Neural Networks

Stefano Gualandi
Università di Pavia, Dipartimento di Matematica

email: stefano.gualandi@unipv.it
twitter: @famo2spaghi
blog: http://stegua.github.com
web: http://matematica.unipv.it/gualandi/opt4ml

Supervised Learning

Definition 1 (Supervised Learning)

Supervised Learning is the task of learning (inferring) a function f that maps input vectors to their corresponding target vectors, by using a dataset containing a given set of pairs of $(input, output)$ samples. Examples:

- Regression: the output vectors take one or more continuous values.
- Classification: the output vectors take one value of a finite number of discrete categories. Special case: binary classification.

Neural Networks

Biological Motivation

1. The human brain is estimated to contain a densely interconnected network of around 10^{11} neurons

Biological Motivation

1. The human brain is estimated to contain a densely interconnected network of around 10^{11} neurons
2. Each neuron is connected to around others 10^4 neurons. Neurons activity is typically excited or inhibited through connections to other neurons.

Biological Motivation

1. The human brain is estimated to contain a densely interconnected network of around 10^{11} neurons
2. Each neuron is connected to around others 10^4 neurons. Neurons activity is typically excited or inhibited through connections to other neurons.
3. The switching of neurons is of order of 10^{-3} seconds, much slower than modern computers, which have switching speeds of 10^{-10} seconds

Biological Motivation

1. The human brain is estimated to contain a densely interconnected network of around 10^{11} neurons
2. Each neuron is connected to around others 10^4 neurons. Neurons activity is typically excited or inhibited through connections to other neurons.
3. The switching of neurons is of order of 10^{-3} seconds, much slower than modern computers, which have switching speeds of 10^{-10} seconds
4. Humans take around 10^{-1} seconds to recognize a friend (compare to switching speed...)

Biological Motivation

1. The human brain is estimated to contain a densely interconnected network of around 10^{11} neurons
2. Each neuron is connected to around others 10^4 neurons. Neurons activity is typically excited or inhibited through connections to other neurons.
3. The switching of neurons is of order of 10^{-3} seconds, much slower than modern computers, which have switching speeds of 10^{-10} seconds
4. Humans take around 10^{-1} seconds to recognize a friend (compare to switching speed...)
5. Human brain performs a highly parallel tasks

Hollywoodian Neural Networks

John Connor: Can you learn stuff you haven't been programmed with so you could be... you know, more human? And not such a dork all the time?

Hollywoodian Neural Networks

John Connor: Can you learn stuff you haven't been programmed with so you could be... you know, more human? And not such a dork all the time?

The Terminator: **My CPU is a neural-net processor; a learning computer**.

Hollywoodian Neural Networks

John Connor: Can you learn stuff you haven't been programmed with so you could be... you know, more human? And not such a dork all the time?

The Terminator: **My CPU is a neural-net processor; a learning computer**. But Skynet presets the switch to read-only when we're sent out alone.

Hollywoodian Neural Networks

John Connor: Can you learn stuff you haven't been programmed with so you could be... you know, more human? And not such a dork all the time?

The Terminator: **My CPU is a neural-net processor; a learning computer**. But Skynet presets the switch to read-only when we're sent out alone.

Sarah Connor: Doesn't want you doing too much thinking, huh?

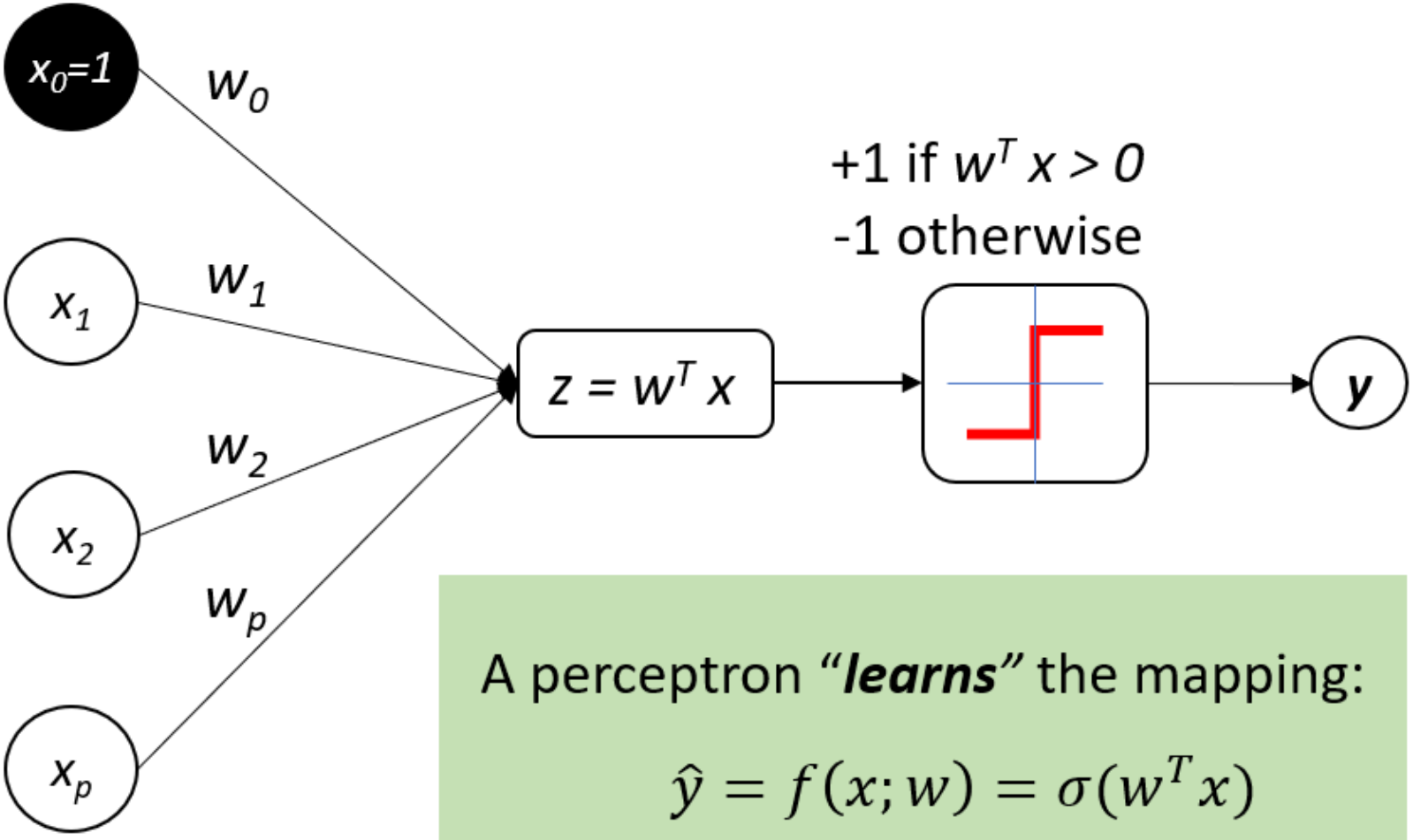
The Terminator: No.

Comics Neural Networks

Thanks to machine-learning algorithms,
the robot apocalypse was short-lived.

Let us start simple: Single Perceptron

Perceptron

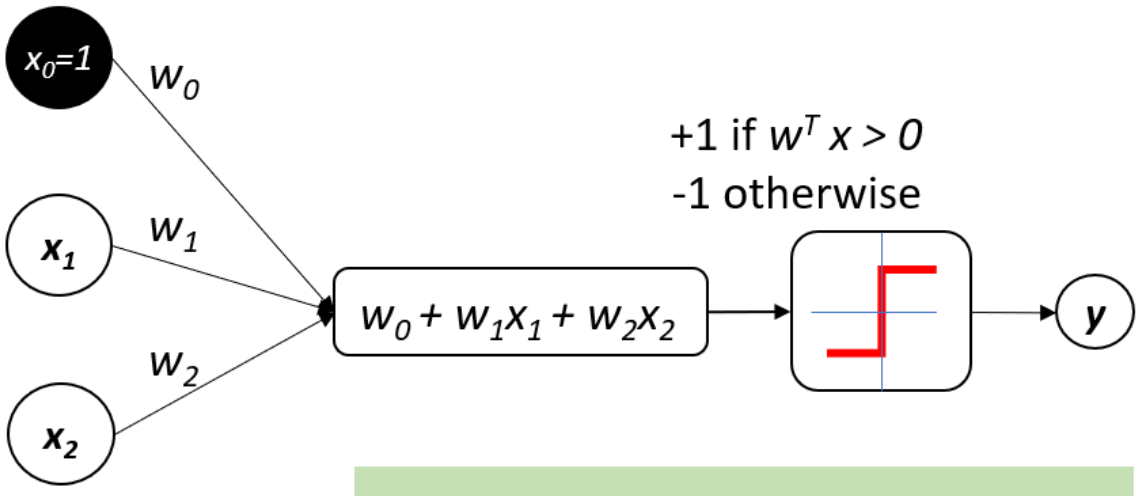


A perceptron "***learns***" the mapping:

$$\hat{y} = f(x; w) = \sigma(w^T x)$$

Perceptron

Logical AND		
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

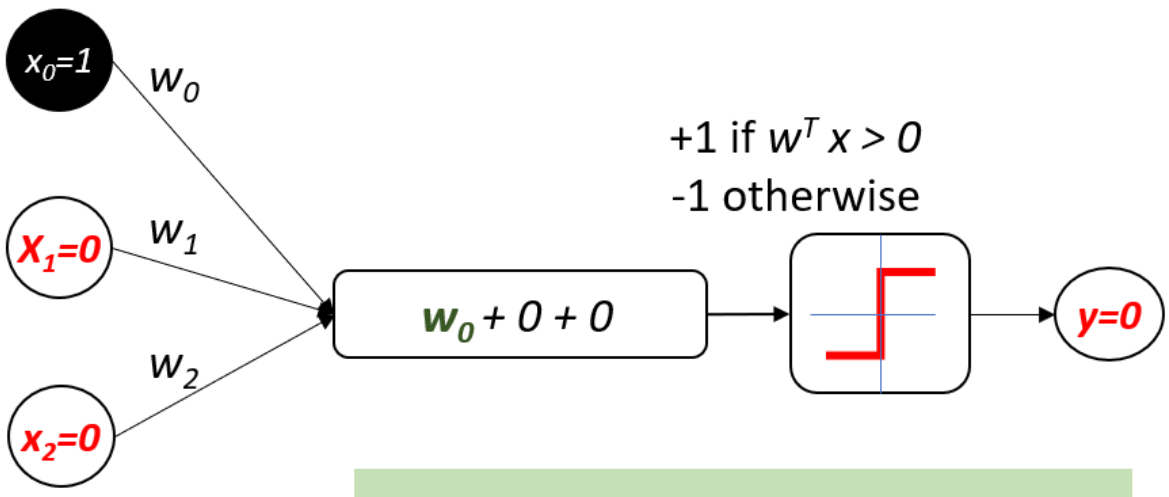


A perceptron "***learns***" the mapping:

$$\hat{y} = f(x; w) = \sigma(w^T x)$$

Perceptron

Logical AND		
x_1	x_2	y
0	**0**	**0**
0	1	0
1	0	0
1	1	1

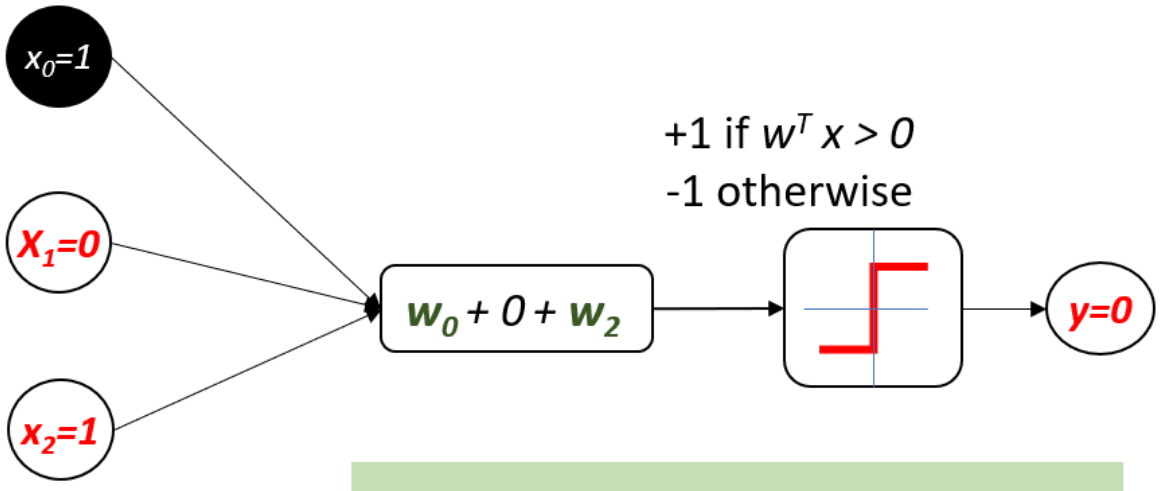


A perceptron "***learns***" the mapping:

$$\hat{y} = f(x; w) = \sigma(w^T x)$$

Perceptron

Logical AND		
x_1	x_2	y
0	0	0
0	**1**	**0**
1	0	0
1	1	1

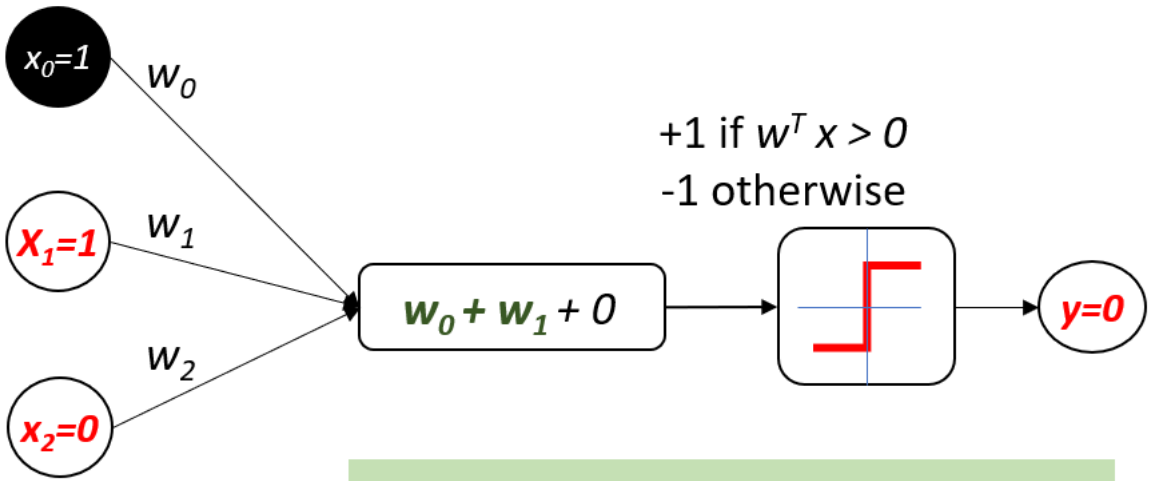


A perceptron "***learns***" the mapping:

$$\hat{y} = f(x; w) = \sigma(w^T x)$$

Perceptron

Logical AND		
x_1	x_2	y
0	0	0
0	1	0
1	**0**	**0**
1	1	1

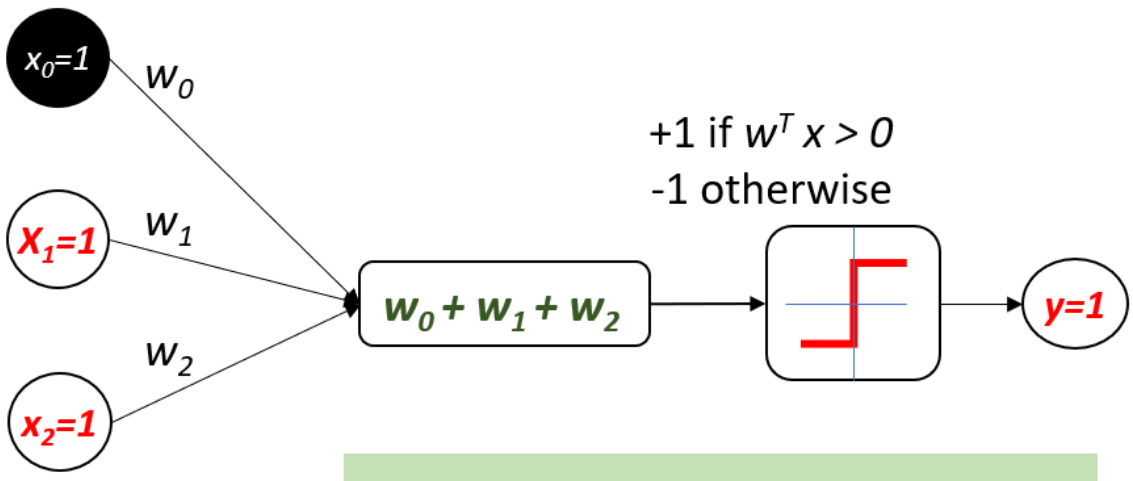


A perceptron "***learns***" the mapping:

$$\hat{y} = f(x; w) = \sigma(w^T x)$$

Perceptron

Logical AND		
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	**1**	**1**



A perceptron "***learns***" the mapping:

$$\hat{y} = f(x; w) = \sigma(w^T x)$$

Perceptron

AND		
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

OR		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

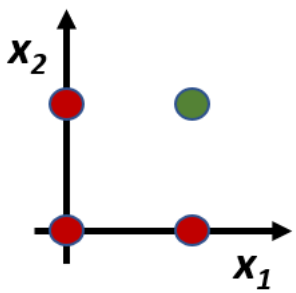
XOR		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

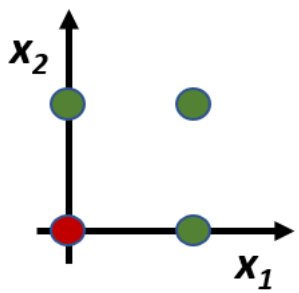
Perceptron

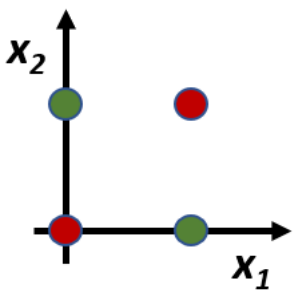
AND		
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

OR		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0





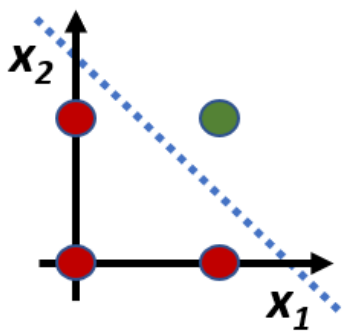


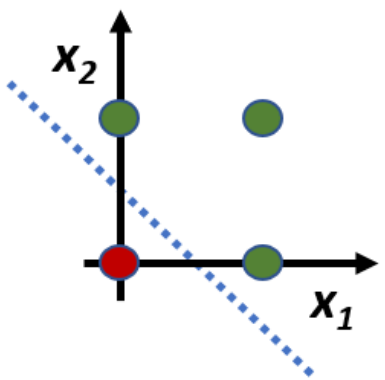
Perceptron

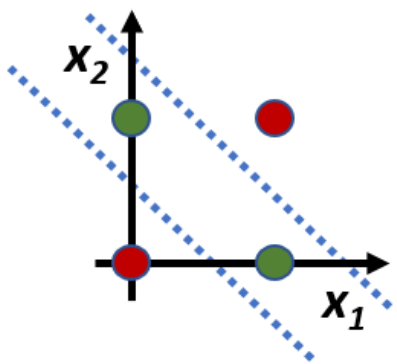
AND		
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

OR		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0







Learning: Minimize the Loss Function

A simple choice of loss function is:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{2} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{1}$$

where, for the Perceptron Network, we have

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{w}) = \text{step}\left(\boldsymbol{w}^t \boldsymbol{x}\right) \tag{2}$$

Learning: Minimize the Loss Function

A simple choice of loss function is:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{1}$$

where, for the Perceptron Network, we have

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{w}) = \text{step}\left(\boldsymbol{w}^t \boldsymbol{x}\right) \tag{2}$$

Given $(\boldsymbol{x}_i, y_i), i = 1, \ldots, N$, the learning task become

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} L(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{w})) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \text{step}\left(\boldsymbol{w}^T \boldsymbol{x}_i\right)\right)^2 \tag{3}$$

Learning: Minimize the Loss Function

A simple choice of loss function is:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{1}$$

where, for the Perceptron Network, we have

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{w}) = \text{step}\left(\boldsymbol{w}^t \boldsymbol{x}\right) \tag{2}$$

Given $(\boldsymbol{x}_i, y_i), i = 1, \ldots, N$, the learning task become

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} L(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{w})) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \text{step}\left(\boldsymbol{w}^T \boldsymbol{x}_i\right)\right)^2 \tag{3}$$

and hence we need to solve

$$\nabla L(\boldsymbol{y}, \text{step}\left(\boldsymbol{w}^T \boldsymbol{x}_i\right)) = 0 \tag{4}$$

Learning: Minimize the Loss Function

The directional derivative of L w.r.t. w_j is

$$\frac{\partial L(\boldsymbol{y}, \text{step}\left(\boldsymbol{w}^T \boldsymbol{x}_i\right))}{\partial w_j} = \sum_{i=1}^{N} (y_i - \hat{y}_i)\, \boldsymbol{x}_{ij} \tag{5}$$

Learning: Minimize the Loss Function

The directional derivative of L w.r.t. w_j is

$$\frac{\partial L(\boldsymbol{y}, \text{step}\left(\boldsymbol{w}^T \boldsymbol{x}_i\right))}{\partial w_j} = \sum_{i=1}^{N} (y_i - \hat{y}_i)\, \boldsymbol{x}_{ij} \tag{5}$$

and, hence, if we perform a gradient descent algorithm on the weight, we must iterate over:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma \sum_{i=1}^{N} (y_i - \hat{y}_i)\, \boldsymbol{x}_i \tag{6}$$

where γ is the step size (aka, *learning rate*).

Training: Backpropagation Algorithm

Algorithm 1: Single Perceptron Learning

Data: $\boldsymbol{X} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ input vectors, $x_i \in \mathbb{R}^{P-1}$
Data: $\boldsymbol{y} = y_1, \ldots, y_K$ target vectors
Data: γ learning rate
Result: $\boldsymbol{w}^*$ optimal weight of $f(\boldsymbol{x}; \boldsymbol{w})$

$\boldsymbol{w} \leftarrow 0_P$;
for $it \leftarrow 1$ **to** *maxiter* **do**
    $\hat{\boldsymbol{y}} \leftarrow f(\boldsymbol{x}; \boldsymbol{w})$;
    $E \leftarrow \boldsymbol{y} - \hat{\boldsymbol{y}}$;
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma \sum_{i=1}^{N} E_i \boldsymbol{x}_i$;
    if $||E|| \approx 0$ **then**
        break;
return $\boldsymbol{w}$

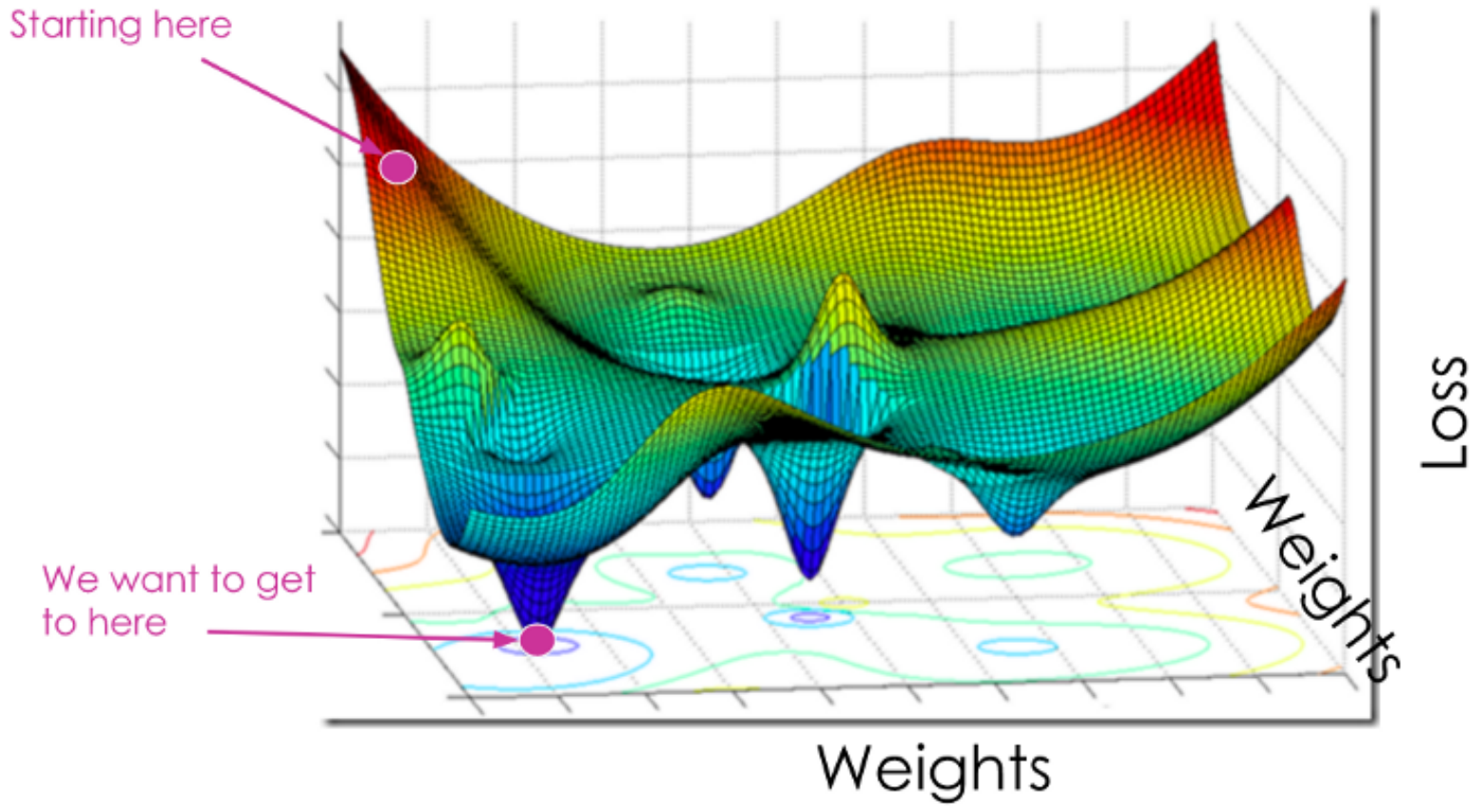
Exercise: Homer's Perceptron

Exercise 1: Implement a Perceptron Neural Network, and train the same networks 3 times:

