Problem 3

Authors: Sid Murching, Suraj Nair, Alex Cui

In [9]:

```
import numpy as np
from P3CHelpers import *
from keras.models import Sequential
import sys
```

3D:

Fill in the generate_traindata and find_most_similar_pairs functions

In [10]:

```
def get_word_repr(word_to_index, word):
    """
    Returns one-hot-encoded feature representation of the specified word given
    a dictionary mapping words to their one-hot-encoded index.

    Arguments:
        word_to_index: Dictionary mapping words to their corresponding index
                       in a one-hot-encoded representation of our corpus.

        word:          String containing word whose feature representation we wish
to compute.

    Returns:
        feature_representation:     Feature representation of the passed-in word.
    """
    unique_words = word_to_index.keys()
    # Return a vector that's zero everywhere besides the index corresponding to <w
ord>
    feature_representation = np.zeros(len(unique_words))
    feature_representation[word_to_index[word]] = 1
    return feature_representation

def generate_traindata(word_list, word_to_index, window_size=4):
    """
    Generates training data for Skipgram model.

    Arguments:
        word_list:     Sequential list of words (strings).
        word_to_index: Dictionary mapping words to their corresponding index
                       in a one-hot-encoded representation of our corpus.

        window_size:   Size of Skipgram window.
                       (use the default value when running your code).

    Returns:
        (trainX, trainY):     A pair of matrices (trainX, trainY) containing train
ing
                              points (one-hot-encoded vectors representing individ
ual words) and
                              their corresponding labels (also one-hot-encoded vec
tors representing words).

                              For each index i, trainX[i] should correspond to a w
ord in
                              <word_list>, and trainY[i] should correspond to one
of the words within
                              a window of size <window_size> of trainX[i].
    """
    trainX = []
    trainY = []
    numWords = len(word_to_index)
    trainX = []
    trainY = []
    allZeroes = [0 for i in range(numWords)]
    for i in range(len(word_list)):
        for j in range(-window_size, window_size + 1):
            if i + j >= 0 and i + j < len(word_list) and j != 0:
                trainXVector = get_word_repr(word_to_index, word_list[i])
                trainX.append(trainXVector)
                trainYVector = get_word_repr(word_to_index, word_list[i+j])
                trainY.append(trainYVector)

    return (np.array(trainX), np.array(trainY))
```

In [12]:

```python
def find_most_similar_pairs(filename, num_latent_factors):
    """
    Find the most similar pairs from the word embeddings computed from
    a body of text

    Arguments:
        filename:           Text file to read and train embeddings from
        num_latent_factors: The number of latent factors / the size of the embedding
    """
    # Load in a list of words from the specified file; remove non-alphanumeric characters
    # and make all chars lowercase.
    sample_text = load_word_list(filename)

    # Create dictionary mapping unique words to their one-hot-encoded index
    word_to_index = generate_onehot_dict(sample_text)
    # Create training data using default window size
    trainX, trainY = generate_traindata(sample_text, word_to_index)

    # TODO: 1) Create and train model in Keras.

    # vocab_size = number of unique words in our text file. Will be useful when adding layers
    # to your neural network
    vocab_size = len(word_to_index)
    model = Sequential()
    model.add(Dense(num_latent_factors, input_dim=(vocab_size)))
    model.add(Dense(vocab_size))
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
                  metrics=['accuracy'])
    fit = model.fit(trainX, trainY)

    print("Hidden layer shape" + str(model.layers[0].get_weights()[0].shape))
    print("Output layer shape" + str(model.layers[1].get_weights()[0].shape))
    # TODO: 2) Extract weights for hidden layer, set <weights> variable below

    weights = model.layers[0].get_weights()[0]

    # Find and print most similar pairs
    similar_pairs = most_similar_pairs(weights, word_to_index)
    for pair in similar_pairs[:30]:
        print(pair)
```

In [6]:

In [ ]:

3G:

Run the function below and report your results for dr_seuss.txt.

In [ ]:

```python
find_most_similar_pairs('data/dr_seuss.txt', 10)
```

In [ ]: