Deep Learning

Neural network (very simplified 1/2)

- NN = h, a large function with numerous parameters (weights θ)

Think about a polynomial function …

$$f(x) = ax + bx^2 + cx^3 + dx^4 \ldots$$

$$y = h(x, \theta)$$

Neural network (very simplified 1/2)

- NN = h, a large function with numerous parameters (weights θ)
- Input: x, as a vector (matrix)
  - Columns: dimension
    - Example: tumors properties (size, genes, etc)
    - Example: 128x128 HU values (a CT slice)
  - Row: samples
- Output: y as a vector (matrix)
  - Example: one value, probability to be maligne/benigne
  - Example: 128x128 values, a segmented image
  - One row per input sample

$$y = h(x, \theta)$$

Neural network (very simplified 2/2)

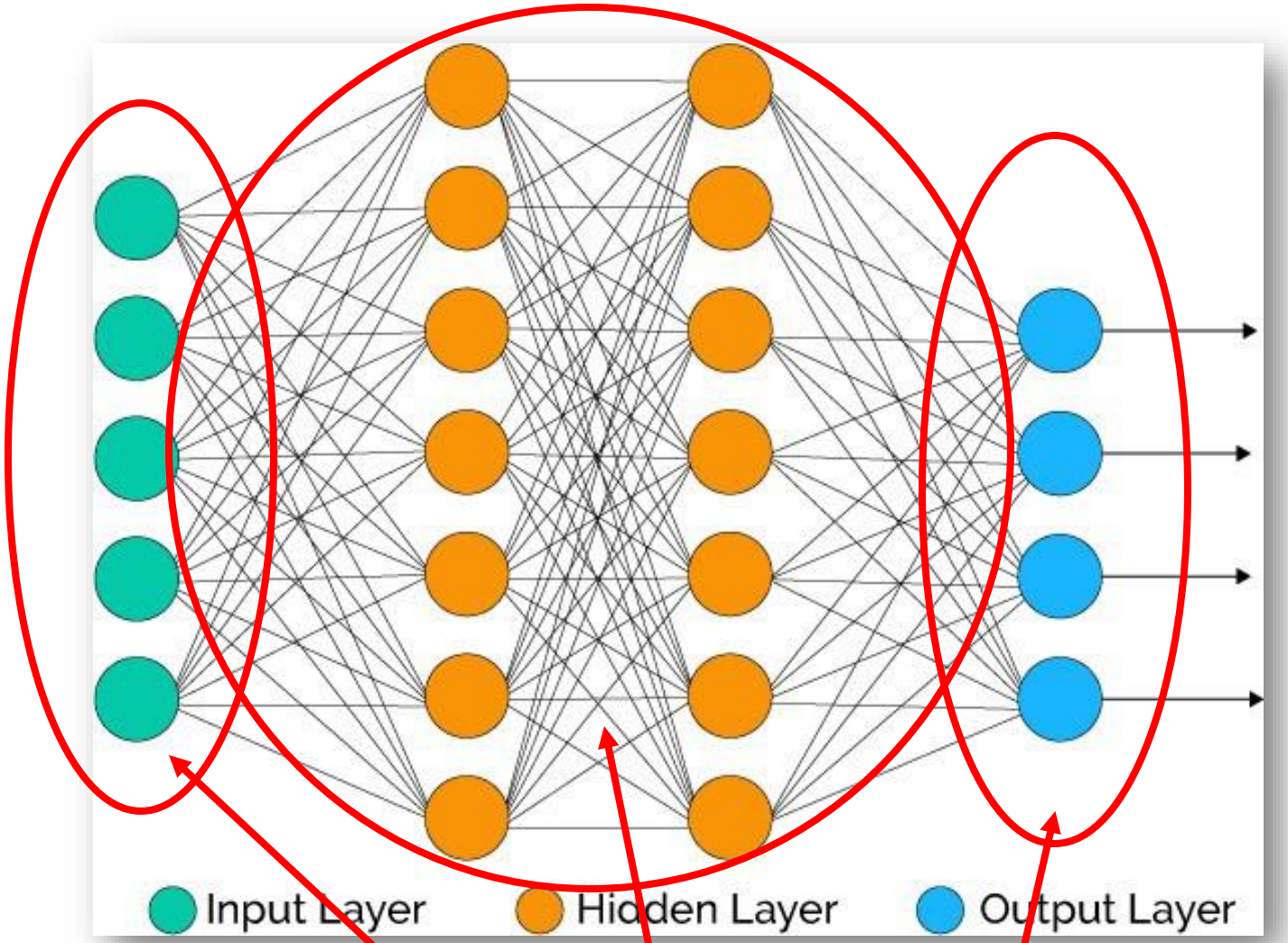
- **Training a NN** = find the values of the parameters θ
  - Input: a large database with **both x and y**
  - Loss function: a function to estimate the predicting error bw real samples from the db and estimated ones from h
  - Optimisation: an iterative algorithm to find theta

