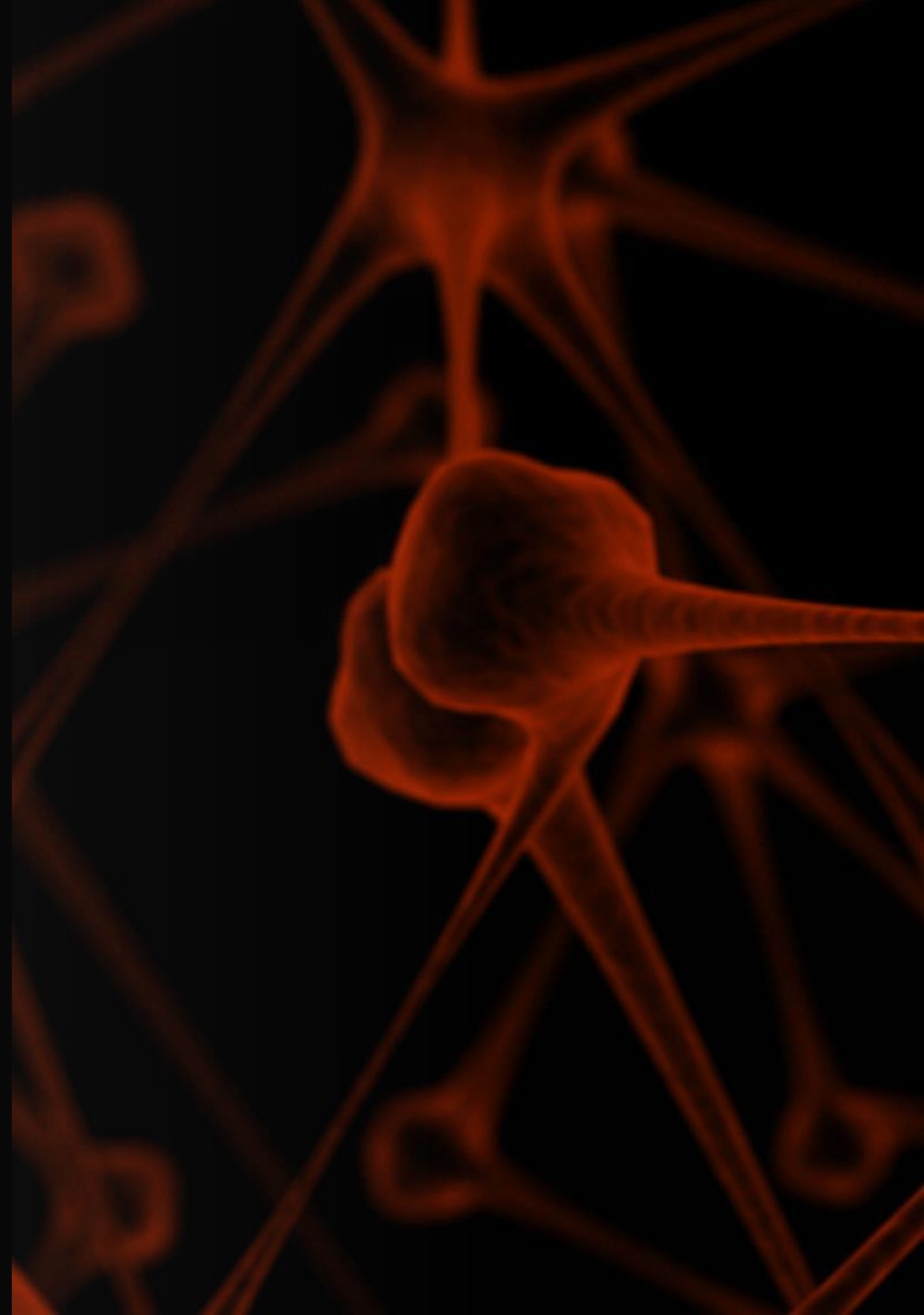
Microsoft

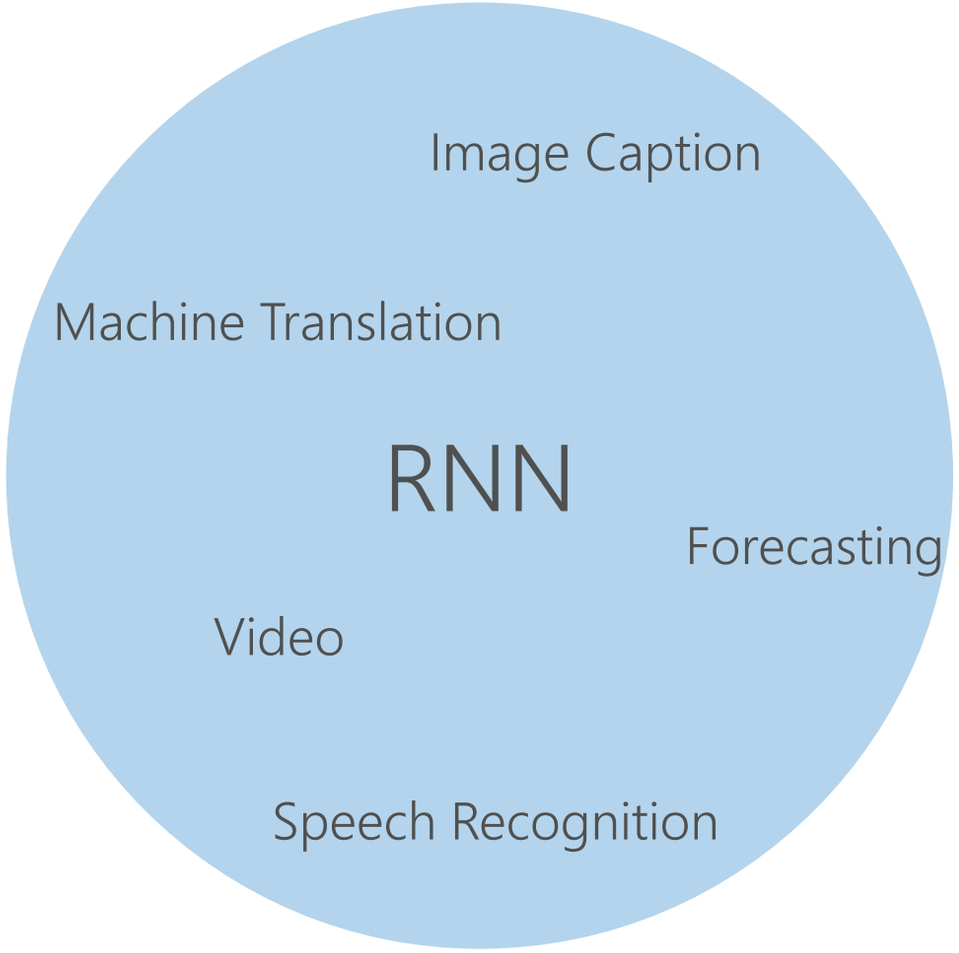
Recurrent Neural Networks for Time Series Forecasting

Yijing Chen, Dmitry Pechyoni, Angus Taylor, Vanja Paunic

Tutorial Introduction



Tutorial Introduction

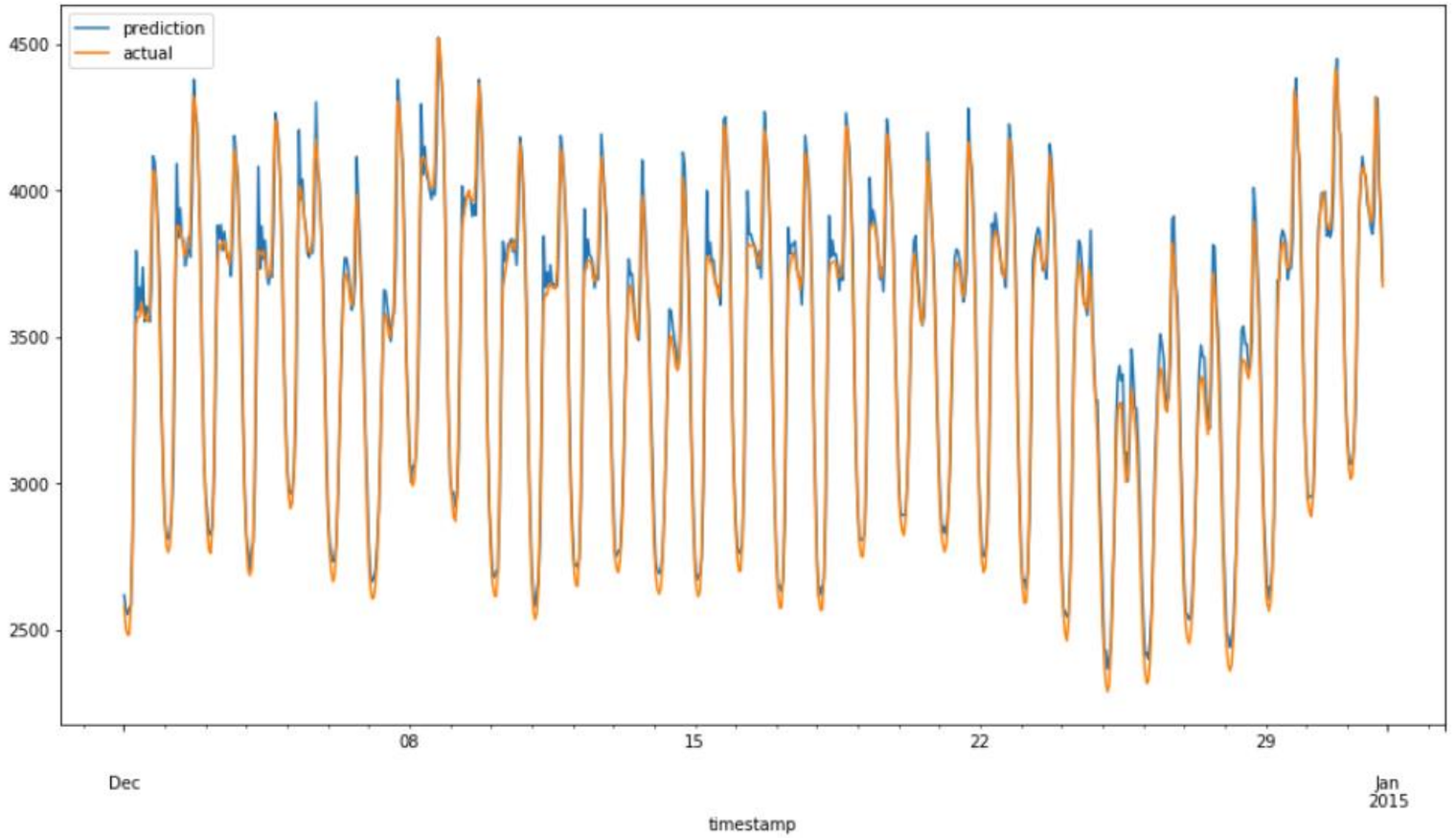


Tutorial Goals

- Understand the basics of recurrent neural networks (RNN) and advanced RNN architectures including LSTM and GRU
- Know how to use Keras to build RNN model for time series forecasting
- Know a number of techniques and tricks that are important for building successful RNN-based time series forecasting models

Tutorial Goals

By the end of this tutorial, you will have the knowledge to build RNN model to forecast New England energy consumption.



Target Audience

- 200 level
- Know the basics of Python
- You need to understand the basic concepts of machine learning and have some experience of building machine learning models
- No prior experience with time series or neural networks is required

Please provide feedback to help us improve!
aka.ms/rnnfeedback

Learning Path

Agenda	Time (mins)
Tutorial Introduction + Pre-requisite Setup	20
Introduction to time series forecasting and time series model	25
Introduction to feedforward neural network(NN) model	25
How to apply feedforward NN model to time series forecasting	25
Break	10
Introduction to recurrent neural network(RNN) model	25
How to apply RNN to one step time series forecasting	25
How to apply RNN to multi-step time series forecasting	30
Other successful RNN model + Conclusion	10
Q&A	10

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

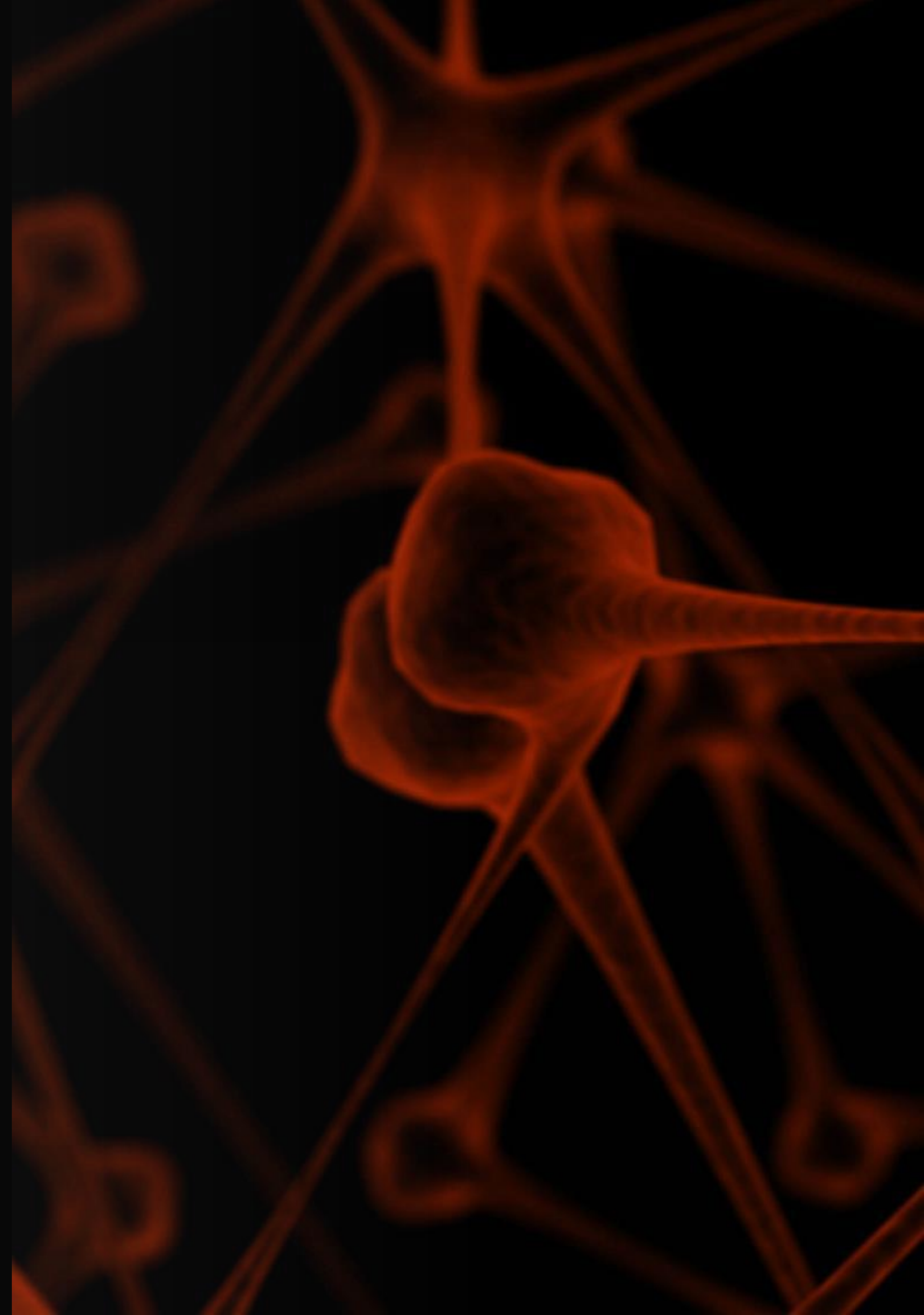
Hands-on Lab

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

Pre-requisite Setup

aka.ms/rnntutorial

Introduction to 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
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	?

Why is Time Series Forecasting important?

- Every vertical has time series data (retail, energy, etc.)
- Forecasts are needed in every industry to guide future decisions
- Forecasts are usually the first step of other processes (e.g. supply chain optimization, budget preparation, etc.)

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?

……

……

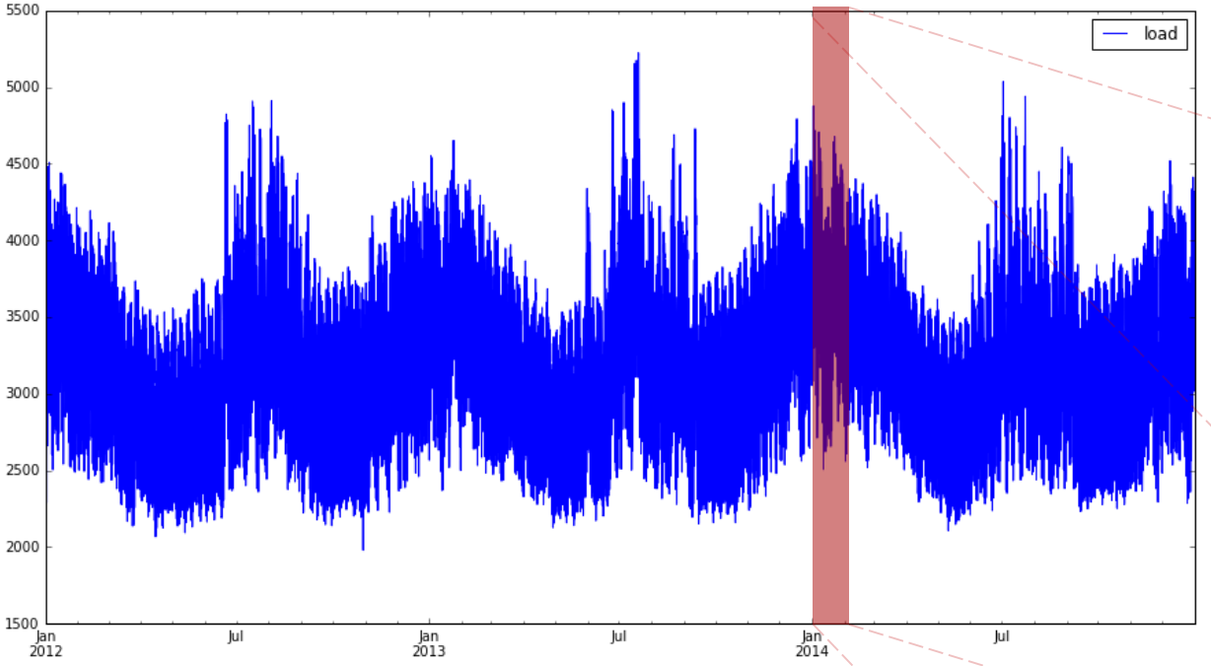
- What technique/model should I use?

