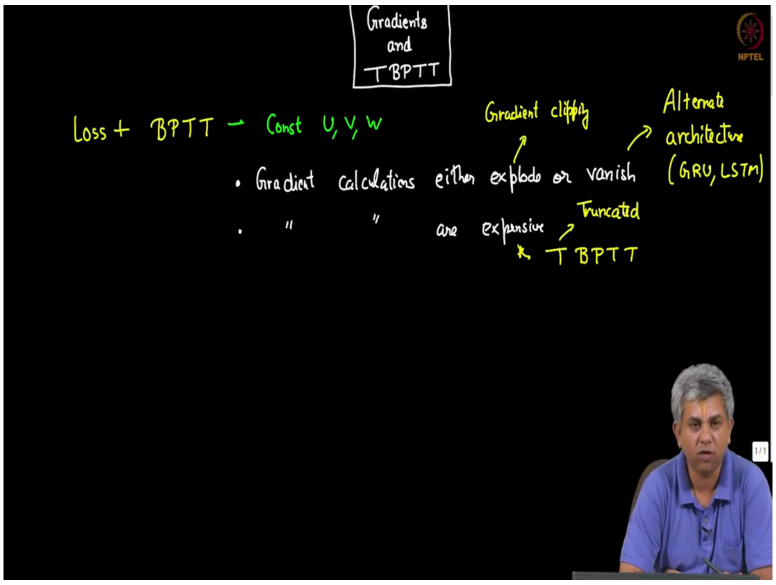
Machine Learning for Engineering and Science Applications
Professor Dr Balaji Srinivasan
Department of Mechanical Engineering
Indian Institute of Technology, Madras
Vanishing Gradients and TBPTT

(Refer Slide Time: 0:32)



Gradients and TBPTT

Loss + BPTT → Const U, V, W

- Gradient calculations either explode or vanish
- " " are expensive

Gradient clipping

Alternate architecture (GRU, LSTM)

Truncated TBPTT

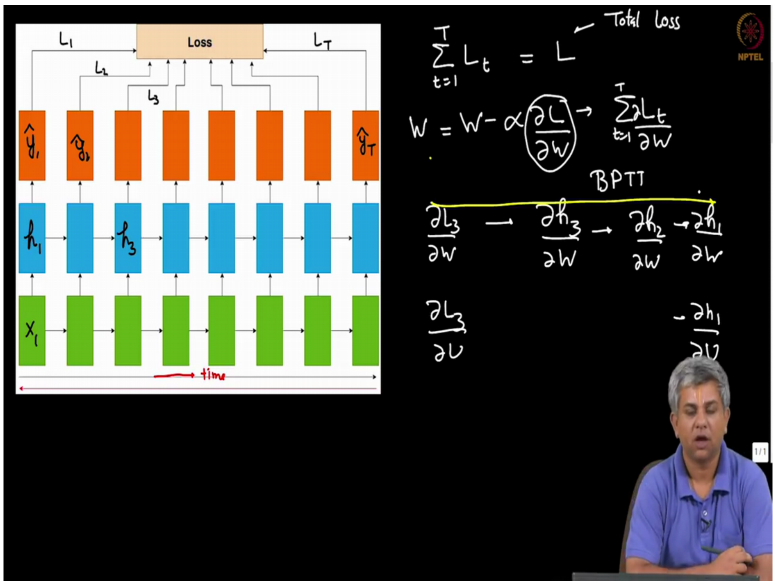
Welcome back. In the last video you saw back propagation through time, which we called BPTT, we also found out how to write the loss function. In this video we will see that what issues come out of defining the loss function this way and what issues come out of trying to do back propagation through time with a constant W. Okay, remember that the basic issue because of which we have to do back propagation through time was because U, V, W, our matrices were constants across time, because of which you had sort of recursive expressions for the loss with respect to W and the loss with respect to U.

Okay, in this video I will tell you what the 2 issues are, which come up. So the 2 main issues that come up is gradient calculations either explode or vanish. So now both of these can happen. Both of these are not ideal. Now, the other thing is the gradient calculations are expensive. You will see that each of these issues has a different solution. The solution for exploding gradients is something called gradient clipping. The solution for vanishing gradients is alternative architectures, specifically in the next few videos not this one, we will see 2 of the alternate architectures called GRU and LSTM.

