Défis en Intelligence Artificielle

Défi 3 : L'IA pour l'analyse et la prévision de séries temporelles

Souhaib BEN TAIEB

UMONS
Université de Mons

Faculté
des Sciences

December 19, 2019

Kaggle competition

- `https://www.kaggle.com/c/hands-on-ai-umons-2019/overview`
- `https://www.kaggle.com/bsouhaib/starter`

Deadlines:

- December 19, 11:55pm: At least one Kaggle submission needs to have been made.
- January 8, 11:55pm: The Kaggle competition closes.
- January 15, 11:55pm: Upload to Moodle your project report and code, one per group.

Contents

Why deep learning models?

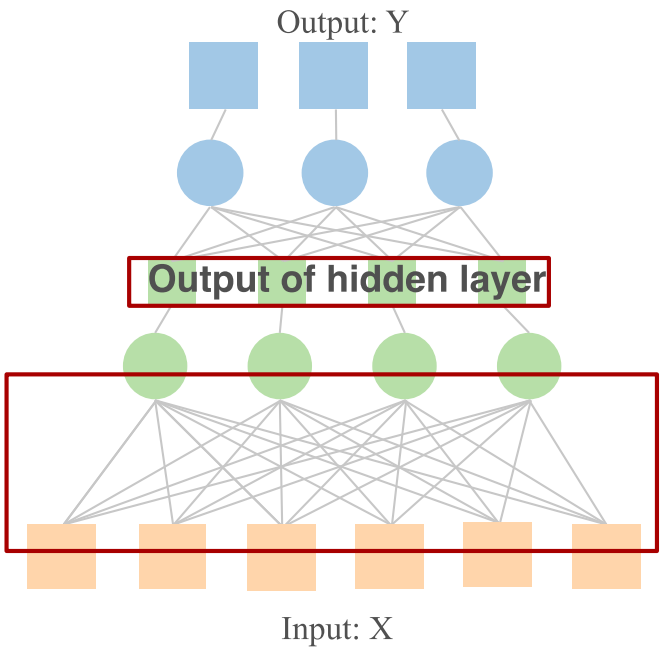
- Deep learning model has been shown perform well in many scenarios
  - 2014 Global Energy Forecasting Competition (link)
  - 2016 CIF International Time Series Competition (link)
  - 2017 Web Traffic Time Series Forecasting (link)
  - 2018 Corporación Favorita Grocery Sales Forecasting (link)
  - 2018 M4-Competition (link)

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

Part I

Convolutional Neural Networks

Fully connected vs convolution layer



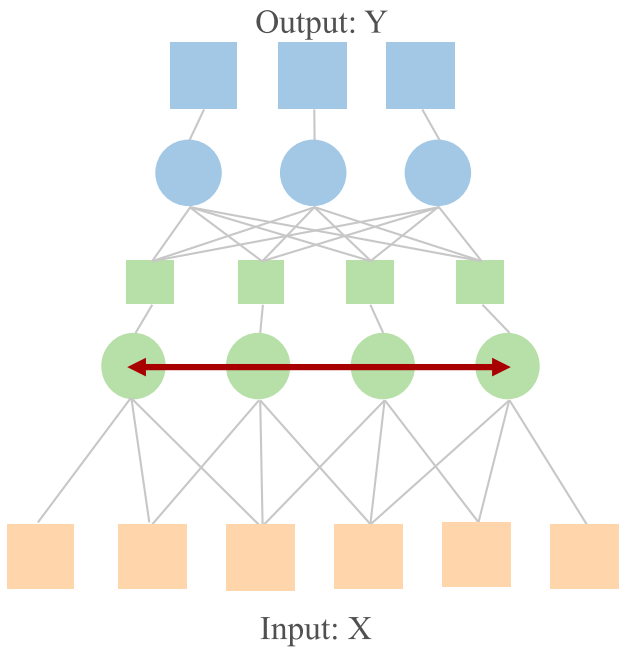
Fully connected:

- units in hidden layer are connected to every unit in previous layer

Fully connected vs convolution layer

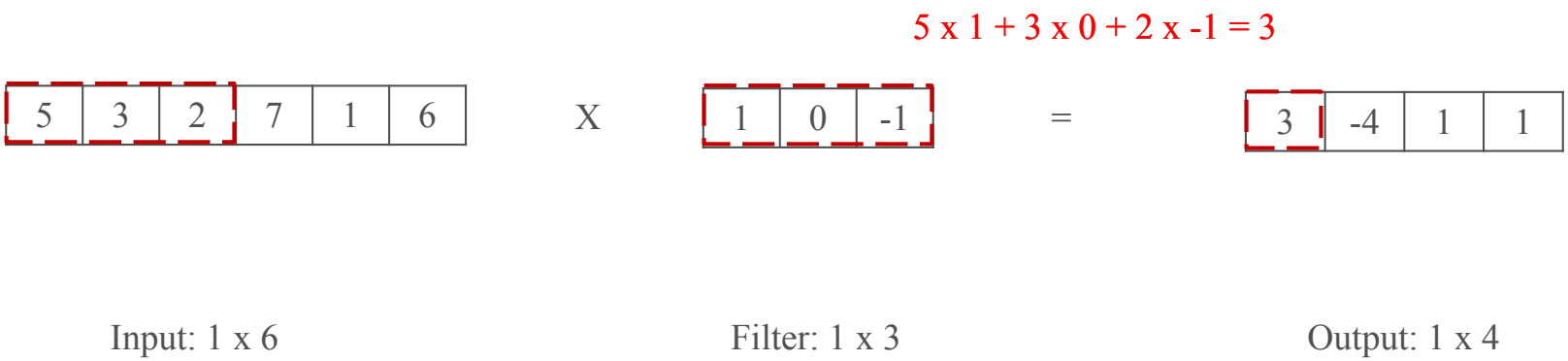
Convolution layer:

- units in hidden layer operate on a field of the input
- weights are shared across input



1D Convolutions

- Apply a 1D filter to all elements of a 1D input vector
- Result is the sum of the element-wise product



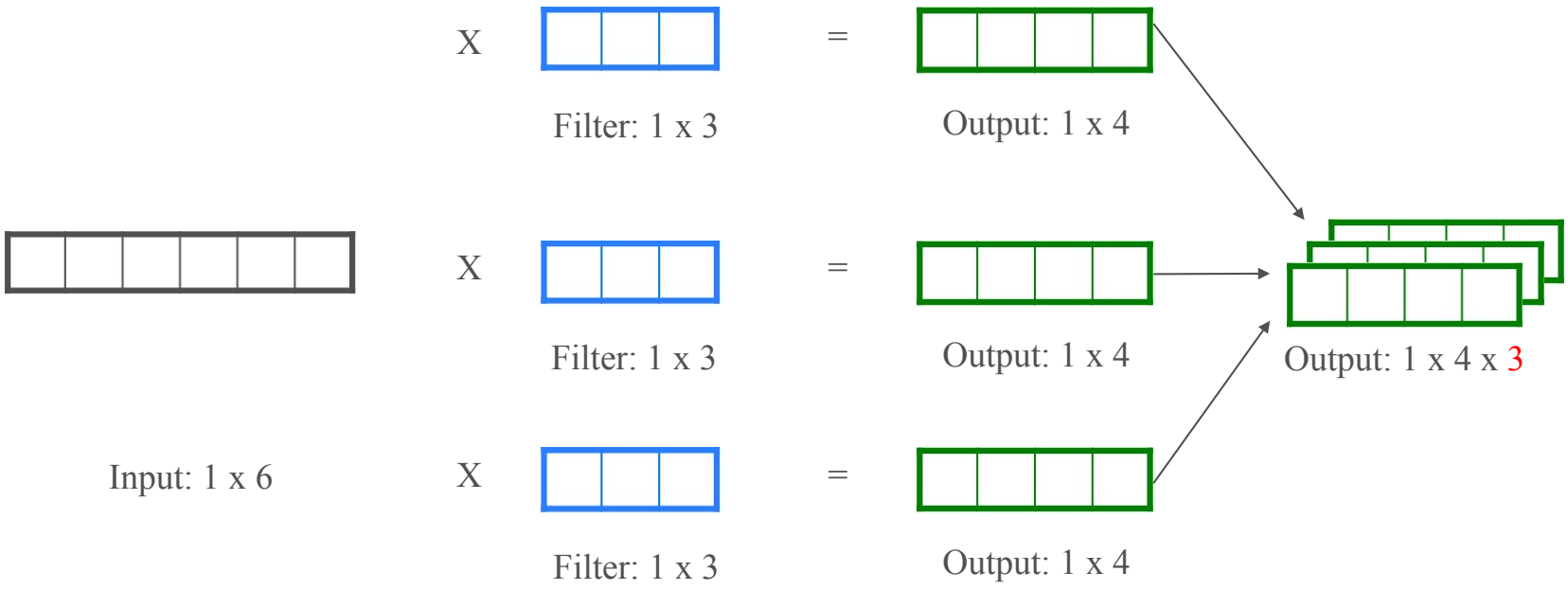
1D Convolutions

- Filters are trained to detect features in the input sequence
- Features can be detected regardless of where they appear in the input sequence

