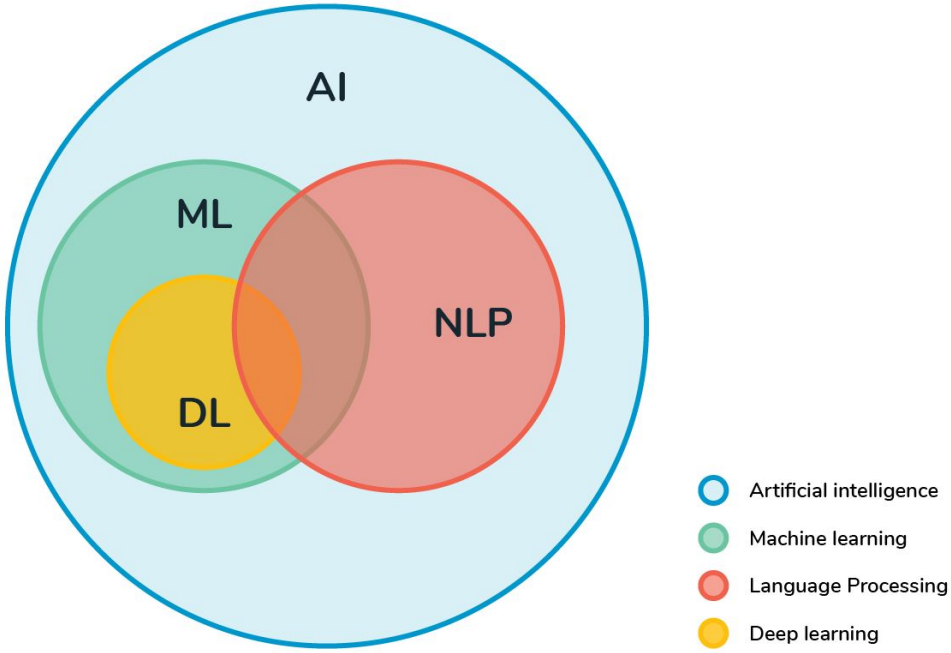
Aprendizaje Profundo

Procesamiento de Lenguaje Natural (NLP)

¿Qué es el Procesamiento de Lenguaje Natural (NLP)?

Es una rama dentro de la Inteligencia Artificial que busca dar a las computadoras la ***capacidad de entender texto y lenguaje hablado*** de la misma manera que los hacen los humanos y ***obtener información importante*** a partir de este lenguaje.

NLP combina Lingüística Computacional (i. e. modelado basado en reglas del lenguaje humano) con Estadistica, Machine Learning y modelos de Aprendizaje Profundo.



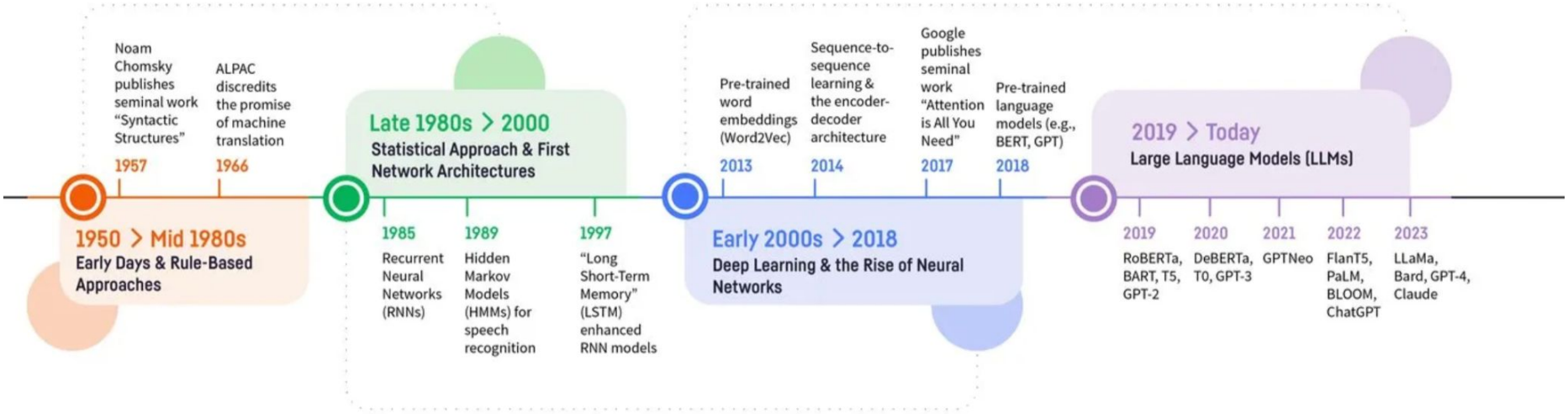
¿Por qué nos interesa trabajar con NLP?

- Las computadoras son muy buenas manejando ***datos numéricos*** de una manera bastante directa y con "poco esfuerzo" de nuestro lado.

- El ***lenguaje*** es algo intrínseco a los seres humanos. Por esta razón, podemos reconocer diferentes tipo de ***tonos*** o interpretar ***sentimientos*** en una conversación (enojo, alegría, tristeza, etc.) o hablar en otro idioma "casi sin pensarlo".

- El Lenguaje Natural se considera un tipo de datos ***no estructurado***. Las computadoras, por su lado, necesitan recibir este tipo de datos de la forma correcta (procesada o estructurada) de manera de poder simular el entendimiento de un ser humano.

NLP: Historia



¿Por qué nos interesa trabajar con NLP?

Datos estructurados

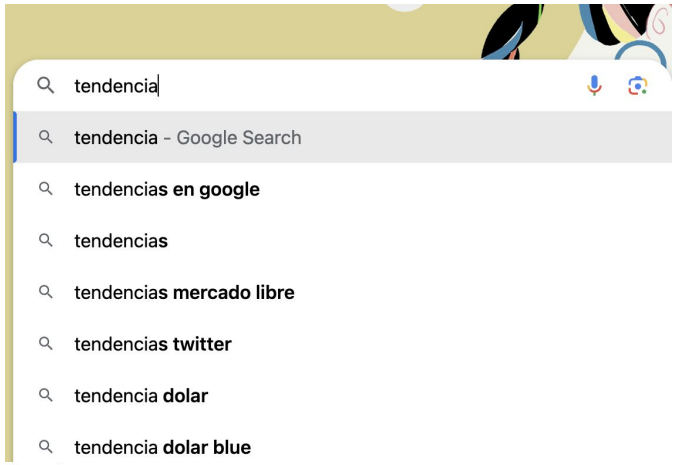
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

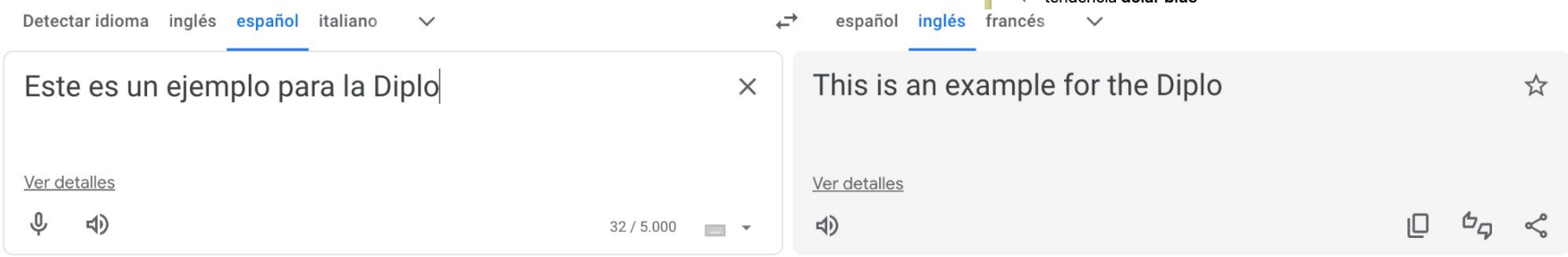
¿Por qué nos interesa trabajar con NLP?

Datos NO estructurados

☺ · The room was very hot in the night and although we asked for air conditioning to be turned on a few times, it finally was the last two days we were there which made it much more comfy. The breakfast was very good and they would make you fresh omelets if you wanted. Very clean and nice comfy large bed. Had a mini frig and fresh water each day as well as tea and coffee in the room.

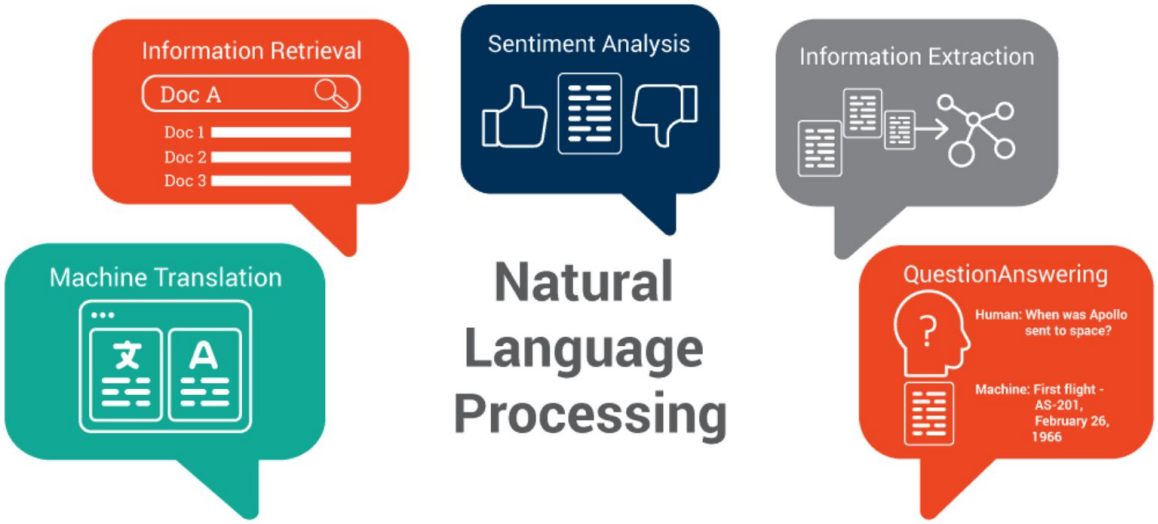
☹ · Had dinner there our last night - it was OK but not quite what we had ordered.



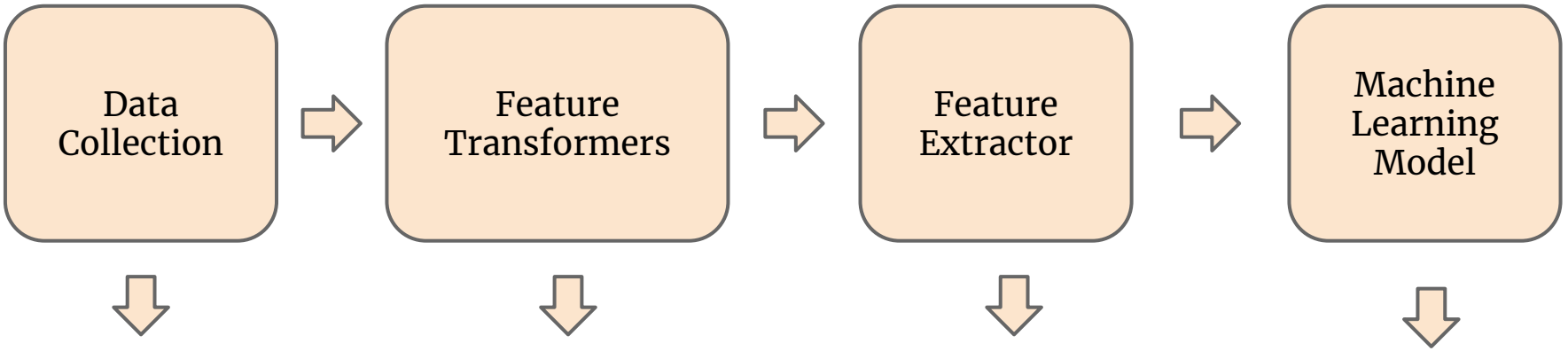


Casos de uso

- Machine Translation.
- Speech Recognition.
- Sentiment Analysis.
- Question Answering.
- Summarization of Text.
- Chatbot.
- Intelligent Systems.
- Text Classifications.
- Character Recognition.
- Spell Checking.
- Spam Detection.
- Autocomplete.
- Named Entity Recognition.
- Predictive Typing.



Pipeline de NLP

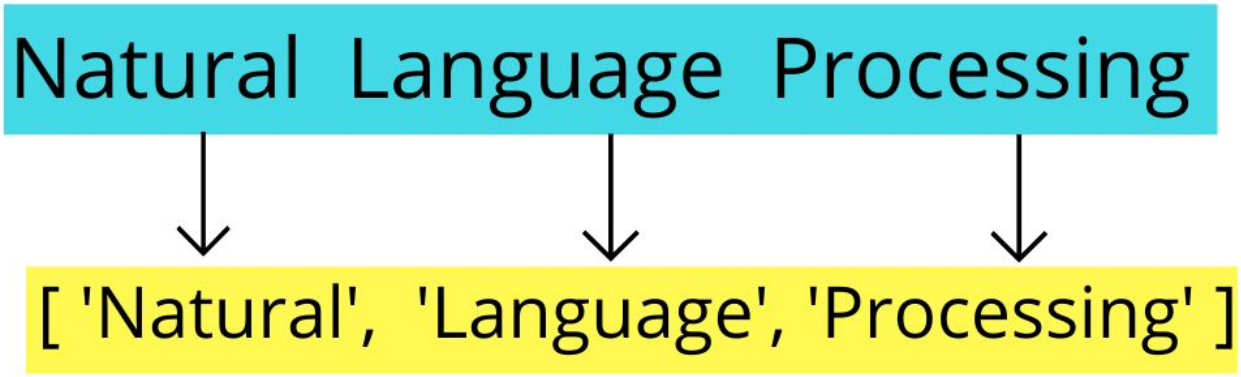


Algunas librerías de Python para NLP

	NLTK (Natural Language Toolkit)	Spacy	Gensim
Descripción	The NLTK Python framework is generally used as an education and research tool. It's not usually used on production applications.	spaCy is an open-source natural language processing Python library designed to be fast and production-ready. spaCy focuses on providing software for production usage.	Gensim is an NLP Python framework generally used in topic modeling and similarity detection. It is not a general-purpose NLP library, but it handles tasks assigned to it very well.
Features	• Tokenization. • Part Of Speech tagging (POS). • Named Entity Recognition (NER). • Classification. • Sentiment analysis. • Packages of chatbots.	• Tokenization. • Part Of Speech tagging (POS). • Named Entity Recognition (NER). • Classification. • Sentiment analysis. • Dependency parsing. • Word vectors.	• Latent semantic analysis. • Non-negative matrix factorization. • TF-IDF.
Casos de uso	• Recommendation systems. • Sentiment analysis. • Building chatbots.	• Autocomplete and autocorrect. • Analyzing reviews. • Summarization.	• Converting documents to vectors. • Finding text similarity. • Text summarization.

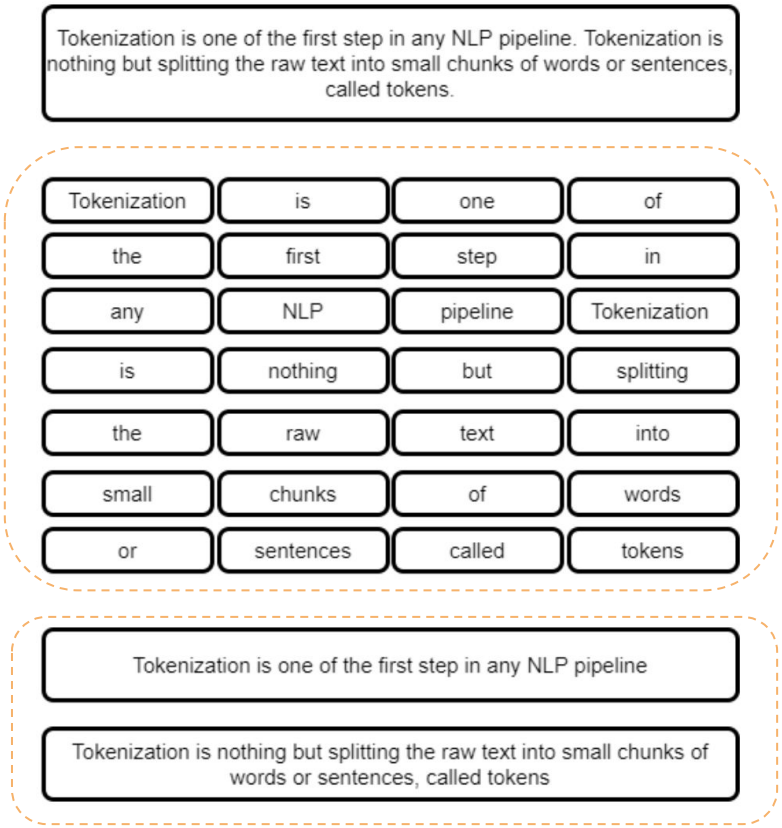
Tokenización

Es el proceso de dividir el texto original en componentes (tokens).



Tokenización

Es el proceso de dividir el texto original en componentes (tokens).



Word Tokenization

Sentence Tokenization

Tokenización

text_example:

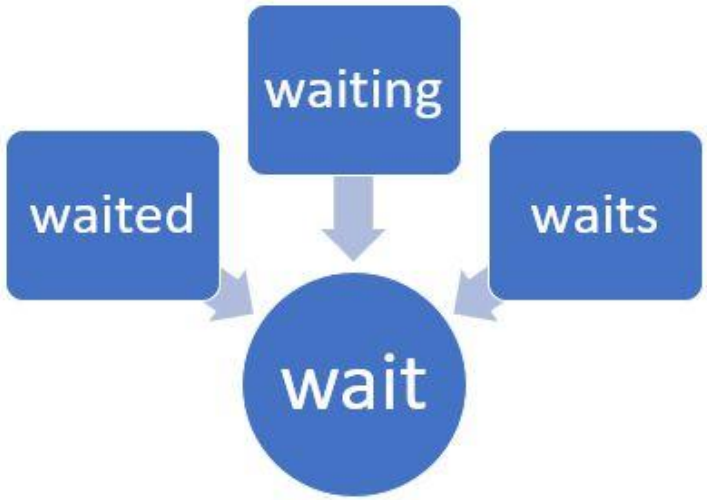
Natural language processing (NLP) is an interdisciplinary subfield of computer science and linguistics.
It is primarily concerned with giving computers the ability to support and manipulate speech.
It involves processing natural language datasets, such as text corpora or speech corpora, using either rule-based or probabilistic (i.e. statistical and, most recently, neural network-based) machine learning approaches.

```
from nltk.tokenize import sent_tokenize, word_tokenize
sentences_lst = sent_tokenize(text_example)
words_lst = word_tokenize(text_example)
```

```
['\nNatural language processing (NLP) is an interdisciplinary subfield of computer science and
linguistics.',
 'It is primarily concerned with giving computers the ability to support and manipulate speech.',
 'It involves processing natural language datasets, such as text corpora or speech corpora, using either
rule-based or probabilistic (i.e.',
 'statistical and, most recently, neural network-based) machine learning approaches.']
```

Stemming

- Es el proceso de producir variaciones morfológicas de una palabra base o raíz.
- Los algoritmos de stemming son comúnmente conocidos como Algoritmos de Stemming o ***Stemmers***.
- El proceso de stemming recibe como entrada las palabras tokenizadas y es usado para ***normalizar el texto*** y ayudar a su procesamiento.
- Suele ser útil para lidiar con la dispersión y/o estandarizar el vocabulario. También permite a los modelos de NLP aprender vínculos entre las palabras ayudando al modelo a comprender su uso en contextos similares.



Stemming

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in words_lst]
stemmed_words
```

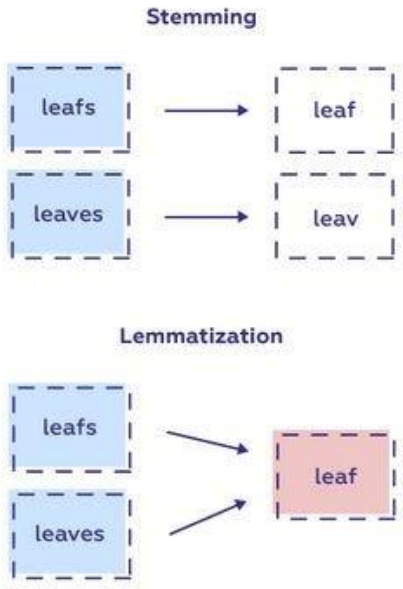
```
['natur',
 'languag',
 'process',
 '(',
 'nlp',
 ')',
 'is',
 'an',
 'interdisciplinari',
 'subfield',
 'of',
 'comput',
 'scienc',
 'and',
 'linguist',
 '.',
 'it',
 'is',
 'primarili',
 'concern',
 'with',
 'give',
```

Lemmatization

- Los algoritmos de **stemming** reconocen cuántas caracteres deben eliminarse pero ***NO CONOCEN el significado en sí de la palabra.***

- En Lemmatization, los algoritmos si tienen este conocimiento y ***utilizan un diccionario para comprender el significado de una palabra antes de reducirla a su raíz o lema.***

- De esta manera, un algoritmo de Lemmatization es capaz de reconocer que ***"better"*** es una palabra derivada de la palabra ***"good"*** mientras que un algoritmo de stemming no sería capaz de hacerlo.



Lemmatization

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("worst")) #Por defecto toma considera "nouns"
print(lemmatizer.lemmatize("worst", pos="a")) #Con el parametro pos podemos cambiar el defecto a "adjetivo"
```

```
worst
bad
```

Stemming vs Lemmatization

- *La Lemmatization es más precisa que el Stemming.* En problemas que involucran chatbots o asistentes virtuales la precisión en el entendimiento de las palabras es crucial.

- Dado que la Lemmatization implica obtener el significado de una palabra a partir de un diccionario o similar, consume bastante tiempo de cómputo. De esta manera, *los algoritmos de Stemming son más rápidos que los algoritmos de Lemmatization.*

- Dependiendo el problema o lo que se esté buscando, puede no haber diferencias entre ambas técnicas en los resultados finales.

Stop words

- Las ***stops words*** son aquellas palabras que se repiten muy frecuentemente en un idioma y que no brindan información relevante en el contexto de un texto.

- Estas palabras pueden dañar el procesamiento de lenguaje natural (ya que ensucian el lenguaje) y por lo tanto es aconsejable ***eliminarlas***.

- Lista de stop words por idioma
  - https://countwordsfree.com/stopwords/

Stop words

```
1 nltk.download("stopwords") # Descargamos las stop words
2 stop_words_english = set(stopwords.words("english"))
3 print(f"Cantidad de stop words en ingles: {len(stop_words_english)}")
4 print(f"Algunos ejemplos: {list(stop_words_english)[:20]}")
```

```
Cantidad de stop words en ingles: 179
Algunos ejemplos: ['am', 've', 'aren', 'than', "didn't", 'so', 'above', 'few', 'from', "yo
```

```
1 stop_words_spanish = set(stopwords.words("spanish"))
2 print(f"Cantidad de stop words en español: {len(stop_words_spanish)}")
3 print(f"Algunos ejemplos: {list(stop_words_spanish)[:20]}")
```

```
Cantidad de stop words en español: 313
Algunos ejemplos: ['habría', 'estuviesen', 'habrían', 'fuerais', 'hubiésemos', 'tendríamos
```

Stop words

```python
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words_spanish = set(stopwords.words("english"))
filtered_list_eng = []
for word in words_lst:
  # casefold es una manera de trabajar indistintamente mayusculas y minusculas
  # ya que las stop words están todas en minúscula
   if word.casefold() not in stop_words_english:
        filtered_list_eng.append(word)

print(f"Cantidad de palabras iniciales: {len(words_lst)}")
print(words_lst)
print('\n')
print(f"Cantidad de palabras luego de remover stop words: {len(filtered_list_eng)}")
print(filtered_list_eng)
```

```
Cantidad de palabras iniciales: 67
['Natural', 'language', 'processing', '(', 'NLP', ')', 'is', 'an', 'interdisciplinary', 'subfield',


Cantidad de palabras luego de remover stop words: 50
['Natural', 'language', 'processing', '(', 'NLP', ')', 'interdisciplinary', 'subfield', 'computer',
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Vectorization

La vectorización consiste en convertir un input crudo (usualmente texto) en vectores de números reales de manera que un algoritmo de machine learning pueda consumirlos y hacer predicciones.

Es un paso dentro de la **extracción de features**. La idea principal es obtener información importante a partir del texto que sean útiles para entrenar el modelo.

Existen diferentes técnicas de vectorización:

- Bag of Word (BoW)
- Term Frequency-Inverse Document Frequency (TF-IDF)
- Word2Vec
- GLove
- Fastext
- Capas de embeddings

Bag of Words

Es la más simple de todas las técnicas. Se basa en 3 pasos principales:

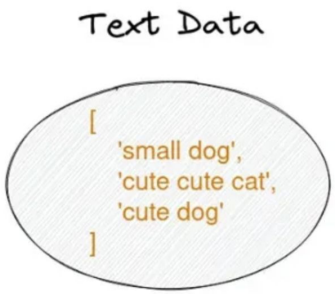
1. *Tokenization*
2. *Creación del Vocabulario.*

   A partir de todos los tokens, se seleccionan todos los tokens únicos y se los ordena en orden alfabético.

3. *Creación del vector.*

   Se crea una matriz esparsa o rala (i.e. con muchos ceros) donde ***cada fila es un vector*** (oración) cuya longitud, es decir las columnas de la matriz, es igual al **tamaño del vocabulario.**

   De esta manera, dada una oración, la fila estará compuesta por ceros y unos donde los 1's identifican tokens de la oración que están presentes en el vocabulario.



cat	cute	dog	small
0	0	1	1
1	2	0	0
0	1	1	0

Bag of Words

```
 1 sentences = sent_tokenize(new_text_example)
 2 sentences
```

```
['\nIn 2003, word n-gram model, at the time the best statistical algorithm, was overperformed by a
multi-layer perceptron (with a single hidden layer and context length of several words trained on up to
14 million of words with a CPU cluster in language modelling) by Yoshua Bengio with co-authors.',
 '[8]\n\nIn 2010, Tomáš Mikolov (then a PhD student at Brno University of Technology) with co-authors
applied a simple recurrent neural network with a single hidden layer to language modelling,=[9] and in
the following years he went on to develop Word2vec.',
 'In the 2010s, representation learning and deep neural network-style (featuring many hidden layers)
machine learning methods became widespread in natural language processing.']
```

```
count_vectorizer = CountVectorizer()
X = count_vectorizer.fit_transform(sentences)
X = X.toarray()
X
```

```
ray([[1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 2, 1, 1, 1, 1, 0, 0, 0, 0,
      1, 0, 1, 2, 1, 2, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 2, 1,
      1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 2, 0, 1, 1, 0, 1, 0, 1, 1,
      0, 0, 3, 1, 0, 2, 0, 1],
     [0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1,
      0, 1, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1,
      0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 2, 1, 0, 1, 0, 0,
      1, 0, 2, 0, 1, 0, 1, 0],
     [0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
      0, 0, 1, 2, 1, 0, 1, 2, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
      0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 1, 0, 0, 0, 0, 0, 0]])
```

```
 1 X.shape
```

```
(3, 74)
```

Bag of Words

```
1 sentences = sent_tokenize(new_text_example)
2 sentences
```

```
['\nIn 2003, word n-gram model, at the time the best statistical algorithm, was overperformed by a
multi-layer perceptron (with a single hidden layer and context length of several words trained on up to
14 million of words with a CPU cluster in language modelling) by Yoshua Bengio with co-authors.',
 '[8]\n\nIn 2010, Tomáš Mikolov (then a PhD student at Brno University of Technology) with co-authors
applied a simple recurrent neural network with a single hidden layer to language modelling,=[9] and in
the following years he went on to develop Word2vec.',
 'In the 2010s, representation learning and deep neural network-style (featuring many hidden layers)
machine learning methods became widespread in natural language processing.']
```

```
1 count_vectorizer.vocabulary_.keys()
2
```

```
dict_keys(['in', '2003', 'word', 'gram', 'model', 'at', 'the', 'time', 'best', 'statistical',
'algorithm', 'was', 'overperformed', 'by', 'multi', 'layer', 'perceptron', 'with', 'single', 'hidden',
'and', 'context', 'length', 'of', 'several', 'words', 'trained', 'on', 'up', 'to', '14', 'million',
'cpu', 'cluster', 'language', 'modelling', 'yoshua', 'bengio', 'co', 'authors', '2010', 'tomáš',
'mikolov', 'then', 'phd', 'student', 'brno', 'university', 'technology', 'applied', 'simple',
'recurrent', 'neural', 'network', 'following', 'years', 'he', 'went', 'develop', 'word2vec', '2010s',
'representation', 'learning', 'deep', 'style', 'featuring', 'many', 'layers', 'machine', 'methods',
'became', 'widespread', 'natural', 'processing'])
```

```
1 X.shape
```

```
(3, 74)
```

TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) es una estadística numérica que pretende reflejar la importancia de una palabra para un documento.

- Mientras que *Bag of Words* se ocupa de asignar frecuencia a las palabras de un texto tiene la desventaja que las preposiciones, conjunciones o conectores tienen la misma importancia que un adjetivo o sustantivo.

- En TF-IDF las palabras que se repiten con demasiada frecuencia no dominan a las palabras menos frecuentes pero importantes.

TF-IDF

TF-IDF tiene dos partes:

1. **Term Frequency (TF)**

   Puede ser entendido como una normalización del score de frecuencia. Se calcula de la siguiente manera:

$$TF = \left(\frac{\text{Number of times keyword is found in document}}{\text{Number of words in document}} \right)$$

TF-IDF

2. IDF = Inverse Document Frequency

1. **DF = Document Frequency:** Proporción de documentos que contienen la keyword

2. **IDF**

$$IDF = \log\left(\frac{\text{Number of documents}}{\text{Number of documents containing the keyword}}\right)$$

TF-IDF

El score final de TF-IDF se obtiene a través de la fórmula

$$TF\text{-}IDF = TF * IDF$$

Algunos comentarios:

- La intuición detrás de TF-IDF es que mientras más común es una palabra a través de todos los documentos menos importante se vuelve.
- De esta manera, mientras más alto es el score más importante es la palabra.
- En la fórmula de IDF se utiliza el logaritmo para atenuar el efecto de IDF en el cálculo final.

TF-IDF

```
from sklearn.feature_extraction.text import  TfidfVectorizer
tfidf = TfidfVectorizer()
transformed = tfidf.fit_transform(sentences)
```

```
df = pd.DataFrame(transformed[0].T.todense(),
      index=tfidf.get_feature_names_out(), columns=["TF-IDF"])
df = df.sort_values('TF-IDF', ascending=False)
df.shape
```

```
(74, 1)
```

```
df.head()
```

	TF-IDF
with	0.326879
by	0.286538
words	0.286538
layer	0.217920
of	0.217920

Word2VEc

- Surgió a partir de un paper de Google en 2013 y revolucionó el estado del arte.
- Tanto en Bag of Words como en TF-IDF cada palabra es tratada como una entidad individual y el contexto semántico se ignora completamente.
- Word2Vec **es la primera representación vectorial que tiene en cuenta el contexto.**
- De esta manera, si los vectores viven en un mismo espacio n-dimensional ***aquellas palabras con un significado similar deberían estar cerca unas de otras en este espacio.*** O dicho de otra manera, palabras que aparecen en contextos similares tienen representaciones vectoriales cercanas en el espacio.

Efficient Estimation of Word Representations in Vector Space

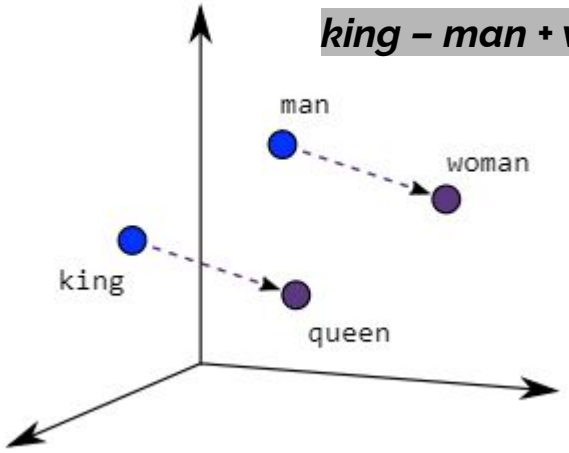
Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

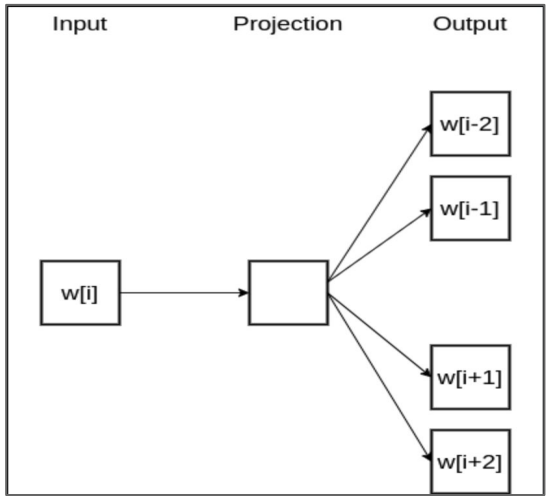
king – man + woman = queen



Word2VEc

A. Skip-Gram:

Se le proporciona una palabra de un oración a una Red Neuronal y se intenta **predecir el contexto** de esa palabra en esa oración



Palabra actual

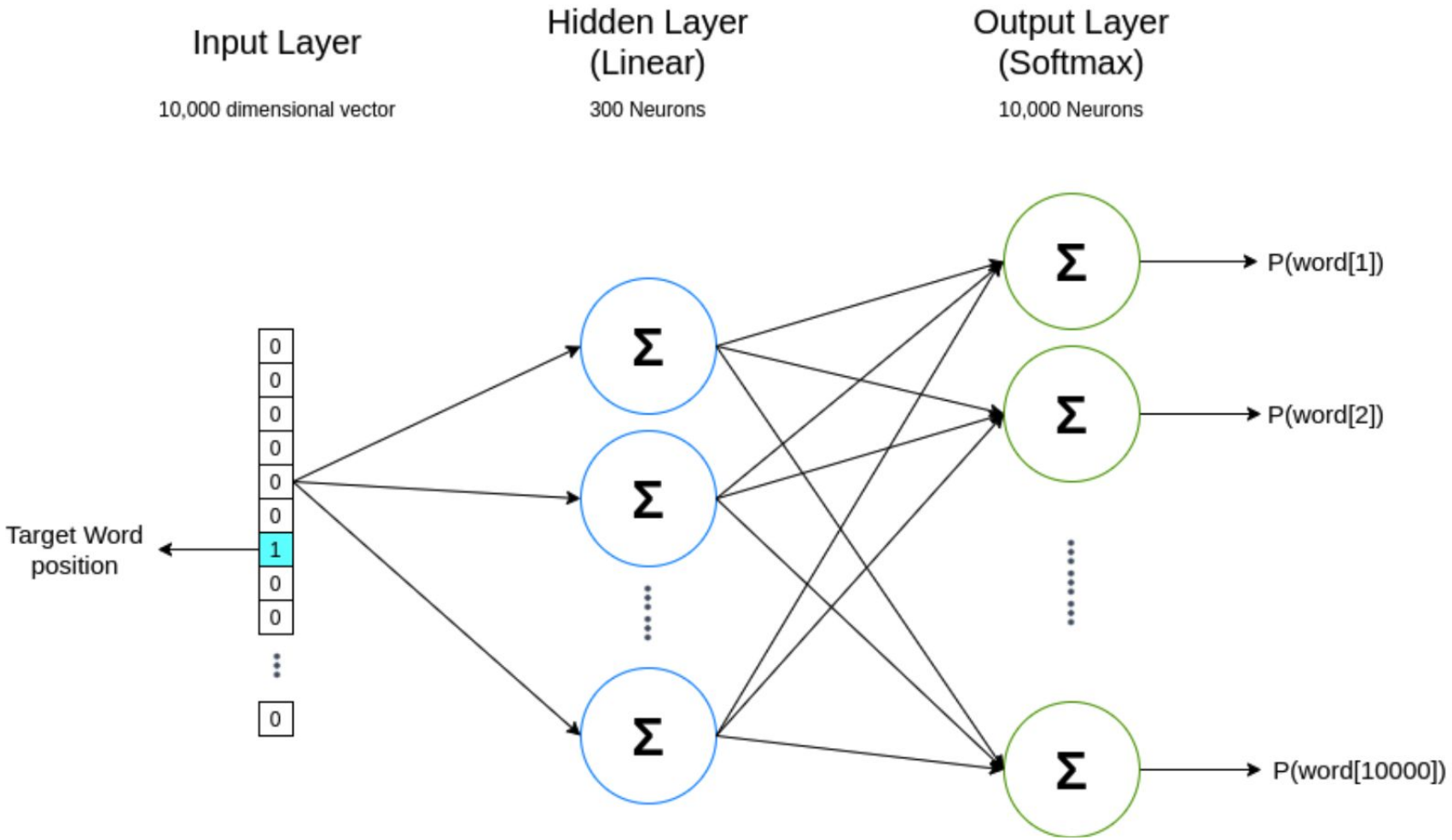
Palabras anteriores

Palabras posteriores

Lo interesante es que en realidad **no se utiliza esta red neuronal entrenada.**

En cambio, el objetivo es simplemente **aprender los pesos de la capa oculta** mientras se predicen correctamente las palabras circundantes. Estos pesos son los **embeddings de las palabras.**

Word2VEc

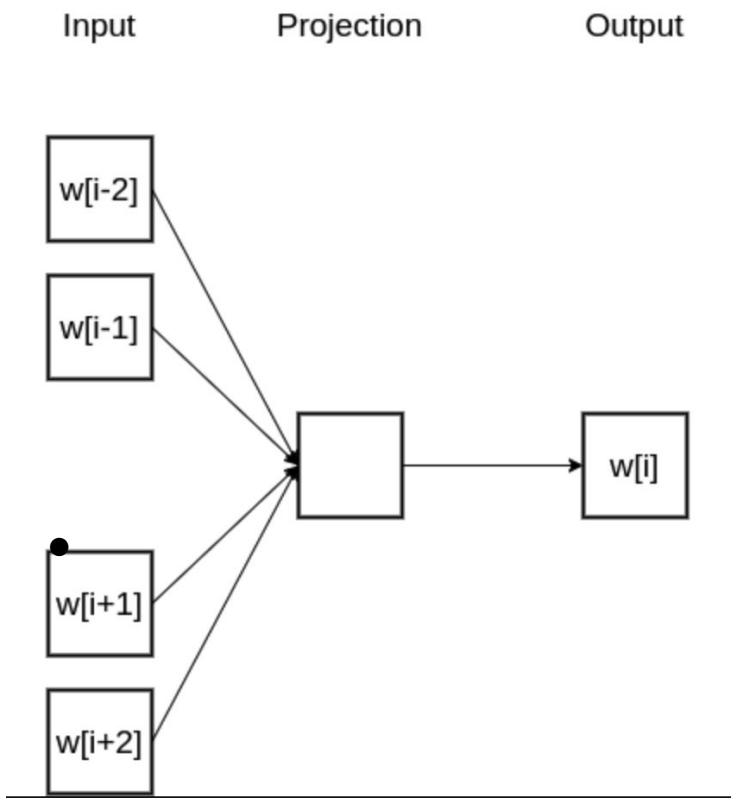

Input Layer
10,000 dimensional vector
Hidden Layer
(Linear)
300 Neurons
Output Layer
(Softmax)
10,000 Neurons
Target Word
position
0
0
0
0
0
0
1
0
0
0
Σ
Σ
Σ
Σ
Σ
Σ
P(word[1])
P(word[2])
P(word[10000])

Word2VEc

B. CBOW

La traducción de CBOW es “Bolsa Continua de Palabras”

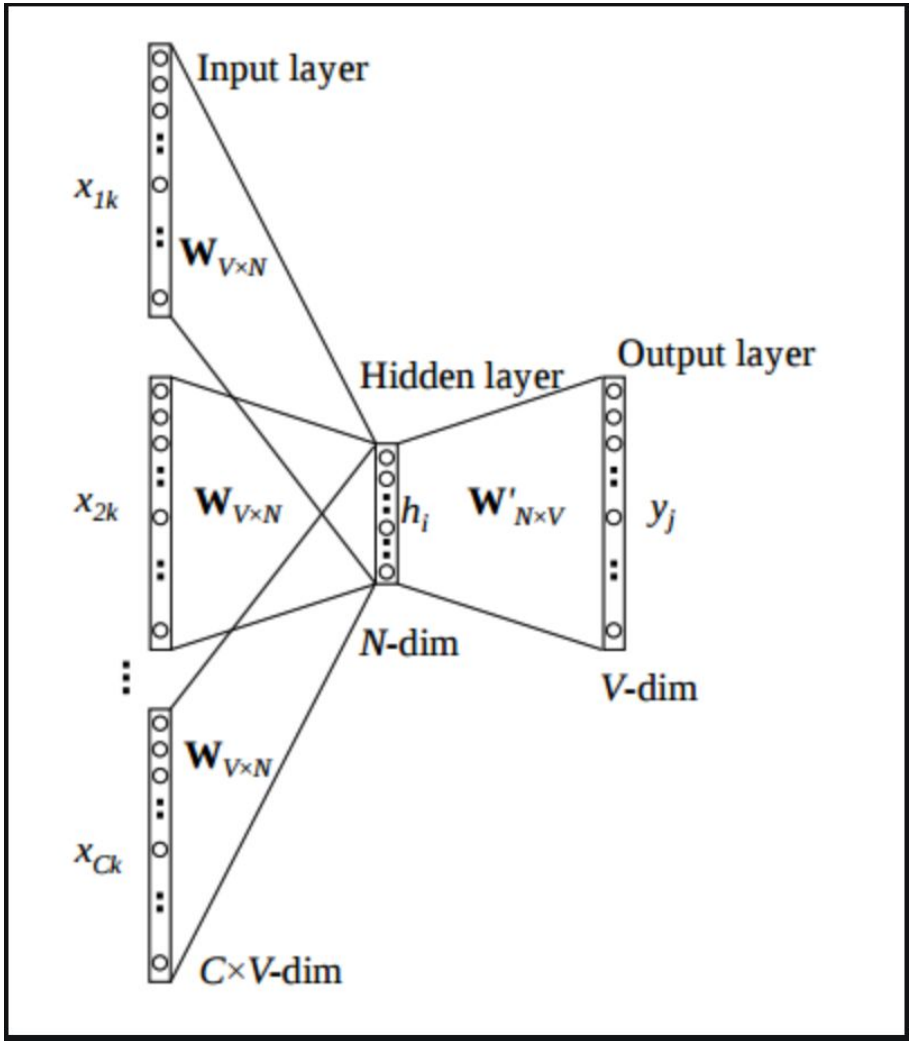
En este enfoque, en lugar de predecir las palabras de contexto, se ingresan estas palabras al modelo y se le pide a la red que prediga la palabra actual.



¿skip-gram o CBOW?

- Skip-gram trabaja bien con pequeños datasets y puede representar mejor las palabras.
- CBOW entrena más rápido que skip-gram y puede representar mejor las palabras frecuentes.

Word2Vec: Arquitectura de CBOW



CBOW Model Architecture

Glove

Glove (Global Vector for words representations) fue desarrollado en Stanford luego de la publicación de Word2Vec y también se encuentra enfocado en crear embeddings en base al contexto.

GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning
Computer Science Department, Stanford University, Stanford, CA 94305
jpennin@stanford.edu, richard@socher.org, manning@stanford.edu

- **Word2vec es un método basado en ventanas**, en el que el modelo se basa en información local para generar embeddings, con lo cual el resultado está limitado por el tamaño de ventana elegido.
- De esta manera, **en Wor2vec la semántica aprendida por una palabra se ve afectada por las palabras que la rodean en la oración original.**
- **Glove captura estadísticas globales y locales para generar los embeddings.**
- El comportamiento semántico que obtiene Glove se obtiene entrenando una **matriz de co-ocurrencias** que se construye con la idea de que co-ocurruencias palabra-palabra son piezas vitales de información.

Glove: Puntos a favor

- Se descubrió que Glove supera a otros modelos en tareas de analogía, similitud de palabras y reconocimiento de entidades nombradas (named entity recognition).
- Dado que incorpora estadísticas globales, puede capturar la semántica de palabras raras y funciona bien incluso en un corpus pequeño.

Fastext

- Fue introducido por Facebook en 2016 y mejoró respecto a los métodos anteriores en cuanto representación de palabras desconocidas.

- En lugar de utilizar palabras para crear embeddings, utiliza **caracteres**. Es decir, las representaciones obtenidas son a nivel letras.

- Los embeddings de palabras obtenidos vía Fastext no se obtienen directamente sino que son una **combinación de los embeddings de las subpalabras.**

- Usar caracteres en lugar de palabras tiene otra ventaja. Se necesitan menos datos para el entrenamiento, ya que una palabra se convierte en cierto modo en su propio contexto, lo que da como resultado mucha más información que se puede extraer de un fragmento de texto.

Bag of Tricks for Efficient Text Classification

Armand Joulin **Edouard Grave** **Piotr Bojanowski** **Tomas Mikolov**
Facebook AI Research
{ajoulin,egrave,bojanowski,tmikolov}@fb.com

Enriching Word Vectors with Subword Information

Piotr Bojanowski* and **Edouard Grave*** and **Armand Joulin** and **Tomas Mikolov**
Facebook AI Research
{bojanowski,egrave,ajoulin,tmikolov}@fb.com

Fastext: n-gramas

Word	n-gram	combinations
reading	3	<re, rea, ead, adi, din, ing, ng>
reading	4	<rea, read, eadi, adin, ding, ing>
reading	5	<read, readi, eadin, ading, ding >
reading	6	<readi, readin, eading, ading>

Material Adicional

[A3 Procesamiento de Lenguaje Natural 2024]