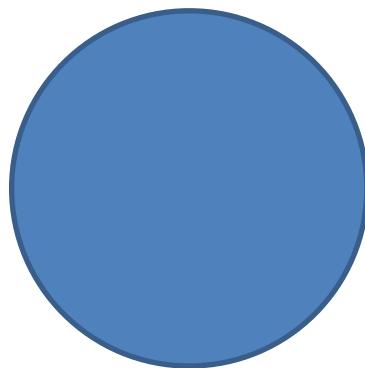


Machine Learning & Deep Learning

Deep Learning

Profesor: Carlos Moreno Morera





Contenido

01

Introducción a las Redes de Neuronas Artificiales

02

Perceptrón multicapa

03

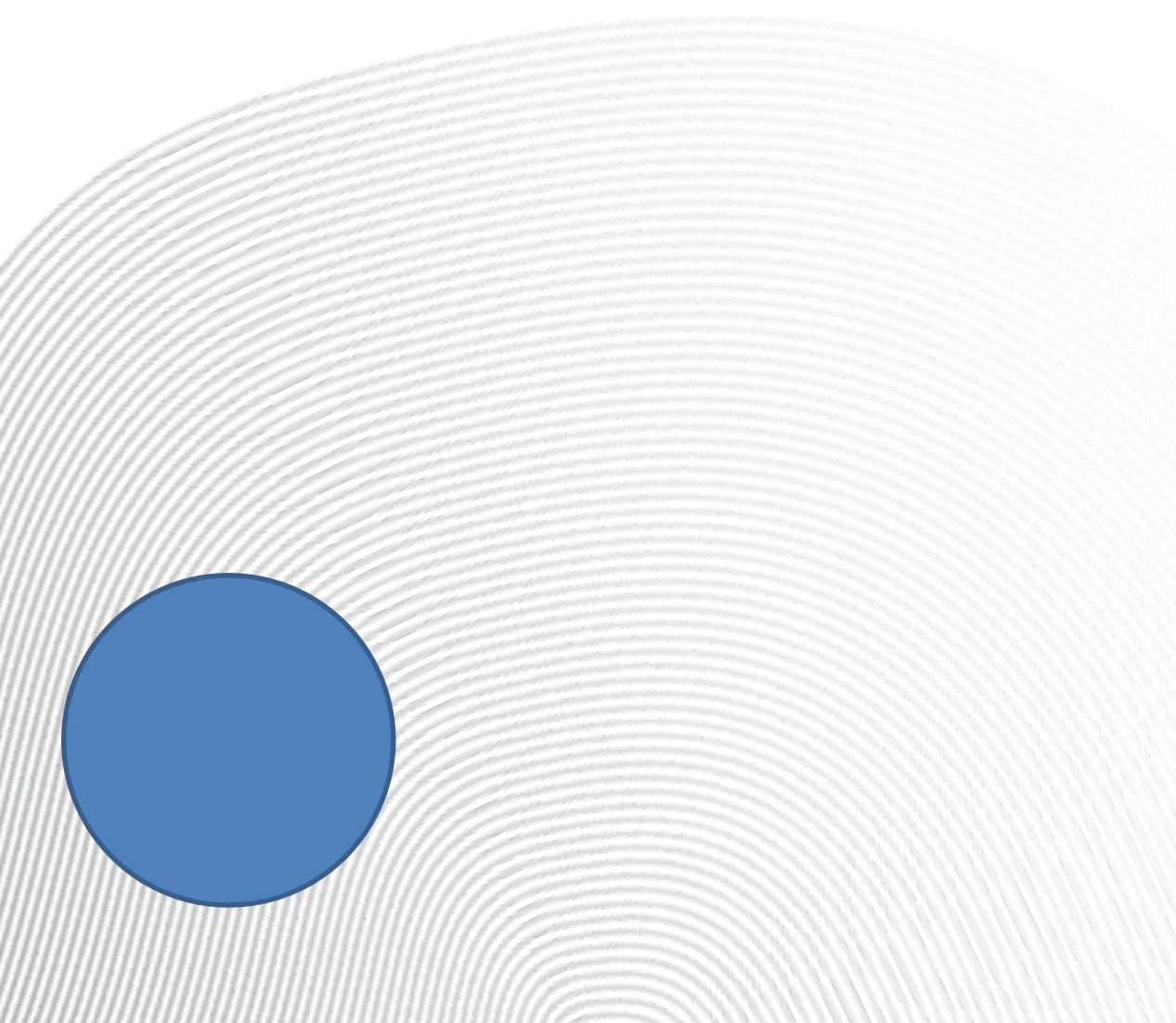
Autoencoders

04

Redes de Neuronas Recurrentes

05

Redes de Neuronas Convolucionales

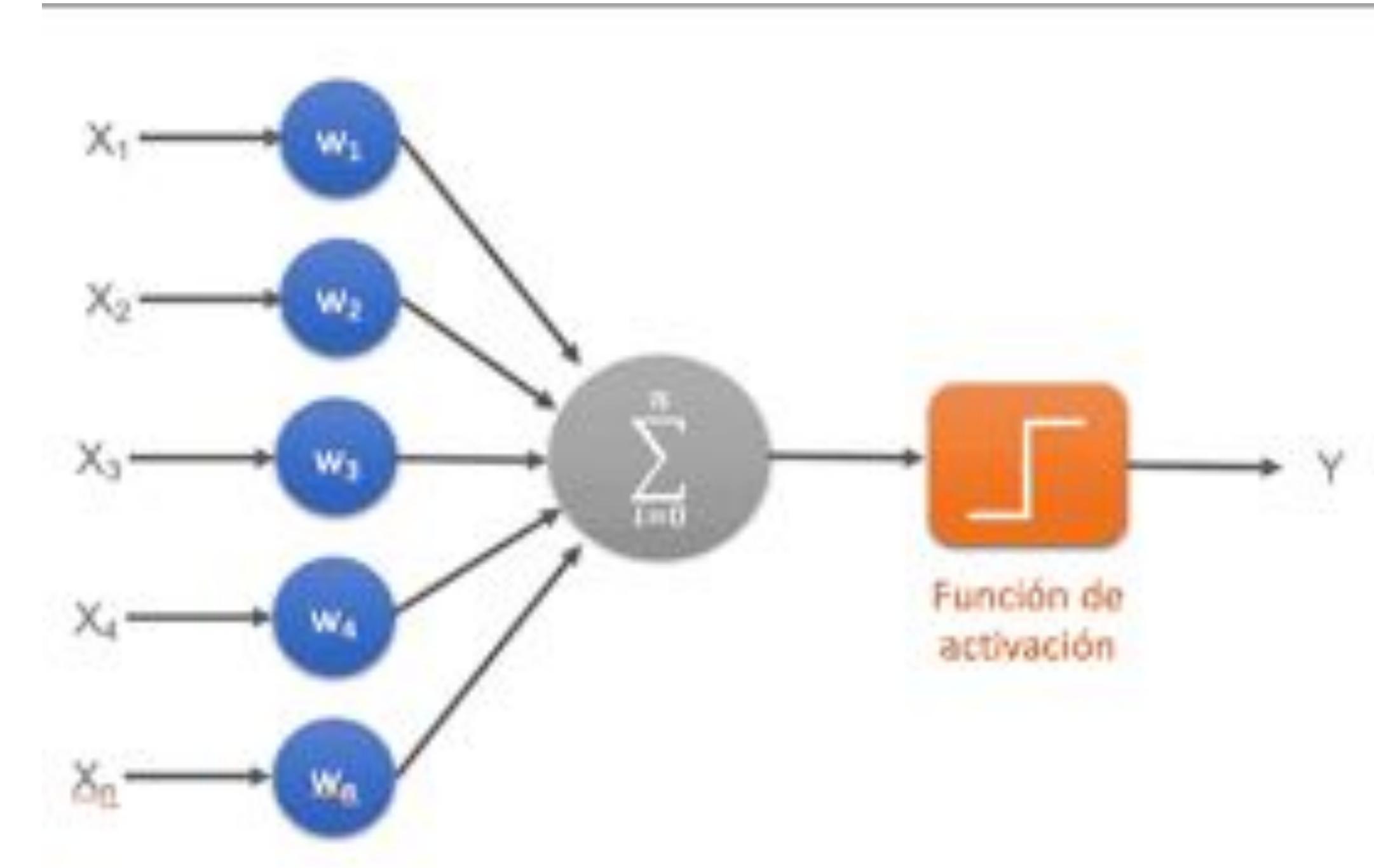
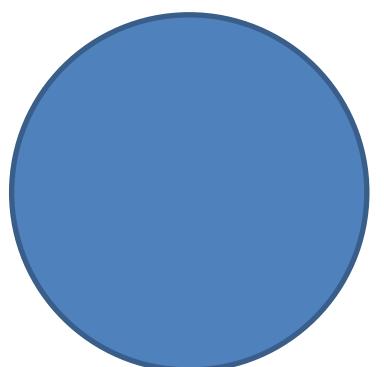




01

Introducción a las Redes de Neuronas Artificiales

Se introducen los conceptos básicos de las RNA así como la definición e implementación del perceptrón simple



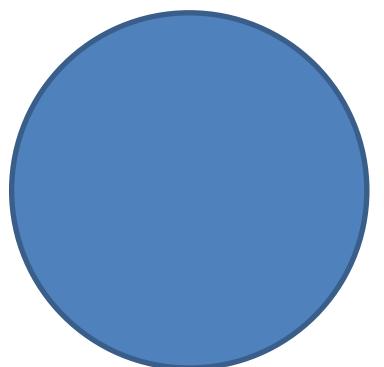
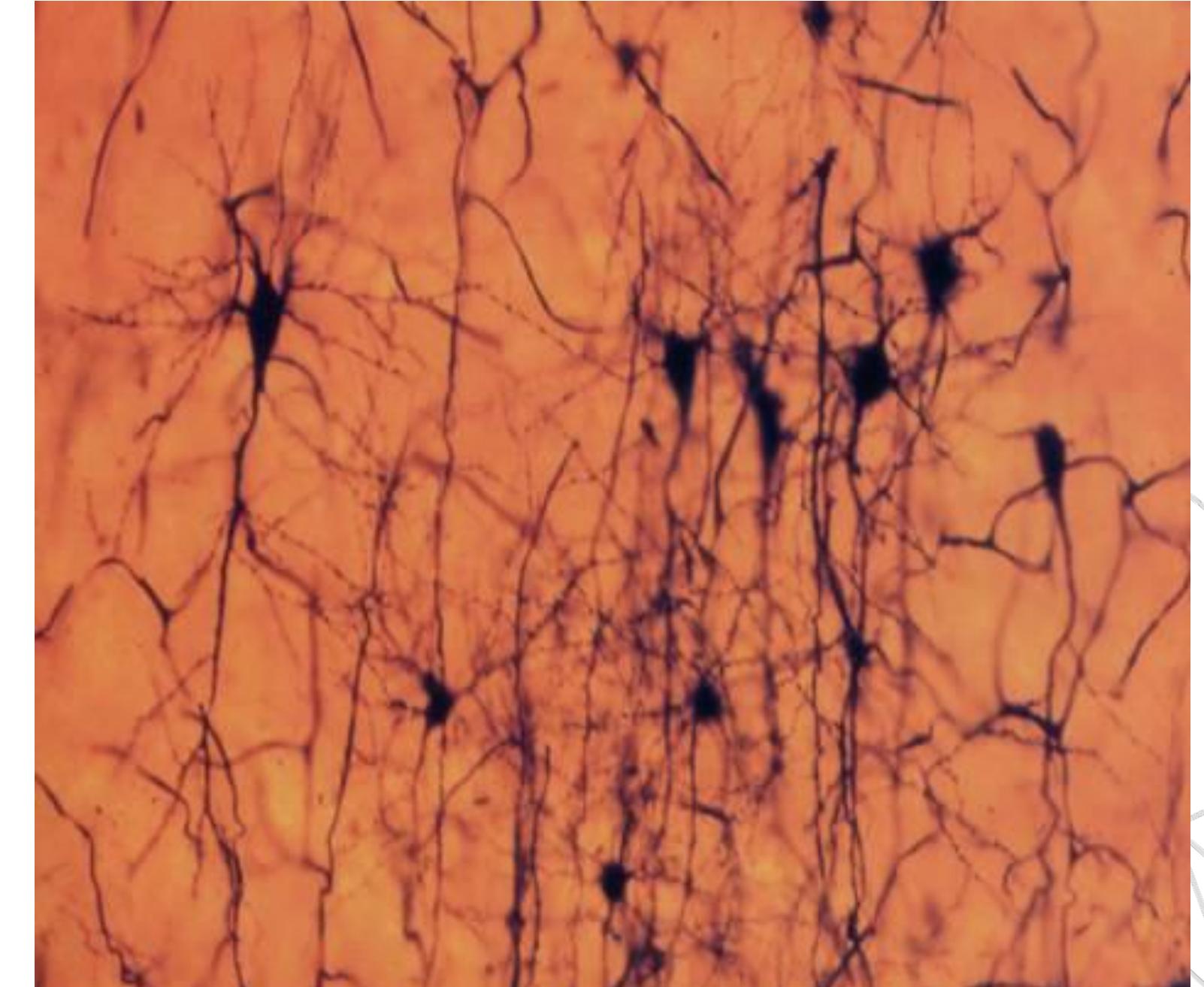


Introducción

Las Redes Neuronales Aritificiales (RNA o ANN en inglés) son sistemas inteligentes que emulan el funcionamiento del sistema nervioso central de los animales.

El cerebro se compone de neuronas interconectadas entre sí, que se transmiten señales en forma de impulsos electro-químicos.

Una RNA se compone de neuronas simuladas (neuronas artificiales) interconectadas entre sí que se transmiten información en forma de números.





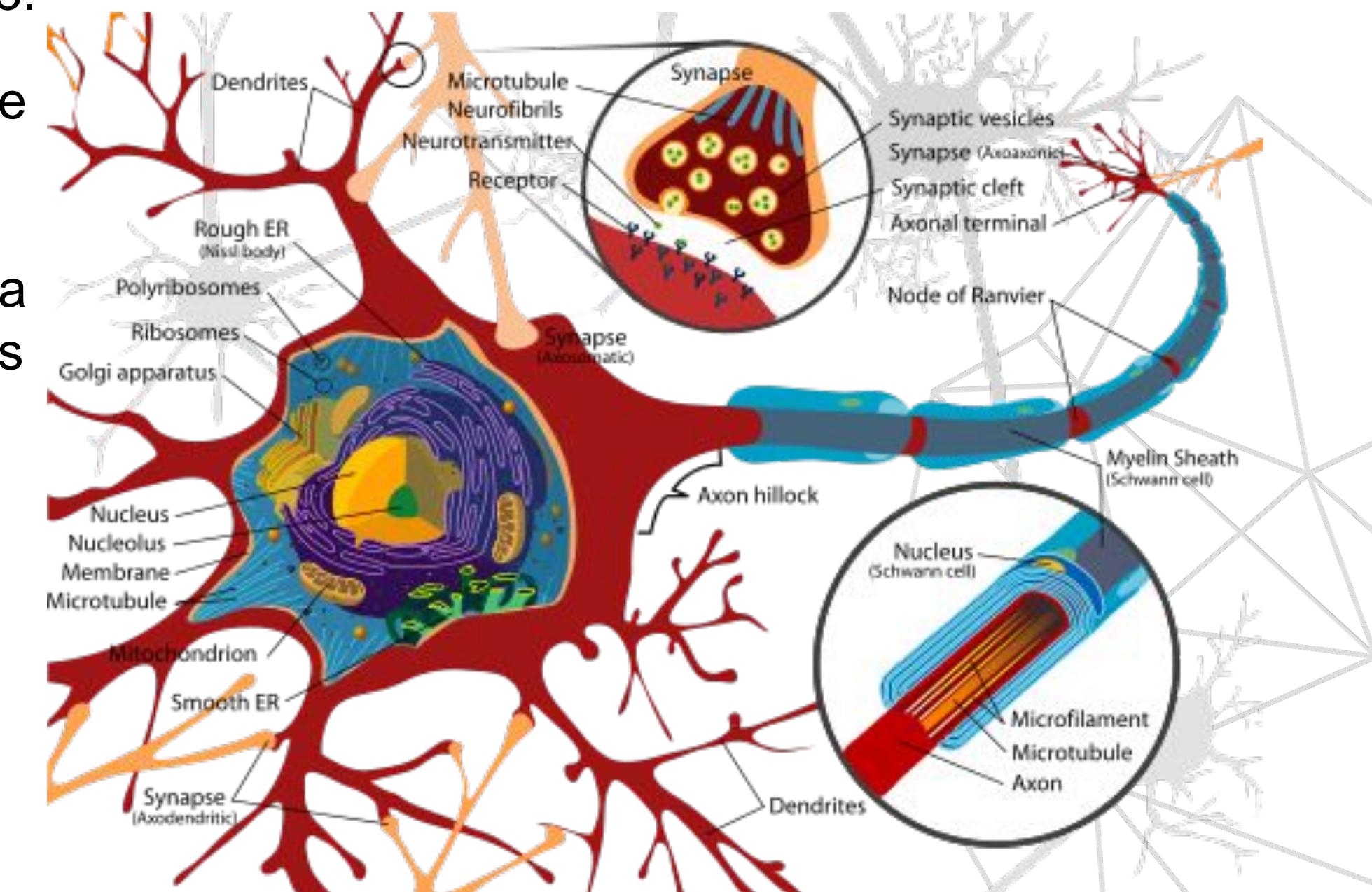
Introducción

Nuestros cerebros cuentan con cientos de millones de neuronas que se interconectan para formar redes neuronales que procesan información.

Cada neurona trabaja como un simple procesador. La interacción masiva entre ellas y su procesamiento en paralelo hacen posible las capacidades del cerebro.

En la imagen se pueden observar las partes de una neurona biológica que transmite información.

Las neuronas mandan señales eléctricas de salida a través de los axones. Asimismo, reciben señales químicas de entrada a través de las dendritas.

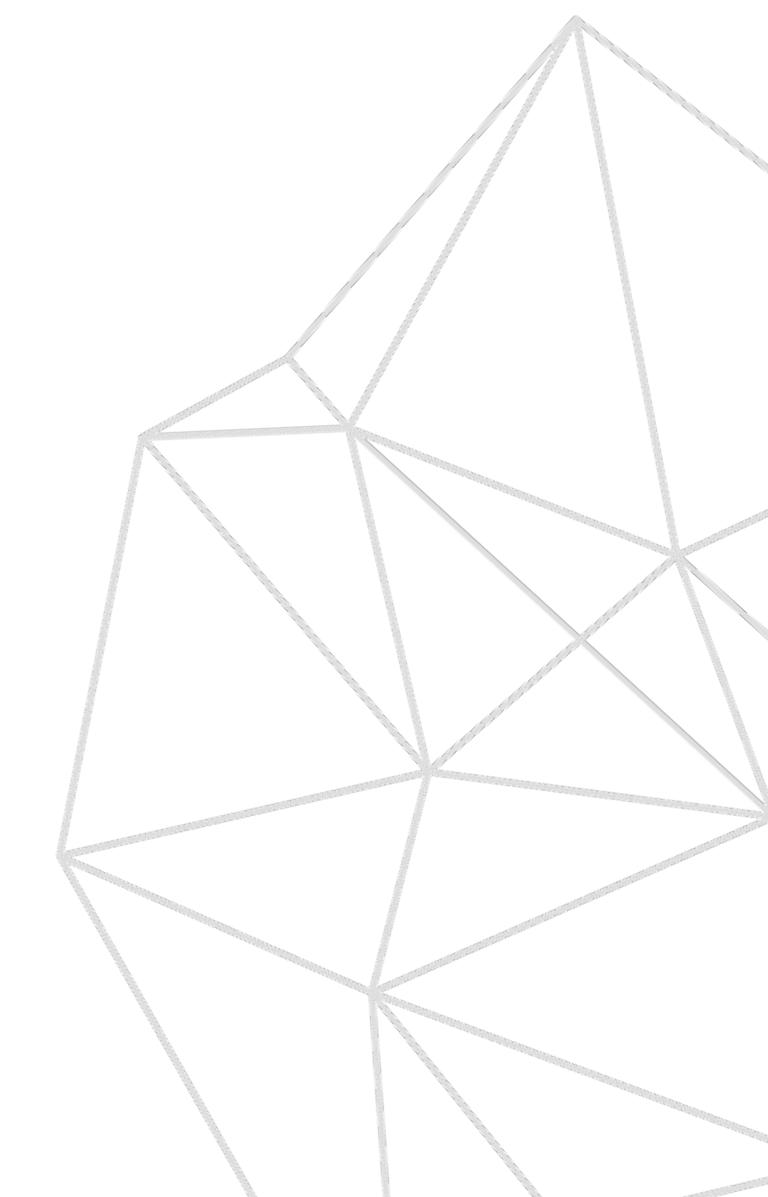
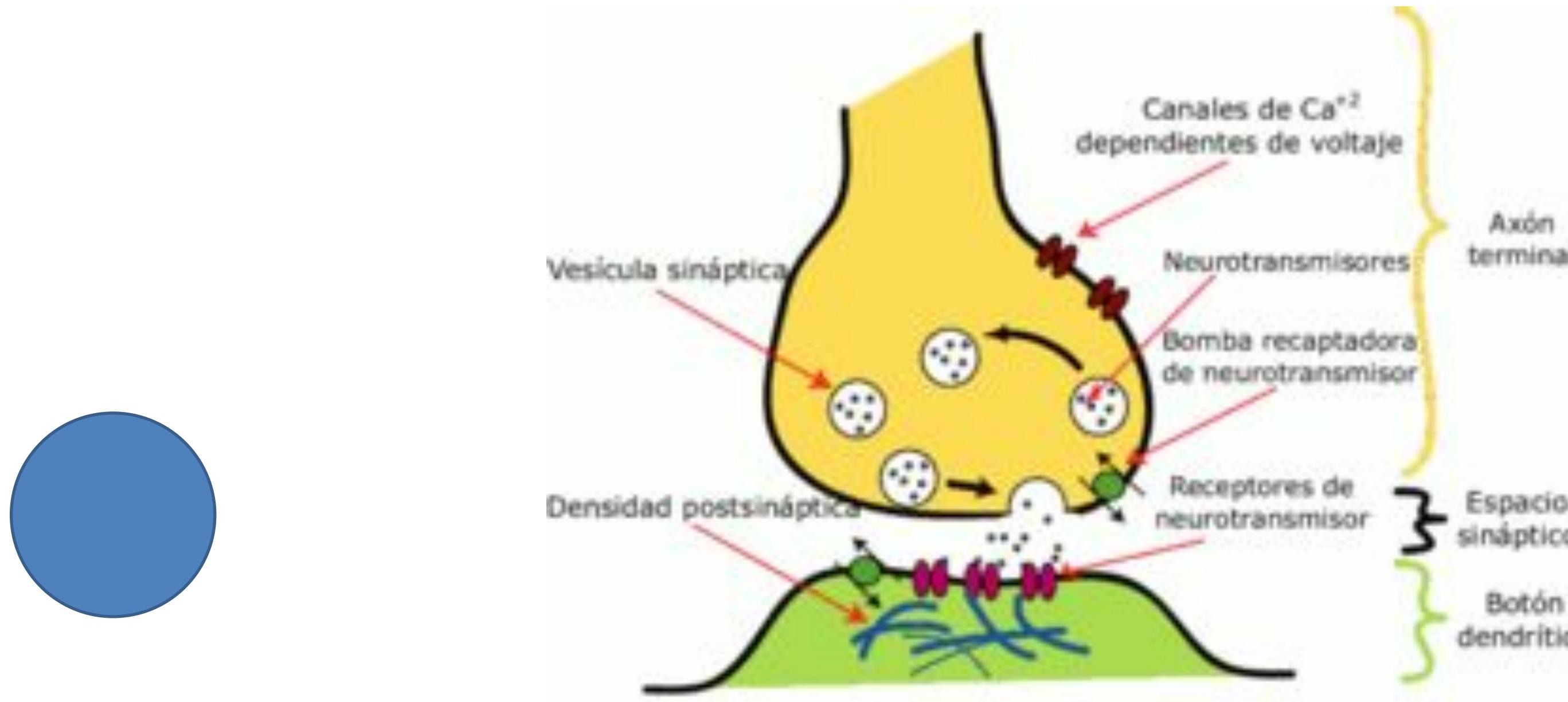




Introducción

Una neurona biológica almacena y transmite información. La conexión entre cada axón y dendrita se llama sinapsis. En la sinapsis ocurre la transformación de una señal eléctrica (salida) en una química (entrada).

Cuando la cantidad de receptores químicos recibidos por una neurona a través de la dendrita supera cierto umbral, dicha neurona es activada y reacciona desencadenando un acción.

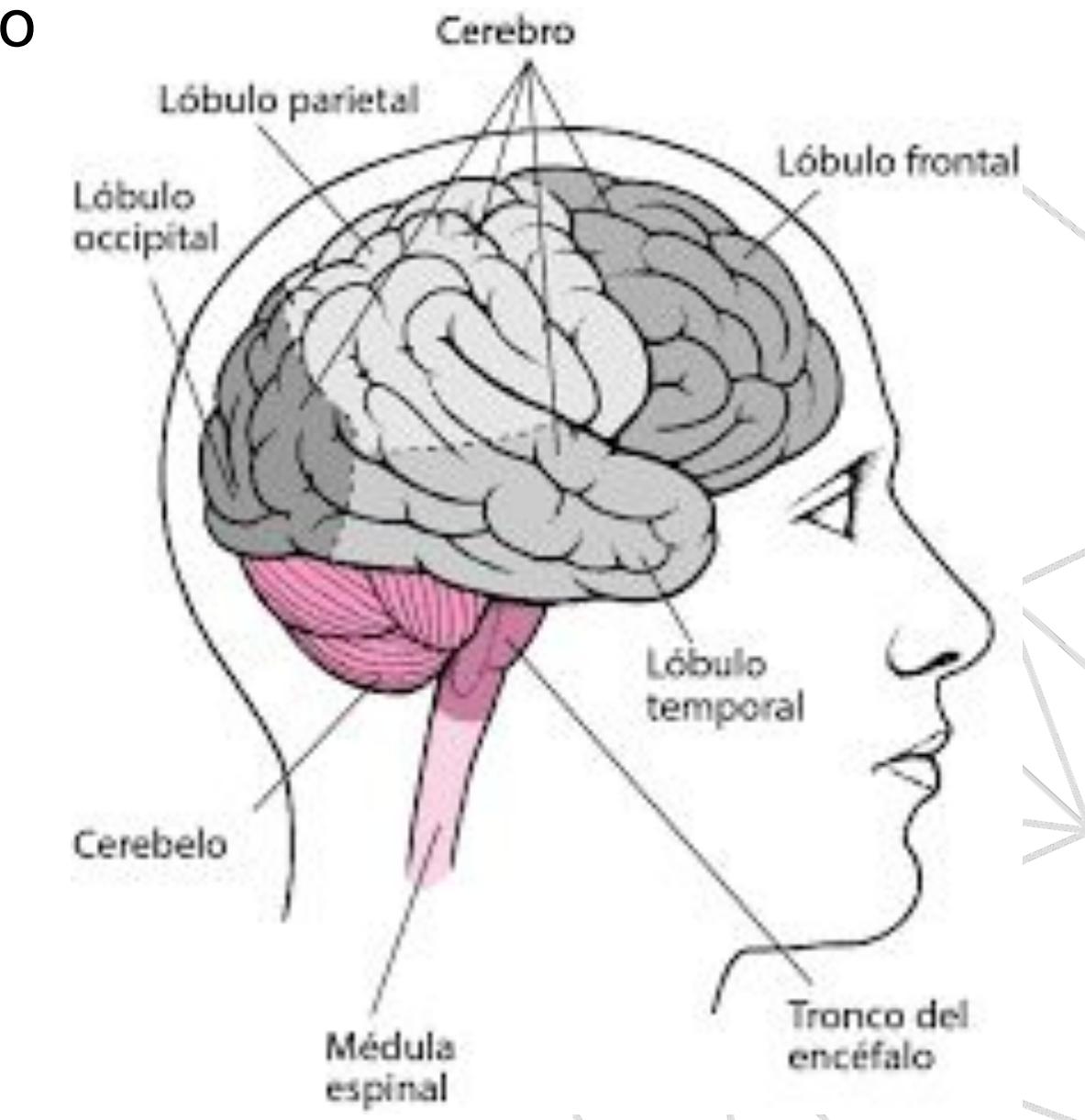
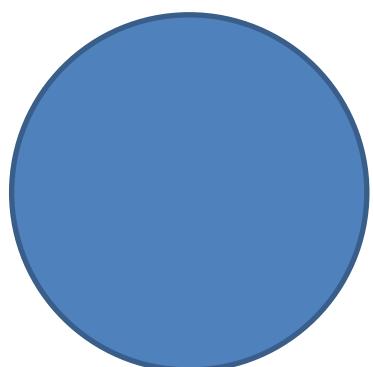




Introducción: características deseables

El cerebro humano constituye un ordenador muy potente, capaz de interpretar información imprecisa suministrada por los sentidos a un ritmo muy rápido. Posee además las siguientes propiedades:

- Es robusto y tolerante a errores: mueren neuronas diariamente sin afectar a su rendimiento.
- Es flexible: se ajusta a nuevos entornos por aprendizaje, por lo que no hay que reprogramarlo.
- Puede manejar información difusa, con ruido o inconsistente.
- Está altamente paralelizado.
- Es pequeño, compacto y consume poca energía.

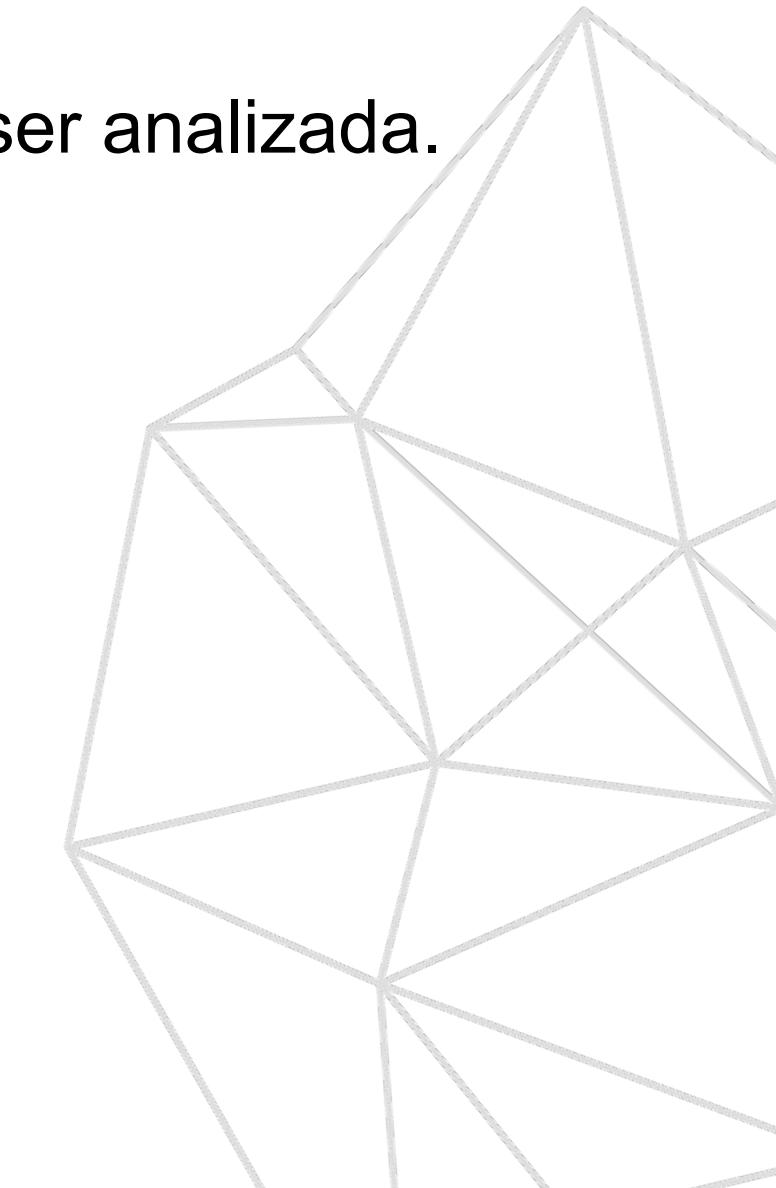
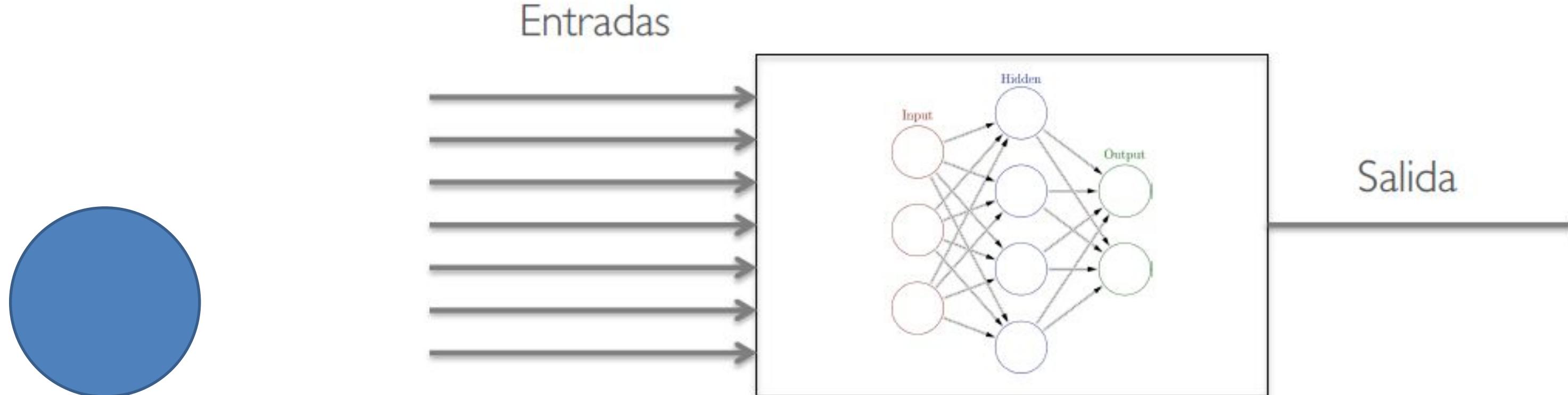




Introducción: aplicación

Por lo general se emplean redes neuronales artificiales (RNA) cuando:

- Se está en presencia de un problema de aprendizaje altamente multidimensional (diversas entradas con valores discretos os reales).
- Los datos tienen ruido.
- Cuando la función objetivo es compleja y/o tiene forma desconocida y no requiere ser analizada.

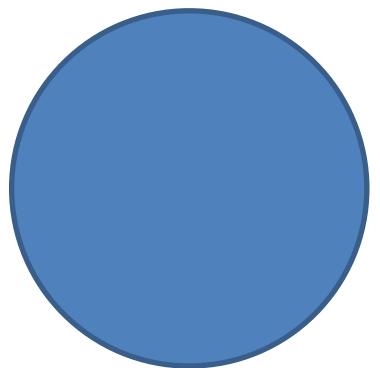




Introducción: aplicación

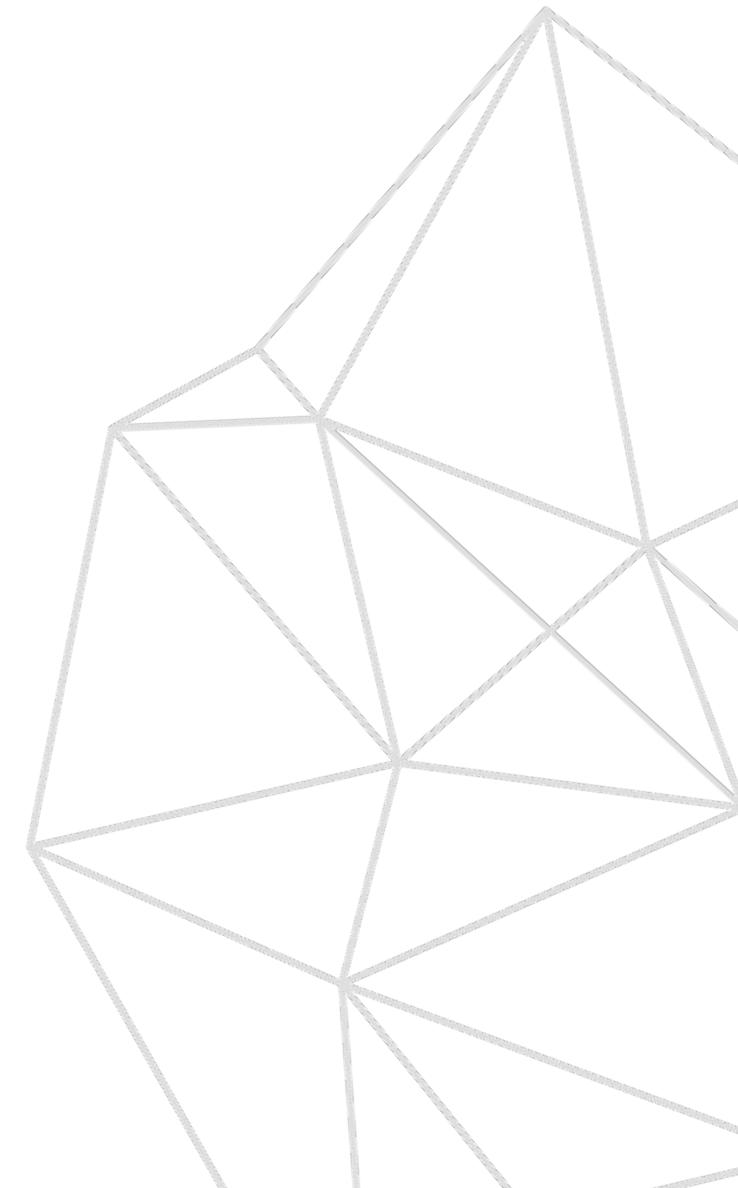
Las aplicaciones de las redes neuronales artificiales son:

- Procesamiento de imágenes y de voz
- Procesamiento de lenguaje natural
- Reconocimiento de patrones
- Planificación
- Interfaces adaptativas para sistemas hombre/máquina.
- Predicción
- Control y optimización
- Filtrado de señales
- Aprendizaje de comportamientos y su simulación



Son útiles para aprendizaje de tipo:

- Supervisado (regresión y clasificación)
- No supervisado (búsqueda de patrones)
- Por refuerzo (toma de decisiones)

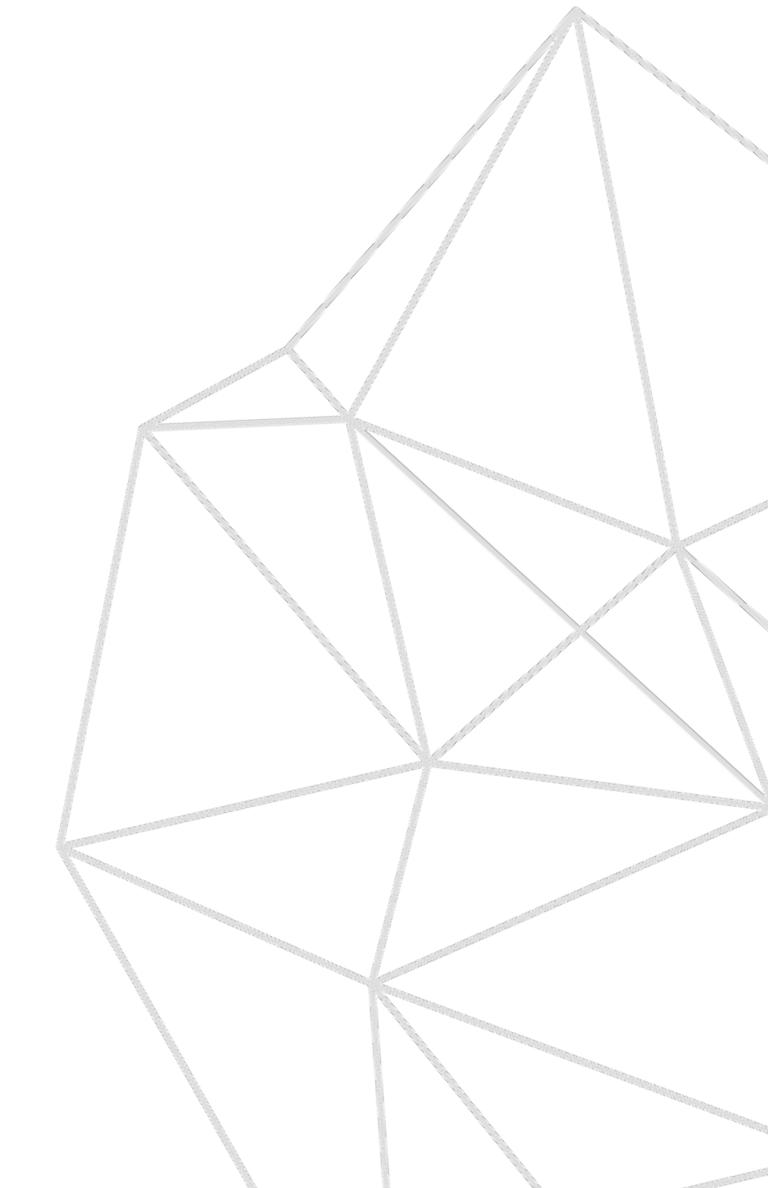
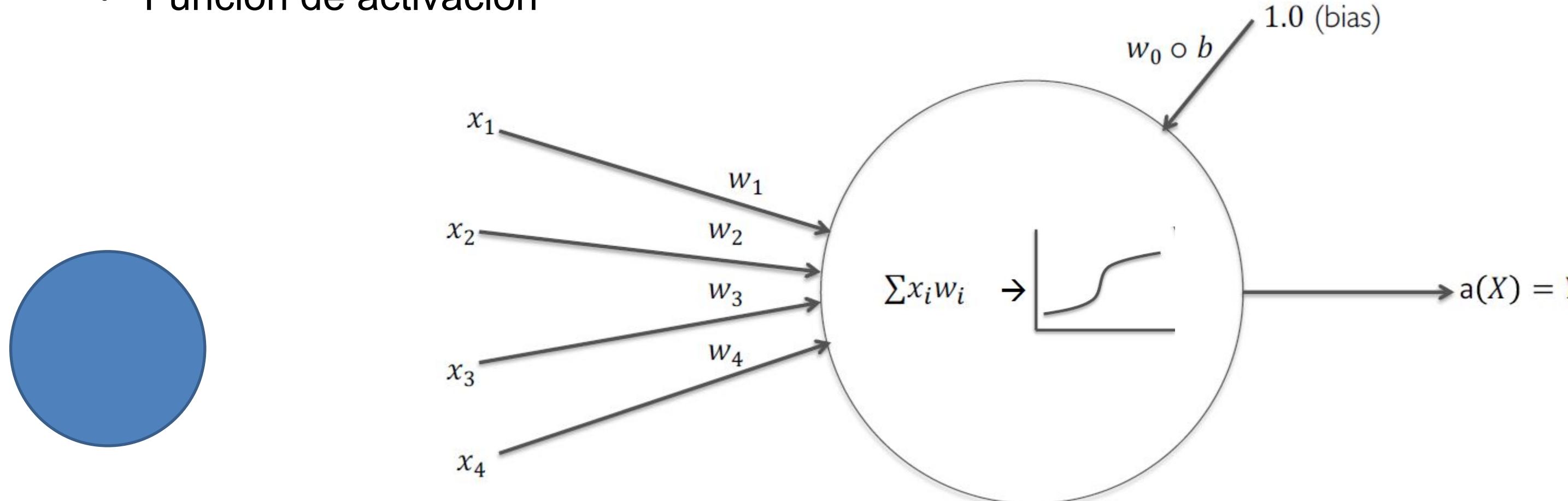




Perceptrón simple

Cada neurona (o perceptrón) consta de un conjunto de entradas y una única salida:

- Entradas ponderadas
- Bias: entrada reguladora
- Sumatorio de las entradas ponderadas
- Función de activación





Perceptrón: función de activación

Determina si la salida se activará o el valor (es preferible que sea continua y no lineal). Algunos ejemplos son:

- Umbral (discontinua):

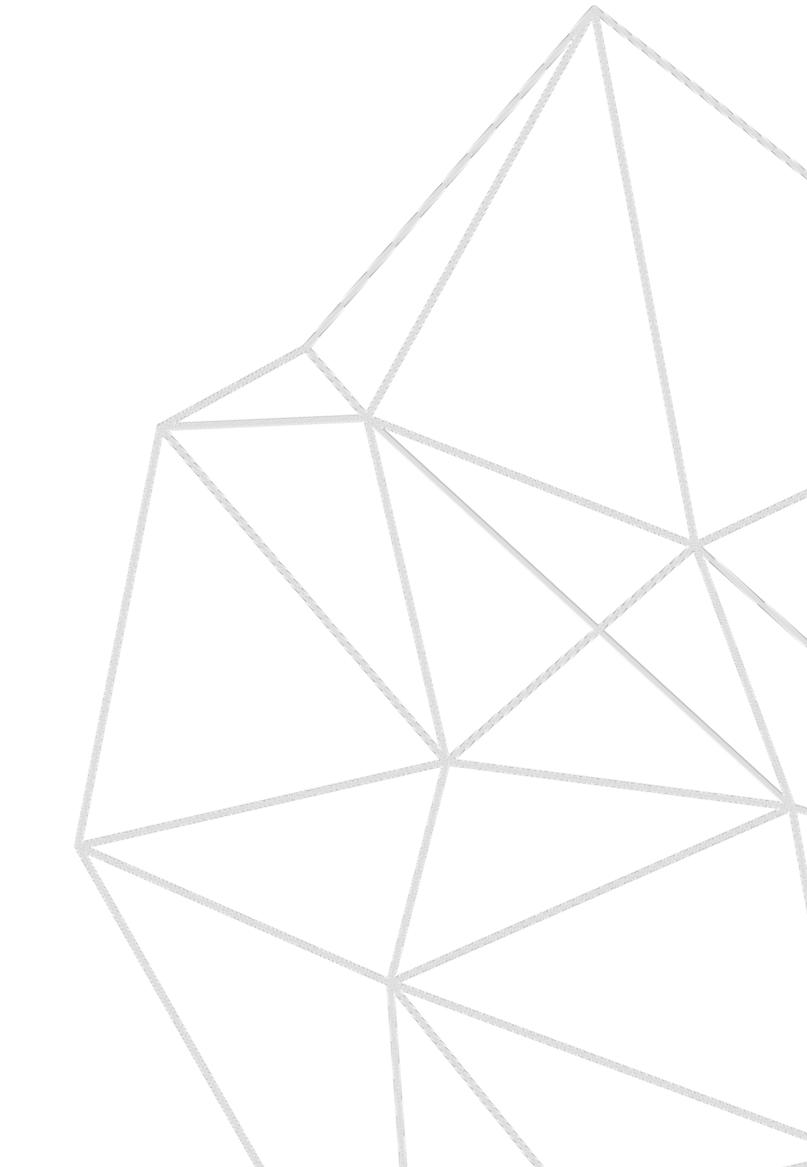
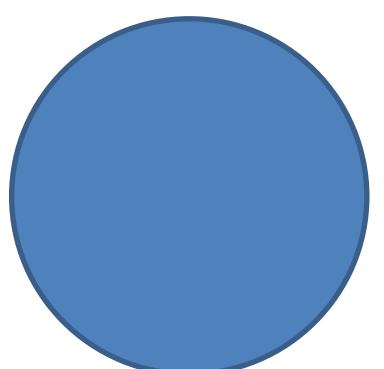
$$\text{umbral}(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}$$

- Sigmoidal (continua y diferenciable en $(0,1)$):

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

- Tangente hiperbólica (continua y diferenciable $(-1, 1)$):

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$





Perceptrón: representación matemática

Cálculo de la salida de una neurona, representación matemática:

- Umbral:

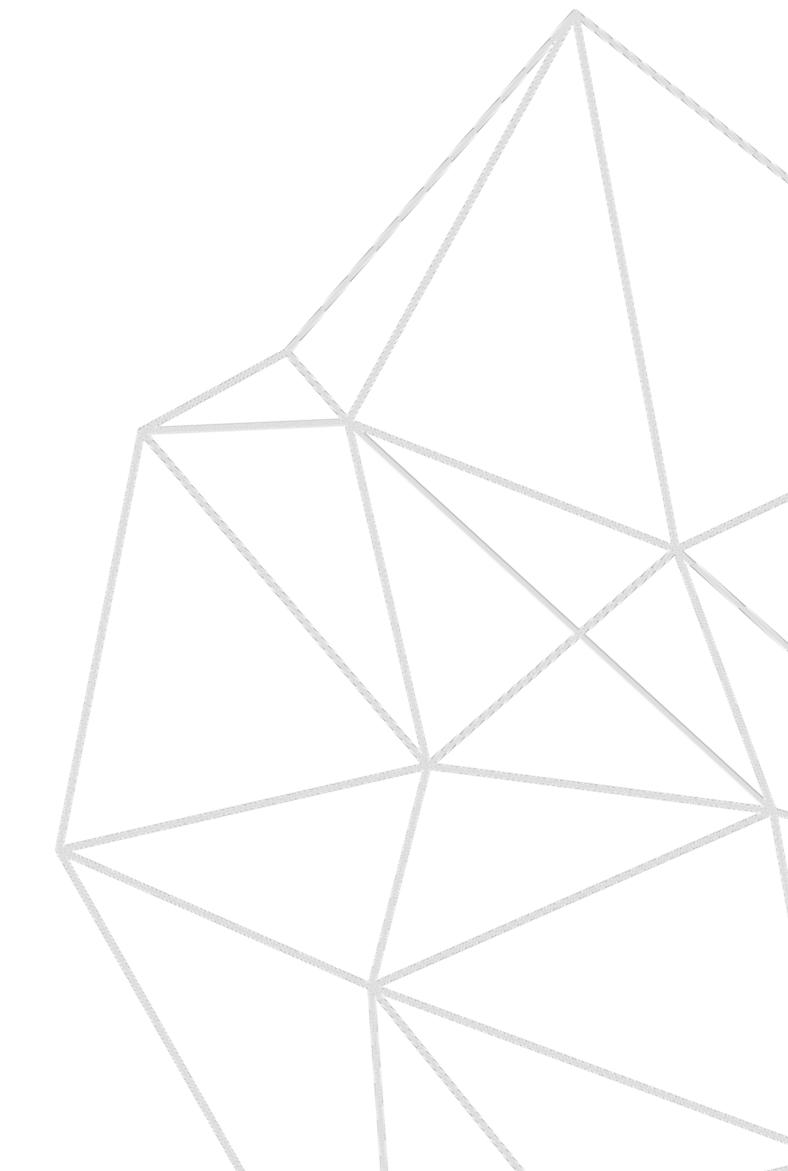
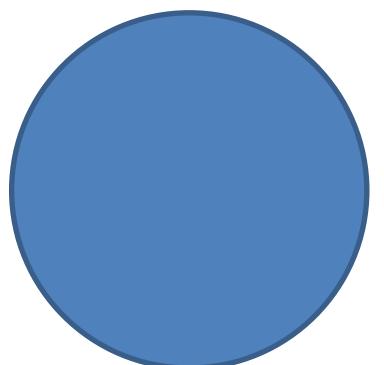
$$a(X) = \text{umbral}(z(X)) = \begin{cases} 1 & \text{si } \sum \omega_i x_i > 0 \\ 0 & \text{si } \sum \omega_i x_i \leq 0 \end{cases}$$

- Sigmoidal:

$$a(X) = \text{sig}(z(X)) = \frac{1}{1 + e^{-\sum \omega_i x_i}}$$

- Tangente hiperbólica:

$$\tanh(x) = \frac{1 - e^{-2 \sum \omega_i x_i}}{1 + e^{-2 \sum \omega_i x_i}}$$

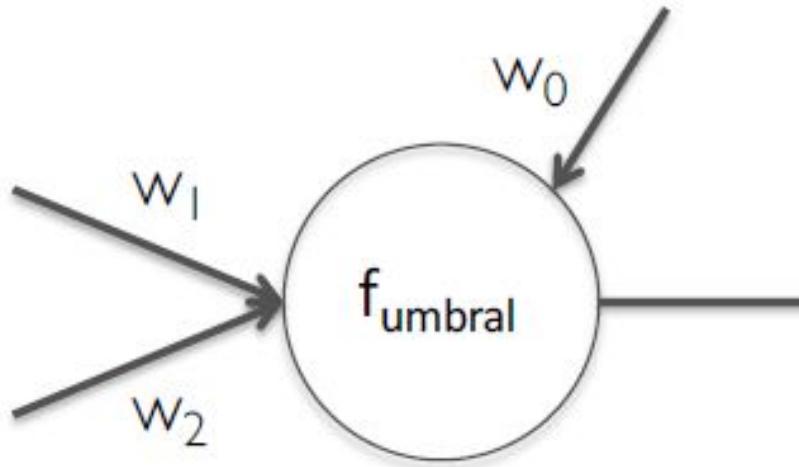




Perceptrón: ejemplos

- OR

0	0	0
0	1	1
1	0	1
1	1	1

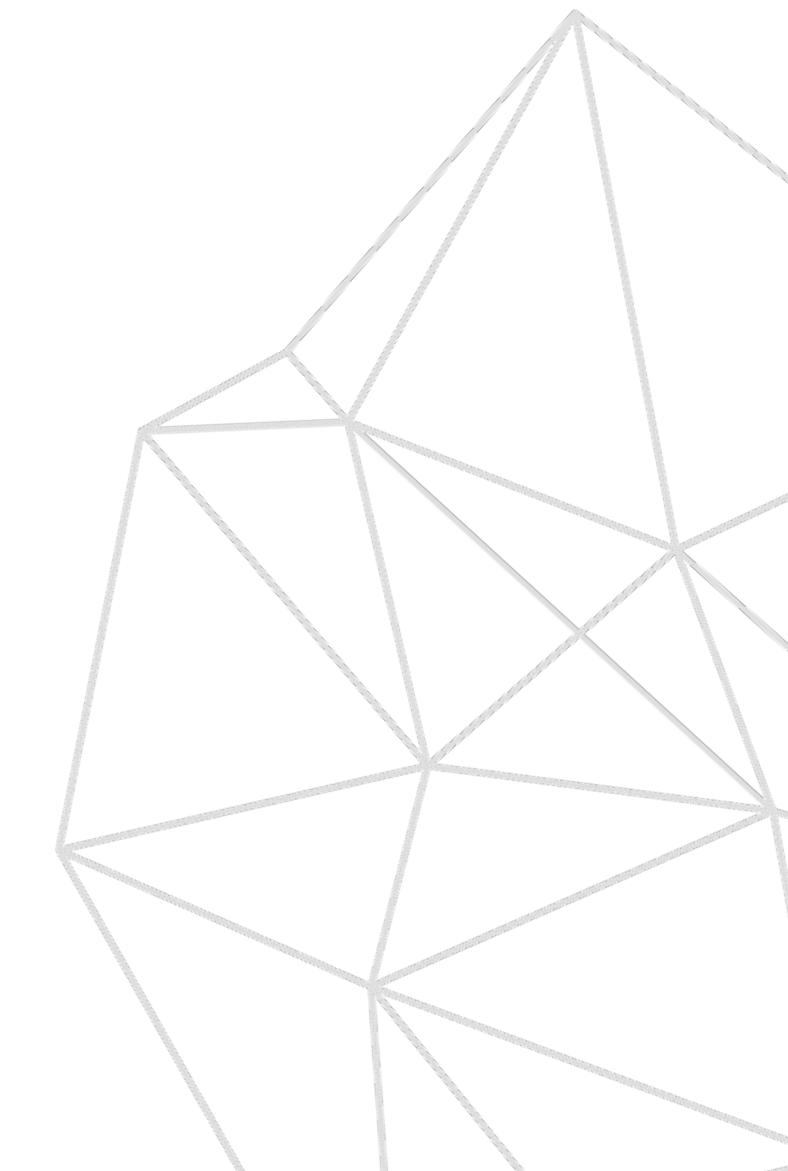
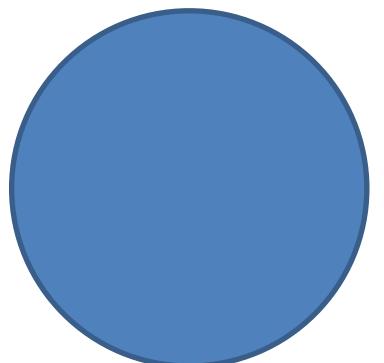


- AND

0	0	0
0	1	0
1	0	0
1	1	1

- XOR

0	0	0
0	1	1
1	0	1
1	1	0

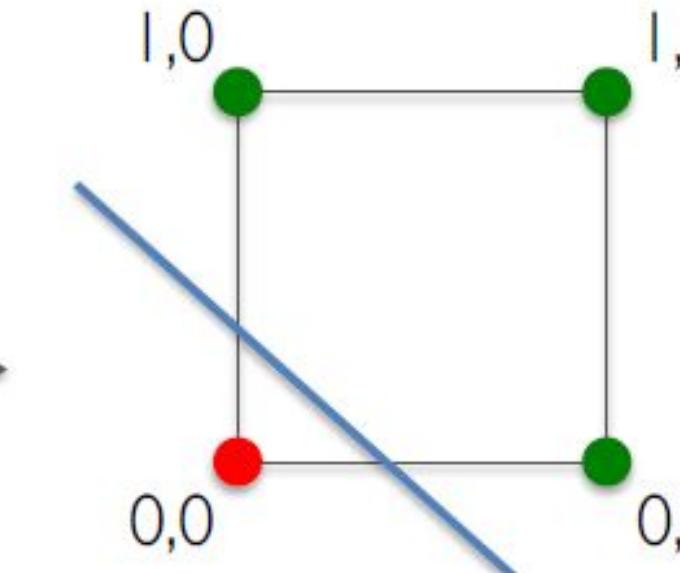
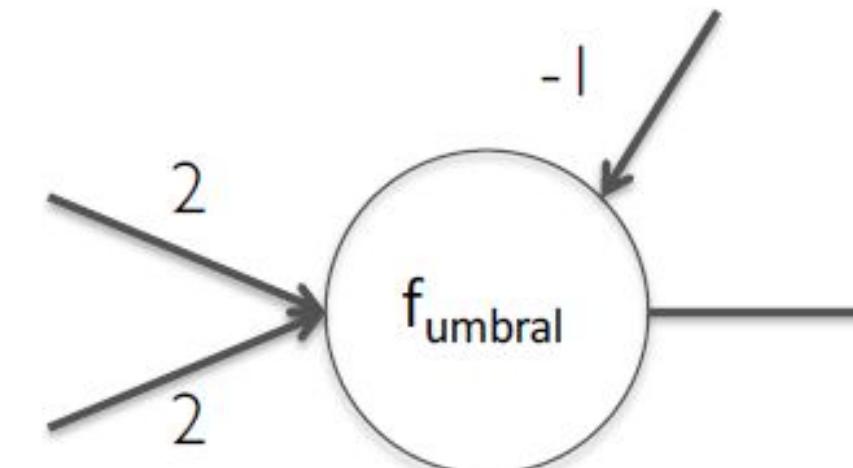




Perceptrón: ejemplos

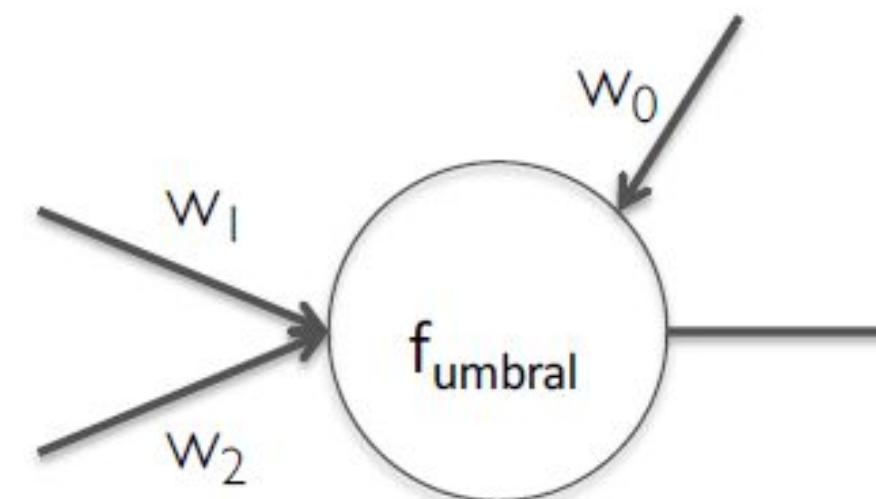
- OR

0	0	0
0	1	1
1	0	1
1	1	1



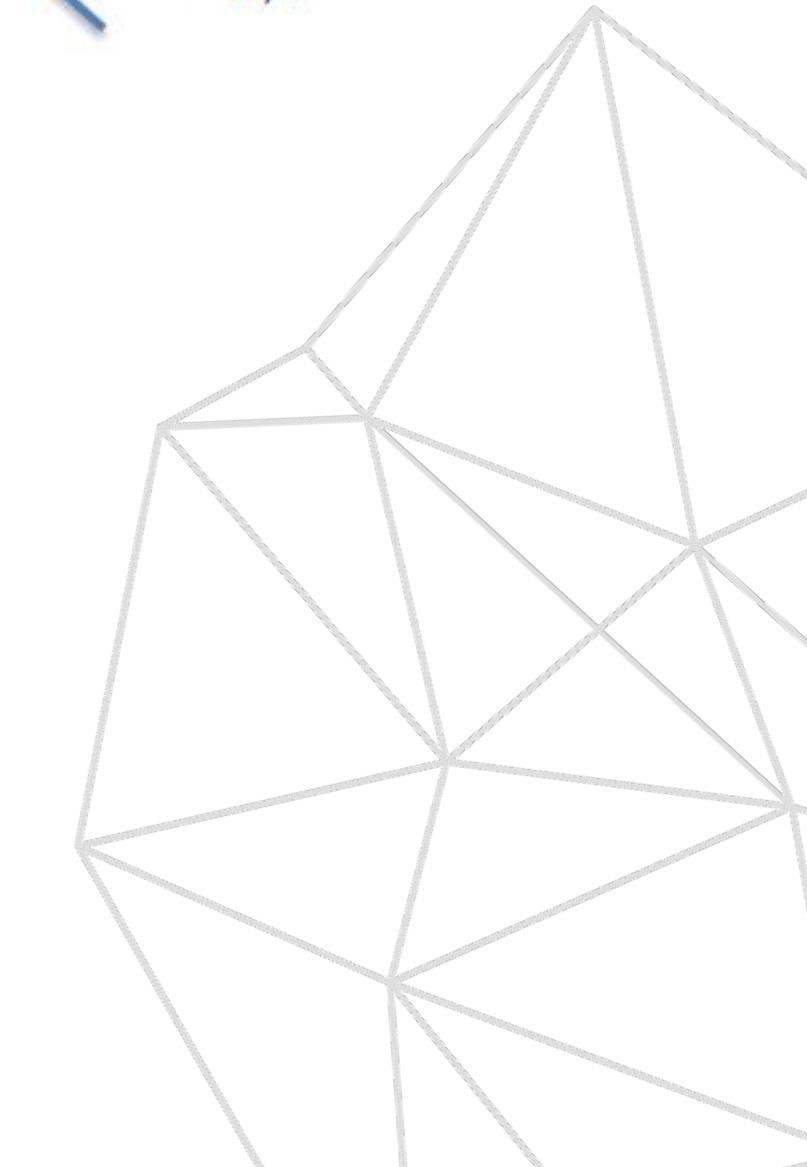
- AND

0	0	0
0	1	0
1	0	0
1	1	1



- XOR

0	0	0
0	1	1
1	0	1
1	1	0

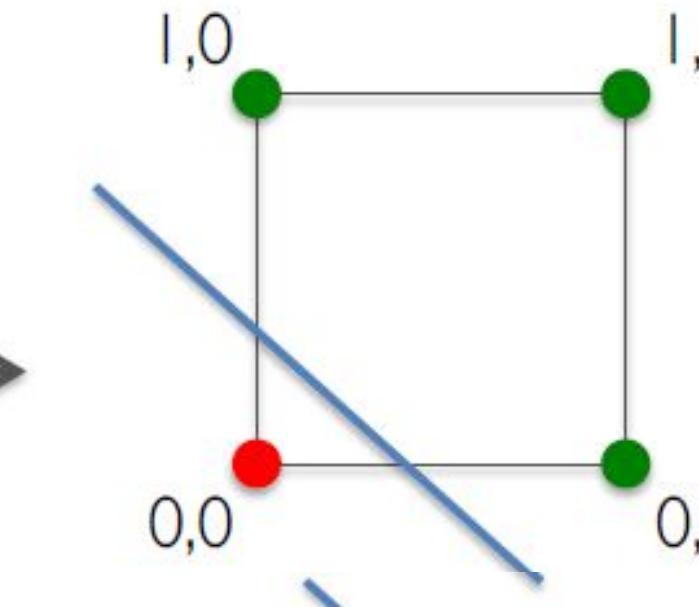
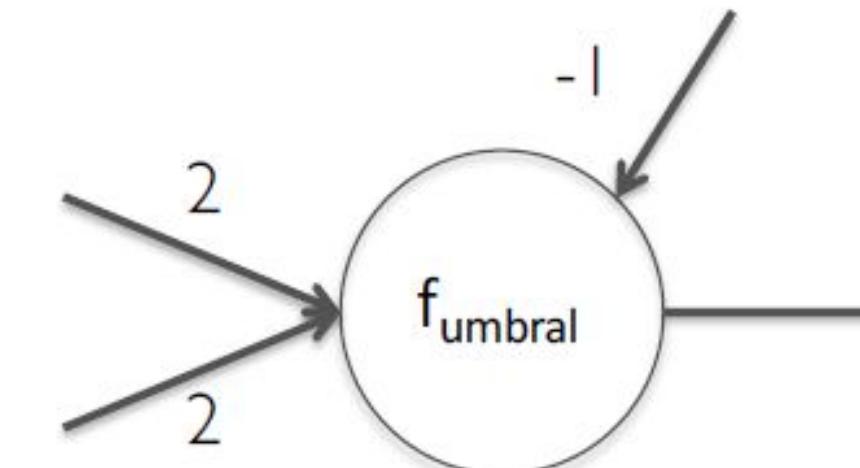




Perceptrón: ejemplos

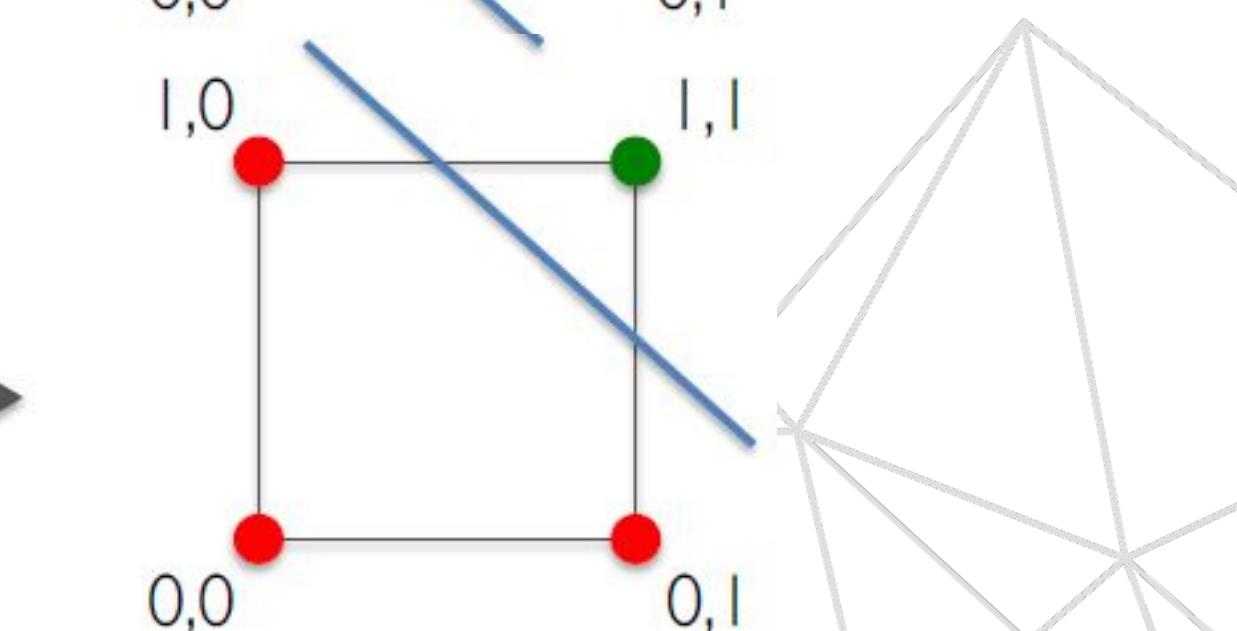
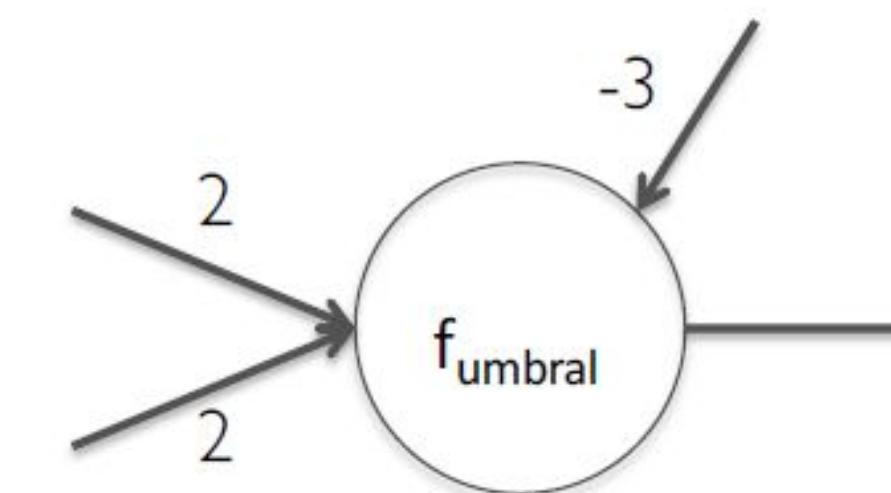
- OR

0	0	0
0	1	1
1	0	1
1	1	1



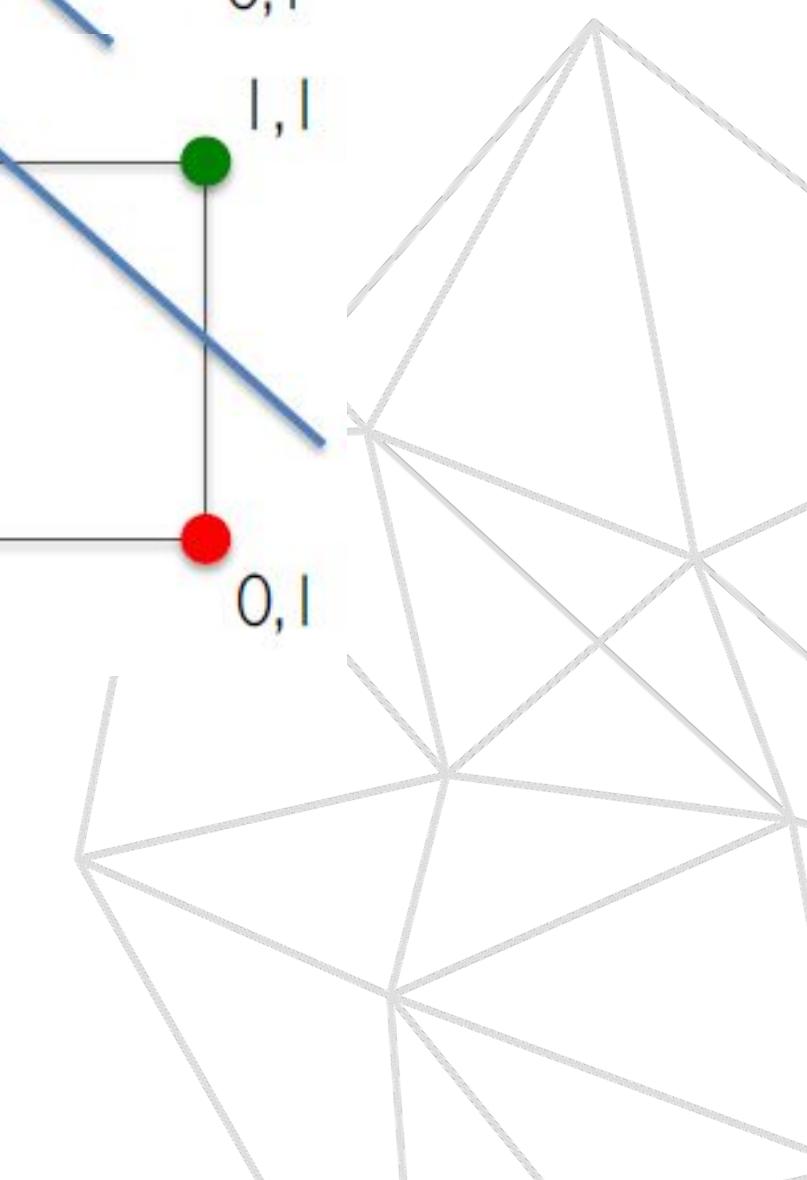
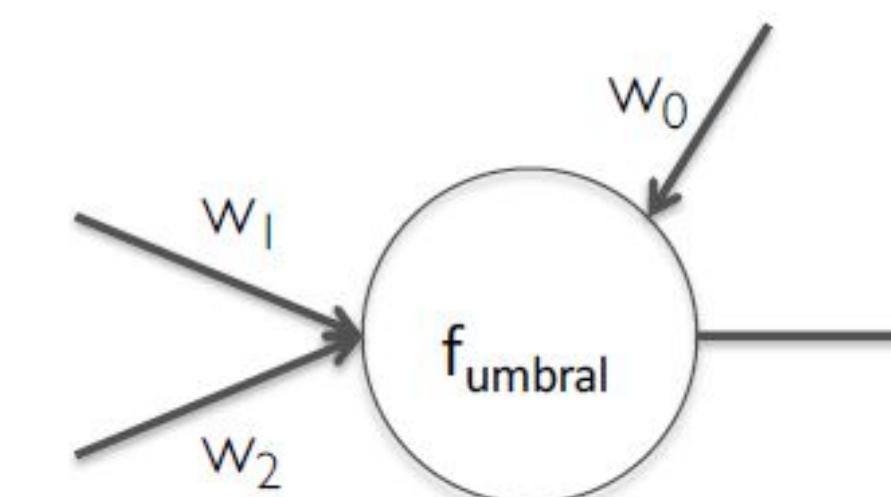
- AND

0	0	0
0	1	0
1	0	0
1	1	1



- XOR

0	0	0
0	1	1
1	0	1
1	1	0

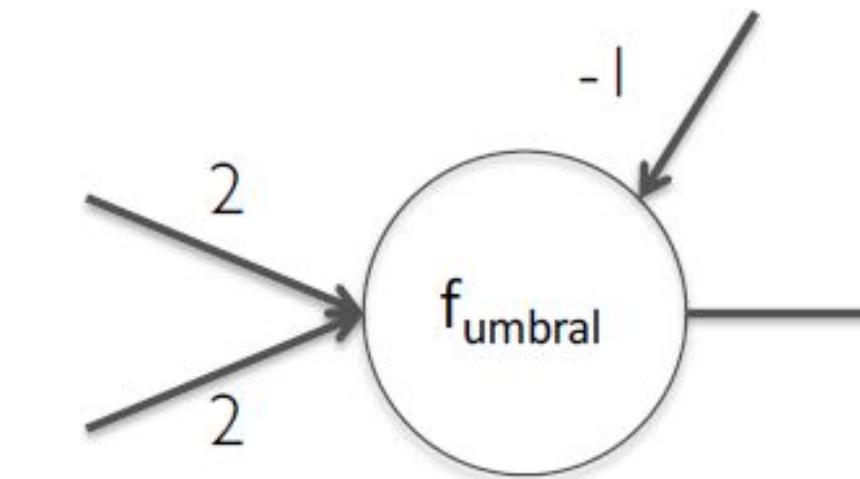




Perceptrón: ejemplos

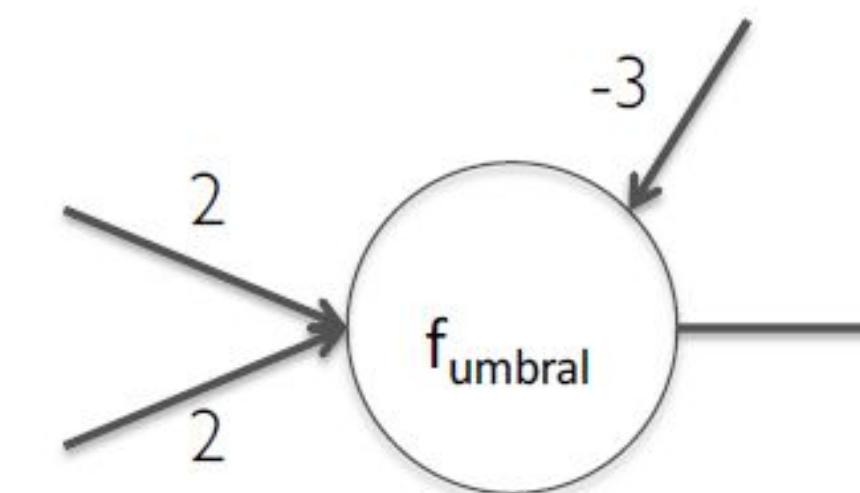
- OR

0	0	0
0	1	1
1	0	1
1	1	1



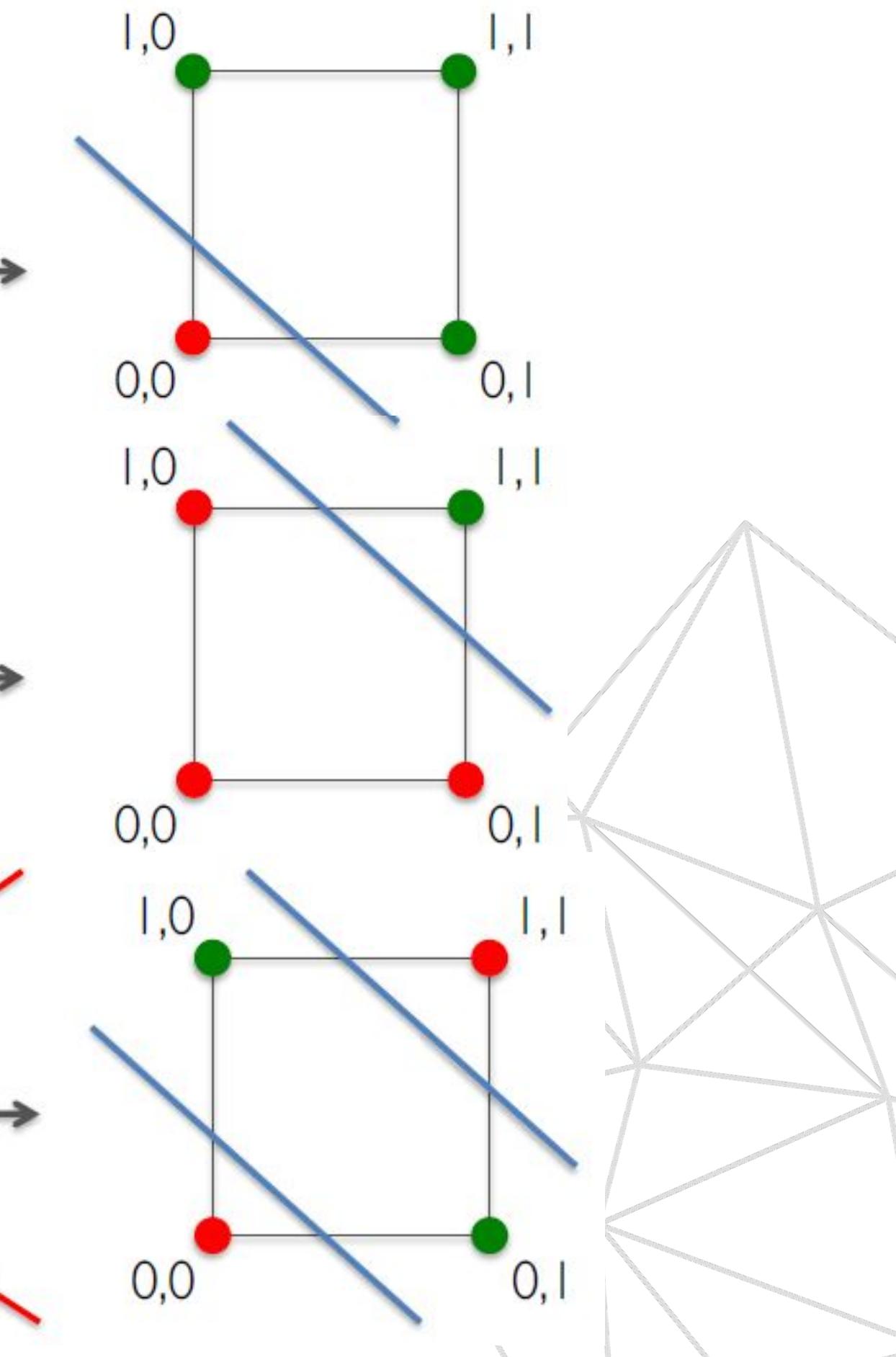
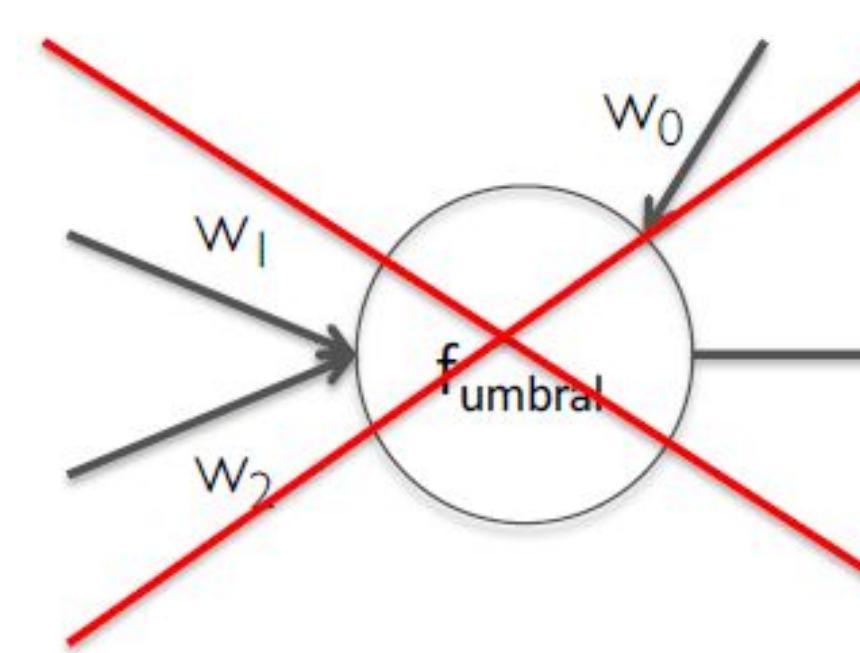
- AND

0	0	0
0	1	0
1	0	0
1	1	1



- XOR

0	0	0
0	1	1
1	0	1
1	1	0

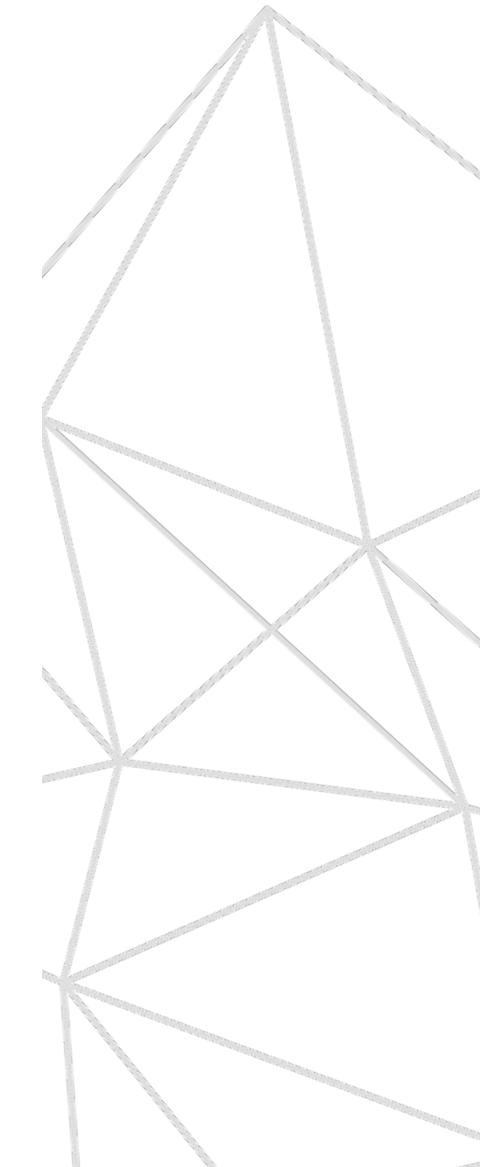
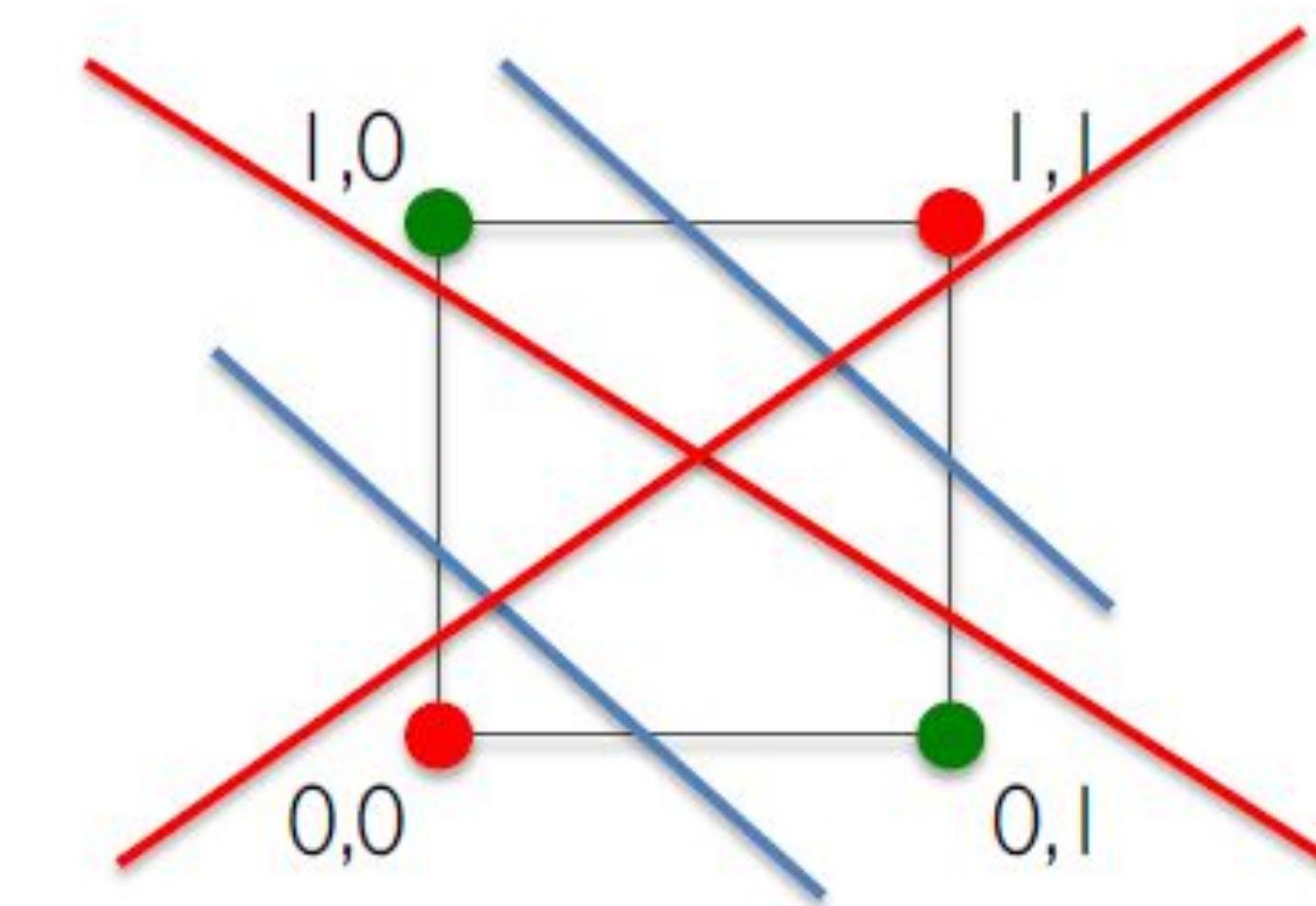
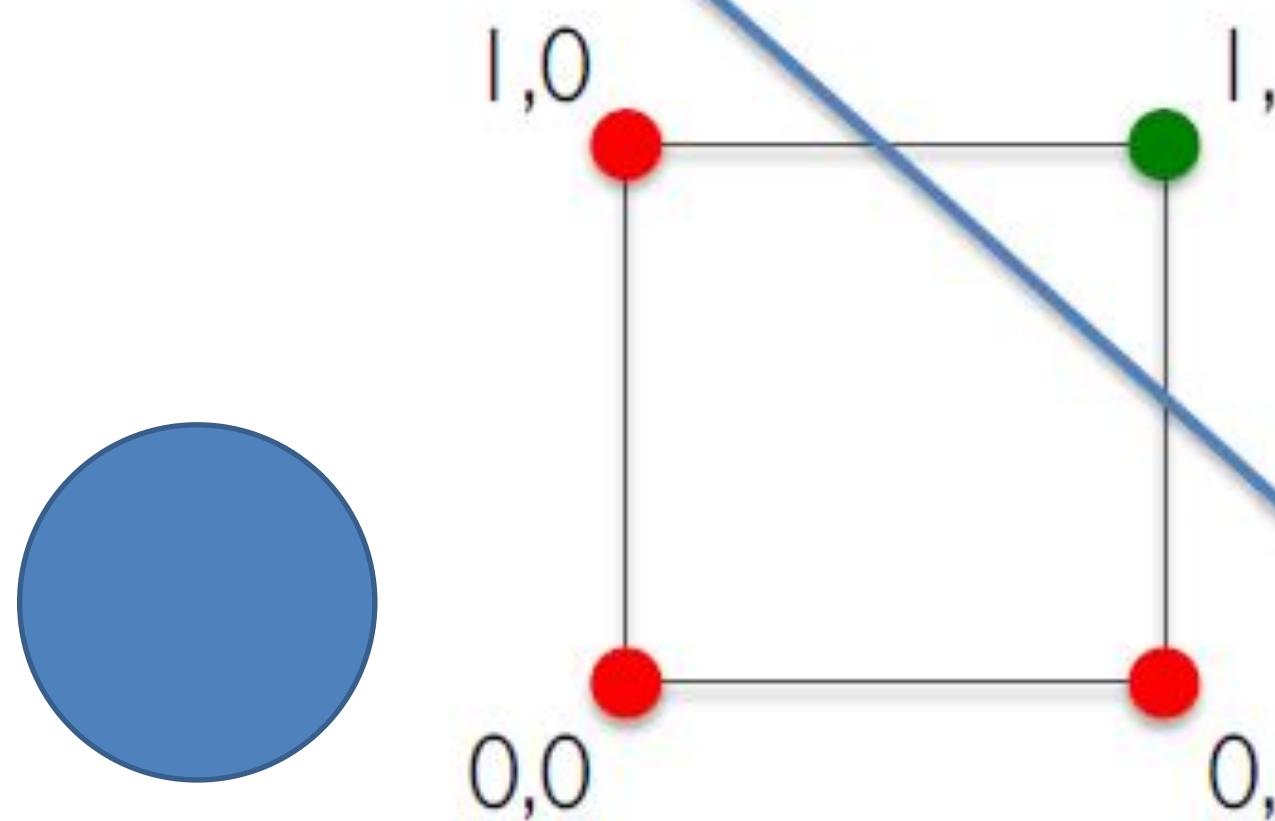




Separabilidad lineal

Los perceptrones simples están limitados por la noción de separabilidad lineal.

- Un concepto linealmente separable es aquel que se puede separar sus clases por medio de un hiperplano.
- La compuerta AND y OR son linealmente separables. La compuerta XOR no es linealmente separable.





Perceptrón: implementación

Existen cuatro formas habituales de programar una red neuronal. Utilizando:

- Programación incremental: es la implementación clásica. Se utilizan vectores para representar entradas y pesos y estos se actualizan aplicando las correspondientes operaciones a cada posición. Es poco eficiente.
- Programación orientada a objetos: las neuronas y conexiones son representadas como objetos con funcionalidad propia que realizan todas las operaciones necesarias para el funcionamiento de la red neuronal. Son eficientes en sistemas con procesamiento paralelo.
- Matrices (vectorización): las entradas pesos y resultados intermedios y finales son representadas como vectores y matrices. Las operaciones se realizan utilizando operaciones matriciales que sacan el máximo partido del entorno de programación y CPU. Su implementación es sencilla.
 - Computación por grafos: se crean grafos que representan las operaciones matemáticas aplicadas de principio a fin. Las variables e incluso las propias operaciones se pueden modificar en tiempo real. Computacionalmente muy eficiente (con vectorización). Se consigue paralelización en CPU y GPU. Librerías: Tensorflow (Keras), Pytorch, etc.



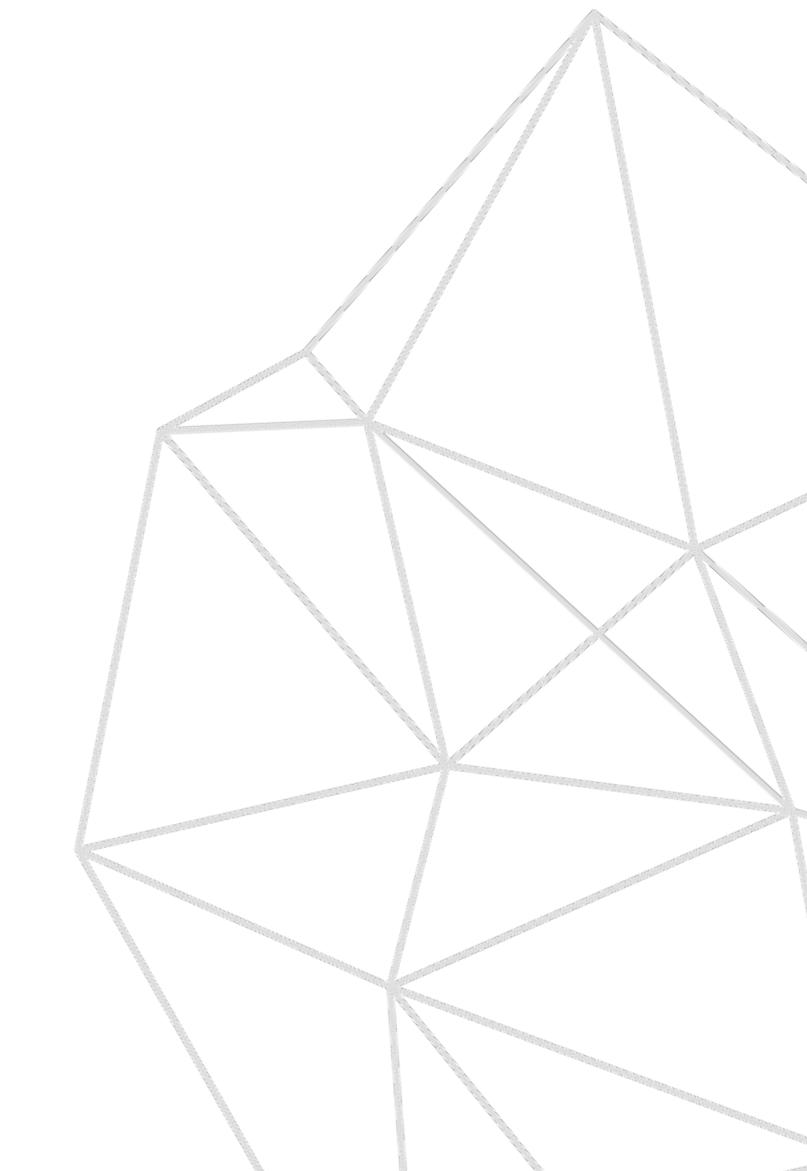
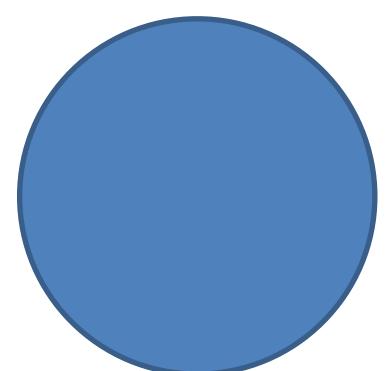
Perceptrón: aprendizaje

Partiendo de un perceptrón con pesos inicializados aleatoriamente, se aplica de forma iterativa la regla de actualización siguiente hasta que la neurona clasifique correctamente:

$$\Delta W = \alpha(y - \hat{y})X$$

Diagrama que muestra la fórmula de actualización de pesos para un perceptrón. Los términos están etiquetados como:

- Constante aprendizaje: α
- Salida estimada: \hat{y}
- Entrada: X
- Salida esperada: y

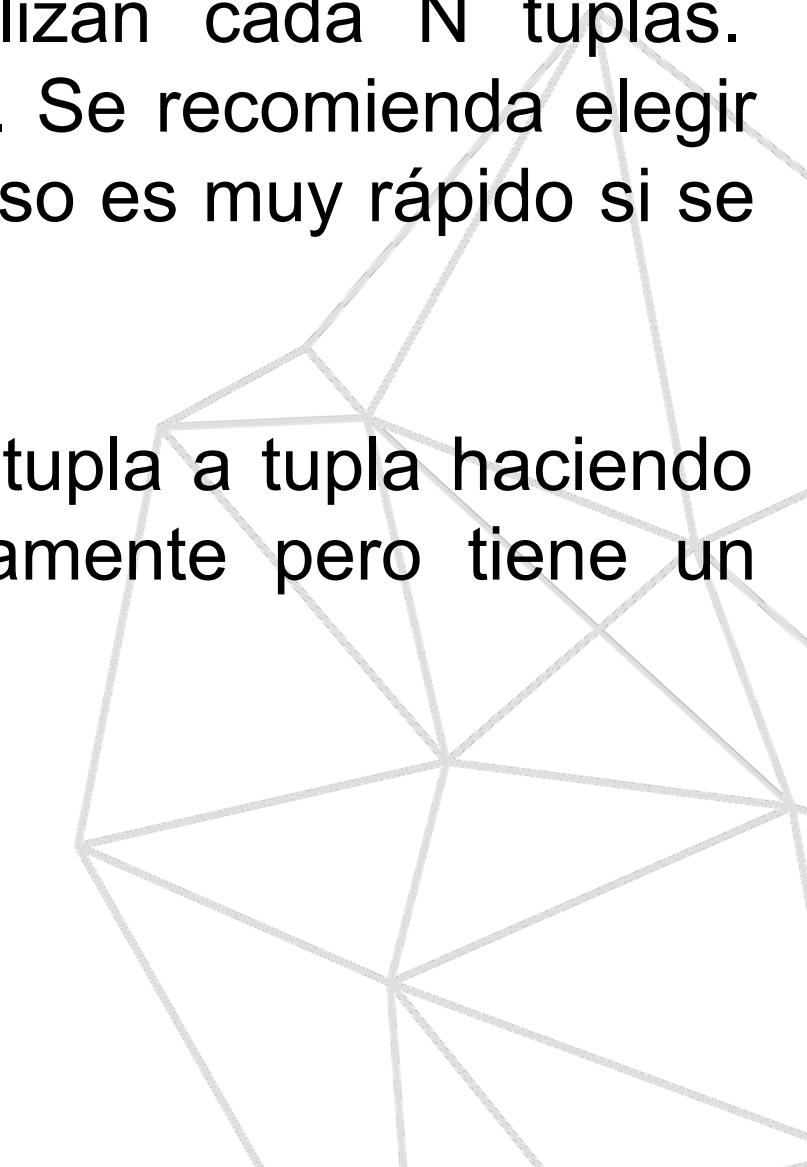
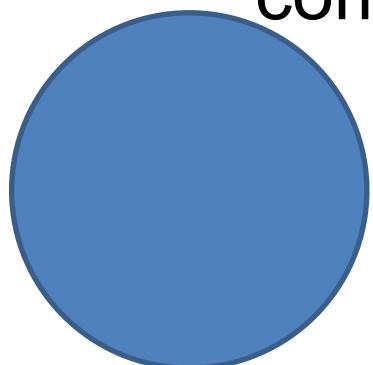




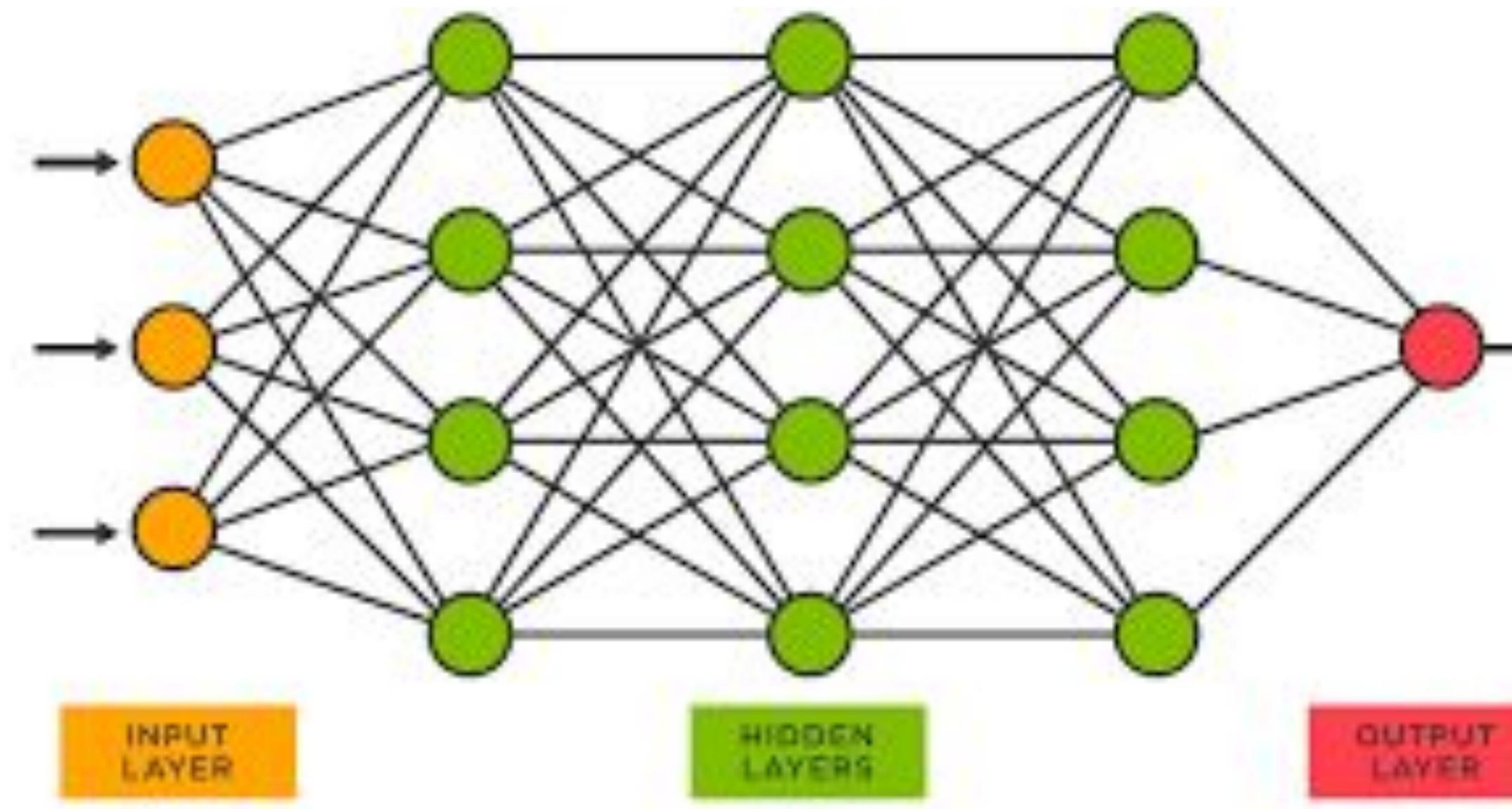
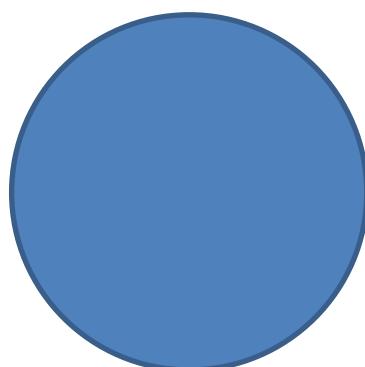
Perceptrón: tipos de aprendizaje

Existen tres tipos de aprendizaje:

- **Batch learning:** los pesos de la neurona (o red neuronal) se actualizan haciendo efectivo lo aprendido al obtener el error una vez recorrido el conjunto completo de datos. Tarda mucho en aplicar lo aprendido, pero el aprendizaje es más estable.
- **Mini-Batch learning:** los pesos de la neurona (o red neuronal) se actualizan cada N tuplas. Normalmente las N se corresponde con potencias de 2: 32, 64, 128, 256 y 512. Se recomienda elegir este valor en función de la memoria de la CPU o GPU. El aprendizaje en este caso es muy rápido si se saca partido de la vectorización.
- **Single Step Learning:** los pesos de la neurona (o red neuronal) se actualizan tupla a tupla haciendo efectivo lo aprendido de forma inmediata. La red aplica lo aprendido rápidamente pero tiene un comportamiento más errático. En términos generales suele ser más lento.



02 Perceptrón multicapa

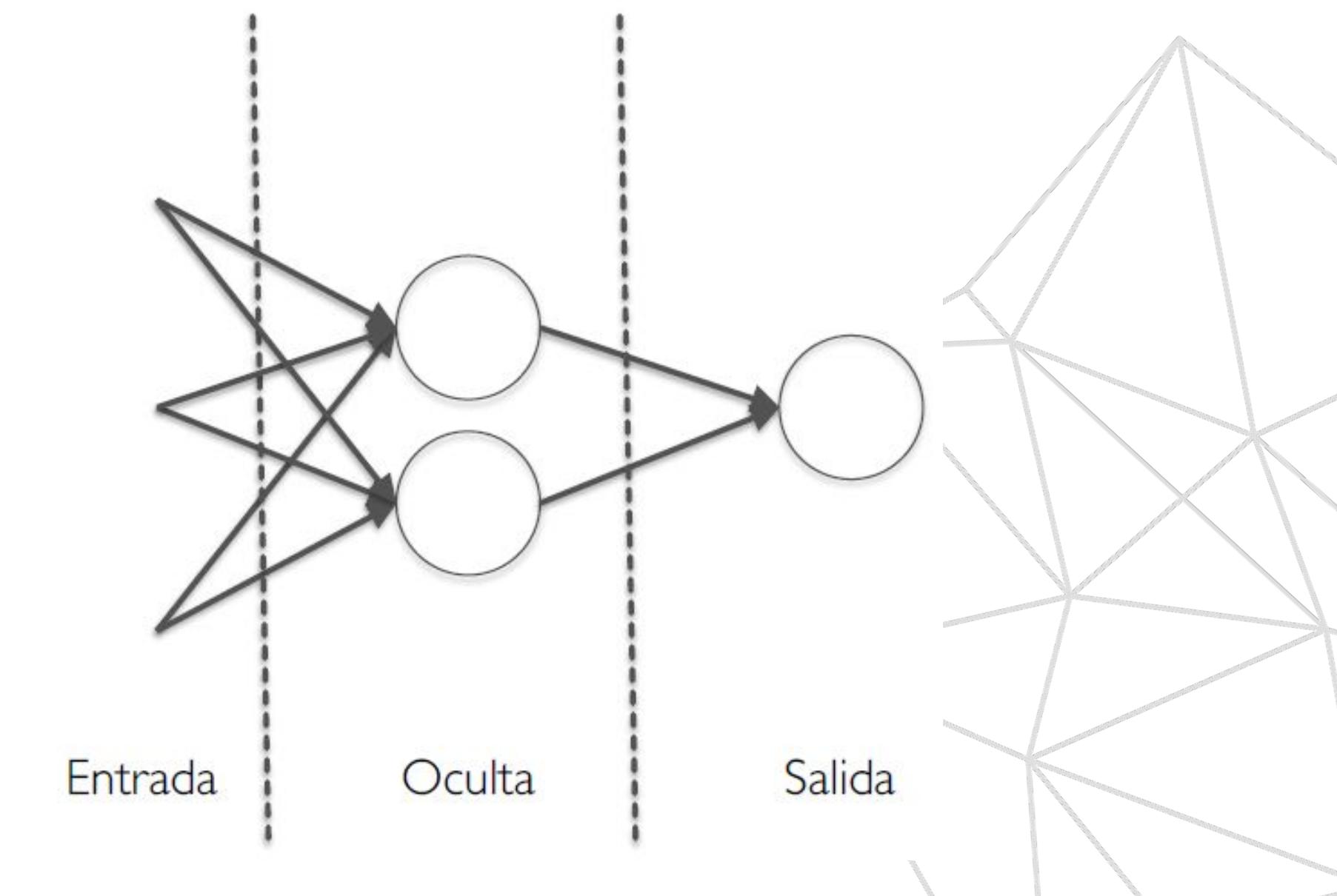
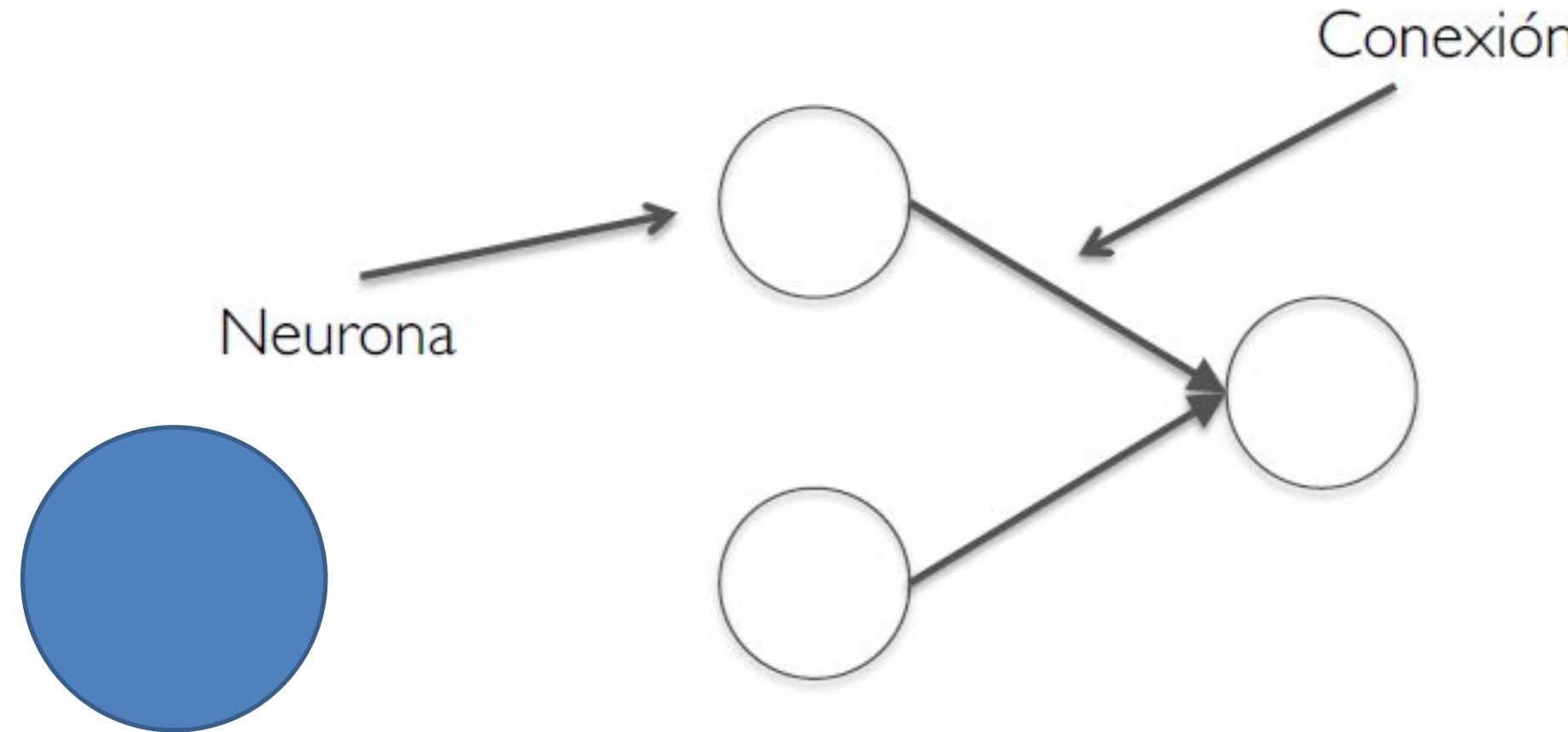




Arquitectura

Las RNA se representan mediante grafos dirigidos que cuentan con nodos (las neuronas) y aristas (sus conexiones).

Las neuronas se agrupan por capas: entrada (reciben datos y los transmiten sin procesarlos), salida (emiten la respuesta de la red a los datos recibidos) y oculta (procesan los datos, pudiendo haber varias).

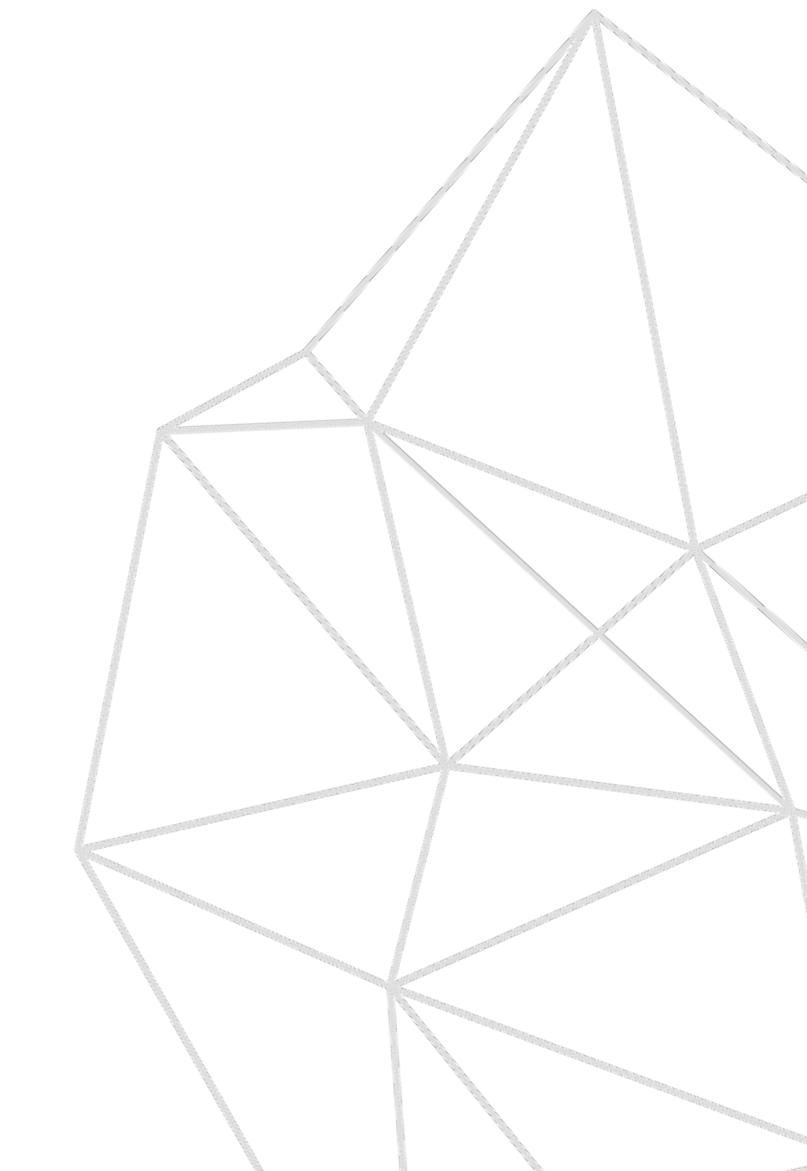
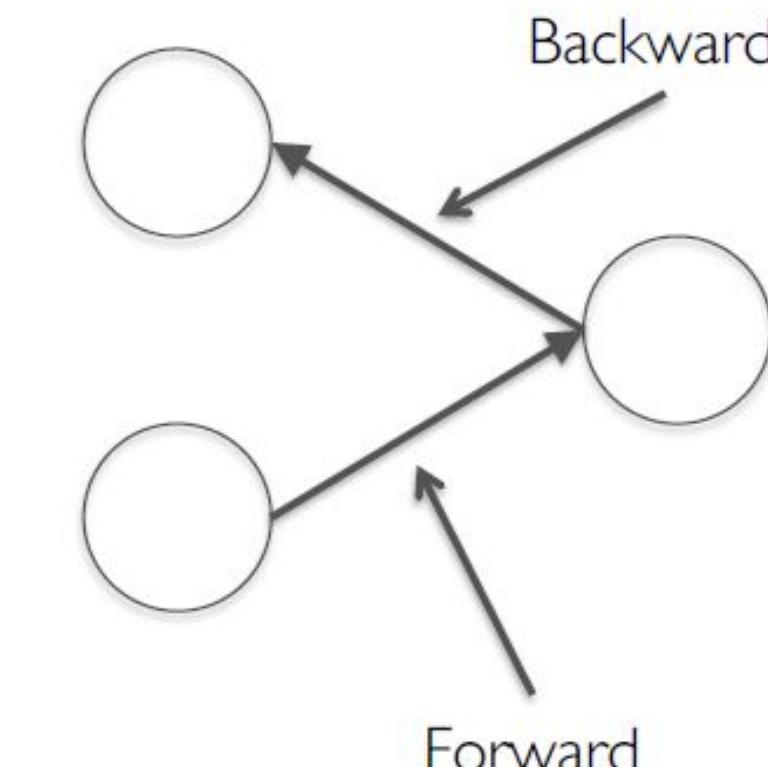
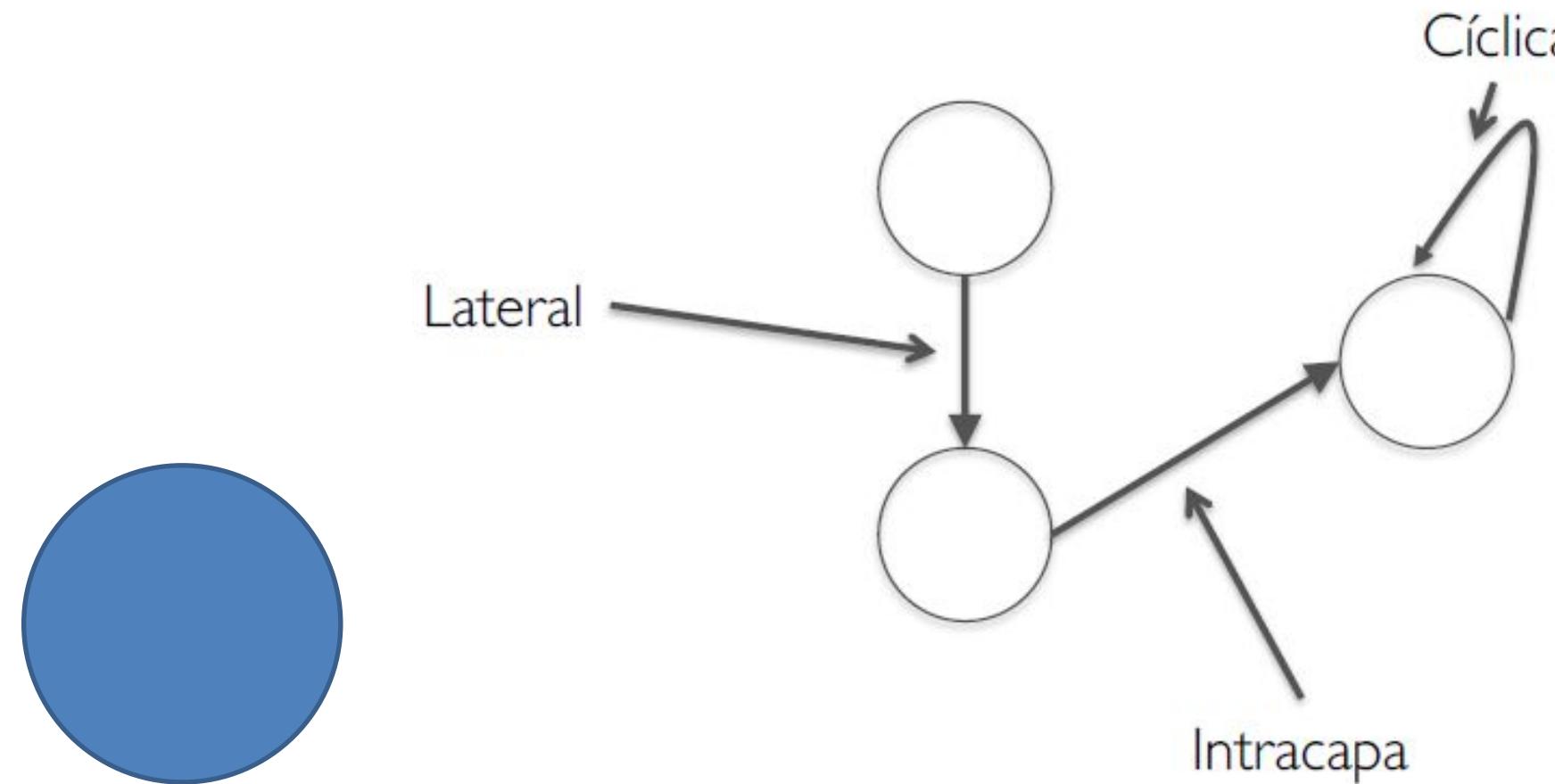




Arquitectura

Las conexiones pueden clasificarse de dos formas distintas:

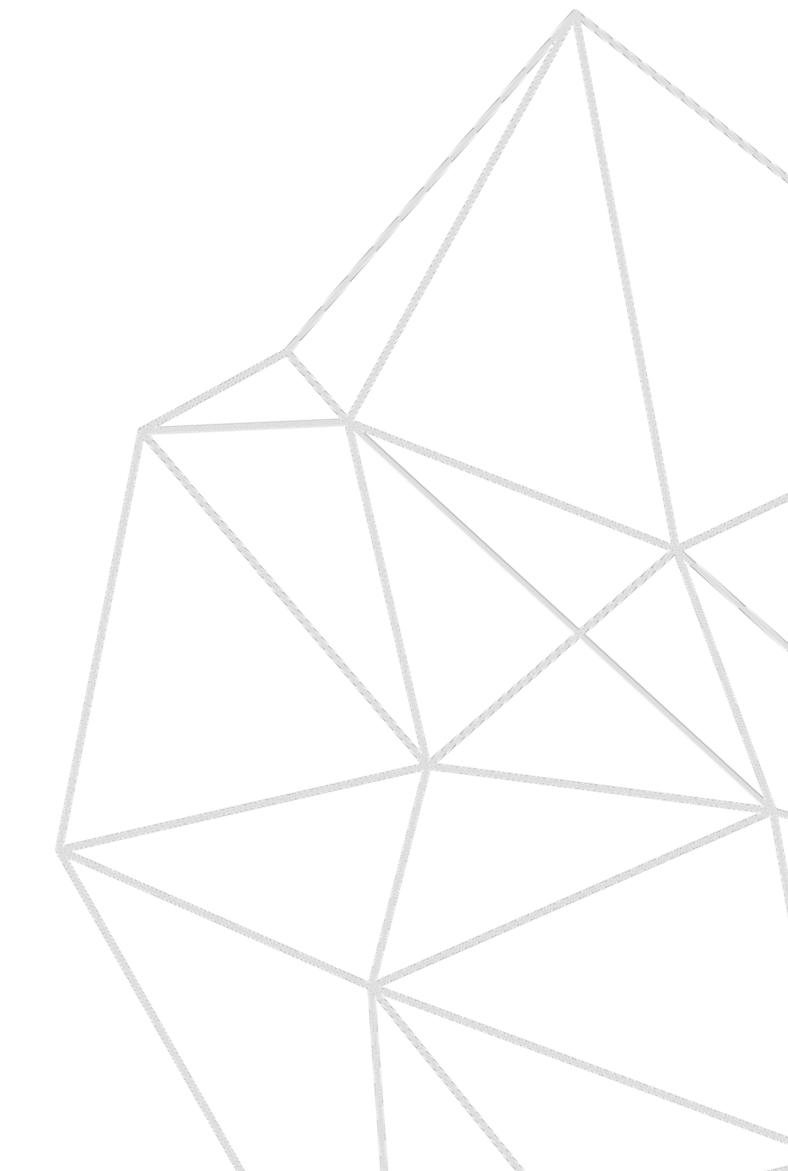
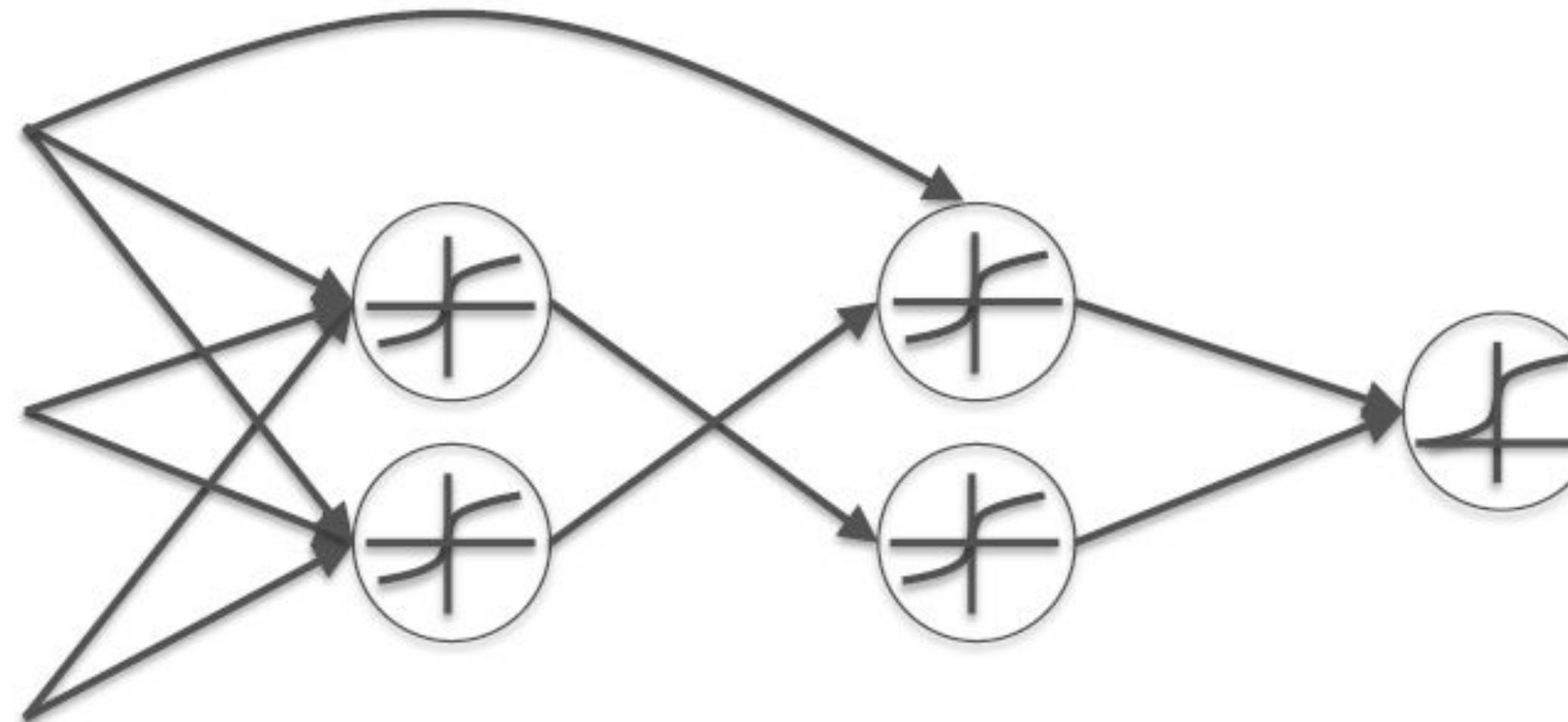
- Por capas: intracapa (conecta neuronas de capas diferentes), laterales (conecta neuronas de una misma capa) y retroalimentación (conecta una neurona consigo misma).
- Por dirección: forward (conecta neuronas en sentido entrada-salida) y backward (conecta neuronas en sentido salida-entrada).





Arquitectura

Para construir un perceptrón multicapa utilizaremos una red neuronal de conexiones de tipo intracapa y sentido forward. Pueden constar de cero o más capas ocultas. Permite entradas continuas. Generalmente usa la función de activación sigmoide y resuelve problemas que no son linealmente separables.

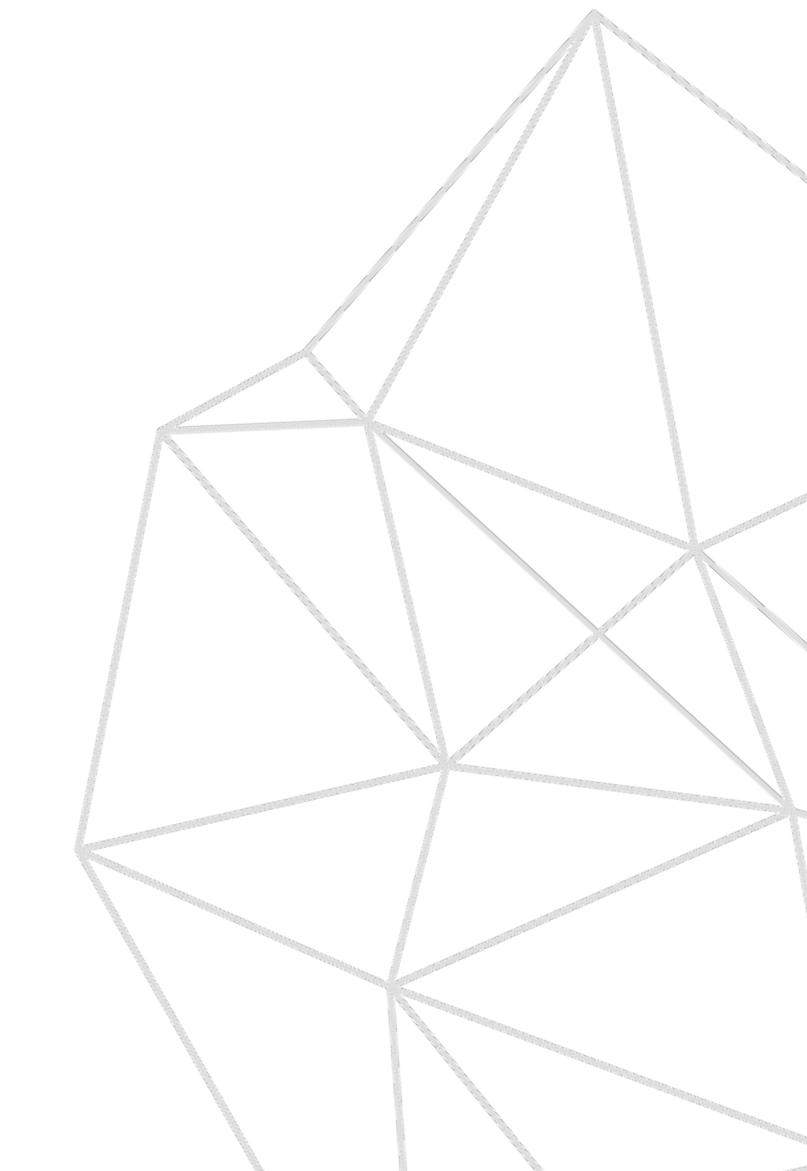
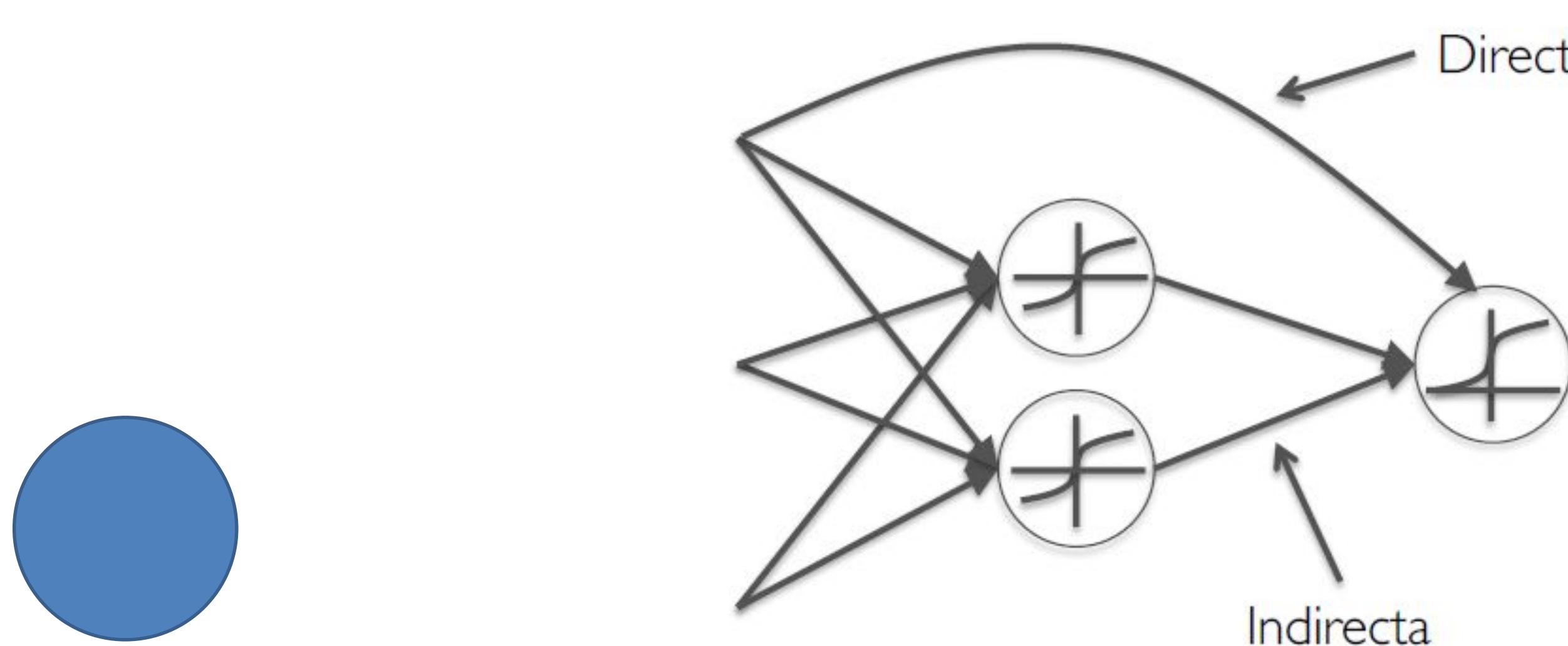




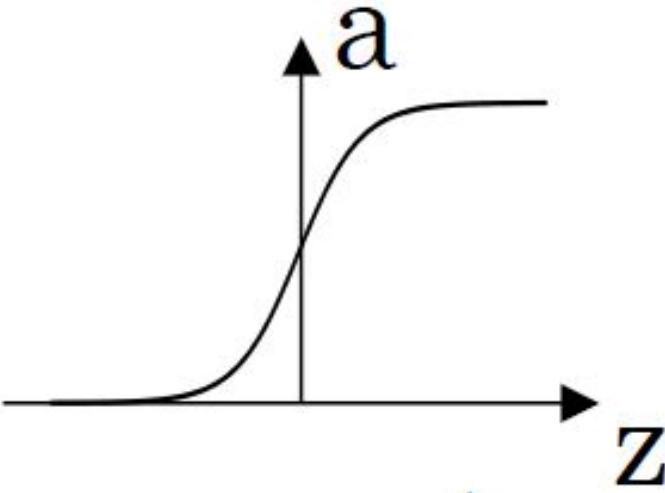
Arquitectura

En el perceptrón multicapa distinguiremos dos tipos de conexiones:

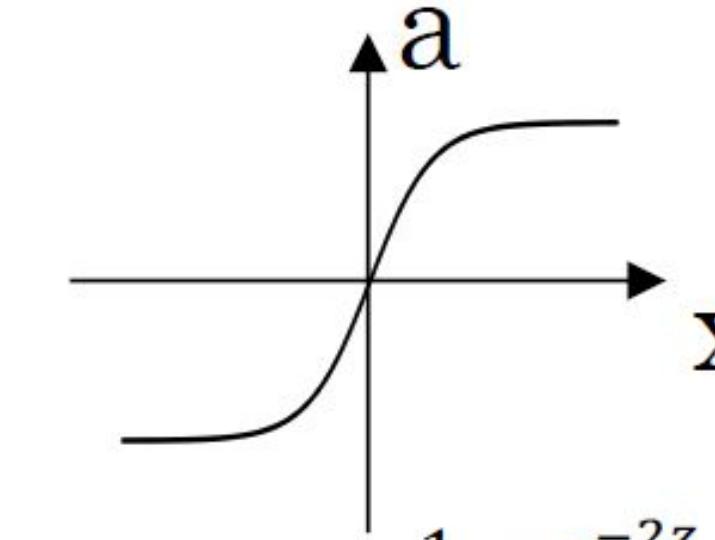
- Directas: conecta neuronas de la capa de entrada con las neuronas de la capa de salida.
- Indirectas: conecta neuronas de una capa con la capa contigua.



Funciones de activación

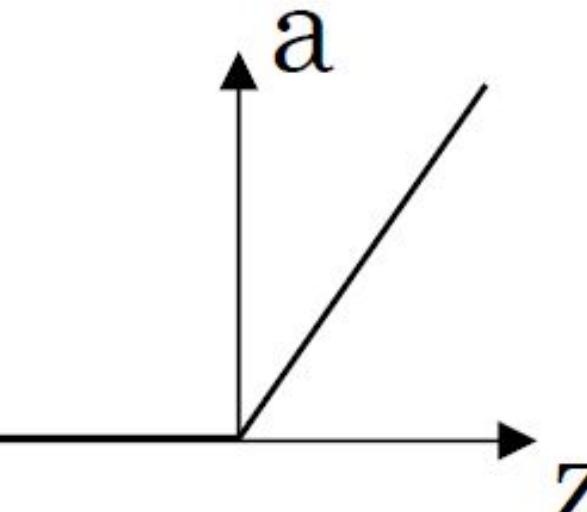


$$\text{sig}(z) = \frac{1}{1 + e^{-z}}$$

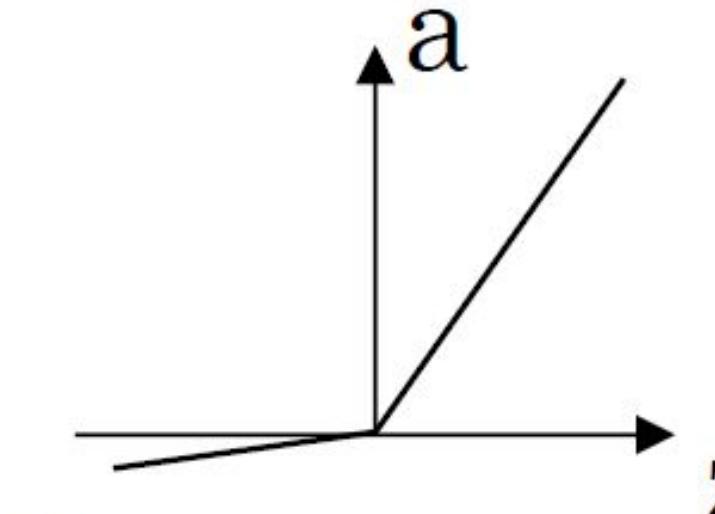


$$\tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$

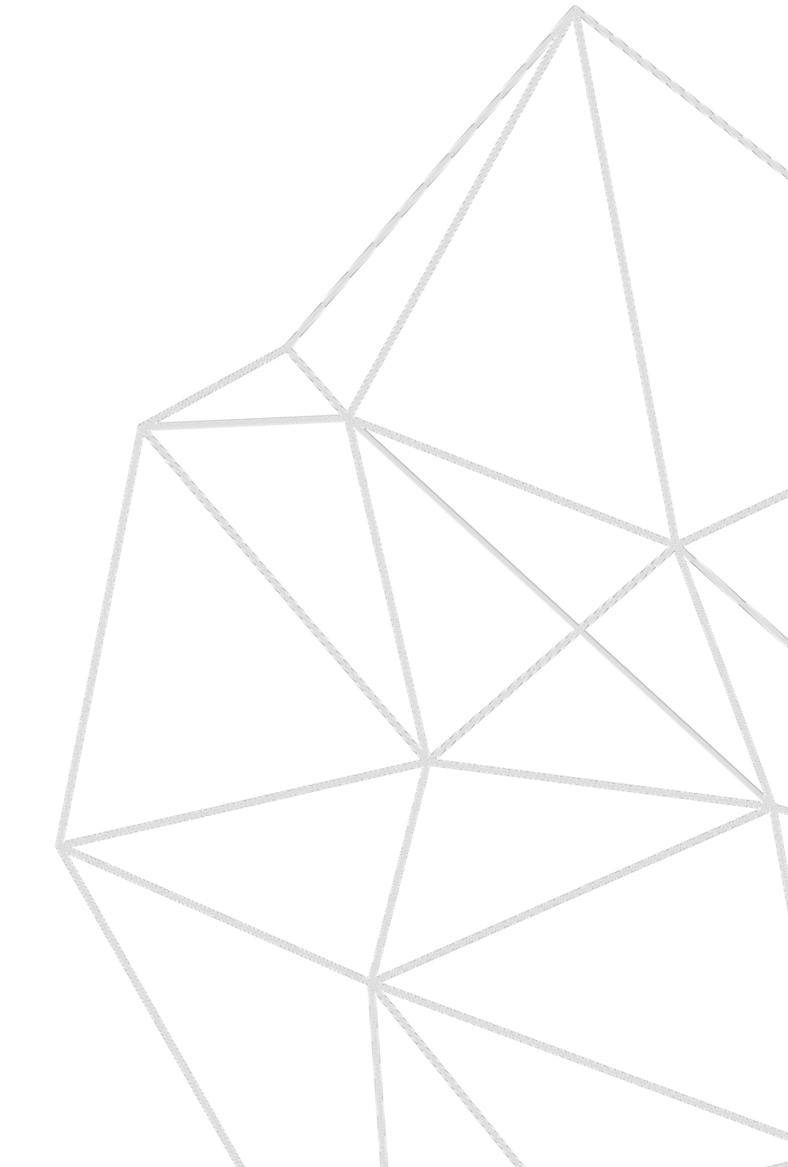
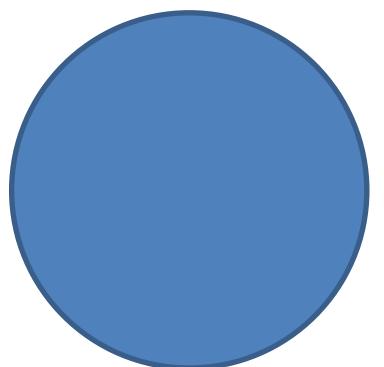
$$\text{Softmax}(Z) = \frac{e^z}{\sum_{j=1}^{n_Z} e^{z_j}}$$



$$\text{ReLU}(z) = \max(0, z)$$



$$\text{Leaky ReLU}(z) = \max(0.1z, z)$$





Inicialización de pesos

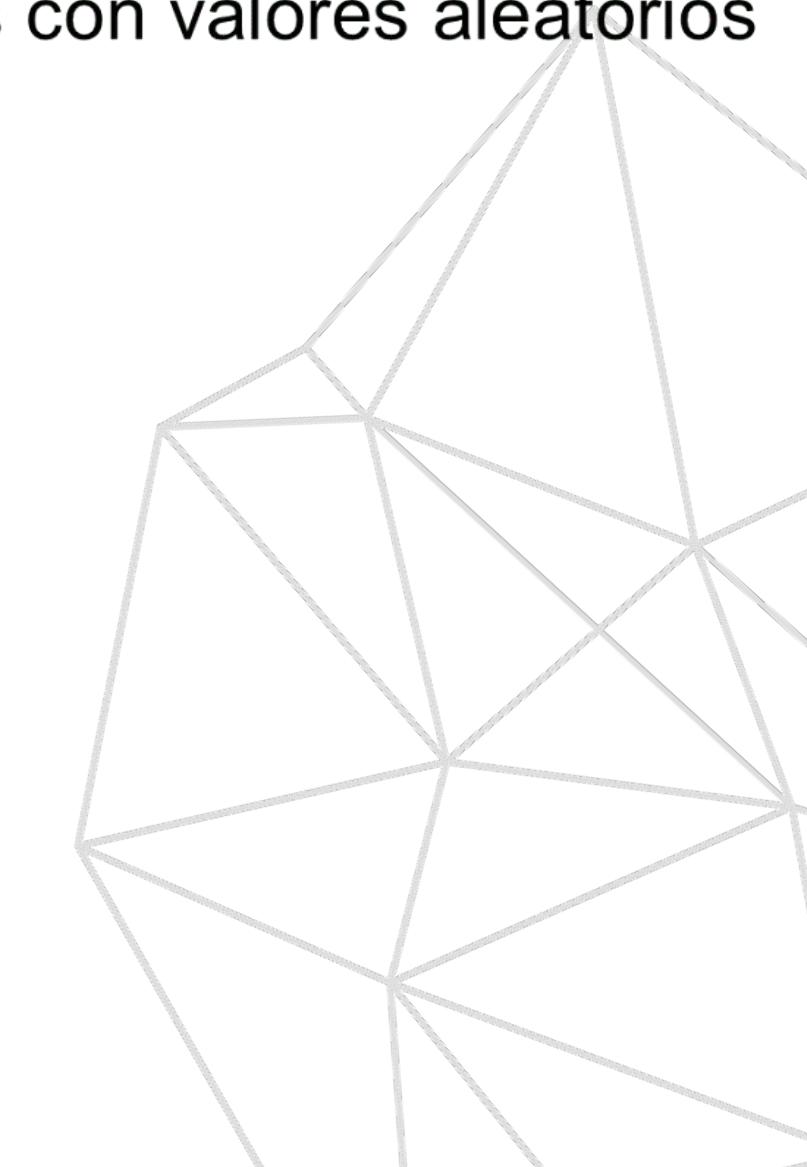
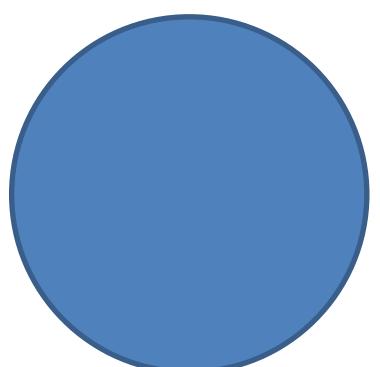
Para capas con función de activación sigmoidal o tangente hiperbólica inicializaremos los pesos con valores aleatorios siguiendo una distribución normal de media cero y:

$$\sigma = \sqrt{\frac{1}{n^{[l-1]}}}$$

Para capas con función de activación RELU o Leaky RELU inicializaremos los pesos con valores aleatorios siguiendo una distribución normal de media cero y:

$$\sigma = \sqrt{\frac{2}{n^{[l-1]}}}$$

Así se reduce la posibilidad de vanishing/exploding de los pesos.

