MULTI-LAYER NETWORKS
(FOR VECTORS)

by

FRANCESCO A. N. PALMIERI

UNIVERSITA' DEGLI STUDI
"LUIGI VANVITELLI"

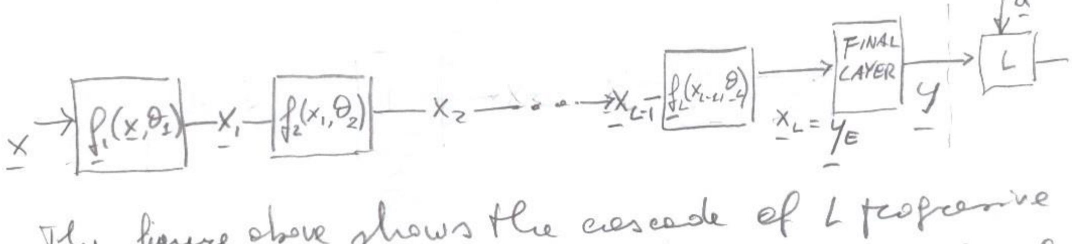
Corso: Signal Processing and Data Fusion
2024

Artificial
Neural networks have been originally inspired by biological networks where one of the main features is massive connectivity. Many paradigms that try to explain the signal processing capabilities of biological brains are based on progressive unfolding of the information coming from the various sensor modalities. Think of visual processing where, after a first transformation provided by the retina circuits, the visual cortex extracts relevant features such as lines, bars and various patterns. This seems to be a strategy that we find both in the relatively simple brains of insects and in much more complex mammal neural architectures.

Therefore, even if we have shown in a previous chapter that two-layer network can already provide universal functional capabilities, there must be intrinsic advantages in using multiple layers.
The reason for the success of multiple-layer architectures is not yet fully understood. Constrained connectivity may be one of reasons why we need multiple layers because of the need to provide sufficient mixing of the input components. Localized feature extraction may be another one.

Much experience has been gained in the last fourty years in researching the appropriate architecture for each problem, and almost invariably, multi-layer neural networks have provided effective solutions. We should not forget that a working system has to provide sufficient generalisation and massive contained connection may provide after learning the necessary smoothness.

In this chapter we consider general multi-layer architectures and provide explicit gradient computation for learning.



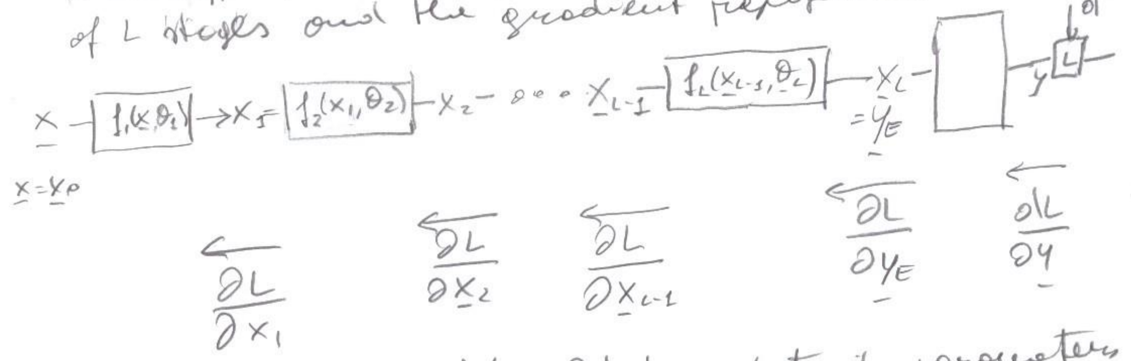
The figure above shows the cascade of L progressive layers that "prepare" the embedding $\underline{y}_E$ for the final layer. Recall from the previous chapter, that the final layer can be
be of three different types: (1) linear for regression (2) linear + sigmoids for multiple binary classification; (3) linear + softmax for M-ary classification.
In all cases the multiple layers have to transform the input vector $\underline{x}$ into the embedding vector $\underline{y}_E$ in a way which is appropriate for the problem to be solved at the output $\underline{y}$. Formally the transformation to the embedding space $\mathcal{Y}_E$ is the nested function

$$\underline{y}_E = \underline{f}_L(\underline{f}_{L-1}(\underline{f}_{L-2}(\ldots \underline{f}_2(\underline{f}_1(\underline{x},\underline{\theta}_1),\underline{\theta}_2),\ldots \theta_{L-2}),\underline{\theta}_{L-1}),\underline{\theta}_L)$$

The input $\underline{x} = \underline{x}_0$ has size N, $\underline{x}_1 \to N_1$, $x_2 \to N_2$ … $X_L \to N_L = N_E$

The layer functions can be of different types and each f_i may depend on a set of parameters θ_i to be learned. More generally some of the layer functions may be fixed with no free parameters as we will see in the following examples.

In all cases the parameters, to be learned, follow a gradient descent algorithm. The gradient is computed on the output loss function $L(y, \underline{d})$ and be "back propagated" following the rule of total differentials. The figure shows the cascade of L stages and the gradient propagation.



$\underline{x}$
$f_1(\underline{x},\theta_1)$
x_1
$f_2(x_1,\theta_2)$
x_2
$\underline{x}_{L-1}$
$f_L(\underline{x}_{L-1},\theta_L)$
$x_L = \underline{y}_E$
y
L
d
$\underline{x} = \underline{x}_0$
$\overleftarrow{\frac{\partial L}{\partial x_1}}$
$\overleftarrow{\frac{\partial L}{\partial \underline{x}_2}}$
$\overleftarrow{\frac{\partial L}{\partial \underline{x}_{L-1}}}$
$\overleftarrow{\frac{\partial L}{\partial \underline{y}_E}}$
$\overleftarrow{\frac{\partial L}{\partial \underline{y}}}$

A <u>parametric block</u> $f_i(x_{i-1},\underline{\theta}_i)$ to update its parameters $\underline{\theta}_i$ needs the gradient $\frac{\partial L}{\partial \theta_i}$, that can be computed from the back propagated gradient $\frac{\partial L}{\partial \underline{x}_i}$ at its output $\underline{x}_i$ as

$$\frac{\partial L}{\partial \underline{\theta}_i} = \left(\frac{\partial \underline{x}_i}{\partial \underline{\theta}_i}\right)^T \frac{\partial L}{\partial \underline{x}_i} \qquad i = 1, \dots, L$$

Backpropagation through layers consists in computing the Jacobian $\left(\frac{\partial \underline{x}_{i+1}}{\partial x_i}\right)$ at each stage and propagate backword as follows:

$$\frac{\partial L}{\partial \underline{x}_i} = \left(\frac{\partial \underline{x}_{i+1}}{\partial \underline{x}_i}\right)^T \frac{\partial L}{\partial \underline{x}_{i+1}} \qquad i = L-1, L-2, \dots, 1$$

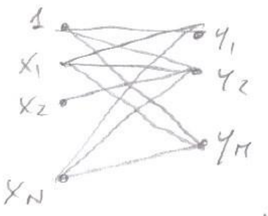
Each layer may be of different type and also be nonparametric, i.e. with no parameters to be learned. In such a case we just backpropagate the gradient through it
In the following we examine different kinds of blocks specifing the Jacobian for backpropagation and also the gradient for parametric learning

LINEAR LAYER

We have already discussed linear transformations both in the Chapter on linear regressors and classifiers and in the two-layer architectures. Linear layers (fully connected) can be inserted at any stage in the multi-layer architecture. They are usually followed by non linearities, but we discuss them here separately as a potential building block of our cascade.
Recall that for a linear transformation we mean always an affine transformation, because we include biases to avoid to have to worry about means and baselines.

