Basics of Neural Networks

Xabier Pérez Couto
Universidade da Coruña/CITIC
xabier.perez.couto@udc.gal

cost
EUROPEAN COOPERATION IN SCIENCE & TECHNOLOGY

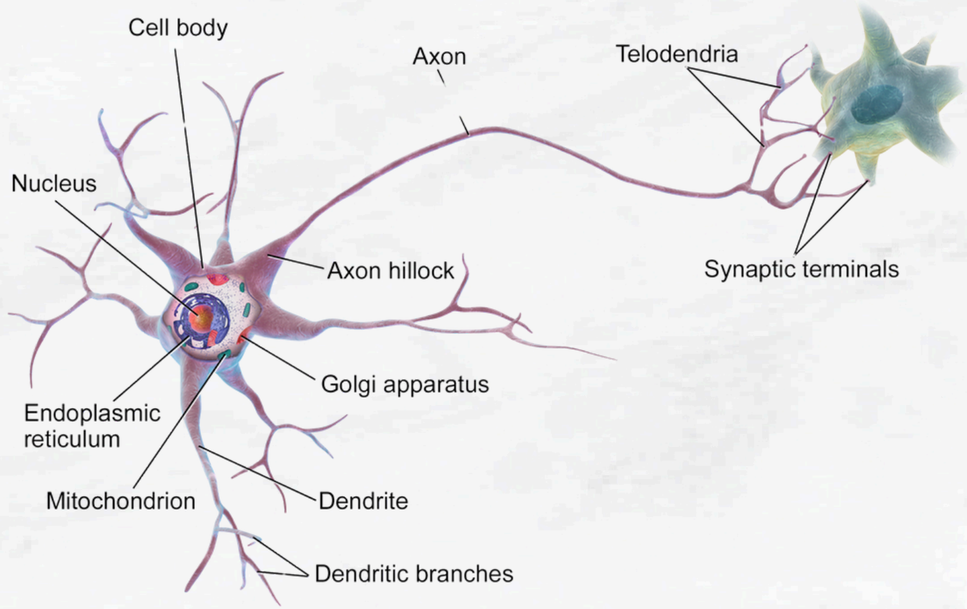
NanoSpace

Funded by the European Union

Artificial Neural Network (ANN)

Machine learning model inspired by the networks of biological neurons found in our brain

First time introduced by McCulloch & Pitts (1943): *“A Logical Calculus of the Ideas Immanent in Nervous Activity“*

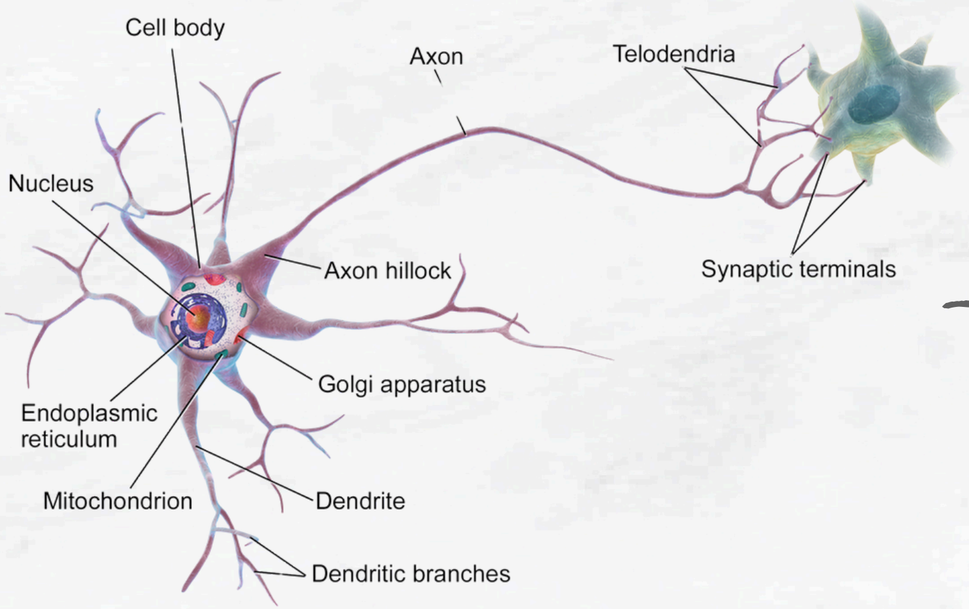


Credit: Bruce Blaus (Creative Commons 3.0)

Artificial Neural Network (ANN)

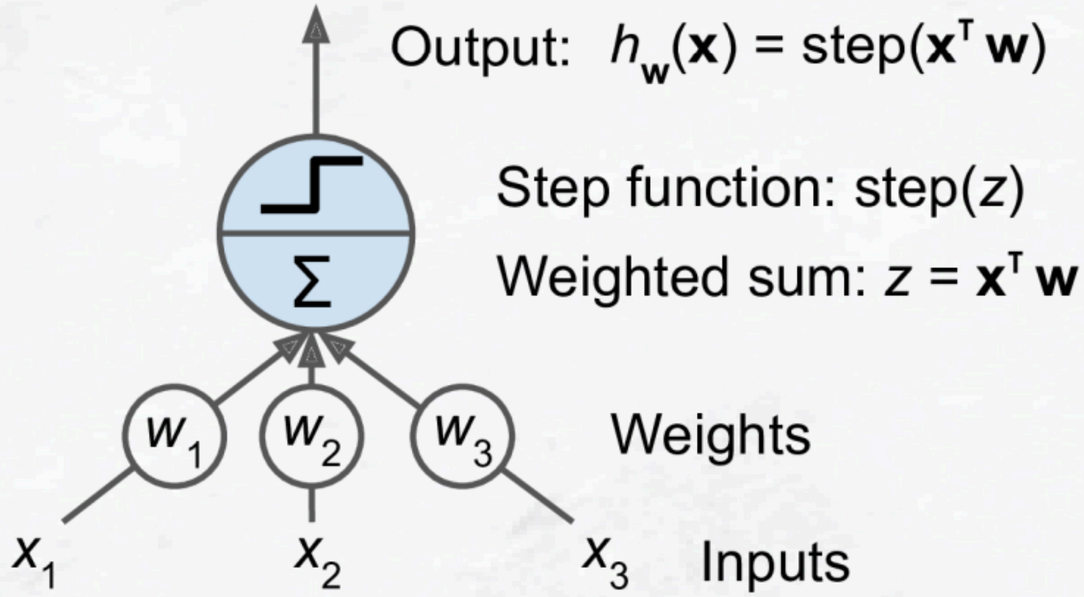
Machine learning model inspired by the networks of biological neurons found in our brain

First time introduced by McCulloch & Pitts (1943): *“A Logical Calculus of the Ideas Immanent in Nervous Activity“*



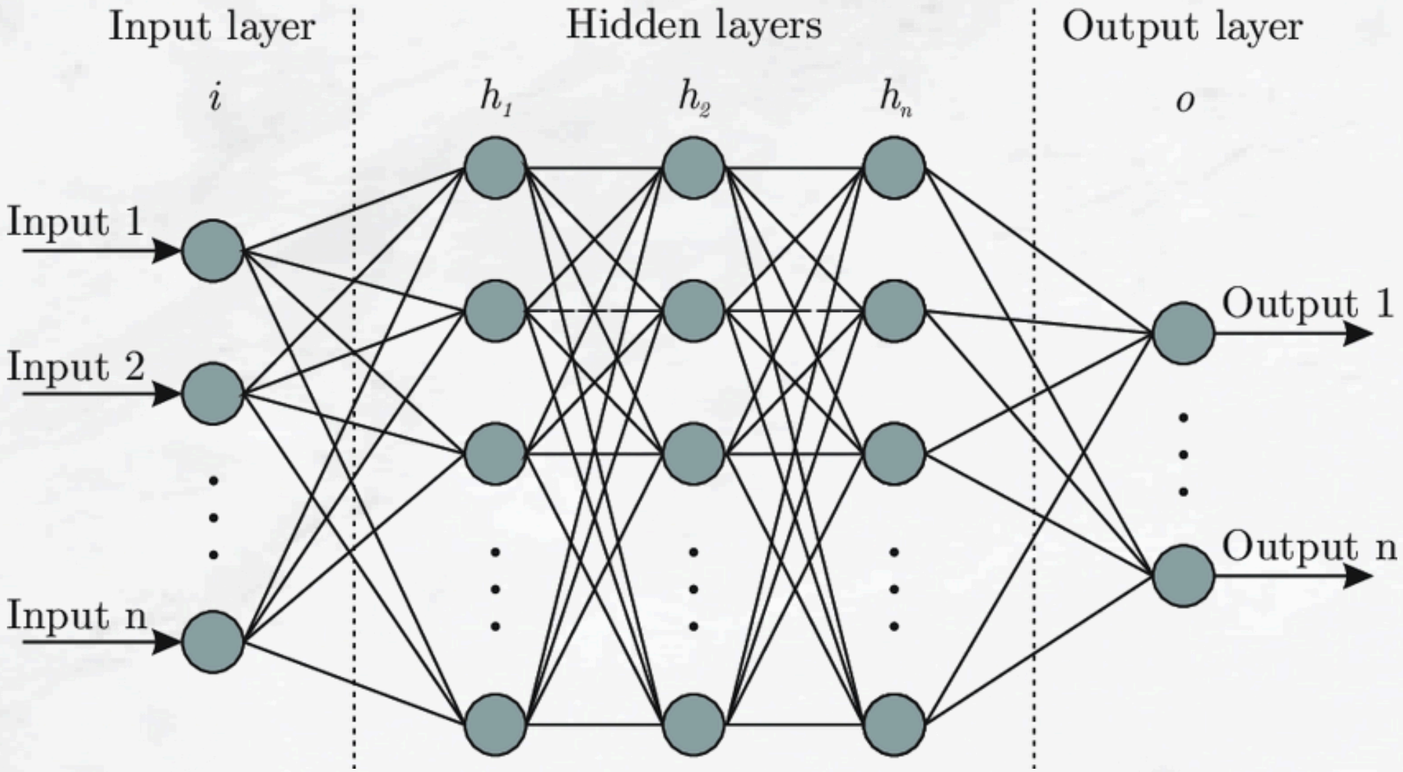
Credit: Bruce Blaus (Creative Commons 3.0)

The Perceptron



Credit: Aurélien Géron

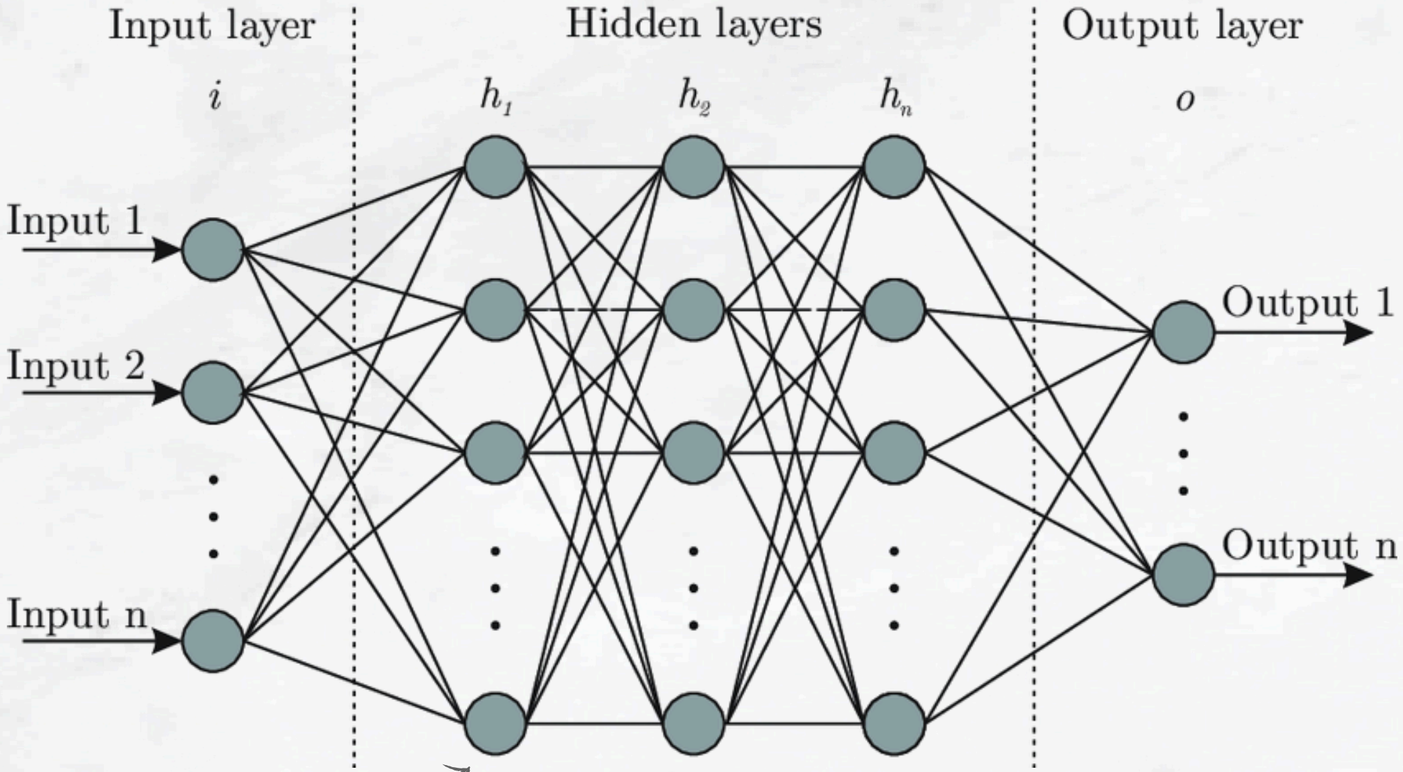
Multilayer Perceptron (MLP)



Credit: Ramón y Cajal, S. (1899)

Multilayer Perceptron (MLP)

Credit: Ramón y Cajal, S. (1899)

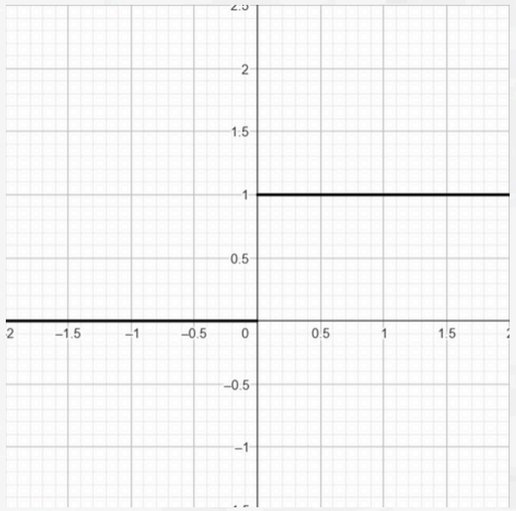


$$h_i = \phi\left(X_i W_i + b_i\right)$$

- X_i : *Matrix with the inputs of layer i*
- W_i : *Matrix with the weights of layer i*
- b_i : *Array with the bias terms of layer i*
- ϕ : ***Activation function = nonlinear function (step function, logistic, tanh…)***

Activation functions (ϕ)

