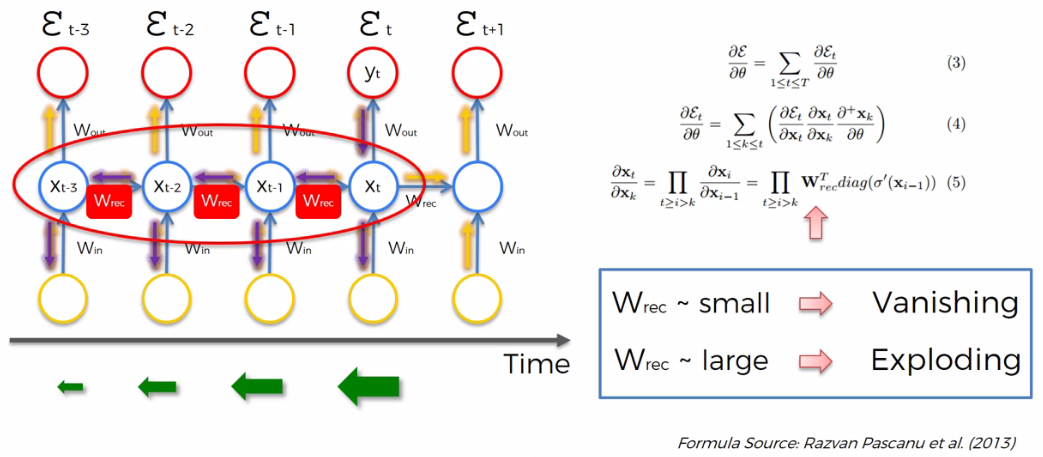
Chapter 12 : Part 2

**Deep Learning**

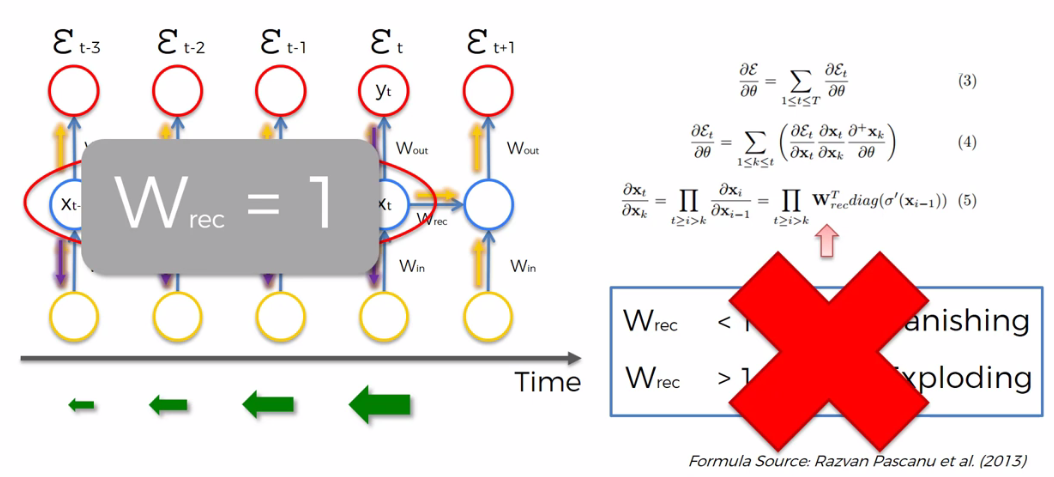
**RNN: LSTM - Long Short-Term Memory**

long short-term memory Introduction

**12.2.1 LSTM**

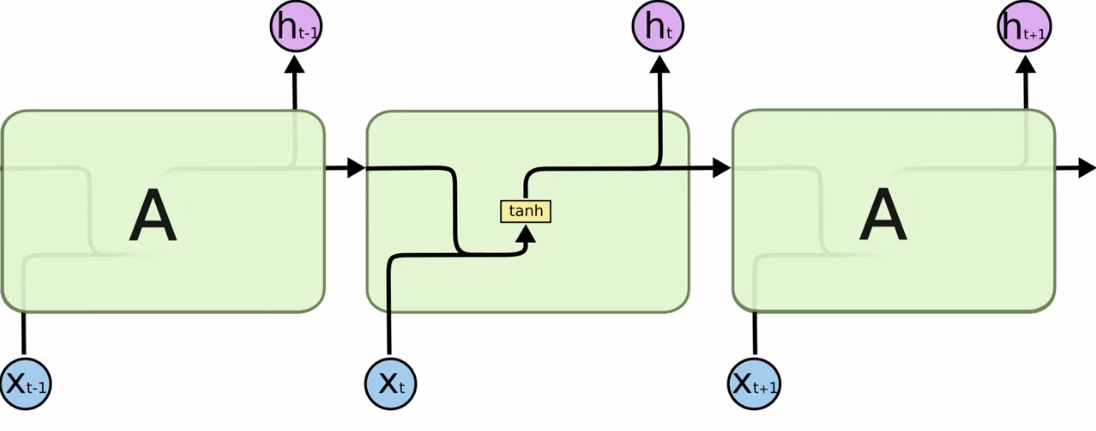


|  |  |
| --- | --- |
| * Here we've got vanishing gradient problem. And as a rule of thumb, we can see here that ifis small, then we have Vanishing Gradient, if is large, then we have Exploding Gradient. To solve this problem there is couple of solutions. But here we only focus on LSTM. |  |
| * The first thing that comes to mind is to make equal one, , and that's exactly what was done in the LSTMs. |  |



|  |  |
| --- | --- |
| * Here we've got a recurrent neural network. With unraveled temporal loop. |  |

* This is what it looks like if you dig inside the RNN. The images are taken from Christopher Olah.

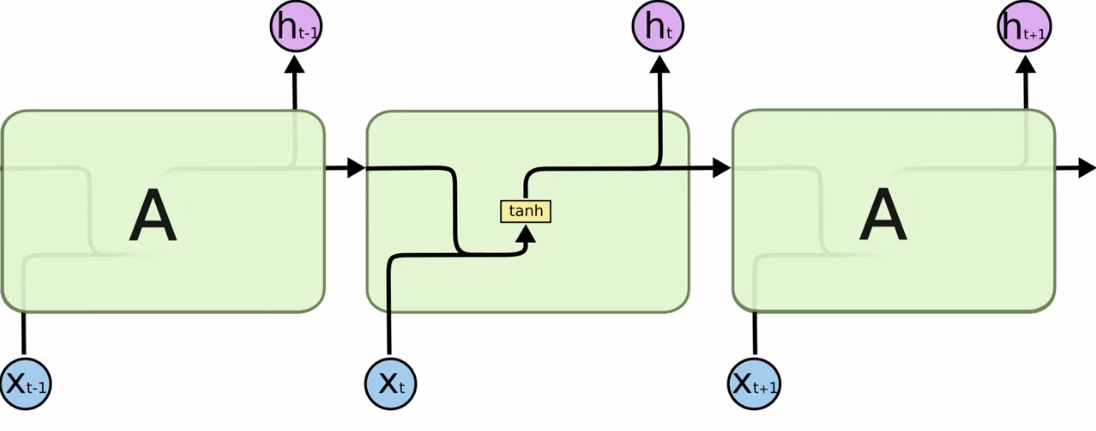


* Here is his blog Very well-written blog with amazing images.

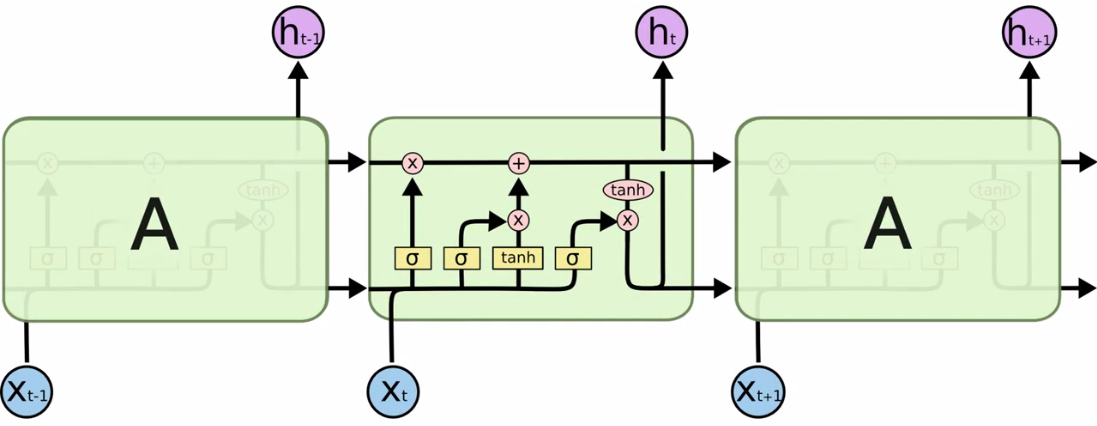
|  |  |
| --- | --- |
|  |  |

**12.2.2 RNN vs LSTM-RNN**

In Following images the first one represents the Simple RNN and 2nd image represents "RNN with LSTM". Here LSTM is a special kind of network applied to RNN.

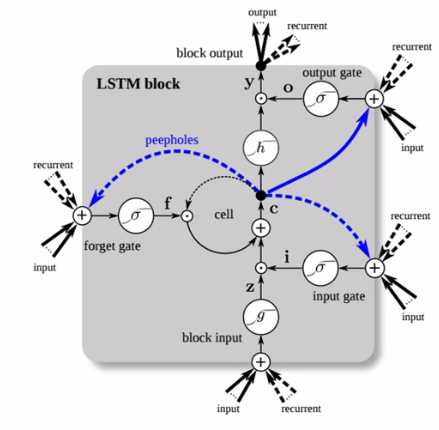


***Standard RNN***



***RNN with LSTM applied***

* Above is a simple representation of LSTM. Following is the detailed representation. Image Source: ***arxiv.org/pdf/1503.04069.pdf***