Input: 1 x 6 Filter: 1 x 3 Output: 1 x 4

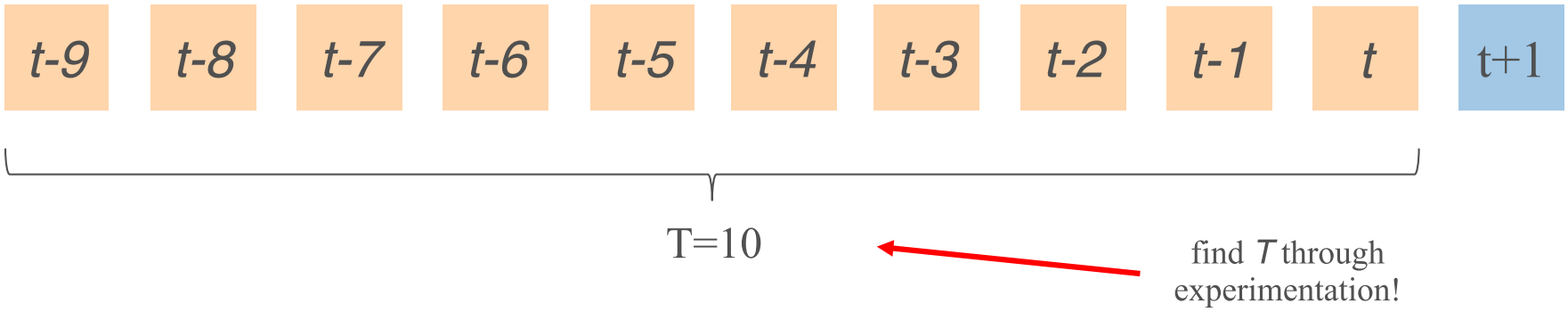
1D Convolutions

- Apply multiple filters to the input data to detect multiple features



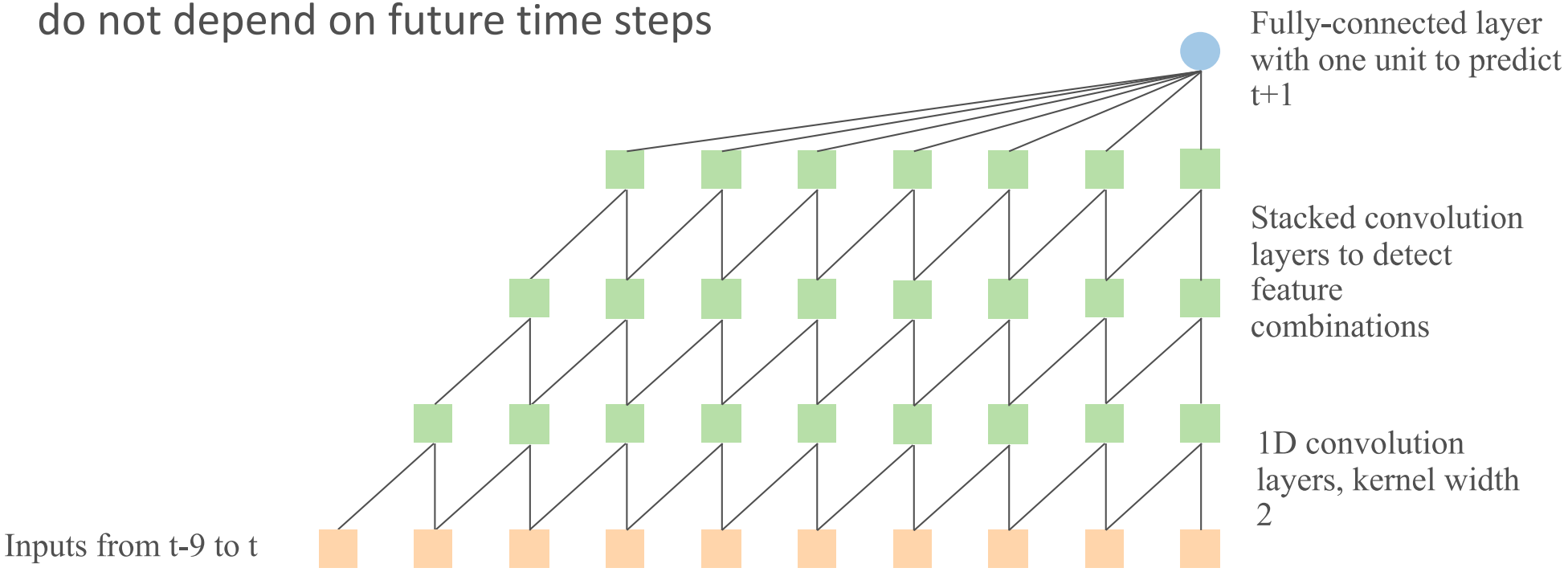
Application to forecasting

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



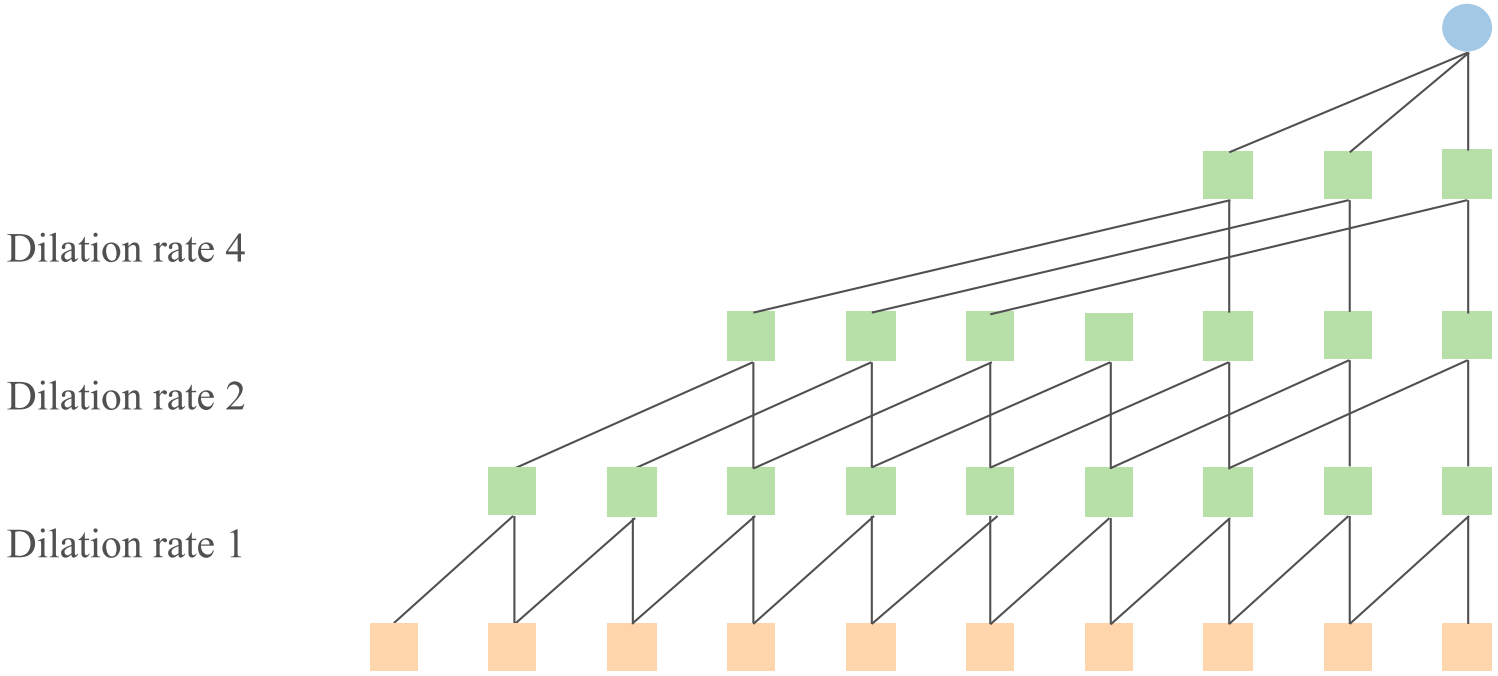
CNNs for forecasting

- Causal convolutions: the output at each time step do not depend on future time steps



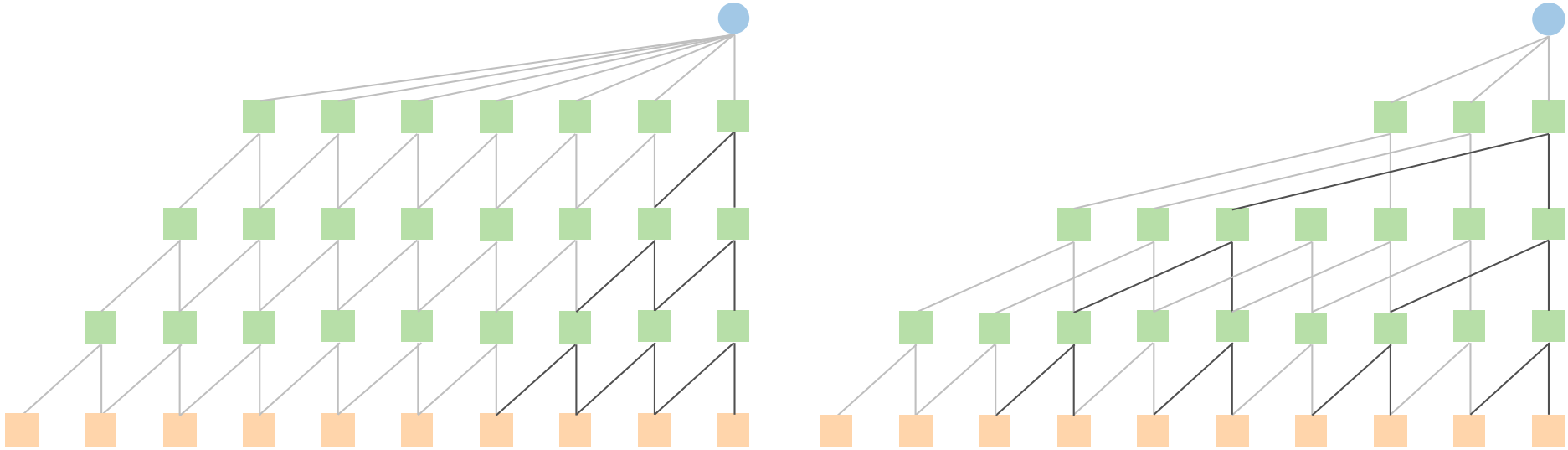
Dilated convolutions

- Skip outputs from previous layers



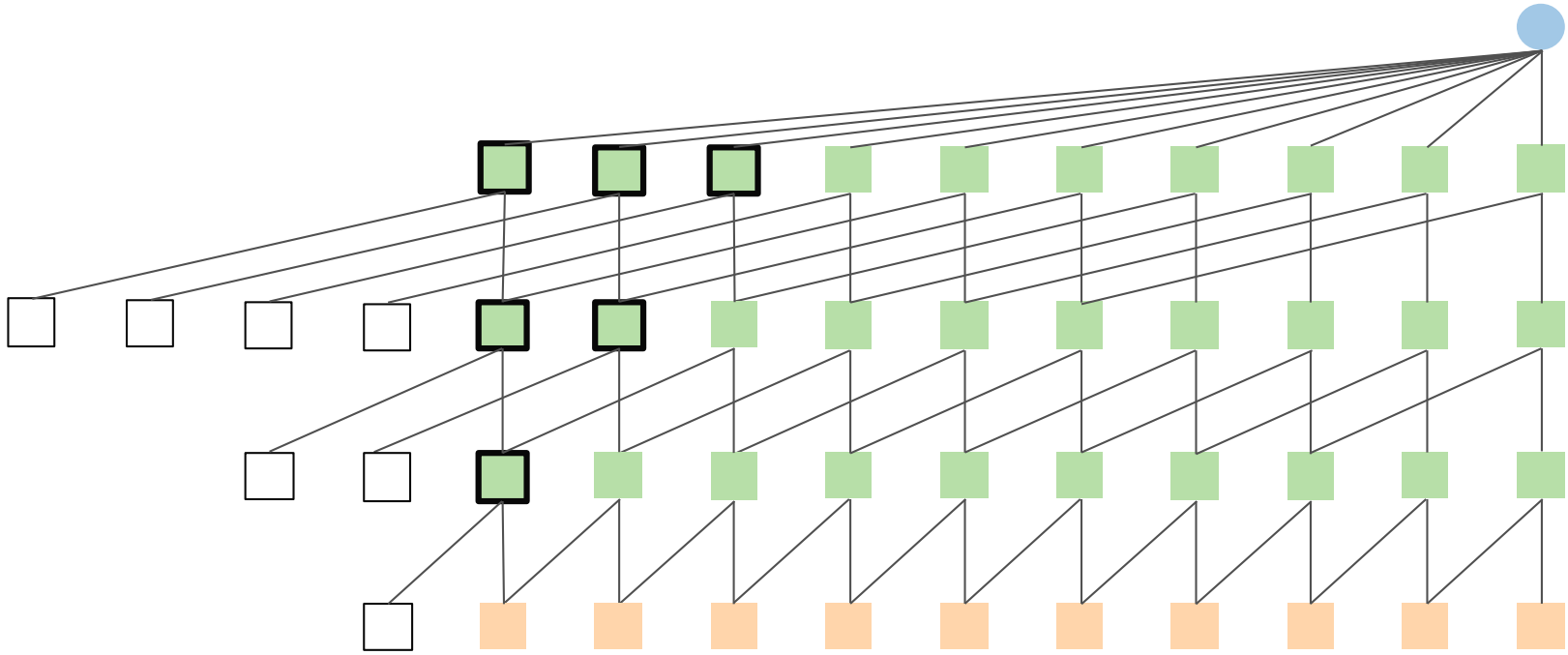
Why dilated convolutions?

- Normal convolutions = more connections = more weights to train
- Reach information from more distant values in the time series



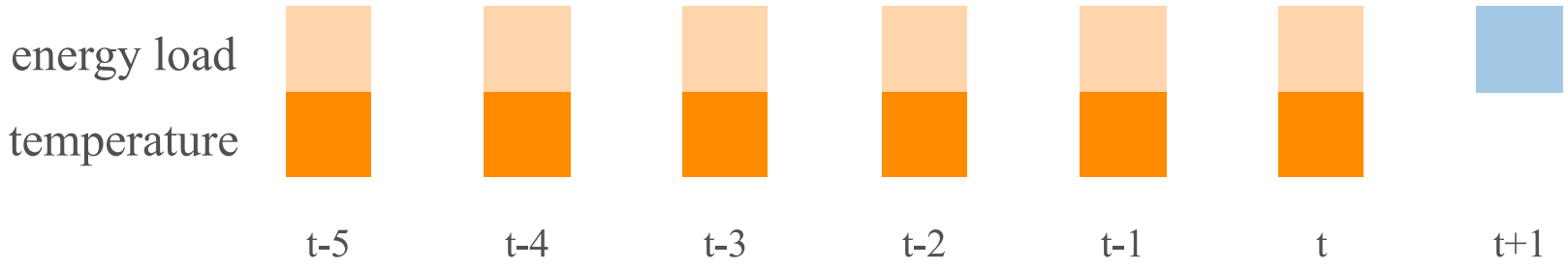
Causal padding

- Padding is necessary to preserve output dimension
- Allows all filter weights to apply to all inputs



