Neural Bag-of-Words Model

SOLUTIONS - do not distribute!

In this notebook, we'll move beyond linear classifiers and implement a neural network for our classification task.

We'll also introduce the TensorFlow Estimator API (https://www.tensorflow.org/extend/estimators), which provides a high-level interface similar to scikit-learn. This involves a few new concepts, such as the idea of a `model_fn` and an `input_fn`, but it greatly simplifies experiments and reduces the need to write tedious data-feeding code.

Outline

- **Part (d):** Model architecture
- **Part (e):** Implementing the Neural BOW model
- **Introduction to `tf.Estimator`**
- **Part (f):** Training, evaluation, and tuning

As with the first half of the assignment, exercised are interspersed throughout the notebook. In particular, Part (d) has 4 questions, Part (e) asks you to write code in `models.py`, and Part (f) has 4 questions plus one optional implementation exercise.

Part (d) Short Answer Questions

Answer the following in the cell below.

1. Let `embed_dim = d`, `hidden_dims = [h1, h2]`, and `num_classes = k`. In terms of these values and the vocabulary size `V`, write down the shapes of the following variables: W_{embed}, $W^{(0)}$, $b^{(0)}$, $W^{(1)}$, $b^{(1)}$, W_{out}, b_{out}. (*Hint: W_{embed} has a row for each word in the vocabulary.*)

2. Using your answer to 1., how many parameters (matrix or vector elements) are in the embedding layer? How about in the hidden layers? And the output layer?

3. Recall that logistic regression can be thought of as a single-layer neural network. What should we set as the values of `embed_dim` and `hidden_dims` such that this model implements logistic regression?

4. Suppose that we have two examples, `[foo bar baz]` and `[baz bar foo]`. Will this model make the same predictions on these? Why or why not?

Part (d) Answers

SOLUTIONS - do not distribute!

1. Using the notation [object] == dim(object),

$$[W_{embed}] = \text{V x d}$$
$$[W^{(0)}] = \text{d x h1}$$
$$[b_0] = \texttt{h1}$$
$$[W_1] = \text{h1 x h2}$$
$$[b_1] = \texttt{h2}$$
$$[W_{out}] = \text{h2 x k}$$
$$[b_{out}] = \texttt{k}$$

2. Embedding layer has: **V * d parameters**
   Hidden layer 0 has: **(d+1) * h1 parameters**
   Hidden layer 1 has: **(h+1) * h2 parameters**
   Output layer has: **(h2 + 1) * k parameters**
   Of course, Big-O notation also works here:
   (ie. Hidden layer 0 is O(d h1)

3. A logistic regression model for the BOW case models each word count as a feature and produces a probability for each class.

   Therefore embed_dim = k (so our embedding matrix is |V| x k).

   Because our embedding table gives us logits directly, hidden_dims = [] and our output is just a softmax function to convert to the max probability.

4. Our model will make the same prediction for [foo bar baz] as for [baz bar foo] since it only considers word counts and doesn't pay attention to word order.

In [4]:
```
print("Examples:\n", train_x[:3])
print("Original sequence lengths: ", train_ns[:3])
print("Target labels: ", train_y[:3])
print("")
print("Padded:\n", " ".join(ds.vocab.ids_to_words(train_x[0])))
print("Un-padded:\n", " ".join(ds.vocab.ids_to_words(train_x[0,:train_ns[0
]])))
```

```
Examples:
 [[   4  606   10 3416    9   26    4 2821 1263   11  108   63 5543   64
     7   13   75   11  277    9   84    6 4243   69 3417   40 1869 2822
     5 8181 1682 5544   48  846 8182    3    0    0    0    0]
 [   4 2823 1870 5545    8   63    4 3418    8    4 2441   64 5546   10
    46  905   13    6 5547    8  680   67   29 3419 2113 5548 1030  847
    11 5549  623    8 8183 5550   11 8184    3    0    0    0]
 [8185 5551 2114 8186    6 8187    8 1530   36    6  167  769 1264    5
     6  167   34  296 8188    9    4   51   36   16    4  307 3420  345
   624    4 1031    5 4244    5  447    8    4  273    3    0]]
Original sequence lengths:  [36 37 39]
Target labels:  [1 1 1]

Padded:
 the rock is destined to be the 21st century 's new `` conan '' and that he
's going to make a splash even greater than arnold schwarzenegger , jean-clau
d van damme or steven segal . <s> <s> <s> <s>
Un-padded:
 the rock is destined to be the 21st century 's new `` conan '' and that he
's going to make a splash even greater than arnold schwarzenegger , jean-clau
d van damme or steven segal .
```

Part (e): Implementing the Neural BOW Model

In order to better manage the model code, we'll implement our BOW model in `models.py`. In particular, you'll need to implement the following functions:

- `embedding_layer(...)`: constructs an embedding layer
- `BOW_encoder(...)`: constructs the encoder stack as described above
- `softmax_output_layer(...)`: constructs a softmax output layer

Follow the instructions in the code (function docstrings and comments) carefully!

In particular, for unit tests to work, you shouldn't change (or add) any `tf.name_scope` or `tf.variable_scope` calls, and must name the variables exactly as documented. (Your model may work just fine, of course, but the test harness will throw all sorts of errors!)

To aid debugging and readability, we've adopted a convention that TensorFlow tensors are represented by variables ending in an underscore, such as `W_embed_` or `train_op_`.

Before you start, be sure to answer the short-answer questions in Part (d). (*We guarantee that this section will be **much** harder if you don't!*)

You may find the following TensorFlow API functions useful:

- `tf.nn.embedding_lookup` (https://www.tensorflow.org/versions/master/api_docs/python/tf/nn/embedding_lookup)
- `tf.nn.sparse_softmax_cross_entropy_with_logits` (https://www.tensorflow.org/versions/master/api_docs/python/tf/nn/sparse_softmax_cross_entropy_with_logits
- `tf.reduce_mean` (https://www.tensorflow.org/versions/master/api_docs/python/tf/reduce_mean) and `tf.reduce_sum` (https://www.tensorflow.org/versions/master/api_docs/python/tf/reduce_sum)

Do your work in `models.py`. When ready, run the cell below to run the unit tests.

In [5]:
```
reload(models)
utils.run_tests(models_test, ["TestLayerBuilders", "TestNeuralBOW"])
```

```
test_embedding_layer (models_test.TestLayerBuilders) ... ok
test_softmax_output_layer (models_test.TestLayerBuilders) ... ok
test_BOW_encoder (models_test.TestNeuralBOW) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.079s

OK
```

Training a Neural Network (the hard way)

In Assignment 1, we trained our simple model with a home-spun training loop, setting up `feed_dict`-s and making calls to `session.run()`. For demonstration, let's do the same here.

