



Deep Learning for Time Series Forecasting

Yijing Chen, Angus Taylor,
Vanja Paunic, Dmitry Pechyoni

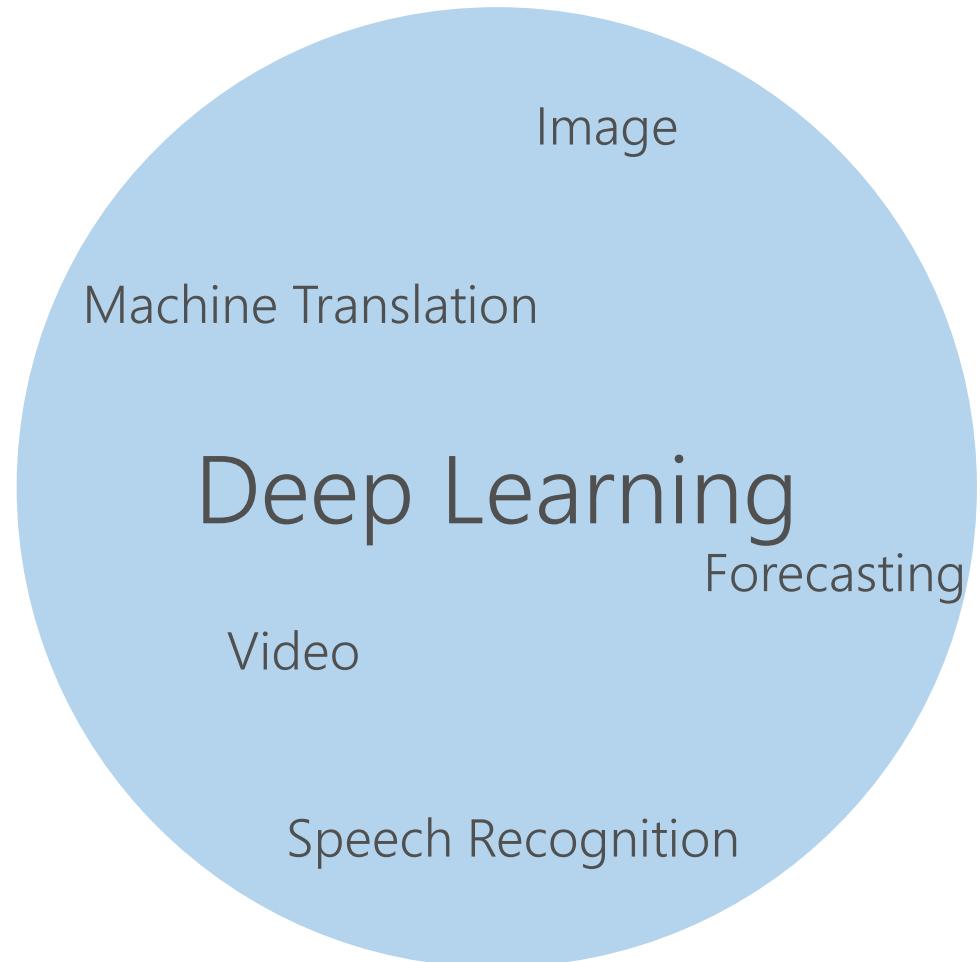
Complete the pre-requisites: aka.ms/dlts/pr



Tutorial Introduction

Tutorial Introduction

This tutorial is about the intersection of DL and Time Series



Tutorial Goals

- Understand the basics of recurrent neural networks (RNN) and convolutional neural networks (CNN) for time series forecasting
- Know how to use Keras to build CNN and RNN model for time series forecasting
- Know a number of techniques and tricks that are important for building successful deep-learning-based time series forecasting models

Target Audience

- Intermediate level
- Know the basics of Python
- Know the basics of neural networks
- Know the basic concepts of machine learning and have some experience of building machine learning models

Learning Path

Agenda	Time (mins)
Tutorial Introduction + Pre-requisite Setup	20
Knowledge Recap	10
Introduction to convolutional neural networks (CNN)	30
Introduction to recurrent neural networks (RNN)	25
Break	10
Encoder-decoder RNN model	25
Hybrid models for time series forecasting	35
Hyper-parameter tuning	15
Conclusion	5
Q&A	5

- Hands-on lab and quizzes
- Q&A during break or in the end

Hands-on Lab

- Form a team of 2-3
- Help your teammates
- We are here to help as well, just raise your hand
- Introduce yourself to your neighbor
- Ice breaker question: what's your favorite ice-cream flavor?

Pre-requisite Setup

aka.ms/dlts/pr



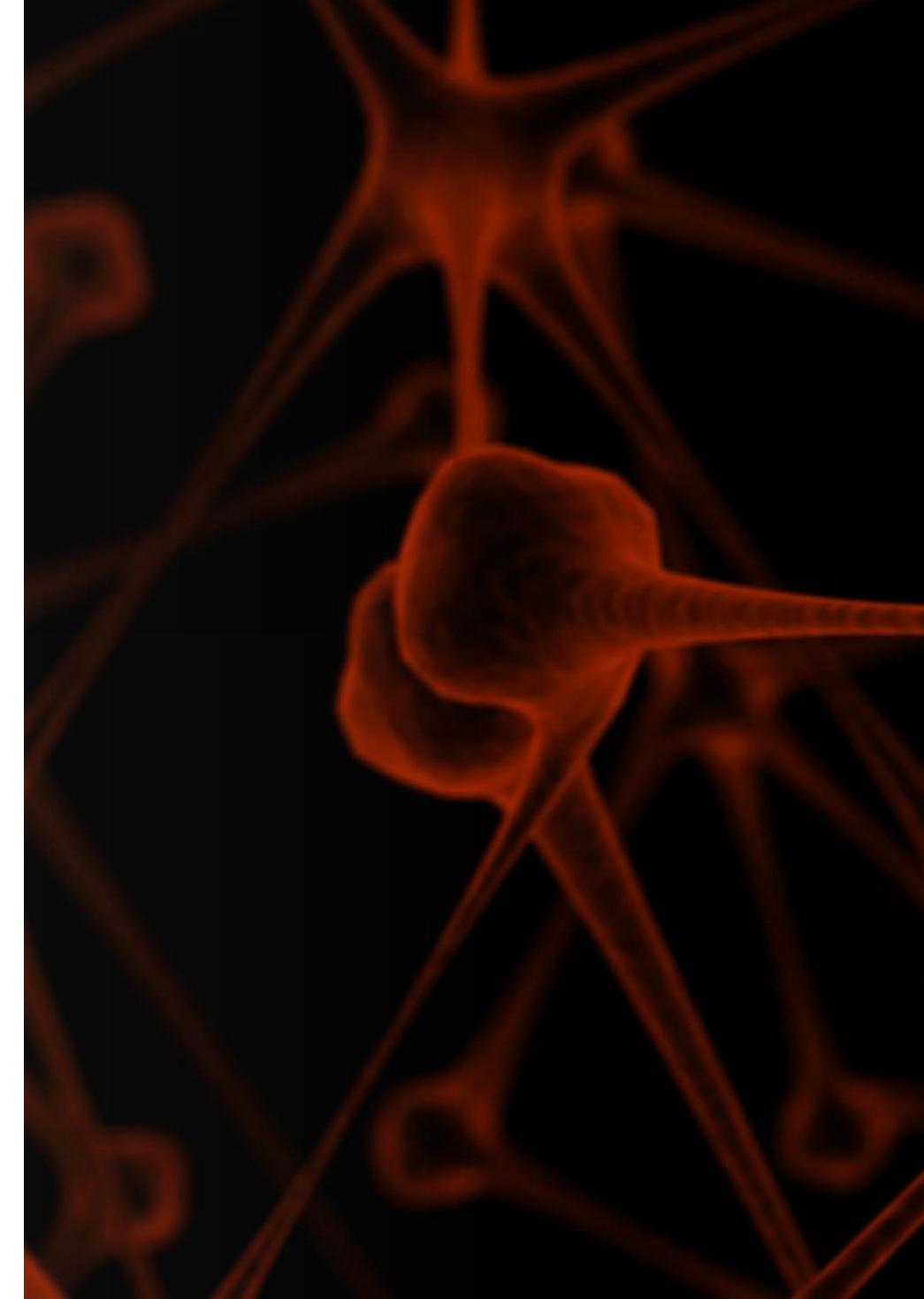
Why Keras?

- 👍 High-level API for deep learning
- 👍 Simplified syntax for common DL tasks
- 👍 Operates over efficient backends (e.g. TensorFlow, CNTK)
- 👎 Less control over low-level operations
- 👎 Sometimes not as efficient as other frameworks



Knowledge Recap

Time Series Forecasting



Time Series & Time Series Forecasting

- *Time series* are observations taken sequentially in time
- *Time series forecasting* predicts future 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	?

The table illustrates a time series dataset. The first 12 rows represent historical data, grouped under a bracket labeled "History". The last three rows represent future data, grouped under a bracket labeled "Future". The "Observation" column shows values increasing from 60 to 72, then dropping to question marks for the future dates.

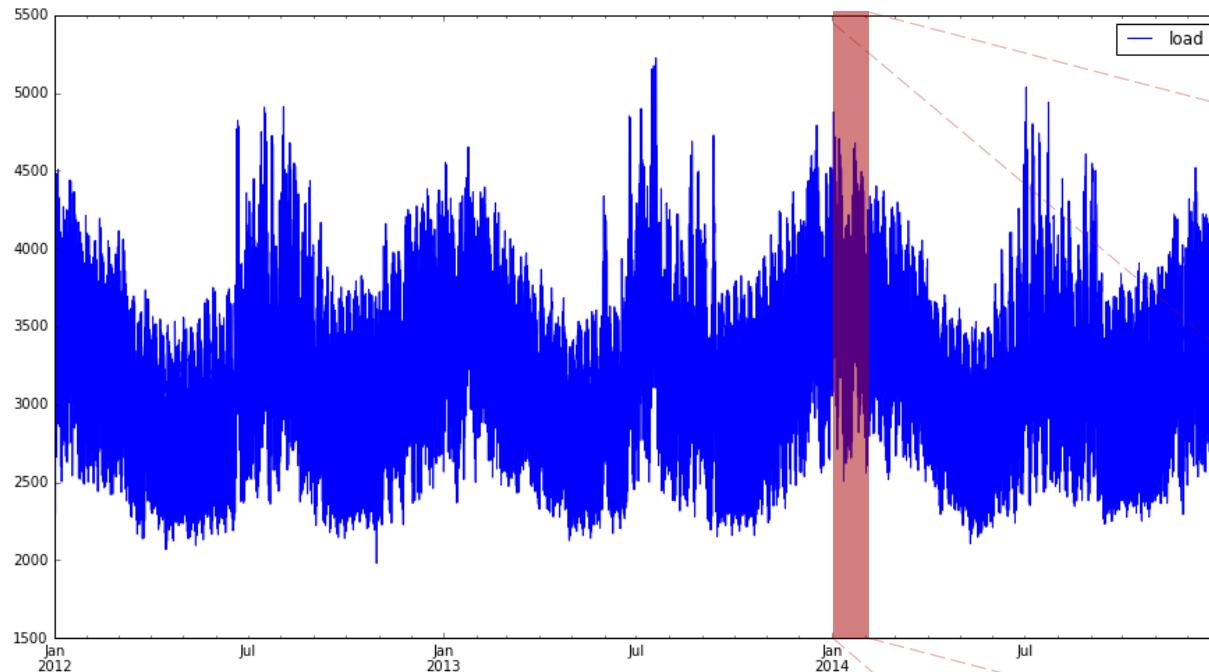
Questions to ask before building forecast model

- Can it be forecast?
 - How well we understand the factors that contribute to it?
 - How much data are available and are we able to gather it?
 - How far in future (*horizon*) we want to forecast?
 - At what temporal frequency are forecasts required?
 - Can forecasts be updated frequently over time or must they be made once and remain static?
-
-
- What technique/model should I use?

Scenario: energy load forecasting

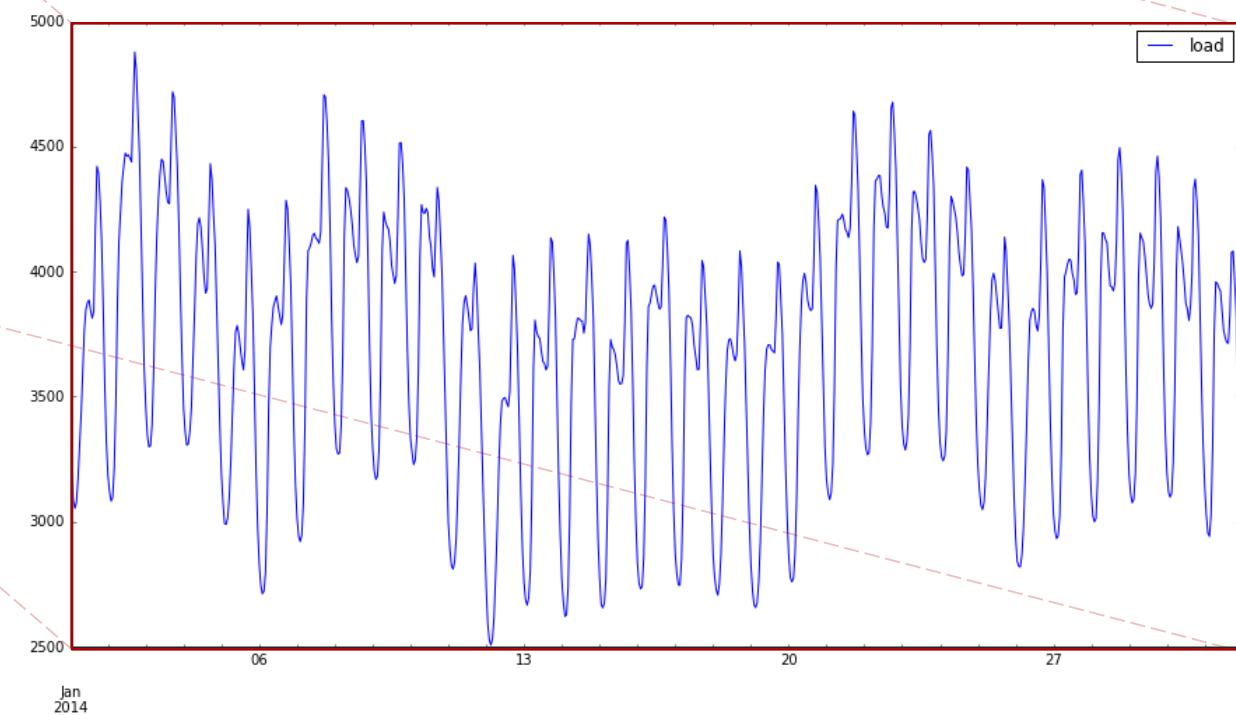
- Energy grid operators keep the supply and demand of electricity on power grids in balance
- They need short term demand forecasts to ensure the reliability of the supply, and long term forecasts for planning
- Data in this example was used in the Global Energy Forecasting Competition in 2014 - GEFCom2014

New England ISO data



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.

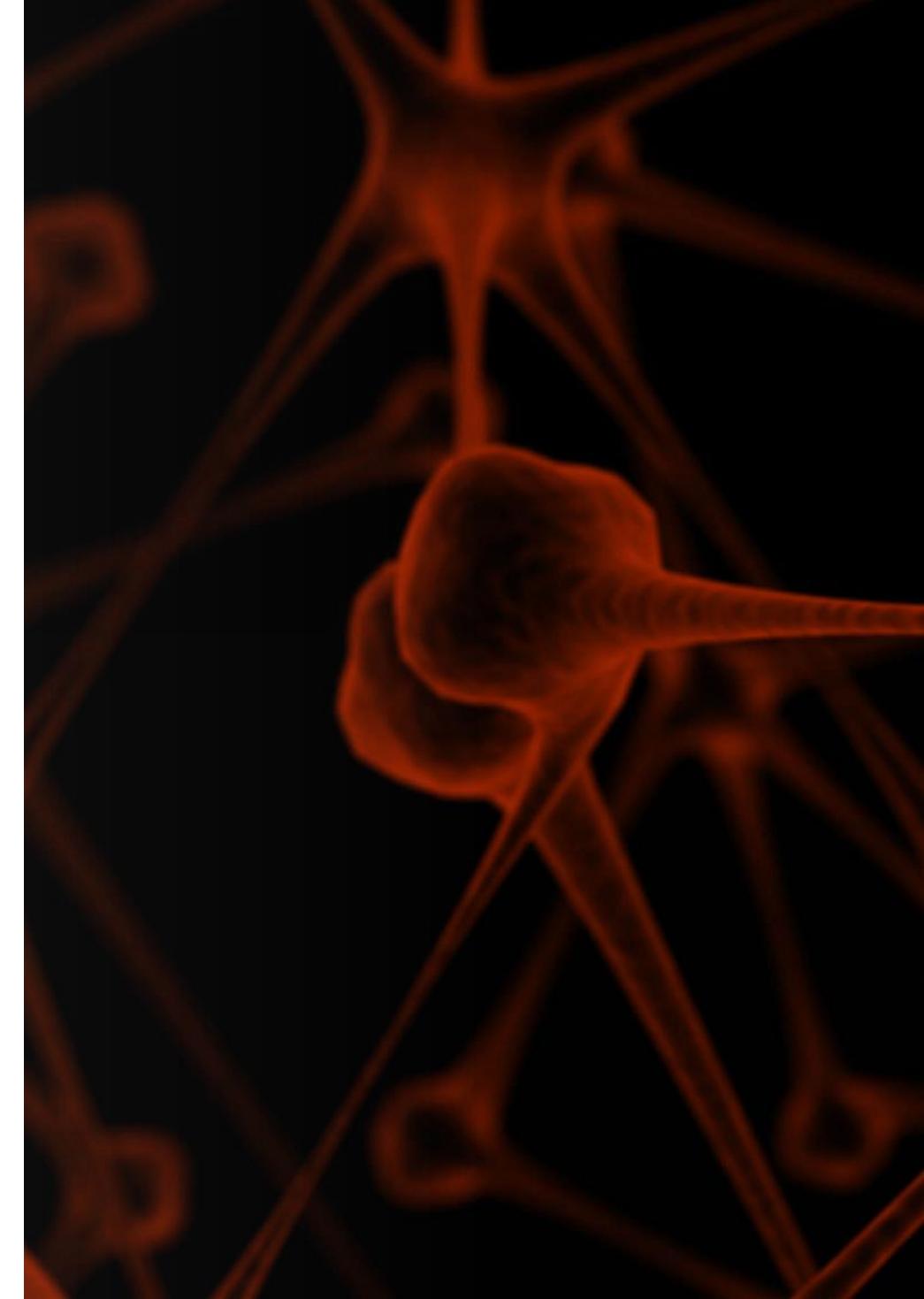
- 26,000 hourly load values
- Annual, weekly and daily seasonality



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](#))
- Non-parametric, greatly simplifies learning
- Any exogenous feature can be easily injected into the model
- RNN is a natural extension of well-studied ARIMA models, but much more flexible and expressive

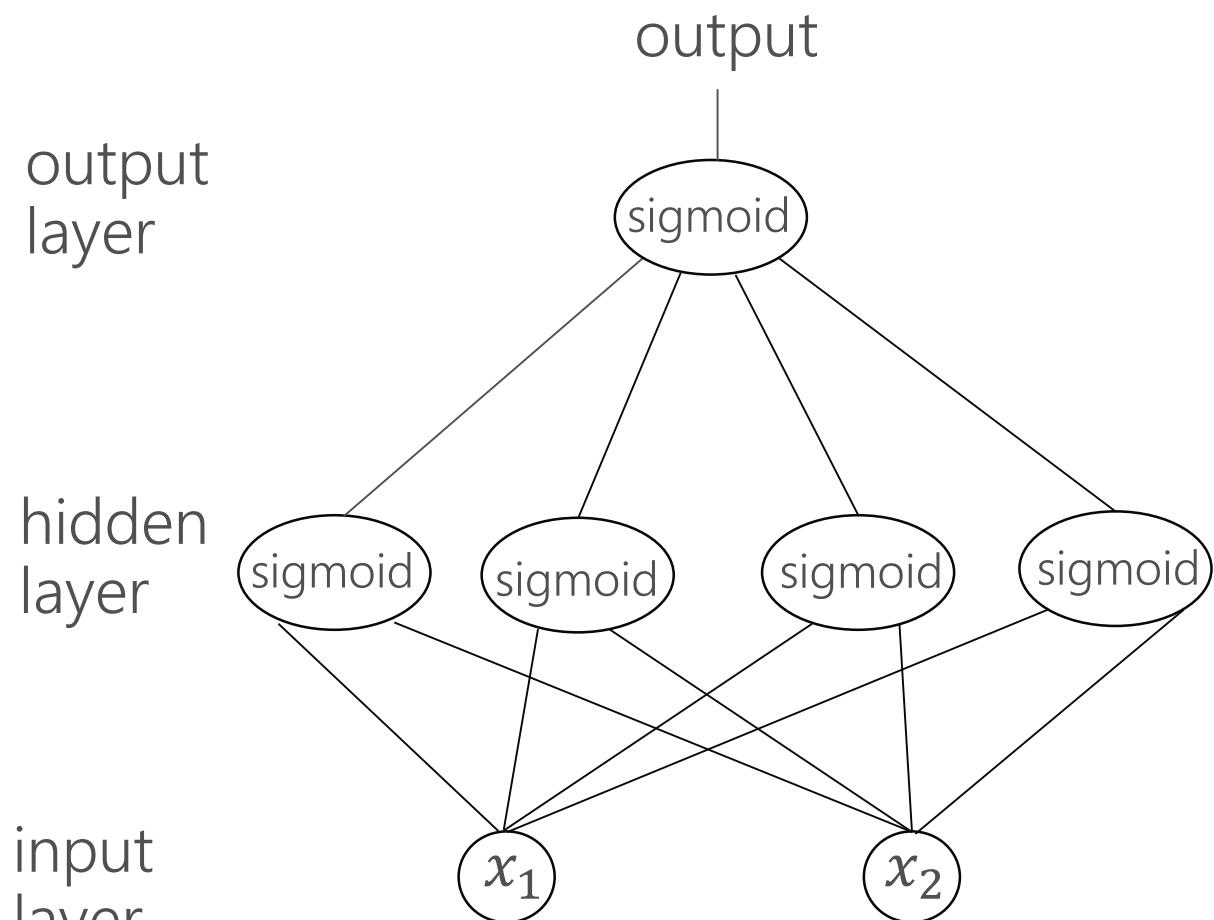
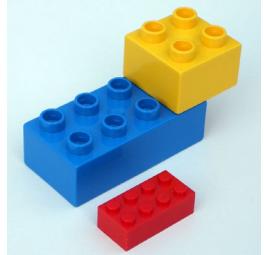
Feedforward Neural Networks Recap



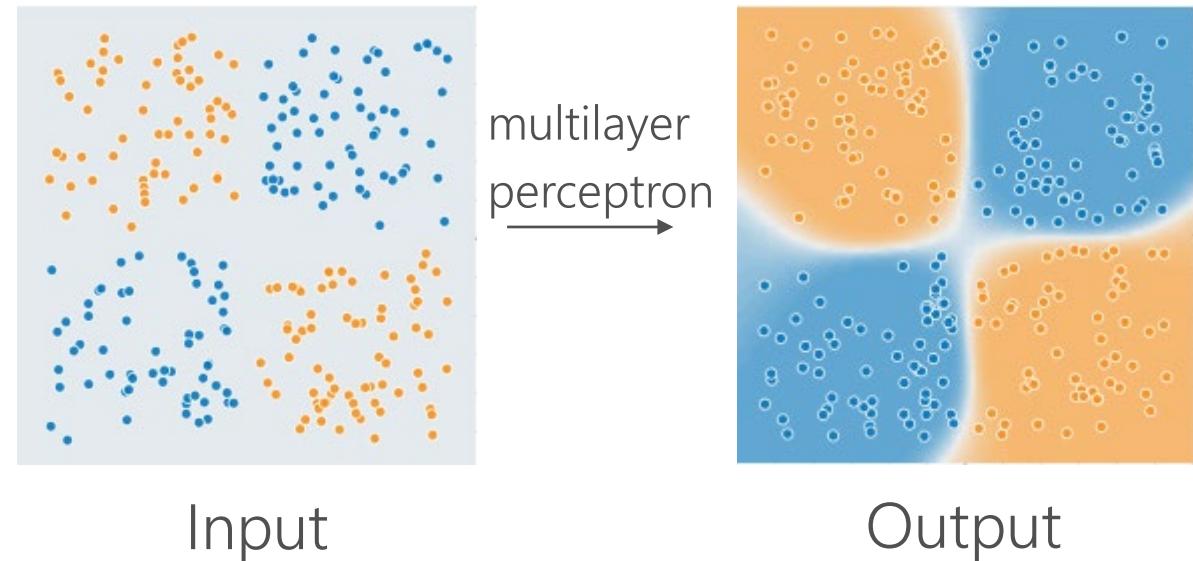
Agenda

- Architecture of Feed-forward Neural Network
- Feed-forward Neural Network Training
 - Loss function
 - Gradient descent (stochastic, batch, mini-batch gradient descent)
 - Backpropagation

Multilayer Perceptron



$$\hat{y} = \begin{cases} 1 & \text{if output} > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$



also known as Feed-Forward Neural Network and Deep Neural Network

Activation Functions

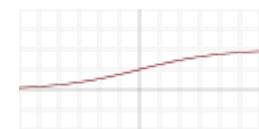
https://en.wikipedia.org/wiki/Activation_function

Applied over $x' = w \cdot x + b$

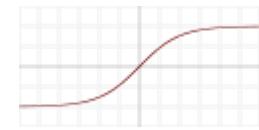
Identity $f(x') = x'$



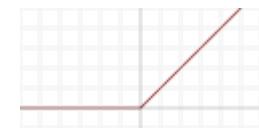
Sigmoid $f(x') = \sigma(x') = \frac{1}{1+e^{-x'}}$



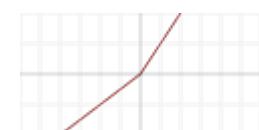
tanh (hyperbolic tangent) $f(x') = \frac{e^{x'} - e^{-x'}}{e^{x'} + e^{-x'}}$



Rectifier linear unit (ReLU) $f(x') = \begin{cases} 0 & \text{for } x' < 0 \\ x' & \text{for } x' \geq 0 \end{cases}$



Parametric rectifier linear unit (PReLU) $f(x') = \begin{cases} \alpha x' & \text{for } x' < 0 \\ x' & \text{for } x' \geq 0 \end{cases}$

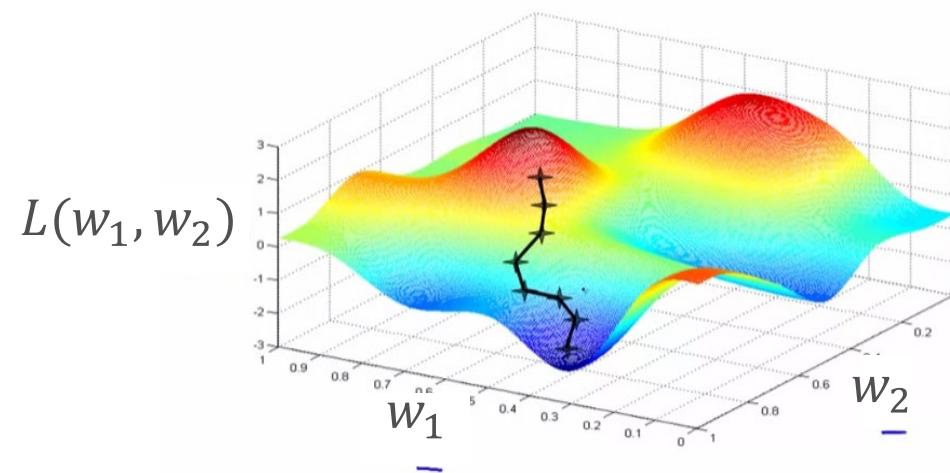


Training: overview

Use training examples to find good weights and biases of the network.

Main components:

- Loss function $L(\text{weights}, \text{biases})$ that measures discrepancy between predicted and true values
- Optimization algorithm that finds weights and biases minimizing the loss function



Picture credit: Andrew Ng, Coursera ML class

Examples of loss function

Commonly used loss functions in time series forecasting:

- Mean-squared-error (MSE): $\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$
- Mean Absolute Percentage Error (MAPE): $\frac{100}{N} \sum_{i=1}^N \left| \frac{y_i - \hat{y}_i}{y_i} \right|$
- Symmetric MAPE (sMAPE): $\frac{100}{N} \sum_{i=1}^N \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2}$

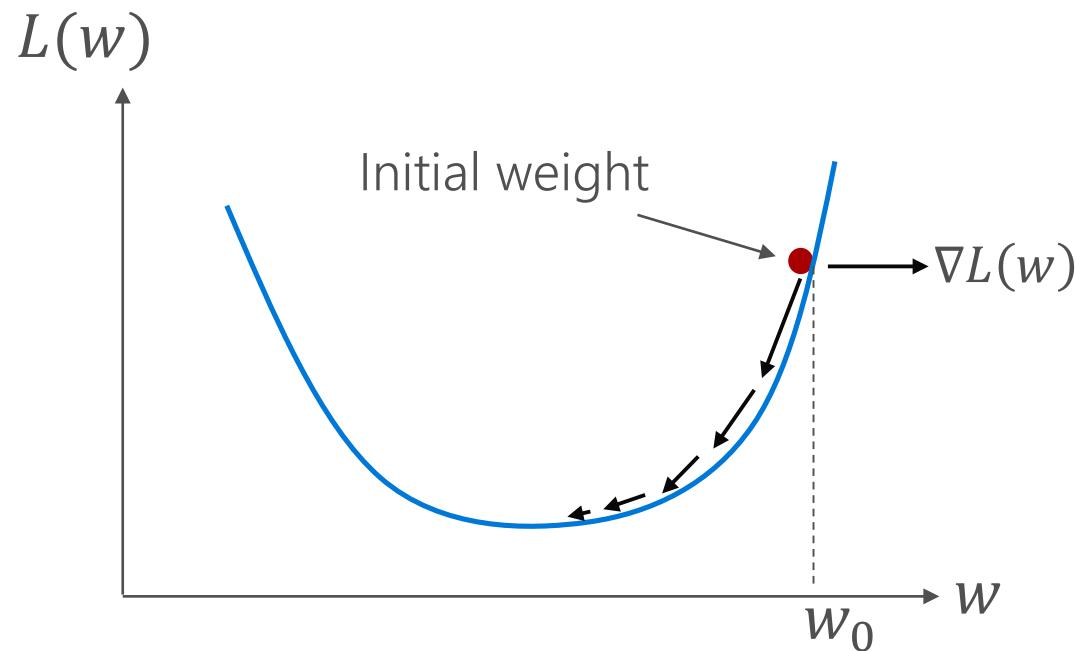
\hat{y}_i is a function of weights w and biases b .

Loss function decomposition $L(w, b) = \frac{1}{N} \sum_{i=1}^N L_i(w, b)$

Optimization algorithm: (Batch) gradient descent

Gradient $\nabla L(w) = \left(\frac{\partial L(w)}{\partial w_1}, \frac{\partial L(w)}{\partial w_2}, \dots, \frac{\partial L(w)}{\partial w_d} \right)$ - direction of the maximal increase of $L(w)$

1D example: $\nabla L(w) = \left(\frac{\partial L(w)}{\partial w} \right)$



Optimization algorithm

Initialization: $w = w_0$

While stopping criterion not met:

$$w = w - \alpha \cdot \nabla L(w)$$

learning rate

Minibatch Stochastic Gradient Descent

Initialization: $w = w_0$

While stopping criterion not met:

 Shuffle examples randomly

 Partition examples into batches of size m

 For minibatch = 1 ... N/m

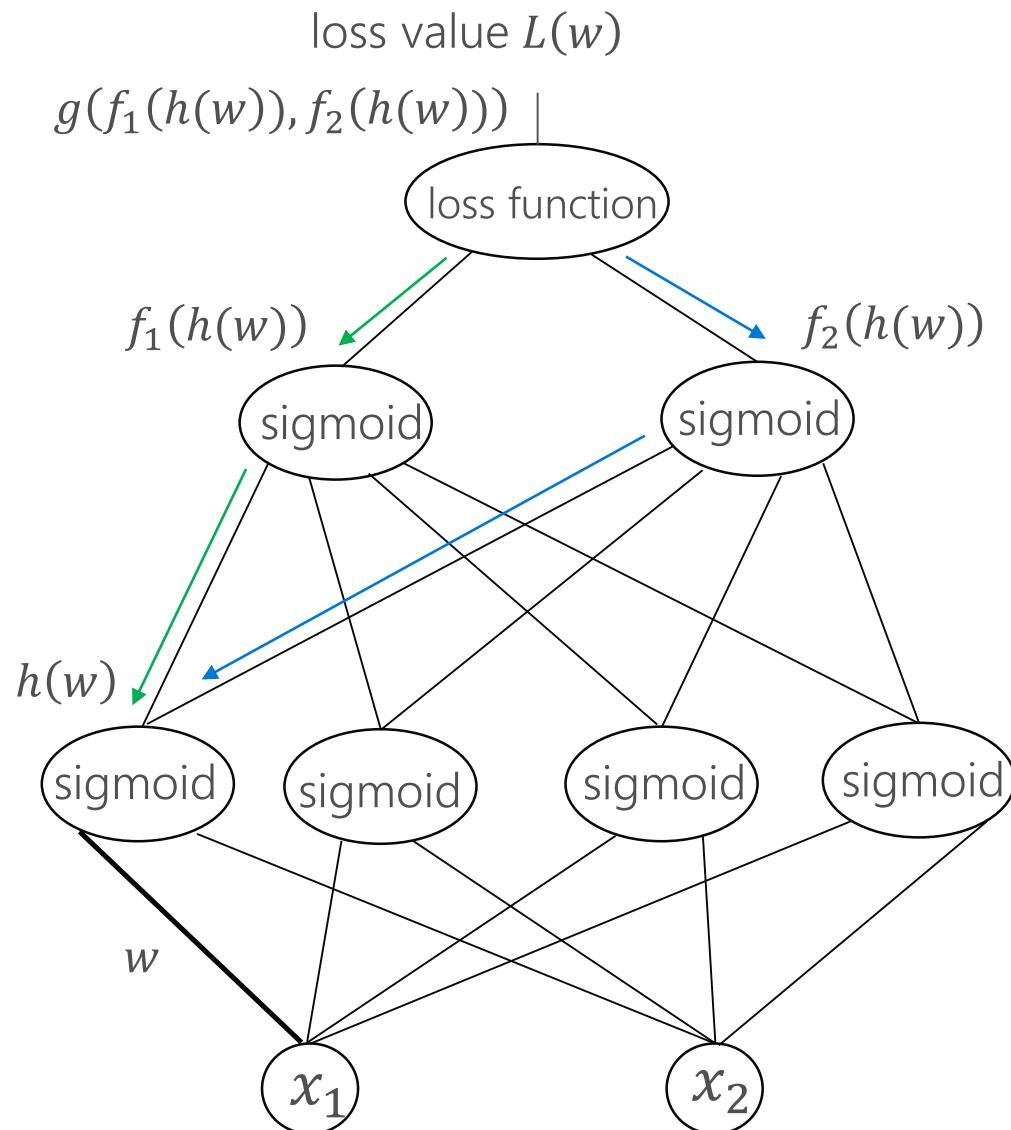
$$w = w - \frac{\alpha}{m} \sum_{i=1}^m \nabla L_i(w)$$

} epoch

gradient from the i -th
example in minibatch

m – mini-batch size (default value 32 in Keras)

Computation of gradients in NN: Backpropagation



Computation of $L'(w)$

Chain rule

$$L(w) = g(f_1(w))$$

$$L'(w) = g'(f_1(h(w))) \cdot f'_1(h(w)) \cdot h'(w)$$

$$L'(w) = g'(f_2(h(w))) \cdot f'_2(h(w)) \cdot h'(w)$$

Total derivative

$$L(w) = g(f_1(w), f_2(w))$$

$$L'(w) = L'(w) + L'(w)$$

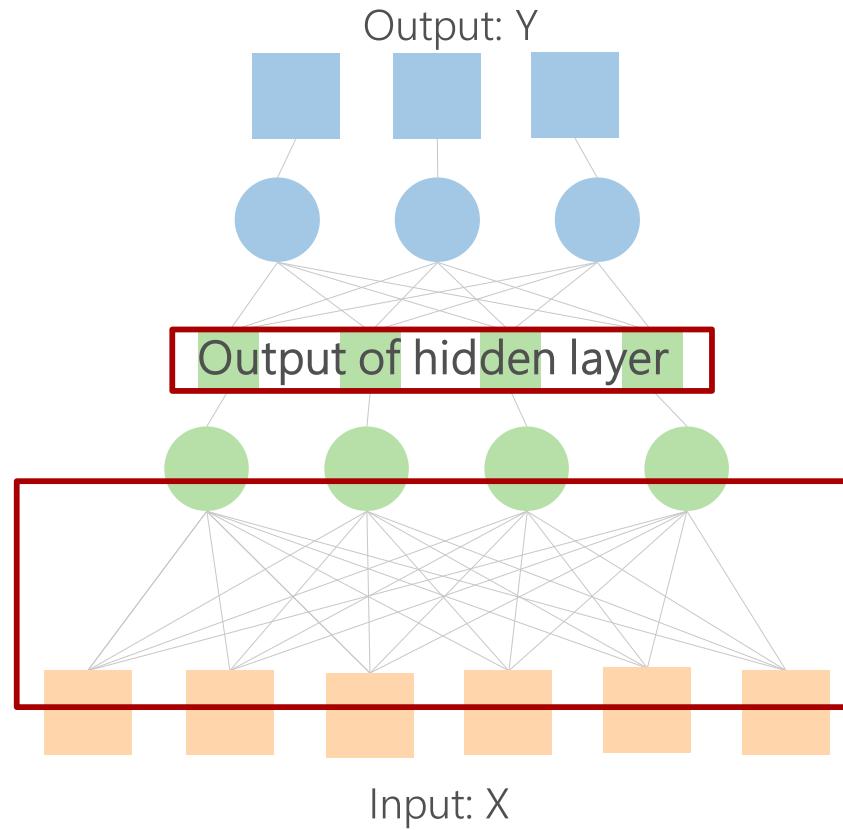
Training tricks

- Early stopping
- Tuning hyperparameters
- Initialization [Tutorial from deeplearning.ai](#)
- Weight regularization [Regularization for deep learning](#)
- Dropout [Dropout: A simple way to prevent neural networks from overfitting](#), [Zoneout: Regularizing RNNs by randomly preserving hidden activations](#)
- Batch normalization [Batch normalization: Accelerating deep network training by reducing internal covariate shift](#), [Recurrent batch normalization](#)
- Learning rate decay [Learning rate schedules and adaptive learning rate methods for deep learning](#)
- Advanced optimization algorithms [An overview of gradient descent optimization algorithms](#)



Introduction to Convolutional Neural Networks

Fully connected vs convolution layer



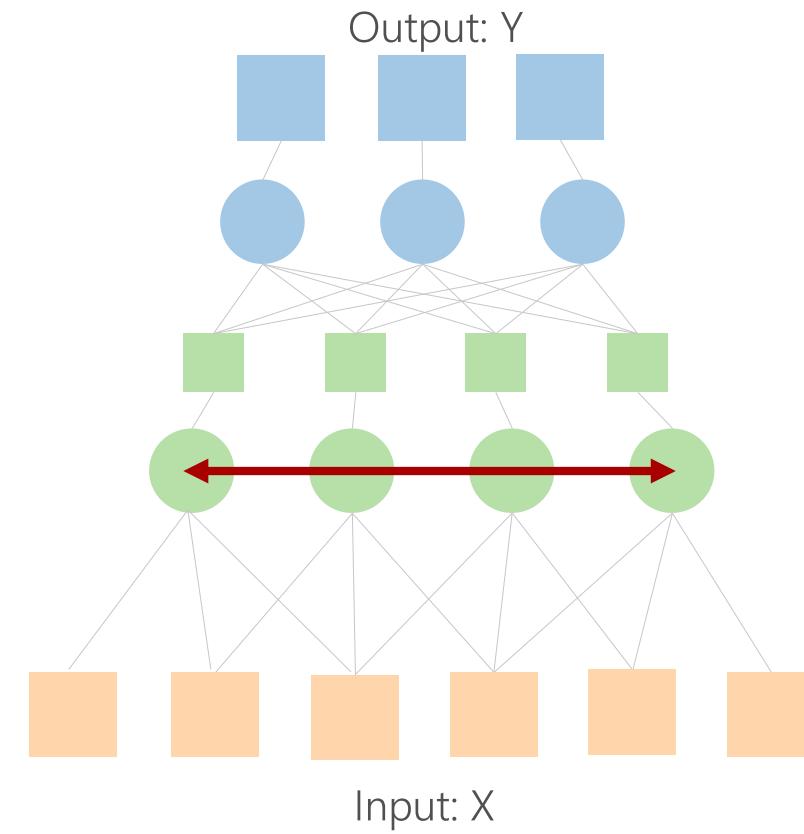
Fully connected:

- 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

$$\begin{array}{c} \boxed{5} \boxed{3} \boxed{2} \boxed{7} \boxed{1} \boxed{6} \\ \times \\ \boxed{1} \boxed{0} \boxed{-1} \\ = \\ \boxed{3} \boxed{-4} \boxed{1} \boxed{1} \end{array}$$

$5 \times 1 + 3 \times 0 + 2 \times -1 = 34$

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

1D Convolutions

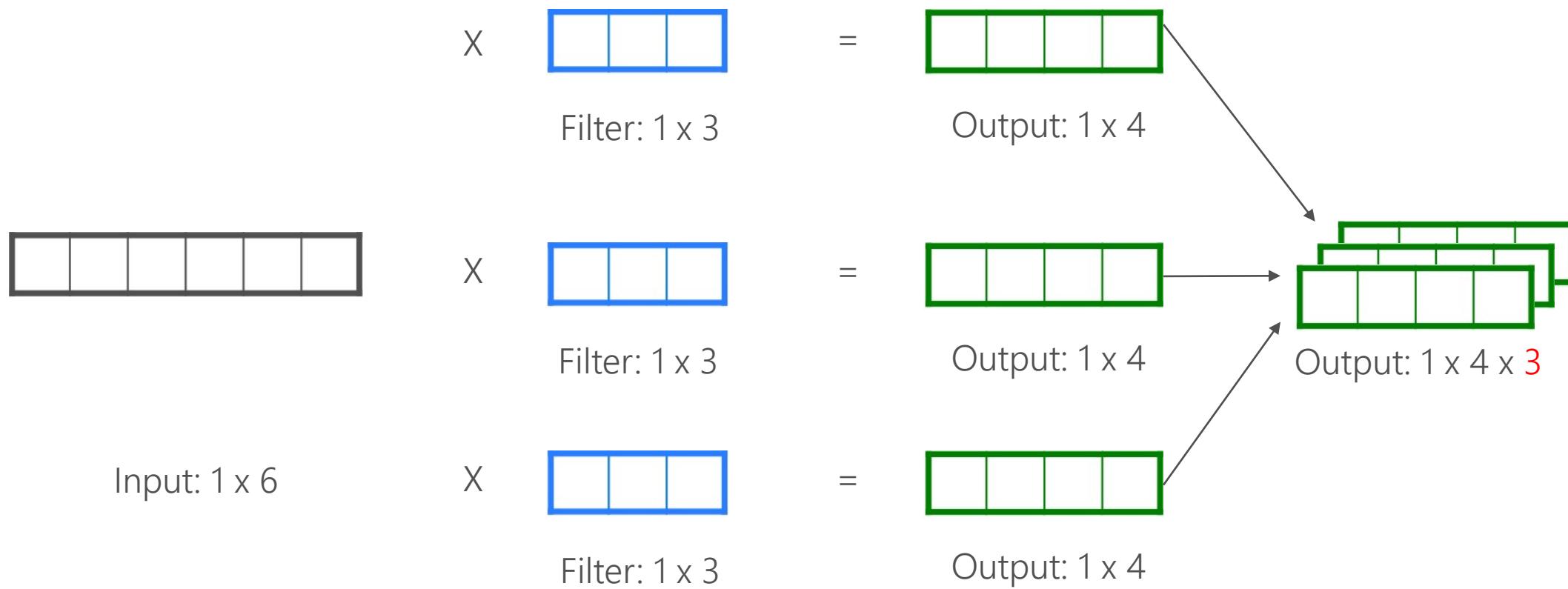
- Filters are trained to detect features in the input sequence
- Features can be detected regardless of where they appear in the input sequence

$$\begin{array}{|c|c|c|c|c|c|} \hline 5 & 3 & 2 & 7 & 1 & 6 \\ \hline \end{array} \quad \times \quad \begin{array}{|c|c|c|} \hline w_1 & w_2 & w_3 \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|c|c|} \hline 3 & 4 & 1 & 1 \\ \hline \end{array}$$

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

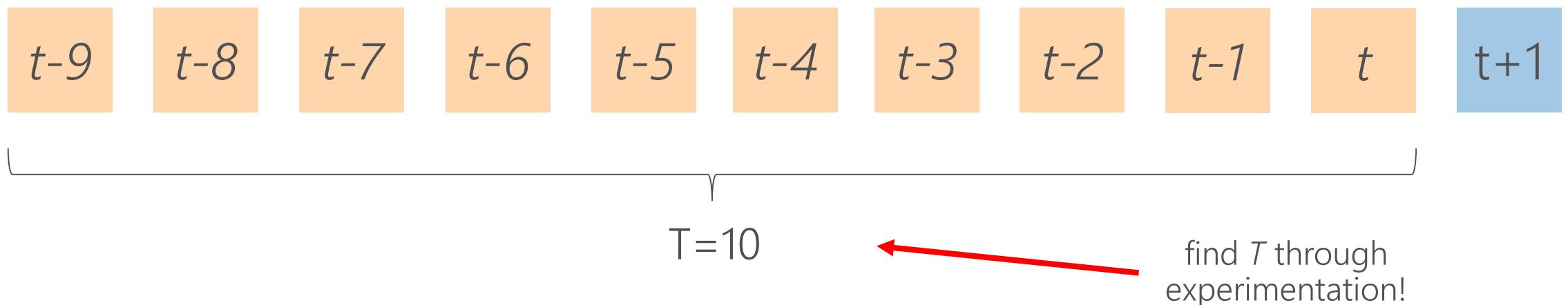
1D Convolutions

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

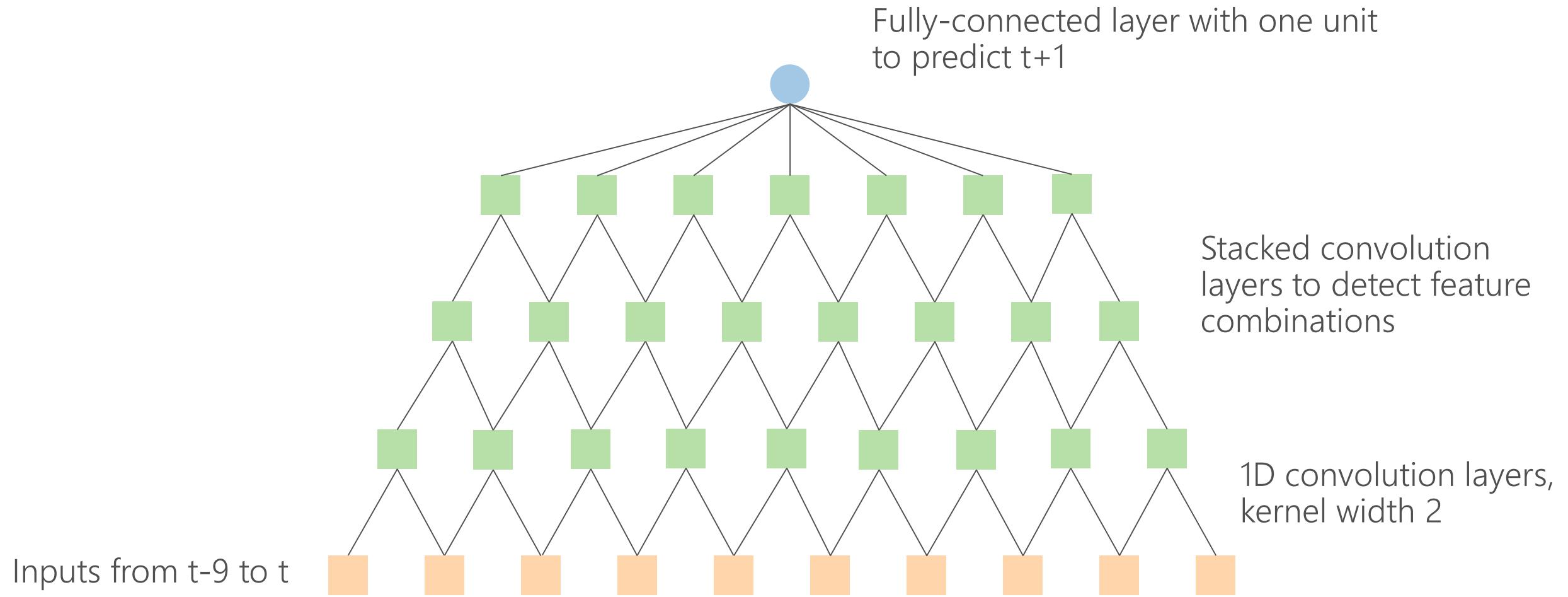


Application to forecasting

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

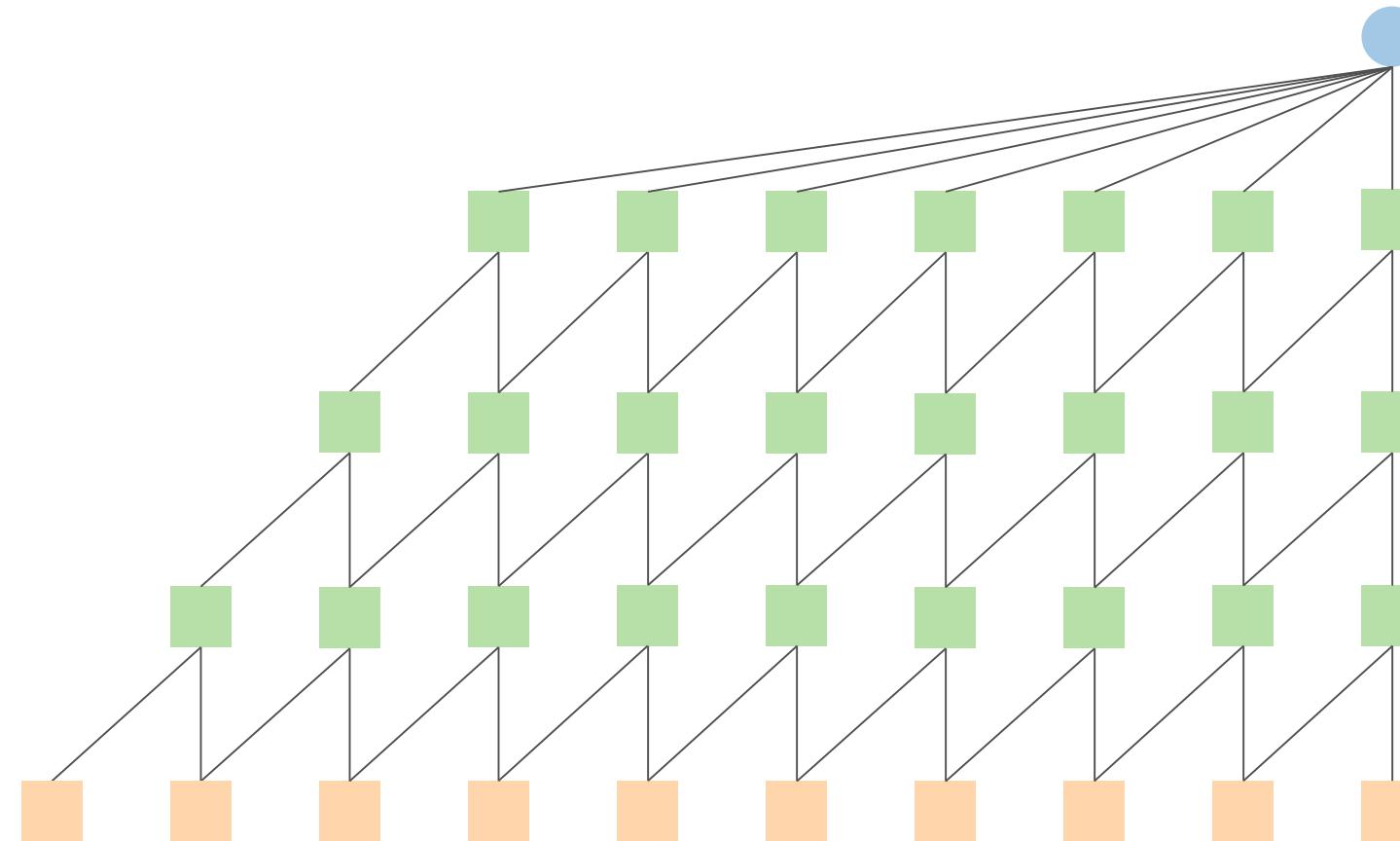


CNN for forecasting



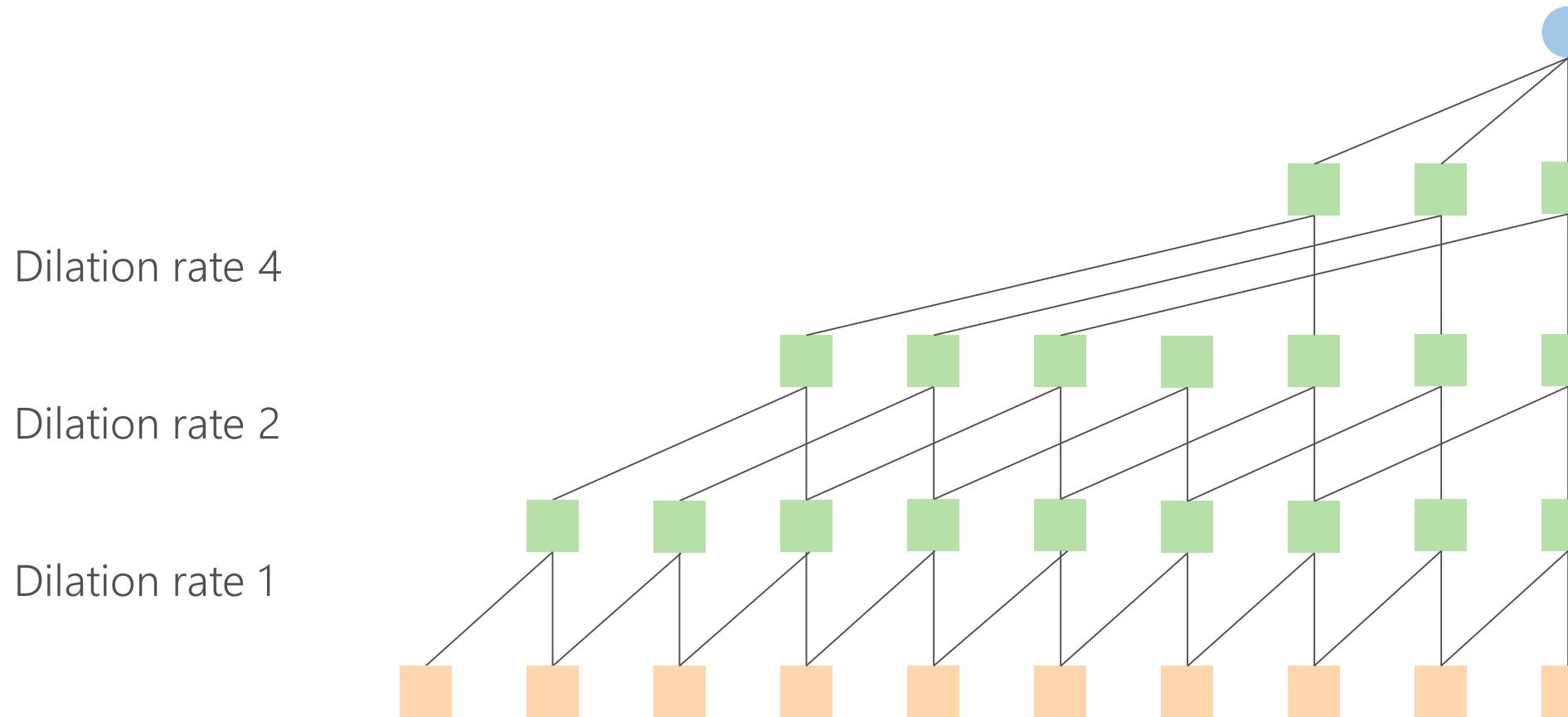
Causal convolutions

- Convolutions are causal if the output at each time step do not depend on future time steps



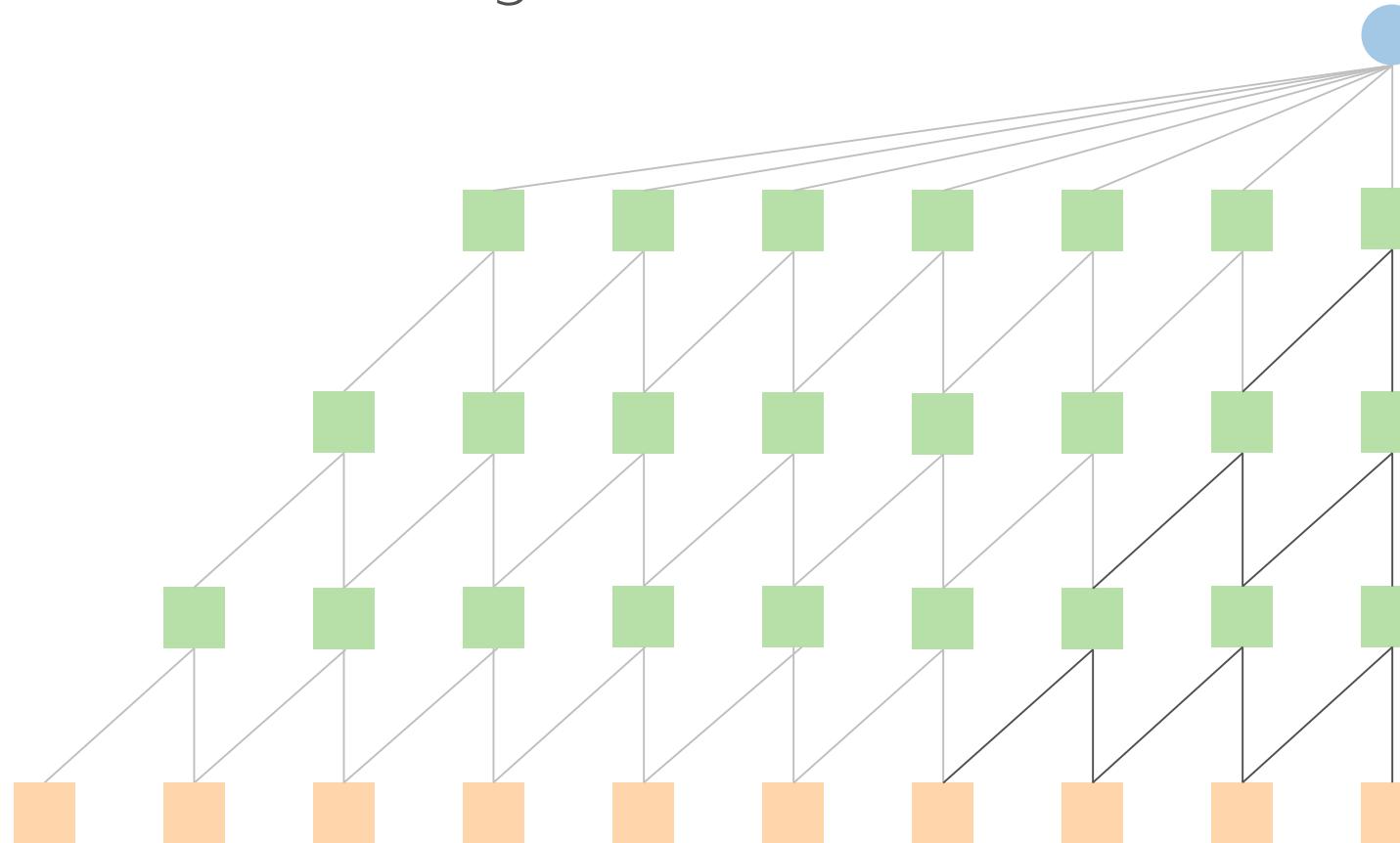
Dilated convolutions

- Skip outputs from previous layers



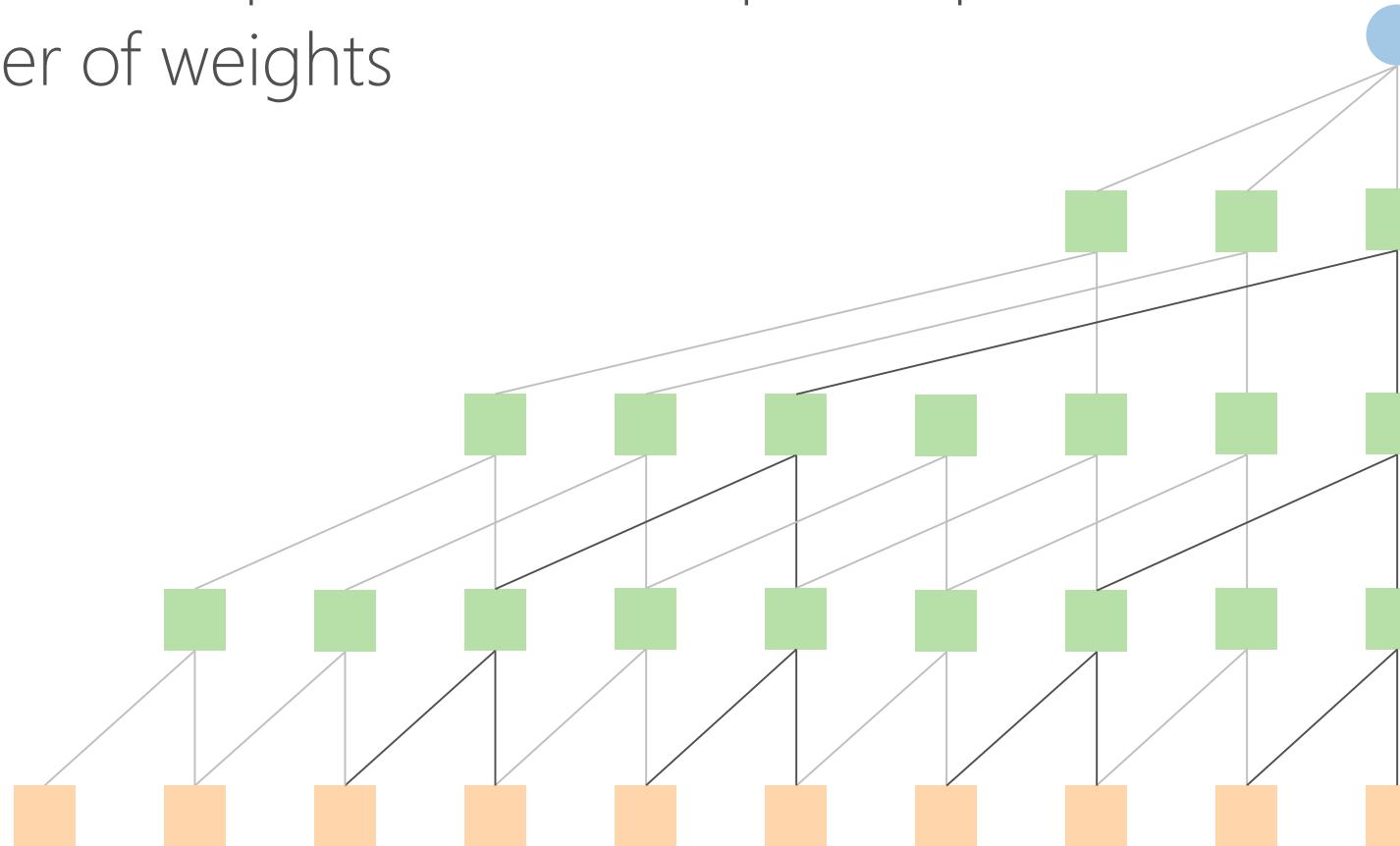
Why dilated convolutions?

- Normal convolutions require more connections
- More connections = more weights to train



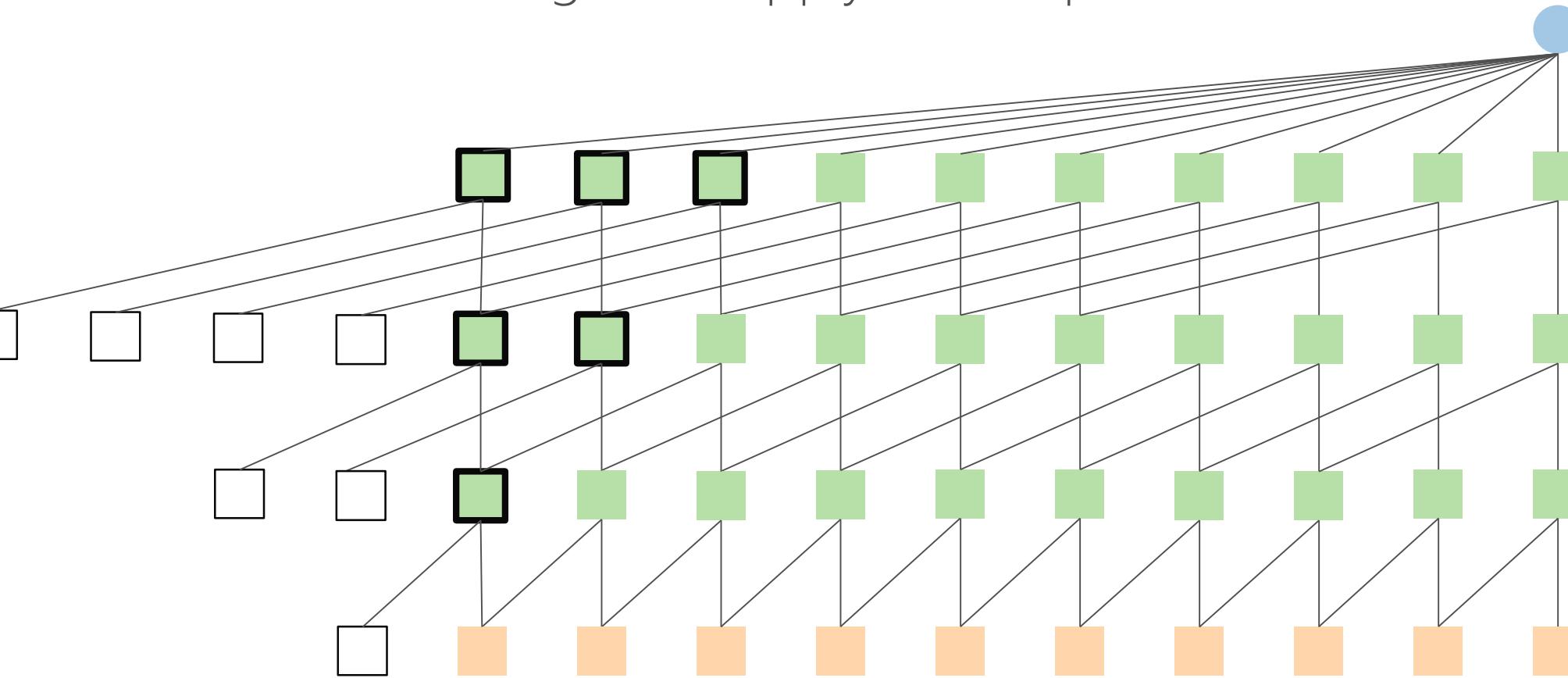
Why dilated convolutions?

- Reach information from more distant values in the time series
- Capture long-term dependencies in input sequence
- Reduce number of weights



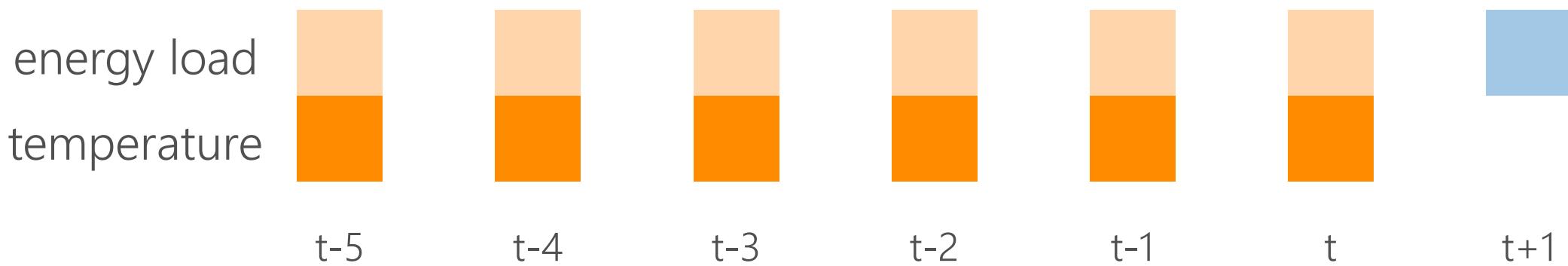
Causal padding

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



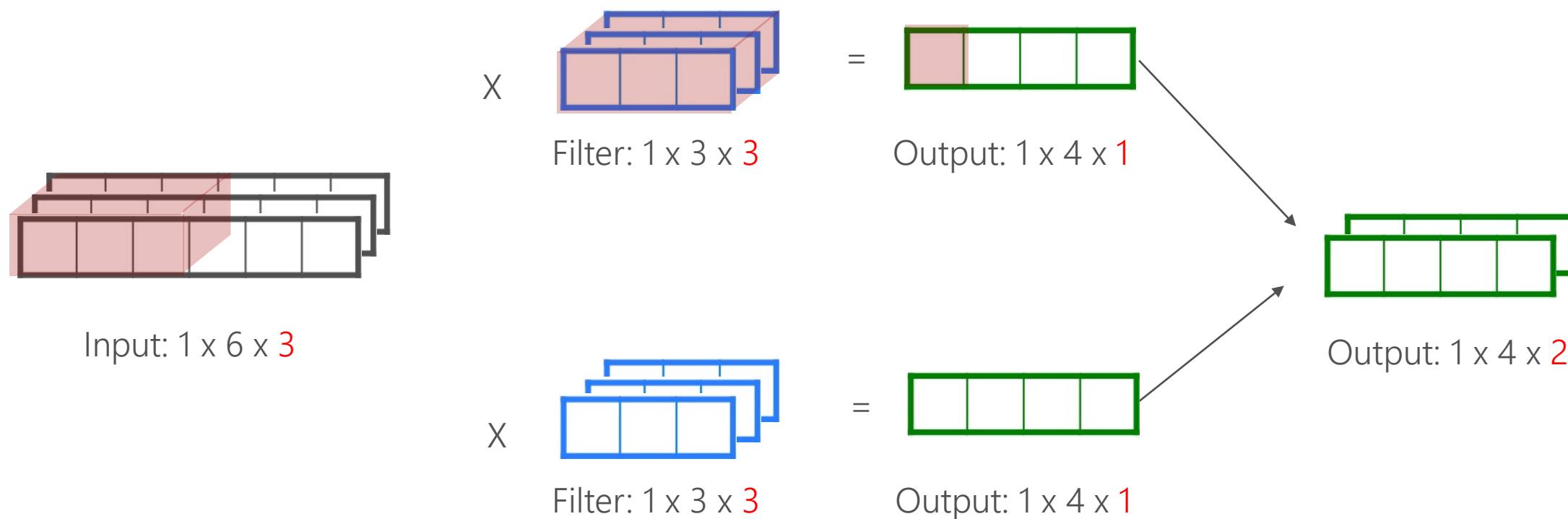
Multivariate time series

- Multiple time series in the input sequence



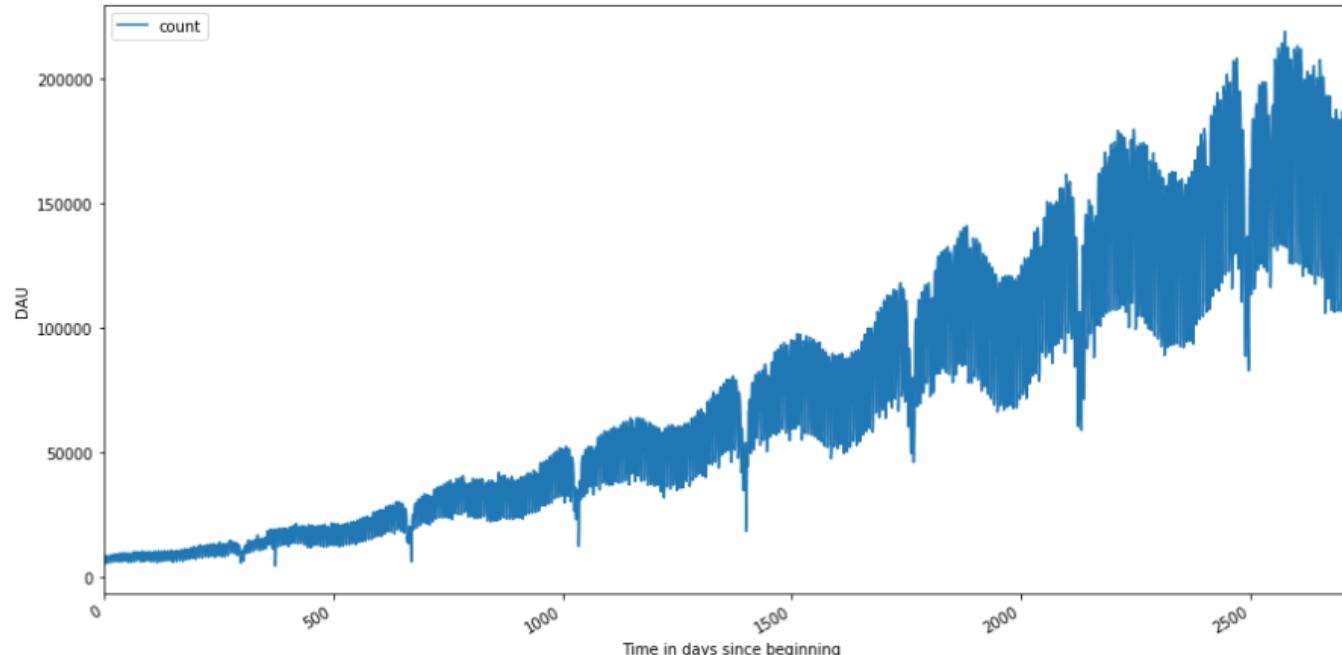
Multivariate time series

- Example: for 3 input time series use 3 channels

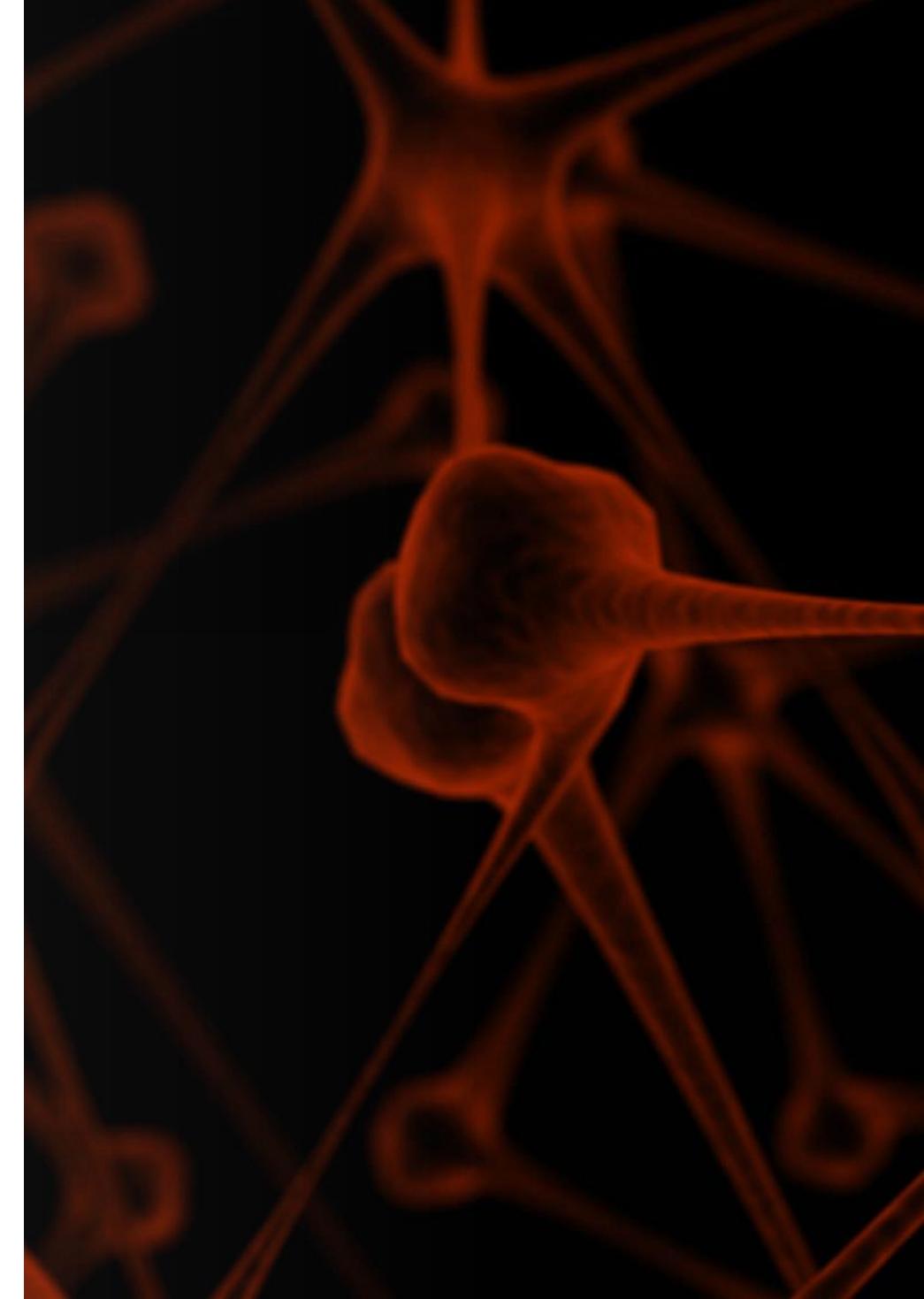


Advanced CNN models for forecasting

- Deep4Cast by Microsoft Research <https://github.com/MSRDL/Deep4Cast>
- Pytorch implementation of Wavenet <https://arxiv.org/pdf/1609.03499.pdf>
- Example forecasting Github daily users



