Deep Learning (Lab)

Lab 1

윤 준 영 (202252001)

1. Train a **linear regression model** using the first 300 samples of real estate price prediction data below. Then, test the regression model on the remaining data.

   Model: Linear

   Optimizer: Adam

   Learning rate: 0.01

   [code]

```
import csv
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchmetrics import R2Score

def split(csv_reader):
    x_train = []
    y_train = []
    x_test = []
    y_test = []
    for i, row in enumerate(csv_reader):
        if i == 0:
            continue
        if i <= 300:
            x_train.append(np.array(row[1:-1], dtype='float'))
            y_train.append(np.array([row[-1]], dtype='float'))
        else:
            x_test.append(np.array(row[1:-1], dtype='float'))
            y_test.append(np.array([row[-1]], dtype='float'))

    x_train = torch.tensor(np.stack(x_train), dtype=torch.float)
    y_train = torch.tensor(np.stack(y_train), dtype=torch.float)
    x_test = torch.tensor(np.stack(x_test), dtype=torch.float)
    y_test = torch.tensor(np.stack(y_test), dtype=torch.float)

    return x_train, y_train, x_test, y_test

def main():
    torch.manual_seed(1)

    with open('/share/DLL/Lab1/Real_estate.csv', newline='') as csvfile:
        csv_reader = csv.reader(csvfile, delimiter=',')
        x_train, y_train, x_test, y_test = split(csv_reader)
        model = nn.Linear(6, 1)
#       optimizer = torch.optim.SGD(model.parameters(), lr=1e-7)
        optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)

        nb_epochs = 2000
        for epoch in range(nb_epochs):
            prediction = model(x_train)
            cost = F.mse_loss(prediction, y_train)
            optimizer.zero_grad()
            cost.backward()
            optimizer.step()

            if epoch % 100 == 0:
                print('Epoch {:4d}/{} Cost: {:.6f}'.format(epoch, nb_epochs, cost.item()))

        with torch.no_grad():
            prediction = model(x_test)
            cost = F.mse_loss(prediction, y_test)
            print('\nCost: ', cost.item())
            r2score = R2Score(num_outputs=1)
            r2 = r2score(prediction, y_test)
            print('R2 score from Pytorch: ', r2.item())

if __name__ == '__main__':
    main()
```

[Result]

```
root@d1a98fca2ad2:~/lab1# python linear_regression.py
Epoch    0/2000 Cost: 109061.718750
Epoch  100/2000 Cost: 100.518723
Epoch  200/2000 Cost: 99.411560
Epoch  300/2000 Cost: 98.795990
Epoch  400/2000 Cost: 98.165184
Epoch  500/2000 Cost: 97.544685
Epoch  600/2000 Cost: 96.942627
Epoch  700/2000 Cost: 96.359932
Epoch  800/2000 Cost: 95.796097
Epoch  900/2000 Cost: 95.251945
Epoch 1000/2000 Cost: 94.729996
Epoch 1100/2000 Cost: 94.233803
Epoch 1200/2000 Cost: 93.767303
Epoch 1300/2000 Cost: 93.333992
Epoch 1400/2000 Cost: 92.936562
Epoch 1500/2000 Cost: 92.576729
Epoch 1600/2000 Cost: 92.255196
Epoch 1700/2000 Cost: 91.971687
Epoch 1800/2000 Cost: 91.725052
Epoch 1900/2000 Cost: 91.513489

Cost:  71.41546630859375
R2 score from Pytorch:  0.5622537136077881
root@d1a98fca2ad2:~/lab1#
root@d1a98fca2ad2:~/lab1#
root@d1a98fca2ad2:~/lab1#
root@d1a98fca2ad2:~/lab1#
```

Cost (MSE Loss): 71.42

R2 Score: 56.23 %

2. Train a **logistic regression model** which discriminates number 1 and 2 from digits dataset. Use the code below to access to the data. Show the loss function decreasing and the accuracy for the test samples.

   Model: Linear, Sigmoid

   Optimizer: Stochastic Gradient Descent

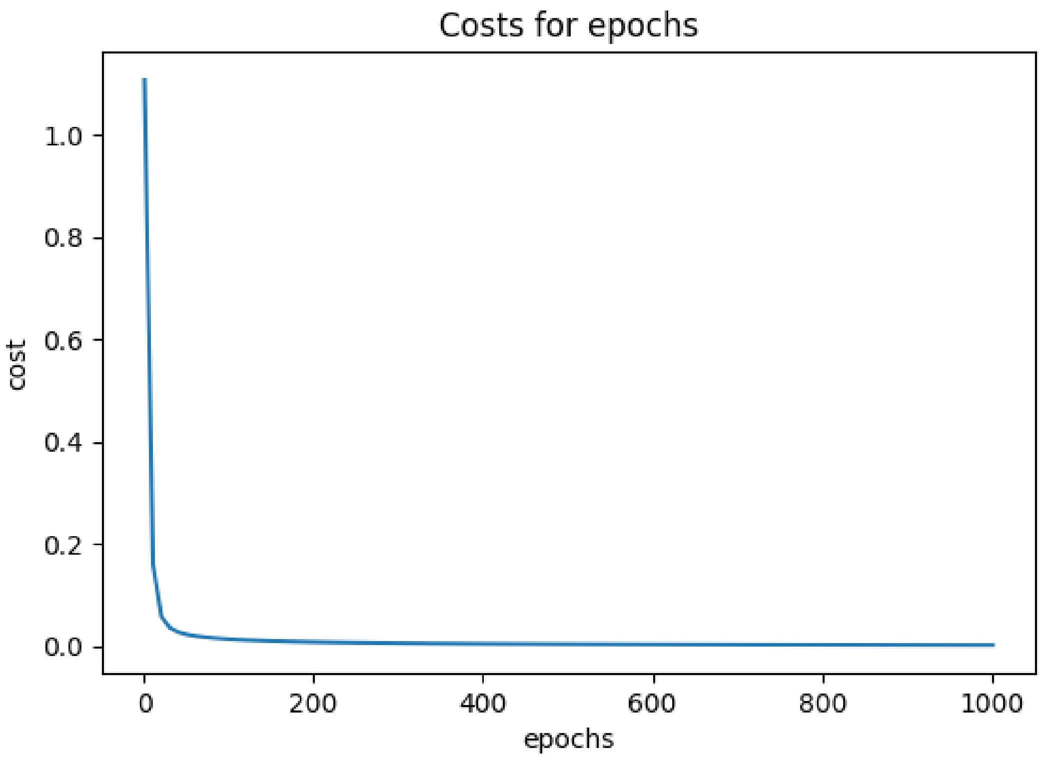
   Learning rate: 0.01

   [code]

```python
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt

def split():
    digits = load_digits()
    x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.target,
        test_size=0.25, random_state=0)

    train_idx = np.where(y_train < 3)
    y_train = y_train[train_idx]
    x_train = x_train[train_idx]
    train_idx = np.where(y_train > 0)
    y_train = y_train[train_idx]
    x_train = x_train[train_idx]

    test_idx = np.where(y_test< 3)
    y_test = y_test[test_idx]
    x_test = x_test[test_idx]
    test_idx = np.where(y_test > 0)
    y_test = y_test[test_idx]
    x_test = x_test[test_idx]

    y_train = np.where(y_train == 1, 0, 1)
    y_test = np.where(y_test == 1, 0, 1)

    x_train = torch.tensor(x_train, dtype=torch.float)
    x_test = torch.tensor(x_test, dtype=torch.float)
    y_train = torch.tensor(y_train, dtype=torch.float).view(y_train.shape[0], 1)
    y_test = torch.tensor(y_test, dtype=torch.float).view(y_test.shape[0], 1)

    return x_train, x_test, y_train, y_test

def plot_cost(nb_epochs, costs):
    costs = np.array(costs)

    fig = plt.figure()
    fig.subplots_adjust(top=0.8)
    ax1 = fig.add_subplot(111)
    ax1.set_ylabel('cost')
    ax1.set_xlabel('epochs')
    ax1.set_title('Costs for epochs')

    ax1.plot(np.arange(1, nb_epochs + 10, 10), costs)
    plt.show()
```

```python
def main():
    torch.manual_seed(1)
    x_train, x_test, y_train, y_test = split()

    model = nn.Sequential(
            nn.Linear(64, 1),
            nn.Sigmoid()
            )
    optimizer = torch.optim.SGD(model.parameters(), lr=1e-2)

    costs = []
    nb_epochs = 1000
    for epoch in range(nb_epochs + 1):
        hypothesis = model(x_train)
        cost = F.binary_cross_entropy(hypothesis, y_train)
        if epoch % 10 == 0:
            costs.append(cost.item())

        optimizer.zero_grad()
        cost.backward()
        optimizer.step()

        if epoch % 10 == 0:
            prediction = hypothesis >= torch.FloatTensor([0.5])
            correct_prediction = prediction.float() == y_train
            accuracy = correct_prediction.sum().item() / len(correct_prediction)
            print('Epoch {:4d}/{} Cost: {:.6f} Accuracy {:2.2f}%'.format(
                epoch, nb_epochs, cost.item(), accuracy * 100))

    with torch.no_grad():
        hypothesis = model(x_test)
        cost = F.binary_cross_entropy(hypothesis, y_test)
        prediction = hypothesis >= torch.FloatTensor([0.5])
        correct_prediction = prediction.float() == y_test
        accuracy = correct_prediction.sum().item() / len(correct_prediction)
        print('Cost: {:.6f} Accuracy {:2.2f}%'.format(cost.item(), accuracy * 100))

    plot_cost(nb_epochs, costs)


if __name__ == '__main__':
    main()
```

[Result]



```
root@d1a98fca2ad2:~/lab1# python logistic_regression.py
Epoch    0/1000 Cost: 1.107729 Accuracy 62.87%
Epoch   10/1000 Cost: 0.160818 Accuracy 94.49%
Epoch   20/1000 Cost: 0.058029 Accuracy 97.79%
Epoch   30/1000 Cost: 0.035865 Accuracy 99.26%
Epoch   40/1000 Cost: 0.027395 Accuracy 99.63%
Epoch   50/1000 Cost: 0.022889 Accuracy 99.63%
Epoch   60/1000 Cost: 0.019981 Accuracy 100.00%
Epoch   70/1000 Cost: 0.017877 Accuracy 100.00%
Epoch   80/1000 Cost: 0.016247 Accuracy 100.00%
Epoch   90/1000 Cost: 0.014930 Accuracy 100.00%
Epoch  100/1000 Cost: 0.013833 Accuracy 100.00%
Epoch  110/1000 Cost: 0.012902 Accuracy 100.00%
Epoch  120/1000 Cost: 0.012099 Accuracy 100.00%
Epoch  130/1000 Cost: 0.011398 Accuracy 100.00%
Epoch  140/1000 Cost: 0.010779 Accuracy 100.00%
Epoch  150/1000 Cost: 0.010228 Accuracy 100.00%
Epoch  160/1000 Cost: 0.009734 Accuracy 100.00%
Epoch  170/1000 Cost: 0.009289 Accuracy 100.00%
Epoch  180/1000 Cost: 0.008885 Accuracy 100.00%
Epoch  190/1000 Cost: 0.008517 Accuracy 100.00%
Epoch  200/1000 Cost: 0.008180 Accuracy 100.00%
Epoch  210/1000 Cost: 0.007870 Accuracy 100.00%
Epoch  220/1000 Cost: 0.007583 Accuracy 100.00%
Epoch  230/1000 Cost: 0.007319 Accuracy 100.00%
Epoch  240/1000 Cost: 0.007072 Accuracy 100.00%
Epoch  250/1000 Cost: 0.006843 Accuracy 100.00%
Epoch  260/1000 Cost: 0.006629 Accuracy 100.00%
Epoch  270/1000 Cost: 0.006429 Accuracy 100.00%
Epoch  280/1000 Cost: 0.006241 Accuracy 100.00%
Epoch  290/1000 Cost: 0.006064 Accuracy 100.00%
Epoch  300/1000 Cost: 0.005898 Accuracy 100.00%
Epoch  310/1000 Cost: 0.005740 Accuracy 100.00%
Epoch  320/1000 Cost: 0.005592 Accuracy 100.00%
Epoch  330/1000 Cost: 0.005451 Accuracy 100.00%
Epoch  340/1000 Cost: 0.005318 Accuracy 100.00%
Epoch  350/1000 Cost: 0.005191 Accuracy 100.00%
Epoch  360/1000 Cost: 0.005070 Accuracy 100.00%
Epoch  370/1000 Cost: 0.004955 Accuracy 100.00%
Epoch  380/1000 Cost: 0.004846 Accuracy 100.00%
Epoch  390/1000 Cost: 0.004741 Accuracy 100.00%
Epoch  400/1000 Cost: 0.004641 Accuracy 100.00%
Epoch  410/1000 Cost: 0.004546 Accuracy 100.00%
Epoch  420/1000 Cost: 0.004454 Accuracy 100.00%
Epoch  430/1000 Cost: 0.004367 Accuracy 100.00%
Epoch  440/1000 Cost: 0.004282 Accuracy 100.00%
Epoch  450/1000 Cost: 0.004201 Accuracy 100.00%
Epoch  460/1000 Cost: 0.004124 Accuracy 100.00%
Epoch  470/1000 Cost: 0.004049 Accuracy 100.00%
Epoch  480/1000 Cost: 0.003977 Accuracy 100.00%
Epoch  490/1000 Cost: 0.003908 Accuracy 100.00%
Epoch  500/1000 Cost: 0.003841 Accuracy 100.00%
```

```
