Homework 6

Author: Boqian Wang

Question 1

X1, X2 and y are the randomly simulated data that we need.

In [ ]:
```
import random
import numpy as np

X1 = np.zeros(1000)
X2 = np.zeros(1000)
y = np.zeros(1000)

for i in range(1000):
    X1[i] = random.uniform(0, 1)
    X2[i] = random.uniform(0, 1)
    error = random.gauss(0, 0.1)
    y[i] = X1[i]**2 + X2[i]**2 + error

X = np.stack((X1, X2), axis=1)
```

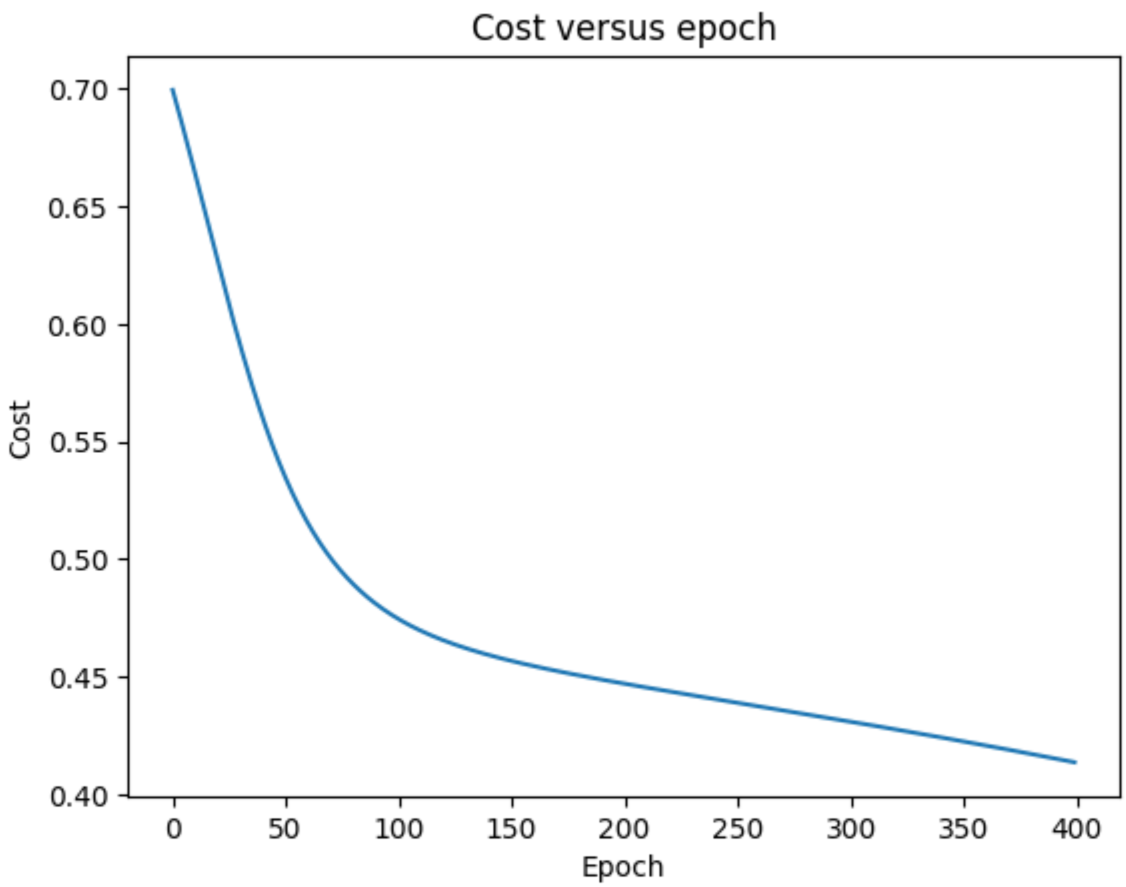
Question 2

In [ ]:
```
import numpy as np
def sigmoid(x):
    return (1 / (1 + np.exp(-x)))

def setParameters(X, Y, hidden_size):
  np.random.seed(3)
  input_size = X.shape[0] # number of neurons in input layer
  output_size = Y.shape[0] # number of neurons in output layer.
  W1 = np.random.randn(hidden_size, input_size)*np.sqrt(2/input_size)
  b1 = np.zeros((hidden_size, 1))
  W2 = np.random.randn(output_size, hidden_size)*np.sqrt(2/hidden_size)
  b2 = np.zeros((output_size, 1))
  return {'W1': W1, 'W2': W2, 'b1': b1, 'b2': b2}

def forwardPropagation(X, params):
  Z1 = np.dot(params['W1'], X)+params['b1']
  A1 = np.tanh(Z1)
  Z2 = np.dot(params['W2'], A1)+params['b2']
  y = sigmoid(Z2)
  return y, {'Z1': Z1, 'Z2': Z2, 'A1': A1, 'y': y}

def cost(predict, actual):
  m = actual.shape[1]
  cost__ = np.sqrt(np.sum(np.square(predict - actual)) / m) # Use rmse rather than
  return np.squeeze(cost__)

def backPropagation(X, Y, params, cache):
  m = X.shape[1]
  dy = cache['y'] - Y
  dW2 = (1 / m) * np.dot(dy, np.transpose(cache['A1']))
  db2 = (1 / m) * np.sum(dy, axis=1, keepdims=True)
