ITE4053 Deep Learning course
Practice 2

2016026080 Dajin Han

- **Experiment environments**

OS	macOS Catalina
Memory	16GB
Language	python3
Tools	Jupyter notebook

- **Update Practice2 code**

The code used in Practice2 was modified and structured to create a network with a Layer classes. With Network class, you can create a network with multiple layer easily. You can create it by just putting together a list of layer when creating network object. Now Network class can have a layer that can manage stride option.

■ Network Class

```python

# # 2. Interface of neural network
class Network:

    def __init__(self, layers):
        self.layers = layers
        self.inputs = ['tmp' for i in layers]

    def forward(self, X):
        input_mat = X
        for idx, layer in enumerate(self.layers):
            self.inputs[idx]=input_mat
            input_mat = layer.forward(input_mat)

        return input_mat

    def backward(self, dx_next, learning_rate):
        # dx_next == da_now (x->z->a)
        for layer, input_mat in zip(reversed(self.layers), reversed(self.inputs)):
            dx_next=layer.backward(input_mat, dx_next, learning_rate)

    def train(self, X, y, learning_rate):
        pred_y = self.forward(X)
        dy_of_dx = -y / pred_y + (1 - y) / (1 - pred_y)
        self.backward(dy_of_dx, learning_rate)

    def predict(self, X):
        return np.round(self.forward(X))

    def loss(self, X, y):
        pred_y = self.forward(X)
        return -np.mean(y * np.log(pred_y) + (1 - y) * np.log(1 - pred_y))

    def print_layers(self):
        for layer_idx, layer in enumerate(self.layers):
            print('Layer #{}.  input : {}\t kernel : {}\t output : {}'.format(layer_idx+1, layer.INPUT_DIM, layer.KERNEL_DIM, layer.OUTPUT_DIM))

```

I made Forward(), Backward() functions operate continuously about layers inside the network object. We can check the inference time using predict function and can check training time using train function. Print_layer function is made to make sure created network has accurate layers inside.

