Neural networks and the backpropagation algorithm

S. Marchand-Maillet

Computer Science – University of Geneva

1 introduction

Given a function $f : \mathbb{R}^D \to \mathbb{R}^d$, we wish to approximate (regress) it from training data $\mathcal{X} \times \mathcal{Y} = \{(x_i, y_i)\}_{i=1}^N$ (where $y_i = f(x_i)$) using a mapping

$$\begin{array}{rccc} \phi_\theta : & \mathbb{R}^D & \to & \mathbb{R}^d \\ & x_i & \mapsto & \phi_\theta(x_i) \end{array}$$

parameterized with parameters θ (using notation from statistics).

In machine learning terms, f is the mapping that underlies the phenomenon we wish to learn and is unknown. We can access annotated samples $\mathcal{X} \times \mathcal{Y} = \{(x_i, y_i)\}_{i=1}^N$ and we wish to train a learner ϕ_θ by adjusting its parameters θ to minimize a loss function $\mathcal{L}(\mathcal{X} \times \mathcal{Y}; \theta)$.

Here, we present $\phi_\theta(\cdot)$ as a feedforward neural network taking D input values and producing d output values. We will further consider the practical case of the squared loss:

$$\mathcal{L}(\mathcal{X} \times \mathcal{Y}; \theta) = \frac{1}{2}\frac{1}{N}\sum_{i=1}^{N} \|\phi_\theta(x_i) - y_i\|_2^2 = \frac{1}{2N}\sum_{i=1}^{N}\sum_{k=1}^{d} (\phi_\theta(x_i)_{[k]} - y_{i[k]})^2 \tag{1}$$

Note. When the label y_i is of dimension $d = 1$, the squared loss reduces to the classical

$$\mathcal{L}(\mathcal{X} \times \mathcal{Y}; \theta) = \frac{1}{2N}\sum_{i=1}^{N} (\hat{y}_i - y_i)^2 = \frac{1}{2N}\sum_{i=1}^{N} (\phi_\theta(x_i) - y_i)^2$$

A Neural Network model is a Directed Acyclic Graph (DAG) of units (referred to as "neurons") organized in layers (see figure 2 for an example). In the particular case of a feedforward network, conventionally, connections between neurons only exist from one layer to the next.

1.1 "Neuron" model and notation

A generic neuron is located at layer $l \in [\![L]\!]$ in a network of L layers and takes its inputs from the outputs of other neurons. To ease the presentation and because this is the dominant "feedforward" model, we assume that the neuron takes its inputs from the output of the neurons in the previous layer $l-1$ (see figure 1)[a].

A neuron (l, k) located at index $k \in [\![N_l]\!]$ of layer $l \in [\![L]\!]$ comprising N_l neurons:

1. receives its inputs from weighted connections with weights w_{jk}^l
2. aggregates these inputs into a linear sum into its activation $a_k^l = \sum_{j=0}^{N_{l-1}} w_{jk}^l \phi_k^{l-1}$
3. outputs the result of its activation function $\varphi_k^l(\cdot)$ applied to its activation: $\phi_k^l = \varphi_k^l(a_k^l)$

Note that in this model, the weights are attached to the neuron itself. They form the adjustable parameters of the model

$$\theta = \{w_{jk}^l \parallel j \in [\![N_{l-1}]\!], k \in [\![N_l]\!], l \in [\![L]\!]\}$$

The hyperparameters of the model concern its structure and choice of activation functions, i.e $L > 0$, $\{N_l\}_{l=1}^L$ and $\{\varphi_k^l\}_{k,l=1}^{N_l,L}$.

Also following the feedforward structure, we fix the following boundary conditions for the model:

$$\begin{cases} \phi_k^0 = x_{i[k]} & N_0 = D \quad \text{(input)} \\ \phi_\theta(x_i) = \phi^L(x_i) & N_L = d \quad \text{(output)} \end{cases}$$

[a] A more generic formulation using the notation for graph adjacency is left as exercise.

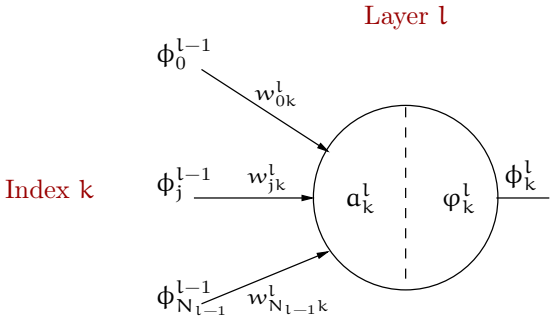

Figure 1: A generic neuron in a feedforward network, taking input from the previous layer

where the notation $\phi_k^l(x_i)$ indicates the output of neuron (l,k) when the network takes x_i as input. $a_k^l(x_i)$ is defined similarly.

Using variable homogenization, it is also customary to consider that $\phi_0^l = 1$ where a bias (hence w_0^l) should be introduced in order to escape from the pure linear activation.

Note. The logistic neuron defines its activation functions as the logistic function:

$$\varphi_k^l(z) = \frac{1}{1+e^{-z}} \qquad \forall k \in [\![N_l]\!] \; \forall l \in [\![L]\!]$$

The present study is more general. It is left as exercise to model the logistic regression as a neural network.

As a result we have the following system of equations:

$$\begin{cases} \phi_k^l = \varphi_k^l(a_k^l) & \forall k \in [\![N_l]\!] \quad \forall l \in [\![L]\!] \\ a_k^l = \displaystyle\sum_{j=0}^{N_{l-1}} w_{jk}^l \phi_j^{l-1} & \forall k \in [\![N_l]\!] \quad \forall l \in [\![L]\!] \\ \phi_k^0 = x_{i\,[k]} & \forall k \in [\![D]\!] \\ \phi_0^l = 1 & \forall l \in [\![L-1]\!] \\ \phi_\theta(x_i)_{[k]} = \phi_k^L(x_i) & \forall k \in [\![d]\!] \quad \forall i \in [\![N]\!] \\ \mathcal{L}(\mathcal{X} \times \mathcal{Y}; \theta) = \dfrac{1}{2N} \displaystyle\sum_{i=1}^{N} \sum_{k=1}^{d} (\boldsymbol{\phi}_\theta(x_i)_{[k]} - y_{i\,[k]})^2 \end{cases}$$

to adjust the parameters of the model $\theta = \left\{ w_{jk}^l \parallel j \in [\![N_{l-1}]\!], k \in [\![N_l]\!], l \in [\![L]\!] \right\}$.

Note. In this document, the assumption is that of a DAG. The connectivity may be more general than "feedforward" (e.g with "skip-connections" connecting layers that are not successive). Only the notation would change.

1.2 Training the network using the backpropagation algorithm

The aim of training is the minimization of the approximation (regression, prediction) error (the "loss"):

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \, \mathcal{L}(\mathcal{X} \times \mathcal{Y}; \theta)$$

As indicated, the parameters of the models are the weights $\theta = \{w_{jk}^l\}_{j,k,l}$.

The algorithm used for training is the gradient descent over the error function to adjust the parameters. In other words, the simple structure of the algorithm is as shown in algorithm 1.

The key element of the algorithm is therefore the computation of all values of

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^l} \qquad \forall j \in [\![N_{l-1}]\!], k \in [\![N_l]\!], l \in [\![L]\!]$$

as the variation of the loss against the modification of weights w_{jk}^l.

Algorithm 1 The base structure of the backpropagation algorithm

procedure BACKPROPAGATION($\mathcal{X}$,$\mathcal{Y}$)
  $w_{jk}^{l,(0)} \leftarrow \mathsf{random}()\ \forall j \in [\![N_{l-1}]\!], k \in [\![N_l]\!], l \in [\![L]\!]$ ▷ Initialize random weights
  for iteration $t = 1$ until $t = T$ **do** ▷ or until convergence
    for $i \in [\![N]\!]$ **do** ▷ Gradient iteration for data x_i
      for $j \in [\![N_{l-1}]\!]$, $k \in [\![N_l]\!]$, $l \in [\![L]\!]$ **do**
        $w_{jk}^{l,(t)} \leftarrow w_{jk}^{l,(t-1)} - \frac{1}{N}\eta \frac{\partial \mathcal{L}}{\partial w_{jk}^l}(x_i)$ ▷ $\eta > 0$ is the gradient step (learning rate)

The name "backpropagation" for this algorithm comes from the realization that when the network has more than one layer ($L > 1$), there is no direct access to $\frac{\partial \mathcal{L}}{\partial w_{jk}^l}$. The chain rule should be used and the gradient propagated back ("backpropagated") from the end of the network towards its input.

In the specific case of the squared loss, given $p \in [\![N_{l-1}]\!]$, $r \in [\![N_l]\!]$, $m \in [\![L]\!]$ we write

$$\frac{\partial \mathcal{L}}{\partial w_{pr}^m} = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{d} \frac{\partial \phi_\theta[k]}{\partial w_{pr}^m}(x_i) \left(\phi_\theta(x_i)[k] - y_i[k]\right)$$

where $\phi_\theta[k]$ is the k^{th} component function of $\phi_\theta(\cdot)$. By definition, $\phi_\theta[k](x_i) = \phi_k^L(x_i)$ so we can write

$$\frac{\partial \phi_\theta[k]}{\partial w_{pr}^m}(x_i) = \frac{\partial \phi_k^L}{\partial w_{pr}^m}(x_i) \overset{[1]}{=} \left[\frac{\partial \phi_k^L}{\partial a_k^L} \cdot \frac{\partial a_k^L}{\partial w_{pr}^m}\right](x_i)$$

[1]: using the chain rule for derivation.

Now,

$$\frac{\partial \phi_k^l}{\partial w_{pr}^m} = \frac{\partial \varphi_k^l}{\partial a_k^l} \cdot \frac{\partial a_k^l}{\partial w_{pr}^m} \tag{2}$$

$$\frac{\partial a_k^l}{\partial w_{pr}^m} = \sum_{j=0}^{N_{l-1}} w_{jk}^l \frac{\partial \phi_j^{l-1}}{\partial w_{pr}^m} \tag{3}$$

$$\Rightarrow \quad \frac{\partial \phi_k^l}{\partial w_{pr}^m} = \frac{\partial \varphi_k^l}{\partial a_k^l} \cdot \sum_{j=0}^{N_{l-1}} w_{jk}^l \frac{\partial \phi_j^{l-1}}{\partial w_{pr}^m}$$

With the following particular cases:

$$\frac{\partial a_r^m}{\partial w_{pr}^m} = \frac{\partial}{\partial w_{pr}^m}\left[\sum_{j=0}^{N_{m-1}} w_{jr}^m \phi_j^{m-1}\right] = \phi_p^{m-1}$$

and for $p = 0$,

$$\frac{\partial a_r^m}{\partial w_{0r}^m} = \phi_0^{m-1} = 1 \qquad \forall m \in [\![L]\!]$$

or, if $k \neq r$ or $1 < l < m$, w_{pr}^m is not a involved in the computation of a_k^l and

$$\frac{\partial a_k^l}{\partial w_{pr}^m} = \frac{\partial}{\partial w_{pr}^m}\left[\sum_{j=0}^{N_{l-1}} w_{jk}^l \phi_j^{l-1}\right] = 0 \tag{4}$$

finally, if $m = 1$ and $p \in [\![D]\!]$,

$$\frac{\partial a_r^1}{\partial w_{pr}^1}(x_i) = \frac{\partial}{\partial w_{pr}^1}\left[\sum_{j=0}^{N_0} w_{jr}^1 \phi_j^0\right](x_i) = \phi_p^0(x_i) = x_i[p]$$

This set of equations (in particular (3) and (2)) creates a recursive chain of computation for derivatives, starting from layer L backpropagating values towards the first (input) layer. Note that equation (4) confirms that the backtracking of derivation of $\frac{\partial a_k^l}{\partial w_{pr}^m}$ concern only paths in the network containing w_{pr}^m and stopping at layer m (equation (4)). We can therefore define the recursive algorithm 2 where the highlighted functions ensure the recursive backpropagation.

Clearly, the weights $\{w_{jk}^l\}_{jkl}$ may be stored in 3D arrays. Similarly, the values for $\{a_k^l(x_i)\}_{ikl}$ and $\{\phi_k^l(x_i)\}_{ikl}$ may be stored using matrices (for every x_i) or 3D arrays. Algebraic operations (addition, multiplication and more) can be defined over these so-called "tensors" and accelerated using GPUs (or TPUs).