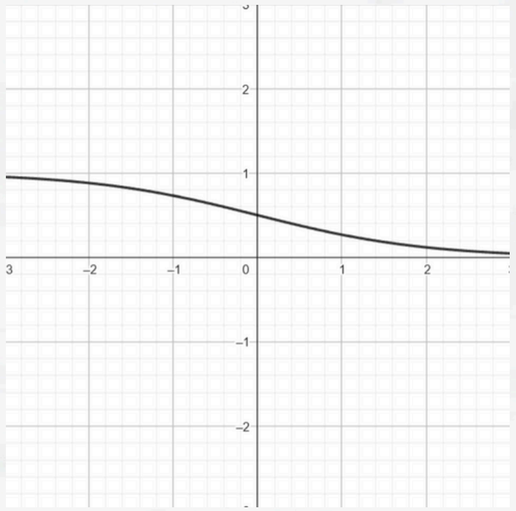
Binary



$$\phi(x) = \begin{cases} 1, & \text{if } x \geqslant 0 \\ 0, & \text{if } x < 0 \end{cases}$$

- Biologically inspired
- High abstraction
- Easy to compute
- Zero gradient in the domain

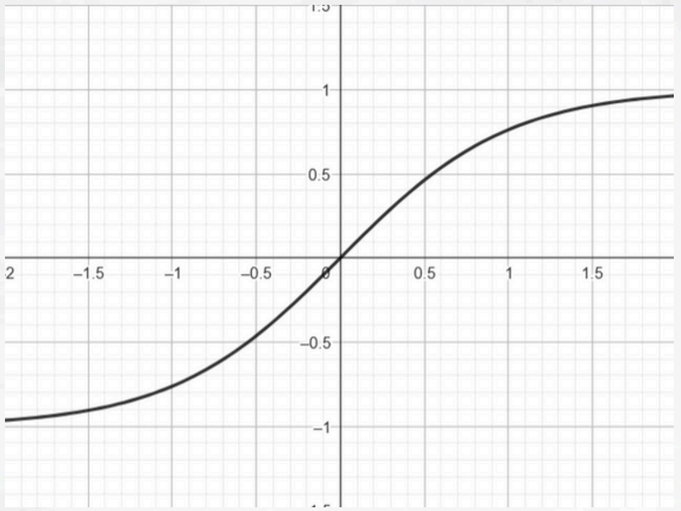
Logistic/Sigmoid



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

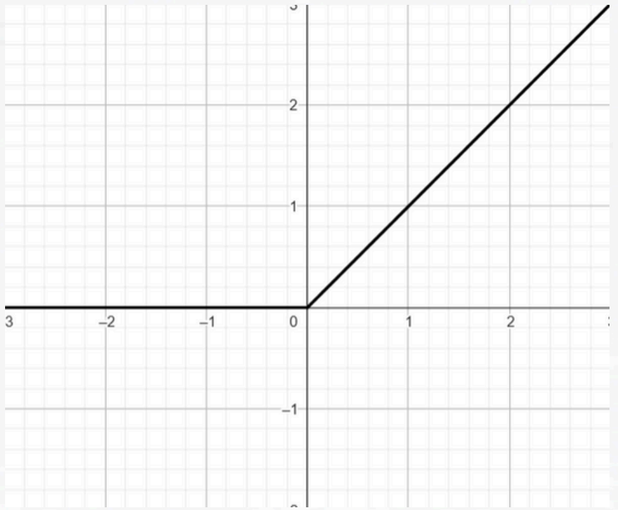
- No zero gradient
- But the gradient tend to be zero in -inf and -inf
- Computationally expensive

Hiperbolic Tangent (tanh)



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

Rectified Linear Unit (ReLU)

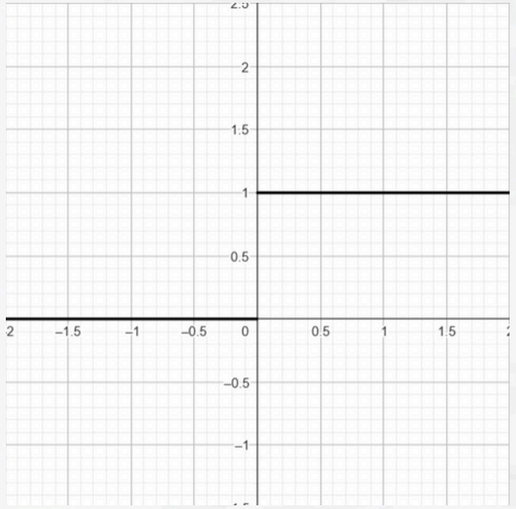


$$\phi(x) = \begin{cases} x, & \text{if } x \geqslant 0 \\ 0, & \text{if } x < 0 \end{cases}$$

- Usable gradient
- High abstraction
- Easy to compute
- Variations: GeLU, Leaky ReLU, ...

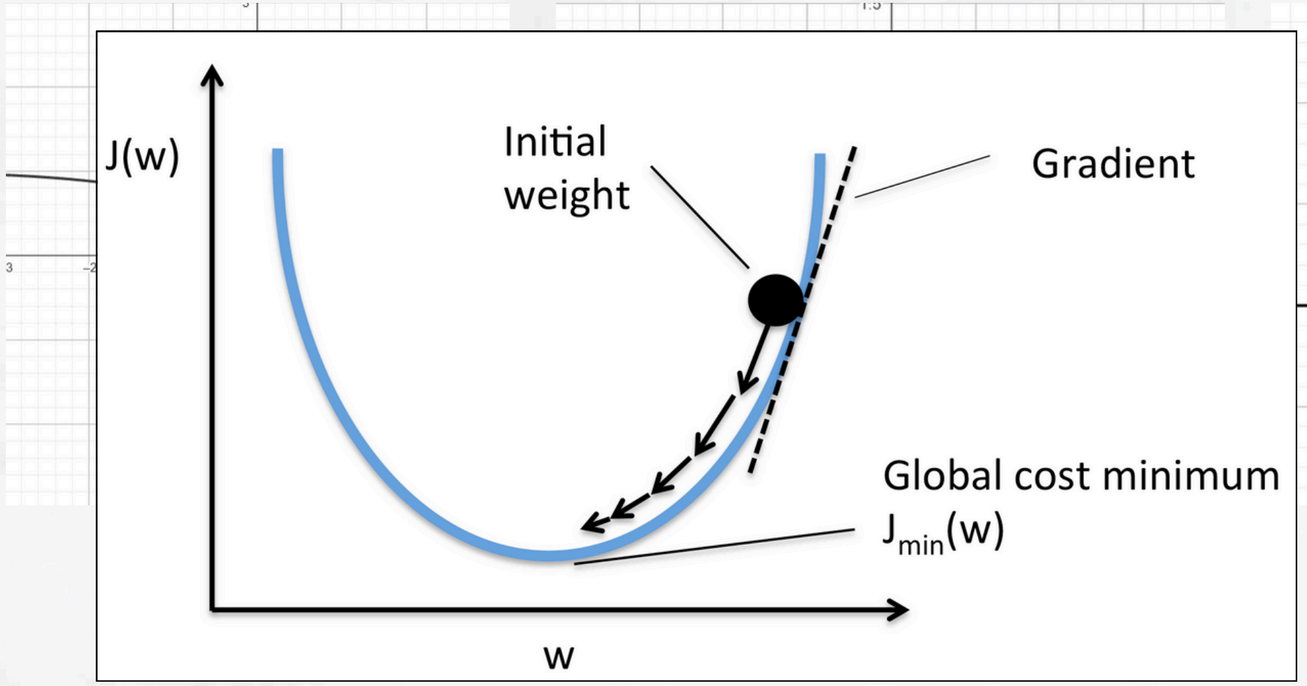
Activation functions (ϕ)

Binary



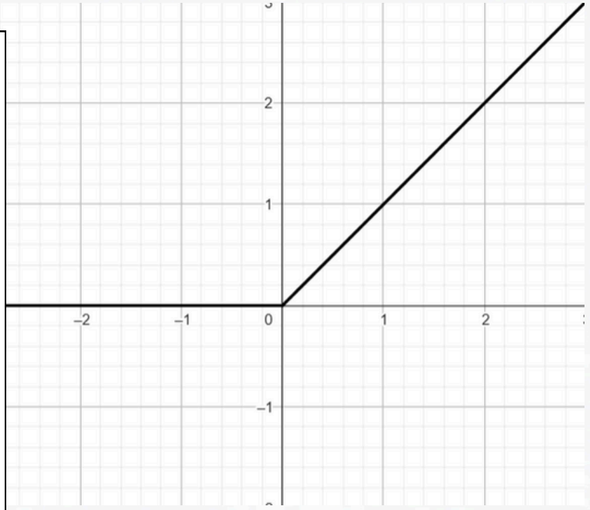
$$\phi(x) = \begin{cases} 1, & \text{if } x \geqslant 0 \\ 0, & \text{if } x < 0 \end{cases}$$

- Biologically inspired
- High abstraction
- Easy to compute
- Zero gradient in the domain



- No zero gradient
- But the gradient tend to be zero in -inf and -inf
- Computationally expensive

Rectified Linear Unit (ReLU)

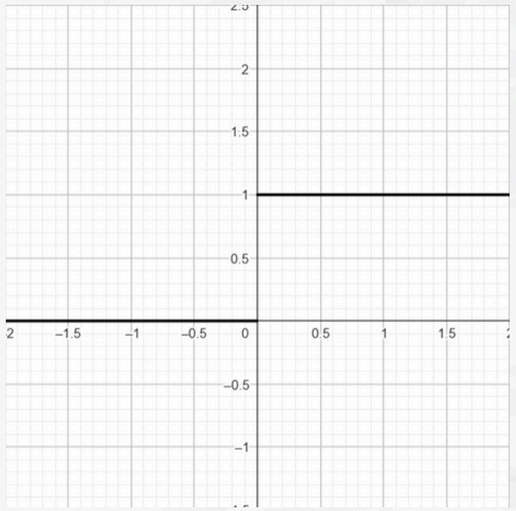


$$\phi(x) = \begin{cases} x, & \text{if } x \geqslant 0 \\ 0, & \text{if } x < 0 \end{cases}$$

- Usable gradient
- High abstraction
- Easy to compute
- Variations: GeLU, Leaky ReLU, ...