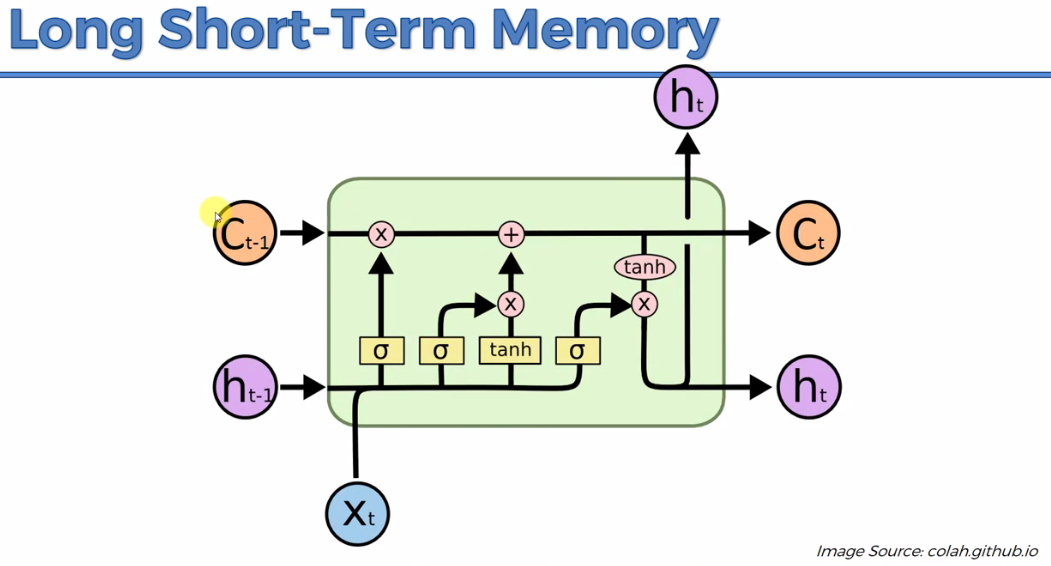


**12.2.3 How LSTM works**

|  |  |
| --- | --- |
| * So here we've got the inside-look of our simple-Standard-RNN. And this is where the problem lies. * This operation that happens here is actually a ***neural-network-layer-operation***.   ***In a simple word:*** outputs *coming* into this *module* and this operation's applied and then goes into the *next* *module* operation's applied, and so on.   * When we *back* *propagate*, it goes through all of those *operation*, and that's where the *weights* are applied (that's where the is sitting). And through this *back* *propagation*, the *gradient* *vanishes*, which means that the *weights* *cannot* be *updated* properly or fast enough to *train* the network *properly*. This is the *standard* *RNN*, |  |
| * Now Following image is the LSTM version of RNN.      * Notice that the main point here was setting . Well that's this line that *flows straight* here (the upper line), that *pipeline* at the top of the *LSTM* version of *RNN*.      * There's not much going on, just two *pointwise* *operations* (removal & addition) and no *Complex Neural Network Layered Operations* happening in this line. | |
| * Actually LSTMs have this pipeline as a *memory cell* or we can call it *memory pipeline*. This ***pipeline*** flows through time, sometimes it faces *pointwise* *operations* to remove/add something. * But mostly it *flows through times* freely and therefore when you *back* *propagate* through these LSTMs, you don't face *vanishing* *gradient* problem. | |
| * However, all complex-operations are brought out to this down part (image on the Right). |  |

**12.2.4 LSTM operations**

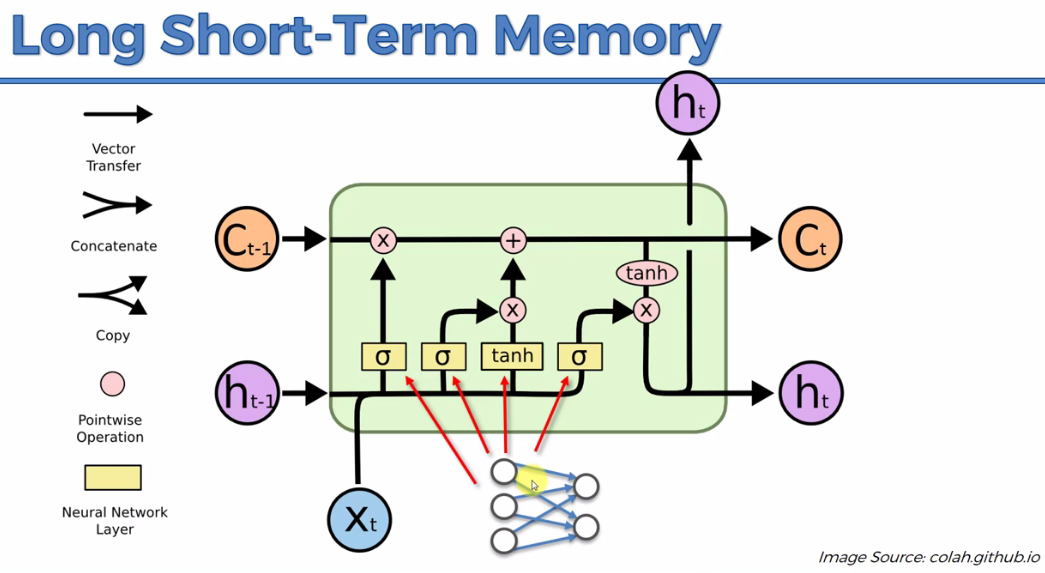
Now let's focus on just one time-step or a single module/block of LSTM. In following representation we have:



* The VARIABLES in this Diagram:
* stands for memory or memory-cell.
* represents the *memory-cell* for current LSTM-module/block.
* And is the *memory-cell* coming from previous LSTM- block.
* is output. Here we can see there is two and one
* One (goes up) goes out into the world, and
* another (on the Right) goes to the next LSTM-module/ block.
* represents the output from the previous module.
* represents the input for current LSTM-block
* So an LSTM a module takes in three inputs: input data , previous modules output-data , memory-cell .
* And it produces two outputs: current modules memory-cell and output-data , (one copy of goes to next LSTM-module).
* All variables are vectors: The important thing is that everything here is a vector. I.e. , , and , are all vectors.
* Each of them *doesn't represent single value*, they are actually *vectors* containing *multiple* *values*.
* The LEGENDS in this Diagram: And let's go through the legend.

|  |  |  |
| --- | --- | --- |
|  | Vector transfers: Any *line* here (in this drawn LSTM-module) is a *vector* being *transferred*. | |
|  | Concatenation: Anywhere in this drawn LSTM-module, if see that there's two lines combining into one. Actually they don't become one single line. But these two lines are *running* in *parallel*.     * You're *not actually combining*, concatenation means that you're combing these two lines *on top of each other*. Basically you have two pipes running in parallel feeding into these neural network layer operations (simultaneously). | |
|  | Copy: In this drawn LSTM-module, if see that there's one line splitting into two (or more).  or  or   * It means that the memories go straight ahead and just copy it when the line splits. | |
|  | Pointwise operations: We've got a couple of *pointwise* *operations* here (5 *pointwise* *operations* are used in this diagram). | |
| Valves: The X's are valves and they all have names. There are three kind of valves.  ***Forget*** valve, ***F***.  ***Memory*** valve, ***V***.  ***Output*** valve, ***O***.  Forget valve is basically controlled by the *layer* *operation* . represents *sigmoid* *function* (decide between ***0*** and ***1***). Based on the decision made by the *layer* *operation* foregut valve ***F*** will close/open.  If it's open, *memory* flows through *freely* through *memory-pipeline*.  If it's closed then *memory* is *cut* *off.* Therefore it's not transferred further and then new memory just will be added in next (in the joint) based on the *memory valve* opened/closed. |  |
| Memory valve, which is also controlled by *layer* *operation* . represents *sigmoid* *activation function*. The value of *- layer operation*, is added (or somewhat added) or not-added to the *memory-pipeline*  based on the decision of the memory-valve ***V***.   * Why we're using Sigmoid Activation Function is because they are from ***0*** to ***1***. ***0*** stands for ***close***, ***1*** stands for ***open***. |  |
| Output valve, also controlled by *layer* *operation* and it decides whether the output can go or not. |  |
| T-shaped joint: And then we've got *T-shaped joint*, where *connects* the memory coming form *memory-valve* to the *memory-pipeline*. Here additional memories got added if the *memory-valve* is opened. |  |
| Tangent operation: Tangent operation, works with values between minus one and one. It's another *pointwise* operation (*layer* *operation*). | |
|  | The neural network layer operations: Above we talked about *pointwise operations*. (Pointwise is like ***element*** ***by*** ***element*** over vector, if you wanna multiply a vector by zero, you multiply every element by zero. Or multiply a vector by certain amount is multiply each element of the vector with that amount).   * But the things going on inside those NN-layer operations are bit more complex than *pointwise operations*. * Here on those NN-layer operations we've got a *NN-layer* coming in and then we've got a *NN-layer* coming out. * Everything here is a vector. So we've got a *Layer of these Sigmoids* ( or ) which *controlling* the *valve* for each one of these *elements* in the vector of *memory*.   So it's *not* just *one* *Sigmoid*. Its a whole *layer of Sigmoids* coming in and then we've got a layer coming out. | |

|  |  |
| --- | --- |
| ***Forget*** valve, ***F***.  ***Output*** valve, ***O***.  ***Memory*** valve, ***V***.   * In literature you will see letters ***F***, ***V***, and ***O*** in the actual formulas representing these valves.      * But we imagine it like a real valve (in plumbing). Where we've got *water* or basically *something* flowing through, and then you can either close it or you can open it, or you can *close it to some extent*. |  |



**12.2.5 LSTM steps**

So we're ready to look into above LSTM diagram in step by step.

|  |  |
| --- | --- |
| * Step 1: Here two values enter to the LSTM-block. One is *input-value* and other is the *output-value* coming from a ***previous LSTM-block***.   They are *combined*, ***layer-operation*** decides whether this value should go ahead or not (i.e. *forget-valve* should be *closed* or *open* or *somewhat* *closed* or *open*)   * ***Somewhat*** means to ***some degreee***, but not to a ***large degree***. |  |
| * Step 2: In 2nd step we've got and again they first flow parallel, and combined in or operation. Basically and are *layers of neurons*, or decides which values *can* pass and which *cannot* or *somewhat* pass. |  |
| * Step 3: Then we've got the memory *flowing* through *memory pipeline*. We've got the *forget valve*, *joint-operation* and *memory valve*. , closed. * If *forget valve* is opened memory can *flow* and we're adding in some memory if *memory-valve* is *open*. * Or we can let this whole memory flow through, then keep *memory-valve* closed, keep *forget valve* open, the memory won't change. * Or we can keep *forget valve* closed and keep *memory-valve* open, then we can update the memory completely. |  |
| * Step 4: Finally we've got these two values and combined and decide what part of the *memory pipeline* is going to become *output* of this *LSTM-module* : is it going to be *full-output* or *partial-output*. |  |

* Example: *Google Translate* example for English to Czech.

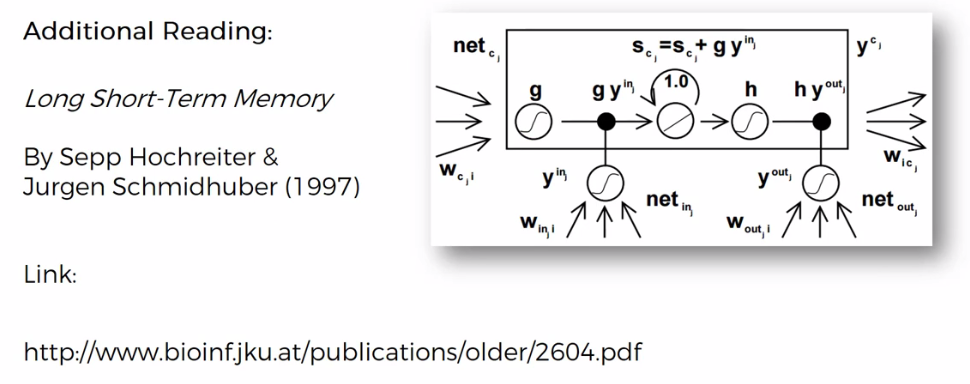
|  |  |
| --- | --- |
|  |  |

Now for example, we can consider our *Google Translate* example. Here we are translating English to Czech. Remember in Czech gender matters. So changing *Boy* into *Girl* in Czech we need to change gender related word. Here RNN works as follows:

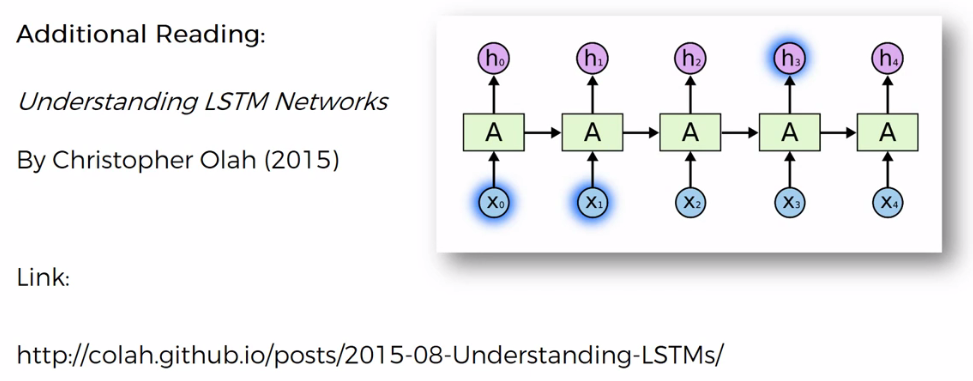
* First if it is *Boy* then *Memory* flows *freely* in the *memory* *pipeline*.
* If it is *Girl* (or female name like "Amanda") *Sigmoid-layer-function* can detect that and *Forget-valve* is closed. Old memory *cannot* *pass* through the memory *pipeline*. Then new memory added via *Memory-valve*.
* In addition we have extracted gender and other data (*capitalized*, *singular/plural*, no. of letters etc.).
* Output-valve extract the information. For example *Gender* is extracted and send this output as an information to the *next LSTM-module*, so that next module can act according to this Gender (change the gender related word).

**12.2.6 Additional Readings**

* In terms of additional reading, you could definitely reference the original paper by our two authors who created LSTMs.



* If you *don't wanna get* that *deep* into *mathematics* and into the technical stuff there's the great blog by *Christopher* *Olah*,

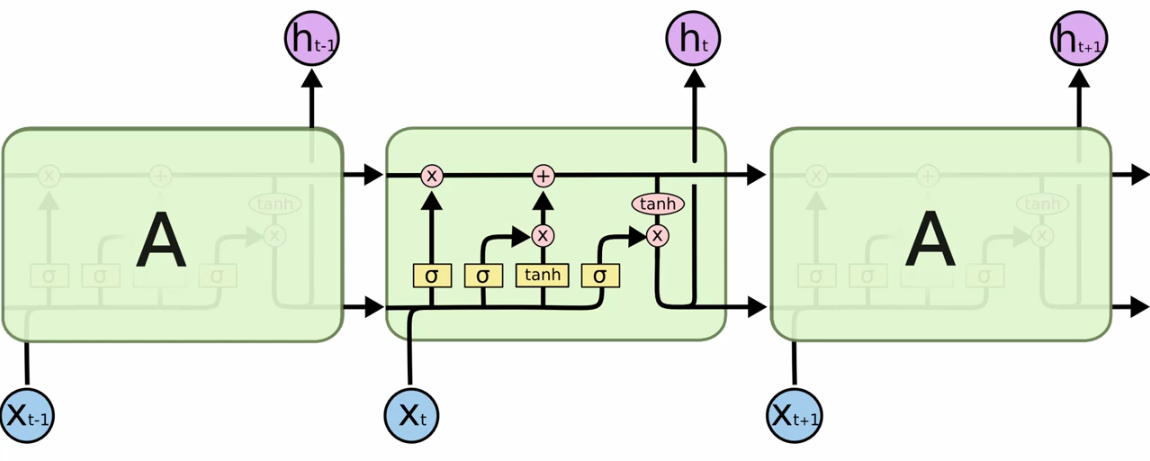


* And there's another blog by *Shi Yan*. *Understanding LSTM and its diagrams*. Those Diagrams are a bit more in-depth, so there's a bit less space saving, but diagrams might be easier to understand in some cases. No mathematics whatsoever, just plain intuition. So also highly recommend this blog.



**12.2.7 LSTM in action - Examples**

Today we're going to look at some practical applications where LSTM in action. We're gonna look at how LSTMs work inside those applications. Here we've got our LSTM architecture.

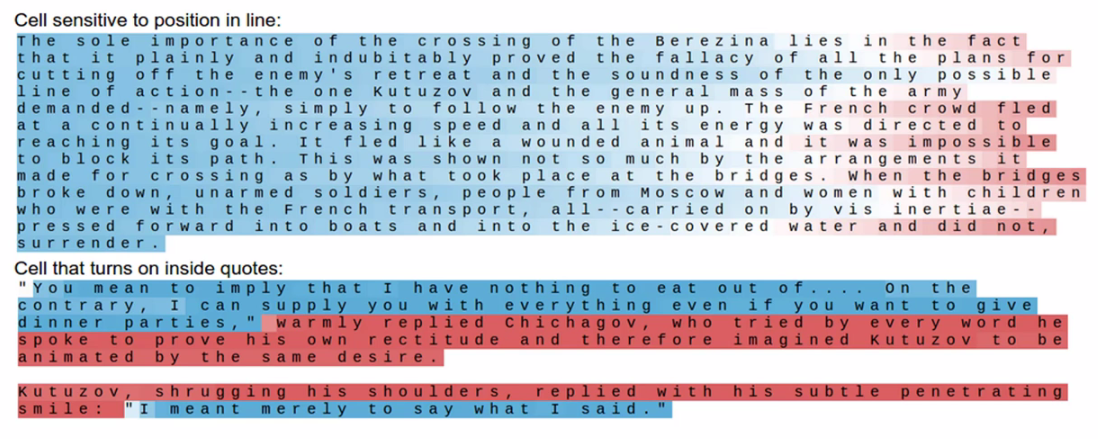


|  |  |
| --- | --- |
| * At first we're going to look to the *poinwise-tangent-function* and how it fires up. * The images that we're going to look are all from *Andrej Karpathy's* blog. This blog is called The *"Unreasonable Effectiveness of Recurrent Neural Networks"*. And the paper that *Andrej* published along with that will be linked at end. * The *poinwise-tangent-function* that we are looking at works in the range ***[-1, 1]***. * According to the paper, the color for ***-1*** gonna be ***red***, ***+1*** is gonna be ***blue***. |  |

* *Andrej Karpathy's* blog:Unreasonable Effectiveness of Recurrent Neural Networks

|  |
| --- |
|  |
| * We are going to look following kind of image where the color codes are done by RNN. |

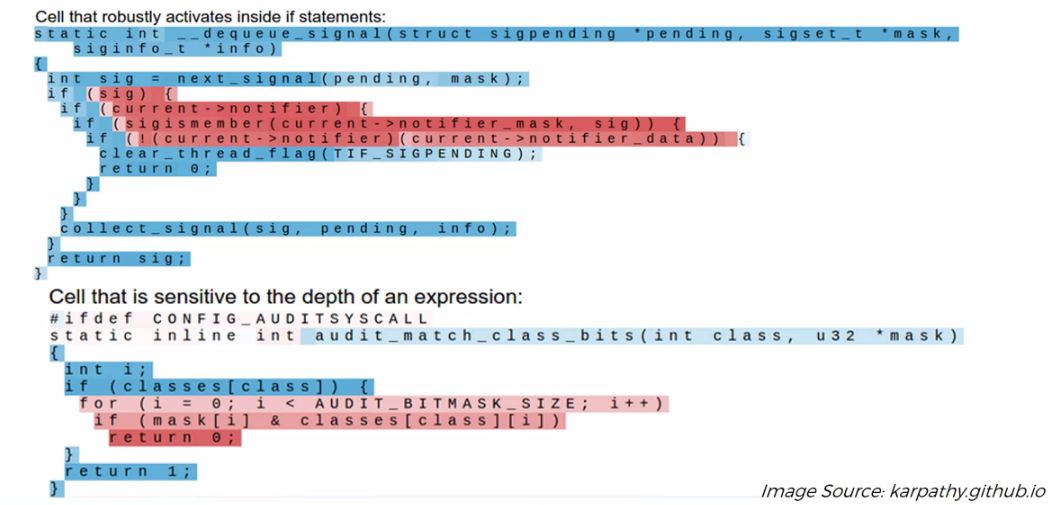
* Example 1: Example of Trained RNN on text: Here's some text, which is given to an RNN, which *learned* to *read* *text* and kind of *create* *text* and *predict* what *text* is coming *next*. And here, this is a snippet from the War and Peace.



* Here two neurons are acting,

1. Cell sensitive to position of line
2. Cell that turns on inside quotes

* Cell sensitive to position of line (detecting the end of the line): We can see that when we get *towards* the *end* of the *line* it's activating. How does it know when it's the end of the line?
* Here in this text we have about 80 symbols per line approximately, so the neuron is *counting* how many *symbols* have *passed* and it's trying to *predict* when the new line character (is an invisible character. i.e. "\n") is going to appear.
* Cell that turns on inside quotes: Then you've got a cell that turn on inside quotes. It detects the texts that are inside a quotation mark.
* But from the color code (red = turn on), we can see the cell is activating outside the quotes. This cell may acting wrong but one way or another, it's *activating* either *inside* the *quotes* or *outside* the *quotes.* It is keeping track of what's happening.
* So very similar to what we discusses previously, where we were *keeping track* of the *subject*.
* The subject could be *gender* (male or female) or understand things like if it's a *singular* or *plural* so that that would effect our *verbs* in our *translation*.
* Same thing is happening here, the *cell* is detecting *quotes*, if you're *inside* or *outside* *quotes* because that effects the *rest* of the *text*. So if the *cell* find a *quotation* *mark* then its expecting other *quotation* *mark* for the *end* of the *quote*.
* Example 2: Example of Trained RNN on Programming language Source code. Here we got some *code* of the *Linux* *OS*.



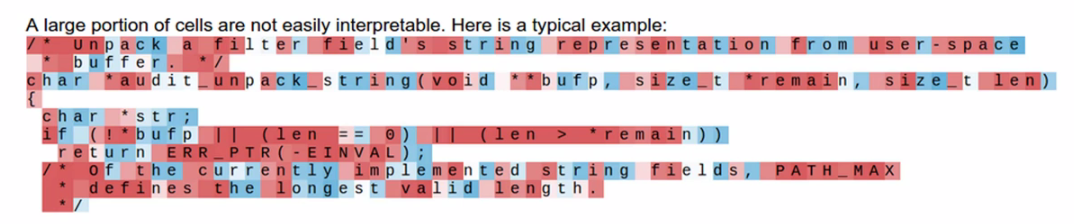
* Cell activates inside the if statements (detects the if statemnet):
* We can see that a cell activates inside if statements.
* If we look closely then we can see that the cell is actually active on the if statements conditions. It detects the ***curly-braces*** "***{}***" and ***normal braces*** "***()***".
* That's how it detects the *if* statements *appearance*.
* Cell that is sensitive to the depth of an expression: In the above image we can also see that the cell is sensitive to how *deep* you are inside of a *nested* *expression*.
* As we go deeper and the expression gets more and more nested, this cell keeps track of that (notice the *red-color gets deeper* in the p of nested if statement).
* So here the *cell* is *using* it's *memory* to keep track of that.
* It's doing that by tracking the **curly-brace** "**{}**" and **indent**.

