Dan Beninson
Instituto de Tecnología Nuclear

Universidad Nacional de San Martín

Comisión Nacional de Energía Atómica

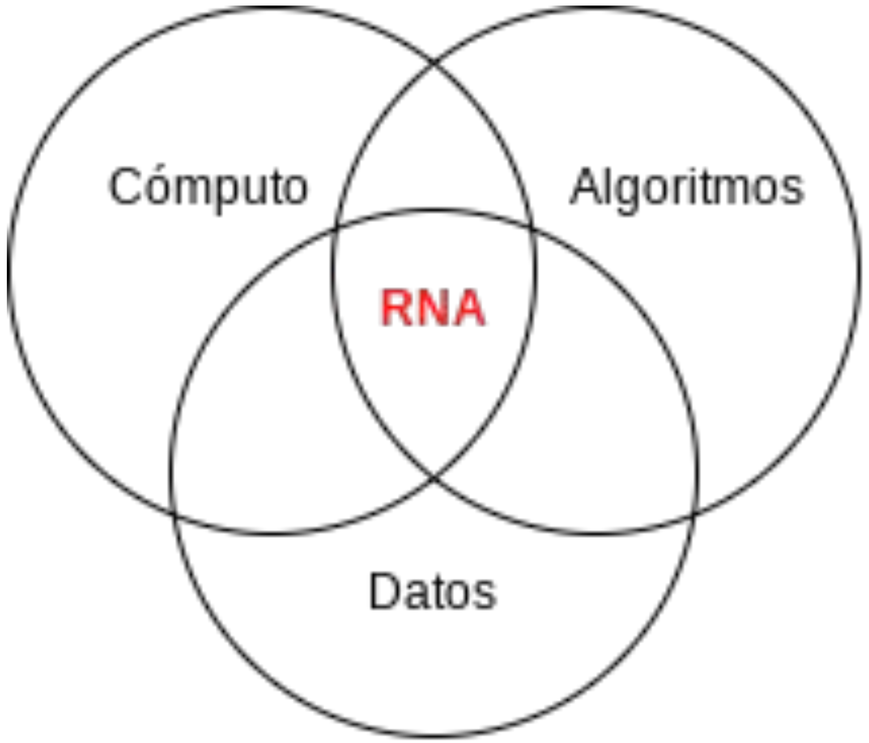
Nociones de Aprendizaje Automático para Aplicaciones Nucleares

Redes neuronales

Un poco de historia y contexto

- **Nueva ola de interés por las redes neuronales**:
  - Estamos viviendo otra gran oleada de interés por las RNA.
  - A diferencia de las anteriores, hay razones para creer que esta vez es diferente.
- **Factores clave del cambio**
  - **Datos masivos**: hoy existen enormes volúmenes de información para entrenar modelos.
  - **Rendimiento superior**: las RNA suelen superar a otras técnicas en problemas grandes y complejos.
  - **Mayor potencia de cómputo**:
    - Crecimiento sostenido por la ley de Moore.
    - Impulso de la industria del videojuego, que masificó las GPU.
  - **Algoritmos mejorados**: pequeñas variaciones en el entrenamiento han tenido impactos enormes en eficiencia y desempeño.

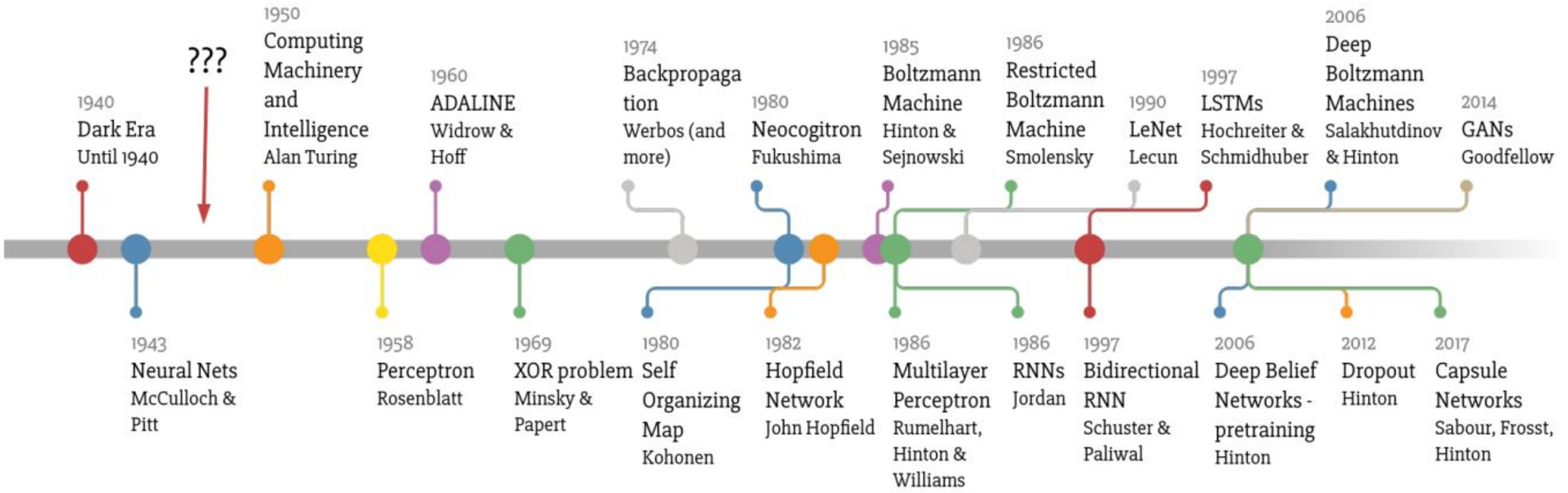
Un poco de historia y contexto



Un poco de historia y contexto

- **1943**. McCulloch and Pitts. Un modelo computacional simplificado de cómo las neuronas biológicas podrían trabajar juntas en cerebros de animales para realizar cálculos complejos utilizando lógica proposicional. Esta fue la primera arquitectura de red neuronal artificial.
- **1957**. Rosenblatt. Uno de los primeros algoritmos basados en el comportamiento de las neuronas físicas. Implementado como un programa, pero luego convertido en una máquina.
- **1969**, Minsky y Papert. En su libro "Perceptrons" demuestran que el perceptrón no puede aprender la función XOR. Comienza la “**edad oscura”** de las redes neuronales.
- Años **1980**. Primer retorno de las Redes Neuronales, pero rápidamente superada por otros algoritmos como las Máquinas de Soporte Vectorial (SVM).
- Años **2000**. Segundo retorno, impulsada por el aumento en la capacidad computacional y la disponibilidad de datos, junto con técnicas de entrenamiento eficientes.

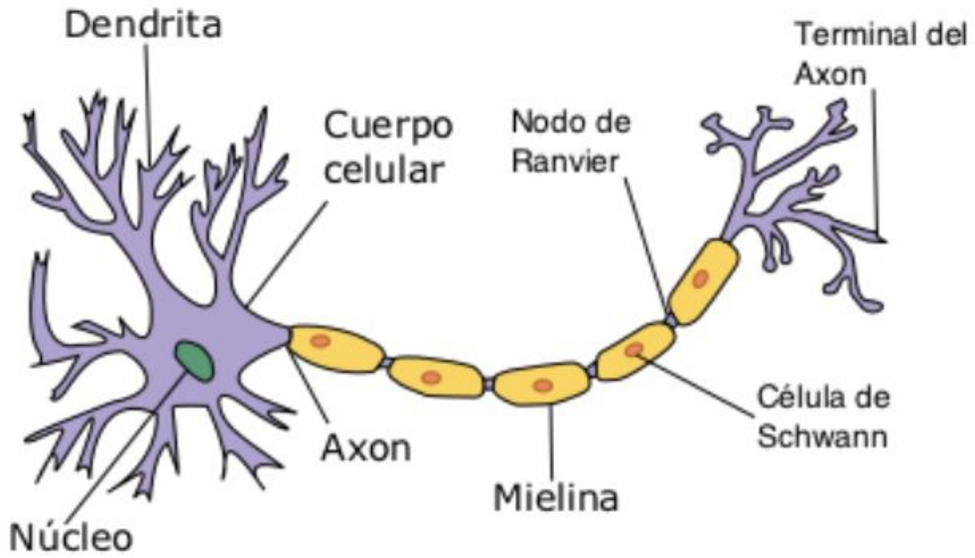
Un poco de historia y contexto



Made by Favio Vázquez

Neuronas Biológicas

- Nuestro cerebro está compuesto por cerca de 10^11 neuronas de muchos tipos. Redes de forma arbórea de nervios llamados **dendritas** están conectadas al cuerpo de la célula o **soma**, donde el núcleo de la célula está localizado.
- Desde el cuerpo se extiende una única fibra llamada **axón** y en los extremos de estos se encuentran los transmisores de las uniones sinápticas, o **sinapsis**, a otras neuronas.



Fuente: Wikipedia

Neuronas Biológicas

- La transmisión de una señal de una célula a otra en una sinapsis es un proceso químico complejo en el que se liberan sustancias transmisoras específicas desde el lado emisor de la unión.
- El efecto es aumentar o disminuir el potencial eléctrico dentro del cuerpo de la célula receptora y, si este potencial alcanza un umbral, se envía un impulso (de fuerza y duración fijas por el axón), es ahí cuando decimos que la célula ha “disparado”.
- Luego el pulso se ramifica a través de la arborización axonal a uniones sinápticas con otras células.

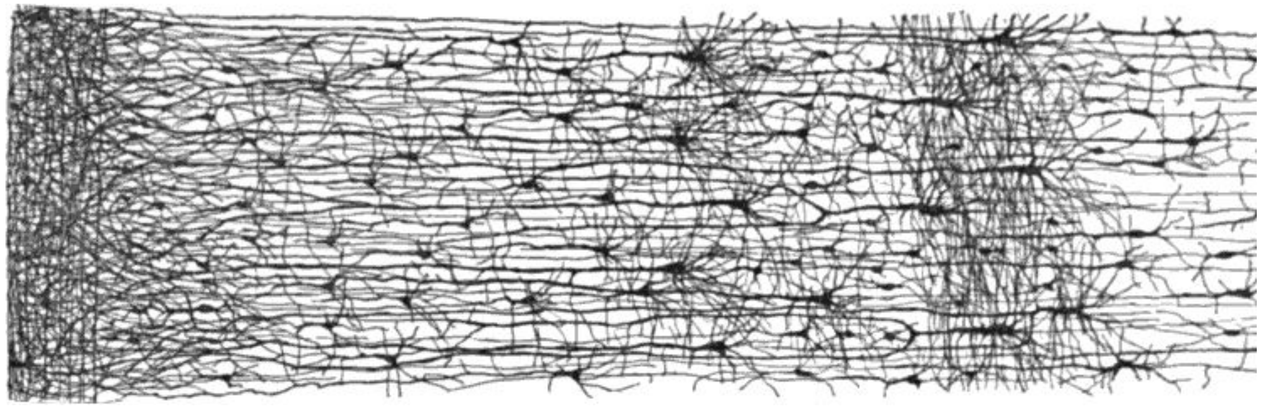


Image by Bruce Blaus (Creative Commons 3.0). Reproduced from https://en.wikipedia.org/wiki/Neuron.

