Methodological approaches in Machine Learning: Neural networks

A. N. Yannacopoulos

November 12, 2025

Contents

1 Basic idea

Neural networks are essentially models $f : X \to Y$ where X is an input space and Y is an output space where the model function f is of a specific type consisting of a composition of affine functions with nonlinear functions (called activation functions). For a general introduction to neural networks see Calin (2020), Bengio et al. (2017). Our presentation here is based on Calin (2020).

An example of a neural network is given in figure 1. A neural network consists of several variables, which are collected in layers.

- The features $X = (x_1, \cdots, x_n)$ are all collected in the first layer (often called the zeroth layer) which is the input layer.
- The inputs are used to construct intermediate variables (using affine functions) $a_i^{(\ell)}$, which are then imported into specific nonlinear functions ϕ, to provide new features. This is done in internal layers, called hidden layers. This procedure can be repeated as many times as we wish, i.e. a NN can can a large number of hidden layers. A NN with many hidden layers is called a Deep Neural Network (DNN).
- At the last hidden layers we take the final set of internal variables and these are used to construct the responses $Y = (y_1, \cdots, y_m)$ again using affine functions of the final set of internal variables and feeding them into the nonlinear functions ϕ. This is doen in the last layer which is called the output layer. Clearly, y_i are complicated functions of the x_j and this is our model.

The nonlinear functions ϕ are called the activation functions.

The example of figure 1 displays a neural network with 3 hidden layers. We have an 4 dimensional input ($X = (x_1, x_2, x_3, x_4)$ relabeled as $(a_1^{(0)}, a_2^{(0)}, a_3^{(0)}, a_4^{(0)})$ and a 3 dimensional response ($Y = (y_1, y_2, y_3)$, relabeled as $(a_1^{(4)}, a_2^{(4)}, a_3^{(4)})$). The $a_j^{(\ell)}$, $\ell = 1, 2, 3$ are the intermediate variables, produced in the 3 hidden layers.

How are the intermediate variables produced?

Simply by using arbitrary weights W to take linear combinations of the other variables, and then feed the result into the activation function ϕ. For example, to go from the input layer to the first hidden layer

$$\begin{aligned} \bar{a}_j^{(1)} &= \sum_{i=1}^{4} w_{ij}^{(1)} a_i^{(0)}, \\ a_j^{(1)} &= \phi(\bar{a}_j^{(1)}), \end{aligned}$$

for all $j = 1, \cdots, 5$. The same procedure, is repeated for hidden layers (using different weights) and the for the output layer. The activation functions are fixed, and the parameters of the model are the weights that are used. Collecting everything together we end up with a parametric model

$$y = f(x; W)$$

and the arbitrary weights are to be chosen so that $f(x; W)$ matches as closely as possible the observations for the responses, given the features X. The fidelity of the model is provided in terms of a loss function L.

Hence, we have

$$\begin{aligned} L(W) &= L(y, f(x, W)), \\ W^* &= \arg\min_W L(y, f(x, W)). \end{aligned}$$

Even for our simple model in figure 1 we end up with an enormous number of parameters:

- 20 for the transition between the input layer and the first hidden layer.
- 25 for the transition between the first hidden layer and the second.
- 25 for the transition between the second hidden layer and the third.
- 15 for the transition between the third hidden layer and the output layer.
- 85 parameters in total!

Models of this type, though complicated and sometimes difficult to interpret offer great potential for modelling complex responses in terms of features. Because of that they have become very popular models in machine learning and data analysis.

2 Towards a general formulation

Clearly, we need a consistent and systematic way to describe and construct such models.

We will first use the following labeling for the layers:

- $\ell = 0$ Input layer

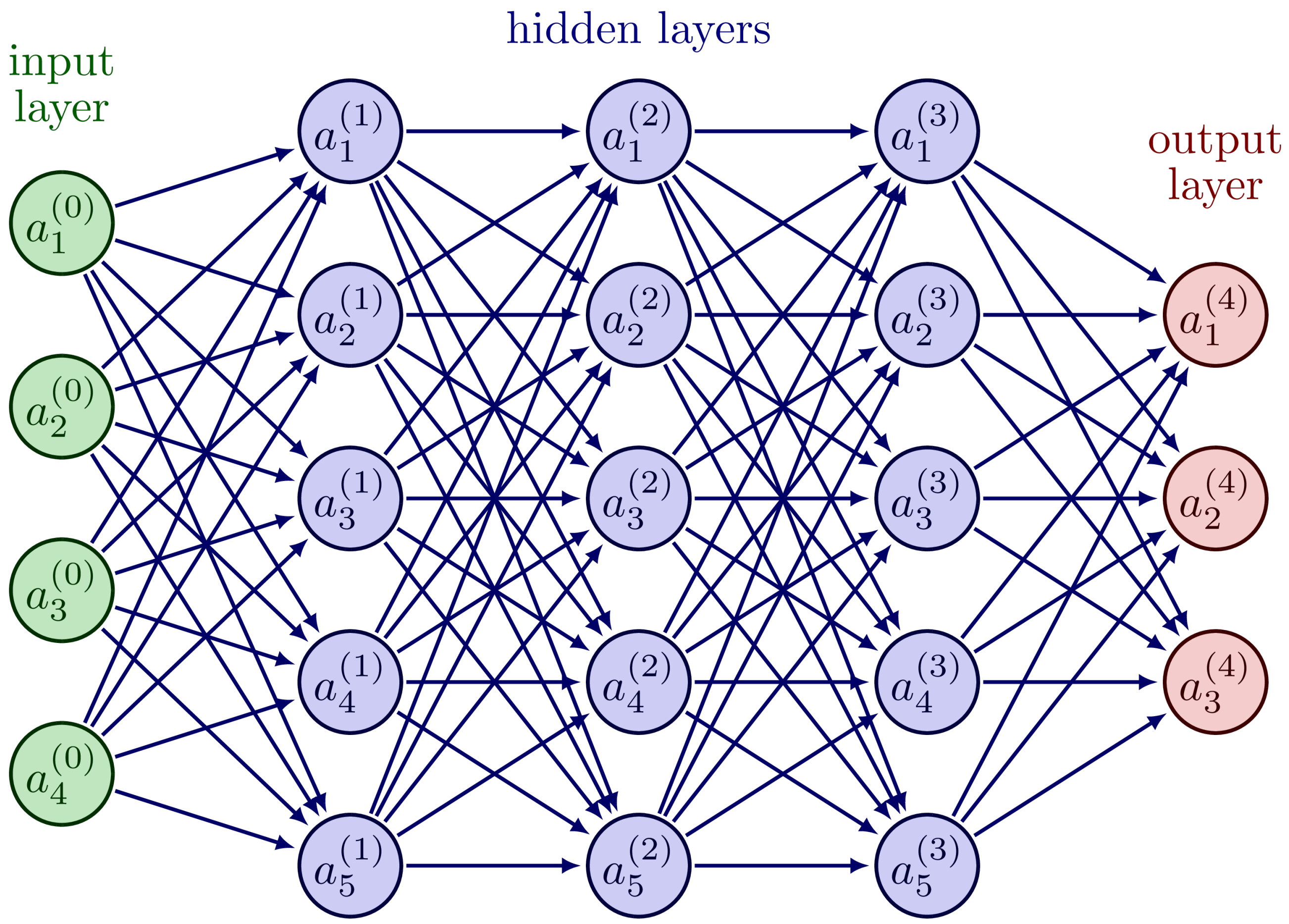

Figure 1: A simple neural network

- $\ell = 1, \cdots, L-1$ Hidden layers
- $\ell = L$ Output layer.
- By d_ℓ will denote the number of nodes in layer ℓ.

We will also use the following convention for the weights:

- $W^{(\ell)} = (w_{ij}^{(\ell)})$ the weights connecting the $\ell - 1$ -layer with the ℓ- layer. In the above $i = 1, \cdots, d_{\ell-1}$ and $j = 1, \cdots, d^\ell$ hence $W^{(\ell)} \in \mathbb{R}^{d_{\ell-1} \times d_\ell}$

In terms of this formulation:

$$\bar{a}_j^{(\ell)} = \sum_{i=1}^{d_{\ell-1}} w_{ij}^{(\ell)} a_i^{(\ell-1)},$$
$$a_j^{(\ell)} = \phi(\bar{a}_j^{(\ell)}), \quad j = 1, \cdots, d_\ell.$$

