upGrad

#LifeKoKaroLift

Post-Graduate Diploma in ML/AI

08-03-2020

upGrad

Course : Machine Learning

Lecture On : RNN

Instructor : Manish Kumar

Session - Agenda

upGrad

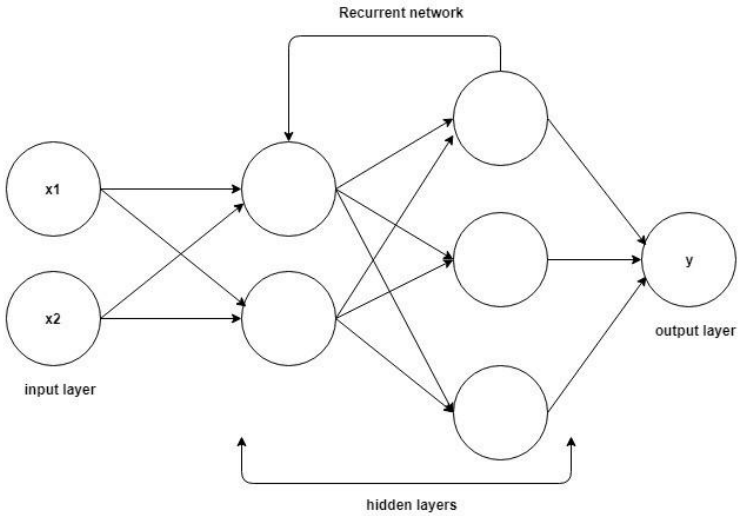
- RNN Introduction
- Industry Use-cases
- Working of RNN
- Backpropagation in RNN
- LSTM
- GRU
- Pros & Cons of each model

What is RNN

upGrad

Recurrent Neural Networks (RNNs) add an interesting twist to basic neural networks. A vanilla neural network takes in a fixed size vector as input which limits its usage in situations that involve a 'series' type input with no predetermined size.

RNNs are designed to take a series of input with no predetermined limit on size.

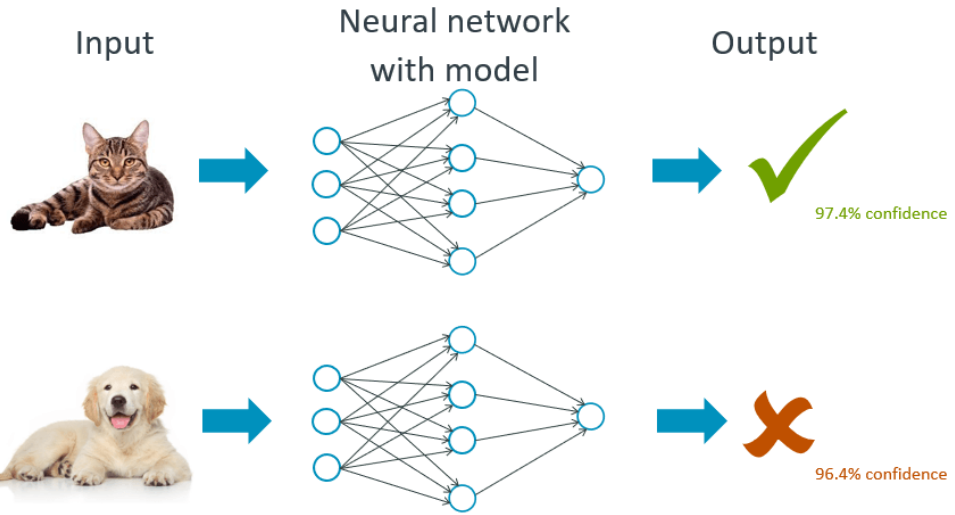


What's wrong with the Fully connected?

upGrad

A feedforward can be exposed to any collection of images, but the next image it is exposed to will not necessarily depend on the first one

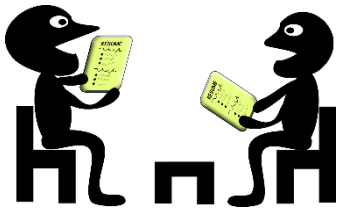
Example : If we showed an 5 cat images in a row the net won't perceive 3 dog images next right?



