Word embeddings

Zoltan Miklos

University of Rennes 1

2022

Text in a natural language

- Dictionary : 300 000 words
- Words have many forms (conjugation, etc.)
- Words we find in texts might not be in the dictionary (e.g. conjugation, abbreviations, orthographic errors, etc.)
- Text is a sequence of words
- Sparse object : some words are frequent, but most of the words in the dictionary are very rarely used (power-low distribution)

How to use neural networks to learn insights about text ?

- The input to neural networks is a vector (matrix)
- Could we transform a text to a vector ? (or a word to a vector ?)
- There are a number of options, but some of these options are unpractical

One hot encoding of categorical attributes

One hot encoding is a widely used encoding of categorical attributes in data mining :

- We can convert categorical attributes to numeric, for example $\{red, green, blue\} \rightarrow \{1, 2, 3\}$
- (but if we follow this strategy, we add an implicit ordering to data)
- Alternatively, we can use a one-hot encoding $\{001, 010, 100\}$:
  - a vector (of size equal to the number of categorical values),
  - such that it has 0 and 1 values (all but one is 0).
  - The index of the value 1 "encodes" a specific categorical value.

One hot encoding of words

- Chose a dictionary (of size n) of all words
- For each word, associate a vector in the n dimensional space
- For the word that is the i-th in the dictionary, use a vector, whose i-th component is 1, and all other components are 0, so we have a vector of the form $(0, \ldots, 1, 0, \ldots, 0)$.

How to process textual data with neural networks ?

- we could use one hot encoding however
  - there are a lot of words in the dictionary, so the input layer could be very large
  - there is a lot connection and correlation between the words (the words “orange” and “juice” appear more frequently together than “ice” and “kangoroo”). Such connections are useful, but they are absent in one hot encoding
- often we embed the words in a vector space and we give this is the input to the network

Word embedding : dense representation of words in a vector space

- $Dictionary = \{\text{a, abracadabra, } \ldots\}$
- Vector space $\mathbb{R}^n$
- $embedding$ is a mapping $embedding : Dictionary \to \mathbb{R}^n$
- n is a hyper-parameter, often $n = 300$
- the dimension n is **much smaller** than the size of the dictionary
- We would like to "learn" word embeddings such that words with similar meaning correspond to vectors that are close to each other

Similar meaning, similar vectors

- What is the meaning of a word ? (very difficult question) when do two words have similar meaning ?
- Heidegger : the meaning of words is related to the set of other words that are used in context
- Idea : if we compare the context of words with similar meaning, they tend to overlap. Use the overlap of contexts as a proxy to identify words with similar meaning
- How to measure the similarity of vectors ? Via cosine similarity

$$sim(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{ab}}{|\mathbf{a}| \cdot |\mathbf{b}|}$$

Word embeddings

- $word2vec$ (embedding technique) reshaped the landscape of NLP projects
- More recently (ca 2019) the are even more advanced embedding techniques like ELMO, BERT, etc.
- GPT-1, GPT-2, GPT-3 ... : word embeddings trained on a large text corpus (applications exploiting GPT-3 generate nearly human quality texts)
- Controversies : some people consider these as weapons (as one can generate real-looking propaganda, spread over social media, and influence the outcome of the elections)
- Training word embeddings is extremely costly (computation, energy, ...), so often you can use already pre-trained word embeddings

Analogical queries and vector algebra

- Query : $Man \rightarrow Women : King \rightarrow ?X$
- Query answer (expected) : $?X = Queen$
- The cosine similarities between these vectors is high :

$$sim_{cos}(e_X, e_{king} - e_{man} + e_{woman}) \text{ is close to 1}$$

- Rappel : cosinus similarity of vectors $\mathbf{a}$ and $\mathbf{b}$:

$$sim_{cos}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{ab}}{|\mathbf{a}||\mathbf{b}|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}}$$

Bias in word embeddings

- Be careful, word embeddings are learned models, and contain human bias that is present in the training set (e.g. more data about man than woman, etc.)
- Predictions based on word embeddings (that are biased) can lead to biased predictions and indeed, there are a number of controversies (E.g. what is the profession of the woman in white coat in the hospital ? more likely a nurse than a doctor, but for men the prediction is more often a doctor)
- There is a lot of literature on this problem. It is beyond the scope of this class, but you should be aware of these issues

A future profession : “algorithmic bias engineer”

- Algorithmic fairness methods are emerging
- One should identify and eliminate these biases (in the data or in predictions or other downstream tasks)
- If this is possible : eliminating these biases is not only a technical issue, but involves societal questions, politics

How to learn word embeddings ? Small corpus

If you have a small dataset : do not learn the embeddings ! (as the quality might be poor)

- you should use pre-trained vectors.
- There are several publicly-available pre-trained word embeddings that were trained based on a large corpus.
- You can also exploit these to start and additional training, specific to the given text. (you start the training with the pre-trained vectors and update with the available corpus)

How to learn word embeddings ? Large corpus

Learning word embeddings requires large datasets, and high computing resources.

- General method for learning word embeddings : Train a neural network with 1 hidden layer, on a large corpus
- The vectors in the hidden layer correspond to the word embeddings
- Technique : unsupervised feature learning with auto-encoders

Skip-grams for a sentence

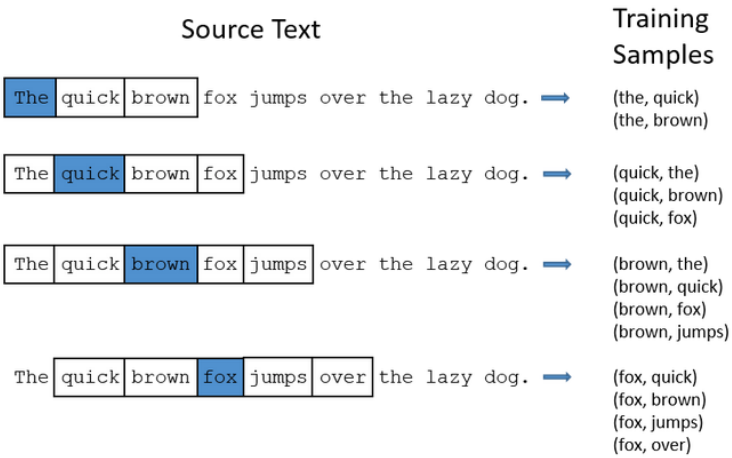
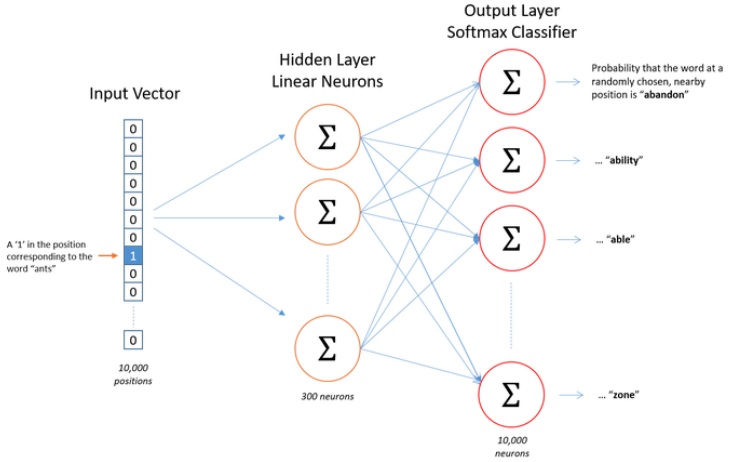


Figure – http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

The Skip-gram model

- Input layer : input vector w_I (one-hot encoded)
- Output layer (with softmax activation)
- Hidden layer : N dimension (for example $N = 300$, training parameter)

The Skip-gram model : network architecture



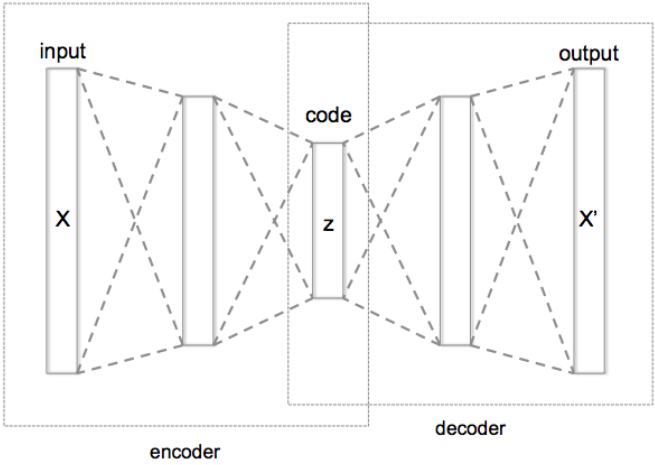
Skip gram : practical training

- There are a lot of weights to learn (in the toy example, vocab : 10000, embedding size : 300, two matrices : of size 3 million each)
- There is a need to apply additional techniques to improve the learning time and quality
  - word pairs : "nouvelle observateur" has a different meaning as "nouvelle" and "observateur"
  - Frequent words : the, and, a, or, etc. do not contribute understanding the context of a particular word
  - **Negative sampling** : positive example (orange, juice), negative examples, that is, unrelated words : (orange, supra-conductivity), (orange, socialism), ... Use a randomly selected words with 1 positive and k negative (k=5-20), and one iteration only update the selected word

Continous Bag of Words (CBOW)

- Skip-gram model : given a word, we try to predict the context (a set of words)
- CBOW model : given a set of words (context), we try to predict the word that is most likely

Autoencoder



Recurrent neural networks and NLP

- Text is a sequence of words
- If we represent words as vectors (word embedding), we can use the recurrent neural network techniques (LSTM, etc.)
- The length of documents (containing text) is not fix. RNNs eliminate this problem, and also the number of parameters does not depend on the size of the text.
- This is widely used, but "processing one word at a time" model does not allow to address some problems. For example,
  - we would like to understand a (very-long) phrase :
    $The\ king, who \ldots [long\ text, 1\ page] \ldots is\ dead.$
  - to understand who is dead, we need to remember a long text, the last word might not be enough.

Bidirectional RNNs

- Humans do not read the text sequentially : experiments with human subjects show that we look a bit to the left and to the right at the same time
- Bidirectional RNNs can better capture the context
- We use two RNNs (forward and backward directions) and combine their predictions with a small feed-forward network

Part of speech tagging

- Part of speech (POS) tagging : determining the lexical category of a word (verb, noun, adjective, etc.)

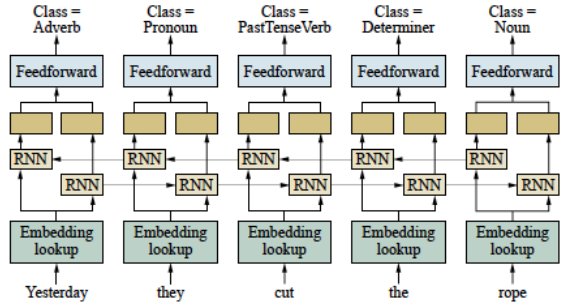


Figure 25.5 A bidirectional RNN network for POS tagging.

Sequence to sequence learning : motivation

Sequence to sequence learning :
input a sequence,
output a sequence
Sequence to sequence learning has lot of applications (examples) :

- Machine translation
- Activity detection
- Music generation
- Speech recognition
- DNA sequence analysis

Sequence to sequence models

A major break-through (e.g. in machine translation), but still some limitations

- Bias to nearby context
- Fixed context size
- Training large models is time consuming and computationally intensive