■ **Layer Class**

```python

# # 1. Layer
class Layer:

    def __init__(self, INPUT_DIM=(1,2,1), kernel=(1,2,1,1), stride=(1,1), OUTPUT_DIM=(1,1,1)):
        self.INPUT_DIM = INPUT_DIM                                  # INPUT_DIM  == (row_i, column_i, channel_i)
        self.OUTPUT_DIM = OUTPUT_DIM                                # OUTPUT_DIM == (row_o, column_o, channel_o)
        self.KERNEL_DIM = kernel                                    # kernel     == (row_k, column_k, channel_i, channel_o)
        self.stride = stride                                        # stride     == (row_s, column_s)

#         self.w = np.zeros(self.KERNEL_DIM, dtype=np.float64)       # w_dim      == (row_k, column_k, channel_i) * channel_o
#         self.b = np.zeros(self.OUTPUT_DIM[-1], dtype=np.float64)   # b_dim      == 1 * channel_o
        self.w = np.random.uniform(-1, 1, self.KERNEL_DIM)          # w_dim      == (row_k, column_k, channel_i) * channel_o
        self.b = np.random.uniform(-1, 1, self.OUTPUT_DIM[-1])      # b_dim      == 1 * channel_o

    def forward(self, X):
        MIN_MARGIN = 2 ** -53

        for row_idx in range(self.OUTPUT_DIM[0]):
            row_s = row_idx * self.stride[0]
            row_e = row_s + self.KERNEL_DIM[0]
            for column_idx in range(self.OUTPUT_DIM[1]):
                column_s = column_idx * self.stride[1]
                column_e = column_s + self.KERNEL_DIM[1]
                for channel_o_idx in range(self.OUTPUT_DIM[2]):

                    tmp_z = self.w[:,:,:,channel_o_idx] * X[:, row_s:row_e, column_s:column_e, :]  #       (row_k, column_k, channel_i, 1)
                                                                                                    # * (-1, row_k, column_k, channel_i)
                                                                                                    # = (-1, row_k, column_k, channel_i)
                    tmp_z = np.sum(tmp_z, axis=(1,2,3)) + self.b[channel_o_idx]                     # (-1, 1)
                    if len(tmp_z.shape)==1:
                        tmp_z=tmp_z.reshape(-1,1)
                    if row_idx == 0 and column_idx == 0 and channel_o_idx == 0:
                        self.z = tmp_z
                    else:
                        self.z = np.concatenate((self.z, tmp_z), axis=1)


        self.z = self.z.reshape((-1,)+self.OUTPUT_DIM)                          # (-1, row_o , column_o , channel_o)


        self.a = 1 / (1 + np.exp(-self.z))                                      # (-1, row_o , column_o , channel_o)
        self.a = np.maximum(np.minimum(1 - MIN_MARGIN, self.a), MIN_MARGIN)     # (-1, row_o , column_o , channel_o)
        return self.a                                                           # (-1, row_o , column_o , channel_o)


    def backward(self, X, dx_next, learning_rate):
        # x_next == a_now
        # dx_next == da_now
        # dx_next == d_loss / dx_next == d_loss / da_now
        # == da
        da = dx_next                                                            # (-1, row_o , column_o , channel_o)
        dz = self.a * (1 - self.a) * da                                         # (-1, row_o , column_o , channel_o)
        dx = np.zeros(X.shape, dtype=np.float64)                                # (-1, row_i , column_i , channel_i)
        # check each "filters" in total kernel!
        for channel_o_idx in range(self.OUTPUT_DIM[2]):
            for row_idx in range(self.OUTPUT_DIM[0]):
                row_s = row_idx * self.stride[0]
                row_e = row_s + self.KERNEL_DIM[0]
                for column_idx in range(self.OUTPUT_DIM[1]):
                    column_s = column_idx * self.stride[1]
                    column_e = column_s + self.KERNEL_DIM[1]

                    # set dw
                    X_selected = X[:, row_s:row_e, column_s:column_e, :]                         # (-1, row_k, column_k, channel_i)
                    dz_selected = dz[:, row_idx, column_idx, channel_o_idx].reshape(-1,1,1,1)    # (-1,     1,        1,         1)

                    if row_idx == 0 and column_idx == 0:
                        dw = X_selected * dz_selected                                            #   (-1, row_k, column_k, channel_i)
                    else:                                                                        # * (-1,     1,        1,         1)
                        dw += X_selected * dz_selected                                           # = (-1, row_k, column_k, channel_i)

                    # set dx
                    w_selected = self.w[:,:,:,channel_o_idx]                                     #     (row_k, column_k, channel_i, 1)
                    dx[:, row_s:row_e, column_s:column_e, :] += dz_selected * w_selected         # (-1, row_k, column_k, channel_i, 1)

            dw = np.mean(dw, axis=0)                                                             # (1, row_k, column_k, channel_i)
            db = np.mean(np.sum(dz[:,:,:,channel_o_idx], axis=(1,2)), axis=0)                    # (1, 1)

            # update weights of current "filter" in total kernel
            # current "filter" == channel_th filter of kernel
            self.w[:,:,:,channel_o_idx] -= learning_rate * dw
            self.b[channel_o_idx] -= learning_rate * db

        dx = dx.reshape((-1,)+self.INPUT_DIM)                                   # (-1, row_i, column_i, channel_i)
        return dx                                                               # (-1, row_i, column_i, channel_i)

```

In Practice_2, I coded the program that perform backpropagation in neural network, with simple input shape. Shape of input is simple, It was (2,1). But in Practice_3 I made Layer class that can manage every shape of nodes. I used np.prod()+np.sum() combination instead of np.dot() because it is easier to create code. But maybe it takes more time. So I plan to convert the code of np.prod() type to the code of np.dot() type in next assignment.

- ## Practice_3

■ Framework

1. Create Samples

```python
# # 3. Train Model
def create_samples_mul(sample_num):
    X = np.random.uniform(-2, 2, (sample_num, 2))
    y = (X[:,0]*X[:,0] > X[:,1]).astype(float)
    return X, y
```

2. Make Network

```python
model = nn.Network(
    layers = [
        nn.Layer(INPUT_DIM=(1,1,2), kernel=(1,1,2,1), stride=(1,1), OUTPUT_DIM=(1,1,1)),
    ]
)
model = nn.Network(
    layers = [
        nn.Layer(INPUT_DIM=(1,1,2), kernel=(1,1,2,1), stride=(1,1), OUTPUT_DIM=(1,1,1)),
        nn.Layer(INPUT_DIM=(1,1,1), kernel=(1,1,1,1), stride=(1,1), OUTPUT_DIM=(1,1,1)),
    ]
)
model = nn.Network(
    layers = [
        nn.Layer(INPUT_DIM=(1,1,2), kernel=(1,1,2,3), stride=(1,1), OUTPUT_DIM=(1,1,3)),
        nn.Layer(INPUT_DIM=(1,1,3), kernel=(1,1,3,1), stride=(1,1), OUTPUT_DIM=(1,1,1)),
    ]
)
```

3. Run

```python
for epoch in range(epochs):
    model.train(train_X, train_y, learning_rate)
```

■ Result

◆ Task1

```
(py37) dajinhan@Dajinui-MacBookPro src % python task1.py
train_samples :  1000 (1000, 1, 1, 2)
test_samples  :  100 (1000, 1, 1, 1)
epochs        :  1000
learning_rate :  2.0
Layer #1.  input : (1, 1, 2)    kernel : (1, 1, 2, 1)   output : (1, 1, 1)

Result
w0 : [[ 0.01240477 -1.81908519]]
b0 : [1.96309279]
train_loss : 0.3646168421868392
test_loss : 0.42397818520041225
train_acc : 78.8
test_acc : 76.0
time_gen_data : 0.0001029968261718 75
time_gen_network : 6.318092346191406e-05
time_train : 0.12581682205200195
time_predict : 4.291534423828125e-05
```

- Task2

```
(py37) dajinhan@Dajinui-MacBookPro src % python task2.py
train_samples :  1000 (1000, 1, 1, 2)
test_samples  :  100 (1000, 1, 1, 1)
epochs        :  1000
learning_rate :  2.0
Layer #1.  input : (1, 1, 2)     kernel : (1, 1, 2, 1)   output : (1, 1, 1)
Layer #2.  input : (1, 1, 1)     kernel : (1, 1, 1, 1)   output : (1, 1, 1)

Result
w0 : [[ 0.4506131  -5.45470348]]
b0 : [-0.05487192]
w1 : [[7.54804055]]
b1 : [-0.37571145]
train_loss : 0.32652805454192346
test_loss : 0.367468997427564
train_acc : 81.0
test_acc : 77.0
time_gen_data : 9.799003601074219e-05
time_gen_network : 7.486343383789062e-05
time_train : 0.2170090675354004
time_predict : 6.914138793945312e-05
```

- Task3

```
(py37) dajinhan@Dajinui-MacBookPro src % python task3.py
train_samples :  1000 (1000, 1, 1, 2)
test_samples  :  100 (1000, 1, 1, 1)
epochs        :  1000
learning_rate :  2.0
Layer #1.  input : (1, 1, 2)     kernel : (1, 1, 2, 3)   output : (1, 1, 3)
Layer #2.  input : (1, 1, 3)     kernel : (1, 1, 3, 1)   output : (1, 1, 1)

Result
w0 : [[-6.70119471  7.04424811]
 [ 0.31519597 -3.38269263]
 [-3.28112508 -6.47573989]]
b0 : [[-3.15152542]
 [-3.5862235 ]
 [ 0.64664444]]
w1 : [[10.31693809 10.16086042  6.90681025]]
b1 : [-5.24633751]
train_loss : 0.030728213627777275
test_loss : 0.02920473777890654
train_acc : 99.8
test_acc : 100.0
time_gen_data : 0.00010800361633300781
time_gen_network : 7.82012939453125e-05
time_train : 0.4577038288116455
time_predict : 0.00010180473327636719
```

- Comparison

	Task #1	Task #2	Task #3
Accuracy_train	0.788	0.810	0.998
Accuracy_test	0.760	0.770	1.000
Time_train	0.126	0.217	0.458
Time_test	0.00004	0.00007	0.0001

- What I learned

The process of figuring out how the interior of the Convolution layer was constructed was beneficial. Especially, the time to think about how to construct the internal backpropagation formula was helpful for me.
And when I first initialized the weight value in layer to zero, the accuracy was just 60%. But when I initialed it randomly, the accuracy increased to more than 90%. Some articles explain it is because of the activation function sigmoid. Initial distribution of weight value can cause the local minimum. Maybe I need to study more about this.