Introduction to Machine Learning (Spring 2019)

Homework #5 (50 Pts, June 5)

Student ID 2014310407

Name 이 준 혁

Instruction: We provide all codes and datasets in Python. Please write your code to complete Convolutional Neural Network Classifier. Compress 'Answer.py' & your report ONLY and submit with the filename 'HW5_STUDENT_ID.zip'.

(1) [30 pts] Implement CNN Classifier in 'Answer.py' with the loss function as follows:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i,$$

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

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

(a) [Convolution 2D] Implement convolution function in 'Answer.py' ('convolution2d').

```
def convolution2d(x, kernel, stride):
& Conv_Width can be calculated using 'Height', 'Width', 'Kernel size', 'Stride'
    """
    height, width = x.shape
    kernel_size = kernel.shape[0]
    conv_out = None
    # ============================== EDIT HERE ==============================
    conv_height = (height - kernel_size) // stride + 1
    conv_width = (width - kernel_size) // stride + 1

    conv_out = np.zeros((conv_height, conv_width))

    for ind_x in range(conv_height):
        for ind_y in range(conv_width):
            conv_out[ind_x][ind_y] = np.sum(np.multiply(x[ind_x * stride : ind_x * stride +
kernel_size,
                                                ind_y * stride : ind_y * stride + kernel_size],
kernel))
    # ========================================================================
    return conv_out
```

(b) [ReLU] Implement ReLU activation in 'Answer.py' ('ReLU').

```
class ReLU:

    def forward(self, z):

        out = None
        # ============================== EDIT HERE ==============================
        out = z * (z > 0)
        self.zero_mask = 1 * (z > 0)
        # =======================================================================
        return out

    def backward(self, d_prev):

        # ============================== EDIT HERE ==============================
        dz = np.multiply(d_prev, self.zero_mask)
        # =======================================
```

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

```
class ConvolutionLayer:
    def convolution(self, x, kernel, bias=None, stride=1, pad=0):
        """

        batch_size, in_channel, _, _ = x.shape

        if pad > 0:
            x = self.zero_pad(x, pad)

        _, _, height, width = x.shape
        out_channel, _, kernel_size, _ = kernel.shape
        assert x.shape[1] == kernel.shape[1]

        conv = None
        # ============================== EDIT HERE ==============================

        conv_height = (height - kernel_size) // stride + 1
        conv_width = (width - kernel_size) // stride + 1

        conv = np.zeros((batch_size, out_channel, conv_height, conv_width))

        for batch in range(batch_size):
            for out in range(out_channel):
                for inp in range(self.W.shape[1]):
                    conv[batch,out,:,:] += convolution2d(x[batch,inp,:,:],
                                                         self.W[out,inp,:,:], self.stride)
                    if bias is not None:
                        conv[batch, out, :, :] += self.b[out]

        # =======================================================================
        return conv
```

```
def backward(self, d_prev):

    batch_size, in_channel, height, width = self.x.shape
    out_channel, _, kernel_size, _ = self.W.shape

    if len(d_prev.shape) < 3:
        d_prev = d_prev.reshape(*self.output_shape)

    self.dW = np.zeros_like(self.W, dtype=np.float64)
    self.db = np.zeros_like(self.b, dtype=np.float64)
    dx = np.zeros_like(self.x, dtype=np.float64)
    # ============================== EDIT HERE ==============================

    # db
    self.db = np.sum(d_prev, axis = (0,2,3))

    # dx, dW

    d_prev_expand = self.zero_pad(d_prev, pad = self.kernel_size - 1)
    W_rotate = np.rot90(self.W, 2, axes = (2, 3))

    for batch in range(batch_size):
        for out in range(out_channel):
            for inp in range(in_channel):
                self.dW[out, inp] += convolution2d(self.x[batch, inp],
                  d_prev[batch, out], self.stride)
                if self.pad:
                    dx[batch, inp] += convolution2d(d_prev_expand[batch, out],
                  W_rotate[out, inp],  self.stride)[self.pad:-self.pad, self.pad:-self.pad]
                else:
                    dx[batch, inp] += convolution2d(d_prev_expand[batch, out],
                  W_rotate[out, inp], self.stride)

    # ================================================================
    return dx

def zero_pad(self, x, pad):

    batch_size, in_channel, height, width = x.shape
    # ============================== EDIT HERE ==============================
    padded_x = np.zeros((batch_size, in_channel, height + 2 * pad, width + 2 * pad))
    padded_x[:,:,pad:height+pad,pad:width+pad] = x[:,:,:,:]
    # ================================================================
    return padded_x
```

(d) [Max-Pooling Layer] Implement a max-pooling layer in 'Answer.py' ('MaxPoolingLayer').

```
class MaxPoolingLayer:

    def forward(self, x):
            max_pool = None
        batch_size, channel, height, width = x.shape

        self.mask = np.zeros_like(x)
        # ============================ EDIT HERE ============================
        pool_height = (height - self.kernel_size) // self.stride + 1
        pool_width = (width - self.kernel_size) // self.stride + 1

        max_pool = np.zeros((batch_size, channel, pool_height, pool_width))

        for batch in range(batch_size):
            for chan in range(channel):
                for ind_x in range(pool_height):
                    for ind_y in range(pool_width):
                        offset = np.unravel_index(np.argmax(x[batch, chan, ind_x * self.stride :
                         ind_x * self.stride + self.kernel_size, ind_y * self.stride : ind_y *
                         self.stride + self.kernel_size]), (self.kernel_size, self.kernel_size))

                        index = np.array([ind_x* self.stride,ind_y* self.stride])
                        index += [offset[0], offset[1]]

                        max_pool[batch, chan, ind_x, ind_y] = x[batch, chan, index[0], index[1]]
                        self.mask[batch, chan, index[0], index[1]] = 1

        # ===================================================================
        self.output_shape = max_pool.shape
        return max_pool

    def backward(self, d_prev=1):

        d_max = None
        if len(d_prev.shape) < 3:
            d_prev = d_prev.reshape(*self.output_shape)
        batch, channel, height, width = d_prev.shape
        # ============================ EDIT HERE ============================
        d_prev_expand = np.zeros_like(self.mask)

        for bat in range(batch):
            for chan in range(channel):
                for h in range(height):
                    for w in range(width):
                        d_prev_expand[bat, chan, h * self.stride : h * self.stride +
                         self.kernel_size, w * self.stride : w * self.stride + self.kernel_size]
                         = d_prev[bat, chan, h, w]


        d_max = np.multiply(d_prev_expand, self.mask)
        # ===================================================================
        return d_max
```

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

```
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(x, self.W) + self.b
        # =======================================================================
        return self.out

    def backward(self, d_prev):

        self.dW = np.zeros_like(self.W, dtype=np.float64)   # Gradient w.r.t. weight (self.W)
        self.db = np.zeros_like(self.b, dtype=np.float64)   # Gradient w.r.t. bias (self.b)
        dx = np.zeros_like(self.x, dtype=np.float64)        # Gradient w.r.t. input x
        # ============================== EDIT HERE ==============================

        self.dW = np.matmul(np.transpose(self.x), d_prev)
        self.db = np.sum(d_prev, axis=0)

        dx = np.transpose(np.matmul(self.W, np.transpose(d_prev)))
        # =======================================================================
        return dx

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):

        batch_size = self.y.shape[0]
        dx = None
        # ============================== EDIT HERE ==============================
        loss_grad = np.divide(self.y_hat - self.y, batch_size)
        dx = np.multiply(d_prev, loss_grad)
        # =======================================================================
        return dx
```

```python
def ce_loss(self, y_hat, y):
    self.loss = None
    eps = 1e-10
    self.y_hat = y_hat
    self.y = y
    # ============================== EDIT HERE ==============================
    batch_size = y.shape[0]

    pred_log = np.log(y_hat + eps)
    self.loss = np.sum(np.multiply(y, -pred_log))
    self.loss /= batch_size
    # =======================================================================
    return self.loss
```

(2) [20 Pts] Experiment results

(a) you are given a small MNIST dataset with 5 labels (0, 1, 2, 3, 4), which originally has 10 labels. Given CNN architecture and hyperparameters as below, build the classifier and adjust hyperparameters to achieve best test accuracy. (Your best accuracy should be at least 0.8 if the model is trained correctly.)

