Maestría en Inteligencia Artificial Aplicada (MNA)

CBOW/skip-gram y modelos Secuenciales

Procesamiento de Lenguaje Natural (NLP)

Luis Eduardo Falcón Morales

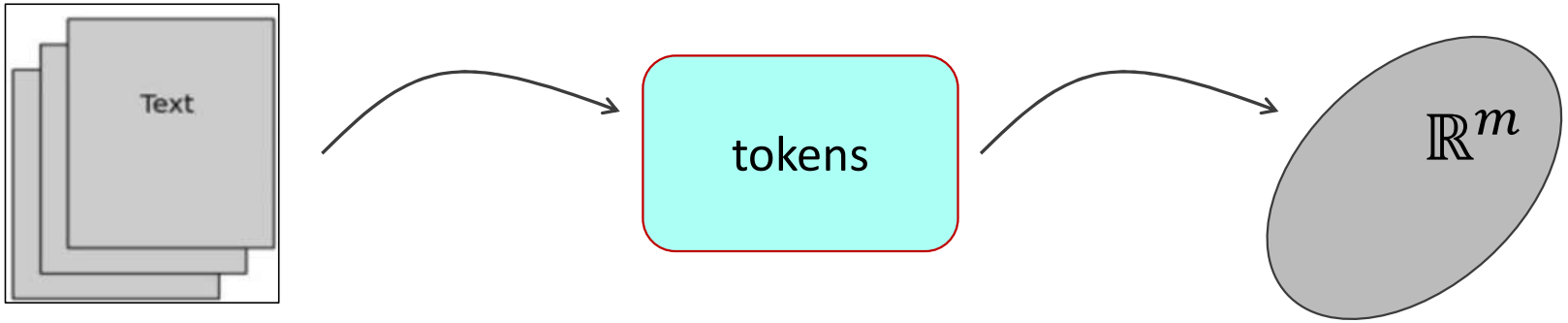
Vectorización, CBOW/Skip-gram y Modelos Secuenciales

Mucha de la complejidad que involucran los problemas de análisis de textos radica en la dependencia o relación que guarda una palabra con las anteriores y posteriores en un documento.

El problema es similar a los problemas relacionados con variables temporales, como lo puede ser un índice financiero a lo largo del tiempo.

En esta semana estudiaremos varios de los modelos de vectorización de palabras que en los últimos años han estado generando gran impacto con soluciones a problemáticas de la vida real.

Proceso de Vectorización / Word Embedding



El proceso de **vectorización**, transformación de texto plano a un vector de valores numéricos, puede realizarse de varias formas: **segmentación por caracteres, palabras, n-gramas o enunciados**.

Este proceso de vectorización es también usualmente llamado Incrustación de Palabras (Word Embedding en inglés).

Métodos para generar Word Embeddings

Existen diferentes métodos para generar representaciones vectoriales de palabras (Word Embeddings), también llamados vectores continuos:

aprendizaje

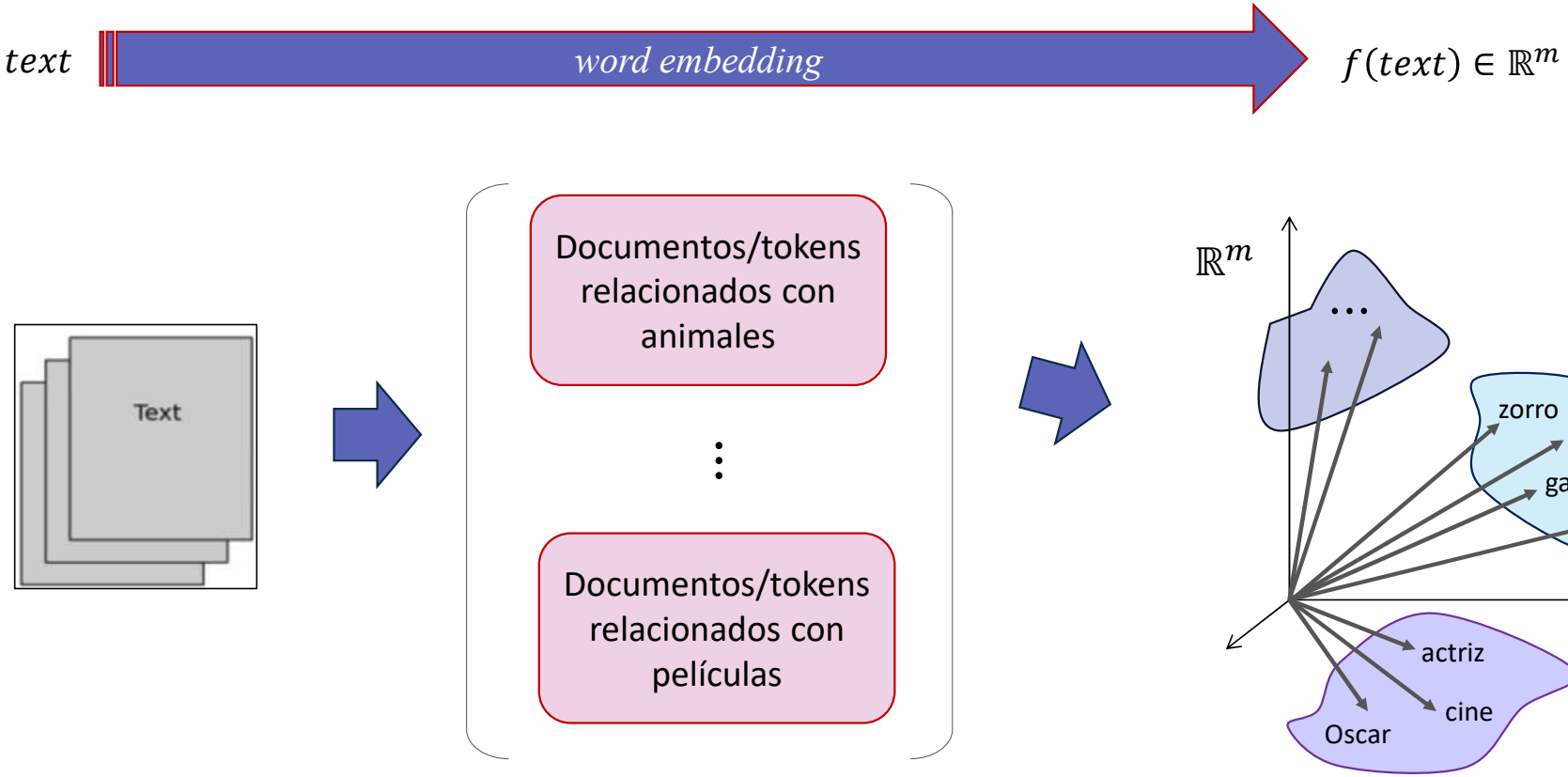
- Métodos basados en conteo
  - **Tf-idf, Latent Sentiment Analysis (LSA)**

pre-entrenados

- Métodos basados en aprendizaje de bolsa de palabras (BOW)
  - **Word2Vec** (Google - 2013): CBOW & Skip Gram
  - **Glove** (Stanford University - 2014)
  - **Fast-Text** (Facebook - 2016)

- Métodos basados en la contextualización de las palabras/enunciados:
  - **ELMo** (2018), **BERT** (2019), **GPT** (2022)

La estructura semántica de un documento nos habla de la estructura de los enunciados, a nivel local y global, y las relaciones de cohesión y coherencia entre ellos.

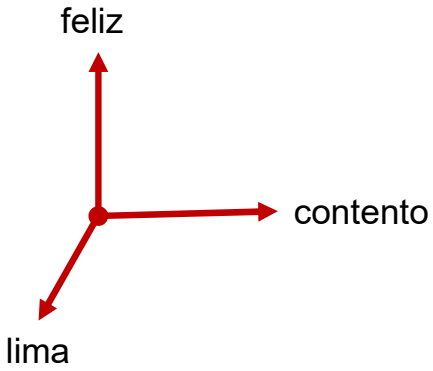


- Es muy variable, pero en promedio en un idioma dado:
  - Cantidad de palabras diferentes: 1 millón aprox.
  - Cantidad de palabras de mayor frecuencia de uso: 200,000 aprox.
  - Cantidad de palabras dada una temática particular: 25,000 aprox.

Tokenización *one-hot encoding*:

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
feliz	contento	lima	limón

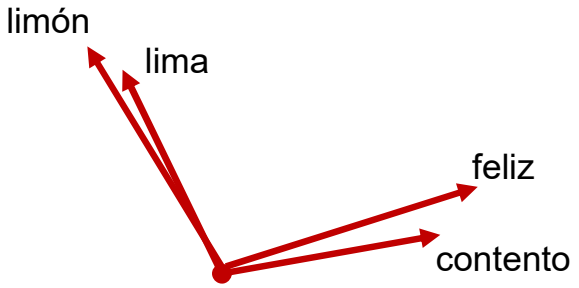
En este diccionario de 4 palabras, palabras similares tienen asociados vectores discretizados linealmente independientes, lo cual no permite relacionarlos por su significado “similar”.



Tokenización *word embedding*:

0.91	0.89	0.02	0
0.78	0.82	0.07	0.1
0.05	0.03	0.94	0.9
0.01	0.1	0.66	0.7
feliz	contento	lima	limón

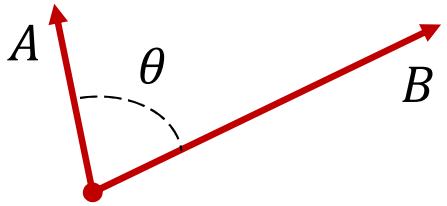
En este diccionario de 4 palabras, palabras similares tienen asociados vectores reales continuos “similares”.



Métricas de similaridad

Existen varias métricas para aplicar el concepto de “similaridad” entre palabras.

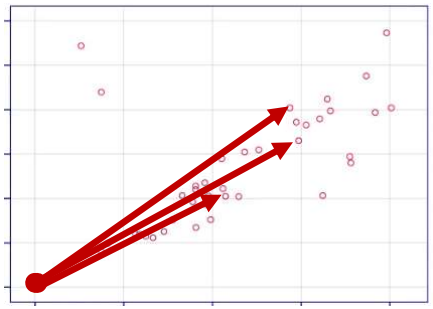
En particular se puede utilizar el **coseno** del ángulo entre vectores para representar la **similaridad** entre dos tokens:



$$cos(\theta) = \frac{\langle A, B \rangle}{\|A\|\|B\|}$$

$$-1 \leq cos(\theta) \leq 1$$

De amanera análoga, puede utilizarse el **coeficiente de correlación de Pearson**:

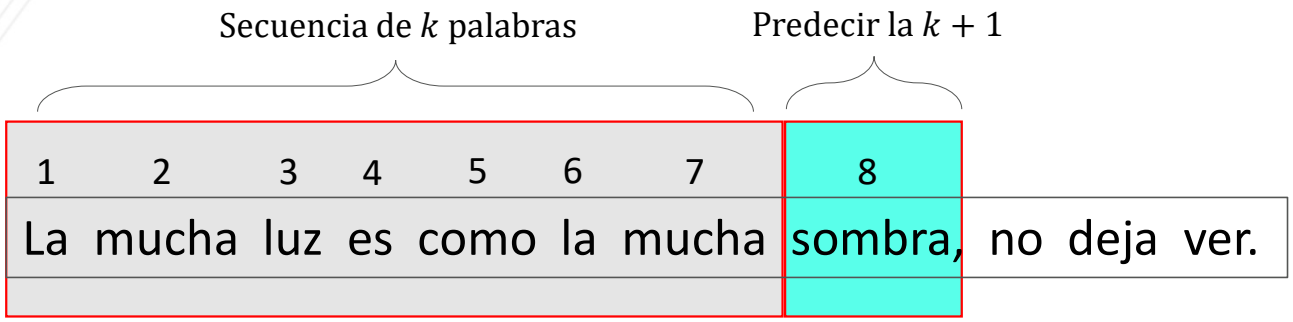


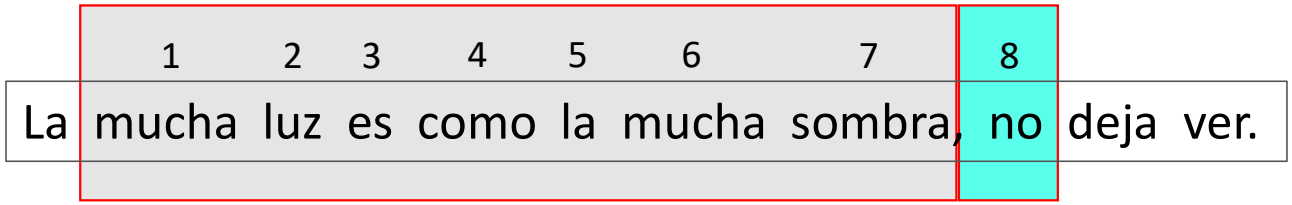
$$\rho = \frac{cov[X, Y]}{\sqrt{Var[X]\, Var[Y]}}$$

