Word embedding or word representation

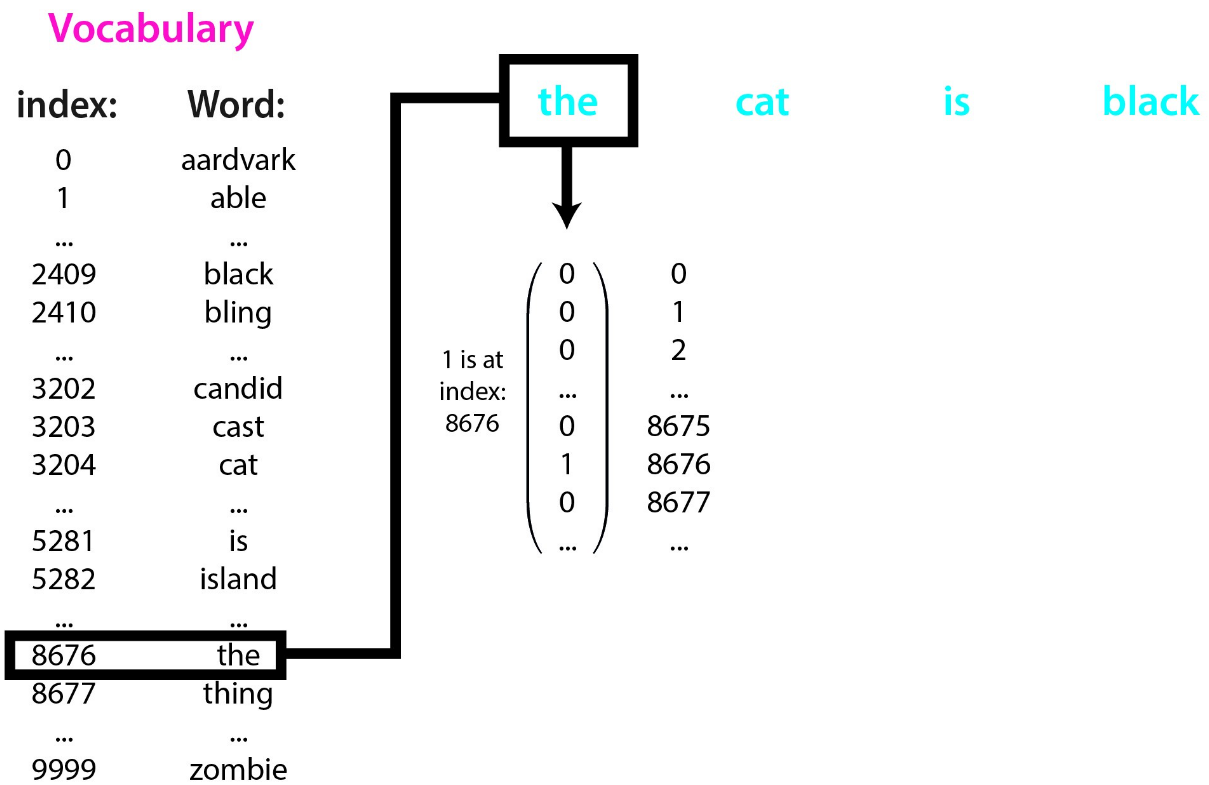
Michel RIVEILL

Michel.RIVEILL@univ-cotedazur.fr

Represent a word by an Integer

First idea

- Step 1 : build a dictionary
- Step 2 : replace each word by it's rank inside the dictionary
- Step 3 : One Hot Encode each word



How to represent a word

- First idea
  - Build a dictionary
    - For each word, we obtain a number
  - One hot encode each word
- Not really a good idea
  - Very sparse representation,
  - Very large representation vector representation = vocabulary size
  - Not express similarity between different word
    - Cosine similarity (x, y) = $\frac{x^T y}{|x||y|} \in [-1, 1]$
    - With one hot encoding, similarity is always equal to 0

- The objective of the vectorization approach is to try to grasp the similarity and analogy relationships between different words.

From sparse vector to dense vector

- We take 5 words from our vocabulary (“aardvark”, “black”, “cat”, “duvet” and “zombie”)
- Their embedding vectors created by the one-hot encoding method look like:

sparse one-hot
encoding of words

aardvark	1	0	0	...	0	0	0
black	0	0	...	1	...	0	0
cat	0	0	...	1	...	0	0
duvet	0	0	...	1	...	0	0
zombie	0	0	0	...	0	0	1

An aardwark

- These vectors can be used to represent a word but do not carry any meaning.
- Whatever the 2 words chosen, whatever similarity we choose:
  - similarity(W_1, W_2) = 0

From sparse vector to dense vector

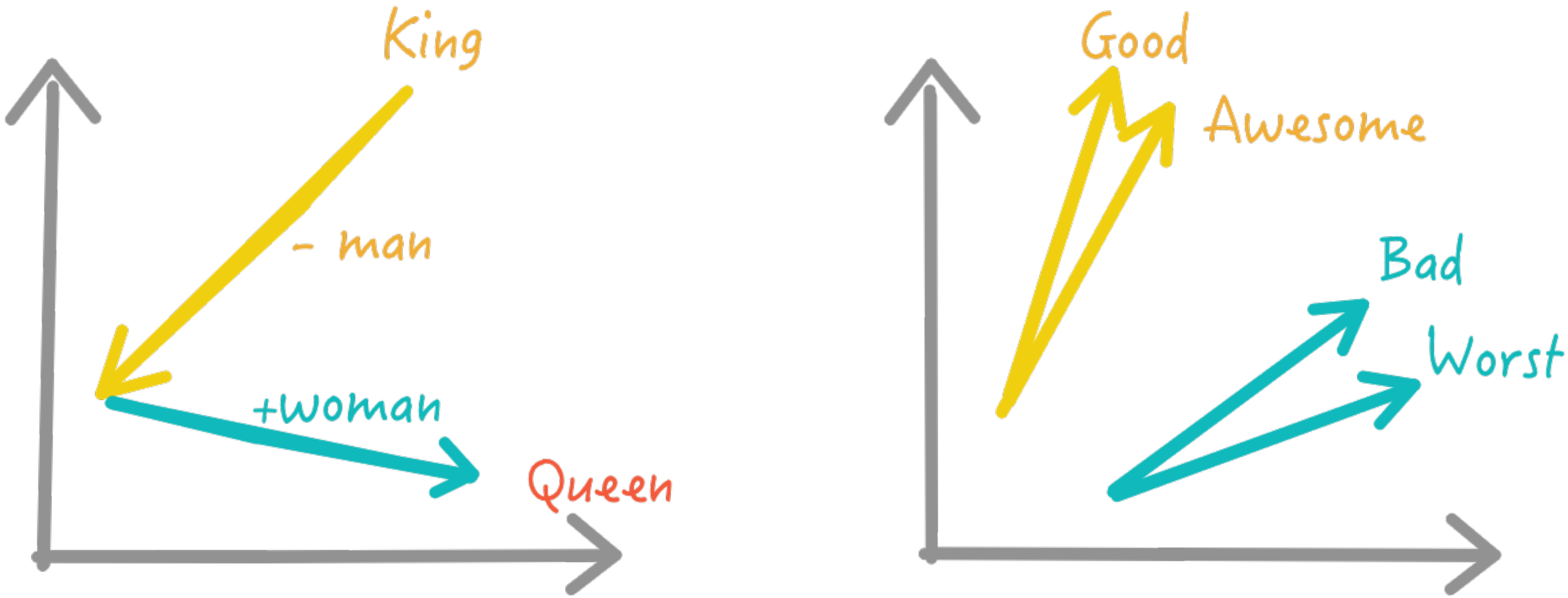
- We know that words are these rich entities with many layers of connotation and meaning.
- Let's hand-craft some semantic features for these 5 words.
- Specifically, let's represent each word as having some sort of value between 0 and 1 for four semantic qualities,:
  - "animal", "fluffiness (duveteux)", "dangerous", and "spooky (effrayant)":

	animal	fluffiness	dangerous	spooky
aardvark	0.97	0.03	0.15	0.04
black	0.07	0.01	0.20	0.95
cat	0.98	0.98	0.45	0.35
duvet	0.01	0.84	0.12	0.02
zombie	0.74	0.05	0.98	0.93

An aardwark

From sparse vector to dense vector

- The process to transform word to vectors are called word embeddings or word representations
- Main properties searched

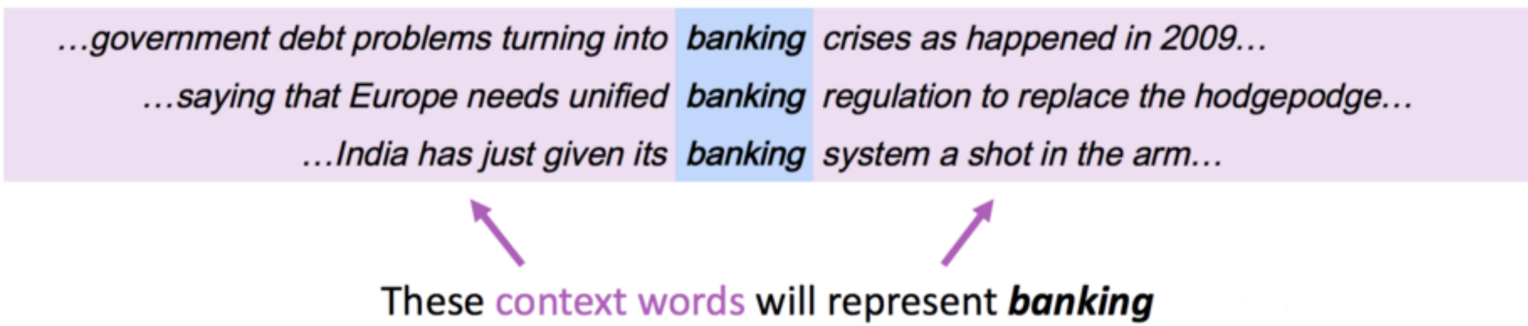


a) Learns Analogy

b) Similar Words have same angles

Representing words by their context

- Core idea:
  - A word's meaning is given by the words that frequently appear close-by
  - One of the most successful ideas of modern statistical NLP!
- When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of w to build up a representation of w



Word Representations

Traditional Method - Bag of Words Model + OneHot representation	Word Embeddings
• Uses one hot encoding • Each word in the vocabulary is represented by one bit position in a HUGE vector. • For example, if we have a vocabulary of 10000 words, and "Hello" is the 4th word in the dictionary, it would be represented by: 0 0 0 1 0 0 . . . . 0 0 0 0 • Context information is not utilized	• Stores each word in as a point in space, where it is 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] • Dimensions are basically projections along different axes, more of a mathematical concept.

How to build these magic vectors…

1. How do you build these super-intelligent vectors, which seem to have such magical powers?
2. How do you use these vectors to find the friends of a word?

- Let's start by looking at the best known methods for constructing such vector representations of low-dimensional words according to their context
  - Co-occurrence matrix with the SVD
  - Keras embedding layers
  - Word2vec approach

- But, we already know a solution:
  - Keras' Embedding layer can transform an integer into a vector of a given size specifically trained for a given problem on the vocabulary of the train set.

Co-occurrence Matrix with Singular Value Decomposition

Build dense vector based on co-occurrence matrix

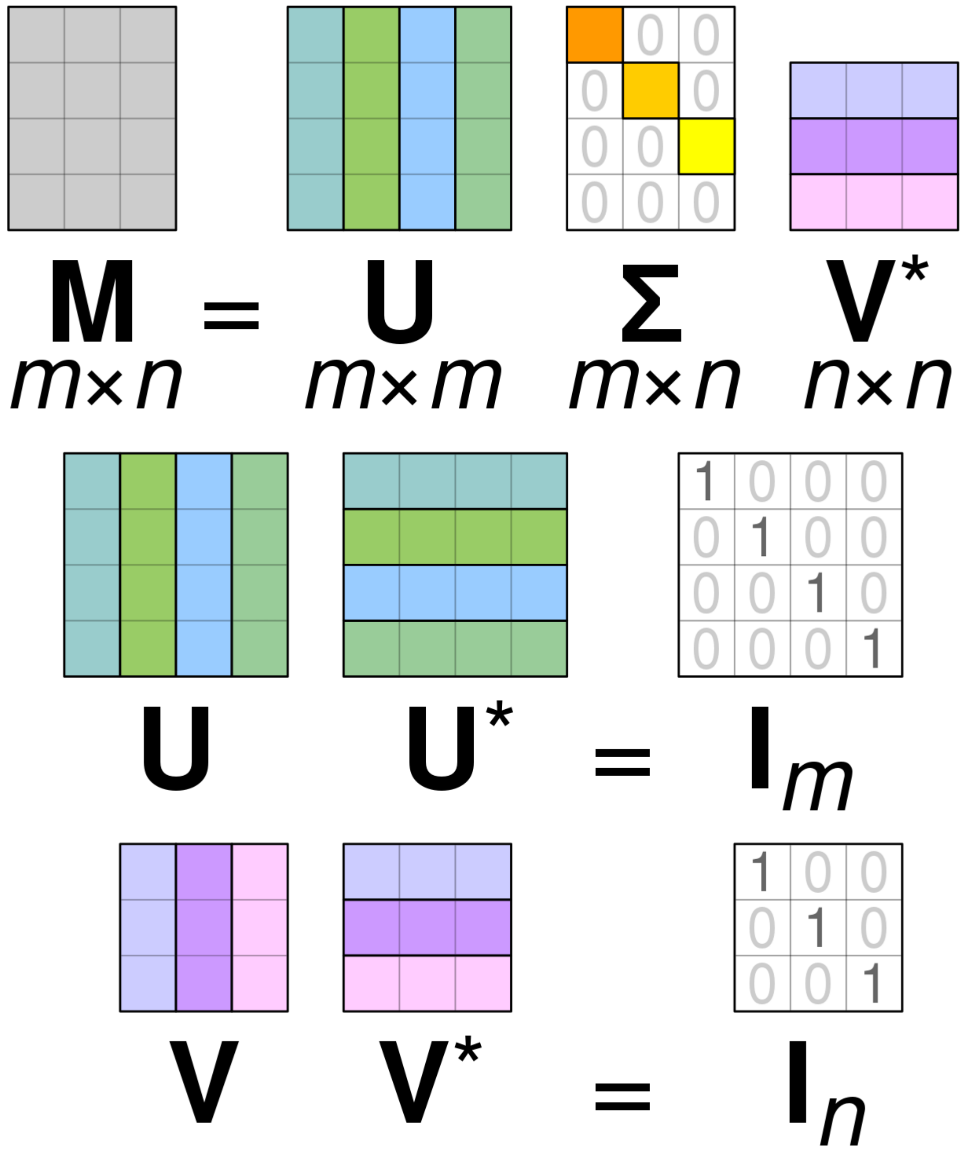
- A toy example:
  - Corpus = ["I like deep learning.", "I like NLP. ", "I enjoy flying. "]
- The co-occurrence matrix : put +1 if the line word is before/after the column word

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

- It is possible to encode a word using this co-occurrence matrix.
  - For example the vector of "like" is [2, 0, 0, 1, 0, 1, 0, 0]
- But the size of a vector is equal to the size of the vocabulary.

- To reduce the size of the vocabulary,
  - an SVD decomposition or a PCA approach can be used.

SVD decomposition



co-occurrence matrix is symmetric

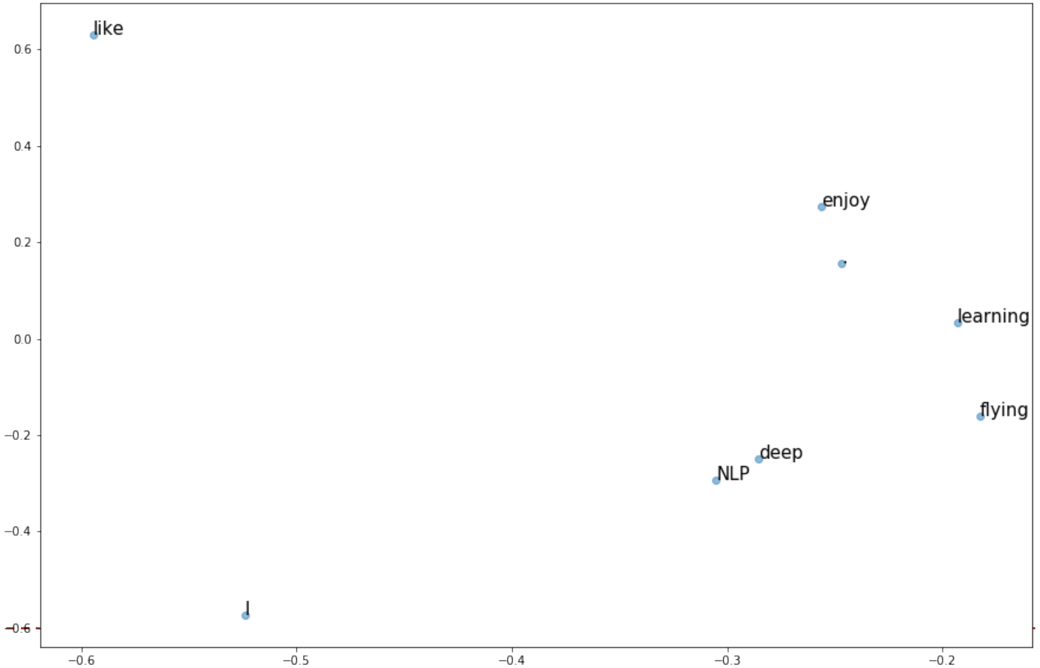
→ m=n

→ |U| = |V*|

SVD decomposition of the co-occurrence matrix

- The co-occurrence matrix is symmetrical
  - Each of the elements of U and V have the same absolute value
  - The value of Σ are singular values sorted in descending order
  - The columns $u_1,\ldots,u_m$ of U form an orthonormal base

  - We can use the first columns of U in order to represent each word
    - i.e. The part with the most important Eighen value
  - In our toy example



With Keras embedding layer

In Keras:
Use Tokenizer in order to transform a word an Integer

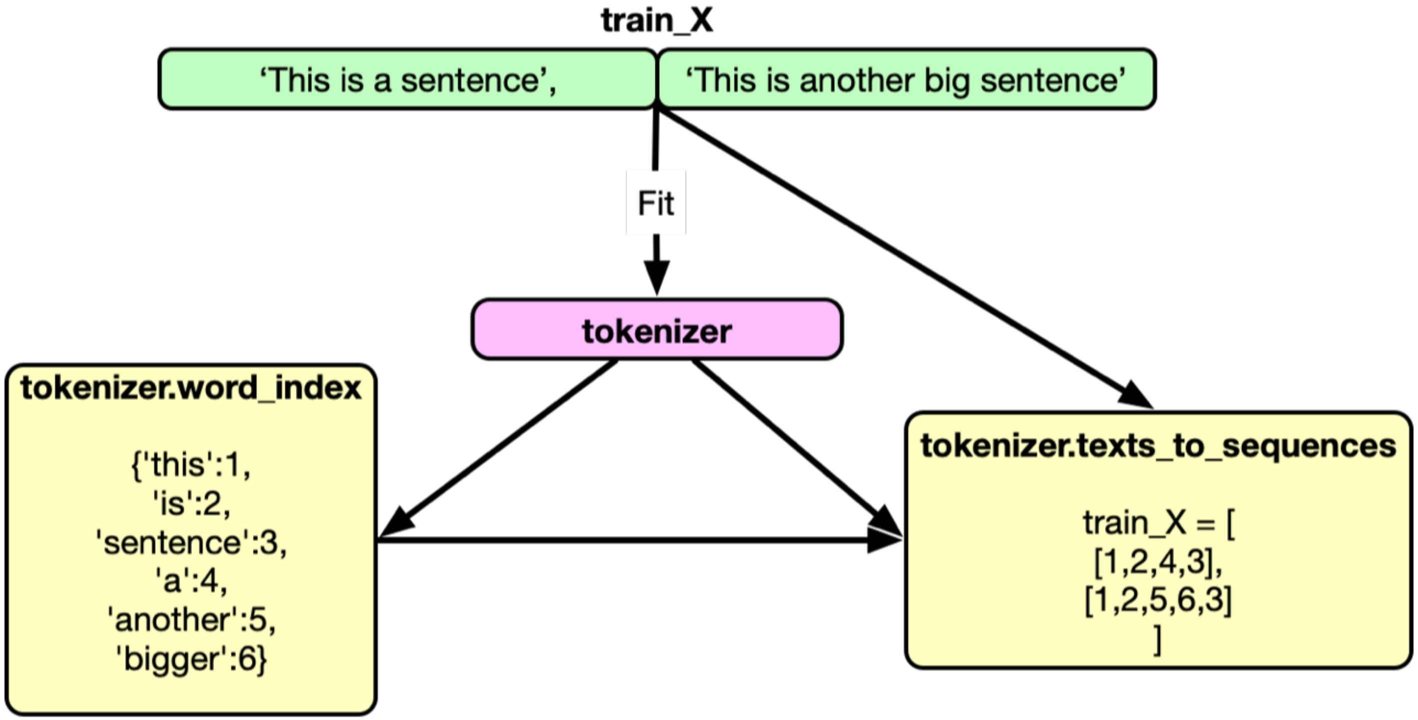
- From keras.preprocessing.text import Tokenizer

- This class allows to vectorize a text corpus, by turning each text into
  - a sequence of integers: each integer being the index of a token in a dictionary
  - or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf...
  - **num_words**: The maximum number of words to keep, based on word frequency. Only the most common num_words-1 words will be kept.
  - **filters**: A string where each element is a character that will be removed from the texts.
  - **lower**: convert the texts to lowercase.
  - **split**: Separator for word splitting.
  - **oov_token**:
    - if given, it will be added to word_index and used to replace out-of-vocabulary words during text_to_sequence calls
    - By default, oov_token = 0

- Stage 1: fit
- Stage 2: texts_to_sequences
  - Transform text to a sequence of words
  - Use **text_to_word_sequence**

