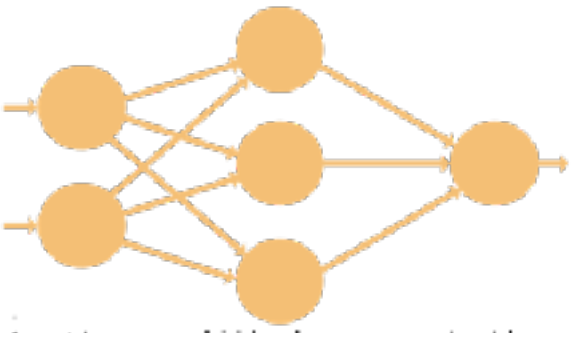
Neural Networks

Neural Networks are essentially our way of trying to mimic the all powerful functions of the human brain. When I first heard of it, I was pretty excited and a little skeptical because I didn't really think we could achieve something so complex. Turns out we're not too bad at it. It starts with a simple understanding of the human brain.

The human brain is made up of a bunch of neurons that communicate with other neurons as a network through electrical signals. So we take that concept and we create an entity that mimics a neural and string as many of these as we need together to create a network of neurons, or in other words - a Neural Network.

Just so that we're on the same page, think of a neuron as a cell that takes some inputs, applies a function to it and gives you an output. Simple right? When you assemble a network of hundreds of these little things, it can get pretty powerful.

Neural Networks are designed to work best when you have layers or neurons working together. The depth (number of layers) and the width (number of neurons in a layer) of the network are key to getting good results. Designing the topology of a neural network in a task in itself.



The circles are the neurons and the lines are the inputs and outputs to them. Each of these inputs is multiplied by an associated weight and the combination of the weights and inputs to each neuron are passed through a function (these are called activations) and the output of the function proceeds to the next layer.

The first and the last layer are the input and output layers. The middle layer is the hidden layer and there can be as many of them as needed.

The fundamental concept of a neural network is that it keeps learning based on the data you give it. Just like how humans work, we learn from experiences. Just like with all other Machine Learning algorithms, there is a training phase and a testing phase.

During the training phase, data is fed into the network and with every forward pass there is a certain output emitted, this output is compared with the expected output for the particular input and the loss is calculated. Then you do a backward pass through the network and calculate how much of an effect each neuron had on the overall output and adjust the weights accordingly. This backward pass is called Back Propagation.

While testing, you pass in inputs to the network and it predicts what it 'thinks' the next item in the sequence will be.

The reason neural networks have become so popular now is because people are starting to realise how powerful these networks can be. Research on Artificial Neural Networks started many years

ago but one of the biggest problems involving them was the huge amount of computation required to run them. Even with the highly advanced hardware we have today, simple neural networks can take up to days to train effectively.

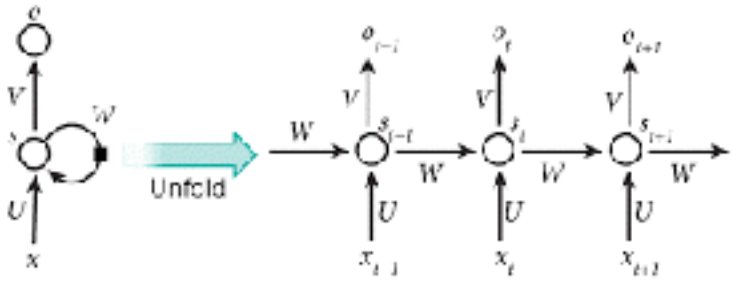
There are so many applications to Neural Networks, from creating chat bots, to creating a poker playing computer that can beat some of the worlds best players. As the complexity of the task increases, the complexity of the neural network also increases. Variants of Neural Networks have been researched and designed and the results of these networks have been astonishing.

The focus of this article series is going to be on introducing some of the variants of Neural Networks the are largely used today, i.e. Recurrent Neural Networks (RNN), Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU). Keep in mind that things might get pretty funky and mathematical, but don't worry about it. You'll manage just fine even if you aren't a math wizard.

I'll start with Recurrent Neural Networks.

The simple network you see in the photo above, simply takes an input, passes it through a couple of neurons and give you an ouput. The assumption with the simple NN is that the inputs are independent of each other. In most practical cases, this is an assumption that can't be made. Think of an application where you're training an NN to learn the alphabet, or english sentences. Each letter is depends on the previous letter and it's the combination of these letters that need to be learnt. Think of an RNN to sort of have the ability to remember the previous inputs.

RNN makes use of information that is sequential in time. The 'Recurrent' part is because each input in the sequence goes through the entire network and each output is dependent on previous steps.



This diagram shows you perfectly how the recurrence kicks in. Here you have a hidden state 's' which you can think of as the network' s memory. You can see from the diagram how at each step in time 't' the output at that step depends not only on the input at that step but also at the 'memory' from the previous step.

The parameters 'U', 'V' and 'W' are the weights associated with the inputs, outputs and hidden state respectively.

When all inputs have been passed through this network, the loss is calculated and Back Propagation takes place. The key thing to note here is that now you will back propagate not only through the networks, but also through each of the steps in time. This is called Back Propagation Through Time (BPTT) and you can find a brilliant explanation for BPTT here.

Since I promised you fancy math, here are the equations involved in an RNN.

** Insert equations here **
Solving these equations is how you would practically implement an RNN.