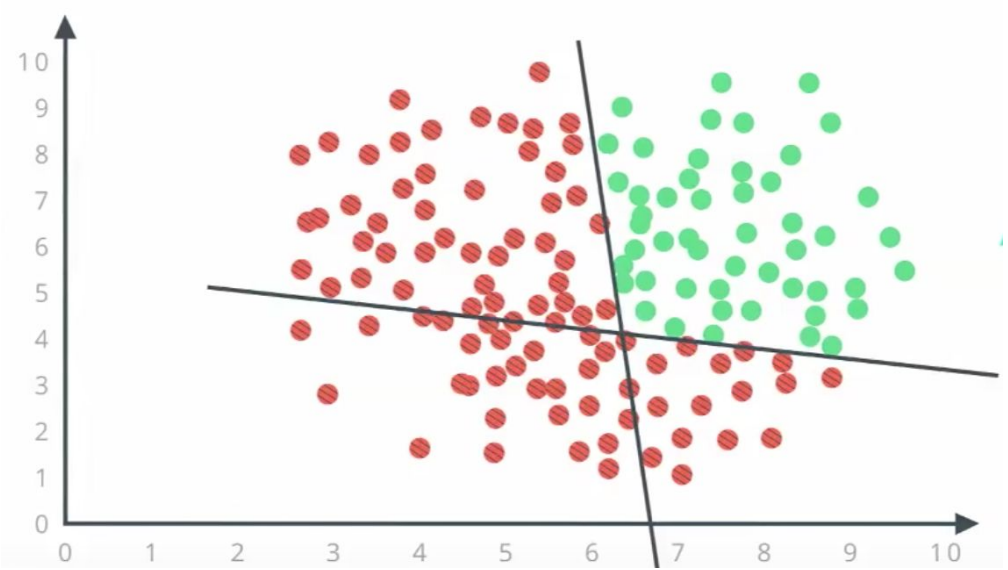
Deep Learning y Redes Neuronales

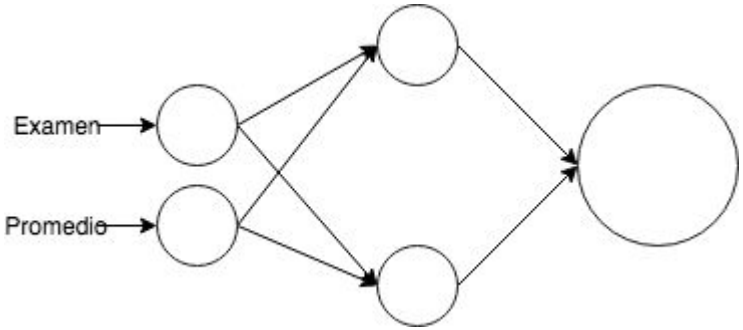
Redes neuronales

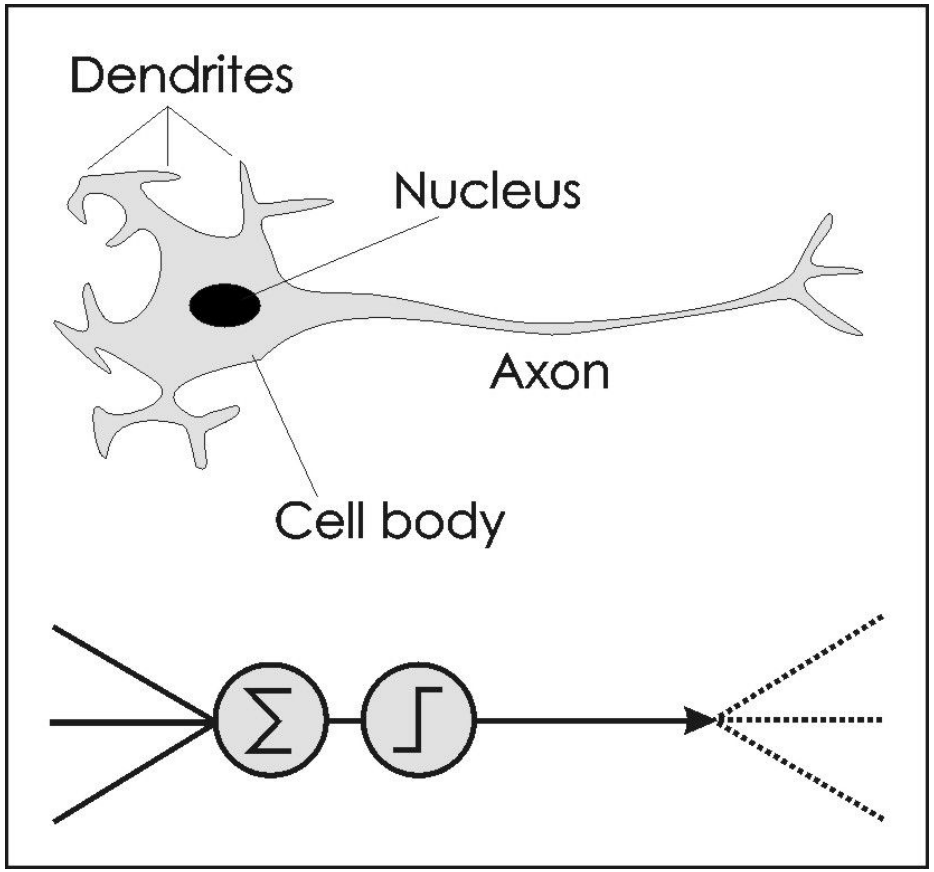
- ¿El punto está arriba de la primera línea?
- ¿El punto está a la derecha de la otra línea?
- ¿La respuesta a las otras dos preguntas fue sí?



Redes neuronales

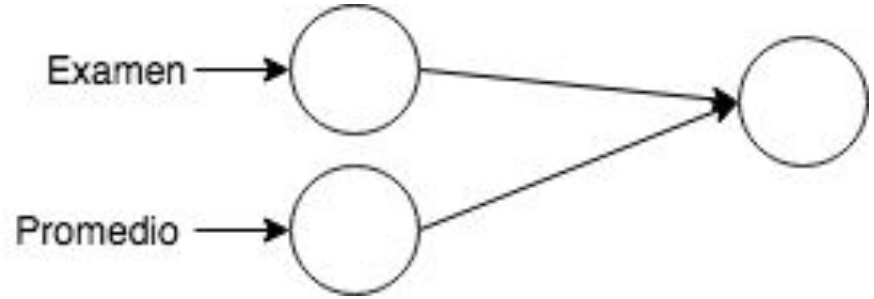
- ¿El punto está arriba de la primera línea?
- ¿El punto está a la derecha de la otra línea?
- ¿La respuesta a las otras dos preguntas fue sí?




Dendrites
Nucleus
Axon
Cell body
Σ

Perceptrón

Input o feature



Features

- ¿Todos son igual de importantes?
- Podemos asignar un peso (W) a cada entrada/feature (X).

V = W1X1 * W2X2

$$\sum_{i=1}^{m} w_i x_i$$

- X1 calificación del colegio.
- W1 peso de la calificación del colegio.
- X2 calificación del examen.
- W2 peso de la calificación del examen.

Ejemplo

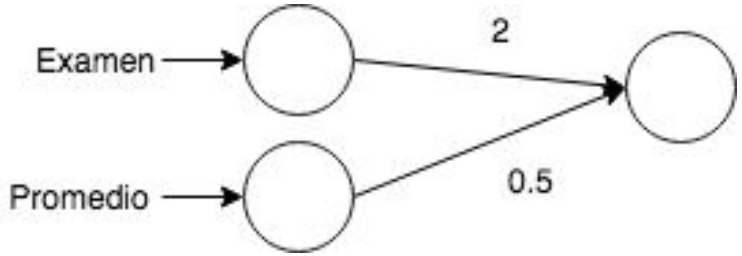
- Si queremos que el examen sea más importante, podemos asignar pesos como los siguientes

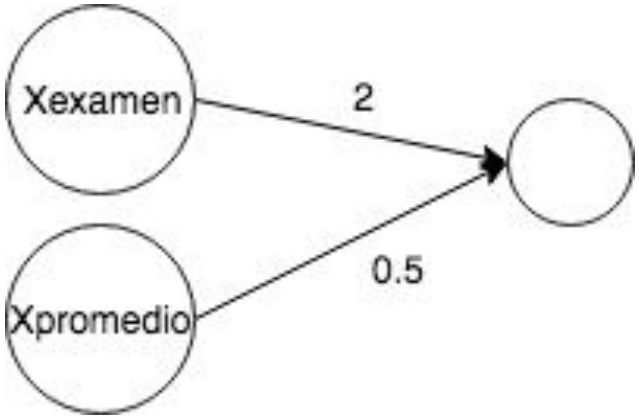
$V = 0.5X1 + 2X2$

	Promedio Colegio	Examen Admisión	Valor
A	10	9	23
B	8	9	22
C	6	6	18

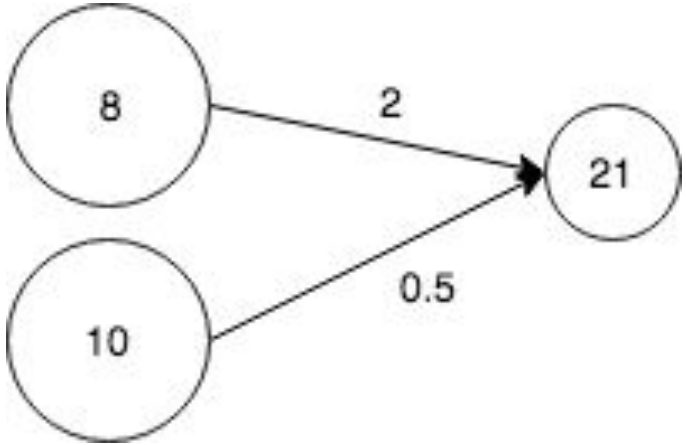
- Nosotros decidimos a partir de qué valor pasará el alumno (20?) - **bias**

Weights (pesos)





Ejemplo



Perceptrón

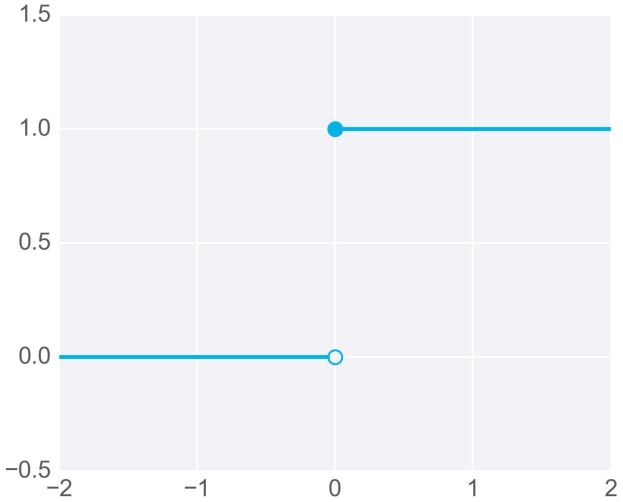
- Se asigna un bias.
- Si el valor de la neurona es más alto que el -bias, la neurona se enciende.

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

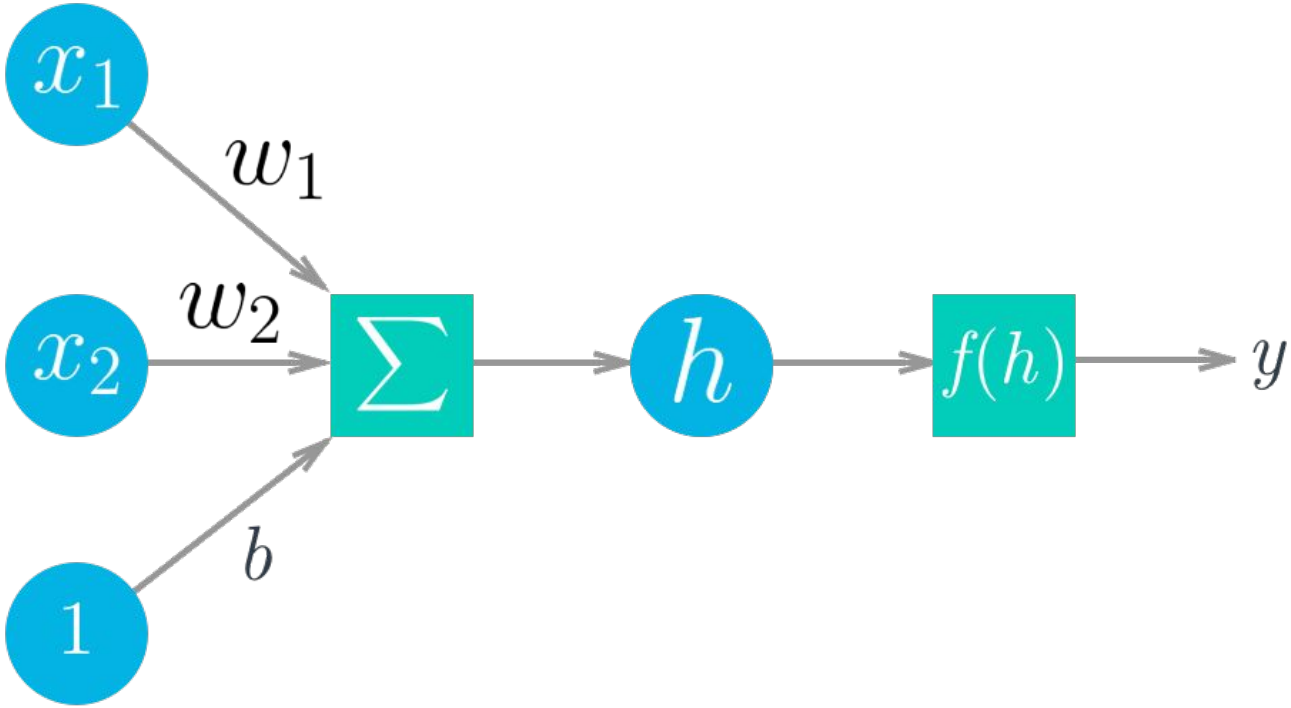
- En nuestro caso, b sería -20

Perceptrón

- Aprende a través de ejemplos.
- Pesos (w) y bias (b) se van ajustando viendo los ejemplos
- Su **función de activación** es una función escalón

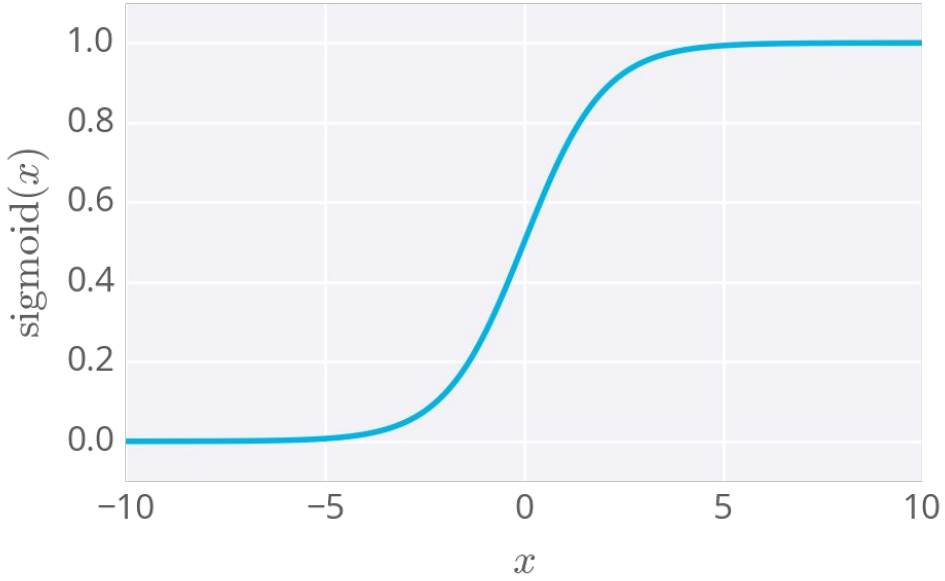


$$f(h) = \begin{cases} 0 & \text{if } h < 0 \\ 1 & \text{if } h \geq 0 \end{cases}$$


x_1
w_1
w_2
x_2
$\sum$
h
$f(h)$
y
b
1

Función de activación

- Aquí usamos una función de activación sencilla, pero realmente puede **cualquier** función (derivable).



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

Su output puede ser cualquier número entre 0 y 1.

Repasando

- Nosotros vamos a definir cuántas neuronas hay y cómo están conectadas.
- Mientras más capas de neuronas, la red puede comprender información más compleja (Deep Learning).
- Las conexiones tienen un peso.
- Las neuronas ocultas tienen un bias.
- Los pesos y bias se asignan aleatoriamente y se van ajustando con entrenamiento (ejemplos).
- Nosotros definimos cómo las neuronas procesan la información (funciones de activación)
- **Una red neuronal puede aprender cualquier cosa con una buena arquitectura, suficiente información, y suficiente entrenamiento.**

90's: We will use AI to help diagnose and cure disease

2018:

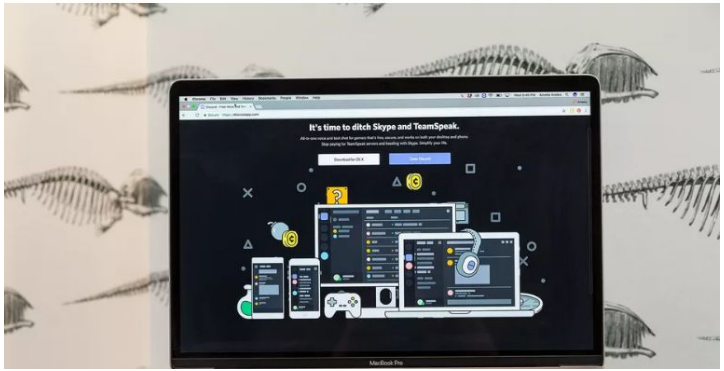
WEB \ TECH \ ARTIFICIAL INTELLIGENCE

Discord shut down a chat group that shared fake celebrity porn edited with artificial intelligence

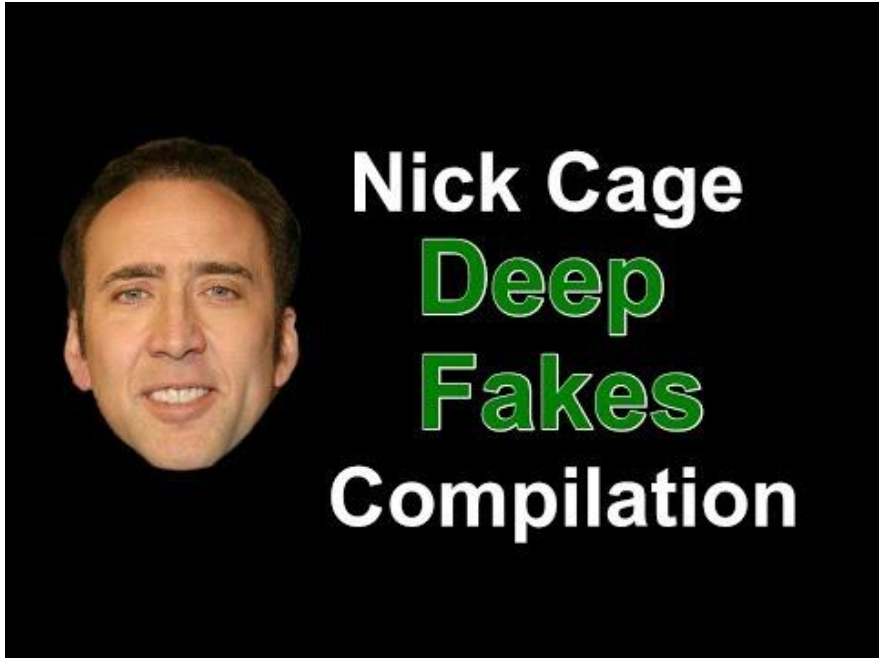
13

The group contained channels dedicated to creating and sharing the videos

By Andrew Liptak | @AndrewLiptak | Jan 28, 2018, 5:21pm EST

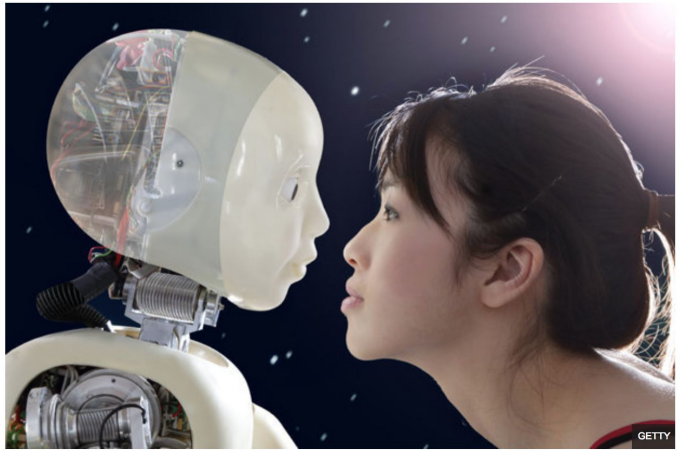


DeepFakes



Google supercomputer creates its own 'AI CHILD'

A GOOGLE supercomputer has created its own "AI child" that is capable of outperforming humans.



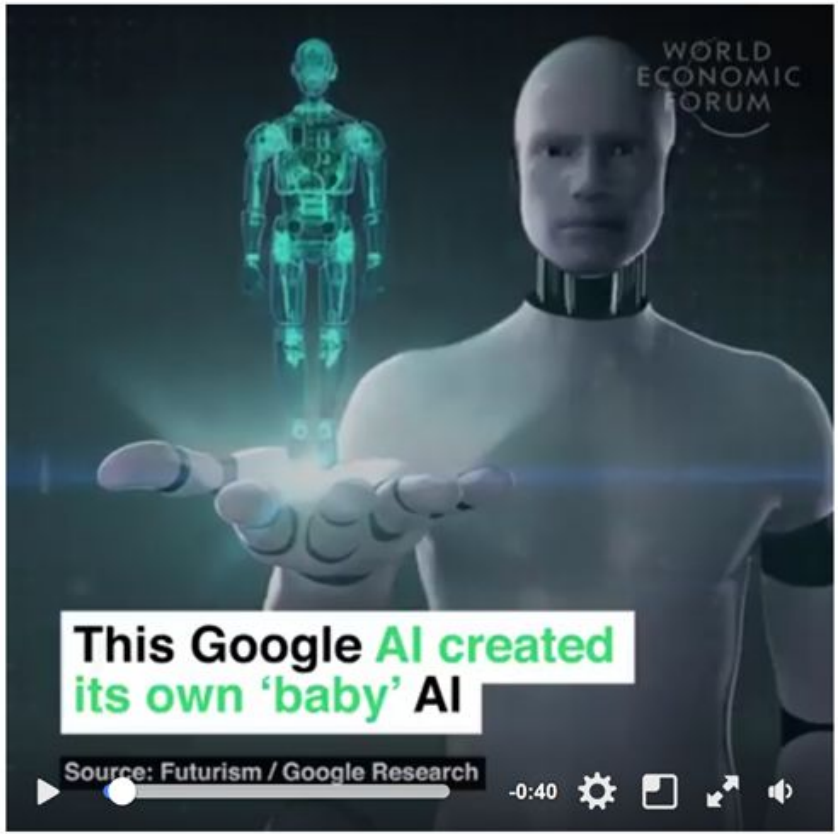
BREAKTHROUGH: Google has created its own AI Child

Google Intern :

grid_search.py

```python
from keras.layers import *
from keras.models import *
from .data import load_data

x, y, x_test, y_test = load_data()

def get_model(num_layers):
    model = Sequential()
    for _ in range(num_layers):
        model.add(Dense(100, activation='sigmoid'))
    model.compile(loss='mse', optimizer='sgd')
    return model

best_model = None
best_loss = None

for i in range(1, 10):
    model = get_model(i)
    model.fit(x, y)
    loss = model.evaluate(x_test, y_test)
    if best_loss is None or loss < best_loss:
        best_loss = loss
        best_model = model
```

Media :



Hyperparameters

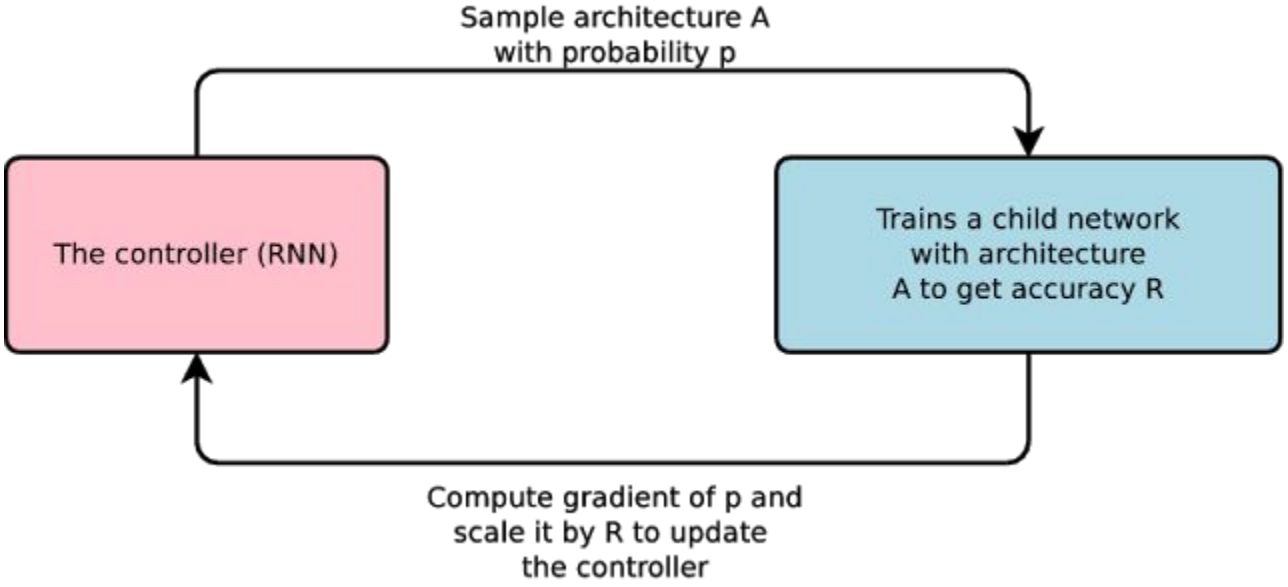
- Modelo - fórmula matemática con un número de parámetros que se **aprende** de la información.
  - **Model training**
- ¿Qué es lo que se ajusta con la información?
- Hay parámetros que no se aprenden directamente de la información.
  - Cómo debe aprender el modelo
  - Estos son **hiperparámetros**
  - Se eligen según los que funcionen mejor en el **training set**
  - Ejemplo: función de activación

