Fundaments of Machine learning for and with engineering applications: Filters NN RNN

Reidar B. Bratvold, Enrico Riccardi[1]

Department of Energy Resources, University of Stavanger (UiS)[1]

Apr 16, 2024

University of Stavanger

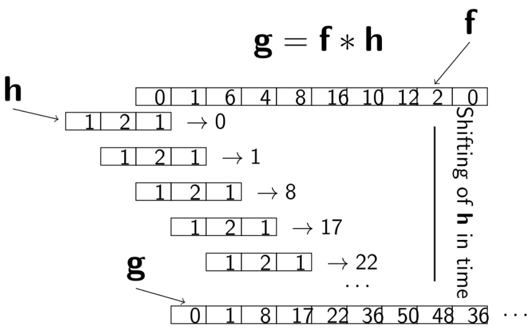
Filtering

Convolution

Convolution involves a window function that changes another function by *sliding* over it and performing local multiplications and additions. Depending on the shape of the convolution function we can perform

- Smoothings
- Deformations
- Differentiations

Convolution



Convolution

In general we can write the convolution between a function f and a convolving (deforming) function h as:

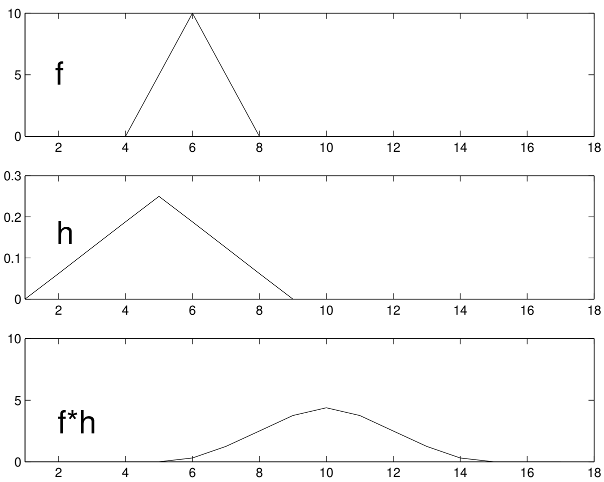
$$g(t) = \sum_{m=-\infty}^{\infty} f(m)h(m-t)$$

Often we use a more compact notation:

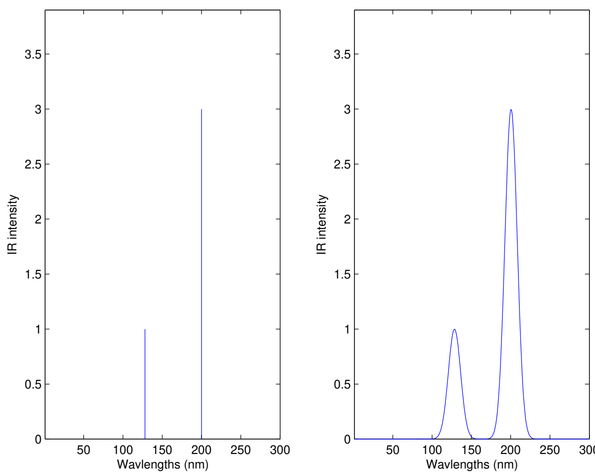
$$g(t) = f(t) * h(t) = h(t) * f(t)$$

where $*$ is the convolution operator

Convolution function



Peak broadening



Convolution operator properties

The convolution operator follows the distributive rule:

$$f_1(t) * [f_2(t) + f_3(t)] = f_1(t) * f_2(t) + f_1(t) * f_3(t)$$

It also follows the associative rule regarding order:

$$f_1(t) * [f_2(t) * f_3(t)] = [f_1(t) * f_2(t)] * f_3(t)$$

Convolution operator properties

Repeated convolutions

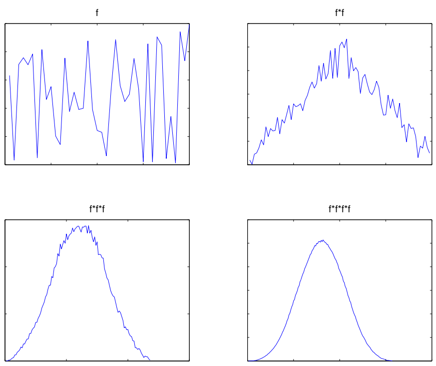
Take any function $g(t)$ and convolve it with any function $f(t)$ multiple times:

$$\begin{aligned} a_1(t) &= g(t) * f(t) \\ a_2(t) &= g(t) * a_1(t) \\ a_3(t) &= g(t) * a_2(t) \\ &\vdots \\ a_n(t) &\rightarrow \text{gaussian}(t) \end{aligned}$$

i.e. the result with always converge to a Gaussian function

Convolution properties

Any signal convolved with itself repeated many times will converge to a Gaussian function:



