

# Backward propagation with Mellin convolutions

## 1 Derivative of the $\chi^2$

For the sake of simplicity we consider an uncorrelated  $\chi^2$  for one single experimental point with central value  $F$  and standard deviation  $\sigma$ :

$$\chi^2 = \left( \frac{\hat{F} - F}{\sigma} \right)^2, \quad (1.1)$$

where  $\hat{F}$  is the corresponding theoretical prediction.  $\hat{F}$  is typically computed as a convolution integral between two distinct sets of quantities  $C_i$  and  $N_i$ :

$$\hat{F} \equiv \sum_i C_i \otimes N_i = \mathbf{C} \otimes \mathbf{N}, \quad (1.2)$$

The details of the convolution sign  $\otimes$  are not important, it suffices to know that it involves an integral over a set of input variables  $\{\xi_p\}$ . Therefore, the theoretical prediction  $\hat{F}$  can be regarded as a *functional* of the functions  $N_i$ <sup>1</sup>, *i.e.*  $\hat{F} \equiv \hat{F}[\{N_i\}]$ . It follows that the  $\chi^2$  is also a functional of the functions  $N_i$ , *i.e.*  $\chi^2 \equiv \chi^2[\{N_i\}]$ . In the case we are interested in, the functions  $N_i$  are the outputs of a feed-forward neural network with  $L$  layers (including input and output layers) parametrised by a set of weights  $\omega_{ij}^{(\ell)}$  and biases  $\theta_i^{(\ell)}$ . We assume that the nodes of the  $\ell$ -th layer have all the same activation function  $\sigma_\ell$  associated. The output of the neural network can be written as:

$$\begin{aligned} N_i \equiv N_i(\{\xi_p\}; \{\omega_{ij}^{(\ell)}, \theta_i^{(\ell)}\}) &= \sigma_L \left( \sum_{j^{(1)}}^{N_{L-1}} \omega_{ij^{(1)}}^{(L)} y_{j^{(1)}}^{(L-1)} + \theta_i^{(L)} \right) \\ &= \sigma_L \left( \sum_{j^{(1)=1}}^{N_{L-1}} \omega_{ij^{(1)}}^{(L)} \sigma_{L-1} \left( \sum_{j^{(2)=1}}^{N_{L-2}} \omega_{j^{(1)}j^{(2)}}^{(L)} y_{j^{(2)}}^{(L-2)} + \theta_{j^{(1)}}^{(L-1)} \right) + \theta_i^{(L)} \right) \\ &= \dots \end{aligned} \quad (1.3)$$

We want to compute the following derivatives:

$$\frac{\partial \chi^2}{\partial \omega_{ij}^{(\ell)}} = 2 \left( \frac{\mathbf{C} \otimes \mathbf{N} - F}{\sigma^2} \right) \mathbf{C} \otimes \frac{\partial \mathbf{N}}{\partial \omega_{ij}^{(\ell)}}, \quad (1.4)$$

and:

$$\frac{\partial \chi^2}{\partial \theta_i^{(\ell)}} = 2 \left( \frac{\mathbf{C} \otimes \mathbf{N} - F}{\sigma^2} \right) \mathbf{C} \otimes \frac{\partial \mathbf{N}}{\partial \theta_i^{(\ell)}}. \quad (1.5)$$

Let us first focus on Eq. (1.4). We define:

$$\begin{aligned} x_i^{(\ell)} &= \sum_{j=1}^{N_{\ell-1}} \omega_{ij}^{(\ell)} y_j^{(\ell-1)} + \theta_i^{(\ell)}, \\ y_i^{(\ell)} &= \sigma_\ell \left( x_i^{(\ell)} \right), \\ z_i^{(\ell)} &= \sigma'_\ell \left( x_i^{(\ell)} \right), \end{aligned} \quad (1.6)$$

---

<sup>1</sup> In fact,  $\hat{F}$  is also a functional of the functions  $C_i$  but we assume these functions to be given.

so that we can apply the chain rule:

$$\begin{aligned}
\frac{\partial N_k}{\partial \omega_{ij}^{(\ell)}} &= \frac{\partial y_k^{(L)}}{\partial \omega_{ij}^{(\ell)}} \\
&= z_k^{(L)} \frac{\partial x_k^{(L)}}{\partial \omega_{ij}^{(\ell)}} \\
&= \sum_{j^{(1)}=1}^{N_{L-1}} \left[ z_k^{(L)} \omega_{kj^{(1)}}^{(L)} \right] \frac{\partial y_{j^{(1)}}^{(L-1)}}{\partial \omega_{ij}^{(\ell)}} \\
&= \sum_{j^{(1)}=1}^{N_{L-1}} \sum_{j^{(2)}=1}^{N_{L-2}} \left[ z_k^{(L)} \omega_{kj^{(1)}}^{(L)} \right] \left[ z_{j^{(1)}}^{(L-1)} \omega_{j^{(1)}j^{(2)}}^{(L-1)} \right] \frac{\partial y_{j^{(2)}}^{(L-2)}}{\partial \omega_{ij}^{(\ell)}} \\
&= \dots
\end{aligned} \tag{1.7}$$

