Word Embedding

Garth Mortensen

November 23, 2018

Matlab has an amazing blog post here about text analysis using word embedding, transfer learning, etc.

Overview

In this assignment, you will exercise word embedding by creating your own short sentences. More specifically,

Questions

1. Create **EXACTLY** 20 sentences. The maximum length of each sentence is FIVE (5). Two sample sentences may look like: “Excellent work”, “Good students work very hard”.

2. Create word embeddings for the vocabulary (i.e. unique words) in your sentences. You can choose one **OR** more word embedding methods from the embedding layer approach, the CBOW approach, the Skip-Gram approach, or the GloVe approach.

Create a WORD document to answer the following questions for EACH of the word embedding method you choose:

1. Describe your embedding approach, architecture, and all the parameters (i.e. epochs, batches) you used.

After trying GloVe and Word Embedding unsuccessfully, I use trainWordEmbedding to train a word2Vec model using Skip Gram or CBoW. Example found (here). Default options were used.

```
dimensionality = 50;
minCounter = 3;
epochs = 100;
modelChoice = 'skipgram'

emb = trainWordEmbedding(tokens, ...
    'Dimension', dimensionality, ...
    'MinCount', minCounter, ...
    'NumEpochs', epochs, ...
    'Model', modelChoice) % skipgram is default, else Model = 'cbow'
```

2. Describe the dimensionality of your word vectors.

My original file.txt was 20 sentences, 5 words each. After importing, a series of cleaning steps were followed:

1) Lowercase. 2) Convert to string. 3) Split into lines. 4) Remove empty lines. 5) Strip (trim) leading and trailing space 6) Replace punctuation with spaces. After these steps, the 20x1 array was tokenized, which turned it into an object class. From it, stop words were deleted, and any words below 2 character or longer than 15. The words were then reduced (normalized) to their stems using the Porter algorithm, However, it is the original tokenized words that are fed into the word embedding function.

3. List the vocabulary in your training set.

words = i, walk, the, s, we, shoes, rain

4. Use the t-SNE method to reduce your word vectors to 2-dimension and plot the 2-D points in a figure. Each point (vector) in the figure MUST be labeled with the original words in your vocabulary. (**NOTE**: you may need to adjust your training sentences so words with certain meaning cluster closer together in your t-SNE figure)

See image in code.

5. List the training time to train your word embedding model.

It took 2.16 second to train using the skip gram method.

Code Start

Table of Contents

Pre-Processing

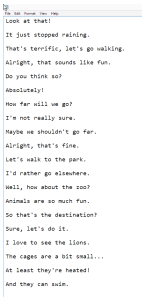
Let's first clear variables and close graphs.

```
% Clear the worksapce
clear all
clc
close all
```

We'll measure how long it takes to run this entire worksheet, including training the RNN, using a stopwatch method. This computer is running an i7-6500 with 16GB DDR4-2400 RAM. You can use CPU-Z to determine your RAM speed. In Matlab, we can also look at our GPU specs.

Load Data

The dataset we're loading appears as:




```
% local dir. This is actually not case sensitive!
% cd 'C:\Users\grm\Google Drive\aStThomas\7ArtificalIntelligence\Assignments\7 Word Embeddings\
% dataOrig = fileread('file.txt'); % textscan(fid, format)

% alternative
% filename = "sonnetsPreprocessed.txt";
% str = extractFileText(filename);
% textData = split(str,newline);
% documents = tokenizedDocument(textData);

%matlab online dir
cd '/MATLAB Drive/Portfolio/AI/Word Embeddings/';
dataOrig = fileread('file.txt')
```

```
dataOrig =
    'Look at that!

     It just stopped raining.

     I hate rain. Let's walk!

     Walking sounds like fun!

     Now that it's not raining.

     I love walking after rain.

     How far will we walk?

     Do we have much time?

     We have 30 minutes.

     A short walk is okay.

     Let's walk to the park.

     I'd rather walk elsewhere.

     Walk to the zoo then?

     Yeah, I love the zoo.

     Where are my shoes?
```

```
I haven't seen your shoes.

I can't go without shoes!

There they are.

Oh no, they're all wet.

Everything's wet after the rain.'
```

Clean Data

Lowercase.

```
datalower = lower(dataOrig);
```

Convert to string.

```
data = string(datalower);
```

Split into lines.

```
data = splitlines(data);
% Think this next line is superfluous
%data(1:end)
```

Remove empty lines.

```
TF = (data == "");
data(TF) = []
```

```
data = 20×1 string array
    "look at that! "
    "it just stopped raining. "
    "i hate rain. let's walk!"
    "walking sounds like fun!"
    "now that it's not raining."
    "i love walking after rain."
    "how far will we walk?"
    "do we have much time?"
    "we have 30 minutes."
    "a short walk is okay."
    "let's walk to the park."
    "i'd rather walk elsewhere."
    "walk to the zoo then?"
    "yeah, i love the zoo."
    "where are my shoes?"
    "i haven't seen your shoes."
    "i can't go without shoes!"
    "there they are."
    "oh no, they're all wet."
    "everything's wet after the rain."
```

Strip (trim) leading and trailing space

```
data = strip(data);
```

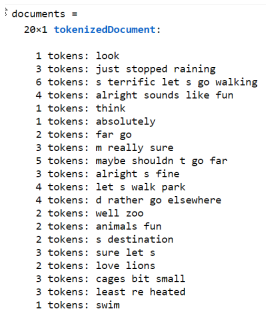
Replace punctuation with spaces

```
punct = [ "." , "?" , "!" , "," , ";" , ":" , "..." , '"' , "'" ];
data = replace(data, punct, " ");
```

issue with contractionary ' so i brought it back

```
% alternative
%cleanTextData = erasePunctuation(data)
```

Tokenize



Effect of tokenization

```
tokens = tokenizedDocument(data)
```

```
tokens =
  20×1 tokenizedDocument:

    3 tokens: look at that
    4 tokens: it just stopped raining
    6 tokens: i hate rain let s walk
    4 tokens: walking sounds like fun
    6 tokens: now that it s not raining
    5 tokens: i love walking after rain
    5 tokens: how far will we walk
    5 tokens: do we have much time
    4 tokens: we have 30 minutes
    5 tokens: a short walk is okay
    6 tokens: let s walk to the park
    5 tokens: i d rather walk elsewhere
    5 tokens: walk to the zoo then
    5 tokens: yeah i love the zoo
    4 tokens: where are my shoes
    6 tokens: i haven t seen your shoes
    6 tokens: i can t go without shoes
    3 tokens: there they are
    6 tokens: oh no they re all wet
    6 tokens: everything s wet after the rain
```

Remove a list of stop words

```
documents = removeWords(tokens, stopWords);
```

Remove short and long words

```
documents = removeShortWords(documents, 2);
documents = removeLongWords(documents, 15);
```

Interesting...why would we do that? for I'm = i m? Here, "I" & "am" are not important. I can't think of any important two letter words...three, plenty.

Stemming

The Porter stemming algorithm (or ‘Porter stemmer’) is a process for removing the commoner morphological and inflexional endings from words in English. Its main use is as part of a term normalisation process that is usually done when setting up Information Retrieval systems.

```
% Normalize the words using the Porter stemmer.
documents_norm = normalizeWords(documents)
```

```
documents_norm =
  20×1 tokenizedDocument:

    1 tokens: look
    3 tokens: just stop rain
    4 tokens: hate rain let walk
    4 tokens: walk sound like fun
    1 tokens: rain
    3 tokens: love walk rain
    2 tokens: far walk
    1 tokens: time
    1 tokens: minut
    3 tokens: short walk okai
    3 tokens: let walk park
    3 tokens: rather walk elsewher
    2 tokens: walk zoo
    3 tokens: yeah love zoo
    1 tokens: shoe
    3 tokens: haven seen shoe
    1 tokens: shoe
    0 tokens:
    1 tokens: wet
    3 tokens: everyth wet rain
```

The magic. Use normalizeWords to reduce words to a root form. To lemmatize English words, or reduce them to their dictionary forms, set 'Style' to 'lemma'. The algo is found here: https://www.cs.odu.edu/~jbollen/IR04/readings/readings5.pdf

Time and place = Cambridge (Oxford), 1980. See mouse cursor in screenshot.

Lemmatize is based on (V)owel (C)onsonant patterns, as described here. Return to this link and understand it, such a tough thing to solve.

```
% CVCV...C
% CVCV...V
% VCVC...C
% VCVC...V
```

Supplementary

Bag of Words (BoW)

Bag-of-words = term-frequency counter.