Mean Smooth operator

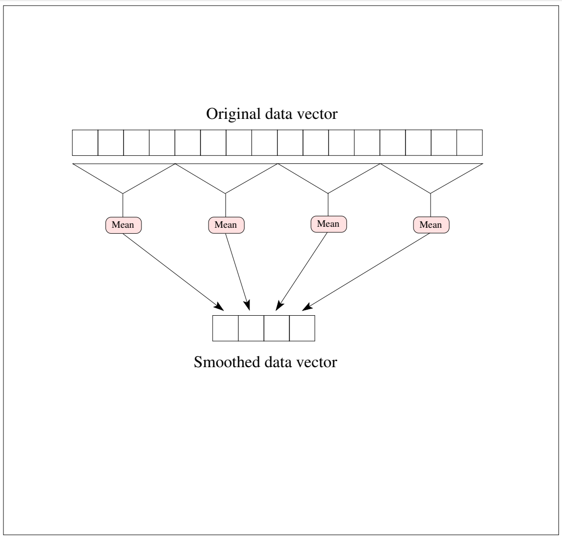
This is a very simple method which works as follows:

- Assume data vector x which contains n data points
- Start with the k first points, i.e. $[x_1, x_2, \cdots, x_k]$ and compute mean u_1 of these k points
- Take the next k points, $[x_{k+1}, x_{k+2}, \cdots, x_{2k}]$ and compute the mean u_2 of these k points
- Continue with this process until the data vector x is exhausted of points

There are two effects of this preprocessing:

- The new data vector u is of length approximately $1/k$'th of compared to the original
- Each element u_j has less noise due the cancelling effects of computing the mean

Mean Smooth operator



Running Average Smooth operator

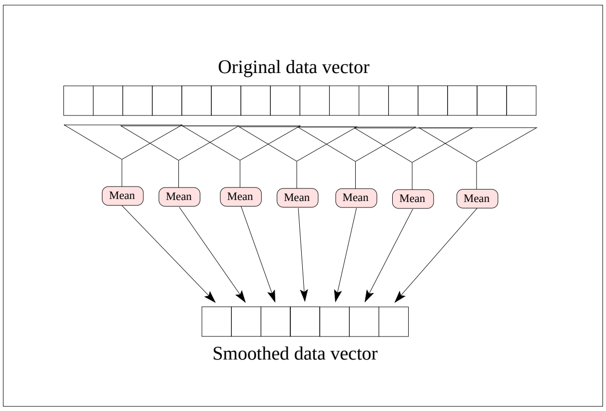
Let f be the original data profile and g the smooth version of this profile. Then we have:

$$g(i) = \sum_{j=-m}^{m} \frac{f(i+j)}{2m+1}$$

where m is the number of points in the window

Running Average Smooth operator

This is similar to the mean smoother but moves in shorter steps than the whole window length



Convolution or Moving average?

If we have a window with 3 points ($m = 1$) and we want to calculate the new value of point no. 5 in the original profile:

$$g(5) = [0 \cdot f(3) + 1 \cdot f(4) + 1 \cdot f(5) + 1 \cdot f(6) + 0 \cdot f(7)] \cdot \frac{1}{3}$$

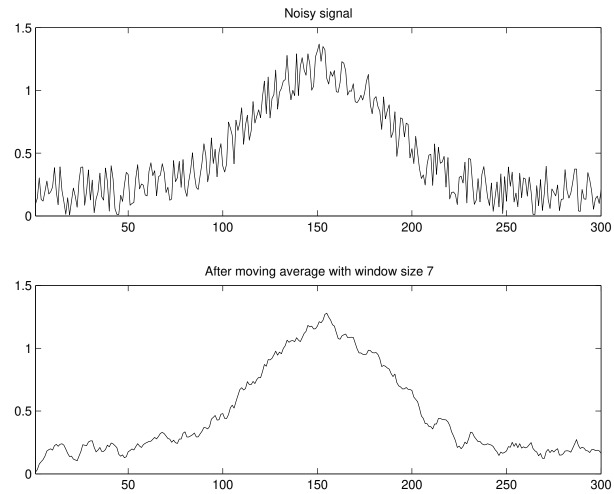
$$g(5) = \frac{1}{3}[0 \quad 1 \quad 1 \quad 1 \quad 0] \cdot [f(3) \quad f(4) \quad f(5) \quad f(6) \quad f(7)]^T$$

Convolution or Moving average?

A moving average IS the convolution between the vector **f** and a vector of ones (times a constant), i.e. :

$$\text{moving average} = \mathbf{f} * \mathbf{h} = \mathbf{f} * \frac{1}{n}[1 \quad 1 \cdots 1 \quad 1]$$

Moving average



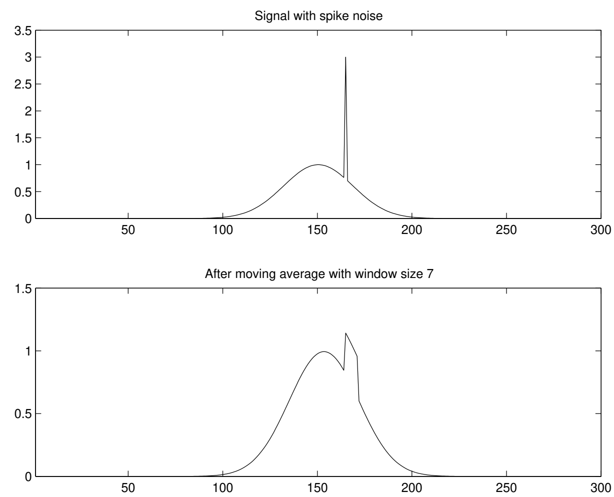
Moving average problems

- Broadening of peaks
- *Spike-noise* affects the smoothed profile

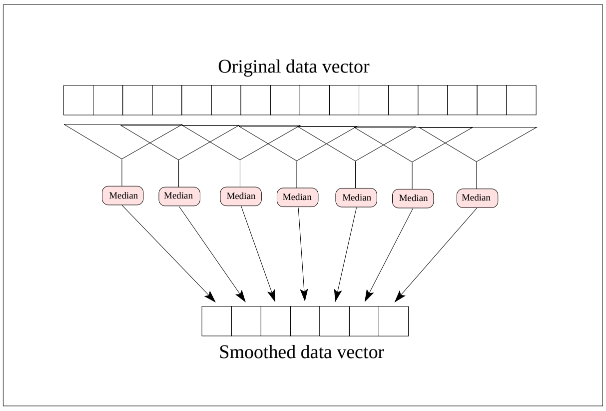
A better alternative:

Use the median instead of the mean, then we don't have the problems with *spike-noise*. However, this filter is not linear

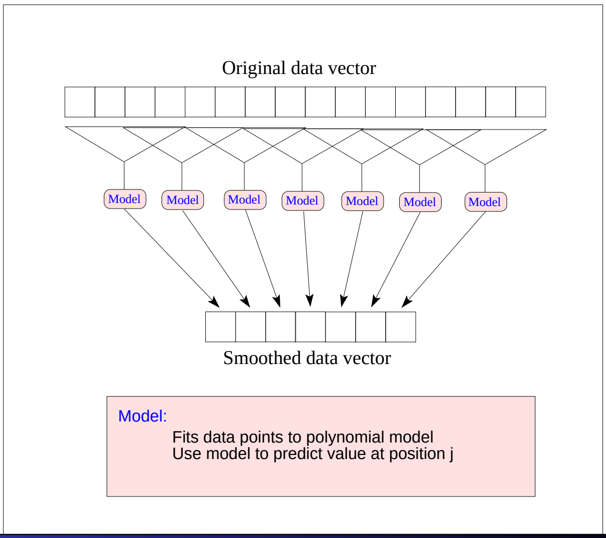
Running average example



Running mediam



Polynomial smoothing



Neural Network -NN-

One of the most famous models in Machine Learning is **Neural Network**.

The name comes from how the brain functions: a set of connected neurons that are either off or active.

