ANN

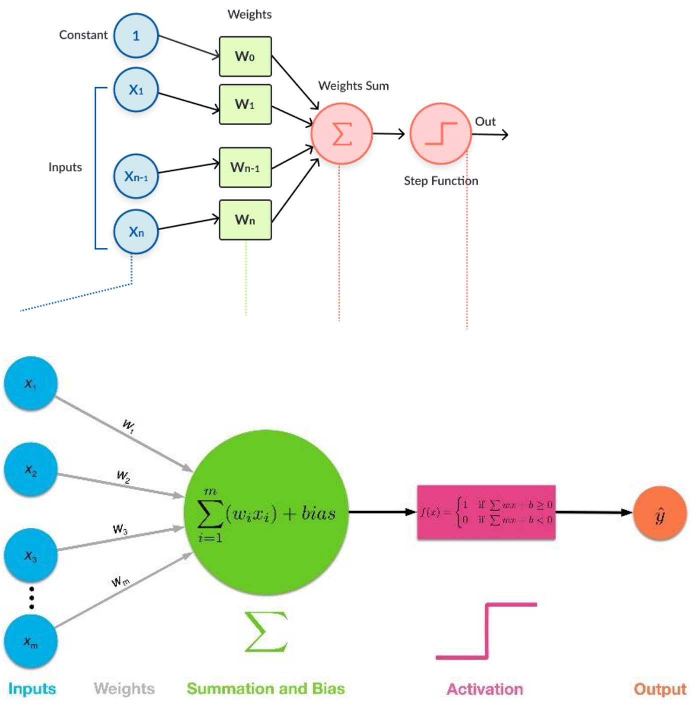
Artificial Neural Network is one of the most beautiful and basic concepts of Supervised Deep Learning. ANNs are considered nonlinear statistical data modeling tools where the complex relationships between inputs and outputs are modeled or patterns are found.

ANN can be used for both regression and classification problems but is rarely used for predictive modelling because it usually tries to over-fit the relationship. ANN is generally used in cases where what has happened in past is repeated almost exactly in same way.

Types

- Perceptron

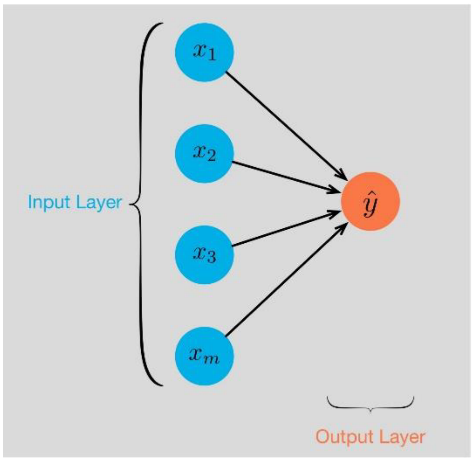
Perceptron is an artificial neuron network model which doesn’t have any hidden layer. So the perceptron is a special type of a unit or a neuron and it uses step function as the activation function

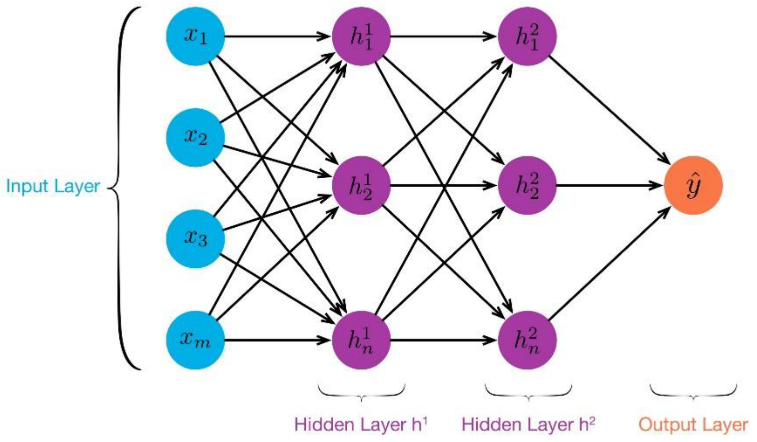


- Multi-layer ANN

A multilayered network means that you have at least one hidden layer

More sophisticated than the perceptron, a Multi-layer ANN (Convolutional Neural Network, Recurrent Neural Network) is capable of solving more complex classification and regression tasks thanks to its hidden layers





We know that a layer of ANN just performs a non-linear transformation of its inputs from one vector space to another. If we take a classification problem as an example, we want to separate out the classes by drawing a decision boundary. Here the input data in its given form is not separable. So by performing non-linear transformations at each layer, we are able to project the input to a new vector space, and draw a complex decision boundary to separate the classes. The main benefit of having a deeper model is being able to do more non-linear transformations of the input and drawing a more complex decision boundary.

Example

Input

The inputs are simply the measures of our features. For a single soil sample, this would be an array of values for each measurement. For example, we may have an input of:

$$[\ 58\ ,\ 1.3\ ,\ 11\ ]$$

representing 58% moisture, 1.3mm grain size, and 11 micrograms iron per kg soil weight. These inputs are what will be modified by the perceptron.

Weights

Weights represent scalar multiplications. Their job is to assess the importance of each input, as well as directionality. For example, does more iron contribute a lot or a little to height? Does it make the tree taller or shorter? Getting these weights right is a very difficult task, and there are many different values to try.

If we multiply weights by the inputs, we get the following result:

$$58 * 0.2 = 11.6$$
$$1.3 * 9.6 = 12.48$$
$$11 * -0.9 = -9.9$$

Transfer Function

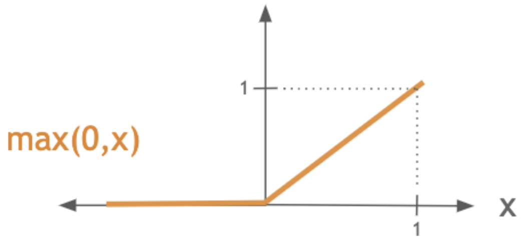
The transfer function is different from the other components in that it takes multiple inputs. The job of the transfer function is to combine multiple inputs into one output value so that the activation function can be applied. This is usually done with a simple summation of all the inputs to the transfer function.

On its own, this scalar value is supposed to represent some information about the soil content. This value has already factored in the importance of each measurement, using the weights. Now it is a single value that we can actually use. You can almost think of this as an arbitrary weighted index of the soil's components. If we have a lot of these indexes, it might become easier to predict tree height using them. Before the value is sent out of the perceptron as the final output, however, it is transformed using an activation function.

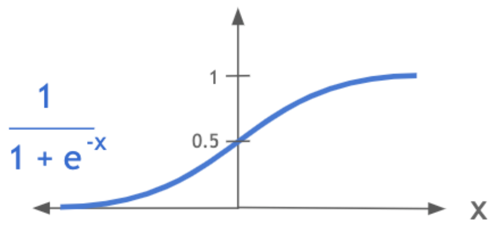
$$11.6 + 12.48 + -9.9 = 14.18$$

An Activation function will transform the number from the transfer function into a value that dramatizes the input. Often times, the activation function will be non-linear.

For now, just remember that introducing non-linearity to the perceptron helps avoid the output varying linearly with the inputs and therefore allows for greater complexity to the model. Below are two common activation functions.



ReLU is a simple function that compares zero with the input and picks the maximum. That means that any negative input comes out as zero, while positive inputs are unaffected. This is useful in situations where negative values don't make much sense, or for removing linearity without having to do any heavy computations.

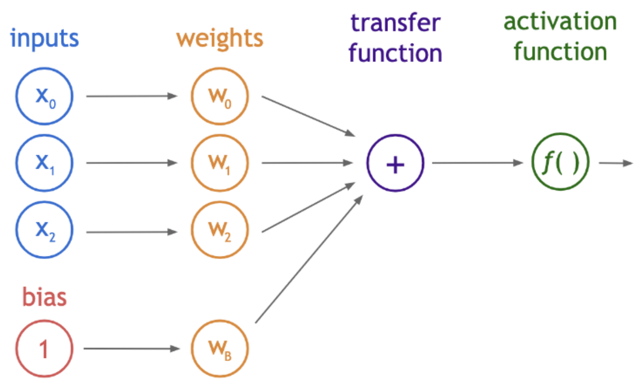


The sigmoid function does a good job of separating values into different thresholds. It is particularly useful for values such as z-scores, where values towards the mean (zero) need to be looked at carefully since a small change near the mean may significantly affect a specific behavior, but where values far from the mean probably indicate the same thing about the data. For example, if soil has lots and lots of moisture, a small addition to moisture probably won't affect tree height, but it if has a very average level of moisture then removing some small amount of moisture could affect the tree height significantly. It emphasizes the difference in values if they are closer to zero.