Note:

* It's very important to remember that none of this is actually hard coded into the neural network. It's doing those things on its own. All of this is learned by network itself.
* Through thousands of iterations and Epochs, using many hidden states (or the *actual* *memory* *cells*, and it assigns them to keep *track* of certain *things* based on what it thinks is *important*).. It assigns *different hidden* *states* of it to keep *track* of *different* *things/subject*.

So it's really evolving on its own and deciding what's important and what's not,

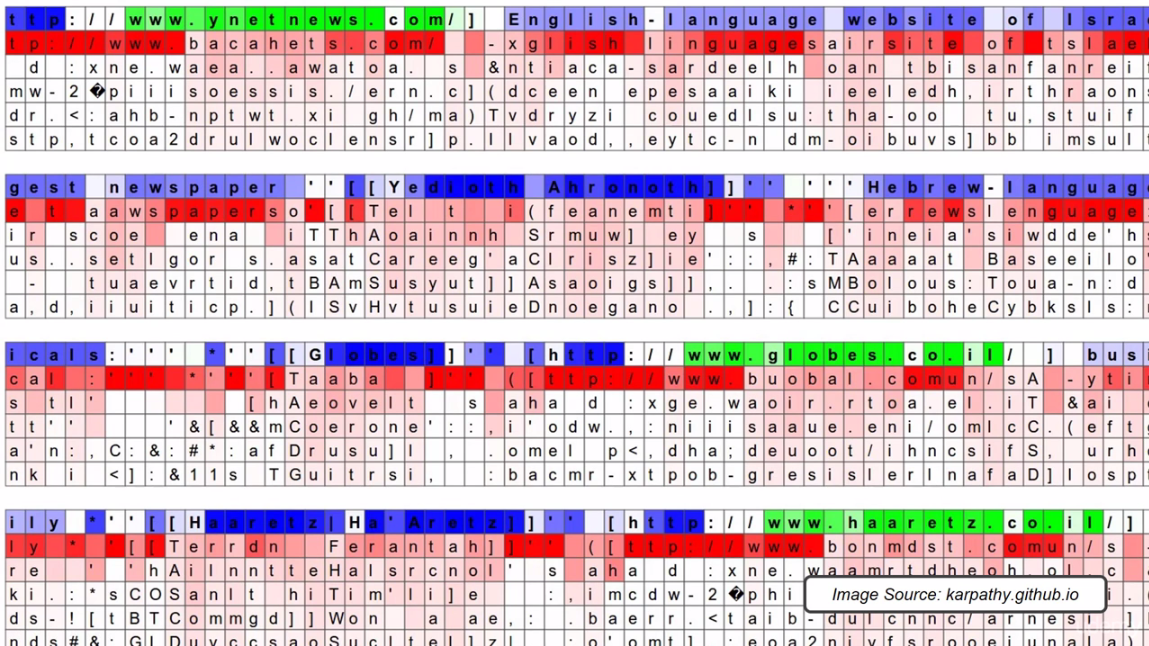
* Example 3: Example of Trained RNN on some text. But it is unrecognizable to human. It's portion of some source-code of a program.
* *A large portion of cells are not easily interpretable*. Here is a typical example of a cell that *we can't really understand what it's doing*. And according to *Andrej* *Karpathy*, about *95%* of the *cells* are *like* *this*.



* The cell is doing something here, but it's just not recognizable to us what is happening there.
* It's much more like the *example* of *CNNs* of the *previous* *chapter*, where after applying the filters or feature-detectors if we're looking out for the detected-features, in the ***Pooled-layer***, ***Flattened-Layer***, ***CNN*** can recognize it but ***human*** cannot.
* Same thing is happening here, by the time the *processed* *feature* get to the *last* *layer*, they're completely unrecognizable to the human eye. But they make sense to the machines. Most the time, 95% of the time, you can't really tell what's going on.

But those 5% of the time, those were the Example 1 and Example 2 that we looked at.

* Example 4: Now we are going to look another output (We are now analyze tyhe output in our LSTM-RNN after it's going through the *pointwise-tangent-operation*, passed through the *output-valve* or *output-gate* and now were gonna be looking at what's being produced over there in "*output* ".)



* We're going to look at the actual output. Above image is another example from *Andrej* *Karpathy's* blog. Here, it's not just showing us if it's active or not but also we can see the guessed letters.
* There are Six rows.
* At the top row, its showing if it's active or not. *Green* means active and *Blue* means *not active*.
* In the other five lines it is saying what's the neural network is predicting, what letter is going to appear in next. Notice the 2nd row letters some letters matched diagonally-left (next in 1st row) to some letters of 1st-row. More *deep-red* means more possibility. *Dark-red* means a very *likely* *prediction*, and *light* *red* means *unlikely* *prediction*.
* We can also see the NN also detecting the *non-active blue* text. For example: "**English-language**".



* If we look closely we can see that this network is looking for *URLs*. That's why all *URLs* are *green* here. Following image shows how the NN is guessing the next appearing letters.

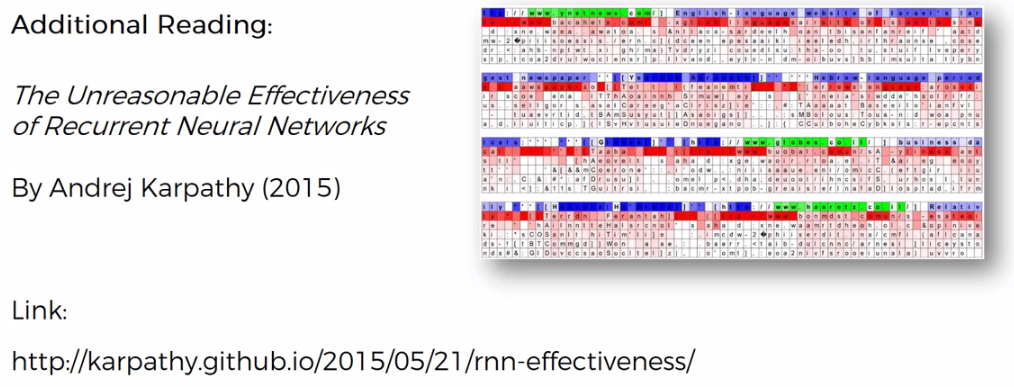




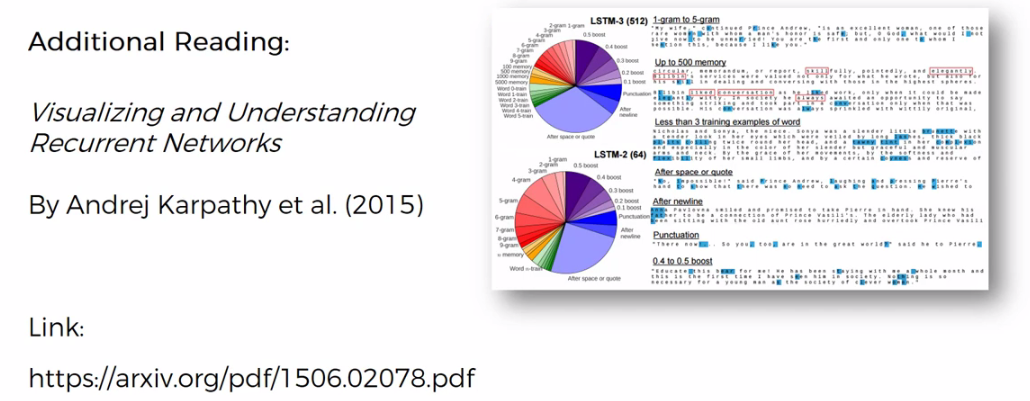
* Some most-probable-predictions (deep-red) also gets wrong. Eg: "**.com**" for "**.co.il**".
* Most of the time it can predict "www" successfully.

**12.2.8 Additional Readings**

* For more, check out his blog, karpathy.github.io. There's a couple more of these examples. And more of the previous examples that we looked at.
* Actually it is important to know what's going on inside the neural network. Because RNN, CNN, ANN those are so advanced (and complex) that we need to analyze what's going on inside those architectures (we need to study them and treat them as an alien-being !!!! ).



* Also, we've got *Andrej Karpathy* and others research paper. Which was published in *2015*. It's called *Visualizing and Understanding Recurrent Networks*. There's not too much math.



* In the paper, they're like neuroscientists trying to understand what's going on. So they open up the brain of the neural network and monitor what's happening in one specific neuron, or different neurons.
* As if they're exploring some alien, as if they're exploring some kind of extra-terrestrial being and how it thinks.
* We know that humans created these LSTMs and RNNs, these are just things that work on our computers. But because they are so *advanced* and involve so many *different* *elements* to them, they became so *complex*, we now need to study them as if they're *separate* *beings* that exists on its own.

In a few more years or maybe a decade from now, these things are going to be able to think completely on their own.

**12.2.9 Different Versions of LSTMs**

Here we have studied the *Standard version* of *LSTM*. But there could be different version. Here we are going to quickly cover off the variations of long short-term memory architectures.

|  |  |
| --- | --- |
| Standard LSTM. | Variations of LSTM |
| So here is the standard LSTM which we've discussed. | * Variation no. 1: When you add Peepholes. * Notice these lines are added, connecting these sigmoid activation functions (NN-layer-operations). * These lines providing the information about the *current* *state* of the *memory cell (memory-pipeline)* to the *NN-layer-operations* like a *peephole*. * These allow *NN-layer-operations* to decide about the valves with taking into account what is actually sitting there in the memory. |

|  |  |
| --- | --- |
| * Variation no. 2: Connected *Forget-valve* and *Memory-valve* with ***-1*** *pointwise-operator*. * Instead of having a separate decision for the *memory-valve*, now you have a *combined* *decision* for the *forget-valve* and the *memory-valve*. * Whenever you add something into memory, so whenever you close the *forget-valve* off whenever this is ***0***, *memory-valve* becomes the opposite i.e. it opened. ***-1*** *pointwise-operator* turns ***0*** into ***1***. So it makes sense to combine them sometimes. |  |
| * Variation no. 3: A very popular modification called *Gated Recurring Units*, GRUs, for short. * It completely get rid of the C-pipeline (cell-pipeline or Memory-Pipeline), and they replace it with the H-pipeline (the *Output-pipeline*), which is the *hidden-pipeline*, which we had before at the *bottom*. * It simplifies things, bit *less* *flexible*, but in terms of how many things are being *controlled* and *monitored*. It might look a bit more *convoluted*, but in reality it is a bit *simpler*. * You only have 3 valves, two are connected as well. * The constant behind it is to get rid of the memory cell (memory-pipeline) and just have this *one pipeline* to takes care of everything. |  |

* Additional reading: A good paper to check out is called "LSTM A Search Space Odyssey" by Klaus Greff and others, 2015.
* There they compared quite a few different LSTMs. You might like this research that they did.

