Embedding Vectors

Jeff Allard

Lead Data Scientist

5/3 Bancorp

https://github.com/AllardJM/DataScience_Meetup_Presentations

About

- Highly effective in several domains e.g. Natural Language Processing (Word2Vec)
- Finding more applications across machine learning problems generally

An Embedding Vector is simply a series of numbers that attempts to encode latent features of an object, e.g.

- A word / sentence
- A customer
- A movie
- ……

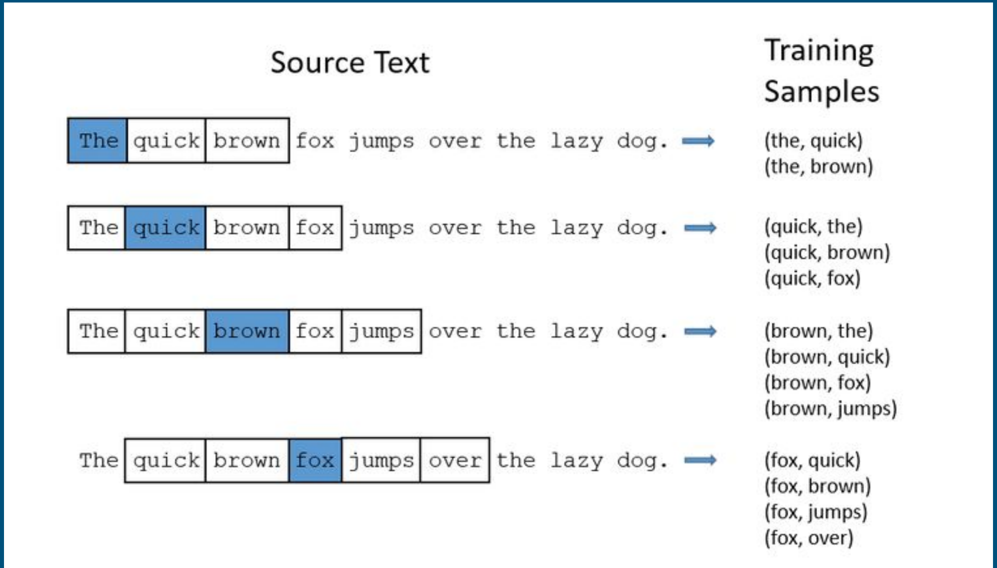
Canonical Example -- Text Encoding

- Traditionally, models, say text classification, utilized a bag of words approach or some variant
  - Length of the variable = size of the vocabulary - the vector is sparse - all zeros except for position "i" which represents the specific term
  - "Cat": [0,0,0...,0,0,1,0,0,....0]
  - "Pet": [0,0,1...,0,0,0,0,0,....0]
- Problems:
  - Very large sparse vectors are hard to work with in downstream algorithms
  - Related words encoded orthogonally
  - Need for heuristics such as word stemming, stop word removal, user maintained dictionaries of synonyms / domain specific words etc

Solution: Word2Vec

- Originally published by Google researcher in 2013
- Many variants and subsequent alternatives
- The basic idea:
    - Represent a word as as dense vector
    - “Cat” = [0.2,-1.25,-0.0256,...0.45]
- Benefits
    - Size of the vector is much smaller than Bag-of-Words (e.g. 200)
    - Related words are similar in vector space terms (e.g cosine similarity)

Word2Vec : Basic Idea of Skip-gram method



- Window through a corpus
- Select a word (e.g. quick)
- Predict the likelihood that another word (e.g. fox) is within the context of the word -- context here is a window size of 2
- Train a neural network with positive and randomly chosen negative examples
- The weights from a shallow net are the vectors for a given word
- **Words with similar context will have learned representations that are similar**
  - E.g. Cat and Pet, Smart and Intelligent

Word2Vec

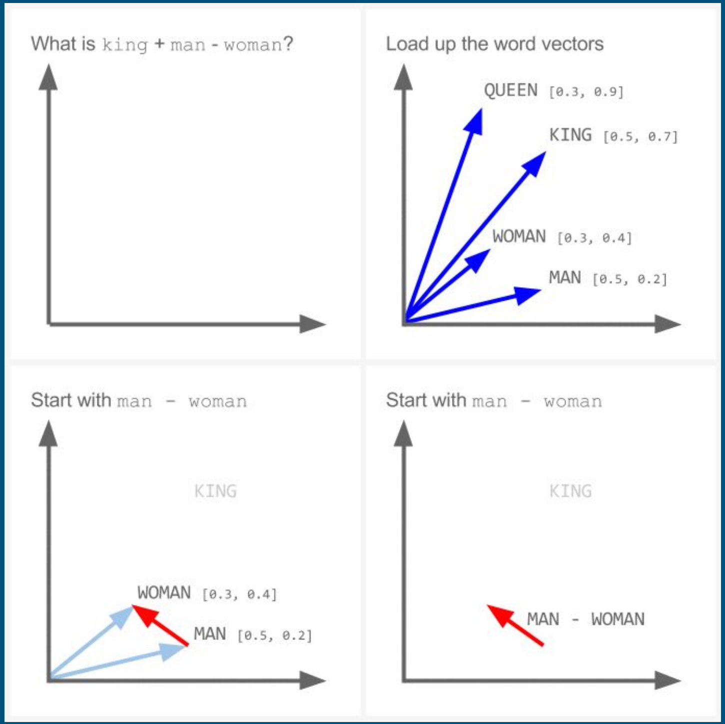
- Intuitively….the elements of a vector describe concepts or characteristics of a term
  - E.g. One element of “King” may describe the strength of “gender”
  - Found that simple operations on these vectors displayed deeper concepts

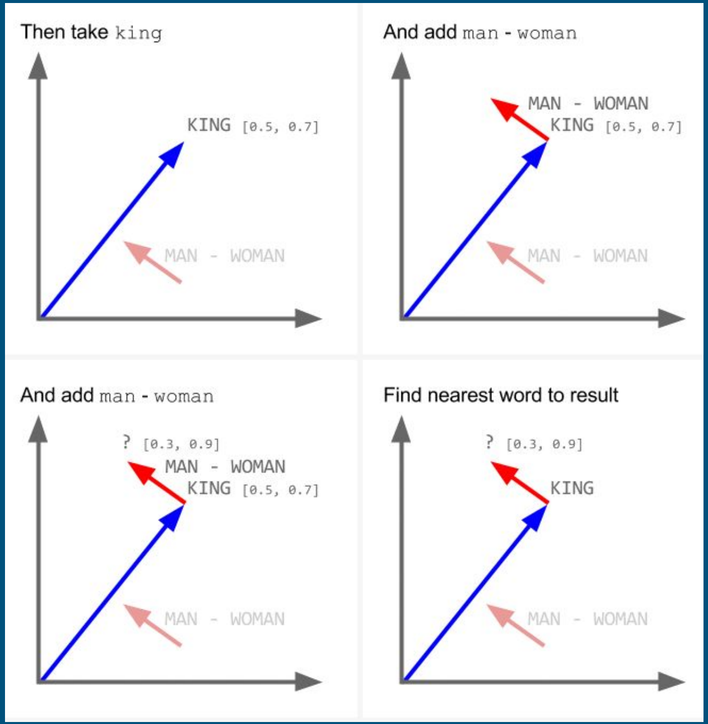
[king] - [man] + [woman] ~= [queen]

- If we take the vector for “king”, subtract the vector for “man” and add the vector of “woman”, the resulting vector will be the closest to….the vector for queen (cosine similarity)
    - King - man removes the male gender component from the concept of monarch

Word2Vec

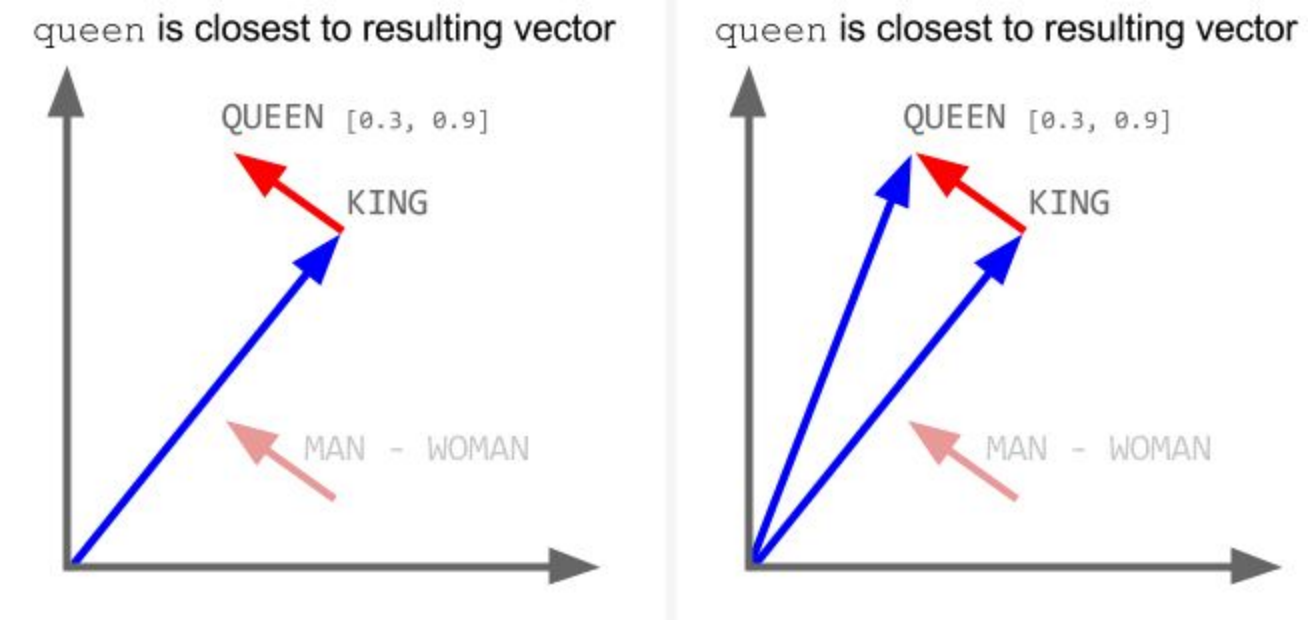
- [0.5,0.2] - [0.3,0.4] = [0.2,-0.2]
  (Man - Woman)
- [0.5,0.7] - [0.2,-0.2] = [0.3,0.9]
  (King + (Man - Woman)) ~ Queen





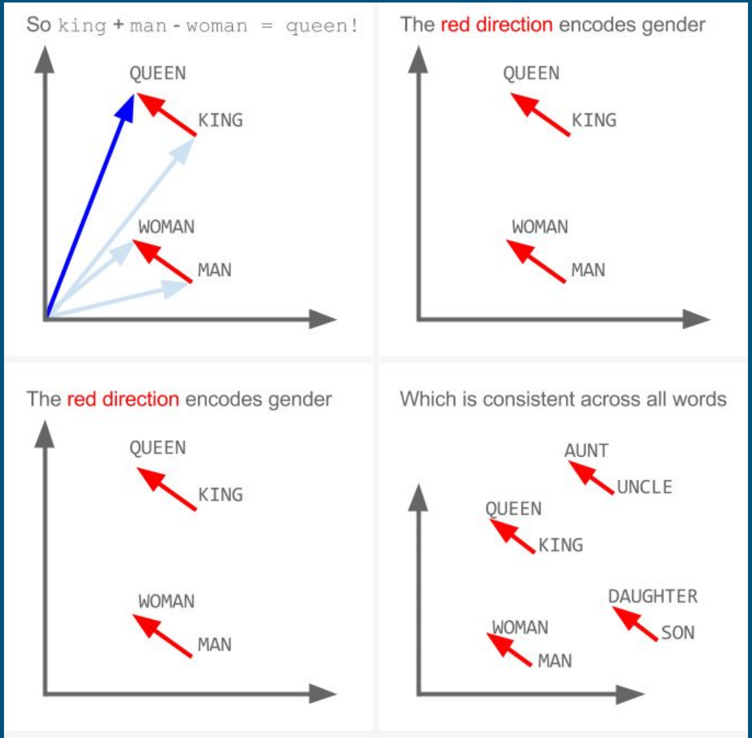
https://multithreaded.stitchfix.com/assets/images/blog/vectors.gif

Word2Vec

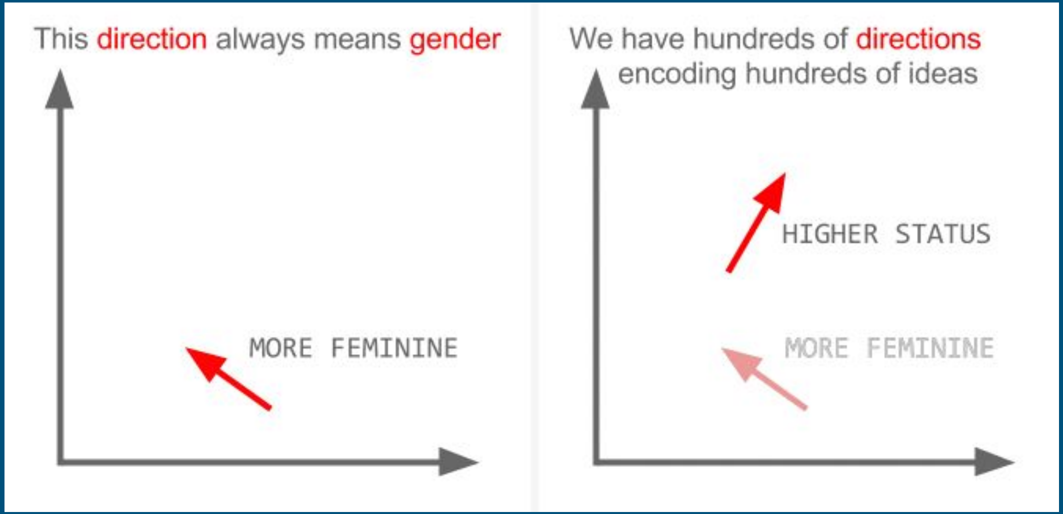


https://multithreaded.stitchfix.com/assets/images/blog/vectors.gif

Word2Vec



https://multithreaded.stitchfix.com/assets/images/blog/vectors.gif



Word2Vec

- Closing Thoughts
  - Intuitively the components of the vectors are concepts or mixtures of concepts
  - In practice not always interpretable
  - The vector dimension is often chosen empirically
  - Easy to train vectors on your own domain if you have enough data, else use pre-trained ones (e.g. trained on massive corpus like wikipedia) and fine tune them (transfer learning) for your task

Entity Embeddings

- Generalization - Not just for words or text
- A word is just some ‘token’, so extend this idea to things like
  - Customer IDs
  - Movie IDs
  - Zip codes
  - ……
- Being used for state of the art predictive models
  - Allows use of high cardinality nominal variables like a customer’s census tract
  - Even used for lower cardinality variables like a customers State (replacing dummy variables)

Application : Recommendation Engines

User Latent Factors			
#1	#2	#3	User ID
0.8949101073	-0.6265794545	0.74757374	0
0.7947220104	0.8636220878	0.312756333	1
-0.2558703818	0.8342031778	0.081368916	2
0.940665854	-0.5790454715	0.962063664	3
0.9222317515	0.9514060606	0.432131189	4
0.8462968955	0.1530153498	-0.50570987	5
0.1811727399	0.1472382755	0.756643752	6

Movie Latent Factors			
#1	#2	#3	Movie ID
0.8239053285	0.8610092645	0.083420775	0
0.3149946319	0.2860168437	0.179063335	1
0.528859128	-0.8185830713	-0.519957348	2
-0.3441695914	0.1525521576	0.133796819	3
0.851516663	-0.2228508417	0.227523797	4

User Factor #1: How much user 0 likes action movies?

Movie Factor #3: How much action movie 0 has?

Embeddings in Keras - Example 1

```python
vocab_size=5  #we have 5 words in our vocabulary (0,1,2,3,4) -- generally think of this as 5 unqiue tokens
             #(e.g. words, symbols, user IDs, Movie IDs, Product IDs )
embedding_size=3 #there are three latent factors that describe our words


embedding_layer = Embedding(output_dim=embedding_size, \
                            input_dim=vocab_size, \
                            input_length=1,\
                            mask_zero=True)

x = Input(shape=[1])
embedding = embedding_layer(x)
model = Model(inputs=x, outputs=embedding)

print(model.summary())
print("")
print(" ")
print("------------------------------------------------------------------")
print("------------------------------------------------------------------")

print("Input shape: ", model.input_shape) #this "model" inputs a single number
print("Output shape: ", model.output_shape) #this "model" exports a length (embedding_size) vector

print("Weight Matrix shape: ", np.array(model.get_weights()).shape) #shape of the embedding matrix is
                                                                    #(1, vocab_size, embedding_size)
print(" ")
print(" ")
model.get_weights() #The embedding weights
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 1)                 0
_________________________________________________________________
embedding_4 (Embedding)      (None, 1, 3)              15
=================================================================
Total params: 15
Trainable params: 15
Non-trainable params: 0
_________________________________________________________________
None


------------------------------------------------------------------
------------------------------------------------------------------
Input shape:  (None, 1)
Output shape:  (None, 1, 3)
Weight Matrix shape:  (1, 5, 3)


[array([[-0.02434868,  0.02641512,  0.02833296],
        [-0.0264437 , -0.04659697,  0.00371159],
        [ 0.02535654, -0.01593039,  0.0152105 ],
        [-0.04532808, -0.006714  , -0.02591713],
        [-0.00216939,  0.04102891, -0.01888945]], dtype=float32)]
```

```python
#simply a lookup

X = np.array(([2],[2],[1])) #3rd, 3rd, 2nd words
model.predict(X)
```

```
array([[[ 0.02535654, -0.01593039,  0.0152105 ]],

       [[ 0.02535654, -0.01593039,  0.0152105 ]],

       [[-0.0264437 , -0.04659697,  0.00371159]]], dtype=float32)
```

Embeddings in Keras - Example 2

```python
vocab_size_1=7  #User IDs?
vocab_size_2=5 #movie IDs?

embedding_size=3 #constant


embedding_layer_1 = Embedding(output_dim=embedding_size, input_dim=vocab_size_1,input_length=1, mask_zero=False)
embedding_layer_2 = Embedding(output_dim=embedding_size, input_dim=vocab_size_2,input_length=1, mask_zero=False)

userIDs = Input(shape=[1])
movieIDs = Input(shape=[1])


embedding_users = embedding_layer_1 (userIDs)
embedding_movies = embedding_layer_2(movieIDs)

x= concatenate([embedding_users,embedding_movies])
x=Flatten()(x)

model = Model(inputs=[userIDs,movieIDs], outputs=x)

print(model.summary())
print(" ")
print(" ")
print("-----------------------------------------------------------------")
print("-----------------------------------------------------------------")
print(" ")
print("Input shape: ", model.input_shape) #this "model" inputs a single number
print("Output shape: ", model.output_shape) #this "model" exports a length (embedding_size) vector

print(" ")
print("User weight Matrix shape: ", np.array(model.get_weights()[0]).shape)
print("Movies weight Matrix shape: ", np.array(model.get_weights()[1]).shape)

print(" ")
print("User embedding weights")
print("-----------------------------------------------------------------")
print(model.get_weights()[0])
print(" ")
print(" ")
print("Movie embedding weights")
print("-----------------------------------------------------------------")
print(model.get_weights()[1])
```

2 Embedding matrices concatenated and flattened

```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_5 (InputLayer)            (None, 1)            0

input_6 (InputLayer)            (None, 1)            0

embedding_5 (Embedding)         (None, 1, 3)         21          input_5[0][0]

embedding_6 (Embedding)         (None, 1, 3)         15          input_6[0][0]

concatenate_1 (Concatenate)     (None, 1, 6)         0           embedding_5[0][0]
                                                                 embedding_6[0][0]

flatten_1 (Flatten)             (None, 6)            0           concatenate_1[0][0]
==================================================================================================
Total params: 36
Trainable params: 36
Non-trainable params: 0

None


-----------------------------------------------------------------
-----------------------------------------------------------------

Input shape:  [(None, 1), (None, 1)]
Output shape:  (None, 6)

User weight Matrix shape:  (7, 3)
Movies weight Matrix shape:  (5, 3)

User embedding weights
-----------------------------------------------------------------
[[-0.01192247  0.0029695  -0.04093463]
 [ 0.03142424 -0.04858527  0.01714227]
 [-0.00115309  0.0027717   0.0452025 ]
 [-0.00717747  0.00594933  0.00543087]
 [-0.00803168 -0.04429523 -0.00055258]
 [ 0.00224601 -0.01125713 -0.01098534]
 [-0.01774049 -0.04440355  0.04540284]]


Movie embedding weights
-----------------------------------------------------------------
[[ 0.01013019  0.04473348 -0.03012686]
 [ 0.01903756 -0.02693369  0.02577094]
 [-0.01932679  0.01362795  0.03613641]
 [-0.02342827  0.00045323 -0.019732  ]
 [-0.04033861  0.01433581 -0.03953437]]
```

```python
X = [np.array(([1])),np.array(([2]))] #2nd UserID and 3rd Movie embeddings concatenated
model.predict(X)
```

```
array([[ 0.03142424, -0.04858527,  0.01714227, -0.01932679,  0.01362795,
         0.03613641]], dtype=float32)
```

Demo

- Highly effective deep learning model for recommending items to Pinterest users
- Using only user and item IDs
- Training from scratch with gradient descent
- Keras and Tensorflow

https://github.com/AllardJM/DataScience_Meetup_Presentations/tree/master/April_2017_EmbeddingVectors