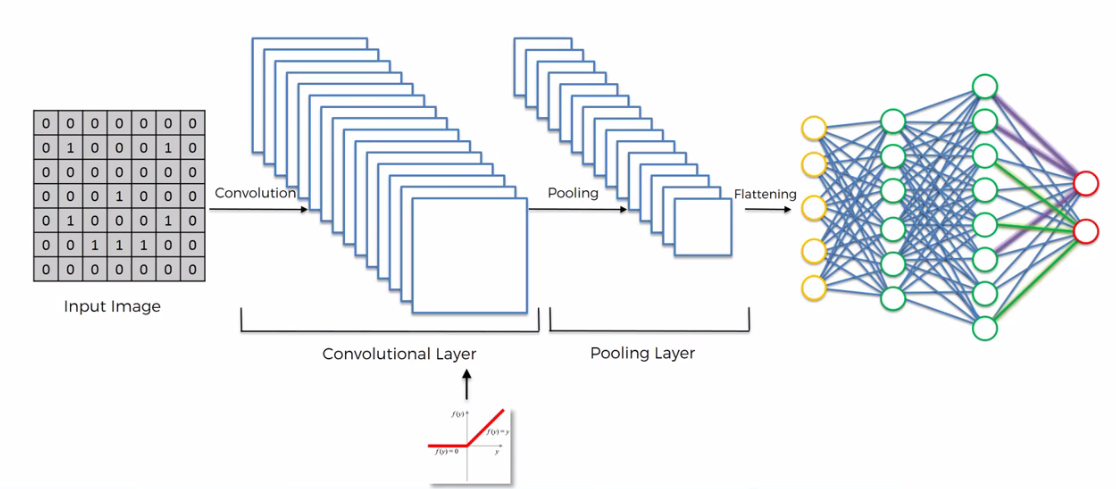
Chapter 9 : Part 4

**Deep Learning**

**CNN: Softmax and Cross-Entropy &**

**Summery**

**9.4.1 Summery**



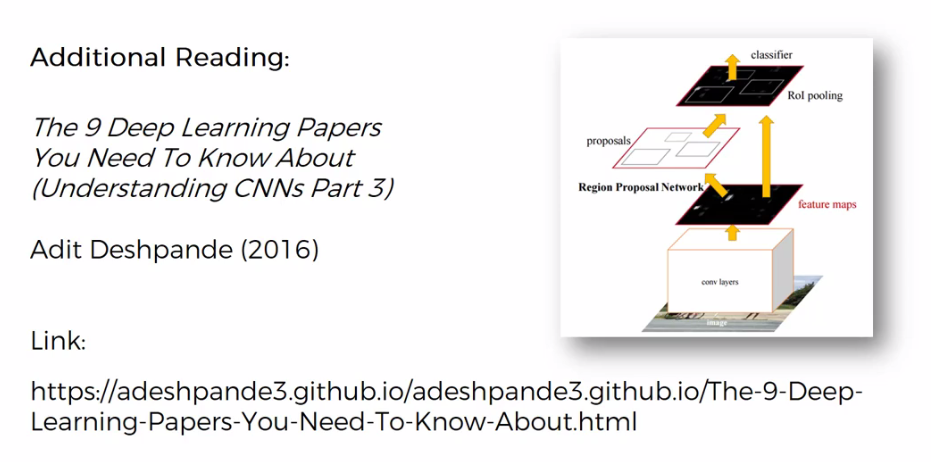
* Let's summarize what we've talked about.
* Step-1: We started with an *input* *image*. On the *input* *image* we applied multiple different *Feature-Detectors /Filters* to create these feature maps. And this is our ***Convolutional Layer***.
* Then on top of that ***Convolutional Layer*** we applied the ReLU to *remove* *linearity* or *increase non-linearity* in our images.
* Step-2: Then we applied a Pooling Layer to our ***Convolutional Layer***. From every single ***Feature-Map*** we created a Pooled-Feature-Map.
* The main purpose of the Pooling Layer is to make sure that we have Spatial Invariants in our images.
* So basically if an object in an image *tilts* or *twists* or is a *bit different* to the ideal scenario then we can still pick up that objects ***feature***
* Also Pooling significantly *reduces* *the* size of our images.
* Moreover ***pooling*** helps with ***avoiding*** any kind of ***overfitting*** of our data or overall by getting rid of a lot of that data.
* But at the same time ***Pooling*** preserves the ***main features*** that we're after just because ***Max pooling*** is used.
* Step-3: Then we ***flattened*** all of the ***pooled*** ***images*** into one ***long vector*** or ***column*** of all of these values and we put that into an ANN.
* Step-4: We then introduce a *Fully Connected Artificial Neural Network* where all of these features are processed through a NN.
* Then we have Final Fully Connected Layer which performs the ***voting*** towards the ***classes*** that we're after and then all of this is *trained* through a *forward propagation* and *back propagation* process. With lots of ***iterations*** and ***Epochs*** and in the end we have a very well defined ***neural*** ***network***.
* Another important thing is not only the ***weights*** are trained in ANN part but also the ***feature detectors/filters*** are trained.
* Both are adjusted in that same Gradient Decent process and that allows us to come up with the best feature maps.

In the end we get a fully trained CNN which can recognize images and classify them. That's how Convolutional Neural Networks work.

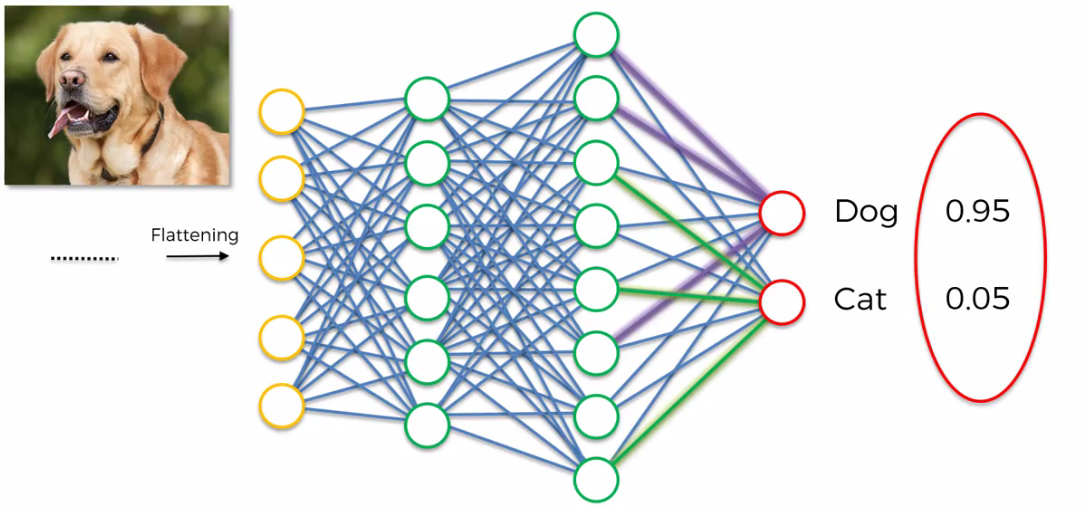
**9.4.2 Additional Reading on CNN**

If you'd like to do some additional reading then there's a great blog by ***Adit*** ***Deshpande*** from 2016. So the blog is called "The Nine deep learning papers you need to know about (understanding CNN's part 3)".

* This blog actually gives you a short overview of 9 different CNN's that have been created by people like you and others which you can then go ahead and study further.
* Just keep this blog in mind are these nine papers maybe after the ***practical tutorials*** or maybe after you do some ***additional training*** in the space of ***deep learning*** slowly you can then ***reference*** these works.
* You will get a lot of value by looking through other people's NNs and how they structured. Their CNN will help you understand ***what are the best practices*** and why people did certain things in a certain way and that will ***help you with your architecture of NN*** because ***ANN*** and ***CNN*** are not an exception.
* They are like an *architecture challenge*, you have to come up with an *idea* and then *structure* and then *adjust* it and *tweak* it to get the best possible design and the best possible and optimal performance.