If we define the vectors

$$\bar{a}^{(\ell)} = (\bar{a}_1^{(\ell)}, \cdots, \bar{a}_{d_\ell}^{(\ell)})^T,$$
$$a^{(\ell)} = (a_1^{(\ell)}, \cdots, a_{d_\ell}^{(\ell)})^T,$$

we see that

$$\bar{a}^{(\ell)} = (W^{(\ell)})^T \bar{a}^{(\ell-1)},$$

We further define the function $\Phi_d : \mathbb{R}^d \to \mathbb{R}^d$ that acts on vectors componentwise as

$$a := (a_1, \cdots, a_d) \mapsto \Phi_d(a) := (\phi(a_1), \cdots, \phi(a_d))$$

In terms of this function we see that

$$a^{(\ell)} = \Phi_{d_\ell}((W^{(\ell)})^T a^{(\ell-1)}). \tag{1}$$

We can express this as

$$a^{(\ell)} = \Phi_{d_\ell} \circ A_{\ell,\ell-1} a^{(\ell-1)}, \tag{2}$$

where $A_{\ell,\ell-1} : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}$ is an affine map. This is true for every ℓ.

3 Forward propagation

Forward propagation is the procedure which takes us from

- the inputs $(x_1, \cdots, x_{d_0}) = (a_1^{(0)}, \cdots, a_{d_0}^{(0)})$
- to the outputs $(y_1, \cdots, y_{d_L}) = (a_1^{(L)}, \cdots, a_{d_L}^{(L)})$.

This can be obtained by performing the steps (1) (equiv. (2)) repeatedly, from layer to layer.

$$\begin{gathered}\text{Input} = x = a^{(0)} \mapsto a^{(1)} = (\Phi_{d_1} \circ A_{1,0})a^{(0)} \mapsto a^{(2)} = (\Phi_{d_2} \circ A_{2,1})a^{(1)} = (\Phi_{d_2} \circ A_{2,1}) \circ (\Phi_{d_1} \circ A_{1,0})a^{(0)} \\ \mapsto \ \cdots \mapsto \cdots \mapsto \ \cdots \mapsto \\ (\Phi_{d_L} \circ A_{d_L,d_{L-1}})a^{(L-1)} = (\Phi_{d_L} \circ A_{d_L,d_{L-1}}) \circ \cdots \circ (\Phi_{d_2} \circ A_{2,1}) \circ (\Phi_{d_1} \circ A_{1,0})a^{(0)} = y = \text{Output}\end{gathered}$$

We will make a distinction in notation between

- $\bar{a}_j^{(\ell)} = \sum_{k=1}^{d_{\ell-1}} w_{kj}^{(\ell)} a_k^{(\ell-1)}$ the **signal into node j, layer ℓ** and
- $a_j^{(\ell)} = \phi(\bar{a}_j^{(\ell)})$ the **output of node j, layer ℓ**

$\bar{a}_j^{(\ell)}$ is the signal that enters node j of the layer ℓ and is then fed into the activation function ϕ to produce the output of the j node of the layer ℓ, which is denoted by $a_j^{(\ell)}$.

Definition 3.1 (Forward propagation)**.** The procedure that takes us from $x \mapsto y$ (as a function of all the parameters – weights W^ℓ, $\ell = 1, \cdots, L$) is called forward propagation. Using the notation $\prod$ for composition of functions the forward propagation procedure can be understood as the action of the function

$$\mathcal{N} := \prod_{\ell=0}^{L} \Phi_{d_\ell} \circ A_{d_\ell,d_{\ell-1}},$$

on the inputs $x = a^{(0)}$.

Definition 3.2 (Neural network)**.** Given a NN structure contained in the vector $\mathbf{L} = (d_0, \cdots, d_L)$, and a collection of weights $\mathbf{W} = (W^{(1)}, \cdots, W^{(L)})$, the function

$$\mathcal{N}_{\mathbf{L},\mathbf{W}} := \prod_{\ell=0}^{L} \Phi_{d_\ell} \circ A_{d_\ell,d_{\ell-1}},$$

is called a Neural Network (NN).

4 Fitting a NN model to a set of data

Definition 4.1 (Neural network model)**.** Given the data $(x, y) \in X \times Y$ a neural network model for these data is a function

$$f(\cdot; \mathbf{L}, \mathbf{W}) := \mathcal{N}_{\mathbf{L},\mathbf{W}} : X \to Y,$$

where $\mathbf{L}$ and $\mathbf{W}$ are considered as parameters of the function, that are to be fitted to the actual data.

Given an NN model corresponding to the (hyper)parameters $\mathbf{L}, \mathbf{W}$ and data (x, y),

$$y_{pred} := f(x; \mathbf{L}, \mathbf{W}),$$

is called the prediction for y, in terms of the NN model.

The training of the model is the procedure of choosing the weights $\mathbf{W}$, so that y_{pred} is as close as possible to the observed data y, as reported by a chosen loss function $L(y_{pred}, y)$. In particular,

$$\begin{aligned} L(\mathbf{W}) := L(y_{pred}, y) = L(f(x; \mathbf{L}, \mathbf{W}), y), \\ \mathbf{W}^* \in \arg\min_{\mathbf{W}} L(\mathbf{W}). \end{aligned}$$

The training of the NN model then reduces to the solution of an optimization problem, the minimization of $L(\mathbf{W})$. This can be effected by a suitable choice of an optimization algorithm, possible choices being:

- Gradient Descent
- Stochastic Gradient Descent
- ADAM

For example if using the gradient descent algorithm, we should calculate the gradient of L with respect to $\mathbf{W}$ (equiv. with respect to each $w_{ij}^{(\ell)}$, $\ell = 1, \cdots, L$, and all relevant i, j) and use the iterative scheme

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \alpha D L_{\mathbf{W}}(\mathbf{W}_k), \ \ \alpha > 0,$$

until some convergence criterion is met. The other algorithms are similar.

5 Backward propagation

Backward propagation is an algorithm that allows us to calculate the gradient $DL_{\mathbf{W}}$ so that we can efficiently implement the optimization scheme for the estimation of the optimal weights. Recall that this gradient provides us with an estimate of the sensitivity of the output of the NN (and consequently the loss function) with respect to the variation of the weights.

The essence of the algorithm is the following:

- $\mathbf{W} = (W^{(1)}, \cdots, W^{(L)})$ so $DL_{\mathbf{W}}$ can be broken into contributions of the form $DL^{W^{(\ell)}}$, $\ell = 1, \cdots, L$, i.e., contributions of the total sensitivity on the weights of each layer separately.

- Recall the special form of the NN function $f_{\mathbf{L},\mathbf{W}}(\cdot) = f(\cdot; \mathbf{W})$ (and consequently of the loss function). Since

$$f(x;\mathbf{W}) = \underbrace{(\Phi_{d_L} \circ \underbrace{A_{d_L,d_{L-1}}}_{(W^{(L)})^T}) \circ \cdots \circ \underbrace{(\Phi_{d_2} \circ \underbrace{A_{2,1}}_{(W^{(2)})^T}) \underbrace{\circ (\Phi_{d_1} \circ \underbrace{A_{1,0}}_{(W^{(1)})^T}) a^{(0)},}_{a^{(1)}}}_{a^{(2)}}}_{a^{(L)}=y_{pred}}$$

$$A_{d_\ell,d_{\ell-1}} = (W^{(\ell)})^T, \quad a^{(0)} = x,$$

each $W^{(\ell)}$ affects only the particular component of this function and not any other component.

Using the distinction between the signal entering each node in each layer (denoted by $\bar{a}^{(\ell)} = (\bar{a}_1^{(\ell)}, \cdots, \bar{a}_{d_\ell}^{(\ell)})^T$) and the output of the corresponding node (denoted by $a^{(\ell)} = (a_1^{(\ell)}, \cdots, a_{d_\ell}^{(\ell)})^T$), we see that

$$\begin{aligned} \bar{a}^{(\ell)} &= A_{\ell,\ell-1} a^{(\ell-1)}, \\ a^{(\ell)} &= \Phi_{d_\ell}(\bar{a}^{(\ell)}). \end{aligned}$$

In particular $W^{(\ell)}$ affects only the intermediate input (signal) $a^{(\ell)}$. Of course this is fed into the other layers $\ell+1, \cdots, L$ and its effect propagates forward.

Hence the error term is a function of the form

$$L(y_{pred}, y) = L(f(x;\mathbf{W}), y) = L(a^{(L)}(\mathbf{W})),$$

where we have omitted the dependence on x, y which are the data and considered as given, and replaced $f(x;\mathbf{W})$ which is the output of the NN by $a^{(L)}$ which of course depends on all weights. We may also consider $a^{(L)}$ as a function of $a^{(L-1)}$, which in turn is a function of $a^{(L-2)}$ etc ...

We work component wise to get that for every $\ell = 0, \cdots, L$ and every i, j,

$$\frac{\partial L}{\partial w_{ij}^{(\ell)}} = \frac{\partial L}{\partial \bar{a}_j^{(\ell)}} \frac{\partial \bar{a}_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} = \frac{\partial L}{\partial \bar{a}_j^{(\ell)}} \frac{\partial}{\partial w_{ij}^{(\ell)}} (\sum_{k=1}^{d_{\ell-1}} w_{kj}^{(\ell)} \phi(\bar{a}_k^{(\ell-1)})) =: \Delta_j^{(\ell)} \phi(\bar{a}_i^{(\ell-1)}), \tag{3}$$

where from now on we will use the convenient notation

$$\Delta_j^{(\ell)} := \frac{\partial L}{\partial \bar{a}_j^{(\ell)}}.$$

In the above, we made use of the fact that $\bar{a}_k^{(\ell-1)}$ only depend on $w_{pq}^{(\ell-1)}$ and this will not contribute to the partial derivatives with respect to $w_{ij}^{(\ell)}$. Recall also that, strictly speaking, L depends directly on $a^{(L)}$, and implicitly on $a^{(L-1)}, a^{(L-2)}, \cdots, a^{(0)}$. This means that using the chain rule we can connect $\Delta^{(L)}$ (which is calculated directly since L depends on $a^{(L)}$ to $\Delta^{(L-1)}$, then to $\Delta^{(L-2)}$, etc until we get to $\Delta^{(0)}$. This is the essence of back propagation.

The general step component wise is

$$\Delta_j^{(\ell-1)} = \frac{\partial L}{\partial \bar{a}_j^{(\ell-1)}} = \sum_{k=1}^{d_\ell} \frac{\partial L}{\partial \bar{a}_k^{(\ell)}} \frac{\partial \bar{a}_k^{(\ell)}}{\partial \bar{a}_j^{(\ell-1)}} = \sum_{k=1}^{d_\ell} \Delta_k^{(\ell)} \frac{\partial \bar{a}_k^{(\ell)}}{\partial \bar{a}_j^{(\ell-1)}}$$

i.e. $\Delta^{(\ell-1)}$ is connected to $\Delta^{(\ell)}$. We must also calculate $\frac{\partial \bar{a}_k^{(\ell)}}{\partial \bar{a}_j^{(\ell-1)}}$. This is

$$\frac{\partial \bar{a}_k^{(\ell)}}{\partial \bar{a}_j^{(\ell-1)}} = \frac{\partial}{\partial \bar{a}_j^{(\ell-1)}} (\sum_{m=1}^{d_{\ell-1}} w_{mk}^{(\ell)} \phi(\bar{a}_m^{(\ell-1)})) = w_{jk}^{(\ell)} \phi'(\bar{a}_j^{(\ell-1)}).$$

Collecting all these together we obtain

$$\Delta_j^{(\ell-1)} = \sum_{k=1}^{d_\ell} w_{jk}^{(\ell)} \phi'(\bar{a}_j^{(\ell-1)}) \Delta_k^{(\ell)} = \phi'(\bar{a}_j^{(\ell-1)}) \sum_{k=1}^{d_\ell} w_{jk}^{(\ell)} \Delta_k^{(\ell)}. \tag{4}$$

On the L-th layer we can calculate $\Delta^{(L)}$ directly in terms of

$$\begin{aligned} \Delta_j^{(L)} &= \frac{\partial L}{\partial \bar{a}_j^{(L)}}(y_{pred}) = \frac{\partial}{\partial \bar{a}_j^{(L)}} L(\underbrace{\Phi_{d_L}(\bar{a}^{(L)})}_{:=y_{pred}}) = \frac{\partial}{\partial \bar{a}_j^{(L)}} L(\underbrace{\phi(\bar{a}_1^{(L)})}_{:=y_{pred,1}}, \cdots, \underbrace{\phi(\bar{a}_{d_L}^{(L)})}_{:=y_{pred,d_L}}) \\ &= \frac{\partial}{\partial a_j^{(L)}} L(\phi(\bar{a}_1^{(L)}), \cdots, \phi(\bar{a}_{d_L}^{(L)})) \frac{\partial a_j^{(L)}}{\partial \bar{a}_j^{(L)}} = \frac{\partial}{\partial a_j^{(L)}} L(\phi(\bar{a}_1^{(L)}), \cdots, \phi(\bar{a}_{d_L}^{(L)})) \phi'(\bar{a}_j^{(L)}) \end{aligned} \tag{5}$$

where we use the interpretation

$$\frac{\partial}{\partial a_j^{(L)}} L(\phi(\bar{a}_1^{(L)}), \cdots, \phi(\bar{a}_{d_L}^{(L)}) = \frac{\partial}{\partial y_{pred,j}} L(y_{pred_1}, \cdots, y_{pred,d_L}). \tag{6}$$

Upon combining (3), (4) and (5) we obtain

$$\begin{cases} \dfrac{\partial L}{\partial w_{ij}^{(\ell)}} = \Delta_j^{(\ell)} \phi(\bar{a}_i^{(\ell-1)}), & \ell = 0, \cdots, L, \\ & i = 1, \cdots, d_{\ell-1}, \\ & j = 1, \cdots, d_\ell, \\ \Delta_j^{(L)} = \dfrac{\partial L}{\partial y_{pred,j}} (\underbrace{\Phi_{d_L}(\bar{a}^{(L)})}_{a^{(L)} = y_{pred}}) \phi'(\bar{a}_j^{(L)}), & 1 \leq j \leq d_L, \\ \Delta_j^{(\ell-1)} = \phi'(\bar{a}_j^{(\ell-1)}) \displaystyle\sum_{k=1}^{d_\ell} w_{jk}^{(\ell)} \Delta_k^{(\ell)}, & 1 \leq j \leq d_{\ell-1}. \end{cases} \tag{7}$$

Algorithm (7) is the famous backpropagation algorithm.

- Matrix formulation of the backpropagation algorithm: To obtain a matrix formulation of the backpropagation algorithm we require the definition of a new type of product between vectors called the Hadamard product. This is defined as $\odot : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ which when acting between two vectors $a = (a_1, \cdots, a_d)$, $b = (b_1, \cdots, b_d)$ provides a new vector $a \odot b = (a_1 b_1, \cdots, a_d b_d)$.

  In terms of the Hadamard product we can define the vectors

$$\begin{aligned} \Delta^{(\ell)} &= (\Delta_1^{(\ell)}, \cdots, \Delta_{d_\ell}^{(\ell)})^T, \\ \Psi^{(\ell)} &= (\Psi_1^{(\ell)}, \cdots, \Psi_{d_\ell}^{(\ell)})^T = (\phi'(\bar{a}_1^{(\ell)}), \cdots, \phi'(\bar{a}_{d_\ell}^{(\ell)}))^T \end{aligned}$$

  and write

$$\Delta^{(\ell-1)} = \Psi^{(\ell)} \odot (W^{(\ell)} \Delta^{(\ell)})$$

Moreover, upon defining

$$D_{a^{(L)}} L = (\frac{\partial L}{\partial a_1^{(L)}}, \cdots, \frac{\partial L}{\partial a_{d_L}^{(L)}})^T,$$

or equivalently

$$D_{y_{pred}} L = (\frac{\partial L}{\partial y_{pred,1}}, \cdots, \frac{\partial L}{\partial y_{pred,d_L}})^T,$$

the gradient of the loss function with respect to the output of the NN, $a^{(L)} = y_{pred}$, we can express the equation for Δ^L in terms of

$$\Delta^{(L)} = D_{a^{(L)}} L \odot \Psi^{(L)}.$$

Finally, we may collect all the partial derivatives $\frac{\partial L}{w_{ij}^{\ell}}$ for all $i = 1, \cdots, d_{\ell-1}$, $j = 1, \cdots, d_\ell$, into a matrix denoted by

$$\frac{\partial L}{\partial W^{(\ell)}} = \left(\frac{\partial L}{\partial w_{ij}^{(\ell)}}\right)_{i=1,\cdots d_{\ell-1}}^{j=1,\cdots,d_\ell} = \left(\Delta_j^{(\ell)} \phi(\bar{a}_i^{(\ell-1)})\right)_{i=1,\cdots d_{\ell-1}}^{j=1,\cdots,d_\ell} \in \mathbb{R}^{d_{\ell-1}\times d_\ell}$$

In terms of the vector and matrix notation introduced here we can express

$$\frac{\partial L}{\partial W^{(\ell)}} = \Phi_{d_{\ell-1}}(\bar{a}^{(\ell-1)})(\Delta^{(\ell)})^T \in \mathbb{R}^{d_{\ell-1}\times d_\ell}$$

We can collect all the above for the matrix form of the backpropagation algorithm, as

$$\begin{aligned} \Delta^{(L)} &= D_{a^{(L)}} L \odot \Psi(\bar{a}^{(L)}) \\ \Delta^{(\ell-1)} &= \Psi^{(\ell)} \odot (W^{(\ell)}\Delta^{(\ell)}), \;\; \ell = L, \cdots, 1, \\ \frac{\partial L}{\partial W^{(\ell)}} &= \Phi_{d_{\ell-1}}(\bar{a}^{(\ell-1)})(\Delta^{(\ell)})^T, \;\; \ell = L, \cdots, 1. \end{aligned} \tag{8}$$

The above algorithm is a backward iterative algorithm: Starting from $\Delta^{(L)}$ we work backwards and obtain the remaining $\Delta^{(\ell)}$, $\ell = L, \cdots, 1$, in terms of the iterative scheme $\Delta^{(\ell-1)} = \Psi^{(\ell)} \odot (W^{(\ell)}\Delta^{(\ell)})$.

The exact form of the loss function is essentially needed only at the last step (here initial since we are thinking of it in terms of the backward induction scheme). We provide two classic examples

- Quadratic loss function: Here we choose the loss function

$$L(y_{pred}, y) = \frac{1}{2N}\sum_{k=1}^{N} \|y_{pred}^{(k)} - y^{(k)}\|_{\mathbb{R}^{d_L}}^2 = \frac{1}{N}\sum_{k=1}^{N} L^{(k)},$$
$$L^{(k)}(y_{pred}^{(k)}, y^{(k)}) = \frac{1}{2}\|y_{pred}^{(k)} - y^{(k)}\|_{\mathbb{R}^{d_L}}^2.$$

  Our notation is as follows: Our data are expressed as $\{(x^{(k)}, y^{(k)}),\;\; k = 1, \cdots, N\}$ where $x^{(k)} \in \mathbb{R}^{d_0}$ and $y^{(k)} \in \mathbb{R}^{d_L}$ for each $k = 1, \cdots, N$, denoting the features vector (d_0 features) and the response vector (d_L responses) for each datum. Then for each $x^{(k)} \in \mathbb{R}^{d_0}$, considered as input to the NN we obtain the corresponding output denoted by $y_{pred}^{(k)}$. The loss for each observation k is to be calculated as the Euclidean norm (in $\mathbb{R}^{d_L}$) of the error $y_{pred}^{(k)} - y^{(k)}$ between the prediction of the NN for the features vector (input) $x^{(k)}$ and the actual observation, corresponding to this datum, $y^{(k)}$.

  For each datum point (i.e. for each fixed k) we see that the corresponding $L^{(k)}$ is quadratic and (dropping the superscript (k) for notational convenience)

$$D_{y_{pred}^{(k)}} L^{(k)} = (y_{pred}^{(k)} - y^{(k)}).$$

- The cross entropy function: This is a good choice for classification problems to two classes 0 and 1. This function is of the form

$$-\sum_{k=1}^{N} y^{(k)} \ln y_{pred}^{(k)} - \sum_{k=1}^{N} (1 - y^{(k)}) \ln(1 - y_{pred}^{(k)}) = \sum_{k=1}^{N} L_k(y_{pred}^{(k)}, y^{(k)})$$

  The output here is one dimensional $d^L = 1$, for each datum $k = 1, \cdots, N$. We can then calculate

$$\frac{\partial}{\partial y_{pred}^{(k)}} L^{(k)}(y_{pred}^{(k)}, y^{(k)}) = \frac{1}{y_{pred}^{(k)}(1 - y_{pred}^{(k)})}(y_{pred}^{(k)} - y^{(k)}).$$

  In the special case where ϕ is the logistic function $\Psi(y_{pred}^{(k)}) = y_{pred}^{(k)}(1 - y_{pred}^{(k)})$ and the corresponding $\Delta^{(L)}$ simplifies considerably.

6 Why do neural networks work?

The reason why NN work and allow us to represent data is a deep result in mathematical analysis, related to the density of the set of functions that consists of compositions of affine functions and activation functions in general classes of functions such as for example the class of continuous functions. Such results are collected under the general framework of Universal Approximation Theorems (UAT). This is an important part of the mathematics of machine learning, as it "legitimizes" their use and makes them a credible tool for the representation of data.

There are many different versions of UAT and the list is ever expanding to meet the needs of modern trends and applications of deep learning. The typical form of such a theorem is

- Set a network architecture
- Set an activation function
- Show that a general class of functions (a function space) can be reconstructed in terms of linear combinations and limits of linear combinations of NNs of the architecture and activation functions as above (density of the set of NNs in the chosen function space).

We will need some definitions.

Definition 6.1 (Density)**.** Let X be a function space, considered as a metric space (X, d). The set $Y \subset X$ is dense in X if for every $\epsilon > 0$ and every $x \in X$, there exists $y_\epsilon \in Y$ such that $d(x, y_\epsilon) < \epsilon$. Equiv. for every $x \in X$, there exists a sequence $(y_n)_{n\in\mathbb{N}} \subset Y$ such that $y_n \to x$ in (X, d) (i.e. $d(y_n, x) \to 0$ as $n \to \infty$).

In other words, Y is dense in X if any element of X can be approximated as close as possible with an element from Y.

The way that density theorems are used in UAT, is to set X to be a general function space (for example $X = C(\mathbb{R}^{d_0}; \mathbb{R}^{d_L})$, the space of continuous functions from the set of inputs to the set of outputs) and $Y = \mathcal{NN}_{\mathbf{D}}$, the set of NN functions of the form as in Definition 4.1 for any set of weights $\mathbf{W}$.

Here we give a sample of this theory, by presenting an important UAT, the Cybenko theorem (1989) (see Cybenko (1989)). It covers, the case of shallow networks (i.e. networks with small number of hidden layers). The version we present is for a single layer NN.

Theorem 6.2 (Cybenko). *Let $\sigma : \mathbb{R} \to [0,1]$ be a continuous sigmoidal function, i.e. a function such that*

$$\lim_{x\to-\infty} \sigma(x) = 0, \quad \lim_{x\to\infty} \sigma(x) = 1.$$

Then, the set

$$Y = \{\sum_{j=1}^{n} a_j \sigma(w_j^T x + \theta_j), \ | \ n \in \mathbb{N}, \ w_j \in \mathbb{R}^{d_0}, \ a_j, \theta_j \in \mathbb{R}\}$$

is dense in $X = C([0,1]^{d_0}; \mathbb{R})$.

The set Y can be considered as the set of NNs consisting of a single hidden layer (with n neurons) with d_0 inputs, denoted as x. The vectors $w_j \in \mathbb{R}^{d_0}$ are the vectors of weights used for each neuron, whose signals are then introduced into a sigmoidal activation function σ. These are then composed in terms of the a_j into the output $y \in \mathbb{R}$ (so here $d_L = 1$). The $\theta_j \in \mathbb{R}$ are considered as biases for each neuron signal (they can be included in x by adding an extra component 1 to the x vector).

The result of Cybenko shows that if we have no restriction on the number of modes of the single hidden layer (or the weights or other parameters) we can approximate any continuous function using such a network, as long as we use the proper type of activation function, which has to be a continuous sigmoidal activation function.

By changing the structure of the NN we can drop the condition of the activation function being sigmoidal. An example of such a case is by using the so called $\Sigma\Pi$ networks, which is covered by the UAT of Hornick, Stinchcombe and White (1989) (see Hornik et al. (1989)).

Theorem 6.3 (Hornick, Stinchcombe and White (1989)). *Let $\phi : \mathbb{R} \to \mathbb{R}$ be a continuous, non constant activation function. Then the set*

$$Y = \{f(x) = \sum_{k=1}^{M} \beta_k \prod_{j=1}^{N_k} \phi(w_{jk}^T x + \theta_{jk}), \ | \ w_{jk} \in R^{d_0}, \theta_{jk}, \beta_k \in \mathbb{R}, M, N_k \in \mathbb{N}\}$$

is dense in $X = C([0,1]^{d_0}; \mathbb{R})$.

The set Y consists of a class of NNs with a different architecture (or structure) which are called $\Sigma\Pi$-networks.

References

Bengio, Y., I. Goodfellow, and A. Courville (2017). *Deep learning*, Volume 1. MIT press Cambridge, MA, USA.

Calin, O. (2020). *Deep learning architectures.* Springer.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems 2*(4), 303–314.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators. *Neural networks 2*(5), 359–366.