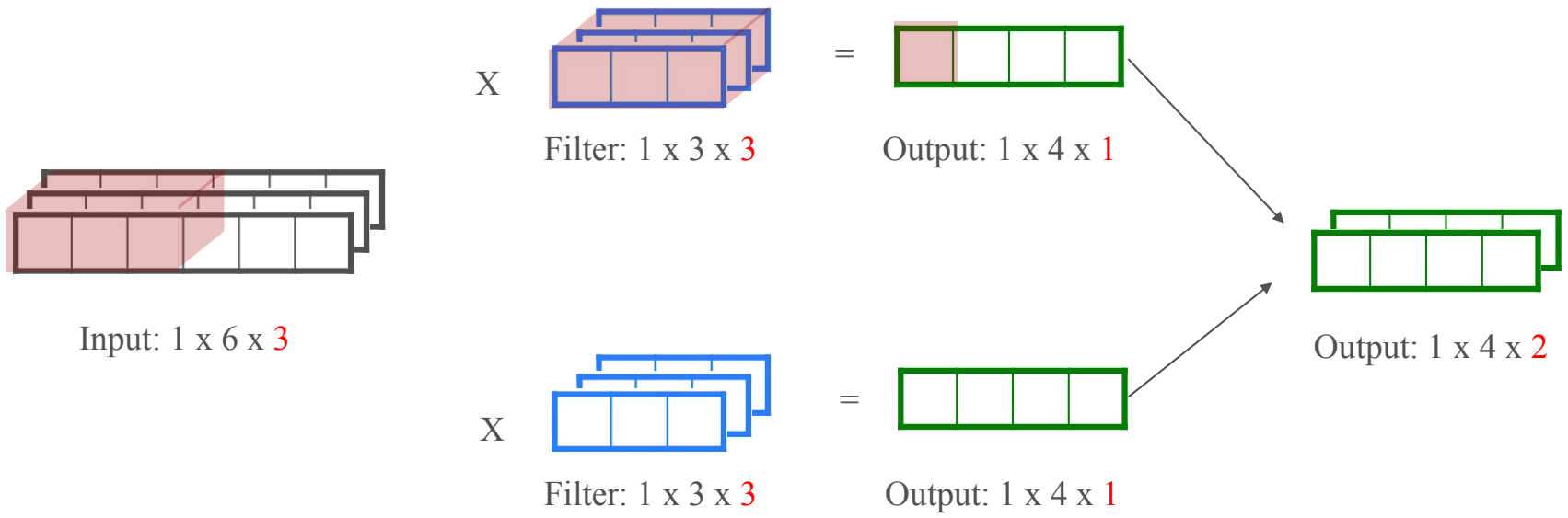
Multivariate time series

- Multiple time series in the input sequence



Multivariate time series

- Example: for 3 input time series use 3 channels



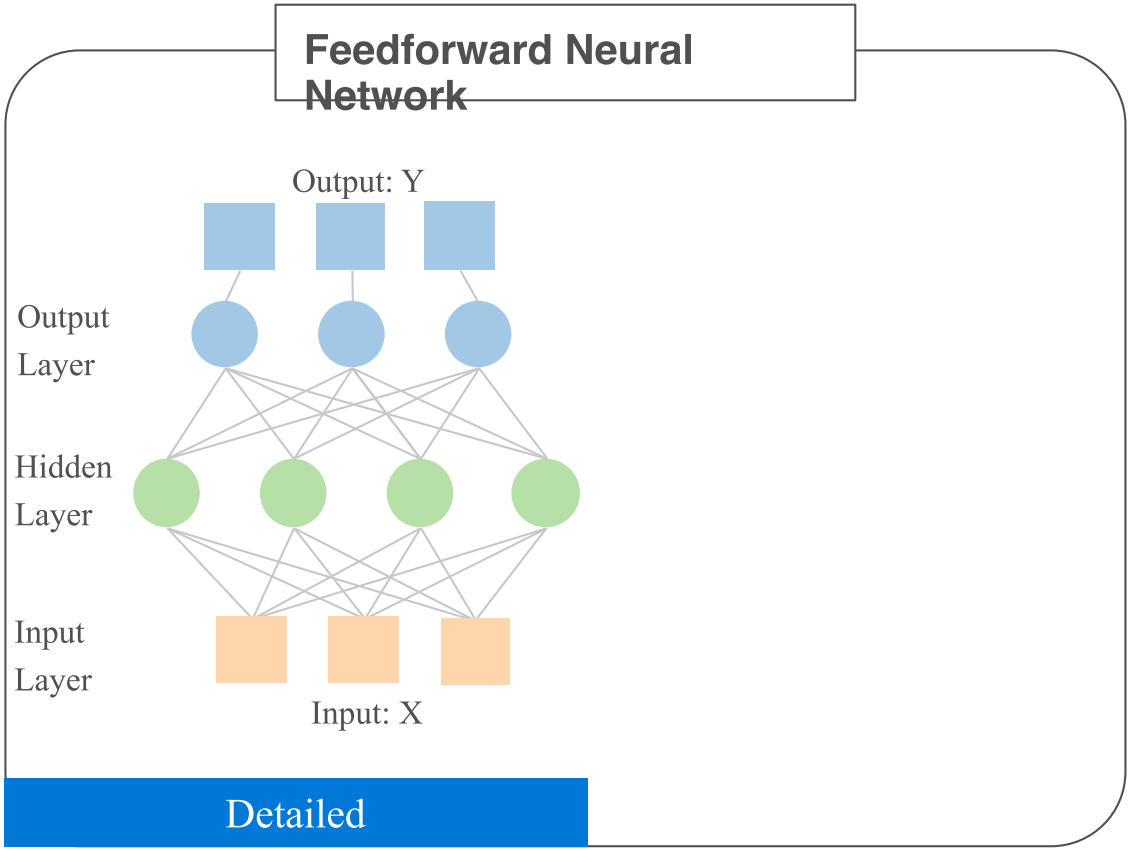
Part II

Reccurent Neural Networks

Agenda

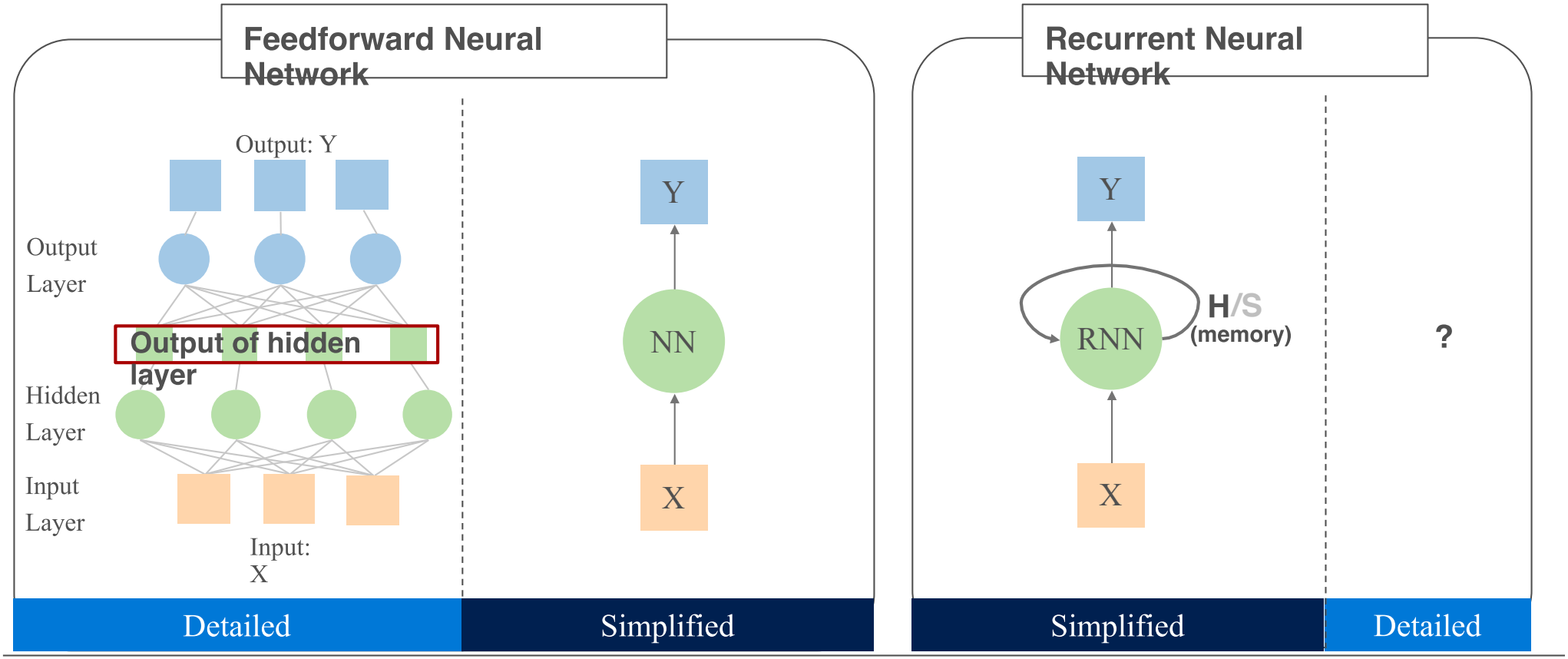
- What are RNNs?
- How RNNs are trained: Backpropagation through time (BPTT)
- Vanilla RNN and its gradient problems
- Other RNN units
  - GRU
  - LSTM
- RNN stacking
- RNN for one step time series forecasting
- Encode-decoder RNN for multi-step time series forecasting

What are RNNs?



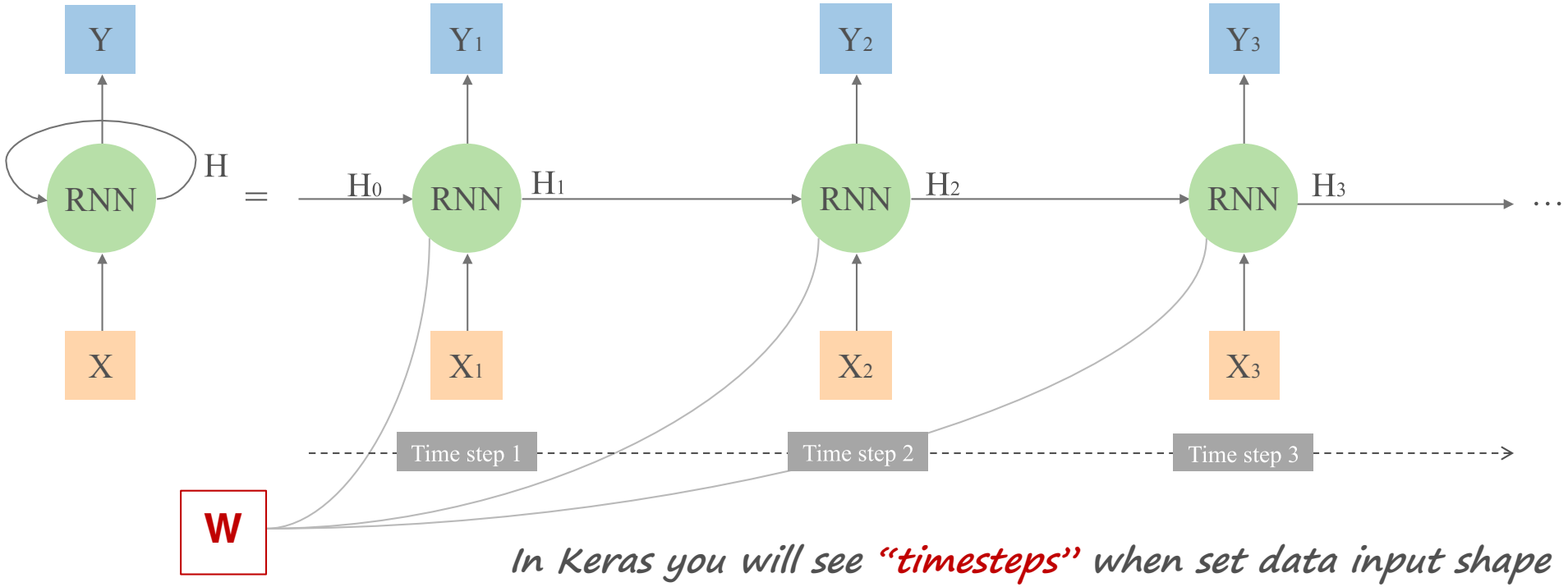
What are RNNs?

RNN has internal hidden state which can be fed back to network



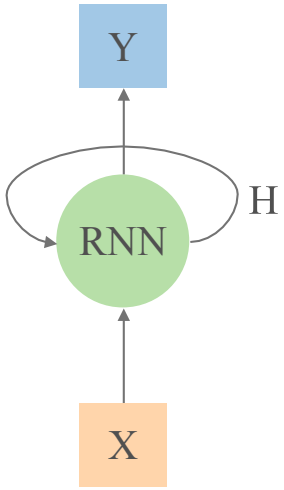
Unrolled RNN

The same weight and bias shared across all the steps



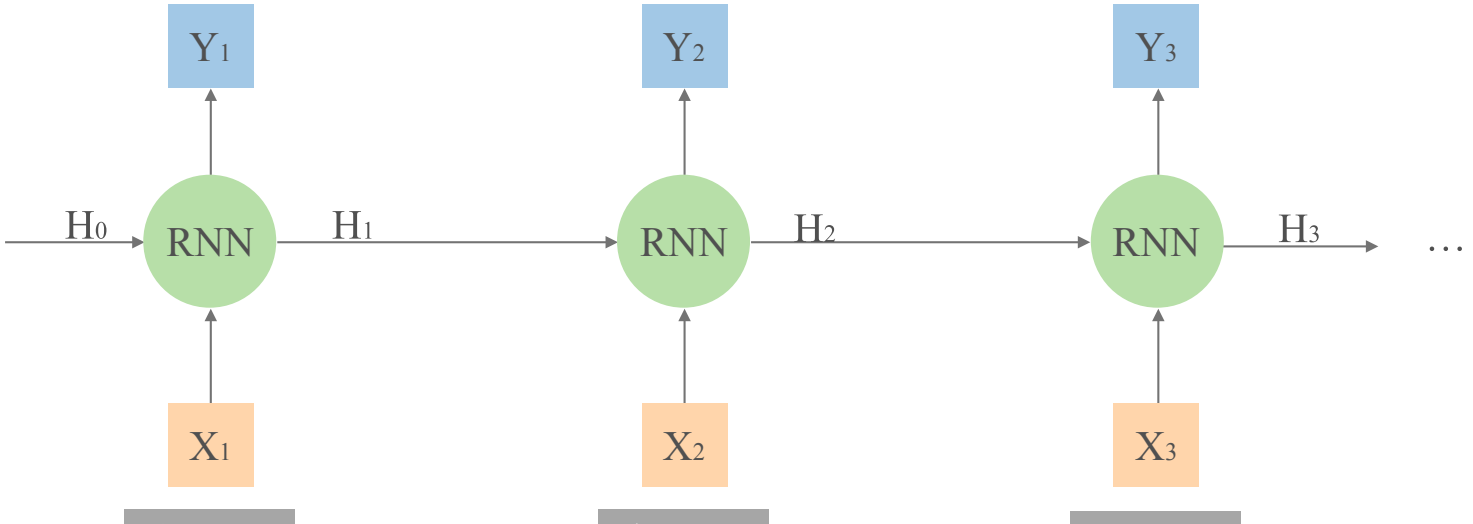
In Keras you will see "timesteps" when set data input shape

Unrolled RNN

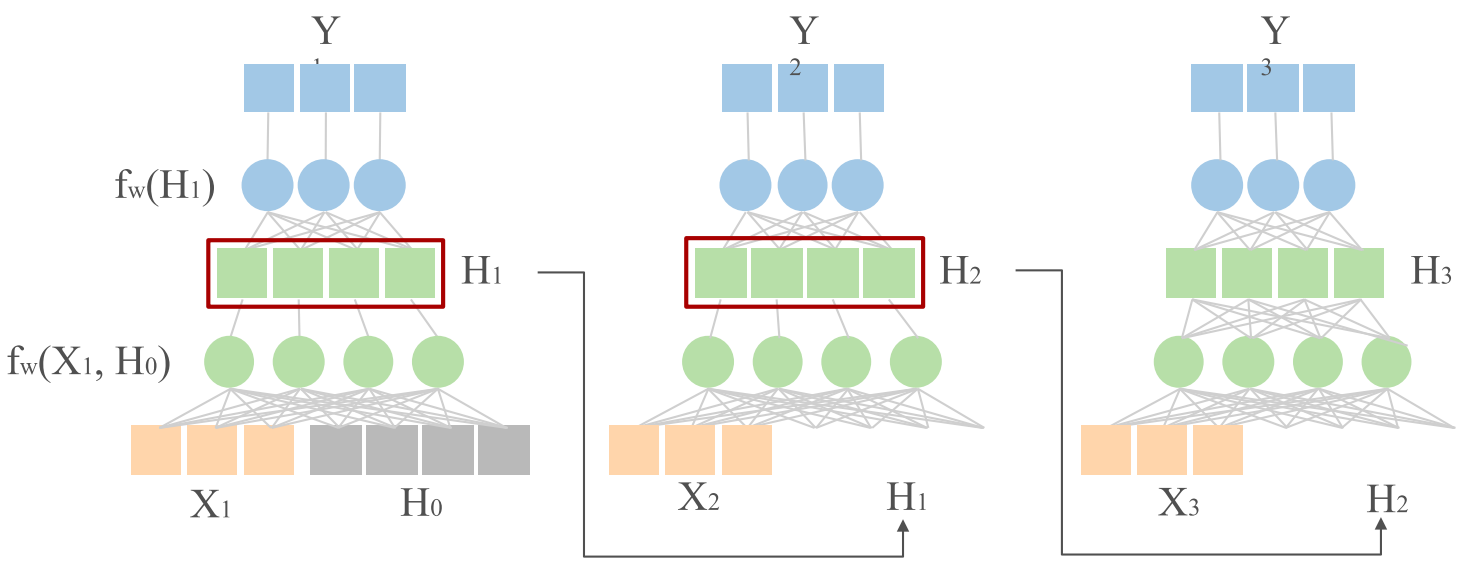


=

Simplified



Detailed

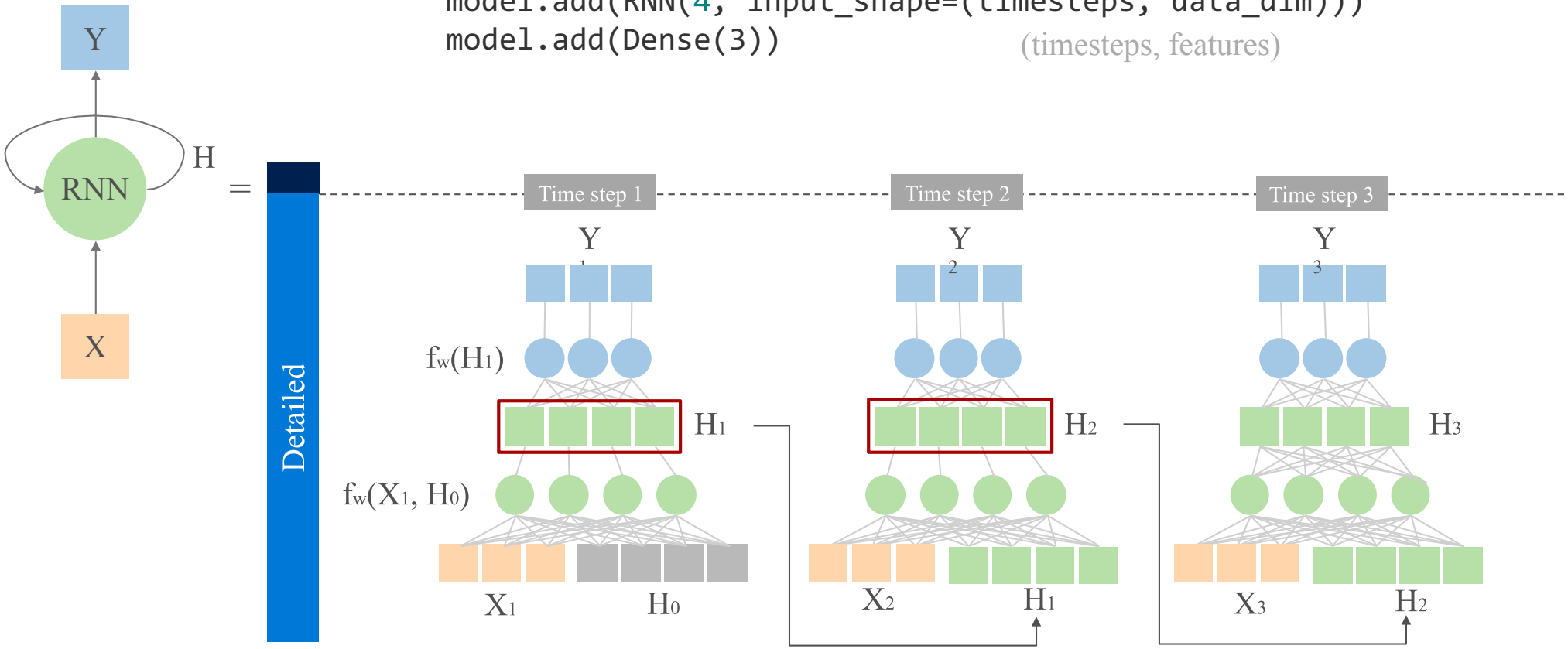


Unrolled RNN

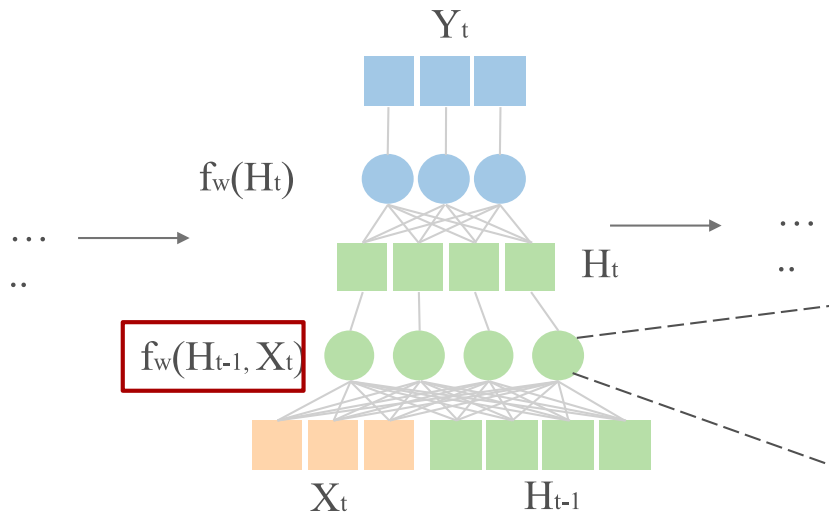
In Keras, the parameter "units" is dimensions of hidden state.
(Think of it as feedforward neural network number of units in hidden layer.)

```
model = Sequential()
model.add(RNN(4, input_shape=(timesteps, data_dim)))
model.add(Dense(3))
```

(timesteps, features)



Vanilla RNN



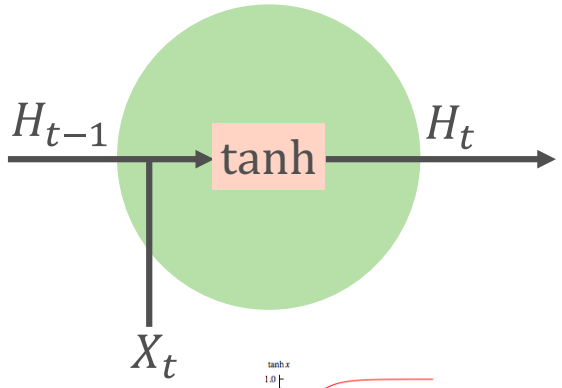
some function with parameter W

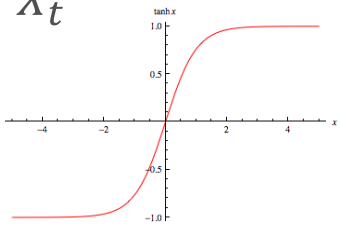
$$H_t = f_W(H_{t-1}, X_t)$$

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

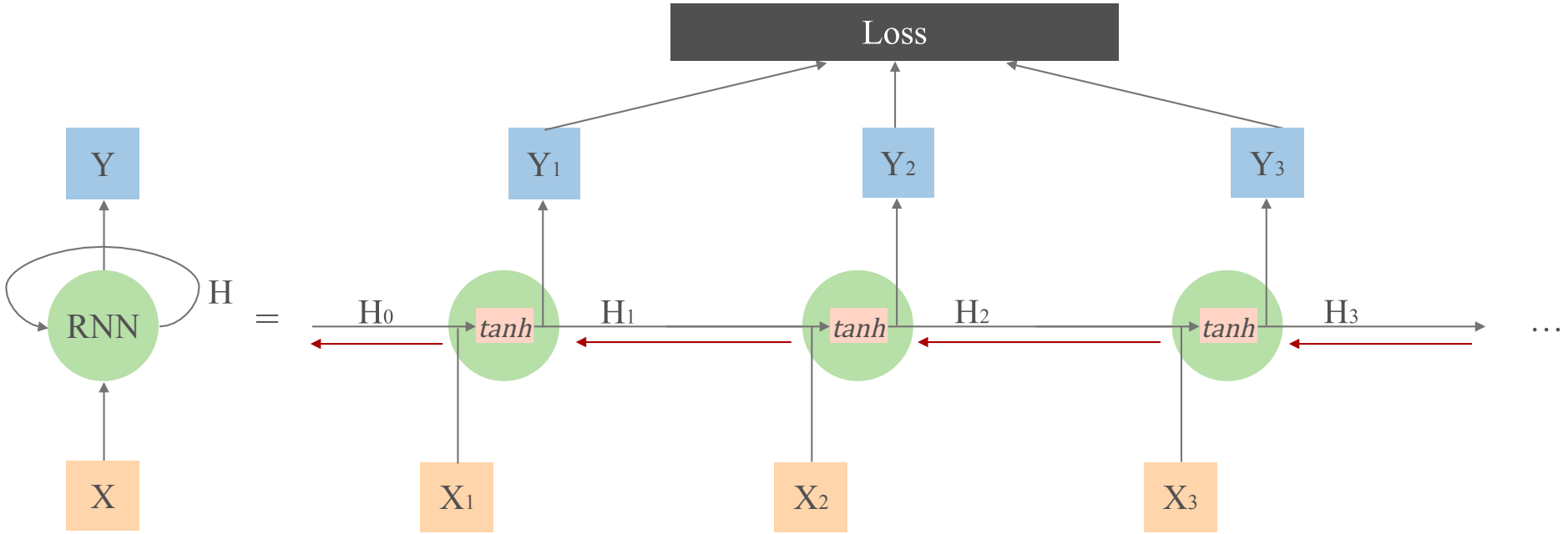
Vanilla RNN:

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





Vanilla RNN BPTT



Computing gradient of h_0 involves repeated tanh and many factors of W

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*……)

100 time steps is similar to 100 layers feedforward neural net

Exploding Gradient

- Exploding gradients are obvious. Your gradients will become NaN (not a number) and your program will crash
- Solution: Gradient clipping

  Clip the gradient when it goes higher than a threshold

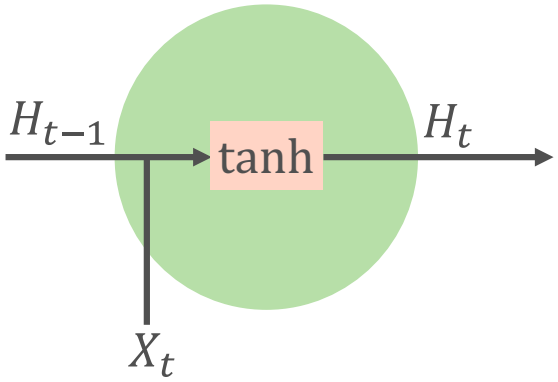
Vanishing Gradient

- Vanishing gradients are more problematic because it's not obvious when they occur or how to deal with them
- Solutions:
  - Change activation function to ReLU
  - Proper initialization
  - Regularization
  - Change architecture to LSTM or GRU

Gated Recurrent Unit (GRU)

