Microsoft

Deep Learning for Time Series Forecasting
基于深度学习的时间序列预测

Angus Taylor, Vanja Paunic, Henry Zeng
Yijing Chen, Dmitry Pechyoni

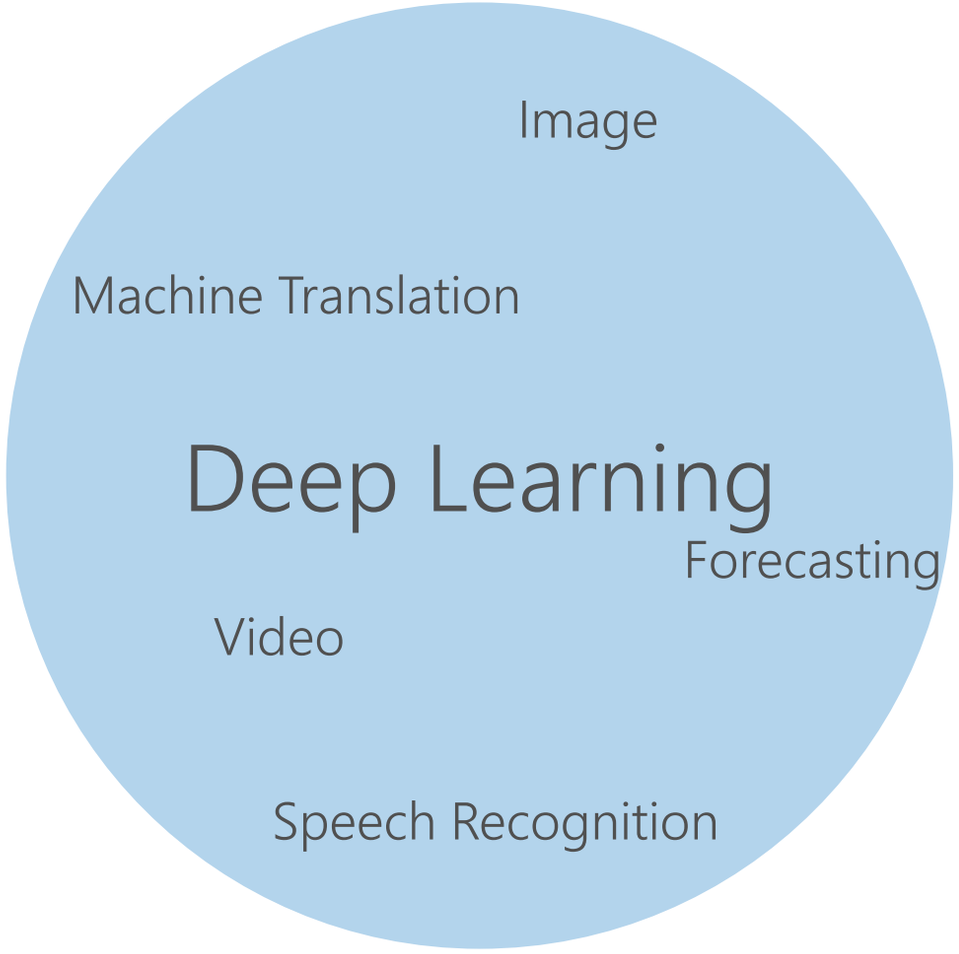
Complete the pre-requisites (请先完成课前要求): aka.ms/dlts/pr

Microsoft

Tutorial Introduction
课程介绍

Tutorial Introduction 课程介绍

This tutorial is about the intersection of DL and Time Series



Tutorial Goals 课程目标

- Understand the basics of recurrent neural networks (RNN) and convolutional neural networks (CNN) for time series forecasting
  理解通过RNN**和**CNN进行时间序列预测的概要

- Know how to use Keras to build CNN and RNN model for time series forecasting
  如何通过Keras**来构建用于**时间预测的CNN**及**RNN**模型**

- Know a number of techniques and tricks that are important for building successful deep-learning-based time series forecasting models
  了解构建时间序列预测深度学习模型的经验和技巧

Target Audience 目标受众

- 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 学习路径

Agenda	Time (mins)	Primary Language
Tutorial Introduction + Pre-requisite Setup 课程介绍 + 课前准备	20	Chinese
Knowledge Recap 基础知识回顾	15	Chinese
Introduction to convolutional neural networks (CNN) 卷积神经网络	30	English
Introduction to recurrent neural networks (RNN) 循环神经网络	30	English
Break 中场休息	10	
Encoder-decoder RNN model 编码-解码 循环神经网络	15	English
Hybrid models for time series forecasting 时间序列预测的混合模型	25	English
Hyper-parameter tuning 超参数调优	20	Chinese
Conclusion 课程总结	5	Chinese
Q&A 问答环节	10	

- Hands-on lab and quizzes
- Q&A during break or in the end
- This tutorial will be delivered in English and Chinese

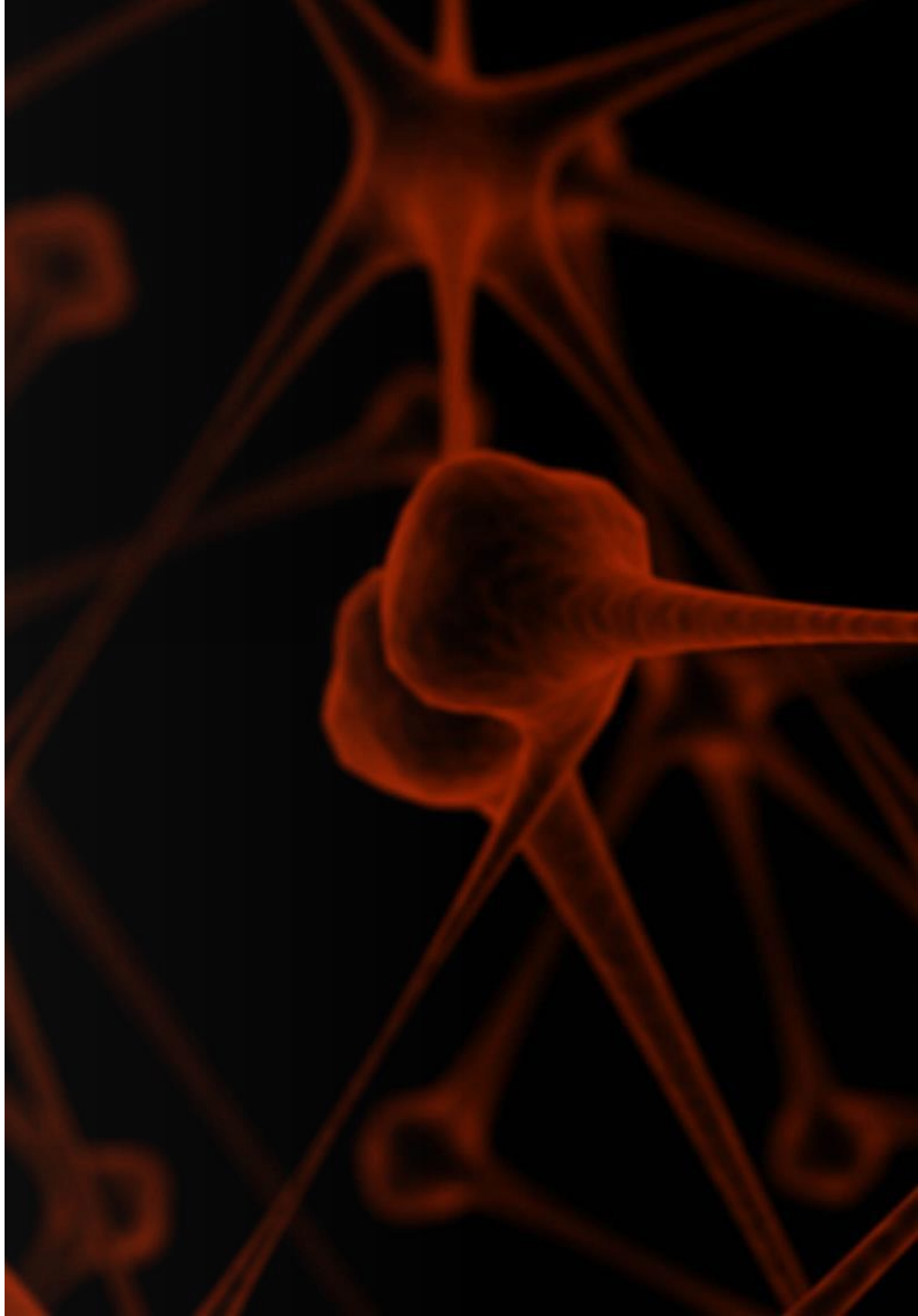
Hands-on Lab 动手实验

- Form a team of 2-3 组建2-3**人小**团队
- Help your teammates **帮助**团队成员完成实验
- 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?

Microsoft

Pre-requisite Setup
课前准备

aka.ms/dlts/pr



Why Keras?

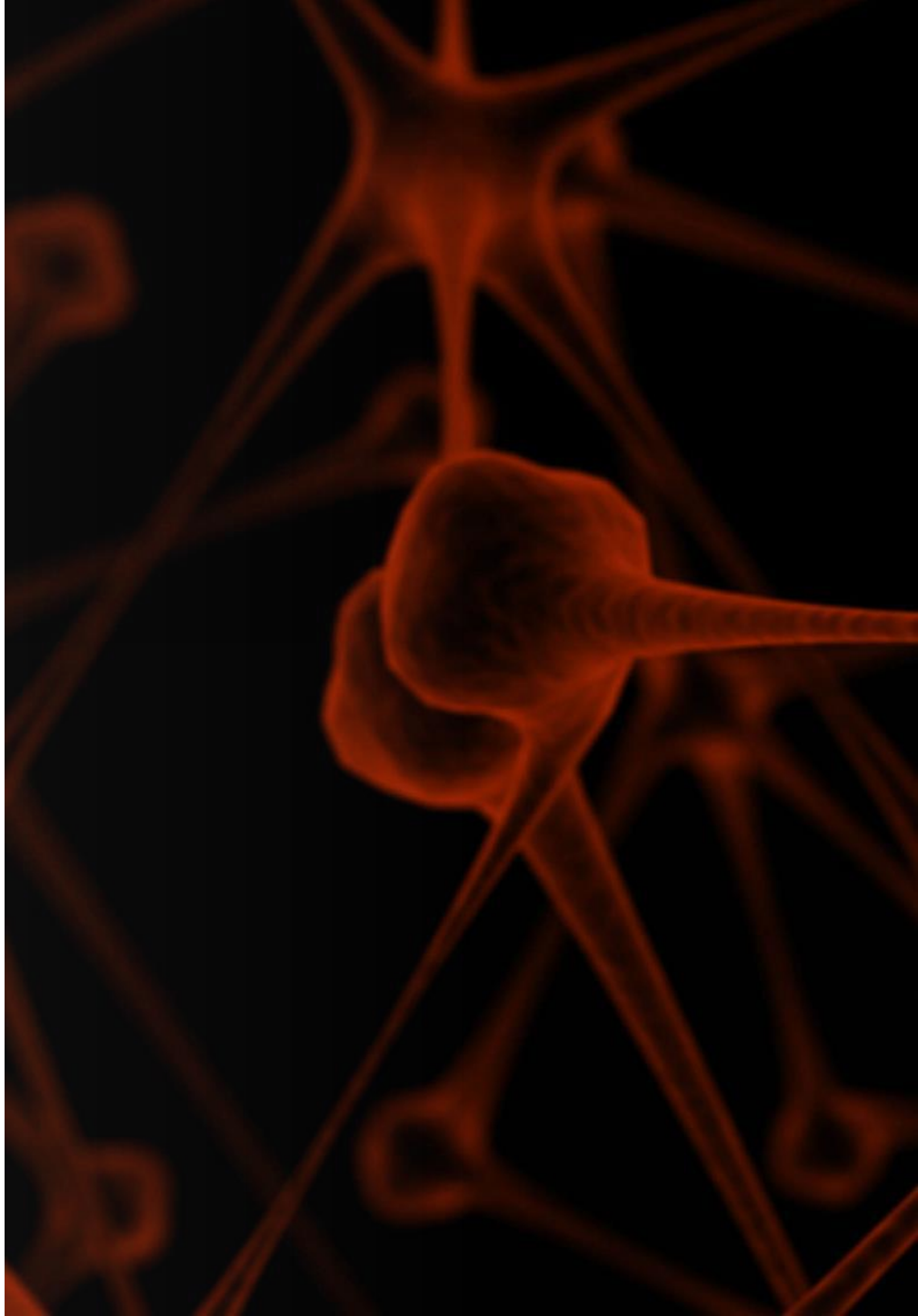
- High-level API for deep learning
- Simplified syntax for common DL tasks
- Operates over efficient backends (e.g. TensorFlow, CNTK)
- Less control over low-level operations
- Sometimes not as efficient as other frameworks

Microsoft

Knowledge Recap
基础知识回顾

Microsoft

Time Series Forecasting
时间序列预测



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
History	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
Future	2018-06-13	?
	2018-06-14	?
	2018-06-15	?

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?

……

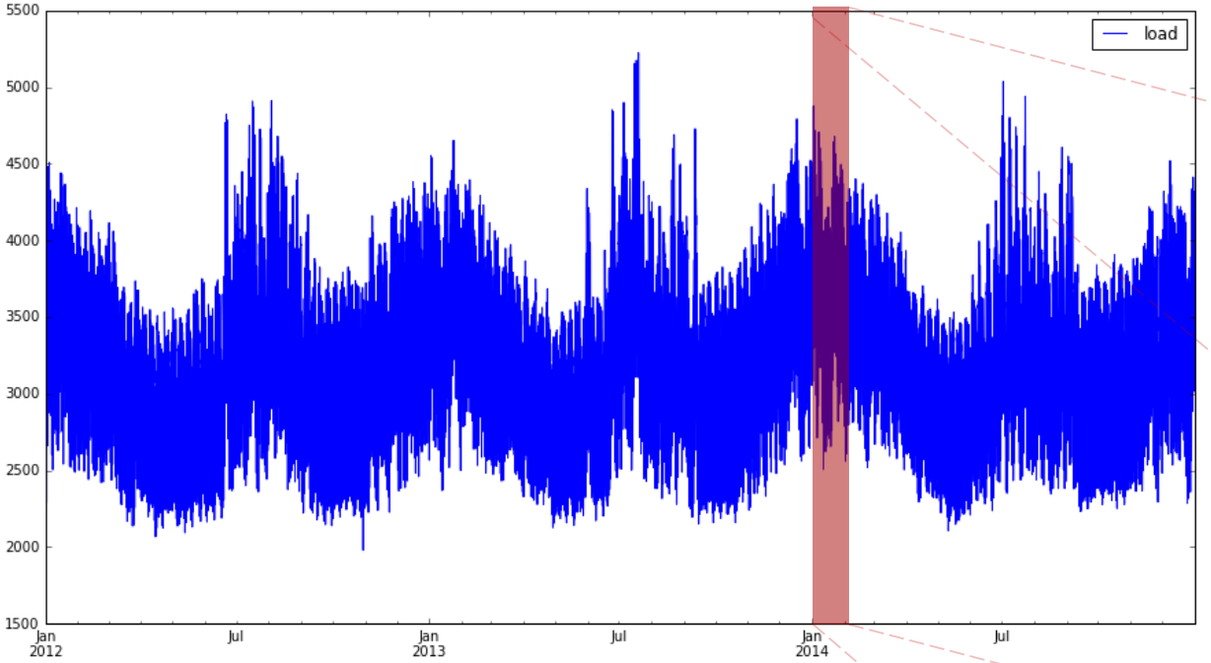
……

- What technique/model should I use?

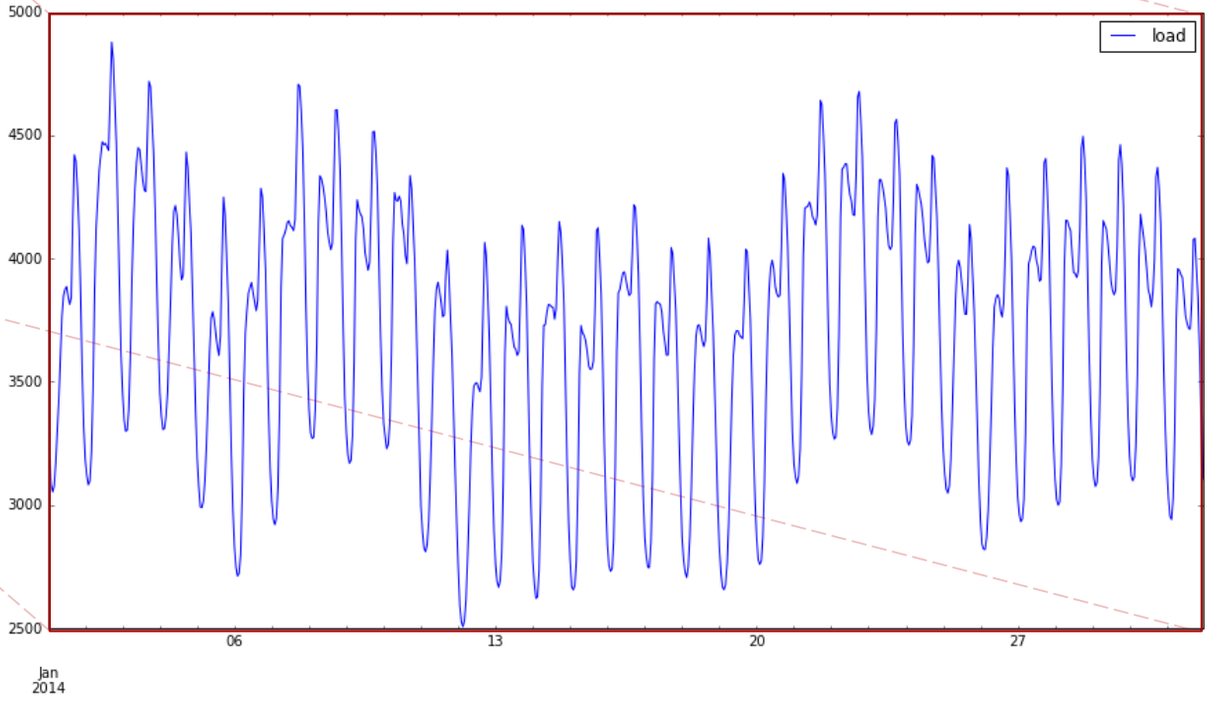
Scenario: energy load forecasting

- 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



- 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.

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)

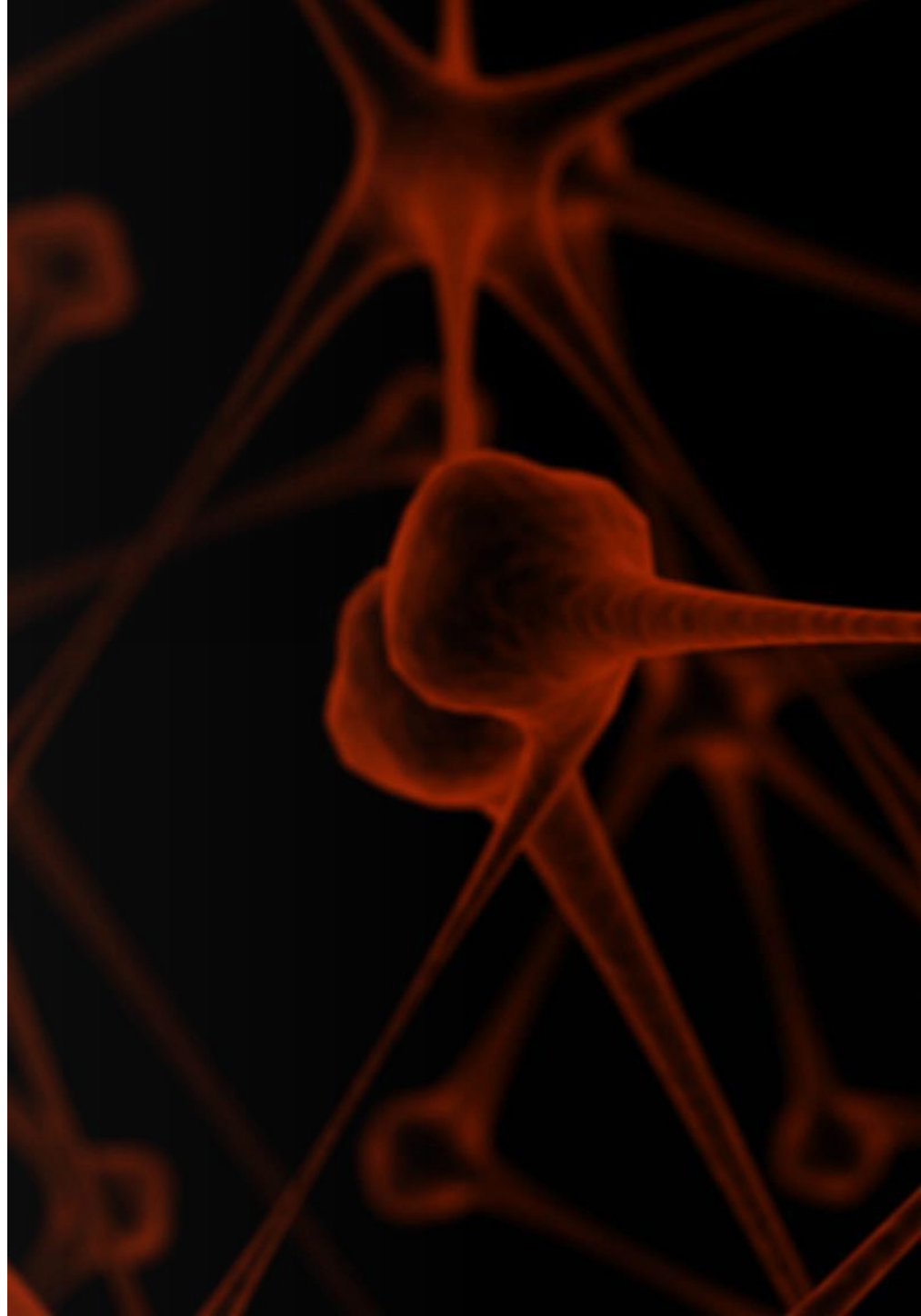
- Non-parametric, greatly simplifies learning

- 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

Microsoft

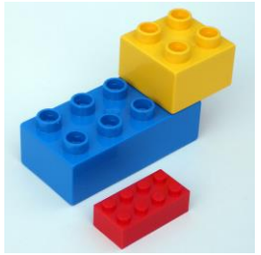
Feedforward Neural Networks Recap
前馈神经网络(FNN)概要

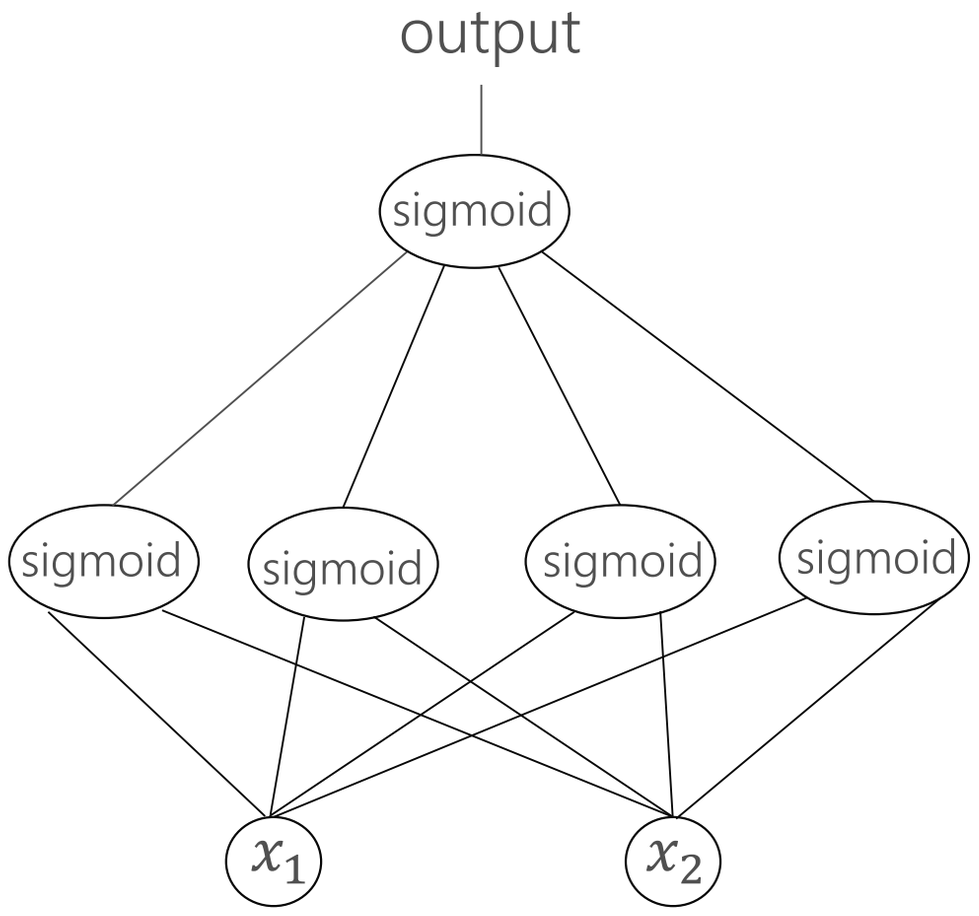


Agenda

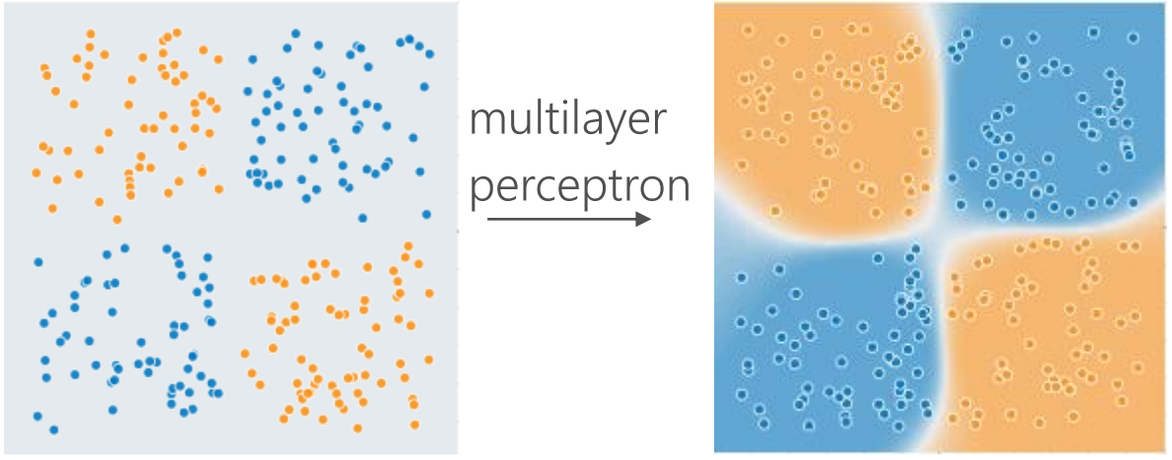
- Architecture of Feed-forward Neural Network
- Feed-forward Neural Network Training
  - Loss function
  - Gradient descent (stochastic, batch, mini-batch gradient descent)
  - Backpropagation

