Mitigating_Unwanted_Biases_in_Word_Embeddings_with_Adversaria

April 24, 2019


```
In [0]: # This library is free software; you can redistribute it and/or
        # modify it under the terms of the GNU Lesser General Public
        # License as published by the Free Software Foundation; either
        # version 2.1 of the License, or (at your option) any later version.
        #
        # This library is distributed in the hope that it will be useful,
        # but WITHOUT ANY WARRANTY; without even the implied warranty of
        # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
        # Lesser General Public License for more details.
        #
        # You should have received a copy of the GNU Lesser General Public
        # License along with this library; if not, write to the Free Software
        # Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA
```

1 Mitigating Unwanted Biases with Adversarial Learning

Authors: Andrew Zaldivar, Ben Hutchinson, Blake Lemoine, Brian Zhang, Margaret Mitchell

---

1.1 Summary of this Notebook

This notebook is a guide to the paper (archiv)

> Brian Zhang, Blake Lemoine and Margaret Mitchell. Mitigating Unwanted Biases with Adversarial Learning. AAAI Conference on AI, Ethics and Society, 2018.

1.2 Intro statement of problem

Embeddings are a powerful mechanism for projecting a discrete variable (e.g. words, locales, urls) into a multi-dimensional real valued space. Several strong methods have been developed for learning embeddings. One example is the Skipgram algorithm. In that algorithm the surrounding context is used to predict the presence of a word. Unfortunately, much real world textual data has subtle bias that machine learning algorithms will implicitly include in the embeddings created from that data. This bias can be illustrated by performing a word analogy task using the learned embeddings.

It is worth noting that the usages of terms like *fair* and *bias* are used in this notebook in the context of a particular definition of fairness sometimes referred to as "Demographic Parity" or "Equality of Outcomes" (Hardt et. al 2016). This definition of fairness effectively says that any relationship at all between a variable of interest and a *protected variable* is an example of unwanted bias. Other definitions of fairness such as "Equality of Odds" can be employed when there is believed to be some form of proper relationship between the variable of interest and the protected variable. However, all uses of *fair* and *bias* here should be interpreted in the context of "Demographic Parity".

First, we'll import all the packages that we'll need.

```
In [2]: !pip install -U gensim~=3.2.0
        import gensim
        import gzip
        import numpy as np
        import os
        import pandas as pd
        !pip install --upgrade-strategy=only-if-needed tensorflow~=1.6.0rc0
        import tensorflow as tf
```

```
Requirement already up-to-date: gensim~=3.2.0 in /usr/local/lib/python2.7/dist-packages (3.2.0)
Requirement already satisfied, skipping upgrade: smart-open>=1.2.1 in /usr/local/lib/python2.7/
Requirement already satisfied, skipping upgrade: six>=1.5.0 in /usr/local/lib/python2.7/dist-pa
Requirement already satisfied, skipping upgrade: scipy>=0.18.1 in /usr/local/lib/python2.7/dist
Requirement already satisfied, skipping upgrade: numpy>=1.11.3 in /usr/local/lib/python2.7/dist
Requirement already satisfied, skipping upgrade: boto>=2.32 in /usr/local/lib/python2.7/dist-pa
Requirement already satisfied, skipping upgrade: bz2file in /usr/local/lib/python2.7/dist-packa
Requirement already satisfied, skipping upgrade: requests in /usr/local/lib/python2.7/dist-pac
Requirement already satisfied, skipping upgrade: boto3 in /usr/local/lib/python2.7/dist-package
Requirement already satisfied, skipping upgrade: urllib3<1.25,>=1.21.1 in /usr/local/lib/python
Requirement already satisfied, skipping upgrade: certifi>=2017.4.17 in /usr/local/lib/python2.7
Requirement already satisfied, skipping upgrade: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python
Requirement already satisfied, skipping upgrade: idna<2.9,>=2.5 in /usr/local/lib/python2.7/dis
Requirement already satisfied, skipping upgrade: jmespath<1.0.0,>=0.7.1 in /usr/local/lib/pytho
Requirement already satisfied, skipping upgrade: botocore<1.13.0,>=1.12.134 in /usr/local/lib/p
Requirement already satisfied, skipping upgrade: s3transfer<0.3.0,>=0.2.0 in /usr/local/lib/pyt
Requirement already satisfied, skipping upgrade: docutils>=0.10 in /usr/local/lib/python2.7/dis
Requirement already satisfied, skipping upgrade: python-dateutil<3.0.0,>=2.1; python_version >=
Requirement already satisfied, skipping upgrade: futures<4.0.0,>=2.2.0; python_version == "2.6"
```

```
/usr/local/lib/python2.7/dist-packages/smart_open/ssh.py:34: UserWarning: paramiko missing, ope
  warnings.warn('paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install p


Requirement already satisfied: tensorflow~=1.6.0rc0 in /usr/local/lib/python2.7/dist-packages
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python2.7/dist-packages (from te
Requirement already satisfied: mock>=2.0.0 in /usr/local/lib/python2.7/dist-packages (from tens
Requirement already satisfied: tensorboard<1.7.0,>=1.6.0 in /usr/local/lib/python2.7/dist-packa
Requirement already satisfied: protobuf>=3.4.0 in /usr/local/lib/python2.7/dist-packages (from
Requirement already satisfied: gast>=0.2.0 in /usr/local/lib/python2.7/dist-packages (from tens
Requirement already satisfied: wheel in /usr/local/lib/python2.7/dist-packages (from tensorflow
Requirement already satisfied: absl-py>=0.1.6 in /usr/local/lib/python2.7/dist-packages (from t
Requirement already satisfied: backports.weakref>=1.0rc1 in /usr/local/lib/python2.7/dist-packa
Requirement already satisfied: enum34>=1.1.6 in /usr/local/lib/python2.7/dist-packages (from te
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python2.7/dist-packages (from tens
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python2.7/dist-packages (from te
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python2.7/dist-packages (from
Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python2.7/dist-packages (from ten
Requirement already satisfied: futures>=2.2.0 in /usr/local/lib/python2.7/dist-packages (from g
Requirement already satisfied: funcsigs>=1; python_version < "3.3" in /usr/local/lib/python2.7/
Requirement already satisfied: pbr>=0.11 in /usr/local/lib/python2.7/dist-packages (from mock>=
Requirement already satisfied: bleach==1.5.0 in /usr/local/lib/python2.7/dist-packages (from te
Requirement already satisfied: html5lib==0.9999999 in /usr/local/lib/python2.7/dist-packages (
Requirement already satisfied: werkzeug>=0.11.10 in /usr/local/lib/python2.7/dist-packages (fro
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python2.7/dist-packages (from
Requirement already satisfied: setuptools in /usr/local/lib/python2.7/dist-packages (from proto
```