1_CNN_dilated.ipynb notebook



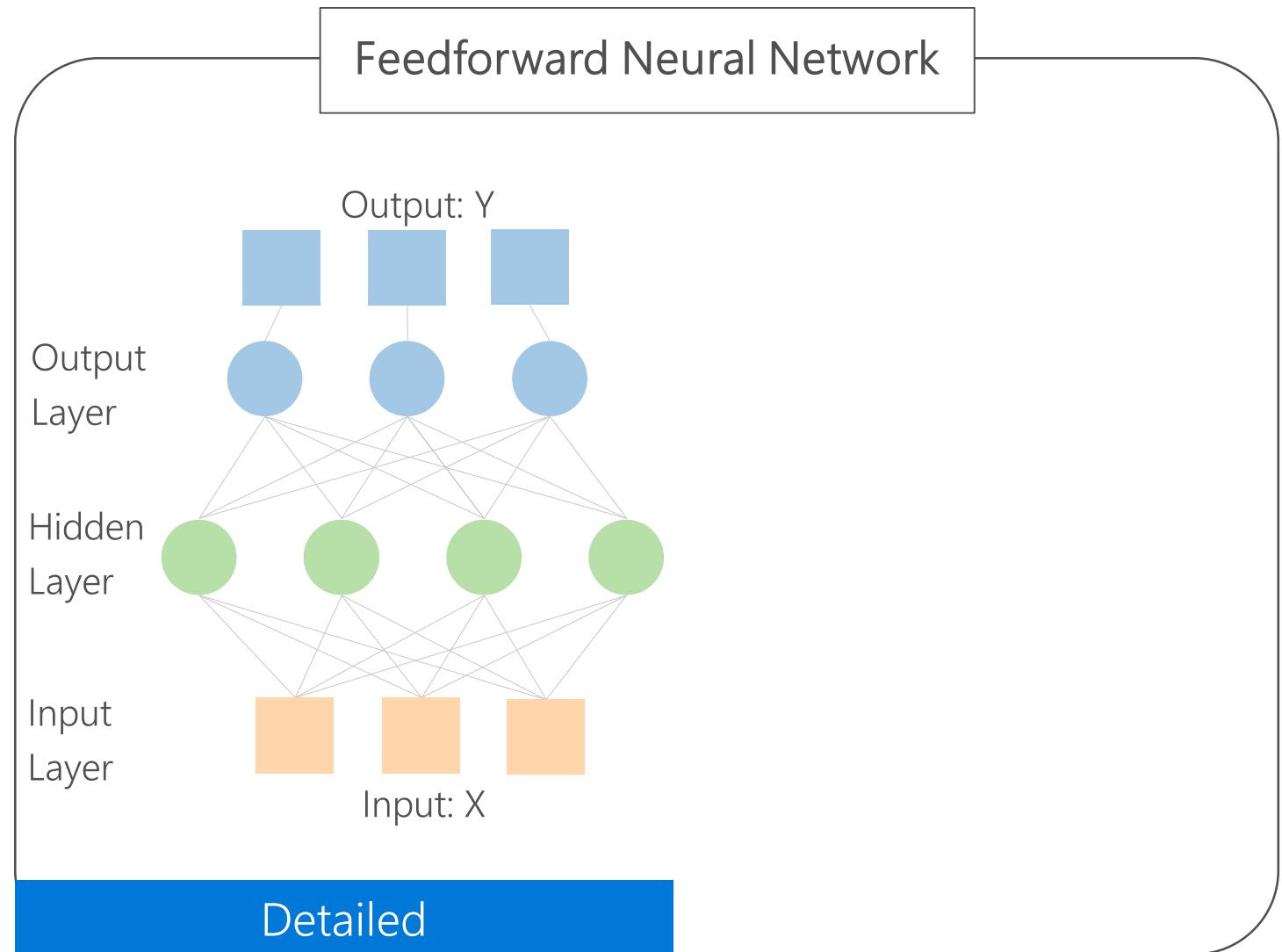


Introduction to Recurrent Neural Networks

Agenda

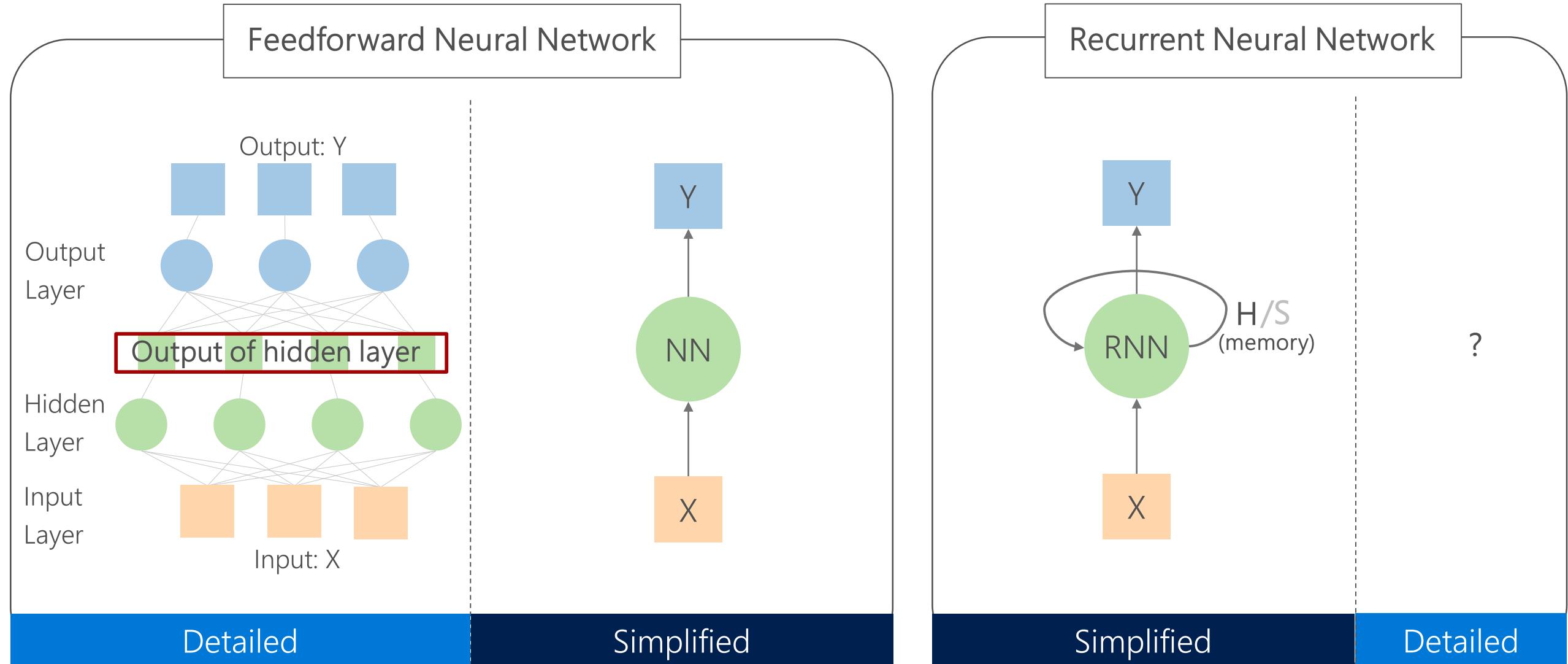
- What are RNNs?
- How RNNs are trained: Backpropagation through time (BPTT)
- Vanilla RNN and its gradient problems
- Other RNN units
 - GRU
 - LSTM
- RNN stacking

What are RNNs?



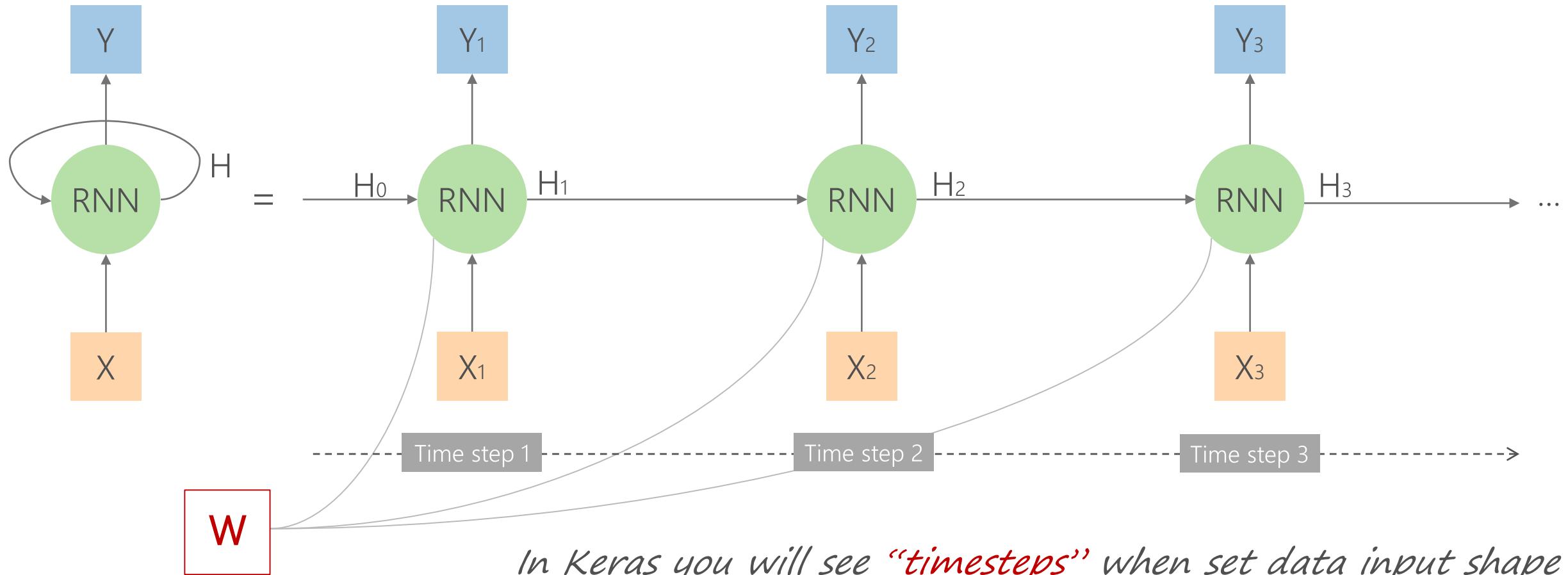
What are RNNs?

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

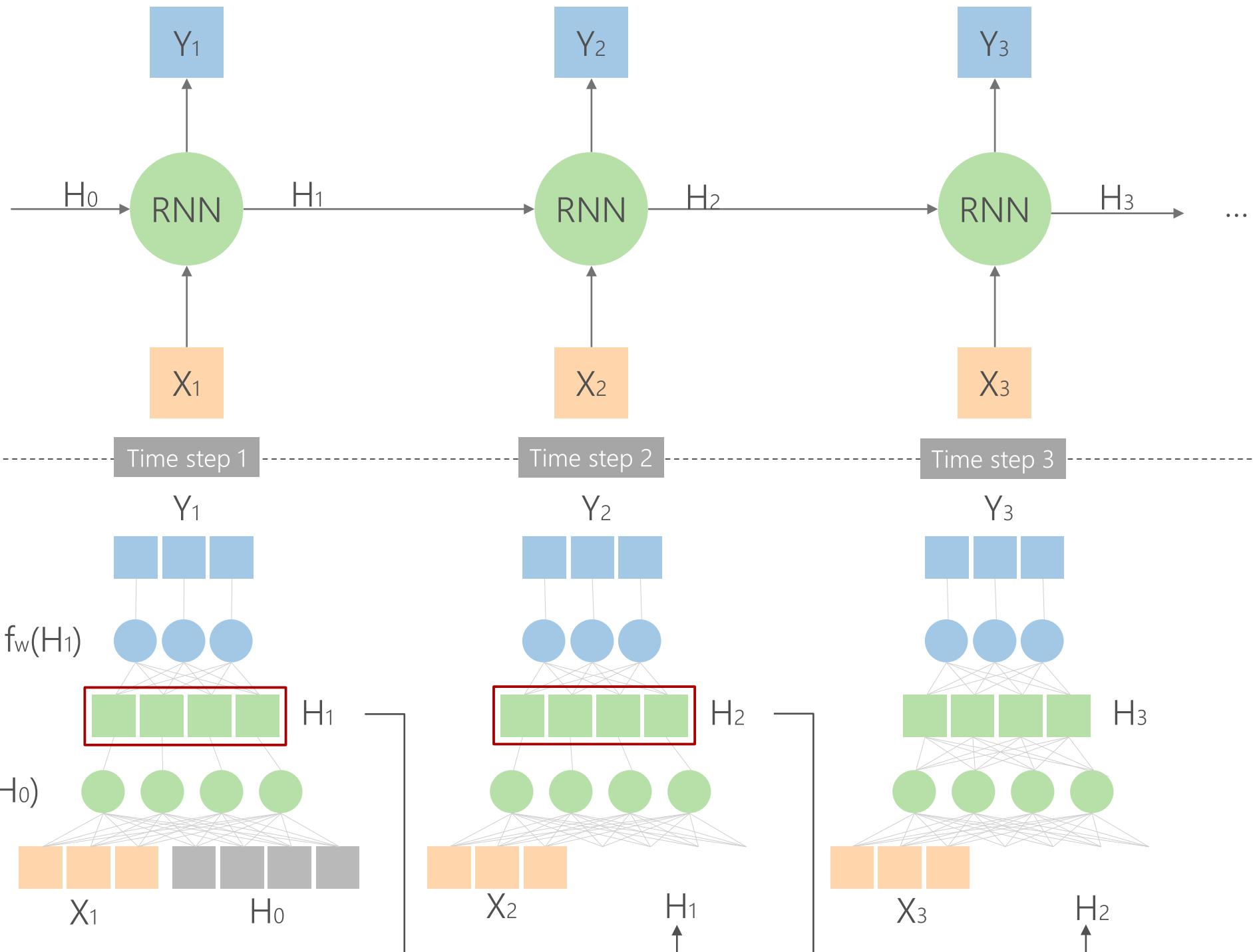
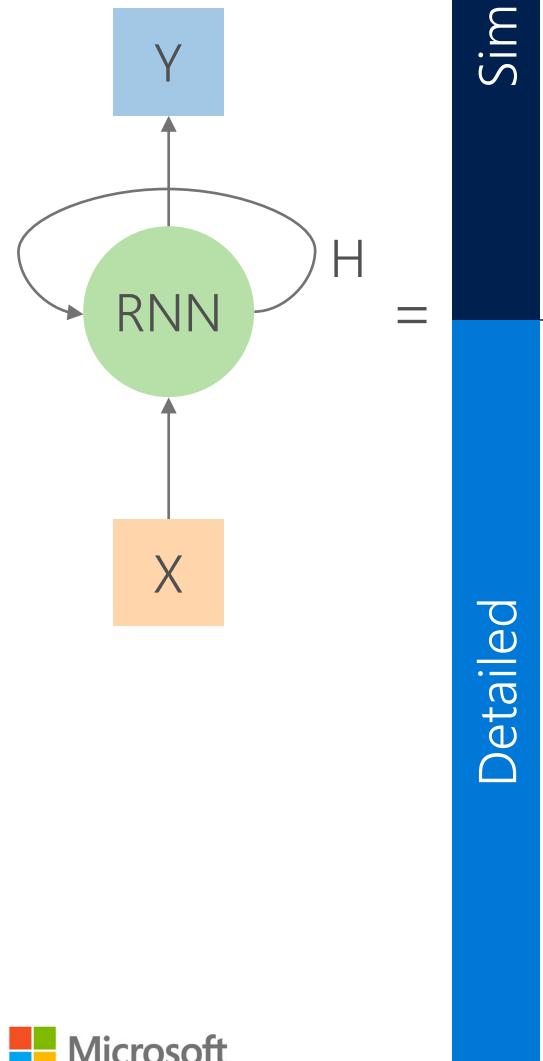


Unrolled RNN

The same weight and bias shared across all the steps



Unrolled RNN

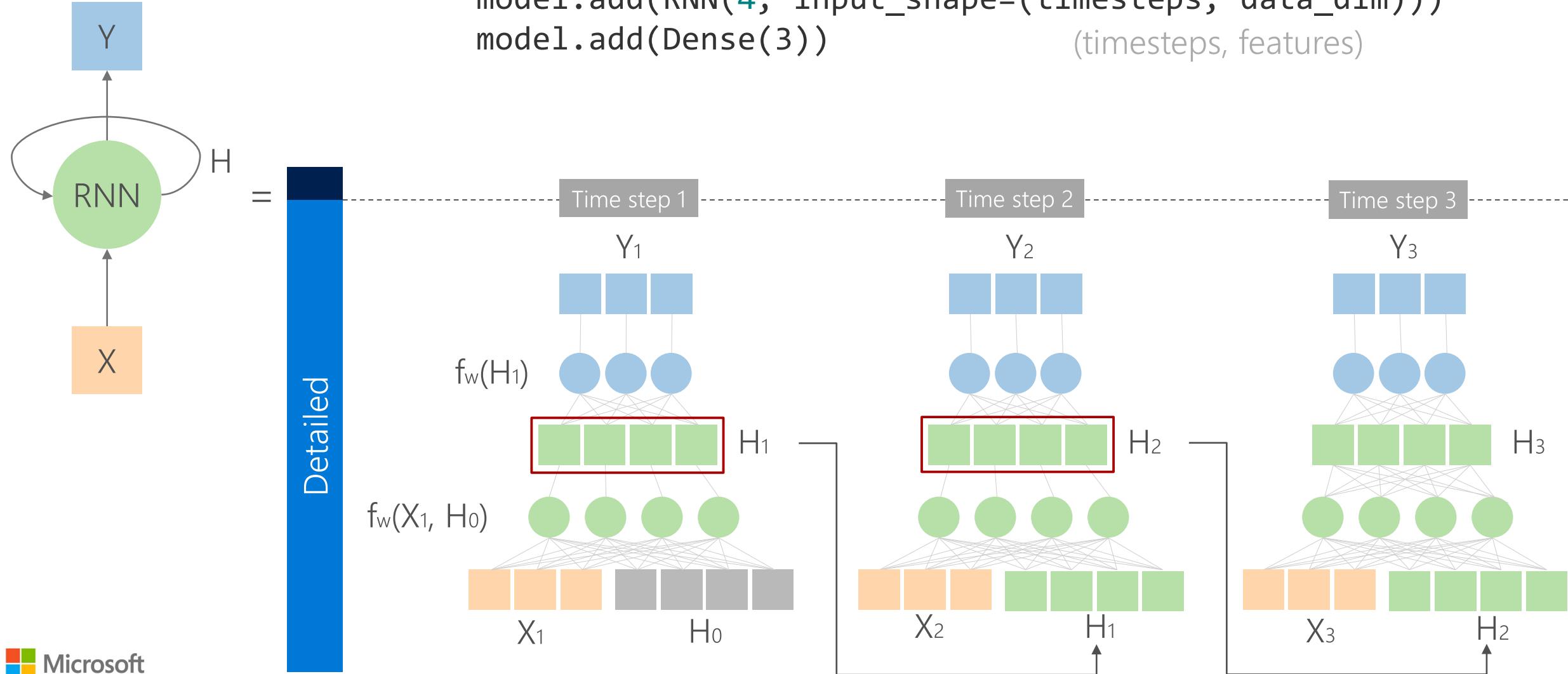


Unrolled RNN

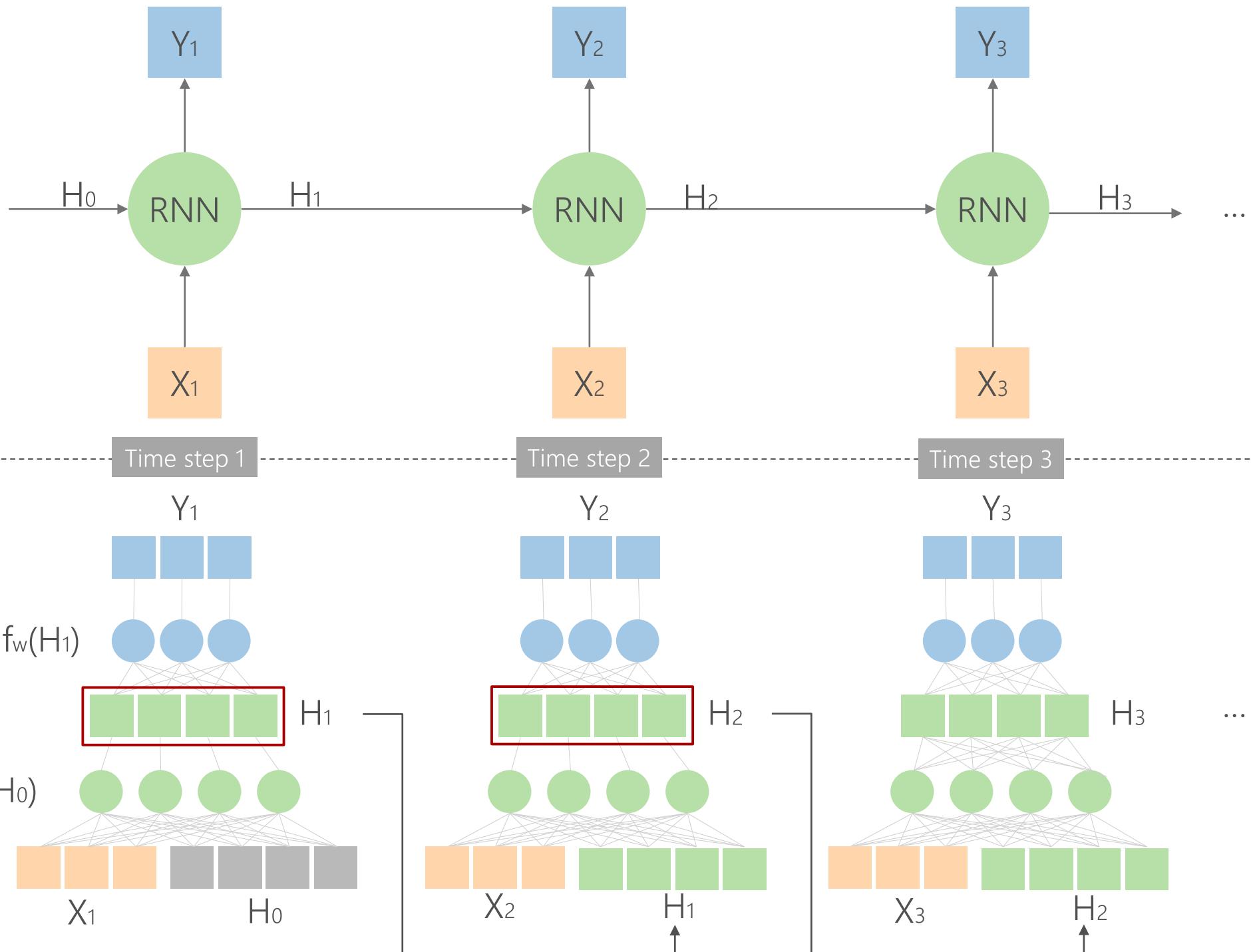
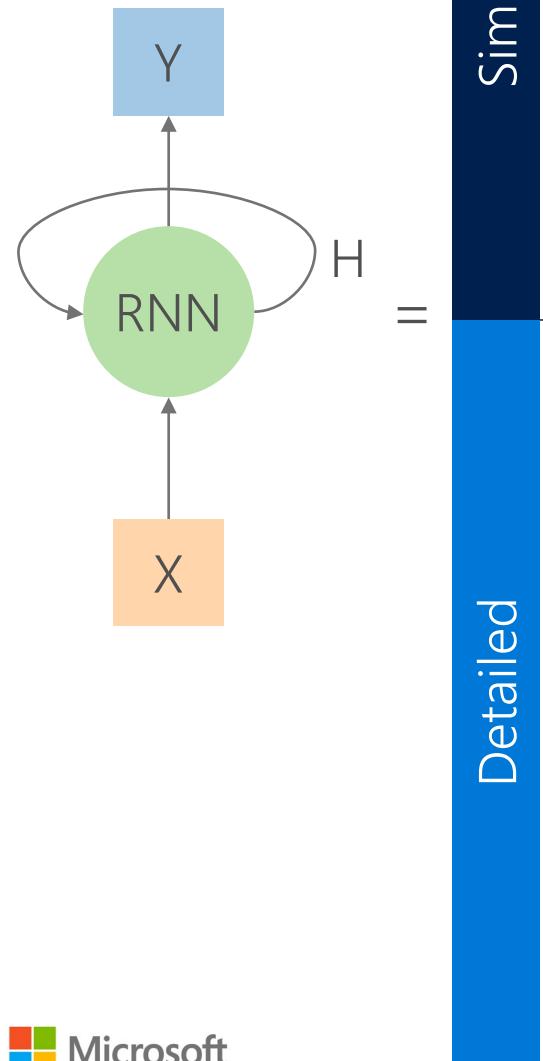
*In Keras, the parameter “units” is dimensions of hidden state.
(Think of it as feedforward neural network number of units in hidden layer.)*

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

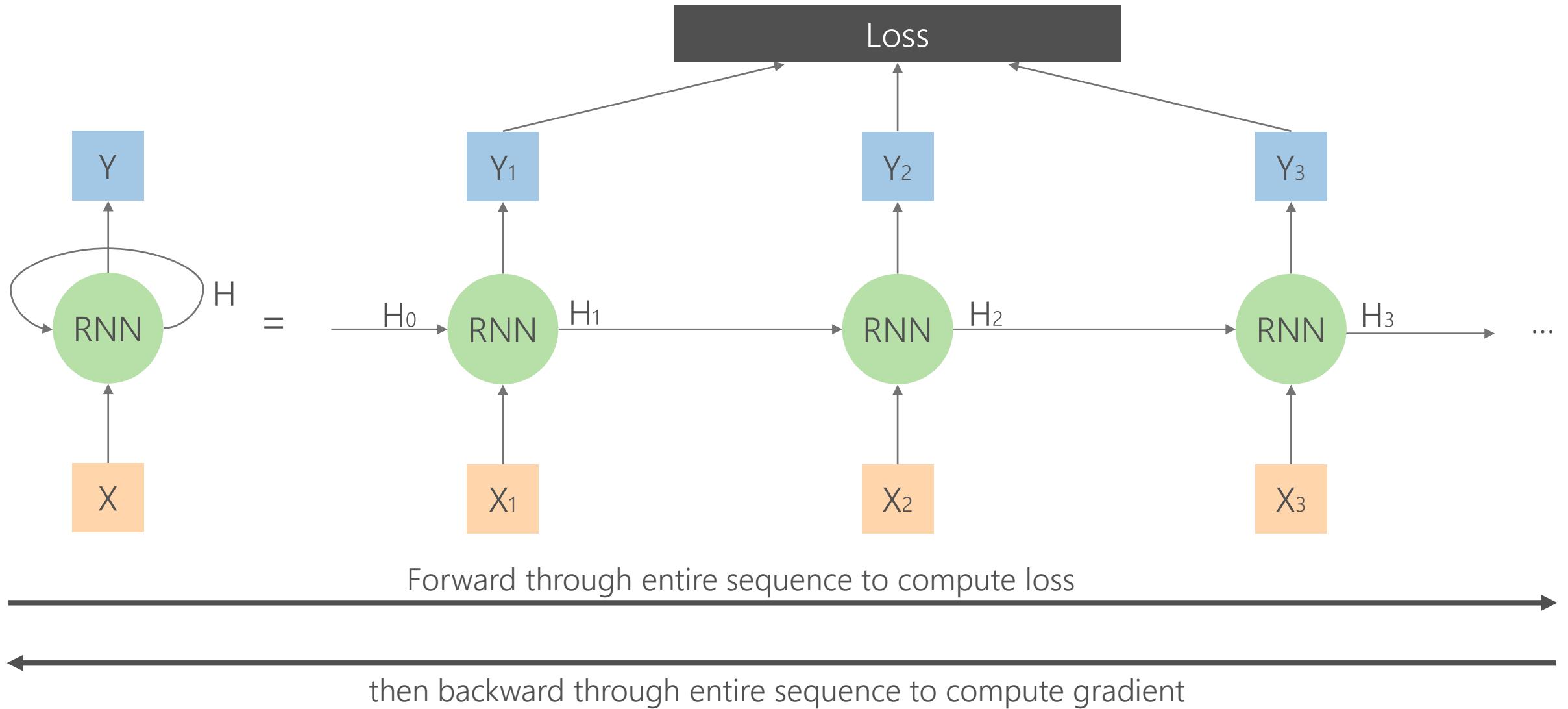
(timesteps, features)



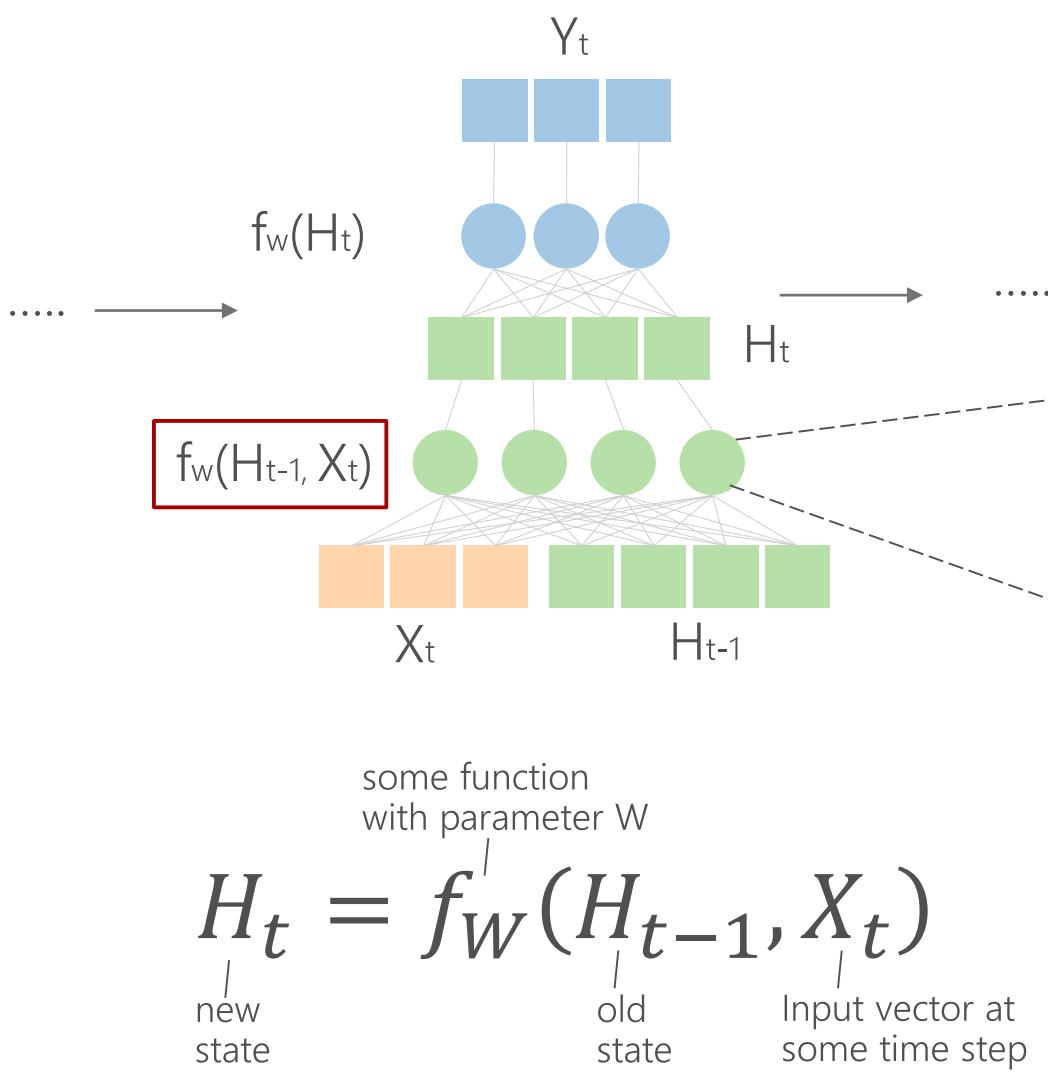
Unrolled RNN



Backpropagation through time (BPTT)

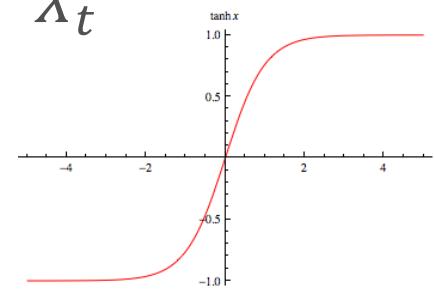
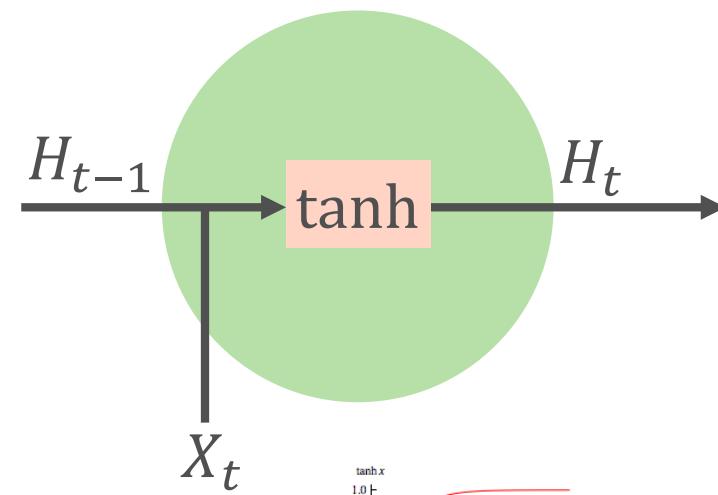


Vanilla RNN

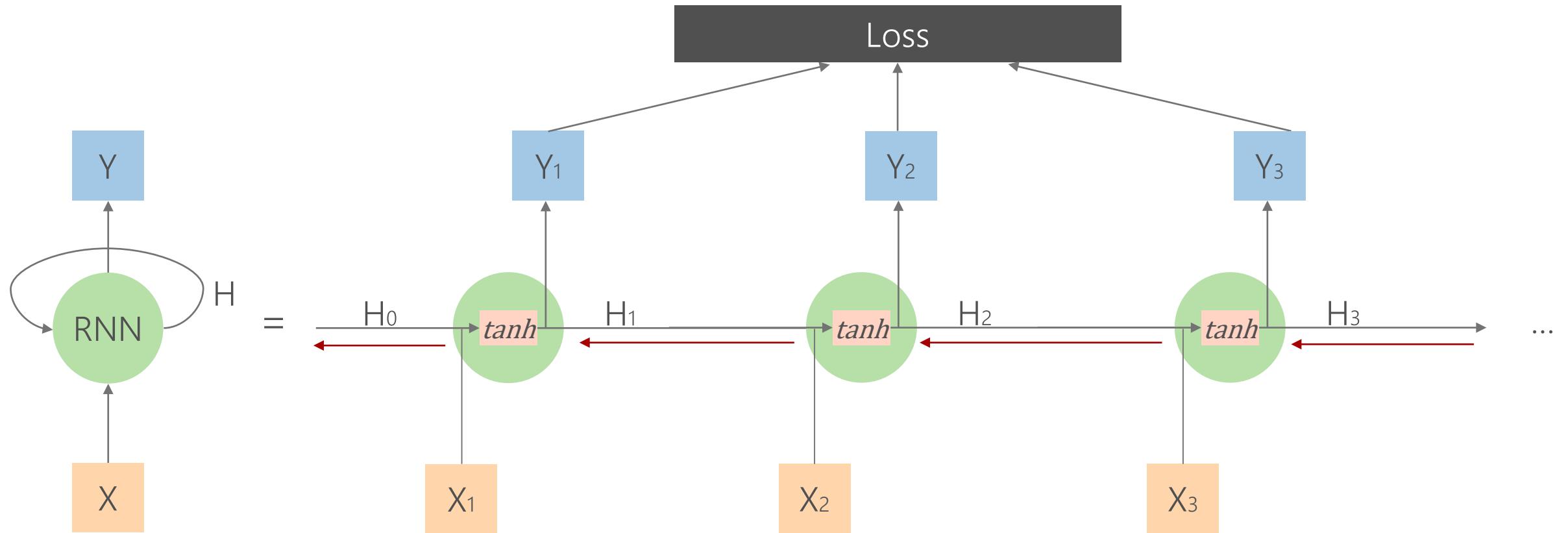


Vanilla RNN:

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



Vanilla RNN BPTT

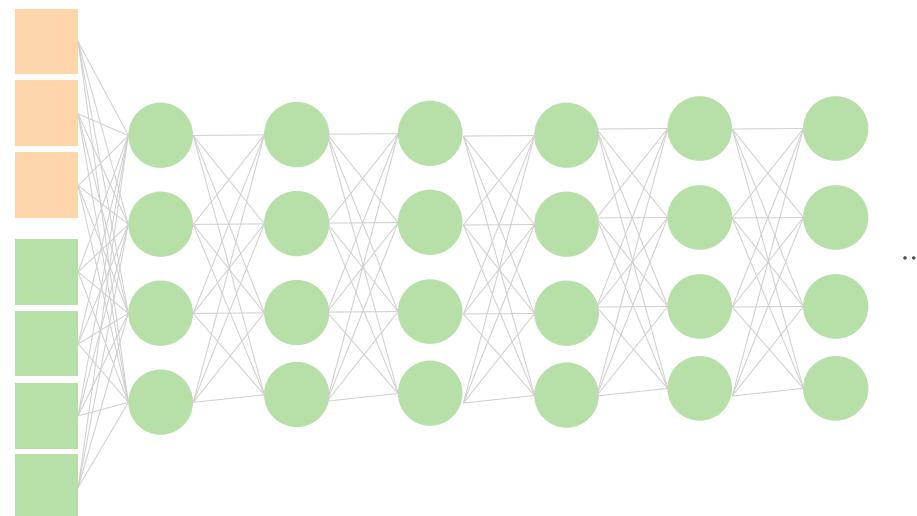


Computing gradient of h_0 involves repeated tanh and many factors of W

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

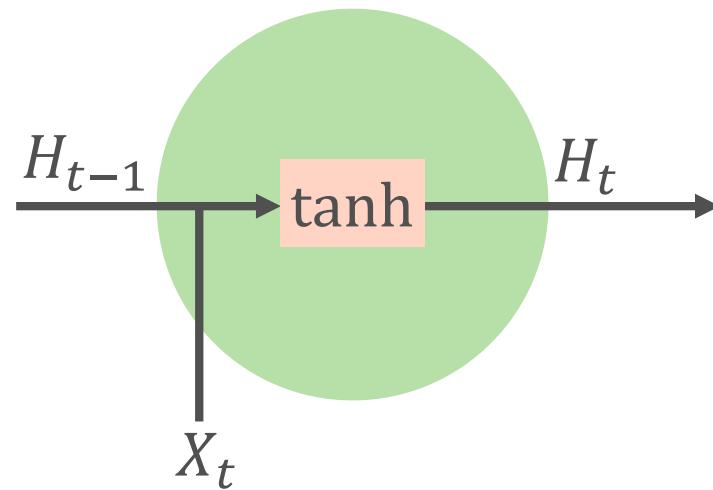
Vanishing Gradient

- 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)

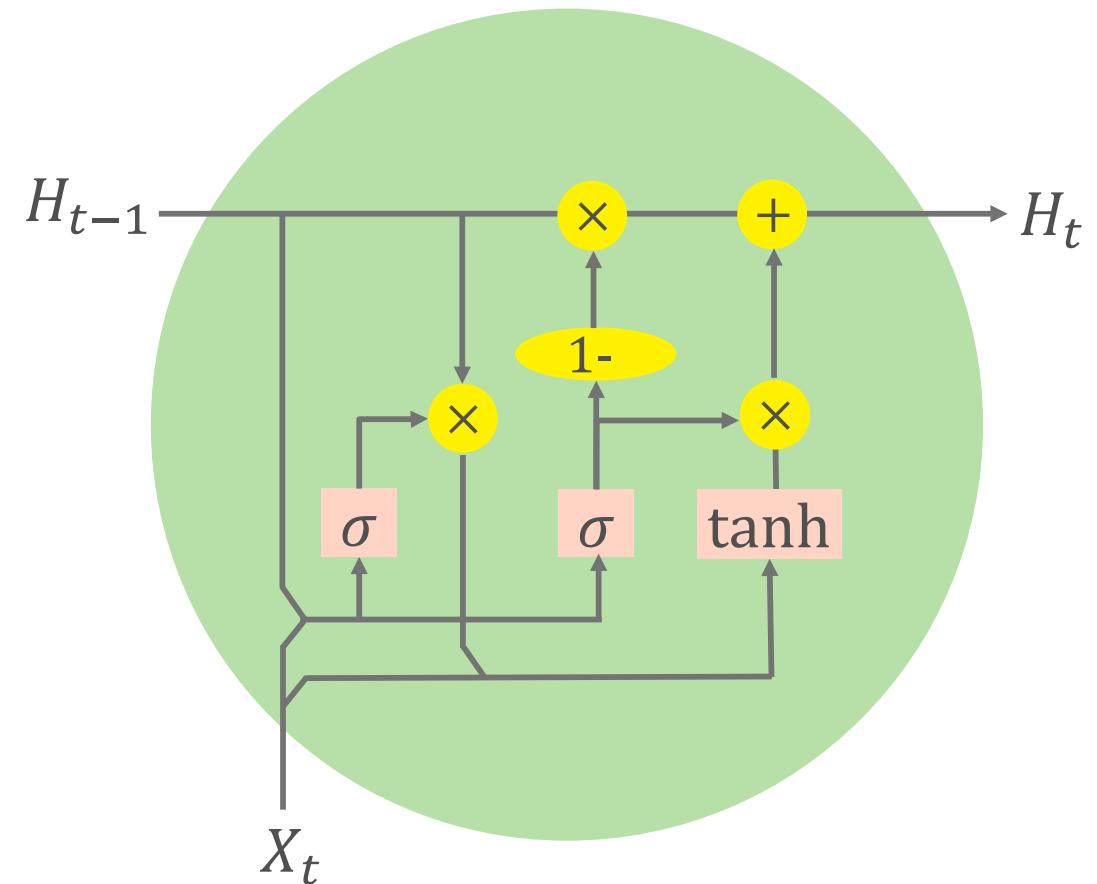
Vanilla RNN:

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

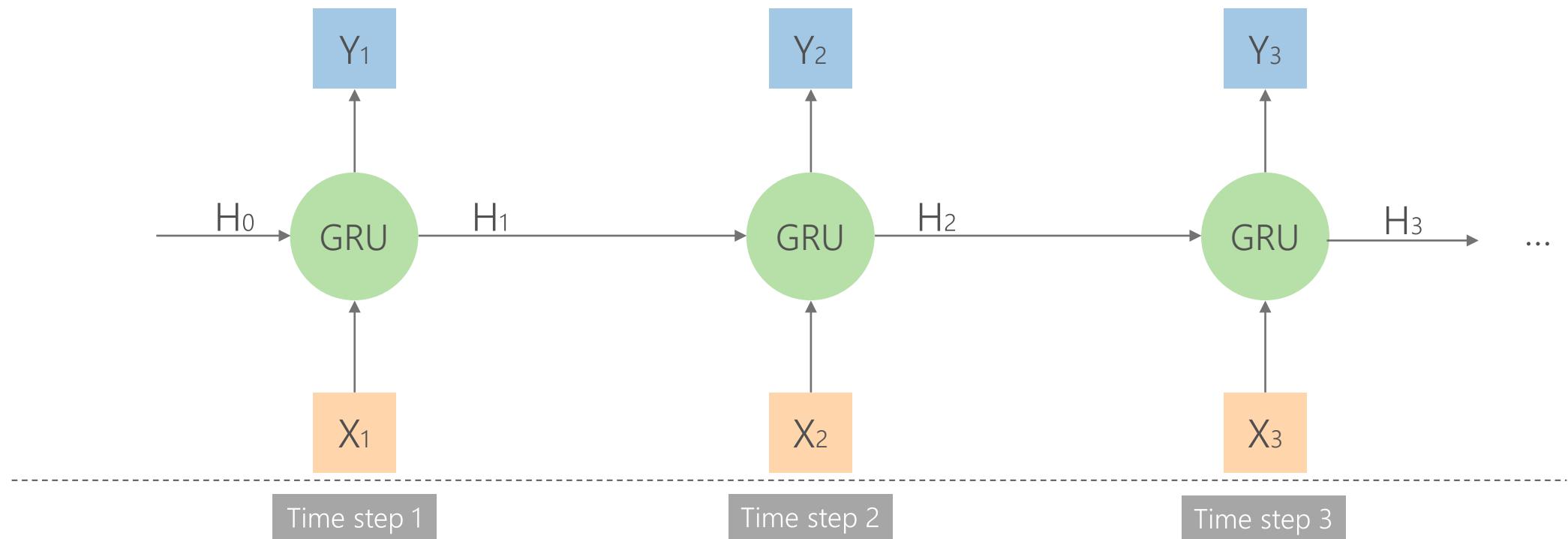


GRU:

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

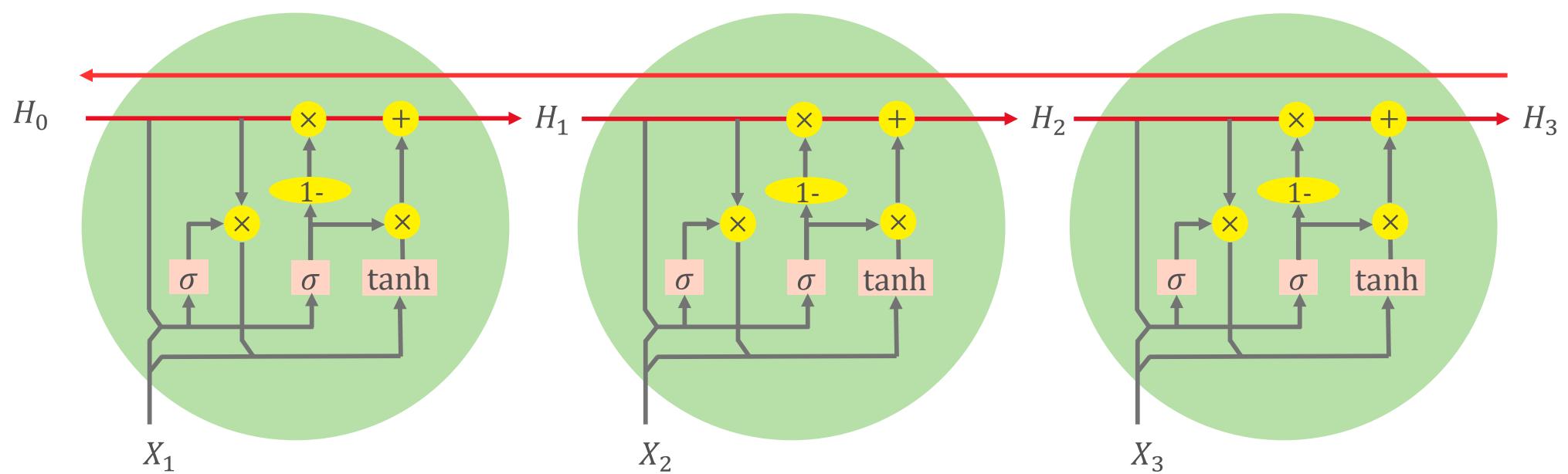


GRU

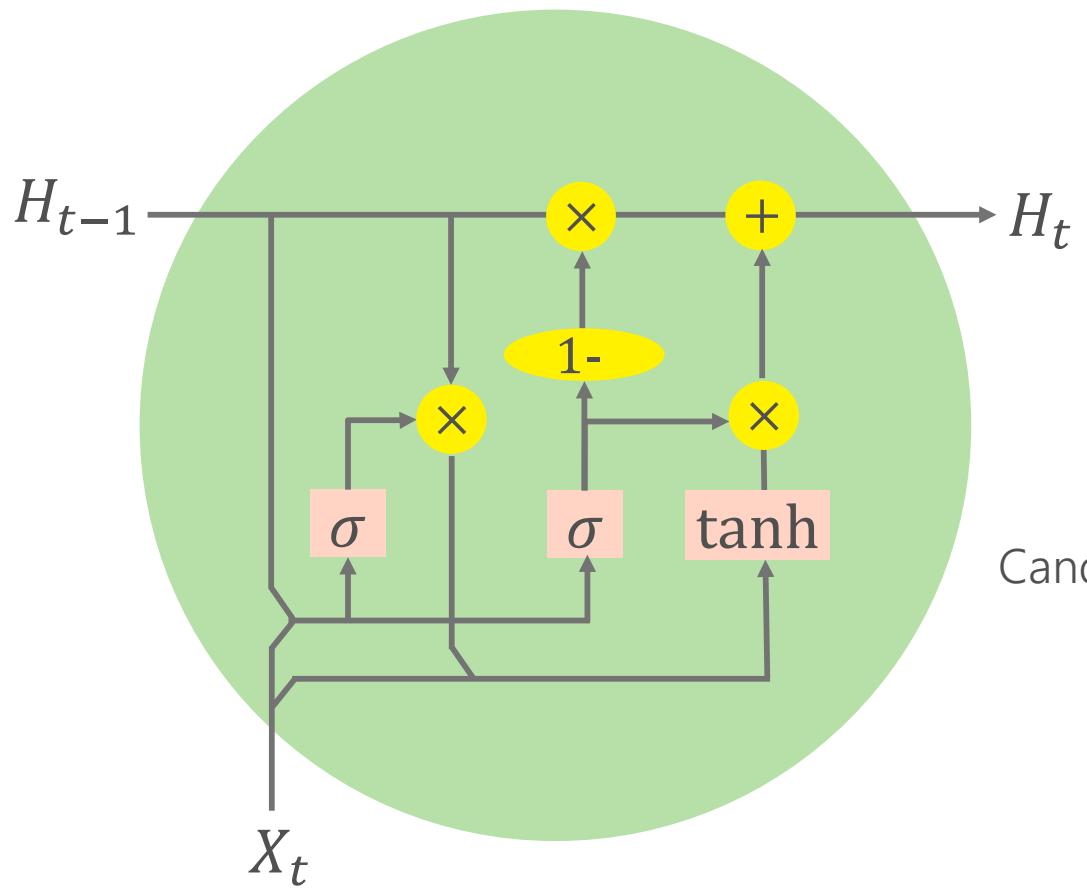


Uninterrupted gradient flow

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



Gated Recurrent Unit (GRU)