As I had said earlier, you should see them optional methods for RNN's, just like you had specific architectures in CNN's, such as Le Net, Alex Net, etc. And finally for expensive gradient calculations or expensive back propagation, we have something called truncated back propagation through time, T stands for truncated. So these are the 3 issues that we will be looking at within this video or at least we will see the origins of these and I will give you a brief introduction to truncated back propagation through time. Now all of these are very the context of you getting a flavour of how all RNN's differ from ANN's and CNN's.

There are of course like everything else in this course, there are a lot of details within this which we will not be able to cover within this course. This is basically an overview course and we will try to give you an overview of this, so that you can understand some issues when they come up, when you try to train RNN.

(Refer Slide Time: 3:14)



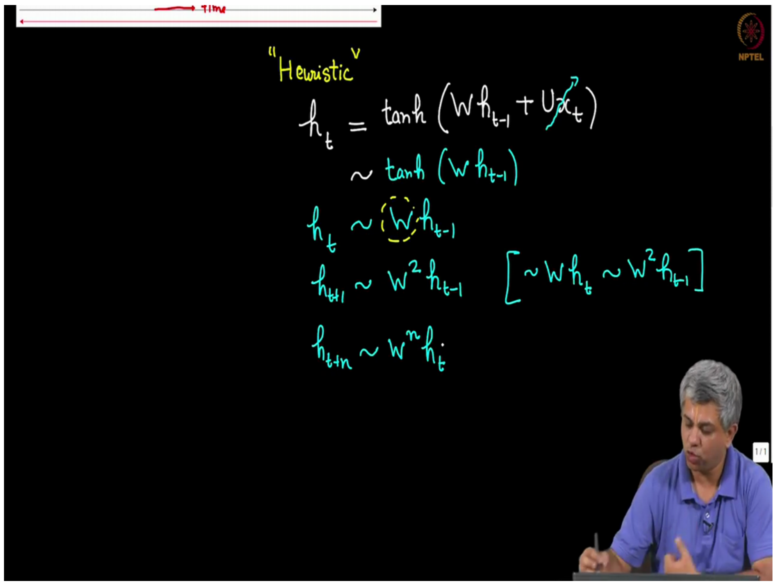
$$\sum_{t=1}^{T} L_t = L \leftarrow \text{Total loss}$$

$$W = W - \alpha \frac{\partial L}{\partial W} \rightarrow \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

BPTT

$$\frac{\partial L_3}{\partial W} \rightarrow \frac{\partial h_3}{\partial W} \rightarrow \frac{\partial h_2}{\partial W} \rightarrow \frac{\partial h_1}{\partial W}$$

$$\frac{\partial L_3}{\partial U} \quad - \frac{\partial h_1}{\partial U}$$



"Heuristic"

$$h_t = \tanh(W h_{t-1} + U \cancel{x_t})$$

$$\sim \tanh(W h_{t-1})$$

$$h_t \sim W h_{t-1}$$

$$h_{t+1} \sim W^2 h_{t-1} \quad [\sim W h_t \sim W^2 h_{t-1}]$$

$$h_{t+n} \sim W^n h_t$$

So let us look at a typical RNN with, let us say this is the time axis, as before this is the input, this is the hidden layer, sorry, this is the hidden layer and this is the output. Let us say this is X1 H1, Y1 hat, then this is L1, this is Y2 hat, this is L2, so on and so forth, let us say if this is YT hat, this is LT and we sum all these up, as we saw in the previous video, Sigma of LT from T equal to 1 to capital T is our total loss, okay. For gradient cancellations, remember when we do something like W matrix is W - Alpha times Del L del W, if you are doing simple variant descent.

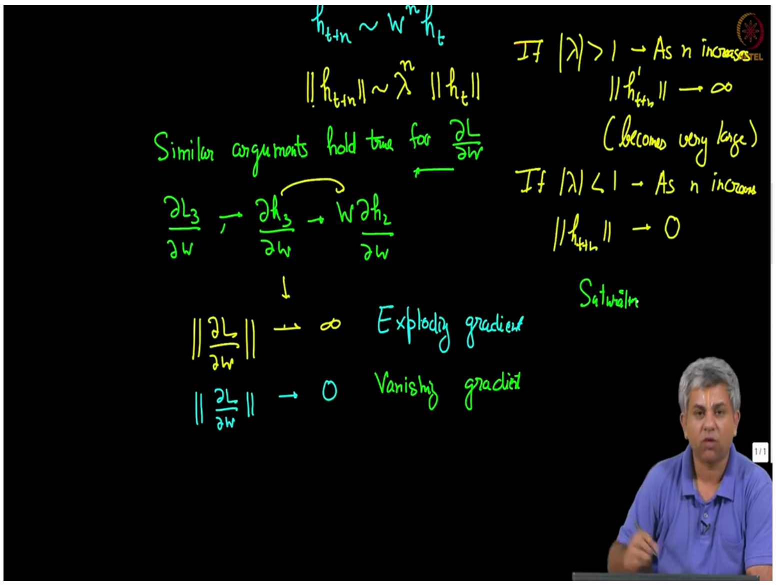
This term has to be calculated as Sigma of LT. Further we saw that you cannot simply calculate, let us say if I have L3, I cannot simply calculate L3 del W in the usual way, it involves something like del H3 del W and we saw that del H3 del W in turn involves del H2 del W, which involves del H1 del W. We saw that this is true for del L3 del U also. This is basically what we call back propagation through time, because none of these terms is independent. Now this kind of dependency creates several problems. So let us look at a heuristic description.

Okay, I want to point this out, this is not exact, this is not exact way of rigorous mathematics, but this is heuristic. Heuristic means rough, okay, that is a sort of a handwaving argument, it does go through mathematically also but that is well be on the scope of this course. We will just give you an intuition for why this kind of thing causes a problem, okay. Now note that when you have something like HT, our general expression was HT is tanh of W times HT -1 + U times XT.

Now I am going to say a few things, so please notice this, this goes approximately, assume that you can somehow ignore XT, you cannot but let us say for now that you can ignore it. So then this goes as tanh W HT -1. Let me further ignore this tanh, then I will say HT goes as W HT -1. What does this go that mean, order of magnitude, okay. So you are trying to guess how much is HT affected by HT -1. So you can see that HT scales as W times HT -1.

Long long ago, in week 1, we saw ideas of something called eigenvalues, eigenvectors, etc. Now I am going to make a further approximation very shortly but please do recall that. So, if HT is W times HT -1, then HT +1 would be WT, W Square times HT -1. Why? This will go as W of HT which goes as W Square of HT -1. So, in general HT +1 n will go as W power n +1 times HT -1. Just to be clear, I will use this as W power n HT. So, the weight matrix as you go through time, okay, so the weight matrix keeps on constantly multiplying. So H3 would be so-called W Square times H1 and if I have let say something like H5, that will be W power 4 times H1 so on and so forth.

(Refer Slide Time: 8:02)



$$h_{t+n} \sim W^n h_t$$

$$\|h_{t+n}\| \sim \lambda^n \|h_t\|$$

Similar arguments hold true for $\frac{\partial L}{\partial W}$

$$\frac{\partial L_3}{\partial W} \rightarrow \frac{\partial h_3}{\partial W} \rightarrow W\frac{\partial h_2}{\partial W}$$

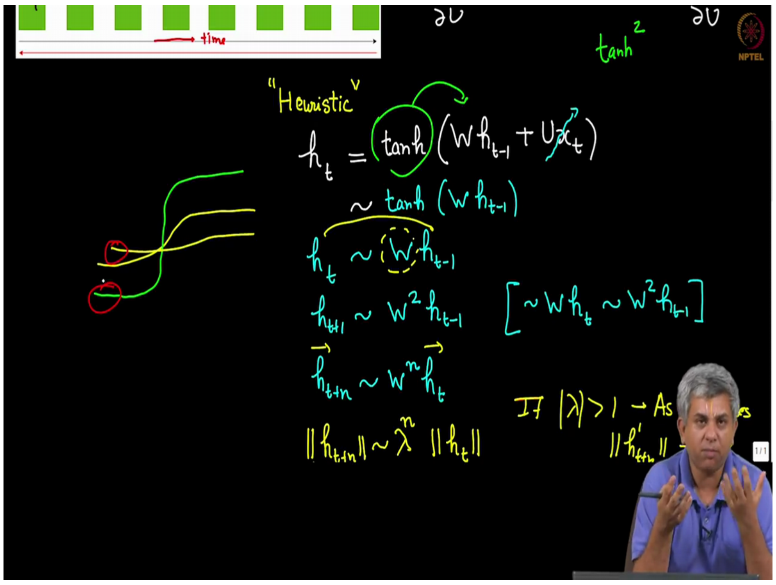
$\left\|\frac{\partial L}{\partial W}\right\| \rightarrow \infty$ Exploding gradient

