Outline

Importance of

“*Deep Architecture*”

Why deep architecture should be important?

- It claimed that ...
- " Artificial Neural Network is a **Universal Model for General AI** "
- That means …
- "ANN can learn any computable formula"
  - It has been proved,
    - Scarselli & Tsoi, 1998
    - Schäfer & Zimmermann, 2006
  - *But it fails without depth!*

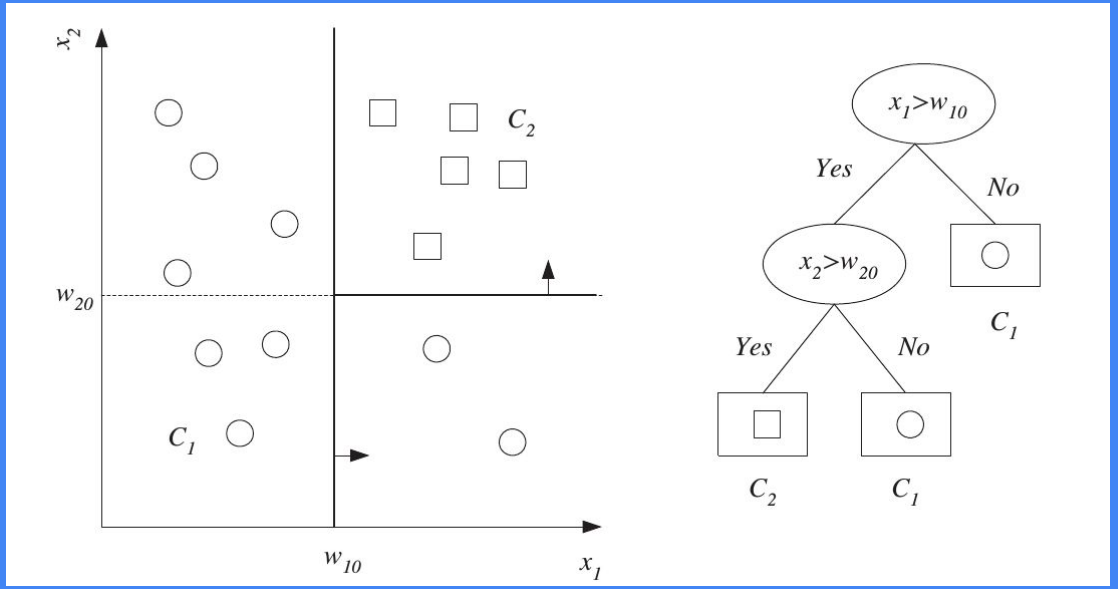
What we talk about when we talk about intelligence?

- To what extent a creature or a machine should *understand* things so that we could call it *intelligent*?
- Understanding sophisticated Logical compounds
- Understanding Arithmetics and Algebra
- Understanding association between words and meanings
- …

A simple interpretable classification model using *Decision Trees*

درخت تصمیم

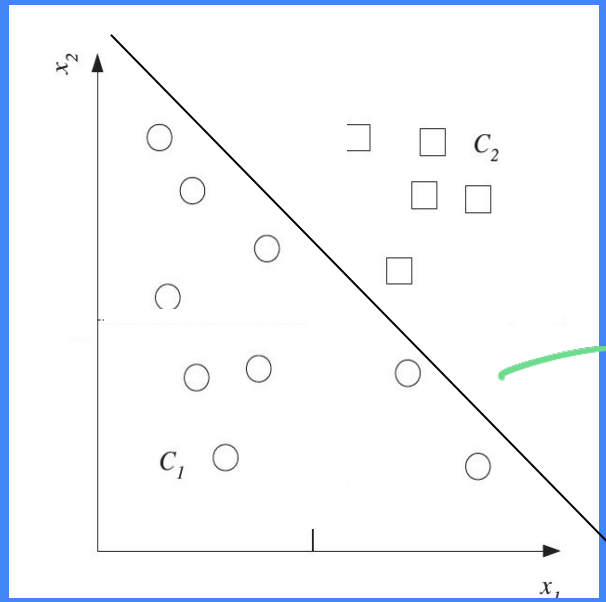
- Why ANN? Isn't logic enough?!
- To be intelligent,
  organize your "IF … THEN …" rules



... also can be achieved by a *Linear Model*

دسته‌بند خطی

- Logic helps us to understand each other
- Arithmetics helps us to understand nature
- To be intelligent,
  find the right way to “Multiply and Sum”

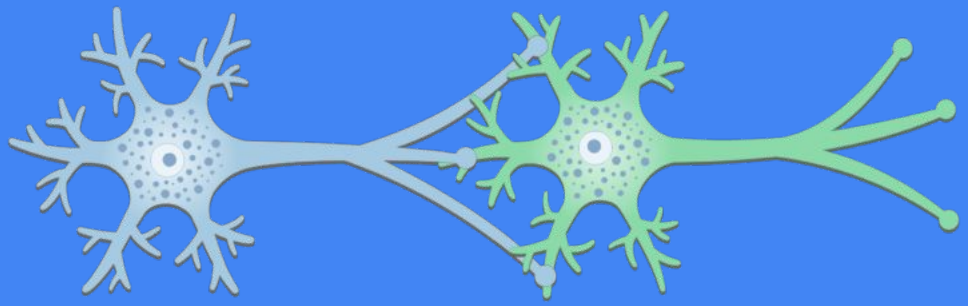


$$ax_1 + bx_2 + c = 0$$

A simple linear model:

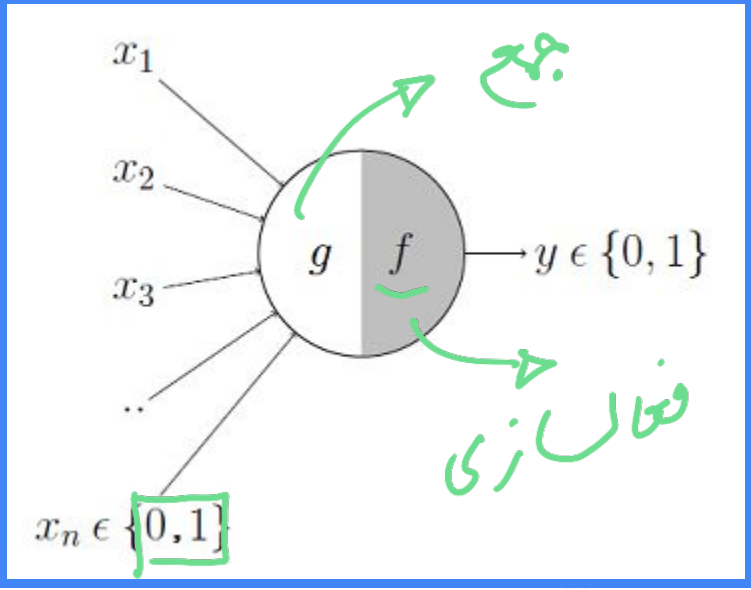
McCulloch-Pitts Neuron

- Twoard **General AI**, let's mimicking **nature**



A simple linear model:

McCulloch-Pitts Neuron



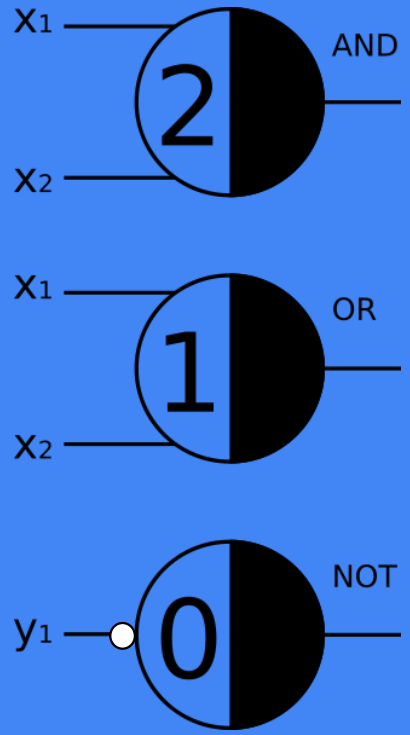
$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

- ** Only excitory inputs*

$$y = f(g(\mathbf{x})) = 1 \quad if \quad g(\mathbf{x}) \geq \theta$$
$$= 0 \quad if \quad g(\mathbf{x}) < \theta$$

Power of *McCulloch-P itts Neuron*

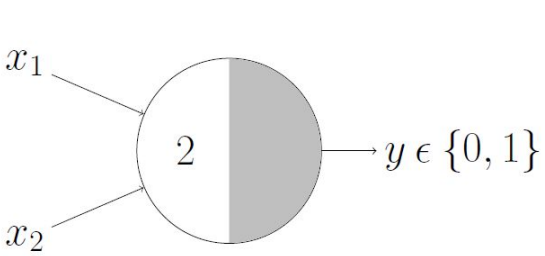
- Boolean functions can be computed by M-P neurons



