Chapter 8 : Part 2

**Deep Learning**

**ANN: How NNs works**

How an Neural Network works and Learns

**8.2.1 How an NN works**

In an NN where *multiple neuron* present in a *hidden* *layer*, different neuron takes different decisions. Each neuron picks *different* *input* variables according to the *conditions* given to the *neurons*.

* Walk through an Example: We're going to be looking at a property evaluation problem. We're going to look at a *neural* *network* that takes in some *parameters* of our property and *evaluates* it.
* We are not going to train the network (a very important part in neural networks is training them up). We're going to pretend is *already trained up* and that will allow us to focus on the *application* *side*.
* Let's say we have some input parameters: *area* in square feet, *number of bedrooms*, *distance to the city* in Miles, *age of the property*. All of those four are going to comprise our *input layer*.
* There could be *more* *parameters*, now for *simplicity* *sake* we're going to look at just this *four* for now.

|  |  |
| --- | --- |
| * Most of the ML algorithms (regression/classification) that exist can be represented in this form and this is basically a *diagrammatic* *representation* of how you deal with. * This shows us how powerful NNs are. Even *without the hidden layers* we are ready. We have a representation that works for most other M L algorithms. |  |

* The basic form: It's very basic form of a *neural* *network*. It only has an *input* *layer* and an *output* *layer* and no *hidden* *layers*. Our output layer is the price that we're predicting.
* In this form these inputs variables would be *weighted* *up* by the *Synapses*, and then the output would be calculated. For instance the price could be calculated as the ***weighted sum*** of all of the inputs.
* Here we could use any of the *activation* *functions*: *sigmoid* or *threshold*.

|  |  |
| --- | --- |
| * The Hidden layers- The advantage of NNs: In neural networks we have hidden layers, which is an advantage that gives us lots of flexibility and power that increase the accuracy. * Now we're going to understand how that hidden layer gives us that extra power. We're going to walk through this example. Since we assume that this NN has already trained up, then we're just going to *walk step by step through* how the *neural network* will deal with the *input* *variables* and calculate in the *hidden* *layers* and then calculate the *output-layer*. * We've got *all four variables* on the left and we're going to start with the *top Neuron* on the *hidden layer*. |  |

* Not all variables are important for some neuron: Now we previously saw that all of the *neurons* from the *input* *layer* they have *synapses* connecting each one of them to the *top neuron* in the *hidden layer*.
* But those *synapses* have WEIGHTS. Now some weights will have a non-zero value and some weights will have zero value, because *not all* *inputs* will be *valid* or all inputs *won't be importan*t for *every* *single* *neuron*. Some inputs will not be important and neglected by some neurons.

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

* Here we can see two variables and the ***area*** and the ***distance to the city*** are important *for that first neuron* whereas ***bedrooms*** and ***age*** are not.
* We can explain as: The further away you get from the city the cheaper real estate becomes, hence the space in square feet of properties becomes larger. So for the same price you can get a larger property the further away you go from the city.
* And probably what this neuron is doing: it is looking for ***area*** variable which are not so far from the city but have a large area. So for their distance from the city they have an unfair area.
* So that neuron might be picking out those specific properties and it will activate the activation function only when the certain criteria is met. It performs on calculations inside itself and it combines those two variables ***area*** and the ***distance to the city*** and that contributes to the price in output.
* And therefore this first neuron doesn't really care about the variables ***bedrooms*** and ***age*** because it's focused on that specific thing.
* Now let's not even draw these lines for the synapses that are neglected.
* That's where the Power of the Neural Network comes from because you have *so many of these Neurons* each focusing on specific criteria.

|  |
| --- |
| * Let's take one in the middle neuron. Here we've got three parameters feeding ***Area***, ***Bedrooms*** and ***Age***. So what exactly that neuron is doing? Why this neuron through all of the thousands of examples of properties has found out that the ***Area***, ***Bedrooms*** and ***Age,*** combination of those parameters is important? * The reason could be: In the area/city data this model is trained, there are some people looking for *Larger properties* with *lots of bedrooms* and the *age of the property is low* (i.e. new property). Thos people could be *New couples with new jobs and better income* or could be *larger families with old parents and grandchild's*. The common thing about those people that, they don't care about the distance from the city. * Hence this specific neuron is looking for these three properties (variables), as soon as that criteria is met the neuron fires up and the *combination* of these three parameters *occurs*. |