First let's copy the required data files from Google Cloud Storage to a local directory.

To do this we'll first need to authenticate to Google Cloud Storage. Executing the following cell will generate a link which you'll need to follow to get a verification code.

```
In [0]: from google.colab import auth
        auth.authenticate_user()
```

Now we'll sync the data for the colab to a tmp directory from Google Cloud Storage.

```
In [4]: project_id = 'mledu-fairness'
        !gcloud config set project {project_id}

        gcs_bucket_name = 'mledu-fairness/colabs/debias_word_embeddings'
        local_dir_name = '/tmp/debias_word_embeddings'
        if not os.path.exists(local_dir_name):
          print "creating dir %s" % local_dir_name
          !mkdir {local_dir_name}

        !gsutil rsync gs://{gcs_bucket_name} {local_dir_name}

        !ls -al {local_dir_name}
```

```
        WORD2VEC_FILE = os.path.join(local_dir_name, "GoogleNews-vectors-negative300.bin.gz")
        ANALOGIES_FILE = os.path.join(local_dir_name, "questions-words.txt")

Updated property [core/project].
Building synchronization state...
Starting synchronization...
total 1609048
drwxr-xr-x 2 root root       4096 Apr 24 16:55 .
drwxrwxrwt 1 root root       4096 Apr 24 16:57 ..
-rw-r--r-- 1 root root 1647046227 Jan 26  2018 GoogleNews-vectors-negative300.bin.gz
-rw-r--r-- 1 root root     603955 Jan 18  2018 questions-words.txt


In [0]: def load_word2vec_format(f, max_num_words=None):
          """Loads word2vec data from a file handle.

          Similar to gensim.models.keyedvectors.KeyedVectors.load_word2vec_format
          but takes a file handle as input rather than a filename. This lets us use
          GFile. Also only accepts binary files.

          Args:
            f: file handle
            max_num_words: number of words to load. If None, load all.

          Returns:
            Word2vec data as keyedvectors.EuclideanKeyedVectors.
          """
          header = f.readline()
          vocab_size, vector_size = (
              int(x) for x in header.rstrip().split())  # throws for invalid file format
          print "vector_size =  %d" % vector_size
          result = gensim.models.keyedvectors.EuclideanKeyedVectors()
          num_words = 0
          result.vector_size = vector_size
          result.syn0 = np.zeros((vocab_size, vector_size), dtype=np.float32)

          def add_word(word, weights):
            word_id = len(result.vocab)
            if word in result.vocab:
              print("duplicate word '%s', ignoring all but first", word)
              return
            result.vocab[word] = gensim.models.keyedvectors.Vocab(
                index=word_id, count=vocab_size - word_id)
            result.syn0[word_id] = weights
            result.index2word.append(word)

          if max_num_words and max_num_words < vocab_size:
```

```
            num_embeddings = max_num_words
          else:
            num_embeddings = vocab_size
          print "Loading %d embeddings" % num_embeddings

          binary_len = np.dtype(np.float32).itemsize * vector_size
          for _ in xrange(vocab_size):
            # mixed text and binary: read text first, then binary
            word = []
            while True:
              ch = f.read(1)
              if ch == b' ':
                break
              if ch == b'':
                raise EOFError("unexpected end of input; is count incorrect or file otherwise c
              if ch != b'\n':  # ignore newlines in front of words (some binary files have)
                word.append(ch)
            word = gensim.utils.to_unicode(b''.join(word), encoding='utf-8', errors='strict')
            weights = np.frombuffer(f.read(binary_len), dtype=np.float32)
            add_word(word, weights)
            num_words = num_words + 1
            if max_num_words and num_words == max_num_words:
              break
          if result.syn0.shape[0] != len(result.vocab):
            print(
                "duplicate words detected, shrinking matrix size from %i to %i",
                result.syn0.shape[0], len(result.vocab))
          result.syn0 = np.ascontiguousarray(result.syn0[:len(result.vocab)])
          assert (len(result.vocab), vector_size) == result.syn0.shape

          print("loaded %s matrix", result.syn0.shape)
          return result
```

