



Deep Learning for Time Series Forecasting

Jacob Spoelstra

Content by:

Yijing Chen, Angus Taylor,
Vanja Paunic, Dmitry Pechyoni

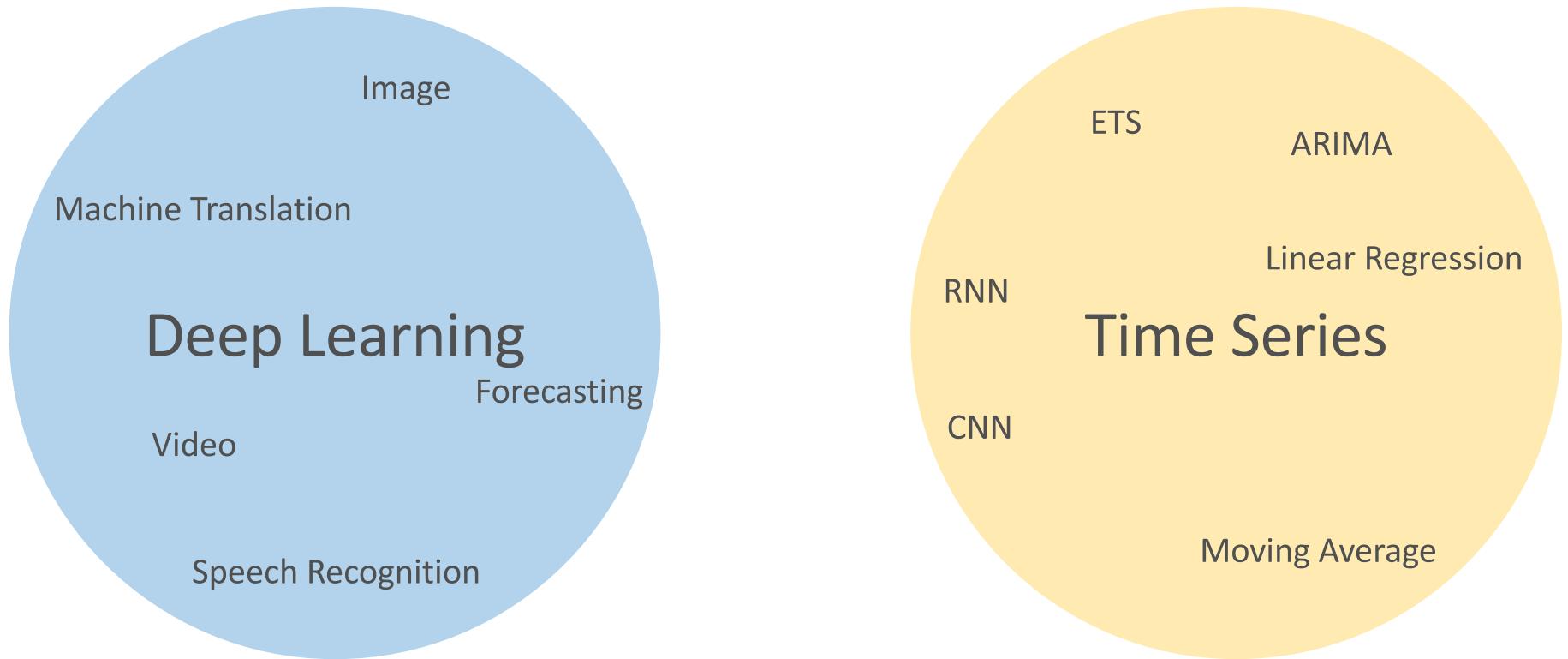
Complete the pre-requisites: aka.ms/dlts-tutorial/pr



Tutorial Introduction

Tutorial Introduction

This tutorial is about the intersection of DL and Time Series



Tutorial Learning Goals

- Understand the principles of recurrent neural networks (RNN) and convolutional neural networks (CNN) for time series forecasting
- Be able to use Keras to implement and train 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

- Beginner to intermediate level
- Familiar with Python
- Working knowledge of neural networks
- Know the basic concepts of machine learning and have some experience of building machine learning models

Agenda

Agenda	Time (mins)
Tutorial Introduction + Pre-requisite Setup	30
Knowledge Recap	20
Introduction to convolutional neural networks (CNN)	40
Hands-on: Quiz 1	30
Introduction to recurrent neural networks (RNN)	30
Lunch & Activity	180
Hands-on: Quiz 2	30
Encoder-decoder RNN model	30
Hands-on: Quiz 3	30
Hybrid models for time series forecasting	30
AutoML	30
Hands-on: Quiz 4	30
Conclusion	10

Hands-on Labs

- Form teams 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: who's going to win the Super Bowl?



Setup

aka.ms/dlts-tutorial/pr





Introduction



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 10 rows represent historical data, grouped under a bracket labeled "History". The last 3 rows represent future data, grouped under a bracket labeled "Future". The "Date" column shows dates from June 4 to June 12, with the remaining three days (June 13-15) marked with question marks to indicate they are forecasts.

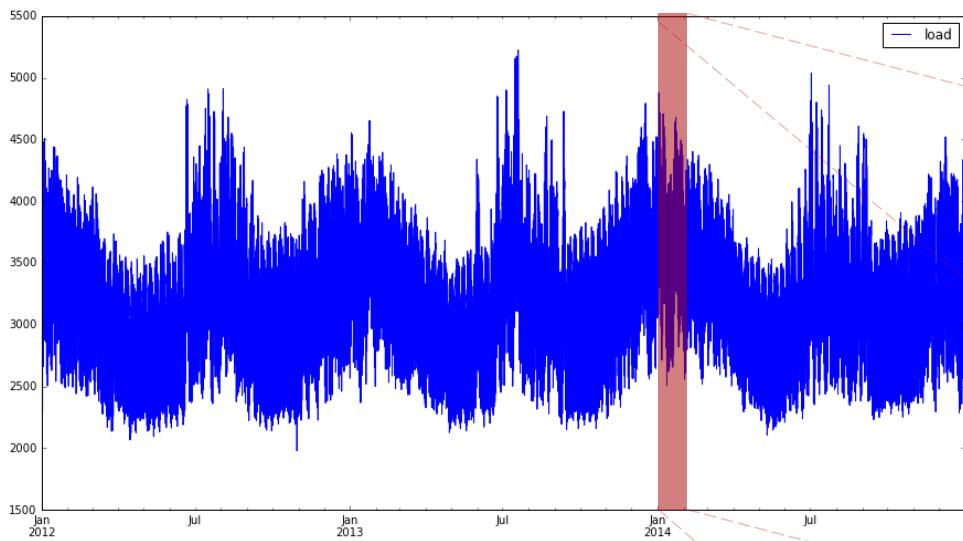
Questions to ask before building forecast model

- 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?
-
-
- What technique/model should I use?

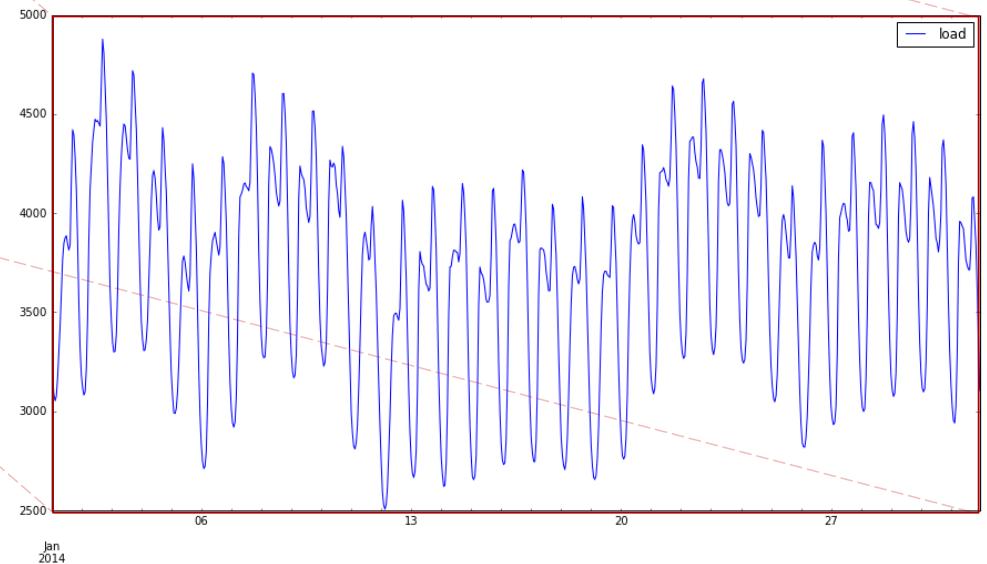
Scenario: energy load forecasting

- Energy grid operators keep the supply and demand of electricity on power grids in balance
- 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



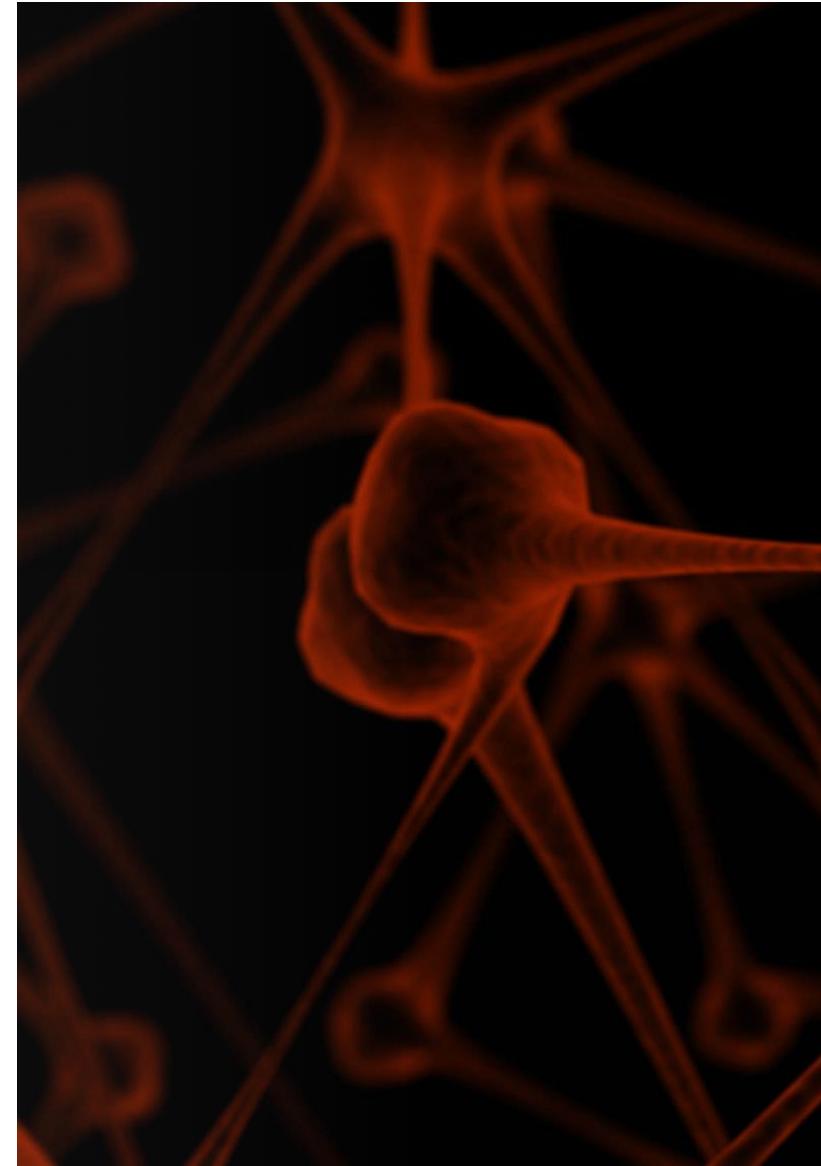
Tao Hong, Pierre Pinson, Shu Fan, Hamidreza Zareipour, Alberto Troccoli and Rob J. Hyndman,
"Probabilistic energy forecasting: Global Energy Forecasting Competition 2014 and beyond",
International Journal of Forecasting, vol.32, no.3, pp 896-913, July-September, 2016.

Why deep learning models?

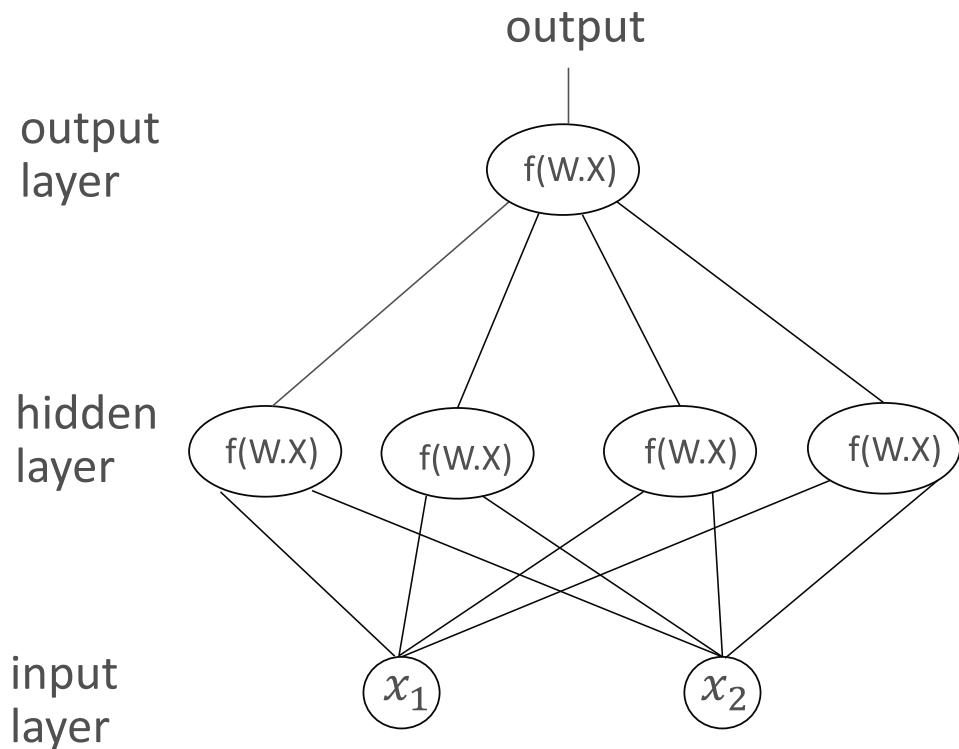
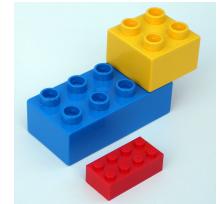
- Deep learning model has been shown perform well in many scenarios
 - 2014 Global Energy Forecasting Competition ([link](#))
 - 2016 CIF International Time Series Competition ([link](#))
 - 2017 Web Traffic Time Series Forecasting ([link](#))
 - 2018 Corporación Favorita Grocery Sales Forecasting ([link](#))
 - 2018 M4-Competition ([link](#))
- Non-parametric
- Flexible and expressive
- Easily inject exogenous features into the model
- Learn from large time series datasets



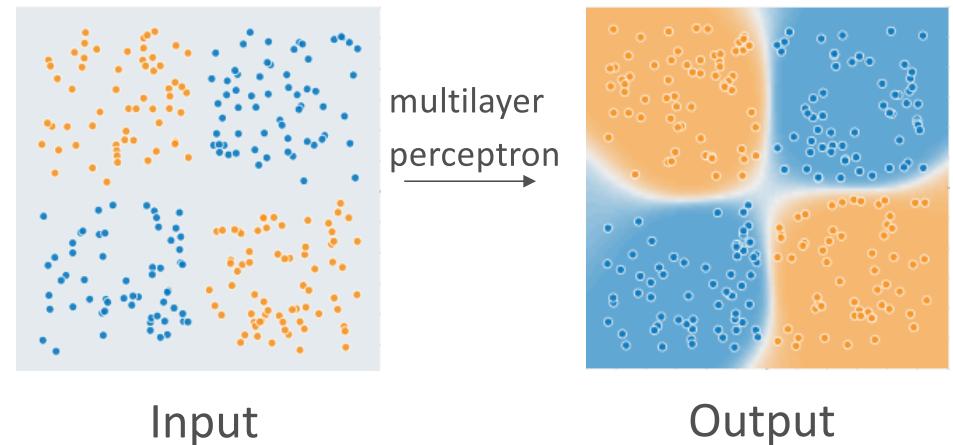
Neural Networks Refresher



Multilayer Perceptron



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

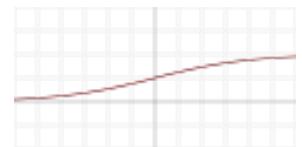


also known as Feed-Forward Neural Network, Artificial Neural Network

Non-Linear Activation Functions

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

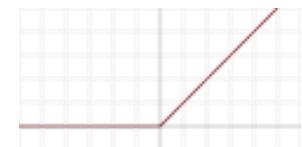
Sigmoid $f(x') = \sigma(x') = \frac{1}{1+e^{-x'}}$



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



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

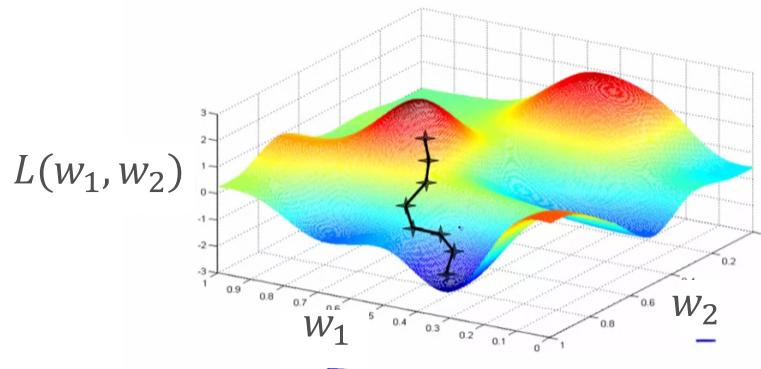


Training: overview

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):

$$\frac{100}{N} \sum_{i=1}^N \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

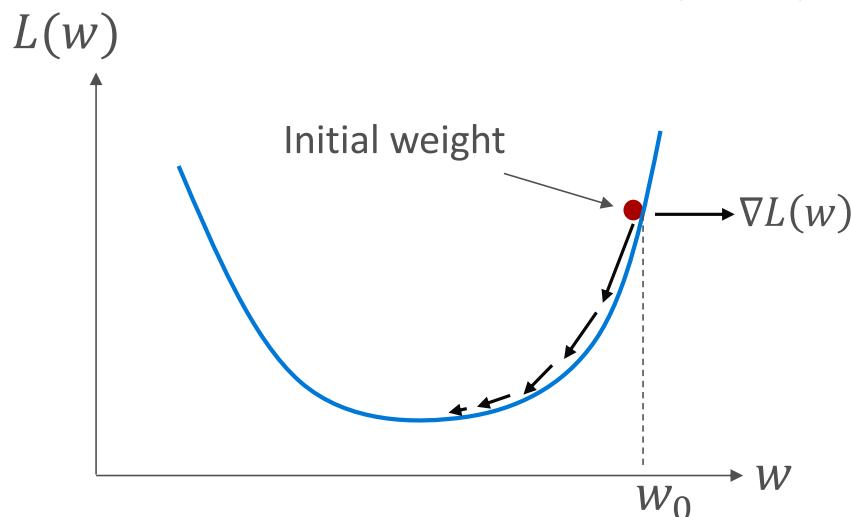
- Symmetric MAPE (sMAPE):

$$\frac{100}{N} \sum_{i=1}^N \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2}$$

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

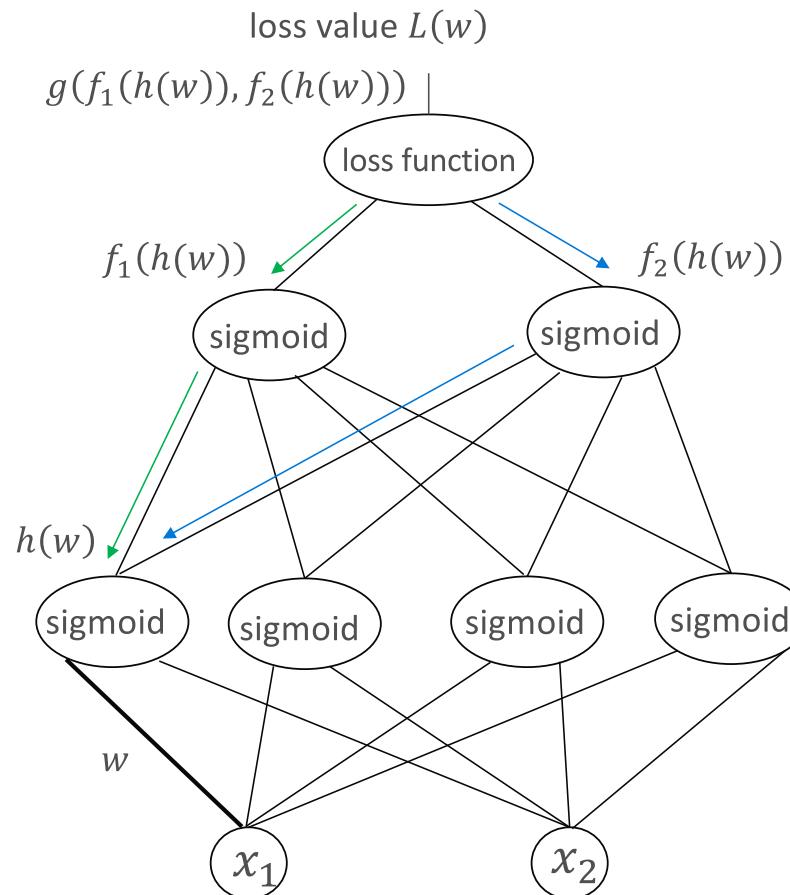
$$w = w - \frac{\alpha}{m} \sum_{i=1}^m \nabla L_i(w)$$

} 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

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

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

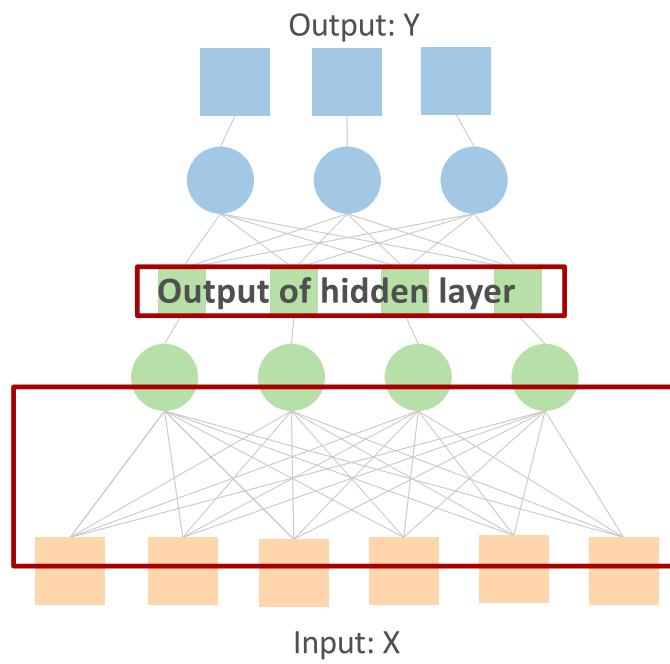
Training tricks

- 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



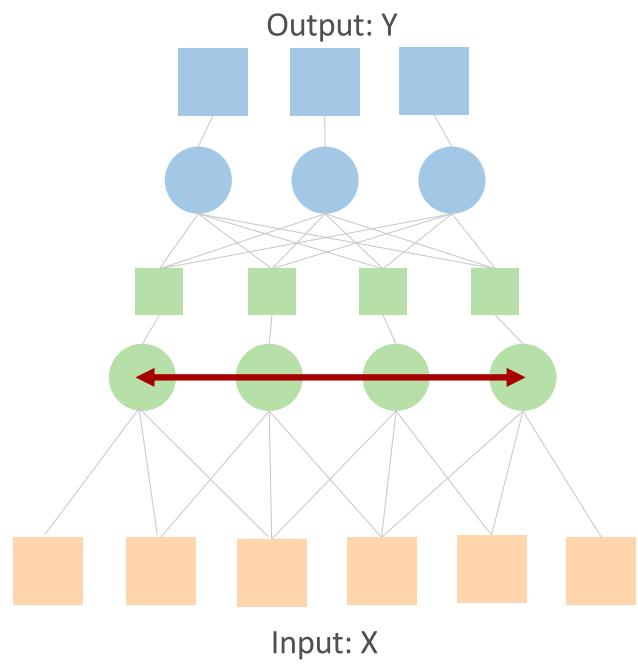
Fully connected:

- 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

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

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

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

1D Convolutions

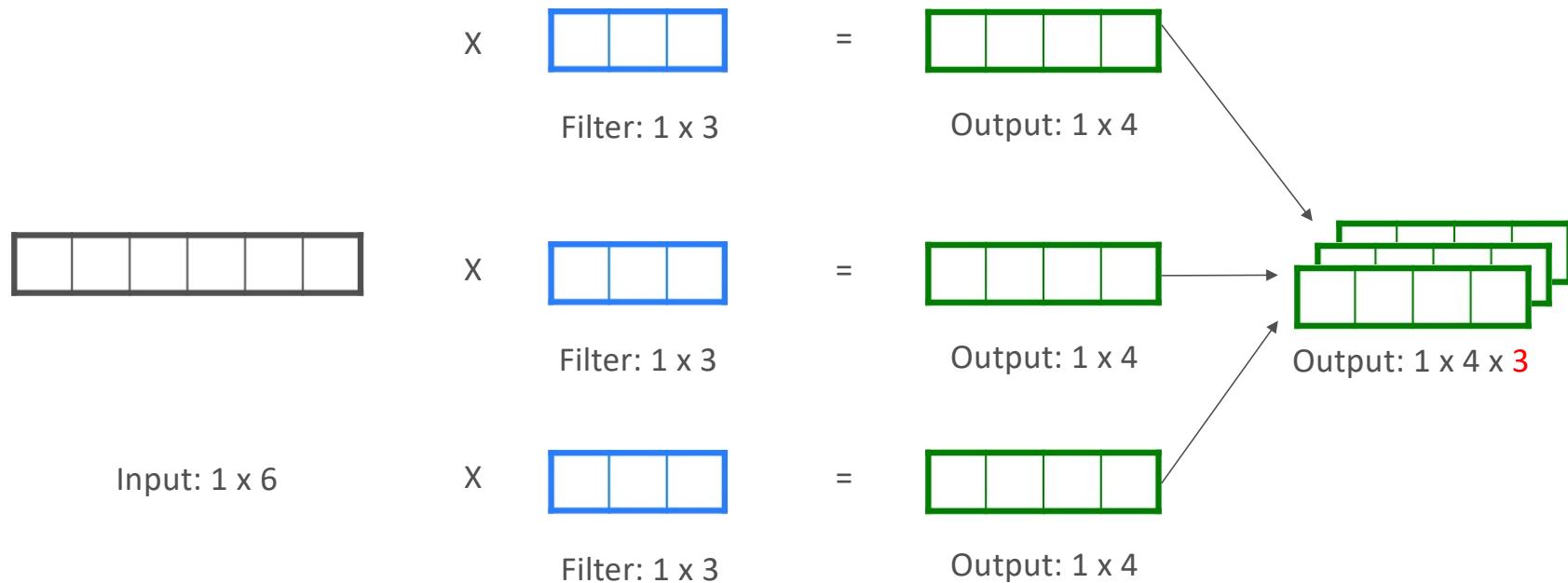
- Filters are trained to detect features in the input sequence
- Features can be detected regardless of where they appear in the input sequence

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

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

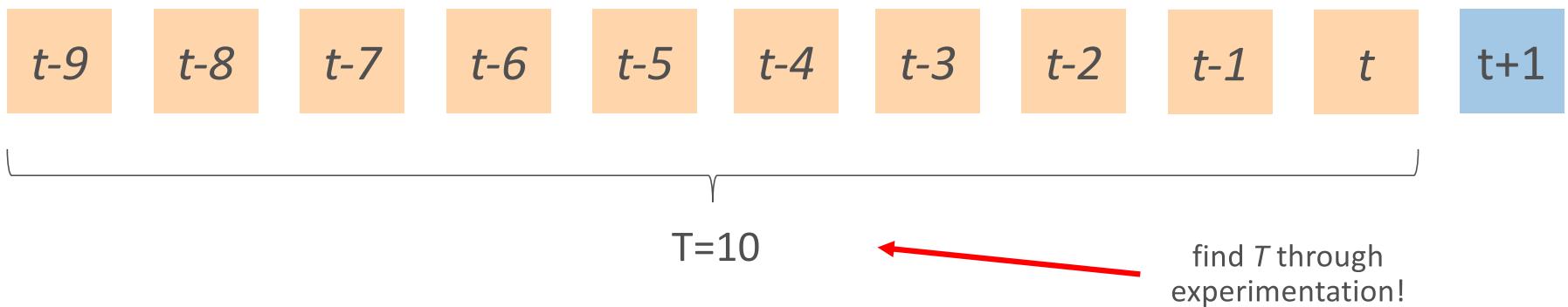
1D Convolutions

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



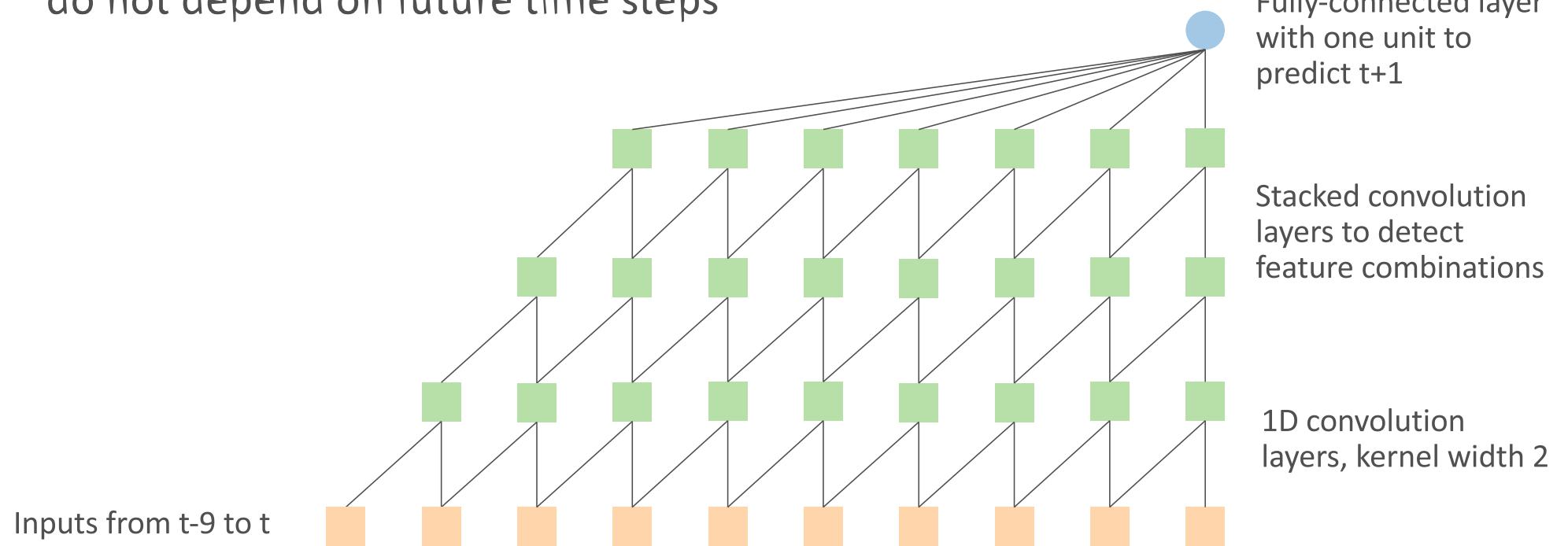
Application to forecasting

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



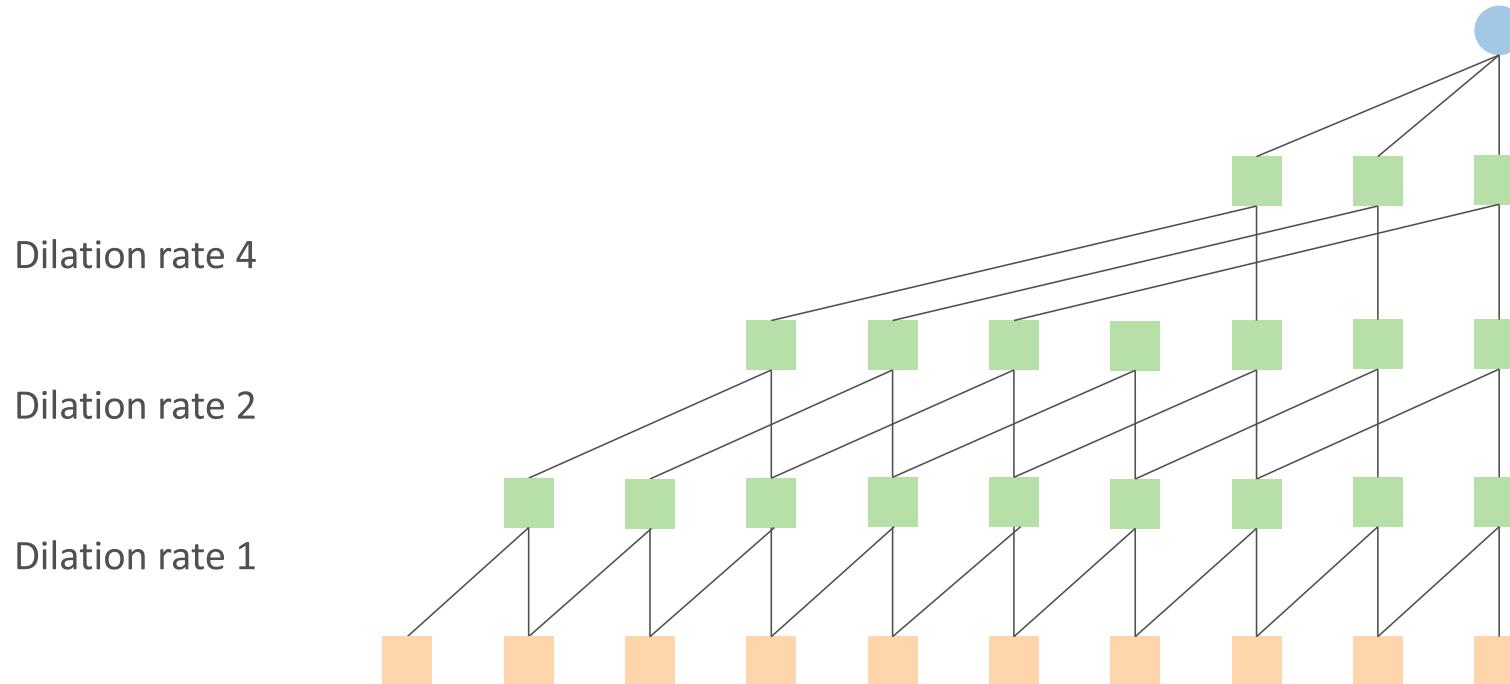
CNNs for forecasting

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



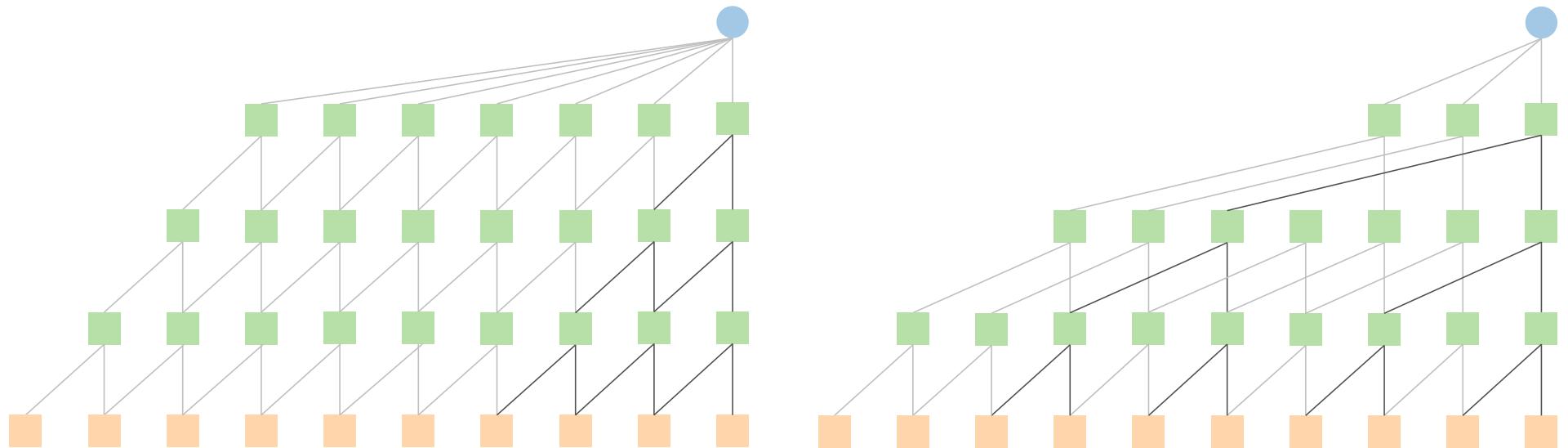
Dilated convolutions

- Skip outputs from previous layers



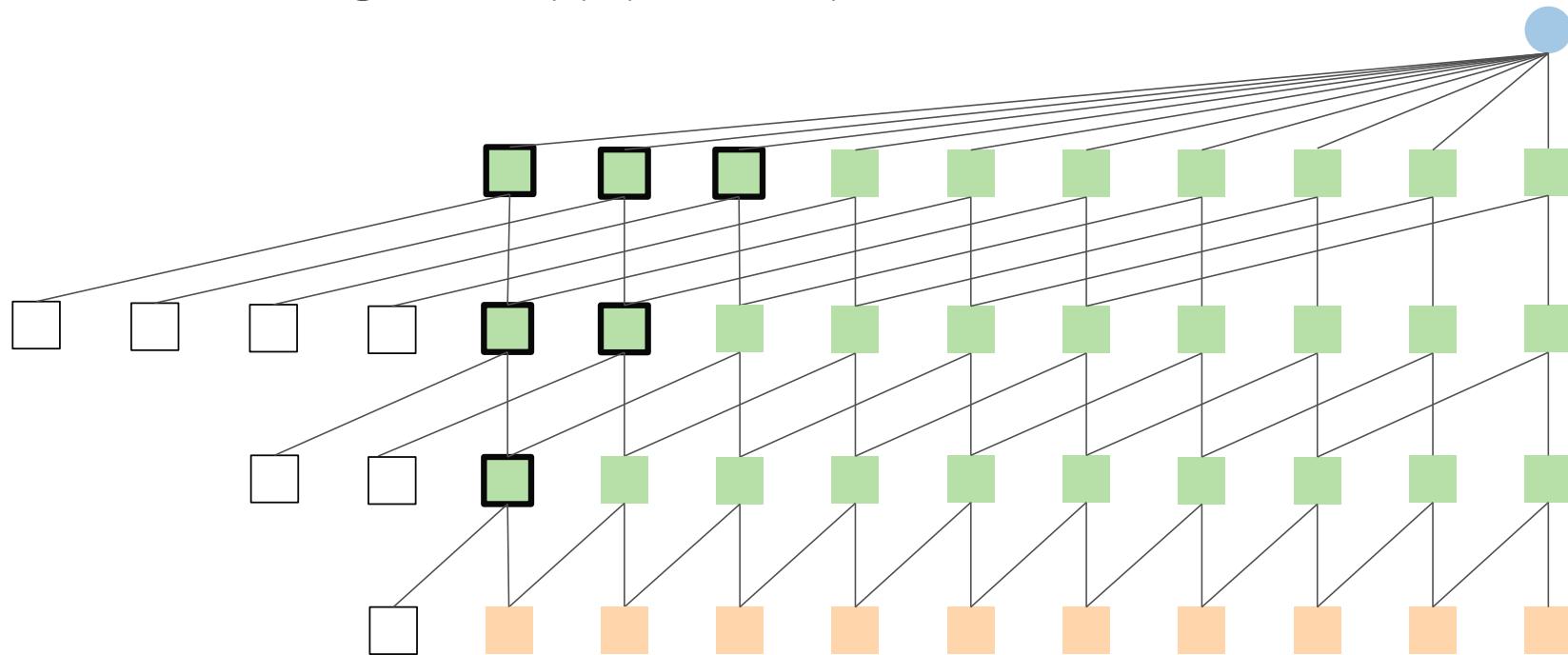
Why dilated convolutions?

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



Causal padding

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



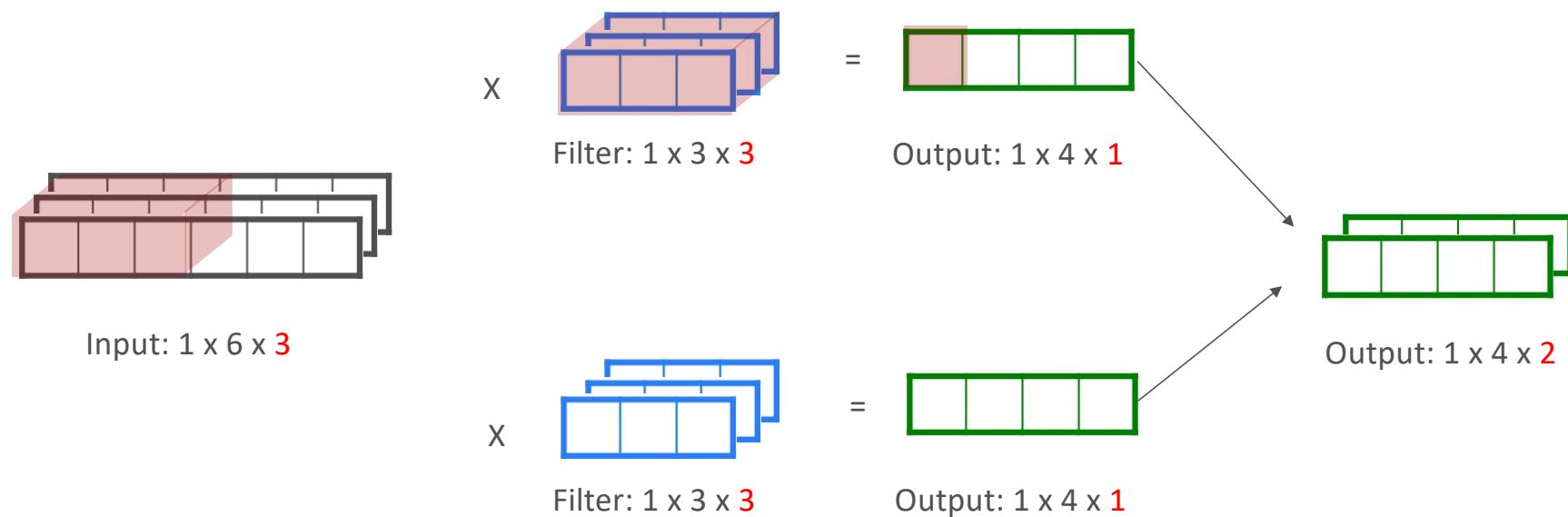
Multivariate time series

- Multiple time series in the input sequence



Multivariate time series

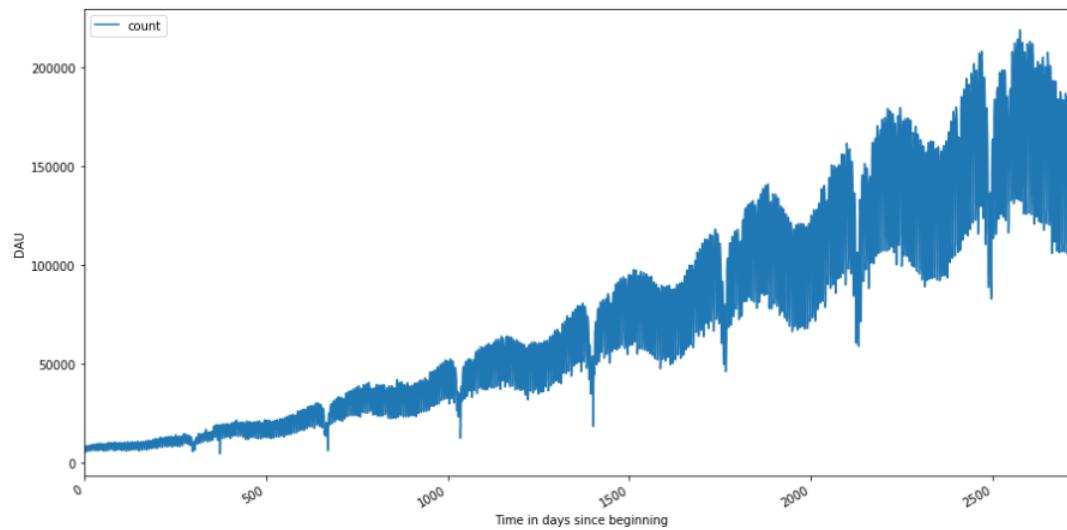
- Example: for 3 input time series use 3 channels¹



¹From color channels in images

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



Hands-on Experiment



Open: 1_CNN_dilated.ipynb notebook

Hands-on Quiz



Open: Quiz_1_CNN.ipynb notebook

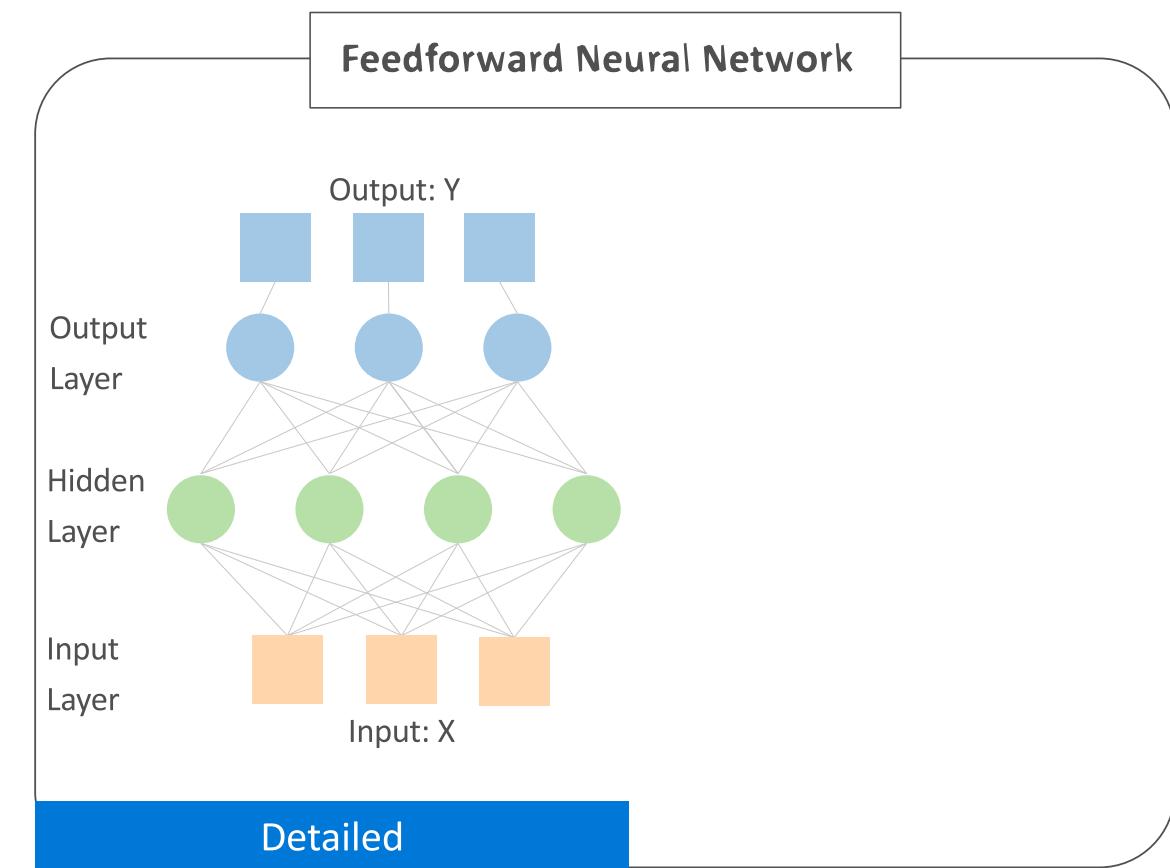


Introduction to Recurrent Neural Networks

Agenda

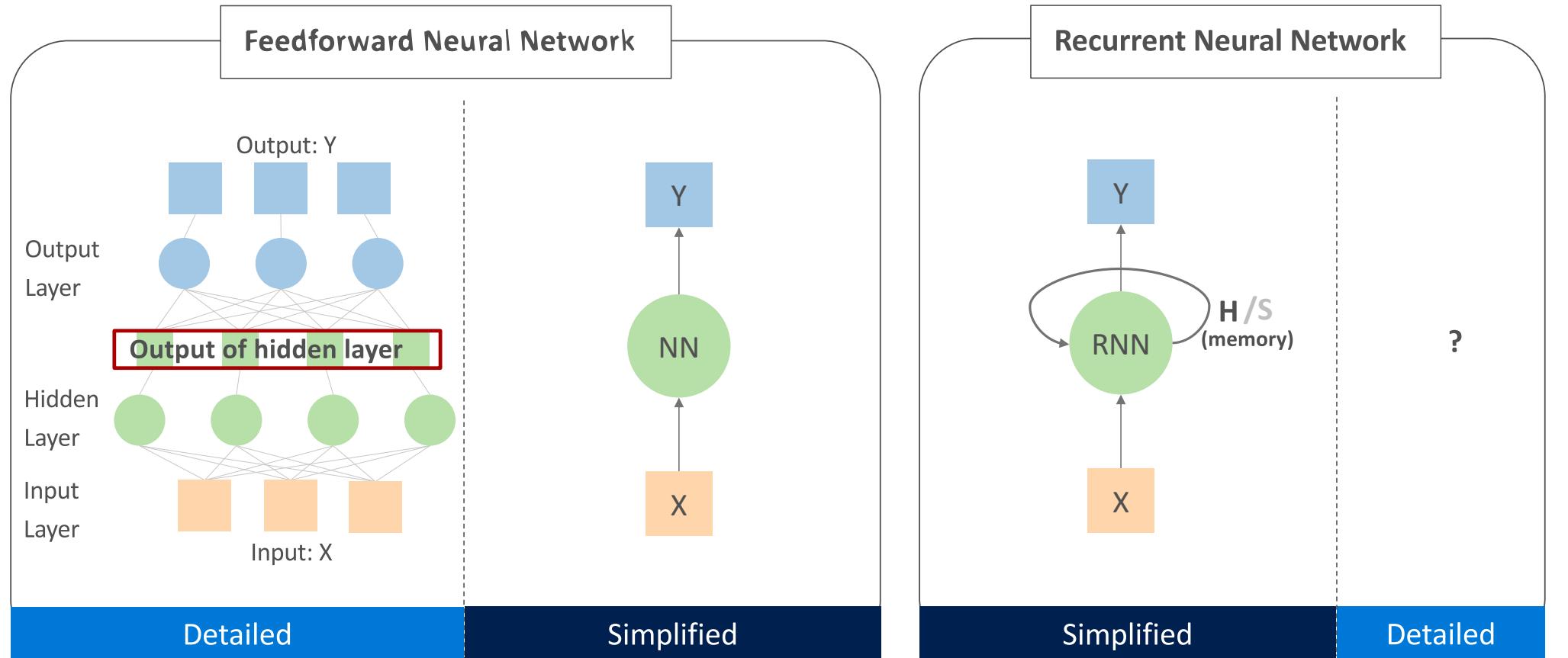
- What are RNNs?
- How RNNs are trained: Backpropagation through time (BPTT)
- Vanilla RNN and its gradient problems
- Other RNN units
 - GRU
 - LSTM
- RNN stacking
- RNN for one step time series forecasting
- Encode-decoder RNN for multi-step time series forecasting

What are RNNs?



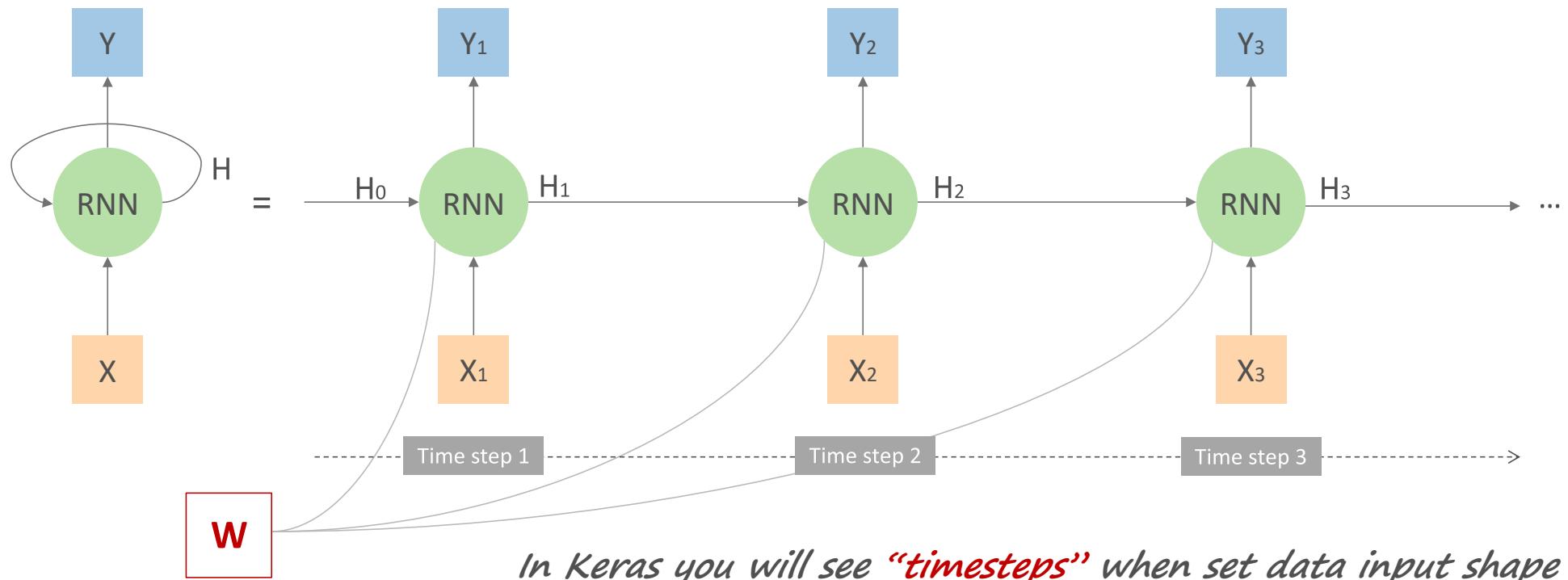
What are RNNs?

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

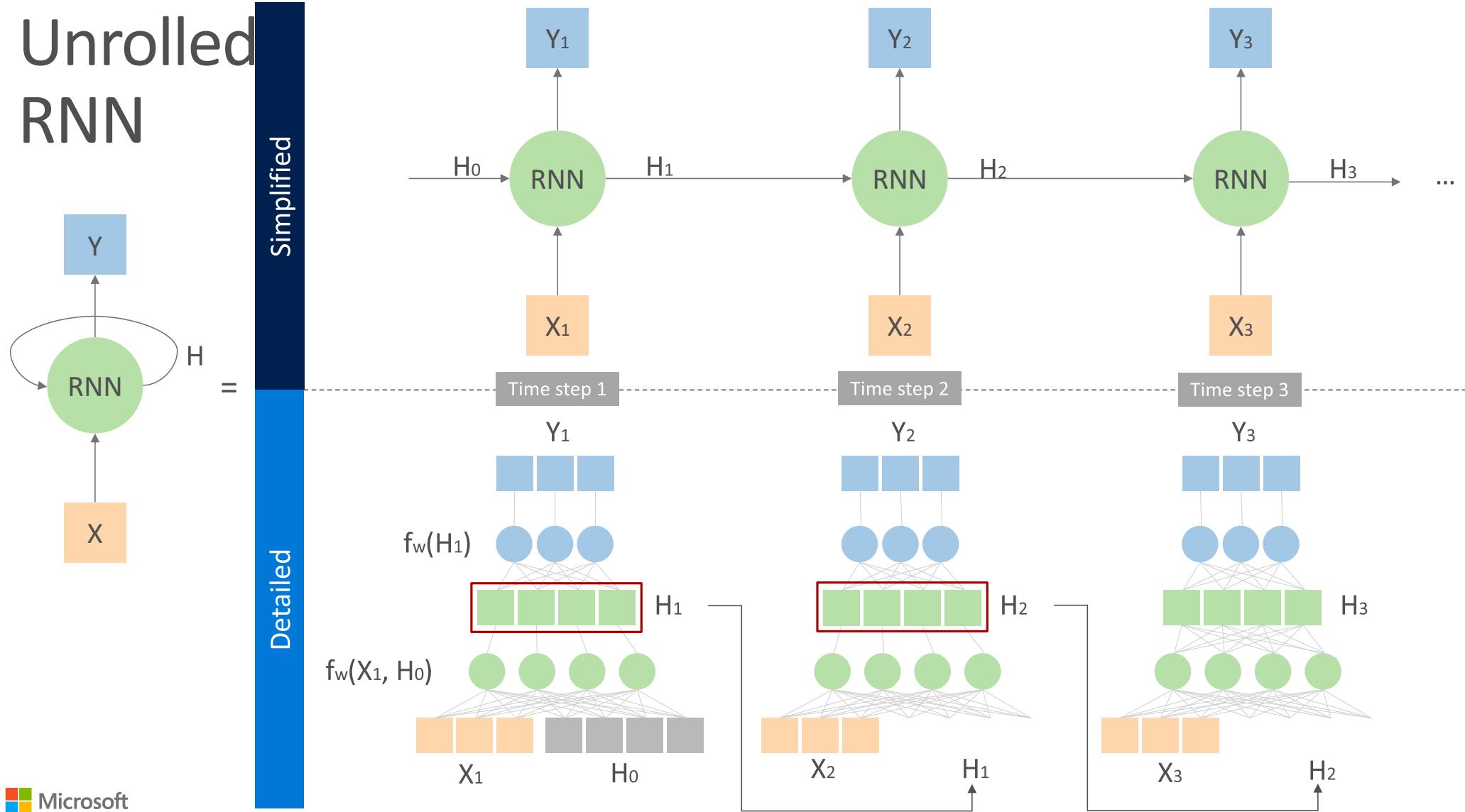


Unrolled RNN

The same weight and bias shared across all the steps



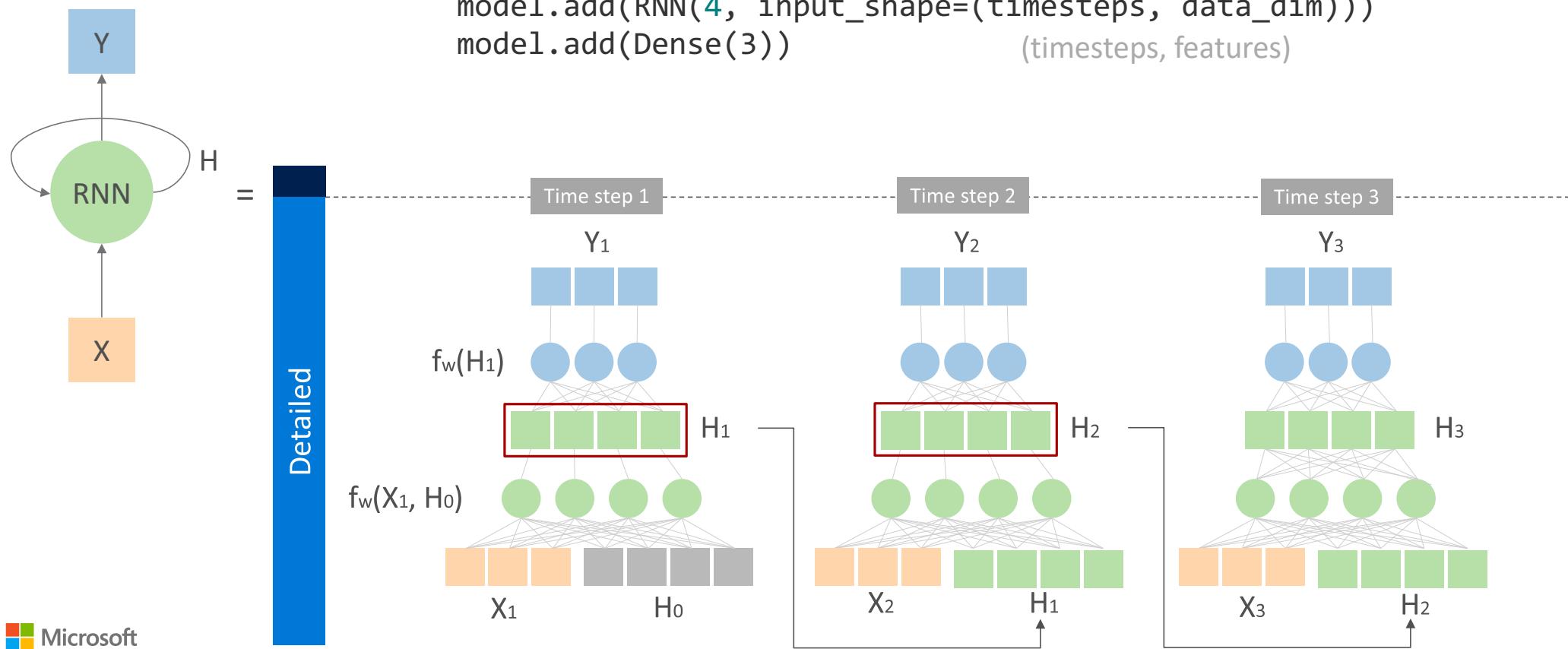
Unrolled RNN



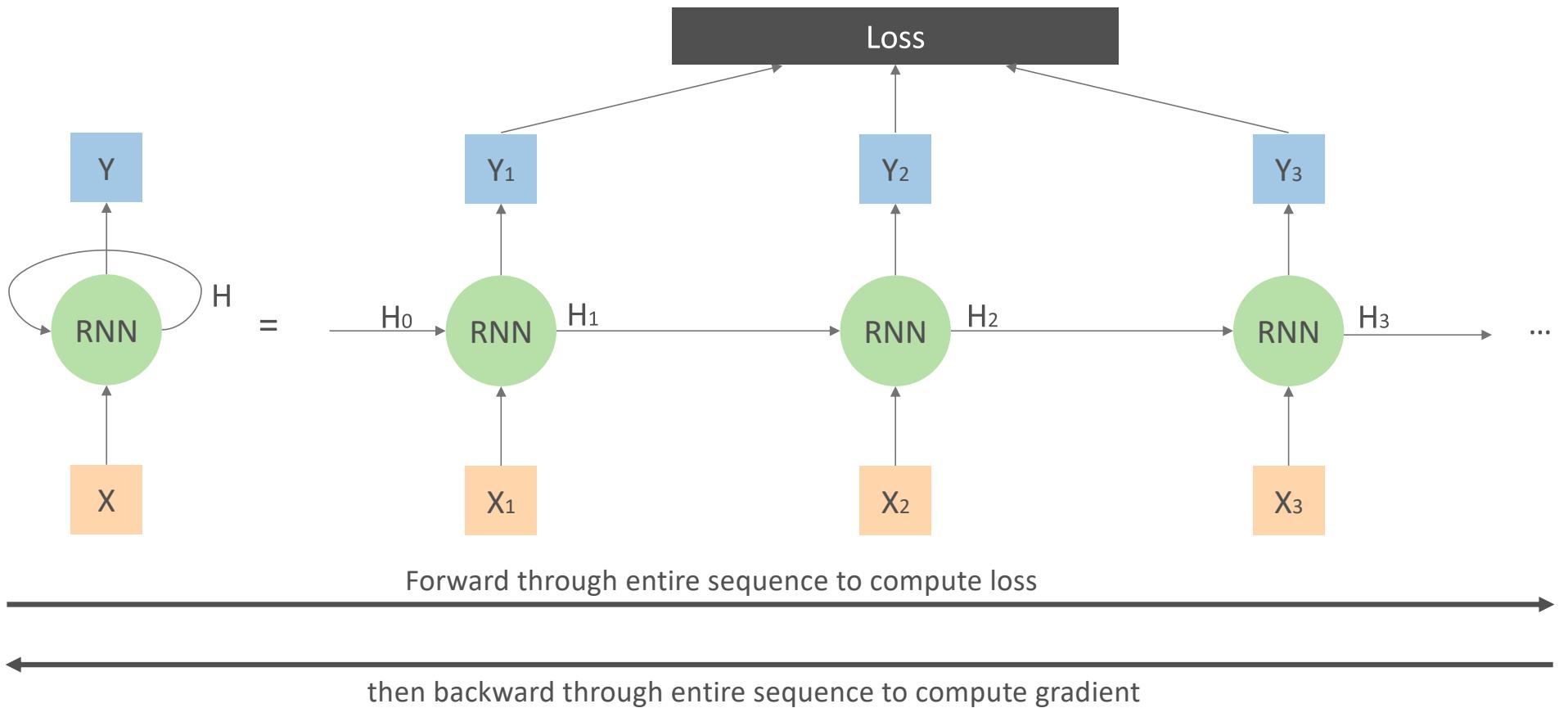
Unrolled RNN

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

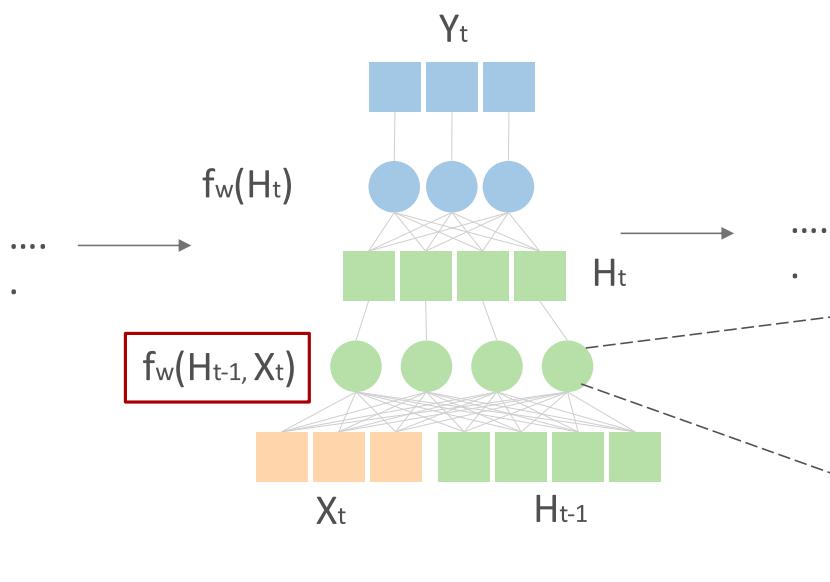
```
model = Sequential()  
model.add(RNN(4, input_shape=(timesteps, data_dim)))  
model.add(Dense(3))  
                                         (timesteps, features)
```