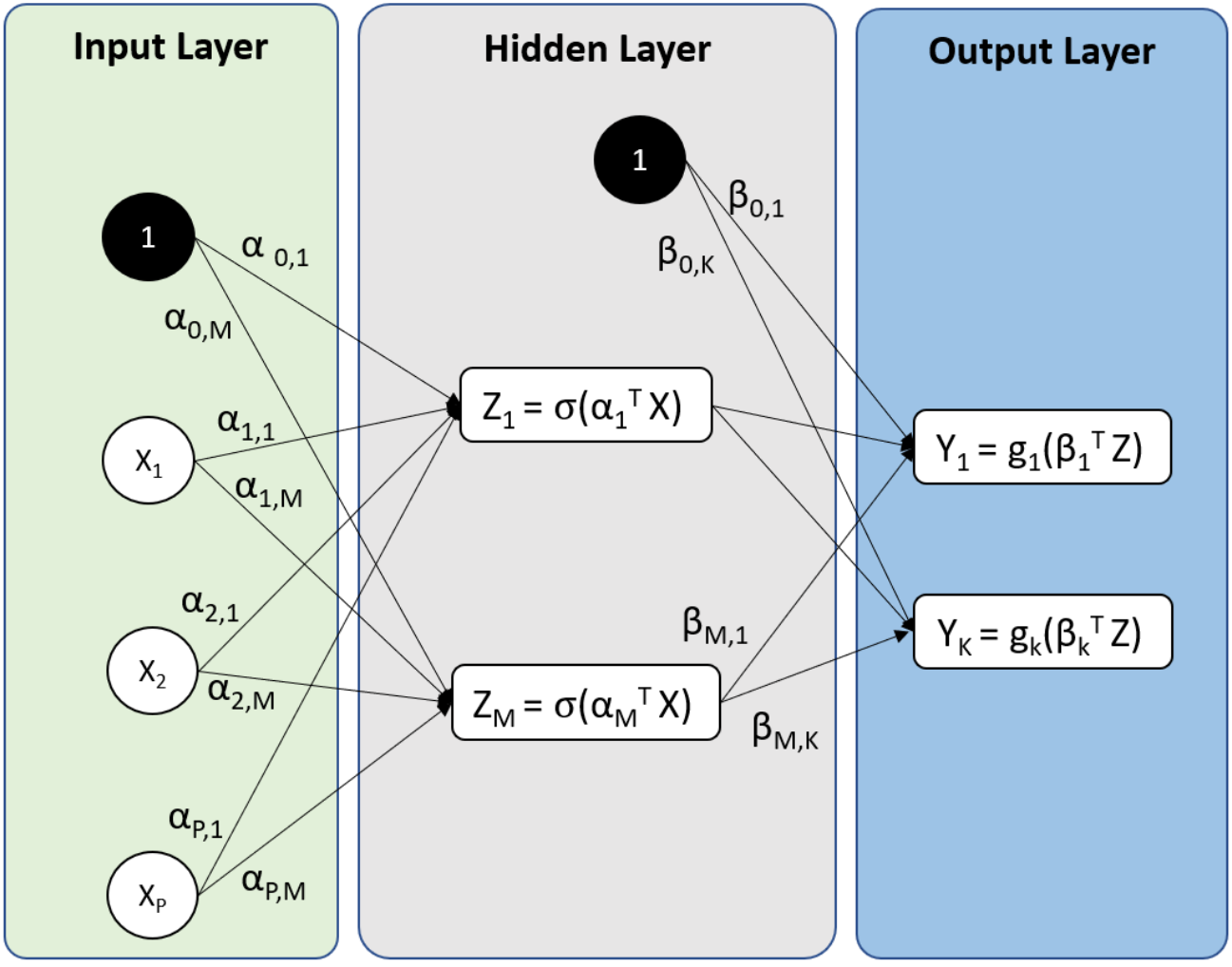
1. Train to learn the `AND` logical function
2. Train to learn the `OR` logical function
3. Train to learn the `XOR` logical function

Discuss how to initialize the weights and the *learning rate* (i.e., the step size in gradient descent).

Loss Function Landscape (micro-example)



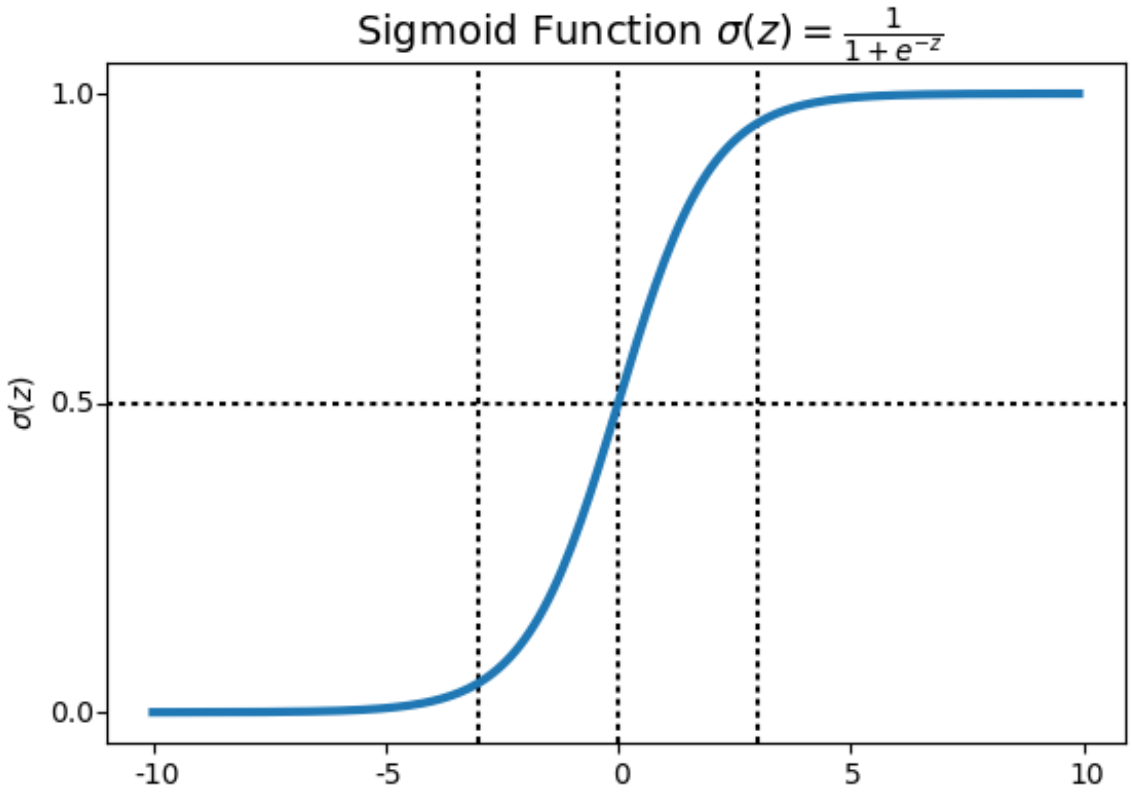
Multilayer Neural Network



$\alpha \in R^{P \times M}$

$\beta \in R^{M \times K}$

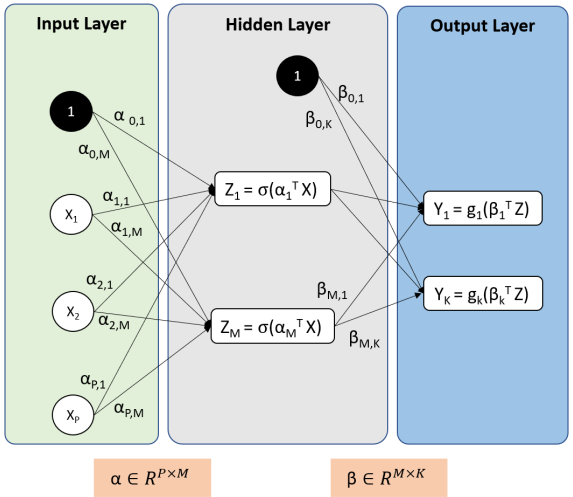
Activation Function: Sigmoid



Other activation functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Multi Layer Networks



Nelle reti neurali a due livelli, per un problema di classificazione a K classi, con M neuroni nel livello nascosto, abbiamo che

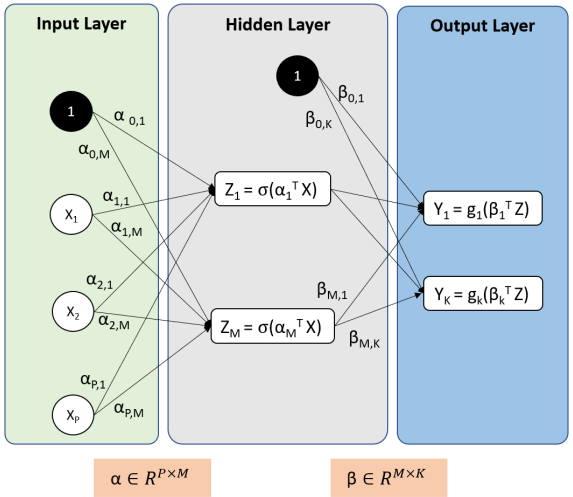
$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T x), \quad m = 1, \ldots, M \tag{7}$$

$$Y_k = g_k(\beta_{0k} + \beta_k^T Z), \quad k = 1, \ldots, K \tag{8}$$

in cui $Z = (Z_1, \ldots, Z_M)$, e $\sigma(v)$ è una funzione di attivazione, di solito posta pari a

$$\sigma(v) = \frac{1}{1 + e^{-v}}$$

Funzione di output



La funzione di ouput $g_k(v)$ permette una trasformazione finale del vettore degli output.

Per i problemi di regressione si usa di solito la funzione $g_k(v) = v$.

