

Training Neural Networks

Stefano Gualandi

Università di Pavia, Dipartimento di Matematica

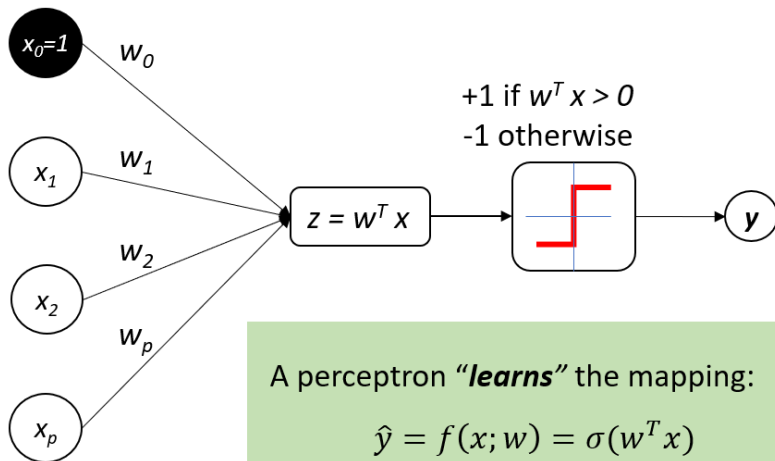
email: stefano.gualandi@unipv.it
twitter: [@famo2spaghi](https://twitter.com/famo2spaghi)
blog: <http://stegua.github.com>

Comics Neural Networks



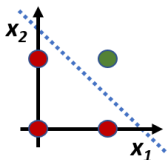
Thanks to machine-learning algorithms,
the robot apocalypse was short-lived.

Perceptron

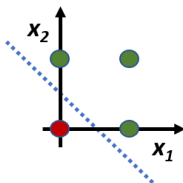


Perceptron

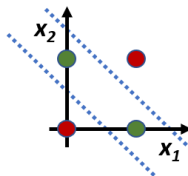
AND		
x_1	x_2	y
0	0	0 ●
0	1	0 ●
1	0	0 ●
1	1	1 ●



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:

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

where, for the Perceptron Network, we have

$$\hat{\mathbf{y}} = f(\mathbf{x}; \mathbf{w}) = \text{step}(\mathbf{w}^t \mathbf{x}) \quad (2)$$

Learning: Minimize the Loss Function

A simple choice of loss function is:

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

where, for the Perceptron Network, we have

$$\hat{\mathbf{y}} = f(\mathbf{x}; \mathbf{w}) = \text{step}(\mathbf{w}^T \mathbf{x}) \quad (2)$$

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

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

Learning: Minimize the Loss Function

A simple choice of loss function is:

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

where, for the Perceptron Network, we have

$$\hat{\mathbf{y}} = f(\mathbf{x}; \mathbf{w}) = \text{step}(\mathbf{w}^T \mathbf{x}) \quad (2)$$

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

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

and hence we need to solve

$$\nabla L(\mathbf{y}, \text{step}(\mathbf{w}^T \mathbf{x}_i)) = 0 \quad (4)$$

Learning: Minimize the Loss Function

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

$$\frac{\partial L(\mathbf{y}, \text{step}(\mathbf{w}^T \mathbf{x}_i))}{\partial w_j} = \sum_{i=1}^N (y_i - \hat{y}_i) x_{ij} \quad (5)$$

Learning: Minimize the Loss Function

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

$$\frac{\partial L(\mathbf{y}, \text{step}(\mathbf{w}^T \mathbf{x}_i))}{\partial w_j} = \sum_{i=1}^N (y_i - \hat{y}_i) x_{ij} \quad (5)$$

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

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

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

Training: Backpropagation Algorithm

Algorithm 1: Single Perceptron Learning

Data: $\mathbf{X} = \mathbf{x}_1, \dots, \mathbf{x}_N$ input vectors, $\mathbf{x}_i \in \mathbb{R}^{P-1}$

Data: $\mathbf{y} = y_1, \dots, y_K$ target vectors

Data: γ learning rate

Result: $\mathbf{w}^*$ optimal weight of $f(\mathbf{x}; \mathbf{w})$

```
1  $\mathbf{w} \leftarrow \mathbf{0}_P$ ;  
2 for  $it \leftarrow 1$  to maxiter do  
3    $\hat{\mathbf{y}} \leftarrow f(\mathbf{x}; \mathbf{w})$ ;  
4    $E \leftarrow \hat{\mathbf{y}} - \mathbf{y}$ ;  
5    $\mathbf{w} \leftarrow \mathbf{w} - \gamma \sum_{i=1}^N E_i \mathbf{x}_i$ ;  
6   if  $\|E\| \approx 0$  then  
7     break;  
8 return  $\mathbf{w}$ 
```

Exercise: Homer's Perceptron

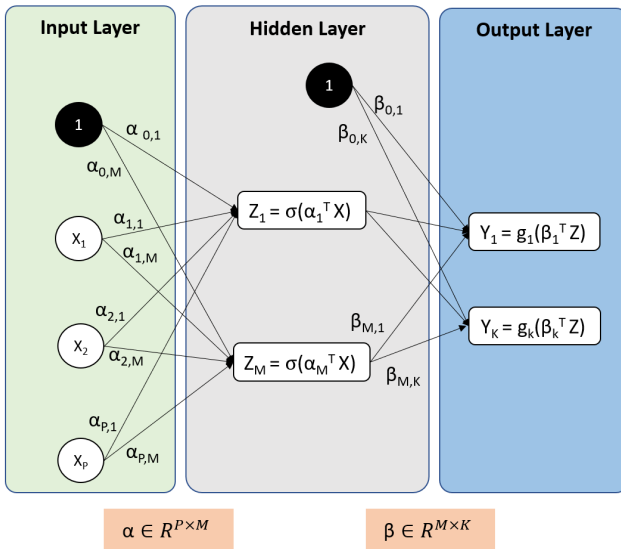


Exercise 1: Implement a Perceptron Neural Network, and train the same networks 3 times:

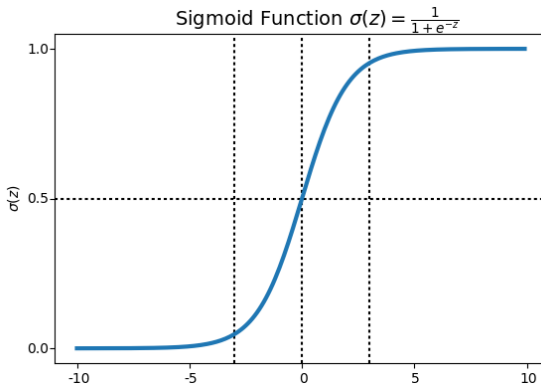
- ① Train to learn the AND logical function
- ② Train to learn the OR logical function
- ③ Train to learn the XOR logical function

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

Multilayer Neural Network



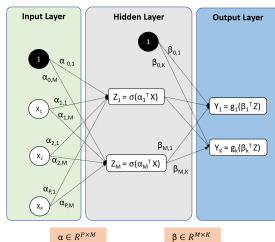
Activation Function: Sigmoid



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)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric 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

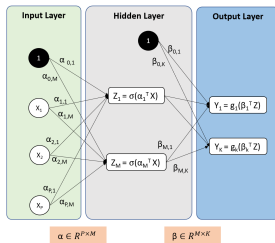
$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T x), \quad m = 1, \dots, M \quad (7)$$

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

in cui $Z = (Z_1, \dots, Z_M)$, 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 output $g_k(v)$ 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

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

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

Fitting di Reti a due strati: Parametri

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

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

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

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

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

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

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.

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

con derivate

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

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

Fitting di Reti a due strati: Back-propagation

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

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

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

in cui γ_r è il *learning rate*.

Fitting di Reti a due strati: Back-propagation

Se ora scriviamo le derivate parziali come

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

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

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 *locali*: 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

$$\gamma_r \rightarrow 0, \quad (18)$$

$$\sum_r \gamma_r = \infty, \quad (19)$$

$$\sum_r \gamma_r^2 < \infty. \quad (20)$$

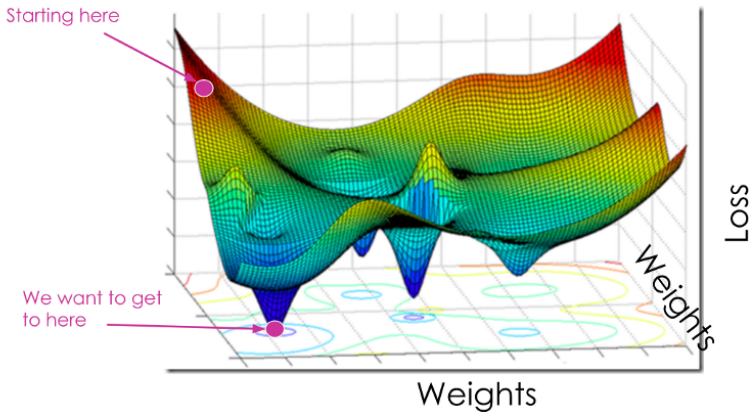
Queste richieste sono soddisfatte per esempio da $\gamma_r = \frac{1}{r}$

Backpropagation as Automatic Differentiation

DEMO ON LEARNING WEIGHTS: Backpropagation Algorithms

<https://www.youtube.com/watch?v=Ilg3gGewQ5U>

Loss Function Landscape (micro-example)



Exercise 2



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 identity 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.



Getting Started with Deep Learning

An Introduction to Deep Learning

Duration:
8 Hours

Price:
\$90.00

You are enrolled in this course

Businesses worldwide are using artificial intelligence (AI) to solve their greatest challenges. Healthcare professionals use AI to enable more accurate, faster diagnoses in patients. Retail businesses use it to offer personalized customer shopping experiences. Automakers use it to make personal vehicles, shared mobility, and delivery services safer and more efficient. Deep learning is a powerful AI approach that uses multi-layered artificial neural networks to deliver state-of-the-art accuracy in tasks such as object detection, speech recognition, and language translation. Using deep learning, computers can learn and recognize patterns from data that are considered too complex or subtle for expert-written software.

In this course, you'll learn how deep learning works through hands-on exercises in computer vision and natural language processing. You'll train deep learning models from scratch, learning tools and tricks to achieve highly accurate results. You'll also learn to leverage freely available, state-of-the-art pre-trained models to save time and get your deep learning application up and running quickly.

Subject:

Fundamental, Computer Vision

Tags:

data science
computer vision &
machine vision
education & training
ai / deep learning
big data & data mining

Select PayPal as payment method, and apply PROMOCODE:

DLITEACH1222_17_MSya_44

<https://courses.nvidia.com/courses/course-v1:>

[DLI+S-FX-01+V1/](https://courses.nvidia.com/courses/course-v1:DLI+S-FX-01+V1/)

Thanks for your attention!

