**Introduction to Machine Learning (Spring 2020)**

**Homework #4 (50 Pts, Due Date : June 8)**

**Student ID 2015313754**

**Name 길태형**

**Instruction:** We provide all codes and datasets in Python. Please write your code to complete Convolutional Neural Network Classifier.

* HW4\_STUDENT\_ID\_Name.zip : compress ‘Answer.py’ and ‘main.py’
* HW4\_STUDENT\_ID\_Name.pdf : Your document converted into pdf

Please follow the submission format. We will not be held responsible for any penalty caused by non-compliance.

**(1) [40 pts]** Implement CNN Classifier in ‘Answer.py’ with the loss function as follows:

**[IMPORTANT]** In this assignment, we make **assumptions** as follows for simplicity:

* All elements of input are covered by conv/maxpool filters and all elements of conv/maxpool filters always cover the input. (see ‘check\_\*\_validity’ function in ‘Answer.py’)
  + Any cases such as (5x5) input, (2x2) kernel, stride 2, pad 0 are not allowed and don’t need to be considered. (kernel is out of bounds)
  + e.g.)
* Assume image shape is (row - column). In a maxpooling layer, if there are duplicate elements to pool, maxpool an element with priority: 1) smallest row index first, 2) smallest column index first.

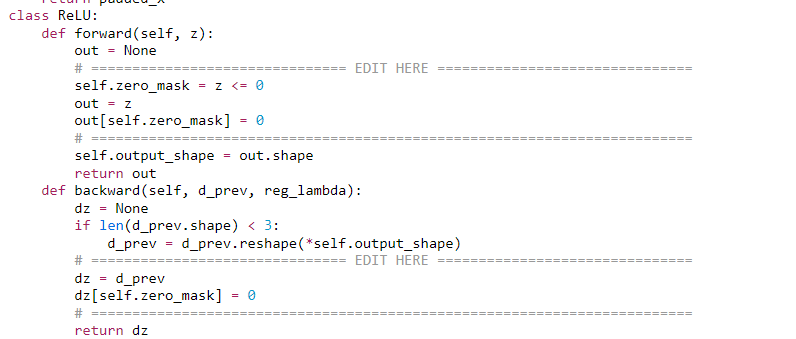
|  |  |
| --- | --- |
| **A** | **B** |
| **10** | **10** |
| **C** | **D** |
| **10** | **4** |

1. Smallest row index first: A,B,C -> A, B
2. Smallest column index first: A,B ->A
3. Pool 10 from A
4. **[Activation]** Implement ReLU, Sigmoid, Tanh activation in ‘Answer.py’ (‘ReLU’, ‘Sigmoid’, ‘Tanh’).
5. **[FC Layer & Softmax]** Implement a FC, softmax layer in ‘Answer.py’ (‘FCLayer’, ‘SoftmaxLayer’).
6. **[zero\_pad]** Implement function ‘zero\_pad’ in ‘Answer.py’.
7. **[Convolution Layer]** Implement a convolution layer in ‘Answer.py’ (‘ConvolutionLayer’).
8. **[Max-Pooling Layer]** Implement a max-pooling layer in ‘Answer.py’ (‘MaxPoolingLayer’).

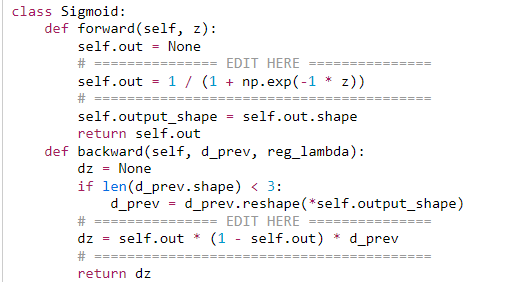
**Answer: Fill your code here. You also have to submit your code to i-campus.**

