Homework 8

Wendi Chen

1 Q1

1.1 Is deeper better in Deep Learning?

The answer is **Yes**. According to Universality Theorom, any continuous function can be realized by a network with one hidden layer if enough hidden neurons is given. However, the number of the neurons is limited, so how we arrange them matters. According to the study mentioned in the keynotes[1], a shallow 1-hidden-layer network using the same number of parameters as the 7-hidden-layer one leads to five percentage points worse word-error rate. Besides, a deeper network does better in modularization. That means the network can have some "basic classifiers" which works as module for the following classifiers so that the classifiers can be trained by little data.

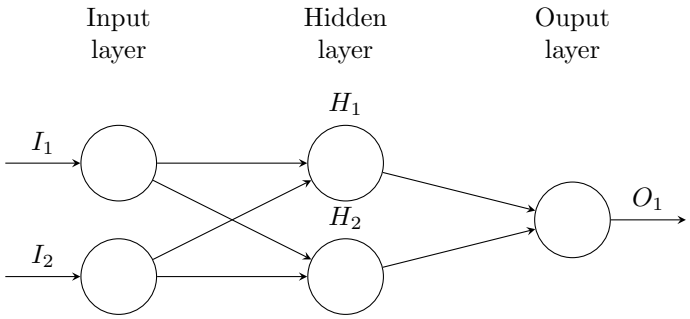
1.2 What parameters/designs/structures should be carefully concerned to obtain high performance in Deep Learning?

- connection between layers
- number of neurons in each layer & number of layers
- activation function & loss function
- optimization method(gradient descent, SGD, momentum, etc.)
- learning rate/adaptive learning rate
- batch division
- training times
- dropout

[1]Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." Interspeech. 2011.

2 Q2

The structure of the Neural Network is shown as follow. The network uses ReLU as the activation function, SGD as optimization method and L1-loss as loss function.



I have implemented it in PyTorch.

```
import torch
import torch.nn as nn
import numpy as np

#data
x = np.array([[0,0],[1,1],[1,0],[0,1]])
y = np.array([[0],[0],[1],[1]])
x = torch.Tensor(x).float()
y = torch.Tensor(y).float()

#network design
class XOR(nn.Module):
    def __init__(self):
        super(XOR, self).__init__()
        self.input_layer = nn.Linear(2,2)
        self.relu = nn.ReLU()
        self.output_layer = nn.Linear(2,1)

    def forward(self,x):
        o1 = self.relu(self.input_layer(x))
        o2 = self.relu(self.output_layer(o1))
        return o2

xor = XOR()
loss_function = nn.L1Loss()
optimizer = torch.optim.SGD(xor.parameters(),
            lr=1e-3, momentum=0.9)


```

```
#train
for epoch in range(5000):
    out = xor(x)
    loss = loss_function(out,y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

#test
out = xor(x)
print(out)
```

As the test result listed below, the training result is not bad.

```
  tensor([[0.0000],
          [0.0000],
          [1.0005],
          [0.9990]], grad_fn=<ReluBackward0>)
```

3 Q3

For question **a**, $y_1 = 4$ and $y_2 = 0$.
For question **b**, $y_1 = 0$ and $y_2 = 1.125$.