Chapter 8 : Part 1

**Deep Learning**

**ANN: Artificial Neural Network**

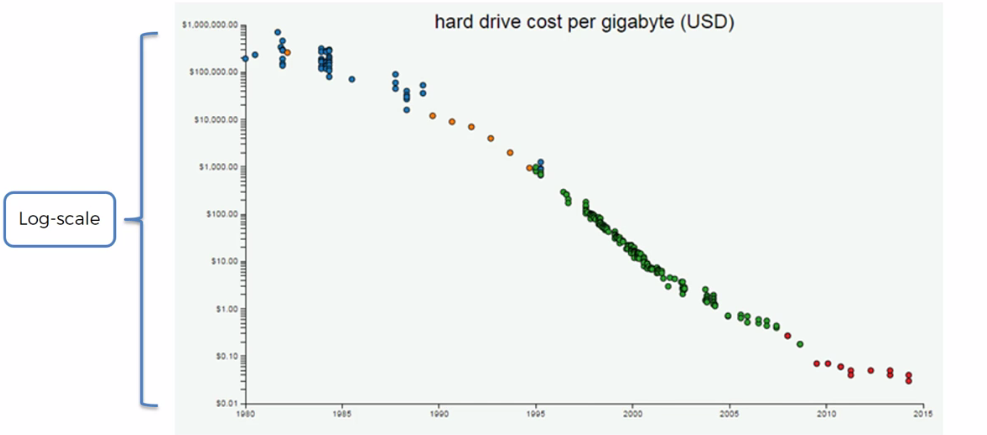
Introduction to Neural Network

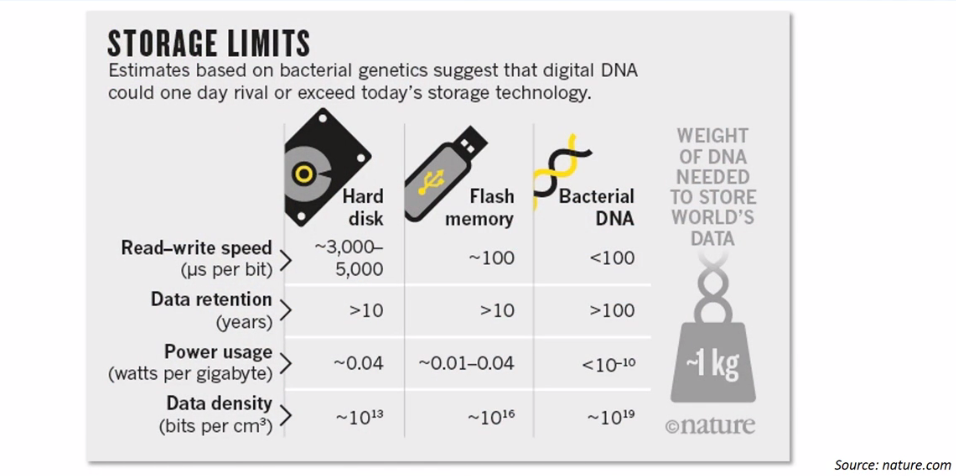
**8.1.1 History of Deep Learning**

Deep learning was invented in the 60s and 70s, in the 80s so people are talking about them a lot. But that trend became slow and died during the following decades. The reason was, the idea was not so clear and technology wasn't ready.

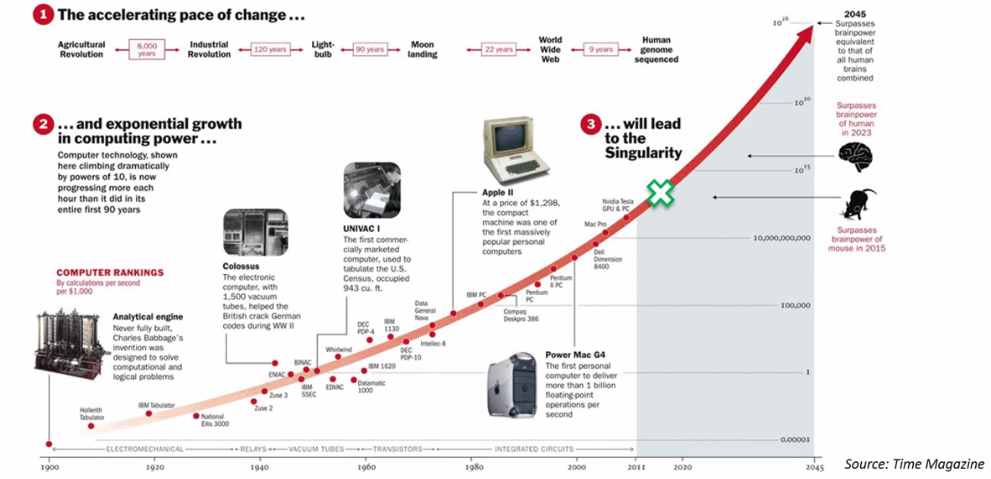
From 2000 to 2020 the *storage* *capacity* become ***large*** ***enough*** and the *processing* *power* of the computers ***increased***. That’s why this Deep Learning concept became so popular in the recent years.

And in deep learning to work properly, you need two things: a ***lot of data*** and ***processing*** ***power*** (strong computers to process that data).