Industry Use-cases

upGrad

Recurrent Neural Networks, or RNNs, were designed to work with sequence prediction problems.



- Machine Translation
- Question Answering
- Language Modelling
- Text Generation
- Named Entity Recognition
- Text Summarization

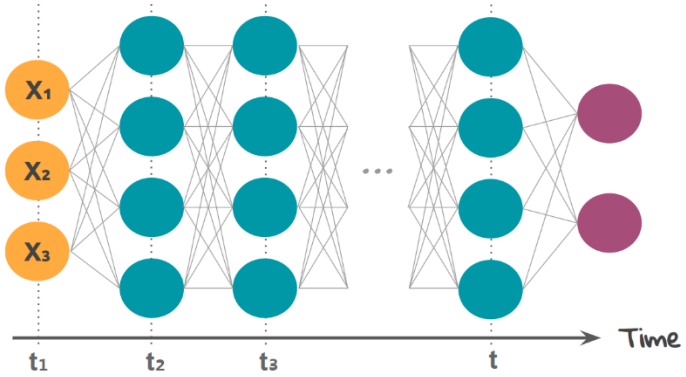
- Speech Recognition
- Speaker Verification
- Speech Enhancement
- Text-To-Speech

- Gesture Recognition
- Stock Market Prediction
- Code Generation
- Sql Chatbots

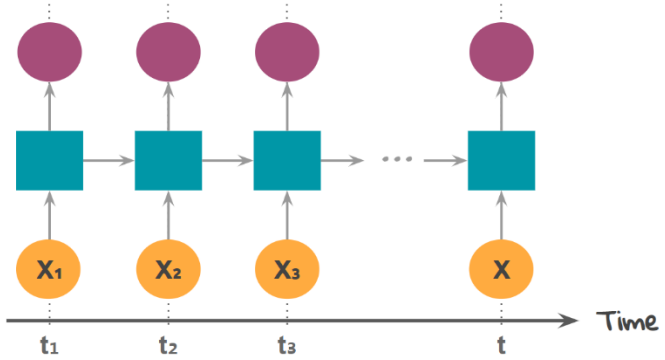
Did you notice a common characteristic ?

upGrad

Speech, text, video and time series are all "sequential." The data is not static or discontinuous, but forgoing and successive.



ANN



RNN

Did you notice a common characteristic ?

upGrad

Therefore if we change the order in a sequential data, it becomes significantly different. A sentence will lose its meaning. And when it comes to DNA, this change might create… a wolverine .

Poll Question

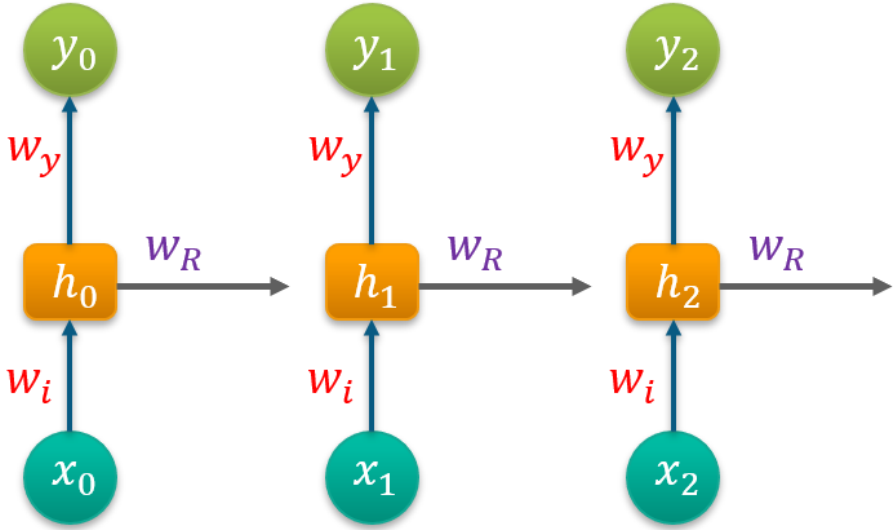
Given an n-character word, we want to predict which character would be the n+1th character in the sequence. For example, our input is “predictio” (which is a 9 character word) and we have to predict what would be the 10th character.
Which neural network architecture would be suitable to complete this task?

A) Fully-Connected Neural Network
B) Recurrent Neural Network
C) Convolutional Neural Network
D) None of the above

Poll Question

Given an n-character word, we want to predict which character would be the n+1th character in the sequence. For example, our input is “predictio” (which is a 9 character word) and we have to predict what would be the 10th character.
Which neural network architecture would be suitable to complete this task?

A) Fully-Connected Neural Network
B) Recurrent Neural Network
C) Convolutional Neural Network
D) None of the above



$$h^{(t)} = g_h\left(w_i x^{(t)} + w_R h^{(t-1)} + b_h\right)$$

$$y^{(t)} = g_y\left(w_y h^{(t)} + b_y\right)$$

w : weight matrix b : bias

Challenges with training an RNN

upGrad

RNN uses backpropagation algorithm, but for every timestep (Backpropagation through time[BTT])

RNN can suffer greatly from two problems

1. Vanishing gradients
2. Exploding gradients.

Challenges with training an RNN

upGrad

Let's take an example of text generation use-case
Input-sequence : *"I grew up in France,…….. I speak French fluently"*.

We can see from the example above that for the RNN to predict the word ***"French"*** which comes at the end of the sequence, it would need information from the word ***"France"***, which occurs further back at the beginning of the sentence.

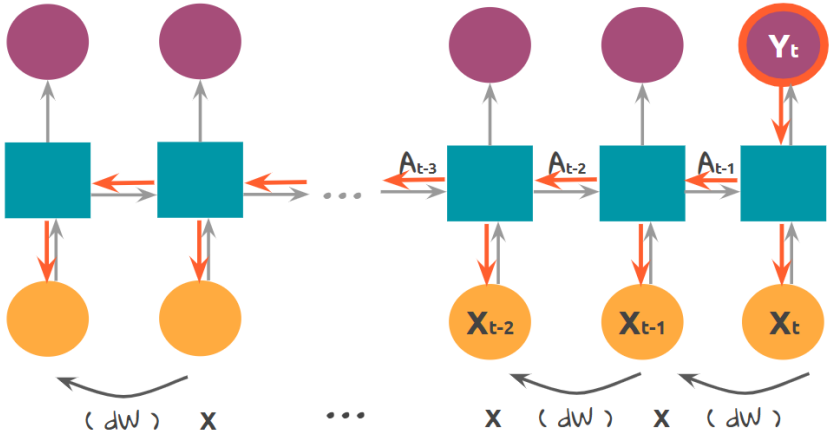
Unfortunately, in practice as this distance becomes wider, RNNs have a hard time learning these dependencies because it encounters either a vanishing or exploding gradient problem.

Challenges with training an RNN

upGrad

when the gradients are bigger than 1. The updated values become so big to use it for optimizing. This is called ***exploding gradients***.

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***.



Poll Question

Exploding gradient problem is an issue in training deep networks where the gradient gets so large that the loss goes to an infinitely high value and then explodes.
What is the probable approach when dealing with "Exploding Gradient" problem in RNNs?

A) Use modified architectures like LSTM and GRUs
B) Dropout
C) Gradient clipping
D) All of the above

Poll Question

Exploding gradient problem is an issue in training deep networks where the gradient gets so large that the loss goes to an infinitely high value and then explodes.
What is the probable approach when dealing with "Exploding Gradient" problem in RNNs?

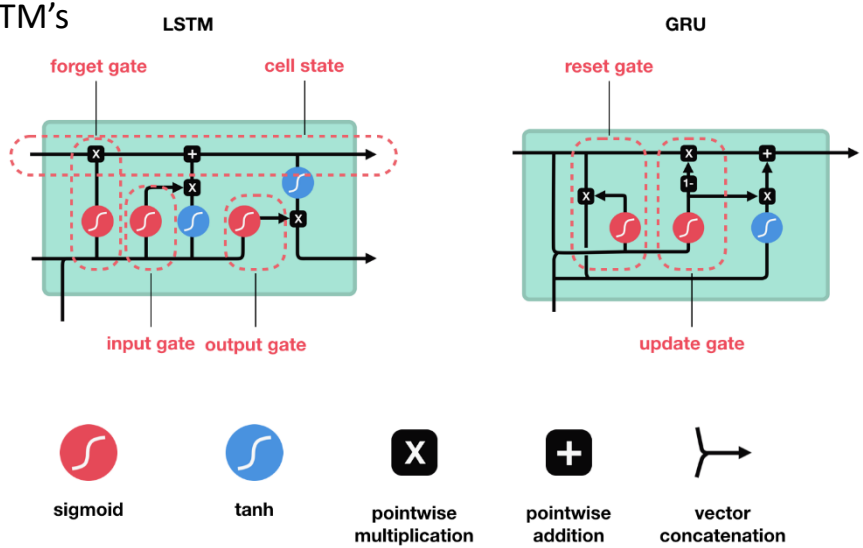
A) Use modified architectures like LSTM and GRUs
B) Dropout
C) Gradient clipping
D) All of the above

How to fix this?

upGrad

We can't remember too many things so 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 exactly what's done in GRU's & LSTM's



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gates

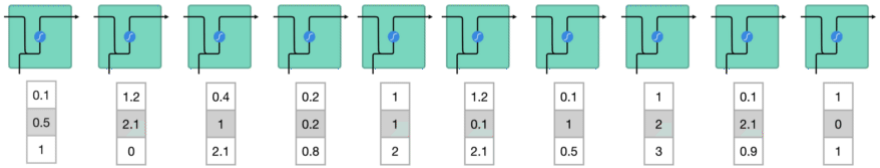
upGrad

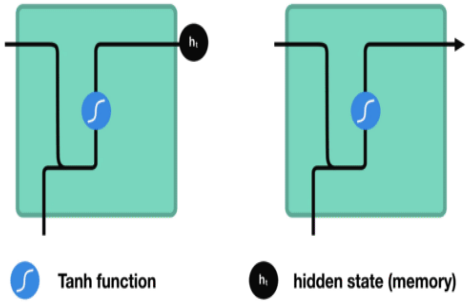
These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions. Almost all state of the art results based on recurrent neural networks are achieved with these two networks (LSTM's and GRU's).

Let's recall how RNN works

upGrad

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

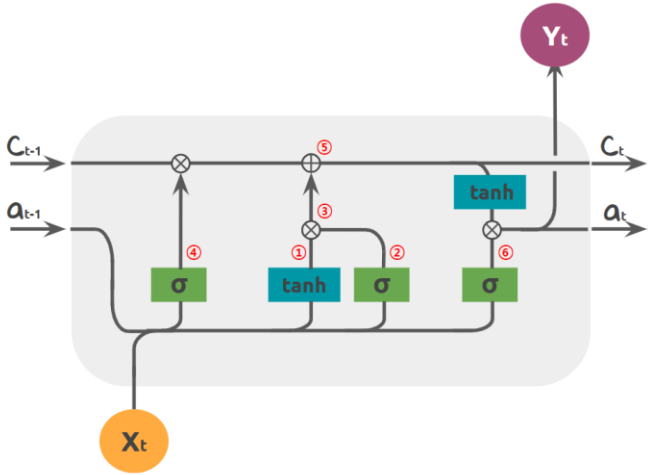




While processing, it passes the previous hidden state to the next step of the sequence. The hidden state acts as the neural networks memory. It holds information on previous data the network has seen before.

The 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.



‹ LSTM ›

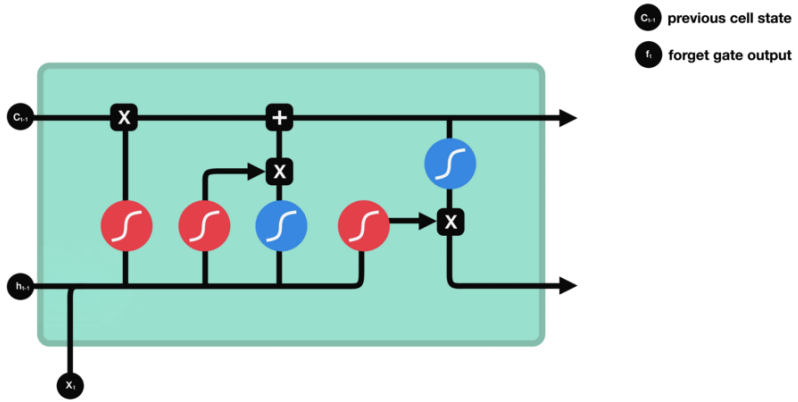
Forget Gate

upGrad

This 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. Values come out between 0 and 1.

The closer to 0 means to forget, and the closer to 1 means to keep.



Input Gate

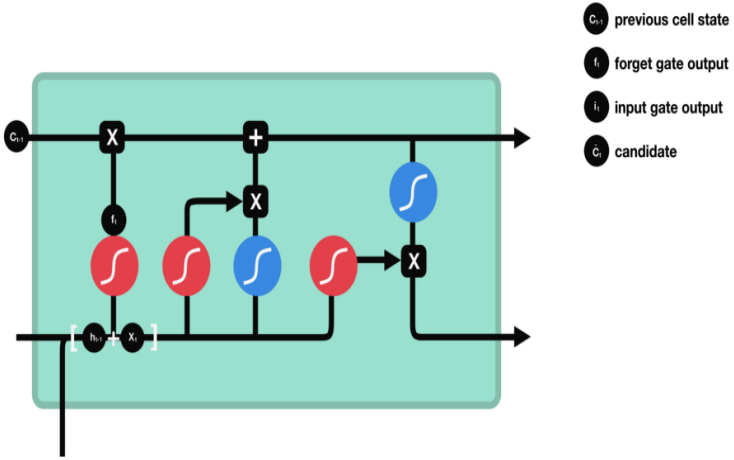
upGrad

To update the cell state, we have the input gate.

First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important.

We also the hidden state and current input into the tanh function which returns a value between -1 and 1 to help regulate the network. Then you multiply the tanh output with the sigmoid output.

The sigmoid output will decide which information is important to keep from the tanh output.



Output Gate

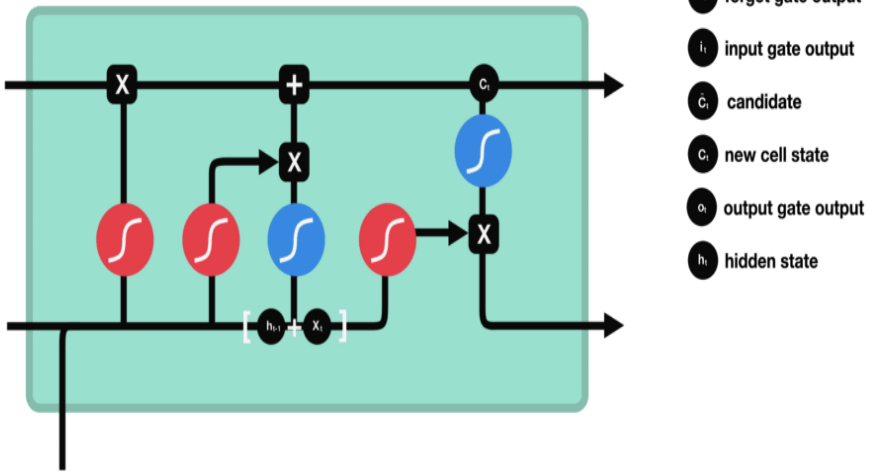
upGrad

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.