Per i problemi di classificazione si usa la stessa funzione *softmax* usata per la regressione logistica

$$g_k(v) = \frac{e^{v_k}}{\sum_{l=1}^{K} e^{v_l}}$$

Fitting di Reti a due strati: Parametri

Denotiamo l'insieme dei pesi di una rete neurale con θ, che consiste di

$$\alpha_{0m}, \alpha_m; m = 1, 2, \ldots, M : \text{sono } M(p+1) \text{ pesi} \quad (9)$$

$$\beta_{0k}, \beta_k; k = 1, 2, \ldots, K : \text{sono } K(M+1) \text{ pesi} \quad (10)$$

Fitting di Reti a due strati: Parametri

Denotiamo l'insieme dei pesi di una rete neurale con θ, che consiste di

$$\alpha_{0m}, \alpha_m; m = 1, 2, \ldots, M : \text{sono } M(p+1) \text{ pesi} \quad (9)$$

$$\beta_{0k}, \beta_k; k = 1, 2, \ldots, K : \text{sono } K(M+1) \text{ pesi} \quad (10)$$

Per i problemi di regressione si usa come *Loss function*

$$L(\alpha, \beta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

Fitting di Reti a due strati: Parametri

Denotiamo l'insieme dei pesi di una rete neurale con θ, che consiste di

$$\alpha_{0m}, \alpha_m; m = 1, 2, \ldots, M : \text{sono } M(p+1) \text{ pesi} \quad (9)$$

$$\beta_{0k}, \beta_k; k = 1, 2, \ldots, K : \text{sono } K(M+1) \text{ pesi} \quad (10)$$

Per i problemi di regressione si usa come *Loss function*

$$L(\alpha, \beta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

Per i problemi di classificazione, oltre alla precedente, si usa la Loss function

$$L(\alpha, \beta) = -\sum_{k=1}^{K} \sum_{i=1}^{N} y_{ik} \log f_k(x_i)$$

e il classificatore restituisce $G(x) = \arg\max f_k(x_i)$.

Fitting di Reti a due strati: Back-propagation

L'approccio più generale per minimizzare la Loss function è attraverso metodi del gradiente, che in questo contesto vengono chiamate di Back-propagation, che nel caso di errore quadratico medio viene calcolato come segue.

$$L(\alpha,\beta) = \sum_{i=1}^{N} L_i = \sum_{k=1}^{K}\sum_{i=1}^{N}(y_{ik} - f_k(x_i))^2 \tag{11}$$

con derivate

$$\frac{\partial L_i}{\partial \beta_{km}} = -2\,(y_{ik} - f_k(x_i))\,g'(\beta_k^T Z_i) Z_{mi}, \tag{12}$$

$$\frac{\partial L_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))\,g'(\beta_k^T Z_i)\beta_{km}\sigma'(\alpha_m^T x_i) X_{il}. \tag{13}$$

Fitting di Reti a due strati: Back-propagation

Date le derivate prima precedenti, le regole di aggiornamento per un metodo di gradiente sono

$$\beta_{km}^{r+1} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial L_i}{\partial \beta_{km}^{(r)}} \tag{14}$$

$$\alpha_{ml}^{r+1} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial L_i}{\partial \alpha_{ml}^{(r)}} \tag{15}$$

in cui γ_r è il *learning rate*.

Fitting di Reti a due strati: Back-propagation

Se ora scriviamo le derivate parziali come

$$\frac{\partial L_i}{\partial \beta_{km}} = \delta_{ki} Z_{mi}, \tag{16}$$

$$\frac{\partial L_i}{\partial \alpha_{ml}} = s_{mi} X_{il}. \tag{17}$$

le quantità δ_{ki} e s_{mi} sono gli *errori* del modello corrente allo strato nascosto e a quello di output. Possiamo scrivere

$$s_{mi} = \sigma'(\alpha_m^T X_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki},$$

che viene chiamata l'equazione di *back-propagation*.

Fitting di Reti a due strati: Back-propagation

Il calcolo dei pesi θ viene eseguito con un algoritmo che ha due fasi.

1. Nella prima fase (*forward pass*) si calcolano i valori predetti di $\hat{f}(x_i)$ con l'equazioni che definiscono la rete.
2. Nella seconda fase (*backward pass*) si calcolano gli errori δ_{ki} che vengono poi propagati all'indietro per trovare gli errori s_{mi}.
3. Alla fine gli errori δ_{ki} e s_{mi} sono usati per calcolare i gradienti per le regole di aggiornamento.

I vantaggi di questo metodo sono la semplicità e il fatto di essere "locale": ogni neurone passa e riceve informazioni solo dai neuroni con cui condivide una connessione, e quindi si presta a metodi di calcolo parallelo.

Fitting di Reti a due strati: Learning rate

Il learning rate γ_r di solito viene preso costante (determinato in maniera empirica).

In teoria per garantire convergenza, si dovrebbe avere

$$\gamma_r \rightarrow 0, \tag{18}$$

$$\sum_r \gamma_r = \infty, \tag{19}$$

$$\sum_r \gamma_r^2 < \infty. \tag{20}$$

Queste richieste vengon soddisfatte per esempio da $\gamma_r = \frac{1}{r}$

Exercise 2: T2's Perceptron

Exercise 2: Implement a Multilayer Neural Network, and train the same networks 3 times:

1. Train to learn the `XOR` logical function
2. The Levefre's Boolean function
3. The identiy function over 8 bits, i.e., $(0,1,0,0,0,0,0,0) \rightarrow (0,1,0,0,0,0,0,0)$. Look at the internal weights and draw some conclusions.

Julia: Editor + Shell