Chapter 12 : Part 1

**Deep Learning**

**RNN: Recurrent Neural Network**

Introduction

**12.1.1 What we will learn in this Chapter**

RNN is one of the most advanced algorithms that exists in the world of Supervised Deep Learning. We will cover following topics in this chapter.

1. The idea behind Recurrent Neural Networks (RNN): We'll see how they compare to the *human* *brain*, we'll understand what makes them *unique* and *special* as compared to regular *ANN*.
2. The Vanishing Gradient Problem: It has been a major road block in the development and utilization of RNN, something that prevented RNNs from being what they are now.
3. Long Short-Term Memory (LSTM): One of the most popular solutions to the *Vanishing Gradient Problem* is the *Long Short-Term Memory* or *LSTM* neural networks.

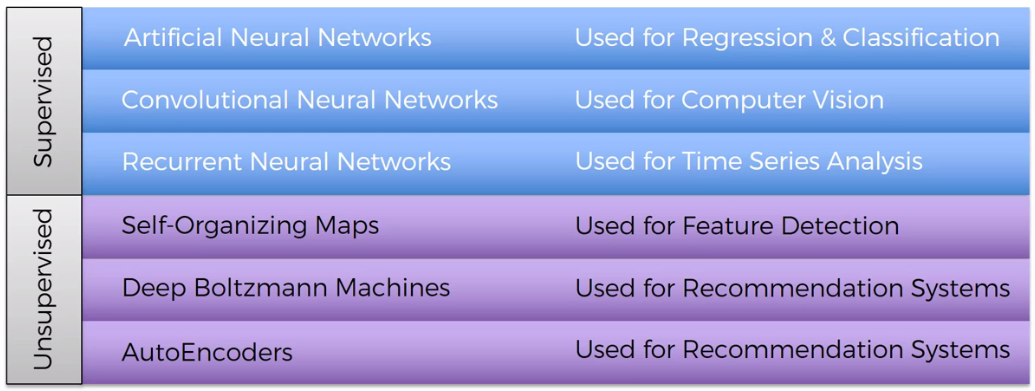
* Here we'll talk about *LSTM* *architecture*.
* We will find out exactly LSTMs work and what that complex structure is inside them by break it down into simple terms.

1. LSTM Variations: We’ll see some other options of LSTMs exist out there, what other architectures you might come across in your work.
2. Step by Step Example: We'll look at some great examples posted by one of the researchers.

* We'll understand even better on an *intuitive* *level* how LSTMs actually work, how they think.
* Here we'll be like *neuroscientists* trying to understand what's going on in the *brain* of an LSTM.

**12.1.2 Deep Learning Methods and Human Brain**

* Here we have break down some ***Deep Learning Algorithms*** into ***supervised*** and ***unsupervised*** categories. So ANN, CNN and RNN are supervised algorithms. And SOM, Deep Boltzmann Machines and AutoEncoders are unsupervised algorithms.



|  |  |
| --- | --- |
| * We can compare the whole concept behind deep learning with the human brain. We can get kind of similar functions as the human brain has. * It would be pretty cool if we could somehow link the different ***branches*** of ***deep learning*** that we've discussed, or the algorithms that we discussed. |  |

* Here we've got the brain, it's got main three parts.

1. Cerebrum: Which is all of this colored part. Frontal lobe, Temporal lobe, Occipital lobe, Parietal lobe.
2. Cerebellum: Which is underneath Cerebrum there and that's the little brain.

* We discussed cerebellum in ANNs chapter (that was a big orange picture). But it doesn't represent ANN.