In its essence,

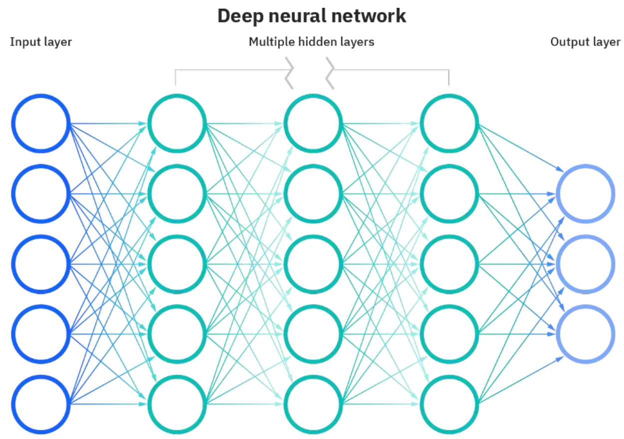
> NN is a large set of linear regressions executed both in parallel and in series.

Conventional NN are composed by:

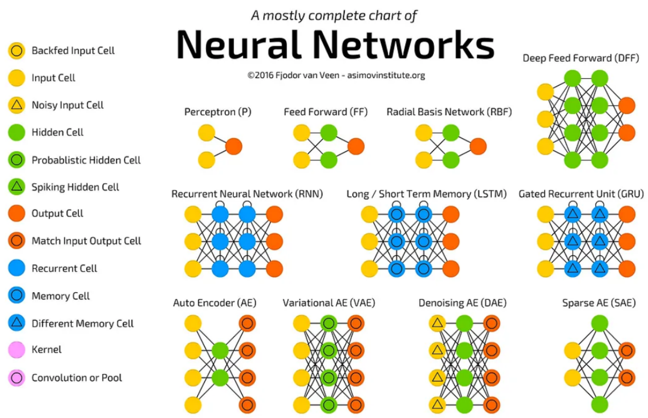
1. Input layer
2. Hidden layers
3. Output layers

which, together, can approximately approximate any function in any dimension.

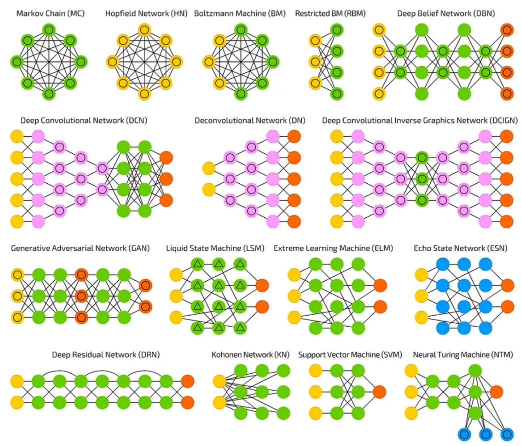
deep Neural Network



NN Architecture



NN Architecture



NN Architecture

Neural net suffers from overfitting problems.

Even more parameters

The number of nodes, the architecture, the number of hidden layer etc are NN hyperparameters

Unfortunately, at the current status of knowledge, the best architecture can be found only by trial and error.

The best fitting NN usually have a poor validation (generalizability).

NN is a very expensive approach and it should be used only if really needed.

NN types

There are many tipes of NN:

- ANN (artificial NN), just another name for NN
- DNN (deep) deep neural network
- RNN (recurrent NN) for audio
- CNN (convolutional NN) for images
- Autoencoder (for PCA) to compress to a latent space and decompress data
- Deep autoencoder (for interpretability)
- Physics informed NN (to merge NN to differential equations)
- ... and more ...

tensorflow, pytorch and keras are the most popular and popular libraries for NN.

Sequential Data

ORDER MATTERS

- Language Models
- Time series

Language model

- Prediction of the next word
- Prediction of next sentence

Time Series

- Weather data
- Stock market
- Monitoring
- Trajectories
- Etc

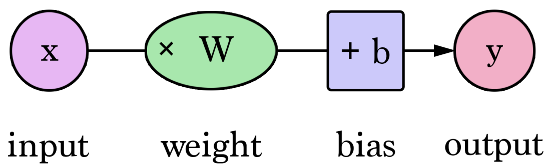
Feedforward (FF) vs Recurrent NN (RNN)

FF network

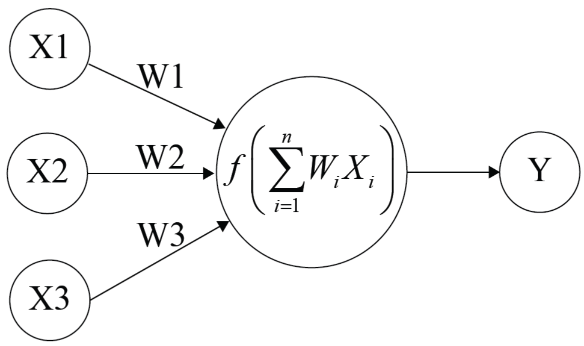
- One set of input
- One set of output
- Different parameters at each layer

- Multiple input set
- Multiple output
- Same parameter set

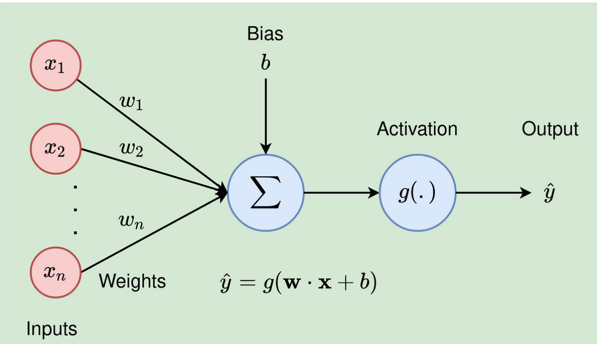
NN



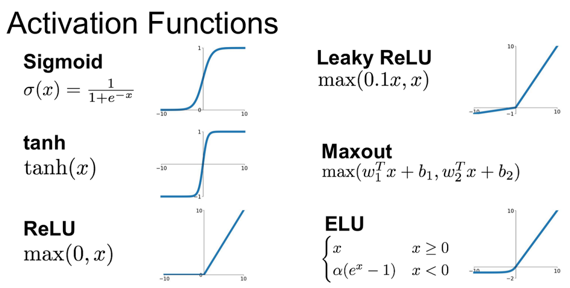
NN



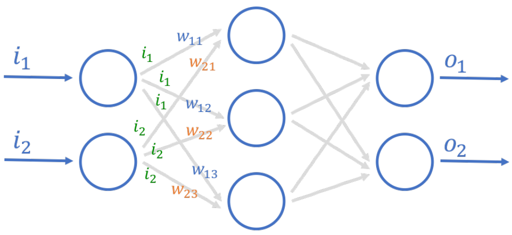
NN



NN

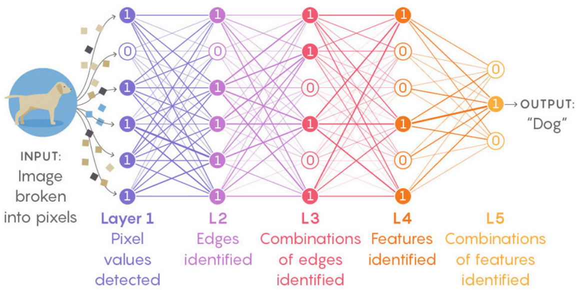


NN

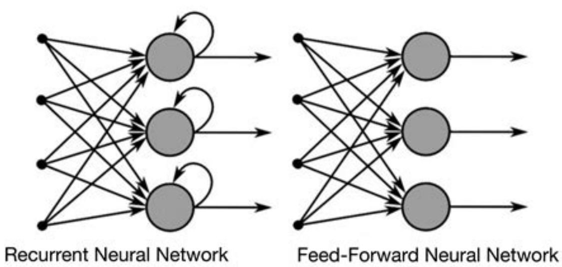


$$\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} (w_{11} \times i_1) + (w_{21} \times i_2) \\ (w_{12} \times i_1) + (w_{22} \times i_2) \\ (w_{13} \times i_1) + (w_{23} \times i_2) \end{bmatrix}$$

NN



RNN



RNN

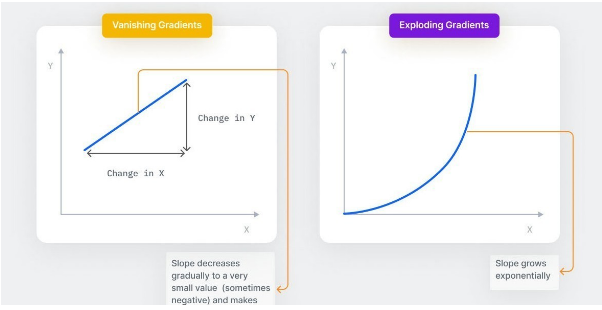
Advantages

- Model size is fixed
- Each info is stored/learned
- The weights can be forwarded

Problems

- Computationally demanding: long training times
- Problematic with Long series
- It can diverge (explode) or gradient vanish
- It cannot be very deep
- Unable to handle long time dependencies

RNN problems



RNN problems

Exploding gradients

- Large weights updats
- Gradient descent diverge (solution method)

Vanishing gradients

- Weights get margnially upgraded
- Very slow convergence speed

LSTM: Long Short Term Memory

NB...

Filters forget data...

What about we porpousely forget data?

/pause LSTM includes Forget Gates

Automatic filter!

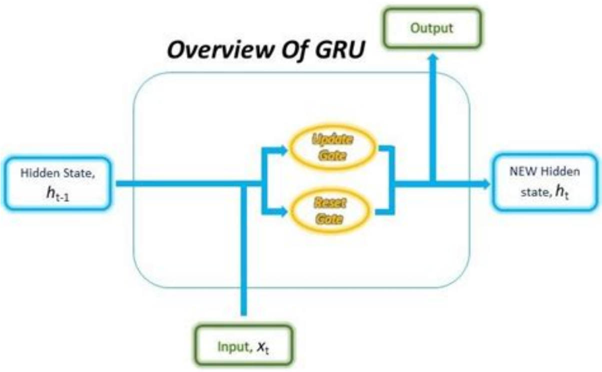
The forget gates learn to forget what is not interesting

This is extremely useful but also rather worryome: you have no control!

LSTM is an advanced version of GRU (Gated Recurrent Units)... what is GRU?

GRU

GRU



GRU

Reset gates

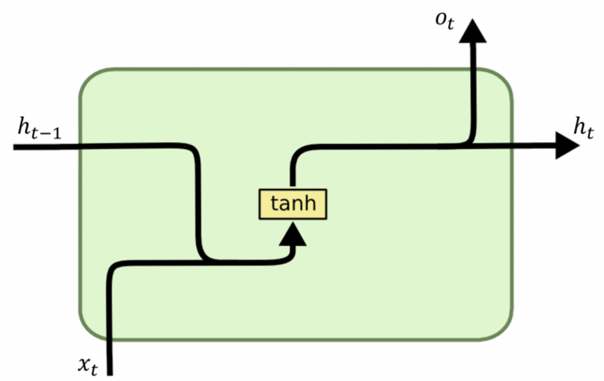
To capture short-term dependencies

Update gates

To capture Long-term dependencies

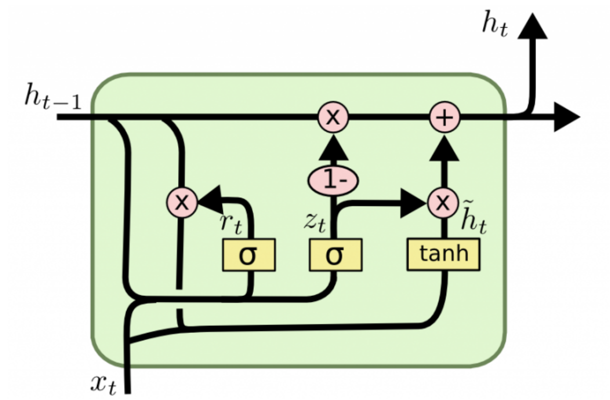
Each gate has its own weight

RNN



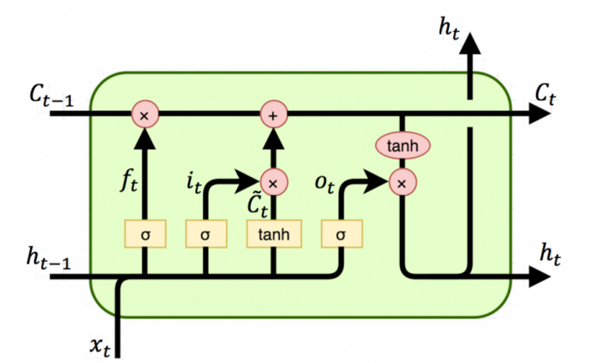
x_t: input vector, h_t: hidden layer vector o_t: output vector

GRU



x_t: input vector, h_t: hidden layer vector o_t: output vector, r_t: reset factors, z_t: update factors

LSTM



x_t: input vector, h_t, C_t: hidden layer vector o_t: output vector, r_t: reset factors, z_t: update factors