UNIVERSITÀ DEGLI STUDI DI SALERNO
UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Carmen Bisogni, PhD Student

C.A.S.A. Course

CONTEXT AWARE SECURITY ANALYTICS IN COMPUTER VISION
Lesson 6

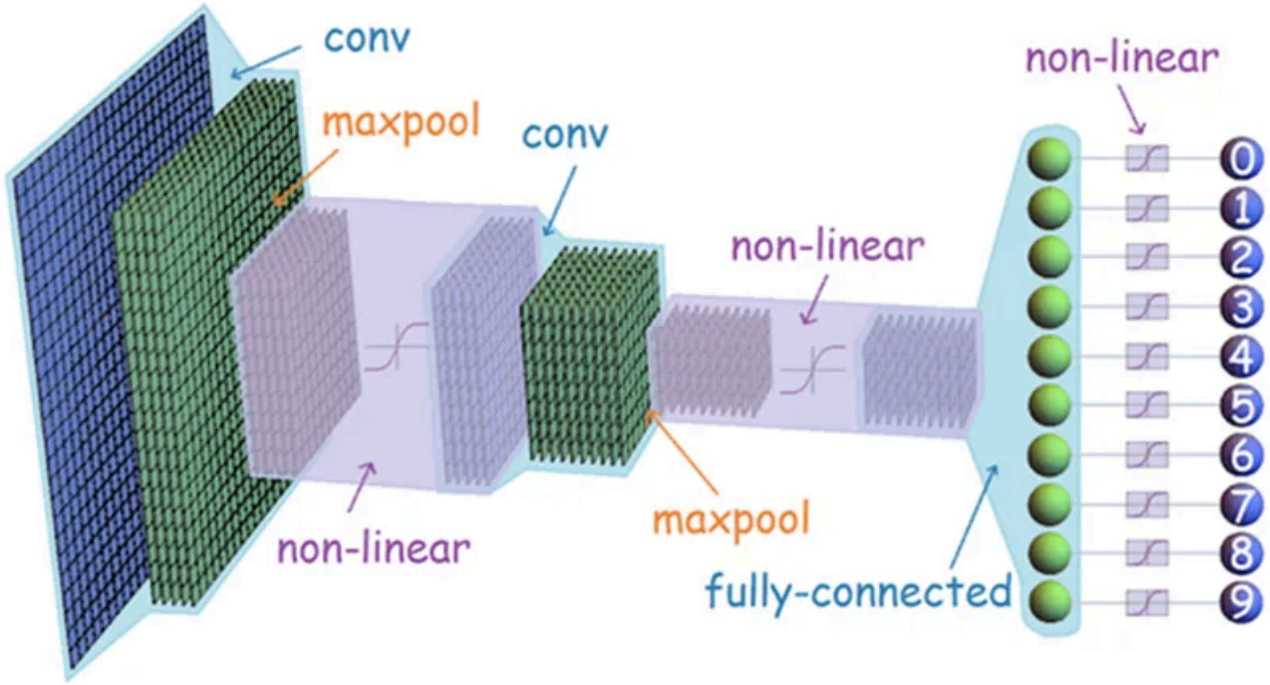
Convolutional Neural Network: Structure

A ConvNet usually has 3 types of main layers:

1)Convolutional Layer (CONV)

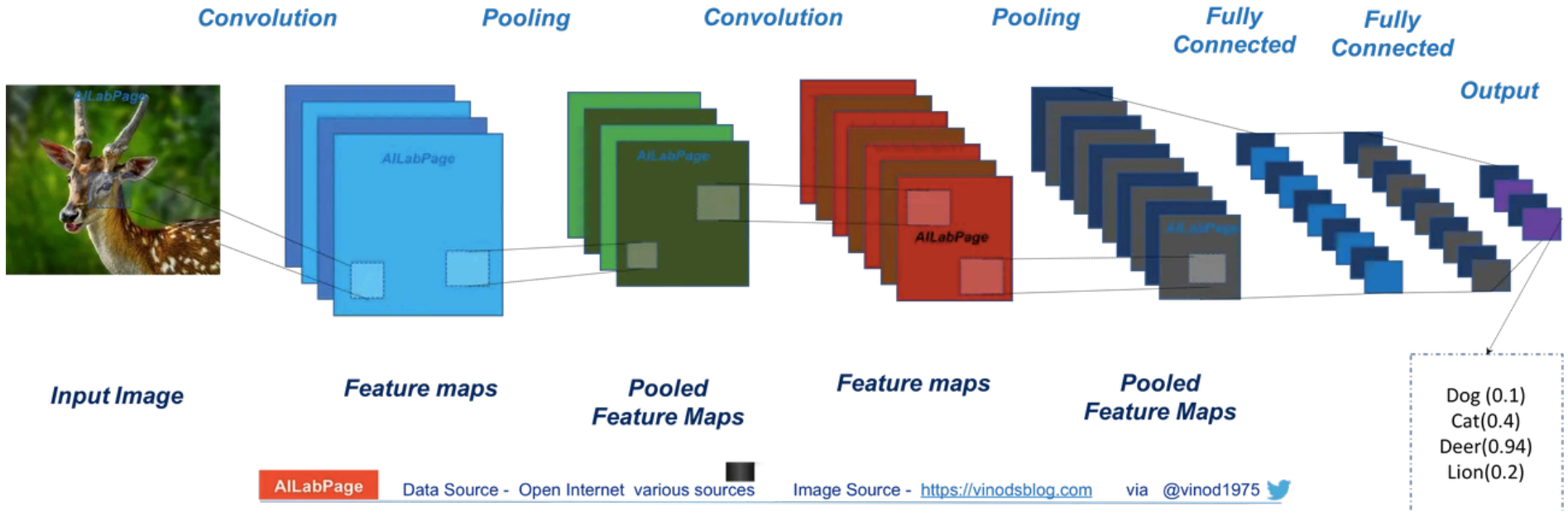
2) Pooling Layer (POOL)

3) Fully Connected Layer (FC)



Convolutional Neural Network: Structure

After passing through the fully connected layers, the final layer uses the softmax activation function (instead of ReLU) which is used to get probabilities of the input being in a particular class (classification).
And so finally, we have the probabilities of the object in the image belonging to the different classes!






What happend when the output is not a class?
What happend when the input has not fixed dimension?

