# 1

# Fully Connected NN Model

## 1.1 Derivatives with respect to model parameters and input

Consider a fully connected neural network (FNN) model, with $k-1$ hidden layers, with input $Z$ and output $Y$ of the form

$$\begin{aligned} W_1 &= \mathbf{A}_1 Z + \mathcal{B}_1 \\ Z_1 &= f_1(W_1) \\ &\vdots \\ W_{k-1} &= \mathbf{A}_{k-1} Z_{k-2} + \mathcal{B}_{k-1} \\ Z_{k-1} &= f_{k-1}(W_{k-1}) \\ W_k &= \mathbf{A}_k Z_{k-1} + \mathcal{B}_k \\ Y &= f_k(W_k), \end{aligned} \tag{1.1}$$

where $f_i(W), i = 1, \ldots, k$ are scalar functions applied to each elements of the vector $W$. The parameters of the network are $\theta = \{\mathbf{A}_1, \mathcal{B}_1, \ldots, \mathbf{A}_k, \mathcal{B}_k\}$ and the goal is to compute the derivative with respect to each elements of these matrices and vectors and, also, with respect to the input $Z$. In fact as we will see we can group the derivatives into matrices for each level separately and compute them recursively.

Let us start our derivations. For (stochastic) gradient type algorithms we usually have some form of scalar cost function $\phi(Y)$ that we like to compute its gradient with respect to the network parameters and input. Therefore, starting from the output and going backwards, we can write

$$\nabla_{[\mathbf{A}_k \mathcal{B}_k]} \phi(Y) = \big(\nabla_Y \phi(Y) \odot f_k'(W_k)\big)[Z_{k-1}^\intercal \; 1] \tag{1.2}$$

$$\nabla_{Z_{k-1}} \phi(Y) = \mathbf{A}_k^\intercal \big(\nabla_Y \phi(Y) \odot f_k'(W_k)\big), \tag{1.3}$$

where $f'(W)$ denotes the derivative of the scalar function $f(\cdot)$ applied to each element of the vector $W$ and $\odot$ denotes element-by-element multiplication. Let us now go one step deeper into the network. We then have

$$\nabla_{[\mathbf{A}_{k-1} \mathcal{B}_{k-1}]_{ij}} \phi(Y) = \big(\nabla_{Z_{k-1}} \phi(Y)\big)^\intercal \frac{\partial Z_{k-1}}{\partial [\mathbf{A}_{k-1} \mathcal{B}_{k-1}]_{ij}},$$

from which we conclude that

$$\nabla_{[\mathbf{A}_{k-1}\mathcal{B}_{k-1}]}\phi(Y) = \big(\nabla_{Z_{k-1}}\phi(Y) \odot f'_{k-1}(W_{k-1})\big)[Z_{k-2}^{\intercal}\ 1]. \tag{1.4}$$

Similarly

$$\nabla_{[Z_{k-2}]_i}\phi(Y) = \big(\nabla_{Z_{k-1}}\phi(Y)\big)^{\intercal}\frac{\partial Z_{k-1}}{\partial [Z_{k-2}]_i},$$

therefore

$$\nabla_{Z_{k-2}}\phi(Y) = \mathbf{A}_{k-2}^{\intercal}\big(\nabla_{Z_{k-1}}\phi(Y) \odot f'_{k-1}(W_{k-1})\big). \tag{1.5}$$

We can now deduce the following recursion formula (adopting the engineering induction). For $i = k, k-1, \ldots, 1$ define

$$\begin{aligned} \mathscr{V}_i &= \mathscr{U}_i \odot f'_i(W_i), \ \text{ initializing with } \mathscr{U}_k = \nabla_Y \phi(Y), \\ \mathscr{U}_{i-1} &= \mathbf{A}_i^{\intercal}\mathscr{V}_i, \end{aligned} \tag{1.6}$$

which generates $\{\mathscr{V}_k, \ldots, \mathscr{V}_1\}$ and $\{\mathscr{U}_{k-1}, \ldots, \mathscr{U}_0\}$. Then

$$\begin{aligned} \nabla_{[\mathbf{A}_i\mathcal{B}_i]}\phi(Y) &= \mathscr{V}_i[Z_{i-1}^{\intercal}\ 1] \\ \nabla_Z \phi(Y) &= \mathscr{U}_0. \end{aligned} \tag{1.7}$$

### 1.1.1 Example

Let us apply these formulas to the simple case of one hidden layer. We have that the neural network equations are of the form

$$\begin{aligned} W_1 &= A_1 Z + B_1 \\ Z_1 &= f_1(W_1) \\ W_2 &= A_2 Z_1 + B_2 \\ Y &= f_2(W_2), \end{aligned}$$

Suppose we are interested in some function $\phi(Y)$ of the output. According to our formulas we define:

$$\mathscr{U}_2 = \nabla_Y \phi(Y),\ \ \mathscr{V}_2 = \mathscr{U}_2 \odot f'_2(W_2),\ \ \mathscr{U}_1 = A_2^{\intercal}\mathscr{V}_2,\ \ \mathscr{V}_1 = \mathscr{U}_1 \odot f'_1(W_1),\ \ \mathscr{U}_0 = A_1^{\intercal}\mathscr{V}_1,$$

that allows us to compute the gradients

$$\nabla_{[A_2 B_2]}\phi(Y) = \mathscr{V}_2[Z_1^{\intercal}1],\ \ \nabla_{[A_1 B_1]}\phi(Y) = \mathscr{V}_1[Z^{\intercal}1],\ \ \nabla_Z \phi(Y) = \mathscr{U}_0.$$

## 1.2 Update of Network Parameters

If for example our goal is to select the network parameters to *minimize* the expectation of $\phi(Y)$ then we can apply a stochastic gradient descent of the form

$$[\mathbf{A}_k \mathcal{B}_k]_t = [\mathbf{A}_k \mathcal{B}_k]_{t-1} - \mu \nabla_{[\mathbf{A}_k \mathcal{B}_k]_{t-1}}\phi(Y_t).$$

However it has been observed in practice that normalizing the derivatives produces more stable solutions. In particular we can use the ADAM scheme (Kingma, D. P., Ba, J. L. (2015). Adam:

a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13) which consists in computing an estimate of the power of each derivative element

$$\mathbf{P}_{[\mathbf{A}_k \mathcal{B}_k]}(t) = (1-\lambda)\mathbf{P}_{[\mathbf{A}_k \mathcal{B}_k]}(t-1) + \lambda \left(\nabla_{[\mathbf{A}_k \mathcal{B}_k]_{t-1}} \phi(Y_t)\right)^{(.2)}$$

where for a matrix $\mathbf{Q}$ we denote with $\mathbf{Q}^{(.2)}$ the element-by-element raise to the power 2 and $\lambda \ll 1$. We initialize with

$$\mathbf{P}_{[\mathbf{A}_k \mathcal{B}_k]}(1) = \left(\nabla_{[\mathbf{A}_k \mathcal{B}_k]_0} \phi(Y_1)\right)^{(.2)}.$$

The power estimate must be updated after we compute the gradient $\nabla_{[\mathbf{A}_k \mathcal{B}_k]_{t-1}} \phi(Y_t)$.

When updating the network parameters, we can normalize each gradient element using the corresponding power estimate as follows

$$[\mathbf{A}_k \mathcal{B}_k]_t = [\mathbf{A}_k \mathcal{B}_k]_{t-1} - \mu \nabla_{[\mathbf{A}_k \mathcal{B}_k]_{t-1}} \phi(Y_t) ./ \sqrt{c + \mathbf{P}_{[\mathbf{A}_k \mathcal{B}_k]}(t)}$$

where $./$ denotes element-by-element division, $c$ is a small number to avoid division by 0, and the square-root ($\sqrt{\ }$) is again applied element-by element onto the matrix $c + \mathbf{P}_{[\mathbf{A}_k \mathcal{B}_k]}(t)$. This way we normalize each derivative and make the elements of the same order of magnitude which then allows for the use of a single step size $\mu$ for all adaptations. Otherwise, if we do not normalize, and use a single $\mu$ then we experience completely different convergence rates per parameter element!