Hyperparameters

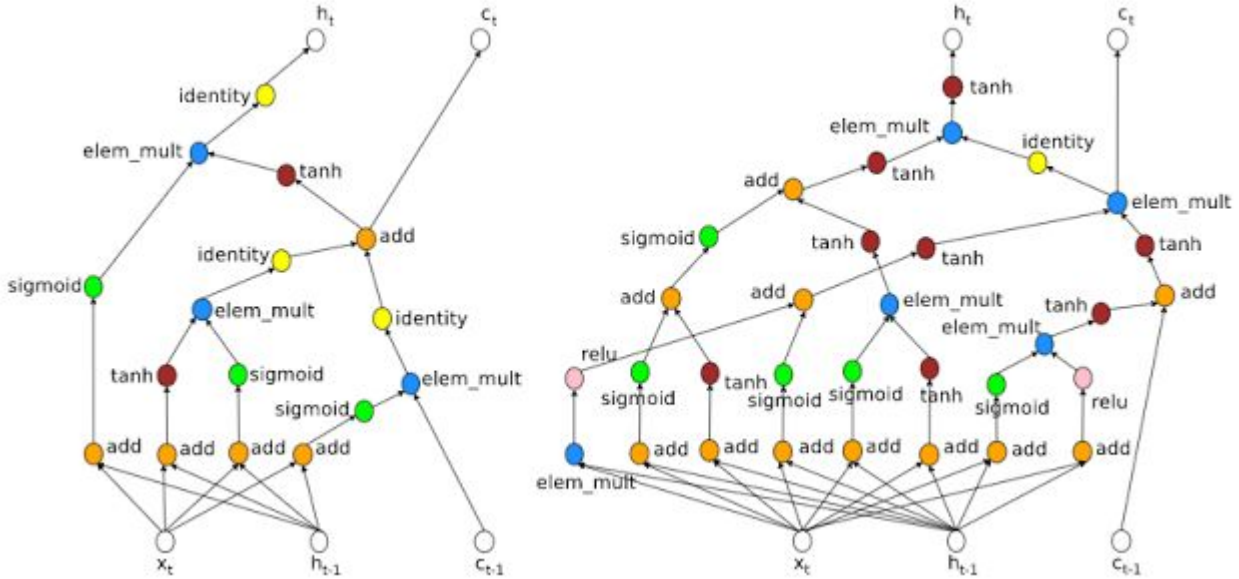
- Optimizador (Gradient Descent)
- Función de pérdida
- Número de hidden layers
- Número de neuronas en capas ocultas
- Epochs
- Learning rate
- Activation function (Sigmoid, ReLU, ...)

AutoML

- Un modelo de 10 capas tiene 10^{10} redes candidatas.
- Diseñar la arquitectura manualmente puede ser doloroso.



AutoML


h_t
c_t
identity
elem_mult
tanh
add
identity
sigmoid
elem_mult
identity
tanh
sigmoid
elem_mult
sigmoid
add
add
add
add
x_t
h_t-1
c_t-1
h_t
c_t
tanh
elem_mult
identity
add
tanh
elem_mult
sigmoid
tanh
tanh
tanh
add
add
elem_mult
tanh
elem_mult
add
relu
sigmoid
tanh
sigmoid
sigmoid
tanh
sigmoid
relu
elem_mult
add
add
add
add
add
add
add
add
x_t
h_t-1
c_t-1

Problema

Inputs			Output
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

```
X = np.array([[0,0,1], [0,1,1], [1,0,1], [1,1,1]])
y = np.array([[0], [1], [1], [0]])
np.random.seed(1)
```

Diseño de arquitectura

- ¿Cuántas neuronas de entrada hay?

Diseño de arquitectura

- ¿Cuántas neuronas de entrada hay?

- ¿Cuántas neuronas de salida?

Diseño de arquitectura

- ¿Cuántas neuronas de entrada hay?

- ¿Cuántas neuronas de salida?

- Tendremos una hidden layer

Inicializar los pesos

- Aleatorios
- Promedio de 0

```
# Se inicializa el peso con promedio de 0
weights0 = 2 * np.random.random((3,4)) - 1
weights1 = 2 * np.random.random((4,1)) - 1
```

Definición de Activación

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

```
def sigmoid(x):
    return 1/(1+np.exp(-x))

def sigmoid_deriv(x):
    return x*(1-x)
```

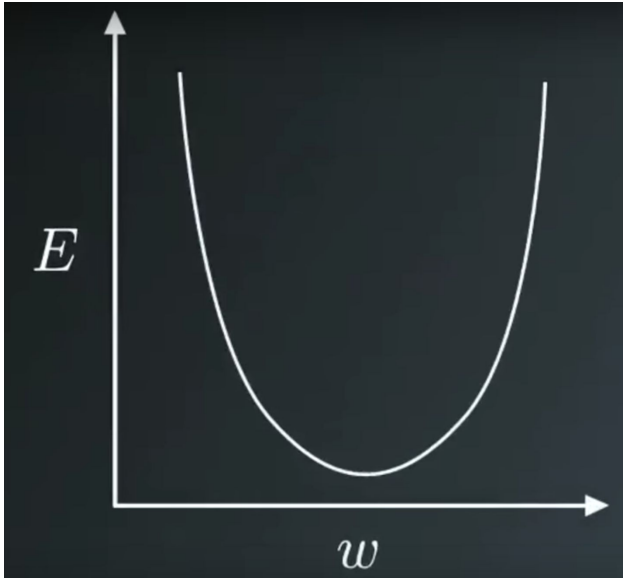
Para cada neurona:

activación (input * weight matrix)

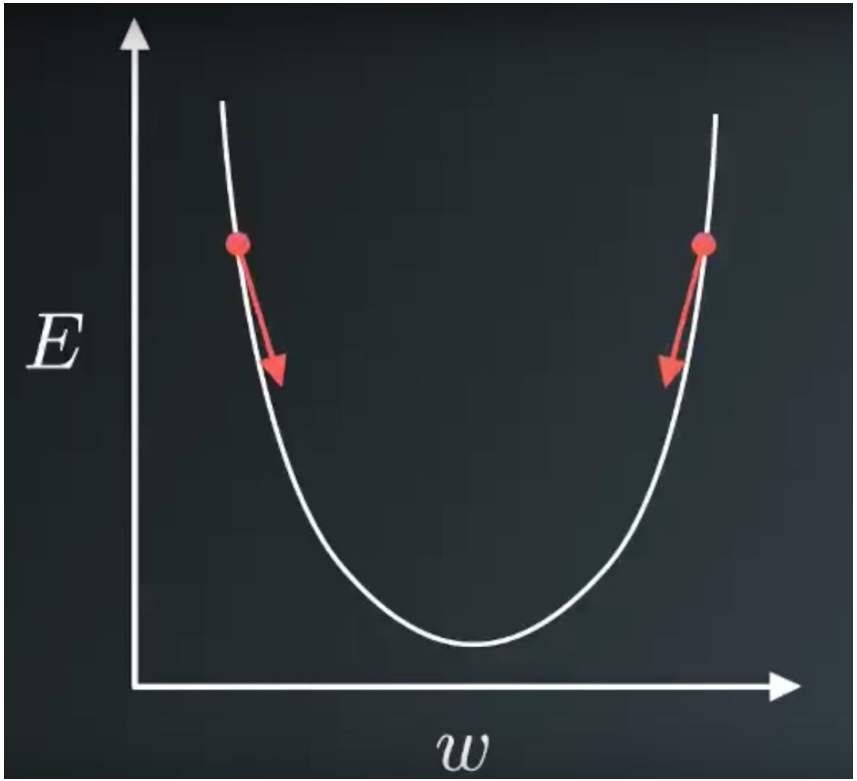
FeedForward o Forward Propagation

```
# Feed forward
l0 = X
l1 = sigmoid(np.dot(l0, weights0))
l2 = sigmoid(np.dot(l1, weights1))
```

Calcular el error



Gradient Descent



$$\Delta w = -gradient$$

Backpropagation

- Calcular la derivada parcial de la función de error respecto **a cada peso** individual de la red neuronal.
- Permite calcular el error en cada neurona

```
l2_error = y - l2
l2_delta = l2_error*sigmoid_deriv(l2)

l1_error = l2_delta.dot(weights1.T)
l1_delta = l1_error * sigmoid_deriv(l1)

weights1 += l1.T.dot(l2_delta)
weights0 += l0.T.dot(l1_delta)
```

Actualizamos peso y bias

$$w_k \to w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l}.$$

Algoritmo de Redes Neuronales

1. **Forward Propagation**: Calcular suma de entrada y activación de cada neurona aplicando multiplicaciones de matrices iterativamente.
2. Calcular la **función de error/pérdida** en la última capa L. Depende de la función de pérdida y el training sample específico.
3. **Backpropagation**: Calcula el error en cada neurona de cada capa. Se utiliza multiplicación de matrices con derivadas para hacer esto.
4. Calcula la derivada del costo respecto a los pesos y la derivada del costo respecto a los biases. Esto es parte de **Gradient Descent**.
5. Actualiza los pesos y bias.

Otras funciones de pérdida

- Mean Squared Error
  - Bueno para regresión (especialmente linear)
- Mean Squared Logarithmic Error
- Mean Absolute Error
- Mean Absolute Percentage Error
- Kullback Divergence
- Cross Entropy
  - Bueno para clasificación

https://keras.io/losses/

https://isaacchanghau.github.io/2017/06/07/Loss-Functions-in-Artificial-Neural-Networks/

Arsenal

- Optimizadores (sgd, rmsprop, adam, ...)
- Funciones de pérdida (mean squared error, categorical crossentropy)
- Funciones de activación (sigmoid, ReLU, softmax, tanh, ...)

¿Cómo diseñar una arquitectura?

- Definir número de neuronas en capa de entrada y salida.
- Definir número de hidden layers. Definir número de neuronas en cada capa.
- Definir funciones de activación para capa de salida.
- Definir funciones de activación para otras capas.
- Definir loss function.
- Definir optimizer.
- Definir epoch y learning_rate.

- Ser feliz :)

Resumiendo...

1. Utilizamos **FeedForward** para ir de inicio a final de la red neuronal.
2. Calculamos el error con una **Loss Function.**
3. Utilizamos un **Optimizer (Gradient Descent)** para minimizar el error.
4. Se utiliza **Backpropagation** para ajustar los pesos y bias para minimizar el error.
5. Para esto se necesita utilizar la derivada parcial del Loss Function y la derivada de las **funciones de activación**.
6. En **regresión lineal**, utilizamos un **training set** para aprender los parámetros. Esto crea un **modelo** que describe el comportamiento.
7. Si se entrena mucho, se puede tener **overfitting**. Si se entrena poco, **underfitting**.