NLP

Word Embeddings

Msc. Rodrigo Cardenas Szigety
rodrigo.cardenas.sz@gmail.com

Programa de la materia

Clase 1: Introducción a NLP, Vectorización de documentos.
Clase 2: Preprocesamiento de texto, librerías de NLP y Rule-Based Bots.
Clase 3: Word Embeddings, CBOW y SkipGRAM, representación de oraciones.
Clase 4: Redes recurrentes (RNN), problemas de secuencia y estimación de próxima palabra.
Clase 5: Redes LSTM, análisis de sentimientos.
Clase 6: Modelos Seq2Seq, traductores y bots conversacionales.
Clase 7: Celdas con Attention. Transformers, BERT & ELMo, fine tuning.
Clase 8: Cierre del curso, NLP hoy y futuro, deploy.

*Unidades con desafíos a presentar al finalizar el curso.
*Último desafío y cierre del contenido práctico del curso.

Problemas con word2vec/TF-IDF

Palabras u oraciones de significado similar son “ortogonales”

Viajando en colectivo

Voy arriba del bus

La dimensión de los vectores depende de la dimensión del vocabulario

La representación es esparsa

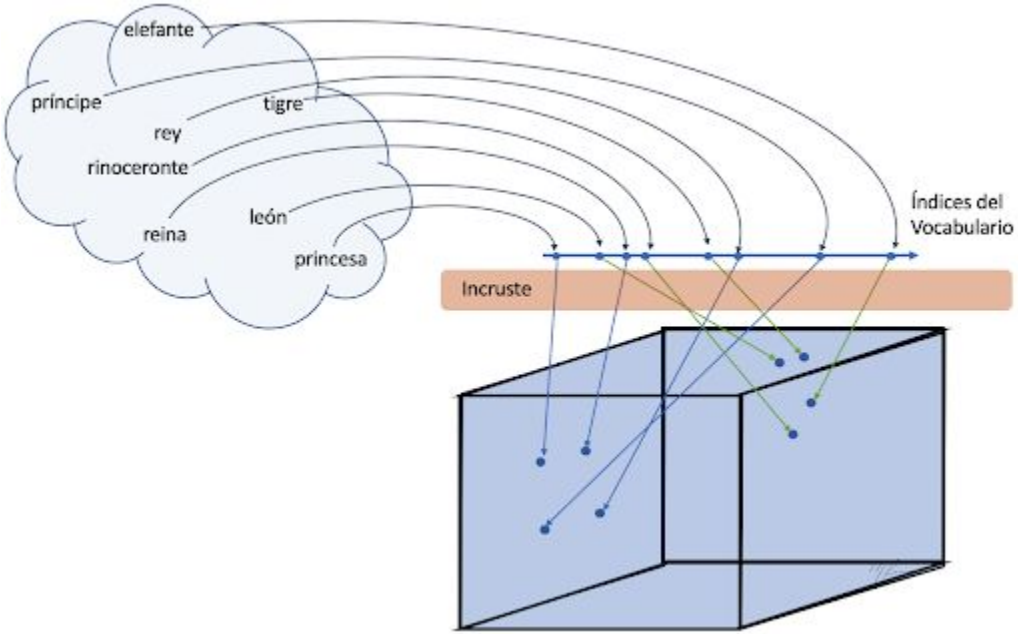
	cat	the	quick	brown	fox	jumped	over	dog	bird	flew	...	kangaroo	house
	0	1	0	0	0	0	0	0	0	0	...	0	0
	0	0	1	0	0	0	0	0	0	0	...	0	0
	0	0	0	1	0	0	0	0	0	0	...	0	0
	0	0	0	0	1	0	0	0	0	0	...	0	0
	0	0	0	0	0	1	0	0	0	0	...	0	0
	0	0	0	0	0	0	1	0	0	0	...	0	0
	0	1	0	0	0	0	0	0	0	0	...	0	0
	0	0	0	1	0	0	0	0	0	0	...	0	0
	0	0	0	0	0	0	0	1	0	0	...	0	0

time

Dictionary Size

Embeddings

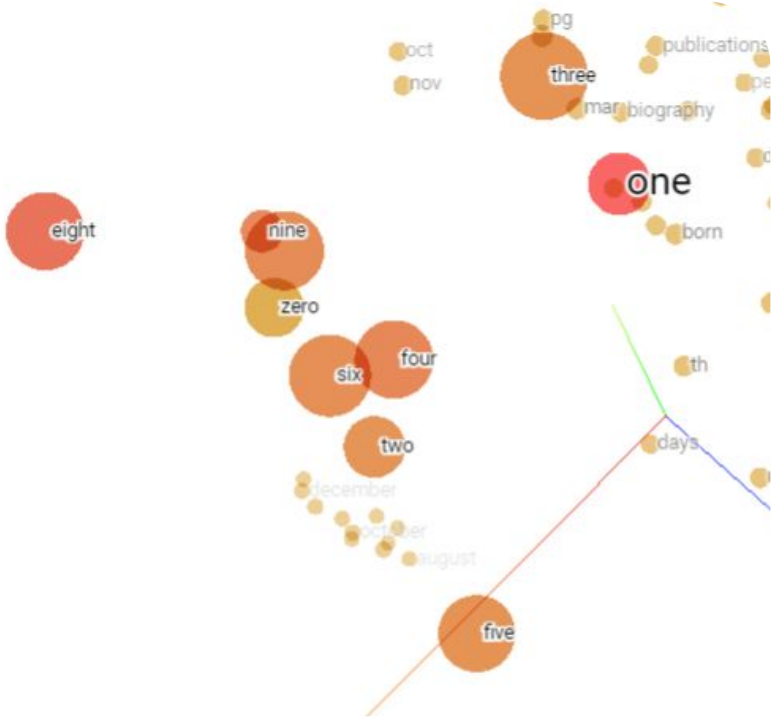
Un embedding es la representación numérica densa de tamaño fijo de un dato estructurado o no estructurado (mapear imagenes o palabras a números)



Word Embeddings

Las palabras que tienen un significado similar tendrán una representación similar como embeddings

http://projector.tensorflow.org/

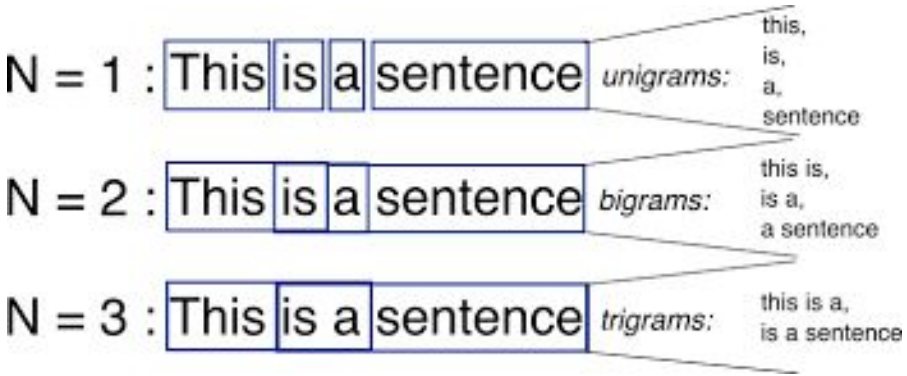


Para qué podemos utilizar word Embeddings

N-GRAM

"Subsecuencia de N elementos de una secuencia dada"

Otra forma de agrupar las palabras distinto a word2vec (N=1, unigram) en donde se busca aumentar el poder de generalización y a su vez poder hacer más variado el vocabulario.



Character-level unigrams

Text	Token Sequence	Token Value
Dogs	1	D
Dogs	2	o
Dogs	3	g
Dogs	4	s

Character-level bigrams

Text	Token Sequence	Token Value
Dogs	1	Do
Dogs	2	og
Dogs	3	gs

Character-level trigrams

Text	Token Sequence	Token Value
Dogs	1	Dog
Dogs	2	ogs

GloVe y fastText

Embeddings pre-entrenados basados en diferentes topologías:

GloVe

- Entrenado con textos de Wikipedia, Common Crawl y GigaWord 5
- Modelo basado en CBOW y Skip-Gram
- Basado en word2vec (primera implementación de CBOW y Skip-Gram)

fastText

- Basado en N-GRAM de caracteres en vez de palabras permitiendo entender mejor los sufijos y prefijos
- Entrenado con una colección de 8 corpus (portales de noticias, reviews, WIkipedia)
- Permite crear mejores embeddings para palabras “raras” (basado CBOW o Skip-Gram)
- Puede crear un embedding de una palabra que nunca vió

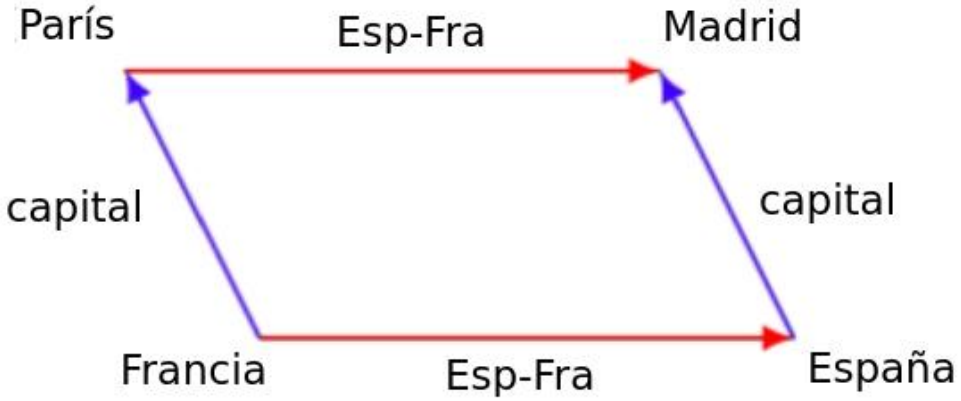
Operaciones con Embeddings: tests de analogías

Una forma de testear la calidad de embeddings es probar su desempeño en tests de analogías:

París es a Francia lo que Madrid es a España.
Madrid y París corresponden a España y Francia

$$\vec{paris} - \vec{francia} \approx \vec{madrid} - \vec{espana}$$

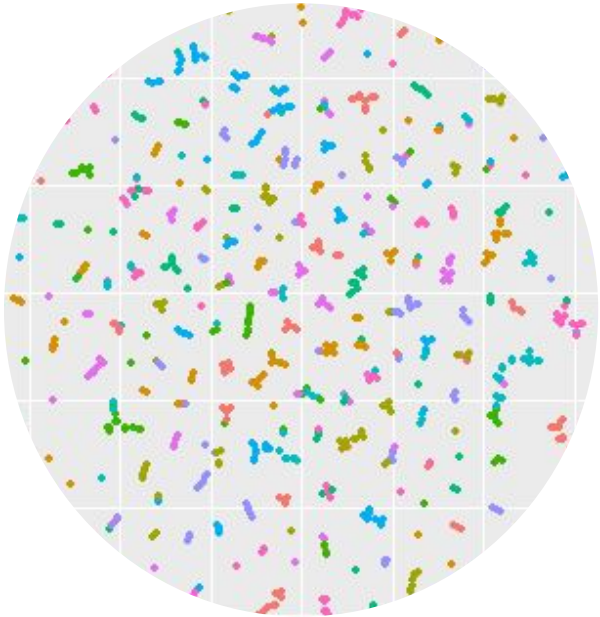
$$simcos(\vec{paris} - \vec{francia}, \vec{madrid} - \vec{espana}) \approx 1$$



t-SNE (t-distributed stochastic neighbor embedding)

"Técnica de reducción de dimensionalidad especialmente para graficar en 2D embeddings (vectores de muchas dimensiones)" LINK

- Balancea las características locales y globales de los datos (perplexity)
- No produce siempre el mismo resultado
- Modifica las distancias originales de los datos a fin de priorizar una mejor visualización o interpretación visual



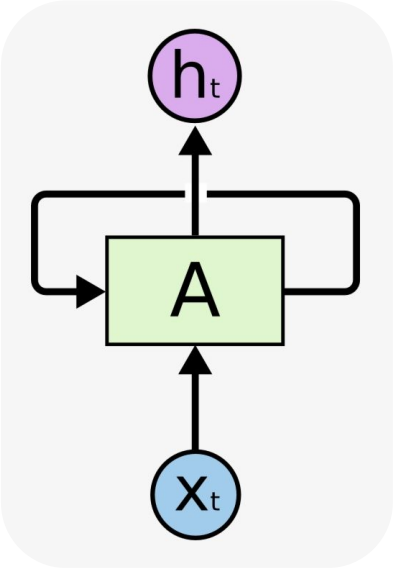
Embeddings Glove y Fasttext

Link al Colab

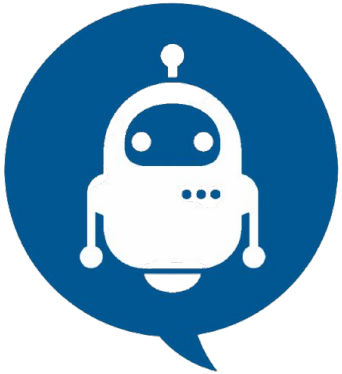
LINK

¿Dónde utilizaremos Embeddings?

Redes RNN



Seq2Seq

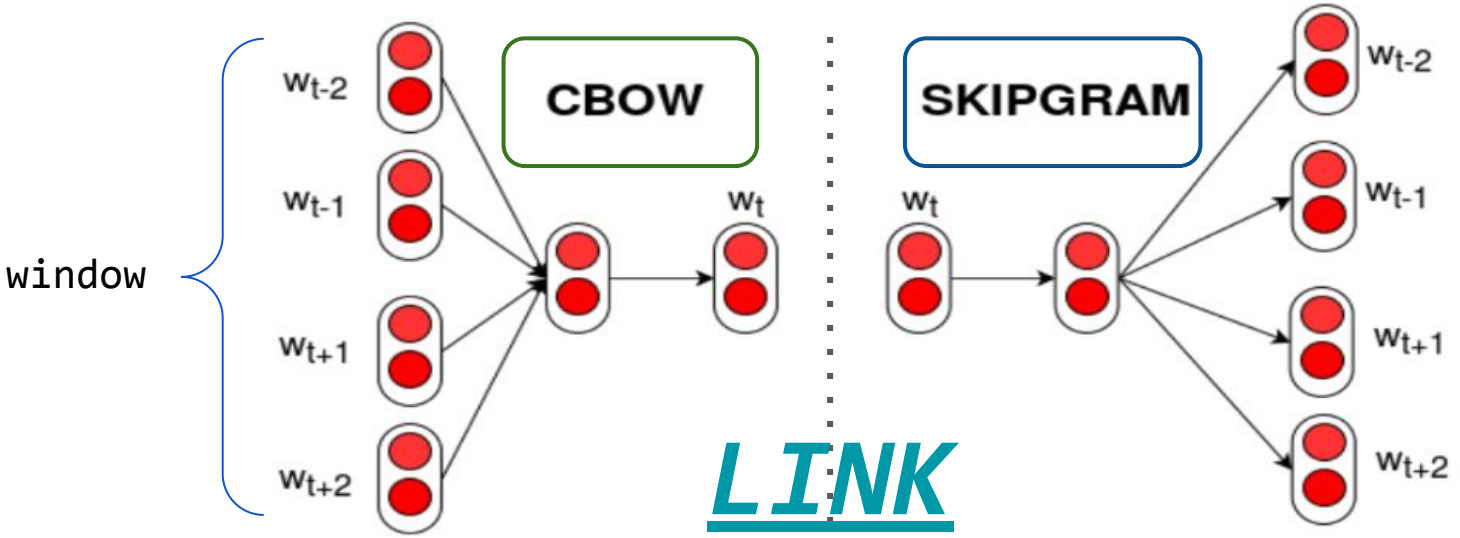


BERT

Serán los pilares de todo lo que construyamos en el resto del curso

¿Cómo podemos crear nuestros word Embeddings?

Aprendiendo (con redes neuronales) vectores para cada palabra que maximicen la relación entre las palabras de contexto y la palabra objetivo,



LINK

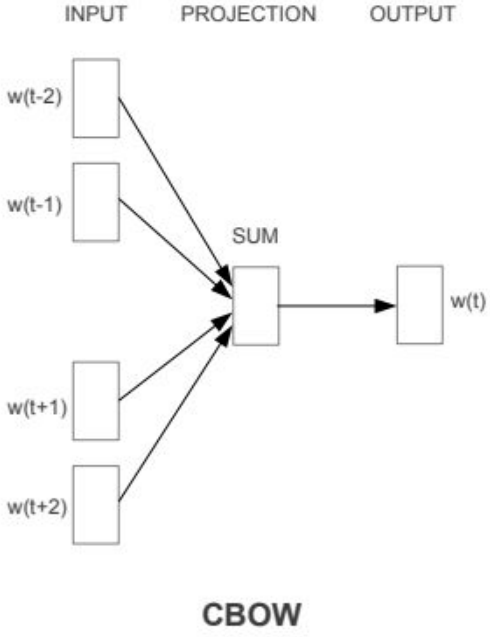
$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_t|w_{t+j})$$

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|w_t)$$

Se utiliza ***OneHotEncoding*** como representación del texto a la entrada del modelo

Continuous Bag of Words Model (CBOW)

Utiliza como entrada el contexto de la palabra objetivo (palabras a izquierda y derecha de ella). El tamaño de la ventana determina cuántas palabras se tomarán para contextualizar el embedding.

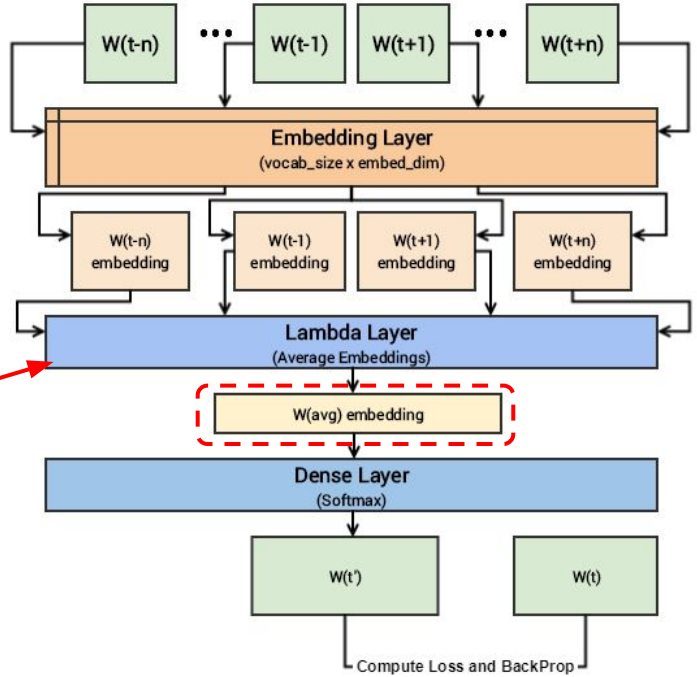


CBOW - Entrenamiento

LINK

Para entrenar necesitamos tener el vocabulario del corpus y las sentencias organizadas por el tamaño de la ventana de entrada.

Los embeddings de cada palabra son el embedding promedio de todas las veces que se utilizó en el corpus.

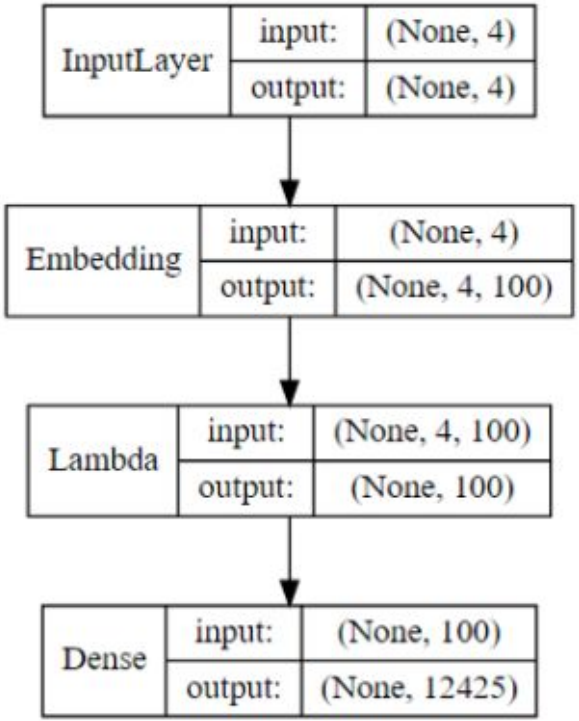



```
cbow = Sequential()
cbow.add(Embedding(input_dim=vocab_size, output_dim=embed_size, input_length=window_size*2))
cbow.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(embed_size,)))
cbow.add(Dense(vocab_size, activation='softmax'))
cbow.compile(loss='categorical_crossentropy', optimizer='rmsprop')
```

CBOW - Entrenamiento

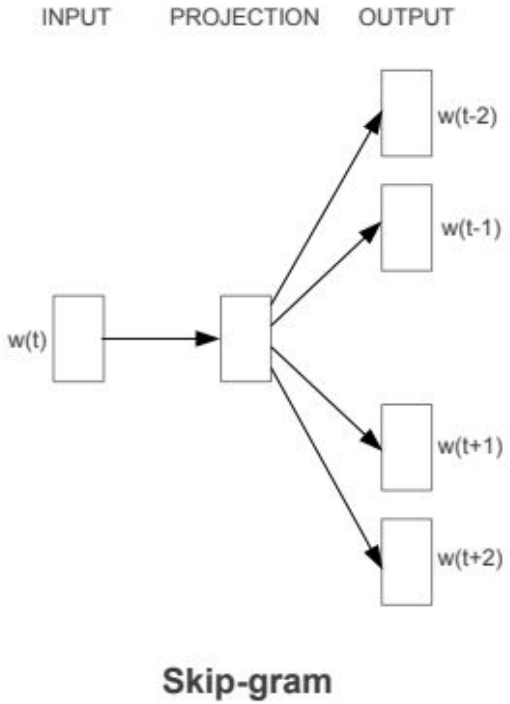
Con tan solo un corpus de 12425 palabras distintas y embedding de 100 dimensiones hay que entrenar **2.5 Millones de parámetros**

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 4, 100)            1242500
_________________________________________________________________
lambda_1 (Lambda)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 12425)             1254925
=================================================================
Total params: 2,497,425
Trainable params: 2,497,425
Non-trainable params: 0
_________________________________________________________________
```



Skip-Gram

Al contrario de CBOW, este modelo intenta predecir las palabras que rodean (contexto) a una palabra objetivo. Se divide el output como pares [target, context]



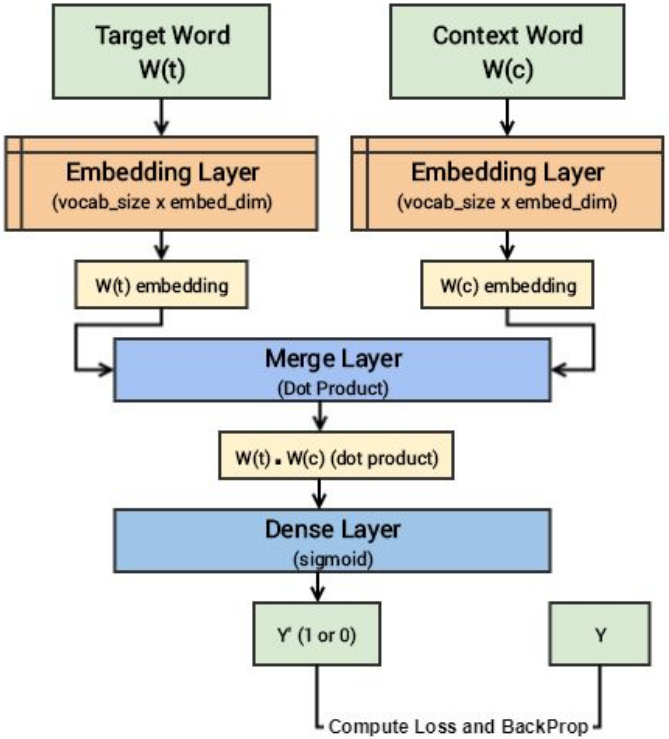
I will have orange juice and eggs for breakfast	(will, I)	(will, have)	(will, orange)	
I will have orange juice and eggs for breakfast	(have, I)	(have, will)	(have, orange)	(have, juice)
I will have orange juice and eggs for breakfast	(orange, will)	(orange, have)	(orange, juice)	(orange, and)
I will have orange juice and eggs for breakfast	(juice, have)	(juice, orange)	(juice, and)	(juice, eggs)
I will have orange juice and eggs for breakfast	(and, orange)	(and, juice)	(and, eggs)	(and, for)
I will have orange juice and eggs for breakfast	(eggs, juice)	(eggs, and)	(eggs, for)	(eggs, breakfast)
I will have orange juice and eggs for breakfast	(for, and)	(for, eggs)	(for, breakfast)	

Skip-Gram - Entrenamiento

LINK

Por cada par [target, context] el sistema determina si las palabras tiene significado en contexto (1) o no lo tiene (0), buscando así acercar las palabras que tienen significado juntas (que se espera que estén juntas en el texto)

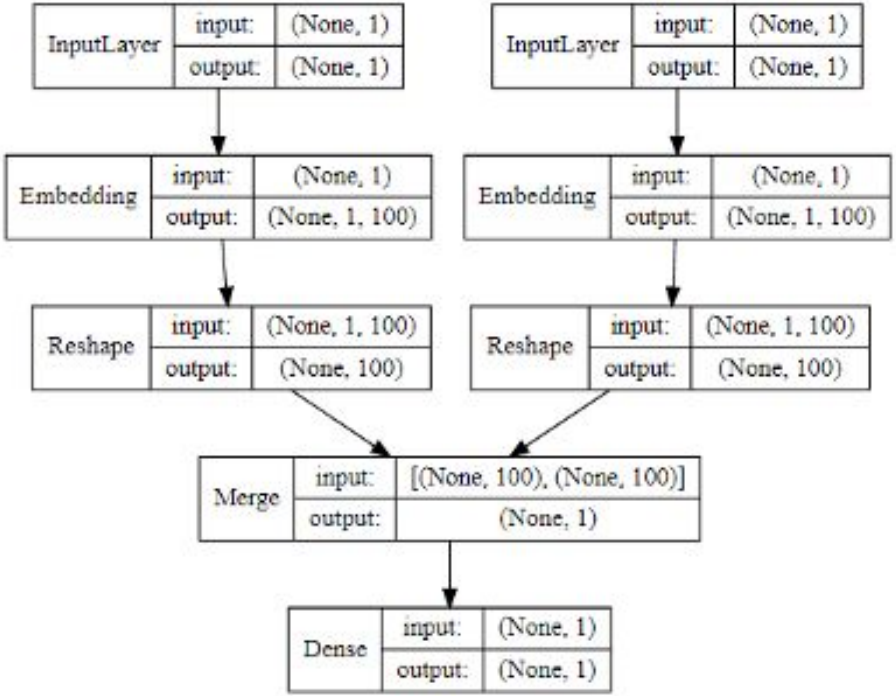
```
word_model = Sequential()
word_model.add(Embedding(vocab_size, embed_size,
                         embeddings_initializer="glorot_uniform",
                         input_length=1))
word_model.add(Reshape((embed_size, )))

context_model = Sequential()
context_model.add(Embedding(vocab_size, embed_size,
                  embeddings_initializer="glorot_uniform",
                  input_length=1))
context_model.add(Reshape((embed_size,)))

model = Sequential()
model.add(Merge([word_model, context_model], mode="dot"))
model.add(Dense(1, kernel_initializer="glorot_uniform", activation="sigmoid"
model.compile(loss="mean_squared_error", optimizer="rmsprop")
```



Skip-Gram - Entrenamiento

Skip-Gram requiere más datos para lograr un buen resultado pero obtiene más información sobre el contexto del corpus en sus embeddings.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
merge_2 (Merge)              (None, 1)                 0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 2
=================================================================
Total params: 2,485,002
Trainable params: 2,485,002
Non-trainable params: 0
_________________________________________________________________
```



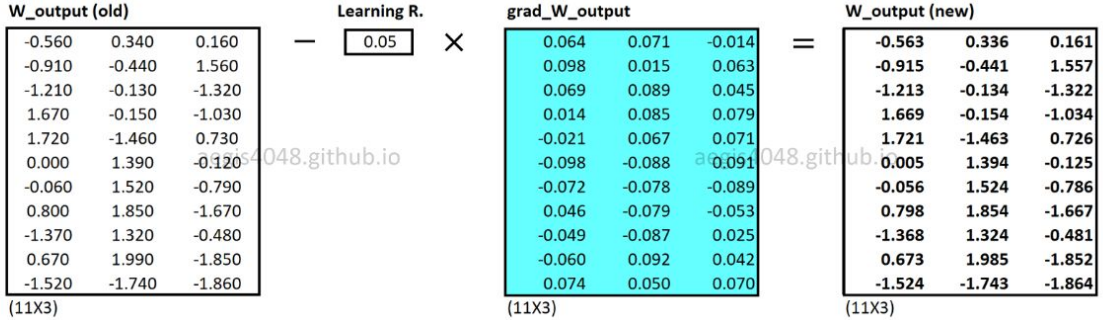
Negative sampling

LINK

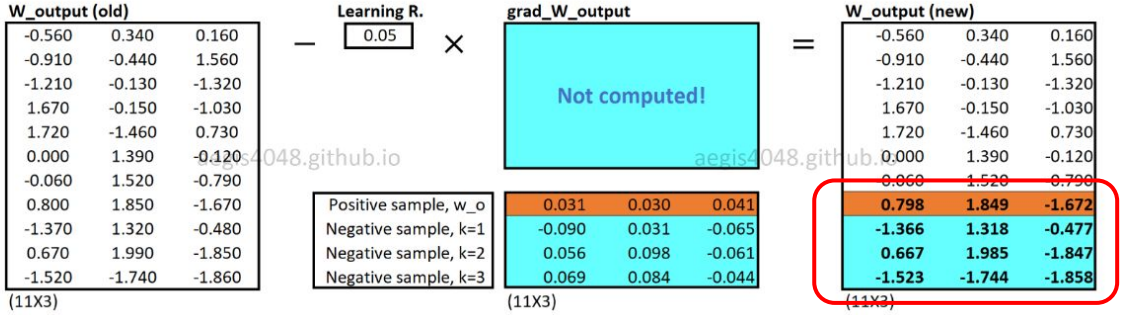
En SkipGram/CBOW la cantidad de parámetros a entrenar en la softmax es enorme:

Parametros = vocab_size * embedding_size → millones de parámetros

Vanilla Skip-Gram



Negative Sampling



En cada iteración se observa la palabras [target, contexto] y "K" palabras aleatorias del corpus
El objetivo es optimizar cómputo.
Además funciona como regularización.

Gensim - Doc2Vec paragraph embeddings

LINK

Utilizaremos esta librería que nos facilita generar embeddings tipo Skip-Gram o CBOW de nuestros corpus

- Librería de Python
- Existe desde 2009
- Muy popular y muy simple de utilizar

Generación de embeddings con Gensim

Desafio

Crear sus propios vectores con Gensim basado en lo visto en clase con otro dataset.
Probar términos de interés y explicar similitudes en el espacio de embeddings. Graficarlos.
Sacar conclusiones

¡Muchas gracias!