Backpropagation through time (BPTT)



Vanilla RNN



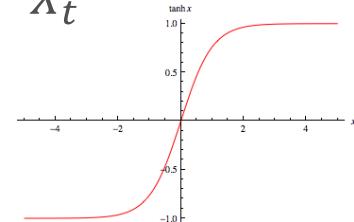
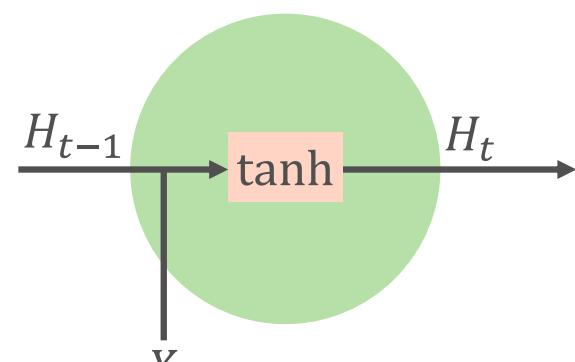
$$H_t = f_W(H_{t-1}, X_t)$$

some function with
parameter W

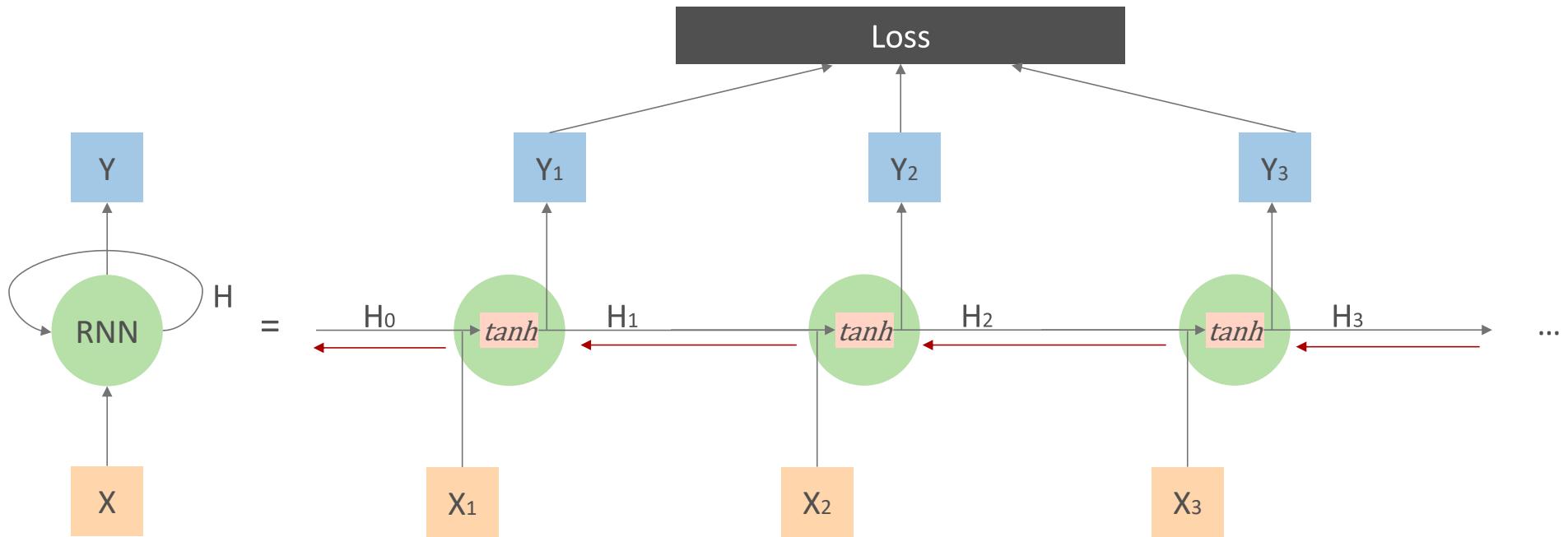
new state old state Input vector at some time step

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

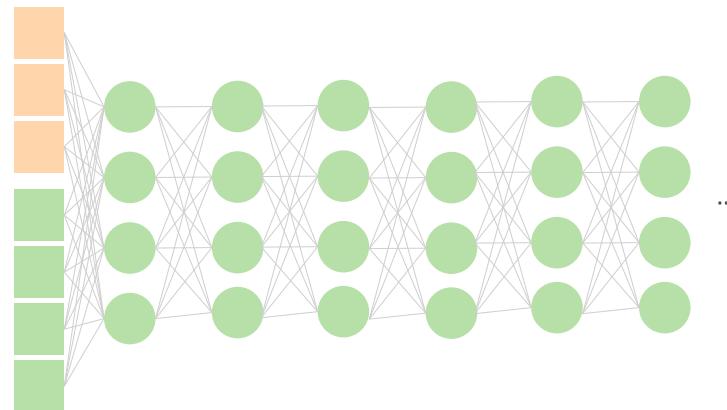


Computing gradient of h_0 involves repeated tanh and many factors of W

Vanilla RNN Gradient Problems

Computing gradient of h_0 involves repeated $tanh$ and many factors of W which causes:

- **Exploding gradient** (e.g. $5*5*5*5*5*5*5*.....$)
- **Vanishing gradients** (e.g. $0.7*0.7*0.7*0.7*0.7*0.7*0.7*.....$)



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
- Heuristic: Gradient clipping
 - Clip the gradient when it goes higher than a threshold

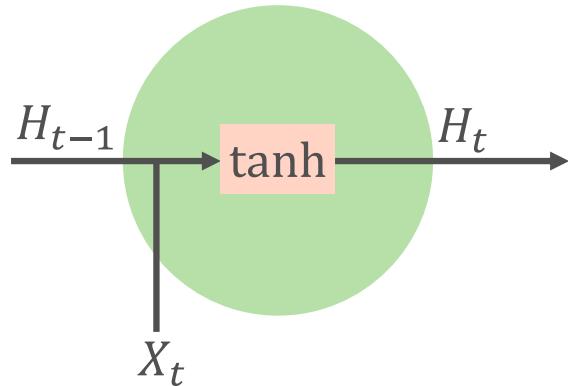
Vanishing Gradient

- 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)

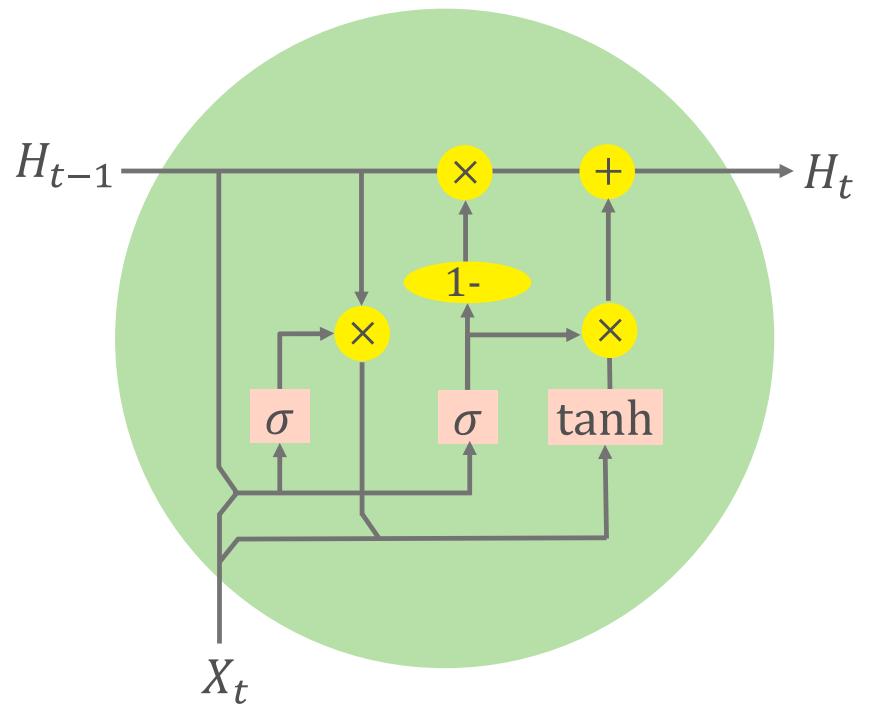
Vanilla RNN:

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

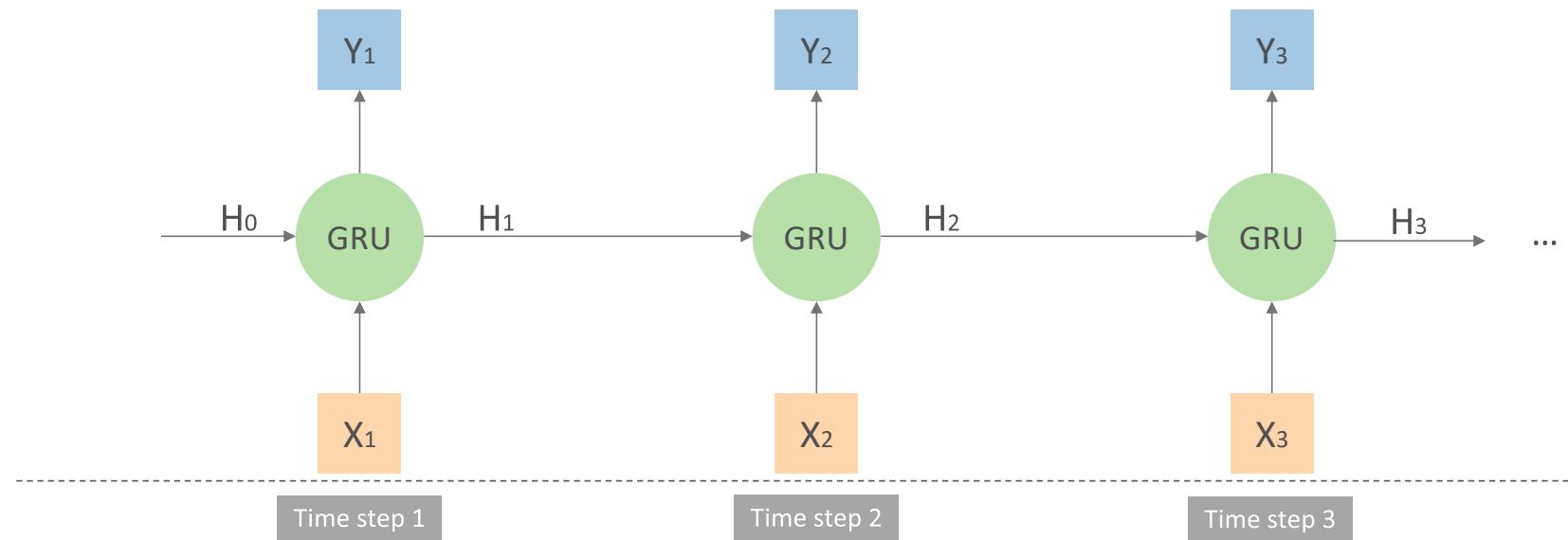


GRU:

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

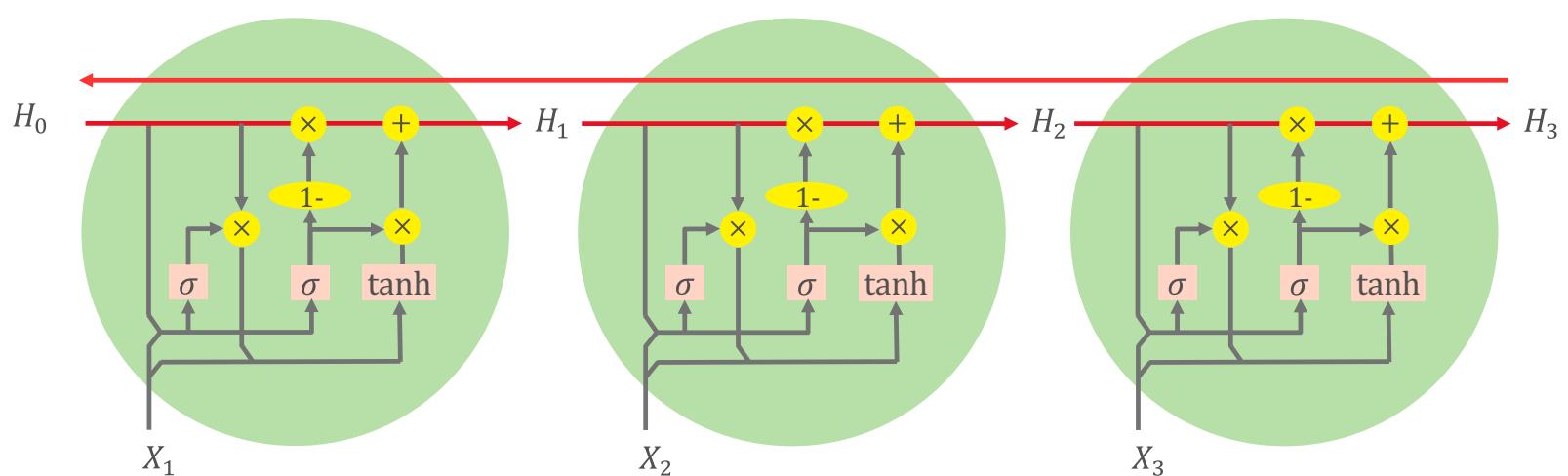


GRU

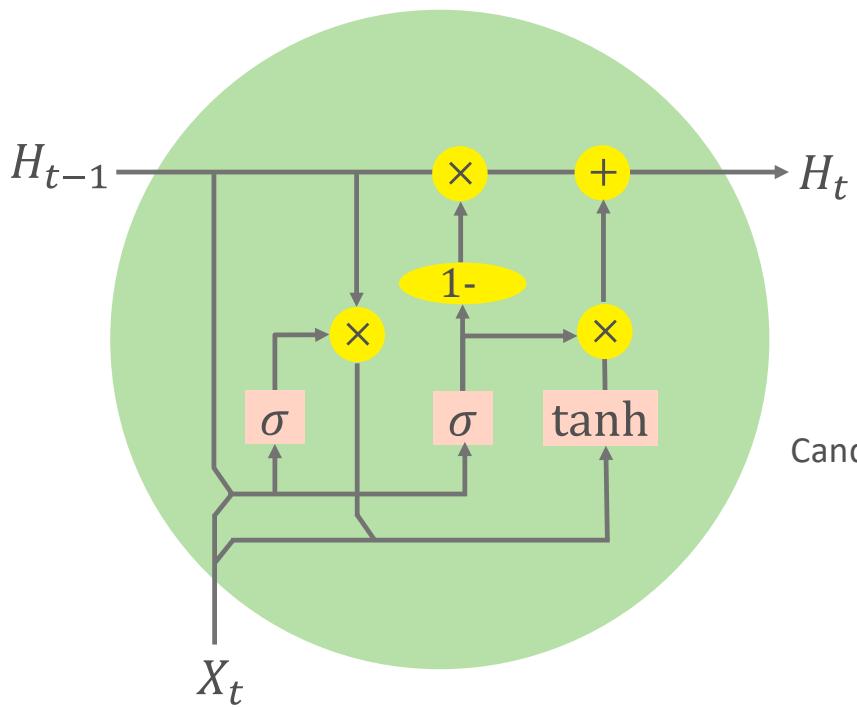


Uninterrupted gradient flow

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



Gated Recurrent Unit (GRU)



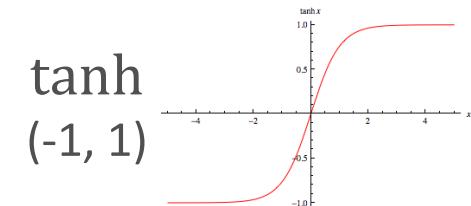
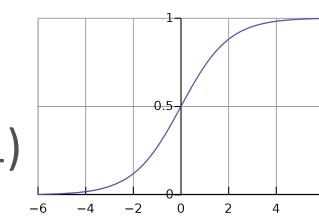
Hidden state: $H_t = (1 - z_t) \times H_{t-1} + z_t \times \tilde{H}_t$

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

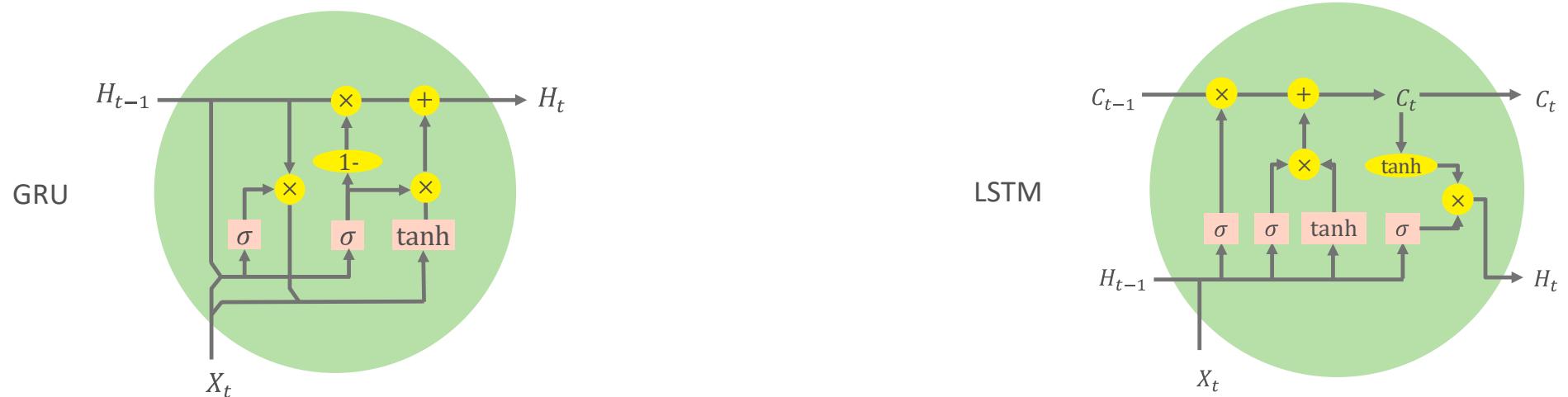
Candidate gates/states: $\tilde{H}_t = \tanh(W \cdot [r_t \times H_{t-1}, X_t])$

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](#)]

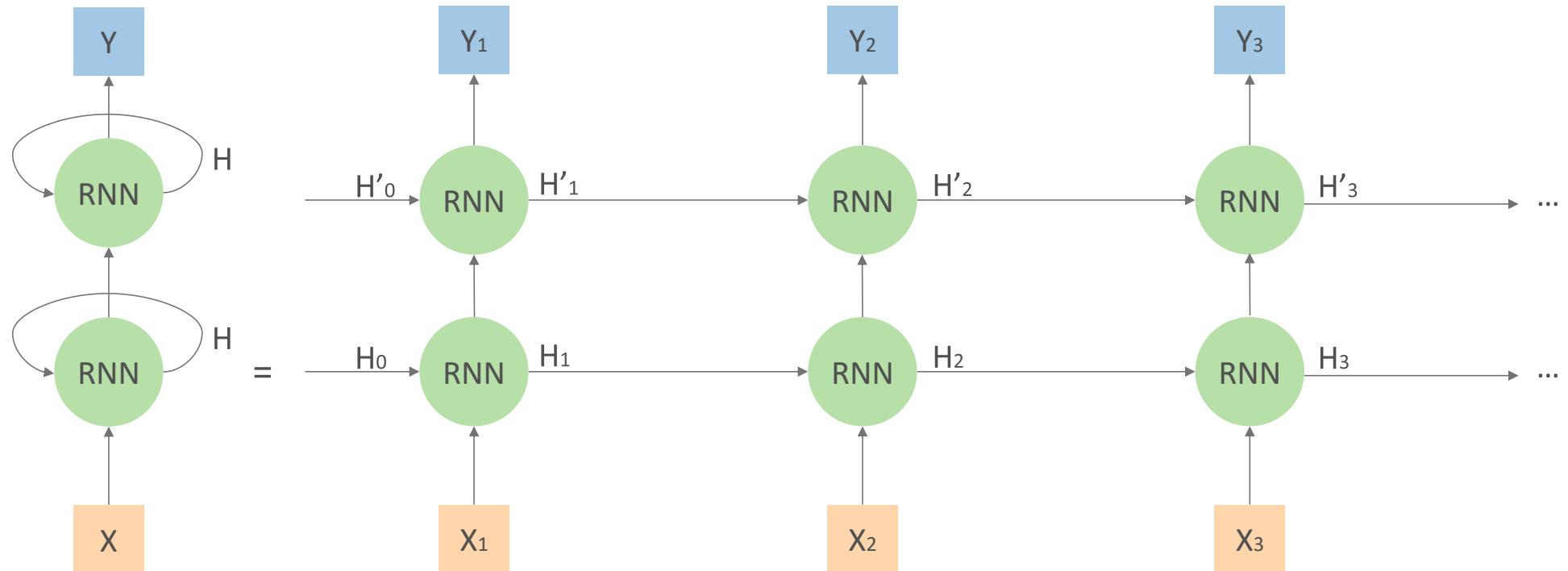


GRU vs LSTM (Long Short Term Memory)



