In [22]:

```
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
import sys
import sklearn
import matplotlib
import scipy
%matplotlib inline


filename = 'amputation2.zip'


# 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 = 1000


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[:100])
print('Sample data', data[:250], [reverse_dictionary[i] for i in data[:250]])

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 = 3       # How many words to consider left and right.
num_skips = 2         # How many times to reuse an input to generate a label.
num_sampled = 250      # 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 = 50     # 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]))
```

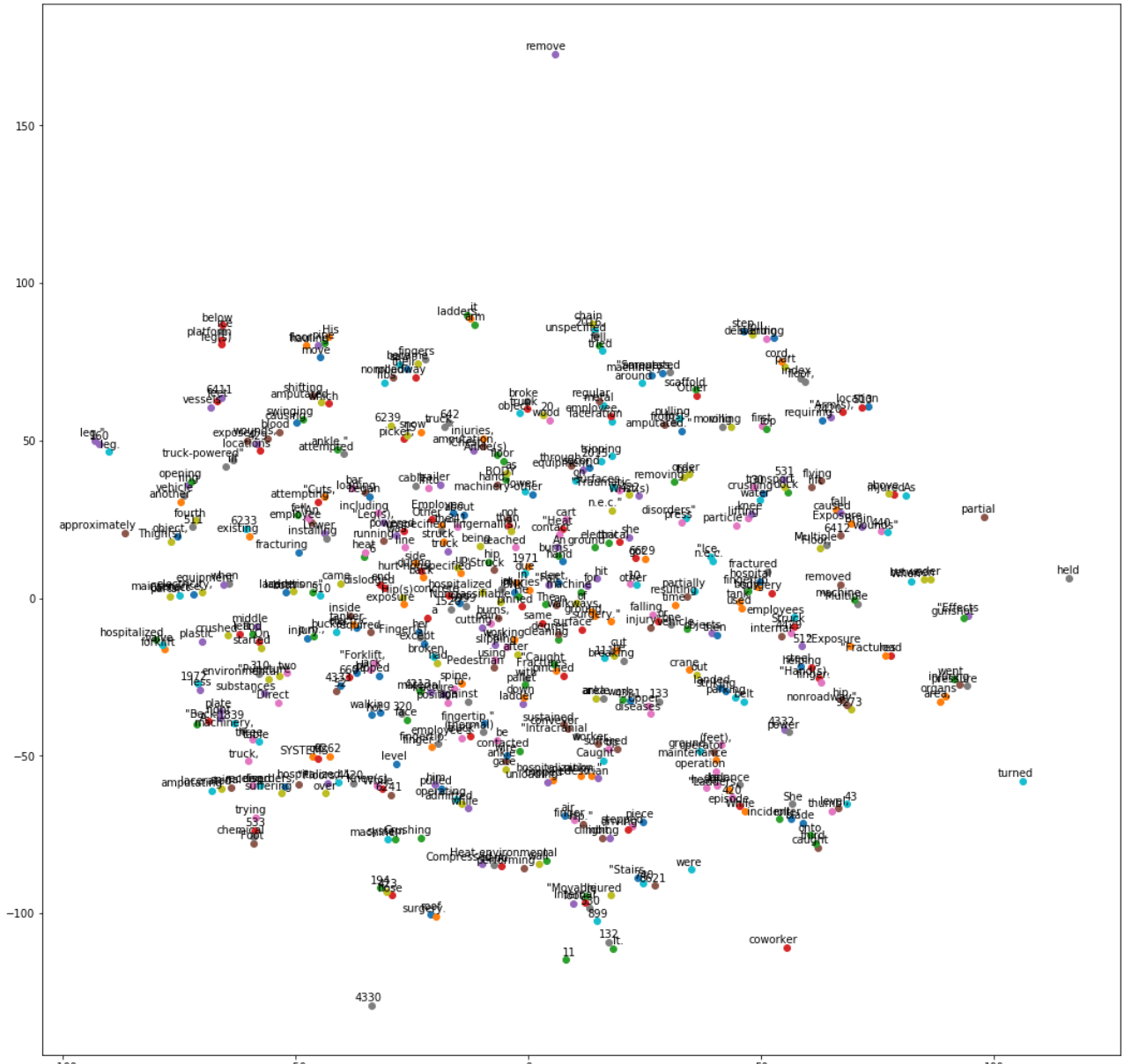
```
144
145   # Compute the average NCE loss for the batch.