We've implemented a wrapper function, `models.classifier_model_fn`, which uses the functions you wrote in **Part (e)** to build a model graph. It takes as input `features` and `labels` which contain input and target tensors, as well as `model` and `params` which configure the model.

Exercise (not graded): Read through the code for `classifier_model_fn()` in `models.py`. Where is the code you wrote in Part (e) called? Where is the loss function set up, and what loss is used? How is the optimizer set up, and what options are available? What types of predictions are returned in the `predictions` dict?

Using this function directly, we can write a simple training loop similar to Assignment 1's `train_nn()`:

In [6]:
```
import models; reload(models)

x, ns, y = train_x, train_ns, train_y
batch_size = 32

# Specify model hyperparameters as used by model_fn
model_params = dict(V=ds.vocab.size, embed_dim=50, hidden_dims=[25], num_class
es=len(ds.target_names),
                    encoder_type='bow',
                    lr=0.1, optimizer='adagrad', beta=0.01)
model_fn = models.classifier_model_fn

total_batches = 0
total_examples = 0
total_loss = 0
loss_ema = np.log(2)  # track exponential-moving-average of loss
ema_decay = np.exp(-1/10)  # decay parameter for moving average = np.exp(-1/hi
story_length)
with tf.Graph().as_default(), tf.Session() as sess:
    ##
    # Construct the graph here. No session.run calls - just wiring up Tensors.
    ##
    # Add placeholders so we can feed in data.
    x_ph_  = tf.placeholder(tf.int32, shape=[None, x.shape[1]])  # [batch_siz
e, max_len]
    ns_ph_ = tf.placeholder(tf.int32, shape=[None])              # [batch_siz
e]
    y_ph_  = tf.placeholder(tf.int32, shape=[None])              # [batch_siz
e]

    # Construct the graph using model_fn
    features = {"ids": x_ph_, "ns": ns_ph_}  # note that values are Tensors
    estimator_spec = model_fn(features, labels=y_ph_, mode=tf.estimator.ModeKe
ys.TRAIN,
                              params=model_params)
    loss_     = estimator_spec.loss
    train_op_ = estimator_spec.train_op

    ##
    # Done constructing the graph, now we can make session.run calls.
    ##
    sess.run(tf.global_variables_initializer())

    # Run a single epoch
    t0 = time.time()
    for (bx, bns, by) in utils.multi_batch_generator(batch_size, x, ns, y):
        # feed NumPy arrays into the placeholder Tensors
        feed_dict = {x_ph_: bx, ns_ph_: bns, y_ph_: by}
        batch_loss, _ = sess.run([loss_, train_op_], feed_dict=feed_dict)

        # Compute some statistics
        total_batches += 1
        total_examples += len(bx)
        total_loss += batch_loss * len(bx)  # re-scale, since batch loss is me
an
        # Compute moving average to smooth out noisy per-batch loss
```

```
        loss_ema = ema_decay * loss_ema + (1 - ema_decay) * batch_loss

        if (total_batches % 25 == 0):
            print("{:5,} examples, moving-average loss {:.2f}".format(total_ex
amples,
                                                                      loss_ema
))
    print("Completed one epoch in {:s}".format(utils.pretty_timedelta(since=t0
)))
```

```
  800 examples, moving-average loss 0.67
1,600 examples, moving-average loss 0.53
2,400 examples, moving-average loss 0.46
3,200 examples, moving-average loss 0.50
4,000 examples, moving-average loss 0.61
4,800 examples, moving-average loss 0.47
5,600 examples, moving-average loss 0.46
6,400 examples, moving-average loss 0.43
Completed one epoch in 0:00:01
```

Part (f): Training and Evaluation

The cell below defines some model params and sets up a checkpoint directory for TensorBoard.

Use the following default parameters to start, as given below in `model_params`:

```
embed_dim = 50
hidden_dims = [25]  # single hidden layer
optimizer = 'adagrad'
lr = 0.1  # learning rate
beta = 0.01  # L2 regularization
```

Note: Due to a bug in TensorFlow, if you re-use the same checkpoint directory (even after deleting the contents) it will sometimes fail to write the event data for TensorBoard. To work around this, the code below creates a new checkpoint directory each time with a name derived from the timestamp. You may want to delete these after a few runs, since they can take up ~35MB each. To do so just run:

```
# On command line
rm -rfv /tmp/tf_bow_sst_*
```

In [7]:
```
import models; reload(models)

# Specify model hyperparameters as used by model_fn
model_params = dict(V=ds.vocab.size, embed_dim=50, hidden_dims=[25], num_class
es=len(ds.target_names),
                    encoder_type='bow',
                    lr=0.1, optimizer='adagrad', beta=0.01)

checkpoint_dir = "/tmp/tf_bow_sst_" + datetime.datetime.now().strftime("%Y%m%d
-%H%M")
if os.path.isdir(checkpoint_dir):
    shutil.rmtree(checkpoint_dir)
# Write vocabulary to file, so TensorBoard can label embeddings.
# creates checkpoint_dir/projector_config.pbtxt and checkpoint_dir/metadata.ts
v
ds.vocab.write_projector_config(checkpoint_dir, "Encoder/Embedding_Layer/W_emb
ed")

model = tf.estimator.Estimator(model_fn=models.classifier_model_fn,
                               params=model_params,
                               model_dir=checkpoint_dir)
print("")
print("To view training (once it starts), run:\n")
print("    tensorboard --logdir='{:s}' --port 6006".format(checkpoint_dir))
print("\nThen in your browser, open: http://localhost:6006")
```

```
Vocabulary (16,474 words) written to '/tmp/tf_bow_sst_20180303-2044/metadata.
tsv'
Projector config written to /tmp/tf_bow_sst_20180303-2044/projector_config.pb
txt
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': '/tmp/tf_bow_sst_20180303-2044',
'_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_step
s': None, '_save_checkpoints_secs': 600, '_session_config': None, '_keep_chec
kpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_step_count_ste
ps': 100, '_service': None, '_cluster_spec': <tensorflow.python.training.serv
er_lib.ClusterSpec object at 0x7f97ff9a14a8>, '_task_type': 'worker', '_task_
id': 0, '_master': '', '_is_chief': True, '_num_ps_replicas': 0, '_num_worker
_replicas': 1}

To view training (once it starts), run:

    tensorboard --logdir='/tmp/tf_bow_sst_20180303-2044' --port 6006

Then in your browser, open: http://localhost:6006
```

In [8]:
```
# Training params, just used in this cell for the input_fn-s
train_params = dict(batch_size=32, total_epochs=20, eval_every=2)
assert(train_params['total_epochs'] % train_params['eval_every'] == 0)

# Construct and train the model, saving checkpoints to the directory above.
# Input function for training set batches
# Do 'eval_every' epochs at once, followed by evaluating on the dev set.
# NOTE: use patch_numpy_io.numpy_input_fn instead of tf.estimator.inputs.numpy
_input_fn
train_input_fn = patched_numpy_io.numpy_input_fn(
                    x={"ids": train_x, "ns": train_ns}, y=train_y,
                    batch_size=train_params['batch_size'],
                    num_epochs=train_params['eval_every'], shuffle=True, seed=
42
                )

# Input function for dev set batches. As above, but:
# - Don't randomize order
# - Iterate exactly once (one epoch)
dev_input_fn = tf.estimator.inputs.numpy_input_fn(
                    x={"ids": dev_x, "ns": dev_ns}, y=dev_y,
                    batch_size=128, num_epochs=1, shuffle=False
                )

for _ in range(train_params['total_epochs'] // train_params['eval_every']):
    # Train for a few epochs, then evaluate on dev
    model.train(input_fn=train_input_fn)
    eval_metrics = model.evaluate(input_fn=dev_input_fn, name="dev")
```

```
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Saving checkpoints for 1 into /tmp/tf_bow_sst_20180303-2044/m
odel.ckpt.
INFO:tensorflow:loss = 1.10304, step = 1
INFO:tensorflow:global_step/sec: 108.269
INFO:tensorflow:loss = 0.591804, step = 101 (0.927 sec)
INFO:tensorflow:global_step/sec: 113.507
INFO:tensorflow:loss = 0.56995, step = 201 (0.881 sec)
INFO:tensorflow:global_step/sec: 114.001
INFO:tensorflow:loss = 0.467347, step = 301 (0.877 sec)
INFO:tensorflow:global_step/sec: 112.884
INFO:tensorflow:loss = 0.412307, step = 401 (0.886 sec)
INFO:tensorflow:Saving checkpoints for 433 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:Loss for final step: 0.37697.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:44:39
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-433
INFO:tensorflow:Finished evaluation at 2018-03-03-20:44:40
INFO:tensorflow:Saving dict for global step 433: accuracy = 0.716743, cross_e
ntropy_loss = 0.589964, global_step = 433, loss = 0.72532
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-433
INFO:tensorflow:Saving checkpoints for 434 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:loss = 0.187939, step = 434
INFO:tensorflow:global_step/sec: 99.9729
INFO:tensorflow:loss = 0.241181, step = 534 (1.003 sec)
INFO:tensorflow:global_step/sec: 112.471
INFO:tensorflow:loss = 0.33393, step = 634 (0.889 sec)
INFO:tensorflow:global_step/sec: 115.472
INFO:tensorflow:loss = 0.325257, step = 734 (0.866 sec)
INFO:tensorflow:global_step/sec: 116.067
INFO:tensorflow:loss = 0.225925, step = 834 (0.862 sec)
INFO:tensorflow:Saving checkpoints for 866 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:Loss for final step: 0.184974.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:44:45
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-866
INFO:tensorflow:Finished evaluation at 2018-03-03-20:44:45
INFO:tensorflow:Saving dict for global step 866: accuracy = 0.75344, cross_en
tropy_loss = 0.577482, global_step = 866, loss = 0.712166
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-866
INFO:tensorflow:Saving checkpoints for 867 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:loss = 0.158775, step = 867
INFO:tensorflow:global_step/sec: 112.324
INFO:tensorflow:loss = 0.179848, step = 967 (0.893 sec)
INFO:tensorflow:global_step/sec: 116.59
INFO:tensorflow:loss = 0.246038, step = 1067 (0.858 sec)
INFO:tensorflow:global_step/sec: 116.155
INFO:tensorflow:loss = 0.177162, step = 1167 (0.861 sec)
INFO:tensorflow:global_step/sec: 116.003
```

```
INFO:tensorflow:loss = 0.132086, step = 1267 (0.862 sec)
INFO:tensorflow:Saving checkpoints for 1299 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:Loss for final step: 0.1097.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:44:50
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-1299
INFO:tensorflow:Finished evaluation at 2018-03-03-20:44:50
INFO:tensorflow:Saving dict for global step 1299: accuracy = 0.751147, cross_
entropy_loss = 0.66972, global_step = 1299, loss = 0.814589
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-1299
INFO:tensorflow:Saving checkpoints for 1300 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:loss = 0.122394, step = 1300
INFO:tensorflow:global_step/sec: 110.461
INFO:tensorflow:loss = 0.14109, step = 1400 (0.909 sec)
INFO:tensorflow:global_step/sec: 113.775
INFO:tensorflow:loss = 0.161843, step = 1500 (0.878 sec)
INFO:tensorflow:global_step/sec: 112.727
INFO:tensorflow:loss = 0.13644, step = 1600 (0.887 sec)
INFO:tensorflow:global_step/sec: 116.441
INFO:tensorflow:loss = 0.130521, step = 1700 (0.859 sec)
INFO:tensorflow:Saving checkpoints for 1732 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:Loss for final step: 0.103687.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:44:55
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-1732
INFO:tensorflow:Finished evaluation at 2018-03-03-20:44:55
INFO:tensorflow:Saving dict for global step 1732: accuracy = 0.752294, cross_
entropy_loss = 0.687566, global_step = 1732, loss = 0.834786
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-1732
INFO:tensorflow:Saving checkpoints for 1733 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:loss = 0.115005, step = 1733
INFO:tensorflow:global_step/sec: 111.349
INFO:tensorflow:loss = 0.124808, step = 1833 (0.901 sec)
INFO:tensorflow:global_step/sec: 114.444
INFO:tensorflow:loss = 0.145478, step = 1933 (0.874 sec)
INFO:tensorflow:global_step/sec: 116.242