$x_i \in \{0, 1\}$

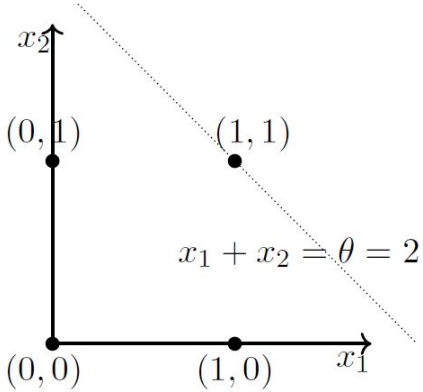
$w_i \in \{+1, -1\}$

Logical *AND*

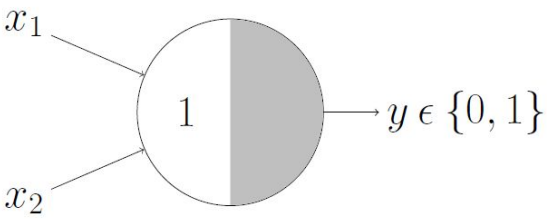


AND function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$$

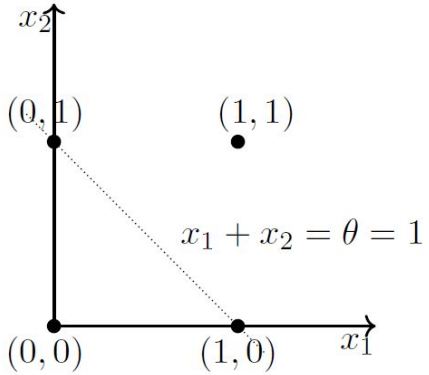


Logical *OR*

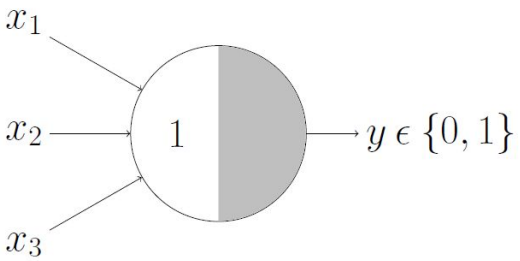


OR function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$

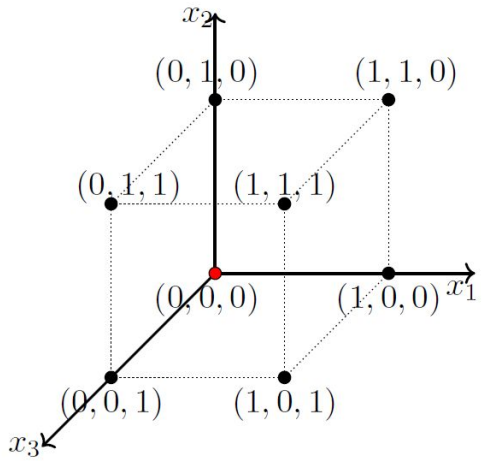


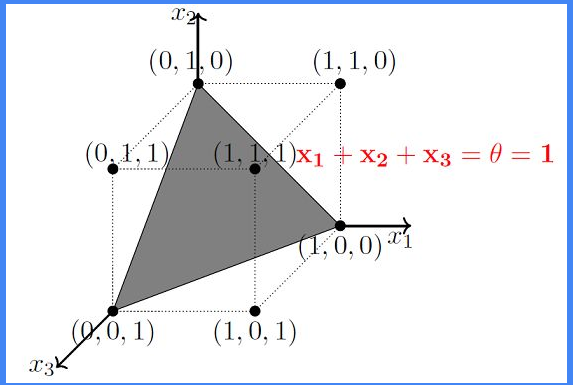
Logical *OR* with 3 inputs



OR function

$$x_1 + x_2 + x_3 = \sum_{i=1}^{3} x_i \geq 1$$

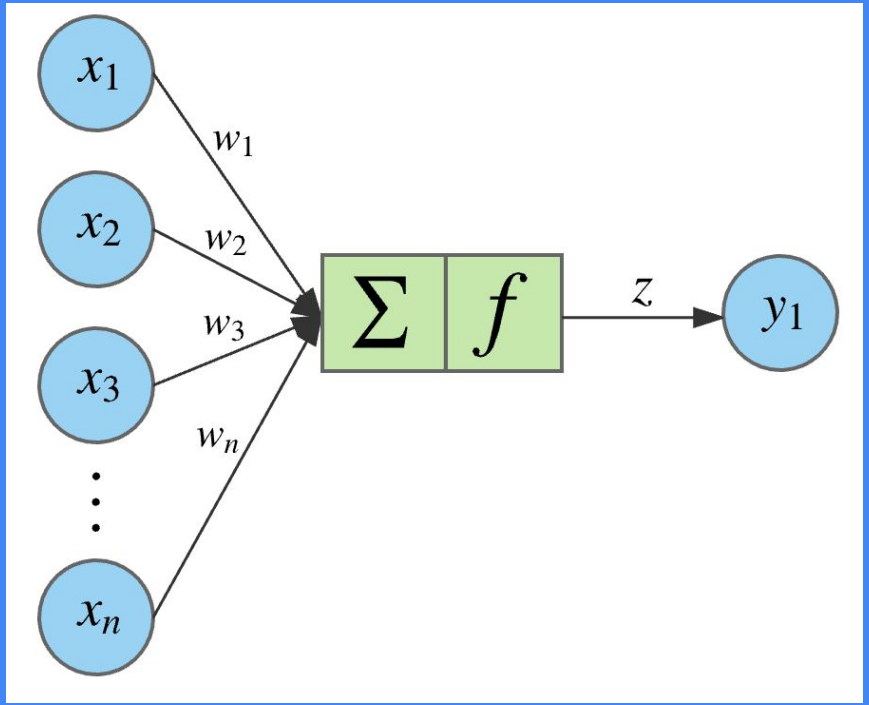




Limitation of M-P neurons

- What about **non-boolean (say, real) inputs?**
- Do we always **need to hand code the threshold?**
- Are all inputs equal?
  What if we want to **assign more importance to some inputs?**
- What about **functions which are not linearly separable?**
  - For example; XOR function

Assign more importance to some inputs by introducing ***weights***

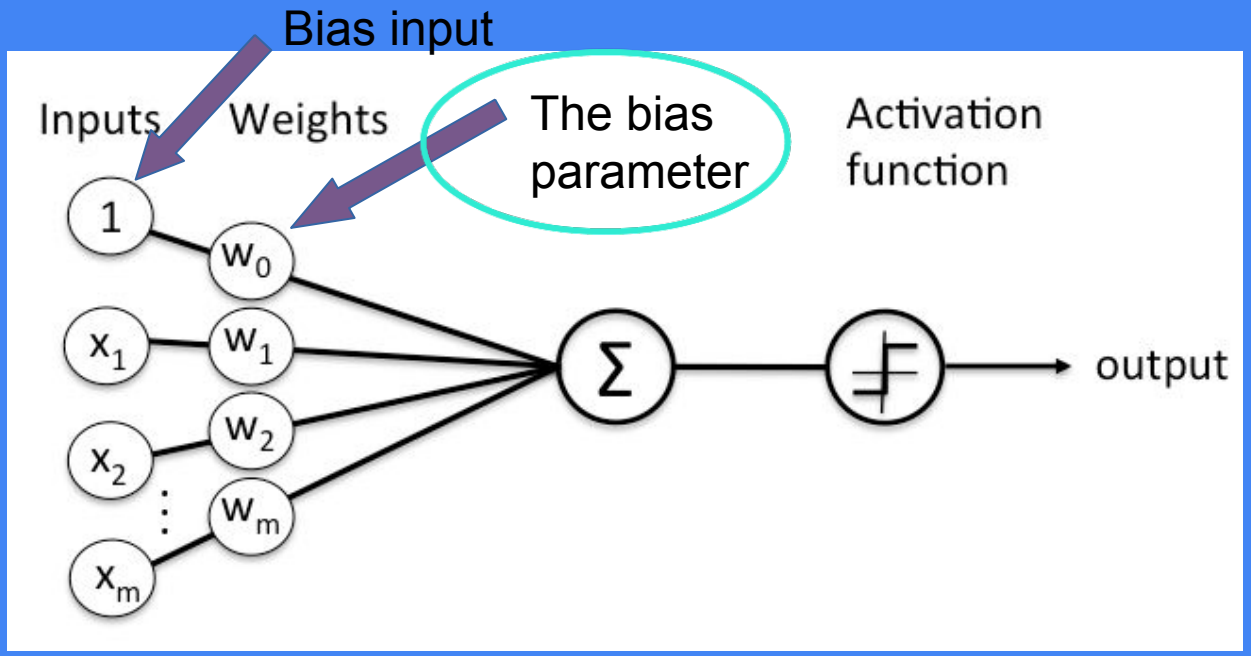


Assign more importance to some inputs by introducing ***weights***

$$z = f(x \cdot w) = f\left(\sum_{i=1}^{n} x_i w_i\right)$$

$$x \in d_{1\times n},\ w \in d_{n\times 1},\ z \in d_{1\times 1}$$

… and use a **bias** to handle the **threshold**



… and use a **bias** to handle the **threshold**

$$z = f(b + x \cdot w) = f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

$$x \in d_{1\times n},\ w \in d_{n\times 1},\ b \in d_{1\times 1},\ z \in d_{1\times 1}$$

$z = f(x \cdot w)$
if $z \geq \theta \Rightarrow x \in C_1$
if $z < \theta \Rightarrow x \in C_2$

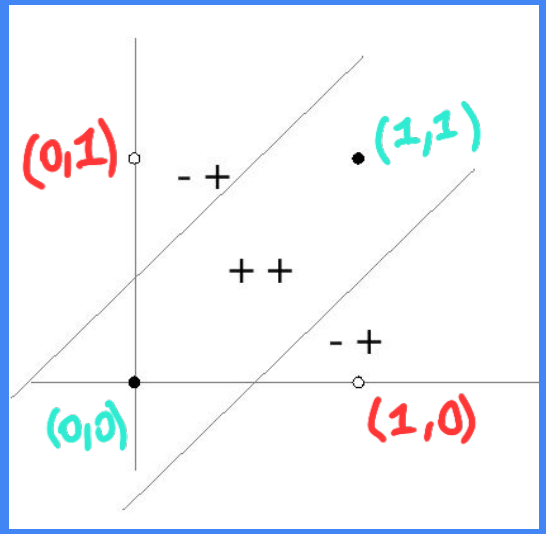
Eqiv. (f inc)

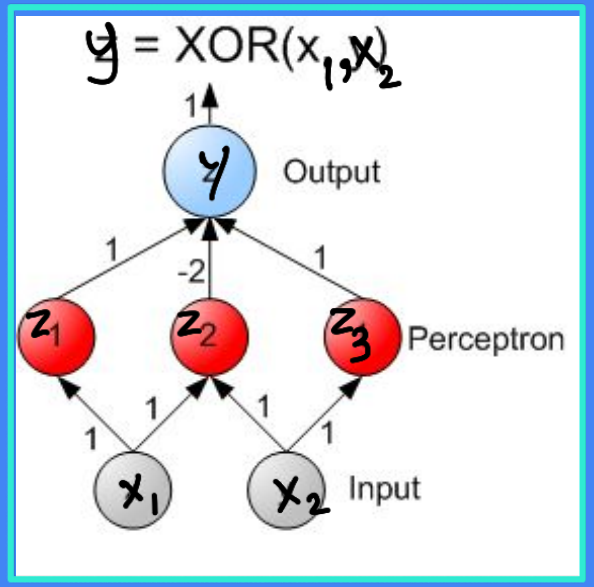
$b = f^{-1}(0) - f^{-1}(\theta)$

$z = f(b + x \cdot w)$
if $z \geq 0 \Rightarrow x \in C_1$
if $z < 0 \Rightarrow x \in C_2$

… and, add **extra layers** to model *non-linear functions*

- A **Hidden Layer** is required to model the Logical XOR gate

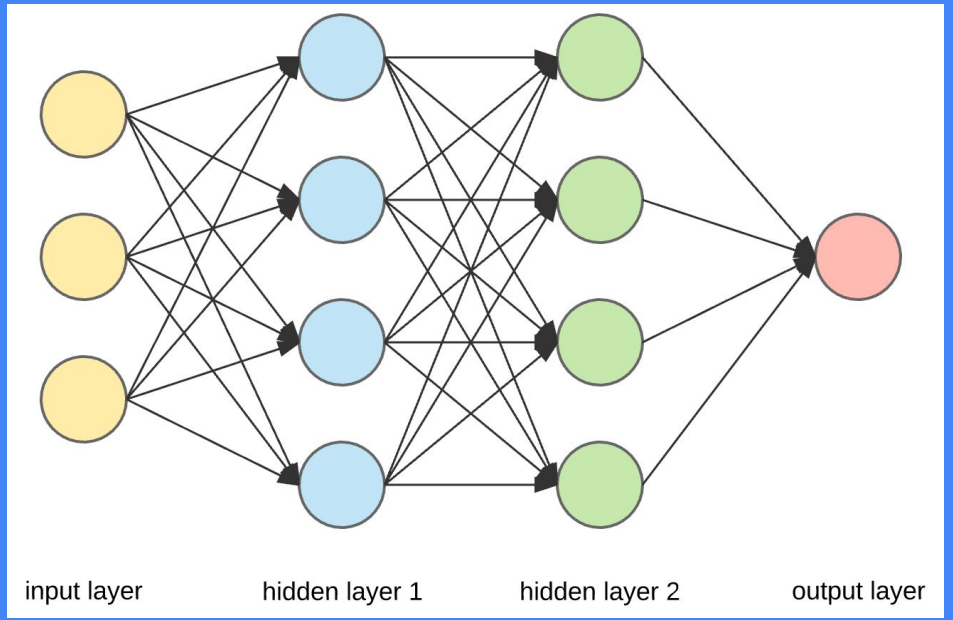




Feed-forward **Multi-layer Preceptron** - *a general approximator* ✓✓

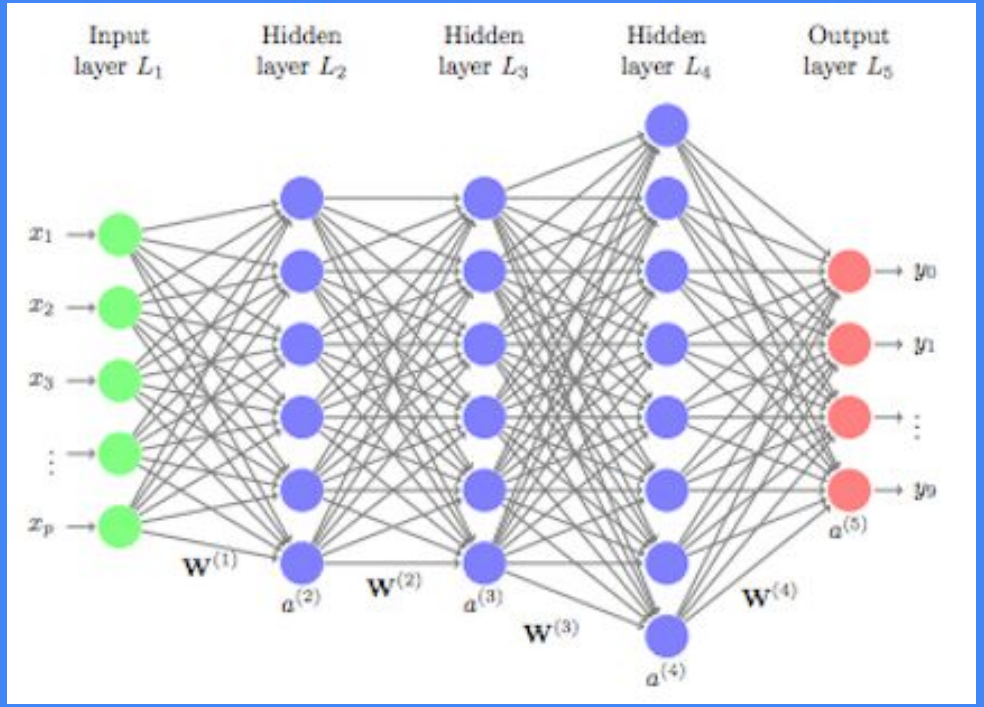
(MLP)

- A single-output Multi-layer Perceptron



Feed-forward **Multi-layer Preceptron** - *a general approximator*

- A multi-output Multi-layer Perceptron



Feed-forward **Multi-layer Preceptron** - *a general approximator*

- A multi-output Multi-layer Perceptron

$$a_1 = f(x \cdot W_1)$$

$$a_2 = f(a_1 \cdot W_2)$$

$$y = f(a_2 \cdot W_3)$$

$$\Downarrow$$

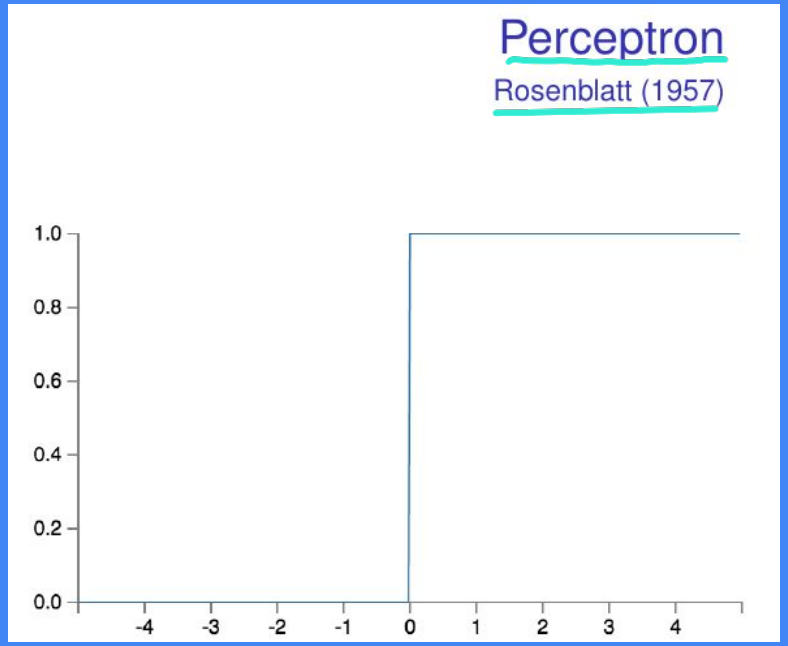
$$y = f(\, f(\, f(x \cdot W_1) \cdot W_2) \cdot W_3)$$

Choices for *the activation functions*

- Reapiting layers with *linear activation function* does not yield any powerful model but the linear model.
- **The general approximator** requires non-linear (differentiable) activation functions;
  - Sing (non-differentiable)
  - Sigmoid
  - Tanh
  - ReLU (non-differentiable)

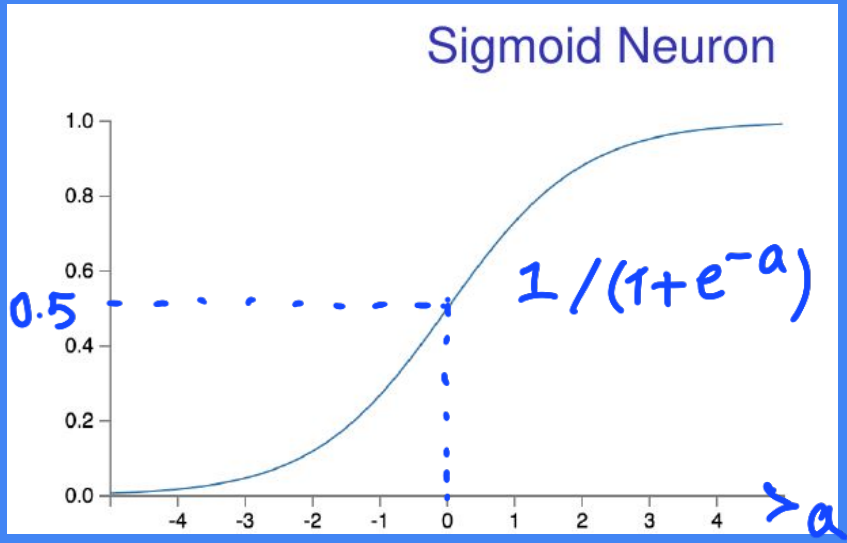
Choices for *the activation functions*

- Step function (equiv. Sign function behavior)
  - as used in the classic Linear Perceptron



Choices for *the activation functions*

- Sigmoid function
  - as used in Multi-layer Perceptron



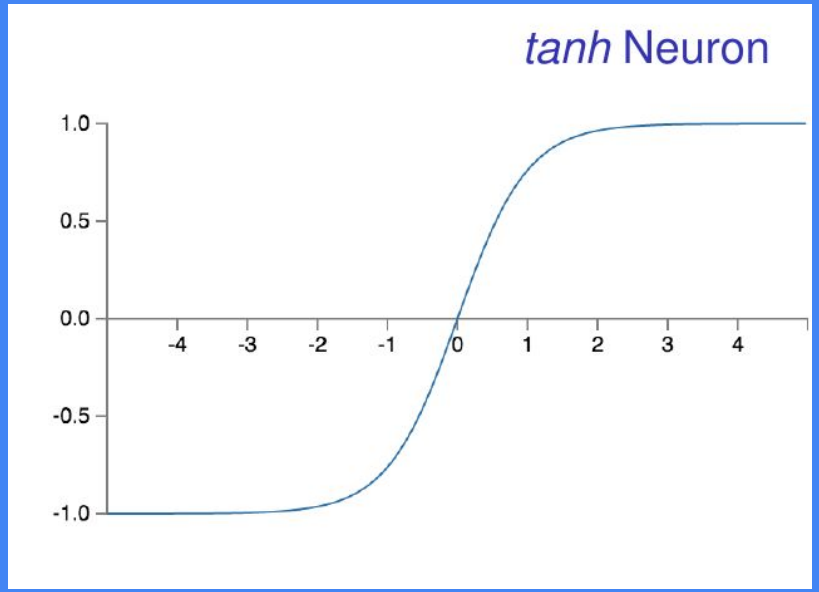
logit

$$\sigma(a) = \frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$

a

Choices for *the activation functions*

- Tanh function (equiv. behaves like Sigmoid)
  - as used in Multi-layer Perceptron



$$\sigma(z) = \frac{1 + \tanh(z/2)}{2}$$

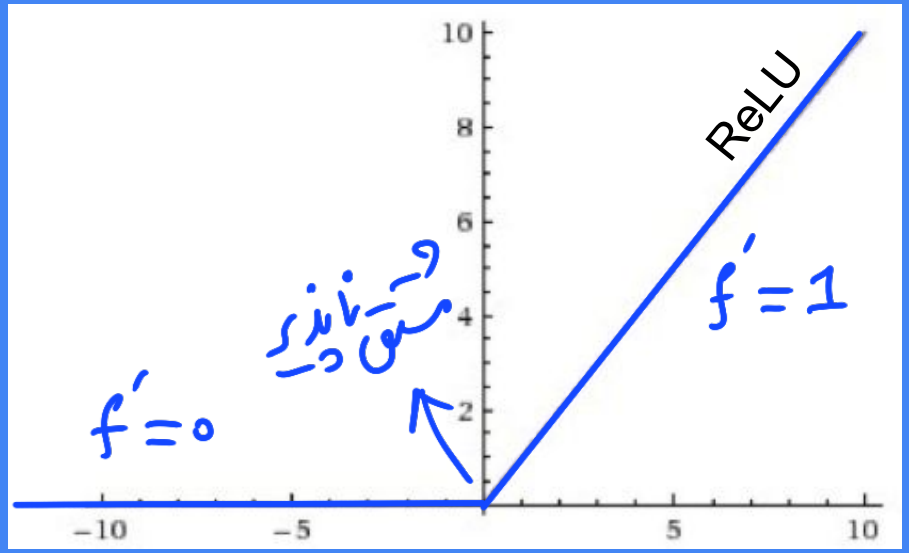
Choices for *the activation functions*

- ReLU: Rectified Linear Units
  - widely used in deep neural nets; particularly in Convolutional Neural Nets

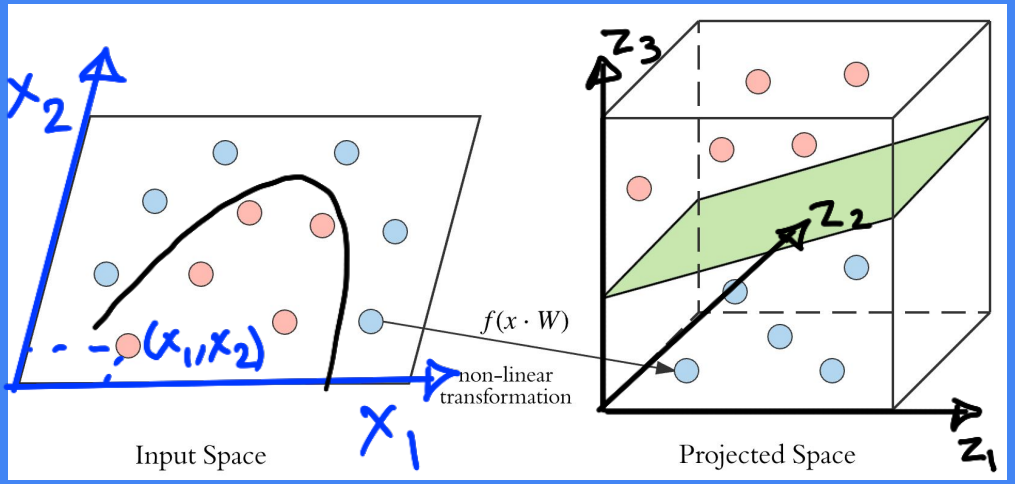
تابع یک‌سویه خطی

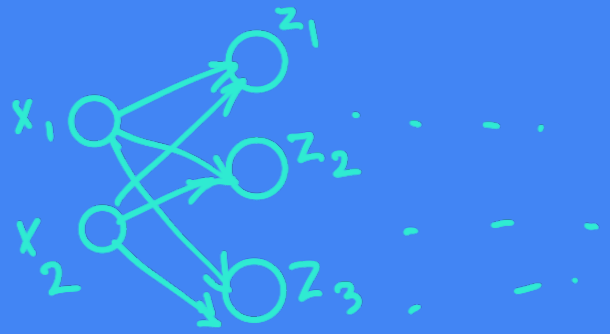
$$ReLU(a) = \begin{cases} a & \text{if } a \geq 0 \\ 0 & \text{if } a < 0 \end{cases}$$

