TD 2: Deep Neural Network by PyTorch

1. Objective

In this exercise, we will implement Deep Neural Network by using PyTorch framework. In the first part, you will work on an example to see how to implement a deep neural network (two hidden layers) by PyTorch. In the second part, you will implement another deep neural network with more hidden layers.

After this exercise, you will:

- Have an overview about PyTorch framework and its components
- How to convert data from numpy to PyTorch component
- Know how to implement the Deep Neural Network by using PyTorch

2. Software and libraries

Packages require:

- PyTorch library
- Torchvision
- matplotlib

3. Problem

In this exercise, we solve the same problem with the previous exercise (binary classification) from data (h5 files).

4. Review

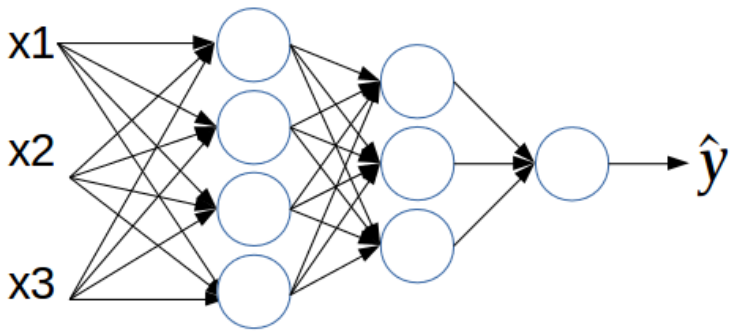

Figure 1: The architecture of two hidden layers neural network

In this exercise, you will build a 3-layers neural network as Figure 1. The network includes 3 layers (2 hidden layers). The hidden units are 4 and 3 for the hidden layers, respectively. The final layer has one node which present the score of prediction (0 or 1). The process is as following:

- Load dataset
- Define the network architecture
- Define the loss function
- Define the update rule to update parameters values
- Train the model
- Test the trained model by predicting a (or several) test image(s)

4.1. Load dataset

Question 1: Implement the function ***load_data()*** to load the data from h5 file. Then, convert them into the input vector to input the network. **Hints**: You can use numpy functions to finish this work.

```
import numpy as np
import matplotlib.pyplot as plt
import h5py

train_path = 'data/train_catvnoncat.h5'
test_path = 'data/test_catvnoncat.h5'

def load_data(train_path, test_path):
    train_dataset = h5py.File(train_path,'r')
    train_set_X = np.array(train_dataset['train_set_x'][:])
    train_set_Y = np.array(train_dataset['train_set_y'][:])

    test_dataset = h5py.File(test_path,'r')
    test_set_X = np.array(test_dataset['test_set_x'][:])
    test_set_Y = np.array(test_dataset['test_set_y'][:])

    classes = np.array(test_dataset['list_classes'][:])
    return train_set_X, train_set_Y, test_set_X, test_set_Y, classes

def reshape_data(x_dataset, y_dataset):
```

```
    x_dataset_reshape = x_dataset.reshape((x_dataset.shape[0], x_dataset
.shape[1] * x_dataset.shape[2] * x_dataset.shape[3]))
    y_dataset_reshape = y_dataset.reshape((y_dataset.shape[0],-1))
    return x_dataset_reshape, y_dataset_reshape
```

Question 2: Implement ***convert_to_tensor()*** function to convert the data from the numpy matrix to PyTorch Tensor. **Hints**: Using DataLoader in PyTorch.

```
import torch
import torch.utils.data

def normalize(imgs):
    return imgs / 255.

def convert_to_tensor(numpy_array):
    data = normalize(numpy_array)
    return torch.from_numpy(data)

train_x_dataset = convert_to_tensor(train_set_X_reshape)
train_x_dataset = train_x_dataset.float()
train_y_dataset = torch.from_numpy(train_set_Y_reshape)
test_x_dataset = convert_to_tensor(test_set_X_reshape)
test_x_dataset = test_x_dataset.float()
test_y_dataset = torch.from_numpy(test_set_Y_reshape)

# show an example
img = test_x_dataset[0].view((3,64,64))
imshow(img)

b_size = 4
train = torch.utils.data.TensorDataset(train_x_dataset,train_y_dataset)

train_loader = torch.utils.data.DataLoader(train, batch_size=b_size, shu
ffle=True)

test = torch.utils.data.TensorDataset(test_x_dataset,test_y_dataset)
test_loader = torch.utils.data.DataLoader(test, batch_size=b_size, shuff
le=True)
```

4.2. Network architecture

Question 3: Implement your network model (including forward function) by inherited **nn.Module**.

Hints: We use **ReLU** activation function for all hidden layers and **Sigmoid** function for the last layer.

```
import torch.nn as nn
import torch.nn.functional as F
```

```
3.
4.  class Network(nn.Module):
5.      def __init__(self):
6.          super(Network, self).__init__()
7.          self.layer1 = nn.Linear(12288,4)
8.          self.layer2 = nn.Linear(4,3)
9.          self.layer3 = nn.Linear(3,1)
10.     def forward(self, x):
11.         x = F.relu(self.layer1(x))
12.         x = F.relu(self.layer2(x))
13.         x = F.sigmoid(self.layer3(x))
14.         return x
15. net = Network()
16. print(net)
```

Question 4: Define the loss function. **Hints**: We use cross entropy loss (**CrossEntropyLoss**).

```
1.  criterion = nn.CrossEntropyLoss()
```

Question 5: Define the update rule to update the parameters of the model. **Hints**: We use Stochastic Gradient Descent with learning rate = 0.1 and momentum = 0.9.

```
1.  import torch.optim as optim
2.  optimizer = optim.SGD(net.parameters(), lr = 0.1, momentum = 0.9)
```

4.3. Training

Question 6: Write the ***train()*** function to train the network in 100 epochs.

```
1.  def train(train_loader, epochs):
2.      for epoch in range(epochs):
3.          running_loss = 0.0
4.          for i, data in enumerate(train_loader,0):
5.              inputs, labels = data
6.              #print (torch.max(labels,1)[1])
7.              #clear the parameter gradients
8.              optimizer.zero_grad()
9.
10.             # forward + backward + optimize
11.             outputs = net(inputs)
12.             #print (outputs)
13.             loss = criterion(outputs, torch.max(labels,1)[1])
14.             loss.backward()
15.             optimizer.step()
16.
17.             running_loss += loss.item()
18.             if i % 50 == 49:    # print every 50 mini-batches
```

```
                print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1, runnin
    g_loss / 50))
                running_loss = 0.0
    print('Finished training!')

epochs = 100
train(train_loader,epochs)
```

4.4. Predict the test images

Question 7: Implement the ***predict()*** function to predict a test image.

```
def predict(test_loader):
    dataiter = iter(test_loader)
    images, labels = dataiter.next()
    print (images.shape, labels.shape)
    images_batch = images.view((b_size,3,64,64))
    grid = torchvision.utils.make_grid(images_batch)
    print(grid.shape)
    imshow(grid)

    outputs = net(images)
    print(outputs)

predict(test_loader)
```

5. Exercises

In this section, you will implement a deep neural network (4 hidden layers) to solve the same problem. Your model is as below:

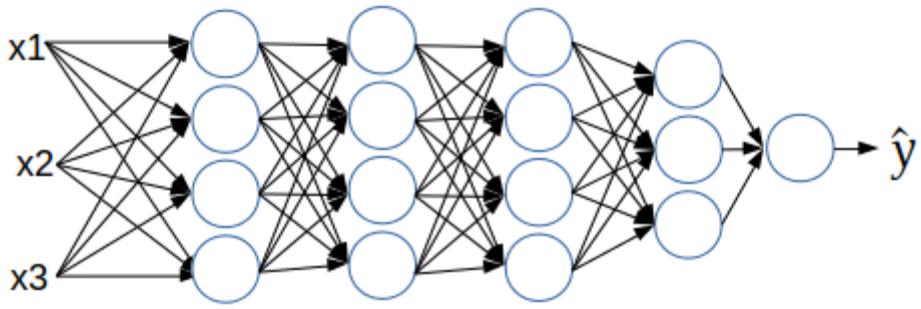

Figure 2 The architecture of a deep neural network

Exercise 1: Implement your network model (including forward function) by inherited ***nn.Module***. **Hints**: We use ***ReLU*** activation function for all hidden layers and ***Sigmoid*** function for the last layer.

Exercise 2: Define the loss function and update rule to update the parameters of the model.

Hints: You can try to use CrossEntropyLoss and SGD first, then change to other kinds of loss functions and update rules.

Exercise 3: Write the ***train()*** function to train the network in 100 epochs (or 1000 epochs)

Exercise 4: Implement the ***predict()*** function to predict a test image.

Exercise 5: Try to train and test your network by using GPU for computing.