Multilayer Perceptron





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



Input

Output

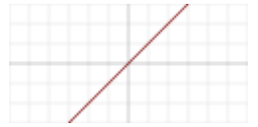
also known as Feed-Forward Neural Network and Deep Neural Network

Activation Functions

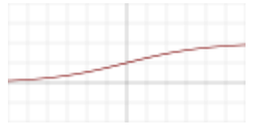
https://en.wikipedia.org/wiki/Activation_function

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

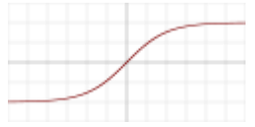
Identity $f(x') = x'$



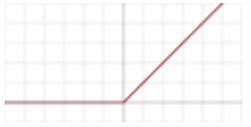
Sigmoid $f(x') = \sigma(x') = \frac{1}{1+e^{-x'}}$



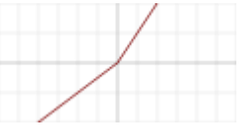
tanh (hyperbolic tangent) $f(x') = \frac{e^{x'}-e^{-x'}}{e^{x'}+e^{-x'}}$



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) $f(x') = \begin{cases} \alpha x' \text{ for } x' < 0 \\ x' \text{ for } x' \geq 0 \end{cases}$

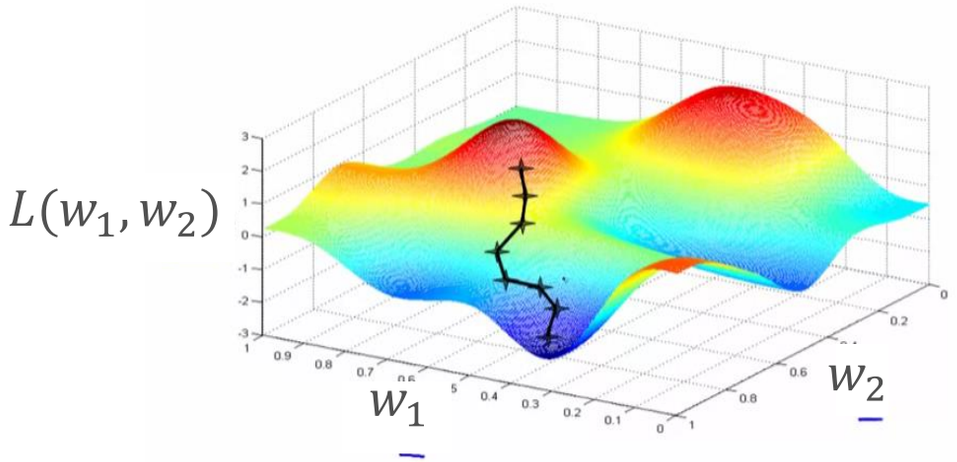


Training: overview

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

Main components:

- Loss function $L(weights, biases)$ 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): $\frac{100}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2}$

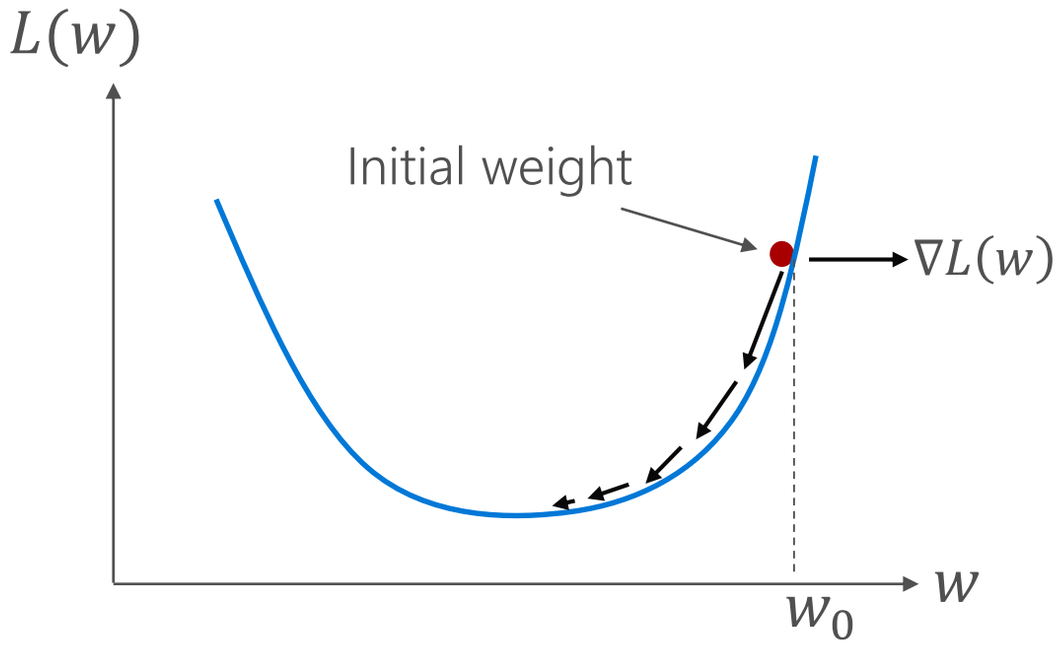
$\hat{y}_i$ is a function of weights w and biases b.

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

Optimization algorithm: (Batch) gradient descent

Gradient $\nabla L(w) = \left(\frac{\partial L(w)}{\partial w_1}, \frac{\partial L(w)}{\partial w_2}, \ldots, \frac{\partial L(w)}{\partial w_d}\right)$ - 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

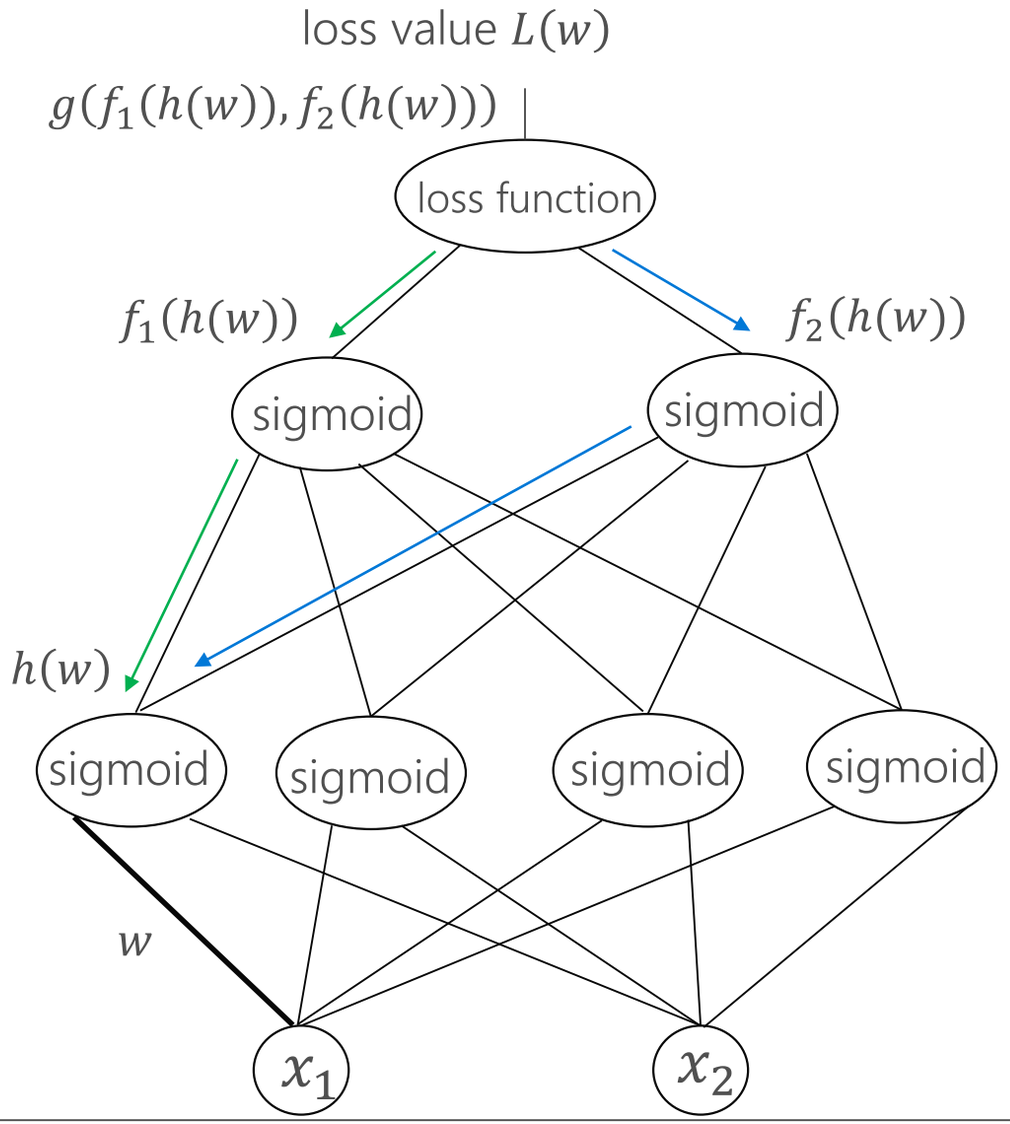
$$w = w - \frac{\alpha}{m}\sum_{i=1}^{m} \nabla L_i(w)$$

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

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

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

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

Total derivative

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

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

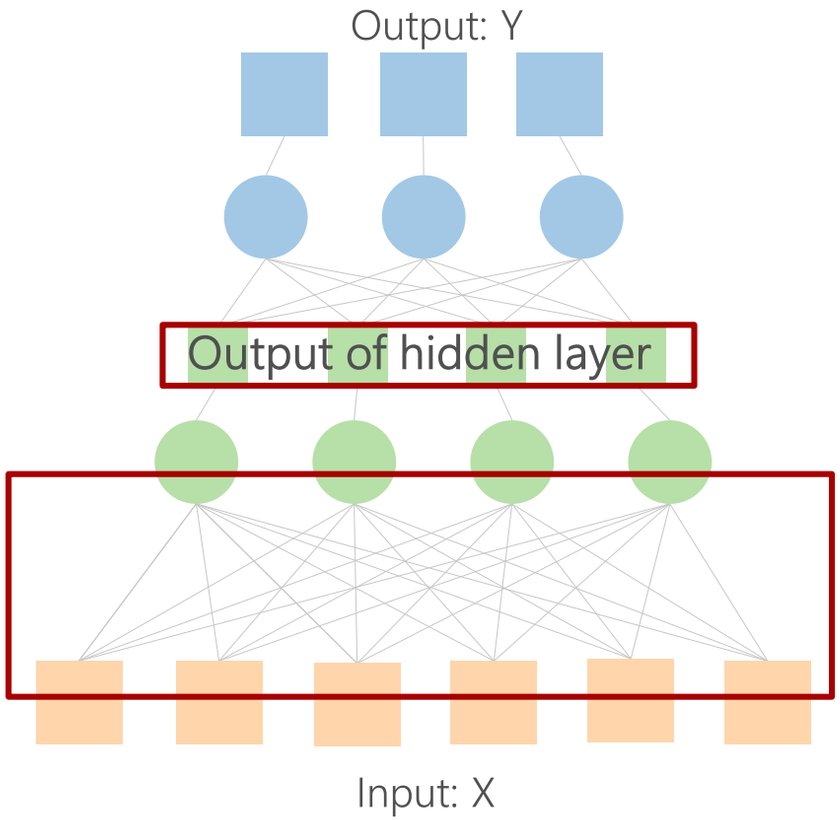
Training tricks 训练技巧

- 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

Microsoft

Introduction to Convolutional Neural Networks
卷积神经网络(CNN)介绍

Fully connected vs convolution layer



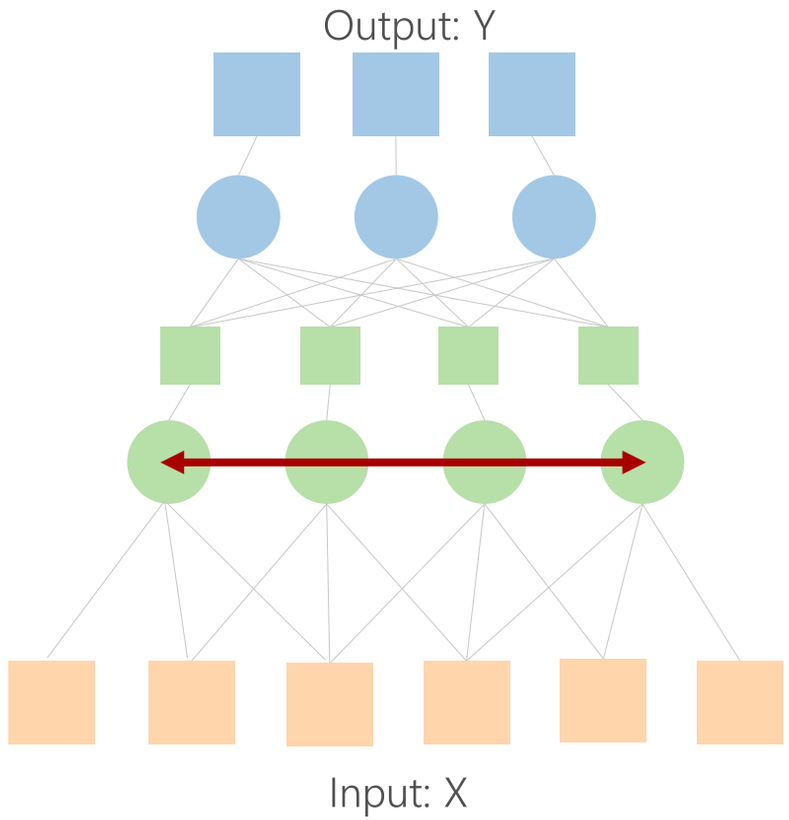
Fully connected:

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

Fully connected vs convolution layer

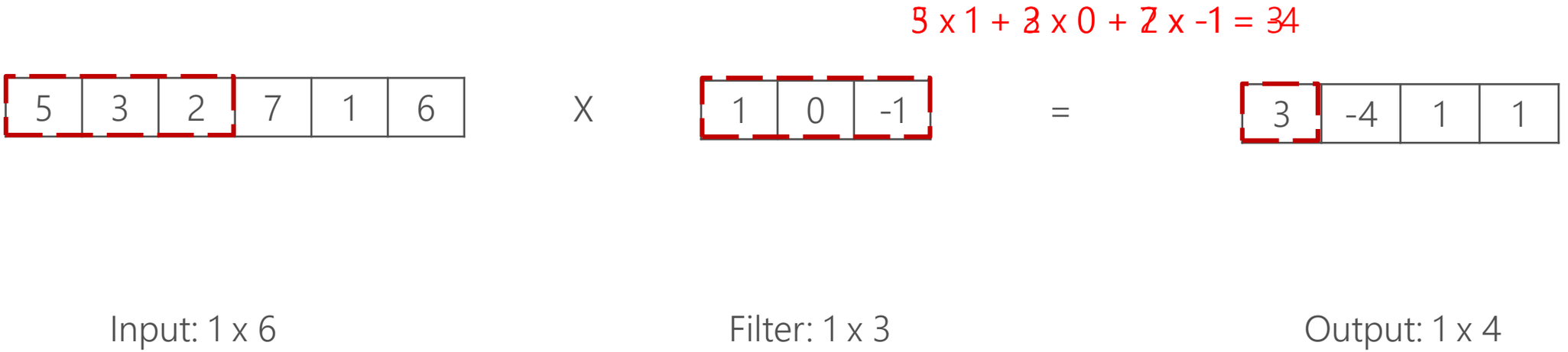
Convolution layer:

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



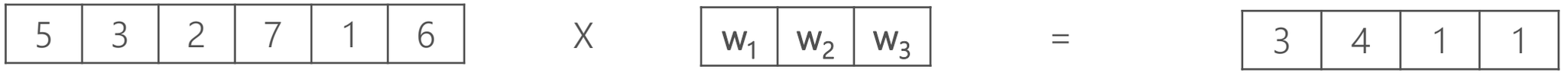
1D Convolutions

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



1D Convolutions

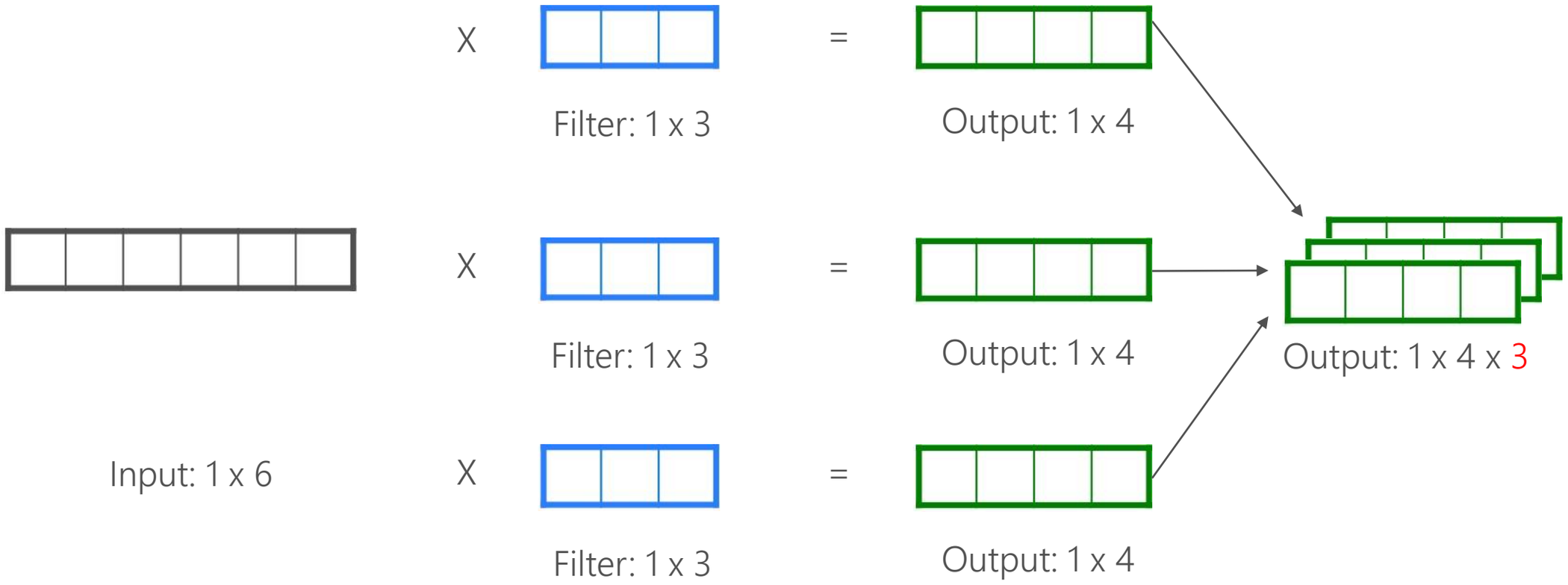
- Filters are trained to detect features in the input sequence
- Features can be detected regardless of where they appear in the input sequence