$\left\|\frac{\partial L}{\partial W}\right\| \rightarrow 0$ Vanishing gradient

If $|\lambda| > 1 \rightarrow$ As n increases $\|h'_{t+n}\| \rightarrow \infty$ (becomes very large)

If $|\lambda| < 1 \rightarrow$ As n increases $\|h_{t+n}\| \rightarrow 0$

Saturation



"Heuristic"

$$h_t = \boxed{\tanh}(W h_{t-1} + U\cancel{x_t})$$

$$\sim \tanh(W h_{t-1})$$

$$h_t \sim W h_{t-1}$$

$$h_{t+1} \sim W^2 h_{t-1} \quad [\sim W h_t \sim W^2 h_{t-1}]$$

$$\vec{h}_{t+n} \sim W^n \vec{h}_t$$

$$\|h_{t+n}\| \sim \lambda^n \|h_t\|$$

If $|\lambda| > 1 \rightarrow$ As

$\|h'_{t+n}\|$

$\tanh^2$

Now suppose, again all these are heuristic arguments but it is already the remarkably good approximations, unfortunately I cannot show you this within the scope of this course. So, but if I have norm of HT + n, notice HT in the vector, remember H is simply a vector. But if I take its norm, which now, let us say 2 norm, it does not vector which norm it is, it will be some factor times norm of HT. Remember norm is a scaler, so this is a number, you are trying to find out the size of HT + n, that will be some number times HT.

And it turns out that it scales approximately as the eigenvalues of W to the power n. Another way to see this is volume that W is diagonal, I have W is diagonal, all it will have, W power n will simply be, all its eigenvalues or all its diagonal terms to the power n. Now which eigenvalue, we will see shortly. It will either be the largest or the smallest. So the worst-case scenario is this will be the largest, the best case scenario or the smallest case scenario is if this is the smallest.

Now what all this argument, you know, we have made about 6 or 7 approximations here, but all this is supposed to show is that there is a scaling going on here. As long as I use the same W, okay, as long as I use the same W, which I do for RNN's, throughout time, what happens is these vectors constantly get larger in magnitude or constantly get smaller in magnitude. And like the example we took in an earlier video of the sensor case, if you have a large number of times sequences, I called layers but you can see it as a time sequence.

Okay, so if you have a large number of time steps, this number, I benefit a small, you know, for example even if it adds to 1.01, over time it is going to get to be a huge number. This is the power with, this is the power of the exponential function or of the power function. So, this, if lambda is greater than 1 or modify is greater than 1, then as n increases, norm of HT + n tends to infinity, okay. Tends to infinity means more appropriately becomes very large, okay. All this remember is simply heuristic. Similarly if lambda is less than 1, this makes as n increases, norm of HT + n actually tends to 0.

Now this is simply for H, you can show that and I would request you to try this out by looking at the expressions in BPTT, the similar arguments hold true for del L del W also. That is to say, if you remember, if I look at del L3 del W, the expression in the last video, this depended on Del HP del W. And if you look up the expression, Del H3 del W will look something like W times Del H2 del W. There will be other terms there but this will be a primary term that is sitting there, you will have Sigma prime, etc.