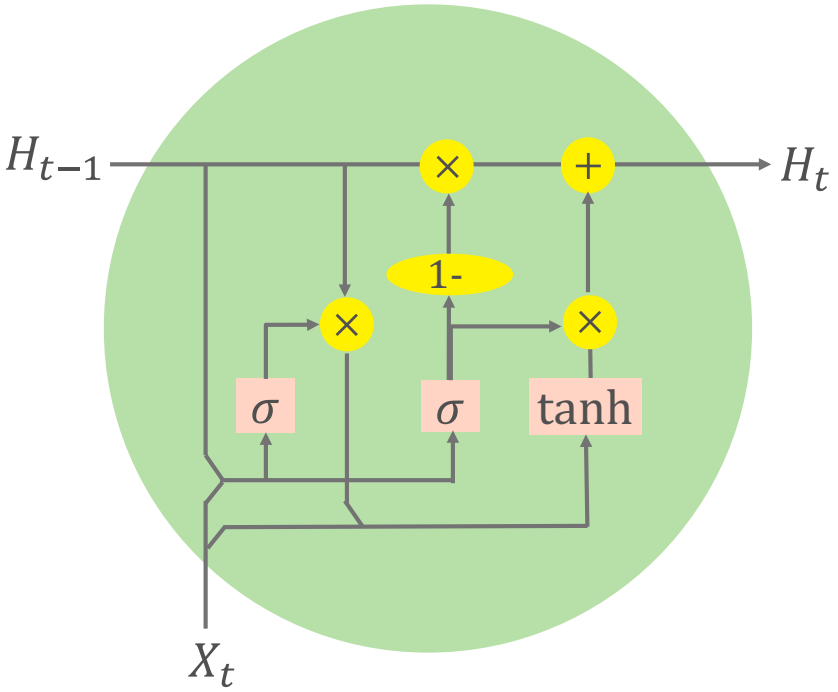
Vanilla RNN:

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

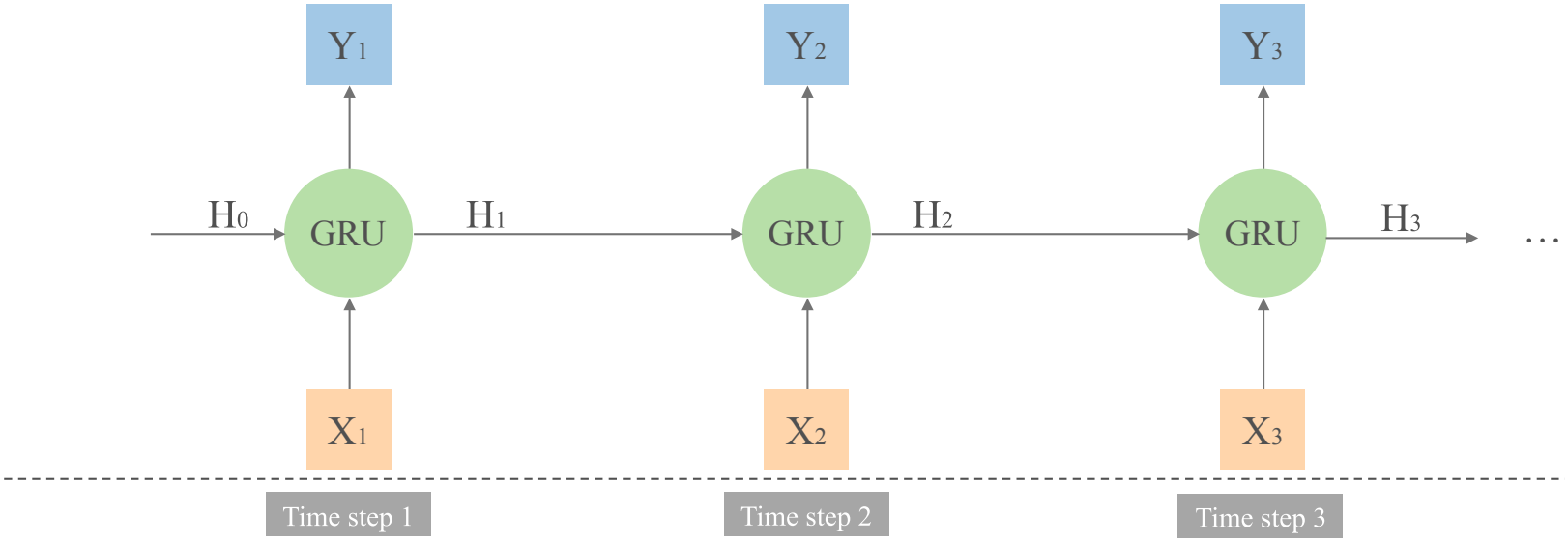


GRU:

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

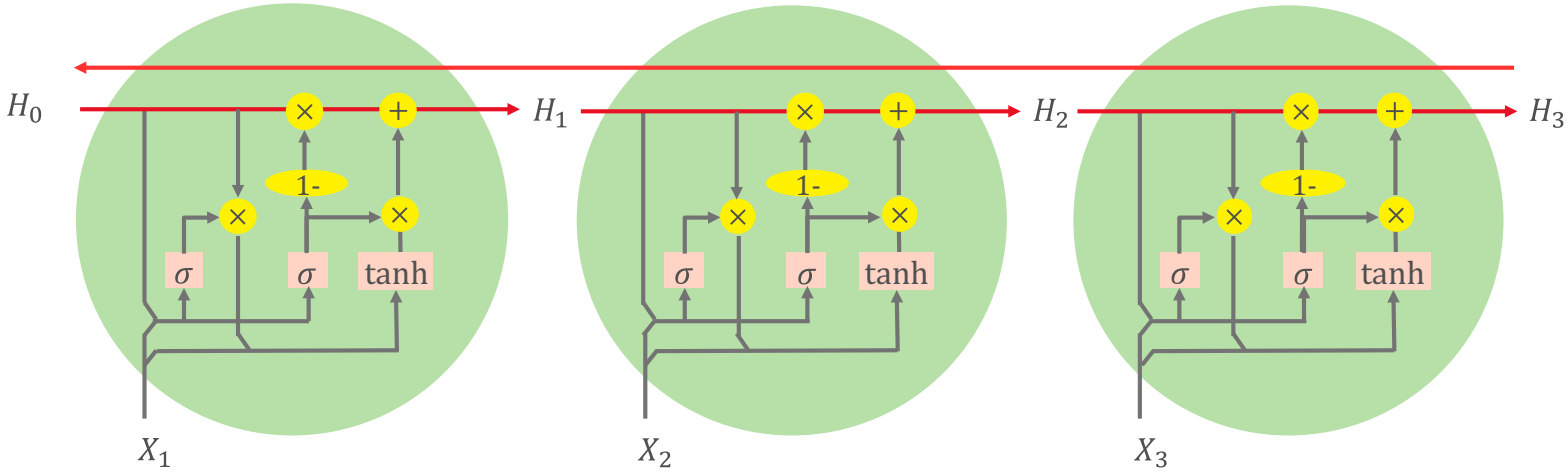


GRU

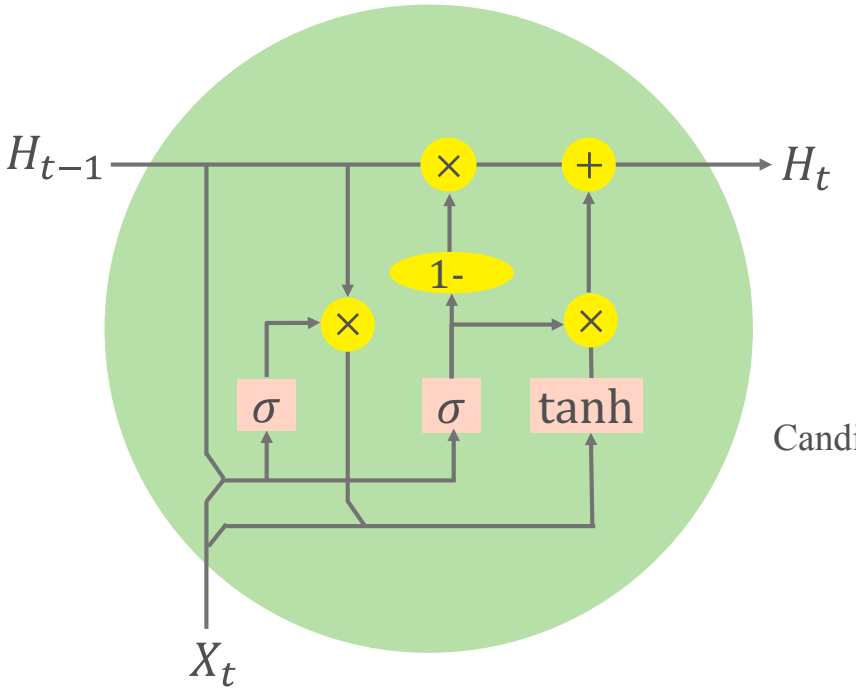


Uninterrupted gradient flow

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



Gated Recurrent Unit (GRU)



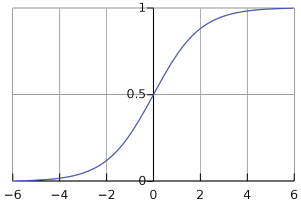
Hidden state: $H_t = (1 - z_t) \times H_{t-1} + z_t \times \widetilde{H}_t$

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

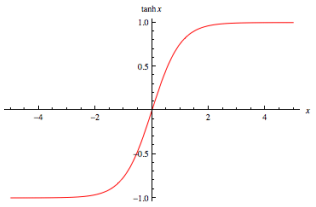
Candidate gates/states: $\widetilde{H}_t = \tanh(W \cdot [r_t \times H_{t-1}, X_t])$

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

σ
(0,1)



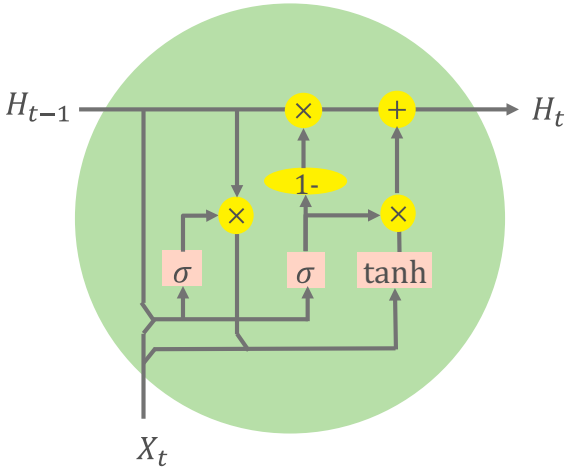
tanh
(-1, 1)



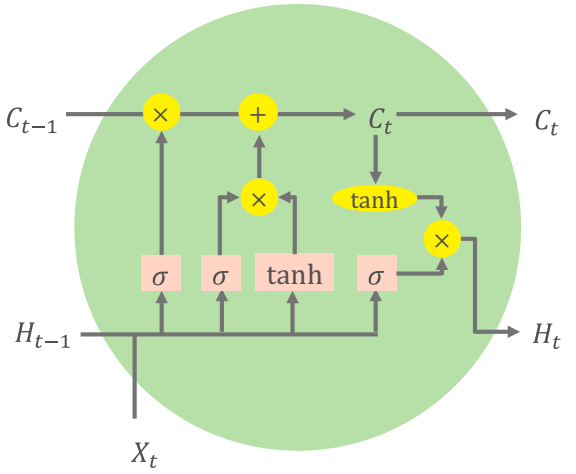
GRU [Learning phrase representations using rnn encoderdecoder for statistical machine translation, Cho et al. 2014]

GRU vs LSTM (Long Short Term Memory)

GRU



LSTM



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

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: $\tilde{H}_t = \tanh(\ W \cdot [r_t \times H_{t-1}, X_t]\ )$

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

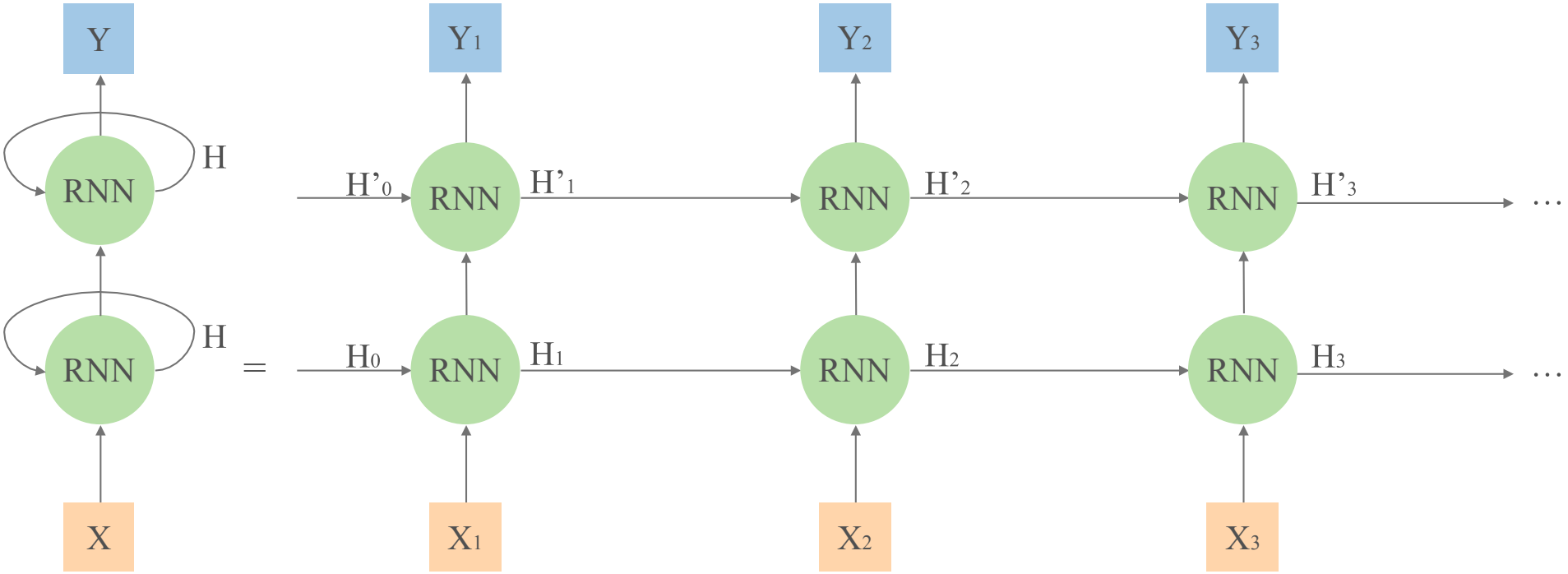
Output gates: $o_t = \sigma(\ W_o \cdot [H_{t-1}, X_t]\ )$

LSTM: A Search Space Odyssey, Greff et al., 2015

Long Short-Term Memory, Hochreiter & Schmidhuber (1997)

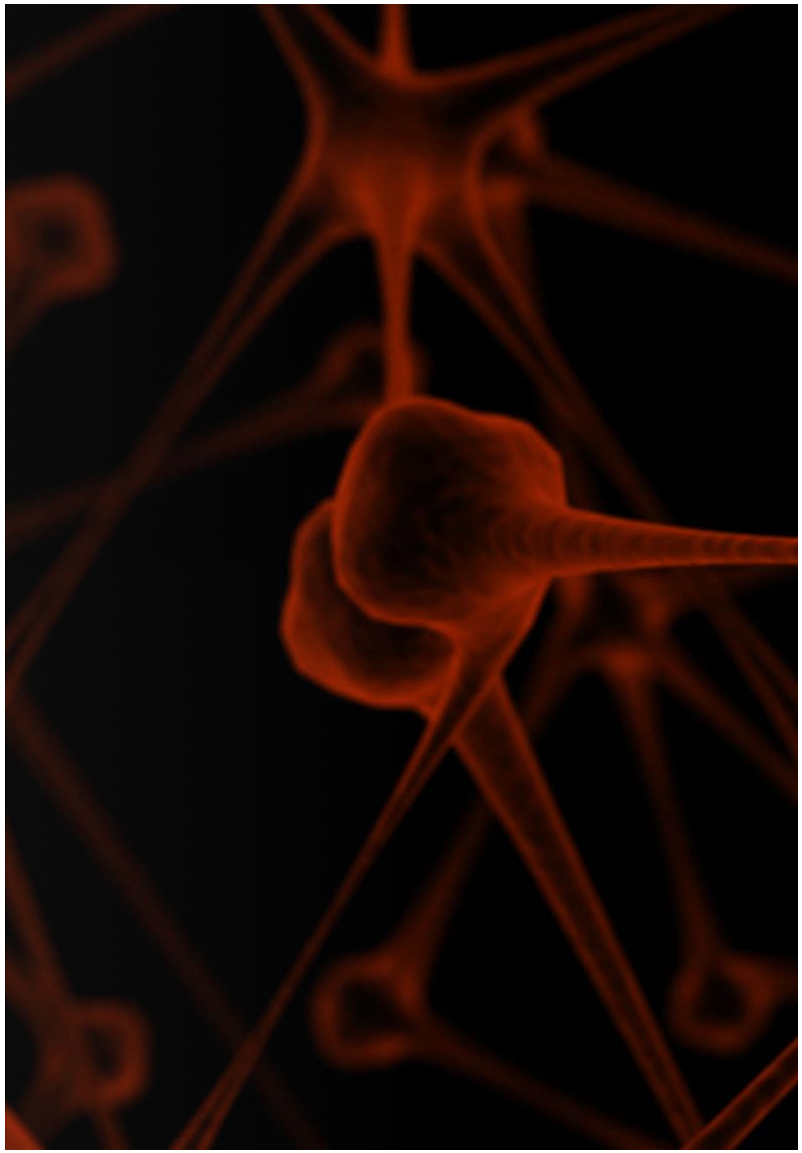
RNN Stacking

To learn more complex relationships, we can go deep by stacking the cells.



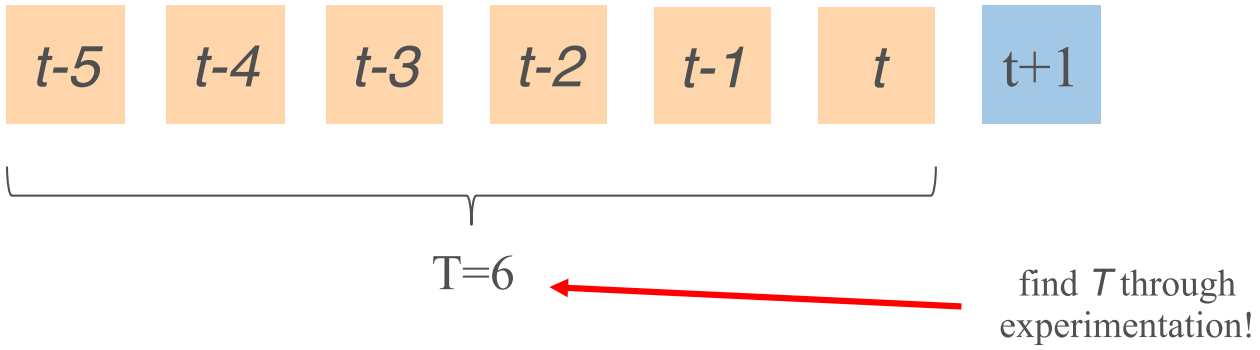

```
model = Sequential()
model.add(RNN(4, return_sequences=True, input_shape=(timesteps, data_dim)))
model.add(RNN(4))
```

RNN for one step time series forecasting

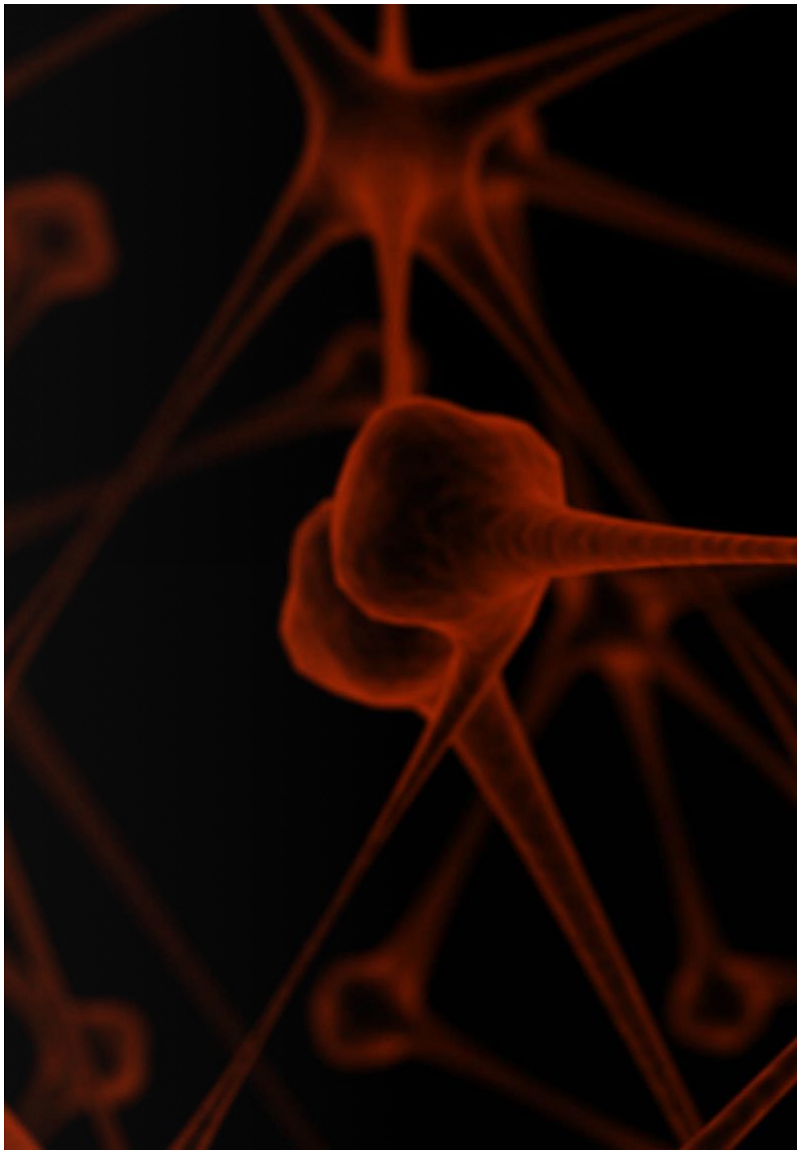


One-step forecast

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

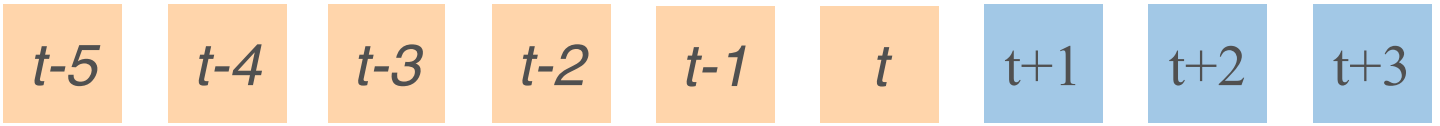


RNN for multi-step time series forecasting

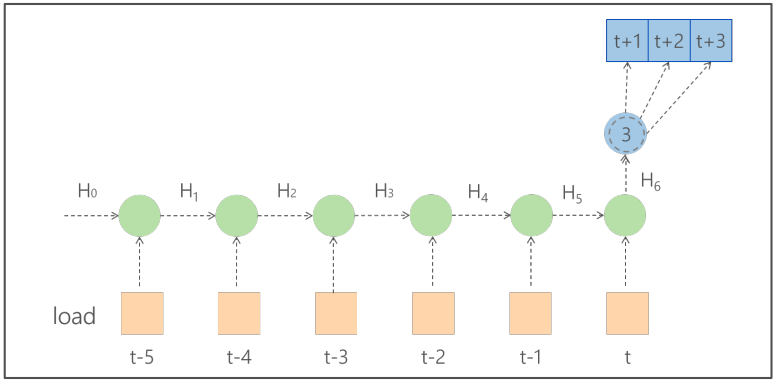


Multi-step forecast

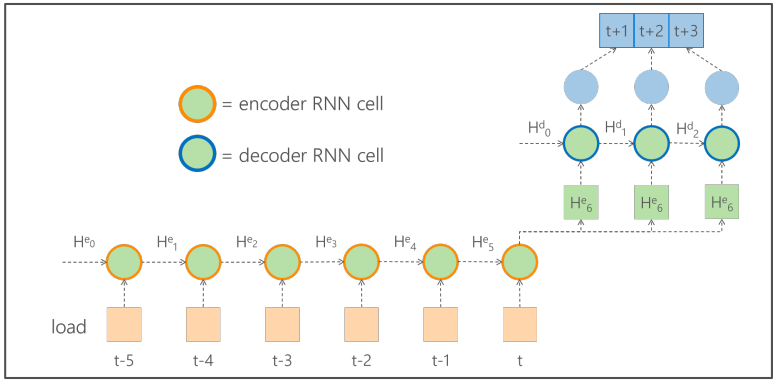
- Assuming we are at time *t* …
- … predict the values at times (*t+1, … , t+HORIZON*) …
- … conditional on the previous *T* values of the time series



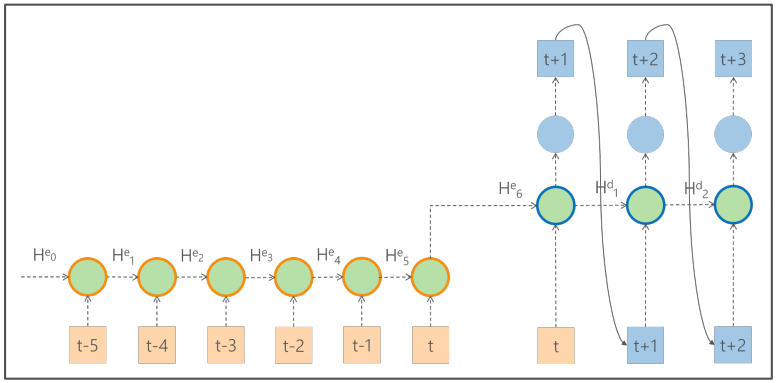
Multi-step forecast



Vector output

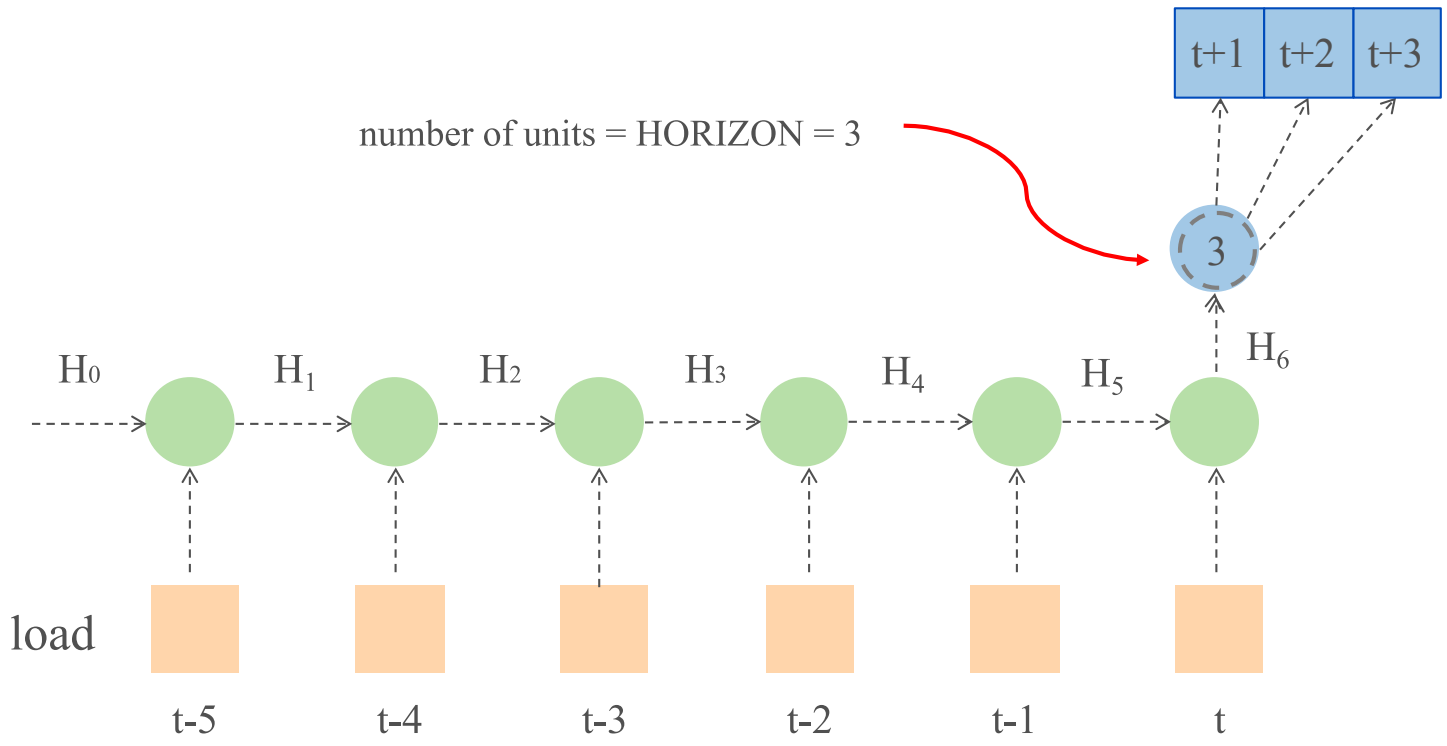


Simple encoder-decoder

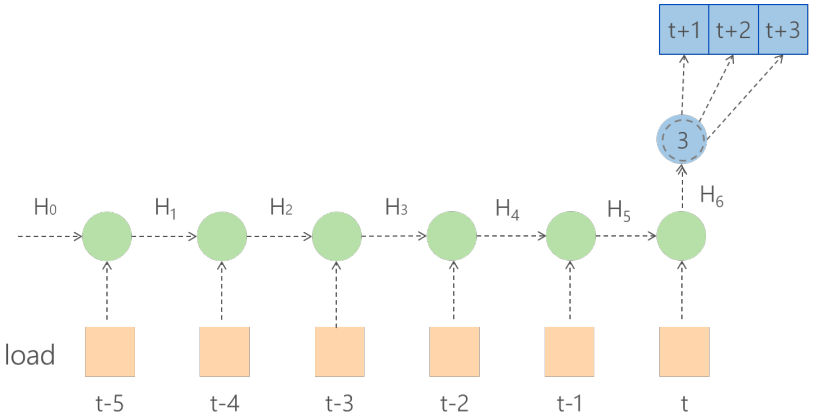


Recursive encoder-decoder

Vector output approach

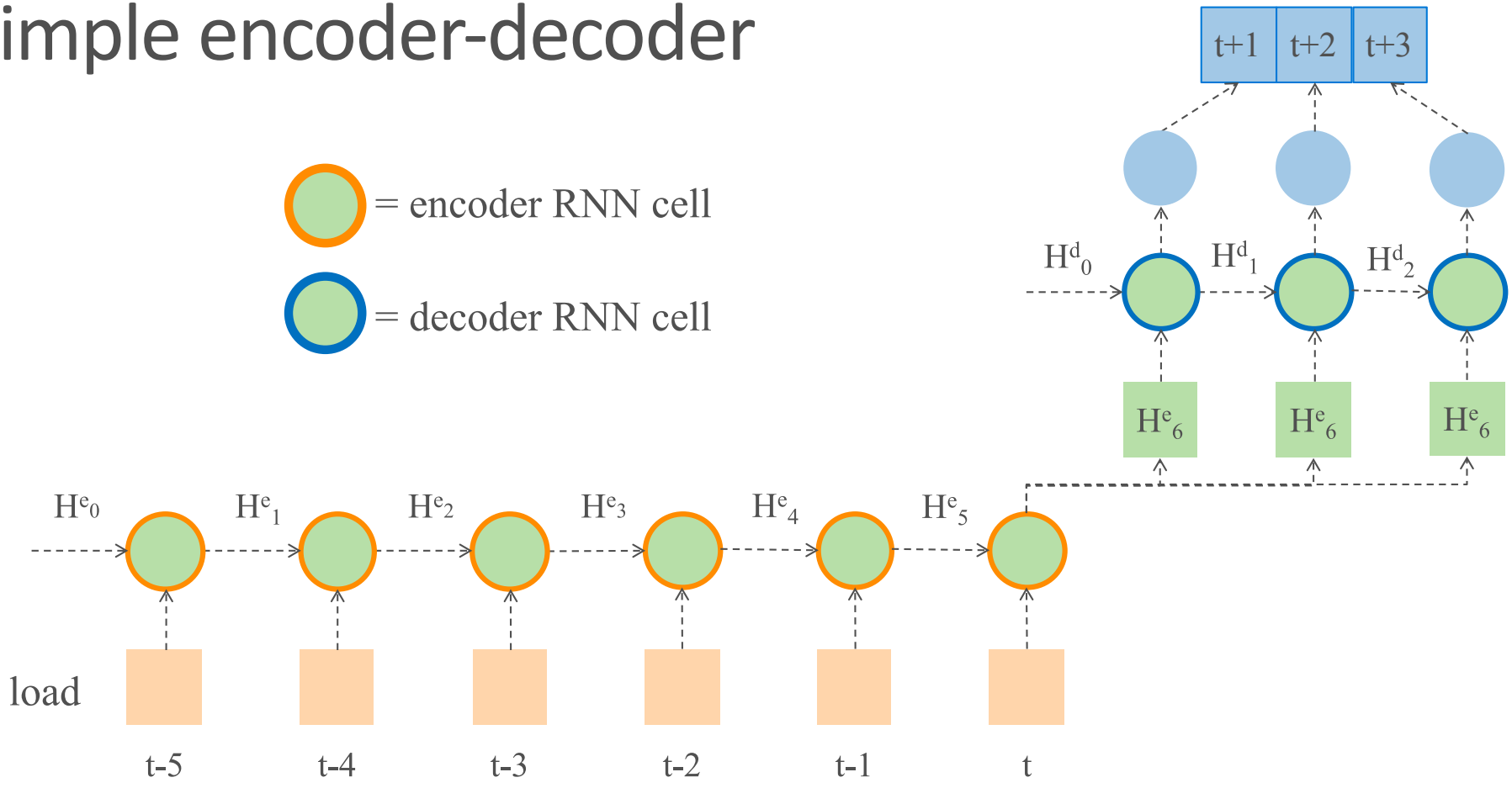


Vector output approach

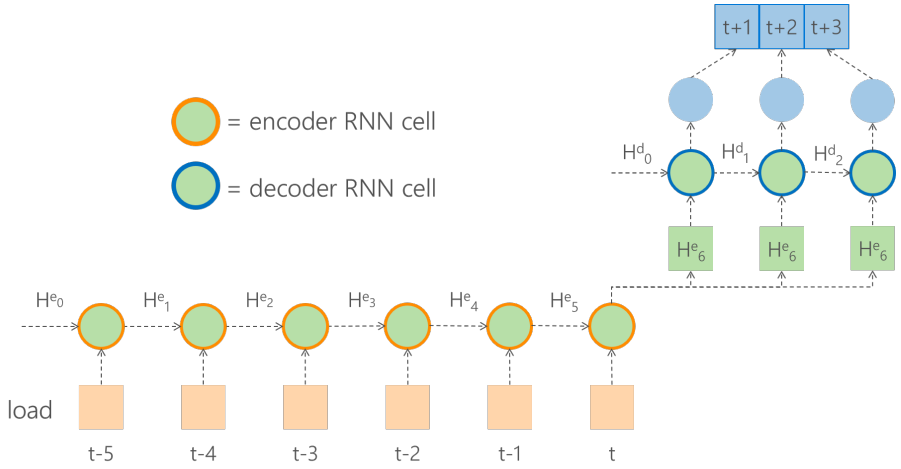


- Simplest to implement
- Fastest to train
- The forecasts are not dynamic

Simple encoder-decoder



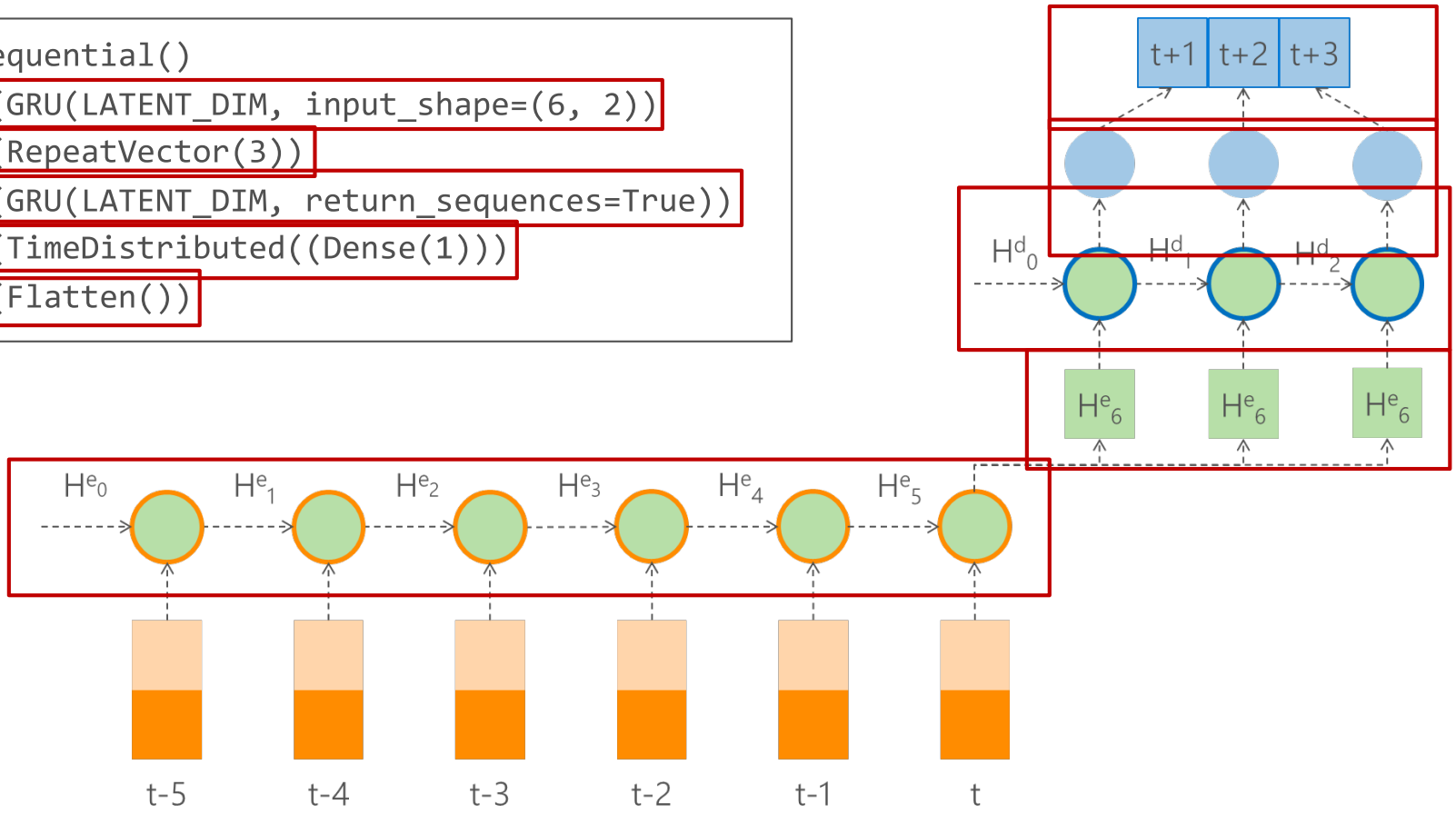
Simple encoder-decoder



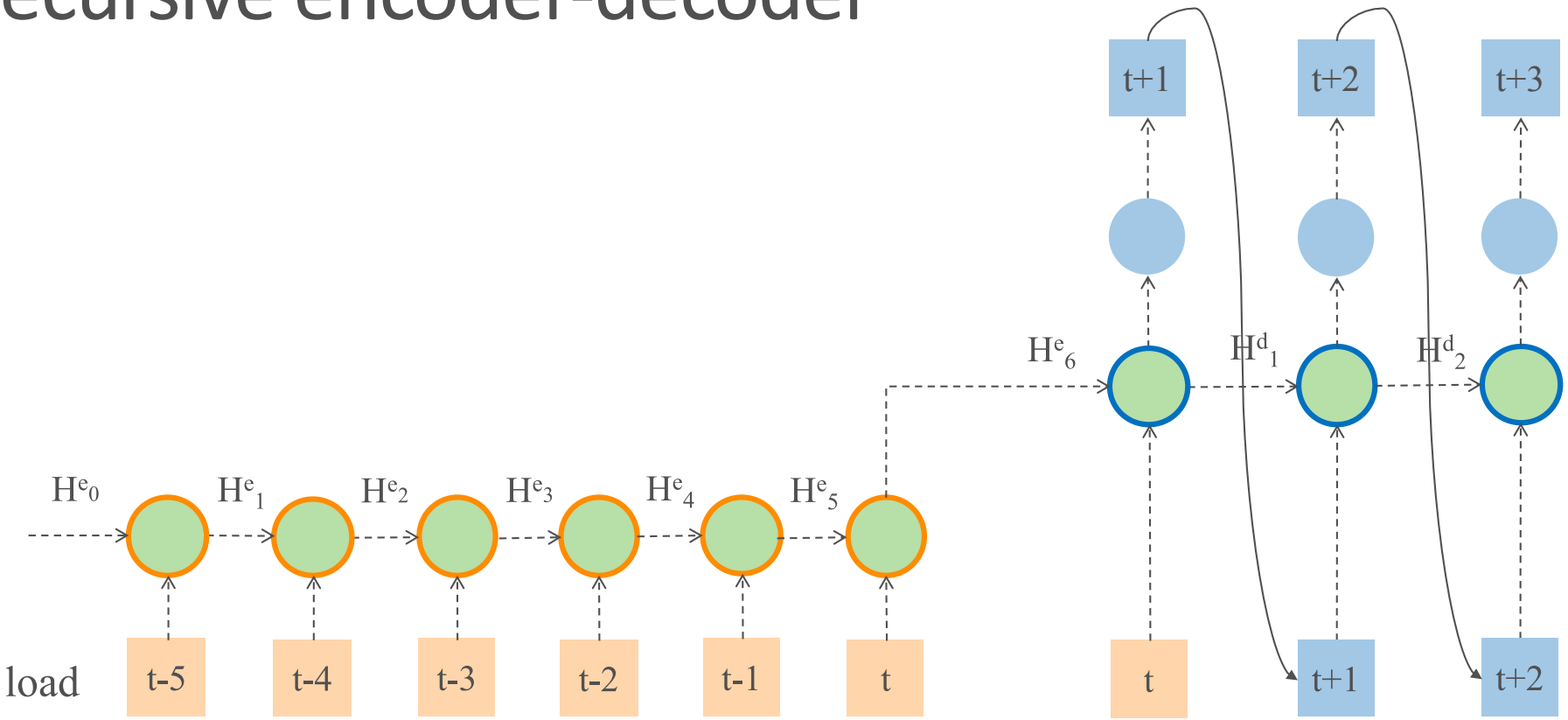
- Fairly simple to implement
- Tries to capture dependencies between forecasted time steps through decoder hidden state
- Slower to train with stacked RNN layers

Simple encoder-decoder

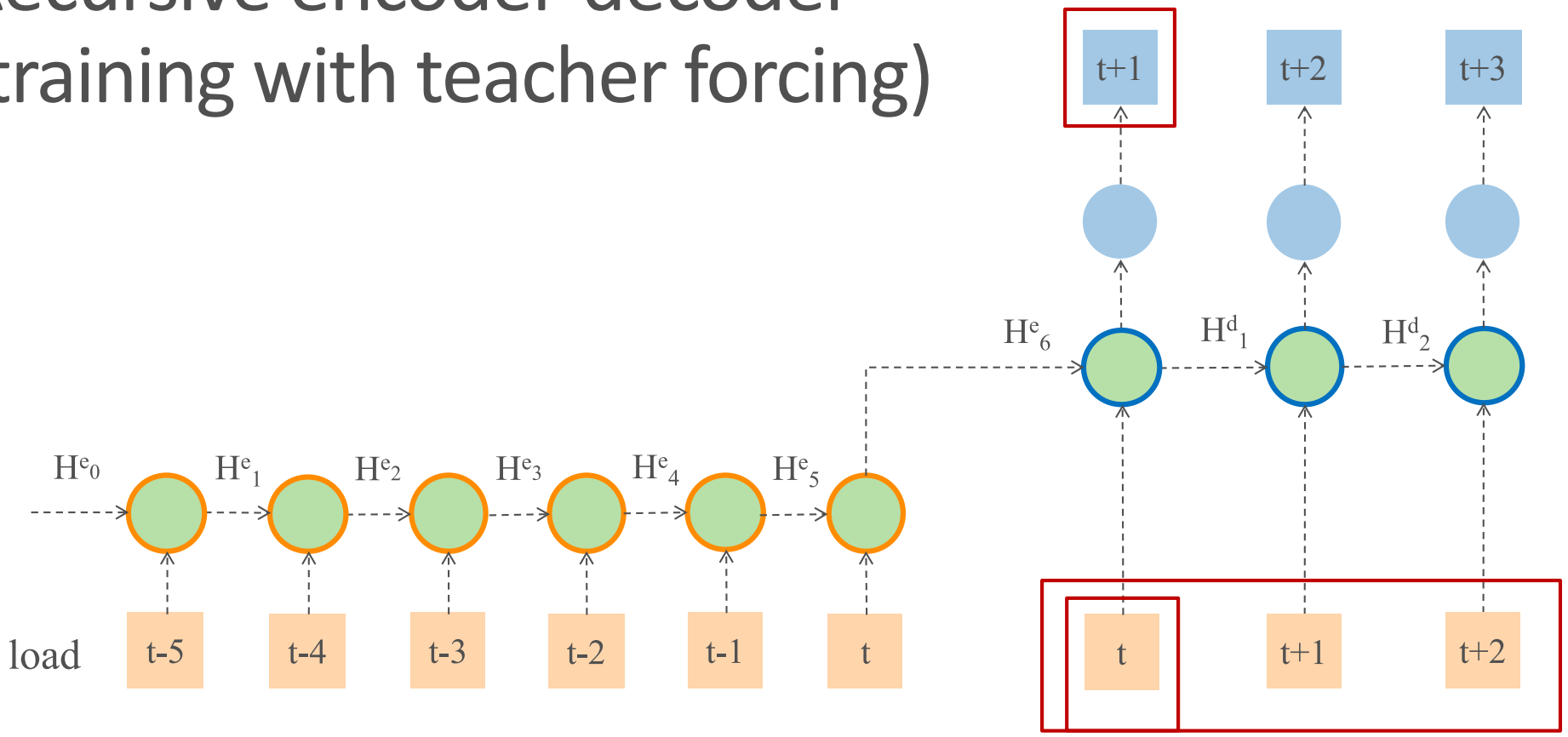
```
model = Sequential()
model.add(GRU(LATENT_DIM, input_shape=(6, 2))
model.add(RepeatVector(3))
model.add(GRU(LATENT_DIM, return_sequences=True))
model.add(TimeDistributed((Dense(1)))
model.add(Flatten())
```



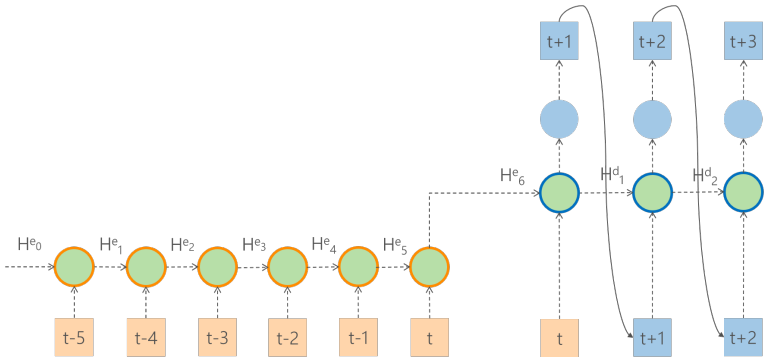
Recursive encoder-decoder



Recursive encoder-decoder (training with teacher forcing)



Recursive encoder-decoder



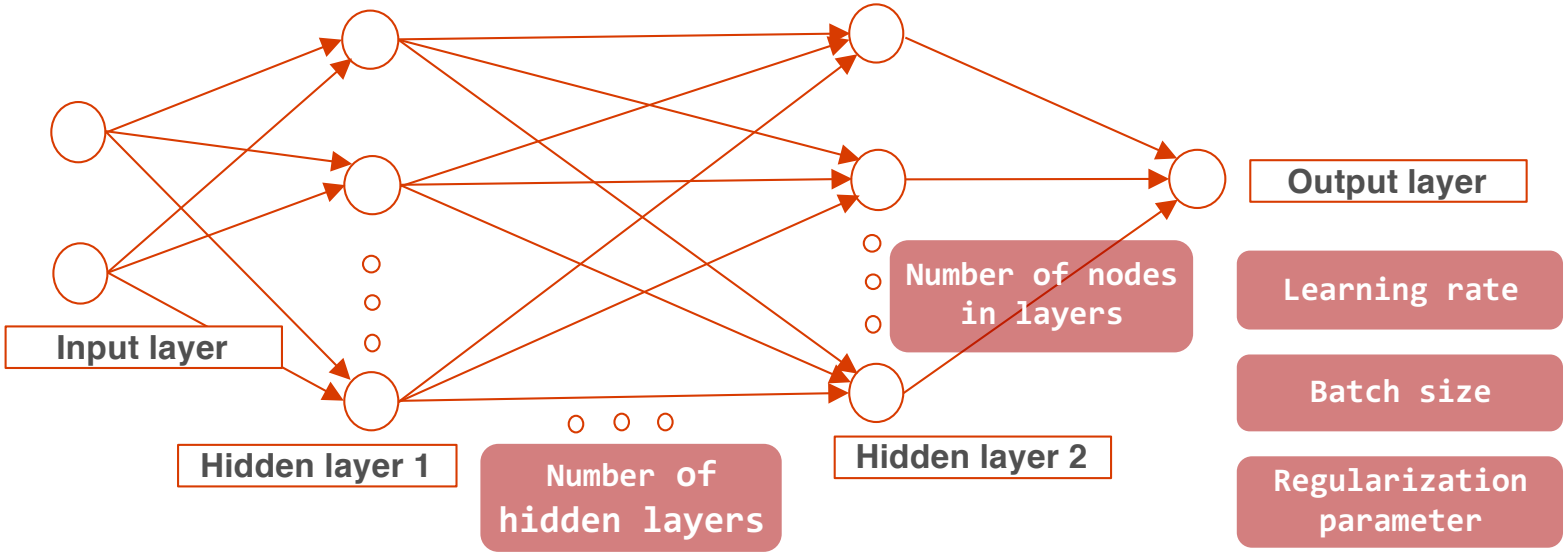
- Tries to capture dependencies between forecasted time steps through recursive decoder
- Variable length output (can generate forecasts of variable horizons)
- More difficult to implement
- Slower to train and slower to generate forecasts
- Error propagation due to recursive decoder

Part III

Hyperparameter tuning

What are hyperparameters?

- Adjustable parameters that govern model training
  - **Architecture**: number of layers, number of cells in each layer, connections
  - **Optimization**: learning rate, mini-batch size, stopping criterion, initialization
  - **Loss function**: weight of regularization term
- Chosen prior to training, stay constant during training, e.g.



Hyper-parameter tuning

- Search across various hyperparameter configurations
- Find the configuration that results in best performance

Challenges

- Huge search space to explore
- Sparsity of good configurations
- Expensive evaluation
- Limited time and resources

Summary

- Basics and traditional forecasting
  - Time series features (trend, seasonality, cycle)
  - Autocorrelation
  - Stationarity
  - Transformations and statistical tests
  - Time series decomposition
  - Time series models (SARIMAX)
- Regression-based forecasting
  - Forecasting strategies (recursive, direct, multi-output)
  - Time series cross-validation
- (Deep) neural network architectures for forecasting
  - Convolutional neural networks
  - Reccurent neural networks (RNN, LSTM, GRU)
  - One-step and multi-step ahead forecasting

Acknowledgments

These slides are heavily based on the tutorial **"Deep Learning For Time Series Forecasting"** by Yijing Chen, Angus Taylor, Vanja Paunic, Dmitry Pechyoni at Microsoft. We would like to thank Yijing Chen for sharing the materials with us.