**CSC 578 Project 3**

**Yiyang Yang**

MNIST Softmax Regression

[ X] Included commented code

[ X] Included good answers for Questions 1.1 to 1.3

MNIST Convolution

[X ] Included 2 versions of commented code

[X ] Included good answers for Questions 2.1 to 2.12

[ X] Included 2 requested graph images.

Vector representations of words

[X ] Included commented code

[X ] Included requested outputs

[X ] Included good answers for Questions 3.1 to 3.5

[ X] Did NOT include the MNIST data with my submission

Incomplete, Details \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Not sure, Details \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**MNIST Softmax Regression**

Code:

#Import dataset from tensorflow

from tensorflow.examples.tutorials.mnist import input\_data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot = True)

#Start session

import tensorflow as tf

sess = tf.InteractiveSession()

#Create two placeholders to

x = tf.placeholder(tf.float32, shape = [None, 784])

y\_ = tf.placeholder(tf.float32, shape = [None, 10])

#Two variables weight and bias

W = tf.Variable(tf.zeros([784, 10]))

b = tf.Variable(tf.zeros([10]))

sess.run(tf.global\_variables\_initializer())

#Regression model

y = tf.matmul(x, W) + b

#Loss function

cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels = y\_, logits = y))

#Train the model

train\_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross\_entropy)

#Apply gradient descent

for \_ in range(10000):

batch = mnist.train.next\_batch(100)

train\_step.run(feed\_dict = {x: batch[0], y\_: batch[1]})

#Check the results and get the accuracy

correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y\_, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

print(accuracy.eval(feed\_dict = {x: mnist.test.images, y\_: mnist.test.labels}))

Question 1.1: What was the final accuracy?

0.9133

Change the code to make it run for 10,000 epochs and run it again.

Question 1.2: What was the final accuracy after this modification?

0.9208

Question 1.3: Was the difference surprising or not? Explain.

I don’t think it is surprising, more epochs make it more accuracy.

**MNIST Convolutional Network**

Code:

**Version 1 (Original code):**

#Define two function to get variables

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

def conv2d(x, W):

return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1],

strides=[1, 2, 2, 1], padding='SAME')

#First convolutional layer

W\_conv1 = weight\_variable([5, 5, 1, 32])

b\_conv1 = bias\_variable([32])

x\_image = tf.reshape(x, [-1, 28, 28, 1])

h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)

h\_pool1 = max\_pool\_2x2(h\_conv1)

#Secod Convolutional Layer

W\_conv2 = weight\_variable([5, 5, 32, 64])

b\_conv2 = bias\_variable([64])

h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

h\_pool2 = max\_pool\_2x2(h\_conv2)

#Densely Connected Layer

W\_fc1 = weight\_variable([7 \* 7 \* 64, 1024])

b\_fc1 = bias\_variable([1024])

h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 7\*7\*64])

h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

#Dropout

keep\_prob = tf.placeholder(tf.float32)

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

#Readout Layer

W\_fc2 = weight\_variable([1024, 10])

b\_fc2 = bias\_variable([10])

y\_conv = tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2

#Train the model and get the accuracy

cross\_entropy = tf.reduce\_mean(

tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_, logits=y\_conv))

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

correct\_prediction = tf.equal(tf.argmax(y\_conv, 1), tf.argmax(y\_, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

for i in range(20000):

batch = mnist.train.next\_batch(50)

if i % 100 == 0:

train\_accuracy = accuracy.eval(feed\_dict={

x: batch[0], y\_: batch[1], keep\_prob: 1.0})

print('step %d, training accuracy %g' % (i, train\_accuracy))

train\_step.run(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 0.5})

print('test accuracy %g' % accuracy.eval(feed\_dict={

x: mnist.test.images, y\_: mnist.test.labels, keep\_prob: 1.0}))

**Version 2:**

#Convolutional network

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev = 0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape = shape)

return tf.Variable(initial)

def conv2d(x, W):

return tf.nn.conv2d(x, W, strides = [1, 1, 1, 1], padding = 'SAME')

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x, ksize = [1, 2, 2, 1], strides = [1, 2, 2, 1], padding = 'SAME')

#Variable summary report

def variable\_summaries(var):

with tf.name\_scope('summaries'):

mean = tf.reduce\_mean(var)

tf.summary.scalar('mean', mean)

with tf.name\_scope('stddev'):

stddev = tf.sqrt(tf.reduce\_mean(tf.square(var - mean)))

tf.summary.scalar('stddev', stddev)

tf.summary.scalar('max', tf.reduce\_max(var))

tf.summary.scalar('min', tf.reduce\_min(var))

tf.summary.histogram('histogram', var)

#Convolutional Layer

def conv\_layer(input, input\_dim, output\_dim, name = 'conv'):

with tf.name\_scope(name):

W\_conv = weight\_variable([5, 5, input\_dim, output\_dim])

b\_conv = bias\_variable([output\_dim])

conv = tf.nn.relu(conv2d(input, W\_conv,) + b\_conv)

variable\_summaries(W\_conv)

variable\_summaries(b\_conv)

variable\_summaries(conv)

return max\_pool\_2x2(conv)

