

# INTRODUCTION TO WORD EMBEDDINGS

Yogesh Kulkarni

November 24, 2020

# Word Vectors

## What are Word Vectors/Embeddings?

- ▶ Word Embeddings are the texts converted into numbers
- ▶ There may be different numerical representations of same text.
- ▶ Many Machine Learning algorithms and almost all Deep Learning Architectures are incapable of processing strings or plain text in their raw form.
- ▶ They require numbers as inputs to perform any sort of job, be it classification, regression etc. in broad terms.
- ▶ So, for the computer to be able to "understand" a vector representation of a word is required.

## Different types of Word Vectors

- ▶ (Traditional) Frequency based Embedding:
  - ▶ One-hot
  - ▶ Count Vector
  - ▶ TF-IDF Vector
  - ▶ Co-Occurrence Vector
- ▶ (Modern) Prediction based Embedding:
  - ▶ Word2vec (Google)
  - ▶ Global Vector Representations (GloVe) (Stanford)

# One Hot

In traditional NLP, we regard words as discrete symbols: **hotel**, **conference**, **motel**

Means one 1, the rest 0s



Words can be represented by **one-hot** vectors:

**motel** = [0 0 0 0 0 0 0 0 0 1 0 0 0]

**hotel** = [0 0 0 0 0 0 1 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g. 500,000)

(Ref: Word Embeddings - Elena Voita, Yandex Research)

## One Hot: Problem

Example: in web search, if user searches for “**Seattle motel**”, we would like to match documents containing “**Seattle hotel**”.

But:

**motel** = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

**hotel** = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal.

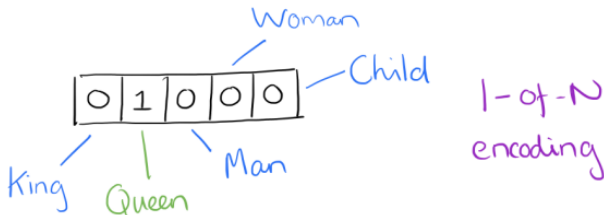
There is no natural notion of **similarity** for one-hot vectors!

These vectors do not contain information about a **meaning** of a word.

(Ref: Word Embeddings - Elena Voita, Yandex Research)

## One Hot: Example

One-hot: Suppose our vocabulary has only five words: King, Queen, Man, Woman, and Child. We could encode the word 'Queen' as:



No meaningful comparison possible. We will look at some vectorization schemes that can capture “meaning”, somewhat.

# Count Vector

- ▶ Corpus:
  - ▶ D1: He is a lazy boy. She is also lazy.
  - ▶ D2: Neeraj is a lazy person.
- ▶ Dictionary is a list of unique tokens(words)  
=['He','She','lazy','boy','Neeraj','person']
- ▶ Count Matrix:

	He	She	lazy	boy	Neeraj	person
D1	1	1	2	1	0	0
D2	0	0	1	0	1	1



## Count Vector

	Document 1	Document 2	Document 3	Document 4	Document 5	Document 6	Document 7	Document 8
Term(s) 1	10	0	1	0	0	0	0	2
Term(s) 2	0	2	0	0	0	18	0	2
Term(s) 3	0	0	0	0	0	0	0	2
Term(s) 4	6	0	0	4	6	0	0	0
Term(s) 5	0	0	0	0	0	0	0	2
Term(s) 6	0	0	1	0	0	1	0	0
Term(s) 7	0	1	8	0	0	0	0	0
Term(s) 8	0	0	0	0	0	3	0	0

← Word Vector  
(Passage Vector)

Document Vector

## TF-IDF vectorization

- ▶ It takes into account not just the occurrence of a word in a single document but in the entire corpus.
- ▶ Down weight the common words occurring in almost all documents and give more importance to words that appear in a subset of documents.
- ▶ Corpus:

Document 1

Term	Count
This	1
is	1
about	2
Messi	4

Document 2

Term	Count
This	1
is	2
about	1
Tf-idf	1

## TF-IDF vectorization

- ▶  $TF = (\text{Number of times term } t \text{ appears in a document}) / (\text{Number of terms in the document})$
- ▶ So,  $TF(\text{This}, \text{Document1}) = 1/8$  and  $TF(\text{This}, \text{Document2}) = 1/5$
- ▶  $IDF = \log(N/n)$ , where,  $N$  is the number of documents and  $n$  is the number of documents a term  $t$  has appeared in.
- ▶ So,  $IDF(\text{This}) = \log(2/2) = 0$ .
- ▶  $TFIDF = TF * IDF$
- ▶ Dictionary is made of a list of unique tokens(words)
- ▶ Similar to Count Matrix, TFIDF matrix is made with TFIDF values in it.