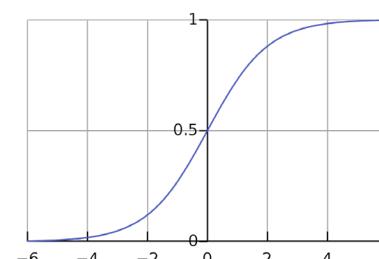


$$\text{Hidden state: } H_t = (1 - z_t) \times H_{t-1} + z_t \times \tilde{H}_t$$

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

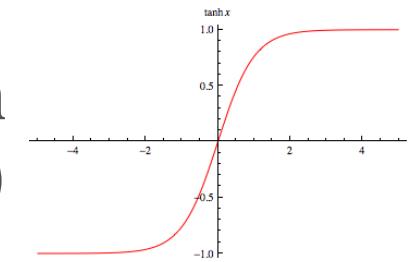
$$\text{Candidate gates/states: } \tilde{H}_t = \tanh(W \cdot [r_t \times H_{t-1}, X_t])$$

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



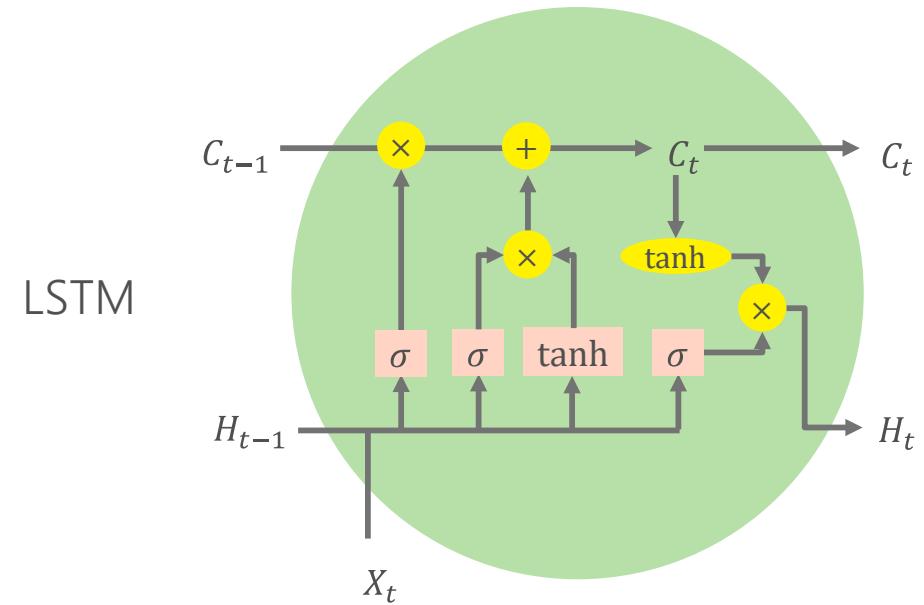
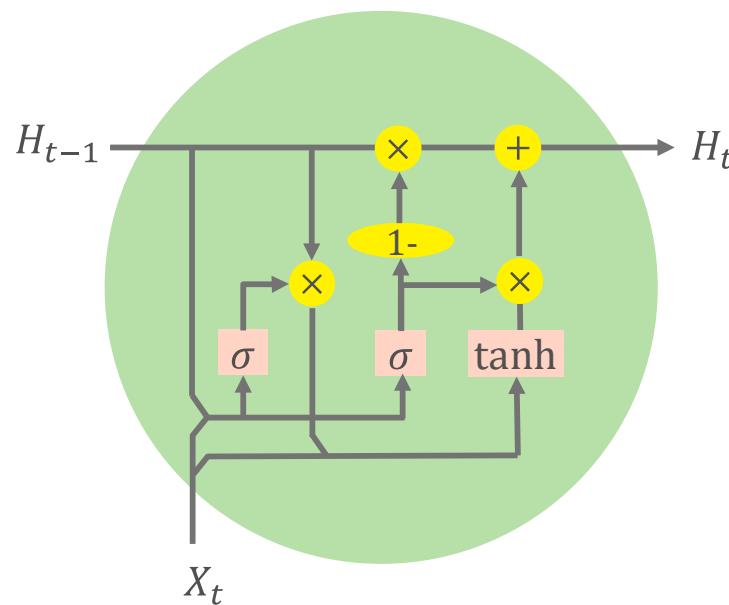
$$\sigma(0,1)$$

$$\tanh(-1, 1)$$



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

GRU vs LSTM (Long Short Term Memory)



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

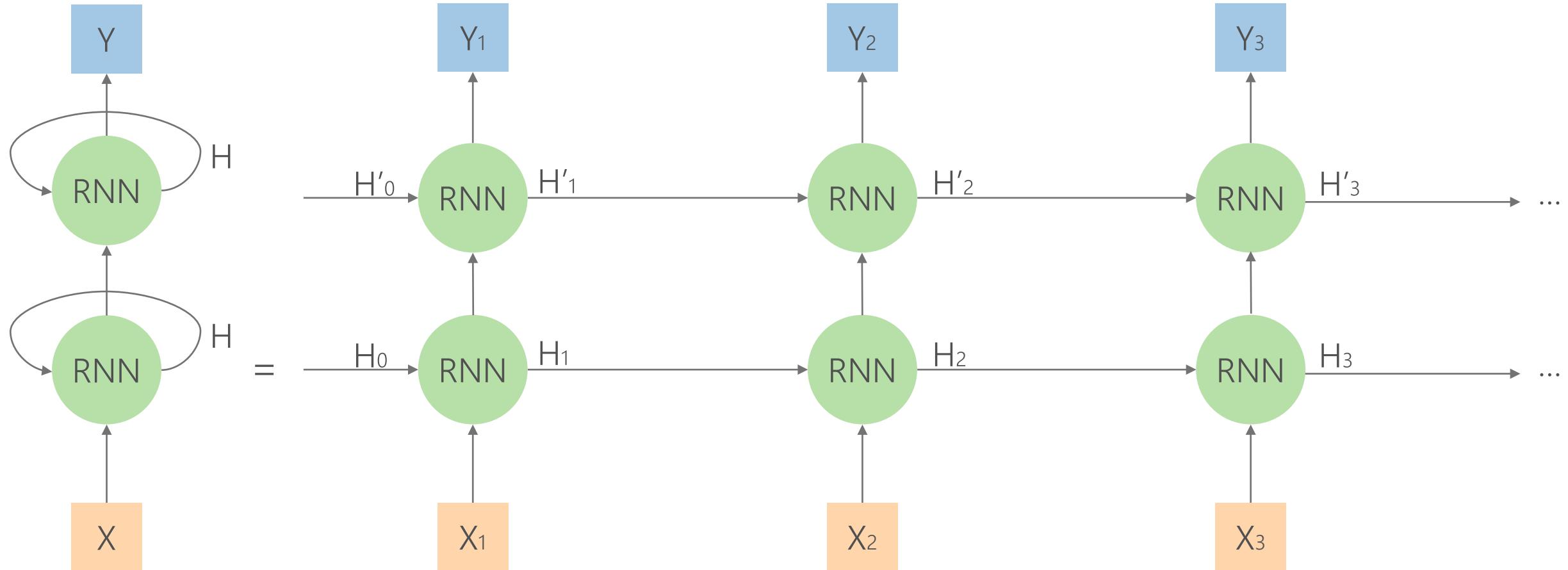
Hidden cell state: $C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$
 Hidden state: $H_t = o_t \times \tanh(C_t)$

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

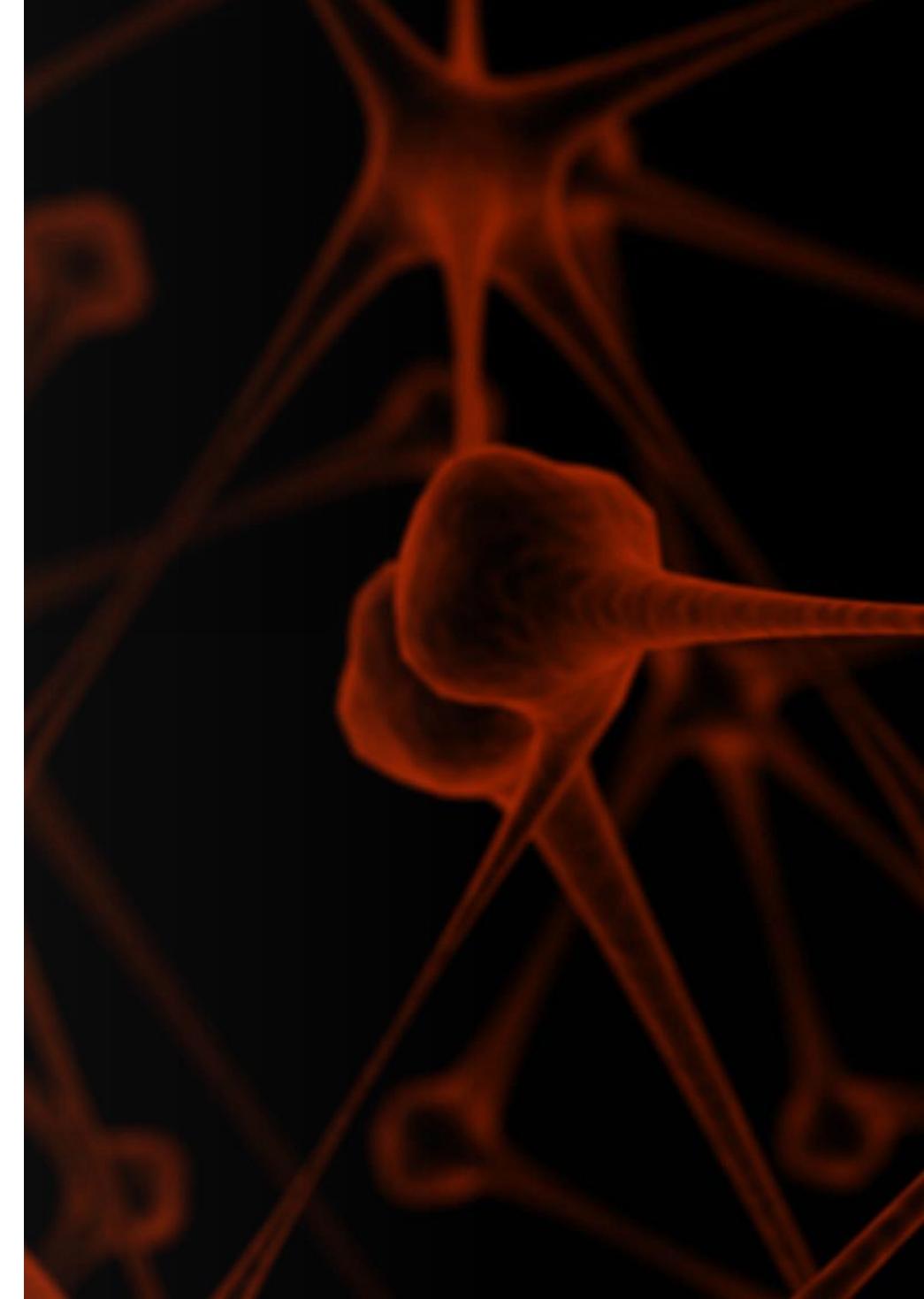
RNN Stacking

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



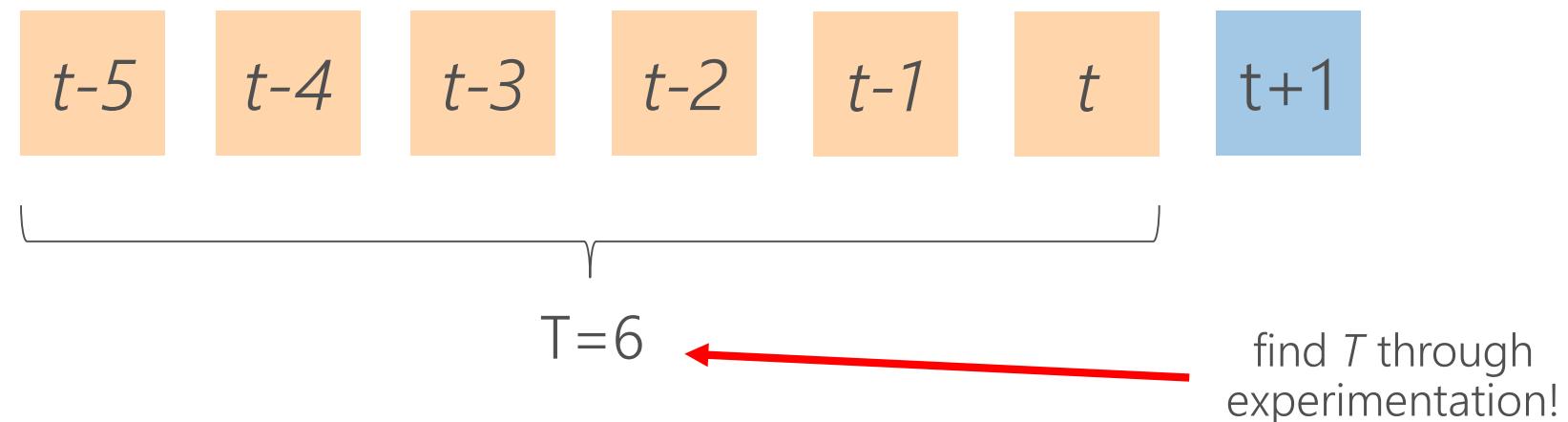
```
model = Sequential()  
model.add(RNN(4, return_sequences=True, input_shape=(timesteps, data_dim)))  
model.add(RNN(4))
```

RNN for one step time
series forecasting
RNN



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



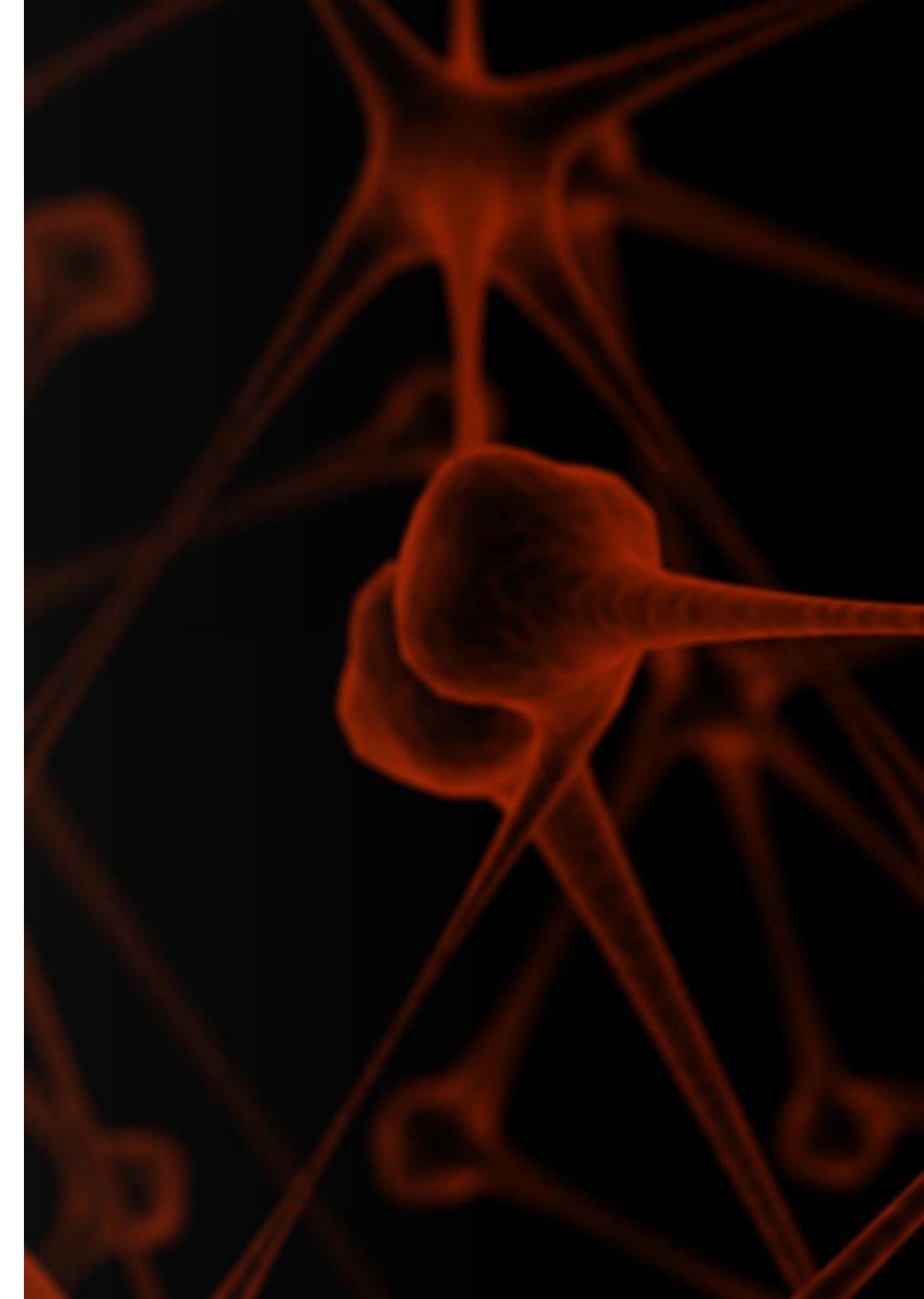
Hands-on Experiment

Open: 2_RNN.ipynb notebook

Hands-on Quiz

Open: Quiz_RNN.ipynb notebook

RNN for multi-step time series forecasting

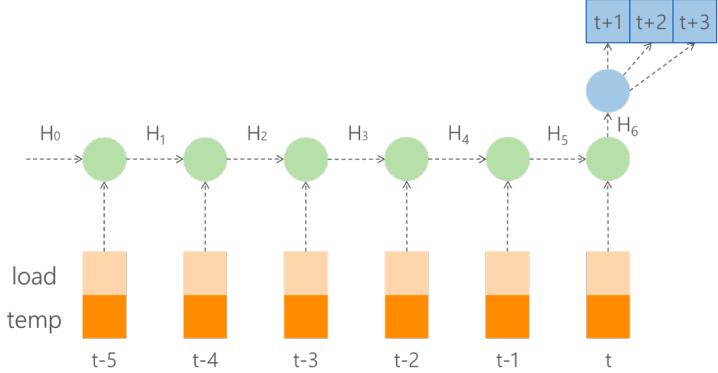


Multi-step forecast

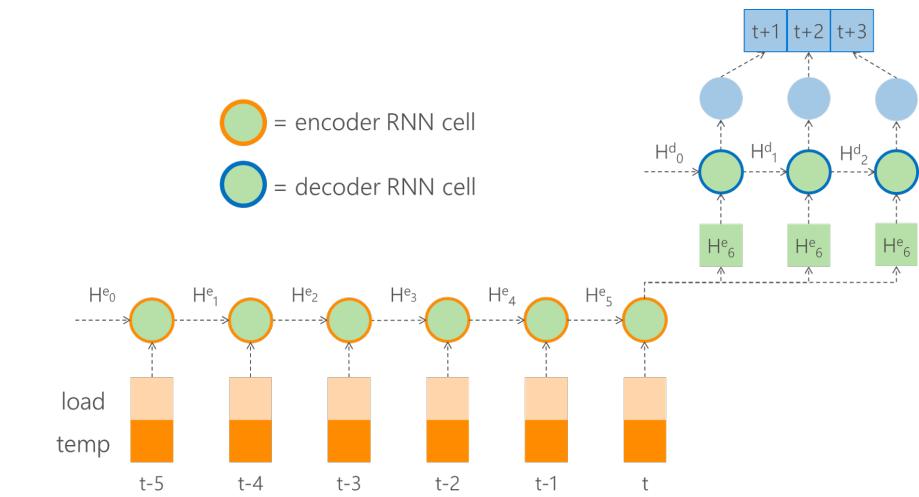
- Assuming we are at time t ...
- ... predict the values at times $(t+1, \dots, t+HORIZON)$...
- ... conditional on the previous T values of the time series