Neuronas Artificiales

- McCulloch y Pitts propusieron un modelo muy simple de la neurona biológica que más tarde se conoció como neurona artificial: tiene una o más entradas (on/off) y una salida binaria.
- La neurona artificial activa su salida cuando más de un cierto número de sus entradas están activas. En su trabajo McCulloch y Pitts demostraron que, incluso con un modelo tan simplificado, es posible construir una red de neuronas artificiales que pueda calcular cualquier proposición lógica que se desee.
- En 1957 Rosenblatt desarrolla el perceptrón.

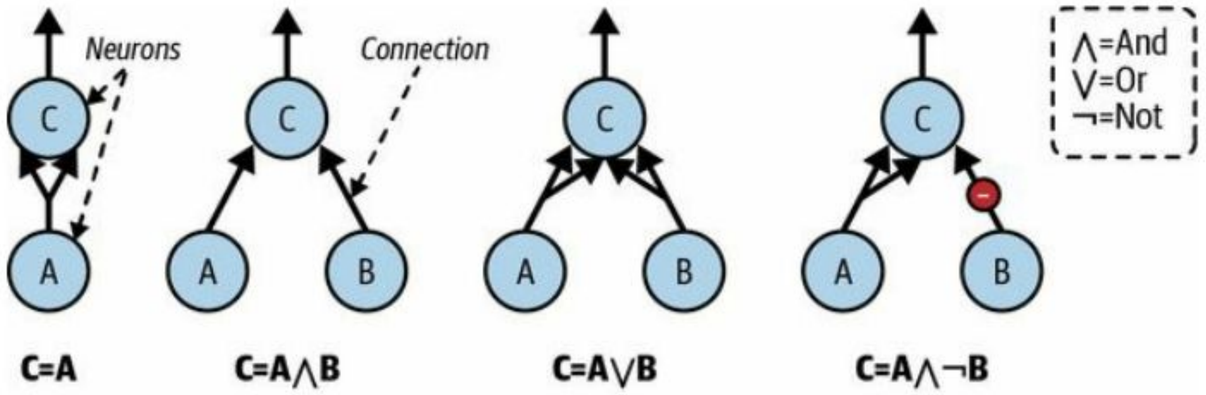


Imagen del Gerón

El Perceptrón

- El perceptrón es una de las arquitecturas más sencillas, inventada en 1957 por Frank Rosenblatt. Se basa en una neurona artificial ligeramente diferente llamada unidad lógica umbral (TLU), o a veces unidad umbral lineal (LTU). **Las entradas y salidas son números** (en lugar de valores binarios on/off), y **cada conexión de entrada está asociada a un peso**.
- La TLU calcula primero una función lineal de sus entradas y luego aplica una función escalón al resultado.
- Una única TLU puede utilizarse para una **clasificación binaria lineal simple**. Calcula una función lineal de sus entradas y, si el resultado supera un umbral, emite la clase positiva. En caso contrario, selecciona la clase negativa.

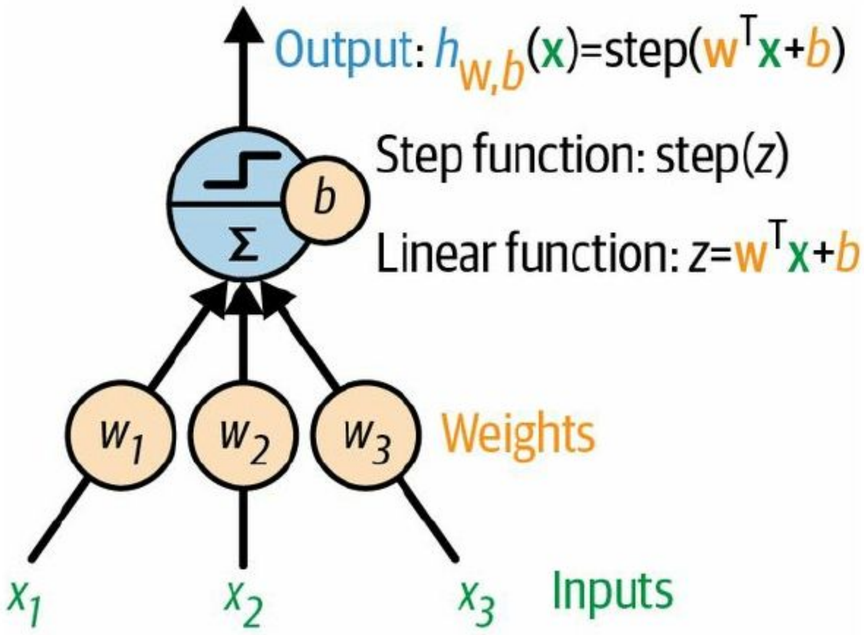
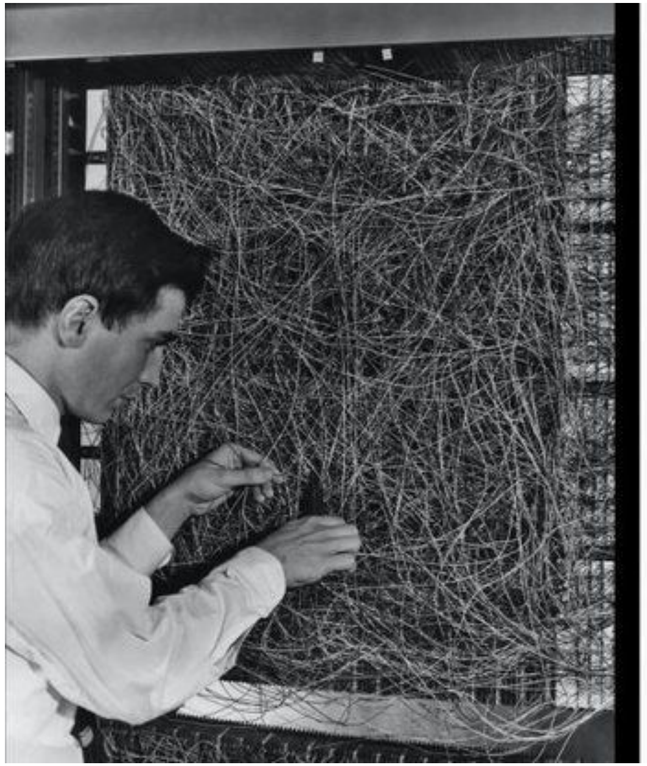


Imagen del Gerón

El Perceptrón



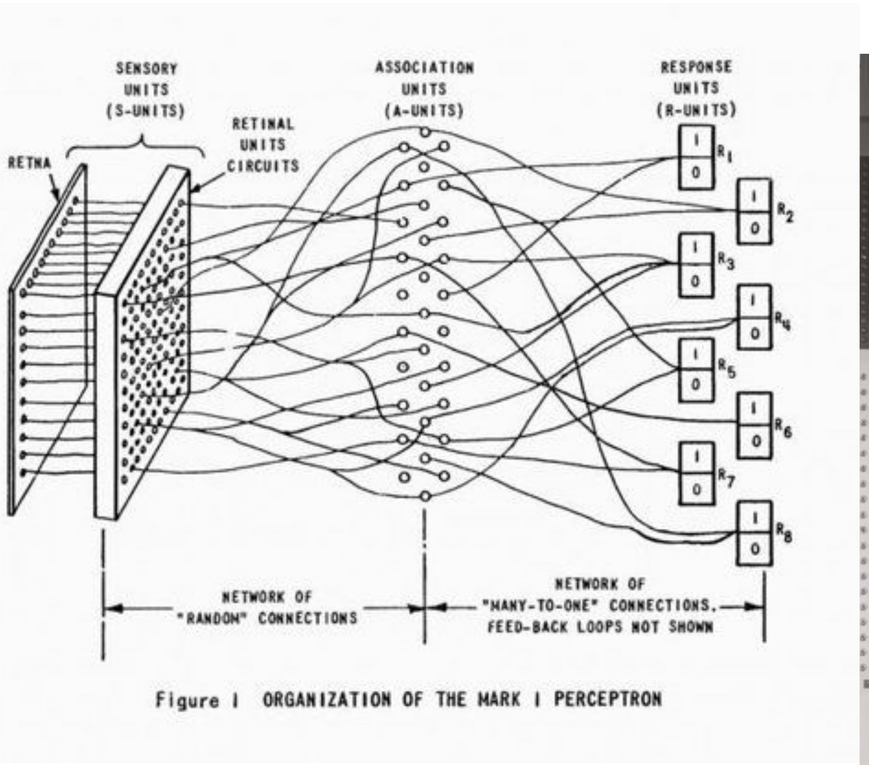
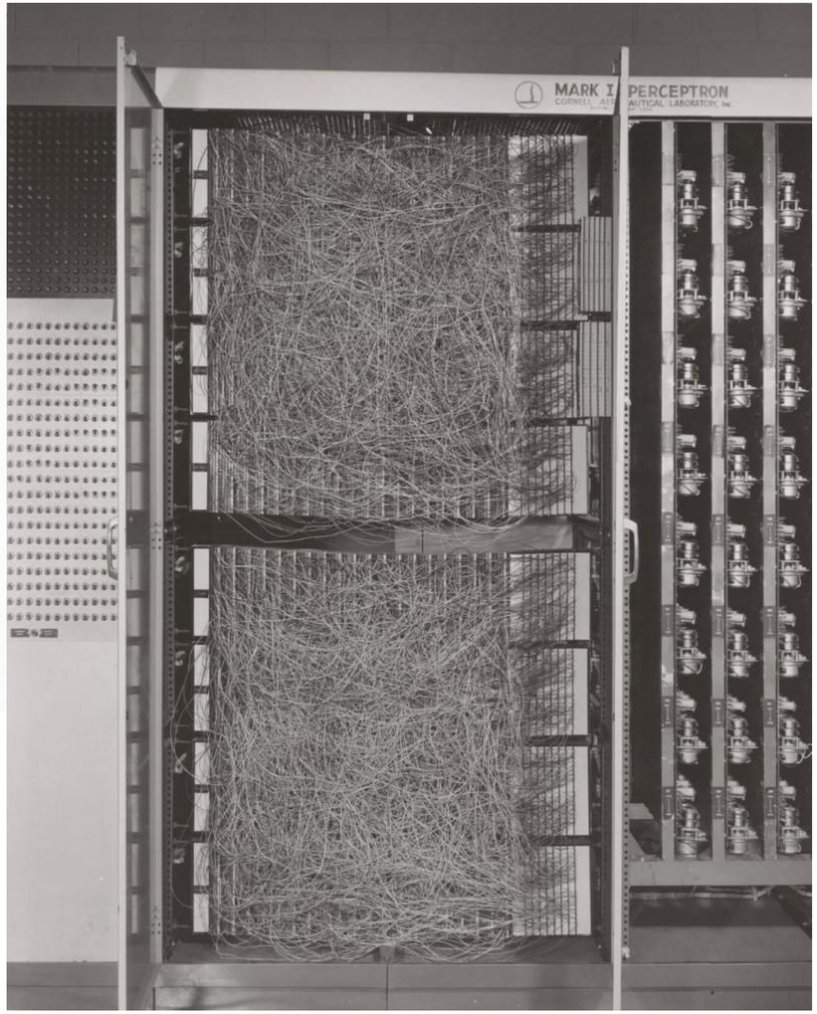


Figure 1 ORGANIZATION OF THE MARK I PERCEPTRON



El Perceptrón

- Un perceptrón se compone de una o más TLU organizadas en una sola capa, donde cada TLU está conectada a cada entrada.
- Este tipo de capa se denomina capa totalmente conectada o **capa densa**. Las entradas constituyen la capa de entrada. Y puesto que la capa de TLUs produce las salidas finales, se denomina **capa de salida**.
- **Este perceptrón puede clasificar instancias simultáneamente en tres clases binarias diferentes, lo que lo convierte en un clasificador multiclase.**

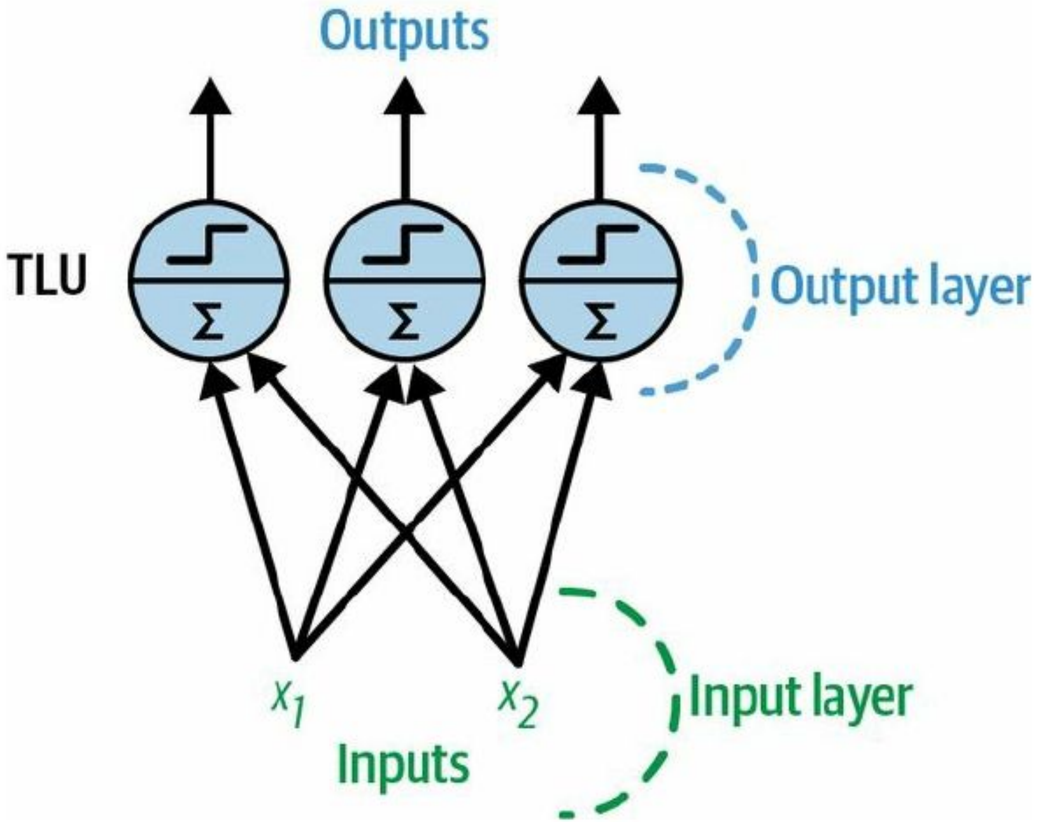


Imagen del Gerón

El Perceptrón

- ¿Cómo se entrena?
  - Donald Hebb, en su libro de 1949 "*The Organization of Behavior*", sugirió que cuando una neurona biológica activa a otra con frecuencia, la conexión entre ellas se **fortalece**.
  - Los perceptrones se entrenan usando una variante de la regla de Hebb que considera el error que comete la red al hacer una predicción.
  - La regla de aprendizaje del perceptrón **refuerza las conexiones** que ayudan a reducir los errores de predicción.
  - El perceptrón recibe un vector de entrada *X* y un vector de pesos *W* más un sesgo entrenable b (que generalmente se representa como una entrada 1 a un peso)

$$\mathbf{x} = (x_1, x_2, \ldots, x_n) \qquad \mathbf{w} = (w_1, w_2, \ldots, w_n)$$

El Perceptrón

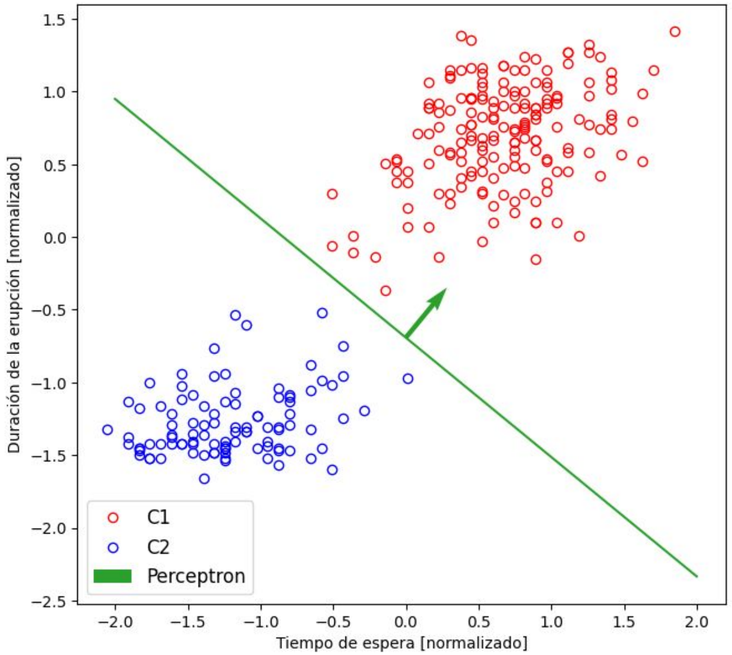
- Entonces, la salida del perceptrón se calcula aplicando la función de activación ***f***

$$\hat{y} = f(\mathbf{w} \cdot \mathbf{x} + b)$$

- Durante el entrenamiento, se compara la predicción $\hat{y}$ con la etiqueta real $y \in \{-1, +1\}$.
- Si hay error se ajustan los pesos en la dirección del error, η es la tasa de aprendizaje:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta\,(y - \hat{y})\,\mathbf{x}$$

$$b \leftarrow b + \eta\,(y - \hat{y})$$



- El proceso se repite para todas las instancias hasta que los pesos convergen o no hay más errores.

El Perceptrón

- El umbral de decisión de cada neurona de salida es lineal, por lo que los perceptrones **son incapaces de aprender patrones complejos.**
- Sin embargo, si las instancias de entrenamiento son linealmente separables, Rosenblatt demostró que este algoritmo convergerá a una solución.
- En su monografía de 1969 "*Perceptrones*", Marvin Minsky y Seymour Papert destacaron una serie de debilidades graves de los perceptrones, en particular, el hecho de que son incapaces de resolver algunos problemas triviales (por ejemplo, el problema de clasificación OR exclusivo (XOR).
- Esto es válido para cualquier otro modelo de clasificación lineal pero se esperaba mucho más de los perceptrones, y algunos quedaron tan decepcionados que abandonaron las redes neuronales en favor de problemas de más alto nivel, como la lógica y la resolución de problemas. La falta de aplicaciones prácticas tampoco ayudó.

El Perceptrón Multicapa

- Resulta que algunas de las limitaciones de los perceptrones pueden eliminarse apilando múltiples perceptrones.
- La RNA resultante se denomina **perceptrón multicapa** (MLP).
- Un MLP puede resolver el problema XOR.

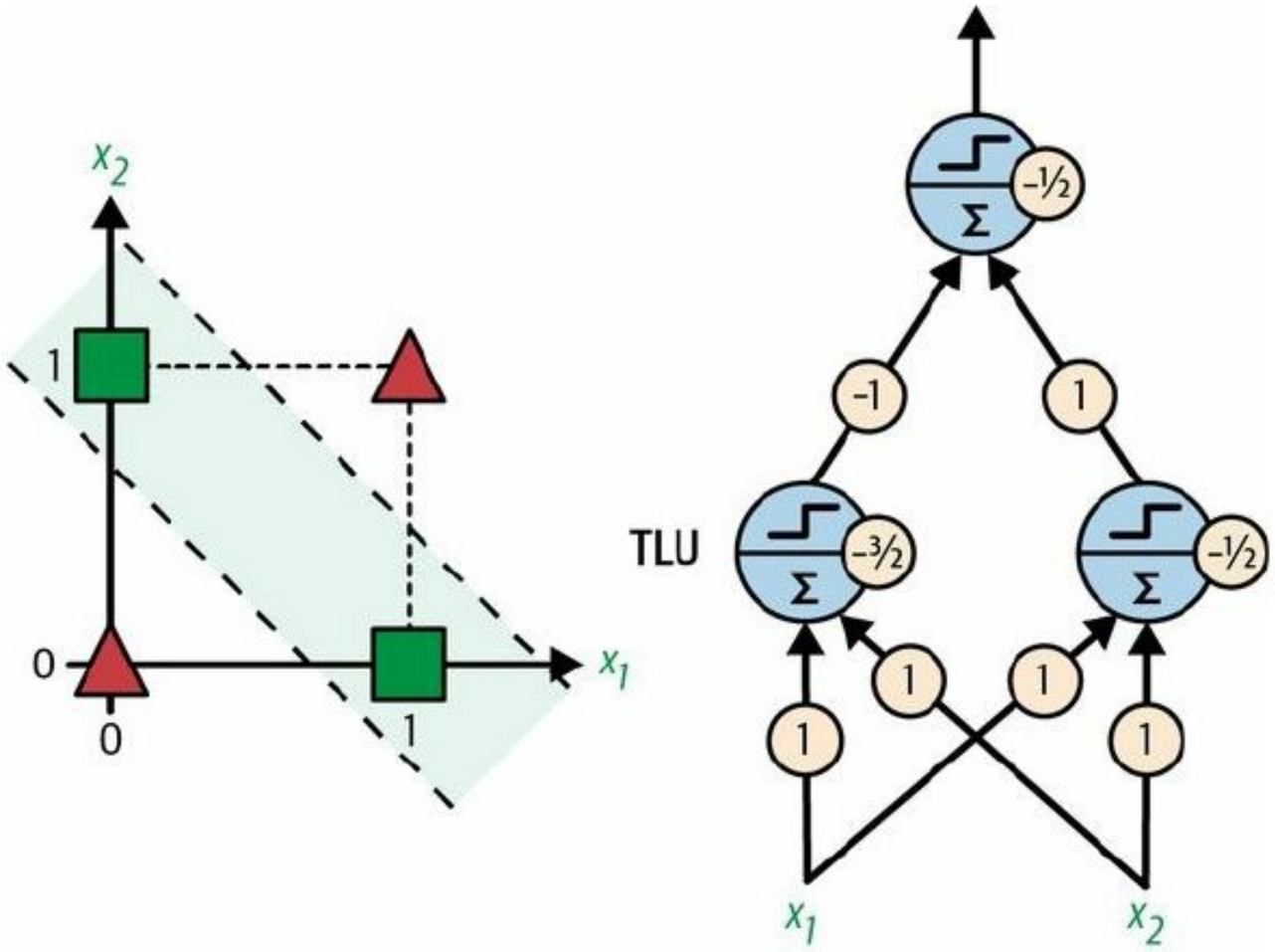


Imagen del Gerón

¿Qué es una Red Neuronal? Parte 1 : La Neurona | DotCSV

El Perceptrón Multicapa

- Un MLP se compone de **una capa de entrada**, una o más capas de TLUs llamadas **capas ocultas**, y una capa final de TLUs llamada capa de **salida**.
- El problema es que era **muy dificil de entrenar,** a principios de la década de 1960, varios investigadores discutieron la posibilidad de utilizar el *descenso de gradiente* para entrenar redes neuronales, pero esto requiere hacer cálculos del error del modelo con respecto a sus parámetros; no estaba claro en ese momento cómo hacer esto de manera eficiente con un modelo como este (especialmente con las computadoras que tenían en ese entonces).

