조주현
박형민교수님
2021년 12월 1일 수요일

인공지능(딥러닝)개론 HW03

HW02의 목표는 MNIST_fashion data를 RNN과 GRU로 학습해 예측값이 각각 84%, 89%를 달성하는 것이다.

1. Library

```
1 import torch
2 import torchvision
3 from torchvision import datasets, models, transforms
4 import torchvision.transforms as transforms
5 from torch.utils.data import DataLoader, Dataset, TensorDataset
6 from torch.utils.data.dataset import random_split
7
8 import torch.nn as nn
9 import torch.nn.functional as F
10 import torch.optim as optim
11
12 import numpy as np
13 import matplotlib.pyplot as plt
```

가장 먼저 학습에 필요한 library들을 호출하는 것이다, 주로 실습시간에 배운 모듈을 호출했고, line 6는 train, validation set 분리할 때 사용하는 함수이다.

2. parameter와 dataset, dataloader 선언

```
1 train_dataset, val_dataset = random_split(train_dataset, [6000,54000]) # 0.1 validation
2
3 train_loader = DataLoader(dataset=val_dataset,
4                           batch_size = batch_size,
5                           shuffle = True)
6
7 val_loader = DataLoader(dataset=val_dataset,
8                         batch_size = batch_size,
9                         shuffle = False)
10
11 test_loader = DataLoader(dataset=test_data,
12                           batch_size=batch_size,
13                           shuffle=False)
```

train_loader와 test_loader는 실습시간 사용한 코드와 동일한 코드를 사용했지만, validation set을 위해 60,000개의 train data중에서 54,000개는 training에, 6,000개는 validation에 이용했다. 보통 validation은 20%를 사용했지만, 이번 과제의 경우 training이 잘 되지 않아 validation data set을 최소한으로 줄여 학습을 진행했다.

```
1 print('number of training dataset:', len(train_loader) * batch_size)
2 print('number of test dataset:', len(test_data.data))
3
4 print('type and shape of dataset:', train_loader.dataset.dataset.data[0].shape)
```

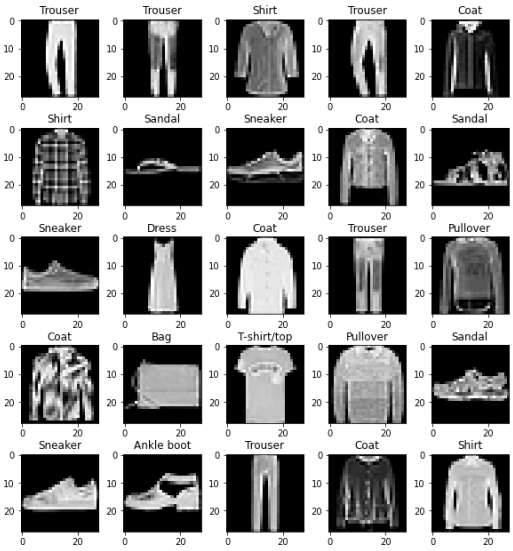
```
number of training dataset: 54016
number of test dataset: 10000
type and shape of dataset: torch.Size([28, 28])
```

Training dataset과 test dataset 그리고 image의 크기를 확인하는 함수이다. 각각 54,016개의 training dataset, 10,000개의 test dataset, image의 크기는 28*28이다.

3. MNIST_fashion plot

```
1 row, col = 5,5
2 count = 1
3 plt.figure(figsize=(10,10))
4 plt.subplots_adjust(left=0.125, bottom=0.1,  right=0.9, top=0.9, wspace=0.2, hspace=0.4)
5
6 for a in range(row):
7   for b in range(col):
8     plt.subplot(row, col, count)
9     count += 1
10    plt.imshow(test_data.data[count].reshape(28,28), cmap='gray')
11    plt.title(test_data.classes[test_data[count][1]])
```

test_data의 이미지를 25개 꺼내서 확인하는 함수이다. 5 x 5로 출력하였고, line 4는 subplot간 겹치지 않기 위해 조정해주는 함수이다. 출력 결과는 아래와 같다.



3. Structure 구현

```
1 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
2
3 class RNN(nn.Module):
4   def __init__(self, intput_size, hidden_size, num_layers, num_classes):
5     super(RNN, self).__init__()
6     self.hidden_size = hidden_size
7     self.num_layers = num_layers
8     self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True, dropout=0.4)
9     self.fc = nn.Linear(hidden_size, num_classes)
10
11    torch.nn.init.xavier_uniform_(self.fc.weight) # weight initializer
12
13
14  def forward(self, x):
15    # set initial hidden states
16    h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
17
18    #Forward propagate RNN
19    out, _  = self.rnn(x, (h0)) # output: tensor [batch_size, seq_length, hidden_size]
20
21    #Decode the hidden state of the last time step
22    out = self.fc(out[:,-1,:])
23
24    return out
25
26 model = RNN(input_size, hidden_size, num_layers, num_classes).to(device)
```

RNN 모델의 구조이다. 위 함수는 기본적인 RNN 모델을 구성하는 함수이며 GRU 모델을 구성하기 위해선 line 8에 self.rnn = nn.GRU(…) 로 선언해주면 된다. Dropout rate는 0.4로 선언했으며, 이는 경험적으로 선언한 값이다. 또한 overfitting을 방지하기 위해 line 11에 xavier initializer를 선언해 주었다.

4. Loss and Optimizer

```
1 criterion = nn.CrossEntropyLoss()
2 optimizer = torch.optim.Adam(model.parameters(), lr=0.0005)
```

Loss function은 cross entropy, optimizer는 Adam을 사용하였고 Learning rate는 0.0005로 설정했다.

5. Model save

```
68   if best_model_loss > val_loss_value :
69     best_model_loss = val_loss_value
70     best_epoch = epoch
71     print("your model is saved as RNN_epoch_{}.pth\n\n".format(best_epoch+1))
72     torch.save(model.state_dict(), "RNN_epoch_{}.pth".format(best_epoch+1))
```

Validation data를 이용해 loss값을 구한 뒤 loss값이 감소하면 해당 모델을 저장하는 함수이다. Line 71을 통해 모델이 저장되었는지 여부를 알 수 있다.

6. Result

loss값이 가장 낮은 model을 이용해 test dataset을 예측한 결과 아래와 같은 정확도 값이 나왔다.

RNN: Test Accuracy of RNN model on the 10000 test images: 86.21%
GRU: Test Accuracy of RNN model on the 10000 test images: 91.05%

7. Discussion

RNN과 GRU의 parameter 수의 차이는 약 3배이다. 아래 함수를 통해 구조와 parameter 수를 확인할 수 있다.

```
def count_parameters(model):
  return sum(p.numel() for p in model.parameters() if p.requires_grad)
print("Parameters in GRU model")
for p in model.parameters():
  print(p.size())
print("\n")

model_hp = count_parameters(model)
print('model\'s hyper parameters', model_hp)
```

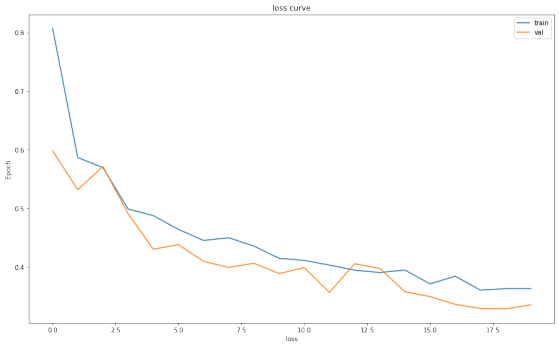
출력 결과는 아래와 같다. 왼쪽과 오른쪽은 RNN과 GRU 모델의 결과이다.

```
Parameters in RNN model
torch.Size([256, 28])
torch.Size([256, 256])
torch.Size([256])
torch.Size([256])
torch.Size([256, 256])
torch.Size([256, 256])
torch.Size([256])
torch.Size([256])
torch.Size([256, 256])
torch.Size([256, 256])
torch.Size([256])
torch.Size([256])
torch.Size([10, 256])
torch.Size([10])


model's hyper parameters 338954
```