1. Brainstem: Which connects the brain to the organs such as: arms, legs and so on.

|  |  |
| --- | --- |
| * Now, in the ***CEREBRUM*** has four lobes: | 1. Frontal lobe, 2. Temporal lobe, 3. Occipital lobe, 4. Parietal lobe |

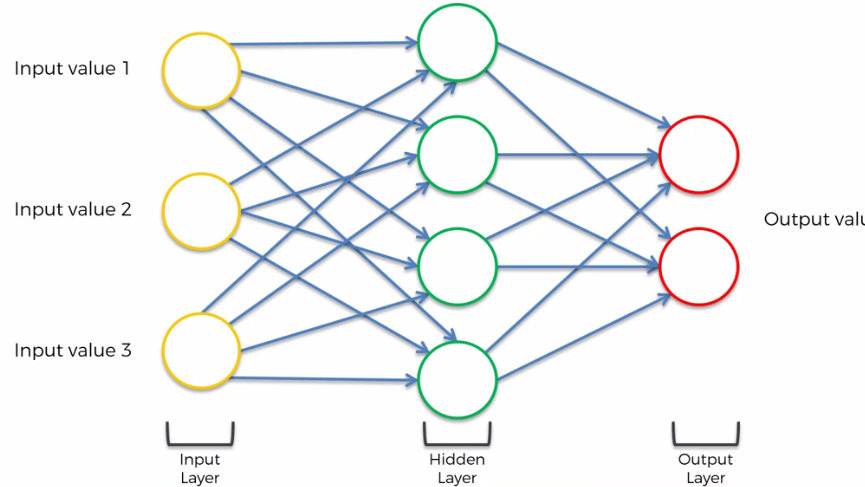
* Weights are Long Term Memory of a neural network - TEMPORAL lobe: ***ANN*** is the main part of ***deep*** ***learning***. In ANN we had, *Input/Output* layers and *Hidden* *layers*. Those layers are created by nodes called *neurons*. Inside each *neuron* we had "*activation-function*" also we had "*weights*" for *Synapses*. We had *forward/back* *propagation*, cost function.
* The fact that ANNs can learn through prior experience, or through prior impulse, and through prior observations.
* But the main thing about ANNs, are the Weights. And philosophically those ***weights*** represent ***long term memory***.
* So once you've run your ANN and you've trained it, you can switch it off. But it remembers the weights. So whatever input you put into it, it will process it the same way as it would *yesterday*, as it will *tomorrow*, as it will the day after.
* So, the ***weights*** are ***long term memory*** of a ***neural*** ***network***.
* That's why the ANN similar to temporal lobe. Philosophically, ANNs are a start to deep learning and they represent long term memory. So, we've to put them in the ***temporal*** ***lobe*** because the ***Temporal Lobe*** is responsible for long term memory.
* CNN represents OCCIPITAL lobe: Then, ***CNN*** represents ***vision***, ***recognition*** of images and objects and so on, so that's the OCCIPITAL lobe.
* RNN represents Frontal lobe: ***RNNs*** are much more like ***short*** ***term*** ***memory***. They can remember things that just happened in the previous couple of observations and apply that knowledge going forward.
* So, ***RNNs*** similar to the frontal lobe. That's where we have a lot of the short term memory.

*(The frontal lobe also is responsible for personality, behavior, motor cortex, working memory, and lots of other things.)*

**12.1.3 How RNN works**

Here, we've got a simple ANN. Three inputs, two outputs and one hidden layer.

* We turn this into an RNN by squashing it. But remember that those neurons, the whole network, is still there. Nothing changed, we just squashed it for our purposes.

