CS 563: Word Embedding

Outline

1. Why word representation?
2. Non semantic word representations
    a. One-hot vector representation
3. Semantic word representation
    a. Distributional hypothesis
    b. Co-occurrence matrix based representation
    c. Language model
    d. FFNN language model
    e. Skip-gram model
    f. Continuous Bag of Words model (CBoW)

Why word representation?

Definition : Word (Oxford Dictionary)

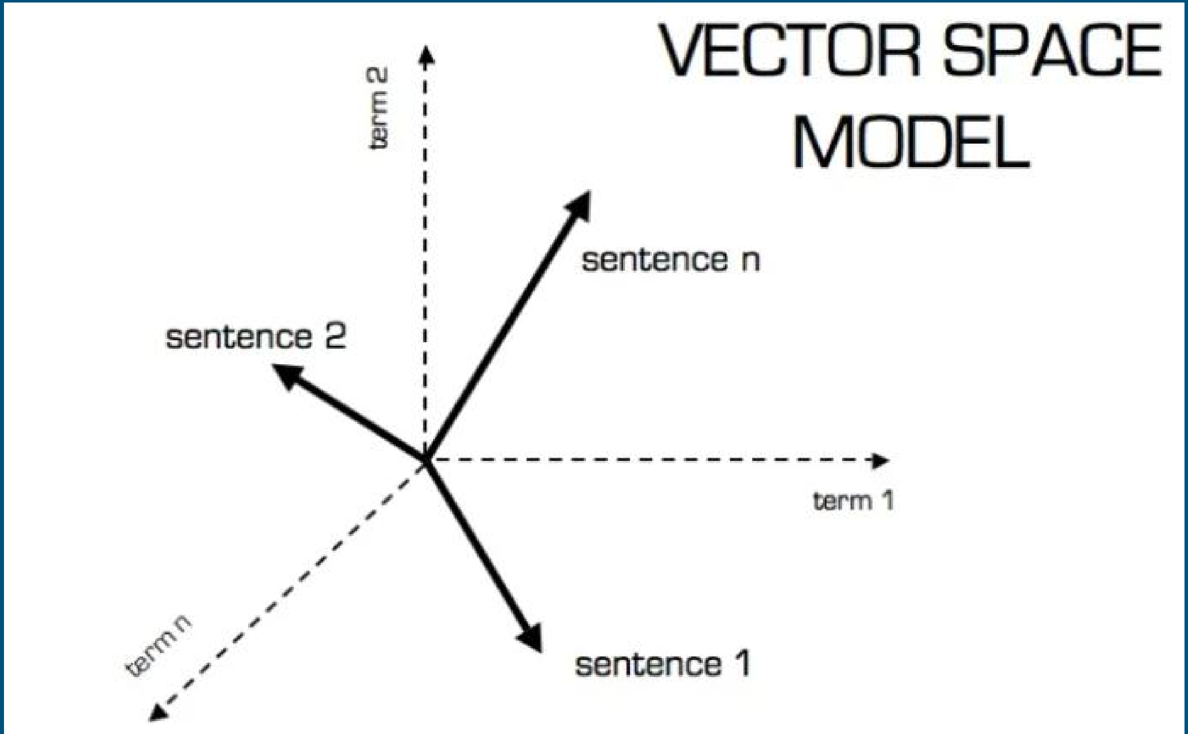
A word is a single distinct meaningful element of speech or writing, used with others (or sometimes alone) to form a sentence

- Words are stitched together to form a sentence
- Proper representation of words is essential for text representation

Vector Space Model

- Texts are represented as vectors of numbers instead of original textual representation
- Many approaches to VSM

Vector Space Model



Non-semantic word representation

The vast majority of rule-based and statistical NLP work regards words as atomic symbols

One-hot vector representation of words:

- Assign a unique id to each unique word in the corpus
- Convert these unique ids to one-hot vectors

Non-semantic word representation

- **Sentence:** RMS Titanic was a British passenger liner
- **Unique Ids:** [1, 2, 3, 4, 5, 6, 7]
- **One-hot representation**: [[1,0,0,0,0,0,0], [0,1,0,0,0,0,0], [0,0,1,0,0,0,0], [0,0,0,1,0,0,0], [0,0,0,0,1,0,0], [0,0,0,0,0,1,0], [0,0,0,0,0,0,1]]

Non-semantic word representation (continued...)

