ecnológico
Monterrey

NLP 2

Inteligencia artificial avanzada para la ciencia de datos II Modulo 5 NLP 2

Word Embeddings

Hot Encoding

Until now we have discuss some vector representations using a **frequency** vocabulary dimension to encode our text.

This encoding method is called **Hot encoding** and it is useful for ML models like (sentiment analysis), however semantic is not well represented.

For example assume a simple one word representation in a 3 vocabulary hot encoding.

Is there something in common between king and men?

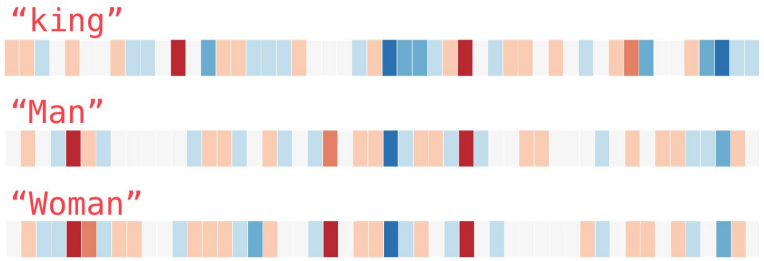
	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

Token	**Queen**	**woman**	**men**
king	0	0	0
men	0	0	1
woman	0	1	0

Embeddings

Embeddings is a technique to create vector word representations in which **semantic** is well represented in a dimension space.

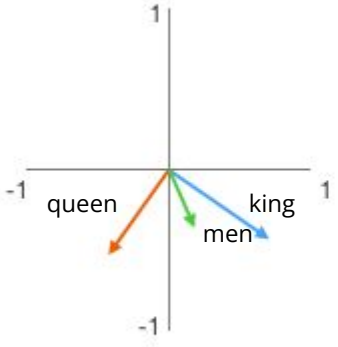
For example, **king and men** should be “more similar” in their vector representation than **king and woman** also **queen and woman** should be similar in a way.



Cosine similarity

If we are able to represent **semantics** in our vectors we will be able to compare to tokens on their meanings.

King and men vectors will have a smaller cosine distance than **queen and men.**



cosine_similarity(-0.4 0.8 , -0.3 0.2) = 0.87

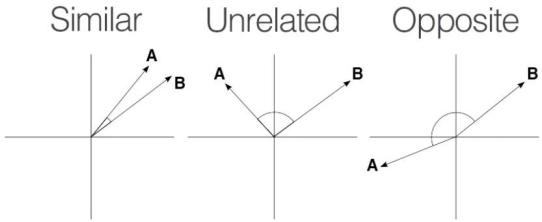
cosine_similarity(-0.4 0.8 , -0.5 -0.4) = -0.20

Semantic analogies

Embeddings and their cosine distances are also useful to find **synonyms, antonyms and relations.**

Also, by using vector arithmetic operations we are able to detect common or probable word neighbors given a **context.**

In the example if we “remove” king from man and we “add” woman , the we will get a resulting vector **similar** to Queen.

