REPORT

HANYANG UNIVERSITY

Assignment 2

[text-classification]

과	목	딥러닝및응용
담 당 교 수		최용석
학	과	컴퓨터소프트웨어학부
학	년	4학년
학	번	2018063218
이	름	최성진
제 출	일	2021.06.01

1. EDA

```
-----------------------------------EDA-----------------------------------
데이터의 개수는 총 40000개입니다
데이터의 예시와 Label은 다음과 같습니다.
 Data : <sos> big hair big boobs bad music and a giant safety pin these are
 Label: negative
  Data Average Length : 235
  Data Max Length     : 2494
```

※ 데이터 해석(파파고)

큰 머리 큰 가슴 나쁜 음악과 거대한 안전핀 이 단어들은 이 끔찍한 영화를 가장 잘 묘사할 수 있는 단어들이다. 나는 느끼한 공포 영화를 좋아한다. 그리고 나는 수백 편의 영화를 보았다. 그러나 이것은 최악의 상황이었다. 줄거리가 종이처럼 얇고 우스꽝스러운 것은 연기가 혐오스럽기 짝이 없다. 대본이 가장 웃기는 것은 c와의 마지막 대결이다.opp 그리고 그가 살인자가 누구인지를 어떻게 알아냈는지 그것은 단지 끔찍하게 쓰여진 옷들이 역겹고 재미있는 동등한 치수로써 많은 가슴 튕기는 남자들이 실제로 그것을 입었다는 것을 보여주는 컷티셔츠를 입는 것이다 그리고 음악은 거의 하루아침에 반복 재생되는 합성 쓰레기이다.쓰레기 같은 음악 가슴과 구급대원들이 시체를 치우는 장면 그리고 체육관은 여전히 사별을 위해 문을 닫지 않는다 농담은 제쳐두고 이 영화는 80년대 재앙을 되돌아보고 그때 모든 것이 얼마나 나빴는지 잘 웃는 것이 유일한 매력인 정말 나쁜 영화다.

해석했을 때 굉장히 부정적인 단어가 있지만 줄임말이나 등장인물 등 해석할 수 없는 용어가 며착지 있었습니다. 이에 따라 dropout을 통해 한 단어를 집중적으로 학습하지 않도록 하게 되었습니다.

2. Hyper Parameter

SEQUENCE_LENGTH = 200
EMBEDDING_DIM = 500
NUM_EPOCHS = 10

1) Sequence length:

당연히 sequence length가 짧은 것보다 긴 것이 성능이 잘나왔습니다.
1 Epoch 기준 accuracy가 0.5 -> 0.77로 다른 것을 수정하지 않아도 굉장히 빠르게 학습하는 것으로 나왔습니다.
다만, seqeunce length가 길수록 학습을 하는 속도는 느려졌습니다. 200 이상일 경우 colab에서 OOM이 발생하여 200에서 정했습니다.

2) Embedding dimension :

결정적 요인이라고 생각하지 않았지만 어느정도 효과가 있었습니다.
150에서 500차원으로 올렸을 때, 1 Epoch 기준 accuracy가 0.71에서 0.76으로 올랐습니다.

3) Epoch :

CNN과 다르게 굉장히 빠르게 수렴하는 것으로 확인했습니다. 따라서, 10번의 EPOCH중 적당한 숫자로 골라서 Early Stopping했습니다.

3. 코드설명

```
import numpy as np
import tensorflow as tf
import tensorflow.contrib.rnn as rnn_cell

# 배치로 학습하기 위해 data를 batch만큼 가져오는 모듈
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)

# zero_pad를 하기위해 사이즈 측정
def get_vocabulary_size(X):
    return max([max(x) for x in X]) + 1 # plus the 0th word

# vocabulary 가져오기
def fit_in_vocabulary(X, voc_size):
    return [[w for w in x if w < voc_size] for x in X]

# zero-padding
def zero_pad(X, seq_len):
    return np.array([x[:seq_len - 1] + [0] * max(seq_len - len(x), 1) for x in X])

tf.reset_default_graph()
def build_classifier(x, vocabulary_size, EMBEDDING_DIM, HIDDEN_SIZE):
    n_layers = 4
```

```
    # 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
    # 원래는 RNN cell을 사용했지만 평균 단어수가 200단어가 넘어가기 때문에 long term dependency를 해결하기 위해 LSTM cell을 사용하였습니다.
    cells = [rnn_cell.BasicLSTMCell(HIDDEN_SIZE) for layer in range(n_layers)]
    # 한가지 셀에 집중적으로 학습하는 것을 방지하기 위해 Dropout을 적용
    cells_drop = [rnn_cell.DropoutWrapper(cell, output_keep_prob=KEEP_PROB)for cell in cells]
    multi_layer_cell = rnn_cell.MultiRNNCell(cells_drop)

    rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, batch_embedded, dtype=tf.float32)

    # 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)
    hypothesis = tf.nn.softmax(logits)

    return hypothesis, logits

ckpt_path = "output/"

#Hyper Parameter
SEQUENCE_LENGTH = 200
EMBEDDING_DIM = 500
HIDDEN_SIZE = 150
BATCH_SIZE = 256
NUM_EPOCHS = 10
learning_rate = 0.001
KEEP_PROB = 0.75

np.load.__defaults__=(None, True, True, 'ASCII')

x_train = np.load("/content/drive/MyDrive/Colab Notebooks/assignment2/data/x_train.npy")
y_train = np.load("/content/drive/MyDrive/Colab
```

```
Notebooks/assignment2/data/y_train.npy")
x_test = np.load("/content/drive/MyDrive/Colab Notebooks/assignment2/data/x_test.npy")

# For EDA
# import json

# avg = 0
# max = -1
# for i in range(len(x_train)):
#     if max < len(x_train[i]):
#         max = len(x_train[i])
#     avg += len(x_train[i])/40000

#     index_to_word = json.load(open("/content/drive/MyDrive/Colab Notebooks/assignment2/data/dictionary.json"))
# print("————————————————————————EDA————————————————————————")
# print("데이터의 개수는 총 {}개입니다".format(len(x_train)))
# print("데이터의 예시와 Label은 다음과 같습니다.")
# print(" Data : "+''.join([index_to_word[str(index)]+" " for index in x_train[0]]))
# print(" Label: {}".format("negative" if y_train[0]==0 else "positive"))
# print("  Data Average Length : {}".format(int(avg)))
# print("  Data Max Length     : {}".format(max))

dev_num = len(x_train) // 4

x_dev = x_train[:dev_num]
y_dev = y_train[:dev_num]

x_train = x_train[dev_num:]
y_train = y_train[dev_num:]

y_train_one_hot = tf.squeeze(tf.one_hot(y_train, 2))
y_dev_one_hot = tf.squeeze(tf.one_hot(y_dev, 2))


# Sequences pre-processing
vocabulary_size = get_vocabulary_size(x_train)
x_dev = fit_in_vocabulary(x_dev, vocabulary_size)
x_train = zero_pad(x_train, SEQUENCE_LENGTH)
x_dev = zero_pad(x_dev, SEQUENCE_LENGTH)
```

```
batch_ph = tf.placeholder(tf.int32, [None, SEQUENCE_LENGTH], name='batch_ph')
target_ph = tf.placeholder(tf.float32, [None, 2], name='target_ph')

y_pred, logits = build_classifier(batch_ph, vocabulary_size, EMBEDDING_DIM, HIDDEN_SIZE)

loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=target_ph, logits=logits))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)

# Accuracy metric
is_correct = tf.equal(tf.argmax(y_pred, 1), tf.argmax(target_ph, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

total_batch = int(len(x_train)/BATCH_SIZE) if len(x_train)%BATCH_SIZE == 0 else int(len(x_train)/BATCH_SIZE) + 1
print(total_batch)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    print("학습시작")

    for epoch in range(NUM_EPOCHS):
        start = 0
        shuffled_idx = np.arange(0, len(x_train))
        np.random.shuffle(shuffled_idx)

        for i in range(total_batch):
            batch = batch_data(shuffled_idx, BATCH_SIZE, x_train, y_train_one_hot.eval(), i * BATCH_SIZE)
            sess.run(optimizer, feed_dict={batch_ph: batch[0], target_ph: batch[1]})

        dev_accuracy = accuracy.eval(feed_dict={batch_ph: x_dev, target_ph: np.asarray(y_dev_one_hot.eval())})
        print("Epoch : %d, Accuracy : %f" % (epoch+1,dev_accuracy))


        # for save
        saver = tf.train.Saver()
```

```
        saver.save(sess, ckpt_path)
        saver.restore(sess, ckpt_path)

    dev_accuracy = accuracy.eval(feed_dict={batch_ph: x_dev, target_ph:
np.asarray(y_dev_one_hot.eval())})
    print("dev 데이터 Accuracy: %f" % dev_accuracy)

    # 밑에는 건드리지 마세요
    x_test = fit_in_vocabulary(x_test, vocabulary_size)
    x_test = zero_pad(x_test, SEQUENCE_LENGTH)

    test_logits = y_pred.eval(feed_dict={batch_ph: x_test})
    np.save("result", test_logits)
```

4. 최종결과

```
학습시작
Epoch : 1, Accuracy : 0.776400
Epoch : 2, Accuracy : 0.814600
Epoch : 3, Accuracy : 0.809200
Epoch : 4, Accuracy : 0.815900
Epoch : 5, Accuracy : 0.809200
Epoch : 6, Accuracy : 0.776300
Epoch : 7, Accuracy : 0.808100
Epoch : 8, Accuracy : 0.808600
Epoch : 9, Accuracy : 0.807800
Epoch : 10, Accuracy : 0.806100
```

sequence length = 100

```
학습시작
Epoch : 1, Accuracy : 0.787700
Epoch : 2, Accuracy : 0.832100
Epoch : 3, Accuracy : 0.843900
Epoch : 4, Accuracy : 0.827400
Epoch : 5, Accuracy : 0.824800
Epoch : 6, Accuracy : 0.839400
Epoch : 7, Accuracy : 0.815800
Epoch : 8, Accuracy : 0.835500
Epoch : 9, Accuracy : 0.833400
Epoch : 10, Accuracy : 0.829600
dev 데이터 Accuracy: 0.831100
```

sequence length = 150

```
학습시작
Epoch : 1, Accuracy : 0.712900
Epoch : 2, Accuracy : 0.555900
Epoch : 3, Accuracy : 0.834600
Epoch : 4, Accuracy : 0.843400
Epoch : 5, Accuracy : 0.858400
Epoch : 6, Accuracy : 0.831000
Epoch : 7, Accuracy : 0.841000
Epoch : 8, Accuracy : 0.832500
Epoch : 9, Accuracy : 0.830700
Epoch : 10, Accuracy : 0.847600
dev 데이터 Accuracy: 0.848000
```

sequence length = 200

```
학습시작
Epoch : 1, Accuracy : 0.586000
INFO:tensorflow:Restoring parameters from output/
Epoch : 2, Accuracy : 0.649400
INFO:tensorflow:Restoring parameters from output/
Epoch : 3, Accuracy : 0.729100
INFO:tensorflow:Restoring parameters from output/
Epoch : 4, Accuracy : 0.807700
INFO:tensorflow:Restoring parameters from output/
Epoch : 5, Accuracy : 0.832200
INFO:tensorflow:Restoring parameters from output/
Epoch : 6, Accuracy : 0.836200
INFO:tensorflow:Restoring parameters from output/
Epoch : 7, Accuracy : 0.844100
INFO:tensorflow:Restoring parameters from output/
Epoch : 8, Accuracy : 0.843700
INFO:tensorflow:Restoring parameters from output/
Epoch : 9, Accuracy : 0.855900
INFO:tensorflow:Restoring parameters from output/
Epoch : 10, Accuracy : 0.853400
INFO:tensorflow:Restoring parameters from output/
dev 데이터 Accuracy: 0.855000
```

최종 하이퍼파라미터 튜닝