$$y = h(x, \theta)$$

- Long process, need GPU
- To be performed only once
- The results is h, a set of parameters values
- Can be used with any x, to guess y

$$Loss(y, \hat{y})$$

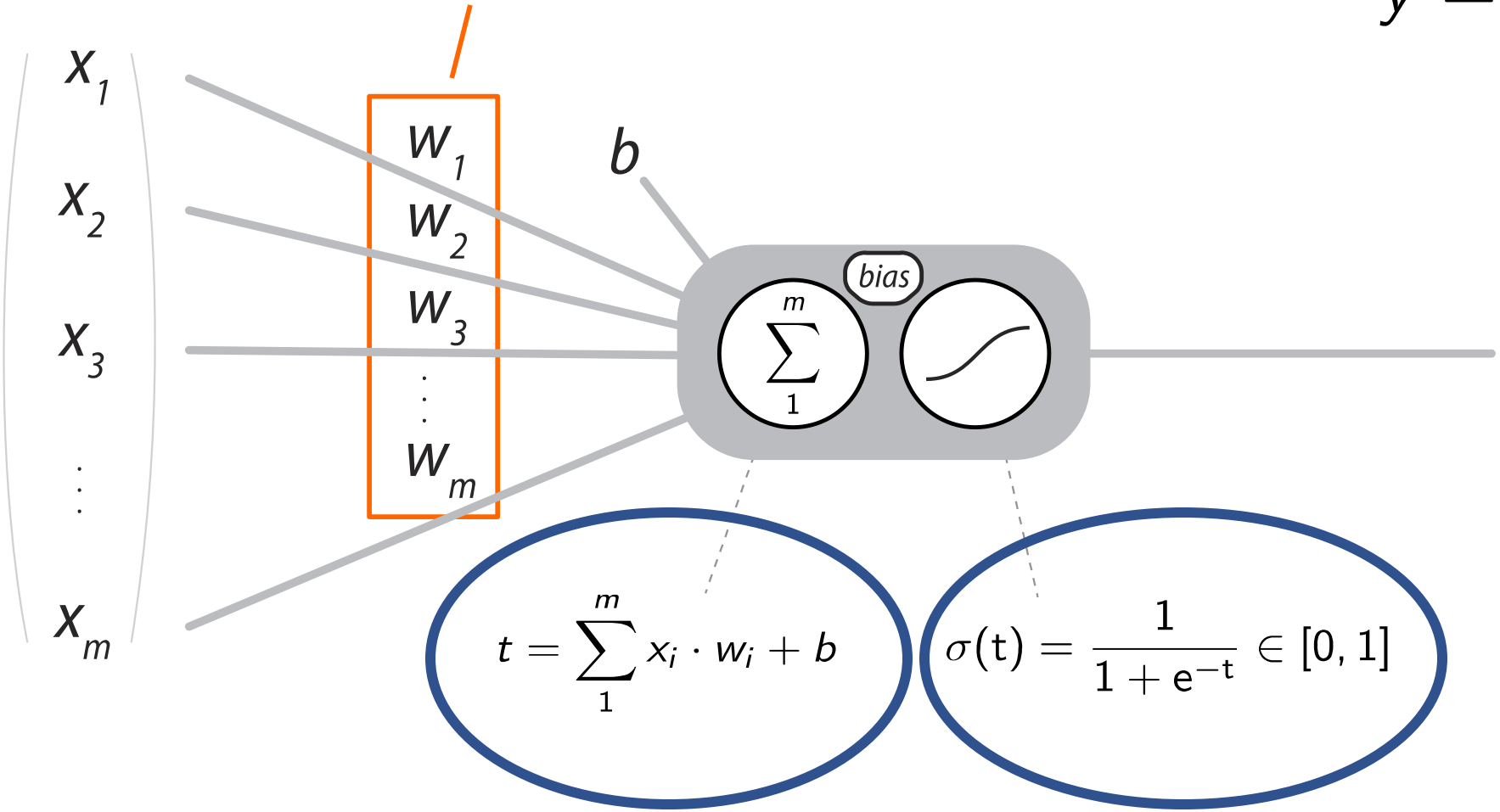
Known y Predicted y



$$h(x, \theta) = y$$

Determined by the minimisation of a cost function J(Θ)

$$\hat{y} = \sigma(\Theta^T \cdot X + b)$$



and $\bar{y} = \begin{cases} 0 & \text{if } \hat{y} < 0.5 \\ 1 & \text{if } \hat{y} \geq 0.5 \end{cases}$

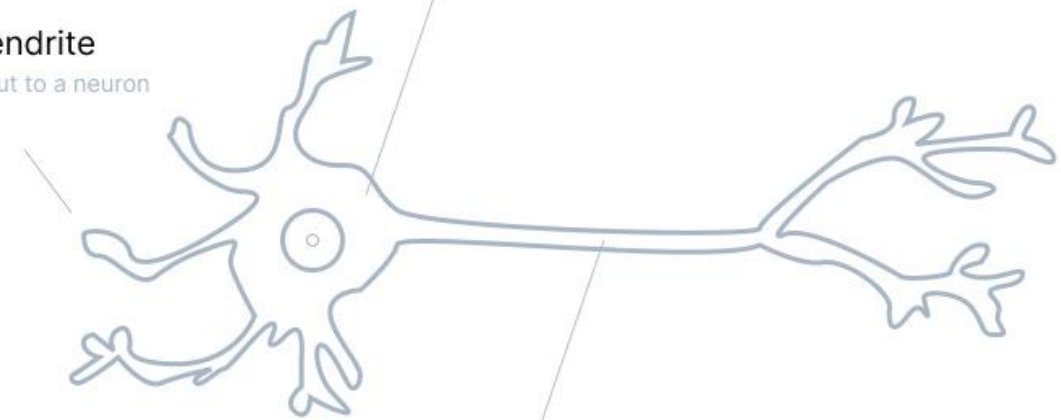
Input	Bias / Weight	Activation function	Output
X	Θ	$\sigma(t)$	$\hat{y}$

Cell body

Information is segregated into "useful" and "not useful" categories.

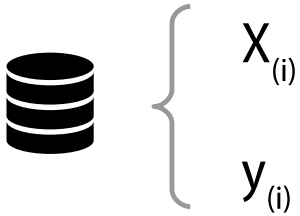
Dendrite

Input to a neuron



Axon

Output to the neuron



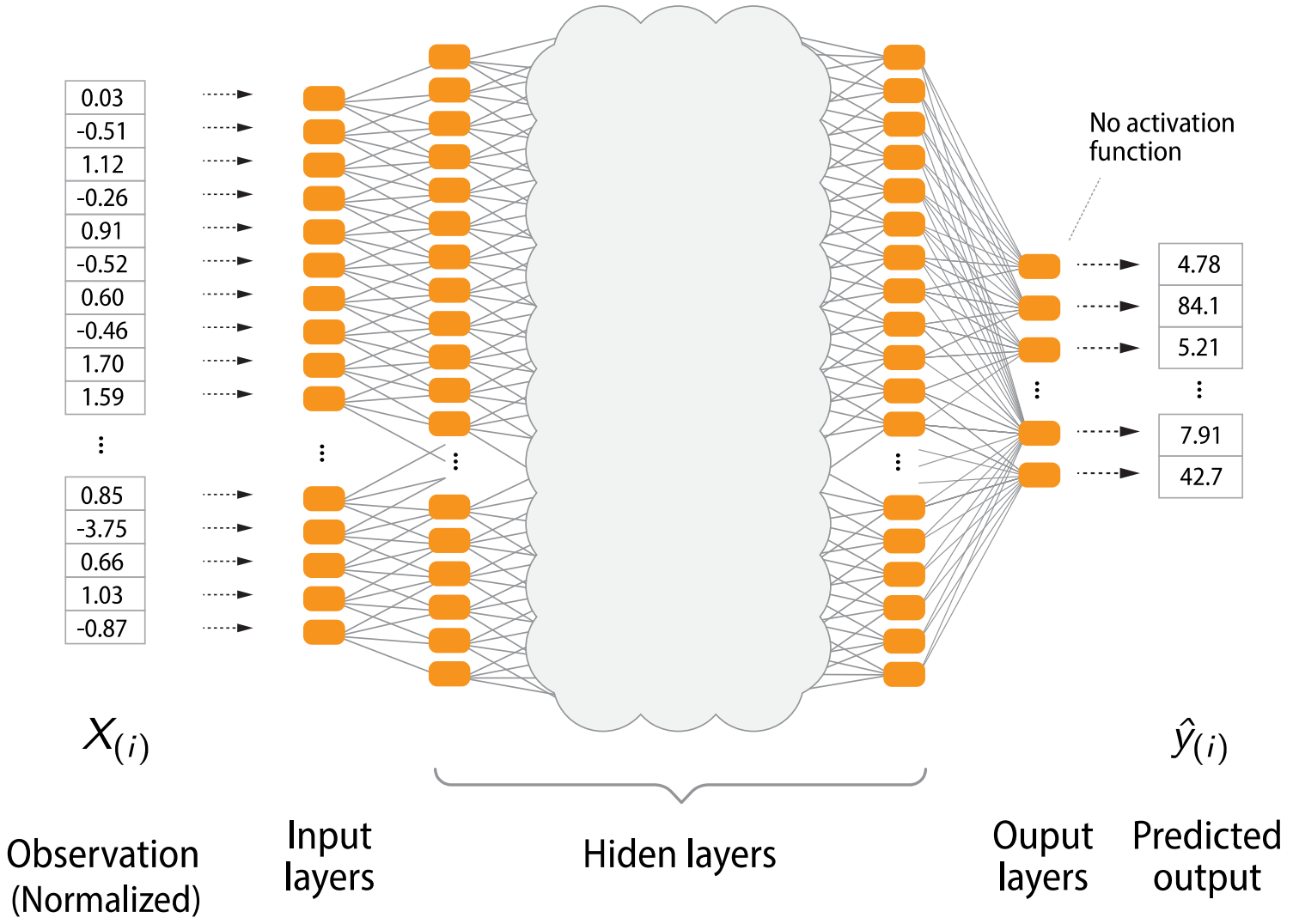
$X_{(i)}$

$y_{(i)}$

Train set

$X_{(i)}$: Observations

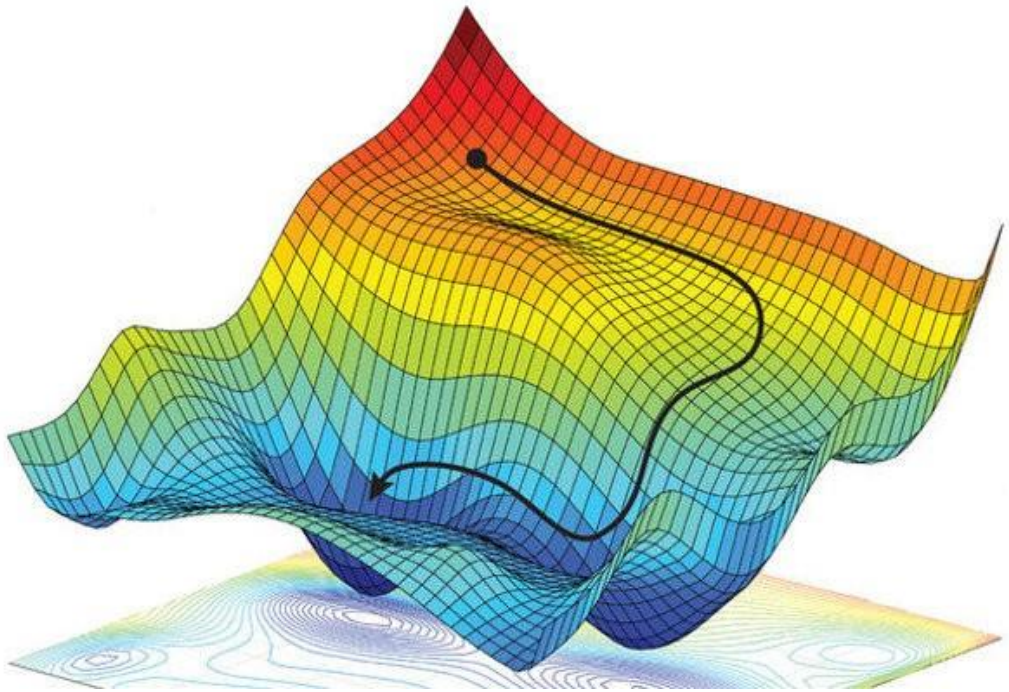
$y_{(i)}$: Expected output

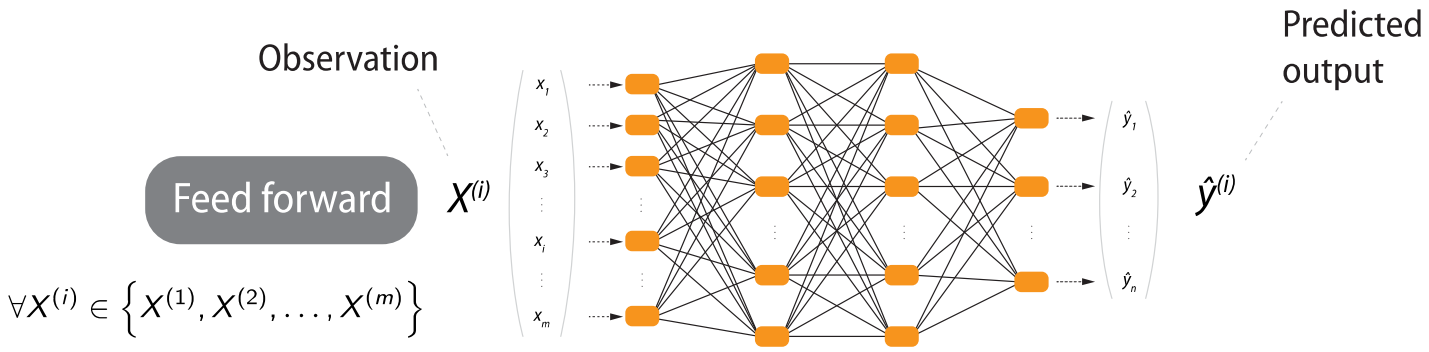


Find the parameters values