Algorithm 2 The recursive backpropagation algorithm

function $\frac{\partial \phi_k^l}{\partial w_{pr}^m}(\boldsymbol{x}_i)$ ▷ Use chain rule since $\phi_k^l = \varphi_k^l(a_k^l)$
  return $\frac{\partial \varphi_k^l}{\partial a_k^l} \cdot \frac{\partial a_k^l}{\partial w_{pr}^m}(\boldsymbol{x}_i)$

function $\frac{\partial \varphi_k^l}{\partial a_k^l}(\boldsymbol{x}_i)$ ▷ Activation function
  Depends on l , k and the definition of $\varphi_k^l(z)$
  e.g if $\varphi_k^l(z) = z$ **return** 1
  e.g if $\varphi_k^l(z) = z^2$ **return** $2a_k^l(\boldsymbol{x}_i)$
  e.g if $\varphi_k^l(z) = \frac{1}{1+e^{-z}}$ **return** $\varphi(a_k^l(\boldsymbol{x}_i))(1 - \varphi(a_k^l(\boldsymbol{x}_i)))$

function $\frac{\partial a_k^l}{\partial w_{pr}^m}(\boldsymbol{x}_i)$ ▷ Activation
  if $l > m$ **then return** $\sum_{j=0}^{N_{l-1}} w_{jk}^l \frac{\partial \phi_j^{l-1}}{\partial w_{pr}^m}(\boldsymbol{x}_i)$
  if $l = m$ **then**
    if $k = r$ **then return** $\phi_p^{m-1}(\boldsymbol{x}_i)$
    else return 0
  if $l = 1$ **then return** $\boldsymbol{x}_{i[p]}$

function $\frac{\partial \mathcal{L}}{\partial w_{jk}^l}(\boldsymbol{x}_i)$
  return $\sum_{r=1}^{d} \frac{\partial \phi_r^L}{\partial w_{jk}^l}(\boldsymbol{x}_i) \left(\phi_r^L(\boldsymbol{x}_i) - \boldsymbol{y}_{i[k]}\right)$

procedure BACKPROPAGATION($\mathcal{X}$,$\mathcal{Y}$)
  $w_{jk}^{l,(0)} \leftarrow \mathsf{random}()\ \forall j \in [\![N_{l-1}]\!], k \in [\![N_l]\!], l \in [\![L]\!]$ ▷ Initialize random weights
  for iteration $t = 1$ until $t = T$ **do** ▷ or until convergence
    for $i \in [\![N]\!]$ **do**
      Forward pass computes and stores values for $a_k^l(\boldsymbol{x}_i)$ and $\phi_k^l(\boldsymbol{x}_i)$ $\forall k, l$ ▷ Forward pass for $\boldsymbol{x}_i$
      for $j \in [\![N_{l-1}]\!]$, $k \in [\![N_l]\!]$, $l \in [\![L]\!]$ **do**
        $w_{jk}^{l,(t)} \leftarrow w_{jk}^{l,(t-1)} - \frac{1}{N}\eta \frac{\partial \mathcal{L}}{\partial w_{jk}^l}(\boldsymbol{x}_i)$ ▷ $\eta > 0$ is the gradient step (learning rate)

1.3 Example

1.3.1 Quadratic regression

Given $\mathcal{X} \times \mathcal{Y}$, a quadratic regression considers the squared loss and a learner of the quadratic family:

$$\phi_\theta(x_i)_{[k]} = (\omega_{2k}^\top x_i)^2 + \omega_{1k}^\top x_i + \omega_{0k}^\top \mathbf{1}_D \tag{5}$$

ϕ_θ may be represented with a 3-layer feedforward network with $N_1 = 2$, $\varphi_1^1(z) = z$, $\varphi_2^1(z) = z^2$, $\varphi_2^2(z) = z$, as represented in figure 2.