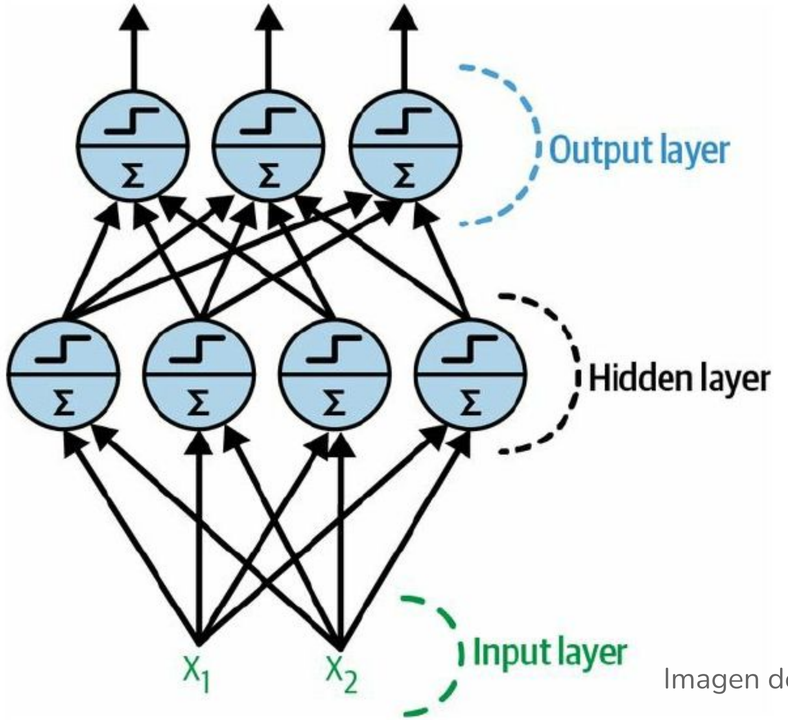


Imagen del Gerón

El Perceptrón Multicapa

- Un MLP se compone de **una capa de entrada**, una o más capas de TLUs llamadas **capas ocultas**, y una capa final de TLUs llamada capa de **salida**.
- El problema es que era **muy dificil de entrenar,** a principios de la década de 1960, varios investigadores discutieron la posibilidad de utilizar el *descenso de gradiente* para entrenar redes neuronales, pero esto requiere hacer cálculos del error del modelo con respecto a sus parámetros; no estaba claro en ese momento cómo hacer esto de manera eficiente con un modelo como este (especialmente con las computadoras que tenían en ese entonces).

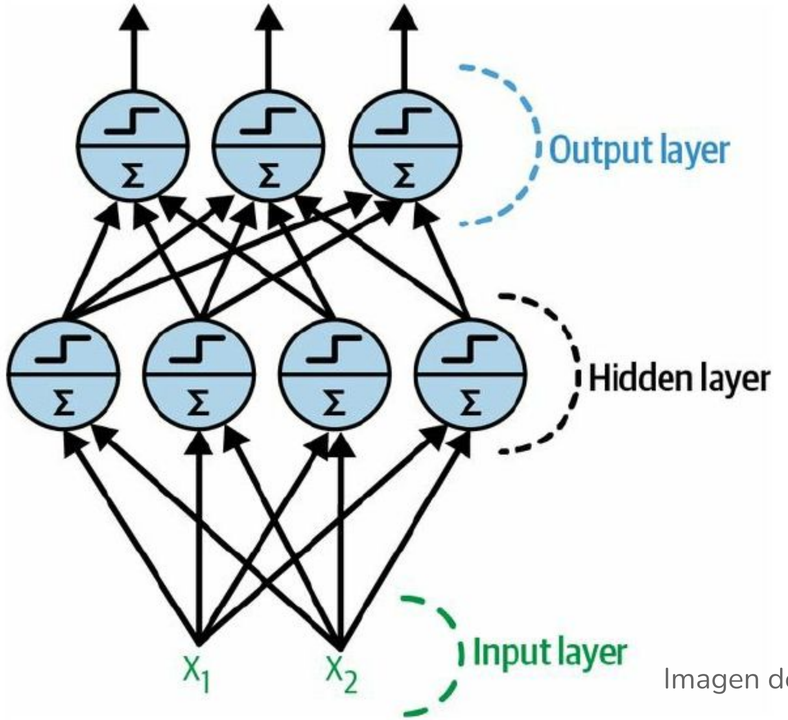


Imagen del Gerón

Cuando una RNA contiene muchas capas ocultas se denomina red neuronal profunda (DNN).

Funciones de Activación

- Si bien los MLP permiten aprender muchos patrones necesitamos darle más poder expresivo para que pueda realizar tareas que vayan más allá de la clasificación lineal.
- Pensemos que si tenemos un problema de regresión vamos a necesitar poder predecir un número, y hasta ahora solo podemos predecir 0 o 1 (o -1, 1) .
- Para que pueda realizar tareas más complejas lo que se puede cambiar es la función escalón para que pueda devolver cualquier número.
- Esto no solo sirve para poder resolver problemas de regresión, al usarlo en todas las unidades de la red le agrega una *no linealidad* que es uno de los factores que le da a las redes neuronales todo su poder.

Funciones de Activación

- Si bien los MLP permiten aprender muchos patrones necesitamos darle más poder expresivo para que pueda realizar tareas que vayan más allá de la clasificación lineal.
- Pensemos que si tenemos un problema de regresión vamos a necesitar poder predecir un número, y hasta ahora solo podemos predecir 0 o 1 (o -1, 1) .
- Para que pueda realizar tareas más complejas lo que se puede cambiar es la función escalón para que pueda devolver cualquier número.
- Esto no solo sirve para poder resolver problemas de regresión, al usarlo en todas las unidades de la red le agrega una *no linealidad* que es uno de los factores que le da a las redes neuronales todo su poder.

Teorema de aproximación universal: Cada función continua que mapea intervalos de números reales a algún intervalo de salida de números reales puede ser aproximado arbitrariamente por un perceptrón multicapa con una sola capa oculta.

Funciones de Activación

- **Capas internas**
  - **Función sigmoide** o logística: Da valores entre 0 y 1.
  - **Tangente hiperbólica**: Da valores entre -1 y 1, similar a la sigmoide, pero centrada en 0, lo que suele llevar a un mejor rendimiento en ciertas redes.
  - **ReLU** (Rectified Linear Unit): Da 0 si la entrada es negativa, y la misma entrada si es positiva.

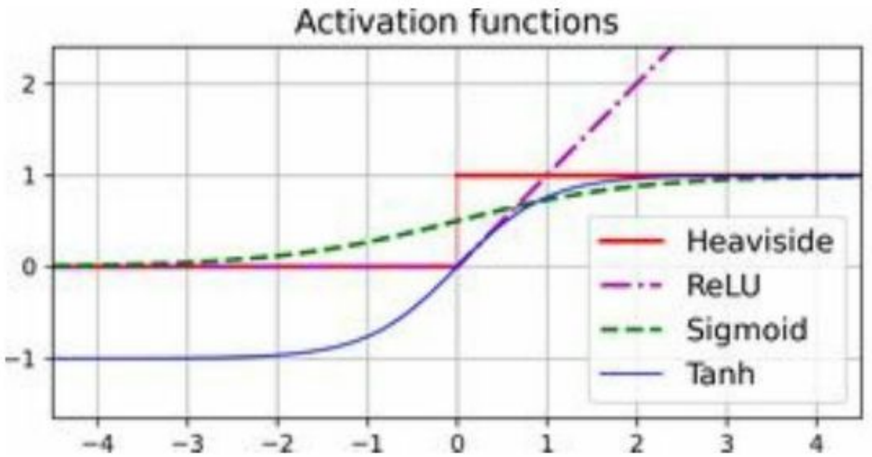


Imagen del Gerón

Funciones de Activación

- **Capas internas**
  - **Función sigmoide** o logística: Da valores entre 0 y 1.
  - **Tangente hiperbólica**: Da valores entre -1 y 1, similar a la sigmoide, pero centrada en 0, lo que suele llevar a un mejor rendimiento en ciertas redes.
  - **ReLU** (Rectified Linear Unit): Da 0 si la entrada es negativa, y la misma entrada si es positiva.
- **Capas de salida**
  - **Función sigmoide (si, también):** Como da valores entre 0 y 1 se puede interpretar como una probabilidad, útil en problemas de **clasificación binaria**.
  - **Softmax:** Utilizada en la capa de salida de redes para clasificación multiclase, convierte los valores de entrada en probabilidades para cada clase.
  - **Activación lineal:** Si el problema es de regresión se puede optar por no ponerle función de activación a la salida y simplemente devolver el número.

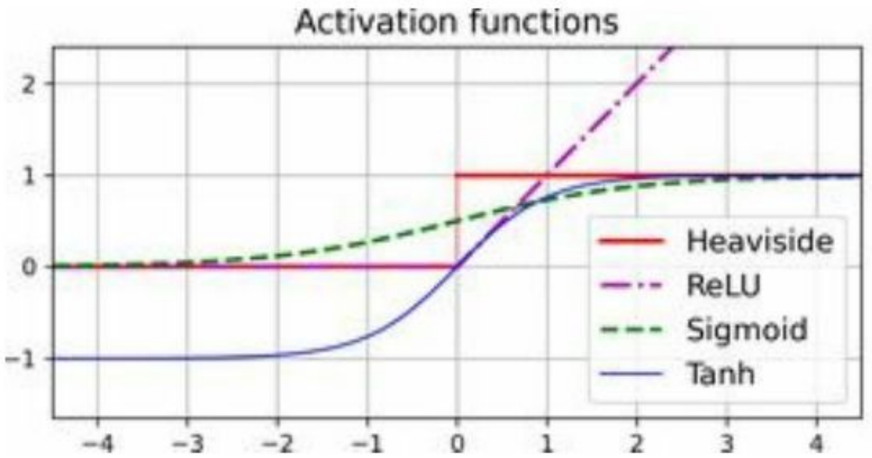


Imagen del Gerón

Funciones de Activación

- La elección de las funciones de activación depende del tipo de problema a resolver (regresión, clasificación, etc.) y de las características de la red neuronal.
  - En las capas ocultas, ReLU y sus variantes suelen ser una elección común.
  - En la capa de salida, **Softmax** es usada para clasificación multiclase, **sigmoide** para clasificación binaria y **lineal** para problemas de regresión.
- También se tiene que definir la **función de pérdida**, esto es, la función que va a medir el error en las predicciones.

Función de Pérdida

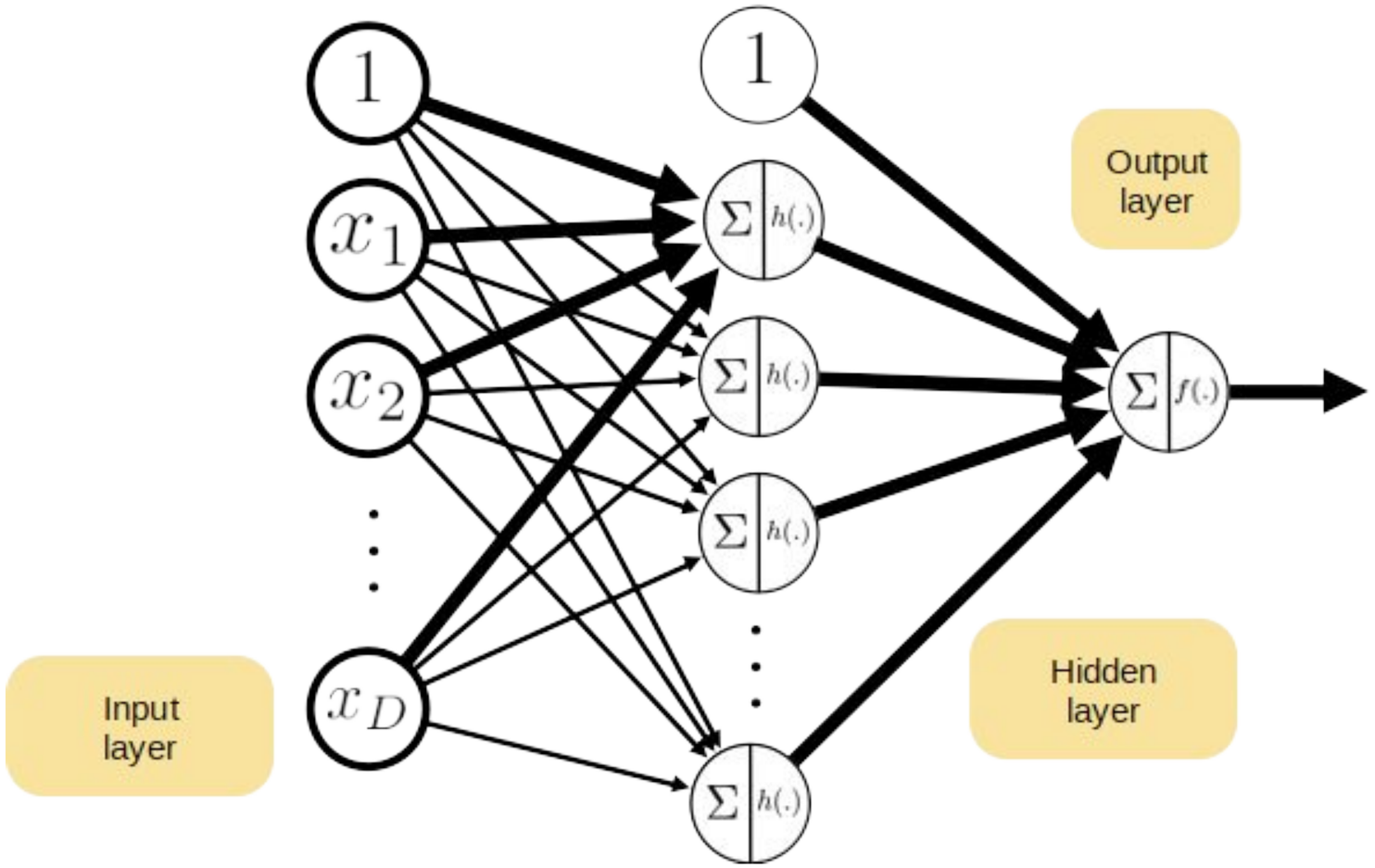
- La función de pérdida mide cuánto difieren las predicciones del modelo respecto a los valores reales.
- Es el **objetivo** que se minimiza durante el entrenamiento mediante el descenso por gradiente.
- Su elección depende del tipo de problema y de la salida de la red.

Martín Makler A.P. 4.0 (2023)

	Capa de salida		
Problema	**Tamaño**	**Activación**	**Error**
Regresión	N	$f(x) = x$	MSE RMSE
Clasificación Binaria	1	$f(x)$ = sigmoide	Cross-entropy
Clasificación Multi-class	K	$f(x)$ = softmax	Multiclass Cross-entropy

La entropía cruzada mide las discrepancias entre dos distribuciones de probabilidad.
Neural Networks Part 6: Cross Entropy

Esquema general



Feed-forward fully-connected

$$y = f\left(w_0^{(2)} + \sum_{j=1}^{M} w_j^{(2)} z_j\right)$$

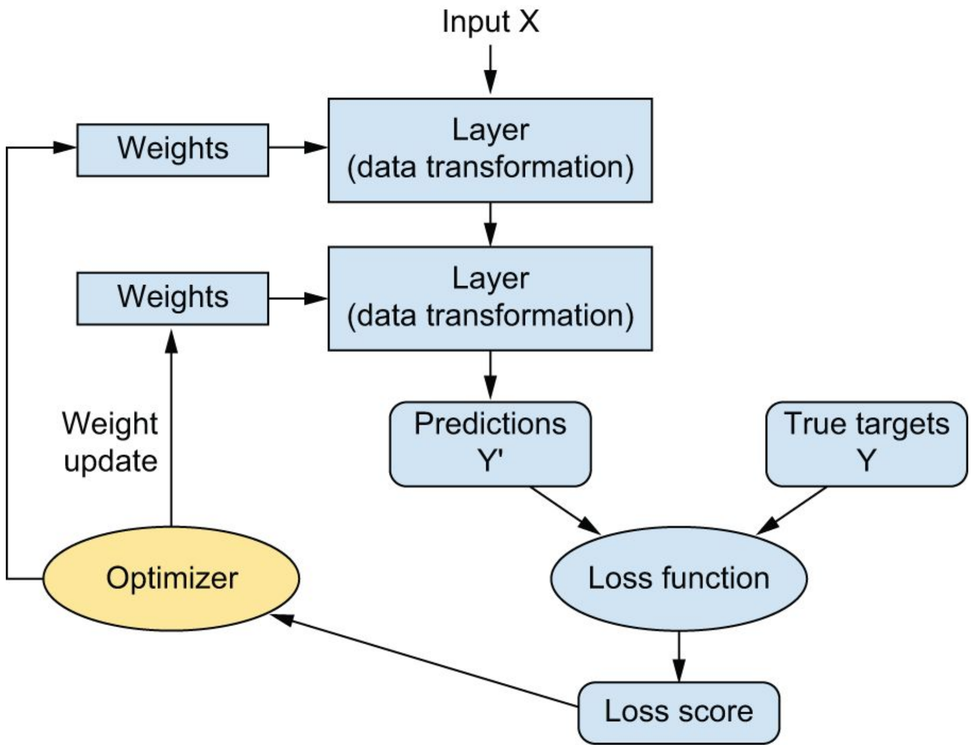
where

$$z_j = h\left(w_{j0}^{(1)} + \sum_{i=1}^{D} w_{ji}^{(1)} x_i\right)$$

Capítulo 5 del Bishop

Martín Makler A.P. 4.0 (203)

Esquema general



DEEP LEARNING with Python (link)

Backpropagation

- Recién en 1970 Seppo Linnainmaa introduce una técnica para calcular todos los gradientes automática y eficientemente.
- Este algoritmo se denomina ahora **diferenciación automática de modo inverso**. En sólo dos pasadas por la red **(una hacia delante y otra hacia atrás)**, es capaz de calcular los gradientes del error de la red neuronal con respecto a cada uno de los parámetros del modelo.
- En otras palabras, **puede averiguar cómo debe ajustarse cada peso de conexión** y cada sesgo para reducir el error de la red neuronal. Estos gradientes se pueden utilizar para realizar un paso de descenso de gradiente. Si repite este proceso de calcular los gradientes automáticamente y realizar un paso de descenso de gradiente, el error de la red neuronal disminuirá gradualmente hasta que finalmente alcance un mínimo.
- Esta combinación de diferenciación automática en modo inverso y descenso de gradiente se denomina ahora **retropropagación** (o *backpropagation*).

3blue1brown: Neural Networks (videos 1 al 3)

Backpropagation (Intuición)

- Una red neuronal es una cadena de funciones encadenadas: cada capa transforma la salida de la anterior.

$$y = f_3(f_2(f_1(x)))$$

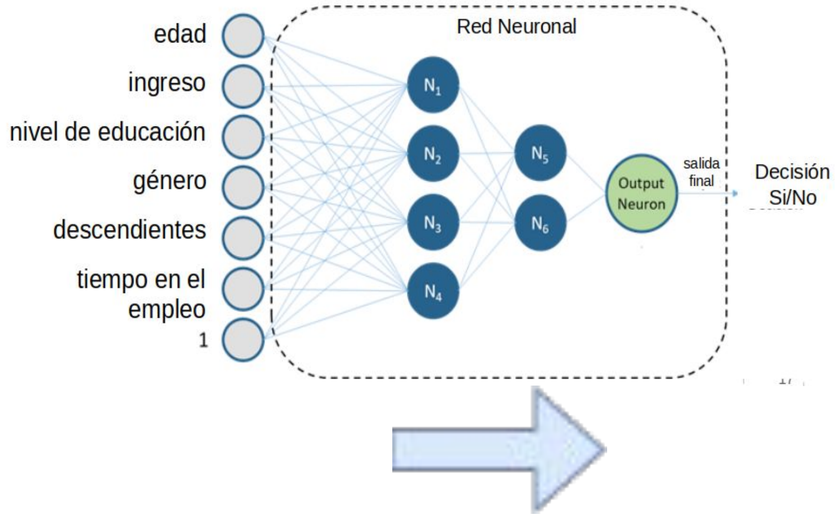
- La pérdida ***L*** depende de la salida ***y***, que a su vez depende de todas las capas y, en consecuencia, de todos los pesos intermedios. Para saber cómo cambiar un peso interno, debemos saber cómo su cambio afecta al error total.
- Gracias a la regla de la cadena, podemos “propagar” esa influencia hacia atrás:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_i}$$

- Cada derivada cuenta cómo un pequeño cambio en un peso afecta la salida final y, por lo tanto, el error.
- Así, derivar “todo en el medio” permite repartir la responsabilidad del error entre todos los parámetros y ajustar cada uno en la dirección correcta.

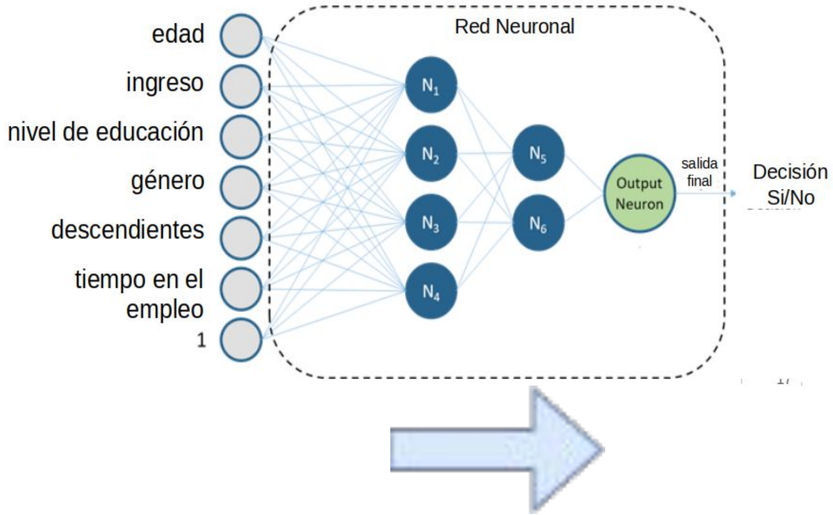
Backpropagation

1. La red toma un subconjunto de los datos de entrenamiento (*mini-batch*) y lo utiliza como entrada. Se calcula la salida de cada capa, que a su vez se utiliza como entrada para la siguiente capa, hasta llegar a la capa de salida.



