Deep Learning 2020 **(Due: 11th Nov'2020)**

Submission Assignment #1

Instructor: Jakub Tomczak *Name:* Asif Anwar, *Netid:* aar216

1 Question answers

Question 1 *To calculate the cross entropy loss, we need to calcuate the derivatives of the softmax function. then we can workout the derivative of the loss. below are the steps to derive the derivatives.*

Deriving $\frac{\delta y_i}{\delta o_i}$ for i = j.

$$\frac{\delta y_i}{\delta o_i} = \frac{\delta \frac{\exp o_i}{\sum_k \exp o_k}}{\delta o_i} = \frac{\exp o_i \sum_k \exp o_k - \exp o_j \cdot \exp o_i}{(\sum_k \exp o_k)^2} = \frac{\exp o_i (\sum_k \exp o_k - \exp o_j)}{(\sum_k \exp o_k)^2}$$

$$= \frac{\exp o_i}{\sum_k \exp o_k} \cdot \frac{\sum_k \exp o_k - \exp o_j}{\sum_k \exp o_k} = y_i(1 - y_i)$$

Deriving $\frac{\delta y_i}{\delta o_i}$ for i $\neq$ j.

$$\frac{\delta y_i}{\delta o_j} = \frac{\delta \frac{\exp o_i}{\sum_k \exp o_k}}{\delta o_j} = \frac{0 - \exp o_j \cdot \exp o_i}{(\sum_k \exp o_k)^2} = \frac{\exp o_j}{\sum_k \exp o_k} \cdot \frac{expo_i}{\sum_k \exp o_k} = y_j \cdot y_i$$

Derivatives of loss $L = -\sum y_i \log(p_i)$, here y_i is the target value and p_i is the output value. so,

$$\frac{\delta L}{\delta o_i} = -\sum_k y_k \frac{\delta \log p_i}{\delta o_i} = -\sum_k y_k \frac{\delta \log p_i}{\delta p_k} \cdot \frac{\delta p_k}{\delta o_i} = -\sum y_k \frac{1}{p_k} \cdot \frac{\delta p_k}{\delta o_i}$$

from the above section we can apply the derivation of softmax function to calculate the derivative of loss.

$$\frac{\delta L}{\delta o_i} = -y_i(1 - p_i) - \sum_{k \neg i} y_k \frac{1}{p_k}(-p_k p_i) = -y_i(1 - p_i) + \sum y_k p_i = p_i(y_i + \sum_{k \neg i} y_k) - y_i = p_i - y_i$$

- The derivatives of $\frac{\delta y_i}{\delta o_i}$ and $\frac{\delta y_i}{\delta o_j}$, for i = j is $y_i(1 - y_i)$ and for i $\neq$ j is $y_j \cdot y_i$
- The derivatives of $\frac{a\delta l}{\delta o_i} = y_i - p_i$, here y_i is the target value and p_i is the output value.(Dahal, 2017)

Question 2 *The network setup was constructed as per the given requirements. Only the math and random libraries have been used respectively for exponental, log and random function to build the network. Vital portion of the code is shared below.*

Support Functions: To design the network, below activation, loss and gradient calculation functions were used. Nested *for loops* were used to build different list to simply the code.

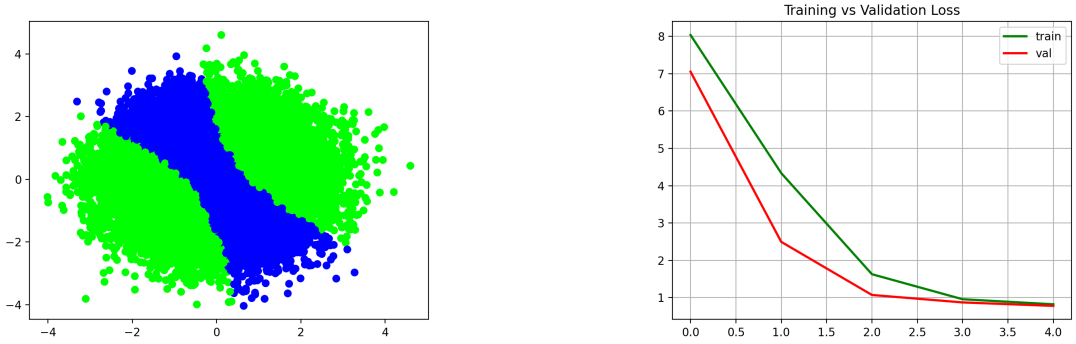
```
def sigmoid(inputs):
    return [(i/(i + math.exp(-i))) for i in inputs]
def softmax(inputs):
    exp_val = [math.exp(i) for i in inputs]
    return [exp/sum(exp_val) for exp in exp_val]
def log_loss(y_hat, y_target):
    return sum([-math.log(y_hat[i]) * y_target[i] for i in range(len(y_hat))])
def gradient_loss(y_hat, y_target):
    return [y - yt for y, yt in zip(y_hat,y_target) ]
def gradient_sigmoid(layer, grads):
    return [g*exp*(1-exp) for g, exp in zip(grads, layer.activation_output)]
```

