[course site]

#DLUPC

Day 1 Lecture 3

Perceptron

Antonio Bonafonte
antonio.bonafonte@upc.edu

Associate Professor
Universitat Politecnica de Catalunya

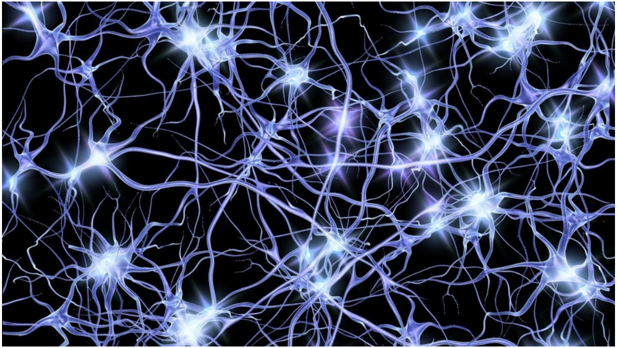
Index

ANN: Artificial Neural Networks

Proposed in 1965.

Renaissance $\approx$ 2010 (Deep Learning)

Inspired on biological Neural Networks



Biological Neural Networks

Human Brain

86 billion neurons

In average: 7 000 synapses (connexions)

Purkange cells:

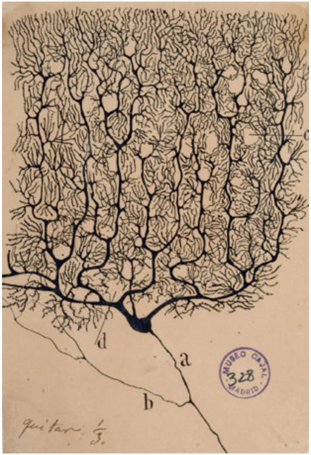
Input: 1000 dentritic branches, tens of thousands of connexions

Output: 100 connexions

Total synapses: $\approx 10^{12}$

300 million synapses per mm^3

Plasticity (learning)



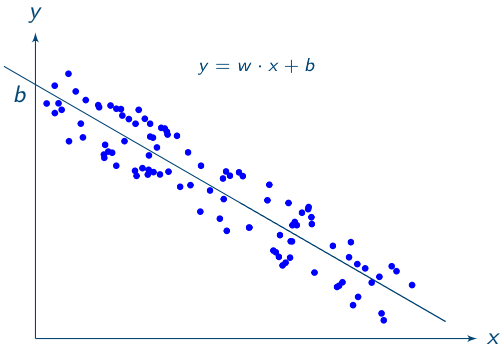
Ramon y Cajal Drawings

Notation

$(\mathbf{X}, \mathbf{Y})$	training examples
$\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_N$	input features
$\mathbf{x}_n$	vector with components $\mathbf{x}_n(j) \in \mathbb{R}$ or $\mathbf{x}_n(j) \in \{0, 1\}$)
$\mathbf{Y} = y_1, \ldots, y_N$	labels (regression: $\mathbb{R}$; class. $\{0, 1\}$)
$\hat{y} = f_\theta(\mathbf{x})$	prediction by model $f_\theta(.)$

Input features and output: binary (1,0), *one-hot* encoding or *normalized* real-values.

Basic component: linear regression



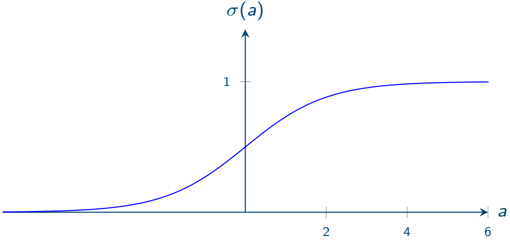
$$y = \mathbf{w}^T \cdot \mathbf{x} + b \equiv \bar{\mathbf{w}}^T \cdot \bar{\mathbf{x}}$$

(In the plot, **x** is a escalar value)

Example: x: mean speed; y: fuel comsumption km/l

Basic component: logistic regression

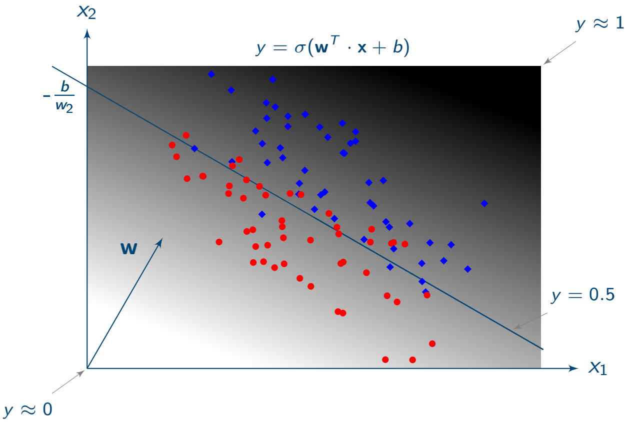
$$y = \sigma(\mathbf{w}^T \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^\mathbf{T} \cdot \mathbf{x} + b)}}$$



$$a = \mathbf{w}^T \cdot \mathbf{x} + b = \|\mathbf{w}\| \cdot \|\mathbf{x}\| \cdot \cos(\alpha) + b$$

Small $\|\mathbf{w}\|$ $\rightarrow$ smoother transition boundaries.

Basic component: logistic regression

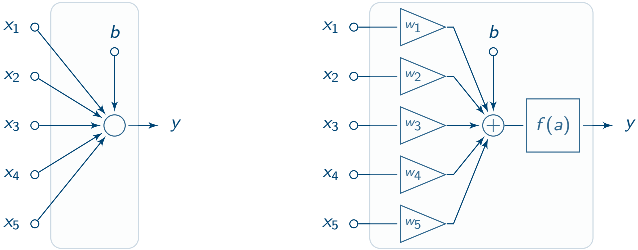


$$\theta = \{\mathbf{w}, b\} \quad \mathbf{w} = (w_1, w_2)^T$$

Example: x: (`mean_spectrum, mean_f0`); y: `IS_MUSIC`

Perceptron (Neuron)

Activation: $a = \mathbf{w}^T \cdot \mathbf{x} + b$ Output: $y = f(a)$



$f(.)$: activation function

Linear regression: $f(a) = a$

Logistic regression: $f(a) = \sigma(a)$

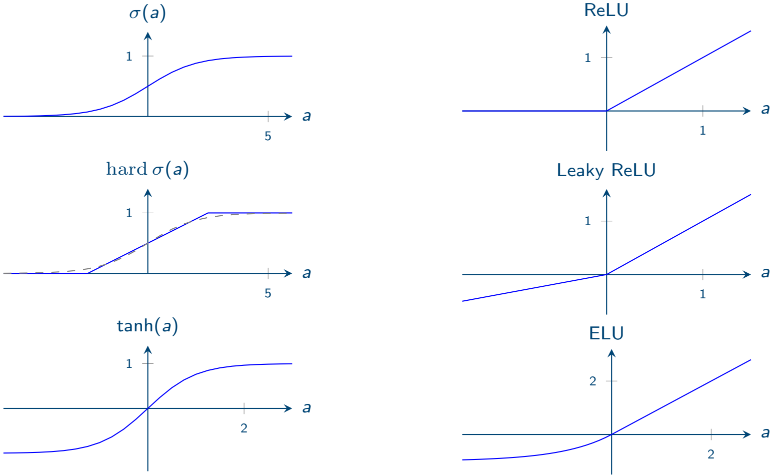
Activation functions

Activation function (non-linearities) are fundamental to combine neurons to model complex functions.

Some activation functions:

- Sigmoid $\sigma(.)$, $\tanh(.) = 2 \cdot \sigma(.) - 1$
- ReLU (rectified linear unit), Leaky ReLU, Parametric Leaky ReLU, ELU (exponential linear unit)
- softmax, maxout

Some Activation functions



Training

Training: estimate the parameters θ of model $f_\theta()$ from training data.

