Introduction to Word2Vec

STATGU4243/STATGR5243

TA : Claire He

Outline

- Introduction to textual modelling via bag-of-words approach
- Word2vec: understanding embeddings
- Word2vec: a dive-in in the architecture
- Word2vec in practice
- Further references

Natural Language Processing (NLP) methods

Assuming you have visualised and cleaned your data, you might want to now **model** some sort of relationship between texts in corpuses, semantic information etc. How do we do this?

Natural language processing gives you a toolbox of methods that tackle language modelling.

- Initial bag of words approach
- Word2Vec

Textual data and modelling: how?

Imagine you have a corpus of reviews of past attendance of courses given in the Statistics department. Let's say we extract the following sentence from one review:

"Applied Data Science is the best course I've taken during my semester!"

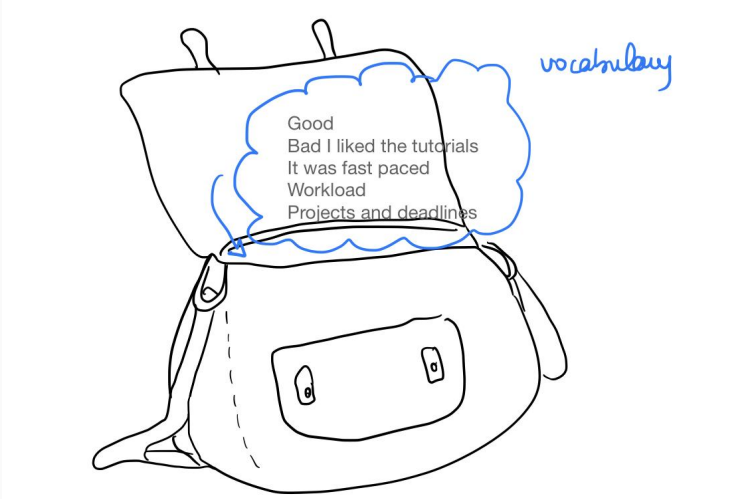
How would you represent the sentence? What is the main information conveyed through the sentence?

Bag-of-words approach

Answer: **vectorize** the text!

The most immediate approach one might think of is : a “bag of words” where the frequency of a word in the document is captured but not the order of apparition or the semantic construction.

This is useful for classification tasks.



Bag-of-words approach

Review1 : “ADS is the best class I’ve taken during the semester!”

Review2 : “ADS is a project-based course.”

Review3: “no exam in the semester. Best course!”

What problems do you see?

Document	ADS	Is	The	Best	Course	I’ve	Taken	During	Semester	A	Project-based	No	Exam	Class	In
Review1	1	1	2	1	0	1	1	1	1	0	0	0	0	1	0
Review2	1	1	0	0	1	0	0	0	0	1	1	0	0	0	0
Review3	0	0	1	1	1	0	0	0	0	0	0	1	1	0	1

Bag-of-words approach

What problems can you see happening?

- If the corpus gets big: the vocabulary grows and the more 0 there are.
- We lose the structure of the sentence: we know what occurred in the text but not **where.**
- Words' semantics are not captured.
- Words that appear frequently might not be the most important: ex *the, a,* …

Perks:

- Simple and computationally easy to do

N-grams and TF-IDF

Solutions to the problems?

- Semantics: Use *n-grams*: sequence of *n*-words often appearing together

An N-gram is an N-token sequence of words: a 2-gram (more commonly called a bigram) is a two-word sequence of words like “please turn”, “turn your”, or “your homework”, and a 3-gram (more commonly called a trigram) is a three-word sequence of words like “please turn your”, or “turn your homework”.

- Importance of words: TF-idf:

The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. tf–idf is one of the most popular term-weighting schemes today; 83% of text-based recommender systems in digital libraries use tf–idf

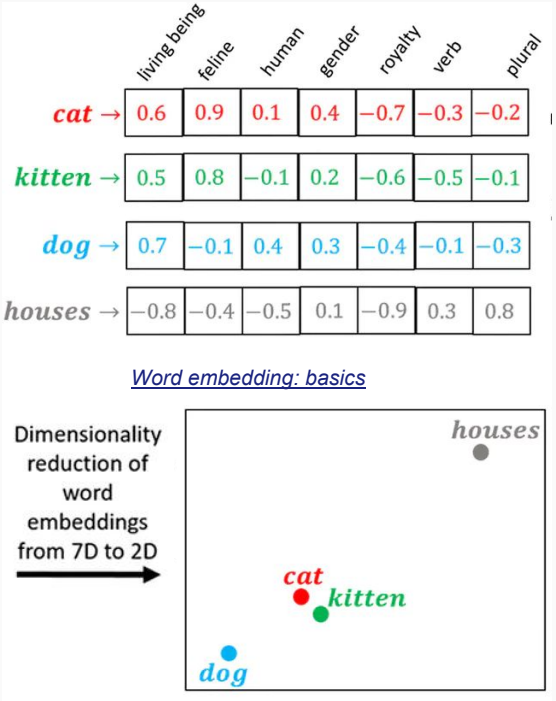
Word embedding

We want a representation of the words of the documents of our corpus (constituting a certain vocabulary) that allows:

- **Structure of sentences (contextualisation)**
- Semantic proximity: ex. *Cat, kitten, dog, house*

In the example against, words are embedded in a 7D-space reduced to a 2D-space.

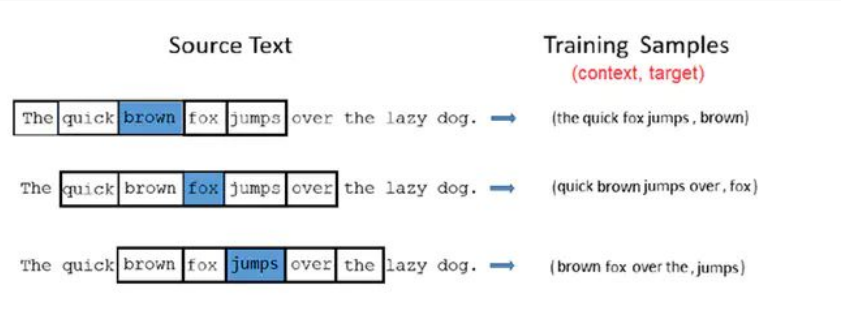
Word embedding allows for “logical” operations with words, the following formula is popular to sum this idea: *queen = king - man + woman*

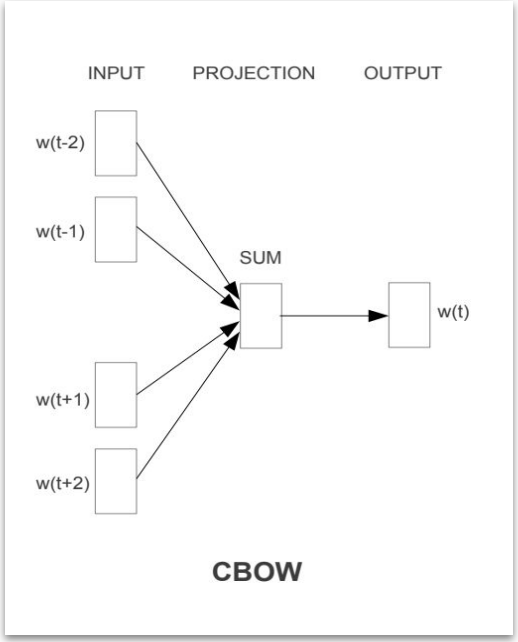


Word2vec: embedding

Word2Vec has a 2 algorithms CBOW (continuous bag-of-words, below) and skipgram which are 2-layer NNs trained for the following tasks:

CBOW: from context, predict target word

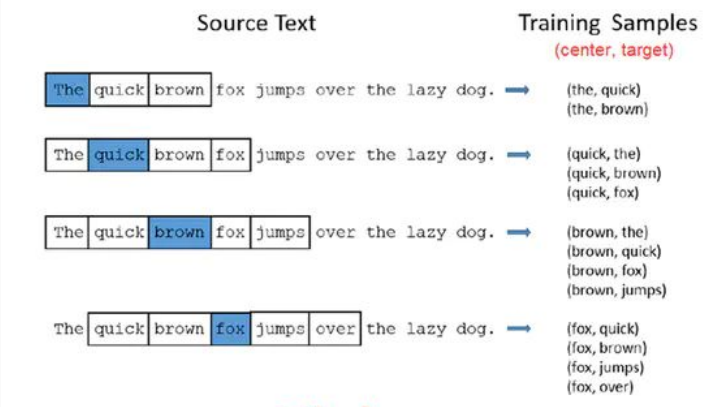


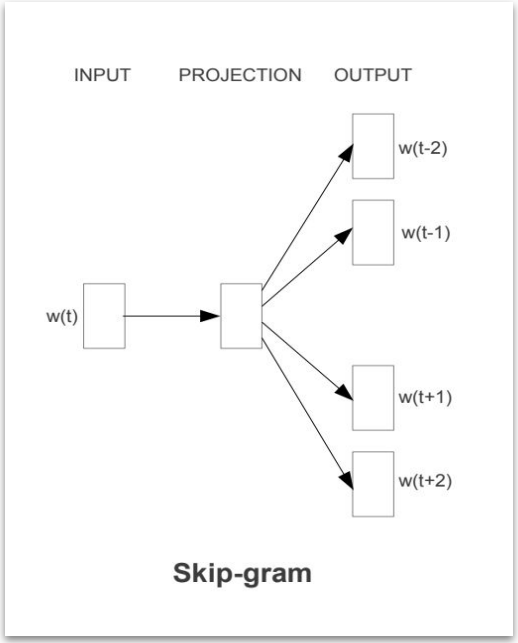


Word2vec: embedding

Word2Vec has a 2 algorithms CBOW (continuous bag-of-words) and skipgram (below) which are 2-layer NNs trained for the following tasks:

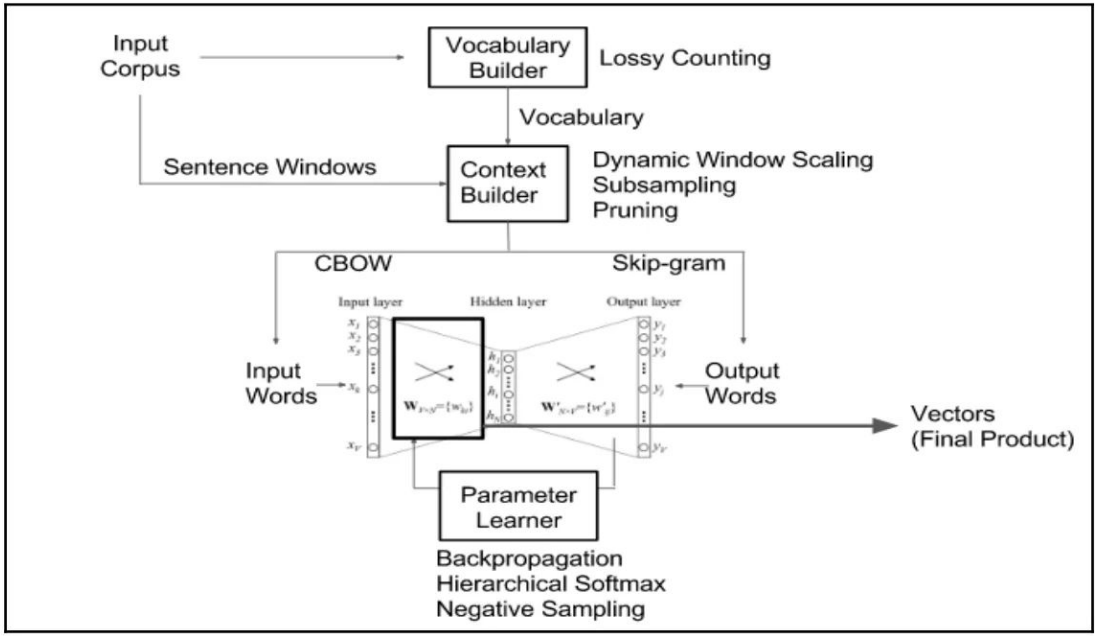
Skipgram: from word, predict context





Word2vec: architecture

- **Vocabulary builder**: build vocabulary in a “bag-of-words” fashion (index and count of words)
- **Context builder**: takes the vocabulary and a window of nearby words: the target word is the input of the skipgram, the context words are the input of the CBOW
- **Model** (either CBOW either skip-gram)

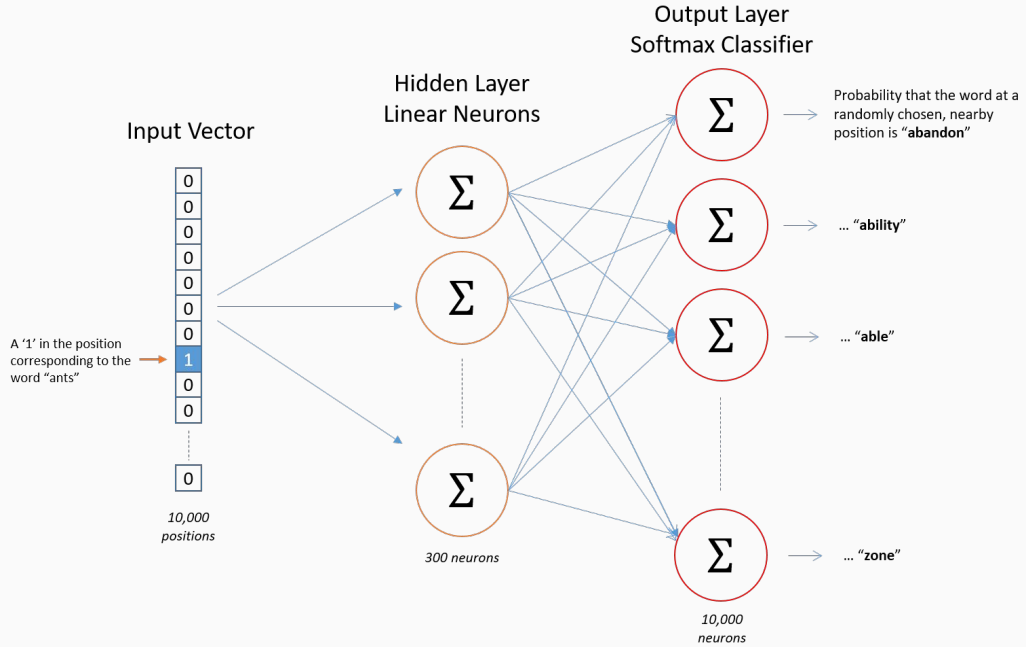


Word2vec: workings of the skipgram

Skipgram takes the target word and we want to predict the probability of the word being a context word.

Hidden layer is a fully connected (dense) layer and the weights are the *word embeddings*.

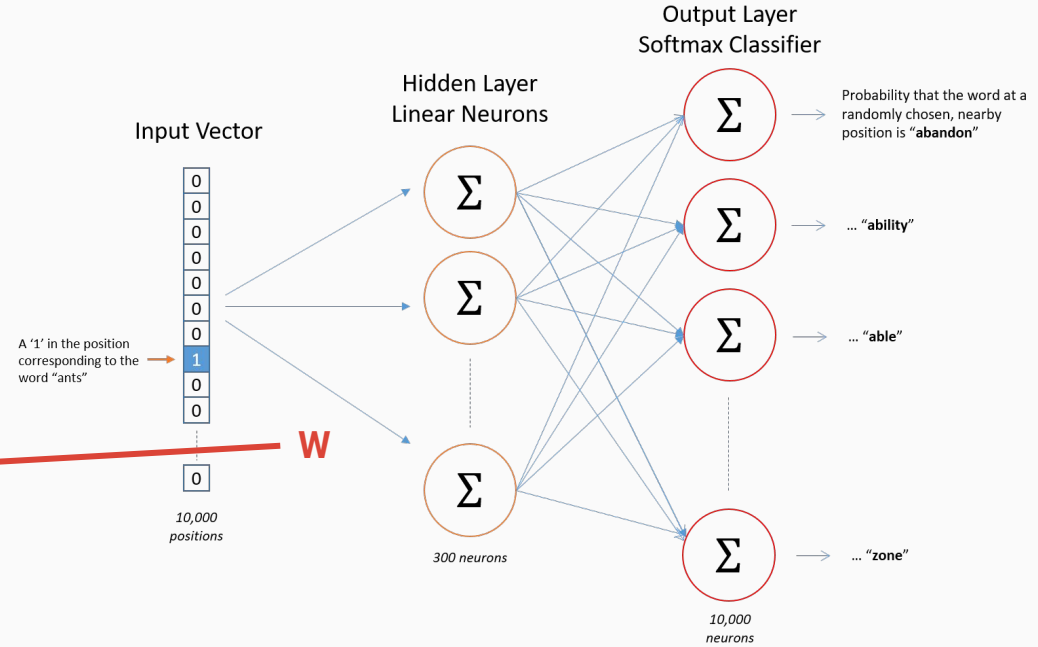
Take a vocabulary of 10,000 words which we represent with 300 features (dim. reduction).



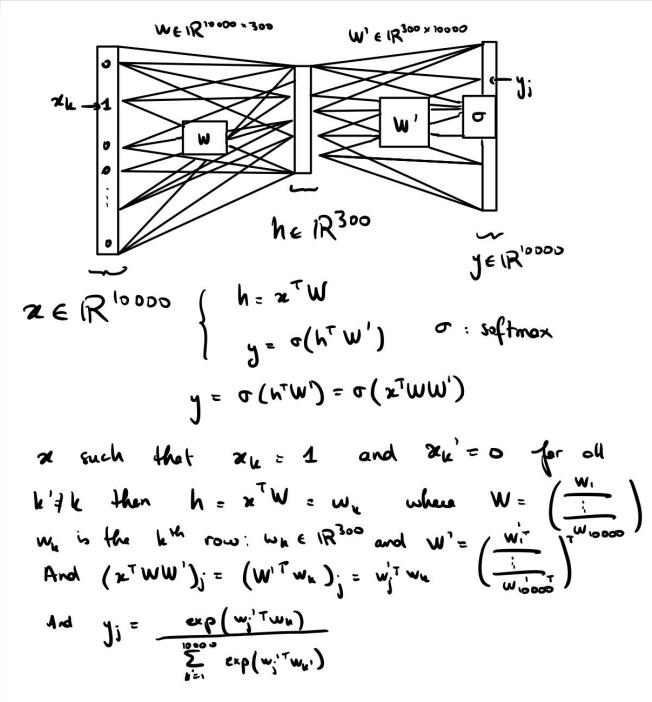
Word2vec: workings of the skipgram

Take a vocabulary of 10,000 words which we represent with 300 features (dim. reduction) the word embedding is the weight matrix *W* learnt by the network.

$$W = \begin{pmatrix} w_{1,1} & \cdots & w_{1,300} \\ w_{2,1} & \cdots & w_{2,300} \\ \vdots & \ddots & \vdots \\ w_{9\,999,1} & \ddots & \vdots \\ w_{10\,000,1} & \cdots & w_{10\,000,300} \end{pmatrix} \in \mathbb{R}^{10\,000 \times 300}$$



Some mathy stuff for the brave



This is how the weights are propagated forward. Backpropagation needs us to estimate the weights which usually uses some optimisation schemes to minimise a loss function (cross entropy here is appropriate).

Popular methods include stochastic gradient (the version of gradient descent that is appropriate to large architectures like neural networks), adam optimisers, etc.

Other technical verbiage on the architecture are techniques used for computational results (see negative sampling and hierarchical softmax).

Word2vec in practice

Ok, all the maths and theory is nice, but how do I use word2vec in practice?

- Python users: *word2vec* in *gensim* package doc
- R users: *word2vec* doc

Word2vec is a pretrained network: so you do not need to code the architecture from scratch! You only need to preprocess your data into tokens and retrain the network on your training data.

Worked example

Worked example

Further references

Now word2vec is no longer the state of the art in NLP models. If you are interested, here is some further readings in ***transformers***

- Hugging face's crash course in transformers
- BERT

If you are interested in the mathy/engineer aspects of word2vec and NN in general:

- post that works out the details of word2vec on backpropagation in particular
- Stanford CS course on DL for NLP https://cs224d.stanford.edu/
- Implementing word2vec from almost scratch