Layer class: *The layer class is the heart of this network. The network initialization is done based on input size and number of neuron. The weights as sets by using a random function.Later weights were changed to given values. In the* forward_pass *function the layer output is calculated and then based on* activation *parameter activation function was called. In* backward_pass *function layer gradient is calculate.*

```
class layer():
    def __init__ (self, input_size, neuron_no):
        self.neuron_no = neuron_no
        self.biases = [0. for i in range(neuron_no)]
        self.weights = [[.01*random.randint(0,9) for i in range(neuron_no)] for i in range(
    input_size)]
    def forward_pass(self, inputs, activation = 'sigmoid'):
        self.inputs = inputs
        self.outputs = [sum([w[n]*x for w,x in zip(self.weights, self.inputs)]) + self.biases[
    n] for n in range(self.neuron_no)]
        if activation == 'sigmoid': self.activation_output = sigmoid(self.outputs)
        elif activation == 'softmax': self.activation_output = softmax(self.outputs)
    def backward_pass(self, grads):
        self.dw = [[i*g for g in grads] for i in self.inputs]
        self.db = grads
        self.grads = [sum([w[i] * grads[i] for i in range(len(w))]) for w in self.weights]
```

Question 3 *The synthetic data has been loaded with the given* load_synth *function. Then below* SGD *function has been used to optimize the weights and biases for training.A total of 60,000 training and 10,000 validation data has been used for the training. The* learning rate was kept to a very minimum level. The model training was just 5 epochs to avoid over-fitting.(Harris, 2020)

```
def SGD(layer, lr):
    updated_weights = [ [w - lr * dw for w, dw in zip(ws, dws)] for ws, dws in  zip(layer.
    weights, layer.dw)]
    layer.weights = updated_weights
    layer.biases = [(b - lr*db) for b,db in zip(layer.biases, layer.db)]
```



(a) Class distribution

(b) Loss per epochs

Figure 1: Class and Epoch Loss for synthetic data

Information on training is given below

- The training was done over 60,000 train and 10,000 validation data
- Learning rate was set to low (0.00001)
- A total of 5 epochs are used
- One hot encoding is used to fit the target data with output data (Brownlee, 2017)
- Each instances of data was forward sequencialy on each epochs
- Above stochastic gradient descent (SGD) function was used to optimize the weights and biases.

Question 4 *The MNIST data set has been down loaded with provided function. The network was structured as per the given instructions. The mini_batch method has been implemented and source code has been updated to handle vectorized data using numpy library. The vital information of the network and its output is shared below*

- The network was initialized with given input size and number of neurons.

```
# Setting up the network
input_layer = layer(neuron_no=300, input_size=784)
output_layer = layer(neuron_no=10, input_size=300)
```

- Total number of training and validation dataset are 60,000 and 10,000 respectively.
- Input data was normalized and target data was converted oneHotEncoded to match the network.
- Learning rate 0.01 is used
- total 20 epochs were used to training
- Vectorization was done by using numpy libraries
- Mini batch was introduced to training the network

Question 5 *The objective of this section is to designing a neural network by implementing sigmoid, softmax activation function, cross-entropy loss and stochastic gradient descent. And then train the network over MINIST dataset. The goal is to reduce the loss and improve the accuracy of the network by changing the hyperparameters. The details of the experiments is shared below:*

2 Problem statement

Tuning the hyperparameters to improve the neural network for MINIST dataset.