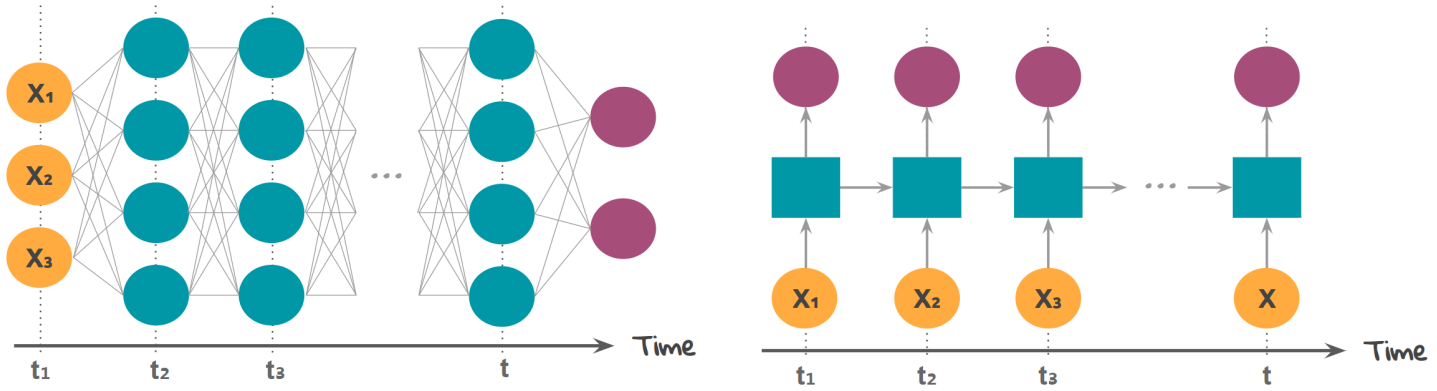
Recurrent Neural Network (RNN)

Let’s start with an example.
In human molecules, DNA(deoxyribonucleic acid) is composed of a chain of 4 different types of nucleotides: A, C, G, and T. Genetic information exists in the successive combination of the nucleotides. Therefore these orders are essential for all kinds of living things and the sequence model has high usage in predicting the uncovered part of genes.

The data is not static or discontinuous, but forgoing and successive. But what do we mean by sequential or consecutive when it comes to data?

Recurrent Neural Network (RNN)

In classical ANN, we had sample data and passed them through layers from left to right. This means that we will input the data at a time and then they will travel toward the output layer. It was feedforward propagation and had only one direction of the flow. To learn we will propagate the error back (backpropagation).
In the case of RNN, however, the data aren't inputted at the same time. As you can see the picture on the right, we will input X1 first and then input X2 to the result of X1 computation. So in the same way, X3 is computed with the result from X2 computation stage.

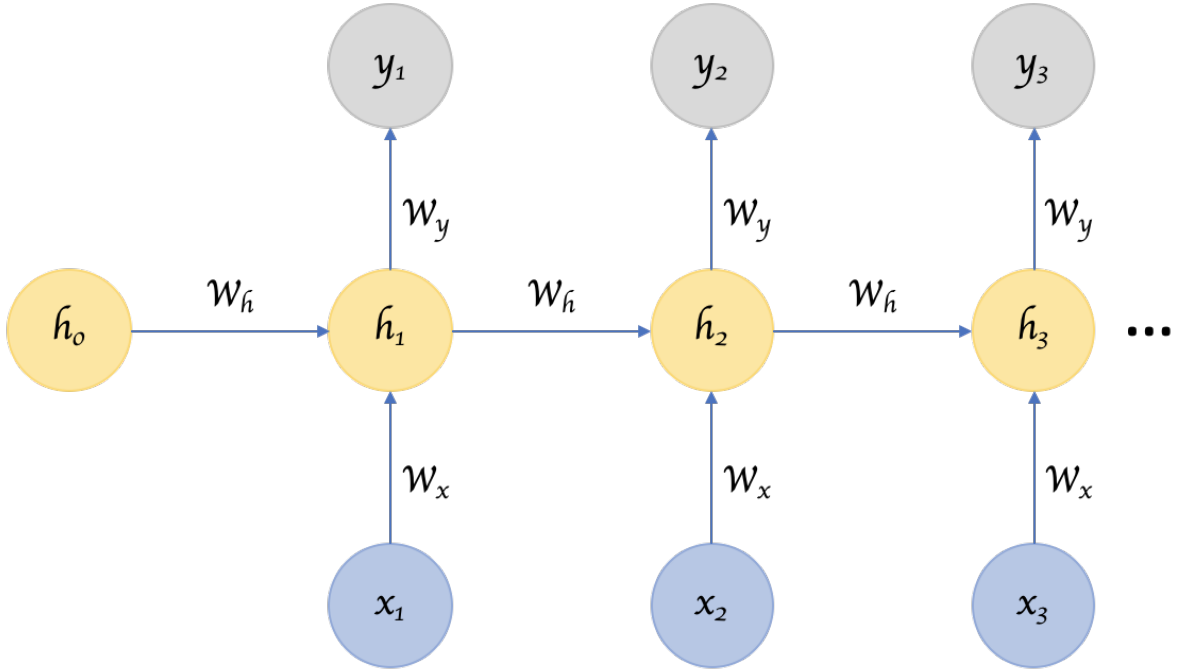


Recurrent Neural Network (RNN)

Therefore when it comes to data, ‘sequential’ means we have an order in time between the data. When it’s ANN, there isn’t any concept of order in X1, X2 and X3. We just input them at once. In the case of RNN, however, they are inputted at different times. Therefore if we change the order, it becomes significantly different. A sentence will lose its meaning. And when it comes to DNA, this change might create… a mutant.

Recurrent Neural Network (RNN)

Recurrent Neural Network remembers the past and it's decisions are influenced by what it has learnt from the past.
While RNNs learn similarly while training, in addition, they remember things learnt from prior input(s) while generating output(s). It's part of the network. RNNs can take one or more input vectors and produce one or more output vectors and the output(s) are influenced not just by weights applied on inputs like a regular NN, but also by a "hidden" state vector representing the context based on prior input(s)/output(s).



Recurrent Neural Network (RNN)

There is no pre-set limitation to the size of the vector.

