조문기

1. Objective

   이 프로젝트에서는 Kaggle competition을 통해서 Quora Question Pairs에 관해 진행이 되는데, 그 내용은 다음과 같다. Quora에는 많은 질문들이 올라오는데, 중복된 질문들이 있다면 그 질문에 관해서 답을 주는 것들이 이득을 취할 수 있기 때문에 우리가 만약 그러한 질문들의 유사성을 판단해서 결정을 내려 주기를 바라는 것 이다.

2. Strategy_1

   나의 경우 Keras를 사용해서 sentence-pair classification을 위한 딥러닝 모델을 구현했다. 맨 처음에 전처리와 자연어처리를 하게 되는데, 다행히도 sample.csv 가 정갈하게 되어 있어서 String, Character 등 으로 먼저 변환할 필요는 없었다. 다만 처리한 부분은 크게 2 가지로 나누어 진다. 첫째는 '알파벳과 종결어미로만 표현하기' 라는 것인데, 다음 코드를 보는 것이 이해가 빠를 수 있다.

```
text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)
text = re.sub(r"what's", "what is ", text)
text = re.sub(r"\'s", " ", text)
text = re.sub(r"\'ve", " have ", text)
text = re.sub(r"can't", "cannot ", text)
text = re.sub(r"n't", " not ", text)
text = re.sub(r"i'm", "i am ", text)
text = re.sub(r"\'re", " are ", text)
text = re.sub(r"\'d", " would ", text)
text = re.sub(r"\'ll", " will ", text)
text = re.sub(r",", " ", text)
text = re.sub(r"\.", " ", text)
text = re.sub(r"!", " ! ", text)
text = re.sub(r"\/", " ", text)
text = re.sub(r"\^", " ^ ", text)
text = re.sub(r"\+", " + ", text)
text = re.sub(r"\-", " - ", text)
text = re.sub(r"\=", " = ", text)
text = re.sub(r"'", " ", text)
text = re.sub(r"(\d+)(k)", r"\g<1>000", text)
text = re.sub(r":", " : ", text)
text = re.sub(r" e g ", " eg ", text)
text = re.sub(r" b g ", " bg ", text)
text = re.sub(r" u s ", " american ", text)
text = re.sub(r"\0s", "0", text)
text = re.sub(r" 9 11 ", "911", text)
text = re.sub(r"e - mail", "email", text)
text = re.sub(r"j k", "jk", text)
text = re.sub(r"\s{2,}", " ", text)
```

   옆의 sub함수를 통해서 하는 방법은 다음과 같다. 알파벳과 숫자, 그리고 종결어미와 계산기호를 제외한 모든 문자는 공백으로 처리하며, 우리가 일상생활에서 사용하는 축약어는 그 문자를 풀어서, 그리고 아무래도 질문의 대다수를 포함하는 곳이 미국이므로 우리를 아메리카로 변경을 해서 사용을 하도록 한다. '\문자' 로 받아들이는 것을 그냥 '문자' 로 변환하도록 한다.

3. Method_1

   첫째 방법이 전처리의 영역이었다면, 둘째는 자연어처리의 방법이었다. 각각의 단어에 고유번호를 만들고 문장을 그 인덱스의 연결로 처리를 하고자 한다. 이를 돕기 위해서 Keras의 Tokenizer를 필요로 한다.

```
tokenizer = Tokenizer(num_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(texts_1 + texts_2 + test_texts_1 + test_texts_2)
```

조문기

Tokenizer.fit_on_texts는 입력에 맞게 내부의 단어 인덱스를 만드는 함수여서 각각의 단어마다 고유의 인덱스가 할당된 것을 알 수 있다. 그후 이들의 연결, 즉 시퀀스로 만들어야 하는데 이는 Tokenizer.tests_to_sequence를 통해서 매우 쉽게 얻어 낼 수 있다. 그 전에 워드의 총 갯수를 구해두는 것이 나중에 도움이 된다.

```
sequences_1 = tokenizer.texts_to_sequences(texts_1)
sequences_2 = tokenizer.texts_to_sequences(texts_2)
test_sequences_1 = tokenizer.texts_to_sequences(test_texts_1)
test_sequences_2 = tokenizer.texts_to_sequences(test_texts_2)

word_index = tokenizer.word_index
print('Found %s unique tokens' % len(word_index))
```

여기서 문장의 길이가 각각 다르기 때문에 이제 서로 같게 처리를 해주는 방식을 이용하는데 이 함수는 pad_sequences를 통해서 서로 같은 입력 길이를 하는데, 빈공간은 0으로 채워지게 된다.

```
data_1 = pad_sequences(sequences_1, maxlen=MAX_SEQUENCE_LENGTH)
data_2 = pad_sequences(sequences_2, maxlen=MAX_SEQUENCE_LENGTH)
labels = np.array(labels)
```

이제 입력 데이터를 통해서 단어 임베딩을 해야하는데, 나의 경우는 워드투 백터를 이용해서 임베딩을 하게 해 두었다. GoogleNews-vectors-negative300.bin을 통해서 사용을 하게 되어 있는데, 이는 구글 도메인에 파일명만 기입해도 받을 수 있으니 명시하지 않겠다. 그리고 용량이 커서 zip파일 속에는 없을 터이니 채점하실때 받아주시면 감사하겠다. 워드 임베딩에서는 인덱스를 단어로 다시 재 변환해서 그 단어로 임베딩을 찾는 방법을 사용을 한다.

```
embedding_layer = Embedding(nb_words,
        EMBEDDING_DIM,
        weights=[embedding_matrix],
        input_length=MAX_SEQUENCE_LENGTH,
        trainable=False)
```

LSTM 2개로 듀얼 인코더를 만들고 우리가 loss 함수를 구하는 부분을 만들어서 진짜면 1, 가짜면 0에 가까워지도록 훈련을 시키고자 한다. 그래서 다음과 같은 코드가 필요하다.

4. Strategy_2

모델을 학습시키기 이전에 weight를 설정해서 넣어주도록 한 점이 약간 다른 방법이었다. 점수가

조문기

```
x1 = lstm_layer(embedded_sequences_1)
y1 = lstm_layer(embedded_sequences_2)
lstm_layer = LSTM(num_lstm, dropout=rate_drop_lstm, recurrent_dropout=rate_drop_lstm)

merged = concatenate([x1, y1])
merged = Dropout(rate_drop_dense)(merged)
merged = BatchNormalization()(merged)

merged = Dense(num_dense, activation=act)(merged)
merged = Dropout(rate_drop_dense)(merged)
merged = BatchNormalization()(merged)

model = Model(inputs=[sequence_1_input, sequence_2_input], \
        outputs=preds)
model.compile(loss='binary_crossentropy',
        optimizer='nadam',
        metrics=['acc'])
```

잘 나온 것들의 weight를 리서치 해봤는데, 0일때 1.309정도, 1일때 0.472정도를 weight를 한다면 좋은 결과가 나온다고 찾을 수 있었다.

```
if re_weight:
    class_weight = {0: 1.309028344, 1: 0.472001959}
else:
    class_weight = None
```

5. Method_2

이제 머신러닝을 시키도록 하는데, 한번에 처리하고자 하는 입력의 개수는 2048개, 그리고 섞는 것도 가능하다구 주며, 위에서 설정한 weight들을 넣어서 계산을 해 보았다. 데이터의 학습 바퀴수를 10으로 설정했는데, 노트북이 GPU가 없는 상태여서 총 12시간 정도 걸렸다.

```
model.load_weights(bst_model_path)
bst_val_score = min(hist.history['val_loss'])

print('Start making the submission before fine-tuning_moon')

preds = model.predict([test_data_1, test_data_2], batch_size=8192, verbose=1)
preds += model.predict([test_data_2, test_data_1], batch_size=8192, verbose=1)
preds /= 2

submission = pd.DataFrame({'test_id':test_ids, 'is_duplicate':preds.ravel()})
```

머신러닝에서 가장 loss가 낮은 것을 찾아서 이제 dataFrame을 저장하고 그 후 그것을 제출하도록 해 주면 된다.

조문기

6. Comparison

   word2vec가 보여주는 performance가 GloVe보다 더 강력하게 보여지는것 같다. 두개를 받아서 각각을 다 실행해 보았는데, GloVe의 경우 loss가 조금 큰것 같았다.