Spring 2021 Introduction to Deep Learning

Homework Assignment 2

Name: Yanhan Zhang   NetID: yz963

Problem1:

(a) The inputs I choose are: w1=0.7854 (pi/4), x1=1, w2=-0.7854(-pi/4), x2=1.
The computational graph and the calculated forward values are shown as follow.

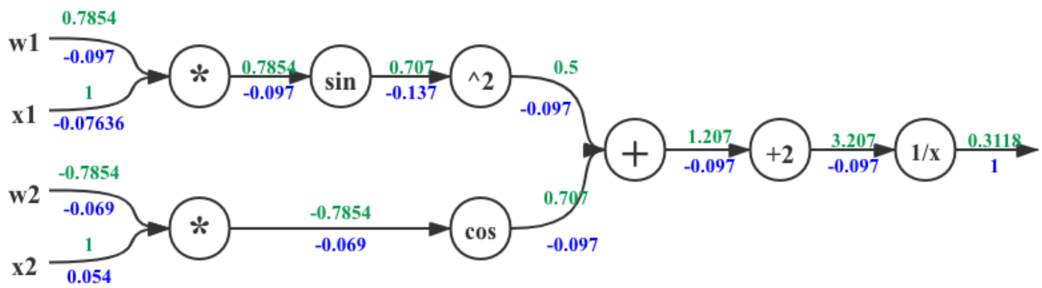

Figure 1. The computational graph of problem1.

Forward:

$$f(x,w) = \frac{1}{2+\sin^2(x_1 w_1) + \cos(x_2 w_2)}$$

$$= \frac{1}{2+\sin^2(1*0.7854) + \cos(1*(-0.7854))}$$

$$= \frac{1}{2+\sin^2(0.7854) + \cos(-0.7854)} = \frac{1}{2+\left(\frac{\sqrt{2}}{2}\right)^2 + \frac{\sqrt{2}}{2}}$$

$$= \frac{1}{2+\frac{1}{2}+\frac{\sqrt{2}}{2}} \approx \frac{1}{3.207} \approx 0.3118$$

Backward:

$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial(2+\sin^2(x_1w_1)+\cos(x_2w_2))}\frac{\partial(2+\sin^2(x_1w_1)+\cos\,(x_2w_2))}{\partial w_1}$$

$$= -\frac{1}{(2+\sin^2(x_1w_1)+\cos(x_2w_2))^2}\frac{\partial(\sin^2(x_1w_1)+\cos(x_2w_2))}{\partial w_1}$$

$$= -\frac{1}{(3.207)^2}\frac{\partial(\sin^2(x_1w_1)+\cos(x_2w_2))}{\partial w_1}$$

$$= -0.097 * \frac{\partial(\sin^2(x_1w_1))}{\partial w_1}$$

$$= -0.097 * 2\sin(x1w1) * \frac{\partial(\sin(x_1w_1))}{\partial w_1}$$

$$= -0.097 * 2\sin(0.7854) * \cos(x_1w_1) * \frac{\partial(\mathrm{x}_1w_1)}{\partial w_1}$$

$$= -0.097 * \sqrt{2} * \cos(0.7854) * x_1 = -0.097$$

$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial(2+\sin^2(x_1w_1)+\cos(x_2w_2))}\frac{\partial(2+\sin^2(x_1w_1)+\cos\,(x_2w_2))}{\partial x_1}$$

$$= -\frac{1}{(2+\sin^2(x_1w_1)+\cos(x_2w_2))^2}\frac{\partial(\sin^2(x_1w_1)+\cos(x_2w_2))}{\partial x_1}$$

$$= -\frac{1}{(3.207)^2}\frac{\partial(\sin^2(x_1w_1)+\cos(x_2w_2))}{\partial x_1}$$

$$= -0.097 * \frac{\partial(\sin^2(x_1w_1))}{\partial x_1}$$

$$= -0.097 * 2\sin(x1w1) * \frac{\partial(\sin(x_1w_1))}{\partial x_1}$$

$$= -0.097 * 2\sin(0.7854) * \cos(x_1w_1) * \frac{\partial(\mathrm{x}_1w_1)}{\partial x_1}$$

$$= -0.097 * \sqrt{2} * \cos(0.7854) * w_1 = -0.07636$$

$$\frac{\partial f}{\partial w_2} = \frac{\partial f}{\partial(2+\sin^2(x_1w_1)+\cos(x_2w_2))}\frac{\partial(2+\sin^2(x_1w_1)+\cos\,(x_2w_2))}{\partial w_2}$$

