Optimization of a Neural Network

Jordi Castro

Neural networks are a type of functions, or mathematical tools, that have enjoyed a great acceptance to solve several types of problems. Two reasons which explain this success have been that they are easy to use and the fact that they allow to solve in an acceptable way some kind of problems where other techniques required a great amount of time or were not able to find an enough accurate solution. When using a neural network, the first thing to do is to adjust some parameters (called weights) which will determine how it will operate. The objective of this assignment will be finding these parameters (network calibration) using optimization techniques. First, we will introduce a simple neural network, and next we will describe the optimization problem to be solved, using the AMPL package.

1 Neural networks behavior

A neural network is just a mathematical function $F : \mathbb{R}^{n_e} \rightarrow \mathbb{R}^{n_o}$ which receives an input vector $x_e \in \mathbb{R}^{n_e}$ and generates an output vector $x_o \in \mathbb{R}^{n_o}$. The input vector corresponds to a set of known values. The neural network is evaluated with these values, and it computes a particular output result (made up of one or more values, depending of the dimension n_o of the output vectors x_o). To understand how a neural network works, we will consider a very simple one, as shown in the Figure 1. The neural network consists of a set of nodes (called "neurons" in the neural networks jargon), which transmit information in the direction indicated by the arcs. These nodes are organized in layers. In the particular case shown by the Figure 1, we have seven nodes organized in three layers. In this particular network, the lower layer (with four nodes) receives four input values ($x_i,\ i = 1, ..., 4$), while the output layer, with only one neuron, gives us the result of the network. So, in this case, we have that $n_e = 4$ and $n_o = 1$.

The network works in the following way. Each node i receives an "input" I_i, and transforms it into an "output" O_i. This transformation is done through some particular function $f(x)$. Two of the most used functions are

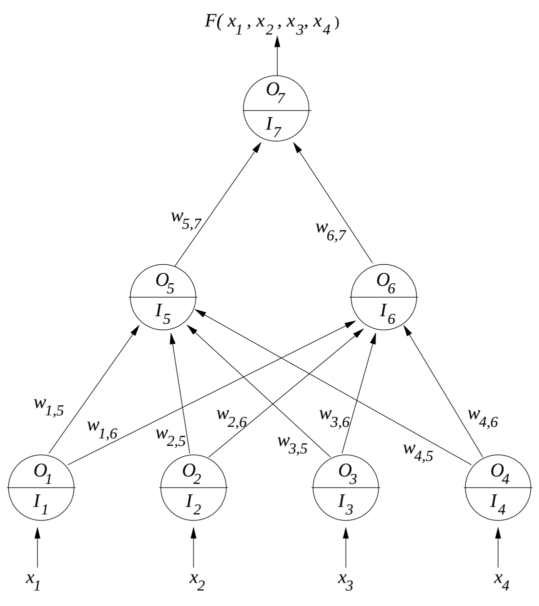

Figure 1: Simple neural network

the hyperbolic tangent function $tgh(x) = (e^x - e^{-x})/(e^x + e^{-x})$, and the sigmoid function

$$f(x) = \frac{1}{1 + e^{-x}},$$

which has the plot shown in Figure 2. We can observe that the sigmoid function always return a value between 0 and 1 (the hyperbolic tangent returns values between -1 and 1). In this assignment we will use the sigmoid function. So, the values O_i of a neuron's output will be calculated as $O_i = f(I_i)$, where f will be the sigmoid function. The output O_7 of the last neuron gives us the result of the neural network.

Let us see now how we can calculate the inputs I_i of each neuron i. They will be obtained adding up the outputs of the neurons j which send information to the neuron i, after being multiplied by the values $w_{j,i}$, shown in Figure 1. These values $w_{j,i}$, named "weights", are used to weight the information that arrives at each neuron. For example, at the node 5 of the Figure 1, the input' I_5 will be calculated as:

$$I_5 = w_{1,5}O_1 + w_{2,5}O_2 + w_{3,5}O_3 + w_{4,5}O_4.$$

A similar computation will be done for the other nodes. This way, for some input values x_1, x_2, x_3 i x_4 to the neural network, we will get the following

