[Music]

So welcome to this fifth lecture of the six week of the course in machine learning. The theme of this week is Recurrent Neural Networks, so looking at our map for this week you can see where we are we really in the last three lectures we looked into feed-forward networks with back propagation, and actually recurrent neural networks is an area which is clearing within that tradition. And the key point here is the problem with the networks we looked at up to now, not necessarily are very good at handling complex data sequences and temporal starters. So recurrent neural network have the ambition to remedy those weaknesses. Recurrent neural network is a class of artificial neural network which are able to handle sequences or complex data in space and time. This allows RNNs to exhibit temporal dynamic behaviors. So unlike feed-forward neural networks RNN have a memory, some kind of memory or persistent state that affects forthcoming computations. It should be observed that many RNN systems implements memory indirectly by unfolding of the network into time step using hidden units. So I can say this is a stateless fashion and we will come back to that in more detail. Only some RNN has explicit cell states which we call the Stateful fashion. So RNN can use their internal state independently on how others are that is implemented to process temporal sequential structures, and varying lengths of inputs and outputs. This makes them applicable two tasks such as handwriting recognition or speech recognition. So one could say that the if you look at the two big areas actually today of machine learning, one has to do with language and speech and one have to do with images. So one can say that RNN really contributes to make this kind of networks efficient in language and speech domain, while will come later when we look at convolutional networks we will say that they target at the goal to make a RNN efficient in the image analysis domain. So RNN performs the same task for every element of a sequence that's the reason for the name, recurrent and that's very important because if one should treat elements of a sequence differently that would create a very very complex machinery. So the elegance here is that you can handle elements in a sequence but you treat them in a similar manner. So finally our RNN have cycles in contrast to ANNs. RNN takes both the output of the network from the previous time step as input and uses the internal state from the previous time step as a starting point for the current time step. Obviously there are many different situations where we have more complex input then we have used to be handled in the ANN case, so you can see at the far left of this slide we can see the really clear one-to-one mapping. So we have one we are well defined objects, discrete objects that map to some output values or some output class. Okay so then to the right you

can see different more complex situation so once situation could be that travel there complex input object like an image with you really in order to handle that kind of objects you have to divide it up you have a single input but you need some ability to look into that object and divide it up in some fashion. So here so you can say the second case is you have a complex object and you have to map it on a certain number of values, of certain number of classes. So another case is you explicitly have a single and explicit sequences input could be a word, and you want to map this sequence of word on to something like an opinion (05:23) or whatever it could be. A third example is that you have a sequence of input like in a sentence in language and so for example machine relation you can say this sequence of what should be translated in another sequence. So that is another situation. So finally we have the more complex case you have many inputs, you have many outputs could be happening for example in in video analysis and applications like that.

RNN has received substantial interest in the neural network world and can also boost many success stories in particular in the context of applications in language and speech. As a consequence of its popularity the area has many alternative lines are development that are totally trivial to follow. So it is not so easy always to understand the limits of this sub area, therefore we will start to discuss what we call a Vanilla RNN the single layered RNN that can be unrolled and replaced with strictly feed forward acyclic neural network. Requirement for the Vanilla is that it but time can be discretized which in turn requires that the duration effect of single neuron activities are finite, also can finite Impulse Recurrent Network. But a network that lacks this property is called an infinite recurrent network but obviously we will focus on the finite type. So after looking into Vanilla RNN and we will look at two natural extensions of that, so one extension is to allow dependencies not only backward in time but also forward. The vanilla RNN enables us to at the later stage to look at earlier states were actually at earlier elements of the sequence we study, but we could also reduce mechanisms that we could look forward in time, so that's what we do in the bi-directional case. But you can also we can also stack RNNs, you know having multi-layer systems and one reason for that is that if we look at an application like in language we have letters and letters for word is a sequence of letters but you can have a sentence with sequence of words and then we can say you have a paragraph that is a sequence of sentences. So actually when you got a long sequence on language input you don't only want to look at sequence on one level you may want to look at all these sequences on the various levels of abstraction. So therefore one can think of that you have one RNN neuron on each of these levels and obviously the layers can

be literally involved as for the single layered case. So necessarily these two extension doesn't destroy the nice property of the vanilla RNN it can be unfolded into a strictly field work. Furthermore the RNN does not solve, the vanilla ones does not solve but rather makes problems like vanishing gradient or exploding gradient even worse. Vanilla RNN also has a problem to handle a very long sequences, of course many times due to that long sequences rate very deep networks and deep networks can open spawns. So we will look into one attempt to handle these problems Short Long-Term Memory as SLTM which introduced a more complex machinery its neuron, essentially by having the purpose of getting better control of the signal processing in and among the units. So finally what is worth mentioning is that there are some architectures that are classified as RNNs but are not able to handle dynamically handle input sequences, however these systems have cycles and they have internal memory so that's the reason for including them. An example you know such is an associative memory architecture like a Hopfield network we also will discuss separately later.

Let's now look into what it means to unfold a vanilla RNN. So we consider the case we have multiples time steps of input, multiple times that of internal state and multiple times that's our outputs, which actually happens when we need to be there when we handle a sequence. So we can unfold the original RNN which has a cycle into a graph without any cycles. So this means that we in a way duplicate, we create copies of their unit where the cycle is removed, so the output from the first instance goes into the second, the input of the second goes into the third and so on and eventually output in each stage then of course could be fed upwards from any of the states to two other Network structures. So we can see that the output Y an internal state "U" from the previous step are passed onto the network as input for processing in the next time step. So biases also of course is still there I mean biases is as a instinct inherited from the general concept of neural network but we will this consider that for the moment because it would just block the site. So a key thing here is that the network does not change between the unfolded time step, the same weights are used for each time step is only the outputs and internal states that differ. So exactly what we do is replicate the exact structure without modifications. One can also remodel so as you see in the middle here when the most natural thing is to view this on the same level but then you make copies for each instance of the sequence or instance in time but one can also reshape that so graphically one can say that what this does is that it's add to the depth of the network, so that the copies is that going our layer in our sample they are layered in a in a vertical way but with this is very just different ways of depicting it. The important thing here is that by this transformation it is done in a

great way ensures that the result of the unfolding becomes just a normal feed-forward Network. Of course by unfolding and times if we have a long sequence we create a very deep feed-forward Network one player for each input sequence.

On this slide you can simply see another example how have the unfolding that can looks like but you can input in RNN for every instance you get output them from every instance but you also feed the internals of the internal state from instance to instance. The consequence of unfolding for the learning process as follows, so if you carefully do what we described in the last slide and you actually get a straightforward feed-forward Network, it's potentially possible to still update the weights that means to learn through back propagation. So in the back propagation of error will given time step depends on the activation of the network at the prior time step, so error can be propagated back to the first input times that of the sequence of that error gradient can be calculated the weights of the network can be updated. So with only marginal modifications we can talk about then back propagation through time, and it doesn't matter whether we backpropagate through the normal layers of the network or we backpropagate through those unrolled levels of the RNN, I mean of course there are subtlety here, so for a recurrent network the loss function depends on the activation of the hidden layer not only through its influence on the output layer but also through its influence on the hidden layer at the next steps. So small complications are more or less the same procedures can be applied.

The most natural extension of the vanilla RNN is to introduce many levels of RNN or stacked RNN. Obviously each layer in the stack can be individually unfolded as for the single layer case. The main reason having many layers is that the layers correspond to different levels of abstraction or aggregation for the sequential and temporal data items. As an example in word processing the first layer can model sequences of characters, the second layer level sequences of words, the third level sequences of sentences etc. The risk by increasing number of layers of course is to make problems like vanishing gradient even worse but we still because we unfold as you see I mean every layer is still considered to be possible to unfold, so this means that the resulting structure is still then a conventional feed-forward network, but with some of its problems increased of course. The next extension is what is called bi-directional recurrent neural networks. So by introduction of that the output layer can get information from past and future states simultaneously, means both backwards and forward links are enabled. So the principle in realizing that is to split the neurons of a regular RNN in two directions one for positive time direction for States and another for negative time direction. So one can use

there's a second step of unfolding, so you have the normal unfolding to handle the sequences but in order to being able to look both forward and backward for every already unfolded sequence you duplicate that with one copy for the forward direction and the other for the backward direction. So actually the these two hidden layers are can both be connected to the same output and they are independent of each other, that doesn't mean that one direction feeds something into the other, so they can be treated separately and be used to give information to later levels one by one. Of course a natural extension now is to combine a stacking of RNN by the bidirectional model, so nothing prevents that even bidirectional recurrent networks can be can be stacked. So obviously to sum up we are still working with a scenario here where everything we talked about can be unfolded and the net result of everything we have done would still be before feed forward network, which is a certain beauty about it.