$$= -\frac{1}{(2+\sin^2(x_1w_1)+\cos(x_2w_2))^2}\frac{\partial(\sin^2(x_1w_1)+\cos(x_2w_2))}{\partial w_2}$$

$$= -\frac{1}{(3.207)^2}\frac{\partial(\sin^2(x_1w_1)+\cos(x_2w_2))}{\partial w_2}$$

$$= -0.097 * \frac{\partial(\cos(x_2w_2))}{\partial w_2} = -0.097 * (-\sin(x_2w_2)) * \frac{\partial(x_2w_2)}{\partial w_2}$$

$$= 0.097 * \sin(-0.7854) * x_2 = 0.097 * \left(-\frac{\sqrt{2}}{2}\right) * 1 = -0.069$$

$$
\frac{\partial f}{\partial x_2} = \frac{\partial f}{\partial(2+\sin^2(x_1 w_1)+\cos(x_2 w_2))}\frac{\partial(2+\sin^2(x_1 w_1)+\cos\,(x_2 w_2))}{\partial x_2}
$$

$$
= -\frac{1}{(2+\sin^2(x_1 w_1)+\cos(x_2 w_2))^2}\frac{\partial(\sin^2(x_1 w_1)+\cos(x_2 w_2))}{\partial x_2}
$$

$$
= -\frac{1}{(3.207)^2}\frac{\partial(\sin^2(x_1 w_1)+\cos(x_2 w_2))}{\partial x_2}
$$

$$
= -0.097 * \frac{\partial(\cos(x_2 w_2))}{\partial x_2} = -0.097 * (-\sin(x_2 w_2)) * \frac{\partial(x_2 w_2)}{\partial x_2}
$$

$$
= 0.097 * \sin(-0.7854) * w_2 = 0.097 * \left(-\frac{\sqrt{2}}{2}\right) * (-0.7854)
$$

$$
= 0.054
$$

(b) The program is more than 100 lines and you can get access to it by click here.
The result of my program is shown as follow. It's values are the same with (a).

```
w1=0.7853981633974483, x1=1, w2=-0.7853981633974483, x2=1
Forward:
Inputs: [0.7853981633974483, 1]; Type: multiply; Forward result: 0.7853981633974483.
Inputs: [-0.7853981633974483, 1]; Type: multiply; Forward result: -0.7853981633974483.
Input: 0.7853981633974483; Type: sin; Forward result: 0.7071067811865475.
Input: -0.7853981633974483; Type: cos; Forward result: 0.7071067811865476.
Input: 0.7071067811865475; Type: square; Forward result: 0.4999999999999999.
Inputs: (0.4999999999999999, 0.7071067811865476); Type: add; Forward result: 1.2071067811865475.
Input: 1.2071067811865475; Type: add2; Forward result: 3.2071067811865475.
Input: 3.2071067811865475; Type: reciprocal; Forward result: 0.31180751631538306.

Backward:
Input: 3.2071067811865475; Gradient: 1; Type: reciprocal; Backward result: -0.09722392723076786.
Input: 1.2071067811865475; Gradient: -0.09722392723076786; Type: add2; Backward result: -0.09722392723076786.
Inputs: (0.4999999999999999, 0.7071067811865476); Gradient: -0.09722392723076786; Type: add; Backward result: (-0.09722392723076786, -0.09722392723076786).
Input: 0.7071067811865475; Gradient: -0.09722392723076786; Type: square; Backward result: -0.13749539647692677.
Input: 0.7853981633974483; Gradient: -0.13749539647692677; Type: sin; Backward result: -0.09722392723076786.
Input: -0.7853981633974483; Gradient: -0.09722392723076786; Type: cos; Backward result: -0.06874769823846338.
Inputs: [0.7853981633974483, 1]; Gradient: -0.09722392723076786; Type: multiply; Backward result: (-0.09722392723076786, -0.07635949388533224).
Inputs: [-0.7853981633974483, 1]; Gradient: -0.06874769823846338; Type: multiply; Backward result: (-0.06874769823846338, 0.05399431593429113).
```

Figure 2. The running result of my program for problem 1.

Problem2:

(a) The inputs I choose are shown in the following computational graph and the calculation procedures are easy to know.
The computational graph and the calculated forward values are shown as follow.

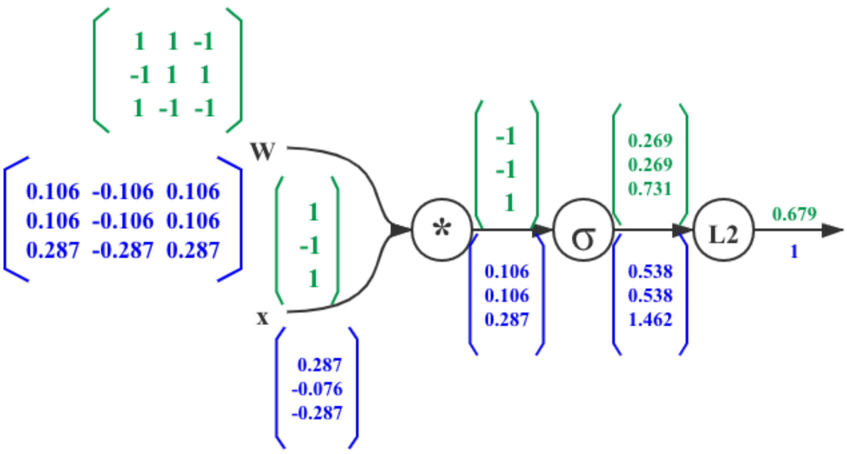

Figure 3. The computational graph of problem2.

(b) I have begun to implement an AI framework of my own in several weeks. A layer called Dense has already been programmed and simple perceptron-like models are able to be built. The calculation in this problem can be done by using my own framework. The core codes used to calculate *W*x*, *sigmoid* and l2 loss are shown as follow. The codes of the total framework can be found in [this GitHub url](#).

```python
class Layer:
    def __init__(self, input_layers=[], output_shape=0, use_bias=False, weights_initializer=np.zeros,
                 bias_initializer=np.zeros):
        self.input_shapes = list(map(lambda l: l.get_output_shape(), input_layers))
        # self.input_layers = input_layers
        self.input_layers = dict(map(lambda t: (t[1], t[0]), enumerate(input_layers)))
        self.output_shape = output_shape
        self.output_layers = {}
        self.cur_inputs = [[]] * len(input_layers)
        self.cur_inputs_ready_flags = set()
        self.cur_deltas_ready_flags = set()
        self.cur_deltas = []
        self.cur_outputs = []
        self.use_bias = use_bias
        # pend
        self.starts = set()
        if input_layers:
            list(map(lambda layer: self.starts.update(layer.get_starts()), input_layers))
        else:
            self.starts.add(self)
        list(map(lambda layer: layer.append_output_layer(self), input_layers))

    def get_output_shape(self):
        return self.output_shape

    def get_output_layers(self):
        return self.output_layers.keys()

    def get_input_shape(self):
        return self.input_shapes

    def get_input_layers(self):
        return self.input_layers.keys()

```

Figure 4. Layer (base class of Dense) (part 1)

```python
    def get_starts(self):
        return self.starts

    def set_cur_input(self, _layer_ref, values):
        self.cur_inputs[self.input_layers[_layer_ref]] = values
        self.cur_inputs_ready_flags.add(self.input_layers[_layer_ref])

    def append_cur_delta(self, _layer_ref, values):
        self.cur_deltas.append(values)
        self.cur_deltas_ready_flags.add(self.output_layers[_layer_ref])

    def append_output_layer(self, _out_layer):
        # self.output_layers.append(_out_layer)
        self.output_layers[_out_layer] = len(self.output_layers)

    def clear_cur_inputs_flags(self):
        self.cur_inputs_ready_flags = set()

    def clear_cur_deltas_flags(self):
        self.cur_deltas_ready_flags = set()
        self.cur_deltas = []

    def forward(self):
        raise NotImplementedError

    def backward(self):
        raise NotImplementedError

    @property
    def can_forward(self):
        if len(self.cur_inputs_ready_flags) == len(self.input_layers):
            return True
        return False

    @property
    def can_backward(self):
        if len(self.cur_deltas_ready_flags) == len(self.output_layers):
            return True
        return False
```