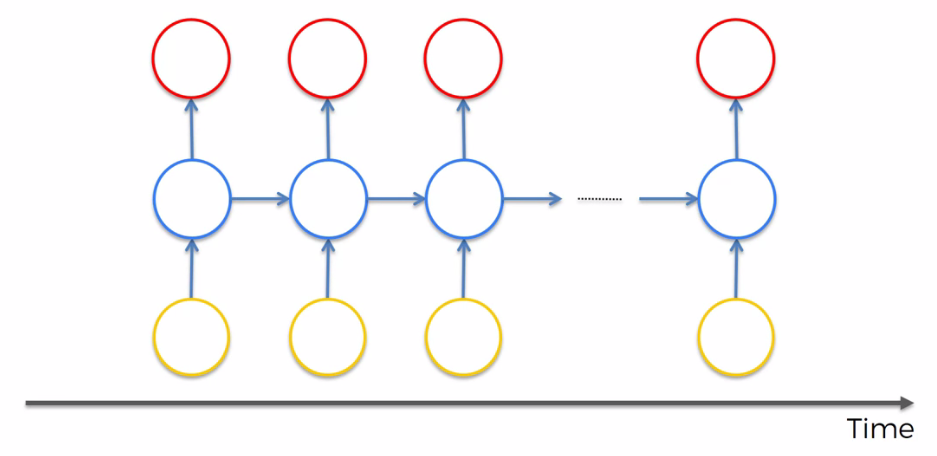


|  |  |
| --- | --- |
| * Then to simplify things we're just going to change these *multipliers/synapses* into *two*, then we're gonna *twist* thing whole thing, to make it *vertical* because that's the *standard* *representation*. |  |

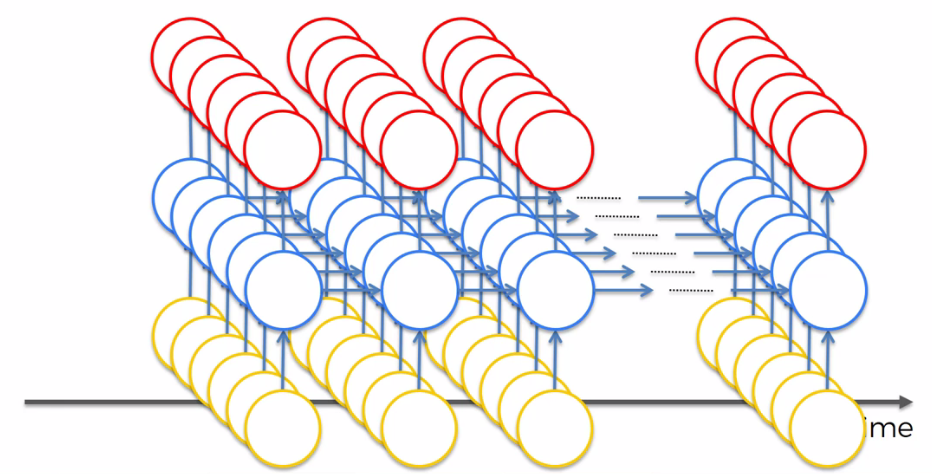
* Then in terms of *neural* *metrics* we're gonna *color* them, instead of *green* we're gonna color the *hidden* *layers* in *blue*.
* And we're gonna add a *loop*, represents the *temporal* *loop*. Means that this hidden layer not only gives an output but also feeds back into itself.

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

* If we unwind, or unroll, this temporal loop then it represents the following ANNs

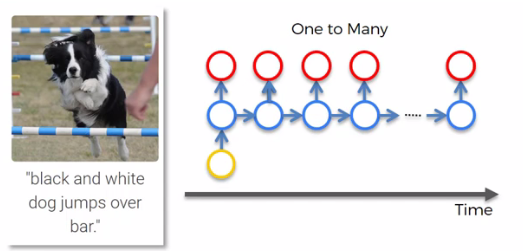


* But keep in mind: we're looking in a new dimension, that the layers are actually still there (all *circles* now represents *layers*).
* We just remember that each one of these circles is not one neuron, it's a whole layer of neurons.
* Short Term Memory: In above figure we can see that, we've *inputs* coming into the neurons, then we got *outputs*, but also the neurons are *connecting* to *themselves* through *time*.
* So, that's the whole *concept* of *short term memory*, that they remember what was in that neuron just previously and before that.
* It just *remembers* what it was *previously*, and that allows them to *pass information* on to *themselves* in the *future* and analyze things.