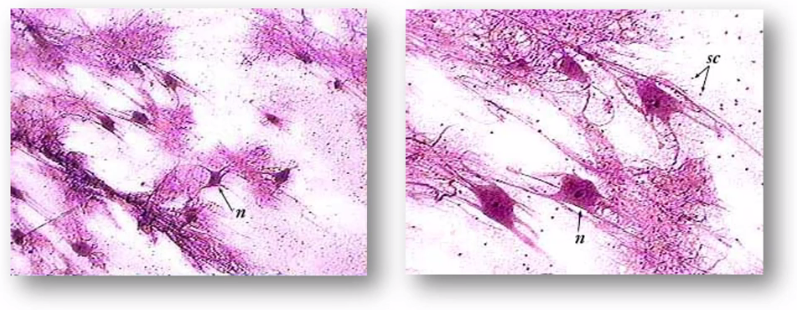
DNA storage Comparison

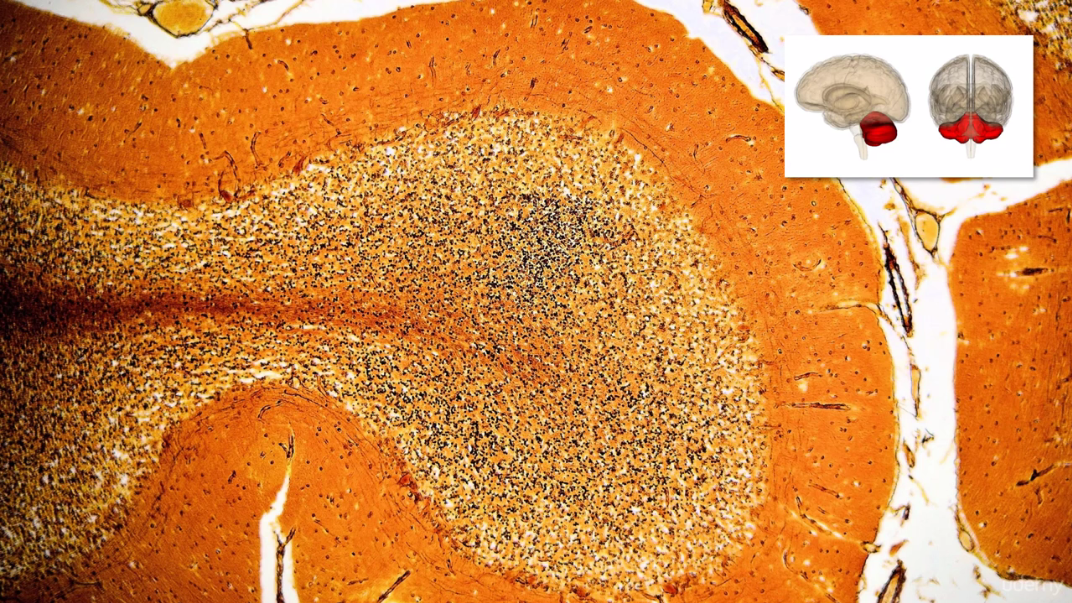


Processing capacity is increasing

|  |  |
| --- | --- |
| * What is deep learning: This gentleman over here is ***Jeffrey Hinton***, known as the godfather of Deep Learning. And he did research on deep learning in the 80s and he's done lots and lots of work lots of research papers he's published in deep learning. Right now he works at **Google**. * The idea behind deep learning is: to look at the human brain. Here we've got some neurons, they have a body, branches, tails and a nucleus in the middle and that's that's basically what a neuron looks like in the human brain. * There's approximately 100 billion neurons all together so these are individual neurons these are actually motor neurons because they're bigger they're easier to see but nevertheless there's a hundred billion neurons in the human brain. |  |

* And each of neurons connected to as many as about a ***thousand*** of its ***neighbors***.

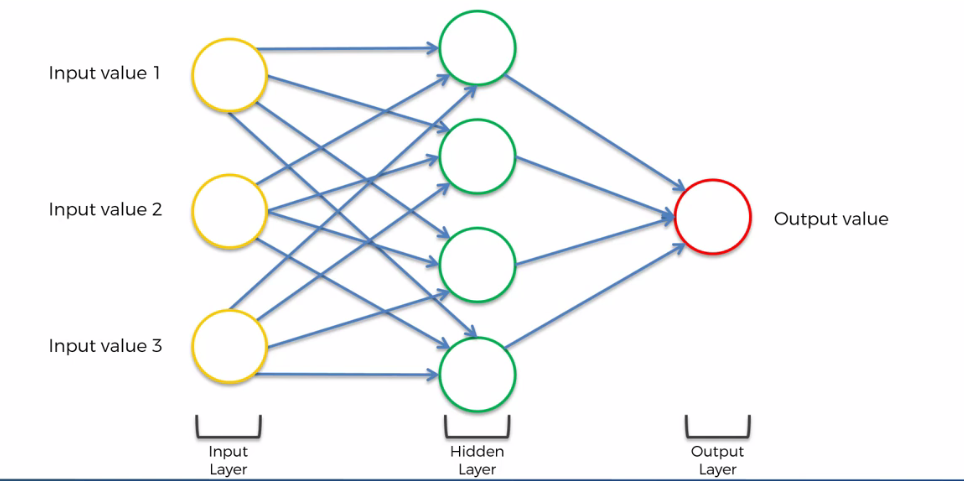




Actual section of Cerebellum

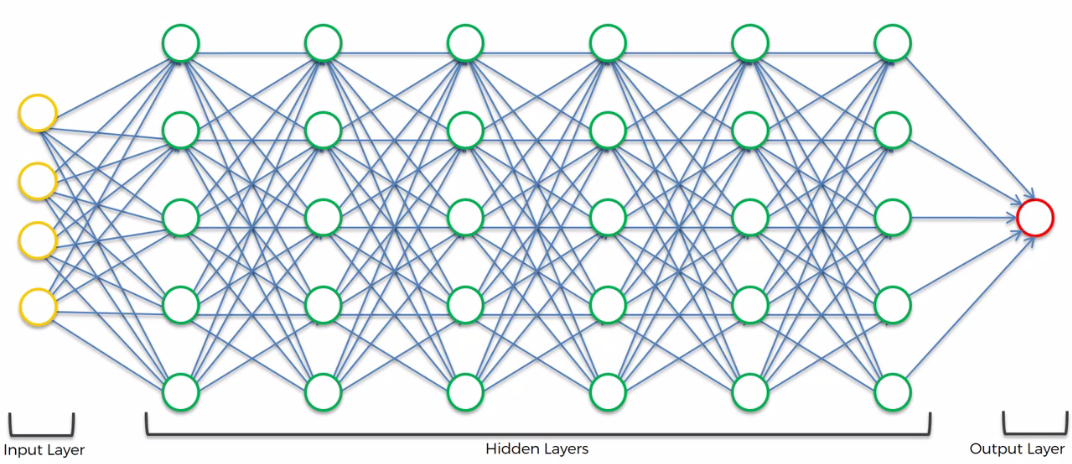
*Above* *image* shows how many *neurons* there are, like billions and billions and billions of neurons all connecting each other forming a *network*. And that's what we're going to be trying to *recreate* in our computer.

* How do we recreate this in a computer: We'll create an artificial structure called an ***artificial neural net*** where we have ***nodes*** or ***neurons***. We're going to have some



Shallow-Learning

* Input layer: Neurons for input values: These are values that you know about a certain situation. For instance: you're *modeling* *something*, you want to *predict* *something* you always could have some *input* something to *start*., then that's called the input layer.
* Output layer: Then you have the output. That's of value that you ***want to predict***. EG: For a transaction in a bank, is this a fraud-transaction it's a real-transaction and so on. So that's going to be output layer.
* Hidden layer(s): Between *Input layer* and *Output layer* we're going to have a *hidden* *layer*. The input layers neurons connected to a hidden layer neurons that neurons are connect to output.
* Shallow-Learning and Deep-Learning: For a few Hidden Layers it is called Shallow-Learning. When the Hidden Layer increases then it is called Deep Learning.



Deep Learning

* And that's how the *input* *values* are *processed* through all these *hidden* *layers* just like in the human brain. Then we have an output. So that's what Deep-learning is all about on a very abstract level.

**What we will learn in this section**

1. The Neuron: There'll be a little bit of Neuroscience and we'll find out a bit about how the human brain works and why we are trying to replicate that. And we'll also see what the *main building block* of a Neural Network of the Neuron looks like.
2. The Activation Function: We'll talk about the *activation* *function* and we'll look at a couple of *examples* of *activation* *functions* that you could use in your *neural* *networks* and we'll find out which ones of them is the most *commonly* used in *neural* *networks* and in which *layers* you'd rather use.
3. How do Neural Networks work? (example): We're not going jump into the *learning part* directly, instead we're actually going to go into the *working of the Neural Networks first* because that way by *seeing a Neural Network in action* that will allow us to understand what we're aiming towards what our goal is.

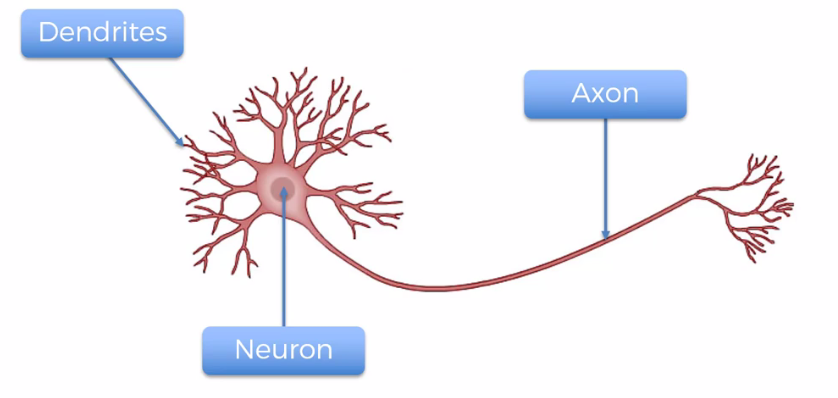
* So here we'll look at an example of a neural network: we're going to look at a very simplified hypothetical example: to *predict* *housing* *prices*.

