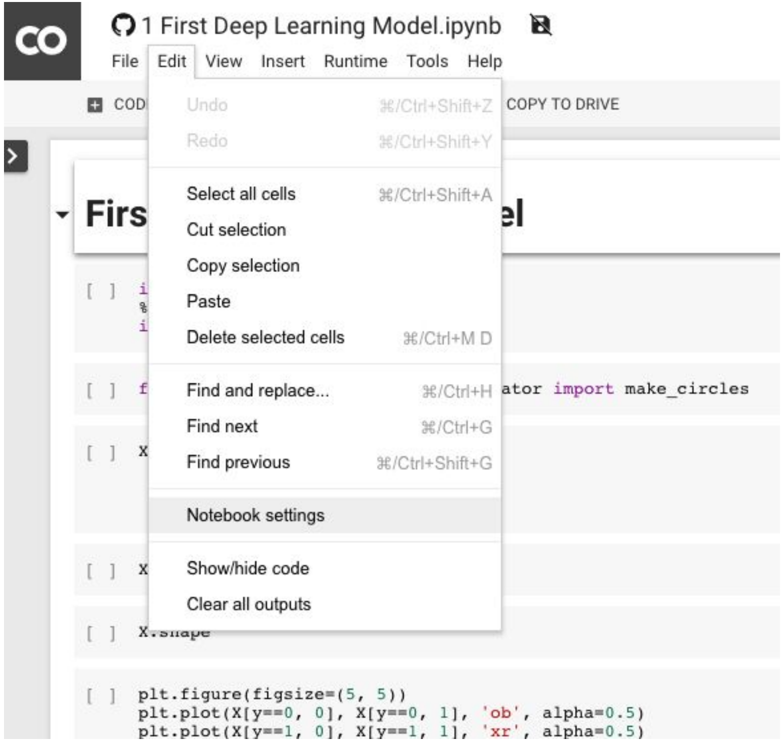
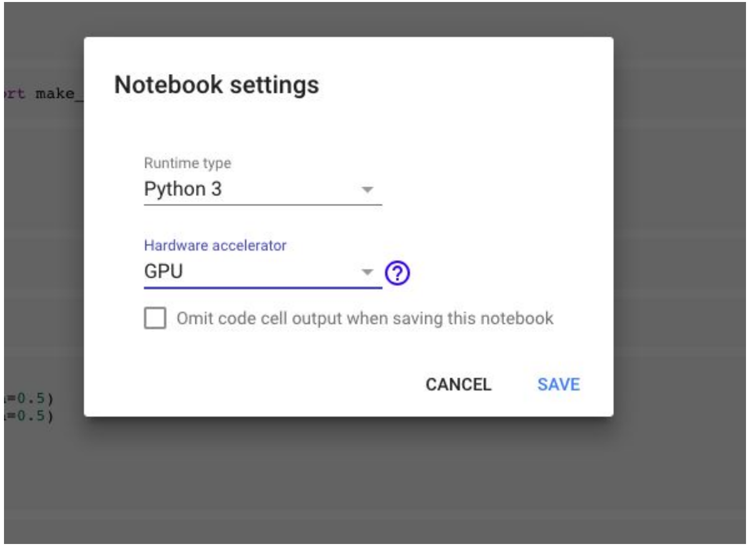
RNN