146   # tf.nce_loss automatically draws a new sample of the negative labels each
147   # time we evaluate the loss.
148   # Explanation of the meaning of NCE loss:
149   #   http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
150   loss = tf.reduce_mean(
151       tf.nn.nce_loss(weights=nce_weights,
152                      biases=nce_biases,
153                      labels=train_labels,
154                      inputs=embed,
155                      num_sampled=num_sampled,
156                      num_classes=vocabulary_size))
157
158   # Construct the SGD optimizer using a learning rate of 1.0.
159   optimizer = tf.train.GradientDescentOptimizer(1.0).minimize(loss)
160
161   # Compute the cosine similarity between minibatch examples and all embeddings.
162   norm = tf.sqrt(tf.reduce_sum(tf.square(embeddings), 1, keep_dims=True))
163   normalized_embeddings = embeddings / norm
164   valid_embeddings = tf.nn.embedding_lookup(
165       normalized_embeddings, valid_dataset)
166   similarity = tf.matmul(
167       valid_embeddings, normalized_embeddings, transpose_b=True)
168
169   # Add variable initializer.
170   init = tf.global_variables_initializer()
171
172 # Step 5: Begin training.
173 num_steps = 100
174
175 with tf.Session(graph=graph) as session:
176   # We must initialize all variables before we use them.
177   init.run()
178   print('Initialized')
179
180   average_loss = 0
181   for step in xrange(9):
182     batch_inputs, batch_labels = generate_batch(
183         batch_size, num_skips, skip_window)
184     feed_dict = {train_inputs: batch_inputs, train_labels: batch_labels}
185
186     # We perform one update step by evaluating the optimizer op (including it
187     # in the list of returned values for session.run()
188     _, loss_val = session.run([optimizer, loss], feed_dict=feed_dict)
189     average_loss += loss_val
190
191     if step % 2000 == 0:
192       if step > 0:
193         average_loss /= 2000
194       # The average loss is an estimate of the loss over the last 2000 batches.
195       print('Average loss at step ', step, ': ', average_loss)
196       average_loss = 0
197
198     # Note that this is expensive (~20% slowdown if computed every 500 steps)
199     if step % 10000 == 0:
200       sim = similarity.eval()
201       for i in xrange(valid_size):
202         valid_word = reverse_dictionary[valid_examples[i]]
203         top_k = 250  # number of nearest neighbors
204         nearest = (-sim[i, :]).argsort()[1:top_k + 1]
205         log_str = 'Nearest to %s:' % valid_word
206         for k in xrange(top_k):
207           close_word = reverse_dictionary[nearest[k]]
208           log_str = '%s %s,' % (log_str, close_word)
209         print(log_str)
210   final_embeddings = normalized_embeddings.eval()
211
212 # Step 6: Visualize the embeddings.