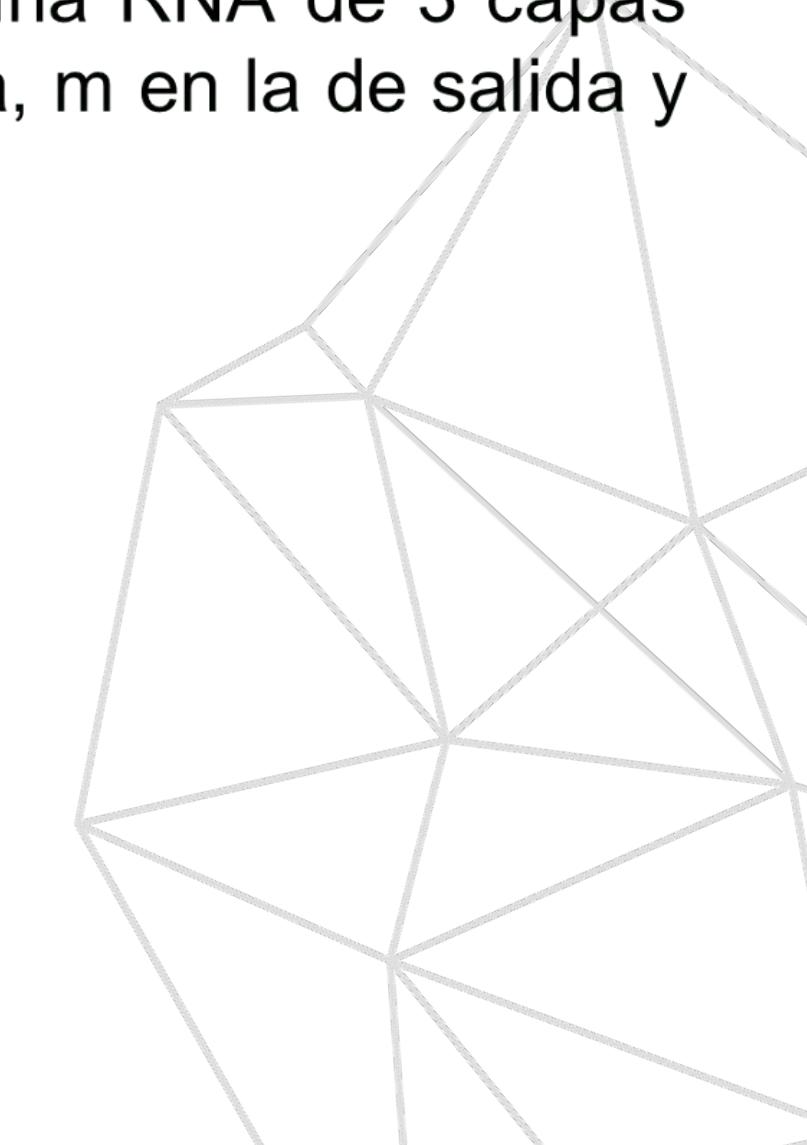
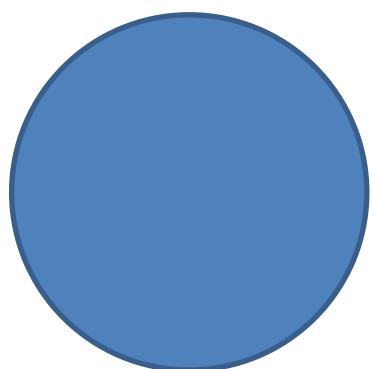




Base matemática

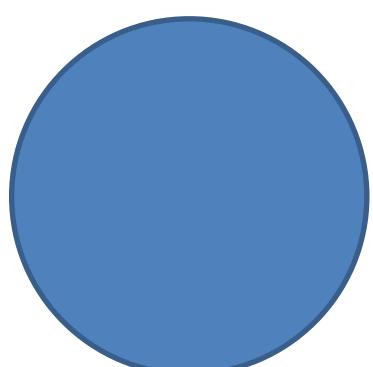
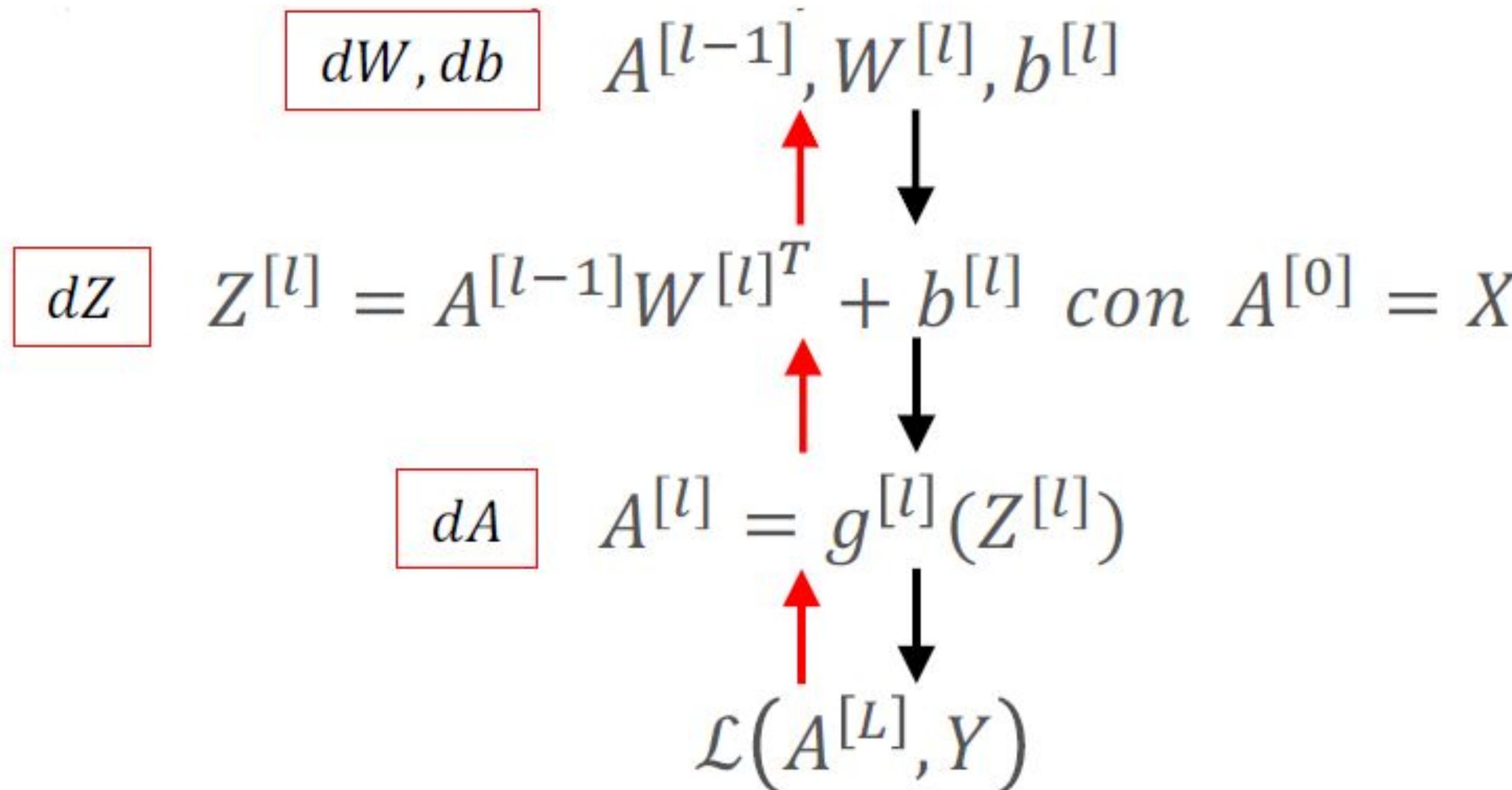
Las redes neuronales se sustentan sobre dos teoremas matemáticos:

- Teorema de aproximación universal: dada cualquier función continua, existe una RNA de 3 capas de propagación hacia delante, con un número finito de neuronas en la capa oculta capaz de aproximar dicha función.
- Teorema de Kolgomorov: dada cualquier función continua $f(0,1)^n \rightarrow \mathbb{R}^m$ existe una RNA de 3 capas de propagación hacia delante, con n elementos de proceso en la capa de entrada, m en la de salida y $(2n + 1)$ en la capa oculta, que implementa dicha función de forma exacta.



Actualización de pesos: Backpropagation

Se obtiene el error respecto a la salida esperada y se propaga hacia atrás para obtener los correspondientes gradientes de aprendizaje de cada neurona:





Actualización de pesos: Backpropagation

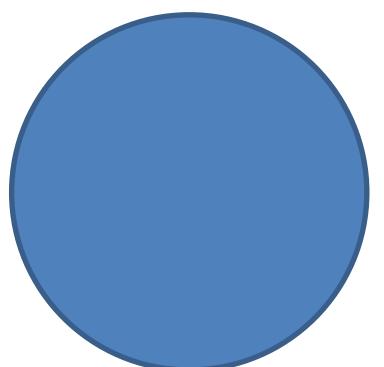
El error se propaga hacia atrás multiplicándolo por los pesos. De este modo solo se propaga el error que corresponde a cada neurona (una fracción del total).

$$dZ^{[L]} = A^{[L]} - Y$$

$$dZ^{[l]} = dZ^{[l+1]} W^{[l+1]} * g^{[l]'}(Z^{[l]}) \quad * \text{(Element wise)}$$

$$dW^{[l]} = \frac{1}{m} dZ^{[l]}{}^T A^{[l-1]}$$

$$db^{[l]} = \frac{1}{m} \sum_{i=1}^{i=m} dZ^{[l](i)}$$



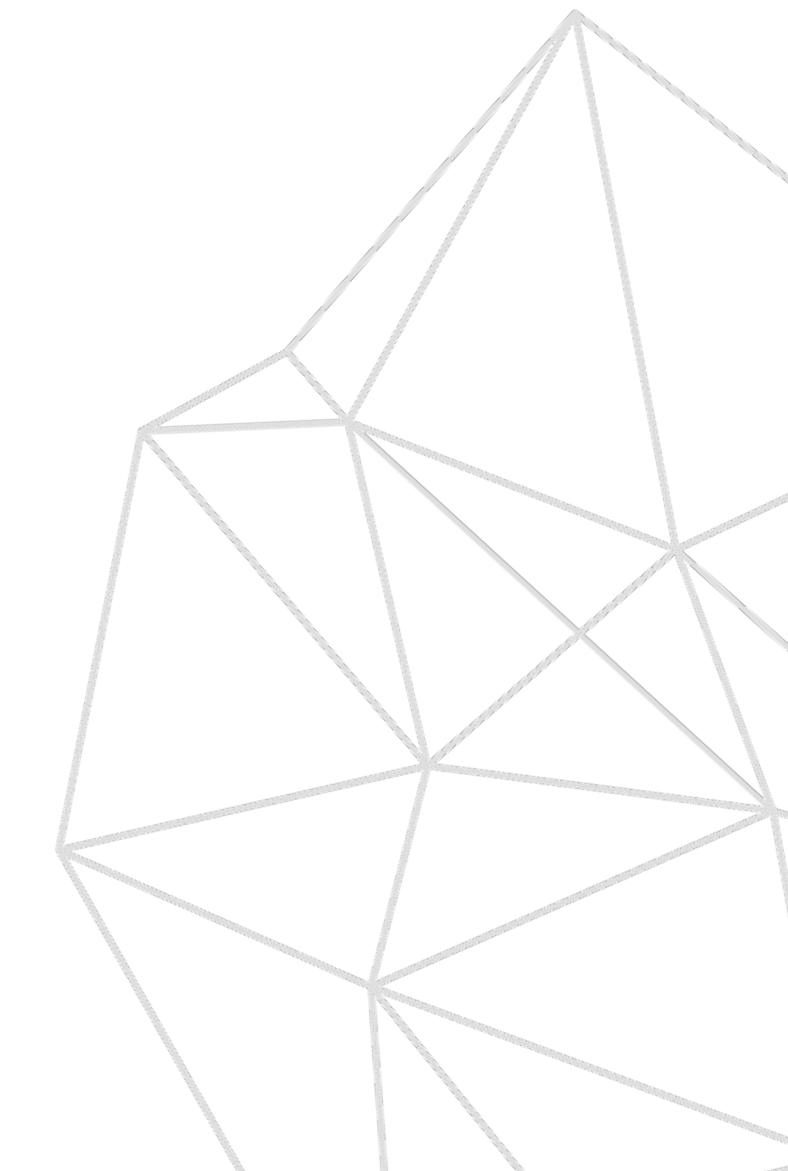
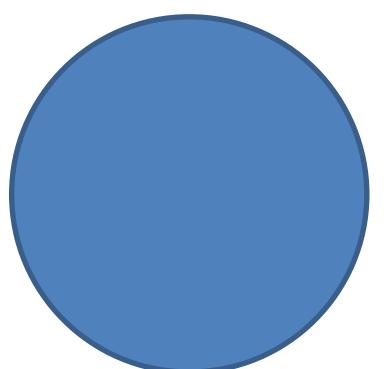


Actualización de pesos: Backpropagation

Por último se actualizan los pesos para cada entrada de cada neurona de acuerdo al incremento obtenido.

$$\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \alpha d\mathbf{W}^{[l]}$$

$$\mathbf{b}^{[l]} := \mathbf{b}^{[l]} - \alpha d\mathbf{b}^{[l]}$$





Backpropagation: Algoritmo

Backpropagation(datos, α)

 Iniciar los pesos $W^{[l]}$, $b^{[l]}$

 Repetir mientras no se cumpla condición de parada

 Para cada batch \in Conjunto de datos:

 //calcular la salida de la red neuronal $A^{[L]}$ (feedforward)

 //calcular el gradiente de error de la capa de salida

$$dZ^{[L]} = A^{[L]} - Y$$

 //calcular $dW^{[l]}$, $db^{[l]}$ para los pesos entre la capa oculta y de salida

$$dW^{[l]} = \frac{1}{m} dZ^{[l]}{}^T A^{[l-1]}$$

$$db^{[l]} = \frac{1}{m} \sum_{i=1}^{i=m} dz^{[l]}(i)$$

 Para cada capa oculta

 //calcular el gradiente de error de la capa oculta

$$dZ^{[l]} = dZ^{[l+1]} W^{[l+1]} * g^{[l]}'(Z^{[l]})$$

 //calcular $dW^{[l]}$, $db^{[l]}$ para los pesos de todas conexiones

 // entre las capa oculta y capa de entrada y ocultas

$$dW^{[l]} = \frac{1}{m} dZ^{[l]}{}^T A^{[l-1]}$$

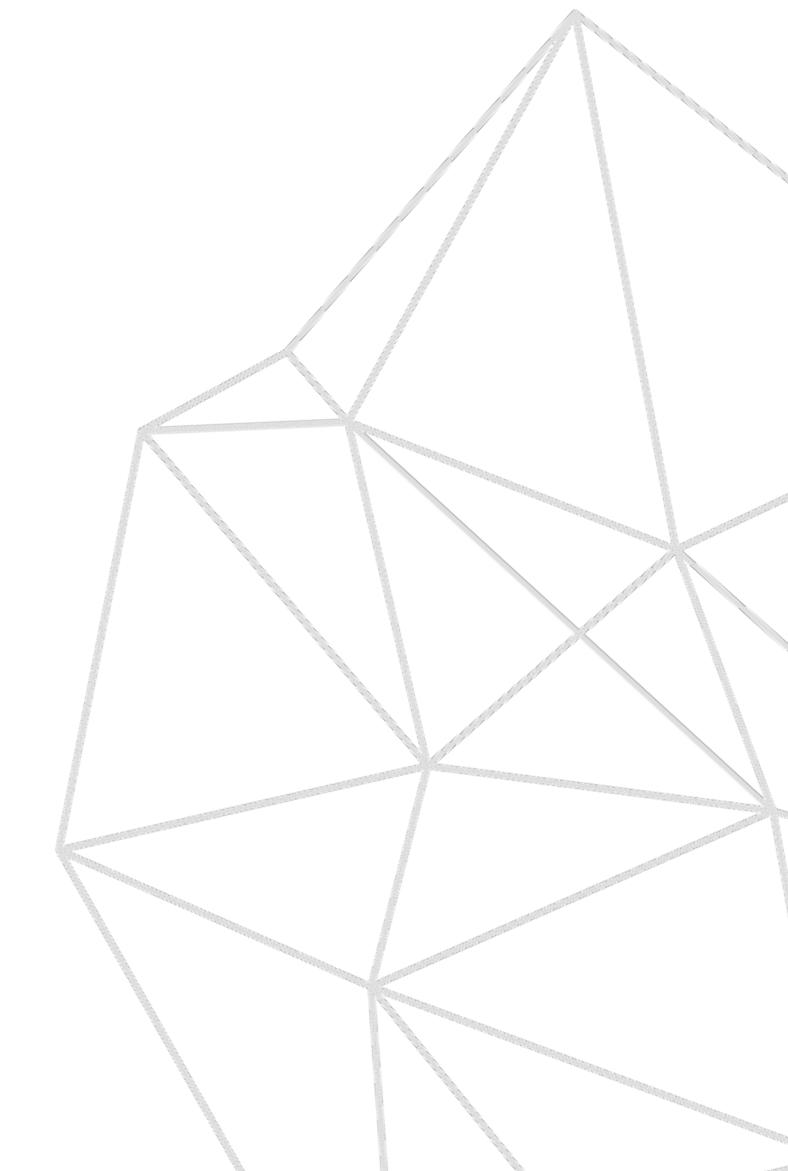
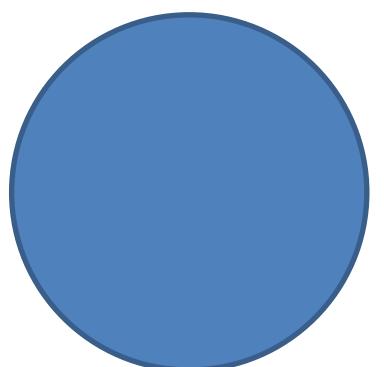
$$db^{[l]} = \frac{1}{m} \sum_{i=1}^{i=m} dz^{[l]}(i)$$

 Para cada capa

 //actualizar los pesos

$$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$$

$$b^{[l]} := b^{[l]} - \alpha db^{[l]}$$



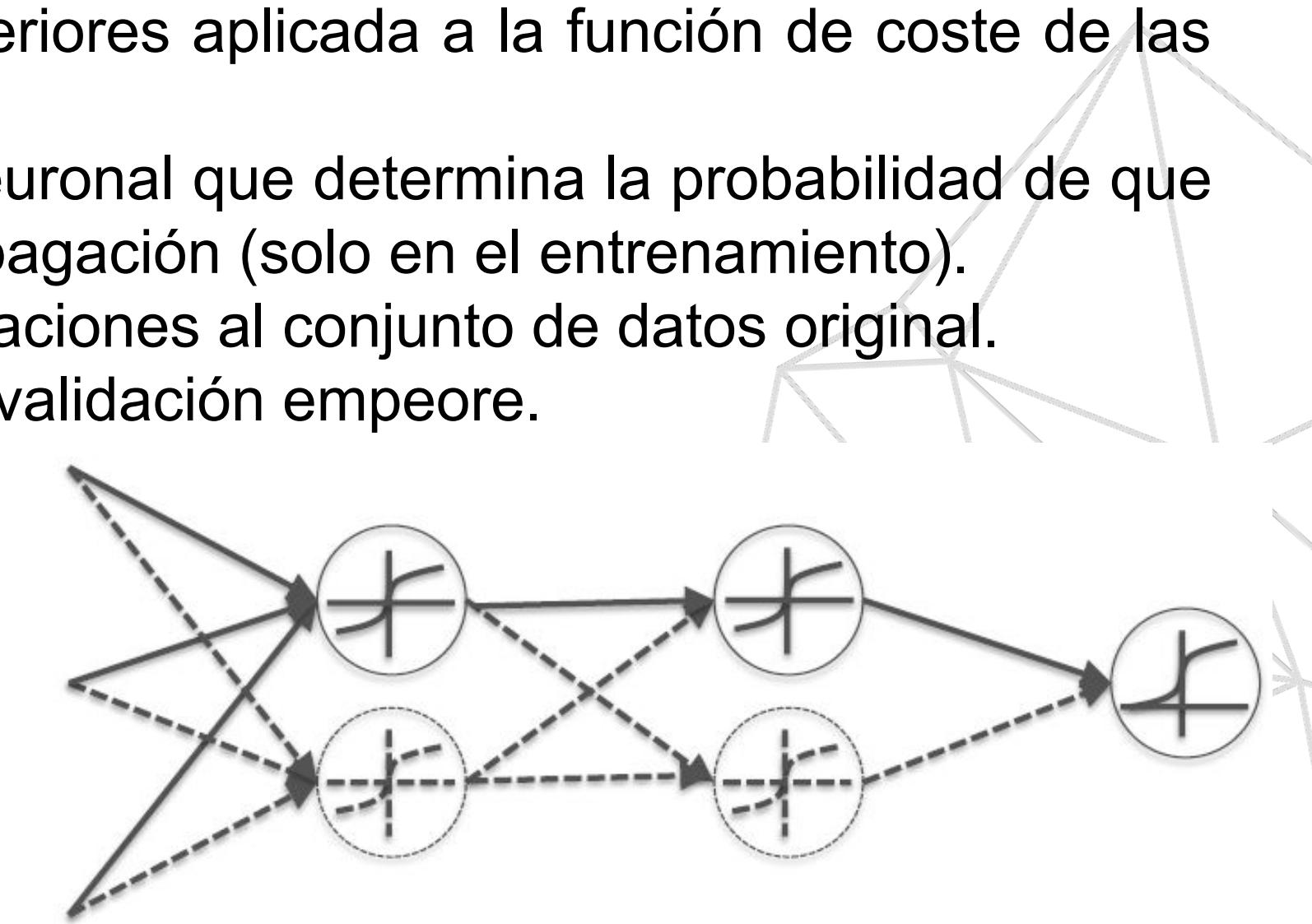
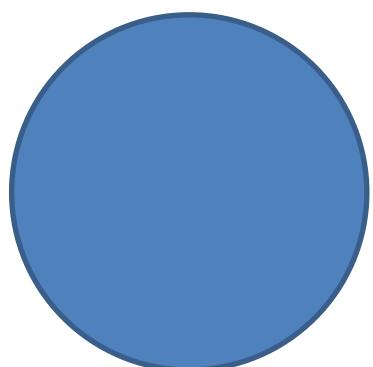


Descenso del gradiente: regularización

El objetivo de la regularización es reducir la varianza del sistema. Al igual que vimos en los temas anteriores, es posible ajustar la actualización de los pesos de las neuronas utilizando esta técnica.

Gracias a la regularización es posible reducir el valor de los pesos, siendo posible llegar a anular neuronas. Existen diversos métodos:

- L2: análoga a la regularización L2 vista en los temas anteriores aplicada a la función de coste de las neuronas.
- Dropout: se asigna una probabilidad por capa de la red neuronal que determina la probabilidad de que cada neurona de esa capa funcione o no durante una propagación (solo en el entrenamiento).
- Data augmentation: crear nuevos datos a partir de modificaciones al conjunto de datos original.
- Early Stopping: parar el entrenamiento cuando el error de validación empeore.
- Batch normalization: normalizar las entradas de las capas



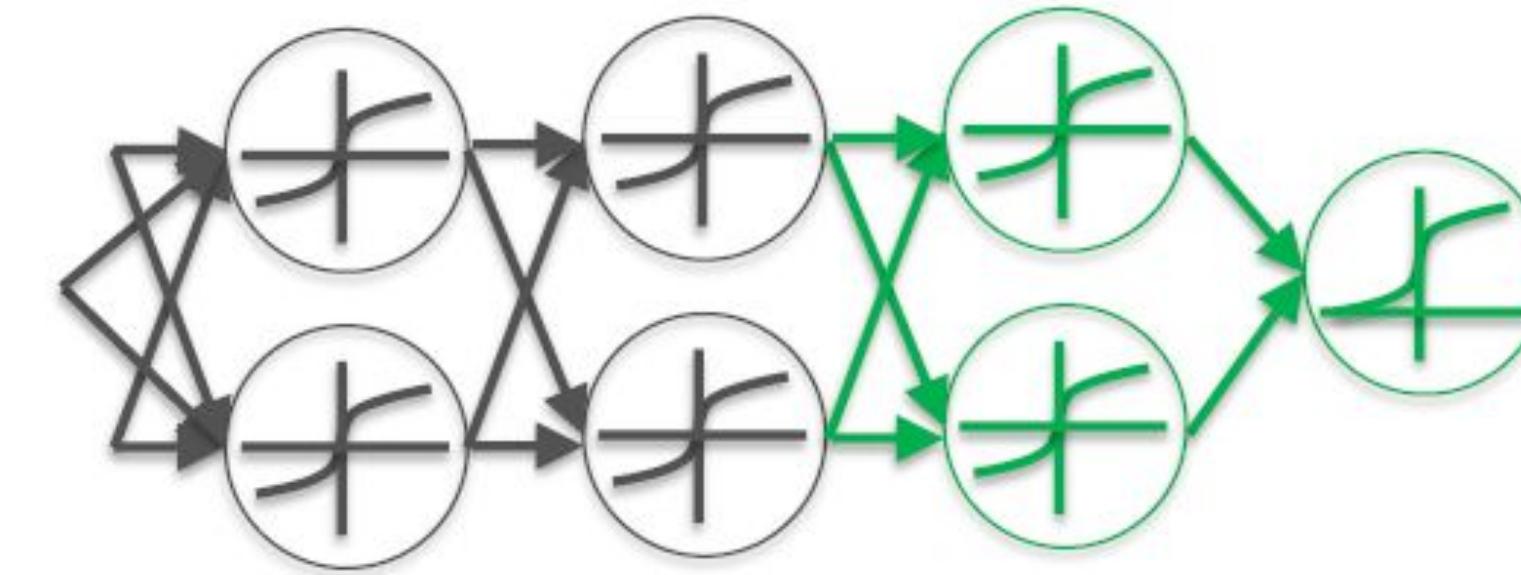
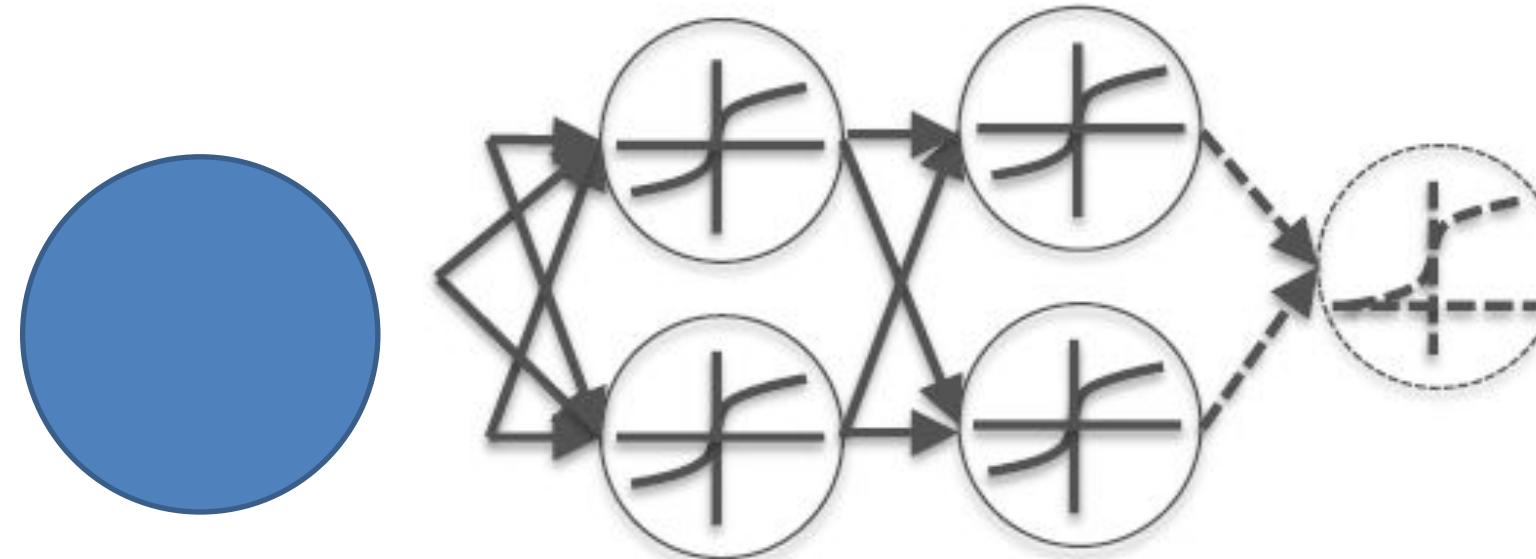


Transfer learning

Las redes neuronales artificiales requieren muchos datos de entrenamiento. En caso de no contar con ellos, una de las posibles soluciones que podemos usar es la conocida como transfer learning.

Hay ciertas redes neuronales ya construidas que funcionan razonablemente bien para problemas similares al nuestro (con el mismo dominio, mismas entradas, etc.). La idea del transfer learning es utilizar una red neuronal entrenada para otro problema similar reemplazando las últimas capas.

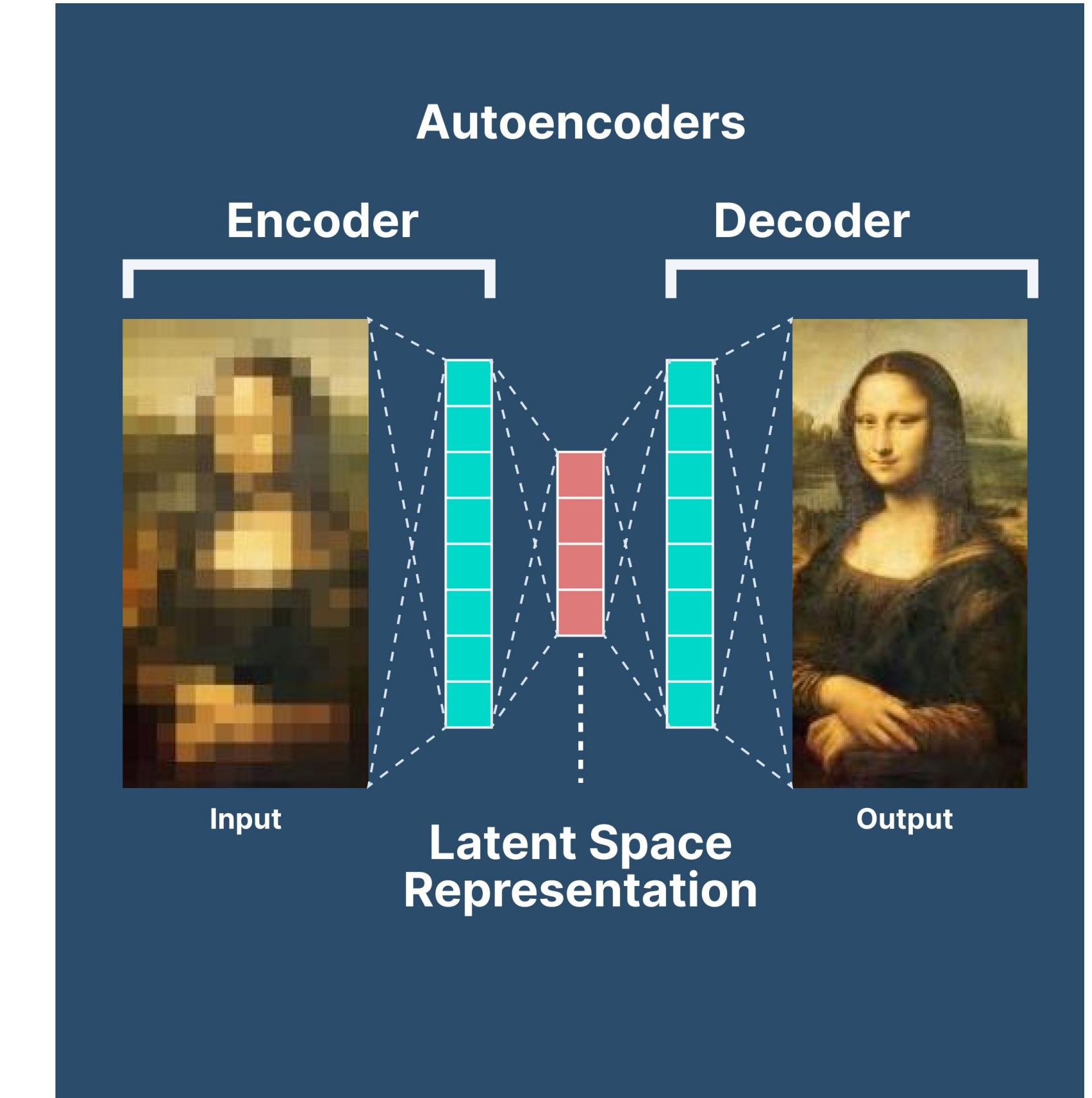
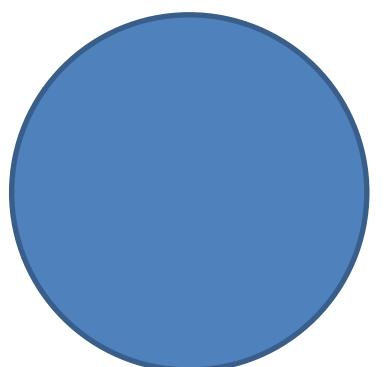
Cuando se tienen muchos datos para el problema a resolver entrenaremos la red completa, mientras que si contamos con pocos solo entrenaremos la parte de la red reemplazada (fine-tunning).





03

Autoencoders





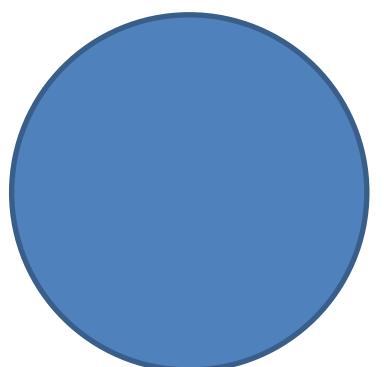
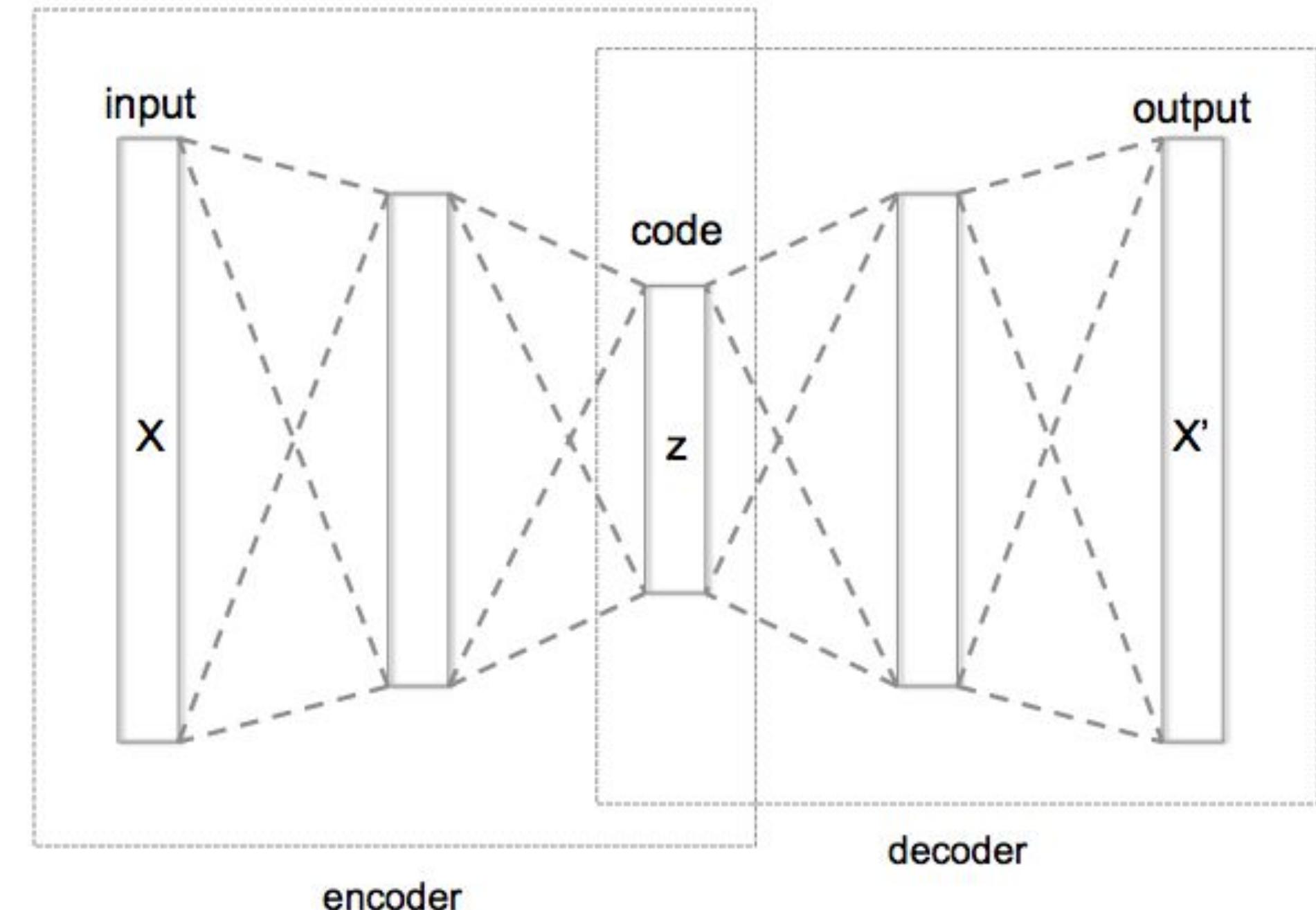
Introducción

Un autoencoder es una red de neuronas simétrica formada por:

- **Encoder:** Red neuronal de reconocimiento.
- **Decoder:** Red neuronal de generación.

La capa central, **vector latente** (code), es una representación del conjunto de datos de entrada de dimensión menor.

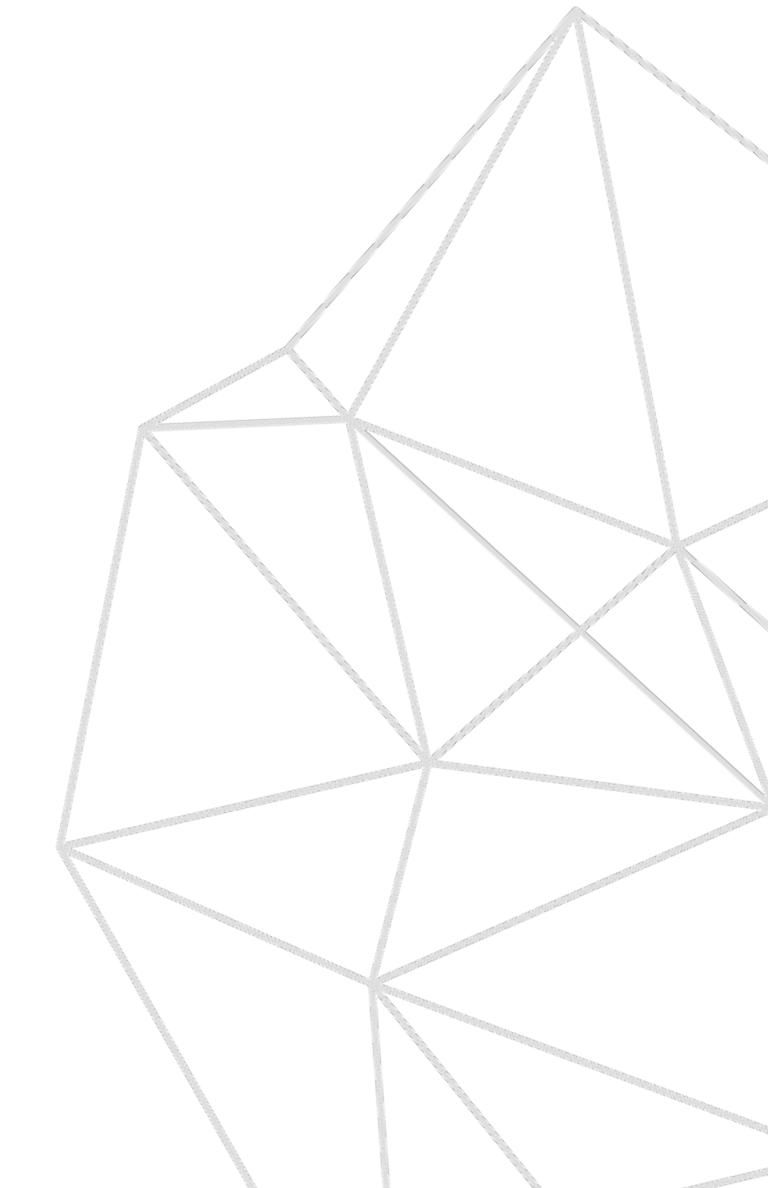
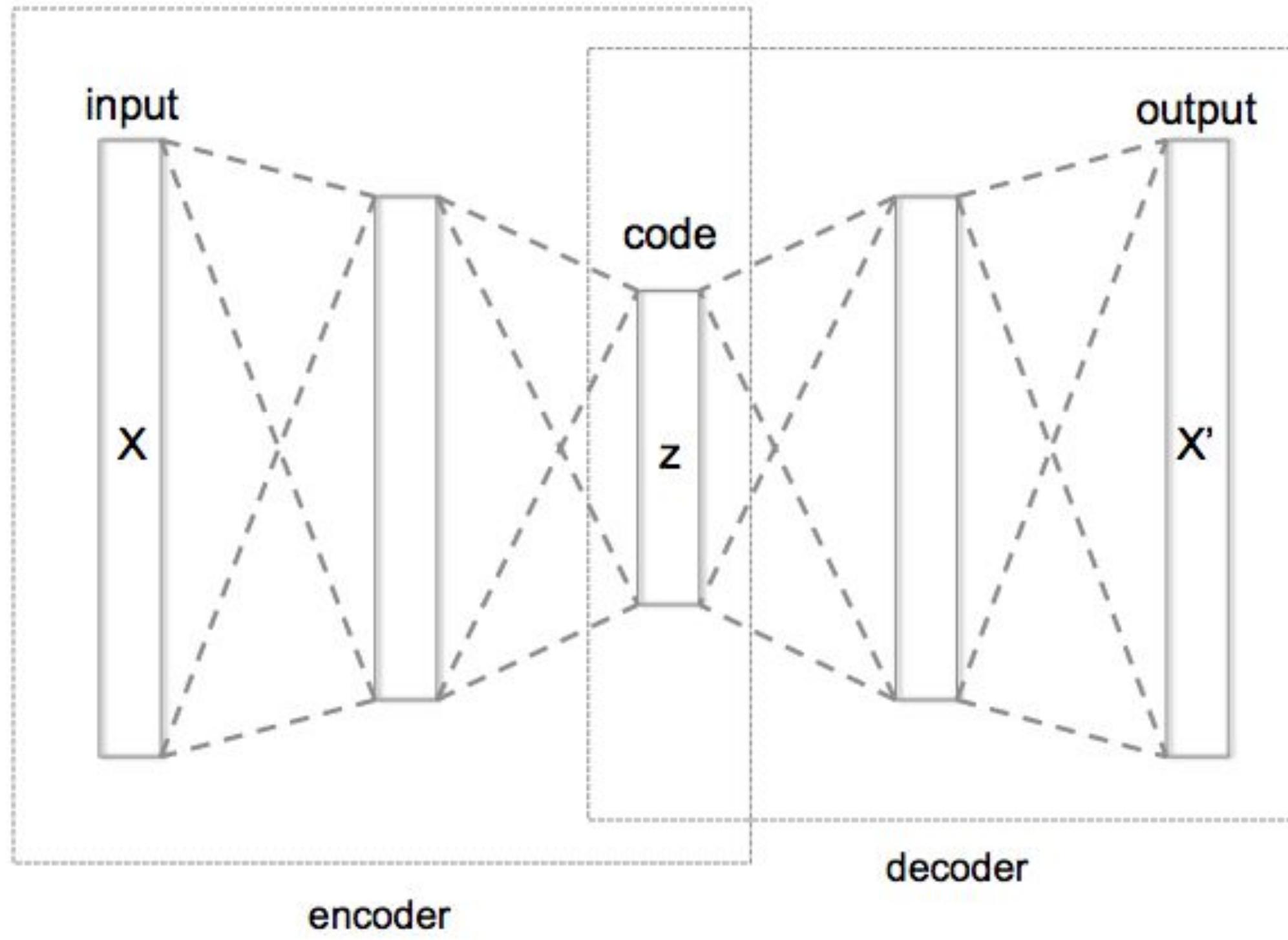
Esto nos permite representar los datos en un espacio de dimensión menor denominado **espacio latente**.





Introducción

El objetivo de un autoencoder es devolver un vector X' , idéntico/similar al conjunto de datos de entrada X . Nuestra función de perdida vendrá definida por la diferencia entre X y X' .

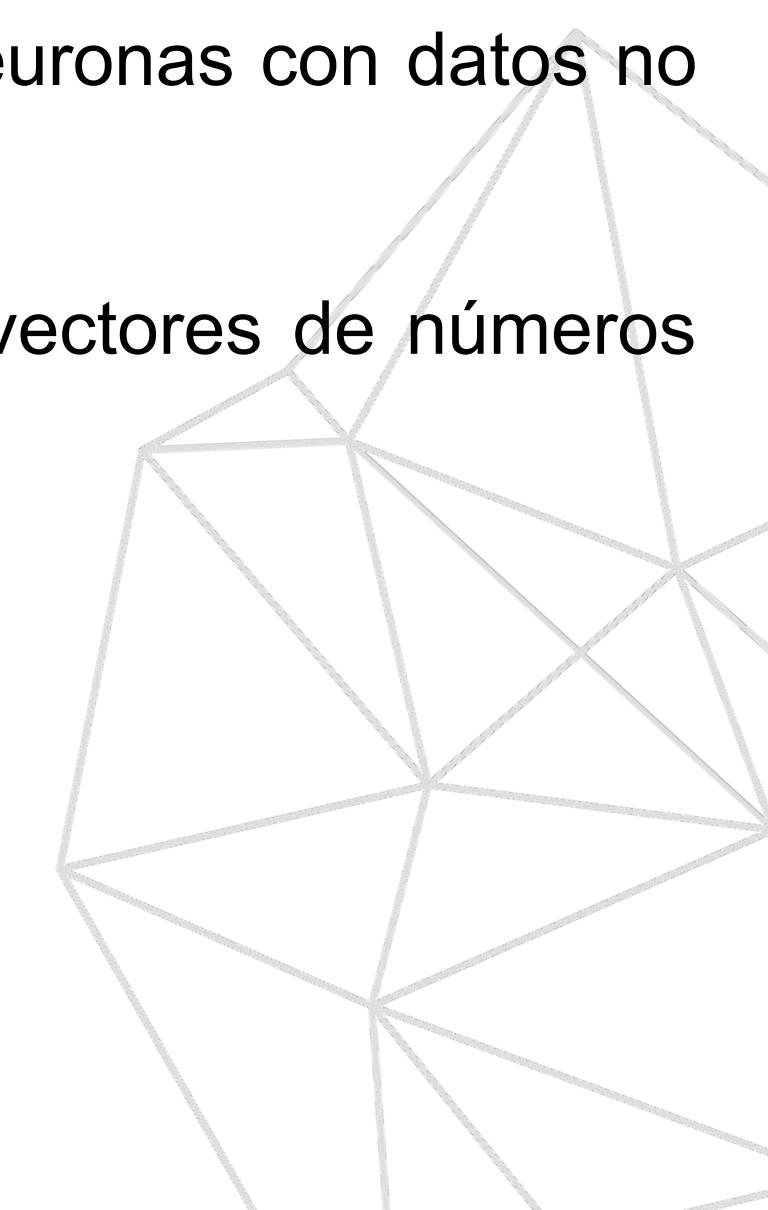
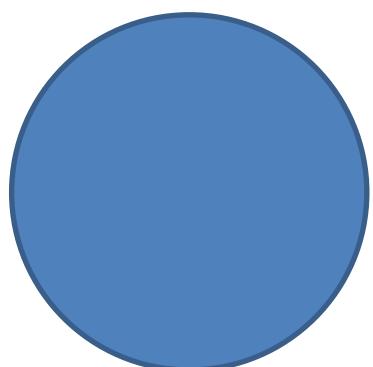




Introducción

Los autoencoders son útiles para:

1. Reducción de dimensionalidad (feature extraction), por ejemplo: Principal Component Analysis (PCA)
2. Modelos generativos, por ejemplo: Variational Autoencoders
3. Preentrenamiento no supervisado, por ejemplo: Preentrenamiento de redes de neuronas con datos no supervisados cuando no se tiene suficientes datos de entrenamiento con etiqueta
4. Word Embeddings, por ejemplo: Representación del lenguaje natural mediante vectores de números reales

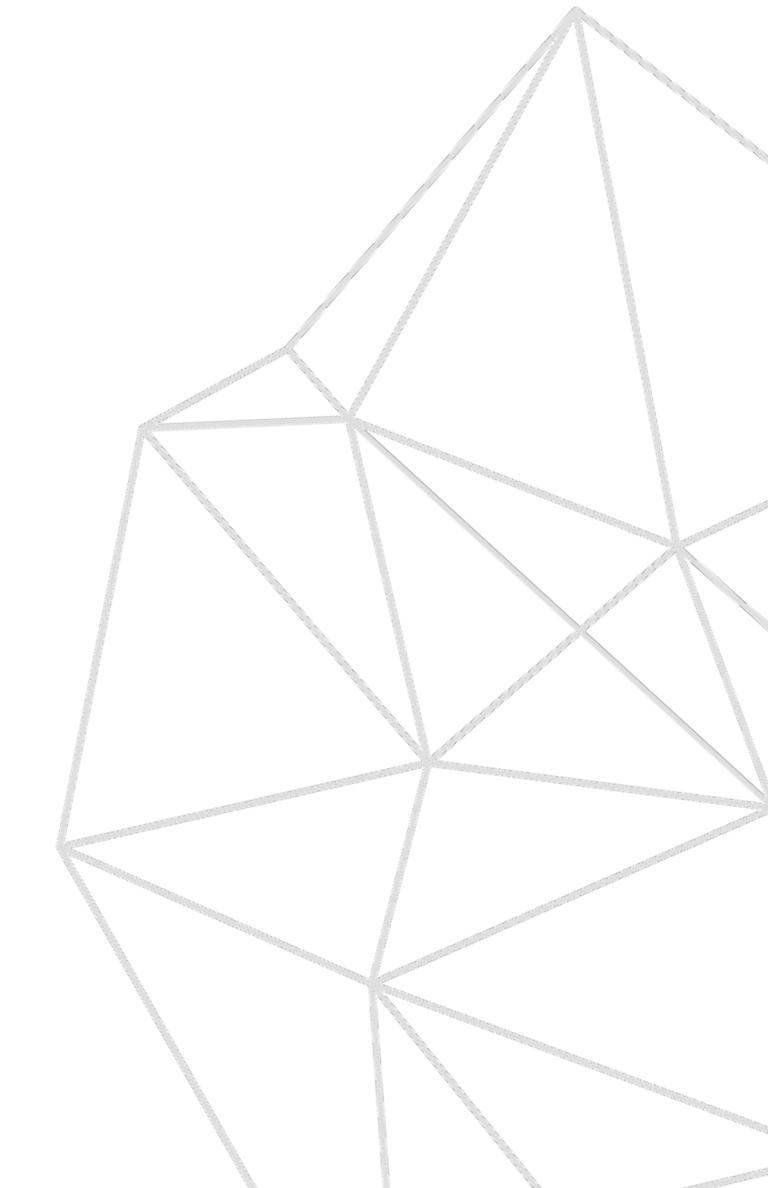
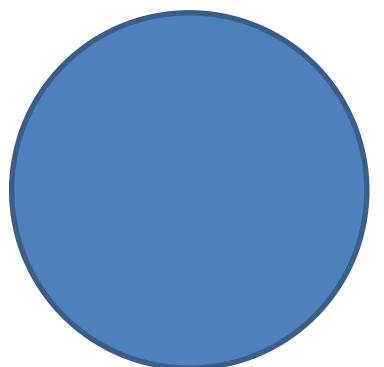
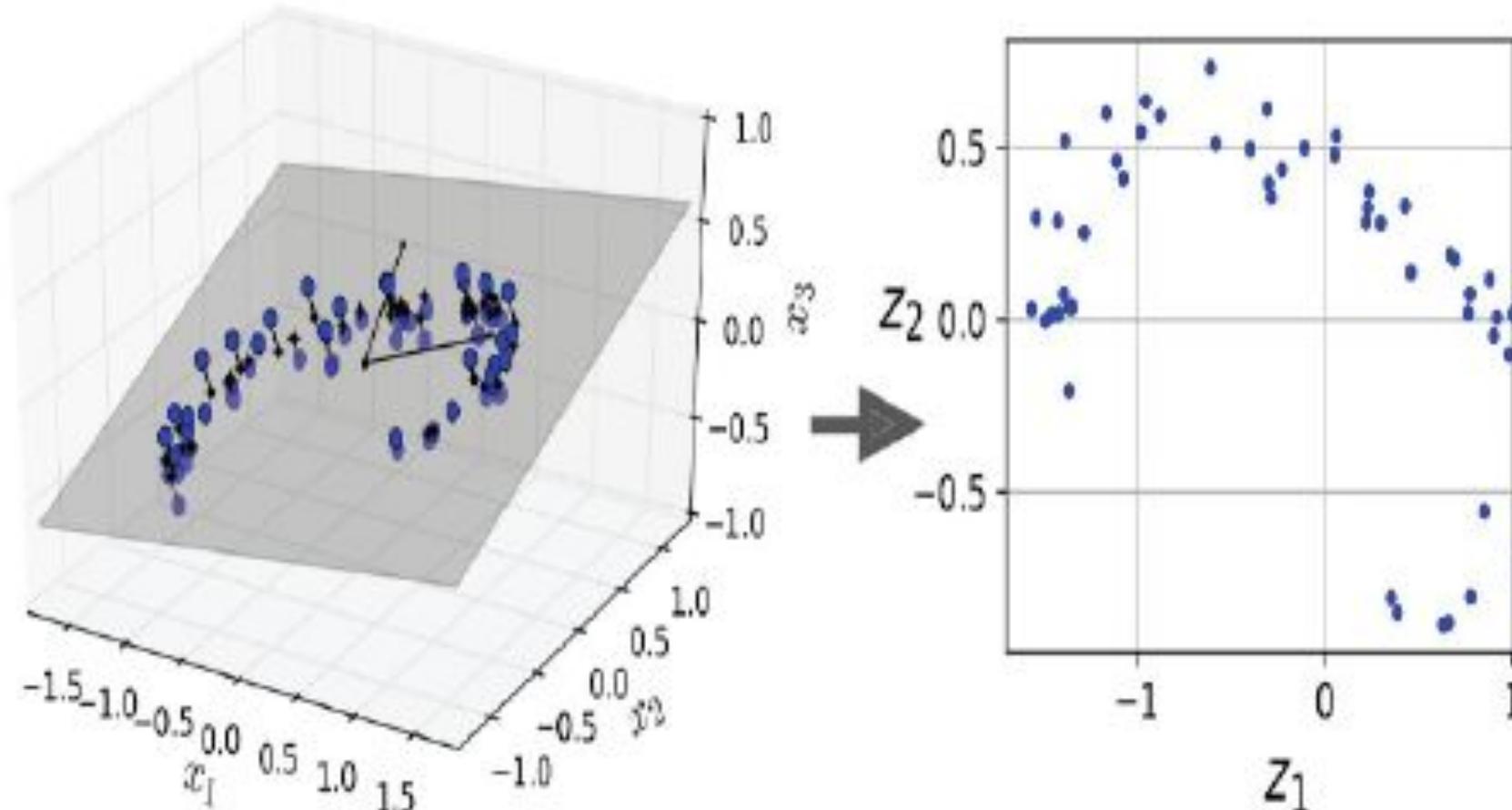




Introducción

4. Reducción de dimensionalidad (feature extraction), por ejemplo:

- Principal Component Analysis (PCA)

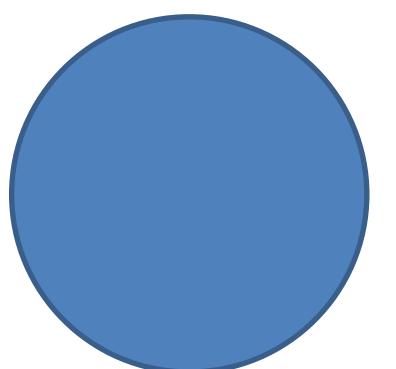
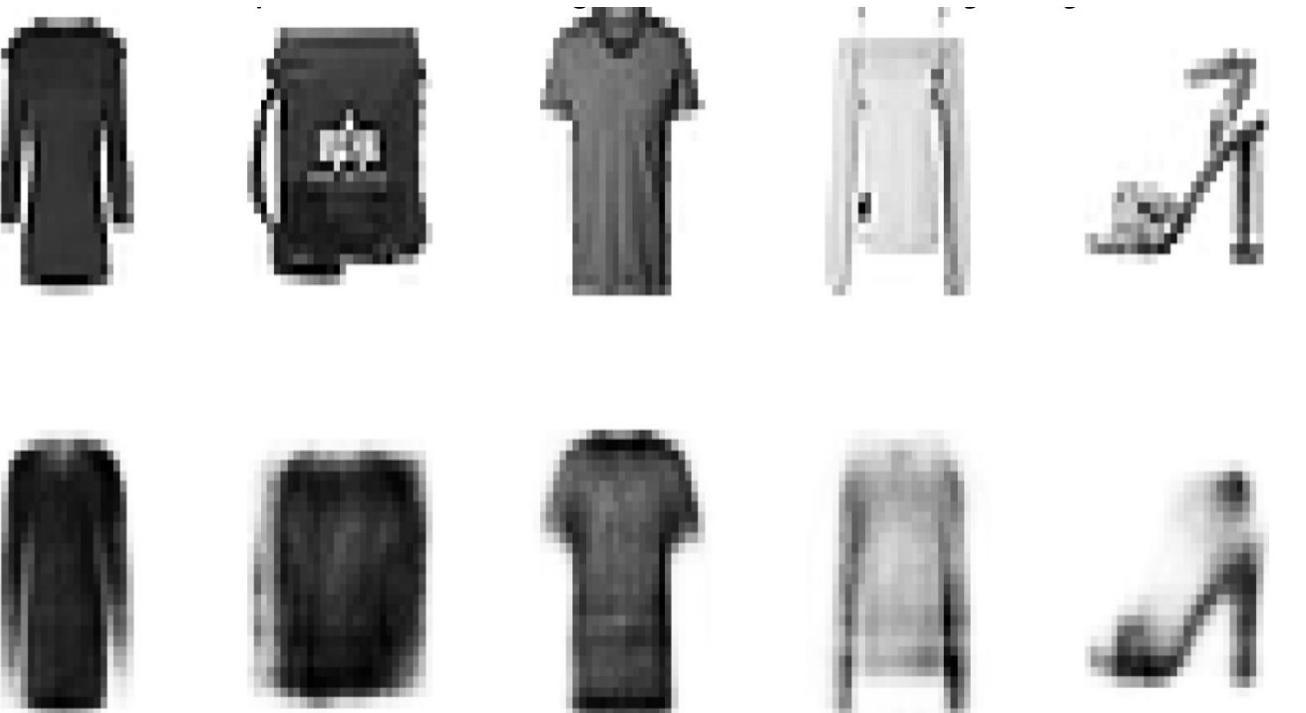




Introducción

5. Modelos generativos, por ejemplo:

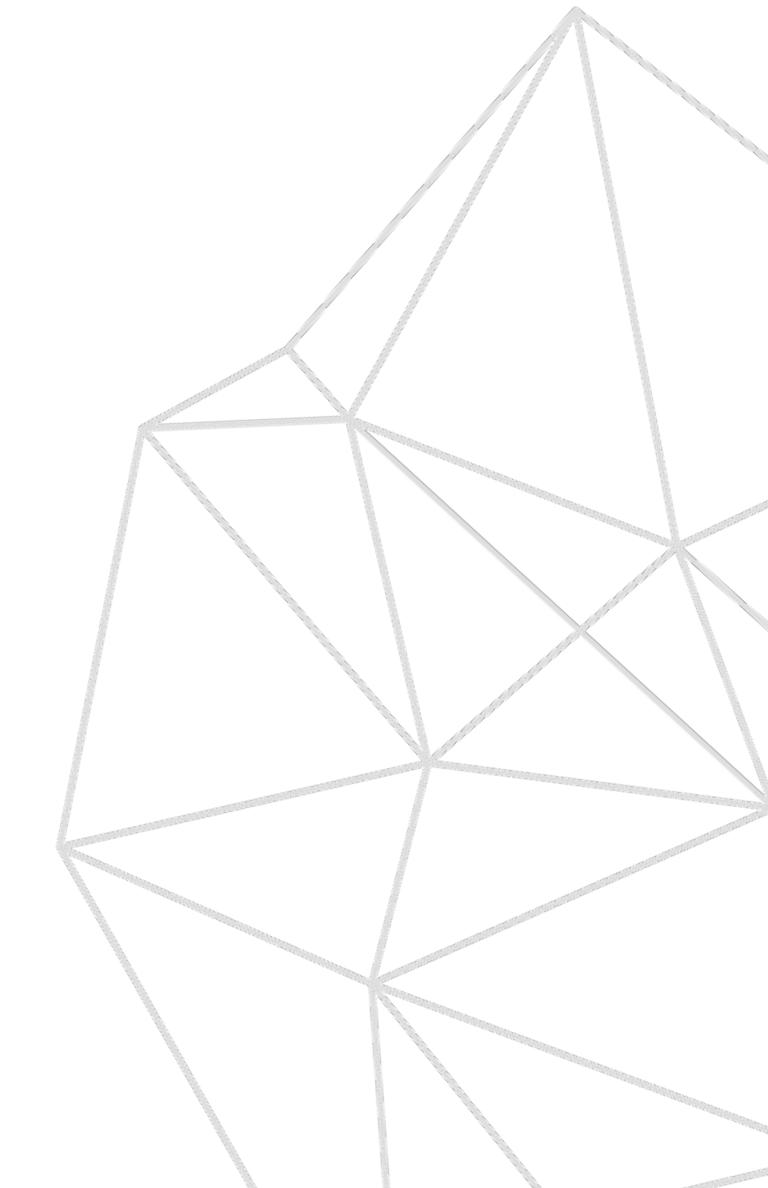
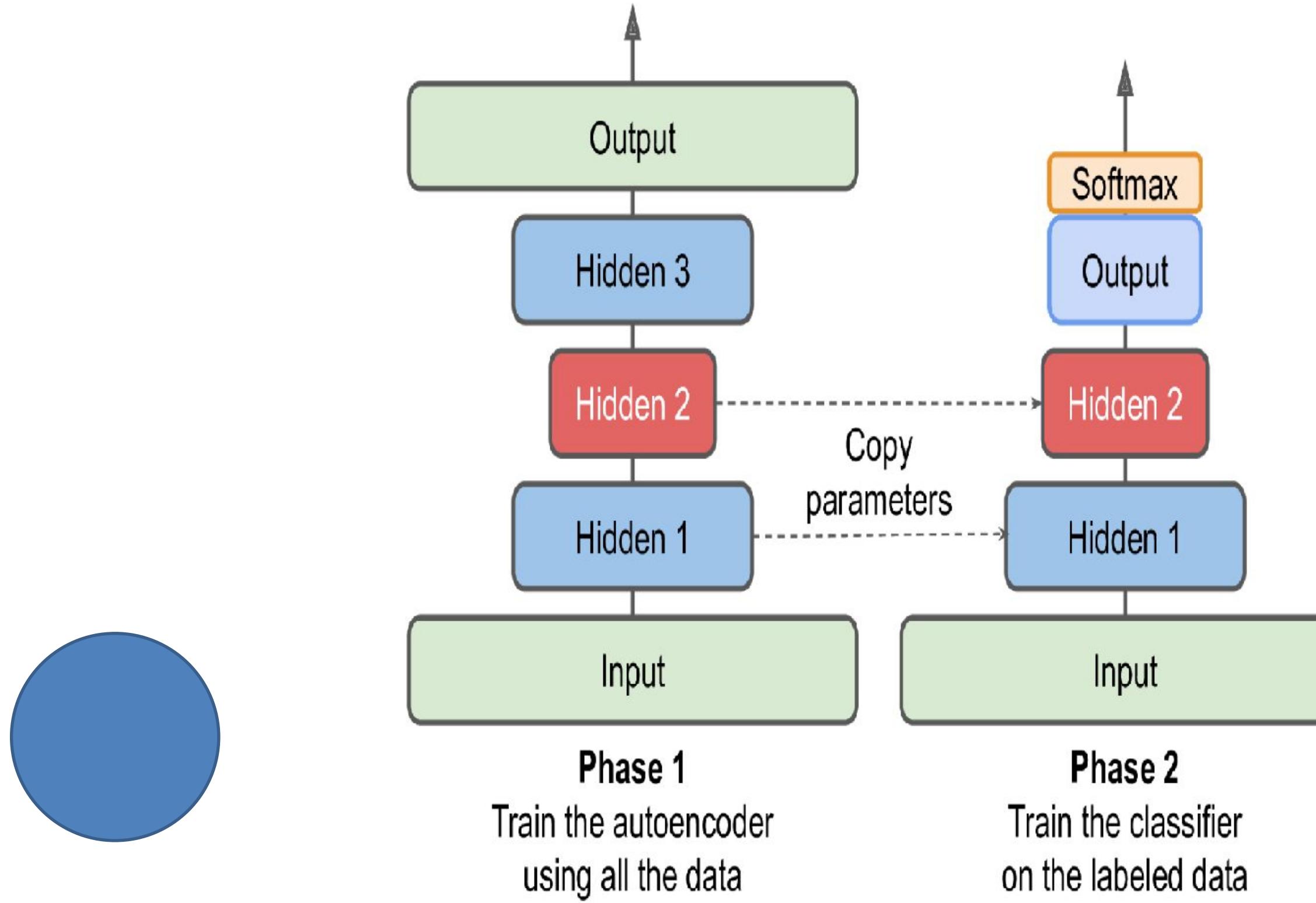
- Variational Autoencoders





Introducción

6. Preentrenamiento no supervisado

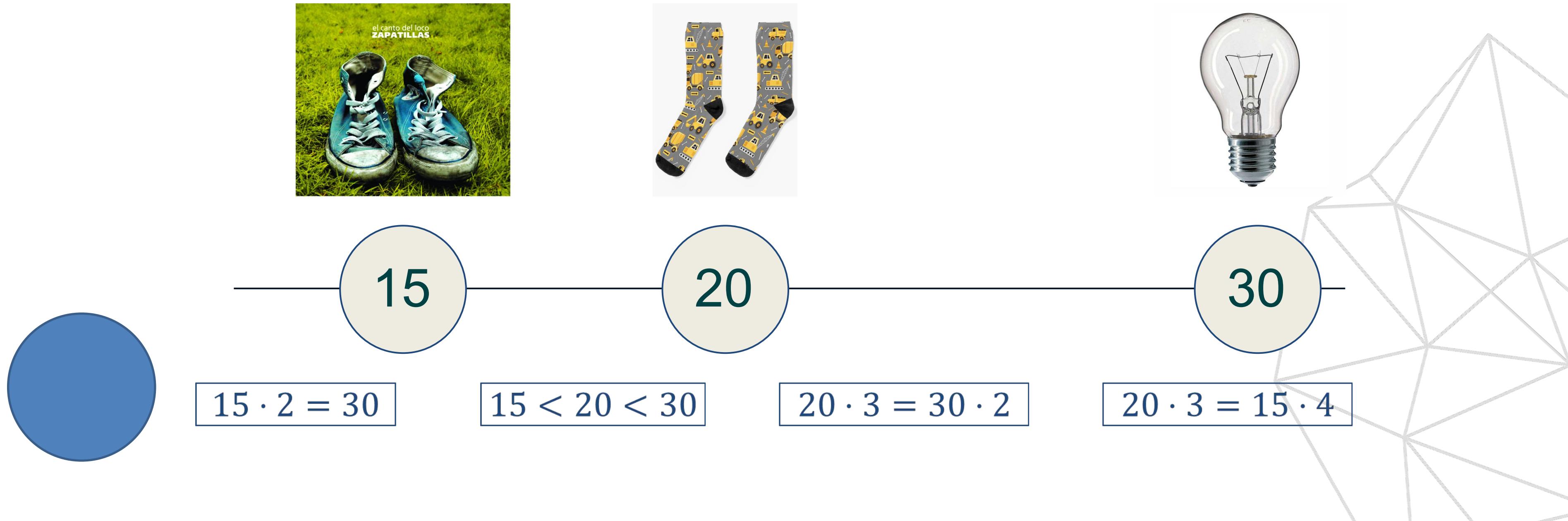




Introducción: Word embedding

¿Cómo representar las palabras para el procesamiento del lenguaje natural?

- Asignación numérica arbitraria: Vectorizar los token asignando un número arbitrario es útil, pero tiene problemas con el establecimiento de relaciones numéricas y operacionales:

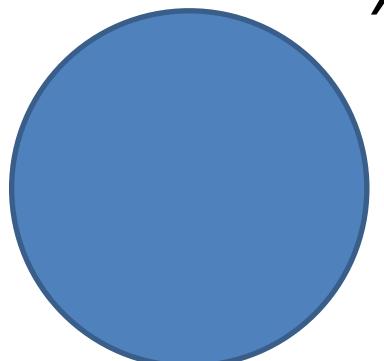




Introducción: Word embedding

¿Cómo representar las palabras para el procesamiento del lenguaje natural?

- One-hot encoding, por ejemplo:
 - Hombre: $[0, 0, \dots, 0, 0, 1]$
 - Mujer: $[0, 0, \dots, 0, 1, 0]$
 - Profesor: $[0, 0, \dots, 1, 0, 0]$
 - Etc.
- Inconvenientes:
 - Equidistancia de vectores
 - Número elevado de entradas.
 - No existe relación entre las entradas y la codificación
 - » ¿Diferencia entre árbol y arenque?: $[0, 0, \dots, 0, 1, 1]$



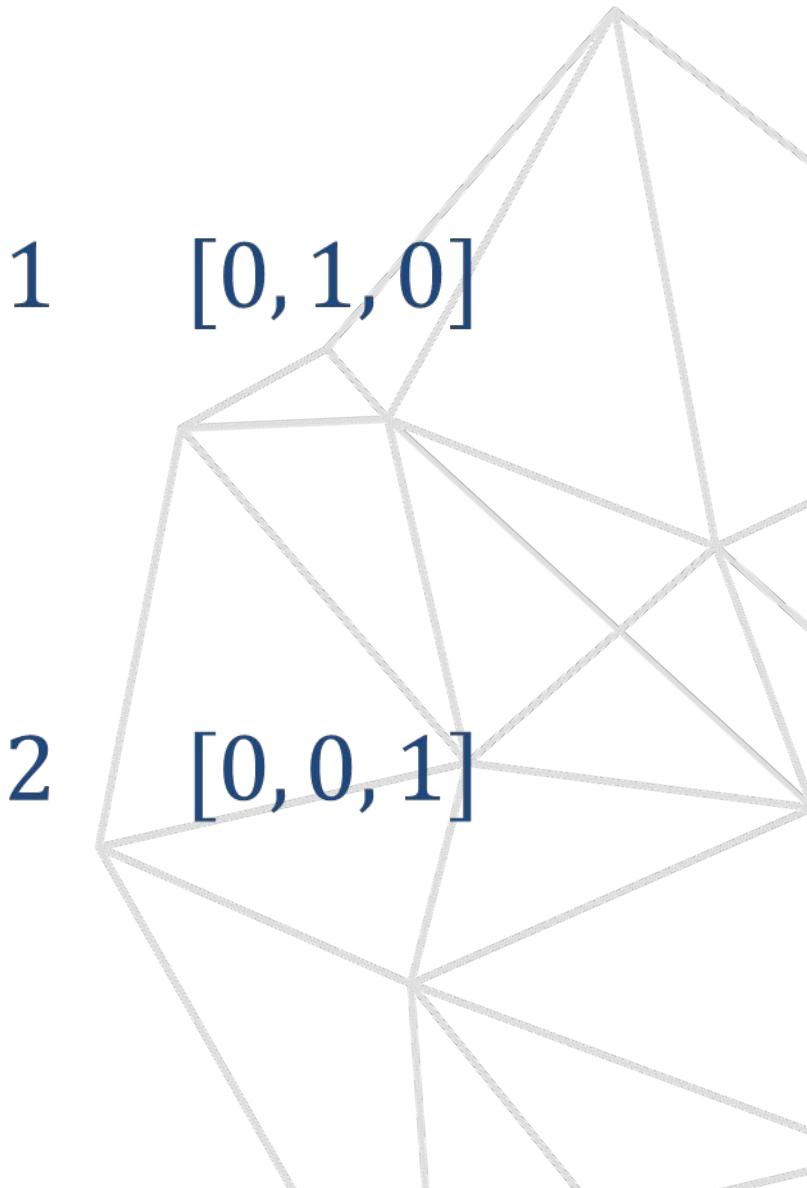
$$= 0 \quad [1, 0, 0]$$



$$= 1 \quad [0, 1, 0]$$



$$= 2 \quad [0, 0, 1]$$



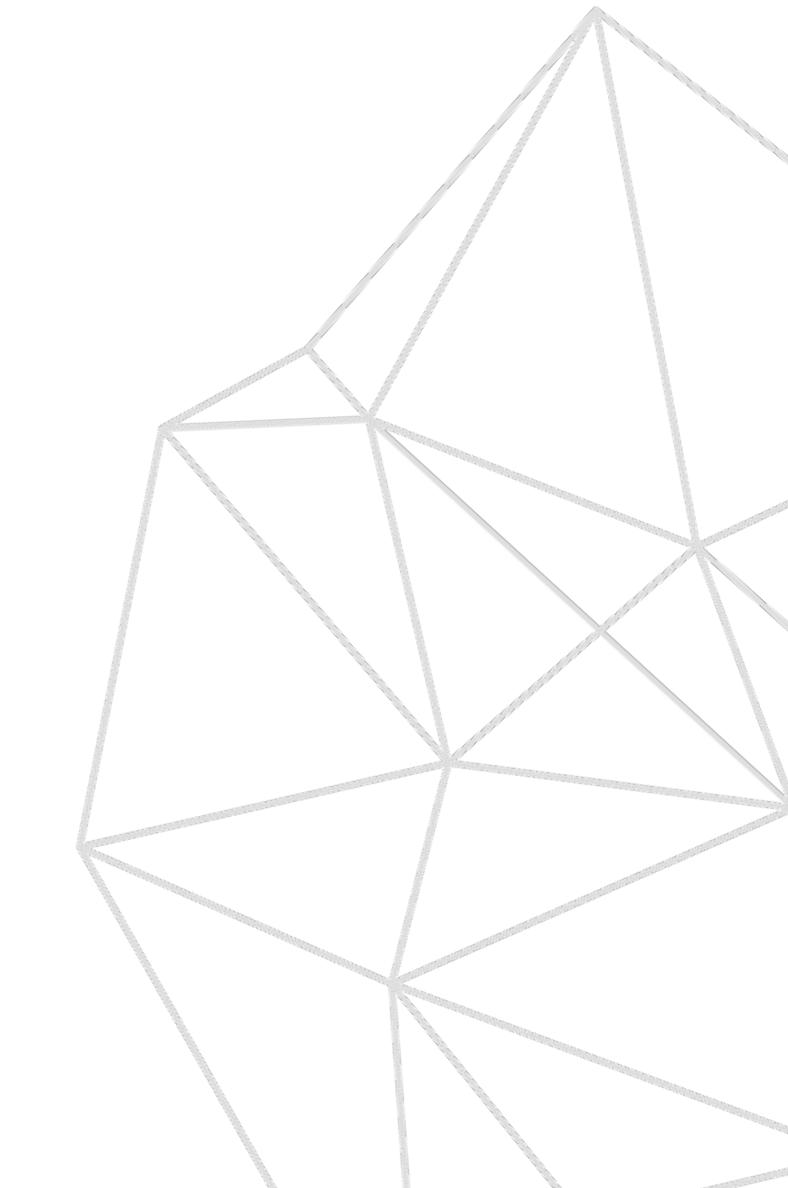
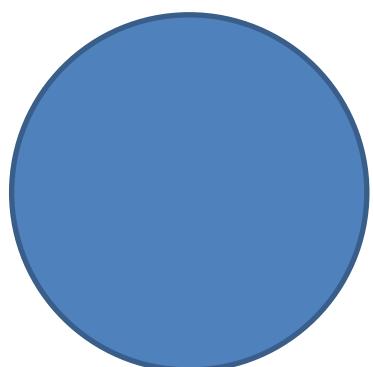


Introducción: Word embedding

¿Cómo representar las palabras para el procesamiento del lenguaje natural?

- Word embedding: Las palabras se representan mediante vectores de números reales:
 $e_{palabra}$

	Hombre	Mujer	Profesor	Profesora	León	Leona
Raza	0.75	0.75	0.75	0.75	0.23	0.23
Género	-1.00	1.00	-1.00	1.00	-1.00	1.00
Pelaje	0.25	0.20	0.25	0.20	0.80	0.75
Alimentación	0.25	0.2	0.23	0.22	0.33	0.37
Trabaja	0.5	0.5	1.00	1.00	0.2	0.8



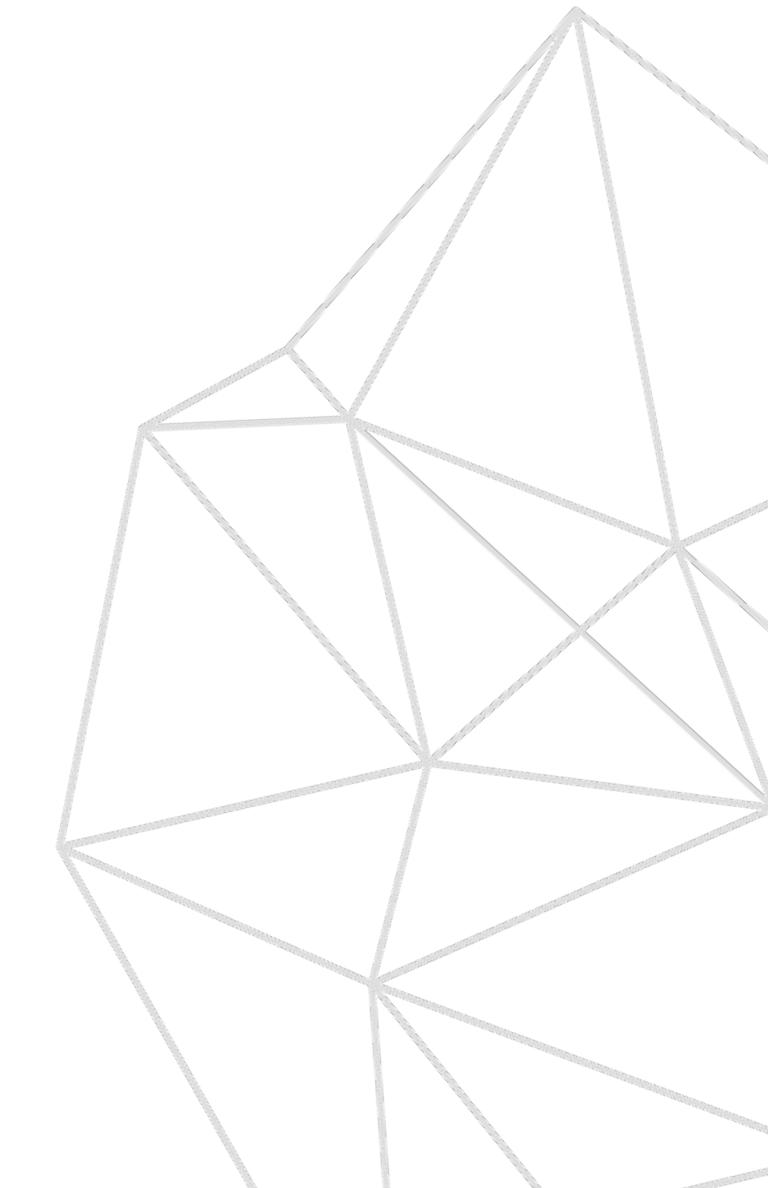
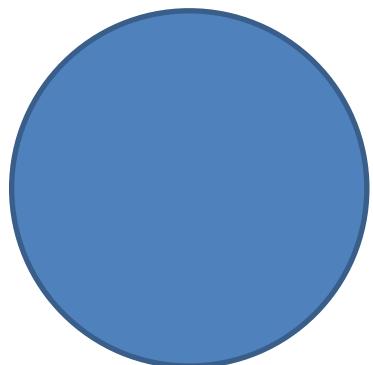


Introducción: Word embedding

¿Qué palabras representan animales de la misma raza o similar que mujer?

- Hombre
- Profesor
- Profesora

	Hombre	Mujer	Profesor	Profesora	León	Leona
Raza	0.75	0.75	0.75	0.75	0.23	0.23
Género	-1.00	1.00	-1.00	1.00	-1.00	1.00
Pelaje	0.25	0.20	0.25	0.20	0.80	0.75
Alimentación	0.25	0.2	0.23	0.22	0.33	0.37
Trabaja	0.5	0.5	1.00	1.00	0.2	0.8



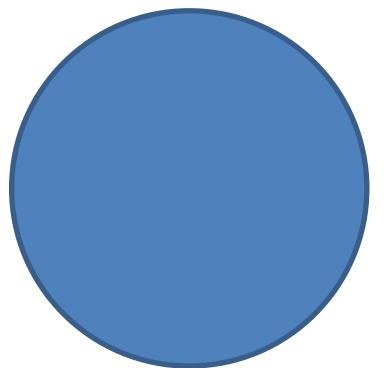


Introducción: Word embedding

¿Palabras con el mismo género que mujer?

- Leona
- Profesora

	Hombre	Mujer	Profesor	Profesora	León	Leona
Raza	0.75	0.75	0.75	0.75	0.23	0.23
Género	-1.00	1.00	-1.00	1.00	-1.00	1.00
Pelaje	0.25	0.20	0.25	0.20	0.80	0.75
Alimentación	0.25	0.2	0.23	0.22	0.33	0.37
Trabaja	0.5	0.5	1.00	1.00	0.2	0.8



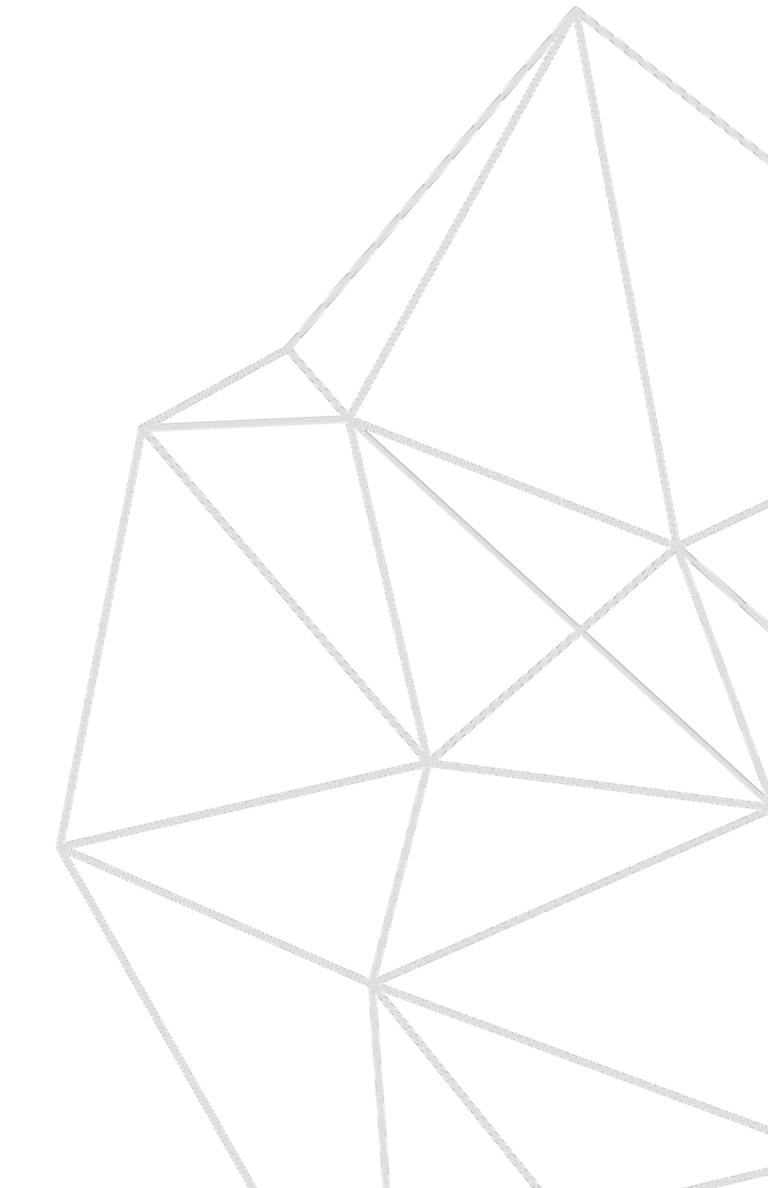
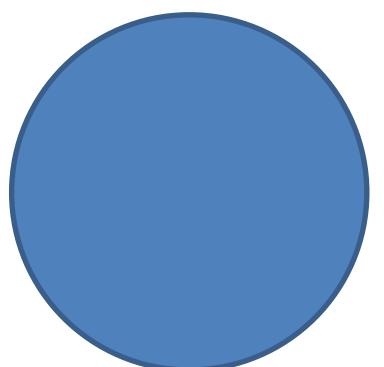


Introducción: Word embedding

Diferencias entre hombre y mujer

- La alimentación y el pelaje son ligeramente diferentes.
- La diferencia principal es el **género**.

	Hombre	Mujer	Diferencia
Raza	0.75	0.75	0.00
Género	-1.00	1.00	-2.00
Pelaje	0.25	0.20	0.05
Alimentación	0.25	0.20	0.05
Trabaja	0.50	0.50	0.00



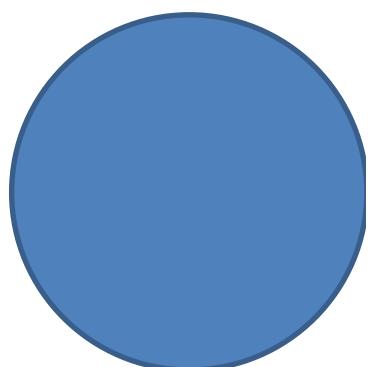


Introducción: Word embedding

Diferencias entre profesor y profesora; tenemos un resultado similar:

- La alimentación y el pelaje son ligeramente diferentes.
- La diferencia principal es el **género**.

	Profesor	Profesora	Diferencia
Raza	0.75	0.75	0.00
Género	-1.00	1.00	-2.00
Pelaje	0.25	0.20	0.05
Alimentación	0.23	0.22	0.01
Trabaja	1.00	1.00	0.00





Introducción: Word embedding

Sabiendo que la diferencia entre hombre y mujer es de genero

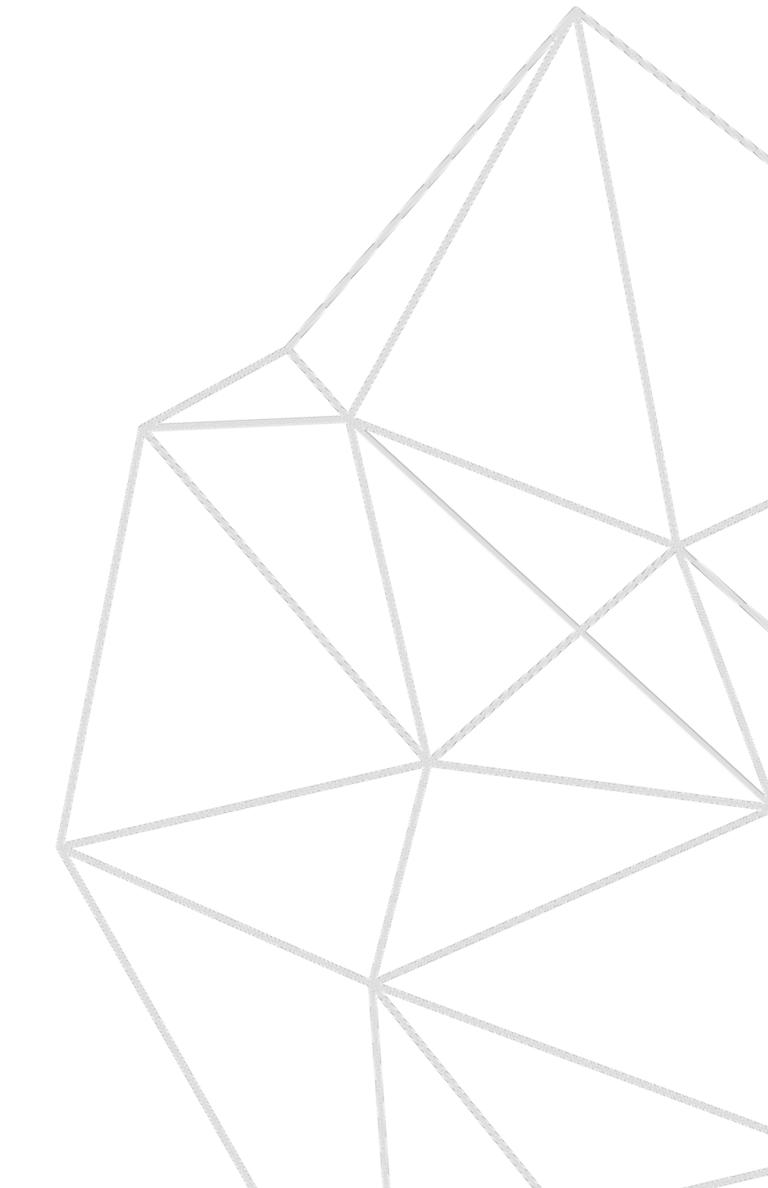
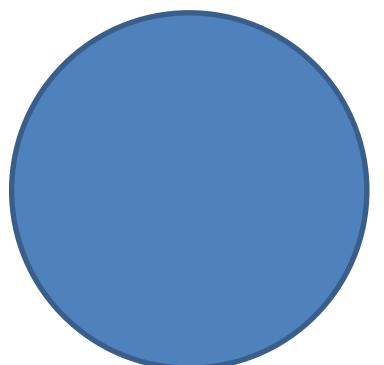
$$e_{\text{hombre}} - e_{\text{mujer}} = [0.00, -2.00, 0.05, 0.05, 0.00]$$

¿Podemos saber cuál es la palabra ω que presenta una relación similar respecto a la palabra profesor?

$$e_{\text{profesor}} - e_{\omega} \approx [0.00, -2.00, 0.05, 0.05, 0.00]$$

$$\omega = \arg \max_{\omega} \text{sim} (e_{\omega}, e_{\text{profesor}} - e_{\text{hombre}} + e_{\text{mujer}})$$

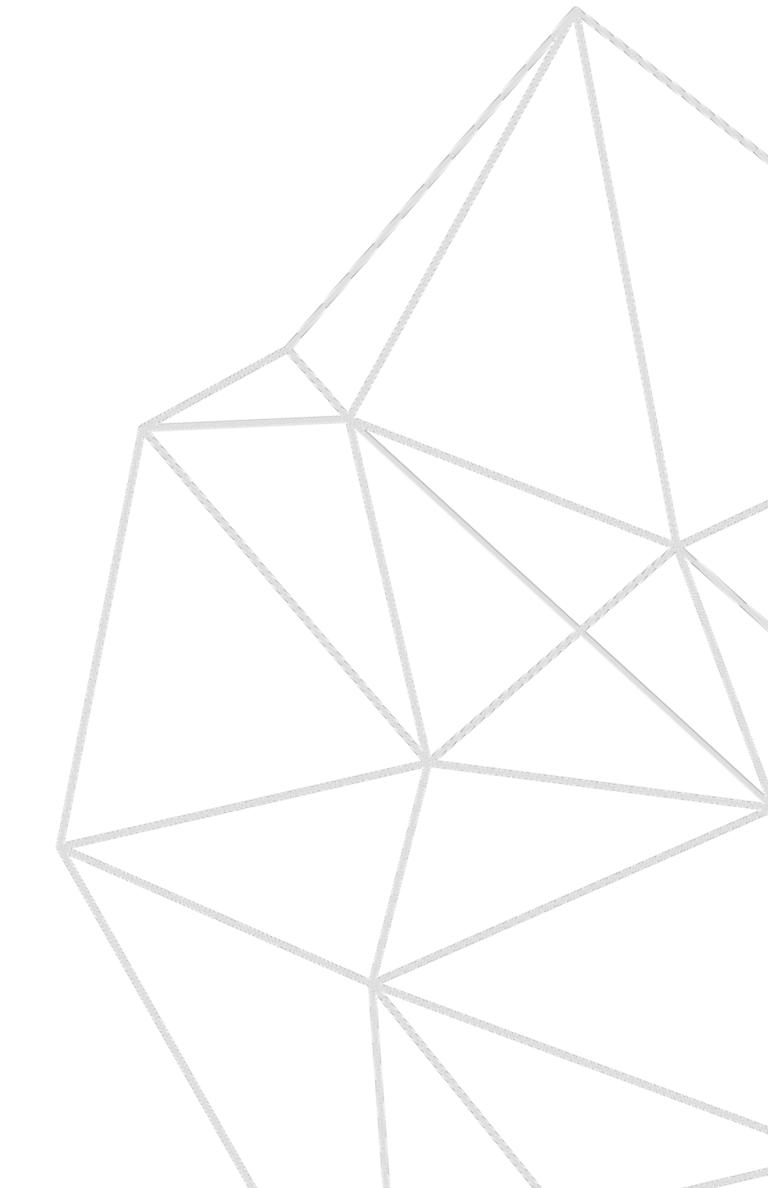
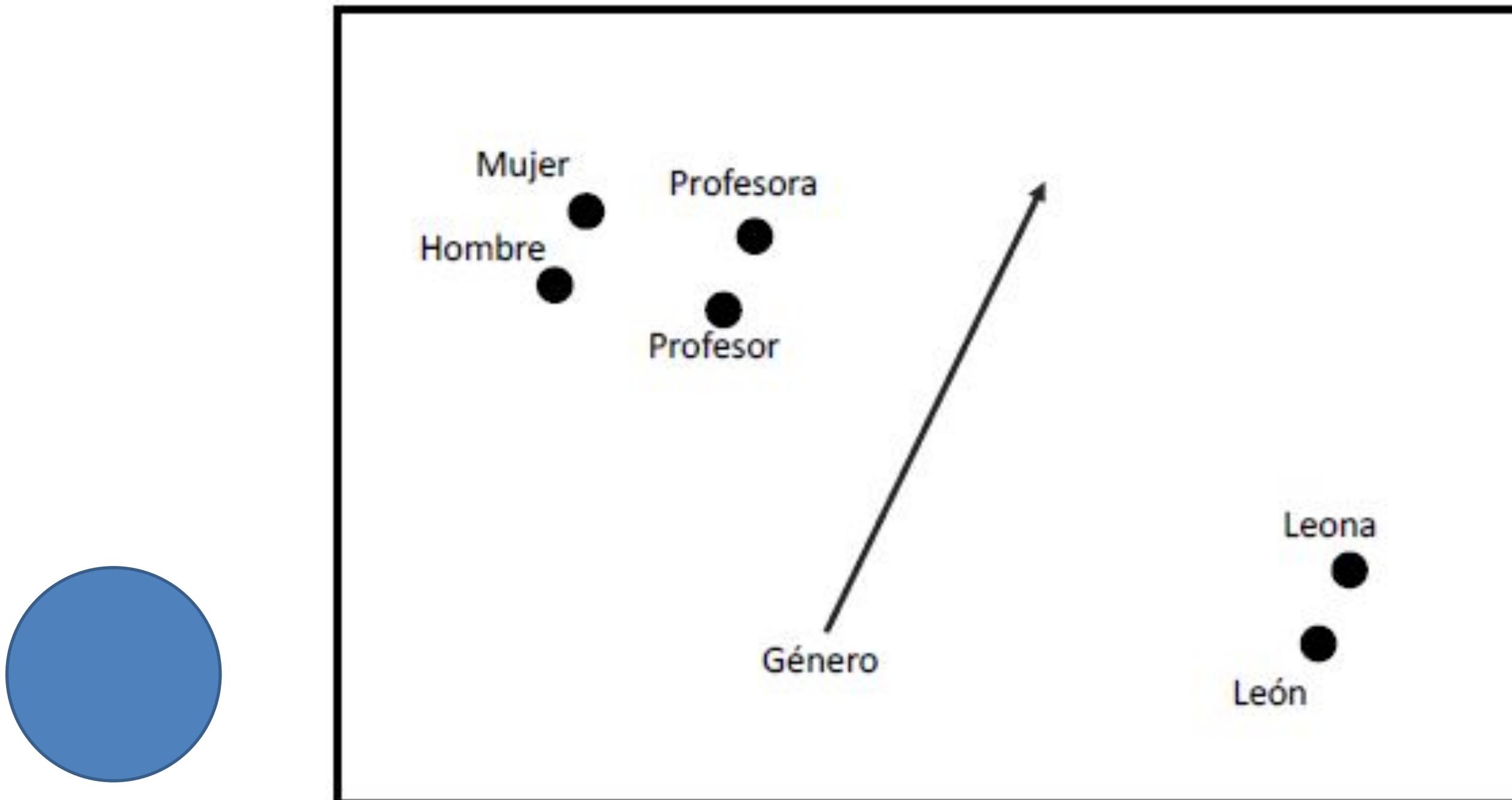
$$e_{\text{profesora}} \approx [0.75, 1.00, 0.2, 0.18, 1.00]$$





Introducción: Word embedding

Word embedding permite codificar información acerca de la semejanza entre palabras.
Representación gráfica con t-SNE





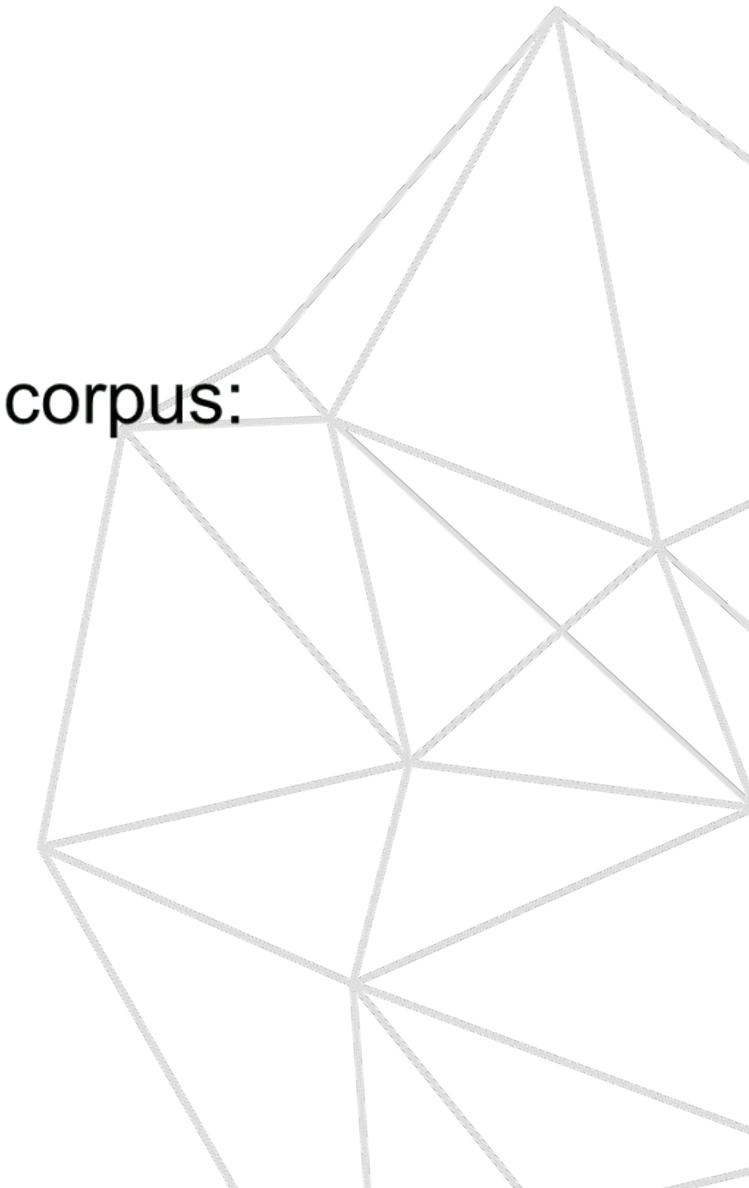
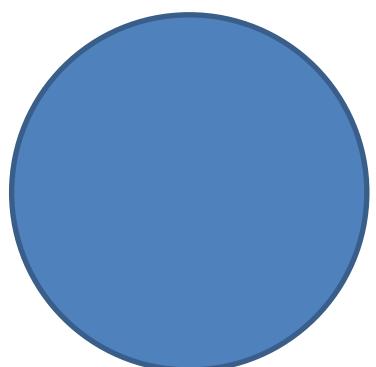
Introducción: Word embedding

En realidad no se utilizan variables como género, raza, etc.

- Las variables y sus valores son obtenidos **automáticamente** a partir de un corpus y usos de otros corpus. Por ejemplo:
- Existencia de una relación entre león y leona:
El **león** caza en la sabana africana.
La **leona** caza en la sabana africana.

Objetivo:

- Obtener la matriz que contiene todos vectores e para todas las palabras de nuestro corpus:
- **Embedding Matrix.**

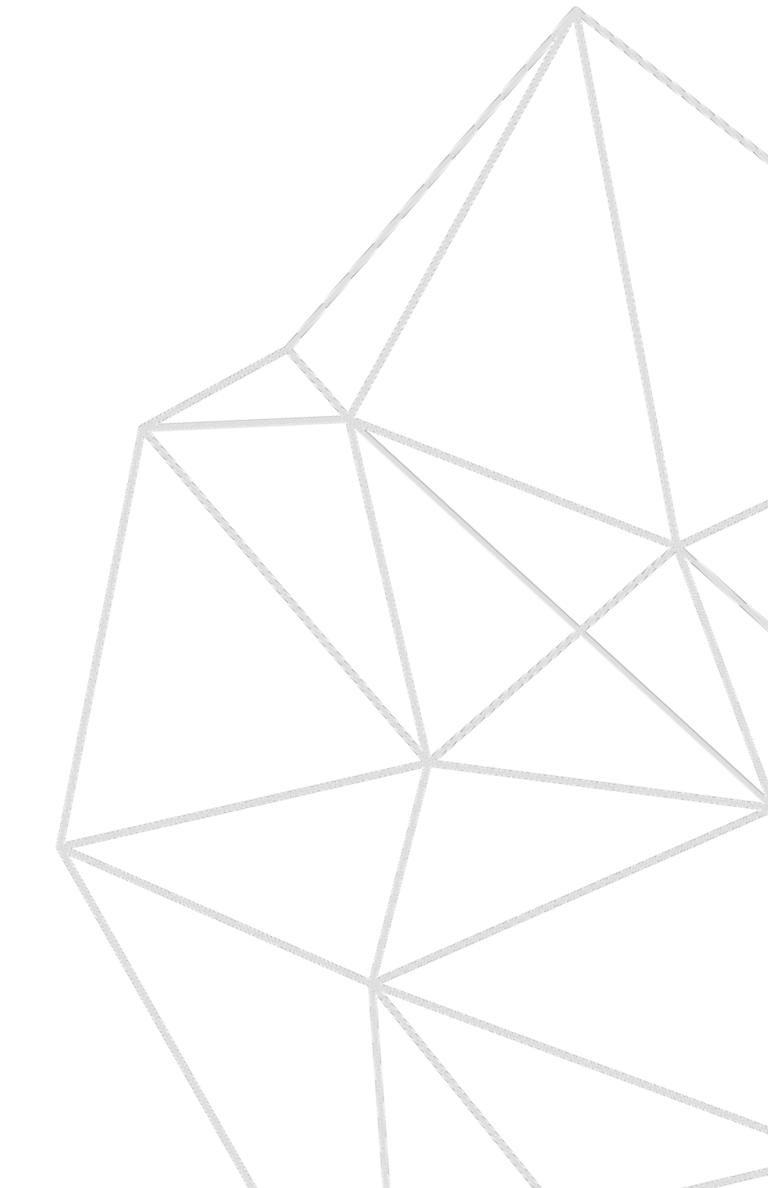
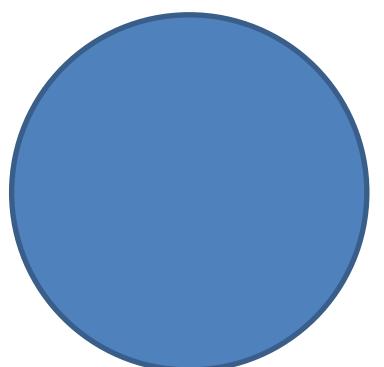




Embedding Matrix

Matriz con todas las representaciones (embeddings) de las palabras del corpus

e_{hombre}	e_{mujer}	e_{profesor}	$e_{\text{profesora}}$...	$e_{\text{león}}$	e_{leona}
0.75	0.75	0.75	0.75	...	0.23	0.23
-1.00	1.00	-1.00	1.00	...	-1.00	1.00
0.25	0.20	0.25	0.20	...	0.80	0.75
0.25	0.2	0.23	0.22	...	0.33	0.37
...
0.5	0.5	1.00	1.00	...	0.2	0.8





Word2Vec: Entrenamiento

Obtención de los vectores del *embedding matrix* por medio del entrenamiento de una red neuronal.

Tipos:

- **CBOW**: A partir de un **contexto**, palabra o conjunto de palabras sin orden (*Bag of Words, BOW*), se intenta **predecir la palabra** más probable.

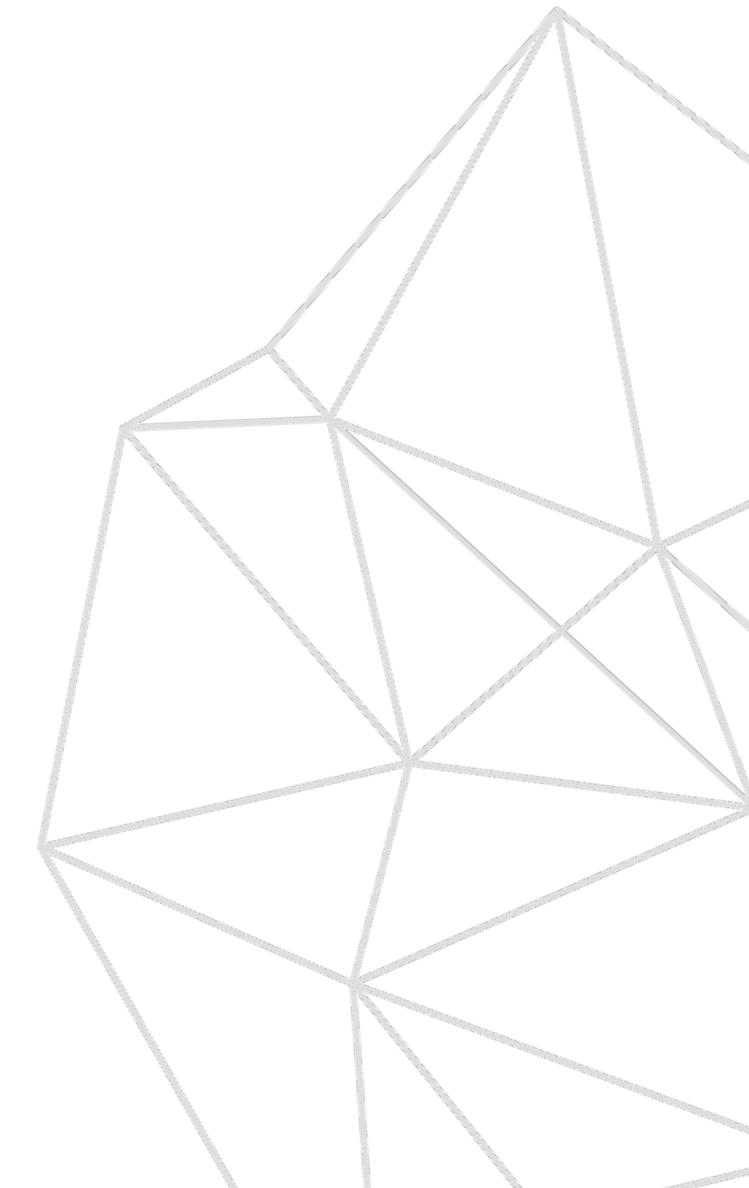
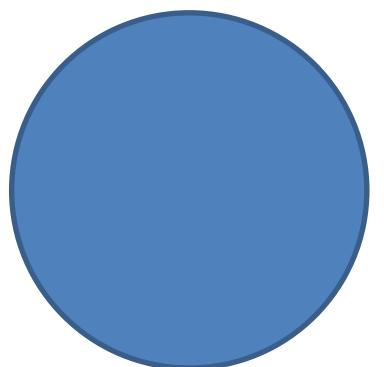
Contexto à Palabra

- **Skip-Gram**: A partir de una **palabra**, se intenta **predecir el contexto** más probable.

Palabra à Contexto

Contexto:

- N palabras cercanas, anteriores y/o posteriores.
Palabra anterior y/o posterior.

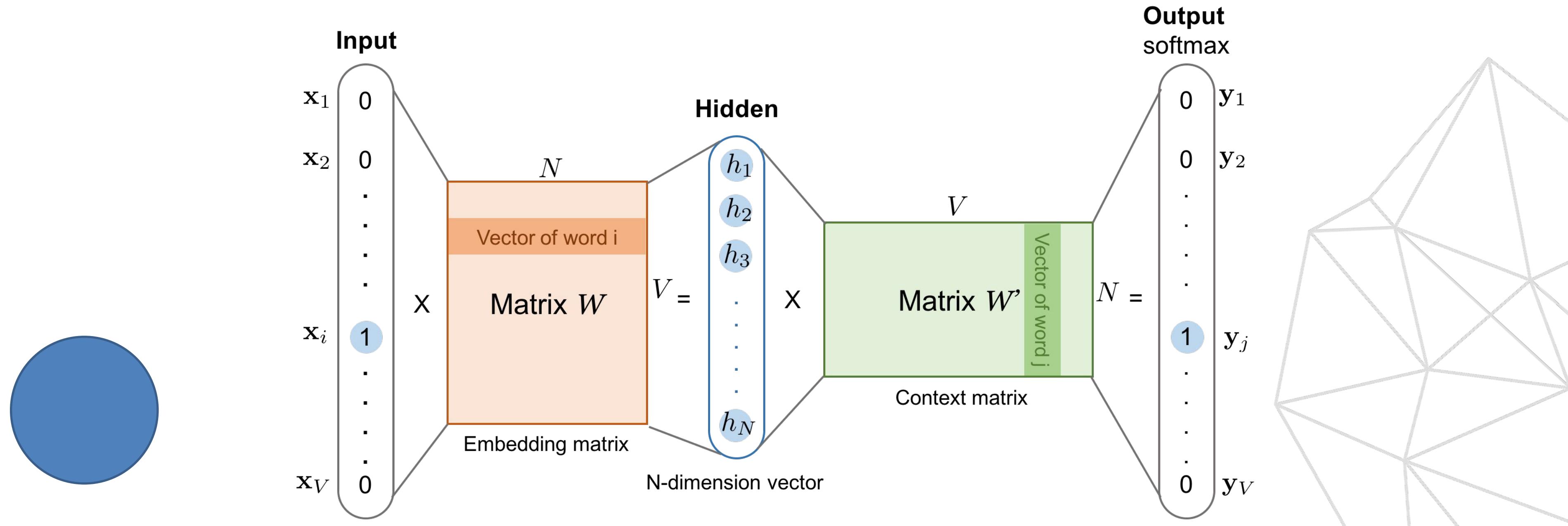




Word2Vec: Entrenamiento

Obtención de los vectores de tamaño N por medio del entrenamiento de una red neuronal de N neuronas ocultas.

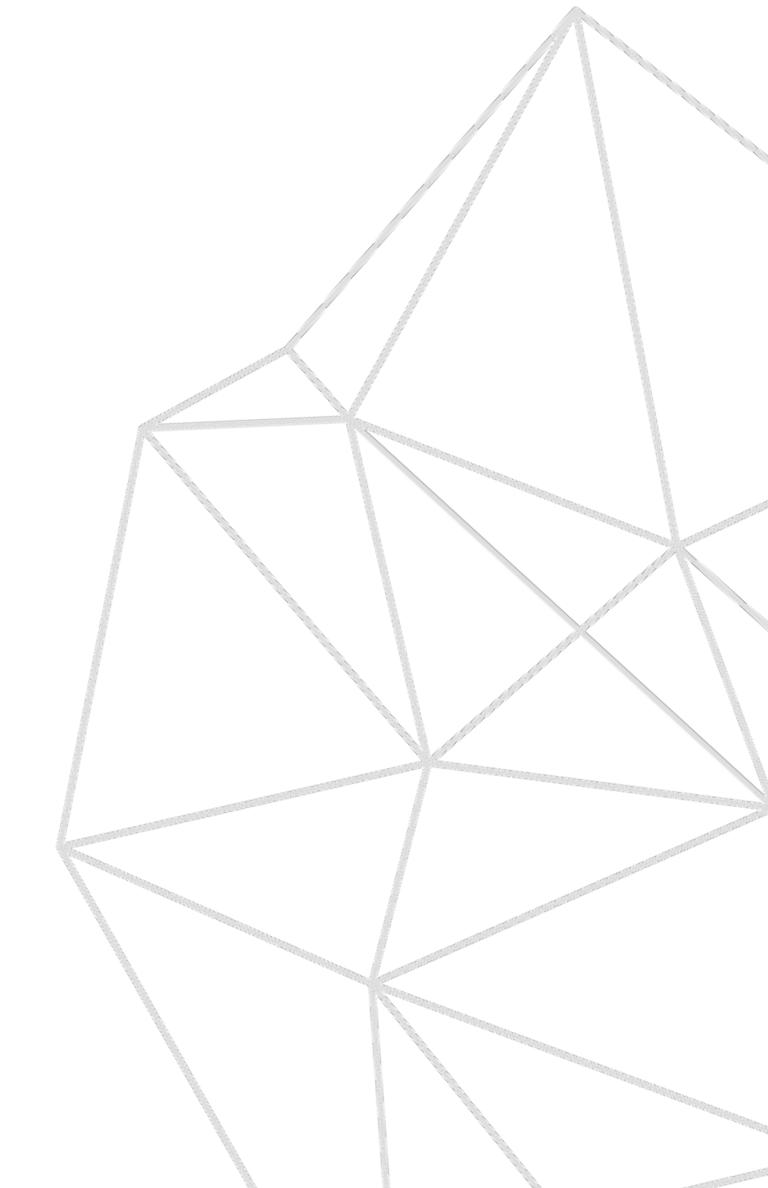
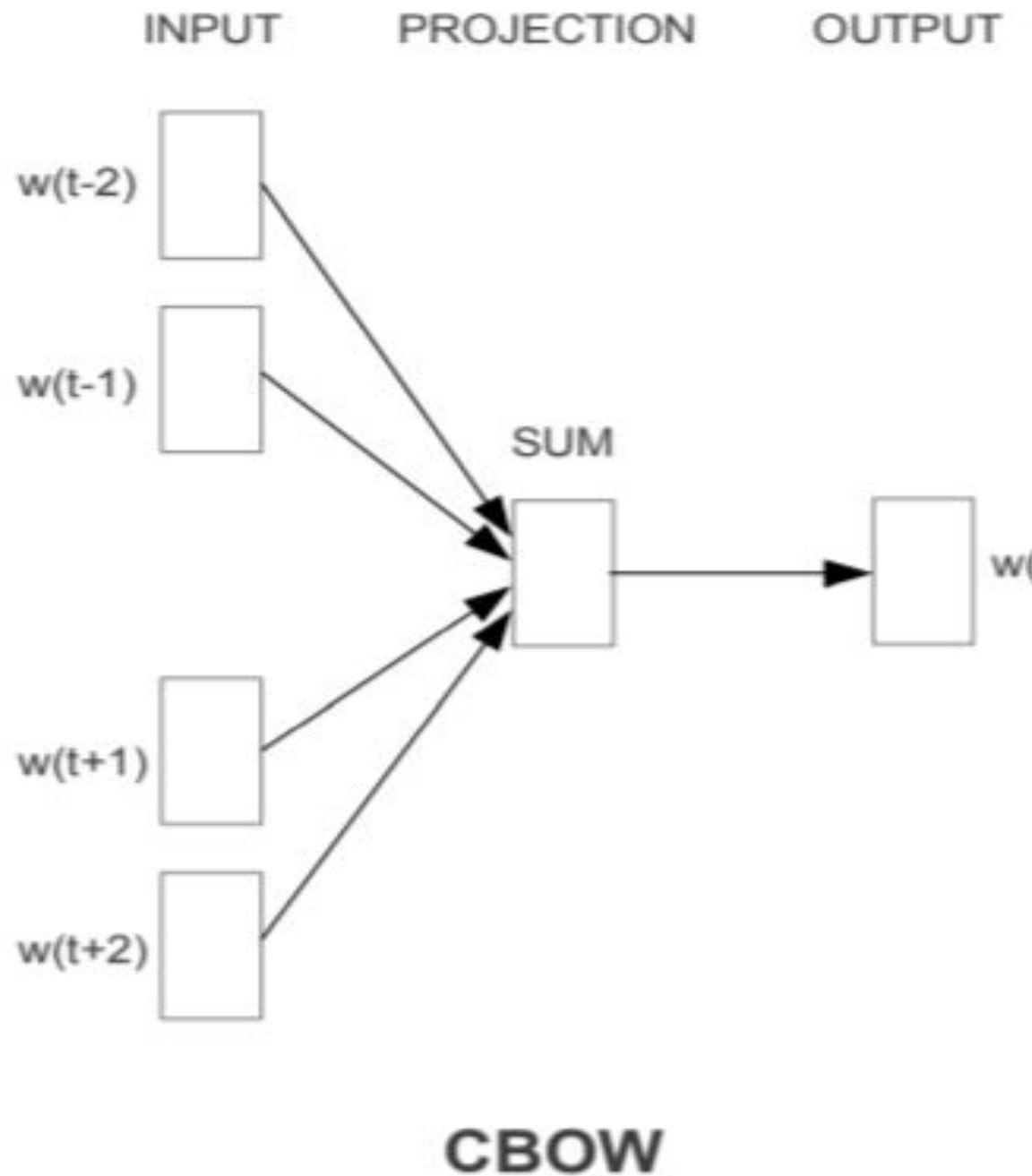
Entrada: One-Hot encoding del corpus.





Word2Vec: Entrenamiento

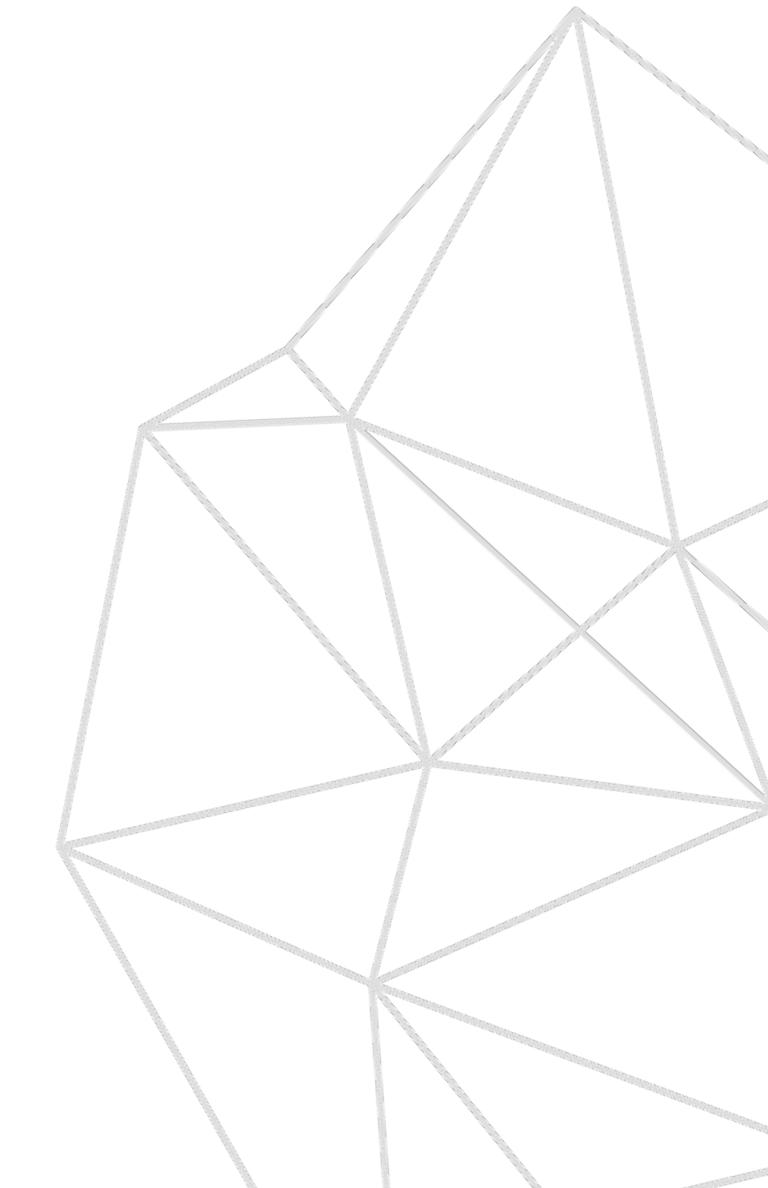
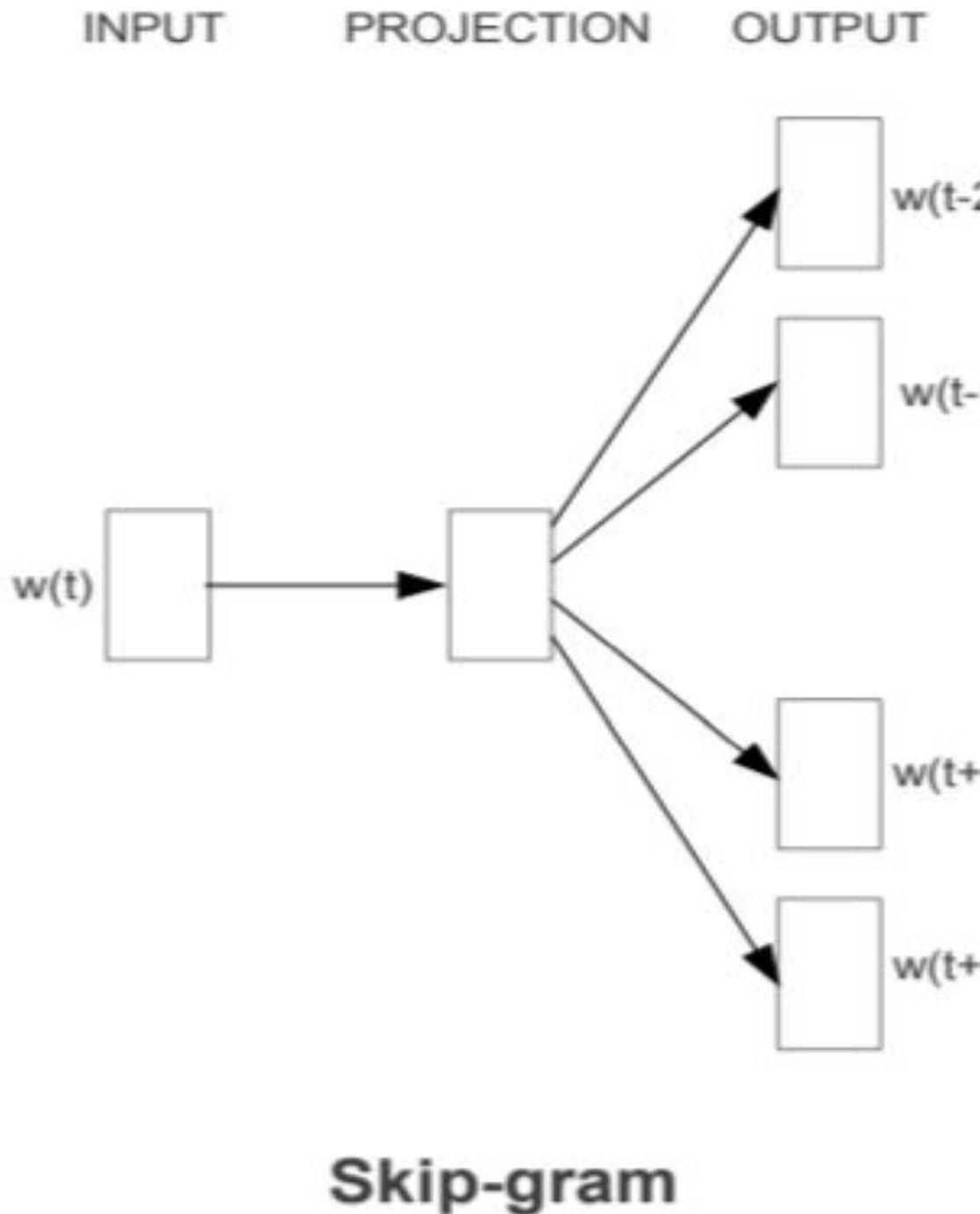
CBOW: A partir de un **contexto**, palabra o conjunto de palabras sin orden (*Bag of Words, BOW*), se intenta **predecir la palabra** más probable: **contexto** \rightarrow **palabra**





Word2Vec: Entrenamiento

Skip-Gram: A partir de una **palabra**, se intenta **predecir el contexto** más probable: **palabra à contexto**





Word2Vec: Entrenamiento

Función de activación para la capa de salida:

- Softmax:

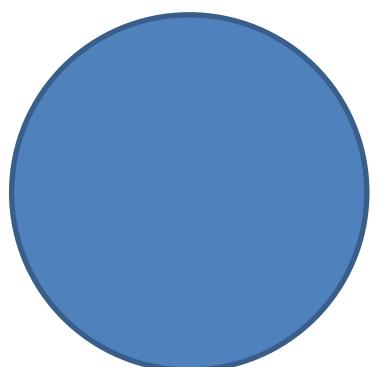
Para un corpus de dimensión C , la salida será un vector de dimensión C representando las probabilidades:

- CBOW: $P(\text{palabra}|\text{contexto})$
- SKIM-GRAM: $P(\text{contexto}|\text{palabra})$

Inconveniente:

Para C muy grande, el coste computacional del cálculo de la función de activación es muy elevado.

Solución: **Softmax jerárquico**.





Word2Vec: Entrenamiento

Función de activación para la capa de salida:

- Sigmoide: **Negative Sampling**

Solución más eficiente que softmax.

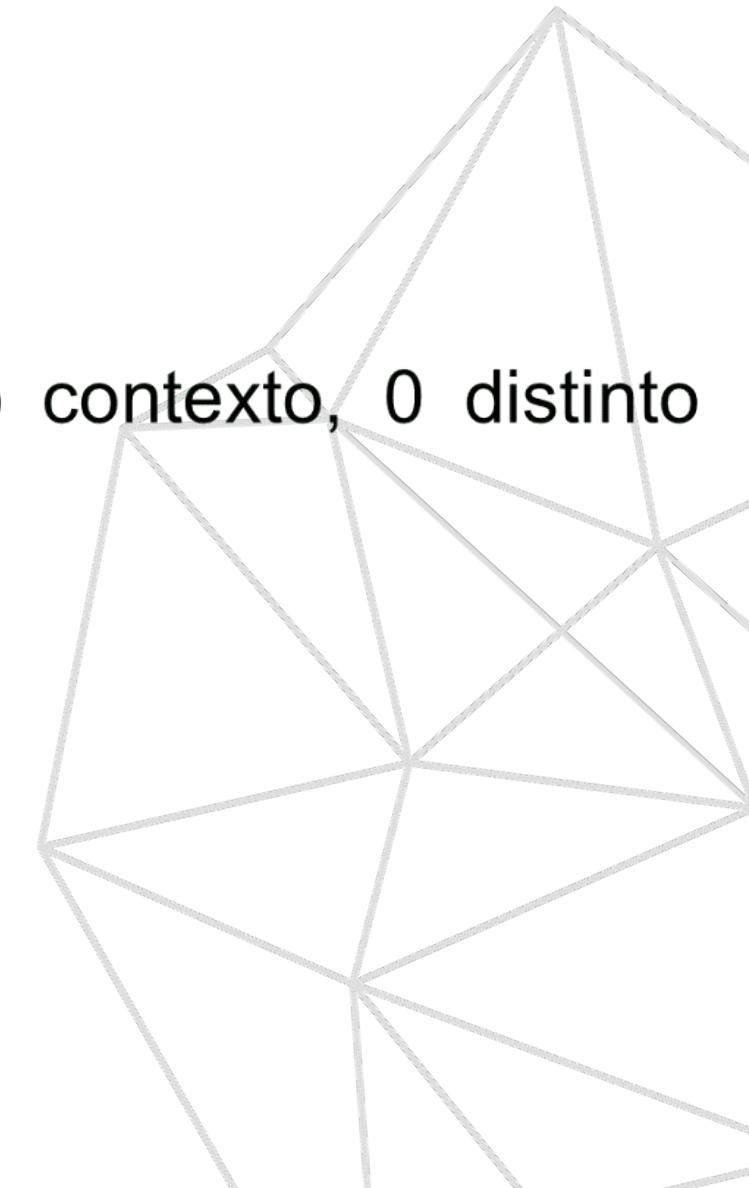
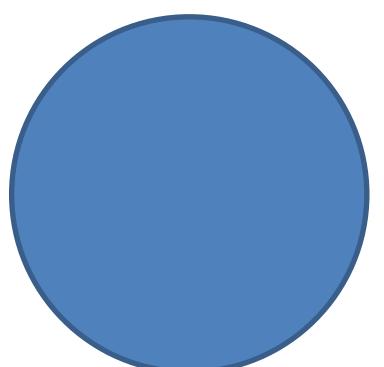
Salida $P(y = 1 | \text{contexto}, \text{palabra})$

- Palabras del mismo contexto: ~1
- Palabras de contextos diferentes: ~0

Contexto	Palabra	y
perro	salvaje	1
perro	zumo	0
perro	el	0
perro	piña	0
perro	cama	0

Entrenamiento:

- Se proporciona un conjunto de tuplas (contexto, palabra) y la salida (1 mismo contexto, 0 distinto contexto).





Word2Vec: Entrenamiento

Función de activación para la capa de salida:

- Sigmoide: **Negative Sampling**

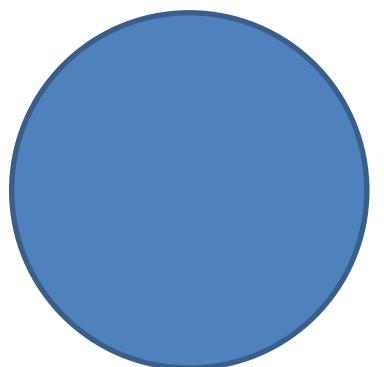
Solución más eficiente que softmax.

Salida $P(y = 1 | \text{contexto}, \text{palabra})$

- Palabras del mismo contexto: ~1
- Palabras de contextos diferentes: ~0

Entrenamiento:

- Se proporciona un conjunto de tuplas (contexto, palabra) y la salida (1 mismo contexto, 0 distinto contexto).
 - En lugar de entrenar el corpus completo, se entrena un conjunto reducido de tamaño $k + 1$.
 - » Reducción coste computacional
 - » $k = 5 - 20$ corpus pequeños
 - » $k = 2 - 5$ corpus grandes





Word2Vec: Entrenamiento

Función de activación para la capa de salida:

- Sigmoide: **Negative Sampling**

Solución más eficiente que softmax.

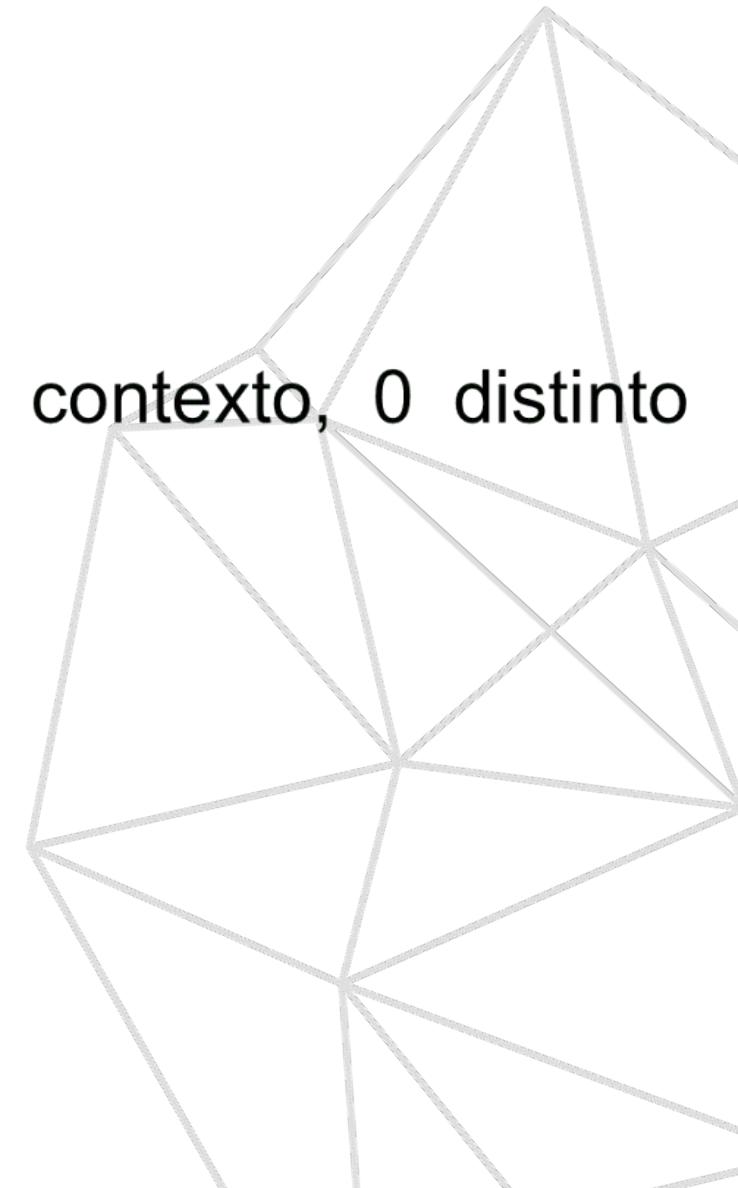
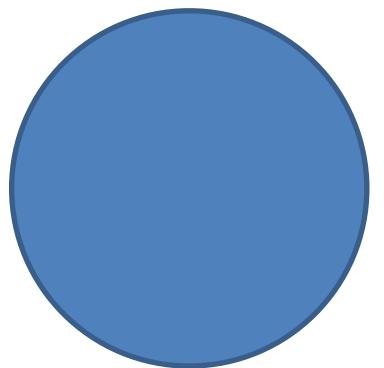
Salida $P(y = 1 | \text{contexto}, \text{palabra})$

- Palabras del mismo contexto: ~1
- Palabras de contextos diferentes: ~0

Entrenamiento:

- Se proporciona un conjunto de tuplas (contexto, palabra) y la salida (1 mismo contexto, 0 distinto contexto).
 - En lugar de entrenar el corpus completo, se entrena un conjunto reducido de tamaño $k + 1$.
 - Elección de pares para la palabra w_j , (f frecuencia)

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^C f(w_j)^{3/4}}$$





Word2Vec: Entrenamiento

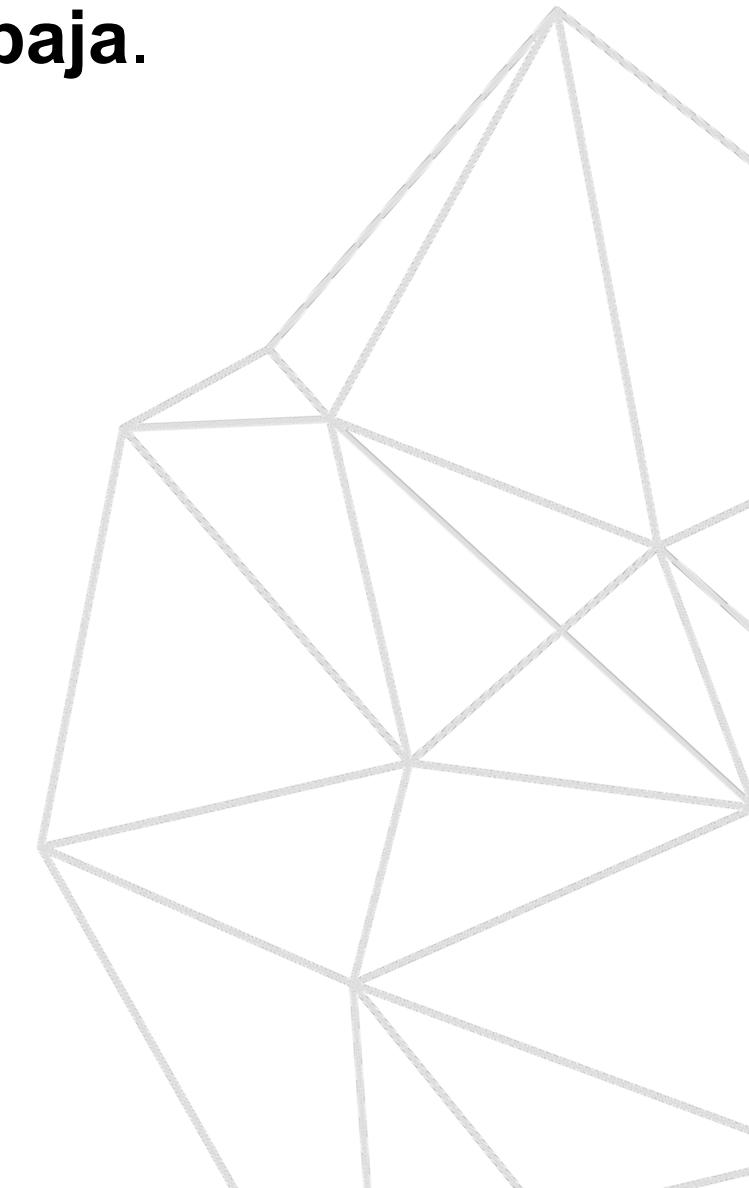
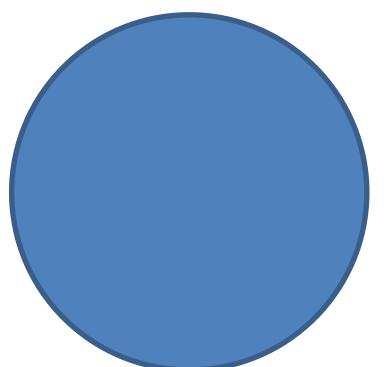
Función de activación para la capa de salida:

- Softmax jerárquico

Mejores resultados para palabras con frecuencias **bajas**

- Sigmoide: Negative Sampling

Mejores resultados para palabras con frecuencias **altas** y vector latente de dimensión **baja**.





Word2Vec: Dimensiones

Dimensión del Word Embedding:

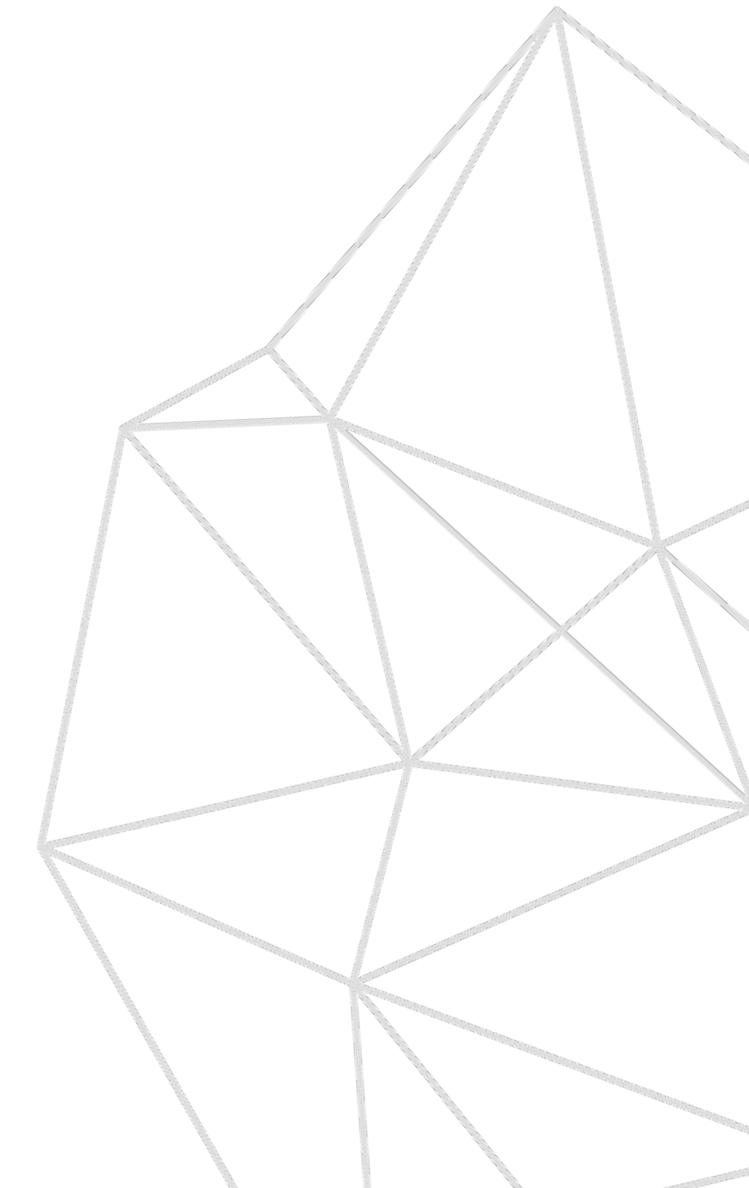
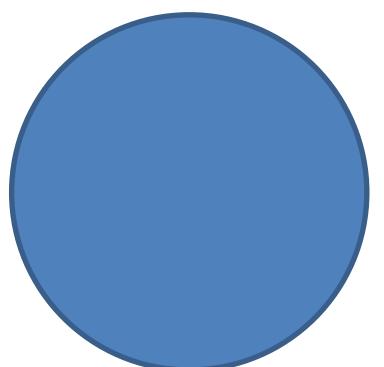
La calidad del Word Embedding es mayor cuanto mayor sea la dimensión del vector latente, aunque llega un momento en que esta mejora es marginal.

- Entre 100 y 1000 da buenos resultados.

Dimensión del contexto:

CBOW: ~5 palabras

Skip-Gram: ~10 palabras





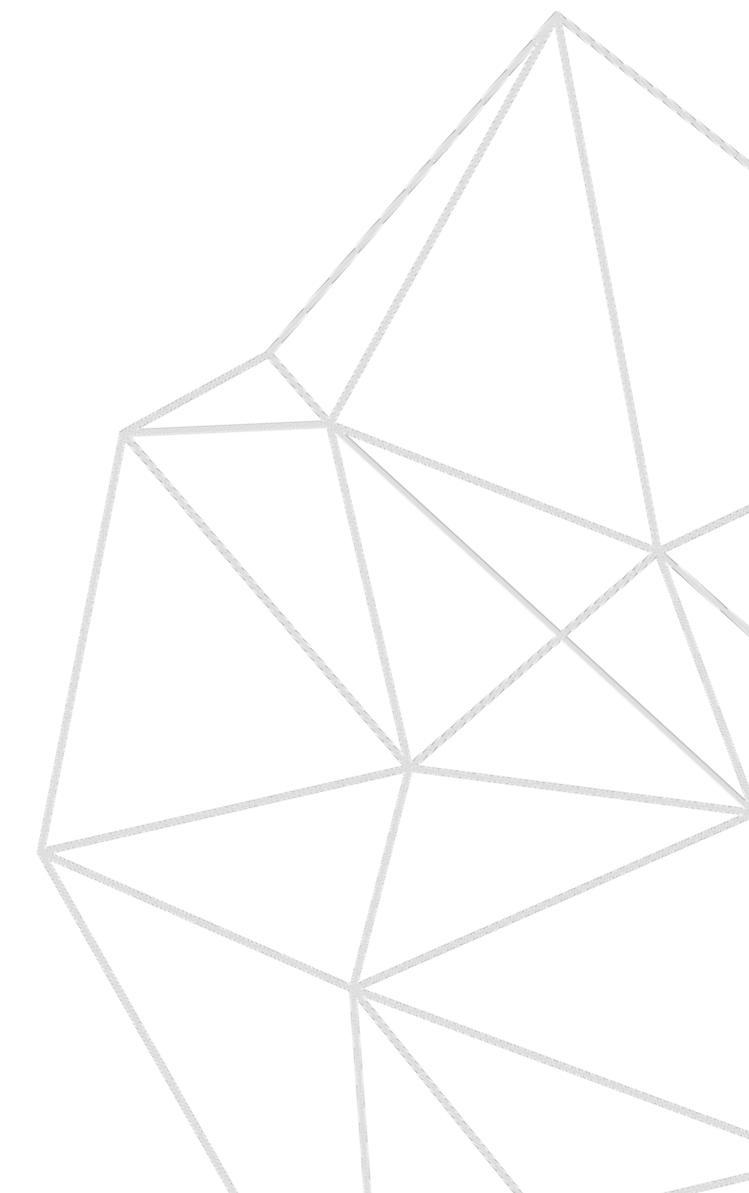
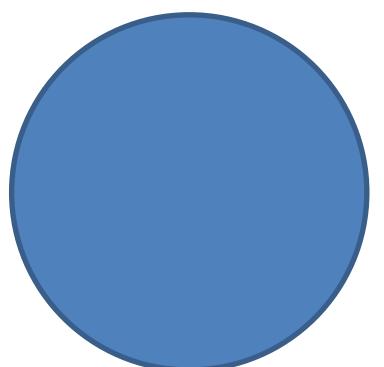
Word2Vec: Rendimiento

CBOW:

- Mejor rendimiento
- Peores resultados

Skip-Gram:

- Más lento
- Mejores resultados, sobre todo con palabras poco frecuentes





GLOVE: Alternativa a Word2Vec

Alternativa más eficiente a Word2Vec

Objetivo:

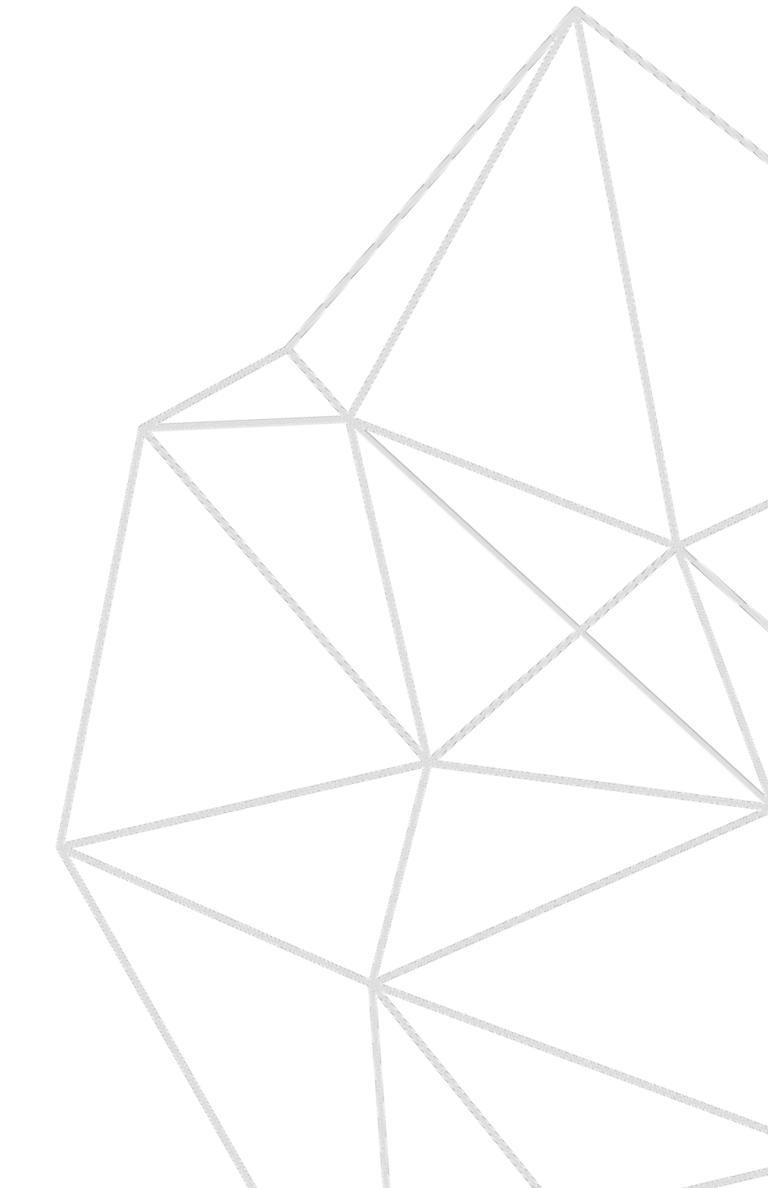
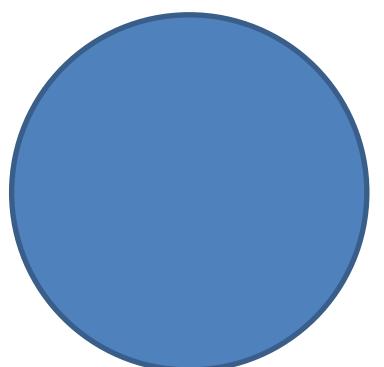
- Obtener vectores a partir de la información acerca del número de veces, X_{ij} , que aparece una palabra j en el contexto de la palabra i .

Propiedad:

$$X_{ij} = X_{ji}$$

$$\min \sum_{i=1}^C \sum_{j=1}^C f(X_{ij}) (\theta_i^T e_j + b_i - b'_j - \log X_{ij})^2$$

Da más importancia a
las palabras cercanas





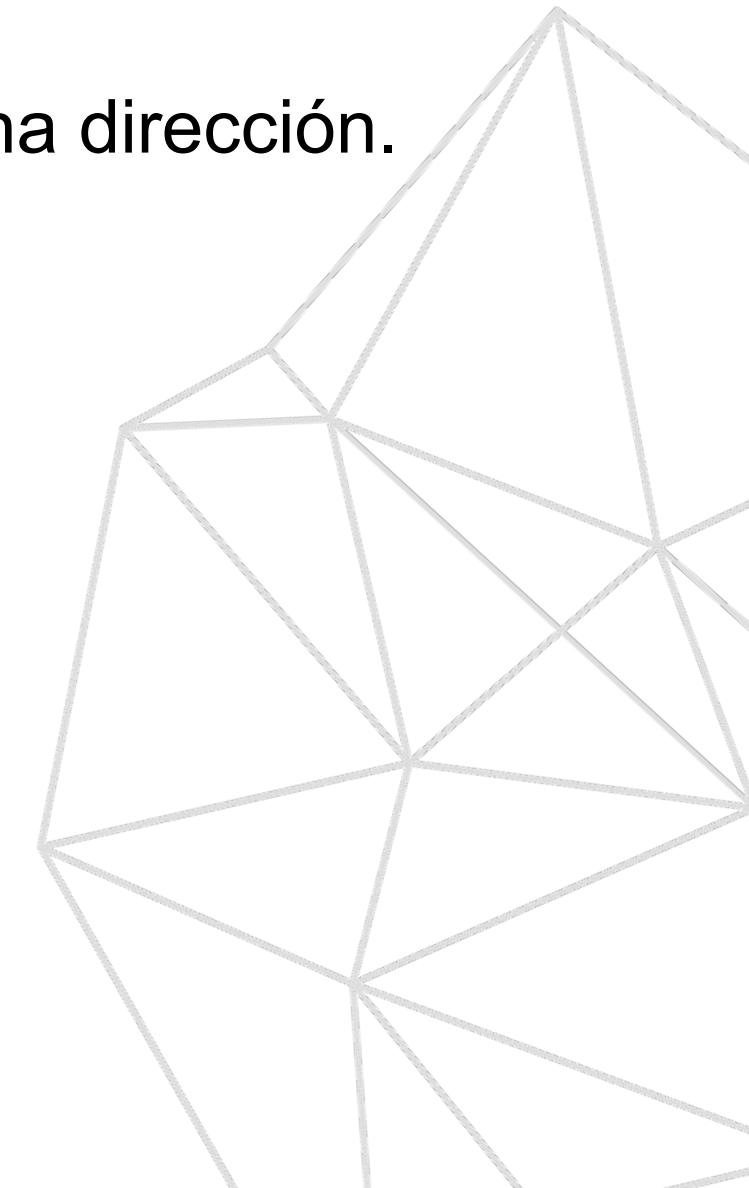
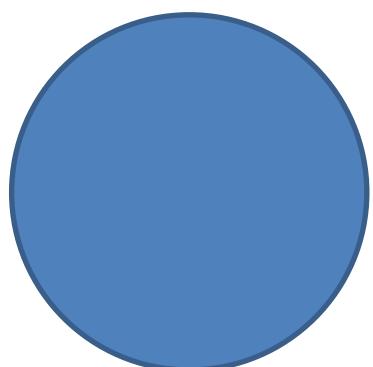
Embedding Matrix: Sesgo

El corpus y su uso puede incluir sesgos que pueden distorsionar los resultados, por ejemplo:

- Lenguaje sexista à hombre es a doctor lo que mujer es a enfermera.

¿Cómo reducir el sesgo?

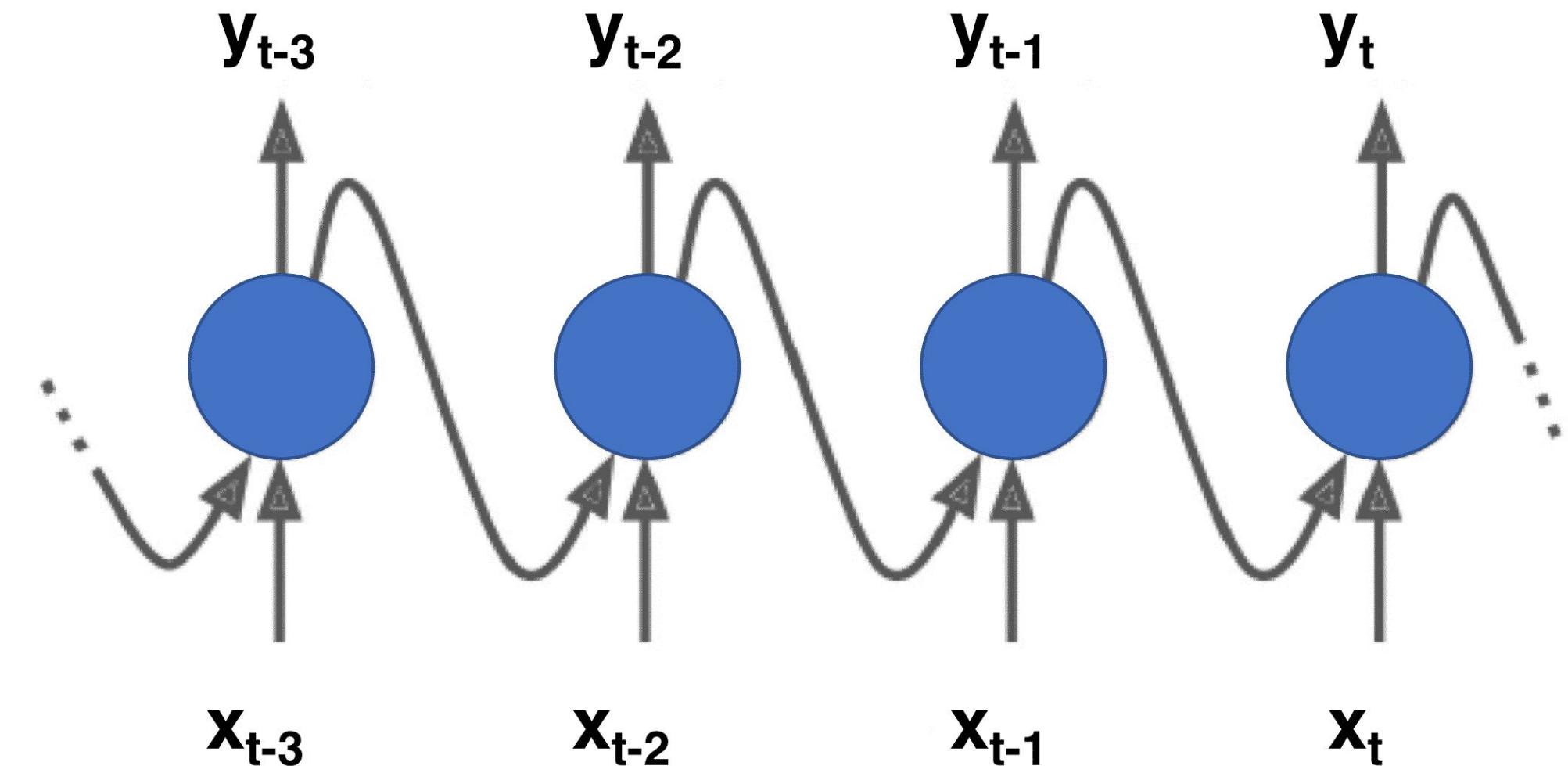
1. Identificar la dirección del sesgo.
2. Neutralizar el sesgo de aquellas palabras que no tengan que tener variación en dicha dirección.
3. Igualar pares que deberían tener la misma distancia en dicha dirección.





04

Redes de Neuronas Recurrentes

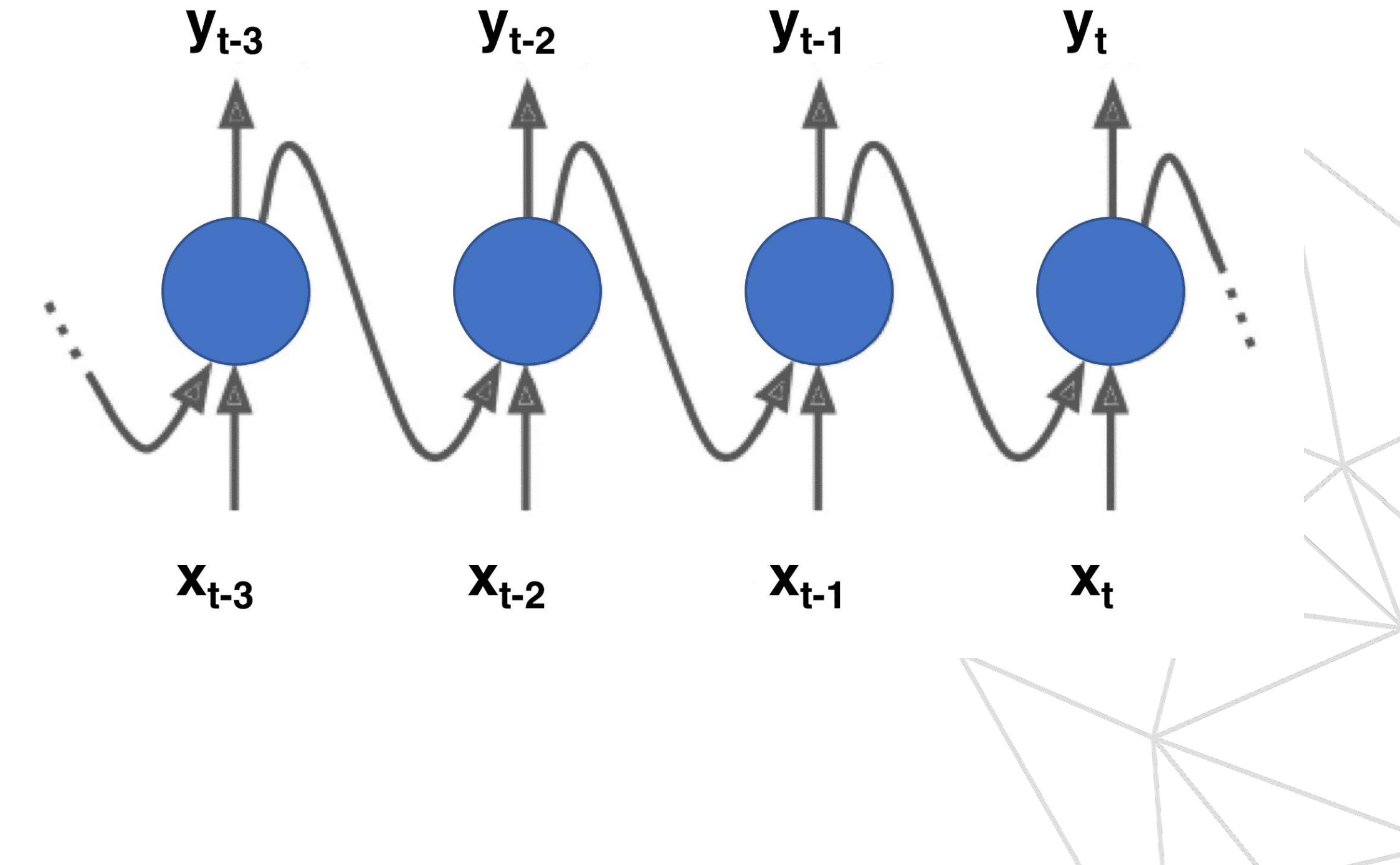




Introducción

Redes de neuronas orientadas al procesamiento de información secuencial:

- Reconocimiento de voz.
- Traducción.
- Análisis de sentimiento.
- Análisis de secuencias (ADN).
- Trigger words.
- Generación de música.





Introducción

Secuencias de datos de entrada y/o salida.

- Notación:

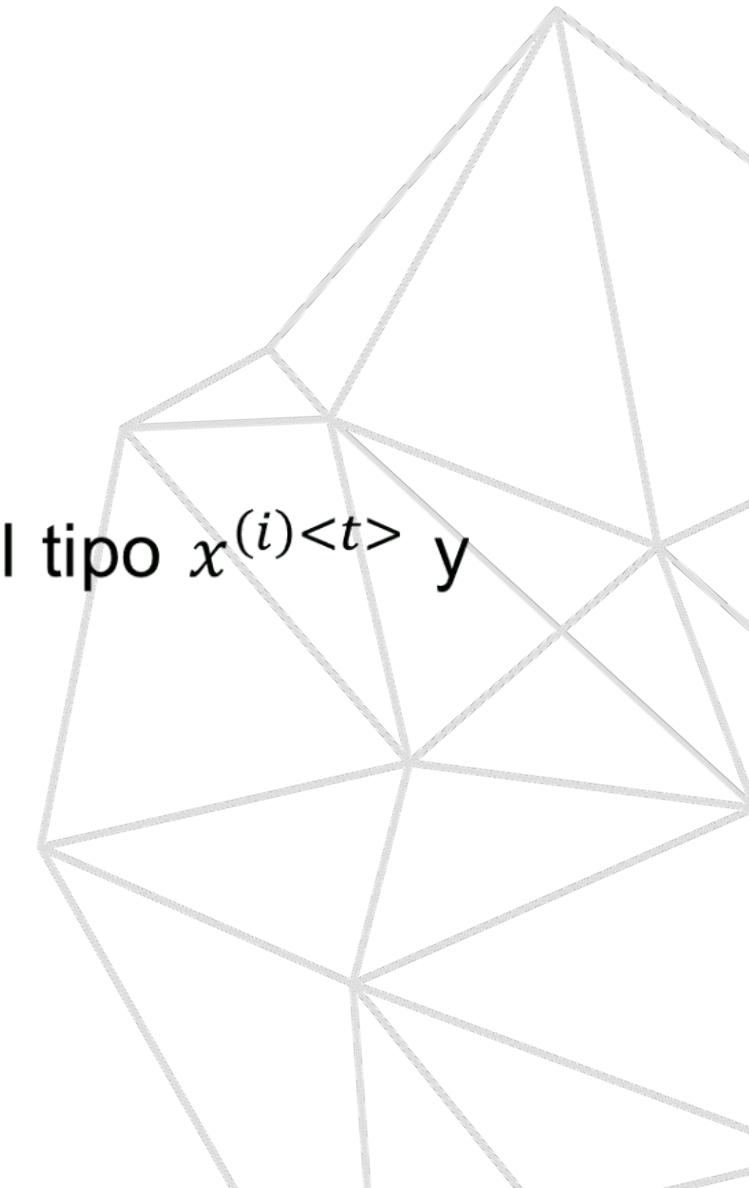
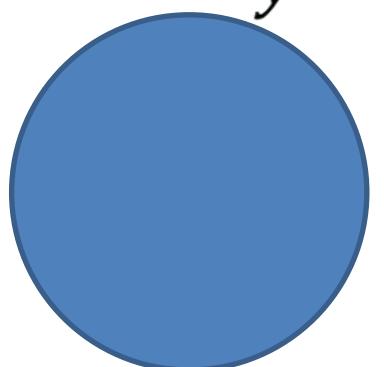
El	Perro	de	San	Roque	no	tiene	rabo
$x^{<1>}$	$x^{<2>}$	$x^{<3>}$	$x^{<4>}$	$x^{<5>}$	$x^{<6>}$	$x^{<7>}$	$x^{<8>}$
$y^{<1>}$	$y^{<2>}$	$y^{<3>}$	$y^{<4>}$	$y^{<5>}$	$y^{<6>}$	$y^{<7>}$	$y^{<8>}$

- Longitud de la secuencia:

$$Tx = 8$$

$$Ty = 8$$

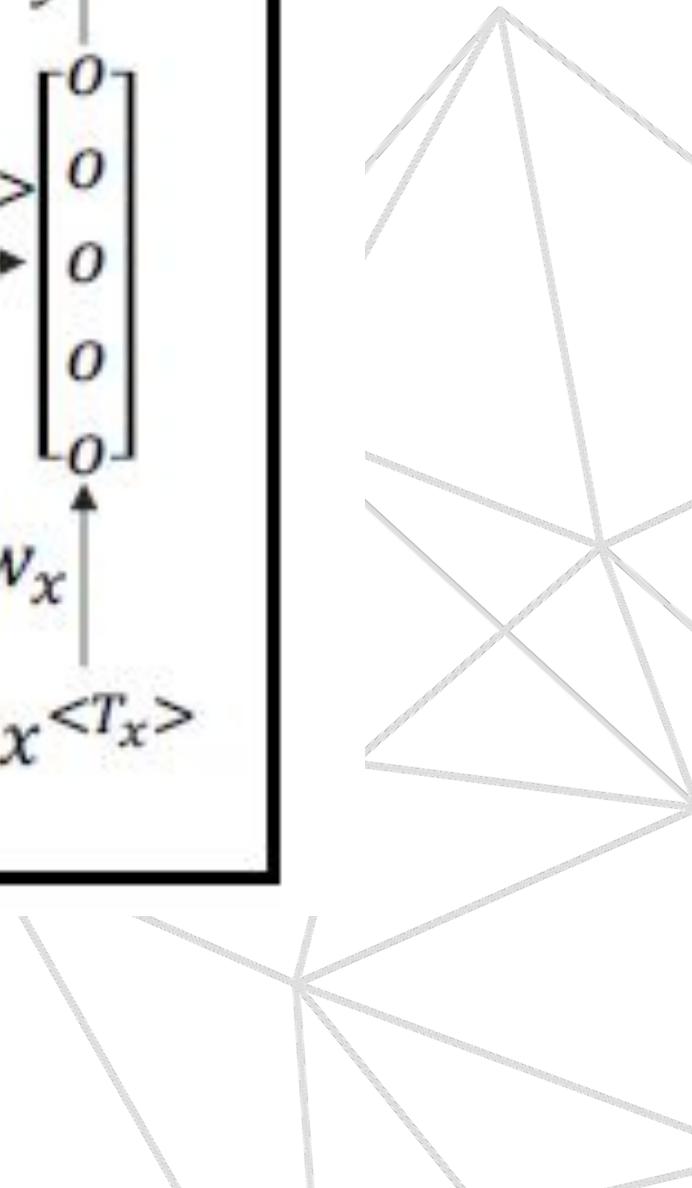
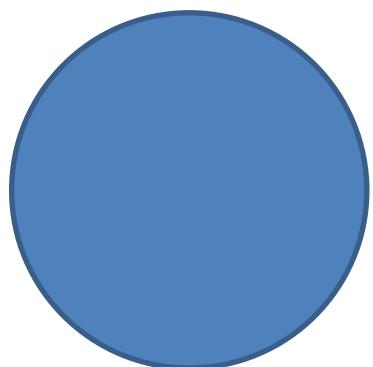
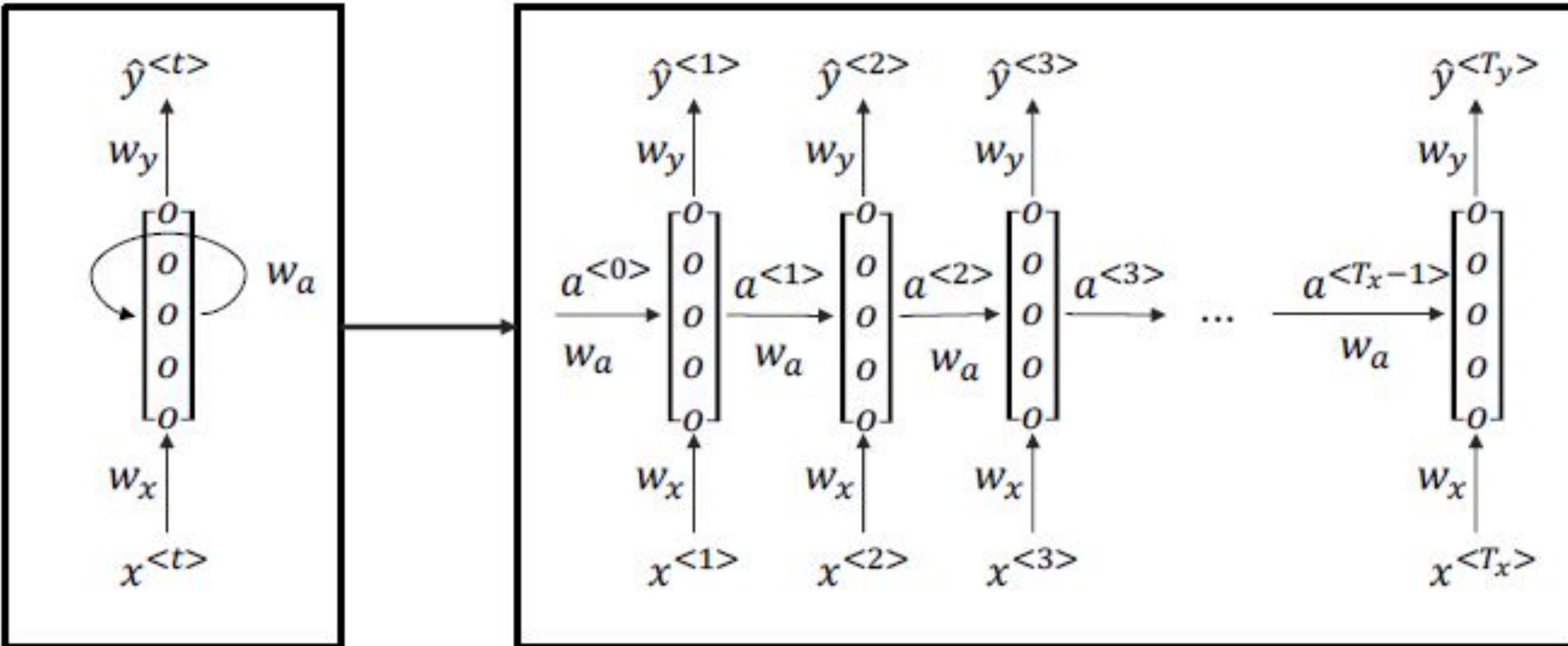
- Una tupla $(x^{(i)}, y^{(i)})$ con longitudes $Tx^{(i)}$ y $Ty^{(i)}$ está compuesta por elementos del tipo $x^{(i)<t>}$ y $y^{(i)<t>}.$





Introducción

RNN unidireccional





Introducción

Secuencias de datos de salida.

- Intuición (modelado de lenguaje):

El	Perro	de	San	Roque	no	tiene	rabo
$y^{<1>}$	$y^{<2>}$	$y^{<3>}$	$y^{<4>}$	$y^{<5>}$	$y^{<6>}$	$y^{<7>}$	$y^{<8>}$

- $\hat{y}^{<1>}$ representa una distribución de probabilidades.

$P(word^{<1>})$ para cualquier palabra $word$

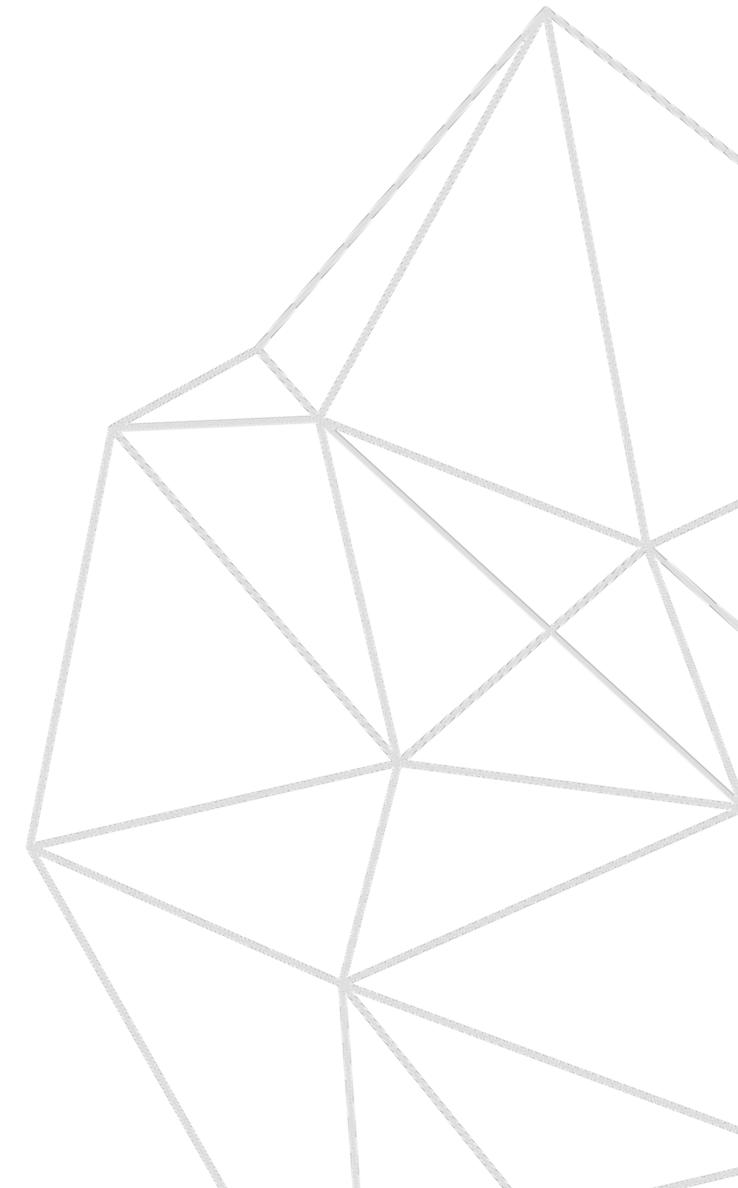
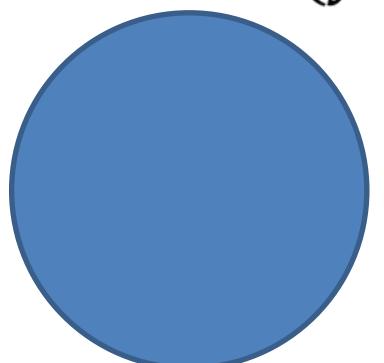
- $\hat{y}^{<2>} = P(word^{<2>}|word^{<1>})$

- $\hat{y}^{<3>} = P(word^{<3>}|word^{<1>}, word^{<2>})$

- ...

- $P(y^{<1>}, y^{<2>}, \dots, y^{<8>}) =$

$$\begin{aligned} & P(y^{<1>}) \\ & P(y^{<2>}|y^{<1>}) \\ & P(y^{<3>}|y^{<1>}, y^{<2>}) \dots \end{aligned}$$



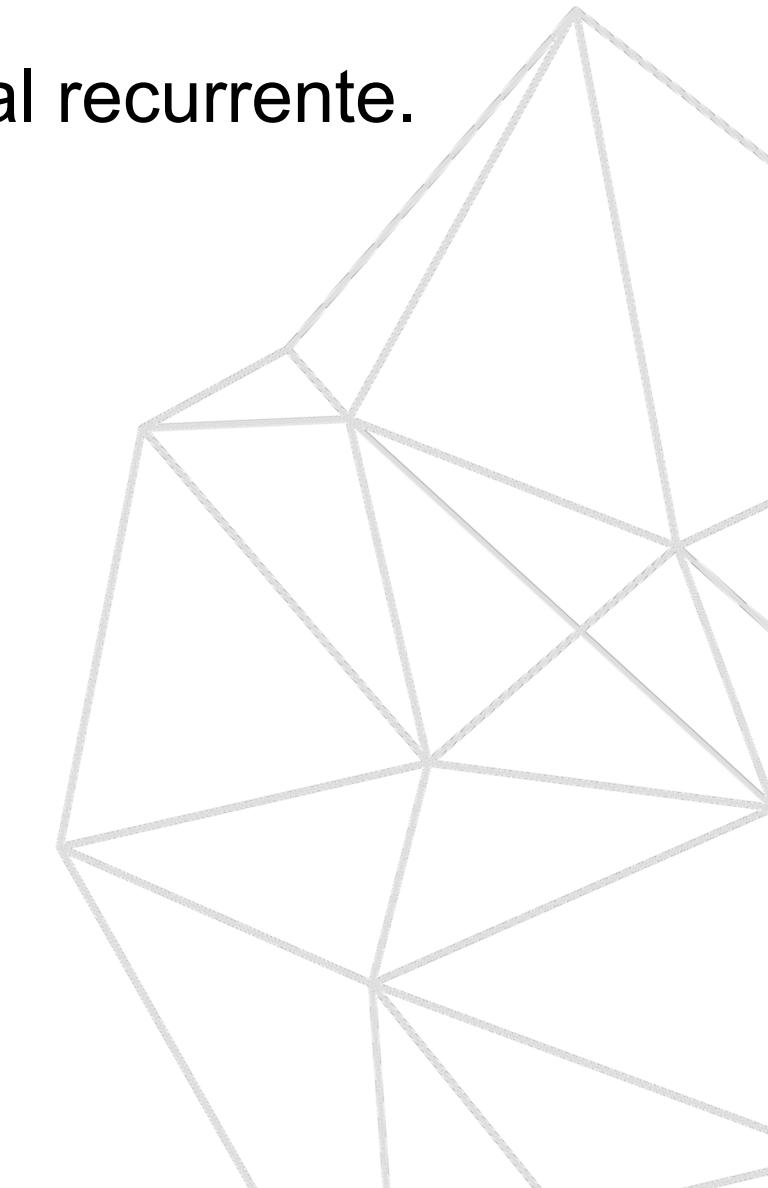
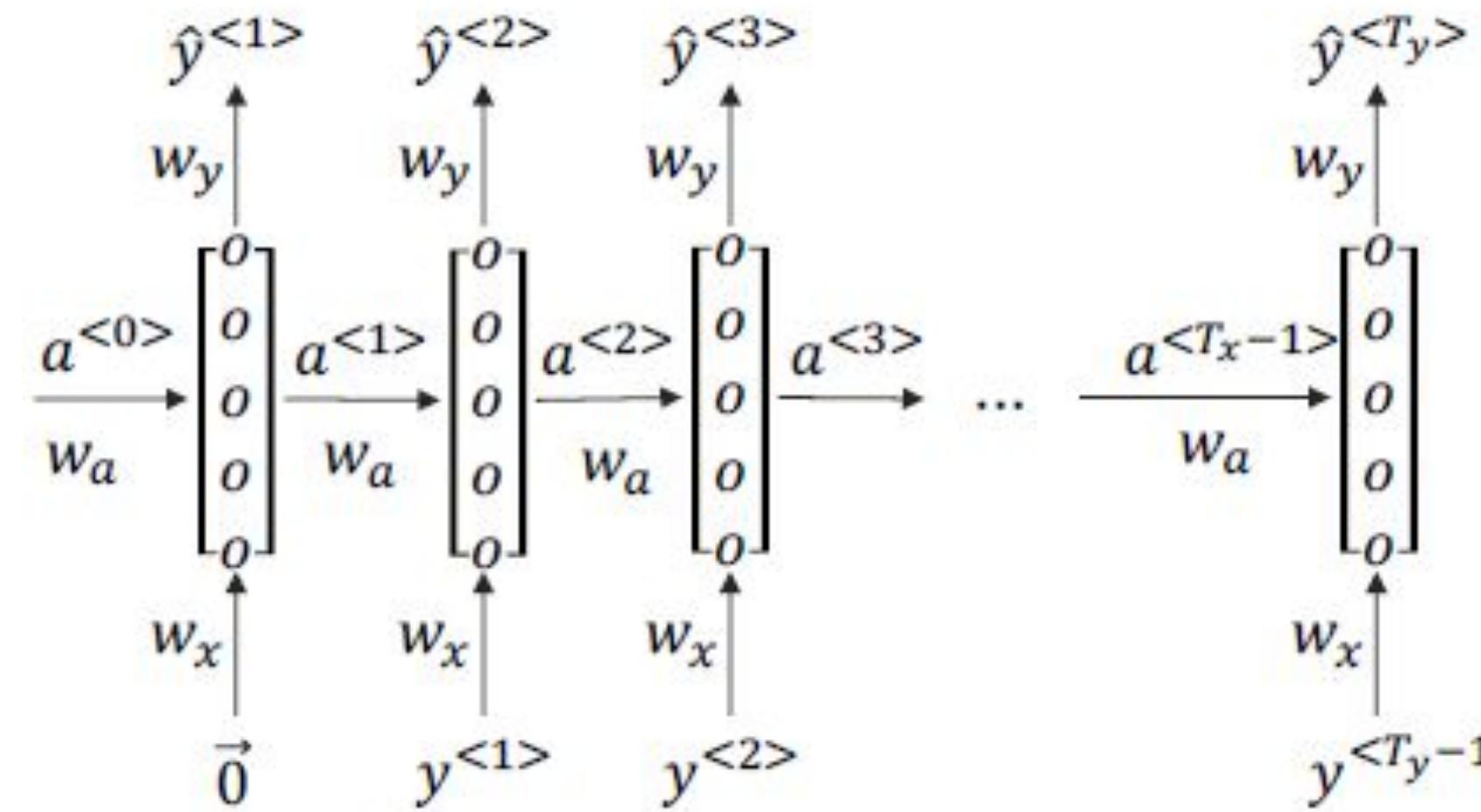


Introducción

RNN unidireccional

El	Perro	de	San	Roque	no	tiene	rabo
$y^{<1>}$	$y^{<2>}$	$y^{<3>}$	$y^{<4>}$	$y^{<5>}$	$y^{<6>}$	$y^{<7>}$	$y^{<8>}$

- Podemos generar frases mediante el muestreo de las distribución de la red neuronal recurrente.

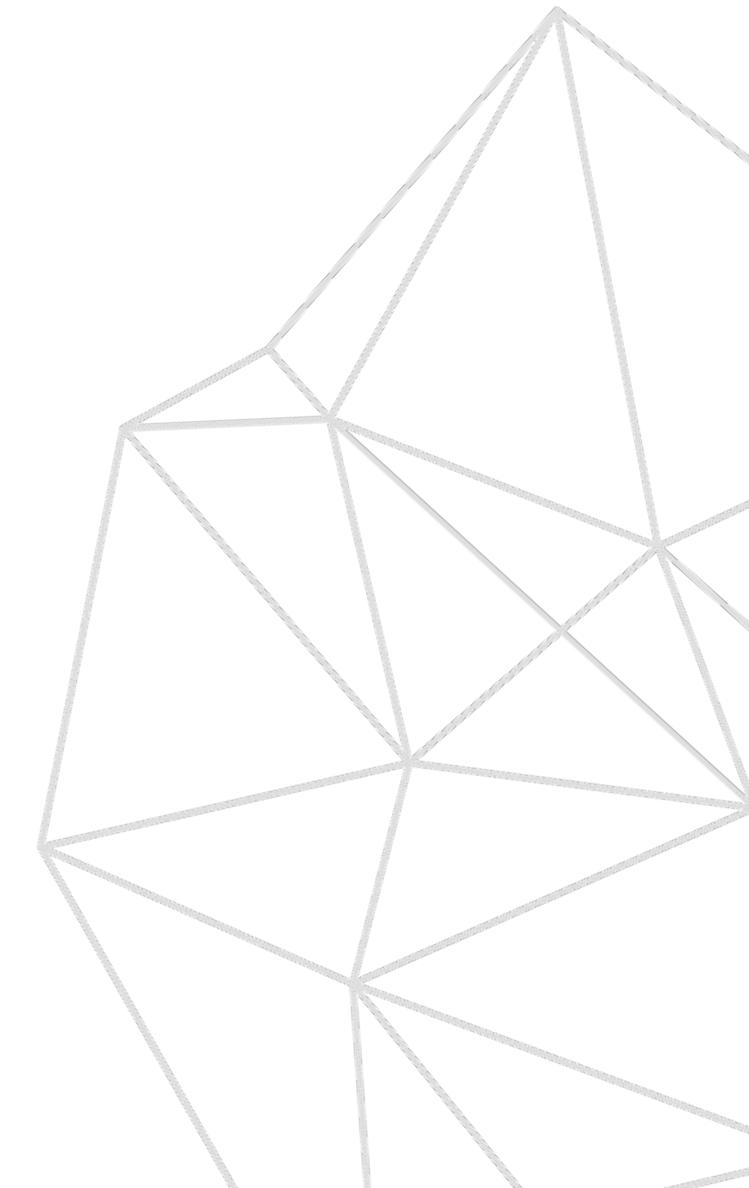
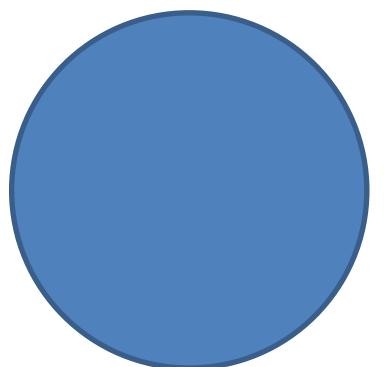




Introducción

RNN unidireccional

- FeedForward
 - $a^{<t>} = g_a(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$
 $= g_a(W_a[a^{<t-1>}, x^{<t>}] + b_a)$
 - $\hat{y}^{<t>} = g_y(W_ya^{<t>} + b_y)$
- Donde
 - g_a : tanh o ReLU (preferiblemente tanh, ReLU es bastante inestable).
 - g_y : sigmoidea o softmax
 - $a^{<0>} = \vec{0}$





Introducción

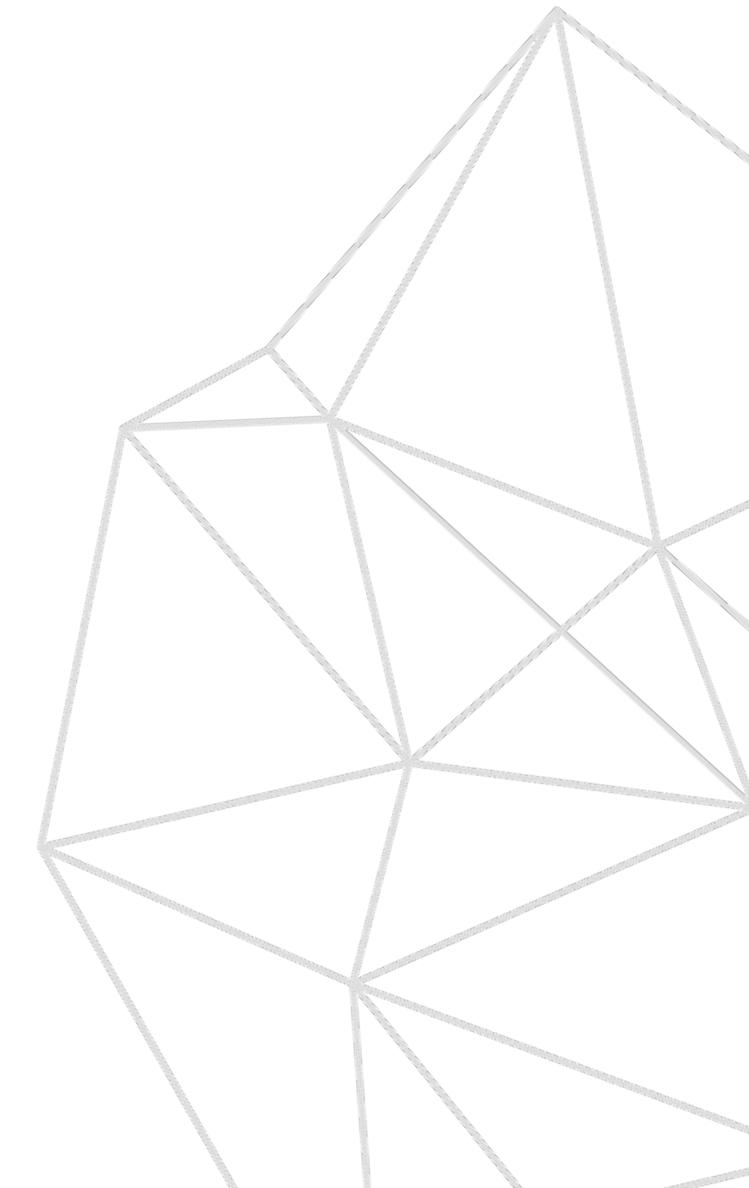
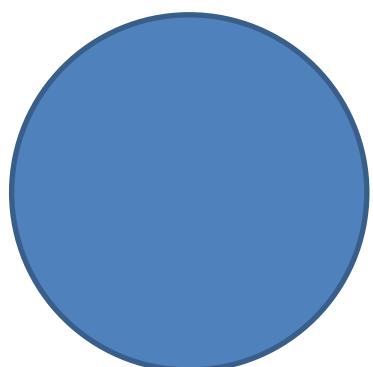
RNN unidireccional

- Backpropagation a través del tiempo
 - Función de pérdida para el momento t para función de activación sigmoidea:

$$\begin{aligned}\mathcal{L}^{(t)}(\hat{y}^{(t)}, y^{(t)}) = & -y^{(t)} \log \hat{y}^{(t)} \\ & -(1 - y^{(t)}) \log(1 - \hat{y}^{(t)})\end{aligned}$$

- Función de perdida para una tupla

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}^{(t)}(\hat{y}^{(t)}, y^{(t)})$$





Introducción

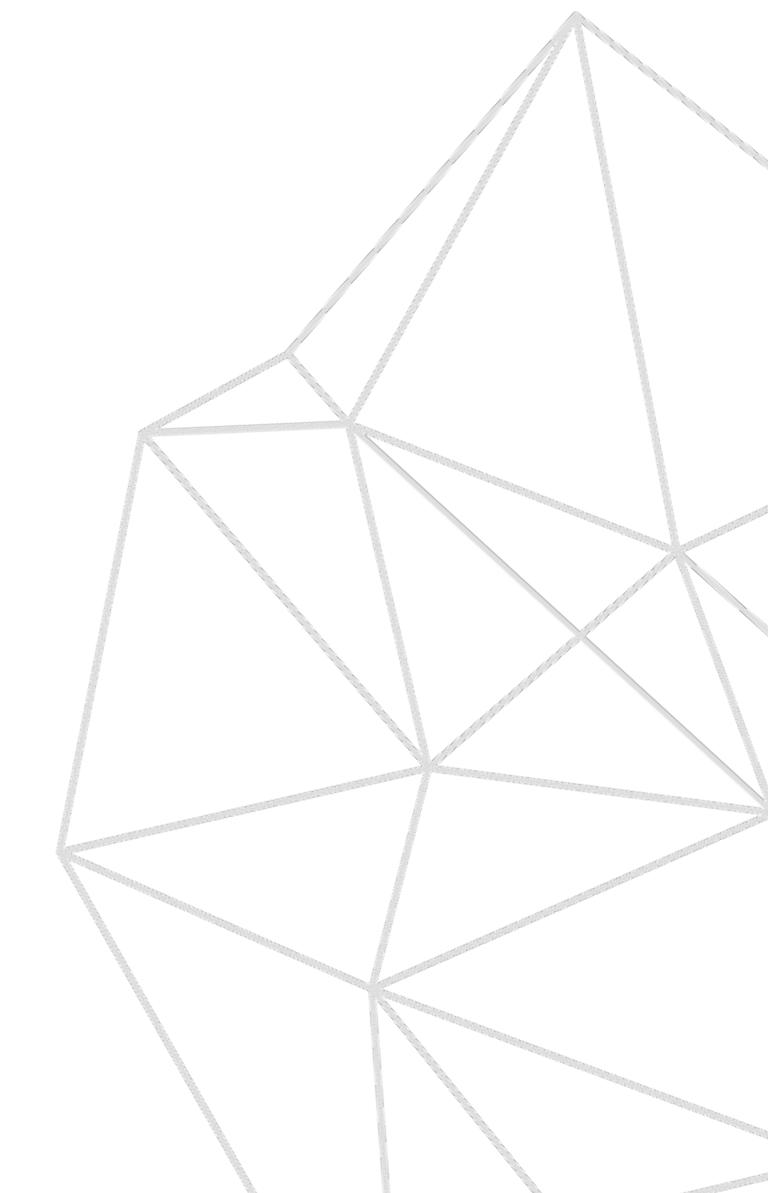
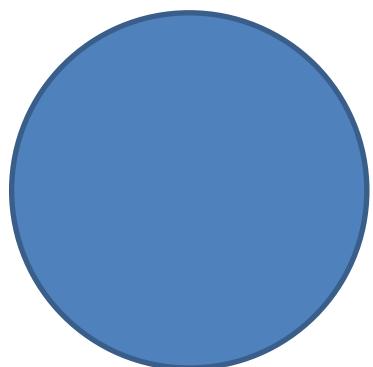
RNN unidireccional

- Backpropagation a través del tiempo
 - Función de pérdida para el momento $< t >$ para función de activación softmax:

$$\mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) = - \sum_{i=1}^m y_i^{<t>} \log \hat{y}_i^{<t>}$$

- Función de perdida para una tupla

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

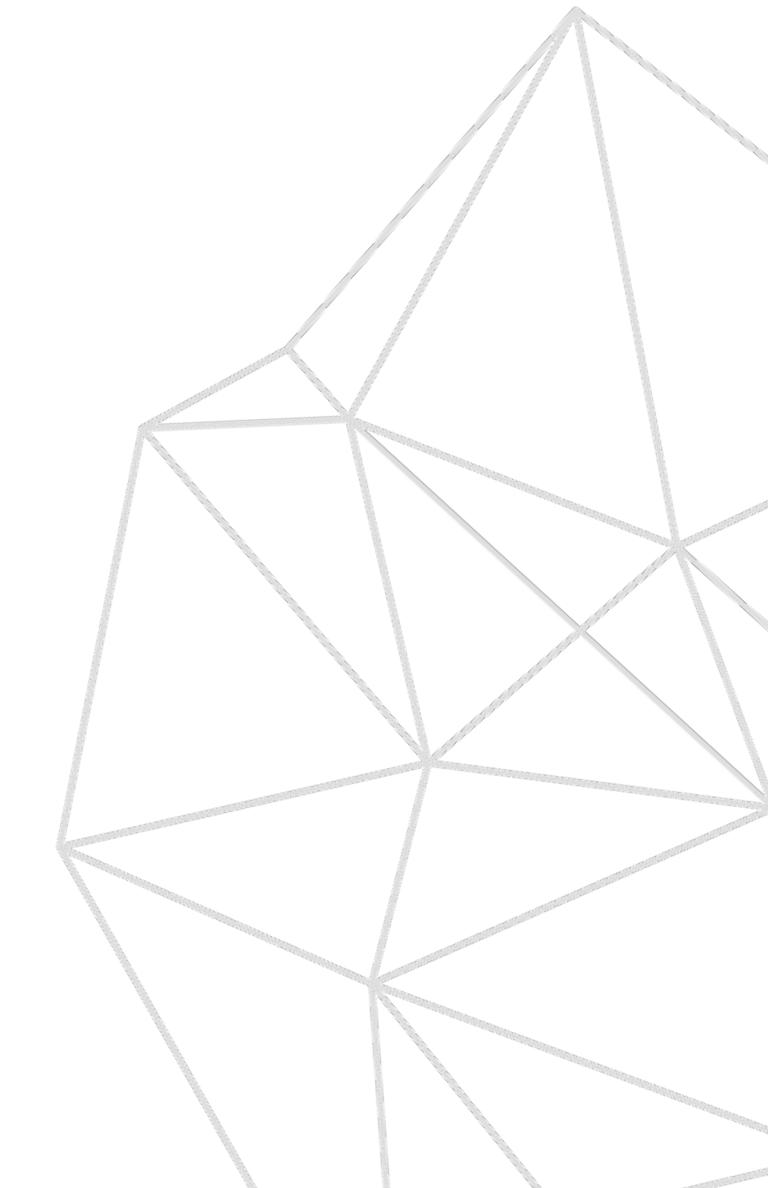
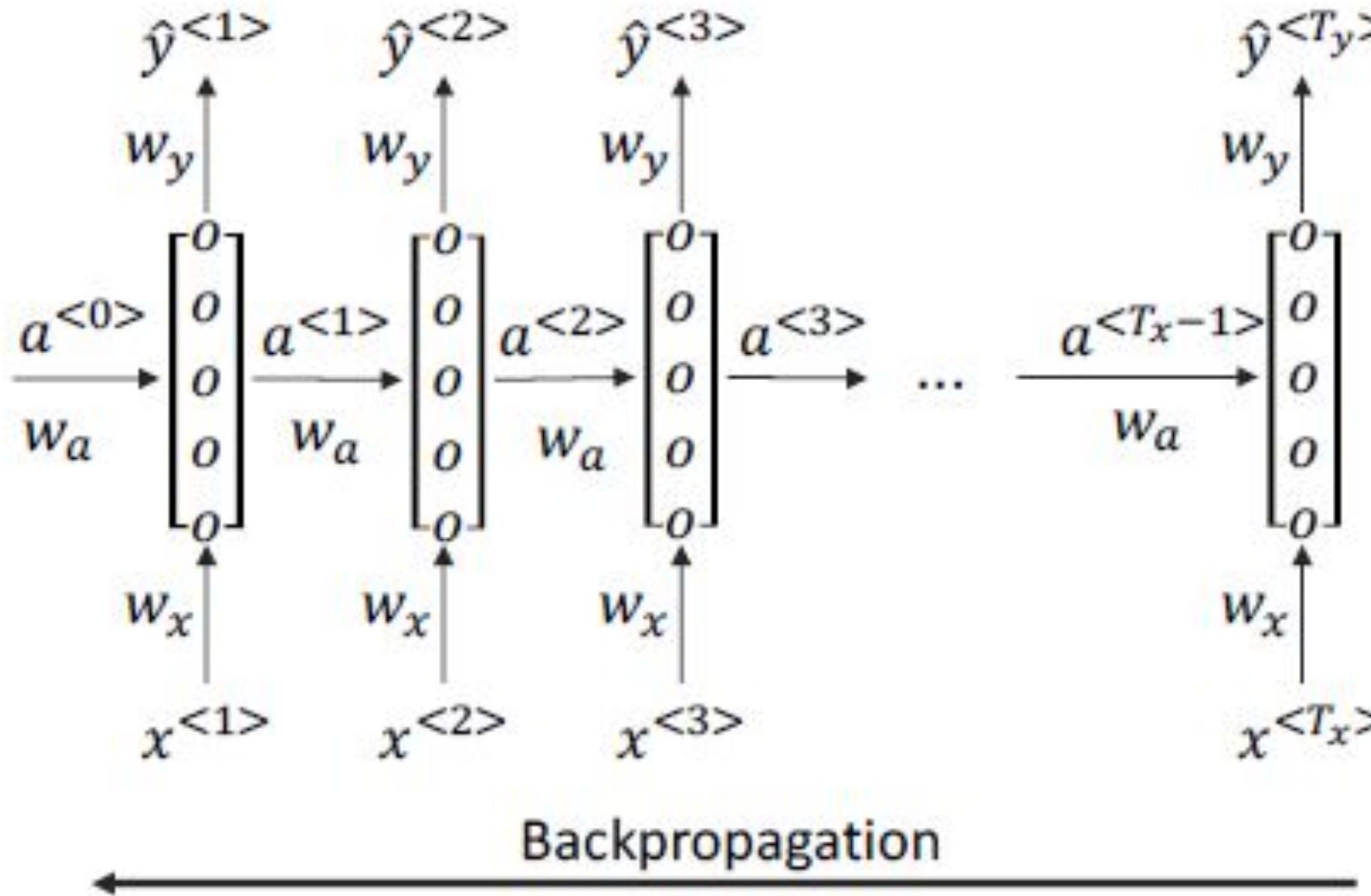




Introducción

RNN unidireccional

- Backpropagation a través del tiempo

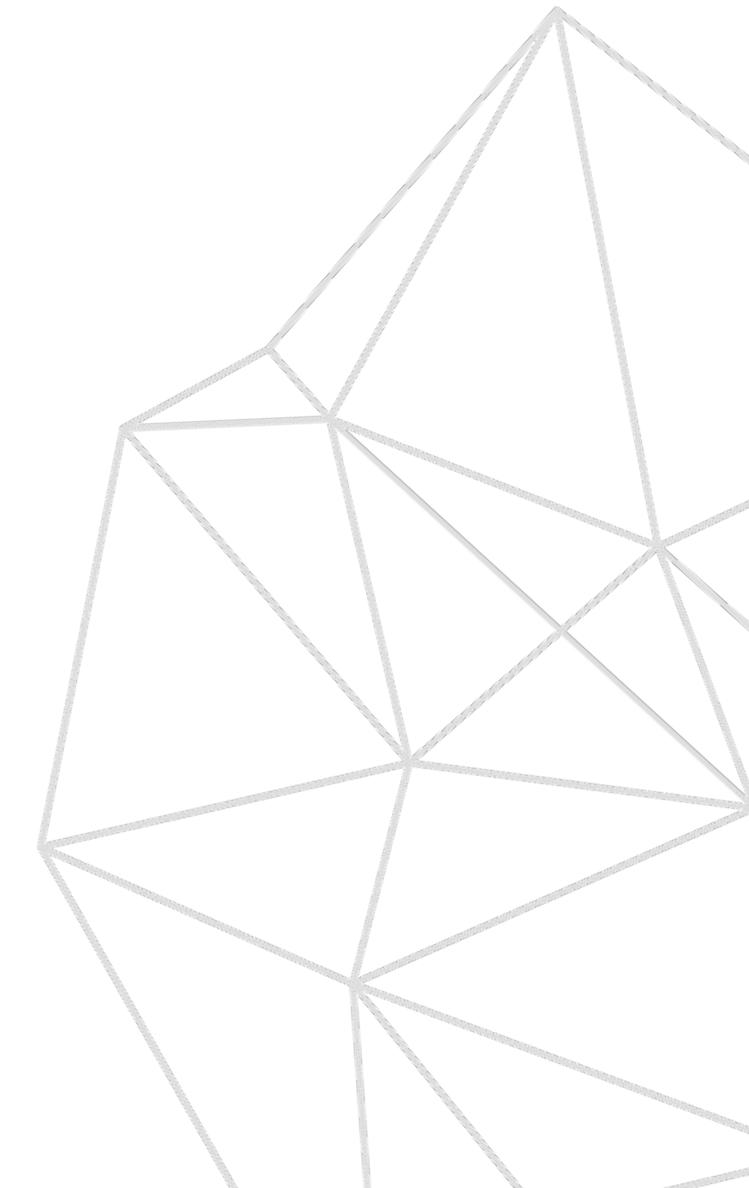
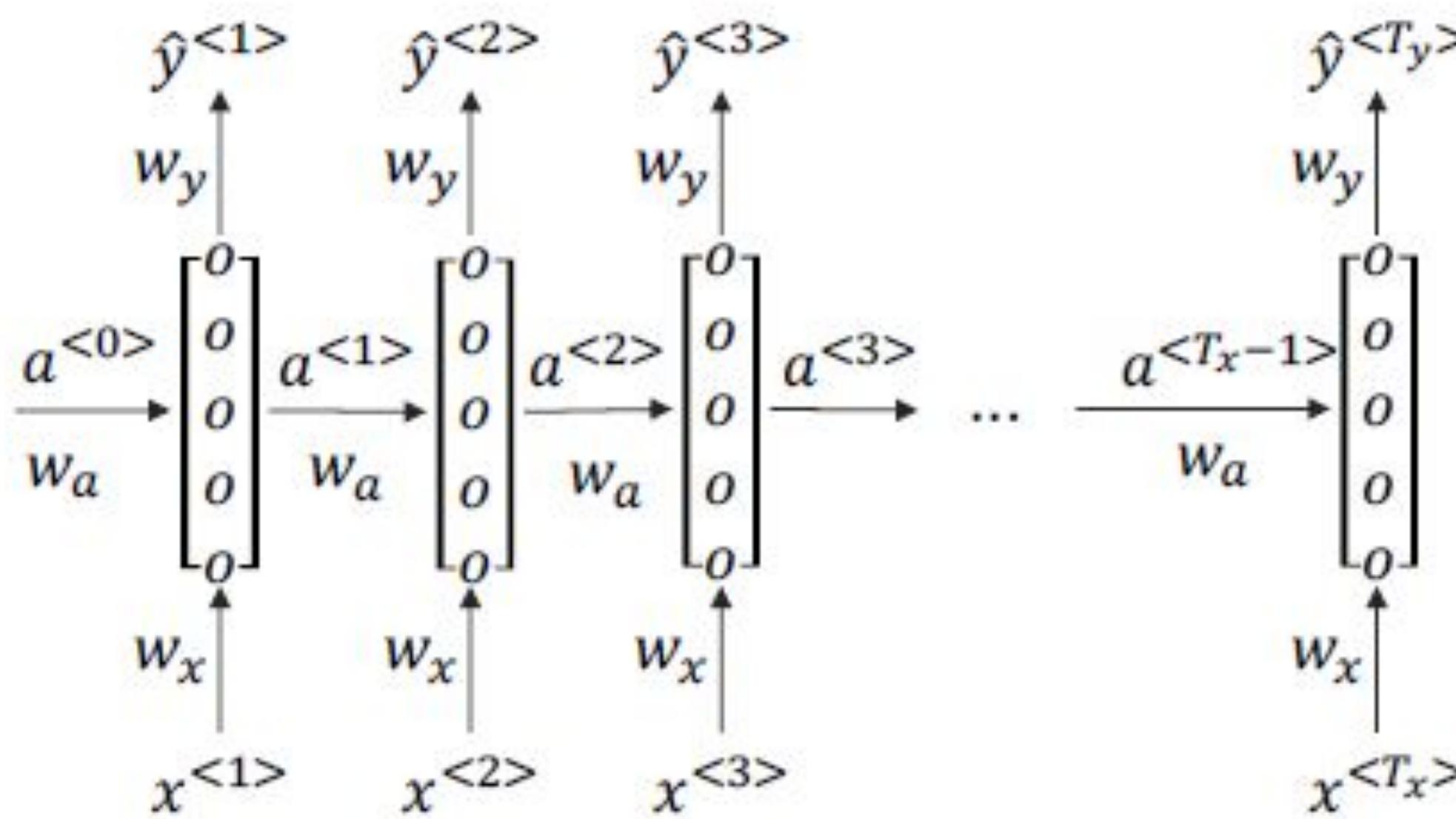




Tipos de RNN

N:N

- Por ejemplo, Trigger words

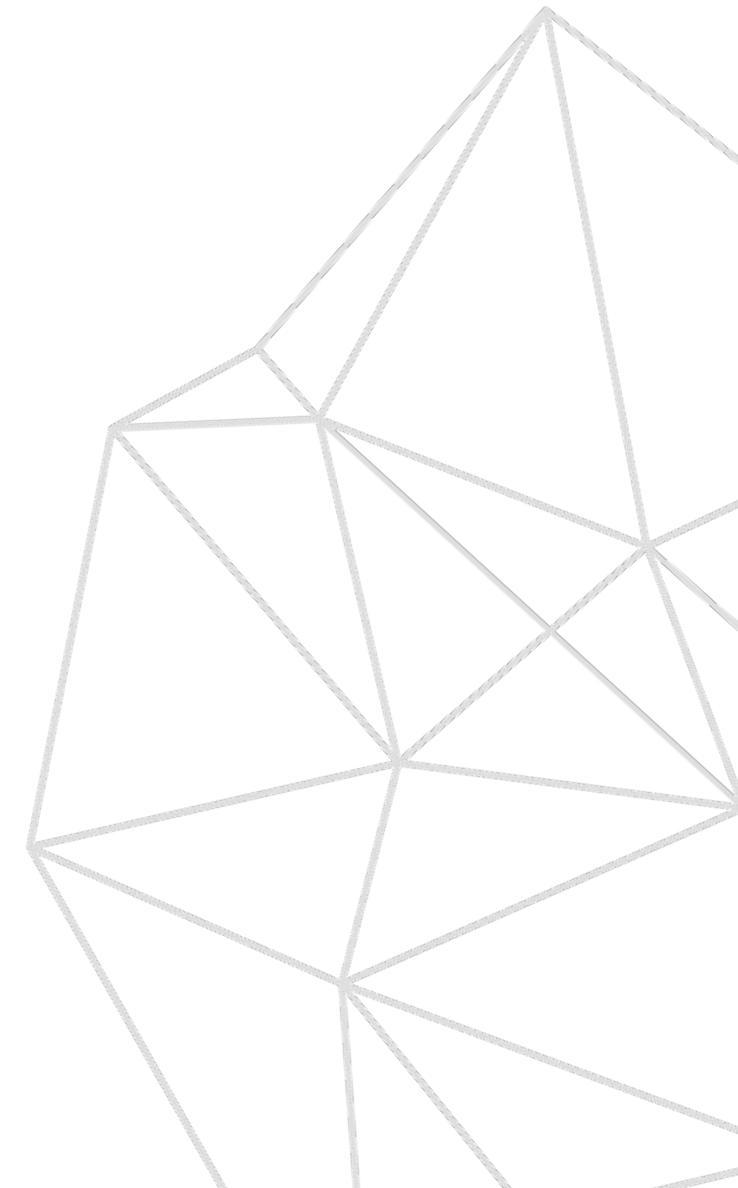
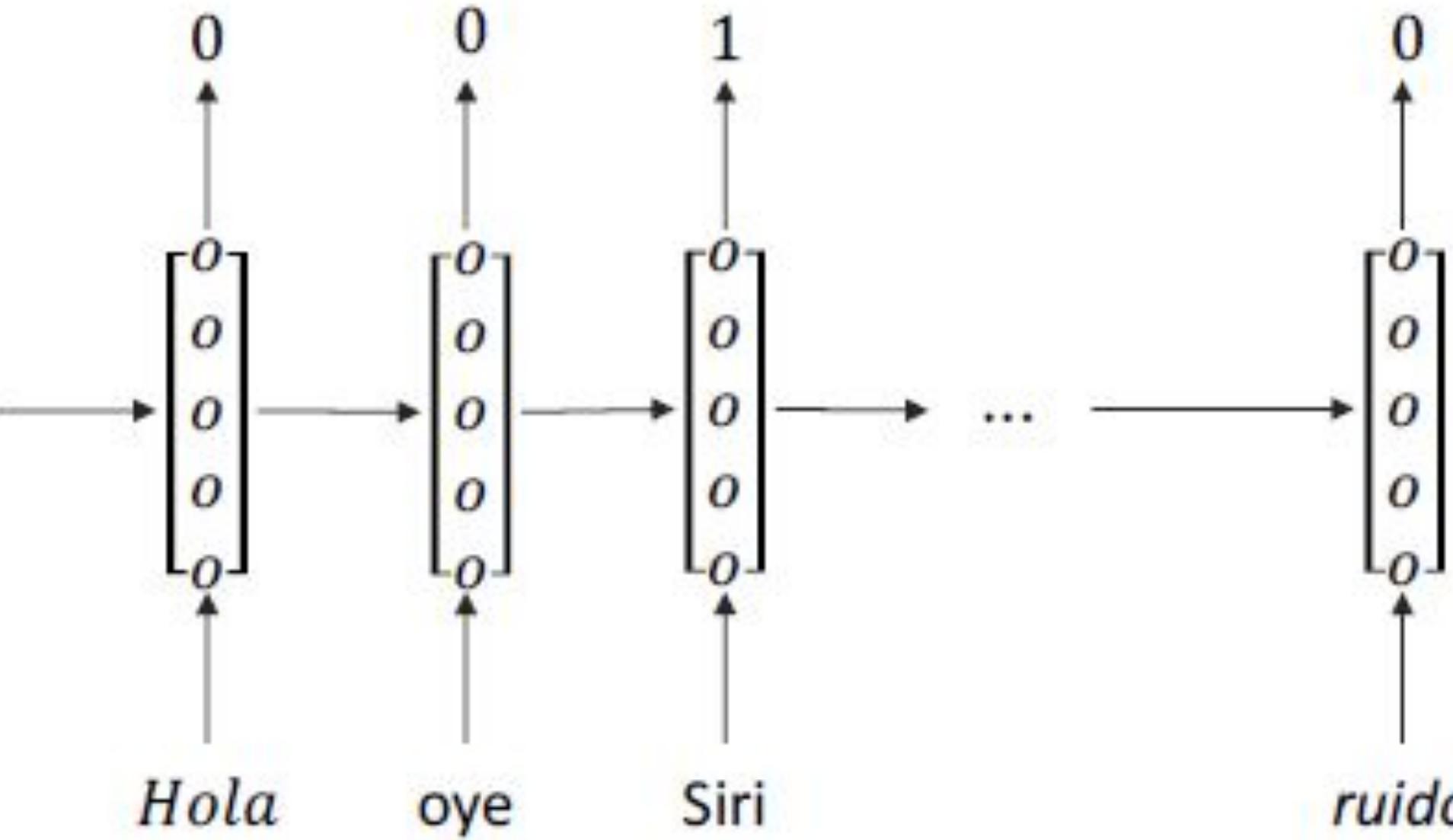




Tipos de RNN

N:N

- Por ejemplo, Trigger words

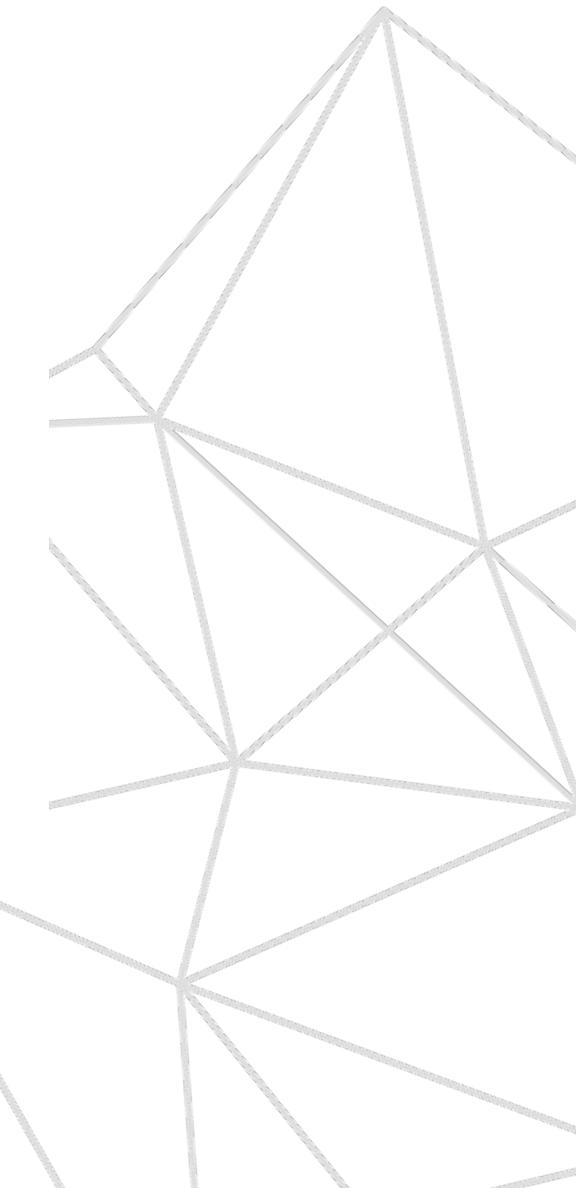
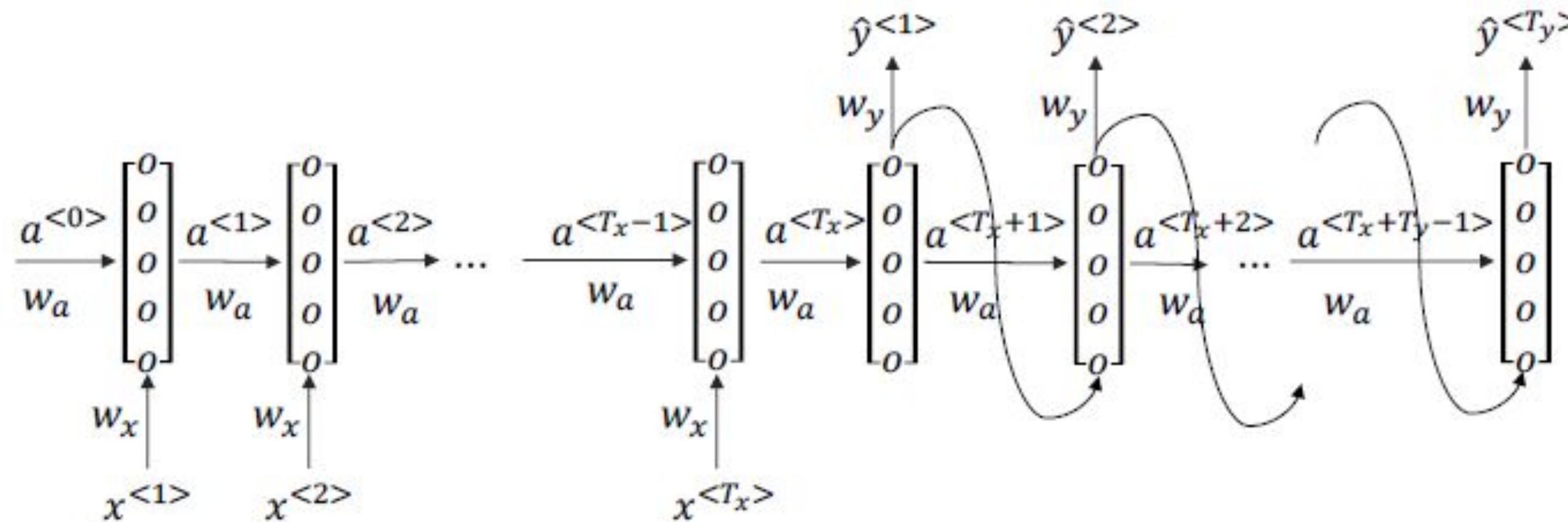




Tipos de RNN

N:N (encoder-decoder)

- Por ejemplo, traducción

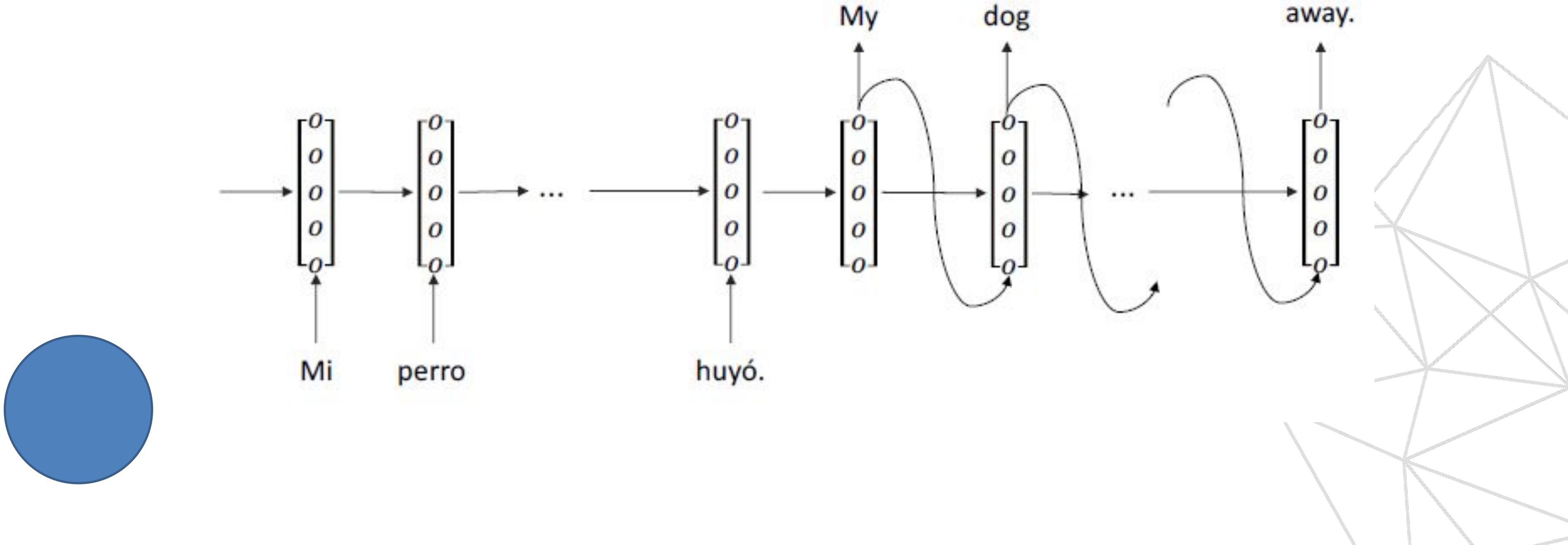




Tipos de RNN

N:N (encoder-decoder)

- Por ejemplo, traducción

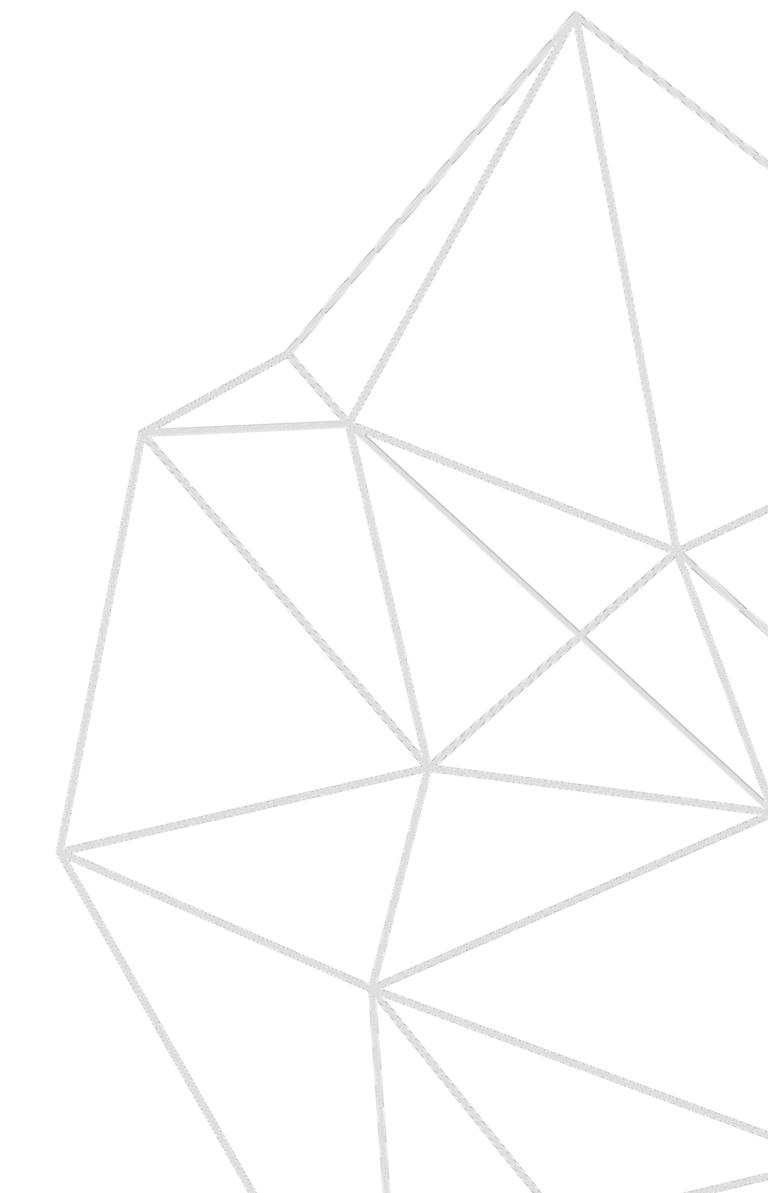
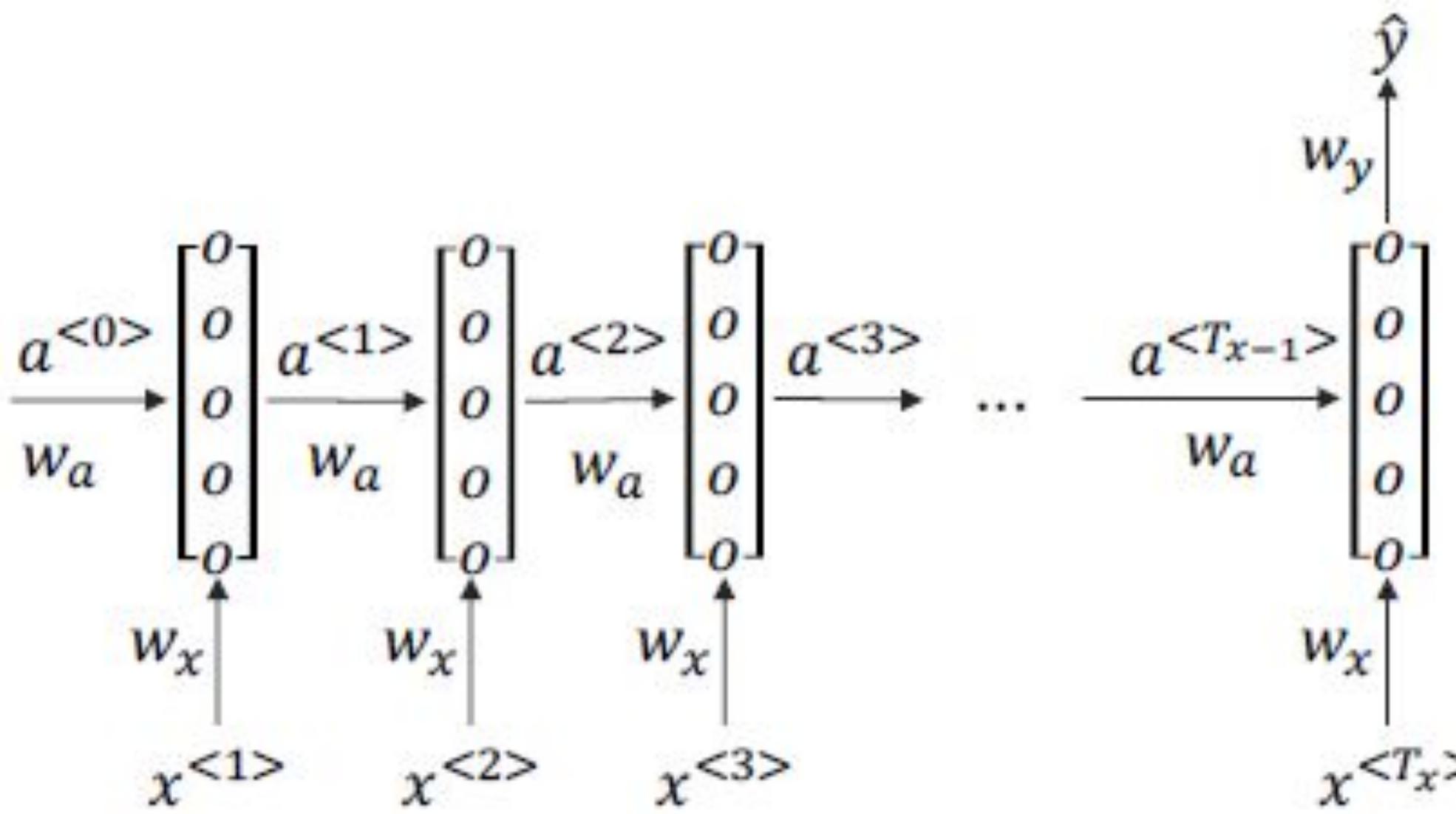




Tipos de RNN

N:I

- Por ejemplo, análisis de sentimiento

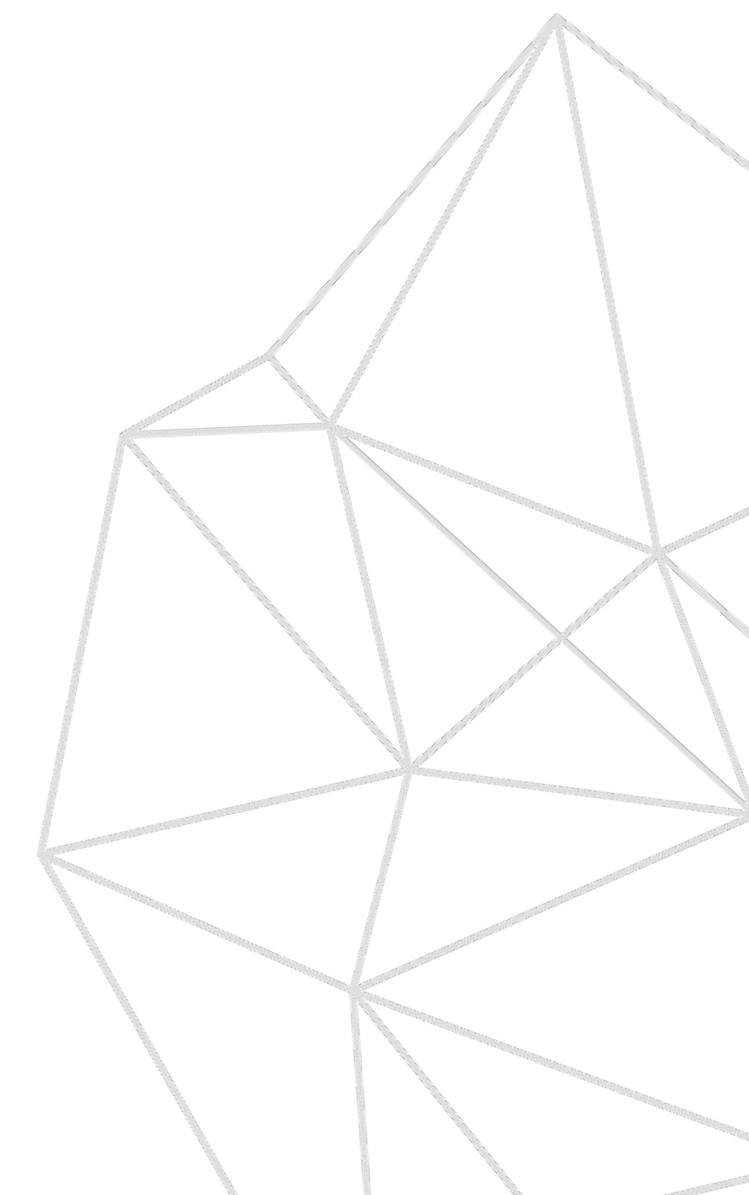
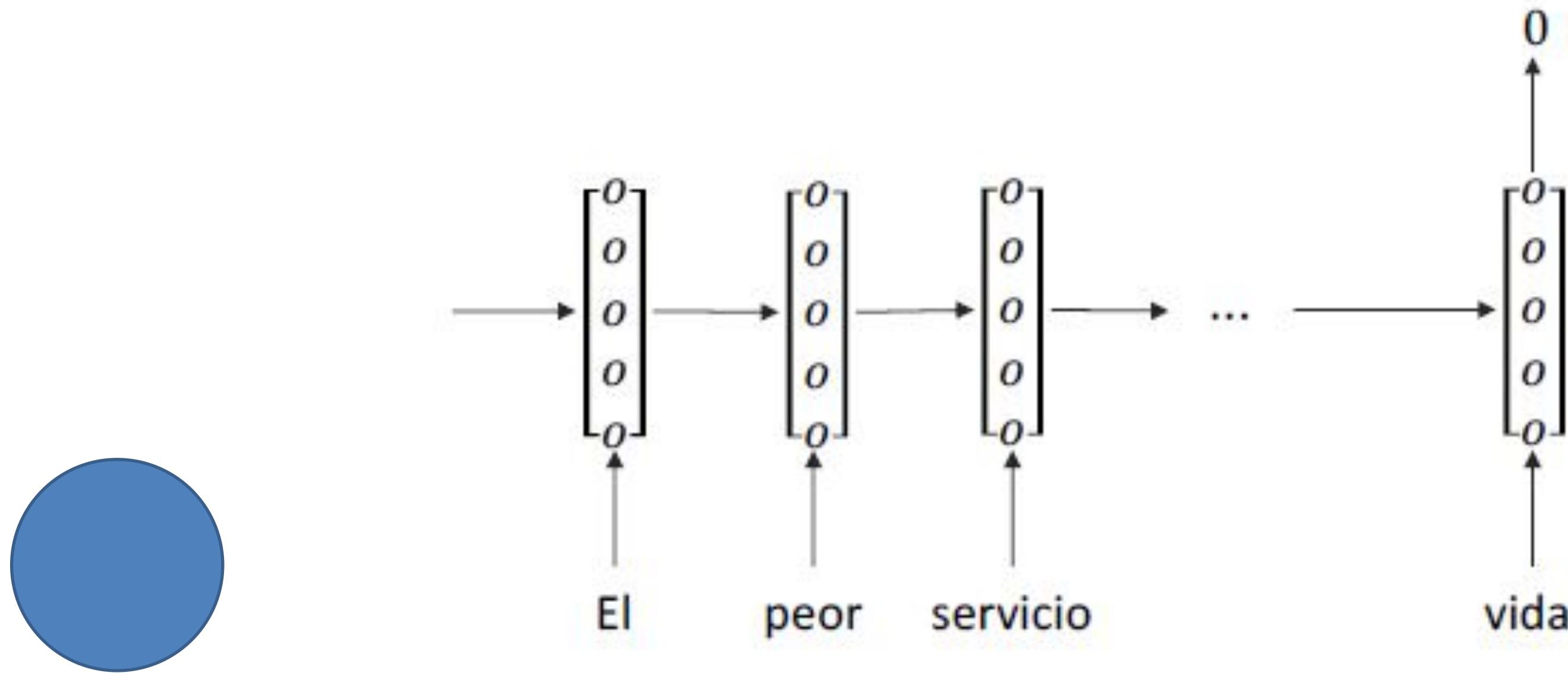




Tipos de RNN

N:I

- Por ejemplo, análisis de sentimiento

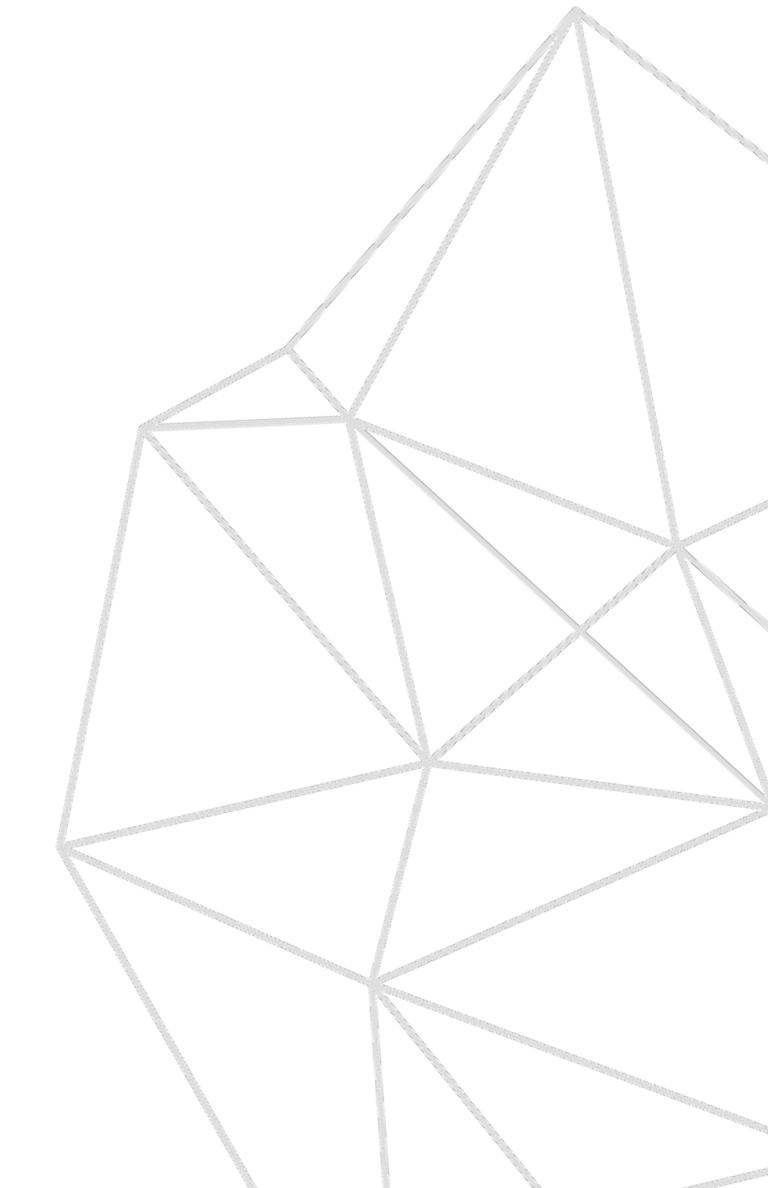
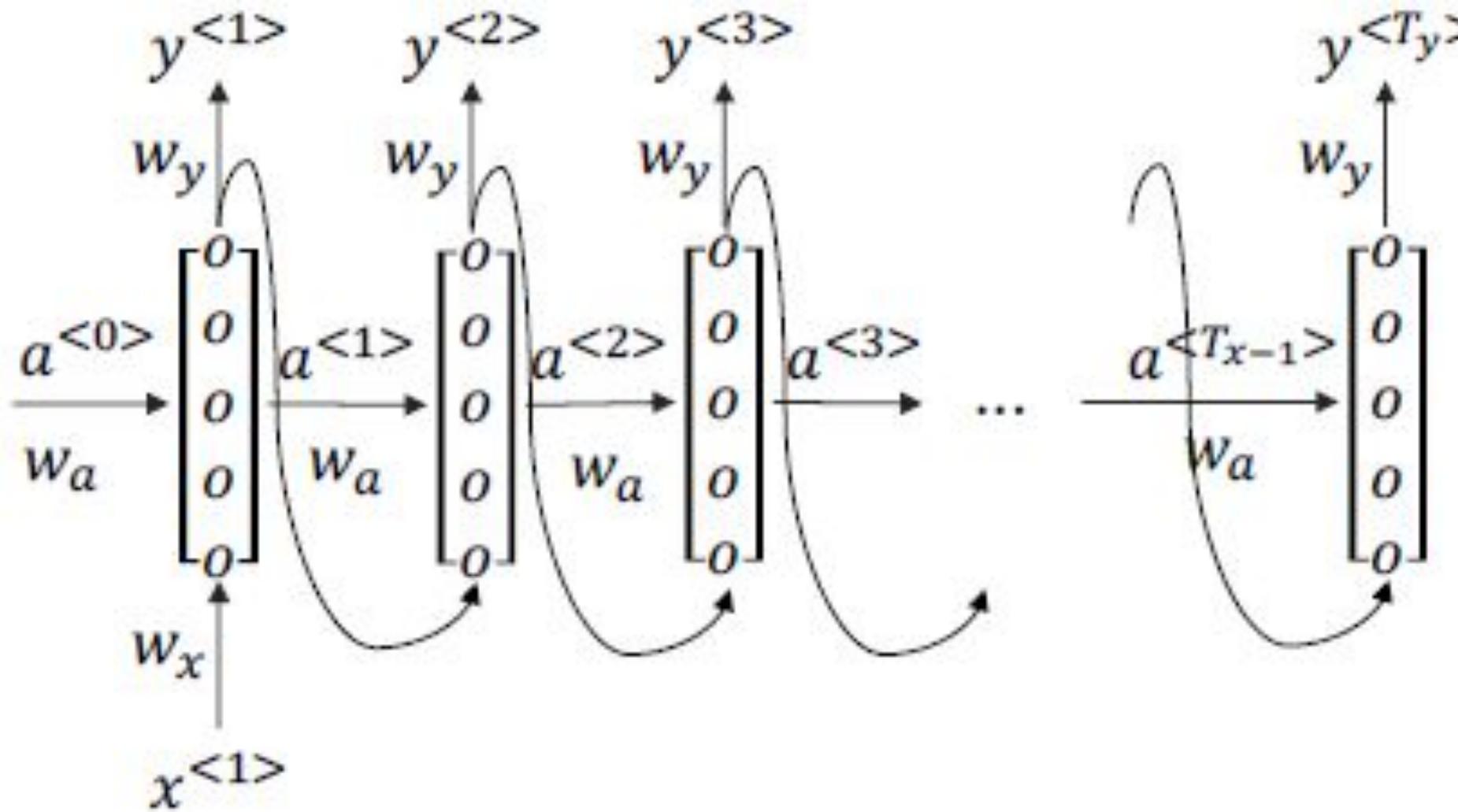




Tipos de RNN

I:N

- Por ejemplo, generación de texto/música

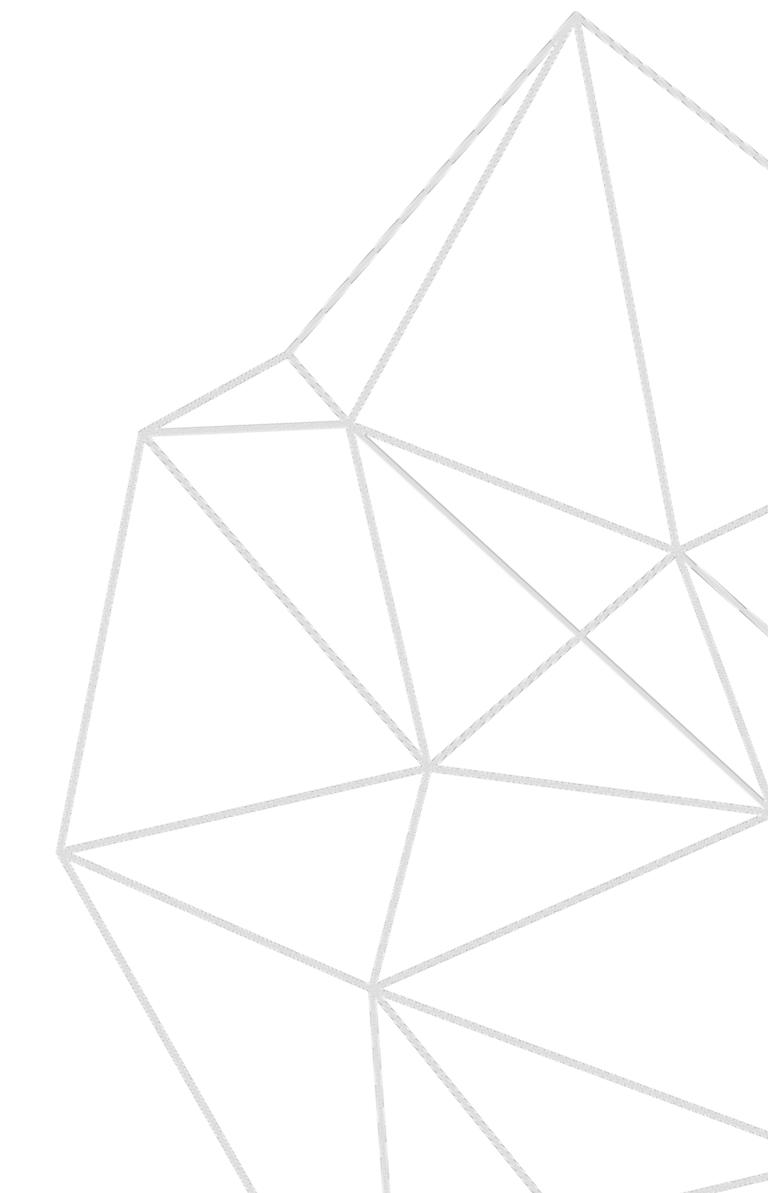
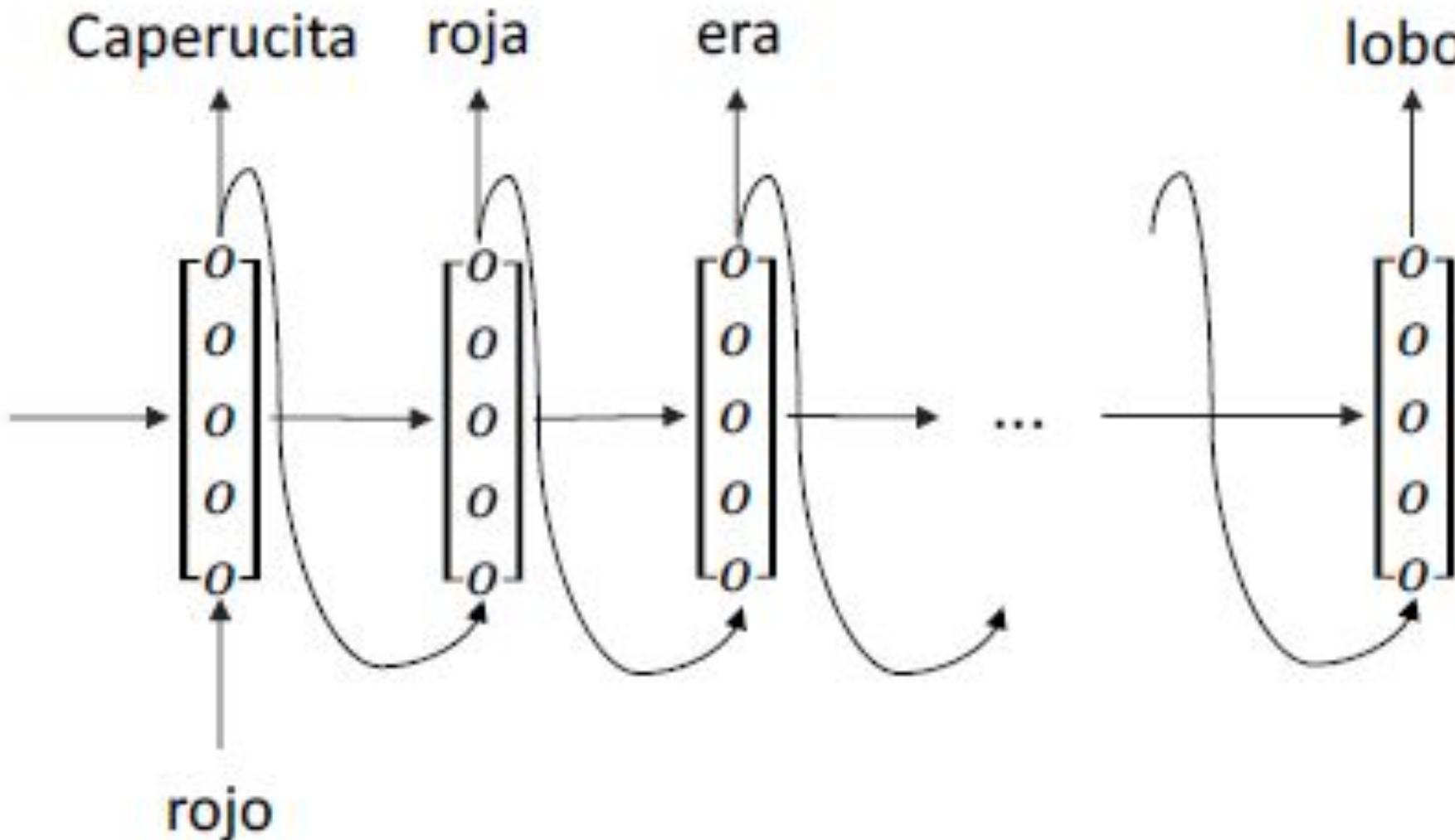




Tipos de RNN

I:N

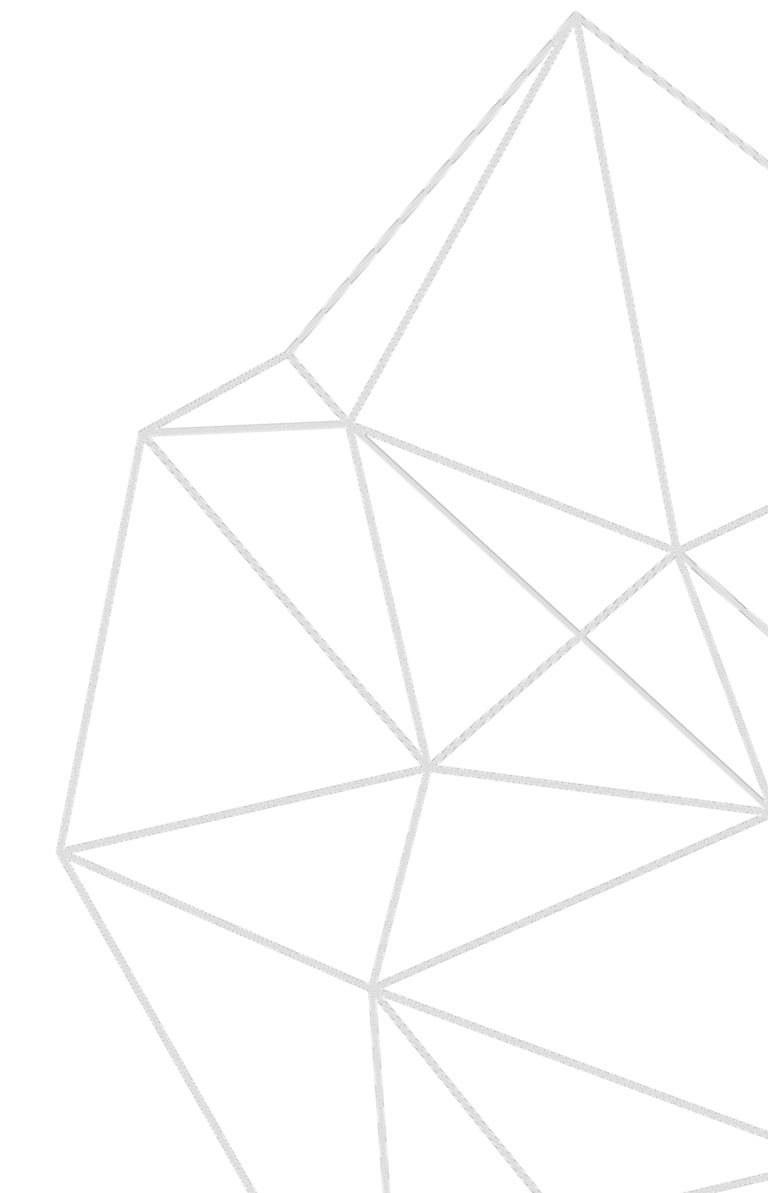
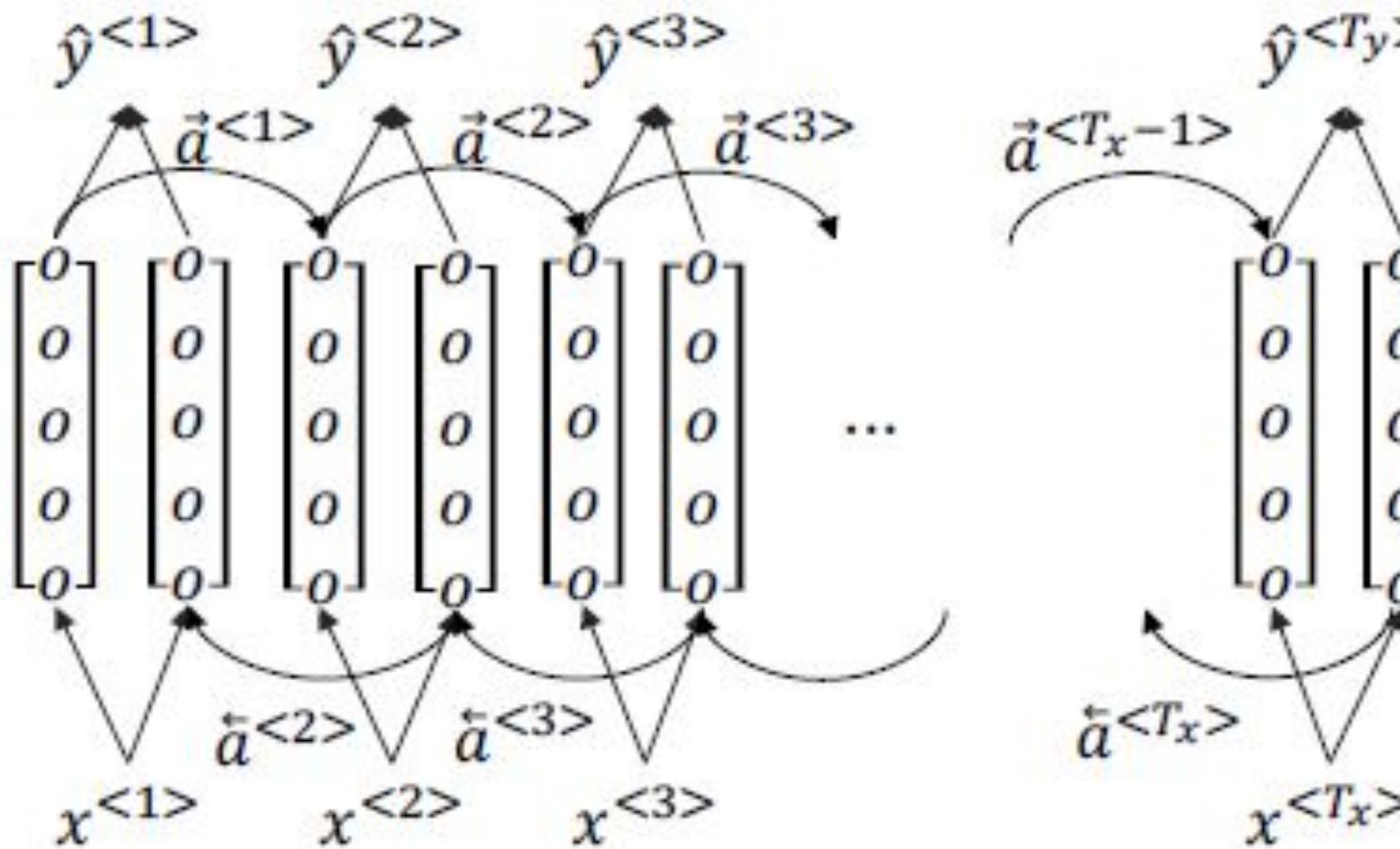
- Por ejemplo, generación de texto/música



Tipos de RNN: Bidireccionales

Utiliza tanto la formación del pasado como del futuro

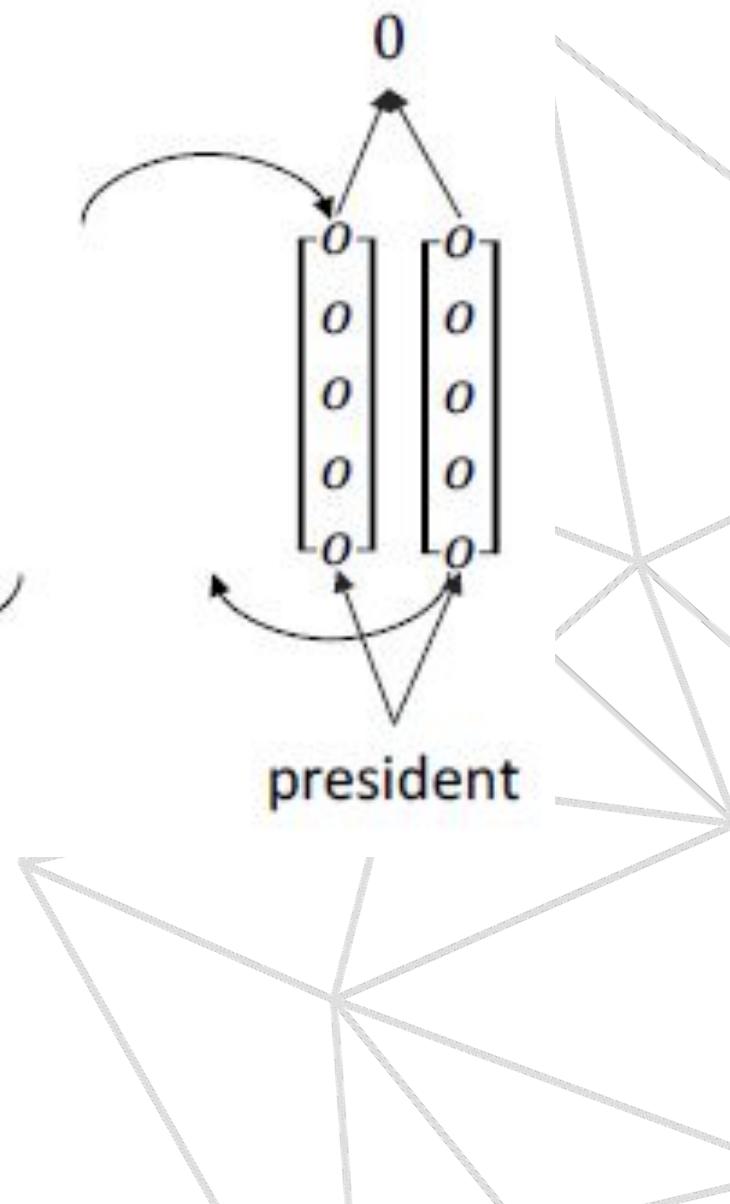
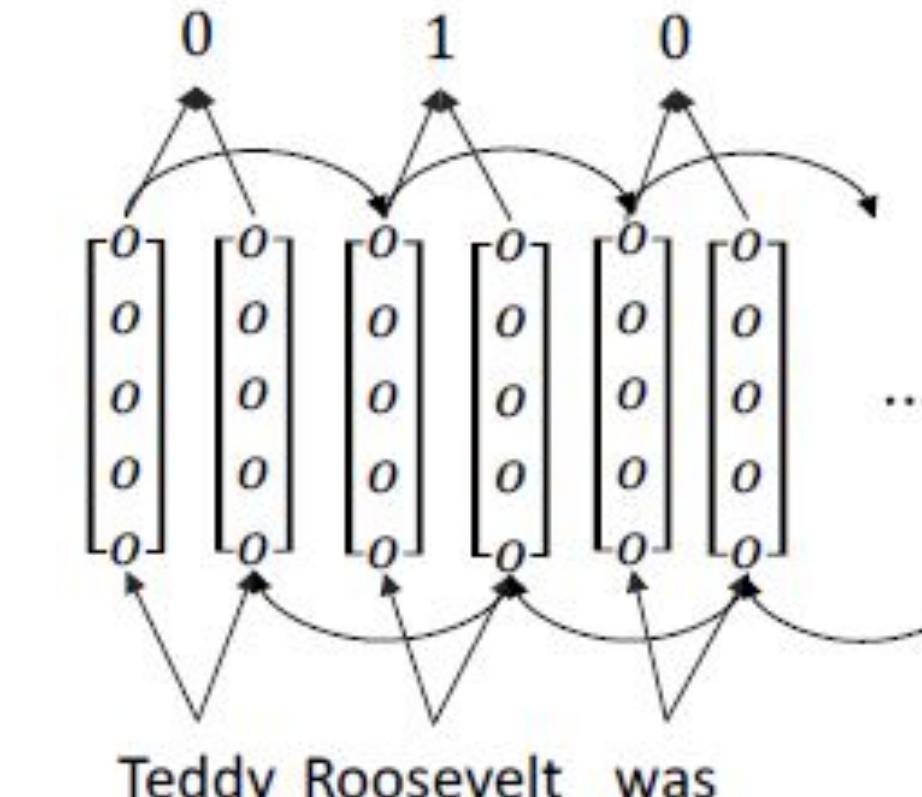
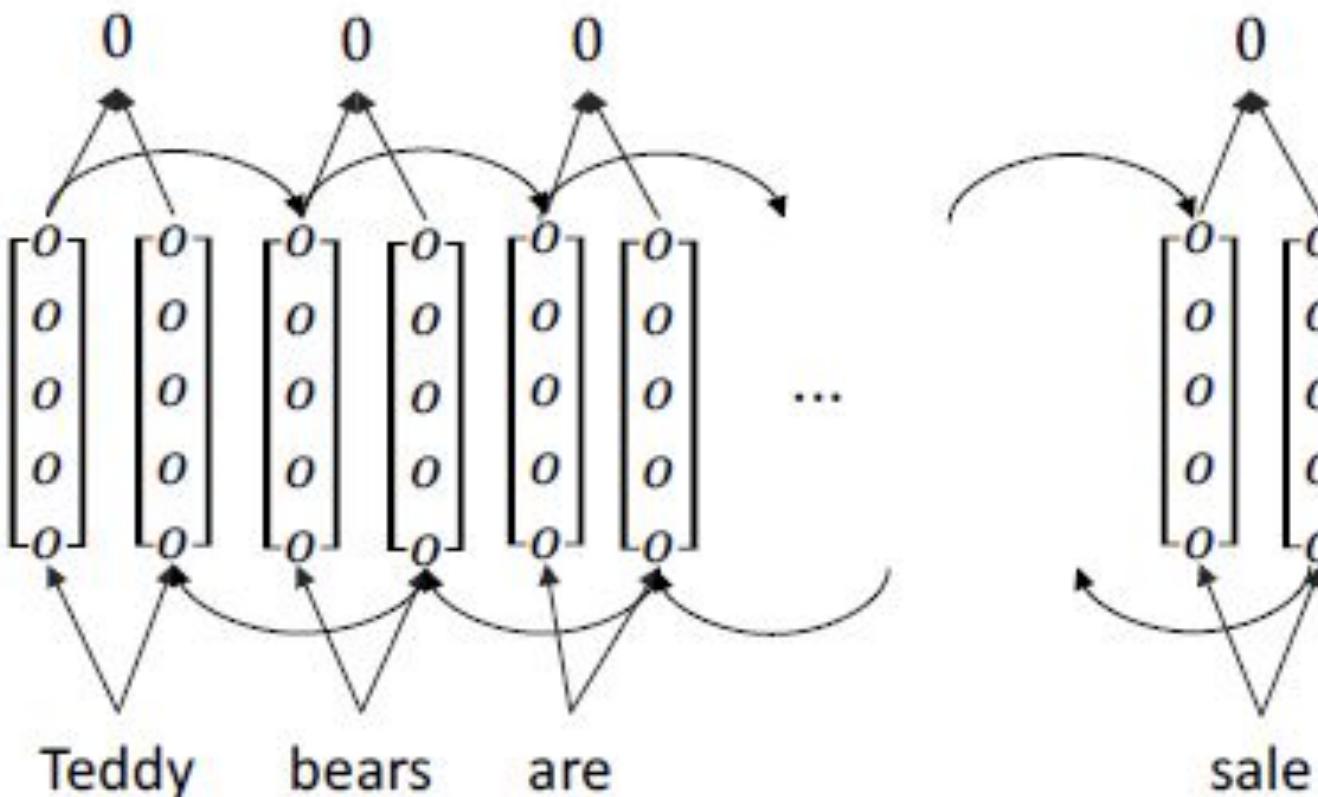
- Buenos resultados para NLP con bloques LSTM





Tipos de RNN: Bidireccionales

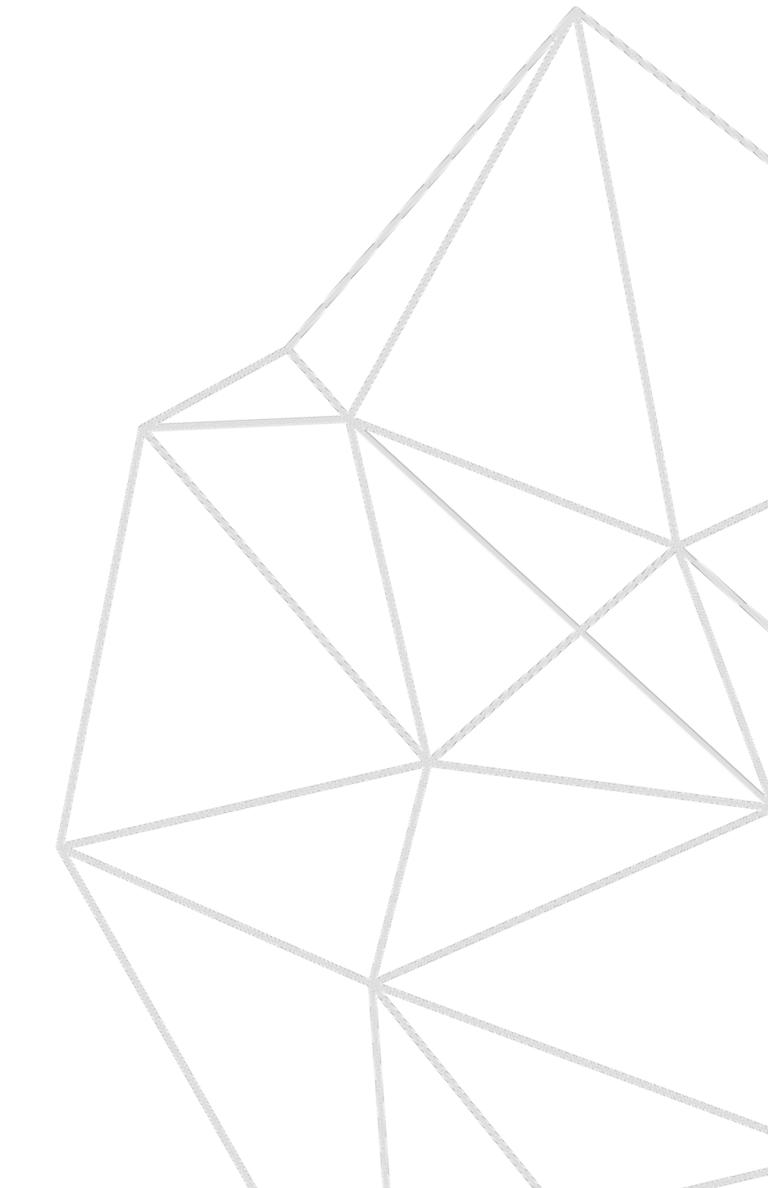
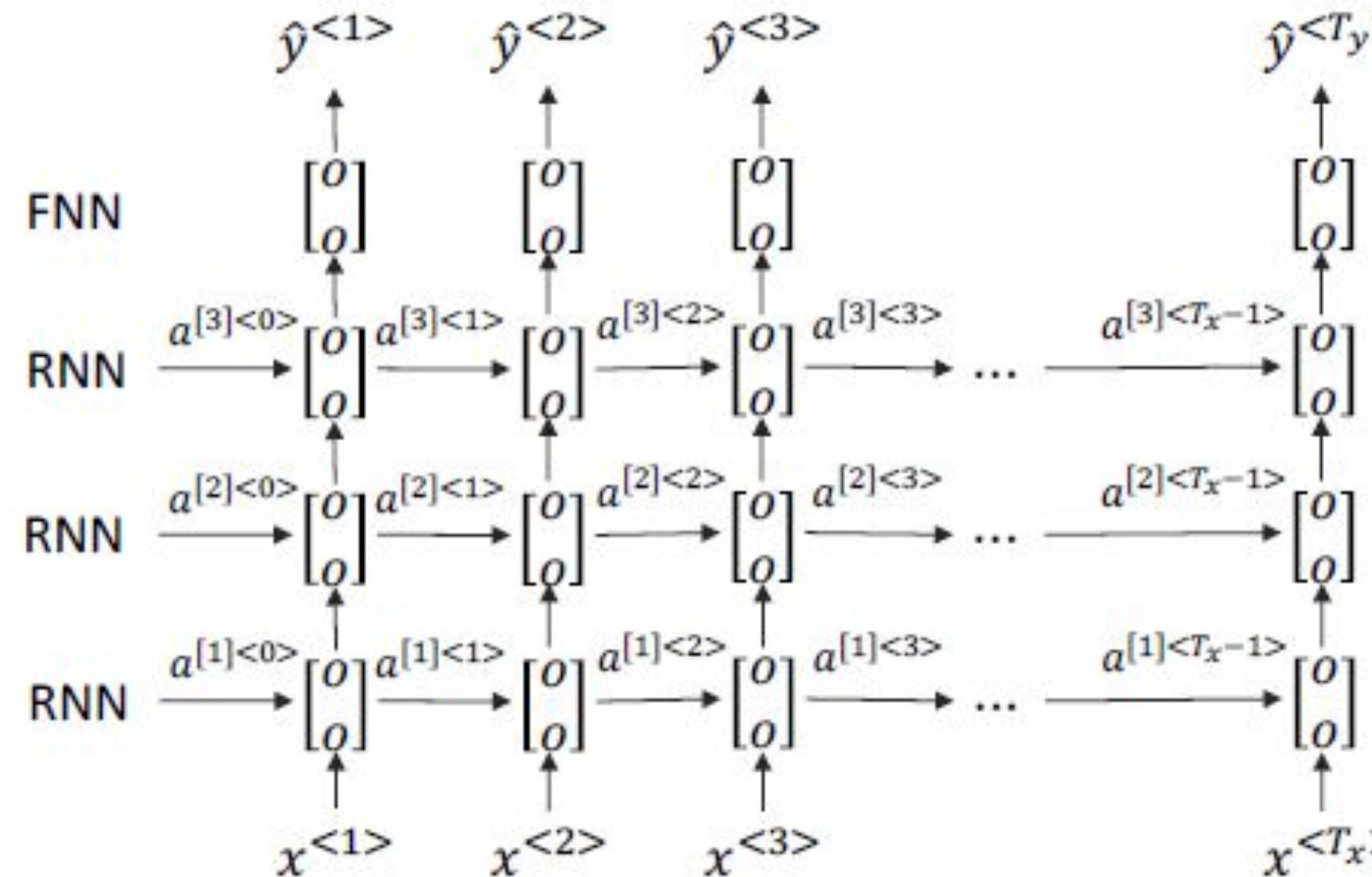
- Por ejemplo, reconocimiento de nombres propios
 - **Teddy** bears are on sale.
 - **Teddy** Roosevelt was a US president



Tipos de RNN: Deep RNN

Se pueden desarrollar RNN con varias capas ocultas

- 3 capas ocultas de RNN ya es “profundo”. No obstante, es habitual conectar las RNN con Deep FNN.





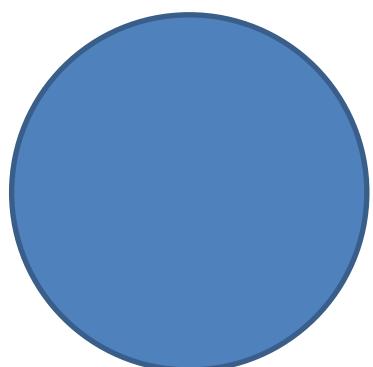
Vanishing/Exploding gradients

Para secuencias muy largas, RNN también tiene problemas de atenuación/explosión de los gradientes.

- A partir de 10.

Por ejemplo:

- **El perro** de San Roque, que fue adoptado por mi primo de Morales del Vino las navidades pasadas cuando vino a visitarme, no **tiene** rabo.
- **El perro** de San Roque, que fue adoptado por mi primo de Morales del Vino las navidades pasadas cuando vino a visitarme, no **tienen** rabo.

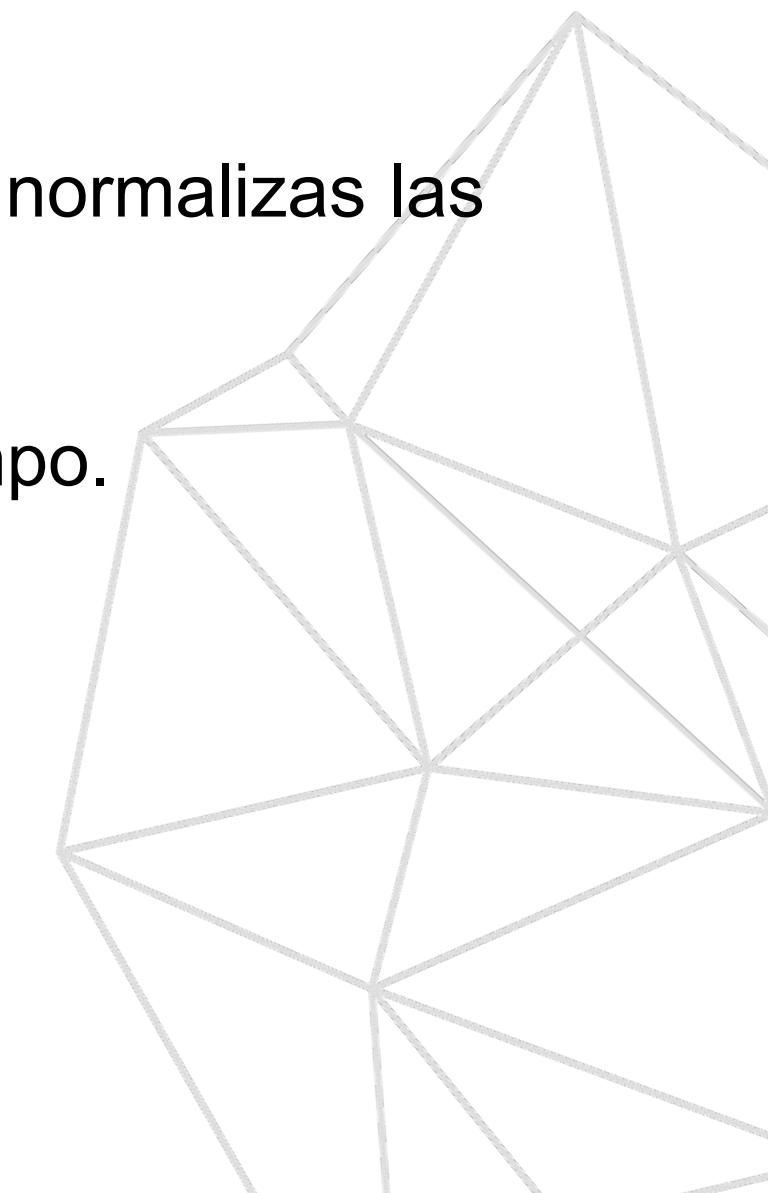
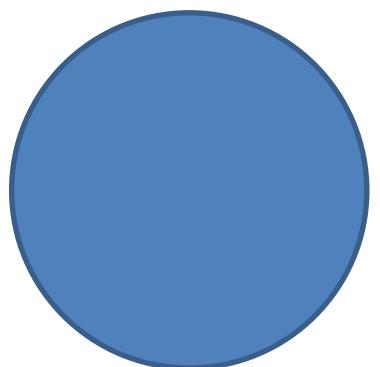




Vanishing/Exploding gradients

Solución:

- Utilizar **tanh** frente a ReLu.
- Constante de aprendizaje baja.
- Batch Normalization:
 - Poco útil.
 - Alternativa: Layer Normalization.
 - » En lugar de normalizar las variables independientemente a nivel de batch, normalizas las todas las variables a nivel de tupla.
- Dropout y recurrent dropout.
- Utilizar unidades/células que faciliten la propagación de los datos a lo largo del tiempo.
 - Long Short Term Memory (LSTM).
 - Gate Recurrent Unit (GRU).

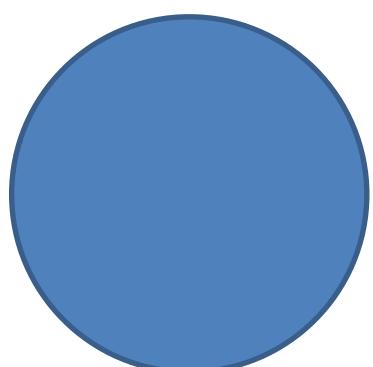




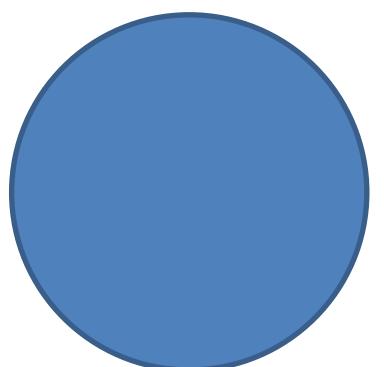
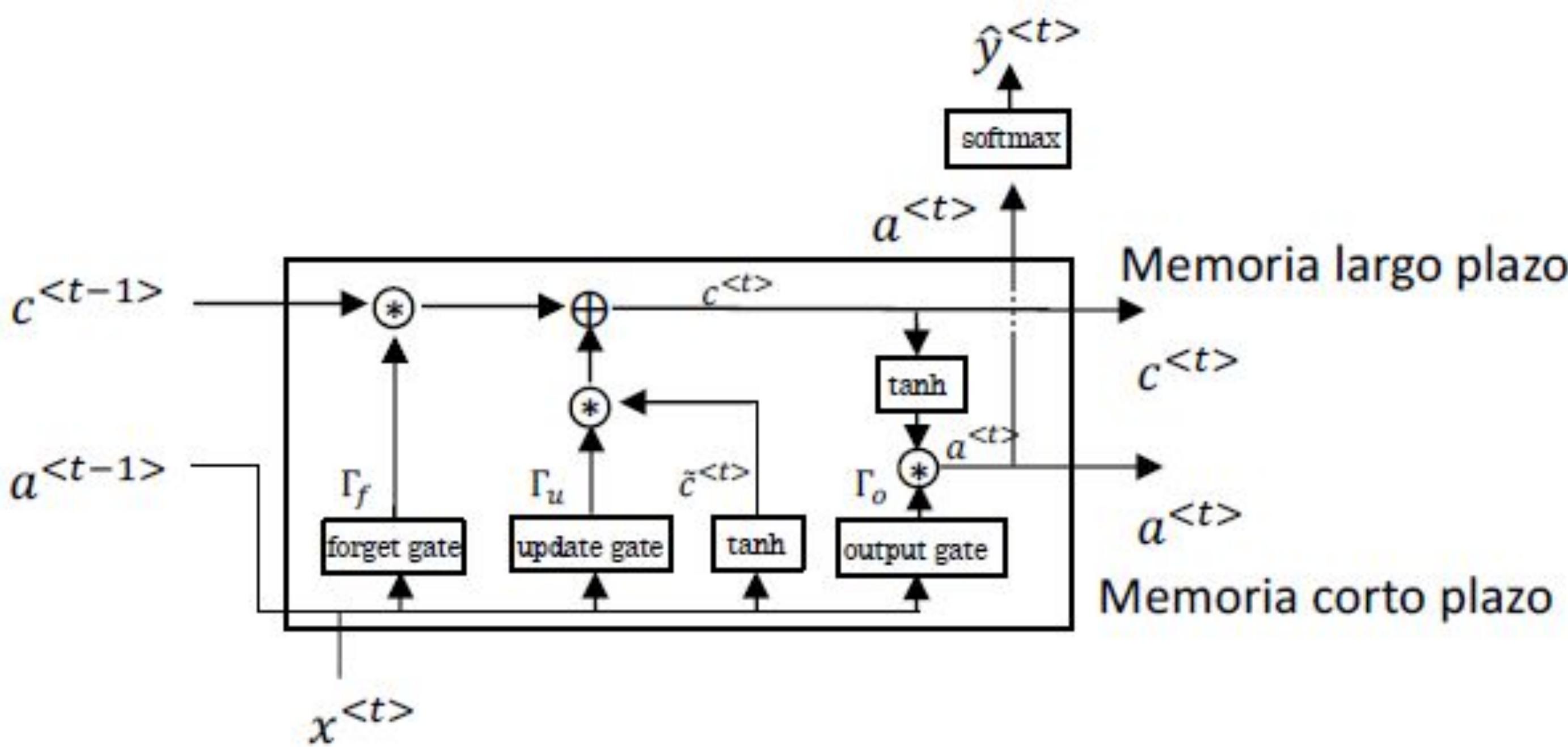
Long Short Term Memory

RNN con dos líneas de propagación de información en el tiempo:

- Memoria a corto plazo
 - Recupera la información ya sea inmediata o de varias iteraciones atrás en el tiempo para obtener la salida de cada capa.
 - Se corresponde con la salida y de la RNN
- Memoria a largo plazo
 - Se puede incorporar información a corto plazo para que esta se propague sin dificultades en el tiempo.



Long Short Term Memory: Unidad LSTM





Long Short Term Memory

- Posible recuerdo $\tilde{c}^{<t>}$ para t :

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

- Probabilidad de recordar $\tilde{c}^{<t>}$:

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

- Probabilidad de olvidar $\tilde{c}^{<t-1>}$:

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

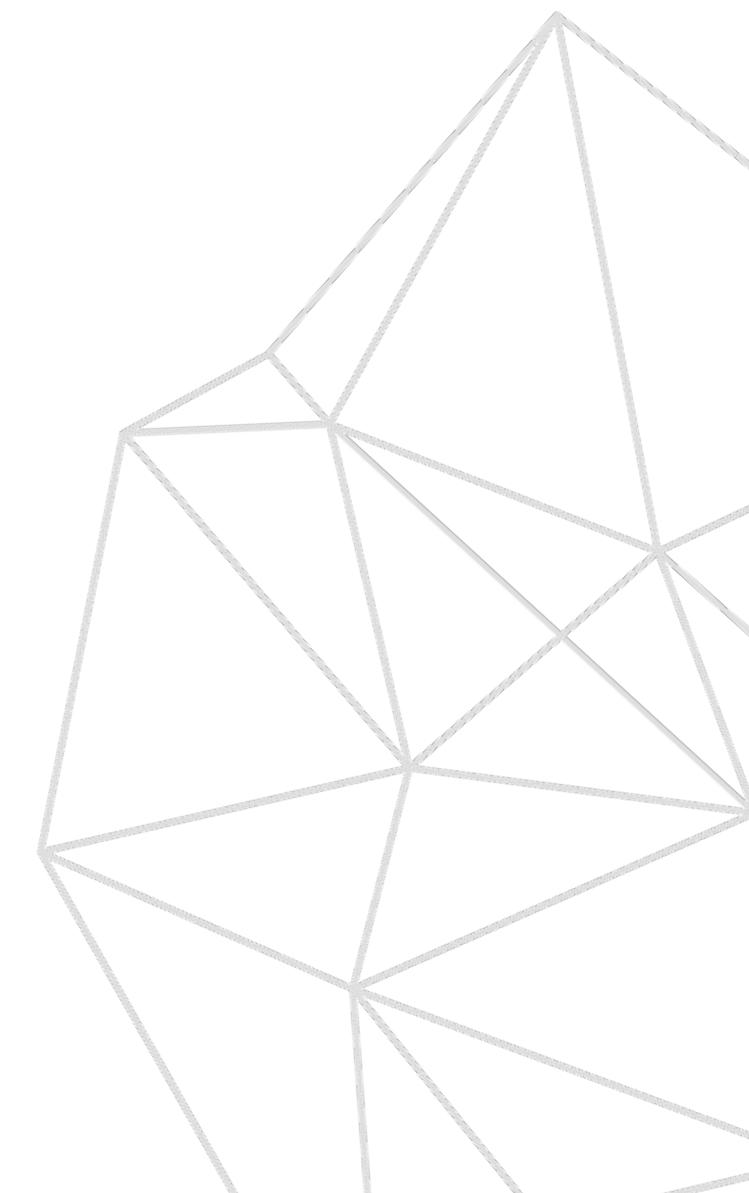
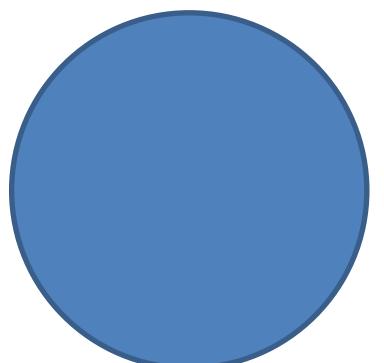
- Probabilidad transmitir el recuerdo $c^{<t>}$:

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

- Recuerdo c y salida a para t :

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$$

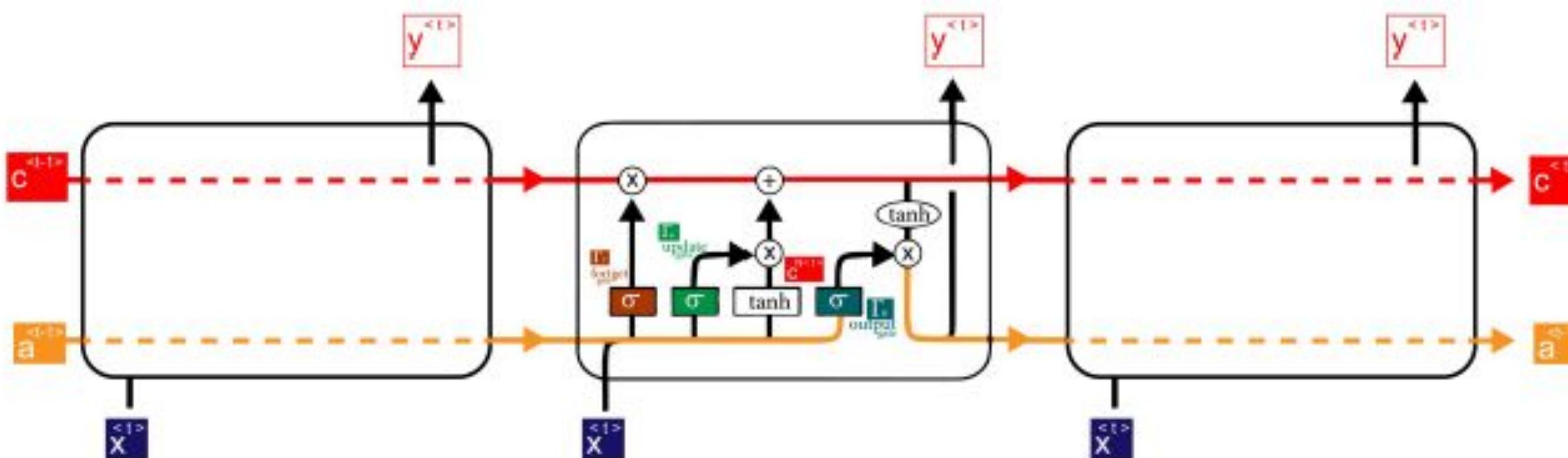




Long Short Term Memory: Unidad LSTM

Aprendiendo los pesos adecuados es fácil transmitir los recuerdos del pasado hacia el futuro (línea roja).

- Es posible transmitir recuerdos desde el principio hasta el final con precisión.





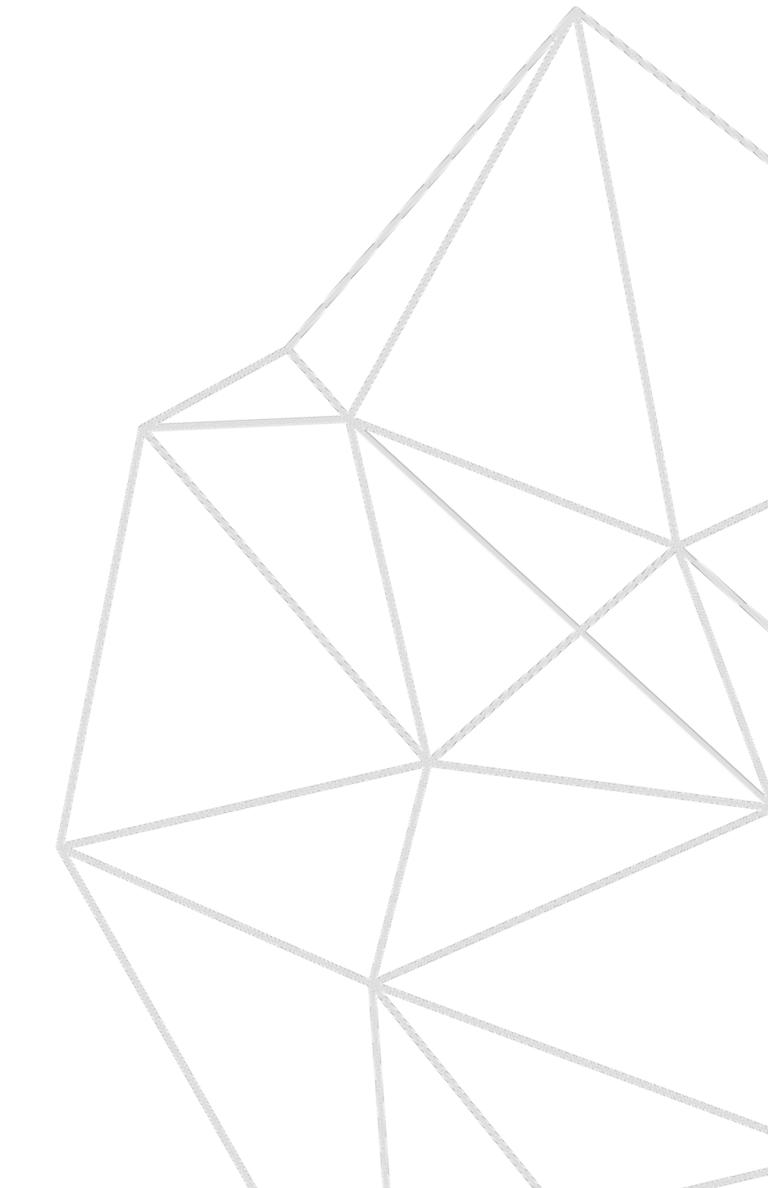
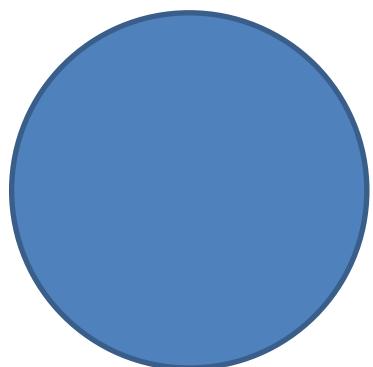
Long Short Term Memory: Peephole

Peephole LSTM es una variante que alimenta las compuertas:

- Forget y update con $c^{}$
- Output gate con $c^{}$

De modo que las compuertas tienen más información para poder tomar las decisiones para olvidar y recordar.

No existe un conocimiento claro de cuándo funciona mejor. Es necesario probar.

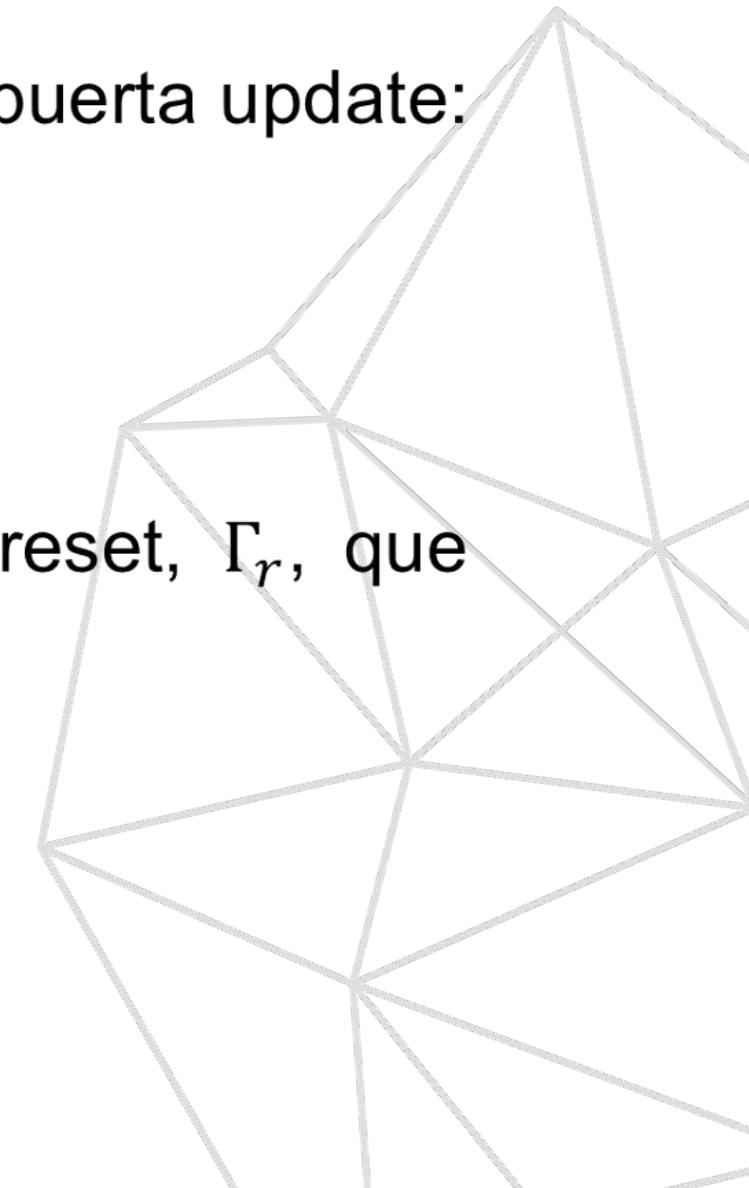
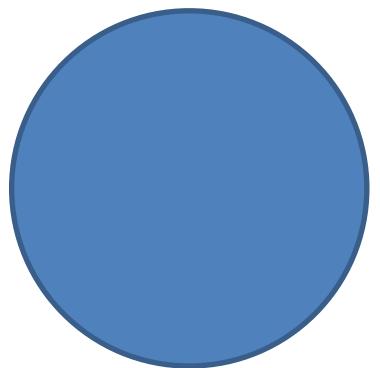




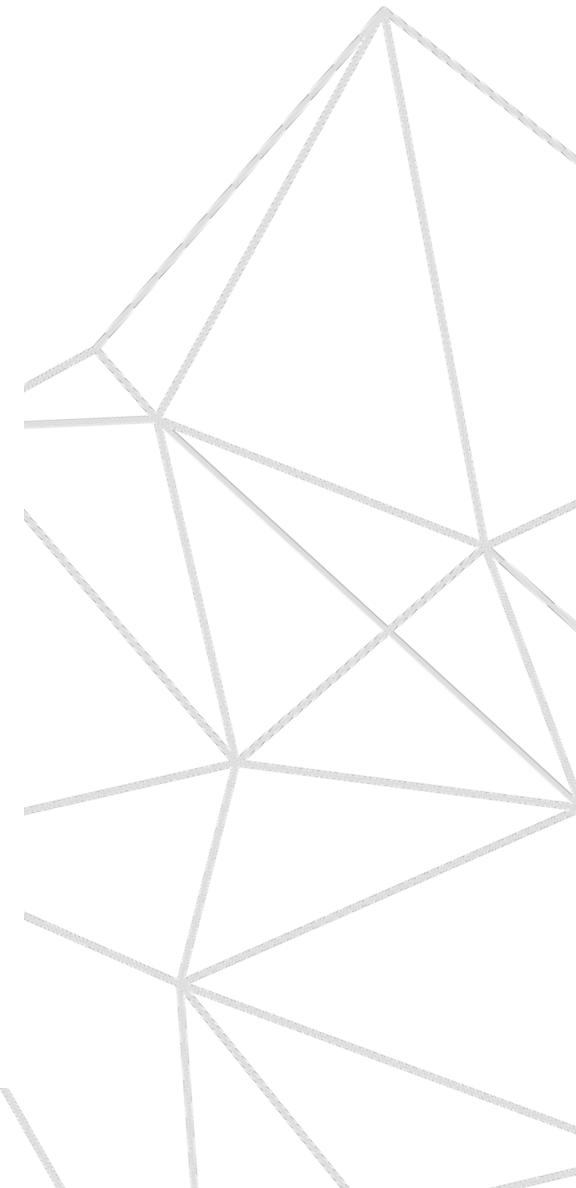
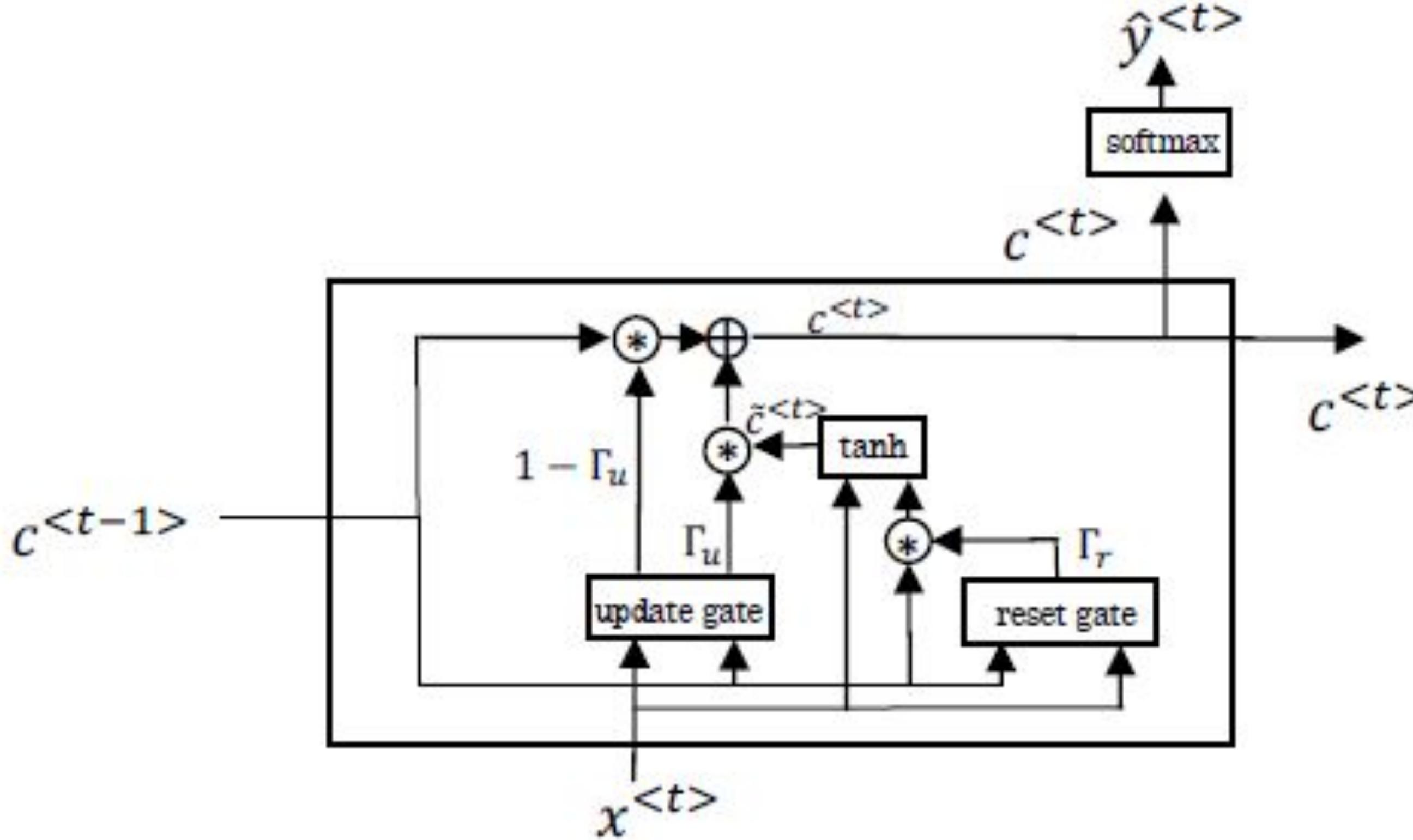
Gate Recurrent Unit

Versión simplificada de los LSTM

- En lugar de tener dos líneas de propagación, memoria a **corto y largo plazo**, solo existe **una línea de propagación** como en una neurona recurrente normal.
 - Propongamos c a través del tiempo: $a^{<t>} = c^{<t>}$.
- Las compuertas forget y update son reemplazadas por una sola compuerta, la compuerta update:
 Γ_u
 - Forget: $1 - \Gamma_u$
 - Update: Γ_u
- Se elimina la compuerta output y se crea una nueva compuerta de relevancia/reset, Γ_r , que determina qué información es importante o se puede olvidar.



Gate Recurrent Unit





Gate Recurrent Unit

Propagamos c a través del tiempo $a^{<t>} = c^{<t>}.$

- Relevancia (reset) de $c^{<t-1>}:$

$$\Gamma_r = \sigma(W_r [c^{<t-1>} , x^{<t>}] + b_r)$$

- Posible recuerdo $\tilde{c}^{<t>}$ para $t :$

$$\tilde{c}^{<t>} = \tanh(W_c [\Gamma_r * c^{<t-1>} , x^{<t>}] + b_c)$$

- Probabilidad de recordar $\tilde{c}^{<t>}$ en lugar de $c^{<t-1>}$

$$\Gamma_u = \sigma(W_u [c^{<t-1>} , x^{<t>}] + b_u)$$

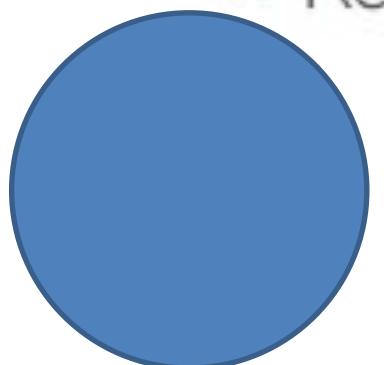
- Recuerdo para $t :$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

Cuando Γ_u es cercano a 0, mantiene los recuerdos sin ningún tipo de atenuación.

$$\Gamma_u = \sigma(W_u [c^{<t-1>} , x^{<t>}] + b_u)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$



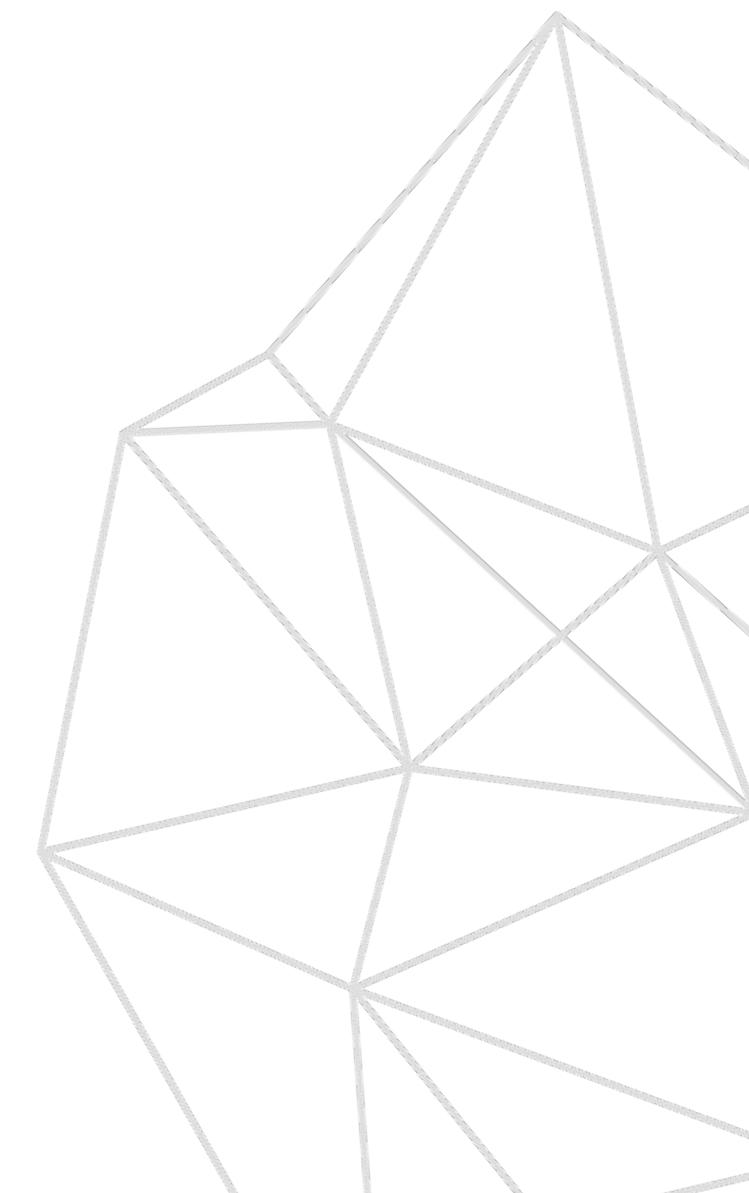
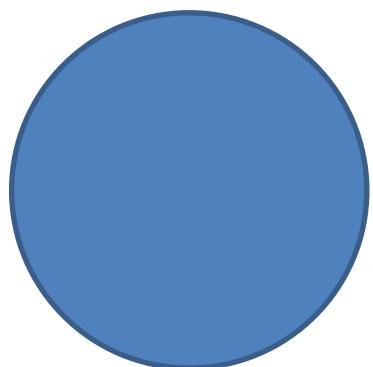


Problema de las largas secuencias

Incluso con las mejoras de las LSTM y GRU, las RNN vuelven a tener problemas con secuencias de tamaño superior a 100.

- Solución:
 - Utilizar una capa convolutiva para alimentar la entrada de la RNN con secuencias de entradas en cada momento t
 - Utilizar capas convolutivas que se retroalimentan en el futuro.

»WaveNet





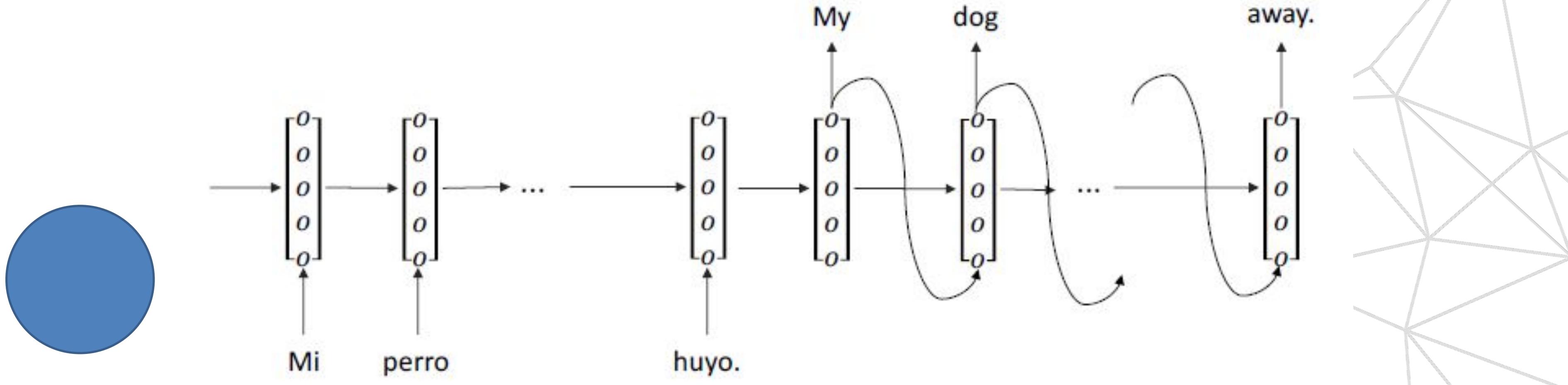
Beam Search

Se escoge la palabra con mayor probabilidad (softmax) en cada momento t (Greedy Search).

- **Problema:** Respuestas parciales con mayor probabilidad pueden no tener la mayor probabilidad conjunta final.

$$P(\text{cat}|X, My) > P(\text{dog}|X, My)$$

$$P(\text{My cat run away}|X) < P(\text{My dog run away}|X)$$

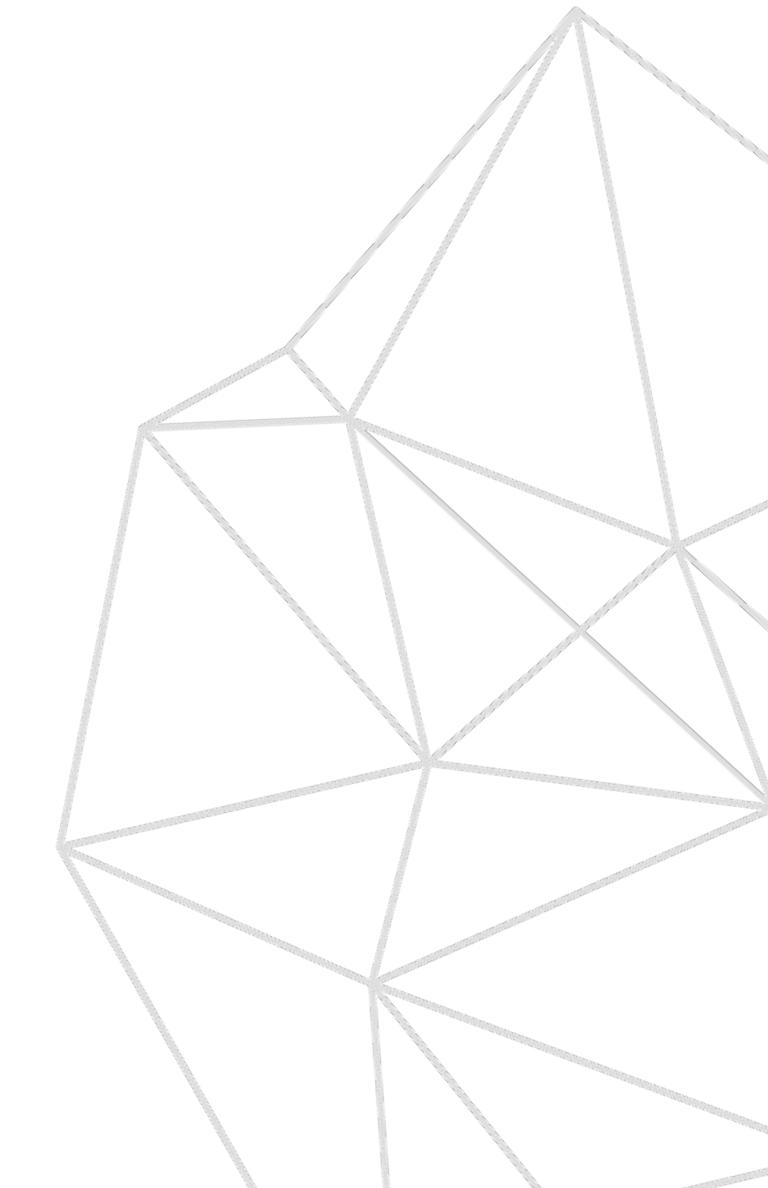
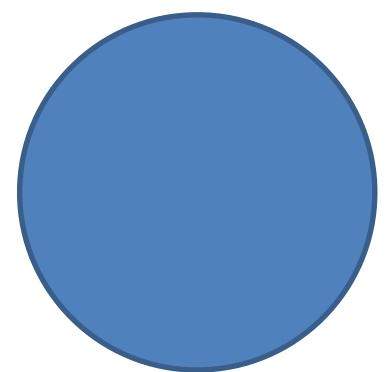




Beam Search

- **Solución:** Explorar varias ramas al mismo tiempo (las más probables)
 - Búsqueda Beam de grado 3.

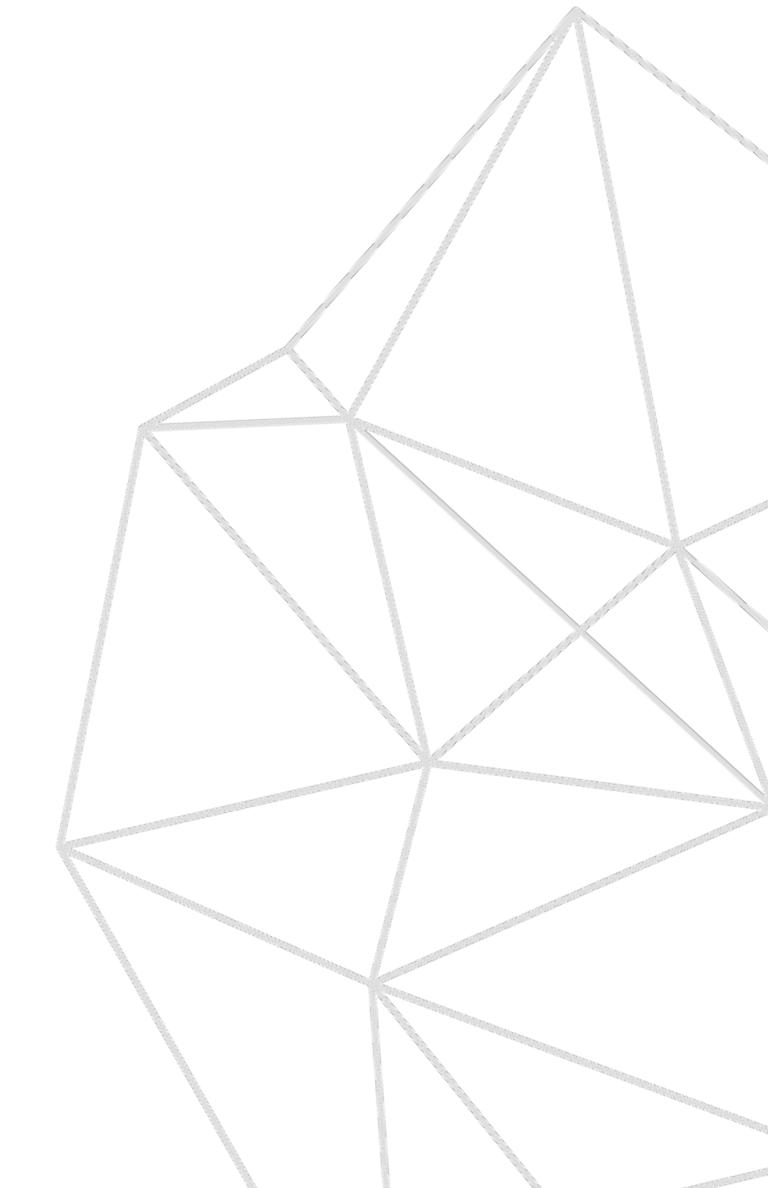
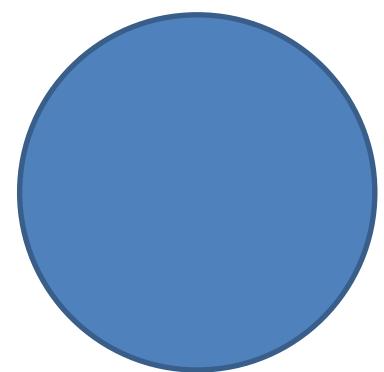
» My
» The
» A





Beam Search

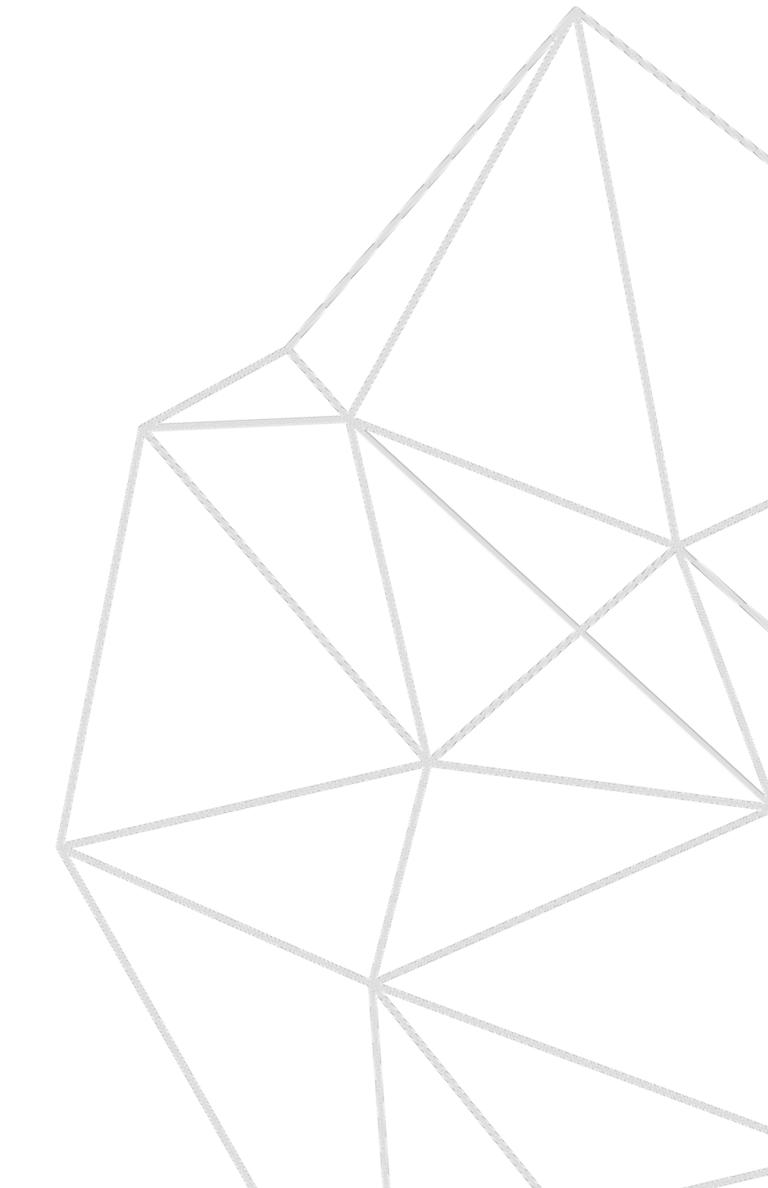
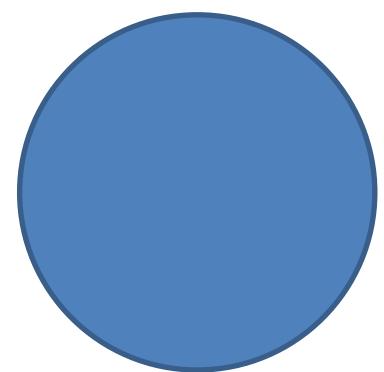
- **Solución:** Explorar varias ramas al mismo tiempo (las más probables)
 - Búsqueda Beam de grado 3.
 - » My cat
 - » My dog
 - » The dog





Beam Search

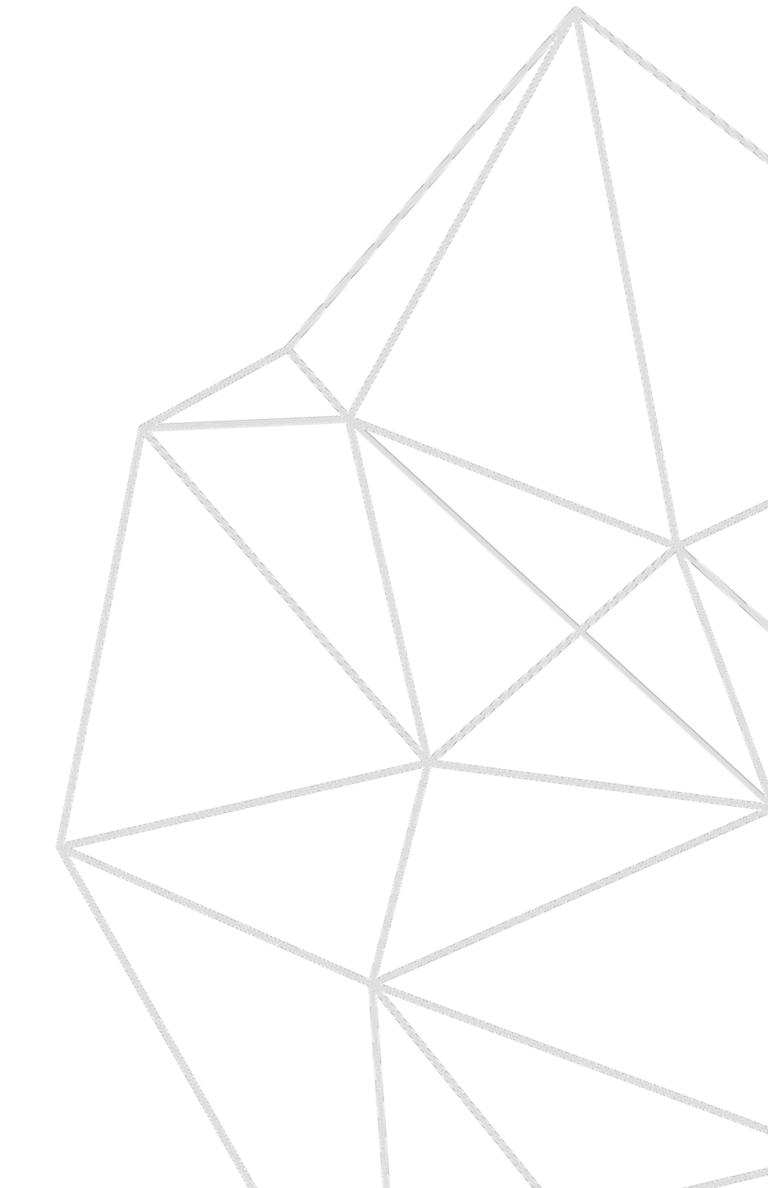
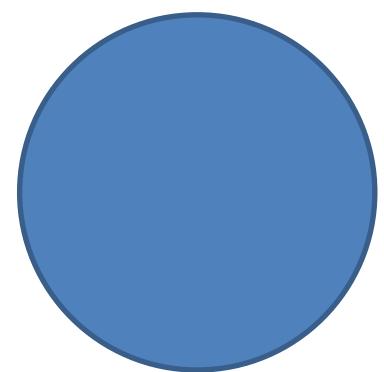
- **Solución:** Explorar varias ramas al mismo tiempo (las más probables)
 - Búsqueda Beam de grado 3.
 - » My cat is
 - » My dog run
 - » The dog run





Beam Search

- **Solución:** Explorar varias ramas al mismo tiempo (las más probables)
 - Búsqueda Beam de grado 3.
 - » My cat is running
 - » My dog run away.
 - » The dog run away.





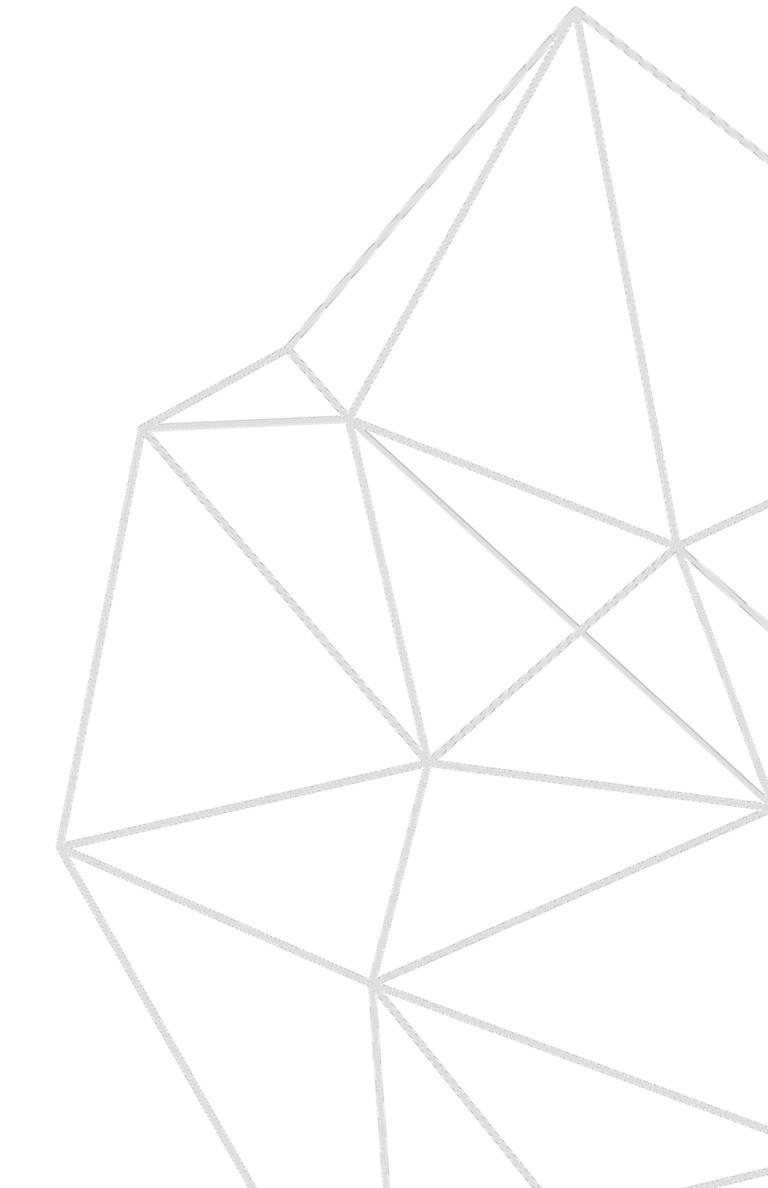
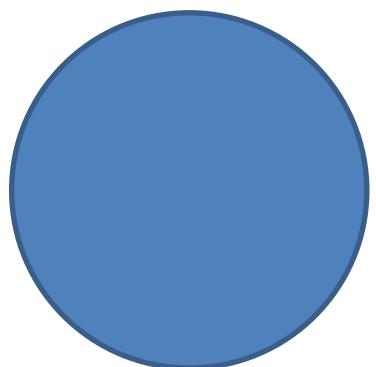
Beam Search

- **Solución:** Explorar varias ramas al mismo tiempo (las más probables)

- Búsqueda Beam de grado 3.

- » My cat is running away.
- » **My dog run away.**
- » The dog run away.

$$P(\text{My dog is running away}|X) < P(\text{The dog run away}|X)$$
$$P(\text{The dog run away}|X) < P(\text{My dog run away}|X)$$

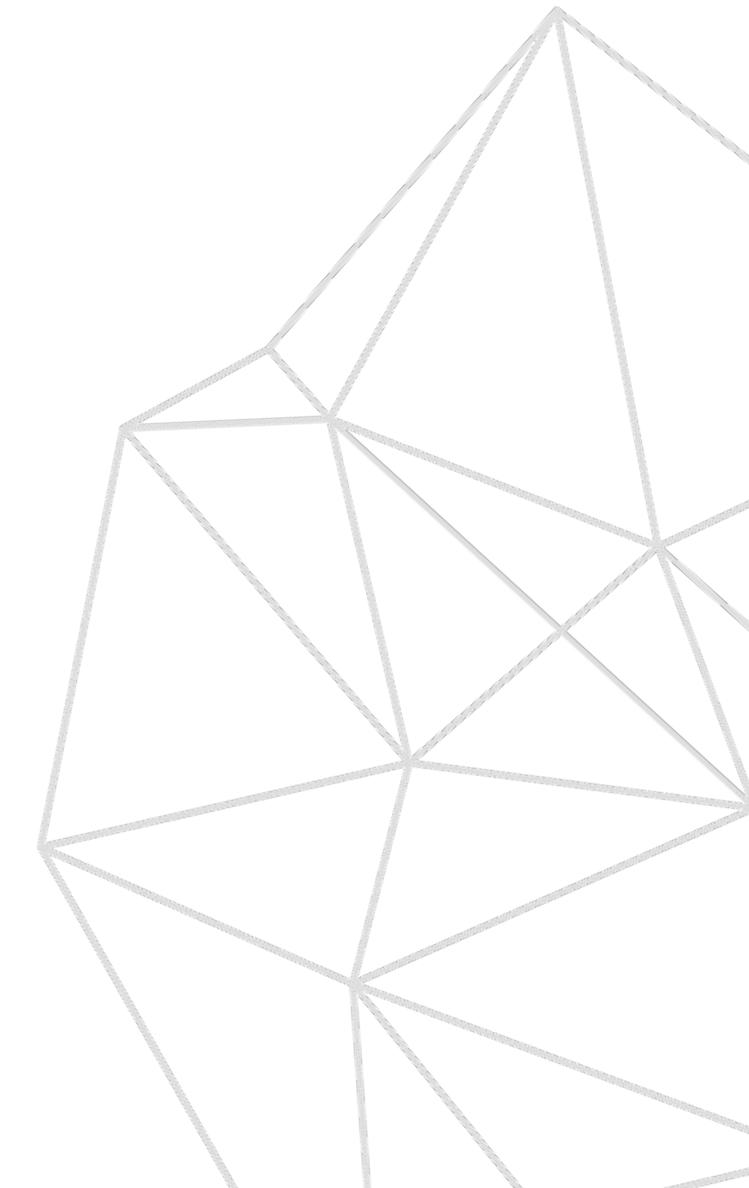
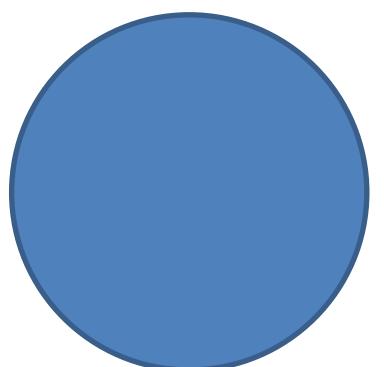




Beam Search

Tamaño óptimo de B

- A mayor tamaño, mejores resultados, pero con un coste computacional mayor.
- Producción:
 - Valores pequeños cercanos a 10.





Beam Search

Problema con secuencias largas

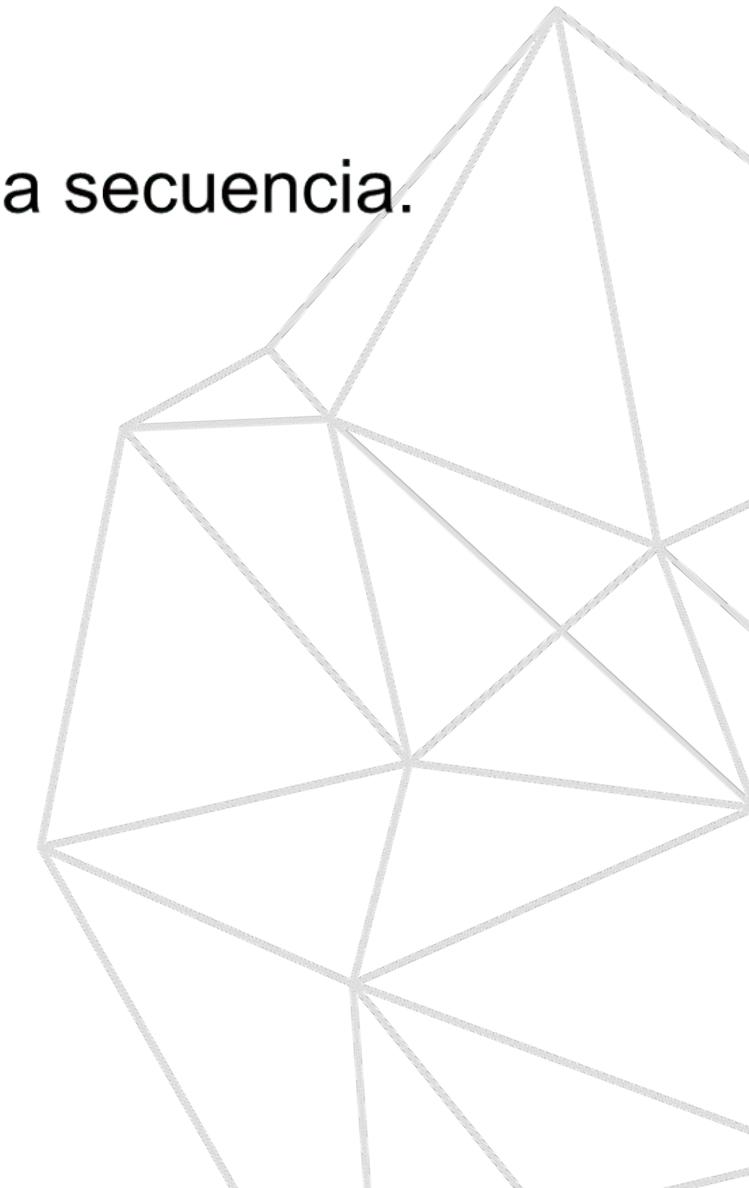
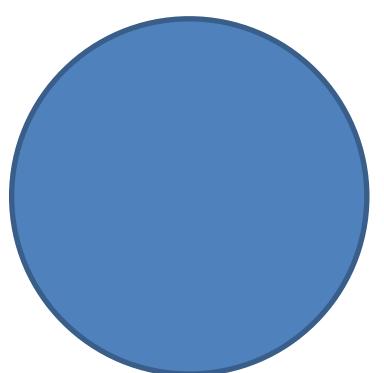
- Utilizar (hiperparámetro de normalización $\alpha = 0.7$)

$$\arg \max_y \frac{1}{T_y^\alpha} \sum_{t=1}^{T_y} \log P(y^{} | x, y^{<1>}, \dots, y^{$$

- Donde α es un hiperparámetro para la normalización en función de la longitud de la secuencia.
- Valor típico $\alpha = 0.7$

- En lugar de

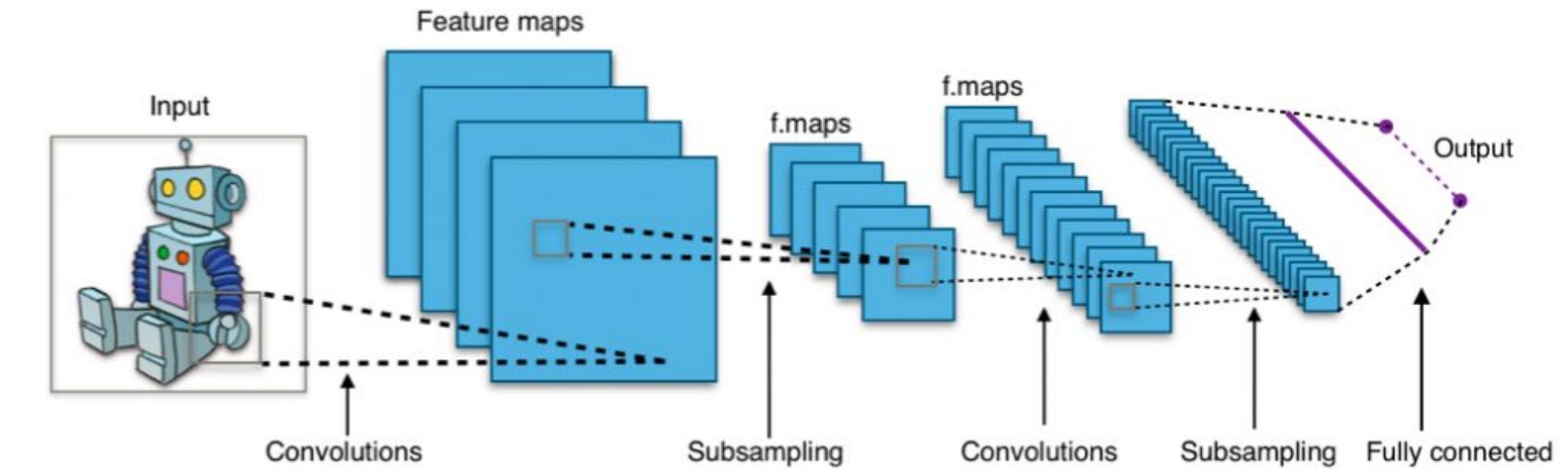
$$\arg \max_y \prod_{t=1}^{T_y} P(y^{} | x, y^{<1>}, \dots, y^{$$





05

Redes de Neuronas Convolucionales





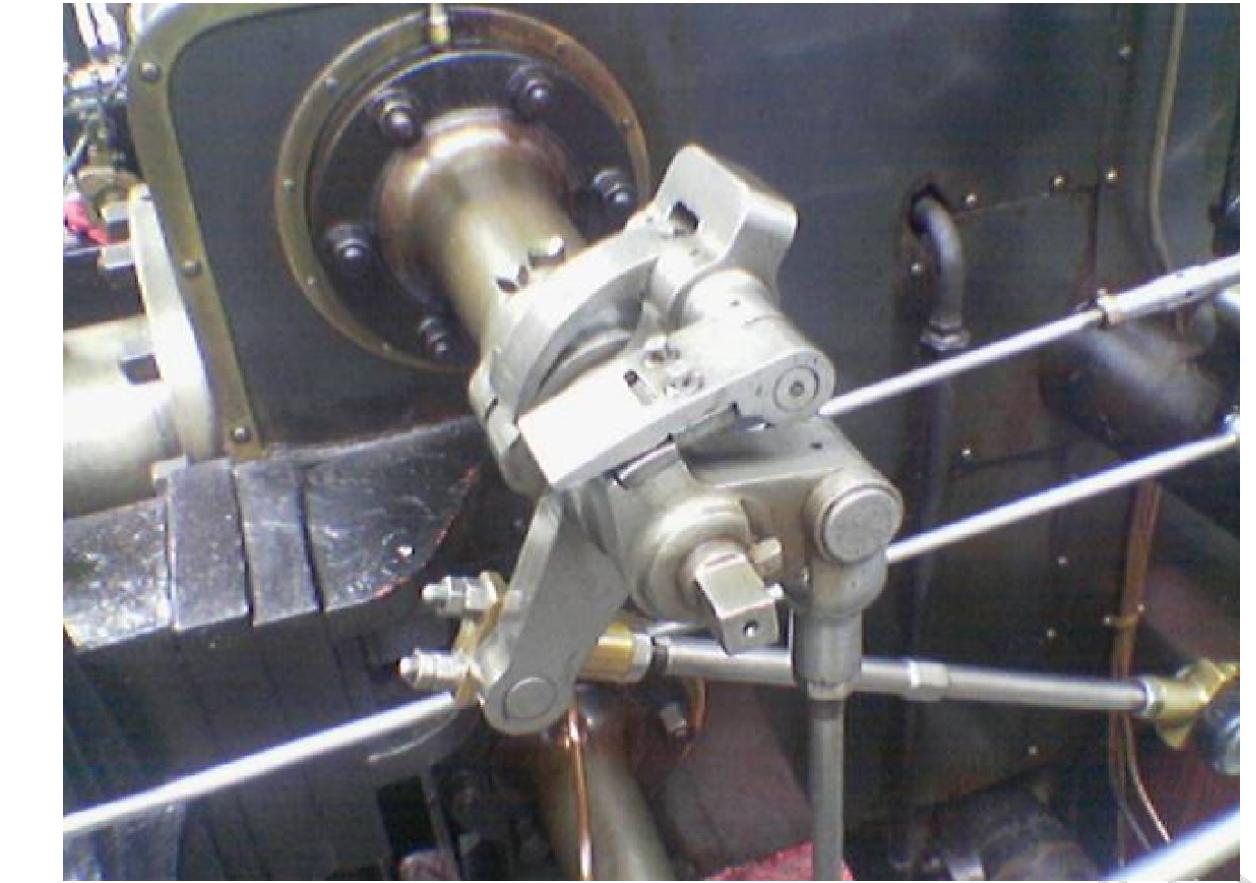
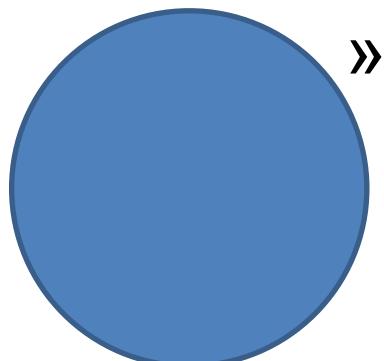
Introducción

Redes de neuronas orientadas al procesamiento de imágenes:

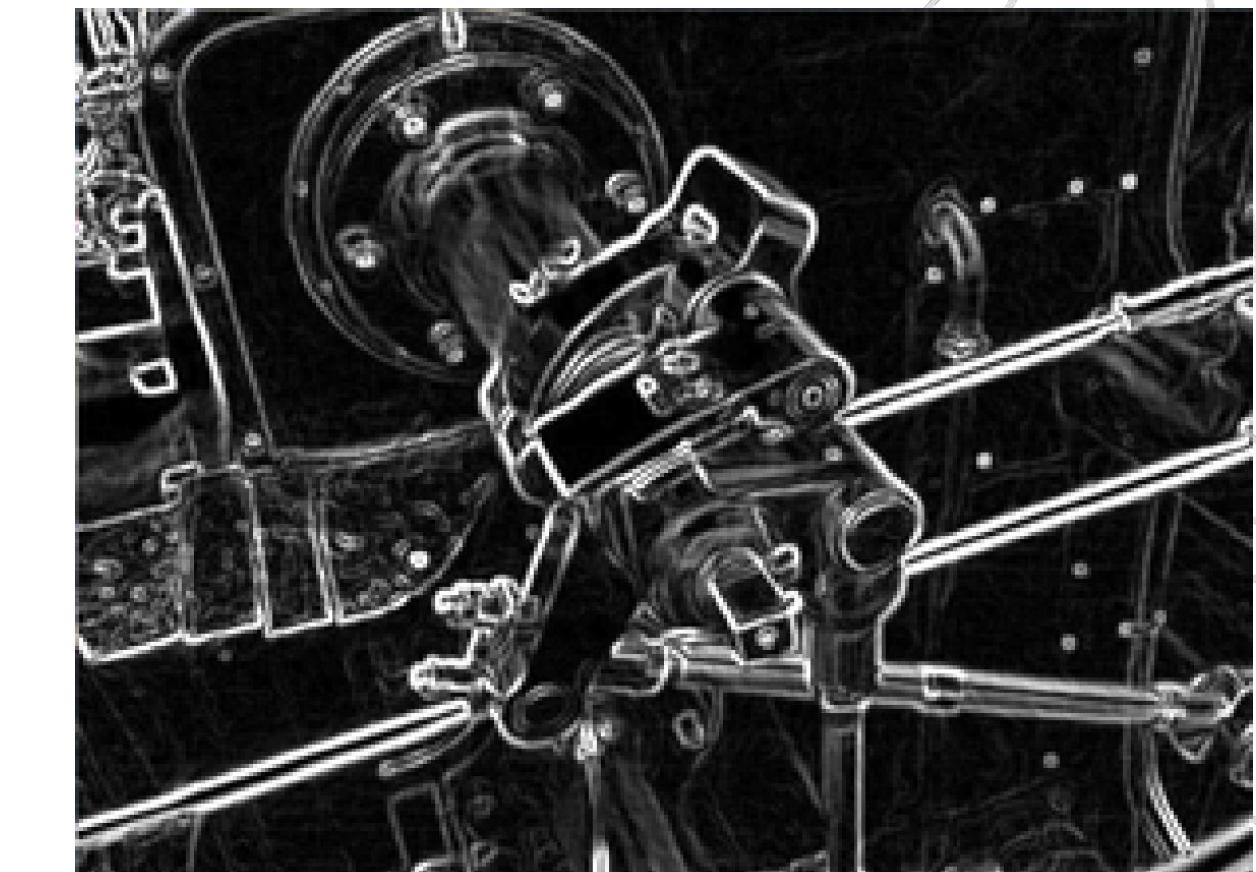
- Clasificación de imágenes.
- Reconocimiento facial.
- Detección de objetos.

Entrada de la red neuronal: Imágenes:

- Número de entradas
 - Ancho x Altura x Canales
 - » $64 \times 64 \times 3 = 12,288$ Entradas
 - » $1 \text{ MP} \times 3 = 3.000.000$ Entradas
- Pesos en la primera capa oculta.
 - Con 1000 neuronas en la primera capa oculta
 - » 12.288.000 pesos
 - » 3.000.000.000 pesos



Ejemplo de detección de bordes con filtro Sobel





Detección de bordes

Se utiliza una máscara (filtro) para detectar bordes con el operador convolución * (en realidad llamado cross-correlation)

1	0	2	5	3	3
2	1	3	8	4	4
2	4	2	7	4	9
4	3	4	1	9	7
4	2	5	6	3	7
2	3	5	8	8	9

$$\begin{matrix} & \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} & = & \begin{matrix} -2 \\ \vdots \\ \vdots \\ \vdots \end{matrix} \end{matrix}$$

Filtro o kernel
vertical

$$1 * 1 + 2 * 1 + 2 * 1 + 0 * 0 + 1 * 0 + 4 * 0 - 2 * 1 - 3 * 1 - 2 * 1 = -2$$



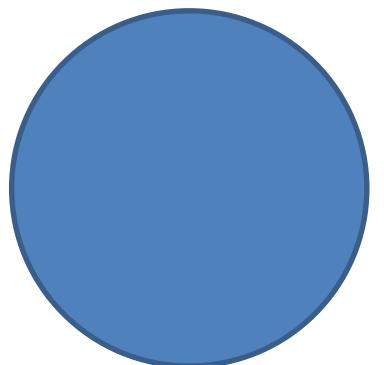
Detección de bordes

Se utiliza una máscara (filtro) para detectar bordes con el operador convolución *

1	0	2	5	3	3
2	1	3	8	4	4
2	4	2	7	4	9
4	3	4	1	9	7
4	2	5	6	3	7
2	3	5	8	8	9

$$\begin{matrix} * & \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} & = & \begin{matrix} -2 & -15 & & \\ & & & \\ & & & \\ & & & \end{matrix} \end{matrix}$$

Filtro o kernel
vertical





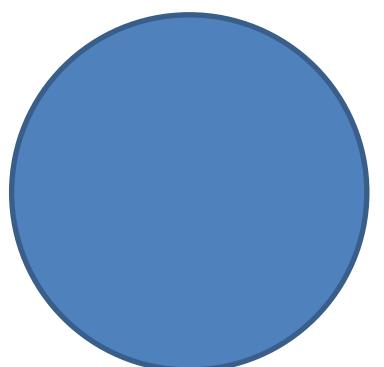
Detección de bordes

Se utiliza una máscara (filtro) para detectar bordes con el operador convolución *

1	0	2	5	3	3
2	1	3	8	4	4
2	4	2	7	4	4
4	3	4	1	9	7
4	2	5	6	3	7
2	3	5	8	8	9

$$\begin{matrix} * & \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} & = & \begin{matrix} -2 & -15 & -4 & 9 \\ -1 & & & \\ & & & \\ & & & \end{matrix} \end{matrix}$$

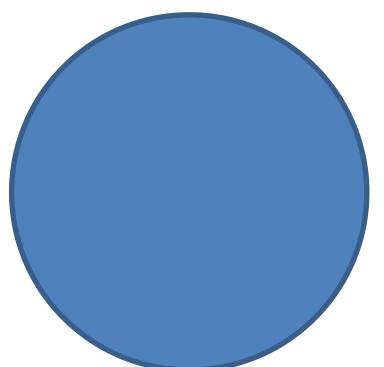
Filtro o kernel
vertical





Detección de bordes

Se utiliza una máscara (filtro) para detectar bordes con el operador convolución *



1	0	2	5	3	3
2	1	3	8	4	4
2	4	2	7	4	4
4	3	4	1	9	7
4	2	5	6	3	7
2	3	5	8	8	9

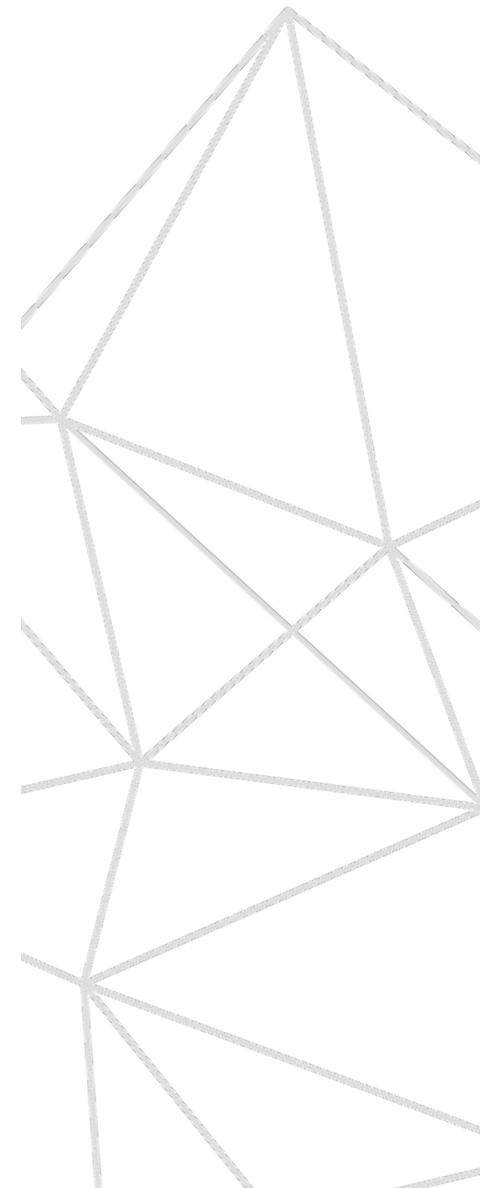
*

1	0	-1
1	0	-1
1	0	-1

=

-2	-15	-4	9
-1	-8	-8	1
-1	-5	-5	-4
-4	-7	-6	-8

Filtro o kernel
vertical

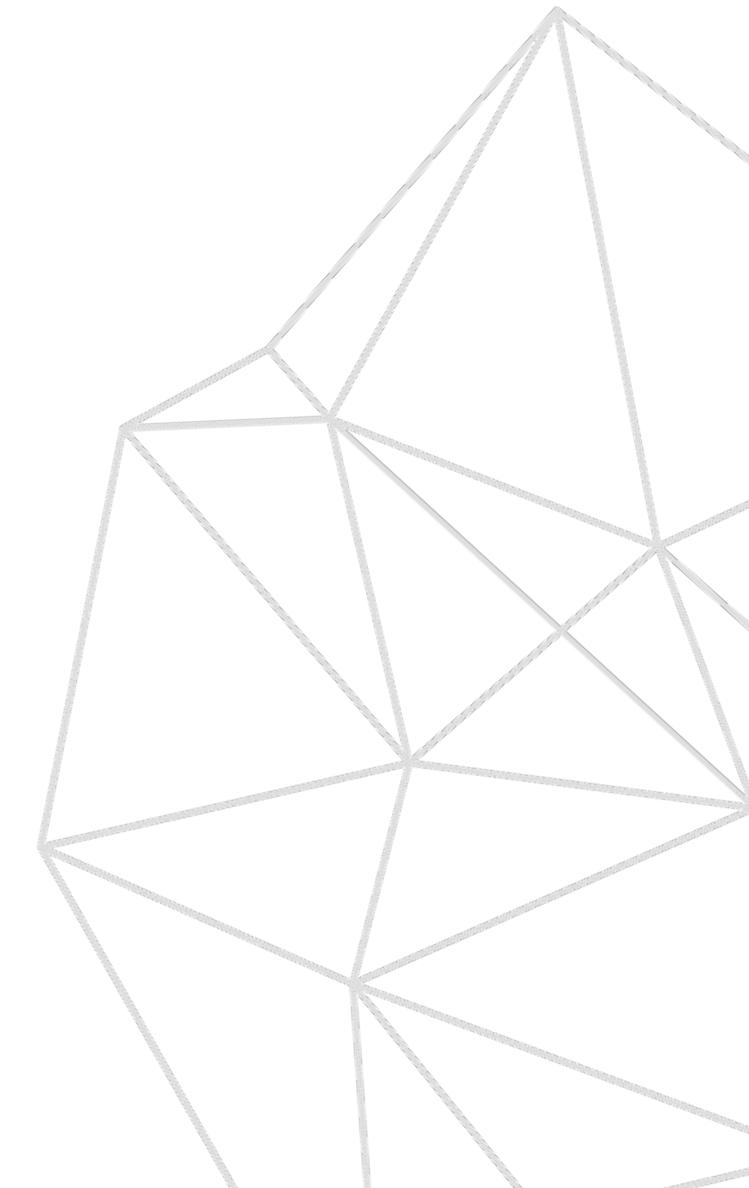
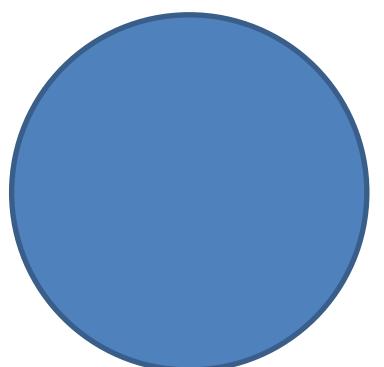




Detección de bordes

Se utiliza una máscara (filtro) para detectar bordes con el operador convolución *

- Algunas dimensiones típicas de los filtros son:
 - 1×1
 - 3×3
 - 5×5
 - 7×7
- Impar:
 - De modo que se puedan identificar la casilla / fila / columna central.
 - La operación *padding* se pueda aplicar de forma simétrica.





Detección de bordes

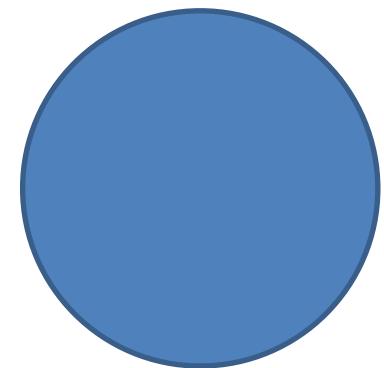
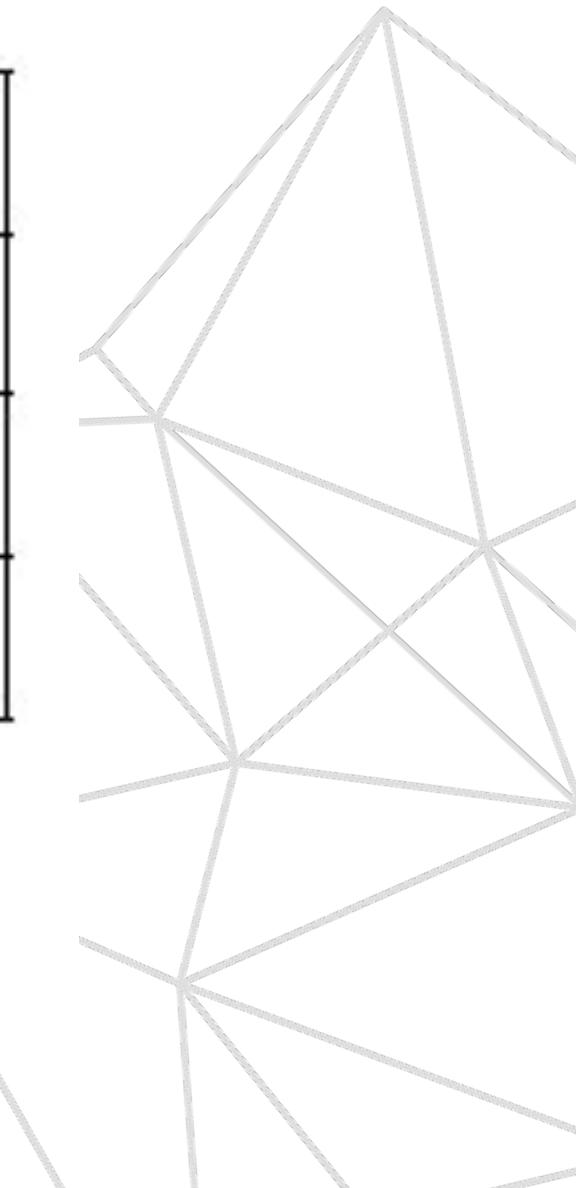
Se utiliza una máscara (filtro) para detectar bordes con el operador convolución *

5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0

$$\begin{matrix} * & \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} & = & \begin{matrix} 0 & 15 & 15 & 0 \\ 0 & 15 & 15 & 0 \\ 0 & 15 & 15 & 0 \\ 0 & 15 & 15 & 0 \end{matrix} \end{matrix}$$

Filtro o kernel
vertical

0	15	15	0
0	15	15	0
0	15	15	0
0	15	15	0



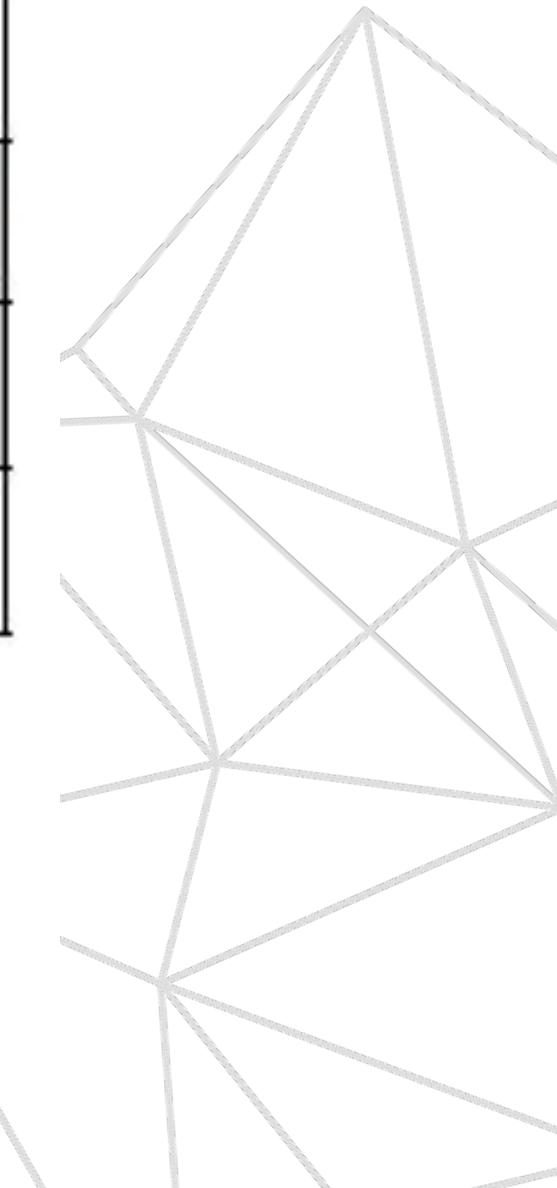
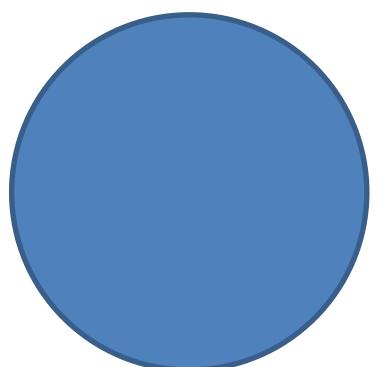


Detección de bordes

Se utiliza una máscara (filtro) para detectar bordes con el operador convolución *

$$\begin{matrix} 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} * \begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix} = \begin{matrix} 0 & 0 & 0 & 0 \\ 15 & 15 & 15 & 15 \\ 15 & 15 & 15 & 15 \\ 0 & 0 & 0 & 0 \end{matrix}$$

Filtro o kernel horizontal





Detección de bordes

Se utiliza una máscara (filtro) para detectar bordes con el operador convolución *

5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
0	0	0	5	5	5
0	0	0	5	5	5
0	0	0	5	5	5

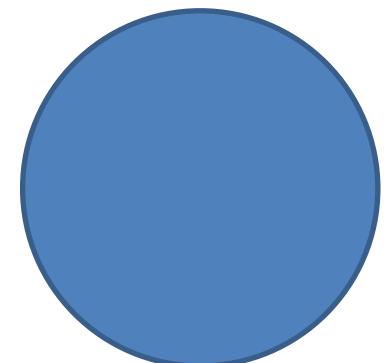
*

1	1	1
0	0	0
-1	-1	-1

=

0	0	0	0
15	5	-5	-15
15	5	-5	-15
0	0	0	0

Filtro o kernel
horizontal





Detección de bordes

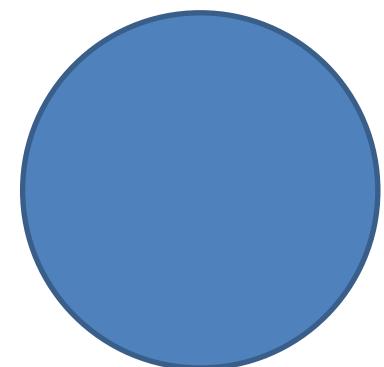
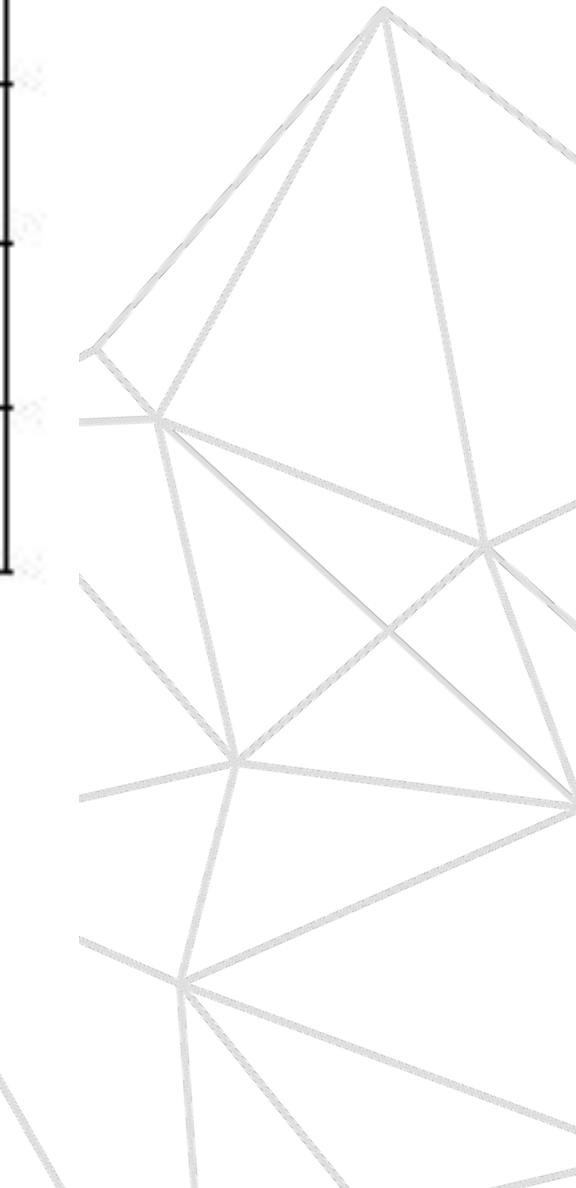
Se utiliza una máscara (filtro) para detectar bordes con el operador convolución *

5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
0	0	0	5	5	5
0	0	0	5	5	5
0	0	0	5	5	5

$$\begin{matrix} * & \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix} \end{matrix}$$

Filtro Sobel
vertical

$$= \begin{matrix} \begin{matrix} 0 & 20 & 20 & 0 \\ 0 & 10 & 10 & 0 \\ 0 & -10 & -10 & 0 \\ 0 & -20 & -20 & 0 \end{matrix} \end{matrix}$$





Detección de bordes

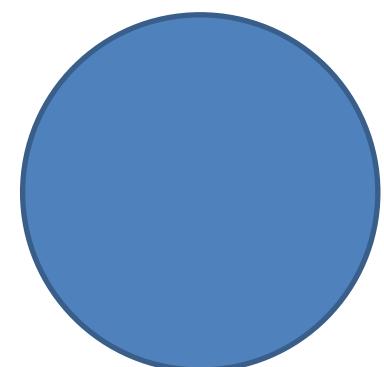
Se utiliza una máscara (filtro) para detectar bordes con el operador convolución *

5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
0	0	0	5	5	5
0	0	0	5	5	5
0	0	0	5	5	5

$$\begin{matrix} * & \begin{matrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{matrix} \end{matrix}$$

Filtro Scharr
vertical

$$= \begin{matrix} \begin{matrix} 0 & 180 & 180 & 0 \\ 0 & 150 & 150 & 0 \\ 0 & -150 & -150 & 0 \\ 0 & -180 & -180 & 0 \end{matrix} \end{matrix}$$





Detección de bordes

Los valores del filtro son parámetros optimizables en una red neuronal.

5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
0	0	0	5	5	5
0	0	0	5	5	5
0	0	0	5	5	5

*

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Filtro

=

El filtro resultante puede ser vertical, horizontal o incluso con una inclinación diferente

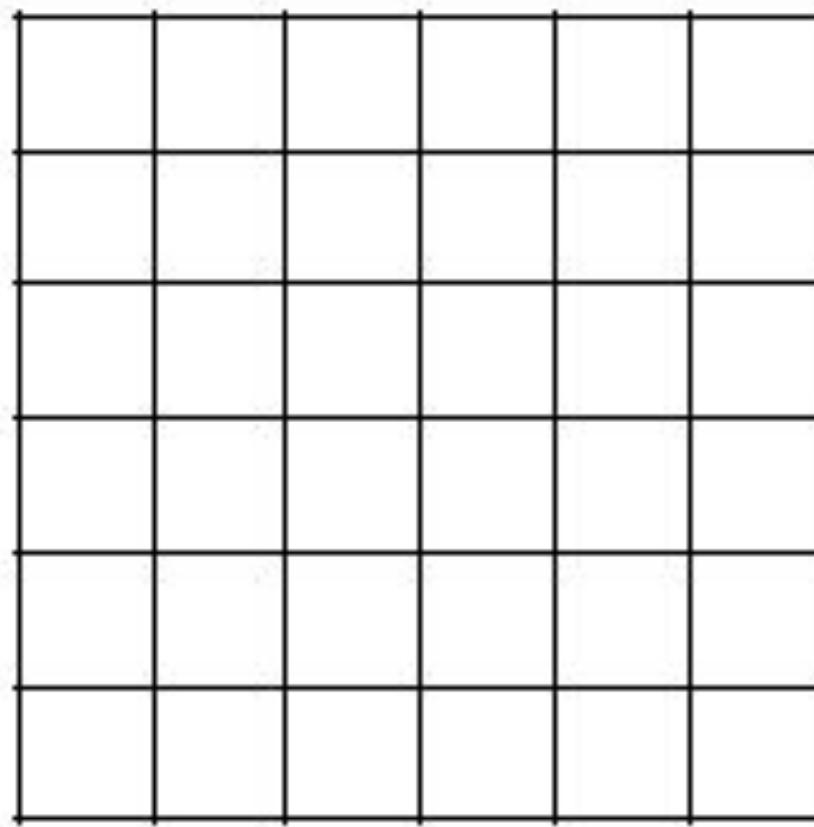
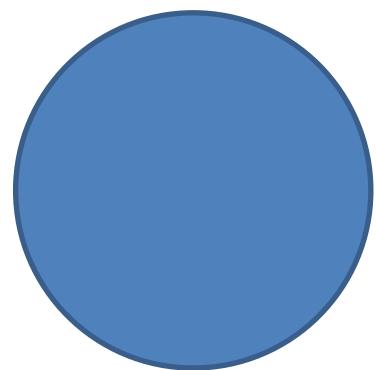


Padding

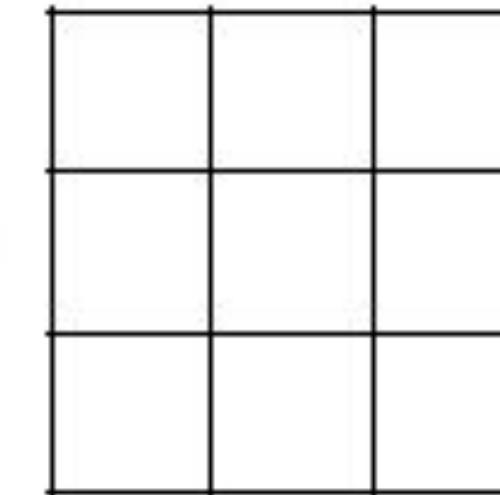
La matriz resultante al aplicar un filtro tiene un tamaño:

$$\begin{aligned}n - f + 1 \times n - f + 1 \\6 - 3 + 1 \times 6 - 3 + 1 = 4 \times 4\end{aligned}$$

¿Qué hacemos si queremos que no se reduzca el tamaño?

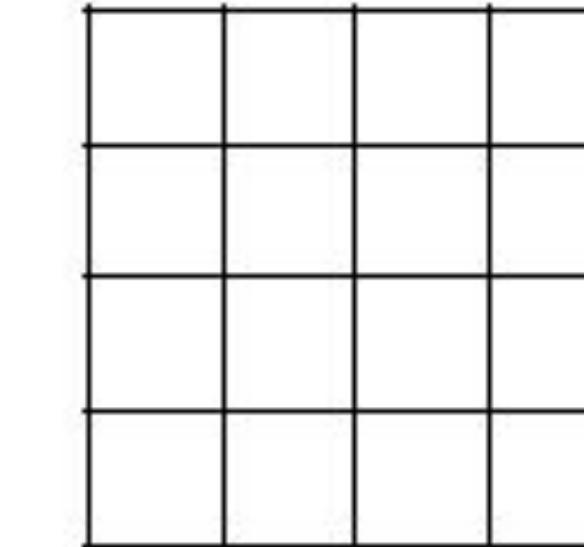


*



Filtro

=



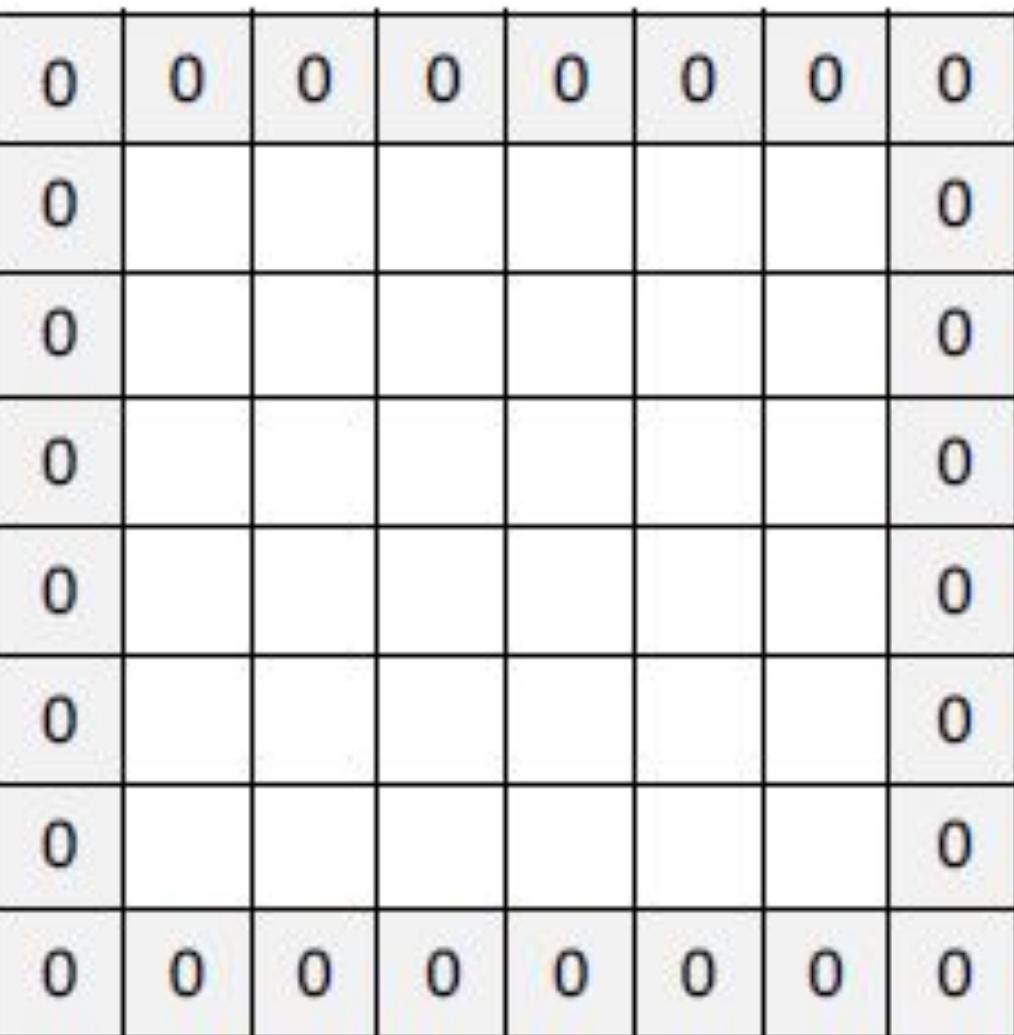


Padding

¿Qué hacemos si queremos que no se reduzca el tamaño?

- Aumentamos el tamaño de la matriz original

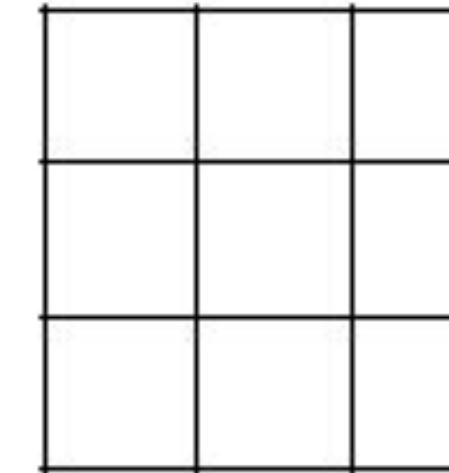
$$n + 2p - f + 1 \times n + 2p - f + 1$$



A 8x8 input matrix with all elements set to 0. The last column contains values 0, 0, 0, 0, 0, 0, 0, 0 from top to bottom. The second column from the right contains values 0, 0, 0, 0, 0, 0, 0, 0 from top to bottom. The first two columns are entirely 0s.

0	0	0	0	0	0	0	0
0							0
0							0
0							0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

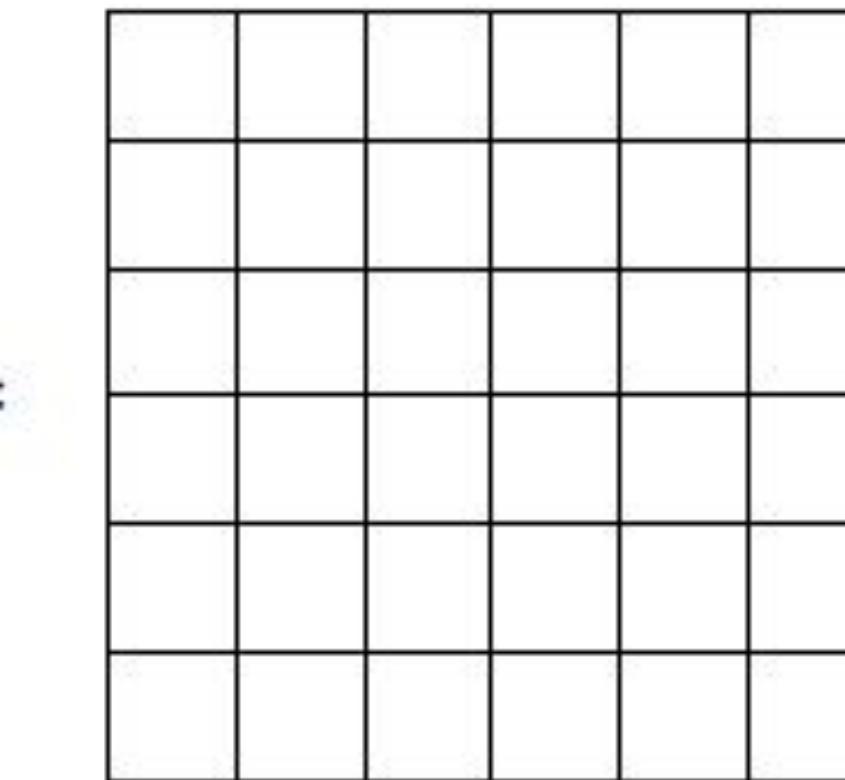
*



A 3x3 filter matrix with all elements set to 0.

0	0	0
0	0	0
0	0	0

Filtro



A 6x6 result matrix with all elements set to 0.

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0



Padding = 1



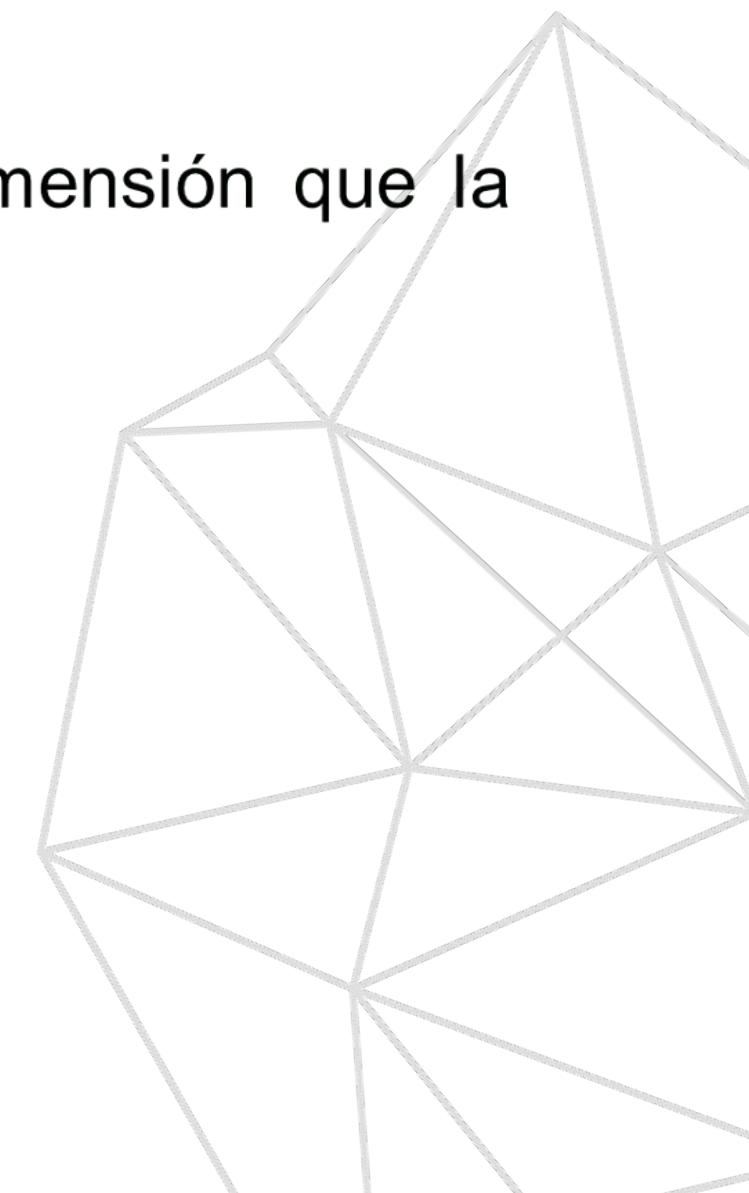
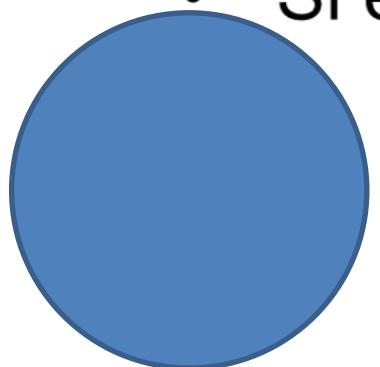
Padding: Tipos

Dimensión de la matriz resultante

$$n + 2p - f + 1 \times n + 2p - f + 1$$

Podemos clasificar el *padding* como:

- *Valid*: No se aplica padding:
 - $p = 0$
 - *Dimensión matriz resultante*: $n - f + 1 \times n - f + 1$
- *Same*: Se aplica *padding* de modo que la matriz resultante tenga la misma dimensión que la matriz inicial.
 - $p = \frac{f-1}{2}$
 - Dimensión matriz resultante $n \times n$
- Si el filtro no es impar, el *padding* puede tener que aplicarse de forma asimétrica





Stride

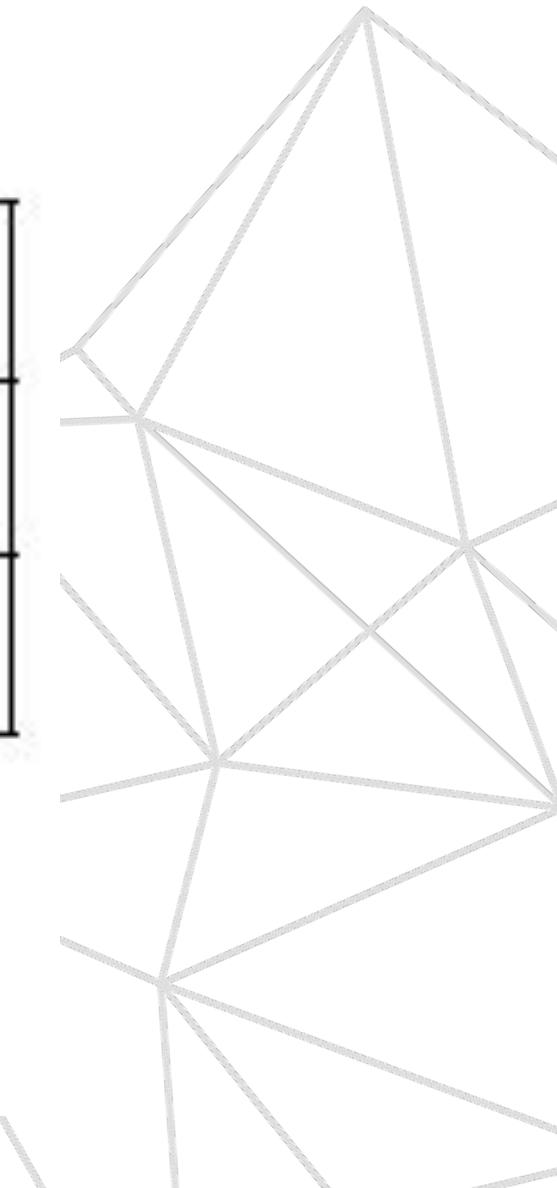
Los filtros se aplican desplazando consecutivamente el filtro una sola posición ($s = 1$), pero se puede aplicar un salto diferente.

- Ejemplo para $s = 2$

5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0

$$\begin{matrix} 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \end{matrix} * \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} = \begin{matrix} 0 & & \\ & & \\ & & \\ & & \\ & & \\ & & \end{matrix}$$

Filtro





Stride

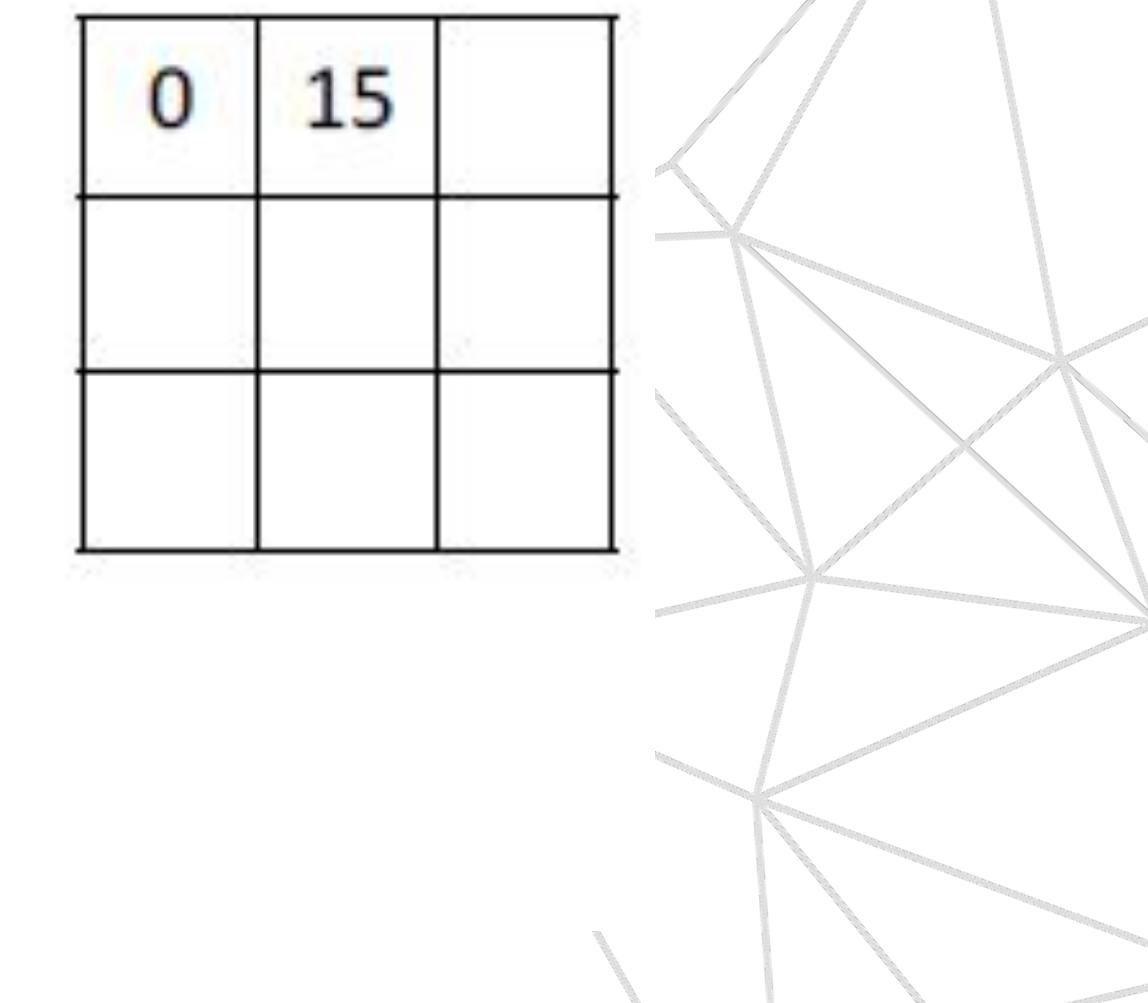
Los filtros se aplican desplazando consecutivamente el filtro una sola posición ($s = 1$), pero se puede aplicar un salto diferente.

- Ejemplo para $s = 2$

5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0

$$\begin{matrix} 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \end{matrix} * \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} = \begin{matrix} 0 & 15 & \\ & & \end{matrix}$$

Filtro





Stride

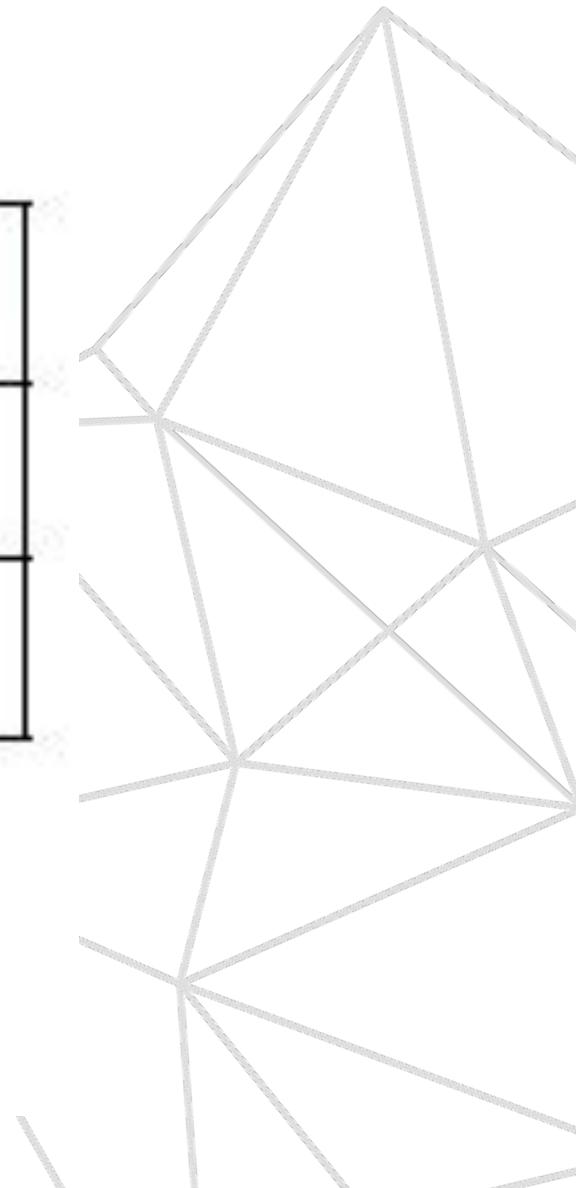
Los filtros se aplican desplazando consecutivamente el filtro una sola posición ($s = 1$), pero se puede aplicar un salto diferente.

- Ejemplo para $s = 2$

5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0

$$\begin{matrix} 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \end{matrix} * \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} = \begin{matrix} 0 & 15 & 0 \\ & & \\ & & \end{matrix}$$

Filtro





Stride

Los filtros se aplican desplazando consecutivamente el filtro una sola posición ($s = 1$), pero se puede aplicar un salto diferente.

- Ejemplo para $s = 2$

5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0

$$\begin{matrix} * & \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} & = & \begin{matrix} 0 & 15 & 0 \\ 0 & & \\ & & \end{matrix} \end{matrix}$$

Filtro

0	15	0
0		





Stride

Los filtros se aplican desplazando consecutivamente el filtro una sola posición ($s = 1$), pero se puede aplicar un salto diferente.

- Ejemplo para $s = 2$

5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0
5	5	5	5	0	0	0

$$\begin{matrix} 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 \end{matrix} * \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} = \begin{matrix} 0 & 15 & 0 \\ 0 & 15 & 0 \\ 0 & 15 & 0 \end{matrix}$$

Filtro





Convolución

Para una imagen de dimensión $n \times n$

- Con padding de tipo p

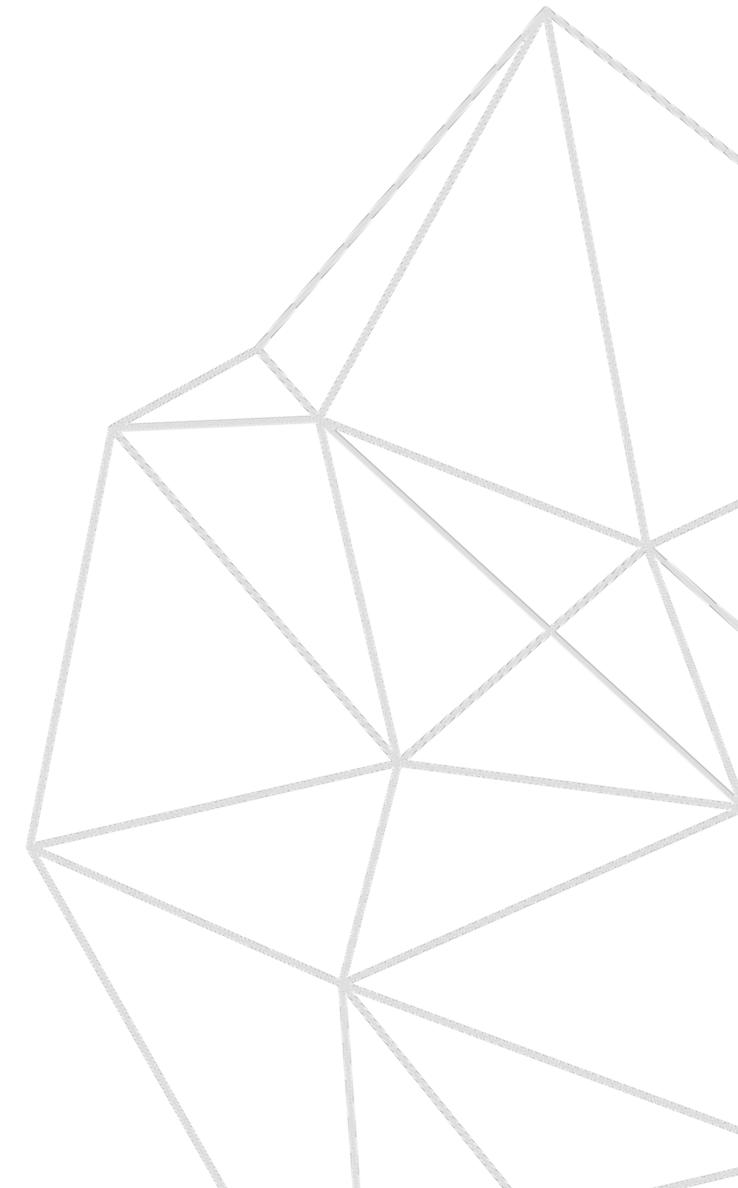
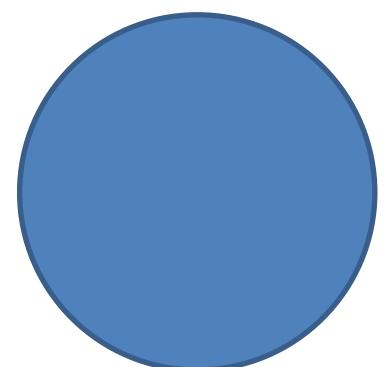
Al que se aplica un filtro de dimensión $f \times f$

- Con un stride s

La imagen resultante será de dimensión

$$\frac{n+2p-f}{s} + 1 \times \frac{n+2p-f}{s} + 1$$

¡Cuidado, el tamaño resultante puede ser una fracción!

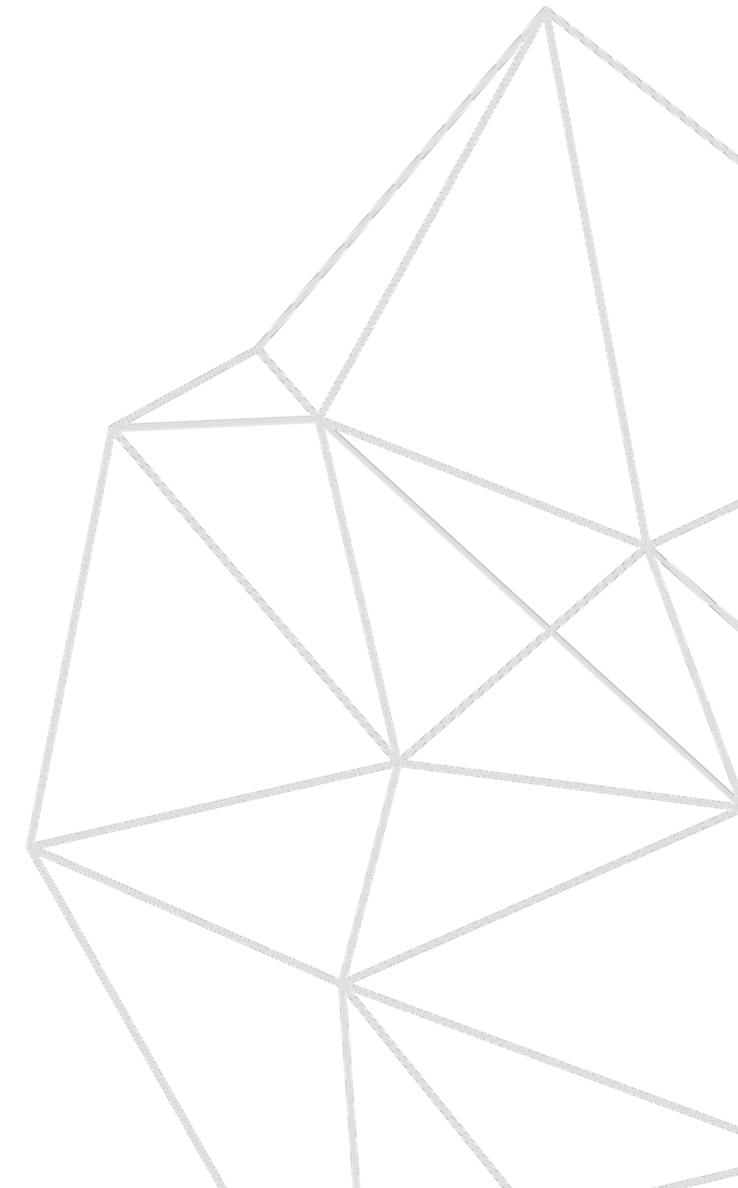
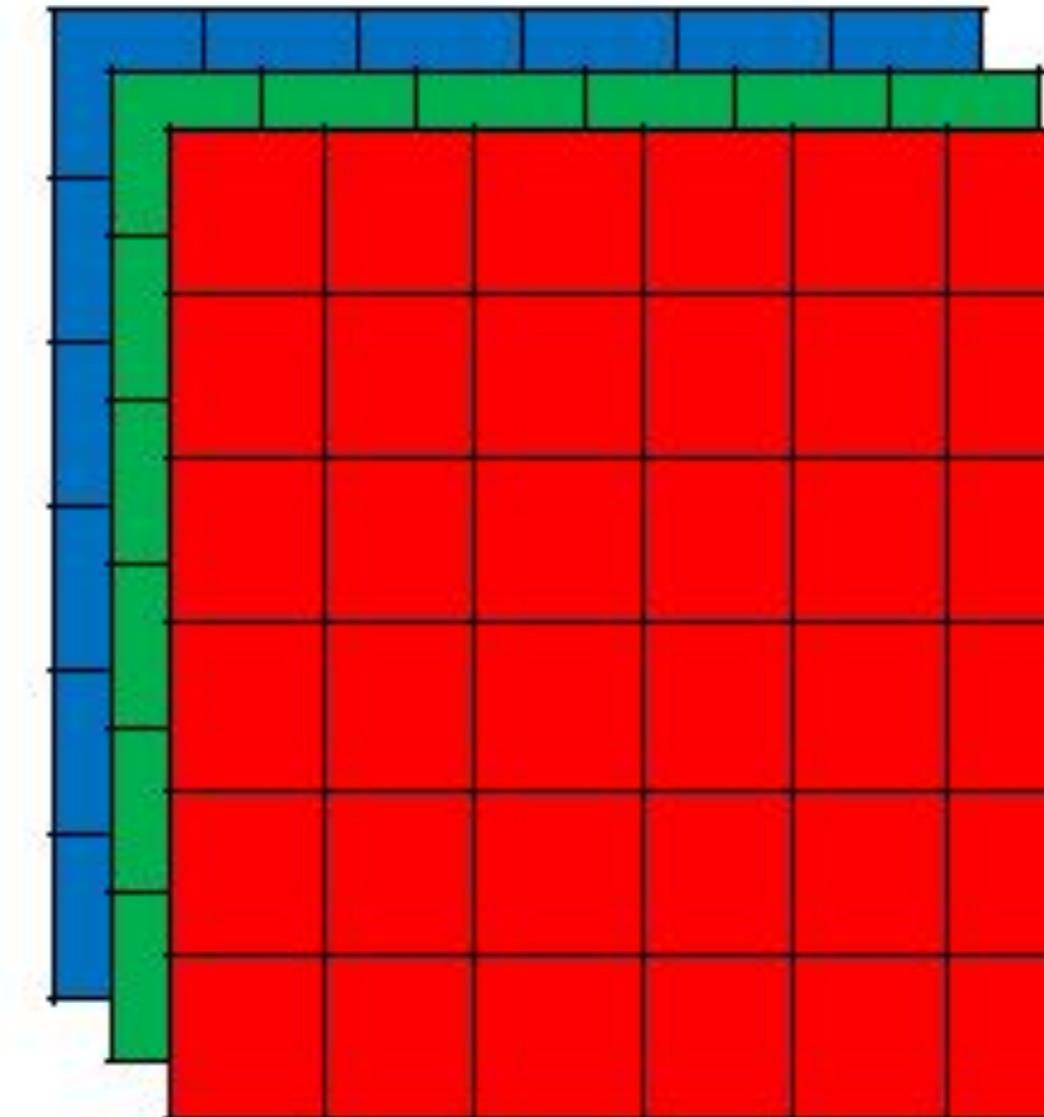




Convolución: Imágenes RGB

Las imágenes a color cuentan con tres canales (RGB)

- Red (rojo)
- Green (verde)
- Blue (azul)

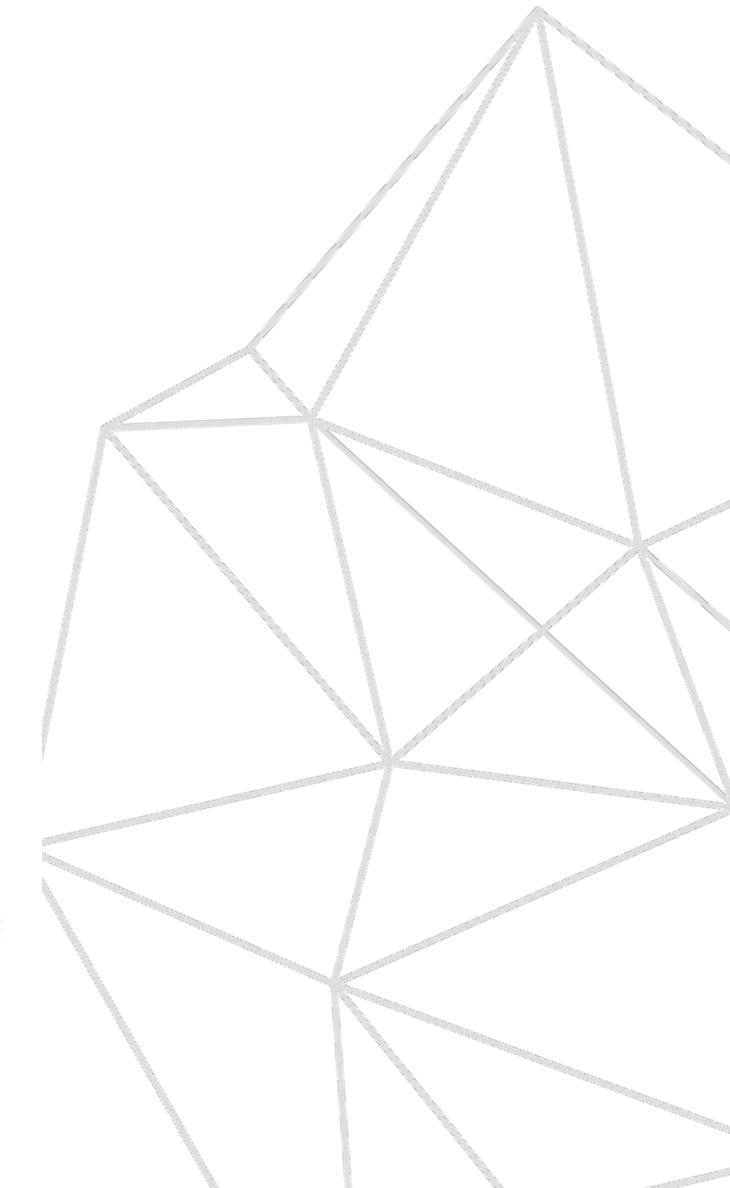
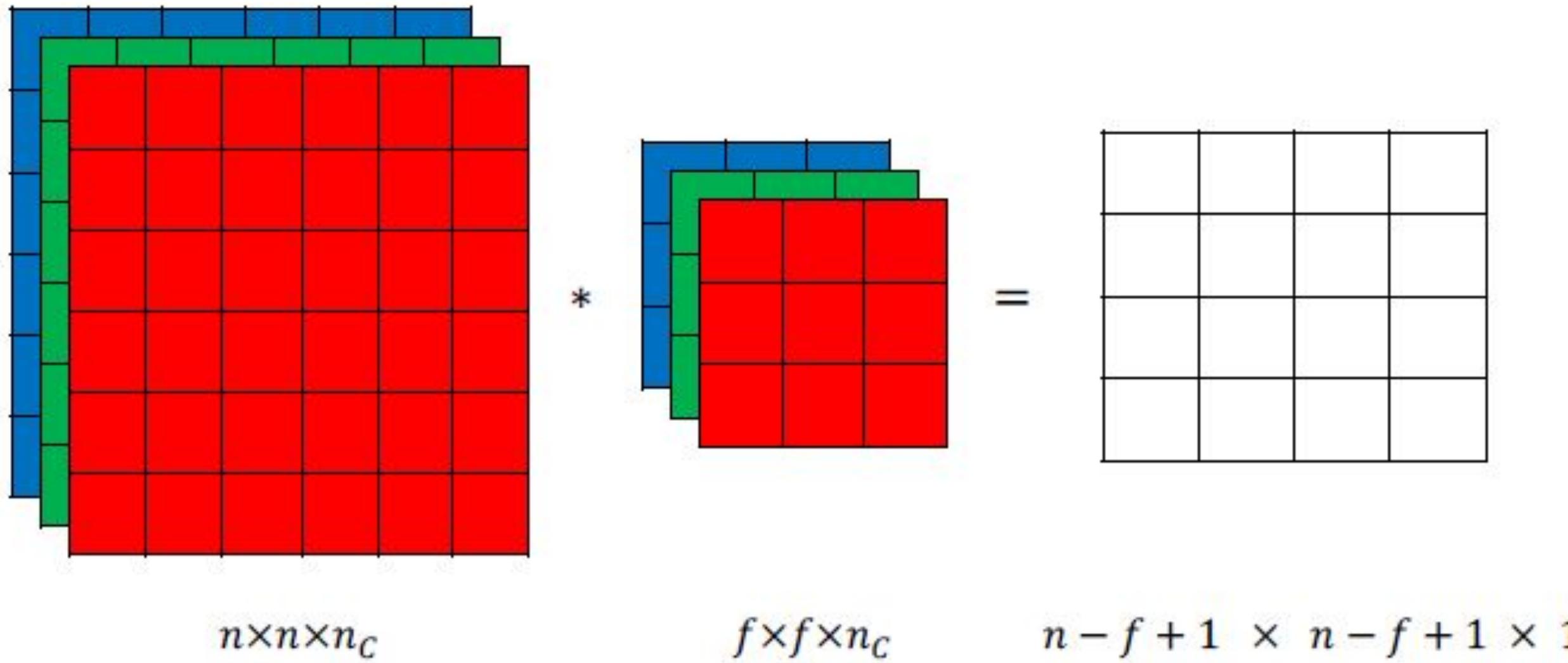
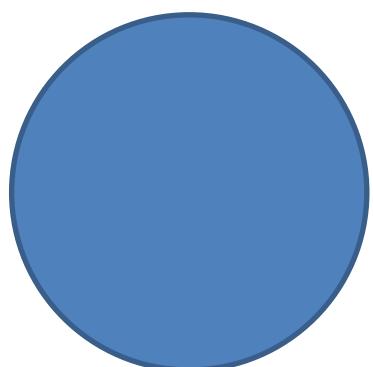




Convolución: Imágenes RGB

Las imágenes a color cuentan con tres canales (RGB)

Aplicaremos un filtro para cada canal C

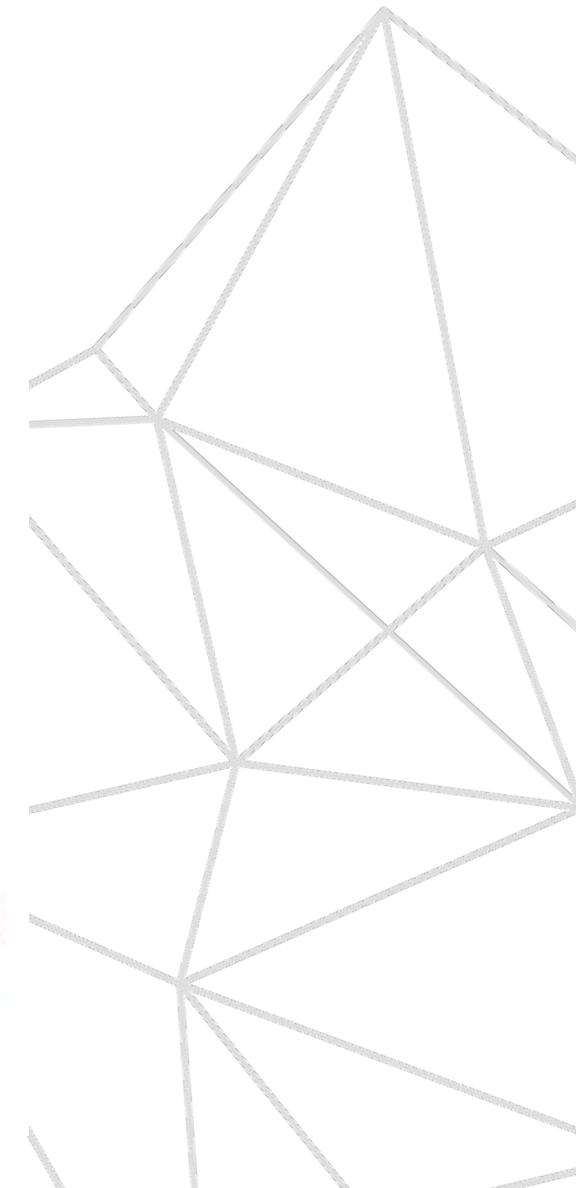
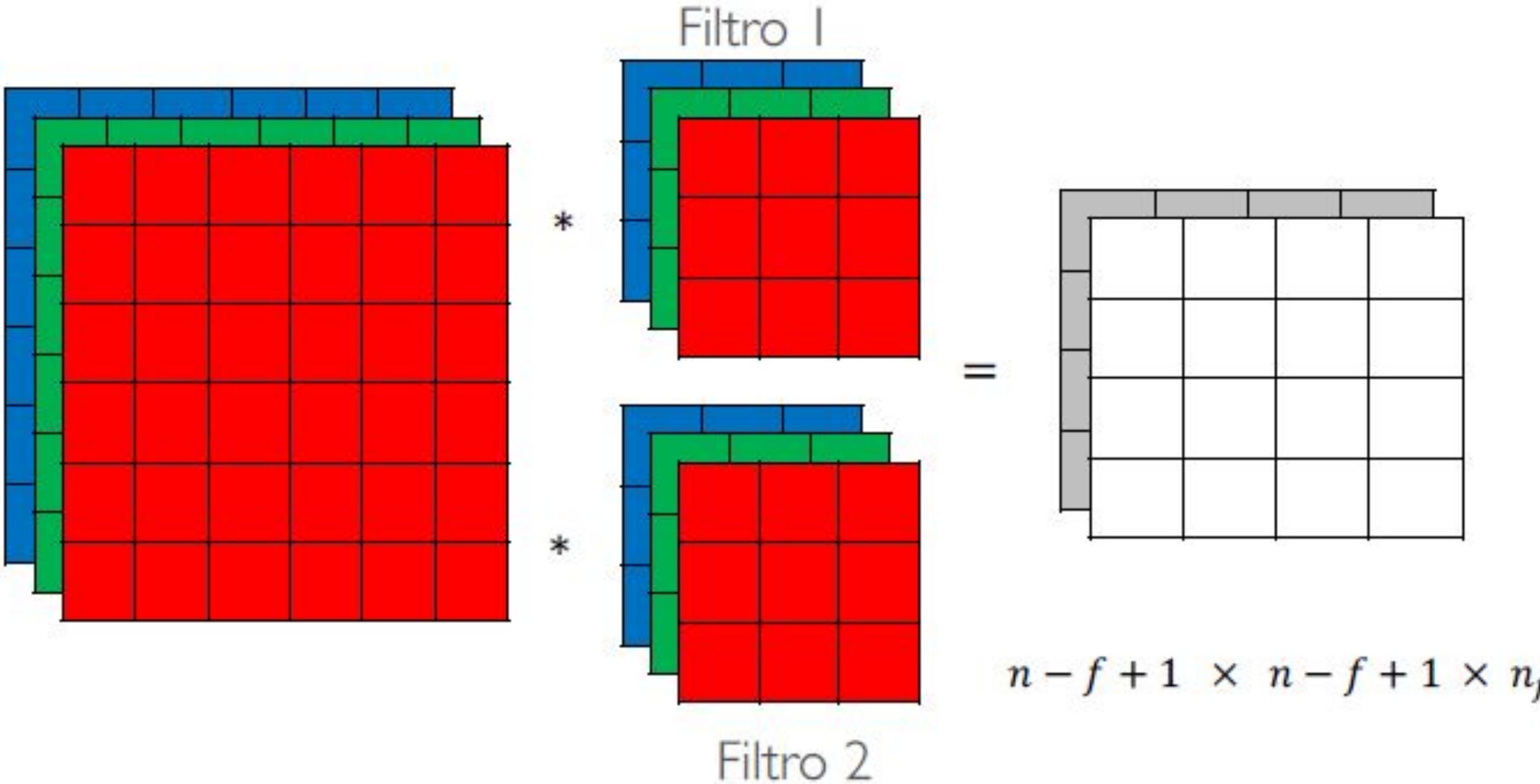




Convolución: Imágenes RGB

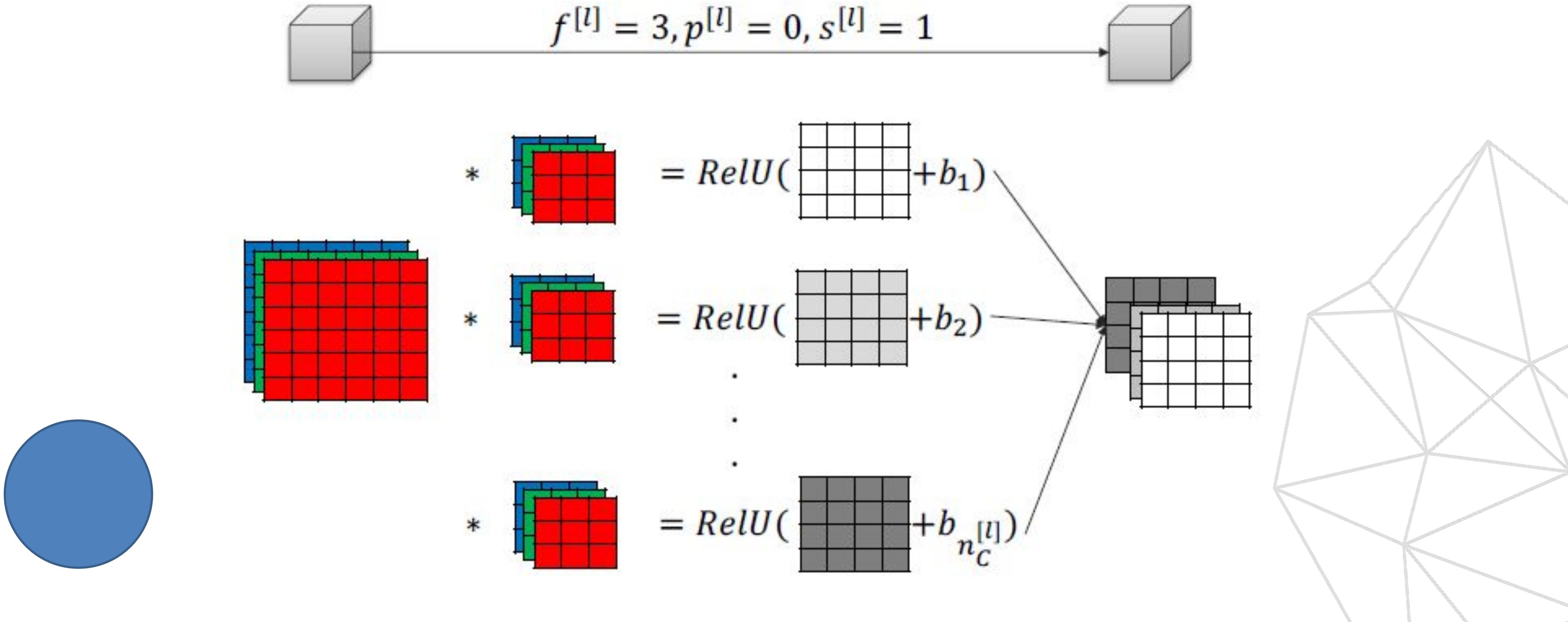
Podemos aplicar varios filtros.

- Por ejemplo, uno vertical y otro horizontal



CNN: Convolución

Capa de una Red Neuronal Convolucional con $n_c^{[l]}$ filtros



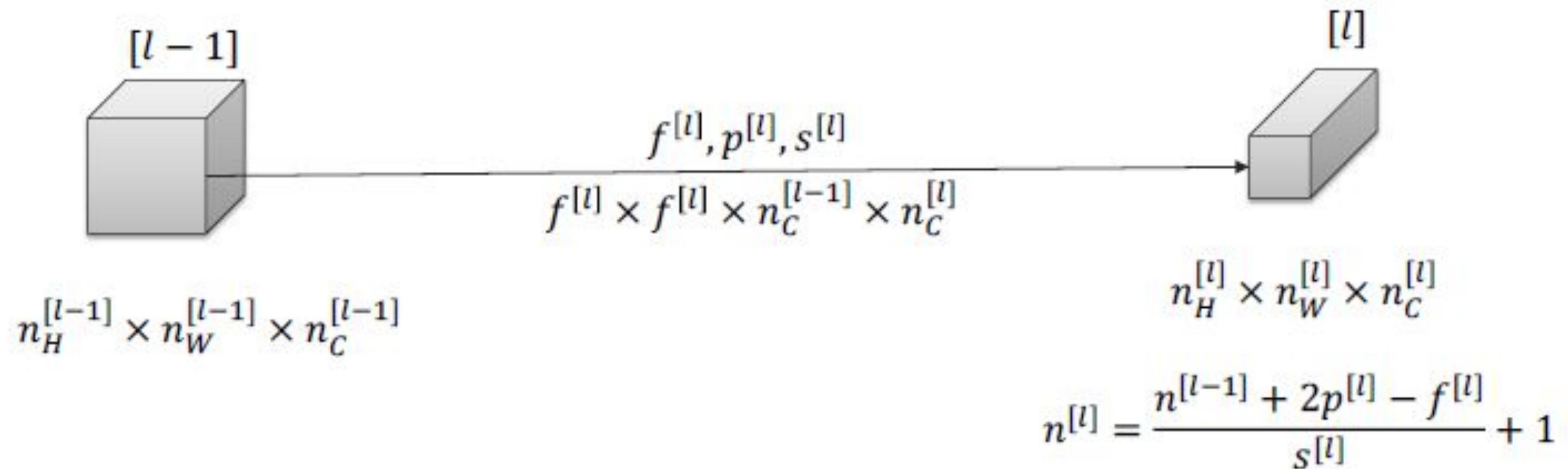


CNN: Convolución

¿Cuántos parámetros son necesarios para esta operación?

- Pesos: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
- Bías: $n_c^{[l]}$

Existe una **reducción** importante de los pesos. Dado que el patrón buscado por un filtro puede aparecer en más de un sitio. Los parámetros de los filtros son reutilizados en toda la imagen.

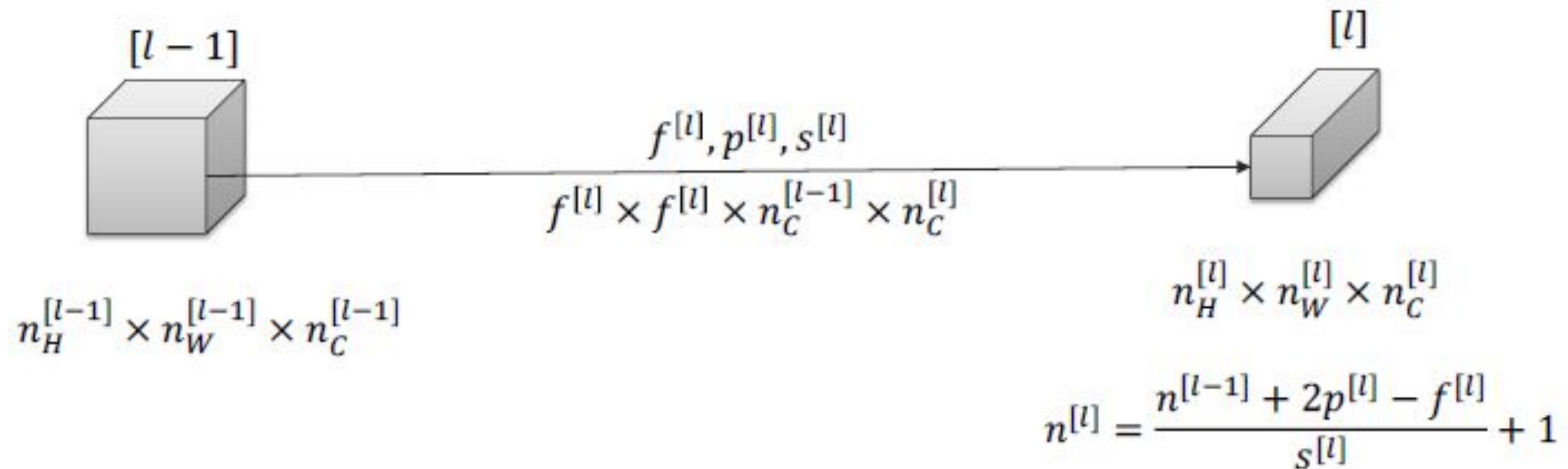




CNN: Convolución

¿Cuántos parámetros son necesarios para esta operación?

Es mejor utilizar varias capas con filtros de dimensión 3 que de mayor dimensión (a excepción de la primera capa). De este modo se **reduce** el número de **parámetros** y los **resultados son mejores**.

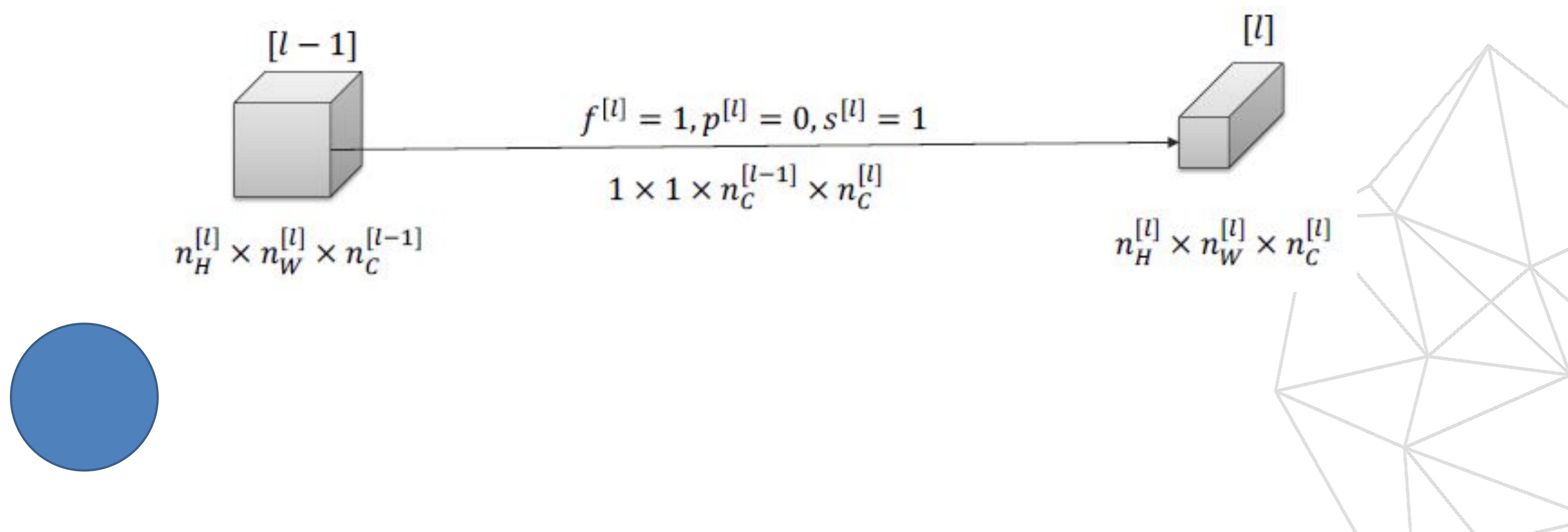




CNN: Convolución 1 x 1

Se puede aplicar un filtro de dimensión 1 x 1 para:

- Crear nuevos canales resultado de la combinación de los canales anteriores ponderados.
- Redimensionar el número de canales de una capa a otra.

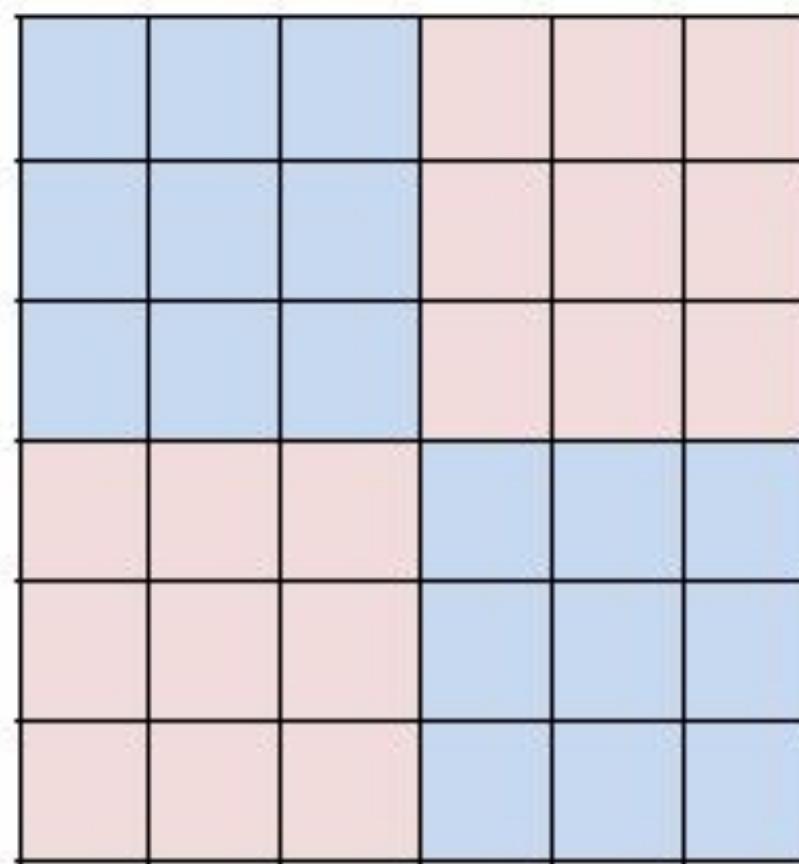
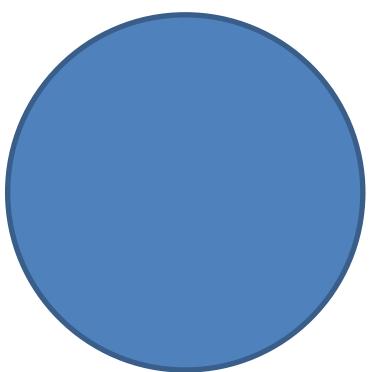




CNN: Pool

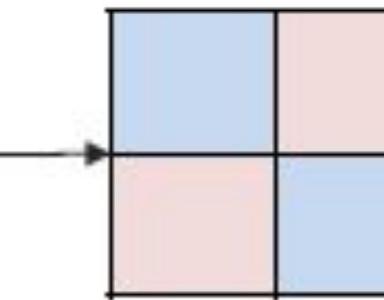
Propiedades:

- Permite resumir la información de una matriz.
- Invarianza: Permite hacer pequeñas translaciones, rotaciones y escalados en las imágenes de modo que la CNN sea robusta a pequeñas variaciones de un mismo objeto.



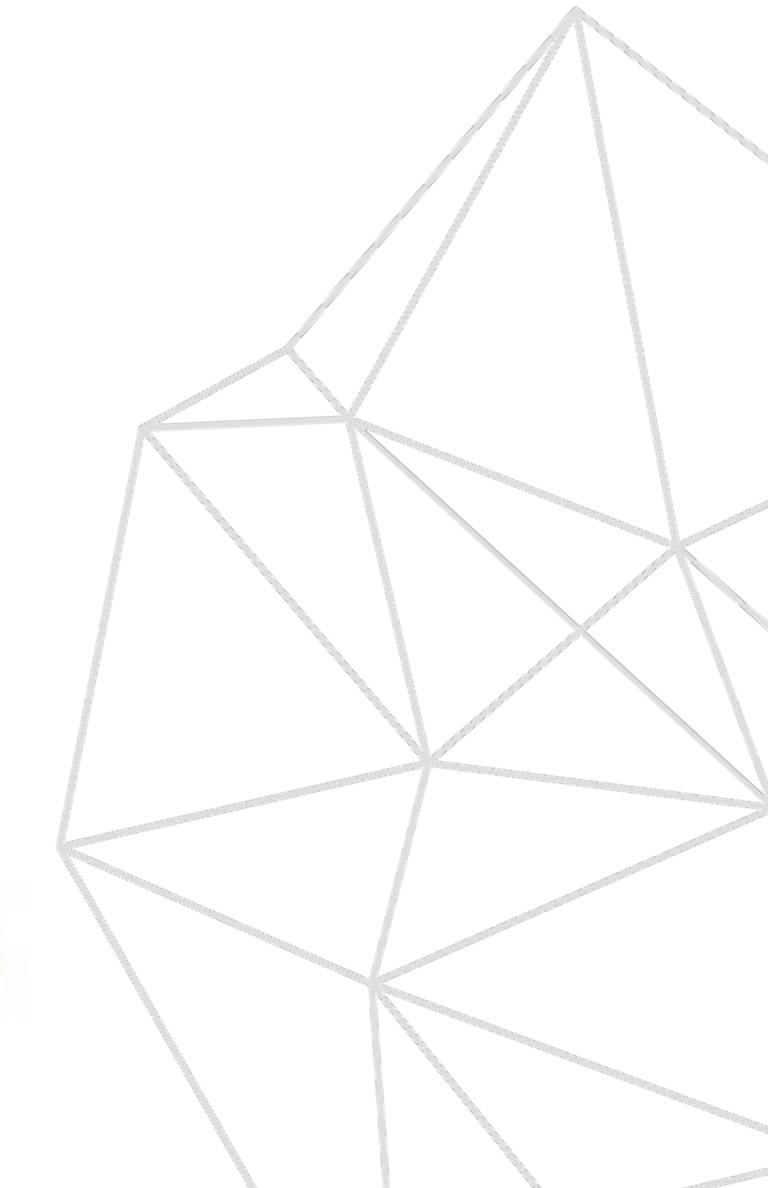
$$f'^{[l]} = 3, s'^{[l]} = 3$$

Pool



$$n'_H \times n'_W \times n_C^{[l]}$$

$$n'^{[l]} = \frac{n^{[l]} - f'^{[l]}}{s'^{[l]}} + 1$$

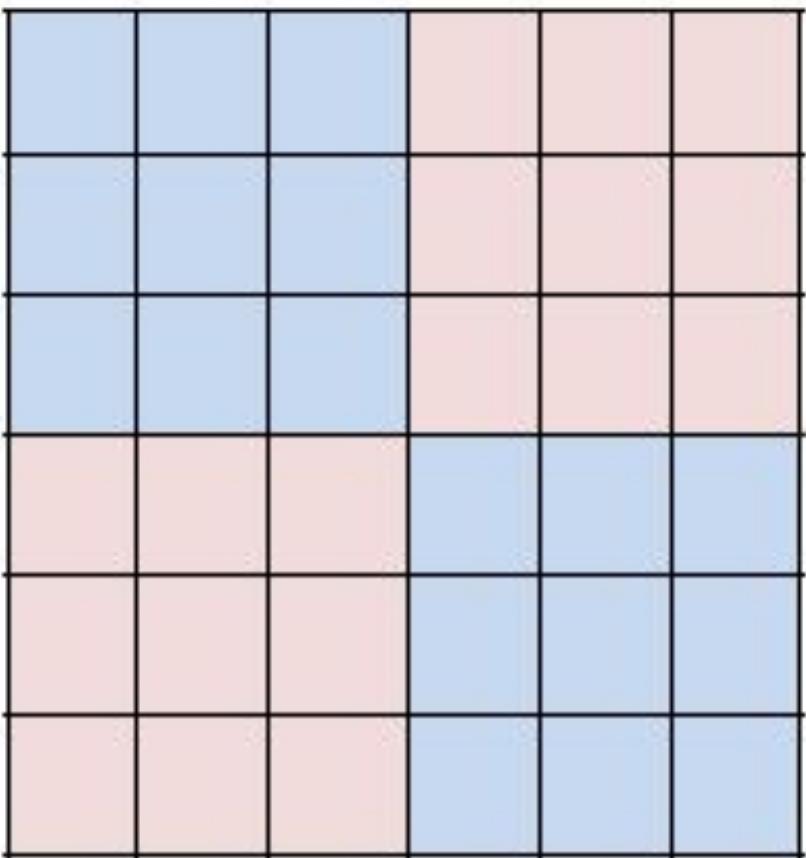
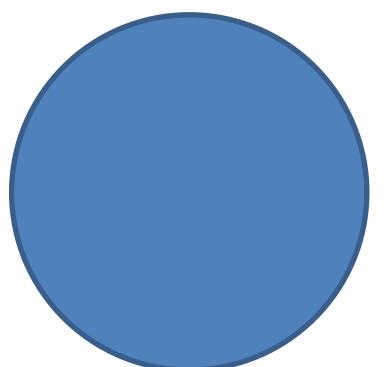




CNN: Pool

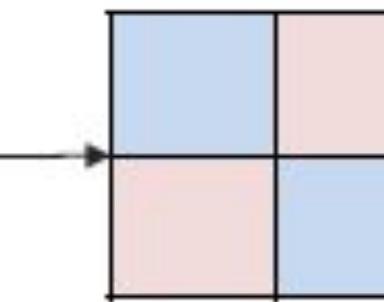
Operaciones habituales:

- **Máximo**
- Media



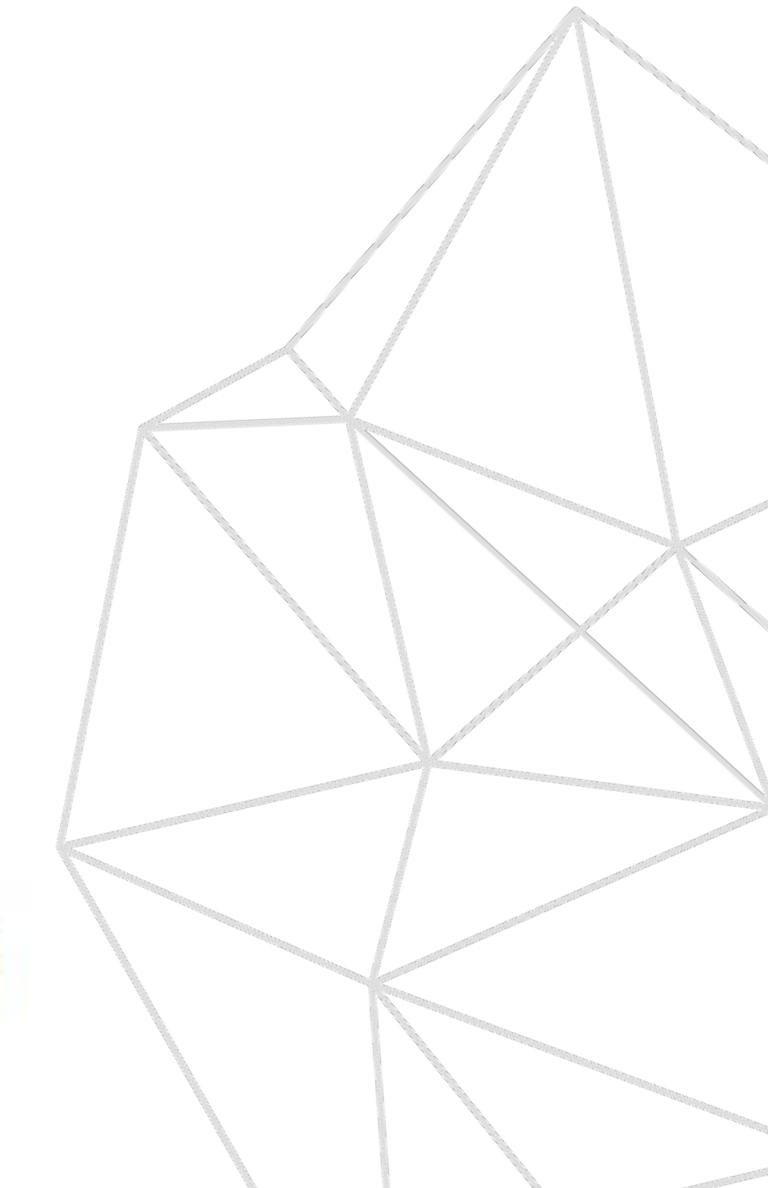
$$f'^{[l]} = 3, s'^{[l]} = 3$$

Pool



$$n'_H \times n'_W \times n_C^{[l]}$$

$$n'^{[l]} = \frac{n^{[l]} - f'^{[l]}}{s'^{[l]}} + 1$$





CNN: Pool

Operaciones habituales:

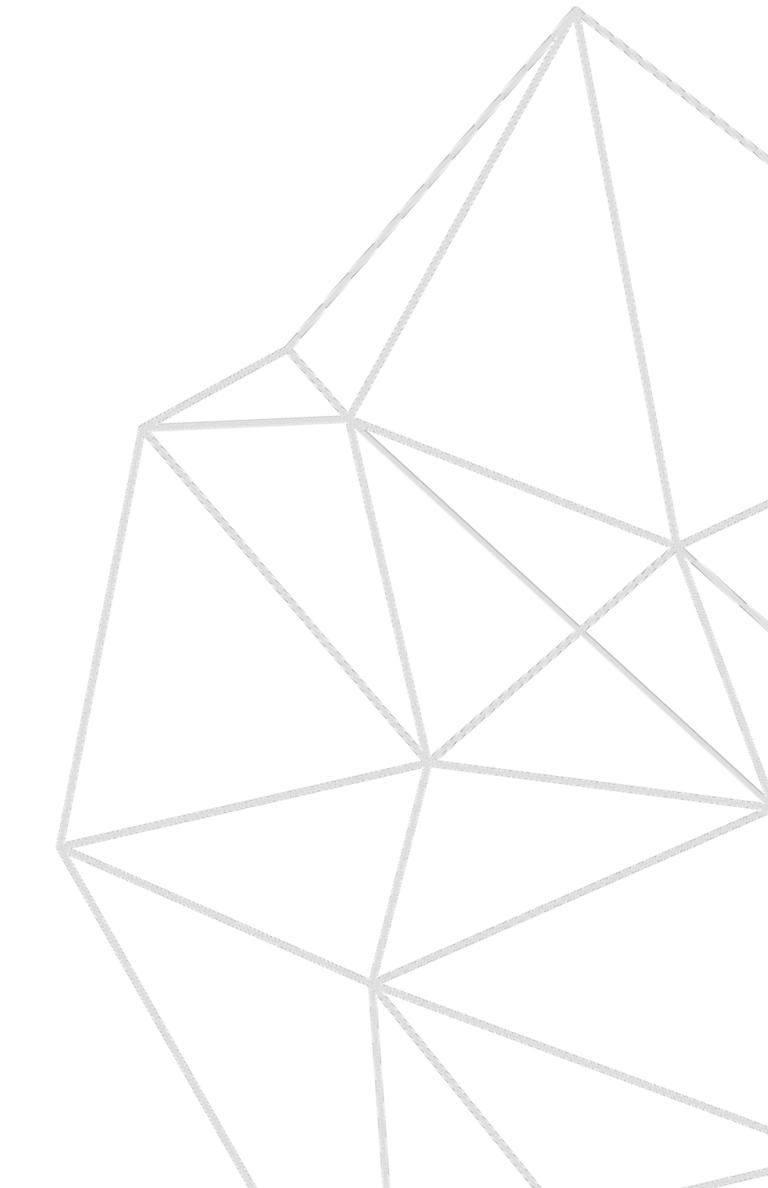
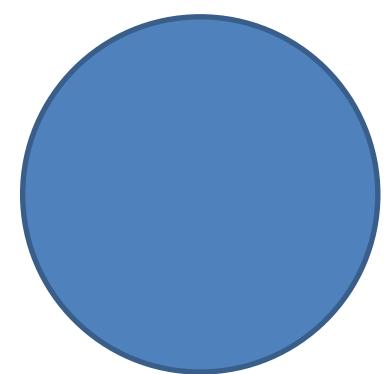
- **Máximo:** Más común.
 - Se queda con los rasgos más representativos y tiene mejores propiedades de invarianza.
- Media

1	7	3	8	2	5
1	2	6	5	4	1
3	2	6	1	9	2
6	3	4	3	5	3
4	4	1	3	1	9
2	5	1	5	6	7

$$f_p^{[l]} = 3, s^{[l]} = 3$$

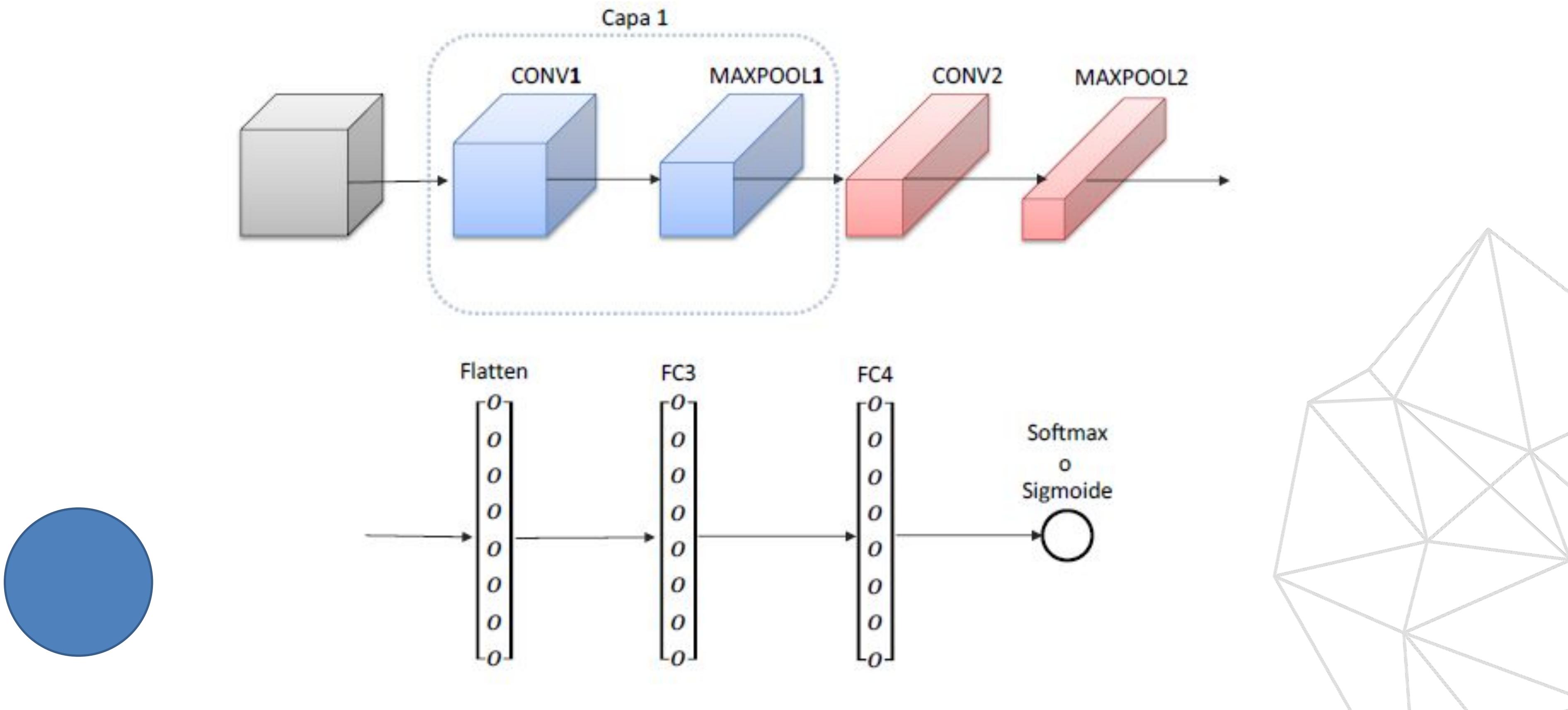
Max Pool

7	9
6	9





CNN: Red neuronal convolucional

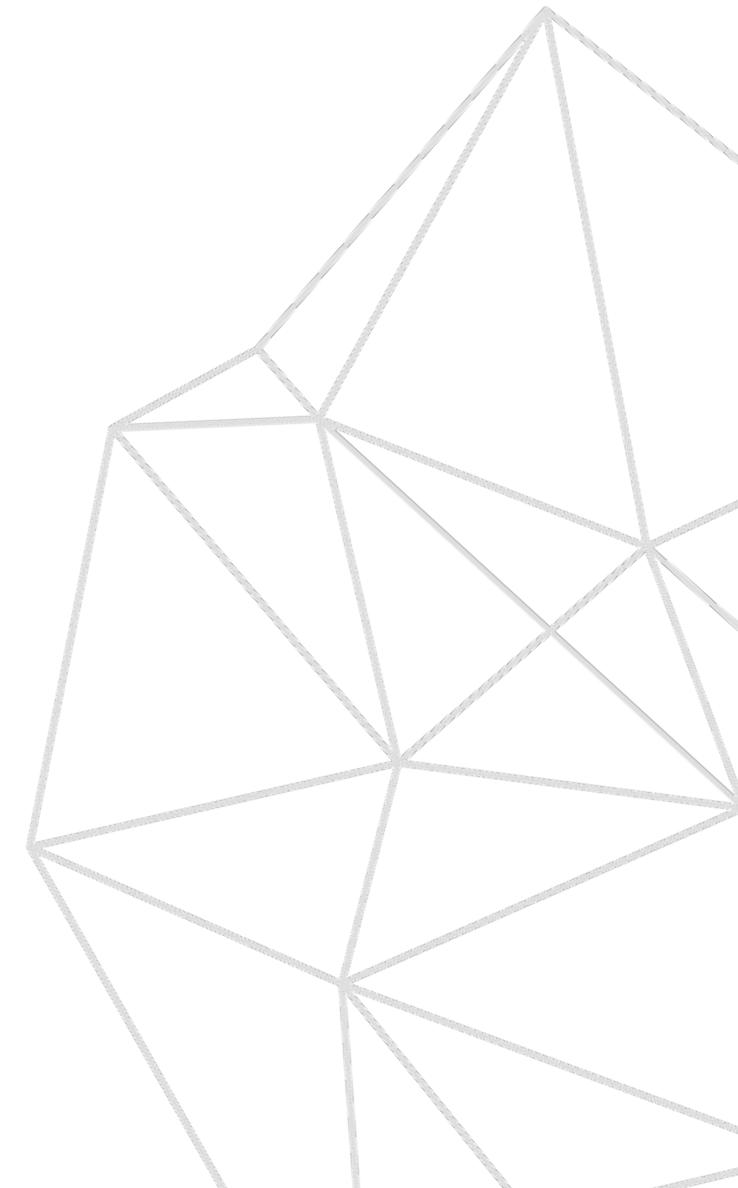
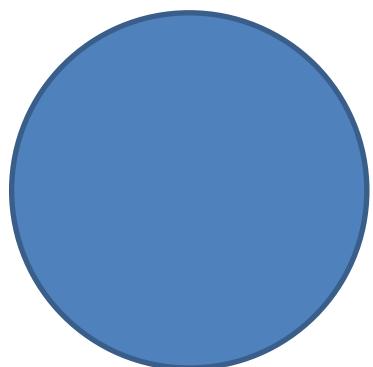




CNN: Redes populares

Básicas (de más pequeña a más grande):

- LeNet-5
 - Utilizado en bancos para obtener los números manuscritos.
- AlexNet
 - Regularización: Dropout y Data Augmentation.
- VGG
- ResNet: Residual Block
- Inception: Stacking



**Muchas gracias
por vuestra
atención.**

Carlos Moreno Morera
Consultor de IA en IBM
carmor06@ucm.es

