



Objetivos de aprendizaje de la sesión

-Comprender el funcionamiento básico de las Redes Neuronales

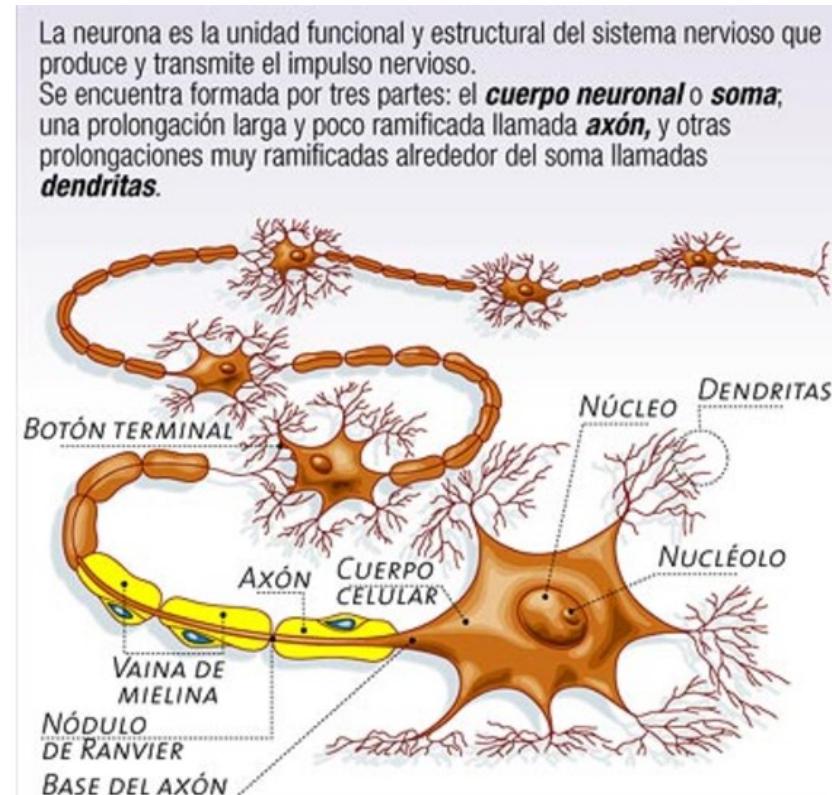
-Entender los conceptos de Deep Learning, Forward Propagation, Back Propagation, Función de Costo y Funciones de Activación

Redes Neuronales

El modelo de neuronas artificiales inspirado en el comportamiento biológico de las neuronas y en cómo se organizan formando la estructura del cerebro, publicado por McCulloch y Pitts en 1943, se considera el primer trabajo en el campo de la Inteligencia Artificial.

En una neurona, podemos reconocer diferentes partes:

- El cuerpo central, llamado soma, que contiene el núcleo celular.
- Una prolongación del soma, el axón.
- Una ramificación terminal, dendritas.
- Una zona de conexión entre una neurona y otra, conocida como sinapsis.

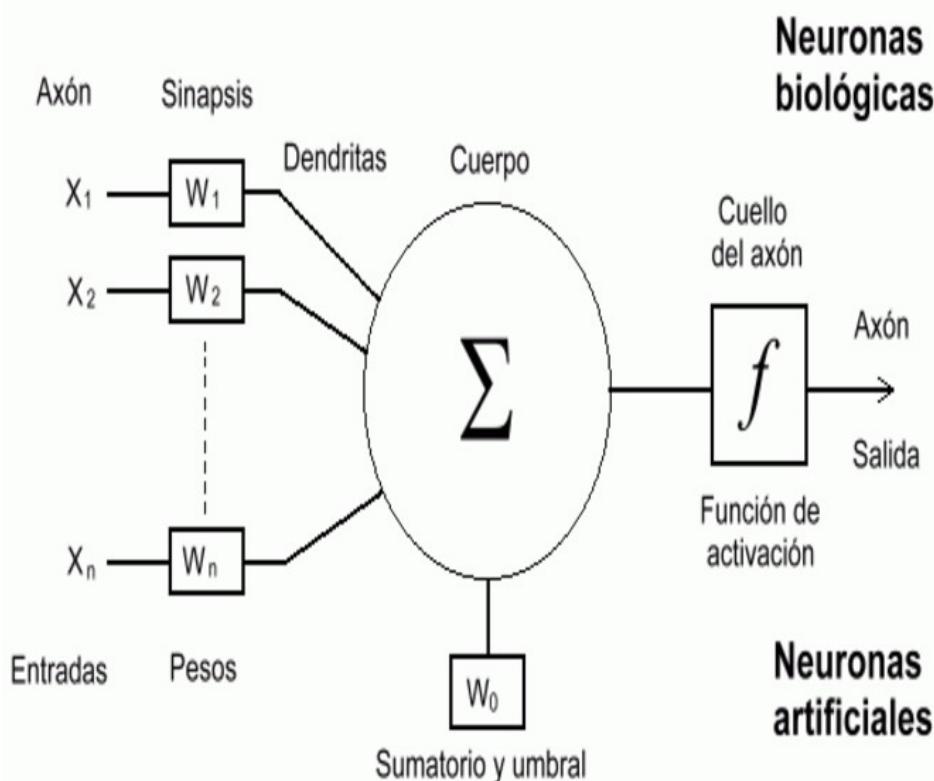


Trece años más tarde, en 1956, se acuñaría el propio término “Inteligencia Artificial” por John McCarthy, Marvin Minsky y Claude Shannon en una conferencia en Dartmouth. En 1958 Frank Rosenblatt diseña la primera red neuronal artificial, el Perceptrón.

Modelo neuronal con n entradas, que consta de

- Un conjunto de entradas x_1, \dots, x_n
- Los pesos sinápticos w_1, \dots, w_n , correspondientes a cada entrada
- Una función de agregación, Σ
- Una función de activación, f
- Una salida:

$$Y = f \left(\sum_{i=0}^n w_i x_i \right)$$



Si por ejemplo, se quisiera determinar si va a llover o no, y sabemos que esto depende esencialmente de la diferencia de temperatura entre el aire cercano a la superficie y el aire en la altura, que debe ser más frío; y por otra parte, también depende de la humedad, con lo que, si se dan ambas variables llueve.

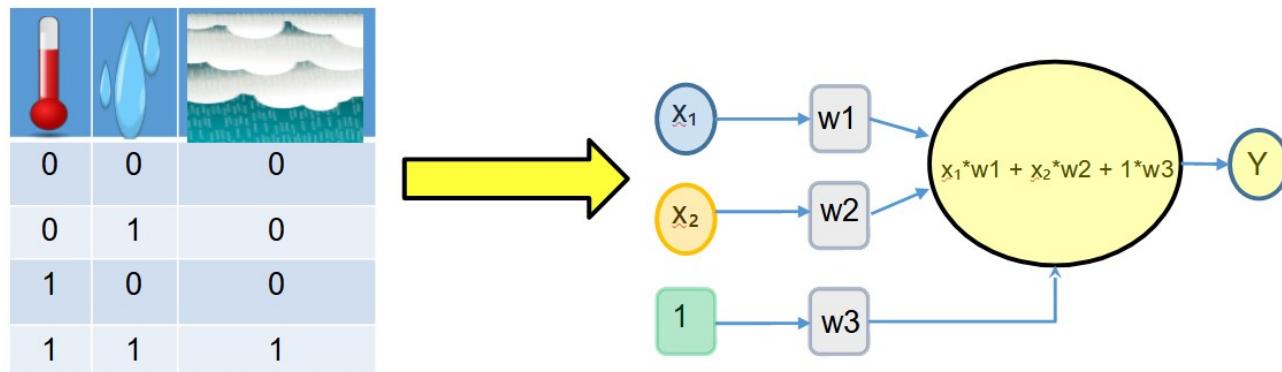
Las dos variables son: x_1 : Diferencia de temperatura. x_2 : Humedad

Teniendo en cuenta que cada una es binaria, es decir si hay la diferencia de temperatura necesaria o no, y si hay la suficiente humedad, entonces se puede armar la tabla:



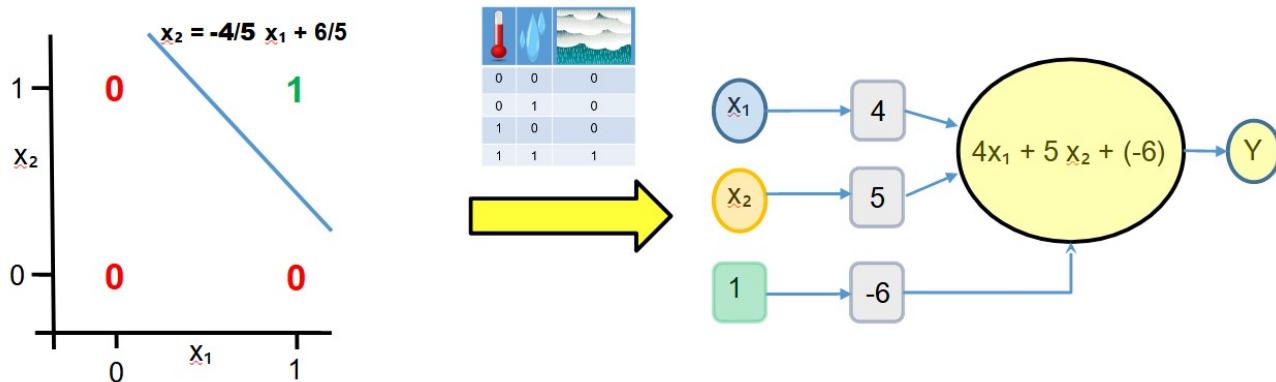
No	No	No
No	Sí	No
Sí	No	No
Sí	Sí	Sí

Convirtiendo los "Sí" y "No" en "1" y "0" respectivamente nos queda el siguiente esquema:



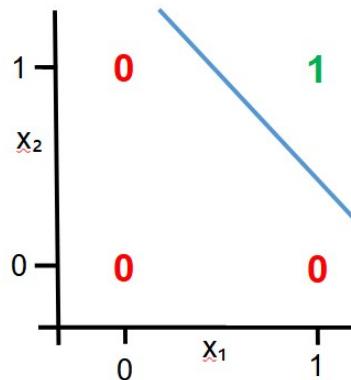
Se pueden ver las 2 entradas, más un valor de sesgo ó "bias", con su entrada que siempre vale 1. Por otra parte, también están los pesos sinápticos, es decir, los ponderadores (w_1 , w_2 y w_3).

El siguiente paso, es encontrar los valores de w_1 , w_2 y w_3 que separen correctamente las clases:

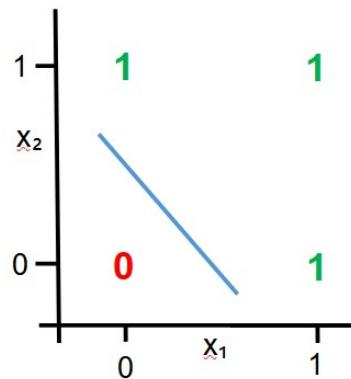


La recta $x_2 = -4/5 x_1 + 6/5$ logra separar correctamente las clases. Notar que cuando x_1 y x_2 tienen el valor 1, la sumatoria es 3, es decir, mayor a 0, ante cualquier otra combinación de valores de x_1 y x_2 el resultado es menor a 0. Entonces, una neurona artificial, en su núcleo, es un problema de **Regresión Lineal**.

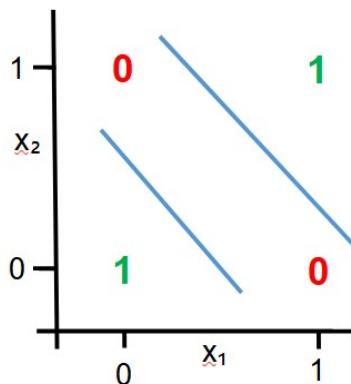
Por analogía, podemos ver que esta gráfica, corresponde a la tabla de verdad de las compuertas lógicas **AND**



Así mismo, se podría presentar un problema, cuya solución corresponda con la compuerta lógica **OR**

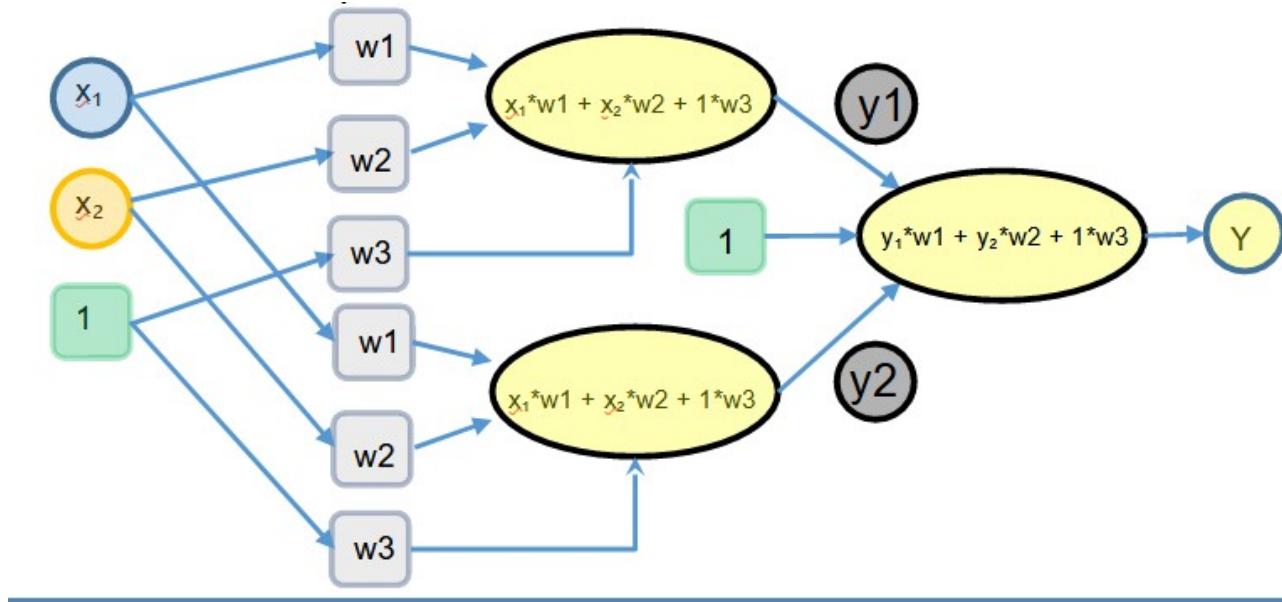


Pero que pasa con la compuerta lógica **XOR**, mediante la gráfica, se puede ver que no es posible separar las clases con una sola recta.

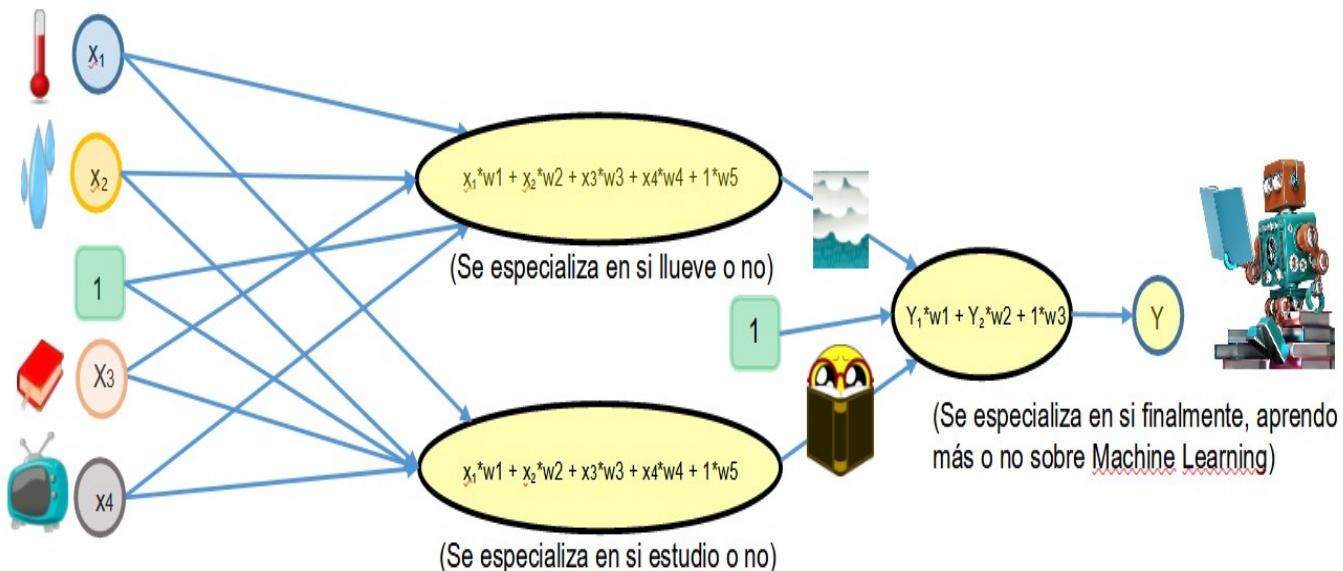


Este problema existe desde que se formuló la neurona artificial, y dio origen a las redes neuronales, ya que,

con dos neuronas, sí es posible resolverlo:

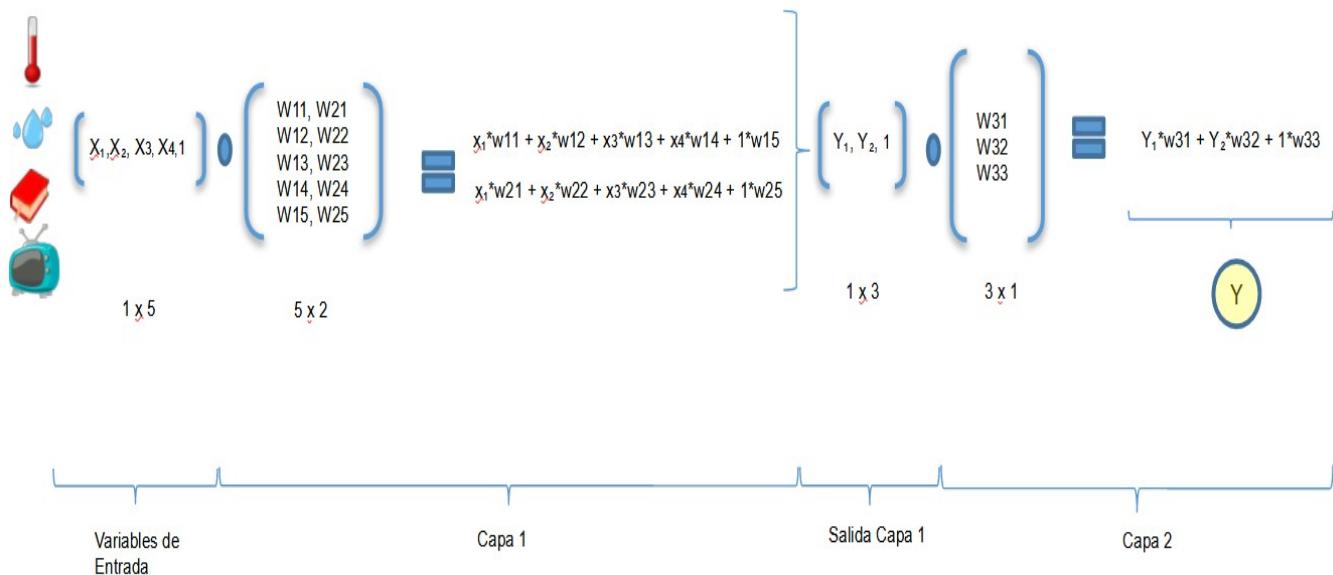


- Agregar capas a la red, permite obtener **conocimiento jerarquizado**.
- Siguiendo el ejemplo, de si va a llover o no, podemos querer predecir información más compleja, como por ejemplo si podremos aprender más sobre Machine Learning, agregando dos entradas más:
 - Voy a leer el material sobre el tema que tengo disponible.
 - Me voy a quedar mirando tele.



- Notar que ahora, es una red de 3 neuronas, con 13 w

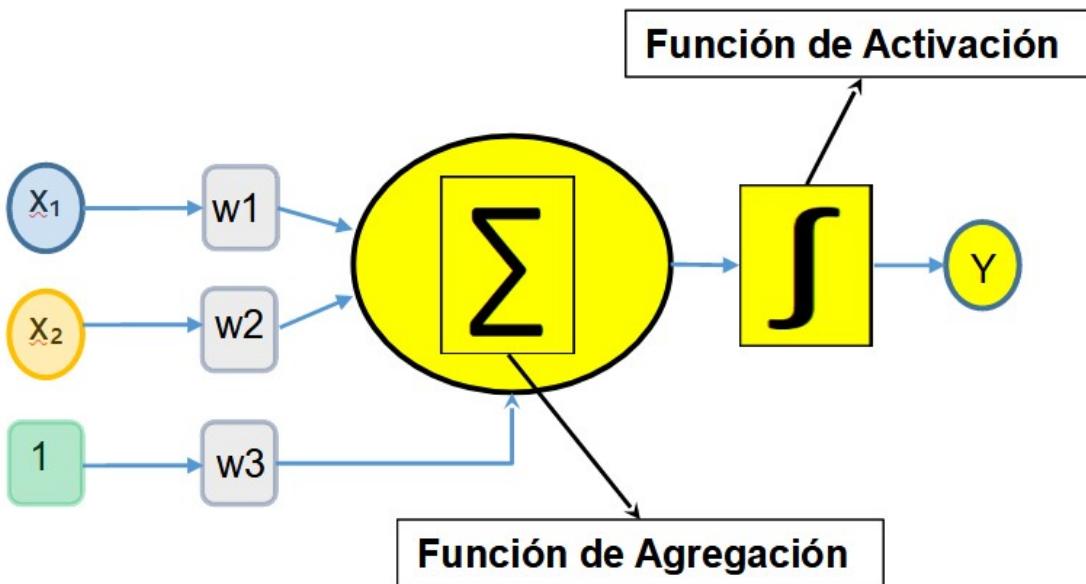
A través del álgebra matricial, este proceso se puede representar como el **producto escalar**:



Sin embargo, lo que implementa la función de agregación, es una función lineal. Lo que hace que una neurona artificial no sea sólo una función lineal es la **función de activación**, que consiste en, dado un valor umbral, arrojar una salida 0 ó 1:

Para nuestro ejemplo, la función de activación puede definirse:

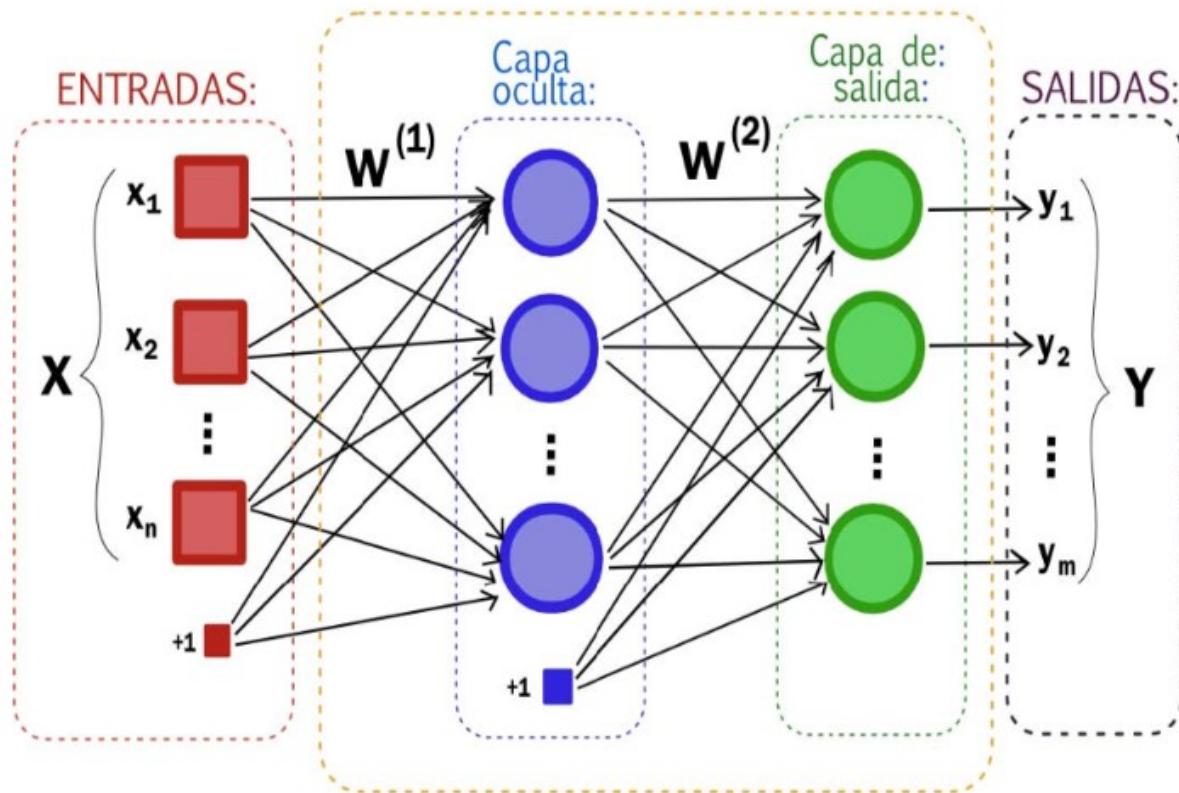
$$Y = \begin{cases} 1, & \sum > 0 \\ 0, & \sum < 0 \end{cases}$$



Deep Learning

Las neuronas se agrupan en capas, la primera capa (de entrada) tiene tantas neuronas como variables de entrada tiene el planteo del problema, mientras que la ultima capa tantas neuronas como salidas haya, si es binaria, puede ser sólo una neurona. Las capas que están entre la capa de entrada y la de salida, se llaman capas ocultas. Cada neurona tiene sus propios pesos/parámetros. En aplicaciones comunes suelen ser desde miles a millones de parámetros para toda la red. Esa obtención de conocimiento jerarquizado, es lo que da

nombre al **Aprendizaje Profundo ó Deep Learning**, y requiere de encontrar esos pesos W de manera eficiente, bajo la condición de realizar correctamente una tarea objetivo.

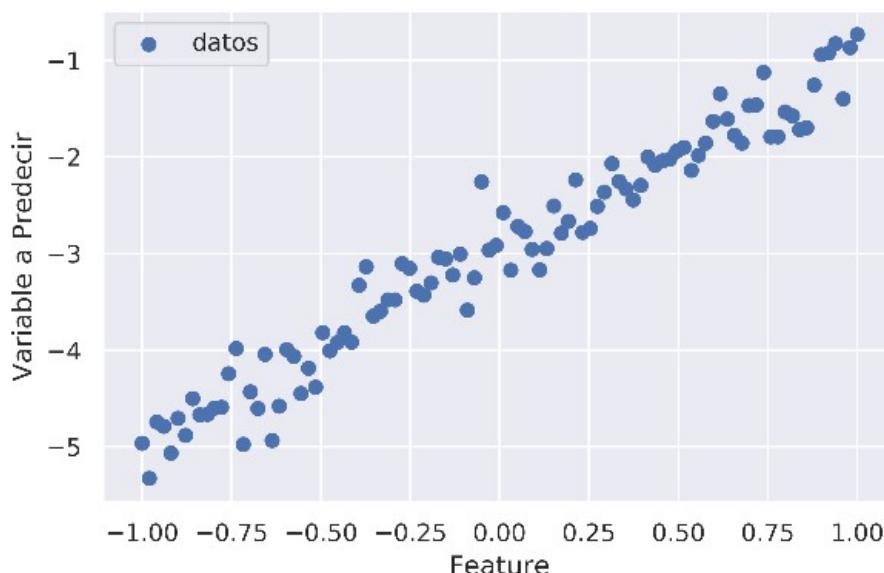


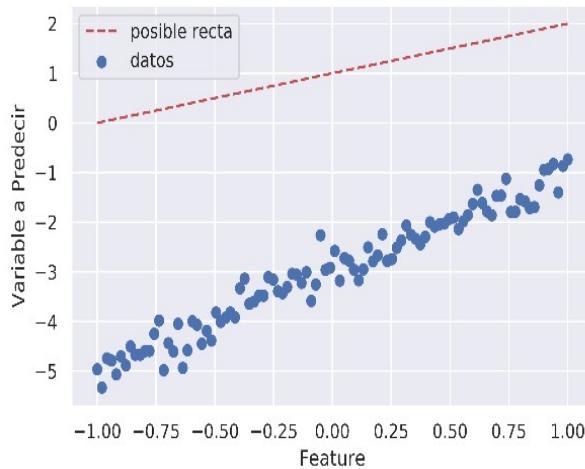
Aprendizaje de Redes Neuronales

Es un problema de Regresión Lineal, por ese motivo buscamos entonces $Y = mX + b$ que mejor ajuste a los datos

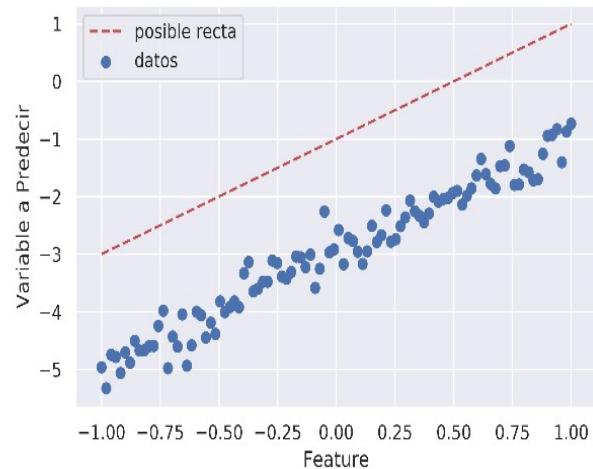
- m : pendiente
- b : ordenada al origen

1. Es necesario definir cuál es el mejor ajuste a los datos.
2. Se puede probar con distintos valores de m y b , y quedarse con el que mejor ajusta.



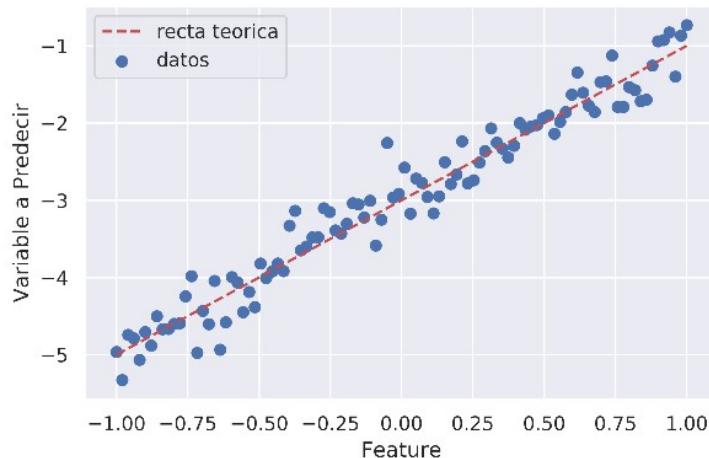


$$m = 1, b = 1$$



$$m = 2, b = -1$$

Con un modelo paramétrico ($y = mx + b$, m y b parámetros), se puede hacer una búsqueda por fuerza bruta para encontrar el m y b. Y guiar la búsqueda con una función de costo. Muy similar a GridSearch, pero en este caso es para buscar los parámetros del modelo, no sus hiperparámetros. Ejemplo de hiperparámetro: grado del polinomio. Esta clase de problemas se conocen como problemas de **optimización**:



$$m = 2, b = -3$$

Descenso por Gradiente

Computacionalmente, es muy costosa la búsqueda de esos parámetros ya que crece con las dimensiones. Entonces, ¿cuáles son las opciones?

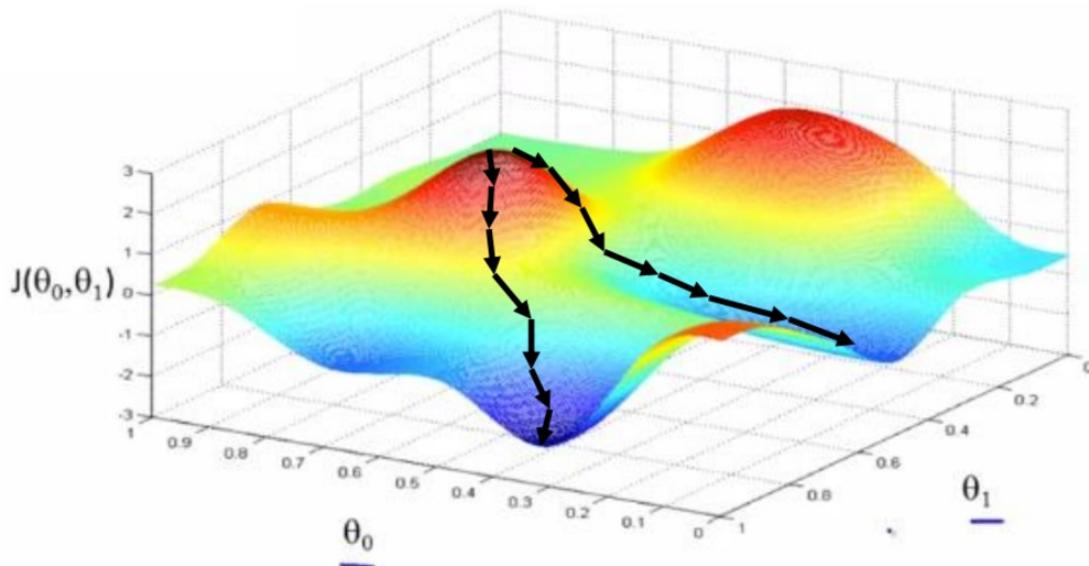
Formas analíticas (fórmulas matemáticas) que calculen exactamente el mínimo de la función de costo y a qué parámetros corresponde ese mínimo. Pocos casos pueden resolverse así. Ejemplo: cuadrados mínimos para la regresión lineal.

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

m y b , como vector

Datos, en forma matricial

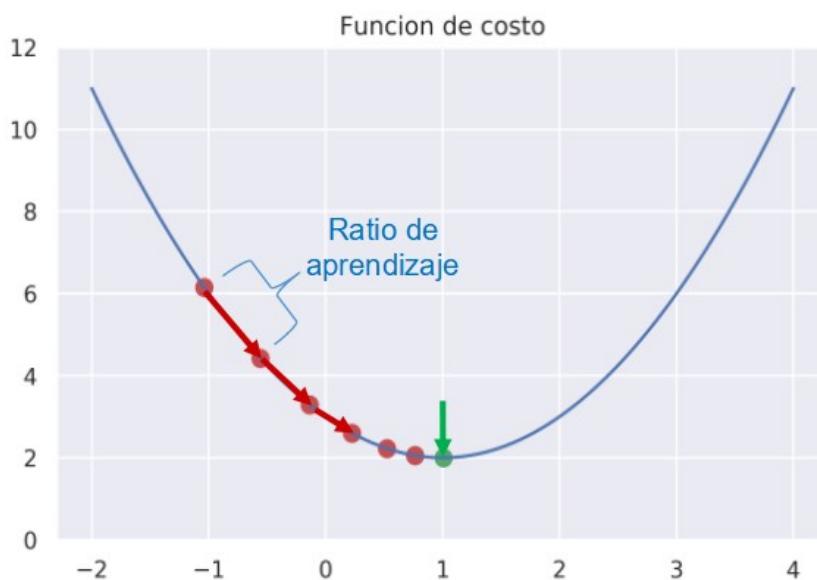
Mezclar los dos mundos, es decir, combinar la búsqueda en un espacio de parámetros con una guía que nos diga dónde buscar. **Descenso por Gradiente** consiste en buscar un mínimo global mediante iteraciones en las cuales se va descendiendo en la función de costo $J(\theta_0, \theta_1)$:



Se quiere explorar el mínimo de la función de costo, pero sin hacerlo de forma exhaustiva

Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos este paso hasta converger al mínimo
3. Nos fijamos la dirección de decrecimiento en ese punto. Técnicamente, **derivamos o calculamos el gradiente**.
4. Nos movemos según el **ratio de aprendizaje** (es el tamaño de la flecha)
5. **Actualizamos los valores de los parámetros.**



- **Es necesario definir una función de costo/pérdida.** La función de costo depende del problema (clasificación, regresión, etc).
- La función de costo es una **función de los parámetros de la red**.
- Los mejores parámetros de la red son aquellos que **minimicen la función de costo**.

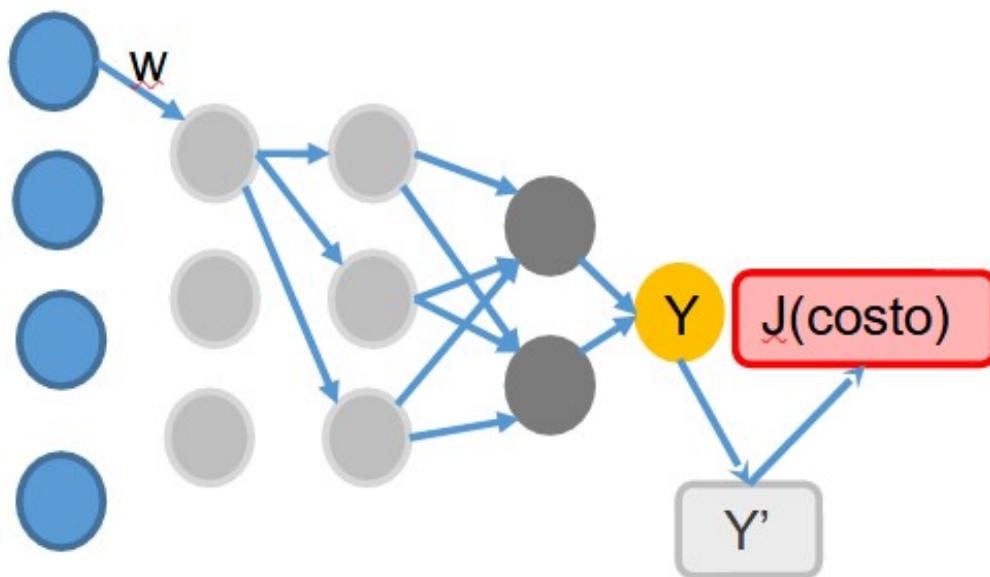
- Como explorar todo ese espacio de parámetros exhaustivamente (similar a Grid Search) es imposible, se utiliza una técnica que lo haga eficientemente. Esa técnica es Descenso por Gradiente.
- **Reescalar garantiza la convergencia del algoritmo de Descenso por Gradiente** en el entrenamiento.

Enlaces recomendados:

- [Regresión Lineal y Mínimos cuadrados ordinarios] (<https://www.youtube.com/watch?v=w2RJ1D6kz-o>)
- [¿Qué es el descenso por Gradiente?] (https://www.youtube.com/watch?v=A6FiCDoz8_4)
- [¿Qué es el descenso por Gradiente (2)?] (https://www.youtube.com/watch?v=-_A_AAxqzCg)
- <http://www.benfrederickson.com/numerical-optimization/>
- <https://www.desmos.com/calculator/l0puzw0zvm>

Fordward Propagation

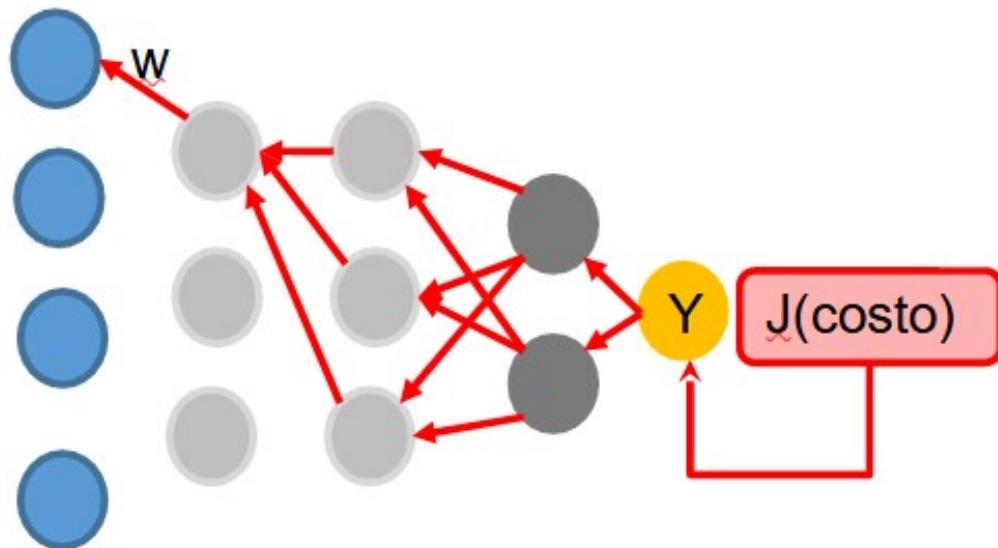
- Cada valor w , afecta a la siguiente capa, y por ende, a todo el resto de la red neuronal, por tal motivo tiene parte de la responsabilidad en la función de Costo final.
- Esa función de Costo o de pérdida, determina cuán lejos está el resultado de la red contra el resultado esperado.
- El Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.
- En cada una de esas iteraciones, tiene que calcular el costo. **El costo depende de las instancias de entrenamiento y de los parámetros que tengamos hasta ese momento.**
- Calcular el costo con las instancias de entrenamiento es lo que se conoce como **Forward propagation..**



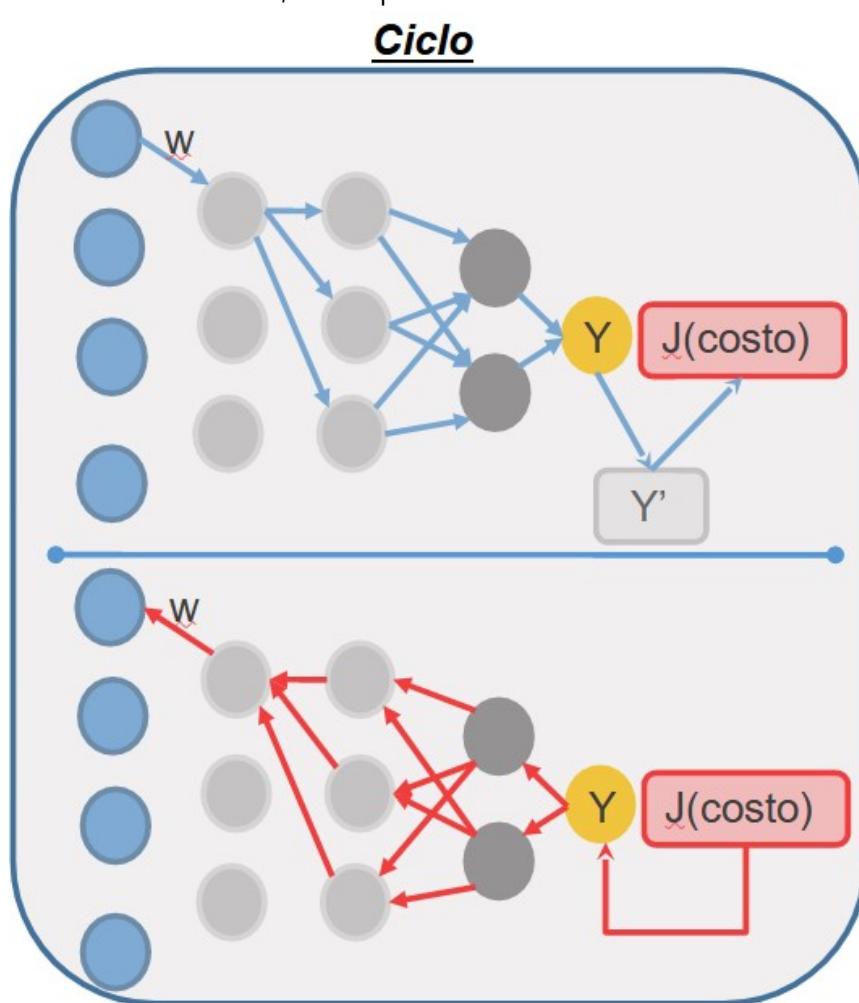
Back Propagation

Con el costo calculado, se actualizan los valores de los parámetros ponderadores de las entradas, los w , que se inicializan con valores aleatorios. 1) Se deriva el costo con respecto a la función de activación ó predicción. 2) Se deriva la predicción con respecto a la función de agregación. 3) Se deriva la función de agregación, con respecto a los parámetros w . 4) Se actualizan los parámetros.

- Para derivar la composición de funciones $J(Y(f(X)))$ se utiliza la denominada **regla de la cadena**.
- Así, se propaga esa derivada hacia atrás, hasta llegar a los parámetros w iniciales, pasando capa tras capa por toda la red..



- Entonces, con los datos de entrenamiento, la red ajusta sus parámetros, para adaptarse a la salida esperada.
- Se llama **epoch ó ciclo**, al proceso completo desde un input de entrada, hasta la consecución de la función de coste, y posterior aplicación del algoritmo de Backpropagation de todos los ejemplos de entrenamiento.
- El **tamaño de lote ó batch size** es el número de ejemplos de entrenamiento en un ciclo. Cuanto mayor sea el tamaño del lote, más espacio de memoria se necesitará.



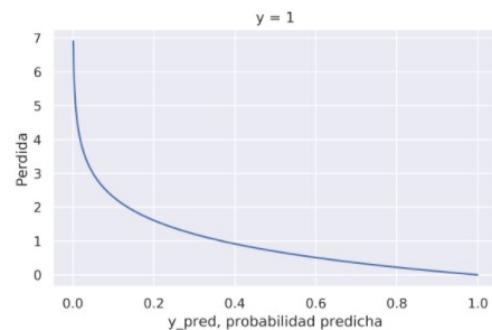
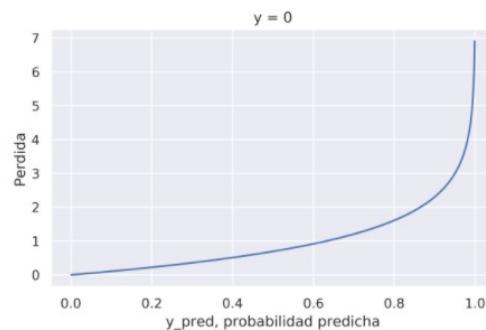
Enlace recomendado:

- [Un lego a la vez] (<https://medium.com/latinxinai/un-lego-a-la-vez-explicando-la-matem%C3%A1tica-de-como-las-redes-neuronales-aprenden-ae582ab91da6>)
- [Backpropagation] (https://www.youtube.com/watch?v=eNIqz_noix8)
- [Las Matemáticas de Backpropagation] (<https://www.youtube.com/watch?v=M5QHwkkHgAA>)
- [Aprendizaje profundo] (<https://www.youtube.com/watch?v=aircAravnKk>)
- [Descenso de Gradiente] (<https://www.youtube.com/watch?v=lHZwWFHWa-w>)
- [Playground Tensorflow] (<https://playground.tensorflow.org>)

Función de Costo – Entropía Cruzada

- Define una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta.
- Pérdida para una instancia, caso binario: etiquetas $y = 0$ y 1 .

$$L(\hat{y}, y) = -y * \log(\hat{y}) - (1 - y) * \log(1 - \hat{y})$$



- Pérdida para una instancia

$$L(\hat{y}, y) = -y * \log(\hat{y}) - (1 - y) * \log(1 - \hat{y})$$

- Costo para todas las instancias

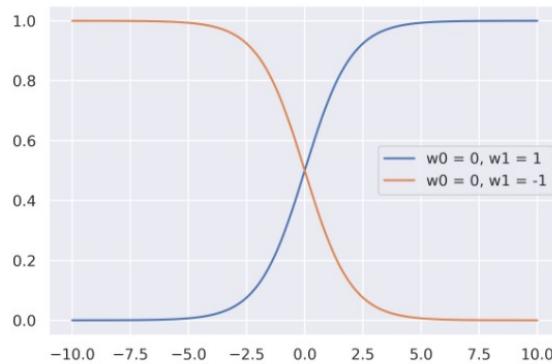
$$J(\bar{W}) = \frac{1}{n} \sum_{i=0}^{n-1} L(\hat{y}^{(i)}, y^{(i)})$$

- Costo para todas las instancias, caso 1D

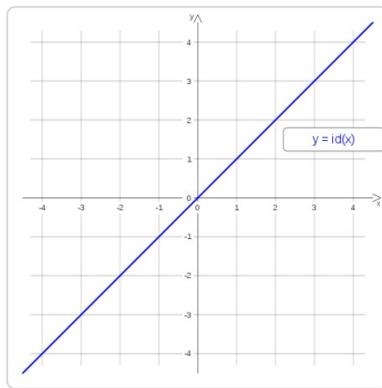
$$J(w_0, w_1) = \frac{1}{n} \sum_{i=0}^{n-1} L(\hat{y}^{(i)}, y^{(i)})$$

Funciones de Activación

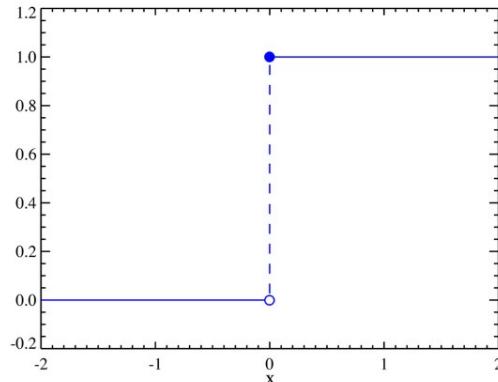
- Sigmoid / Logística:



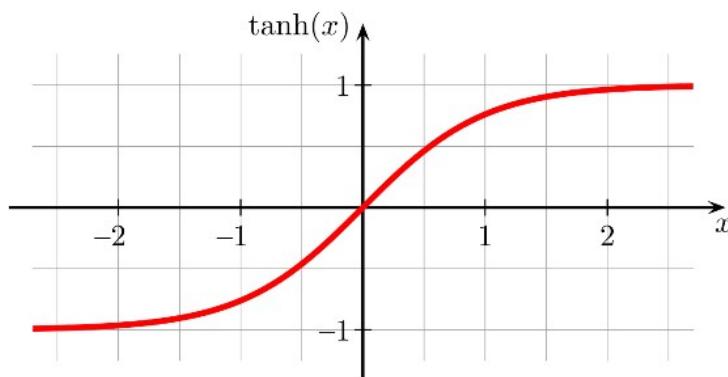
- Identidad: $f(x) = x$



- Escalón: $f(x) = 0 \text{ si } x; 1 \text{ si } x >= 0$

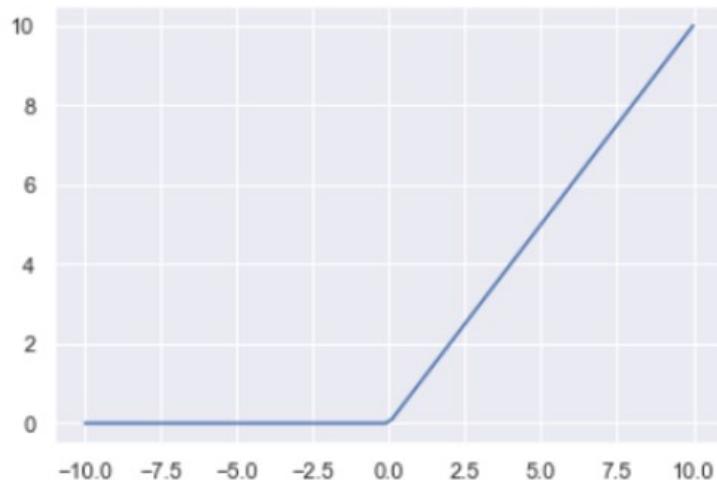


- Tangente Hiperbólica: $f(x) = \tanh(x) < 0$



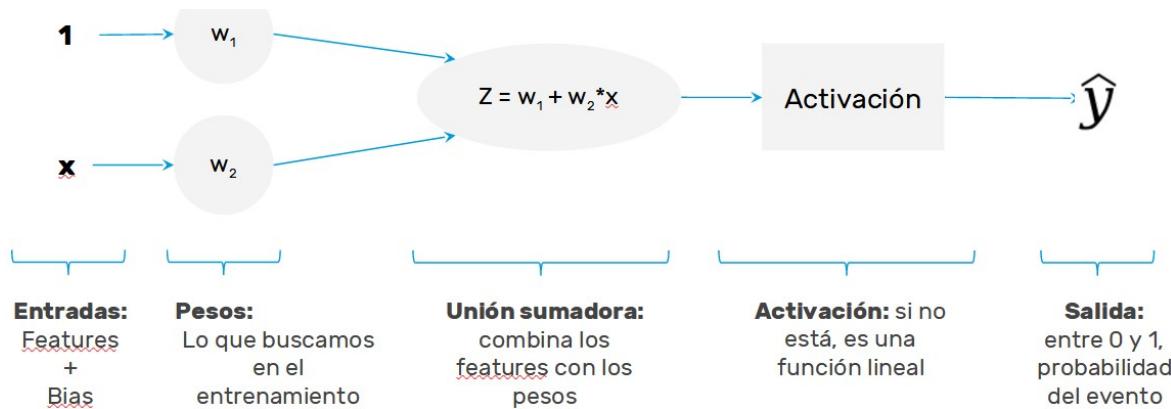
- ReLU (Rectified Linear Units): $f(x) = 0 \text{ si } x < 0; x \text{ si } x >= 0$

- En problemas de clasificación, lo más común es encontrar ReLU en las capas interiores y Sigmoid en la salida.



Perceptrón

- Esta unidad neural, cuyo modelo se diseñó en 1958, se denominó Perceptrón.
- No era sólo una función lineal, debido a que contaba con una función de activación.
- La función de activación, devuelve probabilidades, que están entre 0 y 1.



Perceptrón Multicapa

- La limitación con el perceptrón, es que sólo encuentra fronteras lineales, y como hemos visto, ampliar la red resuelve ese problema.
- Multiclasa: La cantidad de neuronas en la capa de salida tiene que ser igual a la cantidad de clases buscadas.

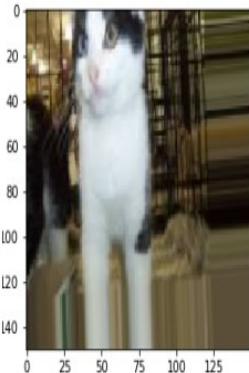
	Funciones de activación	Costos (Keras)
Multiclasa	1. Sigmoid/logística 2. Softmax	Categorical_crossentropy
Regresión	Identidad	1. mean_squared_error 2. mean_absolute_error 3. Otras

Regularización

El problema fundamental de es la tensión existente entre optimización y generalización, donde **la optimización se considera respecto al proceso de ajustar un modelo para conseguir el mejor rendimiento posible sobre los datos de entrenamiento** (es donde se concentra la parte de aprendizaje en el Aprendizaje Automático), y **la generalización es respecto a lo bien que el modelo entrenado se comporta sobre datos que no ha visto anteriormente**. El objetivo es conseguir una buena generalización, pero es precisamente la parte que no podemos controlar, ya que solo podemos ajustar el modelo en función los datos de entrenamiento. Lo ideal sería poder conseguir una mayor, y lo más variada posible, cantidad de datos de entrenamiento, así, como cualquier proceso de aprendizaje, automático o natural: un modelo entrenado con más datos tendrá más herramientas para extraer su aprendizaje a situaciones nuevas. Como no siempre se pueden conseguir más datos, lo que hacemos es limitar la capacidad de aprendizaje del modelo, para prevenir el sobreajuste. El conjunto de técnicas utilizadas para tal fin, se conoce como Regularización.

Data Augmentation

Cuando trabajamos con imágenes, hay algunas formas de incrementar el tamaño del conjunto de entrenamiento por medio de operaciones básicas: traslación, rotación, escalado, volteado, etc.



Parada temprana (early stopping)

Estrategia basada en validación cruzada, cuando observamos que el rendimiento en validación comienza a empeorar, paramos el entrenamiento del modelo.

L1 (Lasso)

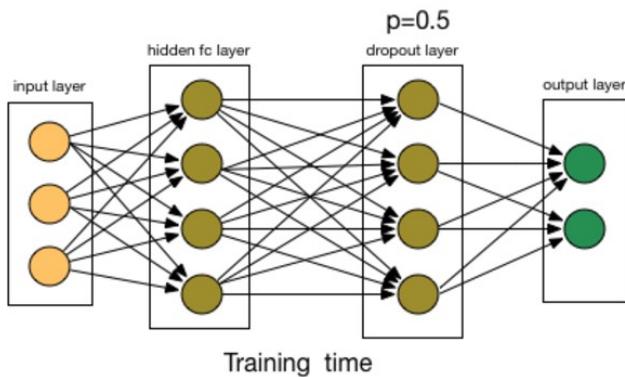
Coste es proporcional al valor absoluto de los pesos (matemáticamente, la norma L1 de los pesos). Suele dar como resultado que muchos pesos tomen el valor 0, por lo que a veces se identifica con un procedimiento de compresión de la red neuronal.

L2 (Ridge)

Coste es proporcional al cuadrado de los valores de los pesos (matemáticamente, la norma L2 de los pesos). En el contexto de las redes neuronales, a esta norma también se le llama weight decay, ya que fuerza a que los pesos tiendan a 0.

Dropout

Funciona como una capa que “apaga” neuronas de la capa anterior al azar. Al hacerlo, obliga a que ninguna se aprenda “de memoria” una muestra, sino que tengan que aprender entre todas.



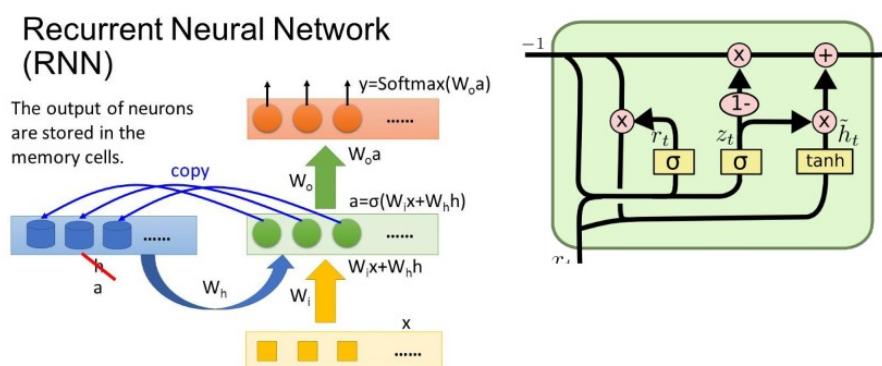
Redes Neuronales - Librerías

Con Python es posible programar desde cero y diseñar por completo una arquitectura de red neuronal. Sin embargo, también es posible implementar librerías que ya traen resuelto gran parte del código ocultando diferentes niveles de detalle dicha arquitectura.

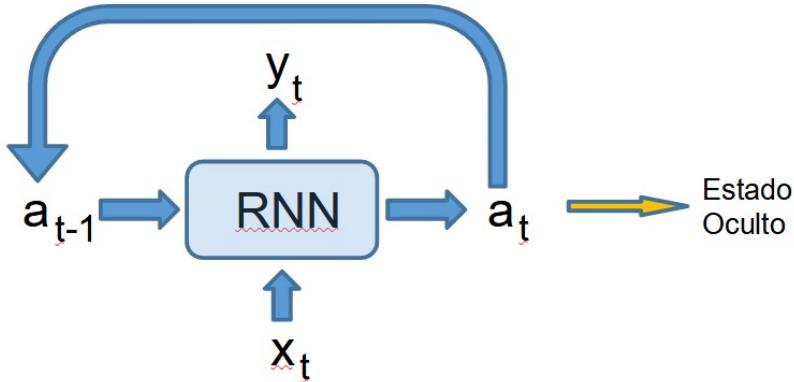


Redes Neuronales Recurrentes

Las Redes Neuronales Recurrentes ó RNN, además de tener conexiones con la capa anterior, las tienen con la misma capa. Con lo que combina la memoria de cada neurona con el input de la capa anterior al retroalimentarse.

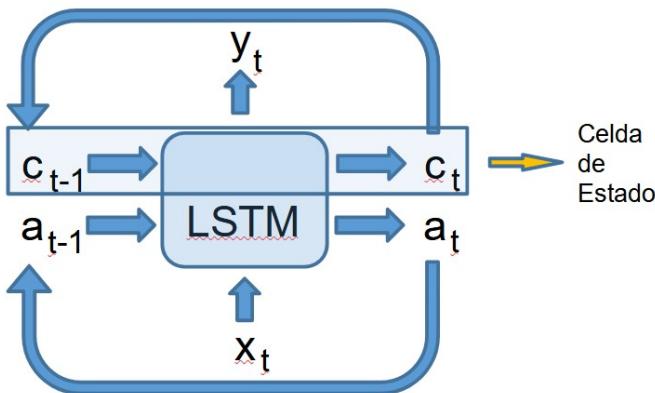


- Procesa Secuencias de datos.
- Cada elemento está correlacionado con el siguiente.
- El tamaño de los datos de entrada es variable.



Hay numerosas variantes de asombrosa complejidad: **LSTM** se utilizan para reconocimiento de voz, texto manuscrito y patrones.

- Agrega celda de estado.
- Compuertas "forget" y "update"
- Compuerta de salida



Enlaces sugeridos:

- <https://experiments.withgoogle.com/ai/ai-duet/view/>
- [Redes Transformers] (https://www.youtube.com/watch?v=Wp8NocXW_C4&feature=youtu.be)

Redes Neuronales Convolucionales

- La Red Neuronal Convolucional ó CNN es el tipo de red neuronal más utilizada para el reconocimiento y procesamiento de imágenes.
- Inspiradas en los procesos biológicos, en los que el patrón de conectividad entre neuronas se asemeja a la organización de la corteza visual de los seres vivos.
- Son una variante del Perceptrón Multicapa y usan relativamente poco preprocesamiento en comparación con otros algoritmos de clasificación de imágenes.
- Procesa la información mediante filtros y tiene la capacidad de calcular automáticamente los pesos de los filtros.
- La forma en que se guarda una imagen, consiste básicamente en una matriz de valores que van de 0 a 255, e indican el valor del color para cada posición en el plano 2D, además están los canales, correspondientes a los valores RGB, así que esta matriz tiene 3 dimensiones.
- Al analizar esa información, lo que queda como entrada para una red neuronal, es el array respectivo.

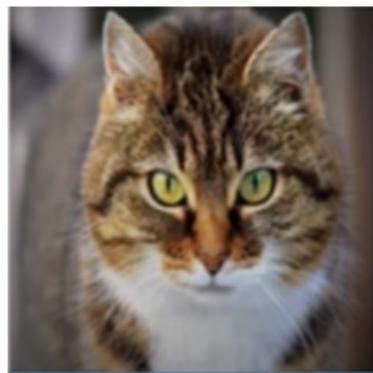
Definición regular:

- 64x64x3
- 12288

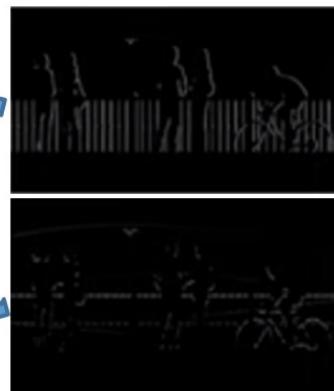


Buena definición:

- 1000x1000x3
- 3000000
- Si la segunda capa solo tiene 1000 neuronas, la matriz w nos queda de 3 millones por mil.



- En lugar de las capas completamente conectadas, la CNN aplica lo que se denomina capa convolucional, su funcionamiento es mucho más performante y de naturaleza muy diferente.
- Su principal característica está en reconocer los bordes de la imagen, tanto verticales como horizontales.



¿Cómo funciona? Tenemos el ejemplo de una matriz de 6x6 a la cual, se le aplica una operación de convolución, que consistirá en un filtro, otra matriz más chica, en este caso de 3x3. El resultado será una matriz de 4x4:

$$\begin{matrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{matrix} * \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} = \begin{matrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{matrix}$$

$$3*1+1*1+2*1 + 0*0+5*0+7*0 + 1*(-1)+8*(-1)+2*(-1) = -5$$

$$\begin{matrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{matrix} * \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} = \begin{matrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{matrix}$$

$$0*1+5*1+7*1 + 1*0+8*0+2*0 + 2*(-1)+9*(-1)+5*(-1) = -4$$

$$\begin{matrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{matrix} * \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} = \begin{matrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{matrix}$$

$$1*1+8*1+2*1 + 2*0+9*0+5*0 + 7*(-1)+3*(-1)+1*(-1) = 0$$

$$\begin{matrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{matrix} * \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} = \begin{matrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{matrix}$$

$$1*1+6*1+2*1 + 7*0+2*0+3*0 + 8*(-1)+8*(-1)+9*(-1) = -16$$

- Si en la matriz original, hay sectores donde es muy marcada la diferencia numérica de los pixeles, significa que son zonas diferenciadas de la imagen.
- De esta forma queda claro el sentido de la transición de claro a oscuro y viceversa:

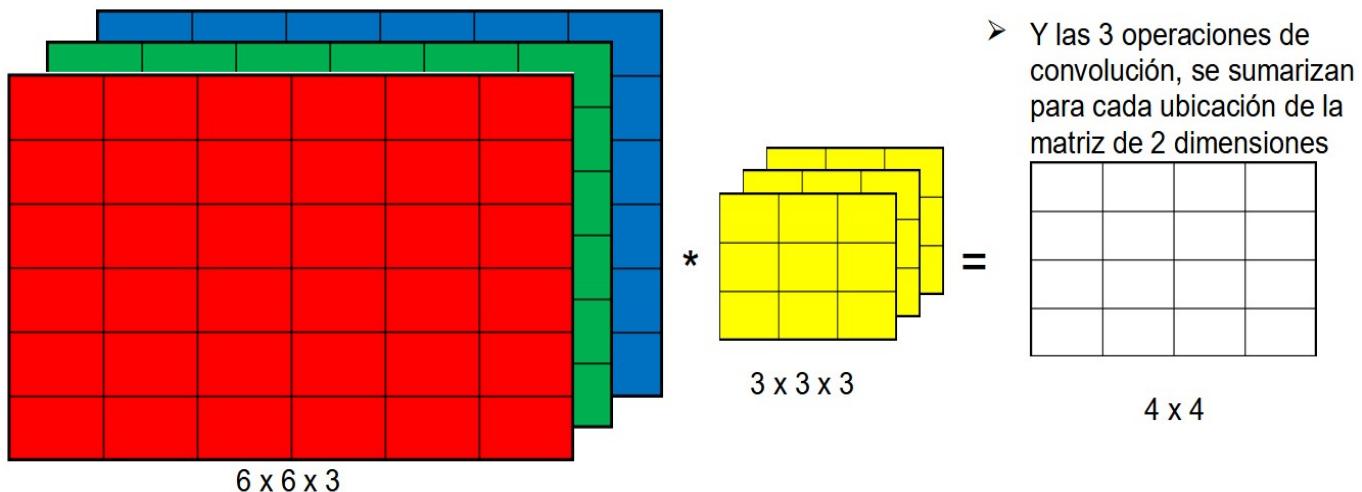
$$\begin{array}{ccccccc}
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0
 \end{array} * \begin{array}{ccc}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{array} = \begin{array}{cccc}
 0 & 30 & 30 & 0 \\
 0 & 30 & 30 & 0 \\
 0 & 30 & 30 & 0 \\
 0 & 30 & 30 & 0
 \end{array}$$

$$\begin{array}{ccccccc}
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10
 \end{array} * \begin{array}{ccc}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{array} = \begin{array}{cccc}
 0 & -30 & -30 & 0 \\
 0 & -30 & -30 & 0 \\
 0 & -30 & -30 & 0 \\
 0 & -30 & -30 & 0
 \end{array}$$

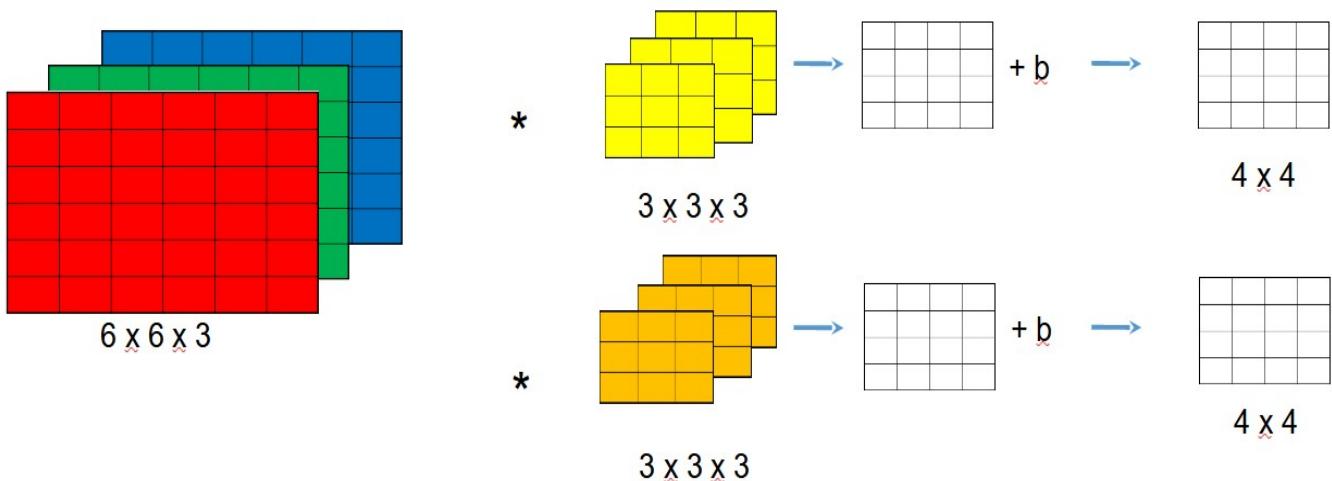
- Los filtros pueden aplicarse también de manera horizontal, el resultado siempre va a ser una matriz donde queden identificados los límites en la imagen, es decir los bordes:

$$\begin{array}{ccccccc}
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10
 \end{array} * \begin{array}{ccc}
 1 & 1 & 1 \\
 0 & 0 & 0 \\
 -1 & -1 & -1
 \end{array} = \begin{array}{cccc}
 0 & 0 & 0 & 0 \\
 30 & 10 & -10 & -30 \\
 30 & 10 & -10 & -30 \\
 0 & 0 & 0 & 0
 \end{array}$$

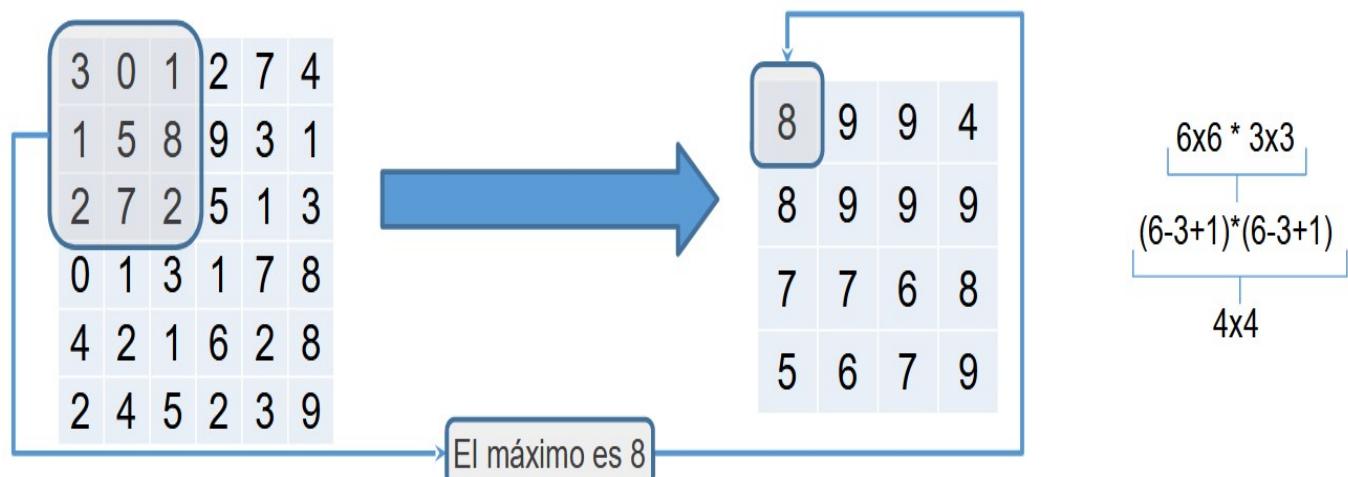
- Surgen dos inconvenientes, la imagen se hace más pequeña y además, hay píxeles que son repasados menos veces por el filtro, aquellos ubicados en los bordes.
- Para resolverlo, se agranda la matriz agregando 1 o 2 píxeles más, quedando entonces para el ejemplo, una matriz de 8x8.
- Por otra parte, la matriz de filtros, usualmente es impar, es decir, 3x3, 5x5 o 7x7.
- Una variante es cambiar el **salto ó strided convolution** a 2 en lugar de 1.
- Si la imagen es RGB, entonces en realidad tendremos 3 canales, así que el proceso de convolución es además sobre una tercera dimensión:



- Para crear una capa de la red neuronal convolucional, se aplica un **sesgo** y una función de activación **ReLU**:



- La capa de **agrupación máxima ó max pooling** es un nuevo tipo de filtro, que se queda con el valor máximo.
- A veces, esta capa cuenta junto con la capa de convolución como una única capa, debido a que solo tiene hiperparámetros, y no tiene parámetros que ajustar, o sea que no se practica propagación hacia atrás.



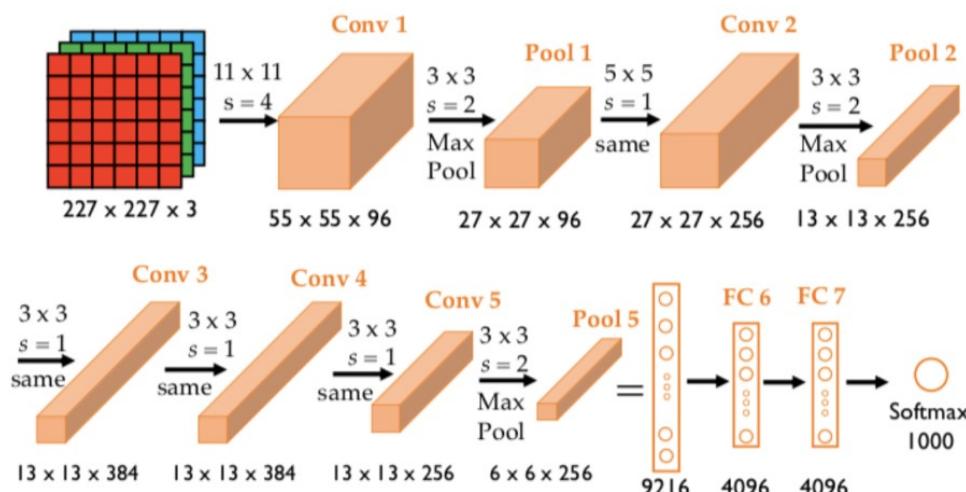
En una CNN entonces, se encuentran los 3 tipos de capas:

1. Convolucional (CONV)
2. Pool para agrupar (Pool)
3. Totalmente conectada (FC: Fully connected)

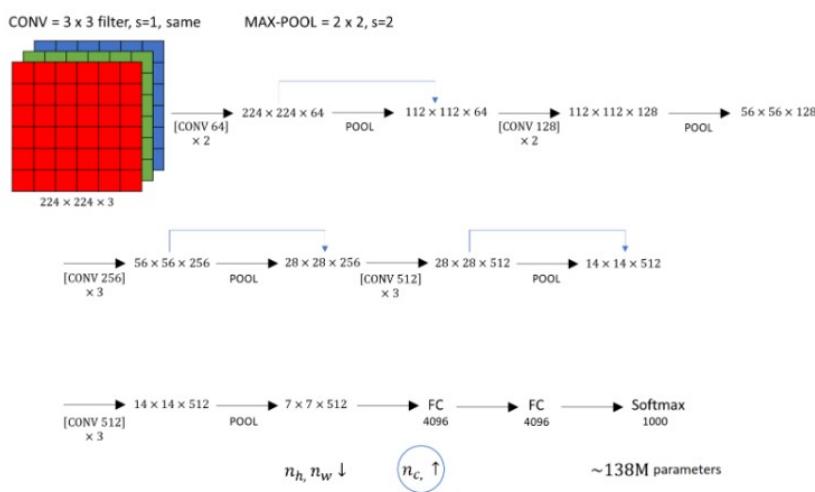
A medida que la red neuronal se hace más profunda, disminuye el tamaño de la matriz original, aumentando el número de canales:

	Forma	Tamaño	Parámetros
Input	(32,32,3)	3072	0
CONV1 (f=5, s=1)	(28,28,8)	6272	208
POOL1	(14,14,8)	1568	0
CONV2 (f=5, s1)	(10,10,6)	1600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48001
FC4	(84,1)	84	10081
Softmax	(10,1)	10	841

AlexNet es una red con 60 millones de parámetros.



VGG-16 es un ejemplo de red con 138 millones de parámetros.



Neural Style Transfer

Contenido C



Estilo S

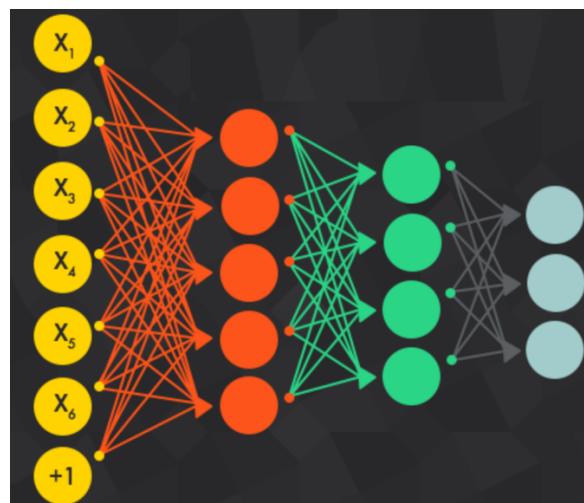


Imagen generada G

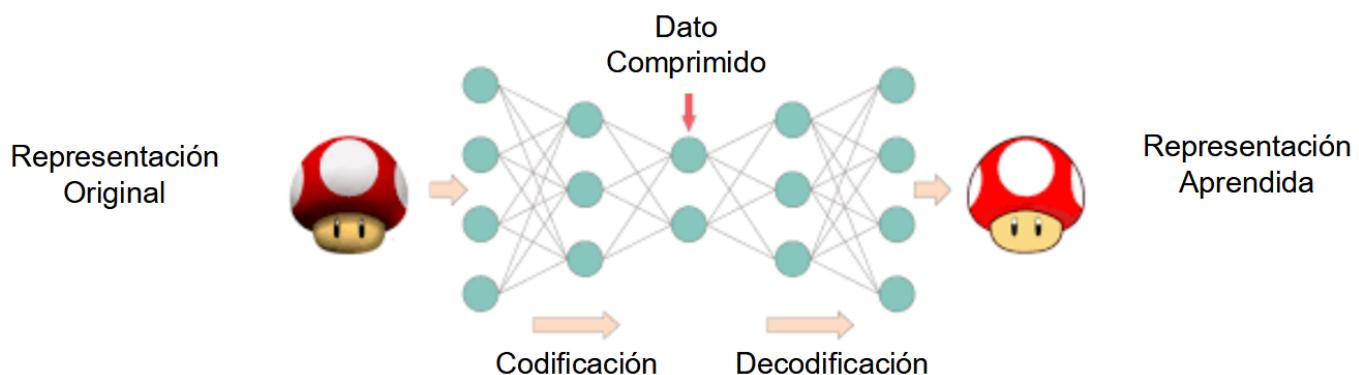


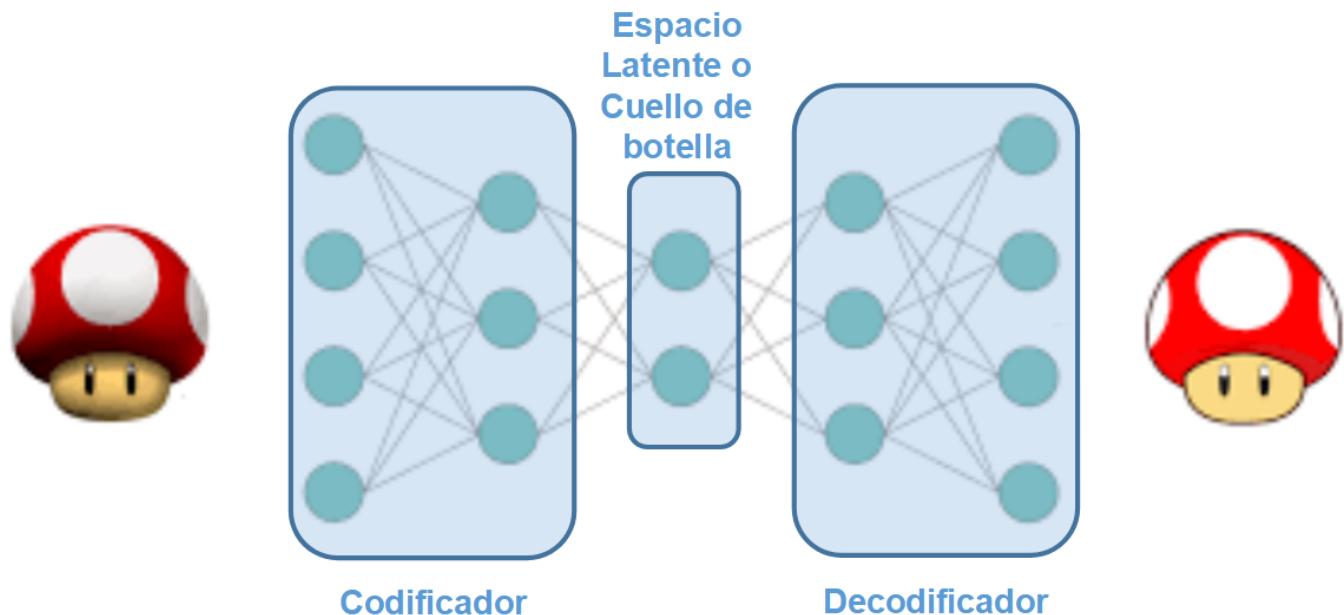
Autoencoders

- Los Autoencoders permiten obtener una representación compacta de los datos de entrada.
- Por medio de la extracción de características, se reduce la dimensionalidad.
- En la imagen se observan 6 características iniciales en la entrada y en la salida sólo 3.



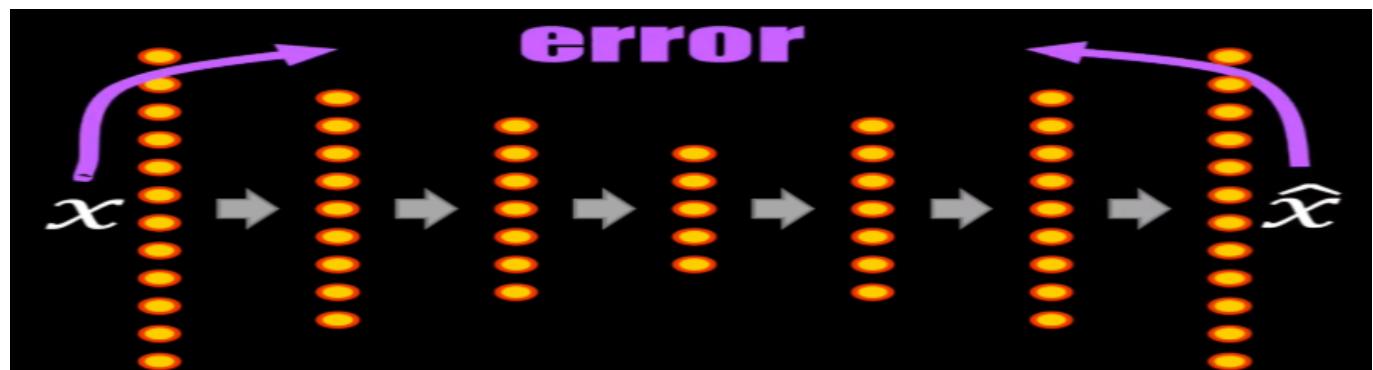
La idea del Autoencoder es entrenar a la red neuronal para que aprenda a generar la salida para el mismo dato de entrada



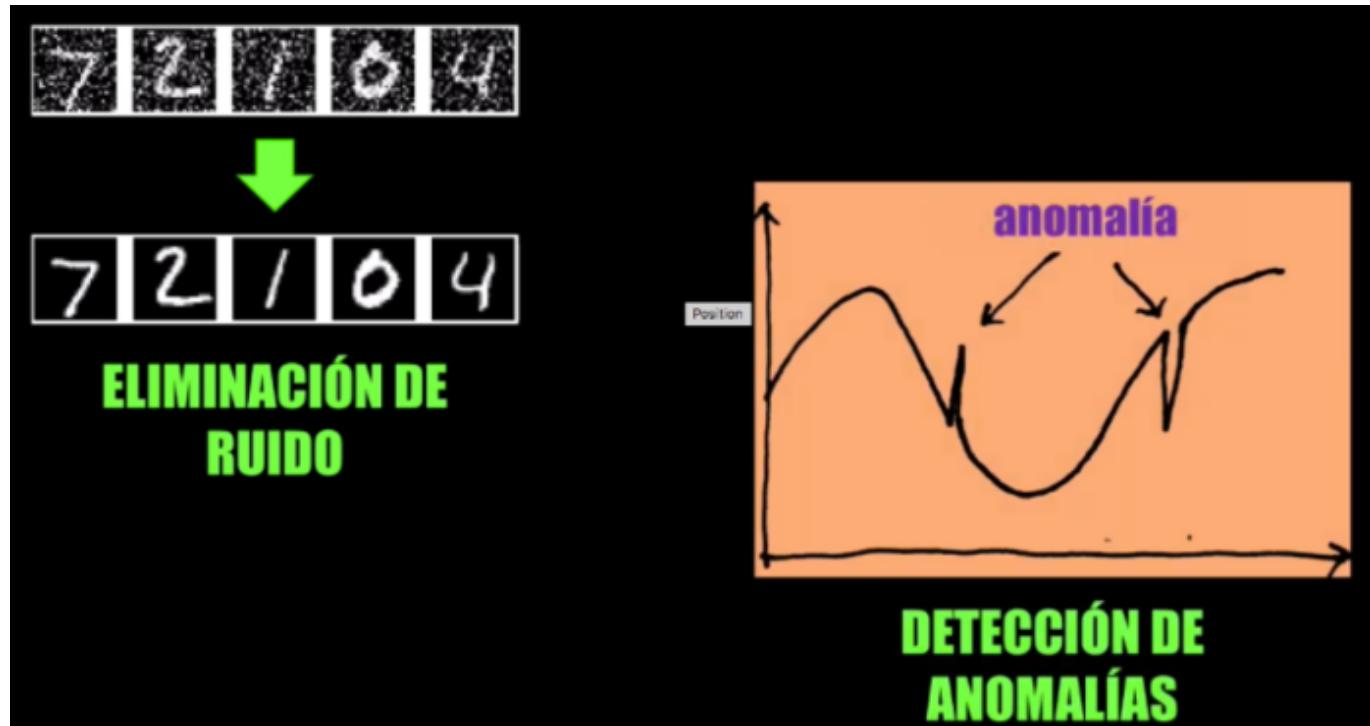


- El **codificador** permite comprimir el dato de entrada.
- El **espacio latente o cuello de botella**, es la representación compacta obtenida.
- El **decodificador** reconstruye la entrada a partir del espacio latente.

El Autoencoder es entrenado de forma similar a una red neuronal, sin embargo, la función de error usada para actualizar los coeficientes es simplemente el resultado de comparar, punto a punto, el dato reconstruido con el dato original.

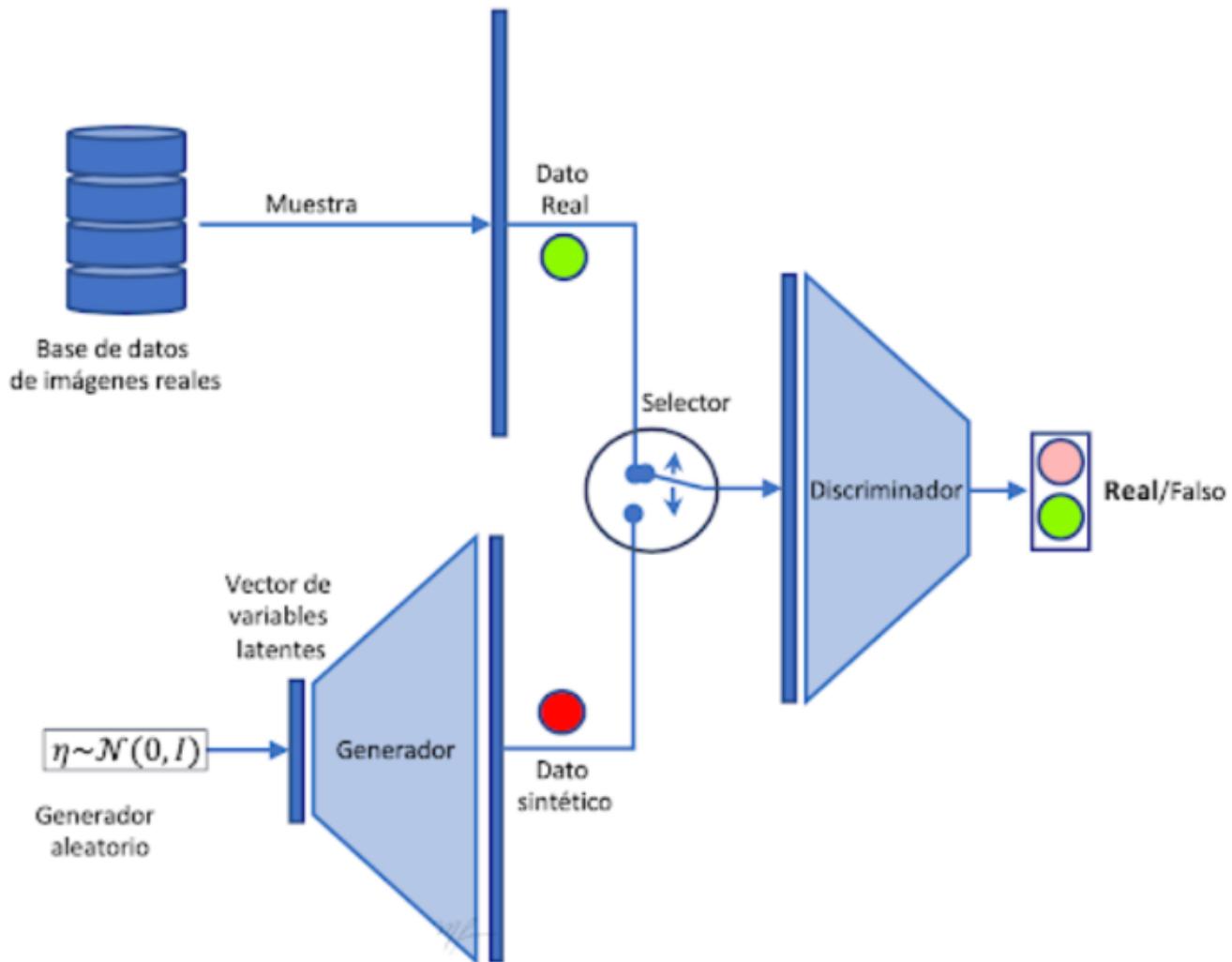


- Es un ejemplo de aprendizaje no supervisado, ya que durante el entrenamiento no definimos la categoría a la que pertenece cada entrada, lo que queremos es que la salida sea precisamente el mismo dato de entrada.
- Se utiliza para eliminar ruido en imágenes o para detectar anomalías en una serie de datos:

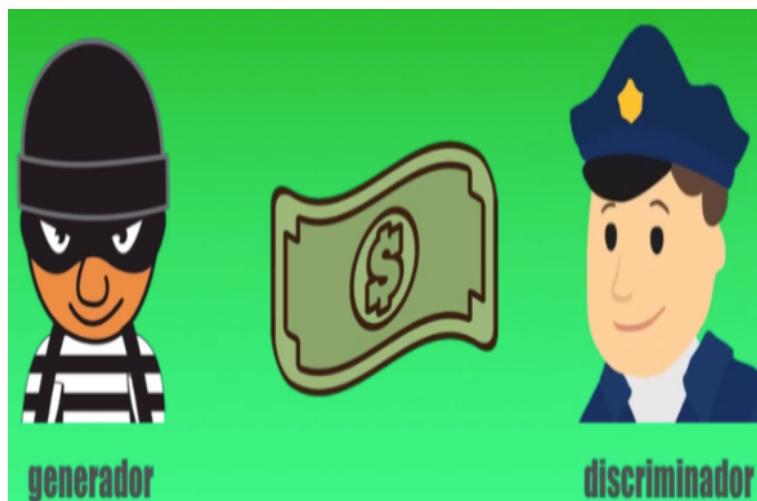


Redes Adversarias Generativas (GAN)

- Con estas redes se logra generar imágenes totalmente artificiales que se asemejan a la distribución de los datos reales usados durante el entrenamiento.
- Se tienen dos modelos compitiendo, que pueden ser Redes Neuronales o Convolucionales: Un Generador y un Discriminador.



- La competencia entre estos dos modelos se puede ver a través de una analogía.
- El Generador es como un falsificador, que intenta producir billetes falsos sin que estos sean detectados.
- Mientras que el Discriminador es como el policía, que intenta detectar estos billetes falsos.
- La competición hace que ambos mejoren sus métodos.



- Se entrena un primer modelo, el Discriminador (policía), para que sea capaz, por ejemplo, de reconocer rostros humanos. Este discriminador será simplemente un clasificador, como por ejemplo una Red Convolutacional.

- Si ingresamos una imagen con un rostro humano, la salida generada por el discriminador será igual a 1, mientras que si ingresamos otra imagen diferente, la salida será 0.



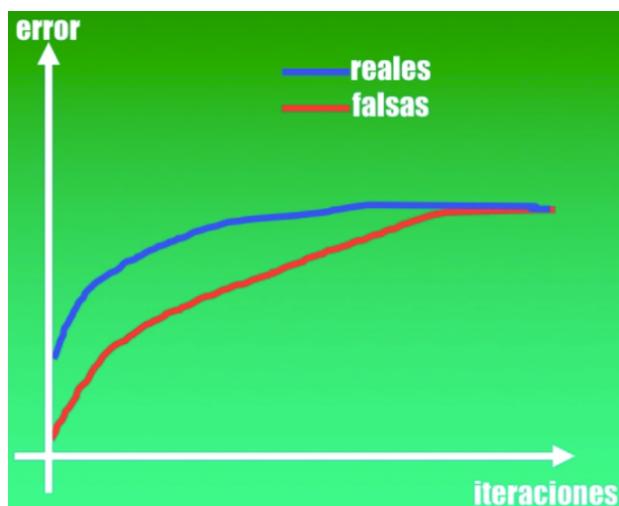
- Se crea un segundo modelo, el Generador (ladrón) cuyo objetivo es que sea capaz de tomar una entrada aleatoria y a la salida generar algo muy parecido a una imagen de un rostro humano.



- Los dos modelos se entrena n simultáneamente buscando que al final sea el Generador el vencedor de la competencia.
- Para saber si el entrenamiento es adecuado, se analiza el error del Discriminador, tanto con imágenes reales como con las imágenes falsas, obtenidas con el Generador.
- Al inicio del entrenamiento es de esperar que las imágenes obtenidas con el Generador no sean similares a un rostro. Así, para el Discriminador resultará muy fácil diferenciar entre una imagen real y una imagen falsa, y por tanto el error en uno y otro caso será muy pequeño.
- A medida que avanza el proceso de entrenamiento, el Generador aprenderá poco a poco a producir imágenes cada vez más parecidas a un rostro humano.
- Estos dos modelos combinados reciben el nombre de Red Adversaria Generativa.



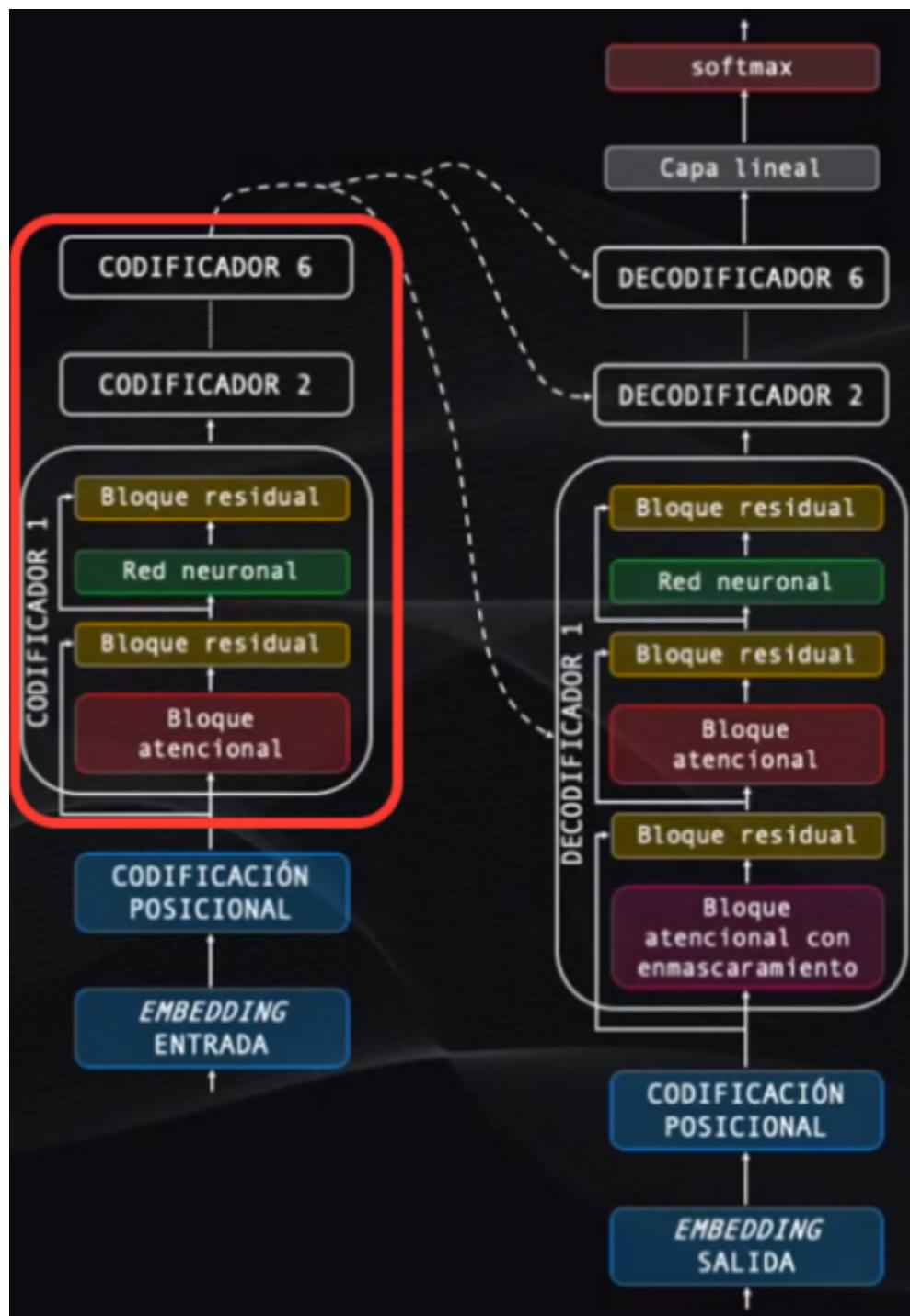
- A medida que avanza el entrenamiento el Generador logrará “confundir” al Discriminador, y por tanto el error de dicho discriminador será cada vez más alto, lo cual quiere decir que no estará en capacidad de diferenciar claramente una imagen real de una falsa.



- Al final del entrenamiento, idealmente la salida del Discriminador será precisamente 0.5, lo cual quiere decir que habrá sido engañado por completo por el Generador.
- Se concluye entonces que el Generador ha aprendido la distribución de los datos de entrada y por tanto ha aprendido a replicar con precisión esta distribución.



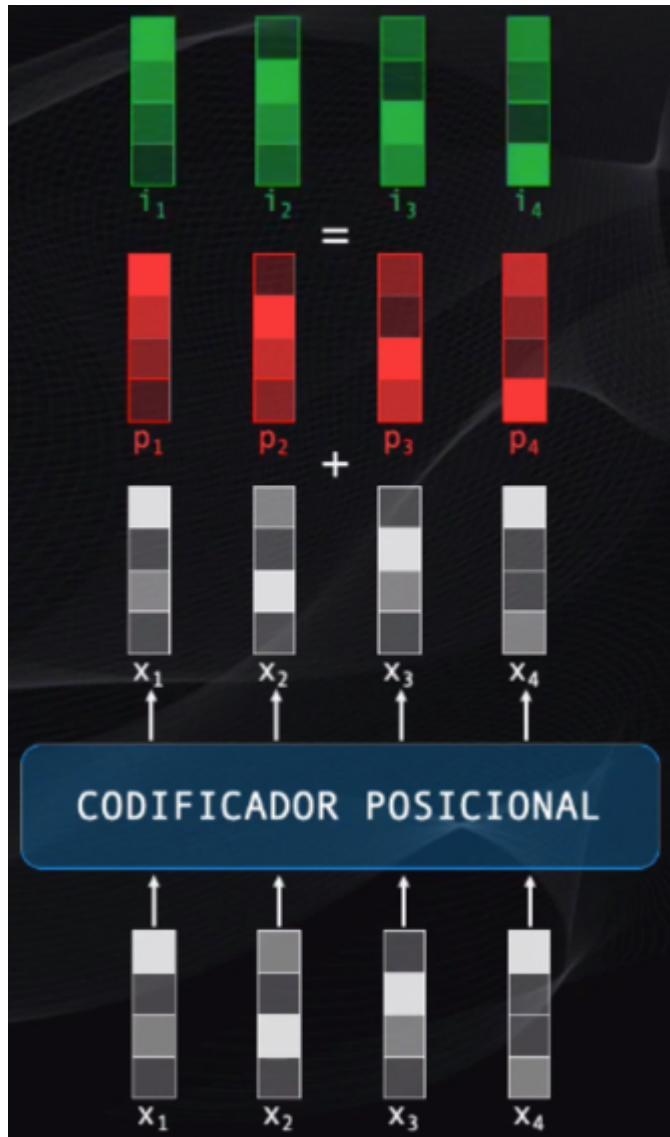
- Desarrolladas como una alternativa al problema de la traducción de texto de un idioma a otro.
- Analizan secuencias de palabras muy extensas usando un mecanismo que se llama **atención**.
- **Procesan toda la secuencia en paralelo**, y no en serie como ocurre con las Redes Recurrentes.



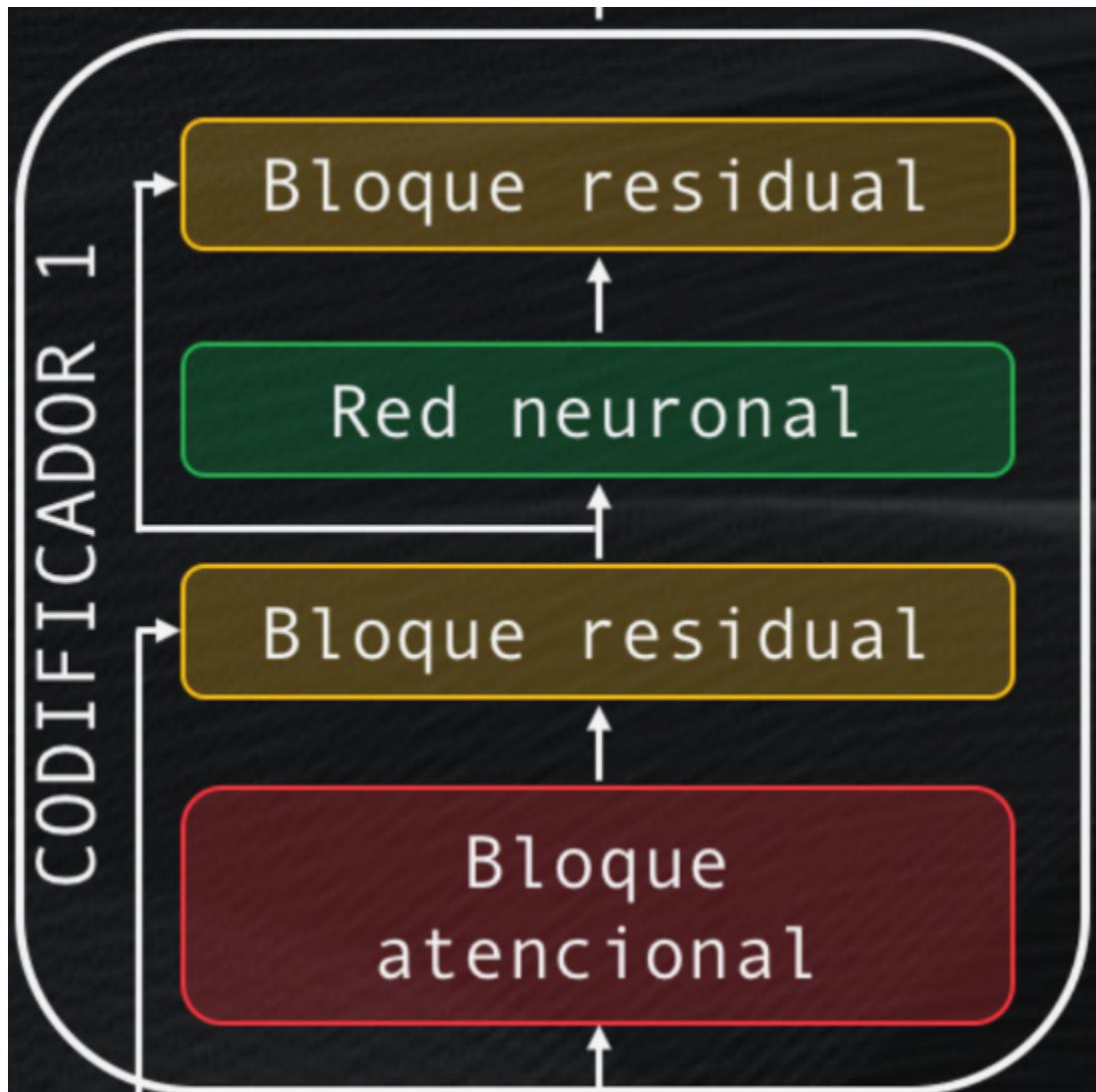
- La secuencia es inicialmente convertida en una representación numérica usando un **Embedding**.



- Con el **codificador de posición** se le indica a la red el orden en que se encuentran las palabras dentro del texto, para lograr el procesamiento en paralelo de la secuencia.
- El codificador indica la posición relativa de cada token dentro de la secuencia.



- Se continua con el bloque de codificación, que contiene seis codificadores, todos con una estructura idéntica.
- Cada codificador tiene cuatro elementos:
 - un bloque atencional
 - un bloque de conexión residual
 - una red neuronal
 - otro bloque de conexión residual

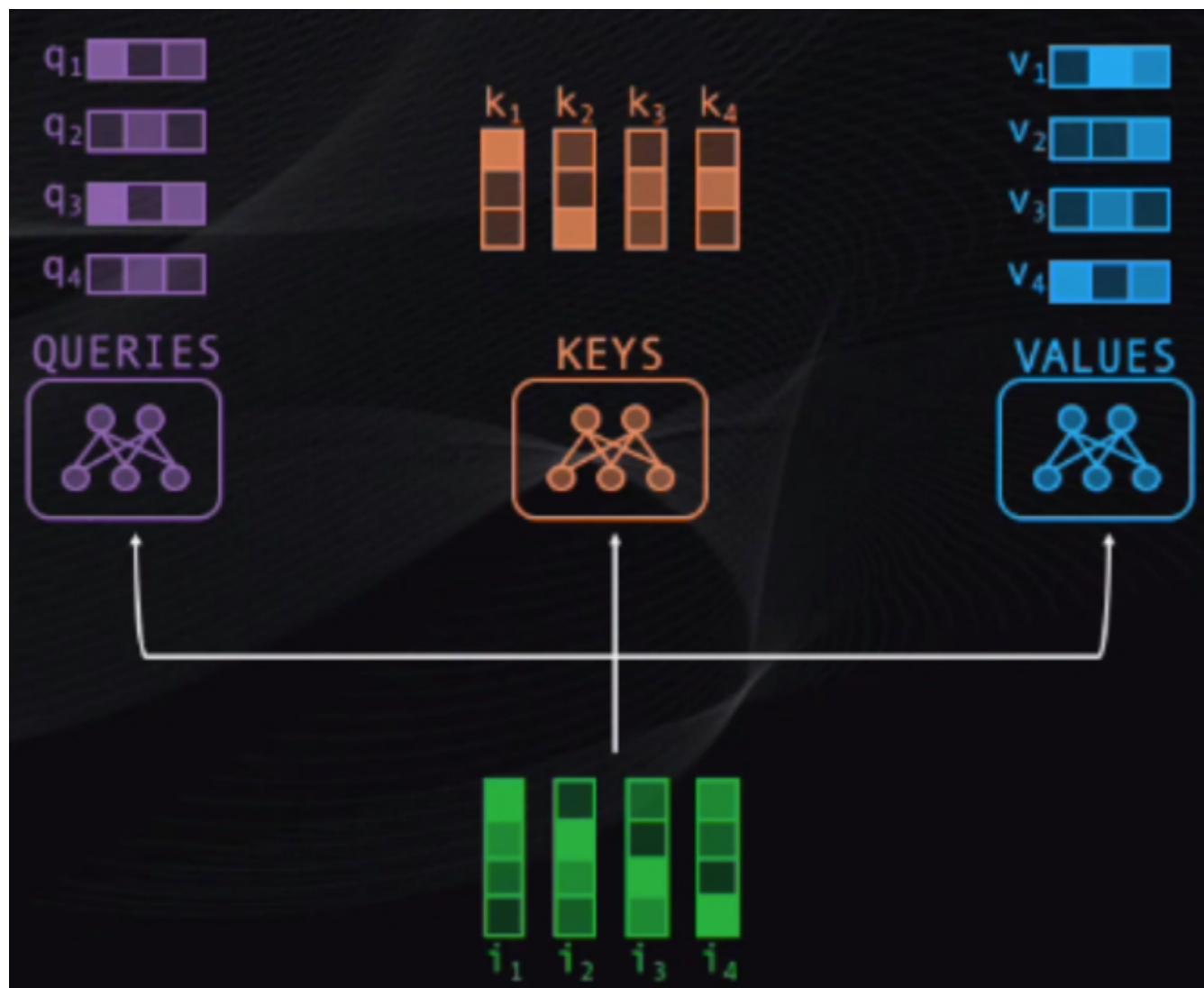


- El **bloque atencional**, que es tal vez el más importante de toda la red, se encarga de analizar la totalidad de la secuencia de entrada de manera simultánea y de **encontrar relaciones entre varias palabras de esta secuencia**.
- Si el texto de entrada es “I love Italian food”, podemos ver que hay al menos dos posibles asociaciones entre palabras: el verbo “love” y el sujeto (“I”) y el sustantivo “food” asociado al adjetivo “Italian”. Pero además entre estas dos frases (I Love e Italian Food) también hay una asociación.

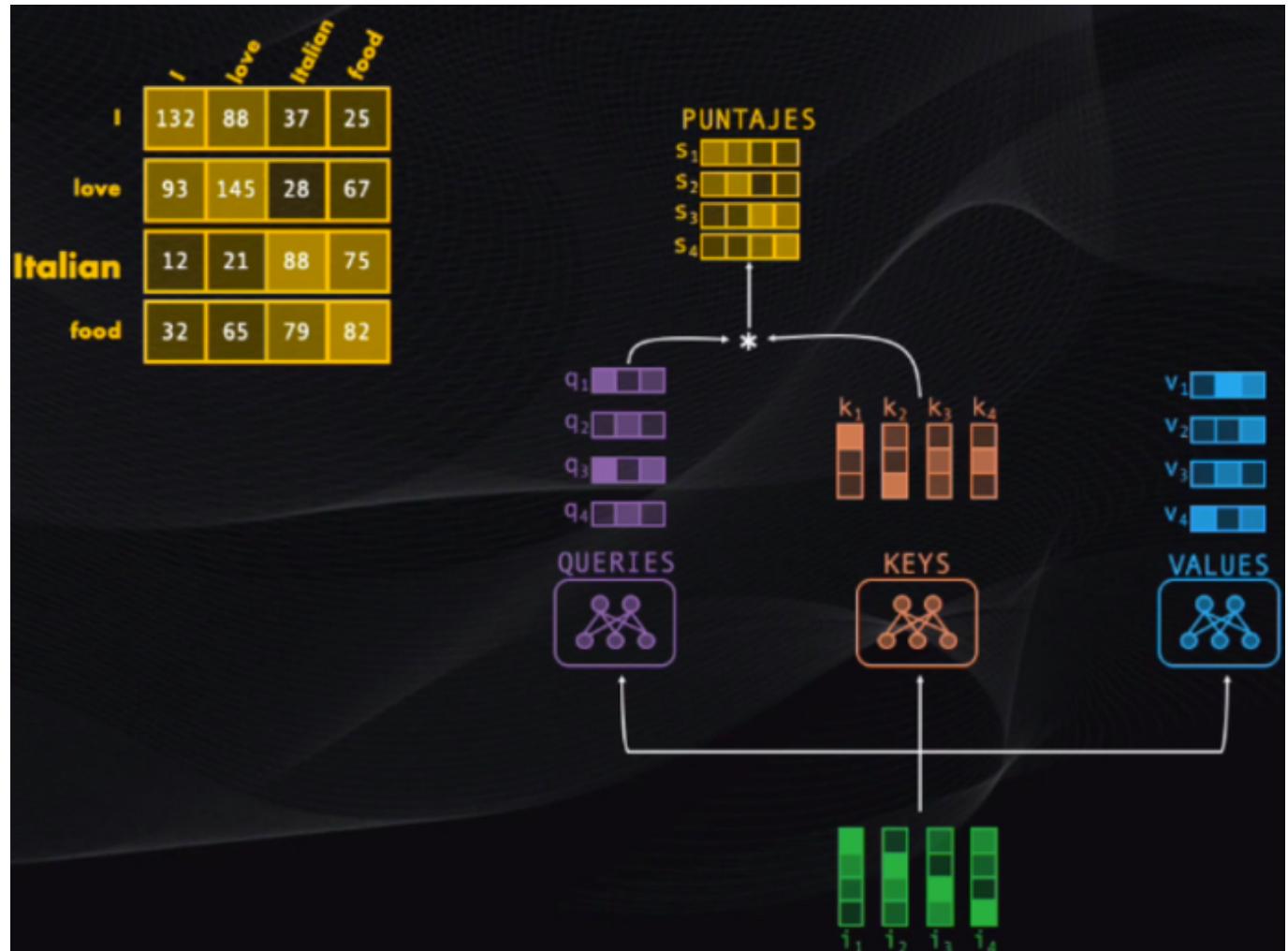


- El bloque atencional expresa numéricamente las relaciones que existen a diferentes niveles dentro de la secuencia, y luego codifica cada una de ellas con esta información del contexto, indicando así cuáles son los elementos del texto a los que se deben prestar más atención al momento de hacer la traducción.
- Esta es precisamente la manera como **las Redes Transformer “comprenden” este contexto** para codificar adecuadamente cada palabra.

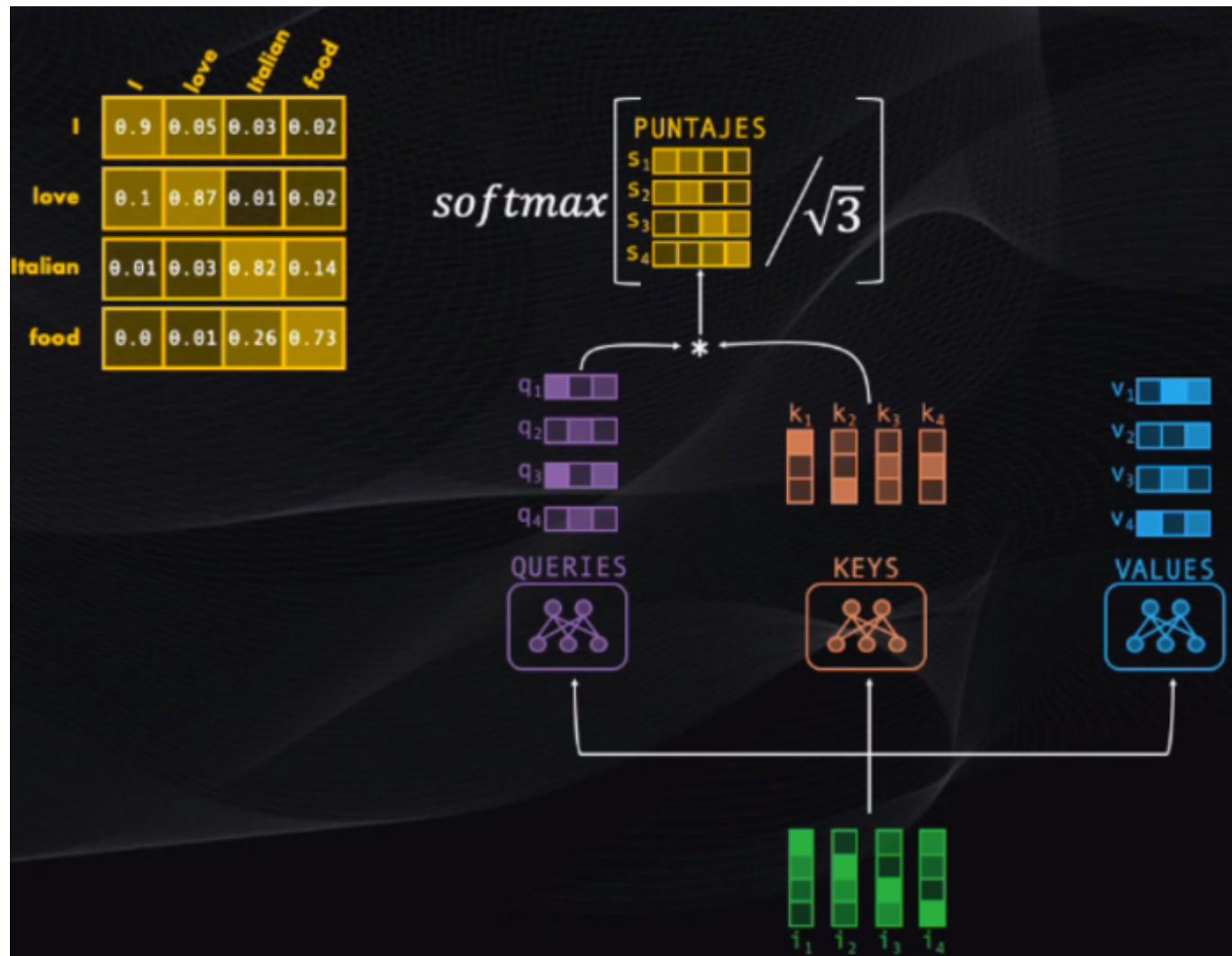
- Para lograr esto en primer lugar los tokens se llevan simultáneamente a tres pequeñas redes neuronales, entrenadas para calcular los vectores "query", "key" y "value". Estos vectores son simplemente tres representaciones alternativas de los tokens originales.



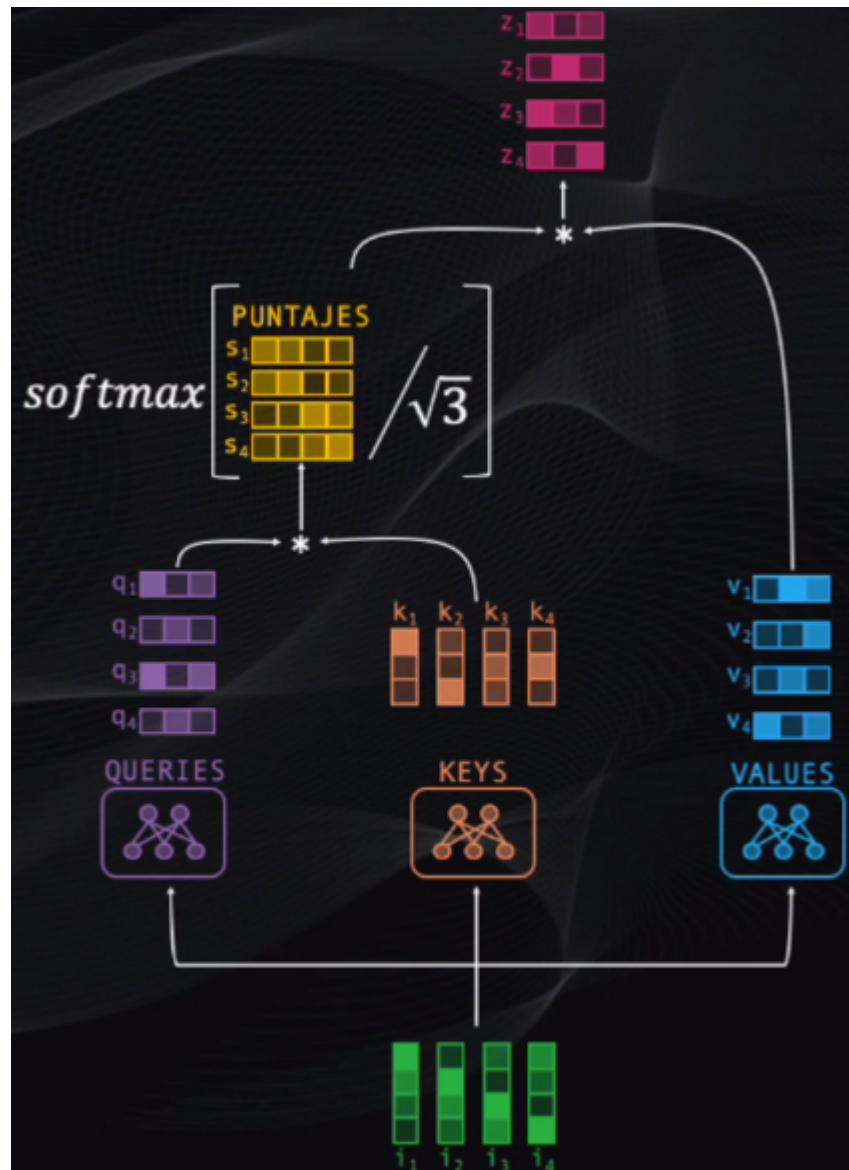
- Se toma cada token y se compara con cada uno de los keys existentes.
- Esta comparación es simplemente una multiplicación de vectores, y con esto se obtendrá un puntaje que mide el grado de asociación entre pares de palabras.



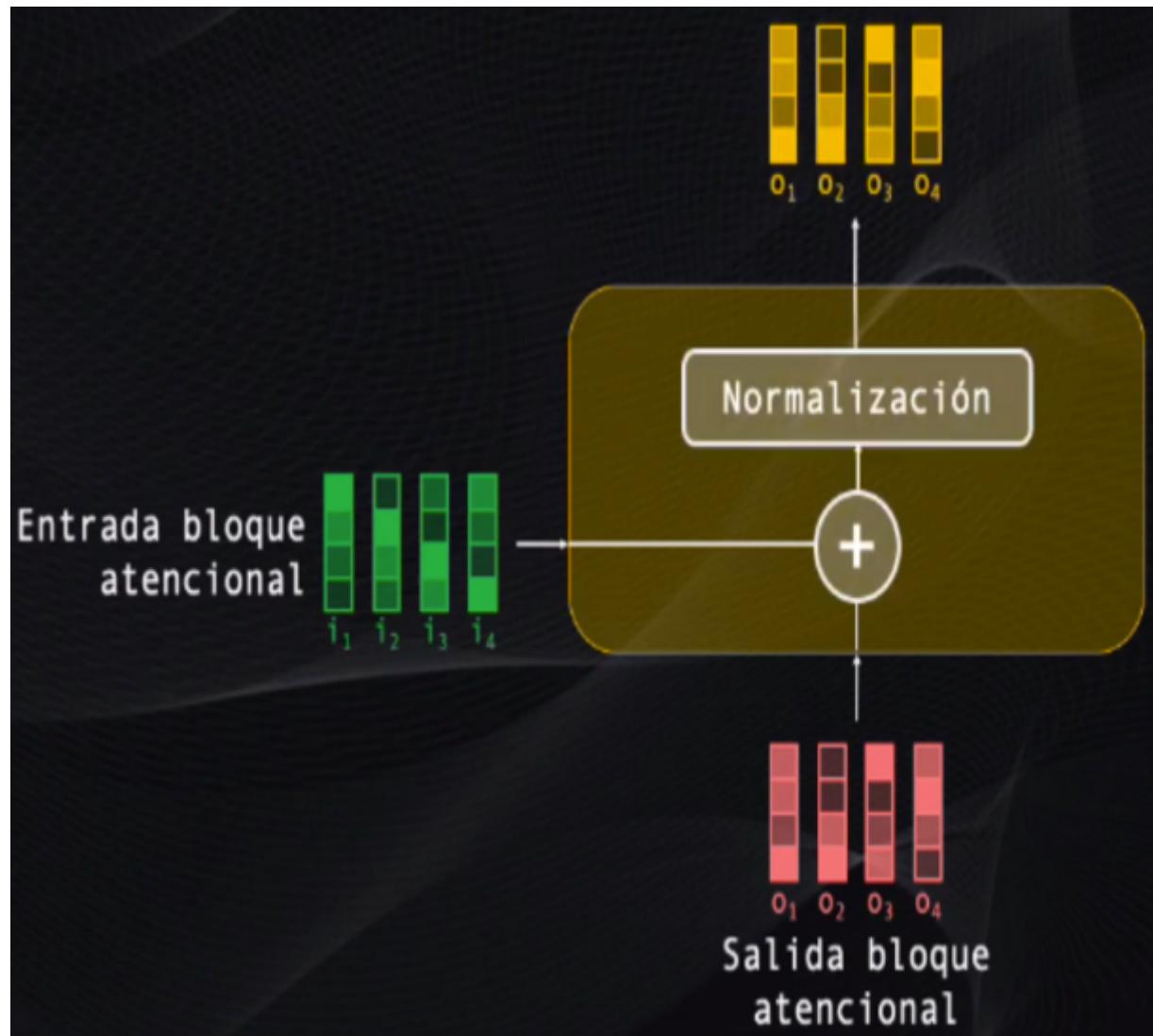
- Se representa cada puntaje como una probabilidad entre cero y uno.
- Se usan estos puntajes para ponderar cada uno de los vectores values, indicando así la **importancia de cada palabra** al momento de la codificación de los tokens.



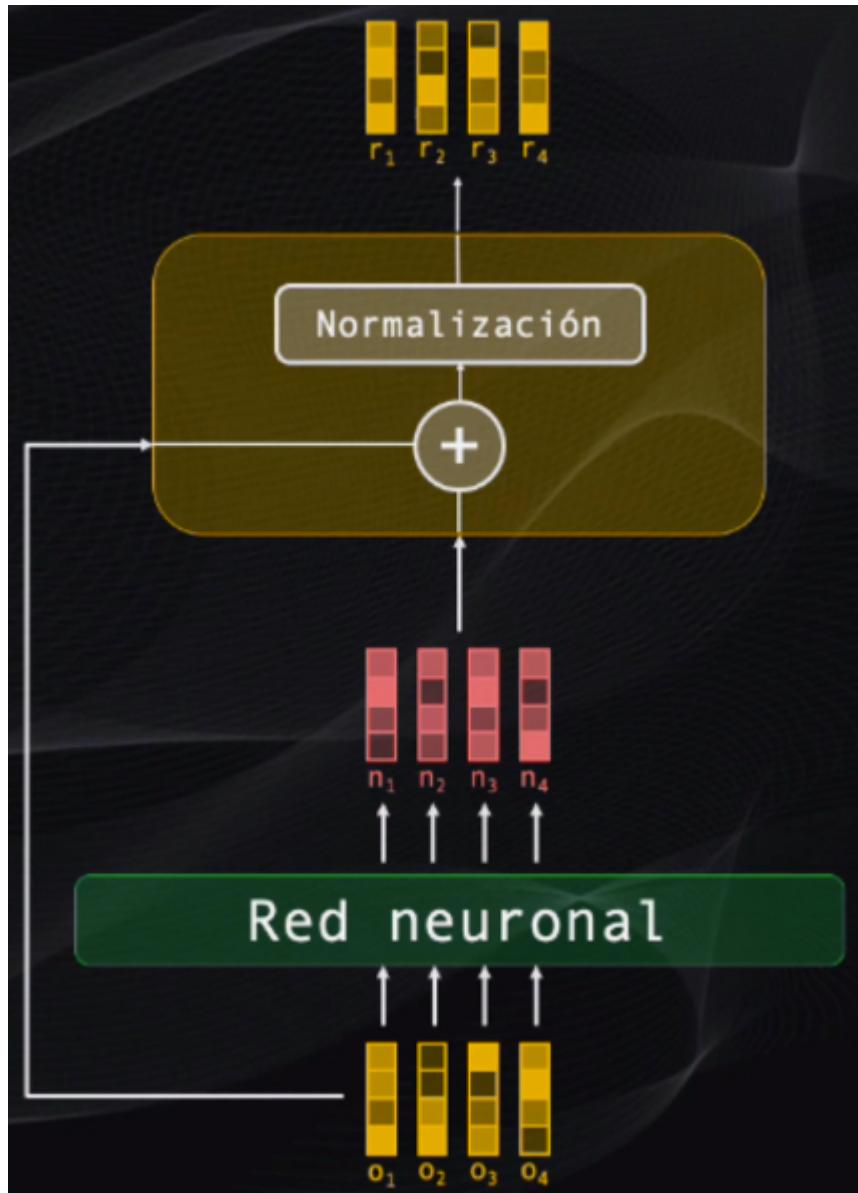
- Se condensa toda esta información resultante de la comparación en un solo vector por cada token.
- La matriz de puntajes se multiplica por la matriz de values: el resultado serán cuatro nuevos tokens, que contendrán la codificación de **la información de contexto más relevante para cada palabra de la secuencia**.



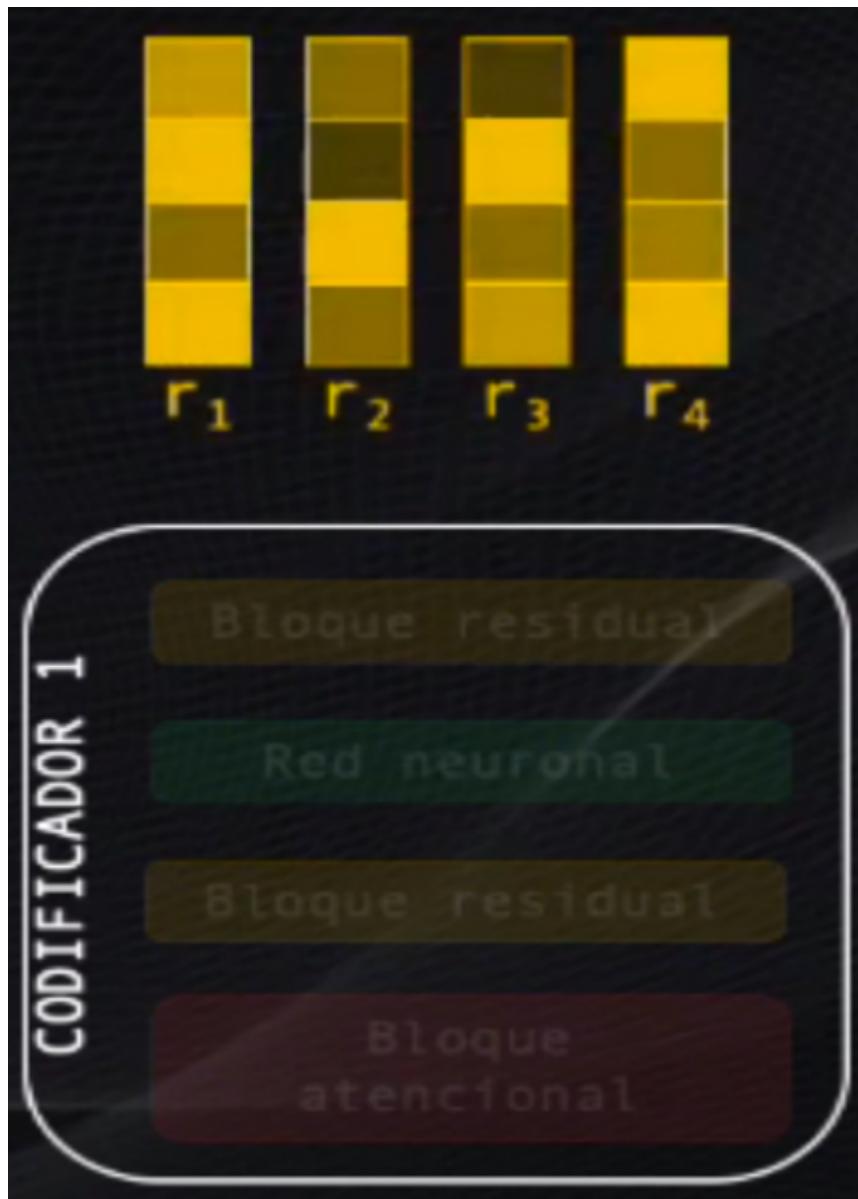
- El bloque atencional toma los tokens iniciales y codifica en los tokens resultantes los elementos de la secuencia a los que se debe dar más relevancia.
- Al usar múltiples bloques atencionales es posible detectar y codificar asociaciones entre palabras y grupos de palabras a diferentes niveles.
- Las salidas de estos bloques se combinan en una última red neuronal que condensa toda la información resultante en un único vector para cada token de entrada.
- Al bloque residual se llevan tanto la entrada como la salida del bloque atencional.
- Esta etapa toma los dos datos, los suma y luego los normaliza para que tengan la escala adecuada requerida por el siguiente bloque.



- Una red neuronal procesa en paralelo todos los vectores de la secuencia, tomando la información atencional de las capas anteriores y consolidándola en una única representación.
- La entrada y la salida de esta red neuronal son luego llevadas a un bloque residual que tiene exactamente las mismas características del bloque anterior.



- El bloque de codificación en resumen toma los tokens de entrada, los procesa en paralelo y entrega a la salida una representación que contiene información atencional sobre las diferentes relaciones entre palabras o grupos de palabras de la secuencia, importantes al momento de la traducción.



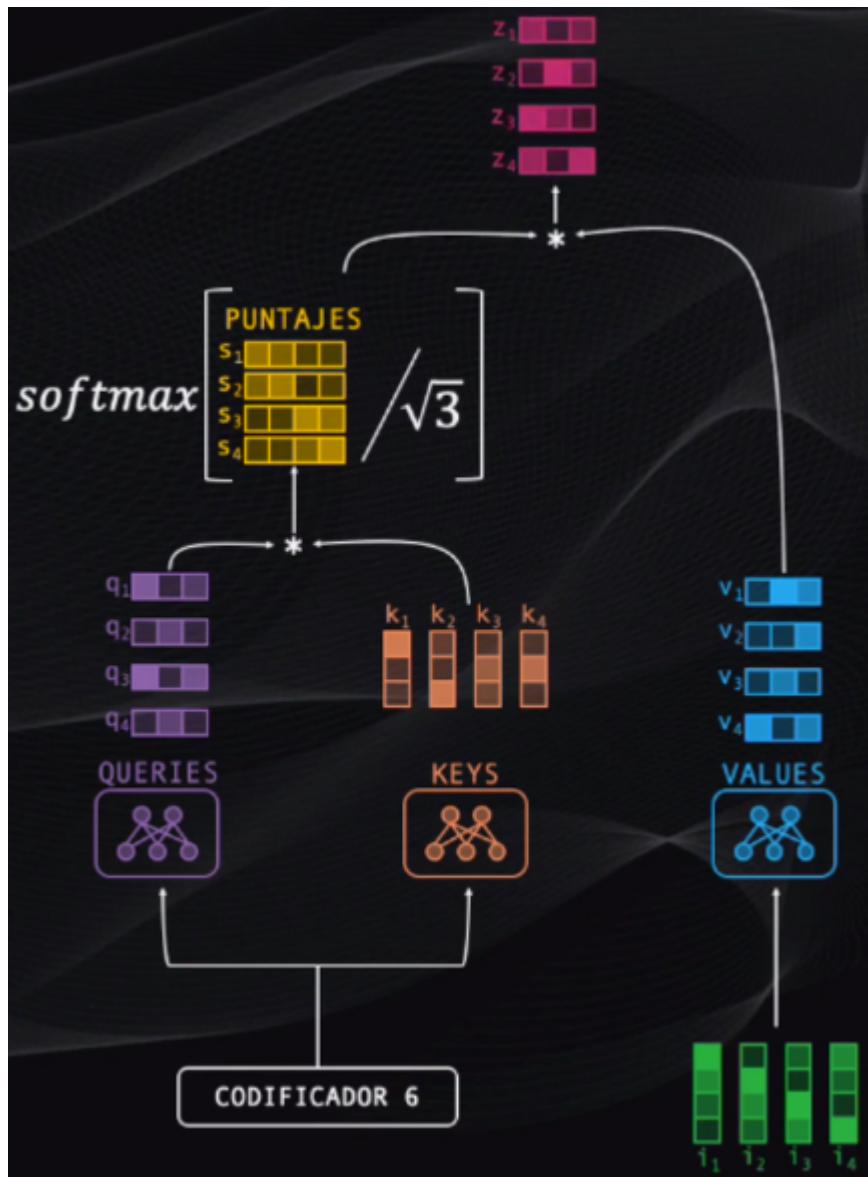
- La traducción comienza con la palabra clave “inicio”, la cual es codificada con el Embedding y posicionalmente.
- Al ingresar al primer decodificador es procesada por el bloque atencional de enmascaramiento.
- En este bloque, el decodificador debe prestar atención únicamente a la palabra generada actualmente y a las anteriores, no a las futuras.

	Amo	la	comida	Italiana
Amo	0.9	0.01	0.05	0.04
la	0.07	0.85	0.12	0.06
comida	0.02	0.03	0.88	0.07
Italiana	0.05	0.05	0.07	0.83

- Se agrega un bloque que enmascara las palabras a las que durante la decodificación no se debe prestar atención.



- El bloque atencional se enfoca tanto en la secuencia original como en la de salida y para ello toma la salida del codificador y las lleva a las redes “queries” y “keys”, mientras que el nodo “values” usa como entrada el dato proveniente del bloque residual anterior.
- El codificador le indica al decodificador a qué elementos debe prestar más atención al momento de generar la secuencia de salida.



- La capa lineal, que es una red neuronal que toma el vector producido por el decodificador y lo transforma en un vector mucho más grande.
- Si el traductor aprende 10000 palabras (es decir el tamaño del vocabulario), entonces el vector de salida de la capa lineal tendrá precisamente 10000 elementos.
- La capa Softmax toma cada elemento de este vector y lo convierte en una probabilidad, todas con valores positivos entre 0 y 1.
- La posición con la probabilidad más alta será seleccionada y la palabra asociada con dicha posición será precisamente la salida del modelo en ese instante de tiempo.



La principal ventaja de la arquitectura Transformer es que no requiere de la secuencia de entrada y salida para ser procesada en orden, como ocurre en los modelos de tipo recurrente. Esto significa que la arquitectura Transformer puede procesar la entrada en paralelo, lo que la hace mucho más eficiente en términos de tiempo de entrenamiento y memoria requerida.

En resumen, la arquitectura Transformer es una red neuronal diseñada para procesar secuencias de entrada y generar una secuencia de salida. Utiliza capas de atención para aprender una representación de la entrada y generar la salida. Al utilizar capas de atención en lugar de secuencias recurrentes, la arquitectura Transformer puede procesar la entrada en paralelo, lo que la hace más eficiente y escalable para tareas de procesamiento de lenguaje natural y otros tipos de secuencias.

- [Fuente] (<https://www.codificandobits.com/blog/redes-transformer/>)

BERT

- Se ha convertido en el primer modelo de Machine Learning multipropósito capaz de interpretar el lenguaje humano en diferentes aplicaciones.
- Una Red Transformer procesa el texto en dos fases: una para la codificación encargada de procesar el texto de entrada y de codificarlo numéricamente extrayendo su información más relevante, y una fase de decodificación que se encarga de generar una nueva secuencia de texto (que es útil por ejemplo cuando se hace la traducción de un idioma a otro).

- En ambos casos cada bloque está conformado por múltiples codificadores y decodificadores.
- Es un modelo capaz de codificar el texto, obteniendo así una representación numérica que permita su correcta interpretación.
- Es el resultado de tomar la Red Transformer y quedarnos únicamente con el bloque de codificación.



- [Fuente] (<https://www.codificandobits.com/blog/bert-en-el-natural-language-processing/>)

GPT

Las redes transformers han hecho posible un avance sustancial en la generación de texto, así, Google desarrollo BERT, y luego OpenIA desarrollo ChatGPT, un modelo que además de generar texto, lo hacía de una forma aplicable a una conversación por chat con un humano. Entonces, GPT es un modelo generativo con entrenamiento no supervisado, pero ChatGPT funciona por entrenamiento por refuerzo, y está entrenado con datos residentes en plataformas como Wikipedia, al igual que BARD, la propuesta de Google análoga a ChatGPT, hay que tener en cuenta que los resultados pueden no ser exactos, por tanto, siempre vale la pena revisar la información que resulta del uso de estas herramientas. A partir de la arquitectura BERT, se puede construir una red neuronal de tipo GPT (Generative Pre-trained Transformer) que se especializa en la generación de texto. La red GPT utiliza la misma arquitectura de codificación bidireccional que BERT, pero se entrena para predecir la siguiente palabra en una secuencia de texto dada. Una vez que se tiene entrenada la red GPT, se puede implementar un modelo de lenguaje conversacional como ChatGPT, que utiliza la red GPT para generar respuestas coherentes y contextuales en una conversación en lenguaje natural. ChatGPT se entrena en una gran cantidad de datos de conversación para aprender a generar respuestas que sean relevantes y coherentes con el contexto de la conversación.

- [¿Cómo funciona ChatGPT? La revolución de la Inteligencia Artificial] (<https://www.youtube.com/watch?v=FdZ8LKiJBhQ>)
- [ChatGPT Prompt Engineering for Developers: A short course from OpenAI and DeepLearning.AI] (https://www.youtube.com/watch?v=H4YK_7MAckk)
- [Algunos precedentes, con la participación de Alan Turing: "La computadora que redactaba cartas de amor"] (https://www.bbc.com/mundo/noticias/2012/10/121015_love_letter_computer_du)

Homework