213
214
215 # 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=(20, 20))  # 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=70, 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)
```

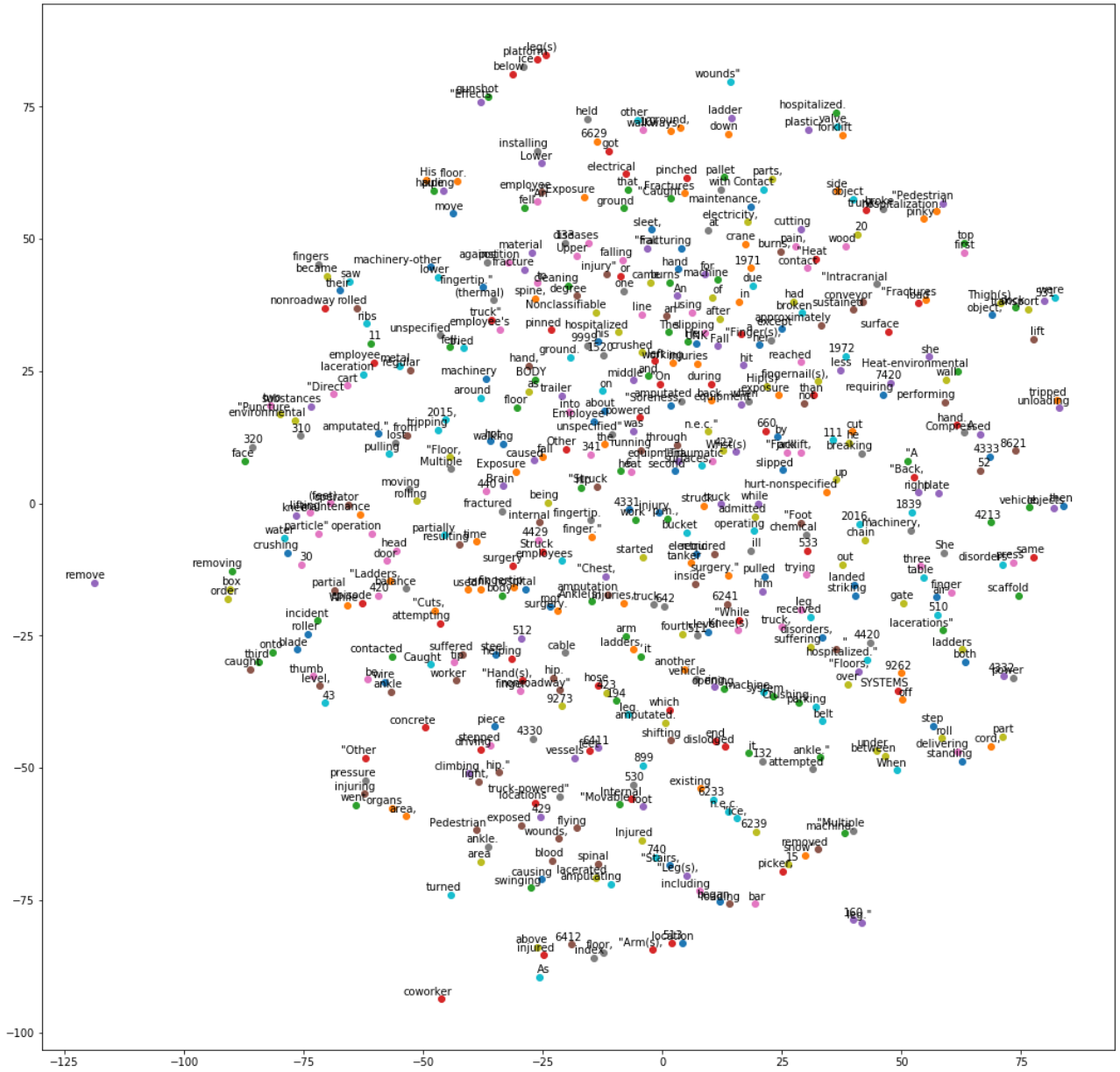
```
s', 28123), ('employee', 26864), ('to', 23731), ('unspecified"', 14143), ('of', 14116), ('o
n', 13730), ('his', 11784), ('in', 10660), ('The', 10541), ('or', 9318), ('An', 8071), ('an',
7920), ('"An', 7834), ('when', 7648), ('n.e.c."', 6844), ('111', 6734), ('by', 6363), ('Fract
ures', 6199), ('fell', 5927), ('left', 5901), ('right', 5804), ('from', 4955), ("employee's",
4423), ('with', 4201), ('he', 4124), ('lower', 4117), ('level', 3983), ('fall', 3754), ('He',
3749), ('feet', 3706), ('while', 3673), ('caught', 3393), ('finger', 3371), ('at', 3312), ('b
etween', 3067), ('object', 3067), ('hand', 2923), ('struck', 2754), ('6', 2720), ('same', 268
1), ('equipment', 2453), ('suffered', 2416), ('into', 2403), ('vehicle', 2336), ('hospitalize
d', 2293), ('due', 2259), ('her', 2252), ('injuries', 2205), ('that', 2181), ('"On', 2161),
('for', 2144), ('injury"', 2143), ('pain,', 2118), ('9999', 2105), ('Nonclassifiable', 2105),
('1972', 2104), ('"Soreness,', 2104), ('hurt-nonspecified', 2104), ('machine', 2090), ('Othe
r', 2033), ('Fall', 1971), ('level,', 1966), ('10', 1954), ('amputated', 1933), ('over', 192
2), ('other', 1913), ('slipped', 1913), ('off', 1901), ('it', 1868), ('than', 1848), ('truc
k', 1759), ('approximately', 1717), ('heat', 1681), ('index', 1679), ('up', 1661), ('were', 1
625), ('except', 1623), ('back', 1600), ('body', 1586), ('injured', 1578), ('middle', 1554),
('"Struck', 1548), ('using', 1525), ('fractured', 1489), ('parts,', 1485), ('broken', 1463),
('causing', 1445), ('working', 1436), ('ground', 1428), ('"Other', 1415), ('ladder', 1412),
('resulting', 1410), ('slipping', 1392), ('"Multiple', 1390), ('out', 1386), ('operating', 13
86)]
```

In [23]:

```
%matplotlib inline
def plot_with_labels(low_dim_embs, labels):
  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=(10, 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=5, 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,)

except ImportError as ex:
  print('Please install sklearn, matplotlib, and scipy to show embeddings.')
  print(ex)
```

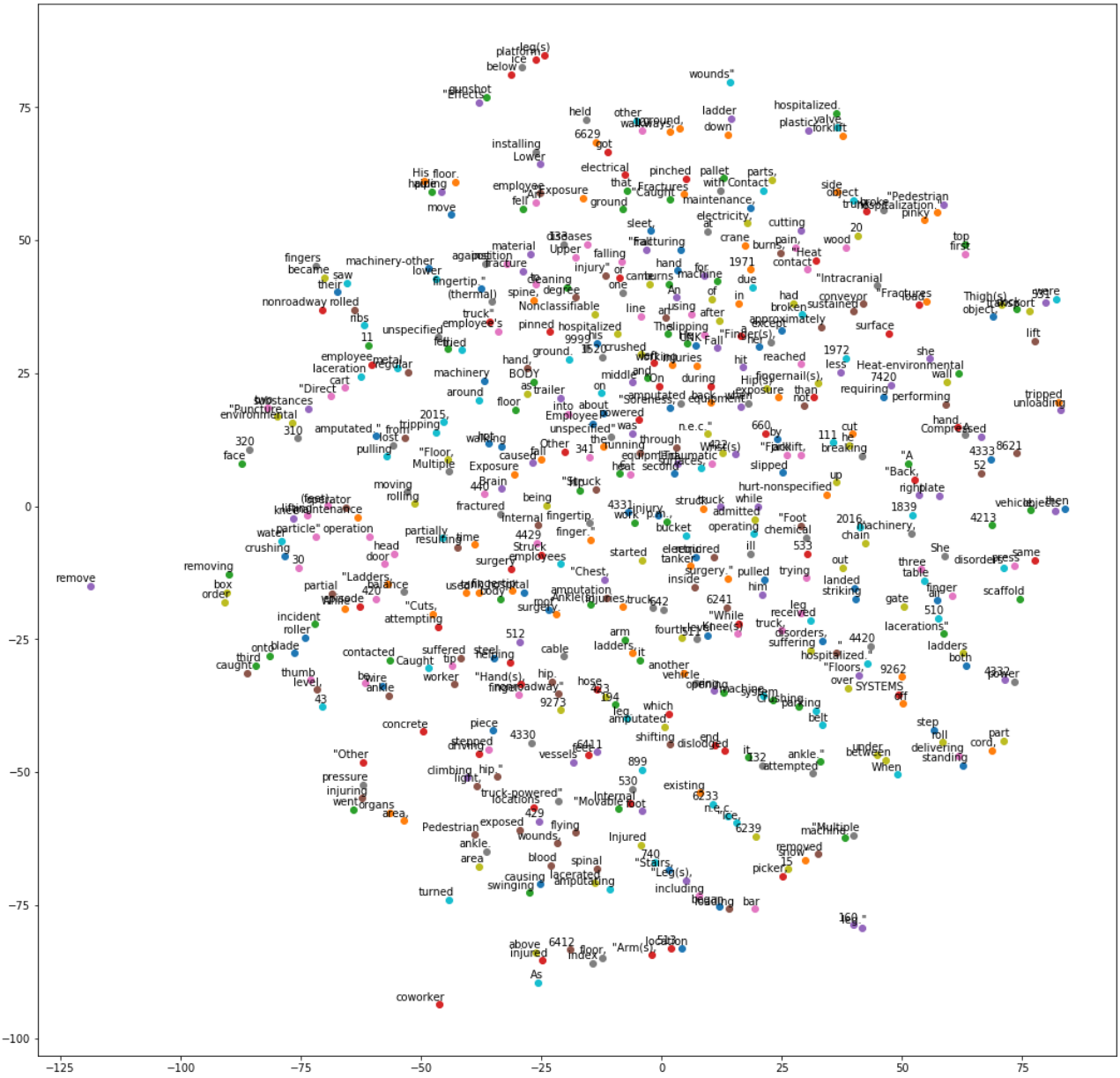

remove
coworker
turned
held
partial
were
150
100
50
0
−50
−100
−100
−50
0
50
100

In [24]:
```
%matplotlib inline
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

tsne = TSNE(perplexity=15, 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)
```



In [25]:
```
%matplotlib inline
def plot_with_labels(low_dim_embs, labels):
  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')
plot_with_labels(low_dim_embs, labels)
plt.savefig(filename)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-25-ec0cd2ef35d7> in <module>()
     13                 va='bottom')
     14 plot_with_labels(low_dim_embs, labels)
---> 15 plt.savefig(filename)