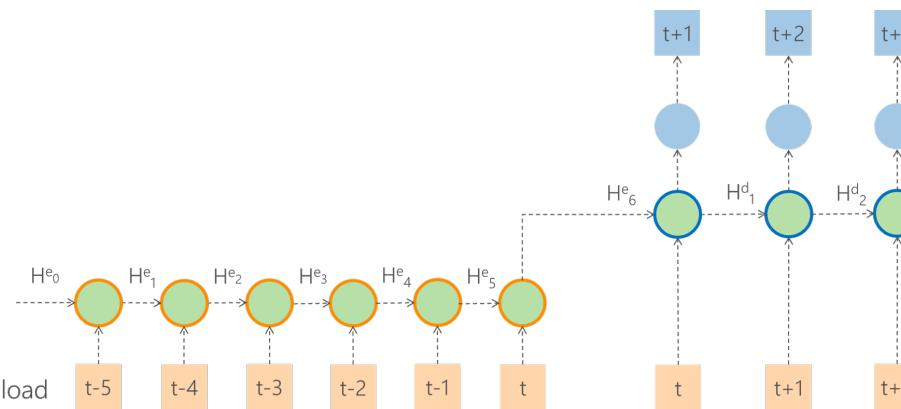
Multi-step forecast



Vector output

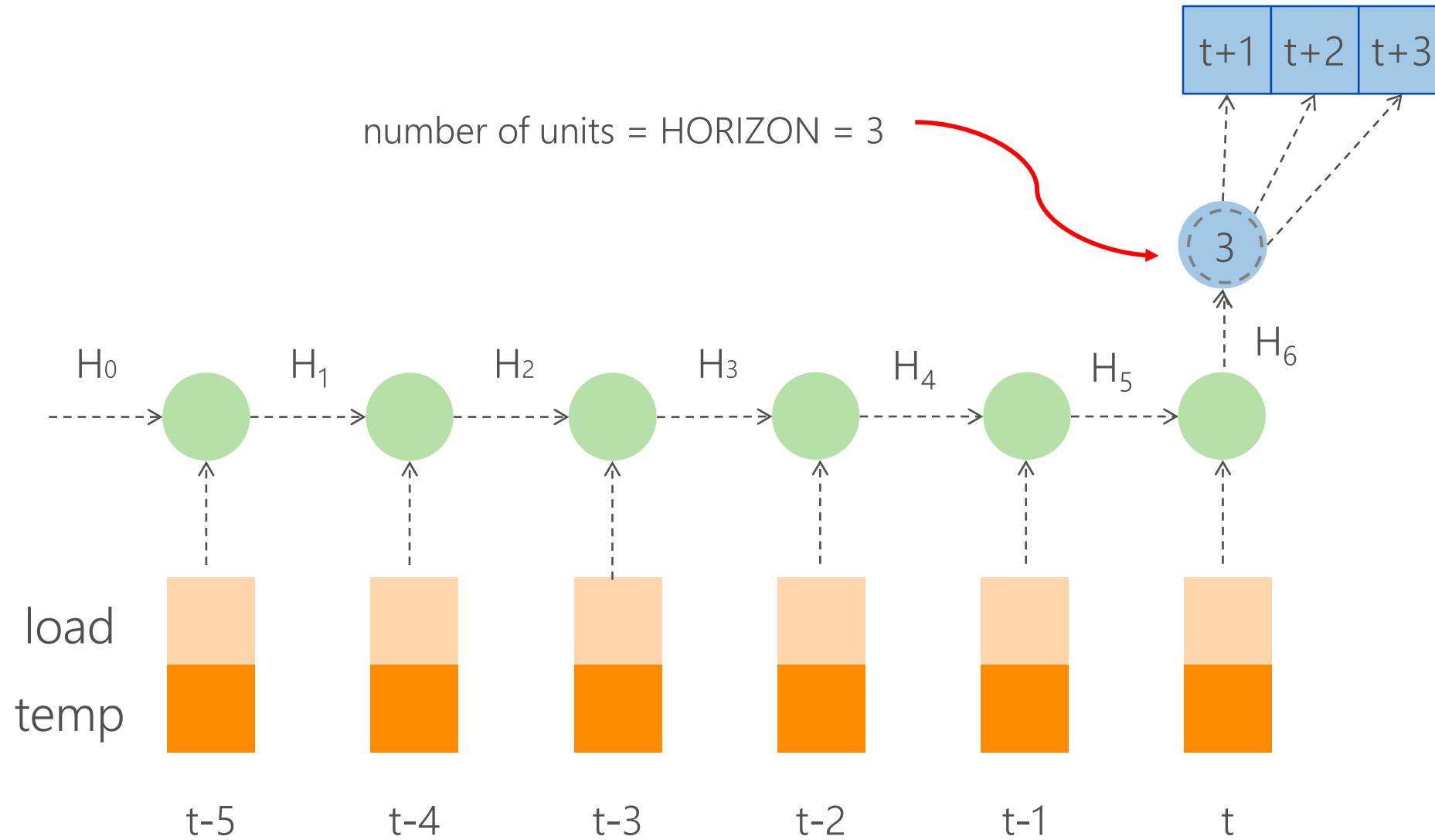


Simple encoder-decoder

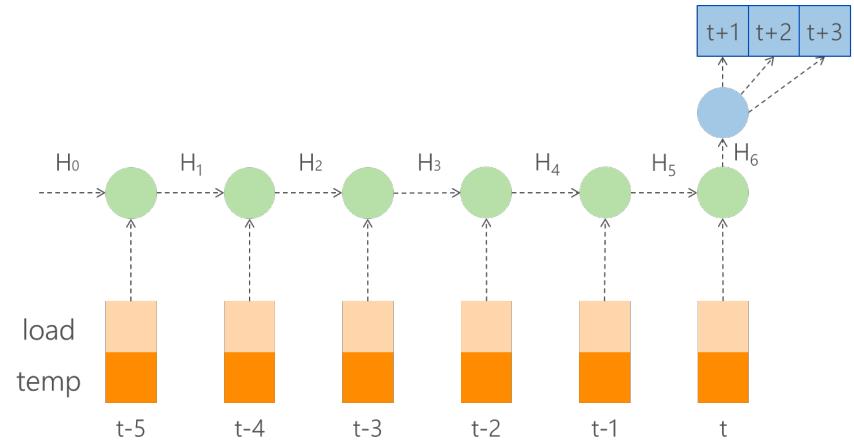


Recursive encoder-decoder

Vector output approach



Vector output approach



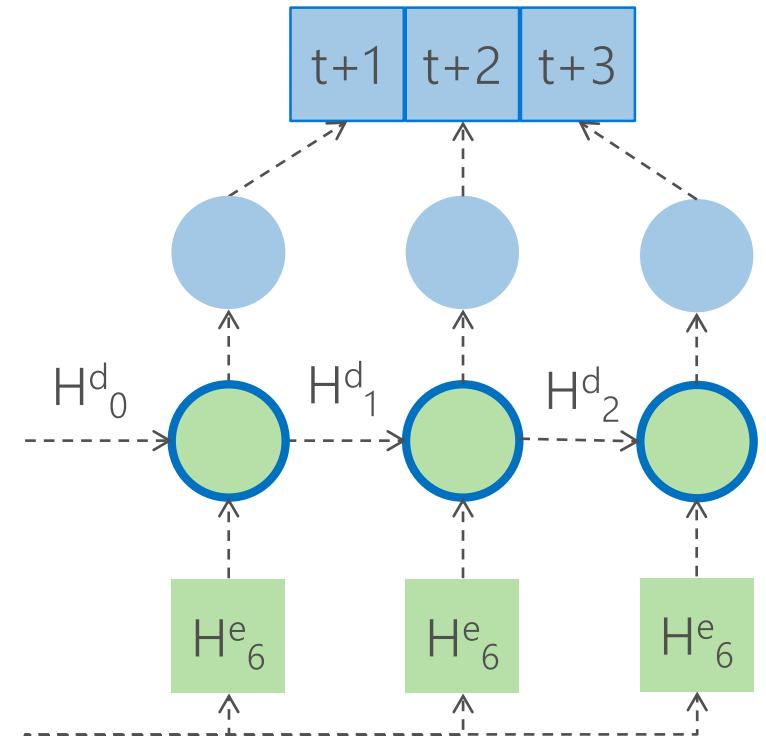
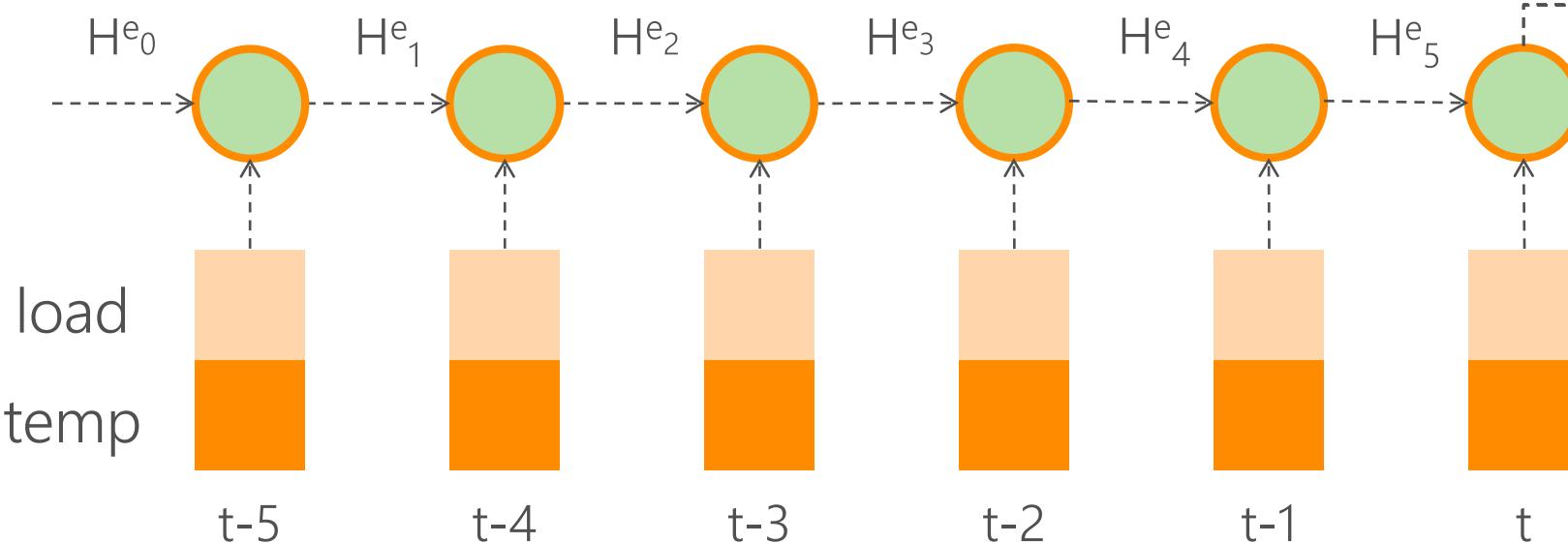
- 👍 Simplest to implement
- 👍 Fastest to train
- 👎 Does not model dependencies between predicted outputs

Reference notebook: [multi_step_RNN_vector_output.ipynb](#)

Simple encoder-decoder

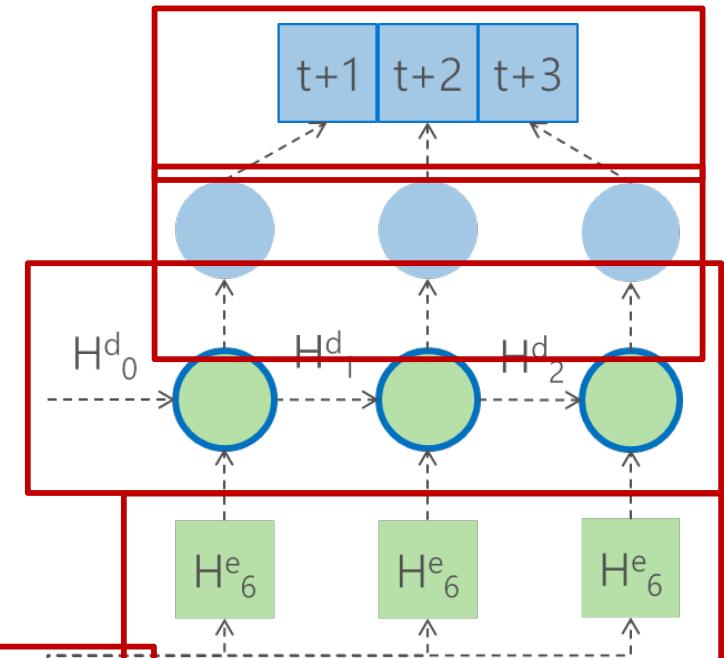
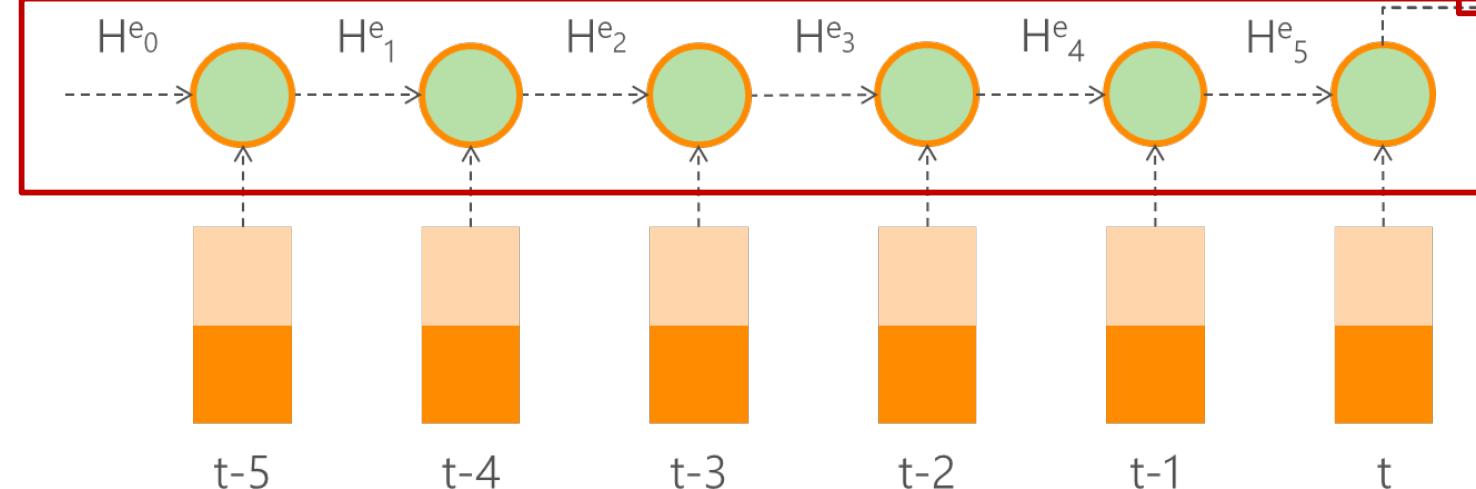
 = encoder RNN cell

 = decoder RNN cell

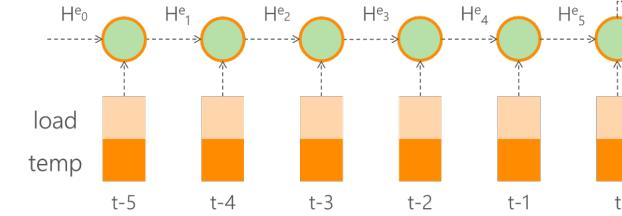
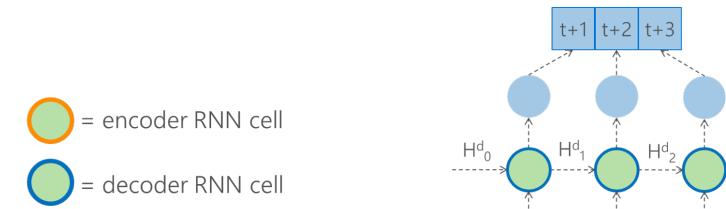


Simple encoder-decoder

```
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())
```



Simple encoder-decoder

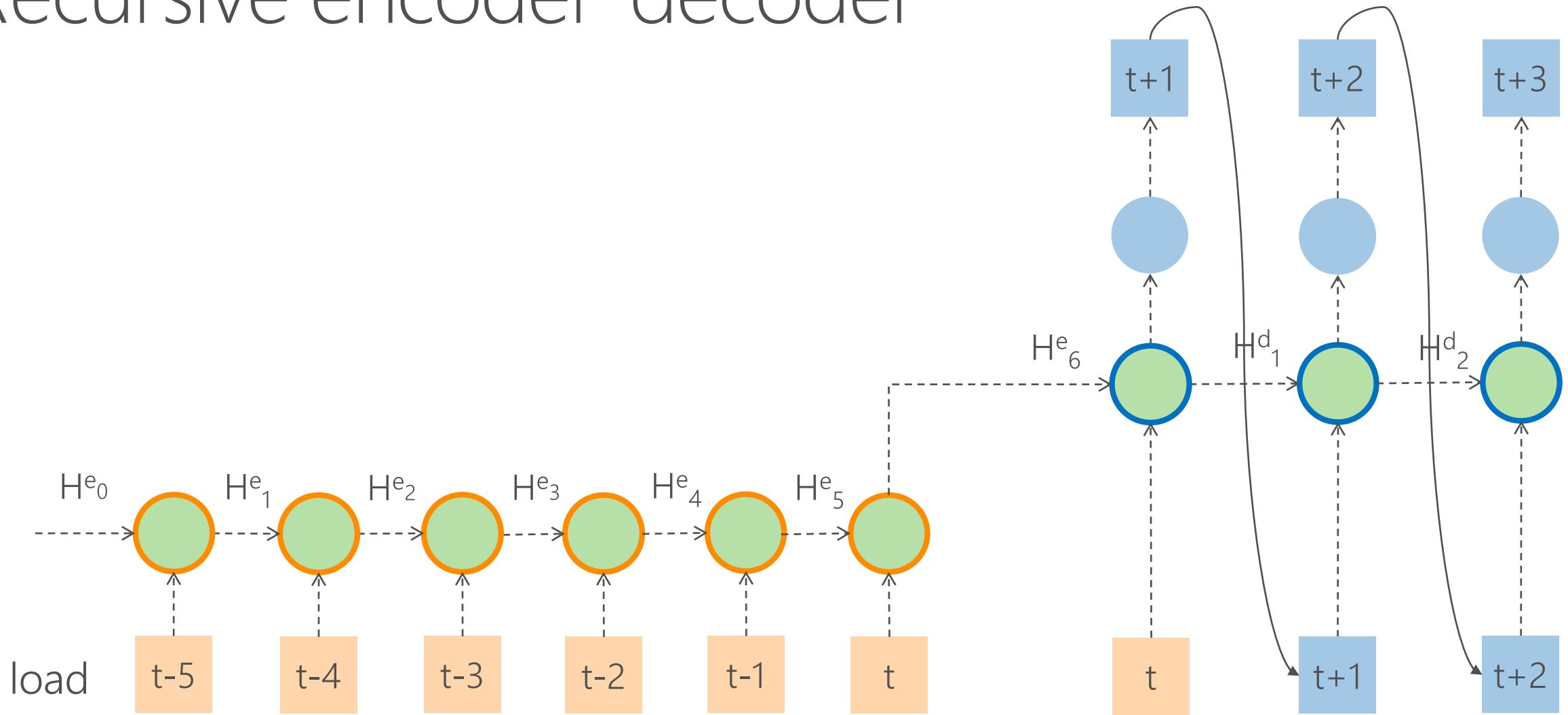


- 👍 Fairly simple to implement
- 👍 Tries to capture dependencies between forecasted time steps through decoder hidden state
- 👎 Slower to train with stacked RNN layers

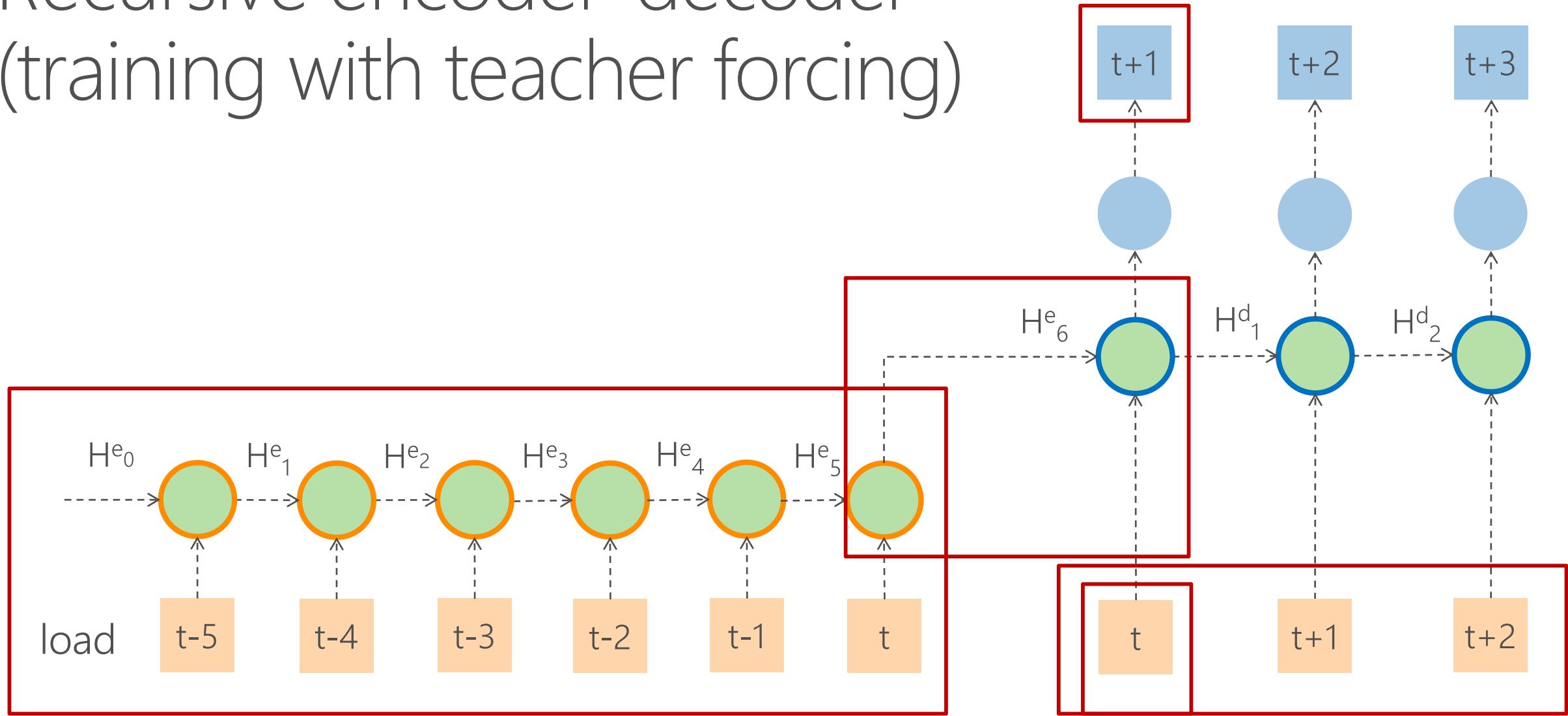
Hands-on Experiment

Open 3_RNN_encoder_decoder.ipynb

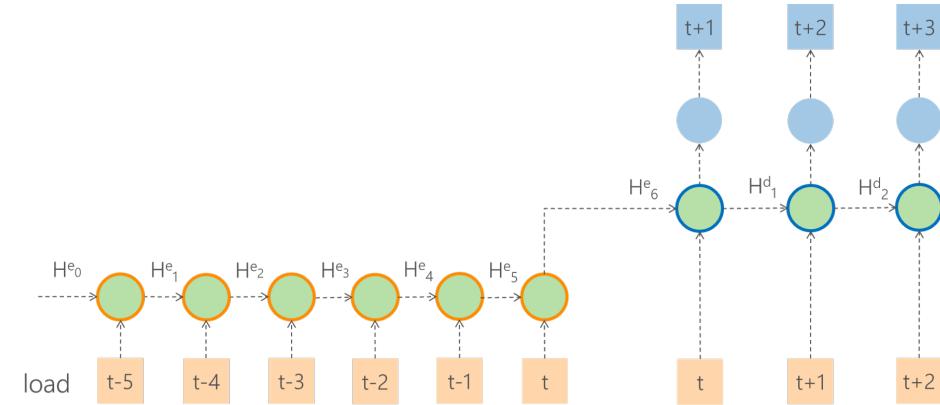
Recursive encoder-decoder



Recursive encoder-decoder (training with teacher forcing)



Encoder-decoder with teacher forcing

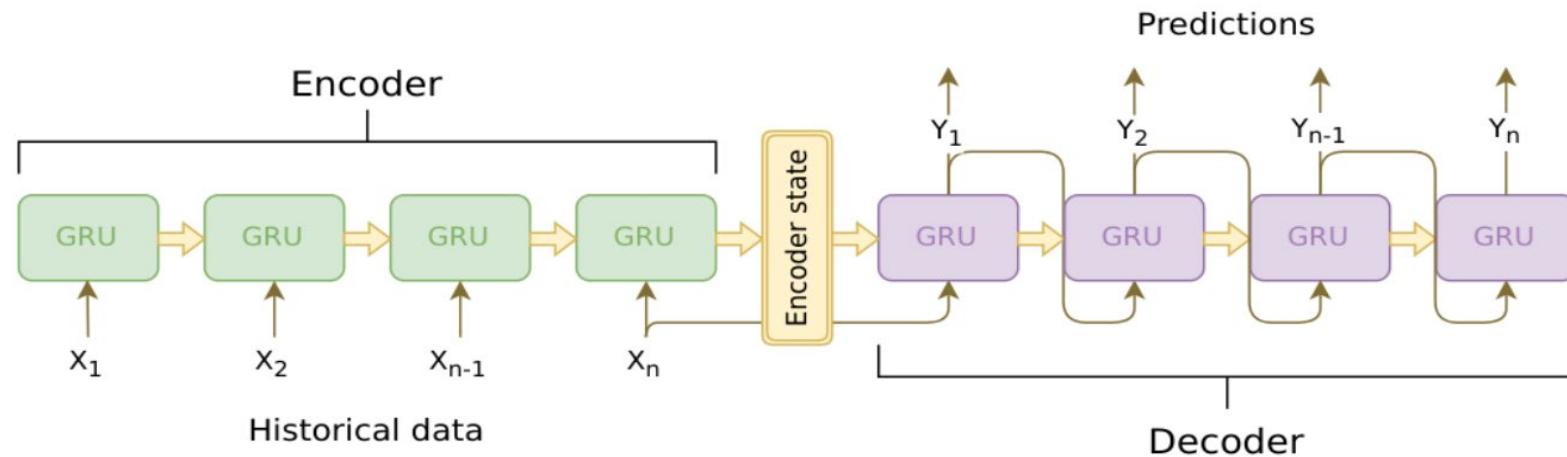


- 👍 Tries to capture dependencies between forecasted time steps through recursive decoder
- 👍 Variable length output (can generate forecasts of variable horizons)
- 👎 More difficult to implement
- 👎 Slower to train and slower to generate forecasts
- 👎 Error propagation due to recursive decoder

Web traffic forecasting competition

Forecast traffic of 145K Wikipedia pages

Winning solution: github.com/Arturus/kaggle-web-traffic



- Feature engineering: transform univariate to multivariate time series by adding seasonality features:
$$x_t \rightarrow (x_t, x_{t-\text{quarter}}, x_{t-\text{year}})$$
- COCOB optimizer that doesn't require learning rate tuning and converges considerably faster, custom implementation
- Sophisticated technique for building ensemble of RNN models, mainly for reducing model variance

Hands-on Quiz

Open Quiz_RNN_encoder_decoder.ipynb



Hybrid models for time series forecasting

Agenda

- ES-RNN
- Deep State Space Model
- Multi-Horizon Quantile Recurrent Forecaster

ES-RNN

S. Smyl, 2018

<https://eng.uber.com/m4-forecasting-competition/>

Winner of M4 competition

Simple Exponential Smoothing (SES)

Time-series: y_1, y_2, \dots, y_t

$$\hat{y}_{t+1|t} = \alpha \cdot y_t + (1 - \alpha) \cdot \hat{y}_{t|t-1}$$

↑
forecast of next value

↑
actual current value

↑
forecast of current value

↑
learned parameter

Equivalent form:

forecasting

$$\hat{y}_{t+1|t} = l_t \quad \xleftarrow{\text{level}}$$

smoothing

$$l_t = \alpha \cdot y_t + (1 - \alpha) \cdot l_{t-1}$$

Exponential Smoothing (ES)

forecasting

$$\hat{y}_{t+1|t} = l_t \cdot s_{t+1}$$

seasonality term

smoothing

$$l_t = \alpha \cdot (y_t / s_t) + (1 - \alpha) \cdot l_{t-1}$$

forecast of next level

forecast of current level

actual value of the current level

$$s_{t+m} = \gamma \cdot (y_t / l_t) + (1 - \gamma) \cdot s_t$$

forecast of the next seasonality term

forecast of the current seasonality term

actual value of the current seasonality term

m - length of seasonality

ES-RNN

$$\hat{y}_{t+1|t} = l_t \cdot s_{t+1} \cdot RNN(x_t)$$

$$l_t = \alpha \cdot (y_t / s_t) + (1 - \alpha) \cdot l_{t-1}$$

$$s_{t+m} = \gamma \cdot (y_t / l_t) + (1 - \gamma) \cdot s_t$$

deseasonalized normalized
time-series



$$x_t = y_t / (l_t \cdot s_t)$$

ES-RNN

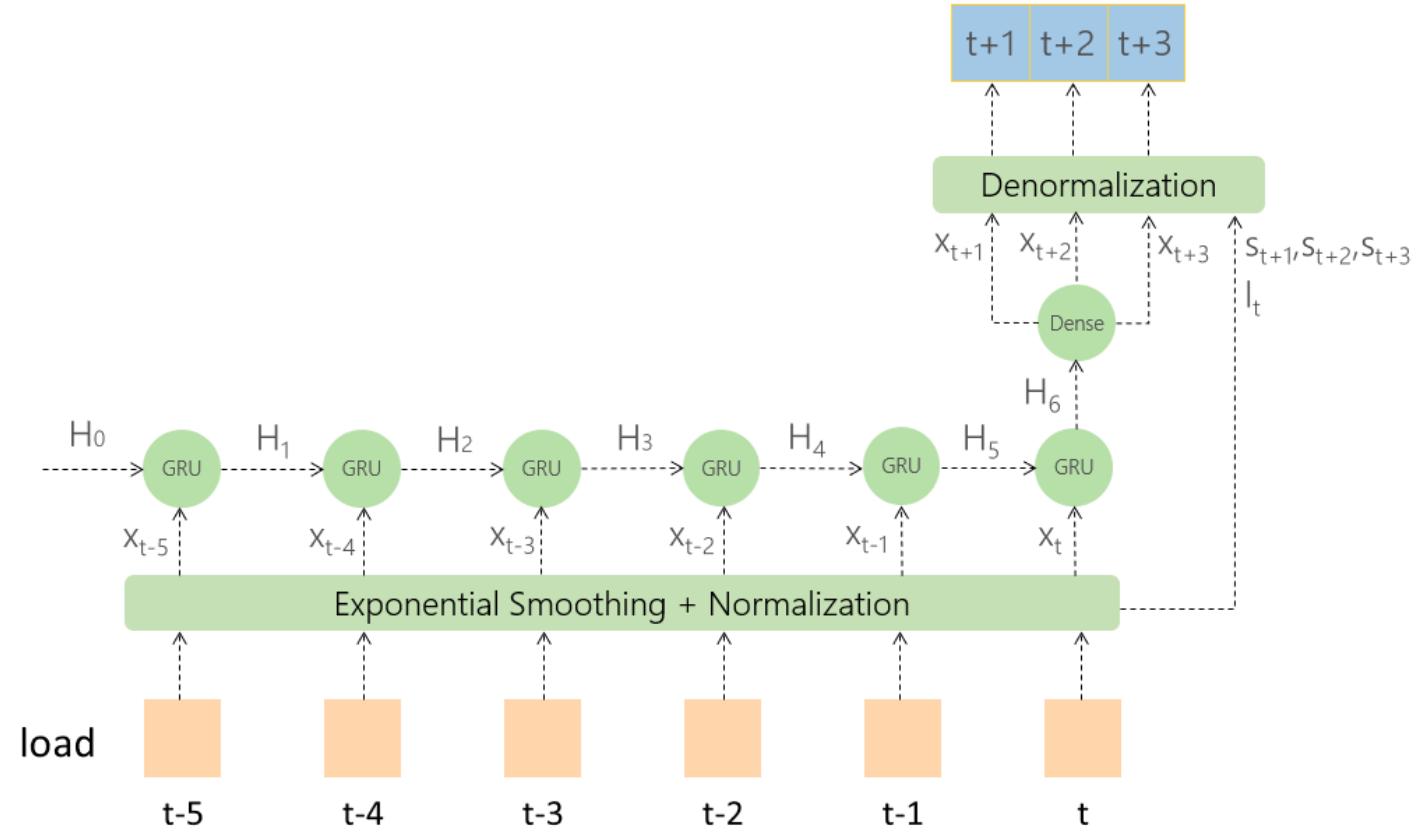
ES-RNN can be used to forecast multiple time-series