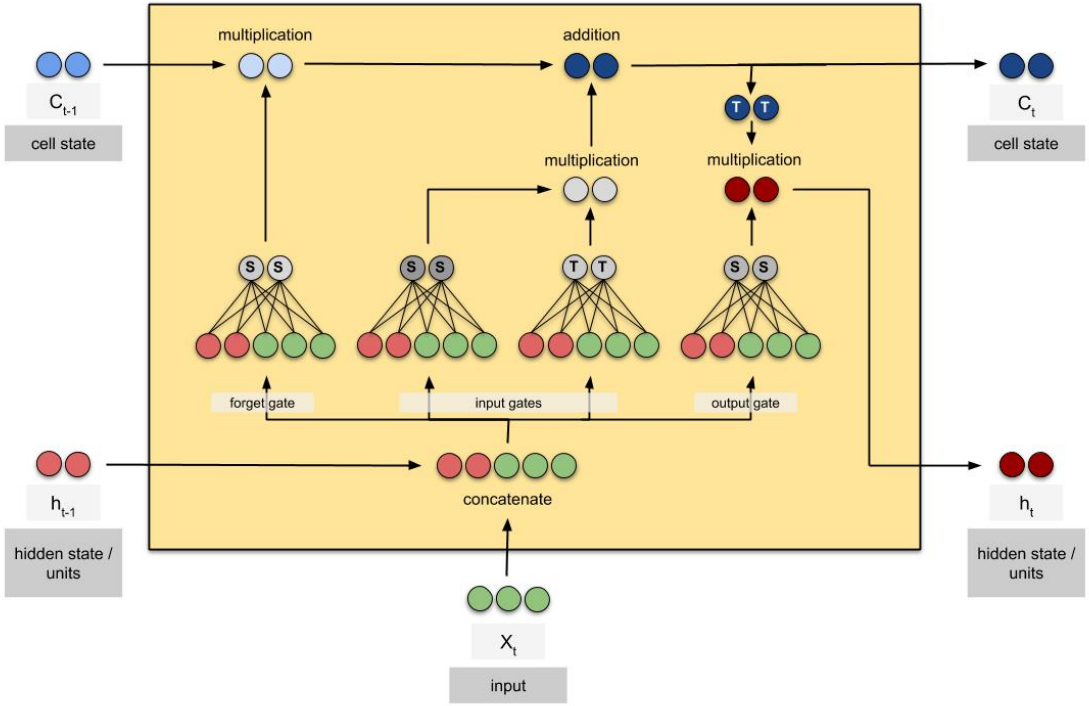
Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry.

The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

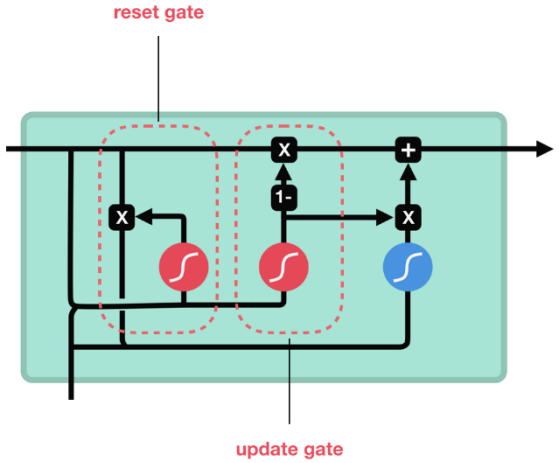


LSTM

upGrad


multiplication
addition
C_{t-1}
cell state
C_t
cell state
multiplication
multiplication
forget gate
input gates
output gate
h_{t-1}
hidden state / units
concatenate
h_t
hidden state / units
X_t
input

