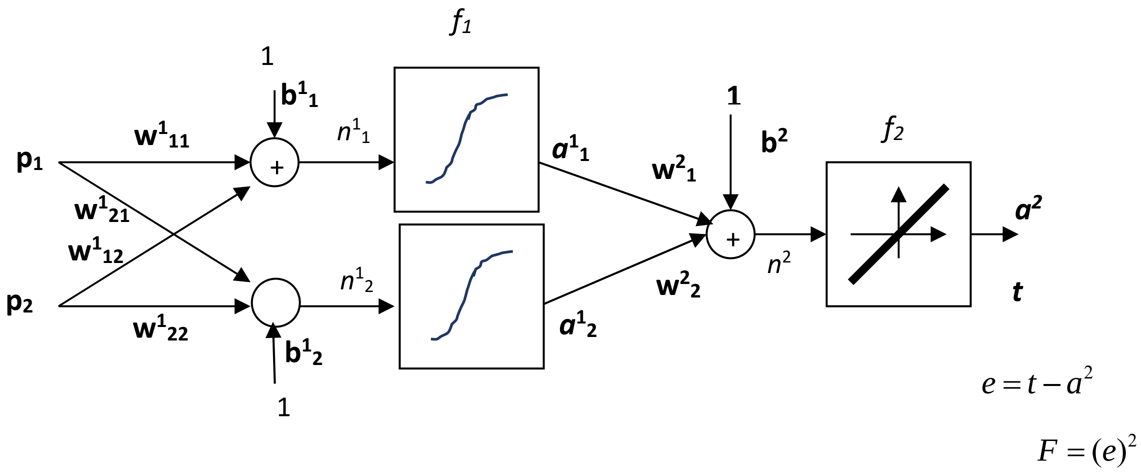
Exemplo 4.9.1. Backpropagation in a two layers network

Consider the NN in the following figure.



For a given input, the output a^3 must equal the target *t.* The activation functions can by anyone, and are represented by f_1 e f_2 respectively in the first and second layers..

Computing the error, $e=t-a^2$

$$
\begin{aligned}
e = t - a^2 &= t - f^2(n^2) = t - f^2(w_1^2 a_1^1 + w_2^2 a_2^1 + b^2) = \\
&= t - f^2\left[w_1^2 f^1(n_1^1) + w_2^2 f^1(n_2^1) + b^2 \right] = \\
&= t - f^2\left[w_1^2 f^1(w_{11}^1 p_1 + w_{12}^1 p_2 + b_1^1) + w_2^2 f^1(w_{21}^1 p_1 + w_{22}^1 p_2 + b_2^1) + b^2 \right]
\end{aligned}
$$

This means that the objective function, given by the square of the error, is a function of 9 arguments,

$$F = (e)^2 = \textit{função}(w_1^2, w_2^2, w_{11}^1, w_{12}^1, w_{21}^1, w_{22}^1, b_1^1, b_2^1, b^2)$$

that are just the parameters that we want to find.

To minimize the error, one has to minimize *F* with respect to each of its arguments, following the gradient method: to minimize any function f (x) in order to its argument x, we start at a point x(0) and then iteration is performed according to

$$x(k+1) = x(k) - \alpha \left. \frac{\partial f}{\partial x} \right|_{x(k)}$$

until convergence is obtained for a minimum, where the derivative is null and so x(k+1)=x(k) afterwards. But in practice the derivative is never null, and we adopt a criteria to stop when th derivative in under a certain pre-defined threshold .

For example w_{11}^1, we must to calculate the chain of derivatives from *F* to w_{11}^1:

$$\frac{\partial F}{\partial w_{11}^1} = \frac{\partial F}{?} \frac{?}{?} \frac{?}{?} \frac{?}{?} \frac{?}{?} \frac{?}{\partial w_{11}^1}$$

To replace the ?? let us return to the figure. We have to follow all the paths starting if *F* and ending in w_{11}^1 . We obtain

$$\frac{\partial F}{\partial w_{11}^1} = \frac{\partial F}{\partial e} \frac{\partial e}{\partial a^2} \frac{\partial a^2}{\partial n^2} \frac{\partial n^2}{\partial a_1^1} \frac{\partial a_1^1}{\partial n_1^1} \frac{\partial n_1^1}{\partial w_{11}^1}$$

Considering now the concrete relations among the several variables,

$$\frac{\partial F}{\partial w_{11}^1} = 2e \text{ .(-1). } \dot{f}^2(n_2).\ w_1^2.\ \dot{f}^1(n_1^1).\ p_1$$

Similarly for w_{12}^1,

$$\frac{\partial F}{\partial w_{12}^1} = \frac{\partial F}{\partial e} \frac{\partial e}{\partial a^2} \frac{\partial a^2}{\partial n^2} \frac{\partial n^2}{\partial a_1^1} \frac{\partial a_1^1}{\partial n_1^1} \frac{\partial n_1^1}{\partial w_{12}^1}$$

Making the calculations,

$$\frac{\partial F}{\partial w_{12}^1} = 2e \text{ .(-1). } \dot{f}^2(n_2).\ w_1^2.\ \dot{f}^1(n_1^1).\ p_2$$

Na for the bias b_1^1,

$$\frac{\partial F}{\partial b_1^1} = \frac{\partial F}{\partial e} \frac{\partial e}{\partial a^2} \frac{\partial a^2}{\partial n^2} \frac{\partial n^2}{\partial a_1^1} \frac{\partial a_1^1}{\partial n_1^1} \frac{\partial n_1^1}{\partial b_1^1} =$$

$$= 2e \text{ .(-1). } \dot{f}^2(n_2).\ w_1^2.\ \dot{f}^1(n_1^1).1$$

We can remark the chain of derivatives is the same for the two weights and the bias in layer 1, except for the last derivative.

This common part expresses the way by which n_1^1 influences the criterion *F.* It can be called the *sensitivity of the criteria with respect to n_1^1,* or the sensitivity in relation to the activation of neuron 1 of the layer 1. Let us then call it s_1^1. So we have simply

$$\frac{\partial F}{\partial w_{11}^1} = s_1^1 p_1 \qquad \frac{\partial F}{\partial w_{21}^1} = s_1^1 p_2 \qquad \frac{\partial F}{\partial b_1^1} = s_1^1.1$$

For w_{21}^1, and applying the same definition of sensitivity ,

$$\frac{\partial F}{\partial w_{21}^1} = \frac{\partial F}{\partial e}\ \frac{\partial e}{\partial a^2}\ \frac{\partial a^2}{\partial n^2}\ \frac{\partial n^2}{\partial a_2^1}\ \frac{\partial a_2^1}{\partial n_2^1}\ \frac{\partial n_1^1}{\partial w_{12}^1} = \frac{\partial F}{\partial n_2^1}\ \frac{\partial n_2^1}{\partial w_{12}^1}$$
$$= s_2^1 p_1$$

For w^1_{22} and for b^1_2 ,

$$\frac{\partial F}{\partial w_{22}^1} = s_2^1 p_2 \qquad \frac{\partial F}{\partial b_2^1} = s_2^1.1 = s_2^1$$

Reorganizing for the first layer we obtain the vectorial expressions.

$$\begin{bmatrix} \dfrac{\partial F}{\partial w_{11}^1} \\ \dfrac{\partial F}{\partial w_{12}^1} \\ \dfrac{\partial F}{\partial b_1^1} \end{bmatrix} = s_1^1 \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} \dfrac{\partial F}{\partial w_{21}^1} \\ \dfrac{\partial F}{\partial w_{22}^1} \\ \dfrac{\partial F}{\partial b_2^1} \end{bmatrix} = s_2^1 \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix}$$

Or still in a more compact form

$$\begin{bmatrix} \dfrac{\partial F}{\partial w_{11}^1} & \dfrac{\partial F}{\partial w_{21}^1} \\ \dfrac{\partial F}{\partial w_{12}^1} & \dfrac{\partial F}{\partial w_{22}^1} \\ \dfrac{\partial F}{\partial b_1^1} & \dfrac{\partial F}{\partial b_2^1} \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix} \begin{bmatrix} s_1^1 & s_2^1 \end{bmatrix}$$

For the second layer we must compute the derivatives in order to $w^2{}_1$, $w_2{}^2$, and b^2

$$\frac{\partial F}{\partial w_1^2} = \frac{\partial F}{\partial e}\ \frac{\partial e}{\partial a^2}\ \frac{\partial a^2}{\partial n^2}\ \frac{\partial n^2}{\partial w_1^2} = 2.e.(-1).\ \dot{f}^2(n^2).a_1^1$$

$$\frac{\partial F}{\partial w_2^2} = \frac{\partial F}{\partial e}\ \frac{\partial e}{\partial a^2}\ \frac{\partial a^2}{\partial n^2}\ \frac{\partial n^2}{\partial w_2^2} = 2.e.(-1).\ \dot{f}^2(n^2).a_2^1$$

$$\frac{\partial F}{\partial b^2} = \frac{\partial F}{\partial e}\ \frac{\partial e}{\partial a^2}\ \frac{\partial a^2}{\partial n^2}\ \frac{\partial n^2}{\partial b^2} = 2.e.(-1).\ \dot{f}^2(n^2).1$$

Calling now s^2 to the sensitivity of F to the activation of n_2,

$$s^2 = \frac{\partial F}{\partial e}\ \frac{\partial e}{\partial a^2}\ \frac{\partial a^2}{\partial n^2}$$

We can write

$$\begin{bmatrix} \dfrac{\partial F}{\partial w_1^2} \\ \dfrac{\partial F}{\partial w_2^2} \\ \dfrac{\partial F}{\partial b^2} \end{bmatrix} = s^2 \begin{bmatrix} a_1^1 \\ a_2^1 \\ 1 \end{bmatrix}$$

Looking close at this equation, we see that it is analogous to those of the first layer, but now the inputs of the second layer are the outputs of the first layer.