  dZ1 = np.dot(np.transpose(params['W2']), dy) * (1-np.power(cache['A1'], 2))
  dW1 = (1 / m) * np.dot(dZ1, np.transpose(X))
  db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)
  return {"dW1": dW1, "db1": db1, "dW2": dW2, "db2": db2}

def updateParameters(gradients, params, learning_rate = 0.02):
    W1 = params['W1'] - learning_rate * gradients['dW1']
    b1 = params['b1'] - learning_rate * gradients['db1']
    W2 = params['W2'] - learning_rate * gradients['dW2']
    b2 = params['b2'] - learning_rate * gradients['db2']
    return {'W1': W1, 'W2': W2, 'b1': b1, 'b2': b2}

def fit(X, Y, learning_rate, hidden_size, number_of_iterations = 5000):
  params = setParameters(X, Y, hidden_size)
  cost_ = []
  for j in range(number_of_iterations):
    y, cache = forwardPropagation(X, params)
    costit = cost(y, Y)
    gradients = backPropagation(X, Y, params, cache)
    params = updateParameters(gradients, params, learning_rate)
    cost_.append(costit)
  return params, cost_

X, Y = X.T, y.reshape(1, y.shape[0])
params, cost  = fit(X, Y, 0.02, 5, 400)
```

Question 3

In [ ]:
```
import matplotlib.pyplot as plt
plt.title("Cost versus epoch")
plt.xlabel("Epoch")
plt.ylabel("Cost")
plt.plot(cost_)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x214554c11d0>]
```



Question 4

In [ ]:
```
import os
import torch
from torch import nn
from torch.utils.data import DataLoader
```

In [ ]:
```
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")
```

```
Using cpu device
```

