Term Assignment #2

Text Classification

```
한양대학교 컴퓨터소프트웨어학부 2015004911 임성준
딥러닝및응용(ITE4053, 12878) - 최용석 교수님
```

---

1. 개요

수업시간에 배운 다양한 방법들을 이용해서 text 분류 task 성능을 향상한다.

2. 개발환경

```
OS : macOS Big Sur 11.3
Language : Python 3.7.10
Source code editor : Visual Studio Code
Runtime environment : Google Colaboratory
```

3. 코드 설명

- model4student.py

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

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)

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

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

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

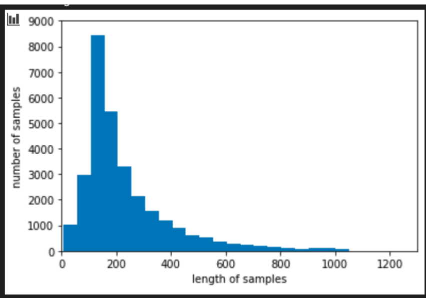
```python
def build_classifier(x, vocabulary_size, EMBEDDING_DIM, HIDDEN_SIZE,
ATTENTION_SIZE):
    # Embedding layer
    embeddings_var = tf.Variable(tf.random_uniform([vocabulary_size,
EMBEDDING_DIM], -1.0, 1.0), trainable=True)
    batch_embedded = tf.nn.embedding_lookup(embeddings_var, x)

    # 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_SIZE,
return_alphas=True)

    # Dropout
    drop = tf.nn.dropout(attention_output, keep_prob)

    # 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(drop, W), b)
    hypothesis = tf.nn.sigmoid(logits)

    return hypothesis, logits

def attention(inputs, attention_size, return_alphas=False):
    inputs_shape = inputs.shape
    sequence_length = inputs_shape[1].value
    hidden_size = inputs_shape[2].value

    # Trainable parameters
    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'):
        # Applying fully connected layer with non-linear activation to each of
the B*T timestamps;
        #  the shpae of 'v' is (B,T,D)*(D,A)=(B,T,A), where A=attention_size
        v = tf.tanh(tf.tensordot(inputs, W_omega, axes=1) + b_omega)

    # For each of the timestamps its vector of size A from 'v' is reduced with
'u' vector
    vu = tf.tensordot(v, u_omega, axes=1, name='vu') # (B,T) shape
```

```
    alphas = tf.nn.softmax(vu, name='alphas') # (B,T) shape

    # Output of (Bi-)RNN is reduced with attention vector; the result has (B,D)
shape
    output = tf.reduce_sum(inputs * tf.expand_dims(alphas, -1), 1)

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

ckpt_path = "output/"

SEQUENCE_LENGTH = 50
EMBEDDING_DIM = 100
HIDDEN_SIZE = 150
BATCH_SIZE = 32
ATTENTION_SIZE = 50
NUM_EPOCHS = 3
learning_rate = 0.001

# Load the data set
np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

x_train = np.load("data/x_train.npy")
y_train = np.load("data/y_train.npy")
x_test = np.load("data/x_test.npy")

np.load = np_load_old

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))

SEQUENCE_LENGTH = 700

# 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')
keep_prob = tf.placeholder(tf.float32, name='keep_prob')

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

loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=target_ph,
logits=logits))
optimizer = tf.train.AdamOptimizer(
    learning_rate=learning_rate,
    beta1=0.9,
    beta2=0.999,
    epsilon=1e-08,
    use_locking=False,
    name='Adam').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

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print("학습시작")

    for epoch in range(NUM_EPOCHS):
        start = 0
        avg_cost = 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)
            c, _ = sess.run([loss, optimizer], feed_dict={batch_ph: batch[0],
target_ph: batch[1], keep_prob: 0.8})
            avg_cost += c/total_batch
        # Epoch, loss 값 확인
```

```
        print('Epoch: ', '%d/%d' %(epoch+1, NUM_EPOCHS), 'Cost =',
'{:.9f}'.format(avg_cost))


        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()), keep_prob: 1})
    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, keep_prob: 1})
    np.save("result", test_logits)
```

- GRU cell 적용
- Attention 적용
- Dropout 적용
- Sequence length = 700
    - Data의 분포를 확인한 결과,
      *SEQUENCE_LENGTH = 50*은 약 **2%**, *SEQUENCE_LENGTH = 500*은 약 **92%**, *SEQUENCE_LENGTH = 700*은 약 **97%**
      의 data를 포함하는 것으로 보였다. 때문에 *SEQUENCE_LENGTH*를 700으로 늘려서 구현했다.




```python
def belowThreshold(thre, words) :
    cnt = 0
    for word in words:
        if(len(word) <= thre) :
            cnt += 1
    print(f"percentage of sample that shorter threshold {thre} : {round((cnt / len(words)) *
100,3)}")

print("threshold : 50")
belowThreshold(50,x_train)
print("threshold : 500")
belowThreshold(500,x_train)
print("threshold : 700")
belowThreshold(700,x_train)
print("threshold : 800")
belowThreshold(800,x_train)
```

```
threshold : 50
percentage of sample that shorter threshold 50 : 2.303
threshold : 500
percentage of sample that shorter threshold 500 : 91.997
threshold : 600
percentage of sample that shorter threshold 600 : 95.113
threshold : 700
percentage of sample that shorter threshold 700 : 96.947
threshold : 800
percentage of sample that shorter threshold 800 : 98.177
```

- Batch size = 32
  - Batch size를 줄여서 보다 세밀하게 data를 관찰할 수 있도록 한다.
- Hidden size를 너무 크게 늘리거나, Sequence length를 너무 크게 잡으면 Colab에서 GPU가 부족하다는 에러가 나왔다.

4. 실행 결과

- `기존 코드`

```
학습시작
2021-06-03 14:20:10.745781: I tens
Epoch:   1/10 Cost = 0.856422592
WARNING:tensorflow:From /usr/local
Instructions for updating:
Use standard file APIs to check fc
Epoch:   2/10 Cost = 0.726216217
Epoch:   3/10 Cost = 0.687364064
Epoch:   4/10 Cost = 0.602969589
Epoch:   5/10 Cost = 0.490745439
Epoch:   6/10 Cost = 0.374959936
Epoch:   7/10 Cost = 0.289341752
Epoch:   8/10 Cost = 0.212062182
Epoch:   9/10 Cost = 0.159786433
Epoch:  10/10 Cost = 0.105352999
dev 데이터 Accuracy: 0.722800
```

- Accuracy : 0.722
- loss : 0.105

- `GRU, Attention, Dropout 추가`

```
학습시작
2021-06-03 14:24:01.001958: I ter
Epoch:  1/10 Cost = 0.626364080
WARNING:tensorflow:From /usr/loca
Instructions for updating:
Use standard file APIs to check f
Epoch:  2/10 Cost = 0.499466959
Epoch:  3/10 Cost = 0.407364491
Epoch:  4/10 Cost = 0.332707145
Epoch:  5/10 Cost = 0.262839389
Epoch:  6/10 Cost = 0.193627140
Epoch:  7/10 Cost = 0.131728237
Epoch:  8/10 Cost = 0.077089170
Epoch:  9/10 Cost = 0.047205914
Epoch:  10/10 Cost = 0.030626440
dev 데이터 Accuracy: 0.746500
```

- Accuracy : 0.746
- loss : 0.030

- `Sequence length = 700`

```
학습시작
2021-06-03 14:53:55.390528: I ter
Epoch:  1/10 Cost = 0.580699016
WARNING:tensorflow:From /usr/loca
Instructions for updating:
Use standard file APIs to check f
Epoch:  2/10 Cost = 0.320384231
Epoch:  3/10 Cost = 0.218370579
Epoch:  4/10 Cost = 0.148811920
Epoch:  5/10 Cost = 0.096041826
Epoch:  6/10 Cost = 0.058967415
Epoch:  7/10 Cost = 0.034436308
Epoch:  8/10 Cost = 0.023412413
Epoch:  9/10 Cost = 0.015005232
Epoch:  10/10 Cost = 0.006230409
dev 데이터 Accuracy: 0.880700
```

- Accuracy : 0.880
- loss : 0.006

- `[Fianl code]Batch size = 32`

```
학습시작
2021-06-03 15:10:14.813669: I ten
Epoch:  1/10 Cost = 0.390650589
WARNING:tensorflow:From /usr/loca
Instructions for updating:
Use standard file APIs to check f
Epoch:  2/10 Cost = 0.189975363
Epoch:  3/10 Cost = 0.110325985
Epoch:  4/10 Cost = 0.054327581
Epoch:  5/10 Cost = 0.026857243
Epoch:  6/10 Cost = 0.016111297
Epoch:  7/10 Cost = 0.009143088
Epoch:  8/10 Cost = 0.008243516
Epoch:  9/10 Cost = 0.008381778
Epoch:  10/10 Cost = 0.005553094
dev 데이터 Accuracy: 0.890000
```

- **Accuracy : 0.890**
- loss : 0.005