ECE194N HW 3

RNN Report

March 8, 2019

Student: Erik Rosten
Perm Number: 7143571
Email: erosten@ucsb.edu

Department of Electrical and Computer Engineering, UCSB

a. Visualizing the Data

The code for visualizing the data is below

```
import reader
import numpy as np


raw_data = reader.ptb_raw_data('simple-examples/data/')
train_data, valid_data, test_data, _, word_dict = raw_data

train_data = np.array(train_data)
valid_data = np.array(valid_data)
test_data = np.array(test_data)
word_dict = dict(zip(word_dict.values(),word_dict.keys()))



sentence = ''
keys = []
sentenceNum = 1
for j in range(valid_data.shape[0]):
   ind = valid_data[j]
   if (word_dict[ind] == '<eos>'):
      print('\n\n')
      print('Sentence {}'.format(sentenceNum))
      print('\"{}\" {} '.format(sentence, np.array(keys)))
      sentence = ''
      keys = []
      sentenceNum = sentenceNum + 1
      if (sentenceNum == 11):
         break
      continue

   if (j > 0):
      sentence = sentence + ' '

   sentence = sentence + word_dict[ind]
   keys.append(ind)
```

where I have modified the reader code to return the word_dict. Running this code returns

```

Sentence 1
"consumers may want to move their telephones a little closer to the tv set" [1132 93 358
    5 329 51 9836  6  326 2476   5    0  662  388]



Sentence 2
" <unk> <unk> watching abc 's monday night football can now vote during <unk> for the
    greatest play in N years from among four or five <unk> <unk>" [ 1 1 2974 2158 9 381
    1068 2347 89 99 847 198 1 11
    0 3383 1119   7    3   72   20  211  346   36  258    1    1]

```

```


Sentence 3
" two weeks ago viewers of several nbc <unk> consumer segments started calling a N number for advice on various <unk> issues" [ 75 422 195 3917 4 249 1795 1 580 3528 892 2374 6 3
  297   11 2709   16 1186    1  250]



Sentence 4
" and the new syndicated reality show hard copy records viewers ’ opinions for possible airing on the next day ’s show" [ 8 0 35 9922 3747 464 710 2998 2037 3917 134 6145 11 494
 5894   16    0  130  272    9  464]



Sentence 5
" interactive telephone technology has taken a new leap in <unk> and television programmers are racing to exploit the possibilities" [9958 732 503 30 641 6 35 6498 7 1 8  761 9967  26
 6587    5 6415    0 6574]



Sentence 6
" eventually viewers may grow <unk> with the technology and <unk> the cost" [1413 3917 93 1552 1 22 0 503    8    1    0  361]



Sentence 7
" but right now programmers are figuring that viewers who are busy dialing up a range of services may put down their <unk> control <unk> and stay <unk>" [ 29 382 99 9967 26 7428 10 3917 56 26 3248 8846 52 6
  880    4  323   93  335  118   51    1  350    1    8 1337    1]



Sentence 8
" we ’ve been spending a lot of time in los angeles talking to tv production people says mike parks president of call interactive which supplied technology for both abc sports and nbc ’s consumer minutes" [ 64 573 58 508 6 581 4 103 7 639 747 1921 5 662
  359  108   44 5458 6149  70    4  786 9958   41 7746 503   11  179
 2158 1259    8 1795    9  580 1495]



Sentence 9
" with the competitiveness of the television market these days everyone is looking for a way to get viewers more excited" [ 22 0 9643 4 0 761 47 144 171 1376 13 735   11    6
  229    5  188 3917   45 9684]

```

```


Sentence 10
" one of the leaders behind the expanded use of N numbers is call interactive a joint venture of giants american express co. and american telephone & telegraph co" [ 54 4 0 815 1116 0 2439 269 4 3 1619 13 786 9958
   6  795  818    4 2172 140 1021  95    8  140  732   82 3133 570]
```