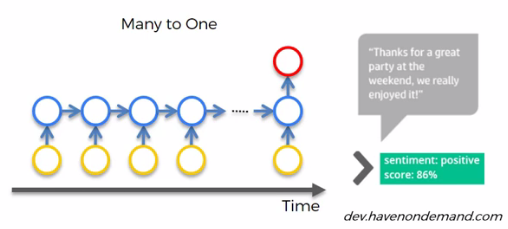


* For example: In our case we are talking about RNN in this chapter and to figure it out we learned ANN in previous chapters (i.e. we keep in our memory ANN stuffs like: Activation-function, neuron, weights etc from previous chapter so that we can understand RNN). It enables us to understand RNN in this chapter. So to learn new things in this current chapter we need to remember previous chapters (so some kind of short term memory is happening in our Brain).
* And same thing here, we are mimicking the human brain.
* Short Term Memory is powerful: Long term memory is great, but short term memory is more powerful. And that's where recurrent neural networks come in.
* Example: Following are some examples from Karpathy blog, ***karpathy.github.io***,
* One to many relationships: This is when you have ***one*** ***input*** and have ***multiple*** ***outputs***. An example of this is an image where a *Computer Describes The Image*. So, you have one input, the image, and that would go through a CNN and then it would be fed into an RNN, and then the computer would come up with words to describe the image. And this is an actual computer describing the image.

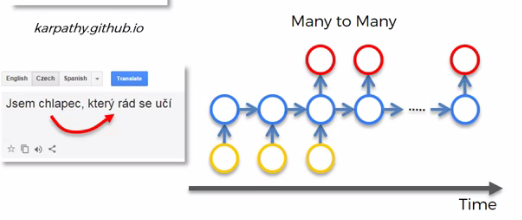
"***Black and white dog jumps over bar***".



* This is a computer that looked at this image and it recognize the "black and white dog" from its training using , the long term memory (ANN *weights*), and using *CNN*.
* And then the RNN allows it to *make sense out of the sentence*. So, you can see that the sentence actually flows. There's an *and*, there's an *over the bar*, and then there's like a *verb*, there's a *noun*, and so on. So, basically the *RNN* is what allows it to put a sentence together in this case.
* Many to one: An example would be *sentiment analysis*. So, when you have a lot of text and from that text you kind of need to find out that: is this a Positive comment/ Negative comment? What's the chance that it's a positive, or how negative is that comment?

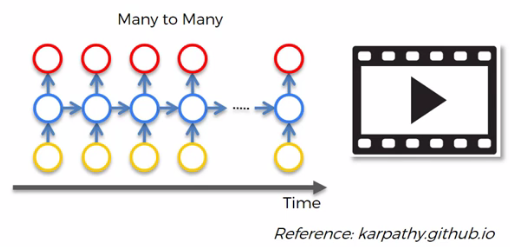


* Many to many: Here, we've got an example of Google Translator. In translation it find out the related word.

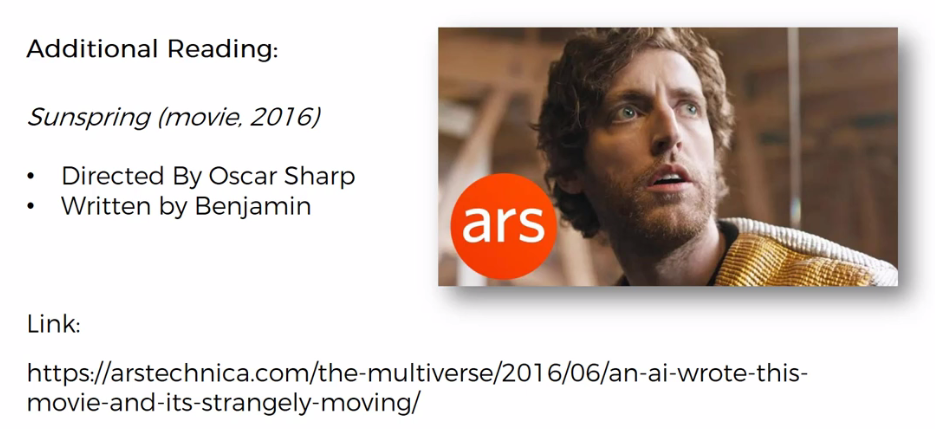


|  |  |
| --- | --- |
|  |  |

* So, if I say here from English to Czech. I say, "I am a boy who likes to learn". In other languages it is important what gender your person is, right? So, here boy is male. It finds the *male-gender related word*. If I change this to *girl* in *English* nothing changes. But in *Czech*, the other words *change*.
* Another many to many example, you can use RNNs to subtitle movies.

