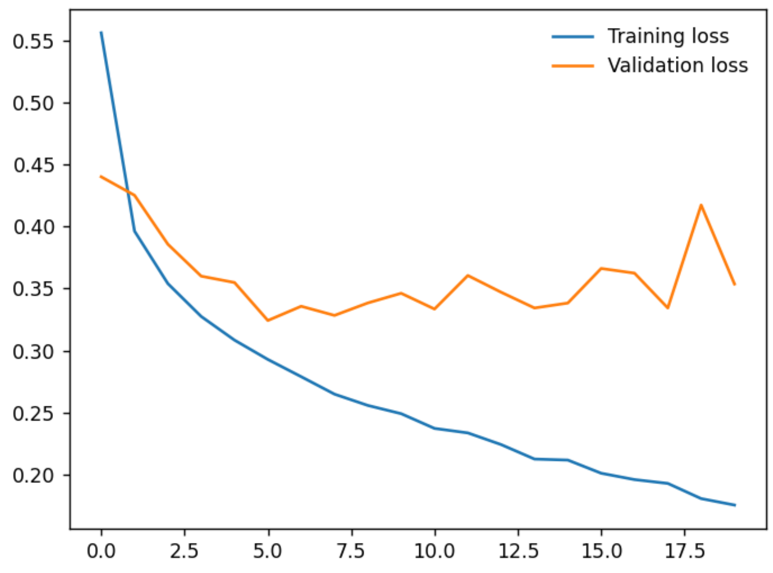
Seed	Train Error	Test Error
8681	0.134	0.14
4003	0.14	0.151

Pytorch Neural Network

Seed = 8681. Takes 2 Epochs to reach 14% misclassification rate on training. Adam filter.



Seed = 4003. Takes 3 Epochs to reach 14% misclassification rate on training. Adam filter.



Summarize of the misclassification errors on Pytorch neural network.

Seed	Train Error	Test Error
8681	0.113	0.073
4003	0.109	0.068

Discussion

In the pytorch implementation neural network, the log_softmax activation function is used at the output layer. The Adam optimization technique is used to train the neural network and NLLLoss is used as the loss function for a better numerical stability to provide robust gradient computation. With the help of pytorch API, it is easy to implement this high-level optimization technique. But for the custom neural network implementation, the mini batch gradient descent is chosen to train the network with a small batch size of 128. By trying different training methods, for the custom implementation, we can see that for the first 20 epochs, the losses is improving quickly and once it is over 20, it becomes stable around 0.003 where the training accuracy is around 0.80. And also with two different random seeds, the initialization of the initial weights would be different, as I observed that when seed is equal to 4003, it takes few epochs to reach the misclassification rate of 14%. And for Pytorch implementation, as it meets the misclassification rate pretty quick, I extend the epochs to let it run until 20 and found when the epochs is over 17, there is a slight increase in the validation loss which might identify the occurrence of overfitting since the model is trained so well on the training loss and loss the accuracy on validation sets. So the epochs between 5.0 and 10.0 could provide a better result of the test accuracy.

Summary

By experimenting two ways of implementing neural networks, I deepened my understanding on the back propagation algorithm by implementing the mini batch gradient descent. And I also found that overfitting and underfitting can still happen in the neural network if the epochs is too small or too large, which it is always important to test the model with different epochs. If I have more time, I would love to increase the epochs of my custom implementation to see what value of epoch will cause the overfitting to occur.