GPUs on Colab





2 Types of NN that we care about

- CNN and RNN

Outline

- Finish MNIST
- Compare to MLP

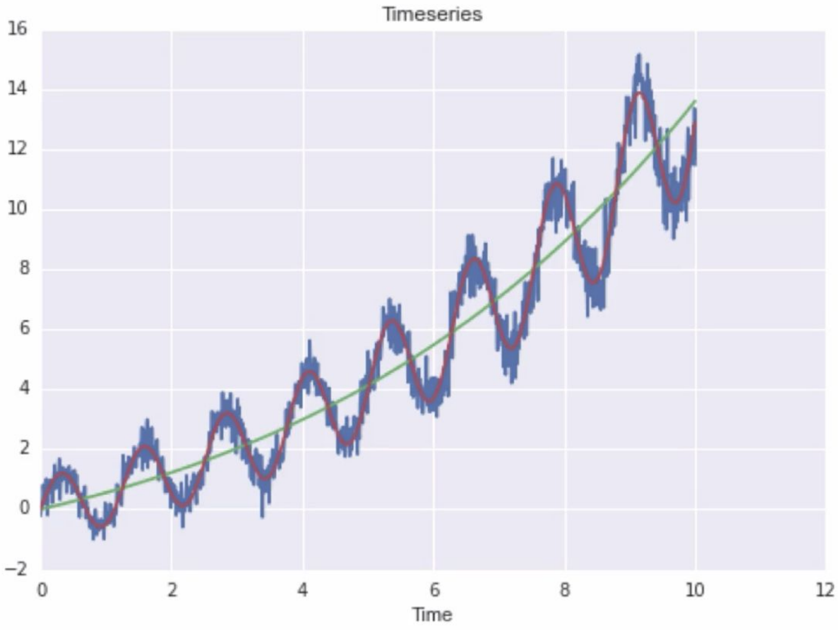
New

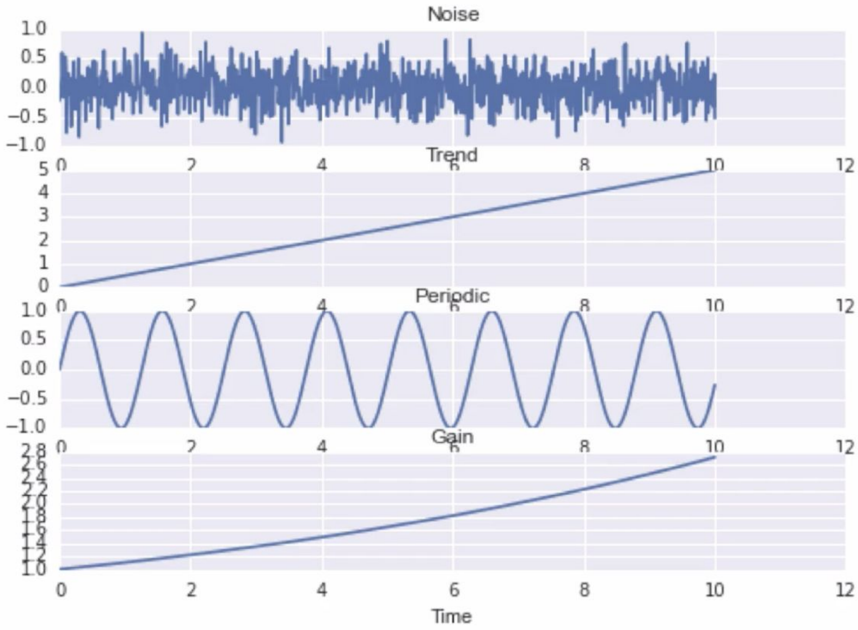
- Recurrent NN
- Vanishing and Exploding gradient
- LSTM and others
- Applications - any ordered series of data points
  - Translation - Not only english french (images to captions, song signal to lyrics, etc)
  - Language Modeling
  - Caption Generating
  - Program execution
  - Forecasting
  - Music
  - Games

Time Series

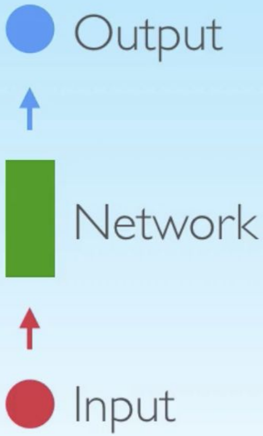
-

TREND AND SEASONALITY



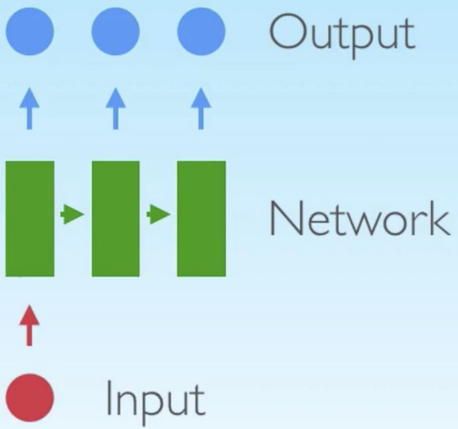


ONE TO ONE



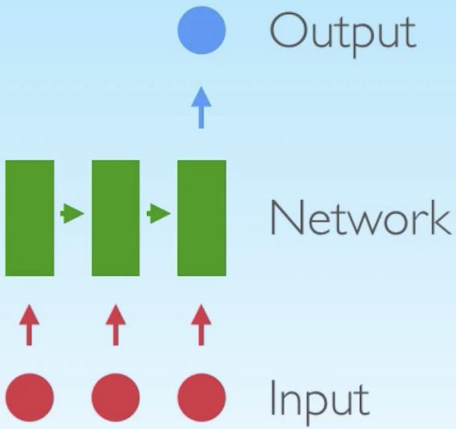
- Point-wise Forecasting
- Classification (fixed input/output size)

ONE TO MANY



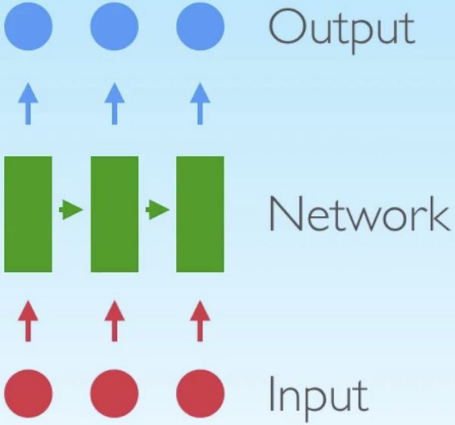
- Sequence output from single input
- e.g. image captioning

MANY TO ONE



- Sequence input, single output
- e.g. sentiment analysis from text

MANY TO MANY

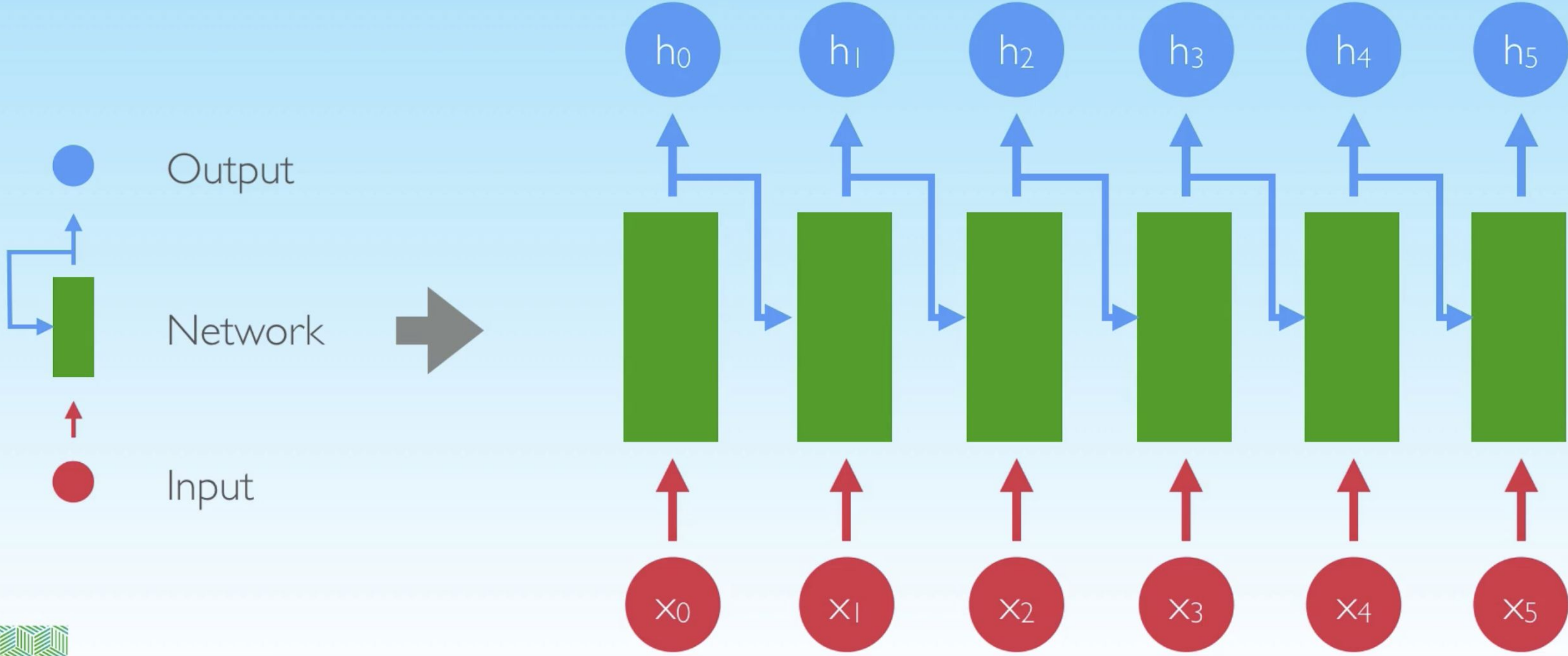


- Synced input output
- e.g. video frame classification

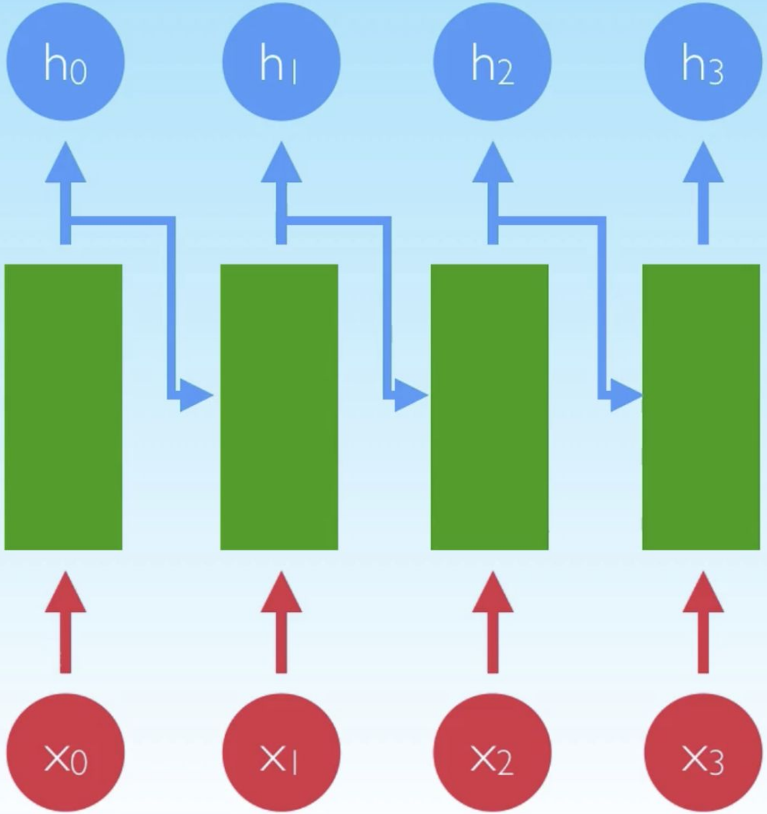
RECURRENT NEURAL NETWORKS

- connections between units form a **directed cycle**
- networks with **internal state**

UNROLL TIME



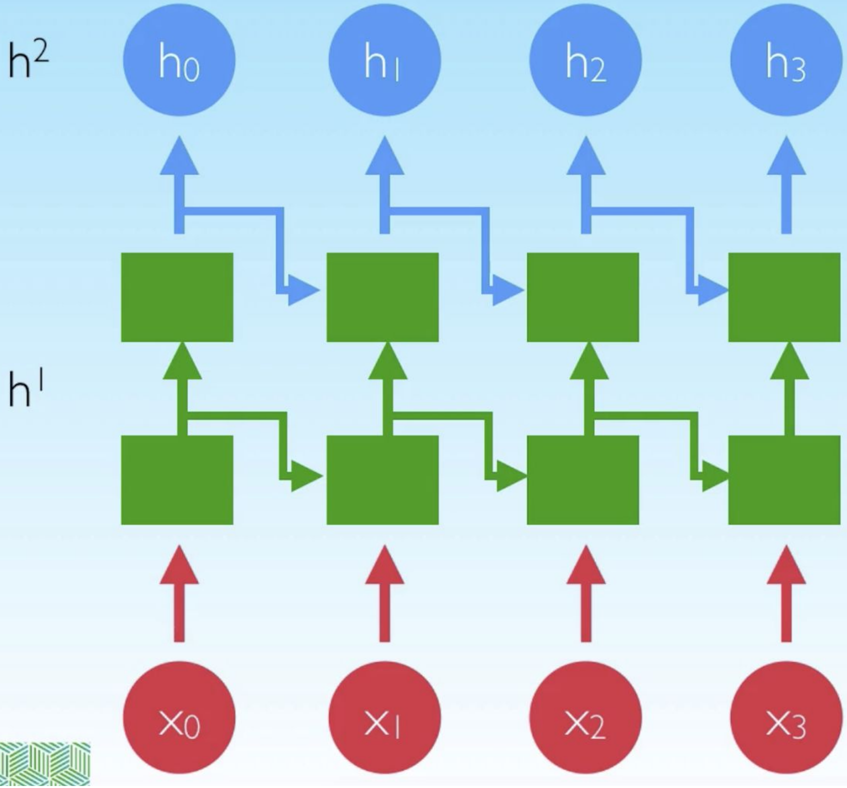
UNROLL TIME



$$h_t = \tanh(w\, h_{t-1} + u\, x_t)$$

- w, u do not depend on t
- same weights at all times

DEEP RNN



$$h_t^2 = \tanh(w^2 h_{t-1}^2 + u^2 h_t^1)$$

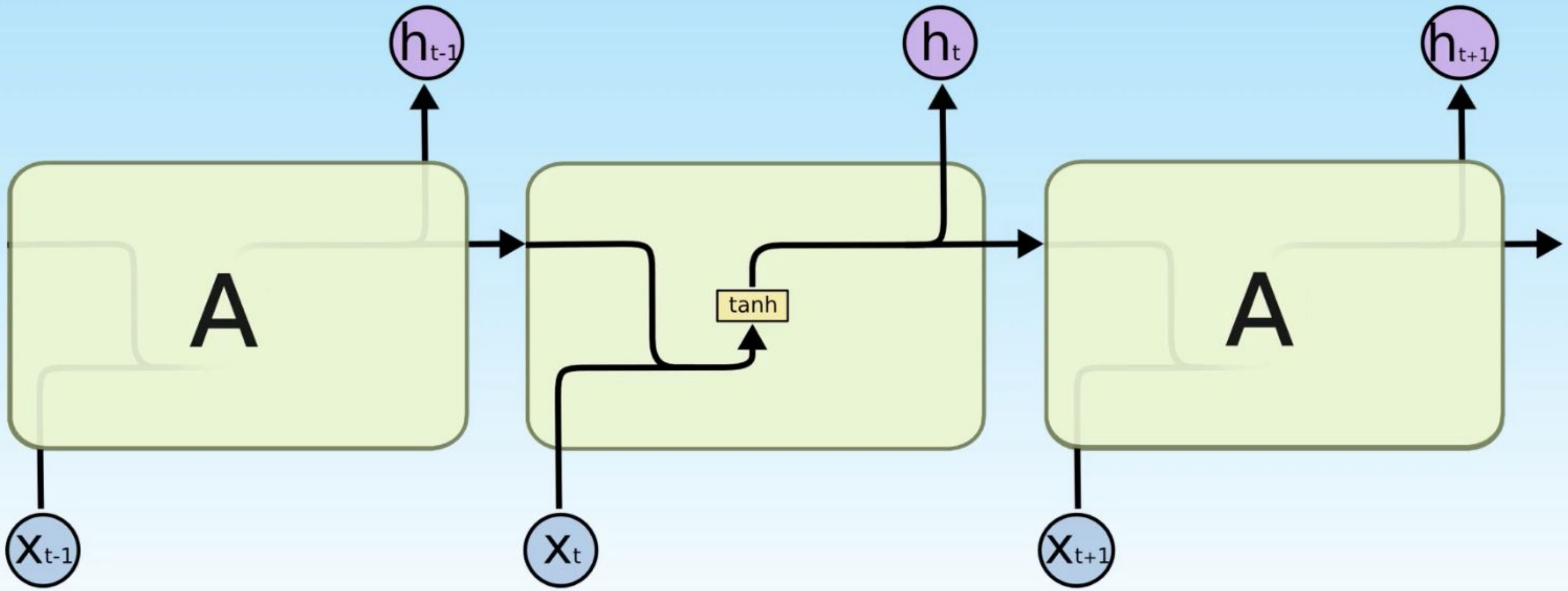
Second Layer

$$h_t^1 = \tanh(w^1 h_{t-1}^1 + u^1 x_t)$$

First Layer

Vanishing / Exploding Gradient

VANILLA RNN



Learning Long-Term Dependencies with Gradient Descent is Difficult

Yoshua Bengio, Patrice Simard, and Paolo Frasconi, *Student Member, IEEE*

***Abstract*— Recurrent neural networks can be used to map input sequences to output sequences, such as for recognition, production or prediction problems. However, practical difficulties have been reported in training recurrent neural networks to perform tasks in which the temporal contingencies present in the input/output sequences span long intervals. We show why gradient based learning algorithms face an increasingly difficult problem as the duration of the dependencies to be captured increases. These results expose a trade-off between efficient learning by gradient descent and latching on information for long periods. Based on an understanding of this problem, alternatives to standard gradient descent are considered.**

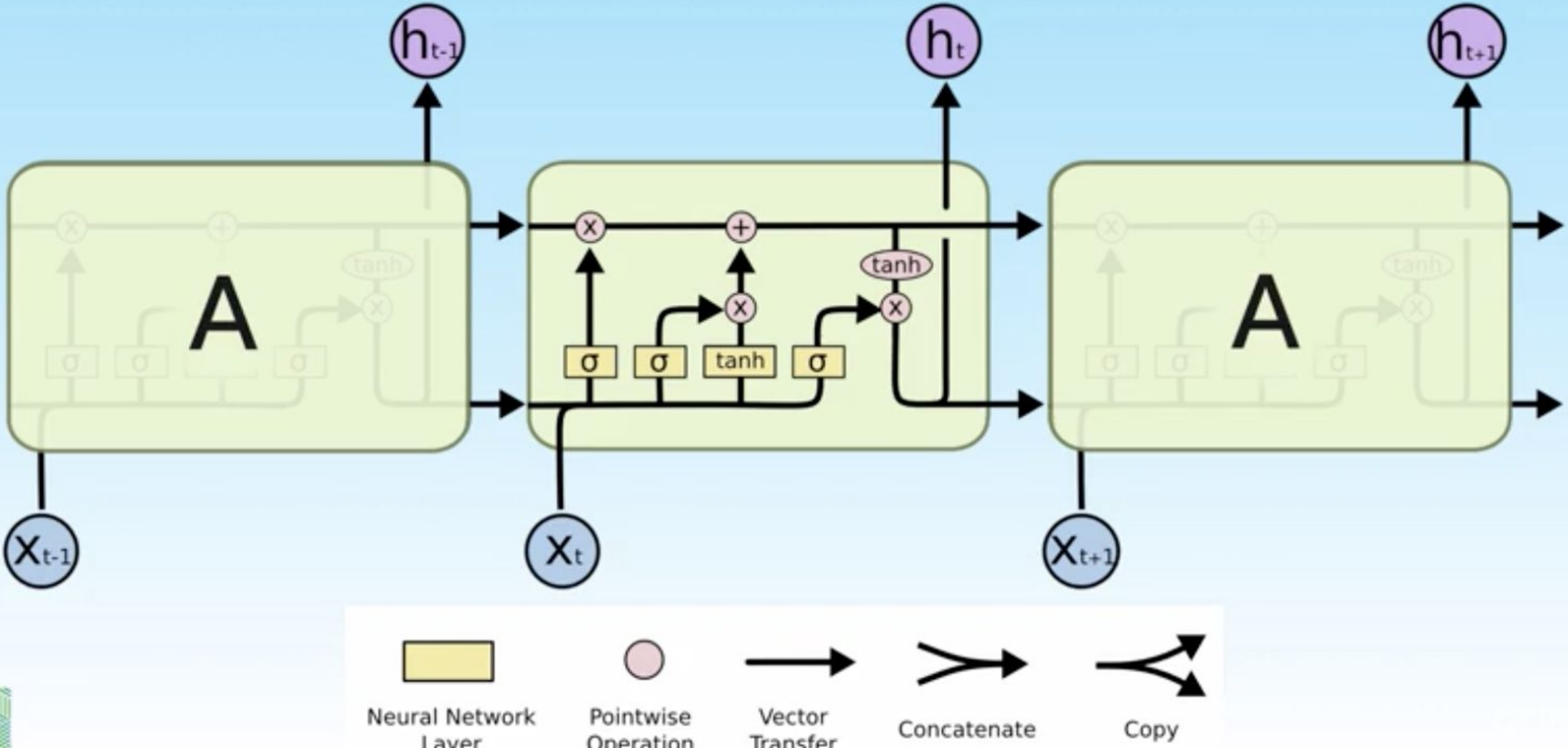
I. Introduction

WE ARE INTERESTED IN training recurrent neural

a fully connected recurrent network) but are local in time; i.e., they can be applied in an on-line fashion, producing a partial gradient after each time step. Another algorithm was proposed [10], [18] for training constrained recurrent networks in which dynamic neurons—with a single feedback to themselves—have only incoming connections from the input layer. It is local in time like the forward propagation algorithms and it requires computation only proportional to the number of weights, like the back-propagation through time algorithm. Unfortunately, the networks it can deal with have limited storage capabilities for dealing with general sequences [7], thus limiting their representational power.