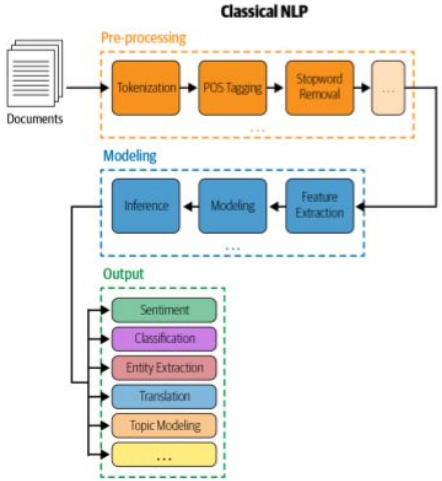


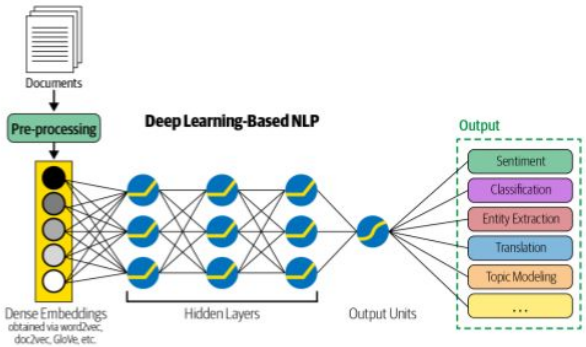
Language modeling

Embeddings is the final result of a group of models and techniques created from late 80's to 2013,let's discuss the path and evolution of this models until word2vec. The first step to create or language model it's the representation of our examples in terms of the events.

Until this point in the course, language events have a **pure statistical model** (bag of words, TF/IDF), even N-gram is a frequency model that uses compound tokens to improve probability.

By using **deep learning (Neural networks),** we are able to create models based on pure observation and pattern recognition models.



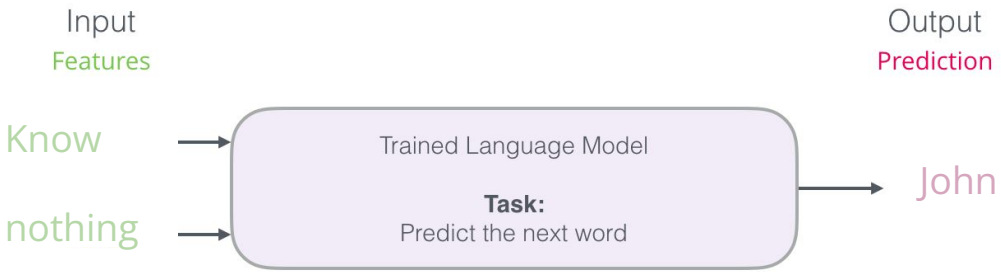


Time series

As we have discuss with n-grams model, text it's representable as a **time series event.**

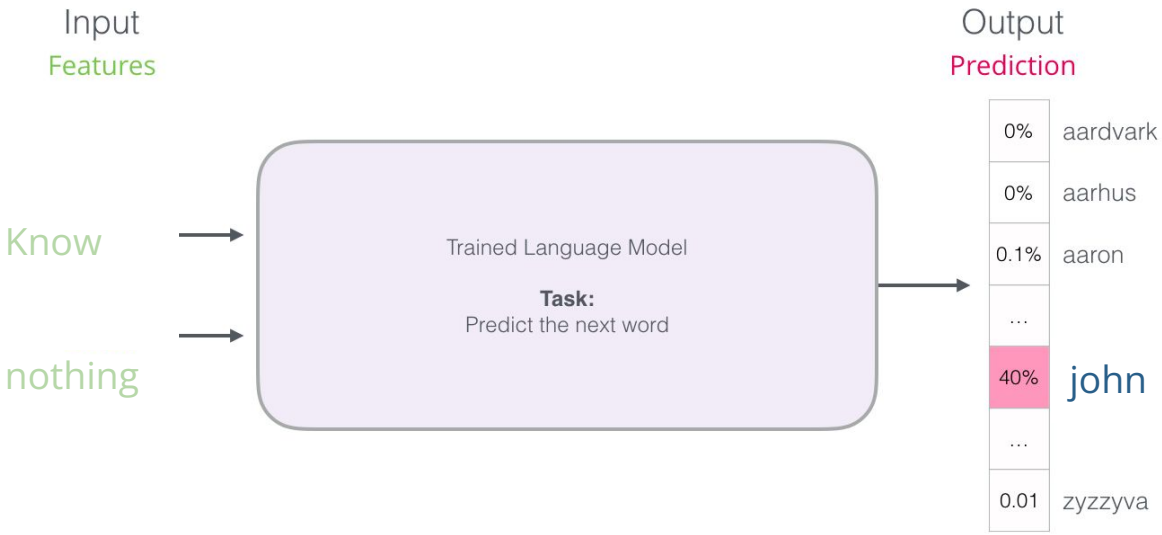
This is important because in a prediction model, we must be able to find this patterns in the time domain.