1. How do Neural Networks learn?: We will move on to understanding how Neural Networks learn.
2. Gradient Descent: This is also part of neural networks learning and we'll find out how great the advantage of Gradient Descent are.
3. Stochastic Gradient Descent: It's a it's a continuation of the Gradient Descent tutorial but it's an even better and even stronger method and we'll find out exactly how it works.
4. Backpropagation: And finally we'll wrap things up by mentioning the important things about back propagation and summarizing everything in a step by step set of instructions for running your artificial neural networks.

**8.1.2 The Neuron**

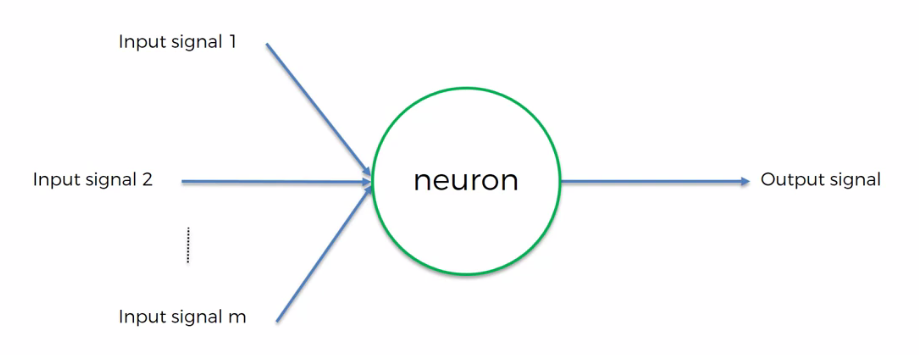
Neuron is the basic building block of Artificial Neural Networks. Now before recreate it, lets talk about Biological Neuron.

* Neurons by themselves are pretty much useless. It's like an ant. For example an ant can't do anything, but five ants together can pick something up. And if you have a million ants they can build a whole colony they can build an anthill. They can act like an Organism.
* Same thing with the neurons. By itself it's not that strong but when you have lots of neurons together they work together to do magic.

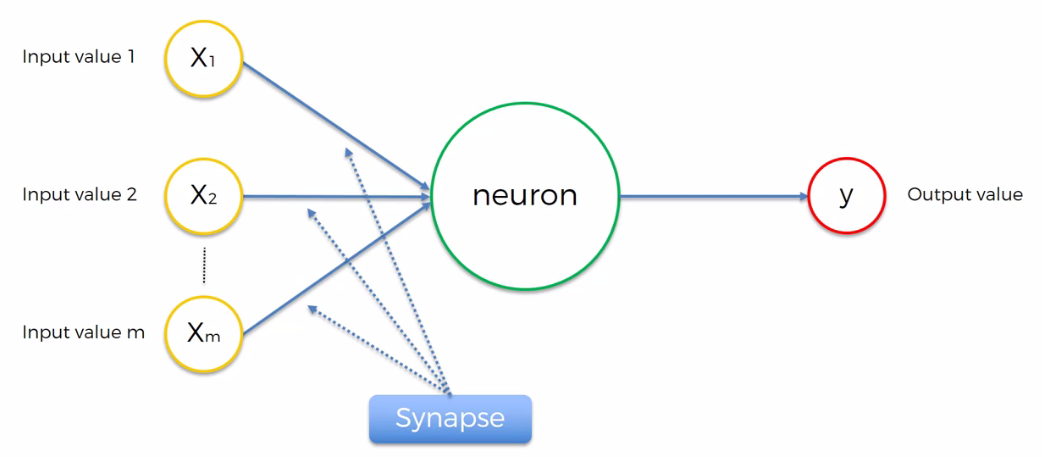


|  |  |
| --- | --- |
| * How do they work together: That's what the ***Dendrites*** and ***Axons***. * ***Dendrites*** are kind of like the ***receivers*** of the signal for the neuron and * ***Axon*** is the ***transmitter*** of the signal for the neuron. And here's an image of how it all works conceptually. * We can see *Dendrites* are connected to *Axons* of other neurons that are like even further away above it (*axon* doesn't actually touch the *dendrite*, it has been proven that there is no physical connection there). * And then the *signal* travels *down* its *axon* and connects or passes on to the *dendrites* of the next neuron and that's how they're connected. * Connection between the neurons are called the ***Synapse (neuronal junction)*** you can see over there in that little image that's figure bracket is a ***Synapse***. |  |

* And we will use those terminology in our Artificial Neural Networks (ANN). So instead of calling our *artificial neurons linking lines axons* or *dendrites* (whose connection it is, coming or going signals) we're going to call them just ***Synapses***.
* Because the important thing is where the signal is passed*doesn't matter who that element belongs to*. They're just a representation of the signal pass. So basically that's how a neuron works.
* How are we going to represent neuron in a machine: So here's our ***neuron*** also sometimes called the ***node***. The ***neuron*** gets some input signals and it has an output signal (Dendrites and Axons). We are call them ***Synapses***.



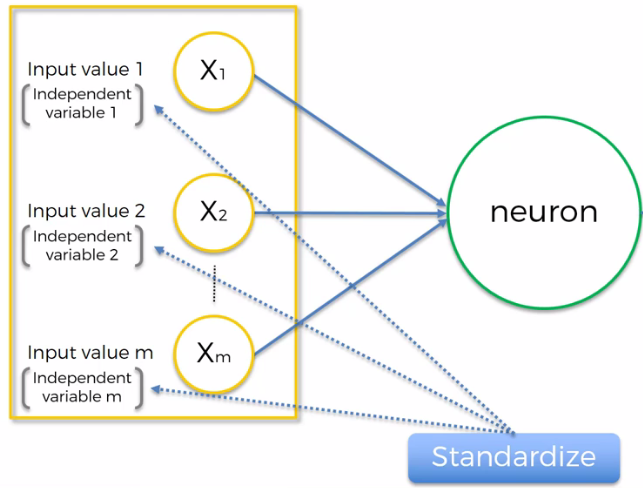
* These ***input*** ***signals***, we're going to present them of other ***neurons*** as well. So in this specific case you can see that this neuron is a Green neuron and is getting signals from Yellow neurons.
* Color indication: In this section we're stick to a certain color coding regime where yellow means an input layer (all of the neurons that are on the outer layer on the first front of where are the signals coming in).
* And by *signal*, we mean just basically *input* *values*. Like in a simple linear regression we have input values and then we have a predicted value. Same thing is in here.



* So we have *input* *values* as the Yellow ones and then on the right *output-value* will be Red. And *hidden* *layers* will be *Green*.
* In terms of the *input* *layer* the way to think about it as of the *human* *brain* the *input* *layer* is your *senses* i.e.whatever you can see, hear, feel, touch or smell.
* More simply your brain is sitting in your head (dark box) it cannot feel anything and inputs are coming as electric signals from the senses.
* So for *Humans*  it is our *Five-senses* as Input-layer and for our ANN it is just *input-data*.
* One other thing, here in this specific example we're looking at a *neuron* which is getting its signals from the *input layer (yellow) neurons*. Sometimes you'll have *neurons* which get their *signal* from other *hidden layer neurons* (i.e. from other *green* *neurons*) and the concept is going to be exactly the *same*. For simplicity we're portraying his example.

**8.1.3 How Neuron works in ANN : Terminologies**

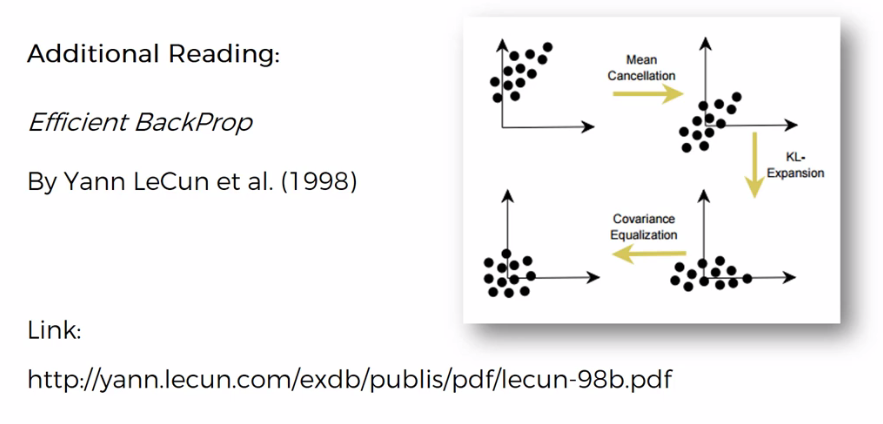
* Input-layers: In the input layer, *inputs* are in fact *independent* *variables*. These *independent variables* are all for *one single observation*. So think of it as just one row in your database (think your observation points as (a, b, c, d, . . . ) a multi-dimensional vectors).
* Eg: independent variables maybe age, deposits in the bank, drive/walk, salary etc. But that's all descriptors of one specific person (*one single observation*.) that can be either in your training model or prediction.



* Standardize/Normalize the variables: You need to standardize these variables so you have a ***mean of zero*** and a ***variance*** ***one***.
* Or you can also sometimes normalize them instead of standardize them. Meaning that instead of making a ***mean of zero*** and a ***variance*** ***one,*** you just ***subtract*** the ***minimum value*** from ***each element*** of a ***column*** and then ***divide*** by the (maximum – minimum) by the range of your values and therefore you get values ***between 0 and 1***.
* And it depends on this scenario you might want to do standardize or normalize.

Basically you want all of these variables to be quite similar because these values are going to go into a neural network and is going to be easier for the neural network to process them if they're all about the same.

If you want to read more about standardization/normalization and other things read following paper:*Yann LeCun* is a leading *Deep Learning* expert works at *Meta (facebook)* as Chief Scientist. And is close friend to *Geoffrey Hinton*. In this paper you'll learn more about ***centralization*** and ***normalization***.

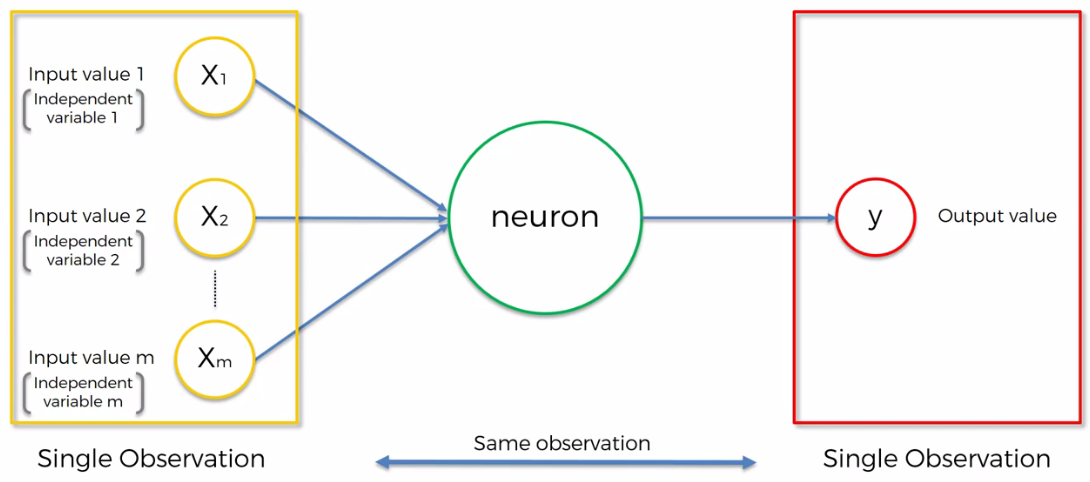


* Output values: These values can be different types. Output value can be:

1. ***Continuous***. For instance price it can be
2. ***Binary*** (yes/no). For instance a person will exit or will stay in a shop.
3. ***Categorical*** verbal.

In the case of categorical variable, your output value won't be just one, it'll be *several output values* because these will be a dummy variables which will be representing your categories.

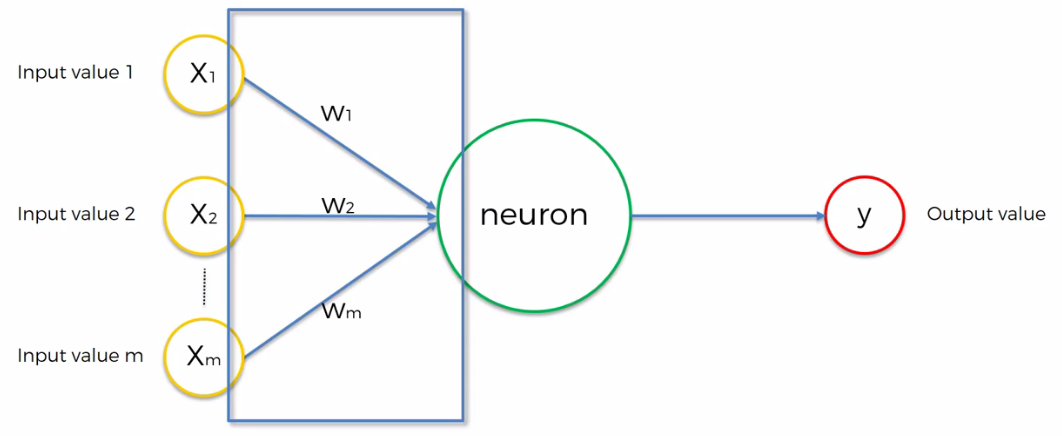
|  |  |
| --- | --- |
|  |  |



* All those things are happening to singe observation: Now consider simple case of one output value. Whatever ***inputs*** you putting in, that's for *one row* and then the ***output*** you get that is for that *same exact row*. Or if you're training your neural network then you're putting the inputs in for that one row you're getting the output in for that one specific row.
* For simplicity, you can think it like a simple (multivariate) regression. Same thing here it's nothing too complex. We're just putting in values we are getting output. But just remember that ***every time*** it's ***one*** ***row*** you're dealing with. *Don't get confused and start thinking that* ***these are different rows that you're putting into your artificial neural network****.*
* Synapses: The Synapses, they all actually get assigned with weights.

In short weights are crucial to ANN functioning. Because weights are how neural networks learn. By adjusting the ***weights*** the ***neural*** ***network*** *decides* in every *single case* what *signal* is *important* and what *signal*  is *not important* to certain ***neuron***.

* What *signal* gets passed along or not. Or what strength to what extent signals get passed along. So weights are crucial. They are the things that get adjusted through the process of learning.
* When you're training an ANN you're basically adjusting all of the weights in all of the Synapses across this whole neural network.
* And that's where Gradient Descent and Back Propagation come into play.



*Synapses assigned with weights*

* Neurons: Signals go into the neuron and a few things happen.

1. First thing and the first step is corresponding weights of variables will be added together i.e. the weighted sum of all of the input values.
2. Secondly an *activation* *function* is applied to this *weighted* *sum*.

* Basically *activation* *function* is a function that is *assigned* to a *neuron* or *to this* wholelayer *and it is applied to this* weightedsum.
* The ***activation function*** decides if it ***needs to pass on a signal*** to the **next neuron (or layer)**. i.e. depending on the ***activation function*** the Neuron will either pass on a signal it or it won't pass the signal on.

1. And that's exactly what happened here in step three. The ***neuron*** passes on that ***signal*** to the ***next neuron*** down the line.

|  |  |  |
| --- | --- | --- |
|  |  |  |

And that's how you got input values, weights. And what happens in neuron where you've got weighted sum and activation function applied on weighted sum that is passed on line and that is just *repeated throughout* the *whole neural network on and on and on* and depending on how big, how many neurons you have how many Synapses you have your neural network.

Notes:

* Normalization is a *data preparation technique* that is frequently used in machine learning. The process of *transforming the columns* in a dataset to the *same scale* is referred to as *normalization*. Every dataset *does not need* to be *normalized* for machine learning. It is only required when the ranges of characteristics are different.

1. Min-Max Scaling (Normalization): Subtract the ***minimum*** value from each column’s ***highest value*** and ***divide*** by the ***range***. Each new column has a ***minimum*** value of ***0*** and a ***maximum*** value of ***1***.
2. Standardization Scaling: The term “***standardization***” refers to the process of ***centering*** a variable at ***zero*** and ***standardizing*** the ***variance*** at ***one***. Subtracting the mean of each observation and then dividing by the standard deviation is the procedure:

* Normalization and standardization: Normalization and standardization are not the same things. ***Standardization***, interestingly, refers to ***setting the mean to zero*** and the ***standard deviation to one***. ***Normalization*** in machine learning is the process of ***translating*** data into the ***range [0, 1]*** (or any other range) or simply transforming data onto the unit sphere.
* EPOCH: An epoch is a term used in machine learning and indicates the *number of passes* of the entire *training* *dataset* the machine learning algorithm has completed. Datasets are usually grouped into *batches* (especially when the amount of data is very large). Some people use the *term* *iteration* loosely and refer to putting *one* *batch* through the model as *an* *iteration*.
* If the ***batch size*** is the whole training dataset then the number of epochs is the number of iterations. For practical reasons, this is usually not the case. Many models are created with ***more*** than one ***epoch***. The general relation where ***dataset size*** is ***d***, number of ***epochs*** is ***e***, number of ***iterations*** is ***i***, and ***batch size*** is ***b*** would be ***d\*e = i\*b***.

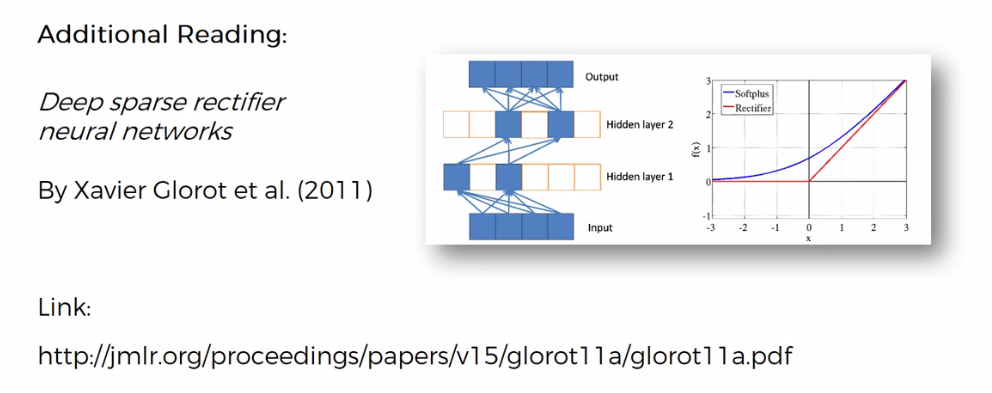
**8.1.4 The ACTIVATION FUNCTION**

|  |  |
| --- | --- |
| We talked about the *structure* of one *neuron*, it has some inputs values coming in. Synapses got some weights then it *adds* up the *weights* and then apply the *activation* *function*. In step 3, it passes on the signal to the next *neuron* (the decision of pass/no-pass comes from the *activation* *function*.). |  |

* Types of activation function: We're going to look at four different types of activation functions. Of course there are more different types of activation function but these are the popular ones.

|  |  |
| --- | --- |
| 1. Threshold function: On the ***X axis*** you have the ***weighted sum*** of ***inputs***. On the y axis, you have 0 to 1 scale. It's basically kind of, ***yes-no*** type of function.  * If the value is *less than zero* then the threshold function passes on ***0***. * If the value is *more than zero or equal to zero* then threshold function passes on a ***1***. |  |
| 1. Sigmoid Function: It's a function which is used in the *logistic* *regression*.  * The *benefit* of this function is that it is *smooth* (unlike the ***threshold*** ***function***). It's just nice and smooth gradual progression. So anything below ***0*** it doesn't suddenly goes to ***0***, actually it tends to ***0*** gradually. And it approximates towards ***1*** for *+ve* values. * *Sigmoid* *function* is very *useful* in the *final* *layer* (the *output* *layer*) of ANN. Especially when you're trying to *predict* *probabilities*. |  |
| 1. Rectifier Function: Rectifier Function is one of the most used functions in artificial neural networks - ANN. even though it has a kink is one of the  * It goes all the way to zero (for *–ve* values) and then from there it's *gradually progresses* as the *input value increases* (for +ve values). We use this function in this section. |  |
| 1. Hyperbolic Tangent Function: It's very similar to the *sigmoid* *function* but here the *hyperbolic tangent* function goes ***below 0***. So the values go from ***0*** to ***1*** or approximately to ***1*** and go from ***0*** to (–ve) ***-1*** on the other side. |  |

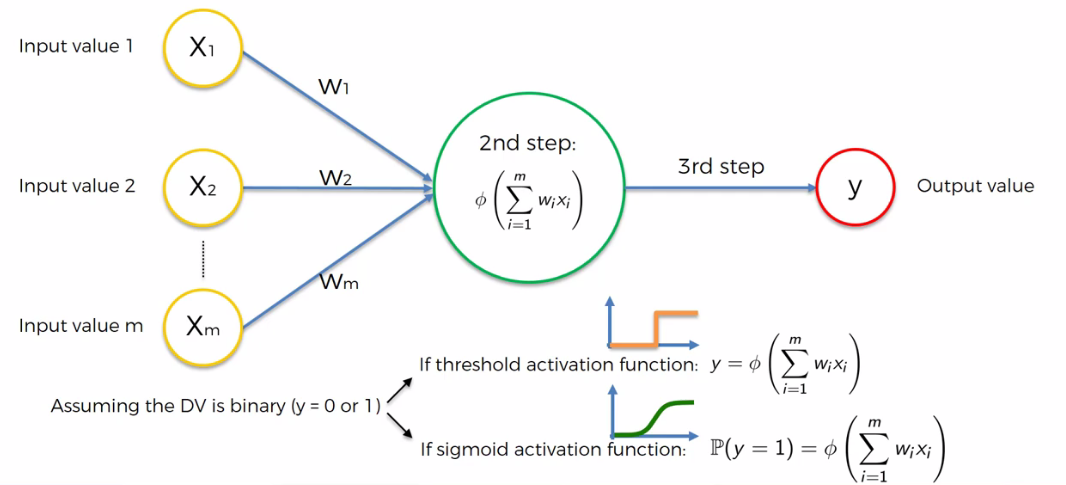
* Read more in the following paper: There you will find out exactly why the *rectifier* *function* is such a *valuable* *function*, why it's so *popularly* used.
* For now we *don't really need to know* all of those things. We're just going to start *applying* *them* which you start *using* them *more* *and* *more* . And so when you feel *comfortable* with the *practical* *side* of things then you can go and refer to this paper and then you will be able to *soak in that knowledge much quicker* and it will make *much* *more* *sense*.



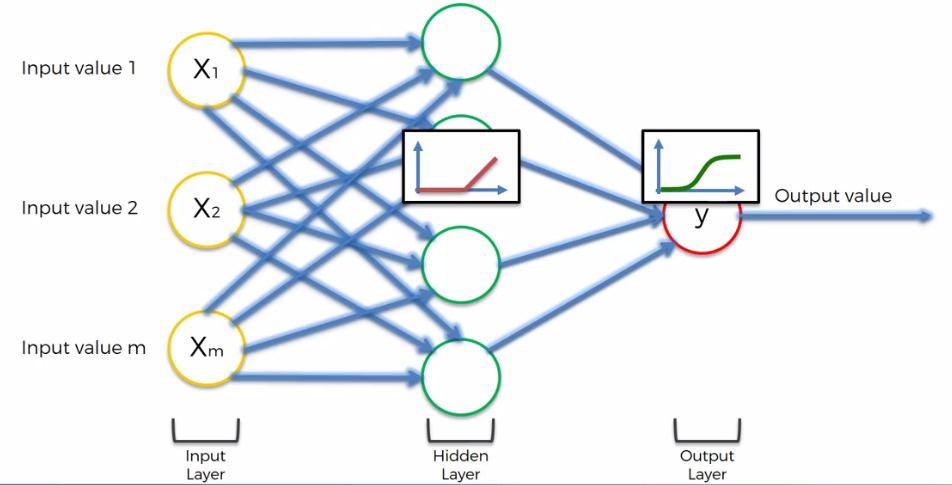
* How to apply different activation function: Which type of activation function is used in which layer.
* Example 5.1.1: We've got an example here of a ***neural network*** of just ***one*** ***neuron*** and an output layer. The question is ***assuming*** that your ***Dependent Variable (Output)*** is binary (either ***0*** or ***1***) which activation function would you use.
* There are two options: And those are just two examples. If you have a ***binary output (dependent) variable***.

1. Threshold Activation Function: Because we know that it's between ***0*** and ***1***. It fits perfectly to this requirement and therefore you could you say ***Y*** equals the ***threshold*** ***function*** of your ***weighted-sum*** and that's it.
2. Sigmoid Activation Function: It is actually between 0 and 1 just what we need. But at the same time you could use it as is the probability of ***Y*** being ***yes*** or ***no***.

* So we want ***Y*** to be ***0*** or ***1*** but instead we'll say that the Sigmoid Activation Function tells us of the probability of ***Y*** being ***equal*** to ***1***.
* That's very similar to the logistic regression approach.



* Example 5.1.2: Now let's have a look at another practical application. What about if we had in your all natural like follows?



* In the first layer we have some inputs. They are sent to our *first hidden layer* and then *an activation function* is applied. And usually we will apply a Rectifier Activation Function here.
* And then from there the signals would be passed on to the output layer where the Sigmoid Activation Function would be applied and that would be our ***final*** ***output*** (predict a probability).
* And this ***combination of two- activation function*** is going to be quite common where in the hidden layers we apply the ***rectifier function*** and then for output layer we apply the ***sigmoid*** ***function***.