#Fully Connected Layer

def fc\_layer(input, input\_dim, output\_dim, name):

with tf.name\_scope(name):

W\_fc = weight\_variable([input\_dim, output\_dim])

b\_fc = bias\_variable([output\_dim])

activation = tf.nn.relu(tf.matmul(input, W\_fc) + b\_fc)

variable\_summaries(W\_fc)

variable\_summaries(b\_fc)

variable\_summaries(activation)

return activation

x\_image = tf.reshape(x, [-1, 28, 28, 1])

tf.summary.image('input', x\_image, 3)

conv1 = conv\_layer(x\_image, 1, 32, 'conv1')

conv2 = conv\_layer(conv1, 32, 64, 'conv2')

flat = tf.reshape(conv2, [-1, 7\*7\*64])

fc1 = fc\_layer(flat, 7\*7\*64, 1024,'fc1')

#Dropout

with tf.name\_scope('dropout'):

keep\_prob = tf.placeholder(tf.float32)

tf.summary.scalar('dropout\_keep\_probability', keep\_prob)

h\_fc1\_drop = tf.nn.dropout(fc1, keep\_prob)

y\_conv = fc\_layer(h\_fc1\_drop, 1024, 10, 'fc2')

#Get accuracy

with tf.name\_scope('total'):

cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels = y\_, logits = y\_conv))

tf.summary.scalar('cross\_entropy', cross\_entropy)

with tf.name\_scope('train'):

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

with tf.name\_scope('accuracy'):

with tf.name\_scope('correct\_prediction'):

correct\_prediction = tf.equal(tf.argmax(y\_conv, 1), tf.argmax(y\_, 1))

with tf.name\_scope('accuracy'):

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

tf.summary.scalar('Accuracy', accuracy)

#Train model and write summary

merged = tf.summary.merge\_all()

writer = tf.summary.FileWriter('D:\CSC 578\Project 3')

writer.add\_graph(tf.get\_default\_graph())

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

for i in range(20000):

batch = mnist.train.next\_batch(50)

if i % 10 == 0:

summary = sess.run(merged, feed\_dict = {x: batch[0], y\_: batch[1], keep\_prob: 1.0})

writer.add\_summary(summary, i)

if i % 100 == 0:

train\_accuracy = accuracy.eval(feed\_dict = {x: batch[0], y\_: batch[1], keep\_prob: 1.0})

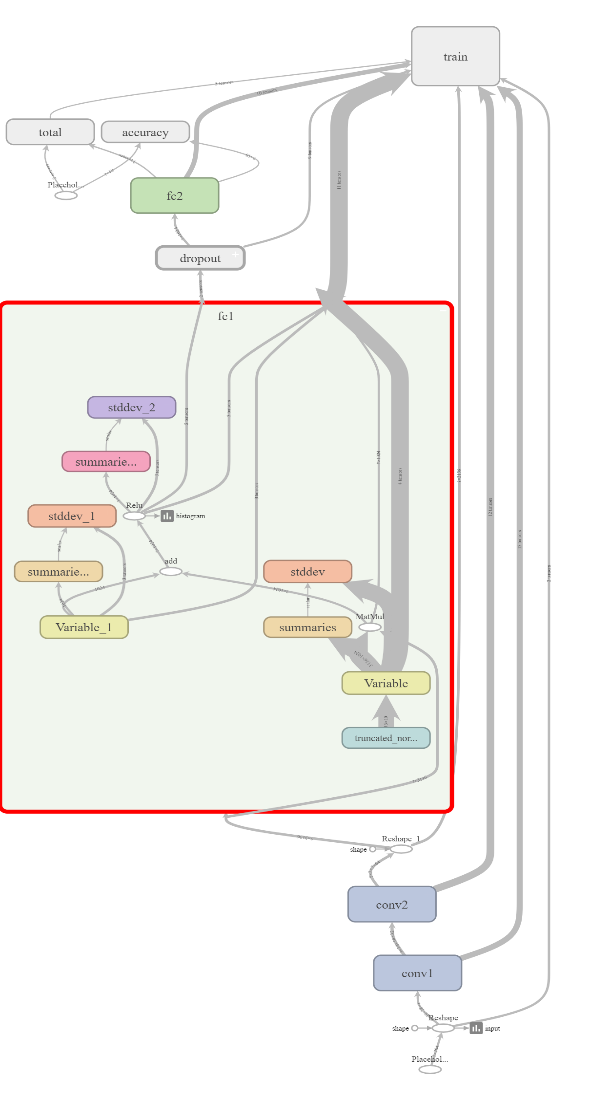
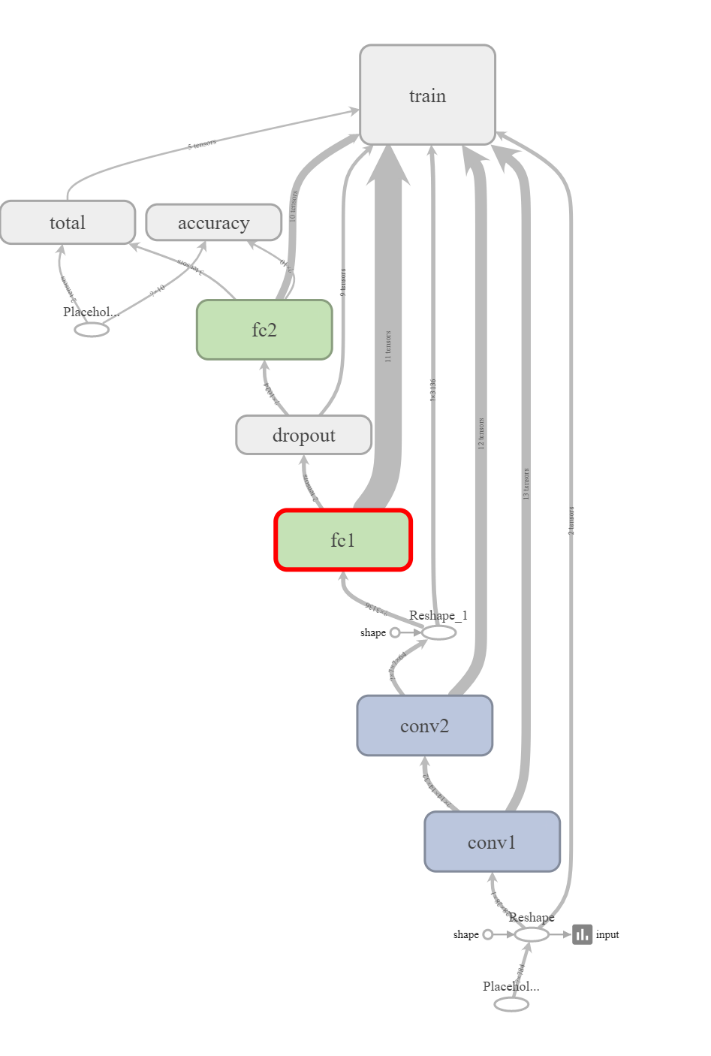
print('Step %d, training accuracy %g' % (i, train\_accuracy))

train\_step.run(feed\_dict = {x: batch[0], y\_: batch[1], keep\_prob: 0.5})

print('Test accuracy %g' % accuracy.eval(feed\_dict = {x: mnist.test.images, y\_:mnist.test.labels, keep\_prob: 1.0}))

Question 2.1: What was the final accuracy (on the test set, of course!) for the convolutional model?

Test accuracy: 0.9929



Question 2.2: What information can you get out of it?

The graph shows all the layers before training, it makes the black box clearer.

Question 2.3: What do the nodes in the graph tell you?

These nodes tell me all the train procedures, and all the variables.

Question 2.4: What do the histograms for the biases show about their distributions?

Right skewed.

Question 2.5: What do the histograms for the weights show about their distributions?

Nearly normal distribution

Question 2.6: Anything different about the weights near 0?

There is not significant different when weight near 0.

Question 2.7: What is your interpretation of this?

I think the model works well at this situation

Question 2.8: What do the bias histograms show about their distributions?

Right skewed

Question 2.9: What do the weight histograms show about their distributions?

Nearly normal distribution

Question 2.10: Anything different about weights near 0?

I think there are not any significant difference.

Question 2.11: What is your interpretation of this?

I think the model works well at this situation.

Question 2.12: Can you figure out what is displayed under the DISTRIBUTIONS tab? (I can't, but it looks cool.)

I think it looks like a 2D projection of the histograms of those varaibles.

**Vector Representations of Words**

**Code:**

from \_\_future\_\_ import absolute\_import

from \_\_future\_\_ import division

from \_\_future\_\_ import print\_function

import collections

import math

import os

import random

from tempfile import gettempdir

import zipfile

import numpy as np

from six.moves import urllib

from six.moves import xrange # pylint: disable=redefined-builtin

import tensorflow as tf

# Step 1: Download the data.

url = 'http://mattmahoney.net/dc/'

# pylint: disable=redefined-outer-name

#This function is going to download files

def maybe\_download(filename, expected\_bytes):

"""Download a file if not present, and make sure it's the right size."""

local\_filename = os.path.join(gettempdir(), filename)

if not os.path.exists(local\_filename):

local\_filename, \_ = urllib.request.urlretrieve(url + filename,

local\_filename)

statinfo = os.stat(local\_filename)

if statinfo.st\_size == expected\_bytes:

print('Found and verified', filename)

else:

print(statinfo.st\_size)

raise Exception('Failed to verify ' + local\_filename +

'. Can you get to it with a browser?')

return local\_filename

filename = maybe\_download('text8.zip', 31344016)

# Read the data into a list of strings.

def read\_data(filename):

"""Extract the first file enclosed in a zip file as a list of words."""

with zipfile.ZipFile(filename) as f:

data = tf.compat.as\_str(f.read(f.namelist()[0])).split()

return data

vocabulary = read\_data(filename)

print('Data size', len(vocabulary))

# Step 2: Build the dictionary and replace rare words with UNK token.

vocabulary\_size = 50000

#Creating dataset with word count

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:

index = dictionary.get(word, 0)

if index == 0: # dictionary['UNK']

unk\_count += 1

data.append(index)

count[0][1] = unk\_count

reversed\_dictionary = dict(zip(dictionary.values(), dictionary.keys()))

return data, count, dictionary, reversed\_dictionary

# Filling 4 global variables:

# data - list of codes (integers from 0 to vocabulary\_size-1).

# This is the original text but words are replaced by their codes

# count - map of words(strings) to count of occurrences

# dictionary - map of words(strings) to their codes(integers)

# reverse\_dictionary - maps codes(integers) to words(strings)

data, count, dictionary, reverse\_dictionary = build\_dataset(vocabulary,

vocabulary\_size)

del vocabulary # Hint to reduce memory.

print('Most common words (+UNK)', count[:5])

print('Sample data', data[:10], [reverse\_dictionary[i] for i in data[:10]])

data\_index = 0

# Step 3: Function to generate a training batch for the skip-gram model.

def generate\_batch(batch\_size, num\_skips, skip\_window):

global data\_index

assert batch\_size % num\_skips == 0

assert num\_skips <= 2 \* skip\_window

batch = np.ndarray(shape=(batch\_size), dtype=np.int32)

labels = np.ndarray(shape=(batch\_size, 1), dtype=np.int32)

span = 2 \* skip\_window + 1 # [ skip\_window target skip\_window ]

buffer = collections.deque(maxlen=span)

if data\_index + span > len(data):

data\_index = 0

buffer.extend(data[data\_index:data\_index + span])

data\_index += span

for i in range(batch\_size // num\_skips):

context\_words = [w for w in range(span) if w != skip\_window]

words\_to\_use = random.sample(context\_words, num\_skips)

for j, context\_word in enumerate(words\_to\_use):

batch[i \* num\_skips + j] = buffer[skip\_window]

labels[i \* num\_skips + j, 0] = buffer[context\_word]

if data\_index == len(data):

buffer[:] = data[:span]

data\_index = span

else:

buffer.append(data[data\_index])

data\_index += 1

# Backtrack a little bit to avoid skipping words in the end of a batch

data\_index = (data\_index + len(data) - span) % len(data)

return batch, labels

batch, labels = generate\_batch(batch\_size=8, num\_skips=2, skip\_window=1)

for i in range(8):

print(batch[i], reverse\_dictionary[batch[i]],

'->', labels[i, 0], reverse\_dictionary[labels[i, 0]])

# Step 4: Build and train a skip-gram model.

batch\_size = 128

embedding\_size = 128 # Dimension of the embedding vector.

skip\_window = 1 # How many words to consider left and right.

num\_skips = 2 # How many times to reuse an input to generate a label.

num\_sampled = 64 # Number of negative examples to sample.

# We pick a random validation set to sample nearest neighbors. Here we limit the

# validation samples to the words that have a low numeric ID, which by

# construction are also the most frequent. These 3 variables are used only for

# displaying model accuracy, they don't affect calculation.

valid\_size = 16 # Random set of words to evaluate similarity on.

valid\_window = 100 # Only pick dev samples in the head of the distribution.

valid\_examples = np.random.choice(valid\_window, valid\_size, replace=False)

graph = tf.Graph()

with graph.as\_default():

# Input data.

train\_inputs = tf.placeholder(tf.int32, shape=[batch\_size])

train\_labels = tf.placeholder(tf.int32, shape=[batch\_size, 1])

valid\_dataset = tf.constant(valid\_examples, dtype=tf.int32)

# Ops and variables pinned to the CPU because of missing GPU implementation

with tf.device('/cpu:0'):

# Look up embeddings for inputs.

embeddings = tf.Variable(

tf.random\_uniform([vocabulary\_size, embedding\_size], -1.0, 1.0))

embed = tf.nn.embedding\_lookup(embeddings, train\_inputs)

# Construct the variables for the NCE loss

nce\_weights = tf.Variable(

tf.truncated\_normal([vocabulary\_size, embedding\_size],

stddev=1.0 / math.sqrt(embedding\_size)))

nce\_biases = tf.Variable(tf.zeros([vocabulary\_size]))

# Compute the average NCE loss for the batch.

# tf.nce\_loss automatically draws a new sample of the negative labels each

# time we evaluate the loss.

# Explanation of the meaning of NCE loss:

# http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

loss = tf.reduce\_mean(

tf.nn.nce\_loss(weights=nce\_weights,

biases=nce\_biases,

labels=train\_labels,

inputs=embed,

num\_sampled=num\_sampled,

num\_classes=vocabulary\_size))

# Construct the SGD optimizer using a learning rate of 1.0.

optimizer = tf.train.GradientDescentOptimizer(1.0).minimize(loss)

# Compute the cosine similarity between minibatch examples and all embeddings.

norm = tf.sqrt(tf.reduce\_sum(tf.square(embeddings), 1, keep\_dims=True))

normalized\_embeddings = embeddings / norm

valid\_embeddings = tf.nn.embedding\_lookup(

normalized\_embeddings, valid\_dataset)

similarity = tf.matmul(

valid\_embeddings, normalized\_embeddings, transpose\_b=True)

# Add variable initializer.

init = tf.global\_variables\_initializer()

# Step 5: Begin training.

num\_steps = 100001

with tf.Session(graph=graph) as session:

# We must initialize all variables before we use them.

init.run()

print('Initialized')

average\_loss = 0

for step in xrange(num\_steps):

batch\_inputs, batch\_labels = generate\_batch(

batch\_size, num\_skips, skip\_window)

feed\_dict = {train\_inputs: batch\_inputs, train\_labels: batch\_labels}

# We perform one update step by evaluating the optimizer op (including it

# in the list of returned values for session.run()

\_, loss\_val = session.run([optimizer, loss], feed\_dict=feed\_dict)

average\_loss += loss\_val

if step % 2000 == 0:

if step > 0:

average\_loss /= 2000

# The average loss is an estimate of the loss over the last 2000 batches.

print('Average loss at step ', step, ': ', average\_loss)

average\_loss = 0

# Note that this is expensive (~20% slowdown if computed every 500 steps)

if step % 10000 == 0:

sim = similarity.eval()

for i in xrange(valid\_size):

valid\_word = reverse\_dictionary[valid\_examples[i]]

top\_k = 8 # number of nearest neighbors

nearest = (-sim[i, :]).argsort()[1:top\_k + 1]

log\_str = 'Nearest to %s:' % valid\_word

for k in xrange(top\_k):

close\_word = reverse\_dictionary[nearest[k]]

log\_str = '%s %s,' % (log\_str, close\_word)

print(log\_str)

final\_embeddings = normalized\_embeddings.eval()

# Step 6: Visualize the embeddings.

# pylint: disable=missing-docstring

# Function to draw visualization of distance between embeddings.

def plot\_with\_labels(low\_dim\_embs, labels, filename):

assert low\_dim\_embs.shape[0] >= len(labels), 'More labels than embeddings'

plt.figure(figsize=(18, 18)) # in inches

for i, label in enumerate(labels):

x, y = low\_dim\_embs[i, :]

plt.scatter(x, y)

plt.annotate(label,

xy=(x, y),

xytext=(5, 2),

textcoords='offset points',

ha='right',

va='bottom')

plt.savefig(filename)

try:

# pylint: disable=g-import-not-at-top

from sklearn.manifold import TSNE

import matplotlib.pyplot as plt

tsne = TSNE(perplexity=30, n\_components=2, init='pca', n\_iter=5000, method='exact')

plot\_only = 500

low\_dim\_embs = tsne.fit\_transform(final\_embeddings[:plot\_only, :])

labels = [reverse\_dictionary[i] for i in xrange(plot\_only)]

plot\_with\_labels(low\_dim\_embs, labels, os.path.join(gettempdir(), 'tsne.png'))

except ImportError as ex:

print('Please install sklearn, matplotlib, and scipy to show embeddings.')

print(ex)

**First Run:**

Average loss at step 92000 : 4.67031042731

Average loss at step 94000 : 4.72828114879

Average loss at step 96000 : 4.67888811088

Average loss at step 98000 : 4.58808357739

Average loss at step 100000 : 4.70625274384

Nearest to be: been, have, is, by, was, were, become, are,

Nearest to into: from, through, during, glosses, bucharest, careless, on, upanija,

Nearest to were: are, was, have, had, is, be, been, amalthea,

Nearest to new: creeks, cebus, aedh, marx, second, victoriae, ovary, joram,

Nearest to s: his, gland, and, operatorname, escuela, ursus, four, coke,

Nearest to six: seven, eight, five, four, three, nine, ursus, two,

Nearest to seven: eight, six, five, four, nine, three, zero, ursus,

Nearest to from: into, in, during, across, circ, four, against, eight,

Nearest to may: can, could, will, would, must, should, might, cannot,

Nearest to time: circ, damp, zimmer, counterculture, huntington, untouchables, dub, seabed,

Nearest to which: that, this, what, but, it, also, agouti, pulau,

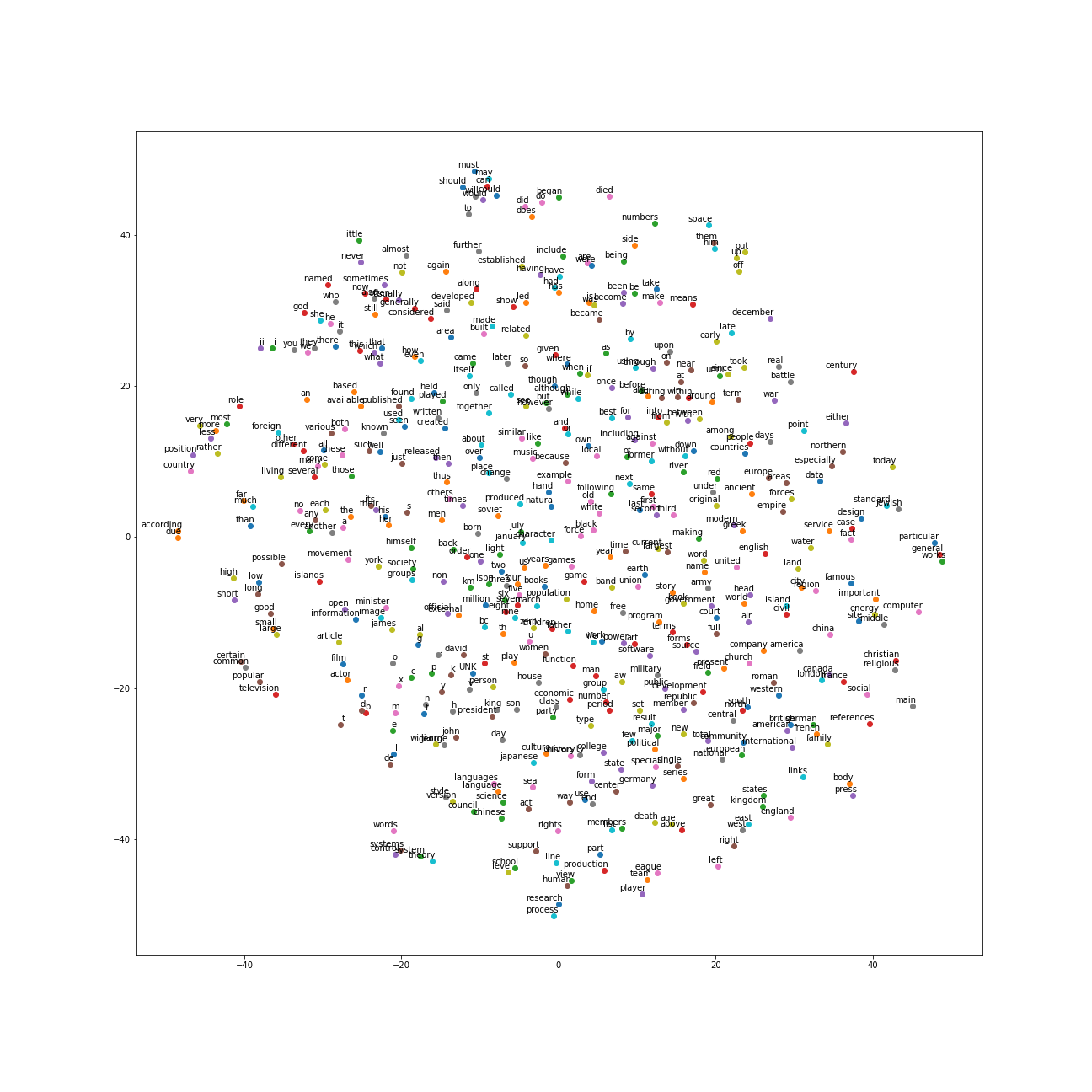
Nearest to also: often, now, which, still, akihabara, originally, peacocks, who,

Nearest to so: thaler, dinar, retroviral, circ, arctos, dasyprocta, fife, intensify,

Nearest to nine: eight, seven, six, five, zero, four, three, ursus,

Nearest to zero: eight, five, seven, nine, four, six, three, thaler,

Nearest to states: lymphoma, mills, michelob, confederate, dasyprocta, kingdom, gogh, timeless,



**Second Run:**

Average loss at step 92000 : 4.67540076864

Average loss at step 94000 : 4.72052379036

Average loss at step 96000 : 4.69934005034

Average loss at step 98000 : 4.58196404159

Average loss at step 100000 : 4.69652464581

Nearest to he: it, she, they, there, who, operatorname, nationalists, originally,

Nearest to from: into, in, through, during, under, between, while, recitative,

Nearest to zero: five, eight, seven, four, nine, six, three, ursus,

Nearest to seven: eight, six, five, four, nine, three, zero, circ,

Nearest to at: in, during, on, peacocks, circ, thibetanus, near, from,

Nearest to can: may, would, will, could, must, should, might, cannot,

Nearest to as: agouti, microsite, microcebus, apatosaurus, abet, indictment, dasyprocta, when,

Nearest to years: theremin, times, heuristics, nourished, four, abet, episodes, inches,

Nearest to while: where, but, after, and, although, though, when, from,

Nearest to these: some, many, several, such, both, they, various, which,

Nearest to th: six, seven, eight, nine, ursus, kapoor, altenberg, twentieth,

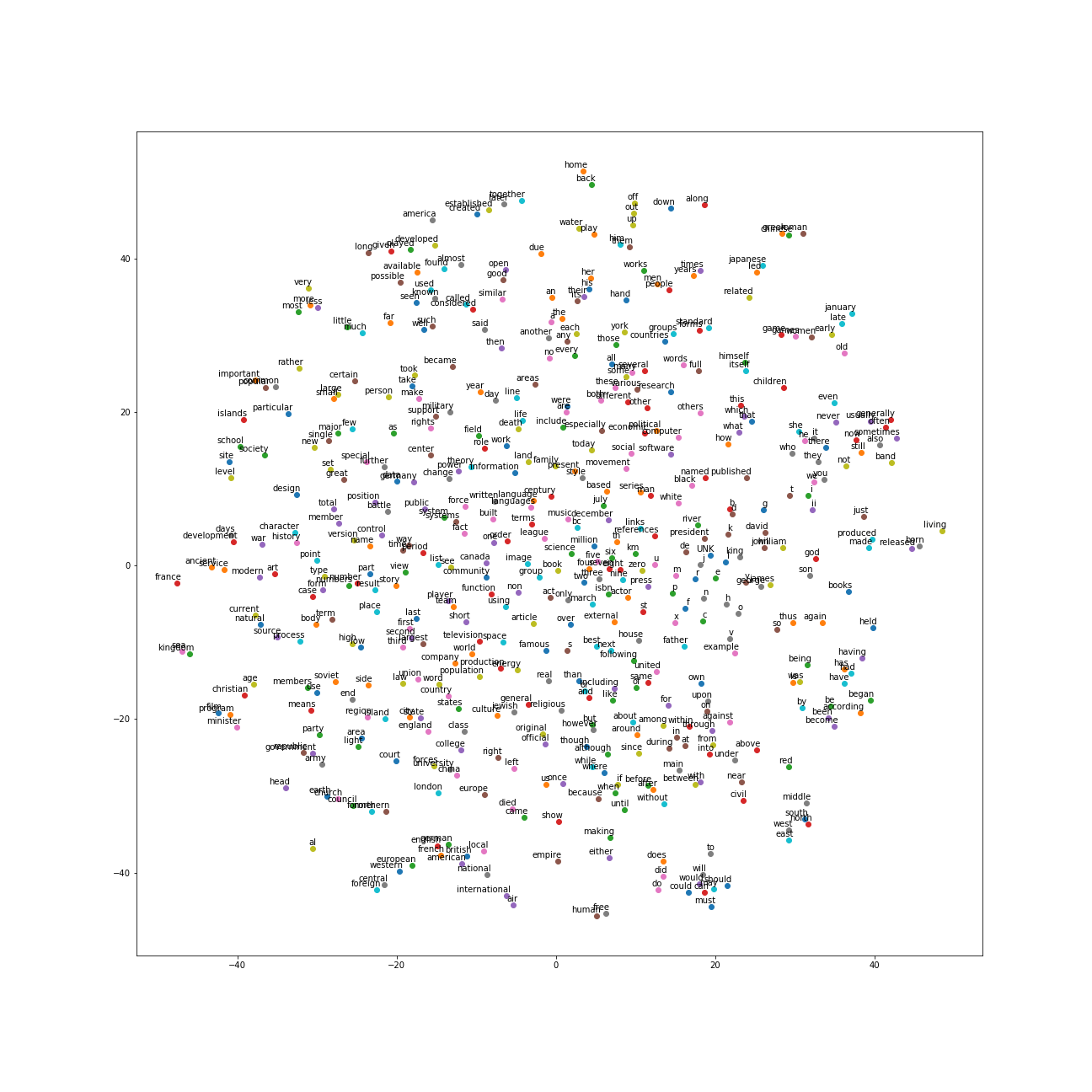
Nearest to s: his, ursus, kifl, and, agouti, four, fingerprints, acetaldehyde,

Nearest to other: including, many, boutros, various, these, several, circ, southwestern,

Nearest to its: their, his, the, her, nathanael, some, ursus, agouti,

Nearest to who: he, they, also, and, which, monitored, linebarger, i,

Nearest to however: but, that, which, operatorname, and, pontificia, though, abet,



**Third Run:**

Average loss at step 92000 : 4.66649640357

Average loss at step 94000 : 4.72834549069

Average loss at step 96000 : 4.69100591993

Average loss at step 98000 : 4.59216784859

Average loss at step 100000 : 4.69684509552

Nearest to he: it, she, they, who, there, leeward, but, we,

Nearest to were: are, have, had, was, be, is, while, iit,

Nearest to history: revival, cn, support, afrikaans, agouti, dac, stitch, presbyterianism,

Nearest to have: had, has, were, are, be, include, circ, titration,

Nearest to war: angolan, predict, sharpen, rush, fashions, constantine, benzyl, sects,

Nearest to not: generally, to, it, typically, they, quite, you, mitral,

Nearest to and: or, but, callithrix, thaler, iit, agouti, ursus, circ,

Nearest to two: three, four, five, six, one, seven, circ, eight,

Nearest to by: be, was, as, during, allegro, iit, michelob, with,

Nearest to three: five, four, seven, six, eight, two, circ, nine,

Nearest to or: and, callithrix, busan, agouti, recitative, than, ursus, circ,

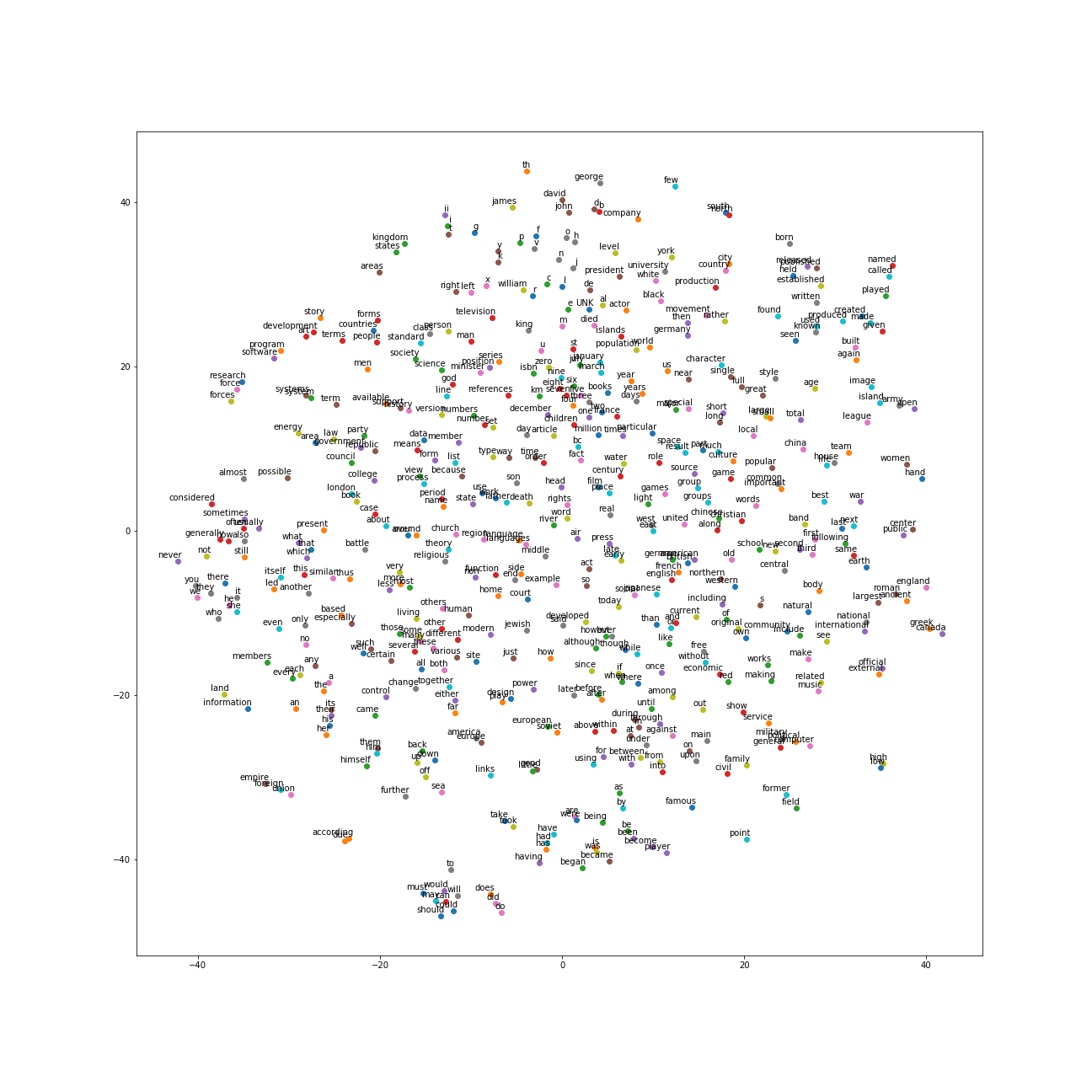
Nearest to of: circ, in, microcebus, agouti, including, callithrix, recitative, ursus,

Nearest to one: two, six, seven, five, dasyprocta, four, ursus, eight,

Nearest to when: if, after, before, during, but, recitative, for, while,

Nearest to other: many, some, various, these, operatorname, different, agouti, including,

Nearest to i: t, ii, we, g, they, omari, ursus, UNK,



Question 3.1: In about a paragraph, describe (in your own words, of course) this task. What is the purpose, the data, the general idea for the learning approach.

This task is going to classify the word and then get the nearest words of the target word, the data is a document call text8.zip. This is the word2vec model, it will find the most common word in the document and then do the classification to find the word nearest to them from the document.

Question 3.2: How did it determine which words to find the neighbors of?

I think this model is make the words into vectors and then compare the similarity of these vectors and get the result of the classification.

Question 3.3: Were there any surprises in the results (good or bad)? Give example outputs.

Yes, there are. The nearest of those number such as six and seven in first run, there is a word call ‘ursus’ in it, I think this should be the bad result. The good result is also those number, and the preposition word such as from, these are the good results.

Question 3.4: What could this be useful for (in your own words)?

I think this will be very helpful for AI and human to learn a new language, since it will cluster all the similar word together, and easy to learn a class of words. Also, this will be very powerful in translation.

Question 3.5: What is noise-contrastive training?

Word2vec will get the target word from the other noise words, after calculation target word will be given high probability and noise words will be given low probability and then get the nearest word for the target word.