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

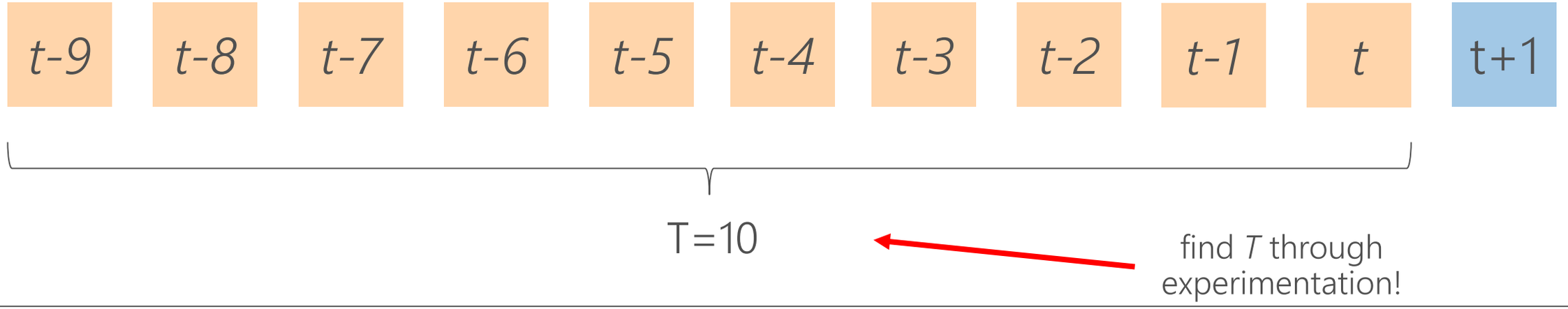
1D Convolutions

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



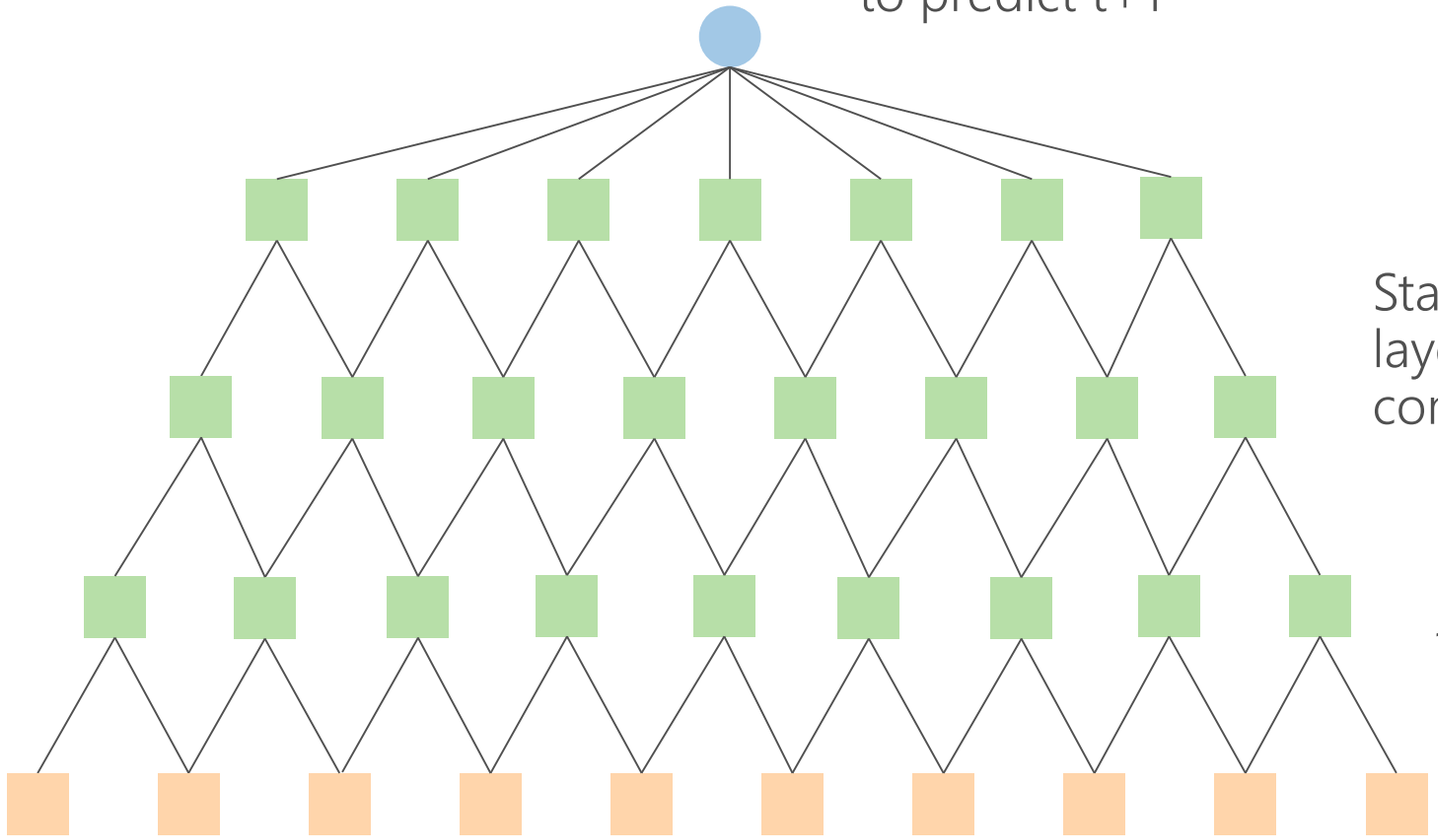
Application to forecasting

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



CNN for forecasting

Fully-connected layer with one unit to predict t+1



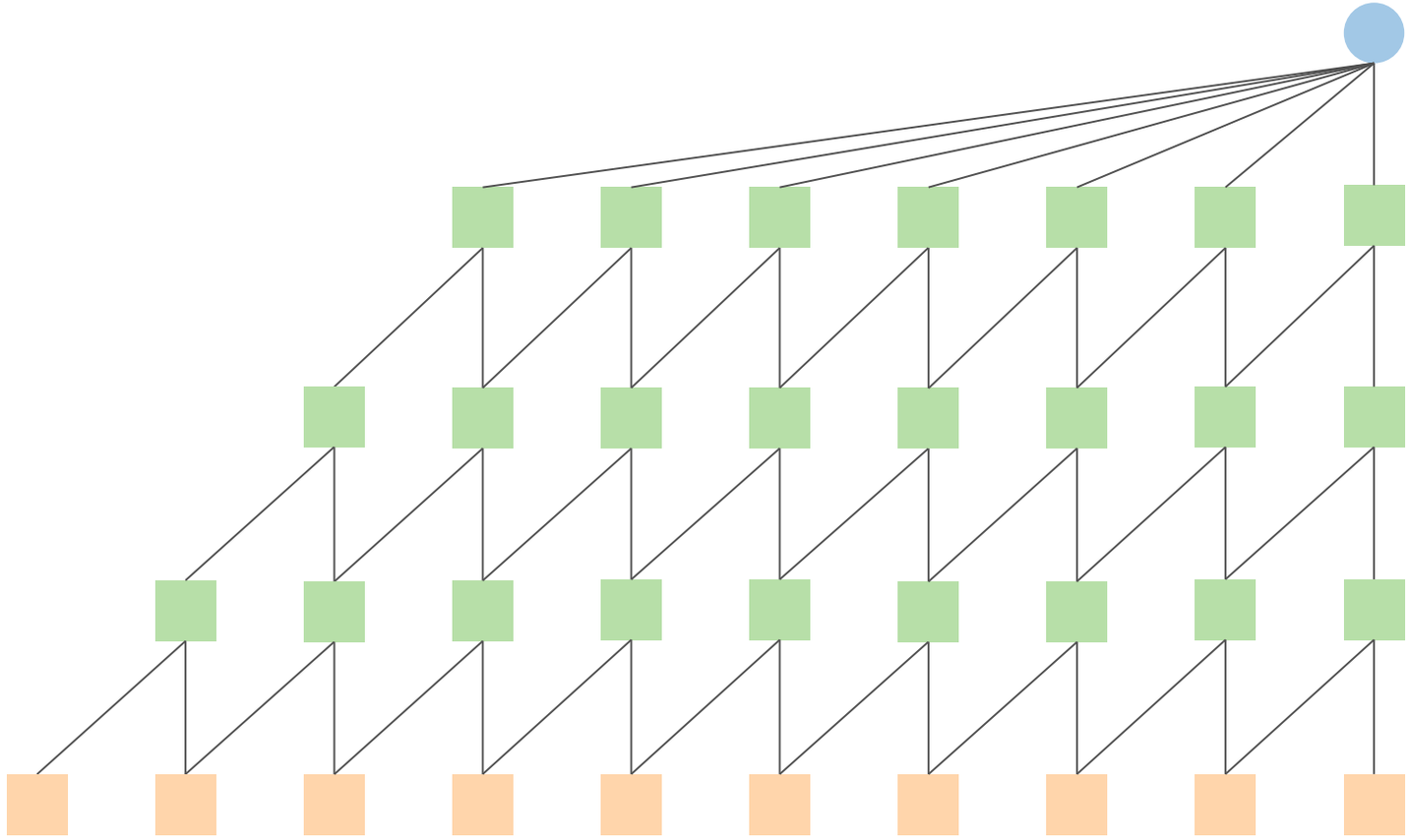
Stacked convolution layers to detect feature combinations

1D convolution layers, kernel width 2

Inputs from t-9 to t

Causal convolutions

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



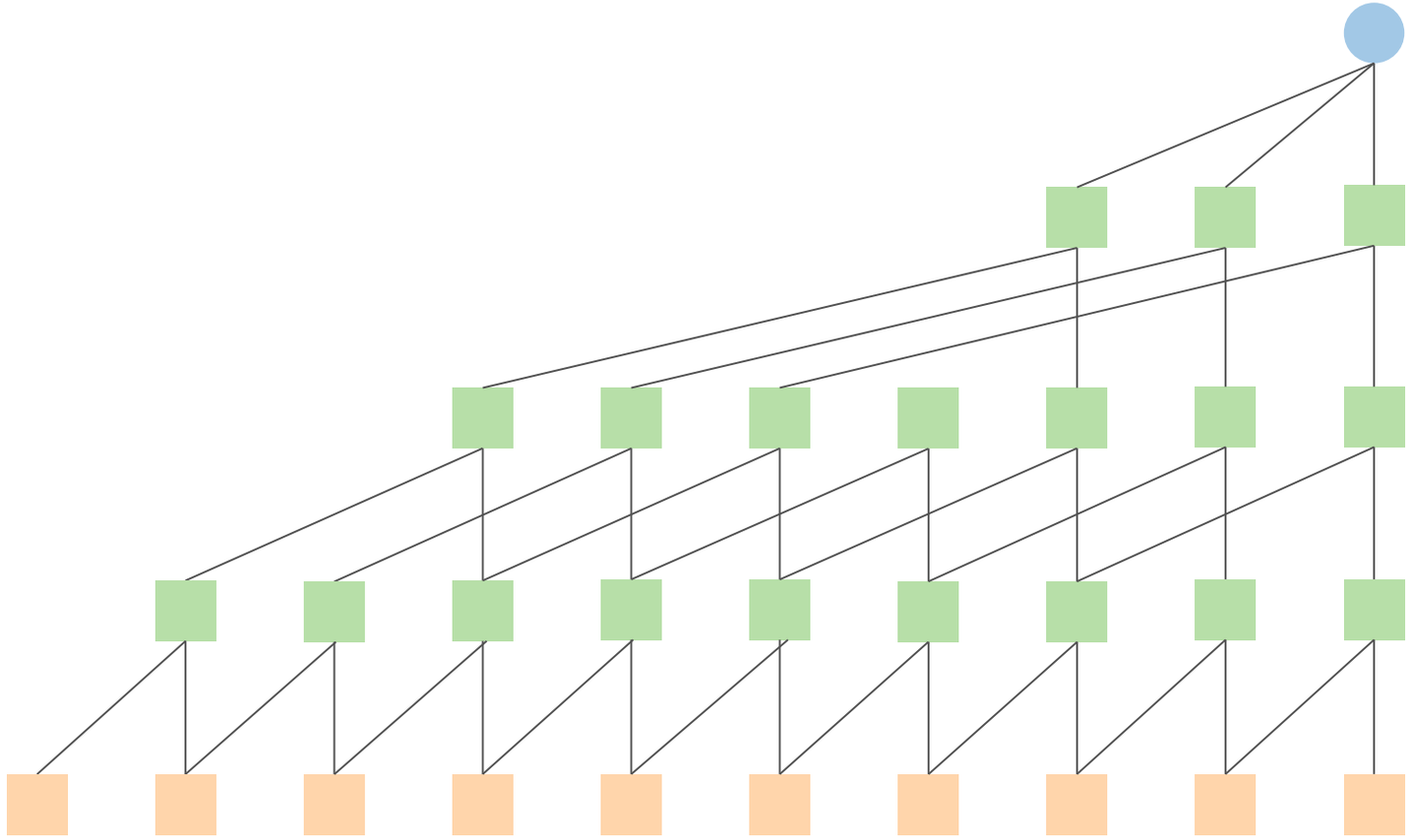
Dilated convolutions 空洞卷积

- Skip outputs from previous layers

Dilation rate 4

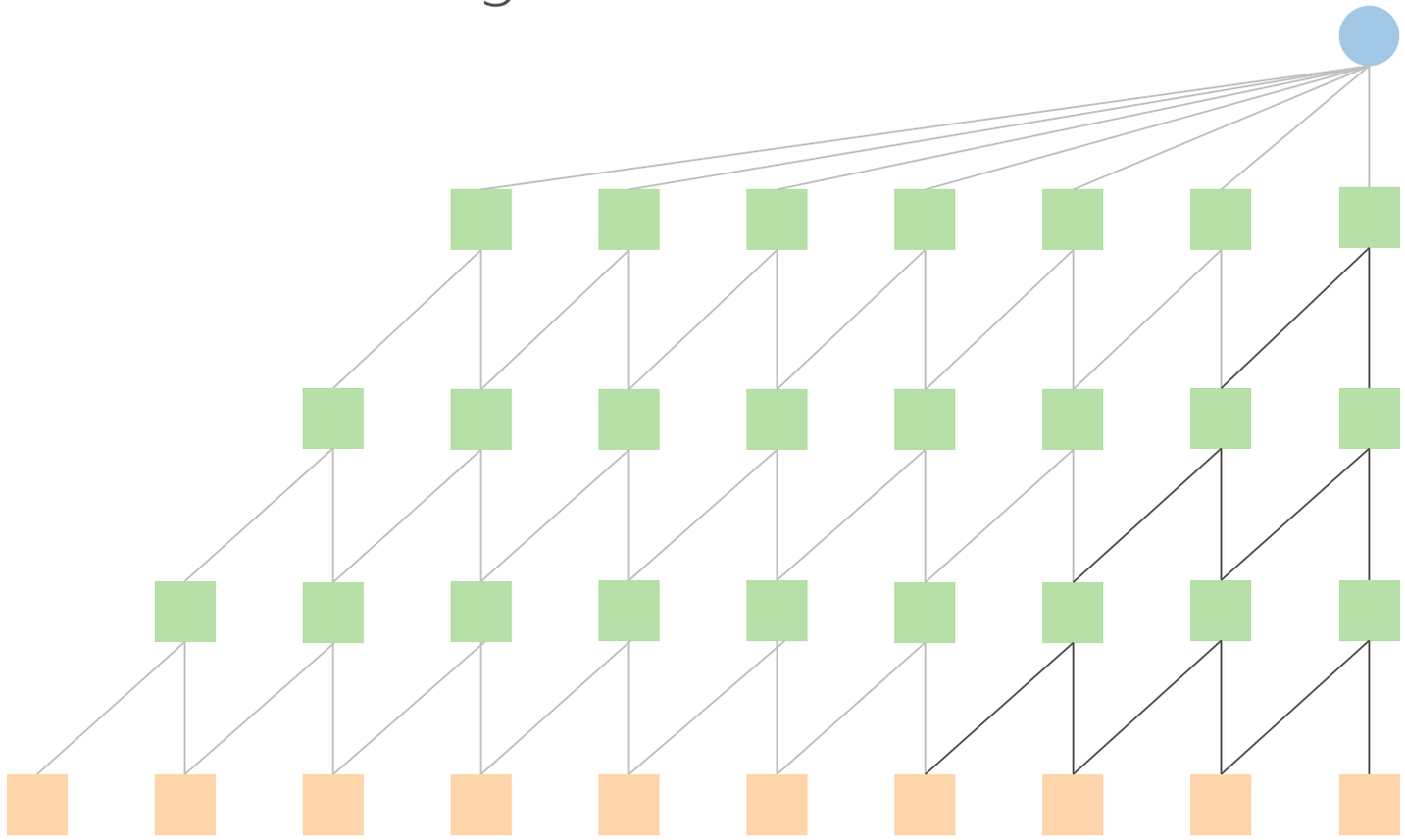
Dilation rate 2

Dilation rate 1



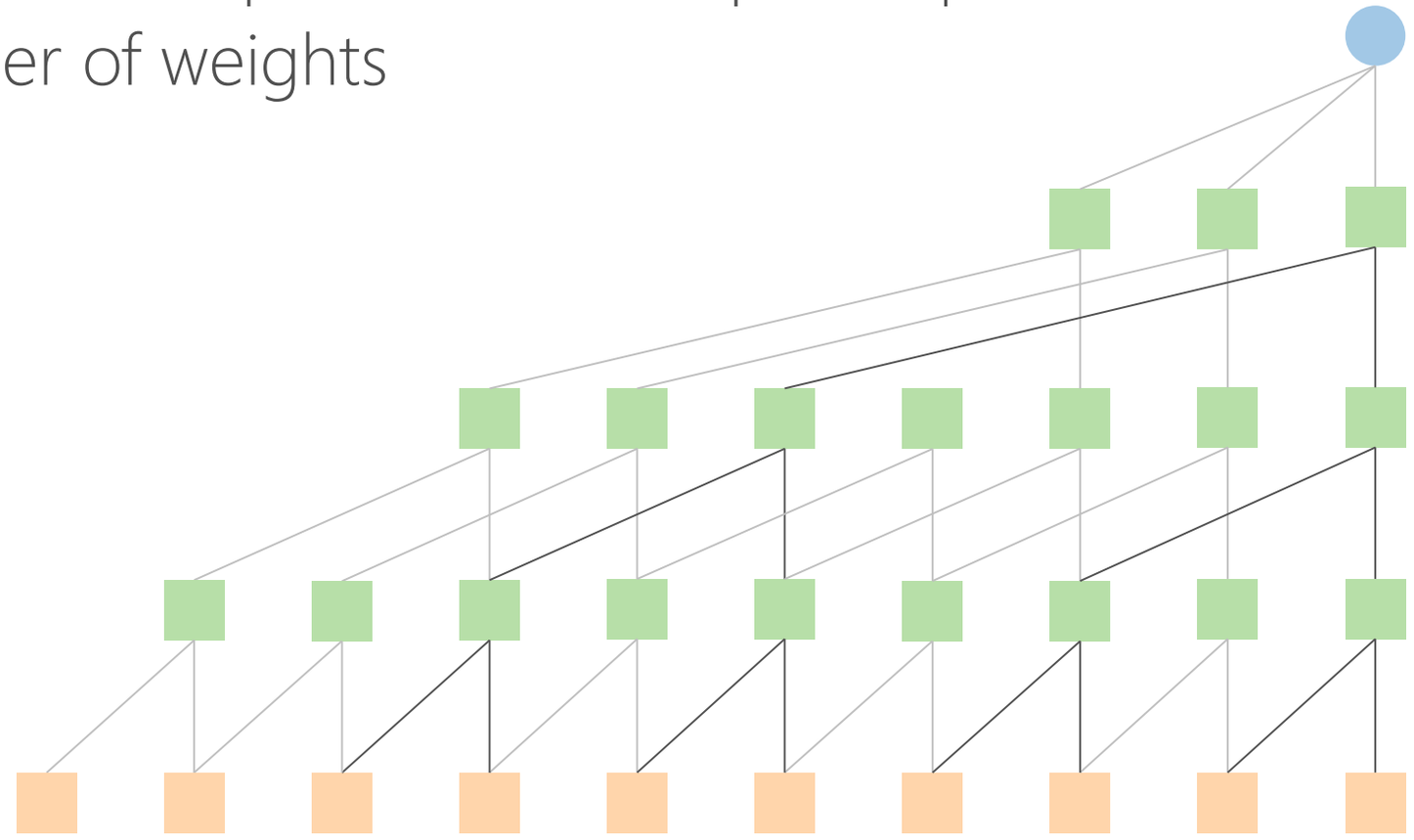
Why dilated convolutions?

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



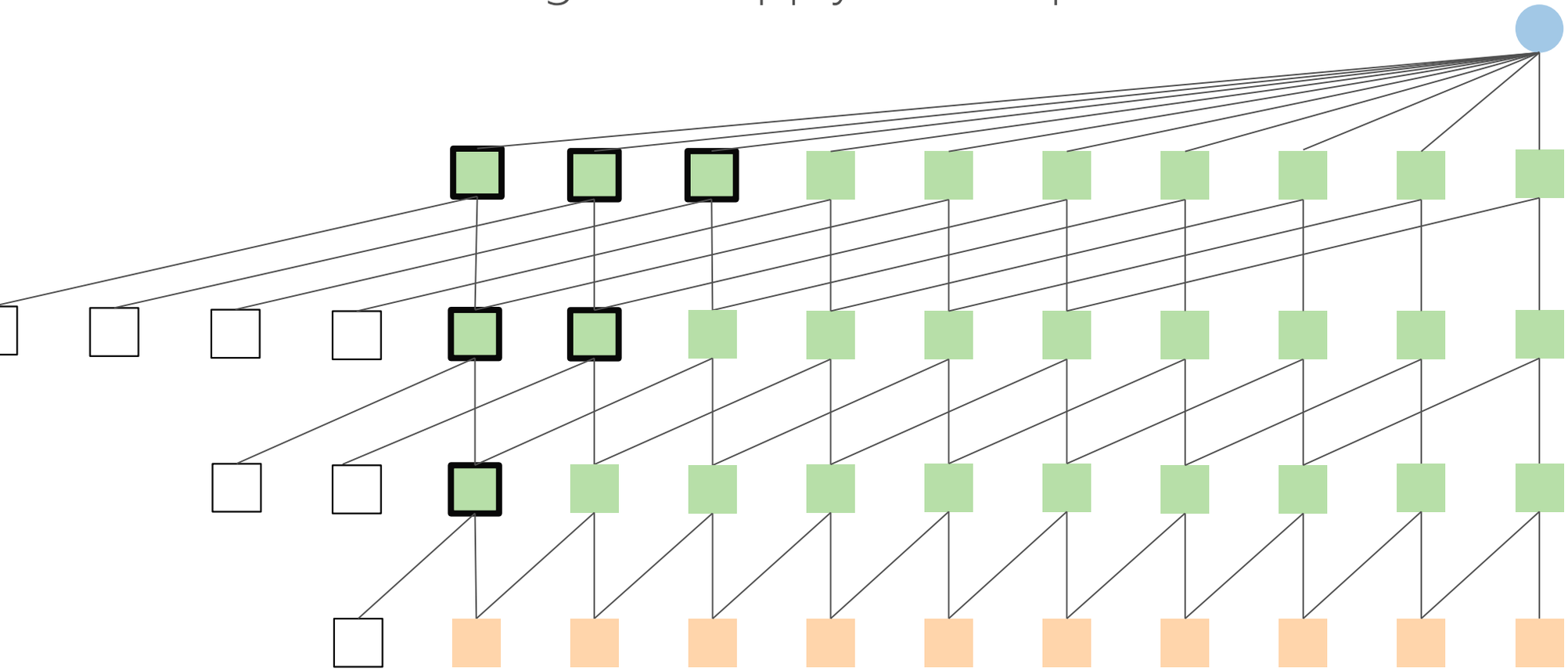
Why dilated convolutions?

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



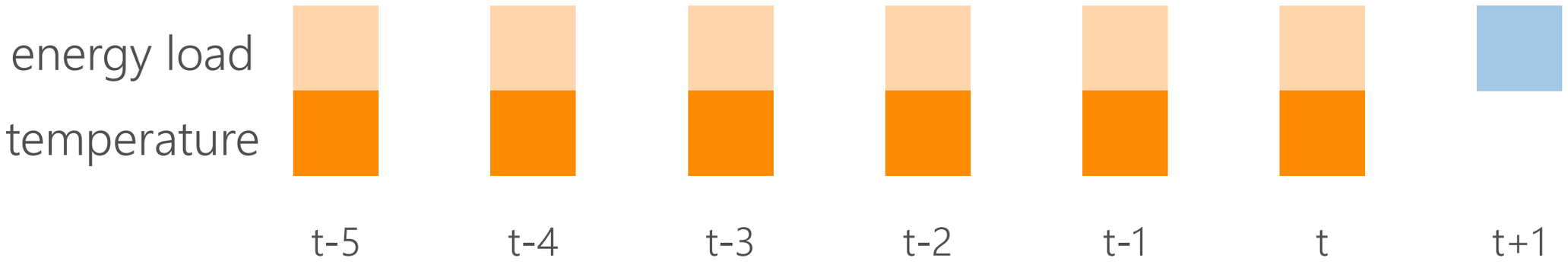
Causal padding

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



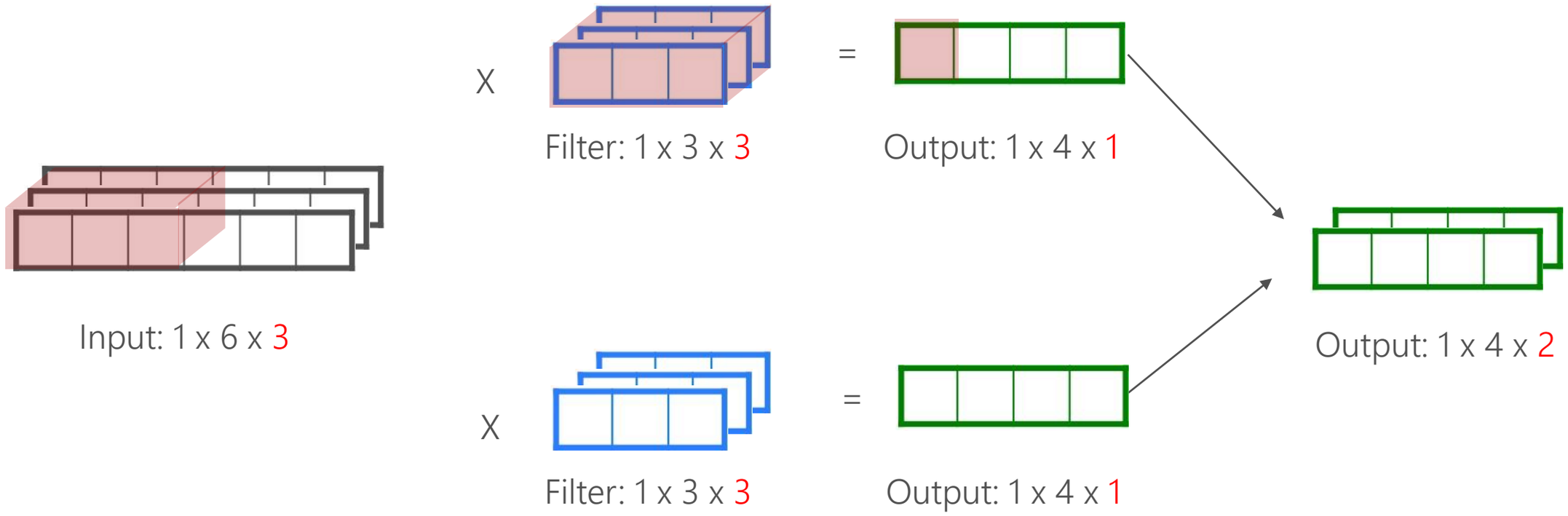
Multivariate time series
多变量时间预测

- Multiple time series in the input sequence



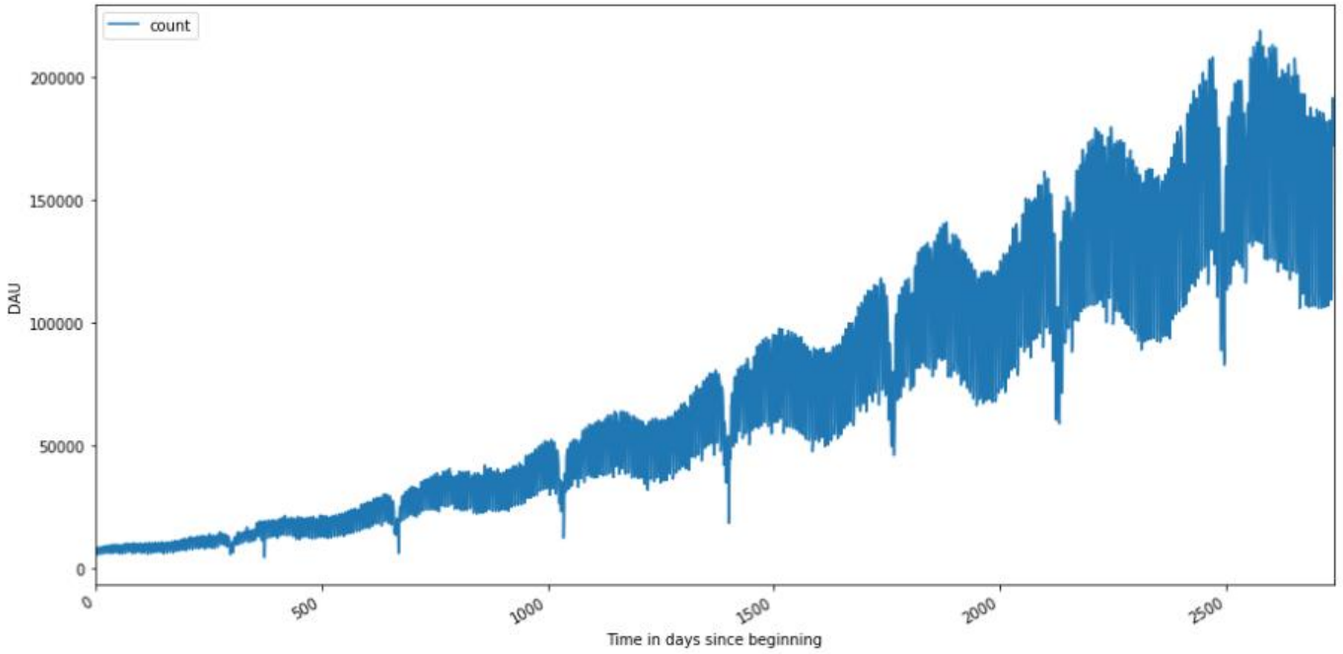
Multivariate time series
多变量时间预测

- Example: for 3 input time series use 3 channels



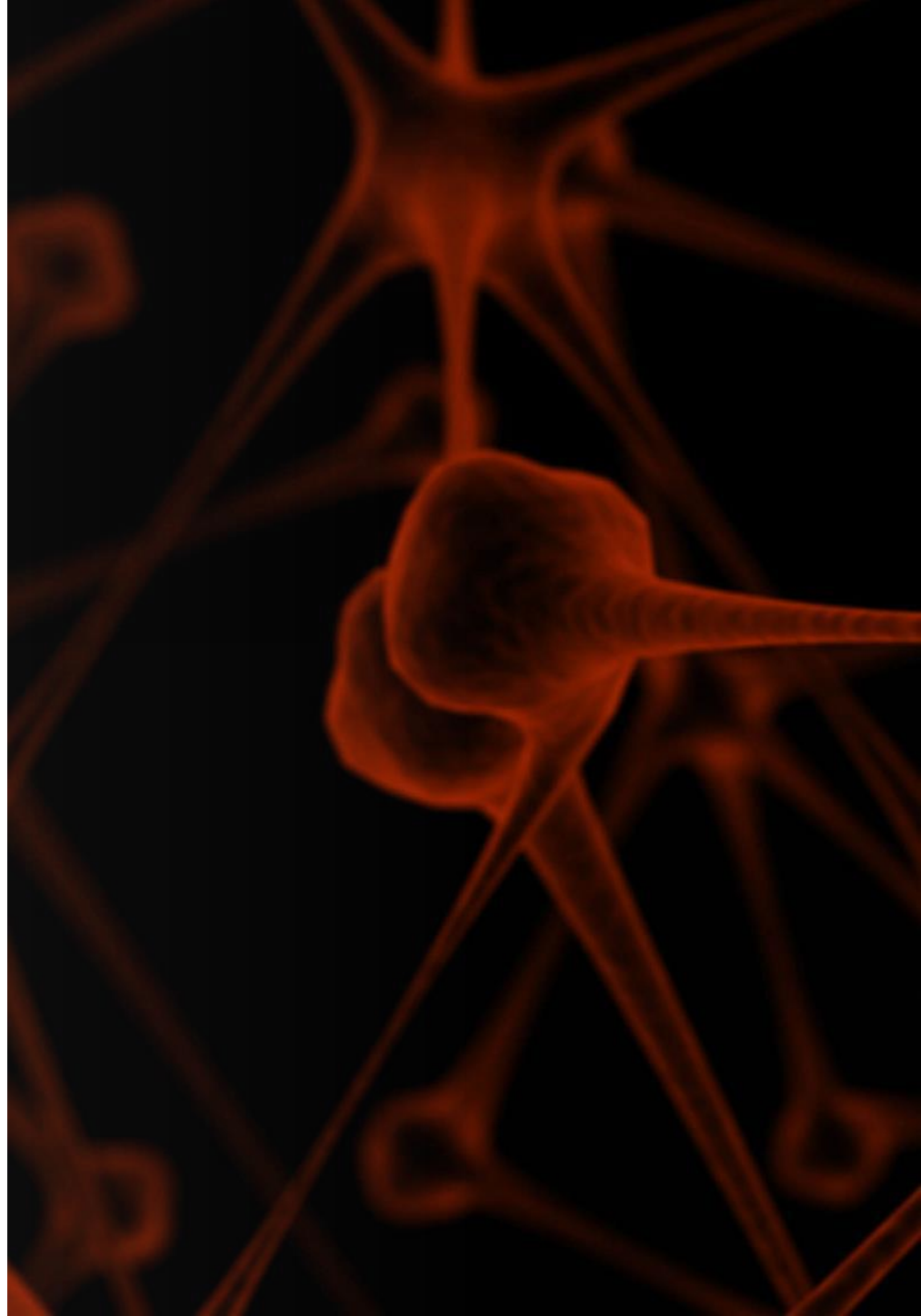
Advanced CNN models for forecasting

- 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



Microsoft

动手实验
1_CNN_dilated.ipynb notebook



Microsoft

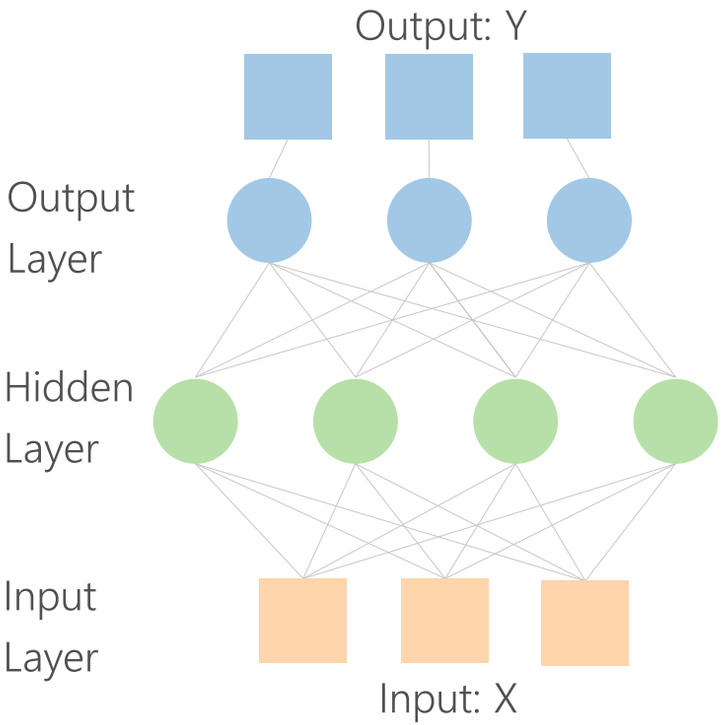
Introduction to Recurrent Neural Networks
循环神经网络(RNN)介绍

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?

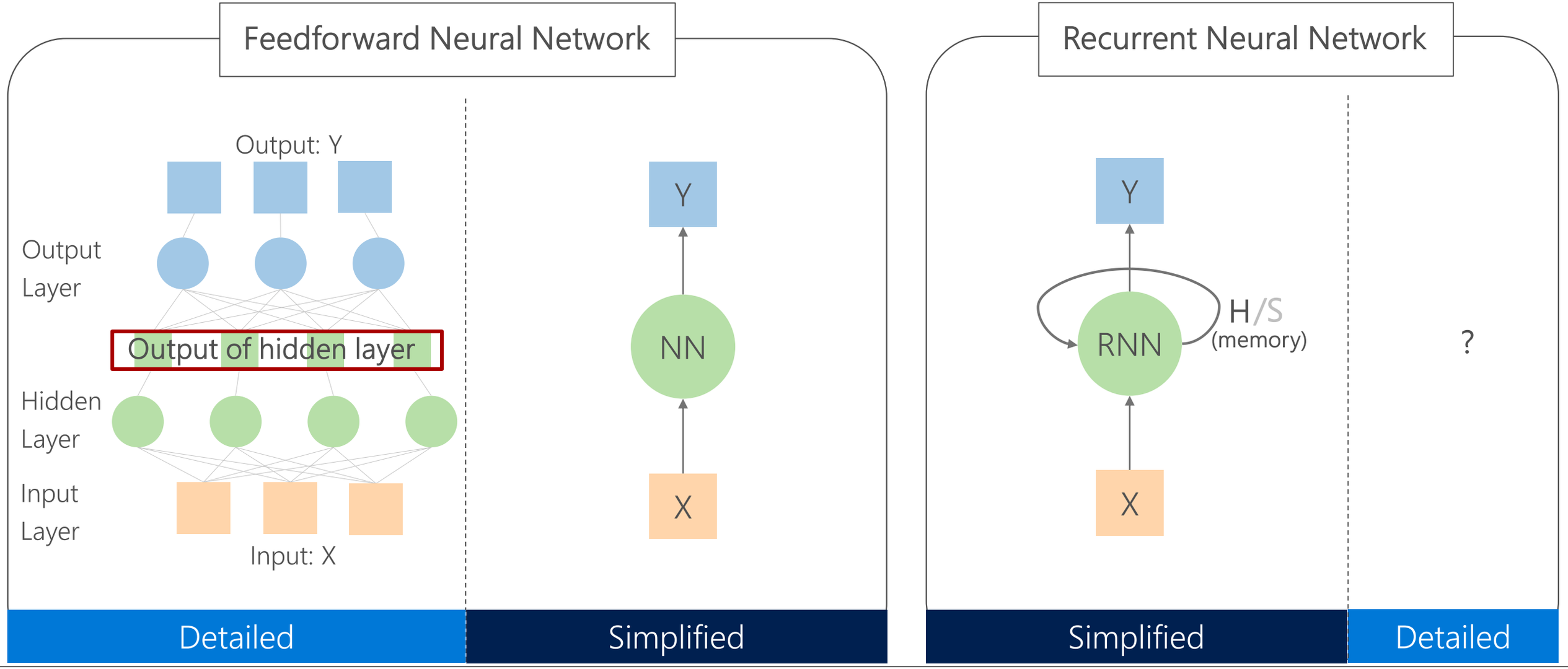
Feedforward Neural Network



Detailed

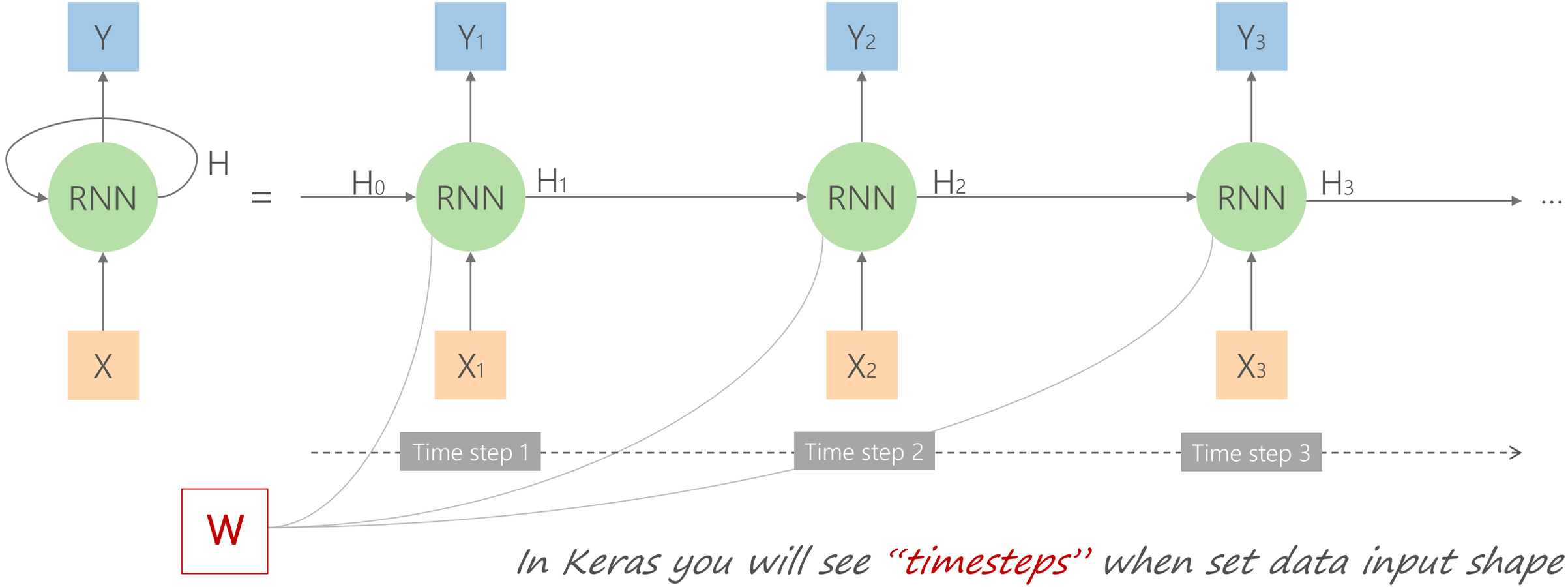
What are RNNs?

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



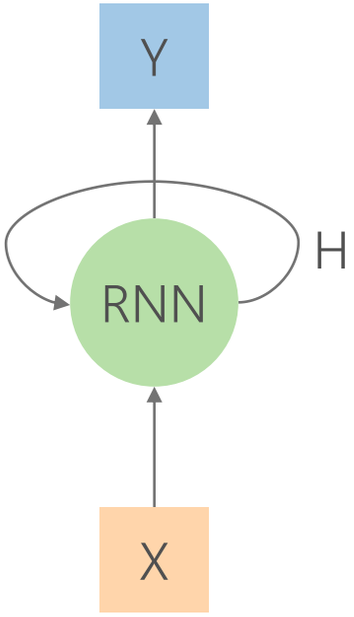
Unrolled RNN

The same weight and bias shared across all the steps



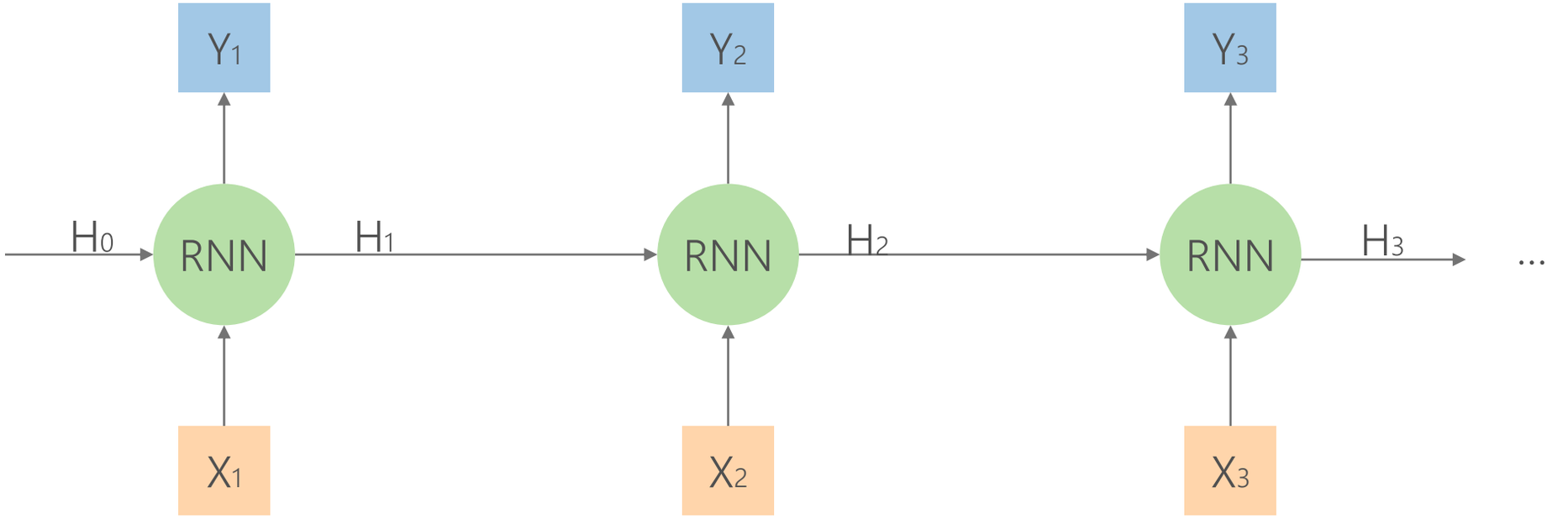
In Keras you will see "timesteps" when set data input shape

Unrolled RNN

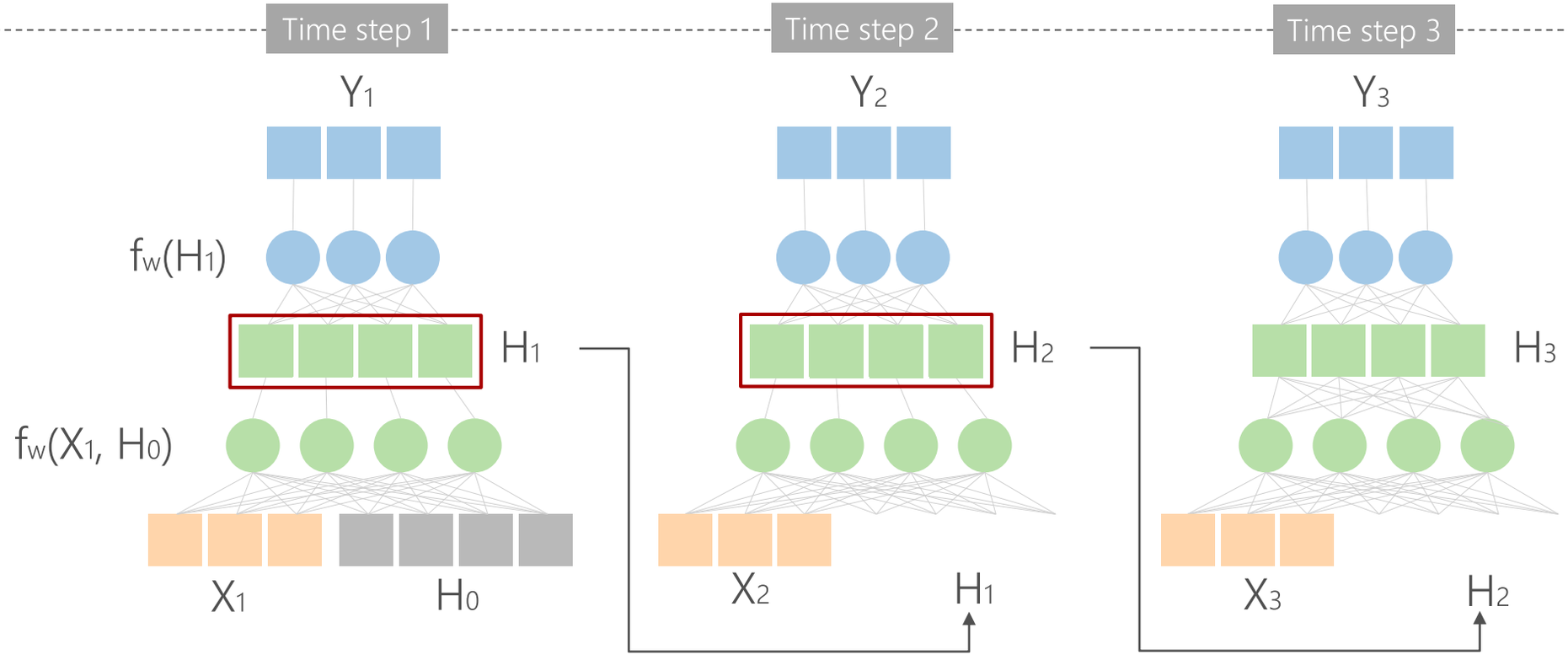


=

Simplified



Detailed

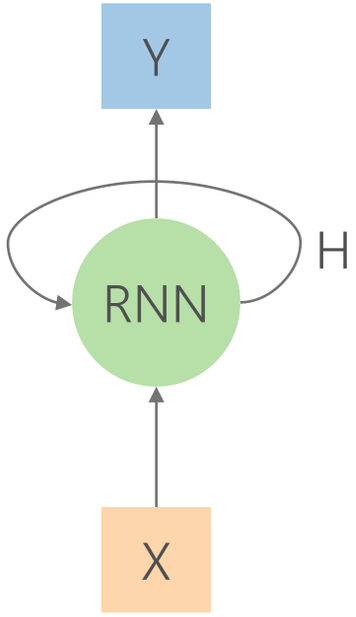


Unrolled RNN

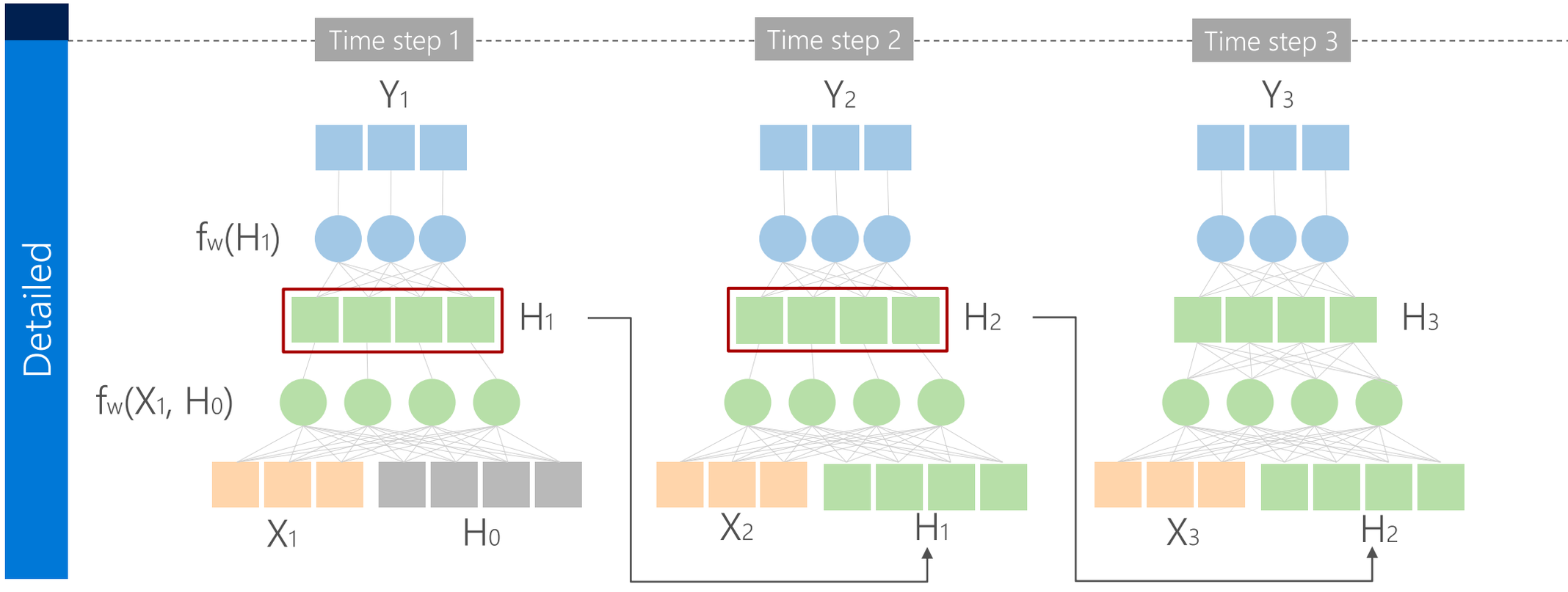
In Keras, the parameter "units" is dimensions of hidden state.
(Think of it as feedforward neural network number of units in hidden layer.)

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

(timesteps, features)

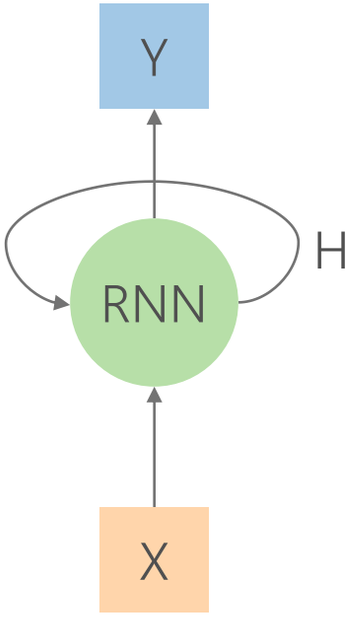


=



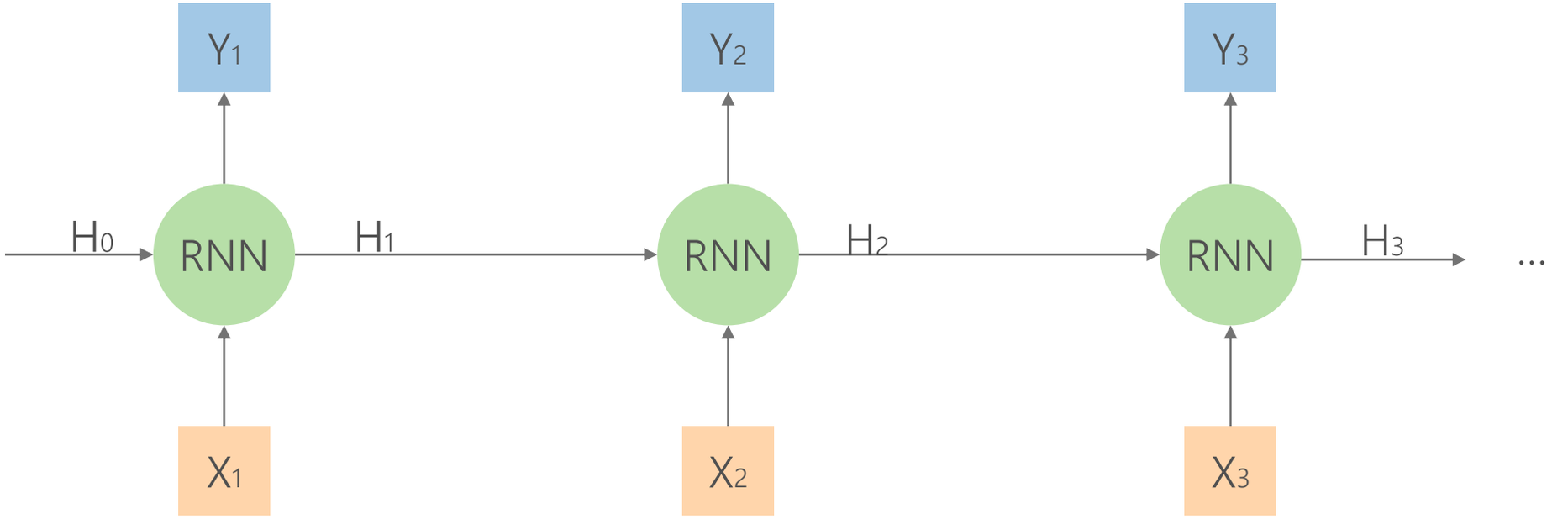
Microsoft

Unrolled RNN

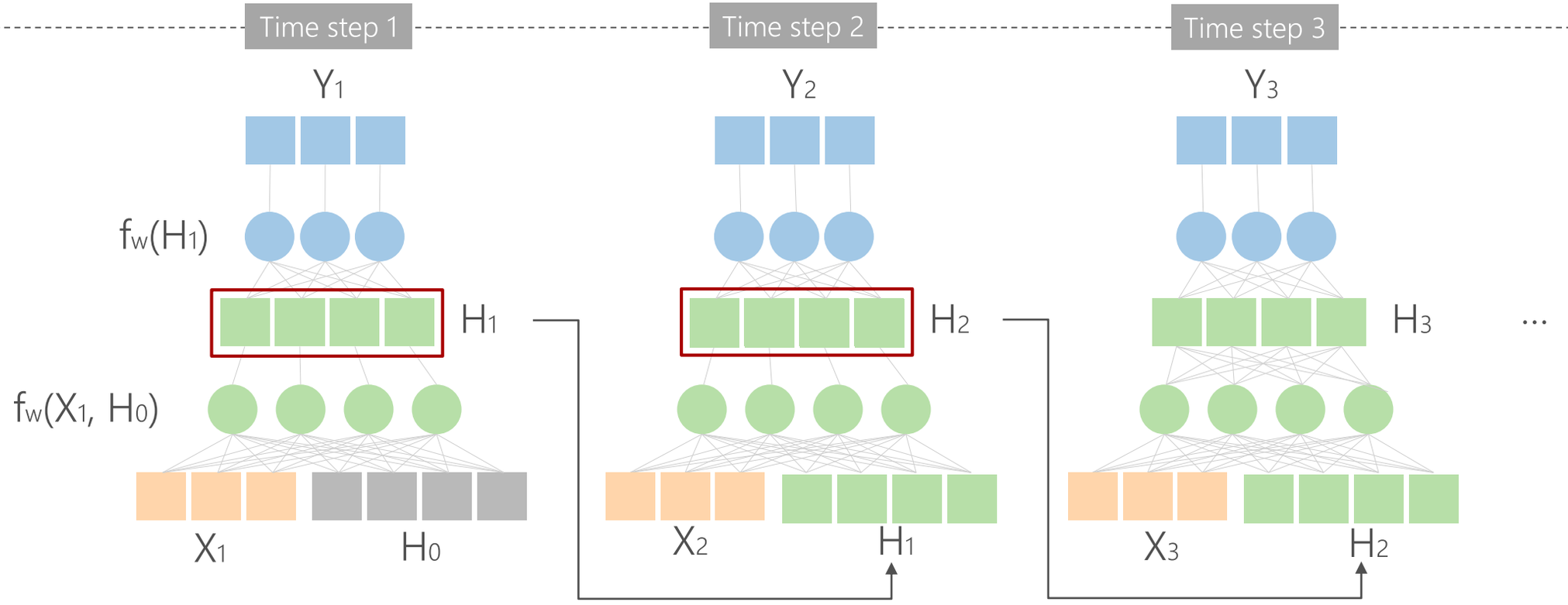


=

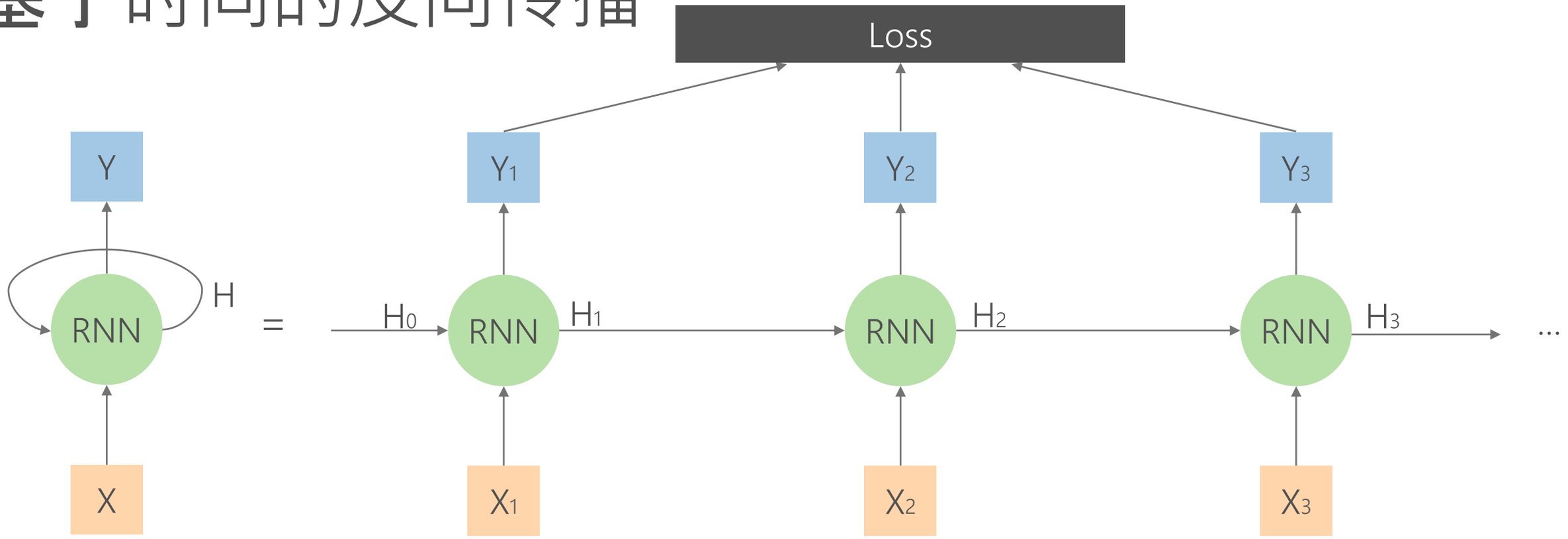
Simplified



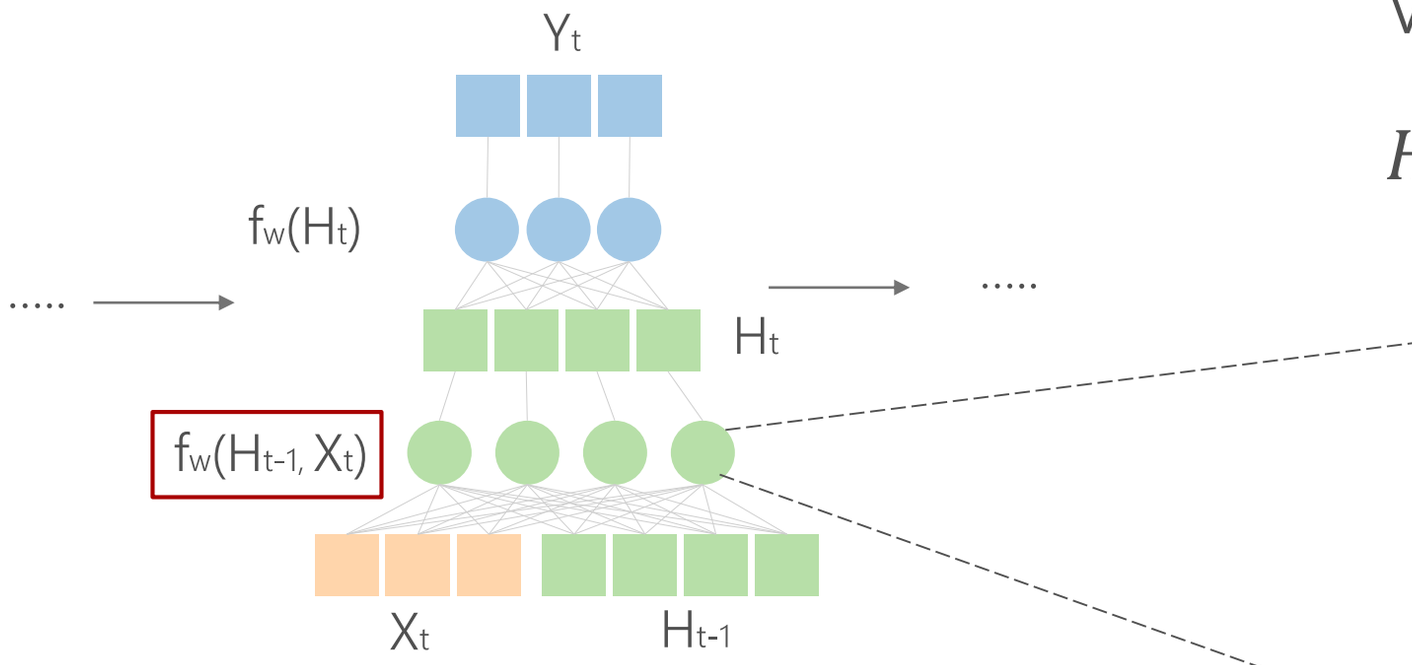
Detailed



Backpropagation through time (BPTT)
基于时间的反向传播



Vanilla RNN



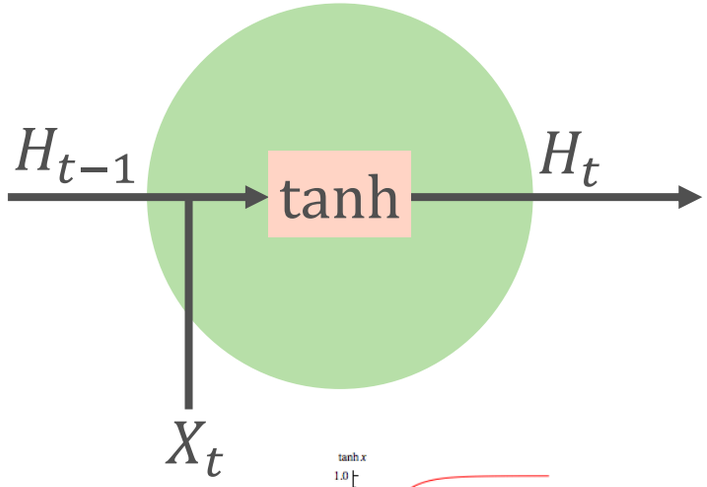
some function with parameter W

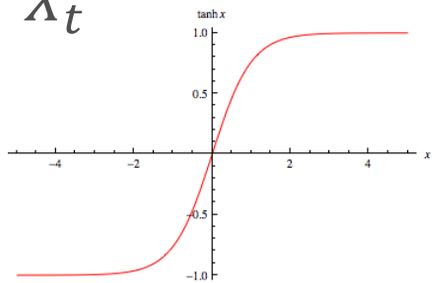
$$H_t = f_W(H_{t-1}, X_t)$$

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

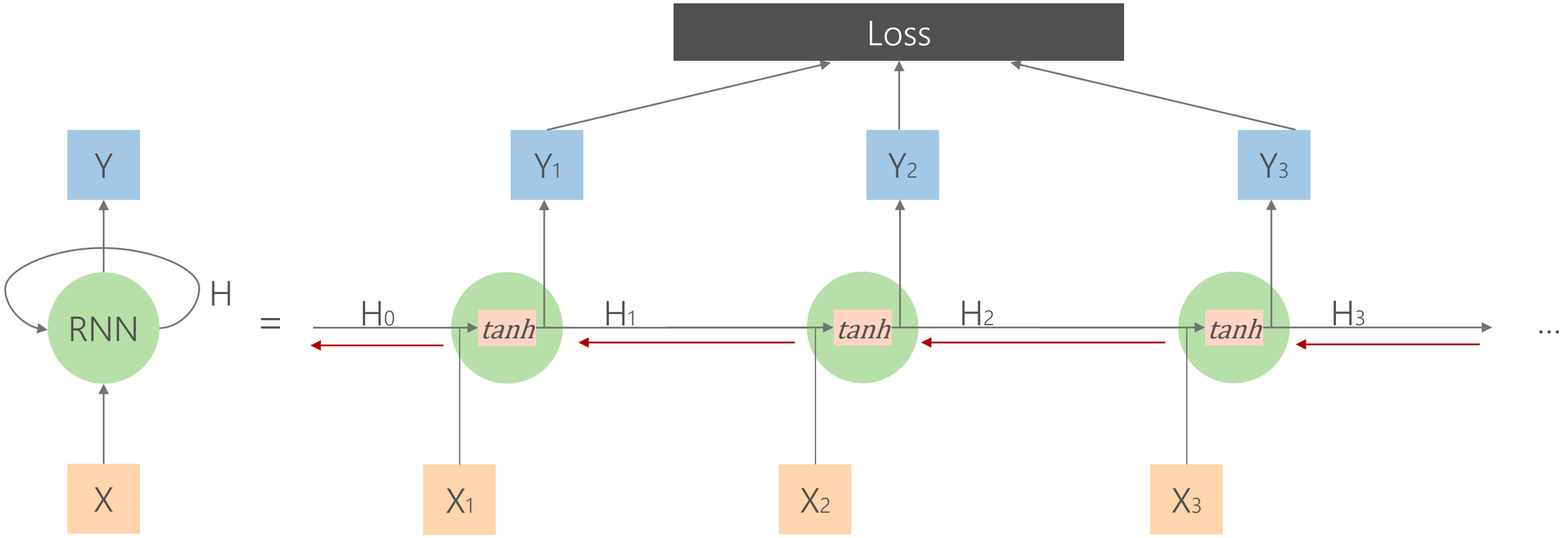
Vanilla RNN:

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





Vanilla RNN BPTT

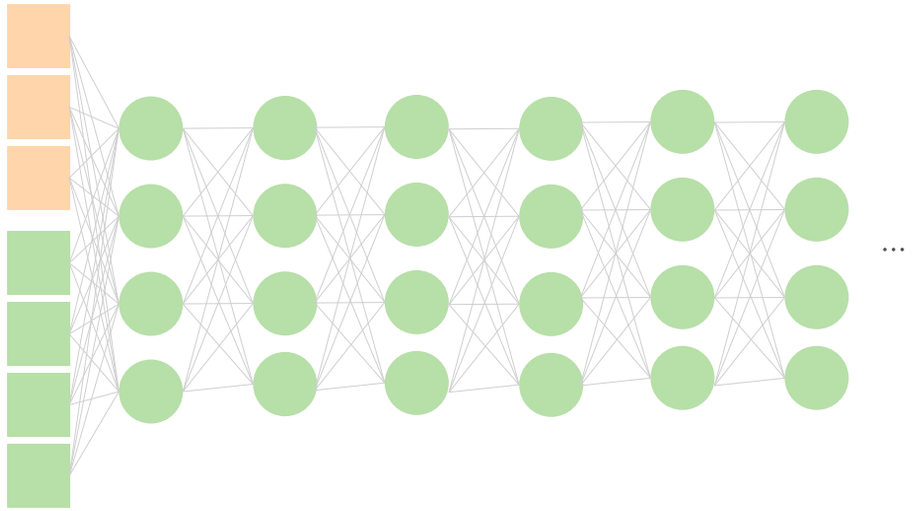


Computing gradient of h_0 involves repeated tanh and many factors of W

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
梯度爆炸

- 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

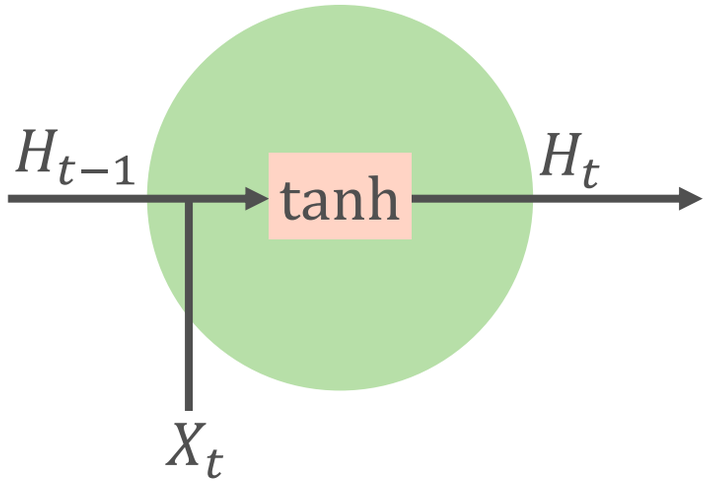
Vanishing Gradient
梯度消亡

- 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)

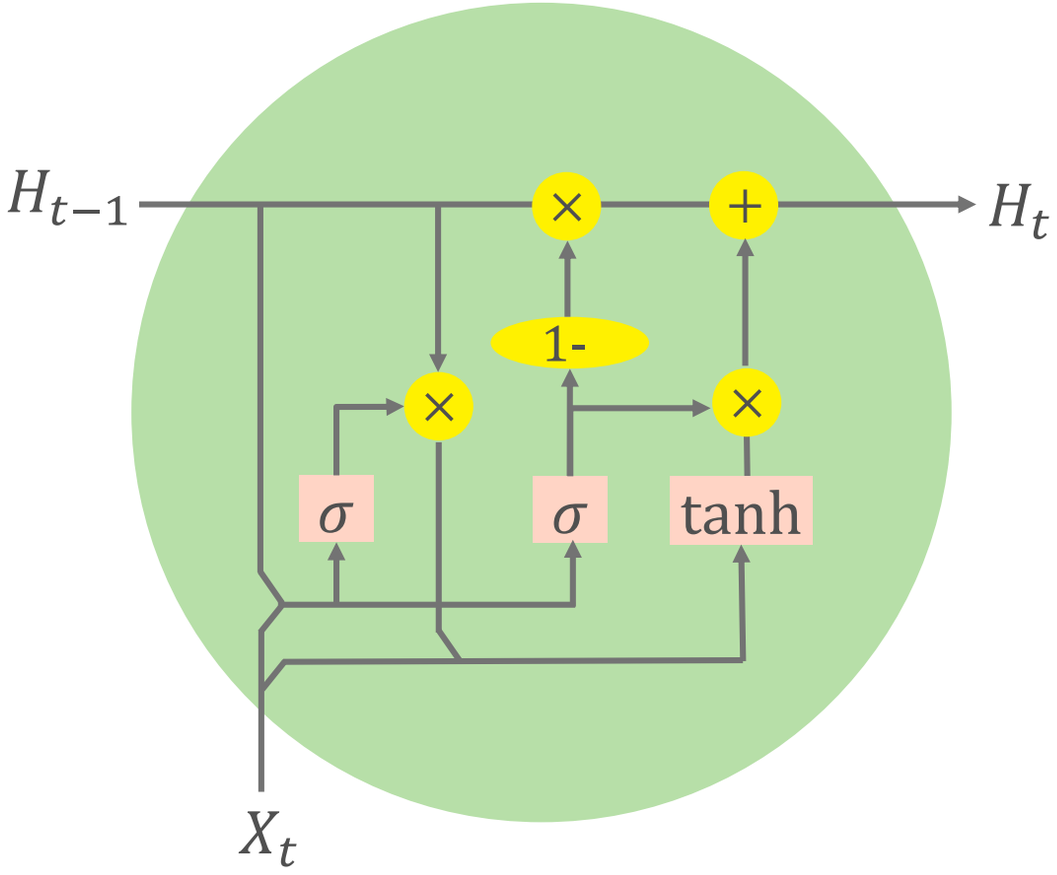
Vanilla RNN:

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

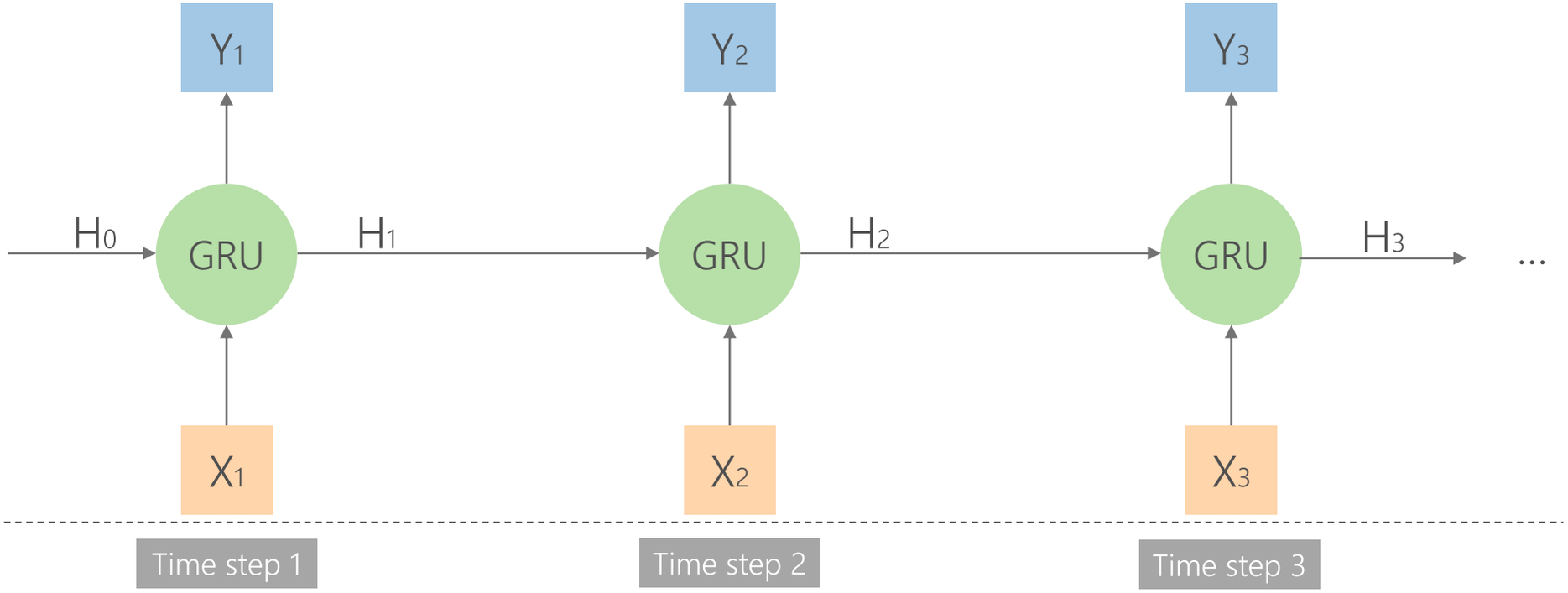


GRU:

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

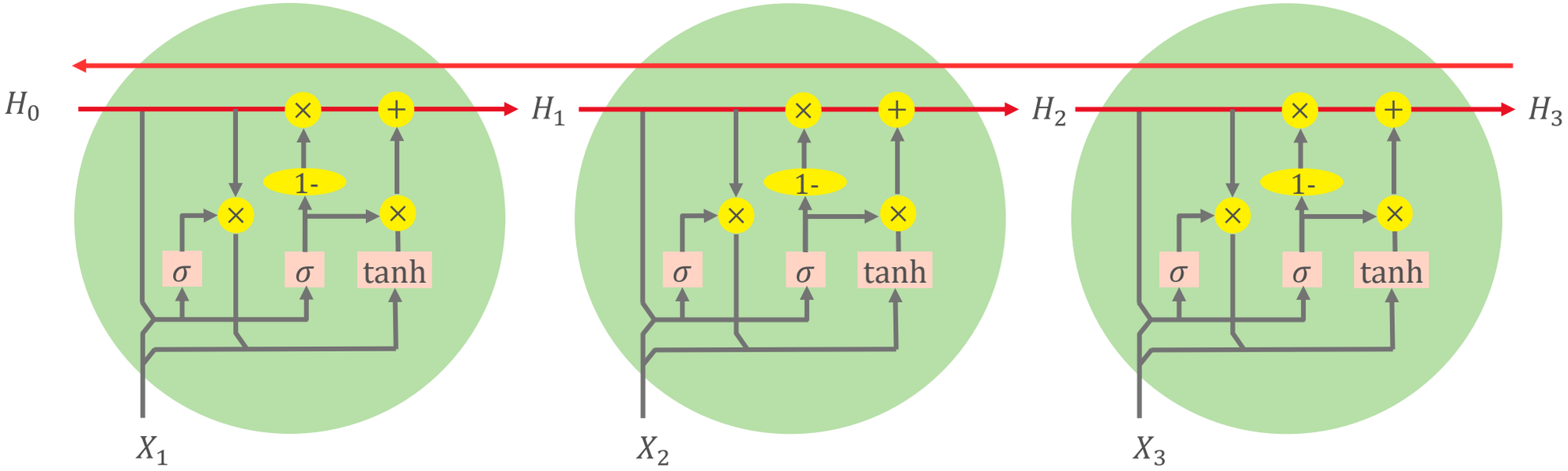


GRU

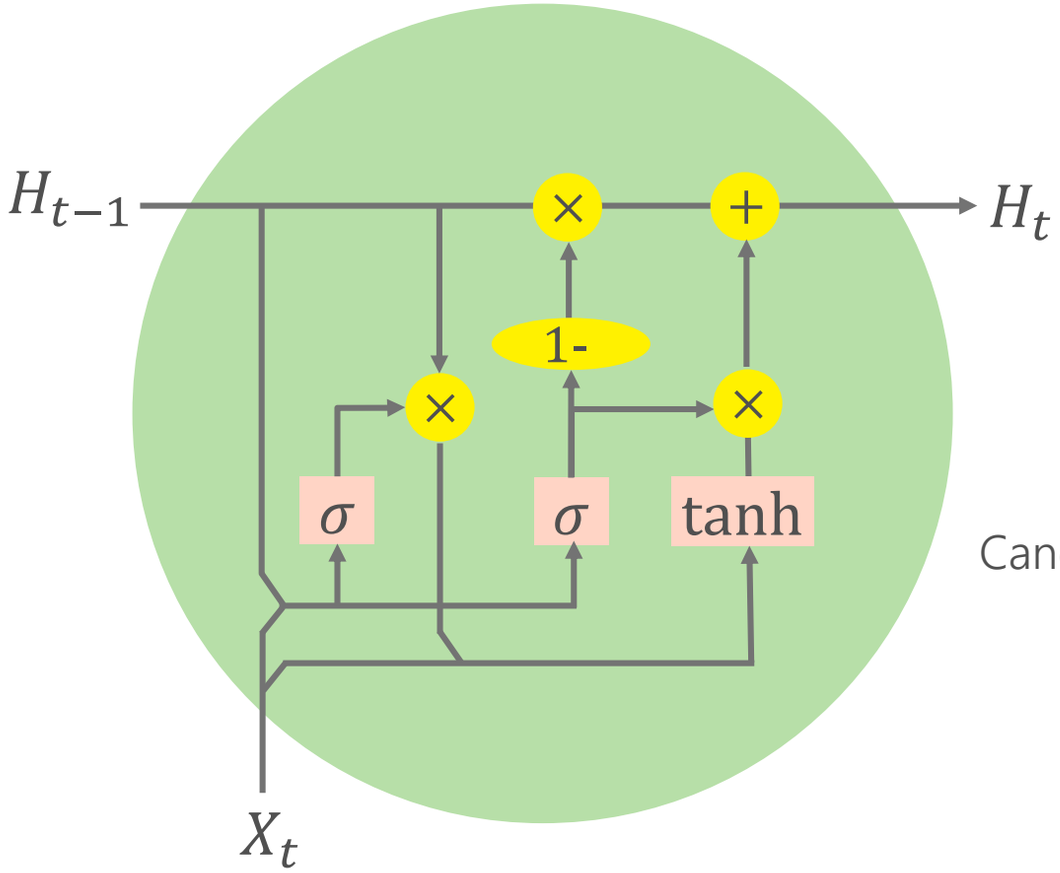


Uninterrupted gradient flow

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



Gated Recurrent Unit (GRU)