**( a )**

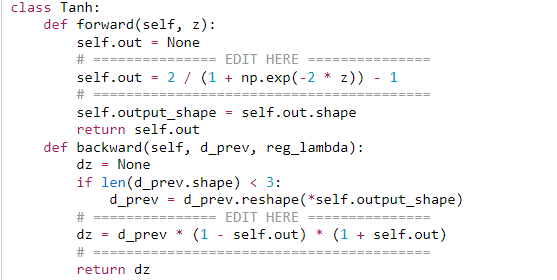
**-ReLU**



**-Sigmoid**

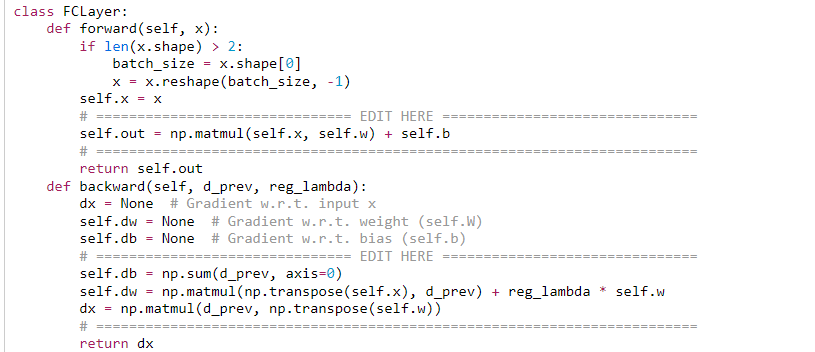


**-Tanh**



**( b )**

**-FCLayer**



**-SoftMaxLayer**

class SoftmaxLayer:

def forward(self, x):

self.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

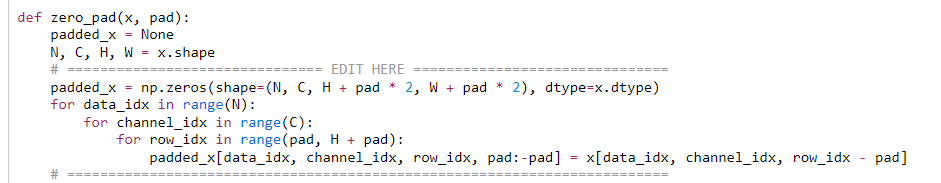
# =============================== EDIT HERE ===============================

self.loss = (-1 / y\_hat.shape[0]) \* np.sum(self.y \* np.log(eps + self.y\_hat))

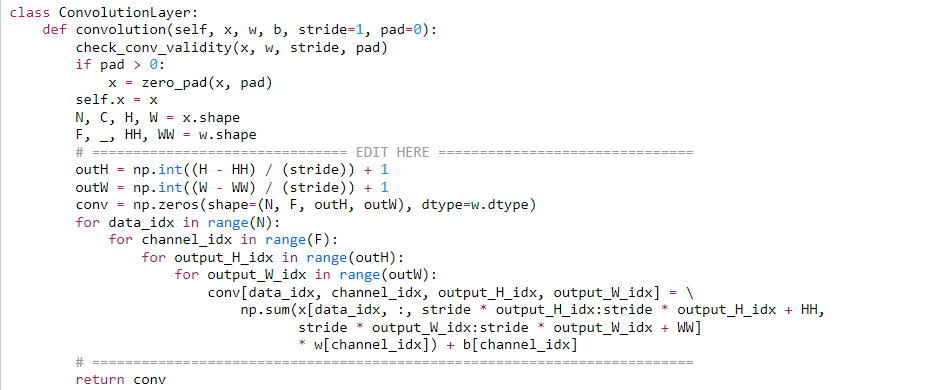
# =========================================================================