Local parameters for each time series: $\alpha, \gamma, s_0, s_1, \dots, s_{m-1}$

Global parameters, shared by all time series: weights of connections inside RNN

Cost function depends on all local and global parameters

Local and global parameters are learned by minimizing cost function (using BPTT)



Deep State Space Model

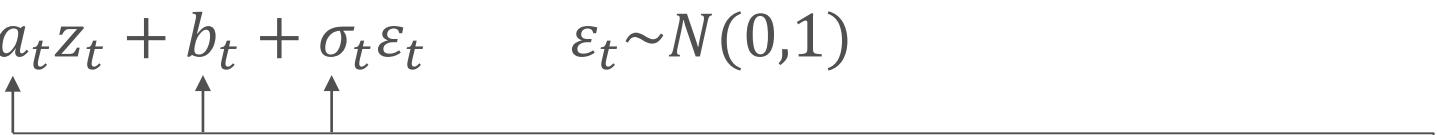
S.S. Rangapuram et al., 2018

<https://papers.nips.cc/paper/8004-deep-state-space-models-for-time-series-forecasting>

State Space Model

Time-series: y_1, y_2, \dots, y_t

Assumption 1: time series observation y_t has a hidden latent state z_t that encodes trend and seasonality patterns

$$y_t = a_t z_t + b_t + \sigma_t \varepsilon_t \quad \varepsilon_t \sim N(0,1)$$


Assumption 2: latent state z_t evolves over time

$$z_{t+1} = F_t z_t + g_t \bar{\varepsilon}_t \quad \bar{\varepsilon}_t \sim N(0,1)$$


parameters

ARIMA and Exponential Smoothing are State Space Models

Deep State Space Model

Time-series: y_1, y_2, \dots, y_t

$$y_t = a_t z_t + b_t + \sigma_t \varepsilon_t \quad \varepsilon_t \sim N(0,1)$$

$$z_{t+1} = F_t z_t + g_t \bar{\varepsilon}_t \quad \bar{\varepsilon}_t \sim N(0,1)$$

Parameters: $a_t, b_t, \sigma_t, F_t, g_t$

Feature vectors: x_1, x_2, \dots, x_t

Use LSTM to learn parameters of state space models from feature vectors

Assumption: feature vector x_{t+1} is known when forecasting y_{t+1}

Deep State Space Model can be trained to forecast multiple time-series

Multi-Horizon Quantile Recurrent Forecaster

R. Wen et al., 2017

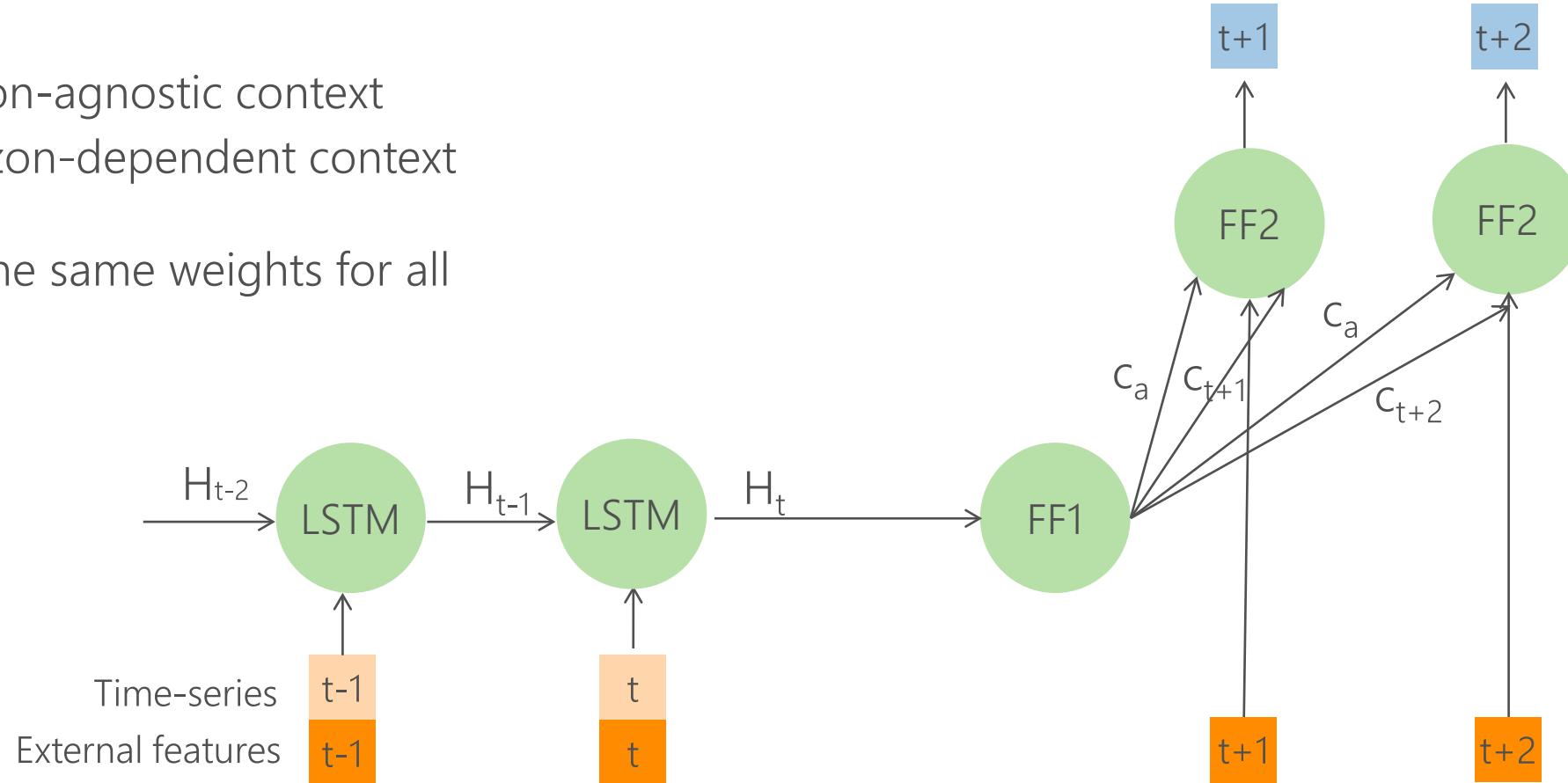
arxiv.org/abs/1711.11053

Winner of GEFCom 2014 competition

Architecture: LSTM encoder + feed-forward neural network

c_a - horizon-agnostic context
 c_{t+i} - horizon-dependent context

FF2 has the same weights for all horizons



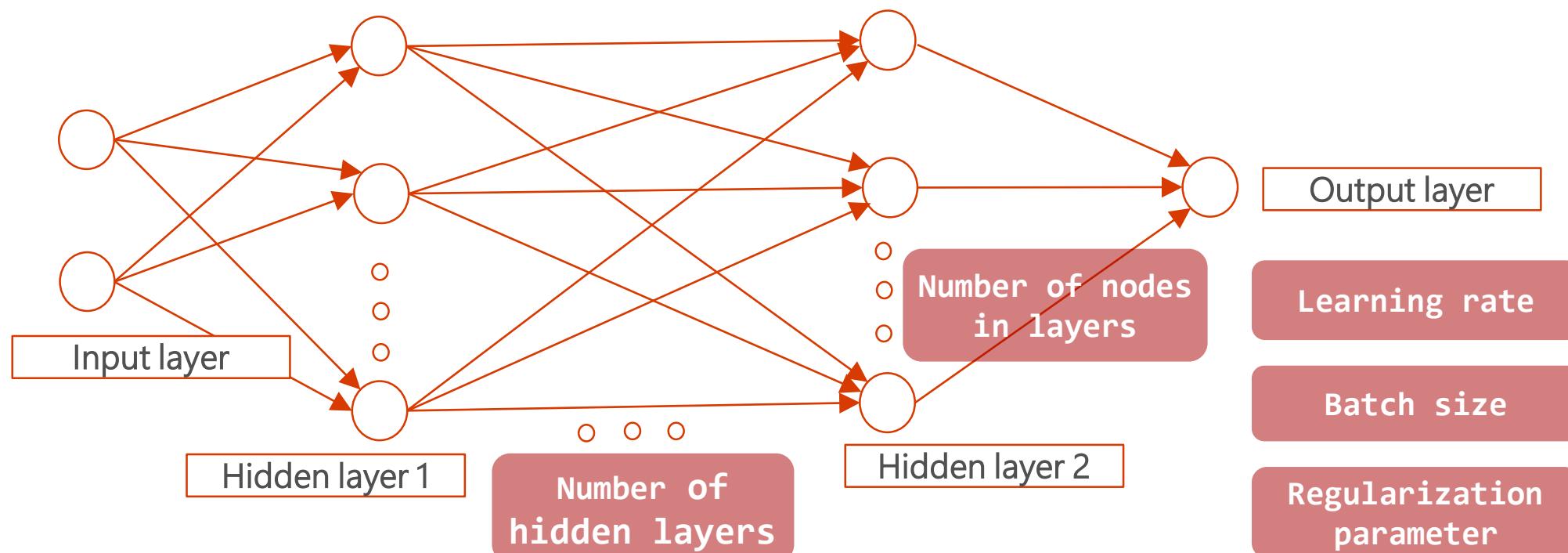
External features: weekday/holiday indicators, day of week, hour of the day, weather data



Hyper-parameter 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.



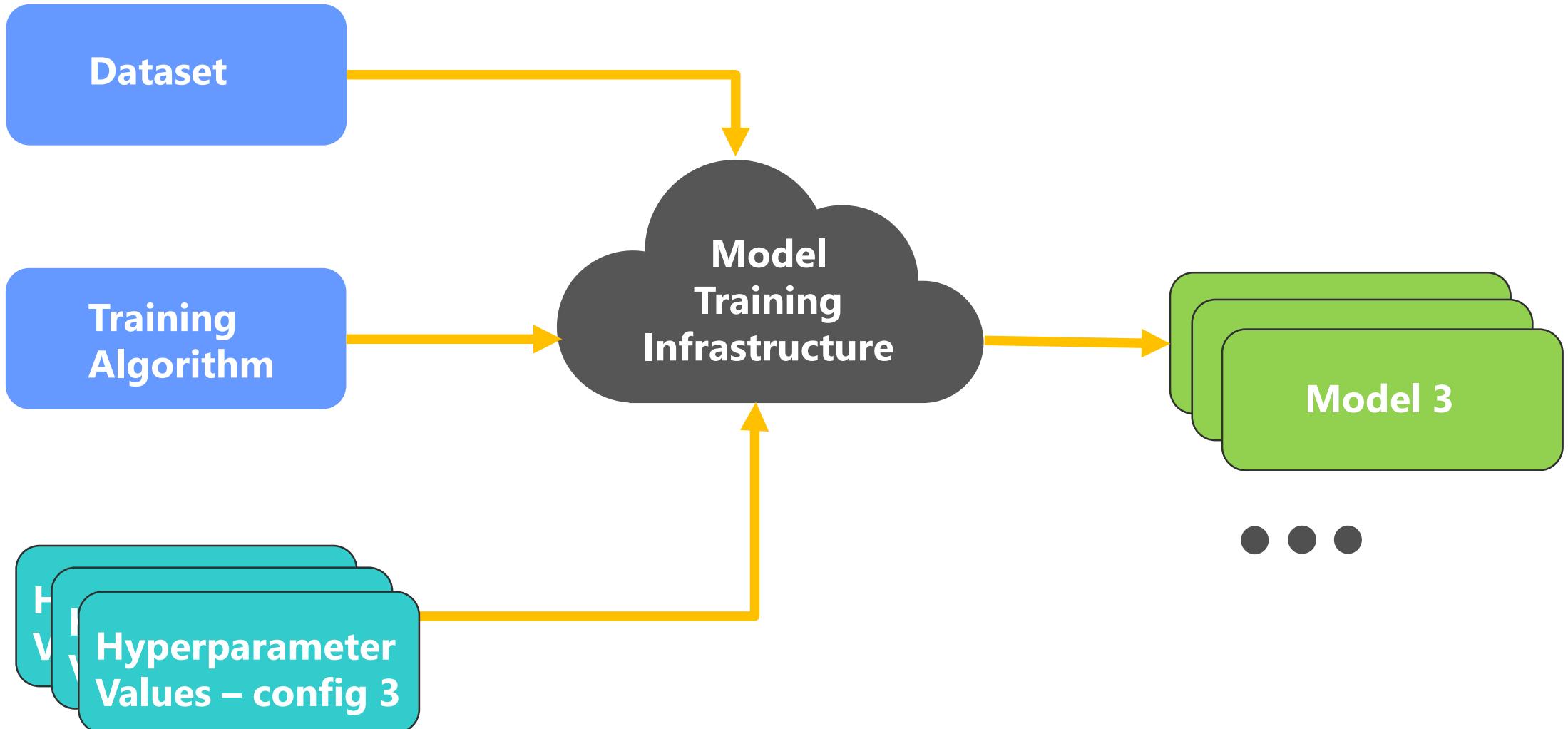
Hyper-parameter tuning

- 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

Typical manual approaches for hyperparameter tuning



Automated Hyperparameter Tuning in Azure ML

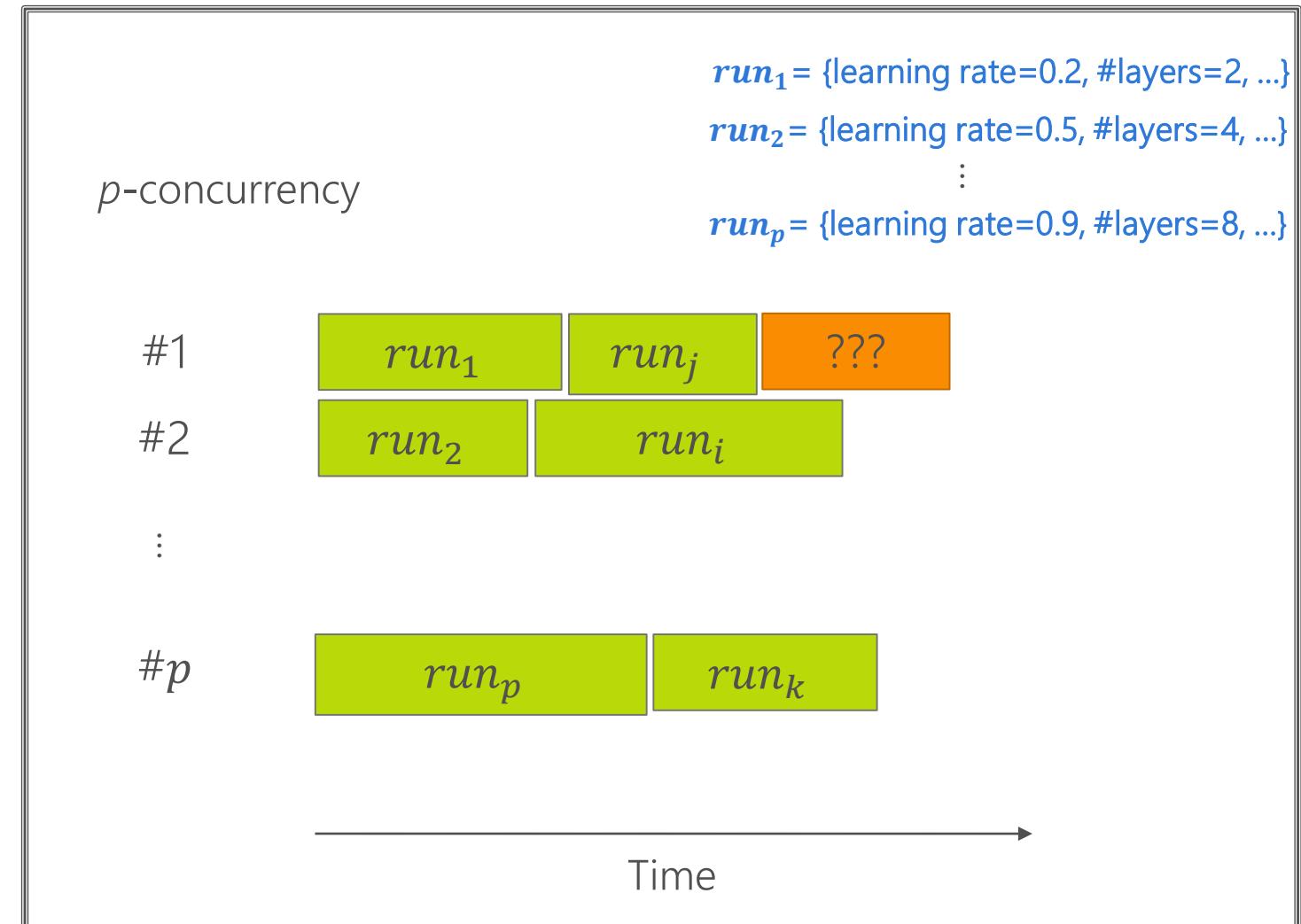
- Automate the exploration process
- Efficient use of resources
- Optimize model for a specified metric
- Offers control over resource budget
- Apply to different models and learning domains
- Training framework agnostic
- Visualize all configurations in one place

Azure Machine Learning Service (Hyperdrive)

Launch multiple parallel training runs

- (A) Generate new runs
- Which parameter configuration to explore?

- (B) Manage resource usage of active runs
- How long to execute a run?



Sampling Methods

- Random Sampling

$$1 - (1 - 0.05)^n > 0.95$$

We get $n \geq 60$. Ta-da!

The moral of the story is: *if at least 5% of the points on the grid yield a close-to-optimal solution, then random search with 60 trials will find that region with high probability.* The condition of the if-statement is

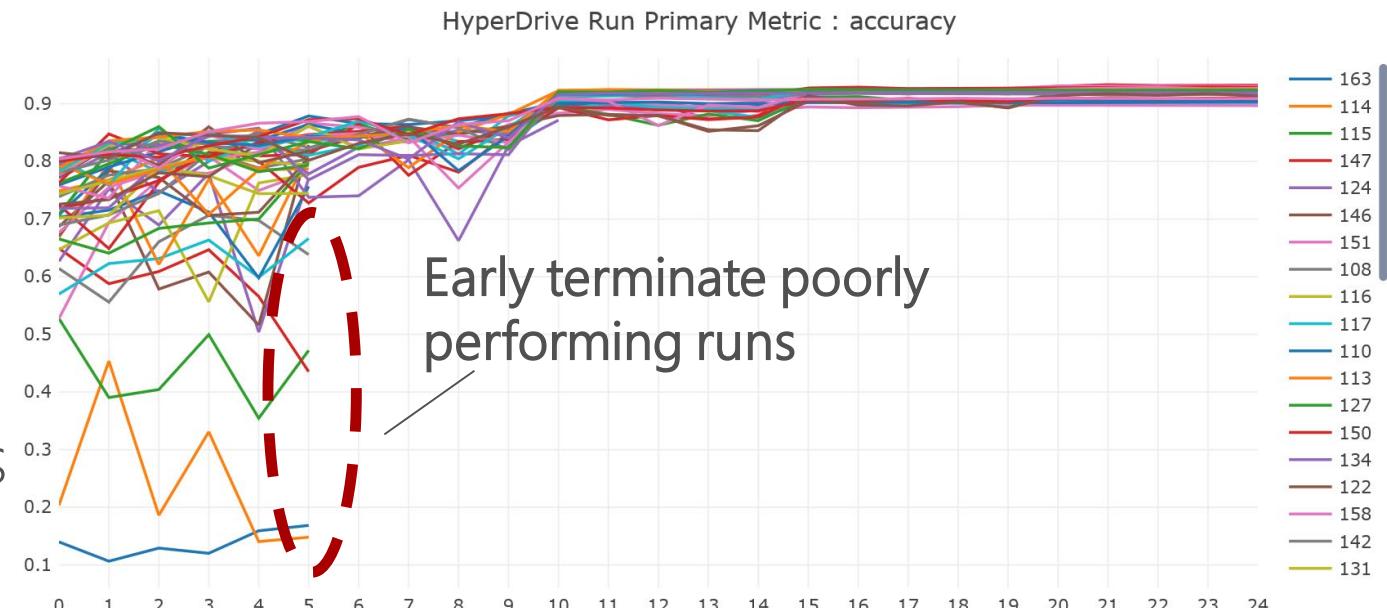
- Grid Sampling
- Bayesian Sampling

Managing Active Jobs

- Evaluate training runs for specified primary metric
- Early terminate poorly performing runs
- Use resources to explore new configurations

Supported early termination policies

- Bandit (slack factor / amount)
- Median stopping
- Truncation selection (lowest % of runs)



User inputs & system outputs

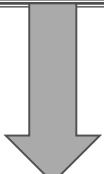
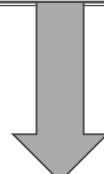
Hyperparameter Tuning Config

- Hyperparameter ranges
- Optimization metric
- Early Termination Policy
- Resource allocation
(`max_total_runs`, `max_duration_minutes`,
`max_concurrent_runs`)



Training script
+
Training data

Recommended
hyperparameter
configuration & run metrics



Hyperparameter Tuning in Azure ML

Azure ML

Define the parameter search space

Specify a primary metric to optimize

Specify early termination criteria for poorly performing runs

Allocate resources for hyperparameter tuning

Launch an experiment with the above configuration

Visualize the training runs

Select the best performing configuration for your model

Tuning hyperparameters in the tutorial

- `hyperparameter_tuning/README.md`
- Python SDK for
 - creating / cancelling
 - obtaining results
 - Elastic compute (auto-scaling)
 - Remote compute can be CPU / GPU cluster
- `hyperparameter_tuning.ipynb` notebook is used to tune several approaches:
 - Feed-forward network multi-step multivariate
 - RNN multi-step
 - RNN teacher forcing
 - RNN encoder decode
 - CNN



Conclusions

Techniques not covered

- Probabilistic prediction [Multi-horizon quantile-based forecaster](#)
- Advanced architectures
 - dropout / zoneout [Zoneout: Regularizing RNNs by randomly preserving hidden activations](#)
 - memory networks [memory networks](#)
- Advanced training techniques
 - state-of-the-art optimization algorithms
 - recurrent batch normalization [recurrent batch normalization](#)
 - stateful training [stateful training](#)
- Temporal convolutional neural networks [An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling](#)

Learning resources

- Source code of examples of how to train deep learning models with Keras
URL: aka.ms/dlts
- Blogs
 - machinelearningmastery.org
 - dacatay.com
- Competitions
 - Kaggle
- State-of-the-art results
 - Conferences: NIPS, ICLR, ICML
 - Hot-off-the-oven papers: arxiv-sanity.org

References

- [1] [Dropout: A simple way to prevent neural networks from overfitting,](#)
- [2] [Zoneout: Regularizing RNNs by randomly preserving hidden activations](#)
- [3] [Batch normalization: Accelerating deep network training by reducing internal covariate shift,](#)
- [4] [Recurrent batch normalization](#)
- [5] [Learning rate schedules and adaptive learning rate methods for deep learning](#)
- [6] [An overview of gradient descent optimization algorithms](#)
- [7] [Learning phrase representations using rnn encoderdecoder for statistical machine translation](#)
- [8] [LSTM: A Search Space Odyssey](#)
- [9] [Bayesian optimization with robust Bayesian neural networks](#)
- [10] [Hyperband: A novel bandit-based approach for hyperparameter optimization](#)
- [11] [Neural architecture search with reinforcement learning](#)
- [12] [Multi-horizon quantile-based forecaster](#)
- [13] [Population based training of neural networks](#)
- [14] [Memory networks](#)
- [15] [Stateful training](#)
- [16] [An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling](#)
- [17] [Winning solution of web traffic forecasting competition](#)
- [18] [Probabilistic energy forecasting: Global Energy Forecasting Competition 2014 and beyond](#)

Key Takeaways

- Deep learning models are very powerful models, sometimes are the most accurate models.
- Neural network models share many techniques/tricks
- Tuning hyperparameters and other tricks are important for creating an accurate model



Q & A

Please provide feedback to help us improve!