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

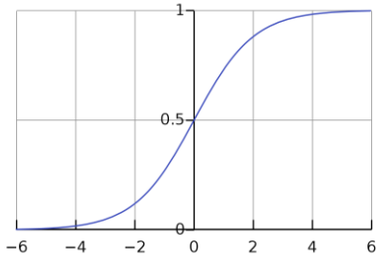
Update gates: $z_t = \sigma(W_z \cdot [H_{t-1}, X_t])$

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

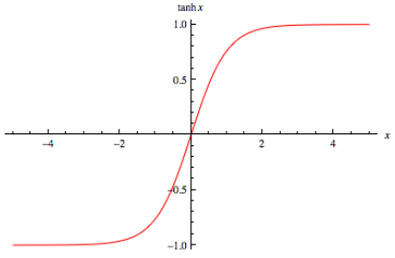
Reset gates: $r_t = \sigma(W_r \cdot [H_{t-1}, X_t])$

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

σ
(0,1)

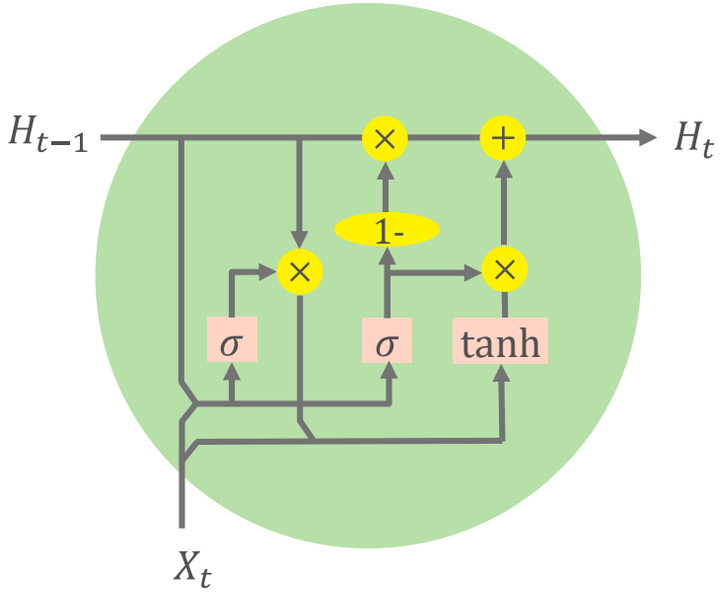


tanh
(-1, 1)

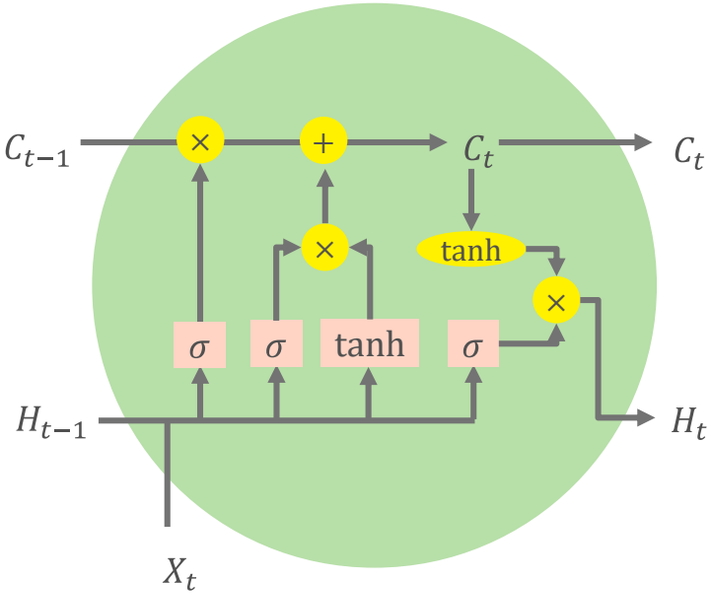


GRU vs LSTM (Long Short Term Memory)

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

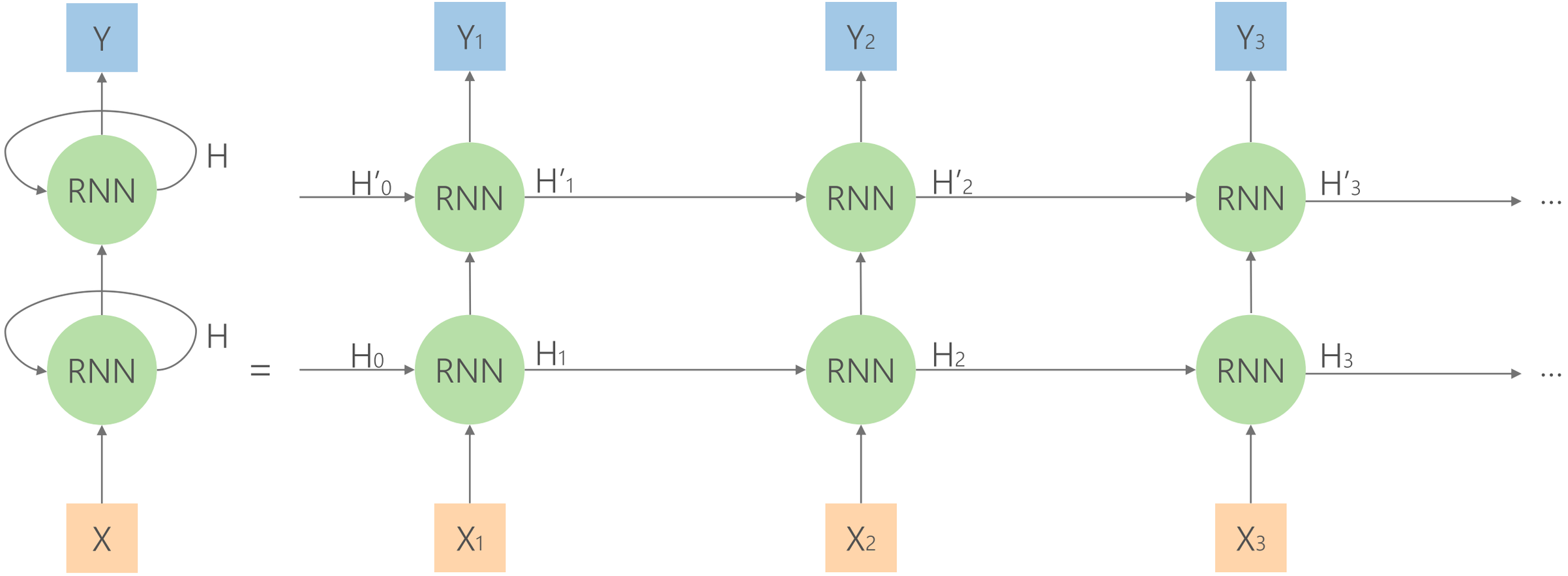
Input gates: $i_t = \sigma(\ W_i \cdot [H_{t-1}, X_t]\ )$

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

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

RNN Stacking

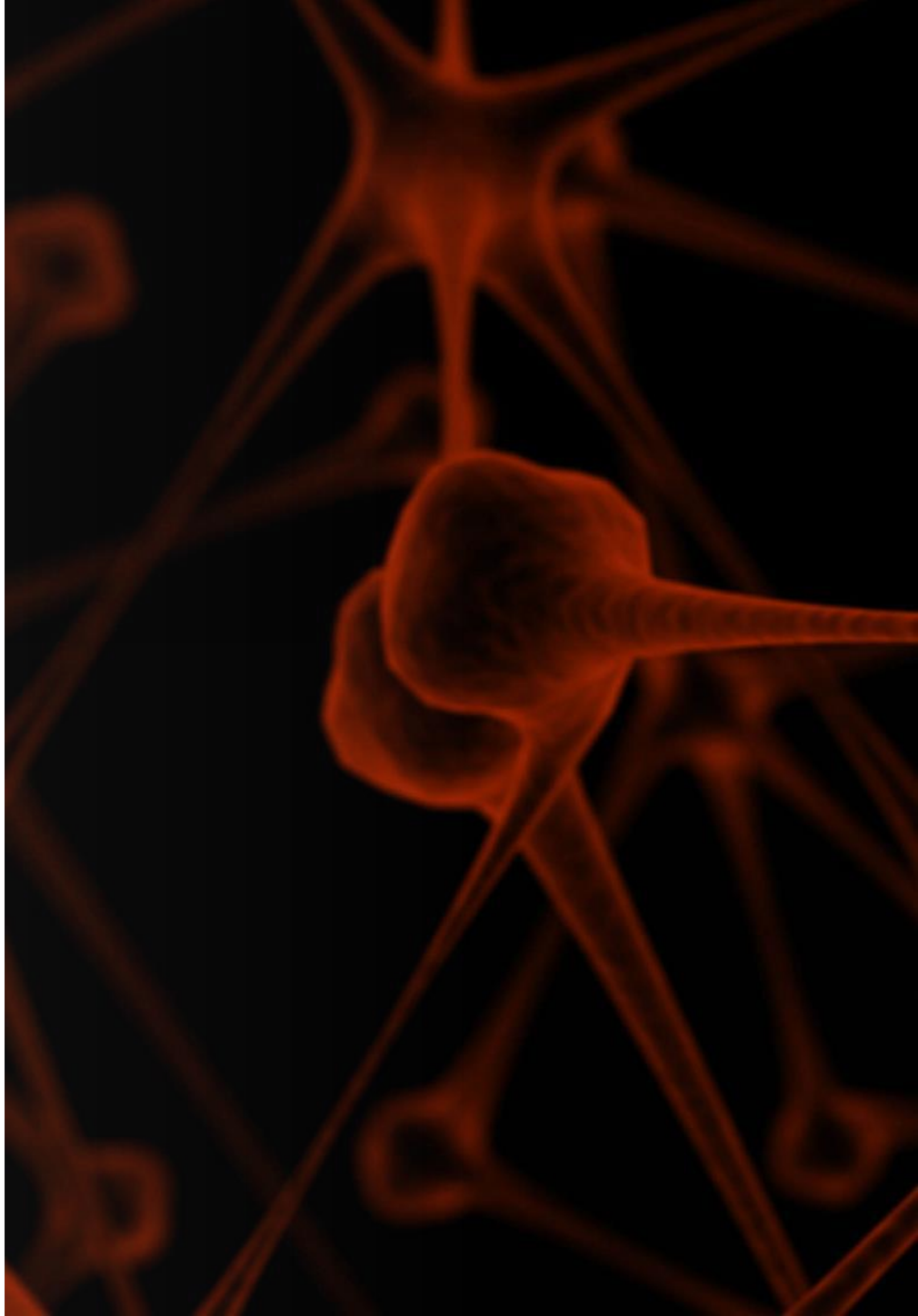
To learn more complex relationships, we can go deep by stacking the cells.




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

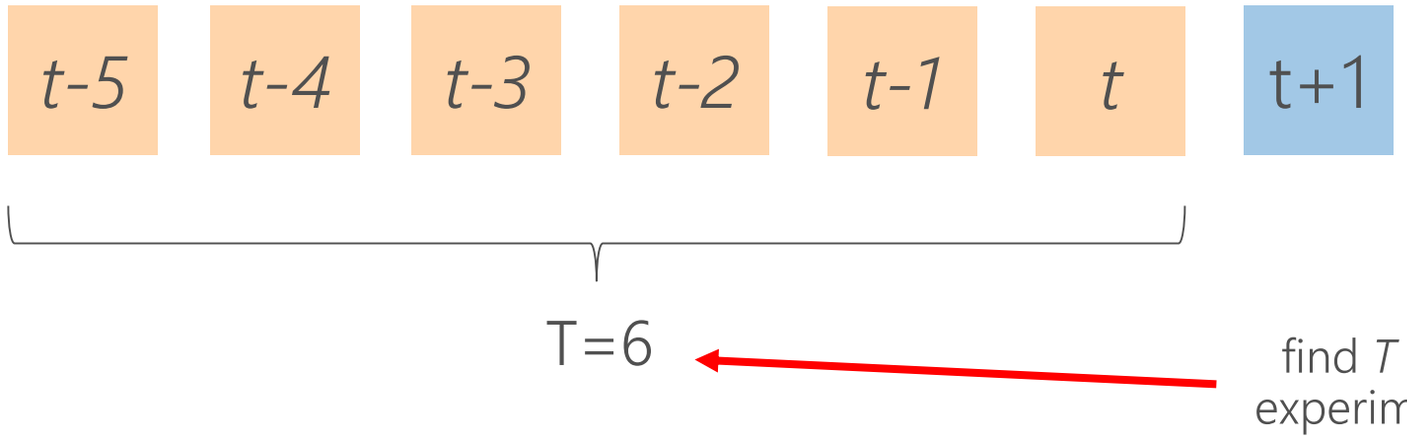
Microsoft

RNN for one step time series forecasting
RNN单步时间预测



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
动手实验

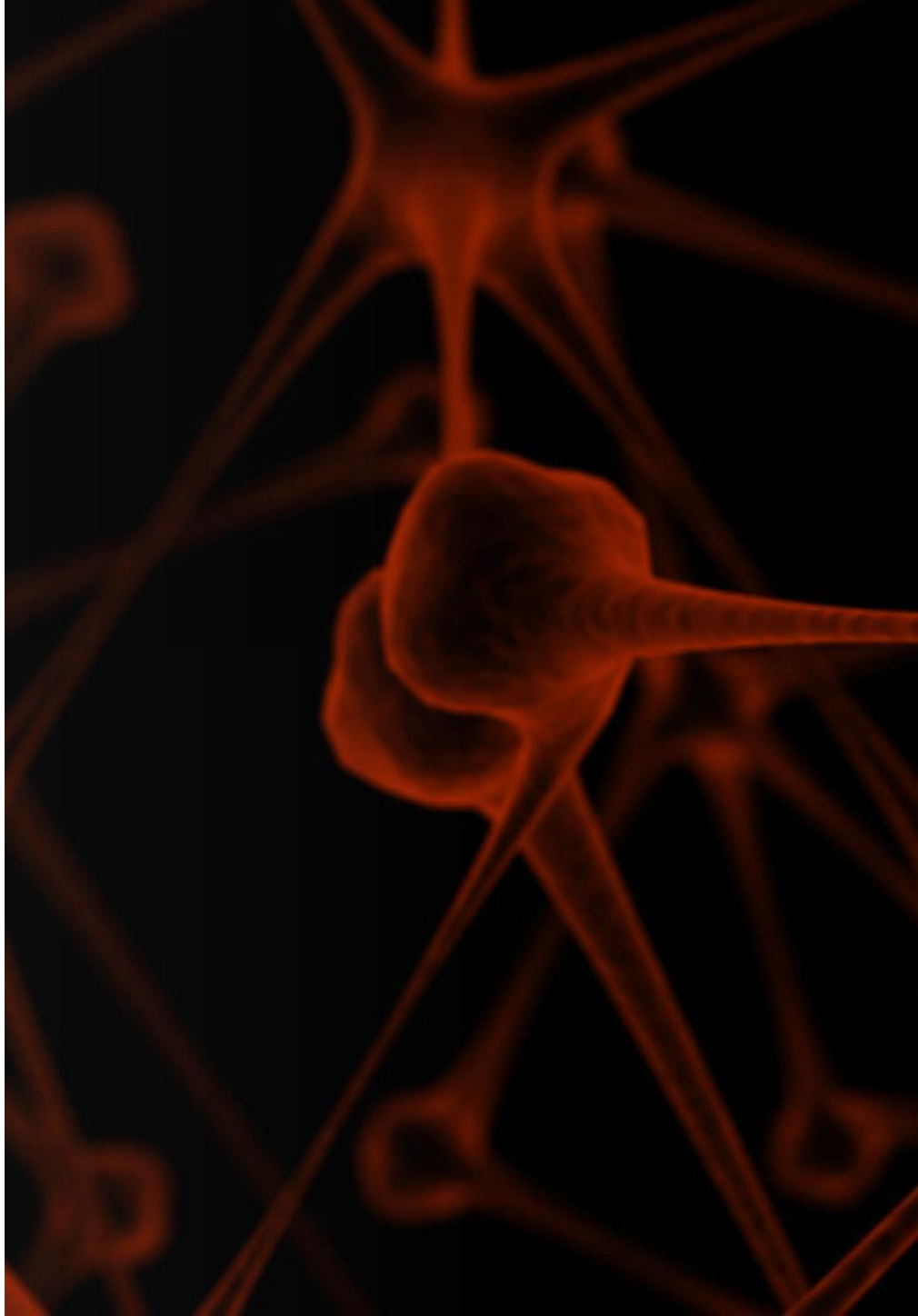
Open: 2_RNN.ipynb notebook

Hands-on Quiz
动手实验小测试

Open: Quiz_RNN.ipynb notebook

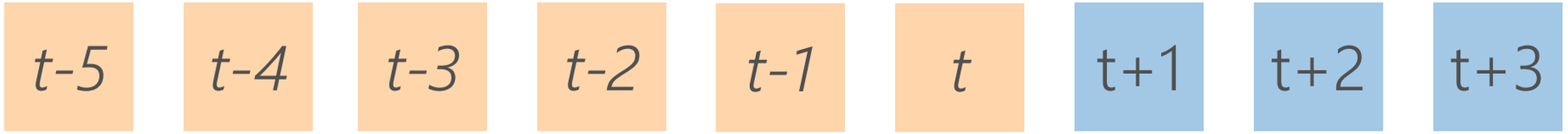
Microsoft

RNN for multi-step time series forecasting
RNN多步时间序列预测

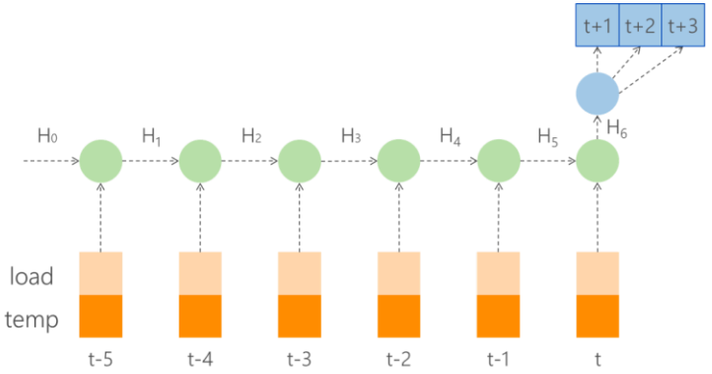


Multi-step forecast

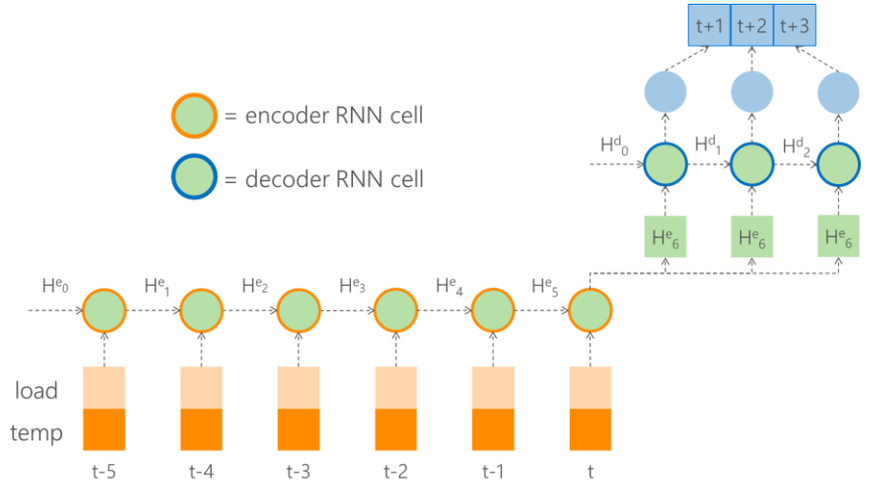
- Assuming we are at time *t* …
- … predict the values at times (*t+1*, … , *t+HORIZON*) …
- … conditional on the previous *T* values of the time series