Figure 5. Layer (base class of Dense) (part 2)

```python
class Dense(Layer):
    def __init__(self, input_layers, output_shape, activation=None, use_bias=False,
                 weights_initializer=np.random.standard_normal,
                 bias_initializer=np.random.standard_normal):
        super().__init__(input_layers, output_shape, use_bias, weights_initializer, bias_initializer)
        self.weights = weights_initializer((self.input_shapes[0], self.output_shape))
        self.delta = np.zeros((self.input_shapes[0], self.output_shape))
        self.activation = activation
        if use_bias:
            self.weights = np.append(self.weights, [bias_initializer(output_shape)], axis=0)
            self.delta = np.append(self.delta, [np.zeros(output_shape)], axis=0)

    def forward(self):
        if self.use_bias:
            self.cur_inputs[0] = np.append(self.cur_inputs[0], np.ones((len(self.cur_inputs[0]), 1)), axis=1)
        self.cur_outputs = np.matmul(self.cur_inputs[0], self.weights)
        print('Forward values:\n{}'.format(np.array(self.cur_outputs)))
        if self.activation:
            self.before_activation = np.copy(self.cur_outputs)
            self.cur_outputs = self.activation(self.cur_outputs, Directions.forward)
            print('Activated:\n{}'.format(np.array(self.cur_outputs)))
        list(map(lambda ol: ol.set_cur_input(self, self.cur_outputs), self.output_layers.keys()))
        self.clear_cur_inputs_flags()

    def backward(self):
        # todo: filter inputs which are actually no need to go
        mean_inputs = np.array([np.mean(self.cur_inputs[0], axis=0)])
        self.delta = np.sum(self.cur_deltas, axis=0)
        if self.activation:
            print('Backward gradients:\n{}'.format(np.transpose(-self.delta)))
            self.delta = self.activation(self.before_activation, Directions.backward, self.delta)
        print('Current gradients: \n{}'.format(np.transpose(-np.matmul(np.transpose(mean_inputs), self.delta))))
        backward_delta = np.matmul(self.delta, np.transpose(self.weights))
        print('Backward gradients:\n{}'.format(np.transpose(-self.delta)))
        if self.use_bias:
            backward_delta = backward_delta[:, :-1]
        list(map(lambda layer: layer.append_cur_delta(self, backward_delta), self.input_layers))
        self.weights += np.matmul(np.transpose(mean_inputs), self.delta)
        self.clear_cur_deltas_flags()
```

Figure 6. Dense class

```python
def sigmoid(z, direction, back_grad=0):
    def forward(z_):
        return 1 / (1 + np.exp(-z_))

    def backward(z_, back_grad_):
        return back_grad_ * forward(z_) * (1 - forward(z_))

    return forward(z) if direction == Directions.forward else backward(z, back_grad)
```

Figure 7. Sigmoid activation function

```python
import numpy as np


def l2_loss(y_true, y_pred, learning_rate):
    # print('yt: {}, yp: {}'.format(y_true, y_pred))
    loss_value = 0.5 * np.sum(np.square(y_pred-y_true)) / y_true.shape[0]
    grad = np.mean(y_true - y_pred, axis=0) * learning_rate
    # print('lv: {}, g: {}'.format(loss_value, grad))
    return 2 * loss_value, 2 * grad
```

Figure 8. l2 loss

The codes using my framework to solve this problem is shown as follow.

```python
def ass2_p2(lr, epochs):
    il = Input(1)
    dl1 = Dense([il], 3, weights_initializer=np.ones)
    dl1.weights = np.array([[1, -1, 1]], dtype=np.float)
    dl2 = Dense([dl1], 3, weights_initializer=np.zeros, activation=sigmoid)
    dl2.weights = np.array([[1, 1, -1], [-1, 1, 1], [1, -1, -1]], dtype=np.float).transpose()
    ol = Output([dl2], 3, loss_function=l2_loss, learning_rate=lr)
    inputs = np.ones([1, 1])
    outputs = np.zeros([1, 3])
    sess = Session([ol], inputs, outputs)
    sess.train(epochs)
    return sess
```

Figure 9. model codes

The calculation results are shown as follow.

```
Python 3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 13:42:17)
In[2]: from samples.olp_reg_1d import ass2_p2
In[3]: sess = ass2_p2(1, 1)
Forward values:
[[ 1. -1.  1.]]
Forward values:
[[-1. -1.  1.]]
Activated:
[[0.26894142 0.26894142 0.73105858]]
Backward gradients:
[[0.53788284]
 [0.53788284]
 [1.46211716]]
Current gradients:
[[ 0.10575419 -0.10575419  0.10575419]
 [ 0.10575419 -0.10575419  0.10575419]
 [ 0.28746968 -0.28746968  0.28746968]]
Backward gradients:
[[0.10575419]
 [0.10575419]
 [0.28746968]]
Current gradients:
[[ 0.28746968]
 [-0.07596131]
 [-0.28746968]]
Backward gradients:
[[ 0.28746968]
 [-0.07596131]
 [-0.28746968]]
epoch: 0; loss: 0.6791056216455496.
```

Figure 10. The running result of my program for problem 1.

The results are the same with my results in (a).