Argentina programa 4.0

Universidad Nacional de San Martín

Módulo 3
Aprendizaje Automático

Argentina programa 4.0

Universidad Nacional de San Martín

Módulo 3
Aprendizaje Automático

Semana 6. *Redes Neuronales Artificiales*

Contenidos del módulo

ML Clásico

- Árboles de Decisión
- Métodos de Ensemble
  - Bagging / Pasting —> Random Forests
  - Boosting
- Support Vector Machines

Deep Learning

- Redes Neuronales
- Redes Neuronales Convolucionales
- Auto-Encoders / Auto-Encoders Variacionales
- Redes Neuronales Recurrentes (LSTM, otras)
- Extras:
  - Generative Adversarial Networks (GAN)
  - Reinforcement Learning

Contenidos del módulo

ML Clásico

- Árboles de Decisión
- Métodos de Ensemble
  - Bagging / Pasting —> Random Forests
  - Boosting
- Support Vector Machines

Deep Learning

- Redes Neuronales
- Redes Neuronales Convolucionales
- Auto-Encoders / Auto-Encoders Variacionales
- Redes Neuronales Recurrentes (LSTM, otras)
- Extras:
  - Generative Adversarial Networks (GAN)
  - Reinforcement Learning

Contenidos del módulo

- **Redes Neuronales**
  - Introducción: inspiración biológica y algo de historia.
  - Presentación de la neurona o unidad, un modelo estadístico simple en sí mismo.
  - Funciones de activación.
  - Las capas de salida y sus funciones de activación.
  - Arquitectura Feed-forward totalmente conectada (el perceptrón multicapa).
  - Las redes neuronales como aproximadores universales.
  - número de parámetros
  - entrenamiento del modelo; SGD

Deep Learning

- Redes Neuronales
- Redes Neuronales Convolucionales
- Auto-Encoders / Auto-Encoders Variacionales
- Redes Neuronales Recurrentes (LSTM, otras)
- Extras:
  - Generative Adversarial Networks (GAN)
  - Reinforcement Learning

¿Por qué Redes Neuronales Artificiales (ANNs)?

No son los métodos de Aprendizaje Automático más simples

No necesariamente son los mejores ni los más rápidos

Primer sistema verdaderamente de Inteligencia Artificial

Versátiles: no necesitan tener un modelo predefinido, aprende las características

Aún ampliamente utilizadas en las más diversas aplicaciones

Incorporan la mayoría de las características del Aprendizaje Automático

Ejemplo de cómo modelos no lineales simples llevan a comportamientos complejos

Todavía se pueden programar desde cero (+ disponibles en varias bibliotecas)

Puerta de entrada al Aprendizaje Profundo

Hoy: ¿Qué son las Redes Neuronales Artificiales y cómo funcionan?

Un poco de história

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.

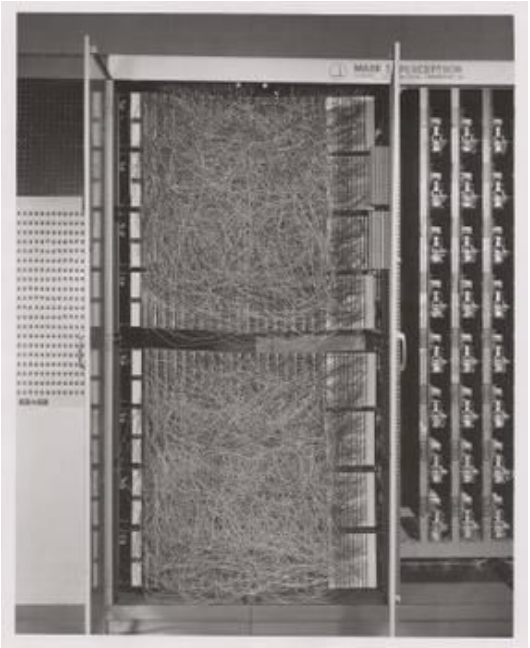
1957. Rosenblatt. Uno de los primeros algoritmos basados en el comportamiento de las neuronas físicas. Consulta "The Organization of Behavior" de Donald Hebb.

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.

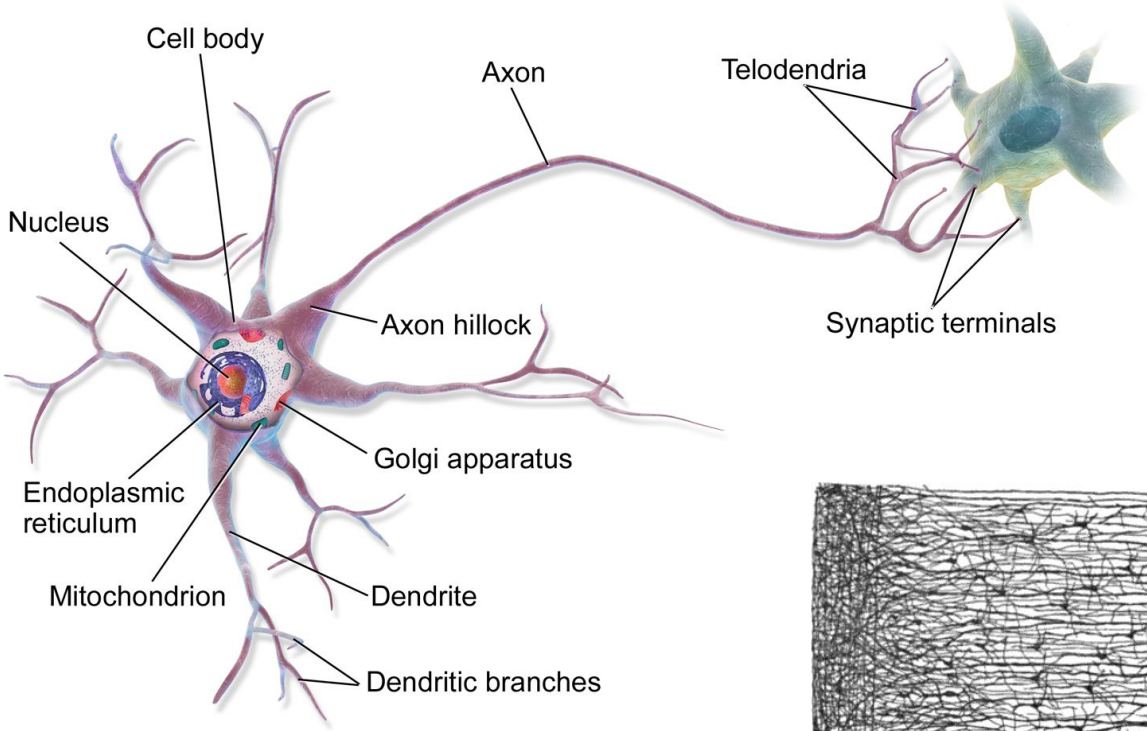
Años 1980. Primer retorno de las Redes Neuronales, pero rápidamente superada por otros algoritmos como las Máquinas de Soporte Vectorial (SVM).