b. Training the model

Modifying the given code to save loss and running the code gives

```
Epoch: 1 Learning rate: 1.000
0.004 perplexity: 6695.571 speed: 8623 wps
0.104 perplexity: 841.522 speed: 21685 wps
0.204 perplexity: 618.627 speed: 22346 wps
0.304 perplexity: 500.656 speed: 22739 wps
0.404 perplexity: 432.179 speed: 22809 wps
0.504 perplexity: 387.865 speed: 22326 wps
0.604 perplexity: 349.699 speed: 22334 wps
0.703 perplexity: 323.348 speed: 22494 wps
0.803 perplexity: 302.308 speed: 22624 wps
0.903 perplexity: 282.956 speed: 22716 wps
Epoch: 1 Train Perplexity: 268.422
Epoch: 1 Valid Perplexity: 179.301
Epoch: 2 Learning rate: 1.000
0.004 perplexity: 208.967 speed: 24266 wps
0.104 perplexity: 149.908 speed: 21422 wps
0.204 perplexity: 157.417 speed: 22361 wps
0.304 perplexity: 152.346 speed: 22704 wps
0.404 perplexity: 149.530 speed: 22890 wps
0.504 perplexity: 147.265 speed: 23018 wps
0.604 perplexity: 142.691 speed: 23108 wps
0.703 perplexity: 140.512 speed: 23173 wps
0.803 perplexity: 138.554 speed: 23199 wps
0.903 perplexity: 134.969 speed: 23223 wps
Epoch: 2 Train Perplexity: 132.864
Epoch: 2 Valid Perplexity: 142.850
Epoch: 3 Learning rate: 1.000
0.004 perplexity: 143.674 speed: 23878 wps
0.104 perplexity: 104.587 speed: 23382 wps
0.204 perplexity: 113.822 speed: 23413 wps
0.304 perplexity: 111.091 speed: 23445 wps
0.404 perplexity: 110.110 speed: 23460 wps
0.504 perplexity: 109.426 speed: 23489 wps
0.604 perplexity: 106.796 speed: 23493 wps
0.703 perplexity: 106.130 speed: 23490 wps
0.803 perplexity: 105.512 speed: 23481 wps
0.903 perplexity: 103.273 speed: 23481 wps
Epoch: 3 Train Perplexity: 102.256
Epoch: 3 Valid Perplexity: 132.594
Epoch: 4 Learning rate: 1.000
0.004 perplexity: 117.351 speed: 22860 wps
```

```
0.104 perplexity: 85.203 speed: 23450 wps
0.204 perplexity: 93.827 speed: 23464 wps
0.304 perplexity: 91.566 speed: 23470 wps
0.404 perplexity: 91.025 speed: 23452 wps
0.504 perplexity: 90.677 speed: 23475 wps
0.604 perplexity: 88.782 speed: 23485 wps
0.703 perplexity: 88.584 speed: 23494 wps
0.803 perplexity: 88.384 speed: 23501 wps
0.903 perplexity: 86.737 speed: 23512 wps
Epoch: 4 Train Perplexity: 86.164
Epoch: 4 Valid Perplexity: 128.316
Epoch: 5 Learning rate: 0.500
0.004 perplexity: 101.607 speed: 23739 wps
0.104 perplexity: 71.511 speed: 23541 wps
0.204 perplexity: 77.490 speed: 23531 wps
0.304 perplexity: 74.558 speed: 23525 wps
0.404 perplexity: 73.472 speed: 23517 wps
0.504 perplexity: 72.606 speed: 23505 wps
0.604 perplexity: 70.490 speed: 23519 wps
0.703 perplexity: 69.718 speed: 23540 wps
0.803 perplexity: 68.935 speed: 23554 wps
0.903 perplexity: 67.016 speed: 23546 wps
Epoch: 5 Train Perplexity: 65.996
Epoch: 5 Valid Perplexity: 119.056
Epoch: 6 Learning rate: 0.250
0.004 perplexity: 82.694 speed: 23130 wps
0.104 perplexity: 59.016 speed: 23488 wps
0.204 perplexity: 64.249 speed: 23440 wps
0.304 perplexity: 61.726 speed: 23426 wps
0.404 perplexity: 60.722 speed: 23531 wps
0.504 perplexity: 59.913 speed: 23481 wps
0.604 perplexity: 58.063 speed: 23460 wps
0.703 perplexity: 57.312 speed: 23466 wps
0.803 perplexity: 56.502 speed: 23462 wps
0.903 perplexity: 54.753 speed: 23464 wps
Epoch: 6 Train Perplexity: 53.782
Epoch: 6 Valid Perplexity: 118.198
Epoch: 7 Learning rate: 0.125
0.004 perplexity: 72.797 speed: 24196 wps
0.104 perplexity: 52.220 speed: 23343 wps
0.204 perplexity: 57.053 speed: 23377 wps
0.304 perplexity: 54.856 speed: 23410 wps
0.404 perplexity: 53.933 speed: 23446 wps
0.504 perplexity: 53.173 speed: 23493 wps
0.604 perplexity: 51.488 speed: 23498 wps
0.703 perplexity: 50.781 speed: 23494 wps
0.803 perplexity: 49.991 speed: 23476 wps
0.903 perplexity: 48.373 speed: 23483 wps
Epoch: 7 Train Perplexity: 47.455
Epoch: 7 Valid Perplexity: 119.343
Epoch: 8 Learning rate: 0.062
0.004 perplexity: 68.244 speed: 23580 wps
0.104 perplexity: 48.842 speed: 23515 wps
0.204 perplexity: 53.414 speed: 23490 wps
```

```
0.304 perplexity: 51.359 speed: 23501 wps
0.404 perplexity: 50.514 speed: 23483 wps
0.504 perplexity: 49.779 speed: 23483 wps
0.604 perplexity: 48.176 speed: 23496 wps
0.703 perplexity: 47.481 speed: 23490 wps
0.803 perplexity: 46.703 speed: 23473 wps
0.903 perplexity: 45.150 speed: 23460 wps
Epoch: 8 Train Perplexity: 44.260
Epoch: 8 Valid Perplexity: 120.259
Epoch: 9 Learning rate: 0.031
0.004 perplexity: 66.016 speed: 24117 wps
0.104 perplexity: 47.151 speed: 22937 wps
0.204 perplexity: 51.539 speed: 22904 wps
0.304 perplexity: 49.519 speed: 22993 wps
0.404 perplexity: 48.710 speed: 23037 wps
0.504 perplexity: 48.002 speed: 23073 wps
0.604 perplexity: 46.448 speed: 23068 wps
0.703 perplexity: 45.756 speed: 23065 wps
0.803 perplexity: 44.982 speed: 23085 wps
0.903 perplexity: 43.462 speed: 23086 wps
Epoch: 9 Train Perplexity: 42.586
Epoch: 9 Valid Perplexity: 120.535
Epoch: 10 Learning rate: 0.016
0.004 perplexity: 64.642 speed: 23044 wps
0.104 perplexity: 46.198 speed: 22610 wps
0.204 perplexity: 50.520 speed: 22409 wps
0.304 perplexity: 48.504 speed: 22626 wps
0.404 perplexity: 47.706 speed: 22884 wps
0.504 perplexity: 47.015 speed: 23025 wps
0.604 perplexity: 45.492 speed: 22952 wps
0.703 perplexity: 44.809 speed: 22868 wps
0.803 perplexity: 44.042 speed: 22834 wps
0.903 perplexity: 42.541 speed: 22901 wps
Epoch: 10 Train Perplexity: 41.671
Epoch: 10 Valid Perplexity: 120.385
Epoch: 11 Learning rate: 0.008
0.004 perplexity: 63.739 speed: 23199 wps
0.104 perplexity: 45.621 speed: 23363 wps
0.204 perplexity: 49.911 speed: 23356 wps
0.304 perplexity: 47.912 speed: 23224 wps
0.404 perplexity: 47.124 speed: 23025 wps
0.504 perplexity: 46.446 speed: 23045 wps
0.604 perplexity: 44.942 speed: 23097 wps
0.703 perplexity: 44.270 speed: 23078 wps
0.803 perplexity: 43.512 speed: 22958 wps
0.903 perplexity: 42.023 speed: 22952 wps
Epoch: 11 Train Perplexity: 41.159
Epoch: 11 Valid Perplexity: 120.067
Epoch: 12 Learning rate: 0.004
0.004 perplexity: 63.187 speed: 23541 wps
0.104 perplexity: 45.286 speed: 23184 wps
0.204 perplexity: 49.561 speed: 23173 wps
0.304 perplexity: 47.580 speed: 23181 wps
0.404 perplexity: 46.801 speed: 23073 wps
```

```
0.504 perplexity: 46.130 speed: 23139 wps
0.604 perplexity: 44.639 speed: 23209 wps
0.703 perplexity: 43.972 speed: 23263 wps
0.803 perplexity: 43.221 speed: 23306 wps
0.903 perplexity: 41.740 speed: 23336 wps
Epoch: 12 Train Perplexity: 40.881
Epoch: 12 Valid Perplexity: 119.783
Epoch: 13 Learning rate: 0.002
0.004 perplexity: 62.876 speed: 23633 wps
0.104 perplexity: 45.096 speed: 23555 wps
0.204 perplexity: 49.362 speed: 23525 wps
0.304 perplexity: 47.394 speed: 23554 wps
0.404 perplexity: 46.623 speed: 23563 wps
0.504 perplexity: 45.958 speed: 23584 wps
0.604 perplexity: 44.475 speed: 23578 wps
0.703 perplexity: 43.813 speed: 23566 wps
0.803 perplexity: 43.065 speed: 23570 wps
0.903 perplexity: 41.589 speed: 23577 wps
Epoch: 13 Train Perplexity: 40.733
Epoch: 13 Valid Perplexity: 119.608
Test Perplexity: 115.036
Test Loss: 4.745
```