$$-1 \leq \rho \leq +1$$

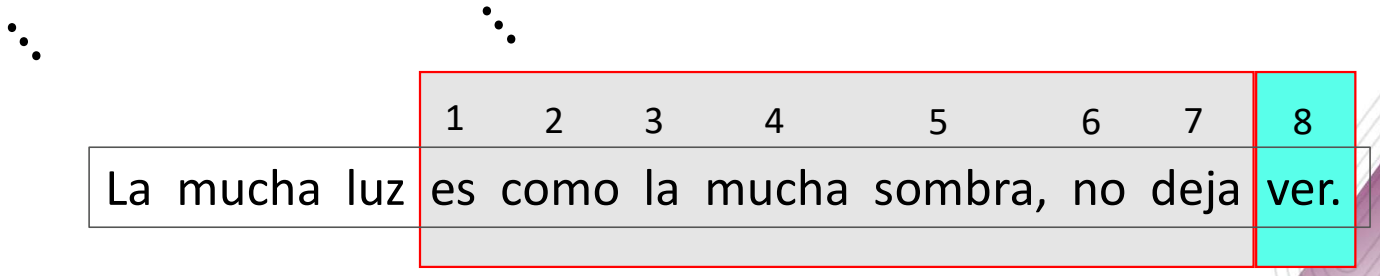
Bag-of-Words: En el modelo BOW no importa el orden de las palabras, solo las palabras que aparecen a cierta distancia de la palabra central.

Se entrena una red neuronal con secuencia de palabras, digamos de longitud k, y el objetivo es predecir la palabra $k + 1$:





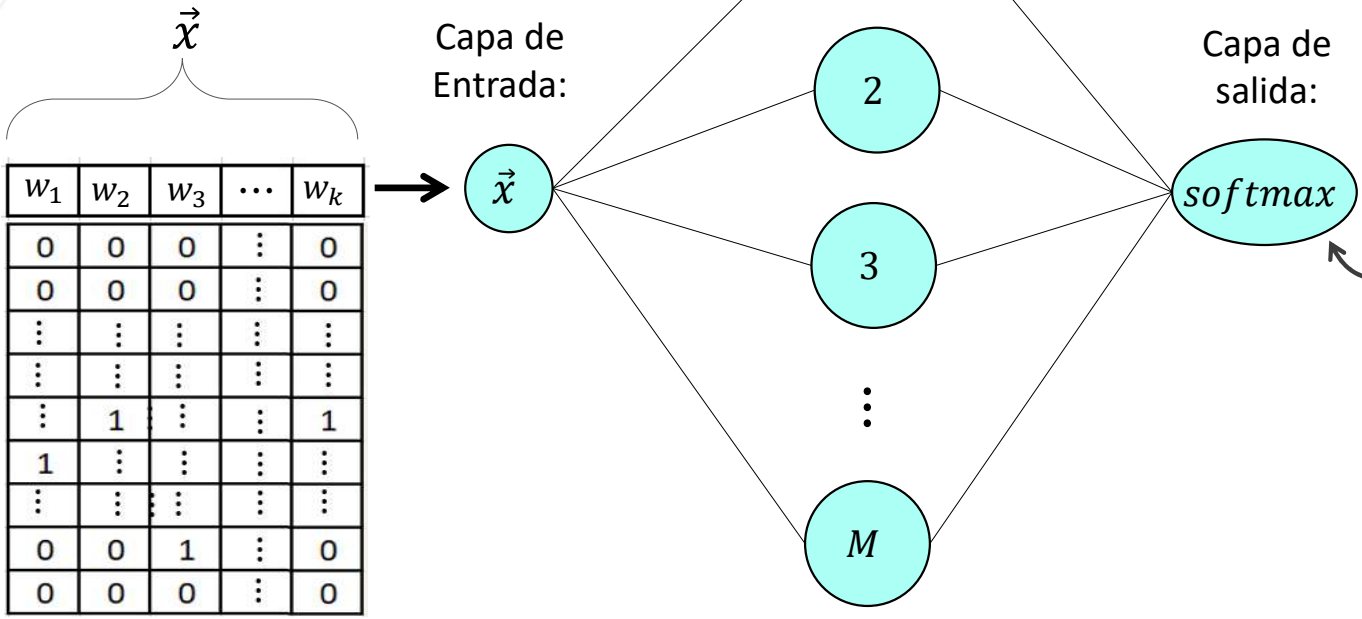
Por ejemplo, en la siguiente figura se ilustra el conjunto de BOW generados en el caso de una secuencia de palabras de longitud 5, o bien, a una distancia 2 de la palabra central.



Octavio Paz

Aprendizaje: Red Neuronal con Capa Embebida para obtener los *Word Embedding*

La matriz $\vec{x}$ está formada por los vectores OHE que "acompañan" a la palabra "central" w_t en un dato muestral de entrenamiento.



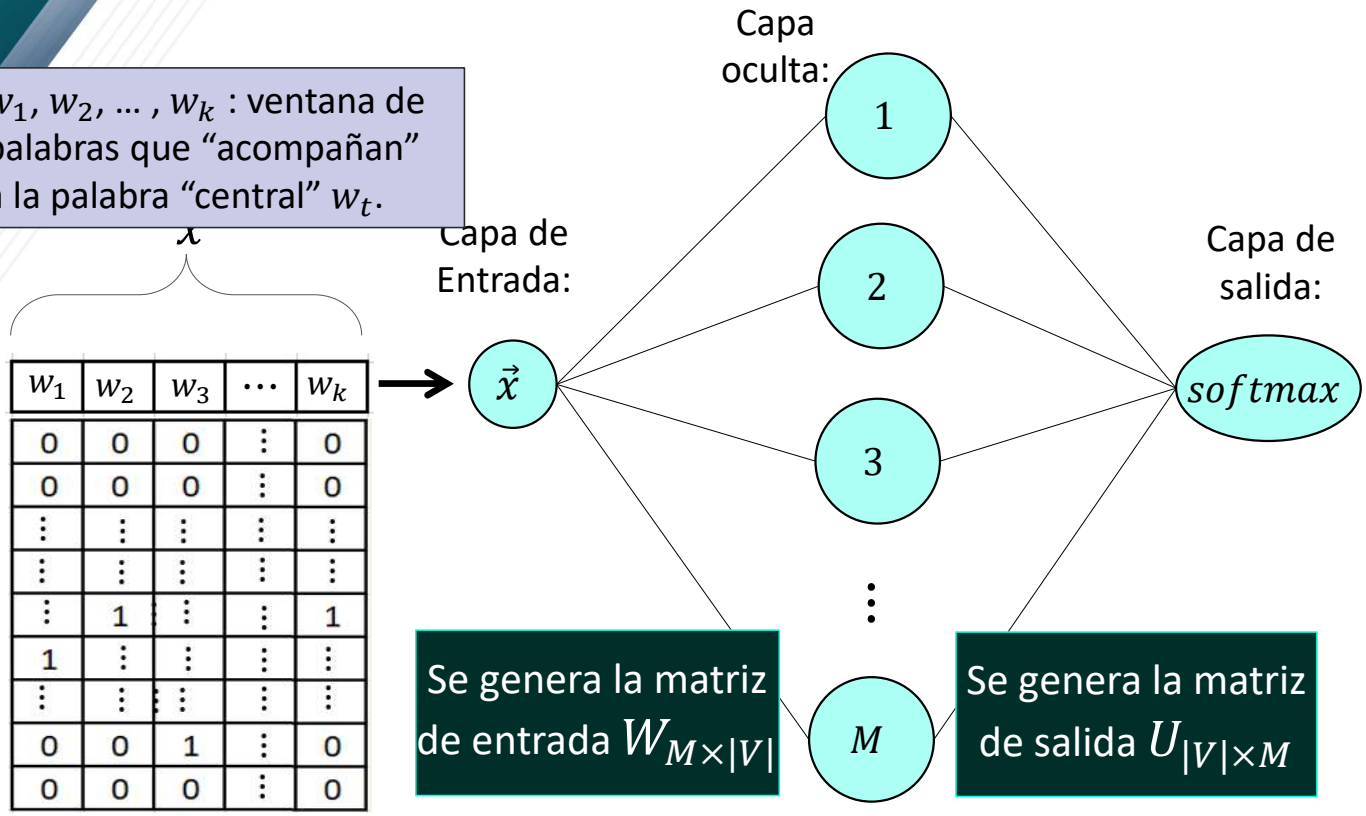
Probabilidad de que la palabra de salida $\widehat{w}_t$ sea w_t.

$|V|$: número de tokens del diccionario.

M : Dimensión del espacio *word embedding*.

Entrada:
Secuencia de palabras (tokens) con One-Hot-Encodding (OHE) con una longitud de ventana fija, de un vocabulario de $|V|$ palabras, en la vecindad de la palabra w_t.

$w_1, w_2, \dots, w_k$: ventana de palabras que "acompañan" a la palabra "central" w_t.



Probabilidad máxima de que la palabra de salida $\widehat{w}_t$ sea igual a w_t.

$|V|$: número de tokens del diccionario.

M : Dimensión del espacio *word embedding*.

Del entrenamiento de esta red neuronal obtenemos dos matrices de transformación $W_{M\times|V|}$ y $U_{|V|\times M}$, donde a cada una de las $|V|$ palabras del vocabulario se le asociada un vector continuo de dimensión M.

Tokenization

```
docs = [ 'Muy bien hecho lo hecho',
         'Excelente trabajo muchacho',
         'Sigue como hasta ahora',
         'Lo hiciste muy bien',
         'Que siga el buen ánimo',
         'Pésimo trabajo',
         'Que mal desempeño',
         'Muy mal hecho',
         'Que trabajo tan pobre',
         'No está bien']
```

```
labels = np.array([1,1,1,1,1, 0,0,0,0,0])
```

Diccionario

```
tokens.index_word
```

```
{1: '#',
 2: 'muy',
 3: 'bien',
 4: 'hecho',
 5: 'trabajo',
 6: 'que',
 7: 'lo',
 8: 'mal',
 9: 'excelente',
 10: 'muchacho',
 11: 'sigue',
 12: 'como',
 13: 'hasta',
 14: 'ahora'
```

Sequences, length=6 docs

```
print(padded_docs)
```

```
[[ 2  3  4  7  4  0]
 [ 9  5 10  0  0  0]
 [11 12 13 14  0  0]
 [ 7 15  2  3  0  0]
 [ 6 16 17 18 19  0]
 [20  5  0  0  0  0]
 [ 6  8 21  0  0  0]
 [ 2  8  4  0  0  0]
 [ 6  5 22 23  0  0]
 [24 25  3  0  0  0]]
```

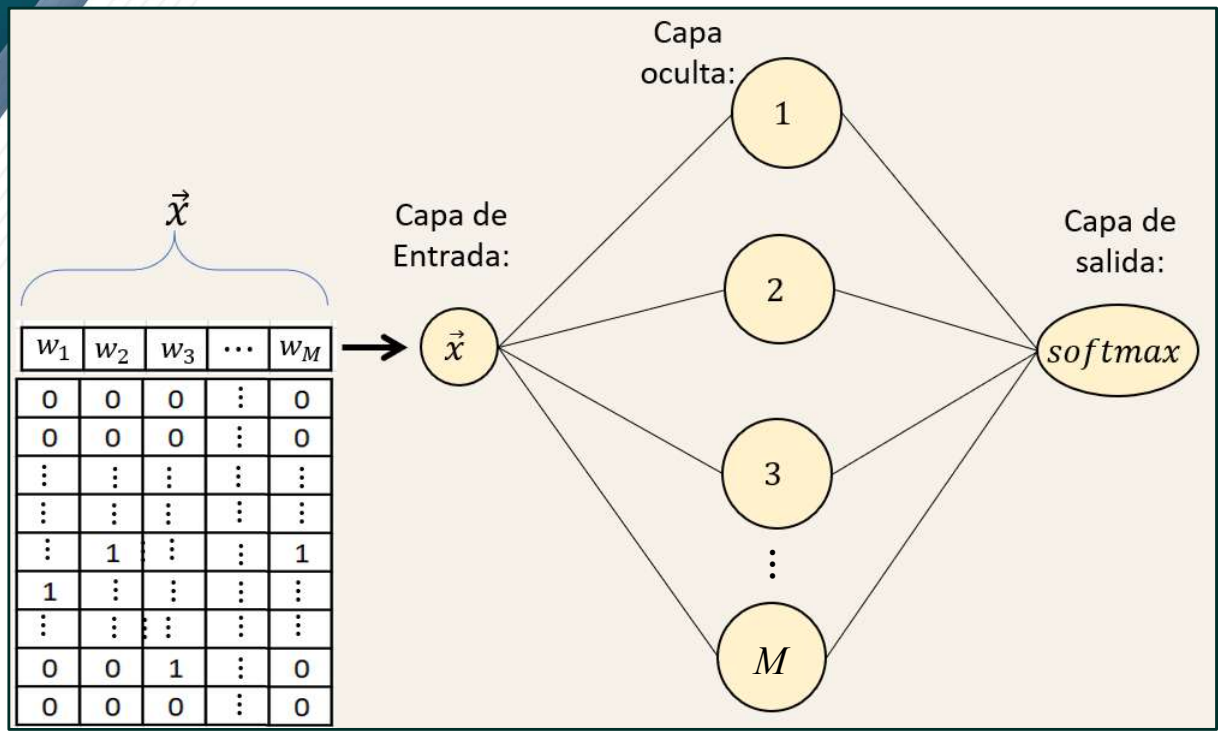
Dense Vectors, dim=4

```
matrixEmbedding.weights
```

```
ariable 'embedding_1/embeddings:0' shape=(26, 4) dtype=fl
[[-0.04464644, -0.05516564,  0.00996508, -0.02029327],
 [-0.00343312,  0.0278131 , -0.00697067, -0.04113562],
 [-0.0008171 ,  0.08894813,  0.07018009, -0.00795872],
 [ 0.080949  ,  0.01199691, -0.03984289, -0.0675302 ],
 [ 0.04170085,  0.06890322,  0.07722773,  0.01145003],
 [-0.00337593,  0.02143945, -0.006995  , -0.07491899],
 [ 0.01655192, -0.07871365,  0.08654396, -0.07193401],
 [ 0.01155774,  0.06323144,  0.00806769,  0.06426584],
 [-0.06300905,  0.02952315,  0.01216486, -0.00665407],
 [-0.00523054,  0.07882133, -0.00823868,  0.05557589],
```

Cada palabra (token) del vocabulario tiene asociado un vector denso de dimensión 4.

Word Embedding: Word → Dense Vector



Regla de Bayes:

$$P(w_t \mid w_1, w_2, \ldots, w_M) = \frac{P(w_t)P(w_1 \cap w_2 \cap \cdots \cap w_M \mid w_t)}{P(w_1 \cap w_2 \cap \cdots \cap w_M)}$$

- La función $softmax$ de la capa de salida nos da la probabilidad de que una palabra w_t sea la de mayor probabilidad, dado el contexto de palabras muestrales de entrada w_1, $w_2, \ldots, w_M$, donde w_t es el vector central.
- Si nuestro diccionario es de 10,000 palabras, se deben encontrar dichas probabilidades para todas ellas.
- La longitud de los vectores embebidos será igual al número de neuronas M.
- Pero tengamos en cuenta que cada una de las probabilidades que se deben encontrar para todas las palabras del diccionario, son probabilidades condicionales, es decir, dado el contexto de palabras de entrada w_1, $w_2, \ldots, w_M$.

Hierarchical Softmax

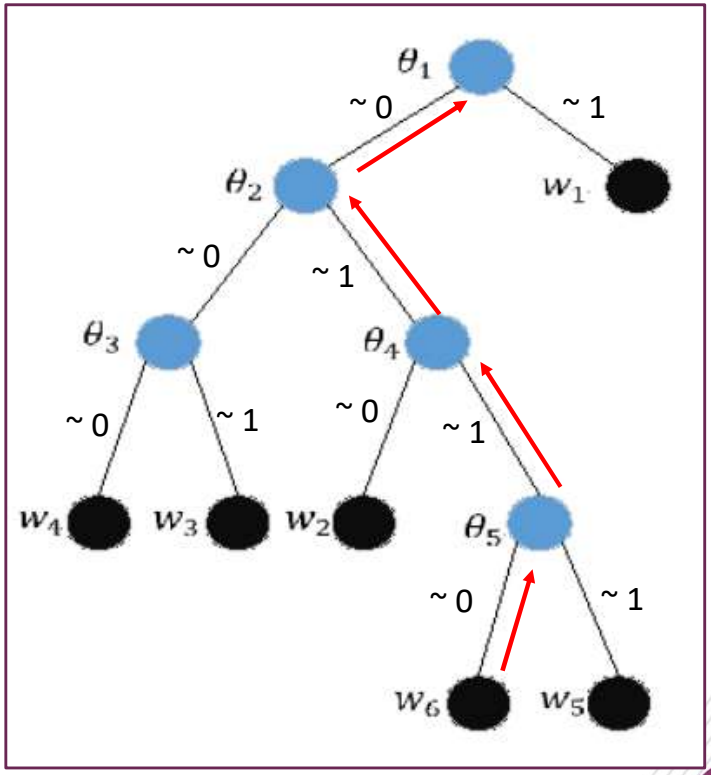
- Debido al costo computacional de la softmax en la capa de salida, se buscan métodos alternativos que lo aproximen, el método "hierarchical softmax" es uno de ellos.

- Construir primeramente el árbol binario de Hauffman para las $|V|$ palabras del diccionario, el cual lo hace considerando la frecuencia de uso de cada palabra. Cada palabra del vocabulario será una de las hojas del árbol. Este método construye un árbol binario de forma tal que las palabras más raras se encuentra en capas más profundas del árbol y las más frecuentes se encuentran en las capas más cercanas al nodo raíz.

- Dicho árbol binario tendrá $|V|$ hojas (que son cada una de las palabras del vocabulario) y $|V| - 1$ nodos internos (nodos azules en la Figura).

- Una vez entrenado el árbol, se aproximan las probabilidades condicionales con las ramas involucradas: producto punto y sigmoide.

- Complejidad $log_2(|V|)$.



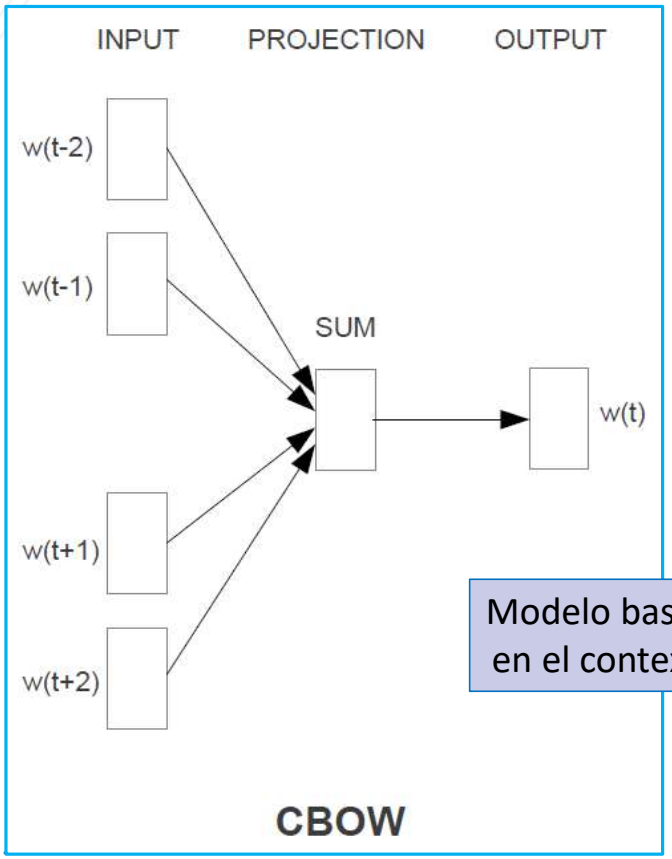
Palabras de menor frecuencia (o raras) aparecen en niveles más bajos del árbol.

Negative Sampling

- Este es otro método alternativo al uso de “softmax”.
- El “negative sampling” selecciona las palabras que están involucradas en la ventana de la palaba central (positive sampling) y aleatoriamente selecciona una cantidad parecida de palabras que no están relacionadas con la palabra central (negative sampling) y con todas ellas realiza el entrenamiento.
- La cantidad de palabras utilizada es obviamente mucho menor que las del vocabulario mismo.
- En la práctica, el método “hierarchical softmax” es mejor cuando se tienen muchas palabras poco frecuentes en el vocabulario, mientras que el “negative sampling” funciona mejor con palabras de alta frecuencia y dimensiones no muy altas de los vectores embebidos.

Word2Vec (Google) : CBOW : continuous bag-of-words

La arquitectura CBOW se llama así, ya que como en cualquier Bag-of-Words, aquí tampoco el orden de las palabras se está considerando: el orden se pierde al tomar el promedio de los vectores de densidad.



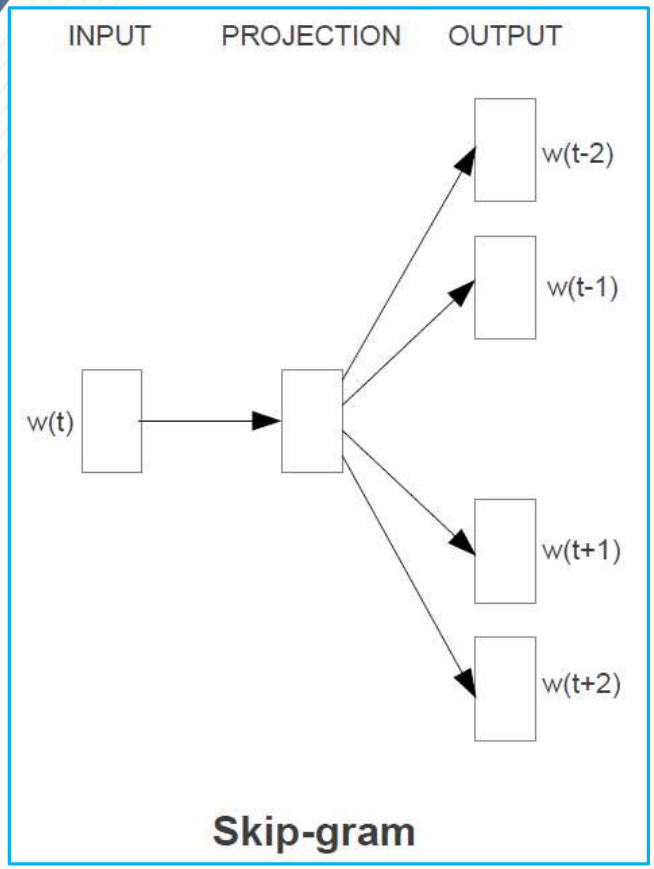
Modelo basado en el contexto.

Los autores de hecho entrenaron el modelo con diferentes longitudes de los vectores de densidad: 20, 50, 80, 100, 300, 600, 640 y 1000.

Preprocesamiento:

- Se aplica un pre-procesado de los documentos, entre ellos la eliminación de *stop-words*.
- En el artículo original sí se consideraron signos de puntuación.
- Se forma el vocabulario deseado.
- Indexar cada palabra/token del vocabulario.
- Seleccionar la dimensión M deseada de los Word embeddings.

Skip-gram : *Continuous Skip-gram model*



Modelo con la arquitectura Skip-gram:

- Modelo análogo al CBOW, pero en este caso se tiene una sola palabra de entrada w_t.
- Como un problema de clasificación, se busca ahora la probabilidad que maximice la aparición de un conjunto de palabras a una distancia preestablecida, antes o después de w_t.
- Nuevamente observa que el orden de aparición de las palabras formalmente no se toma en cuenta durante el proceso de entrenamiento y el cálculo de las probabilidades de la softmax. Esta softmax está sobre todo el conjunto del vocabulario

Complejidad de CBOW:

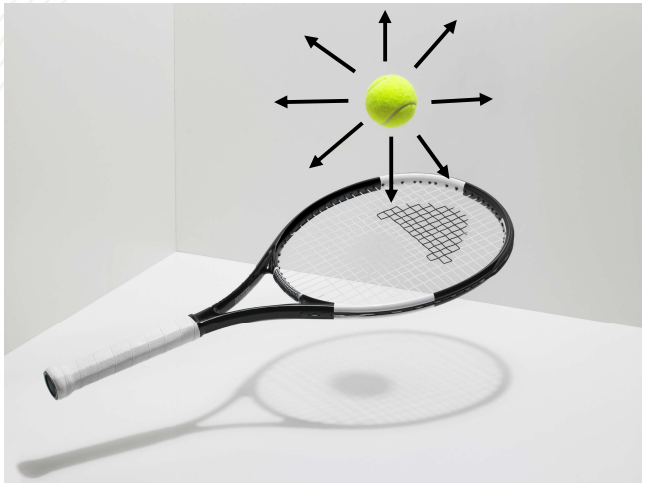
$$Q = D * [N + log_2(|V|)]$$

Complejidad de Skip-gram:

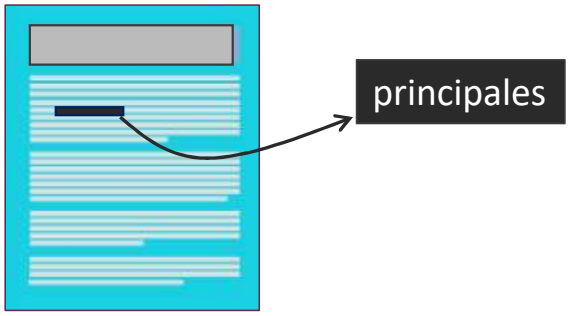
$$Q = C * D * [1 + log_2(|V|)]$$

N : longitud de la secuencia de palabras de entrada.
D : dimensión de los vectores continuos (*word embedings*).
$|V|$: Cantidad de palabras/tokens del vocabulario.
C : Distancia máxima de las palabras de contexto.

Modelos Secuenciales / Recurrentes



Si tenemos la imagen congelada de una pelota que estaba en movimiento, ¿cómo poder predecir la dirección de su movimiento con solo dicha información?



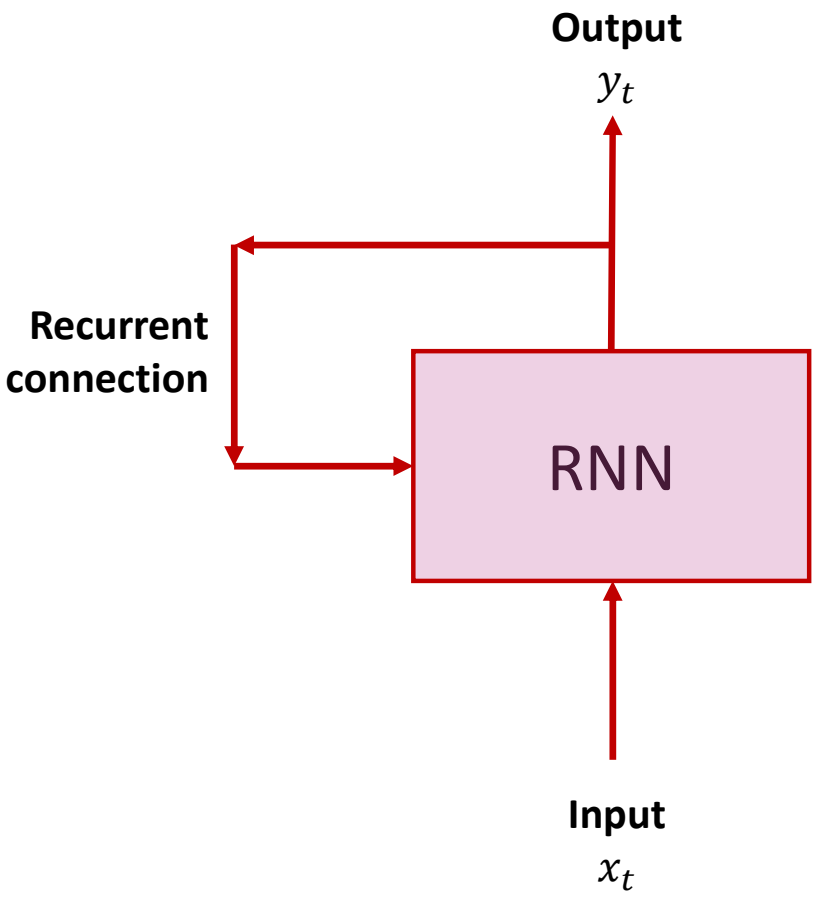
Si conocemos solo una palabra de un texto, ¿cómo saber el significado de dicho texto?

En los modelos Word Embeding no se considera el orden de las palabras, ¿cómo diferenciar entonces enunciados como los siguientes?

La **atención** es mala, pero la **comida** es buena.

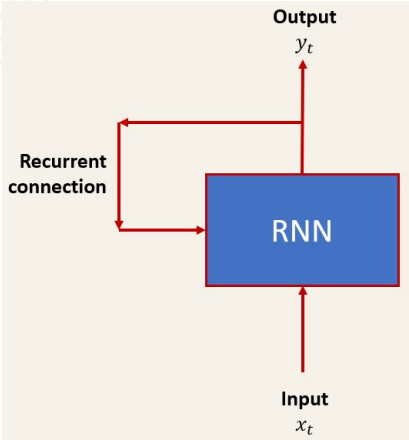
La **comida** es mala, pero la **atención** es buena.

Red Neuronal Recurrente
Recurrent Neural Network (RNN)

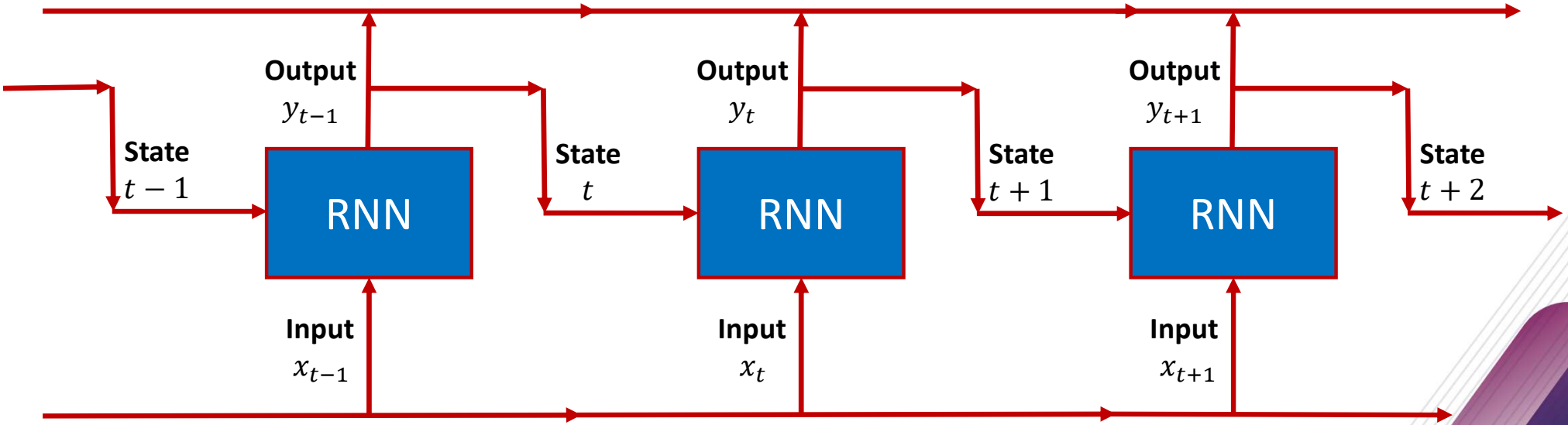


- Las RNN pueden procesar la información considerando el pasado.
- Podemos decir entonces que las RNN tienen memoria.
- Propagan información a lo largo del tiempo.

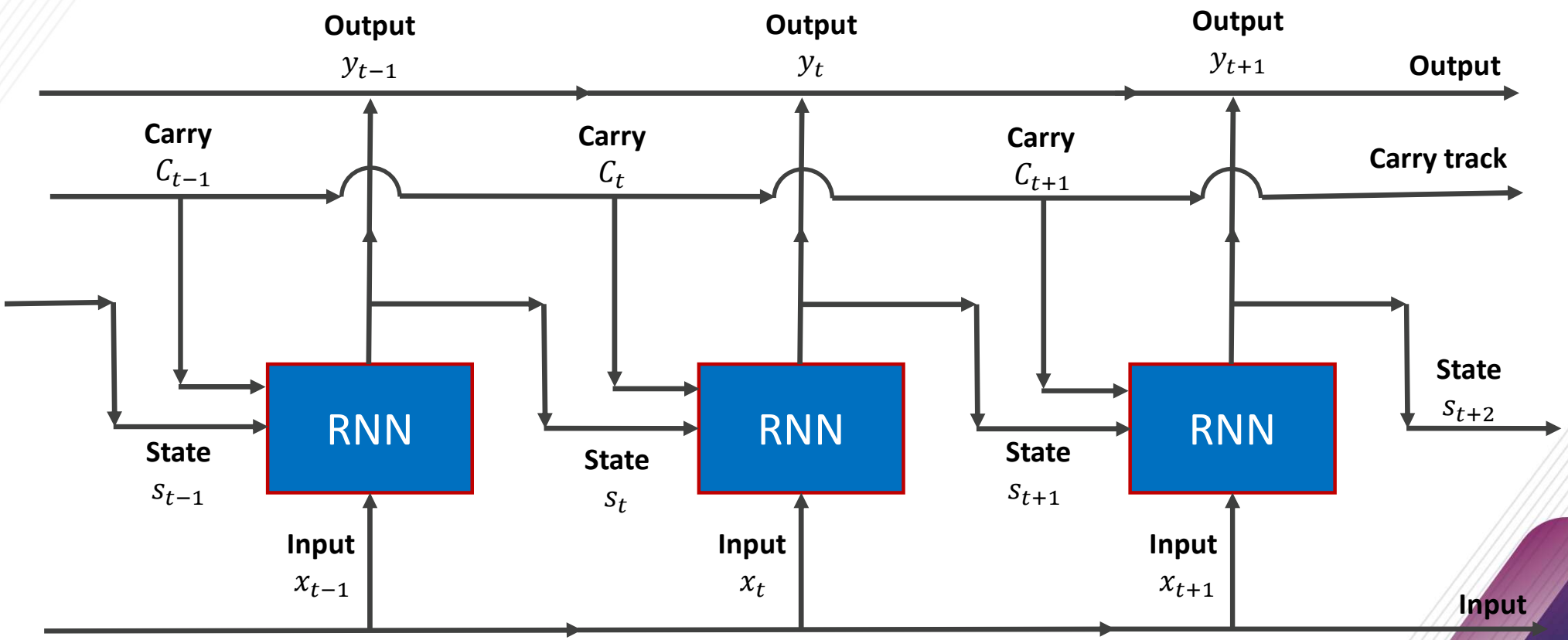
RNN Simple



Una RNN puede desdoblarse a lo largo del tiempo y visualizarla como sigue:

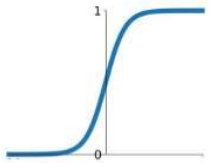


Actualización de los gradientes (*backpropagation*)



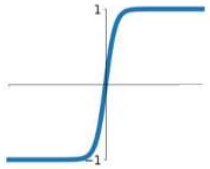
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



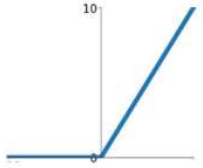
tanh

$\tanh(x)$

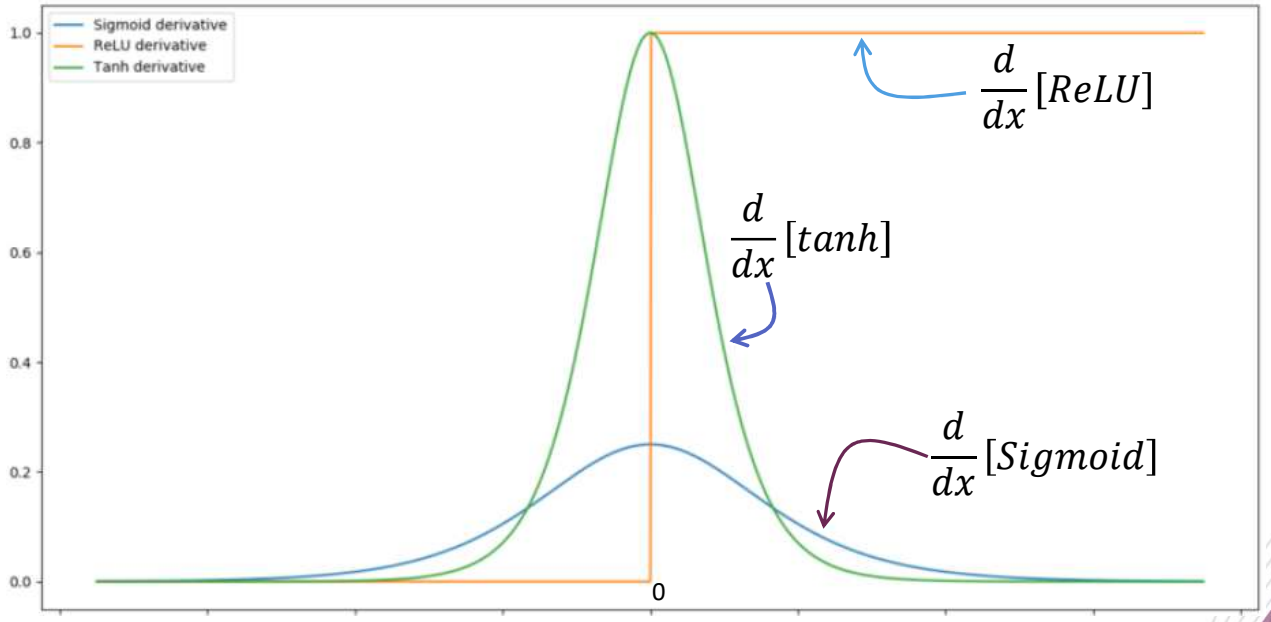


ReLU

$\max(0, x)$



Elegir funciones de activación cuya derivada no se contraiga *demasiado pronto* cuando $x > 0$.



LSTM : Long Short Term Memory & GRU : Gated Recurrent Unit

- En la práctica, el problema de la disminución del gradiente hace que no se puedan entrenar redes con demasiadas capas.
- Para evitar dicho problema se conserva información del pasado a lo largo del tiempo y se va incrustando en los bloques convolucionales para preservar la información.
- En el artículo de Hochreiter & Scmidhuber, 1997, se define la arquitectura LSTM para atacar por primera vez el problema de la memoria y la disminución del gradiente.
- Se utiliza además la función ReLU.
- Los modelos LSTM y GRU son los más conocidos que preservan información a lo largo del tiempo. Podemos decir que todos son de la familia de modelos secuenciales o recurrentes.
- GRU es un modelo más simple que LSTM, en particular LSTM conserva memoria a corto y largo plazo de manera independiente, mientras que GRU tiene un solo tipo de memoria.

Paper: Long short-term memory
S. Hochreiter & J. Scmidhuber. 1997

https://ieeexplore.ieee.org/abstract/document/6795963