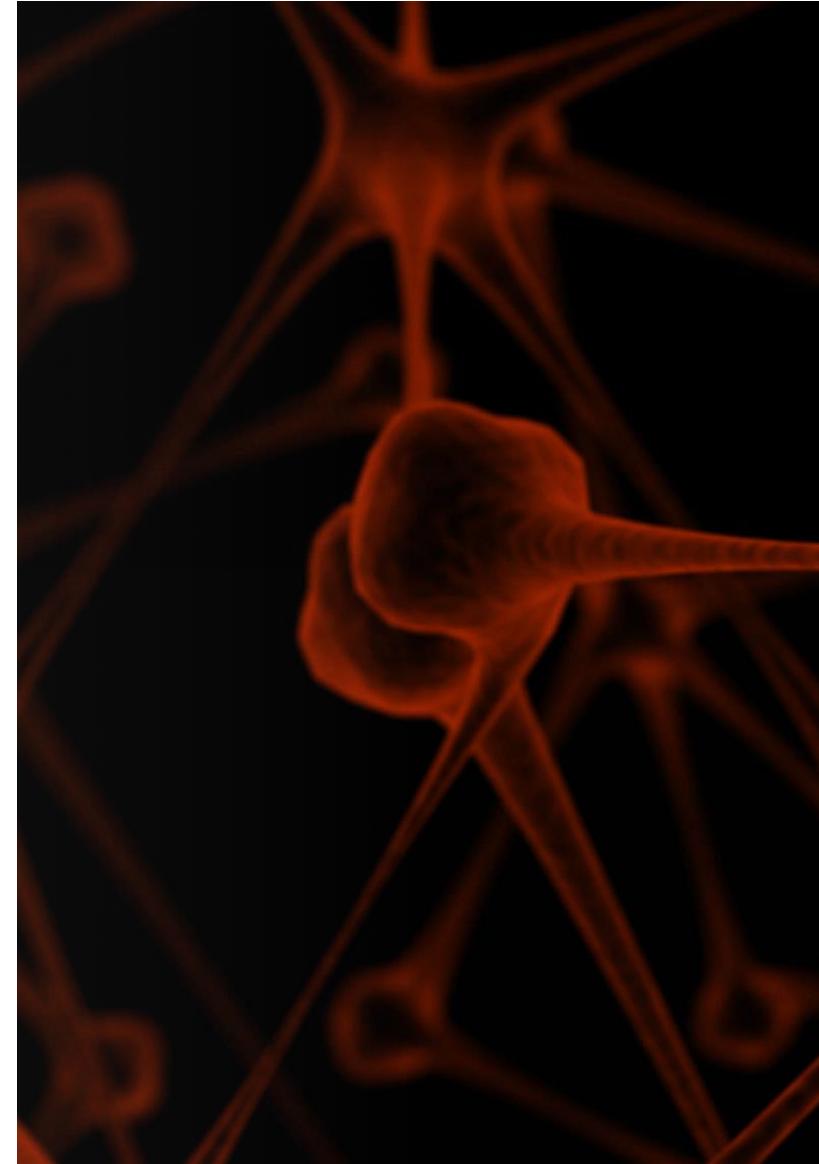
RNN Stacking

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



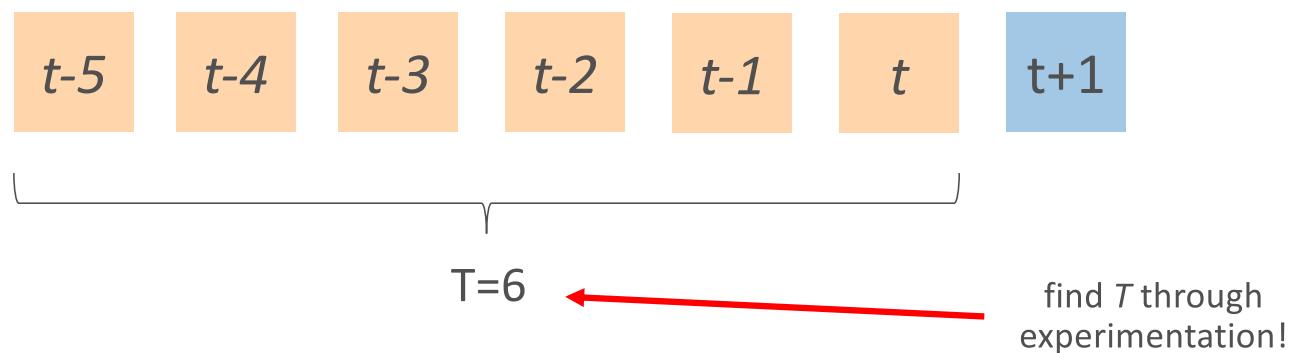
```
model = Sequential()  
model.add(RNN(4, return_sequences=True, input_shape=(timesteps, data_dim)))  
model.add(RNN(4))
```

RNN for one step time series forecasting



One-step forecast

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



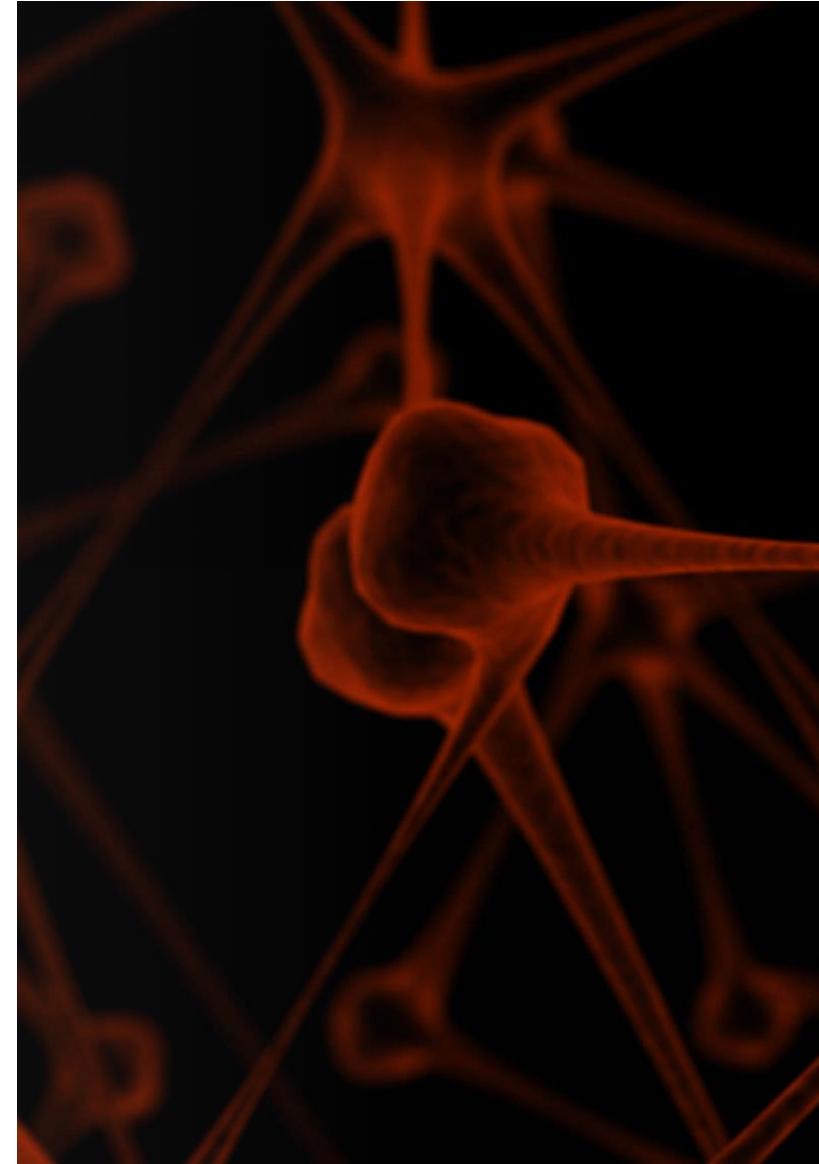
Hands-on Experiment



Open: 2_RNN.ipynb notebook



RNN for multi-step time series forecasting

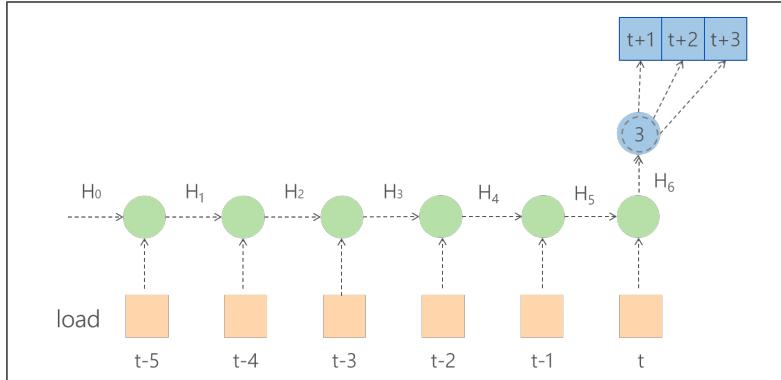


Multi-step forecast

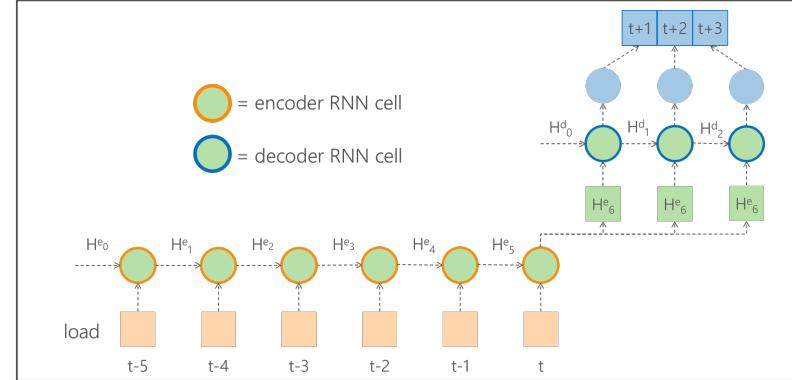
- Assuming we are at time t ...
- ... predict the values at times $(t+1, \dots, t+HORIZON)$...
- ... conditional on the previous T values of the time series



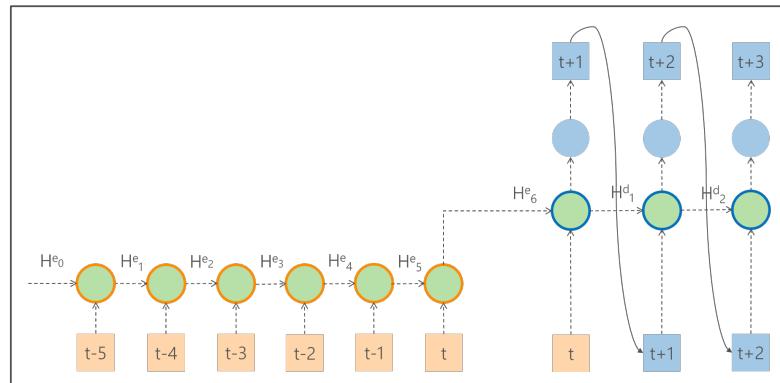
Multi-step forecast



Vector output

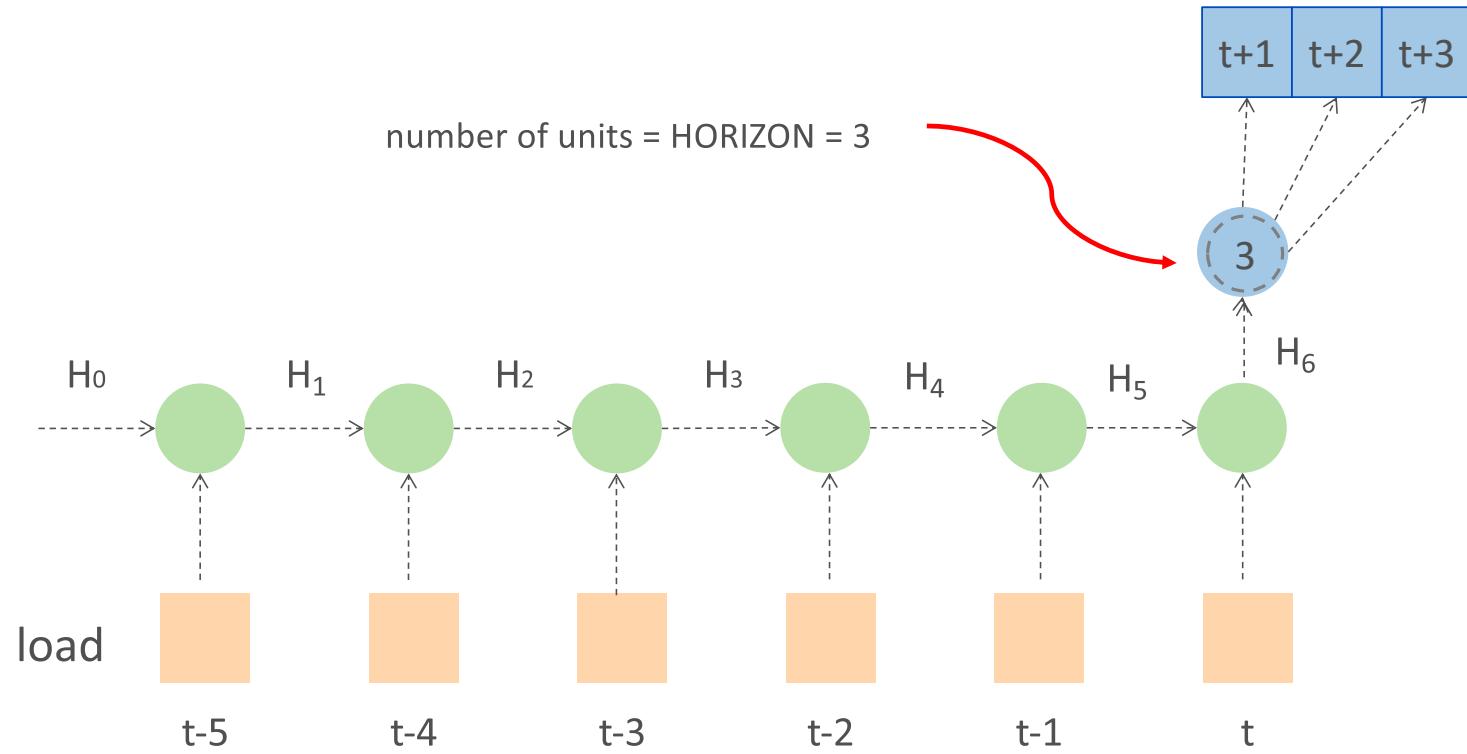


Simple encoder-decoder

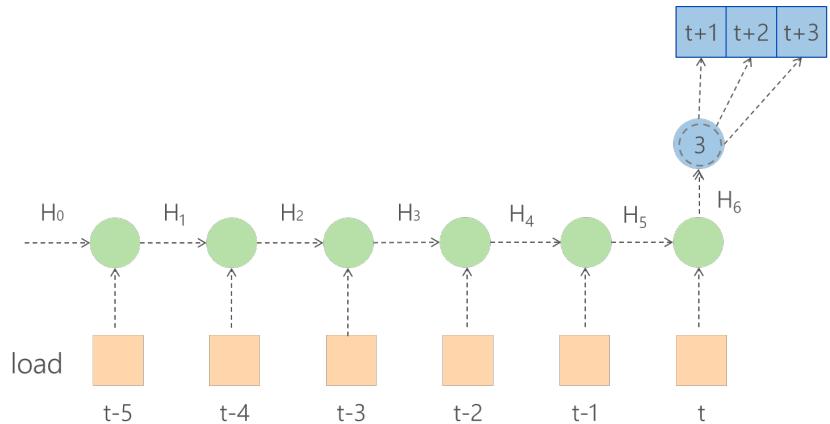


Recursive encoder-decoder

Vector output approach



Vector output approach

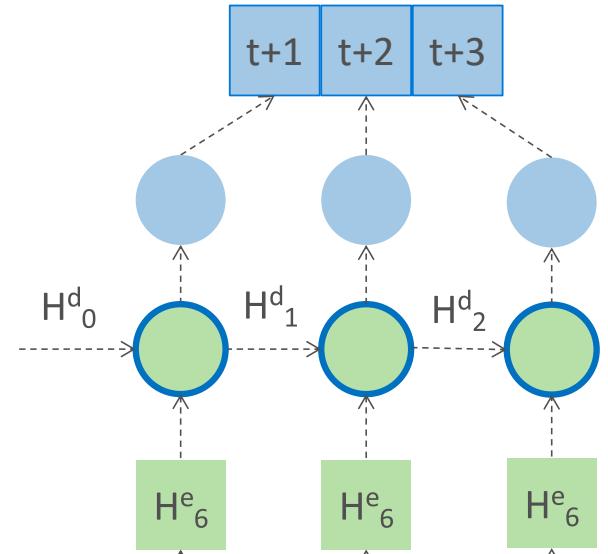
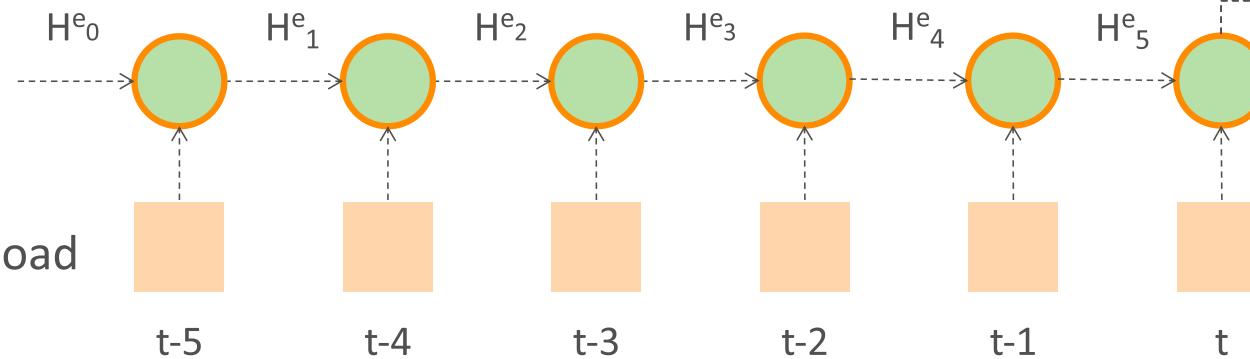


- 👍 Simplest to implement
- 👍 Fastest to train
- 👎 Does not model dependencies between predicted outputs

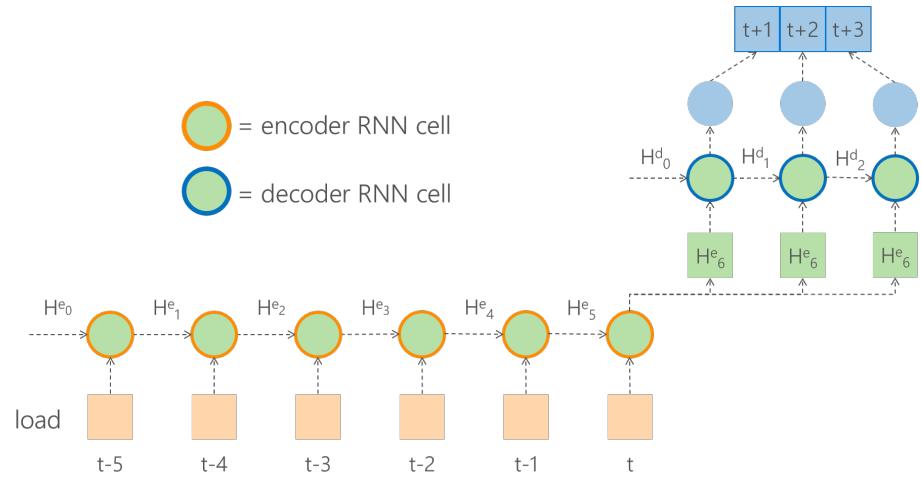
Reference notebook: [multi_step_RNN_vector_output.ipynb](#)

Simple encoder-decoder

- = encoder RNN cell
- = decoder RNN cell



Simple encoder-decoder



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

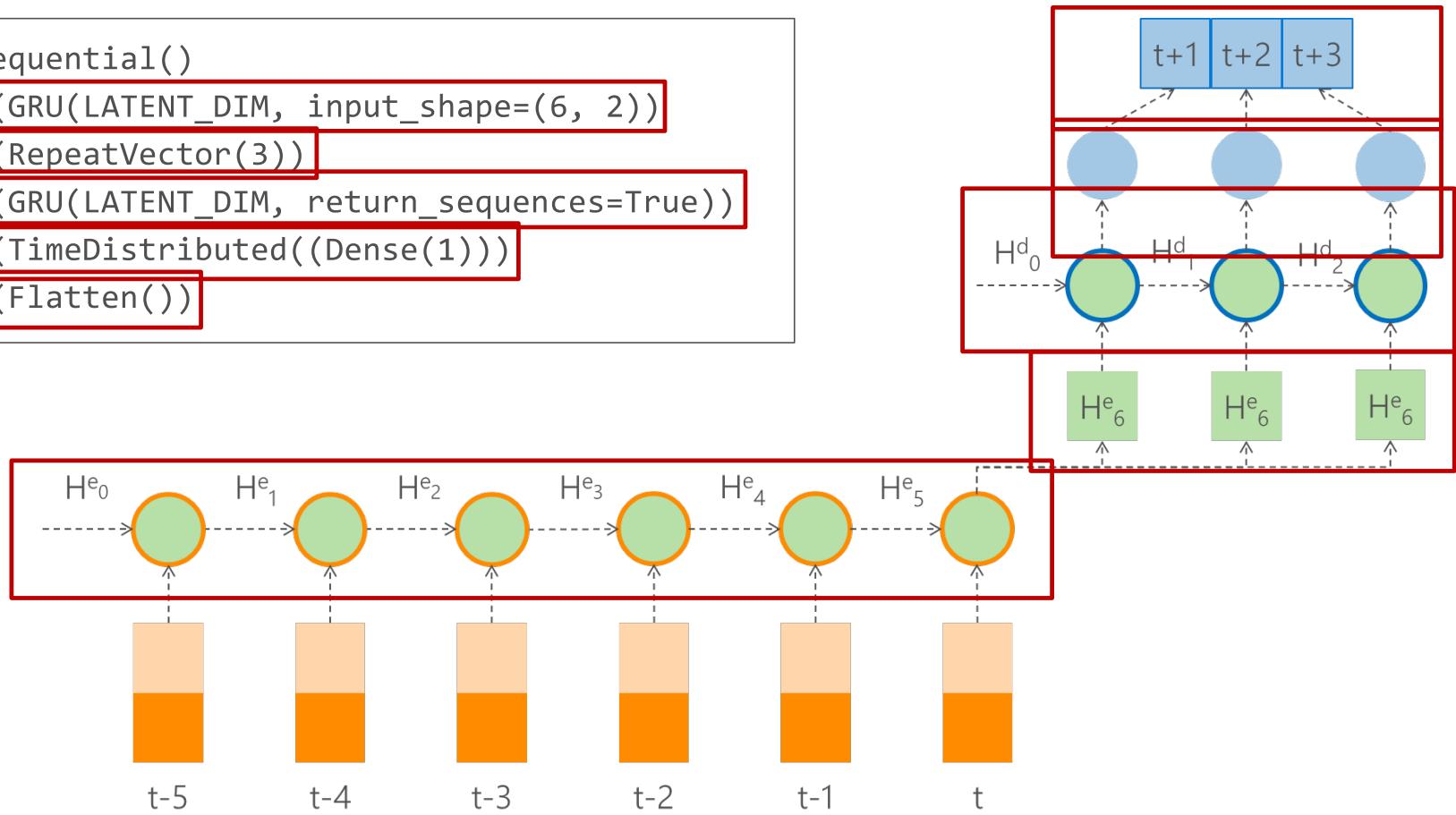
Hands-on Experiment



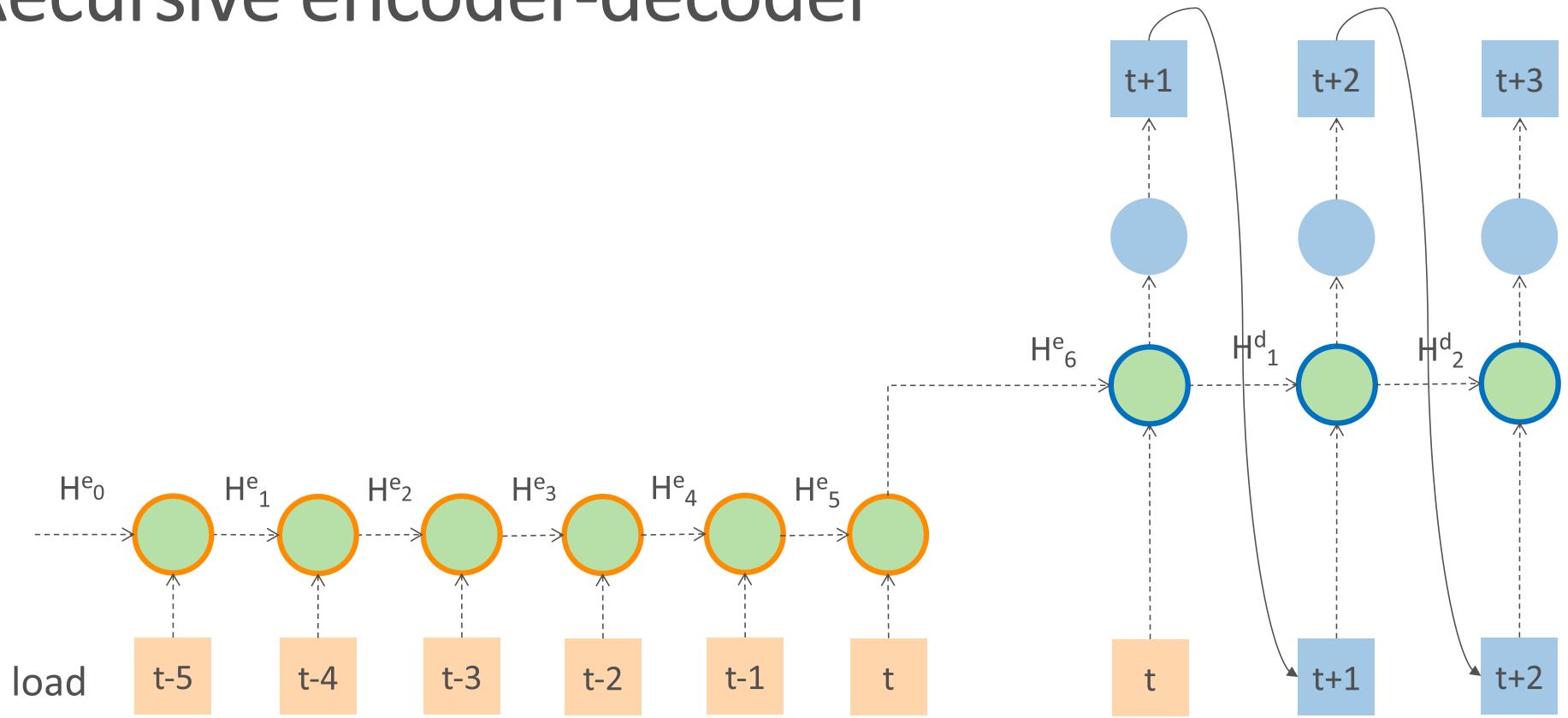
Open: 3_RNN_encoder_decoder.ipynb notebook

Simple encoder-decoder

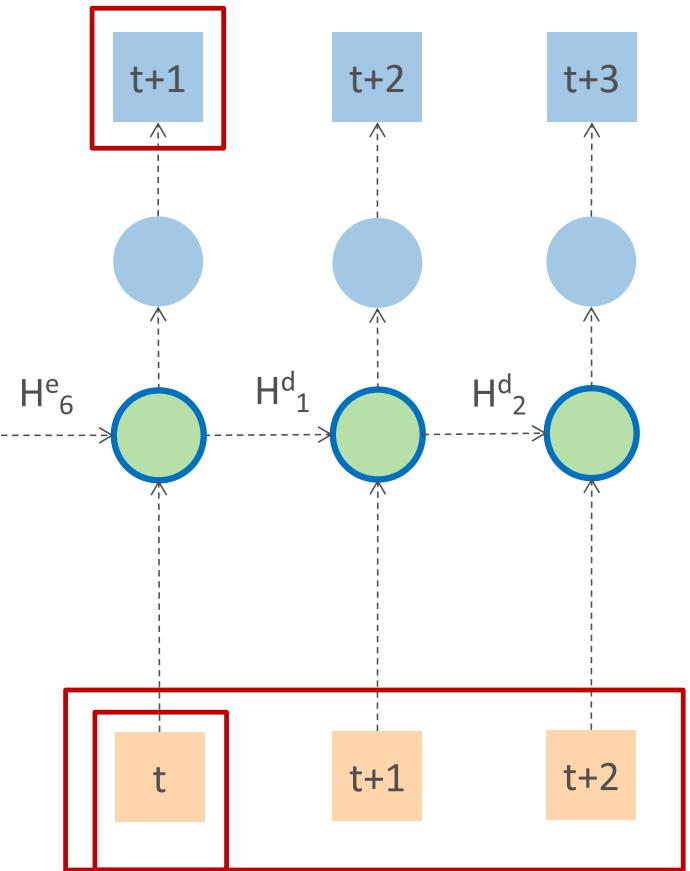
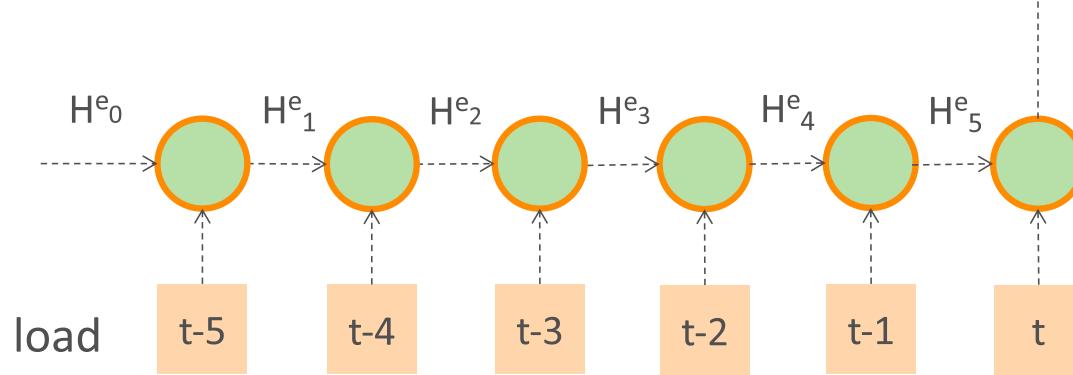
```
model = Sequential()  
model.add(GRU(LATENT_DIM, input_shape=(6, 2)))  
model.add(RepeatVector(3))  
model.add(GRU(LATENT_DIM, return_sequences=True))  
model.add(TimeDistributed(Dense(1)))  
model.add(Flatten())
```



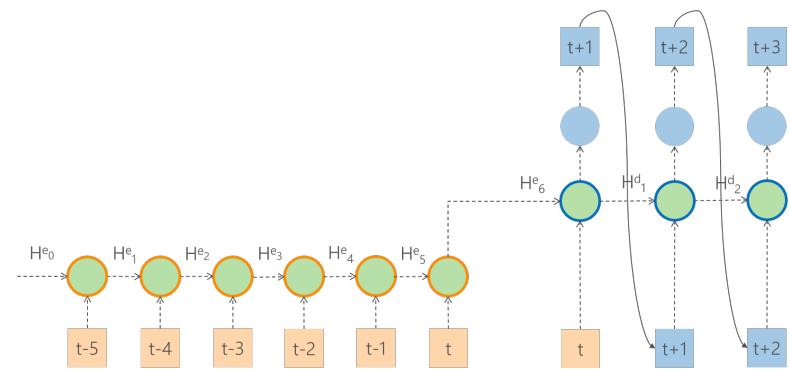
Recursive encoder-decoder



Recursive encoder-decoder (training with teacher forcing)



Recursive encoder-decoder

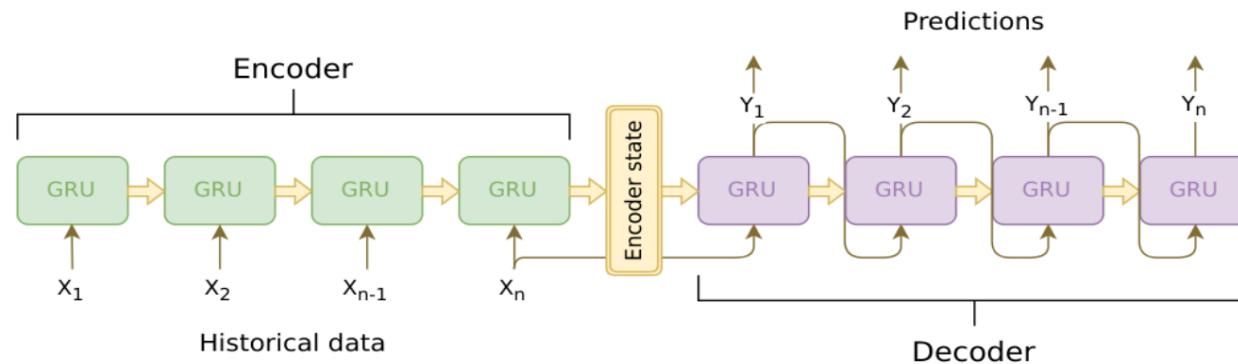


- 👍 Tries to capture dependencies between forecasted time steps through recursive decoder
- 👍 Variable length output (can generate forecasts of variable horizons)
- 👎 More difficult to implement
- 👎 Slower to train and slower to generate forecasts
- 👎 Error propagation due to recursive decoder

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_3_RNN_encoder_decoder.ipynb notebook



Hybrid models for time series forecasting

Agenda

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

ES-RNN (Exponential Smoothing-RNN)

Slawek 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

$$\hat{y}_{t+1|t} = \alpha y_t + \alpha(1 - \alpha)y_{t-1} + \alpha(1 - \alpha)^2 y_{t-2} + \dots$$

learned parameter
↓

Weighted average, where weight decrease exponentially as observations get older.

Weighted average form:

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

↑ ↑ ↑
forecast of actual forecast of
next value current value current value

Component form:

Forecast: $\hat{y}_{t+1|t} = l_t$

Smoothing: $l_t = \alpha y_t + (1 - \alpha)l_{t-1}$
level

In Simple Exponential Smoothing, level is the only component.

Exponential Smoothing (ES)

Forecasting: $\hat{y}_{t+1|t} = l_t \cdot s_{t+1}$ ← seasonality term
(multiplicative method)

Smoothing: $l_t = \alpha \frac{y_t}{s_t} + (1 - \alpha)l_{t-1}$

Seasonal: $s_{t+m} = \gamma \frac{y_t}{l_t} + (1 - \gamma)s_t$

↑
↑
↑
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

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

Smoothing: $l_t = \alpha \frac{y_t}{s_t} + (1 - \alpha)l_{t-1}$

Seasonal: $s_{t+m} = \gamma \frac{y_t}{l_t} + (1 - \gamma)s_t$

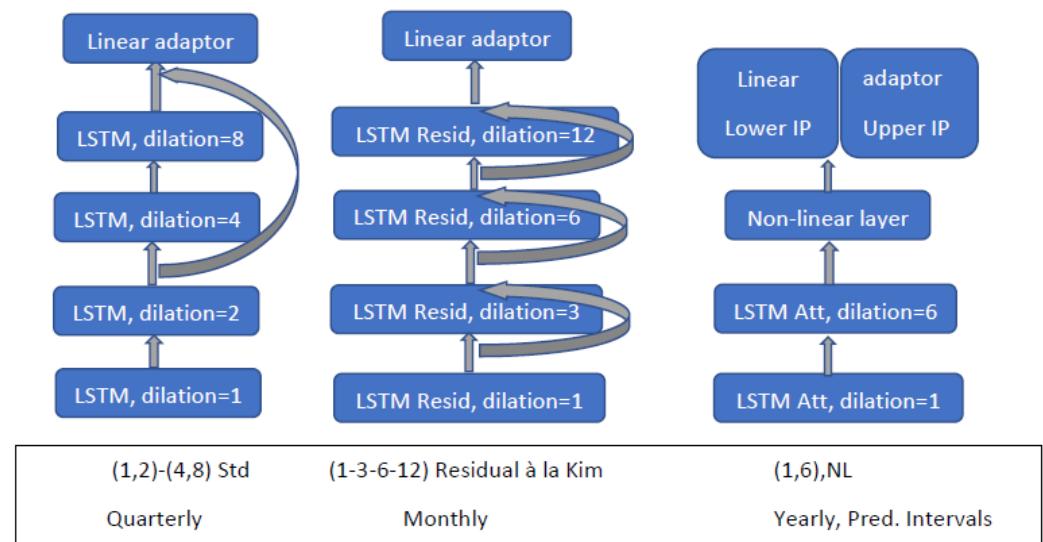
De-seasonalized
and normalized
time-series:

$$x_t = \frac{y_t}{l_t s_t}$$

ES-RNN Neural Architectures

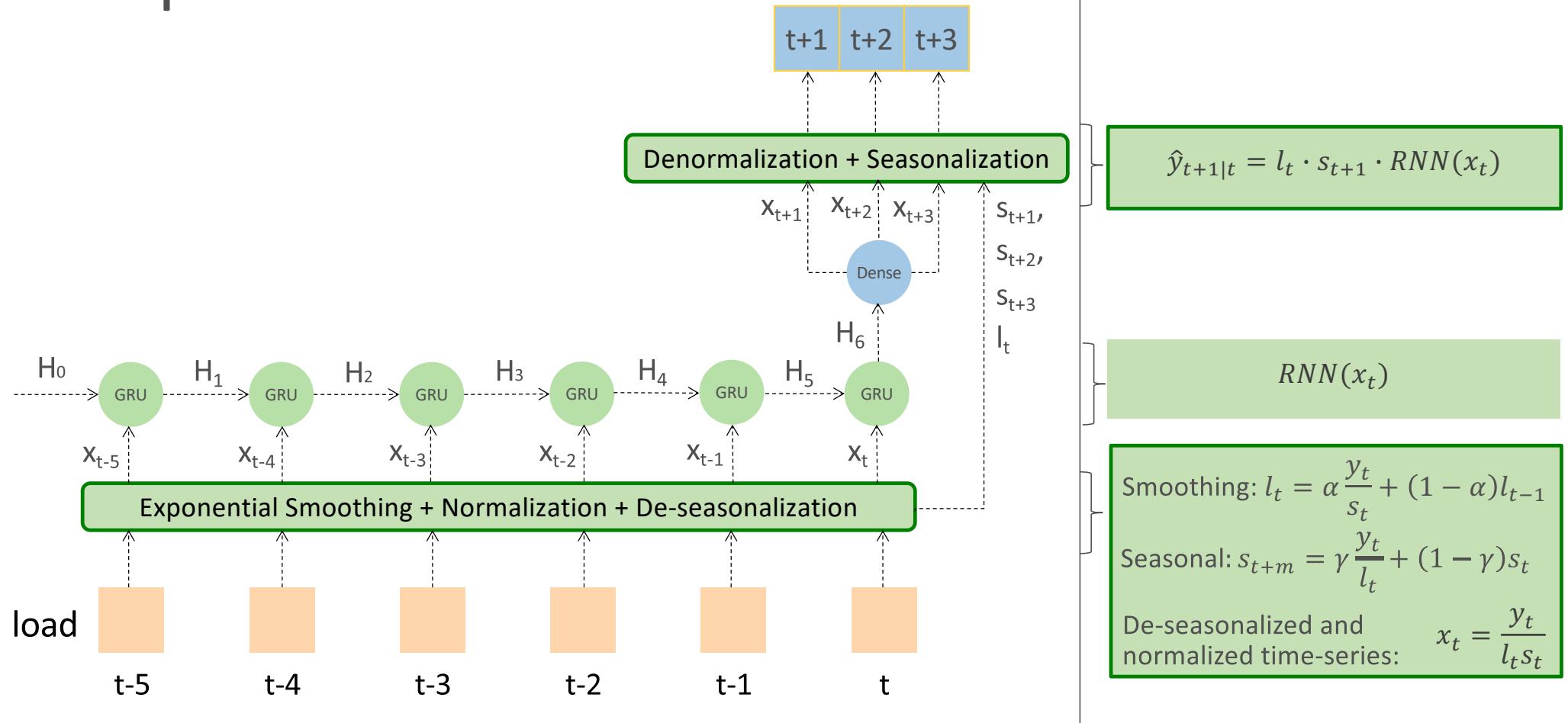
$$\hat{y}_{t+1|t} = l_t \cdot s_{t+1} \cdot RNN(x_t)$$

- Stack of dilated LSTM-based blocks connected with shortcuts.
- ES-RNN uses 3 types of dilated LSTMs:
 - [Dilated Recurrent Neural Networks](#)
 - [Dual-Stage Attention-Based Recurrent Neural Network](#)
 - [Residual LSTM](#)

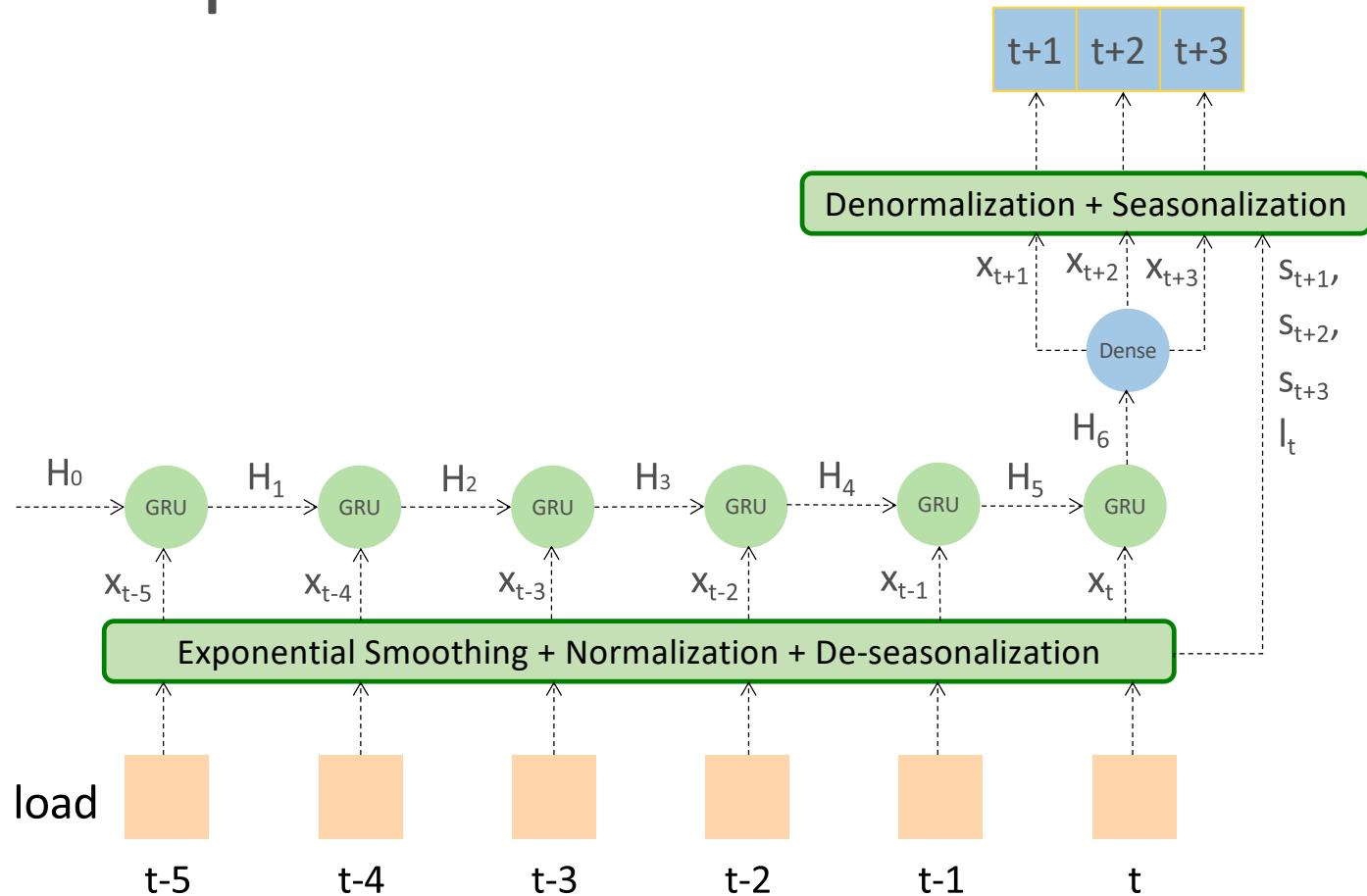


In the example notebook of this tutorial, we simplify it to standard GRU.

Simplified ES-RNN



Simplified 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 which are learned by minimizing cost function (using BPTT)

Hands-on Experiment

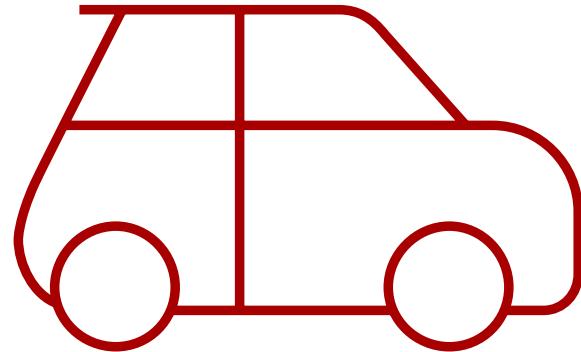


Open: 4_ES_RNN.ipynb notebook



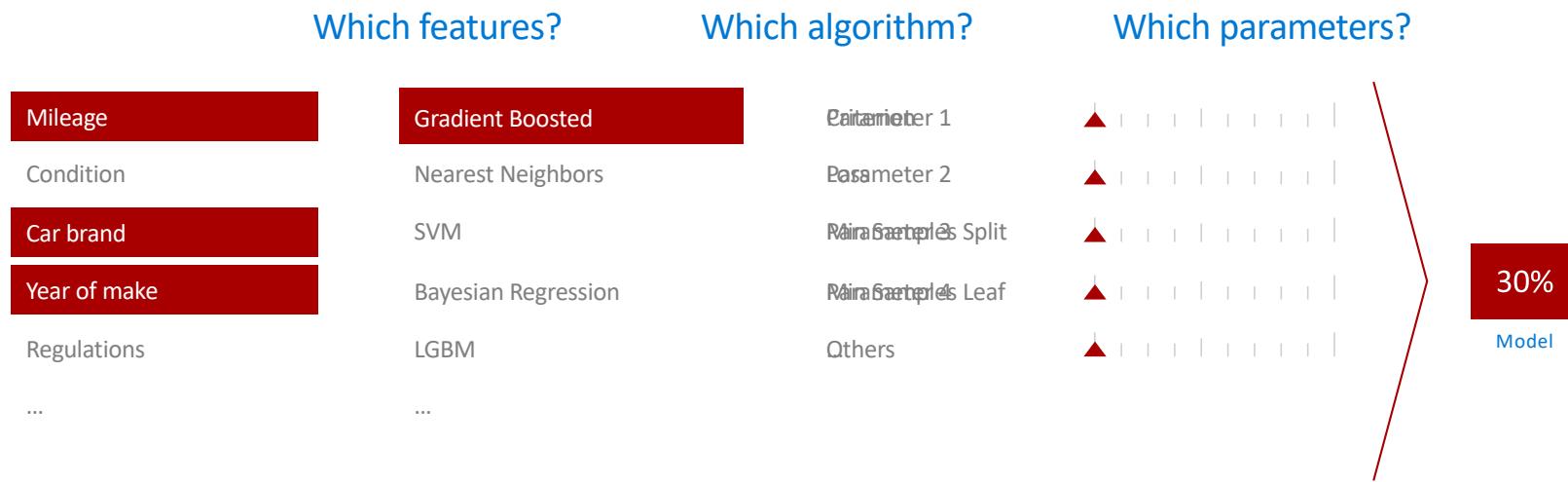
AutoML

Machine Learning Problem Example



How much is this car worth?

Model Creation Is Typically Time-Consuming



Model Creation Is Typically Time-Consuming

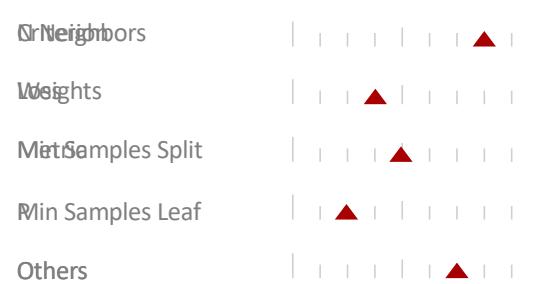
Which features?

Mileage
Condition
Car brand
Year of make
Regulations
...

Which algorithm?

Gradient Boosted
Nearest Neighbors
SVM
Bayesian Regression
LGBM
...

Which parameters?

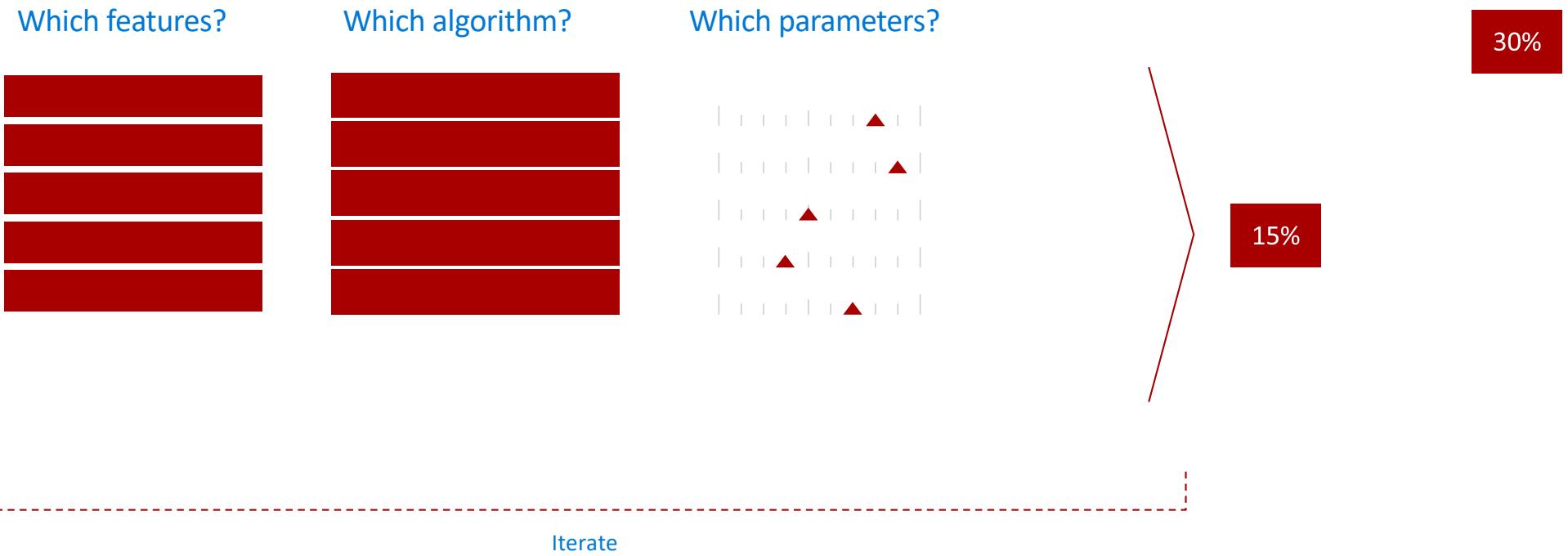


30%

Model

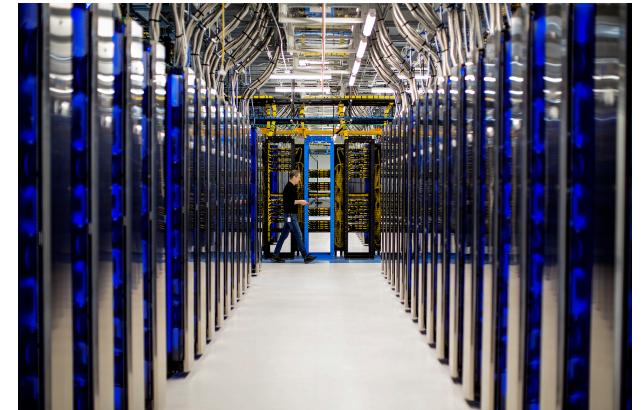
Iterate

Model Creation Is Typically Time-Consuming



Automated Machine Learning Mission

Enable automated building of machine learning with the goal of accelerating, democratizing and scaling AI



Democratize AI

Enable Domain Experts & Developers to get rapidly build AI solutions

Accelerate AI

Improve Productivity for Data Scientists, Citizen Data Scientists, App Developers & Analysts

Scale AI

Build AI solutions at scale in an automated fashion

What is automated machine learning?

Automated machine learning (automated ML) automates the machine learning workflow to find the best model for your data:

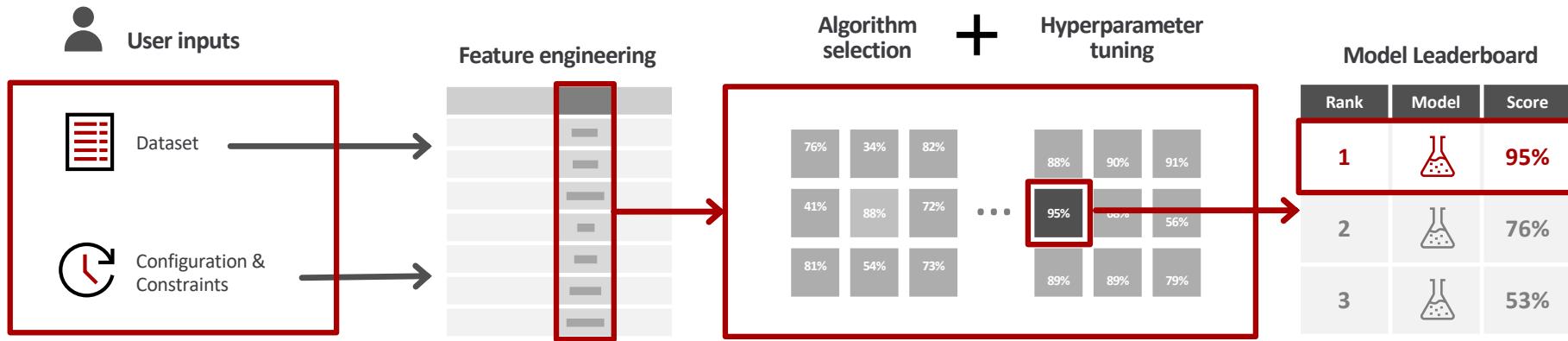
- Data cleansing

- Feature engineering

- Algorithm selection

- Hyperparameter tuning

Automated Machine Learning Under the Hood



What you get with AutoML:

- Improved transparency, control & trust
- Large datasets
- State of the art NLP
- Industry leading time series forecasting
- Ease of use

Guardrails



Class imbalance



Train-Test split, CV, rolling CV



Missing value imputation



Detect high cardinality features



Detect leaky features



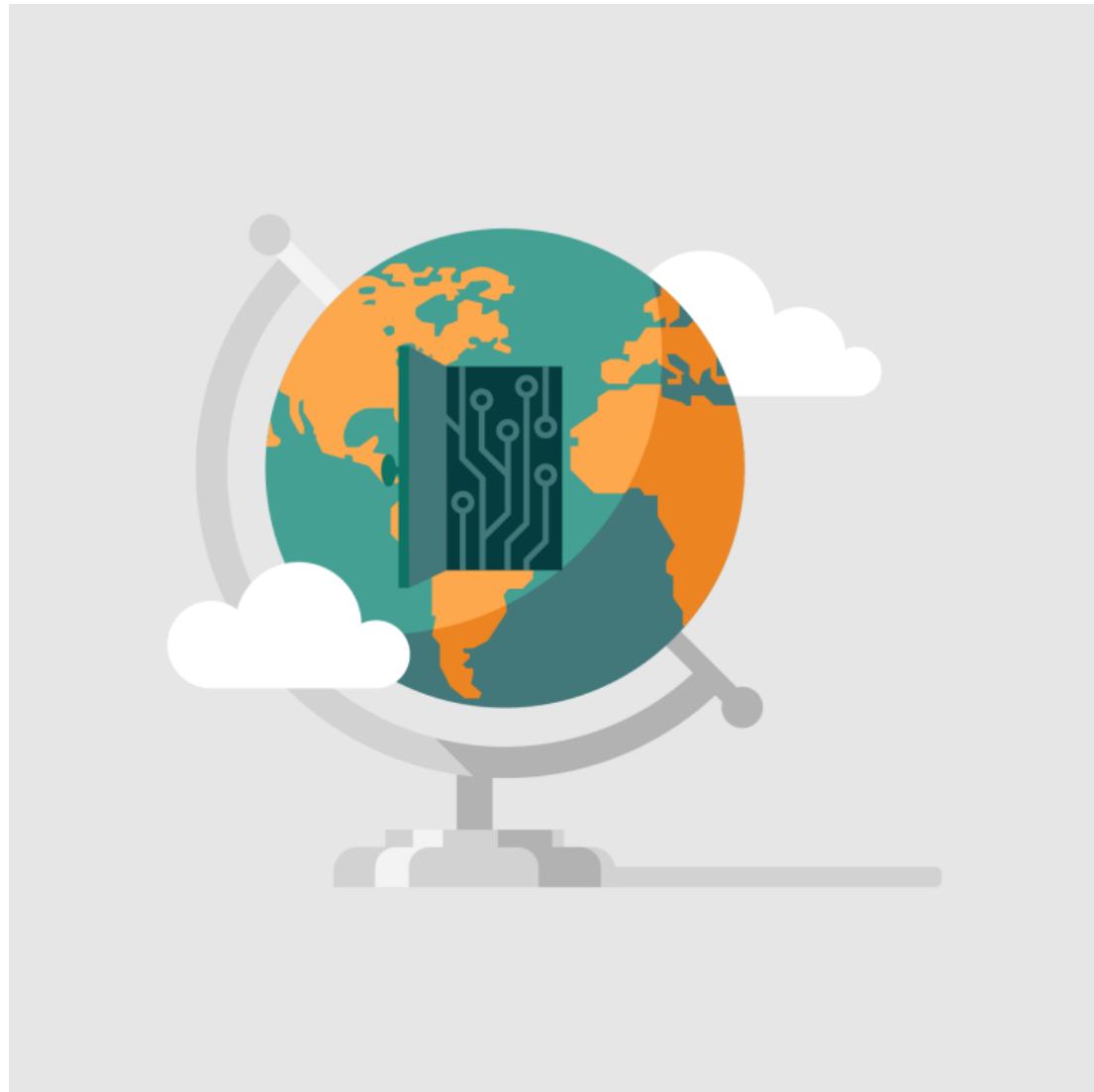
Detect overfitting



Model Interpretability / Feature Importance

Large Datasets

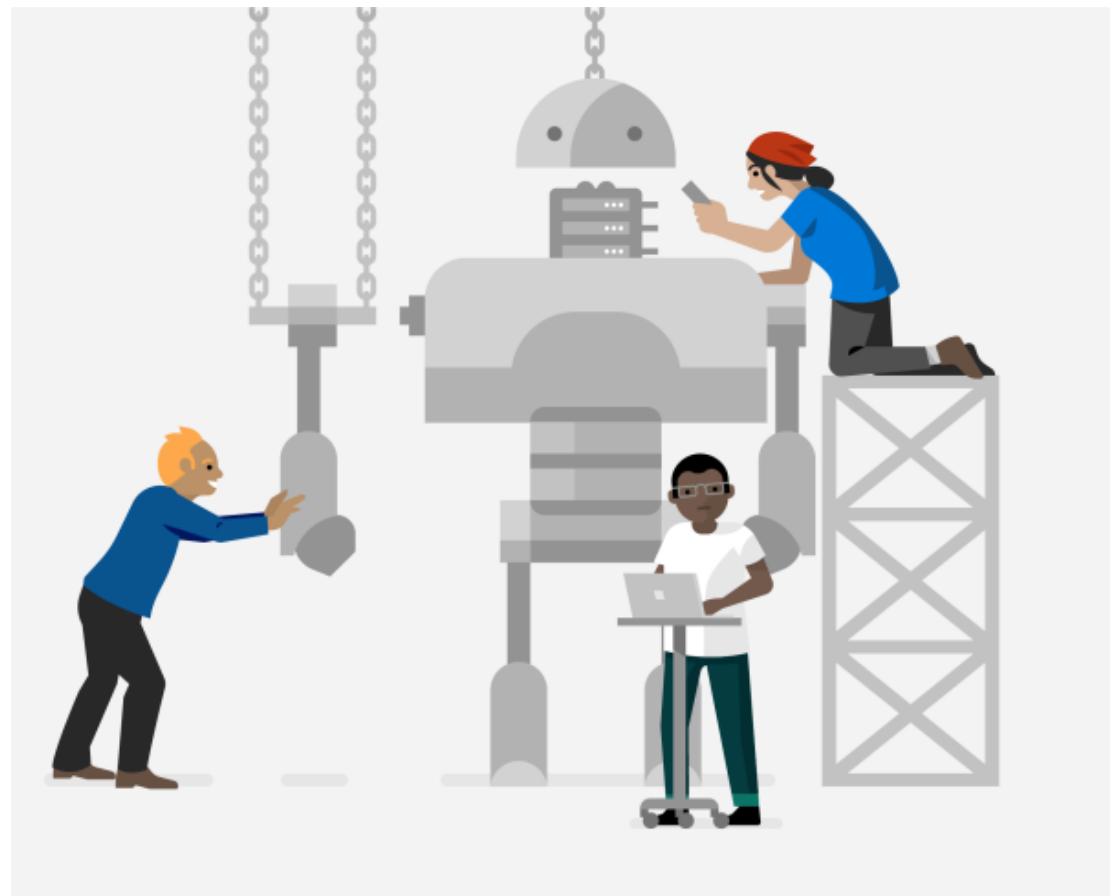
- Support 100GB+ datasets
- Early results in for 1TB
- Classification & Regression



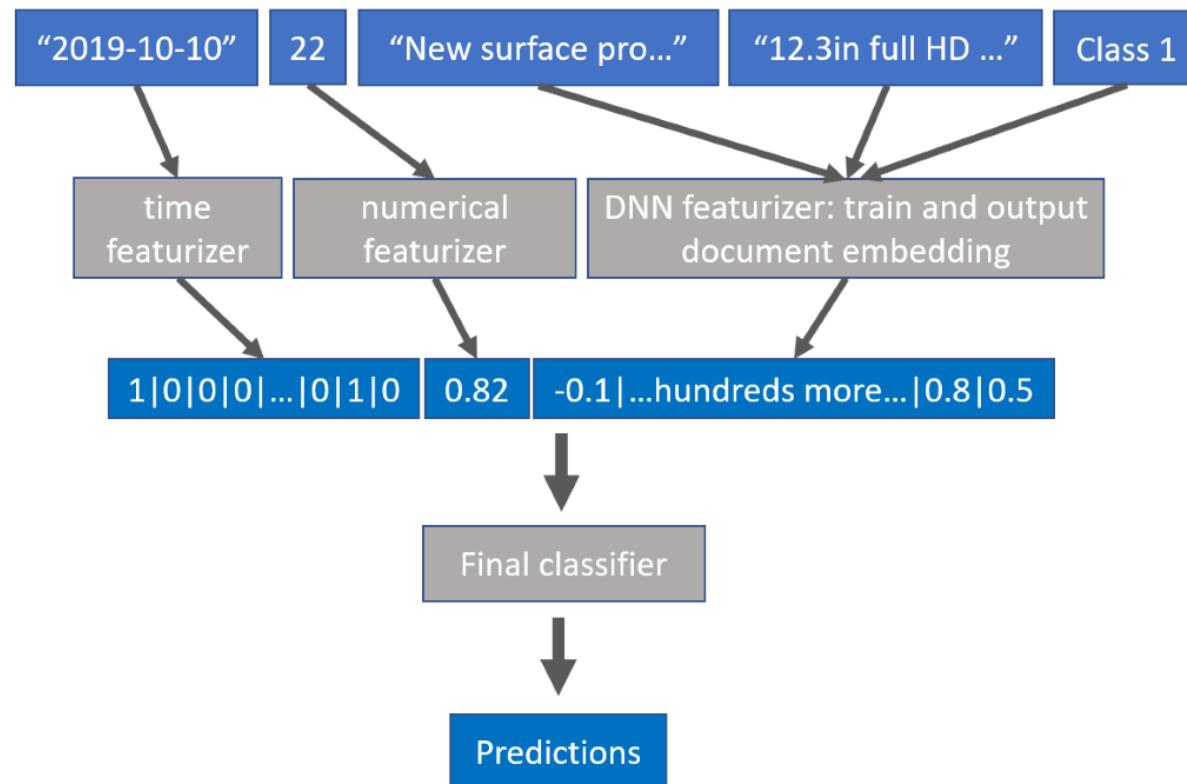
Feature Engineering

Transforming **raw data** into **features** to better represent the problem at hand

- Customizable
- Transparent
- Text, Images, Time, Categoricals



Featurization



Time series forecasting

Overview

- Forecast function
- Rolling-origin cross validation
- Configurable lags
- Rolling window aggregate features
- Holiday detection and featurization

Holiday Detection

Training data

Date	Sales
1/1/2011	50
1/2/2011	1000
1/3/2011	3000

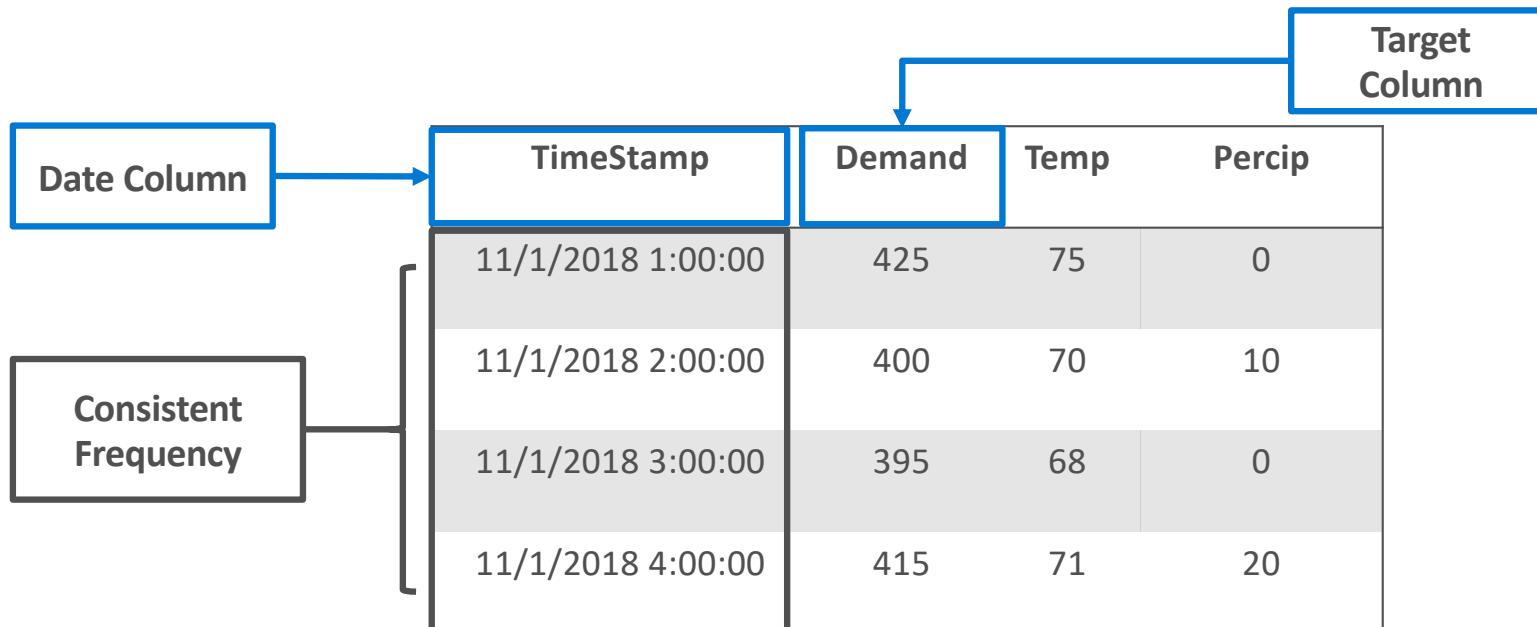
Training data with holiday features

Sales	year	iso	day	wday	holiday	ispaidtimeoff
50	2010	1	5		New year's day	1
1000	2010	2	6	1 day after New year's day		0
3000	2011	3	0		""	0