As evident, the chain rule penetrates into the neural network starting from the output layer all the way back until the  $\ell$ -th layer (that is, the layer where the parameter  $\omega_{ij}^{(\ell)}$  with respect to which we are deriving belongs to). In order to write the formula we are looking for in a closed form we define:

$$z_i^{(\ell)} \omega_{ij}^{(\ell)} = S_{ij}^{(\ell)} \left( = \frac{\partial y_i^{(\ell)}}{\partial y_j^{(\ell-1)}} \right), \tag{1.8}$$

and using the matricial form we can write:

$$\frac{\partial \mathbf{N}}{\partial \omega_{ij}^{(\ell)}} = \mathbf{S}^{(L)} \cdot \mathbf{S}^{(L-1)} \dots \mathbf{S}^{(\ell+1)} \cdot \frac{\partial \mathbf{y}^{(\ell)}}{\partial \omega_{ij}^{(\ell)}}, \tag{1.9}$$

that can be written in a more compact form as:

$$\frac{\partial \mathbf{N}}{\partial \omega_{ij}^{(\ell)}} = \left[ \prod_{\alpha=L}^{\ell+1} \mathbf{S}^{(\alpha)} \right] \cdot \frac{\partial \mathbf{y}^{(\ell)}}{\partial \omega_{ij}^{(\ell)}}. \tag{1.10}$$

In addition, the derivative in the r.h.s. can be computed explicitly and reads:

$$\frac{\partial y_k^{(\ell)}}{\partial \omega_{ij}^{(\ell)}} = z_k^{(\ell)} \frac{\partial x_k^{(\ell)}}{\partial \omega_{ij}^{(\ell)}} = \delta_{ki} z_i^{(\ell)} y_j^{(\ell-1)}. \tag{1.11}$$

It is simple to see that the derivative in Eq. (1.5) takes the form:

$$\frac{\partial \mathbf{N}}{\partial \theta_i^{(\ell)}} = \left[ \prod_{\alpha=L}^{\ell+1} \mathbf{S}^{(\alpha)} \right] \cdot \frac{\partial \mathbf{y}^{(\ell)}}{\partial \theta_i^{(\ell)}}, \tag{1.12}$$

with:

$$\frac{\partial y_k^{(\ell)}}{\partial \theta_i^{(\ell)}} = \delta_{ki} z_i^{(\ell)}. \tag{1.13}$$

The presence of  $\delta_{ki}$  in both Eqs. (1.11) and (1.13) simplifies the computation of the derivatives yielding:

$$\frac{\partial N_k}{\partial \theta_i^{(\ell)}} = \Sigma_{ki}^{(\ell)} z_i^{(\ell)} \quad \text{and} \quad \frac{\partial N_k}{\partial \omega_{ij}^{(\ell)}} = \Sigma_{ki}^{(\ell)} z_i^{(\ell)} y_j^{(\ell-1)}. \tag{1.14}$$

In both cases, the key quantities to be computed are the matrices:

$$\mathbf{\Sigma}^{(\ell)} = \prod_{\alpha=L}^{\ell+1} \mathbf{S}^{(\alpha)}, \tag{1.15}$$

that can be computed recursively moving backward from the output layer as:

$$\mathbf{\Sigma}^{(\ell-1)} = \mathbf{\Sigma}^{(\ell)} \cdot \mathbf{S}^{(\ell)}, \quad (1.16)$$

starting from:

$$\mathbf{\Sigma}^{(L)} = \mathbf{I}. \quad (1.17)$$

The same technology can be used to compute derivatives of the neural network w.r.t. the input variables  $\{\xi_p\}$ . Indeed, a straightforward application of the chain rule discussed above produces the compact result:

$$\frac{\partial \mathbf{N}}{\partial \boldsymbol{\xi}} = \prod_{\alpha=L}^1 \mathbf{S}^{(\alpha)} = \mathbf{\Sigma}^{(0)}, \quad (1.18)$$

or, making the indices explicit:

$$\frac{\partial N_k}{\partial \xi_p} = \Sigma_{kp}^{(0)}. \quad (1.19)$$

A generalisation of feed-forward neural network that might be useful considering is one in which the linear combination of weights, biases, and inputs form the preceding layer that enter the computation of a given node (see first identity of Eq. (1.6)) is replaced by:

$$x_i^{(\ell)} = \sum_{j=1}^{N_{\ell-1}} f(\omega_{ij}^{(\ell)}) y_j^{(\ell-1)} + \theta_i^{(\ell)}, \quad (1.20)$$

where  $f$  is some differentiable function. This change has the effect of changing the matrices  $\mathbf{S}^{(\ell)}$  as follows:

$$S_{ij}^{(\ell)} = z_i^{(\ell)} f(\omega_{ij}^{(\ell)}), \quad (1.21)$$

and the derivatives w.r.t. the weight  $\omega_{ij}^{(\ell)}$  as follows:

$$\frac{\partial N_k}{\partial \omega_{ij}^{(\ell)}} = \Sigma_{ki}^{(\ell)} z_i^{(\ell)} y_j^{(\ell-1)} f'(\omega_{ij}^{(\ell)}). \quad (1.22)$$

An interesting application of this generalisation is obtained by setting  $f(x) = e^x$  [1]. Indeed, upon this choice, one can show that, under the assumption of monotonically increasing activation functions, the resulting neural network is also an increasing monotonic function of *all* of its input variables. However, given an arbitrary number of inputs, one may want to enforce monotonicity only on a subset of them, allowing the others to be non-monotonic. This goal is achieved by using  $f(x) = x$ , rather than  $f(x) = e^x$ , for those links associated with input nodes for which monotonicity is not required. More specifically, suppose one has a neural network with  $N_0$  input nodes and  $N_1$  nodes in the first hidden layer (the rest of the architecture is unimportant). Non-monotonicity on the  $j$ -th input variable is achieved by using the linear function  $f(x) = x$  on the links  $\omega_{ij}^{(1)}$ , with  $i = 1, \dots, N_1$ , *i.e.* those links that connect the  $j$ -th node of the input layer with all nodes of the first hidden layer.

## References

- [1] Zhang, H. & Zhang, Z. Feedforward networks with monotone constraints. *IJCNN'99. International Joint Conference On Neural Networks. Proceedings (Cat. No.99CH36339)*. **3** pp. 1820-1823 vol.3 (1999)