

NLP

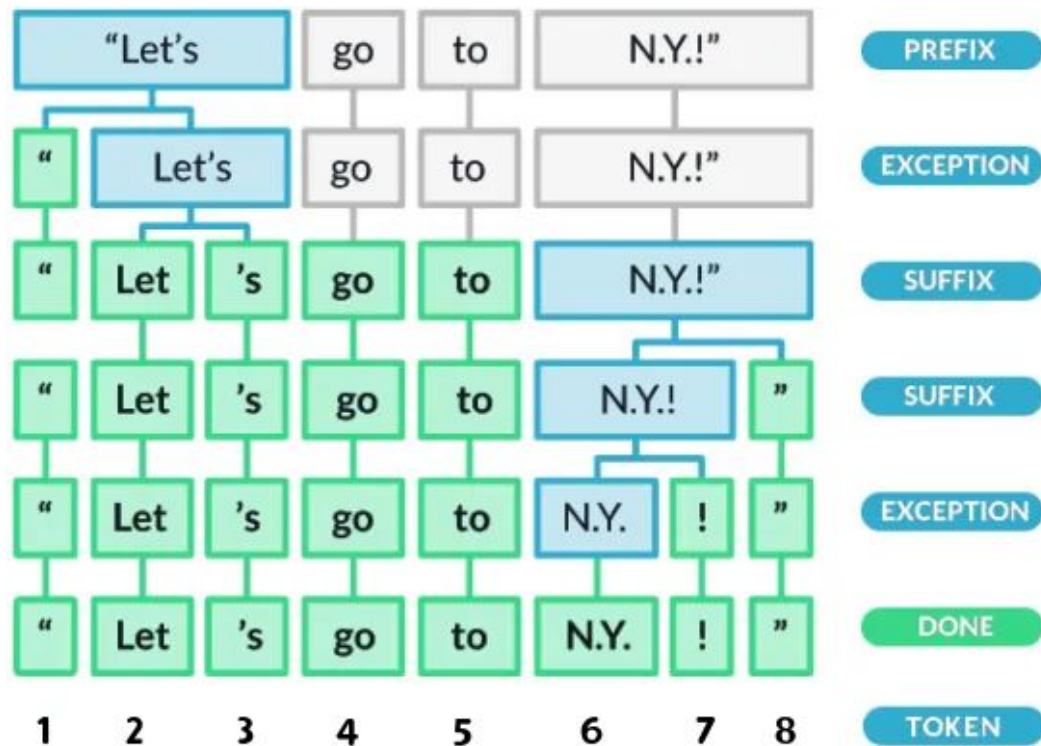
Word Embeddings

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

Segmentar y tokenizar



Proceso en el cual una oración o documento es segmentado en términos individuales. Una vez finalizada la segmentación cada término único es referenciado mediante un token.



Problemas con CountVectorizer/OHE/TF-IDF



Textos de significado similar pueden ser “ortogonales”

Estoy viajando en colectivo

Voy arriba del bus

La dimensión de los vectores depende del tamaño del vocabulario

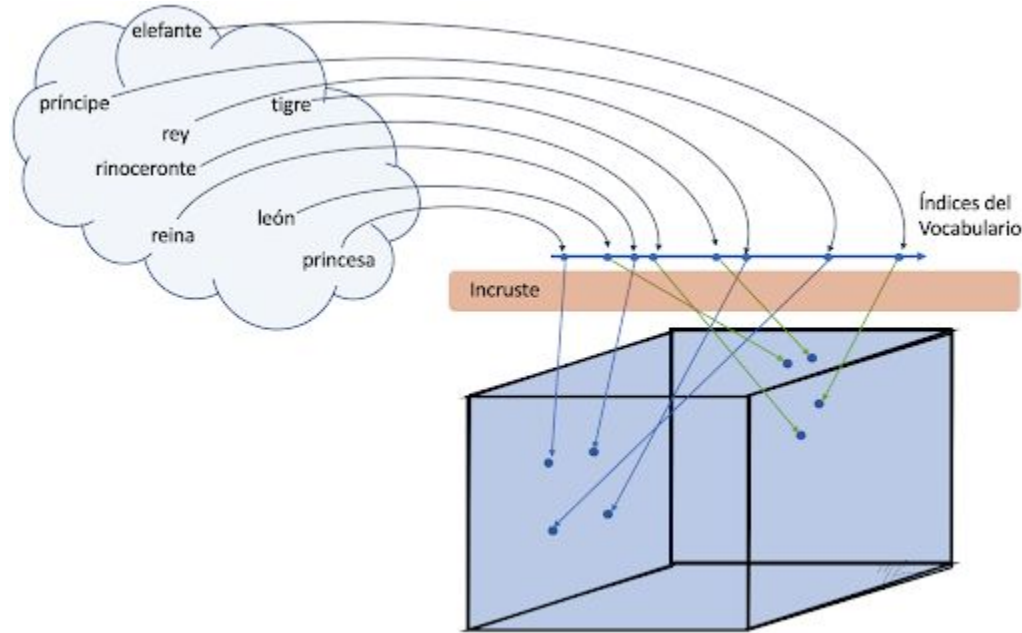
No aprovechamos la dimensionalidad

	cat	the	quick	brown	fox	jumped	over	dog	bird	flew	...	kangaroo	house
time ↓	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
← Dictionary Size →													

Embeddings



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

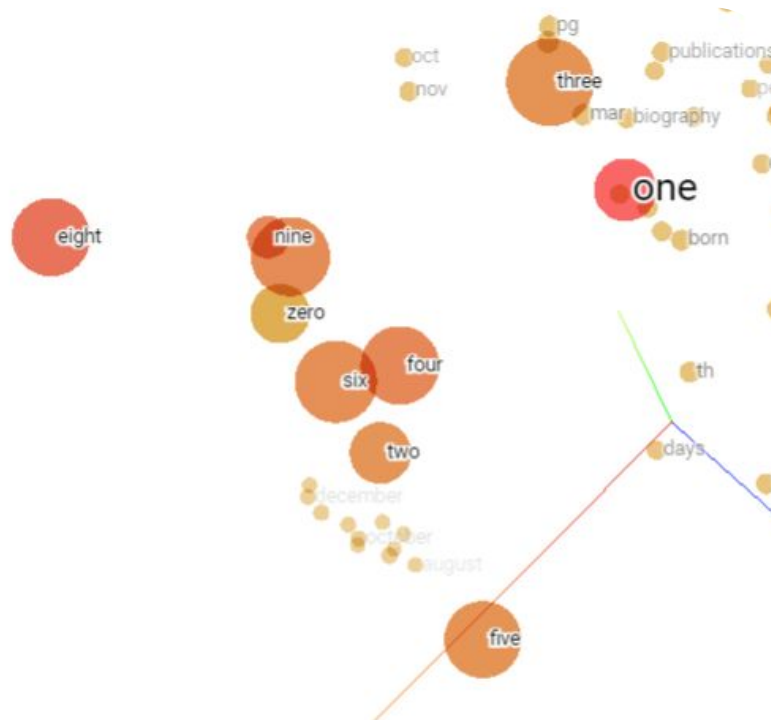


Word Embeddings



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

<http://projector.tensorflow.org/>



GloVe y fastText



Embeddings pre-entrenados basados en diferentes enfoques:

GloVe



Tokenización basada en palabras



Entrenado con textos de Wikipedia, Common Crawl y GigaWord 5



Se basa en calcular la matriz de co-ocurrencia de palabras y estimar el cociente de probabilidad de aparición.

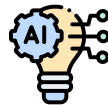
fastText



Tokenización basada en N-Grams de caracteres (3 a 6). Mejora la interpretación de sufijos y prefijos



Entrenado con una colección de 8 corpus (portales de noticias, reviews, Wikipedia)



Basado en word2vec (CBOW/Skip-Gram)



Puede crear un embedding de una palabra que nunca vió

Operaciones con Embeddings: tests de analogías



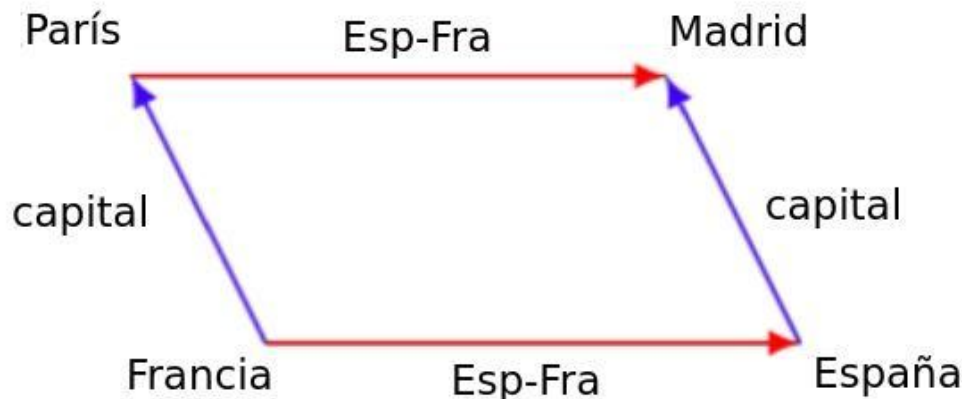
Una forma de testear la calidad de embeddings es probar su desempeño en tests de analogías:

París es a Francia lo que Madrid es a España.

Madrid y París corresponden a España y Francia

$$\overrightarrow{Paris} - \overrightarrow{Francia} \approx \overrightarrow{Madrid} - \overrightarrow{España}$$

$$\text{simcos}(\overrightarrow{Paris} - \overrightarrow{Francia}, \overrightarrow{Madrid} - \overrightarrow{España}) \approx 1$$





Link al Colab

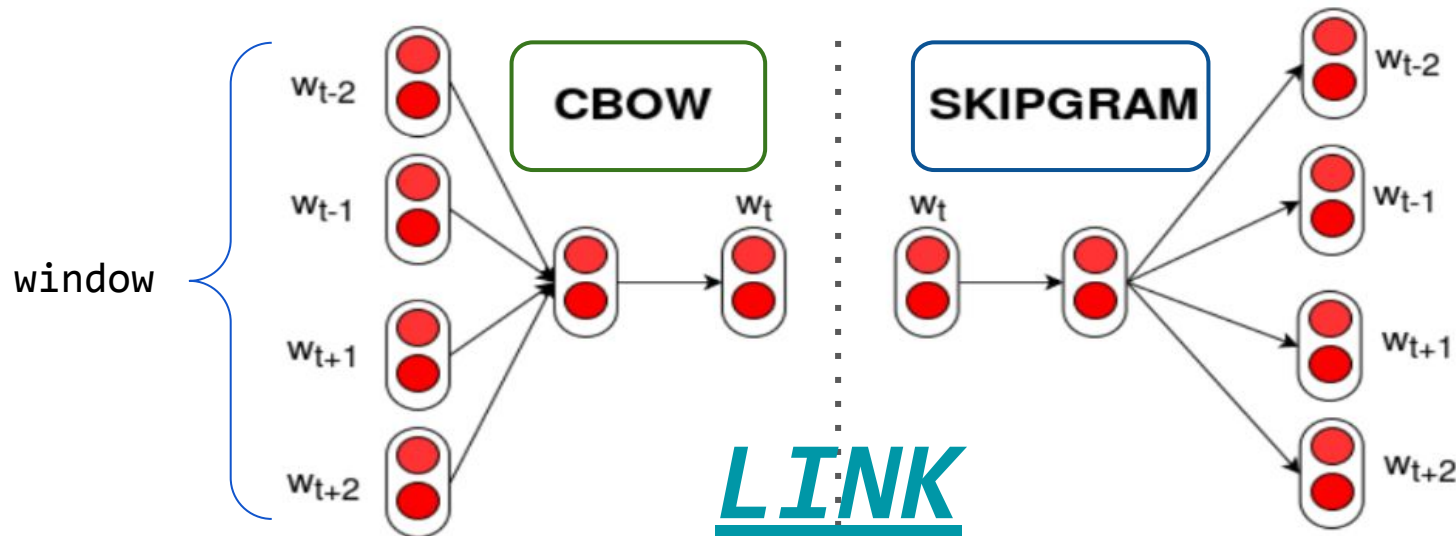


LINK

¿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. Esto es lo que se implementó en la librería **word2vec**.



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

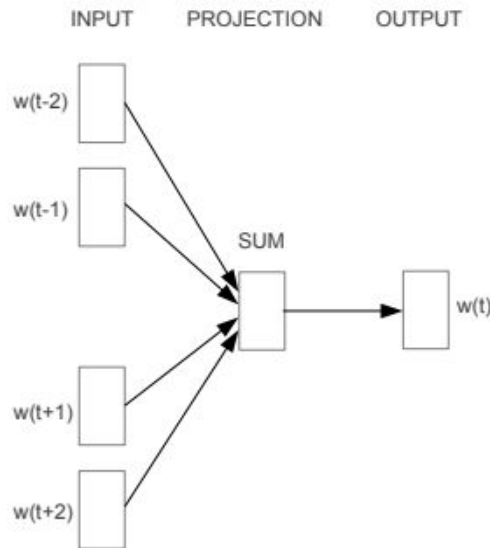
$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_t | w_{t+j})$$

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

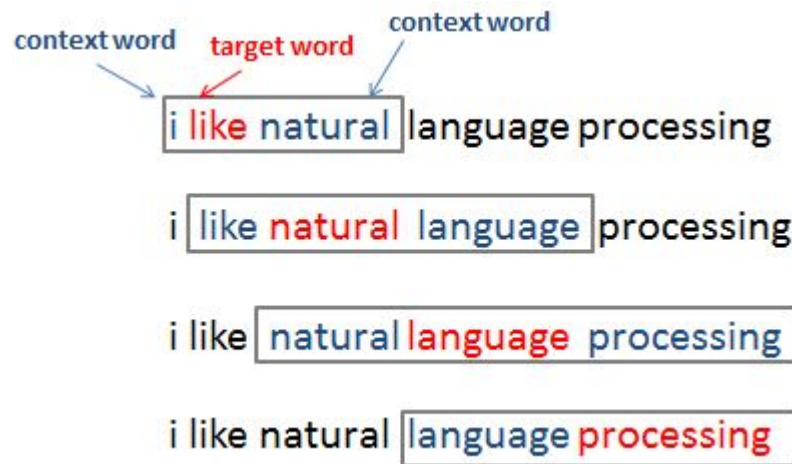
Continuous Bag of Words Model (CBOW)



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



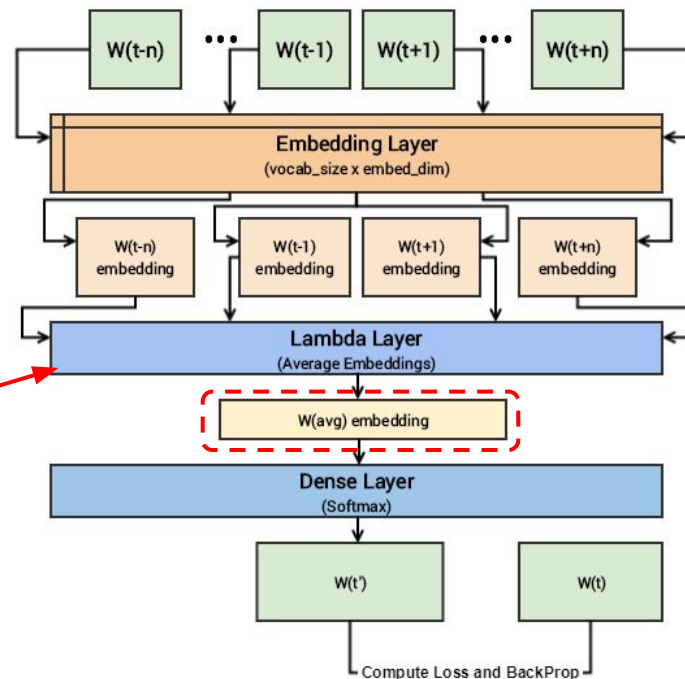
CBOW - Entrenamiento

[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.



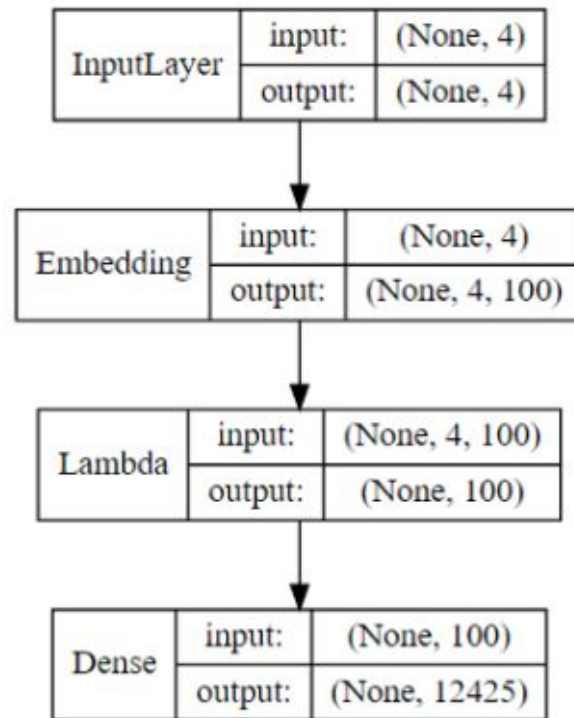
```
cbow = Sequential()
cbow.add(Embedding(input_dim=vocab_size, output_dim=embed_size, input_length=window_size*2))
cbow.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(embed_size,)))
cbow.add(Dense(vocab_size, activation='softmax'))
cbow.compile(loss='categorical_crossentropy', optimizer='rmsprop')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

CBOW - Entrenamiento



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

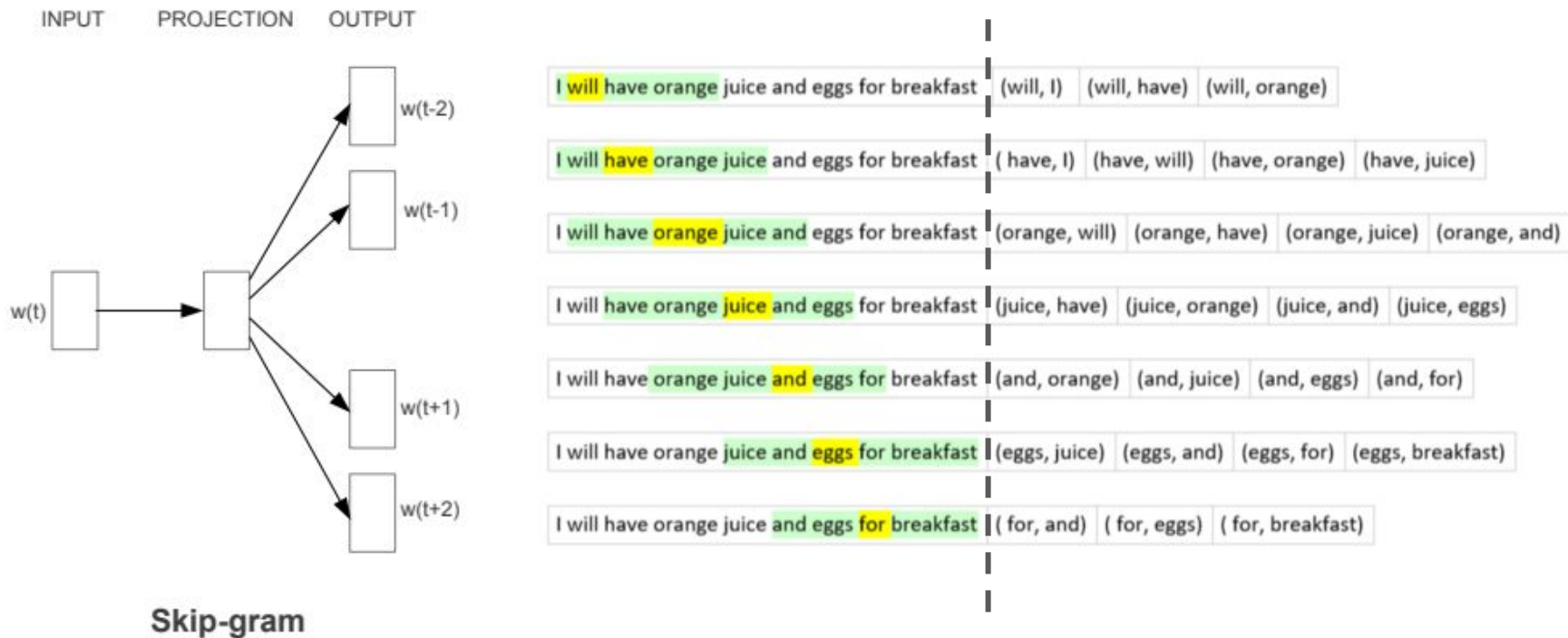
Layer (type)	Output Shape	Param #
=====		
embedding_1 (Embedding)	(None, 4, 100)	1242500
=====		
lambda_1 (Lambda)	(None, 100)	0
=====		
dense_1 (Dense)	(None, 12425)	1254925
=====		
Total params: 2,497,425		
Trainable params: 2,497,425		
Non-trainable params: 0		



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]



Skip-Gram - Entrenamiento

[LINK](#)

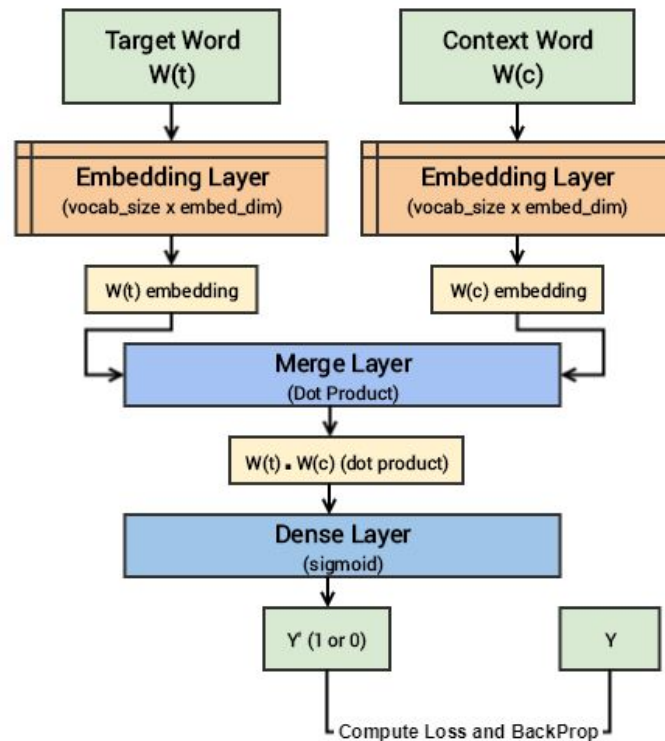


Por cada par [target, context] el sistema determina si las palabras tiene significado en contexto (1) o no lo tiene (0), buscando así acercar las palabras que tienen significado juntas (que se espera que estén juntas en el texto)

```
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"))
```

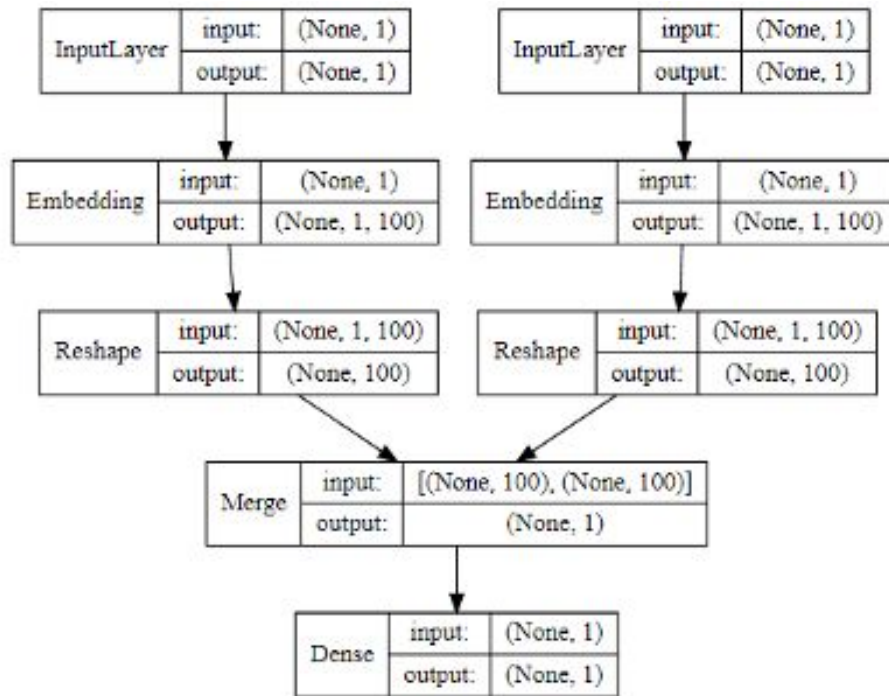


Skip-Gram - Entrenamiento



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:

$\text{Parametros} = \text{vocab_size} * \text{embedding_size} \rightarrow \text{millones de parámetros}$

Vanilla
Skip-Gram

Learning R. \times

-0.560	0.340	0.160
-0.910	-0.440	1.560
-1.210	-0.130	-1.320
1.670	-0.150	-1.030
1.720	-1.460	0.730
0.000	1.390	-0.120
-0.060	1.520	-0.790
0.800	1.850	-1.670
-1.370	1.320	-0.480
0.670	1.990	-1.850
-1.520	-1.740	-1.860

(11X3)

0.064	0.071	-0.014
0.098	0.015	0.063
0.069	0.089	0.045
0.014	0.085	0.079
-0.021	0.067	0.071
-0.098	-0.088	0.091
-0.072	-0.078	-0.089
0.046	-0.079	-0.053
-0.049	-0.087	0.025
-0.060	0.092	0.042
0.074	0.050	0.070

(11X3)

=

-0.563	0.336	0.161
-0.915	-0.441	1.557
-1.213	-0.134	-1.322
1.669	-0.154	-1.034
1.721	-1.463	0.726
0.005	1.394	-0.125
-0.056	1.524	-0.786
0.798	1.854	-1.667
-1.368	1.324	-0.481
0.673	1.985	-1.852
-1.524	-1.743	-1.864

(11X3)

Negative
Sampling

Learning R. \times

-0.560	0.340	0.160
-0.910	-0.440	1.560
-1.210	-0.130	-1.320
1.670	-0.150	-1.030
1.720	-1.460	0.730
0.000	1.390	-0.120
-0.060	1.520	-0.790
0.800	1.850	-1.670
-1.370	1.320	-0.480
0.670	1.990	-1.850
-1.520	-1.740	-1.860

(11X3)

Not computed!		
---------------	--	--

(11X3)

=

-0.560	0.340	0.160
-0.910	-0.440	1.560
-1.210	-0.130	-1.320
1.670	-0.150	-1.030
1.720	-1.460	0.730
0.000	1.390	-0.120
-0.060	1.520	-0.790
0.800	1.850	-1.670
-1.370	1.320	-0.480
0.670	1.990	-1.850
-1.520	-1.740	-1.860

(11X3)

Positive sample, w_o

Negative sample, k=1

Negative sample, k=2

Negative sample, k=3

0.031	0.030	0.041
-0.090	0.031	-0.065
0.056	0.098	-0.061
0.069	0.084	-0.044

(11X3)

-0.560	0.340	0.160
-0.910	-0.440	1.560
-1.210	-0.130	-1.320
1.670	-0.150	-1.030
1.720	-1.460	0.730
0.000	1.390	-0.120
-0.060	1.520	-0.790
0.798	1.849	-1.672
-1.366	1.318	-0.477
0.667	1.985	-1.847
-1.523	-1.744	-1.858

(11X3)

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

Visualizar embeddings en baja dimensionalidad:

[LINK](#)



t-SNE (t-distributed stochastic neighbor embedding)



Técnica de reducción de dimensionalidad no-lineal (a diferencia de PCA).



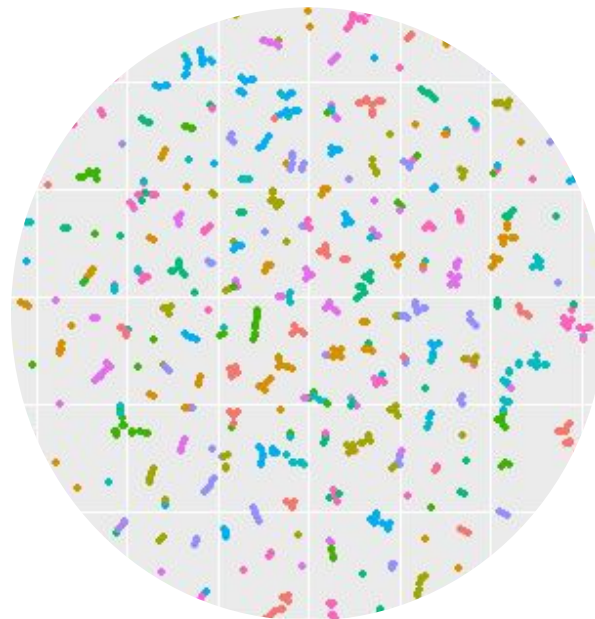
Intenta reproducir en baja dimensionalidad, la localidad de los datos en alta dimensionalidad.



Es estocástica, a priori los resultados no se repiten.



Por su carácter estocástico es sólo recomendable como herramienta de visualización y exploratoria.



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 y nació originalmente para topic modelling
- Muy popular y muy simple de utilizar



Link al Colab



[LINK](#)



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.

Intentar plantear y probar tests de analogías. Graficar los embeddings resultantes.

Sacar conclusiones.



Algunos recursos para descargar corpora de texto



[Project Gutenberg](#)

Compilación de literatura completa de dominio público principalmente en inglés.

[Textos.info](#)

Compilación de literatura completa de dominio público en español.