Scenario:
Energy load forecasting

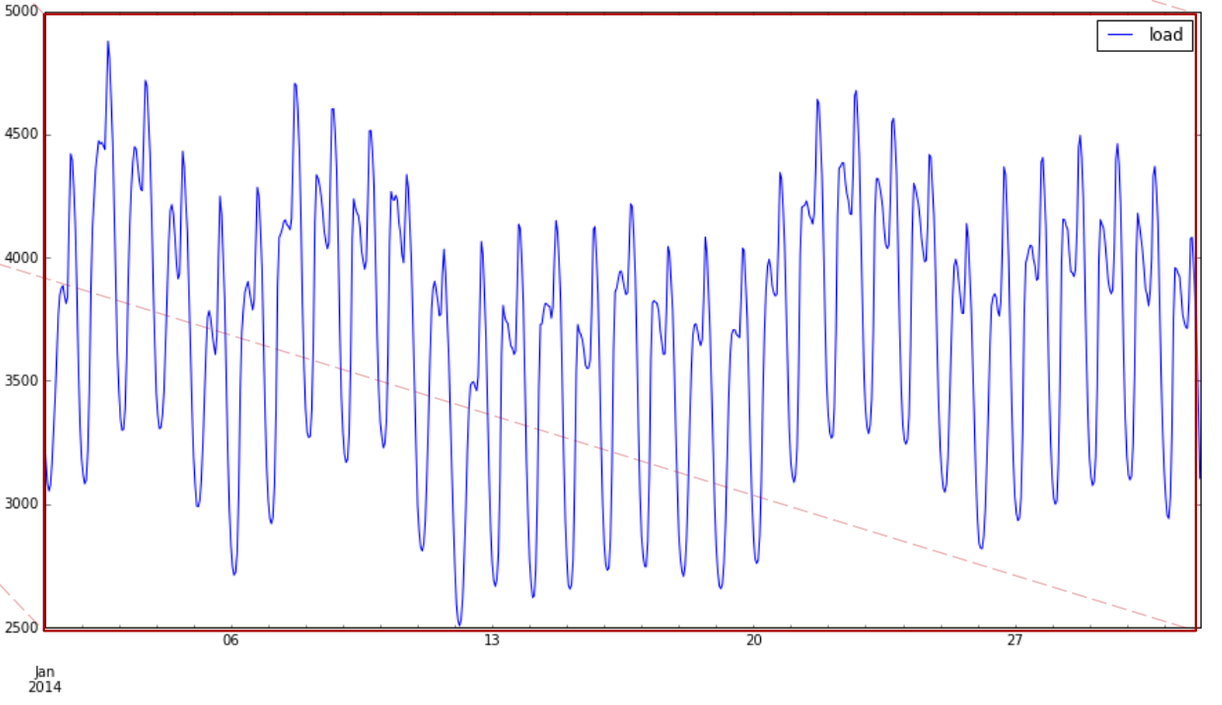
Scenario: energy load forecasting

- 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

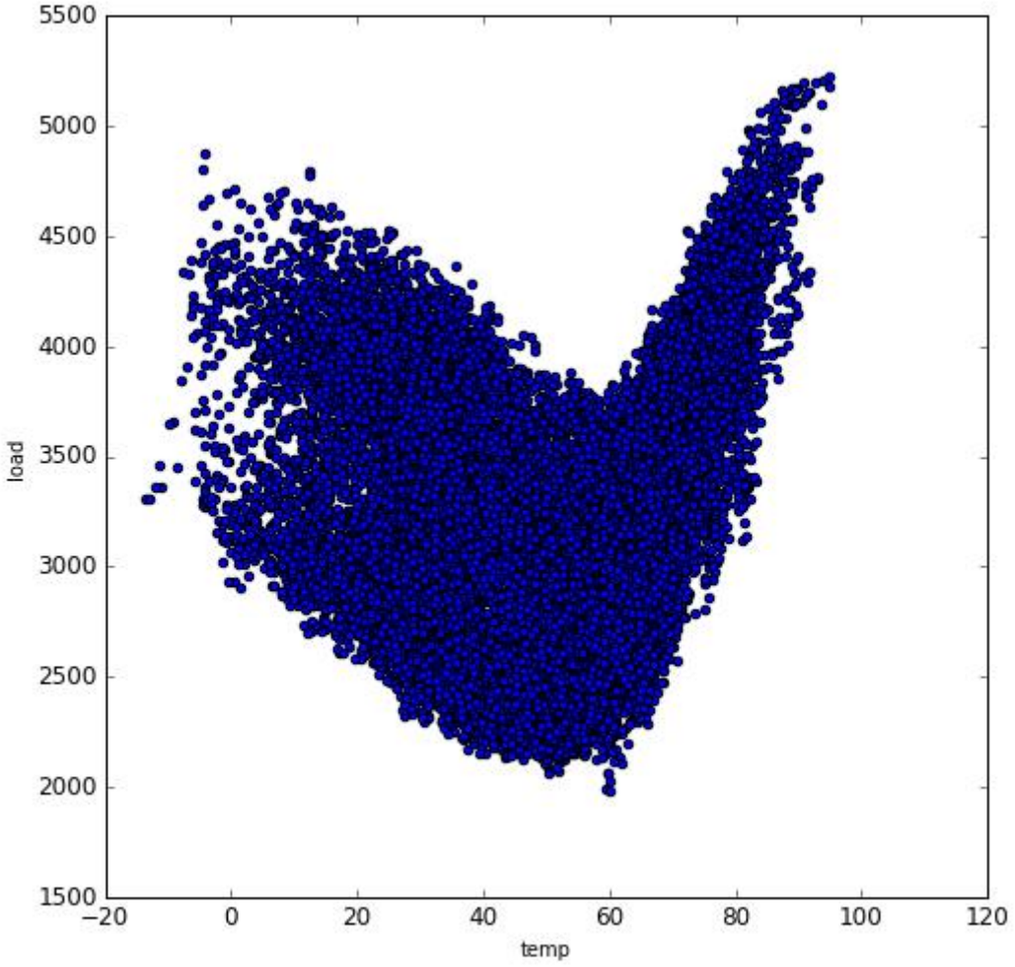


- 26,000 hourly load values
- Annual, weekly and daily seasonality



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.

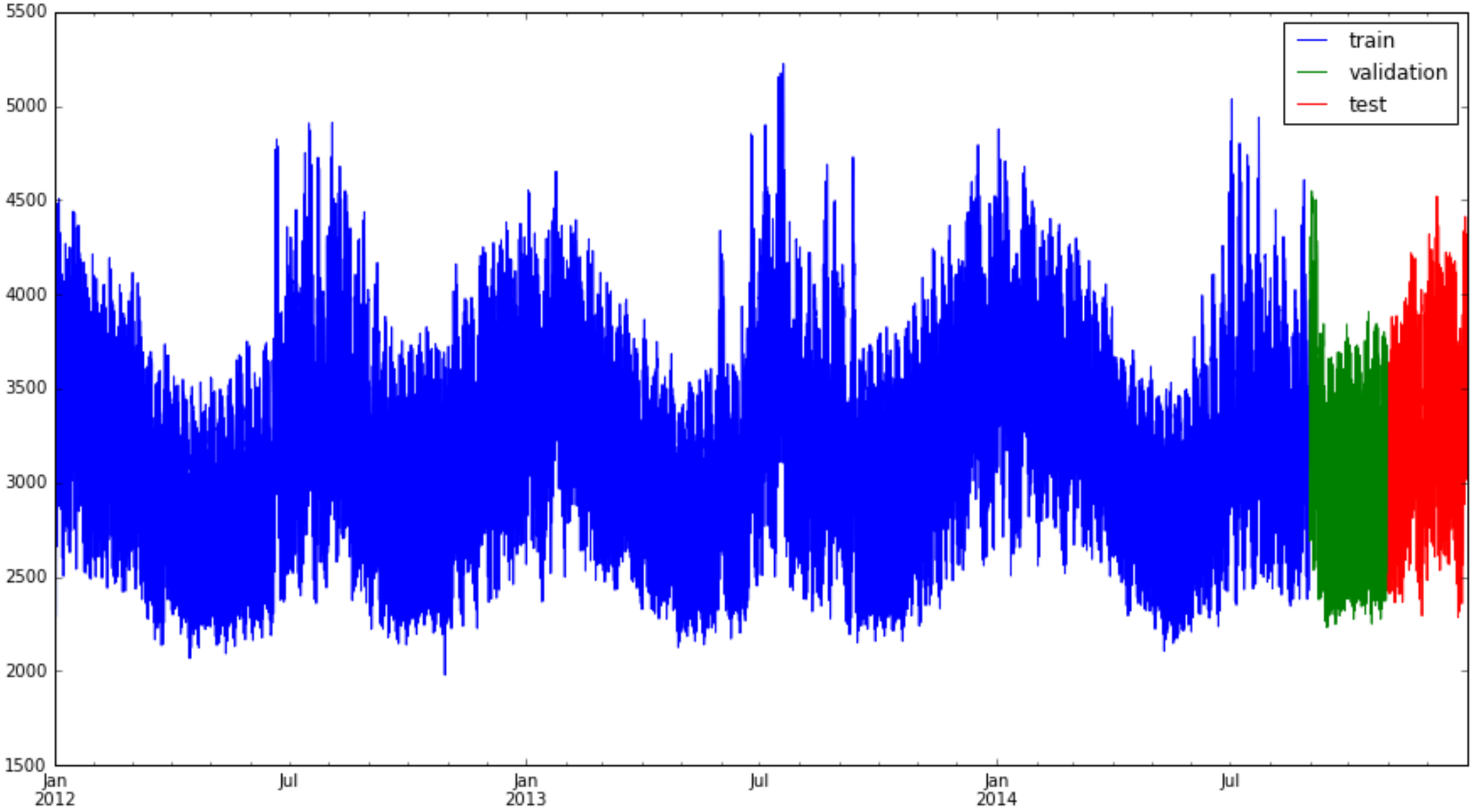
New England ISO data



- Load is also highly dependent on temperature

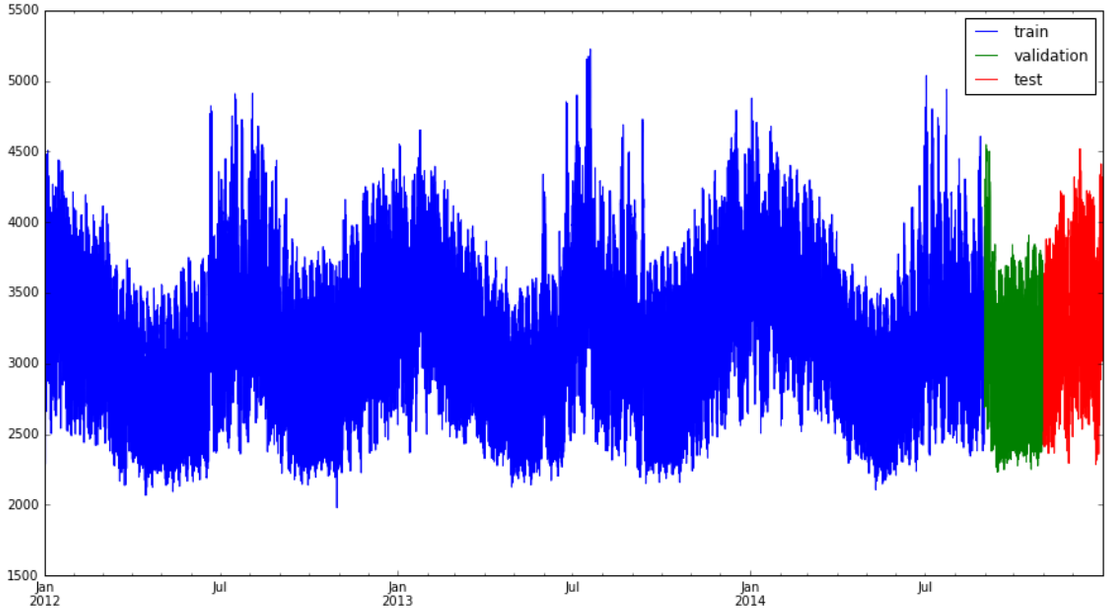
Experimental setup

Training, validation and test sets



Training, validation and test sets

- **Train** – used for model training. Contains no information from the other datasets
- **Validation** – used for testing different model hyperparameters
- **Test** – used to evaluate the model and compare to others



Time series forecasting with ARIMA

- Traditionally time series forecasting is done using a curve fitting methods that aim to capture seasonality, trend and autocorrelation in the data (or some combination of these).

- These methods require a deeper analysis and understanding of the time series to better select the most fitting method.

- Please open 1_time_series_arima.ipynb notebook

Why RNN?

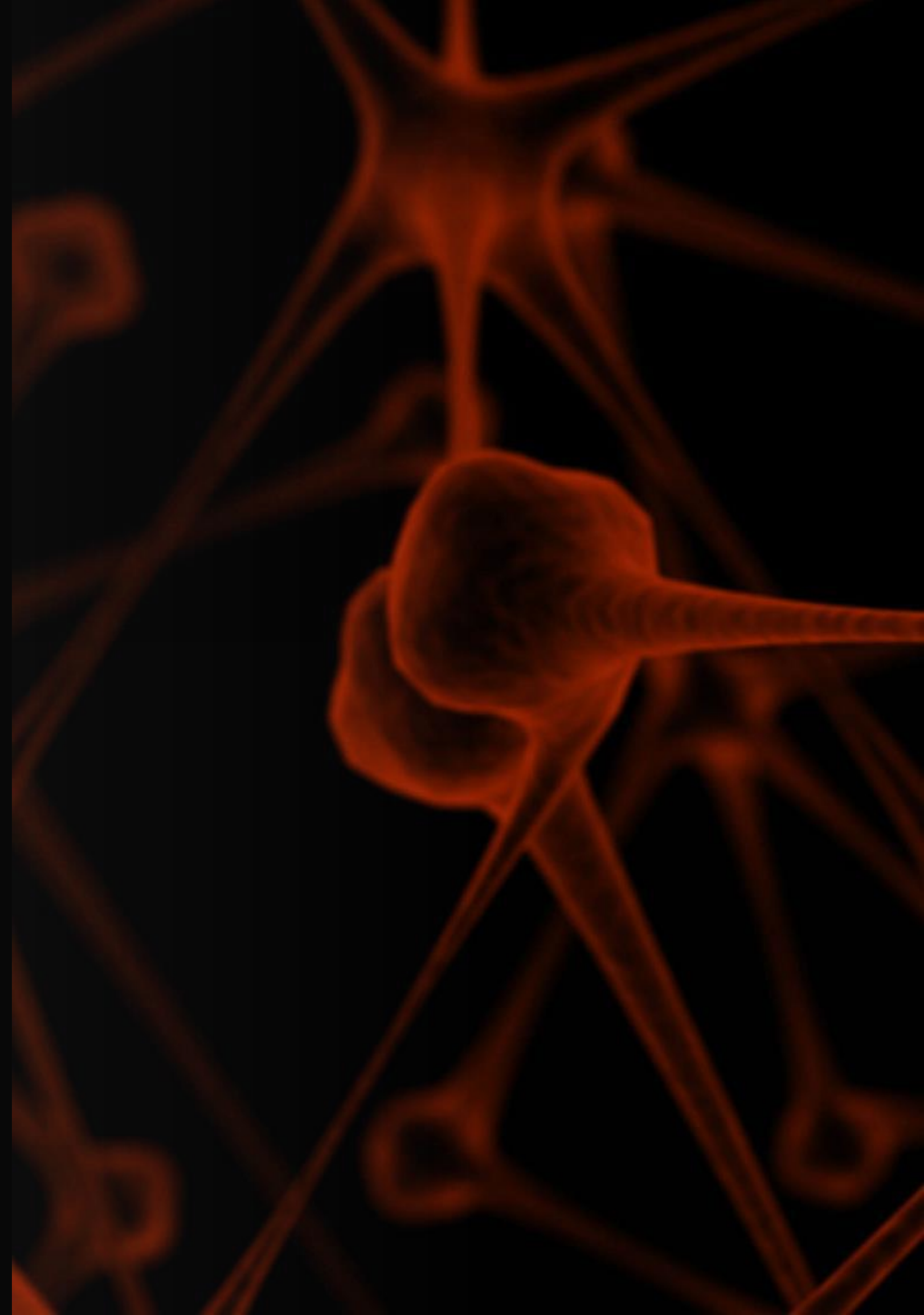
- RNN has been shown perform well in many scenarios
  - 2014 Global Energy Forecasting Competition (link)
  - 2016 CIF International Time Series Competition (link)
  - 2017 Web Traffic Time Series Forecasting (link)
  - 2018 Corporación Favorita Grocery Sales Forecasting (link)

- RNN is non-parametric, that 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

Performance comparison

Algorithm	MAPE (Mean Absolute Percentage Error)
ARIMA	4.9

Introduction to Feedforward Neural Networks

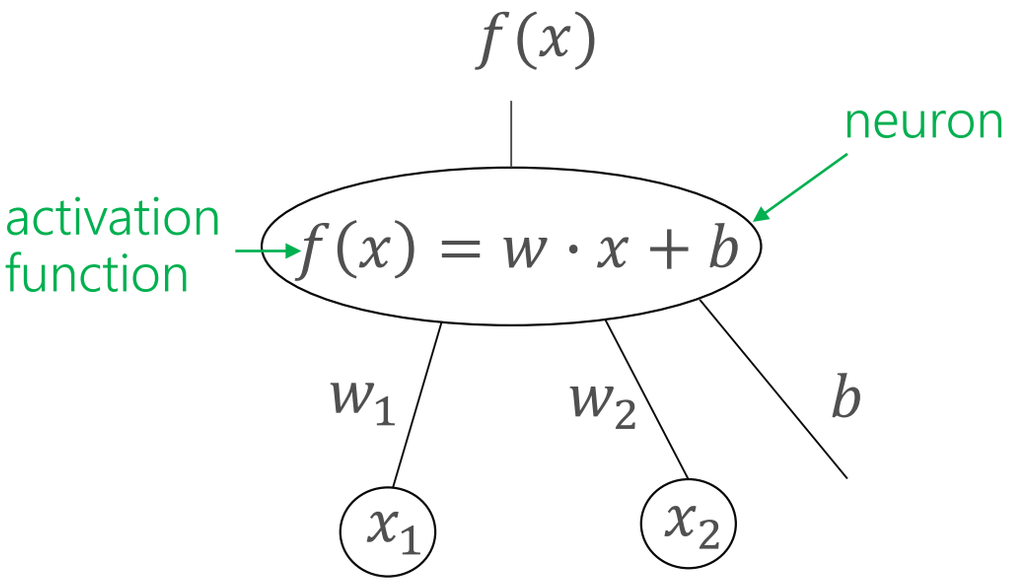


Agenda

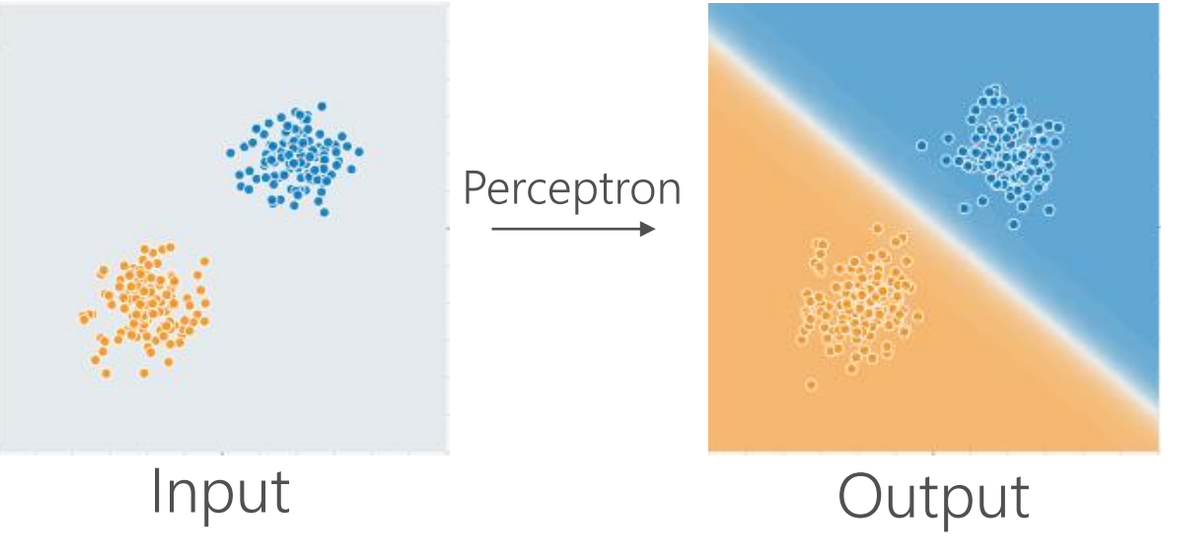
- Perceptron & Multilayer Perceptron
- Activation Functions
- Feed-forward Neural Network Training
  - Loss function
  - Gradient descent (stochastic, batch, mini-batch gradient descent)
  - Backpropagation
- Hands-on: Use Feed-forward Neural Network for time-series forecasting

Perceptron (Rosenblatt, 1957)

training example: $x = x_1 x_2$, y



$$\hat{y} = \begin{cases} 1 & \text{if } f(x) > \text{threshold (e.g. 0.5)} \\ -1 & \text{otherwise} \end{cases}$$



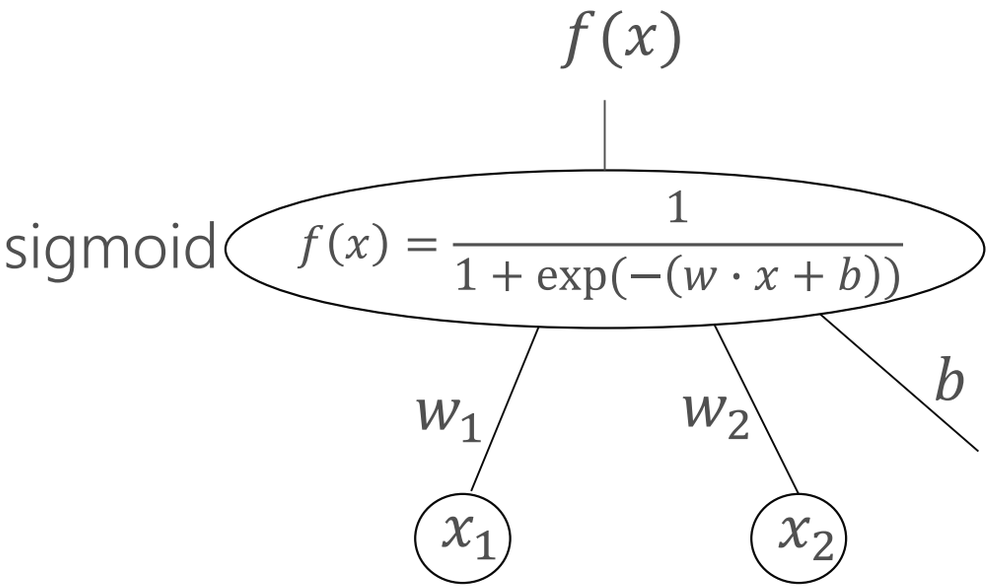
Input

Output

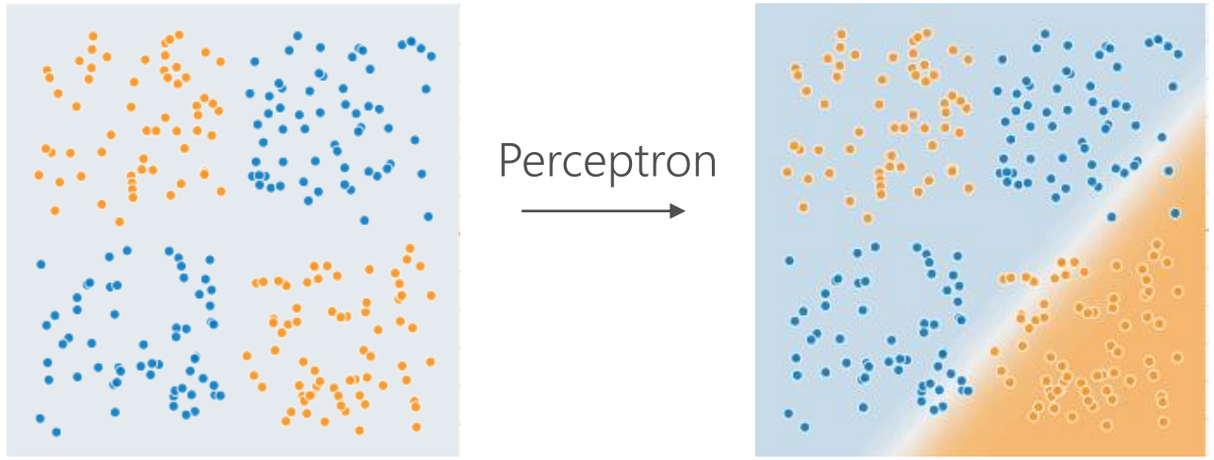
weights $w = w_1 w_2$ and bias b are learned from training examples

Perceptron – Nonlinear classification

Input example: $x = x_1 x_2, y$ Weights: $w = w_1 w_2$ Bias: b



$$\hat{y} = \begin{cases} 1 & \text{if } f(x) > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$



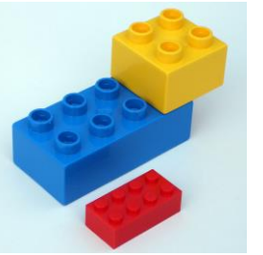
Input Output

Q.: How can we get non-linear boundary?

A1: Use non-linear features (e.g. $x_1 \rightarrow x_1^2$)

A2: Multi-layer perceptron

Multilayer Perceptron

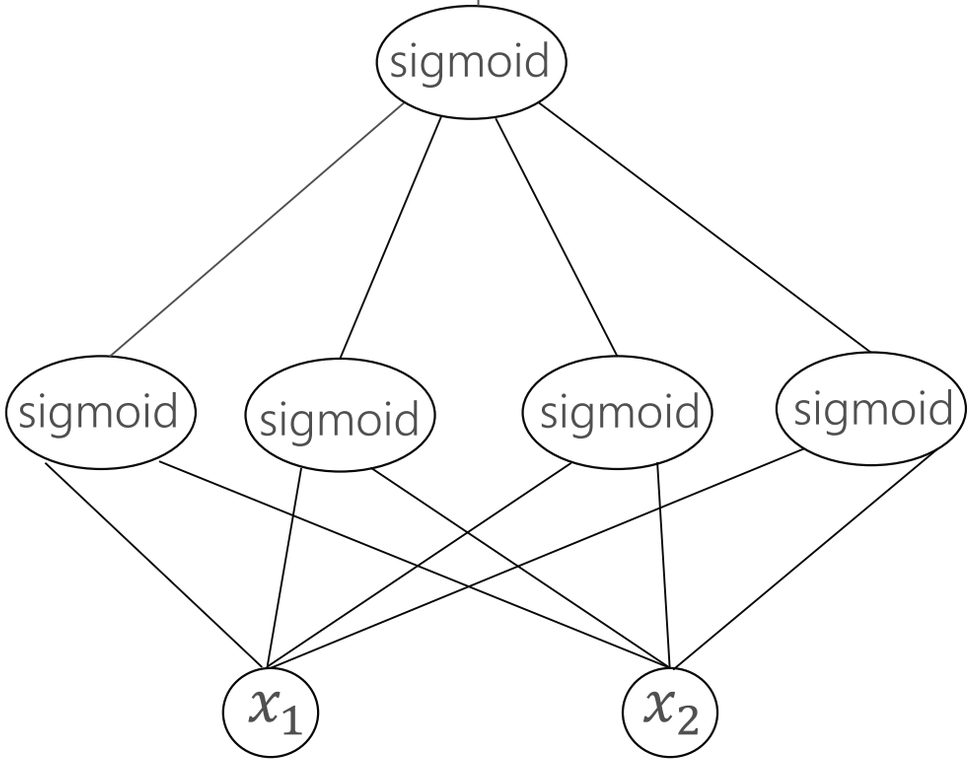


output

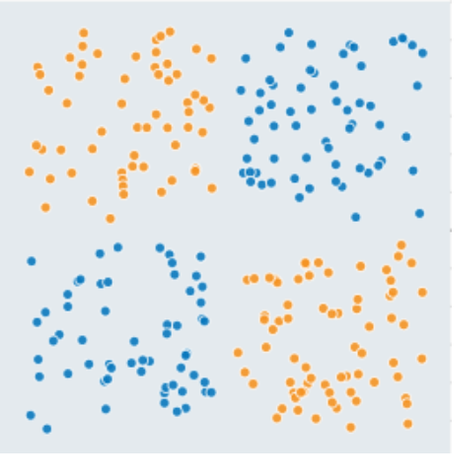
output layer

hidden layer

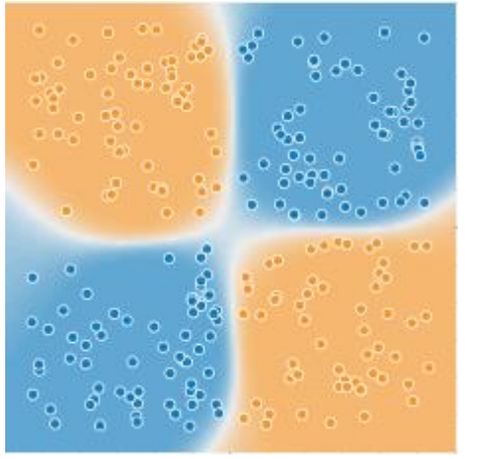
input layer



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



multilayer perceptron



Input

Output

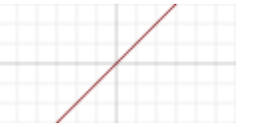
also known as Feed-Forward Neural Network and Deep Neural Network

Activation Functions

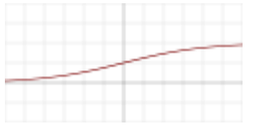
https://en.wikipedia.org/wiki/Activation_function

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

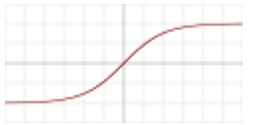
Identity $f(x') = x'$



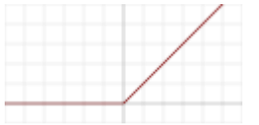
Sigmoid $f(x') = \sigma(x') = \frac{1}{1+e^{-x'}}$



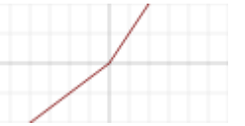
tanh (hyperbolic tangent) $f(x') = \frac{e^{x'}-e^{-x'}}{e^{x'}+e^{-x'}}$



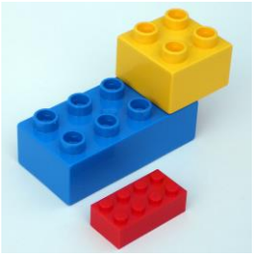
Rectifier linear unit (ReLU) $f(x') = \begin{cases} 0 \text{ for } x' < 0 \\ x' \text{ for } x' \geq 0 \end{cases}$

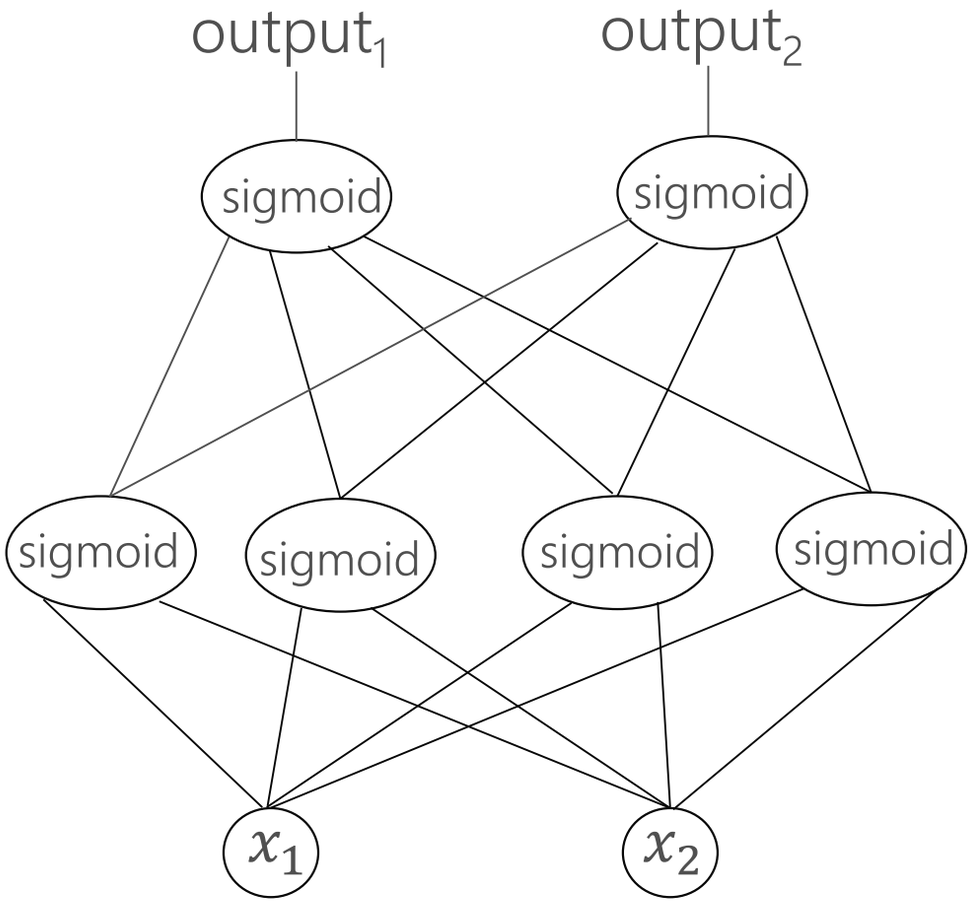


Parametric rectifier linear unit (PReLU) $f(x') = \begin{cases} \alpha x' \text{ for } x' < 0 \\ x' \text{ for } x' \geq 0 \end{cases}$

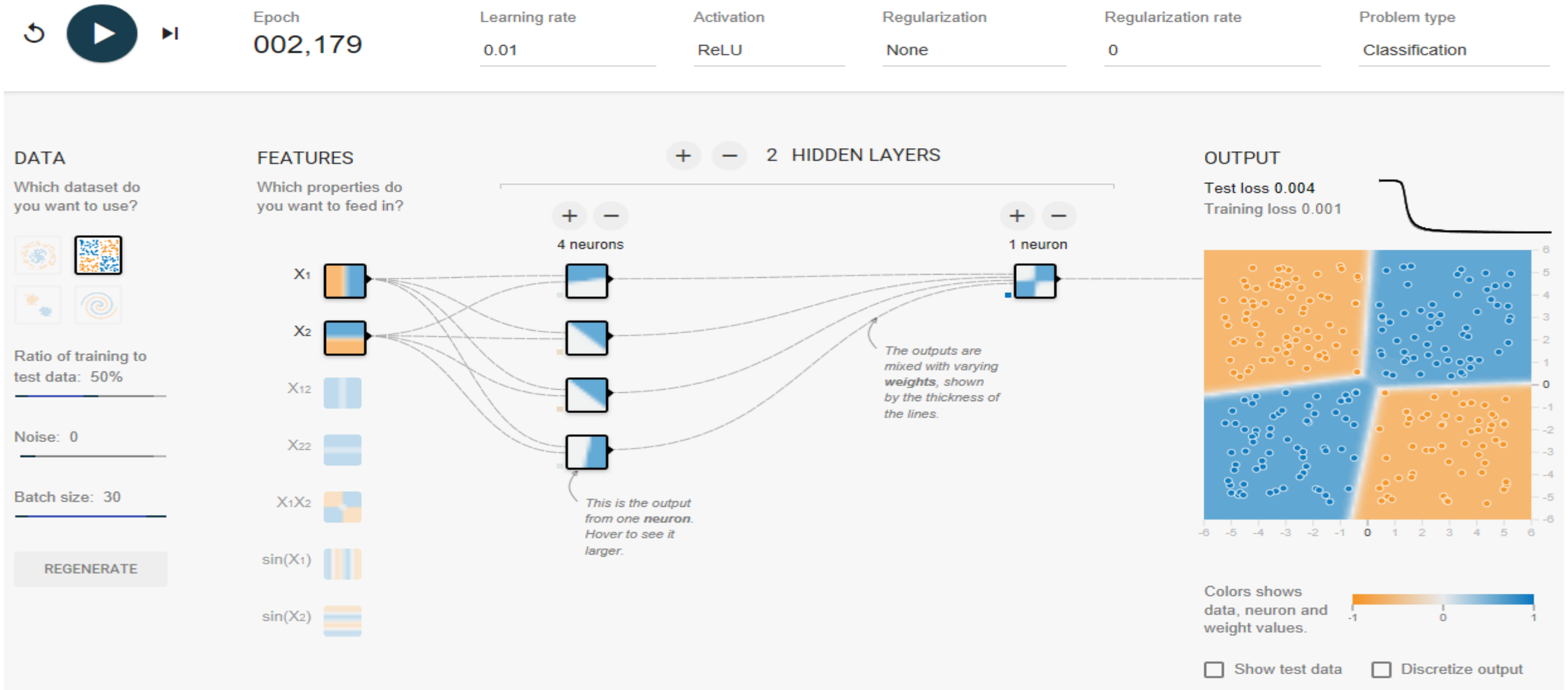


Multiple Outputs





Visualization of Feed-Forward Neural Net



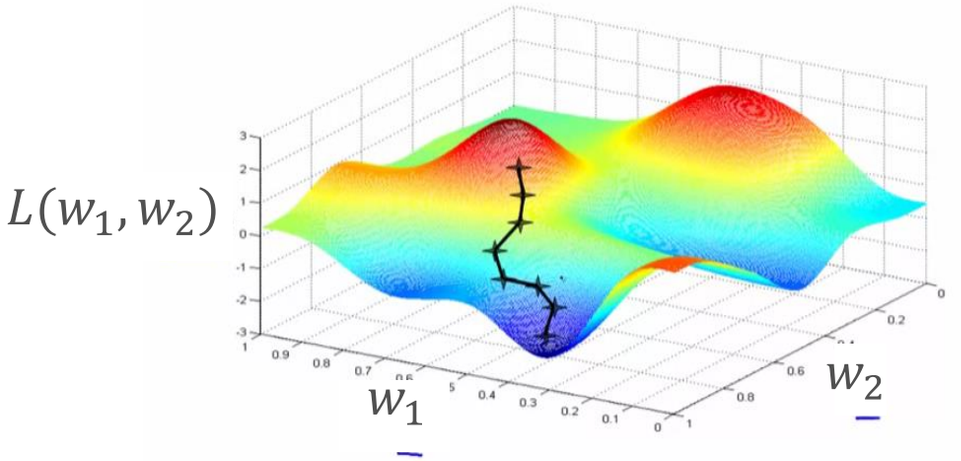
playground.tensorflow.org

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



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

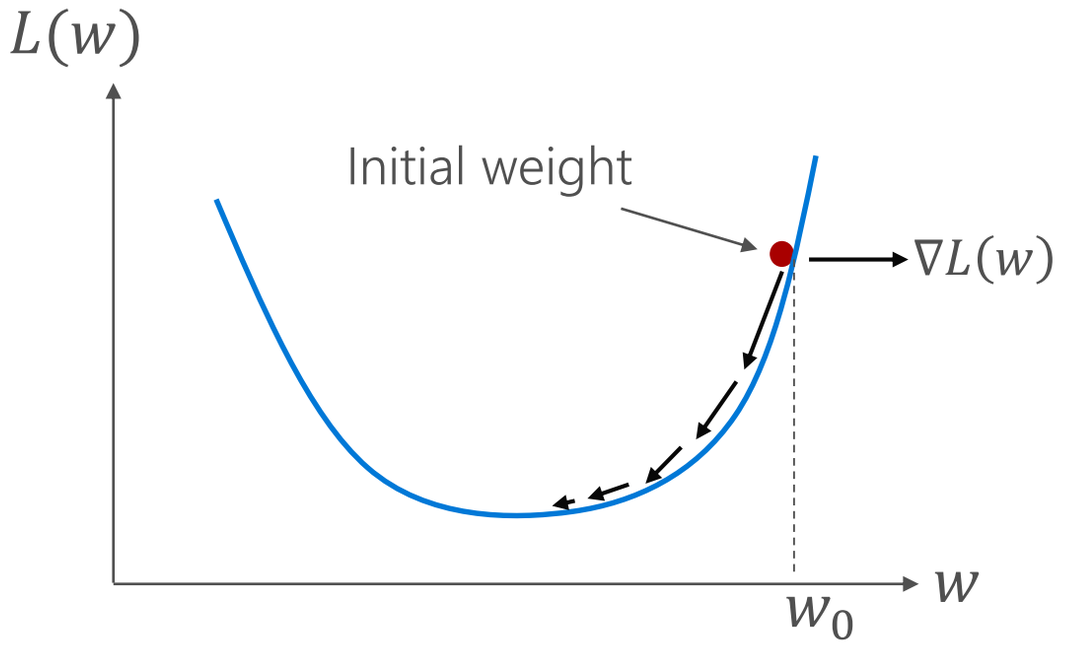
$\hat{y}_i$ is a function of weights w and biases b.

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

Batch gradient descent in machine learning

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

$$\nabla L(w) = \frac{1}{N}\sum_{i=1}^{N} \nabla L_i(w)$$

Initialization: $w = w_0$

While stopping criterion not met:

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

Need to go over all examples to complete a single optimization step.

With large N, gradient descent has a very slow convergence.

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

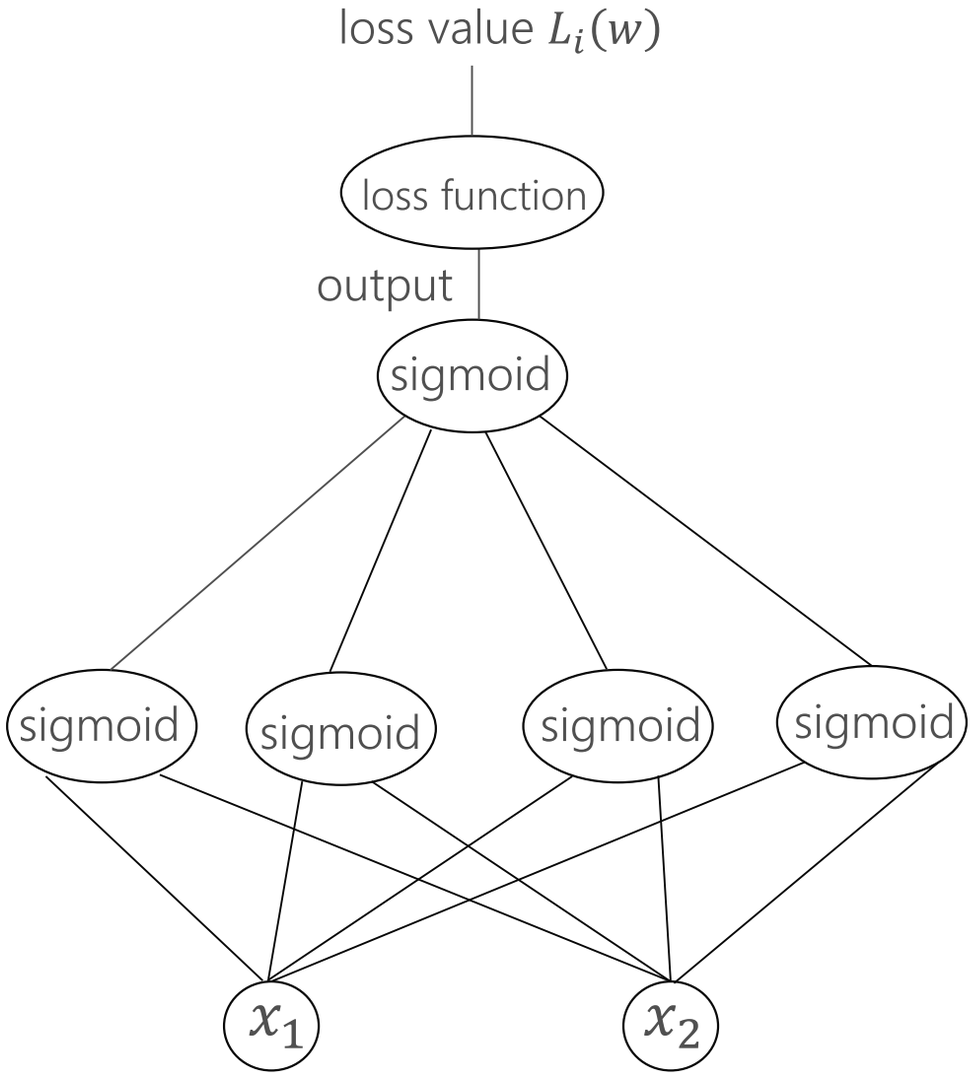
$$w = w - \frac{\alpha}{m}\sum_{i=1}^{m} \nabla L_i(w)$$

epoch

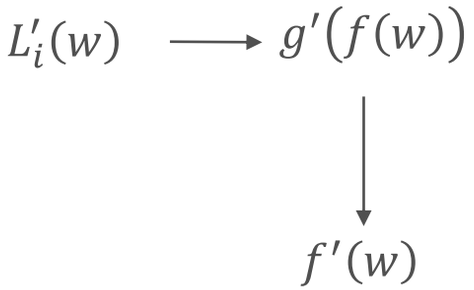
gradient from the i-th example in minibatch

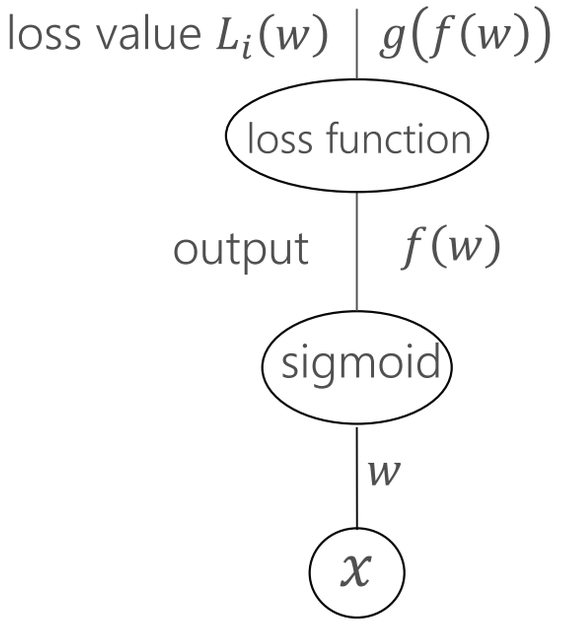
m – mini-batch size (default value 32 in Keras)

Computation of gradients in NN: Backpropagation



Backpropagation – single branch



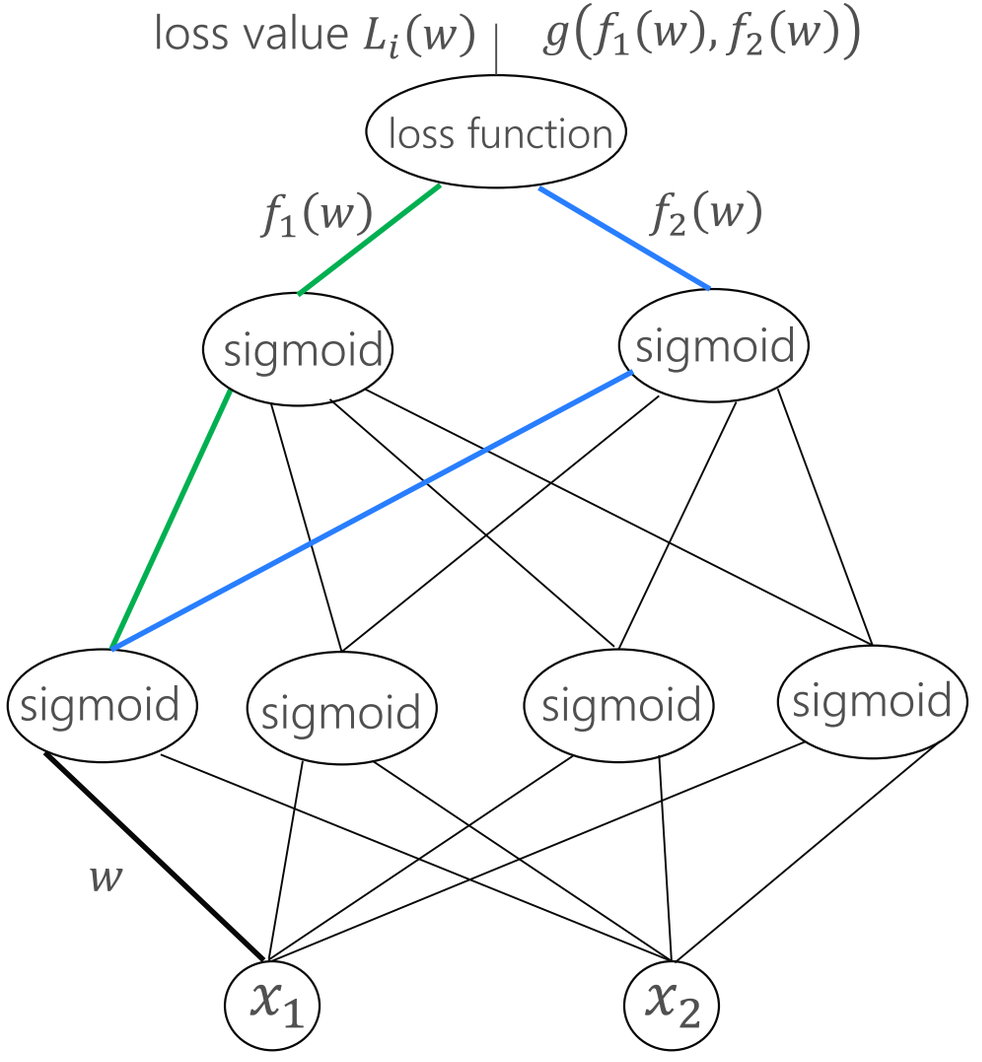


Chain rule

$$L_i(w) = g(f(w))$$

$$L_i'(w) = g'(f(w)) \cdot f'(w)$$

Backpropagation – multiple branches



Chain rule

$$L_i(w) = g\big(f(w)\big) \quad L_i'(w) = g'\big(f(w)\big) \cdot f'(w)$$

Total derivative

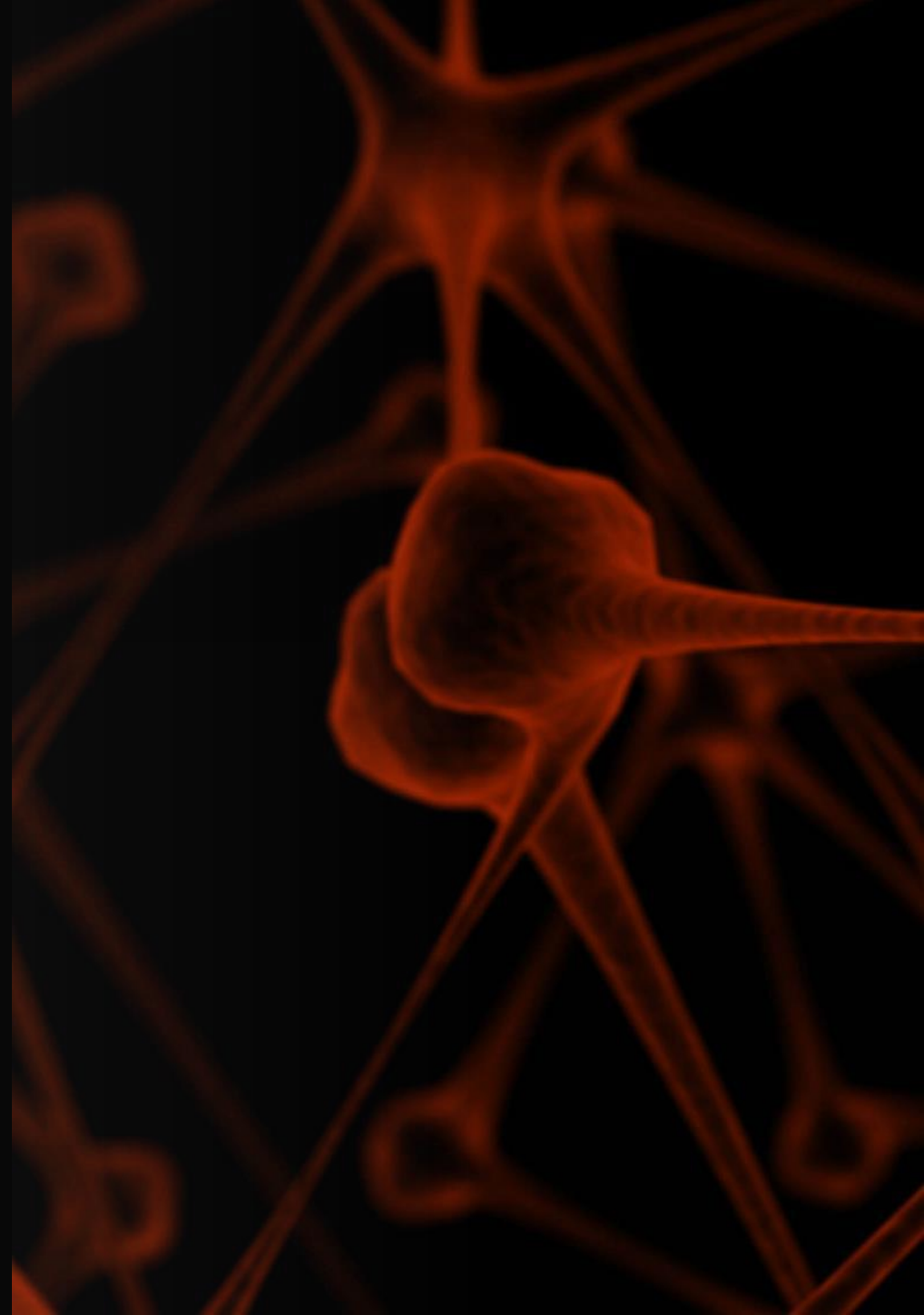
$$L_i(w) = g\big(f_1(w), f_2(w)\big)$$

$$L_i'(w) = \sum_{j=1}^{2} \frac{\partial g}{\partial f_j}\Big(f_j(w)\Big) \cdot \frac{\partial f_j}{\partial w}(w)$$

Training tricks

- Early stopping
- Initialization
- Tuning hyperparameters
- 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

How to apply feed-forward NN to time series forecasting



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

One step ahead forecasting of univariate time series using feed-forward neural network

Open 2_one_step_FF_univariate.ipynb

Initialization of weights

Reminder: Minibatch Stochastic Gradient Descent

Initialization: $w = w_0$

While stopping criterion not met:

Shuffle examples randomly

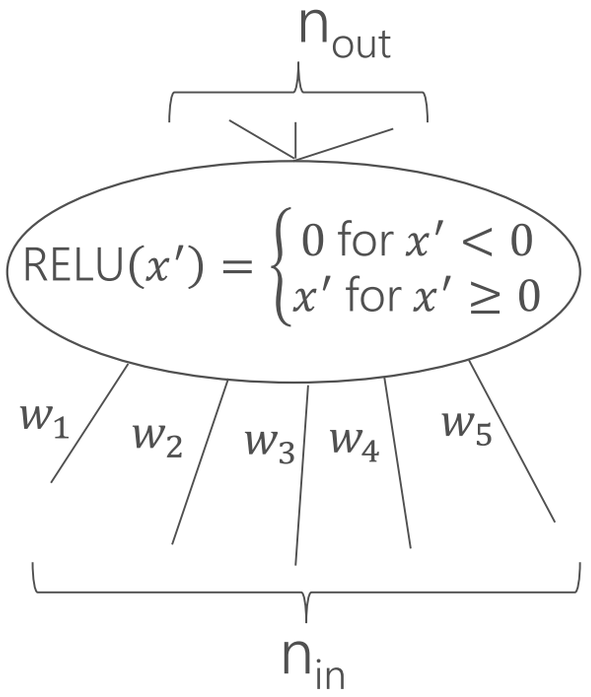
Partition examples into batches of size m

For $\mathbf{minibatch=1...N/m}$ examples

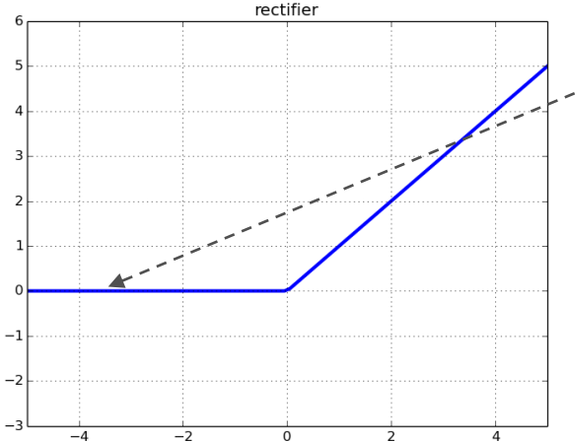
$$w = w - \frac{\alpha}{m}\sum_{i=1}^{m} \nabla L_i(w)$$

What is a good value of w_0?

Initialization of weights



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



large weights
gradient=0, no
backpropagation, “dead neuron”

$x' = w \cdot x$

- Initialize all w_i with zeros
- $w_i \sim N(0, A)$
- Xavier Glorot uniform

$$w_i \sim \text{Uniform}\left[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right]$$

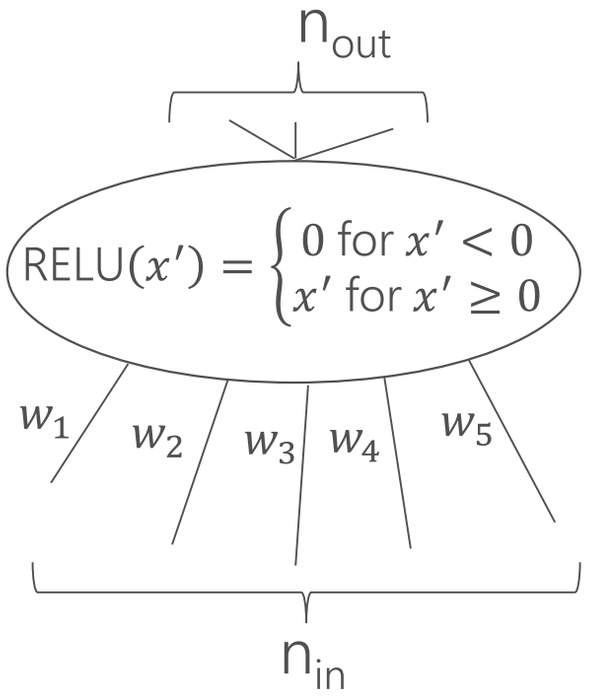
Hands-on Quiz: Initialization of weights

Open Quiz_weight_initialization.ipynb

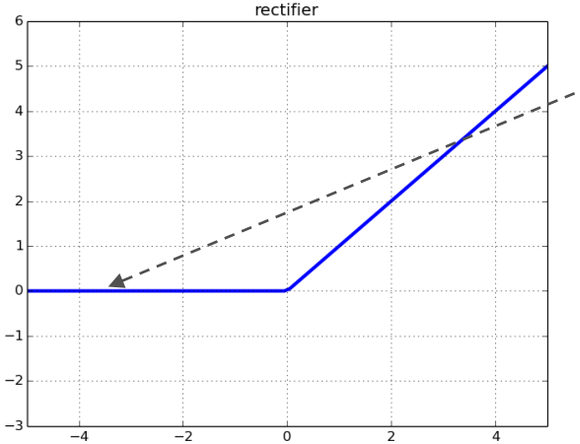
Quiz Results

Initializer	MAPE
Zeros()	0.13
RandomNormal()	between 0.015 and 0.02
glorot_uniform()	between 0.015 and 0.02

Initialization of weights



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



large weights
gradient=0, no
backpropagation, “dead neuron”

$$x' = w \cdot x$$

- Initialize all w_i with zeros - the output of all neurons will look the same ✗
- $w_i \sim N(0, A)$ - weights might be close to 0 and cause vanishing gradients problem ✗
- Xavier Glorot uniform ✓

$$w_i \sim \text{Uniform}\left[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right]$$

Multiple steps ahead forecasting of multivariate time series using feed-forward neural network

ReferenceNotebooks/3_multivariate_FF_vector_output.ipynb

Network tuning

Hyperparameters

- Architecture

  number of layers, number of cells in each layer, connections between cells

- Loss function

  weight of regularization term

- Optimization

  learning rate, mini-batch size, stopping criterion, initialization

Tuning hyperparameters of feed-forward NN using grid search

scripts/FF_multi_step_multivariate/README.md

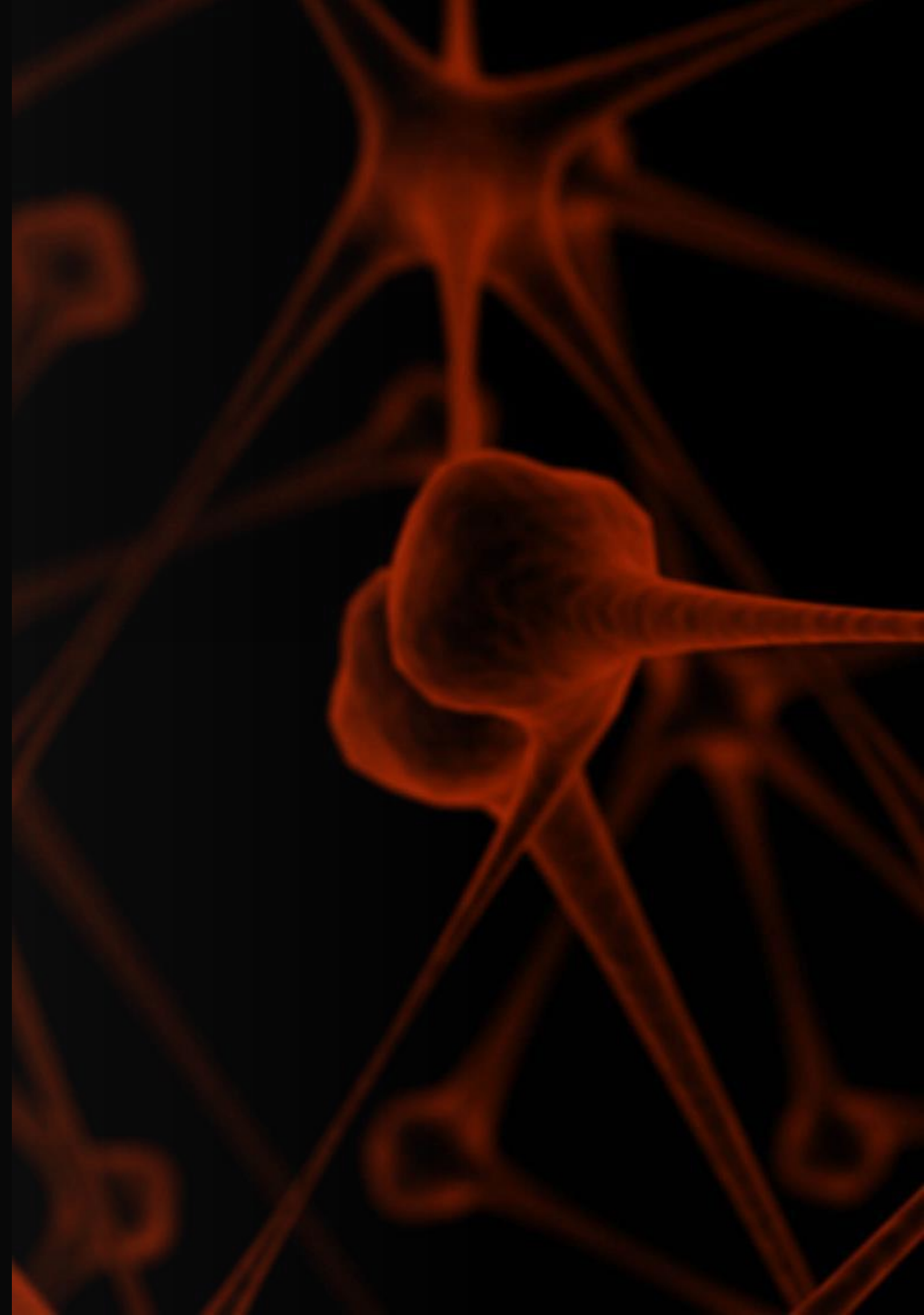
Advanced techniques for tuning hyperparameters

- Bayesian optimization Bayesian optimization with robust Bayesian neural networks
- Multi-armed bandit techniques Hyperband: A novel bandit-based approach for hyperparameter optimization
- Neural architecture search Neural architecture search with reinforcement learning
- Genetic programming Population based training of neural networks

Performance comparison

Algorithm	MAPE (Mean Absolute Percentage Error)
ARIMA	4.9
Feedforward Neural Network	4.27

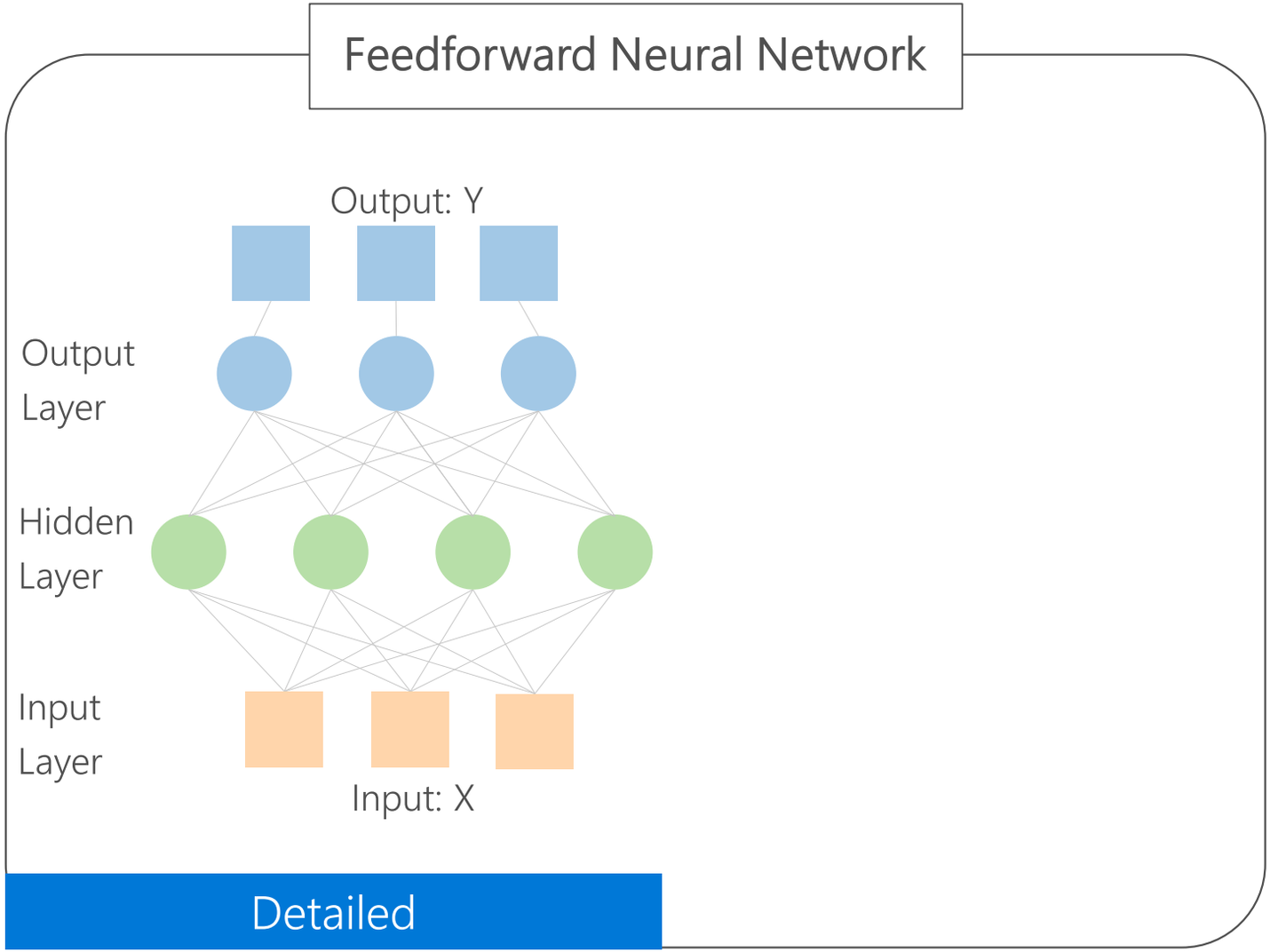
Introduction to Recurrent Neural Networks



Agenda

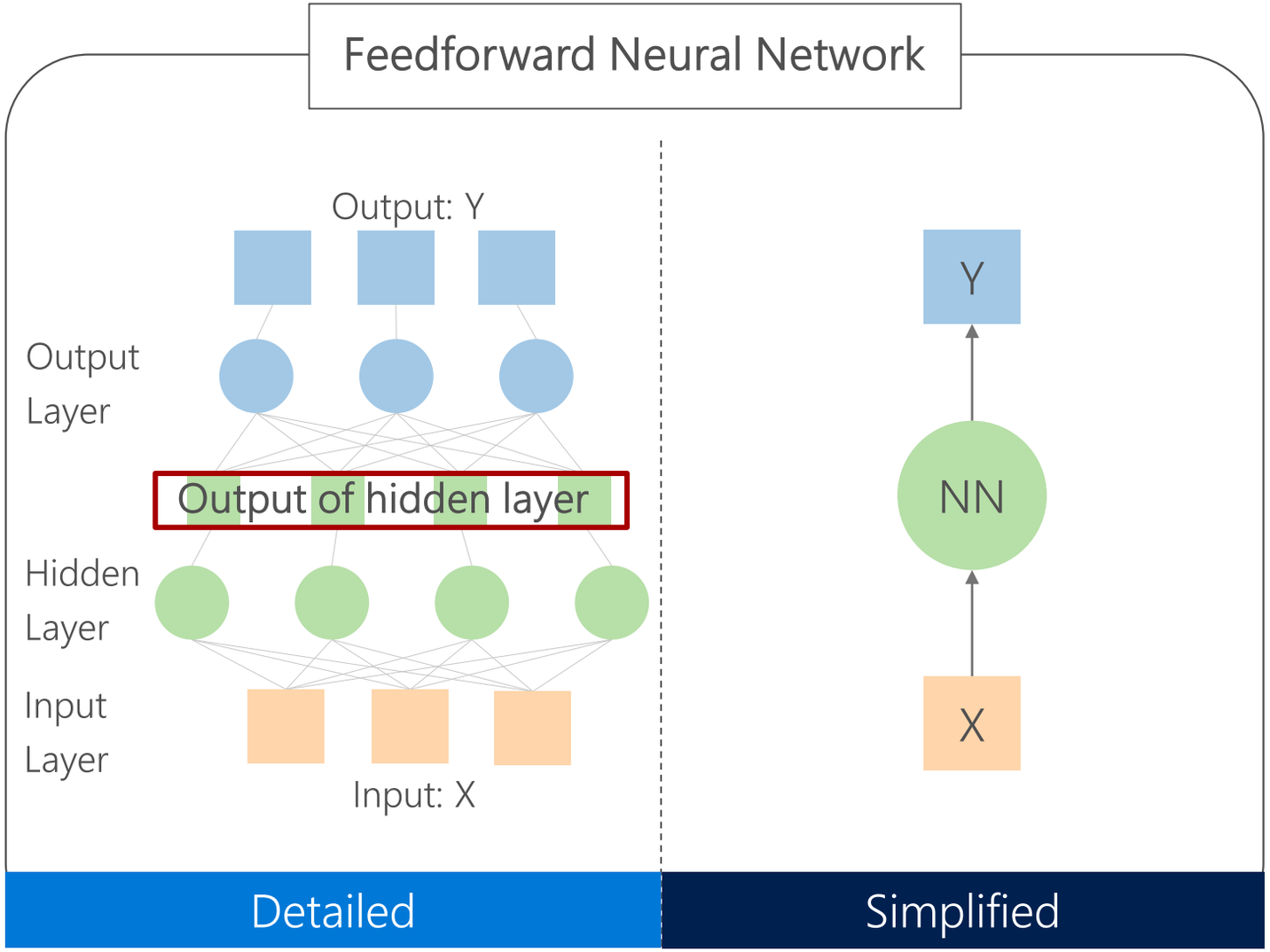
- What are RNNs?
- How RNNs are trained: Backpropagation through time (BPTT)
- Vanilla RNN and its gradient problems
- Other RNN units
  - GRU
  - LSTM
- RNN stacking

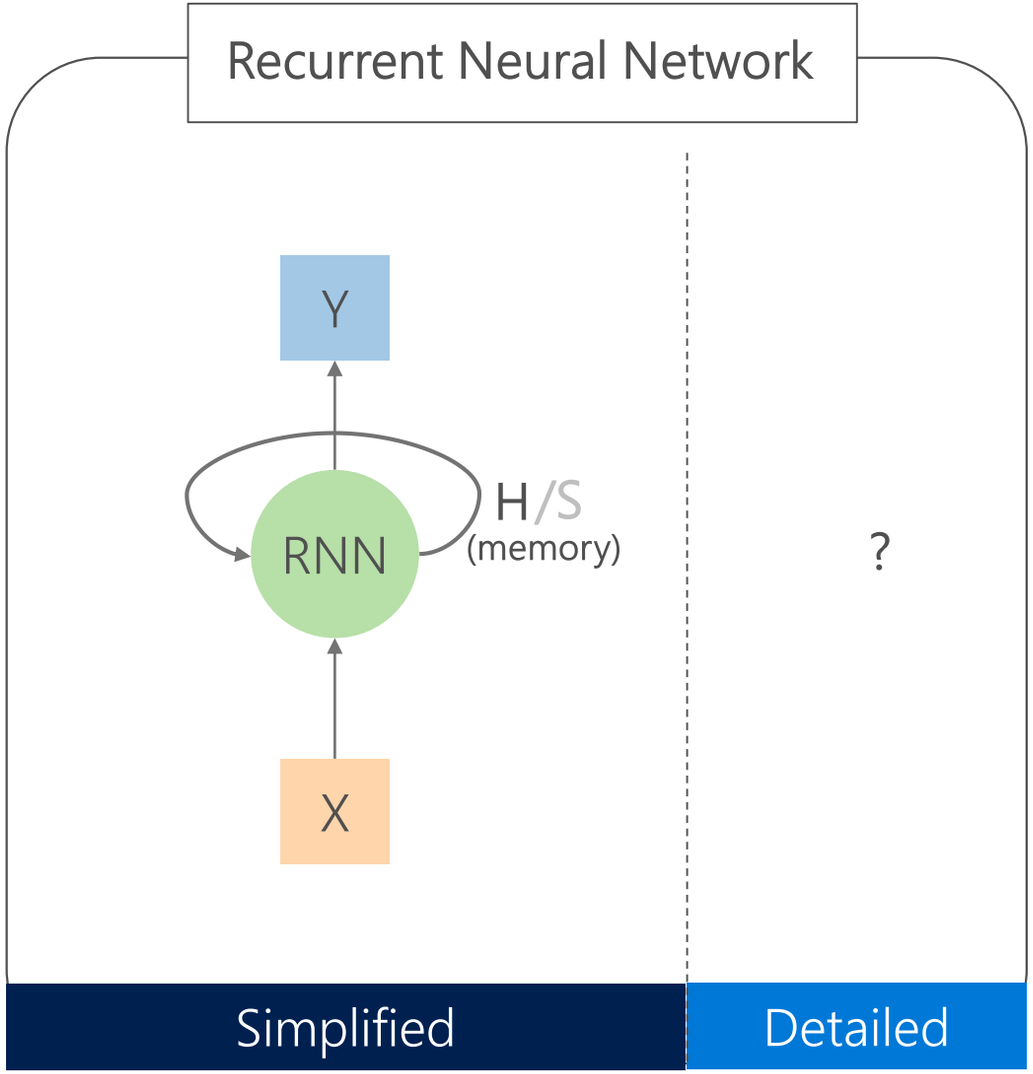
What are RNNs?



What are RNNs?

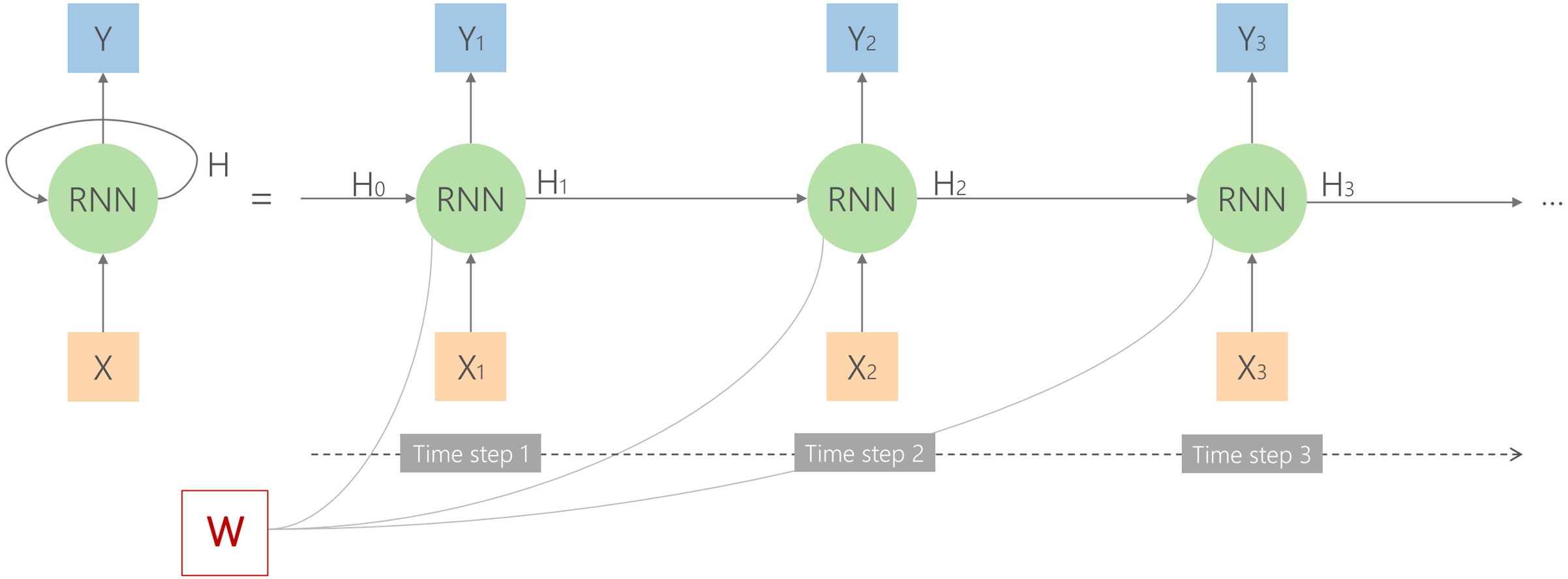
RNN has internal hidden state which can be fed back to network





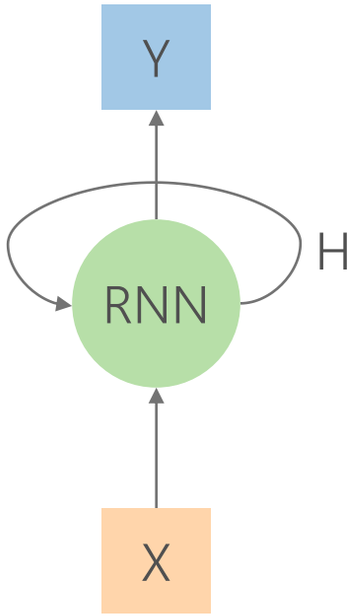
Unrolled RNN

The same weight and bias shared across all the steps



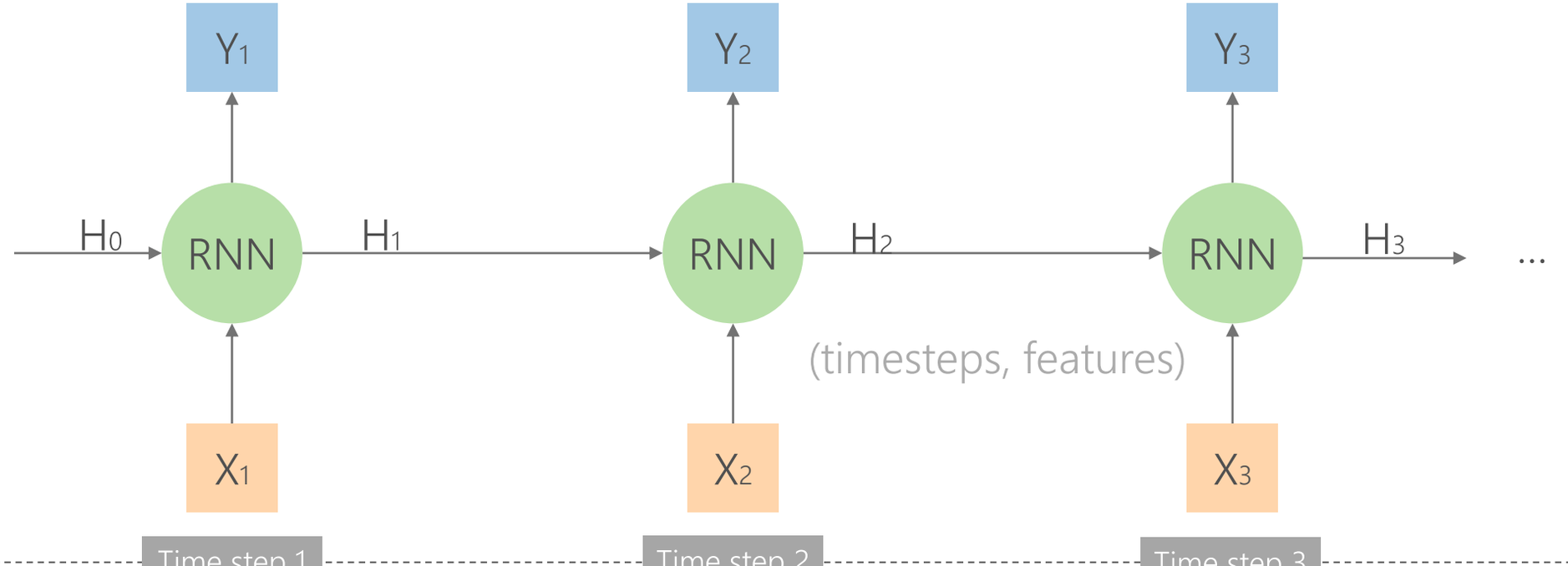
In Keras you will see "timesteps" when set data input shape

Unrolled RNN

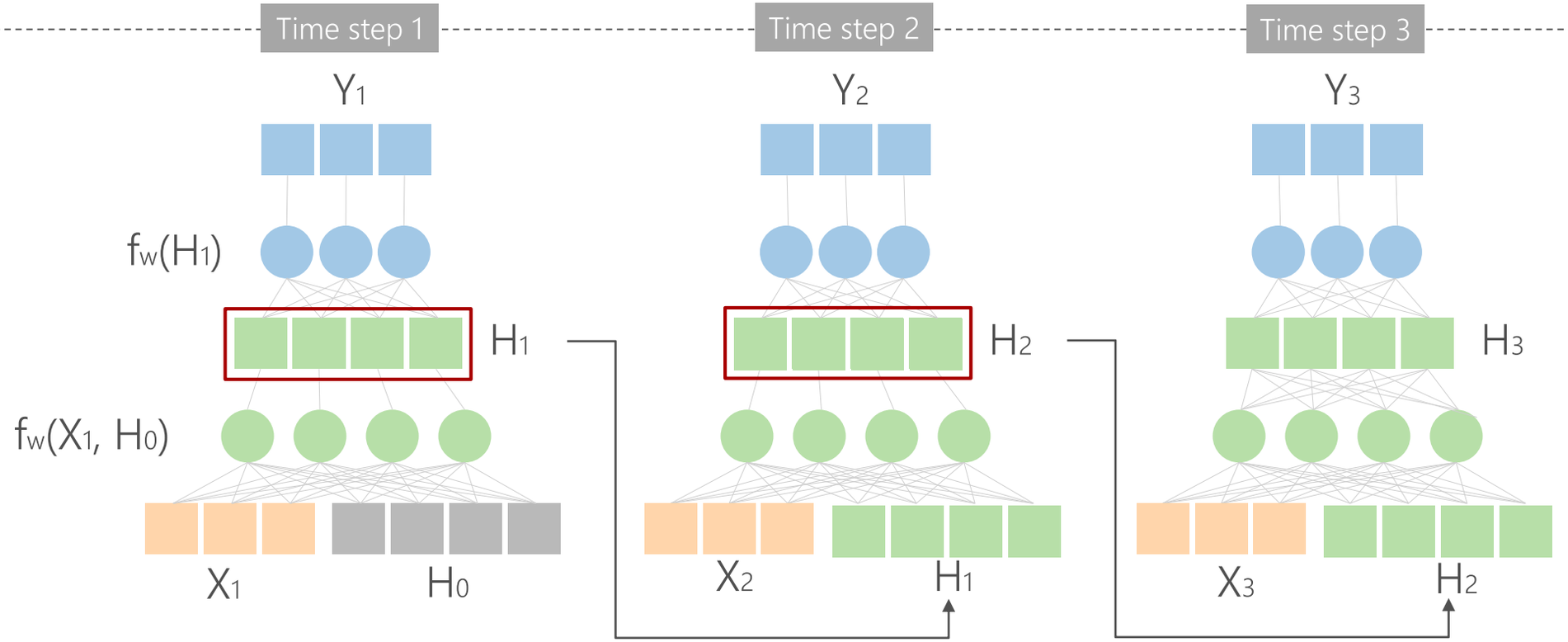


=

Simplified



Detailed

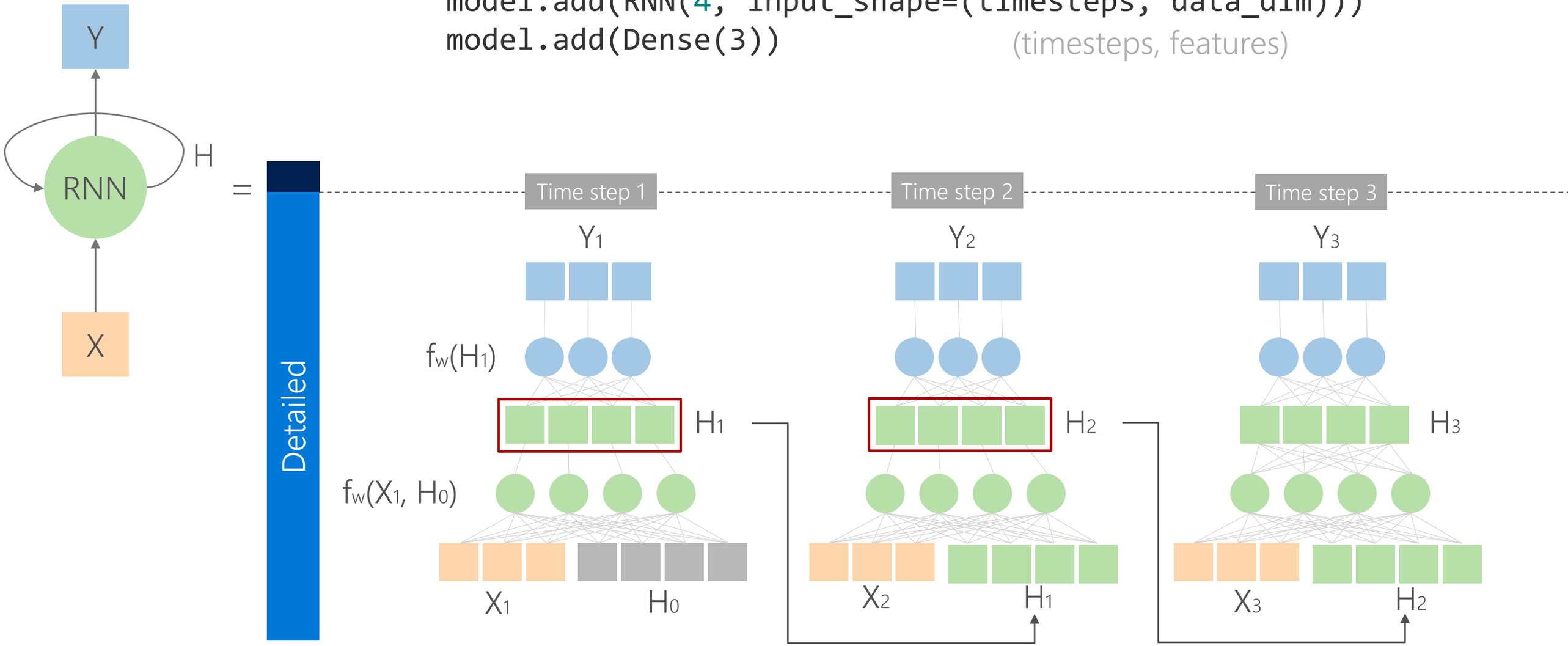


Unrolled RNN

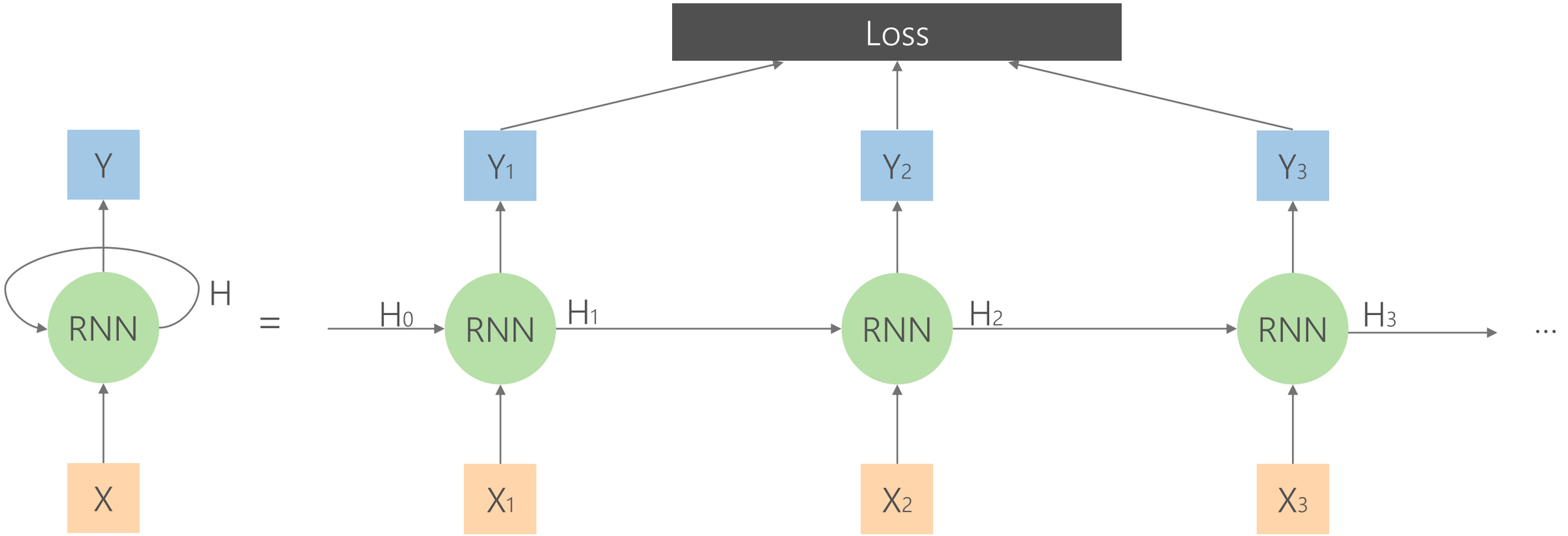
In Keras, the parameter "units" is dimensions of hidden state.
(Think of it as feedforward neural network number of units in hidden layer.)

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

(timesteps, features)



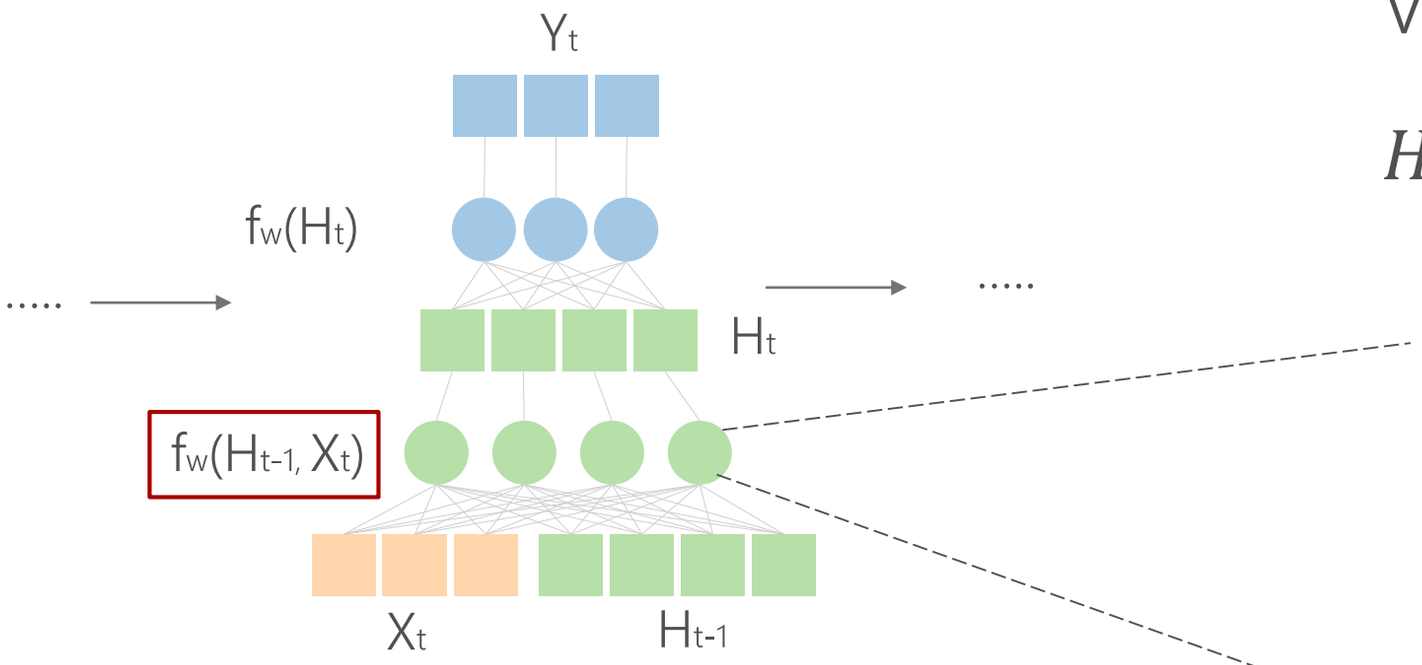
Backpropagation through time (BPTT)



Forward through entire sequence to compute loss

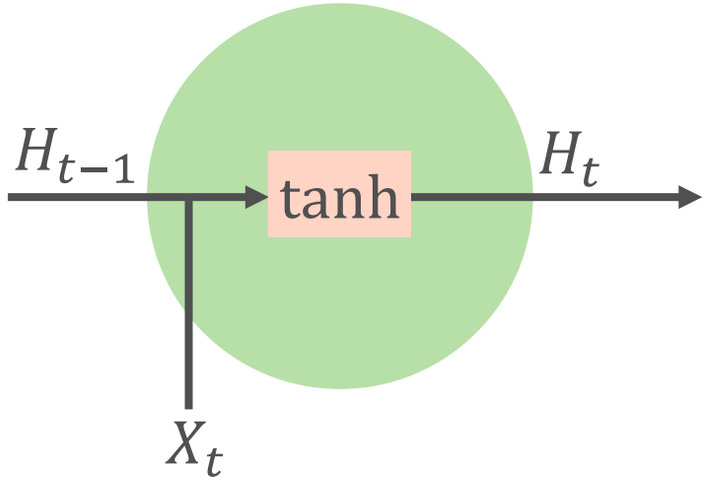
then backward through entire sequence to compute gradient

Vanilla RNN



Vanilla RNN:

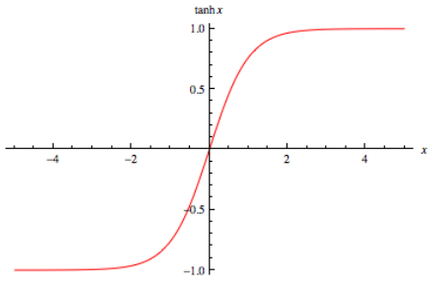
$$H_t = \tanh(W_h H_{t-1} + W_x X_t)$$
$$= \tanh(\mathbf{W} \cdot [H_{t-1}, X_t])$$



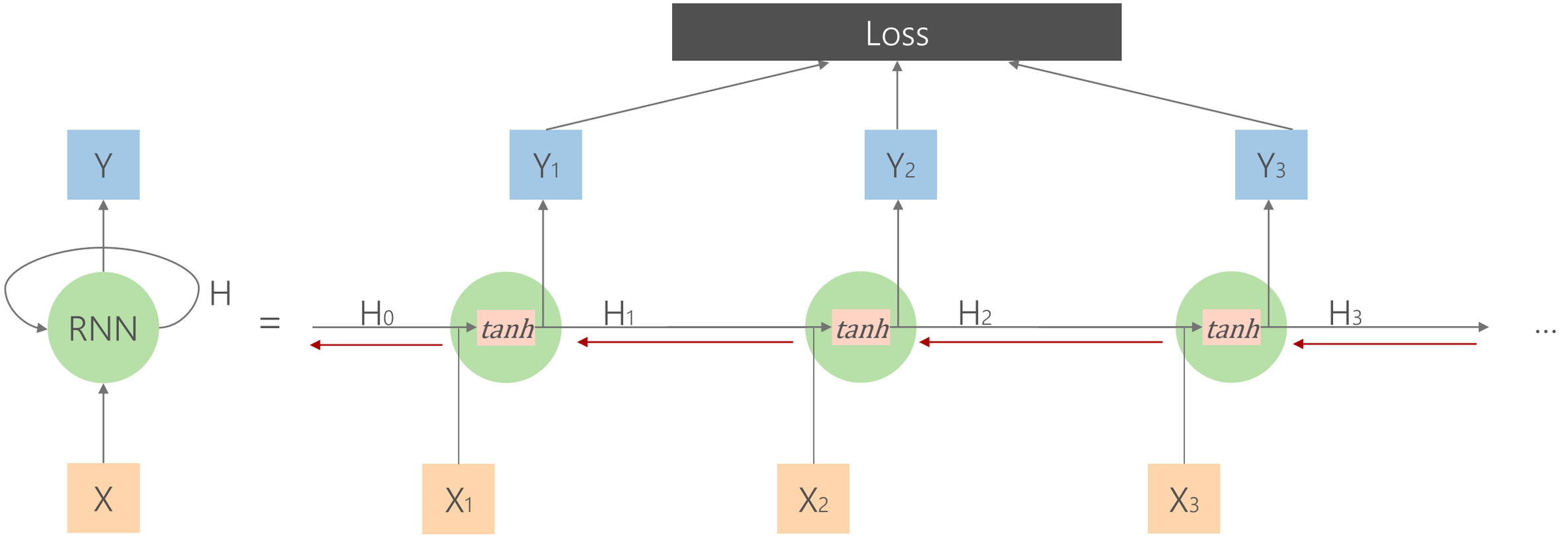
some function with parameter W

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

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



Vanilla RNN BPTT

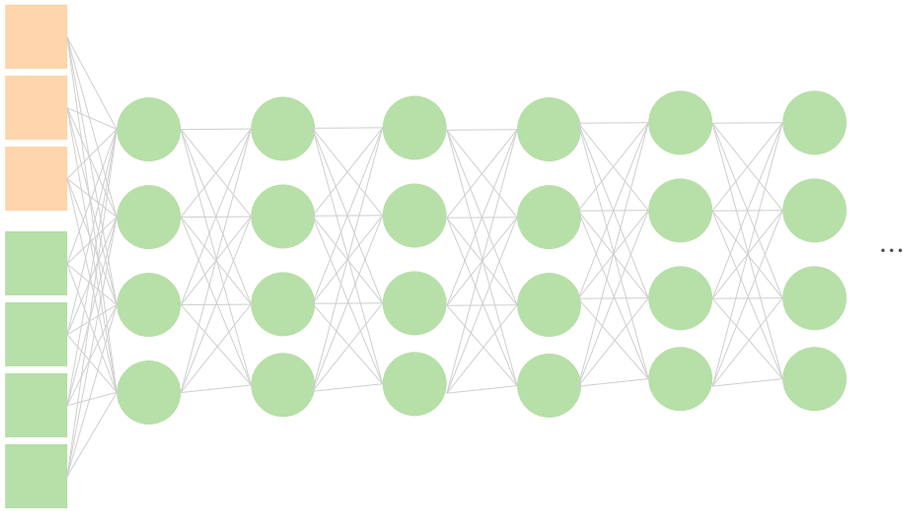


Computing gradient of h_0 involves **repeated tanh and many factors of W**

Vanilla RNN Gradient Problems

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

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



100 time steps is similar to 100 layers feedforward neural net

Exploding Gradient

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

  Clip the gradient when it goes higher than a threshold

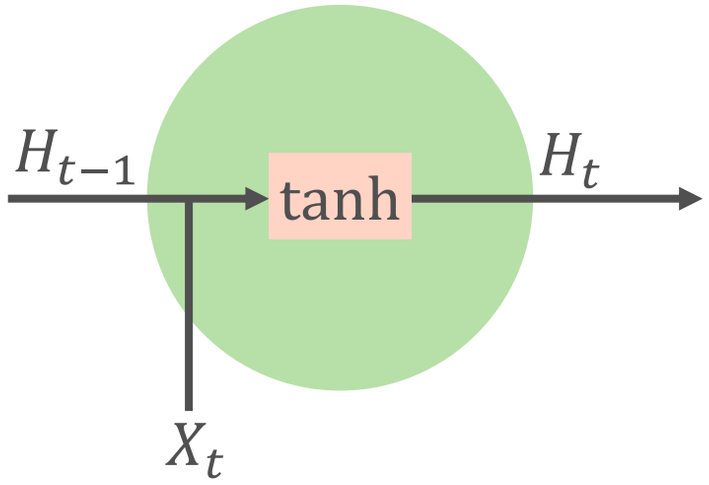
Vanishing Gradient

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

Gated Recurrent Unit (GRU)

