Neural Networks for Dummies

Introduction

Neural networks are 'self-learning' computer programs and are normally created and configured to solve one specific problem.

They are used in cases when it's too hard to use a traditional computer program.

Like in the case I will explain later on: we want to build a network that will recognize handwritten digits.
If you want to recognize handwritten digits with a traditional program, it probably will need many thousands lines of code. Our demo network will only use less than 1000 lines of code, and the best part: the same program code can be used for solving different problems (with just some minor adjustments).

Nowadays many neural networks are used in my different fields:
For instance Thunderbird (mail client) uses a neural network for identifying spam mails.
When the user marks an email as spam, it will train the network with that information.
(More about training later)

Other applications: license plate recognition, OCR, face recognition and so on.

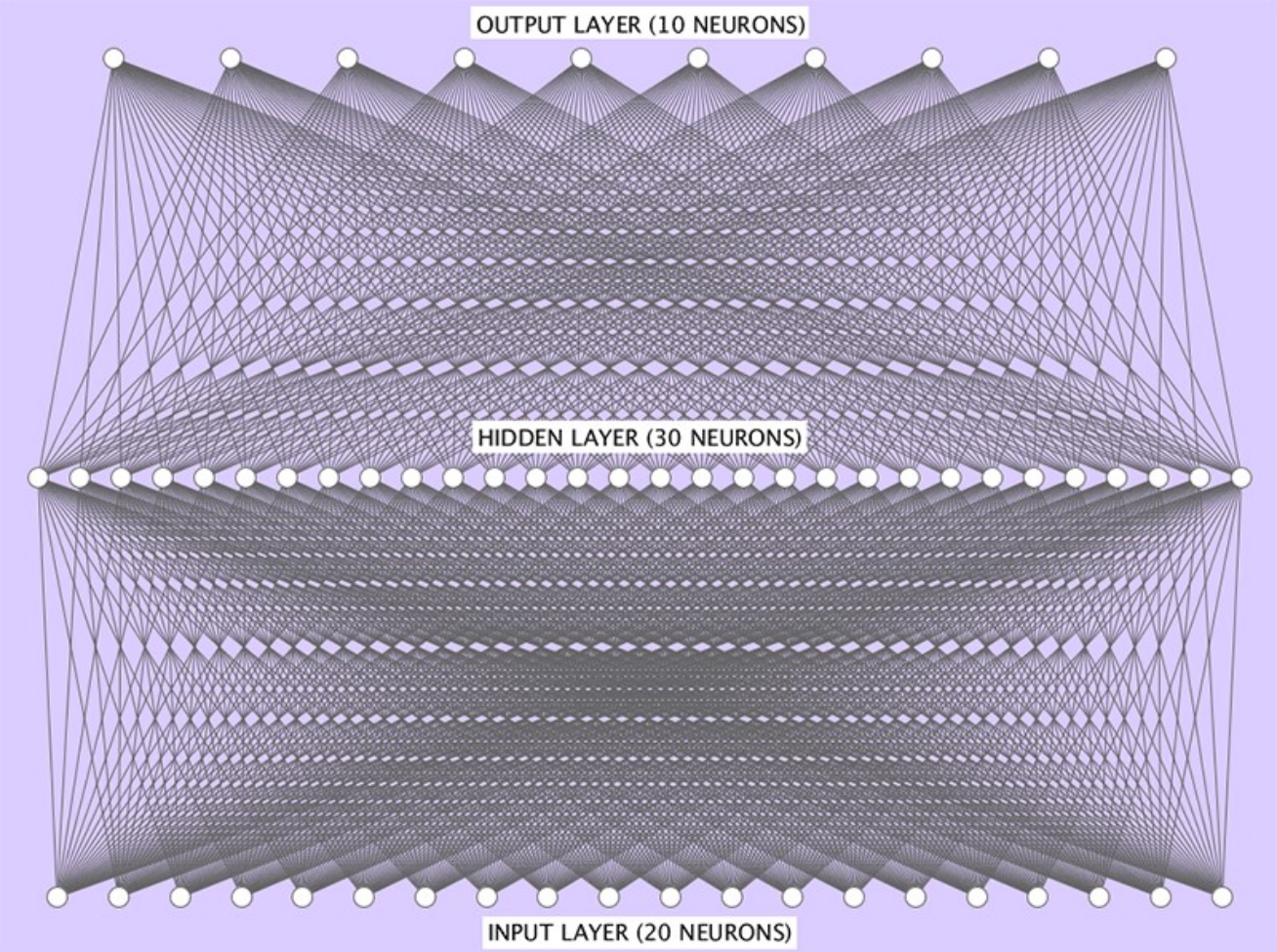
In fact neural networks simulate a tiny part of a human brain.

Like the human brain, a neural network is built with so-called '***neurons***'

A neural network has several layers, built with neurons.

There is always one '***input layer***', zero or more '***hidden layers***' and one '***output layer***'.

Schematic of a neural network with one hidden layer:



All neutrons are identical (same code) with one exception: neurons in the input layer don't have inputs themselves.

Depending on the problem it has to solve, the number of input neurons, the number of hidden layers and the number of neurons of the hidden layers and the number of output neurons will be chosen.

Every neuron in a layer is connected to all the neurons in the next layer (see above illustration).

In the above example there are already (20 * 30) + (30 * 10) = 900 connections (lines)!

In our test case (further on) we will use 196 input neurons, 100 hidden neurons and 10 output neurons. That's (196 * 100) + (100 * 10) = 20,600 connections!

During training and testing often the number of hidden layers and the number of hidden layer neurons will be changed to find the best configuration for performing that specific task.

How does it work?

Well, so far, no one really understands how it actually works... Especially the hidden layers are quite 'mysterious'.

There are two things you can do with the network: '***train the network***' and '***test and/or use the network***'.

Training a Network

Without training a network, the output of the network will be completely random. The network has no idea what to do with the input, so it just makes a guess.

Training is like teaching children: You tell them 2 + 2 = 4. And you tell them 1 + 6 = 7 and so on.
If you repeat that often enough, the child learns how to add two numbers.

Same for the network: during its training period you put inputs in and you tell it what the output should be.

Every neuron has two properties: '***weight***' and '***bias***' (sometimes also called '***threshold***').

These two properties determine what the output value will be and what the neuron will pass on to the neurons in the next layer, based on its inputs.

In short: the weight defines how important the value is; the bias defines how high the weight should be before passing it on to the next layer. Together they calculate the output value.

When a training input is processed, it generates output values based on the current state of the neurons.
These output values in the output layer are compared to the right answer. The properties will be changed a little bit, network will be tested again with the same input and the new output will be compared with the right answer. If the new output is better than the previous one, the new properties will be saved.
So, the more often you train the network, the more reliable the output will become.

Testing / Using a Network

After every training session, the network will be tested: did the reliability improve?

When the reliability is good enough, the network is ready for its job!

Training and testing use different sets of data: '***training-data***' and '***testing-data***'.
After training it, the network should be 'clever' enough to also solve inputs it never saw before.

These training- and testing datasets can be huge: millions of inputs. The basic rule is: the more the better!