Overview (MLP) Overall function

$$y_k(\underline{x}, \underline{w}) = \sigma(a_k) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} \underbrace{h(a_j)}_{z_j}\right)$$

$\underline{w}$: vector of ALL weights in the network

$$= \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

Interpretation: A neural network is simply a non-linear function from a set of input variables $\underline{x}_{D\times 1}: \{x_i\}$, $i \in \{1, D\}$ to a set of output variables $\underline{y}_{K\times 1}: \{y_k\}$, $k \in \{1, K\}$

→ controlled by a vector $\underline{w}$ of adjustable parameters (weights).

* **Architectural Connotation**: gives a spatial computational mechanism to compute a function
* **NN: 'Function Approximator'**

Best: functional closed form e.g., $y(\theta) = \sin\theta$ or at least a closed form formula, with an associated computational procedure.

→ A table represents a function

e.g.: Boolean expression: 'Truth Table' – we exhaustively enumerate all possible inputs, and all possible corresponding outputs! (for an input, → exactly one output, else it would not be a function)

$\underline{x}_i$	$f(\underline{x}_i)$
$\underline{x}_1$	$f(\underline{x}_1)$
$\underline{x}_2$	$f(\underline{x}_2)$
⋮	⋮
$\underline{x}_N$	$f(\underline{x}_N)$

Boolean: it is possible to enumerate all inputs (and the outputs).

→ Not possible for a continuous or finely discretised case!

what if we are given an $\underline{x}_i$ which lies b/w two of the N "training" points $\underline{x}_1$ & $\underline{x}_2$?

→ we will need to interpolate/approximate the output value

OR: use a NN, which should give a "reasonable" output for an "unseen" input (an input which was not a part of the training set)

FRANK ROSENBLATT'S "PERCEPTRON"

[The basic perceptron is linear in $\underline{\phi}(\underline{x})$]

{ for simplicity, we may take $\underline{\phi}(\underline{x})$ as the unit function itself $\underline{\phi}(\underline{x}) = \underline{x}$

→ Two-class model/classifier, which classifies a point as class ± 1 (like an SVM formulation, classes as ± 1 elegant formulation, not 0/1)

$$y(\underline{x}) = f(\underbrace{\underline{w}^T \underline{\phi}(\underline{x})}_{\text{scalar}}), \quad f[a] = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

$\rightarrow t_i = +1$

$\rightarrow t_i = -1$

(the "=" case isn't "that" important)

x_i : [class $\mathcal{C}_1$ ($t_i = +1$) if $\underline{w}^T \underline{\phi}(\underline{x}_i) > 0$
class $\mathcal{C}_2$ ($t_i = -1$) if $\underline{w}^T \underline{\phi}(\underline{x}_i) < 0$]

→ One inequality $\boxed{t_i \, \underline{w}^T \underline{\phi}(\underline{x}_i) > 0}$

(SVM-like formulation)

[criterion for correct classification]

Perceptron Criterion ["this is ONE possibility for a learning algorithm"]

- Penalty for a correct classification = 0
- Incorrect classification: $t_i \, \underline{w}^T \underline{\phi}(\underline{x}_i) < 0$

⇒ To minimise $-t_i \, \underline{w}^T \underline{\phi}(\underline{x}_i)$

∀ data points

$$\boxed{E(\underline{w}) = -\sum_{i \in M} t_i \, \underline{w}^T \underline{\phi}(\underline{x}_i)}$$

→ M: misclassified pattern

→ this formulation does not penalise 'by how much' a pattern is misclassified

$f(\cdot)$ is typically non-linear,

→ generally solved numerically

$$\boxed{\underline{w}^{(\tau+1)} = \underline{w}^{(\tau)} - \eta \nabla E(\underline{w})}$$

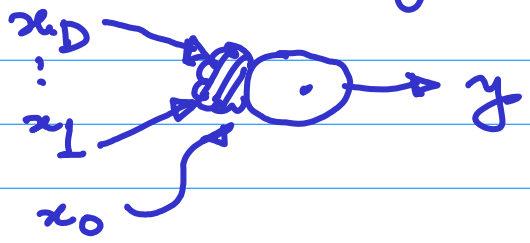
- $(\tau+1)$th iteration of the weight computation
- why?
- learning rate
- numerical gradient

Limitations

* Numerical difficulties with the learning algorithm
* Does not generalise well to $K > 2$ classes
* Based on a linear combination of fixed basis functions.
* Generalisation (Perceptron → MLP) more layers, or feature transformation $\underline{\phi}(\underline{x})$

The 'perceptron' learning mechanism (algorithm) does not necessarily have an architectural connotation

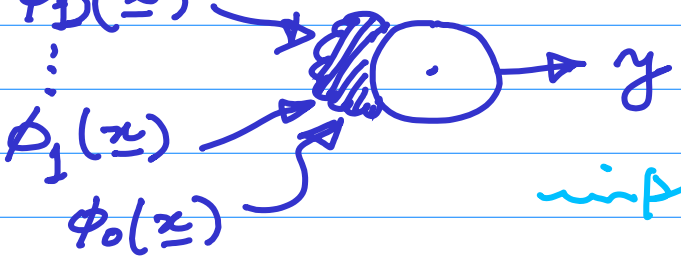
→ BUT can also be explained with a single layer neural network



$f(\underline{w}^T \underline{\phi}(\underline{x}_i))$

architectural connotation

or



(NN implementation)

Example: $\underline{x}$ (D-dimensional) which has terms (scalars) such as the salary, years of residence, outstanding debt, ...

Output $y = \{+1, -1\}$

$+1 \rightarrow$ approve a credit / loan / credit card

$-1 \rightarrow$ deny it

Rule: approve credit if $\sum_{i=1}^{D} w_i x_i >$ threshold

(threshold $= -b$ or $-w_0$)

deny credit if $\sum_{i=1}^{D} w_i x_i <$ threshold

i.e., $h(\underline{x}) \triangleq \text{sgn}\left(\sum_{i=1}^{D} w_i x_i + b\right)$

(linear model)

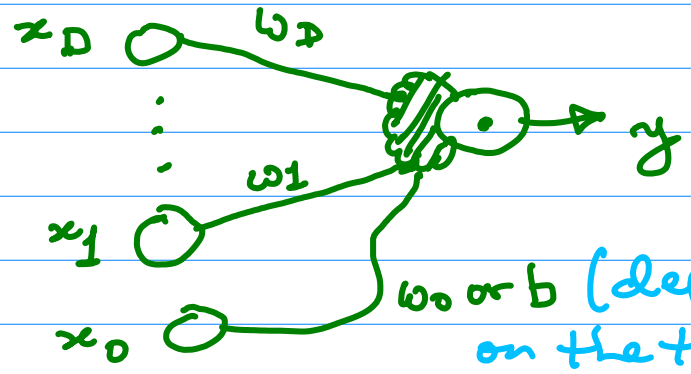
$y_i = y_i(\underline{x}) = y_i(\underline{x}, \underline{w}) = \underline{w}^T \underline{x}_i + b$ or $\underline{w}^T \underline{x}_i + w_0$ OR $\underline{w}^T \underline{x}$

($\underline{w}$: parameters (of our linear model))

("bias" / non-homogeneous representation)

(homogeneous form — indices are 0..D)

Architectural interpretation

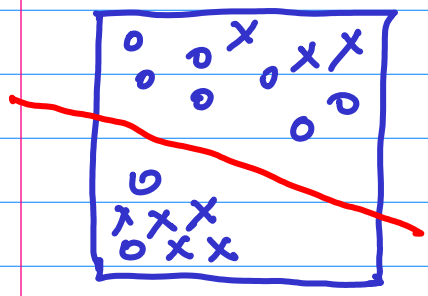


$h(a) = +1, a > 0, t_i = +1$

$= -1, a < 0, t_i = -1$

Learning problem: Learn the weights.

Some weights will end up negative (this parameter will have an adverse effect on the credit approval / credit rating)
e.g., outstanding debt



$$y = y(\underline{x}, \underline{w}) = \underline{w}^T \underline{x} + b$$