Record # of words in each tokenzied document.

```
BOW_docs = bagOfWords(documents_norm);
BOW_docs.NumWords;
full(BOW_docs.Counts);
vocab = BOW_docs.Vocabulary % these are vocab workds
```

```
vocab = 1×26 string array
    "look"    "just"    "stop"    "rain"    "hate"    "let"    "walk"    "sound"    "like"    "fun"    "love"    "f
```

```
% Word cloud
figure, wordcloud(BOW_docs)
```


```
ans =
  WordCloudChart with properties:

          WordData: [1×26 string]
          SizeData: [8 5 3 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    MaxDisplayWords: 100

  Show all properties
```

N-grams are n-count word sequences

info

3-grams: ceramics collectables collectibles

4-grams: serve as the incoming

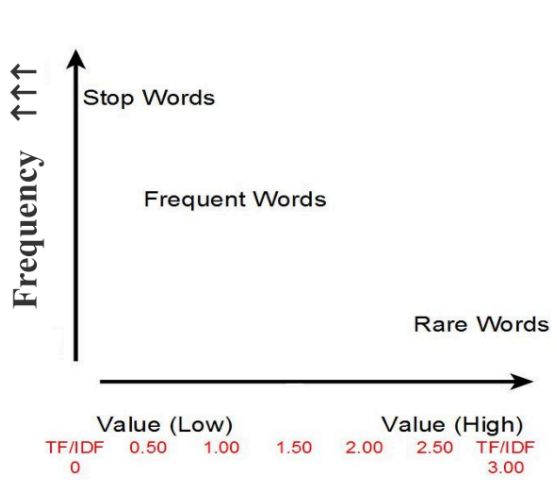
```
% feed it tokenized
bagN = bagOfNgrams(documents_norm, 'NgramLengths', 2);

bagN.Vocabulary
```

```
ans = 1×23 string array
    "just"    "stop"    "hate"    "rain"    "let"    "walk"    "sound"    "like"    "love"    "far"    "short"    "
```

TF/IDF - Term Frequency – Inverse Document Frequency

Source example



The equation for TF/IDF is:

$$weight_{i.j} = tf_{i.j} \text{ x } \log\left(\frac{N}{df_{i.j}}\right)$$

Where,

$tf_{i.j}$ = # of occurrences of i in j

$df_{i.j}$ = # of docs with word i

N = total # of docs

i = words

j = docs

```
M = tfidf(BOW_docs, documents_norm, 'Normalized', true); % this is a sparse matrix. Not diagona
Mmatrix = full(M); % convert sparse matrix into full matrix
% 20 rows bc 20 sentences
% 26 columns bc words...but what are these words?
```

Combine with headers into matrix.

```
vocabMatrix = [vocab; num2cell(Mmatrix)]
```

```
vocabMatrix = 21×26 string array
    "look"      "just"      "stop"      "rain"      "hate"      "let"       "walk"      "sound"     "like"      "
    "2.9957"    "0"         "0"         "0"         "0"         "0"         "0"         "0"         "0"         "
    "0"         "1.7296"    "1.7296"    "0.80038"   "0"         "0"         "0"         "0"         "0"         "
    "0"         "0"         "0"         "0.69315"   "1.4979"    "1.1513"    "0.45815"   "0"         "0"         "
    "0"         "0"         "0"         "0"         "0"         "0"         "0.45815"   "1.4979"    "1.4979"    "
    "0"         "0"         "0"         "1.3863"    "0"         "0"         "0"         "0"         "0"         "
    "0"         "0"         "0"         "0.80038"   "0"         "0"         "0.52902"   "0"         "0"         "
    "0"         "0"         "0"         "0"         "0"         "0"         "0.64792"   "0"         "0"         "
    "0"         "0"         "0"         "0"         "0"         "0"         "0"         "0"         "0"         "
    "0"         "0"         "0"         "0"         "0"         "0"         "0"         "0"         "0"         "
```

