Redes Neuronales para Lenguaje Natural

2023

Grupo de Procesamiento de Lenguaje Natural
Instituto de Computación


Word Embeddings

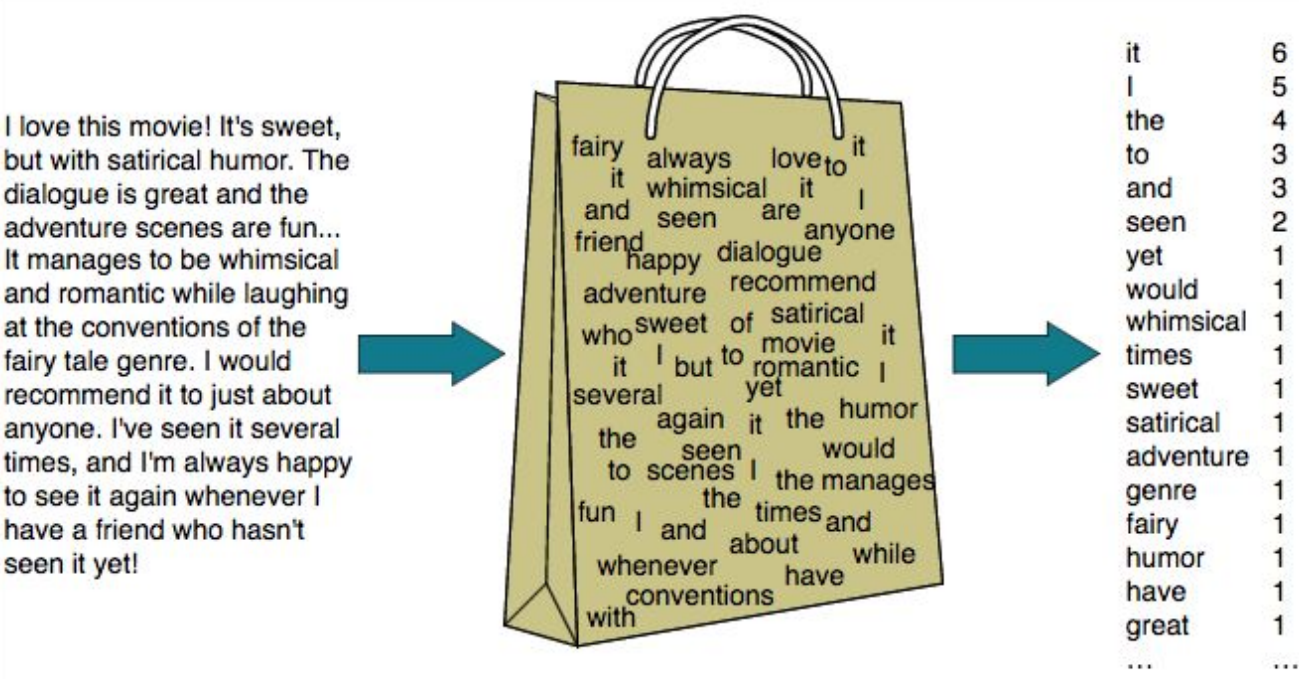
Clasificación de textos

- ¿Cómo vectorizamos la entrada? Aún podemos utilizar lo visto para regresión logística y multinomial
  - – Ingeniería de features
  - – Bag of words
- Este tipo de modelo permite **construir representaciones** de la entrada
  - – **Embedding layer** (look-up matrix)
    - se ajusta durante el entrenamiento como el resto de los parámetros
  - – Se conocen como **word embedding** (vamos a volver sobre esto más adelante)

Motivación

- La entrada de una red neuronal es un vector de activaciones
- Valores numéricos reales
- ¿Cómo representaríamos una imagen?
- ¿Cómo representaríamos una palabra?
- ¿Y una oración?

Representación de textos



- Bag of Words (BOW)
- Hay que considerar un vocabulario fijo
- ¿Cuántas palabras poner?
- Variante: Bag of N-grams

Representación de textos

- Hay palabras que no aportan mucho
  - la, el, en, de, y, que, …

stop words

- ¿Hay forma automática de caracterizar las palabras que aportan más o menos?
  - TF-IDF
  - Calculamos la frecuencia de cada palabra en un documento y en el corpus

Problemas de BOW

Se pierde el orden de las palabras

Son vectores muy grandes y dispersos

Solo sirve para arquitecturas de tamaño fijo

¿Qué hacemos con las palabras desconocidas?

Representación de palabras

Queremos una representación que sea computacionalmente eficiente

- vectores densos de baja dimensionalidad

Idealmente, palabras similares deberían estar más cerca en el espacio, y palabras diferentes deberían estar más lejos

Representación de palabras

Tenemos un idioma compuesto por palabras

¿Cuántas?

¿Las podemos enumerar?

¿Qué palabras consideramos?

Uso de corpus

¿Problemas?

Las palabras son “cercanas” debido a su posición alfabética

1	abrir
2	árbol
3	australopithecus
…	…
3212	correr
…	…
7843	perro
…	…
9999	zanahoria
10000	zoológico

Representación de palabras

Idea más básica: one-hot encoding

		abrir	árbol	australopithecus	…	correr	…	perro	…	zanahoria	zoológico	
árbol	[	0	1	0	…	0	…	0	…	0	0	]
perro	[	0	0	0	…	…	…	1	…	0	0	]
correr	[	0	0	0	…	1	…	0	…	0	0	]

Cada palabra es **ortogonal** a todas las demás

BOW se puede definir como la suma de los one-hot para todas las palabras

Representación de palabras

Hipótesis distribucional: *Palabras que aparecen en contextos similares tienden a tener significados similares*

"You shall know a word by the company it keeps"

Firth, 1957

Inicialmente planteada en los 1950s (Firth, Harris)

La milanesa con queso más rica es la uruguaya.

Sí, es re rica la hamburguesa con queso de ese lugar.

A la milanesa con queso mozzarella y salsa le decimos napolitana.

El otoño es una de las estaciones del año.

¡El verano es una de mis estaciones favoritas!

Hace pila de frío en invierno.

En verano nunca hace frío.

Representación de palabras

Conteos con 4 palabras de contexto

	...	rica	queso	frío	estaciones	...
...						
milanesa		1	2	0	0	
hamburguesa		1	1	0	0	
otoño		0	0	0	0	
verano		0	0	1	0	
invierno		0	0	1	0	
...						

La milanesa con queso más rica es la uruguaya.

Sí, es re rica la hamburguesa con queso de ese lugar.

A la milanesa con queso mozzarella y salsa le decimos napolitana.

El otoño es una de las estaciones del año.

¡El verano es una de mis estaciones favoritas!

Hace pila de frío en invierno.

En verano nunca hace frío.

Representación de palabras

Cada fila de la matriz es la representación de una palabra

Podemos calcular la similitud entre dos palabras con medidas de distancia

- Distancia euclídea
- Producto punto $v \cdot w = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + ... + v_N w_N$
- Similitud coseno $\cos(v, w) = \frac{v \cdot w}{|v||w|}$

Alternativa a contar palabras

PMI: Pointwise Mutual Information

$$PMI(w_1, w_2) = \log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

PPMI: Positive PMI

$$PPMI(w_1, w_2) = \max\left(\log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)}, 0\right)$$

+ suavizado para palabras no frecuentes


word2vec

Vectores dispersos vs vectores densos

Los vectores BOW, TF-IDF o PMI son

- **largos** (largo |V|= 20,000 to 50,000)
- **dispersos** (casi todos los elementos son cero)

Alternativa: aprender vectores que sean

- **cortos** (largo 50-1000)
- **densos** (casi todos los elementos distintos de cero)

Vectores dispersos vs vectores densos

¿Por qué vectores densos?

- Vectores cortos son más fáciles para usar como features en métodos de aprendizaje automático (menos pesos que ajustar)
- Vectores densos pueden **generalizar** mejor que los conteos
- Vectores densos pueden ser mejores para capturar sinonimia:
  - *coche*, *auto* y *automóvil* son sinónimos; pero serían dimensiones diferentes
    - una palabra con vecino *coche* y otra con vecino *automóvil* deberían ser similares, pero no lo serán
- **En la práctica, funcionan mejor**

Métodos habituales para obtener vectores cortos densos

Métodos inspirados en “Modelos de Lenguaje Neuronales”

- word2vec (skipgram, CBOW), GloVe

Descomposición en Valores Singulares (SVD)

- Un caso especial de esto es LSA – Latent Semantic Analysis

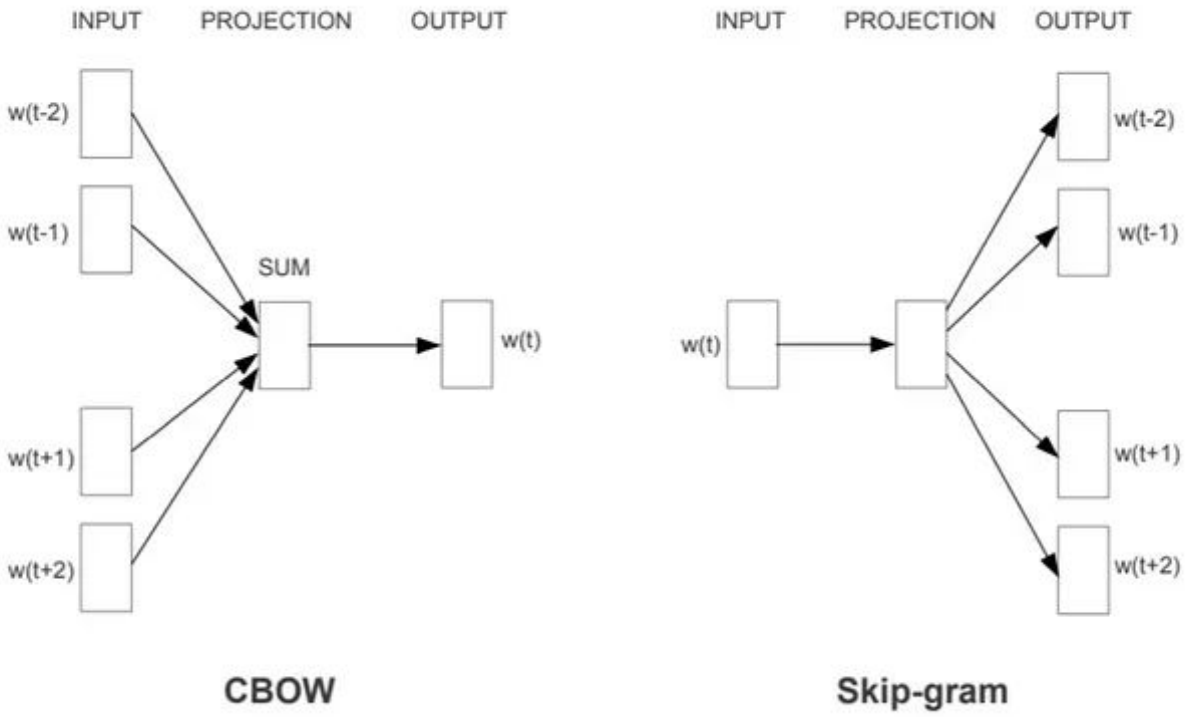
Alternativas a estos “embeddings estáticos”:

- Embeddings contextuales (ELMo, BERT)
- Calculan diferentes embeddings para la palabra en un contexto
- Embeddings separados para cada instancia de la palabra

word2vec

Dos algoritmos de creación de embeddings propuestos por Mikolov, 2013

Auto-supervisados (Representation learning)



word2vec

Método popular para cálculo de embeddings

Muy rápido de entrenar

Código disponible en la web

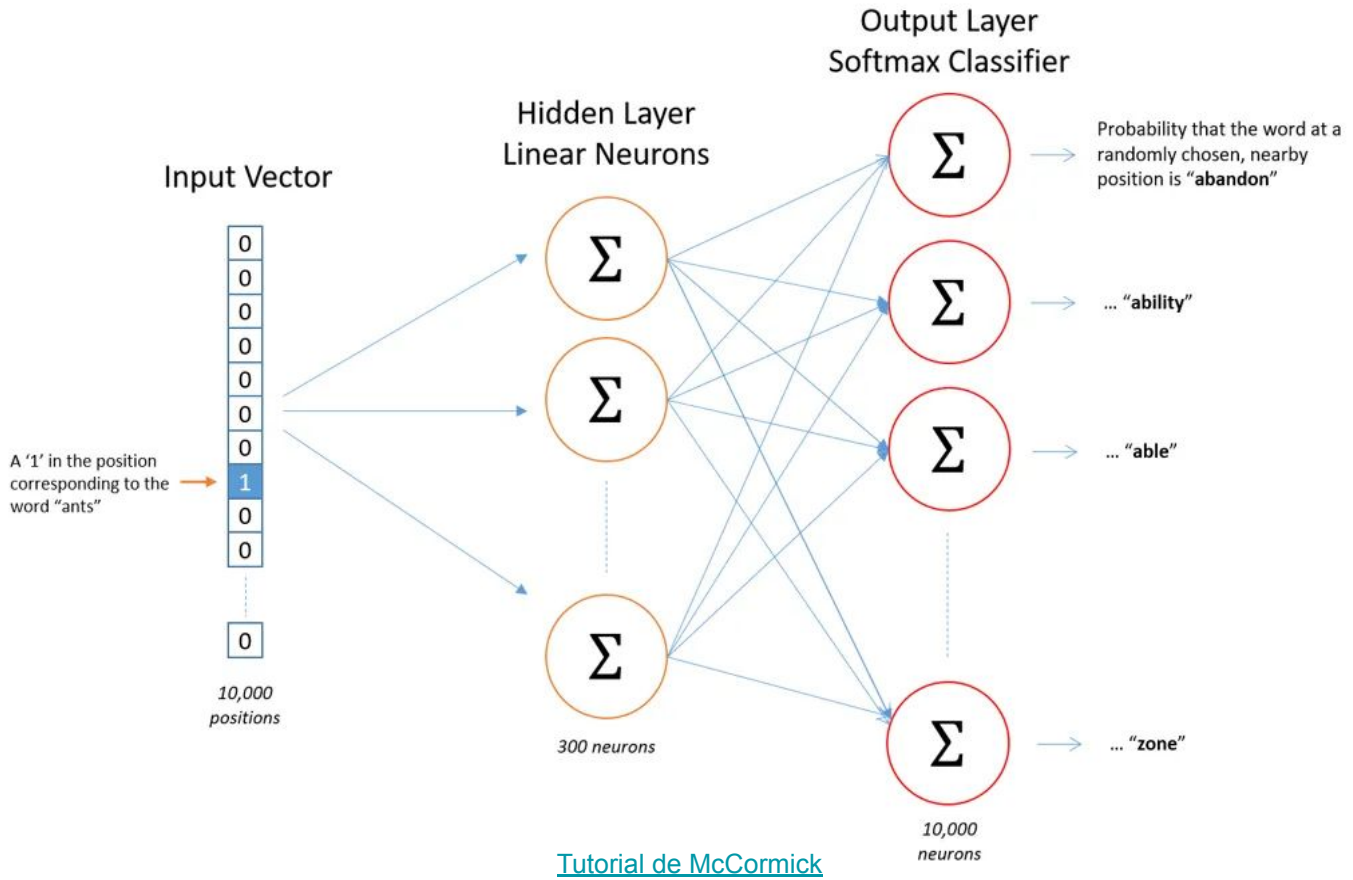
Idea: **predecir** en vez de **contar**

word2vec provee varias opciones. Veremos esta:

skip-gram con negative sampling (SGNS)

word2vec - skip gram

En vez de hacer conteos de palabras, se propone la tarea “ficticia” de predecir qué palabras aparecen alrededor de una palabra objetivo



Tutorial de McCormick

word2vec

En vez de **contar** cuántas veces aparece la palabra *w* cerca de "*apricot*"

- Entrenar un clasificador en una tarea de **predicción** binaria
- Qué tan probable es que *w* aparezca cerca de "*apricot*"?

En realidad no nos importa la tarea en sí

- Pero los pesos que aprenda el clasificador serán nuestros word embeddings

Gran idea: auto-supervisión!

- Una palabra *c* que aparece cerca de *apricot* en el corpus es la "respuesta correcta" según nuestro gold standard supervisado
- No necesitamos etiquetado humano
- Bengio et al. (2003); Collobert et al. (2011)

word2vec

Enfoque: predecir si la palabra candidata *c* es un "vecino"

1. Tratar la palabra objetivo *t* y las palabras vecinas en el contexto *c* como **ejemplos positivos**
2. Sortear aleatoriamente otras palabras del léxico para obtener **ejemplos negativos**
3. Usar regresión logística para entrenar un clasificador que distinga entre ambas clases
4. Usar los pesos aprendidos como embeddings

Jurafsky

Datos de entrenamiento de Skip-Gram

Considerar una ventana de +/- 2 palabras, dada una oración:

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

Objetivo: entrenar un clasificador al que le damos pares candidatos (palabra, contexto)

(apricot, jam)

(apricot, aardvark)

…

Y nos devuelve una probabilidad para cada par:

$P(+|w, c)$

$P(-|w, c) = 1 - P(+|w, c)$

Similitud calculada con producto punto

Recordar: dos vectores son similares si su producto punto es alto

- El coseno es el producto punto normalizado

O sea:

- similitud(w,c) $\propto$ w · c

Normalizaremos luego para obtener una probabilidad

- (el coseno tampoco es una probabilidad)

Transformar productos punto en probabilidades

similitud(w,c) ≈ w · c

Para transformarlo en una probabilidad

Usaremos la función sigmoide de regresión logística:

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1 + exp(-c \cdot w)}$$

$$P(-|w,c) = 1 - P(+|w,c)$$

$$= \sigma(-w \cdot c) = \frac{1}{1 + exp(c \cdot w)}$$

Cómo calcula Skip-Gram P(+|w, c)

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1 + exp(-c \cdot w)}$$

Esto es para una palabra de contexto. Pero tenemos muchas palabras de contexto.

Asumiremos independencia entre todas, así las multiplicamos:

$$P(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

Clasificador Skip-gram: resumen

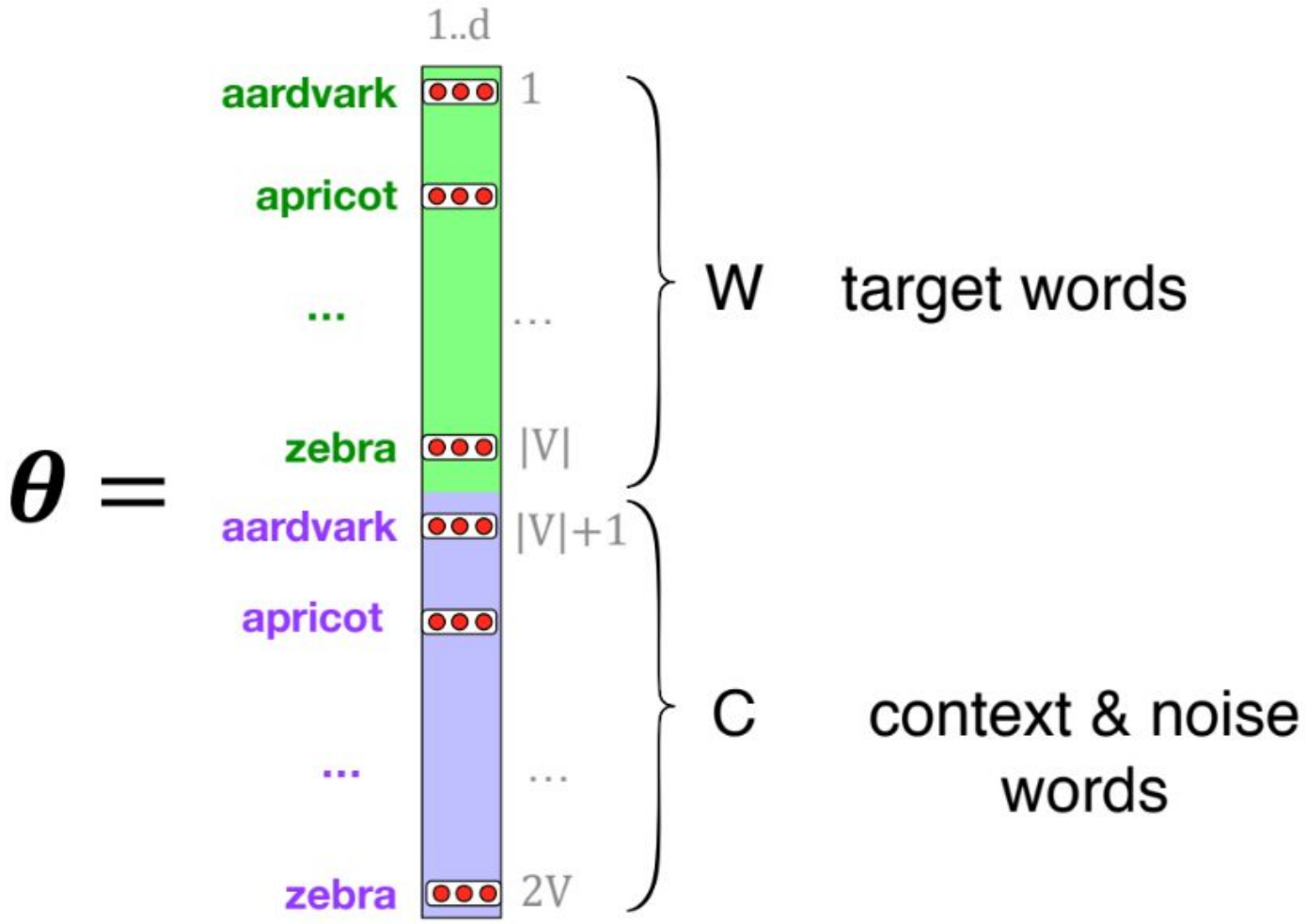
Un clasificador probabilístico, que dados

- una palabra objetivo *w*
- su ventana de contexto de *L* palabras $c_{1:L}$

Estima la probabilidad de que *w* ocurra en la ventana basándose en la similitud entre *w* (embedding) y $c_{1:L}$ (embeddings).

Para calcularlo, necesitamos tener los embeddings de cada palabra.

¿Qué son los embeddings que necesitamos? un conjunto de pesos para w, otro conjunto para c




Entrenamiento

Datos de entrenamiento de Skip-Gram

Considerar una ventana de +/- 2 palabras, dada una oración:

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

Para cada ejemplo positivo, obtendremos k ejemplos negativos, muestreando por frecuencia

ejemplos positivos +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

ejemplos negativos +

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

word2vec: cómo se aprenden los vectores

Dado el conjunto de ejemplos de entrenamiento positivos y negativos, y un conjunto inicial de vectores de embedding

El objetivo del aprendizaje es ajustar esos vectores para que:

- **Maximicen** la similitud de los pares *(w, c_{pos})* de palabra objetivo, palabra contexto seleccionados de los datos positivos
- **Minimicen** la similitud de los pares *(w, c_{neg})* seleccionados de los datos negativos

Función de pérdida para *w* y c_{pos} , c_{neg1} … c_{negk}

Maximizar la similitud entre la palabra objetivo y las palabras reales de contexto, y a la vez minimizar la similitud entre la palabra objetivo y las *k* palabras negativas sorteadas.

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w, c_{pos}) \prod_{i=1}^{k} P(-|w, c_{neg_i})\right] \\
&= -\left[\log P(+|w, c_{pos}) + \sum_{i=1}^{k} \log P(-|w, c_{neg_i})\right] \\
&= -\left[\log P(+|w, c_{pos}) + \sum_{i=1}^{k} \log 1 - P(+|w, c_{neg_i})\right] \\
&= -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]
\end{aligned}
$$

Aprender el clasificador

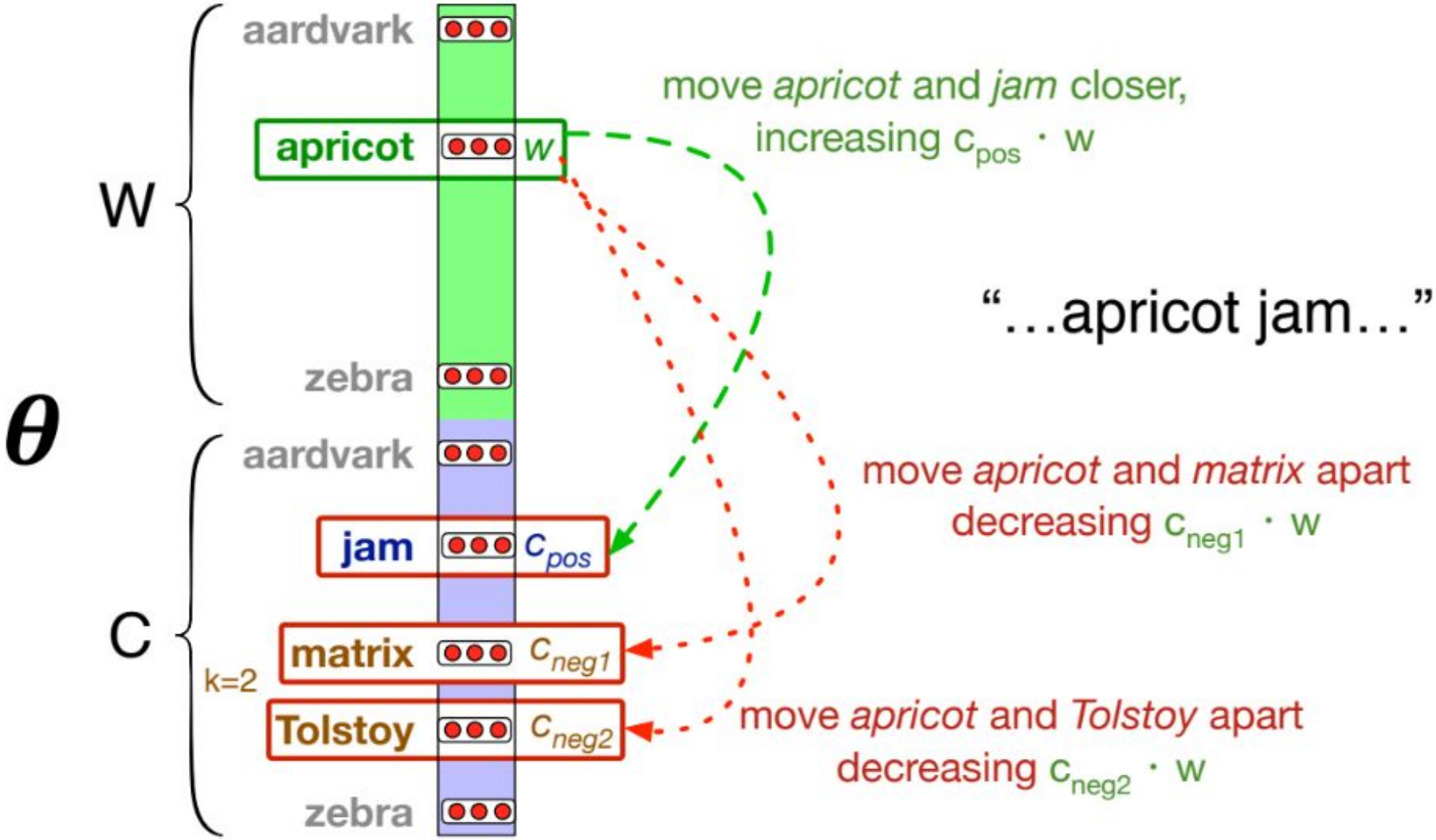
Cómo se entrena?

- SGD: Descenso estocástico por gradiente

Ajustaremos los pesos para hacer que

- los pares positivos sean más probables
- los pares negativos sean menos probables
- iterando sobre todo el conjunto de entrenamiento

Intuición de un paso del descenso por gradiente



Recordemos: descenso por gradiente

- En cada paso
  - Dirección: Nos movemos en la dirección inversa al gradiente de la función de pérdida
  - Magnitud: Nos movemos por el gradiente $\frac{d}{dw}L(f(x;w),y)$ un valor ponderado por el *learning rate η*
  - Learning rate más alto implica mover los pesos *w* más rápido

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w),y)$$

Derivadas de la función de pérdida

$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

$$\frac{\partial L_{CE}(\hat{y}, y))}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

Jurafsky 5.6.1

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w)]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

Actualización de la ecuación en SGD

- Comenzamos con los pesos en matrices C y W inicializados aleatoriamente

$$c_{pos}^{t+1} = c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t}$$
$$c_{neg}^{t+1} = c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t}$$
$$w^{t+1} = w^{t} - \eta[\sigma(c_{pos} \cdot w^{t})]c_{pos} + \sum_{i=1}^{k}[\sigma(c_{neg_i} \cdot w^{t})]c_{neg_i}$$

Dos conjuntos de embeddings

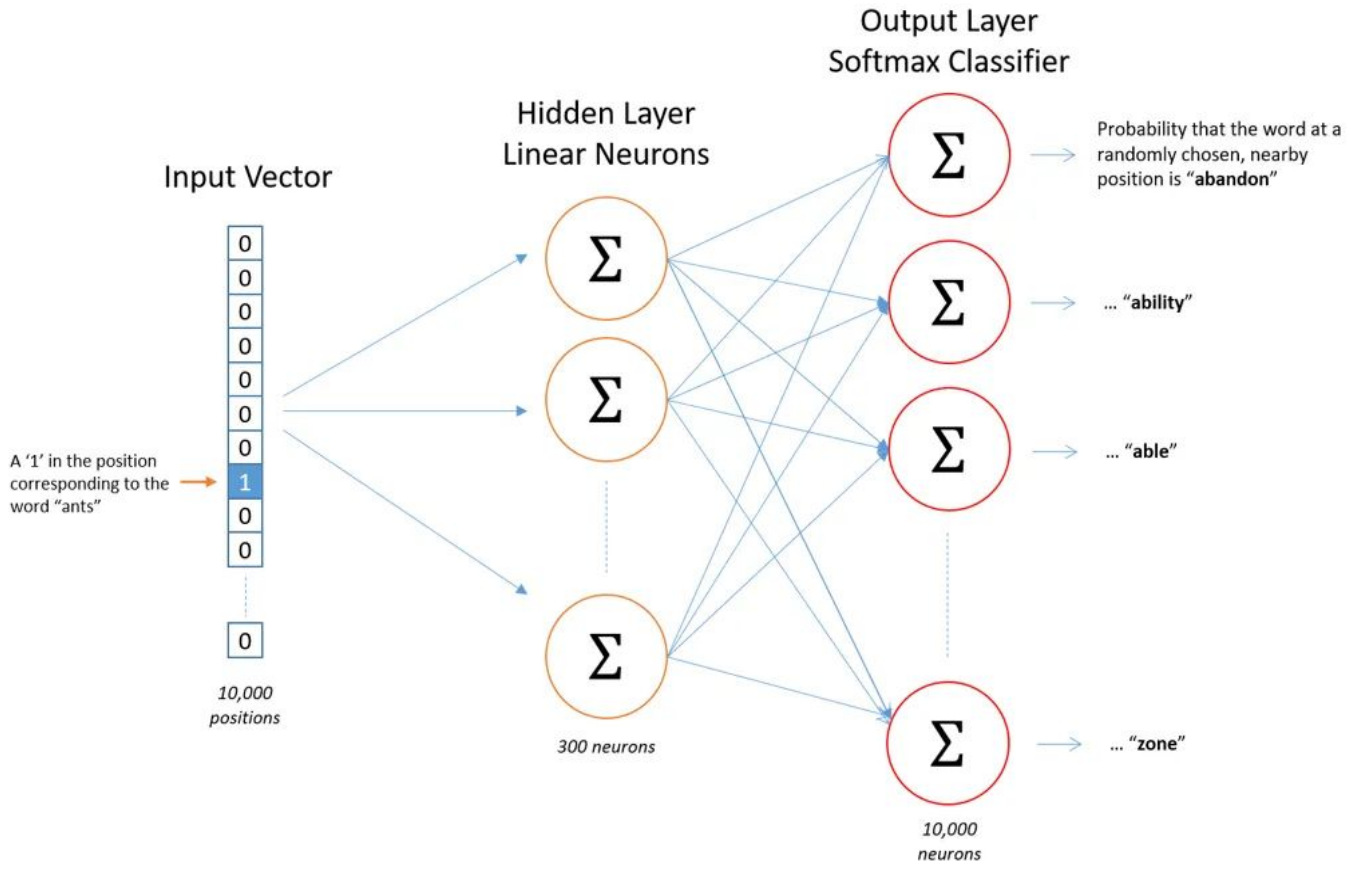
SGNS aprende dos conjuntos de embeddings

Embeddings de la palabra objetivo en la matriz W

Embeddings de palabras contexto en la matriz C

- Es común sumar ambos embeddings, obteniendo como representación de la palabra *i* el vector $w_i + c_i$
- También es común quedarse solo con W: la palabra i se representa con w_i

word2vec - skip gram como red neuronal



Resumen: Cómo aprender embeddings tipo word2vec (skip-gram)

Comenzar con V vectores d-dimensionales aleatorios como embeddings iniciales

Entrenar un clasificador basado en similitud de embeddings

- Del corpus, obtenemos como ejemplos positivos pares de palabras que co-ocurren
- Sorteamos como ejemplos negativos pares de palabras que no co-ocurren
- Entrenamos el clasificador para que distinga entre estas clases mientras se van ajustando los pesos de los embeddings para mejorar la performance del clasificador
- Olvidarse del clasificador: los embeddings son los pesos obtenidos


Word Embeddings