Introduction to Machine Learning (Spring 2020)

Homework #3 (50 Pts, Due Date: May 31st)

Student ID 2015313754

Name 길태형

Instruction: We provide all codes and datasets in Python. Please write your code to complete activation layers (Sigmoid, ReLU, tanh), Fully Connected Layer (FCLayer), Softmax Layer, and L2 regularization. Submit two files as follows:

- 'HW3_STUDENT_ID_YourName.zip': All codes in the directory except 'data' directory and your document
- 'HW3_STUDENT_ID_YourName.pdf': Your document converted into pdf.

NOTE 1: In the next homework, 'Homework #4', you will be reusing your code from 'Homework #3'

(1) [30 pts] Implement functions in ReLU, Sigmoid, Tanh, FCLayer, SoftmaxLayer, Norm in 'Answer.py'.

(a) [Activation Layer] Implement Sigmoid, ReLU, Tanh activation in 'Answer.py' ('Sigmoid', 'ReLU', 'Tanh').

(b) [Fully Connected Layer] Implement Fully Connected layer in 'Answer.py' ('FCLayer').

(c) [Softmax Layer] Implement Softmax layer in 'Answer.py' ('SoftmaxLayer').

Given a mini-batch data $D(X, Y)$, the error function for a mini-batch is defined as follows:

$$E(w) = -\frac{1}{n} \sum_{(x_i, y_i) \in D} y_i * \log(\hat{y}_i) + \frac{1}{2} \lambda \sum_{j=1}^{k} \|w^{(j)}\|_2^2$$

$$where\ \hat{y}_i = softmax\left(w^{(k)^T} z_i^{(k)} + b^{(k)}\right),$$

$$z_i^{(j)} = \sigma\left(w^{(j)} z_i^{(j-1)} + b^{(j)}\right),\ z_i^{(0)} = x_i \quad (j = 1, 2, \dots, k)$$

$$w^{(j)}, b^{(j)}\ is\ the\ parameters\ of\ j-th\ layer\ and\ \sigma\ is\ the\ activation\ function.$$

Answer: Fill your code here. You also have to submit your code to i-campus.

```
class ReLU:

    def forward(self, z):
        out = None
        # ============================== EDIT HERE ==============================
        self.zero_mask=z<=0
        out=z
        out[self.zero_mask]=0
        # =======================================================================
        return out

    def backward(self, d_prev, reg_lambda):
        dz = None
        # ============================== EDIT HERE ==============================
        dz = d_prev
        dz[self.zero_mask]=0
        # =======================================================================
        return dz
```

```
class Sigmoid:

    def forward(self, z):
        self.out = None
        # ============================== EDIT HERE ==============================
        self.out =1/(1+np.exp(-1*z))
        # =======================================================================
        return self.out

    def backward(self, d_prev, reg_lambda):
        dz = None
        # ============================== EDIT HERE ==============================
        dz=self.out*(1-self.out)*d_prev
        # =======================================================================
        return dz
```

```python
class Tanh:
    def forward(self, z):
        self.out = None
        # ============================== EDIT HERE ==============================
        self.out = 2/(1+np.exp(-2*z))-1
        # =======================================================================
        return self.out

    def backward(self, d_prev, reg_lambda):
        dz = None
        # ============================== EDIT HERE ==============================
        dz = d_prev * (1-self.out) * (1+self.out)
        # =======================================================================
        return dz

class FCLayer:
    def forward(self, x):
        if len(x.shape) > 2:
            batch_size = x.shape[0]
            x = x.reshape(batch_size, -1)

        self.x = x
        # ============================== EDIT HERE ==============================
        self.out=np.matmul(self.x,self.W)+self.b
        # =======================================================================
        return self.out

    def backward(self, d_prev, reg_lambda):
        dx = None           # Gradient w.r.t. input x
        self.dW = None      # Gradient w.r.t. weight (self.W)
        self.db = None      # Gradient w.r.t. bias (self.b)

        # ============================== EDIT HERE ==============================
        self.db = np.sum(d_prev,axis=0)
        self.dW = np.matmul(np.transpose(self.x),d_prev)+reg_lambda*self.W
        dx = np.matmul(d_prev,np.transpose(self.W))
        # =======================================================================
        return dx
```

```python
class SoftmaxLayer:
    def forward(self, x):
        y_hat = None
        # ============================== EDIT HERE ==============================
        self.y_hat=softmax(x)
        # =======================================================================
        return self.y_hat

    def backward(self, d_prev=1, reg_lambda=0):
        batch_size = self.y.shape[0]
        dx = None
        # ============================== EDIT HERE ==============================
        dx=(self.y_hat-self.y)/batch_size
        # =======================================================================
        return dx


    def ce_loss(self, y_hat, y):
        self.loss = None
        eps = 1e-10
        self.y_hat = y_hat
        self.y = y
        # ============================== EDIT HERE ==============================
        self.loss = (-1/y_hat.shape[0])*np.sum(self.y*np.log(eps+self.y_hat))
        # =======================================================================
        return self.loss
```

NOTE 2: **You should write your codes in 'EDIT HERE' signs.** It is not recommended to edit other parts. Once you complete your implementation, run the check code ('test_answer.py') to check if it is done correctly.

NOTE 3: **Read the instructions in template codes VERY CAREFULLY.** Functionality and input, output shape of every function must be the same as written.

(2) [20 Pts] Experiment results

(a) [DNN with different activation layer] Report test accuracy on MNIST using three different activation function(Sigmoid, ReLU, Tanh) with given DNN architecture and parameters. Explain the differences among three activation functions (Use only one activation function in one experiment among Sigmoid, ReLU, Tanh)

[DNN Architecture]

Layer name	**Configuration**
FC – 1	Input dim = 784, Output dim = 500
[Sigmoid, ReLU, Tanh]	-
FC – 2	Input dim = 500, Output dim = 500
[Sigmoid, ReLU, Tanh]	-
FC – 3	Input dim = 500, Output dim = 10
Softmax Layer	-

(num_epochs = 100, learning_rate = 0.001, batch_size=128, reg_lambda = 1e-8)

	Sigmoid	**ReLU**	**Tanh**
Test accuracy	**0.6**	**0.87**	**0.89**

Sigmoid 를 사용한 신경망의 Accuracy 가 가장 낮은 값을 보였습니다. 이는, Sigmoid 의 미분값의 절대값이 최대 1/4 로, 작기 때문에 100 에폭 동안 충분한 학습을 가지지 못한것으로 보입니다. Layer 가 output Layer 로부터 멀어질수록, Gradient 가 줄어드는 Vanishing Gradient 문제도 발생할 수 있습니다.

ReLU 를 사용한 신경망의 Accuracy 는 Sigmoid 보다 개선된 Accuracy 를 보였습니다. 이는, ReLU 의 미분값은 1 로, 상위 Layer 에서 전파된 Gradient 를 Activation 의 값이 양수인 경우, 그대로 흘려보내기 때문에, Gradient 의 최대 절대값이 1/4 인 Sigmoid 보다 빨리 학습이 진행된 것으로 볼 수 있습니다. 다만, ReLU 를 Activation Function 으로 사용할 경우, 합성곱의 결과가 음수일 경우에는 하위 Layer 로 Gradient 를 흘려보내지 않기 때문에 학습이 이뤄지지 않는 문제점이 있습니다.

Tanh 는 ReLU 보다 상승된 Accuracy 를 보였는데, 이는 Layer 의 층이 깊지않기 때문에 Vanishing Gradient Problem 이 발생하지 않고, 합성곱이 양수인 Node 에 대해 gradient 를 1 로 통일하는 ReLU 와 달리, Tanh 는 0 초과 1 이하의 다양한 gradient 를 반환하기 때문에, 더 자세한 학습이 이뤄졌다고 볼 수 있습니다. 또한, 합성곱인 음수인 Node 의 뒷단에 대해서는 학습이 진행되지 않는 ReLU Network 의 문제 또한 해결했다고 볼 수 있습니다.

Sigmoid 와 비교한다면, Tanh 의 Gradient 의 범위는 0~1 로, Sigmoid 의 Gradient 범위인 0~0.25 보다 4 배 크므로, 같은 에폭 동안 더 많은 양의 학습이 이뤄졌다고 볼 수 있습니다.

(b) [Deep Neural Networks] Adjust the model settings (# of hidden layers, # of hidden nodes, # of epochs, learning rate etc.) to get the best results on FashionMNIST using 'main.py'. Report your best test accuracy with your fine-tuned hyperparameters. Show the plot of training and validation accuracy every epochs on each case and explain how you determined the model structure or parameters in 4~5 lines.

(batch size = 128)

Answer: Fill the blank in the table. Show the plot of training & validation accuracy with a brief explanation.

	Model structure	# of epochs	Learning rate	L2 lambda	Best Validation Acc.	Final Test Acc.
1st Best	**FC-1(784, 128)** **Tanh1** **FC-2(128, 32)** **Tanh2** **FC-3(32, 10)** **Softmax**	100	0.02	1e-4	0.90	0.88
2nd Best	**FC-1(784, 128)** **ReLU1** **FC-2(128, 32)** **ReLU 2** **FC-3(32, 10)** **Softmax**	100	0.02	1e-4	0.89	0.88

3rd Best	**FC-1(784, 128)** **ReLU1** **FC-2(128, 32)** **ReLU 2** **FC-3(32, 10)** **Softmax**	100	0.015	1e-6	0.89	0.88

Gradient 계산 cost가 적은 ReLU를 주로 사용했습니다. 처음 Learning rate를 0.01, lambda를 1e-8로 설정하여 50 에폭을 학습시켰을때, valid Acc가 증가 추세로 끝나서, 에폭을 100으로 올렸고, 같은 에폭동안 빠르게 모델을 학습시키기 위해서 learning rate를 0.015로 설정했고, overfitting을 방지하기 위해 reg_lambda를 1e-6으로 설정했습니다. 100에폭을 학습시킨 결과, train과 acc간의 결과가 컸기에, reg_lambda를 1e-4로 설정했고, 학습 정도를 유지하기 위해 learning rate를 0.02로 설정했습니다. 2-(a)를 참고하여, 같은 parameter에 activation function만 Tanh로 설정하여, valid Acc를 높였습니다.