인공지능(딥러닝)개론 Project 결과 보고서

2020.12.11 (금) 11:59PM / 20181202 김수미

1. 프로젝트 과제 목표

EMNIST(bymerge) 이미지 데이터셋을 분류하는 인공지능 모델을 설계하되, Evaluation Accuracy 값을 최대화 하는 것이 이번 프로젝트의 목표이다. 대상 데이터셋을 총 두가지 형태로 분류해 볼 것이다.

1) EMNIST(bymerge)_Digit/Letter Classification using LSTM

- EMNIST(bymerge) 데이터를 숫자와 영문자로 분류하는 모델을 만든다.

2) EMNIST(bymerge) Classification using LSTM

- EMNIST(bymerge) 데이터를 총 47 개의 class 로 분류하는 모델을 만든다.

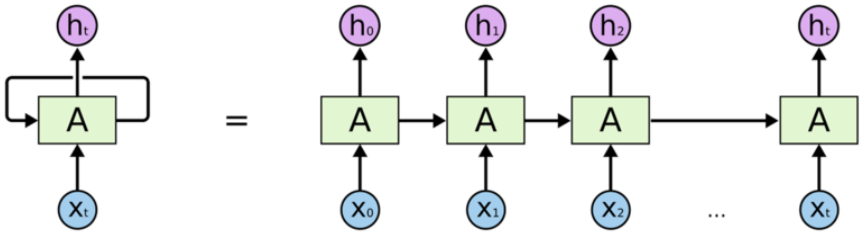
2. 배경 이론

1) EMNIST(bymerge) 데이터셋

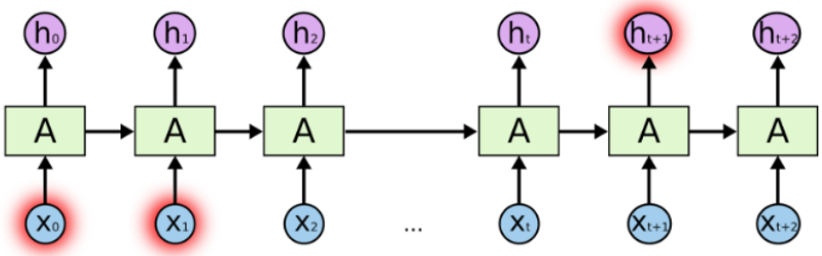
먼저 EMNIST 데이터셋은 영문자와 숫자 손글씨 이미지 데이터로 이루어진 데이터셋이다. 0 번 라벨부터 9 번 라벨은 총 0 부터 9 까지의 숫자 이미지 클래스를 나타내고, 10 번 라벨부터 46 번 라벨은 총 37 개의 영어 대소문자 이미지 클래스를 나타낸다. 영문자 데이터 클래스가 37 개인 이유는 총 26 개의 알파벳의 대/소문자 52 개 중에서 대문자와 소문자가 모양이 비슷해 구별이 어려운 경우(알파벳 C, I, J, K ,L, M, O, P, S, U, V, W, X, Y, Z 가 이에 속한다)는 대문자와 소문자를 하나로 간주하기 때문이다(52 개 – 15 개 = 37 개). 이를 위해 데이터셋을 다운받을 때 'bymerge split' 처리를 해준다.

2) RNN 과 LSTM

과제의 분류 모델에 이용한 순환신경망(RNN)의 한 종류인 LSTM 에 대해 간단하게 살펴보자. 먼저 순환신경망(Recurrent Neural Network, RNN)이란 내부에 루프를 가진 신경망의 한 종류로, hidden node 가 방향성을 가지는 edge 로 연결되어 있어 순환구조를 이룬다. 즉, hidden state 의 결과가 다시 같은 hidden state 의 입력으로 들어가도록 연결되어 있다. 이렇게 매 과정에서 처리한 정보를 state 에 저장하기 때문에 RNN 은 순서 또는 시간이라는 측면을 고려할 수 있으며, 지속적이고 연속적인 데이터를 처리하는데 유리하다. 하지만 제공된 데이터와 참고해야 할 정보의 거리가 멀어지면 RNN 은 두 정보의 연속성을 파악하기 힘들어지고 성능이 저하된다. 입력 정보의 길이 또는 정보 간 간격이 커지면 Vanishing Gradient Problem 이 발생하는 것이다.