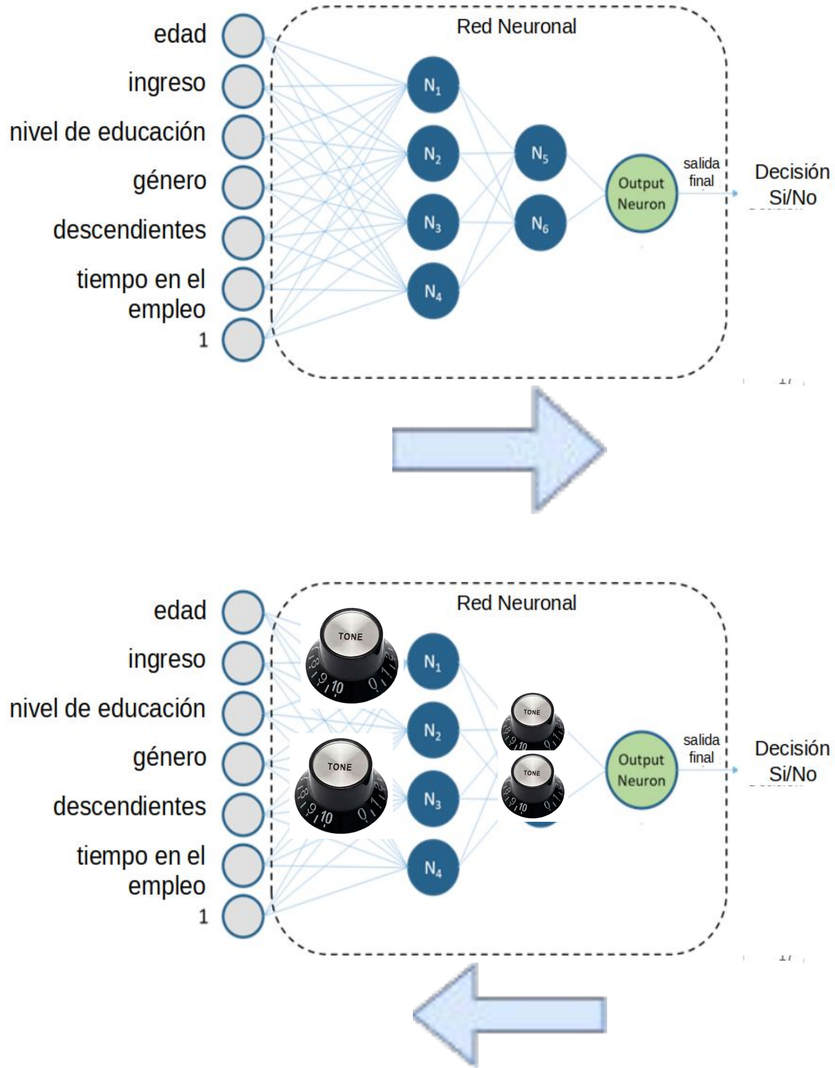
Backpropagation

1. La red toma un subconjunto de los datos de entrenamiento (*mini-batch*) y lo utiliza como entrada. Se calcula la salida de cada capa, que a su vez se utiliza como entrada para la siguiente capa, hasta llegar a la capa de salida.
2. Para cada *mini-batch*, la función de pérdida compara la salida de la red con el resultado deseado (la etiqueta) y devuelve una medida del **error**.



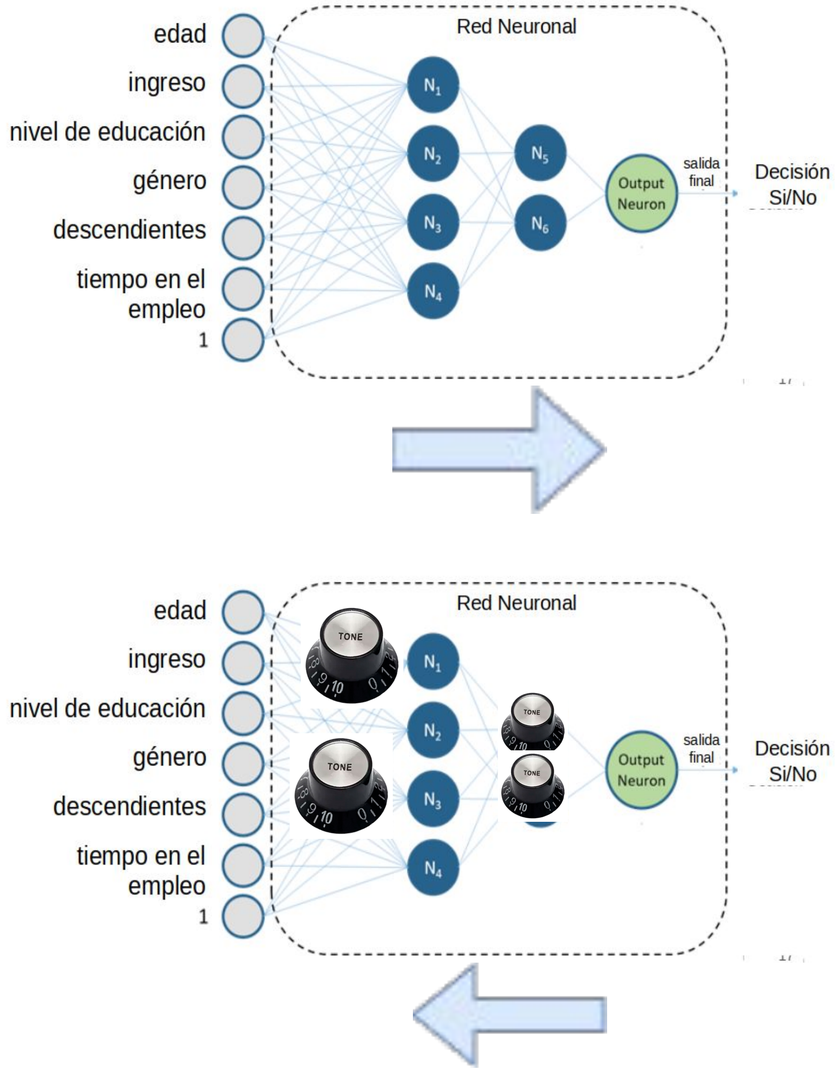
Backpropagation

1. La red toma un subconjunto de los datos de entrenamiento (*mini-batch*) y lo utiliza como entrada. Se calcula la salida de cada capa, que a su vez se utiliza como entrada para la siguiente capa, hasta llegar a la capa de salida.
2. Para cada *mini-batch*, la función de pérdida compara la salida de la red con el resultado deseado (la etiqueta) y devuelve una medida del **error**.
3. Luego, se calcula cuánto contribuyó al error cada parámetro de la red (pesos y sesgos) empezando por la capa de salida. Esto se realiza aplicando la regla de la cadena para propagar los gradientes hacia atrás, capa por capa, hasta llegar a la capa de entrada. Al finalizar, se obtiene el gradiente del error con respecto a todos los parámetros de la red.



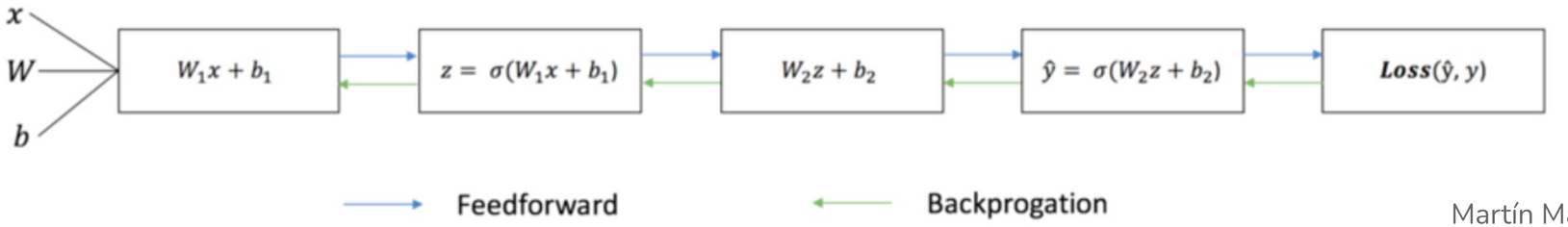
Backpropagation

1. La red toma un subconjunto de los datos de entrenamiento (*mini-batch*) y lo utiliza como entrada. Se calcula la salida de cada capa, que a su vez se utiliza como entrada para la siguiente capa, hasta llegar a la capa de salida.
2. Para cada *mini-batch*, la función de pérdida compara la salida de la red con el resultado deseado (la etiqueta) y devuelve una medida del **error**.
3. Luego, se calcula cuánto contribuyó al error cada parámetro de la red (pesos y sesgos) empezando por la capa de salida. Esto se realiza aplicando la regla de la cadena para propagar los gradientes hacia atrás, capa por capa, hasta llegar a la capa de entrada. Al finalizar, se obtiene el gradiente del error con respecto a todos los parámetros de la red.
4. Finalmente, se aplica un paso de **descenso de gradiente** para ajustar los pesos y sesgos, minimizando el error.



En resumen...

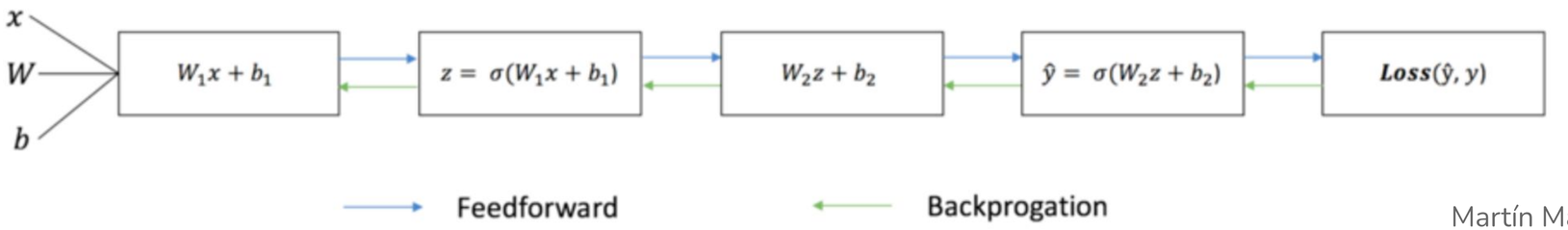
- Entrenar/aprender es obtener los pesos *Wj* que minimizan la función de pérdida.
  - Necesita las direcciones para mover *Wj*.
- Técnicamente es muy difícil de minimizar: muchos parámetros, altamente no lineal
- Una función de activación diferenciable permite el uso directo del descenso del gradiente y otros algoritmos de optimización para el ajuste de los pesos.
  - Elegante truco matemático: ***Backpropagation***
    - Realiza predicciones para un *mini-batch*, mide el error, recorre cada capa a la inversa para medir la contribución de error de cada parámetro y, por último, ajusta los pesos y sesgos para reducir el error.



Martín Makler A.P. 4.0 (2023)

En resumen...

