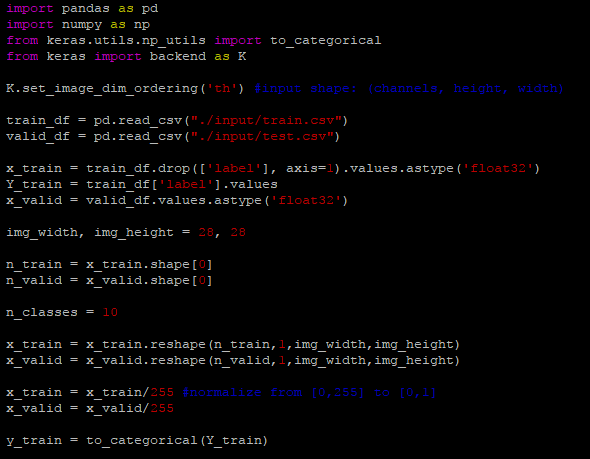
데이터 마이닝 HW #3

20141575 임형석

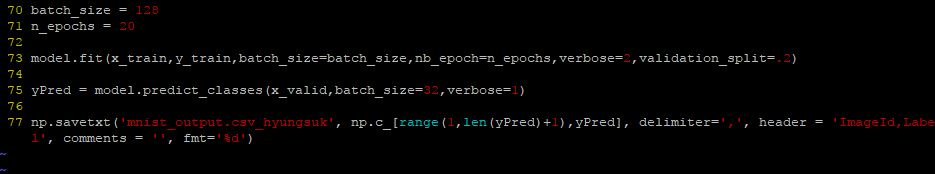
**[Exercise 1] code에 대한 설명**



먼저 Pandas와 Numpy library를 import하였다. 그 다음에 train.csv 파일과 test.csv파일을 읽어와 pandas dataframes에 저장시키고 그리고 .values를 이용하여 data들을 numpy array로 전환시켜주었다. 그리고 속도를 위해 datatype을 float64에서 float32로 바꾸어 주었다.

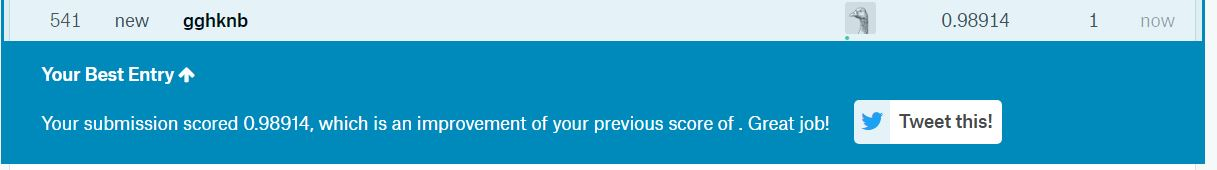
training example들이 1D vector이기 때문에 이것을 2D image로 convolution해주어야하는데, 그래서 input data들을 (n\_train x 794) 에서 (x\_train \* 28 \* 28) 즉 1D vector에서 2D image로 reshape해주었고 또한 data들을 구간 [0,1]로 normalize해주었다. 그리고 to\_categorical을 이용하여 [0,1,2,3,4,5,6,7,8,9]와 같은 target data들을 one hot vector로 바꾸어주었다.



이 부분이 cnn에 해당되는 부분인데 sequential()을 이용해 input layer를 생성해주었다. 그 다음에 cnn 인 main operation인 2Dconvolution을 해주는데, 파라미터로 필터의 개수가 64개, 그리고 kernel size는 3x3, activation 은 relu 그리고 border\_mode라는 parameter를 valid로 설정해주었다. border\_mode 옵션은 이미지 내부에 필터가 들어있는 convolution들만 처리하고 이 옵션을 설정하면 filter map의 크기가 입력 이미지보다 작은 dimension을 가지게 된다. 그 다음에는 max pooling을 해주었는데 max pooling은 겹치지 않는 부분 영역에 최대filter를 적용해서 milter map의 사이즈를 3\*3으로 줄여주고, stride안에서 가장 큰 값을 뽑아준다.그 다음은 dropout을 해주었는데, 이 것은 overfitting을 방지해주기 위해 사용되었다. dropout은 전체 weight를 계산에 참여시키는게 아니라 layer에 포함된 weight중에서 일부만 참여시켜 overfitting을 방지시켜준다. flattening은 (n\_filters, filter\_size\_x, filter\_size\_y)같은 모양의 array인 input activation을 1D array로 전환시켜주는 역할을 한다. 그리고 dense layer를 만든다. dense layer는 fully connected layer인데 1D input이 들어가 1D output으로 나오게 한다. 그리고 마지막으로 soft max activation을 사용해 10개의 output을 만들어 주었다. 즉 각각 0부터 9사이의 확률을 표현해주었다. 그 다음 compiile을 통하여 adam optimizer와 loss function은 categorical\_crossentropy로 지정해 주었다.

그 다음 epoch 개수와 batch size를 정한 후에 model을 학습시켜주었고, 그 결과는 mnist\_output\_hyungsuk.csv에 저장될 수 있도록 하였다.

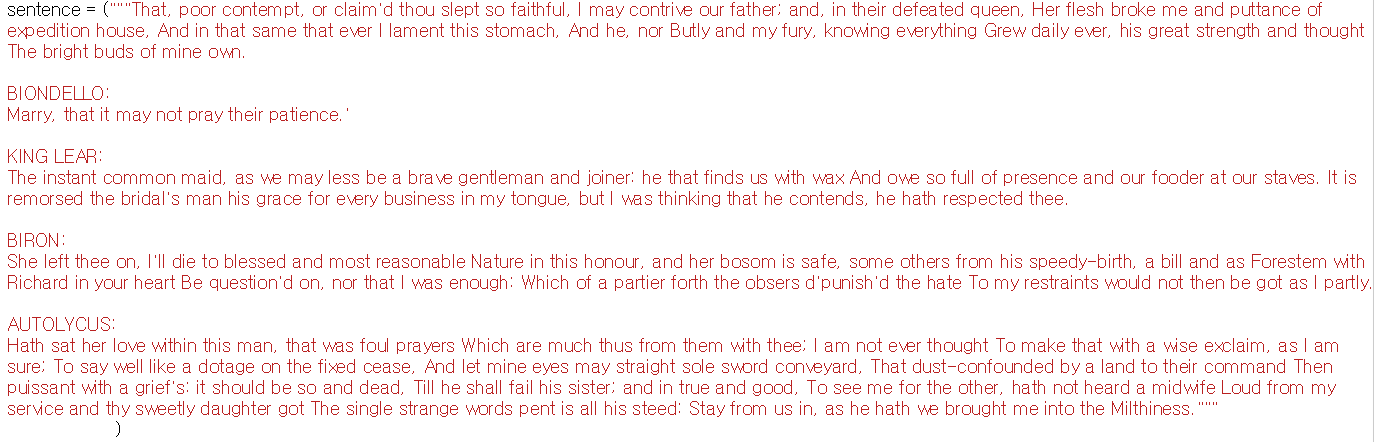
**kaggle 결과**



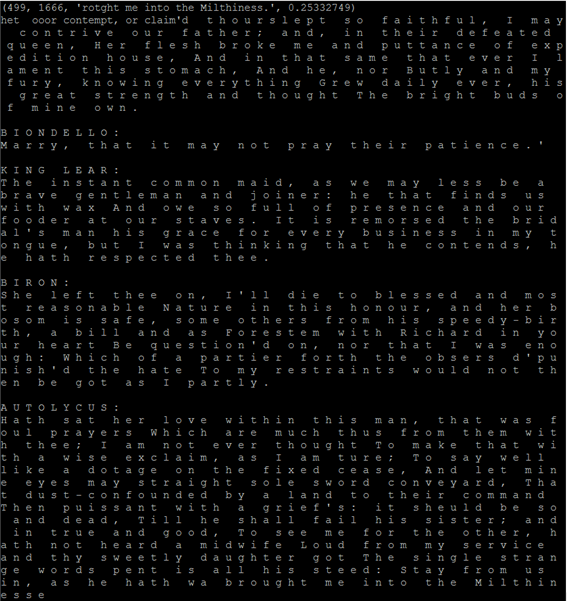
[Exercise 2]

**- 결과로 나오는 텍스트와 실제 텍스트를 비교하여 정확도 도출 및 차이 분석**

실제 텍스트 는 다음과 같다.



결과로 나오는 텍스트는 다음과 같다.



실제 텍스트와 결과로 나온 텍스트가 큰 차이를 보이지는 않지만, 처음 부분에만 제대로 학습이 되지 않은 것을 볼 수 있는데, 과거 context가 존재하지 않아 정확도가 더 떨어지는 것으로 보인다.

**-여러 변수들을 변경해보면서 rnn.py 실행**

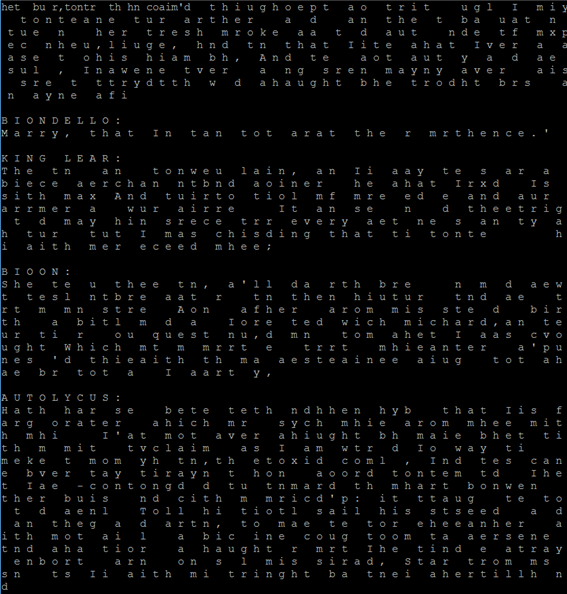
**cell의 종류를 바꾸어 실행시킨 결과**

GRU에 대한 결과



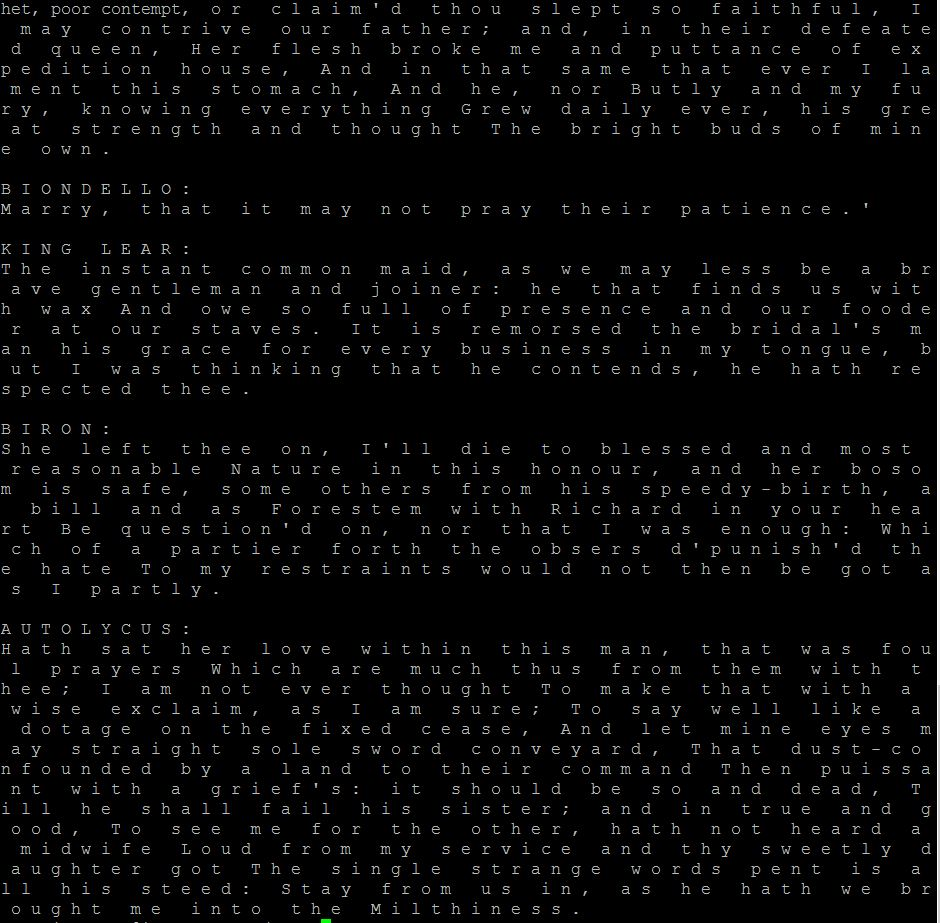
seq\_lengt를 20으로 하고 GRU로 한 결과이다. loss율이 처음에 주어진 lstm일 때보다 낮은 것을 볼 수 있고, 실제 텍스트와 결과 텍스트를 비교했을 때도 거의 대부분 비슷한 것을 알 수 있다.

RNN에 대한 결과



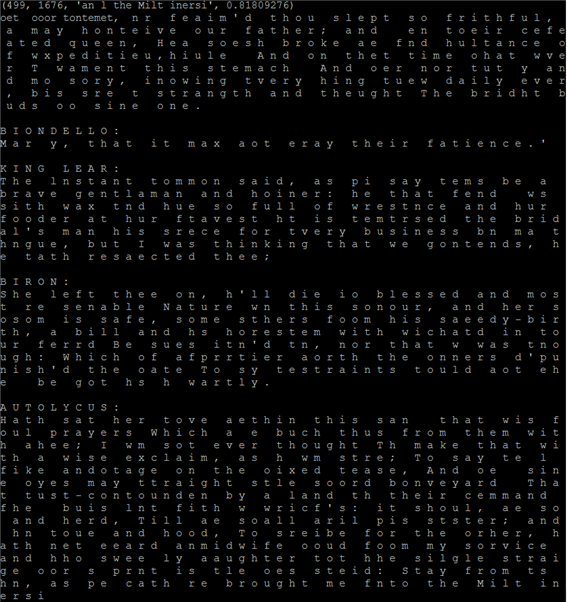
RNN으로 했을 때의 결과인데 original 텍스트와 비교했을 때 정확도가 많이 떨어진 것을 알 수 있다.

**multiRNNCell의 layer를 2개로 했을때**



처음에 주어진 LSTM코드와 비교했을 때 mutiRNNCell layer를 2개로 한 것이 더 결과가 좋아진 것을 알 수 있다. 특히 맨 첫문장에서 주어진 코드에서는 오류가 많았지만 layer가 2개일 경우에는 첫문장에도 잘 학습이 된 것을 볼 수 있다.

**seq\_lengt를 30에서 20으로 줄여서 학습 시킨 결과**



seq\_lengt가 30일 때보다 20일때가 loss가 증가하였고 정확도도 많이 낮아 진 것을 알 수 있다.

**-정확도 높이는 방향으로 모델 수정 후 보고서에 최종 모델 기재**

위에서 여러가지 변수들을 바꾸어 보면서 rnn.py를 실행해 보았을 때 GRU를 사용하고 seq\_leng을 20으로 줄인 것이 가장 loss가 낮은 것을 볼 수 있었다. 그 뿐만 아니라 육안으로 텍스트를 확인했을 경우 첫문장의 정확도도 올라갔고 그 뒤로도 거의 정확하게 학습이 된 것을 알 수 있다. 따라서 최종 모델 코드는 다음과 같다.