Años 2000. Segundo retorno, impulsada por el aumento en la capacidad computacional y la disponibilidad de datos, junto con técnicas de entrenamiento eficientes.

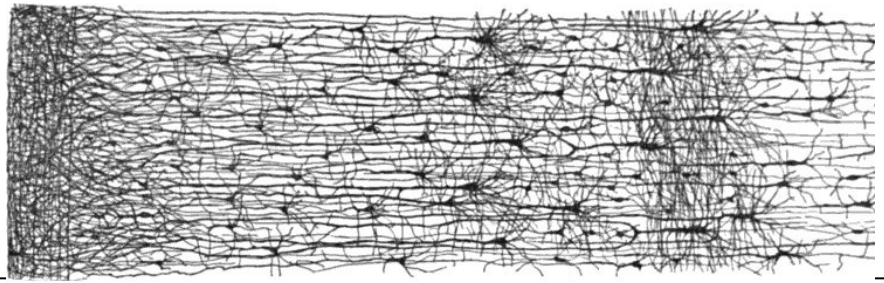


Redes neuronales biologicas

De donde viene la inspiración?



Ramón y Cajal



- Image by Bruce Blaus (Creative Commons 3.0). Reproduced from

Un poco de história

1943
Artificial Neurons and the brain

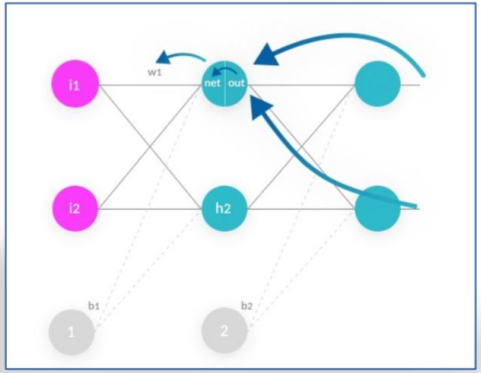
1955
'Artificial Intelligence' is coined

1960
Inference Methods

1975
Backpropagation

1950 the Turing test

1957
Perceptron & first Neural Networks



El perceptron

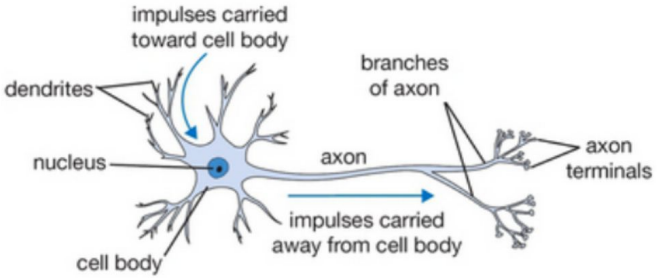
- Bloque de construcción de las Redes Neuronales Artificiales (ANN).
- Comprender: percibir
- "Unidad básica de percepción" en las ANN.
- 1957: Rosenblatt (primero en software, luego en hardware).

- “the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.” [The NYT, 1958]

- "El embrión de una computadora electrónica que [la Armada] espera que pueda caminar, hablar, ver, escribir, reproducirse a sí misma y ser consciente de su existencia". [The New York Times, 1958].

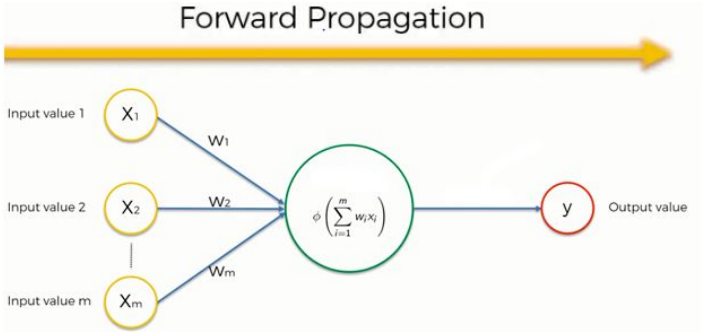
El perceptron

Biological neuron



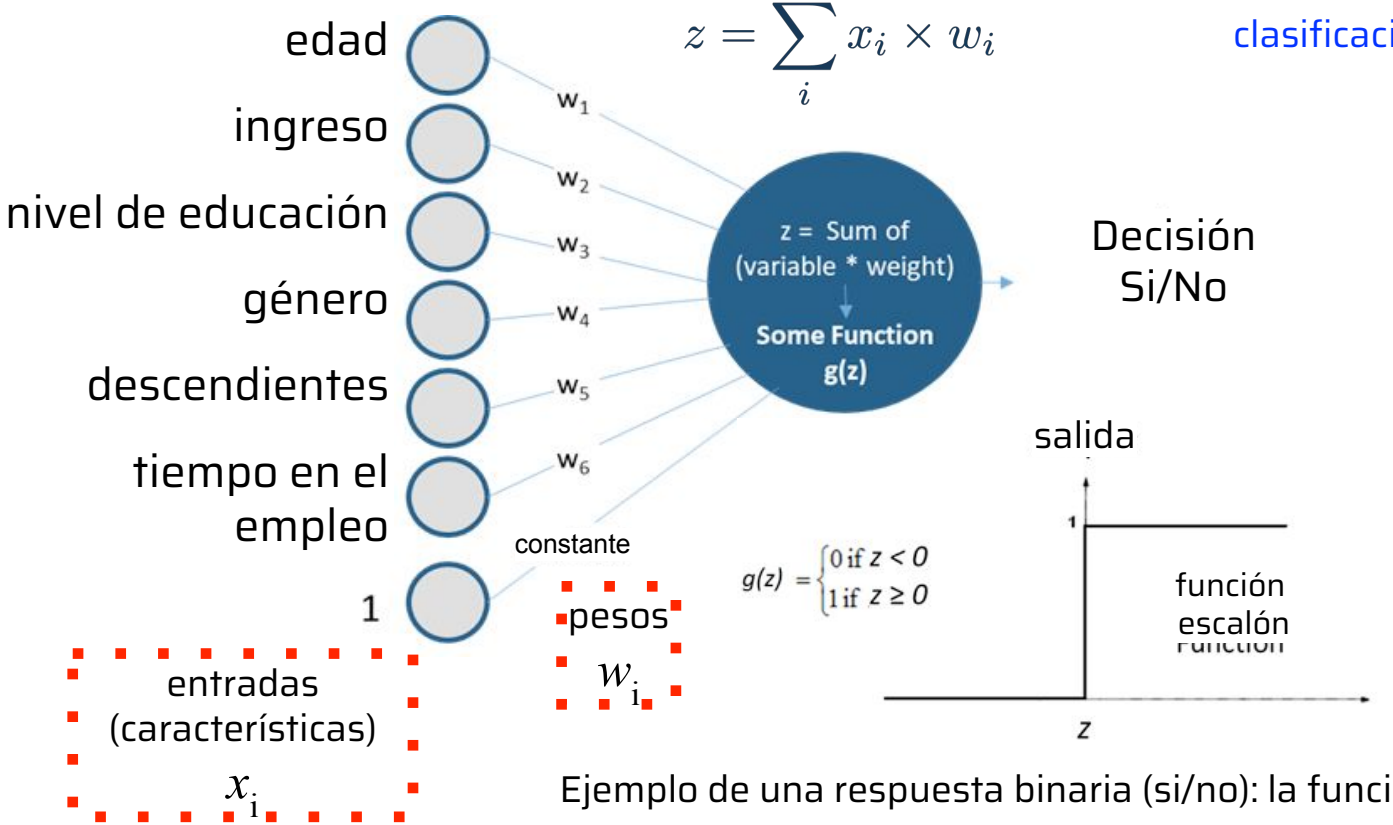
From Stanford cs231n lecture notes

El perceptron



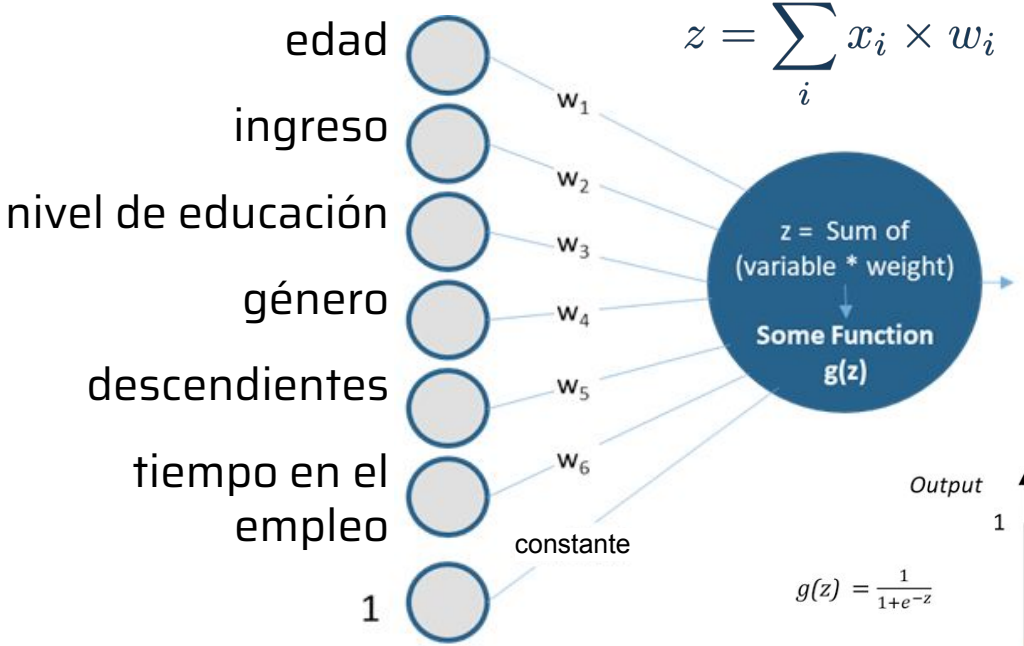
Imitar el cerebro: recibir señales de entrada producir una señal de salida.

El perceptron



Ejemplo de una respuesta binaria (si/no): la función escalón

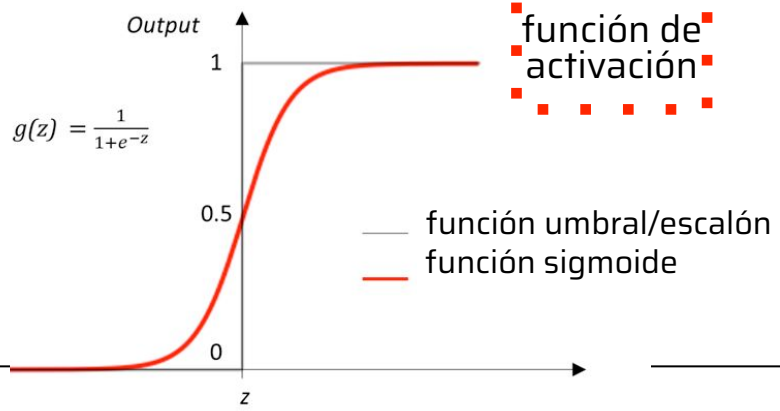
Regresión logística



$$z = \sum_i x_i \times w_i$$

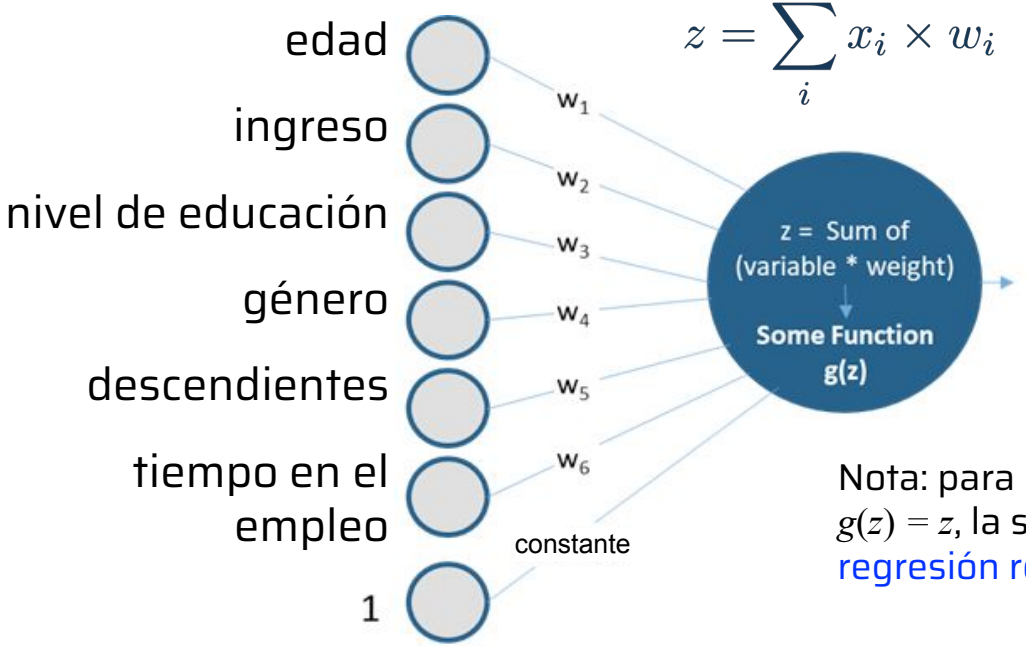
clasificación

Decisión Si/No



Para asignar una "probabilidad"
(y para suavidad/diferenciabilidad):
función sigmoide

Regresión lineal



$$z = \sum_i x_i \times w_i$$

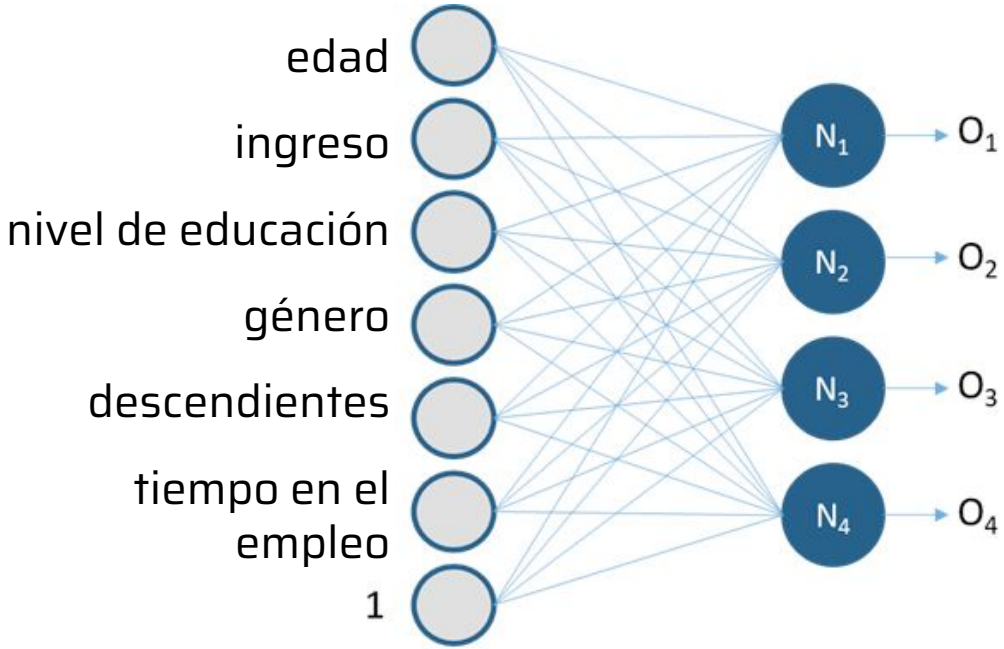
Nota: para una función de activación lineal, $g(z) = z$, la salida es equivalente a la regresión regresión lineal

Permite combinar diferentes entradas y proporciona una salida

El problema: la combinación lineal. Esencialmente, permite trazar líneas (o hiperplanos) que dividen los datos (o ajustar datos por una recta o hiperplanos)

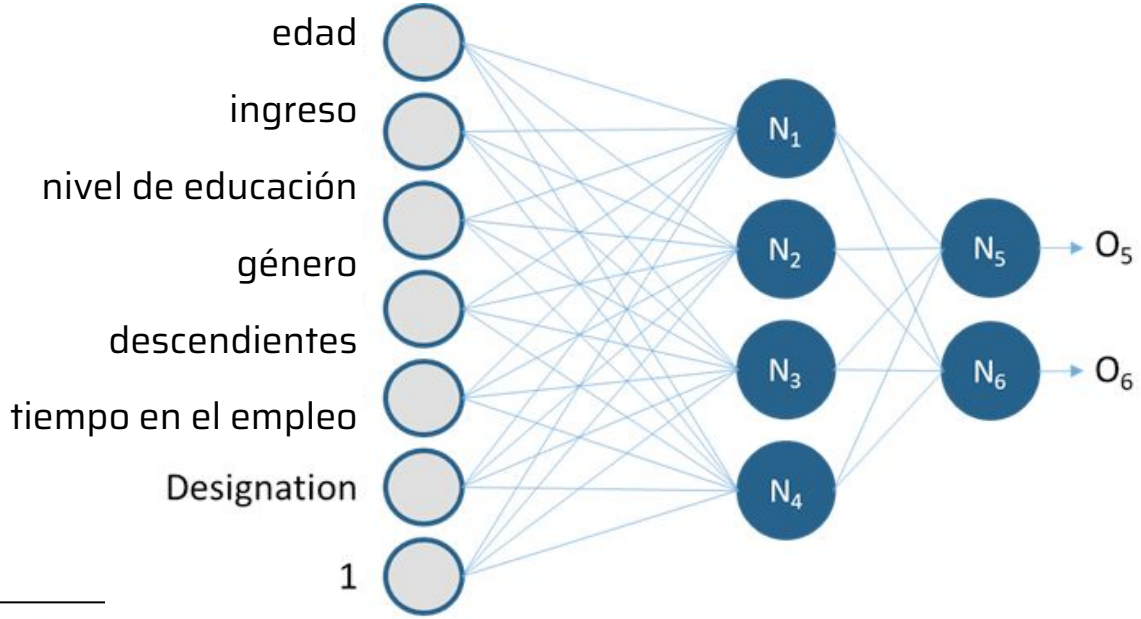
Redes Neuronales Artificiales

- Añadir complejidad: más neuronas (idénticas)
- Cada una con sus propios pesos
- Diferentes salidas para cada una