- Training and validation loss comparison per epochs for MINIST Dataset
- Changes in standard deviation and average of loss and accuracy by using stochastic gradient descent optimizer.
- The influences of the learning rate over the performance of the network
- Applying different data sources and evaluate the network performance

3 Methodology

Overview: *To accomplish the target problems, different training mechanism needs to be used. Initially the training will be done with 55000 data instead of full dataset. with this training data the network can be trained in different stages by changing different parameters and observe the performance impact on the network.*

Strategy *The initially weights and biases of the network will be randomly selected and the network will be training using the mini batch method by sending a segmented data at a time. And then initial performance will be recorded. Then to see if there is any major performance difference of the network for using SGD as the optimizer, the network will be training few times and change in standard deviation and average will be investigated. Then different learning rates will be implemented to verify the performance. Based on the outcome of all the experiments the best hyperparameters will be selected and full train and test set will be used to find the final performance of the network. below pseudo code will be used for training the model*

Data: Load and Normazlize Dataset
initialization the Network;
while *all learning rates and epochs* **do**
  while *Mini batch traning dataset* **do**
    Do forward propagation;
    Calculate loss & accuracy ;
    Do backward propagation;
    Updated weights and biases;
  end
  while *Mini batch validation dataset* **do**
    Do forward propagation;
    Calculate loss & accuracy;
  end
end
Result: Plot the accuracy and loss

Algorithm 1: Pseudo code for trining over different learning rates

4 Experiments

Mini batch over SGD *50000 intense of MINIST dataset was loaded, normalized and reshaped for training. Then the network is training for 20 epochs with learning rate of 0.001 and loss and accuracy was recorded.*

Std and Mean for SGD *To verify the impact of stochastic gradient descent the network was trained for 30 iterations. Then the standard deviation and average of loss and accuracy was observed the view the impact of SGD over loss*

Change in learning rates *The training was conducted with 0.0001, 0.01, 0.05 over 5 epochs to verify the impact on performance*

Selecting hyperparameters *Based on the experiments over different hyperparameters, the best results can be selected. After the selection of the hyperparameters, the network was training with full MNIST dataset along with canonical test set.*

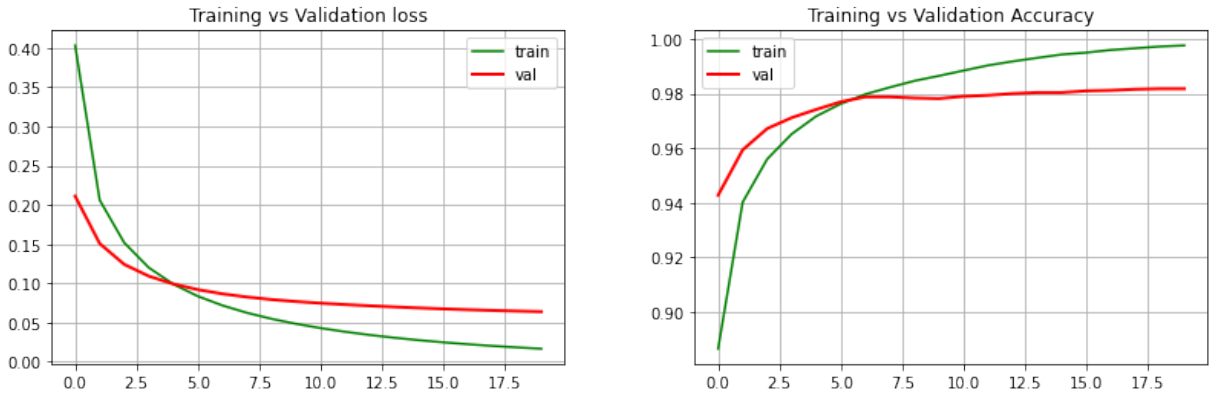
Training over Fashion dataset *To investigate further on the neural network. A final test was performed over Fashion mnist dataset. One extra hidden layer and new activation function was introduced to verify the performance of the network.*

5 Results and discussion

Mini batch over SGD *After training the network over 20 epochs with learning rate 0.001 we can observe that we get fairly good accuracy of 98% validation data and 99.8% over training data. And after epochs 5 there is hardly any changes over accuracy and loss for validation, however the training performance gets a lot better. This is a sign for over fitting. Hence from this we can conclude that we can keep our training epochs limited around 5. This can observe on* Figure 2

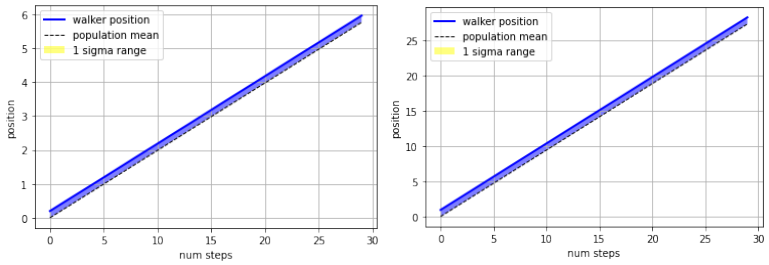
Std and Mean for SGD *From figure 3 it can be observed that the standard deviation and average of loss and accuracy of the training do not changes for 30 iteration. Which means that stochastic gradient descent is performing well and in similar pattern regardless of initial random weights and biases.*

Change in learning rates *From the figure 4 we can observe that learning rate 0.01 and 0.05 are performing better over train and validation data. And where 0.05 and 0.01 are almost results almost similar loss and accuracies. On figure 4.b we can see that learning rate 0.01 performed slightly better for validation accuracy. As the validation accuracy is important here we can conclude that 0.01 learning rate is the ideal for this network given this dataset.*



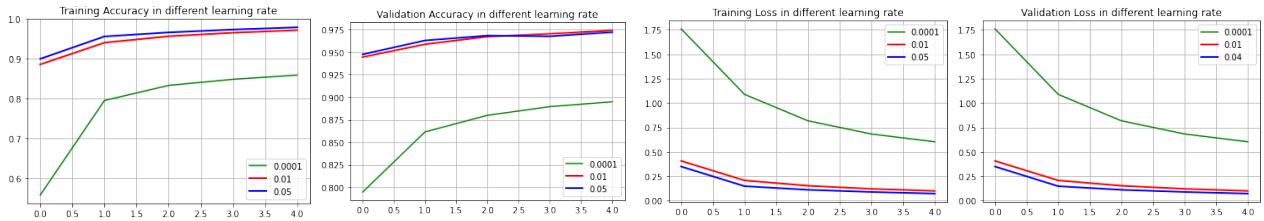
(a) Loss per epoch (b) Accuracy epochs

Figure 2: Loss and Accuracy over 20 epochs



(a) Accuracy (b) Loss

Figure 3: Random walker plotting for loss anc accuracy

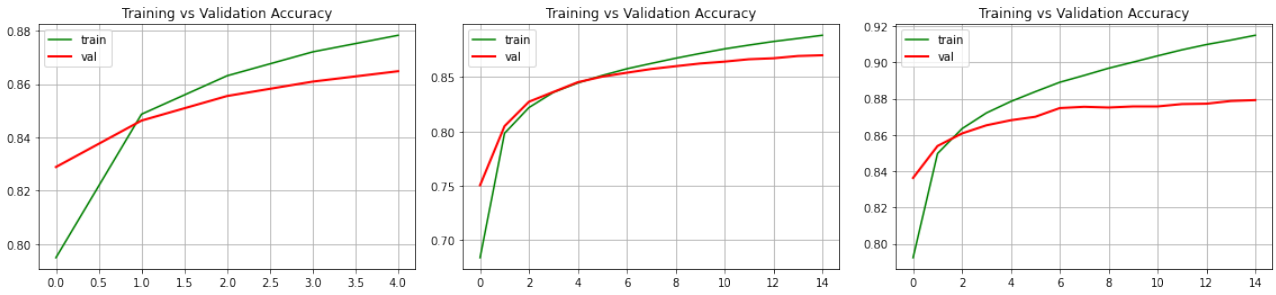


(a) Train Acc (b) Val Acc (c) train Loss (d) Val loss

Figure 4: Impact of learning rate over accuracy

Selecting hyperparameters *After the sequential training over the MNIST dataset, the final selected hyper parameters are epochs=5, learning rate = 0.01, batch size = 50. Using these hyperparameters over the full dataset we get* ***97.5% validation Accuracy***

Training over Fashion dataset *The accuracy of The Fashion MNIST dataset over the same structure is* 86.5%. *After adding another layer with* sigmoid activation *with 300 neurons the accuracy improves slightly to* 87%. *And after implementing* ReLu *activation over the hidden layer, the validation accuracy only increases to* 87.9%. *After two epochs the network tents to overfit as the train accuracy keep raising while validation accuracy remain same.*



(a) 2 layers NN (b) 3 Layers NN (c) 3 Layer with Relu

Figure 5: Validation accuracy over Fashion MNIST

References

Brownlee, J. (2017). How to One Hot Encode Sequence Data in Python.
Dahal, P. (2017). Classification and Loss Evaluation - Softmax and Cross Entropy Loss.
Harris (2020). Neural Networks from Scratch.

Appendix

Colab gist links are shared below for easy access to the project codes

- Question 5: https://colab.research.google.com/gist/tamal2000/ff52b68ccd88cceed90057dc08bd793b/q5.ipynb
- Fashion MNIST: https://colab.research.google.com/gist/tamal2000/64c029c5a6592b89559fe8b14856f5c9/q5-fashion.ipynb