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:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \frac{1}{2}\lambda \sum_{k=1}^{l} \|w^k\|_2^2,$$

$$L_i = -\sum_{j=1}^{C} y_j^i log p_j^i$$

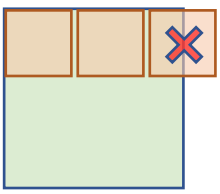
where N is the number of (batch) data, C is the number of classes,

where p_j *is the probaility of class* j *for input* i,

w *is learnable weights of layer* j *except bias term.*

[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

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

(b) [FC Layer & Softmax] Implement a FC, softmax layer in 'Answer.py' ('FCLayer', 'SoftmaxLayer').

(c) [zero_pad] Implement function 'zero_pad' in 'Answer.py'.

(d) [Convolution Layer] Implement a convolution layer in 'Answer.py' ('ConvolutionLayer').

(e) [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

```
class ReLU:
    def forward(self, z):
        out = None
        # ================================ EDIT HERE ================================
        self.zero_mask = z <= 0
        out = z
        out[self.zero_mask] = 0
        # ===========================================================================
        self.output_shape = out.shape
        return out
    def backward(self, d_prev, reg_lambda):
        dz = None
        if len(d_prev.shape) < 3:
            d_prev = d_prev.reshape(*self.output_shape)
        # ================================ EDIT HERE ================================
        dz = d_prev
        dz[self.zero_mask] = 0
        # ===========================================================================
        return dz
```

-Sigmoid

```
class Sigmoid:
    def forward(self, z):
        self.out = None
        # ============== EDIT HERE ==============
        self.out = 1 / (1 + np.exp(-1 * z))
        # =======================================
        self.output_shape = self.out.shape
        return self.out
    def backward(self, d_prev, reg_lambda):
        dz = None
        if len(d_prev.shape) < 3:
            d_prev = d_prev.reshape(*self.output_shape)
        # ============== EDIT HERE ==============
        dz = self.out * (1 - self.out) * d_prev
        # =======================================
        return dz
```

-Tanh

```
class Tanh:
    def forward(self, z):
        self.out = None
        # ============== EDIT HERE ==============
        self.out = 2 / (1 + np.exp(-2 * z)) - 1
        # =======================================
        self.output_shape = self.out.shape
        return self.out
    def backward(self, d_prev, reg_lambda):
        dz = None
        if len(d_prev.shape) < 3:
            d_prev = d_prev.reshape(*self.output_shape)
        # ============== EDIT HERE ==============
        dz = d_prev * (1 - self.out) * (1 + self.out)
        # =======================================
        return dz
```

(b)

-FCLayer

```
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
```

-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

```
def zero_pad(x, pad):
    padded_x = None
    N, C, H, W = x.shape
    # ============================== EDIT HERE ==============================
    padded_x = np.zeros(shape=(N, C, H + pad * 2, W + pad * 2), dtype=x.dtype)
    for data_idx in range(N):
        for channel_idx in range(C):
            for row_idx in range(pad, H + pad):
                padded_x[data_idx, channel_idx, row_idx, pad:-pad] = x[data_idx, channel_idx, row_idx - pad]
    # =======================================================================
```

(d) Convolution Layer

```
class ConvolutionLayer:
    def convolution(self, x, w, b, stride=1, pad=0):
        check_conv_validity(x, w, stride, pad)
        if pad > 0:
            x = zero_pad(x, pad)
        self.x = x
        N, C, H, W = x.shape
        F, _, HH, WW = w.shape
        # ================================ EDIT HERE ================================
        outH = np.int((H - HH) / (stride)) + 1
        outW = np.int((W - WW) / (stride)) + 1
        conv = np.zeros(shape=(N, F, outH, outW), dtype=w.dtype)
        for data_idx in range(N):
            for channel_idx in range(F):
                for output_H_idx in range(outH):
                    for output_W_idx in range(outW):
                        conv[data_idx, channel_idx, output_H_idx, output_W_idx] = \
                            np.sum(x[data_idx, :, stride * output_H_idx:stride * output_H_idx + HH,
                                   stride * output_W_idx:stride * output_W_idx + WW]
                                   * w[channel_idx]) + b[channel_idx]
        # ============================================================================
        return conv
```

```
    def backward(self, d_prev, reg_lambda):
        N, C, H, W = self.x.shape
        F, _, HH, WW = self.w.shape
        _, _, H_filter, W_filter = self.output_shape
        if len(d_prev.shape) < 3:
            d_prev = d_prev.reshape(*self.output_shape)
        self.dw = np.zeros_like(self.w)
        self.db = np.zeros_like(self.b)
        self.dx = np.zeros_like(self.x)
        # ================================ EDIT HERE ================================
        SELF_DW_DEBUG = False
        SELF_DX_DEBUG = False
        for data_idx in range(N):
            for filter_idx in range(F):
                for out_row_idx in range(H_filter):
                    dx_row_start_idx = self.stride * out_row_idx
                    dx_row_end_idx = dx_row_start_idx + HH
                    for out_col_idx in range(W_filter):
                        dx_col_start_idx = self.stride * out_col_idx
                        dx_col_end_idx = dx_col_start_idx + WW
                        self.dw[filter_idx] += self.x[data_idx, :, dx_row_start_idx:dx_row_end_idx,
                                                      dx_col_start_idx:dx_col_end_idx] * d_prev[
                                                          data_idx, filter_idx, out_row_idx, out_col_idx]
```

```
        self.dw /= N
        self.dw += self.w * reg_lambda
        for data_idx in range(N):
            for filter_idx in range(F):
                for out_row_idx in range(H_filter):
                    dx_row_start_idx = self.stride * out_row_idx
                    dx_row_end_idx = dx_row_start_idx + HH
                    for out_col_idx in range(W_filter):
                        dx_col_start_idx = self.stride * out_col_idx
                        dx_col_end_idx = dx_col_start_idx + WW
                        if (SELF_DW_DEBUG):
                            print("in dx")
                            print(self.dx[data_idx, :, dx_row_start_idx:dx_row_end_idx,
                                  dx_col_start_idx:dx_col_end_idx].shape)
                            print(self.w[filter_idx].shape)
                            print(d_prev[data_idx, filter_idx, out_row_idx, out_col_idx].shape)
                            print((self.dx[data_idx, :, dx_row_start_idx:dx_row_end_idx,
                                    dx_col_start_idx:dx_col_end_idx] + self.w[filter_idx]).shape)
                        self.dx[data_idx, :, dx_row_start_idx:dx_row_end_idx, dx_col_start_idx:dx_col_end_idx] += \
                        self.w[filter_idx] * d_prev[data_idx, filter_idx, out_row_idx, out_col_idx]
        self.dx /= N
        if self.pad != 0:
            self.dx = np.asarray(self.dx[:, :, self.pad:-self.pad, self.pad:-self.pad])
        for data_idx in range(N):
            for filter_idx in range(F):
                self.db[filter_idx] += np.sum(d_prev[data_idx, filter_idx])
        self.db /= N
        # ============================================================================
        return self.dx
```

(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

(a) [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	-

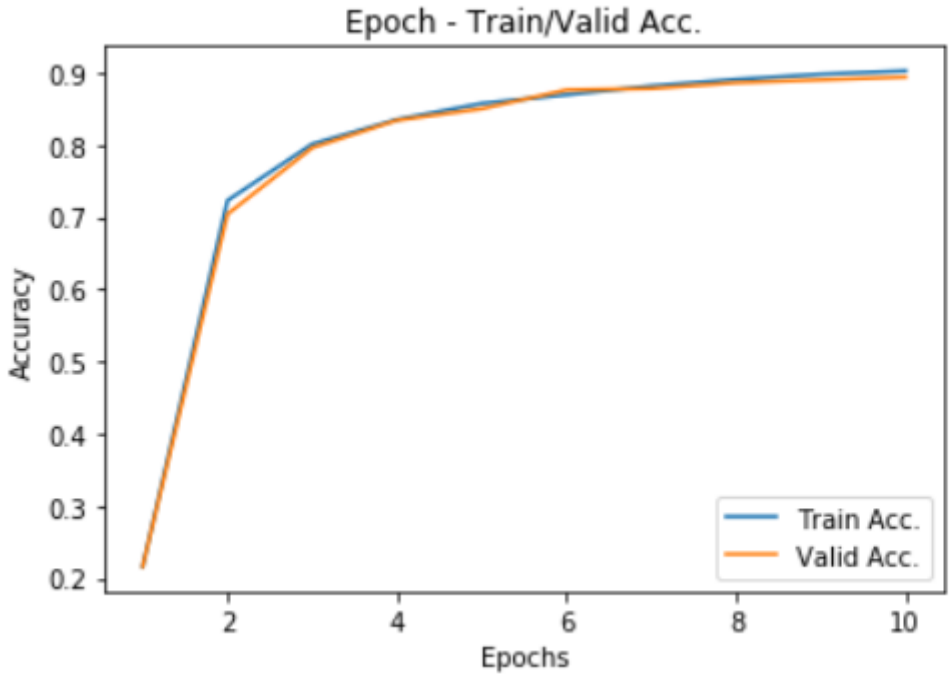
Epochs	Learning rate	Reg. lambda	Batch size
10	0.01	0.001	1000

[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.