Comparing the expressions of the sensitivities s_1^1, s_2^1, e s^2

$$s_1^1 = \dot{f}^1(n_1^1).w_1^2.s^2$$
$$s_2^1 = \dot{f}^1(n_2^1).w_2^2.s^2$$

Defining

$$s^1 = \begin{bmatrix} s_1^1 \\ s_2^1 \end{bmatrix}$$

Will come

$$s^1 = \begin{bmatrix} \dot{f}^1(n_1^1).w_1^2 \\ \dot{f}^1(n_2^1).w_2^2 \end{bmatrix} s^2$$

So, to compute s^1, we backpropagate s_2 through the neurons of the first layer. This is the essence of the backpropagation algorithm.

After the computation of all the partial derivatives, the weights and the bias are updated to new values.

Certainly we have to initialize their values anywhere. Lat us see an example for this case.

Let *P* and *T* be

$$P = \begin{bmatrix} 1 & 4 \\ 2 & 2,5 \end{bmatrix} \qquad T = \begin{bmatrix} 0,5 & 0,5 \end{bmatrix}$$

Initializing the weights and the bias at (for example)

$$\begin{array}{lll} w_{11}^1 = 0,5 & w_{12}^1 = 1,5 & \mathrm{b}_1^1 = 0,3 \\ w_{21}^1 = -0,4 & w_{22}^1 = 3,7 & \mathrm{b}_2^1 = -0,8 \\ w_1^2 = 1 & w_2^2 = -3,7 & \mathrm{b}^2 = 1,7 \end{array}$$

Now we present the first input to the network $p1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

The network computes in a forward way :

$$n_1^1 = 1\times 0,5 + 2\times 2,5 + 0,3 = 3,8 \qquad a_1^1 = \log sig(3,8) = 0,98$$
$$n_2^1 = 1\times(-0,4) + 2\times 3,7 + (-0,8) = 6,2 \qquad a_2^1 = \log sig(6,2) = 1,0$$
$$n^2 = 1\times 0,98 + 1,0\times(-3,7) + 1,7 = -1,02 \qquad a^2 = purlin(-1,02) = -1,02$$

$$e = 0,5 - (-1,02) = 0,5 + 1,02 = 1,52$$

And one forward step is concluded, since the input to the error in the output.

Now let us make a backward pass to compute the partial derivatives and to update the weights and the bias. Let us use the sensitivities defined before,

$$s^2 = \frac{\partial F}{\partial e}\frac{\partial e}{\partial a^2}\frac{\partial a^2}{\partial n^2} = 2.e.(-1).1 = -2e = -2\times 1,52 = -3,04$$

Note that the values of the successive derivatives are computed with the present values of the weights and bias.

Derivative in order to w^1_{11}

$$\frac{\partial F}{\partial w_{11}^1} = s_1^1.p_1 \qquad s_1^1 = \dot{f}^1(n_1^1).w_1^2.s^2$$

Being

$$f^1(n) = \log sig(n) = \frac{1}{1+e^{-n}}$$
$$\frac{\partial f^1}{\partial n} = \frac{-1(-1.e^{-n})}{(1+e^{-n})^2} = \frac{e^{-n}}{(1+e^{-n})^2}$$

and being

$$n_1^1 = 3,8 \ \text{ logo } \ \dot{f}^1(n_1^1) = \frac{e^{-3,8}}{(1+e^{-3,8})^2} = 0,0214$$

We have all we need to compute s^1_1

$$s_1^1 = 0,0214\times 1\times(-3,04) = -0,0651$$

and finally

$$\frac{\partial F}{\partial w_{11}^1} = -0,0651\times 1 = -0,0651$$

We are ready to update the weight following the gradient rule

$$w_{11}^1(1) = w_{11}^1(0) - \alpha \left.\frac{\partial F}{\partial w_{11}^1}\right|_{(0)} = 0,5 - \alpha(-0,0651)$$

We called α the *learning coefficient*. It can be seen that its value determines the amplitude of the variation of the weigh from one iteration to the next one. Generally, it is not advisable a big variation. For example, if we make α=1 we obtain

$$w_{11}^1(1) = w_{11}^1(0) + 0,0651 = 0,5 + 0,0651 = 0,561$$
$$\Delta w_{11}^1 = 0,0651$$

A variation of about 10%, that is not bad.

Derivative in order to w^1_{12}

$$\frac{\partial F}{\partial w_{12}^1} = s_1^1 . p_2 = -0,0651 \times 2 = -0,1302$$

Updating this weight,

$$w_{12}^1(1) = w_{12}^1(0) - \alpha(-0,1302) = 1,5 + 0,1302 = 1,6302 \ \text{ com } \alpha = 1$$
$$\Delta w_{12}^1 = 0,1302$$

Derivative in order to b^1_1

$$\frac{\partial F}{\partial b_1^1} = s_1^1 \qquad b_1^1(1) = b_1^1(0) - \alpha(-0,0652) = 0,3 + 0,0651 = 0,3651 \quad \text{com} \quad \alpha = 1$$

$$\Delta b_1^1 = 0,0651$$

We proceed similarly to all other weights and bias. For example, for w^2_1

$$w_1^2(1) = w_1^2(0) - \alpha . s^2 . a_1^1(0) = 1 - \alpha.(-3,04).0,98 = 1 + 2,98 = 3,98$$
$$\Delta w_1^2 = 2,98$$

Now, after all updates, you can compute the error with the new weights and bias for the same input P1, making another forward pass, and see if it is higher or lower than the previous one. Try it. If it is lower, that is working in the good direction. If it is higher, we cannot conclude, because only after presenting all the inputs we can have the overall picture, by computing the mean square error (MSE).

After all these computations for the first input (forward pass, backward pass), one of two training strategies can be followed:

i) **Iterative training** (in Matlab adapt function): one epoch of training

Untill convergence

1st input

Forward pass;

Backward pass.

Update all w and all b.

Make a forward pass with this input to compute the square of the error and memorize it.

2nd input

Forward pass already with the new weights and bias.

Backward pass.

Update all w and all b.

Make a forward pass with this input to compute the square of the error and sum it to the previous.

....

Last input

Forward pass with updated weigh and bias with the previous input.

Backward pass.

Update all the weights and all the bias.

Make a forward pass with this input to compute the square of the error and sum it to the previous.

Divide the sum of the squared errors by the number of inputs, obtaining the mse. If the mse is lower than the threshold, stop. If not restart with the last weights and bias.

ii) **Batch training** (in Matlab train functions): one epoch of training

Until convergence

1st input

Forward pass

Backward pass

Computation of all Δw and all Δb; keep them in memory

2nd input

Forward pass

Backward pass

Computation of all Δw and all Δb; keep them in memory

....

Last input

Forward pass

Backward pass

Computation of all Δw and all Δb; keep them in memory.

Add all the Δw and all the Δb corresponding to all inputs.

Update the weights and bias with gradient formula with these sums.

Compute the mse by a forward pass with all inputs. If it is lower than the threshold, stop If not restart with the last weights and bias

In batch training, in each epoch, only one update is made. In iterative training an update is made for each input. The evolution of the weight and bias will be different.

Which is the best ?

Depends on the problem and on the available computer. In theory, batch training is closer to the true gradient, but it requires more memory, and the adapt functions take more computational time.

Iterative training is faster and is used frequently in application with a high number of inputs. Real time applications usually use iterative training.

But only experience, trial and error, will give the answer.

In the example iterative training was used.

The backpropagation algorithm is still the working horse in training neural networks, including in deep learning as we will see in Chapt. 5. That is why it is important to understand how it works.

Conjugate gradients methods

You will find backpropagation with variations of the gradient, such as the conjugate gradient technique. They use combinations of the gradient in successive instants in order to introduce second order information to improve the convergence to a minimum, generally local minimum. Some examples follow. The

Consider g_k as the gradient in iteration k and the β_k the value that replaces the gradient in the gradient formula.

$$\Delta g_k = g_{k+1} - g_k$$

$$\text{Hestnes and Stiefel method} \quad \beta_k = \frac{\Delta g_{k-1}^T g_k}{\Delta g_{k-1}^T g_{k-1}}$$

$$\text{Fletcher and Reeves method} \quad \beta_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}}$$

$$\text{Polak and Ribiére method} \quad \beta_k = \frac{\Delta g_{k-1}^T g_k}{g_{k-1}^T g_{k-1}}$$

There are more methods. All try to introduce second order information (second derivatives). The difference of two successive gradients gives information about the gradient of the gradient, i.e, the second derivative (or the Hessian) used by the Newton's method, a second order method.

For more see Hagan and coll. pag 9.16-22.

Coimbra, 12 October 2022.

@ADC.