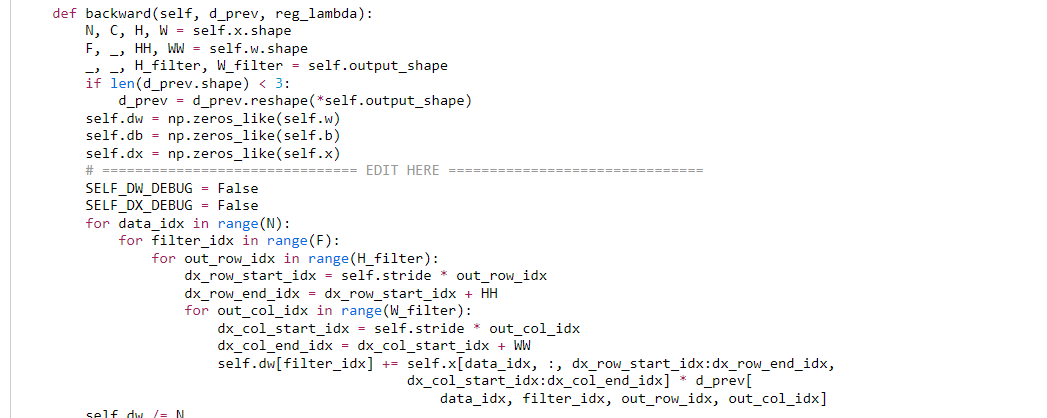
return self.loss

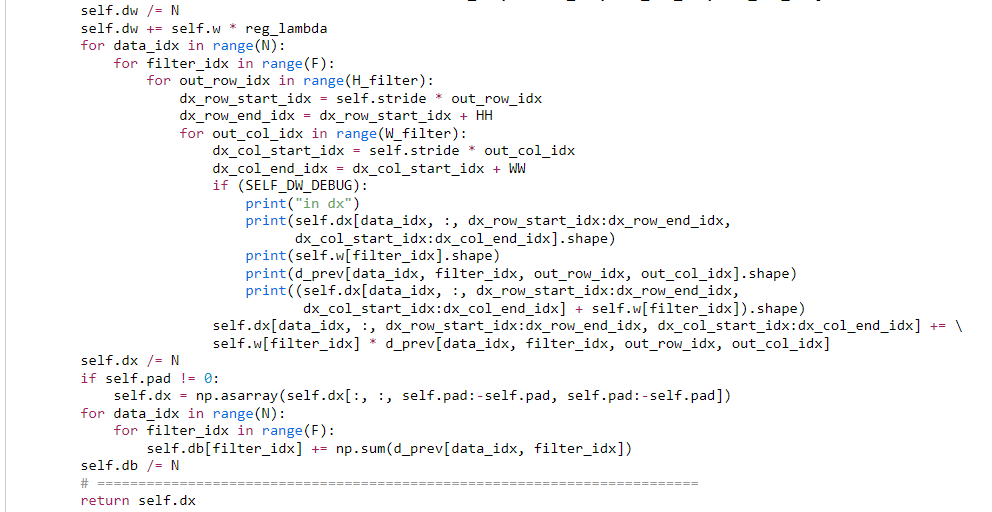
**( c ) zero pad**



**( d ) Convolution Layer**







**( e ) MaxPoolingLayer**

class MaxPoolingLayer:

def forward(self, x):

max\_pool = None

N, C, H, W = x.shape

check\_pool\_validity(x, self.kernel\_size, self.stride)

self.x = x

# =============================== EDIT HERE ===============================

self.maxpool\_mask = np.zeros\_like(self.x)

Pool\_Height = int((H - self.kernel\_size) / self.stride) + 1

Pool\_Width = int((W - self.kernel\_size) / self.stride) + 1

max\_pool = np.zeros(shape=(N, C, Pool\_Height, Pool\_Width))

for data\_idx in range(N):

for channel\_idx in range(C):

for row\_idx in range(Pool\_Height):

row\_start\_idx = self.stride \* row\_idx

row\_end\_idx = row\_start\_idx + self.kernel\_size

for col\_idx in range(Pool\_Width):

col\_start\_idx = self.stride \* col\_idx

col\_end\_idx = col\_start\_idx + self.kernel\_size

window = self.x[data\_idx, channel\_idx, row\_start\_idx:row\_end\_idx, col\_start\_idx:col\_end\_idx]

max\_val = np.finfo(dtype=np.float).min

max\_row\_idx = -1

max\_col\_idx = -1

for i in range(self.kernel\_size):

for j in range(self.kernel\_size):

if (max\_val < window[i, j]):

max\_val = window[i, j]

max\_row\_idx = i

max\_col\_idx = j

max\_pool[data\_idx, channel\_idx, row\_idx, col\_idx] = self.x[

data\_idx, channel\_idx, max\_row\_idx + row\_start\_idx, max\_col\_idx + col\_start\_idx]

self.maxpool\_mask[

data\_idx, channel\_idx, max\_row\_idx + row\_start\_idx, max\_col\_idx + col\_start\_idx] = 1

# =========================================================================

self.output\_shape = max\_pool.shape

return max\_pool

def backward(self, d\_prev, reg\_lambda):

if len(d\_prev.shape) < 3:

d\_prev = d\_prev.reshape(\*self.output\_shape)

N, C, H, W = d\_prev.shape

dx = np.zeros\_like(self.x)

# =============================== EDIT HERE ===============================

for data\_idx in range(N):

for channel\_idx in range(C):

for row\_idx in range(H):

row\_start\_idx = self.stride \* row\_idx

row\_end\_idx = row\_start\_idx + self.kernel\_size

for col\_idx in range(W):

col\_start\_idx = self.stride \* col\_idx

col\_end\_idx = col\_start\_idx + self.kernel\_size

window = self.x[data\_idx, channel\_idx, row\_start\_idx:row\_end\_idx, col\_start\_idx:col\_end\_idx]

max\_val = np.finfo(dtype=np.float).min

max\_row\_idx = -1

max\_col\_idx = -1

for i in range(self.kernel\_size):

for j in range(self.kernel\_size):

if (max\_val < window[i, j]):

max\_val = window[i, j]

max\_row\_idx = i

max\_col\_idx = j

dx[data\_idx, channel\_idx, max\_row\_idx + row\_start\_idx, max\_col\_idx + col\_start\_idx] += d\_prev[

data\_idx, channel\_idx, row\_idx, col\_idx]

# =========================================================================

return dx

**NOTE 1**: **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 codes (‘Checker.py’) to check if it is done correctly.

**NOTE 2**: **Read the instructions in template codes VERY CAREFULLY.** Functionality and input, output shape of any function must be the same as what is written.

**(2) [10 Pts]** Experiment results

1. **[Plot Graph]** You are given a MNIST dataset with 10 labels (0~9). Given CNN architecture and hyperparameters as below, build the classifier and **report train/valid accuracy of the first 3 epochs, best valid accuracy**, and **test accuracy. Draw train/valid accuracy plot – (x-axis : epoch / y-axis : accuracy).**

**Answer: Fill the blank in the table. Show the plot of train/valid accuracy with a brief explanation.**

**[CNN Architecture]**

|  |  |
| --- | --- |
| **Layer name** | **Configuration** |
| **Conv - 1** | Out Channel = 4, Kernel size = 3 Stride = 1, Pad = 1 |
| **ReLU - 1** | - |
| **Conv – 2** | Out Channel = 4, Kernel size = 3 Stride = 1, Pad = 1 |
| **ReLU - 2** | - |
| **Max-pool - 1** | Kernel size = 2, stride = 2 |
| **FC – 1** | Input dim = 784, Output dim = 500 |
| **ReLU - 3** | - |
| **FC - 2** | Input dim = 500, Output dim = 10 |
| **Softmax Layer** | - |

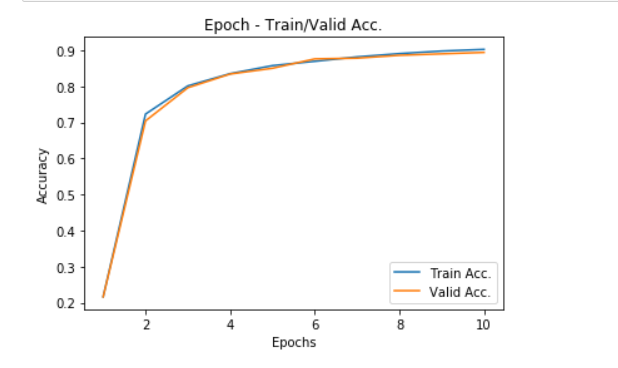
|  |  |  |  |
| --- | --- | --- | --- |
| **Epochs** | **Learning rate** | **Reg. lambda** | **Batch size** |
| 10 | 0.01 | 0.001 | 1000 |

**[Results]**

|  |  |
| --- | --- |
| **Best Valid Acc.** | **Test Acc** |
| 0.89 | 0.90 |

|  |  |  |
| --- | --- | --- |
| **Epoch** | **Train accuracy** | **Valid accuracy** |
| **1** | 0.21622222 | 0.216 |
| **2** | 0.72288889 | 0.704 |
| **3** | 0.80111111 | 0.796 |

**Plot Sample (Values are not correct. Delete when you submit)**



학습이 10에폭밖에 진행되지 않았으므로, overfitting이 진행되지 않았습니다. 따라서, train accuracy와 valid accuracy가 비슷한 값을 보여줬습니다. 초반의 비약적인 상승은, parameter가 random하게 초기화 되어있었기 때문에 성능이 안좋았지만, 학습을 통해 모델이 데이터를 잘 학습한 것으로 볼 수 있습니다.