TensorFlow\_RNN for 垃圾短信預測

import os

import re

import io

import requests

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from zipfile import ZipFile

from tensorflow.python.framework import ops

ops.reset\_default\_graph()

# Start a graph

sess = tf.Session()

# Set RNN parameters

epochs = 20

batch\_size = 250

max\_sequence\_length = 25

rnn\_size = 10

embedding\_size = 50

min\_word\_frequency = 10

learning\_rate = 0.0005

dropout\_keep\_prob = tf.placeholder(tf.float32)

# Download or open data

data\_dir = 'temp'

data\_file = 'text\_data.txt'

if not os.path.exists(data\_dir):

os.makedirs(data\_dir)

if not os.path.isfile(os.path.join(data\_dir, data\_file)):

zip\_url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00228/smsspamcollection.zip'

r = requests.get(zip\_url)

z = ZipFile(io.BytesIO(r.content))

file = z.read('SMSSpamCollection')

# Format Data

text\_data = file.decode()

text\_data = text\_data.encode('ascii', errors='ignore')

text\_data = text\_data.decode().split('\n')

# Save data to text file

with open(os.path.join(data\_dir, data\_file), 'w') as file\_conn:

for text in text\_data:

file\_conn.write("{}\n".format(text))

else:

# Open data from text file

text\_data = []

with open(os.path.join(data\_dir, data\_file), 'r') as file\_conn:

for row in file\_conn:

text\_data.append(row)

text\_data = text\_data[:-1]

text\_data = [x.split('\t') for x in text\_data if len(x) >= 1]

[text\_data\_target, text\_data\_train] = [list(x) for x in zip(\*text\_data)]

# Create a text cleaning function

def clean\_text(text\_string):

text\_string = re.sub(r'([^\s\w]|\_|[0-9])+', '', text\_string)

text\_string = " ".join(text\_string.split())

text\_string = text\_string.lower()

return text\_string

# Clean texts

text\_data\_train = [clean\_text(x) for x in text\_data\_train]

# Change texts into numeric vectors

vocab\_processor = tf.contrib.learn.preprocessing.VocabularyProcessor(max\_sequence\_length,

min\_frequency=min\_word\_frequency)

text\_processed = np.array(list(vocab\_processor.fit\_transform(text\_data\_train)))

# Shuffle and split data

text\_processed = np.array(text\_processed)

text\_data\_target = np.array([1 if x == 'ham' else 0 for x in text\_data\_target])

shuffled\_ix = np.random.permutation(np.arange(len(text\_data\_target)))

x\_shuffled = text\_processed[shuffled\_ix]

y\_shuffled = text\_data\_target[shuffled\_ix]

# Split train/test set

ix\_cutoff = int(len(y\_shuffled)\*0.80)

x\_train, x\_test = x\_shuffled[:ix\_cutoff], x\_shuffled[ix\_cutoff:]

y\_train, y\_test = y\_shuffled[:ix\_cutoff], y\_shuffled[ix\_cutoff:]

vocab\_size = len(vocab\_processor.vocabulary\_)

print("Vocabulary Size: {:d}".format(vocab\_size))

print("80-20 Train Test split: {:d} -- {:d}".format(len(y\_train), len(y\_test)))

# Create placeholders

x\_data = tf.placeholder(tf.int32, [None, max\_sequence\_length])

y\_output = tf.placeholder(tf.int32, [None])

# Create embedding

embedding\_mat = tf.Variable(tf.random\_uniform([vocab\_size, embedding\_size], -1.0, 1.0))

embedding\_output = tf.nn.embedding\_lookup(embedding\_mat, x\_data)

# Define the RNN cell

# tensorflow change >= 1.0, rnn is put into tensorflow.contrib directory. Prior version not test.

if tf.\_\_version\_\_[0] >= '1':

cell = tf.contrib.rnn.BasicRNNCell(num\_units=rnn\_size)

else:

cell = tf.nn.rnn\_cell.BasicRNNCell(num\_units=rnn\_size)

output, state = tf.nn.dynamic\_rnn(cell, embedding\_output, dtype=tf.float32)

output = tf.nn.dropout(output, dropout\_keep\_prob)

# Get output of RNN sequence

output = tf.transpose(output, [1, 0, 2])

last = tf.gather(output, int(output.get\_shape()[0]) - 1)

weight = tf.Variable(tf.truncated\_normal([rnn\_size, 2], stddev=0.1))

bias = tf.Variable(tf.constant(0.1, shape=[2]))

logits\_out = tf.matmul(last, weight) + bias

# Loss function

losses = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(logits=logits\_out, labels=y\_output)

loss = tf.reduce\_mean(losses)

accuracy = tf.reduce\_mean(tf.cast(tf.equal(tf.argmax(logits\_out, 1), tf.cast(y\_output, tf.int64)), tf.float32))

optimizer = tf.train.RMSPropOptimizer(learning\_rate)

train\_step = optimizer.minimize(loss)

init = tf.global\_variables\_initializer()

sess.run(init)

train\_loss = []

test\_loss = []

train\_accuracy = []

test\_accuracy = []

# Start training

for epoch in range(epochs):

# Shuffle training data

shuffled\_ix = np.random.permutation(np.arange(len(x\_train)))

x\_train = x\_train[shuffled\_ix]

y\_train = y\_train[shuffled\_ix]

num\_batches = int(len(x\_train)/batch\_size) + 1

# TO DO CALCULATE GENERATIONS ExACTLY

for i in range(num\_batches):

# Select train data

min\_ix = i \* batch\_size

max\_ix = np.min([len(x\_train), ((i+1) \* batch\_size)])

x\_train\_batch = x\_train[min\_ix:max\_ix]

y\_train\_batch = y\_train[min\_ix:max\_ix]

# Run train step

train\_dict = {x\_data: x\_train\_batch, y\_output: y\_train\_batch, dropout\_keep\_prob:0.5}

sess.run(train\_step, feed\_dict=train\_dict)

# Run loss and accuracy for training

temp\_train\_loss, temp\_train\_acc = sess.run([loss, accuracy], feed\_dict=train\_dict)

train\_loss.append(temp\_train\_loss)

train\_accuracy.append(temp\_train\_acc)

# Run Eval Step

test\_dict = {x\_data: x\_test, y\_output: y\_test, dropout\_keep\_prob:1.0}

temp\_test\_loss, temp\_test\_acc = sess.run([loss, accuracy], feed\_dict=test\_dict)

test\_loss.append(temp\_test\_loss)

test\_accuracy.append(temp\_test\_acc)

print('Epoch: {}, Test Loss: {:.2}, Test Acc: {:.2}'.format(epoch+1, temp\_test\_loss, temp\_test\_acc))

# Plot loss over time

epoch\_seq = np.arange(1, epochs+1)

plt.plot(epoch\_seq, train\_loss, 'k--', label='Train Set')

plt.plot(epoch\_seq, test\_loss, 'r-', label='Test Set')

plt.title('Softmax Loss')

plt.xlabel('Epochs')

plt.ylabel('Softmax Loss')

plt.legend(loc='upper left')

plt.show()

# Plot accuracy over time

plt.plot(epoch\_seq, train\_accuracy, 'k--', label='Train Set')

plt.plot(epoch\_seq, test\_accuracy, 'r-', label='Test Set')

plt.title('Test Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend(loc='upper left')

plt.show()