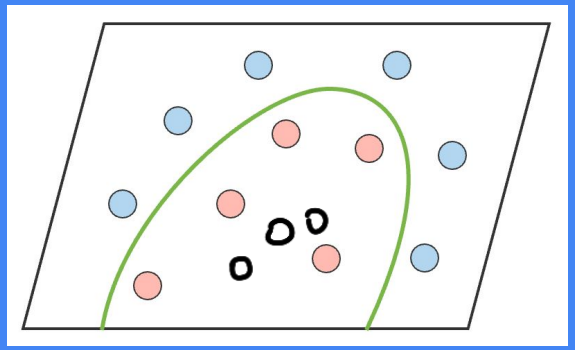
$$ReLU(a) = \max(a, 0)$$



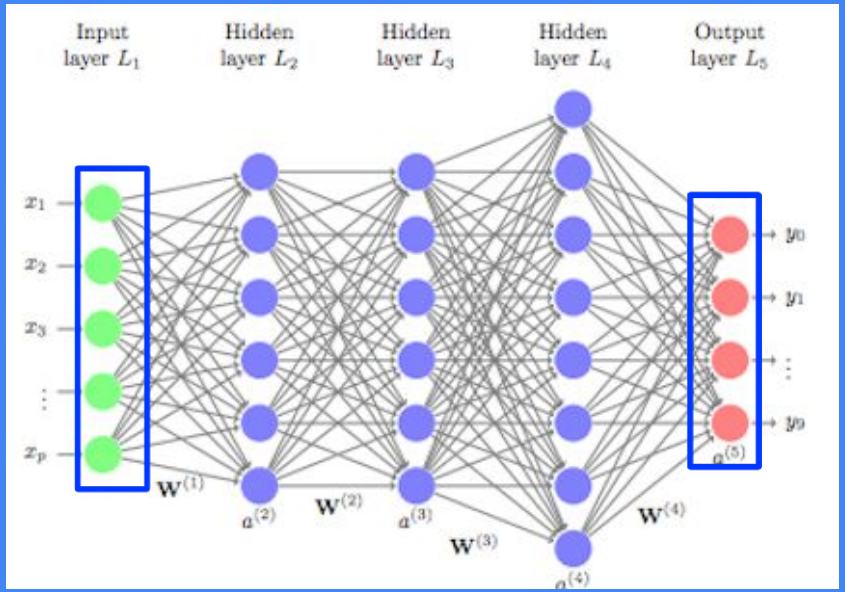
The advantage of **depth**

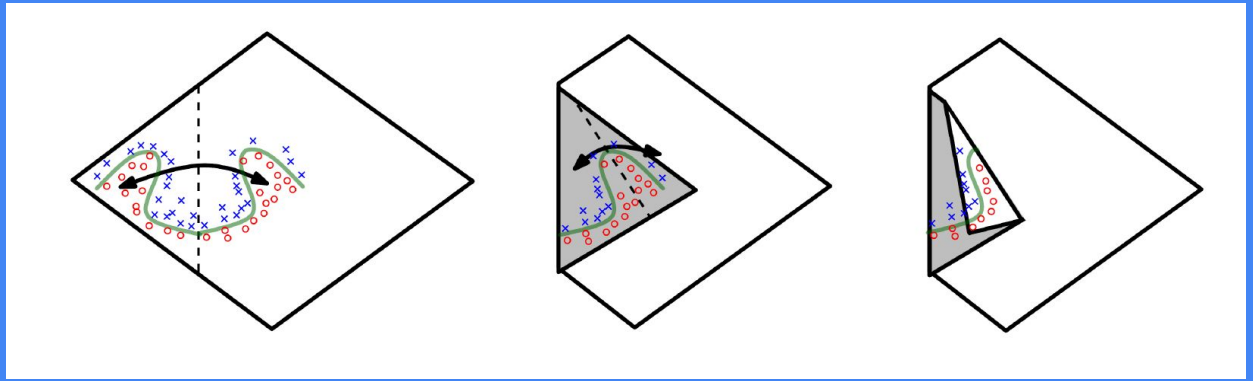






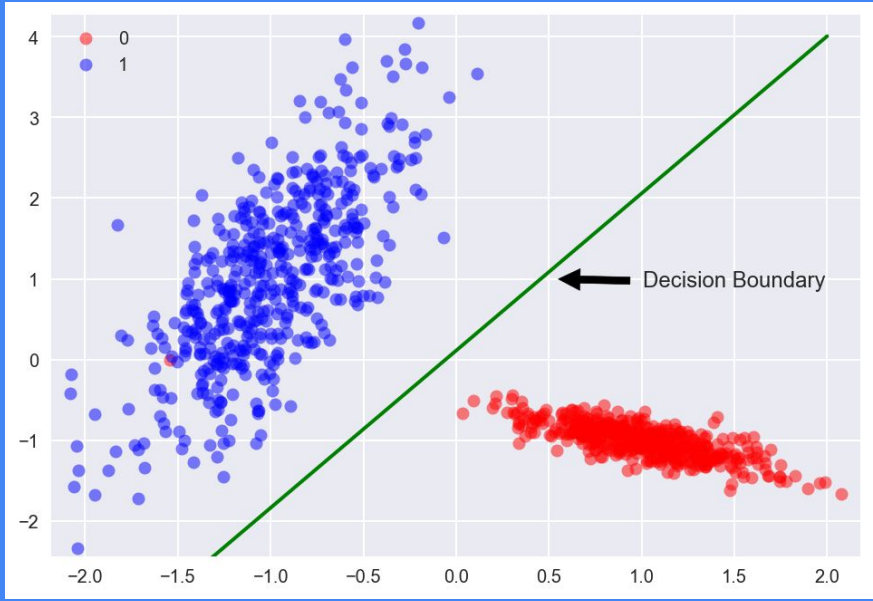
The advantage of **depth**





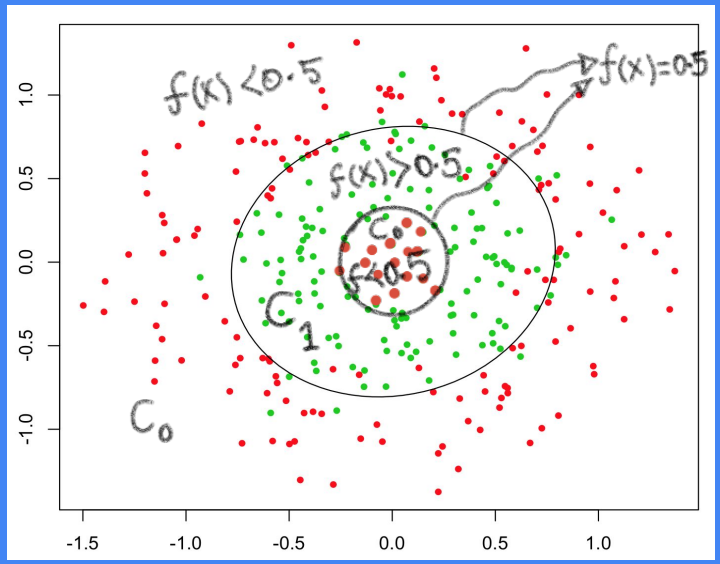
Computational power of MLP w.r.t **depth**

- Any *linearly-separable data* can be learned by a Linear Perceptron - no hidden layer required.



Computational power of MLP w.r.t **depth**

- Any *nonlinear classification problem* can be learned by an MLP with <u>at least one hidden layer</u>
  - required a *continious decision boundary*.



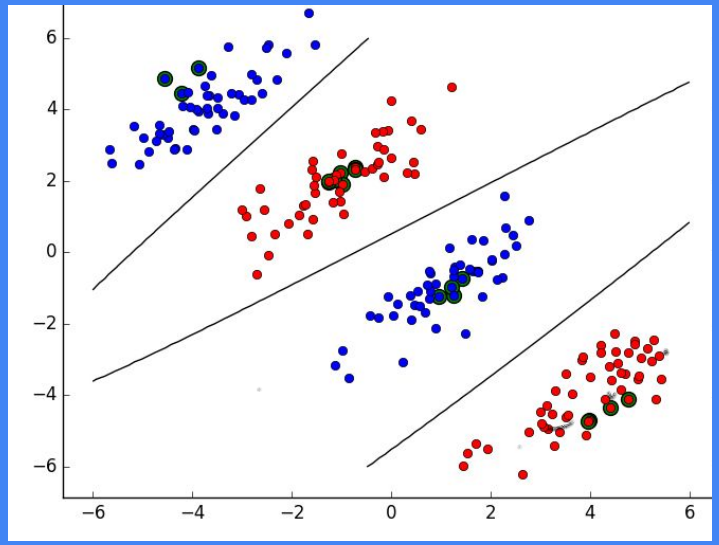
Computational power of MLP w.r.t **depth**

- A multilayer network of neurons with a single hidden layer can be used to *approximate any continuous function* to *any desired precision*.

Mathematically, there is a guarantee that for any function $f(x): R(n) \rightarrow R(m)$, we can always find a neural network with hidden layer/s whose output $g(x)$ satisfies $|g(x)-f(x)| < \Theta$

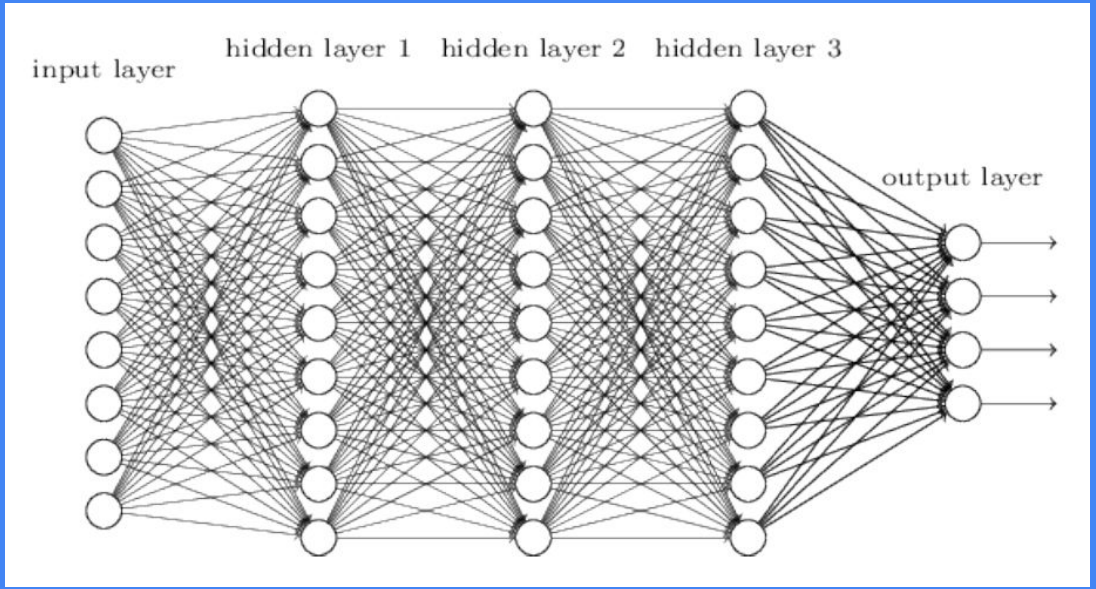
Computational power of MLP w.r.t **depth**

- Any *nonlinear classification problem* can be learned by an MLP with <u>at least two hidden layers</u>,
  - even, the decision boundary is *discontinious*.



The advantage of **depth**

The issue of **depth**

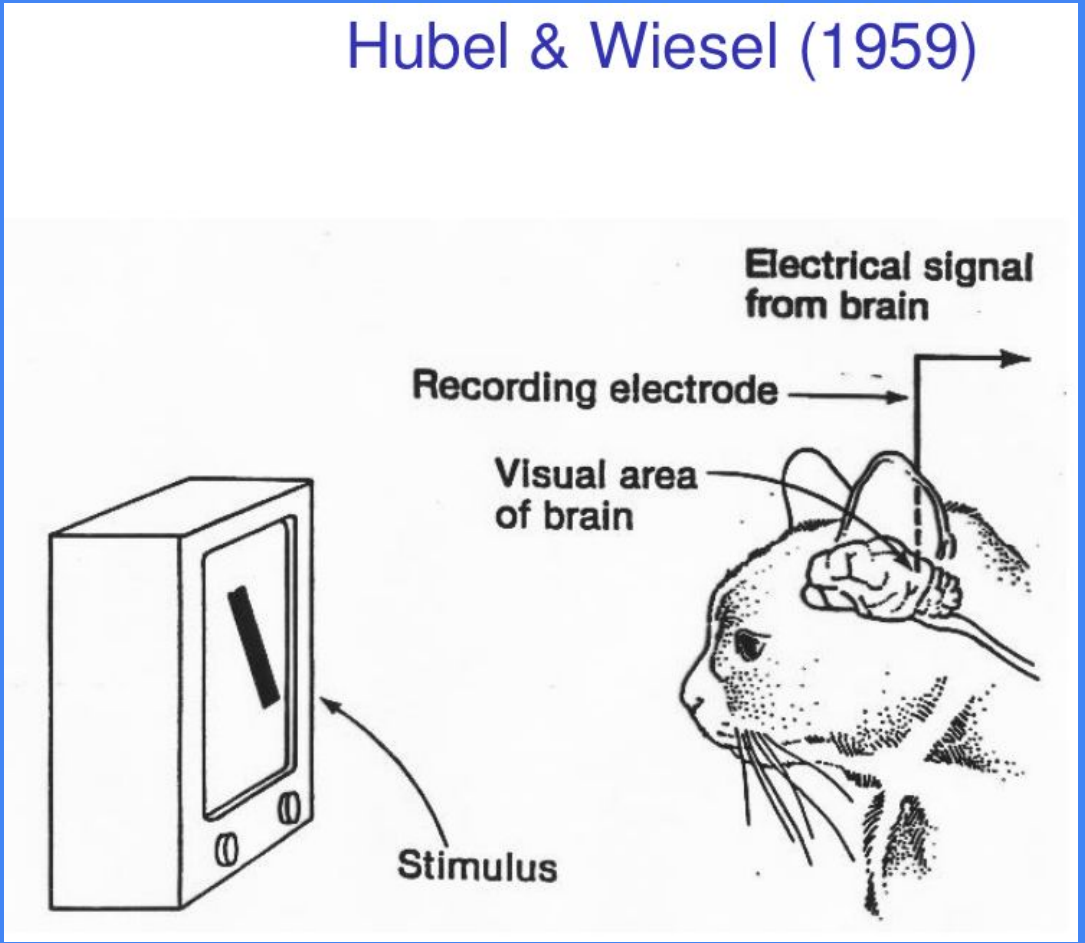


- The number of model parameters factorially grows by depth.
- Failed to achieve a general model, in practice.

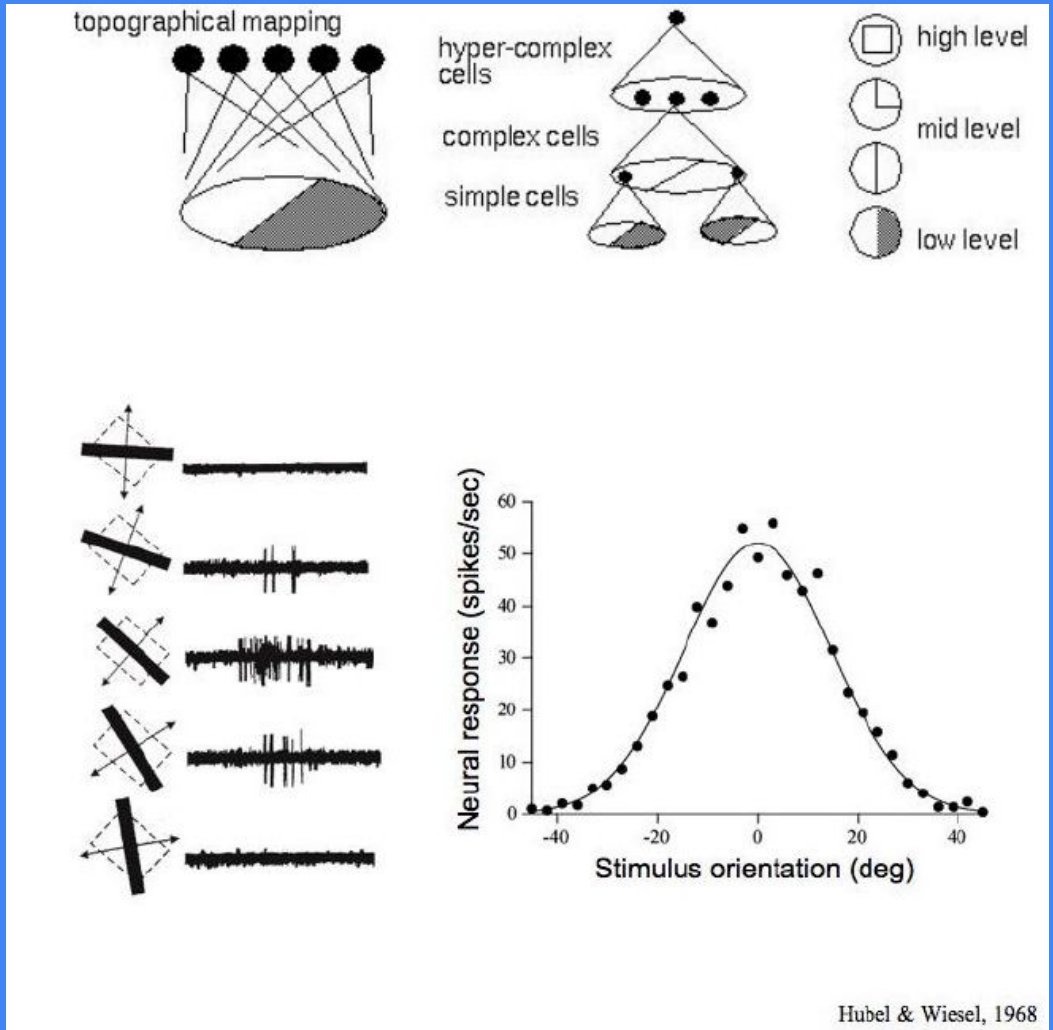
Convolutional Neural Networks

- The issue of factorially growth of number of model parameters could be resolved by "*parameter sharing*" within layers,
  - that the approach used by **Convolutional Neural Networks**,
  - while the advantage of Linear operations is kept.
  - That is also a nature inspired approach ...

Convolutional Neural Networks



Convolutional Neural Networks



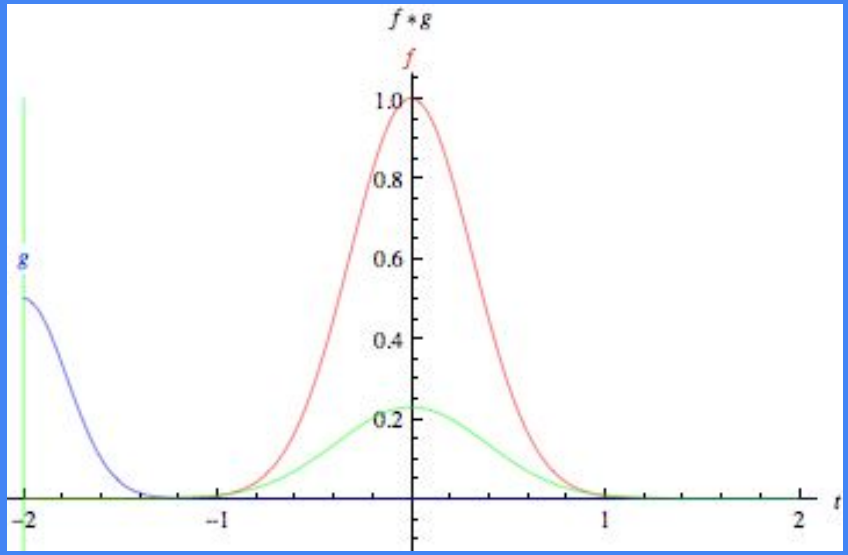
The Convolutional operator

$$f, g : \mathbb{R}^d \to R$$

$$[f \circledast g](x) = \int_{\mathbb{R}^d} f(z)g(x - z)dz.$$

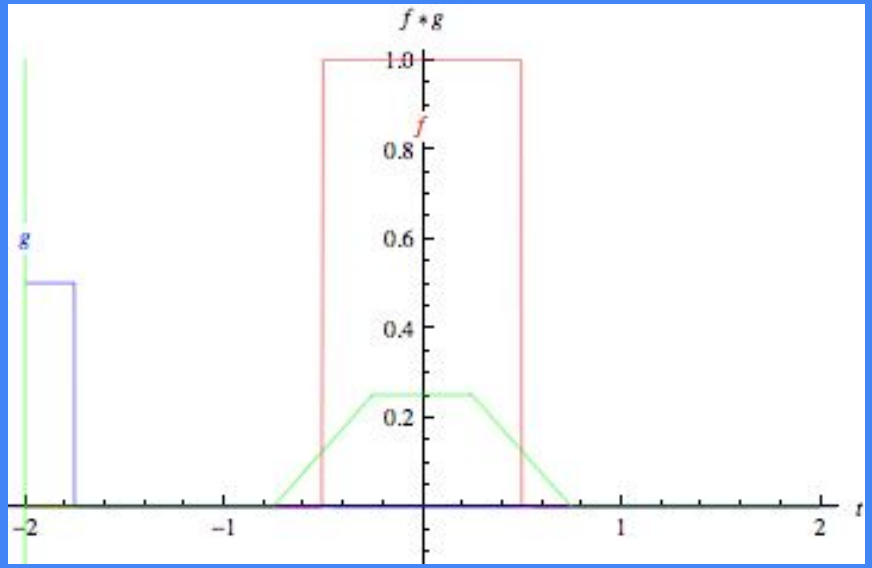
$$[f \circledast g](i) = \sum_a f(a)g(i - a).$$

The Convolutional operator as *Local feature detector*



$$(f \times g)(t) = \int_{-\infty}^{\infty} f(\tau) \times g(t-\tau) d\tau$$

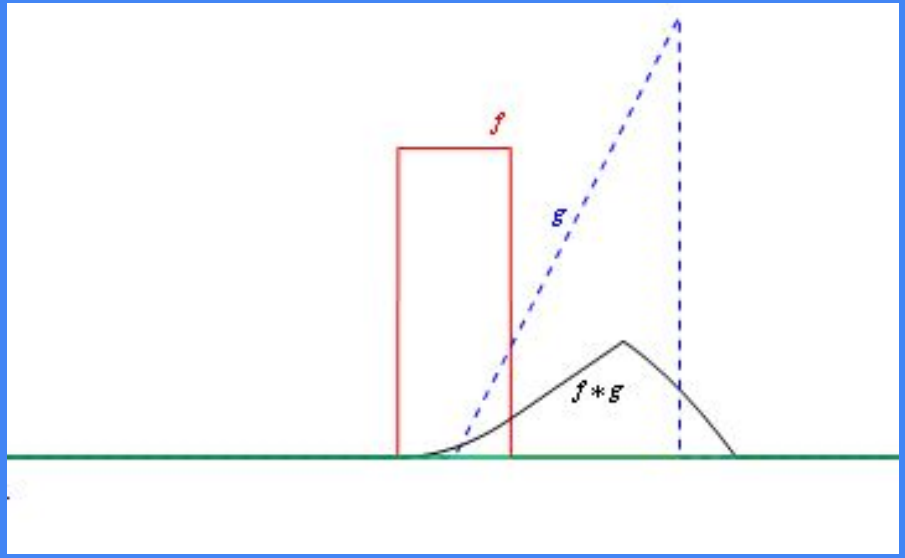
The Convolutional operator as *Local feature detector*



$$(f \times g)(t) = \int_{-\infty}^{\infty} f(\tau) \times g(t-\tau) d\tau$$

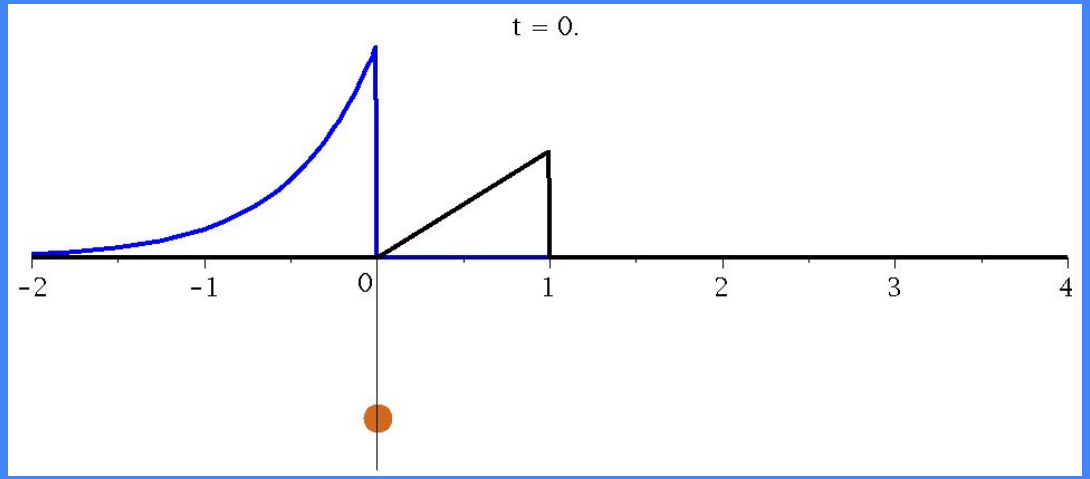
Kernel هسته

The Convolutional operator as *Local feature detector*



$$(f \times g)(t) = \int_{-\infty}^{\infty} f(\tau) \times g(t-\tau) d\tau$$

The Convolutional operator as *Local feature detector*

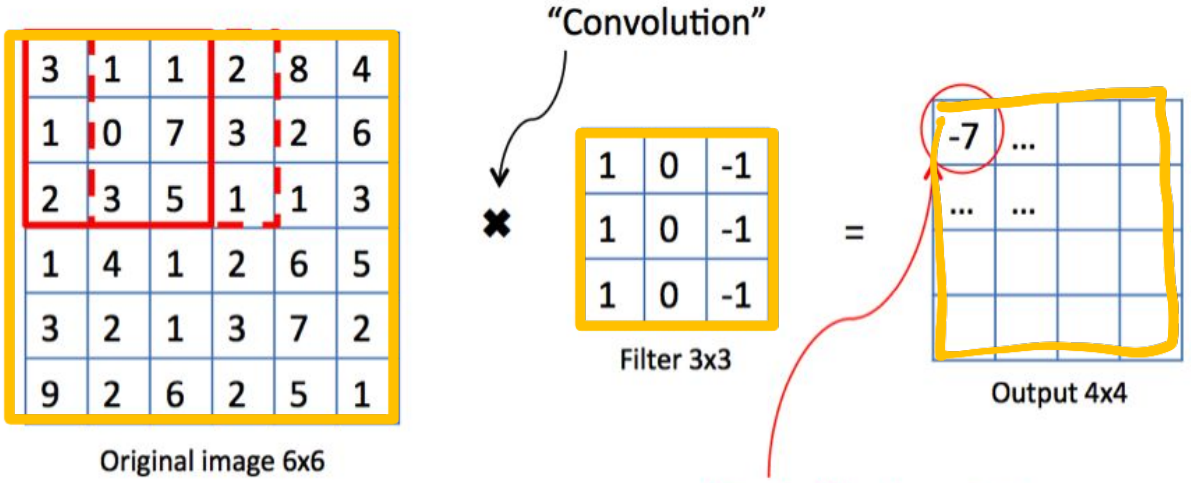


$$(f \times g)(t) = \int_{-\infty}^{\infty} f(\tau) \times g(t-\tau) d\tau$$

The Classic Application of Convolutional Filters in Image Processing

For an image $\mathbf{I}(x, y)$ and a convolutional filter $\mathbf{K}(m, n)$:

$$\mathbf{V}(x, y) = \sum_m \sum_n \mathbf{I}(x + m, y + n)\mathbf{K}(m, n)$$



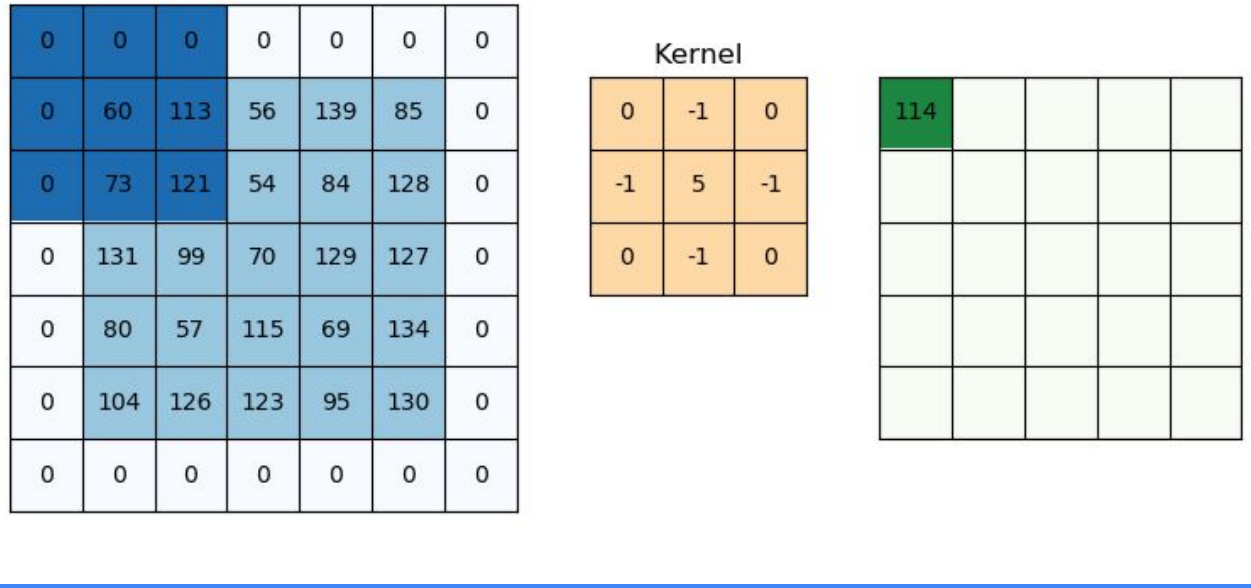
The Classic Application of Convolutional Filters in Image Processing

Example:

$$\mathbf{I} = \begin{bmatrix} 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 1. & 1. & 1. & 0. & 0. \\ 0. & 1. & 1. & 1. & 0. & 0. \\ 0. & 1. & 1. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. \end{bmatrix}, \mathbf{K} = \begin{bmatrix} 1. & -1. \\ 1. & -1. \end{bmatrix}$$

- Also known as convolutional kernel
- Actually, it is not convolution, but more like cross-correlation

The Classic Application of Convolutional Filters in Image Processing



Sliding window operation applied to an image

1. A 3×3 filter (example) is used
2. multiply its values element-wise with the original image
3. then sum them up.
4. Do this for each element by sliding the filter over the whole matrix.

The Classic Application of Convolutional Filters in Image Processing

- What is this specific filter $\mathbf{K}$ for? Where will we get the highest, zero and lowest values? (vertical line detection)

$$\mathbf{I} = \begin{bmatrix} 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 1. & 1. & 1. & 0. & 0. \\ 0. & 1. & 1. & 1. & 0. & 0. \\ 0. & 1. & 1. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. \end{bmatrix}, \mathbf{K} = \begin{bmatrix} 1. & -1. \end{bmatrix}, \mathbf{V} = \begin{bmatrix} 0 & 0. & 0. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. & -1. & 0. \\ 0. & 1. & 0. & 0. & -1. & 0. \\ 0. & 1. & 0. & 0. & -1. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. \end{bmatrix}$$

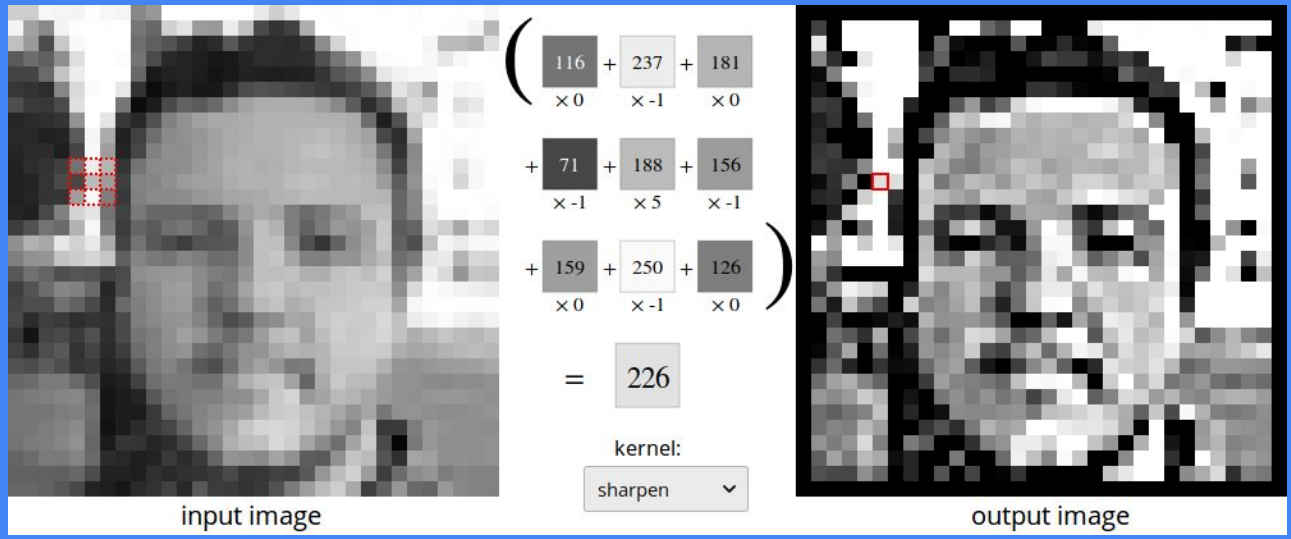
- How about these filters? (bluring and edge detection)

$$\begin{bmatrix} 0. & 0. & 0. & 0. & 0. \\ 0. & 1. & 1. & 1. & 0. \\ 0. & 1. & 1. & 1. & 0. \\ 0. & 1. & 1. & 1. & 0. \\ 0. & 0. & 0. & 0. & 0. \end{bmatrix}, \begin{bmatrix} 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. & 0. \\ 0. & 1. & -4. & 1. & 0. \\ 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \end{bmatrix}$$

https://docs.gimp.org/2.8/en/plug-in-convmatrix.html

Effect of Convolution Kernels;

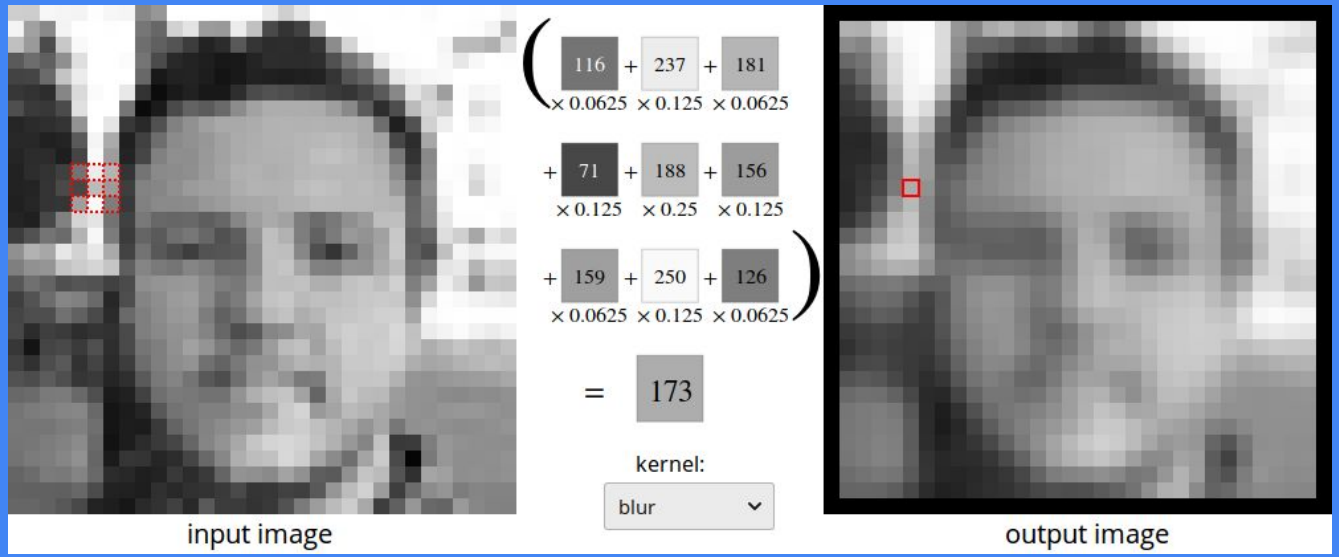
lessons from Image Processing



https://setosa.io/ev/image-kernels/

Effect of Convolution Kernels;

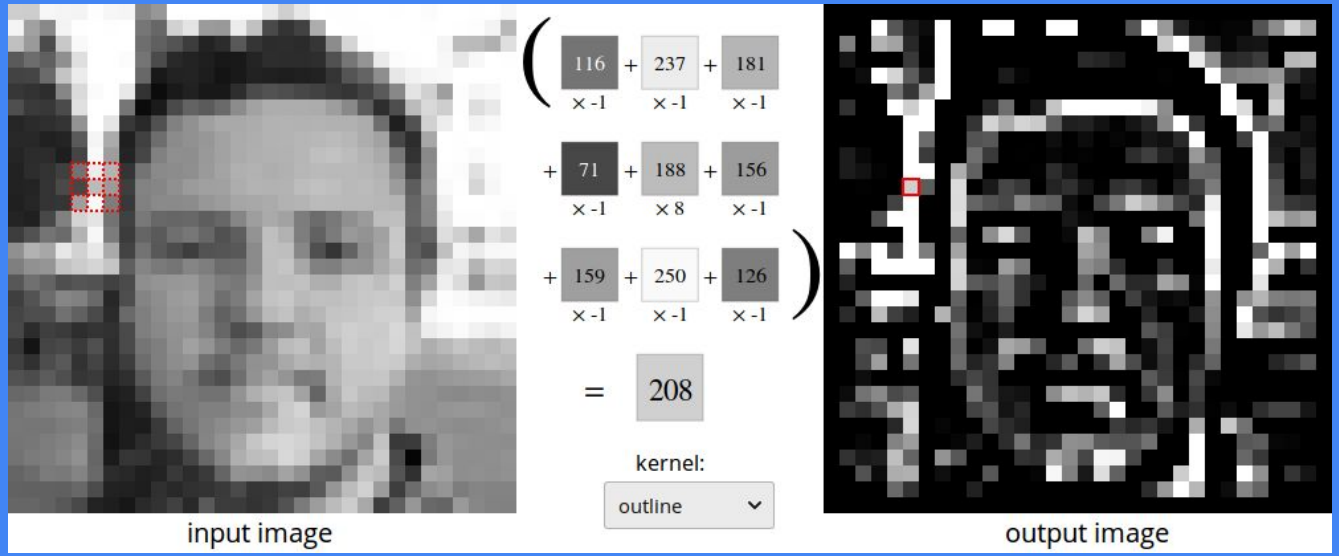
lessons from Image Processing



https://setosa.io/ev/image-kernels/

Effect of Convolution Kernels;

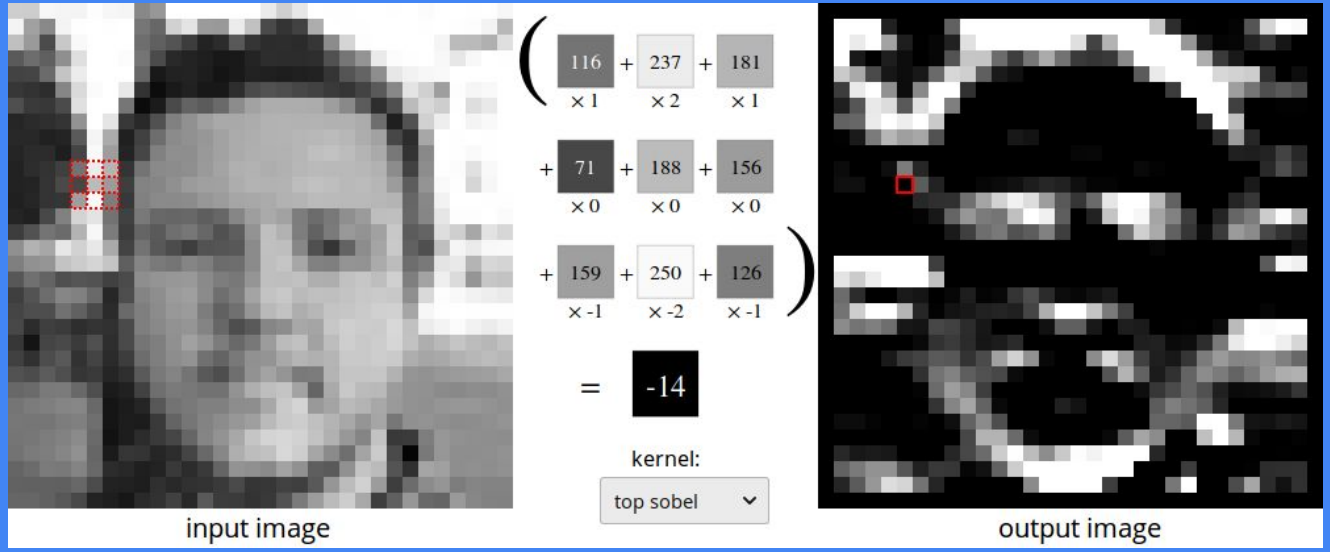
lessons from Image Processing



https://setosa.io/ev/image-kernels/

Effect of Convolution Kernels;

lessons from Image Processing



https://setosa.io/ev/image-kernels/