Activation functions (ϕ)

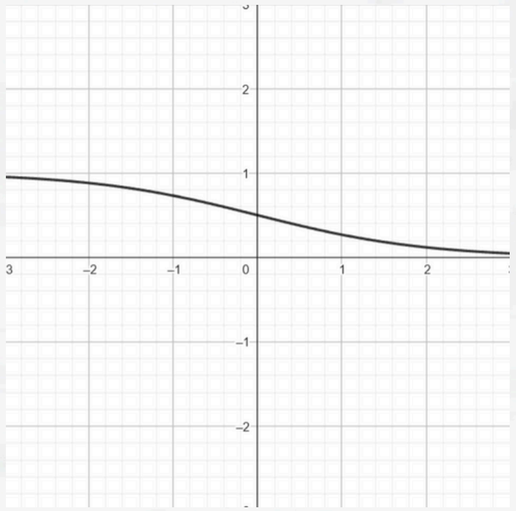
Binary



$$\phi(x) = \begin{cases} 1, & \text{if } x \geqslant 0 \\ 0, & \text{if } x < 0 \end{cases}$$

- Biologically inspired
- High abstraction
- Easy to compute
- Zero gradient in the domain

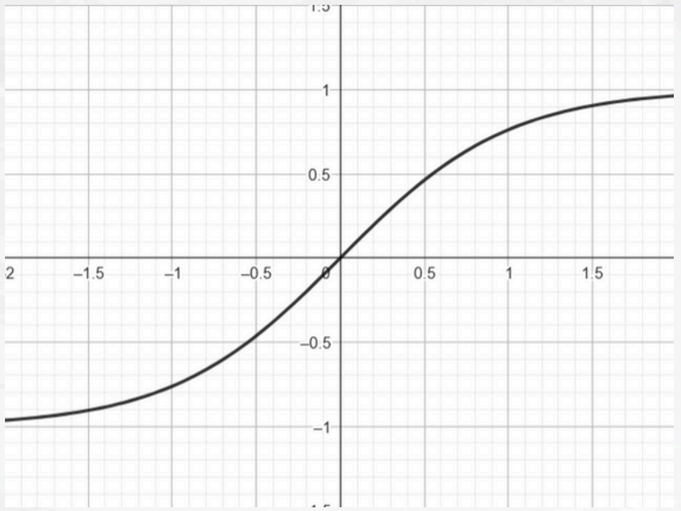
Logistic/Sigmoid



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

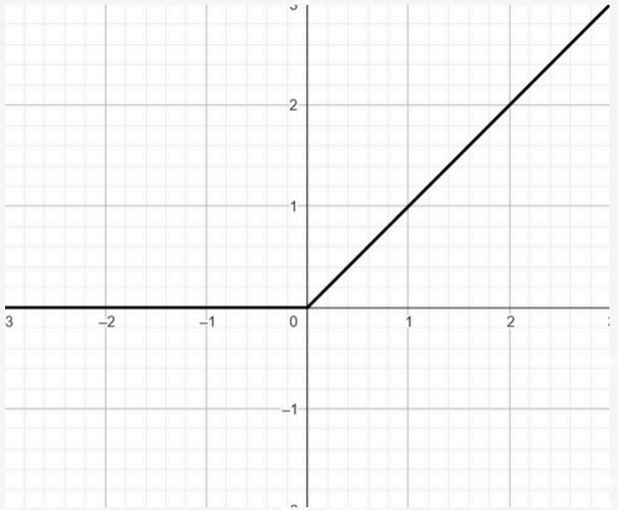
- No zero gradient
- But the gradient tend to be zero in -inf and -inf
- Computationally expensive

Hiperbolic Tangent (tanh)



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

Rectified Linear Unit (ReLU)

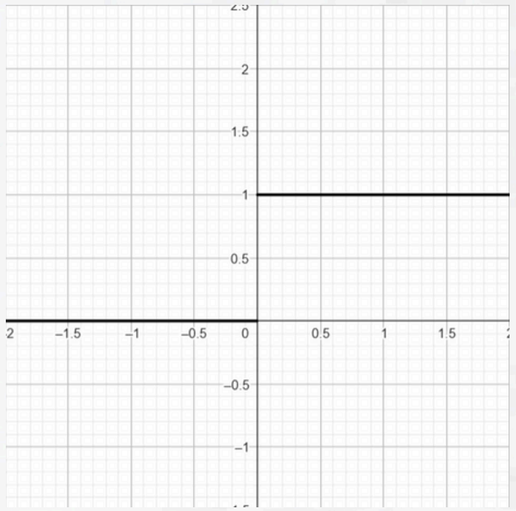


$$\phi(x) = \begin{cases} x, & \text{if } x \geqslant 0 \\ 0, & \text{if } x < 0 \end{cases}$$

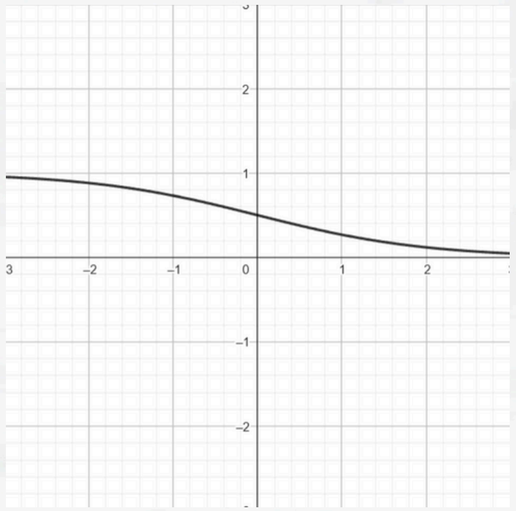
- Usable gradient
- High abstraction
- Easy to compute
- Variations: GeLU, Leaky ReLU, ...

