# Peeking into the black box: Neural Networks

"Neural Networks" and "Deep Learning" have become quite the buzzwords nowadays, even within more traditional industries like insurance. However, ask any Actuary what's the most commonly used model he/she uses, and it is unlikely you'll hear any answer other than GLMs or GBMs.

## What is a Neural Network?

A neural network, as cool as it sounds, is nothing more than a statistical model. Much like its older brother - The simple linear regressor, the neural network also uses parameters and optimization to construct a line-of-best-fit for whatever data it is being presented. Unlike its old brother however, it is modelled from the way biological neural networks in our brains process information.

## Example: Foo Insurance Ltd

We find ourselves back at Foo Insurance Ltd with a rather interesting problem. Esther, our actuarial manager has some data that has been cleaned and normalized from the company's motor book. She wants to build a model to predict if a policyholder is likely to incur an accident claim based on the age of the car, and the policyholder himself/herself. She has heard a lot about neural networks and wants to test the model’s capabilities on this problem.

If you would like to play around with the widgets shown here, you can:

* Click here for the interactive binder version
* Click here for the source code for the interactive tools

INSERT DATASET SCREENSHOT

INSERT DATASET GRAPH

As you can tell, the dataset isn't linearly separable, which is the case in most situations. Although 2 features are a little over-simplified, it will serve the purpose for this article and extends to higher dimensions.

## First Iteration – Single Layered Network

Excited about playing with this new model, Esther proceeds to build a very simple network that looks something like this:

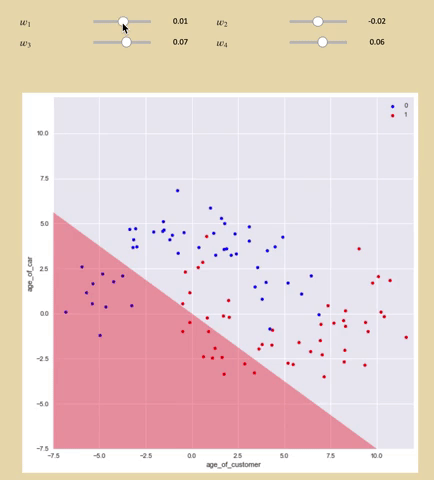
INSERT NODE CHART HERE

Recall that the formula for a neural network output on this dataset (without any bias terms) with 1 layer would look like:

Now, both outputs will give us some kind of probability or likelihood that the response is either 1 or 0. For example, if o\_1 gave an output of 0.6 and o\_2 gave 0.4, the particular data point would then be classified as green instead of grey.

By utilizing some good old high school linear algebra, recall that we can construct a line with this system of 2 equations, particularly:

This equation represents a line which is also known as a decision boundary and is what separates a policyholder which incurs a claim and one that does not.

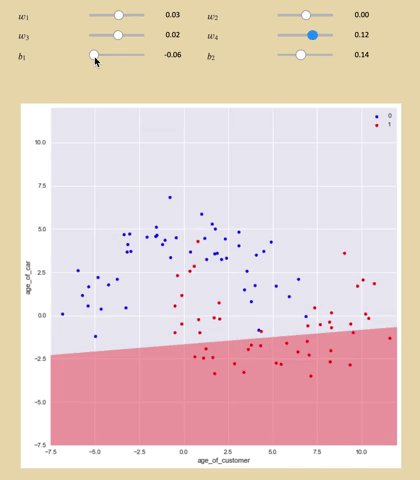


Looks like this first iteration does not really do a good job in properly classifying the 2 types of policyholders. Note that with the current network architecture, the boundary is restricted to rotating on the origin. Wouldn’t it be useful if we could shift this line up and down?

## Second Iteration – Single Layered Network with Bias Terms

Esther now recalls the concepts learnt in Simple Linear Regression, where an intercept is included in the model to give predictions some minimum value where the response turns out to be 0. She then proceeds to add an intercept (also known as a bias term) to the 2 initial equations, which gets:

Again, an equation for the decision boundary that separates 2 categories of policyholders.



Notice that the model now enables the decision boundary to shift up and down like a simple linear regression model, giving Esther more capabilities to accurately model the 2 policyholder categories.

## Third Iteration –Network with Hidden Layer

Even though it is a little better now, remember that, similar to real life, the dataset here is not linearly separable and our current network has is only capable of producing a straight line to differentiate the 2 categories.

Will adding more layers of neurons give the model non-linear capabilities? Well, going back to high school mathematics yet again, recall that a linear combination of 2 linear equations still produces something linear.

Thus, no matter how many layers of neurons Esther decides to stack on top of each other, the decision boundary produced by the model will always be linear! How then, can Esther make the network adapt to non-linear data?

## Fourth Iteration – Multi Layered Network with Activation Functions

The trick to introducing non-linearities in the network is to wrap each output with some non-linear function.

In this example, let’s take the f(x) to be Sigmoid function, meaning:

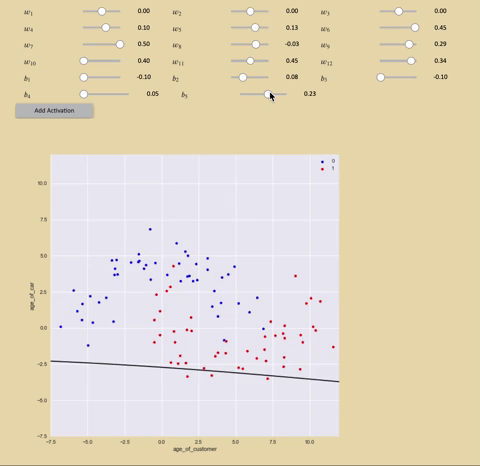
Imagine now, that Esther introduces these activation functions into the model, and uses a single layered network.

INSERT NETWORK GRAPH HERE

...

…

What the actuarial fish. Note that this is the decision boundary for a relatively simple architecture and activation function. While Esther will not be able to analytically solve this for x2, she would chuck this monstrosity into a pre-built solver and an implicit equation graphing tool.



It seems like now Esther has a model that is capable of producing non-linear decision boundaries but is it enough for this task?

If she (or as a matter of fact, you the reader) wanted to, she could definitely spend time to tweak each of the biases and weights manually to get a reasonable decision boundary but a more popular approach is to use algorithms such as “Gradient Descent” to find these optimal values. The only “Descent” you will achieve by doing this manually is the one into madness.

## So, what’s the issue now?

As of now, it seems Esther has a flexible model in which she can explain down to the last weight change. Why then is everyone still calling neural networks a black box?

There are various deeper issues regarding neural networks which we will not discuss here, but it must be stressed that the architecture of the network Esther built here is extremely simple (with 3 hidden neurons) and even then, we can see the effect of combinatorial explosion on the weights while trying to derive the equation for the decision boundary. Imagine if this were a more realistically-sized network with 2 hidden layers, each with 56 hidden neurons? A single weight’s impact and interpretation would not be as clear as it is now, and we would not have as much freedom to tweak the weights individually like we are currently doing.

## Not tractable, so… useless?

Although we currently are not able to interpret the resulting weights/coefficients of a neural network model like we can for a GLM, there are methods we can utilize to “examine the black box from different angles”, giving us some insights on what the network is lacking or doing well for iteration purposes.

If you are interested in finding out how to apply a proper deep learning model to insurance data, check out Jacky Poon’s article: Multitasking Risk Pricing Using Deep Learning to see how powerful a neural network can be!

Maybe the insurance industry isn’t as eager to dive into “partially explainable AI” compared to tech firms like Facebook or Netflix because a decision made by a model that would cause a family to lose their live savings is arguably more important than getting your recommended movie list wrong. Nevertheless, the future is definitely heading in that direction and the actuarial profession must be well prepared for when that time comes.