COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI ICADE CIHS

Unsupervised Learning - Word Embedding

Rodrigo Serna Pérez

March 2022

Today we'll talk about …

1. Natural Language Processing
2. Convert Text to Numbers
3. Word2Vec
4. Embedding Properties
5. Applications for Word2Vec
6. Variations for Word2Vec

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI ICADE CIHS

1

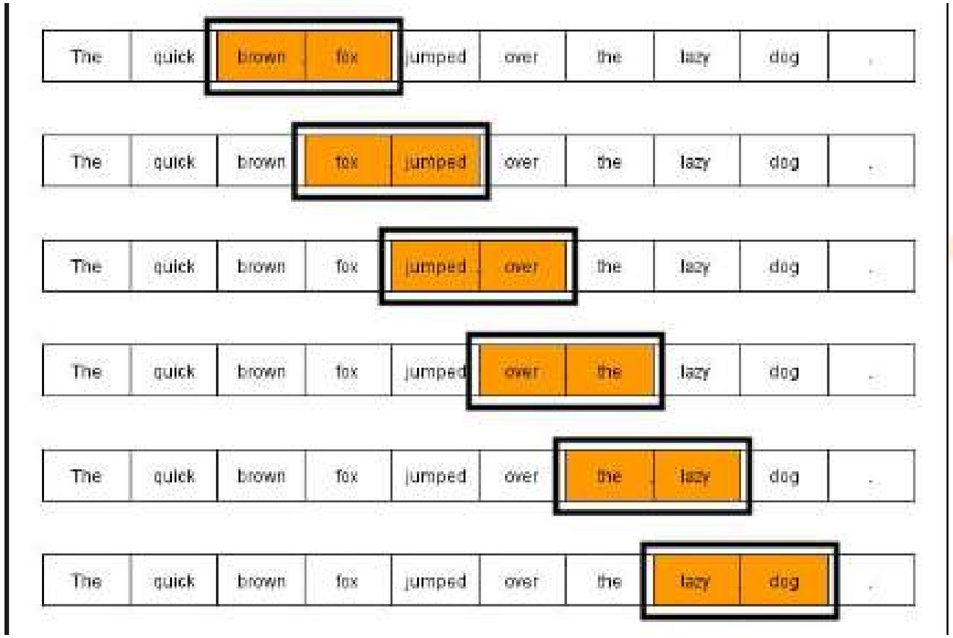
Natural Language Processing

Natural Language Processing

- Natural Language Processing is the field of AI that deals with problems related to language.
- Some of the most important problems are:
  - **Text Classification**: for example, topic or sentiment classification.
  - **Regression on texts**: such as predicting the score given the review of a movie.
  - **Topic Modelling**: splitting texts based on their topic (clustering concept applied to language).
  - **Named Entity Recognition (NER):** detect mentions to people, places, dates, numbers, organizations … in a text.
  - **Language Translation**: translate from one language to another.
  - **Language Generation**
  - **Question answering**: given a question and a context, find the answer to the question.
  - **Automatic Summarization**: generate a shorter text with similar meaning.

Preprocessing concepts

- **Tokenization:** splitting a piece of text into smaller parts (tokens). Normally tokens are words, but other units could be used.
- **N-gram**: tuple of N tokens. They are used to join words that tend to appear together and whose meaning is different that the meaning of individual words (‘New York’ is not a new York).



Preprocessing concepts

- **Lemmatization**: computing the base form of a word. Words are converted into the infinitive, names and adjectives into the singular male name.
  - Extremely complex task, is **not currently solved**.
  - Lemmatization is much more complex in Spanish that in English!!
  - There are tools to perform lemmatization, two types:
    - **Non contextual**: they work as a dict, given a word returns its lemma.
    - **Contextual**: uses the word context to get the lemma. Normally based on deep learning ("vino" as name will not be changed but as a verb will change to "venir").
- **Stemmer**: getting the root of a word (for example "corriendo" converts to "corr"). A bit simpler but loses too much information.

