VENN How-To and Usage

This program is a Visual Editor of Neural Network.

Sections

What is a Neural Network?

A neural network is a statistical model which, given a specific input, returns an output which can correspond to:

- the percentage of how likely the input is part of a specific class - this network is called classifier -;
- the possible result of a generalized function which the network tries to approximate - this network is called function approximator -;
- divide the input in possible groups - or segments - of data - this network is called cluster and performs an action called segmentation.

These are the main types of "simple" neural networks, which are then specialized into serving a specific purpose. We can also find numerous other types of neural networks, such as Convolutional Neural Network (or CNN), Recurrent Neural Network (or RNN), neural network with reinforcement learning, neural network with greed-based learning and so on. But let's initially consider one of those "simple" neural networks' structure.

A network is divided in layers and arches. The mathematical meaning of every pair of layer-arch considering the most common type of layer, the fully-connected -or dense - layer, is:

- Given that an arch correspond to a mathematical function -activation_function-
- Given that a layer is formed by a weight matrix -w_matrix- and a bias matrix -b_matrix-

out_matrix = activation_function(in_matrix*w_matrix + b_matrix)

Where:

- in_matrix is the input matrix which goes into the given arch;

- out_matrix is the output matrix of the layer.

This n-th layer of the model's output is the n+1-th layer of the model's input, and viceversa this layer's input is the n-1-th layer of the model's output.

A computation can be done in training mode or in testing mode: the first part a feed forward is the same, but the training mode also performs afterwards a process of back propagation.

Feed Forward

Feed forward is the process of giving a neural network an input matrix which contains the data that the neural network should be able to generalize for finding an appropriate result. Through this step-by-step process the input are going to be computed - passed though activation functions, multiplied with the layers' weight matrices and added with the layers' bias matrices.

Let's consider for example a three layers model. In our example this model is formed as follows:

- The input, in_matrix, is a [1]x[m] row-matrix
- The output, out_matrix, is [1]x[4] row-vector where, ideally, one of the values is 1 and the other ones are 0s. (For this reason this network is an example of a classifier, where there are 4 possible classes and the sum of the values in the output vector, the percentages of membership to each class, needs to be 1)
- the hidden layer - every layer between the input and the output layers. In this case it is just one - has a weight matrix, w_matrix, [4]x[k], which means that this layer has k neurons, and a bias matrix, b_matrix, [1]x[k]
- A rectified linear (ReLu) activation function, act1, between the input layer and the hidden layer and a sigmoid activation function, act2, between the hidden layer and the output. The sigmoid activation function outputs the values between 0 and 1.
- The output of the hidden layer is defined as out1, while the final output is out2

That being said, let's proceed defining the mathematical formula of the model:

- out1 = act1(in_matrix*w_matrix + b_matrix)
- out2 = act2(out1*out_matrix)

So out2= act2(act1(in_matrix*w_matrix + b_matrix) * out_matrix)

This shows us that the output row vector, [1]x[4] as said earlier, is going to depend on the chosen activation functions as well as the w_matrix and b_matrix of the hidden layer.

Now we just have to give the neural network the power of learning the correct value given an input and an ideal output associated to that input.

Back Propagation

This technique is one of the available methods for granting the power of "learning" to the neural network. This is actually what makes the neural network such a powerful tool.

Without getting too mathematical, this kind of techniques can be summed as the problem of finding the optimal minimum point of the function which is the mathematical representation of the model. In fact a neural network can be imagined as a function of a diagram which divides it's domain into groups which each one represents one of the output classes. This means that, as a function, this model receives an input and its output is contained in one of the subgroups identified by this function - meaning that that input is one of the four classes with a percentage of 100%. The more pairs input-ideal output are given to the model, the more this function is refined and defines subgroups more precisely divided.

The main mathematical tool this technique uses is the derivative of each layer's component with respect to the next layer's output. This means that, from calculating the derivative of the output layer with respect to the loss between the output of the model and the ideal output, then it goes backward.

In this case, the steps are:

- Compute the loss function between the ideal output and the actual output of the model
- Compute the derivative of the hidden layer's weight and bias matrices with respect to the previously calculated loss
- Set each value of the w_matrix and b_matrix as:
    - new_w_matrix = old_w_matrix - (result of the derivative of the hidden layer's weight matrix)
    - new_b_matrix = old_b_matrix - (result of the derivative of the hidden layer's bias matrix)

This operation allows the weight and bias matrices of the hidden layer to change in such a way that, if the same input is given once again to the network, the output will be equal to the ideal output.

This process is repeated for a fairly large amount of different pairs input-ideal output so that the network is supposedly be able to generalize sufficiently so that if in the future a new input (of the same family) is given, it should return the correct output.

This whole process is called training process and can be improved with the help of various optimization functions, which help throughout the entire backpropagation process. Afterwards there is the testing process which, using new pairs input-ideal outputs, checks the percentage of correct "guesses" of the freshly trained model.

We could call the operation of a neural network as a "statistically educated series of guesses".

For more information regarding this subject I would suggest to read a specialized book, or read

the wikipedia page

VENN

Given this quick and oversimplified explanation of what a neural network is, let's dive into this tool.

This tool is designed and built to offer a graphical editor for the construction and manipulation of neural networks.

So far, neural networks had to be built by software specialists through more-or-less complex programs. Even if various frameworks greatly helped the task of building these models, someone new to this area should have initially started learning how to code, before even starting the actual process of creating a neural network.

Another somehow annoying problem was that every framework has its own standard file structure were a neural network model is temporarily saved making it impossible to directly convert a network's model from one framework to another.

Functionalities

This tool let's you to build a blocks-arrows scheme of the neural network you want to design, giving you infinite possible combinations, which makes space for your creativity to think and create everything it wants.

This project removes a layer of complexity to the creation of a neural network: to only thing you have to do is just decide how the network has to be and just drag n' drop the blocks in place.

Every graphical model can be saved in a json structure which can be then moved and sent everywhere. After being saved, the structure can be easily loaded into the system just by pressing a button.

Furthermore, the designed model can be exported in various frameworks. So far the following frameworks are supported: Keras, Tensorflow, Pytorch and FastAI (still in implementation phase ***). However many more frameworks can be easily supported, thanks to the wrapperTemplate class, which automatically performs bothersome tasks and gives you the prototype of functions which need to be implemented for the conversion of the graphical model into your framwork's. Once implemented these few functions, the system is ready to go, and you can start to graphically design your network and export it right away.

Another essential feature is that yo are able to design, create and export the system into your framework of choice and sequentially train and test it with input and output data saved on a file which you can freely add select from inside the program. Once the model is trained you can save it once again. A usable trained model ready for deployment.

Finally a great deal of importance was given to the freedom of choice of the user. A vast number of activation functions, loss functions, optimization functions and layer types are offered, allowing you to use functions which may be available in a framework but not in another. This grants you the

possibility to create every type of neural network, from simple to complex one, without having to go through the same annoying and difficult process of coding the structure, modify it, understanding of to use it, etc., again and again and again.