In [ ]:
```python
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(2, 5),
            nn.ReLU(),
            nn.Linear(5, 5),
            nn.ReLU(),
            nn.Linear(5, 1),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

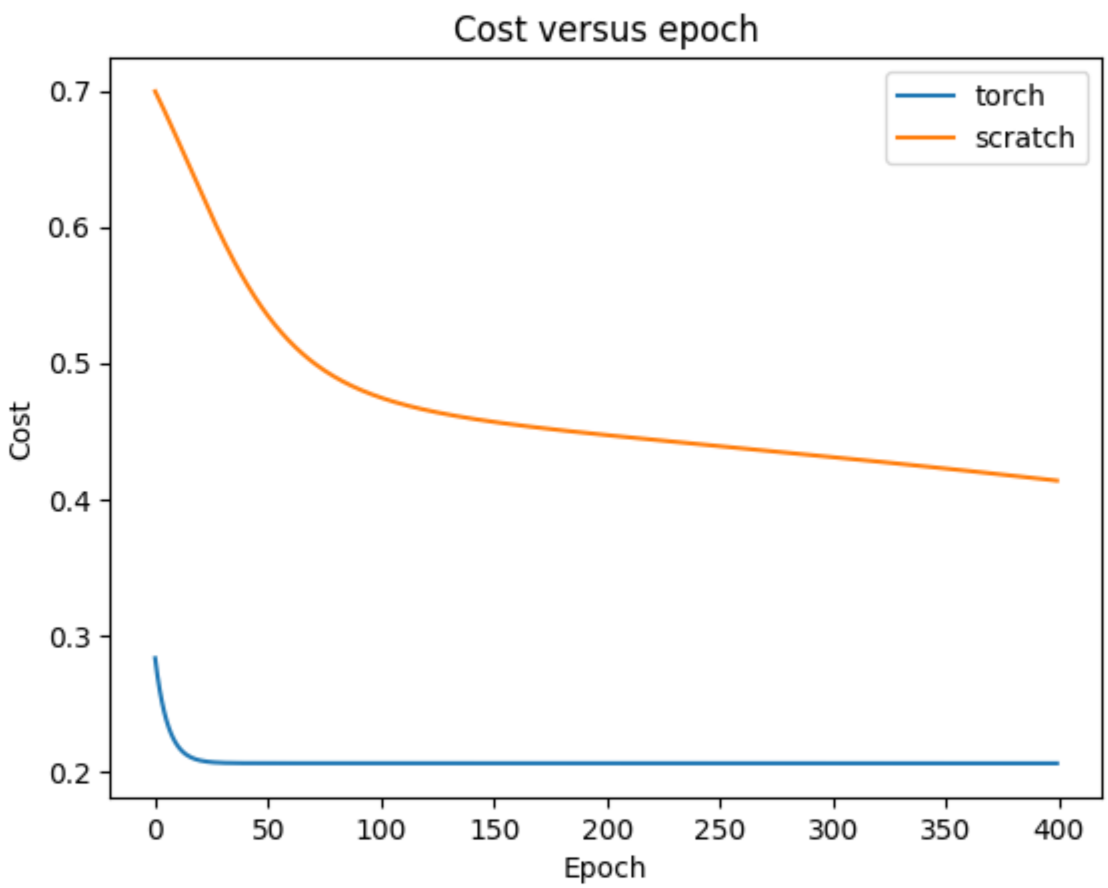
In [ ]:
```python
model = NeuralNetwork().to(device)
print(model)
```

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=2, out_features=5, bias=True)
    (1): ReLU()
    (2): Linear(in_features=5, out_features=5, bias=True)
    (3): ReLU()
    (4): Linear(in_features=5, out_features=1, bias=True)
  )
)
```

In [ ]:
```python
import torch.optim as optim
losses = []
X = X.T
X = torch.Tensor(X)
y = torch.Tensor(y)
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.02)
num_epochs = 400
for epoch in range(num_epochs):
    outputs = model(X)
    loss = criterion(outputs, y)
    losses.append(loss.item())
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
C:\Users\王柏谦\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfr
a8p0\LocalCache\local-packages\Python311\site-packages\torch\nn\modules\loss.py:535:
UserWarning: Using a target size (torch.Size([1000])) that is different to the input
size (torch.Size([1000, 1])). This will likely lead to incorrect results due to broa
dcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
```

In [ ]:
```
plt.plot(losses, label='torch')
plt.plot(cost_, label='scratch')
plt.title("Cost versus epoch")
plt.xlabel("Epoch")
plt.ylabel("Cost")
plt.legend()
plt.show()
```



Reference

Scratch Part: https://www.kaggle.com/code/ihalil95/building-two-layer-neural-networks-from-scratch

Torch Part: https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html

Special Thanks for chatgpt!