A task displays long-term dependencies if prediction of

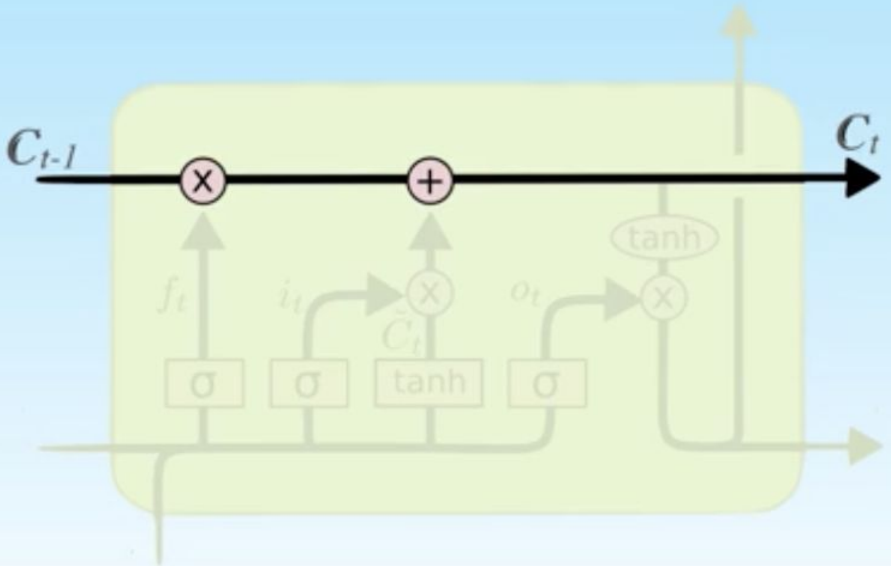
LSTM



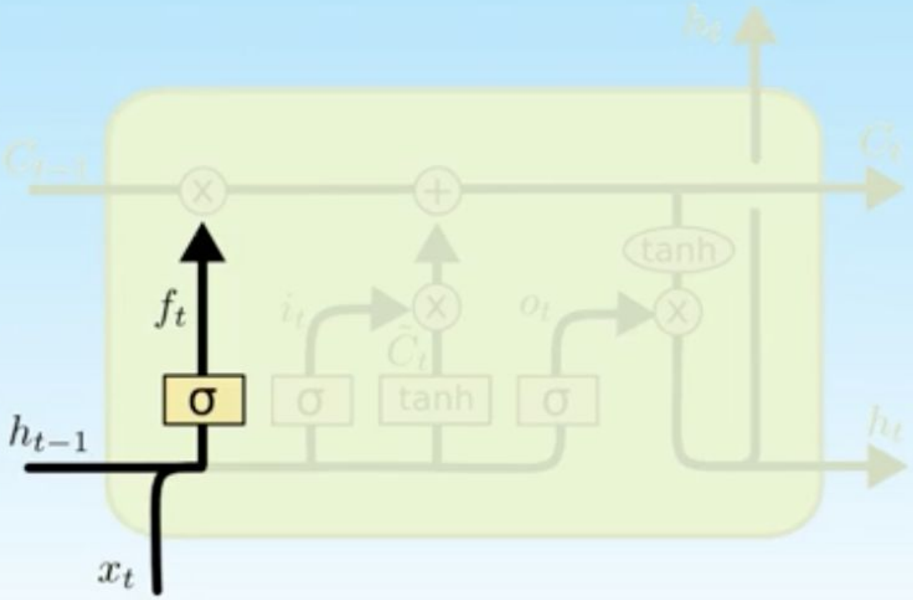
h t-1
h t
h t+1
A
A
σ
σ
tanh
σ
tanh
X t-1
X t
X t+1
Neural Network
Pointwise
Vector
Concatenate
Copy

CELL STATE

- Cell maintains state
- Gates modify information



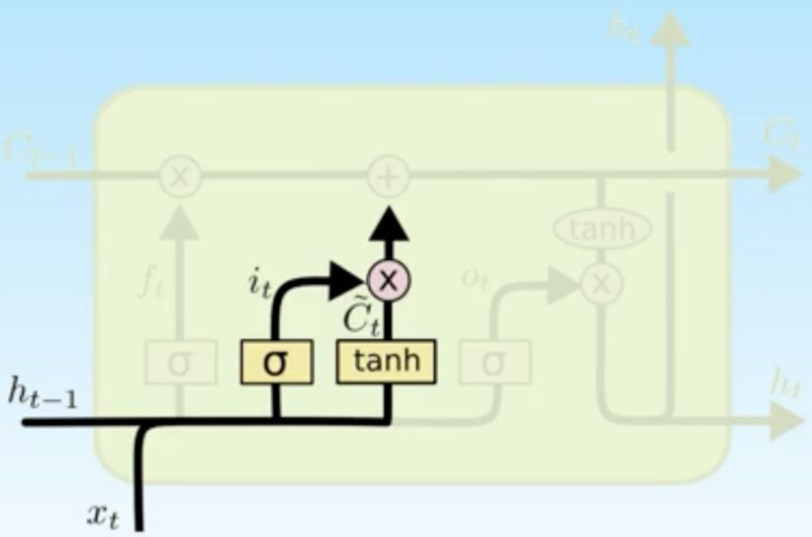
FORGET GATE



Slowly removes information that does not seem to be important from the previous layer.
Combines previous layer output and current input, concatenates them and applies a linear transformation, followed by sigmoid. Output is between 0 and 1. 0 means previous state is forgotten, 1 means previous state is remembered completely

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

INPUT GATE



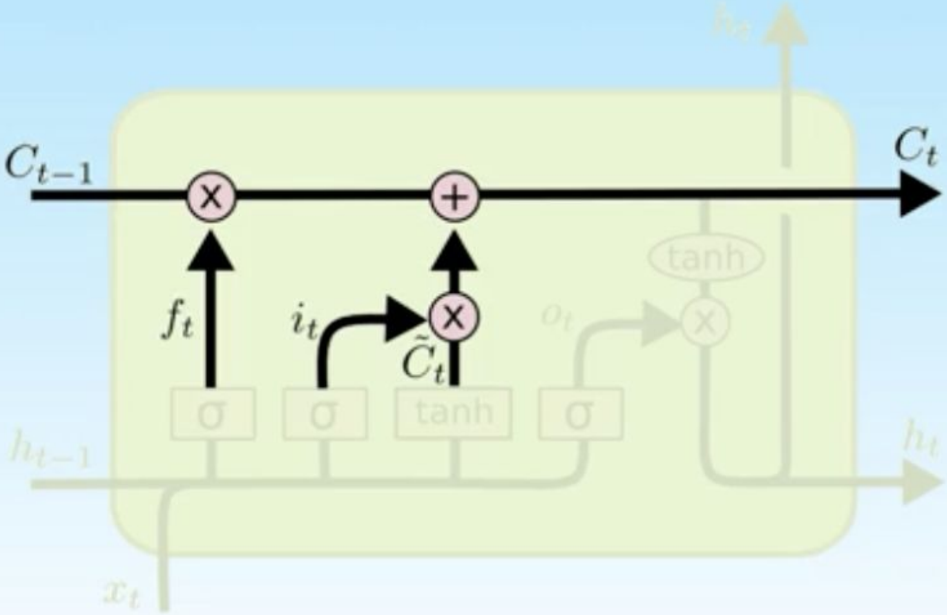
Takes previous output and current input and passes through a sigmoid. Similar to forget gate. Also between 0 and 1

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

Candidate layer. Decides which new features are relevant enough to add to the internal state of the cell

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

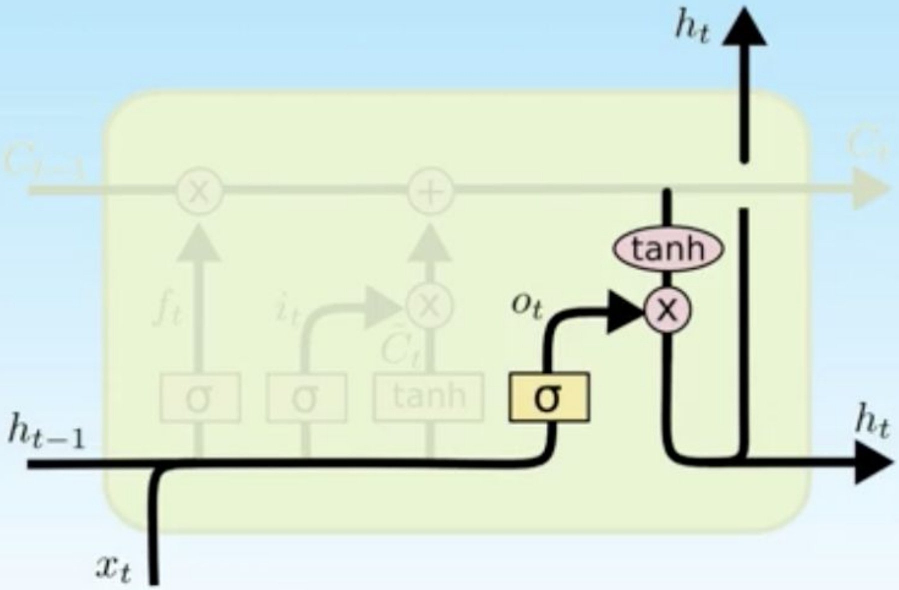
STATE UPDATE



Internal state, previous state multiplied by forget gate output and added to the fraction of the new candidate allowed by input gate

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

OUTPUT GATE



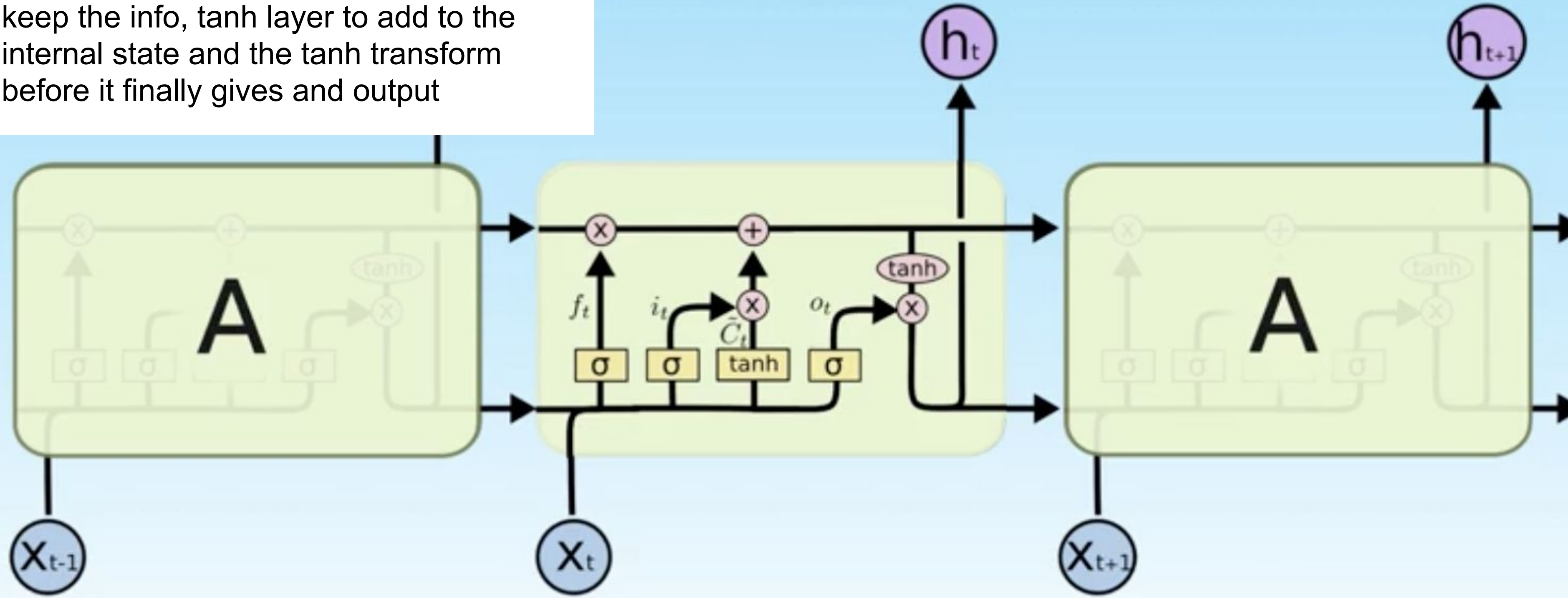
Output gate is the actual output at time t. It combines the internal state with the input and previous state.
Results in the output (this output will also be used in the next state)

$$o_t = \sigma\left(W_o\,[h_{t-1}, x_t] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

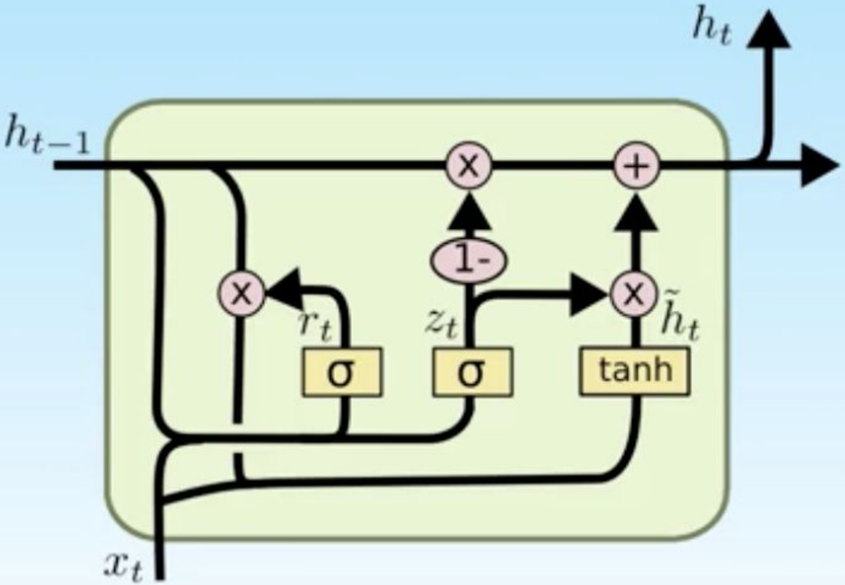
3 gates. All work in same way
But they have independent weights and biases
3 other components. Internal state to keep the info, tanh layer to add to the internal state and the tanh transform before it finally gives and output

LSTM



Only passes output to the next iterations.
Only has 2 gates instead of 3
Gate 1 = previous output and current input
Gate 2 = previous output and current output
Total output is the exponentially weighted moving average or EWMA

GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

rnn.ipynb

Exercise 1

Correct exercise 1

Exercise 2

Correct exercise 2

Show and Tell (Image Captioning)

https://arxiv.org/pdf/1411.4555.pdf

Appendix

https://www.youtube.com/watch?v=4PCktDZJH8E