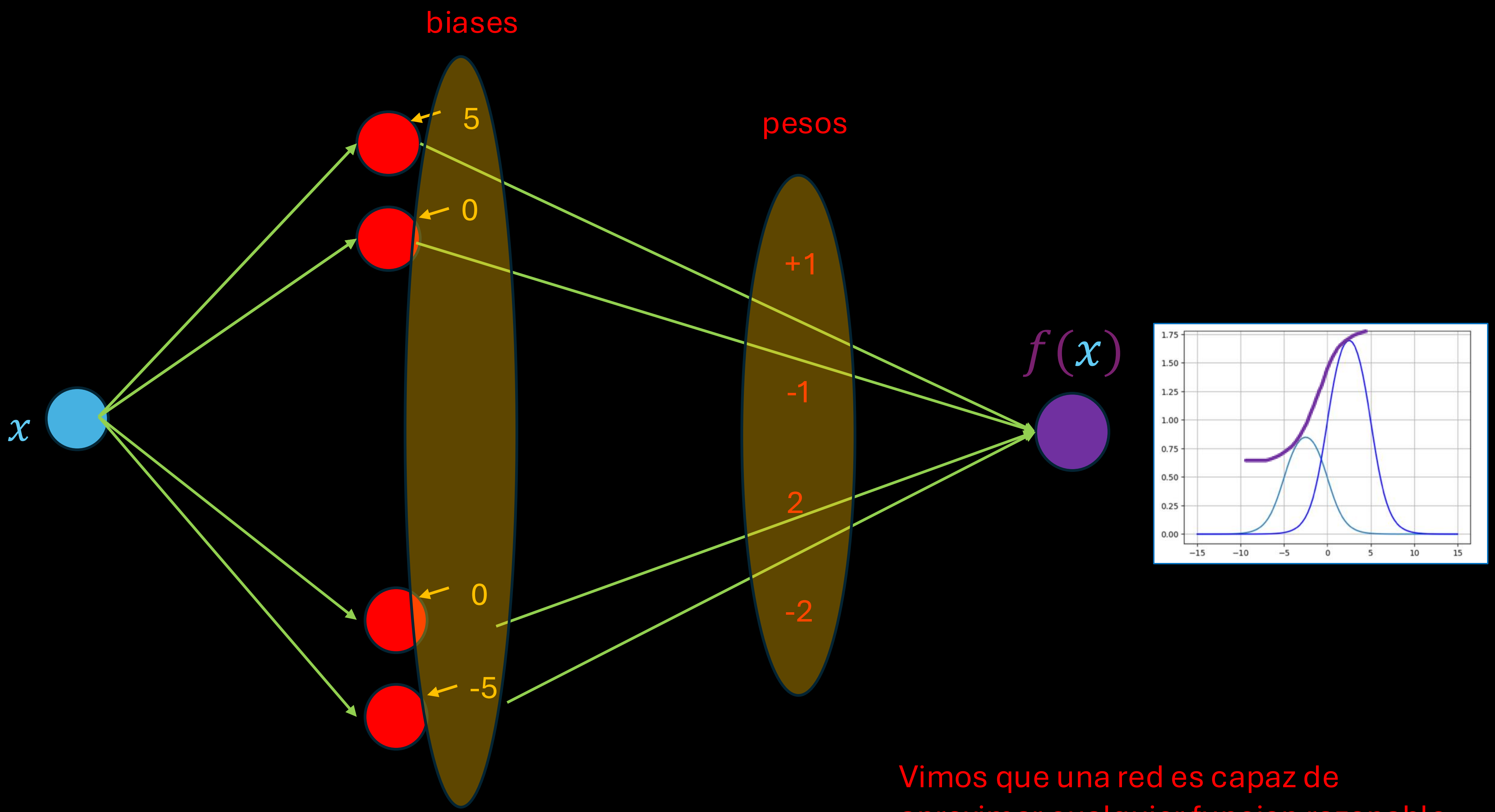
Back propagation

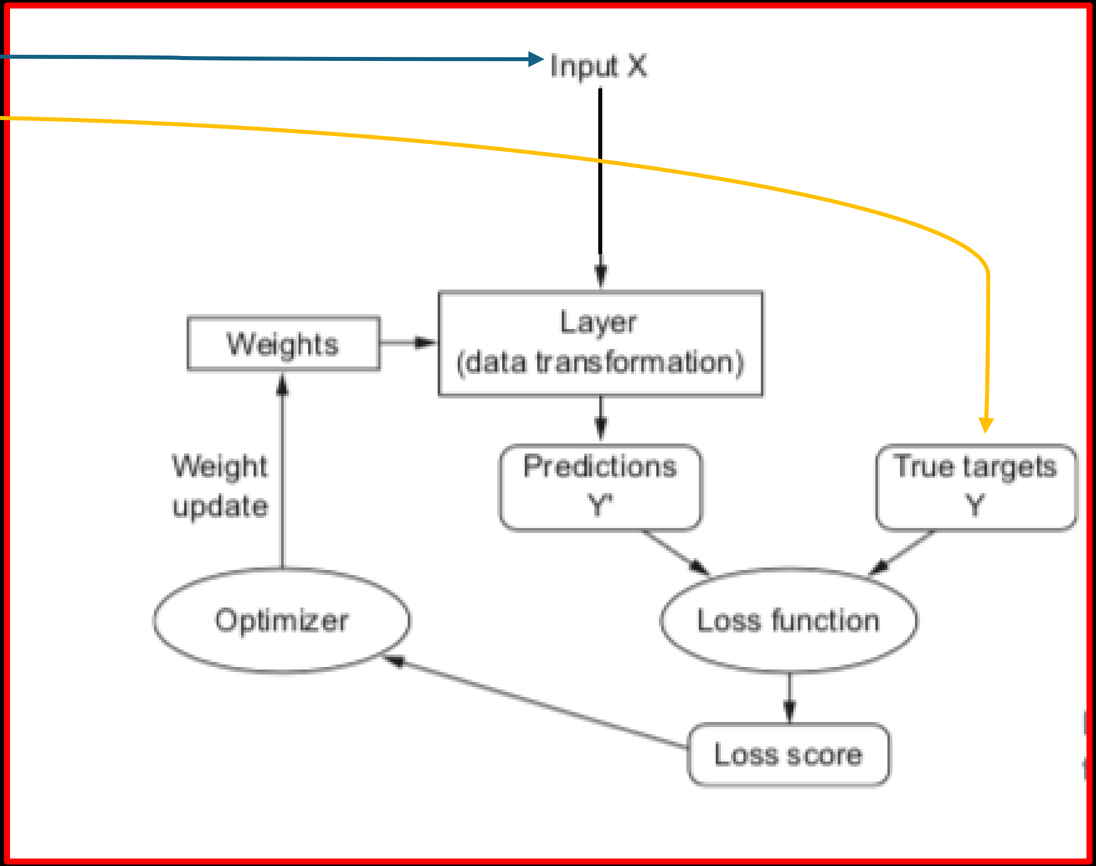
El modo de entrenar una red, que lo cambio todo



Vimos que una red es capaz de
aproximar cualquier funcion razonable

Rossenblatt ya habia propuesto que para entrenar a una red, se podia trabajar iterativamente, computando la diferencia entre lo que plantea la red y lo esperado para el ejemplo, y modificando a la red de tal modo de achicar esa diferencia

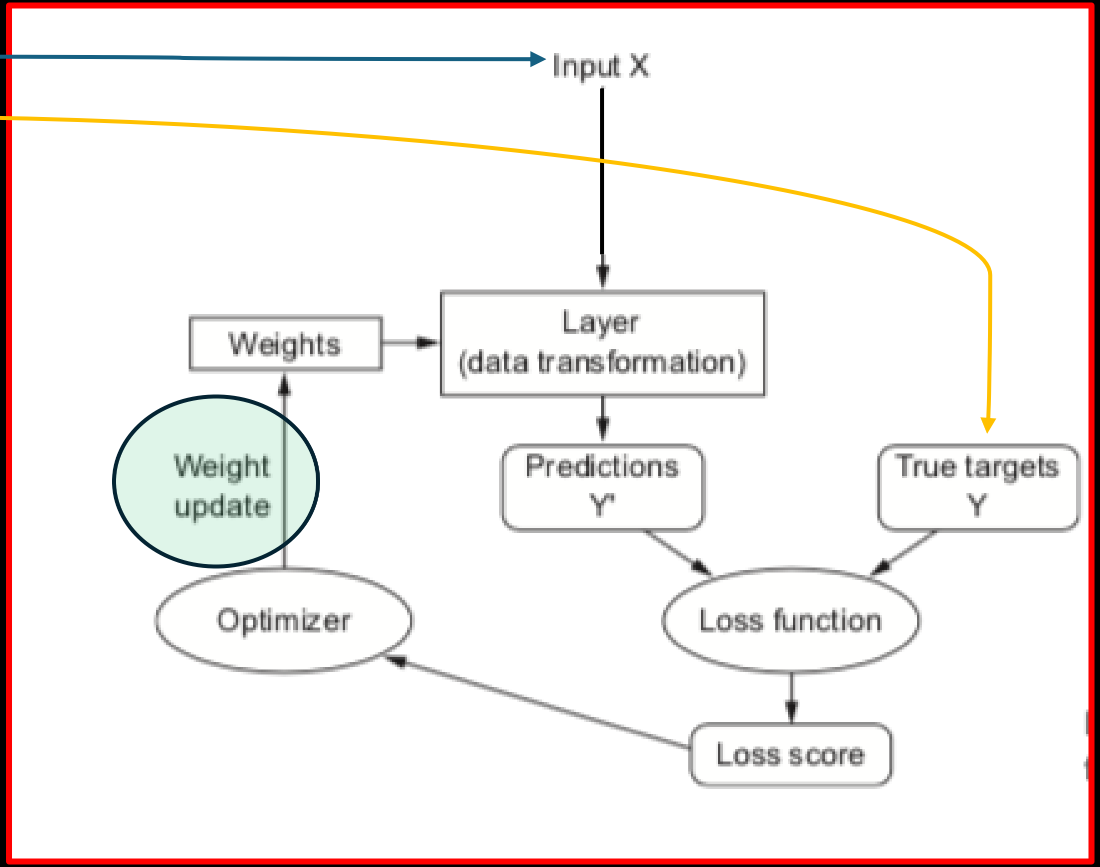
(data, resultado)=(X, Y)



Rossenblatt ya habia propuesto que para entrenar a una red, se podia trabajar iterativamente, computando la diferencia entre lo que plantea la red y lo esperado para el ejemplo, y modificando a la red de tal modo de achicar esa diferencia

(data, resultado)=(X, Y)

Como hacer el update?



Redes de propagación hacia delante y su entrenamiento

(Rumelhart, Hinton, Williams, Nature 1986)

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of the input units. If the input units are directly connected to the output units it is relatively easy to find learning rules that iteratively adjust the relative strengths of the connections so as to progressively reduce the difference between the actual and desired output vectors[2]. Learning becomes more interesting but more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input–output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \qquad (1)$$

Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, y_j, which is a non-linear function of its total input

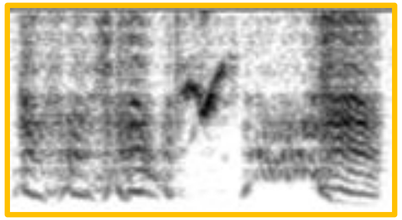
$$y_j = \frac{1}{1+e^{-x_j}} \qquad (2)$$

† To whom correspondence should be addressed.

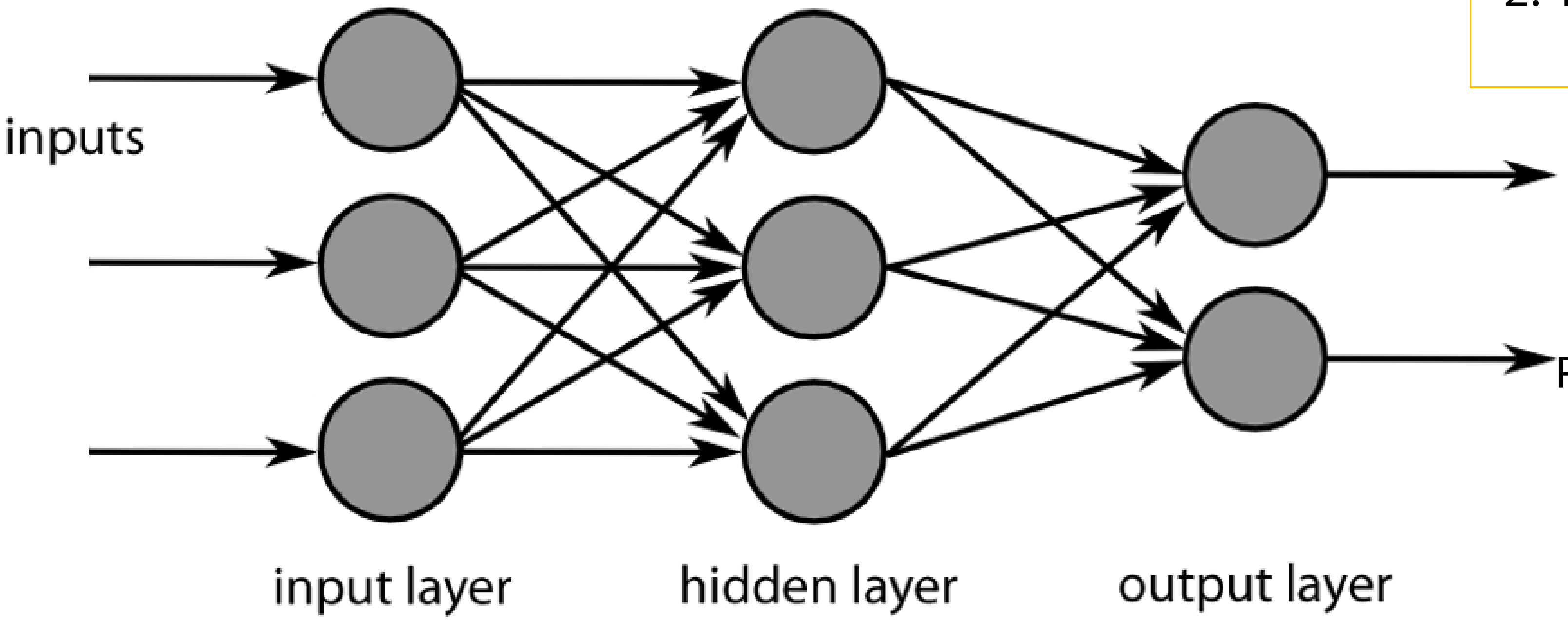
How do these networks work?

a. classification

1. Traduce the input image into a matrix



x_j



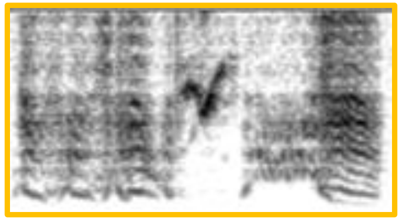
2. For a given set of parameters:

Probability of being the sonogram of the song of a finch

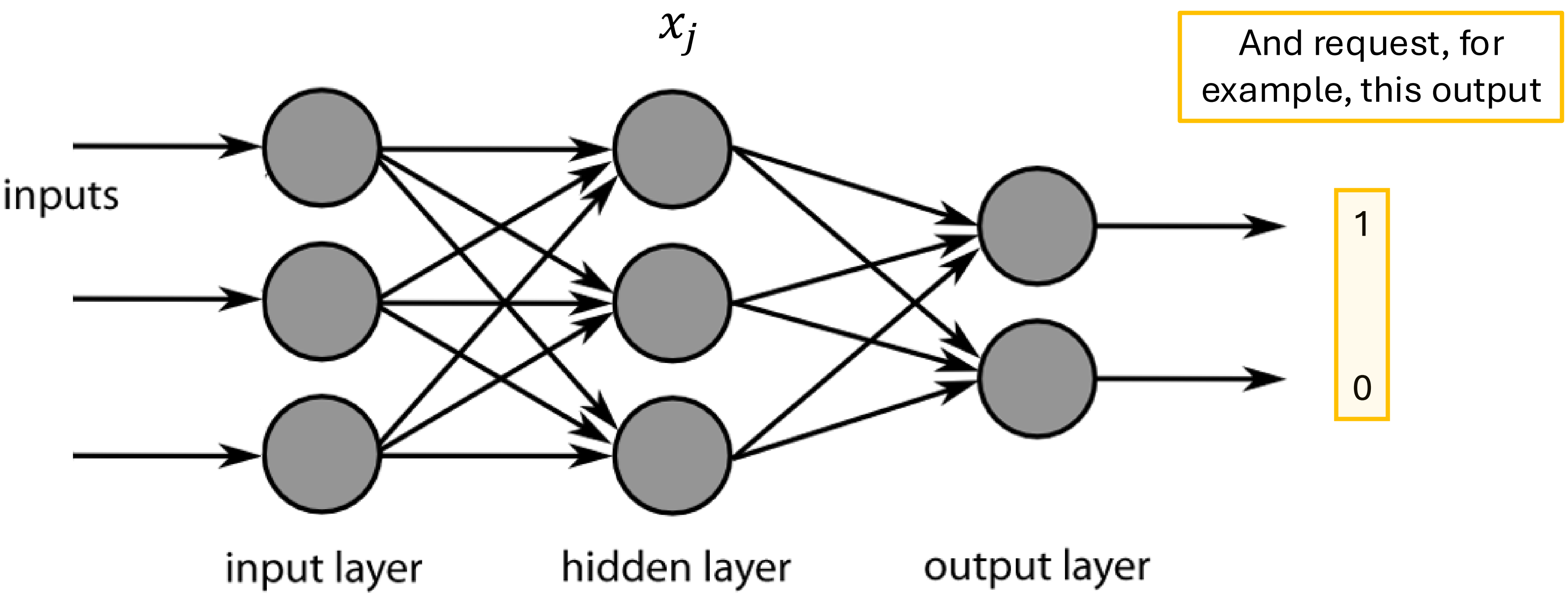
Probability of being the sonogram of the song of a canary

Input, un vector de "features"

$$x_j = S\left(\sum_{i=1}^{n} W_{ji}\, x_i\right), \qquad s(x) = \frac{1}{1+e^{-x}}$$



we traduce
the image
into numbers



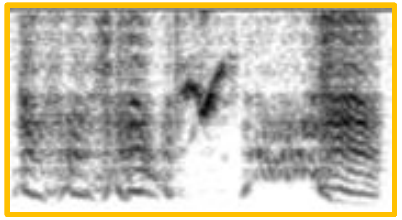
$$x_j = S\left(\sum_{i=1}^{n} W_{ji}\, x_i\right)$$

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

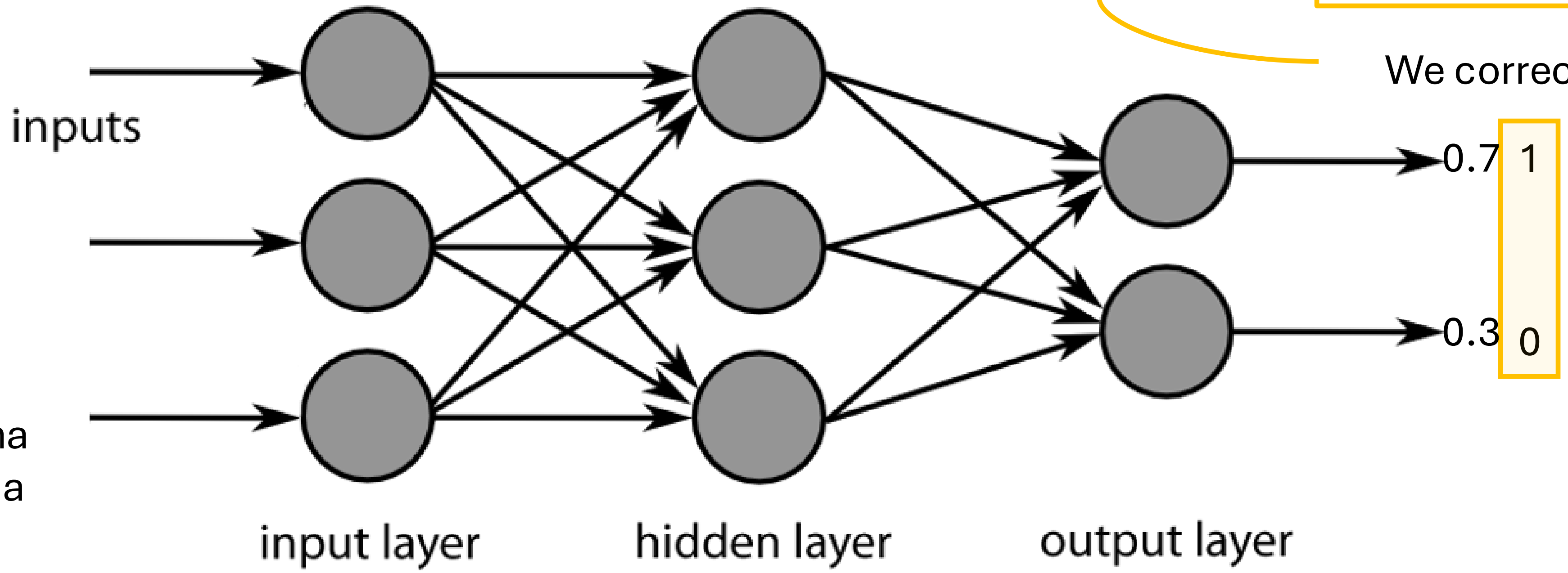
If with the present parameters, we get an error

We correct the weights

x_j



Se traduce el diagrama en numeros sobre una matriz



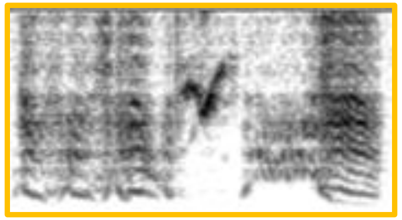
$$x_j = S(\sum_{i=1}^{n} W_{ji}\, x_i)$$

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

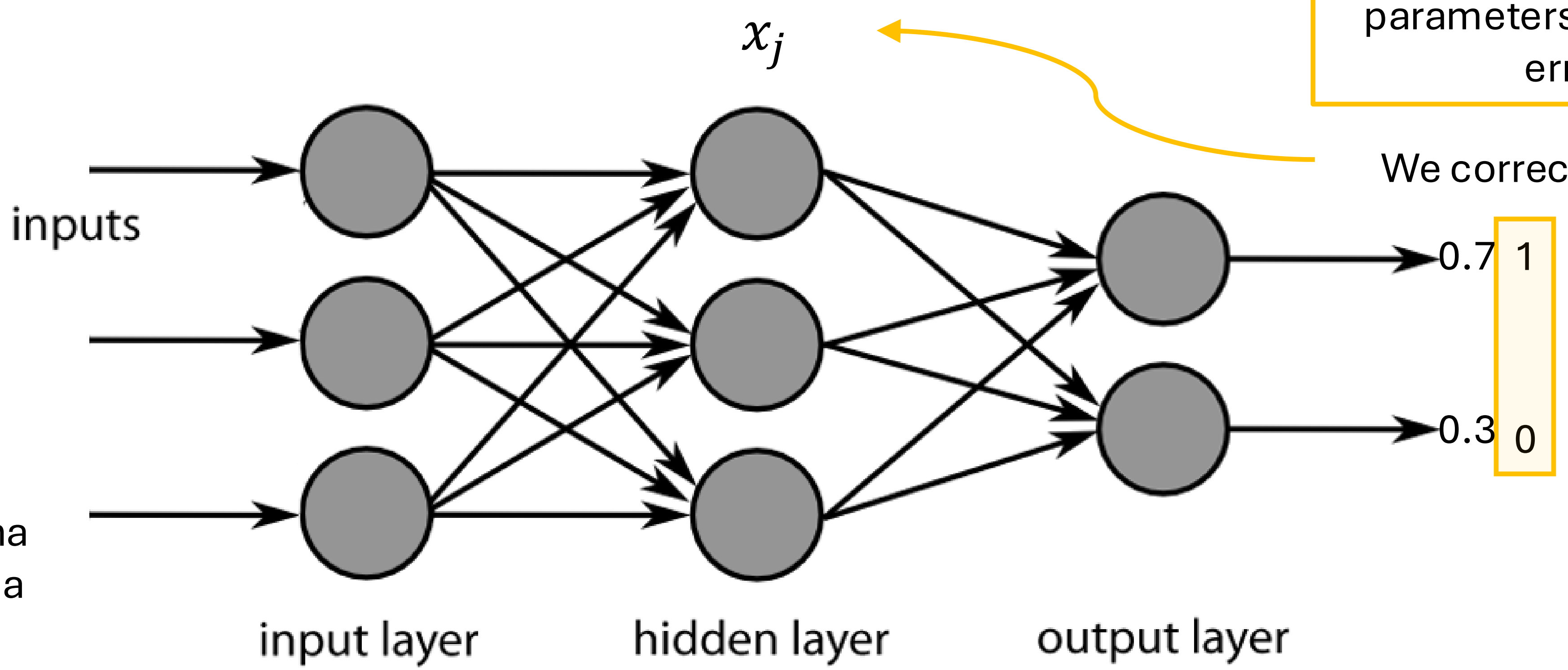
Como hacer para que esa correccion no sea solo un “ir a otro lado”, sino “ir a uno mejor”?

If with the present parameters, we get an error

We correct the weights



Se traduce el diagrama en numeros sobre una matriz



$$x_j = S\left(\sum_{i=1}^{n} W_{ji}\, x_i\right)$$

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

How does it actually work?

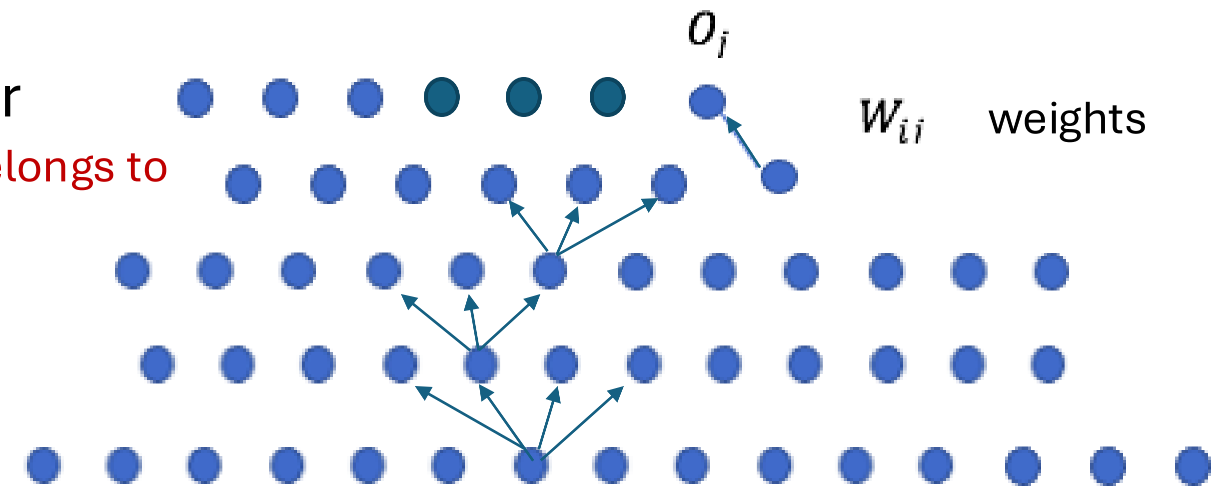
(let us start with one feedforward architecture)

Output layer

The class the image belongs to

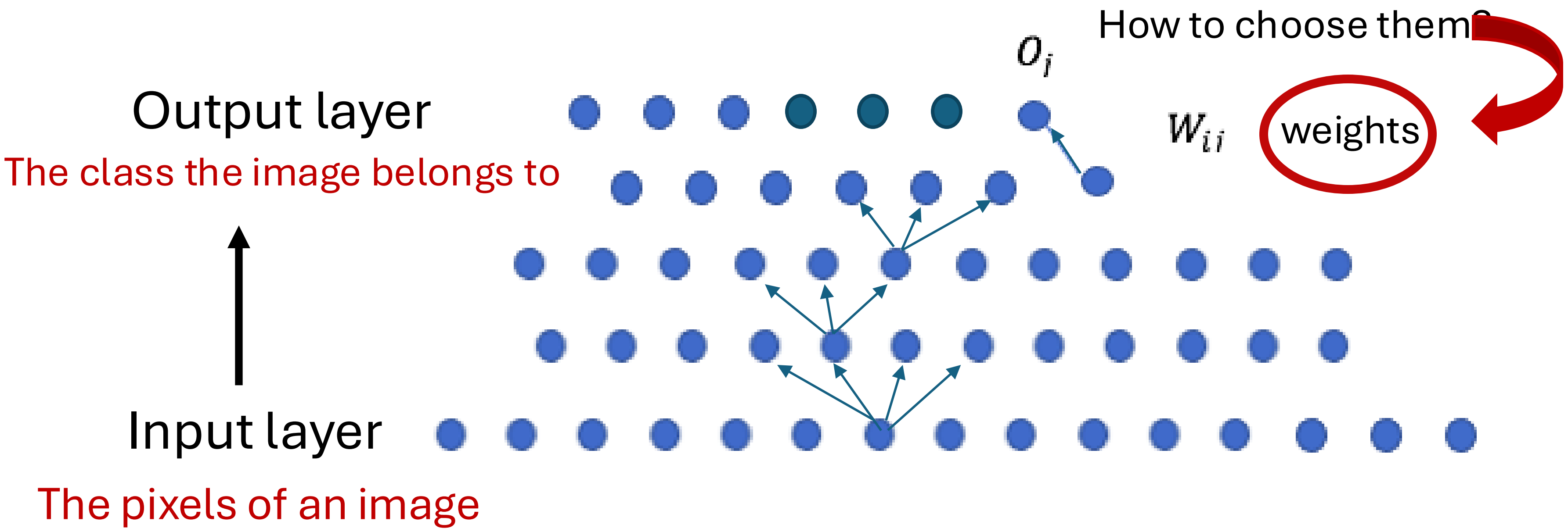
Input layer

The pixels of an image



How does it actually work?

(let us start with one feedforward architecture)

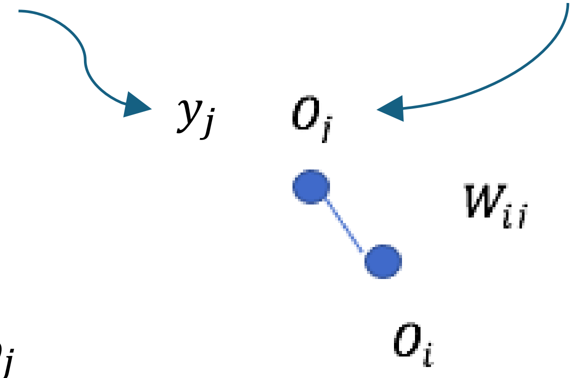


How does it actually work?

Suposse the unit *j is* in the last layer,
and we touch a weight
in the immediately previous layer

Expected value in unit j

Actual output in unit j



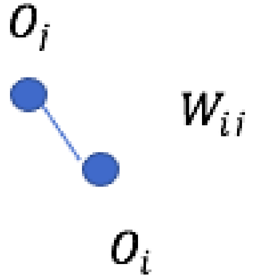
$E = y_j - o_j$

(*E* stands for “error”)

How does it actually work?

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}},$$

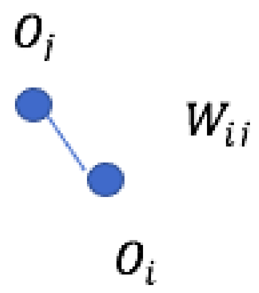
We want to **correct the weight** W_{ij} in order to decrease the error



How does it actually work?

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}},$$

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial O_j} \frac{\partial O_j}{\partial net_j} \frac{\partial net_j}{\partial W_{ij}}.$$



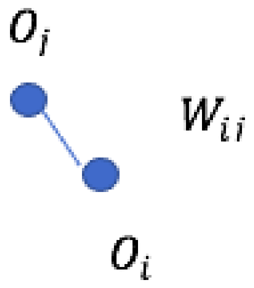
We use the **chain rule**:

the error changes because changing the weight W_{ij}

1. It changes the net activity arriving at j
2. If the net activity changes, the output of j changes
3. If the output of j changes, the error changes

How does it actually work?

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}},$$

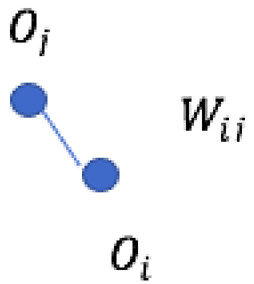


$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial O_j} \frac{\partial O_j}{\partial net_j} \frac{\partial net_j}{\partial W_{ij}},$$

$$\frac{\partial net_j}{\partial W_{ij}} = O_i.$$ since $net_j = \sum W_{ij} O_i$

How does it actually work?

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}},$$



$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial O_j} \frac{\partial O_j}{\partial net_j} \frac{\partial net_j}{\partial W_{ij}}.$$

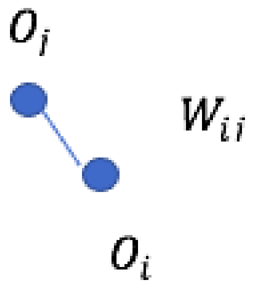
$$\frac{\partial net_j}{\partial W_{ij}} = O_i.$$ Since $net_j = \sum W_{ij} O_i$

As $O_j = S(net_j)$ and $S(net_j) = 1/(1 + e^{-net_j})$,

then $$\frac{\partial O_j}{\partial net_j} = O_j(1 - O_j).$$

How does it actually work?

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}},$$



$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial O_j} \frac{\partial O_j}{\partial net_j} \frac{\partial net_j}{\partial W_{ij}}.$$

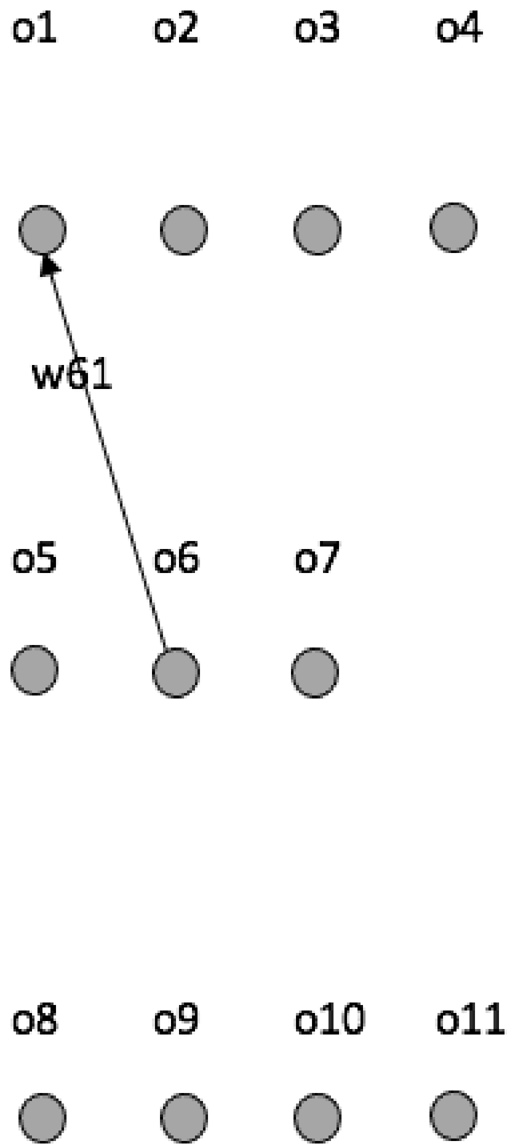
$$\frac{\partial net_j}{\partial W_{ij}} = O_i.$$ Since $net_j = \sum W_{ij} O_i$

as $O_j = S(net_j)$ and $S(net_j) = 1/(1 + e^{-net_j})$,

then $$\frac{\partial O_j}{\partial net_j} = O_j(1 - O_j).$$

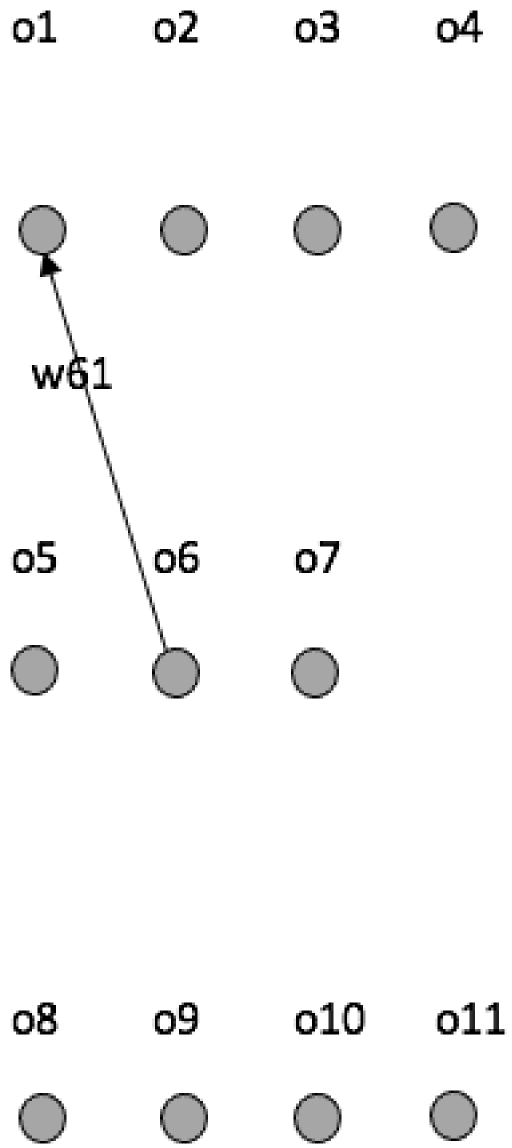
$$\frac{\partial E}{\partial O_j} = (y^j_{elemento} - O_j),$$

Example



$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial net_j}\frac{\partial net_j}{\partial W_{ij}}.$$

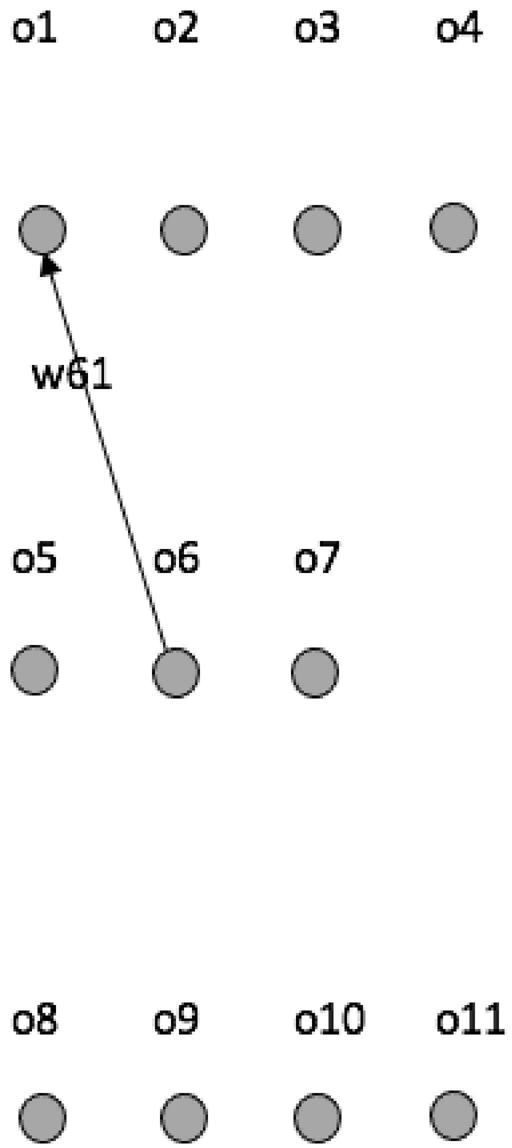
Example



$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial net_j}\frac{\partial net_j}{\partial W_{ij}}.$$

$$\frac{\partial E}{\partial W61} = (o1 - t1)\big(o1(1 - o1)\big)o6$$

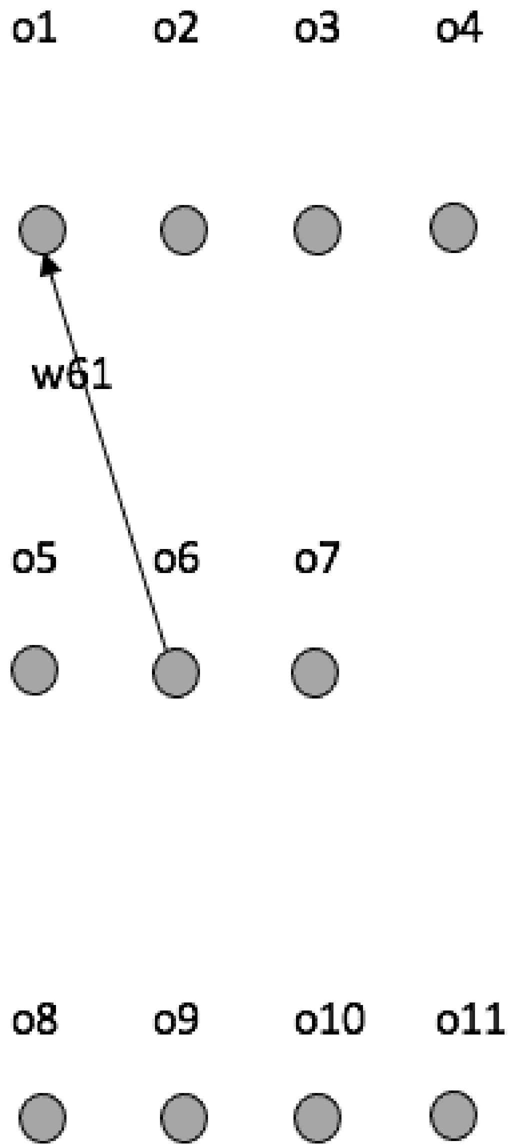
Example



$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial net_j}\frac{\partial net_j}{\partial W_{ij}}.$$

$$\frac{\partial E}{\partial W61} = (o1 - t1)\left(o1(1 - o1)\right)o6$$

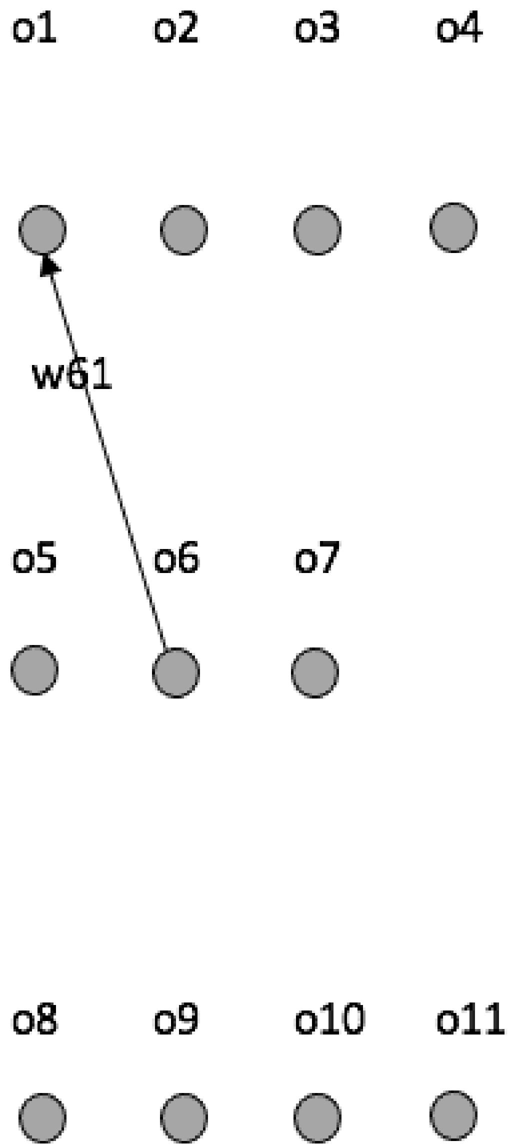
Example



$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial O_j} \frac{\partial O_j}{\partial net_j} \frac{\partial net_j}{\partial W_{ij}}.$$

$$\frac{\partial E}{\partial W61} = (o1 - t1)\big(o1(1 - o1)\big)o6$$

Example

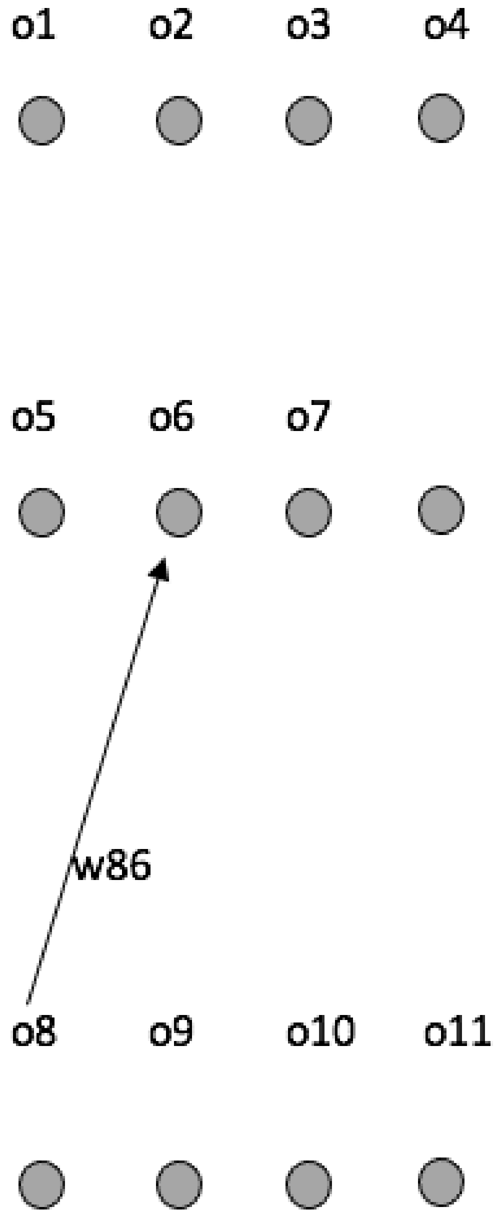


$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial O_j} \frac{\partial O_j}{\partial net_j} \frac{\partial net_j}{\partial W_{ij}}.$$

$$\frac{\partial E}{\partial W61} = (o1 - t1)\big(o1(1 - o1)\big)o6$$

The interesting thing is that
it only depends on the outputs O
that I got with the original W values

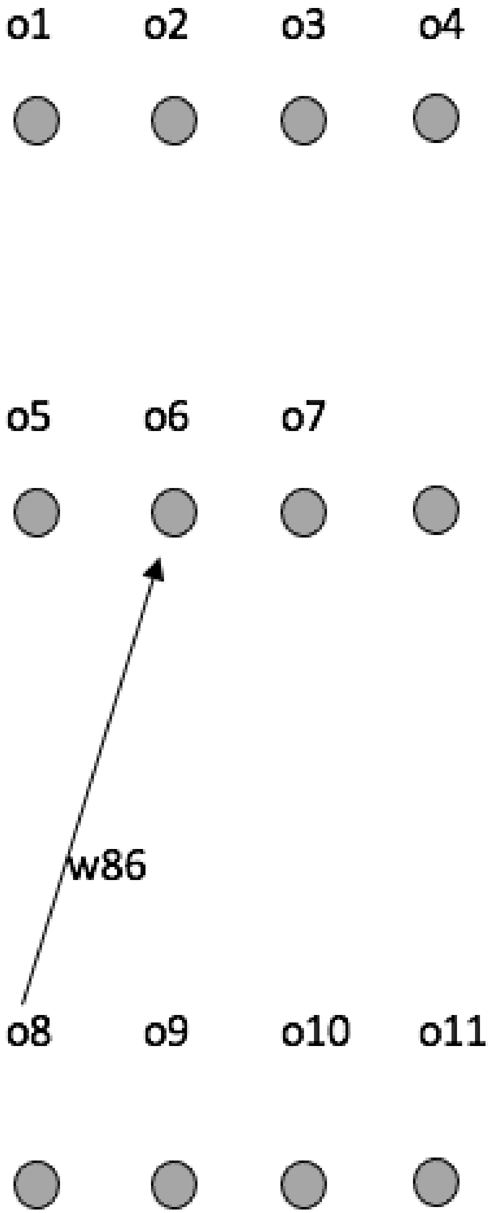
Example 2



If the weight being modified is in the **previous to the last layer**, the error changes due to changes in all the units of the last layer

$$\frac{\partial E}{\partial W86} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6} \frac{\partial O6}{\partial net6} \frac{\partial net6}{\partial W86} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6} \big(o6(1 - o6)\big) o8$$

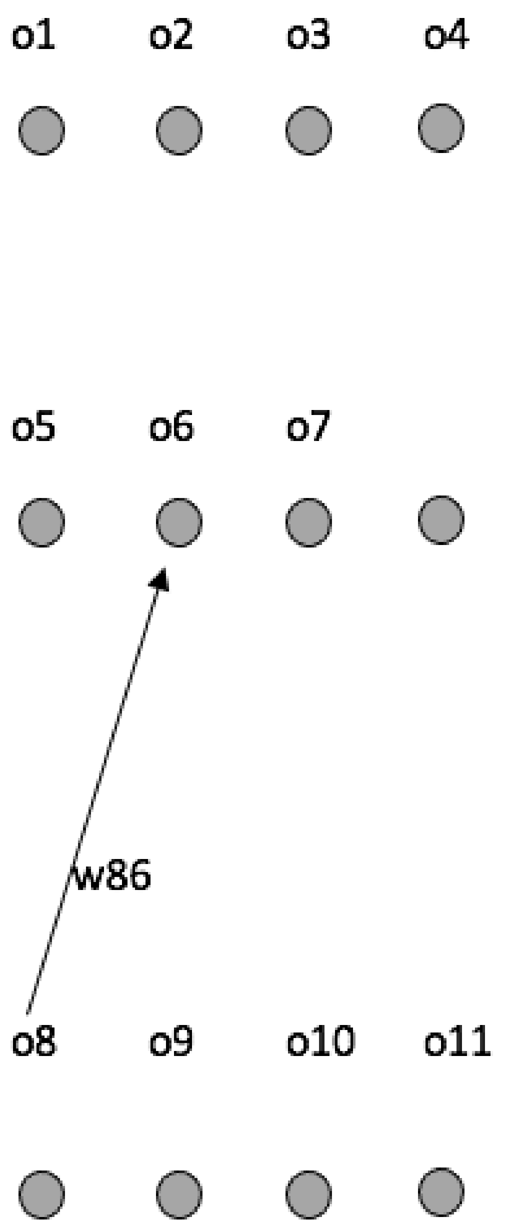
Example 2



$$\frac{\partial E}{\partial W86} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6}\frac{\partial O6}{\partial net6}\frac{\partial net6}{\partial W86} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6}\big(o6(1-o6)\big)o8$$

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6} = \frac{\partial E}{\partial O1}\frac{\partial O1}{\partial net1}\frac{\partial net1}{\partial O6} + \frac{\partial E}{\partial O2}\frac{\partial O2}{\partial net2}\frac{\partial net2}{\partial O6} + \frac{\partial E}{\partial O3}\frac{\partial O3}{\partial net3}\frac{\partial net3}{\partial O6} + \frac{\partial E}{\partial O4}\frac{\partial O4}{\partial net4}\frac{\partial net4}{\partial O6}$$

Example 2

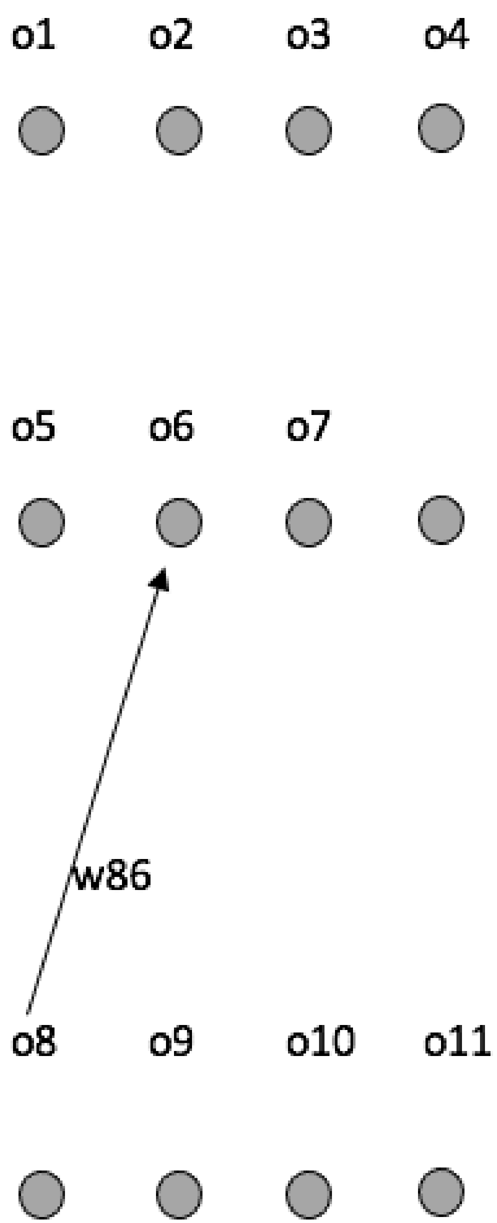


$$\frac{\partial E}{\partial W86} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6}\frac{\partial O6}{\partial net6}\frac{\partial net6}{\partial W86} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6}\big(o6(1-o6)\big)o8$$

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6} = \frac{\partial E}{\partial O1}\frac{\partial O1}{\partial net1}\frac{\partial net1}{\partial O6} + \frac{\partial E}{\partial O2}\frac{\partial O2}{\partial net2}\frac{\partial net2}{\partial O6} + \frac{\partial E}{\partial O3}\frac{\partial O3}{\partial net3}\frac{\partial net3}{\partial O6} + \frac{\partial E}{\partial O4}\frac{\partial O4}{\partial net4}\frac{\partial net4}{\partial O6}$$

$$\begin{aligned}\frac{\partial E}{\partial O6} = &(O1-t1)\big(O1(1-O1)\big)W16 \\ &+(O2-t2)\big(O2(1-O2)\big)W26 \\ &+(O3-t3)\big(O3(1-O3)\big)W36 \\ &+(O4-t4)\big(O4(1-O4)\big)W46\end{aligned}$$

Example 2

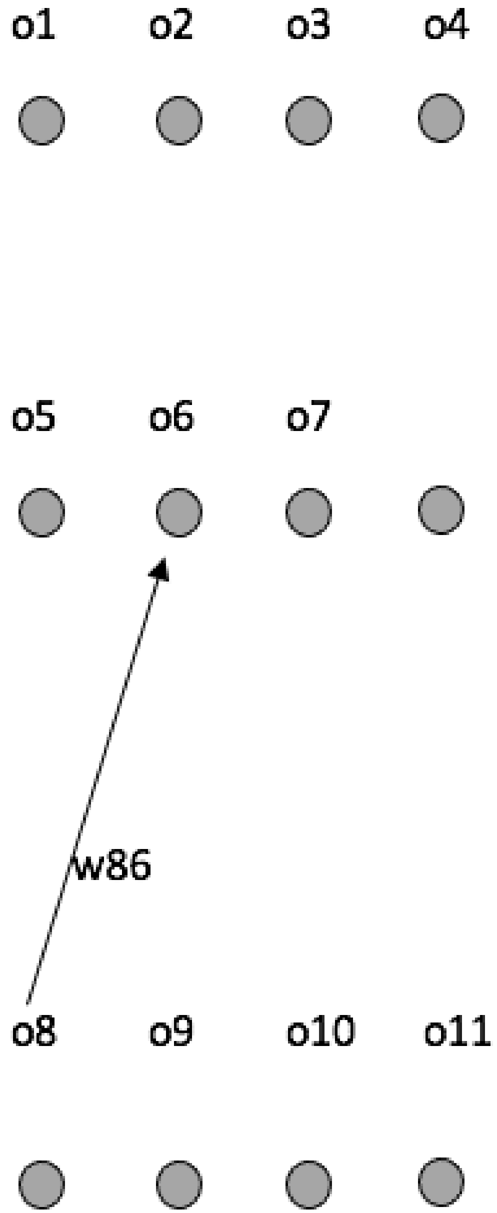


$$\frac{\partial E}{\partial W86} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6}\frac{\partial O6}{\partial net6}\frac{\partial net6}{\partial W86} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6}\big(o6(1 - o6)\big)o8$$

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6} = \frac{\partial E}{\partial O1}\frac{\partial O1}{\partial net1}\frac{\partial net1}{\partial O6} + \frac{\partial E}{\partial O2}\frac{\partial O2}{\partial net2}\frac{\partial net2}{\partial O6} + \frac{\partial E}{\partial O3}\frac{\partial O3}{\partial net3}\frac{\partial net3}{\partial O6} + \frac{\partial E}{\partial O4}\frac{\partial O4}{\partial net4}\frac{\partial net4}{\partial O6}$$

$$\begin{aligned}\frac{\partial E}{\partial O6} = {} & (O1 - t1)\big(O1(1 - O1)\big)W16 \\ & +(O2 - t2)\big(O2(1 - O2)\big)W26 \\ & +(O3 - t3)\big(O3(1 - O3)\big)W36 \\ & +(O4 - t4)\big(O4(1 - O4)\big)W46\end{aligned}$$

Example 2

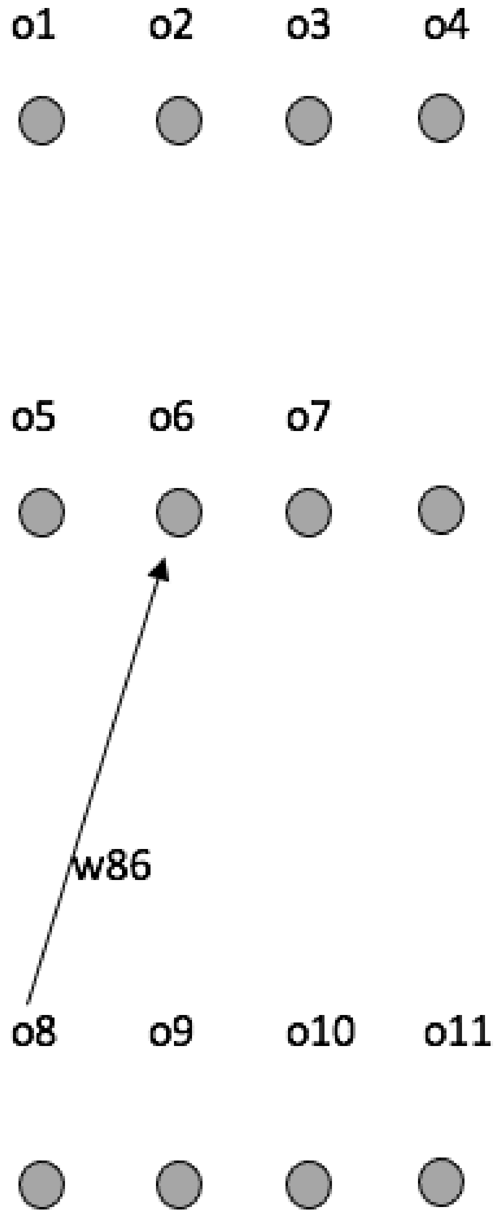


$$\frac{\partial E}{\partial W86} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6}\frac{\partial O6}{\partial net6}\frac{\partial net6}{\partial W86} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6}\big(o6(1-o6)\big)o8$$

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6} = \frac{\partial E}{\partial O1}\frac{\partial O1}{\partial net1}\frac{\partial net1}{\partial O6} + \frac{\partial E}{\partial O2}\frac{\partial O2}{\partial net2}\frac{\partial net2}{\partial O6} + \frac{\partial E}{\partial O3}\frac{\partial O3}{\partial net3}\frac{\partial net3}{\partial O6} + \frac{\partial E}{\partial O4}\frac{\partial O4}{\partial net4}\frac{\partial net4}{\partial O6}$$

$$\begin{aligned}\frac{\partial E}{\partial O6} = {} & (O1-t1)\big(O1(1-O1)\big)W16 \\ & +(O2-t2)\big(O2(1-O2)\big)W26 \\ & +(O3-t3)\big(O3(1-O3)\big)W36 \\ & +(O4-t4)\big(O4(1-O4)\big)W46\end{aligned}$$

Example 2

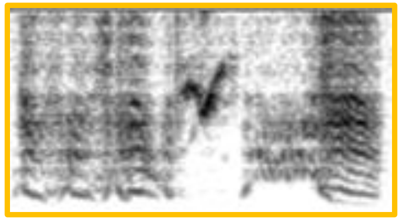


$$\frac{\partial E}{\partial W86} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6}\frac{\partial O6}{\partial net6}\frac{\partial net6}{\partial W86} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6}\big(o6(1-o6)\big)o8$$

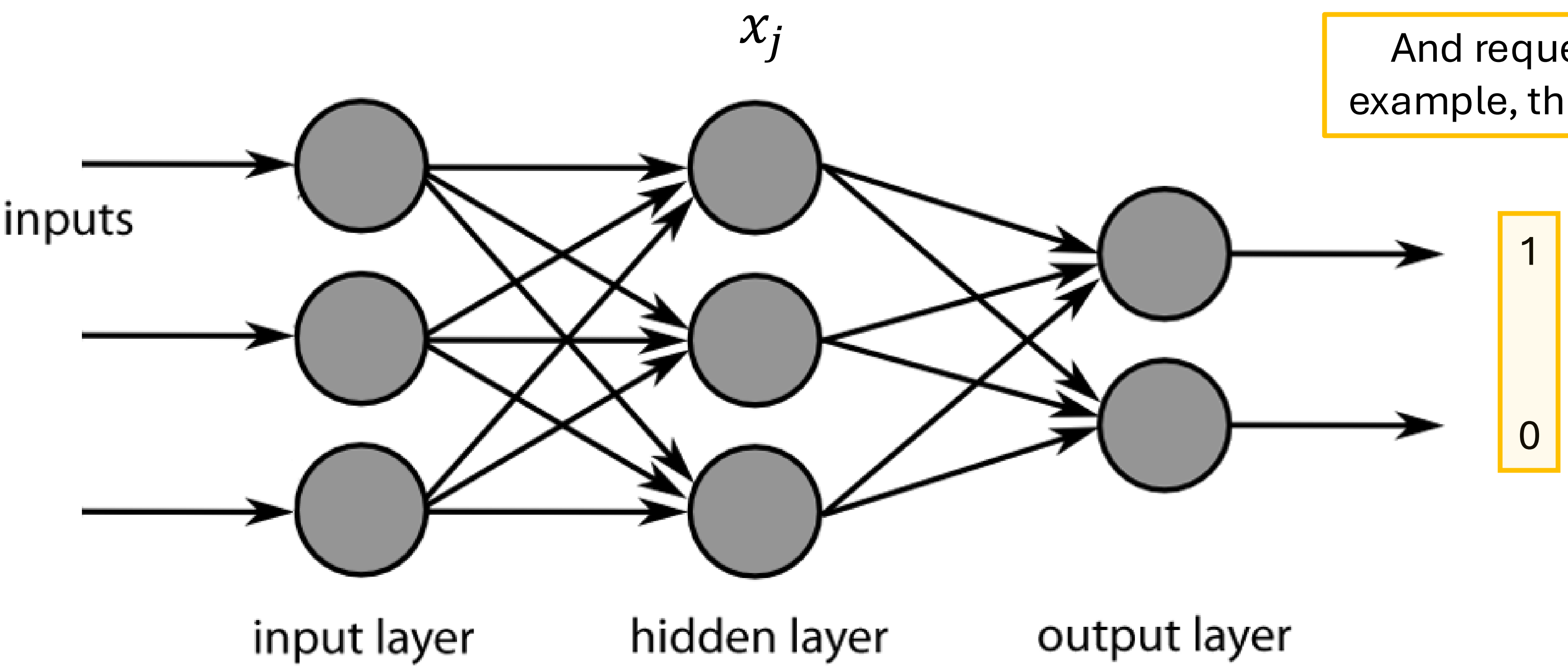
$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}6} = \frac{\partial E}{\partial O1}\frac{\partial O1}{\partial net1}\frac{\partial net1}{\partial O6} + \frac{\partial E}{\partial O2}\frac{\partial O2}{\partial net2}\frac{\partial net2}{\partial O6} + \frac{\partial E}{\partial O3}\frac{\partial O3}{\partial net3}\frac{\partial net3}{\partial O6} + \frac{\partial E}{\partial O4}\frac{\partial O4}{\partial net4}\frac{\partial net4}{\partial O6}$$

$$\begin{aligned}\frac{\partial E}{\partial O6} = {} & (O1-t1)\big(O1(1-O1)\big)W16 \\ & +(O2-t2)\big(O2(1-O2)\big)W26 \\ & +(O3-t3)\big(O3(1-O3)\big)W36 \\ & +(O4-t4)\big(O4(1-O4)\big)W46\end{aligned}$$

Tedious, but only *O* y *W* are involved



we traduce
the image
into numbers



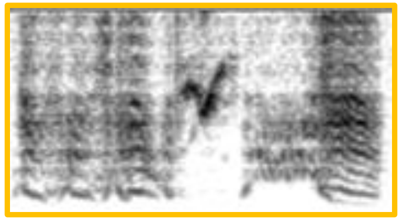
And request, for example, this output

$$x_j = S\left(\sum_{i=1}^{n} W_{ji}\, x_i\right)$$

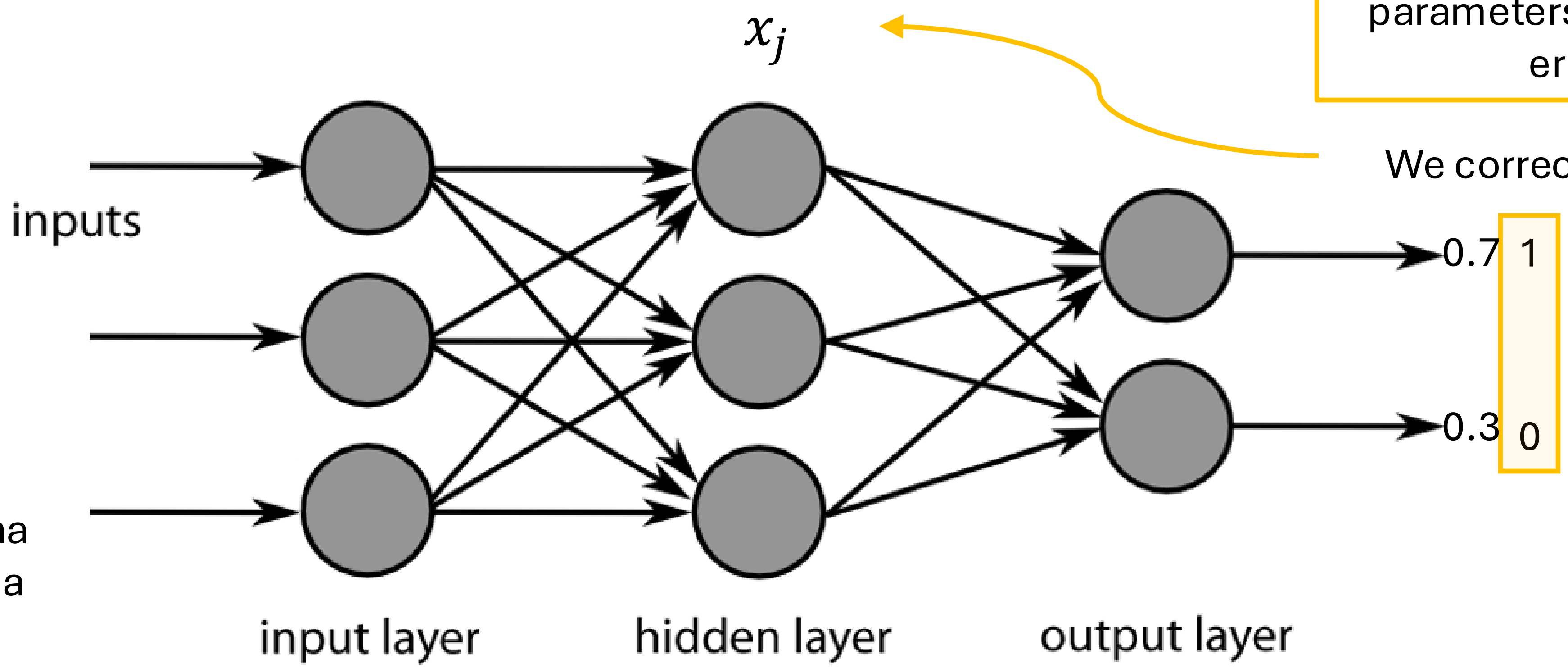
$$s(x) = \frac{1}{1 + e^{-x}}$$

If with the present parameters, we get an error

We correct the weights



Se traduce el diagrama en numeros sobre una matriz



$$x_j = S\left(\sum_{i=1}^{n} W_{ji}\, x_i\right)$$

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

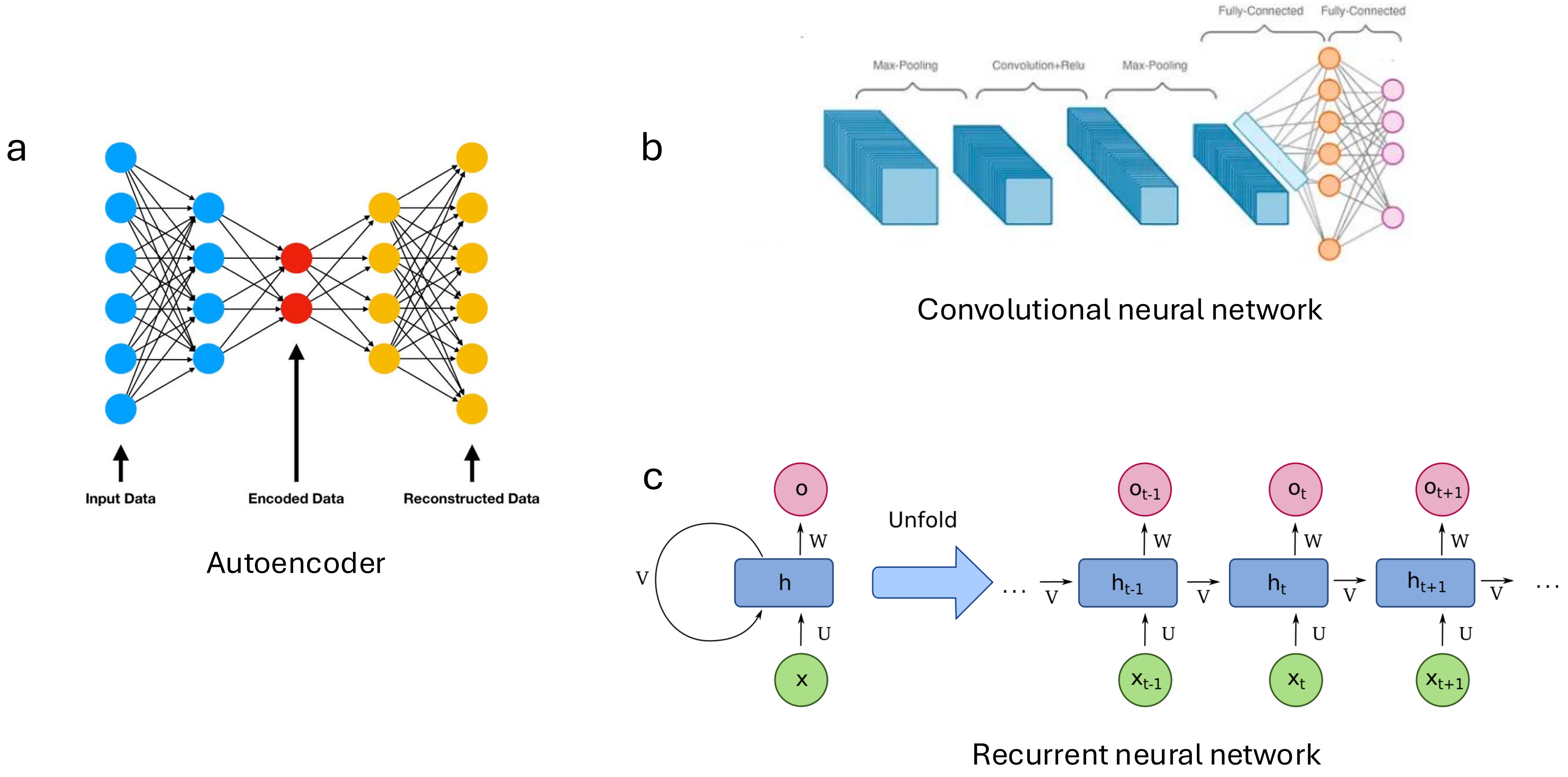
Does this mean that the scary AI that will leave us out of jobs is

= the chain rule?

Does this mean that the scary AI that will leave us out of jobs is

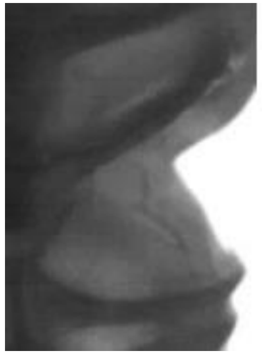
= the chain rule?

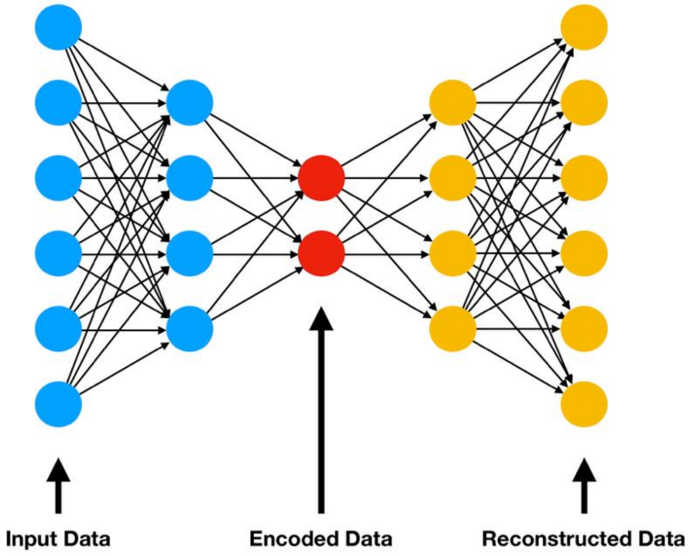
Yeap.

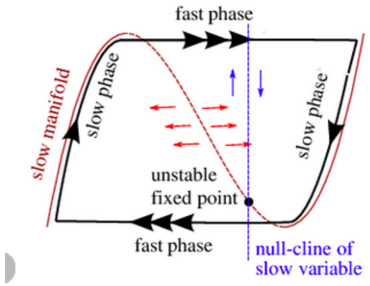


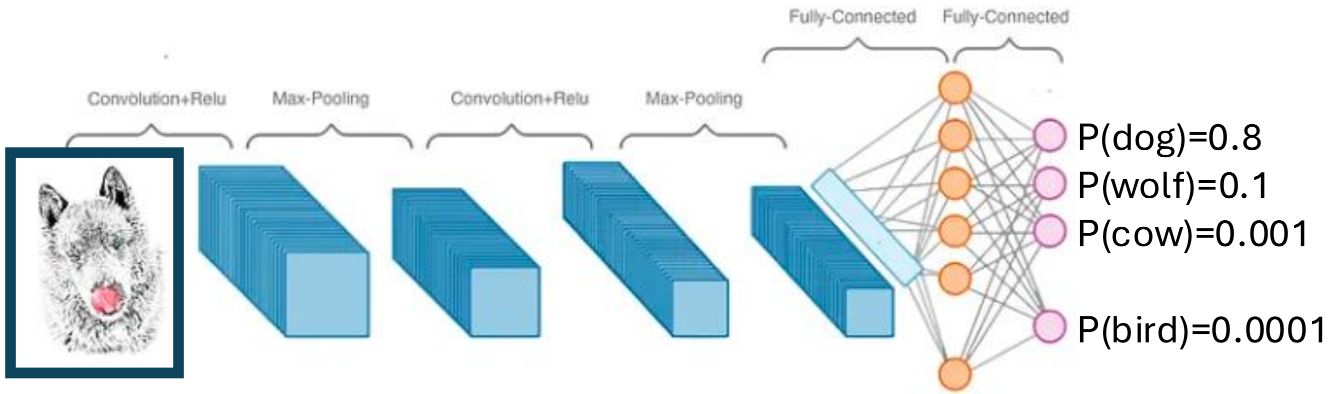
a
Input Data
Encoded Data
Reconstructed Data
Autoencoder
b
Max-Pooling
Convolution+Relu
Max-Pooling
Fully-Connected
Fully-Connected
Convolutional neural network
c
V
h
o
W
U
x
Unfold
...
h_{t-1}
h_t
h_{t+1}
o_{t-1}
o_t
o_{t+1}
x_{t-1}
x_t
x_{t+1}
Recurrent neural network

Autoencoder
(dimensional reduction)



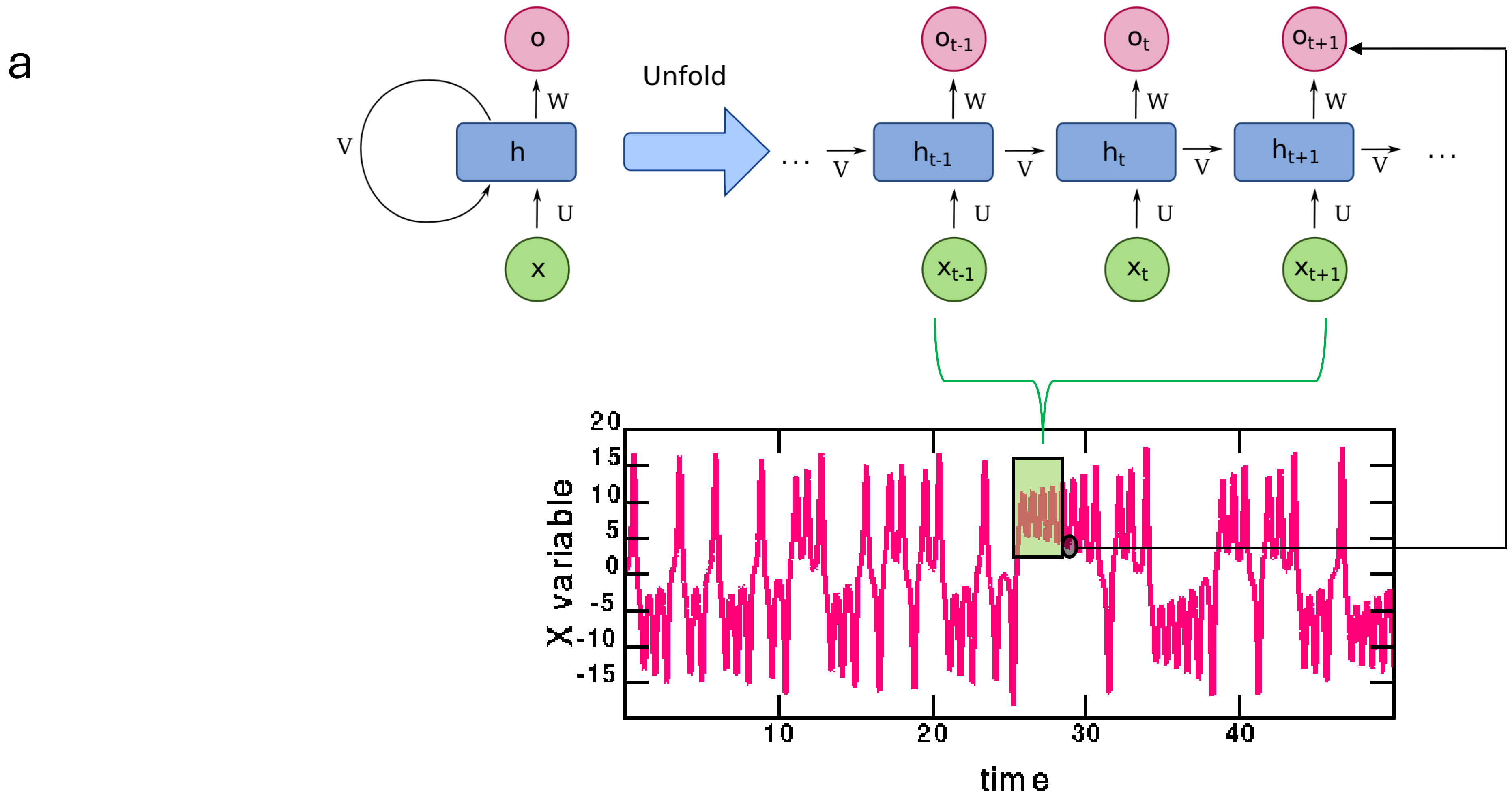




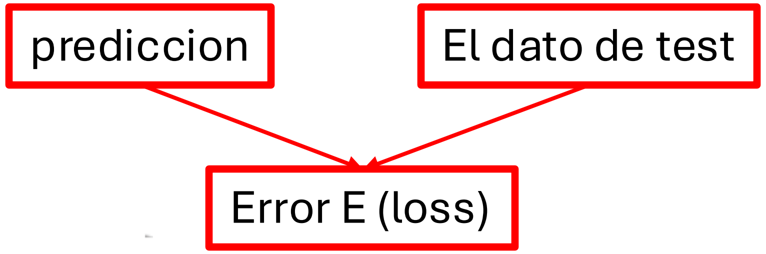


Convolutional neural network
classify

Recurrent neural network
(temporal predictions)



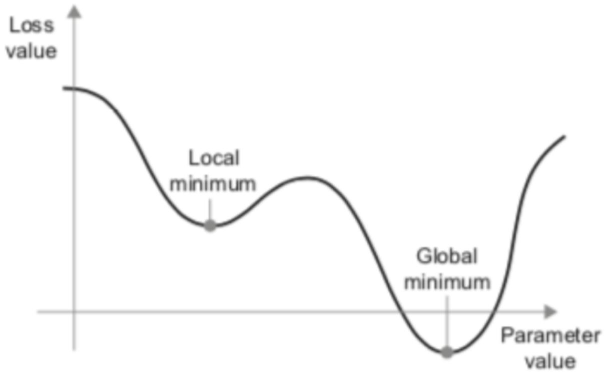
En todas, lo esencial es buscar minimizar el error entre lo predicho y lo deseado segun los ejemplos

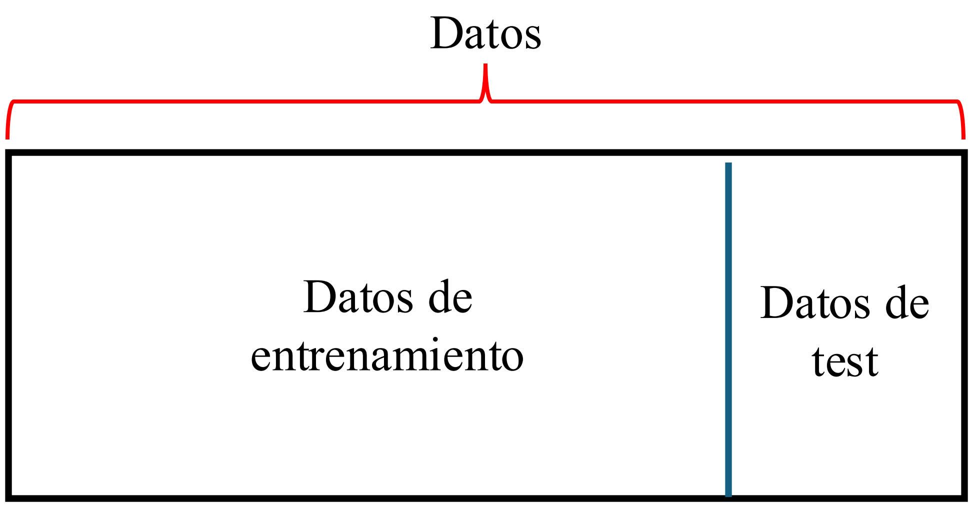


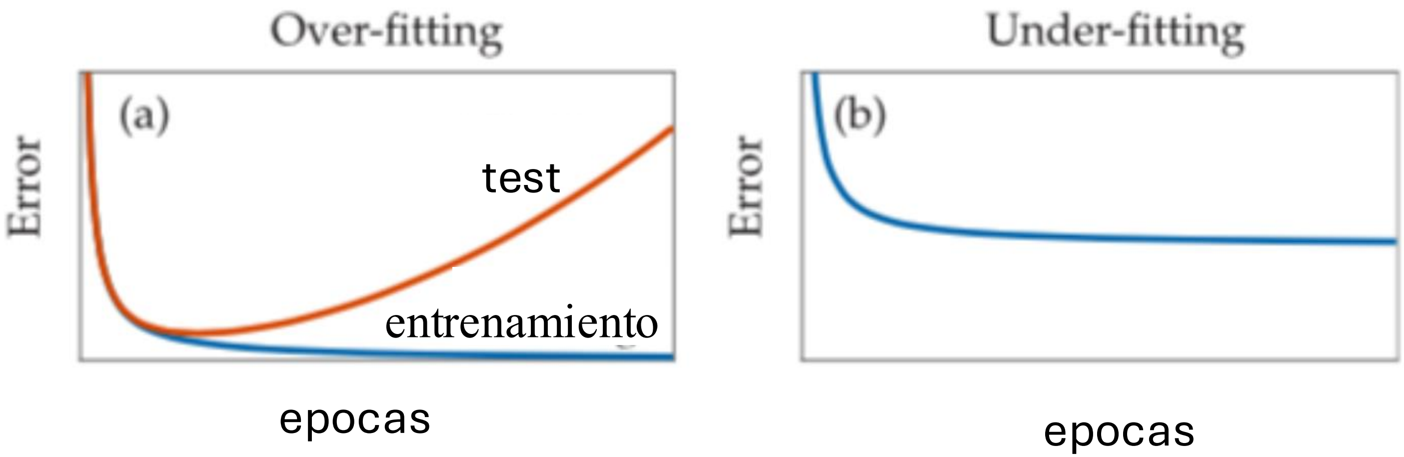
Y lo emplea para calcular los cambios en la red:

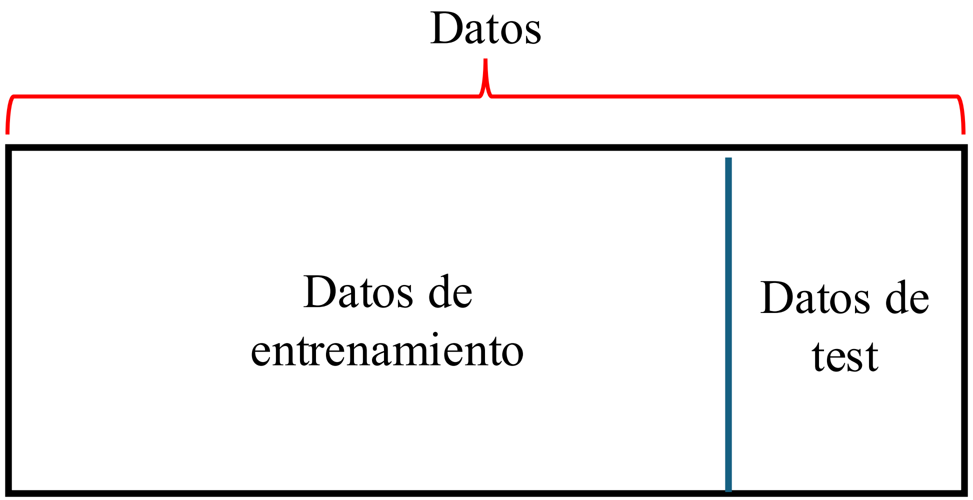
$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}},$$

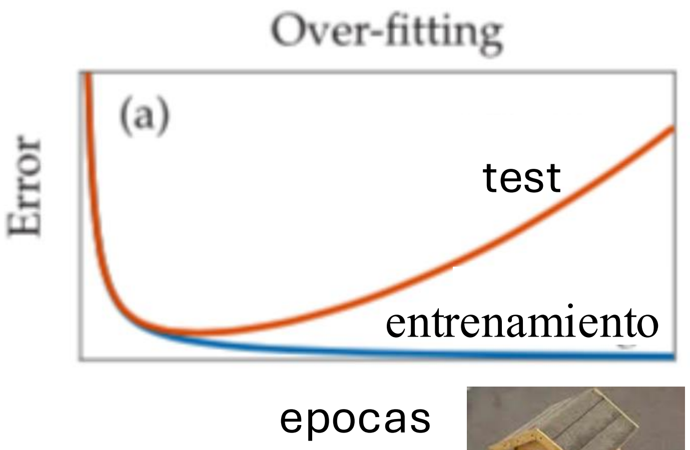
Con la idea de encontrar un mínimo en el cual $\Delta W_{ij} = 0$, resulta que

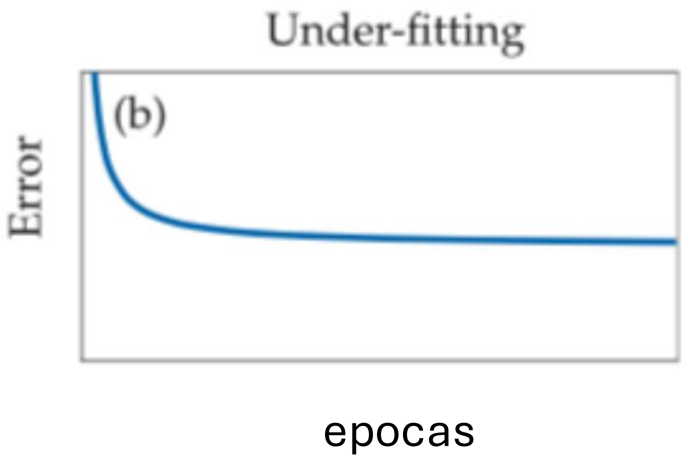



Datos
Datos de entrenamiento
Datos de test


Over-fitting
(a)
Error
test
entrenamiento
epocas
Under-fitting
(b)
Error
epocas

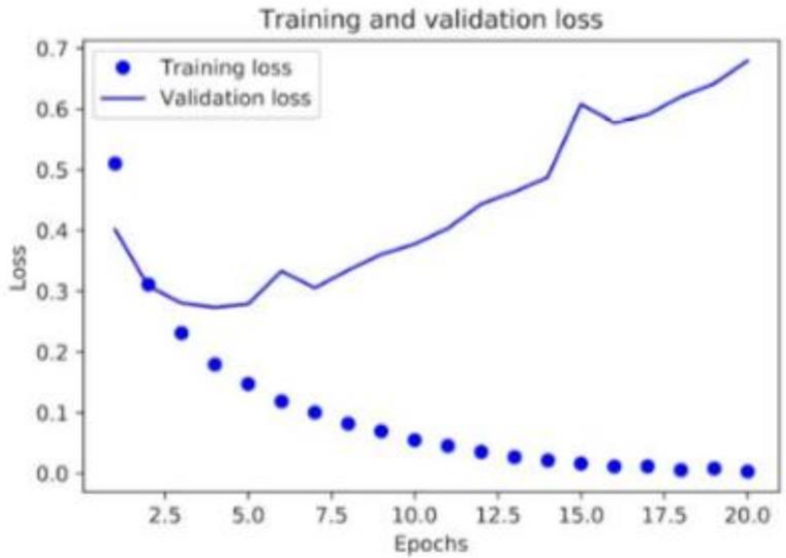






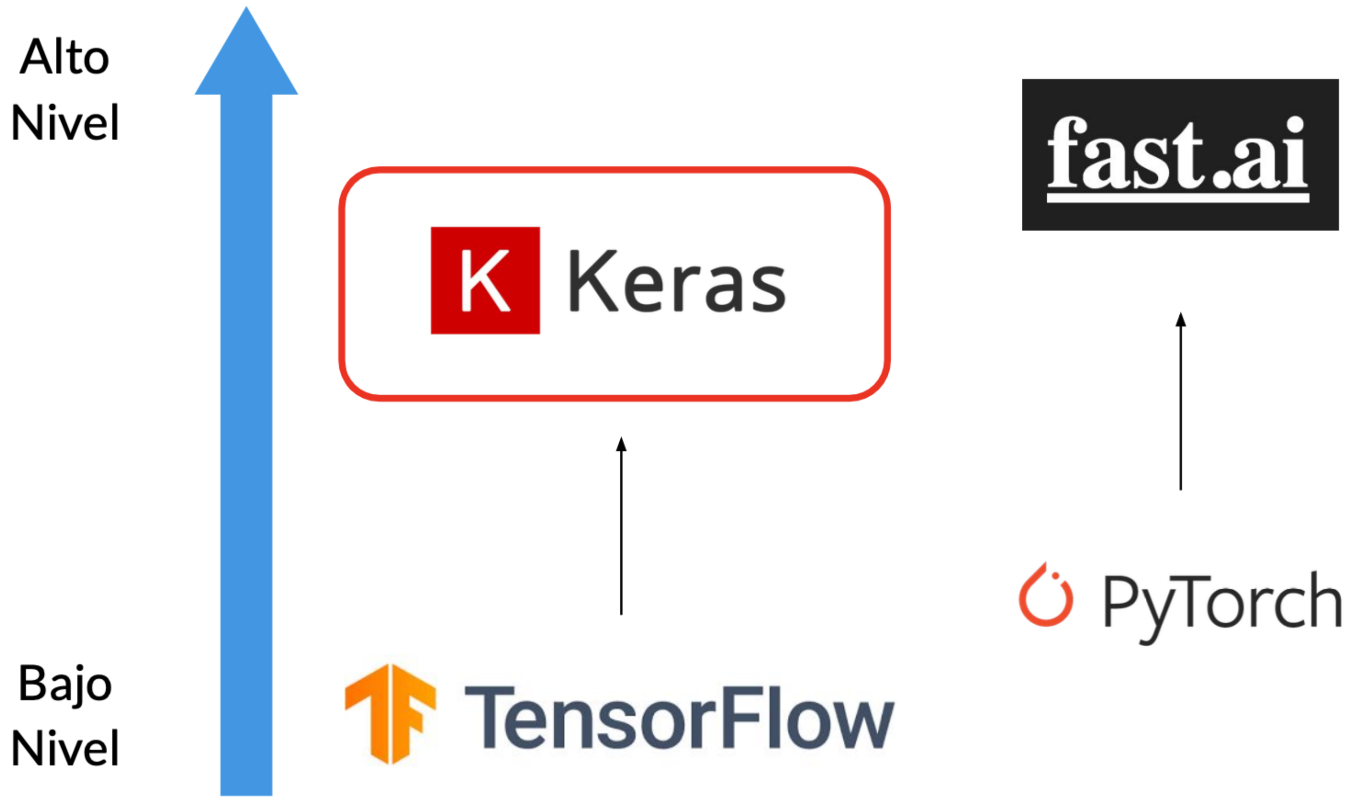
La explicacion mas clara del overfitting

Modos de evitar esto:



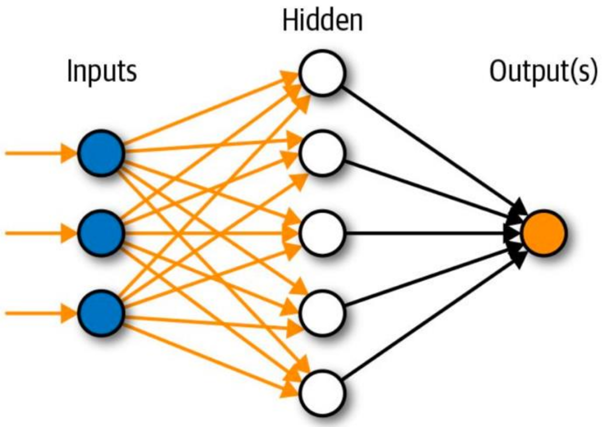
1. **<u>Hacer chico al modelo</u>**. Un modelo con muchos parametros tiene lugar para aprender cada ejemplo sin necesidad de generalizar. Hay que empezar con uno chico, que no ajuste al training set, y empezar a agrandar. Nunca al revés.
2. **<u>Hay que regularizar</u>**. Los modelos, para que sean sencillos y no ajusten caprichosamente a los datos, deben tener pocos pesos, y de tamaños comparables. Una red con un parámetro desproporcionadamente mas grande que el resto probablemente este haciendo contorsiones para ajustar algún conjunto particular de datos. Para esto se penaliza tener muchos parámetros distintos de cero.
3. Hay trucos, como el **<u>dropout</u>** (setear a cero algunos parámetros al azar)

Librerías para redes



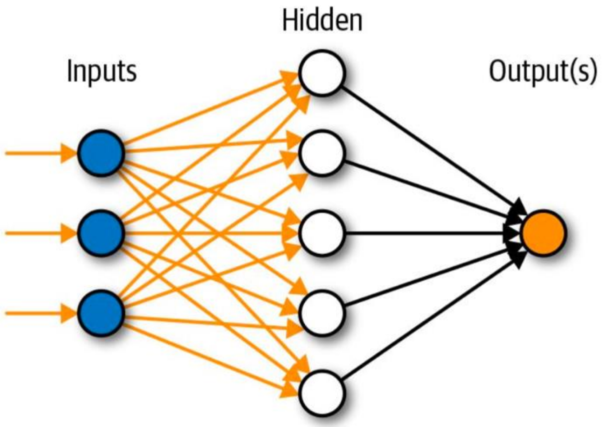
Librerías para redes: Keras

Pasos para entrenar una red neuronal como esta:



Librerías para redes: Keras

Pasos para entrenar una red neuronal como esta:



1) **Crear el Objeto**

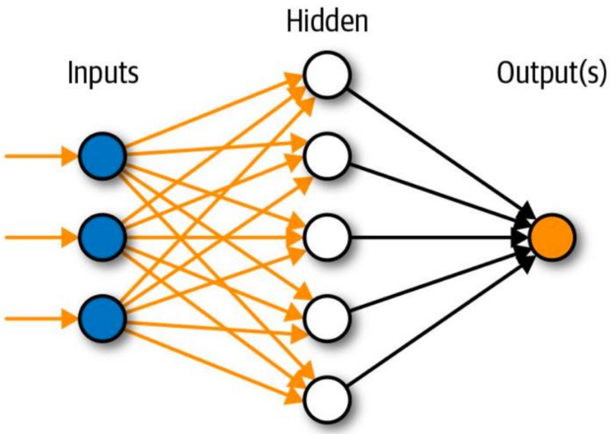
2) **Definir la arquitectura (agregar capas)**

3) **Compilar**

4) **Entrenar**

Librerías para redes: Keras

Pasos para entrenar una red neuronal como esta:



1)
```
from keras.models import Sequential
model = Sequential()
```

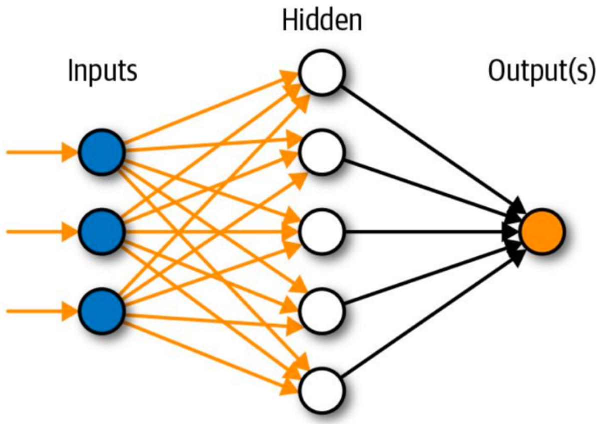
2) **Definir la arquitectura (agregar capas)**

3) **Compilar**

4) **Entrenar**

Librerías para redes: Keras

Pasos para entrenar una red neuronal como esta:



1)
```
from keras.models import Sequential
model = Sequential()
```

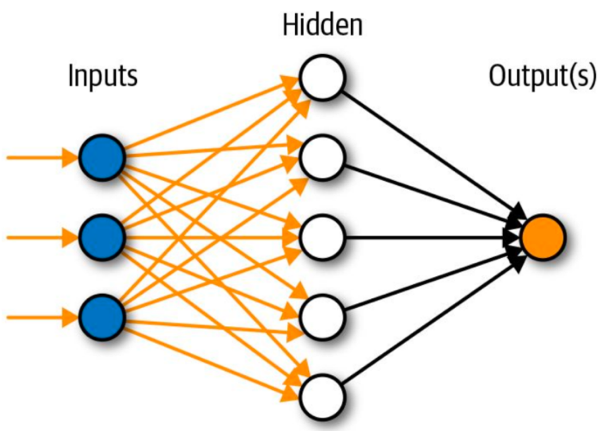
2)
```
model.add(Dense(5), activation = 'sigmoid')
model.add(Dense(1),activation = linear)
```

3) **Compilar**

4) **Entrenar**

Librerías para redes: Keras

Pasos para entrenar una red neuronal como esta:



1)
```
from keras.models import Sequential
model = Sequential()
```

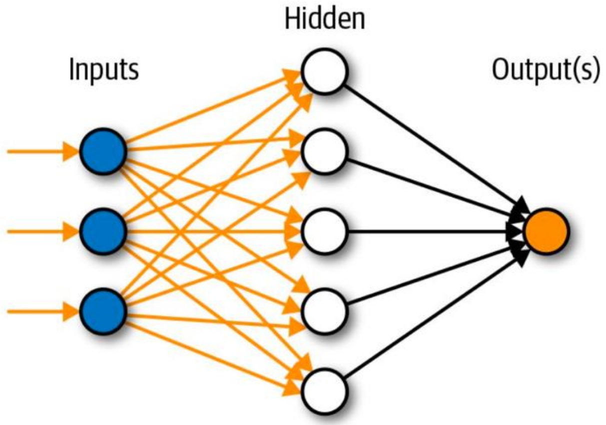
2)
```
model.add(Dense(5), activation = ’sigmoid’)
model.add(Dense(1),activation = linear)
```

3)
```
model.compile(loss = ’mse’, optimizer='adam')
```

4) **Entrenar**

Librerías para redes: Keras

Pasos para entrenar una red neuronal como esta:



1)
```
from keras.models import Sequential
model = Sequential()
```

2)
```
model.add(Dense(5), activation = ’sigmoid’)
model.add(Dense(1),activation = linear)
```

3)
```
model.compile(loss = ’mse’, optimizer='adam')
```

4)
```
model.fit(X,y,batch_size=32, epochs=100)
```

Función de Costo: Definición

Es aquello que buscamos optimizar:

- Costo de una instancia:

$$J_i(\hat{y}_i, y_i) = J_i(f_\Theta(x_i), y_i)$$

Función de Costo: Definición

Es aquello que buscamos optimizar.

- Costo de una instancia:

$$J_i(\hat{y}_i, y_i) = J_i(f_\Theta(x_i), y_i)$$

Predicción del modelo

Valor Real

Depende de los parámetros (pesos de la red)

Función de Costo: Selección

¿Cómo elijo la función de costo J? Hay muchas disponibles.

Clasificación

- Binary Cross Entropy
- Categorical Cross-entropy
- Poisson Loss
- Custom

Regresión

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- MAPE
- Custom

https://neptune.ai/blog/keras-loss-functions

https://keras.io/api/losses/

Función de Costo: Selección

¿Cómo elijo la función de costo J? Hay muchas disponibles.

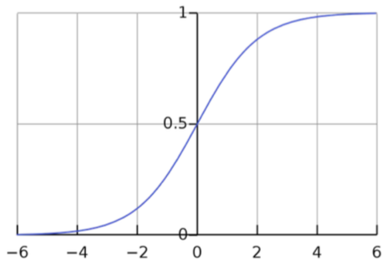
Vamos a ver 3 escenarios:

- **Escenario 1: Clasificación Binaria**
- **Escenario 2: Clasificación Multiclase**
- **Escenario 3: Regresión**

Escenario 1: Clasificación Binaria

- **Activacion: Sigmoide**

$$h(t) = \frac{1}{1+e^{-t}}$$



- **Función de costo: Binary Cross-Entropy**

$$J(\Theta) = \frac{1}{N}\sum_i -y_i \cdot log\left(f_\Theta(x_i)\right) - (1-y_i) \cdot log\left(1 - f_\Theta(x_i)\right)$$

Escenario 1: Clasificación Binaria

- **Activacion:** Sigmoide

```
model.add(layers.Dense(1, activation='sigmoid'))
```

- **Función de costo:** Binary Cross-Entropy

```
model.compile(optimizer="Adam",
              loss=tf.keras.losses.BinaryCrossentropy())
```

Escenario 2: Clasificación Multiclase

- **Activacion:** SoftMax

$$h(t)_j = \frac{e^{t_j}}{\sum_{k=1}^{K} e^{t_j}}$$

Generalización de la función logística (Normalizada)

- **Función de costo:** Categorical Cross-Entropy

$$J(\Theta) = -\frac{1}{N} \sum_i \sum_k y_i \cdot log\left(f_\Theta\left(x_i\right)\right)$$

Generalización de la binaria (k clases)

Escenario 2: Clasificación Multiclase

- **Activacion:** SoftMax

```
model.add(layers.Dense(num_clases, activation='softmax'))
```

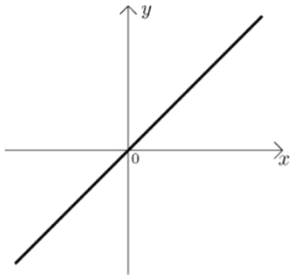
- **Función de costo:** Categorical Cross-Entropy

```
model.compile(optimizer="Adam",
              loss=tf.keras.losses.categorical_crossentropy())
```

Escenario 3: Regresión

- **Activacion: Lineal**

$$h(t) = t$$



- **Función de costo: Mean Squared Error**

$$J(\Theta) = \frac{1}{N} \sum_i (f_\Theta(x_i) - y_i)^2$$

Escenario 3: Regresión

- **Activacion: Lineal**

```
model.add(layers.Dense(1, activation='linear'))
```

- **Función de costo: Mean Squared Error**

```
model.compile(optimizer="Adam", loss='mse')
```