Optimisation = found the min

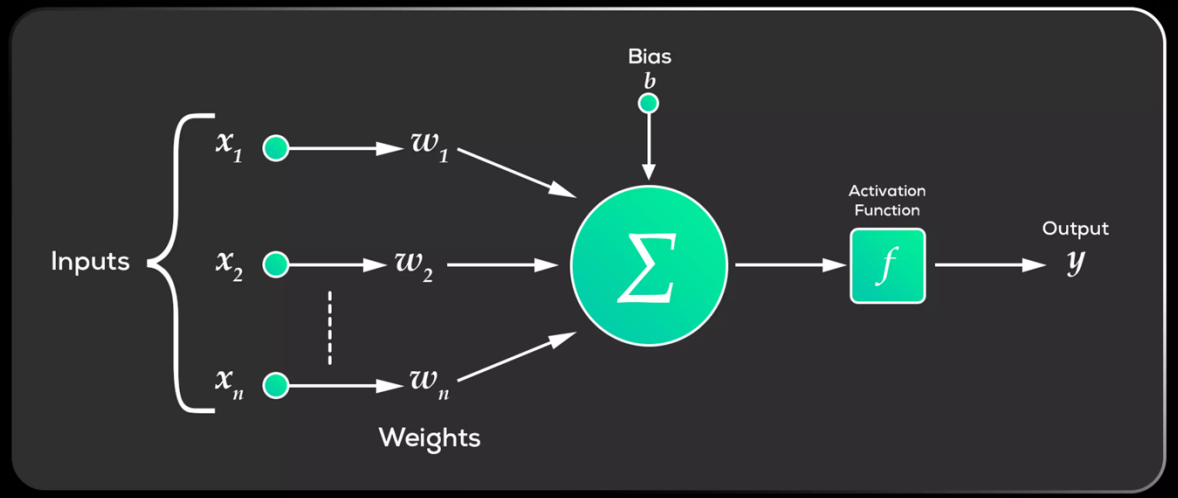
- Parameters = research space dim
- Loss function = diff x and y
- Iterative, number of *epoch*
- Direction = loss gradient
- Learning rate = convergence step





Back-propagation

Learning process



Neural Network Activation Functions: a small subset!