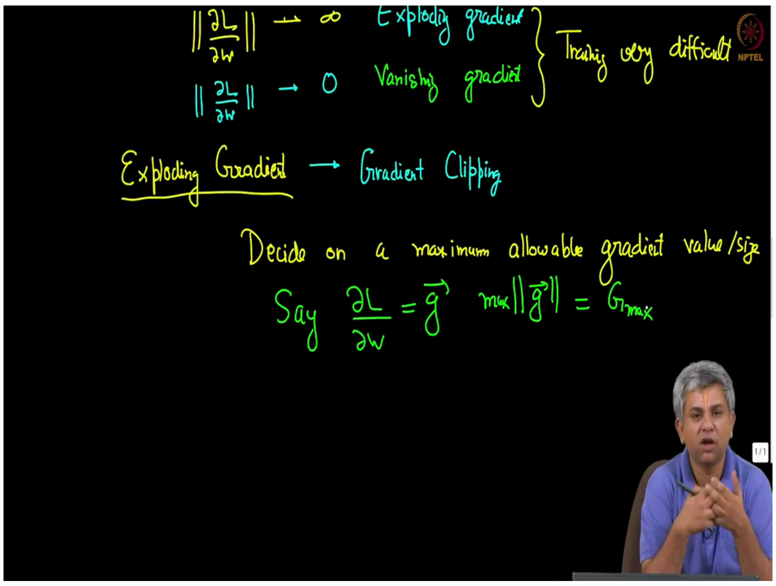
But it depends this way, so that sense, this is very similar to this relationship, except that it backwards in time. So if I look at the gradient of this, this will be W times the gradient of this, so on and so forth. So if you back propagate in time, you are going to get a similar exponential effect that too. So, this case of either the gradient growing, if you have something like del L del W Increasing or tending to infinity, this is called a exploding gradient. And why is this gradient exploding? Simply because the same operation repeated continuously, okay, back and back in time can actually lead to large numbers.

Similarly, Del L Del W tending to 0 is called a vanishing gradient. Now why are these problems? These problems because obviously you are never going to get exactly infinity because you are still dealing with finite number. But recall our discussion in week 3, we have finite precision. If you have finite precision and you have numbers that are growing ever larger, at some point you are going to, in some sense if you actually did the calculation by hand, you will have a large gradient and if you will move to an entirely different space, if you could actually calculate it.

But the problem is the moment it goes about the largest number that your machine can calculate, it will actually show you NAN, not a number or it will show you infinity, so on and so forth. So really speaking, file appreciation machines cannot handle exploding gradient. Similarly U will never actually go to 0. If you do like 0.99 power 1000, it will be very very very small number. But the promise it might actually becomes smaller than 10 power -16, which is the smallest number that you can represent accurately.

So at that point you will no longer train, so that will be called saturation, as we have said several times before. So you will get a very small gradient and that is practically gone. There is another problem, which sits here, which I will not discuss in detail but you notice this tanh, even the tanh is being repeated multiple times, okay. Do you have HT is tanh of W HT -1, HT +1 will be tanh of HT, so you have tanh square, similarly you will have tanh cube.

(Refer Slide Time: 14:58)



$\left\|\frac{\partial L}{\partial w}\right\| \to \infty$ Exploding gradient

$\left\|\frac{\partial L}{\partial w}\right\| \to 0$ Vanishing gradient

} Training very difficult

Exploding Gradient → Gradient Clipping

Decide on a maximum allowable gradient value/size

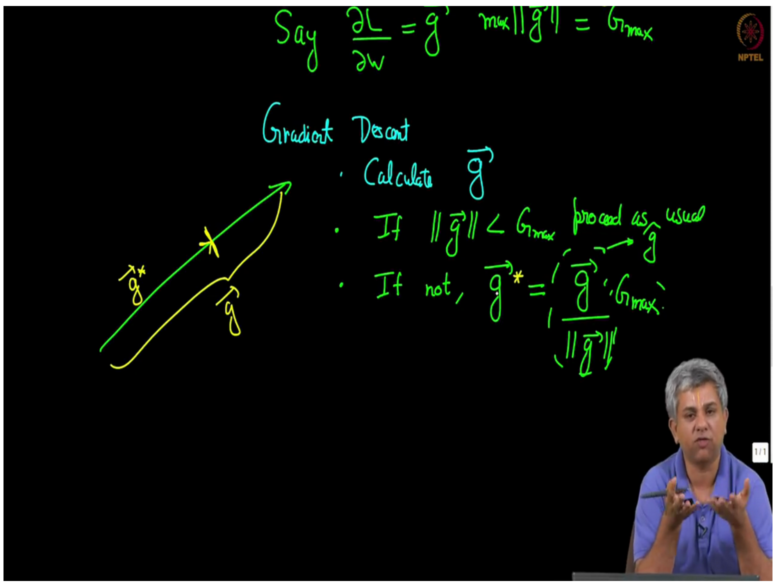
Say $\frac{\partial L}{\partial w} = \vec{g}$ $\max\|\vec{g}\| = G_{max}$

Now remember tanh itself looks like this, tanh square will look like this, tanh cube will look even flatter. And if you take tanh power 100, it will look even smaller and notice in all these cases, gradients become flatter and flatter and flatter and they become very small. No, all these problems put together lead to these 2 issues. The tanh, repeated tanh problem will lead only to the vanishing gradient issue but large number of players can either lead to exploding gradient or it can lead to vanishing gradient, both of these make training very difficult.