Epoch  500/1000 Cost: 0.003841 Accuracy 100.00%
Epoch  510/1000 Cost: 0.003776 Accuracy 100.00%
Epoch  520/1000 Cost: 0.003714 Accuracy 100.00%
Epoch  530/1000 Cost: 0.003654 Accuracy 100.00%
Epoch  540/1000 Cost: 0.003596 Accuracy 100.00%
Epoch  550/1000 Cost: 0.003539 Accuracy 100.00%
Epoch  560/1000 Cost: 0.003485 Accuracy 100.00%
Epoch  570/1000 Cost: 0.003432 Accuracy 100.00%
Epoch  580/1000 Cost: 0.003381 Accuracy 100.00%
Epoch  590/1000 Cost: 0.003331 Accuracy 100.00%
Epoch  600/1000 Cost: 0.003283 Accuracy 100.00%
Epoch  610/1000 Cost: 0.003237 Accuracy 100.00%
Epoch  620/1000 Cost: 0.003192 Accuracy 100.00%
Epoch  630/1000 Cost: 0.003148 Accuracy 100.00%
Epoch  640/1000 Cost: 0.003105 Accuracy 100.00%
Epoch  650/1000 Cost: 0.003063 Accuracy 100.00%
Epoch  660/1000 Cost: 0.003023 Accuracy 100.00%
Epoch  670/1000 Cost: 0.002984 Accuracy 100.00%
Epoch  680/1000 Cost: 0.002946 Accuracy 100.00%
Epoch  690/1000 Cost: 0.002908 Accuracy 100.00%
Epoch  700/1000 Cost: 0.002872 Accuracy 100.00%
Epoch  710/1000 Cost: 0.002837 Accuracy 100.00%
Epoch  720/1000 Cost: 0.002802 Accuracy 100.00%
Epoch  730/1000 Cost: 0.002769 Accuracy 100.00%
Epoch  740/1000 Cost: 0.002736 Accuracy 100.00%
Epoch  750/1000 Cost: 0.002704 Accuracy 100.00%
Epoch  760/1000 Cost: 0.002673 Accuracy 100.00%
Epoch  770/1000 Cost: 0.002643 Accuracy 100.00%
Epoch  780/1000 Cost: 0.002613 Accuracy 100.00%
Epoch  790/1000 Cost: 0.002584 Accuracy 100.00%
Epoch  800/1000 Cost: 0.002555 Accuracy 100.00%
Epoch  810/1000 Cost: 0.002528 Accuracy 100.00%
Epoch  820/1000 Cost: 0.002501 Accuracy 100.00%
Epoch  830/1000 Cost: 0.002474 Accuracy 100.00%
Epoch  840/1000 Cost: 0.002448 Accuracy 100.00%
Epoch  850/1000 Cost: 0.002423 Accuracy 100.00%
Epoch  860/1000 Cost: 0.002398 Accuracy 100.00%
Epoch  870/1000 Cost: 0.002374 Accuracy 100.00%
Epoch  880/1000 Cost: 0.002350 Accuracy 100.00%
Epoch  890/1000 Cost: 0.002327 Accuracy 100.00%
Epoch  900/1000 Cost: 0.002304 Accuracy 100.00%
Epoch  910/1000 Cost: 0.002282 Accuracy 100.00%
Epoch  920/1000 Cost: 0.002260 Accuracy 100.00%
Epoch  930/1000 Cost: 0.002238 Accuracy 100.00%
Epoch  940/1000 Cost: 0.002217 Accuracy 100.00%
Epoch  950/1000 Cost: 0.002196 Accuracy 100.00%
Epoch  960/1000 Cost: 0.002176 Accuracy 100.00%
Epoch  970/1000 Cost: 0.002156 Accuracy 100.00%
Epoch  980/1000 Cost: 0.002137 Accuracy 100.00%
Epoch  990/1000 Cost: 0.002118 Accuracy 100.00%
Epoch 1000/1000 Cost: 0.002099 Accuracy 100.00%
Cost: 0.010524 Accuracy 100.00%
```

Cost (Cross-Entropy Loss): 0.011

Accuracy: 100 %