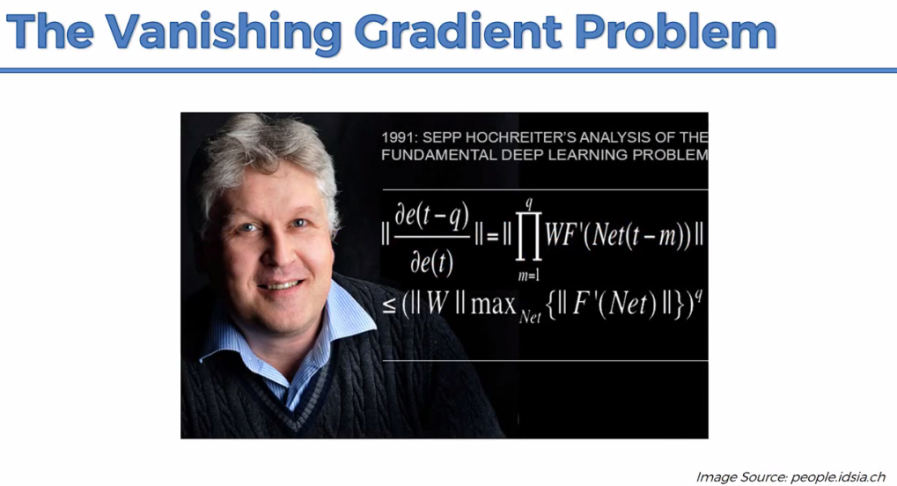


* Additional watching: Here is a movie called ***Sunspring*** in 2016 directed by ***Oscar*** ***Sharp***. And this movie was entirely written by *Benjamin* who is an RNN, an LSTM to be specific.
* There's actually an interview of *Benjamin* and he actually gave himself the *name* of *Benjamin*, that's why they call him *Benjamin*.
* What you'll find amazing is that Benjamin is able to *construct sentences* which kind of *make sense* for the most part, which is good, but what he lacks is kind of the *bigger picture* (relation *between* the *subjects* that makes *sense*.). He cannot come up with a *plot* that *consistently* *makes* *sense*.
* When you're watching, *separate* the *sentences* and you'll see that each sentence on its own more or less, *90%* of the time, makes sense. But overall, he can't properly *link* *sentences* *together* (link is lost to different sentence). And that's the next step for RNNs, we have to fix this in future.



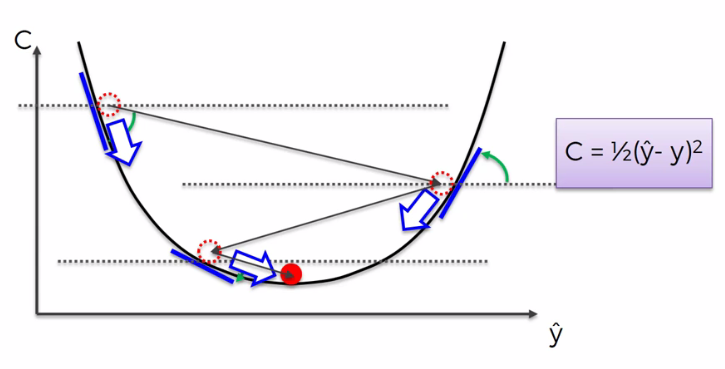
**12.1.4 Vanishing gradient Problem**

Vanishing gradient was first discovered by Sepp Hochreiter. And the second person is Yoshua Bengio (professor at the University of Montreal).



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

* Vanishing gradient problem: So, as you remember, following is the *gradient descent algorithm*. We're trying to find the *global* *minimum* of your *cost* *function*, and that's gonna be the *optimal* *solution*, optimal *setup* for your *neural* *network*.

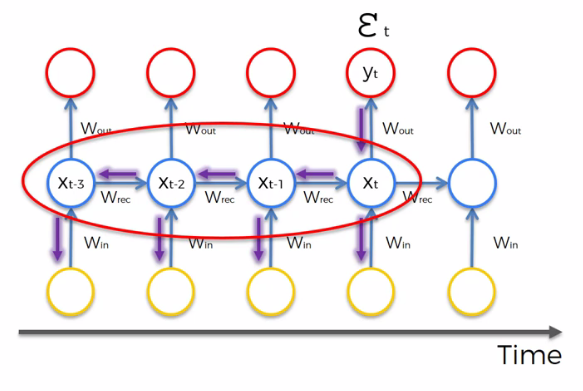


|  |  |
| --- | --- |
| * Our information *travels* *through* NN to get output, and then the *error* is calculated and is *propagated* *back* through the NN to *update* the *weights*. * It is same as *ANN* but here all the *circles* represents *Layers* (not nodes). * Every single *node* here is not just a node, it's a representation of a *whole layer of nodes*. There's lots more neurons behind the ones that we can actually see because each one represents a layer. |  |
|  |  |
| * In a RNN is a similar thing, when your information travels through the network it travels like this: * It travels through *time* and *information* *flows* from previous *time-points*, keeps coming/going through the network, and remember that every node here is a *whole layer of nodes*. |  |

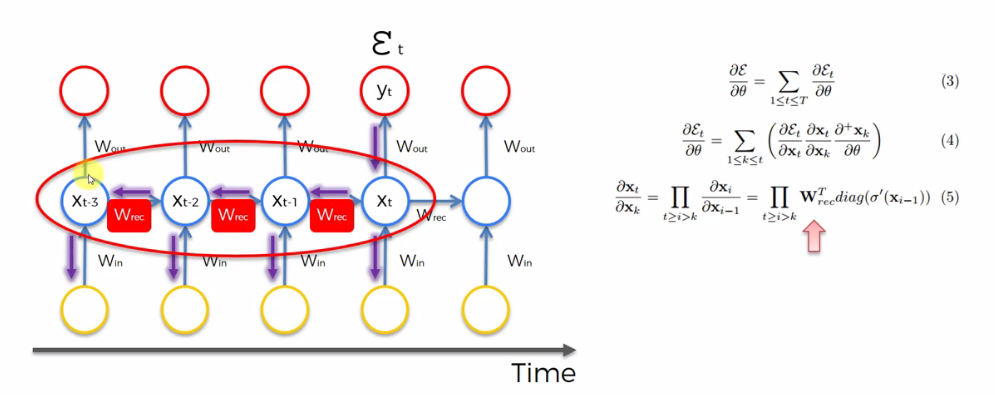
* So, at each point in time you can calculate your ***cost*** ***function***, or your ***error***. During the training, *cost function* compares your *output* (the red circle) to your *desired* *output*. Then you get these values throughout the *time* *series* (for a single *red* *circle*, calculates the *cost* *function*).

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

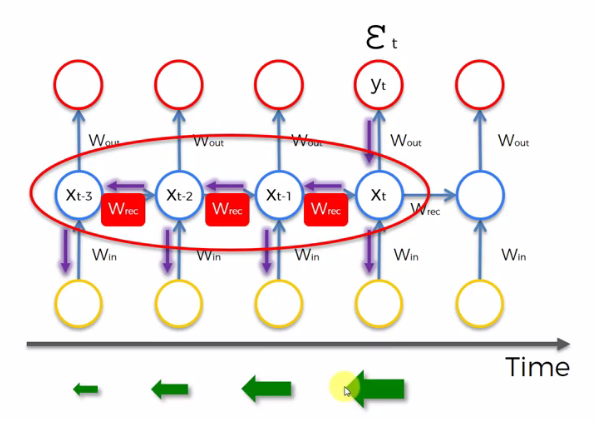
* How weights got updated (Gradient decent occurs): Now let's focus on one value, lets consider just single . Here we've calculated the *cost function* epsilon-t: **.** Now we want to propagate that *cost function* back through the *network*.
* To do this, every *single* *neuron* which *participated* in the calculation of the *output* *associated* with this *cost* *function :* , should *update* their *weights*, in order to *minimize* that *error*.