INFO:tensorflow:loss = 0.130216, step = 2033 (0.860 sec)
INFO:tensorflow:global_step/sec: 116.829
INFO:tensorflow:loss = 0.125862, step = 2133 (0.856 sec)
INFO:tensorflow:Saving checkpoints for 2165 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:Loss for final step: 0.097705.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:45:00
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-2165
INFO:tensorflow:Finished evaluation at 2018-03-03-20:45:01
INFO:tensorflow:Saving dict for global step 2165: accuracy = 0.75, cross_entr
opy_loss = 0.691185, global_step = 2165, loss = 0.838418
INFO:tensorflow:Create CheckpointSaverHook.
```

```
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-2165
INFO:tensorflow:Saving checkpoints for 2166 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:loss = 0.11371, step = 2166
INFO:tensorflow:global_step/sec: 105.66
INFO:tensorflow:loss = 0.119296, step = 2266 (0.950 sec)
INFO:tensorflow:global_step/sec: 113.324
INFO:tensorflow:loss = 0.134251, step = 2366 (0.882 sec)
INFO:tensorflow:global_step/sec: 114.139
INFO:tensorflow:loss = 0.127195, step = 2466 (0.876 sec)
INFO:tensorflow:global_step/sec: 111.874
INFO:tensorflow:loss = 0.123041, step = 2566 (0.894 sec)
INFO:tensorflow:Saving checkpoints for 2598 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:Loss for final step: 0.0955552.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:45:06
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-2598
INFO:tensorflow:Finished evaluation at 2018-03-03-20:45:06
INFO:tensorflow:Saving dict for global step 2598: accuracy = 0.752294, cross_
entropy_loss = 0.686607, global_step = 2598, loss = 0.833749
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-2598
INFO:tensorflow:Saving checkpoints for 2599 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:loss = 0.11146, step = 2599
INFO:tensorflow:global_step/sec: 102.117
INFO:tensorflow:loss = 0.116842, step = 2699 (0.982 sec)
INFO:tensorflow:global_step/sec: 114.17
INFO:tensorflow:loss = 0.128582, step = 2799 (0.876 sec)
INFO:tensorflow:global_step/sec: 111.788
INFO:tensorflow:loss = 0.124567, step = 2899 (0.895 sec)
INFO:tensorflow:global_step/sec: 114.34
INFO:tensorflow:loss = 0.120874, step = 2999 (0.875 sec)
INFO:tensorflow:Saving checkpoints for 3031 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:Loss for final step: 0.0939148.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:45:11
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-3031
INFO:tensorflow:Finished evaluation at 2018-03-03-20:45:11
INFO:tensorflow:Saving dict for global step 3031: accuracy = 0.75344, cross_e
ntropy_loss = 0.681852, global_step = 3031, loss = 0.828951
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-3031
INFO:tensorflow:Saving checkpoints for 3032 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:loss = 0.109869, step = 3032
INFO:tensorflow:global_step/sec: 107.619
INFO:tensorflow:loss = 0.115151, step = 3132 (0.932 sec)
INFO:tensorflow:global_step/sec: 112.362
INFO:tensorflow:loss = 0.124997, step = 3232 (0.899 sec)
INFO:tensorflow:global_step/sec: 108.111
INFO:tensorflow:loss = 0.122526, step = 3332 (0.916 sec)
```

```
INFO:tensorflow:global_step/sec: 112.053
INFO:tensorflow:loss = 0.11915, step = 3432 (0.892 sec)
INFO:tensorflow:Saving checkpoints for 3464 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:Loss for final step: 0.0926181.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:45:16
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-3464
INFO:tensorflow:Finished evaluation at 2018-03-03-20:45:16
INFO:tensorflow:Saving dict for global step 3464: accuracy = 0.752294, cross_
entropy_loss = 0.677709, global_step = 3464, loss = 0.824788
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-3464
INFO:tensorflow:Saving checkpoints for 3465 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:loss = 0.108596, step = 3465
INFO:tensorflow:global_step/sec: 110.428
INFO:tensorflow:loss = 0.113856, step = 3565 (0.909 sec)
INFO:tensorflow:global_step/sec: 108.142
INFO:tensorflow:loss = 0.122356, step = 3665 (0.925 sec)
INFO:tensorflow:global_step/sec: 114.324
INFO:tensorflow:loss = 0.120879, step = 3765 (0.875 sec)
INFO:tensorflow:global_step/sec: 93.0351
INFO:tensorflow:loss = 0.117751, step = 3865 (1.076 sec)
INFO:tensorflow:Saving checkpoints for 3897 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:Loss for final step: 0.0915492.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:45:22
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-3897
INFO:tensorflow:Finished evaluation at 2018-03-03-20:45:22
INFO:tensorflow:Saving dict for global step 3897: accuracy = 0.748853, cross_
entropy_loss = 0.674125, global_step = 3897, loss = 0.821197
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-3897
INFO:tensorflow:Saving checkpoints for 3898 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:loss = 0.107535, step = 3898
INFO:tensorflow:global_step/sec: 109.969
INFO:tensorflow:loss = 0.112832, step = 3998 (0.913 sec)
INFO:tensorflow:global_step/sec: 114.594
INFO:tensorflow:loss = 0.120291, step = 4098 (0.873 sec)
INFO:tensorflow:global_step/sec: 112.599
INFO:tensorflow:loss = 0.119514, step = 4198 (0.888 sec)
INFO:tensorflow:global_step/sec: 114.38
INFO:tensorflow:loss = 0.116594, step = 4298 (0.874 sec)
INFO:tensorflow:Saving checkpoints for 4330 into /tmp/tf_bow_sst_20180303-204
4/model.ckpt.
INFO:tensorflow:Loss for final step: 0.0906442.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:45:28
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-4330
INFO:tensorflow:Finished evaluation at 2018-03-03-20:45:28
INFO:tensorflow:Saving dict for global step 4330: accuracy = 0.752294, cross_
entropy_loss = 0.67096, global_step = 4330, loss = 0.81803
```

Part(f).1: Evaluating Your Model

No solutions for coding part (a)

We won't be releasing code solutions, as this makes it difficult to re-use the assignment in future iterations of the class. If you have specific questions on your code, please come to office hours or post a private question on Piazza. To evaluate on the test set, we just need to construct another `input_fn`, then call `model.evaluate`.

1.) Fill in the cell below, and run it to compute accuracy on the test set. With the default parameters, you should get accuracy around 77%.

In [9]:
```
#### YOUR CODE HERE ####
# Code for Part (f).1

#### END(YOUR CODE) ####
print("Accuracy on test set: {:.02%}".format(eval_metrics['accuracy']))
eval_metrics
```

```
INFO:tensorflow:Starting evaluation at 2018-03-03-20:47:12
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-4330
INFO:tensorflow:Finished evaluation at 2018-03-03-20:47:12
INFO:tensorflow:Saving dict for global step 4330: accuracy = 0.7743, cross_en
tropy_loss = 0.61913, global_step = 4330, loss = 0.763519
Accuracy on test set: 77.43%
```

Out[9]:
```
{'accuracy': 0.77429986,
 'cross_entropy_loss': 0.61912978,
 'global_step': 4330,
 'loss': 0.76351869}
```

We can also evaluate the old-fashioned way, by calling `model.predict(...)` and working with the predicted labels directly:

In [10]:
```
from sklearn.metrics import accuracy_score
predictions = list(model.predict(test_input_fn))  # list of dicts
y_pred = [p['max'] for p in predictions]
acc = accuracy_score(y_pred, test_y)
print("Accuracy on test set: {:.02%}".format(acc))
```

```
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-4330
Accuracy on test set: 77.43%
```

Part (f).2: Evaluating on "Interesting" examples

Write your answer in the cell below.

Question 2.) In the cell below, repeat what you did above, but evaluate the model on the "interesting" examples. Does the neural bag-of-words model perform well here, as compared to the test set as a whole? How about compared to the Naive Bayes baseline? Explain why this might be, in terms of the phenomena you found in Part (b).4.

Part (f).2 Answers

SOLUTIONS - do not distribute!

No solutions for coding part (f.2)

We won't be releasing code solutions, as this makes it difficult to re-use the assignment in future iterations of the class. If you have specific questions on your code, please come to office hours or post a private question on Piazza.

2.) Unsurprisingly, the NeuralBOW model performs worse on the "interesting" examples (71.5%) than on the original full list (77.4%). It does slightly worse than the Naive Bayes model we ran for the Exploration phase (73.3%). Similar to part b.4, the "interesting" examples contain negation clauses where positive-words or negative-words get negated by a single word. This slight increase over Naive Bayes could be a result of the fact that our NeuralBOW model considers negation in the presence of other words. However since it doesn't consider word order the accuracy is not that much better.

In [11]:
```
df = ds.test

gb = df.groupby(by=['root_id'])
interesting_ids = []   # root ids, index into ds.test_trees
interesting_idxs = []  # DataFrame indices, index into ds.test
# This groups the DataFrame by sentence
for root_id, idxs in gb.groups.items():
    # Get the average score of all the phrases for this sentence
    mean = df.loc[idxs].label.mean()
    if (mean > 0.4 and mean < 0.6):
        interesting_ids.append(root_id)
        interesting_idxs.extend(idxs)

print("Found {:,} interesting examples".format(len(interesting_ids)))
print("Interesting ids (into ds.test_trees): ", interesting_ids)
print("")

# This will extract only the "interesting" sentences we found above
test_x_interesting, test_ns_interesting, test_y_interesting = ds.as_padded_arr
ay("test", root_only=True,

   df_idxs=interesting_idxs)
#### YOUR CODE HERE ####
# Code for Part (f).2


acc = 0  # replace with actual value


#### END(YOUR CODE) ####
print("Accuracy on test set: {:.02%}".format(acc))
```

```
Found 246 interesting examples
Interesting ids (into ds.test_trees):  [0, 27, 31, 32, 75, 80, 90, 96, 117, 1
24, 138, 140, 141, 160, 166, 186, 187, 205, 210, 212, 227, 232, 254, 269, 27
1, 285, 296, 307, 312, 327, 335, 373, 397, 399, 406, 407, 410, 426, 447, 511,
512, 516, 521, 534, 539, 563, 577, 588, 606, 610, 611, 637, 640, 645, 655, 66
2, 664, 713, 720, 721, 724, 739, 755, 758, 763, 776, 791, 793, 796, 802, 805,
810, 818, 840, 858, 887, 898, 899, 909, 910, 912, 929, 930, 961, 970, 973, 97
4, 975, 979, 1008, 1032, 1036, 1066, 1067, 1076, 1098, 1101, 1108, 1114, 113
1, 1138, 1142, 1159, 1183, 1185, 1189, 1193, 1198, 1206, 1214, 1215, 1235, 12
41, 1243, 1244, 1261, 1267, 1273, 1275, 1279, 1280, 1293, 1296, 1302, 1303, 1
312, 1318, 1319, 1321, 1322, 1324, 1326, 1328, 1338, 1341, 1346, 1359, 1363,
1371, 1383, 1398, 1402, 1413, 1443, 1452, 1456, 1458, 1462, 1464, 1480, 1481,
1486, 1487, 1488, 1507, 1509, 1513, 1516, 1527, 1537, 1552, 1576, 1582, 1587,
1594, 1597, 1602, 1607, 1608, 1615, 1619, 1622, 1629, 1630, 1639, 1666, 1682,
1688, 1694, 1727, 1728, 1731, 1755, 1763, 1764, 1786, 1789, 1793, 1795, 1798,
1804, 1807, 1817, 1830, 1834, 1850, 1852, 1883, 1885, 1899, 1903, 1908, 1910,
1918, 1929, 1933, 1939, 1940, 1942, 1943, 1945, 1947, 1949, 1950, 1952, 1954,
1964, 1965, 1970, 1972, 1980, 1982, 2008, 2009, 2011, 2021, 2035, 2036, 2038,
2063, 2079, 2085, 2089, 2094, 2096, 2118, 2119, 2121, 2143, 2145, 2149, 2150,
2160, 2164, 2190, 2193]

INFO:tensorflow:Starting evaluation at 2018-03-03-20:47:22
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2044/mode
l.ckpt-4330
INFO:tensorflow:Finished evaluation at 2018-03-03-20:47:22
INFO:tensorflow:Saving dict for global step 4330: accuracy = 0.715116, cross_
entropy_loss = 0.812027, global_step = 4330, loss = 0.939525
Accuracy on test set: 71.51%
```

Part (f): Tuning Your Model

Our default model from Part (e) performs decently, but doesn't manage to beat even the Naive Bayes baseline. We might be able to fix that with a bit of tuning.

Answer the following in the cell below.

Question 3.) Look at your training curves in TensorBoard, after 20 epochs with the default parameters. Do you think that the model would benefit from more training time?

Question 4.) Based on the accuracy trace (on the dev set) and the cross entropy loss curves on the training and dev sets, do you think the model is overfitting?

Answers for Part (f).3 and 4

SOLUTIONS - do not distribute!

3.) Despite the fact that cross-entropy and regulariation have started to plateau out they are still slightly increasing and therefore might benefit a bit by additional training.

4.) Despite the fact that cross-entropy loss for the dev set (blue line) is much higher than that for the training set (orange line) the cross-entropy loss for the dev set never actually increases, so the model doesn't appear to be over-training.

Regularization & Tuning

The baseline model uses L2 regularization to combat overfitting, but this isn't particularly effective with neural networks since a deep network can still learn spurious logical relationships even with small values for the connection weights. Instead, it's common to use *dropout*, in which we randomly "drop out" a subset of the activations by setting them to zero. This prevents units from co-adapting too easily, and often leads to improved generalization

(optional) 5.) In `models.py`, implement dropout by filling in the missing block in the implementation of `fully_connected_layers(...)`. You'll also need to modify your implementation of `BOW_encoder(...)` to pass the `dropout_rate` and `is_training` parameters to `fully_connected_layers(...)`.

Do not apply dropout to the softmax layer, or to the embeddings.

Hint: use `tf.layers.dropout` (https://www.tensorflow.org/api_docs/python/tf/layers/dropout).

We've replicated the training code in the cell below - modify `model_params` and `train_params`, and see if you can improve performance with a bit of tuning (*but don't spend too much time on this!*). Some things that might be worth trying:

- Enable dropout, and experiment with `dropout_rate`
- Train for more epochs (40 or 60). (*But, what happens if you train for too long?*)
- Use more hidden layers
- Use larger embedding and hidden dimensions
- Re-generate the training set with `root_only=False`, which will give set with fine-grained labels

Note: As it turns out, Naive Bayes is actually a pretty strong model for this dataset and it won't be easy to get a neural model to beat it *(see Table 1 from Socher et al. 2013 (http://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf) - our model is closest in design to the* VecAvg *model)*. Don't worry if tuning doesn't seem to help much for this particular problem.

In [12]:
```
# Run this if you implement dropout
reload(models)
utils.run_tests(models_test, ["TestFCWithDropout"])
```

```
test_fc_with_dropout (models_test.TestFCWithDropout) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.074s

OK
```

No solutions for coding part (f.5)

We won't be releasing code solutions, as this makes it difficult to re-use the assignment in future iterations of the class. If you have specific questions on your code, please come to office hours or post a private question on Piazza.

In [13]:
```
import models; reload(models)

# Specify model hyperparameters as used by model_fn
model_params = dict()  # fill this in


# Specify training schedule
train_params = dict()  # fill this in

assert(train_params['total_epochs'] % train_params['eval_every'] == 0)

###
# Don't change anything below this line
###
checkpoint_dir = "/tmp/tf_bow_sst_" + datetime.datetime.now().strftime("%Y%m%d
-%H%M")
if os.path.isdir(checkpoint_dir): shutil.rmtree(checkpoint_dir)
ds.vocab.write_projector_config(checkpoint_dir, "Encoder/Embedding_Layer/W_emb
ed")

model = tf.estimator.Estimator(model_fn=models.classifier_model_fn, params=mod
el_params, model_dir=checkpoint_dir)
print("\nTo view training (once it starts), run:\n")
print("    tensorboard --logdir='{:s}' --port 6006".format(checkpoint_dir))
print("\nThen in your browser, open: http://localhost:6006\n")

train_input_fn = patched_numpy_io.numpy_input_fn(
                    x={"ids": train_x, "ns": train_ns}, y=train_y,
                    batch_size=train_params['batch_size'],
                    num_epochs=train_params['eval_every'], shuffle=True, seed=
42)
dev_input_fn = patched_numpy_io.numpy_input_fn(
                    x={"ids": dev_x, "ns": dev_ns}, y=dev_y,
                    batch_size=128, num_epochs=1, shuffle=False)
for _ in range(train_params['total_epochs'] // train_params['eval_every']):
    model.train(input_fn=train_input_fn)
    model.evaluate(input_fn=dev_input_fn, name="dev")
```

```
Vocabulary (16,474 words) written to '/tmp/tf_bow_sst_20180303-2047/metadata.
tsv'
Projector config written to /tmp/tf_bow_sst_20180303-2047/projector_config.pb
txt
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': '/tmp/tf_bow_sst_20180303-2047',
'_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_step
s': None, '_save_checkpoints_secs': 600, '_session_config': None, '_keep_chec
kpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_step_count_ste
ps': 100, '_service': None, '_cluster_spec': <tensorflow.python.training.serv
er_lib.ClusterSpec object at 0x7f97fca6a400>, '_task_type': 'worker', '_task_
id': 0, '_master': '', '_is_chief': True, '_num_ps_replicas': 0, '_num_worker
_replicas': 1}

To view training (once it starts), run:

    tensorboard --logdir='/tmp/tf_bow_sst_20180303-2047' --port 6006

Then in your browser, open: http://localhost:6006

INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Saving checkpoints for 1 into /tmp/tf_bow_sst_20180303-2047/m
odel.ckpt.
INFO:tensorflow:loss = 0.885986, step = 1
INFO:tensorflow:global_step/sec: 177.72
INFO:tensorflow:loss = 0.469538, step = 101 (0.566 sec)
INFO:tensorflow:global_step/sec: 187.469
INFO:tensorflow:loss = 0.498528, step = 201 (0.534 sec)
INFO:tensorflow:global_step/sec: 193.262
INFO:tensorflow:loss = 0.473801, step = 301 (0.517 sec)
INFO:tensorflow:global_step/sec: 189.095
INFO:tensorflow:loss = 0.423145, step = 401 (0.529 sec)
INFO:tensorflow:Saving checkpoints for 433 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.416534.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:47:34
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-433
INFO:tensorflow:Finished evaluation at 2018-03-03-20:47:34
INFO:tensorflow:Saving dict for global step 433: accuracy = 0.669725, cross_e
ntropy_loss = 0.649828, global_step = 433, loss = 0.784433
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-433
INFO:tensorflow:Saving checkpoints for 434 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:loss = 0.188234, step = 434
INFO:tensorflow:global_step/sec: 173.648
INFO:tensorflow:loss = 0.26589, step = 534 (0.579 sec)
INFO:tensorflow:global_step/sec: 164.732
INFO:tensorflow:loss = 0.356204, step = 634 (0.608 sec)
INFO:tensorflow:global_step/sec: 164.053
INFO:tensorflow:loss = 0.363771, step = 734 (0.608 sec)
INFO:tensorflow:global_step/sec: 162.838
INFO:tensorflow:loss = 0.293923, step = 834 (0.614 sec)
INFO:tensorflow:Saving checkpoints for 866 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
```

```
INFO:tensorflow:Loss for final step: 0.24476.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:47:38
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-866
INFO:tensorflow:Finished evaluation at 2018-03-03-20:47:39
INFO:tensorflow:Saving dict for global step 866: accuracy = 0.761468, cross_e
ntropy_loss = 0.566895, global_step = 866, loss = 0.714818
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-866
INFO:tensorflow:Saving checkpoints for 867 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:loss = 0.181924, step = 867
INFO:tensorflow:global_step/sec: 167.501
INFO:tensorflow:loss = 0.210957, step = 967 (0.600 sec)
INFO:tensorflow:global_step/sec: 164.124
INFO:tensorflow:loss = 0.278434, step = 1067 (0.609 sec)
INFO:tensorflow:global_step/sec: 158.954
INFO:tensorflow:loss = 0.220856, step = 1167 (0.629 sec)
INFO:tensorflow:global_step/sec: 152.199
INFO:tensorflow:loss = 0.171707, step = 1267 (0.657 sec)
INFO:tensorflow:Saving checkpoints for 1299 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.145337.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:47:43
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-1299
INFO:tensorflow:Finished evaluation at 2018-03-03-20:47:43
INFO:tensorflow:Saving dict for global step 1299: accuracy = 0.75, cross_entr
opy_loss = 0.671933, global_step = 1299, loss = 0.832578
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-1299
INFO:tensorflow:Saving checkpoints for 1300 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:loss = 0.143874, step = 1300
INFO:tensorflow:global_step/sec: 172.931
INFO:tensorflow:loss = 0.167661, step = 1400 (0.581 sec)
INFO:tensorflow:global_step/sec: 183.571
INFO:tensorflow:loss = 0.203001, step = 1500 (0.545 sec)
INFO:tensorflow:global_step/sec: 185.411
INFO:tensorflow:loss = 0.136007, step = 1600 (0.539 sec)
INFO:tensorflow:global_step/sec: 184.922
INFO:tensorflow:loss = 0.140291, step = 1700 (0.541 sec)
INFO:tensorflow:Saving checkpoints for 1732 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.141041.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:47:47
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-1732
INFO:tensorflow:Finished evaluation at 2018-03-03-20:47:47
INFO:tensorflow:Saving dict for global step 1732: accuracy = 0.75344, cross_e
ntropy_loss = 0.704521, global_step = 1732, loss = 0.867512
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-1732
INFO:tensorflow:Saving checkpoints for 1733 into /tmp/tf_bow_sst_20180303-204
```

```
7/model.ckpt.
INFO:tensorflow:loss = 0.139353, step = 1733
INFO:tensorflow:global_step/sec: 176.419
INFO:tensorflow:loss = 0.139551, step = 1833 (0.570 sec)
INFO:tensorflow:global_step/sec: 186.295
INFO:tensorflow:loss = 0.169868, step = 1933 (0.537 sec)
INFO:tensorflow:global_step/sec: 149.292
INFO:tensorflow:loss = 0.134326, step = 2033 (0.670 sec)
INFO:tensorflow:global_step/sec: 187.396
INFO:tensorflow:loss = 0.146286, step = 2133 (0.534 sec)
INFO:tensorflow:Saving checkpoints for 2165 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.122978.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:47:50
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-2165
INFO:tensorflow:Finished evaluation at 2018-03-03-20:47:51
INFO:tensorflow:Saving dict for global step 2165: accuracy = 0.748853, cross_
entropy_loss = 0.737417, global_step = 2165, loss = 0.899896
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-2165
INFO:tensorflow:Saving checkpoints for 2166 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:loss = 0.132699, step = 2166
INFO:tensorflow:global_step/sec: 172.357
INFO:tensorflow:loss = 0.131742, step = 2266 (0.583 sec)
INFO:tensorflow:global_step/sec: 189.464
INFO:tensorflow:loss = 0.154877, step = 2366 (0.528 sec)
INFO:tensorflow:global_step/sec: 191.557
INFO:tensorflow:loss = 0.13012, step = 2466 (0.522 sec)
INFO:tensorflow:global_step/sec: 194.585
INFO:tensorflow:loss = 0.138189, step = 2566 (0.514 sec)
INFO:tensorflow:Saving checkpoints for 2598 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.117286.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:47:54
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-2598
INFO:tensorflow:Finished evaluation at 2018-03-03-20:47:54
INFO:tensorflow:Saving dict for global step 2598: accuracy = 0.74656, cross_e
ntropy_loss = 0.741358, global_step = 2598, loss = 0.902936
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-2598
INFO:tensorflow:Saving checkpoints for 2599 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:loss = 0.131494, step = 2599
INFO:tensorflow:global_step/sec: 173.171
INFO:tensorflow:loss = 0.132259, step = 2699 (0.580 sec)
INFO:tensorflow:global_step/sec: 185.455
INFO:tensorflow:loss = 0.147066, step = 2799 (0.539 sec)
INFO:tensorflow:global_step/sec: 187.585
INFO:tensorflow:loss = 0.127109, step = 2899 (0.533 sec)
INFO:tensorflow:global_step/sec: 189.915
INFO:tensorflow:loss = 0.13624, step = 2999 (0.526 sec)
INFO:tensorflow:Saving checkpoints for 3031 into /tmp/tf_bow_sst_20180303-204
```

```
7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.114515.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:47:58
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-3031
INFO:tensorflow:Finished evaluation at 2018-03-03-20:47:58
INFO:tensorflow:Saving dict for global step 3031: accuracy = 0.745413, cross_
entropy_loss = 0.738933, global_step = 3031, loss = 0.899646
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-3031
INFO:tensorflow:Saving checkpoints for 3032 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:loss = 0.129296, step = 3032
INFO:tensorflow:global_step/sec: 176.498
INFO:tensorflow:loss = 0.131, step = 3132 (0.570 sec)
INFO:tensorflow:global_step/sec: 187.925
INFO:tensorflow:loss = 0.142916, step = 3232 (0.532 sec)
INFO:tensorflow:global_step/sec: 190.519
INFO:tensorflow:loss = 0.125466, step = 3332 (0.525 sec)
INFO:tensorflow:global_step/sec: 183.567
INFO:tensorflow:loss = 0.13365, step = 3432 (0.545 sec)
INFO:tensorflow:Saving checkpoints for 3464 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.112423.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:48:02
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-3464
INFO:tensorflow:Finished evaluation at 2018-03-03-20:48:02
INFO:tensorflow:Saving dict for global step 3464: accuracy = 0.752294, cross_
entropy_loss = 0.734868, global_step = 3464, loss = 0.894893
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-3464
INFO:tensorflow:Saving checkpoints for 3465 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:loss = 0.127591, step = 3465
INFO:tensorflow:global_step/sec: 174.564
INFO:tensorflow:loss = 0.130037, step = 3565 (0.576 sec)
INFO:tensorflow:global_step/sec: 192.033
INFO:tensorflow:loss = 0.139981, step = 3665 (0.521 sec)
INFO:tensorflow:global_step/sec: 192.156
INFO:tensorflow:loss = 0.124285, step = 3765 (0.520 sec)
INFO:tensorflow:global_step/sec: 189.657
INFO:tensorflow:loss = 0.132052, step = 3865 (0.527 sec)
INFO:tensorflow:Saving checkpoints for 3897 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.110869.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:48:05
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-3897
INFO:tensorflow:Finished evaluation at 2018-03-03-20:48:06
INFO:tensorflow:Saving dict for global step 3897: accuracy = 0.74656, cross_e
ntropy_loss = 0.730545, global_step = 3897, loss = 0.889995
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-3897
```

```
INFO:tensorflow:Saving checkpoints for 3898 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:loss = 0.126226, step = 3898
INFO:tensorflow:global_step/sec: 179.948
INFO:tensorflow:loss = 0.129009, step = 3998 (0.559 sec)
INFO:tensorflow:global_step/sec: 193.244
INFO:tensorflow:loss = 0.137765, step = 4098 (0.517 sec)
INFO:tensorflow:global_step/sec: 193.276
INFO:tensorflow:loss = 0.123207, step = 4198 (0.517 sec)
INFO:tensorflow:global_step/sec: 192.926
INFO:tensorflow:loss = 0.130796, step = 4298 (0.518 sec)
INFO:tensorflow:Saving checkpoints for 4330 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.109684.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:48:09
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-4330
INFO:tensorflow:Finished evaluation at 2018-03-03-20:48:09
INFO:tensorflow:Saving dict for global step 4330: accuracy = 0.748853, cross_
entropy_loss = 0.726504, global_step = 4330, loss = 0.885498
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-4330
INFO:tensorflow:Saving checkpoints for 4331 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:loss = 0.125046, step = 4331
INFO:tensorflow:global_step/sec: 175.784
INFO:tensorflow:loss = 0.128157, step = 4431 (0.572 sec)
INFO:tensorflow:global_step/sec: 186.279
INFO:tensorflow:loss = 0.136011, step = 4531 (0.537 sec)
INFO:tensorflow:global_step/sec: 190.263
INFO:tensorflow:loss = 0.122305, step = 4631 (0.526 sec)
INFO:tensorflow:global_step/sec: 193.117
INFO:tensorflow:loss = 0.129784, step = 4731 (0.518 sec)
INFO:tensorflow:Saving checkpoints for 4763 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.108678.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:48:13
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-4763
INFO:tensorflow:Finished evaluation at 2018-03-03-20:48:13
INFO:tensorflow:Saving dict for global step 4763: accuracy = 0.747706, cross_
entropy_loss = 0.722865, global_step = 4763, loss = 0.881489
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-4763
INFO:tensorflow:Saving checkpoints for 4764 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:loss = 0.124047, step = 4764
INFO:tensorflow:global_step/sec: 179.055
INFO:tensorflow:loss = 0.127405, step = 4864 (0.562 sec)
INFO:tensorflow:global_step/sec: 192.903
INFO:tensorflow:loss = 0.134576, step = 4964 (0.518 sec)
INFO:tensorflow:global_step/sec: 193.963
INFO:tensorflow:loss = 0.121472, step = 5064 (0.515 sec)
INFO:tensorflow:global_step/sec: 190.76
INFO:tensorflow:loss = 0.128942, step = 5164 (0.524 sec)
```

```
INFO:tensorflow:Saving checkpoints for 5196 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.107818.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:48:17
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-5196
INFO:tensorflow:Finished evaluation at 2018-03-03-20:48:17
INFO:tensorflow:Saving dict for global step 5196: accuracy = 0.748853, cross_
entropy_loss = 0.719579, global_step = 5196, loss = 0.8779
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-5196
INFO:tensorflow:Saving checkpoints for 5197 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:loss = 0.123172, step = 5197
INFO:tensorflow:global_step/sec: 181.99
INFO:tensorflow:loss = 0.126752, step = 5297 (0.552 sec)
INFO:tensorflow:global_step/sec: 193.696
INFO:tensorflow:loss = 0.133374, step = 5397 (0.516 sec)
INFO:tensorflow:global_step/sec: 193.389
INFO:tensorflow:loss = 0.120725, step = 5497 (0.517 sec)
INFO:tensorflow:global_step/sec: 189.657
INFO:tensorflow:loss = 0.128231, step = 5597 (0.527 sec)
INFO:tensorflow:Saving checkpoints for 5629 into /tmp/tf_bow_sst_20180303-204
7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.10706.
INFO:tensorflow:Starting evaluation at 2018-03-03-20:48:20
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-5629
INFO:tensorflow:Finished evaluation at 2018-03-03-20:48:20
INFO:tensorflow:Saving dict for global step 5629: accuracy = 0.745413, cross_
entropy_loss = 0.71656, global_step = 5629, loss = 0.874625
```

In [14]:
```
test_input_fn = patched_numpy_io.numpy_input_fn(
                    x={"ids": test_x, "ns": test_ns}, y=test_y,
                    batch_size=128, num_epochs=1, shuffle=False)
eval_metrics = model.evaluate(input_fn=test_input_fn, name="test")
print("Accuracy on test set: {:.02%}".format(eval_metrics['accuracy']))
```

```
INFO:tensorflow:Starting evaluation at 2018-03-03-20:48:21
INFO:tensorflow:Restoring parameters from /tmp/tf_bow_sst_20180303-2047/mode
l.ckpt-5629
INFO:tensorflow:Finished evaluation at 2018-03-03-20:48:21
INFO:tensorflow:Saving dict for global step 5629: accuracy = 0.769907, cross_
entropy_loss = 0.657831, global_step = 5629, loss = 0.812804
Accuracy on test set: 76.99%
```