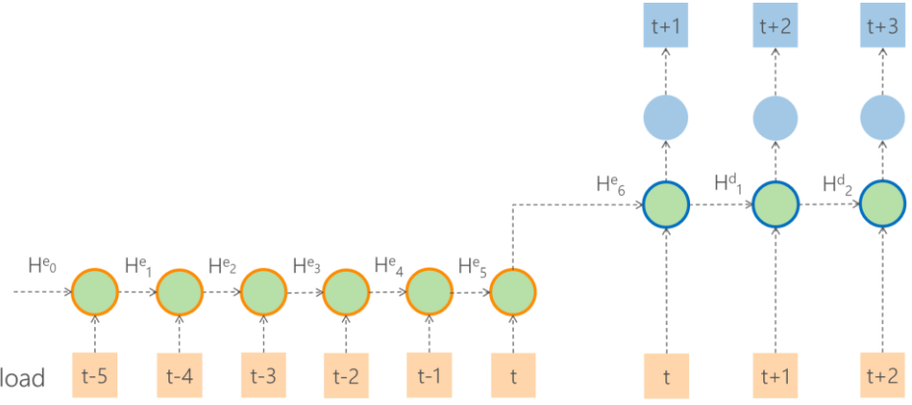
Multi-step forecast



Vector output

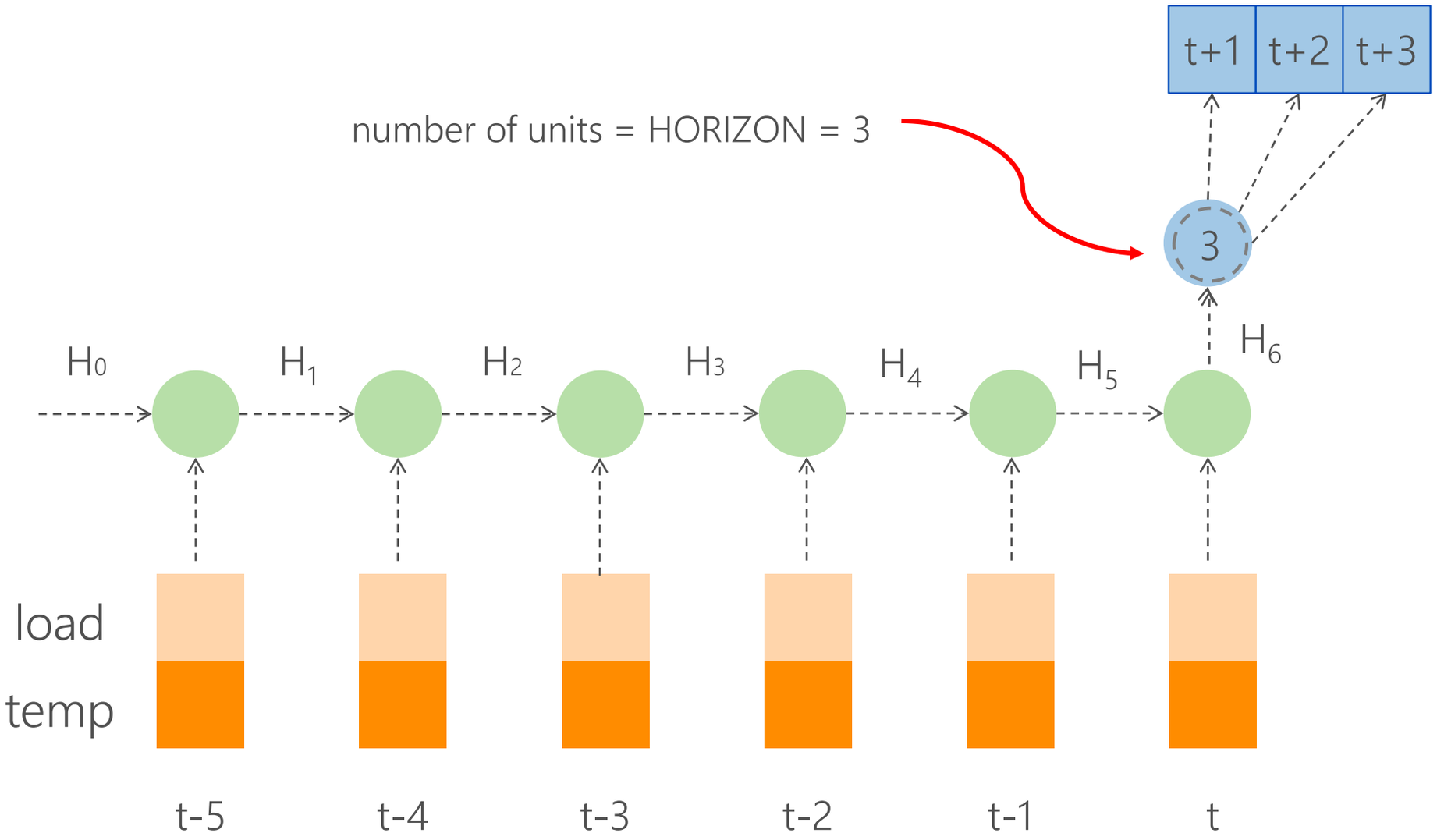


Simple encoder-decoder

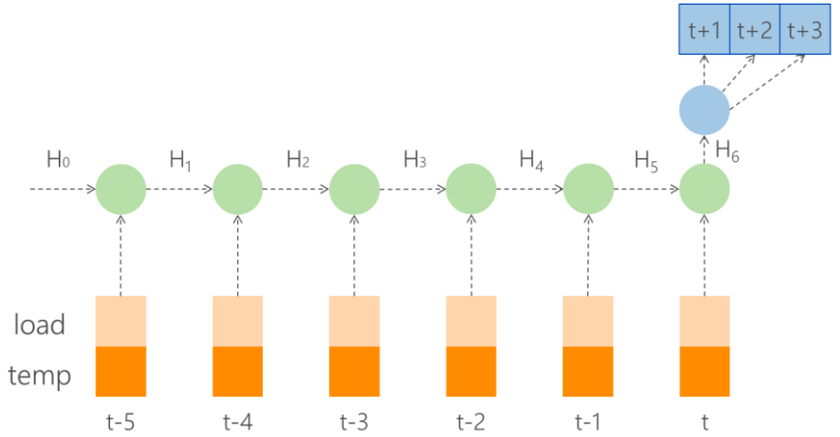


Recursive encoder-decoder

Vector output approach



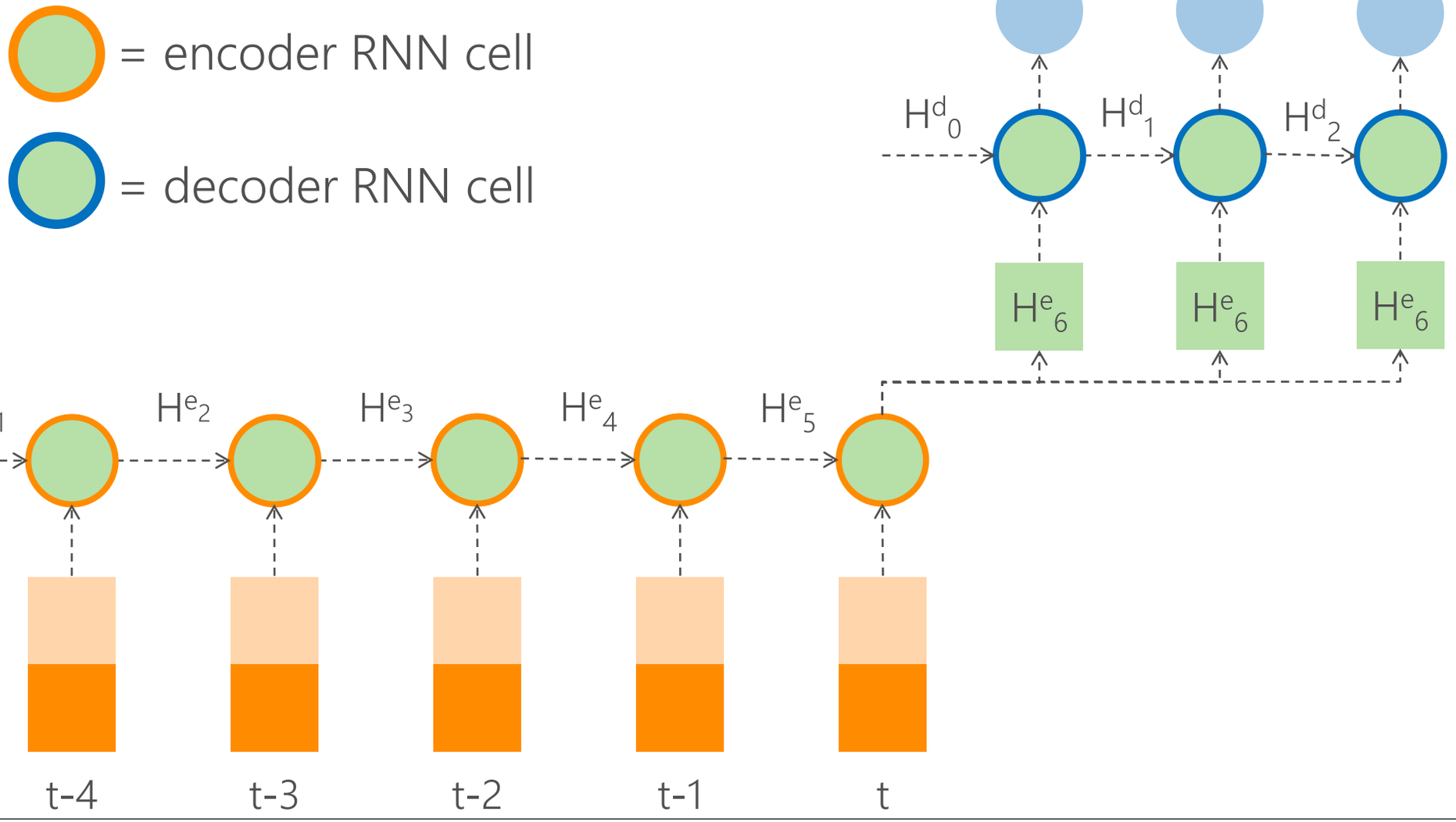
Vector output approach



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

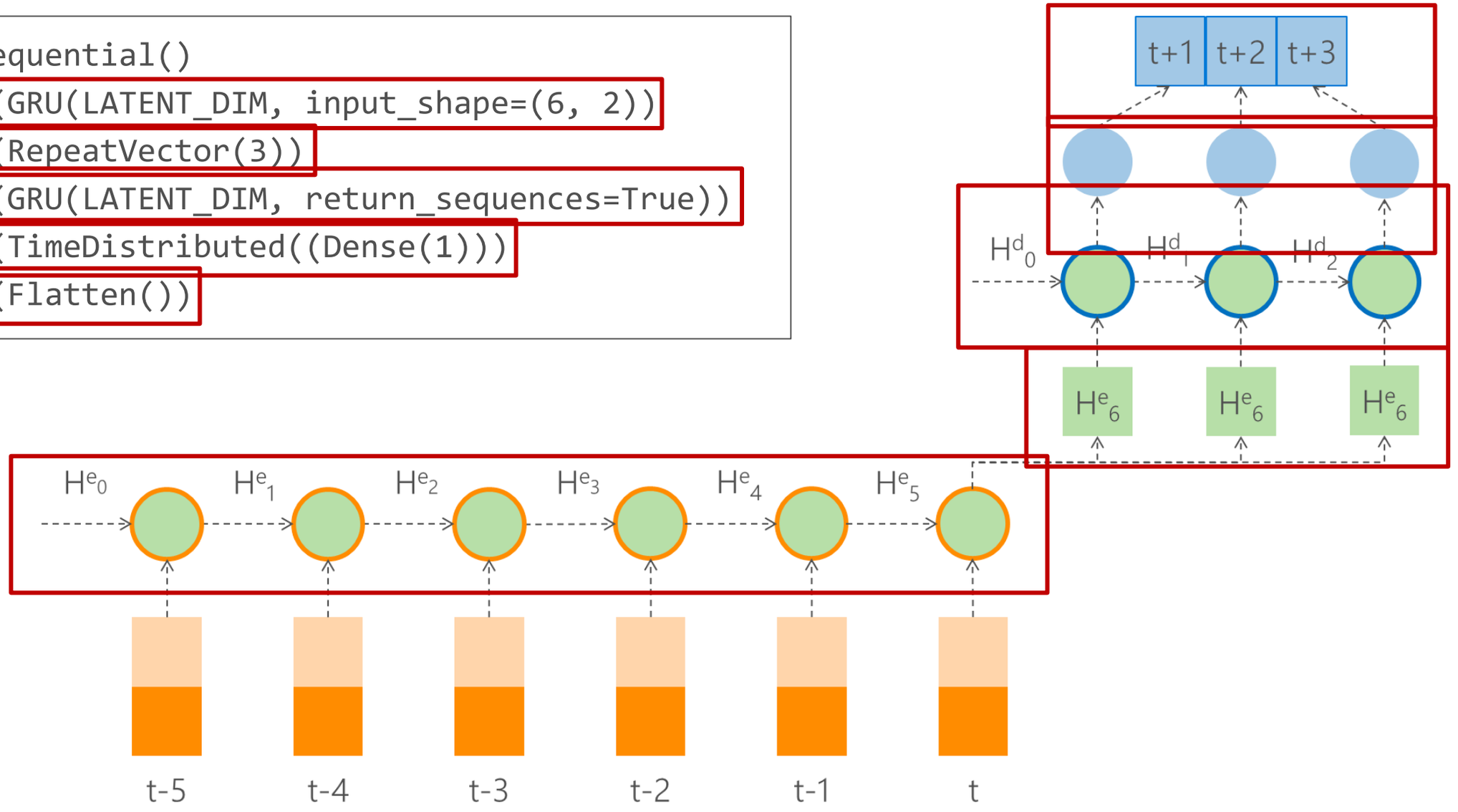
Reference notebook: multi_step_RNN_vector_output.ipynb

Simple encoder-decoder

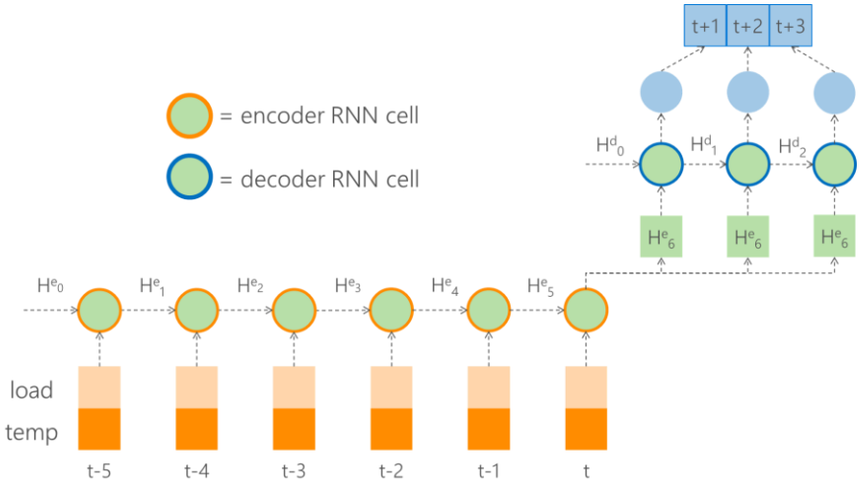


Simple encoder-decoder

```
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

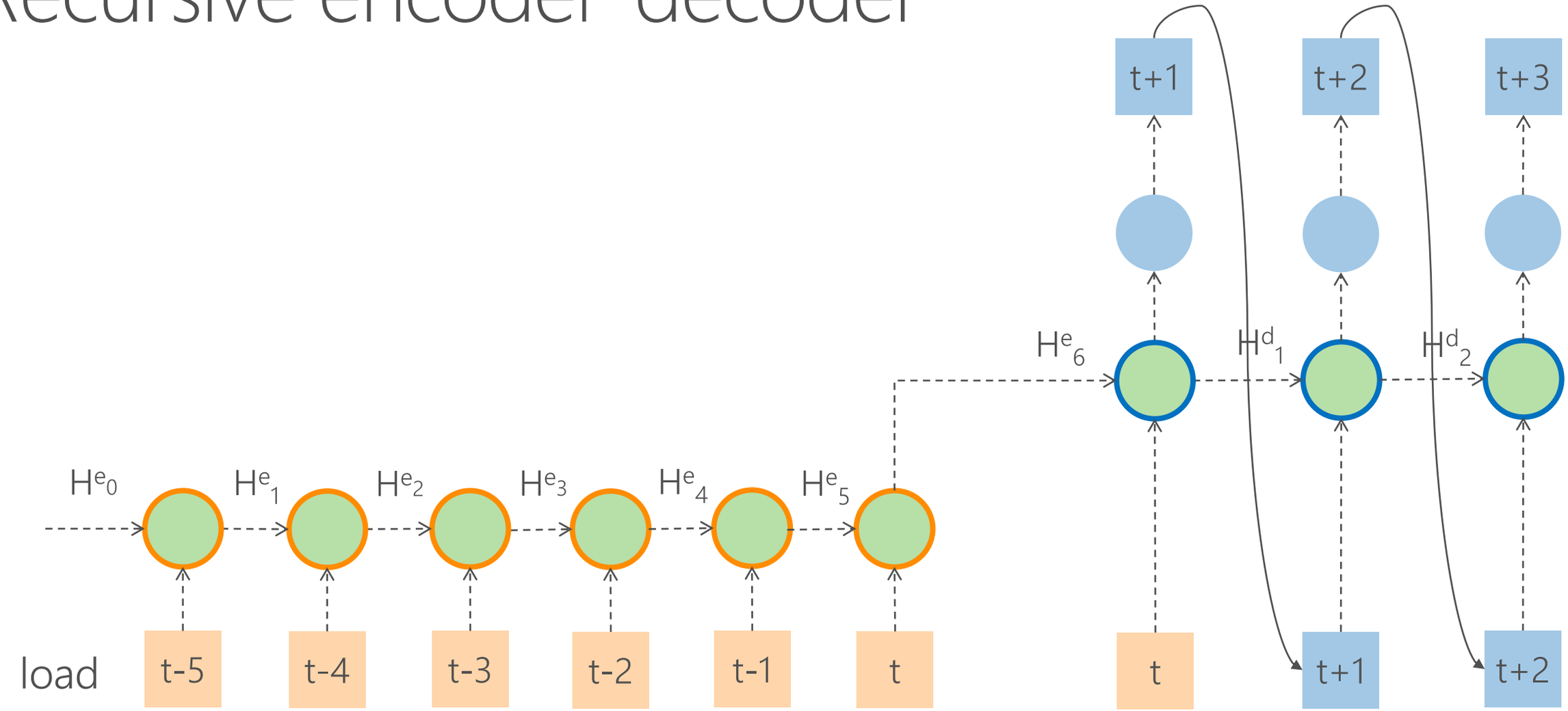


- Fairly simple to implement
- Tries to capture dependencies between forecasted time steps through decoder hidden state
- Slower to train with stacked RNN layers

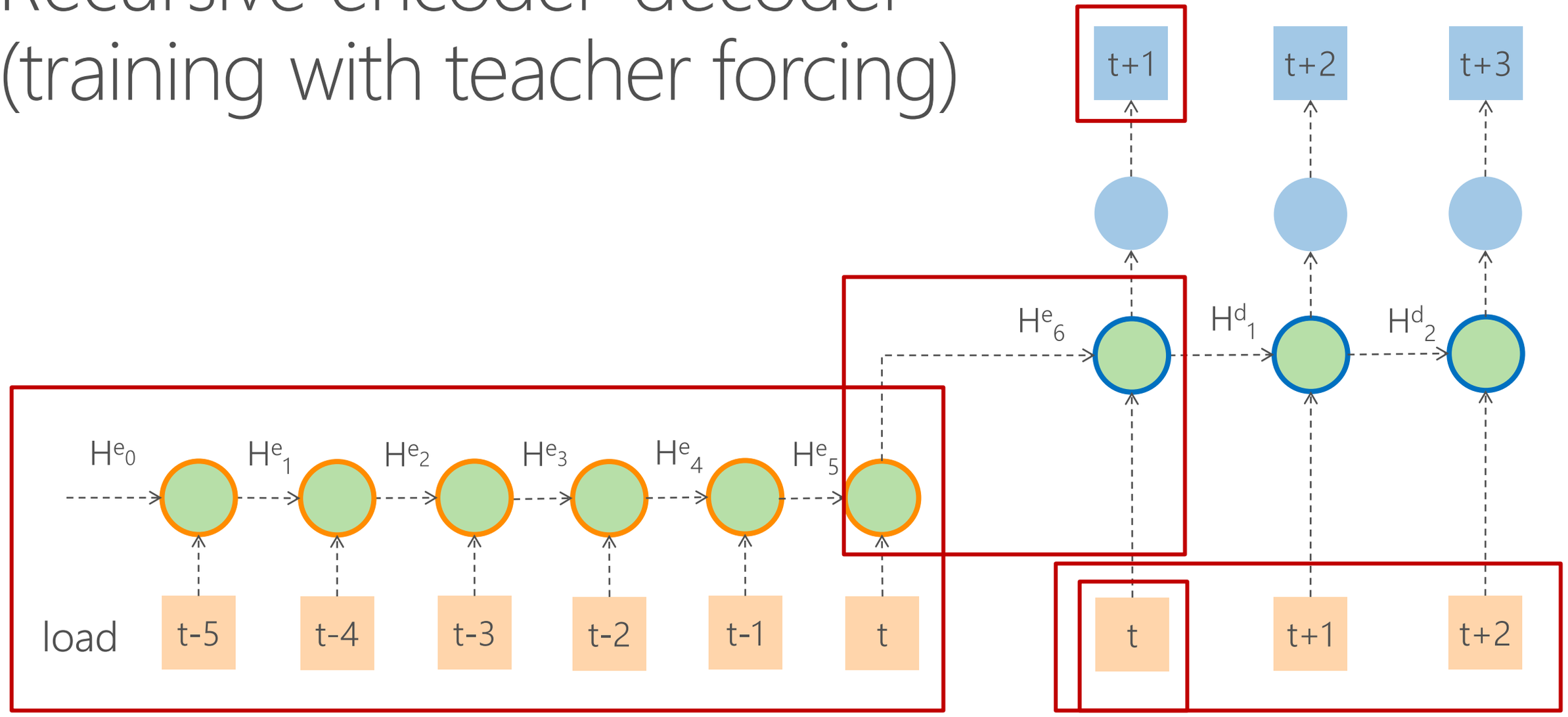
Hands-on Experiment
动手实验

Open 3_RNN_encoder_decoder.ipynb

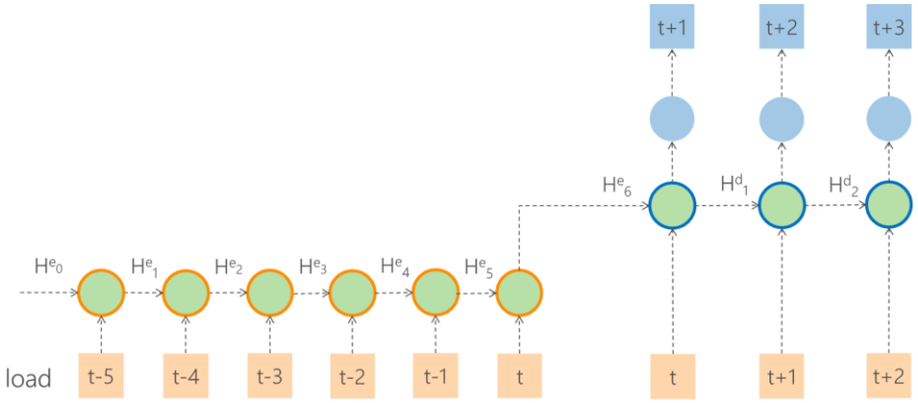
Recursive encoder-decoder



Recursive encoder-decoder (training with teacher forcing)



Encoder-decoder with teacher forcing

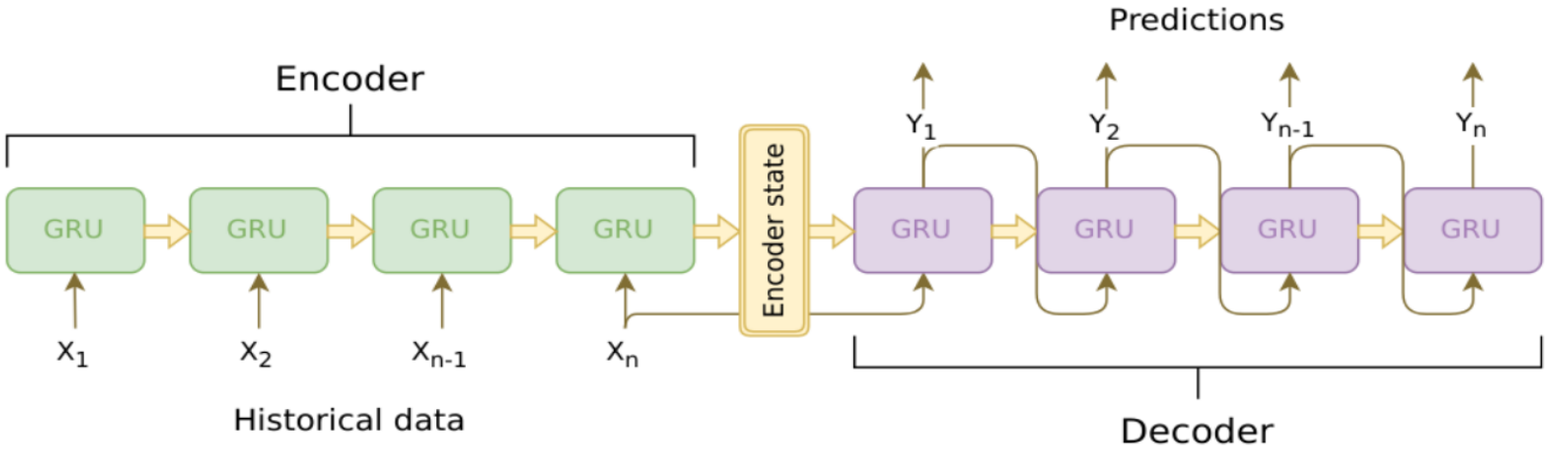


- Tries to capture dependencies between forecasted time steps through recursive decoder
- Variable length output (can generate forecasts of variable horizons)
- More difficult to implement
- Slower to train and slower to generate forecasts
- Error propagation due to recursive decoder

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
动手实验小测试

Open Quiz_RNN_encoder_decoder.ipynb

Microsoft

Hybrid models for time series forecasting
时间序列预测的混合模型

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: $y_1, y_2, \ldots, y_t$

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

forecast of next value

actual current value

learned parameter

forecast of current value

Equivalent form:

forecasting $\hat{y}_{t+1|t} = l_t$ ← 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

actual value of the current level

forecast of current level

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

forecast of the next seasonality term

actual value of the current seasonality term

forecast 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

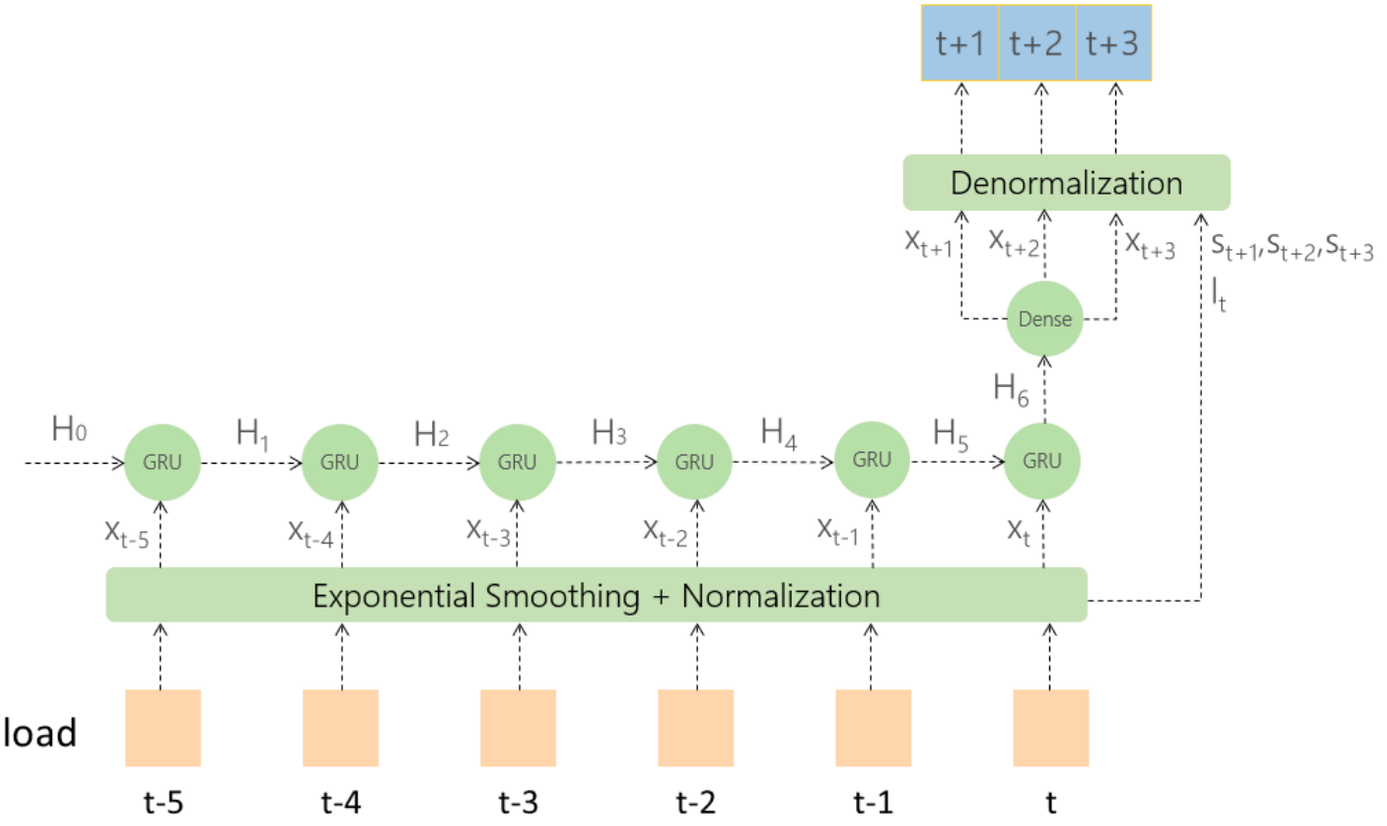
ES-RNN can be used to forecast multiple time-series

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: $y_1, y_2, \ldots, y_t$

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

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

Assumption 2: latent state z_t evolves over time

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

parameters

ARIMA and Exponential Smoothing are State Space Models

Deep State Space Model

Time-series: $y_1, y_2, \ldots, y_t$

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

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

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

Feature vectors: $x_1, x_2, \ldots, x_t$

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

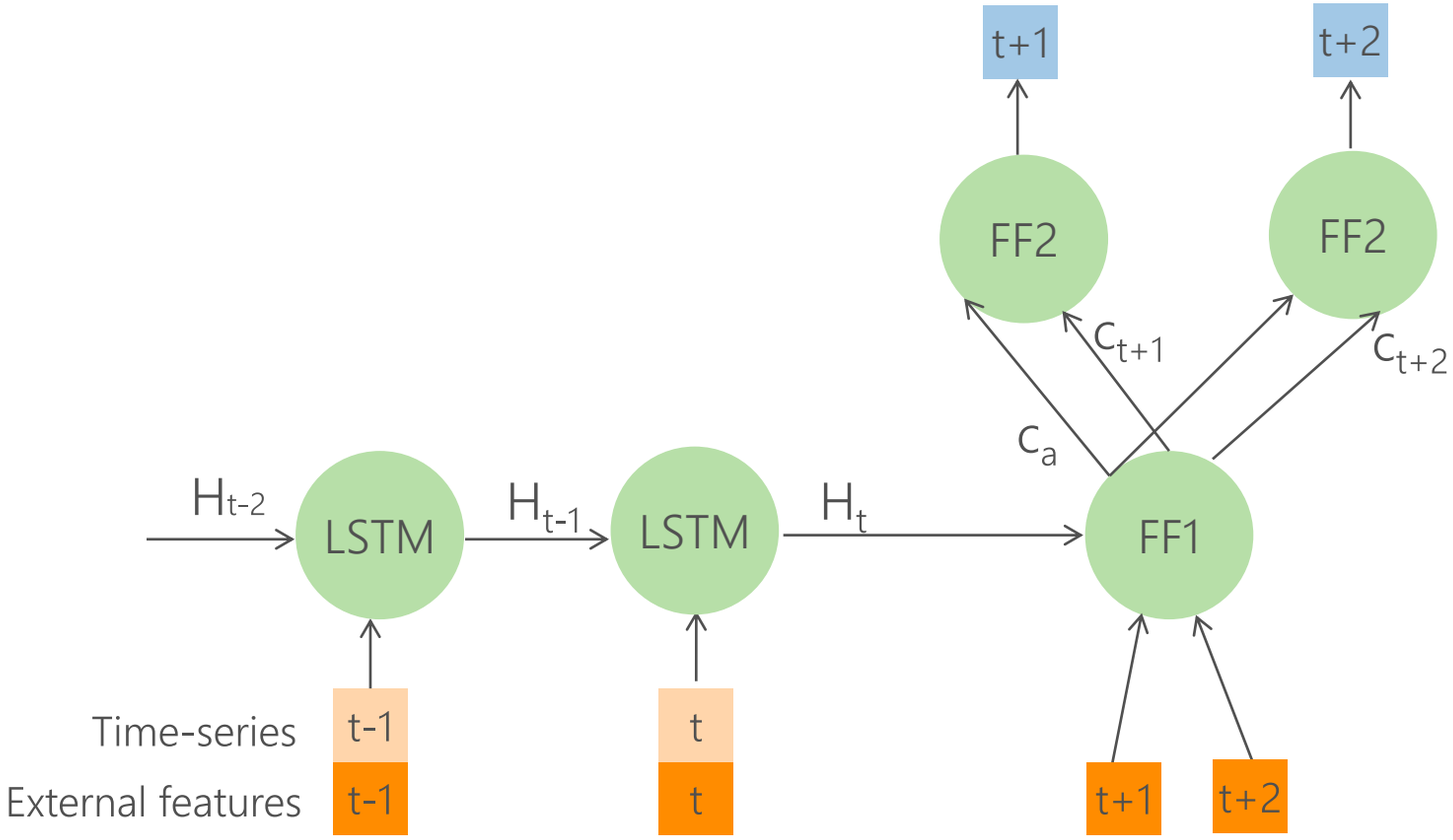
Multi-Horizon Quantile Recurrent Forecaster

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

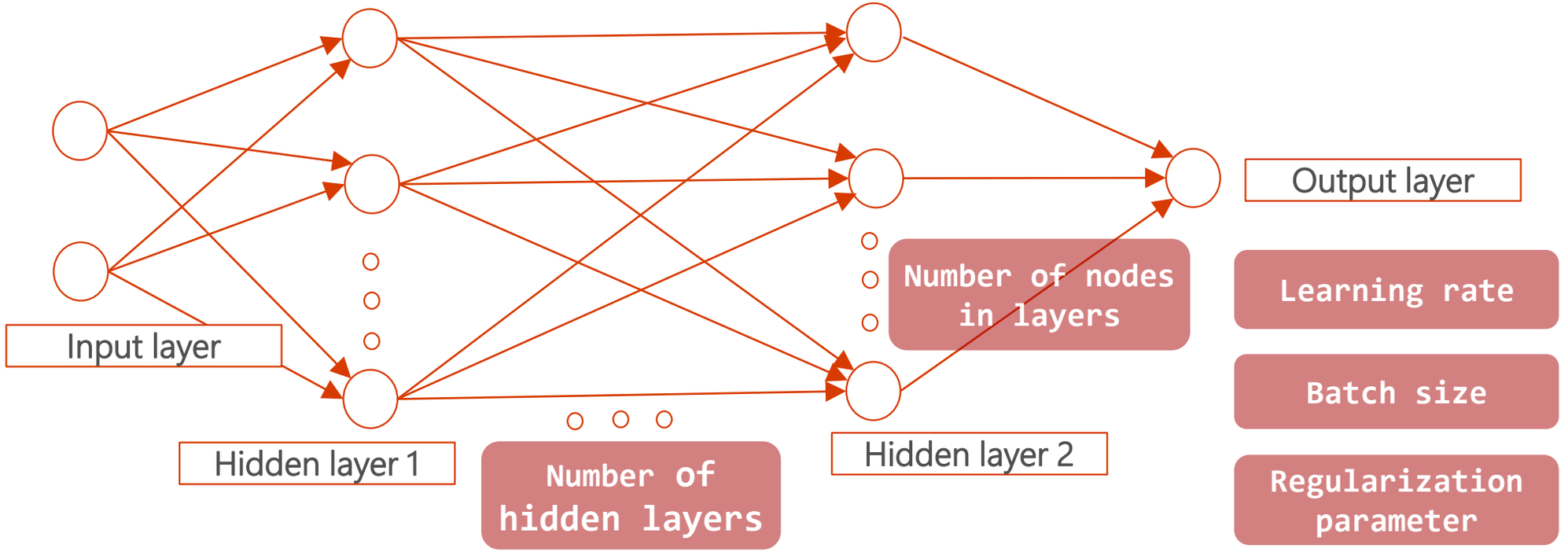
★ External features: weekday/holiday indicators, day of week, hour of the day, weather data

Microsoft

Hyper-parameter Tuning
超参数调优

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.



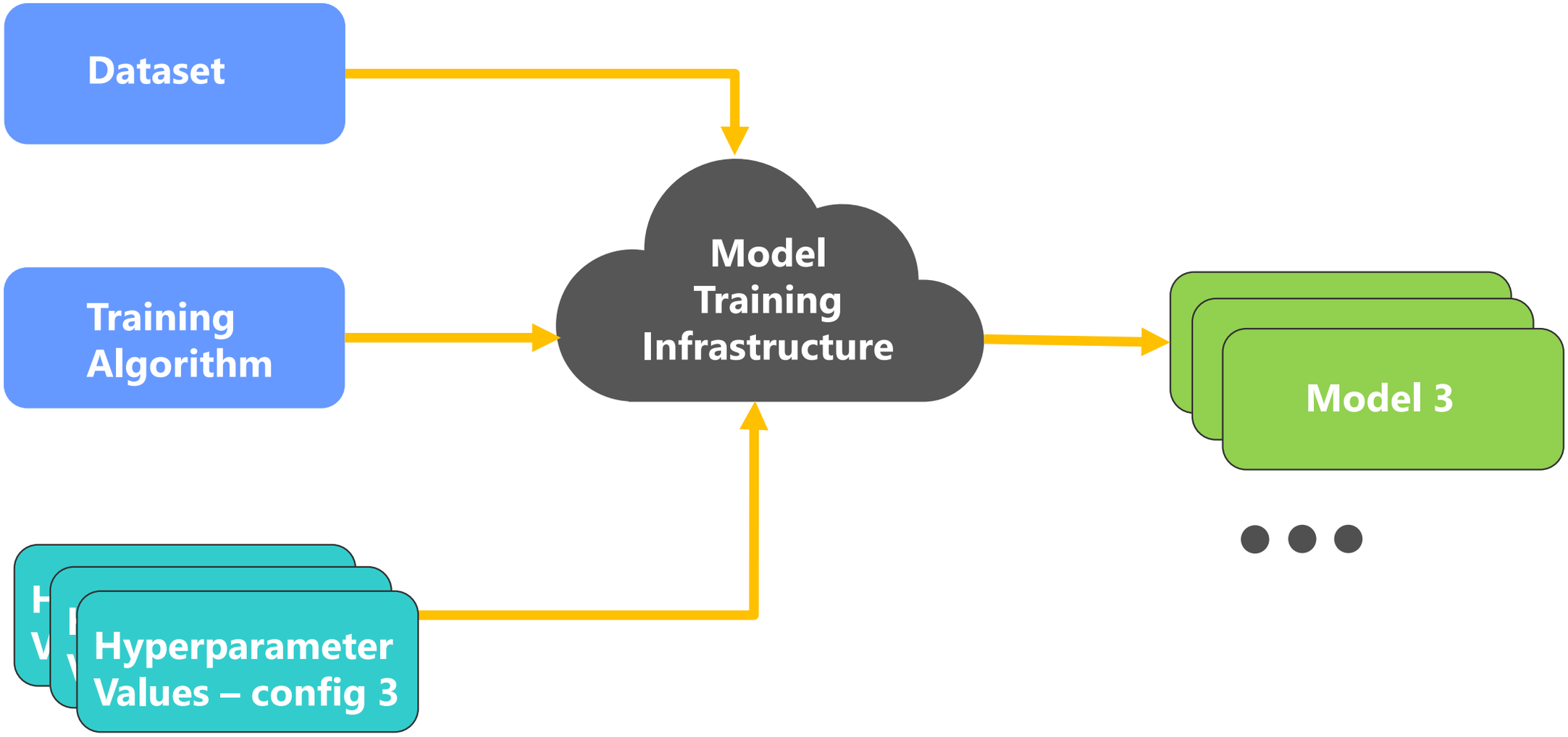
Hyper-parameter tuning

- 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

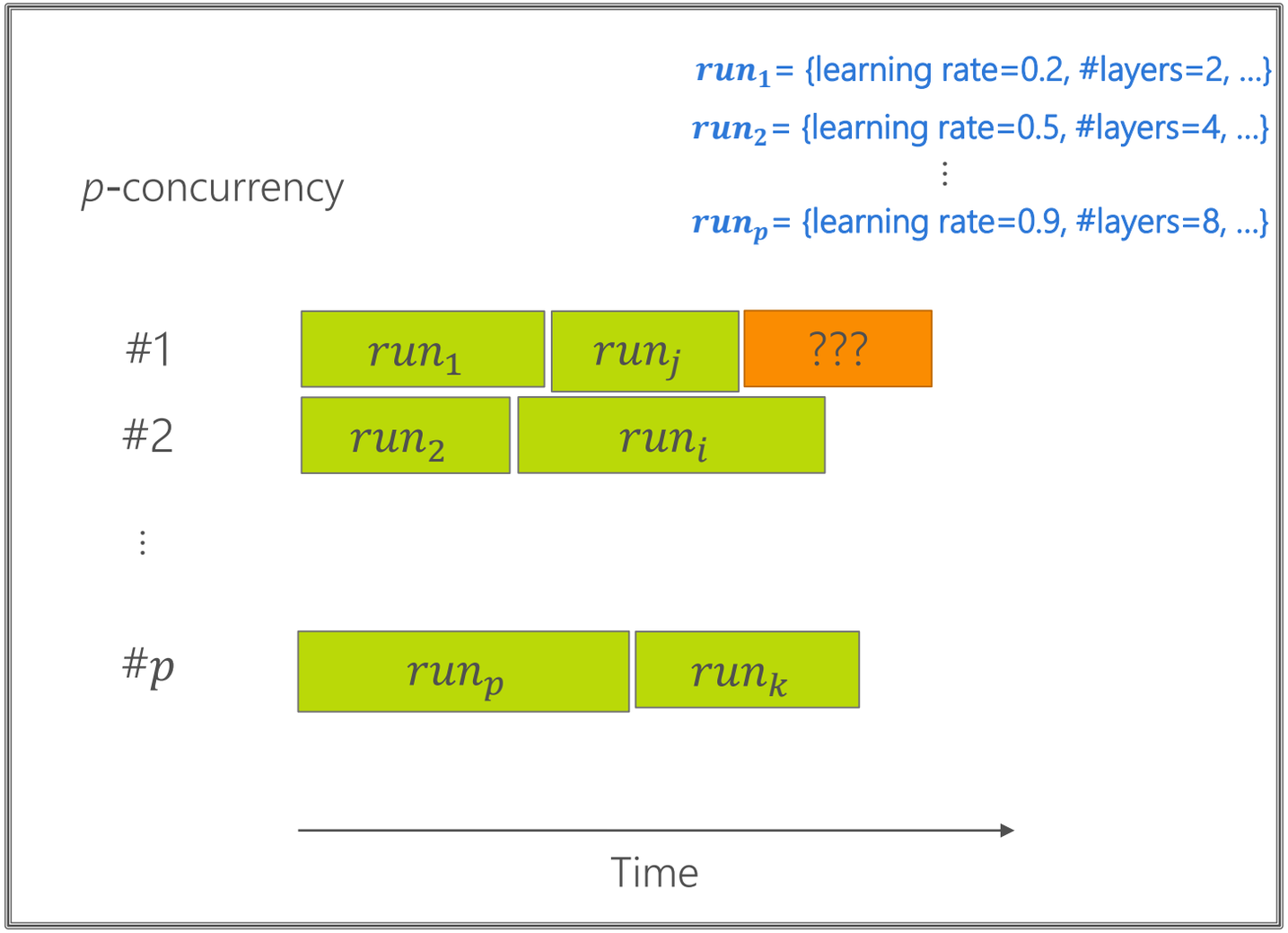
- Automate the exploration process
- Efficient use of resources
- Optimize model for a specified metric
- Offers control over resource budget
- Apply to different models and learning domains
- Training framework agnostic
- Visualize all configurations in one place

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 采样方式

- Random Sampling

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

We get $n >= 60$. 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

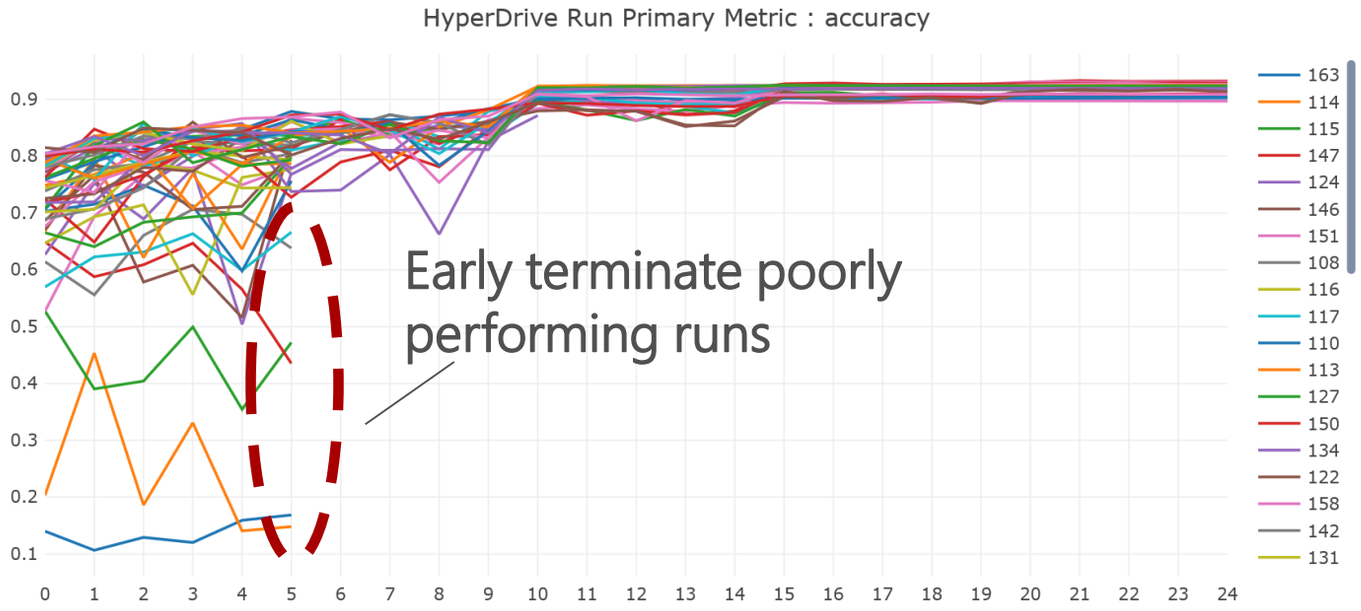
- Grid Sampling
  - 类似于Sklearn grid search
  - **支持**DL **参数**tuning
- Bayesian Sampling
  - **基于**贝叶斯优化理论， **最好 的方式**
  - **只支持**choice, uniform 类型参数

Managing Active Jobs

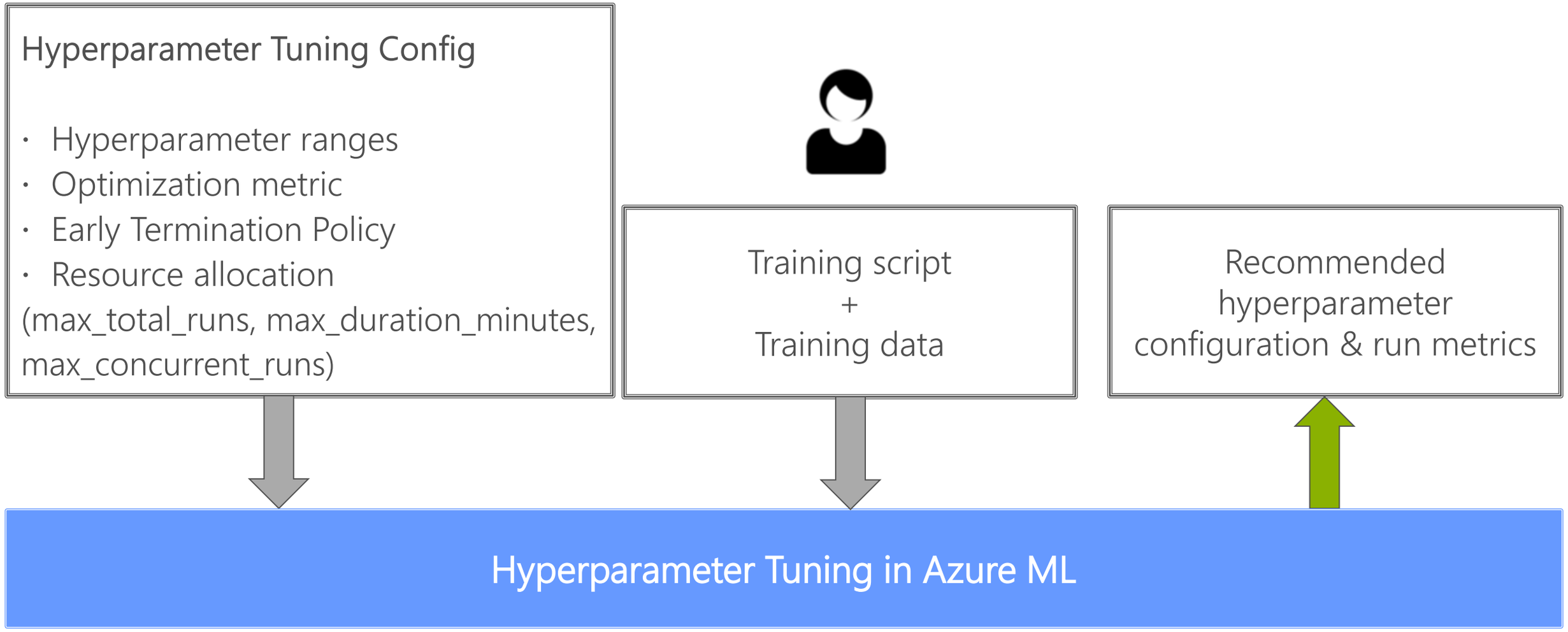
- Evaluate training runs for specified primary metric
- Early terminate poorly performing runs
- Use resources to explore new configurations

Supported early termination policies

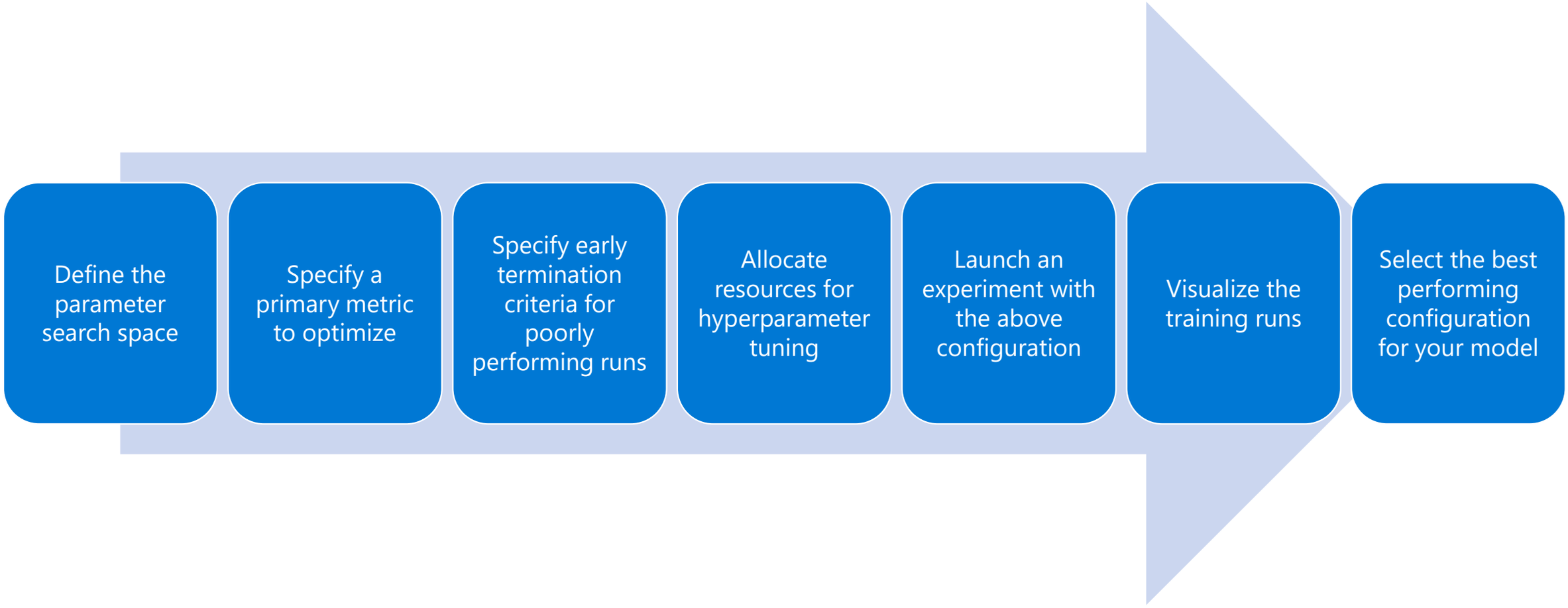
- Bandit (slack factor / amount)
- Median stopping
- Truncation selection (lowest % of runs)



User inputs & system outputs



Azure ML 超参数优化流程



Tuning hyperparameters in the tutorial
代码演示

- 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

Microsoft

Conclusions

Techniques not covered
未涵盖的话题及技巧

- 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 关键收获

- 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

Microsoft

Q & A

Please provide feedback to help us improve!

请提供反馈帮助我们做得更好