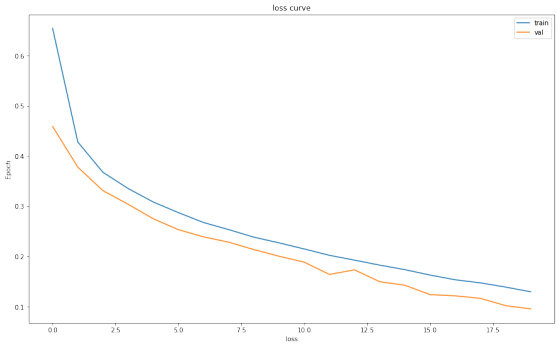
```
Parameters in GRU model
torch.Size([768, 28])
torch.Size([768, 256])
torch.Size([768])
torch.Size([768])
torch.Size([768, 256])
torch.Size([768, 256])
torch.Size([768])
torch.Size([768])
torch.Size([768, 256])
torch.Size([768, 256])
torch.Size([768])
torch.Size([768])
torch.Size([10, 256])
torch.Size([10])


model's hyper parameters 1011722
```

parameter수가 큰 만큼 학습도 더 잘 일어났다. Learning curve를 확인하면 아래와 같다.

RNN loss curve



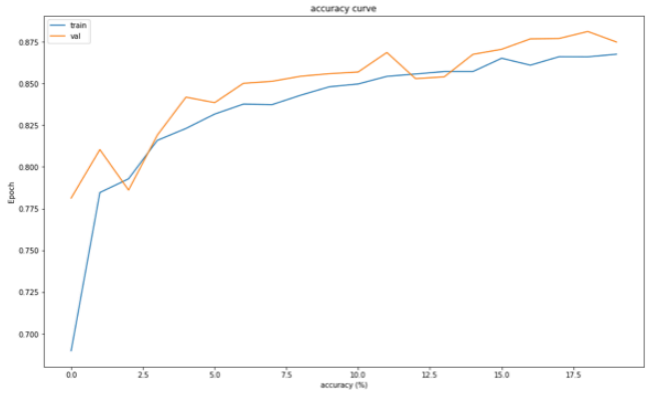
GRU loss curve



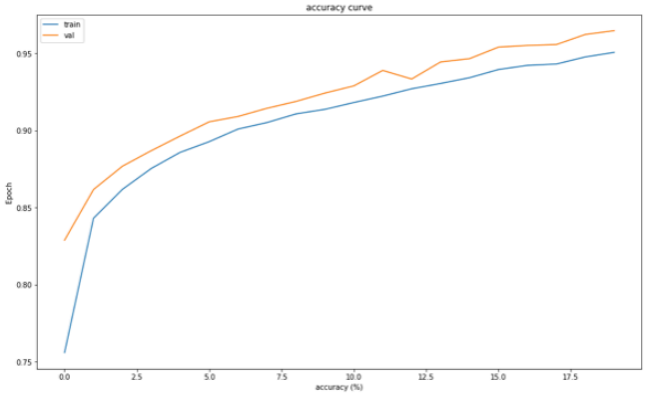
각 그래프를 확인했을 때 GRU 모델이 RNN 모델보다 일정하게 감소하는 것을 볼 수 있다. 게다가 loss값의 크기 자체도 작은 것을 확인할 수 있다. Hidden state와 layer를 동일하게 준 것을 보았을 때 각 모델의 유의미한 성능의 차이가

있음을 알 수 있다. 다만 GRU의 경우 validation과 test사이의 정확도 값이 컸다. 먼저 accuracy curve의 그래프를 보면 아래와 같다.

