[Music]

Welcome to the third lecture the sixth week in the course machine learning, today we will talk about the model of a single neuron in ANN. We are now in the middle block of lectures which all handles the kind of neural network that we called feed-forward networks. We did study the perceptron which was a precursor, today we will study a single neuron as part of a multi-layer network and then finally we will look at the full multi-layer problem and the learning algorithms needed for that.

An artificial neural network has typically many neurons and several layers in contrast to the single neuron perceptron that we discussed in the earlier lecture. In this lecture we will study a single neuron but of the kind that builds up complex networks today. We will look at how the neuron performs in a forward feeding manner and how its input weights are updated in each cycle. We will use the same kind of examples as we used for the perceptron, even if we study the neuron in isolation it will perform in the same manner as part of the larger network An ANN network consists of units and connection between units. A is considered the predecessor of B and be the successor of B in such a network. The output of A is the input to B etc. The behavior of input-output units are special in the sense that they simply output external input, so to say without any process they are introduced to create a homogeneous model. A slight disclaimer about the learning function in this case, so when we will talk about the learning in here which is the Delta rule essentially that learning function is relevant for the single neuron case. In the multi-layer case we will have a neuron will be even the single neuron will be handled differently because the learning rule in that case will be the back propagation however.

At this point in time you have seen a model of a neuron many times, but obviously we will repeat it here, so you have a body of the neuron you have inputs you have weights, you have a summation function where you sum up in every phase, the product of the input multiplied by the weights. Then you have an output and that output depends on the sum is greater than zero in that case, the output is not just for sum but a function applied to that sum and that function we call activation function and we will come back to that in a minute, and that results in an output but we are also normally for supervised learning we have a target value for the output to compare it. So the difference between the target value and the output value is the basis for error estimate that we call E. And as you see the threshold of the neuron is handled as already described through an extra input to the neuron.(04:13)So the core computation of ANN which is also something you should be familiar with now, is that you assign weights to all the inputs you decide on a threshold, you remodel the threshold in the way we described in terms of an additional input, you can then sum, make the weighted sum of them of the inputs and then the output is decided as the activation function of that sum if the sum is greater than 0 otherwise put is 0.

That's what you heard the last few slide was more or less repetition for good or for bad. Now we come to some new parts. So the big difference between the neuron we talked about now and the perceptron is the introduction of the transfer function. As you can see here to the top left you can see a step function, so if you choose activation factor as being the step function you get the same functionality as the present. But there are all other choices and I'm now going to comment on some of the aspects you should think about when choosing the activation function. Sometimes the activation function is called a squashing function because for the reason that certain such function squashes or saturates values as a depth asymptotic end, obviously not all do but some do so therefore this name have come up. So if you look at the various aspects here, so one aspect is non-linearity. One can actually show that if you choose a nonlinear activation function then a two layer neural network can be proven to approximate a universal function. The identity activation function with just leaves the input value untouched doesn't satisfy that property, so actually by including this kind of non-linearity in our model we are able to handle also nonlinear problem situations. Also by choice of function we could introduce a finite range, and actually that kind of situations tend to be more stable and efficient. Another thing that is important is that this function we have is continuously differentiable, because you will see a little later here that the weight updating mechanism we have is actually based on a gradient based techniques which demands differentiability on components of the model. In some cases one can handle of course the situations where the segments are differentiable so it could be cases where there are some having singular points that can be handled in a special manner. Another property of normal function is monotonicity, so if the function is monotonic, you can also guarantee some nice properties. Also the in the same fashion also the properties of derivative are important for ensuring stability and efficiency. The special circumstance is near origin, so if the function we choose approximates the identity function near the origin, we can battle handle small numbers and as you can we will see later one of the big problems we have with all weight updating methods based on the gradient method and we have the problem of the vanishing gradient which means that if you have gradient that becomes very small then we came to a situation where the ways are not updated at all, so the weight updating gets stuck due to the gradients approaches zero.

So actually then the vanishing gradient problem is a difficult found especially in the gradient based learning methods which includes backpropagation. So the way it's received an update proportional to the partial derivative about the error function with respect to the current weight. So the problem is that then the gradient will be vanishingly small which actually effectively preventing the weight from changing as well. So the problem occurs because even if singular gradients or individual gradients are not too small in this technique these gradients are combined using the chain rule, so as a total result you may end up with a small value. So this problem has been known but it was even more highlighted pretty recently when people started to work with those common recurrent networks because recurrent networks tend to be much deeper and obviously the problems worsens because the chain rule will be applied many many many times. So this is one of the problems that exist for all these kinds of weight updating methods.

Let us now turn to the weight updating rule in this case called the Delta Learning rule. So this rule as has been said is based on the gradient descent method and an error message based on average square errors. So in this case the error measure is assumed to be the square of the difference between the target value and the output value divided by two. And the Delta rule is actually derived it's not the derived here, you will only see the outcome, but the starting point for a derivation is that you look at the derivation of the error with respect to the specific weight value you want to adjust, as you can see the convenience of the number two, the denominator on their measure because when we make this derivation the derivation of the square, the two numbers will cancel each other and we get a simpler formula. So the Delta rule becomes as follows, as so the new updated value is the old plus a learning rate parameter as I have understood right now that all these methods always have a learning rate parameter that may in a way damp the effect of the church (13:16) and it's up to decide whether it's one which is means no mechanic (13:25) or a value between zero or one. So the learning rate parameter multiplied with the difference between the target value and output value, times the derivative of the output function with respect to its argument which the argument is the sum the weighted sum of the inputs and then finally also multiplied by the input value for the specific connection. So the intuition here is of course that the part from making the update dependent on the difference between reference and output, and the fact that it could be affected by a damping learning rate parameter that you are the factors here to be explained and the reason why you introduced also the input value here is that you want to kind of normalize this with respect, so it shouldn't be so that you update the weight, it should be so that the input value in a self effects the updating. So by in including the input value you kind of normalize the effect among the different inputs. The derivative of the of the transfer function is there because the whole formula is derived through the derivation of the error with respect to the weight where were actually the transfer function is a component. In the linear case this derivative is one, so when you take the derivative of a linear function it's one, so therefore this factor disappears and you get a simplified function at the bottom.

So here a very simple example it's exactly the example that we used for the perceptron but you can see the difference we have chosen here this activation function at the top right and obviously the introduction of that function will affect the outcome on the result, so you see that Y here we can becomes 0.16, because as you see the function is linear in that region and then at the bottom you can see the updates of the of the weights because as you remember in the linear case we can use the simpler simplified formula and we do not have to care about the derivative, also it's assumed here that the learning rate is 1 also to simplify expressions at this moment. Ok so you also get another example here which is also the same as for the perceptron and you can see with three inputs and the result is shown in the tabular form. So this is the end of the lecture we have a look at the mechanism of single neuron and how weight updating can take place in only considering a single neuron. So now in the next lecture we look at full multi-layer the case and the learning record in that case is back propagations thank you for your attention goodbye.