## TF-IDF Inferencing

Once tf-idf model is trained on a corpus of documents, what happens when a test word or a document is given? How does it calculate tf-idf vector for it?

- ▶ For just one test word, its tf would be 1, then do we just multiply idf of that word from trained model? Are tf values in trained model irrelevant?
- ▶ Even for test document, does `inverse_transform` take into account tfs of current test document or tfs within trained model?
- ▶ Answer: the test word or document is 'ADDED' to the corpus and retraining/calculation is done to arrive at meaningful numbers.

(Ref: <https://stackoverflow.com/questions/55707577/how-does-tfidfvectorizer-compute-scores-on-test-data> )

## Co-Occurrence Matrix

- Co-occurrence ' For a given corpus, the co-occurrence of a pair of words say  $w_1$  and  $w_2$  is the number of times they have appeared together in a Context Window.
- Context Window ' Context window is specified by a number and the direction. So what does a context window of 2 (around) means?

Quick	Brown	Fox	Jump	Over	The	Lazy	Dog
-------	-------	-----	------	------	-----	------	-----

- For Corpus = " He is not lazy. He is intelligent. He is smart."

	He	is	not	lazy	intelligent	smart
He	0	4	2	1	2	1
is	4	0	1	2	2	1
not	2	1	0	1	0	0
lazy	1	2	1	0	0	0
intelligent	2	2	0	0	0	0
smart	1	1	0	0	0	0

## Co-Occurrence Matrix

Red box- It is the number of times 'He' and 'is' have appeared in the context window 2 and it can be seen that the count turns out to be 4.

He	is	not	lazy	He	is	intelligent	He	is	smart
He	is	not	lazy	He	is	intelligent	He	is	smart
He	is	not	lazy	He	is	intelligent	He	is	smart
He	is	not	lazy	He	is	intelligent	He	is	smart

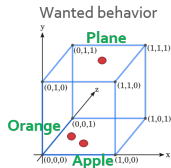
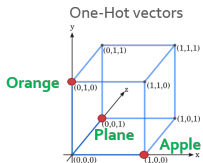
While the word 'lazy' has never appeared with 'intelligent' in the context window and therefore has been assigned 0 in the blue box.

## Good Vector Representation

- ▶ To have "Semantic" (meaning-wise) representation, the Similar words should be close to each other in the hyper dimensional space.
- ▶ Non-similar words should be far apart from each other in the hyper dimensional space.

# Good Vector Representation

- ▶ Traditional One Hot Encoding:
  - ▶ Apple = [1, 0, 0]
  - ▶ Orange = [0, 1, 0]
  - ▶ Plane = [0, 0, 1]



- ▶ Very few cells participate in the representation.



## But, What is “meaning”?

What is “bardiwac”?

Anyone?

## Lets try again, with examples

What is “bardiwac”?

- ▶ He handed her a glass of bardiwac.
- ▶ Beef dishes are made to complement the bardiwac.
- ▶ Nigel staggered to his feet, face flushed from too much bardiwac.
- ▶ Malbec, one of the lesser-known bardiwac grapes, responds well to Australia's sunshine.
- ▶ I dined off bread and cheese and this excellent bardiwac.
- ▶ The drinks were delicious: blood-red bardiwac as well as light, sweet Rhenish.

Now, anyone?

## At least one can guess now

What is “bardiwac”?

“Bardiwac is a red alcoholic beverage made from grapes ”

Context helps . . .

## Distributed Semantics/Meaning

- ▶ A bottle of bardiwac is on the table.
- ▶ Everybody likes bardiwac.
- ▶ Don't have bardiwac before you drive.
- ▶ We make bardiwac out of corn.

## Distributed Semantics/Meaning

- ▶ A bottle of xxxxxxxx is on the table.
- ▶ Everybody likes xxxxxxxx.
- ▶ Don't have xxxxxxxx before you drive.
- ▶ We make xxxxxxxx out of corn.

What other words fit into these places?

Won't they be similar to bardiwac?