2

Convert text to numbers

Machine Learning on texts

- We are interested into **applying machine learning** to texts.
- For example, given a list of texts labeled with their sentiment (1 for positive, 0 for negative), an algorithm could learn how to automictically label new samples.
- Let’s build a Logistic Regression!!

```
from sklearn.linear_model import LogisticRegression

X_train = ["One of the best movies I have ever seen!!",
           "I will never buy an Iphone again"]
y_train = [1, 0]
LogisticRegression().fit(X_train, y_train)
```

Ups … Didn’t work …

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-646-02706a3bc129> in <module>
      4               "I will never buy an Iphone again"]
      5 y_train = [1, 0]
----> 6 LogisticRegression().fit(X_train, y_train)
```

```
~/bbva/open0marketsjwvjgyzoo4hak0n/venv/lib/python3.6/site-packages/sklearn/utils/validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    671                     array = array.astype(dtype, casting="unsafe", copy=False)
    672                 else:
--> 673                     array = np.asarray(array, order=order, dtype=dtype)
    674             except ComplexWarning as complex_warning:
    675                 raise ValueError("Complex data not supported\n"

~/bbva/open0marketsjwvjgyzoo4hak0n/venv/lib/python3.6/site-packages/numpy/core/_asarray.py in asarray(a, dtype, order)
     81
     82     """
---> 83     return array(a, dtype, copy=False, order=order)
     84
     85

ValueError: could not convert string to float: 'One of the best movies I have ever seen!!'
```

Machine Learning and Natural Language Processing

- Machine Learning algorithms **only work on numbers**!!
- We need to convert texts and words into numbers … but there isn't an intuitive way to do it.
- **General model for a text**: a text is a sequence of categorical variables (each word, a timestamp).

$$\underbrace{\textbf{I}}_{T_1}\ \underbrace{\textbf{saw}}_{T_2}\ \underbrace{\textbf{you}}_{T_3}\ \underbrace{\textbf{playing}}_{T_4}\ \underbrace{\textbf{with}}_{T_5}\ \underbrace{\textbf{your}}_{T_6}\ \underbrace{\textbf{brother.}}_{T_7}$$

Count vectorizer

- The easiest way to convert a text into numbers is with a **count vectorizer**.
- In a language with N different words, we will **convert a text into a vector of length N**.
- Position i of the vector is the number of times word i appears in the text.

text	again	an	belongs	best	book	but	buy	ever	favourite	have	his	iphone	is	it	most	movies	my	never	not	of	one	seen	the	to	will	writters
One of the most movies I have ever seen!	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	1	1	0	0	0
I will never buy an Iphone again	1	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0
The book belongs to one of my favourite writters, but it is not his best one.	0	0	1	1	1	1	0	0	1	0	1	0	1	1	0	0	1	0	1	1	2	0	1	1	0	1

Count Vectorizer

- Once the vectors have been generated, **we can train any algorithm with them**.

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer

X_train = ["One of the best movies I have ever seen!!",
           "I will never buy an Iphone again",
          "The book belongs to one of my favourite writters, but it is not his best one."]
y_train = [1, 0, 0]
cv = CountVectorizer().fit(X_train)
X_cv_train = cv.transform(X_train)
lr = LogisticRegression().fit(X_cv_train, y_train)
print("Now works :)")
```

```
Now works :)
```

Count Vectorizer

- Count Vectorizer models **do not take the order of words into account**
  - Problematic when texts are long.
- **Cannot generalize to unseen words**.
  - Training sample: “the movie was terrible”. Model will learn word “terrible” is associated to negative sentiments.
  - Test sample: “the movie was awful”, being “awful” an unseen word during training.
  - **Model will not be able to predict the test sample is negative**, although “terrible” and “awful” are synonyms.

Word embedding

- **Word Embedding techniques** assign a vector to each word.
- Simplest method: **one hot vectors**
  - Given a vocabulary of N words, each word of a sentence will be converted into a vector of N elements, being all equal to 0 except one that will be 1.
  - Example:

The cat sat on the mat

The: [0 1 0 0 0 0 0]

cat: [0 0 1 0 0 0 0]

sat: [0 0 0 1 0 0 0]

on: [0 0 0 0 1 0 0]

the: [0 0 0 0 0 1 0]

mat: [0 0 0 0 0 0 1]

Once encoded, it will be considered a **sequence of numeric variables**. Typical algorithms to process it are CNNs, RNNs and LSTMs.