C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\pyplot.py in savefig(*args, **kwargs)
    699 def savefig(*args, **kwargs):
    700     fig = gcf()
--> 701     res = fig.savefig(*args, **kwargs)
    702     fig.canvas.draw_idle()   # need this if 'transparent=True' to reset colors
    703     return res

C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\figure.py in savefig(self, fname, **kwarg
s)
   1832             self.set_frameon(frameon)
   1833
-> 1834         self.canvas.print_figure(fname, **kwargs)
   1835
   1836         if frameon:

C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\backend_bases.py in print_figure(self, fi
lename, dpi, facecolor, edgecolor, orientation, format, **kwargs)
   2168
   2169         # get canvas object and print method for format
-> 2170         canvas = self._get_output_canvas(format)
   2171         print_method = getattr(canvas, 'print_%s' % format)
   2172

C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\backend_bases.py in _get_output_canvas(se
lf, format)
   2111         raise ValueError('Format "%s" is not supported.\n'
   2112                          'Supported formats: '
-> 2113                          '%s.' % (format, ', '.join(formats)))
   2114
   2115     def print_figure(self, filename, dpi=None, facecolor=None, edgecolor=None,

ValueError: Format "zip" is not supported.
Supported formats: eps, jpeg, jpg, pdf, pgf, png, ps, raw, rgba, svg, svgz, tif, tiff.
```



In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```