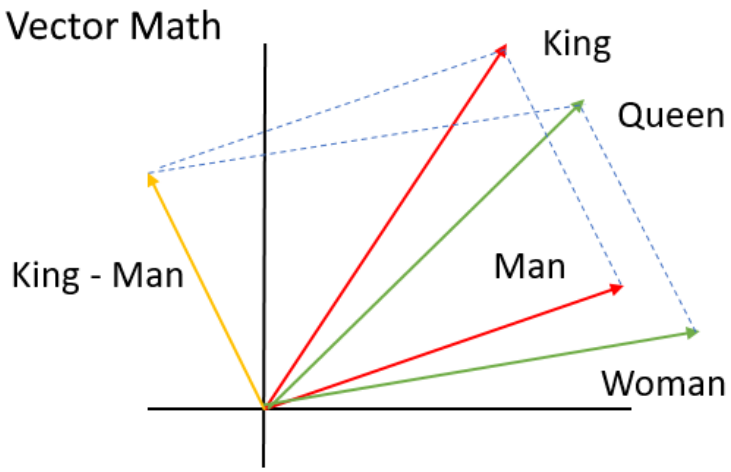
"0"	"0"	"0"	"0"	"0"	"0"	"0.52902"	"0"	"0"	"
"0"	"0"	"0"	"0"	"0"	"1.3294"	"0.52902"	"0"	"0"	"
"0"	"0"	"0"	"0"	"0"	"0"	"0.52902"	"0"	"0"	"
"0"	"0"	"0"	"0"	"0"	"0"	"0.64792"	"0"	"0"	"
"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"
"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"
"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"
"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"
"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"
"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"
"0"	"0"	"0"	"0.80038"	"0"	"0"	"0"	"0"	"0"	"

TF-IDF is nice, but vocab affects dimensionality. Also, it is very sparse. Finally, there is no meaning to the order of words. In reality, having the same words in different order changes the meaning. Also, different words can habe similar meanings, such as cancer and tumor, or dog and hound.

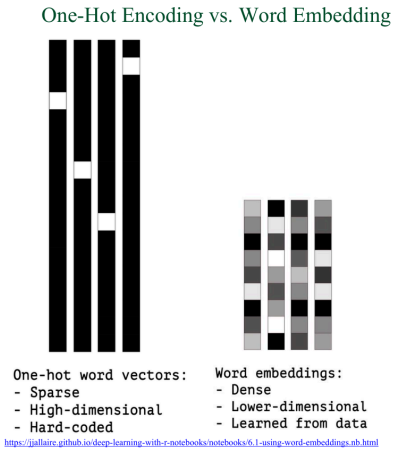
Word Embedding

Word embedding "embeds" words into a vector space model based on how often a word appears close to other words. Done at an internet scale, you can attempt to capture the semantics of the words in the vectors, so that similar words have similar vectors. One very famous example of how word embeddings can represent such relationship is that you can do a vector computation like this (Takeuchi).

$$\text{queen} \approx \text{king} - \text{man} + \text{woman}$$



Due to the shortcomings of BoW and TF-IDF, we will explore word embedding. Word embeddings are an improvement over traditional BoWs, which are too sparse. To do word embedding for better word representations, first do one hot encoding.



Similar words should have similar representations. Individual words are represented as number vectors, often high-dim. Each word is represented by a point in the embedding space. These are usually generated using NNs. There are three main word embedding techniques:

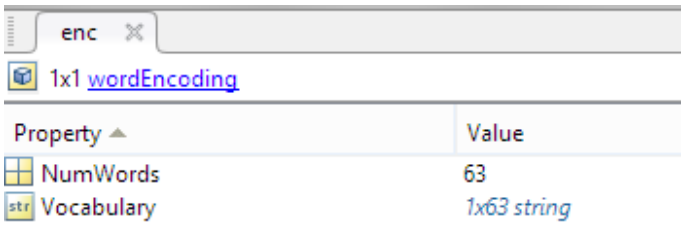
1. Embedding layer
2. Word2Vec (Skip Gram and CBoW)
3. GloVe (pretrained)

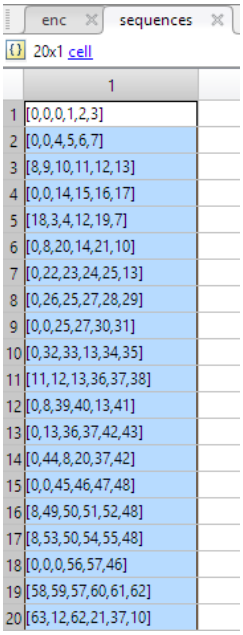
Training Embedded Layer

Encode

For this, you take the raw text, then tokenize before doing the following.

```
enc = wordEncoding(tokens);
sequences = doc2sequence(enc, tokens);
```





enc and sequences

Length of this vector should equal docs = 20

```
YTrain = categorical([1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0]');
```

What is it for?