$(\mathbf{X}, \mathbf{Y})$	training examples
$\mathbf{X} = \mathbf{x}_1, \dots, \mathbf{x}_N$	input features
$\mathbf{x}_n$	vector with components $\mathbf{x}_n(j) \in \mathbb{R}$ or $\mathbf{x}_n(j) \in \{0, 1\}$)
$\mathbf{Y} = y_1, \dots, y_N$	labels (regression: $\mathbb{R}$; class. $\{0, 1\}$)
$\hat{y} = f_\theta(\mathbf{x})$	prediction by model $f_\theta(.)$
$\mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}})$	loss function

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}}) \equiv \underset{\theta}{\operatorname{argmin}}\, \frac{1}{N} \sum_{n=1}^{N} \ell\left(y_n, f_\theta(\mathbf{x}_n)\right)$$

Loss Functions

The loss function:

- Related with final metric (e.g., classification rate, number of clicks, revenue, ...)
- Nice mathematical properties (to find $\min_\theta(.)$).

Example (Regression loss function)

Squared error:

$$\ell(y, \hat{y}) = (y - \hat{y})^2$$

Example (Classification loss function)

Binary cross entropy:

$$\ell(y, \hat{y}) = -\big(y \cdot \ln \hat{y} + (1 - y) \cdot \ln(1 - \hat{y})\big)$$

Solving $\text{argmin}_\theta(.)$

First idea:

1. Take derivatives of the loss with respect each parameter and equalize with zero.
2. Solve set of $|\theta|$ equations with $|\theta|$ variables.

This works for *losses* (e.g. squared error), easy *activation function* (identity) and architectures (e.g. one perceptron).

In general, millions of non-linear equations.

Gradient descent

Finding the minimum in θ is a key aspect on deep learning.[1]

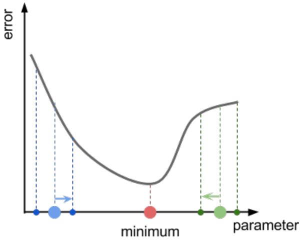
Most methods are based on the iterative-method *gradient descent*

1. Initialize: $\theta^{(0)}$, $i = 0$
2. Repeat till convergence from $i = 0, 1, \ldots$:
   - Compute gradient of the loss function with respect θ at $\theta^{(i)}$
   - $\theta^{(i+1)} = \theta^{(i)} - \alpha \nabla \mathcal{L}\left(Y, f_{\theta^{(i)}}(X)\right)$

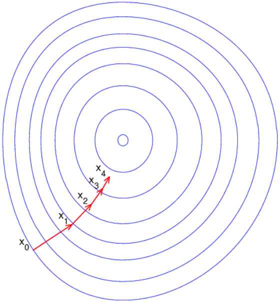
If the *learning rate* α is small enough,

$$\mathcal{L}\left(Y, f_{\theta^{(i+1)}}(X)\right) \leq \mathcal{L}\left(Y, f_{\theta^{(i)}}(X)\right)$$

[1] An introduction in this course, *D2L3 - Optimization*



$$\theta^{(i+1)} = \theta^{(i)} - \alpha \nabla \mathcal{L}\left(Y, f_{\theta^{(i)}}(X)\right)$$
$$\mathcal{L}\left(Y, f_{\theta^{(i+1)}}(X)\right) \leq \mathcal{L}\left(Y, f_{\theta^{(i)}}(X)\right)$$



From *Wikipedia*

$$\theta^{(i+1)} = \theta^{(i)} - \alpha \nabla \mathcal{L}\left(Y, f_{\theta^{(i)}}(X)\right)$$
$$\mathcal{L}\left(Y, f_{\theta^{(i+1)}}(X)\right) \leq \mathcal{L}\left(Y, f_{\theta^{(i)}}(X)\right)$$

Let's assume the model $\hat{\mathbf{y}}_n = f_\theta(\mathbf{x}_n)$ is just a perceptron:
$\hat{y}_n = f(z_n) = f(\mathbf{w} \cdot \mathbf{x}_n) = f\left(\sum_{j=1}^{J} w_j \cdot x_n(j)\right)$

Derivative with respect to a particular parameter: w_k:

$$
\begin{aligned}
\frac{\partial \mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}})}{\partial w_k} &= \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \ell\left(y_n, \hat{y}_n\right)}{\partial w_k} \\
&= \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \ell\left(y_n, \hat{y}_n\right)}{\partial \hat{y}_n} \cdot \frac{\partial \hat{y}_n}{\partial w_k} \\
&= \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \ell\left(y_n, \hat{y}_n\right)}{\partial \hat{y}_n} \cdot f'(z_n) \cdot \frac{\partial z_n}{\partial w_k} \\
&= \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \ell\left(y_n, \hat{y}_n\right)}{\partial \hat{y}_n} \cdot f'(z_n) \cdot x_k
\end{aligned}
$$

(See exercises with particular cases at the end of these slides)

Conclusions

- The perceptron is the basic unit in ANN
- It can be used both for linear regression and binary classification
- The weights can be estimated with iterative algorithms for any differentiable activation function and loss.
- Can be extended to neural networks (D2L1) and better optimization algorithms (D2L2)

Exercises

Exercise 1: regression

Lets assume:

Loss function, squared error	$\ell(y, \hat{y}) = (y - \hat{y})^2$
Activation function, identity	$f(z) = z$

Exercise

1. Show that the derivative of the $\mathcal{L}$ with respect the weight vector is:

$$\frac{\partial \mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}})}{\partial w_k} = \frac{2}{N} \sum_{n=1}^{N} (\hat{y} - y_n) \cdot x_n(k)$$

2. Using previous equation for $k = 1, \ldots, J$, and equalizing with 0, you can get the set of J linear equations and J variables[2]:

$$\mathbf{A} \cdot \mathbf{w} = \mathbf{b}$$

Find the values a_{ij} and b_j for $1 \leq i, j \leq J$.

[2] This part of the exercise only shows that in this particular case you can find a closed solution. It is not relevant to ANN

Exercise 2: classification – BCE

Lets assume:

Loss function, binary cross entropy	$\ell(y, \hat{y}) = -\big(y \cdot \ln \hat{y} + (1-y) \cdot \ln(1-\hat{y})\big)$
Activation function, sigmoid	$f(z) = \sigma(z)$

Exercise

1. Show that the derivative of the sigmoid

$$\sigma'(z) = \sigma(z) \cdot \big(1 - \sigma(z)\big)$$

2. Show that the derivative of the $\mathcal{L}$ with respect the weight vector is:

$$\frac{\partial \, \mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}})}{\partial w_k} = \frac{1}{N} \sum_{n=1}^{N} (\hat{y}_n - y_n) \cdot x_n(k)$$

Exercise 3: classification – MSE

Lets assume:

Loss function, squared error	$\ell(y, \hat{y}) = (y - \hat{y})^2$
Activation function, sigmoid	$f(z) = \sigma(z)$

Exercise

1. Show that the derivative of the $\mathcal{L}$ with respect the weight vector is:

$$\frac{\partial\, \mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}})}{\partial w_k} = \frac{2}{N} \sum_{n=1}^{N} (\hat{y}_n - y_n) \cdot \sigma'(z_n) \cdot x_n(k)$$

2. Comparing this gradient and the one using binary cross entropy, the magnitude of both increases with difference between labels and predictions, $\hat{y}_n - y_n$. Do you think this is a good or bad characteristic of the algorithm?
3. However, the MSE loss introduces the term $\sigma'(z_n)$. Looking to the terms with big errors (e.g.: $y_n = 1$ and $\hat{y}_n \approx 0$), do you think this term helps in the convergence?