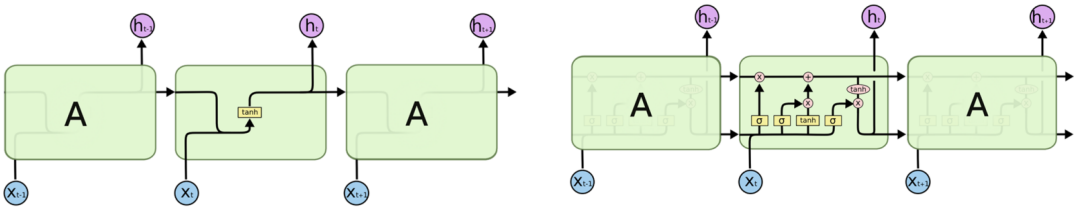


〈 그림 1. RNN 의 구조 〉



〈 그림 2. RNN 의 Vanishing Gradient Problem 〉

이 문제를 극복하기 위해 고안된 것이 LSTM 이다. LSTM 은 RNN 의 hidden state 에 cell state 를 추가한 구조이다. cell state 는 하나의 컨베이어 벨트처럼 전체 체인을 통과한다. 따라서 정보의 길이나 정보 간의 간격이 길어져도 마지막까지 그래디언트가 비교적 잘 전파된다. 아래 그림은 보면 RNN 과 LSTM 의 내부 구조를 도식화 한 것으로, RNN 은 single layer 이지만 LSTM 은 훨씬 복잡한 구조를 가지고 있음을 확인할 수 있다.



〈 그림 3. RNN(좌)과 LSTM(우)의 구조 비교 〉

3. 코드 세부 구성 및 설명

1) Model 1 : EMNIST(bymerge) Digit/Letter Classification using LSTM

- Hyper Parameters :

```
num_classes = 2 #고정값 (digit/letter)
input_size = 28 #고정값
sequence_length = 28 #고정값
num_epochs = 8
```

learning_rate = 0.001
batch_size = 256
hidden_dim = 128
num_layers = 3
drop_percent = 0.2

- Loss Function : Cross Entropy Loss Function
- Optimizer : Adam Optimizer
- Dropout 사용여부 : 예
- Random Seed 값 고정 : 예
- 사용 모델 종류 : LSTM
- 코드 세부 구성 및 설명

해당 코드에서 작성할 모델의 목적은 EMNIST 데이터셋의 숫자 이미지 데이터와 영문자 이미지 데이터를 분류하는 것이다. 숫자 이미지의 경우 라벨 0 을, 영문자 이미지의 경우 라벨 1 을 클래스로 갖기 때문에 num_classes 변수는 2 의 값을 가지게 된다. 또한 EMNIST 데이터셋의 각 데이터가 28 x 28 픽셀의 이미지 데이터이기 때문에, input_size 와 sequence_length 는 각각 28 의 값을 가지게 된다. 데이터셋은 총 697,932 개의 train 데이터와 116,323 개의 test 데이터로 구성되어 있다. 데이터 다운로드는 수업에서 제공된 ProjectUtils.py 의 TypeData class 를 이용했다.

```
class Model1(nn.Module):
  def __init__(self, input_size, hidden_dim, num_layers, num_classes, drop_percent):
    super(Model1, self).__init__()
    self.hidden_size = hidden_dim
    self.num_layers = num_layers

    self.drop_percent = drop_percent
    self.dropout = nn.Dropout(drop_percent)

    self.fc = nn.Linear(hidden_dim, num_classes)
    self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_dim, num_layers=num_layers, batch_first=True)

  def forward(self, x):
    h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
    c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)

    out, _ = self.lstm(x, (h0, c0))
    out = self.dropout(out)
    out = self.fc(out[:,-1,:])

    return out
```

