ML IN 2022

* Supervised Learning   99% by neural networks
  - self-supervised learning
  - Transfer learning
  etc.
* Unsupervised Learning
* Reinforcement Learning   99% using neural networks

NEURAL NETWORKS

Artificial model of neuron (Perceptron)   1940's

↓ Unmet promises & "AI winter"

A neural network   1960's
Feedforward Neural Network
Multi-layer Perceptron
Vanilla neural network

↓ Unmet promizes & better methods (SVM) & 2nd AI winter

Convolutional Neural Network   2000's

* Big bang: AlexNet 2012

  Tools: Caffe → TensorFlow, → PyTorch

  2012 – 2022 New Architectures

Backbones (AlexNet, VGGNet, ResNet, MobileNet, ...)
Encoder-Decoder
* U-Net
* DeepFake

GAN
NerF
Transformer ⇒ Gen. purpose differentiable "computer"

SUPERVISED ML

Essentially 3 main problems that are diff evaluated:

1. Classification
   * Assign one of K classes
   * Perf. meas. classification percent/rate 0...1.0

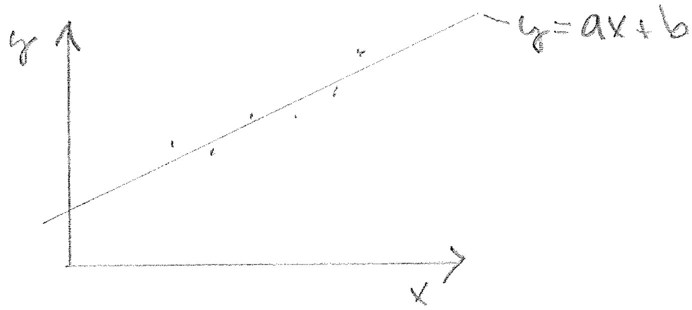
2. Regression
   * Estimate output given inputs
   * Perf. by MSE | MAE

3. Detection
   * Tell if (and when) event X present or not
   * Perf. by precision-recall (ROC curve)
     => two types of errors FP & FN

LINEAR MODEL

For regression:

x: input
y: output
a,b: tr. parameters
aka weights



0: points
1 points
2 points => high school
N points => Least Squares Fit

$$\begin{cases} y_1 = ax_1 + b \Rightarrow b = y_1 - ax_1 \\ y_2 = ax_2 + b \Rightarrow y_2 = ax_2 + y_1 - ax_1 \end{cases}$$

$$\Rightarrow \boxed{a = \frac{y_2 - y_1}{x_2 - x_1}}$$

For classification:

$x_1 \dots x_3$ from class 1

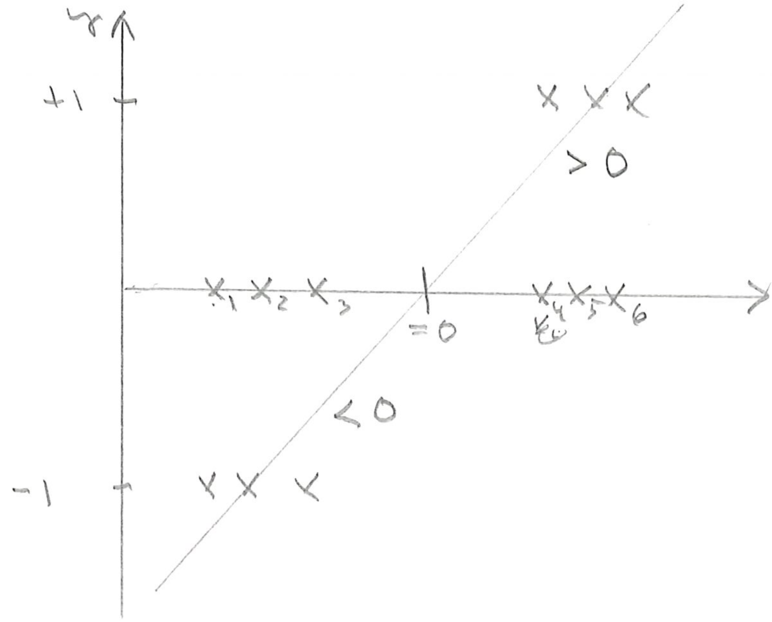
$x_4 \dots x_6$ from class 2

y?

Let's assign

$y=-1$ for C_1

$y=+1$ for C_2



⇒ Fit line

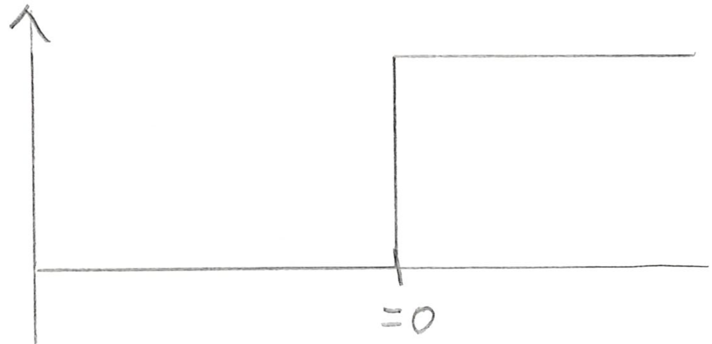
$$\text{class} = \begin{cases} 1 & \text{if } ax+b<0 \\ 2 & \text{if } ax+b>0 \end{cases}$$

Problem: Line fitting error depends on your distance to the discriminator point $y=0$ while intuitively err=0 for correct and err=1 for wrong classification should be used.

Fix:

$y=0$ for C_1

$y=1$ for C_2



Now ground truth and prediction error

$$\|y_{GT} - \hat{y}\| = \{0,1\}$$

or

$$(y_{GT} - \hat{y})^2 = \{0,1\}$$

if $y_{GT} = \hat{y}$ if $y_{GT} \neq \hat{y}$

step function is approximated by sigmoid

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

Linear classifier:

$$y = \text{logsig}(ax+b)$$

Optimization?

No closed form solution, but we can minimize error over N training samples

$$l_{MSE} = \frac{1}{N}\sum_{n=1}^{N}\left(y_{GT}^{(n)} - \hat{y}_n\right)^2 = \frac{1}{N}\sum_{n=1}^{N}\left(y_{GT}^{(n)} - ax_n - b\right)^2 \quad \text{reg.}$$

$$\uparrow$$

"loss" in ML terminology

$$= \frac{1}{N}\sum_{n=1}^{N}\left(y_{GT}^{(n)} - \text{logsig}(ax_n+b)\right)^2 \quad \text{class.}$$

We can go toward (local) minimum by taking a small step μ (learning rate) toward negative gradient.

$$a^{t+1} = a^t - \mu \frac{\partial l_{MSE}}{\partial a}$$

$$b^{t+1} = b^t - \mu \frac{\partial l_{MSE}}{\partial b}$$

$\Rightarrow$ Gradient Descent (GD) algorithm

Start from some random point (a^0, b^0)

Stuck to local minima?

$$\frac{\partial L_{MSE}}{\partial a} = \frac{\partial}{\partial a} \frac{1}{N} \sum_{n=1}^{N} \left(y_{GT}^{(n)} - \operatorname{logsig}(a x_n + b) \right)^2$$

$$\left[\frac{\partial}{\partial x} f^2(x) = 2f(x) \cdot f'(x) \quad \text{chain rule} \right]$$

$$= \frac{2}{N} \sum_{n=1}^{N} \left(y_{GT}^{(n)} - \operatorname{logsig}(a x_n + b) \right) \cdot \frac{\partial}{\partial a} \left(y_{GT}^{(n)} - \operatorname{logsig}(a x_n + b) \right)$$

$$\left[\frac{\partial}{\partial x} \operatorname{logsig}(x) = \operatorname{logsig}(x) \left(1 - \operatorname{logsig}(x) \right) \right]$$