Embedded layer training example

- Select fixed vocabulary size for one-hot. i.e. 50
- Select doc/sentence length d. i.e 4
- Pad/cut shorter/longer doc to d-length.
- Select a vector dim W (embedding space) to train embedding of each word. i.e. 8
- Train an NN for a certain task (i.e. classify sentiment) .
- Activation of each document = word represented as (#doc, W×d) vector.

First, map documents to sequences of numeric indices.

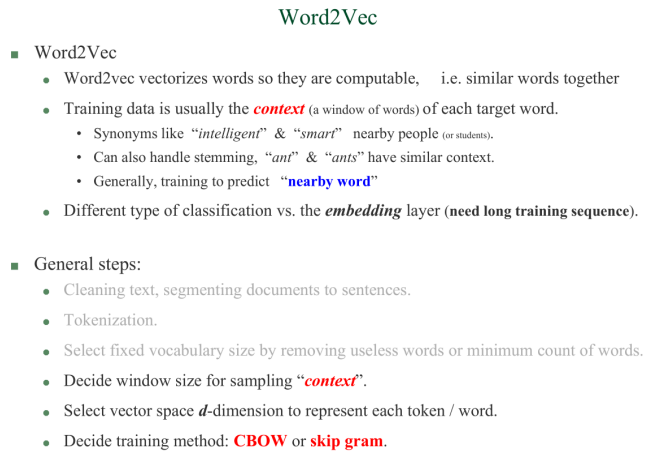
- Rather scaler indices, map words to meaningful numeric vectors.
- The embeddings capture semantic of words.
- Words with similar meanings have similar vectors.

```
inputSize = 1;
embedDim = 300;
numWords = 26; % amount of vocab words
hiddenSize = 100;
numClasses = 2;
maxEpochs = 20;
options = trainingOptions('adam', ...
    'GradientThreshold', 1, ...
    'MaxEpochs', maxEpochs, ...
    'MiniBatchSize', 5, ...
    'Verbose', 1);
Define network architecture.
layers = [ ...
    sequenceInputLayer(inputSize),
    wordEmbeddingLayer(embedDim, numWords),
    lstmLayer(hiddenSize,'OutputMode', 'last'),
    fullyConnectedLayer(numClasses),
    softmaxLayer,
    classificationLayer];
net = trainNetwork(sequences, YTrain, layers, options)
```

Word2Vec

Skip gram

I use trainWordEmbedding to train a model using Skipgram or CBoW. Example found (here).



The text data was already split into documents at newline characters, and then tokenized. E.g.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Now, specify the word embedding dimension to be 50. To reduce the number of words discarded by the model, set `'MinCount'` to 3. To train for longer, set the number of epochs to 50.

```
dimensionality = 50;
minCounter = 3;
epochs = 100;
modelChoice = 'skipgram'
```

```
modelChoice =
'skipgram'
```

```
tic
emb = trainWordEmbedding(tokens, ...
    'Dimension', dimensionality, ...
    'MinCount', minCounter, ...
    'NumEpochs', epochs, ...
    'Model', modelChoice) % skipgram is default, else Model = 'cbow'
```

```
Training: 100% Loss: 0        Remaining time: 0 hours 0 minutes.
emb =
  wordEmbedding with properties:

     Dimension: 50
    Vocabulary: ["i"    "walk"    "the"    "s"    "we"    "shoes"    "rain"]
```

```
toc
```

