Assignment 2:

Behnam Saeedi
(Saeedib@oregonstate.edu)
CS535: Deep Learning
Due Feb 17th 10:00am
Winter 2019

1 Question 1

Write a function that evaluates the trained network (5 points), as well as computes all the sub-gradients of W_1 and W_2 using back-propagation (5 points).

1.1 Answer:

We need to compute the error:

```
def evaluate(self, x, y):
        # INSERT CODE for testing the network
        result = 0
        for i in range(0,y.shape[0]):
                output = self.model.forward(x[i])
                print(output)
                if (y[i] == 1 and output > 0.5) or (y[i] == 0 and output < 0.5):
                        result += 1
                #print ([y[i], output, (y[i] > 0.5 and output > 0.5) or (y[i] < 0.5 and output
        print([result, y.shape[0],result/y.shape[0]])
        return result/y.shape[0]
```

---

2 Question 2

Write a function that performs stochastic mini-batch gradient descent training (5 points). You may use the deterministic approach of permuting the sequence of the data. Use the momentum approach described in the course slides.

2.1 Answer:

let's work out the math first: Forwards propagation activation values:

- **Input layer:** $a_{l0} = x$
- **Hidden layer:** $a_{l1} = z_1 = ReLU$
- **Output layer:** $a_{l2} = z_2 = \frac{1}{1+exp(-W^T\,(\frac{1}{ReLU(W^T\,x-b_1)})-b_1)}$

The overal activation could be generalized as:

$$a_l = W_l^T a_{l-1} + b_l$$

The loss for forwards pass could be computed by:

$$y * \log\left[\frac{1}{1+exp(-W^T\,(\frac{1}{ReLU(W^T\,x-b_1)})-b_1)}\right] + (1-y*)\log(1-\left[\frac{1}{1+exp(-W^T\,(\frac{1}{ReLU(W^T\,x-b_1)})-b_1)}\right])$$

and now we can compute δs:

$$\delta_1 = \frac{\partial E}{\partial z_1} = W^T\delta_2 z_1' \Rightarrow \frac{\partial E}{\partial W_1} = W^T\delta_2 z_1' z_1 x$$

$$\delta_2 = \frac{\partial E}{\partial z_2} = \nabla_{z_2} E z_2' \Rightarrow \frac{\partial E}{\partial W_2} = \nabla_{z_2} E z_2' z_1$$

Weight update:

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial z_2}\frac{\partial z_2}{\partial W} = \frac{\partial E}{\partial z_2}\frac{\partial z_2}{\partial z_1}\frac{\partial z_1}{\partial W}$$

We can break this down and compute each derivative individually:

$$(*)\frac{\partial E}{\partial z_2} = y*\frac{1}{z_2} + (1-y*)\frac{1}{1-z_2}$$

$$(**)\frac{\partial z_2}{\partial z_1} = \sigma(-W^T\ z_1 - b_2).(1-\sigma(-W^T\ z_1 - b_2).(-z_1))$$

$$(***)\frac{\partial z_1}{\partial W} = \sigma(-W^T\ x - b_1).(1-\sigma(-W^T\ x - b_1).(-x))$$

In order to do that we need to compute the gradient:

```
def LT.backward(
        self,
        grad_output,
        learning_rate=0.0,
        momentum=0.0,
        l2_penalty=0.0,
):
        # DEFINE backward function
        return grad_output
def ReLU.backward(
        self,
        grad_output,
        learning_rate=0.0,
        momentum=0.0,
        l2_penalty=0.0,
):
        # DEFINE backward function
        x = np.copy(grad_output)
        x[x > 0] = 1
        x[x == 0] = 0
        x[x < 0] = 0
        return x
def derivative_SCE(self,x): # this is the derivative of sigmoid cross entropy
    return ((self.y_star - 1)*np.exp(x) + self.y_star) /(np.exp(x)+1)
# Now we can ompute gradients
delta_w2 = w2_error * (self.output * (1 - self.output))
        w1_error = self.output * self.w2
        delta_w1 = w1_error * self.ReLU.backward(self.hidden)
        self.w2+= learning_rate *delta_w2 * self.hidden
        tmp_x = (np.copy(self.x))[np.newaxis]
        self.w1 += learning_rate * tmp_x.T.dot(delta_w1[np.newaxis])
        self.update_weights()
```

3 Question 3

Train the network on the attached 2-class dataset extracted from CIFAR-10: (data can be found in the cifar-2class-py2.zip file on Canvas.). The data has 10,000 training examples in 3072 dimensions and 2,000 testing examples. For this assignment, just treat each dimension as uncorrelated to each other. Train on all the training examples, tune your parameters (number of hidden units, learning rate, mini-batch size, momentum) until you reach a good performance on the testing set. What accuracy can you achieve? (20 points based on the report).

3.1 Answer:

In the given number of epochs I only managed to reach an accuracy of around 0.6. I hypothesis that this is due to my small learning rate.

---

4 Question 4

Training Monitoring: For each epoch in training, your function should evaluate the training objective, testing objective, training misclassification error rate (error is 1 for each example if misclassifies, 0 if correct), testing misclassification error rate (5 points).

4.1 Answer:

The error rate in my example was very high, this was due to hidden bugs in my code and my issues with numpy library. I had an error rate of 0.4.

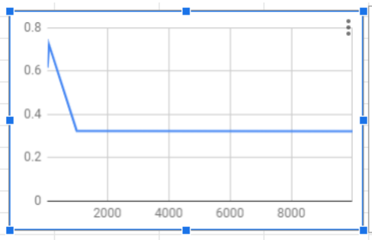
---

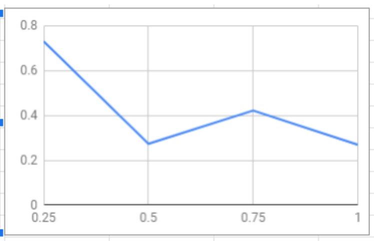
5 Question 5

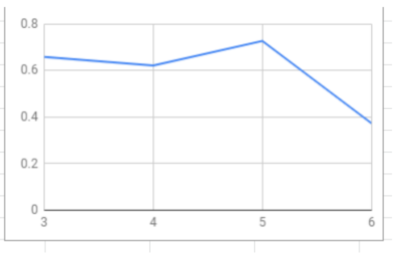
Tuning Parameters: please create three figures with following requirements. Save them into jpg format:

- Test accuracy with different number of batch size.
- Test accuracy with different learning rate.
- Test accuracy with different number of hidden units

5.1 Answer:







---

6 QUESTION 6

Discussion about the performance of your neural network.

6.1 Answer:

The Neural network did not perform very well at all. There must be hidden bugs that I could not find or figure out. I am considering re submitting this assignment later and improve my results.