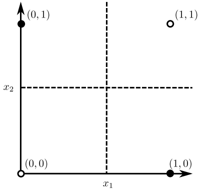
1. Feedforward neural networks

1.2. Structure and optimization criteria

Manel Martínez-Ramón

- The perceptron unit presented before is linear.
- nonlinear problems, as the classic XOR problem, cannot be solved with a perceptron.

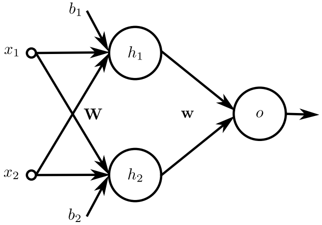


- The data labels are a XOR function of its coordinates: black dots are labelled with $+1$, white dots are labelled with 0.

- A linear function cannot classify the data. It is possible to construct a nonlinear function with several perceptrons in two layers.

A very simplistic neural network

The XOR problem can be soved with a very simple two layer structure.



A simple neural network to solve the XOR problem.

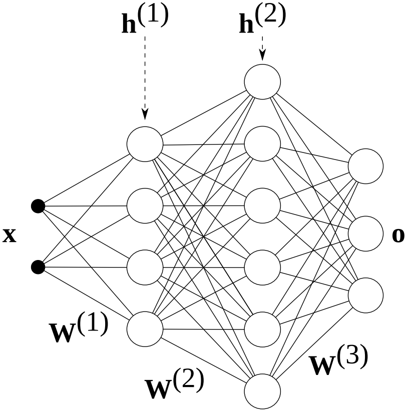
$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \mathbf{W}^\top \mathbf{x} + \mathbf{b} = \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{1}$$

$$\mathbf{h} = \boldsymbol{\sigma}(\mathbf{z}) = \begin{bmatrix} \sigma(z_1) \\ \sigma(z_2) \end{bmatrix} \tag{2}$$

$$o = \phi\left(\mathbf{w}^\top \mathbf{h}\right) \tag{3}$$

Problem: Show that the above structure can solve the XOR problem if $\mathbf{W} = \begin{bmatrix} a & a \\ a & a \end{bmatrix}$ and $\mathbf{b} = \left[-\frac{a}{2}, -\frac{3a}{2}\right]^\top$, and where $a > 0$ is an arbitrary constant where the nonlinear activations are sigmoid functions.

- $L+1$ layers, layer $j=0$ is the input $\mathbf{x}$
- $L-1$ *hidden* layers with D_j nodes with outputs $\mathbf{h}^{(j)}$.
- The last layer implements the output $\mathbf{o}$.
- Layers interconnected by linear weights $\mathbf{W}^{(j)}$.



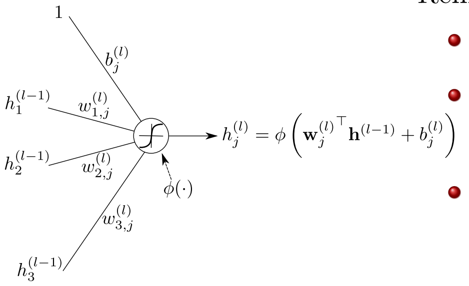
- The layers are interconnected by edges. Each edge contains a weight $w_{i,j}^{(l)}$ that connects the output of node i of layer $l-1$ to the input of node j in layer l. Also, each node has a bias input $b_j^{(l)}$.
- Each node contains an affine transformation of the output of the previous layer as

$$z_j^{(l)} = \sum_{i=1}^{D_{l-1}} w_{i,j}^{(l)} h_i^{(l-1)} + b_j^{(l)} = \mathbf{w}_j^{(l)\top} \mathbf{h}^{(l-1)} + b_j^{(l)} \tag{4}$$

- The output j of layer l, $h_j^{(l)}$ is

$$h_j^{(l)} = \phi\left(z_j^{(l)}\right) = \phi\left(\mathbf{w}_j^{(l)\top} \mathbf{h}^{(l-1)} + b_j^{(l)}\right) \tag{5}$$

A neuron



Remarks:

- $b_j^{(l)}$ implements the bias.
- In some notations, $b_j^{(l)} = w_{0,j}^{(l)}$ and $h_0^{(l-1)} = 1$.
- Element $w_{i,j}^{(l)}$ of vector $\mathbf{w}_j^{(l)}$ represents the connection of node i of layer $l-1$ with node j of layer l.

- The nonlinear activation for a hidden layer is not usually a sigmoidal, as we will see further.

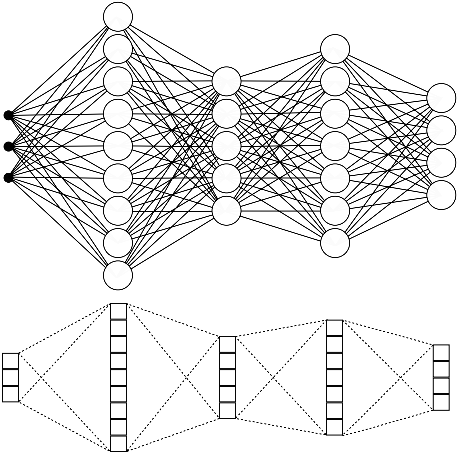
Notation conventions

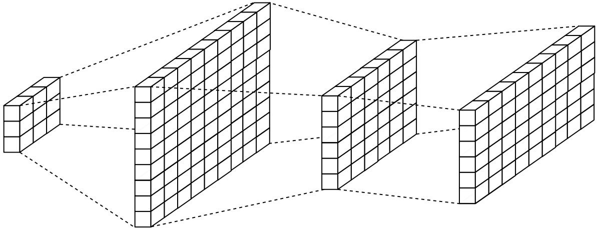
The following general notation conventions are taken in this course:

- Lowercase, normal letters as b, w represent scalars, i.e. $b \in \mathbb{R}$.
- Uppercase normal letters as D, N represent constants.
- Lowercase bold letters as $\mathbf{b}, \mathbf{w}$ are column vectors.
- Uppercase bold letters as $\mathbf{W}$ are arrays. In this module, they are all 2D matrices, i.e, for example, $\mathbf{W} \in \mathbb{R}^{D_1 \times D_2}$ is a matrix of D_1 rows and D_2 columns. In general, an array may have more than 2 dimensions.

The following conventions particular to the structure of neural networks are taken:

- Superindex (l) between parenthesis refer to output layers. $\mathbf{W}^{(l)}$ refers to a matrix of weights that connects layer $l-1$ to layer l.
- Subindex i in vector $\mathbf{w}_i^{(l)}$ mean that this vector is the i-th column of matrix $\mathbf{W}^{(l)}$. $\mathbf{w}_i^{(l)}$ is then a *column vector*.
- $\mathbf{b}^{(l)}$ is a *column vector* containing all the biases of layer l.
- Subindexes i, j in scalar $w_{i,j}^{(l)}$ mean, equivalently, that:
  - $w_{i,j}^{(l)}$ is the connection between node i of layer $l-1$ and node j of layer l.
  - $w_{i,j}^{(l)}$ is $i_t h$ element of vector $\mathbf{w}_j^{(l)}$
  - $w_{i,j}^{(l)}$ is element i, j of matrix $\mathbf{W}^{(l)}$.





- The first activation for hidden nodes to be used was the sigmoid

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

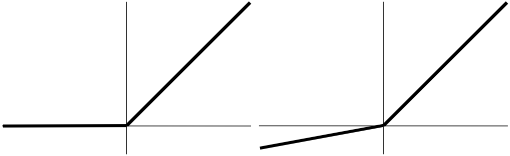
- Modern neural networks use the so-called rectified linear unit (ReLU)

$$\phi(z) = \max(0, z) \tag{6}$$

- ReLU is enough to provide the NN with nonlinear properties.
- A nonzero gradient extension of the ReLU is

$$\phi(z_i) = max\{0, z_i\} + \alpha_i min\{0, z_i\} \tag{7}$$

- Three versions of this activation are extremely useful.
  - *Absolute value* $\phi(z_i, \alpha_i) = max\{0, z_i\} - min\{0, z_i\} = |z_i|$
  - *Leaky ReLU*, for small values of α_i
  - *Parametric ReLU* or PReLU, when α_i is learnable using gradient descent.



Left: Rectified Linear Unit (ReLU). Right: Leaky ReLU.

- Maxout units divide **z** in groups of k elements, outputs the maximum

$$\phi(z)_i = \max_{j \in G^i}(z_j) \tag{8}$$

- Maxout units can generalize any of the above activations.

Training criterion: Maximum Likelihood

- Assume a dataset $\{\mathbf{x}_i, \mathbf{y}_i\}$, $1 \leq i \leq N$. If training outputs $\mathbf{y}_i$ are independent given their corresponding inputs $\mathbf{x}_i$, the joint likelihood is equal to the product of likelihoods, this is

$$p(\mathbf{Y}|\mathbf{X}) = \prod_i p(\mathbf{y}_i|\mathbf{x}_i) \tag{9}$$

- Applying logarithms and dividing by the number of training data N

$$J_{ML}(\boldsymbol{\theta}) = -\frac{1}{N}\log p(\mathbf{Y}|\mathbf{X}) = -\frac{1}{N}\sum_i \log p(\mathbf{y}_i|\mathbf{x}_i) \approx -\mathbb{E}_{\mathbf{x},\mathbf{y}} \log p(\mathbf{y}|\mathbf{x}) \tag{10}$$

where θ represents the set of parameters $w_{j,k}^{(l)}$, $b_k^{(l)}$ to optimize.

- This is the cross entropy between the real and predicted probabilities (as we will see later).

- In order to minimize overfitting, many approaches use a regularization factor in the cost function that minimizes the square norm of the parameters, such as

$$J(\boldsymbol{\theta}) = J_{ML}(\boldsymbol{\theta}) + \lambda \sum_l ||\mathbf{W}^{(l)}||_F^2 \tag{11}$$

where $||\cdot||_F^2$ is the squared Frobenius norm operator.

$$||\mathbf{W}^{(l)}||_F^2 = \sum_{j,k} w_{j,k}^2 \tag{12}$$

- Other forms of regularization as dropout, early stopping or data augmentation will also be discussed.

- Depending on the posterior distribution that we choose, the interpretation and the uses will be different. We will primarily use
  - Gaussian posterior for regression

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} e^{\left(-\frac{1}{2}(\mathbf{y}-\mathbf{z})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{y}-\mathbf{z})\right)} \tag{13}$$

  - Sigmoid activation for binary classification

$$p(y=1|\mathbf{x}) = \frac{1}{1+e^{-yz}} \tag{14}$$

  - Softmax activations for multiclass classification

$$p\left(y=k|\mathbf{x}\right) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{(z_j)}}$$

Gaussian posterior for regression

- Interpreted as a regression model with linear output.
- The estimation error components are considered independent

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{(2\pi\sigma^2)^{D/2}} \exp\left(-\frac{1}{2\sigma^2}||\mathbf{y}-\mathbf{z}||^2\right) \tag{15}$$

- The cost function in Eq. (10) for this model is

$$J_{ML}(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}} \log p(\mathbf{y}|\mathbf{x}) = \mathbb{E}_{\mathbf{x},\mathbf{y}}\left(\frac{1}{2\sigma^2}\|\mathbf{y}-\mathbf{z}\|^2 + \frac{D}{2}2\pi\sigma^2\right)$$

$$= \mathbb{E}_{\mathbf{x},\mathbf{y}}\left(\|\mathbf{y}-\mathbf{z}\|^2\right) + \text{constant} \approx \frac{1}{N}\sum_{i=1}^{N}\|\mathbf{y}_i - \mathbf{W}^{(L)\top}\mathbf{h}^{(L-1)}\|^2$$

The output $\mathbf{z}$ and the regressors $\mathbf{y}$ are assumed to be vectors (multitask regression).

Sigmoid posterior for binary classification

- We assume an unnormalized log-likelihood as

$$\log \tilde{p}(y|\mathbf{x}) = yz \longrightarrow \tilde{p}(y|\mathbf{x}) = e^{yz} \tag{16}$$

which must be normalized as

$$p(y|\mathbf{x}) = \frac{e^{yz}}{\sum_{y'=0}^{1} e^{yz}} = \frac{e^{yz}}{1+e^{z}} \tag{17}$$

- Therefore

$$\begin{aligned} p(y=0|\mathbf{x}) &= \frac{1}{1+e^{z}} \\ p(y=1|\mathbf{x}) &= \frac{e^{z}}{1+e^{z}} \end{aligned} \tag{18}$$

- Since the sigmoid function is $\sigma(a) = \frac{1}{1+e^{-a}}$, then

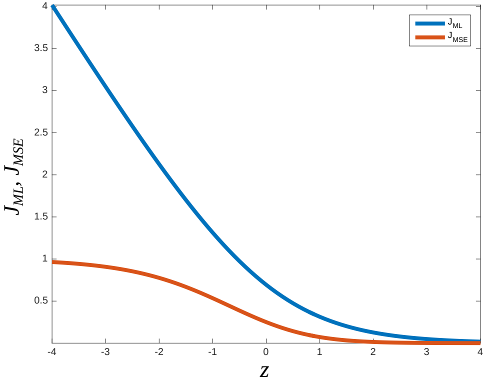
$$p(y|\mathbf{x}) = \frac{1}{1 + e^{-(2y-1)z}} = \begin{cases} \frac{1}{1+e^{z}}, & y = 0 \\ \frac{1}{1+e^{-z}} = \frac{e^{z}}{1+e^{z}}, & y = 1 \end{cases} \tag{19}$$

- Finally

$$p(y|\mathbf{x}) = \sigma\left((2y-1)\mathbf{w}^{(L)}{}^{\top}\mathbf{h}^{(L-1)}\right) \tag{20}$$

- According to Eq. (10), the cost function is

$$J_{ML}(\boldsymbol{\theta}) = \mathbb{E}\left[\log \sigma((2y-1)z)\right] = -\mathbb{E}\left[\log\left(1 + e^{(1-2y)z}\right)\right] \tag{21}$$



Cross entropy cost function (blue) and MSE cost function (red) for the Bernoulli log likelihood for a single sample with $y = 1$. The cross entropy cost function has a derivative that increases when the value of the function increases. The MSE cost function has a very small derivative when the function tends to its maximum.

- Here the neural network has a vectorized output

$$\mathbf{z} = \mathbf{W}^{(L)\top}\mathbf{h}^{(L-1)} \tag{22}$$

- Then, we model each element z_k in $\mathbf{z}$ as

$$\begin{aligned} z_k &= \log \tilde{p}(y = k|\mathbf{x}) \\ \tilde{p}(y = k|\mathbf{x}) &= e^{z_k} \end{aligned} \tag{23}$$

and then we normalize *softmax* function,

$$p(y = k|\mathbf{x}) = \mathrm{softmax}(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \tag{24}$$

- The log-likelihood function can be interpreted as a cross-entropy between the actual probability of the label and the estimated one:

$$H(q,p) = -\mathbb{E}\left[q \log p\right] \tag{25}$$

For the binary case, the equivalence is straightforward (exercise).

- For the multiclass case:
  1. We first change the multiclass label $y_i = k$, $1 \le k \le K$ by a vector $\mathbf{y}_i = \left[y_i^{(1)}, \cdots, y_i^{(K)}\right]$ where $y_i^{(k)} = 1$ and the rest are zero.
  2. We denote the real probability that $y_i = k$ as

$$q\left(y_i^{(k)} = 1\right) = q_i(k) = \begin{cases} 1, & y_i = k \\ 0, & y_i \neq k \end{cases} \tag{26}$$

- The cross-entropy loss for sample i can be written as

$$\ell_i = -\sum_{k=1}^{K} q_y(k) \log p(y_i = k|\mathbf{x}) \tag{27}$$

- Note that $q_i(k) = y_i^{(k)}$. By changing $p(y_i = k|\mathbf{x})$ by Eq. (24)

$$l_i = -\sum_{k=1}^{K} y_i^{(k)} \log \frac{e^{z_{i,k}}}{\sum_{j=1}^{K} e^{z_{i,j}}} = -\sum_{k=1}^{K} y_i^{(k)} z_{i,k} + \sum_{k=1}^{K} y_i^{(k)} \log \sum_{j=1}^{K} e^{z_{i,j}} \tag{28}$$

- Since only one of the elements of vector $\mathbf{y}_i$ is 1

$$l_i = -\sum_{k=1}^{K} y_i^{(k)} = -\sum_{k=1}^{K} y_i^{(k)} z_{i,k} + \log \sum_{j=1}^{K} e^{z_{i,j}} \tag{29}$$

where $z_{i,k}$ is the k-th output for input sample $\mathbf{x}_i$.