NN User Manual

Version 1

Francisco J. Sánchez

September 2020

Contents

About this manual

In the examples and explanations of code in this manual, the ellipsis indicated by “...” will be used to indicate that something is missing in that part for the program to be valid, but that it does not affect the explanation of the code fragment it is explaining.

This manual does not explain what a neural networks is or how it works.

1 Program structure

A NN program is a neural network and instructions give input values to the networks, select outputs and, based on all of the above, calculate the results. In principle, it is stored in a text file.

The order and position of the networks in the program does not matter, they are global in scope. The instructions are executed in the order that they appear.

This manual attempts to format the code so that it is easily readable and understandable, but any indents, spacing, or line breaks are optional and have no effect on the program.

2 Elements

The elements are networks, layers, and neurons. Combining them you can create neural networks.

Each element has a unique name and therefore cannot be repeated throughout the program. The names will be a set of letters from ‘a’ to ‘z’ (without accents or ‘ñ’) and numbers with the first character being a capital letter.

2.1 Networks

Networks connect layers or other networks one after another, in the order in which they are defined within. They can be defined directly in the program, within another network or within a layer. They connect layers or networks, you cannot mix. They must contain a minimum of two elements.

The way to define a network is with its name followed by the definitions of the elements it connects in parentheses separated by commas.

The outputs of the network will be the outputs of the last element defined in it.

```
R (
  Element1 ...,
  Element2 ...,
  ...
  ElementN ...
)
```

In this network, "Element1" will be executed, its outputs are connected to "Element2" (as explained in 2.3) which is executed afterwards and continues in this way, in the order that they are defined, until "ElementN". In this example, the network outputs are the outputs of "ElementN".

2.2 Layers

Layers connect networks or neurons within other networks. They have to be defined within a network. There cannot be connections between the elements of a layer themselves, these must be with the elements of the previous layer (as inputs) or the next layer (as outputs). They must contain a minimum of one element.

The way to define a layer is with its name followed by the definitions of the elements it contains between brackets separated by commas.

The outputs of the layer will be the union of all the outputs of the elements defined in it.

```
...
C [
  Element1 ...,
  Element2 ...,
  ...
  ElementN ...
]
...
```

In this layer, all the elements will be executed, from "Element1" to "ElementN", in a "parallel" way [1]. The set of outputs of the layer is the union of all the outputs of "Element1", "Element2", up to "ElementN".

2.3 Neurons

Neurons are the ones that perform the calculations in the neural network. They have to be defined within a layer. They consist of one or more inputs, one or more outputs (in the sense of where the resulting value goes), a value from *bias*, and a trigger function.

The way to define a neuron is with its name followed by the properties between braces separated by commas. Properties are defined with the property name (always lowercase), a colon, and the property value. The order in which the properties are listed does not affect.

As has been said there are four properties, which are:

[1] The execution is technically sequence one after another in the order indicated, but since the outputs cannot have connections with none of the inputs of the layer, in a practical way for the user, it is as if it were executed in parallel.

- `input`: Required. Define the inputs of the neuron. Its value with connections between brackets separated by commas, at least one. A connection is the name of the neuron from which to take the output, a colon and the weight by which to multiply that output. Weight is a real number with scientific notation if desired.
- `output`: Required. Defines the neurons to which the neuron's output value will go. Its value is the names of the target neurons in brackets separated by commas, at least one.
- `bias`: Optional. Indicates the value of *bias* of the neuron. The value is a real number with scientific notation if desired. If this property is not indicated, it will take as value 0.
- `activation`: Optional. Indicates the activation function that will be applied to the calculation of the neuron output. It must be the name of one of the functions that the language implements, currently:
  - `identity`
  - `sigmoid`
  - `binary`

  If not indicated, it will take as value `identity`.

The connections defined by `input` and `output` must be logical and valid. If a neuron A has in its `input` another neuron B, B must have in `output` A. A and B must be "adjacent" to each other, there can be no layers between the two. They cannot be on the same layer. A has to appear after B in the network.

```
...
N {
  input: [Input1: 1, Input2: 1.3e-1],
  output: [Output1, Output2, Output3],
  bias: -0.14,
  activation: sigmoid
}
...
```

In this example, neuron "N" is defined with input from neuron "Input1" with weight 1 and from "Input2" with weight $1.3e-1$ ($= 0.13$); whose output value will go to the neurons "Output1", "Output2" and "Output3"; -0.14 bias and sigmoid activation function.

2.3.1 Exceptions in the input and output properties

The first neurons of the main network (one that is indicated directly in the program, not inside any element), in the input layer of the network, obviously there are no other neurons from which to take input. Therefore, in property `input`, instead of the name of another neuron, the name of an input variable is indicated. Furthermore, in this layer, neurons can only have one input (one of the input values). These input variables are indicated in the appropriate instruction (3.1) and can only be used in these neurons. If an input variable is used that is not indicated by an instruction, it will be undefined and therefore invalid.

The last neurons of the main network, in the output layer, obviously there are no other neurons to take the output to, therefore we have to save it in a variable (explained in 2.3.2), which in a way, can be understood how to name the output of the network.

2.3.2 Variables

In the properties of `input` and `output` of neurons, in addition to other neurons, variables can also appear. Variables have the same naming rules as elements (2).

In the property of `output` if a name other than any element appears, the output of that neuron will be assigned to the variable. This variable can now be used as an input to another neuron

(fulfilling the same valid and logical connection rules that exist between neurons) or an internal output (3.2.1). It is in a way, a way of renaming the neuron output.

Keep in mind that the neuron still needs its output to go somewhere, so if you decide to use variables as (only) neuron outputs, these variables have to be used so that the neuron's result goes to, at minus, another neuron (or an output from the output layer).

Variables can be used more than once by reassigning them more values, as long as these values are not overwritten in the same layer, invalidating the connection of two neurons.

Below is a basic example of using variables to connect "N1" and "N2".

```
...
C1 [
  N1{ input: [...], output: [OutputOfN1] },
  ...
],
C2 [
  N2{ input: [OutputOfN1: 0.5, ...], output: [...] },
  ...
]
...
```

3 Instructions

3.1 Input variables

To declare an input variable and assign it a value, enter the name of the variable, an equal symbol and the value.

```
Input = 3.14
```

In that code snippet, 3.14 has been assigned to the input variable "Input".

If at any point in the program you want to change the value of the input variable, you can assign it in the same way.

3.2 Output selection

Before executing a neuron network, we have the option of selecting outputs that we want to return in the next execution.

If no output is selected, all outputs on the network (output layer) with the names given to them will be automatically selected.

To select a variable as output, write a 'greater than' symbol followed by the name of the variable.

```
> Output
```

In this code fragment, the output "Output" is selected so that its value is returned in the next execution that is carried out.

You can select as many outputs as you want. If a name is not defined, `None` will be returned.

3.2.1 Internal outputs and elements as outputs

Internal variables of the network can also be selected as output, this includes both variables that are used to connect neurons and variables that are simply declared in `output` to assign the result to them.

Any element can also be selected for output by using the name of an element instead of a variable. The difference is that selecting items shows a basic explanation of the calculation that is

performed. If it is a neuron the calculation of the neuron is displayed. If it is a network or a layer, the calculations of all the output neurons of the element are shown.

These "internal outputs" to debug a neural network that we suspect is giving some wrong value because we have made a mistake when defining the network.

3.2.2 Special output "all"

You can also select "all" as the output. Selecting it will print the results of all neurons as they are calculated. This does not influence the rest of the selected outputs.

3.3 Running a network

To execute a network, write an arrow with a dash and a 'greater than' symbol (`->`) followed by the name of the network to be executed.

```
...
-> R
```

In that fragment the network "R" defined in the same program is executed.

The network will take the input variables that have previously been assigned and will print the selected outputs on the screen. Once finished, all exits will be unmarked.

4 Run a program

An NN program is executed through script `nn` as follows:

```
nn [-h] [-i input_file] [-I input] [-a] file
```

Being `file` the path to the file to be executed. In addition to the help option (`-h`) there are 3 other options that can be useful when running programs.

- `-i`: Along with a file in which only variables are defined, input variables can be passed to the program and therefore separated from the definition of the neural network.
- `-I`: Along with an input variable declaration, allows input variables to be passed to the program directly as an option when running.
- `-a`: If present, the special output "all" (3.2.2) will be activated during the entire program.

The variables that are passed in this way, either with a file or with the command line, will overwrite those of the program (during the entire execution) if they have the same name.