Why does it make training very difficult? Because we train through gradients, that is really how we train. So because we train through gradients, either exploding gradient or vanishing gradient is a problem, okay. Now how do we handle this? As I have said before, if I look at exploding gradient, it turns out that very sort of simple hack works really well. This trick is called gradient clipping. This is sort of a numerical hack. It is very simple, we decide on a maximum allowable gradient size.

What do I mean by value of gradient? Again remember, gradient is a vector, so you cannot give it a value, you can however give a value to norm of gradient. So let us say we are dealing with Del L del W, let me call it G vector. So I will say that maximum value allowable of G vector is some G Max. You will decide it, okay, you will decide on what you are comfortable with. Just like our cut-off criterion, this is an arbitrary criterion set by, it is sort of an engineering solution to the problem, okay.

(Refer Slide Time: 16:40)


Say $\frac{\partial L}{\partial w} = \vec{g}$ $\max \|\vec{g}\| = G_{max}$

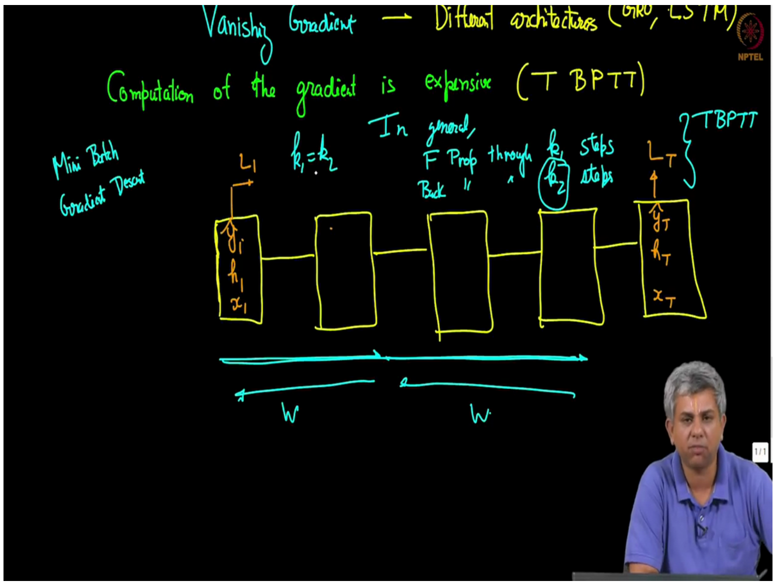
Gradient Descent

- Calculate $\vec{g}$
- If $\|\vec{g}\| < G_{max}$ proceed as usual $\vec{g}$
- If not, $\vec{g}^* = \frac{\vec{g} \cdot G_{max}}{\|\vec{g}\|}$

So, how does your algorithm looks? So you are doing gradient descent let us say. While doing gradient descent you calculate G. Remember she is del L del W, it could be del L del U also similarly for all the values. Now suppose you find out, you calculate norm of G, F norm of G is less than G Max, you proceed as usual. If not, then what do we do, we hold onto the gradient direction, we define a new gradient which is equal to the old gradient. What does this do? Remember this is simply the unit vector in that direction and you multiply it by G Max.

So what this says is my new gradient is in the same direction as the gradient to be calculated but I am cutting down its size. So, I fit for this big and suppose my maximum length allowed was this, I will keep this as my new G vector, just for clarity I will call it G star. This was original G vector. So this is gradient clipping and of course you proceed with this new gradient and each time you hit above a particular gradient. Now this might seem like a simple hack but actually it tends to work well. So this is what we do for exploding gradients.