The GRU is the newer generation of Recurrent Neural networks and is pretty similar to an LSTM. GRU's got rid of the cell state and used the hidden state to transfer information. It also only has two gates, a reset gate and update gate.

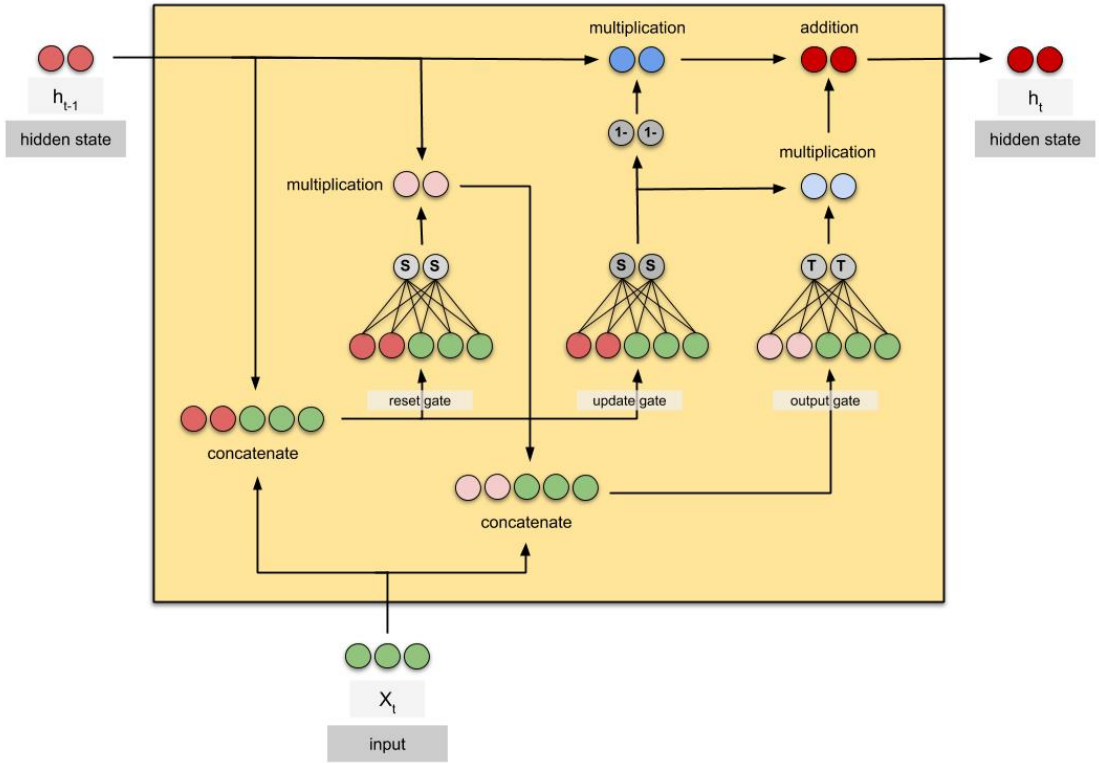


Update Gate

Similar to Forget gate in LSTM networks

Reset Gate

The reset gate is another gate is used to decide how much past information to forget.


multiplication
addition
h_{t-1}
hidden state
h_t
hidden state
1-
1-
multiplication
multiplication
S
S
S
S
T
T
reset gate
update gate
output gate
concatenate
concatenate
X_t
input

upGrad

#LifeKoKaroLift

Thank You!

Introduction to Reinforcement learning by Sutton and barto

https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf