# Notations

Version : 1.00

This document may be updated along the class, pay attention to have the latest version available on Moodle.

The objective of this document is to summarise the mathematical notations and formalisms we use in this class.

We largely follow the notation adopted in the online course on Coursera on Deep Learning by Andrew Ng.

## 1 Training Set

| | |
|---|---|
| $m$ | number of samples in the input dataset |
| $n_x$ | number of input features, dimension of the input feature vector |
| $\mathbf{x}$ | input feature vector of dimension $n_x$ |
| $x_k$ | $k$-th component of the input feature vector ($1 \leq k \leq n_x$) |
| $\mathbf{x}^{(i)}$ | input feature vector of the $i$-th training sample ($1 \leq i \leq m$) |
| $x_k^{(i)}$ | $k$-th component of the input feature vector of the $i$-th training sample |
| $y$ | scalar output variable, also called target output or label |
| $\mathbf{y}$ | output vector (or target output vector) of dimension $n_y$ vector |
| $y_k$ | $k$-th component of the output vector (or target output vector) ($1 \leq k \leq n_y$) |
| $\widehat{y}$ | predicted output, as computed by the mapping function |
| $\widehat{\mathbf{y}}$ | predicted output vector, as computed by the mapping function |
| $(\mathbf{x}, \mathbf{y})$ | input sample (pair of input feature vector and corresponding label vector) |
| $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ | $i$-th input sample of the input dataset ($1 \leq i \leq m$) |

Note that vectors are denoted by **bold** symbols.

The **input dataset** can then be denoted as

$$D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}); i = 1, \ldots, m\} \tag{1}$$

The input dataset is split into a **training dataset** $D^{(train)}$ and a **test dataset** $D^{(test)}$ of size $m^{(train)}$ and $m^{(test)}$, respectively (with $m^{(train)} + m^{(test)} = m$).

The samples of the input features can be composed into a $n_x \times m$-matrix that contains the sample vectors in its columns [1] :

$$\mathbf{X} = \begin{pmatrix} x_1^{(1)} & & \dots & & x_1^{(m)} \\ & \ddots & & & \\ \vdots & & x_k^{(i)} & & \vdots \\ & & & \ddots & \\ x_n^{(1)} & & \dots & & x_n^{(m)} \end{pmatrix} \tag{2}$$

The matrix elements are then given by $X_{k,i} = x_k^{(i)}$.

Similarly, the labels for the different samples can be put into columns of a suitable maths object, i.e. a $n_y \times m$-matrix or a $m$-dimensional row vector in case of scalar a output (if $n_y = 1$) :

$$\mathbf{Y} = \left(y^{(1)}, \dots, y^{(m)}\right) \tag{3}$$

The full training set can then be denoted with $(\mathbf{X}, \mathbf{Y})$, stating that it is composed of the design matrix $\mathbf{X}$ and the corresponding output vector $\mathbf{Y}$.

Note that we use bold capital letters for object that contain several samples.

## 2 Mapping Function

The true but unknown mapping function is denoted by $f(x)$. It is approximated by a suitable parametrised function $h(x) = h_\theta(x)$ that is learned by using the training data. Sometimes, we indicate the parameters as subscript i.e. $h_\theta$ - sometimes we don't and write just $h$. The model function can be used to compute the predicted output from given input features, i.e.

$$\widehat{y} = h_\theta(\mathbf{x}) \tag{4}$$

Similarly, we also write

$$\widehat{Y} = h_\theta(\mathbf{X}) \tag{5}$$

Note that, here, the function $h_\theta$ operates column-wise.

In some books, the notation $\widehat{y} = \widehat{f}(\mathbf{x})$ with $\widehat{f}$ instead of $h_\theta$. This is equivalent - but we prefer to use the $h$ notation to emphasize the main principle of machine learning (against statistics)

1. While we will always stick to this convention in mathematical formulas, we will – for practical reasons – be less strict in iPython Notebooks. E.g. the sklearn function *train_test_split*() uses the samples vectors as rows.

which is to let the machine explore a very large set of *hypothesis* functions $h$ and determine automatically the one that best fits the problem.

In a classification task, the output $y$ is a set of discrete values corresponding to the class labels we want to recognise :

$$y \in \{C_1, \ldots, C_k, \ldots, C_K\} \tag{6}$$

where $K$ is the number of classes.

Often, we are interested in the probability of finding $C_k$ for $k = 1, \ldots, K$ (and not just the class with the highest predicted probability). In this case, we want the network to output a vector of values where the element in the $k$-th position gives the conditional probabilities of finding class $C_k$. The size of the vector corresponds to the number of classes and we associate the position within the vector $k$ with the class label $c$. For simplicity, we assume that the class labels are identical with the position, i.e. $c = k$

$$\hat{\mathbf{y}} = \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{pmatrix} \quad , \quad \hat{y}_k = h_{k,\theta}(\mathbf{x}) = p(k|\mathbf{x}, \theta) \tag{7}$$

## 3 Cost Function

The cost function is denoted by $J = J(\theta)$.

Sometimes, we indicate the type of cost function with a subscript. The two most common forms of cost function are the mean-square error cost function $J_{\text{MSE}}(\theta)$ and the cross-entropy cost function $J_{\text{CE}}(\theta)$ (see expressions below).

Typically, the cost function is an arithmetic average over contributions per sample - referred to as *loss*.

$$J(\theta) = \frac{1}{m} \sum_{n=1}^{m} L(\mathbf{x}^{(i)}, y^{(i)}, \theta) \tag{8}$$

For the two most common cost function we have (by using $\hat{y}^{(i)} = h_\theta(\mathbf{x}^{(i)})$) :

Mean-Square Loss :

$$L_{\text{MSE}}(\mathbf{x}^{(i)}, y^{(i)}; \theta) = \left(y^{(i)} - \hat{y}^{(i)}\right)^2 \tag{9}$$

Cross-Entropy Loss :

$$L_{\text{CE}}(\theta) = -y^{(i)} \log\left(\hat{y}^{(i)}\right) \tag{10}$$

A learning algorithm aims at choosing the $\theta$ that minimise $J(\theta)$.

# 4 Neural Networks

For neural networks the form of the function $h_\theta$ and its parameters have a specific structure. It depends on the network architecture and looks different e.g. for MLPs, CNNs or RNNs.

## 4.1 MLP

In the following, we define the notation used for a multi-layer perceptron (MLP) with $L$ layers. In this numbering, only the hidden layers and the output layer are counted as layers. By also counting the layer with the input features (with index 0) we would have $L+1$ layers.

The variables and parameters associated with layer $l$ ($1 \leq l \leq L$) are marked with super-scripts in square brackets.

| | |
|---|---|
| $n^{[l]}$ | number of neurons in layer $l$ |
| $a_k^{[l]}$ | activation of the $k$-th neuron in layer $l$ (with $1 \leq k \leq n^{[l]}$) |
| $\mathbf{a}^{[l]}$ | vector of activations in layer $l$ (activations vector of dimension $n^{[l]}$ |
| $z_k^{[l]}$ | logit of the $k$-th neuron in layer $l$ (with $1 \leq k \leq n^{[l]}$) |
| $\mathbf{z}^{[l]}$ | vector of logits in layer $l$ (logits vector of dimension $n^{[l]}$ |
| $a_k^{[l],(i)}$ | activation of the $k$-th neuron in layer $l$ for given input sample $(i)$ |
| $\mathbf{a}^{[l],(i)}$ | vector of activations in layer $l$ for given input sample $(i)$ |
| $\mathbf{A}^{[l]}$ | matrix of activations in layer $l$ (one sample per column) |
| $\mathbf{Z}^{[l]}$ | matrix of logits in layer $l$ (one sample per column) |
| $w_{k,j}^{[l]}$ | weight between neuron in layer $l-1$ and index $j$ and neuron in layer $l$ and index $k$. |
| $\mathbf{W}^{[l]}$ | weights matrix of dimensions $(n^{[l]}, n^{[l-1]})$ between layer $l-1$ and layer $l$. |
| $b_k^{[l]}$ | bias of neuron $k$ in layer $l$. |
| $\mathbf{b}^{[l]}$ | bias vector of layer $l$ (length $n^{[l]}$). |

Consistent with the above notation, we set $\mathbf{A}^{[0]} = \mathbf{X}$.

The functional relationship between logits and activations is given by the activation function. This can be dependent on the layer and we will denote it by $g^{[l]}$. Accordingly we write :

$$\mathbf{A}^{[l]} = g^{[l]}\left(\mathbf{Z}^{[l]}\right), \text{etc.}$$

Note that the activation function is applied element-wise.

## 5 Derivatives, Gradients

For applying opimization algorithms, gradients of the cost function with respect to the parameters $\theta$ need to be computed. Here, the symbol $\theta$ stands for all the elements contained in the different weights vectors $\mathbf{W}^{[l]}$ and bias vectors $\mathbf{b}^{[l]}$ $(1 \leq l \leq L)$. Hence, derivatives need to be taken with respect to all these parameters.

In the following, $J$ denotes any function and $\theta_k$ any parameter (or $\theta$ any vector of such.

The derivative of $J$ w.r.t. single parameter $\theta_k$ is denoted by :

$$\frac{\partial J}{\partial \theta_k} \tag{11}$$

As mentioned above, $\theta_k$ may stand for an arbitrary (trainable) parameter such as $w_{k,j}^{[l]}$.

The derivatives w.r.t. to multiple parameters such as $\mathbf{W}^{[l]}$ leads to the gradients. The gradient of $J$ w.r.t. to the parameter vector $\theta$ is given by

$$\nabla_\theta J = \frac{\partial J}{\partial \theta} = \begin{pmatrix} \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{pmatrix} = d\theta \tag{12}$$

So, you should now be able to interpret expressions such as

$$\nabla_{\mathbf{W}^{[l]}} J = \frac{\partial J}{\partial \mathbf{W}^{[l]}} = d\mathbf{W}^{[l]} \tag{13}$$

The last notation $d\theta$ for the gradient is extensively used in Ng' coursera lectures. What is missing here is the reference to the cost function. So, we rather use $\nabla_\theta J$ if not otherwise stated.

Note that the gradient of a function w.r.t. to a vector of parameters leads to a vector of the same dimensions as the parameter vector :

$$\theta = \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_n \end{pmatrix} \rightarrow \nabla_\theta J = \begin{pmatrix} \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{pmatrix} \tag{14}$$

Taking the derivatives of vector-valued functions $\mathbf{g}$ leads to *Jacobian* matrices :

$$\mathbf{g}(\theta) = \begin{pmatrix} g_1(\theta) \\ \vdots \\ g_{n_g}(\theta) \end{pmatrix} \rightarrow \nabla_\theta \mathbf{g} = \begin{pmatrix} \frac{\partial g_1}{\partial \theta_1} & \cdots & \frac{\partial g_1}{\partial \theta_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial g_{n_g}}{\partial \theta_1} & \cdots & \frac{\partial g_{n_g}}{\partial \theta_n} \end{pmatrix} \tag{15}$$

For gradient descent, the parameter vector is updated with the help of the gradient of the cost function. This update rule is expressed as :

$$\theta_k \leftarrow \theta_k - \alpha \frac{\partial J(\theta)}{\partial \theta_k} \tag{16}$$

or in vector notation :

$$\theta \leftarrow \theta - \alpha \nabla_\theta J \tag{17}$$

where we have used the learning rate $\alpha$.