ReLU $\max(0, x)$	**GELU** $\frac{x}{2}\left(1+\tanh\left(\sqrt{\frac{2}{\pi}}\left(x+ax^3\right)\right)\right)$	**PReLU** $\max(0, x)$
ELU $\begin{cases} x \text{ if } x > 0 \\ \alpha(x \exp x - 1) \text{ if } x < 0 \end{cases}$	**Swish** $\frac{x}{1+\exp -x}$	**SELU** $\alpha(\max(0, x)+ \min(0, \beta(\exp x - 1)))$
SoftPlus $\frac{1}{\beta}\log\left(1+\exp(\beta x)\right)$	**Mish** $x\tanh\left(\frac{1}{\beta}\log\left(1+\exp(\beta x)\right)\right)$	**RReLU** $\begin{cases} x \text{ if } x \geq 0 \\ ax \text{ if } x < 0 \text{ with } a \sim \Re(l, u) \end{cases}$
HardSwish $\begin{cases} 0 \text{ if } x \leq -3 \\ x \text{ if } x \geq 3 \\ x(x+3)/6 \text{ otherwise} \end{cases}$	**Sigmoid** $\frac{1}{1+\exp(-x)}$	**SoftSign** $\frac{x}{1+\lvert x\rvert}$
Tanh $\tanh(x)$	**Hard tanh** $\begin{cases} a \text{ if } x \geq a \\ b \text{ if } x \leq b \\ x \text{ otherwise} \end{cases}$	**Hard Sigmoid** $\begin{cases} 0 \text{ if } x \leq -3 \\ 1 \text{ if } x \geq 3 \\ x/6+1/2 \text{ otherwise} \end{cases}$
Tanh Shrink $x - \tanh(x)$	**Soft Shrink** $\begin{cases} x-\lambda \text{ if } x > \lambda \\ x+\lambda \text{ if } x < -\lambda \\ 0 \text{ otherwise} \end{cases}$	**Hard Shrink** $\begin{cases} x \text{ if } x > \lambda \\ x \text{ if } x < -\lambda \\ 0 \text{ otherwise} \end{cases}$

10 Most Common Loss Functions in Machine Learning

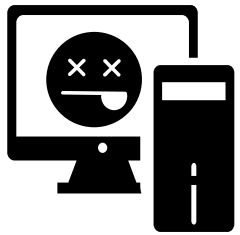
blog.DailyDoseofDS.com

Loss Function Name	Description	Function
Regression Loss Functions		
Mean Bias Error	Captures average bias in prediction. But is rarely used for training.	$\mathcal{L}_{MBE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))$
Mean Absolute Error	Measures absolute average bias in prediction. Also called L1 Loss.	$\mathcal{L}_{MAE} = \frac{1}{N}\sum_{i=1}^{N}\lvert y_i - f(x_i)\rvert$
Mean Squared Error	Average squared distance between actual and predicted. Also called L2 Loss.	$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$
Root Mean Squared Error	Square root of MSE. Loss and dependent variable have same units.	$\mathcal{L}_{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2}$
Huber Loss	A combination of MSE and MAE. It is parametric loss function.	$\mathcal{L}_{\text{Huberloss}} = \begin{cases} \frac{1}{2}(y_i - f(x_i))^2 & : \lvert y_i - f(x_i)\rvert \le \delta \\ \delta(\lvert y_i - f(x_i)\rvert - \frac{1}{2}\delta) & : otherwise \end{cases}$
Log Cosh Loss	Similar to Huber Loss + non-parametric. But computationally expensive.	$\mathcal{L}_{\text{LogCosh}} = \frac{1}{N}\sum_{i=1}^{N} log(cosh(f(x_i) - y_i))$
Classification Loss Functions (Binary + Multi-class)		
Binary Cross Entropy (BCE)	Loss function for binary classification tasks.	$\mathcal{L}_{BCE} = \frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(x_i)) + (1 - y_i) \cdot log(1 - p(x_i))$
Hinge Loss	Penalizes wrong and right (but less confident) predictions. Commonly used in SVMs.	$\mathcal{L}_{\text{Hinge}} = max(0, 1 - (f(x) \cdot y))$
Cross Entropy Loss	Extension of BCE loss to multi-class classification.	$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} . log(f(x_{ij}))$ $N : samples; M : classes$
KL Divergence	Minimizes the divergence between predicted and true probability distribution	$\mathcal{L}_{KL} = \sum_{i=1}^{N} y_i \cdot log(\frac{y_i}{f(x_i)})$

https://playground.tensorflow.org

24 M pixels
(r,v,b) 3x8 bits

3 x 24 M neurons ?!



Summary

- Neural network : large non-linear function, with “neurons”
- Layers: input / **parameters** / output
- Training process: optimization, **loss**, **backpropagation**
- Image specific: **convolutional** neural network

1 Go to wooclap.com

2 Enter the event code in the top banner

Event code
GMCCKU

1 Send @GMCCKU to 06 44 60 96 62

2 You can participate

Disable answers by SMS