Activation functions (ϕ)

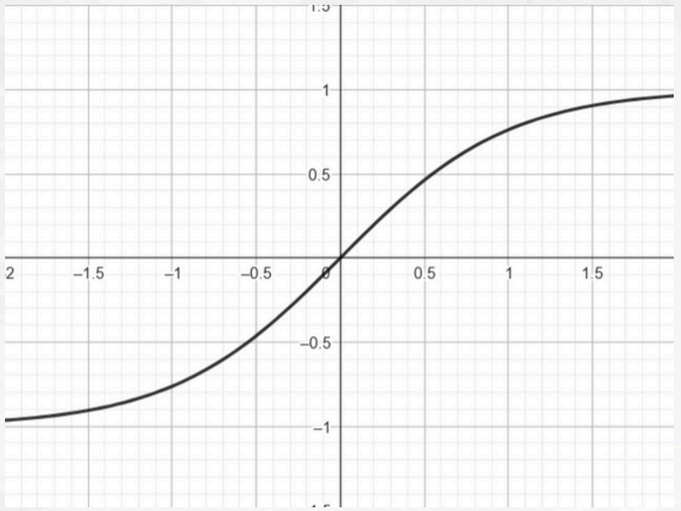
Binary



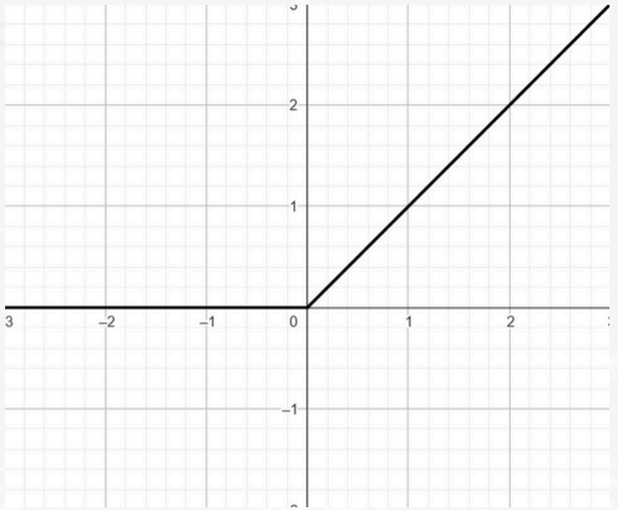
Logistic/Sigmoid



Hiperbolic Tangent (tanh)



Rectified Linear Unit (ReLU)



Note: the activation function of the output layer depends on the type of problem we are dealing with.

Regression MLPs → ReLU (positive outputs), Logistic/Sigmoid or Tanh (bounded outputs), or None (linear output layer for any value)

Classification MLPs → Logistic/Sigmoid for binary classification, and softmax plus for multilabel classification

The Backpropagation algorithm

However, DNNs went through a sad long winter before anyone knew how to train them :(

Fortunately, in 1986 Rumelhart, Hinton and Williams published a groundbreaking paper introducing the *backpropagation algorithm*

The Backpropagation algorithm

However, DNNs went through a sad long winter before anyone knew how to train them :(
Fortunately, in 1986 Rumelhart, Hinton and Williams published a groundbreaking paper introducing the *backpropagation algorithm*

Main idea

1) Initializing the weights and biases randomly
2) Making a prediction $\mathbf{Y_{pred}}$ with inputs **X** (the *forward propagation*)
3) Computing the loss against the ground-truth values $\mathbf{Y_{true}}$,
   e.g. the Mean Squared Error (MSE):

$$L(w,b) = \frac{1}{n}\sum_{k=1}^{n}\left(Y_{true}^{(k)} - Y_{pred}^{(k)}\right)^2$$

4) Updating weights and biases through gradient descent:

$$w_{i+1} = w_i - \alpha\frac{\partial L}{\partial w_i}$$

$$b_{i+1} = b_i - \alpha\frac{\partial L}{\partial w_i}$$

The Backpropagation algorithm

$$w_{i+1} = w_i - \alpha \frac{\partial L}{\partial w_i}$$

We need to compute this, from this:

$$\begin{cases} h_i = \phi(a_i) \\ a_i = w_i X_i + b_i \end{cases}$$

Chain's Rule

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_i}$$

The Backpropagation algorithm

$$w_{i+1} = w_i - \alpha \frac{\partial L}{\partial w_i}$$

We need to compute this, from this:

$$\begin{cases} h_i = \phi(a_i) \\ a_i = w_i X_i + b_i \end{cases}$$

Chain's Rule

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_i}$$

$$\frac{\partial a_i}{\partial w_i} = X_i = a_{i-1}$$

The Backpropagation algorithm

$$w_{i+1} = w_i - \alpha \frac{\partial L}{\partial w_i}$$ ← We need to compute this, from this:

$$\begin{cases} h_i = \phi(a_i) \\ a_i = w_i X_i + b_i \end{cases}$$

Chain's Rule →

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_i}$$

$$\frac{\partial a_i}{\partial w_i} = X_i = a_{i-1}$$

$$\frac{\partial h_i}{\partial a_i} = \phi'(a_i)$$

For ReLU:

$$\phi'(a_i) = \begin{cases} 1, \text{ if } a_i > 0 \\ 0, \text{ if } a_i \leqslant 0 \end{cases}$$

The Backpropagation algorithm

$$w_{i+1} = w_i - \alpha \frac{\partial L}{\partial w_i}$$ ← We need to compute this, from this:

$$\begin{cases} h_i = \phi(a_i) \\ a_i = w_i X_i + b_i \end{cases}$$

Chain's Rule →

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_i}$$

$$\frac{\partial a_i}{\partial w_i} = X_i = a_{i-1}$$

$$\frac{\partial h_i}{\partial a_i} = \phi'(a_i)$$

For ReLU:

$$\phi'(a_i) = \begin{cases} 1, \text{ if } a_i > 0 \\ 0, \text{ if } a_i \leqslant 0 \end{cases}$$

$$\frac{\partial L}{\partial h_i} = \frac{\partial a_{i+1}}{\partial h_i} \frac{\partial L}{\partial a_{i+1}}$$

$$= w_{i+1} \frac{\partial L}{\partial a_{i+1}}$$

$$= w_{i+1} \frac{\partial L}{\partial h_{i+1}} \frac{\partial h_{i+1}}{\partial a_{i+1}}$$

The Backpropagation algorithm

$$w_{i+1} = w_i - \alpha \frac{\partial L}{\partial w_i}$$

We need to compute this, from this:

$$\begin{cases} h_i = \phi(a_i) \\ a_i = w_i X_i + b_i \end{cases}$$

Chain's Rule

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_i}$$

$$\frac{\partial a_i}{\partial w_i} = X_i = a_{i-1}$$

$$\frac{\partial h_i}{\partial a_i} = \phi'(a_i)$$

For ReLU:

$$\phi'(a_i) = \begin{cases} 1, \text{ if } a_i > 0 \\ 0, \text{ if } a_i \leqslant 0 \end{cases}$$

$$\frac{\partial L}{\partial h_i} = \frac{\partial a_{i+1}}{\partial h_i} \frac{\partial L}{\partial a_{i+1}}$$

$$= w_{i+1} \frac{\partial L}{\partial a_{i+1}}$$

Computed in the previous step!

$$= w_{i+1} \frac{\partial L}{\partial h_{i+1}} \boxed{\frac{\partial h_{i+1}}{\partial a_{i+1}}}$$

The Backpropagation algorithm

$$w_{i+1} = w_i - \alpha \frac{\partial L}{\partial w_i}$$ ← We need to compute this, from this:

$$\begin{cases} h_i = \phi(a_i) \\ a_i = w_i X_i + b_i \end{cases}$$

Chain's Rule →

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_i}$$

$$\frac{\partial a_i}{\partial w_i} = X_i = a_{i-1}$$

$$\frac{\partial h_i}{\partial a_i} = \phi'(a_i)$$

For ReLU:

$$\phi'(a_i) = \begin{cases} 1, \text{ if } a_i > 0 \\ 0, \text{ if } a_i \leqslant 0 \end{cases}$$

$$\frac{\partial L}{\partial h_i} = \frac{\partial a_{i+1}}{\partial h_i} \frac{\partial L}{\partial a_{i+1}}$$

$$= w_{i+1} \frac{\partial L}{\partial a_{i+1}}$$

$$= w_{i+1} \frac{\partial L}{\partial h_{i+1}} \frac{\partial h_{i+1}}{\partial a_{i+1}}$$

In the last hidden layer this is just the derivative of the loss function!

Some additional considerations

Scale matters!

Using features with different scales (e.g. some within 0-1 and others within 0-10000) can cause the training to be slow, unstable or even to explode and diverge.

Fast trick: Standardization, i.e. subtract mean, divide by standard deviation

Some additional considerations

Scale matters!

Using features with different scales (e.g. some within 0-1 and others within 0-10000) can cause the training to be slow, unstable or even to explode and diverge.

Fast trick: Standardization, i.e. subtract mean, divide by standard deviation

Divide and conquer

Another must for a good DNN is to split the dataset in a *training set* (~80%; to train the network), a *validation set* (~10%; to choose the best hyperparameters), and the *test set* (~10%; to evaluate the real performance).

Some additional considerations

Scale matters!

Using features with different scales (e.g. some within 0-1 and others within 0-10000) can cause the training to be slow, unstable or even to explode and diverge.

Fast trick: Standardization, i.e. subtract mean, divide by standard deviation

Divide and conquer

Another must for a good DNN is to split the dataset in a *training set* (~80%; to train the network), a *validation set* (~10%; to choose the best hyperparameters), and the *test set* (~10%; to evaluate the real performance).

Try out different architectures, the Sky is the limit! (but care about overfitting)

- *For* most applications a shallow neural network (2-3 hidden layers) is more than sufficient.
- It usually works well to use more neurons in the first layers (to learn low-level features) and progressively less neurons in the following ones.
- *Play it yourself!* → **playground.tensorflow.org**