Applications:

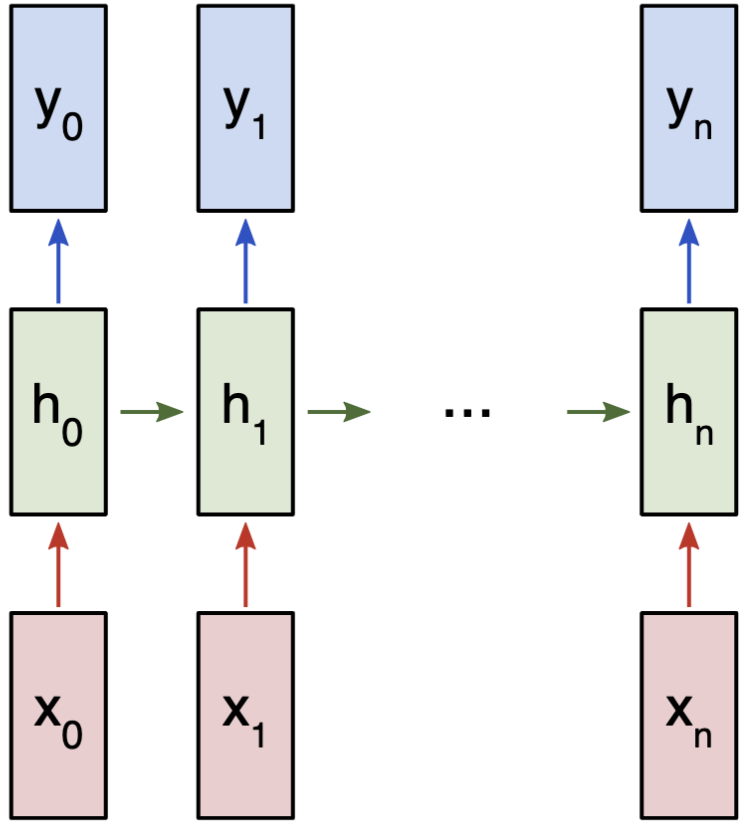
- Machine Translation (e.g. Google Translate) is done with “many to many” RNNs. The original text sequence is fed into an RNN, which then produces translated text as output.
- Sentiment Analysis (e.g. Is this a positive or negative review?) is often done with “many to one” RNNs. The text to be analyzed is fed into an RNN, which then produces a single output classification (e.g. This is a positive review).

Recurrent Neural Network (RNN)

How it works:

Let's consider a "many to many" RNN with inputs x_0, x_1, … x_n that wants to produce outputs y_0, y_1, … y_n. These x_i and y_i are vectors and can have arbitrary dimensions. RNNs work by iteratively updating a hidden state h, which is a vector that can also have arbitrary dimension. At any given step t,

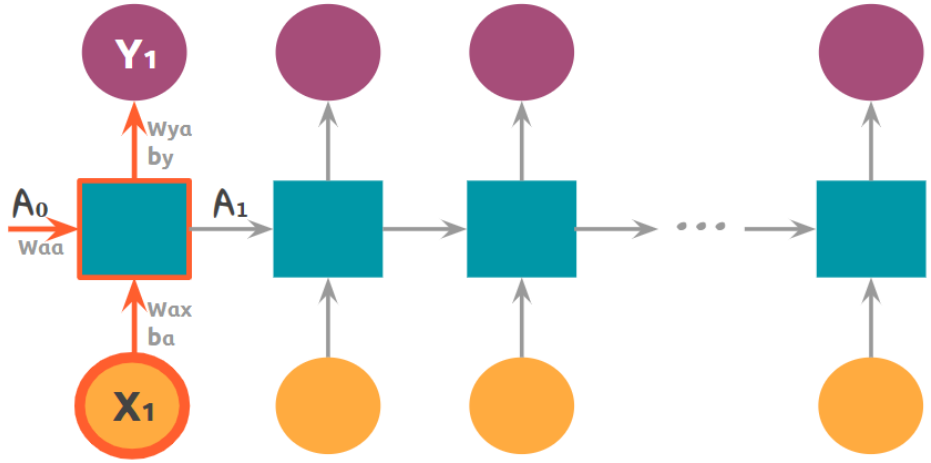
1. The next hidden state h_t is calculated using the previous hidden state h_{t-1} and the next input x_t.
2. The next output y_t is calculated using h_t.



Recurrent Neural Network (RNN)

Details:

X input
A output



- When it was ANN,

$Z_1 = W_1 \cdot X + b_1$

$A_1 = g(Z_1)$

→

- Now in RNN,

$Z_1 = W_{ax} \cdot X_1 + W_{aa} \cdot A_0 + b_a$ ····· ①

$A_1 = g(Z_1)$ ····· ②

$Y_1 = g(W_{ya} \cdot A_1 + b_y)$ ····· ③

① There are weights and bias as we did with simple ANN. It's just adding one more input value A0. And there are two different outputs from the cell. The output of A1 which will go to the next unit(②) and the final output Y1 of the unit cell(③).

Recurrent Neural Network (RNN)

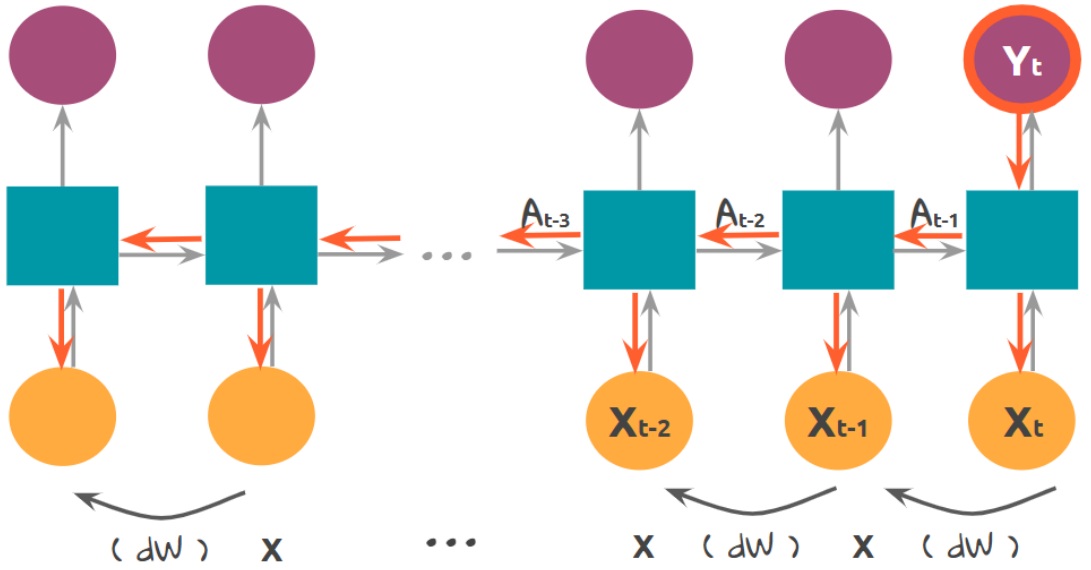
Keep in mind:
The idea of remembering the past is fantastic. But there is one critical problem in back-propagation. Back-propagation is a step for going backward to update the weights of each layer. To update the weights, we get the gradient of the cost function and keep multiplying the gradients at the given layers using chain rule. The actual backward propagation in RNN is a bit complex than this diagram.

Example: The RNN backward propagation takes not only the final output Yt but also all the other output Y used by the cost function.

What happend when the gradients are bigger than 1?
What happend when the gradients are smaller than 1?

Recurrent Neural Network (RNN)

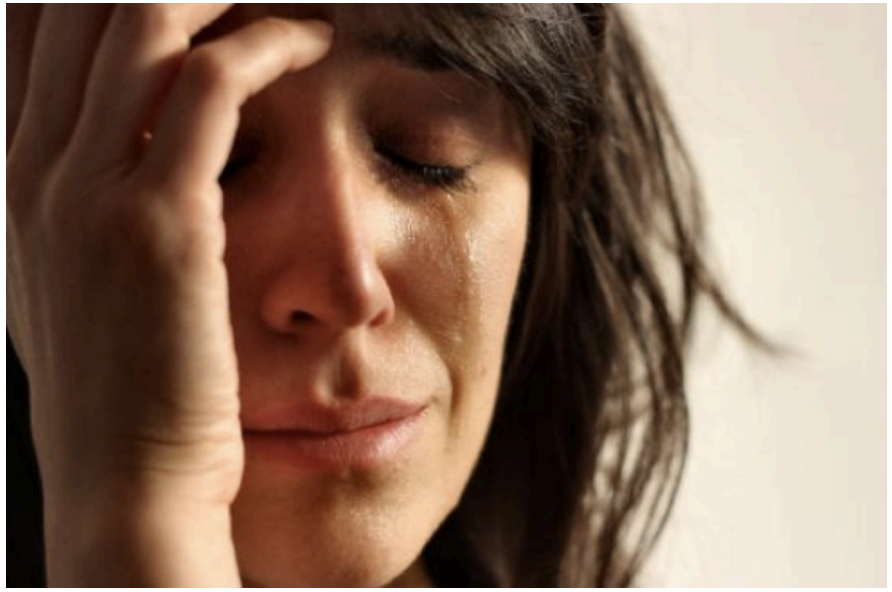
When the gradients are bigger than 1, the updated values become so big to use it for optimizing. This is called **exploding gradients**. But this isn't a severe problem because we can fix the range that gradients can't go over. The real problem occurs when the gradients are smaller than 1. If we keep multiplying the values lower than 1, the result becomes smaller and smaller. After some steps, there will be no significant difference in outcome, and it can't make any update in weights. It is called **vanishing gradients**. It means the back-propagation effect can't go far enough to reach the early stage of layers.

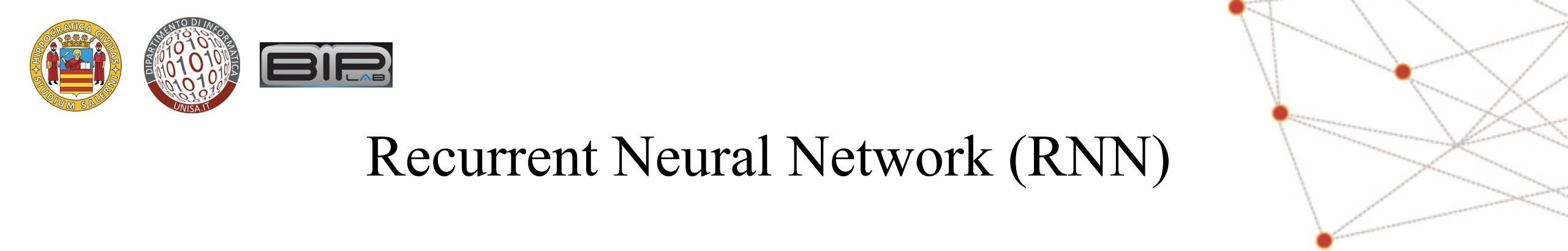


Recurrent Neural Network (RNN)

This is why people say RNN has a bad memory. If the length of the input values gets longer, we can't expect actual optimization. This is a really critical problem because the power of neural networks comes from updating weights.

Would there be other ways to fix this problem?





Recurrent Neural Network (RNN)

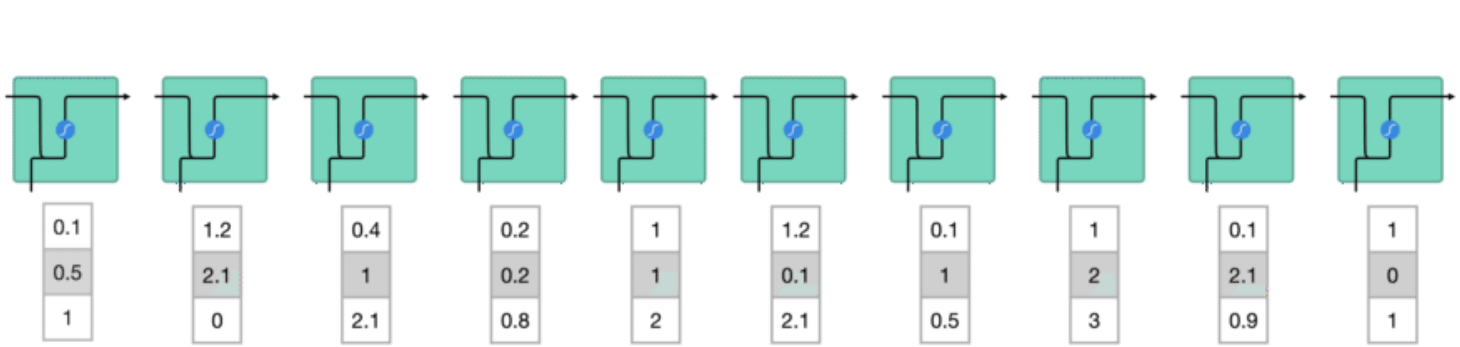
We can’t remember too many things. The same goes for our model. Instead of dragging all the past, maybe it’ll be better to remember selectively. This is like choosing only the important information and forgetting the rest. Having all those past values causes the vanishing gradients. Therefore we’ll give an additional step to simple RNN, which is called Gated Recurrent Units.

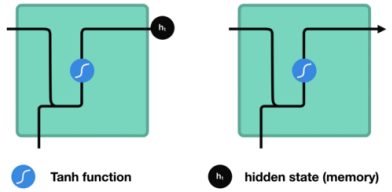
So we arrive at **Long Short Term Memory networks**.
They have internal mechanisms called gates that can regulate the flow of information.
These gates can learn which data in a sequence is important to keep or throw away.

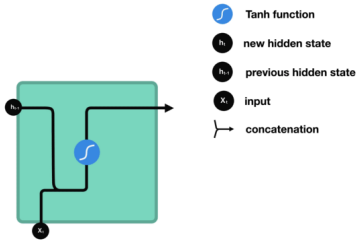
RNN vs LSTM in details

RNN:

An RNN works like this; First words get transformed into machine-readable vectors. Then the RNN processes the sequence of vectors one by one.





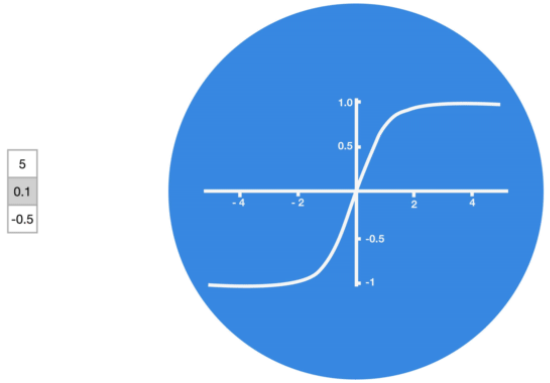


While processing, it passes the previous hidden state to the next step of the sequence. The hidden state acts as the neural networks memory. It holds information on previous data the network has seen before. The input and previous hidden state are combined to form a vector. That vector now has information on the current input and previous inputs. The vector goes through the tanh activation, and the output is the new hidden state, or the memory of the network.

RNN vs LSTM in details

RNN:

The RNN uses a tanh activation to help regulate the values flowing through the network. The tanh function squishes values to always be between -1 and 1.



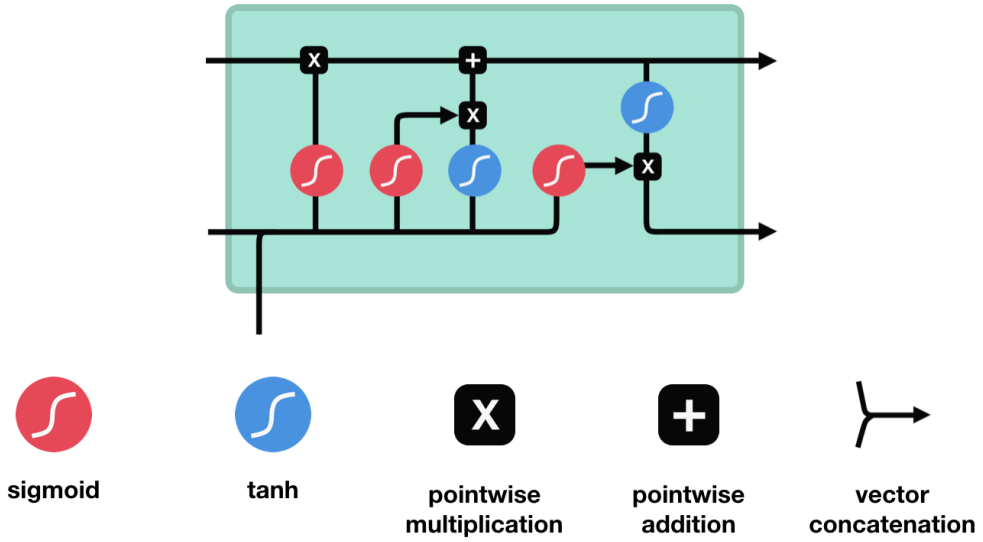
without tanh

with tanh

RNN vs LSTM in details

LSTM:

Processes data passing on information as it propagates forward. The differences are the operations within the LSTM's cells.



These operations are used to allow the LSTM to keep or forget information. Now looking at these operations can get a little overwhelming so we'll go over this step by step.

RNN vs LSTM in details

LSTM:

The core concept of LSTM's are the cell state, and it's various gates. The cell state act as a transport highway that transfers relative information all the way down the sequence chain.

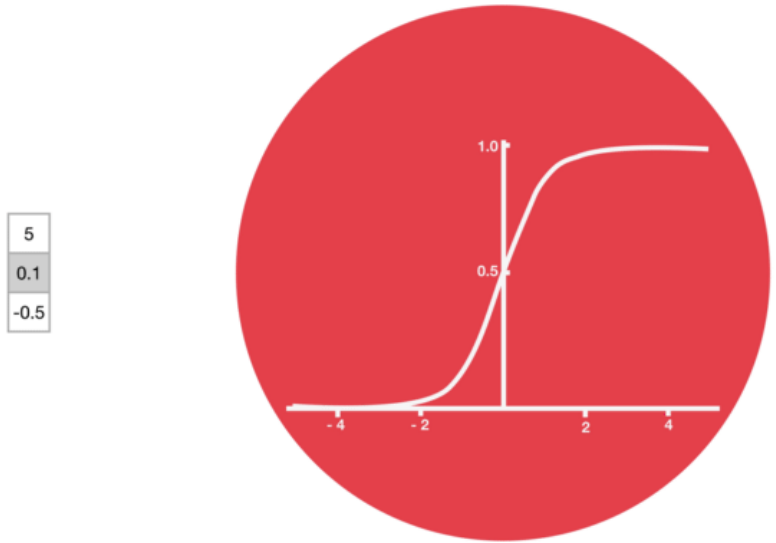
You can think of it as the "memory" of the network. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make it's way to later time steps, reducing the effects of short-term memory.

As the cell state goes on its journey, information get's added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training.

RNN vs LSTM in details

LSTM:

Gates contains sigmoid activations. A sigmoid activation is similar to the tanh activation. Instead of squishing values between -1 and 1, it squishes values between 0 and 1. That is helpful to update or forget data because any number getting multiplied by 0 is 0, causing values to disappears or be “forgotten.” Any number multiplied by 1 is the same value therefore that value stay’s the same or is “kept.” The network can learn which data is not important therefore can be forgotten or which data is important to keep.

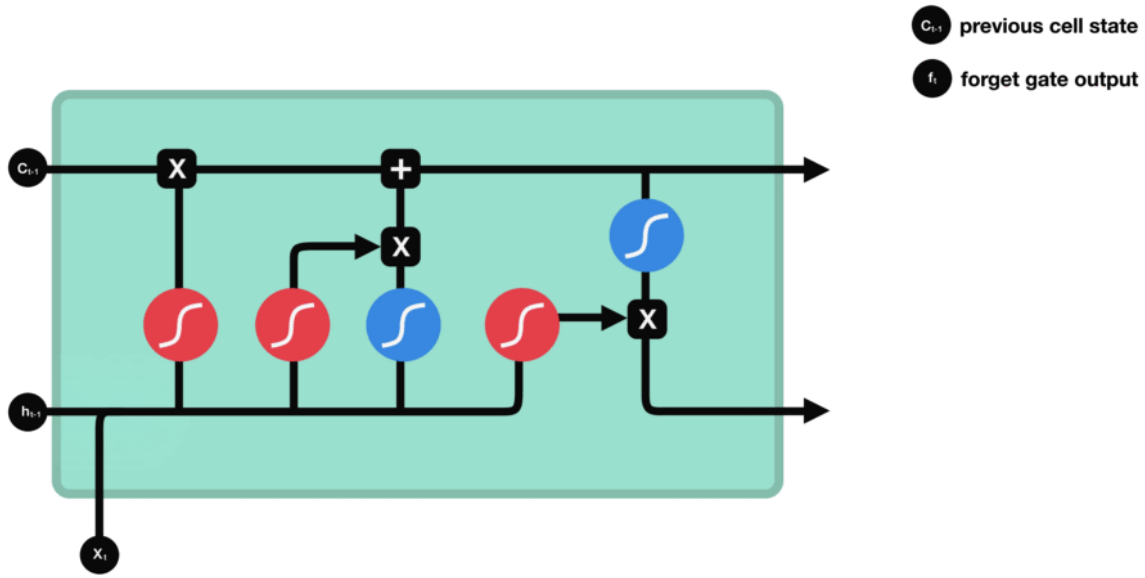


RNN vs LSTM in details

LSTM:

In LSTM we have 3 kind of gate: forget gate, input gate, and output gate.

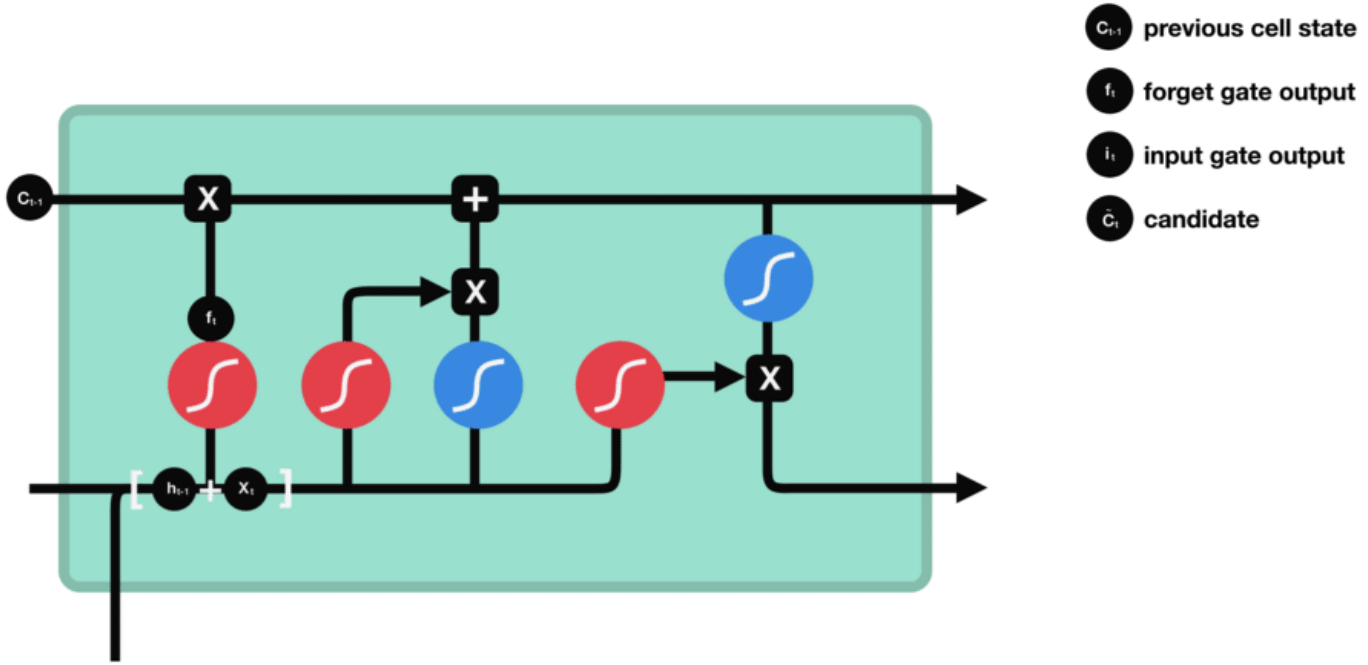
- The forget gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function.



RNN vs LSTM in details

LSTM:

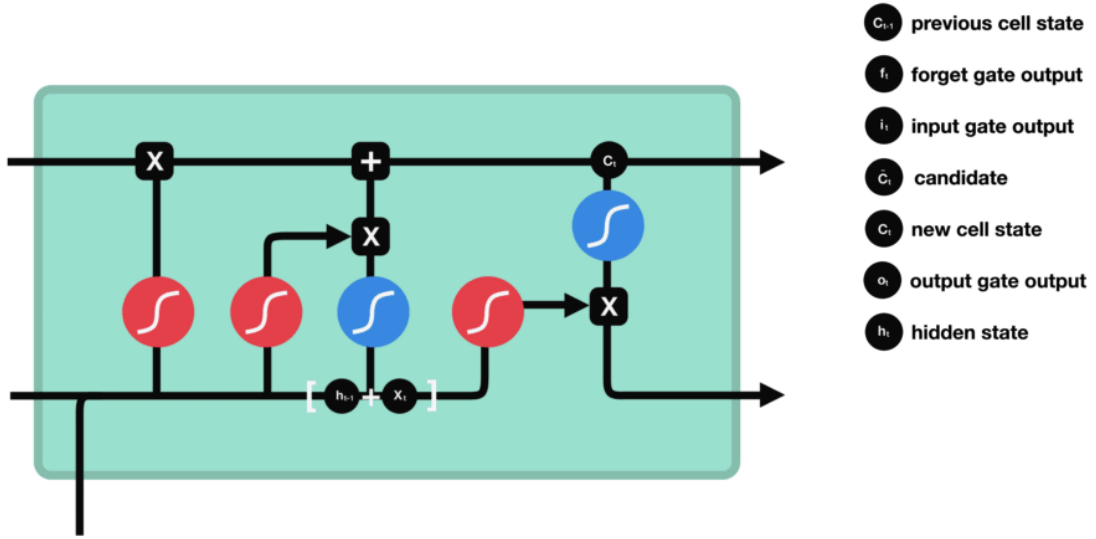
- Input gate. First, we pass the previous hidden state and current input into a sigmoid function. We also pass the hidden state and current input into the tanh function. Then we multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.



RNN vs LSTM in details

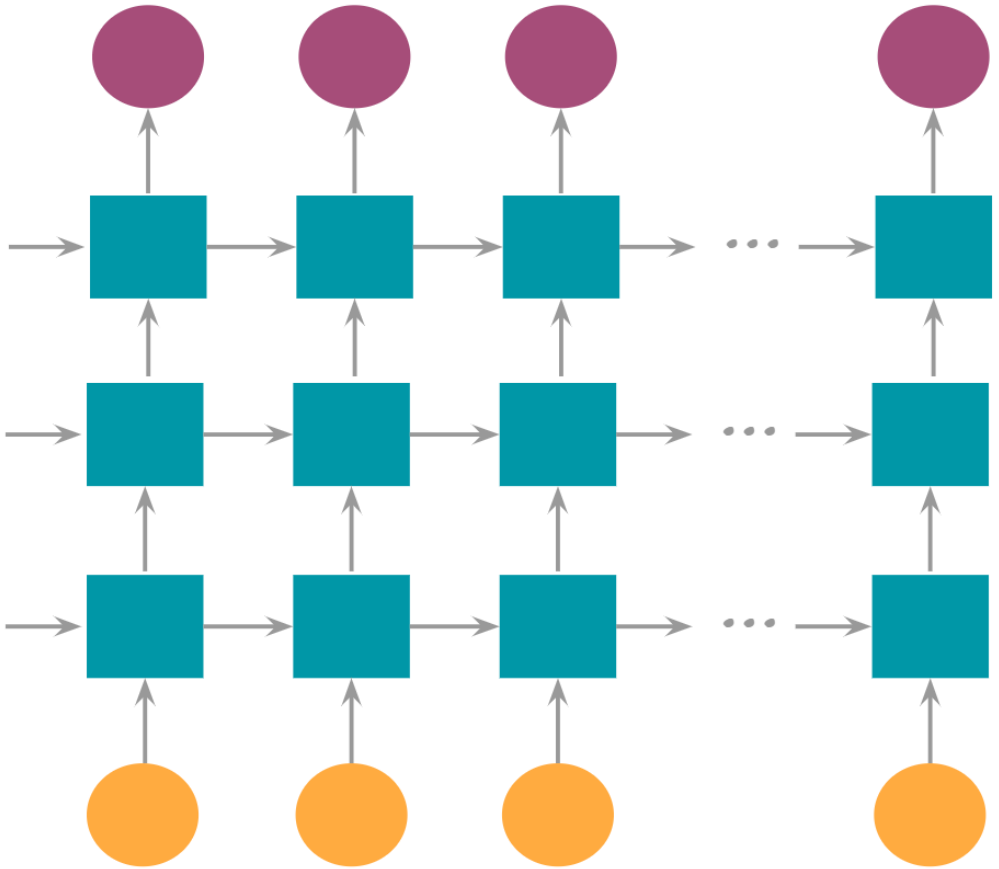
LSTM:

- Output gate. First, we calculate the new cell state by multiplying the previous cell state by the forget vector. The output gate decides what the next hidden state should be. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

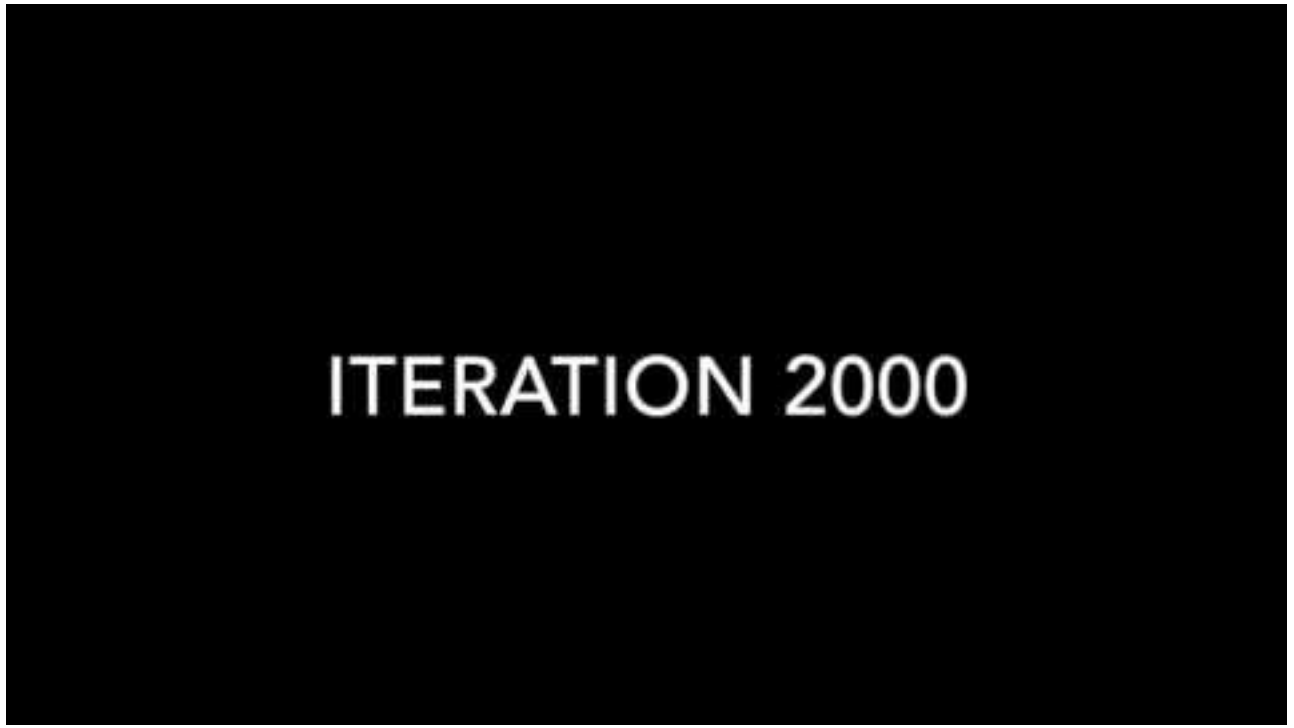


Conclusions about RNN

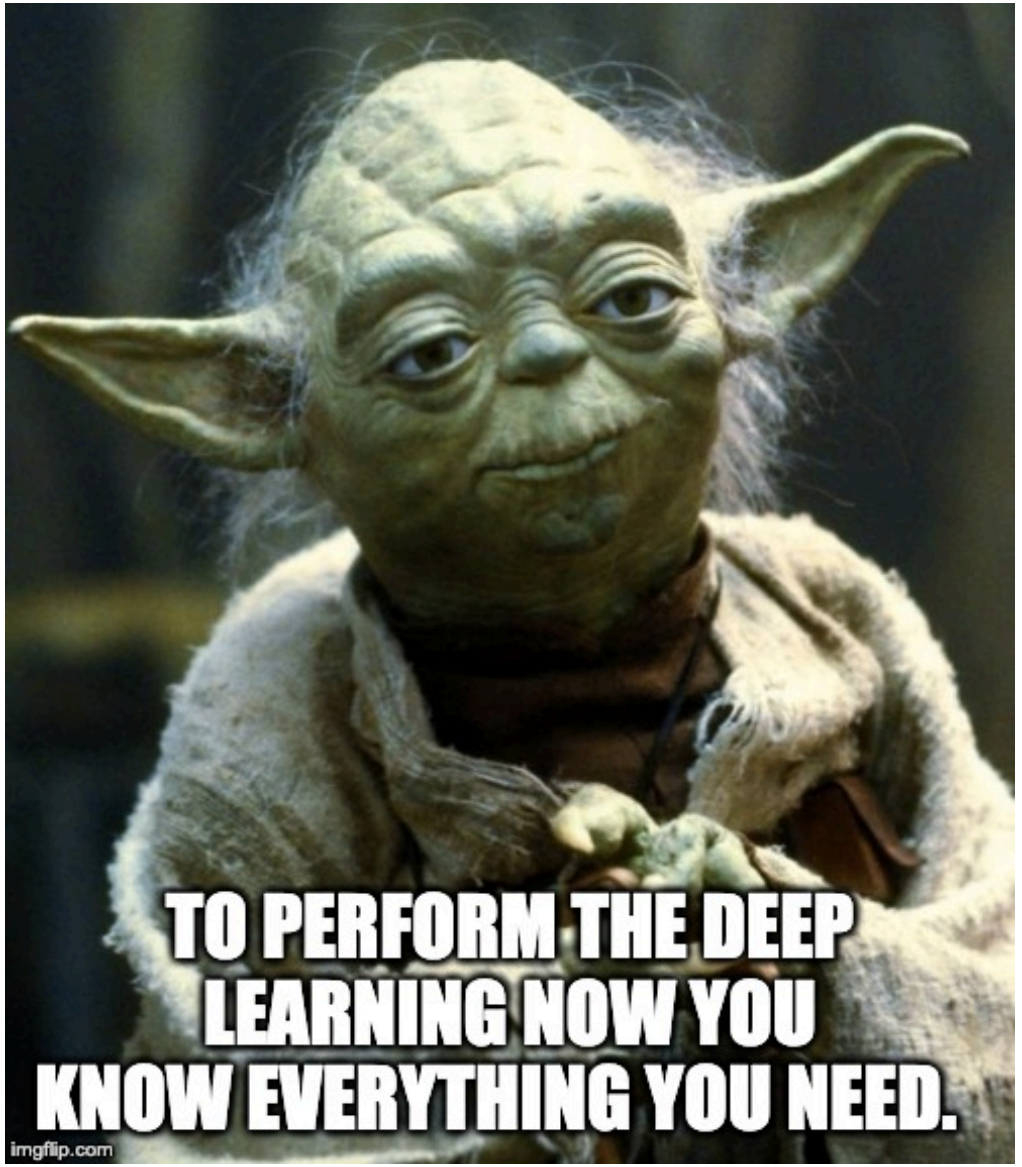
If we say adding more layers goes horizontal in ANN or CNN, it goes vertically in RNN. But the sequence model is already a big model with one or two layers, it may end up overfitting if we add a few more layers on top of that. Therefore applying normalization techniques such as dropout or batch normalization is required.



RNN Application



https://www.youtube.com/watch?v=A2gyidoFsoI


TO PERFORM THE DEEP LEARNING NOW YOU KNOW EVERYTHING YOU NEED.
imgflip.com