Python Code for categorical (one-hot) representation

```
from keras.utils import to_categorical
txt = "RMS Titanic was a British passenger liner that sank in the North Atlantic Ocean in 1912 after striking an iceberg during her maiden voyage from Southampton to New York City"
txt_list = txt.split()
word2id = {}
for i,j in enumerate(list(set(txt_list))):
  word2id[j] = i
txt_index = [word2id[i] for i in txt_list]
txt_one_hot = to_categorical(txt_index)
```

Non-semantic word representation (continued...)

Drawbacks of categorical representation:

- No semantics captured
- All the words are equally different from each other
  - The euclidean distance between any two words is 1.41 units
  - The cosine similarity between any two words is 0
- Curse of dimensionality (the length of the vector depends on the number of words in the corpus)
- The vectors formed are sparse

Semantic word representation

We can get a lot of value by representing a word by means of its neighbors:

“You shall know a word by the company it keeps” **(J. R. Firth 1957: 11)**

Built in Belfast, Ireland, in the United Kingdom the RMS **Titanic** was the second of the three Olympic-class ocean liners.

According to distributional hypothesis, all these words play a role in representing the meaning of the word **Titanic**

Semantic word representation (continued...)

Using co-occurrence matrix to make neighbours represent words.

- Window based co-occurrence matrix captures syntactic (POS) and semantic information
- The matrix is symmetric, i.e. an occurrence is counted irrespective of left or right context
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

Semantic word representation (continued...)

Co-occurrence matrix example -

- Window size = 1

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Semantic word representation (continued...)

Co-occurrence matrix example -

https://colab.research.google.com/drive/10XCsBjW88b9pYiLgWADxVaDLhZHhSeVV

Semantic word representation (continued...)

Code for co-occurence matrix creation:

```
import pandas as pd
import numpy as np
from collections import defaultdict
def co_occurrence(sentences, window_size):
    d = defaultdict(int)
    vocab = set()
    for text in sentences:
        text = text.lower().split()
        # iterate over sentences
        for i in range(len(text)):
            token = text[i]
            vocab.add(token)  # add to vocab
            next_token = text[i+1 : i+1+window_size]
```

Semantic word representation (continued...)

Code for co-occurence matrix creation:

```
for t in next_token:
                key = tuple( sorted([t, token]) )
                d[key] += 1
    # formulate the dictionary into dataframe
    vocab = sorted(vocab) # sort vocab
    df = pd.DataFrame(data=np.zeros((len(vocab), len(vocab)), dtype=np.int16),
                      index=vocab,
                      columns=vocab)
    for key, value in d.items():
        df.at[key[0], key[1]] = value
        df.at[key[1], key[0]] = value
    return df
docs = ["I like deep learning", "I enjoy NLP", "I enjoy flying"]
co_occurrence(docs, window_size=1)
```

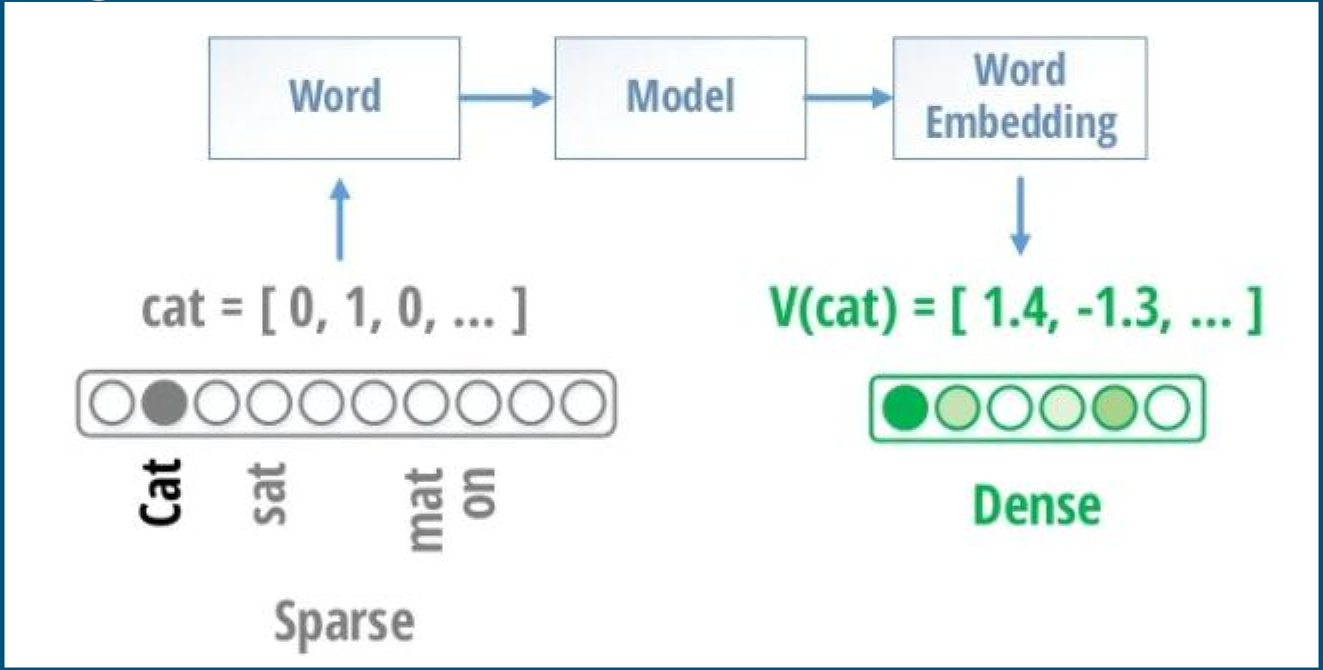
Semantic word representation (continued...)

Problems with simple co-occurrence vectors:

- Increases in size with vocabulary
- Sparsity issue persists
- Very high dimensional: requires a lot of storage

Semantic word representation (continued...)

Embeddings: Dense semantic word representation



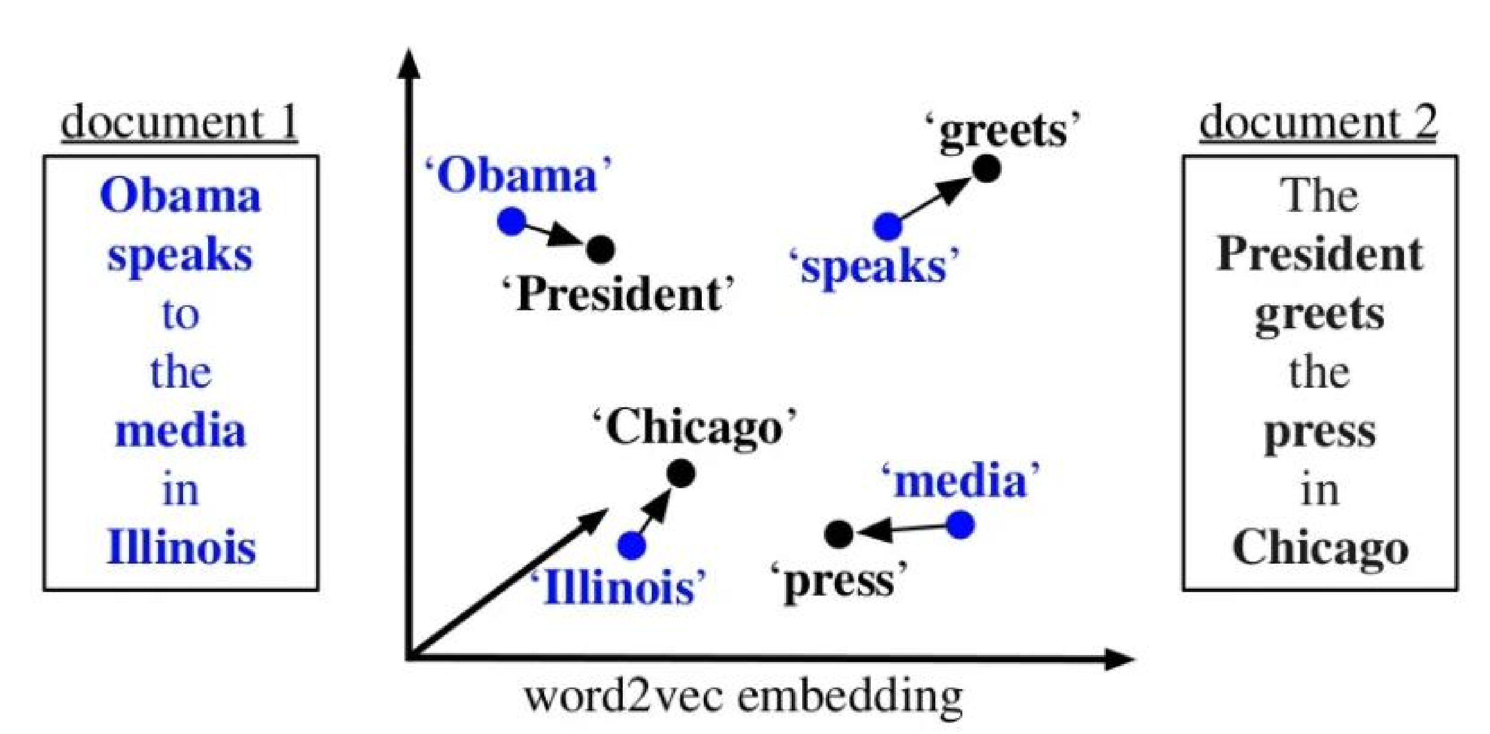
Semantic word representation (continued...)

Embedding Properties: Word analogies

$$\vec{w}_{king} - \vec{w}_{man} + \vec{w}_{woman} \approx \vec{w}_{queen}$$

Semantic word representation (continued…)

Embedding Properties: Able to capture semantic similarity even when no words match



Semantic word representation (continued...)

Word2Vec models

Skip-gram Model:

This is one of the methods used for the creation of Word2Vec word embeddings

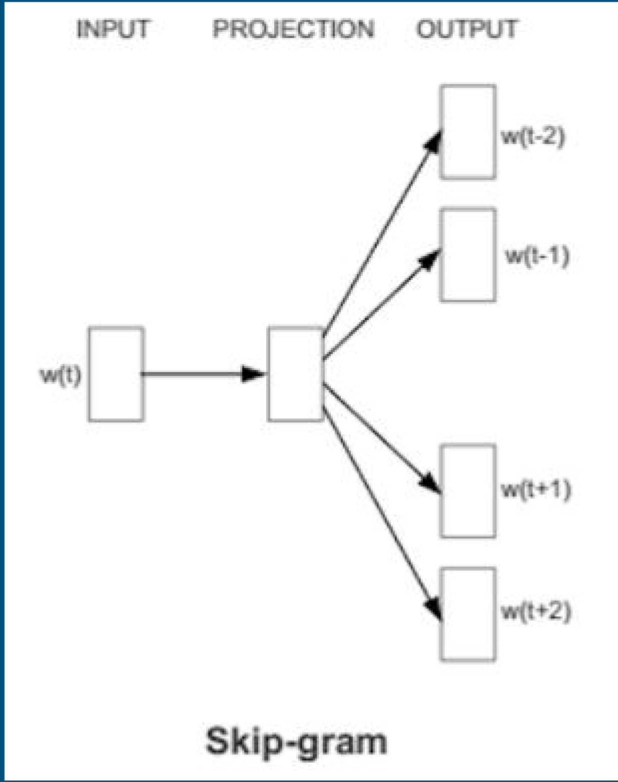
Main ideas behind this method

- Instead of capturing co-occurrence counts directly, predict surrounding words for every word
- Predict surrounding words in a window of length *m* for every word
- Objective function: Maximize the log probability of any context word given the current center word:

$$\text{minimize } J = -\log P(w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m} | w_c)$$

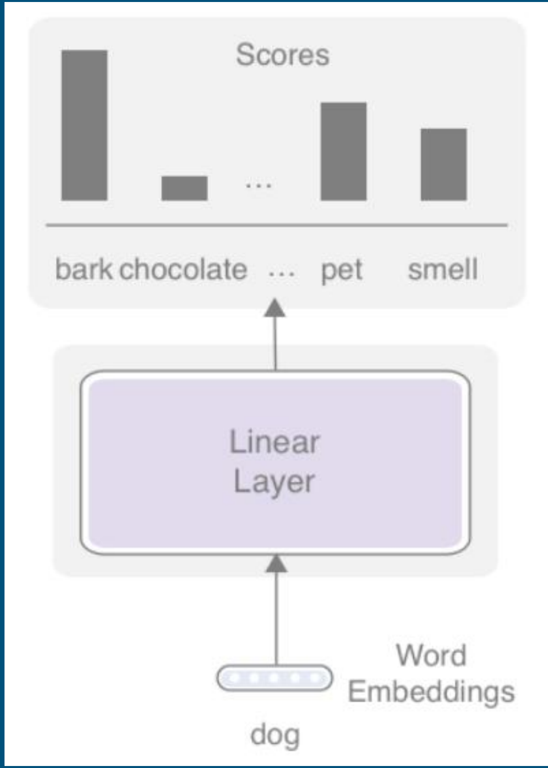
Semantic word representation (continued...)

Skip-gram Model:



Semantic word representation (continued...)

Skip-gram Model:



Semantic word representation

Word2Vec models

Continuous Bag of Words Model:

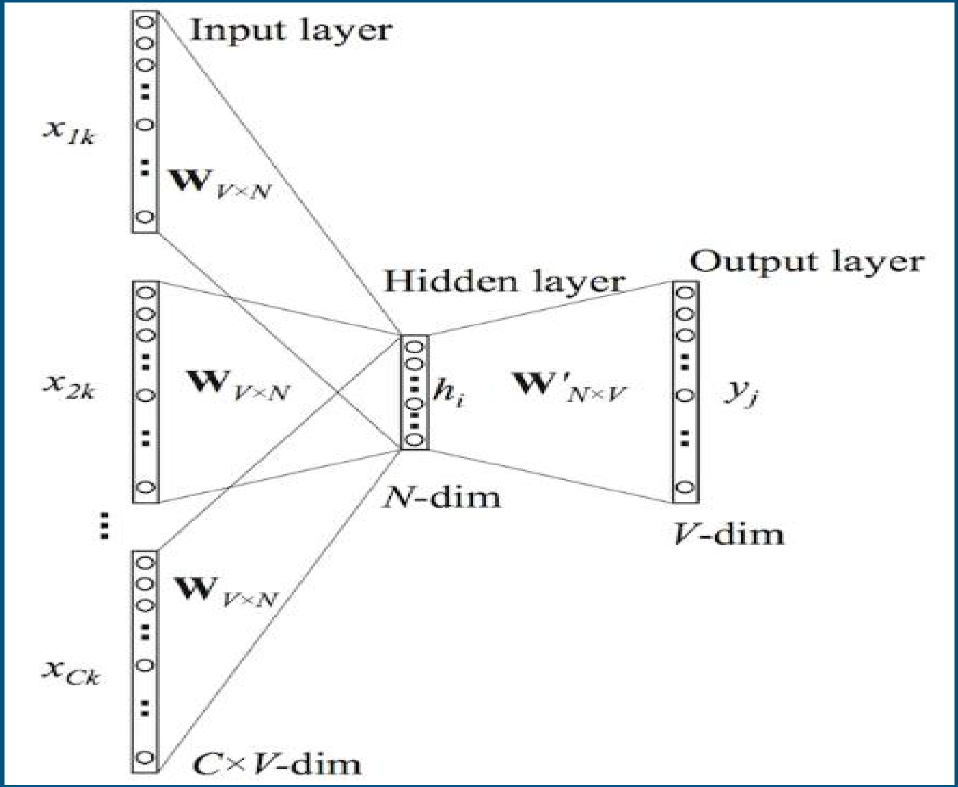
This is another method for creation of Word2Vec word embeddings

Main ideas behind this method

- Predict the current word based on other words in the context window *m*
- Objective function: Maximize the log probability of the current word given the context words

$$\text{minimize } J = -\log P(w_c | w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m})$$

Semantic word representation (continued...)



Semantic word representation (continued…)

Code for word embedding creation:

```
from gensim.models import Word2Vec

sentences = [['this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec'],
      ['this', 'is', 'the', 'second', 'sentence'],
      ['yet', 'another', 'sentence'],
      ['one', 'more', 'sentence'],
      ['and', 'the', 'final', 'sentence']]
# train model
model = Word2Vec(sentences, min_count=1, size=300, sg=0) #sg ({0, 1}, optional) – Training algorithm:
1 for skip-gram; otherwise CBOW.

print(model)
# summarize vocabulary
words = list(model.wv.vocab)
print(words)
# access vector for one word
print(model['sentence'])
```

Semantic word representation (continued…)

Code for word embedding creation:

```
model['this'].size

# save model
model.save('model.bin')
# load model
new_model = Word2Vec.load('model.bin')
print(new_model)
```

Semantic word representation (continued…)

Word2Vec demo:

```
from gensim.test.utils import common_texts, get_tmpfile
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import numpy as np

def cos(x1, x2):
  return np.dot(x1, x2)/(np.linalg.norm(x1)*np.linalg.norm(x2))

!wget -P /root/input/ -c "https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz"
EMBEDDING_FILE = '/root/input/GoogleNews-vectors-negative300.bin.gz' # from above
word2vec = KeyedVectors.load_word2vec_format(EMBEDDING_FILE, binary=True)
print(word2vec["cat"].shape)
print(cos(word2vec['cat'],word2vec['purr']))

print(word2vec.similar_by_vector(word2vec["cat"], topn=10, restrict_vocab=None))
```

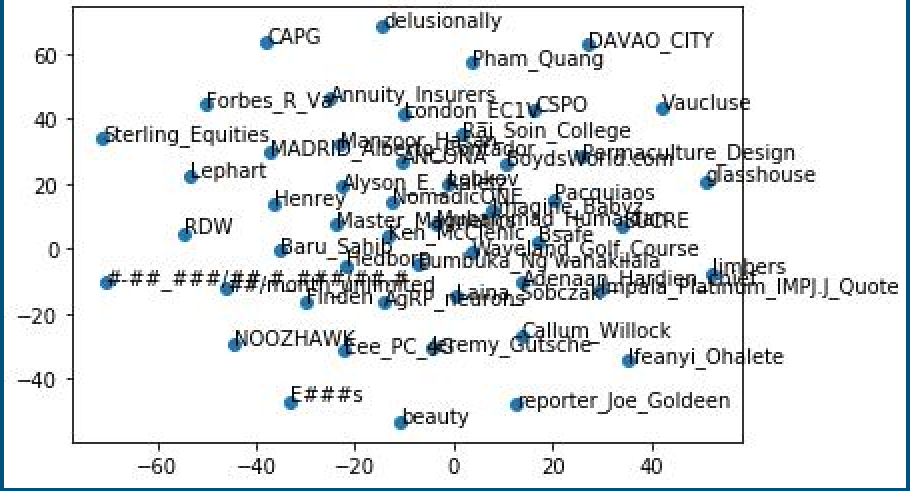
Semantic word representation (continued…)

Word2Vec demo:

Plotting word vectors:

```
import random
vocab = random.sample(list(word2vec.vocab), 50)
X = np.array([word2vec[v] for v in vocab])
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, random_state=0)
np.set_printoptions(suppress=True)
Y = tsne.fit_transform(X)

plt.scatter(Y[:, 0], Y[:, 1])
for label, x, y in zip(vocab, Y[:, 0], Y[:, 1]):
    plt.annotate(label, xy=(x, y), xytext=(0, 0), textcoords='offset points')
plt.show()
```



Thank YOU

Important Links

- https://machinelearningmastery.com/what-are-word-embeddings/
- https://neptune.ai/blog/word-embeddings-guide
- https://www.turing.com/kb/guide-on-word-embeddings-in-nlp
- https://ruder.io/word-embeddings-1/