When you think of activation functions, just remember that it's a nonlinear function that makes the input more dramatic. That is, inputs closer to zero are typically affected more than inputs far away from zero. The output of the ReLU activation function will be:

$$\max(0, 14.18) = 14.18$$

Up until now, I have ignored one element of the perceptron that is essential to its success. It is an additional input of 1. This input always stays the same, in every perceptron. It is multiplied by a weight just like the other inputs are, and its purpose is to allow the value before the activation function to be shifted up and down, independent of the inputs themselves. This allows the other weights (for the actual inputs, not the weight for the bias) to be more specific since they don’t have to also try to balance the total sum to be around 0.



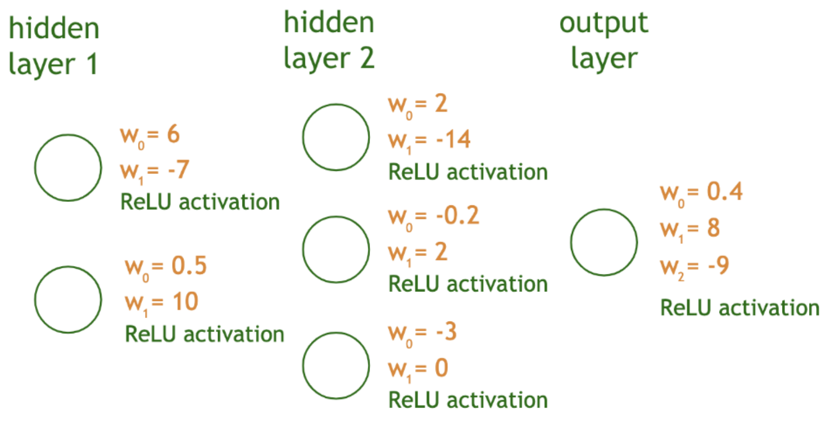
$$f(\; x_0 \cdot w_0 + x_1 \cdot w_1 + x_2 \cdot w_2 + 1 \cdot w_B \;)$$

Example

Input: Here are the values of the two features I will use. They represent 58% moisture and 1.3mm grain size.

$$[\ 58\ ,\ 1.3\ ]$$

I will use the following (random) weights and activation functions for each perceptron. Recall that the ReLU activation function turns negative values into 0 and does not transform positive values:



The first two perceptrons both take the two inputs (blue), multiplies them by the associated weights (yellow), adds them (purple), and then applies the ReLU function (green):

$$\max(\ 0\ ,\ 58 \cdot 6\ +\ 1.3 \cdot -7\ ) = 338.9$$
$$\max(\ 0\ ,\ 58 \cdot 0.5 + 1.3 \cdot 10\ ) = 42$$

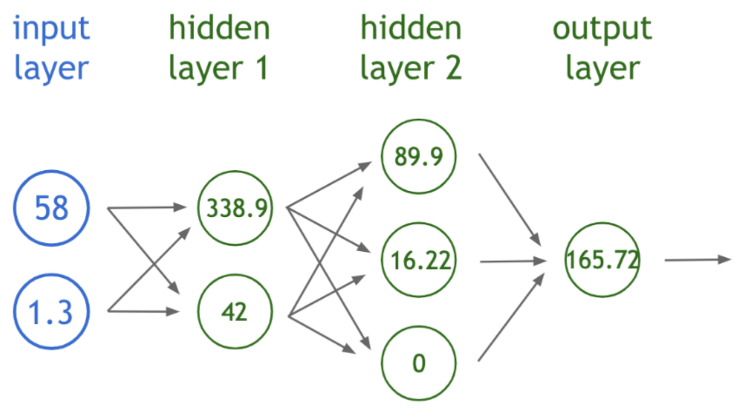
These outputs become the inputs for each perceptron in the third layer. So every perceptron in the second hidden layer (there are three) will use 338.9 and 42 as inputs. Those perceptrons follow the following equations:

$$\max(\ 0\ , 338.9 \cdot 2\ +\ 42 \cdot -14\ ) = 89.9$$
$$\max(\ 0\ , 338.9 \cdot -0.2 + 42 \cdot 2\ ) = 16.22$$
$$\max(\ 0\ , 338.9 \cdot -3\ +\ 42 \cdot 0\ ) = 0$$

For the next layer, however, notice that we now have three, not two, inputs: 89.9, 16.22, and 0. All three inputs have to be included in the equation of the last perceptron, and therefore it will have three weights (in yellow below). Its equation is still as straightforward as the others.

$$\max(0, 89.9 \cdot 0.4 + 16.22 \cdot 8 + 0 \cdot -9) = 165.72$$

As a summary, here are the values each perceptron produced given its inputs



And there you have it! This neural network predicted a tree with a height of 165.72 feet!