Don Baker
Class ID 6
Page 1

Deep Learning Lab 3

I. Introduction

This lab assignment has the following four parts:

- Implement the Text classification with CNN model with new data set which is not used in class
- Implement the Text classification with RNN model with new data set which is not used in class
- Implement the Text classification with LSTM model with new data set which is not used in class
- Compare results of CNN, RNN and LSTM for text classification (same data set for 3 models for comparison) and describe, which model is best for text classification based on your results

II. Objectives

The Objectives of this lab exercise were to run CNN, RNN, LSTM models and compare the results.

III. Approaches/Methods

In order to complete this exercise, the Python programing was broken into three programs that completes the CNN, RNN and LSTM analyses.

IV. Workflow/Parameters

The code for the Python files is attached at the end of the report.

V. Datasets

The datasets used for this assignment is DBpedia which contains the structured content created from data in a Wikipedia File. The datasets contain nearly 1.7 million entities for training and over 200,000 entrieds for testing.

VI. Evaluation and Discussion

The results for the CNN analysis is shown below:

accuracy = 0.07, global_step = 100, loss = .04, accuracy: 72.1%

The results for the RNN analysis is shown below:

step 100: accuracy = 0.15, global_step = 100, loss = .25, accuracy: 16.7%

The results for the LSTM analysis is shown below:

Test accuracy : 52.6 %

VII. Conclusion

CNN gives a better performance for text classification. LSTM results in the second best accuracy with RNN resulting in the worse accuracy.

```
# -*- coding: utf-8 -*-
"""
Created on Tue May  8 20:50:13 2018

@author: Don
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import sys

import numpy as np
import pandas
import tensorflow as tf

FLAGS = None

MAX_DOCUMENT_LENGTH = 100
N_FILTERS = 10
FILTER_SHAPE1 = [20, 256]
FILTER_SHAPE2 = [20, N_FILTERS]
POOLING_WINDOW = 4
POOLING_STRIDE = 2
MAX_LABEL = 15
CHARS_FEATURE = 'chars'  # Name of the input character feature.


def char_cnn_model(features, labels, mode):
  """Character level convolutional neural network model to predict classes."""
  features_onehot = tf.one_hot(features[CHARS_FEATURE], 256)
  input_layer = tf.reshape(
      features_onehot, [-1, MAX_DOCUMENT_LENGTH, 256, 1])
  with tf.variable_scope('CNN_Layer1'):
    # Apply Convolution filtering on input sequence.
    conv1 = tf.layers.conv2d(
        input_layer,
        filters=N_FILTERS,
        kernel_size=FILTER_SHAPE1,
        padding='VALID',
        # Add a ReLU for non linearity.
        activation=tf.nn.relu)
    # Max pooling across output of Convolution+Relu.
    pool1 = tf.layers.max_pooling2d(
        conv1,
        pool_size=POOLING_WINDOW,
        strides=POOLING_STRIDE,
        padding='SAME')
    # Transpose matrix so that n_filters from convolution becomes width.
    pool1 = tf.transpose(pool1, [0, 1, 3, 2])
  with tf.variable_scope('CNN_Layer2'):
    # Second level of convolution filtering.
    conv2 = tf.layers.conv2d(
        pool1,
        filters=N_FILTERS,
        kernel_size=FILTER_SHAPE2,
        padding='VALID')
    # Max across each filter to get useful features for classification.
    pool2 = tf.squeeze(tf.reduce_max(conv2, 1), squeeze_dims=[1])

  # Apply regular WX + B and classification.
  logits = tf.layers.dense(pool2, MAX_LABEL, activation=None)

  predicted_classes = tf.argmax(logits, 1)
  if mode == tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions={
            'class': predicted_classes,
            'prob': tf.nn.softmax(logits)
        })

  loss = tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=logits)
  if mode == tf.estimator.ModeKeys.TRAIN:
    optimizer = tf.train.AdamOptimizer(learning_rate=0.01)
    train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())
    return tf.estimator.EstimatorSpec(mode, loss=loss, train_op=train_op)
  eval_metric_ops = {
      'accuracy': tf.metrics.accuracy(
          labels=labels, predictions=predicted_classes)
  }
  return tf.estimator.EstimatorSpec(
      mode=mode, loss=loss, eval_metric_ops=eval_metric_ops)


def main(unused_argv):
  tf.logging.set_verbosity(tf.logging.INFO)

  # Prepare training and testing data
  dbpedia = tf.contrib.learn.datasets.load_dataset(
      'dbpedia', test_with_fake_data=FLAGS.test_with_fake_data, size='large')
  x_train = pandas.DataFrame(dbpedia.train.data)[1]
  y_train = pandas.Series(dbpedia.train.target)
  x_test = pandas.DataFrame(dbpedia.test.data)[1]
  y_test = pandas.Series(dbpedia.test.target)

  # Process vocabulary
  char_processor = tf.contrib.learn.preprocessing.ByteProcessor(
      MAX_DOCUMENT_LENGTH)
  x_train = np.array(list(char_processor.fit_transform(x_train)))
  x_test = np.array(list(char_processor.transform(x_test)))

  x_train = x_train.reshape([-1, MAX_DOCUMENT_LENGTH, 1, 1])
  x_test = x_test.reshape([-1, MAX_DOCUMENT_LENGTH, 1, 1])

  # Build model
  classifier = tf.estimator.Estimator(model_fn=char_cnn_model)

  # Train.
  train_input_fn = tf.estimator.inputs.numpy_input_fn(
      x={CHARS_FEATURE: x_train},
      y=y_train,
      batch_size=128,
      num_epochs=None,
      shuffle=True)
  classifier.train(input_fn=train_input_fn, steps=100)

  # Predict.
  test_input_fn = tf.estimator.inputs.numpy_input_fn(
      x={CHARS_FEATURE: x_test},
      y=y_test,
      num_epochs=1,
      shuffle=False)
  predictions = classifier.predict(input_fn=test_input_fn)
  y_predicted = np.array(list(p['class'] for p in predictions))
  y_predicted = y_predicted.reshape(np.array(y_test).shape)

  # Score with tensorflow.
  scores = classifier.evaluate(input_fn=test_input_fn)
  print('Accuracy: {0:f}'.format(scores['accuracy']))


if __name__ == '__main__':
  parser = argparse.ArgumentParser()
  parser.add_argument(
      '--test_with_fake_data',
      default=False,
      help='Test the example code with fake data.',
      action='store_true')
  FLAGS, unparsed = parser.parse_known_args()
  tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

```
# -*- coding: utf-8 -*-
"""
Created on Tue May  8 22:53:04 2018

@author: Don
"""

from tensorflow.contrib.rnn import BasicLSTMCell
from tensorflow.python.ops.rnn import bidirectional_dynamic_rnn as bi_rnn
import time
from utils.prepare_data import *

# Hyperparameter
MAX_DOCUMENT_LENGTH = 25
EMBEDDING_SIZE = 128
HIDDEN_SIZE = 64
ATTENTION_SIZE = 64
lr = 1e-3
BATCH_SIZE = 256
KEEP_PROB = 0.5
LAMBDA = 0.0001

MAX_LABEL = 15
epochs = 10

# load data
x_train, y_train = load_data("../dbpedia_data/dbpedia_csv/train.csv")
x_test, y_test = load_data("../dbpedia_data/dbpedia_csv/test.csv")

# data preprocessing
x_train, x_test, vocab, vocab_size = \
    data_preprocessing(x_train, x_test, MAX_DOCUMENT_LENGTH)
print(vocab_size)

# split dataset to test and dev
x_test, x_dev, y_test, y_dev, dev_size, test_size = \
    split_dataset(x_test, y_test, 0.1)
print("Validation size: ", dev_size)

graph = tf.Graph()
with graph.as_default():

    batch_x = tf.placeholder(tf.int32, [None, MAX_DOCUMENT_LENGTH])
    batch_y = tf.placeholder(tf.float32, [None, MAX_LABEL])
    keep_prob = tf.placeholder(tf.float32)

    embeddings_var = tf.Variable(tf.random_uniform([vocab_size, EMBEDDING_SIZE], -1.0, 1.0), 
trainable=True)
    batch_embedded = tf.nn.embedding_lookup(embeddings_var, batch_x)
    W = tf.Variable(tf.random_normal([HIDDEN_SIZE], stddev=0.1))
    # print(batch_embedded.shape)  # (?, 256, 100)

    rnn_outputs, _ = bi_rnn(BasicLSTMCell(HIDDEN_SIZE), BasicLSTMCell(HIDDEN_SIZE),
                            inputs=batch_embedded, dtype=tf.float32)
    # Attention
    fw_outputs = rnn_outputs[0]
    # print(fw_outputs.shape)
    bw_outputs = rnn_outputs[1]
    H = fw_outputs + bw_outputs  # (batch_size, seq_len, HIDDEN_SIZE)
    M = tf.tanh(H) # M = tanh(H)  (batch_size, seq_len, HIDDEN_SIZE)
    # print(M.shape)
    # alpha (bs * sl, 1)
    alpha = tf.nn.softmax(tf.matmul(tf.reshape(M, [-1, HIDDEN_SIZE]), tf.reshape(W, [-1, 1])))
    r = tf.matmul(tf.transpose(H, [0, 2, 1]), tf.reshape(alpha, [-1, MAX_DOCUMENT_LENGTH, 1])) # supposed 
to be (batch_size * HIDDEN_SIZE, 1)
    # print(r.shape)
    r = tf.squeeze(r)
    h_star = tf.tanh(r) # (batch , HIDDEN_SIZE
    # attention_output, alphas = attention(rnn_outputs, ATTENTION_SIZE, return_alphas=True)


    drop = tf.nn.dropout(h_star, keep_prob)
    # shape = drop.get_shape()
    # print(shape)

    # Fully connected layer•dense layer)
    W = tf.Variable(tf.truncated_normal([HIDDEN_SIZE, MAX_LABEL], stddev=0.1))
    b = tf.Variable(tf.constant(0., shape=[MAX_LABEL]))
    y_hat = tf.nn.xw_plus_b(drop, W, b)
    # print(y_hat.shape)

    # y_hat = tf.squeeze(y_hat)

    loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=y_hat, labels=batch_y))
    optimizer = tf.train.AdamOptimizer(learning_rate=lr).minimize(loss)

    # Accuracy metric
    prediction = tf.argmax(tf.nn.softmax(y_hat), 1)
    accuracy = tf.reduce_mean(tf.cast(tf.equal(prediction, tf.argmax(batch_y, 1)), tf.float32))

steps = 10001 # about 5 epoch
with tf.Session(graph=graph) as sess:
    sess.run(tf.global_variables_initializer())
    print("Initialized! ")

    print("Start trainning")
    start = time.time()
    for e in range(epochs):

        epoch_start = time.time()
        print("Epoch %d start !" % (e + 1))
        for x_batch, y_batch in fill_feed_dict(x_train, y_train, BATCH_SIZE):
            fd = {batch_x: x_batch, batch_y: y_batch, keep_prob: KEEP_PROB}
            l, _, acc = sess.run([loss, optimizer, accuracy], feed_dict=fd)

        epoch_finish = time.time()
        print("Validation accuracy and loss: ", sess.run([accuracy, loss], feed_dict={
            batch_x: x_dev,
            batch_y: y_dev,
            keep_prob: 1.0
        }))

    print("Training finished, time consumed : ", time.time() - start, " s")
    print("start predicting:  \n")
    test_accuracy = sess.run([accuracy], feed_dict={batch_x: x_test, batch_y: y_test, keep_prob: 1})
    print("Test accuracy : %f %%" % (test_accuracy[0] * 100))
```

```
# -*- coding: utf-8 -*-
"""
Created on Tue May  8 20:50:13 2018

@author: Don
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import sys

import numpy as np
import pandas
import tensorflow as tf

FLAGS = None

MAX_DOCUMENT_LENGTH = 100
N_FILTERS = 10
FILTER_SHAPE1 = [20, 256]
FILTER_SHAPE2 = [20, N_FILTERS]
POOLING_WINDOW = 4
POOLING_STRIDE = 2
MAX_LABEL = 15
CHARS_FEATURE = 'chars'  # Name of the input character feature.


def char_cnn_model(features, labels, mode):
  """Character level convolutional neural network model to predict classes."""
  features_onehot = tf.one_hot(features[CHARS_FEATURE], 256)
  input_layer = tf.reshape(
      features_onehot, [-1, MAX_DOCUMENT_LENGTH, 256, 1])
  with tf.variable_scope('CNN_Layer1'):
    # Apply Convolution filtering on input sequence.
    conv1 = tf.layers.conv2d(
        input_layer,
        filters=N_FILTERS,
        kernel_size=FILTER_SHAPE1,
        padding='VALID',
        # Add a ReLU for non linearity.
        activation=tf.nn.relu)
    # Max pooling across output of Convolution+Relu.
    pool1 = tf.layers.max_pooling2d(
        conv1,
        pool_size=POOLING_WINDOW,
        strides=POOLING_STRIDE,
        padding='SAME')
    # Transpose matrix so that n_filters from convolution becomes width.
    pool1 = tf.transpose(pool1, [0, 1, 3, 2])
  with tf.variable_scope('CNN_Layer2'):
    # Second level of convolution filtering.
    conv2 = tf.layers.conv2d(
        pool1,
        filters=N_FILTERS,
        kernel_size=FILTER_SHAPE2,
        padding='VALID')
    # Max across each filter to get useful features for classification.
    pool2 = tf.squeeze(tf.reduce_max(conv2, 1), squeeze_dims=[1])

  # Apply regular WX + B and classification.
  logits = tf.layers.dense(pool2, MAX_LABEL, activation=None)

  predicted_classes = tf.argmax(logits, 1)
  if mode == tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions={
            'class': predicted_classes,
            'prob': tf.nn.softmax(logits)
        })

  loss = tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=logits)
  if mode == tf.estimator.ModeKeys.TRAIN:
    optimizer = tf.train.AdamOptimizer(learning_rate=0.01)
    train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())
    return tf.estimator.EstimatorSpec(mode, loss=loss, train_op=train_op)
  eval_metric_ops = {
      'accuracy': tf.metrics.accuracy(
          labels=labels, predictions=predicted_classes)
  }
  return tf.estimator.EstimatorSpec(
      mode=mode, loss=loss, eval_metric_ops=eval_metric_ops)


def main(unused_argv):
  tf.logging.set_verbosity(tf.logging.INFO)

  # Prepare training and testing data
  dbpedia = tf.contrib.learn.datasets.load_dataset(
      'dbpedia', test_with_fake_data=FLAGS.test_with_fake_data, size='large')
  x_train = pandas.DataFrame(dbpedia.train.data)[1]
  y_train = pandas.Series(dbpedia.train.target)
  x_test = pandas.DataFrame(dbpedia.test.data)[1]
  y_test = pandas.Series(dbpedia.test.target)

  # Process vocabulary
  char_processor = tf.contrib.learn.preprocessing.ByteProcessor(
      MAX_DOCUMENT_LENGTH)
  x_train = np.array(list(char_processor.fit_transform(x_train)))
  x_test = np.array(list(char_processor.transform(x_test)))

  x_train = x_train.reshape([-1, MAX_DOCUMENT_LENGTH, 1, 1])
  x_test = x_test.reshape([-1, MAX_DOCUMENT_LENGTH, 1, 1])

  # Build model
  classifier = tf.estimator.Estimator(model_fn=char_cnn_model)

  # Train.
  train_input_fn = tf.estimator.inputs.numpy_input_fn(
      x={CHARS_FEATURE: x_train},
      y=y_train,
      batch_size=128,
      num_epochs=None,
      shuffle=True)
  classifier.train(input_fn=train_input_fn, steps=100)

  # Predict.
  test_input_fn = tf.estimator.inputs.numpy_input_fn(
      x={CHARS_FEATURE: x_test},
      y=y_test,
      num_epochs=1,
      shuffle=False)
  predictions = classifier.predict(input_fn=test_input_fn)
  y_predicted = np.array(list(p['class'] for p in predictions))
  y_predicted = y_predicted.reshape(np.array(y_test).shape)

  # Score with tensorflow.
  scores = classifier.evaluate(input_fn=test_input_fn)
  print('Accuracy: {0:f}'.format(scores['accuracy']))


if __name__ == '__main__':
  parser = argparse.ArgumentParser()
  parser.add_argument(
      '--test_with_fake_data',
      default=False,
      help='Test the example code with fake data.',
      action='store_true')
  FLAGS, unparsed = parser.parse_known_args()
  tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```