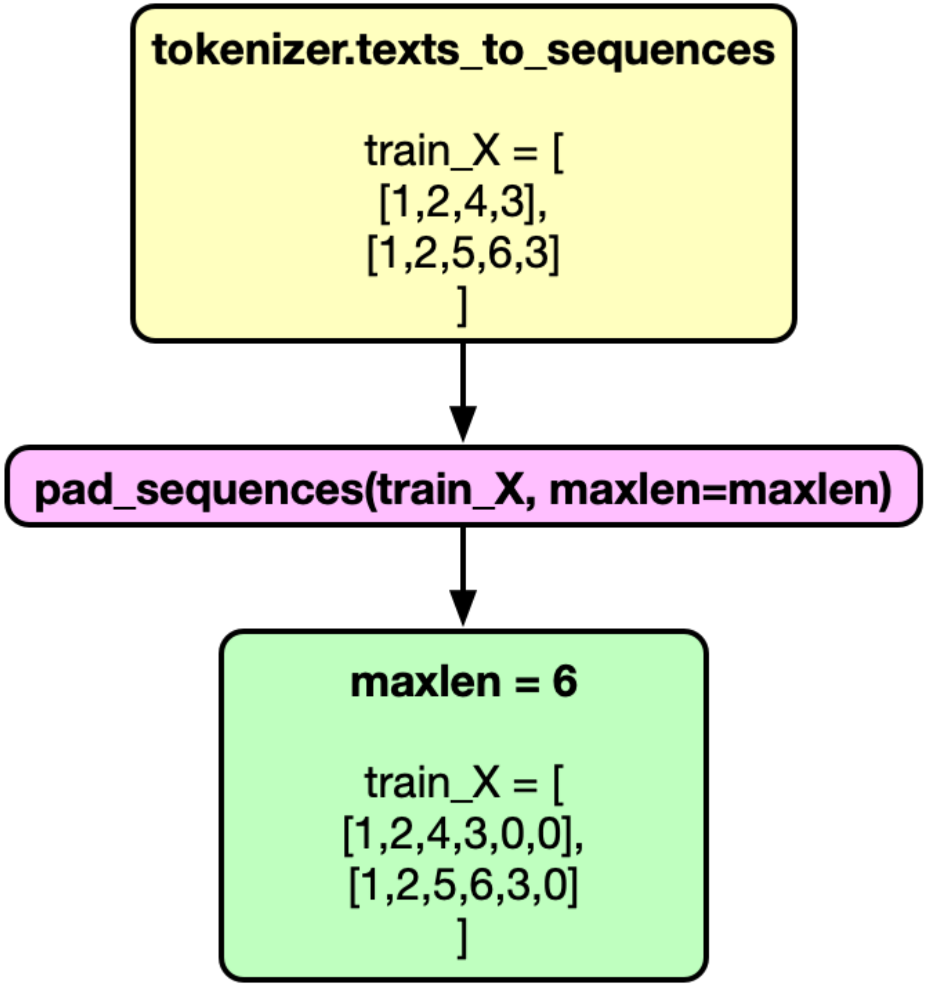
- Eventually: apply padding/truncating

Keras preprocessing from word to index



from keras.preprocessing.text import Tokenizer
Tokenize the sentences
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(train_X~~)+list(test_X)~~)
train_X = tokenizer.texts_to_sequences(train_X) test_X = tokenizer.texts_to_sequences(test_X)

Keras preprocessing padding




```
train_X = pad_sequences(train_X, maxlen=maxlen)
test_X = pad_sequences(test_X, maxlen=maxlen)
```

One Hot Encoding in Keras:

- Use Embedding layer in order to OneHotEncode a sequence of Integer

- from keras.layers import Input, Embedding

- inputs = Input(shape=(SEQUENCE_SIZE,))
- embedding = Embedding(vocabulary_size,
  EMBEDDING_SIZE,
  input_length=SEQUENCE_SIZE)(inputs)

- Layer (type) Output Shape Param #

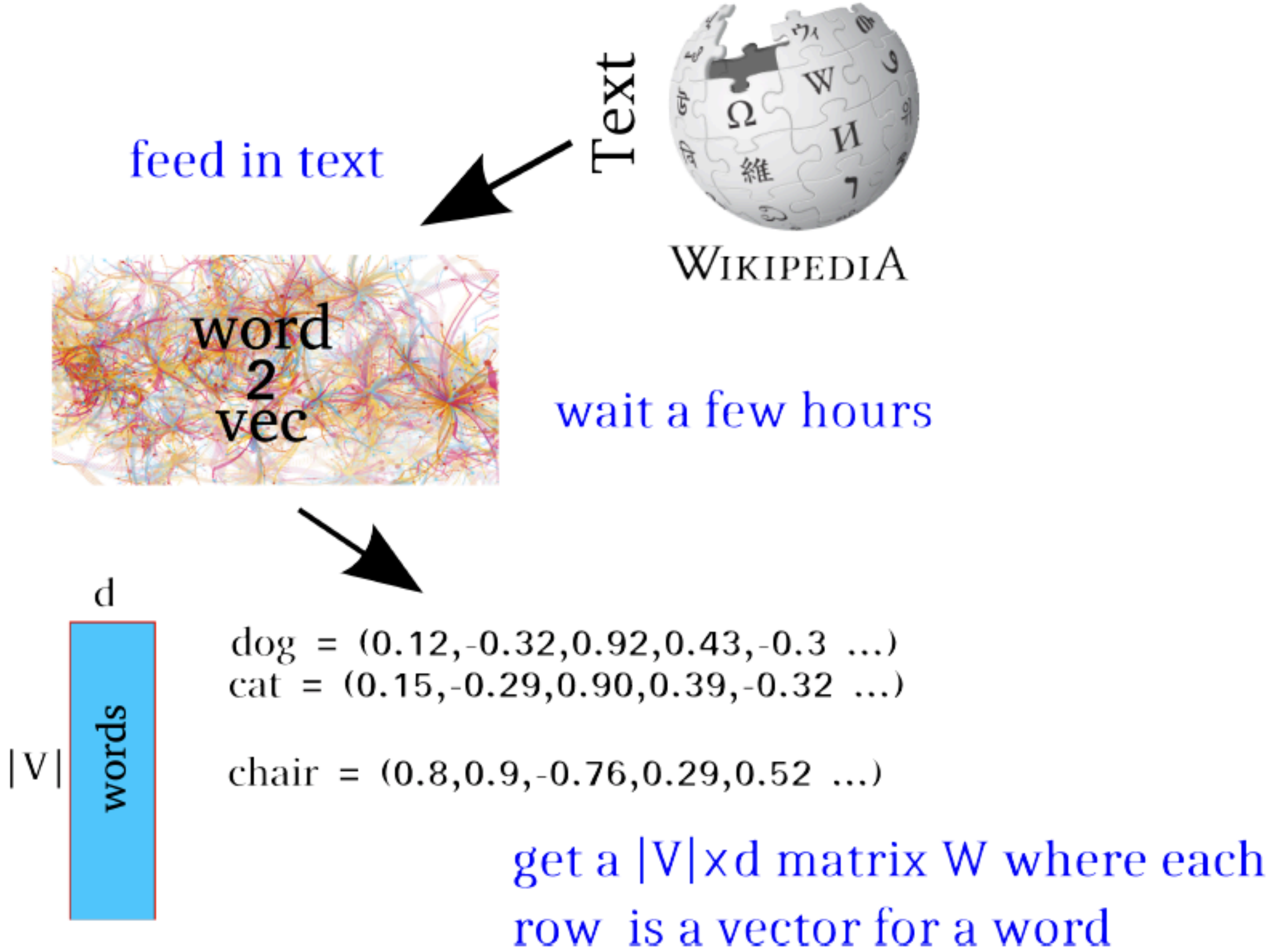
```
Layer (type)                 Output Shape          Param #
=====================================================================
input (InputLayer)           (None, 100)           0
_____________________________________________________________________
embedding (Embedding)        (None, 100, 300)      600000
```

- In this example
  - SEQUENCE_SIZE = 100
  - EMBEDDING_SIZE = 300

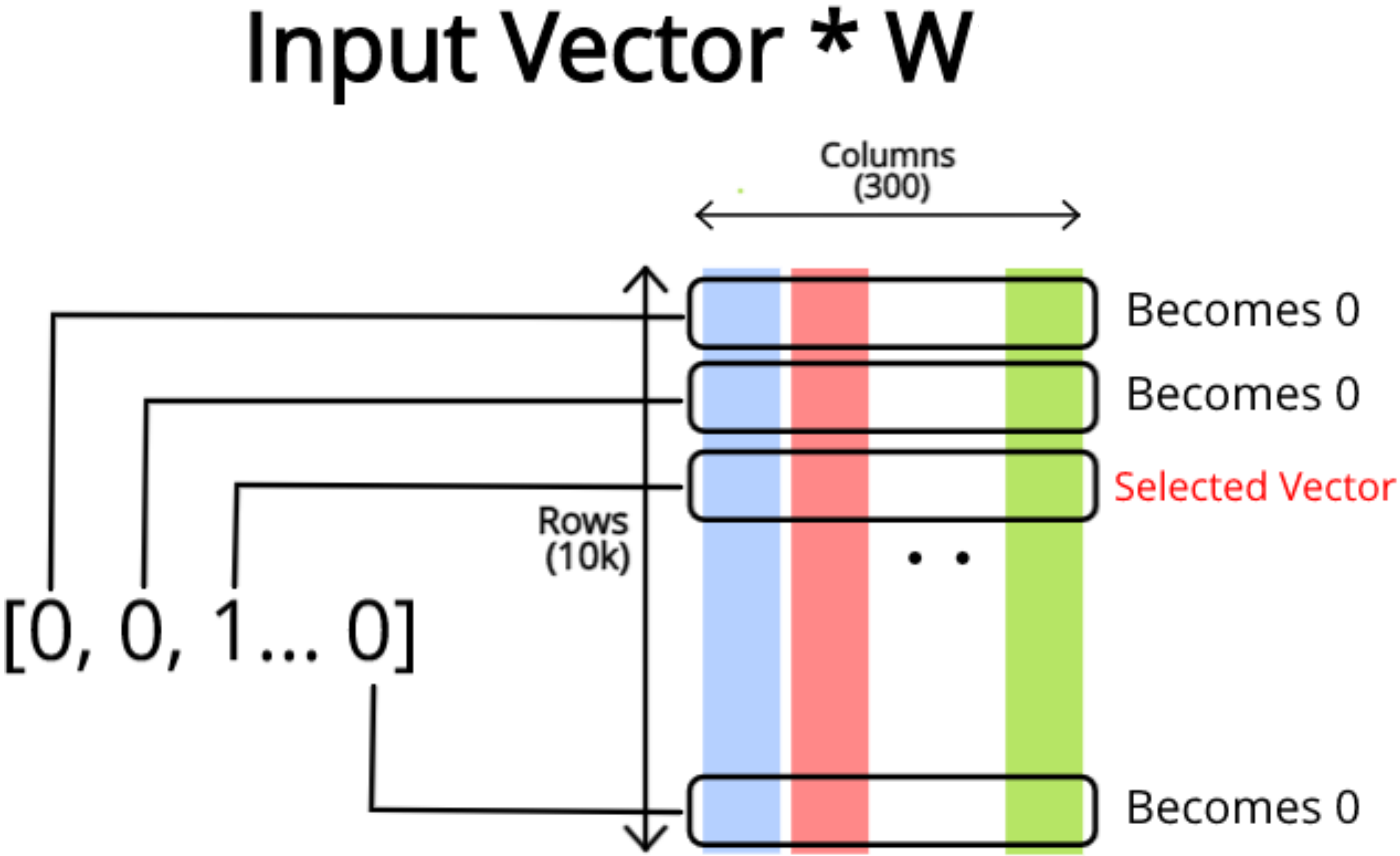
Word2vec
Use pretrained embedding

(Mikolov et al. 2013) is a framework for learning word vectors

word2vec


Text
WIKIPEDIA
feed in text
word
2
vec
wait a few hours
d
|V|
words
dog = (0.12,-0.32,0.92,0.43,-0.3 ...)
cat = (0.15,-0.29,0.90,0.39,-0.32 ...)
chair = (0.8,0.9,-0.76,0.29,0.52 ...)
get a |V|xd matrix W where each
row is a vector for a word

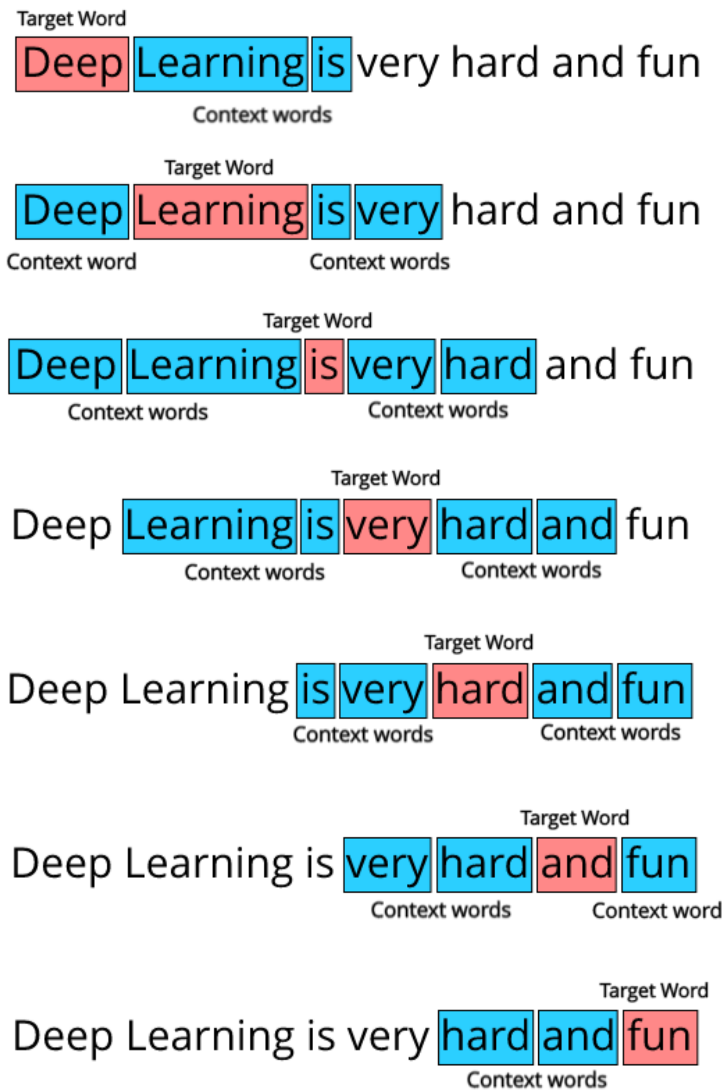
Word2vec


Input Vector * W
Columns
(300)
Rows
(10k)
[0, 0, 1... 0]
Becomes 0
Becomes 0
Selected Vector
Becomes 0

Word2vec family

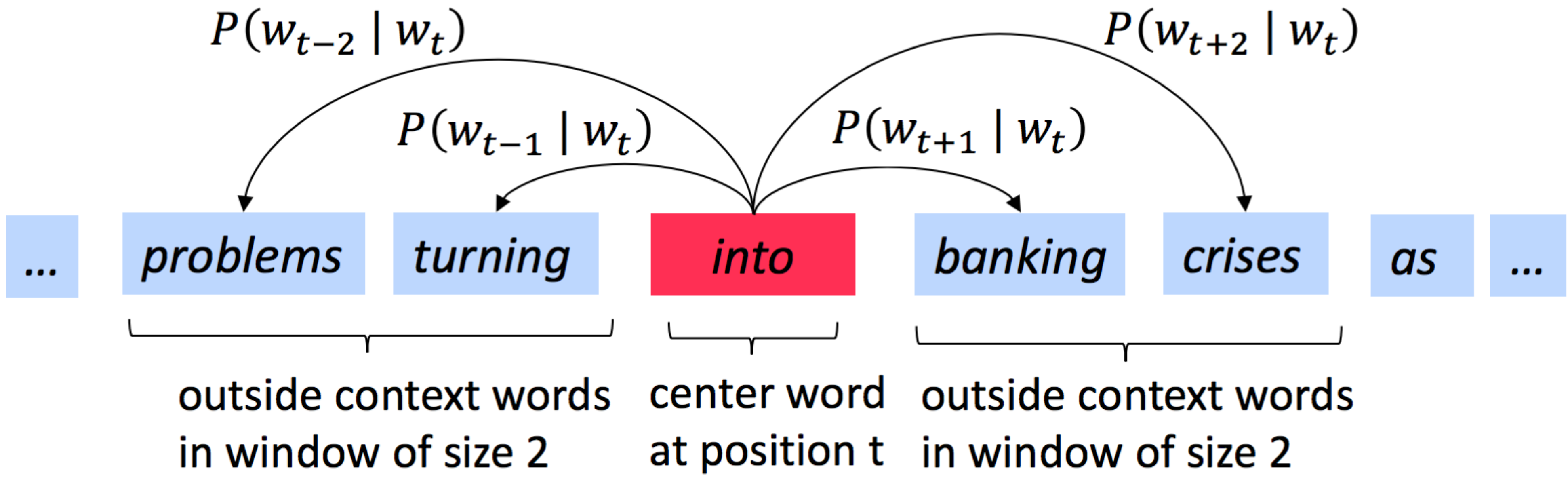
- Word2vec is a group of related algorithms
- Word2vec models are two-layer neural networks
  - Trained on very large corpus (not on the train set)
  - Produce a vector space (dimension 50, 100, 150, 300)
  - Try to capture the linguistic contexts of words.
- Word2vec associates
  - For each word in the corpus a corresponding vector in space.
  - Words that share common contexts in the corpus are located close to each other in space.
- Word2vec was created and published in 2013 by a team of researchers led by Tomas Mikolov at Google and patented.
- The algorithm was then analyzed and explained by other researchers.
- The incorporation of vectors created using the Word2vec algorithm has many advantages over previous algorithms, such as co-occurrence matrices.

What is the context of a word?

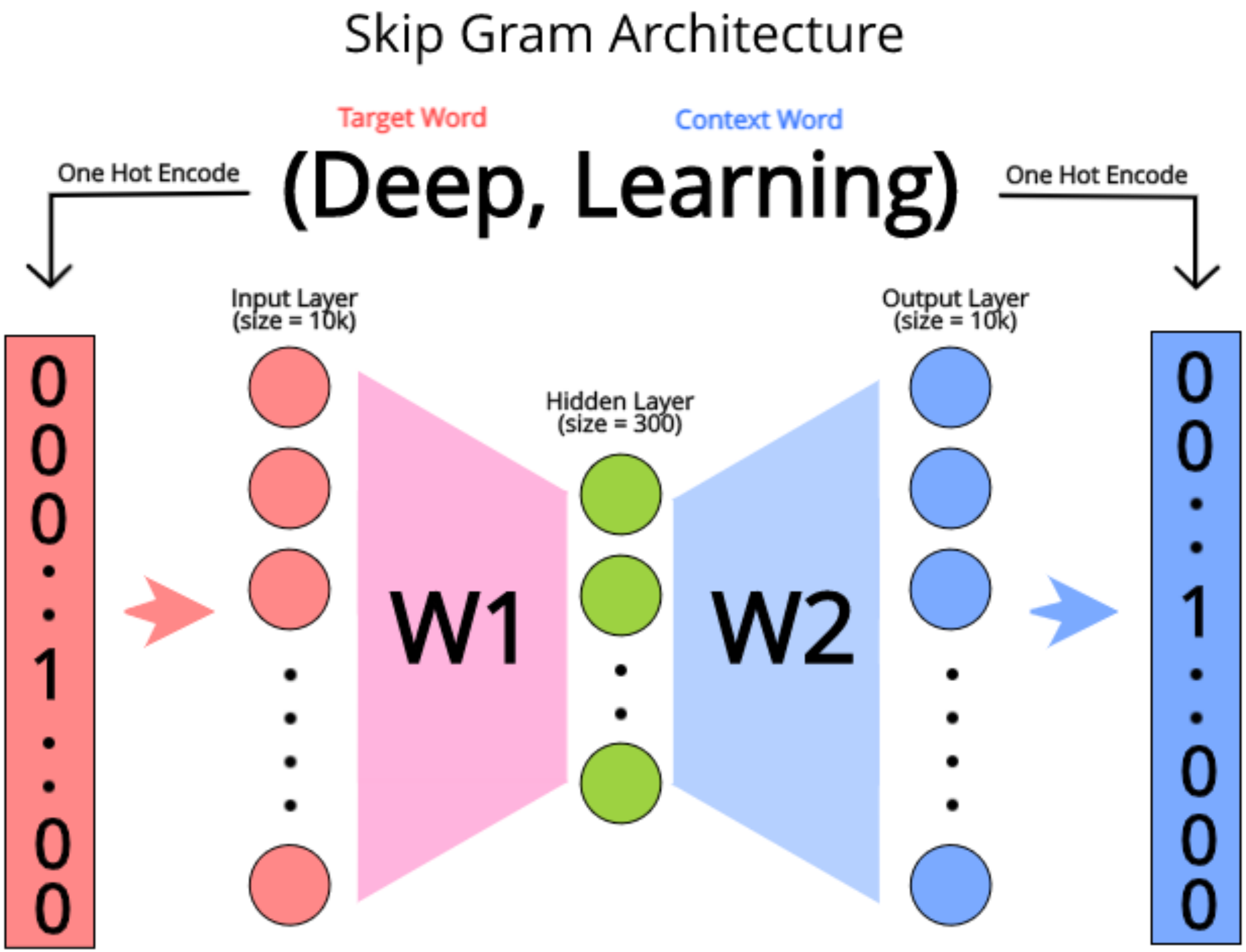


Word2vec

- In order to construct a Word2vec embedding
  - We need have a large corpus of text (Wikipedia ?)
  - 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



Skip Gram approach



Skip Gram in Keras

- inputs = Input(shape=(1,), dtype='int32', name="input")