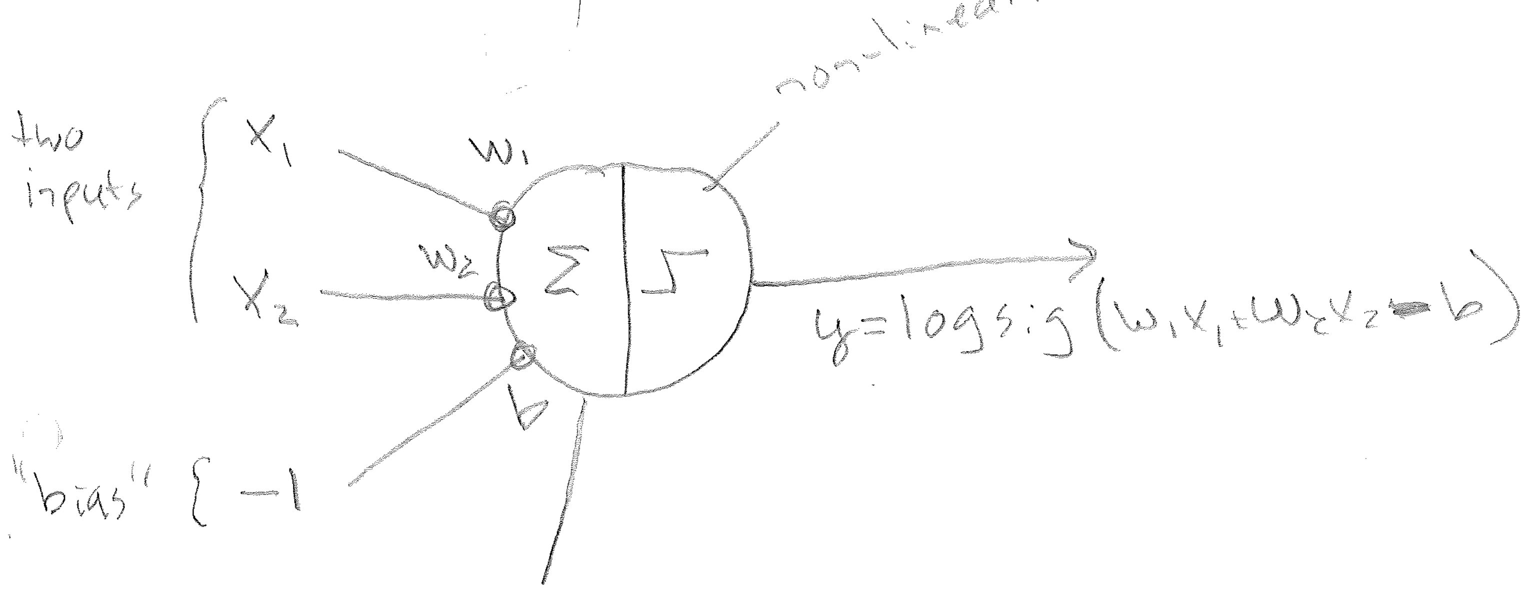
$$= \frac{2}{N} \sum_{n=1}^{N} \left(y_{GT}^{(n)} - \operatorname{logsig}(a x_n + b) \right) \cdot -\operatorname{logsig}(a x_n + b) \cdot \left(1 - \operatorname{logsig}(a x_n + b) \right) \cdot \underbrace{\frac{\partial}{\partial a} (a x_n + b)}_{= x_n}$$

$\Rightarrow$ its annoyingly long, but CPU does that for us once it is programmed!

A SINGLE NEURON

Biological neuron : <NOTEBOOK>

Artificial model:



$$y' = w_1 \cdot x_1 + w_2 \cdot x_2 - b$$

Single neuron is a linear model that can be trained using gradient descent.

<NOTEBOOK>