**Introduction to Deep Neural Networks (Spring 2021)**

**Homework #1 (50 Pts, March 24)**

**Student ID 2018310412**

**Name 고준서 (Junseo Ko)**

**Instruction:** We provide all source codes and datasets in Python. Please write your code to complete two models: *linear regression* and *logistic regression*. Besides, please measure the performance for each model.

**NOTE**: You should write your source code in the **‘EDIT HERE’ part** and do not edit other parts. You can check your code by executing the main code (‘linear\_main.py’ for Linear Regression and ‘logistic\_main.py’ for Logistic Regression).

**TIP 1**: The source code for the perceptron model is provided. Refer to the perceptron model if you are not familiar with the code structure.

**TIP 2**: You can try to implement the Scikit-learn version first and compare it with the results of your code.

[**Submission format**] When you upload your source code, please compress the following files and upload it with the file name **DNN\_HW1\_NAME\_STUDENTID.zip**. Also, convert this file to pdf and upload it as well.

./linear\_sklearn.py

./logistic\_sklearn.py

./models/LinearRegression.py

./models/LogisticRegression.py

**[20 pts]** **Linear regression**

**(1.1) [Implementation]** Implement training and evaluation function in ‘models/LinearRegression.py’ (‘train’ and ‘forward’ respectively) using the gradient descent method. Training should be based on minibatch. Given training data the mean squared error (MSE) loss is defined as follows:

Answer: Fill your code (only EDIT HERE part) here. You also have to submit your code to i-campus.

|  |
| --- |
| **Train part**  for epoch in range(epochs):  loss\_mean = []  for idx in range(0, len(y)//batch\_size+1):  # 이번 배치에 해당하는 x, y를 this\_x, this\_y에 저장  this\_y = y[idx\*batch\_size:(idx+1)\*batch\_size] # 이번 배치의 y값 리스트  this\_x = x[idx\*batch\_size:(idx+1)\*batch\_size] # 이번 배치의 x값 리스트  # forward후 gradient를 구하는 코드 작성  y\_hat = self.forward(this\_x)  y\_err = this\_y - y\_hat.squeeze()  grad = np.multiply(-this\_x.T, y\_err).T  # 배치평균 연산 후 업데이트  mini\_batch\_mean = np.mean(grad, axis = 0)  self.W = optim.update(self.W, mini\_batch\_mean.reshape(self.num\_features,1), lr)  # 로스 function을 계산하여 에포크 로스를 구하기 위해서 loss\_mean에 더해줌  loss\_mean.append(np.mean(np.square(y\_err), axis = 0)\*1/2)  final\_loss = np.mean(loss\_mean)  loss\_mean = []  if not epoch % 250: # 250 Epochs마다 Epoch와 loss를 찍어서 보여줌  print(f"Epoch: {epoch} || Loss: {final\_loss}")  **forward part**  y\_predicted = np.dot(x, self.W) |

**(1.2) [Implementation]** Implement the linear regression with scikit-learn library in ‘/linear\_sklearn.py’. The linear regression using scikit-learn library uses an analytic solution. (Use the default hyperparameters.)   
Please refer to the sample code in the following link:   
<https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html>.  
  
Answer: Fill your code (only EDIT HERE part) here. You also have to submit your code to i-campus.

|  |
| --- |
| LR = LinearRegression()  # TRAIN  LR.fit(train\_x, train\_y)  # EVALUATION  ACC = metric(test\_data[1], LR.predict(test\_data[0])) |

**(1.3) [Experiments]** For ‘Graduate’ and ‘Concrete’ datasets, please tune the number of training epochs and learning rate to minimize MSE. Report your best results for each optimizer. (In the case of ‘Full-batch,’ it is identical to the case where the mini-batch size is equal to the number of data.) Also, explain your results by comparing different methods.

Answer: Fill in the blank of the table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **Batch** | **# of epochs** | **Learning rate** | **MSE** |
| **Graduate (# of data: 400)** | Full-batch | 500 | 0.01 | 0.01 |
| Mini-batch (size=10) | 500 | 0.001 | 0.01 |
| Scikit-Learn |  |  | 0.01 |
| **Concrete**  **(# of data: 824)** | Full-batch | 150,000 | 0.1 | 134.97 |
| Mini-batch (size=10) | 4500 | 0.001 | 136.24 |
| Scikit-Learn |  |  | 134.94 |

The first dataset name “graduate” is easy to train and predict. By using the scikit-learn, I can get a 0.01 MSE score. Also, via my linear regression module with a small number of epochs, I can get a similar score in both full-batch and mini-batch compare to the scikit-learn method. The second dataset name “concrete” is hard to minimize the MSE score. I can get a 134.94 MSE score using the scikit-learn library. For the full-batch method, I can get 134.94 MSE score with epochs of 150,000 and a learning rate of 0.1. In the mini-batch experiment, I can get 136.24 MSE score with epochs of 4500 and a learning rate of 0.001. The reason that the mini-batch method must have a smaller learning rate than the full-batch is that the mini-batch is not represented the whole dataset, so we have to constrain the influence of each update in the batch. However, for the full-batch, each batch of data represents the whole dataset. So, I can use a bigger learning rate than a mini-batch. Also, in full-batch, I have to give bigger epochs than mini-batch because weights are only updated one time in each epoch, we have to update weights many times.

