Multilayer Network Delta rule - Backpropagation

Matteo Gioele Collu - 2056763

In class we have seen how the delta rule is used in order to update a single-layer neural network, where the only layer of neurons happens to be the output layer, which directly contributes to the value of the error that we want to minimize. In such a way it is possible to update easily the neurons' weights by taking the opposite direction of their gradient. If we consider the error as:

$$E = \sum_k \frac{1}{2}(t_k - z_k)^2$$

We define the delta rule as:

$$\delta_k = (t_k - z_k) * \nabla(z_k)$$
$$\Delta_{ik} = \eta * \delta_k * y_i$$

where δ_k represents the gradient with respect to the output k (z_k is the activation of the neuron k), while Δ_{ik} represents the amount that has to be added to the current weight in order to update it and minimize the error(i-k represents the link between the i-th input and the k-th neuron).

When it comes to deal with a multi-layer network, we are faced with neurons that are not directly connected to the error, making it more complex to determine how much each of them is contributing to the error and how.

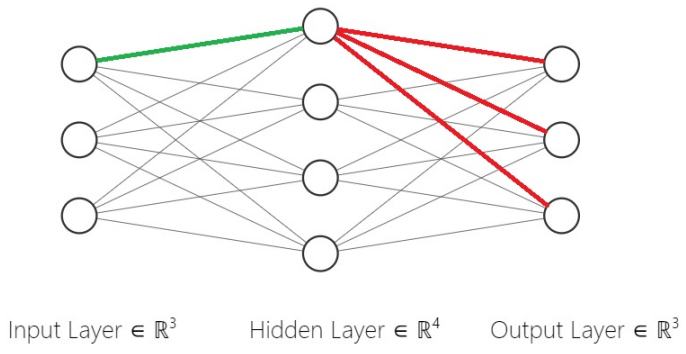

Figure 1: How a previous weight affects the next ones

Moreover, we can see from the Figure 1 how a weight associated to an inner neuron is going to affect and eventually spread its contributed along all the next ones. Then its update must somehow be related to the error that he receives from the later neurons. We can see how it is possible to deal with this complex situation. In order to make this happen, we have to adapt the delta rule to the current situation. The network can be divided in hidden layers and output layer. The output layer is the one which is directly related to the output of the network, then to the error function. In this layer it is possible to apply the delta rule the we have already defined and update the weights accordingly to it. For each layer we will use the information retrieved by its subsequent layer. This information will tell the to the current layer how much each unit is contributing to the error of the next layer. We can rewrite the delta rule as:
For the output layer:

$$\delta_k = (t - z_k) * \nabla(z_k)$$

For an inner layer:

$$\delta_k = \nabla(z_k) * \sum_j W_{kj}\delta_j$$

In both cases:

$$\Delta_{ik} = \eta * \delta_k * y_i$$

As before δ_k refers to the gradient of that specific unit with respect to the error. Since the error is calculated over the last layer in the forward phase, the value of δ for inner units must wait that the values of the next layers have been calculated. This is why this algorithm is called backpropagation and consists of two phases: the forward one, in which the output of the network is calculated, and the backward one, where the weights are updated and the error is propagated starting from the last layer up to the first one.
It is worth note it that the gradient of a unit is obtained by applying the **chain rule** of derivatives. Let's take as example this network where all the units are represented as matrices (Figure2):

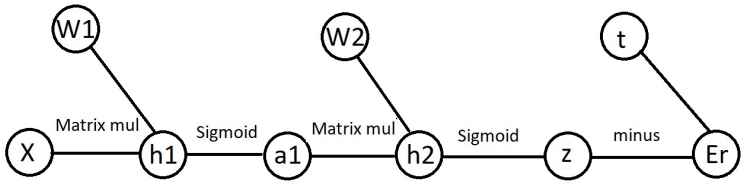

Figure 2: Representation of a NN where all the components are represented singularly

In this example the components are:

- X are the inputs
- W1 are the weights of the first layer
- h1 is the matrix multiplication of the first layer
- a1 is the activation of the first layer
- W2 are the weights of the second layer
- h2 is the matrix multiplication of the second layer between the output of the activation of the first layer and the weights associated
- z is the activation of the second layer, or the output of the network
- t is the target
- Er is the error

With this particular schema, it will be easy to see how the chain rule is applied and how starting from the output layer avoids to repeat some calculations. In this network the forward step corresponds to $\sigma(W2 * \sigma(W1 * X))$, with * intended as matrix multiplication. We would like to know how W1 and W2 contributes to the error, then we would like to know:

$$\frac{\partial Er}{\partial W1} \quad and \quad \frac{\partial Er}{\partial W2}$$

Following the schema and the chain rule, it appears that:

$$\frac{\partial Er}{\partial W2} = \frac{\partial Er}{\partial z}\frac{\partial z}{\partial h2}\frac{\partial h2}{\partial W2}$$

$$\frac{\partial Er}{\partial W1} = \frac{\partial Er}{\partial z}\frac{\partial z}{\partial h2}\frac{\partial h2}{\partial a1}\frac{\partial a1}{\partial h1}\frac{\partial h1}{\partial W1}$$

We can see from the two equations above that they share:

$$\frac{\partial Er}{\partial z}\frac{\partial z}{\partial h2}$$

This information is the δ of the output layer that has been described before and represents the derivative of the layer. As we have seen before, this information will then be passed to a previous layer, which will use it in order to compute its δ. If a new layer was added in the back of the above network

$$\frac{\partial Er}{\partial z}\frac{\partial z}{\partial h2}\frac{\partial h2}{\partial a1}\frac{\partial a1}{\partial h1}$$

would have represented the δ up to the second-last layer. With this, we can see that computing the delta starting from the last layer it's easier and avoid to repeat a lot of calculation.