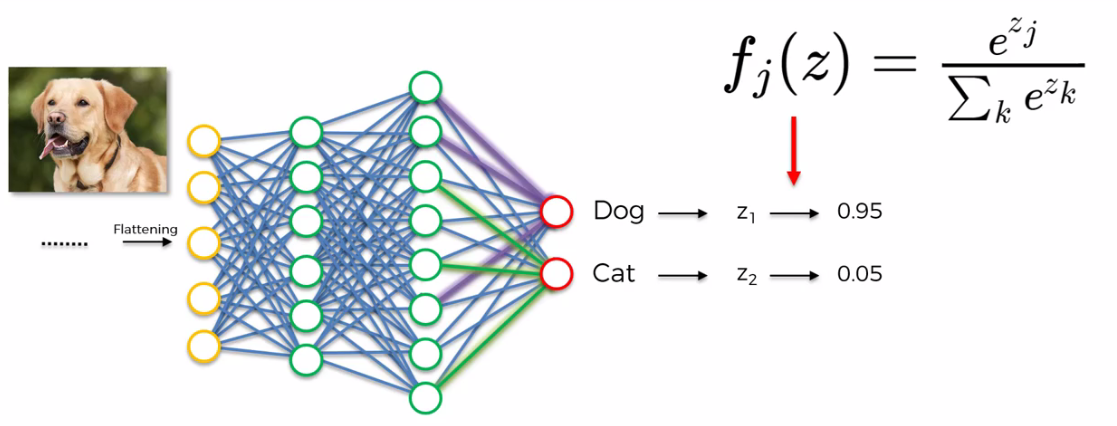
**9.4.3 SoftMax function**



* Here is the CNN that we built to Recognize Cat/Dog in a given image. Now notice the output ***probabilities*** ***0.05 for Cat*** and ***0.95 for Dog***.
* Now the question is: How the ***sum*** of these two values is ***1***. (in case of more than two class Dog, Cat & Bird this Sum of output probabilities is always 1).
* Because as far as we know from ANN there is nothing to say that these two *output-neurons* are connected between each other.

So how would they know about the value that other one is holding? And how would they know to add their probability-values up to 1.

* The answer is : They wouldn't in the classic version of our ANN.
* SoftMax: The only way that they would know about those values if we use a *special function* called the SoftMax function.
* So normally the Dog and the Cat Neurons would have any kind of real values that their sum don’t have to be 1. Say these values are and **.** And .
* Then we would apply the SoftMax function upon and so that the sum of the output becomes 1.

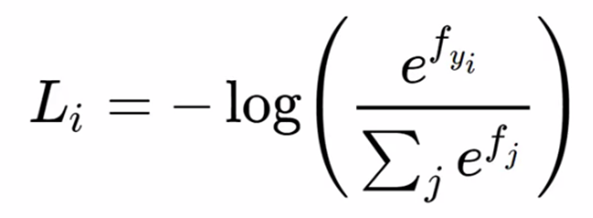


* Here the SoftMax function or the *Normalized Exponential Function* is a Generalization of the Logistic Function that Squashes a k-dimensional vector of arbitrary real values to a *k-dimensional vector of real values in the range of [0, 1]* that add up to 1. i.e. for example ***(23, 45, 79, 45)*** becomes something like ***(0.1, 0.25, 0.4, 0.25)***, it ***normalize*** the ***vector*** in a way so that the sum of the ***elements*** of the vector becomes ***1***.
* The *elements* of the *vector* in our case the *values* returned by *Cat-Neuron* and *Dog-neuron*. SoftMax function brings these values to be in range ***[0, 1]*** and make sure that they add up to ***1***.
* Why SoftMax function is used in CNN: It makes sense to introduce the SoftMax function into CNN because it is strange that a probability of being a dog is 95% and also probability of being a cat is 65%.

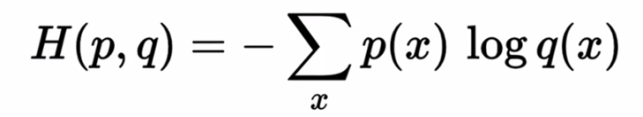
Therefore it's much better when you use SoftMax function to CNN

**9.4.4 Cross-Entropy function**

The ***SoftMax*** *function* is mostly used with the ***Cross Entropy*** *function*. The ***Cross Entropy*** *function*  looks like following:



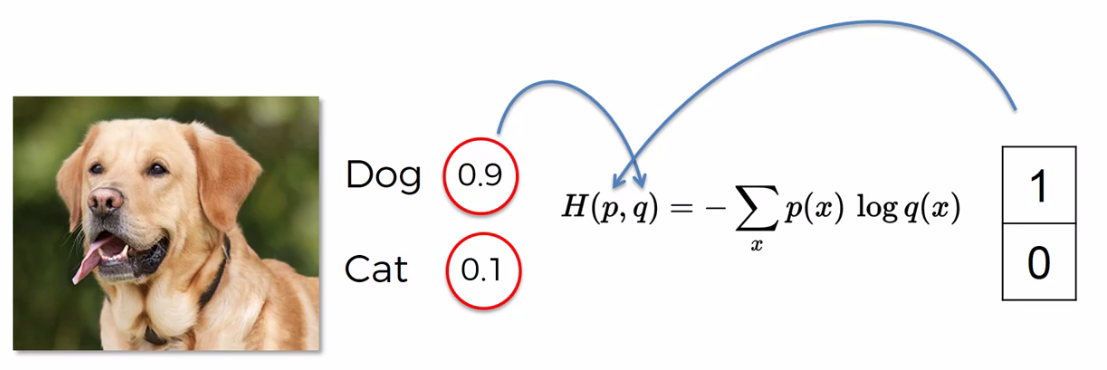
We're actually going to be using a different representation of the ***Cross Entropy*** *function*. that looks like follows:



The results are basically the same. This is just easier to calculate.

|  |  |
| --- | --- |
| * Cross Entropy function: In ANN we had a function called the *Mean Squared Error (MSE) Function* which we used as the Cost Function and our goal was to ***minimize*** the MSE in order to optimize our network performance. * Now in CNN we can still use MSE but after applying the ***Soft-Max function*** we better use *Cross Entropy function* because here "*log*" is applied and we can work with very *small numerical values*. * Note that, in *CNN* we call "*Loss function*" instead of "*Cost function*". "*Loss function*" are "*Cost function*" not same but they are very similar. Here we want to minimize *Loss function* in order to maximize the performance of our network. |  |

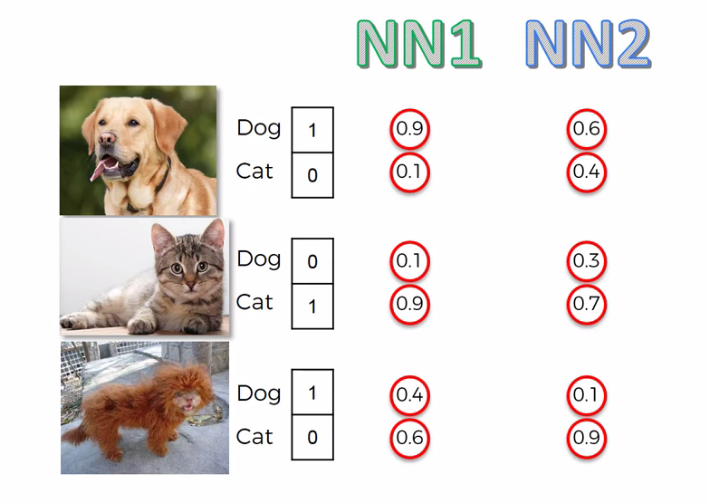
* Example: Let's see an example on how Cross Entropy function can be applied: Here ***p*** is the value of the range i.e. ***1*** or ***0***. And we put the *output-probability* into *q*. The figure shown above.



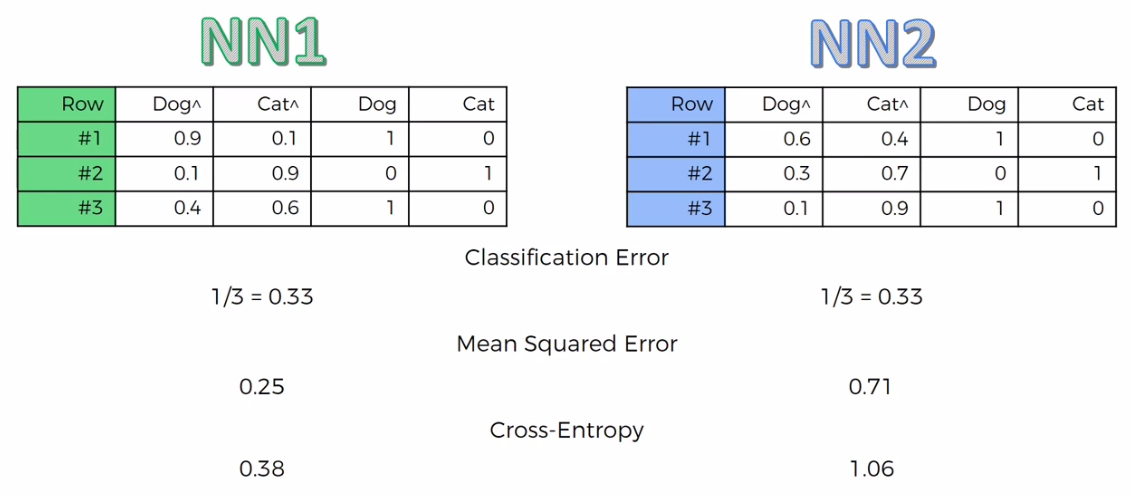
**9.4.5 Cross Entropy function calculates NNs performance more accurately**

Now we consider *two neural networks* and then we pass ***three*** ***images*** of a Dog, a ***Cat*** and another animal (which is actually another dog if you look closely, but difficult to recognize). Those NNs doesn't know abut those animals.

* In following figure the boxed ***0*** and ***1*** represents the ***actual*** ***values*** of being ***Cat/Dog***. So we want to see the performances of these following two NNs.
* Below the values inside the *"Red-Circle"* are the *predicted* *values* from these two NNs.



* So the key here is that even though both NNs got it wrong in the last case, but NN1 shows better performance than NN2. Because in the last case NN1 gave a dog 40% vote where NN2 gave 10% vote.
* Different functions to measure performance of NNs: Now we're going to look at the different functions that they can measure performance of NN1 and NN2
* In following tables: "***Dog^*** " and "***Cat^*** " columns represents the ***predicted*** values. And "***Dog***" and "***Cat***" columns represents the ***actual*** values.
* Notice that even though NN2 was correct for *first 2 trials* but its *performance* were *poor*.
* NN2 votes ***60%*** for Dog where NN1 votes ***90%*** in ***first trial***.
* Also NN2 votes ***70%*** for Cat where NN1 votes ***90%*** in ***second trial***.
* In the ***last trial*** both NN1, NN2 are incorrect, but NN1's performance was better in that trial. In this case NN1 gave a dog ***40%*** vote where NN2 gave ***10%*** vote.
* And so now let's see what kind errors we can calculate to *estimate* the *performance* and *monitor* the *performance of our NNs*.



* Classification error: It is basically just asking it "Did you get it right or not". So for both NNs we got *1 incorrect* out of *3 trials* (animals). So for both NNs we got ***1/3 = 0.33***. So in this case we cannot find the difference between the performance of the NNs (butt we know NN1 performed better in the last case).
* So in case of Classification error, both NNs perform at the same level (but we know that's not true.)
* That's why a classification error is not a good measure especially for the purposes of *Back Propagation*.
* Mean Square Error (MSE): Basically take the *sum of squared errors* and then just take the *average* across your *observations*.
* Here NN1 gets 25% error rate and NN2 gets 71% error rate (because even though NN2 was correct for first 2 trials but its performance were poor, also for 3rd trial NN2's performed poorly).
* So we can see *Mean Square Error* is more accurate than *Classification error*. MSE telling us that NN1 has a much lower error rate than NN2.
* Cross Entropy: Cross Entropy gives *error rate* ***0.38*** for ***NN1*** and ***1.06*** for ***NN2***. We can see that the results are a bit different.