〈 그림 4. Model 1 Python Code 〉

데이터 분류 모델은 RNN 모델의 일종인 LSTM 모델을 이용하여 작성했다. 사실 EMNIST 는 이미지 데이터셋이기 때문에 이미지의 공간 구조를 활용하는 CNN 모델이 더 적합하다고 말할 수 있지만, 연속된

픽셀로 이루어진 이미지 데이터에서 인접한 영역의 픽셀은 서로 연관되어 있으므로 이를 시퀀스 데이터로 볼 수도 있다. 즉 한 EMNIST 데이터셋 이미지의 크기가 28 x 28 픽셀일 때, 시퀀스의 각 원소는 28 개의 픽셀을 가지며 길이가 28 인 시퀀스 데이터로 볼 수 있는 것이다. 실제로 LSTM 모델로 분류를 진행해 보았을 때 높은 정확도를 보이기도 했고, Pytorch 에서 LSTM class 를 제공하기 때문에 모델 레이어 구성이 CNN 보다 훨씬 간편하다는 장점이 있다. 따라서 LSTM 을 모델의 구조로 선택해 프로젝트를 진행하게 되었다.

batch_size, learning_rate, hidden_dim, num_layers, num_epochs 등의 변수는 제한된 시간 안에 가장 높은 정확도를 가지는 모델을 작성하기 위해 매번 정확도와 Loss 값, 소요 시간 등을 확인해 가면서 값을 조정해 주었고, 더 높은 정확도를 확보하기 위해 Dropout 을 추가해 주었다. Dropout 이란 신경망에서 정확도를 높이기 위해 사용되는 기법으로, 네트워크 층에 Dropout 을 적용하면 훈련하는 동안 무작위로 층의 일부 출력 특성을 제외시킨다. Dropout 의 비율은 제외되는 특성의 비율이며, 일반적으로 0.2 에서 0.5 사이의 값으로 지정된다. 이러한 Dropout 기법은 모델의 Overfitting 을 줄여준다. 각 샘플에 대해 뉴런의 일부를 무작위하게 제거하면 뉴런 사이의 부정한 협업을 방지하고 결국 Overfitting 을 감소시킬 수 있는 것이다.

그 다음 def forward(self,x) 에서 2 차원 환경을 형성해주기 위해 torch.zeros 이용하여 h0 과 c0 값을 아래와 같이 설정하고 LSTM 모델에 lstm(x, (h0, c0)) 형식으로 함께 넣어 주었다.

```
h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
```

〈 그림 5. torch.zeros 이용한 h0 과 c0 값 설정 〉

Output(코드상에서는 out)의 Activation function 은 선형 회귀 모델인 Linear 함수를 사용해 주었는데, Pythor 에서 제공하는 LSTM 모델 내부 hidden state 의 Activation function 은 비선형 하이퍼볼릭탄젠트(tanh) 함수를 사용한다. 아래는 LSTM 모델 내부에서 각 레이어가 수행하는 함수이다.

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

〈 그림 6. LSTM 모델 내부 (출처 : https://pytorch.org/) 〉

2) Model 2 : EMNIST(bymerge) Classification using LSTM

- Hyper Parameters :

  num_classes = 47 #고정값 (EMNIST(bymerge))

  input_size = 28 #고정값

  sequence_length = 28 #고정값

  num_epochs = 7

  learning_rate = 0.001

  batch_size = 256

  hidden_dim = 256

  num_layers = 3

  drop_percent = 0.2

- Loss Function : Cross Entropy Loss Function

- Optimizer : Adam Optimizer

- Dropout 사용여부 : 예

- Random Seed 값 고정 : 예

- 사용 모델 종류 : LSTM

- 코드 세부 구성 및 설명

해당 코드에서 작성할 모델의 목적은 EMNIST 데이터셋의 모든 이미지 데이터를 47 개의 클래스로 분류하는 것이다(숫자 10 가지 + 영문자 37 가지). 따라서 num_classes 변수는 47 의 값을 가지게 된다. input_size 와 sequence_length 데이터셋의 개수는 Model 1 과 동일하다.

```
class Model2(nn.Module):
  def __init__(self, input_size, hidden_dim, num_layers, num_classes, drop_percent):
    super(Model2, self).__init__()
    self.hidden_size = hidden_dim
    self.num_layers = num_layers
    self.drop_percent = drop_percent

    self.dropout = nn.Dropout(drop_percent)
    self.fc = nn.Linear(hidden_dim, num_classes)

    self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_dim, num_layers=num_layers, batch_first=True)

  def forward(self, x):
    h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
    c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)

    out, _ = self.lstm(x, (h0, c0))
    out = self.dropout(out)
    out = self.fc(out[:,-1,:])

    return out
```