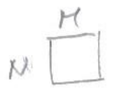
GENERIC LINEAR BLOCK

To facilitate notations consider first a generic linear block $\underline{x} \to \underline{y}$



$$\begin{tikzcd} \underline{x} \arrow[r, dash] & \boxed{(\underline{\underline{W}}, \underline{b})} \arrow[r] & \underline{y} \\ {} & & {} \arrow[ll, "\frac{\partial L}{\partial \underline{x}}" below, phantom] \end{tikzcd}$$

$\xleftarrow{\frac{\partial L}{\partial \underline{x}}}$ $\qquad$ $\xleftarrow{\frac{\partial L}{\partial \underline{y}}}$



$\underline{\underline{W}} = [\underline{w}_1\ \underline{w}_2 \ldots \underline{w}_M]$

$\underline{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{bmatrix}$

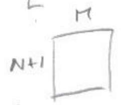
Forward propagation can be written as

$$\underline{y} = \underline{\underline{W}}^T \underline{x} + \underline{b} = \begin{bmatrix} \underline{w}_1^T \\ \underline{w}_2^T \\ \vdots \\ \underline{w}_M^T \end{bmatrix} \underline{x} + \underline{b}$$

or in the augmented form

$$\underline{y} = \underbrace{\left[\underline{b} \ \begin{matrix} \underline{w}_1^T \\ \underline{w}_2^T \\ \vdots \\ \underline{w}_M^T \end{matrix} \right]}_{\underline{\underline{W}}_a^T} \underbrace{\begin{bmatrix} 1 \\ \underline{x} \end{bmatrix}}_{\underline{x}_a}$$

$$\underline{\underline{W}}_a = \begin{bmatrix} \underline{b}^T \\ \underline{w}_1\ \underline{w}_2 \ldots \underline{w}_M \end{bmatrix}$$



The Jacobian is defined and computed as

$$\frac{\partial \underline{y}}{\partial \underline{x}} \triangleq \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_N} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_N} \\ \frac{\partial y_M}{\partial x_1} & \frac{\partial y_M}{\partial x_2} & & \frac{\partial y_M}{\partial x_N} \end{pmatrix} = \begin{bmatrix} \underline{w}_1^T \\ \underline{w}_2^T \\ \\ \underline{w}_M^T \end{bmatrix} = \underline{\underline{W}}^T$$

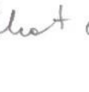
Note that it does not depend on the biases.

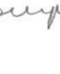
Gradient propagation <u>through</u> the block follows the rule

$$\frac{\partial L}{\partial \underline{x}} = \left(\frac{\partial \underline{y}}{\partial \underline{x}} \right)^T \frac{\partial L}{\partial \underline{y}} = \underline{\underline{W}} \frac{\partial L}{\partial \underline{y}}$$

To train $\underline{W}$ and $\underline{b}$ we need the gradients

$$\frac{\partial L}{\partial \underline{\underline{W}}} \quad \text{and} \quad \frac{\partial L}{\partial \underline{b}}$$





that can be computed from $\frac{\partial L}{\partial \underline{y}}$ as

$$\frac{\partial L}{\partial \underline{\underline{W}}} = \left[\frac{\partial L}{\partial \underline{w}_1}\frac{\partial L}{\partial \underline{w}_2}\cdots\frac{\partial L}{\partial \underline{w}_M}\right] = \left[\frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial \underline{w}_1}, \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial \underline{w}_2}, \dots, \frac{\partial L}{\partial y_M}\frac{\partial y_M}{\partial \underline{w}_M}\right]$$

$$= \left[\frac{\partial L}{\partial y_1}\underline{x}, \frac{\partial L}{\partial y_2}\underline{x}, \dots, \frac{\partial L}{\partial y_M}\underline{x}\right] = \left(\frac{\partial L}{\partial \underline{y}}\right)^T \otimes \underline{x}$$

$$\frac{\partial L}{\partial \underline{b}} = \begin{bmatrix}\frac{\partial L}{\partial b_1}\\ \frac{\partial L}{\partial b_2}\\ \vdots \\ \frac{\partial L}{\partial b_M}\end{bmatrix} = \begin{bmatrix}\frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial b_1}\\ \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial b_2}\\ \vdots \\ \frac{\partial L}{\partial y_M}\frac{\partial y_M}{\partial b_M}\end{bmatrix} = \begin{bmatrix}\frac{\partial L}{\partial y_1}\cdot 1\\ \frac{\partial L}{\partial y_2}\cdot 1\\ \vdots \\ \frac{\partial L}{\partial y_M}\cdot 1\end{bmatrix} = \frac{\partial L}{\partial \underline{y}}$$

In the augmented representation

$$\underbrace{\frac{\partial L}{\partial \underline{\underline{W}}_a}}_{M} = \underbrace{\left(\frac{\partial L}{\partial \underline{y}}\right)^T}_{M} \otimes \underline{x}_a$$

$$\left[\,\|^{N+1}\ \|^{N+1} \cdots \|^{N+1}\,\right] \qquad \|^{N+1}$$

The above results are immediately applied to the ith linear stage

$$\begin{tikzcd} \underline{x}_{i-1} \arrow[r] & \boxed{\underline{\underline{W}}_i, \underline{b}_i} \arrow[r] \arrow[l, bend left, "\frac{\partial L}{\partial \underline{x}_{i-1}}"] & \underline{x}_i \arrow[l, bend left, "\frac{\partial L}{\partial \underline{x}_i}"] \end{tikzcd}$$

The forward propagation is
$\underline{x}_i = f_i(\underline{x}_{i-1}, \underline{\theta}_i) = \underline{W}_i^T \underline{x}_{i-1} + \underline{b}_i$ or in the augmented representation

$\underline{x}_i = \underline{\underline{W}}_{ia}^T \underline{x}_{i-1a}$

the Jacobian is $\frac{\partial \underline{x}_i}{\partial \underline{x}_{i-1}} = \underline{\underline{W}}_i^T$ and the gradient backpropagation is

$$\boxed{\frac{\partial L}{\partial \underline{x}_{i-1}} = \underline{\underline{W}}_i \frac{\partial L}{\partial \underline{x}_i}}$$

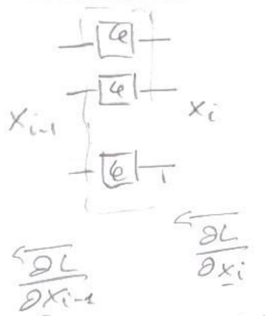
The gradients for $\underline{W}_i$ and $\underline{b}_i$ for learning are

$$\boxed{\frac{\partial L}{\partial \underline{W}_i} = \left(\frac{\partial L}{\partial \underline{x}_i}\right)^T \otimes \underline{x}_{i-1}}, \quad \boxed{\frac{\partial L}{\partial \underline{b}} = \frac{\partial L}{\partial \underline{x}_i}}$$

In the augmented format

$$\boxed{\frac{\partial L}{\partial \underline{\underline{W}}_{ia}} = \left(\frac{\partial L}{\partial \underline{x}_i}\right)^T \otimes \underline{x}_{i-1a}}$$

SET OF NON LINEARITIES



φ
φ
φ
x_{i-1}
x_i
$\frac{\partial L}{\partial x_i}$
$\frac{\partial L}{\partial x_{i-1}}$

$$\underline{x}_i = f_i(x_{i-1}) = \varphi(\underline{x}_{i-1})$$

$$N_i = N_{i-1}$$

There are no parameters to be learned, therefore we have to provide only the propagation rule for gradients.
the Jacobian is diagonal

$$\frac{\partial \underline{x}_i}{\partial \underline{x}_{i-1}} = \mathrm{diag}\left(\varphi'(\underline{x}_{i-1})\right)$$

and the gradient $\frac{\partial L}{\partial \underline{x}_i}$ is backpropagated as

$$\frac{\partial L}{\partial \underline{x}_{i-1}} = \left(\frac{\partial \underline{x}_i}{\partial \underline{x}_{i-1}}\right)^T \frac{\partial L}{\partial \underline{x}_i} = \mathrm{diag}\left(\varphi'(\underline{x}_{i-1})\right)\frac{\partial L}{\partial \underline{x}_i}$$

$$\Big\|_{N_{i-1}=N_i} \quad \overset{N_i}{\left(\diagdown\right)}^{N_i} \quad \Big\|_{N_i}$$

The nonlinearities could be sigmoids, RELUs or many others listed in Appendix x with their derivatives. To be specific consider as an example, logistics and RELUs.

For Logistics $\varphi(\xi) = \frac{1}{1+e^{-\xi}}$, $\varphi'(\xi) = \varphi(\xi)(1-\varphi(\xi))$

the vector of derivatives can be written as

$$\varphi'(\underline{\xi}) = \varphi(\underline{\xi}) \odot (\underline{1} - \varphi(\underline{\xi})) \qquad \underline{1} = \begin{pmatrix}1\\1\\ \vdots\\1\end{pmatrix}$$

therefore

$$\frac{\partial L}{\partial \underline{x}_{i-1}} = \mathrm{diag}\left(\underline{x}_i \odot (\underline{1} - \underline{x}_i)\right)\frac{\partial L}{\partial \underline{x}_i}$$

For RELUs $\varphi(\xi) = r(\xi)$, $\varphi'(\xi) = u(\xi)$

the vector of derivatives can be written as

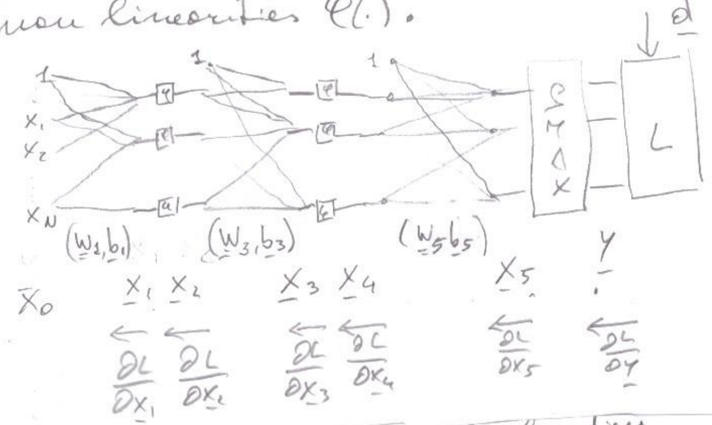
$$\varphi'(\underline{\xi}) = u(\underline{\xi})$$

therefore

$$\frac{\partial L}{\partial \underline{x}_{i-1}} = \mathrm{diag}\left(u(\underline{x}_{i-1})\right)\frac{\partial L}{\partial \underline{x}_i}$$

the RELUs in the gradient backward flow act as gates because they do not propagate the output gradient if the input is negative.

EXAMPLE Consider the following multi-layer MLP architecture for classification. Assume logistic non linearities $\varphi(.)$.


1
x_1
x_2
x_N
φ
S M A X
L
$\underline{d}$
$(\underline{\underline{W}}_1, \underline{b}_1)$
$(\underline{\underline{W}}_3, \underline{b}_3)$
$(\underline{\underline{W}}_5 \underline{b}_5)$
$\underline{y}$
$\underline{x}_0$
$\underline{x}_1$ $\underline{x}_2$
$\underline{x}_3$ $\underline{x}_4$
$\underline{x}_5$
$\frac{\partial L}{\partial \underline{x}_1}$ $\frac{\partial L}{\partial \underline{x}_2}$
$\frac{\partial L}{\partial \underline{x}_3}$ $\frac{\partial L}{\partial \underline{x}_4}$
$\frac{\partial L}{\partial \underline{x}_5}$
$\frac{\partial L}{\partial \underline{y}}$

The network implements the function $\underline{y} = f(\underline{x}, \underline{\theta})$ where $0 \le y_i \le 1$ $i = 1, .., M$ represent the estimated posterior associated to the sample $\underline{x}$.
The parameters to be learned are $\{(\underline{\underline{W}}_1, \underline{b}_1), (\underline{\underline{W}}_3, \underline{b}_3), (\underline{\underline{W}}_5 \underline{b}_5)\}$
the forward flow is

$$\underline{x}_0 = \underline{x} \rightarrow \underline{x}_1 = \underline{\underline{W}}_1^T \underline{x}_0 + \underline{b}_1 \rightarrow \underline{x}_2 = \varphi(\underline{x}_1) \rightarrow \underline{x}_3 = \underline{\underline{W}}_3^T \underline{x}_2 + \underline{b}_2 \rightarrow \underline{x}_4 = \varphi(\underline{x}_3) \rightarrow \underline{x}_5 = \underline{W}_5^T \underline{x}_4 + \underline{b}_5$$

$$\rightarrow \underline{y} = S_{max}(\underline{x}_5)$$

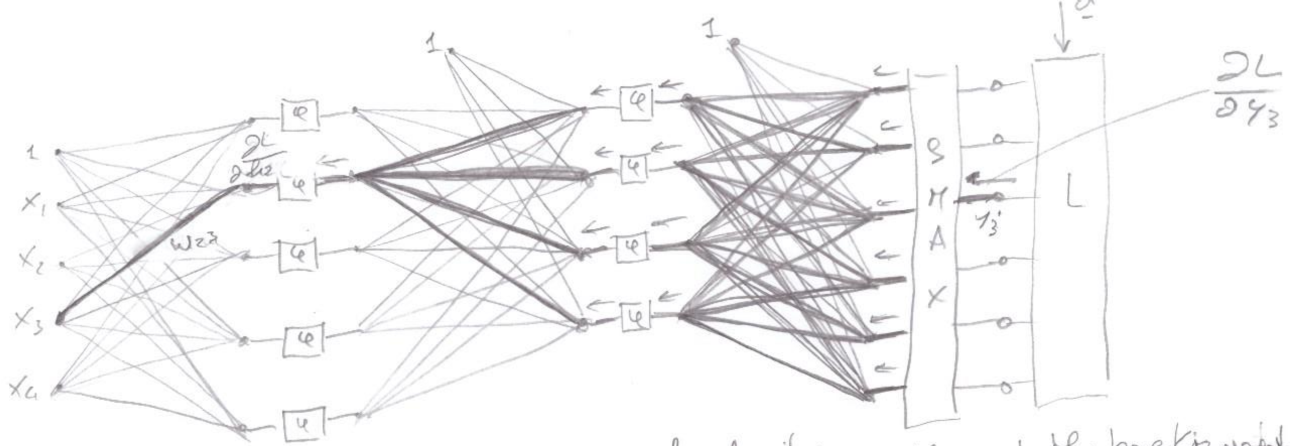
Using the cross-entropy as the loss function, the gradients vectors backpropagated are.

$$\frac{\partial L}{\partial \underline{y}} = -\frac{\underline{d}}{\underline{y}} \rightarrow \frac{\partial L}{\partial \underline{x}_5} = \left(\mathrm{diag}(\underline{y}) - \underline{y}\underline{y}^T\right) \frac{\partial L}{\partial \underline{y}} \rightarrow \frac{\partial L}{\partial \underline{x}_4} = \underline{\underline{W}}_5 \frac{\partial L}{\partial \underline{x}_5} \rightarrow$$

$$\frac{\partial L}{\partial \underline{x}_3} = \mathrm{diag}\left(\underline{x}_4 \odot (\underline{1} - \underline{x}_4)\right) \frac{\partial L}{\partial \underline{x}_4} \rightarrow \frac{\partial L}{\partial \underline{x}_2} = \underline{\underline{W}}_3 \frac{\partial L}{\partial \underline{x}_3} \rightarrow \frac{\partial L}{\partial \underline{x}_1} = \mathrm{diag}\left(\underline{x}_3 \odot (\underline{1} - \underline{x}_3)\right) \frac{\partial L}{\partial \underline{x}_2}$$

The updates on $(W_1, b_1), (\underline{W}_3, \underline{b}_3) (W_5, b_5)$ are based on the gradients

$$\frac{\partial L}{\partial \underline{\underline{W}}_1} = \left(\frac{\partial L}{\partial \underline{x}_1}\right)^T \otimes \underline{x}, \quad \frac{\partial L}{\partial \underline{b}_1} = \frac{\partial L}{\partial \underline{x}_1}, \quad \frac{\partial L}{\partial \underline{\underline{W}}_3} = \left(\frac{\partial L}{\partial \underline{x}_3}\right)^T \otimes \underline{x}_2 \, ; \quad \frac{\partial L}{\partial \underline{b}_3} = \frac{\partial L}{\partial \underline{x}_3} \, ; \quad \frac{\partial L}{\partial \underline{\underline{W}}_5} = \left(\frac{\partial L}{\partial \underline{x}_5}\right)^T \otimes \underline{x}_4$$


1
x_1
x_2
x_3
x_4
w_{23}
$\frac{\partial L}{\partial h_2}$
φ
1
1
SMAX
y_3
L
$\underline{d}$
$\frac{\partial L}{\partial y_3}$

The figure shows an example in which it is evidenced the backpropagation flow for a specific weight w_{23} in the first layer and a specific gradient $\frac{\partial L}{\partial y_i}$ in the output layer. The gradient is sent backward across the network weights and nonlinearities to w_{23}.

The effect of $\frac{\partial L}{\partial y_3}$ is translated in an update on w_{23} proportional to

$\frac{\partial L}{\partial h_2} \cdot x_3$.

SUMMARY OF THE BACKPROPAGATION ALGORITHM

The following steps are the standard gradient updates for the backpropagation algorithm. The procedure is presented in the "batch" version. "Minibatch" and "stochastic" versions are easy modifications.

Our training set $\mathcal{T} = \{(\underline{x}[n], \underline{d}[n]), n = 1,\dots, N_T\}$ will be used many times for learning the system parameters. Each time the training set is used, we say that we perform one "EPOQUE". Typically the parameters obtained at the end of each epoque are used as initial condition for the following one.

INITIALIZATION

At the beginning of each training session, the network parameters $\underline{\theta}$ are usually initialised to random values $\underline{\theta}_0$. Means and variances are usually determined heuristically and may depend on the problem at hand. Loss and gradients are set to zeros

$$L_0 = 0 \qquad \underline{g}_{\underline{\theta}0} = \underline{0}$$

EPOQUE K

Keep from the previous epoque only the network parameters $\underline{\theta}_{K-1}$; Set epoque loss and gradients to zero: $L[0] = 0$, $\underline{g}_{\underline{\theta}}[0] = \underline{0}$.

EXAMPLE $n = 1 : N_T$

- Get example $(\underline{x}[n], \underline{d}[n])$ from the training set;
- Propagate forward $\underline{x}[n]$ computing all the activations in the network.
- Compute the instantaneous loss of the output $\mathcal{L}[n] = L(\underline{y}[n], \underline{d}[n])$
- Compute all instantaneous gradients $\frac{\partial \mathcal{L}[n]}{\partial \underline{\theta}[n]}$ for every parameter backpropagating $\frac{\partial \mathcal{L}[n]}{\partial \underline{y}[n]}$.
- Update the loss with a running mean

$$L[n] = \frac{n-1}{n} L[n-1] + \frac{1}{n} \mathcal{L}[n]$$

- update the gradient for each parameter with a running mean

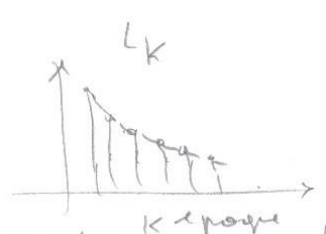
$$g_{\underline{\theta}}[n] = \frac{n-1}{n}\, g_{\underline{\theta}}[n-1] + \frac{1}{n}\frac{\partial L[n]}{\partial \underline{\theta}[n]}$$

<u>END OF EPOQUE K.</u>

If the loss $L_K = L[N_T]$ has reached a satisfactory value $\rightarrow$ EXIT
Otherwise update the parameters

$$\underline{\theta}_K = \underline{\theta}_{K-1} - \mu\, g_{\underline{\theta}}[N_T]$$

and start a new epoque.
Note that the total loss after each epoque could be computed separately using the parameters obtained at the end of that epoque. We presented the algorithm in a combined way <u>with the gradients</u> to present the training set to the network only once for each epoque. The only disadvantage is a useless computation of the gradients in the last epoque.
To keep track of convergence we look at the sequence $\{L_K\}$



Also global statistics about coefficients and performance metrics may be tracked during training.

We have discussed the method with reference to a <u>batch</u> approach. The strategy is easily modified for <u>mini batches</u> simply considering as epoques (subepoques) subsets of the training set. Recall at the end of each subset, we keep only the network parameters for the new batch. The <u>stochastic</u> version simply updates the parameters after each example.

CONSIDERATIONS ABOUT THE BACKPROPAGATION ALGORITHM

The backpropagation algorithm being a gradient search on a nonconvex cost function, will converge inevitably to a local minimum. To reach a global minimum is generally a rather hopeless task because in theory global convergence can be obtained starting from sufficiently many initial conditions. Therefore we have to rely on solutions that represent satisfactory local minimum.

Obviously the solutions obtained must be verified on the validation set for generalization.

In practice a neural network, to be useful in an application, has to go through various cycles of learning perhaps adjusting parameters to obtain a good compromise between ~~performances~~ on the training set and generalization.

Generalization is a crucial feature, as pointed out many times before, and can be thought as a sort of smoothness: if the training set samples are exactly matched the obtained function may be overfitting the data because it interpolates the training points too finely. A smoother function may do a better job in containing the variations outside the training set.

Therefore the problem of obtaining a smooth solution may be posed as a regularization problem

$$\underline{\theta}^{o} = \arg\min_{\underline{\theta}} \left(\sum_{n=1}^{n_2} L(\underline{y}[n], \underline{d}[n]) + \lambda R(\underline{\theta}) \right)$$

where $R(\underline{\theta})$ is a function of the parameters and λ is the Lagrange multiplier that regulates the effect of the additional term. For example $R(\theta)$ may be $R(\underline{\theta}) = \sum_{i=1}^{N_\theta} \theta_i^2$ (Tikhonov regularization).

The contraint usually lead to a decaying term that encourages smaller weights. Smaller weights that contribute the input of a sigmoidal function should give a smoother function. Regulating λ to larger values favors smoothness at the expences of accuracy in the cost function. Vice-versa small λ affect less the cost function encouraging smoothness.

There are other regularization strategies, also on the gradient, to encourage smoothness.

The Most popular ~~stoategies~~ strategies to encourage generalizations are based on simple heuristics:

Heuristic 1 : EARLY STOPPING

This strategy suggests to stop training before reaching convergence. This provides solution that are less accurate, but that perhaps are smoother

Heuristic 2 : ENSEMBLING

We train different networks, using different initial condition, and combine the prediction results. In regression we could use the mean or the median. Max rule could be used for classification (the classification that gets the largest score). This is sometimes called the committee machine.

Another strategy could be to train the system on different subsets of the training set obtaining different networks. The predictions from the networks are then combined (averaged or maxed out)

This strategy in statistics is known as BOOTSTRAP AGGREGATION or BAGGING.
The solutions are expected to be smoother.

Heuristic 3 : DROPOUT

In this strategy during training some hidden units are randomly set to zero at each gradient backpropagation. The number of censored units is usually 50% or less.

In this way the gradient estimates become noisy and may make the system more robust moving the evolution during training in a more scattered pattern that may lead to better minima and better generalization.
Dropout has allowed a major leap in performance for convolutional networks in solving image recognition tasks.

Heuristic 4 : ADD NOISE

Dropout discussed above idea be seen as applying Bernoulli noise to the hidden units during training. The idea can be extended by applying noise during gradient calculations. In regression it can be shown to be equivalent to regularization.
Noise can be also added to the weights

CONVERGENCE SPEED

The gradient updates are controlled by the stepsize parameter $\mu[k]$ that can be made adaptive.

Clearly larger μ correspond to faster updates that may overshoot the valleys of the cost function. Vice versa small μ may be slow and more precise in catching the minima, but they may be bad (high cost) local minima.

Generally at the beginning of training $\mu[k]$ is kept large with progressive decay around convergence. A good compromise that tracks the evolution of the gradient is ADAM widely used and already recorded in some of our previous chapters. Also ADAM has some free parameters that are determined by trial-and-error.