* But we have to note that, it's not just the neurons directly below (directly below that red circle) *:* .
* It's all the neurons that contributed (i.e. all previous time-point), all of these *neurons* as far back as you go. Depending on how many *time-steps* you take.
* For example, you might take *50 time-steps* before, then you have to update weights for all previous *50 time-steps*. You have to *propagate* all the way *back* *through* *time* to those *neurons*.
* Following is the math behind RNNs, we've got , and (stands for weight recurring), and that is the *weight* that is used to *connect* the *hidden* *layers* to *themselves* in the *unrolled* temporal *loop*.



* Here we can see that to propagate from the layer to the next layer , we need to apply .
* In simple word, we are simply multiplying the output by the weight, and then we get to the next layer.
* Here we're multiplying by the *same* exact *weight* in *multiple* *times*, as many times as we need to go through this temporal loop.
* And this is where the problem arises, because when you *multiply* by something *small* your value decreases very quickly (eg: 0.2\*0.2 = 0.04), and from the above formula we can see that.
* Now remember that at the *very-start* of the *propagation process* the weights are assigned *randomly* to *NN* and those random-weights are close to ***0***. Hence due to multiplication of such small values, our ***"gradient"*** decreases from onelayer *to* previouslayer during Back-Propagation.
* A vanishing gradient is bad for the network. Because when the *gradient* as it goes back through the network, it is used to *update* the *weights*, and the *lower* the *gradient* is the *harder* it is for the *network* to *update* the *weights*.
* The lower the gradient gets, the updating the weights get slower. (the higher the gradient the faster it's going to update the weights).

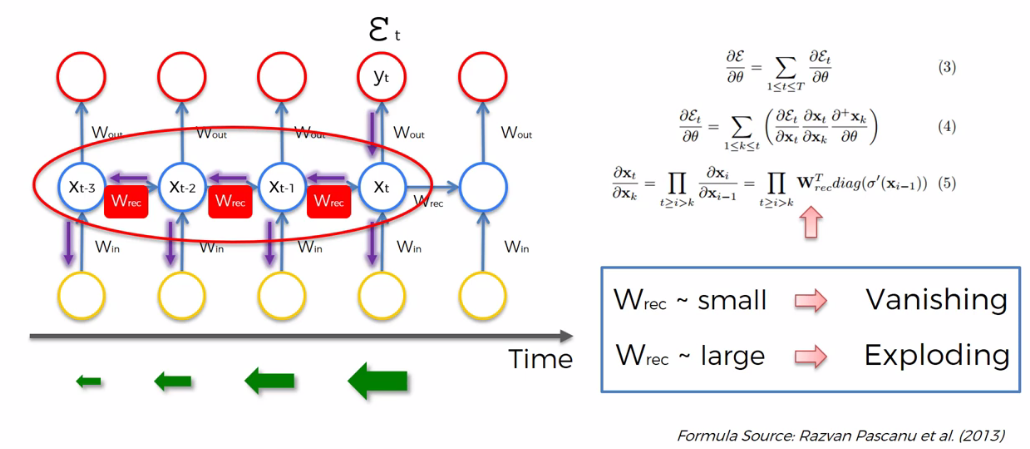


* So for instance say we have 1,000 epochs. Then *some layers* and *part* of our *network* cannot get *trained* properly.
* Having their gradient's so much smaller, they're gonna be updated slower. Therefore by the end of the *1,000 epochs* you might not have the final results there, and some *part* of the *network* is *trained* and some *part* is *not* *trained* (based on their cost function).
* But the problem here is not just that *half of our network* is not trained properly, but also that those *weights*, on those *untrained* *layer* generating *wrong* *outputs*. And those *wrong-outputs* are being used as *inputs* for further *layers*.
* So, the training here has been happening all along based on, inputs that are coming from untrained *neurons*, untrained *layers*.

In simple word that’s the vanishing gradient problem for RNN.