3

Word2Vec

Word2Vec

- A language model could have above 10K different words. That means we must deal with onehot vectors of 10K positions!!
- Idea: use much **lower dimensionality continuous vectors**.
- Assign the word vectors in a way in which **words with similar meanings will have similar vectors.**

$$distance(\text{dog}, wolf) \ll distance(dog, plane)$$

- **Word2Vec: deep learning unsupervised algorithm** that computes vectors in a latent space for each word.

Word2Vec

- **Hypothesis**:

In deep learning, each level learns to
<MASK_WORD> its input data into a slightly more
abstract and composite representation.

- Could you guess which is the hidden word?

Word2Vec

- Word2Vec algorithms are models that **try to predict which a hidden word is given its context**.
- When two words are synonyms, model will never be able to learn which one was the hidden one.
- But if two words have very different meaning, they will appear in completely different contexts and model will learn they are different words.

Word2Vec – Training preprocessing

- Given a dataset of unlabeled texts, we can train a word2vec model. Before we must perform some preprocessing.
- From a sentence like:

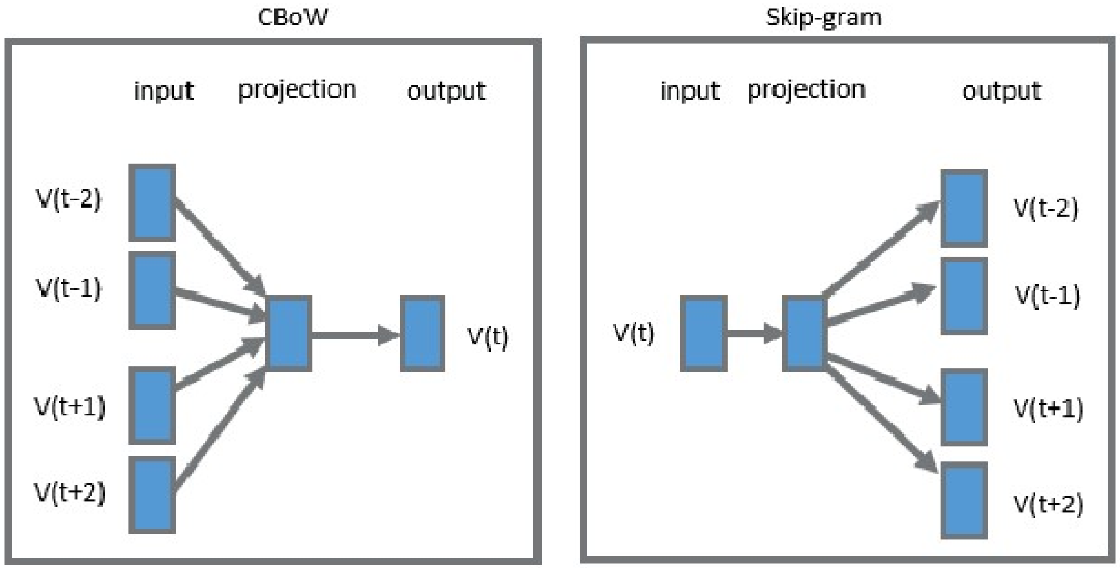
  "I will train a word2vec model using this sample text"

  We must choose a **window length** (hyperparameter, for example 2) and **generate two datasets**:

$$words = \begin{pmatrix} train \\ a \\ word2vec \\ model \\ using \\ this \end{pmatrix} \qquad contexts = \begin{pmatrix} [I, will, a, word2vec] \\ [will, train, word2vec, model] \\ [train, a, model, using] \\ [a, model, this, sample] \\ [model, using, sample, text] \end{pmatrix}$$

Word2Vec architectures

- There are **3 different architectures** used to compute the word2vec.
  - **Skipgram model**
  - **CBOW model**
  - **Negative Sampling model**



Skip-gram model

- Given a word, predict its context.
- Multilabel target.
- Embedding Layer + Dense Layer + Sigmoid Activation

```
[49]: # Skip-gram model
NUMBER_WORDS = 1_000
DIM_EMBEDDING = 50

input_ = Input(shape=(1,), name="word_input")
embedding = Embedding(NUMBER_WORDS, DIM_EMBEDDING, name="embedding")(input_)
embedding = Flatten(name="flat")(embedding)
output = Dense(NUMBER_WORDS, activation="sigmoid", name="final_layer")(embedding)
skipgram = keras.models.Model(input_, output, name="skipgram")
skipgram.summary()
```

```
Model: "skipgram"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 word_input (InputLayer)     [(None, 1)]               0

 embedding (Embedding)       (None, 1, 50)             50000

 flat (Flatten)              (None, 50)                0

 final_layer (Dense)         (None, 1000)              51000

=================================================================
Total params: 101,000
Trainable params: 101,000
Non-trainable params: 0
_________________________________________________________________
```

CBOW

- Given a context, predict the word
- Multiclass problem
- Embedding Layer + Pooling + Dense layer + Softmax activation

```
# CBOW model
NUMBER_WORDS = 1_000
DIM_EMBEDDING = 50
WINDOW_SIZE = 5

input_ = Input(shape=(2* WINDOW_SIZE,), name="context_input")
embedding = Embedding(NUMBER_WORDS, DIM_EMBEDDING, name="embedding")(input_)
embedding_pool = GlobalAveragePooling1D(name="pooling")(embedding)
output = Dense(NUMBER_WORDS, activation="softmax", name="final_layer")(embedding_pool)
skipgram = keras.models.Model(input_, output, name="skipgram")
skipgram.summary()
```

```
Model: "skipgram"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 context_input (InputLayer)  [(None, 10)]              0

 embedding (Embedding)       (None, 10, 50)            50000

 pooling (GlobalAveragePooli  (None, 50)               0
 ng1D)

 final_layer (Dense)         (None, 1000)              51000

=================================================================
Total params: 101,000
Trainable params: 101,000
Non-trainable params: 0
```

Negative Sampling

- **Both CBOW and Skip-gram are hard to train**.
  - Example: 10K words, embedding size of 100.
  - Embedding layer is efficient to train
  - But the final layer has 10K x 100 = 100K weights (plus bias).
  - For every sample, all weights of the final layer are updated!! Extremely expensive!!
- **Negative sampling: binary classification problem**.
  - Two inputs: context and word
  - Choose a context from the contexts dataset
  - Choose a word from the words dataset
  - Predict 1 if word belong the context, 0 if not.

Negative Sampling

- **4 steps in the model**
  - Computes an embedding for the input word.
  - Computes an embedding for the input context.
  - Computes the cosine similarity between word and context.
  - Applies final layer with sigmoid.

$$cosine\ similarity(x,y) = \cos(\theta_{x,y}) = \frac{<x,y>}{||x||^2||y||^2}$$

```python
# Negative Sampling model
NUMBER_WORDS = 1_000
DIM_EMBEDDING = 50
WINDOW_SIZE = 5

input_words = Input(shape=(1, ), name="word_input")
input_context = Input(shape=(2* WINDOW_SIZE,), name="context_input")

embedding_word = Embedding(NUMBER_WORDS, DIM_EMBEDDING, name="embedding_word")(input_words)
embedding_word = Flatten(name="flat")(embedding_word)

embedding_context = Embedding(NUMBER_WORDS, DIM_EMBEDDING, name="embedding_context")(input_context)
embedding_context = GlobalAveragePooling1D(name="pooling")(embedding_context)

x = Dot(axes=-1, name="cosine_similarity", normalize=True)([embedding_word, embedding_context])
output = Dense(1, activation="sigmoid")(x)

negative_sampling = keras.models.Model([input_words, input_context], output, name="negative_sampling_model")
negative_sampling.summary()
```

```
Model: "negative_sampling_model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 word_input (InputLayer)        [(None, 1)]          0           []

 context_input (InputLayer)     [(None, 10)]         0           []

 embedding_word (Embedding)     (None, 1, 50)        50000       ['word_input[0][0]']

 embedding_context (Embedding)  (None, 10, 50)       50000       ['context_input[0][0]']

 flat (Flatten)                 (None, 50)           0           ['embedding_word[0][0]']

 pooling (GlobalAveragePooling1  (None, 50)          0           ['embedding_context[0][0]']
 D)

 cosine_similarity (Dot)        (None, 1)            0           ['flat[0][0]',
                                                                  'pooling[0][0]']

 dense_12 (Dense)               (None, 1)            2           ['cosine_similarity[0][0]']

==================================================================================================
Total params: 100,002
Trainable params: 100,002
Non-trainable params: 0
__________________________________________________________________________________________________
```

Negative Sampling

- **Same embedding can be reused for both words and contexts**.
- Model will learn how to encode a word and its context in a same vector.

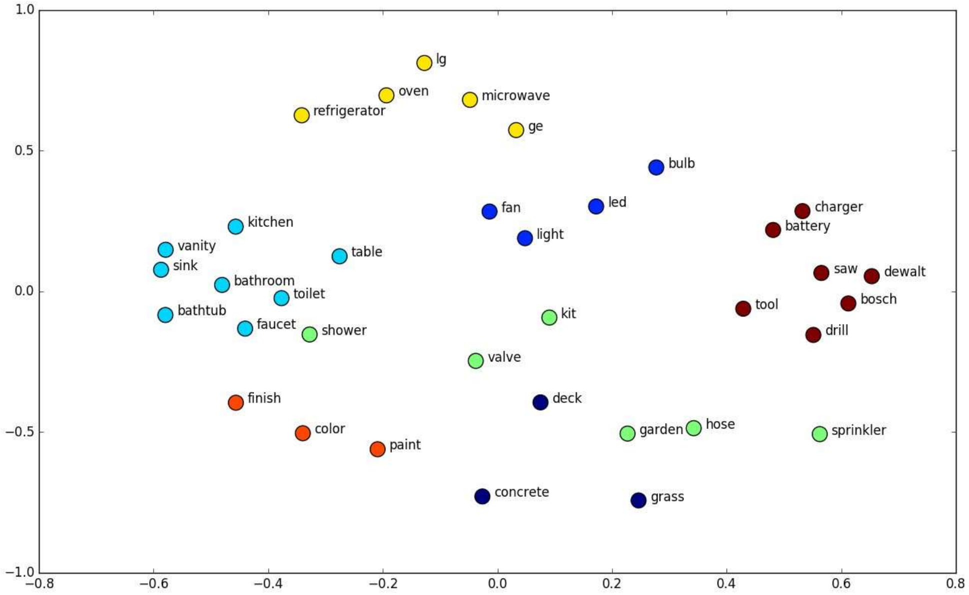
COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI ICADE CIHS

4

Embedding Properties

Properties

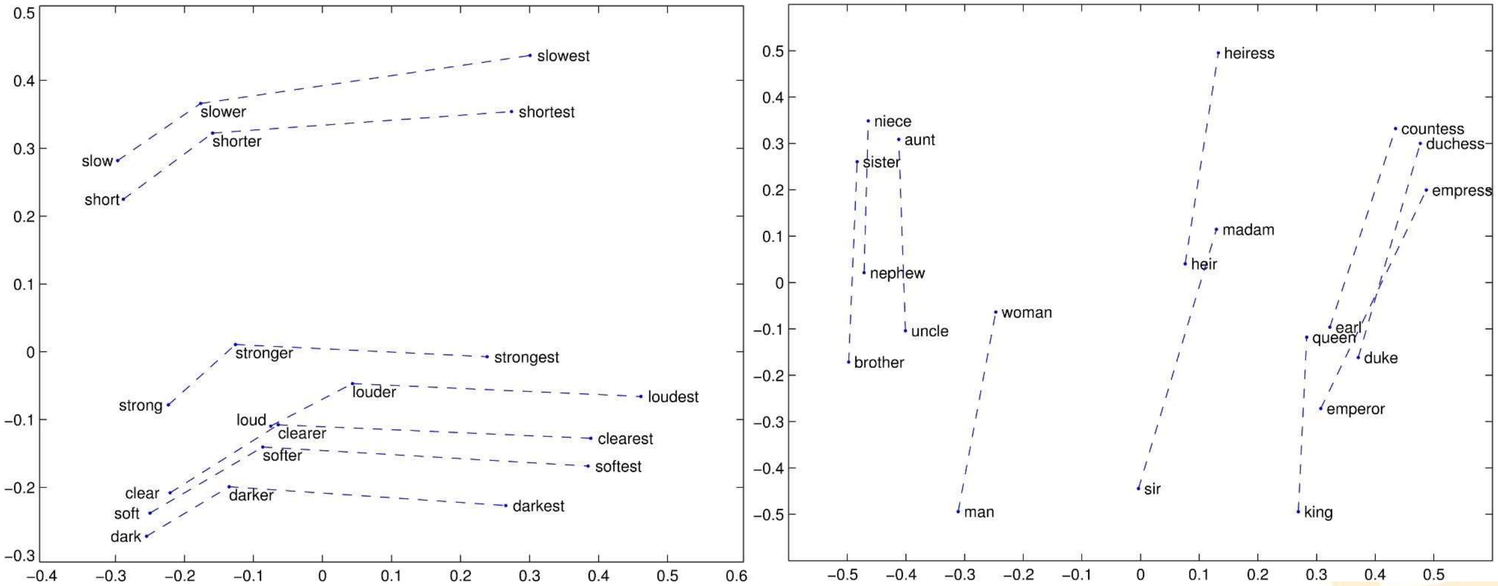
- **Words with similar meanings tend to appear close to each other** in the latent space.
- Example:
  - “king” will appear closed to words such as “queen”, “throne”, “royal” …



Properties

- In the latent space of the word embedding, it is possible to **compute arithmetical operations**.
- Some funny operations appear:

$$king + girl - boy \ \approx queen!!!$$



5

Applications for Word2Vec

Vocabulary reduction

- Machine Learning models' complexity normally **increase with the size of the vocabulary**.
- Word Embedding can be trained on our data and used in order to **detect words with the same meaning** which can be replaced by each other.
- We can also **train our model directly on the computed embeddings** (so the complexity of the model is not dependent on the size of the vocabulary).

Transfer Learning, Semi Supervised Learning

- Use a dataset of unlabeled texts to train a word2vec model.
- Then we **can init the embedding layer of a deep learning model with those vectors**.
- We can freeze the embedding set trainable to False or use them only as initialization.
- **Why it helps improving our models**:
  - Model will learn the embedding of word "nice" is associated to positive labels.
  - Model receives a sentence with word "beautiful", whose embedding is similar to "nice".
  - Model will still be able to associate the sentence to positive labels, even if "beautiful" word was not seen during training.

Sentence Embedding

- Sentence embedding means encoding a sentence into a vector so **sentences with similar meanings have similar vectors**.
- Several tasks can be performed on those vectors:
  - **Topic Modelling**: group sentences that talk about the same topic
  - **Information Retrieval**: search sentences in a long text that talk about certain topic.

6

Variations for Word2Vec

Variations

- **Glove**: similar concept to word2vec, but using other techniques.
- **Fasttext**: can split an unknown word into subtokens and still be able to encode it. Useful when the meaning of a word can be deduced from some of the parts of the word.
  - “hablando” can be split into “habl” and “ando” which explain the meaning of the word.
  - “appendicitis”: can be split into “appendic” and “itis”. If word “appendic” has its own vector and “itis” is associated to illnesses, we can get the meaning of the word.
- **ElMo, Bert, GPT …:** get contextual based vectors for words.
  - Vector of word “banco” will be different depending on the context.
  - Use large and complex deep learning architectures: transformers, attention.

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI ICADE CIHS

comillas.edu