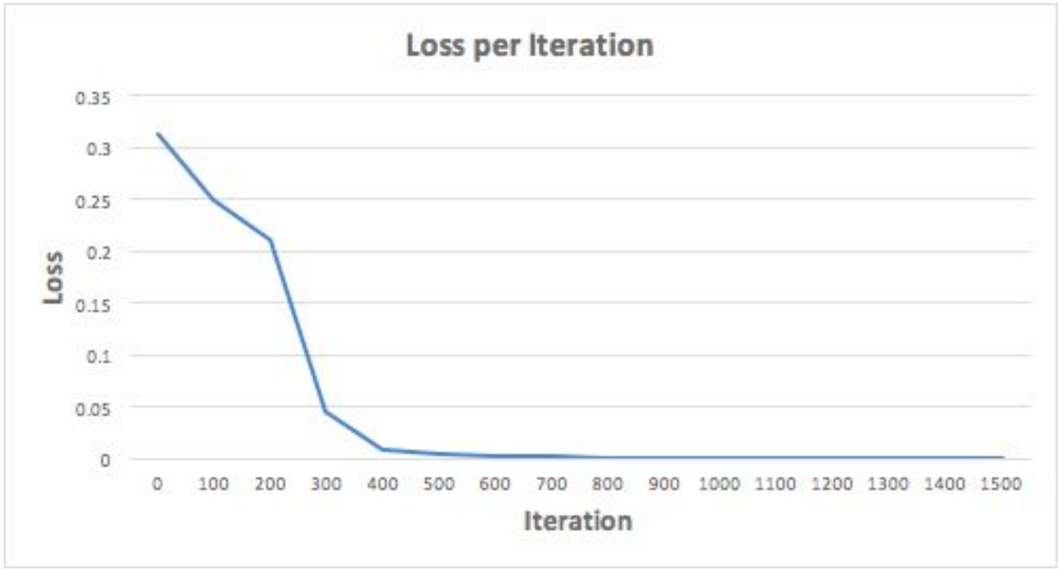
- Entrenar/aprender es obtener los pesos *Wj* que minimizan la función de pérdida.
  - Necesita las direcciones para mover *Wj*.
- Técnicamente es muy difícil de minimizar: muchos parámetros, altamente no lineal
- Una función de activación diferenciable permite el uso directo del descenso del gradiente y otros algoritmos de optimización para el ajuste de los pesos.
  - Elegante truco matemático: ***Backpropagation***
    - Realiza predicciones para un *mini-batch*, mide el error, recorre cada capa a la inversa para medir la contribución de error de cada parámetro y, por último, ajusta los pesos y sesgos para reducir el error.



Martín Makler A.P. 4.0 (2023)

En resumen...

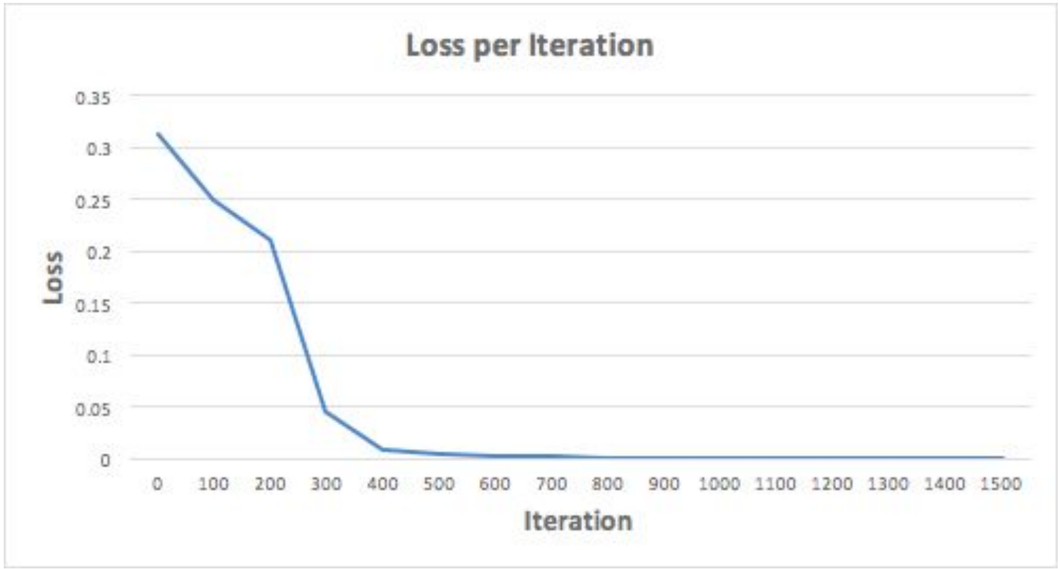
- Después de cada iteración se actualizan los pesos.
- La tasa de aprendizaje determina el "tamaño del paso".
- Después de muchas iteraciones, el proceso debería converger.



Martín Makler A.P. 4.0 (2023)

En resumen...

- Después de cada iteración se actualizan los pesos.
- La tasa de aprendizaje determina el "tamaño del paso".
- Después de muchas iteraciones, el proceso debería converger.



Martín Makler A.P. 4.0 (2023)

Cada pasada por todo el conjunto de entrenamiento es una época (*epoch*), la cantidad de épocas es un hiperparámetro a definir, el tamaño del *batch* es otro y también el paso (*learning rate*).

Ya tenemos una red neuronal

- Con todo esto ya tenemos todo lo necesario para armar nuestra red neuronal, lo que tenemos que hacer es:

Ya tenemos una red neuronal

- Con todo esto ya tenemos todo lo necesario para armar nuestra red neuronal, lo que tenemos que hacer es:
  - Definir cuantas capas de neuronas vamos a querer (es todo un tema...).

Ya tenemos una red neuronal

- Con todo esto ya tenemos todo lo necesario para armar nuestra red neuronal, lo que tenemos que hacer es:
  - Definir cuantas capas de neuronas vamos a querer (es todo un tema...).
  - La entrada ya está definida por los datos que vamos a usar (un problema menos).

Ya tenemos una red neuronal

- Con todo esto ya tenemos todo lo necesario para armar nuestra red neuronal, lo que tenemos que hacer es:
  - Definir cuantas capas de neuronas vamos a querer (es todo un tema...).
  - La entrada ya está definida por los datos que vamos a usar (un problema menos).
  - La función de activación de la salida la define el problema (otro problema menos).

Ya tenemos una red neuronal

- Con todo esto ya tenemos todo lo necesario para armar nuestra red neuronal, lo que tenemos que hacer es:
  - Definir cuantas capas de neuronas vamos a querer (es todo un tema...).
  - La entrada ya está definida por los datos que vamos a usar (un problema menos).
  - La función de activación de la salida la define el problema (otro problema menos).
  - Ídem para la función de pérdida.

Ya tenemos una red neuronal

- Con todo esto ya tenemos todo lo necesario para armar nuestra red neuronal, lo que tenemos que hacer es:
  - Definir cuantas capas de neuronas vamos a querer (es todo un tema...).
  - La entrada ya está definida por los datos que vamos a usar (un problema menos).
  - La función de activación de la salida la define el problema (otro problema menos).
  - Ídem para la función de pérdida.
  - Definir las funciones de activación de las capas internas (se puede probar como funcionan otras, pero se suele usar ReLU).

Ya tenemos una red neuronal

- Con todo esto ya tenemos todo lo necesario para armar nuestra red neuronal, lo que tenemos que hacer es:
  - Definir cuantas capas de neuronas vamos a querer (es todo un tema...).
  - La entrada ya está definida por los datos que vamos a usar (un problema menos).
  - La función de activación de la salida la define el problema (otro problema menos).
  - Ídem para la función de pérdida.
  - Definir las funciones de activación de las capas internas (se puede probar como funcionan otras, pero se suele usar ReLU).
- Una vez definido esto se entrena y que backpropagation haga su magia.

Ya tenemos una red neuronal

- Con todo esto ya tenemos todo lo necesario para armar nuestra red neuronal, lo que tenemos que hacer es:
  - Definir cuantas capas de neuronas vamos a querer (es todo un tema...).
  - La entrada ya está definida por los datos que vamos a usar (un problema menos).
  - La función de activación de la salida la define el problema (otro problema menos).
  - Ídem para la función de pérdida.
  - Definir las funciones de activación de las capas internas (se puede probar como funcionan otras, pero se suele usar ReLU).
- Una vez definido esto se entrena y que backpropagation haga su magia.
- ¿Listo? Ni cerca, elegir las neuronas por capa y la cantidad de capas no es trivial básicamente hay que probar que combinación funciona mejor y tenemos el cuco de siempre...

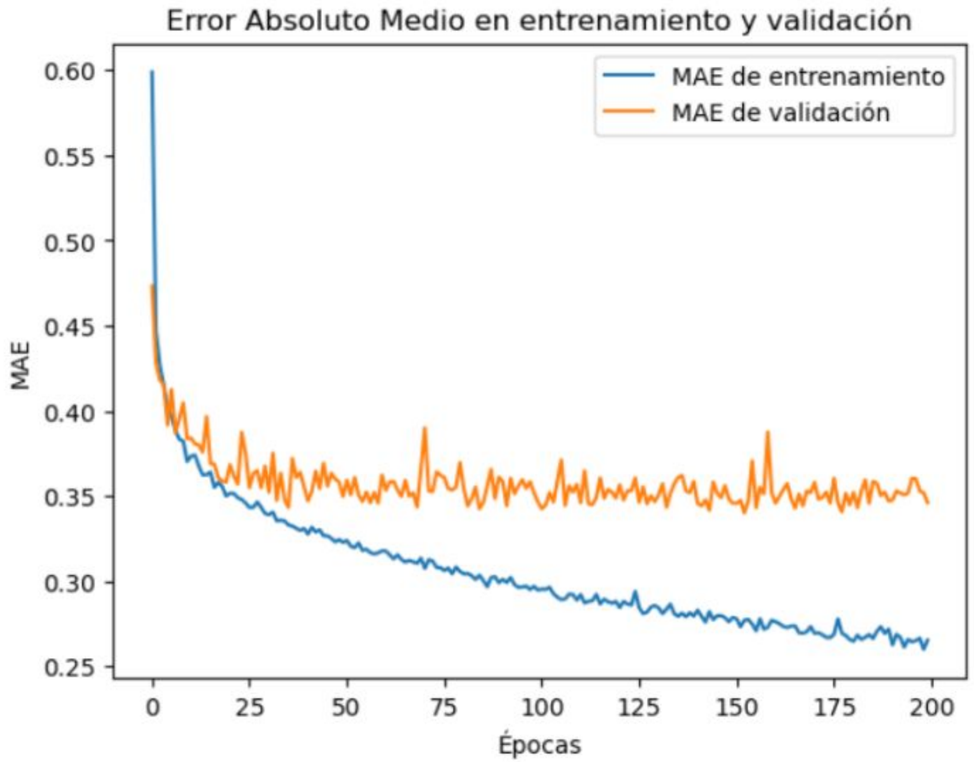
Ya tenemos una red neuronal

- Con todo esto ya tenemos todo lo necesario para armar nuestra red neuronal, lo que tenemos que hacer es:
  - Definir cuantas capas de neuronas vamos a querer (es todo un tema...).
  - La entrada ya está definida por los datos que vamos a usar (un problema menos).
  - La función de activación de la salida la define el problema (otro problema menos).
  - Ídem para la función de pérdida.
  - Definir las funciones de activación de las capas internas (se puede probar como funcionan otras, pero se suele usar ReLU).
- Una vez definido esto se entrena y que backpropagation haga su magia.
- ¿Listo? Ni cerca, elegir las neuronas por capa y la cantidad de capas no es trivial básicamente hay que probar que combinación funciona mejor y tenemos el cuco de siempre **el overfitting.**

Overfitting

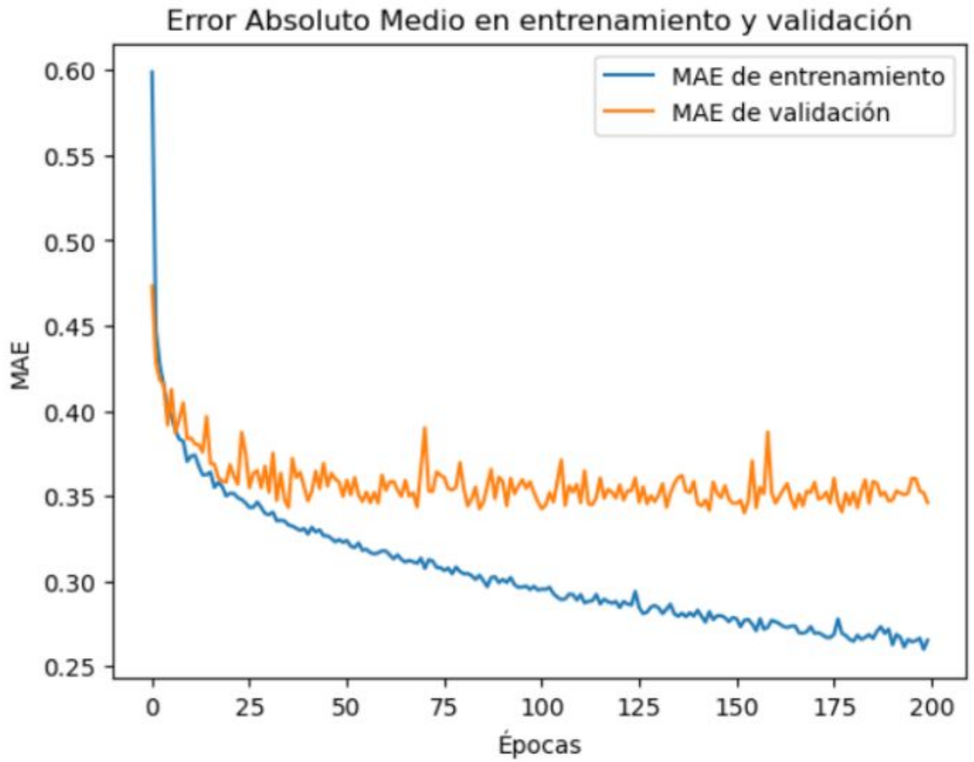
- Las redes neuronales profundas suelen tener decenas de miles de parámetros, a veces incluso millones, como ya vimos esto les da mucha libertad y significa que pueden adaptarse a una enorme variedad de conjuntos de datos complejos.
- Pero esta gran flexibilidad también hace que la red sea propensa a sobreajustar el conjunto de entrenamiento (quizás más que con otros métodos más sencillos).
- Una de las formas más sencillas para tratar de evitar esto es volver a usar el conjunto de validación que ya conocemos y activar lo que conoce como **Early Stopping.**

Overfitting



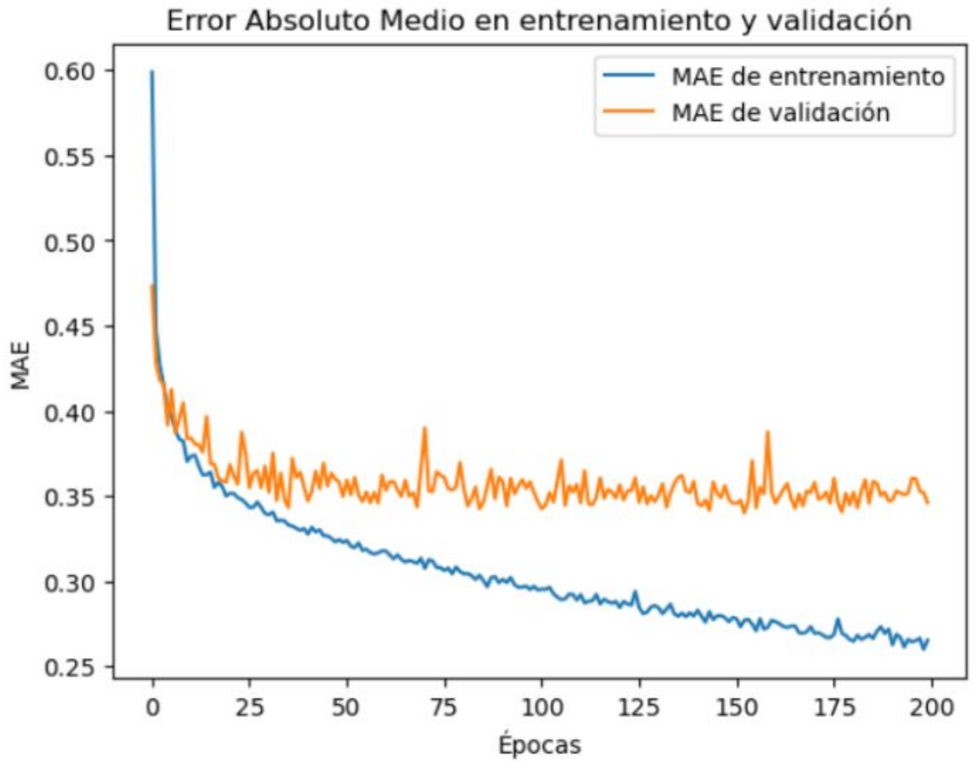
- Como ya vimos, durante el entrenamiento, el modelo es evaluado en cada época utilizando la función de pérdida. Si definimos un conjunto de validación también se calcula este resultado ahí.

Overfitting



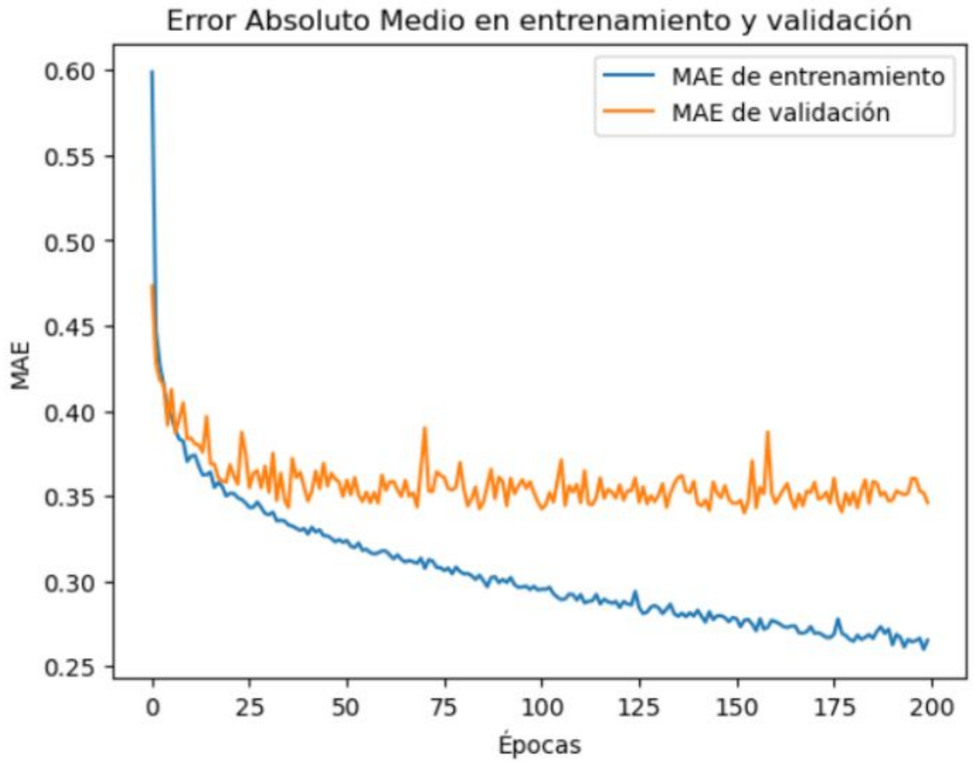
- Como ya vimos, durante el entrenamiento, el modelo es evaluado en cada época utilizando la función de pérdida. Si definimos un conjunto de validación también se calcula este resultado ahí.
- La pérdida en entrenamiento *siempre* va a bajar (generalmente, de hecho si no baja es un problema...).

Overfitting



- Como ya vimos, durante el entrenamiento, el modelo es evaluado en cada época utilizando la función de pérdida. Si definimos un conjunto de validación también se calcula este resultado ahí.
- La pérdida en entrenamiento *siempre* va a bajar (generalmente, de hecho si no baja es un problema...).
- Si la pérdida en el conjunto de validación deja de mejorar o comienza a empeorar después de un cierto número de épocas consecutivas (definido por un parámetro llamado "paciencia"), el entrenamiento se detiene.

Overfitting



- Como ya vimos, durante el entrenamiento, el modelo es evaluado en cada época utilizando la función de pérdida. Si definimos un conjunto de validación también se calcula este resultado ahí.
- La pérdida en entrenamiento *siempre* va a bajar (generalmente, de hecho si no baja es un problema...).
- Si la pérdida en el conjunto de validación deja de mejorar o comienza a empeorar después de un cierto número de épocas consecutivas (definido por un parámetro llamado "paciencia"), el entrenamiento se detiene.
- De esta forma se frena un sobreajuste (y además nos ahorra tiempo, que en redes neuronales es un extra importante...)

Overfitting

- Hay otras opciones un poco más complejas (un par que ya mencionamos) que también se pueden probar para atacar el sobreajuste. Se añaden como opciones a cada capa.
- **Regularización L1** (Lasso): Añade una penalización basada en la suma de los valores absolutos de los pesos en la función de pérdida, tiende a hacer que algunos pesos se reduzcan a cero, lo que resulta en un modelo más esparso (algunos pesos son eliminados).
- **Regularización L2** (Ridge): Añade una penalización basada en la suma de los cuadrados de los pesos en la función de pérdida, previene que los pesos crezcan demasiado, lo que tiende a hacer el modelo más simple.
- **Dropout**: Durante el entrenamiento, de manera aleatoria, apaga (pone a cero) una proporción de neuronas en cada capa. Busca generar redes más robustas evitando que neuronas individuales se especializen demasiado en los datos.

Parada técnica: Resumen hasta ahora

- **Neuronas Artificiales**: Las unidades básicas de las redes neuronales. Reciben entradas, las ponderan con pesos y pasan el resultado por una función de activación.
- **Funciones de Activación**: Transforman la salida de una neurona, añadiendo no linealidad. Ejemplos comunes incluyen Sigmoid, ReLU y Tanh.
- Redes de Perceptrón Multicapa (**MLP**): Estructuras con capas ocultas que permiten a la red neuronal capturar relaciones no lineales en los datos.
- **Backpropagation**: Algoritmo de entrenamiento que ajusta los pesos mediante el cálculo del error y su retropropagación.
- Definir una red es decidir cuántas capas, de cuantas neuronas y elegir funciones de activación y pérdida acordes.
- Esta lista de DotCSV (link) es ideal para ver esto, pueden ver los primeros cuatro videos.