The losses from the text files are plotted with the code below

```python
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import MaxNLocator



def get_csv_data(filename):
    return np.loadtxt(filename,dtype=’float32’,delimiter=’,’)


def plot_csv(filename, y_label, max_or_min):
    data = get_csv_data(filename)
    epochs = data[:,0]
    acc = data[:,1]
    string = ’’
    if (max_or_min == ’max’):
        index = np.argmax(data[:,1])
        string = ’Maximum’
    else:
        index = np.argmin(data[:,1])
        string = ’Minimum’
    # add plot and max/min point
    plt.plot(epochs, acc, linestyle = ’--’, marker = ’o’, color = ’b’)
    plt.plot(epochs[index], acc[index], marker = ’o’, color=’r’)
    # add vertical line
    ax = plt.gca()
    y_min, y_max = ax.get_ylim()
    plt.vlines(epochs[index], y_min, acc[index], linestyle=’dashed’)
    plt.ylim((y_min, y_max))
    # set plot title
    plt.title('{} vs Epoch'.format(y_label))
    # delete x axis ticks that are within one of the max epoch
    x_tick_arr = np.array(list(ax.get_xticks())).astype(int)
    close_indices = np.where(np.logical_or(x_tick_arr==np.int(epochs[index] - 1), x_tick_arr==np.int(epochs[index] + 1)))[0]
    ax.set_xticks(np.delete(x_tick_arr, close_indices))
    # add max index to x axis
    ax.set_xticks(list(ax.get_xticks()) + [epochs[index]])
    # set plot x/y labels
    plt.xlabel('Epoch')
    plt.ylabel('{}'.format(y_label))
    # add legend
    string = string + ' ' + y_label + ': ' + str(acc[index])
    plt.legend(['_nolegend_', string], loc = 'best')
    # make the max point red
    x_min, x_max = ax.get_xlim()
    plt.xlim((0, epochs[-1:] + 1))
    ax.get_xticklabels()[-1].set_color('red')

    plt.show()


plot_csv('epoch_loss_train.txt', 'Training Loss', 'min')
plot_csv('epoch_loss_val.txt', 'Validation Loss', 'min')
```

