Text Classification

컴퓨터 소프트웨어 학부

2018008059 김은수

- 환경
- 코드설명
- 결과설명

1. 환경

- Tensorflow 1.13.1
- Python 3.7.10

```
import tensorflow as tf
import sys

print(tf.__version__)
print(sys.version)
```

```
/usr/local/lib/python3.7/dist-packages/ten
  _np_qint8 = np.dtype([("qint8", np.int8
/usr/local/lib/python3.7/dist-packages/ten
  _np_quint8 = np.dtype([("quint8", np.uin
/usr/local/lib/python3.7/dist-packages/ten
  _np_qint16 = np.dtype([("qint16", np.in
/usr/local/lib/python3.7/dist-packages/ten
  _np_quint16 = np.dtype([("quint16", np.u
/usr/local/lib/python3.7/dist-packages/ten
  _np_qint32 = np.dtype([("qint32", np.in
/usr/local/lib/python3.7/dist-packages/ten
  np_resource = np.dtype([("resource", np
1.13.1
3.7.10 (default, May  3 2021, 02:48:31)
[GCC 7.5.0]
```

2. 코드 설명

- **하이퍼파라매터**

```
ATTENTION_UNITS = 100
SEQUENCE_LENGTH = 500
EMBEDDING_DIM = 100
HIDDEN_SIZE = 150
BATCH_SIZE = 128
NUM_EPOCHS = 10
learning_rate = 0.001
```

Batch size는 128, sequence_length는 500, epoch은 10이다. Atttention 메커니즘의 attention 벡터의 차원은 100으로 설정했다.

Dropout

```
keep_prob = tf.placeholder(tf.float32)
KEEP_PROB = 0.75
```

KEEP_PROB는 0.75로 75%의 노드를 활성화한다.

- **딥러닝 layer**

```
def build_classifier(x, vocabulary_size, EMBEDDING_DIM, HIDDEN_SIZE):
    # Embedding layer
    embeddings_var = tf.Variable(tf.random_uniform([vocabulary_size, EMBEDDING_DIM], -1.0, 1.0), trainable=True)
    batch_embedded = tf.nn.embedding_lookup(embeddings_var, x)

    # RNN layer
    rnn_outputs, states = tf.nn.dynamic_rnn(rnn_cell.GRUCell(HIDDEN_SIZE),batch_embedded, dtype=tf.float32)

    # Attention layer
    attention_output, alphas = attention(rnn_outputs, ATTENTION_UNITS, return_alphas=True)
```

Embedding layer를 지나게 되면 text가 각 단어가 뜻하는 embedding 벡터로 표현된다. 이 값을 RNN cell에 넣어서 output을 받아서 classification할 것이다. 이때 모델로는 GRUCell을 사용한다. GRUCell을 넣어서 나온 rnn_output을 attention 모델에 넣어서 attention output을 도출하고 이를 fully connected layer에 넣어준다.

```
#Fully
W1_fc = tf.get_variable(name="W1_fc", shape=[HIDDEN_SIZE, 64], initializer=tf.contrib.layers.xavier_initializer())
b1_fc = tf.get_variable(name="b1_fc", shape=[64], initializer=tf.contrib.layers.xavier_initializer())
l1_fc = tf.nn.relu(tf.nn.bias_add(tf.matmul(attention_output, W1_fc), b1_fc))
l1_fc = tf.nn.dropout(l1_fc, keep_prob=KEEP_PROB)

W2_fc = tf.get_variable(name="W2_fc", shape=[64, 32], initializer=tf.contrib.layers.xavier_initializer())
b2_fc = tf.get_variable(name="b2_fc", shape=[32], initializer=tf.contrib.layers.xavier_initializer())
l2_fc = tf.nn.relu(tf.nn.bias_add(tf.matmul(l1_fc, W2_fc), b2_fc))
l2_fc = tf.nn.dropout(l2_fc, keep_prob=KEEP_PROB)

W3_fc = tf.get_variable(name="W3_fc", shape=[32, 2], initializer=tf.contrib.layers.xavier_initializer())
b3_fc = tf.get_variable(name="b3_fc", shape=[2], initializer=tf.contrib.layers.xavier_initializer())

logits = tf.nn.bias_add(tf.matmul(l2_fc, W3_fc), b3_fc)
hypothesis = tf.nn.softmax(logits)

return hypothesis, logits
```

Fully connected layer부분은 다음과 같다. 3개의 layer를 거쳐서 HIDDEN_SIZE차원에서 2개의 뉴런으로 나타난다. 1,2 layer에서는 활성화 함수로 Relu함수를 사용하고, 마지막에는 softmax함수를 사용한다.

- **Attention model**

```
def attention(inputs, attention_size, time_major= False, return_alphas = False):
    if isinstance(inputs,tuple):
        input = tf.concat(inputs,2)

    if time_major:
        inputs = tf.array_ops.transpose(inputs, [1,0,2])

    hidden_size = inputs.shape[2].value

    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(inputs, 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(inputs * tf.expand_dims(alphas, -1), 1)

    if not return_alphas:
        return output
    else:
        return output, alphas
```

Attention model은 다음과 같다. input으로 rnn_outputs이 들어오고 이를 attention mechanism을 통해 output을 출력한다.

- **Mini batch를 위한 함수**

```
def batch_data(shuffled_idx, batch_size, data, labels, start_idx):
    idx = shuffled_idx[start_idx:start_idx+batch_size]
    data_shuffle = [data[i] for i in idx]
    labels_shuffle = [labels[i] for i in idx]

    return np.asarray(data_shuffle), np.asarray(labels_shuffle)
```

- **cost함수와 Optimizer**

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y, logits=logits))
train_step = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

Cost 함수로는 softmax_cross_entropy 함수를 사용했으며 optimizer는 Adam optimizer를 사용했다. Learning rate는 0.001로 설정했다. (0.01로 설정했을 때 accuracy 0.5로 성능이 좋지 않았다.)

3. 결과 설명

```
학습시작
Epoch 1
2021-06-04 07:55:21.744466: I tensorflow/str
WARNING:tensorflow:From /usr/local/lib/pytho
Instructions for updating:
Use standard file APIs to check for files wi
Epoch 2
Epoch 3
Epoch 4
Epoch 5
Epoch 6
Epoch 7
Epoch 8
Epoch 9
Epoch 10
dev 데이터 Accuracy: 0.881500
```

- 몇 번 돌리는지에 따라 (num_epoch이 몇 인지에 따라) 큰 차이가 없었고 batch_size 조정으로 인해 학습 속도가 느렸기 때문에 num_epoch은 10으로 설정했다.

- Attention 모델과, GLU cell을 사용했기 때문에 sequence length를 크게 잡아서 한 번에 많은 input을 받아드려도 괜찮을 것이라 생각하여 sequence length를 500으로 잡았고 성능이 올라갔다.
- **Dev 데이터 Accuracy: 0.88150**