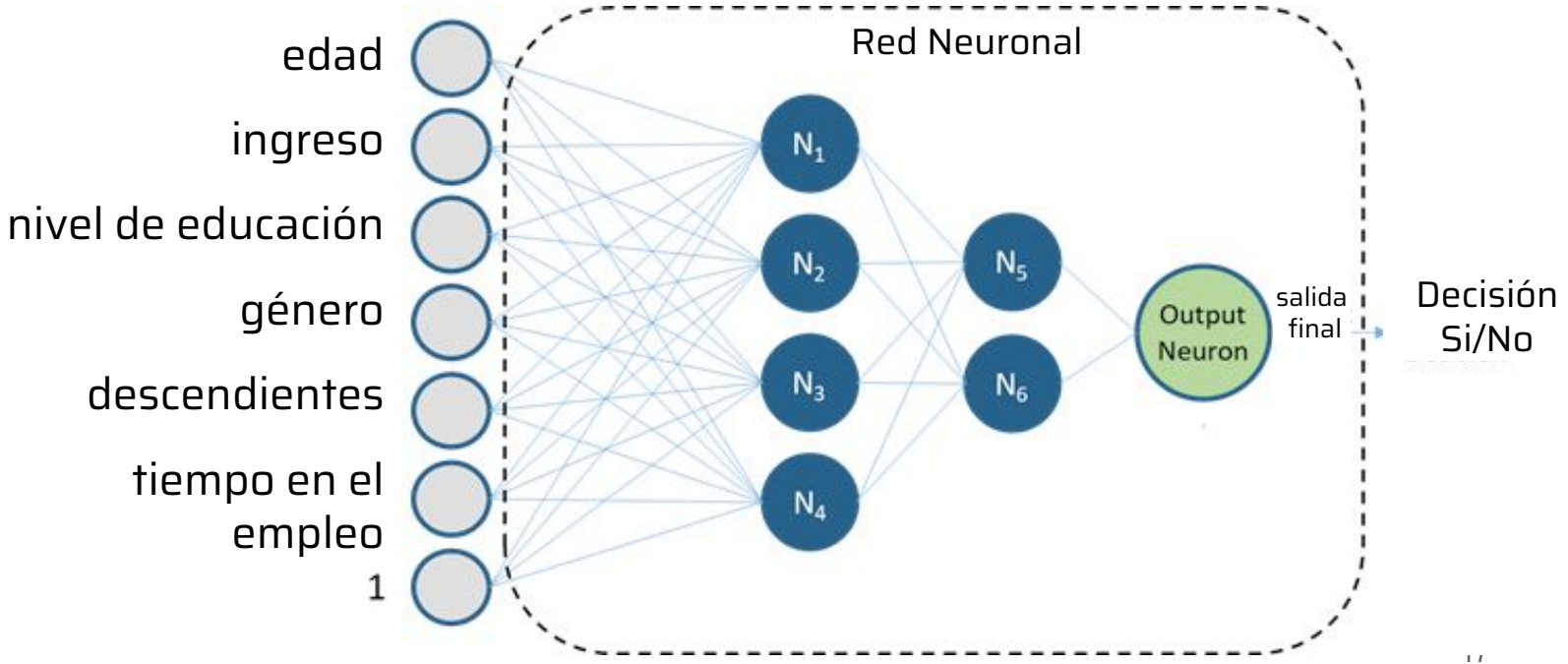
Redes Neuronales Artificiales

- Una capa: hiperplanos que dividen los puntos de datos
- Redes neuronales: imitan al cerebro y conectan las "neuronas"
- Una capa: lineal, más capas: ¡no lineal!

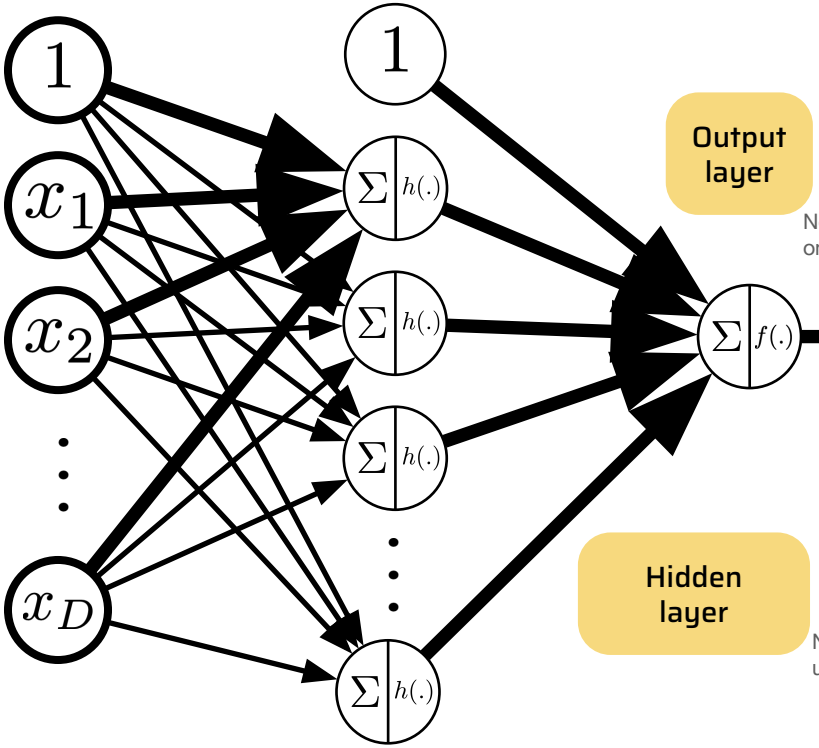


Redes Neuronales Artificiales

- Combinar las salidas en una sola neurona para generar la salida final
- ¡Hemos construido una red neuronal!



Funciones lineales combinadas con activación no lineal



Input layer

Hidden layer

Not always M units

Output layer

Not necessarily one unit

Feed-forward fully-connected

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

where

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

Conexión con otros métodos

Modelos lineales (para regresión o clasificación)

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) \quad .$$

Regresión

$$y(\mathbf{x}, \mathbf{w}) = f\left(w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) \right)$$

Clasificación

Hasta ahora, las funciones base ϕ_j se eligiann de antemano. Tienen buenas propiedades analíticas y computacionales, pero surgen limitaciones con entradas de alta dimensión.

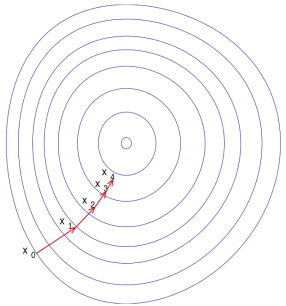
Las Máquinas de Soporte Vectorial (SVM) resuelven algunas de estas limitaciones mediante el truco del kernel; el número de funciones base se vuelve muy grande (potencialmente infinito).

Los métodos de *ensemble* combinan modelos lineales simples para aumentar la capacidad predictiva a expensas de la interpretabilidad.

¿Como entrenar una RNA?

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

Gradient descent

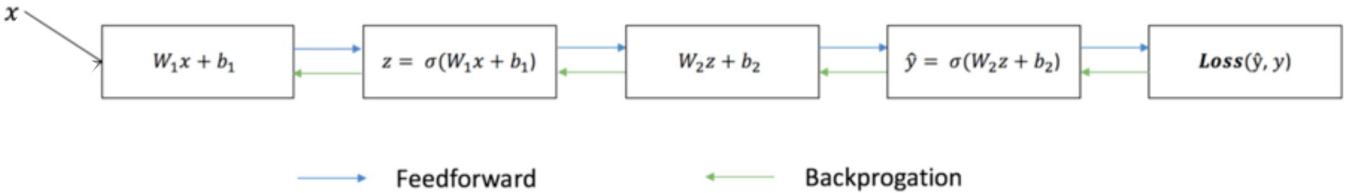


- El entrenamiento es equivalente a minimizar la función de pérdida.
- Debido al gran número de parámetros en el modelo, a menudo se utiliza regularización en los pesos de las capas ocultas (L2, L1 o ElasticNet).
- Función compleja y no convexa.
- Muchos mínimos locales, puntos de silla, etc.
- El gradiente se puede calcular eficientemente utilizando la retropropagación del error (también conocida como regla de la cadena).
- A menudo se utilizan métodos de gradiente (ver ejemplo de Descenso de Gradiente o SGD).
- La complejidad de la función requiere trucos adicionales (por ejemplo, una tasa de aprendizaje programada) y algoritmos específicos.

¿Como entrenar una RNA? Retropropagación

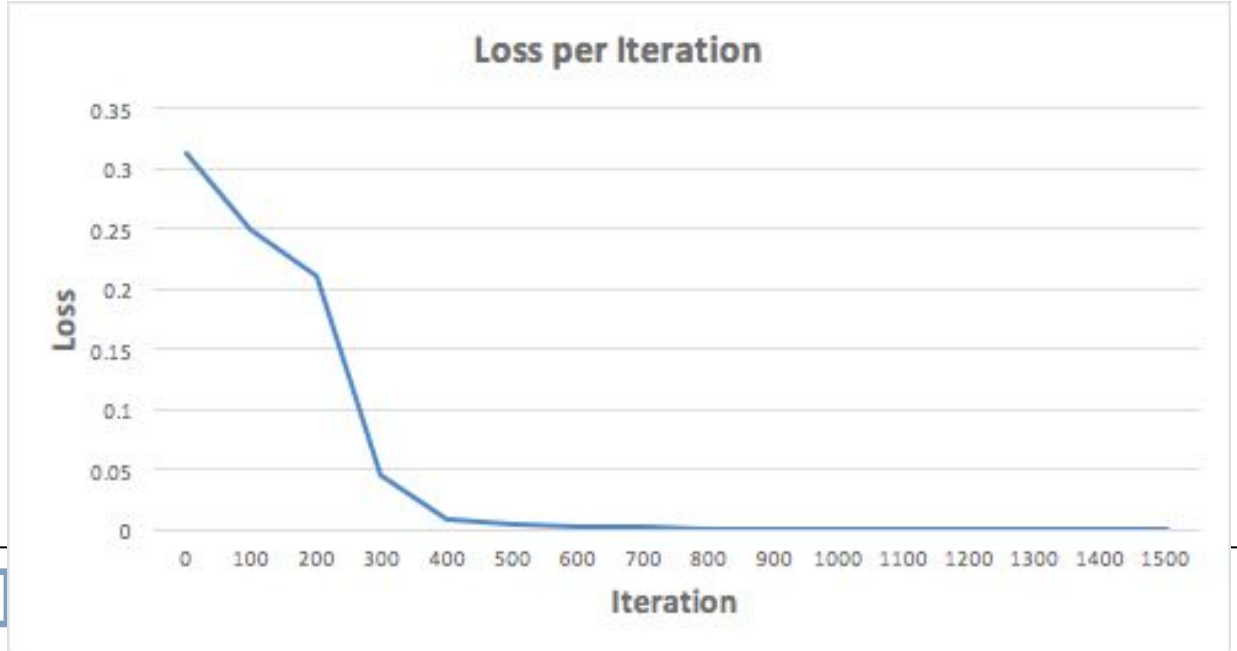
- Entrenar/aprender es obtener los pesos w_j : los que minimizan la función de pérdida
- Necesita las direcciones para mover w_j para disminuir la pérdida (recordar las derivadas, etc...)
- Técnicamente es muy difícil minimizar la función de pérdida: muchos parámetros, altamente no lineal
- Una función de activación suave ("diferenciable") permite el uso directo del descenso del gradiente y otros algoritmos de optimización para el ajuste de los pesos

- Elegante truco matemático: la retropropagación (*backpropagation*, 1986)
- Se parte de la pérdida, se calculan los cambios (derivadas) de los nodos de cada capa "hacia la izquierda" hasta la entrada. Regla de la cadena, producto de matrices
- Reajustar los pesos y calcula de nuevo la pérdida
- Y luego hacia atrás de nuevo, hasta la convergencia.



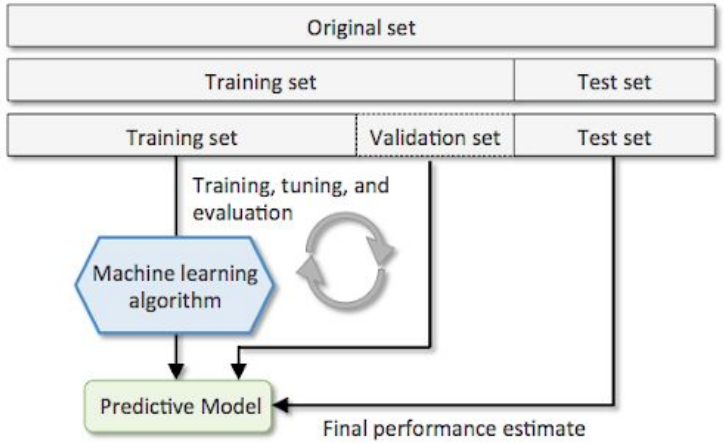
¿Como entrenar una RNA?

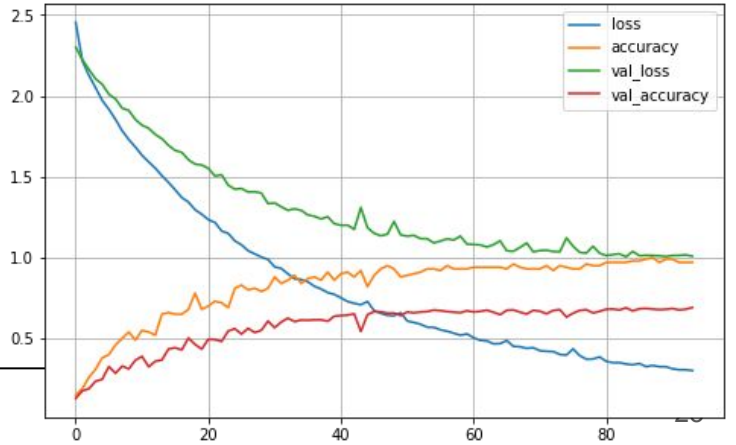
- Después de cada ciclo (iteración) se actualizan los pesos
- La tasa de aprendizaje determina el "tamaño del paso" (sobrepasar x atascado en un mínimo local)
- Después de muchas iteraciones, el proceso debería converger
- ¡Su red neuronal está finalmente entrenada!



¿Cuando parar? evitar el sobreajuste

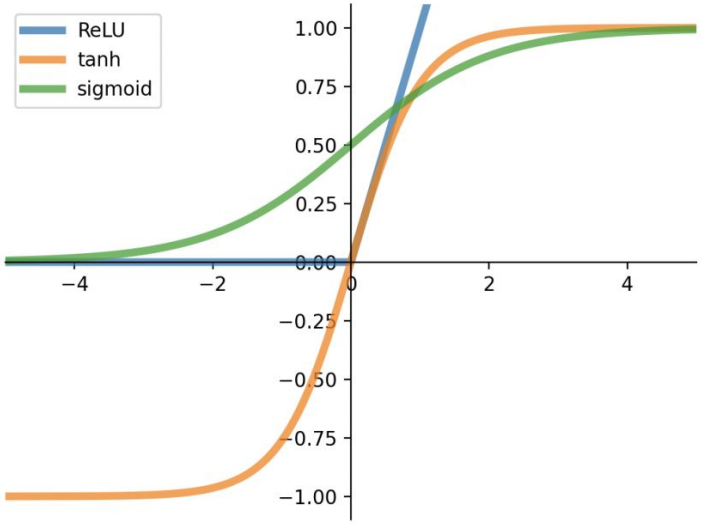
- *Holdout*: dividir los datos de entrenamiento en una muestra de entrenamiento y otra de validación (por ejemplo, 80% + 20%)
- Semejante a validación cruzada, pero se hace una sola vez
  Se puede usar **durante** el entrenamiento.
  Ejemplo: EarlyStopping (un tipo de *callback*)
- La pérdida en la muestra de validación debe ser comparable a la del entrenamiento, de lo contrario estamos sobreajustando
- Otras técnicas de regularización: *Lasso*, *Ridge*, *Dropout*





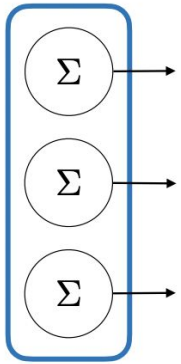
- De ser posible: hacer validación cruzada:
  Repetir la división varias veces, seleccionando aleatoriamente las submuestras: robustez de la RNA y estimación de las barras de error

Tipos y importancia de las funciones activación



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

$$s(z_i) = \frac{\exp(z_i)}{\sum_{k=1}^{K} \exp(z_j)}$$



Capas internas: ReLU (o SeLU, GeLU)
(problema de los gradientes evanescentes)

Construyendo una Red Neuronal Artificial en Keras

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
# Multilayer Perceptron model
model = Sequential()
model.add(Dense(input_dim=6, activation="relu" ...
model.add(Dense(activation="re... ...el initializer="normal"))
model.add(Dens... ...
model.compile optimizer=SGD lr=0.01 , loss='mean_squared_error' metrics=['accuracy'])
model.summary()
```

RNA con la misma arquitectura que el diagrama de ejemplo mostrado anteriormente

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 4)                 28
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 10
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 3
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
```

Además de los pesos w_j, cada neurona tiene también un parámetro de sesgo añadido al producto (por lo tanto, un parámetro adicional por neurona)

Resumen

RNA son versátiles: no importa cuál sea la estructura de los datos o el modelo subyacente

Red neuronal definida por la arquitectura (número de capas, número de neuronas en cada capa, conectividad, etc., también llamados hiperparámetros), por las funciones de activación y por los pesos wj

Dada una arquitectura, una función pérdida, un optimizador, una regularización, encontrar los pesos mediante retropropagación (definir épocas y tasa)

Posibilidad/necesidad de ajustar los hiperparámetros para mejorar el rendimiento

¡¡Ya estás listo para utilizar tu RNA para predecir salidas!!

Como siempre hay dos etapas:

Entrenamiento: el par entrada-salida es fijo, mientras que los pesos varían, de forma a minimizar la función pérdida

Evaluación: los pesos son fijos, mientras que las entradas varían (y la salida objetivo puede ser desconocida)

Comentarios finales

Para entender las RNA se usan todos los conceptos que hemos visto hasta ahora del aprendizaje automático supervisado

No son el método más simple de ML, ni necesariamente el mejor, pero son muy versátiles

Precio a pagar por la versatilidad (muchos parámetros):

Hambrienta de datos (ok, big data)

Intensiva en términos de computación (ok, GPUs)

Hoy: red neuronal artificial de conexión directa (feedforward), totalmente conectada

Camino hacia el aprendizaje profundo y redes neuronales más especializadas

DNN: todas las capas totalmente conectadas

CNN: algunas capas convolucionales

RNN: redes neuronales recurrentes

Comentarios finales

Las redes neuronales son métodos bastante "sencillos", pero potentes

Los avances técnicos han permitido el aprendizaje profundo (por ejemplo, la retropropagación, la activación ReLU, el uso de GPU, etc.)

Hoy en día son triviales de codificar mediante bibliotecas (ej. Keras, PyTorch)

Sin embargo, hay que tener en cuenta su complejidad: arquitectura, función de activación, función de pérdida, tamaño del lote/velocidad de aprendizaje/épocas, optimizadores, etc.

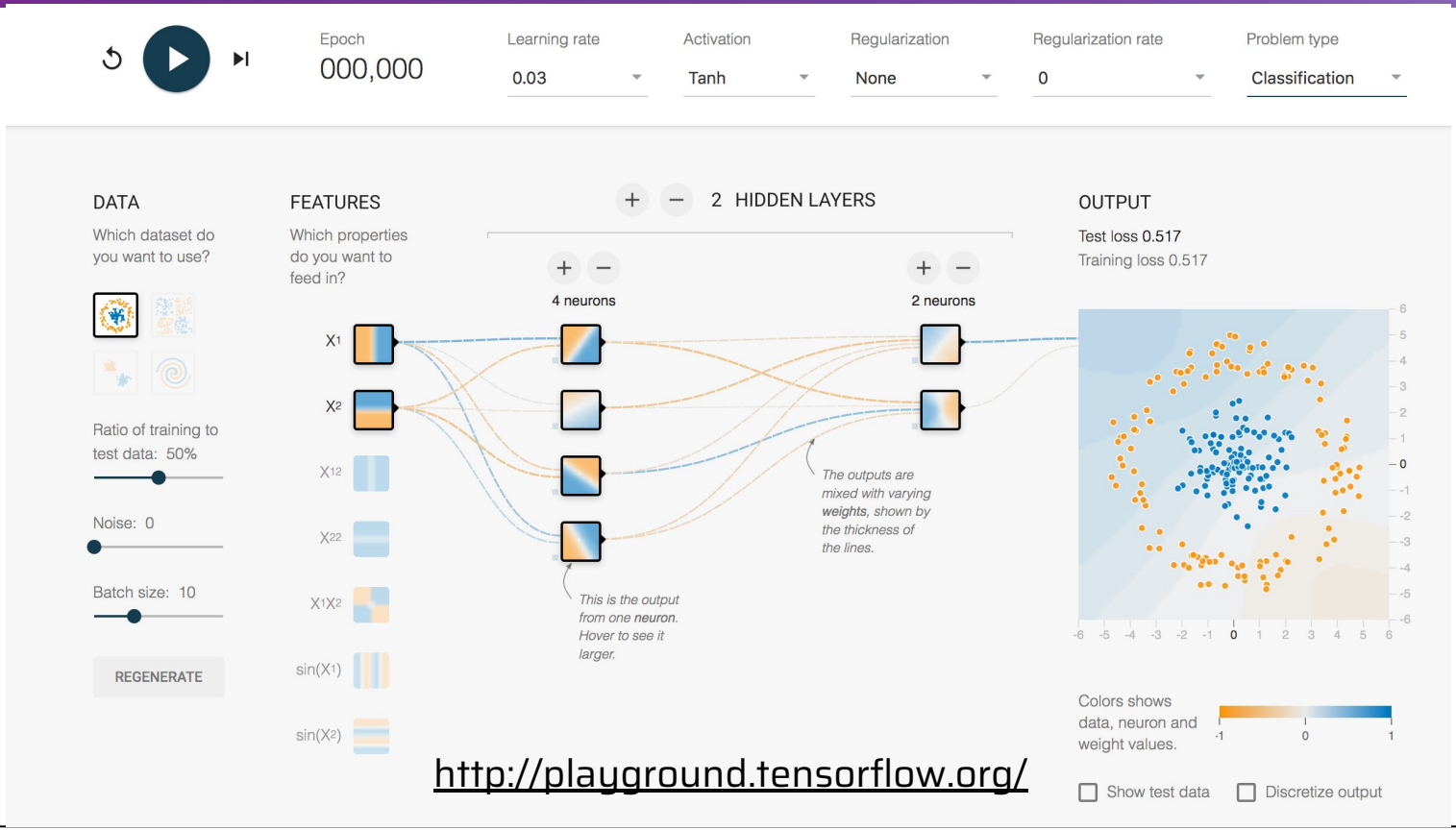
Hoy vamos a ver algunos ejemplos de la construcción y el entrenamiento de una RNA y seguiremos con más detalles en la próxima clase

Para divertirse y aprender sobre redes neuronales sin código

http://playground.tensorflow.org/

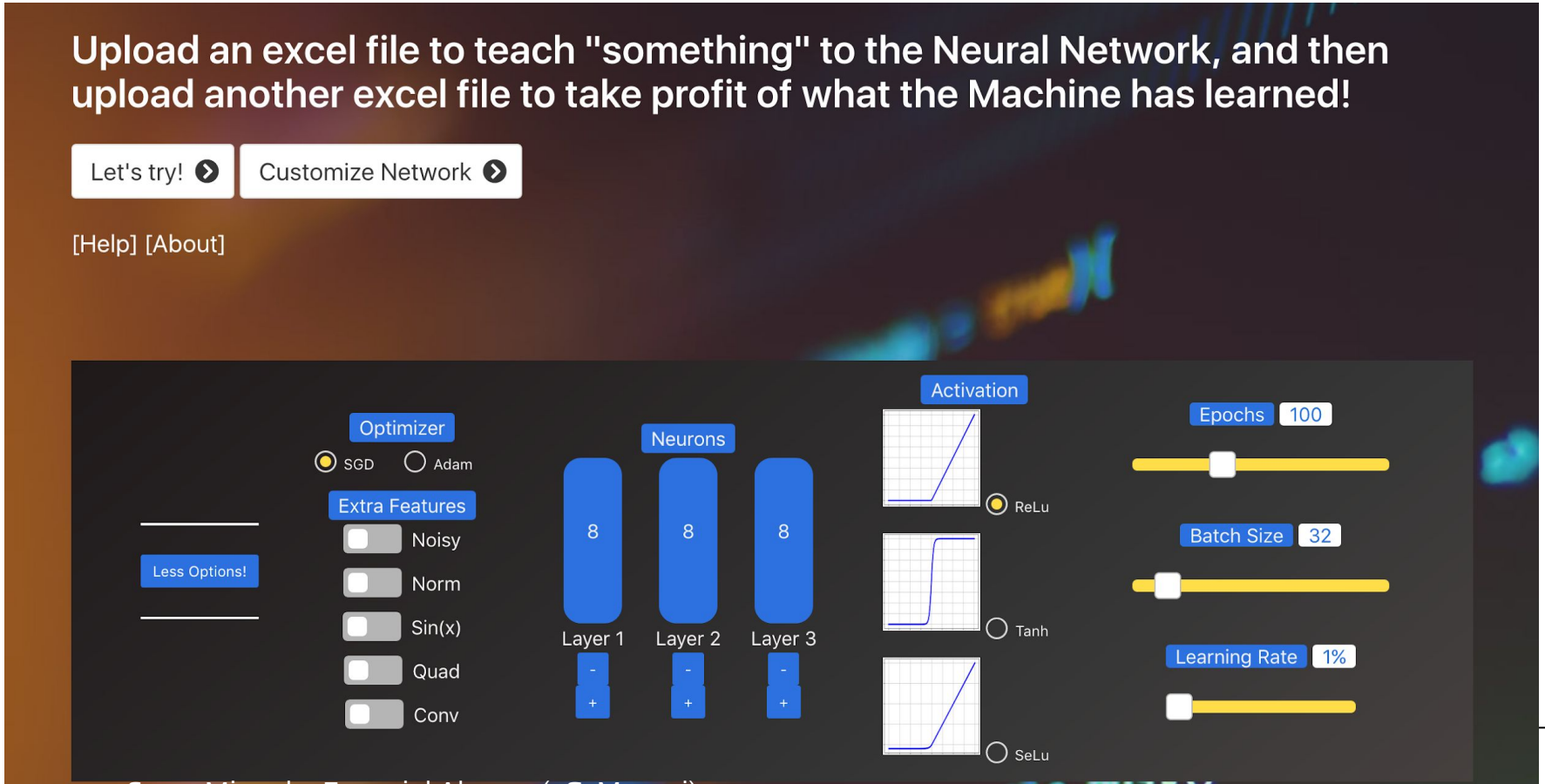
http://ai-friendly.com/

Para divertirse y aprender sobre redes neuronales sin código



http://playground.tensorflow.org/

Para divertirse y aprender sobre redes neuronales sin código



por Cesar Miquel y Ezequiel Alvarez (+G. Mazzei)

Vamos al notebook!

`Notebook_semana_6_RedesNeuronales_1.ipynb`