Calling plotter.py gives the plots below

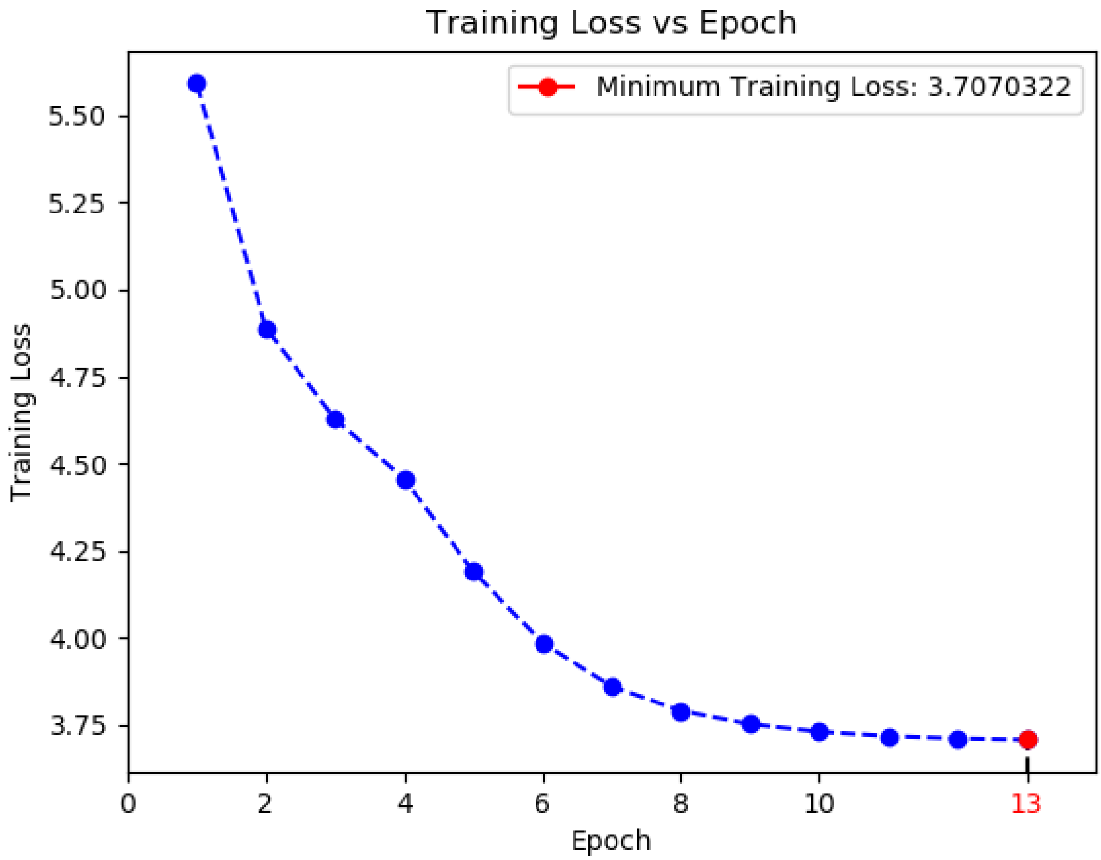

Figure 1: Training Loss vs Epoch

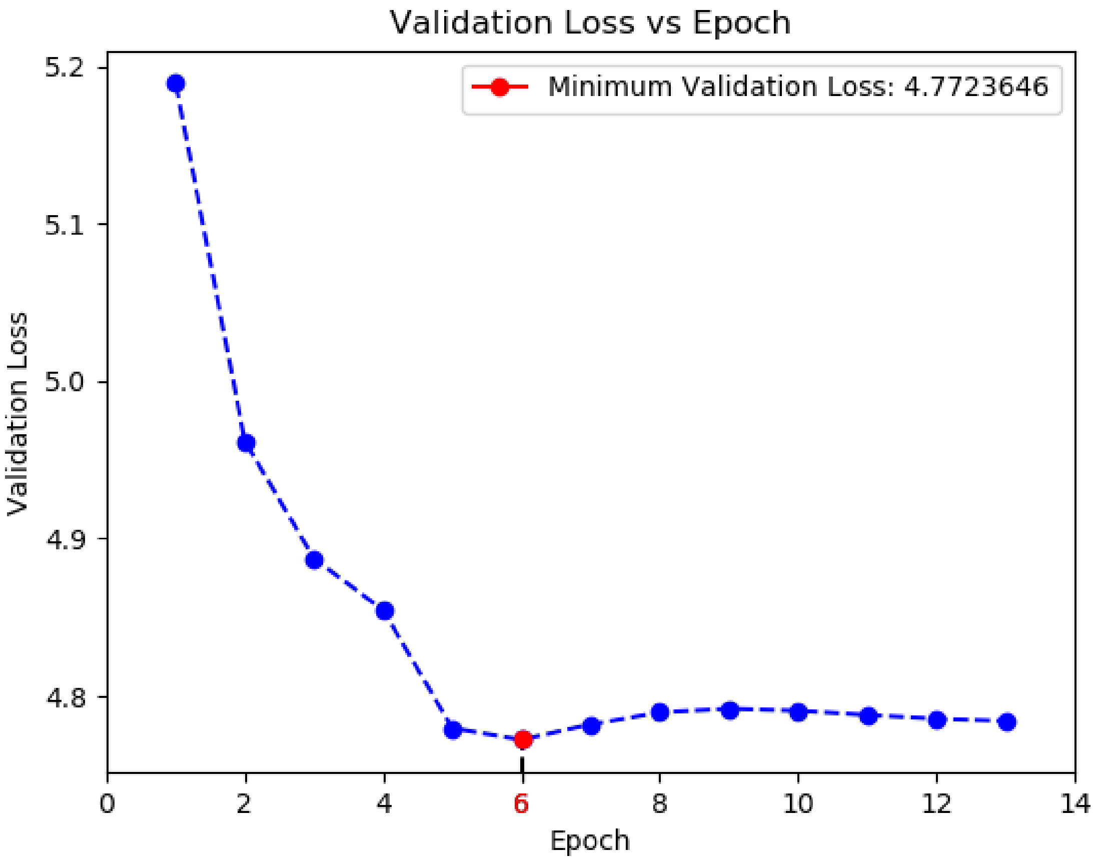

Figure 2: Validation Loss vs Epoch

The model is clearly overfitting, since the training perplexity and loss are far below that of the validation set. This could be solved by using a larger model, but was not used here due to time constraints.

c. Test Results

It can be noted from the output of the training code gives a loss of 4.745, which is right around the validation loss, showing that the validation set is a good representation of the test set. In terms of perplexity, the test set achieved 115, the validation set 119, and the training set 40.733.

d. Predicting Sentences with Four Different Words

The code for this section is below

```
# Copyright 2015 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
```

```
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================

"""Example / benchmark for building a PTB LSTM model.
Trains the model described in:
(Zaremba, et. al.) Recurrent Neural Network Regularization
http://arxiv.org/abs/1409.2329
There are 3 supported model configurations:
===========================================
| config | epochs | train | valid | test
===========================================
| small | 13    | 37.99 | 121.39 | 115.91
| medium | 39   | 48.45 | 86.16 | 82.07
| large | 55    | 37.87 | 82.62 | 78.29
The exact results may vary depending on the random initialization.
The hyperparameters used in the model:
- init_scale - the initial scale of the weights
- learning_rate - the initial value of the learning rate
- max_grad_norm - the maximum permissible norm of the gradient
- num_layers - the number of LSTM layers
- num_steps - the number of unrolled steps of LSTM
- hidden_size - the number of LSTM units
- max_epoch - the number of epochs trained with the initial learning rate
- max_max_epoch - the total number of epochs for training
- keep_prob - the probability of keeping weights in the dropout layer
- lr_decay - the decay of the learning rate for each epoch after "max_epoch"
- batch_size - the batch size
- rnn_mode - the low level implementation of lstm cell: one of CUDNN,
            BASIC, or BLOCK, representing cudnn_lstm, basic_lstm, and
            lstm_block_cell classes.
The data required for this example is in the data/ dir of the
PTB dataset from Tomas Mikolov's webpage:
$ wget http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz
$ tar xvf simple-examples.tgz
To run:
$ python ptb_word_lm.py --data_path=simple-examples/data/
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import time

import numpy as np
import tensorflow as tf

import reader
import util
```

```

from tensorflow.python.client import device_lib

flags = tf.flags
logging = tf.logging

flags.DEFINE_string(
    "model", "small",
    "A type of model. Possible options are: small, medium, large.")
flags.DEFINE_string("data_path", None,
                  "Where the training/test data is stored.")
flags.DEFINE_string("save_path", None,
                  "Model output directory.")
flags.DEFINE_bool("use_fp16", False,
                "Train using 16-bit floats instead of 32bit floats")
flags.DEFINE_integer("num_gpus", 1,
                   "If larger than 1, Grappler AutoParallel optimizer "
                   "will create multiple training replicas with each GPU "
                   "running one replica.")
flags.DEFINE_string("rnn_mode", None,
                  "The low level implementation of lstm cell: one of CUDNN, "
                  "BASIC, and BLOCK, representing cudnn_lstm, basic_lstm, "
                  "and lstm_block_cell classes.")
FLAGS = flags.FLAGS
BASIC = "basic"
CUDNN = "cudnn"
BLOCK = "block"


def data_type():
  return tf.float16 if FLAGS.use_fp16 else tf.float32


class PTBInput(object):
  """The input data."""

  def __init__(self, config, data, name=None):
    self.batch_size = batch_size = config.batch_size
    self.num_steps = num_steps = config.num_steps
    self.epoch_size = ((len(data) // batch_size) - 1) // num_steps
    self.input_data, self.targets = reader.ptb_producer(
        data, batch_size, num_steps, name=name)


class PTBModel(object):
  """The PTB model."""

  def __init__(self, is_training, config, input_):
    self._is_training = is_training
    # self._input = input_
    self._rnn_params = None
    self._cell = None
    self.batch_size = batch_size = config.batch_size
    self.num_steps = num_steps = config.num_steps
    size = config.hidden_size
    vocab_size = config.vocab_size

    self._input_data = tf.placeholder(tf.int32, [batch_size, num_steps])
    self._targets = tf.placeholder(tf.int32, [batch_size, num_steps])

    with tf.device("/cpu:0"):
      embedding = tf.get_variable(
          "embedding", [vocab_size, size], dtype=data_type())
      inputs = tf.nn.embedding_lookup(embedding, self._input_data)

    if is_training and config.keep_prob < 1:
      inputs = tf.nn.dropout(inputs, config.keep_prob)

    output, state = self._build_rnn_graph(inputs, config, is_training)

    softmax_w = tf.get_variable(
        "softmax_w", [size, vocab_size], dtype=data_type())
    softmax_b = tf.get_variable("softmax_b", [vocab_size], dtype=data_type())
    logits = tf.nn.xw_plus_b(output, softmax_w, softmax_b)
    self.sample = tf.multinomial(logits, 1)
     # Reshape logits to be a 3-D tensor for sequence loss
    logits = tf.reshape(logits, [self.batch_size, self.num_steps, vocab_size])

    # Use the contrib sequence loss and average over the batches
    loss = tf.contrib.seq2seq.sequence_loss(
        logits,
        self._targets,
        tf.ones([self.batch_size, self.num_steps], dtype=data_type()),
        average_across_timesteps=False,
        average_across_batch=True)

    # Update the cost
    self._cost = tf.reduce_sum(loss)
    self._final_state = state

    if not is_training:
      return

    self._lr = tf.Variable(0.0, trainable=False)
    tvars = tf.trainable_variables()
    grads, _ = tf.clip_by_global_norm(tf.gradients(self._cost, tvars),
                                   config.max_grad_norm)
    optimizer = tf.train.GradientDescentOptimizer(self._lr)
    self._train_op = optimizer.apply_gradients(
        zip(grads, tvars),
        global_step=tf.train.get_or_create_global_step())

    self._new_lr = tf.placeholder(
        tf.float32, shape=[], name="new_learning_rate")
    self._lr_update = tf.assign(self._lr, self._new_lr)

  def _build_rnn_graph(self, inputs, config, is_training):
    if config.rnn_mode == CUDNN:
      return self._build_rnn_graph_cudnn(inputs, config, is_training)
    else:
      return self._build_rnn_graph_lstm(inputs, config, is_training)

  def _build_rnn_graph_cudnn(self, inputs, config, is_training):
    """Build the inference graph using CUDNN cell."""
    inputs = tf.transpose(inputs, [1, 0, 2])
    self._cell = tf.contrib.cudnn_rnn.CudnnLSTM(
        num_layers=config.num_layers,
        num_units=config.hidden_size,
        input_size=config.hidden_size,
        dropout=1 - config.keep_prob if is_training else 0)
    params_size_t = self._cell.params_size()
    self._rnn_params = tf.get_variable(
        "lstm_params",
        initializer=tf.random_uniform(
            [params_size_t], -config.init_scale, config.init_scale),
        validate_shape=False)
    c = tf.zeros([config.num_layers, self.batch_size, config.hidden_size],
                 tf.float32)
    h = tf.zeros([config.num_layers, self.batch_size, config.hidden_size],
                 tf.float32)
    self._initial_state = (tf.contrib.rnn.LSTMStateTuple(h=h, c=c),)
    outputs, h, c = self._cell(inputs, h, c, self._rnn_params, is_training)
    outputs = tf.transpose(outputs, [1, 0, 2])
    outputs = tf.reshape(outputs, [-1, config.hidden_size])
    return outputs, (tf.contrib.rnn.LSTMStateTuple(h=h, c=c),)

  def _get_lstm_cell(self, config, is_training):
    if config.rnn_mode == BASIC:
      return tf.contrib.rnn.BasicLSTMCell(
          config.hidden_size, forget_bias=0.0, state_is_tuple=True,
          reuse=not is_training)
    if config.rnn_mode == BLOCK:
      return tf.contrib.rnn.LSTMBlockCell(
          config.hidden_size, forget_bias=0.0)
    raise ValueError("rnn_mode %s not supported" % config.rnn_mode)

  def _build_rnn_graph_lstm(self, inputs, config, is_training):
    """Build the inference graph using canonical LSTM cells."""
    # Slightly better results can be obtained with forget gate biases
    # initialized to 1 but the hyperparameters of the model would need to be
    # different than reported in the paper.
    def make_cell():
      cell = self._get_lstm_cell(config, is_training)
      if is_training and config.keep_prob < 1:
        cell = tf.contrib.rnn.DropoutWrapper(
            cell, output_keep_prob=config.keep_prob)
      return cell

    cell = tf.contrib.rnn.MultiRNNCell(
        [make_cell() for _ in range(config.num_layers)], state_is_tuple=True)

    self._initial_state = cell.zero_state(config.batch_size, data_type())
    state = self._initial_state
    # Simplified version of tf.nn.static_rnn().
    # This builds an unrolled LSTM for tutorial purposes only.
    # In general, use tf.nn.static_rnn() or tf.nn.static_state_saving_rnn().
    #
    # The alternative version of the code below is:
    #
    inputs = tf.unstack(inputs, num=self.num_steps, axis=1)
    outputs, state = tf.nn.static_rnn(cell, inputs,
                                    initial_state=self._initial_state)
    # outputs = []
    # with tf.variable_scope("RNN"):
    #   for time_step in range(self.num_steps):
    #     if time_step > 0: tf.get_variable_scope().reuse_variables()
    #     (cell_output, state) = cell(inputs[:, time_step, :], state)
    #     outputs.append(cell_output)
    output = tf.reshape(tf.concat(outputs, 1), [-1, config.hidden_size])
    return output, state

  def assign_lr(self, session, lr_value):
    session.run(self._lr_update, feed_dict={self._new_lr: lr_value})

  def export_ops(self, name):
    """Exports ops to collections."""
    self._name = name
    ops = {util.with_prefix(self._name, "cost"): self._cost}
    if self._is_training:
      ops.update(lr=self._lr, new_lr=self._new_lr, lr_update=self._lr_update)
      if self._rnn_params:
        ops.update(rnn_params=self._rnn_params)
    for name, op in ops.items():
      tf.add_to_collection(name, op)
    self._initial_state_name = util.with_prefix(self._name, "initial")
    self._final_state_name = util.with_prefix(self._name, "final")
    util.export_state_tuples(self._initial_state, self._initial_state_name)
    util.export_state_tuples(self._final_state, self._final_state_name)

  def import_ops(self):
    """Imports ops from collections."""
    if self._is_training:
      self._train_op = tf.get_collection_ref("train_op")[0]
      self._lr = tf.get_collection_ref("lr")[0]
      self._new_lr = tf.get_collection_ref("new_lr")[0]
      self._lr_update = tf.get_collection_ref("lr_update")[0]
      rnn_params = tf.get_collection_ref("rnn_params")
      if self._cell and rnn_params:
        params_saveable = tf.contrib.cudnn_rnn.RNNParamsSaveable(
            self._cell,
            self._cell.params_to_canonical,
            self._cell.canonical_to_params,
            rnn_params,
            base_variable_scope="Model/RNN")
        tf.add_to_collection(tf.GraphKeys.SAVEABLE_OBJECTS, params_saveable)
    self._cost = tf.get_collection_ref(util.with_prefix(self._name, "cost"))[0]
    num_replicas = FLAGS.num_gpus if self._name == "Train" else 1
    self._initial_state = util.import_state_tuples(
        self._initial_state, self._initial_state_name, num_replicas)
    self._final_state = util.import_state_tuples(
        self._final_state, self._final_state_name, num_replicas)


  @property
  def initial_state(self):
    return self._initial_state

  @property
  def cost(self):
    return self._cost

  @property
  def final_state(self):
    return self._final_state

  @property
  def lr(self):
    return self._lr

  @property
  def train_op(self):
    return self._train_op
  @property
  def input_data(self):
    return self._input_data

  @property
  def targets(self):
    return self._targets
  @property
  def initial_state_name(self):
    return self._initial_state_name

  @property
  def final_state_name(self):
    return self._final_state_name



class SmallConfig(object):
  """Small config."""
  init_scale = 0.1
  learning_rate = 1.0
  max_grad_norm = 5
  num_layers = 2
  num_steps = 20
  hidden_size = 200
  max_epoch = 4
  max_max_epoch = 13
  keep_prob = 1.0
  lr_decay = 0.5
  batch_size = 20
  vocab_size = 10000
  rnn_mode = BLOCK


class MediumConfig(object):
  """Medium config."""
  init_scale = 0.05
  learning_rate = 1.0
  max_grad_norm = 5
  num_layers = 2
  num_steps = 35
  hidden_size = 650
  max_epoch = 6
  max_max_epoch = 39
  keep_prob = 0.5
  lr_decay = 0.8
  batch_size = 20
  vocab_size = 10000
  rnn_mode = BLOCK


class LargeConfig(object):
  """Large config."""
  init_scale = 0.04
  learning_rate = 1.0
  max_grad_norm = 10
  num_layers = 2
  num_steps = 35
  hidden_size = 1500
  max_epoch = 14
  max_max_epoch = 55
  keep_prob = 0.35
  lr_decay = 1 / 1.15
  batch_size = 20
  vocab_size = 10000
  rnn_mode = BLOCK


class TestConfig(object):
  """Tiny config, for testing."""
  init_scale = 0.1
  learning_rate = 1.0
  max_grad_norm = 1
  num_layers = 1
  num_steps = 2
  hidden_size = 2
  max_epoch = 1
  max_max_epoch = 1
  keep_prob = 1.0
  lr_decay = 0.5
  batch_size = 20
  vocab_size = 10000
  rnn_mode = BLOCK


def run_epoch(session, model, eval_op=None, verbose=False):
  """Runs the model on the given data."""
  start_time = time.time()
  costs = 0.0
  iters = 0
  state = session.run(model.initial_state)

  fetches = {
      "cost": model.cost,
      "final_state": model.final_state,
  }
  if eval_op is not None:
    fetches["eval_op"] = eval_op

  for step in range(model.input.epoch_size):
    feed_dict = {}
    for i, (c, h) in enumerate(model.initial_state):
      feed_dict[c] = state[i].c
      feed_dict[h] = state[i].h

    vals = session.run(fetches, feed_dict)
    cost = vals["cost"]
    state = vals["final_state"]

    costs += cost
    iters += model.input.num_steps

    if verbose and step % (model.input.epoch_size // 10) == 10:
      print("%.3f perplexity: %.3f speed: %.0f wps" %
            (step * 1.0 / model.input.epoch_size, np.exp(costs / iters),
             iters * model.input.batch_size * max(1, FLAGS.num_gpus) /
             (time.time() - start_time)))

  return np.exp(costs / iters), (costs / iters), logits


def get_config():
  """Get model config."""
  config = None
  if FLAGS.model == "small":
    config = SmallConfig()
  elif FLAGS.model == "medium":
    config = MediumConfig()
  elif FLAGS.model == "large":
    config = LargeConfig()
  elif FLAGS.model == "test":
    config = TestConfig()
  else:
    raise ValueError("Invalid model: %s", FLAGS.model)
  if FLAGS.rnn_mode:
    config.rnn_mode = FLAGS.rnn_mode
  if FLAGS.num_gpus != 1 or tf.__version__ < "1.3.0" :
    config.rnn_mode = BASIC
  return config

def get_sentence(session, model, data, num_samples):
  # get initial state
  samples = []
  state = session.run(model.initial_state)
  # fetches are the final state and the prediction
  fetches = [model.final_state, model.sample]
  sample = None
  # get the next word, can be from multiple words in data
  for x in data:
    feed_dict = {}
    feed_dict[model.input_data] = [[x]]
    for layer_num, (c, h) in enumerate(model.initial_state):
      feed_dict[c] = state[layer_num].c
      feed_dict[h] = state[layer_num].h

    state, sample = session.run(fetches, feed_dict)

  # append next word
  samples.append(sample[0][0])


  # for num_samples, append words
  k = 1
  while k < num_samples:
    feed_dict = {}
    feed_dict[model.input_data] = [[samples[-1]]]
    for layer_num, (c, h) in enumerate(model.initial_state):
      feed_dict[c] = state[layer_num].c
      feed_dict[h] = state[layer_num].h
    state, sample = session.run(fetches, feed_dict)
    samples.append(sample[0][0])
    # if sample is <eos>, return samples early
    if (sample[0][0] == 2):
      return samples

    k += 1
  return samples

def print_sentence(items, id_to_word):
    sentence = ’’
    for item in items:
      sentence = sentence + ’ ’ + id_to_word[item]
    return sentence

def main(_):
  if not FLAGS.data_path:
    raise ValueError("Must set --data_path to PTB data directory")
  gpus = [
      x.name for x in device_lib.list_local_devices() if x.device_type == "GPU"
  ]
```

```
if FLAGS.num_gpus > len(gpus):
  raise ValueError(
      "Your machine has only %d gpus "
      "which is less than the requested --num_gpus=%d."
      % (len(gpus), FLAGS.num_gpus))

raw_data = reader.ptb_raw_data(FLAGS.data_path)
train_data, valid_data, test_data, _, word_to_id = raw_data
id_to_word = dict(zip(word_to_id.values(),word_to_id.keys()))

config = get_config()
eval_config = get_config()
eval_config.batch_size = 1
eval_config.num_steps = 1

with tf.Graph().as_default():
  initializer = tf.random_uniform_initializer(-config.init_scale,
                                              config.init_scale)

  with tf.name_scope("Train"):
    train_input = PTBInput(config=config, data=train_data, name="TrainInput")
    with tf.variable_scope("Model", reuse=None, initializer=initializer):
      m = PTBModel(is_training=True, config=config, input_=train_input)
    tf.summary.scalar("Training Loss", m.cost)
    tf.summary.scalar("Learning Rate", m.lr)


  with tf.name_scope("Test"):
    test_input = PTBInput(
        config=eval_config, data=test_data, name="TestInput")
    with tf.variable_scope("Model", reuse=True, initializer=initializer):
      mtest = PTBModel(is_training=False, config=eval_config,
                       input_=test_input)

  saver = tf.train.Saver()
  titan_v = "0"
  titan_x = "1"
  gpu_options = tf.GPUOptions(visible_device_list=titan_v)
  config = tf.ConfigProto(gpu_options=gpu_options)
  sv = tf.train.Supervisor(logdir=FLAGS.save_path)
  with sv.managed_session() as session:
    sv.saver.restore(session, './ptb_ltsm.ckpt')
    for i in range(1):
      user_input = input("Enter your starting words: ")
      max_num_words = int(input("Max words: "))
      starting_words = user_input.split()
      print("Starting Words: %s" % print_sentence([word_to_id[x] for x in
          starting_words], id_to_word))
      sentence = get_sentence(session, mtest, [word_to_id[word] for word in
          starting_words],max_num_words)
      print("Resulting Sentence: %s" % print_sentence(sentence, id_to_word))
```

```

if __name__ == "__main__":
  tf.app.run()
```

Note the main changes are using placeholders as input instead of hard coded lengths, and the get_sentence and print_sentence methods. The code goes for 20 words or the end of a sentence. The results for four chosen words are below

Word 1: the

```
Enter your starting words: the
Max words: 20
Starting Words: the
Resulting Sentence: <unk> of america and the <unk> few face other than the absolutely
    <unk> of the outcome of a new post
```

Word 2: words

```
Enter your starting words: words
Max words: 20
Starting Words: words
Resulting Sentence: are clearly considered more often than saying it would be worth
    <unk> <eos>
```

Word 3: software

```
Enter your starting words: software
Max words: 20
Starting Words: software
Resulting Sentence: factory policies sweat lower limited and pricing <eos>
```

Word 4: murder

```
Enter your starting words: murder
Max words: 20
Starting Words: murder
Resulting Sentence: a private championship to buy them the championship <unk> available
    in a major market <eos>
```

e. Predicting Sentences with given words

Word 1: north

```
Enter your starting words: north
Max words: 20
Starting Words: north
```

```
Resulting Sentence: american pacific corp. and <unk> in california contributed to this
    london options and mark <eos>
```

Word 2: wall

```
Enter your starting words: wall
Max words: 20
Starting Words: wall
Resulting Sentence: street 's biggest investment investment firms led the identity field
    to farmers in a televised <unk> <unk> <eos>
```

Word 3: truth

```
Enter your starting words: truth
Max words: 20
Starting Words: truth
Resulting Sentence: dr. freeman says when an <unk> imposed a significant N portion of
    the award <eos>
```

Word 4: california

```
Enter your starting words: california
Max words: 20
Starting Words: california
Resulting Sentence: may not give the profile of a massive dollar without their own foot
    <eos>
```

Word 5: health

```
Enter your starting words: health
Max words: 20
Starting Words: health
Resulting Sentence: insurance operation <eos>
```