RNN accuracy curve



GRU accuracy curve



RNN에서 validation의 경우 0.9를 넘지 않았지만 GRU의 경우 0.95 이상으로 증가하는 것을 볼 수 있다. 하지만 test dataset에서의 결과는 위에서 확인했듯이 0.91으로 계산된다. 차이가 커서 저장된 모델들의 test dataset에 대한 정확도를 확인해 보았다. 확인을 위한 함수는 아래와 같다.

```
import os

path = "/content/"
file_list = os.listdir(path)
file_list_pth = [file for file in file_list if file.endswith(".pth")]
file_list_pth.sort()
test_acc_list = []

for i in range(len(file_list_pth)):
  test_model = RNN(input_size, hidden_size, num_layers, num_classes).to(device)
  test_model.load_state_dict(torch.load('/content/'+file_list_pth[i], map_location=torch.device('cpu')))
  test_model.eval()
  correct = 0

  with torch.no_grad():
    ...
    print('Test Accuracy of RNN model:{} is: {}%'.format(file_list_pth[i], 100 * correct / len(test_data)))
```

accuracy를 확인하는 함수는 길이 관계상 생략하였다. 저장된 디렉토리에 확장자명을 기준으로 list를 만들어 모델의 정확도를 확인했다. 결과는 아래와 같다.

```
Test Accuracy of RNN model:GRU_epoch_01.pth is: 81.95%
Test Accuracy of RNN model:GRU_epoch_02.pth is: 85.02%
Test Accuracy of RNN model:GRU_epoch_03.pth is: 86.18%
Test Accuracy of RNN model:GRU_epoch_04.pth is: 87.16%
Test Accuracy of RNN model:GRU_epoch_05.pth is: 88.12%
Test Accuracy of RNN model:GRU_epoch_06.pth is: 88.67%
Test Accuracy of RNN model:GRU_epoch_07.pth is: 88.73%
Test Accuracy of RNN model:GRU_epoch_08.pth is: 89.31%
Test Accuracy of RNN model:GRU_epoch_09.pth is: 89.51%
Test Accuracy of RNN model:GRU_epoch_10.pth is: 89.8%
Test Accuracy of RNN model:GRU_epoch_11.pth is: 90.11%
Test Accuracy of RNN model:GRU_epoch_12.pth is: 90.45%
Test Accuracy of RNN model:GRU_epoch_14.pth is: 90.42%
Test Accuracy of RNN model:GRU_epoch_15.pth is: 90.61%
Test Accuracy of RNN model:GRU_epoch_16.pth is: 90.78%
Test Accuracy of RNN model:GRU_epoch_17.pth is: 90.72%
Test Accuracy of RNN model:GRU_epoch_18.pth is: 90.67%
Test Accuracy of RNN model:GRU_epoch_19.pth is: 90.94%
Test Accuracy of RNN model:GRU_epoch_20.pth is: 91.05%
```

확인해보면 epoch이 11이상일 때 accuracy 값이 많이 증가하지 않은 것을 확인할 수 있다. 이는 validation에 대한 어느정도의 학습이 되어 validation accuracy값이 증가했다고 예상할 수 있다.

모델의 정확도를 증가시키는 방법으로는 layer를 증가시키거나 hidden state를 증가시키는 방법이 있을 것이다. 하지만 layer를 증가시키게 되면 gradient vanishing problem이 일어날 가능성이 있어 유의해야 한다. 본 과제에서는 학습 과정의 시간을 줄이기 위해 Layer을 3으로 고정했고, GRU와 RNN의 성능 비교를 위해 layer와 hidden state의 개수를 동일하게 설정하였다. 세 모델의 trained parameter 는 각각 .pth file로 저장한 후 코드 또한 .py file로 변환하여 저장하였다.