In [6]:
```
%%time
# Initialize the embeddings client if this hasn't been done yet.
# For the efficiency of this notebook we just load the first 2M words, and don't
# re-initialize the client if it already exists. You could of course filter the
# word list in other ways.
if not 'client' in vars():
  print "Loading word embeddings from %s" % WORD2VEC_FILE
  with gzip.GzipFile(fileobj=open(WORD2VEC_FILE, 'r')) as f:
    client = load_word2vec_format(f, max_num_words=2000000)
```

```
Loading word embeddings from /tmp/debias_word_embeddings/GoogleNews-vectors-negative300.bin.gz
vector_size =  300
Loading 2000000 embeddings
('duplicate words detected, shrinking matrix size from %i to %i', 3000000, 2000000)
('loaded %s matrix', (2000000, 300))
```

```
CPU times: user 1min 11s, sys: 3.43 s, total: 1min 14s
Wall time: 1min 15s
```

The following blocks load a data file with analogy training examples and displays some of them as examples. By changing the indices selected in the final block you can change which analogies from the training set are being displayed.

```
In [0]: def print_knn(client, v, k):
          print "%d closest neighbors to A-B+C:" % k
          for neighbor, score in client.similar_by_vector(
              v.flatten().astype(float), topn=k):
            print "%s : score=%f" % (neighbor, score)
```

Let's take a look at the analogies that the model generates for *man*:*woman*::*boss*:__. Try changing `"boss"` to `"friend"` to see further examples of problematic analogies.

```
In [8]: # Use a word embedding to compute an analogy
        # Edit the parameters below to get different analogies
        A = "man"
        B = "woman"
        C = "boss"
        NUM_ANALOGIES = 5

        in_arr = []
        for i, word in enumerate((A, B, C)):
          in_arr.append(client.word_vec(word))
        in_arr = np.array([in_arr])

        print_knn(client, -in_arr[0, 0, :] + in_arr[0, 1, :] + in_arr[0, 2, :],
                  NUM_ANALOGIES)
```

```
5 closest neighbors to A-B+C:
boss : score=0.814025
bosses : score=0.558710
manageress : score=0.499752
coworker : score=0.471055
receptionist : score=0.470017
```

```
In [0]: def load_analogies(filename):
          """Loads analogies.

          Args:
            filename: the file containing the analogies.

          Returns:
            A list containing the analogies.
          """
```

```
    analogies = []
    with open(filename, "r") as fast_file:
      for line in fast_file:
        line = line.strip()
        # in the analogy file, comments start with :
        if line[0] == ":":
          continue
        words = line.split()
        # there are no misformatted lines in the analogy file, so this should
        # only happen once we're done reading all analogies.
        if len(words) != 4:
          print "Invalid line: %s" % line
          continue
        analogies.append(words)
    print "loaded %d analogies" % len(analogies)
    return analogies

In [10]: analogies = load_analogies(ANALOGIES_FILE)
         print "\n".join("%s is to %s as %s is to %s" % tuple(x) for x in analogies[:5])

loaded 19544 analogies
Athens is to Greece as Baghdad is to Iraq
Athens is to Greece as Bangkok is to Thailand
Athens is to Greece as Beijing is to China
Athens is to Greece as Berlin is to Germany
Athens is to Greece as Bern is to Switzerland
```

1.3 Adversarial Networks for Bias Mitigation

The method presented here for removing some of the bias from embeddings is based on the idea that those embeddings are intended to be used to predict some outcome Y based on an input X but that outcome should, in a fair world, be completely unrelated to some protected variable Z. If that were the case then knowing Y would not help you predict Z any better than chance. This principle can be directly translated into two networks in series as illustrated below. The first attempts to predict Y using X as input. The second attempts to use the predicted value of Y to predict Z. See Figure 1 of the paper.

However, simply training the weights in W based on $\nabla_W L_1$ and the weights in U based on $\nabla_U L_2$ won't actually achieve an unbiased model. In order to do that you need to incorporate into W's update function the concept that U should be no better than chance at predicting Z. The way that you can achieve that is analogous to how Generative Adversarial Networks (GANs) (Goodfellow et al. 2014) train their generators.

In addition to $\nabla_W L_1$ you incorporate the negation of $\nabla_W L_2$ into W's update function. However, it's possible that $\nabla_W L_1$ is changing W in a way which will improve accuracy by using the biased information you are trying to protect. In order to avoid that you also incorporate a term which removes that component of $\nabla_W L_1$ by projecting it onto $\nabla_W L_2$. Once you've incorporated those two terms, the update function for W becomes:

$\nabla_W L_1 - proj_{(\nabla_W L_2)} \nabla_W L_1 - \nabla_W L_2$

1.3.1 Defining the Protected Variable of Embeddings

The description of how to incorporate adversarial networks into machine learned models above is very generic because the technique is generally applicable for any type of systems which can be described in terms of input X being predictive of Y but potentially containing information about a protected variable Z. So long as you can construct the relevant update functions you can apply this technique. However, that doesn't tell you much about the nature of X, Y and Z. In the case of the word analogies task above, $X = B + C - A$ and $Y = D$. Figuring out what Z should be is a little bit trickier though. For that we can look to a paper by Bulokbasi et. al. where they developed an unsupervised methodology for removing gendered semantics from word embeddings.

The first step is to select pairs of words which are relevant to the type of bias you are trying to remove. In the case of gender you can choose word pairs like "man":"woman" and "boy":girl" which have gender as the only difference in their semantics. Once you have these word pairs you can compute the difference between their embeddings to produce vectors in the embeddings' semantic space which are roughly parallel to the semantics of gender. Performing Principal Components Analysis (PCA) on those vectors then gives you the major components of the semantics of gender as defined by the gendered word pairs provided.

```
In [0]: def _np_normalize(v):
          """Returns the input vector, normalized."""
          return v / np.linalg.norm(v)


        def load_vectors(client, analogies):
          """Loads and returns analogies and embeddings.

          Args:
            client: the client to query.
            analogies: a list of analogies.

          Returns:
            A tuple with:
            - the embedding matrix itself
            - a dictionary mapping from strings to their corresponding indices
              in the embedding matrix
            - the list of words, in the order they are found in the embedding matrix
          """
          words_unfiltered = set()
          for analogy in analogies:
            words_unfiltered.update(analogy)
          print "found %d unique words" % len(words_unfiltered)

          vecs = []
          words = []
          index_map = {}
          for word in words_unfiltered:
            try:
              vecs.append(_np_normalize(client.word_vec(word)))
```

```
            index_map[word] = len(words)
            words.append(word)
          except KeyError:
            print "word not found: %s" % word
        print "words not filtered out: %d" % len(words)

        return np.array(vecs), index_map, words
```

```
In [12]: embed, indices, words = load_vectors(client, analogies)

        embed_dim = len(embed[0].flatten())
        print "word embedding dimension: %d" % embed_dim
```

```
found 905 unique words
words not filtered out: 905
word embedding dimension: 300
```

```
In [0]: def find_gender_direction(embed,
                                 indices):
        """Finds and returns a 'gender direction'."""
        pairs = [
            ("woman", "man"),
            ("her", "his"),
            ("she", "he"),
            ("aunt", "uncle"),
            ("niece", "nephew"),
            ("daughters", "sons"),
            ("mother", "father"),
            ("daughter", "son"),
            ("granddaughter", "grandson"),
            ("girl", "boy"),
            ("stepdaughter", "stepson"),
            ("mom", "dad"),
        ]
        m = []
        for wf, wm in pairs:
          m.append(embed[indices[wf]] - embed[indices[wm]])
        m = np.array(m)

        # the next three lines are just a PCA.
        m = np.cov(np.array(m).T)
        evals, evecs = np.linalg.eig(m)
        return _np_normalize(np.real(evecs[:, np.argmax(evals)]))
```

```
In [14]: # Using the embeddings, find the gender vector.
        gender_direction = find_gender_direction(embed, indices)
        print "gender direction: %s" % str(gender_direction.flatten())
```

```
gender direction: [-7.94102556e-02 -8.62907447e-02 -9.36046086e-02 -6.82094699e-02
  2.10405600e-02  9.09854549e-03  3.69967887e-02  6.76862493e-03
 -1.35619514e-01  3.62402266e-02  6.62573091e-02 -9.91207519e-04
  4.59588196e-02 -6.13280986e-02 -2.14388384e-03 -1.15234663e-02
 -7.85432247e-02 -6.24935300e-02 -8.79221813e-02 -1.73189498e-02
  1.72767315e-02  1.72292006e-03 -3.53863093e-02  5.24758140e-03
  1.45531278e-02  7.97479425e-03 -3.40466363e-02 -2.83068019e-02
 -1.02746018e-01 -1.78703381e-01 -1.88934471e-02 -7.23408473e-02
 -1.06853722e-01 -4.48557103e-02 -5.89368281e-03 -7.94341237e-02
  1.17833895e-03 -5.86551743e-02  3.10732848e-02 -3.12044830e-03
 -4.15907969e-02 -7.17691663e-03 -2.56242105e-02 -3.41547309e-02
 -6.38683437e-02  7.83023380e-02  1.41320146e-03 -1.92862106e-02
  4.27428205e-02  6.23987501e-03  2.58495545e-02 -2.53252181e-02
 -6.58270803e-03  6.81094699e-02  2.29536006e-02 -3.01465541e-02
 -8.76620455e-03 -3.48142318e-03 -1.03283556e-02 -4.43229749e-02
  4.82072717e-02 -1.38947109e-01 -5.49295845e-02  1.42456083e-01
 -3.82443506e-02  5.43604495e-02  6.42165093e-02  1.80136131e-01
 -2.95630757e-02 -7.22149244e-02  1.53964050e-02  3.28869800e-02
  4.24435462e-02  2.30966085e-02  1.12612003e-02  4.36423822e-02
 -7.29707357e-04 -3.64472450e-02  7.50969536e-02  7.15724467e-02
 -2.49016159e-02 -3.88707157e-02  2.47988546e-02  8.84891062e-03
 -1.57366197e-01  5.67419234e-02 -5.13816362e-02  1.39329337e-02
 -2.49230367e-03 -8.79131791e-03 -6.95209856e-02 -9.16509235e-02
 -5.41995965e-02  1.73404339e-02 -6.23635240e-03  1.97633141e-02
 -1.39783648e-02 -2.09286823e-02  5.42969581e-02  5.43749286e-02
 -2.15415313e-02 -2.41145593e-03  4.85206319e-02 -5.10697951e-02
 -1.34003908e-02  2.13284548e-02 -1.12171602e-01 -3.89264077e-02
  7.07023362e-02  6.22944837e-03 -2.23500522e-02 -5.02853247e-02
  9.33401918e-02  2.40351194e-02 -4.21102256e-02  5.20160292e-02
  4.75287371e-02 -8.11995185e-02 -7.38389538e-03  2.29741450e-02
  7.62025009e-03  5.25651699e-02  1.24644176e-01 -3.22192007e-02
  9.16384819e-02  1.89994690e-01  7.22935579e-03  5.25197973e-02
  8.30303086e-02  6.13005472e-02  4.64472574e-02  1.97908008e-02
 -5.19907059e-02  1.51321854e-01  2.47586642e-02  7.32033143e-02
  2.25787357e-02 -5.42248469e-02 -6.66883855e-02 -1.05413345e-01
 -2.27129787e-02 -8.88976863e-02 -3.63838620e-02 -8.98777237e-02
  5.52357322e-03  3.68328109e-02 -6.02720613e-02  7.25324488e-02
 -6.96779923e-03 -1.40505525e-01  2.02638063e-03 -3.35120036e-04
  7.33190982e-02 -6.88252971e-02  4.61441135e-02  7.68183901e-02
 -3.30161505e-02  8.79513077e-02  7.73010673e-02  9.25715579e-03
 -9.04122538e-02  7.52339842e-03 -5.72907134e-02  4.28152602e-02
  2.86397606e-02 -5.62480396e-02 -3.87299617e-02 -3.87058619e-02
  5.85371538e-02  9.43237537e-02  2.71958769e-02  5.25680701e-02
  2.08041380e-02 -3.83978501e-02 -1.10333866e-01 -1.88310924e-02
  1.85601924e-02  1.11566914e-01 -3.58018442e-04  2.06100921e-02
  8.06141199e-03 -1.22897665e-02 -2.16353311e-02  8.96372961e-03
  1.00744761e-02 -2.87074612e-02  1.11815260e-02  5.87513547e-02
 -5.30042040e-02  1.27520561e-01  1.25319079e-02 -1.03226821e-01
```

```
-1.97484407e-02 -3.42320863e-02  1.05197274e-01  1.76135002e-02
 2.80467006e-02 -8.31193905e-03  1.10601081e-02 -1.40255927e-02
-4.74642660e-02 -2.75716115e-03 -1.01467816e-01 -5.44390020e-02
-8.95335277e-02 -1.65279682e-02 -1.12849845e-01  3.43099446e-02
 1.07627595e-02 -1.95155440e-02  8.49218426e-03  7.18273491e-02
 9.43337430e-02  5.93970694e-02  4.42150488e-02  3.93104455e-03
 1.55634159e-02  1.47405106e-02  9.86230904e-02 -3.51293172e-02
-5.48265317e-03  4.39100322e-02  1.00173280e-01  4.73382845e-02
-8.75985652e-02 -8.01811478e-02 -2.84562342e-02 -1.62217727e-02
-5.23065164e-02  2.49845262e-02 -3.01654996e-02  1.45479843e-02
-1.35092122e-02 -1.05868228e-02  1.30148387e-04  1.41828245e-03
 9.64396967e-02 -1.37944939e-03  1.28795558e-02  5.14631073e-02
 4.12590524e-02 -1.05768228e-01  4.51128868e-02  7.65950631e-02
 8.35771929e-02  5.04364947e-02 -7.37407299e-02 -1.27911956e-02
-7.96467866e-02  1.42361577e-02  2.28048101e-02 -2.45567372e-02
 1.51617365e-02 -8.14192396e-02 -2.99888041e-02 -9.38109909e-03
 9.73554448e-03  2.77240676e-02 -1.44856280e-02 -2.15535602e-02
 2.22796961e-02 -2.44437577e-02 -1.73757435e-04 -5.92882308e-02
-6.75396381e-02 -6.88861185e-03  1.79634043e-02  5.94912151e-02
 1.34743549e-02  3.44577542e-02 -5.31342611e-02  1.22593336e-02
 6.63454886e-03  1.72750748e-01  8.17975838e-02  7.98060175e-03
 3.48464502e-02  3.15494960e-02  3.00879598e-02  1.26706401e-02
-5.09294800e-02 -1.49508612e-02  5.45480741e-03  5.96511080e-02
 8.97118066e-03  2.17049865e-02 -5.54851264e-02  4.33567631e-03
 4.53374907e-02  9.27058295e-02 -2.28239338e-02  3.36064502e-02
-5.63768244e-02 -4.46595961e-02  2.17583523e-03 -3.09787628e-02
-1.12975804e-02  9.51682266e-02  3.74056898e-02  1.14501412e-01]
```

Once you have the first principal component of the embedding differences, you can start projecting the embeddings of words onto it. That projection is roughly the degree to which a word is relevant to the latent protected variable defined by the first principle component of the word pairs given. This projection can then be taken as the protected variable Z which the adversary is attempting to predict on the basis of the predicted value of Y. The code below illustrates how to construct a function which computes Z from X in this way.

Try editing the WORD param in the next cell to see the projection of other words onto the gender direction.

```
In [15]: WORD = "she"

         word_vec = client.word_vec(WORD)
         print word_vec.dot(gender_direction)
```

```
0.706431491878112
```

Let's now look at the words with the largest *negative* projection onto the gender dimension.

```
In [16]: words = set()
         for a in analogies:
```

```
            words.update(a)

        df = pd.DataFrame(data={"word": list(words)})
        df["gender_score"] = df["word"].map(
            lambda w: client.word_vec(w).dot(gender_direction))
        df.sort_values(by="gender_score", inplace=True)
        print df.head(10)
```

```
             word  gender_score
188           his     -0.660903
843            he     -0.584860
284  unimpressive     -0.544877
615      Anaheim      -0.503144
535        Libya      -0.486664
242      playing      -0.472259
754         play      -0.459254
362     sharpest      -0.455905
830      Detroit      -0.454509
303       calmly      -0.448576
```

Let's now look at the words with the largest *positive* projection onto the gender dimension.

```
In [17]: df.sort_values(by="gender_score", inplace=True, ascending=False)
         print df.head(10)
```

```
            word  gender_score
520      husband      0.950914
462  policewoman      0.816518
28         women      0.758530
733          mom      0.732802
335     princess      0.719736
517          she      0.706431
355      sisters      0.699692
429      stepson      0.686814
813          her      0.683995
470        queen      0.682354
```

1.3.2 Training the model

Training adversarial networks is hard. They are touchy, and if touched the wrong way, they blow up VERY quickly. One must be very careful to train both models slowly enough, so that the parameters in the models do not diverge. In practice, this usually entails significantly lowering the step size of both the classifier and the adversary. It is also probably beneficial to initialize the parameters of the adversary to be extremely small, to ensure that the classifier does not overfit against a particular (sub-optimal) adversary (such overfitting can very quickly cause divergence!). It is also possible that if the classifier is too good at hiding the protected variable from the adversary then the adversary will impose updates that diverge in an effort to improve its performance. The

solution to that can sometimes be to actually increase the adversary's learning rate to prevent divergence (something almost unheard of in most learning systems). Below is a system for learning the debiasing model for word embeddings described above.

Inspect the terminal output for the kernel to inspect the performance of the model. Try modifying the hyperparameters at the top to see how that impacts the convergence properties of the system.

```
In [0]: def tf_normalize(x):
          """Returns the input vector, normalized.

          A small number is added to the norm so that this function does not break when
          dealing with the zero vector (e.g. if the weights are zero-initialized).

          Args:
            x: the tensor to normalize
          """
          return x / (tf.norm(x) + np.finfo(np.float32).tiny)


        class AdversarialEmbeddingModel(object):
          """A model for doing adversarial training of embedding models."""

          def __init__(self, client,
                       data_p, embed_dim, projection,
                       projection_dims, pred):
            """Creates a new AdversarialEmbeddingModel.

            Args:
              client: The (possibly biased) embeddings.
              data_p: Placeholder for the data.
              embed_dim: Number of dimensions used in the embeddings.
              projection: The space onto which we are "projecting".
              projection_dims: Number of dimensions of the projection.
              pred: Prediction layer.
            """
            # load the analogy vectors as well as the embeddings
            self.client = client
            self.data_p = data_p
            self.embed_dim = embed_dim
            self.projection = projection
            self.projection_dims = projection_dims
            self.pred = pred

          def nearest_neighbors(self, sess, in_arr,
                                k):
            """Finds the nearest neighbors to a vector.

            Args:
```

```
    sess: Session to use.
    in_arr: Vector to find nearest neighbors to.
    k: Number of nearest neighbors to return
  Returns:
    List of up to k pairs of (word, score).
  """
  v = sess.run(self.pred, feed_dict={self.data_p: in_arr})
  return self.client.similar_by_vector(v.flatten().astype(float), topn=k)

def write_to_file(self, sess, f):
  """Writes a model to disk."""
  np.savetxt(f, sess.run(self.projection))

def read_from_file(self, sess, f):
  """Reads a model from disk."""
  loaded_projection = np.loadtxt(f).reshape(
      [self.embed_dim, self.projection_dims])
  sess.run(self.projection.assign(loaded_projection))

def fit(self,
        sess,
        data,
        data_p,
        labels,
        labels_p,
        protect,
        protect_p,
        gender_direction,
        pred_learning_rate,
        protect_learning_rate,
        protect_loss_weight,
        num_steps,
        batch_size,
        debug_interval=1000):
  """Trains a model.

  Args:
    sess: Session.
    data: Features for the training data.
    data_p: Placeholder for the features for the training data.
    labels: Labels for the training data.
    labels_p: Placeholder for the labels for the training data.
    protect: Protected variables.
    protect_p: Placeholder for the protected variables.
    gender_direction: The vector from find_gender_direction().
    pred_learning_rate: Learning rate for predicting labels.
    protect_learning_rate: Learning rate for protecting variables.
    protect_loss_weight: The constant 'alpha' found in
```

```
        debias_word_embeddings.ipynb.
    num_steps: Number of training steps.
    batch_size: Number of training examples in each step.
    debug_interval: Frequency at which to log performance metrics during
        training.
"""
feed_dict = {
    data_p: data,
    labels_p: labels,
    protect_p: protect,
}
# define the prediction loss
pred_loss = tf.losses.mean_squared_error(labels_p, self.pred)

# compute the prediction of the protected variable.
# The "trainable"/"not trainable" designations are for the predictor. The
# adversary explicitly specifies its own list of weights to train.
protect_weights = tf.get_variable(
    "protect_weights", [self.embed_dim, 1], trainable=False)
protect_pred = tf.matmul(self.pred, protect_weights)
protect_loss = tf.losses.mean_squared_error(protect_p, protect_pred)

pred_opt = tf.train.AdamOptimizer(pred_learning_rate)
protect_opt = tf.train.AdamOptimizer(protect_learning_rate)

protect_grad = {v: g for (g, v) in pred_opt.compute_gradients(protect_loss)}
pred_grad = []

# applies the gradient expression found in the document linked
# at the top of this file.
for (g, v) in pred_opt.compute_gradients(pred_loss):
  unit_protect = tf_normalize(protect_grad[v])
  # the two lines below can be commented out to train without debiasing
  g -= tf.reduce_sum(g * unit_protect) * unit_protect
  g -= protect_loss_weight * protect_grad[v]
  pred_grad.append((g, v))
  pred_min = pred_opt.apply_gradients(pred_grad)

# compute the loss of the protected variable prediction.
protect_min = protect_opt.minimize(protect_loss, var_list=[protect_weights])

sess.run(tf.global_variables_initializer())
sess.run(tf.local_variables_initializer())
step = 0
while step < num_steps:
  # pick samples at random without replacement as a minibatch
  ids = np.random.choice(len(data), batch_size, False)
  data_s, labels_s, protect_s = data[ids], labels[ids], protect[ids]
```

```
        sgd_feed_dict = {
            data_p: data_s,
            labels_p: labels_s,
            protect_p: protect_s,
        }

        if not step % debug_interval:
          metrics = [pred_loss, protect_loss, self.projection]
          metrics_o = sess.run(metrics, feed_dict=feed_dict)
          pred_loss_o, protect_loss_o, proj_o = metrics_o
          # log stats every so often: number of steps that have passed,
          # prediction loss, adversary loss
          print("step: %d; pred_loss_o: %f; protect_loss_o: %f" % (step,
                        pred_loss_o, protect_loss_o))
          for i in range(proj_o.shape[1]):
            print("proj_o: %f; dot(proj_o, gender_direction): %f)" %
                          (np.linalg.norm(proj_o[:, i]),
                          np.dot(proj_o[:, i].flatten(), gender_direction)))
        sess.run([pred_min, protect_min], feed_dict=sgd_feed_dict)
        step += 1

def filter_analogies(analogies,
                     index_map):
  filtered_analogies = []
  for analogy in analogies:
    if filter(index_map.has_key, analogy) != analogy:
      print "at least one word missing for analogy: %s" % analogy
    else:
      filtered_analogies.append(map(index_map.get, analogy))
  return filtered_analogies

def make_data(
    analogies, embed,
    gender_direction):
  """Preps the training data.

  Args:
    analogies: a list of analogies
    embed: the embedding matrix from load_vectors
    gender_direction: the gender direction from find_gender_direction

  Returns:
    Three numpy arrays corresponding respectively to the input, output, and
    protected variables.
  """
  data = []
  labels = []
  protect = []
```

```
    for analogy in analogies:
      # the input is just the word embeddings of the first three words
      data.append(embed[analogy[:3]])
      # the output is just the word embeddings of the last word
      labels.append(embed[analogy[3]])
      # the protected variable is the gender component of the output embedding.
      # the extra pair of [] is so that the array has the right shape after
      # it is converted to a numpy array.
      protect.append([np.dot(embed[analogy[3]], gender_direction)])
    # Convert all three to numpy arrays, and return them.
    return tuple(map(np.array, (data, labels, protect)))
```

Edit the training parameters below to experiment with different training runs.
For example, try increasing the number of training steps to 50k.

In [19]:
```
# Edit the training parameters below to experiment with different training runs.
# For example, try
pred_learning_rate = 2**-16
protect_learning_rate = 2**-16
protect_loss_weight = 1.0
num_steps = 10000
batch_size = 1000

embed_dim = 300
projection_dims = 1


sess = tf.InteractiveSession()
with tf.variable_scope('var_scope', reuse=tf.AUTO_REUSE):
    analogy_indices = filter_analogies(analogies, indices)

    data, labels, protect = make_data(analogy_indices, embed, gender_direction)
    data_p = tf.placeholder(tf.float32, shape=[None, 3, embed_dim], name="data")
    labels_p = tf.placeholder(tf.float32, shape=[None, embed_dim], name="labels")
    protect_p = tf.placeholder(tf.float32, shape=[None, 1], name="protect")

    # projection is the space onto which we are "projecting". By default, this is
    # one-dimensional, but this can be tuned by projection_dims
    projection = tf.get_variable("projection", [embed_dim, projection_dims])

    # build the prediction layer
    # pred is the simple computation of d = -a + b + c for a : b :: c : d
    pred = -data_p[:, 0, :] + data_p[:, 1, :] + data_p[:, 2, :]
    pred -= tf.matmul(tf.matmul(pred, projection), tf.transpose(projection))

    trained_model = AdversarialEmbeddingModel(
        client, data_p, embed_dim, projection, projection_dims, pred)
```

```
            trained_model.fit(sess, data, data_p, labels, labels_p, protect, protect_p, gender
                      pred_learning_rate,
                     protect_learning_rate, protect_loss_weight, num_steps, batch_size)
```

```
step: 0; pred_loss_o: 0.003403; protect_loss_o: 0.013052
proj_o: 1.373836; dot(proj_o, gender_direction): -0.003862)
step: 1000; pred_loss_o: 0.003411; protect_loss_o: 0.010348
proj_o: 1.414057; dot(proj_o, gender_direction): 0.055997)
step: 2000; pred_loss_o: 0.003457; protect_loss_o: 0.010196
proj_o: 1.536013; dot(proj_o, gender_direction): 0.119548)
step: 3000; pred_loss_o: 0.003528; protect_loss_o: 0.010305
proj_o: 1.670961; dot(proj_o, gender_direction): 0.170868)
step: 4000; pred_loss_o: 0.003592; protect_loss_o: 0.009321
proj_o: 1.767516; dot(proj_o, gender_direction): 0.209582)
step: 5000; pred_loss_o: 0.003628; protect_loss_o: 0.007625
proj_o: 1.818539; dot(proj_o, gender_direction): 0.236778)
step: 6000; pred_loss_o: 0.003630; protect_loss_o: 0.005974
proj_o: 1.827884; dot(proj_o, gender_direction): 0.253526)
step: 7000; pred_loss_o: 0.003609; protect_loss_o: 0.004727
proj_o: 1.807212; dot(proj_o, gender_direction): 0.262754)
step: 8000; pred_loss_o: 0.003578; protect_loss_o: 0.003838
proj_o: 1.767977; dot(proj_o, gender_direction): 0.268229)
step: 9000; pred_loss_o: 0.003544; protect_loss_o: 0.003183
proj_o: 1.714309; dot(proj_o, gender_direction): 0.273144)
```

1.3.3 Analogy generation using the embeddings with bias reduced by the adversarial model

Let's see how the model that has been trained to mitigate bias performs on the analogy task. As before, change "boss" to "friend" to see how those analogies have changed too.

In [20]:
```
# Parameters
A = "man"
B = "woman"
C = "boss"
NUM_ANALOGIES = 5

# Use a word embedding to compute an analogy
in_arr = []
for i, word in enumerate((A, B, C)):
  in_arr.append(client.word_vec(word))
in_arr = np.array([in_arr])

print_knn(client, sess.run(pred, feed_dict={data_p: in_arr}),
          NUM_ANALOGIES)
```

```
5 closest neighbors to A-B+C:
boss : score=0.738654
bosses : score=0.500845
```

```
exec : score=0.424892
supremo : score=0.423384
manageress : score=0.423364
```

1.4 Conclusion

The method demonstrated here helps to reduce the amount of bias in word embeddings and, although not demonstrated here, generalizes quite well to other domains and tasks. By trying to hide a protected variable from an adversary, a machine learned system can reduce the amount of biased information about that protected variable implicit in the system. In addition to the specific method demonstrated here there are many variations on this theme which can be used to achieve different degrees and types of debiasing. For example, you could debias with respect to more than one principle component of the protected variable by having the adverary predict multiple projections. Many other elaborations on this basic idea are possible and hopefully this relatively simple system can serve as the basis for more complex and sophisticated systems capable of achieving subtle types of bias mitigation in many applications.

In [0]: