# Time Series Management

Michele Linardi Ph.D.

michele.linardi@orange.fr

# Syllabus

- Recap: Time Serie Forecasting and Deep Learning Fundamentals

- Introduction to convolutional neural networks (CNN)

- **Introduction to recurrent neural networks (RNN)**

- **Encoder-decoder RNN model**

- Implementation of the models with the Keras Library

# Time series data... quick recap

- A **univariate time series** is a sequence of measurements of the same variable collected over time. Most often, the measurements are made at regular time intervals.

# Time Series Forecasting

- Predict Future (Values) based on the past (Historical Data)

| Date | Observation |
|------|-------------|
| 2018-06-04 | 60 |
| 2018-06-05 | 64 |
| 2018-06-06 | 66 |
| 2018-06-07 | 65 |
| 2018-06-08 | 67 |
| 2018-06-09 | 68 |
| 2018-06-10 | 70 |
| 2018-06-11 | 69 |
| 2018-06-12 | 72 |
| 2018-06-13 | ? |
| 2018-06-14 | ? |
| 2018-06-15 | ? |

History — (rows 2018-06-04 through 2018-06-12)

Future — (rows 2018-06-13 through 2018-06-15)

# Question to ask prior to forecast

- Can we forecast?
  - How well we understand the factor influencing the future?
  - How much data we have?*

- How far in the future (horizon) we want to forecast?

- What technique, model should we use ?

# Why Deep Learning models? (1/2)

Deep learning model perform well and better than other methods in many scenarios.

| Model | Features | RMSE | MAPE (%) | Run time (s) |
|---|---|---|---|---|
| FeedForward | Date features + Covariants | **10.73** | 4.20 | 53.35 |
| AutoARIMA | Baseline | 11.00 | **3.52** | 20658.95 |
| DeepAR | Date features | 11.96 | 5.99 | 156.18 |
| FeedForward | Baseline | 12.01 | 3.88 | 78.47 |
| SeasonalNaive | Baseline | 12.47 | 3.86 | **1.20** |
| FeedForward | Date features | 13.93 | 3.95 | 9.37 |
| DeepAR | Baseline | 14.01 | 6.12 | 856.51 |
| DeepAR | Date features + Covariants | 17.79 | 6.04 | 781.09 |
| TrivialIdentity | Baseline | 18.20 | 5.09 | 1.30 |
| NPTS | Baseline | 82.45 | 19.75 | 25.20 |

https://blog.dataiku.com/deep-learning-time-series-forecasting

Z. Tang, P.A. Fishwik, *Feedforward Neural Nets as Models for Time Series Forecasting*, November 1993

# Why Deep Learning models? (2/2)

- Non-parametric
- Flexible and expressive
- Easily inject exogenous features into the model
- Learn from large time series datasets

# Recurrent Neural Network (RNN)
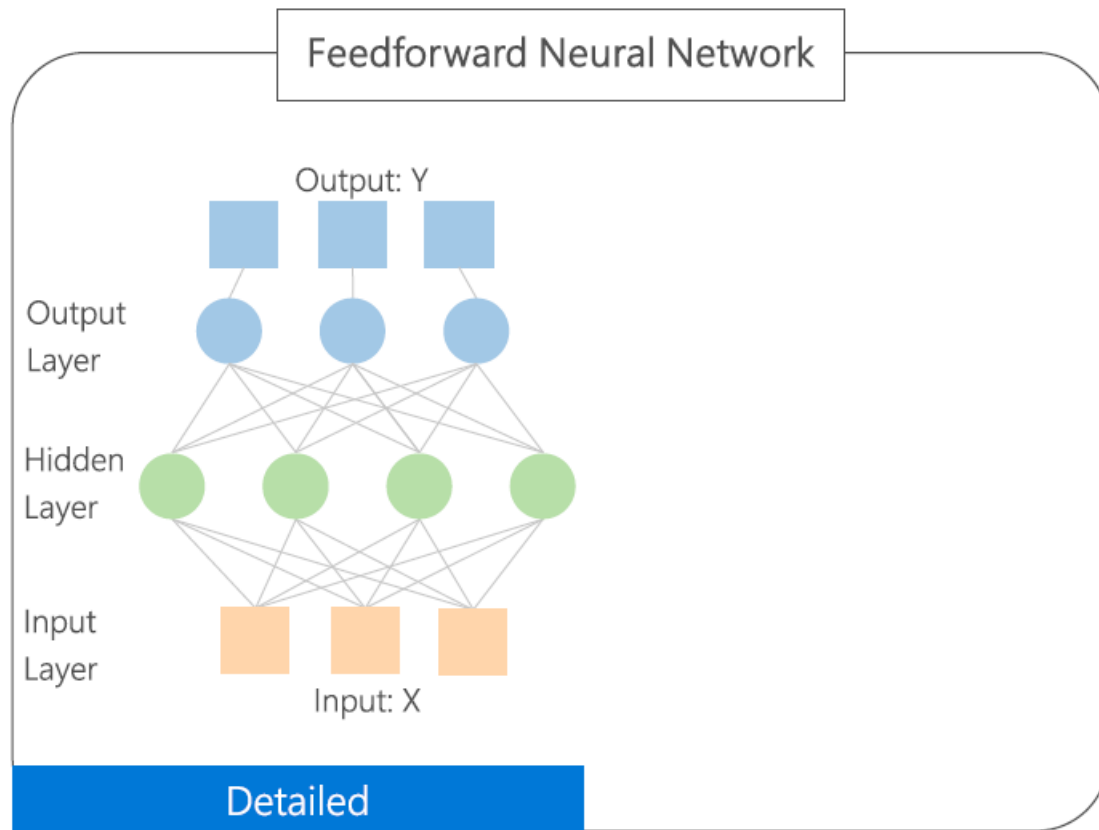# for
# Time series
# Forecasting
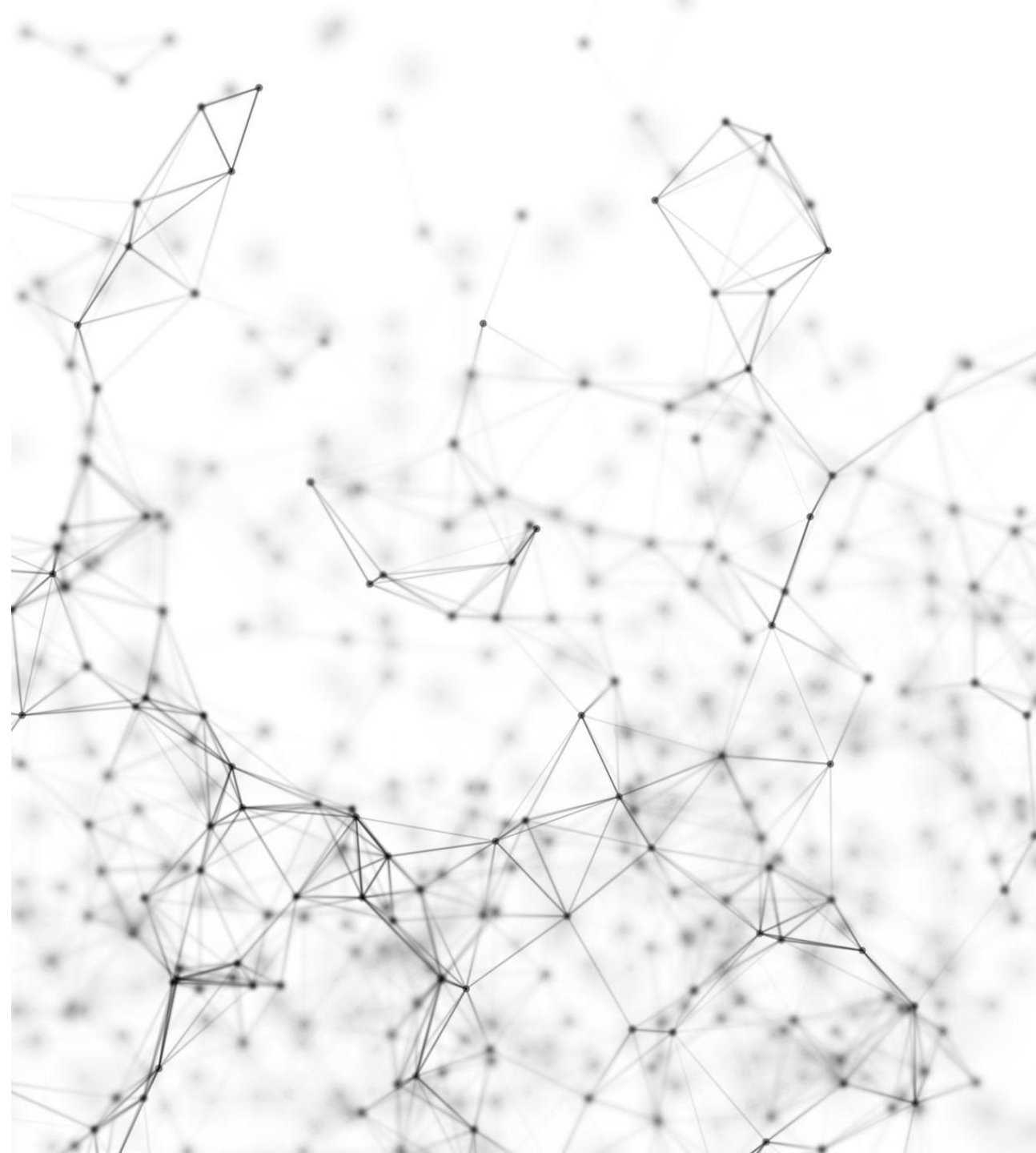
# Roadmap

- What are RNNs?

- How RNNs are trained: Backpropagation through time (BPTT).
- Vanilla RNN and its gradient problems.
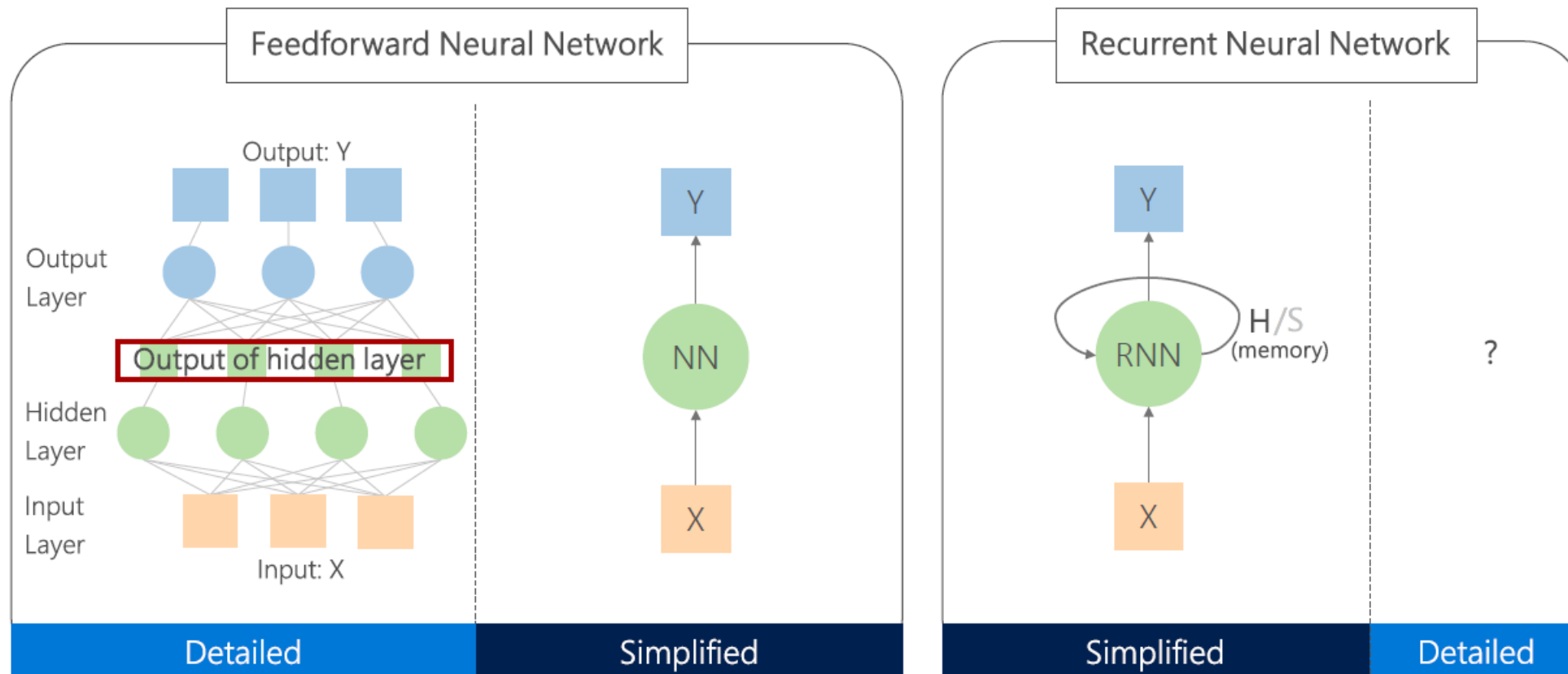- Other RNN units.

RNN architectures:
- GRU
- LSTM
- RNN stacking
- RNN for one step time series forecasting
- Encode-decoder RNN for multi-step time series forecasting

# Recurrent Neural Networks (RNN)

# Recurrent Neural Networks (RNN)

RNN has an internal state (H), which can be fed back to the network

# Recurrent Neural Networks (RNN)

- The same weight and bias shared across all the network steps



RECURRENT LAYER

*In Keras you will see "timesteps" when set data input shape*
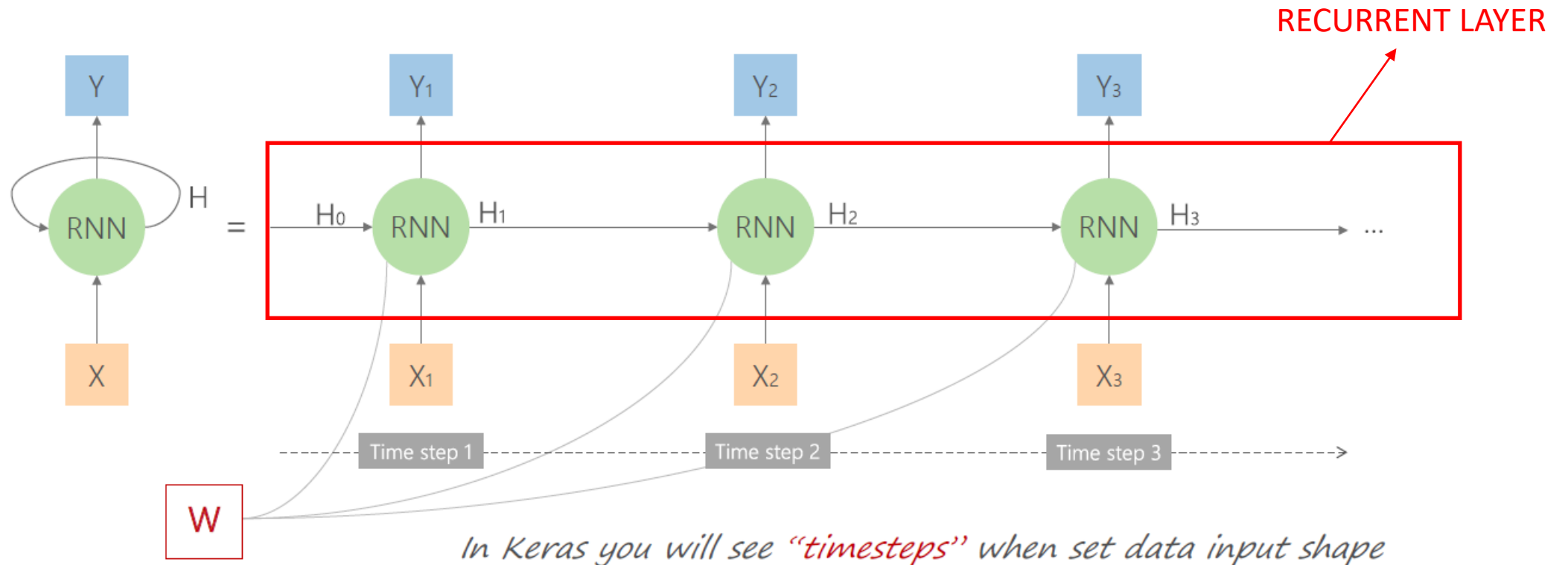
# RNN – First Complete Picture

Internal state ( a.k.a. Hidden state at time t $H_t \in \mathbb{R}^u$) $u := units$
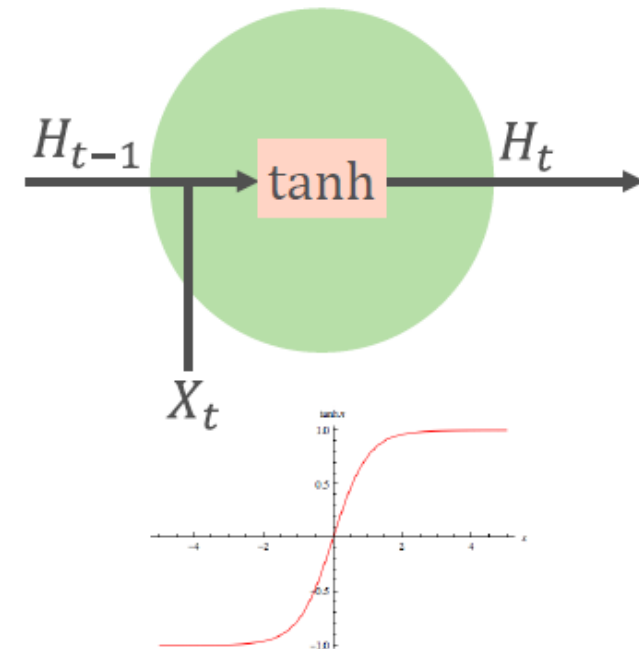
# Vanilla RNN Unit



Vanilla RNN:
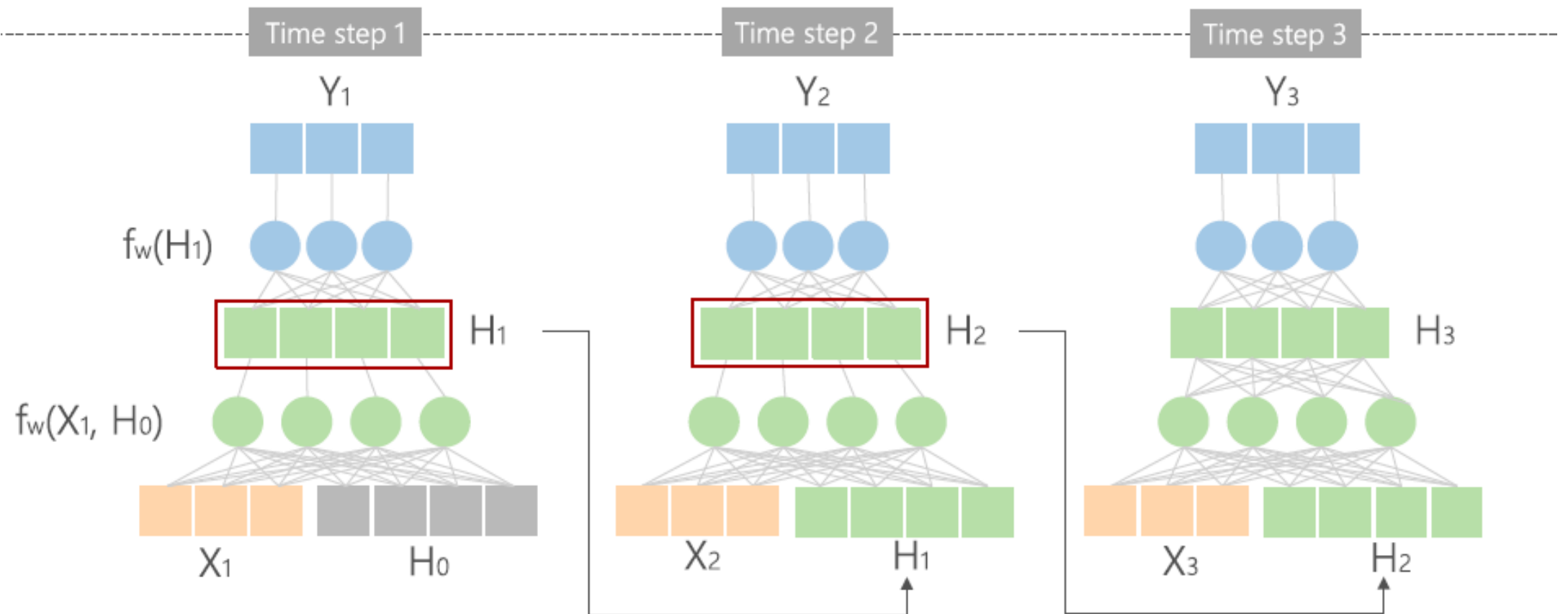
$$H_t = \tanh(W_h H_{t-1} + W_x X_t)$$
$$= \tanh(\mathbf{W} \cdot [H_{t-1}, X_t])$$
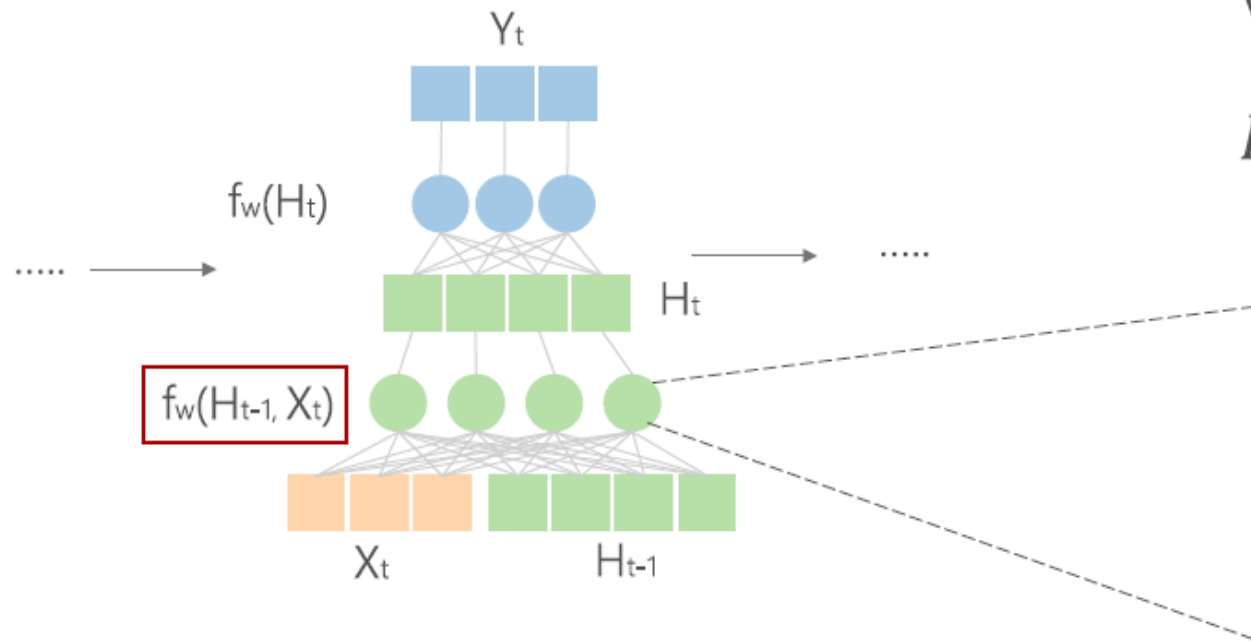
$$H_t = f_W(H_{t-1}, X_t)$$

some function with parameter W

new state — old state — Input vector at some time step

# Back propagation (through time) of Hidden states



$$\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^{T} \left. \frac{\partial \mathcal{L}^{(T)}}{\partial W} \right|_{(t)}$$

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$
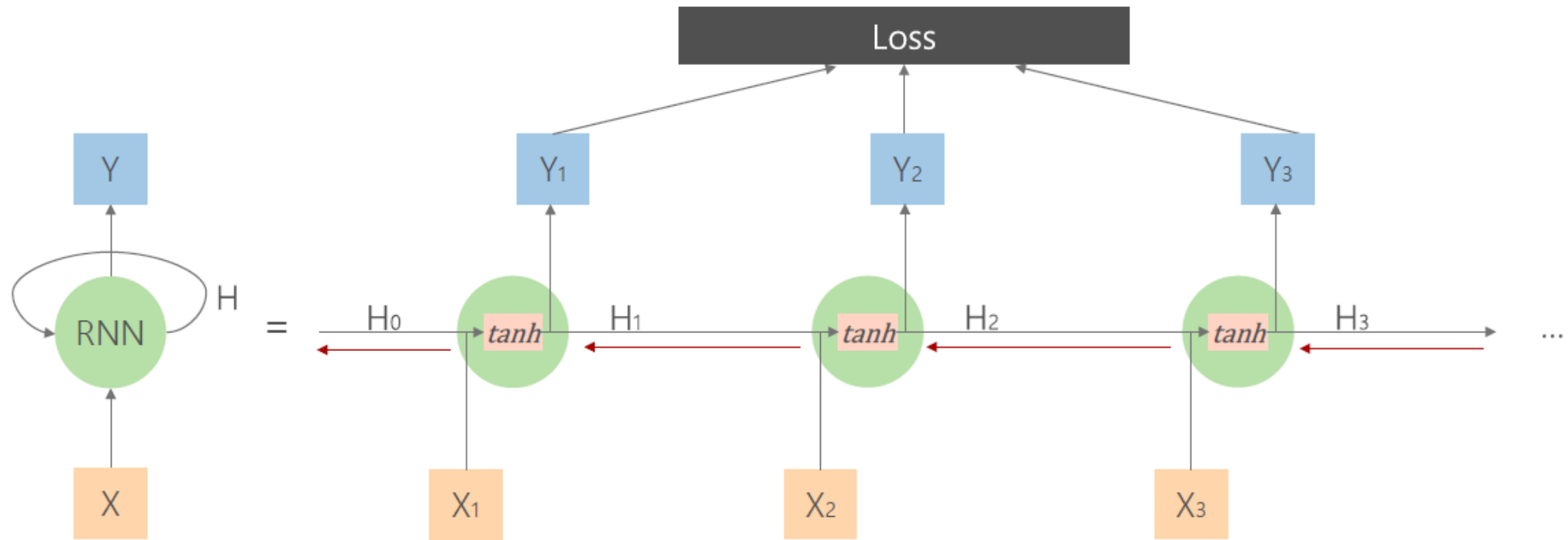
# RNN - Backpropagation



Forward through entire sequence to compute loss

then backward through entire sequence to compute gradient
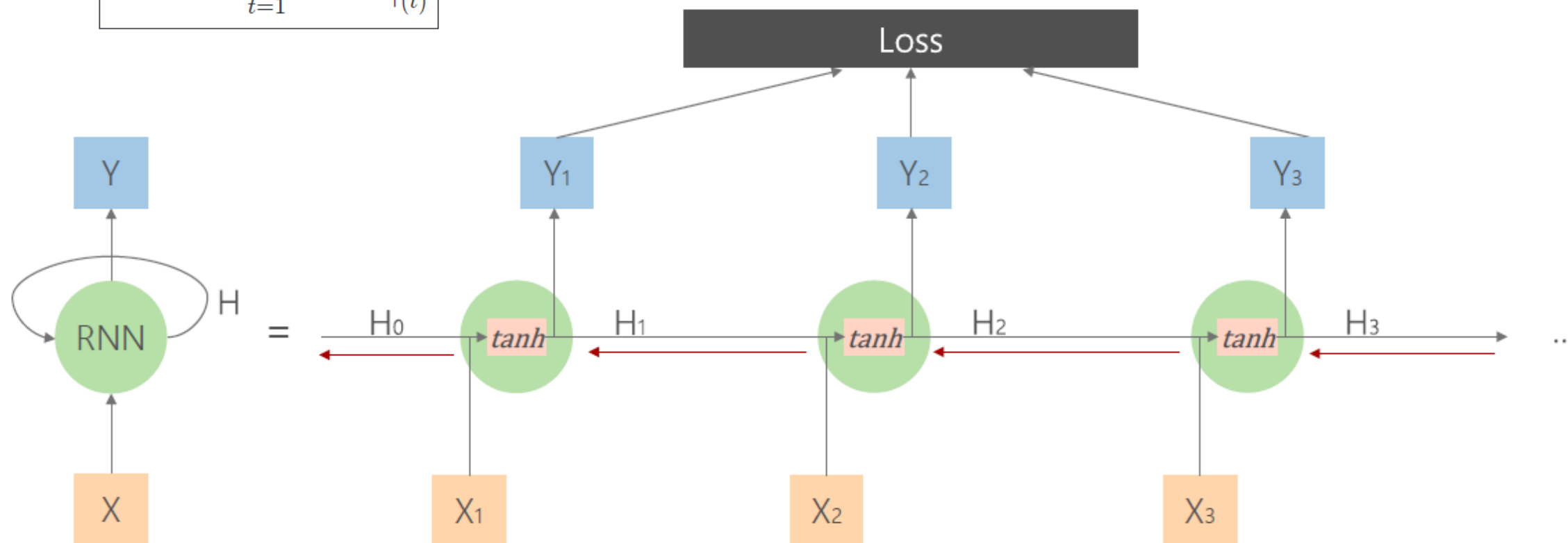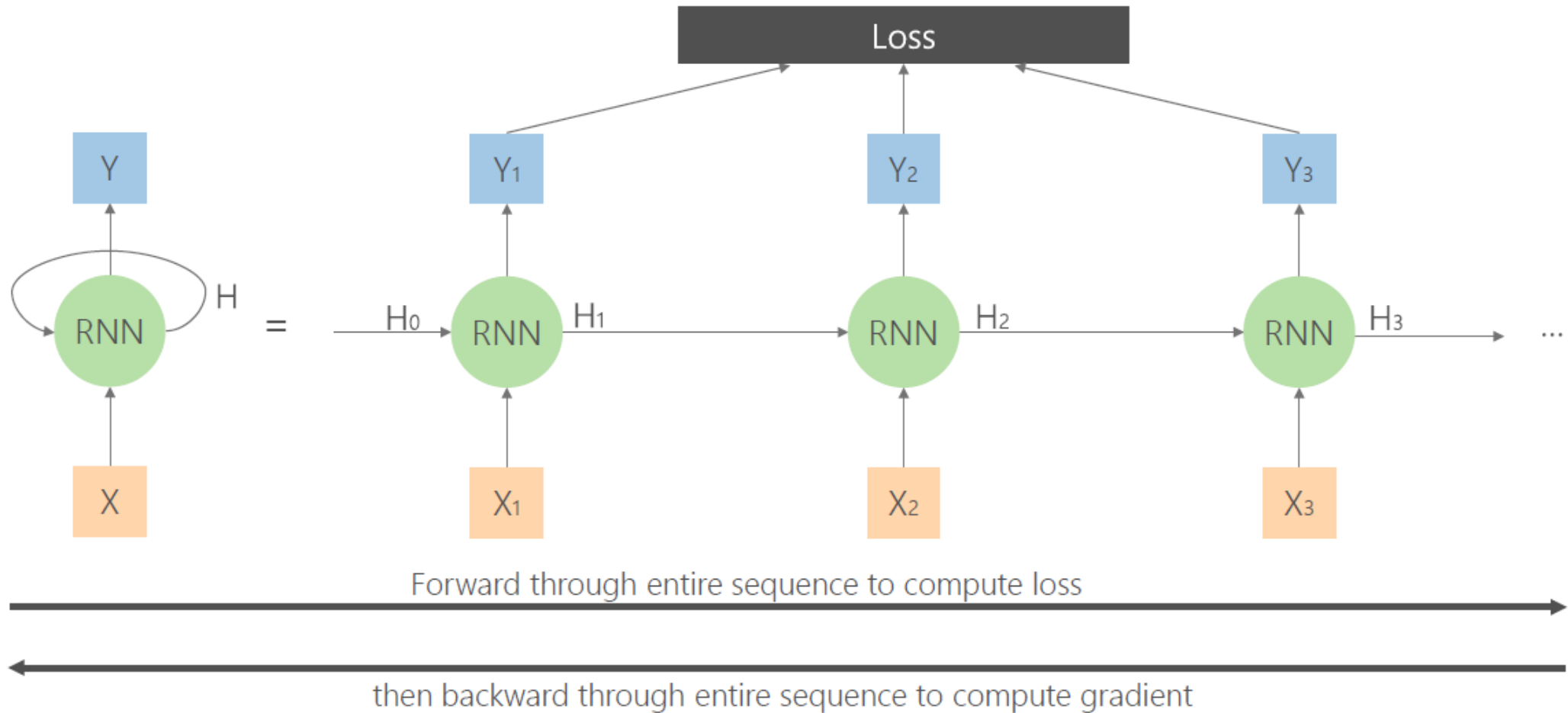
# Back propagation of Hidden states

**Vanilla RNN Gradient Problems**

Computing gradient of $H_0$ involves repeated tanh and many factors of W which causes:

Exploding gradient (e.g. 5*5*5*5*5*5*......)

Vanishing gradients (e.g. 0.7*0.7*0.7*0.7*0.7*0.7*......)

# Explosion of the gradient (e.g. 5*5*5*5*5*5*.....)

Exploding gradients are obvious:

Your gradients will become a large number (e.g., NaN (not a number))

Solution: **Gradient clipping**

Clip the gradient when it goes higher than a threshold

# Gradient Vanishing
# (e.g. 0.7*0.7*0.7*0.7*0.7*0.7*……)

Vanishing gradients are more problematic because it's not obvious when they occur or how to deal with them.

https://keras.io/api/layers/regularizers/

Solutions:

- Change activation function to ReLU
- Proper initialization
- Regularization := [ Cost function = Loss + Regularization terms]
- **Change architecture to LSTM or GRU**

```
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01)
```

# Gated Recurrent Unit (GRU)

Vanilla RNN:

$$H_t = \tanh(\mathbf{W} \cdot [H_{t-1}, X_t])$$
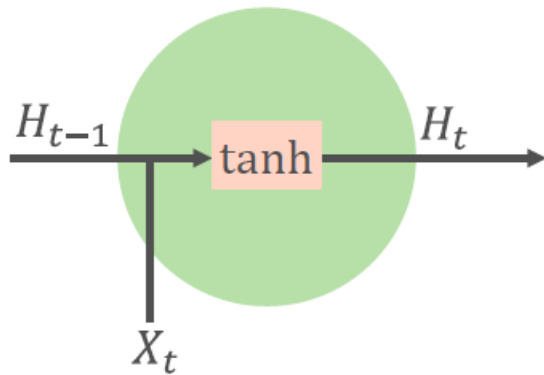


GRU:

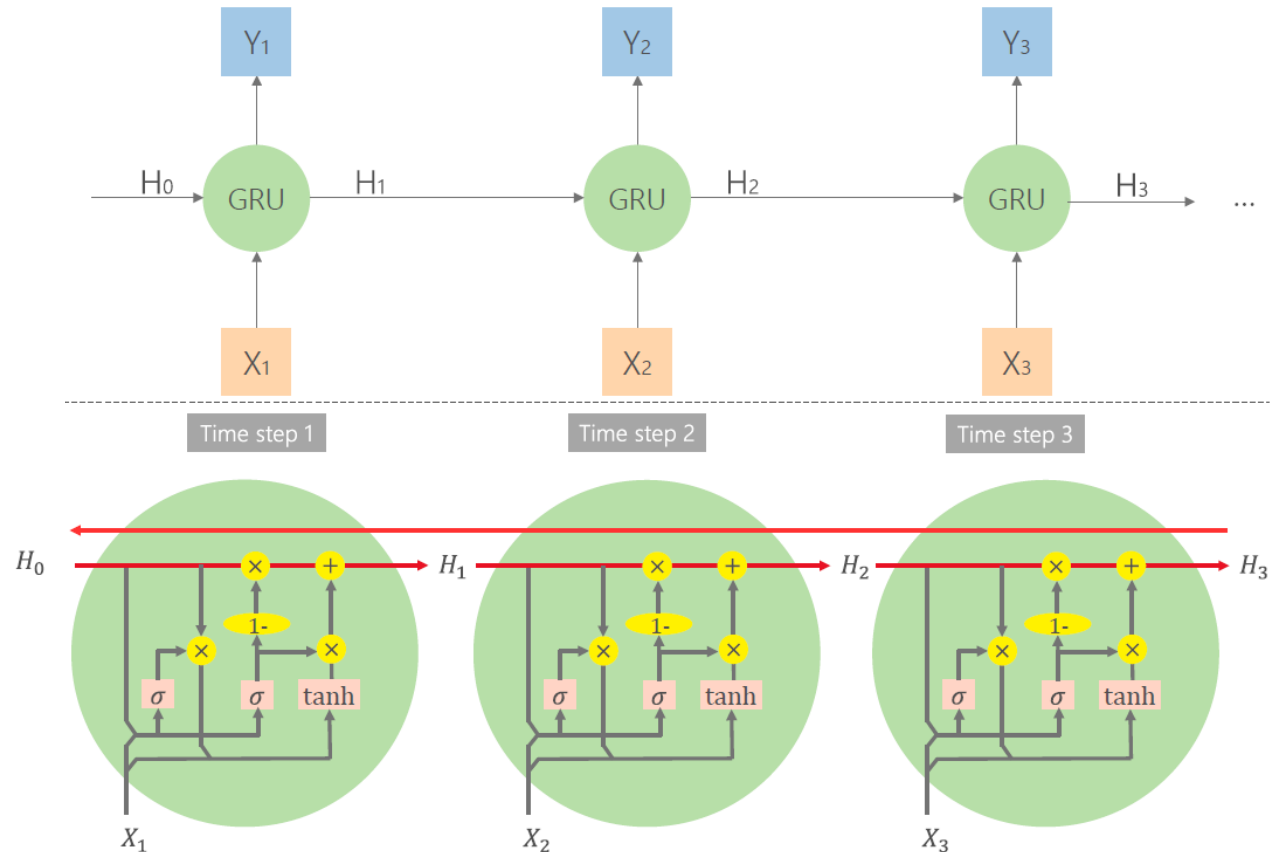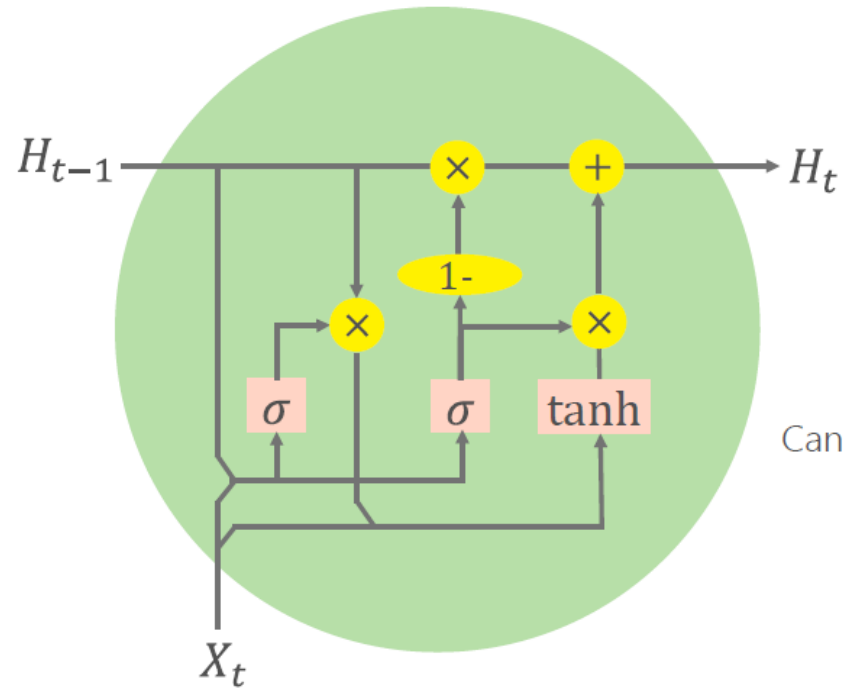$$H_t = (1 - z_t) \times H_{t-1} + z_t \times \widetilde{H}_t$$
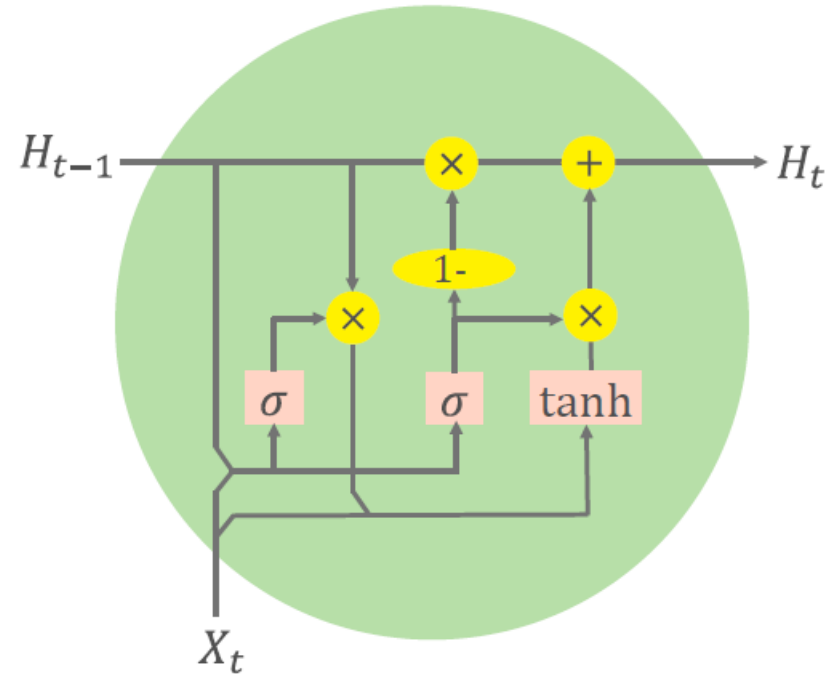
# Uninterrupted gradient flow

State runs straight through the entire chain with minor linear interactions which makes information very easy to pass.

GRU:

$$H_t = (1 - z_t) \times H_{t-1} + z_t \times \tilde{H}_t$$
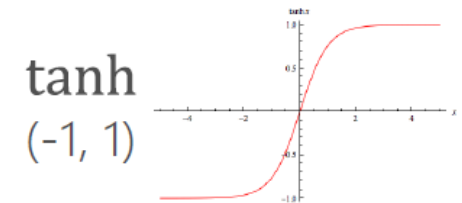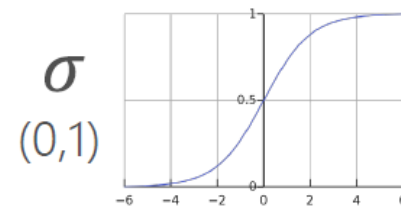
# Gated Recurrent Unit (GRU)



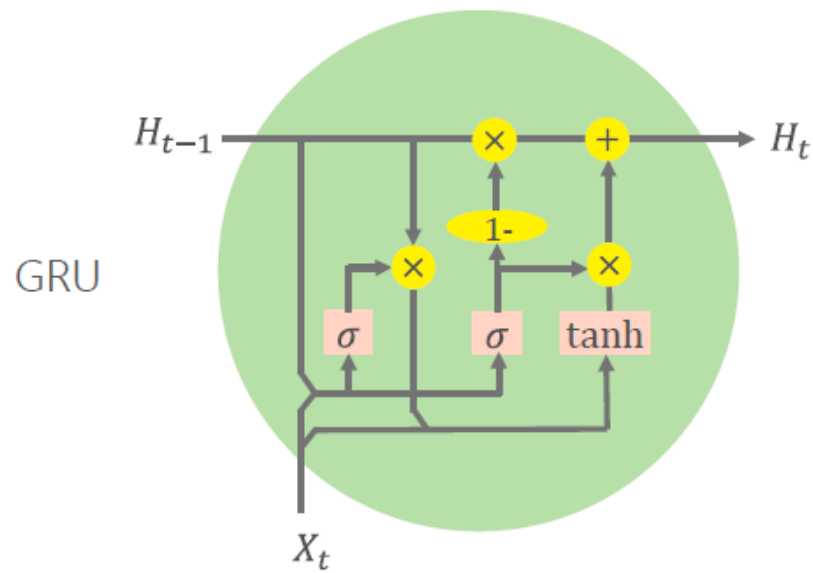Hidden state: $H_t = \boxed{(1 - z_t)} \times H_{t-1} + \boxed{z_t} \times \boxed{\widetilde{H}_t}$

Update gates: $z_t = \sigma(\quad W_z \cdot [H_{t-1}, X_t] \quad)$

Candidate gates/states: $\widetilde{H}_t = \tanh(\ W \cdot [\boxed{r_t} \times H_{t-1}, X_t])$

Reset gates: $r_t = \sigma(\quad W_r \cdot [H_{t-1}, X_t] \quad)$

$\sigma$
(0,1)

tanh
(-1, 1)

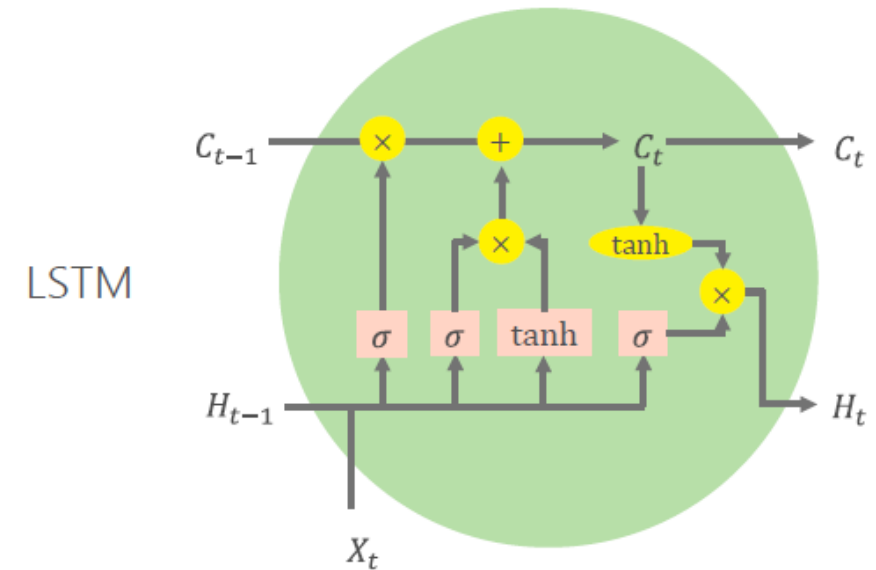# GRU vs LSTM (Long short-term memory) cell



GRU

LSTM

Hidden state: $H_t = (1 - z_t) \times H_{t-1} + z_t \times \widetilde{H}_t$

Update gates: $z_t = \sigma(\ W_z \cdot [H_{t-1}, X_t]\ )$

Reset gates: $r_t = \sigma(\ W_r \cdot [H_{t-1}, X_t]\ )$

Candidate gates/states: $\widetilde{H}_t = \tanh(\ W \cdot [r_t \times H_{t-1}, X_t])$

Hidden cell state: $C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$
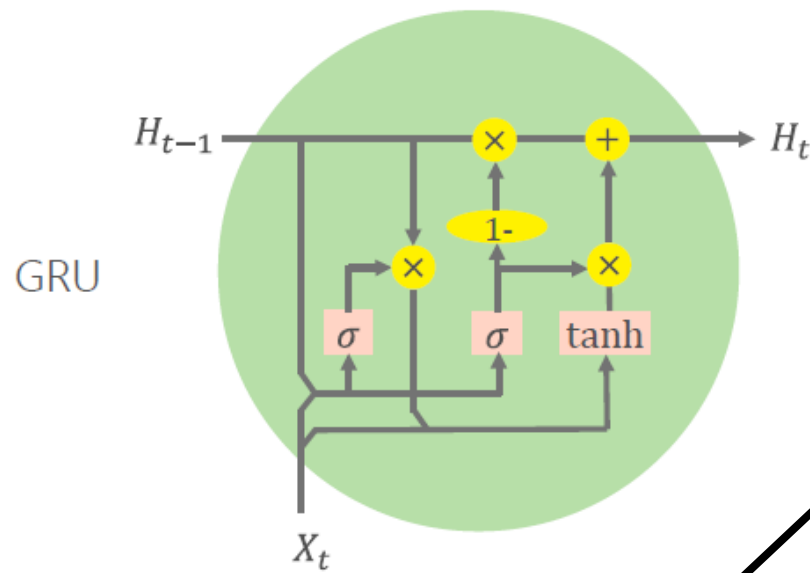
Hidden state: $H_t = o_t \times \tanh(C_t)$

Forget gates: $f_t = \sigma(\ W_f \cdot [H_{t-1}, X_t]\ )$

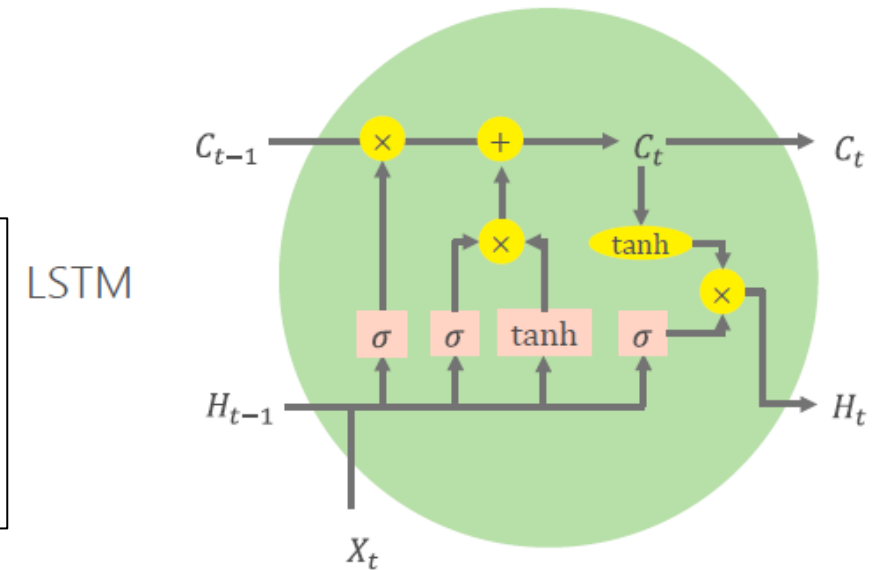Input gates: $i_t = \sigma(\ W_i \cdot [H_{t-1}, X_t]\ )$

Candidate gates/states: $\tilde{C}_t = \tanh(\ W_g \cdot [H_{t-1}, X_t])$

Output gates: $o_t = \sigma(\ W_o \cdot [H_{t-1}, X_t]\ )$

# GRU vs LSTM (Long short-term memory) cell



How much of the candidate activation vector to incorporate into the new hidden state

GRU

Hidden state: $H_t = (1 - z_t) \times H_{t-1} + z_t \times \widetilde{H}_t$

Update gates: $z_t = \sigma( W_z \cdot [H_{t-1}, X_t] )$

Reset gates: $r_t = \sigma( W_r \cdot [H_{t-1}, X_t] )$

Candidate gates/states: $\widetilde{H}_t = \tanh( W \cdot [r_t \times H_{t-1}, X_t])$

LSTM

Hidden cell state: $C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$

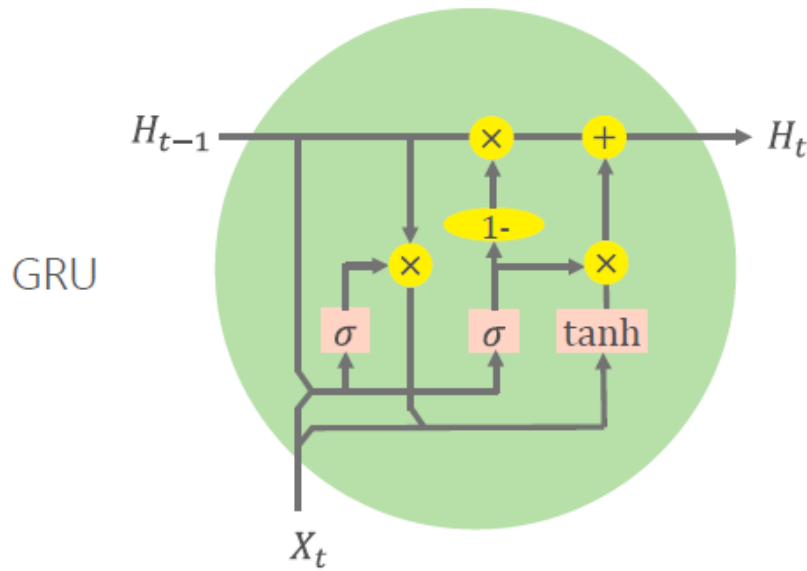Hidden state: $H_t = o_t \times \tanh(C_t)$

Forget gates: $f_t = \sigma( W_f \cdot [H_{t-1}, X_t] )$

Input gates: $i_t = \sigma( W_i \cdot [H_{t-1}, X_t] )$

Candidate gates/states: $\tilde{C}_t = \tanh( W_g \cdot [H_{t-1}, X_t])$

Output gates: $o_t = \sigma( W_o \cdot [H_{t-1}, X_t] )$

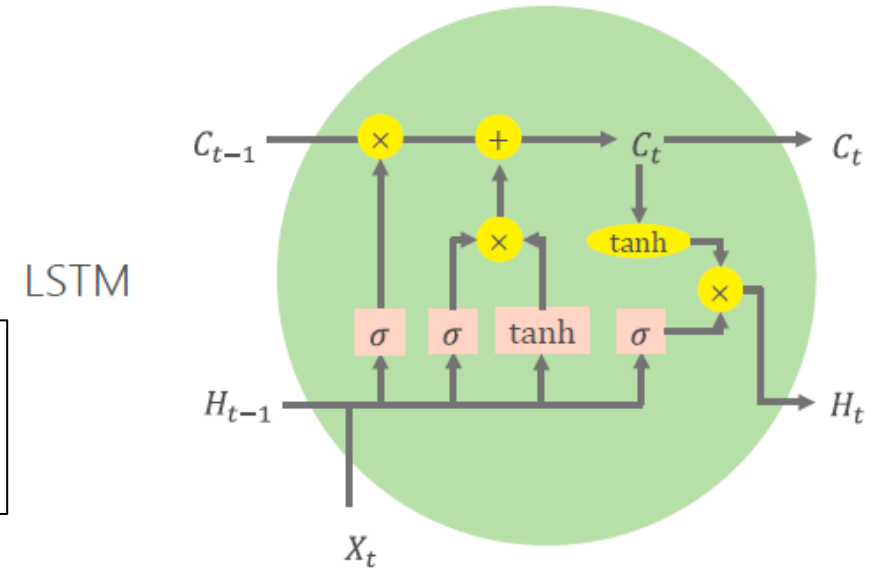# GRU vs LSTM (Long short-term memory) cell



It determines how much of the previous hidden state to forget

Hidden state: $H_t = (1 - z_t) \times H_{t-1} + z_t \times \widetilde{H}_t$

Update gates: $z_t = \sigma(\ W_z \cdot [H_{t-1}, X_t]\ )$

Reset gates: $r_t = \sigma(\ W_r \cdot [H_{t-1}, X_t]\ )$

Candidate gates/states: $\widetilde{H}_t = \tanh(\ W \cdot [r_t \times H_{t-1}, X_t])$

Hidden cell state: $C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$

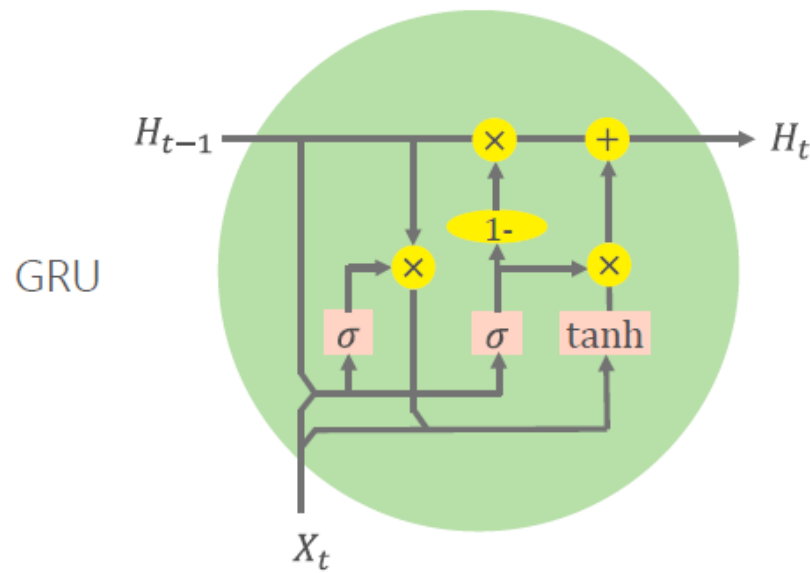Hidden state: $H_t = o_t \times \tanh(C_t)$

Forget gates: $f_t = \sigma(\ W_f \cdot [H_{t-1}, X_t]\ )$

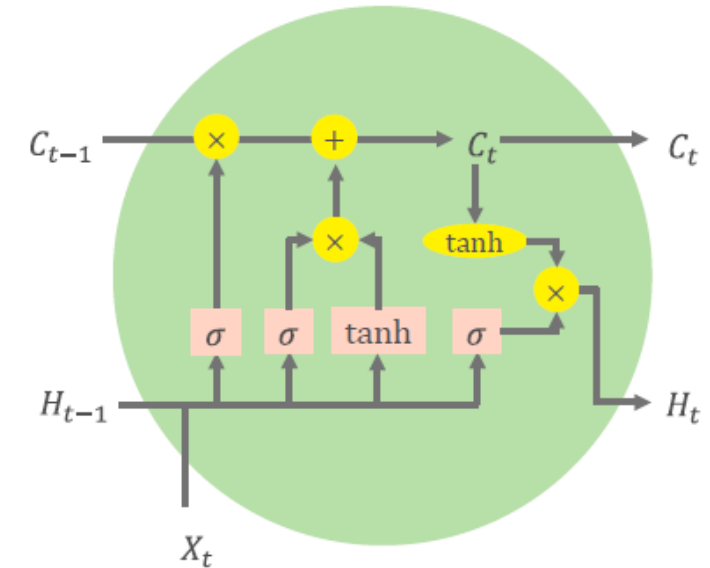Input gates: $i_t = \sigma(\ W_i \cdot [H_{t-1}, X_t]\ )$

Candidate gates/states: $\tilde{C}_t = \tanh(\ W_g \cdot [H_{t-1}, X_t])$

Output gates: $o_t = \sigma(\ W_o \cdot [H_{t-1}, X_t]\ )$

# GRU vs LSTM (Long short-term memory) cell



It combines information from the input and the previous hidden state

**GRU**

Hidden state: $H_t = (1 - z_t) \times H_{t-1} + z_t \times \widetilde{H}_t$

Update gates: $z_t = \sigma( W_z \cdot [H_{t-1}, X_t] )$

Reset gates: $r_t = \sigma( W_r \cdot [H_{t-1}, X_t] )$

Candidate gates/states: $\widetilde{H}_t = \tanh( W \cdot [r_t \times H_{t-1}, X_t])$

**LSTM**

Hidden cell state: $C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$

Hidden state: $H_t = o_t \times \tanh(C_t)$
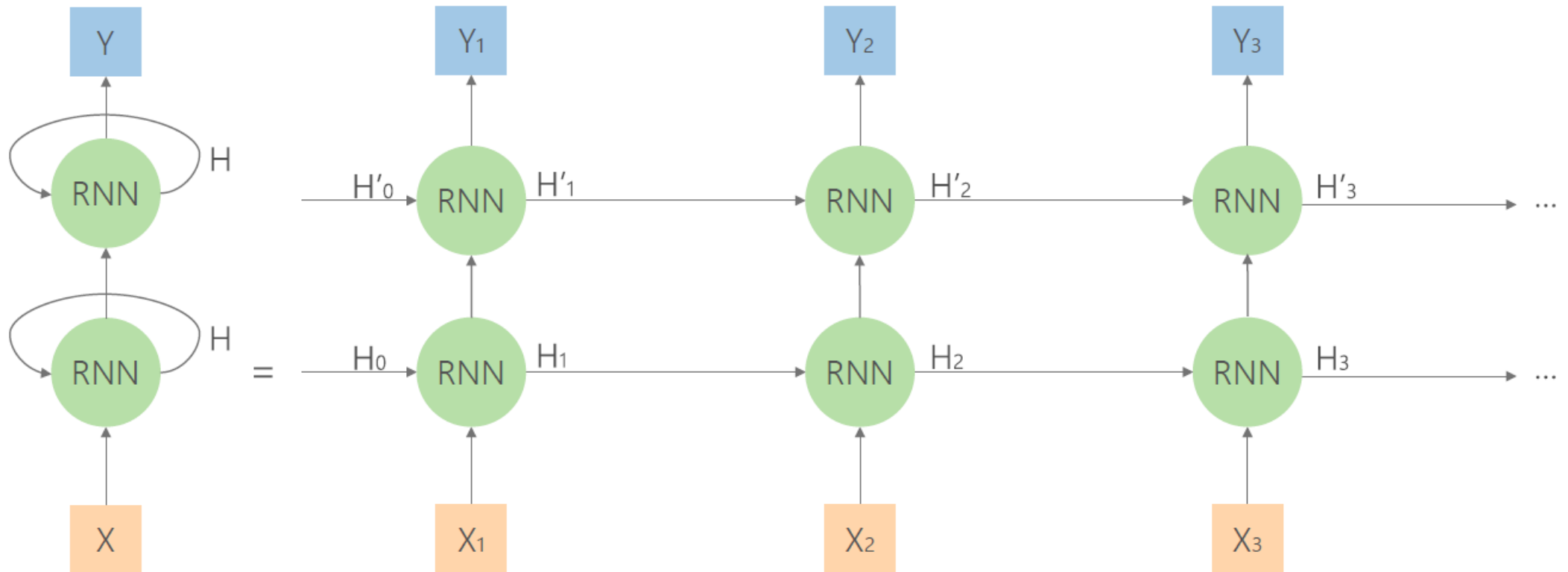
Forget gates: $f_t = \sigma( W_f \cdot [H_{t-1}, X_t] )$

Input gates: $i_t = \sigma( W_i \cdot [H_{t-1}, X_t] )$

Candidate gates/states: $\tilde{C}_t = \tanh( W_g \cdot [H_{t-1}, X_t])$

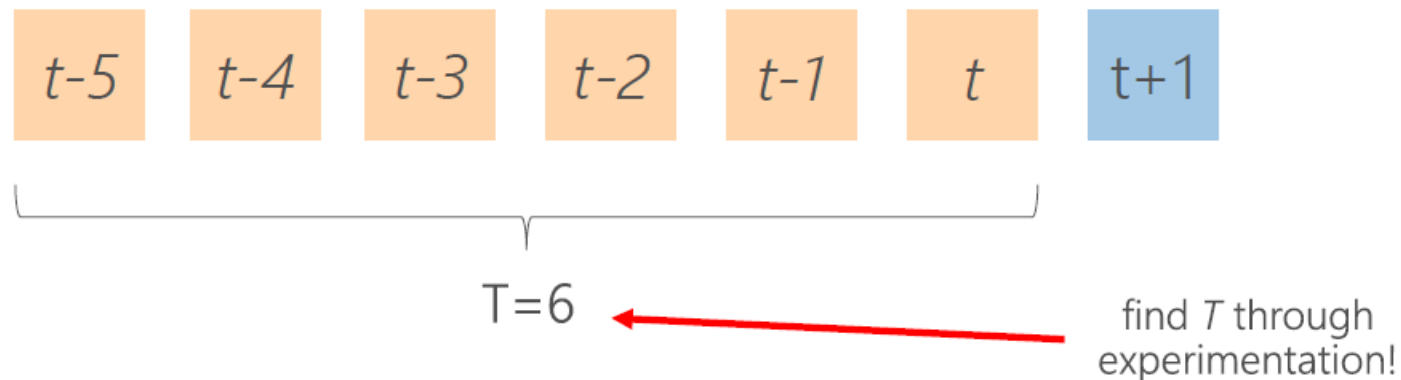Output gates: $o_t = \sigma( W_o \cdot [H_{t-1}, X_t] )$

# Cells Stacking

To learn more complex relationships, we can stack multiple cells

# One step forecasting (HORIZON = 1)

- Assuming we are at time *t* ...
- ... predict the value at time *t+1* ...
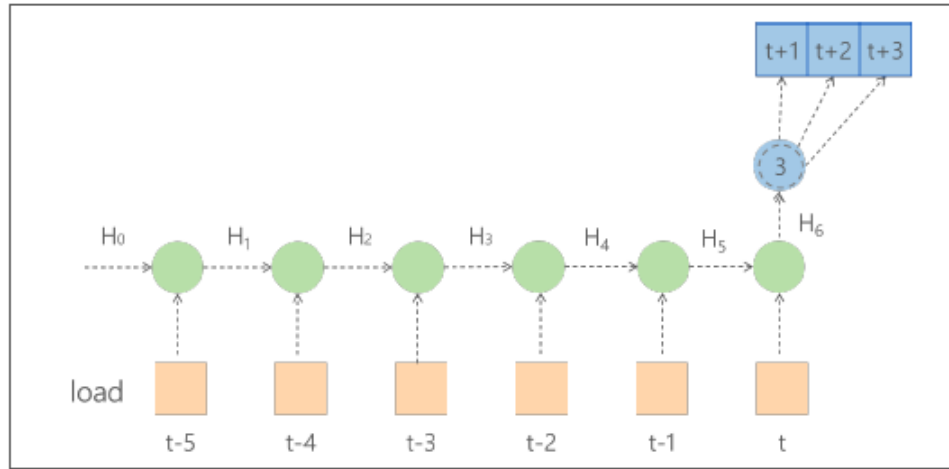- ... conditional on the previous *T* values of the time series

| *t-5* | *t-4* | *t-3* | *t-2* | *t-1* | *t* | t+1 |

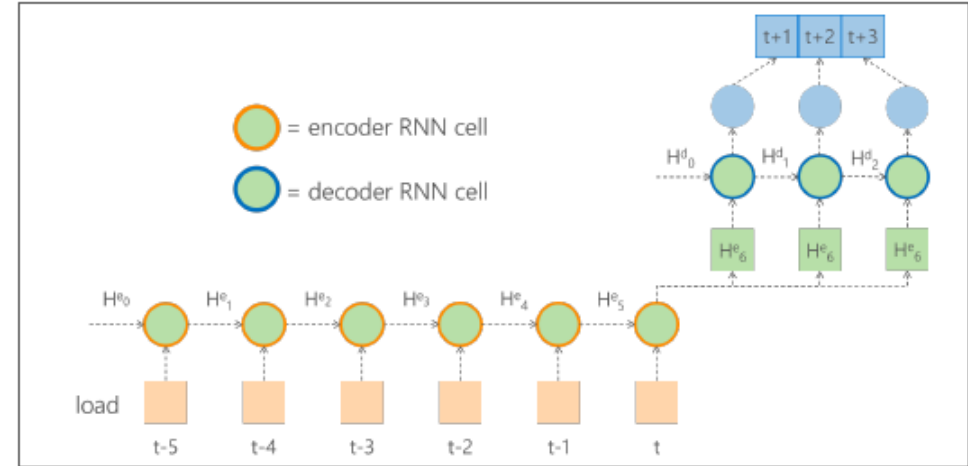T=6  ← find *T* through experimentation!

# Multi-step forecast

- Assuming we are at time $t$ ...
- ... predict the values at times $(t+1, ..., t+HORIZON)$ ...
- ... conditional on the previous $T$ values of the time series

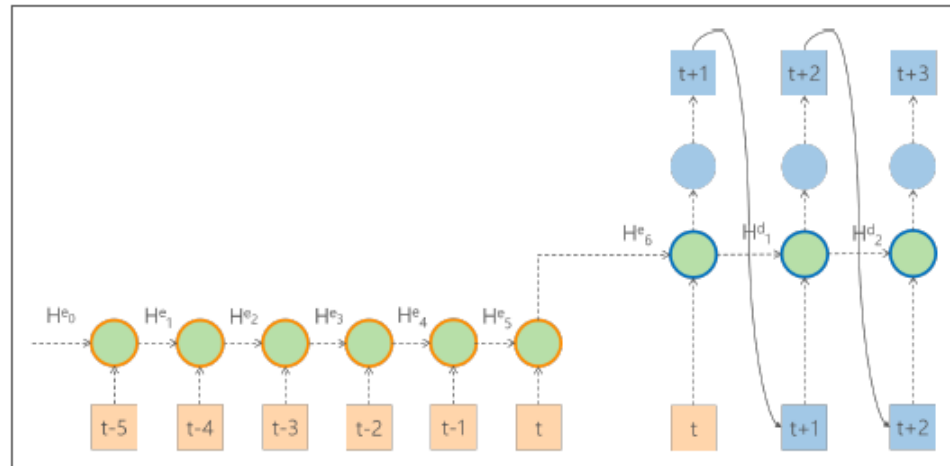| $t-5$ | $t-4$ | $t-3$ | $t-2$ | $t-1$ | $t$ | t+1 | t+2 | t+3 |
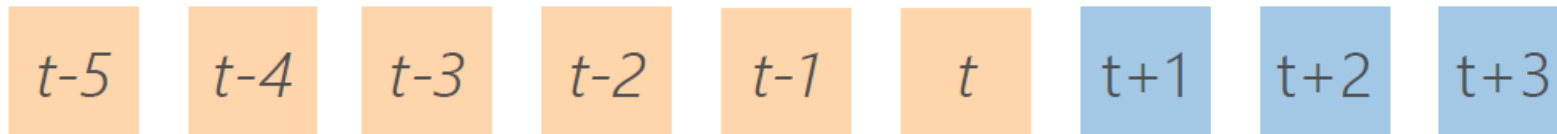
# Multi-step forecast



Vector output

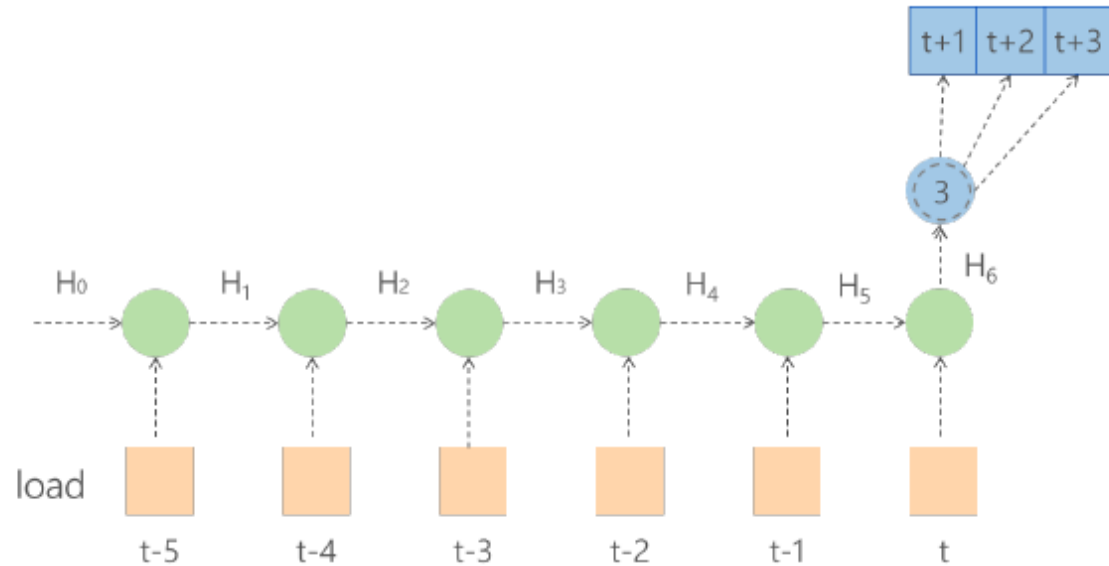Simple encoder-decoder
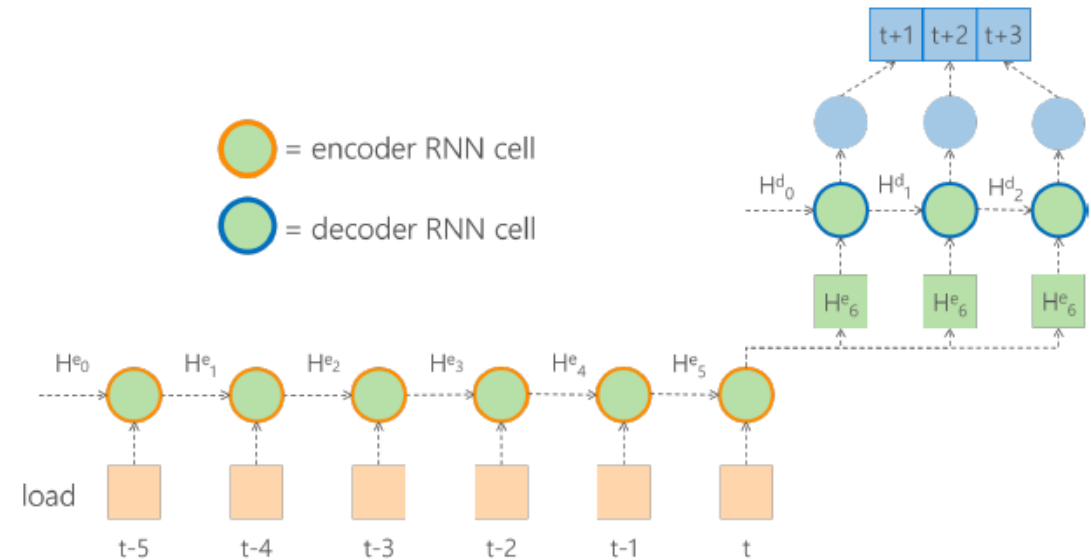
Recursive encoder-decoder

# Vector Output



👍Simplest to implement

👍Fastest to train

👎Does not model dependencies between predicted outputs
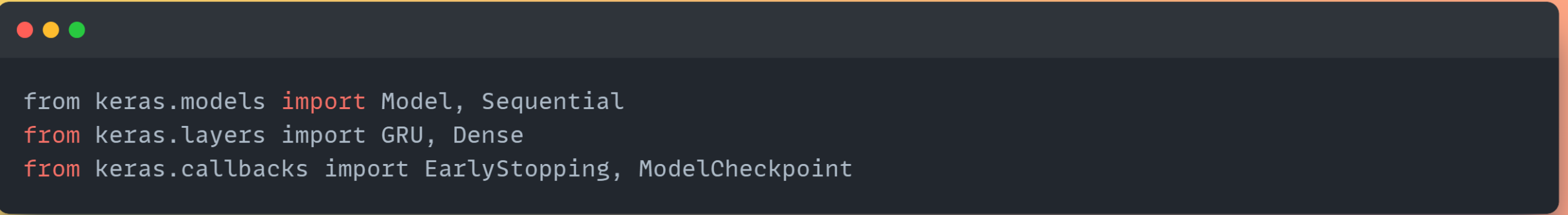
# Simple Encoder - Decoder



👍 Fairly simple to implement

👍 Tries to capture dependencies between forecasted time steps through decoder hidden state
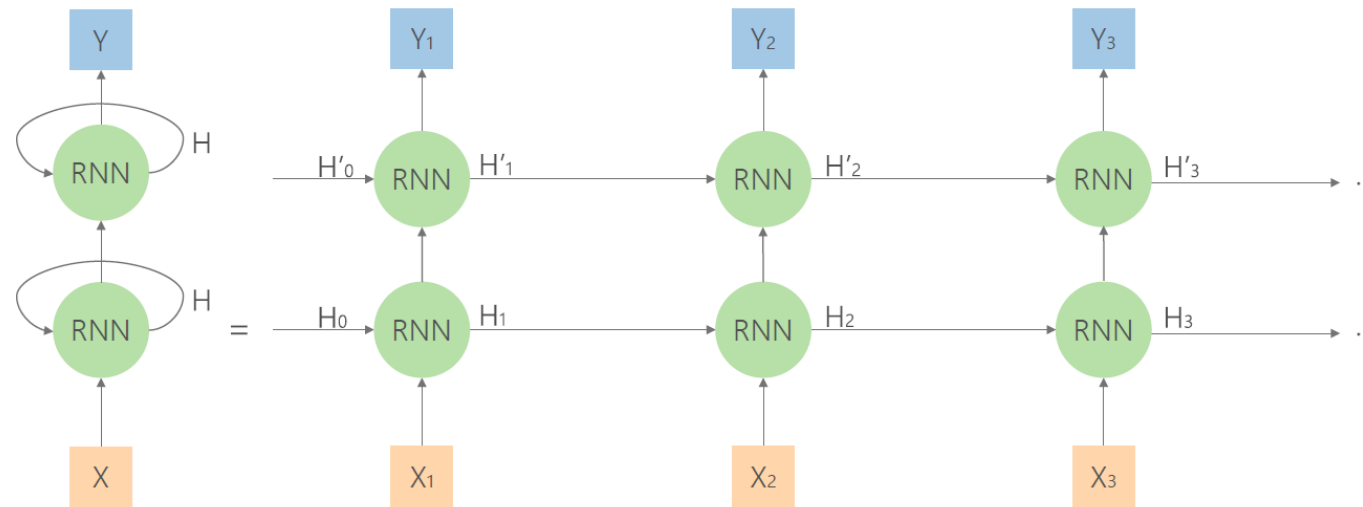
👎 Slower to train with stacked RNN layers

# Import in Keras

```python
from keras.models import Model, Sequential
from keras.layers import GRU, Dense
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

snappify.com

# Cells Stacking

To learn more complex relationships, we can stack multiple cells



```
model = Sequential()
model.add(tf.keras.layers.SimpleRNN(4, (return_sequences = True), (input_shape = (timesteps, data_dim))))
model.add(tf.keras.layers.SimpleRNN(4))
```
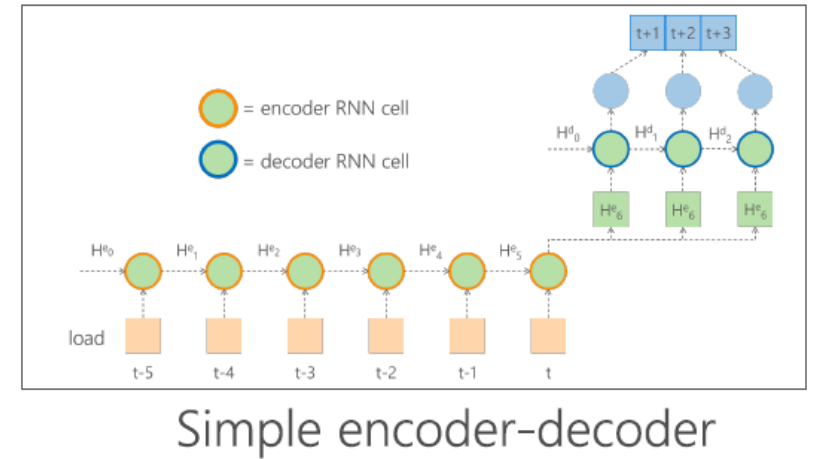
# The GRU layer – 1D time series

```python
model = Sequential()
model.add(GRU(LATENT_DIM, input_shape=(T, 1)))
model.add(Dense(HORIZON))
```

snappify.com

https://keras.io/api/layers/recurrent_layers/gru/

# Multistep forecast architecture 1D time series



Simple encoder-decoder

```python
NEW_HORIZON = ...
model = Sequential()
model.add(GRU(LATENT_DIM, input_shape=(T, 1)))
model.add(RepeatVector(NEW_HORIZON))
model.add(GRU(LATENT_DIM, return_sequences=True))
model.add(TimeDistributed(Dense(1)))
model.add(Flatten())N))
```
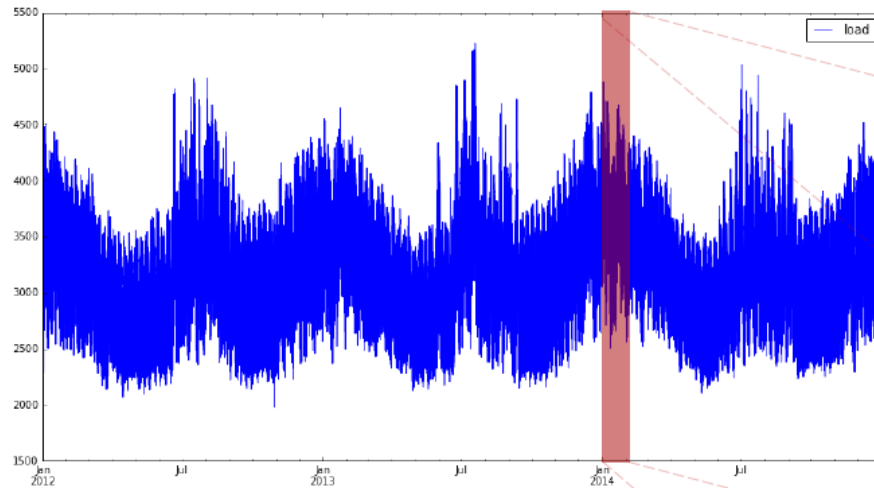
snappify.com

https://keras.io/api/layers/reshaping_layers/repeat_vector/

https://keras.io/api/layers/recurrent_layers/time_distributed/

# Presentation du TD

# Data



New England ISO data

- 26,000 hourly load values
- Annual, weekly and daily seasonality

Tao Hong, Pierre Pinson, Shu Fan, Hamidreza Zareipour, Alberto Troccoli and Rob J. Hyndman, "Probabilistic energy forecasting: Global Energy Forecasting Competition 2014 and beyond", International Journal of Forecasting, vol.32, no.3, pp 896-913, July-September, 2016.
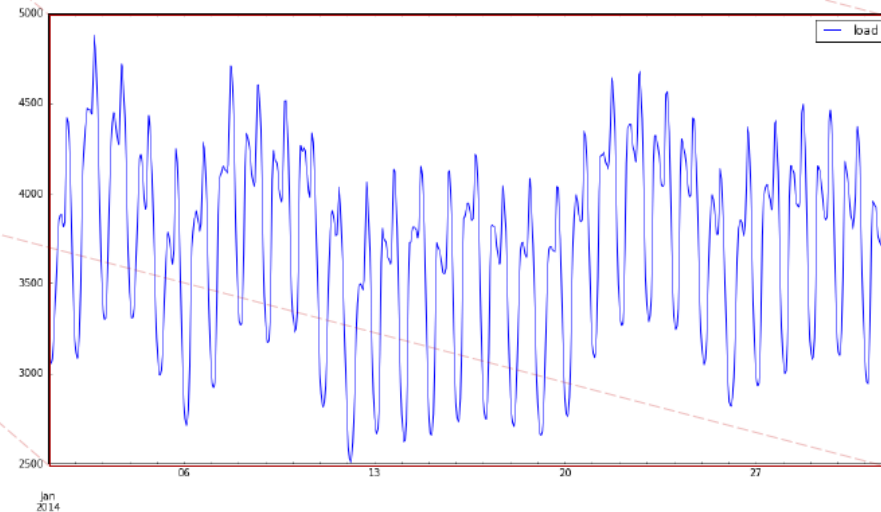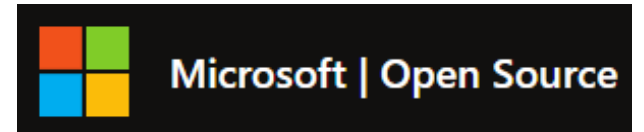
# Notebook

- Open the file (Python Notebook):

### TD_RNN_TS_FORECASTING.ipynb

- Instruction are contained in the notebook

# References

- **https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks**

- **DeepLearningForTimeSeriesForecasting (Microsoft) https://github.com/Azure/DeepLearningForTimeSeriesForecasting**

- **Ben Auffarth** Machine Learning for Time-Series with Python
- https://github.com/PacktPublishing/Machine-Learning-for-Time-Series-with-Python/tree/main/chapter10