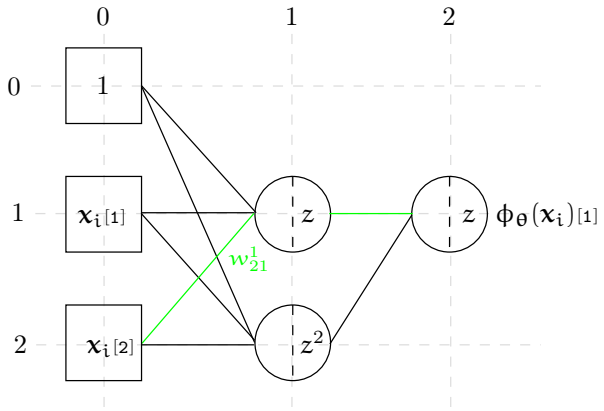

Figure 2: A neural network performing quadratic regression ($D = 2$, $d = 1$). Square nodes simply output their values. The green path shows the chain rule used for computing $\frac{\partial \mathcal{L}}{\partial w_{21}^1}$

Following our notation, we obtain for the case $D = 2$, $d = 1$:

$$\phi_\theta(x_i) = w_{11}^2 \overbrace{(w_{01}^1 + w_{11}^1 x_{i[1]} + w_{21}^1 x_{i[2]})}^{a_1^1} + w_{21}^2 \overbrace{(w_{02}^1 + w_{12}^1 x_{i[1]} + w_{22}^1 x_{i[2]})}^{a_2^1}{}^2 \tag{6}$$

$$= w_{11}^2 \varphi_1^1(a_1^1) + w_{21}^2 \varphi_2^1(a_2^1) \tag{7}$$

Considering a squared loss, one obtains

$$\frac{\partial \mathcal{L}}{\partial w_{jk}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \phi_\theta}{\partial w_{jk}^l}(x_i)(\phi_\theta(x_i) - y_i)$$

and

$$\frac{\partial \phi_\theta}{\partial w_{01}^1}(x_i) = w_{11}^2 \qquad \frac{\partial \phi_\theta}{\partial w_{11}^1}(x_i) = w_{11}^2 x_{i[1]} \qquad \frac{\partial \phi_\theta}{\partial w_{21}^1}(x_i) = w_{11}^2 x_{i[2]}$$

$$\frac{\partial \phi_\theta}{\partial w_{02}^1}(x_i) = 2w_{21}^2(w_{02}^1 + w_{12}^1 x_{i[1]} + w_{22}^1 x_{i[2]}) \qquad \frac{\partial \phi_\theta}{\partial w_{12}^1}(x_i) = 2w_{21}^2 x_{i[1]}(w_{02}^1 + w_{12}^1 x_{i[1]} + w_{22}^1 x_{i[2]})$$

$$\frac{\partial \phi_\theta}{\partial w_{22}^1}(x_i) = 2w_{21}^2 x_{i[2]}(w_{02}^1 + w_{12}^1 x_{i[1]} + w_{22}^1 x_{i[2]}) \qquad \frac{\partial \phi_\theta}{\partial w_{11}^2}(x_i) = w_{01}^1 + w_{11}^1 x_{i[1]} + w_{21}^1 x_{i[2]}$$

$$\frac{\partial \phi_\theta}{\partial w_{21}^2}(x_i) = (w_{02}^1 + w_{12}^1 x_{i[1]} + w_{22}^1 x_{i[2]})^2$$

where one can see that the development of the chain rule follows paths according to equations derived in section 1.2. For example (see figure 2):

$$\frac{\partial \phi_\theta}{\partial w_{21}^1}(x_i) = \frac{\partial \phi_1^2}{\partial a_1^2} \cdot \frac{\partial a_1^2}{\partial \phi_1^1} \cdot \frac{\partial \phi_1^1}{\partial a_1^1} \cdot \frac{\partial a_1^1}{\partial w_{21}^1}(x_i) = 1 \cdot w_{11}^2 \cdot 1 \cdot x_{i[2]}$$

These are the only non-zero terms from the chain

$$\frac{\partial \phi_\theta}{\partial w_{21}^1}(x_i) = \frac{\partial \phi_1^2}{\partial a_1^2} \cdot \sum_{j=0}^{N_1} \left[\frac{\partial a_1^2}{\partial \phi_j^1} \cdot \frac{\partial \phi_j^1}{\partial a_j^1} \cdot \frac{\partial a_j^1}{\partial w_{21}^1} \right](x_i)$$

since $\frac{\partial a_2^1}{\partial w_{21}^1} = 0$ by equation (4).

Note. Developing equation (5) for this case, one can see that the neural model is over-specified: many combinations of weights lead to the same solution. To fix a unique solution, one can e.g fix and not update (clamp) w^1_{02}, w^2_{11} and w^2_{21}.