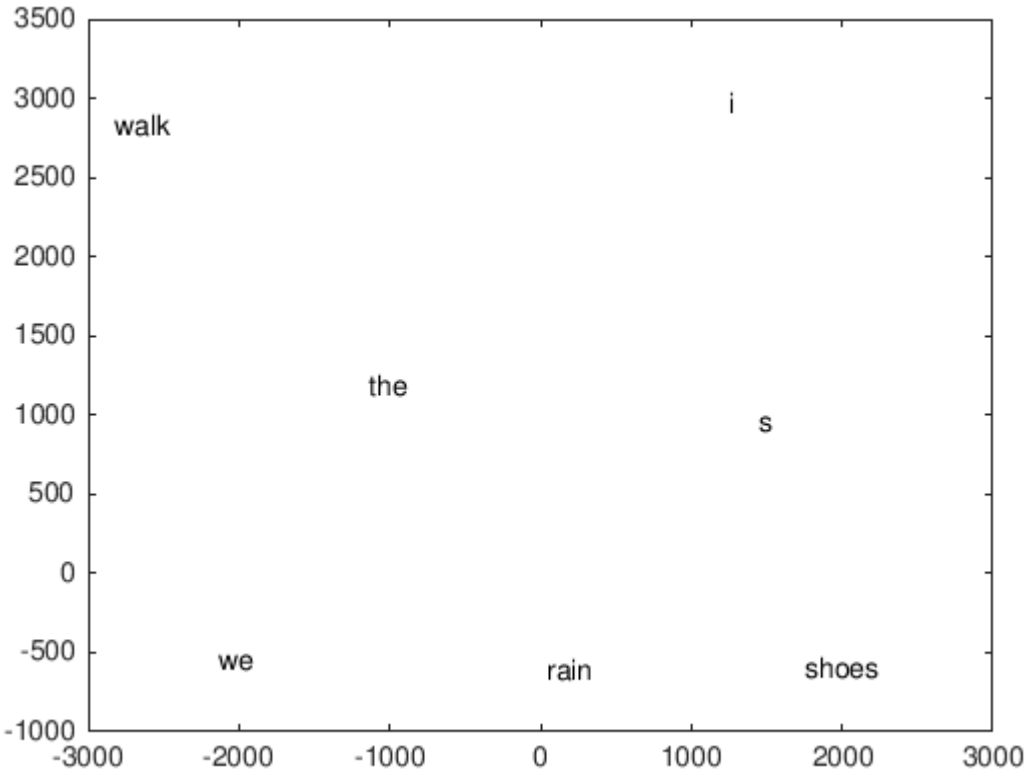
```
Elapsed time is 2.167974 seconds.
```

View the word embedding in a text scatter plot using t-sne.

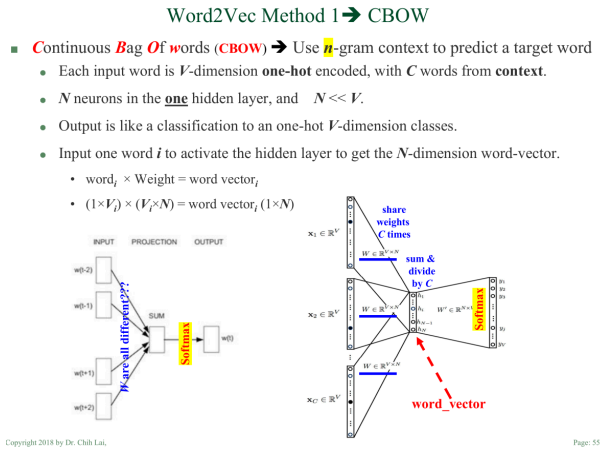
```
words = emb.Vocabulary
```

```
words = 1×7 string array
    "i"    "walk"    "the"    "s"    "we"    "shoes"    "rain"
```

```
V = word2vec(emb,words);
XY = tsne(V);
textscatter(XY,words)
```



Continuous Bag Of words (CBOW)