**12.1.5 Exploding Gradient Problem**

|  |  |
| --- | --- |
| Exploding –gradient: If is small, then you have a *vanishing gradient problem*. If is large you have an *Exploding Gradient Problem*. Of course there is so much parameters in the formula, activation-functions weights etc but in a nutshell these two thing can happen. |  |



**12.1.6 Solutions for Vanishing/Exploding Gradient Problem**



* For the Exploding Gradient problem

1. Truncated Back Propagation: For the exploding gradient you can have *Truncated* *Back* *Propagation*. So, you stop *back* *propagating* after a *certain* *point*, but that's probably not optimal because then you're not updating all the weights.
2. Penalties: You can have *Penalties*. The gradient being penalized and being artificially reduced.
3. Gradient clipping: You can have *Gradient* *Clipping*. So, you could have a *maximum* *limit* for the *gradient*. *Gradient* never go over this *value*, and if it does, then that value just *stays* at *same* as it *propagates* *further* down through a network.

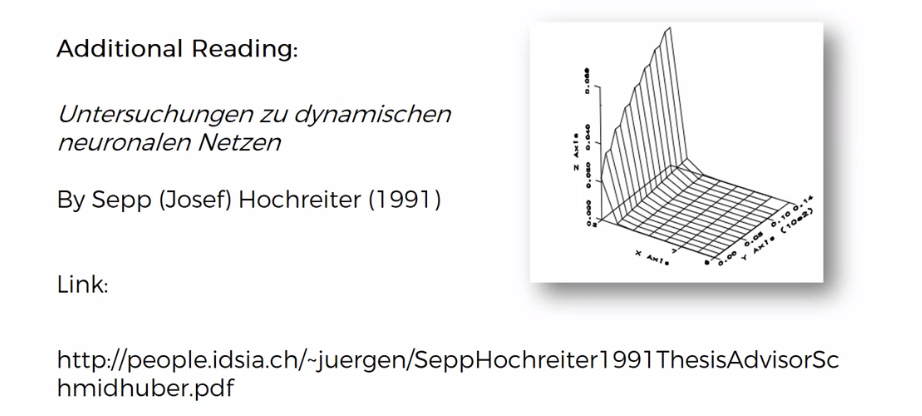
* For the Vanishing Gradient problem

1. weight initialization: You have *weight* *initialization*, where you are smart about how you *initialize* your *weights* to *minimize* the *potential* for *vanishing gradient*.
2. Echo State Networks: You can have, other type of network called the *echo state networks*. They are designed to solve the vanishing gradient problem.
3. Long Short-Term Memory networks, OR THE LSTMS: There's also a different type of network called the long short-term memory networks, or the LSTMs (which are extremely popular). We will talk about it in next section.

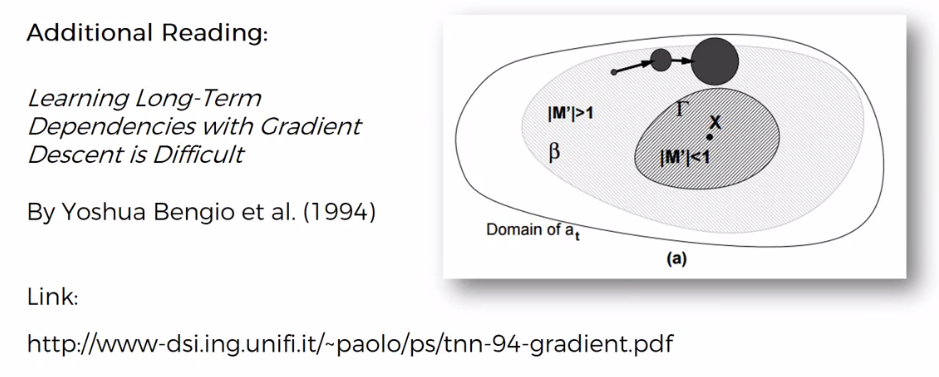
**12.1.7 Additional reading**

The original works by *Sepp Hochreiter* and *Yoshua Bengio* are good to read.

* So, this is *Sepp's* paper in 1991. It's completely in German. If you understand and can read German, then definitely this could be a good read for you.



* Then there's *Yoshua Bengio's* paper which is called *Learning Long Term Dependencies with Gradient Descent is Difficult*, 1994.



* We also recommend looking into this paper called *On The Difficulty Of Training Recurrent Neural Networks* by Razvan Pascanu.