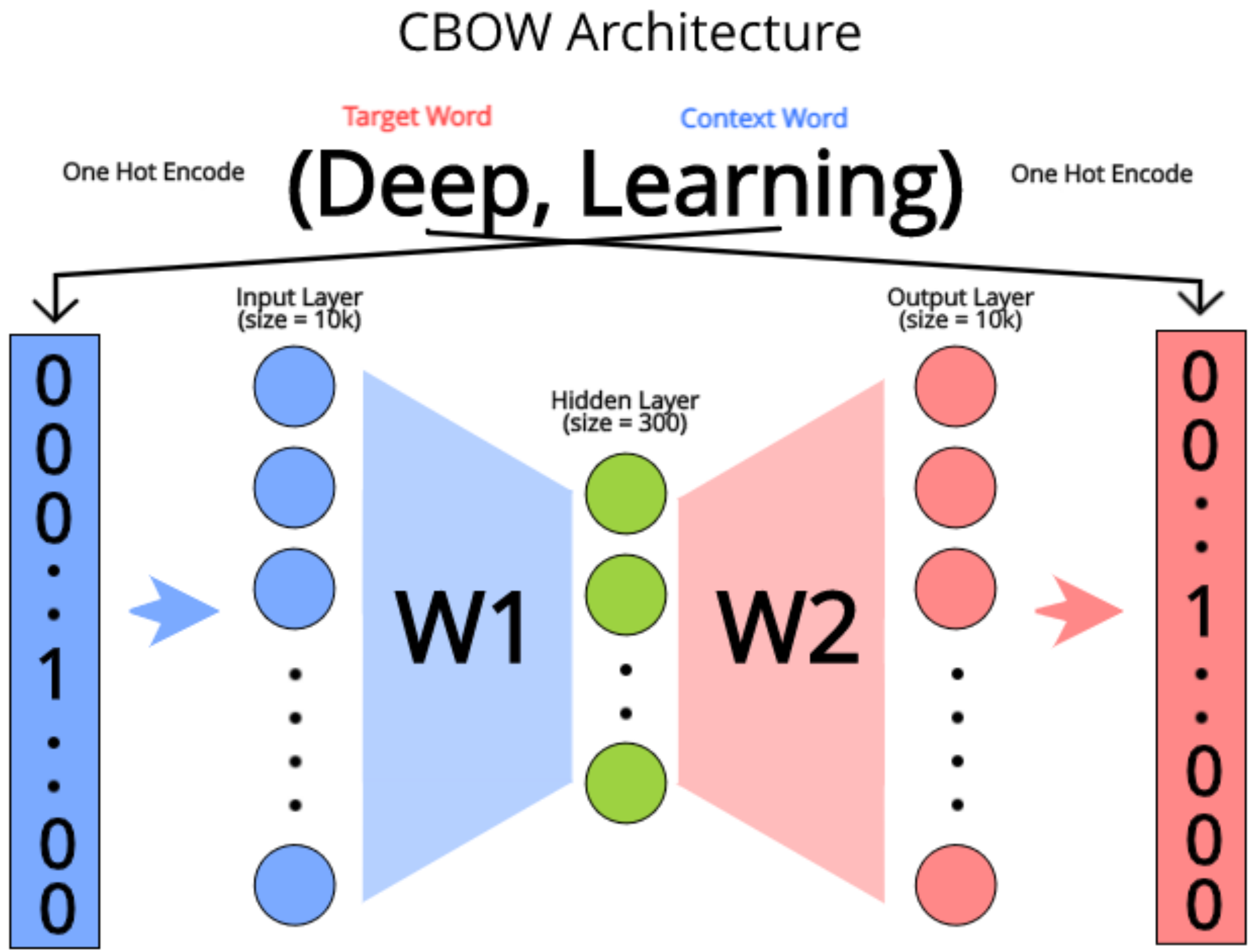
- embedding = Embedding(vocabulary_size, EMBEDDING_SIZE, input_length=1)(inputs)
- flatten = Flatten()embedding)
- output = Dense(vocabulary_size, activation='softmax')(flatten)

- # Model compilation
- model_skipgram = Model(inputs=inputs, outputs=output)
- model_skipgram.compile(optimizer=op, loss='categorical_crossentropy')

- # Keep embedding weight
- embedding_weights_skipgram = model_skipgram.get_weights()[0]

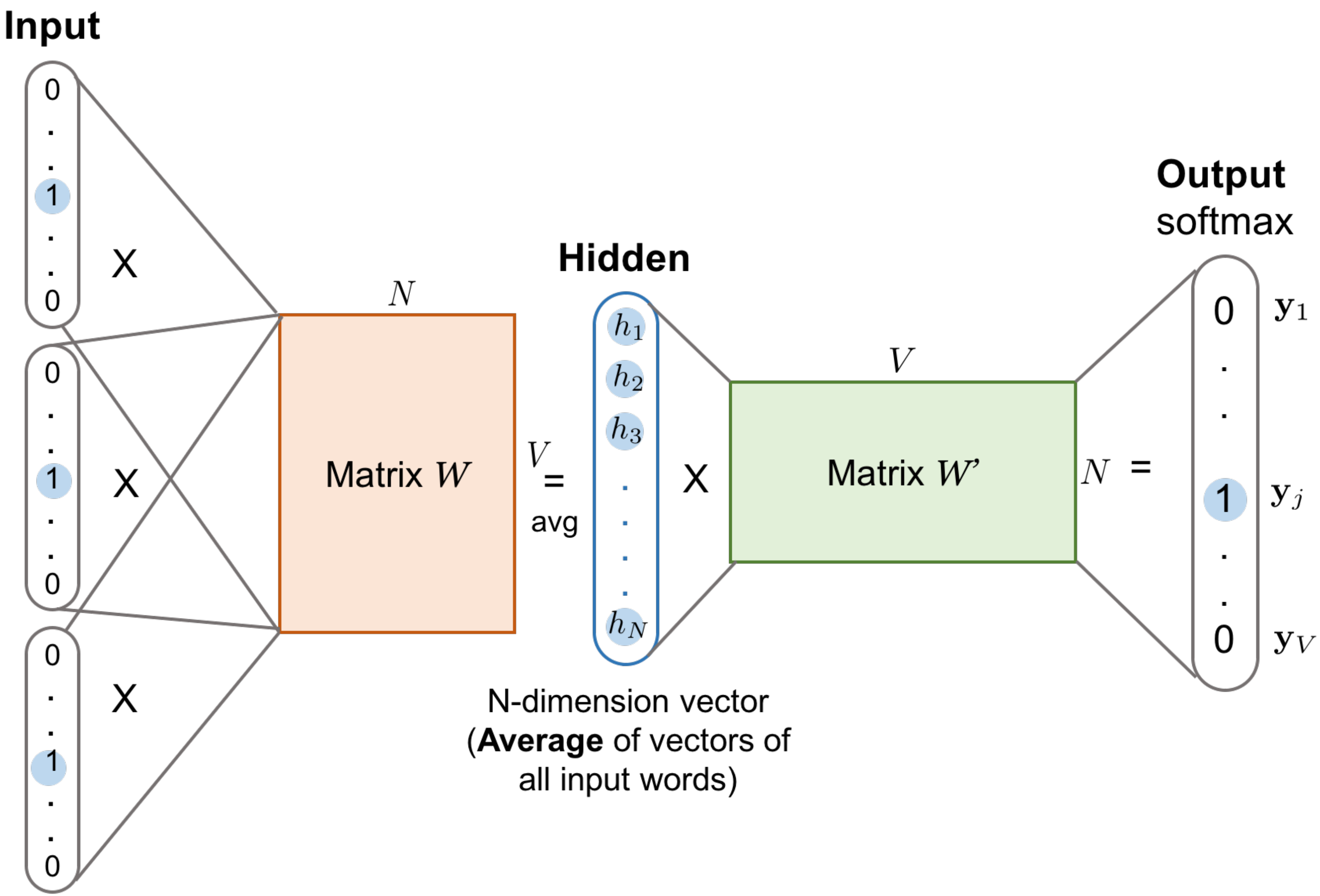
- # Use embedding weight
- weigths[embedding_weights_skipgram['my_word']]

CBow



CBOW Architecture
Target Word
Context Word
One Hot Encode
(Deep, Learning)
One Hot Encode
Input Layer
(size = 10k)
Output Layer
(size = 10k)
Hidden Layer
(size = 300)
W1
W2
0
0
0
1
0
0
0
0
1
0
0
0

Cbow
We can use directly all context word



Cbow in Keras

- inputs = Input(shape=(2*SLIDDING_WINDOWS,), dtype='int32', name="input")

- embedding = Embedding(vocabulary_size, EMBEDDING_SIZE,
  input_length=2*SLIDDING_WINDOWS)(inputs)

- mean_embedding = Lambda(lambda x: K.mean(x, axis=1),
  output_shape=(EMBEDDING_SIZE,))(embedding)
- flatten = Flatten()(mean)
- output = Dense(vocabulary_size, activation='softmax')(added)

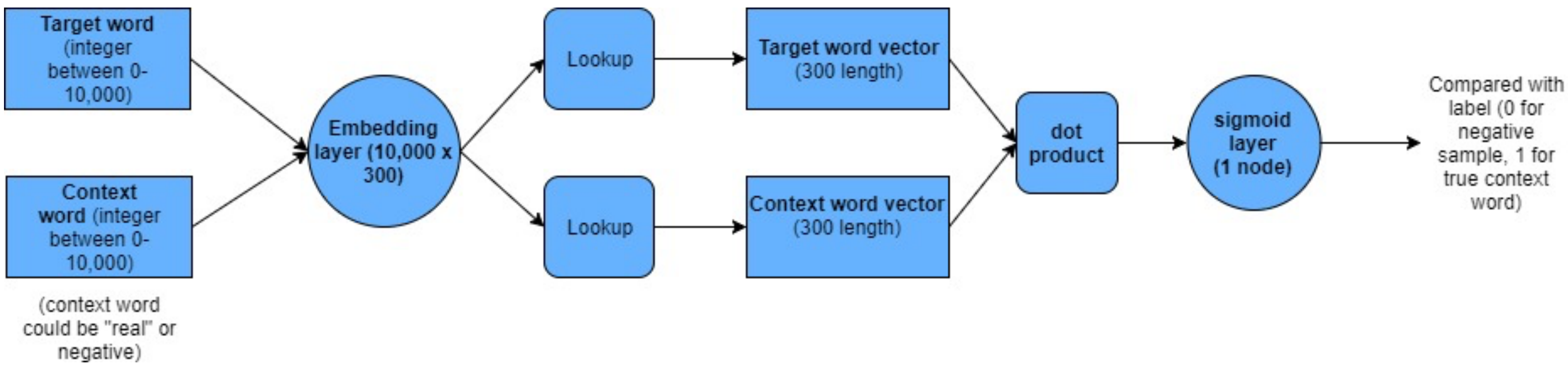
- # Model compilation
- model_cbow = Model(inputs=inputs, outputs=output)
- model_cbow.compile(optimizer=op, loss='categorical_crossentropy’)

- # Keep embedding weight
- embedding_weights_cbow = model_cbow.get_weights()[0]

- # Use embedding weight
- weigths[embedding_weights_cbow[‘my_word’]]

Skip gram model with negative sampling

- It is quite difficult to converge a Skip Gram or CBow model in particular because of the use of a One Hot encoding as a label.
- The idea of a sampling approach is to build a list of word pairs and to associate as a label True or False depending on whether the two words can be found in the same context or not.



Negative sampling in Keras

- Step 1: build sampling list probabilities
  - Use sequence module from Keras
  - ''' *Generates a word rank-based probabilistic sampling table.*

    Used for generating the sampling_table argument for skipgrams.
    sampling_table[i] is the probability of sampling the word i-th most common word in a dataset (more common words should be sampled less frequently, for balance).

    The sampling probabilities are generated according to the sampling distribution used in word2vec

    Arguments
    size: *Int, number of possible words to sample.*
    sampling_factor: *The sampling factor in the word2vec formula (1e-05 by default).*
    '''

  - from keras.preprocessing.sequence import make_sampling_table
  - sampling_table = make_sampling_table(vocabulary_size)

Negative sampling in Keras

- Step 2: build a list of pair of words based of the previous list of probabilities
  - Use sequence module from Keras
  - '''*This function transforms a sequence of word indexes (list of integers) into tuples of words of the form:*
    (word, word in the same window), with label 1 (positive samples).
    (word, random word from the vocabulary), with label 0 (negative samples).

    Arguments
    ***sequence**: A word sequence (sentence), encoded as a list of word indices (integers). Word indices are expected to match the rank of the words in a reference dataset (e.g. 10 would encode the 10-th most frequently occurring token).*
    ***vocabulary_size**: Int, maximum possible word index + 1 (0 is used for unknow word)*
    ***window_size**: Int, size of sampling windows (technically half-window). The window of a word w_i will be [i - window_size, i + window_size+1].*
    ***sampling_table**: 1D array of size vocabulary_size where the entry i encodes the probability to sample a word of rank i.*
    '''
  - from keras.preprocessing.sequence import skipgrams
  - couples, labels = skipgrams(data, vocabulary_size,
    window_size=SLIDDING_WINDOWS,
    sampling_table=sampling_table)

Negative sampling in Keras
The network

- input_target = Input((1,), name="target")
- input_context = Input((1,), name="context")

- embedding = Embedding(vocabulary_size, EMBEDDING_SIZE, input_length=1, name='embedding')
- target = embedding(input_target)
- target = Reshape((EMBEDDING_SIZE, 1), name="target_reshape")(target)
- context = embedding(input_context)
- context = Reshape((EMBEDDING_SIZE, 1), name="context_reshape")(context)

- dot_product = Dot(axes=1, normalize=False, name="dot_product")([target, context])
- dot_product = Flatten()(dot_product)

- output = Dense(1, activation='sigmoid', name="sofmax")(dot_product)

- # create the primary training model
- model_skneg = Model(input=[input_target, input_context], output=output)
- model_skneg.compile(loss='binary_crossentropy', optimizer='rmsprop')

- # Keep embedding weight
- embedding_weights_skneg = model_skneg.get_weights()[0]

Word2vec

- These representations are very good at encoding dimensions of similarity
  - We can visualize the learned vectors by projecting them down to 2 dimensions using for instance something like the t-SNE dimensionality reduction technique.
- Some fun word2vec analogies

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Using word2vec in your research . . .

- Easiest way to use it is via the Gensim libarary for Python (tends to be slowish, even though it tries to use C optimizations like Cython, NumPy)
  - https://radimrehurek.com/gensim/models/word2vec.html

- Original word2vec C code by Google
  - https://code.google.com/archive/p/word2vec/

Gensim lib in Python

- There is no Word2Vec embedding in NLTK.
- With Gensim
  - from gensim.models import Word2Vec
  - from nltk.corpus import brown, gutenberg

  - # build a word2vec model from a corpus
  - b = Word2Vec(brown.sents()) or Word2Vec(gutenberg.sents())

  - b.most_similar(‘man’, topn=4)
  - For brown
    - [('woman', 0.874), ('girl', 0.870), ('boy', 0.838), ('young', 0.784)]
  - For gutenberg
    - [('person', 0.736), ('body', 0.713), ('woman', 0.710), ('lady', 0.677)]

Word2vec in Python

- There are also pre-trained corpora
- Google's pre-trained model for gensim
  - https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
  - Vocabulary of 3 million words
  - Only english
  - The vector length is 300 features
- Wiki pre-trained model
  - https://fasttext.cc/docs/en/pretrained-vectors.html
  - 294 languages
  - Vector length is 300

Summary

2 approaches are preferable

- Build a specific embedding for the vocabulary used and the target task in this case we use the Embedding proposed by Keras.
  - **Disadvantage:** all the words of the **test set** not known by the train set will have a null embedding
- Reuse an existing embedding
  - it is necessary to find an embedding as close as possible to the vocabulary contained in the dataset
  - **Disadvantage:** all words from the **train** and the **test** set that are not embedding dataset will have a null embedding
  - It's some time possible to fine tune the embedding
- Main problem with these approach
  - OOV (Out-of-Vocabulary): the unknown vocabulary has a null vector
  - Polysemy: a word always has the same vector regardless of the context

Conclusion

- The embedding of words has become very important in the last two years with the appearance of new models (BERT, Elmo, etc.) that we will study later as they do not yet have sufficient neural networks.