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

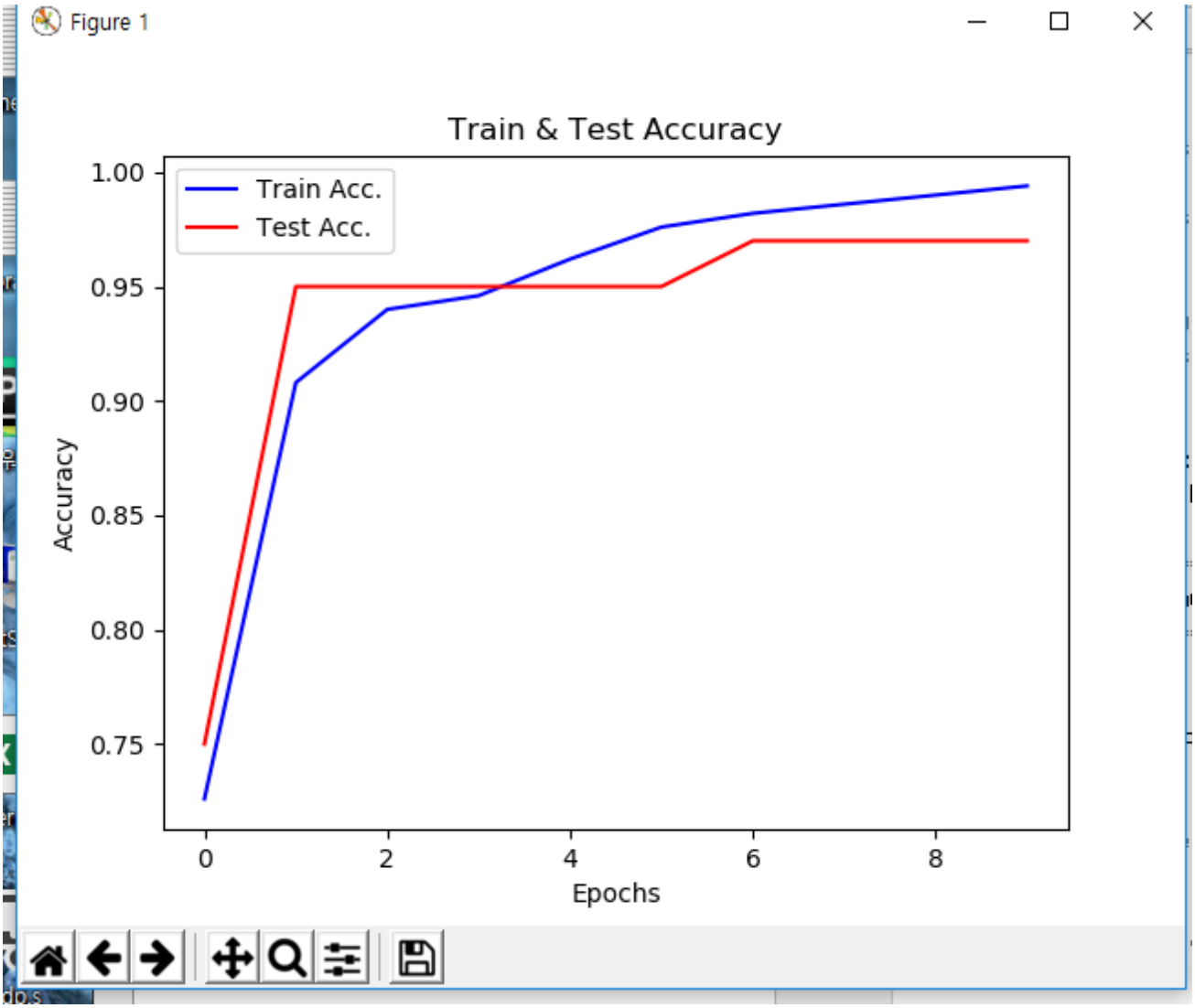
[CNN Architecture]

Layer name	Configuration
Conv - 1	Out Channel = 8, Kernel size = 3 Stride = 1, Pad = 1
ReLU - 1	-
Conv – 2	Out Channel = 8, Kernel size = 3 Stride = 1, Pad = 1
ReLU - 2	-
Max-pool - 1	Kernel size = 2, stride = 2
FC – 1	Input dim = 1568, Output dim = 500
FC - 2	Input dim = 500, Output dim = 5
Softmax Layer	-

[Results]

Epochs	Learning rate	Best Acc.	Best Epoch.
10	0.01	0.97	7

Plot Sample



다음과 같이 training accuracy와 test accuracy가 같이 증가함을 확인하면서 training이 잘 된 다는 사실을 확인할 수 있다. 다만 어떠한 regularization도 주지 않았기 때문에, training accuracy는 증가하지만 test accuracy는 더 이상 오르지 않는, 일종의 overfitting이 약간 생김을 확인할 수 있었다.