* So this is the Power of the Neural Network, it *combines these* parameters *into a* brandnewparameter that helps with the *evaluation* of the *property*.

|  |  |
| --- | --- |
| * Let's look at another neuron, at the very bottom one, for instance this neuron could be picked up just one property: Age. The criteria behind can be: some properties are more valuable when it is too old. For example: a 100 year old property can be a historic place and some Elite/Rich Family want to buy it for "show off their friends". Hence this neuron only aims to the *Age* *variable*(property). * This can be perfect to apply *Rectifier Activation function*, because after certain *age* *limit*, the *value* of the property *gets* *higher*. * And also a neuron can consider only no. of bedrooms and distance. Another neuron can consider all of the variables. And so on. The pint is there can be so many options for the neurons. That’s the power of the NNs. |  |
|  |  |

|  |  |
| --- | --- |
| * So you can see that these *neurons* and this whole *hidden layer situation* allows you to *increase* the *flexibility* of your *neural network* and allows you to look for very specific things and then in combination they predict the price. * That's the power of NNs. Like an ant, by itself cannot build anything. But when you have 100000 ants they can build an Anthill together. And that's the situation here. * ***Each*** ***one*** of these ***neurons*** by itself ***cannot*** ***predict*** the ***price***. But together they have *super powers* and they *predict* *the* *price* and they can do quite an *accurate* job if trained *properly,* set up *properly*. |  |

**8.2.2 How an NN Learns**

There are two fundamentally different approaches to getting a program to do what you want it to do.

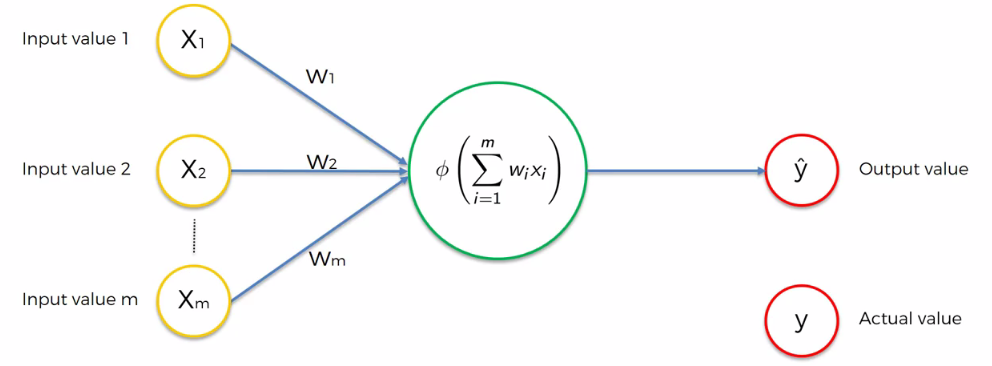
1. Hard coded source-code: where you actually tell the program's *specific rules* and what *outcomes* you want. Guide the program throughout the whole way and define all the *possible options* that the program has to deal with.
2. Neural Networks: On the other hand you have neural networks where you automate the program to be able to understand what it needs to do on its own. In this NN you provided inputs, tell it what you want as outputs and then you let it figure everything out on its own.

* Our goal is to create this network which learns on its own. We going to avoid trying to put in the rules.

