Text Classification – assignment2 보고서

컴퓨터소프트웨어학부

2016025996

조성진

Assignment2 는 주어진 text 의 label 이 0 인지 1 인지 구별하는 model 의 accuracy 를 최대한 높이는것이 목적이다.

과제를 해결하기 위해 기본적인 RNN 에서 LSTM 으로 바꾸고 MultiRNNCell 을 통해 4 개의 LSTM cell 을 사용해도 정확도가 74%정도밖에 나오지 않아 attention 을 적용하였다. 실험에 사용한 모델은 2 가지인데, 2 가지 모델을 hyperparmeter 부분만 바꿔가면서 실험하였다. 과제설명에서 제시된 설명과 image classification 에서 나온 함수들은 추가 설명을 붙이지 않고 build_classifier 에서 모델에 관련된 내용을 설명하겠다. 과제 1 과 마찬가지로 loss 에 regularization 을 하였으며 dropout 을 적용하였다.

테스트에 사용한 두가지 모델중 첫번째 코드는 다음과 같다.

```
cell1 = tf.nn.rnn_cell.LSTMCell(HIDDEN_SIZE, name="c1")

# RNN layer
rnn_outputs1, states1 = tf.nn.dynamic_rnn(cell=cell1, inputs=batch_embedded, dtype=tf.float32)
rnn_outputs = rnn_outputs1

w_omega = tf.Variable(tf.random_normal([HIDDEN_SIZE, ATTENTION_SIZE], stddev=0.1))
b_omega = tf.Variable(tf.random_normal([ATTENTION_SIZE], stddev=0.1))
u_omega = tf.Variable(tf.random_normal([ATTENTION_SIZE], stddev=0.1))

with tf.name_scope('v'):
    v = tf.tanh(tf.tensordot(rnn_outputs, w_omega, axes=1) + b_omega)
    vu = tf.tensordot(v, u_omega, axes=1, name='vu')
    alphas = tf.nn.softmax(vu, name='alphas')

    output = tf.reduce_sum(rnn_outputs * tf.expand_dims(alphas, -1), 1)

# Fully connected layer
W = tf.Variable(tf.random_uniform([HIDDEN_SIZE, 2], -1.0, 1.0), trainable=True)
b = tf.Variable(tf.random_uniform([2], -1.0, 1.0), trainable=True)
#logits = tf.nn.bias_add(tf.matmul(rnn_outputs[:,-1], W), b)
logits = tf.nn.bias_add(tf.matmul(output, W), b)
hypothesis = tf.nn.softmax(logits)

return hypothesis, logits
```

여기서 attention 을 적용하여 각 output 마다 attention score 을 구해서 weighted sum 을 통해 FCN 으로 학습시킨다.

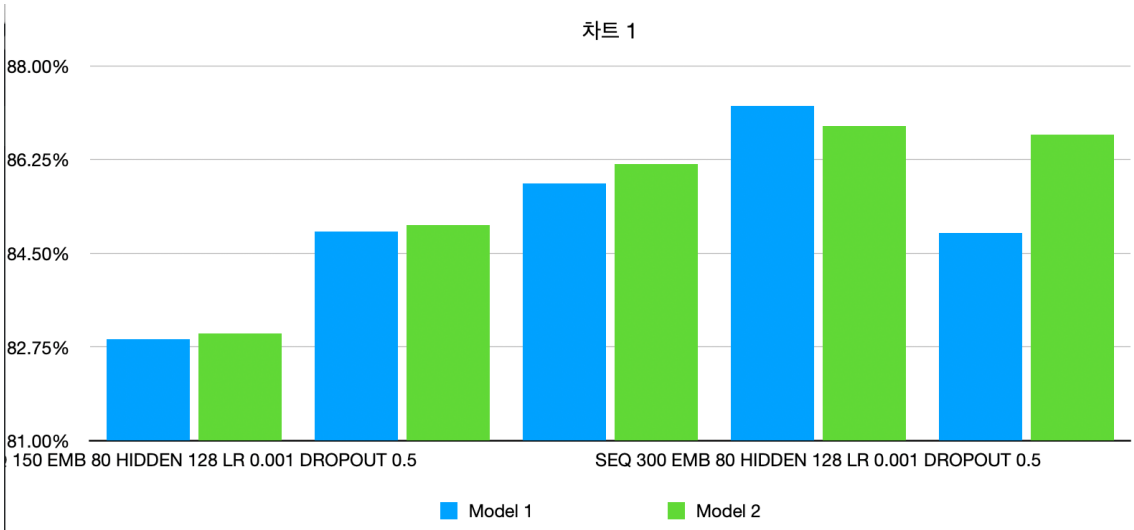
두번째 모델의 코드는 다음과 같다.

```
cell1 = tf.nn.rnn_cell.LSTMCell(HIDDEN_SIZE, name="c1")
cell2 = tf.nn.rnn_cell.LSTMCell(HIDDEN_SIZE, name="c2")

# RNN layer
rnn_outputs1, states1 = tf.nn.dynamic_rnn(cell=cell1, inputs=batch_embedded, dtype=tf.float32)
rnn_outputs2, states2 = tf.nn.dynamic_rnn(cell=cell2, inputs=batch_embedded[::-1], dtype=tf.float32)
rnn_outputs = tf.concat((rnn_outputs1,rnn_outputs2), axis=2)

w_omega = tf.Variable(tf.random_normal([HIDDEN_SIZE*2, ATTENTION_SIZE], stddev=0.1))
b_omega = tf.Variable(tf.random_normal([ATTENTION_SIZE], stddev=0.1))
u_omega = tf.Variable(tf.random_normal([ATTENTION_SIZE], stddev=0.1))

print("rnn_outputs:", rnn_outputs)
print("w_omega:", w_omega)
with tf.name_scope('v'):
    v = tf.tanh(tf.tensordot(rnn_outputs, w_omega, axes=1) + b_omega)
    vu = tf.tensordot(v, u_omega, axes=1, name='vu')
    alphas = tf.nn.softmax(vu, name='alphas')

    output = tf.reduce_sum(rnn_outputs * tf.expand_dims(alphas, -1), 1)

# Fully connected layer
W = tf.Variable(tf.random_uniform([HIDDEN_SIZE*2, 2], -1.0, 1.0), trainable=True)
b = tf.Variable(tf.random_uniform([2], -1.0, 1.0), trainable=True)
#logits = tf.nn.bias_add(tf.matmul(rnn_outputs[:,-1], W), b)
logits = tf.nn.bias_add(tf.matmul(output, W), b)
hypothesis = tf.nn.softmax(logits)

return hypothesis, logits
```

달라진것은 cell 을 만들때 input 을 정방향으로 1 번, 역방향으로 1 번 총 2 번 LSTM 을 통해 학습시키고 concatenation 을 통해 연결하고 attention 을 구한 코드이다. Seq2seq 의 논문에 문장의 순서를 뒤바꿨을때 정확도가 더 높게 나왔다는 결과가 있었고 text classification 을 할때 이런식으로 concatenation 을 통해 attention 을 구했을때 성능이 좀더 좋게 나온다는 글도 봤기에, 두가지 모델을 대상으로 실험을 해봤다.

우선 모두 epoch 는 50 으로 시작했고, 과제 1 번과 마찬가지로 earlystop 을 적용해서 loss 값이 특정 횟수 이상 떨어지면 중단시켰다.

먼저 첫번째로 sequence length 만 변화시키며 테스트 했다.

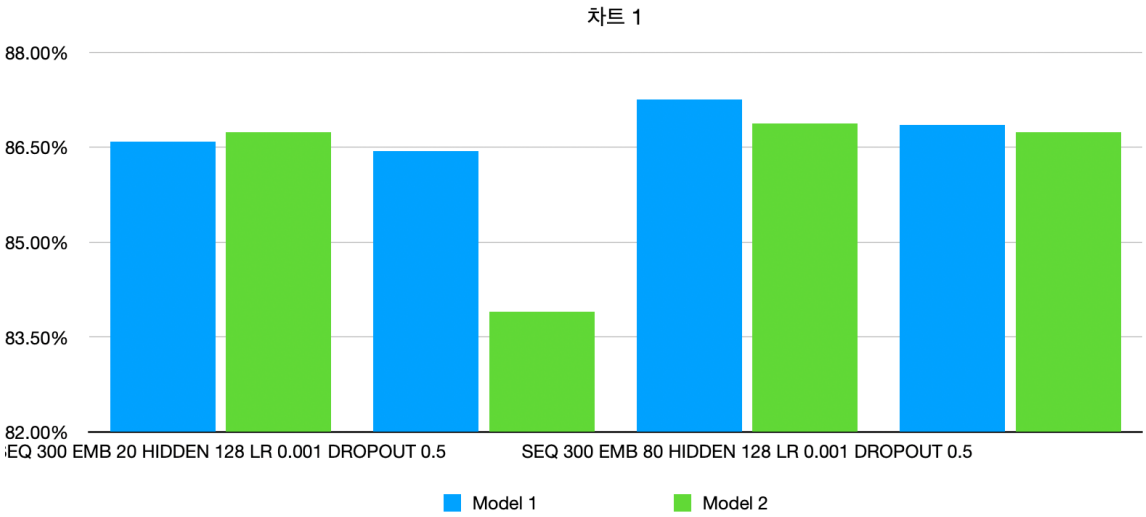


SEQUENCE LENGTH만 변화

	Model 1	Model 2
SEQ 150 EMB 80 HIDDEN 128 LR 0.001 DROPOUT 0.5	82.90%	83.00%
SEQ 200 EMB 80 HIDDEN 128 LR 0.001 DROPOUT 0.5	84.90%	85.02%
SEQ 250 EMB 80 HIDDEN 128 LR 0.001 DROPOUT 0.5	85.80%	86.16%
SEQ 300 EMB 80 HIDDEN 128 LR 0.001 DROPOUT 0.5	87.25%	86.87%
SEQ 350 EMB 80 HIDDEN 128 LR 0.001 DROPOUT 0.5	84.88%	86.72%
SEQ 400 EMB 80 HIDDEN 128 LR 0.001 DROPOUT 0.5	85.60%	86.18%

이 결과 대부분의 경우에 model2 가 성능이 조금 더 좋았지만 최고성능을 냈을때가 model1 에서 나왔다.

이 결과로 sequence length 는 300 으로 고정시키고 다음은 embedding size 를 바꿔보았다.

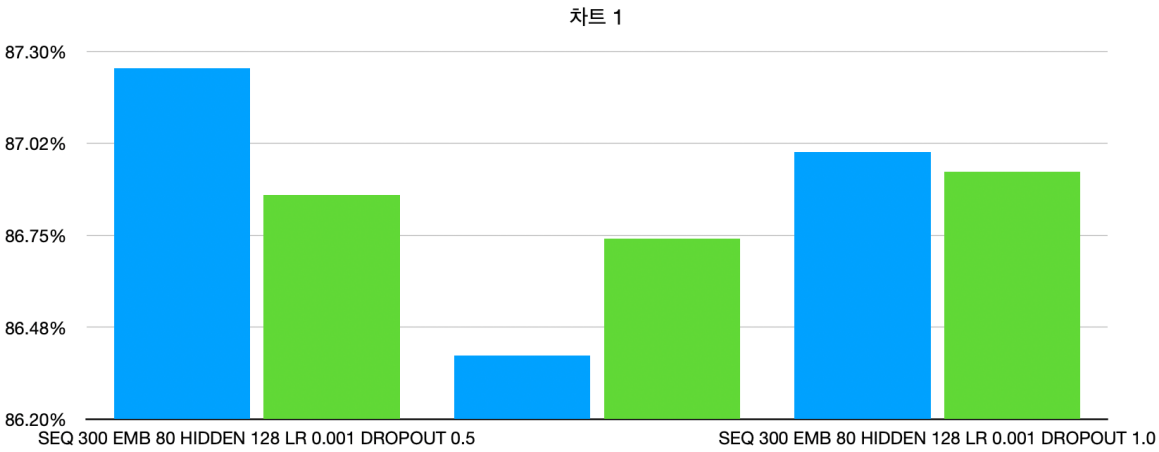


Embedding size만 변화

	Model 1	Model 2
SEQ 300 EMB 20 HIDDEN 128 LR 0.001 DROPOUT 0.5	86.59%	86.74%
SEQ 300 EMB 50 HIDDEN 128 LR 0.001 DROPOUT 0.5	86.44%	83.90%
SEQ 300 EMB 80 HIDDEN 128 LR 0.001 DROPOUT 0.5	87.25%	86.87%
SEQ 300 EMB 100 HIDDEN 128 LR 0.001 DROPOUT 0.5	86.85%	86.74%

이렇게 실험해본 결과 sequence length=300 인 경우에는 model1 이 좋은 경우도 있었다. 최고성능은 마찬가지로 기존에 나왔던 값이 최고 성능이 나왔다.

그리고 마지막으로 dropout rate 를 변화시켜가면서 테스트를 해보았다.



Dropout만 변화

	Model 1	Model 2
SEQ 300 EMB 80 HIDDEN 128 LR 0.001 DROPOUT 0.5	87.25%	86.87%
SEQ 300 EMB 80 HIDDEN 128 LR 0.001 DROPOUT 0.7	86.39%	86.74%
SEQ 300 EMB 80 HIDDEN 128 LR 0.001 DROPOUT 1.0	87.00%	86.94%

결과적으로 SEQ = 300, Embedding size = 80, dropout=0.5 인 model1 으로 dev set 에 대하여 87.25%(최고의 결과가 나왔을때 저장하지 않고 새로운 데이터로 계속 실험해서, 같은 모델로 실행시켜 최종 제출은 87%)의 accuracy 를 보이는 모델로 과제를 완료했다.

```
Acc: 0.8958333  and loss: 0.45502305
dev Accuracy: 0.856500 Saved
Epoch 30
Acc: 0.9166667  and loss: 0.43433544
dev Accuracy: 0.843700 Saved
Epoch 31
Epoch 32
Epoch 33
Epoch 34
Epoch 35
Epoch 36
Acc: 0.9791667  and loss: 0.3418153
dev Accuracy: 0.869800 Saved
Epoch 37
Epoch 38
Epoch 39
Epoch 40
Epoch 41
Epoch 42
Early stopped!
dev 데이터 Accuracy: 0.870000
```