(Refer Slide Time: 18:19)



Vanishing Gradient → Different architectures (GRU, LSTM)
Computation of the gradient is expensive (T BPTT)
In general,
F Prop through k_1 steps
Back " " k_2 steps
$k_1 = k_2$
Mini Batch Gradient Descent
L_1
y_1
h_1
x_1
L_T
TBPTT
y_T
h_T
x_T
W
W

Now what do we do for vanishing gradients? Unfortunately no such simple solution exists. So, for a vanishing gradient, we use different architecture. I will talk about the basic idea of this architecture and why they work in the next video. But 2 such architectures are what are called the gated recurrent unit and long short-term memory. You might remember long short-term memory from what we actually used in the example problem, in the earlier video on sensors etc.

Okay, so that is what we use and we will see how to do that in the next video. Now, finally there was a 3rd problem, which was computation of the gradient, this is expensive. As soon as the solution am going to give right now kind of handles to a certain extent, both the vanishing gradient and the exploding gradient problems, so it is sort of a compound solution. So, if we go back here to this, let us say that this figure actually represents thousands and thousands of time steps, okay. And you want to calculate back propagation through time. Now how would you do it?

The way you would usually do it is you will forward propagate through the whole thing, calculate the whole of LT and then you will back propagate through the whole thing. Now remember the example that we had before this, that example had 65,000 time sequence. Now, would you go back for the full thing and come back through the full thing, by that time almost any correction you give will lead to vanishing or exploding gradient problems, + it would become potentially very expensive just to do one gradient update.

Now, similar to the thing that we had between gradient descent and stochastic gradient descent, you can do something similar with back propagation through time and that is called truncated back propagation through time. So the solution to that is truncated back propagation through time. Now it is a very simple solution, so let us say I am just going to draw boxes here, each box represents an input, hidden layer and output. So, let us say input, hidden layer, output and of course loss, okay, so on and so forth. So let me just draw it at the end here.

So given the sort of time invariant nature of RNN's, remember that we are assuming that the relationship is the same and in fact you can cut it anywhere in the middle and you are going to get exactly the same W, okay. Given that, the basic idea that people found out is a set of training for the whole sequence that you have available to you, you sort of split it up into many many batches. You might recall something called mini batch gradient descent. This is somewhat similar, there are minor differences but in notion, this is very very similar, truncated back propagation through time.

So what do we do? Let us say just as an example to see here. I will forward propagate through 2 steps, okay and back propagate through 2 steps, this is one possibility. If I do that, remember that W is the same everywhere. So I will get some new updated W. When I start here, forward propagates through 2 steps, back propagates through 2 steps, my W is now updated , okay. Now when the W is updated, I will forward propagate through the whole thing, okay. So I keep on doing this.

Now, in general what you do is forward propagate through K1 steps and back prop through K2 steps. The basic idea, okay, so there are many male implementation detail here, luckily you know most packages take care of this for us. So, just for you to get in notion of how all you can train, okay, so that you can make up a new architecture, you can think of all these varieties of ideas. All you are doing is forward propagating through part of the data, back propagating through a different part of the data.

How does this help? If you back propagates through a small amount of data, your gradient will neither blow up, nor will it vanish. Now what is a good rule of thumb? It is actually hard to say for some problems, hundred steps are good for some problems, 10, 20 steps are good, etc. This depends on the type of the problem and you will have to experiment with it like everything else. In some sense, a lot of neural networks is still engineering, it has not come to the level of science as yet. So this basic idea is called truncated back propagation through time.

A simple warning, as of, you know, as of right now K loss as I understand it has K1 equal to K 2, okay, so that is you have to forward propagate and back propagates through the same number of steps for truncated back propagation through time. So as a summary, when you have RNNs, they actually have a strong problem. All neural networks and all CNN's, in fact this has been the major thing that has kept them back for a long time is that they have a problem with training. It is not just the size of the dataset, but sometimes either gradients explode or they vanish.

This happens in everything but it happens particularly in RNN's, because you have the same W across time. Because of that you can either get vanishing gradient very quickly you can get exploding gradient very quickly. You can also get a large amount of computation, especially if you are data size is large for the kind of examples that I showed you earlier. In the case of exploding gradients, use gradient clipping, in the case of vanishing gradients, try and use alternate architectures which I will start from the next video. And in case of handling computational issues, try and use back propagation, truncated back propagation through time, thank you.