Artificial Neural Networks 1

Roger Barlow
The University of Huddersfield

CODATA School on Data Science

Autumn 2021

The course: 5 minilectures

1. Introduction - what are Neural Networks and what can they do for us?
2. A first Neural Network - coding one in R turns out to be rather simple
3. Using an ANN - running your network and what it can do
4. A second Neural Network - downloading and using a more sophisticated program.
5. Conclusions and outlook - where you can go from here

There will also be 2 class zoom meetings

The main use of the internet seems to be

Sharing pictures of cute kittens and cuddly puppies

The human brain is remarkably good at telling which is which

Your brain is amazingly clever

at distinguishing between cats and dogs

The process is quick, powerful, and robust
and it doesn't use conventional flow-chart type logic

The Classification Problem is very general

Scientists don't (often) need to tell cats from dogs. But we do need to make similar decisions

- Particle Physicist: Is this track a pion or a muon?
- Astronomer: Is this blob a star or a galaxy?
- Seismologist: Was that an earthquake or a man-made explosion?
- Oceanographer: Are those sonar blips sharks or dolphins?
- Doctor: Is this patient sick or well?
- Banker: Is this company a sound investment or a big risk?
- IT analyst: Is this email spam or genuine?
- Everybody: Is this event signal or background?

and the brain has a solution.
Can we imitate it?

How does the brain work?

A drastically over-simplified account

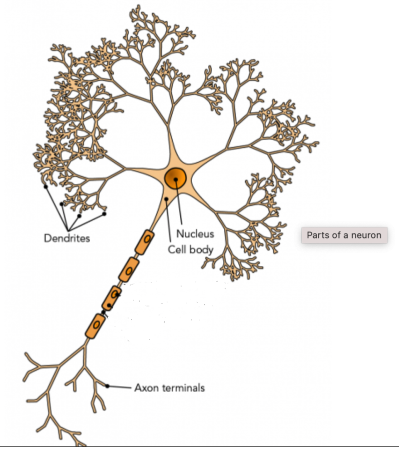
The brain is made of $\sim 100,000,000$ neurons

Each neuron has MANY inputs and MANY outputs

Inputs come from eyes, ears, touch - and other neurons

The neuron combines these into some value

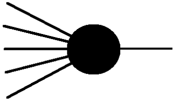
It outputs this value to muscles - and other neurons



Parts of a neuron (Source: Let's Talk Science using an image by Dhp1080 [CC BY-SA 3.0] via Wikimedia Commons).

Imitating a neuron (or node)

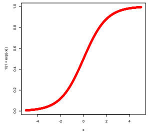
Suppose a neuron has N inputs, $R_1 \dots R_N$ and we want to form a result R.



Simple idea: Add them up. $\sum_j R_j$. Turns out to be too simple.

Better idea: Form a weighted sum $\sum_j w_j R_j$. where w_j expresses the importance (and direction) of each input j

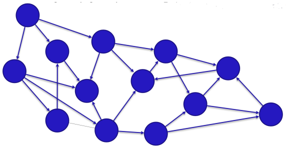
Even better idea: form a weighted sum and feed it through a threshold function to suppress extreme values.



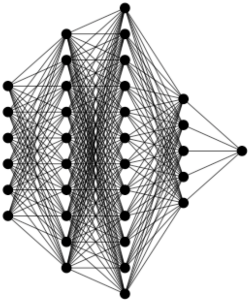
This function is a matter of choice. We will use the sigmoid: $f(x) = \frac{1}{1+e^{-x}}$. A common alternative is $\tanh x$. And there are others.

From Neurons to Networks

From these nodes/neurons you can form all sorts of topologies. General network theory is complicated.



We restrict ourselves to the **Multilayer Perceptron**.
Nodes are arranged in layers:
First layer is the input
Last layer output - ideally 1 (signal) or 0 (background)
In between - 'hidden' layers
Action is synchronised: all first layer effects second effectively simultaneously, then second effects third, and so on
Each thin black line corresponds to a weight:
$w_{ij}^{(\ell)}$ multiplies input j from layer ℓ in forming output i in layer $\ell + 1$



Training the network

Question: How do you set the weights?

Answer: By training.

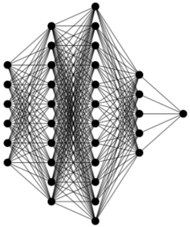
Take samples of known data ('supervised learning').
Present them one at a time to the network and get an output.
Increase the weights that push the output in the right direction, decrease those that push it in the wrong direction
Suppose an event has true value T. (1 for signal and 0 for background) and the network output is R
Define badness B

$$B = \frac{1}{2}(R - T)^2$$

Then change each $w_{ij}^{(\ell)}$ by $-\alpha \frac{\partial B}{\partial w_{ij}^{(\ell)}}$, where α is some small number

Calculating those differentials: Back propagation

if you want to get into the deep algebra



Start with the weights in the final layer: $w_{1j}^{(N)}$

$B = \frac{1}{2}(R - T)^2$ so $\frac{\partial B}{\partial w_{1j}^{(N)}} = (R - T)\frac{\partial R}{\partial w_{1j}^{(N)}}$

$R(x)$ is $\frac{1}{1+e^{-x}}$ with $x = \sum w_{1j}^{(N)} R_j^{(N)}$.

Neat result: $R'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = R(1 - R)$

So $\frac{\partial R}{\partial w_{1j}^{(N)}} = R(1 - R)R_j^{(N)}$ and $\frac{\partial B}{\partial w_{1j}^{(N)}} = (R - T)R(1 - R)R_j^{(N)}$

Now look at the next-to-last-layer weights $w_{jk}^{(N-1)}$

Same algebra until $\frac{\partial x}{\partial w_{jk}^{(N-1)}} = w_{1j}^{(N)} \frac{\partial R_j^{(N)}}{\partial w_{jk}^{(N-1)}} = w_{1j}^{(N)} R_j^{(N)}(1 - R_j^{(N)})R_k^{(N-1)}$

Then look at next-to-next-to last layer, and so on. Pick up lots of $R(1 - R)$ factors and some ws

Conclusion

Neural Networks are amazingly good at classification.
For an artificial neural network, you just set up the structure and then train it, adjusting the weights.

Examples next time.

Activity for you

Think of a classification problem in your field, and how a neural network might be used on it