〈 그림 7. Model 2 Python Code 〉

데이터 분류 모델은 Model 1 에서와 동일한 이유로 LSTM 모델을 사용하였고, 반복적으로 train loss 값과 소요시간, test 정확도를 확인해가면서 hyper parameters 를 조정한 결과 Model 1 과는 다르게 hidden_dim 값을 256 으로, num_epochs 값을 7 로 잡아 주게 되었다. Dropout 등 나머지 부분은 Model1 과 동일하기 때문에 자세한 설명은 생략하겠다.

4. 모델 학습 결과

1) 실험 Colab GPU 환경

- Colab Pro 를 구독하고 있어 가장 속도가 빠른 Tesla P100 GPU 를 할당 받아 진행했다.

```
Fri Dec 11 12:16:40 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 455.45.01    Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   39C    P0    32W / 250W |   2311MiB / 16280MiB |      0%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

〈 그림 8. GPU 실험 환경 〉

2) EMNIST(bymerge)_Digit/Letter Classification using LSTM

- Train 소요 시간 및 Loss Graph

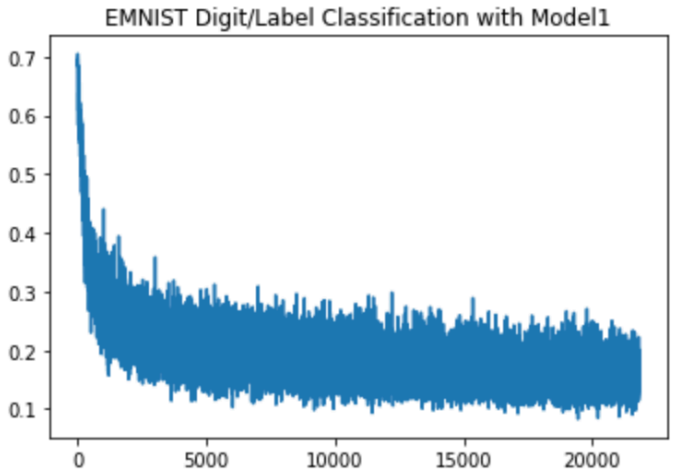
```
Epoch [1/8], Step[1000/2727], Loss:0.2947
Epoch [1/8], Step[2000/2727], Loss:0.1744
Epoch [2/8], Step[1000/2727], Loss:0.1760
Epoch [2/8], Step[2000/2727], Loss:0.1462
Epoch [3/8], Step[1000/2727], Loss:0.1720
Epoch [3/8], Step[2000/2727], Loss:0.1536
Epoch [4/8], Step[1000/2727], Loss:0.1647
Epoch [4/8], Step[2000/2727], Loss:0.1666
Epoch [5/8], Step[1000/2727], Loss:0.1791
Epoch [5/8], Step[2000/2727], Loss:0.1849
Epoch [6/8], Step[1000/2727], Loss:0.1656
Epoch [6/8], Step[2000/2727], Loss:0.1685
Epoch [7/8], Step[1000/2727], Loss:0.1141
Epoch [7/8], Step[2000/2727], Loss:0.1695
Epoch [8/8], Step[1000/2727], Loss:0.1342
Epoch [8/8], Step[2000/2727], Loss:0.1519

>> Train takes 13.89minutes
```

```
loss_list.append(
    float: best_loss
    0.08286681771278381
if loss.item() < best_loss:
```

〈 그림 9. Model 1 : Train 소요 시간 및 best_loss 값 〉

Model 1 을 학습시키는 데는 총 14 분 조금 덜 되게 소요되었다. 가장 학습이 잘 된 epoch 에서 loss 값은 약 0.0828 정도로 측정되었다. 아래 그림은 학습 과정에 따른 Loss 값의 변화를 그래프로 나타낸 것으로, 반복을 여러 번 거칠수록 점차 Loss 값이 감소하는 것을 확인할 수 있다.



〈 그림 10. Model 1 : Loss Graph 〉

- Evaluation Accuracy on Test Data

```
Accuracy of Model1 on the 116480 test images: 92.05528846153847%
```

Test 데이터셋에 대한 분류 정확도는 약 92%로 나왔다. 깎인 8%의 정확도는 숫자 0 과 알파벳 O 를 구별하는 부분에서 발생한 것으로 추정된다

3) EMNIST(bymerge) Classification using LSTM

- Train 소요 시간 및 Loss Graph

```
Epoch [1/7], Step[1000/2727], Loss:0.4990
Epoch [1/7], Step[2000/2727], Loss:0.3504
Epoch [2/7], Step[1000/2727], Loss:0.3609
Epoch [2/7], Step[2000/2727], Loss:0.3041
Epoch [3/7], Step[1000/2727], Loss:0.3362
Epoch [3/7], Step[2000/2727], Loss:0.1997
Epoch [4/7], Step[1000/2727], Loss:0.2417
Epoch [4/7], Step[2000/2727], Loss:0.2449
Epoch [5/7], Step[1000/2727], Loss:0.2363
Epoch [5/7], Step[2000/2727], Loss:0.3308
Epoch [6/7], Step[1000/2727], Loss:0.2128
Epoch [6/7], Step[2000/2727], Loss:0.2347
Epoch [7/7], Step[1000/2727], Loss:0.1941
Epoch [7/7], Step[2000/2727], Loss:0.2678

>> Train takes 13.95 minutes
```

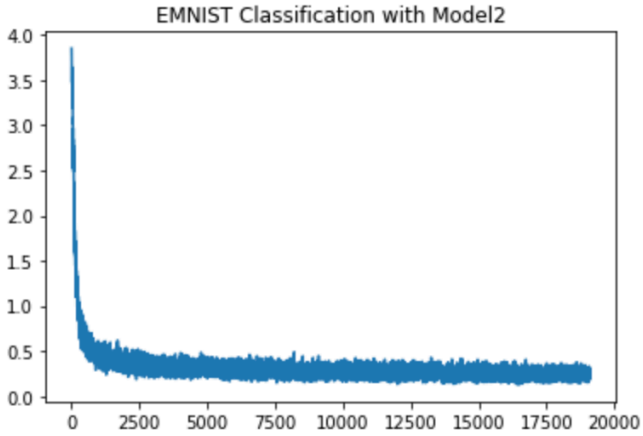
```
loss_list.append(loss.item())
if loss.item() < best_loss:
```

float: best_loss

0.12720723450183868

〈 그림 11. Model 2 : Train 소요 시간 및 best_loss 값 〉

Model 2 를 학습시키는 데는 총 14 분이 조금 덜 되게 소요되었다. 가장 학습이 잘 된 epoch 에서 loss 값은 약 0.1272 정도로 측정되었다. 아래 그림은 학습 과정에 따른 Loss 값의 변화를 그래프로 나타낸 것으로, 반복을 여러 번 거칠수록 점차 Loss 값이 감소하는 것을 확인할 수 있다.



〈 그림 12. Model 2 : Loss Graph 〉

- Evaluation Accuracy on Test Data

```
Accuracy of Model1 on the 116480 test images: 90.07383241758242%
```

Test 데이터셋에 대한 분류 정확도는 약 90%로 측정되었다. 깎인 10%의 정확도는 Model1 에서와 마찬가지로 숫자 0 과 알파벳 O 를 구별하는 부분에서 발생한 것으로 추정된다

5. 참고문헌

- 프랑소와 숄레, 『케라스 창시자에게 배우는 딥러닝』, 길벗(2018)

- GitHub Pages, "LSTM", https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/, 2020.12.11

- Naver Blog, "LSTM", https://url.kr/DtJ7Av, 2020.12.11

+) 기타 참고자료 : 인공지능(딥러닝)개론 수업 자료 및 실습 코드