So let's now go back a little and look at some of the challenges for neural networks feed-forward ones and in particular also for RNN. So there are three kinds of problems the vanishing gradient problem, we talked about that already and the problem there as we already said is that the gradient become too small and the reason for becoming a so small is that the individual derivatives or gradients are combined using the chain rule and the chain rule is applied one time in every step so if you're many steps and you start with the gradient in range of zero to one. Very rapidly they approach zero and the effect of that is that the update of weights are stuck. The opposite problem is that you can also get very large error gradients and this kind of large updates have the effect that the network can become very unstable. So there are ways of handling that too and in many cases it's many times more difficult to defend oneself against the vanishing gradient problem. Also one issue it is the very long sequences and temporal dependencies, of course they are not a direct problem but they indirectly create a problem in particular in both problems of vanishing gradient because they create very deep networks, and these very deep networks are than hampered by the other mentioned performance. Long short term memory networks LSTM are extensions of recurrent neural networks which basically extends their memory function. The core of the approach is to elaborate the interior of a vanilla RNN and unit with the purpose to increase control of signal flows. LSTM was explicitly designed to combat the vanishing and long-term dependency problems LSTM was introduced by Hochreiter and Schmidhuber in 1997 and the unit's of LSTM are used as building blocks for the layers of an RNN which is then often than as a whole called LSTM network. So LSTMs enable RNN to remember their inputs over a longer

period of time this is because LSTM contains their information in a memory that is much like the memory of a computer because general computer because the LSTM can read/write it information from its memory. LSTM's are widely recognized academically as well as commercially extensively used for speech and language processing by companies like Google Apple Microsoft and Amazon.

So hopefully this slide explains the difference between an RNN vanilla unit and a LSTM unit. So actually you can see that apart from the normal signal processing functionality of the RNN neuron the LSTM neuron has a much more complicated interior and essentially the difference is that the LSTM unit apart from anything else has some gating functionality that controls the signal flow.

So the LSTM memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information typically by opening the gates or not based on the important is a science to the information at hand. The assigning of importance happens through weights which are also learned by the algorithms. It simply means that it learns over time which information is important which it's not. Specific to LSTM is the cell state manifested by the horizontal line running through the top of the diagram of the unit. This is manifested by the different "C", "C" for cell state. Also in a LSTM unit you have three gates this case determined whether or not to let new input in input gate, delete information because it is an important forget gate or let it impact the output that the current step output gate. So that's the basic functionality of this kind of unit.

So some comments on structural of aspects of LSTM. So in principle LSTM's can ask for vanilla RNN be unfolded, be stacked in a multi-layer structure, be arranged in a bi-directional fashion all these are possible. If this is done in a stateless fashion that no cell state is use, that is the state information just flows across. Then the resulting network is the normal feed-forward network potentially with backpropagation, I mean so however if the LSTM cells use explicit cell states stateful fashion it becomes more clear and there is no guarantee that exactly this feed-forward Network me can be used as in the default state. But nothing prevents to combine these elements in more complex structures as emission here. So what we have done in this lecture is to look into some of the more important aspects of RNN, so we looked at we call oh we call the Vanilla type that Full or standard RM that can be unfolded. We also have talked about how that can be extended with will multiple layers with bi-directional links and we then also showed that one approach well known approach how the

signal processing can be improved in the LSTM case. There are so many different various in this area that and below in this line you can see a few of them, many networks that are classified that there are RNNs and there are so many approaches moreorless these different approaches relates to the simple ones so for example a gated recurrent unit GRU is a simple version of LSTM. Elman networks are very early kind of standard of RNN's with other simple structures local networks will be described in another context of associative memory and so on. But it's outside the scope of this lecture to go in more details of all this I would say genre of different approaches developed. So this was the end of this lecture thanks for your attention, the next lecture 6.6 will be on the topic Hebbian learning and associated memory. thank you