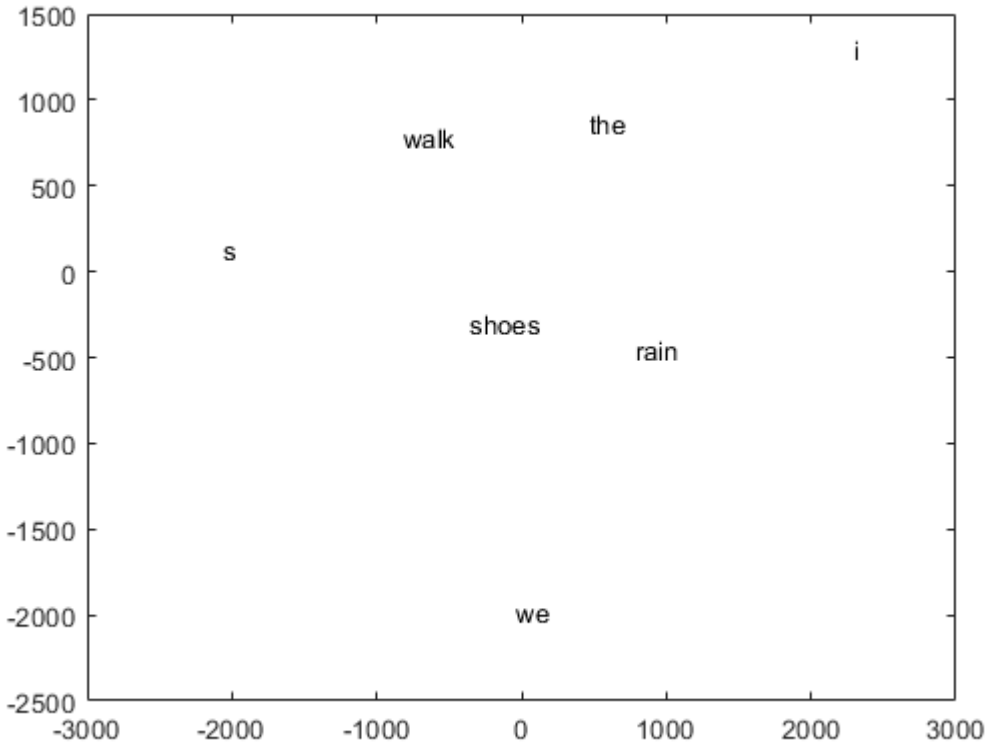

```
embcbow = trainWordEmbedding(tokens, ...
    'Dimension', dimensionality, ...
    'MinCount', minCounter, ...
    'NumEpochs', epochs, ...
    'Model', 'cbow') % skipgram is default, else Model = 'cbow'
```

```
Training: 100% Loss: 0        Remaining time: 0 hours 0 minutes.
embcbow =
  wordEmbedding with properties:

     Dimension: 50
    Vocabulary: ["i"    "walk"    "the"    "s"    "we"    "shoes"    "rain"]
```

View the word embedding in a text scatter plot using t-sne.

```
wordsembcbow = embcbow.Vocabulary;
Vembcbow = word2vec(embcbow,words);
XYembcbow = tsne(Vembcbow);
textscatter(XYembcbow,wordsembcbow)
```



Transfer Learning

FastTex is a Matlab solution to pretrained word embeddings. It contains 16 Billion English token word embeddings. See here on how to implement FastText.

```
embFast = fastTextWordEmbedding
```

```
embFast =
  wordEmbedding with properties:

     Dimension: 300
    Vocabulary: [1×999994 string]
```

View the word embedding in a text scatter plot using t-sne.

```
words = embFast.Vocabulary;
V = word2vec(embFast, words);
XY = tsne(V);
textscatter(XY, words)
```

GloVe

nlp.stanford.edu/data/glove.6B.zip

```
filename = "glove.6B.300d";
if exist(filename + '.mat', 'file') ~= 2
```

```
emb = readWordEmbedding(filename + '.txt');
save(filename + '.mat', 'emb', '-v7.3');
else load(filename + '.mat')
end
v_king = word2vec(emb, 'king')';
```

According to this post, you can use a pre-trained embedding model to initialize the Weights property of the wordEmbeddingLayer. For example:

```
% Import your pretrained word embedding model of choice
emb = readWordEmbedding('existingEmbeddingModel.vec');
```

```
Error using textanalytics.internal.validateReadable (line 25)
Unable to open file 'existingEmbeddingModel.vec' for reading: No such file or directory.

Error in readWordEmbedding (line 22)
    cfile = textanalytics.internal.validateReadable(cfile);
```

```
embDim = emb.Dimension;
numWords = numel(emb.Vocabulary);

% Initialize the word embedding layer
embLayer = wordEmbeddingLayer(embDim, numWords);
embLayer.Weights = word2vec(emb, emb.Vocabulary)';

% If you want to keep the original weights "frozen", uncomment the following line
embLayer.WeightLearnRateFactor = 0
```

The wordEmbeddingLayer with initialized Weights can then be placed in the network before lstmLayer.

Also note that training documents should be mapped according to the vocabulary of the pre-trained embedding model, before passing to the net for training, for example:

```
enc = wordEncoding(tokenizedDocument(emb.Vocabulary,'TokenizeMethod','none'));
XTrain = doc2sequence(enc,documentsTrain,'Length',75);
```

Future project?

https://www.mathworks.com/help/textanalytics/ug/classify-text-data-using-deep-learning.html