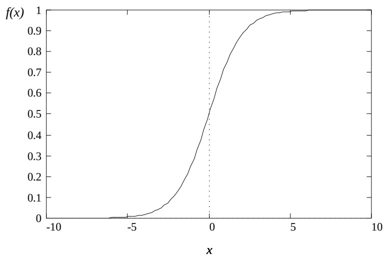

Figure 2: Plot of the sigmoid function $f(x) = 1/(1+e^{-x})$

result:

$$\begin{aligned} F(x_1,x_2,x_3,x_4) &= O_7 = f(I_7) = f(w_{5,7}O_5 + w_{6,7}O_6) = f(w_{5,7}f(I_5) + w_{6,7}f(I_6)) = \\ &= f(w_{5,7}f(\textstyle\sum_{i=1}^4 w_{i,5}O_i) + w_{6,7}f(\sum_{i=1}^4 w_{i,6}O_i)) = \\ &= f(w_{5,7}f(\textstyle\sum_{i=1}^4 w_{i,5}f(x_i)) + w_{6,7}f(\sum_{i=1}^4 w_{i,6}f(x_i))), \end{aligned} \tag{1}$$

being $f()$ the sigmoid function, as shown previously. In this simple case, we have been able to write in detail the function which represents the neural network using the equation (1). If we had a larger amount of nodes and layers, this would not be possible, due to the multiple recursive levels that would appear. However, in order to perform this assignment, this very simple neural network is enough.

It is worth to mention that in neural networks used in real life, apart from the weights w shown by the Figure 1, each node has an additional weight which is always added directly to the input of the network (so, without multiplying it by any "output" of a node of a lower layer). These weights are called "bias" using the neural networks terminology. In this assignment, in order to reduce the number of variables of the problem, we will not consider these additional weights.

Therefore, we see that the behavior of the neural network is governed by the values of the weights w. Depending on the particular values of w, the network will produce different results. Now we should ask: how can we obtain the right weights w?. We need two things: firstly, some data to guarantee the network "learns" the behavior it should have (we will see this clearly later); and secondly, a numerical method to perform this "learning" (the concept of "learning" is also part of the neural networks terminology). Once we have calibrated the network (i.e., we have adjusted the values w), it is ready to give some response, using a vector x_e as the input data.

2 Obtaining the weights w

So, we still have to see how to determine the weights w. There are different ways to do it. But in this assignment we will use one based on nonlinear least-squares (unconstrained minimization problem), where the variables to optimize will be the weights w. To obtain these weights we must have p vectors of input data $x_e \in \mathbb{R}^{n_e}$, and p vectors of output data $x_o \in \mathbb{R}^{n_o}$, which must correspond to the values associated to the input data. The idea is to adjust the weights w of the network in a way that $F(x_{e_i}; w) \approx x_{o_i},\ i = 1, ..., p$, where $F(x; w)$ represents the response of the network when its inputs is the vector x, and it has the weights w. We could say that the p vectors $x_{e_i}, x_{o_i},\ i = 1, ..., p$ are a sample that allows the network "to learn" which is the response for a given input. So, a good way to adjust the w will be to set out a nonlinear least-squares problem, where we minimize the distance between $F(x_{e_i}; w)$ and x_{o_i}. The problem would be:

$$\min_w \sum_{i=1}^{p} ||F(x_{e_i}; w) - x_{o_i}||^2 \tag{2}$$

In the simplest case, where the output of the network has a unique value, we have that $n_o = 1$ (this is the situation that we will consider in this assignment), and the problem to solve in this case can be rewritten as:

$$\min_w \sum_{i=1}^{p} (F(x_{e_i}; w) - x_{o_i})^2 \tag{3}$$

3 Application fields of neural networks

Neural networks are used on different fields. One of them is forecasting. In this case, input data could correspond to n values of a given time series (as, for example, the number of sunny hours of a country, temperature of some compound at each hour, number of annual births of a region, diary consumption of electrical energy —in Kwh— of a given city, etc.), and the result of a neural network would be an estimation about the data corresponding to the next interval.

Another application field is for shape and pattern recognition. For example, let us consider that we can classify a set of objects (or situations) according to their characteristics, and we have a determined number of classes. Then, for an object, we will measure their characteristics, and these measures will be the input vector to the neural network. The output of the network will be one value that tell us to which class the object belongs to, as a function of the input data. For instance, we can think that the measures correspond to some values of clinical tests of a patient, and the output of the network (fed by the previous data) will indicate us which kind of illness does the patient has (so, we have "classified" the patient). One of the advantages (sometimes disadvantages) of neural networks, when they are used for classification, is that they do not require a previous knowledge about the measured data, or existing data. In the example of the classification of a patient according to the clinical data, this means that the neural network does not know if the first measure corresponds to the blood pressure of the patient, or to her cardiac frequency, for example. For these reasons, the application of neural networks is usually simple: we just need to give some data in order to induce the learning process, without thinking about the meaning of this data; the network does not require this additional information. On the other hand, it has the disadvantage that it is difficult to explain the resulting classification as a function of the input data, which would be really useful in some cases (as, for example, the previous case of the illnesses, where we could become aware of certain pathologies). In this assignment, as we will see now, we will work with a very simple case of pattern recognition.

4 Problem to be solved

As previously said, in this assignment we will consider a very simple case of pattern recognition. The vector x_e of input data will have four components $x_i,\ i = 1, ..., 4$. The output would be a unique value (so, $n_o = 1$), denoted by y (so, $x_o = y \in \mathbb{R}$).

Normally, when we use a neural network, we do not know the relation between the input and output data (if we knew it, the neural network would not be needed). But in this case, we will know the relation between input and output, allowing us to check later if the network has done a good adjustment for the weights w by testing its behavior with random input vectors. This input-output relation will be:

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{4} x_i > 2 \\ 0 & \text{if } \sum_{i=1}^{4} x_i \leq 2 \end{cases} \tag{4}$$

So, if the result of adding the four input values is greater than 2, the output will be 1, otherwise it will be 0. It is worth to mention that all the input values $x_i,\ i = 1, ..., 4$ have to be set between 0 and 1. This is a practical requirement of neural networks, in order to guarantee its correct behavior.

In order to perform the "learning" of the neural network, we must have a total of p input vectors with theirs corresponding outputs. In order to generate these values you can use an automatic generator of data in function of the relation (4). To obtain the data for your problem you have to run the provided code `genxndat`, with the command:

```
genxndat file p seed
```

The first parameter, `file`, will be the filename where data will be written. The second parameter, `p`, represents the number of input and output data that will be written into the file. In theory, the learning will be better as p increases. You can use a value of $p = 50$, which will not generate a very large set of data, but it will be enough to guarantee an acceptable learning. The last parameter is a seed (integer value) for the random number generator. You can introduce, for example, your identity card number or your birth-date. So, if you type `genxndat data.dat 6 654321`, you will get a file `data.dat` similar to:

```
.222 0.828 0.061 0.505   0.0
.097 0.676 0.121 0.742   0.0
.859 0.735 0.848 0.530   1.0
.565 0.150 0.625 0.139   0.0
.067 0.428 0.329 0.808   0.0
.832 0.776 0.538 0.210   1.0
```

Each row corresponds to some determined input/output data (we have $p = 6$). In each row, the first four columns correspond to the input values $x_i,\ i = 1, ..., 4$, and the last column indicates the output value y. It is possible to observe how data preserves the relation indicated in (4).

The network that we will consider will be like the one of the Figure 1: a network of three layers and seven nodes (four in the first layer, two in the second one and one node in the last layer). The minimization problem to be solved will be the one in (3), using the expression of the neural network in (1). The variables of the nonlinear least square problem will be the weights w. Observing the Figure 1, we can see that the number of variables to optimize is 10. The resulting unconstrained minimization problem will be solved with the modelling language AMPL using some available solver.

Once the "learning" has been performed, it is necessary to verify that the neural network works properly. This will be done by feeding the network with some input vectors, and checking that the output result matches the rule (4). If there are some input vectors with an incorrect solution, you should check whether they correspond to cases where $\sum_{i=1}^{4} x_i \approx 2$.

5 Some important details!

There are two things that have to be considered when trying to solve this problem with AMPL and some nonlinear solver. The first one is that the weights w do not have any sign constraint (they can be either negative or positive). So, they are free variables. However, if they take very negative values, the evaluation of the sigmoid function will involve $e^x, x \gg 0$, which may result in a numerical error (overflow). Therefore you may be forced to impose a fictitious lower bound on w.

The second thing to take into account refers to the starting point of the optimization process. The solver that you will use with AMPL (like in many others optimization packages) performs an iterative process, and at each iteration it finds a new point w^k. The initial point w^0, unless otherwise stated, is initialized by the solver to some determined value. The problem of this assignment (characterized by the expression (3)), is highly nonlinear, and it has a lot of local minima (due to the sigmoid function $f(x)$). If we allow the solver to use its default initial point w^0, we can easily go to a local minimum where the "learning" of the network is null. To avoid this problem, the initial point is often initialized with random values to avoid going always to the same unsatisfactory local minimum. In this particular case, you may try to provide some initial point like $w^0_{i,j} = 1\ \forall_{i,j}$, but for $w_{6,7}$, which will be initialized to $w^0_{6,7} = -1$. If this initial point does not provide a good local minimum, try to introduce some other values randomly. You can observe if the obtained local minimum is a good minimum with the final value of objective function: it should be close to 0 if the "learning" has been satisfactory (a value of 0 means that there were few errors when solving the nonlinear least squares problem). In AMPL, to set *xinit* as initial value of variable x you may type:

```
var x := xinit;
```

If this doesn't work, then try to move to a different solver. Remind you have many of them available at the NEOS server: `http://www-neos.mcs.anl.gov/`.

6 Presentation of the assignment

You have to provide a report with the following sections:

1. A cover page with your name.

2. The file with the AMPL model for this problem.
3. The result provided by AMPL, reporting the value of the objective function and the obtained weights w. Clearly state which solver you used.
4. The results evaluating the network with some input vectors, checking the good (or bad) behavior of the network.
5. Any other observation or comment you may want to add, or any problem you had when performing the assignment.