ECE471, Selected Topics in Machine Learning – Assignment 5
AG News Dataset Classifier
Nikola Janjušević
October 9, 2018

Result

Test Set Accuracy: 90.32%

Program:

```
import os
import pandas
import collections
import numpy as np
from tqdm import tqdm
import tensorflow as tf

# for tensorboard
logs_path = "./tf_logs/"
# disable tensorflow warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

num_classes = 4
embedding_dim = 32
VOCAB_SIZE = 10000

# hyper-parameters
BATCH_SIZE = 128
NUM_EPOCHS = 2
display_epoch = 1
LEARNING_RATE = .01
# regularization parameters
drop_prob = 0.25
reg_scale = 1e-6

def main():

    class Data(object):
        def __init__(self):
            self.data_train, self.y_train, all_words \
                = get_data_csv('ag_news_csv/train.csv')
            data_test, y_test, _ = get_data_csv('ag_news_csv/test.csv')
            # split test into test and validation set
            [(self.data_test, self.data_val), (self.y_test, self.y_val)] \
                = [np.split(xy,2) for xy in [data_test, y_test]]

            # build dictionaries of test words, num limited to vocab_size
            self.dictionary, self.reversed_dictionary \
                = build_dataset(all_words, VOCAB_SIZE)

            self.train_size = self.y_train.shape[0]

            # encodes strings by dictionary number
            [(self.x_train,longest_str), (self.x_val,_), (self.x_test,_)] \
                = [code_data(x, self.dictionary, max_len=200) \
                        for x in  \
                            [self.data_train, self.data_val, self.data_test] ]
            # maximum sentence length
            self.max_len = longest_str

        def get_batch(self):
            choices = np.random.choice(self.train_size, size=BATCH_SIZE)
            return self.x_train[choices], self.y_train[choices,:]

    print("constructing dataset...")
    data = Data()
    print("done.")

    x = tf.placeholder(tf.int32 , shape=[None , data.max_len])
    y = tf.placeholder(tf.int32 , shape=[None , num_classes])
    embeddings = tf.Variable(
        tf.random_uniform([VOCAB_SIZE , embedding_dim], -1.0, 1.0)
    )
    phase = tf.placeholder(tf.bool) # is_training
    lr = tf.placeholder(tf.float32) # learning rate [not used in this program]

    def f(x):
        x = tf.nn.embedding_lookup(embeddings , x)
        x = tf.layers.flatten(x)
        x = fc_layer(x, 8, is_training=phase)
        x = tf.layers.dense(x, num_classes)
        return x

    # models
    logits = f(x)
    prediction = tf.nn.softmax(logits)

    correct_pred = tf.equal(tf.argmax(prediction , 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_pred , tf.float32))

    # LOSS
    update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
    with tf.control_dependencies(update_ops):
        loss = tf.reduce_mean( tf.losses.softmax_cross_entropy(y, logits) ) \
            + tf.reduce_mean( tf.losses.get_regularization_loss() )
        optim = tf.train.AdamOptimizer(learning_rate=lr).minimize(loss)

    init = tf.global_variables_initializer()

    # Create a summary to monitor cost tensor
    tf.summary.scalar("loss", loss)
    # Create a summary to monitor accuracy tensor
    tf.summary.scalar("accuracy", accuracy)
    # Merge all summaries into a single op
    merged_summary_op = tf.summary.merge_all()

    with tf.Session() as sess:
        sess.run(init)
        # op to write logs to Tensorboard
        summary_writer = tf.summary.FileWriter(logs_path ,
            graph=tf.get_default_graph())
        learning_rate = LEARNING_RATE
        # training
        for epoch in range(NUM_EPOCHS):
            avg_cost = 0.
            num_batches = int( np.ceil( data.train_size / BATCH_SIZE ) )

            for i in tqdm(range(num_batches)):
                xb, yb = data.get_batch()
                fd = {x: xb, y: yb, phase: True , lr: learning_rate}
                loss_np , _, summary \
                    = sess.run([loss , optim , merged_summary_op], feed_dict=fd)
                # logs every batch
                summary_writer.add_summary(summary , epoch * num_batches + i)
                avg_cost  += loss_np/num_batches

            # Display logs per epoch step
            if (epoch+1) % display_epoch == 0:
                print("Epoch:", ’%02d’ % (epoch+1),
                    "cost=", "{:.6f}".format(avg_cost))

            print(’Validation Set Accuracy:’,
                accuracy.eval({x: data.x_val , y: data.y_val , phase: False}))

        # Test the model on separate data
        print(’Test Set Accuracy:’,
            accuracy.eval({x: data.x_test , y: data.y_test , phase: False}))

        print("Run the command line:\n--> tensorboard --logdir=./tf_logs ")

# ------------------------------------------------
# -------------- MODEL FUNCTIONS -----------------
# -----------------------------------------------

# --- CONV LAYER WRAPPER --- w/ L2 regularization
def conv_layer(input, filters, kernel_size, strides=2, is_training=True):
    x = tf.layers.conv2d(
        input, filters, kernel_size, strides=strides, padding='same',
        kernel_regularizer=tf.contrib.layers.l2_regularizer(scale=reg_scale)
    )
    return x

# --- FULLY CONNECTED LAYER WRAPPER ---
# matmul -> BN -> relu -> dropout
def fc_layer(input, units, is_training=True):
    x = tf.layers.dense(
        input, units,
        kernel_regularizer=tf.contrib.layers.l2_regularizer(scale=reg_scale)
    )
    x = tf.layers.batch_normalization(x, training=is_training, renorm=True)
    x = tf.nn.relu6(x)
    x = tf.layers.dropout(x, rate=drop_prob, training=is_training)
    return x

# ----------------------------------------------
# ---------- DATA BUILDING FUNCTIONS ----------
# ----------------------------------------------

# https://stackoverflow.com/questions/38592324/one-hot-encoding-using-numpy/38592416
def get_one_hot(targets, nb_classes):
    res = np.eye(nb_classes)[np.array(targets).reshape(-1)]
    return res.reshape(list(targets.shape)+[nb_classes])

def get_data_csv(pathname):
    # data from csv
    df = pandas.read_csv(pathname, index_col=False, \
        header=None, names=['label', 'headline', 'description'],
        quotechar='"', doublequote=True, lineterminator='\n')
    # joining headline and description into one string
    # https://stackoverflow.com/questions/39571832/how-to-row-wise-concatenate-several-columns-containing-strings
    df['cat'] = df[df.columns[1:]].apply(tuple, axis=1).str.join(' ')
    # puts all the data into one list
    all_words = "".join(df['cat'].tolist()).lower().split()
    # shuffling
    s = np.arange(len(df['label']))
    np.random.shuffle(s)
    x = np.array(df['cat'])[s]
    y = get_one_hot(np.array(df['label']) - 1, num_classes)[s,:]
    return x, y, all_words

# makes dictionary mapping of words to unique integer id
# http://adventuresinmachinelearning.com/word2vec-tutorial-tensorflow/
def build_dataset(words, n_words):
    """Process raw inputs into a dataset."""
    count = [['UNK', -1]]
    count.extend(collections.Counter(words).most_common(n_words - 1))
    dictionary = dict()
    for word, _ in count:
        dictionary[word] = len(dictionary)
    data = list()
    unk_count = 0
    for word in words:
        if word in dictionary:
            index = dictionary[word]
        else:
            index = 0  # dictionary['UNK']
            unk_count += 1
        data.append(index)
    count[0][1] = unk_count
    reversed_dictionary = dict(zip(dictionary.values(), dictionary.keys()))
    return dictionary, reversed_dictionary

# codes any string into an array of integers corresponding to the mapping
# provided by the dictionary. Pads / clips strings to max_len number of words
# if provided.
def code_data(data, dictionary, max_len=None):
    coded_data = []
    longest_str = 0
    for i in range(data.shape[0]):
        word_list = data[i].lower().split()
        for j in range(len(word_list)):
            try:
                key = dictionary[word_list[j]]
            except KeyError:
                key = 0
            word_list[j] = key
        if len(word_list)>longest_str:
            longest_str = len(word_list)
        coded_data.append(word_list)

    if max_len is not None:
        longest_str = max_len

    cd = np.zeros((data.shape[0], longest_str))

    for i in range(len(coded_data)):
        cd[i,:] = np.pad( np.asarray(coded_data[i][:longest_str]), \
            ( 0, longest_str-len(coded_data[i][:longest_str]) ), 'constant')
    return cd, longest_str

if __name__ == "__main__":
    main()
```