# Distributed Semantics/Meaning

- A bottle of \_\_\_\_\_ is on the table. (1)
- Everybody likes \_\_\_\_\_. (2)
- Don't have \_\_\_\_\_ before you drive. (3)
- We make \_\_\_\_\_ out of corn. (4)

What other words fit into these contexts?

	(1)	(2)	(3)	(4)	...
bardiwac	1	1	1	1	
loud	0	0	0	0	
motor oil	1	0	0	1	
tortillas	0	1	0	1	
wine	1	1	1	0	
choices	0	1	0	0	

(Ref: Word Embeddings - Elena Voita, Yandex Research)

## Distributed Semantics/Meaning

Closer ones are ...

Does vector similarity imply semantic similarity?



The **distributional hypothesis**, stated by Firth (1957):

*"You shall know a word by the company it keeps."*

(Ref: Word Embeddings - Elena Voita, Yandex Research)

## Distributed Semantics/Meaning

Closer ones are ...

Does vector similarity imply semantic similarity?



The **distributional hypothesis**, stated by Firth (1957):

*"You shall know a word by the company it keeps."*

(Ref: Word Embeddings - Elena Voita, Yandex Research)



# Distributed Semantics/Meaning

Idea of Co-occurrence counts . . .

## Corpus sentences

He also found five fish swimming in murky water in an old **bathub**.

We do abhor dust and dirt, and stains on the **bathub**, and any kind of filth.

Above At the far end of the garden room a **bathub** has been planted with herbs for the winter.

They had been drinking Cisco, a fruity, wine-based fluid that smells and tastes like a mixture of cough syrup and **bathub** gin.

Science finds that a surface tension on the water can draw the boats together, like toy boats in a **bathub**.

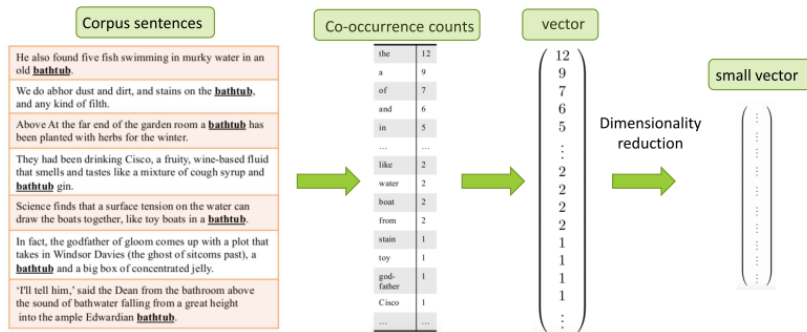
In fact, the godfather of gloom comes up with a plot that takes in Windsor Davies (the ghost of sitcoms past), a **bathub** and a big box of concentrated jelly.

'I'll tell him,' said the Dean from the bathroom above the sound of bathwater falling from a great height into the ample Edwardian **bathub**.

(Ref: Word Embeddings - Elena Voita, Yandex Research)

# Distributed Semantics/Meaning

Calculate Co-occurrences for the context word, that itself becomes its own representation!!!



(Ref: Word Embeddings - Elena Voita, Yandex Research)

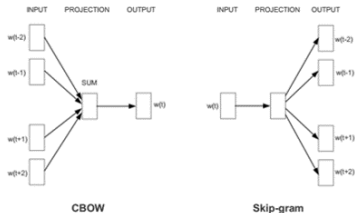
## Word Vectors in Practice

## Building these magical vectors

- ▶ How do we actually build these super-intelligent vectors, that seem to have such magical powers?
- ▶ How to find a word's friends?
- ▶ We will discuss the most famous methods to build such lower-dimension vector representations for words based on their context
  - ▶ Co-occurrence Matrix with SVD
  - ▶ word2vec (Google)
  - ▶ Global Vector Representations (GloVe) (Stanford)

# Language Model

- ▶ A good statistical model for NLP is the conditional probability of the next word  $w$  given its previous ones
- ▶ Takes advantage of both word order, and the fact that temporally closer words have a stronger dependency.
- ▶ Continues Bag - of - Words (CBOW): predicts a word given its context (bidirectional).
- ▶ Skip - Gram: predicts the context given a word (bidirectional).

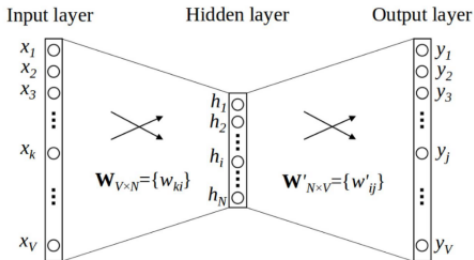


(Ref: Distributed Representations of Words and Phrases and their Compositionality. Mikolov et al., 2013)

## Context windows

- ▶ Context can be anything - a surrounding n-gram, a randomly sampled set of words from a fixed size window around the word
- ▶ For example, assume context is defined as the word following a word.  
 $context(w_i) = w_{i+1}$
- ▶ Corpus : I ate the cat
- ▶ Training Set :  $I|ate, ate|the, the|cat, cat|$ .

# Google's Word2Vec



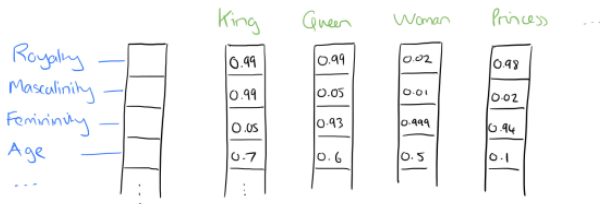
- a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position  $t$  in the text, which has a center word  $c$  and context ("outside") words  $o$
- Use the similarity of the word vectors for  $c$  and  $o$  to **calculate the probability** of  $o$  given  $c$  (or vice versa)
- **Keep adjusting the word vectors** to maximize this probability

Mikolov et al, 2013, <https://arxiv.org/pdf/1310.4546.pdf>

(Ref: Word Embeddings - Elena Voita, Yandex Research)

## Word2Vec

Word2vec (Google): a distributed representation of a word is used and not sparse like One-Hot.

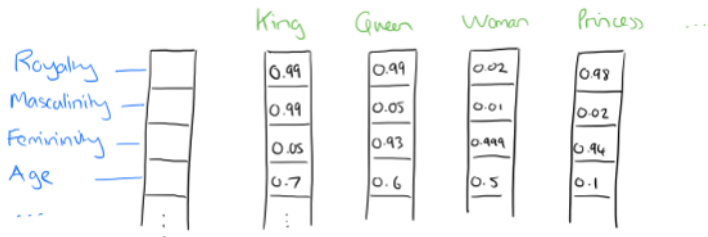


Represent in some abstract way the 'meaning' of a word.



## Word Distributed Representation - Word2Vec

- ▶ All vector cells participate in representing each word.
- ▶ Words are represented by real valued dense vectors of significantly smaller dimensions (e.g. 100 - 1000).
- ▶ Intuition: consider each vector cell as a representative of some feature.



## Word Representations Comparison

### Traditional Method - Bag of Words Model

- ▶ Uses one hot encoding
- ▶ Each word in the vocabulary is represented by one bit position in a HUGE vector.
- ▶ For example, with a vocabulary of 10000 words, and "Hello" is the 4th word in the dictionary: 0 0 0 1 0 0 .  
 . . . . . 0 0 0 0
- ▶ Context information is not utilized

### Modern - Word Vectors

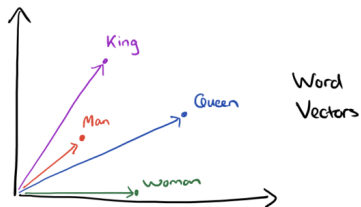
- ▶ Stores each word in as a point in space, represented by a vector of fixed number of dimensions (generally 300)
- ▶ Unsupervised, built just by reading huge corpus
- ▶ For example, "Hello" might be represented as : [0.4, -0.11, 0.55, 0.3 . . . 0.1, 0.02]
- ▶ Context information is utilized

## The Power of Word2Vecs

- ▶ They provide a fresh perspective to ALL problems in NLP, and not just solve one problem.
- ▶ Technological Improvement
- ▶ Rise of deep learning since 2006 (Big Data + GPUs + Work done by Andrew Ng, Yoshua Bengio, Yann Lecun and Geoff Hinton)
- ▶ Application of Deep Learning to NLP - led by Yoshua Bengio, Christopher Manning, Richard Socher, Tomas Mikalov
- ▶ The need for unsupervised learning . (Supervised learning tends to be excessively dependent on hand-labeled data and often does not scale)

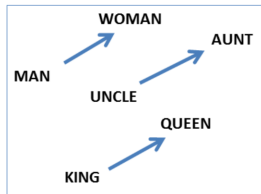
## Examples

Vectors for King, Man, Queen, & Woman:

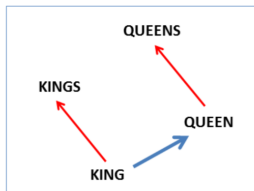


## Examples

Gender relation:



Plural relation:



# Examples

## Word pair relationships:

Table 8: Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

## Country-capital city relationship:

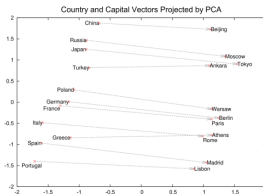


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

## How to evaluate embeddings

Intrinsic: evaluation on a specific/intermediate subtask

- ▶ word analogies: “a is to b as c is to xxxx?”
- ▶ word similarity: correlation of the rankings

Extrinsic: evaluation on a real task

- ▶ take some task (MT, NER, coreference resolution, ...) or several tasks
- ▶ train with different pretrained word embeddings
- ▶ if the task quality is better -j win!

## Applications of Word Vectors



## Applications of Word Vectors

### Word Similarity:

- ▶ Classic Methods : Edit Distance, WordNet, Porter's Stemmer, Lemmatization using dictionaries
- ▶ Easily identifies similar words and synonyms since they occur in similar contexts
- ▶ Stemming (thought → think)
- ▶ Inflections, Tense forms
- ▶ eg. Think, thought, ponder, pondering,
- ▶ eg. Plane, Aircraft, Flight

- Classic Methods : Rule-based machine translation, morphological transformation



## Applications of Word Vectors

Part-of-Speech and Named Entity Recognition:

- ▶ Classic Methods : Sequential Models (MEMM , Conditional Random Fields), Logistic Regression

	POS WSJ (acc.)	NER CoNLL (F1)
State-of-the-art*	97.24	89.31
Supervised NN	<b>96.37</b>	<b>81.47</b>
Unsupervised pre-training followed by supervised NN**	<b>97.20</b>	<b>88.87</b>
+ hand-crafted features***	97.29	89.59

# Applications of Word Vectors

## Sentiment Analysis:

- ▶ Classic Methods : Naive Bayes, Random Forests/SVM
- ▶ Classifying sentences as positive and negative
- ▶ Building sentiment lexicons using seed sentiment sets
- ▶ No need for classifiers, we can just use cosine distances to compare unseen reviews to known reviews.

```
Enter word or sentence (EXIT to break): sad
Word: sad Position in vocabulary: 4067
```

Word	Cosine distance
saddening	0.727309
Sad	0.661083
saddened	0.660439
heartbreaking	0.657351
disheartening	0.650732
Meny_Friedman	0.648706
parishioner_Pat_Patello	0.647586
saddens_me	0.640712
distressing	0.639909
reminders_bobbing	0.635772
Turkoman_Shites	0.635577
saddest	0.634551
unfortunate	0.627209
sorry	0.619405
bittersweet	0.617521
tragic	0.611279
regretful	0.603472

# Applications of Word Vectors

## Sentiment Analysis:

- ▶ Co-reference Resolution: Chaining entity mentions across multiple documents - can we find and unify the multiple contexts in which mentions occurs?
- ▶ Clustering: Words in the same class naturally occur in similar contexts, and this feature vector can directly be used with any conventional clustering algorithms (K-Means, agglomerative, etc). Human doesn't have to waste time hand-picking useful word features to cluster on.
- ▶ Semantic Analysis of Documents: Build word distributions for various topics, etc.

## Weakness of Word Embedding

- ▶ Very vulnerable, and not a robust concept
- ▶ Can take a long time to train
- ▶ Non-uniform results
- ▶ Hard to understand and visualize

## Summary

Machines like humans need four things to understand language

- ▶ Understand the words (semantics)
- ▶ Build the ability to guess (language model)
- ▶ Parse language specific rules and patterns (encoder-decoder, transformers)
- ▶ Build on the experience (pre-training)

Ability to include non-language information (culture, visuals, etc) will improve language models.

# Word Embeddings in Tensorflow

(Ref: How to Use Word Embedding Layers for Deep Learning with Keras by Jason Brownlee)



## Word Embedding (Recap)

- ▶ Word embeddings provide a dense representation of words and their relative similarity.
- ▶ Improvement over sparse representations like Bag of Words
- ▶ Can be pre-built on generic corpus or built from scratch with own corpus

## Tensorflow/Keras Embedding Layer

- ▶ Requires that the input data be integer encoded, so that each word is represented by a unique integer. Tokenizer API can be used to generate this format.
- ▶ Like a lookup table that maps from integer indices (which stand for specific words) to dense vectors (their embeddings).
- ▶ Usage:
  - ▶ Used alone to learn a word embedding that can be saved and used in another model later.
  - ▶ Used as part of a deep learning model where the embedding is learned along with the model itself.
  - ▶ Used to load a pre-trained word embedding model, a type of transfer learning.

## Embedding Layer Input Specification

- ▶ **input\_dim**: Size of the vocabulary. For example, if your data is integer encoded to values between 0-10, then the size of the vocabulary would be 11 words.
- ▶ **output\_dim**: Size of the vector space in which words will be embedded, ie size of the output vectors from this layer for each word. For example, it could be 32 or 100 or even larger.
- ▶ **input\_length**: This is the length of input sequences, as you would define for any input layer of a Keras model. For example, if all of your input documents are comprised of 1000 words, this would be 1000.

```
e = Embedding(200, 32, input_length=50)
```

Above Embedding layer is with a vocabulary of 200 (e.g. integer encoded words from 0 to 199, inclusive), a vector space of 32 dimensions in which words will be embedded, and input documents that have 50 words each.

## Embedding Layer

```
1 # Embed a 1,000 word vocabulary into 5 dimensions.  
embedding_layer = tf.keras.layers.Embedding(1000, 5)
```

- ▶ Initially all 5000 weights are initialized randomly (just like any other layer).
- ▶ During training, they are gradually adjusted via backpropagation.
- ▶ Once trained, the learned word embeddings will roughly encode similarities between words (as they were learned for the specific problem your model is trained on).

(Ref: [https://www.tensorflow.org/tutorials/text/word\\_embeddings](https://www.tensorflow.org/tutorials/text/word_embeddings))

## Embedding Layer Output

- ▶ The output of the Embedding layer is a 2D vector with one embedding for each word in the input sequence of words (input document).
- ▶ If you wish to connect a Dense layer directly to an Embedding layer, you must first flatten the 2D output matrix to a 1D vector using the Flatten layer.

# Word Embeddings via Layers in Keras/Tensorflow

(Ref: How to Use Word Embedding Layers for Deep Learning with Keras by Jason Brownlee)

## Embedding Layer Example: Inputs

Sentiment analysis classification on 10 documents.

```
# define documents
2 docs = ['Well done!',
          'Good work',
          'Great effort',
          'nice work',
          'Excellent!',
          'Weak',
          'Poor effort!',
          'not good',
          'poor work',
          'Could have done better.']
12 # define class labels
labels = array([1,1,1,1,1,0,0,0,0,0])
```

## Embedding Layer Example: Encoding

Integer encoding. Estimating the vocabulary size of 50, which is much larger than needed, but that's ok.

Keras provides the `one_hot()` function that creates a hash of each word as an efficient integer encoding.

```
1 # integer encode the documents
  vocab_size = 50
3 encoded_docs = [one_hot(d, vocab_size) for d in docs]
  print(encoded_docs)
5
  [[6, 16], [42, 24], [2, 17], [42, 24], [18], [17], [22, 17], [27, 42], [22,
    24], [49, 46, 16, 34]]
```



## Embedding Layer Example: Padding

- ▶ The sequences have different lengths and Keras prefers inputs to be vectorized and all inputs to have the same length.
- ▶ Pad all input sequences to have the length of 4.
- ▶ Again, we can do this with a built in Keras function, in this case the `pad_sequences()` function.

```
# pad documents to a max length of 4 words
2 max_length = 4
  padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
4 print(padded_docs)

6 [[ 6 16  0  0]
   [42 24  0  0]
8  [ 2 17  0  0]
   [42 24  0  0]
10 [18  0  0  0]
   [17  0  0  0]
12 [22 17  0  0]
   [27 42  0  0]
14 [22 24  0  0]
   [49 46 16 34]]
```

## Embedding Layer Example: Model

The Embedding has a vocabulary of 50 and an input length of 4. Lets choose a small embedding space of 8 dimensions. The output from the Embedding layer will be 4 vectors of 8 dimensions each, one for each word. We flatten this to a one 32-element vector to pass on to the Dense output layer.

```

1 # define the model
model = Sequential()
3 model.add(Embedding(vocab_size, 8, input_length=max_length))
model.add(Flatten())
5 model.add(Dense(1, activation='sigmoid'))
# compile the model
7 model.compile(optimizer='adam', loss='binary_crossentropy',
               metrics=['accuracy'])
# summarize the model
9 print(model.summary())

```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 4, 8)	400
flatten_1 (Flatten)	(None, 32)	0
dense_1 (Dense)	(None, 1)	33

## Embedding Layer Example: Evaluation

```
# fit the model
2 model.fit(padded_docs, labels, epochs=50, verbose=0)
# evaluate the model
4 loss, accuracy = model.evaluate(padded_docs, labels, verbose=0)
print('Accuracy: %f' % (accuracy*100))
6
Accuracy: 100.000000
```

# Glove Keras/Tensorflow

(Ref: How to Use Word Embedding Layers for Deep Learning with Keras by Jason Brownlee)

## Glove Word Embedding

- ▶ Glove provides a suite of pre-trained word embeddings on their website released under a public domain license
- ▶ The smallest package of embeddings is 822Mb, called "glove.6B.zip". It was trained on a dataset of one billion tokens (words) with a vocabulary of 400 thousand words.
- ▶ There are a few different embedding vector sizes, including 50, 100, 200 and 300 dimensions.
- ▶ You can download this collection of embeddings and we can seed the Keras Embedding layer with weights from the pre-trained embedding for the words in your training dataset.

## Glove Embedding Example: Input

```
1 # define documents
docs = ['Well done!',
3       'Good work',
       'Great effort',
5       'nice work',
       'Excellent!',
7       'Weak',
       'Poor effort!',
9       'not good',
       'poor work',
11      'Could have done better.'],

# define class labels
13 labels = array([1,1,1,1,1,0,0,0,0,0])
# prepare tokenizer
15 t = Tokenizer()
t.fit_on_texts(docs)
17 vocab_size = len(t.word_index) + 1
# integer encode the documents
19 encoded_docs = t.texts_to_sequences(docs)
print(encoded_docs)
21 # pad documents to a max length of 4 words
max_length = 4
23 padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs)
```

## Glove Embedding Example: Load Glove

Load the entire GloVe word embedding file into memory as a dictionary of word to embedding array.

```
# load the whole embedding into memory
2 embeddings_index = dict()
f = open('glove.6B.100d.txt')
4 for line in f:
    values = line.split()
    6 word = values[0]
    coefs = asarray(values[1:], dtype='float32')
    8 embeddings_index[word] = coefs
f.close()
10 print('Loaded %s word vectors.' % len(embeddings_index))
```

## Glove Embedding Example: Prep Input

Create a matrix of one embedding for each word in the training dataset. We can do that by enumerating all unique words in the `Tokenizer.word_index` and locating the embedding weight vector from the loaded GloVe embedding.

```
# create a weight matrix for words in training docs
2 embedding_matrix = zeros((vocab_size, 100))
for word, i in t.word_index.items():
4     embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
6         embedding_matrix[i] = embedding_vector
```

The result is a matrix of weights only for words we will see during training.



## Glove Embedding Example: Embedding Layer

The embedding layer can be seeded with the GloVe word embedding weights. We chose the 100-dimensional version, therefore the Embedding layer must be defined with `output_dim` set to 100. Finally, we do not want to update the learned word weights in this model, therefore we will set the trainable attribute for the model to be False.

```
# define model
2 model = Sequential()
  e = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=4,
    trainable=False)
4 model.add(e)
  model.add(Flatten())
6 model.add(Dense(1, activation='sigmoid'))
  # compile the model
8 model.compile(optimizer='adam', loss='binary_crossentropy',
    metrics=['accuracy'])
  # summarize the model
10 print(model.summary())
  # fit the model
12 model.fit(padded_docs, labels, epochs=50, verbose=0)
  # evaluate the model
14 loss, accuracy = model.evaluate(padded_docs, labels, verbose=0)
  print('Accuracy: %f' % (accuracy*100))
```

## Summary

- ▶ Keras supports word embeddings via the Embedding layer.
- ▶ How to learn a word embedding while fitting a neural network.
- ▶ How to use a pre-trained word embedding in a neural network.

Thanks ... [yogeshkulkarni@yahoo.com](mailto:yogeshkulkarni@yahoo.com)