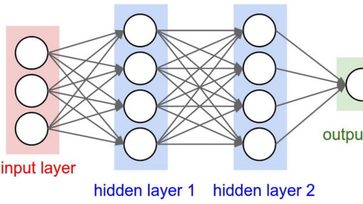
1. What are ANNs?

Artificial neural networks are one of the main tools used in machine learning. As the "neural" part of their name suggests, they are brain-inspired systems which are intended to replicate the way that we humans learn.

Neural networks consist of input and output layers, as well as (in most cases) a hidden layer consisting of units that transform the input into something that the output layer can use.

ANNs have three layers that are interconnected. The first layer consists of input neurons. Those neurons send data on to the second layer, which in turn sends the output neurons to the third layer. ANNs are considered non-linear statistical data modeling tools where the complex relationships between inputs and outputs are modeled or patterns are found. Note that a neuron can also be referred to as a perceptron.



For a basic idea of how a deep learning neural network learns, imagine a factory line. After the raw materials (the data set) are input, they are then passed down the conveyer belt, with each subsequent stop or layer extracting a different set of high-level features. If the network is intended to recognize an object, the first layer might analyze the brightness of its pixels. The next layer could then identify any edges in the image, based on lines of similar pixels. After this, another layer may recognize textures and shapes, and so on. By the time the fourth or fifth layer is reached, the deep learning net will have created complex feature detectors. It can figure out that certain image elements (such as a pair of eyes, a nose, and a mouth) are commonly found together.

Once this is done, the researchers who have trained the network can give labels to the output, and then use backpropagation to correct any mistakes which have been made. After a while, the network can carry out its own classification tasks without needing humans to help every time.

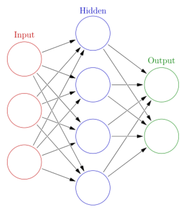
2. Is there a difference between NN and ANN?

Neural Network is a broad term that encompases various types of networks which were shown above. Is ANN one of the types? Well to understand this it is important to realise that neural network alone is not an algorithm but a framework which assists the algorithms to work. ANN is the most basic type of implementation of neurals. ANN was the term coined much earlier and nowadays the two terms are interchangeably used.

3. How does ANN work?

It is truly said that the working of ANN takes its roots from the neural network residing in human brain. ANN operates on something referred to as Hidden State. These hidden states are similar to neurons. Each of these hidden state is a transient form which has a probabilistic behavior. A grid of such hidden state act as a bridge between the input and the output.

We have an input layer which is the data we provide to the ANN. We have the hidden layers, which is where the magic happens. Lastly, we have the output layer, which is where the finished computations of the network are placed for us to use.



Initially the weights of the network can be randomly. When the input in given to the input layer the process moves forward and the hidden layer receives the input combined with the weights. This process goes on till the final layer of output is reached and result is given. When the result is out it is compared to the actual value and a back propagation algorithm comes into play to adjust the weights of the network linkages to better the result. What do the neurons in the layers then do? They are responsible for the learning individually. They consist of activation function that allows the signal to pass or not depending on which activation function is being used and what input came from the previous layer. We'll see activation functions in detail now.

1. Assign Random weights to all the linkages to start the algorithm
2. Using the inputs and the (Input ->Hidden node) linkages find the activation rate of Hidden Nodes
3. Using the activation rate of Hidden nodes and linkages to Output, find the activation rate of Output Nodes
4. Find the error rate at the output node and recalibrate all the linkages between Hidden Nodes and Output Nodes
5. Using the Weights and error found at Output node, cascade down the error to Hidden Nodes
6. Recalibrate the weights between hidden node and the input nodes
7. Repeat the process till the convergence criterion is met
8. Using the final linkage weights score the activation rate of the output nodes

4. Activation Function

Activation functions are really important for a Artificial Neural Network to learn and make sense of something really complicated and Non-linear complex functional mappings between the inputs and response variable.They introduce non-linear properties to our Network.Their main purpose is to convert a input signal of a node in a A-NN to an output signal. That output signal now is used as a input in the next layer in the stack.

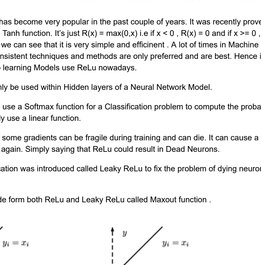
Specifically in A-NN we do the sum of products of inputs(X) and their corresponding Weights(W) and apply a Activation function f(x) to it to get the output of that layer and feed it as an input to the next layer.

Most popular types of Activation functions -

- Sigmoid or Logistic
- Tanh—Hyperbolic tangent
- ReLu -Rectified linear units

Sigmoid Activation function: It is a activation function of form f(x) = 1 / 1 + exp(-x) . Its Range is between 0 and 1. It is a S—shaped curve. It is easy to understand and apply but it has major reasons which have made it fall out of popularity -

- Vanishing gradient problem
- Secondly , its output isn't zero centered. It makes the gradient updates go too far in different directions. 0 < output < 1, and it makes optimization harder.
- Sigmoids saturate and kill gradients.
- Sigmoids have slow convergence.



Hyperbolic Tangent function- Tanh : It's mathamatical formula is f(x) = 1—exp(-2x) / 1 + exp(-2x). Now it's output is zero centered because its range in between -1 to 1 i.e -1 < output < 1 . Hence optimization is easier in this method hence in practice it is always preferred over Sigmoid function . But still it suffers from Vanishing gradient problem.

ReLu- Rectified Linear units : It has become very popular in the past couple of years. It was recently proved that it had 6 times improvement in convergence from Tanh function. It's just R(x) = max(0,x) i.e if x < 0 , R(x) = 0 and if x >= 0 , R(x) = x. Hence as seeing the mathamatical form of this function we can see that it is very simple and efficient . A lot of times in Machine learning and computer science we notice that most simple and consistent techniques and methods are only preferred and are best. Hence it avoids and rectifies vanishing gradient problem . Almost all deep learning Models use ReLu nowadays.

But its limitation is that it should only be used within Hidden layers of a Neural Network Model.

Hence for output layers we should use a Softmax function for a Classification problem to compute the probabilites for the classes , and for a regression problem it should simply use a linear function.

Another problem with ReLu is that some gradients can be fragile during training and can die. It can cause a weight update which will makes it never activate on any data point again. Simply saying that ReLu could result in Dead Neurons.

To fix this problem another modification was introduced called Leaky ReLu to fix the problem of dying neurons. It introduces a small slope to keep the updates alive.

We then have another variant made form both ReLu and Leaky ReLu called Maxout function .

In [ ]: