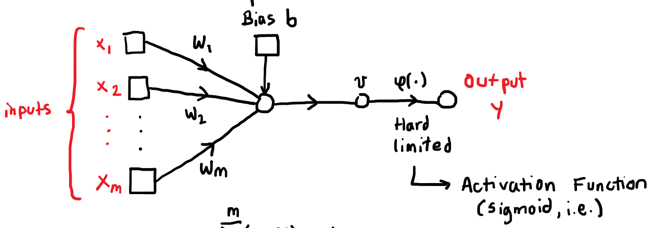
Basic Perceptron

Sunday, March 6, 2022 8:58 AM

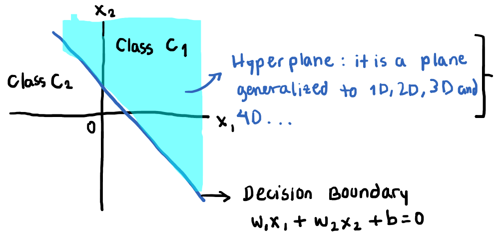
Rosenblatt's Perceptron → 1958



Model: $v = \sum_{i=1}^{m}(w_i x_i) + b$

It is basically a pondered sum of the inputs.
The parameters to modify are only the weights (w) and the bias (b).

→ The perceptron is a simple neural network algorithm used for classification problems. What it does is to generate a hyperplane:



The function adjusts to a hyperplane that separates C_1 and C_2 by updating w (b is constant).

→ Training the perceptron

↳ there are 3 convergence criteria:

→ Convergence conditions

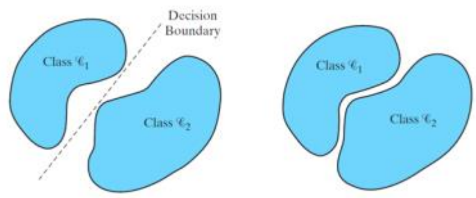
- The overall set of training vectors (input) is well defined (not NaN data, etc).
- The sets are separable: we can decide on this simply by training and if the output is constantly mistaken or accumulated a big error, it is probably because of non-separable data.

Data must be separable by a hyper plane. ← if it does not converge = NOT SEPARABLE

- Training algorithm:

→ $\bar{w}_{(n+1)} = \bar{w}_{(n)} + \eta_{(n)}\bar{x}_{(n)}$ if $\bar{w}_{(n)}\bar{x}_{(n)} = 0$ and $\bar{x}_{(n)}$ is C_1

→ $\bar{w}_{(n+1)} = \bar{w}_{(n)} - \eta_{(n)}\bar{x}_{(n)}$ if $\bar{w}_{(n)}\bar{x}_{(n)} > 0$ and $\bar{x}_{(n)}$ is C_2



There are two different approaches for NN:

Algorithmic approach	Bayesian approach
Brute Force Gradient Descent	Mathematic Input statistics

→ The algorithmic Approach (Haykin algorithm)

Variables and Parameters:

$$\mathbf{x}(n) = (m+1)\text{-by-}1 \text{ input vector}$$
$$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$$
$$\mathbf{w}(n) = (m+1)\text{-by-}1 \text{ weight vector}$$
$$= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T$$
$$b = \text{bias}$$
$$y(n) = \text{actual response (quantized)} \rightarrow \text{output}$$
$$d(n) = \text{desired response} \rightarrow \text{Supervised Learning}$$
$$\eta = \text{learning-rate parameter, a positive constant less than unity}$$

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation.* At time-step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as

$$y(n) = \operatorname{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

where $\operatorname{sgn}(\cdot)$ is the signum function.

4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain

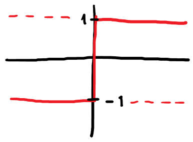
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[\underbrace{d(n) - y(n)}_{\text{error}}]\mathbf{x}(n) \quad \Big\} \text{ n is the iteration}$$

where

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$

5. *Continuation.* Increment time step n by one and go back to step 2.

Signum Function $\operatorname{sgn}(\cdot)$:

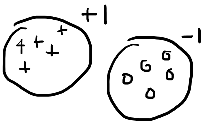


} For any input, the signum function limits it to be either 1 (positive input) or -1 (negative input).

↓

-1 and 1 end up being the classification of the data.

Loss Function: to minimize the loss since the ideal case is



} its sum is zero if they are correctly classified ($d(n)$)

$$w(n+1) = w(n) - \eta(n)\nabla J(w) \qquad \text{where } n \text{ is the iteration}$$

↳ so that the hyperplane decreases along the axis

$$= w(n) + \eta(n)\sum x(n)\, d(n)$$

↳ makes it oscilate to approach the convergence point

→ Bayes classifier → no a priori information (Algorithmic approach)

- We have a cost function (Loss function)

$$J(\bar{w}) = \sum_{\bar{x}(n) \in \mathcal{H}} (-\bar{w}^T \bar{x}(n)\, d(n))$$

- We aim for minimizing the cost function, by looking for the opposite direction of the gradient:

J depends on all $\bar{w}$ values ←

$$\nabla J(\bar{w}) = \sum_{\bar{x}(n) \in \mathcal{H}} (-\bar{x}(n)\, d(n)) \qquad \nabla = \underbrace{\left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_m}\right]^T}_{\text{since we got } m \text{ inputs}}$$

where n is the iteration.

- Thus, the algorithm for training the network as:

$$\bar{w}_{(n+1)} = \overbrace{\bar{w}_{(n)}}^{\text{vector}} - \overbrace{\eta_{(n)}}^{\text{constant}} \overbrace{\nabla J(\bar{w})}^{\text{vector}}$$

this jump depends on η (learning constant)

The hyperplane oscilates towards the final solution } Until convergence near $J=0$

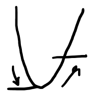
$$= \bar{w}_{(n)} + \eta_{(n)} \sum_{\bar{x}(n) \in \mathcal{H}} (\bar{x}_{(n)}\, d_{(n)})$$

{ cost function

towards the final solution

near $J=0$

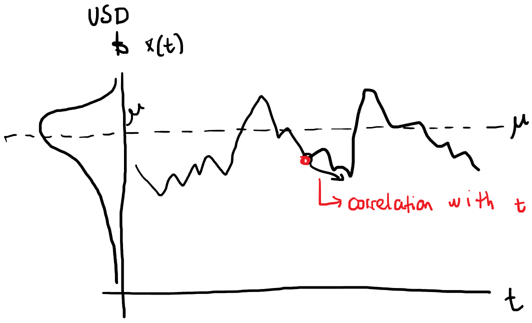
$$= \bar{w}_{(n)} + \eta_{(n)} \sum_{\bar{x}(n) \in \mathcal{H}} \left(\bar{x}_{(n)} \, d_{(n)} \right)$$



cost function

→ Gaussian Classifier (Bayes approach)

What happens since we cannot know the input? Statistics help us to describe the input to the algorithm.

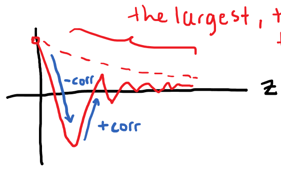


to see immediate tendency

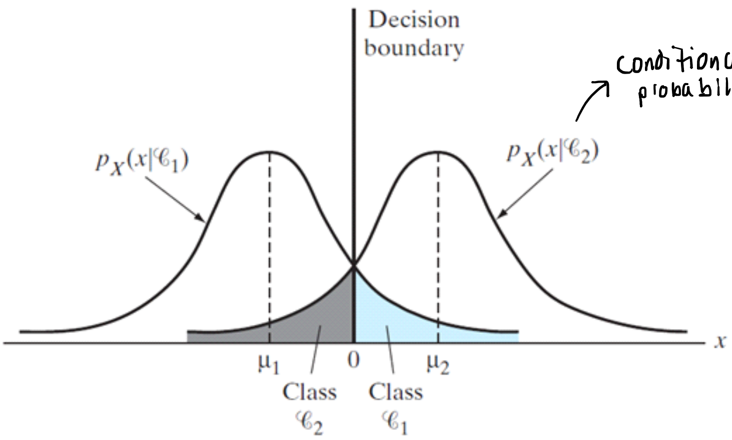
Correlation of $x(t)$ and $x(t+1)$: an important notion

$\sum x(n)\, x(n-z)$ xcorr in python

the largest, the longer previous time should be considered



↳ 'memory' : how previous should you consider?



conditional probability (Bayes)

The Bayesian approach thus informs the input data statistics to the algorithm also, so that the classifier improves.

→ Bayes Classifier – minimum average risk

$$\mathbb{R} = c_{11} P_1 \overbrace{\int_{\boxed{\mathcal{H}_1}} p_{\bar{x}}(\bar{x}|C_1)\, d\bar{x}}^{\text{prob of } \bar{x} \text{ given } C_1} + c_{22} P_2 \overbrace{\int_{\boxed{\mathcal{H}_2}} p_{\bar{x}}(\bar{x}|C_2)\, d\bar{x}}^{\text{prob of } \bar{x} \text{ given } C_2}$$

} correct ones

→ Regions (4)

$$+ \, c_{21} P_1 \int_{\boxed{\mathcal{H}_2}} p_{\bar{x}}(\bar{x}|C_1)\, d\bar{x} + c_{12} P_2 \int_{\boxed{\mathcal{H}_1}} p_{\bar{x}}(\bar{x}|C_2)\, d\bar{x}$$

} false negative/positives

$\bar{x}$ being C_1 but in region 2

$$\Lambda = \frac{p_{\bar{x}}(\bar{x}|C_1)}{\underbrace{p_{\bar{x}}(\bar{x}|C_2)}}$$

] If this is bigger than this → $\Lambda \geq 1$ → classified as C_1, etc.

↳ "The probability that an $\bar{x}$ value belongs to class C_2"

- Probability Density Function for the Gaussian Case

↳ "The probability that an $\bar{x}$ value belongs to class C_2"

- Probability Density Function for the Gaussian Case

$$P_{\bar{x}}(\bar{x}|C_i) = \frac{1}{(2\pi)^{m/2}(\det \mathbb{C})^{1/2}} e^{\wedge}\left(-\frac{1}{2}(\bar{x}-\mu_i)^T \mathbb{C}^{-1}(\bar{x}-\mu_i)\right), \quad i=1,2$$

* Note $\mathbb{C}$ is the correlation matrix

- Applying the logarithm of the function of verisimilitude

$$\log \Lambda(\bar{x}) = -\frac{1}{2}(\bar{x}-\mu_1)^T \mathbb{C}^{-1}(\bar{x}-\mu_1) + \frac{1}{2}(\bar{x}-\mu_2)\mathbb{C}^{-1}(\bar{x}-\mu_2)$$

$$= (\mu_1-\mu_2)^T \mathbb{C}^{-1}\bar{x} + \frac{1}{2}(\mu_2^T \mathbb{C}^{-1}\mu_2 - \mu_1^T \mathbb{C}^{-1}\mu_1)$$

- Leaving the neuron with the following weights (linear problem)

$$y = \bar{w}^T \bar{x} + b$$

$$\bar{y} = \log \Lambda(\bar{x})$$

$$\boxed{\begin{aligned} \bar{w} &= \mathbb{C}^{-1}(\mu_1-\mu_2) \\ b &= \frac{1}{2}(\mu_2^T \mathbb{C}^{-1}\mu_2 - \mu_1^T \mathbb{C}^{-1}\mu_1) \end{aligned}}$$

No iterations
$\mathbb{C}^{-1}$: correlation matrix of all input variables

- Do you have the relationship of all input variables?
  - ↳ YES: Bayesian approach (compute the corr matrix)
  - ↳ NO: Algorithmic approach