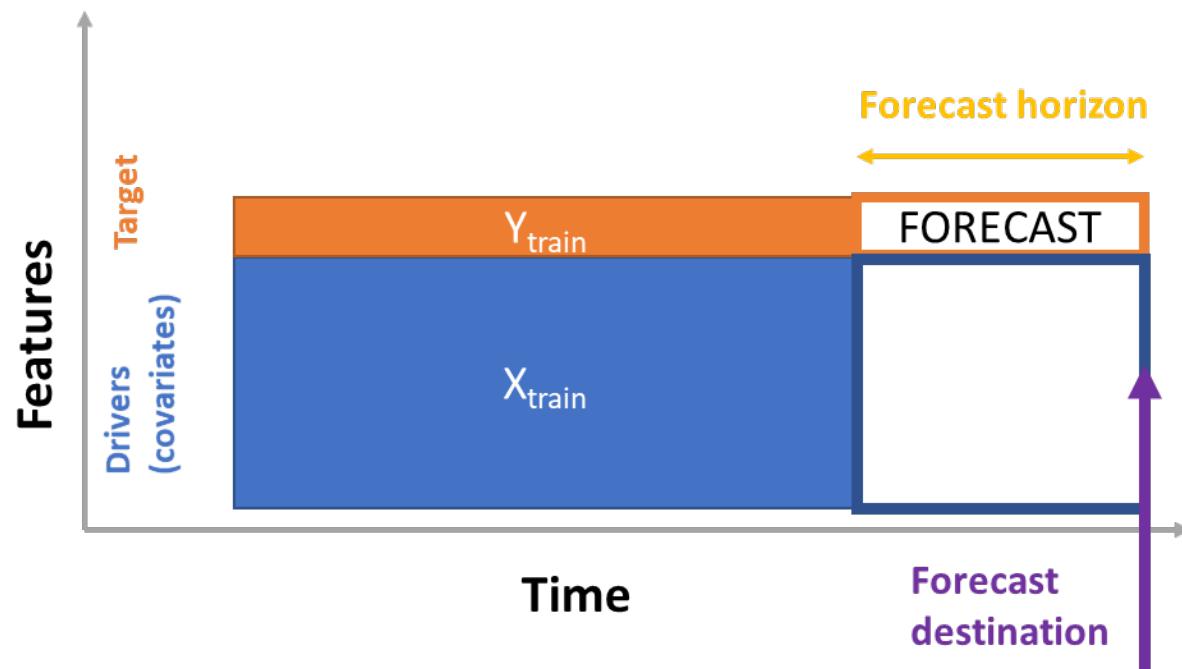


Time Series Forecasting Requirements

- A **target column** for running predictions
- A feature in your data representing a **valid time series**



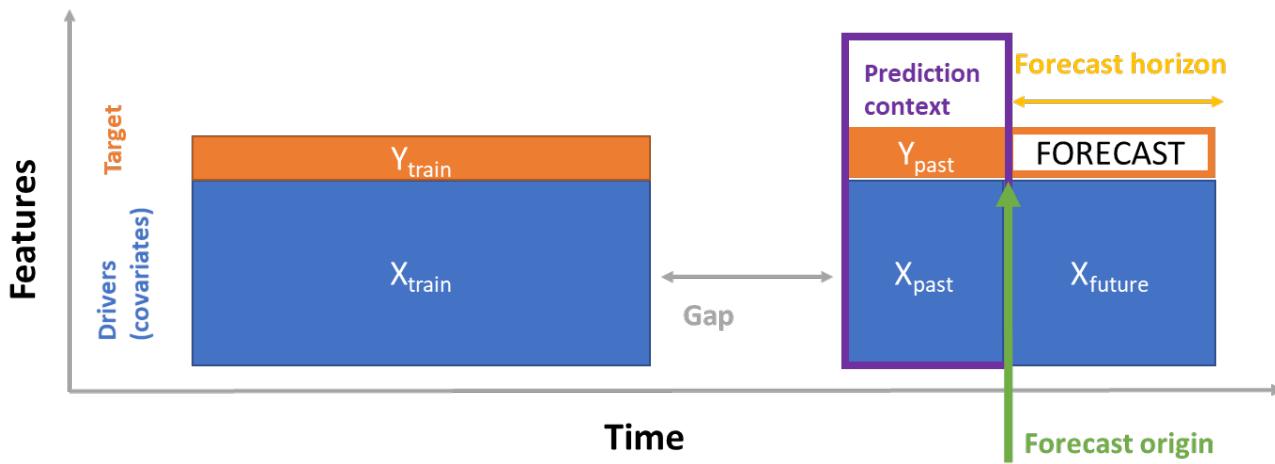
Forecast Horizon



- Prediction
 - How far out do you want to forecast?

Forecast Function

- A gap in your time series...
 - We handle that!
 - Provide prediction context with recent values of the target and drivers (i.e. weather)



Target Rolling Window Aggregation

Date	Device ID	Precip	Precip_min_window 3D	Precip_max_window 3D	Precip_sum_window3 D
1/1/2017	1	25	nan	nan	nan
1/2/2017	1	15	nan	nan	nan
1/3/2017	1	30	min	nan	sum
1/4/2017	1	20	15	30	70
1/5/2017	1	40	15	30	65
1/6/2017	1	30	20	40	90
1/7/2017	1	15	20	40	90
1/8/2017	1	20	15	40	85

Latest features



Deep learning



Auto-settings



Holiday detection



Include weather and other open datasets



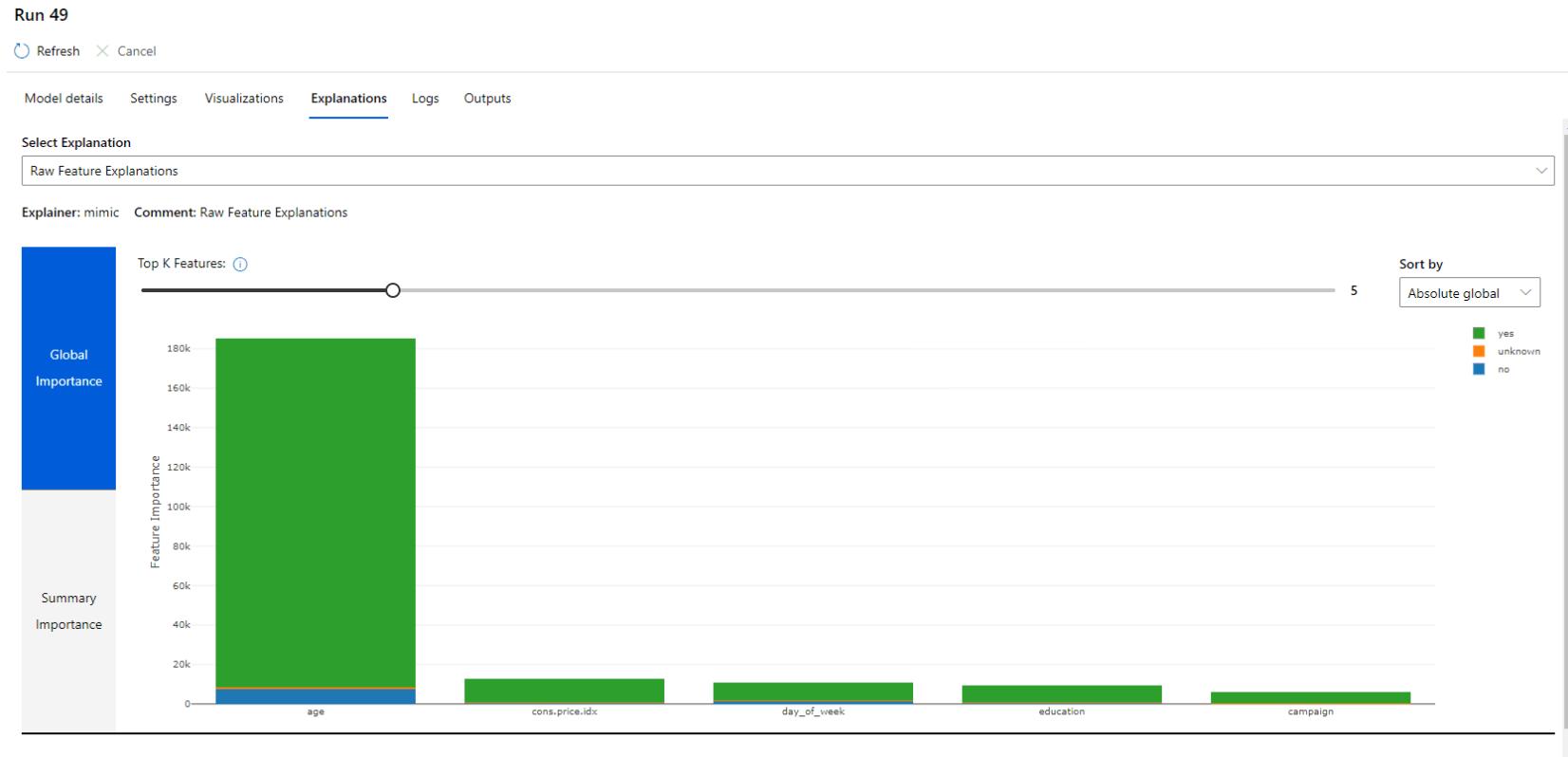
Many models



New learners

Model Interpretability

- Understanding **why** the model made a particular prediction



Hands-on Experiment



Open: automl/1_AzureAutoML_local.ipynb

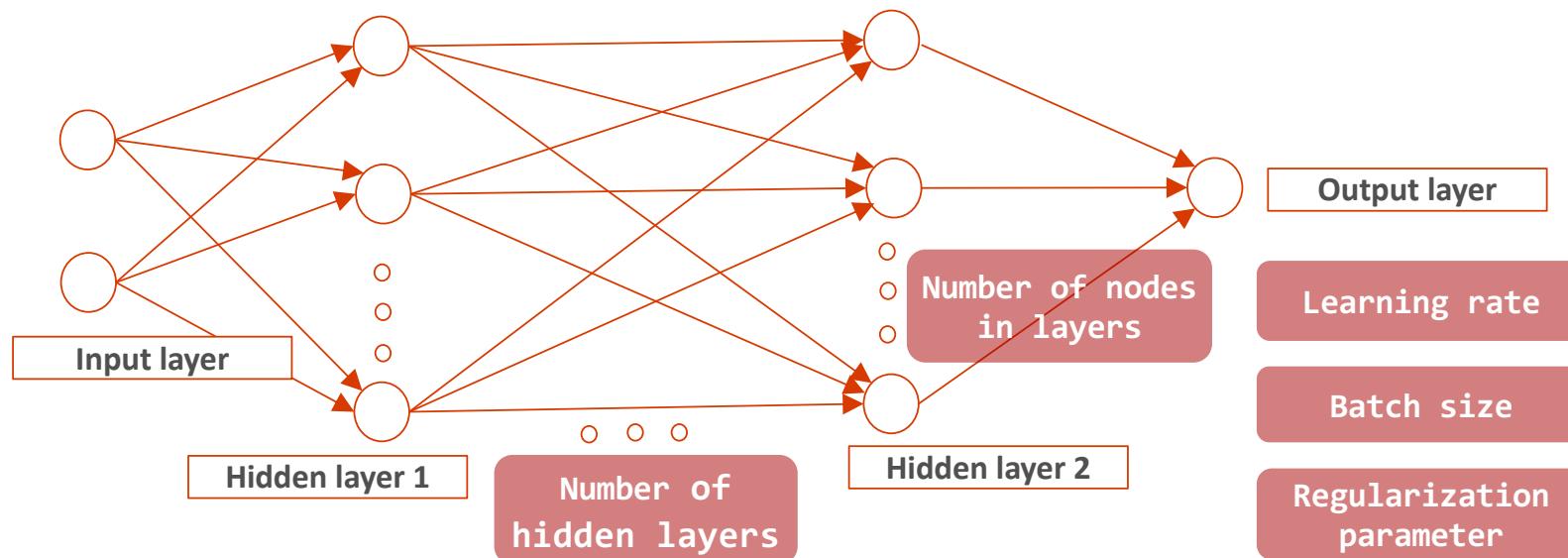
Appendix: Other Hybrid Models



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

Automated Hyperparameter Tuning in Azure ML

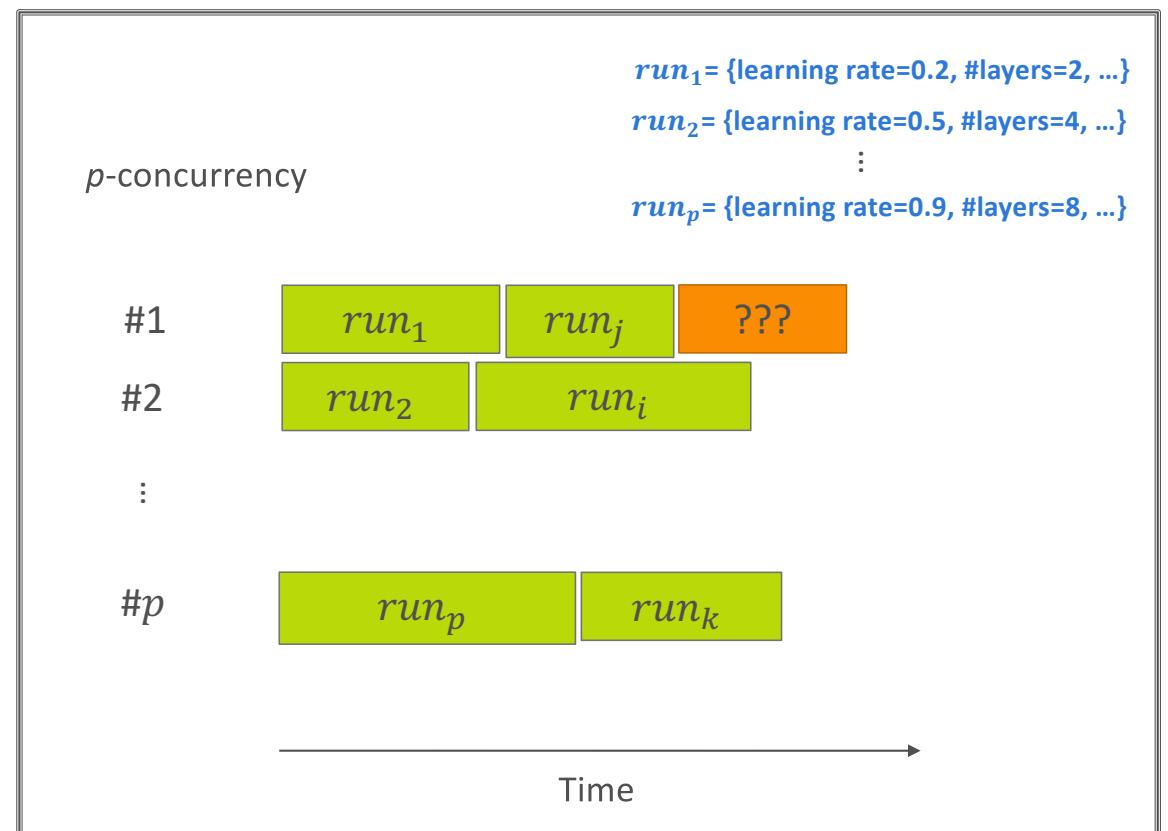
- Automate the exploration process
- Efficient use of resources
- Optimize model for a specified metric
- Offers control over resource budget
- Training framework agnostic
- Visualize all configurations in one place

Azure Machine Learning Service (Hyperdrive)

Launch multiple parallel training runs

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

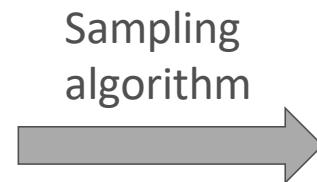
- (B) Manage resource usage of active runs
- How long to execute a run?



Generating New Runs – Sampling algorithms

```
{  
    "learning_rate": uniform(0, 1),  
    "num_layers": choice(2, 4, 8)  
    ...  
}
```

Define hyperparameter search space



```
Config1= {"learning_rate": 0.2,  
          "num_layers": 2, ...}
```

```
Config2= {"learning_rate": 0.5,  
          "num_layers": 4, ...}
```

```
Config3= {"learning_rate": 0.9,  
          "num_layers": 8, ...}
```

...

Supported sampling algorithms –

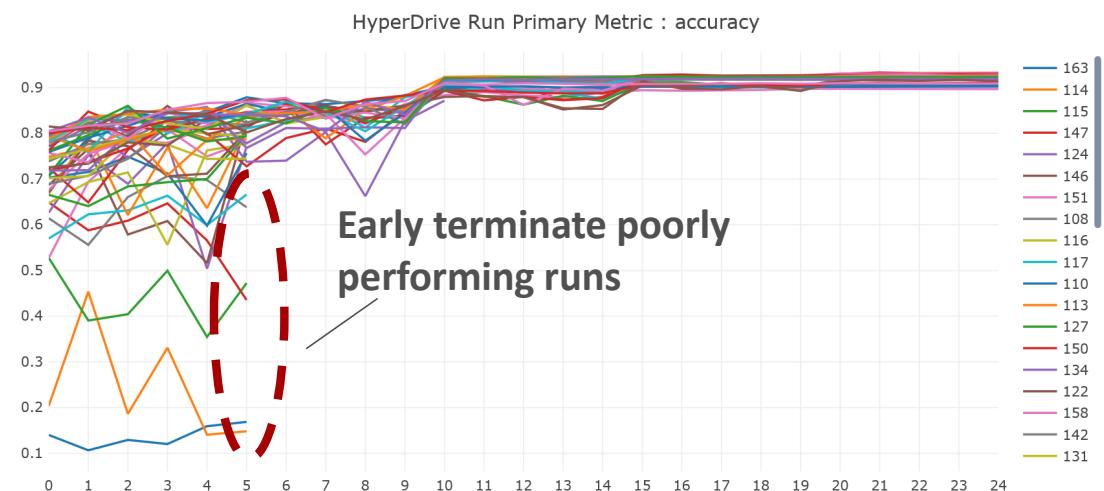
- Grid Sampling
- Random Sampling
- Bayesian Optimization

Managing Active Jobs

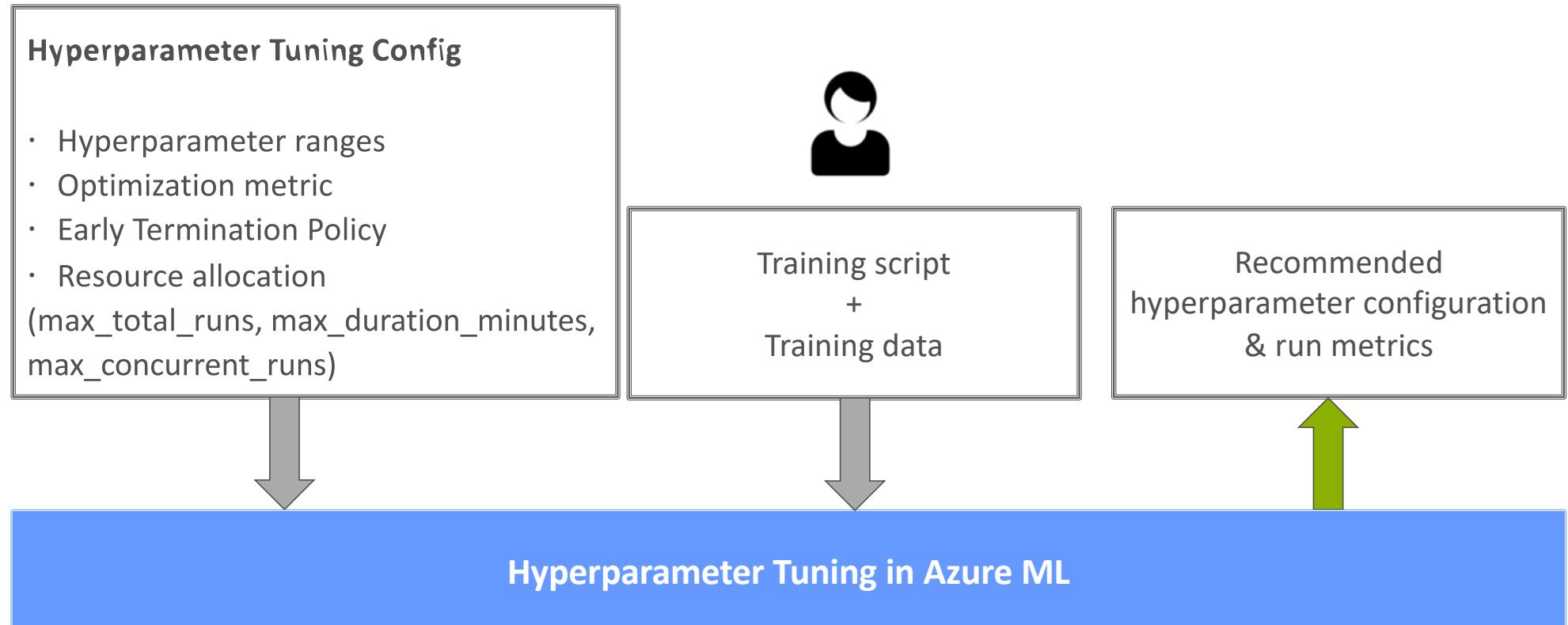
- Evaluate training runs for specified primary metric
- Early terminate poorly performing runs
- Use resources to explore new configurations

Supported early termination policies

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



User inputs & system outputs



Demo

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](#)

References

- [19] S. Smyl, 2018, M4 Forecasting Competition: Introducing a New Hybrid ES-RNN Model
- [20] Hyndman, R.J., & Athanasopoulos, G. (2018) *Forecasting: principles and practice*
- [21] Dilated Recurrent Neural Networks
- [22] Dual-Stage Attention-Based Recurrent Neural Network
- [23] Residual LSTM
- [24] Fast ES-RNN: A GPU Implementation of the ES-RNN Algorithm

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