On recurrent neural networks

Giulio Galvan

21st March 2015

Contents

Chapter 1

Artificial neural networks

1.1 A family of models

An artificial neural network is a network of connected units called neurons or perceptrons, each arc which connects two neurons i and j is associated with a weight wji. Perceptrons share the some structure for all models, what really define the model in the family is how the perceptrons are arrange and how are they conneceted, for example if there are cycles or not. As you can see in figure 1.1 each neuron is *fed* with a set inputs which are the weighted outputs of other neurons. Formally the output of a perceptron ϕ_j is defined as:

$$\phi_j \triangleq \sigma(a_j) \tag{1.1}$$

$$a_j \triangleq \sum_l w_{jl}\phi_l \tag{1.2}$$

where w_{jl} is the weight of the connection between neuron l and neuron j, $\sigma(\cdot)$ is a non linear function and $b_j \in \mathbb{R}$ is called bias.

A neural network looks like: INSERT FIGURE

It sometime useful to think of a neurual network as series of layers, one on top of each other, as depicted in figure **??**. The first layer is called the input layer and its units are *fed* with external inputs, the upper layers are called *hidden layers* because their's outputs are not observed from outside except the last one which is called *output layer* because it's output is the output of the net.

Whene we describe a network in this way is also useful to adopt a different notation: we describe the weights of the net with a set of matrixes W_i one for each layer, and neurons are no more linearly indexed, insted with refer to a neuron with a relative index with respect to the layer; this allows to write easier equations in matrix notation.

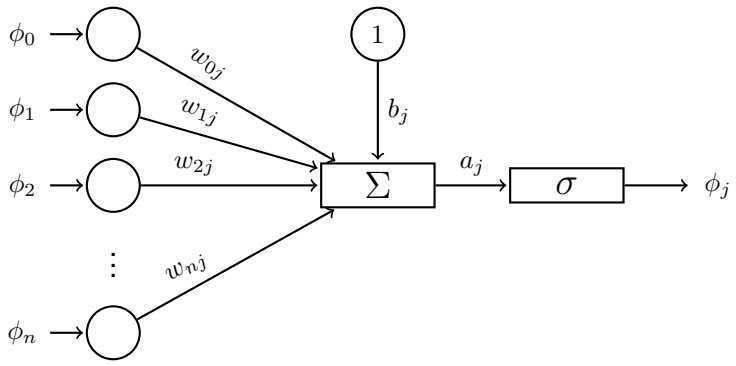

Figure 1.1: Neuron model

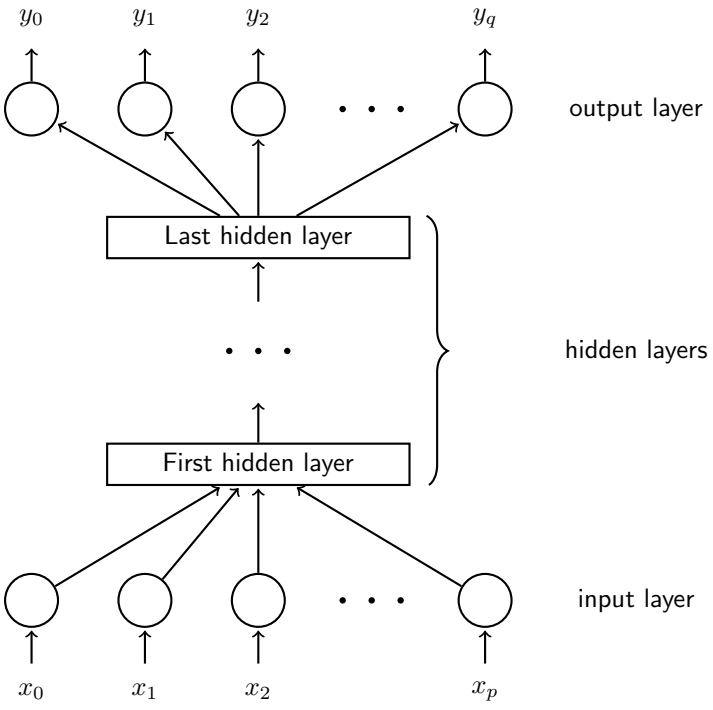

Figure 1.2: Feed forward neural network model

1.2 Feed foward neural networks

A feed foward neural network is an artificial neural network in which there are no cycles, that is to say each layer output is *fed* to the next one and connections to any earlier layer are not possible.

Definition 1. Feed foward neural networks
A feed foward neural network is tuple

$$FFNN \triangleq < \boldsymbol{p}, W, \boldsymbol{b}, \sigma(\cdot), f(\cdot) >$$

- $\boldsymbol{p} \in \mathbb{N}^U, p(k) =$ number of neuron of layer k, where U is the number of layers
- $W \triangleq \{W^k_{p(k+1)\times p(k)}, k = 1, ..., U\}$ is the set of weight matrixes of each layer
- $b \in \mathbb{R}^U$ is the bias vector
- $\sigma(\cdot) : \mathbb{R} \to \mathbb{R}$ is the activation function
- $f(\cdot) : \mathbb{R}^{p(U)} \to \mathbb{R}^{p(U)}$ is the output function

Given a $FFNN$ and an input vector $\boldsymbol{x}$ the output $\boldsymbol{y}$ of the net is defined by the following:

$$\boldsymbol{y} = f(\boldsymbol{\phi}_U) \tag{1.3}$$
$$\boldsymbol{\phi}_i \triangleq \sigma(\boldsymbol{a}_i), \qquad i = 1, ..., U \tag{1.4}$$
$$\boldsymbol{a}_i \triangleq W_{i+1} \cdot \boldsymbol{\phi}_{i-1} + \boldsymbol{b}_i \qquad i = 1, ..., U \tag{1.5}$$
$$\boldsymbol{\phi}_0 \triangleq \boldsymbol{x} \tag{1.6}$$

1.2.1 On expressivness of ffnn

1.2.2 Learning with ffnn

As we have seen in the previous section a model sush as $FFNN$ can approximate arbitrary well any smooth function, so a natural application of feed foward neural networks is machine learning. To model an optimization problem we first need to define a dataset D as

$$D \triangleq \{x^{(i)} \in \mathbb{R}^p, y^{(i)} \in \mathbb{R}^q, i \in [1, N]\} \tag{1.7}$$

Then we need a loss function $L_D : \mathbb{R}^U \to \mathbb{R}_{\geq 0}$ over D defined as

$$L_D(W) \triangleq \frac{1}{N} \sum_{i=1}^{N} L_i(W) \tag{1.8}$$

L_i is an arbitrary loss function for the i^{th} example.

The problem is then fo find a $FFNN$ which minimize L. As we have seen feed foward neural network allow for large customization: the only variables in the optimization problem are the weights, the other parameters are said *hyper-parameters* and are determined *a priori*. Usually the output function is choosen depending on the output, for instance for multi-way classification is generally used the softmax function, for regression a simple identity function.

The activation function $\sigma(\cdot)$ is often choose from:

$$sigmoid(x) = \frac{1}{1 - e^{-x}} \tag{1.9}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{1.10}$$

$$ReLU(x) = \begin{cases} x, & \text{if } x > 0. \\ 0, & \text{otherwise.} \end{cases} \tag{1.11}$$

For what concerns the number of layers and the number of units per layers they are choosen relying on experience or performing some kind of hyper-parameter tuning, which usually consists on training nets with some different configurations of such parameters and choosing the best one.

Once we have selected the values for all hyper-paramters the optimization problem becomes:

$$\min_{W} \quad L_D(W) \tag{1.12}$$

1.2.3 Gradient

We can compute partial derivatives with respect to a single weight w_{lj}, using simply the chain rule, as

$$\frac{\partial L}{\partial w_{lj}} = \frac{\partial L}{\partial a_l} \cdot \frac{\partial a_l}{\partial w_{lj}} = \delta_l \cdot \phi_j$$

where we put

$$\delta_l \triangleq \frac{\partial L}{\partial a_l} \tag{1.13}$$

So we can easily compute $\delta_u = \frac{\partial L^{(i)}}{\partial a_u}$ for each output unit u once we choose a differentiable loss function; note that we don't need the weights for such a computation.

Let $P(l)$ be the set of parents of neuron l, formally:

$$P(l) = \{k : \exists \text{ a link between } l \text{ and } k \text{ with weight } w_{lk}\} \tag{1.14}$$

Again, simply using the chain rule, we can write, for each non output unit l:

$$\delta_l = \sum_{k \in P(l)} \frac{\partial L^{(i)}}{\partial a_k} \cdot \frac{\partial a_k}{\partial a_l} = \sum_{k \in P(l)} \delta_k \cdot \frac{\partial a_k}{\partial \phi_l} \cdot \frac{\partial \phi_l}{\partial a_l} = \sum_{k \in P(l)} \delta_k \cdot w_{kl} \cdot \sigma'(a_l) \tag{1.15}$$

For output units instead we can compute $\delta_u = \frac{\partial L^{(i)}}{\partial a_u}$ directly once we define the loss function.

In the following we rewrite the previously derived equations in matrix notation. Let us recall that the weight matrix for the i^{th} layer is the $p(i) \times p(i-1)$ matrix whose elements $w_{l,k}$ are the weights of the arcs which link neuron k from level $i-1$ to neuron l from level i, where $p(i)$ is the number of neuron layer i is composed of.

We can rewrite equation 1.15 in matrix notation as:

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial \boldsymbol{a}_i} \cdot \frac{\partial \boldsymbol{a}_i}{\partial W_i}^T = \Delta_i \cdot \boldsymbol{\phi}_{i-1}^T \tag{1.16}$$

where

$$\Delta_i \triangleq \frac{\partial L}{\partial \boldsymbol{a}_i} \tag{1.17}$$

$$\Delta_i = W_{i+1}^T \cdot \Delta_{i+1} \circ \sigma(\Delta_i) \tag{1.18}$$

1.3 Recursive neural networks

Bibliography