**[30 pts]** **Logistic Regression**

**(2.1) [Implementation]** Implement training and evaluation function in ‘models/ LogisticRegression.py’ (‘train’ and ‘forward’ respectively) using the gradient descent method. Training should be based on minibatch. Given training data the cross-entropy loss is defined as follows:

Answer: Fill your code (only EDIT HERE part) here. You also have to submit your code to i-campus.

|  |
| --- |
| self.history = {  "loss": [],  "predicts": [],  "epoch": []  }  for epoch in range(epochs):  loss\_mean = []  for idx in range(0, len(y)//batch\_size+1):  # 배치에 해당하는 x, y값을 this\_x, this\_y로 처리  this\_y = y[idx\*batch\_size:(idx+1)\*batch\_size] # 이번 배치의 y값 리스트  this\_x = x[idx\*batch\_size:(idx+1)\*batch\_size] # 이번 배치의 x값 리스트    # forward를 쓰면 0, 1로 이진값이 나오기 때문에, sigmoid만 처리한 값을 얻기 위해 따로 구성  y\_hat = self.\_sigmoid(np.dot(this\_x, self.W)).squeeze()    # Caculate Cost Function and get Loss  cost = -this\_y\*np.log(y\_hat+epsilon) - (1-this\_y) \* np.log(1-y\_hat+epsilon)  L = np.mean(cost)  # 에러를 가지고 gradient계산  y\_err = this\_y- y\_hat  mini\_batch\_grad = np.dot(this\_x.T, -y\_err)/len(this\_y) # 이번 배치의 gradient평균 연산  self.W = optim.update(self.W, mini\_batch\_grad.reshape(-1,1), lr) # 배치 전체의 gradient평균을 연산하여 그것을 기준으로 업데이트    # 에포크 로스를 구하기 위해 미니배치의 로스를 모두 추가해놓음  loss\_mean.append(L)  loss = np.mean(loss\_mean)  self.history["loss"].append(loss)  self.history["predicts"].append(self.forward(x))  self.history["epoch"].append(epoch)  if not epoch % 30 :  print(f"Epoch: {epoch} || Loss: {loss}")  loss\_mean = [] # 에포크 loss를 구하기 위한 리스트 초기화  **Forwar Part**  y\_predicted = np.dot(x, self.W)  y\_predicted = self.\_sigmoid(y\_predicted)  y\_predicted = np.where(y\_predicted > threshold, 1, 0) |

**Please See the code that I submitted via i-campus. Same code, but indentation is little weird in MS word.**

**(2.2) [Implementation]** Implement the logistic regression with scikit-learn library in ‘/linear\_sklearn.py’. Please refer to the sample code in the following link:  
<https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>.

Answer: Fill your code (only EDIT HERE part) here. You also have to submit your code to i-campus.

|  |
| --- |
| LR = LogisticRegression(max\_iter=num\_epochs)  # TRAIN  LR.fit(train\_x, train\_y)  # EVALUATION  test\_x, test\_y = test\_data  pred = LR.predict(test\_x)  ACC = metric(pred.reshape(-1, 1), test\_y) |

**(2.3) [Experiments]** For ‘Titanic’ and ‘Digit’ datasets, please tune the number of training epochs and learning rate to maximize the accuracy. Report your best results for each training method. (In the case of ‘Full-batch,’ it is identical to the case where the mini-batch size is equal to the number of data.) Also, explain your results by comparing different methods.

Answer: Fill in the blank of the table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **Batch** | **# of epochs** | **Learning rate** | **Accuracy** |
| **Titanic**  **(# of data: 779)** | Full-batch | 9000 | 0.005 | 0.824 |
| Mini-batch (size=10) | 300 | 0.0005 | 0.824 |
| Scikit-Learn |  |  | 0.81 |
| **Digit**  **(# of data: 11501)** | Full-batch | 400 | 0.5 | 0.990 |
| Mini-batch (size=10) | 150 | 0.05 | 0.994 |
| Scikit-Learn |  |  | 0.99 |

Task of this two things are classification. So, loss and also accuracy is important. Reason for epochs and learning rate is simliar to linear regression problem. For full-batch, weights are only updated once in each epochs. So, I have to train more large number of epochs in full-batch. In first, I executed mini-batch training and saw accuracy gain and loss decrease. After optimize parameters, I trained full-batch with increase learning rate and epochs.

**(2.4) [Experiments]** For the ‘Titanic’ dataset, execute the logistic regression with full-batch training and mini-batch training. Given the following parameters, draw two plots each:1) a plot whose x-axis and y-axis are **epochs** and **accuracy**, and 2) a plot whose x-axis and y-axis are **epochs** and **loss**. Use ‘matplotlib’ for plotting the graph.

|  |  |
| --- | --- |
| **Parameter Settings** | |
| Batch size | 10 |
| Learning rate | 0.0005 |
| Epsilon | 0.01 |
| Gamma | 0.9 |
| # of Epochs | 30, 60, 90, …, 300 |

Answer: Draw the figure in the blank.

|  |
| --- |
| **# Full batch training**  **# Mini- Batch of Batch size 10** |