**Why would you use Cross Entropy over MSE**

There's several advantages of Cross Entropy over MSE. For instance:

* If your *output* *value* is *too* *small* at the very ***start*** of your ***back propagation***. Then at the very START the ***Gradient*** in your ***Gradient-Decent*** will be very small and it *won't be enough* for our NN to *start* *adjusting* the *weights* and *propagating* in the *right* *direction*.
* But in case of ***Cross Entropy***, there is ***logarithm*** in it. It actually helps the NNs to work with a *very small error* to adjust *Weights* and *propagate* in the *right* *direction*.

i.e. for a little error decrease like ***0.0000001***, NN will ***adjust*** weights and Propagate. So *NNs* will detect *very tiny improvemen*t and works on it. This is possible because there is "Logarithm" in the *Cross-Entropy*.

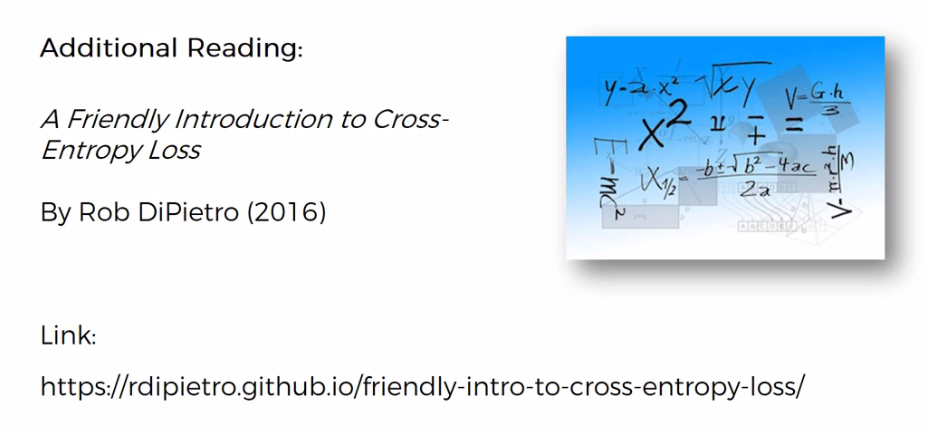
* So CROSS ENTROPY will help your *neural network* get to the *optimal state* more accurately than MSE. In CNN it is important because we applying ***SoftMax-function***, which normalizes results in between ***[0, 1]***, and hence we have to deal with *very small numbers*. Thus *Cross-Entropy* is a better option.
* Actually *Cross-Entropy* will improve your network significantly so that that jump from 1000000 to 1000 in MSE these jump will be very low. In that case MSE won't guide your gradient boosting process or your back propagation in the right direction.
* Even if MSE guide the NN into right direction, but it'll be like a very slow guidance it won't have enough power. But if you do *Cross-Entropy* , it will understand that even though these are *very small adjustments* and a ***tiny changes in absolute terms***, but in ***relative*** terms it's a ***huge improvement***. Because we have used SoftMax before.

Note:

* Important: It is important to note that, CROSS ENTROPY is only the preferred method for classification. But it we deal with Regression which we had in ANN then MSE is a Better option.
* ***Cross entropy*** is better for ***classification*** and ***MSE*** is better for ***regression***.

**9.4.6 Additional Reading**

* To know more about usage of ***Cross entropy*** over MSE, watch Geoffrey Hinton's *"The SoftMax output function"* video. He explains it very well there.
* If you'd like a light introduction into Cross Entropy, then a good article to check out is called *"A Friendly Introduction To Cross Entropy Loss"* by Rob DiPietro.



* If you'd like to dig into the mathematics behind Cross Entropy & SoftMax then check out an article by or a blog called: "*How To Implement A Neural Network Intermezzo 2* " (intermezzo means an intermediate thing).