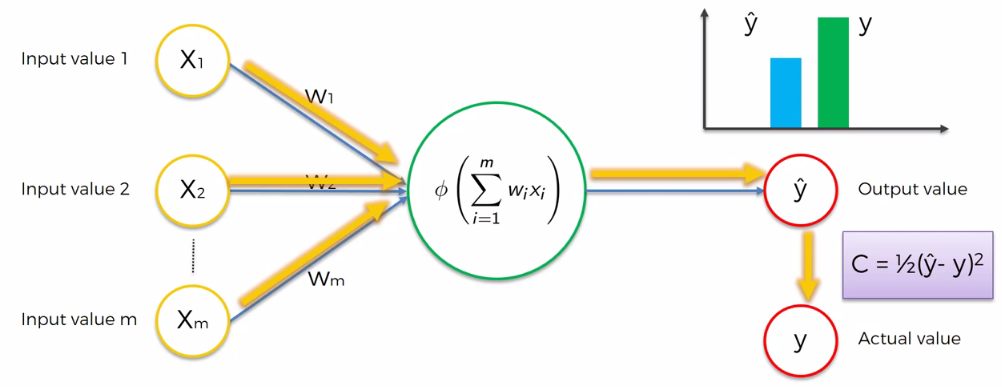


* For example: Distinguish between a Dog and Cat.
* Option 1: You would use hard-coded program using different characters: like the *cat's ears*, look out for *type of nose*, look out for *type of shape* or *colors* etc.
* Option 2: On the other hand for a neural network you just code the neural networks architecture and then you *point* the *neural network* at a *folder* with ***images*** of all these ***cats*** and ***dogs***, which are already categorized. From those images of cats and dogs NN going to learn by itself what a cat or dog looks like.
* Once *NN is trained up* then you give it a *new* *image* of a *cat* or *dog* it will be able to *understand* what it was.

|  |  |
| --- | --- |
| **8.2.3 PERCEPTRON**  Here we have a very *basic* *neural* *network* with a *one* *layer*. It is called a *perception*.  stands for the actual value  output value, it is predicted by the neural network.   * And the perception that was first invented in 1957 by ***Frank*** ***Rosenblatt*** and his whole idea was to create something that can actually learn and adjust itself. |  |



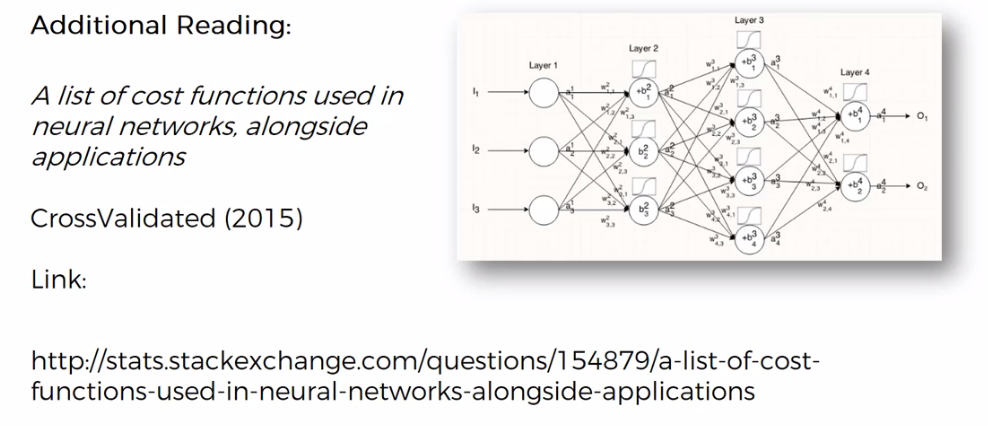
* Let's see how our perception learns: Say we have some *input* *values* that have been supplied to the *PERCEPTION* (basic NN).
* Then the *activation* function is *applied* and we get an *output.*
* Now we're going to plot the output value and actual value on a chart.



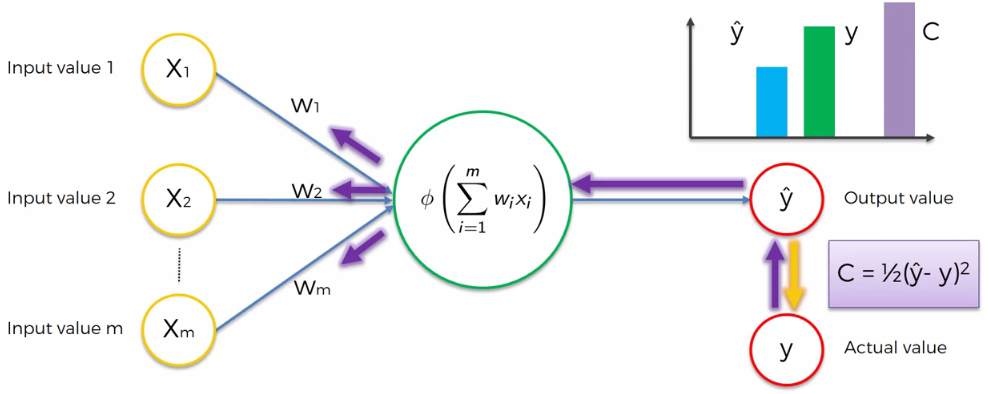
* To make our NN to be able to learn we need to *compare* the *output* *value* to the *actual* *value*. So we use a function called the ***cost function*** and is calculated as:

Now there are many ways of calculating ***cost function***. There are many different cost functions that you can use. We used the simplest form here.