"You know nothing	____?____"
1st Event 2nd event 3rd event	4rd Event
Past	Future



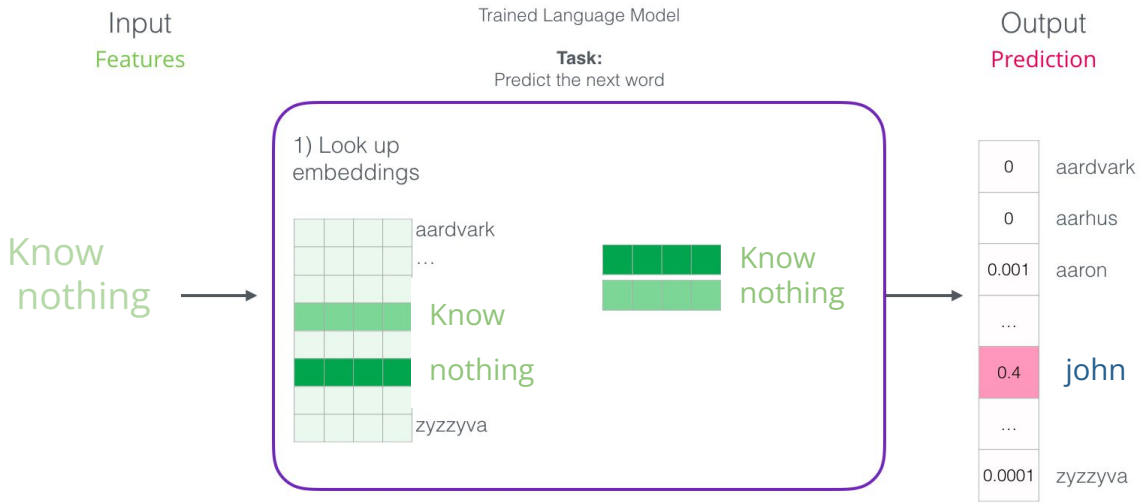
Decoded output

We have learned from hot encode vectors are useful for training, but actually in prediction text tasks we want am output vector so outputs model represents a probability vector output of words.



Dense embeddings layer

Since we will not have feature engineer as in ML, we will need to convert input text into a numerical vector representation, this is were embeddings are truly useful in order to find input time patterns.

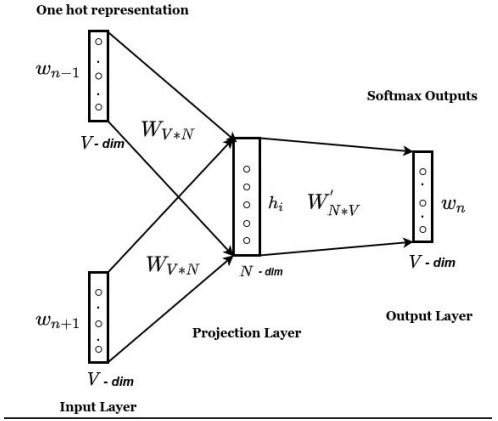


Continuous bag of words (CBOW)

The first model consists in represent time series event using window frames in corpus to create a supervised dataset.

Consider a 5 token window, 1 token will be taken as **label or target** and 4 a feature input.

This model will take a context input (surrounding tokens) to predict missing token (target) .



Corpus window	Input layer	Output layer
You know nothing john snow	**You , know**	[... , Nothing =1, ...]
You **know nothing** john snow	**Know, nothing**	[... , john=1, ...]
You know **nothing john** snow	**nothing, john**	[... , snow=1, ...]

Bi-directional CBOW

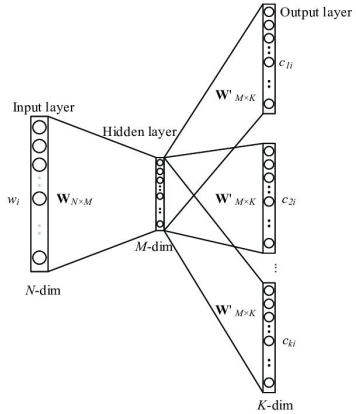
Is the same model but now, context is in both ways (past and future events).

This is useful not only for future predictions but also for missing tokens (mask task)

Corpus window	Input layer	Output layer
You know nothing john snow	**Know, nothing**	**[... , you=1, ...]**
You know nothing john snow	**you, nothing**	**[... , know=1, ...]**
You know nothing john snow	**you, know**	**[... , nothing=1, ...]**

Skip gram model

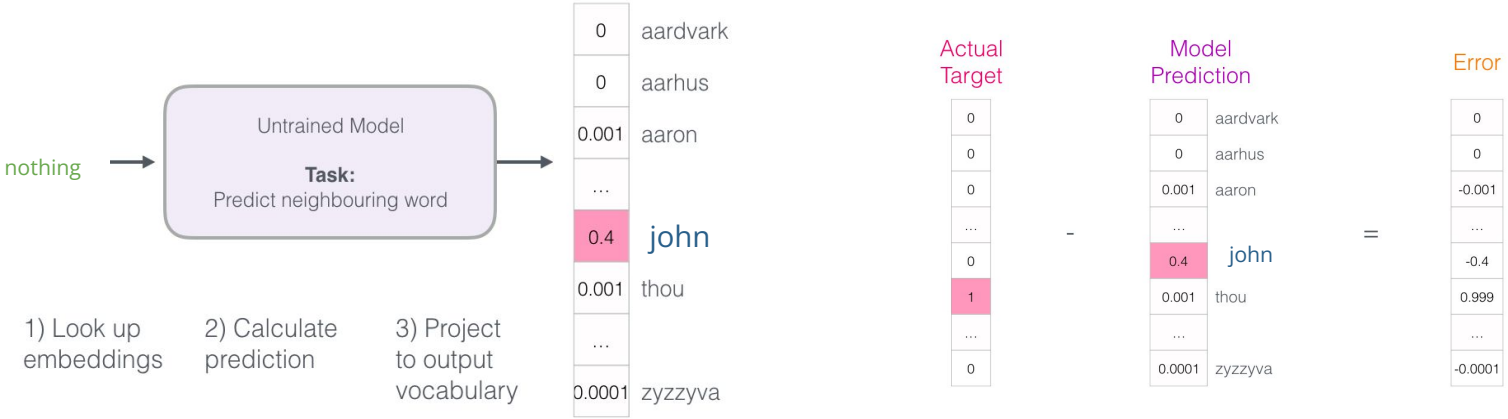
Another model is to invert the neural network model, now we will support as input a token representation (target) and the model output prediction represents a window token set (context)



Corpus window	Input layer	Output layer
You know nothing john snow	**Nothing**	**You**
You know nothing john snow	**Nothing**	**Know**
You **know nothing john** snow	**john**	**know**
You **know nothing john** snow	**john**	**nothing**

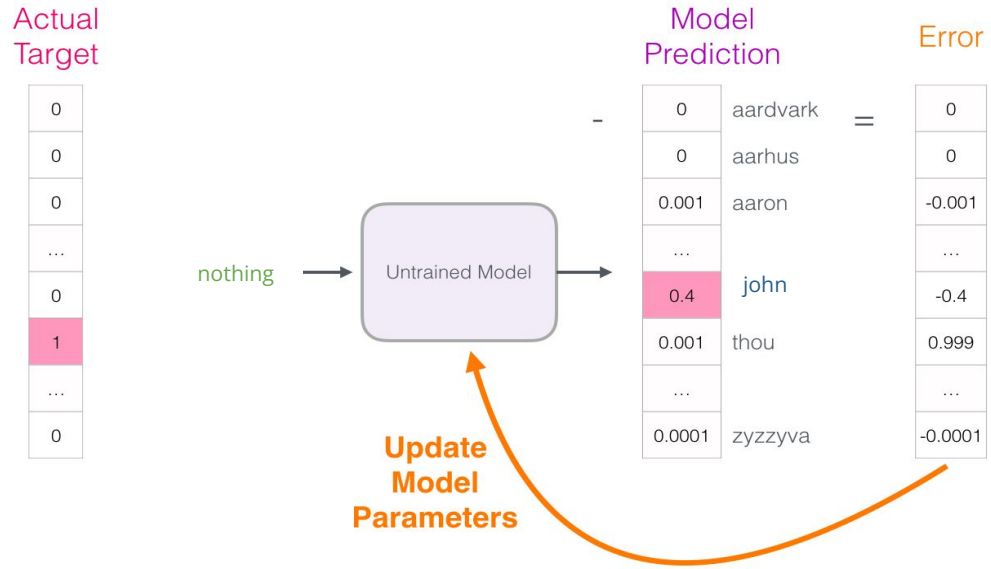
Training models

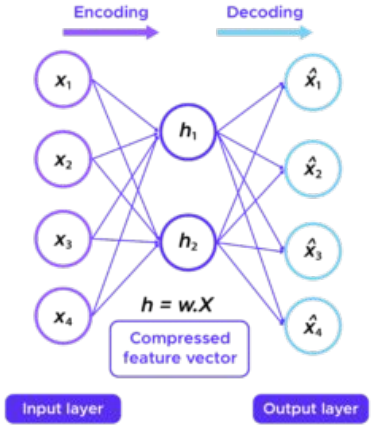
Since both are supervised model, we are able to train our network with a loss function and a weight matrix, because we are able to find an error function.



Error propagation

Now in every epoch (visit all the examples in the dataset) we are able to measure the errors, modify de W matrix weights and optimize the configuration.





$$\iff h = W \cdot X$$

The problem

This early models were useful for simple **predictions with semantics** (n-gram alternative), how ever is truly expensive (we use hot encoding), and there are better ways to deal with time series (we will discuss this later).

One way to avoid having vectors as outputs is to change our model, if we input two tokens, our **model will give us a probability score**

This implies to change our CBOW or skip gram dataset, so every time relation between tokens will be represented as a 100% probability example.

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

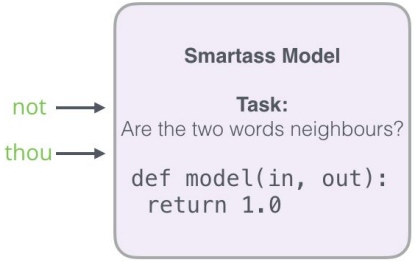
The Solution

Moving to this model will give us some benefit, first of all reducing complexity and hot encoding output.

Also, we will focus on **relations between any tokens** over time and not only forecast predictions.

However, we must include **negative sampling,** this means we need to include not 100% examples, otherwise our model might produce only 1 output (there is no other output in the dataset).

We can achieve this by **randomly giving unrelated tokens** examples.



input word	output word	target
not	thou	1
not		0
not		0
not	shalt	1
not	make	1

Negative examples

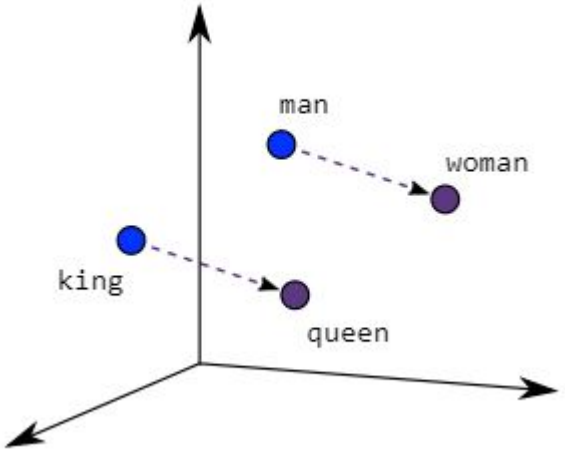
word2vec

Word2vec is a model created by Tomas Mikolov while he was working at Google.

https://arxiv.org/abs/1301.3781

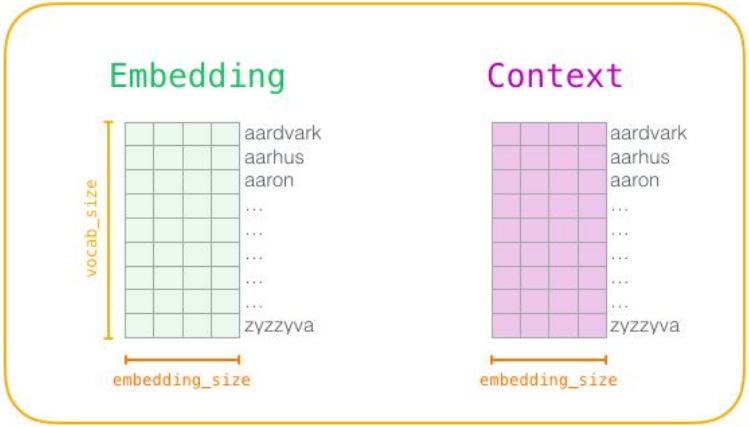
This model is one of the most successful methods to create embeddings using neural networks.

Word2vec needs a big amount of text in the corpus and it can use both methods for dataset vocabulary creation **(cbow and skipgram)**



Preparing model

After the creation of the training dataset by using **cbow** or **skipgram**, word2vec creates two random embedding matrix, each known word in the vocabulary is represented using a vector as target and as context (it doesn't matter at this point the semantics).



Training the model

In the first iteration of the model, all the examples are compared.

This means given to tokens (target and context), model needs to predict if they are related or not

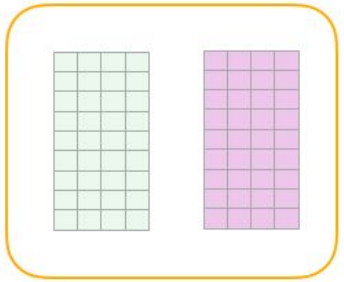
In order to measure the model error in training, the model calculates de **dot product** between vectors and uses a sigmoid function as activation function to translate this into a probability.

Finally target and prediction are compared

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

model



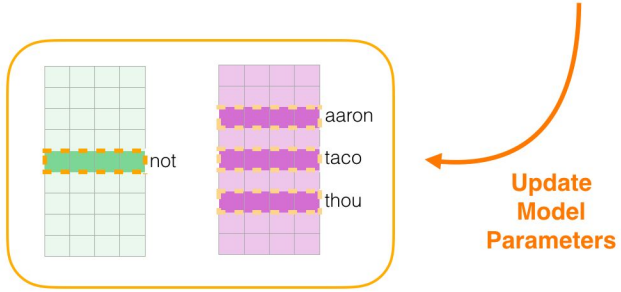
input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68

Epoch iteration

Every vocabulary iteration is known as an epoch.

At the end of every epoch error is optimized by using an optimization algorithm like gradient descent, the algorithm will **modify the vectors to minimize the errors**.

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	−1.11	0.25	−0.25
not	taco	0	0.74	0.68	−0.68



Hyper parameters

As we have discuss word2vec needs a **cbow** or **skipgram** representation of the dataset.

Also the **window size** is important since semantics depends over the time domain.

Finally the **size of the vocabulary** and the distribution between **positive samples and negative** will affect the training of the model.

Window size: 5

Window size: 15

Negative samples: 2

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

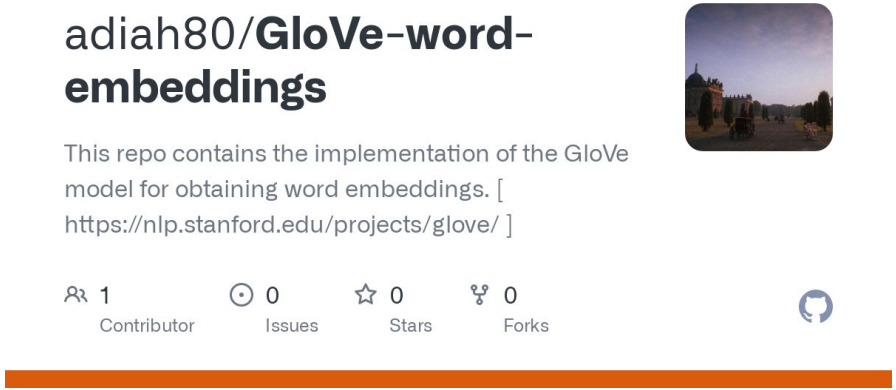
Negative samples: 5

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0
make	finglonger	0
make	plumbus	0
make	mango	0

Global Vectors for Word Representation (glove)

Word2vec is not the only embeddings model, another famous one was created by stanford University.

This method is an hybrid between supervised learning and statistical unsupervised models.

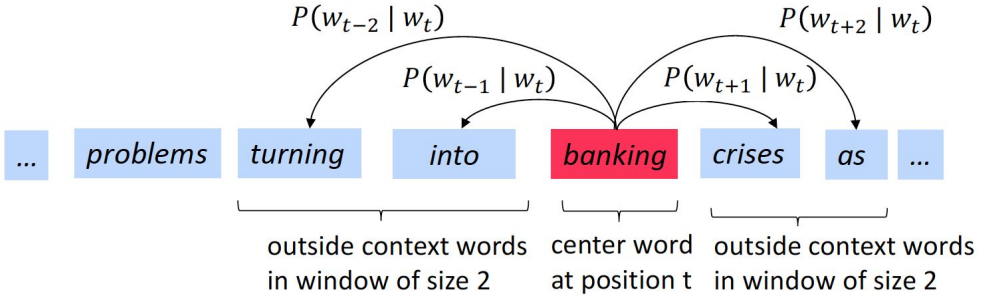


Probability issues

- Word2vec produces a target if the tokens are related by using **cbow** or **skipgram** events.

Intuitively for word2vec if an example of two words are found together, the probability of this event is 1.

However in the hole corpus this probability is not true.



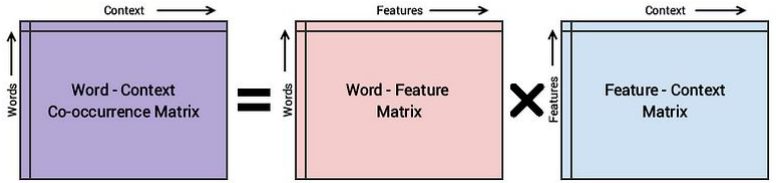
Glove probability

- Glove takes into account the global probability of the event by processing the corpus collocations and producing a co-occurence matrix.

- Basically, everytime two tokens are found together a counter is set +1.

- At the end the probability of this event is based on observation of the corpus and not assuming this is 100% probable.

	the	cat	sat	on	mat
the	0	1	0	1	1
cat	1	0	1	0	0
sat	0	1	0	1	0
on	1	0	1	0	0
mat	1	0	0	0	0



Thanks

Do you have any questions?

emmanuel.paez@tec.mx
Slack #module-5-nlp-1