* Basically the *cost* *function* is telling us about the *error* that we have in our *prediction*. And our goal is to minimize the cost function because the lower the *cost* *function* the closer *output* *value* and *actual* *value* .

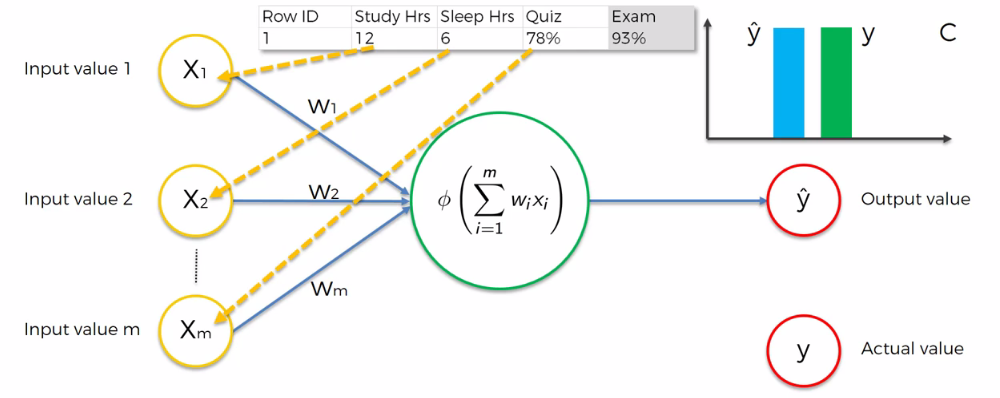


* Next step once we've compared and calculated the cost function, now we're going to feed this information back into the Neural Network. The information going back into the NN and the weights get updated.



**8.2.4 How an NN learns: Single Row of a Dataset**

* Example: Let’s consider following screenshot of the data. From a dataset, of a row where we have: these independent variables- how long you study, how long you sleep and your quiz score. We’re ***predicting*** the ***result*** you're going to get on an exam.



|  |  |
| --- | --- |
| * So for first iterartion, we input these input values (Study hr, Sleep Hr, Quiz) in to NN, then we get . Comparing to actual value , we get the cost value . |  |
| * Once we've compared and calculated the *cost* *function*, we're going to feed this information back into the NN and the weights get updated. |  |

|  |  |
| --- | --- |
| * So we feed these three values into the NN again for the second time (now the weights are updated) then we're going to be comparing the result to . | Iteration 2 |
| * *Cost* *function* is calculated again and *weights* are *adjusted* again. |  |

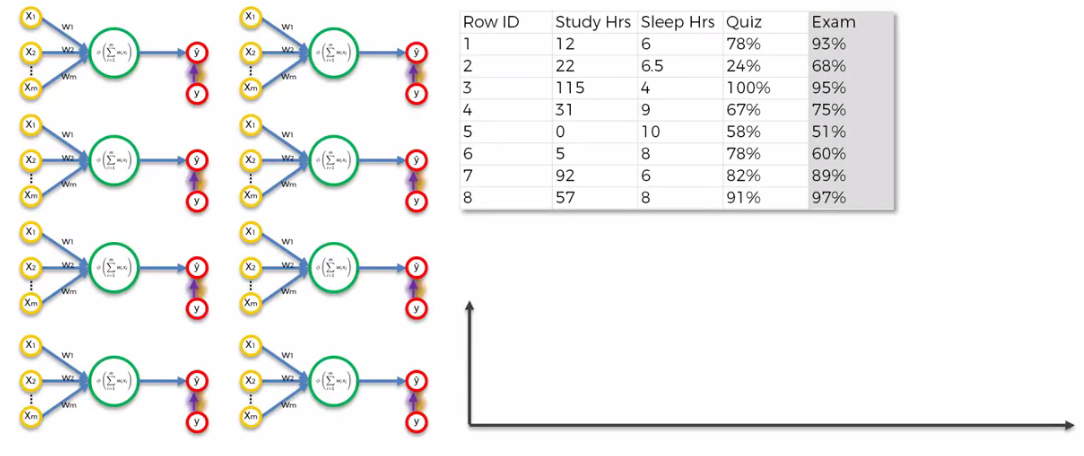
* We continue this iteration until cost-function, is minimum. That is and gets equal. Usually you won't get cost function equal to zero.
* Every time is changing because we've tweaked the weights. Hence *cost* *function* changing also. We get information, then feedback this information to the NN so that the *weights* get *adjusted* *again*.
* Here every time we feed in exactly that *same* *row* because just in this case we're *dealing* *with* *that* *one* *row* into our *neural* *network*.

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

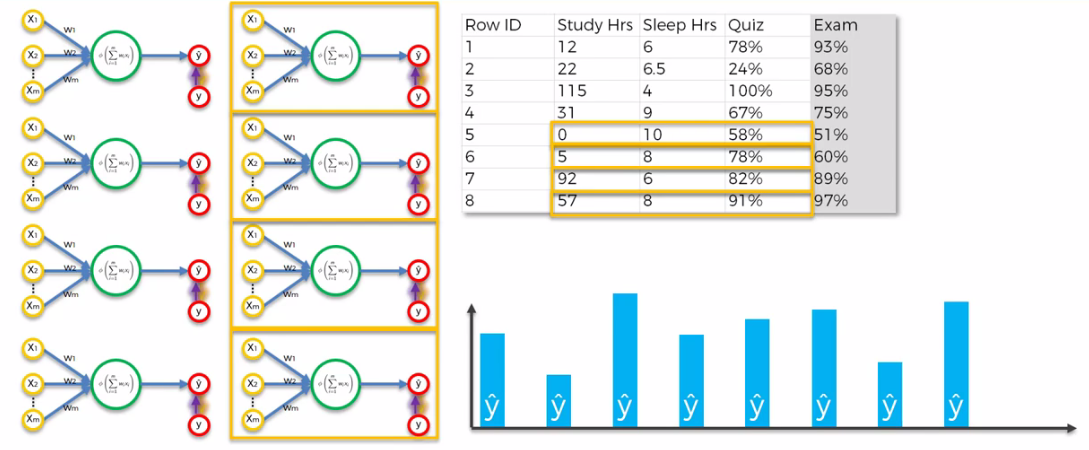
**8.2.5 How an NN learns: Multiple Rows of a Dataset**

|  |  |
| --- | --- |
| * Up until now we've been dealing with just that one row. So here's the full data set. We have eight rows of how long you study, how long you sleep and your quiz score. * And as you can see here on the left we've got eight of these PERCEPTRON. They are all the same PERCEPTRON so this is also important. * I just duplicated eight times for the learning purpose. It is the same NN, we're going to be feeding these data into. |  |

* Epoch: One epoch is when we go through a whole dataset and we train our neural network on all of these rows.

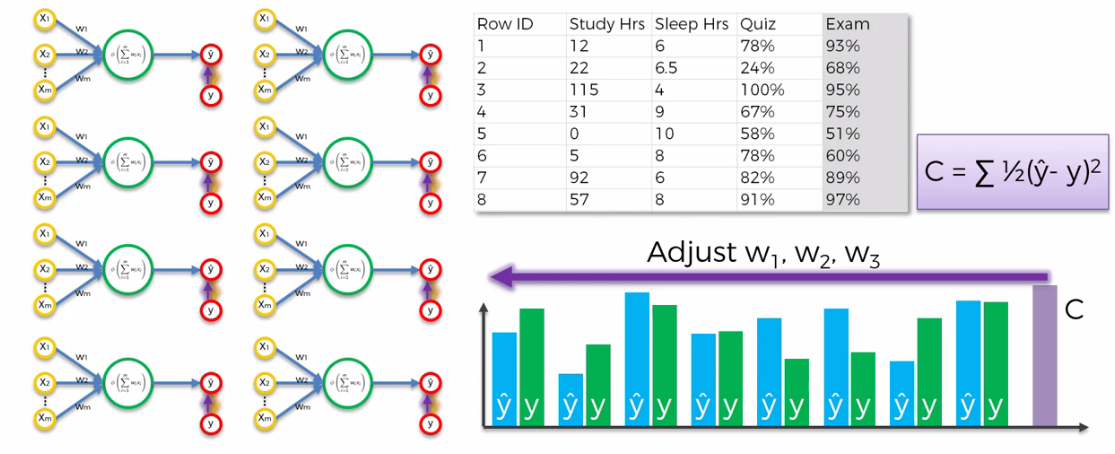


* For first iteration we input the rows one by one and get output for each row. In following graph we calculated outputs for all 8 rows.



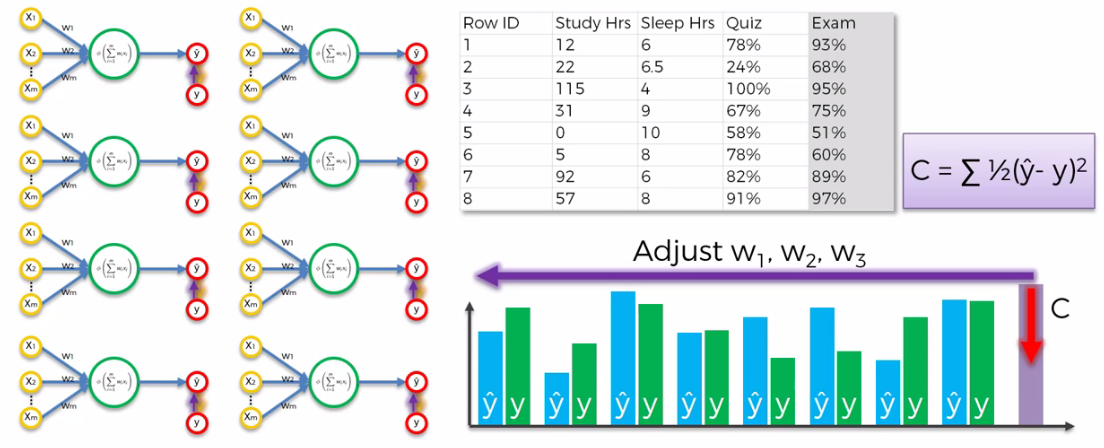
|  |
| --- |
| * Then we compare the *outputs* to the *actual* values. For every single row we have an actual value and corresponding output value. And now based on all of these differences between and we can calculate the cost function which is the sum of all of those squared differences between and . |

* Total Cost Function:
* After we have the *full cost function* we go back and we *update* the *weights*. So we’re going to *update* the weights in that *one* *neural* *network* (there are not eight of NNs there's just one NN) so basically the *weights* are going to be *the same for all of the rows*. All the rows share the same weights.



[So it's not the case that *each row has its own weight*.] Now that was just one iteration (we used all 8 rows, hence ***1 epoch*** is complete).

* Next we're going to run this whole *process* *again*. We're going to *feed every single row* into the *neural network* find out our *cost* *function* and adjust the weight of the Synapses again.
* We iterate until the *cost function is minimum (or stable)*. What we did for one row. But now we're going to be doing it for 8 rows.
* We have here ***8 rows*** but it could be ***800*** ***rows*** or ***eight*** ***thousand*** rows however many rows we have in our ***data-set***. You do this process and then you calculate the *cost* *function*.
* And the goal here is to *minimize* the *cost* *function*
* And as soon as you found a ***minimum cost function***, that is your *Final Neural Network* that means your *Weights* have been *Adjusted* and you have found the optimal weights for this dataset.
* You are done *training* your *dataset* and you're ready to the *testing* *phase* or *application* *phase*. And this whole process is called Back Propagation.



**8.2.6 Additional Reading : List Of Cost Functions**

* So some additional reading that you might want to do for the ***cost*** ***function*** and good article is located on ***"cross-validated"*** website. It's called "A List Of Cost Functions Used In Neural Networks Alongside Applications". You can just Google for that exact search term go to the website and read the article.
* It's actually got some good examples and application or use cases for different cost functions so if you're interested to learn more about cost functions Check out this article.