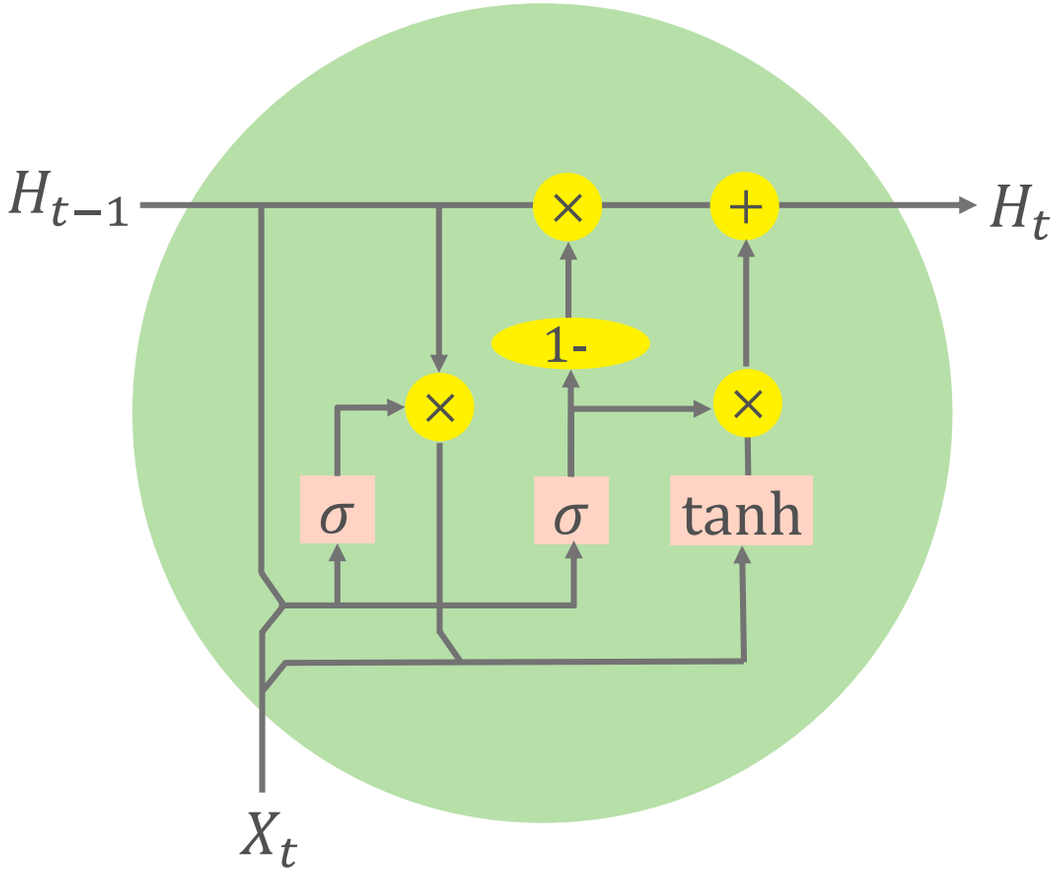
Vanilla RNN:

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

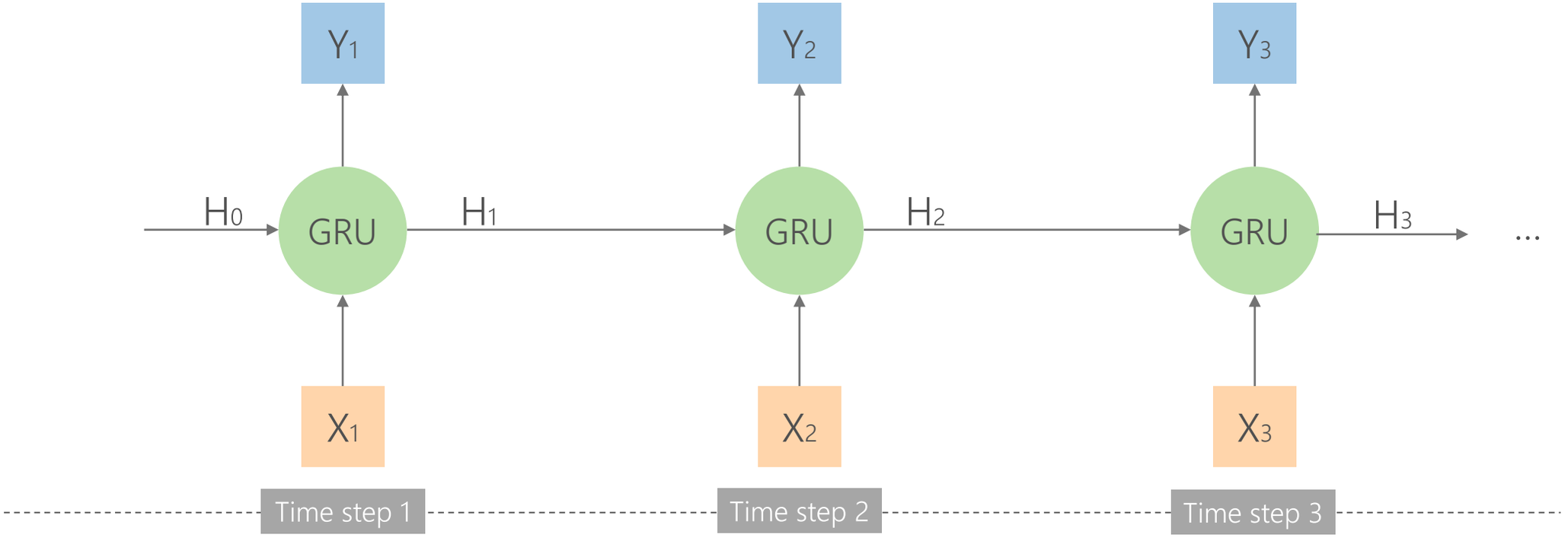


GRU:

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

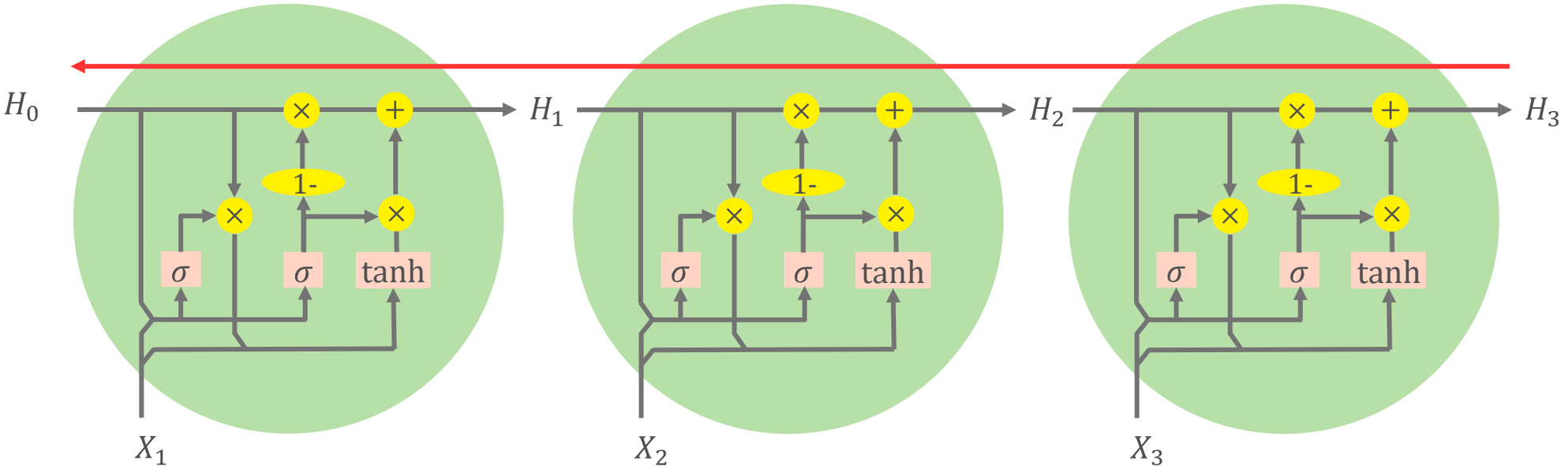


GRU

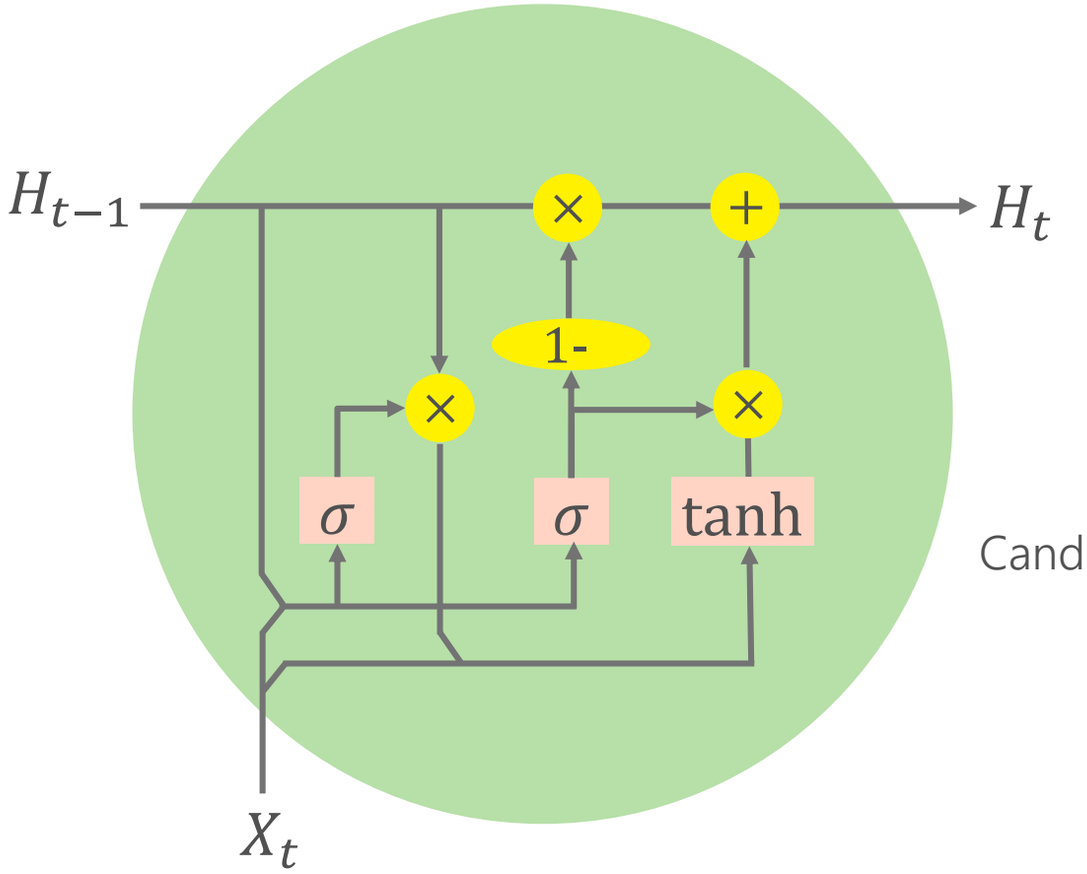


Uninterrupted gradient flow

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



Gated Recurrent Unit (GRU)



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

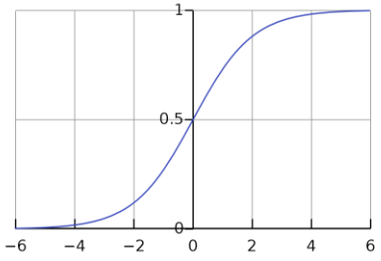
Update gates: $z_t = \sigma(W_z \cdot [H_{t-1}, X_t])$

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

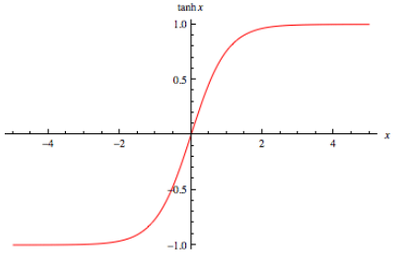
Reset gates: $r_t = \sigma(W_r \cdot [H_{t-1}, X_t])$

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

σ
(0,1)

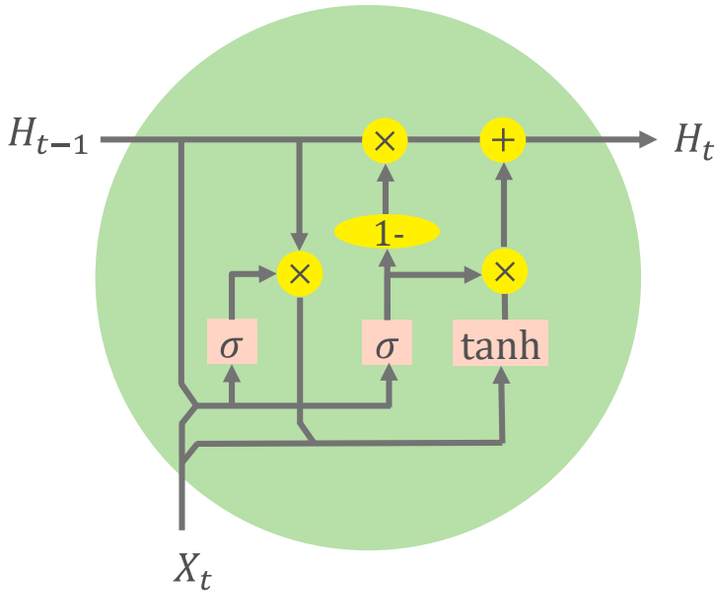


tanh
(-1, 1)

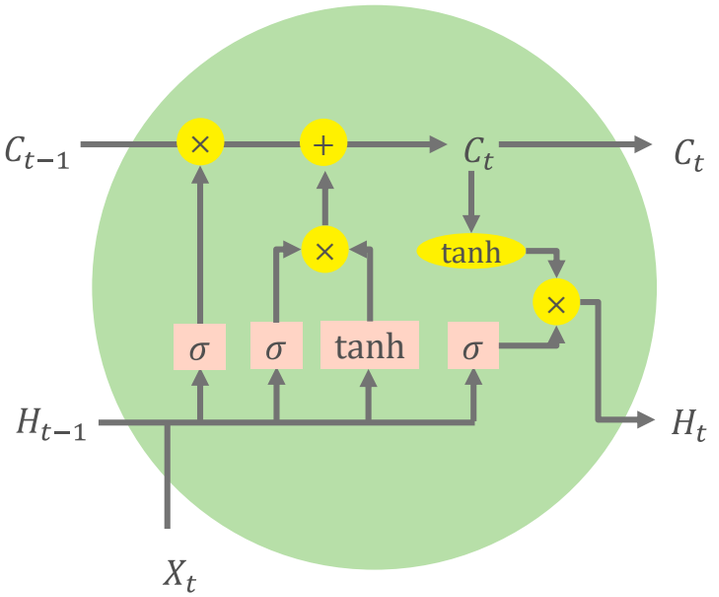


GRU vs LSTM (Long Short Term Memory)

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

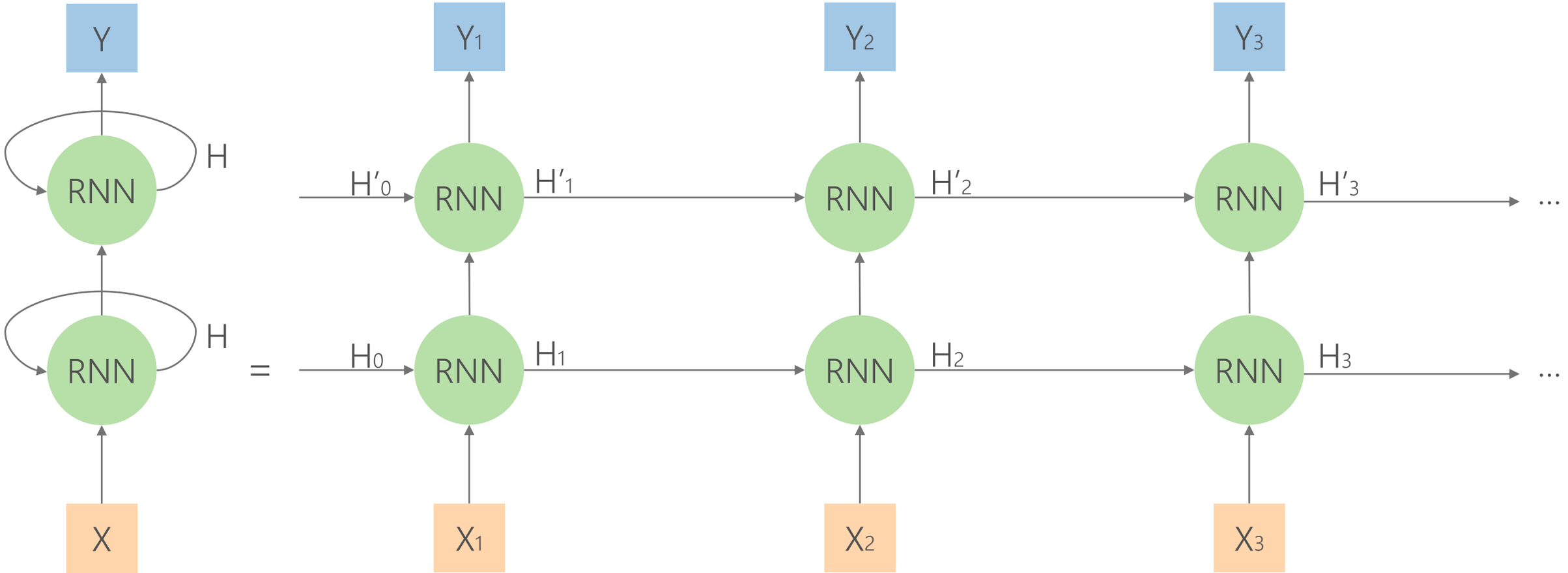
Output gates: $o_t = \sigma(\ W_o \cdot [H_{t-1}, X_t]\ )$

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

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

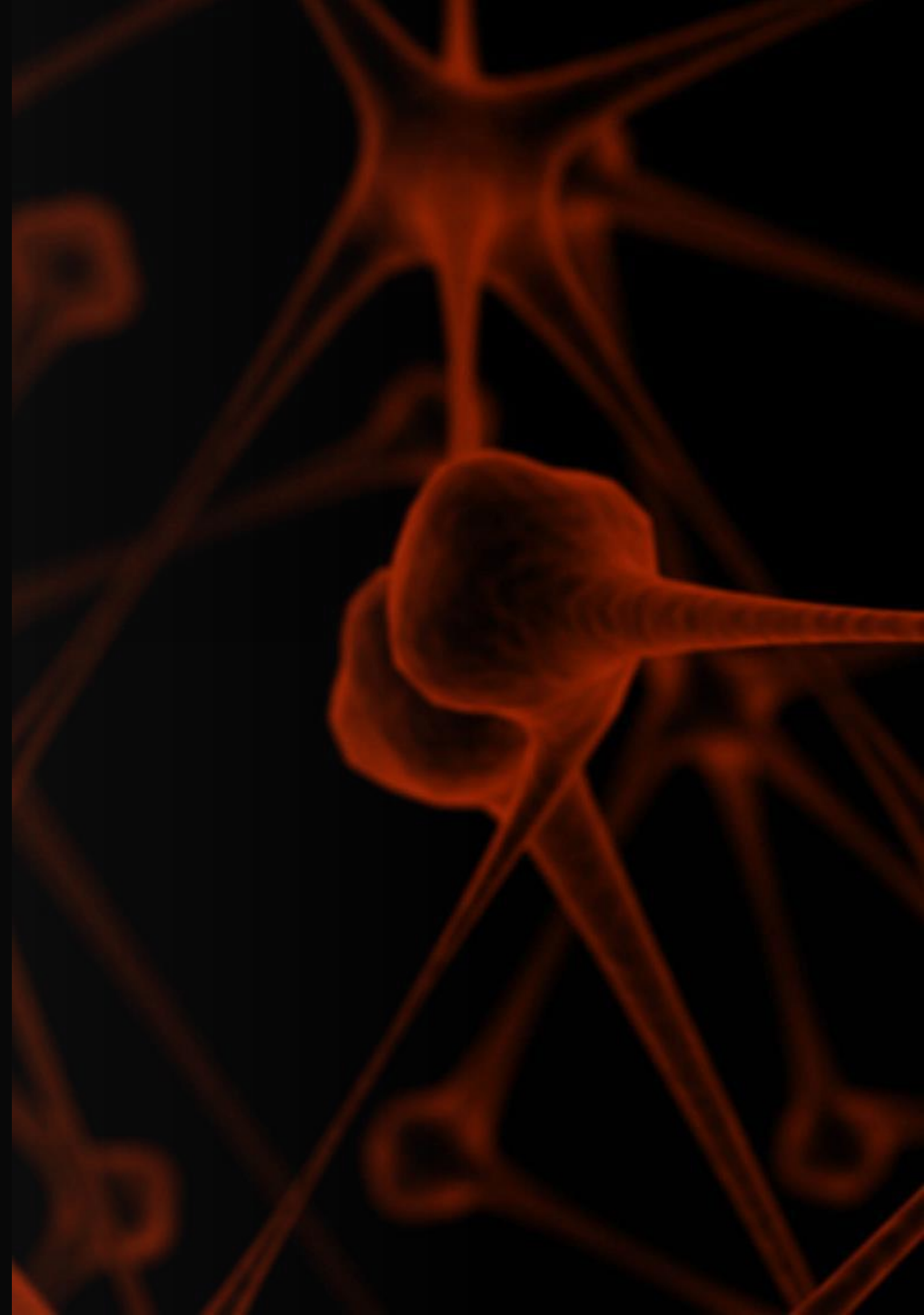
RNN Stacking

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



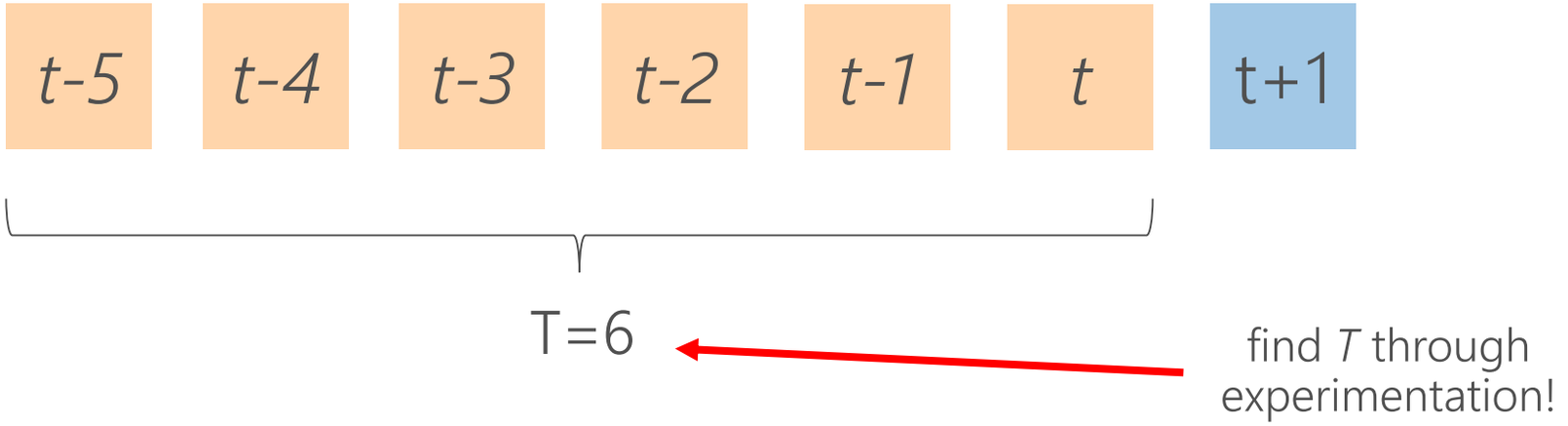

```
model = Sequential()
model.add(RNN(4, return_sequences=True, input_shape=(timesteps, data_dim)))
model.add(RNN(4))
```

RNN for one step time series forecasting



One-step forecast

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



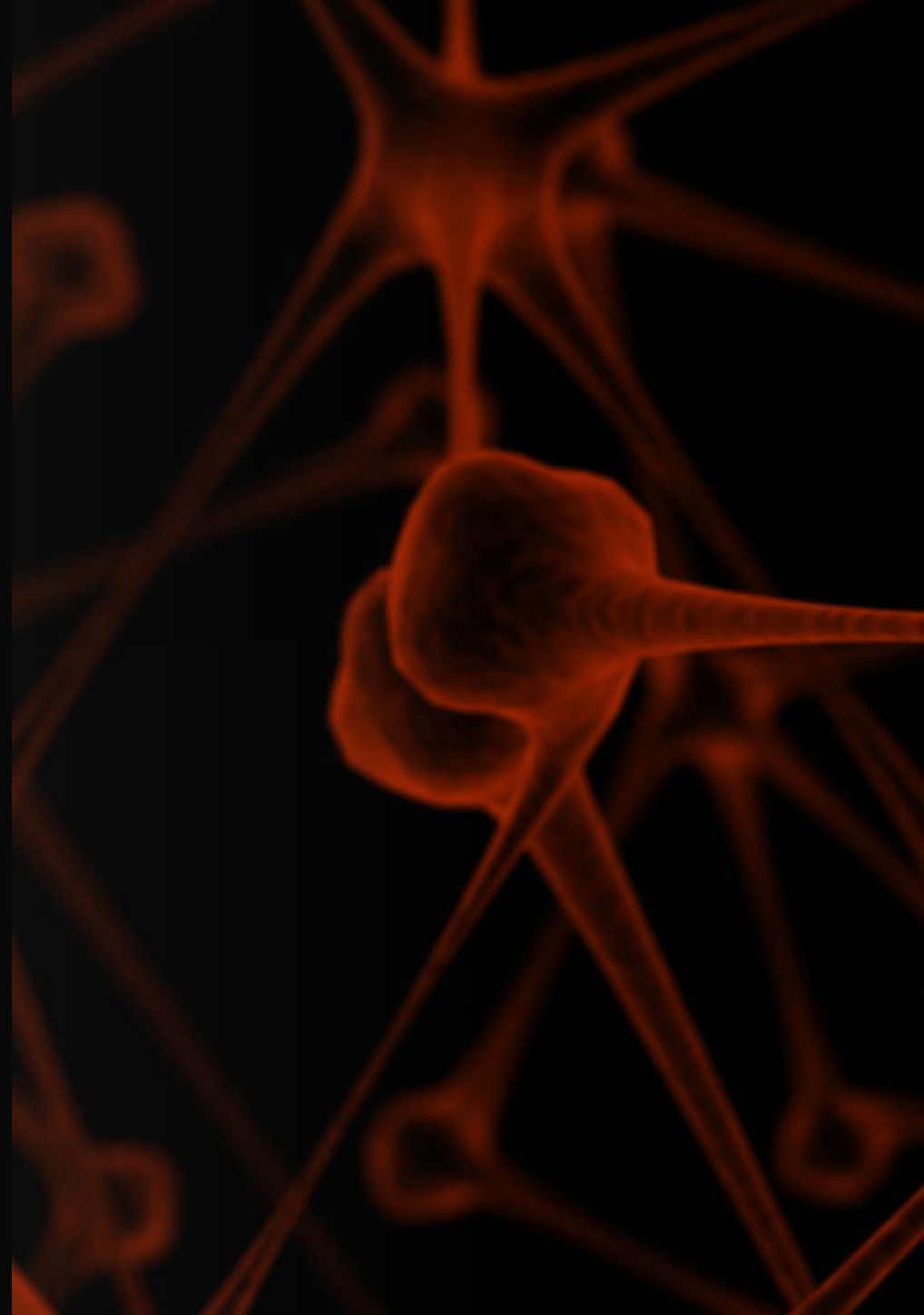
Hands-on Experiment

Open: 3_one_step_RNN_univariate.ipynb

Hands-on Quiz

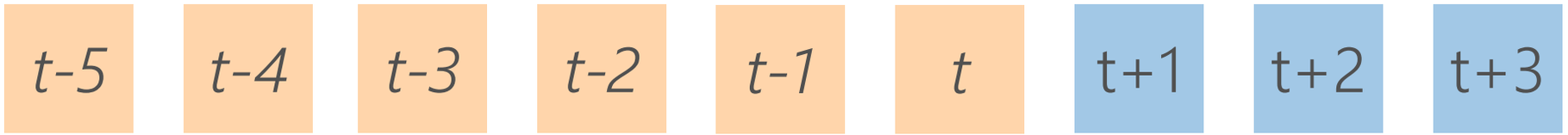
Open: Quiz_one_step_RNN_multivariate.ipynb

RNN for multi-step time series forecasting

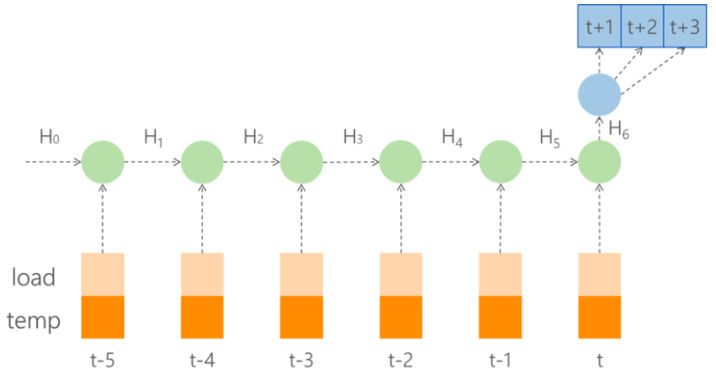


Multi-step forecast

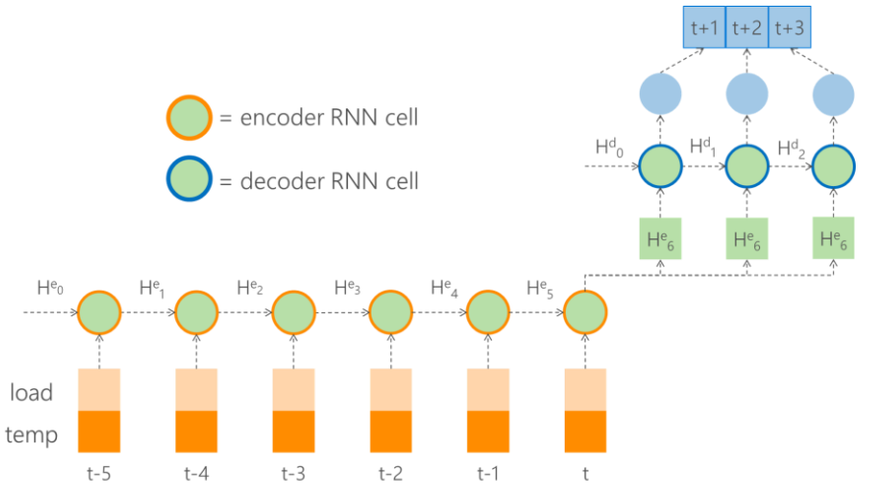
- Assuming we are at time *t* …
- … predict the values at times (*t+1*, … , *t+HORIZON*) …
- … conditional on the previous *T* values of the time series



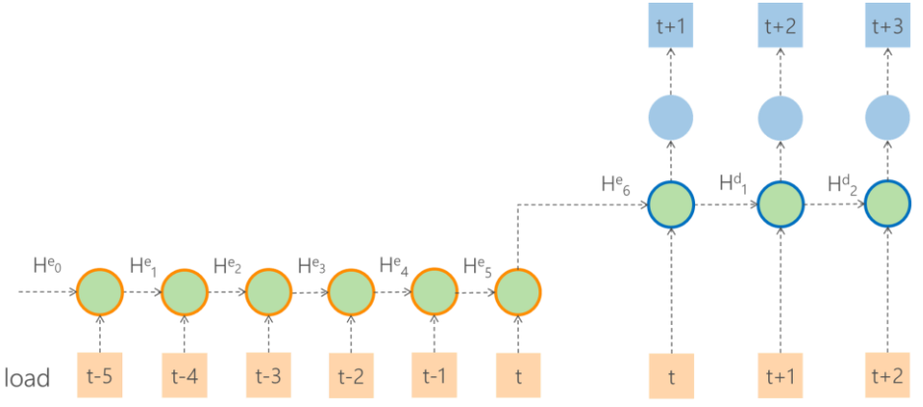
Multi-step forecast



Vector output

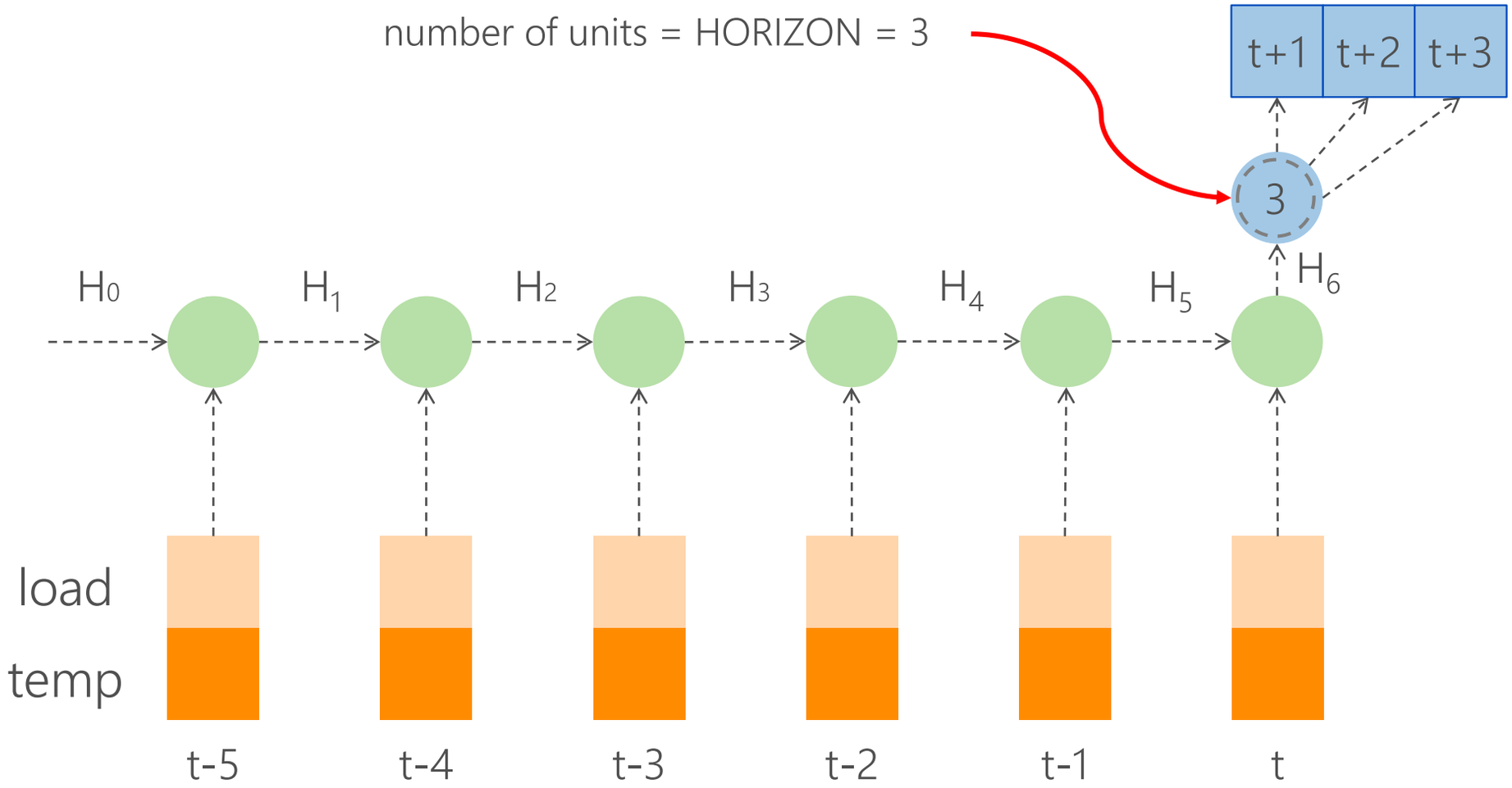


Simple encoder-decoder



Encoder-decoder with teacher forcing

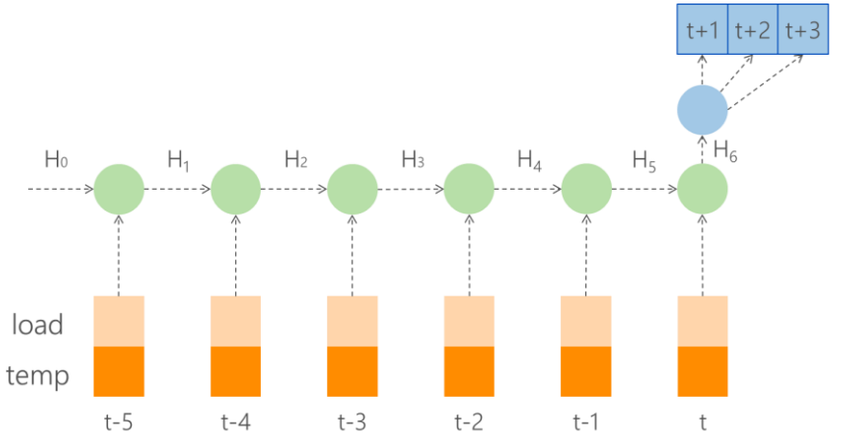
Vector output approach



Multi-step forecasting using a vector output RNN

Open 4_multi_step_RNN_vector_output.ipynb

Vector output approach

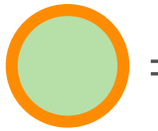


- Simplest to implement
- Fastest to train
- Does not model dependencies between predicted outputs

Performance comparison

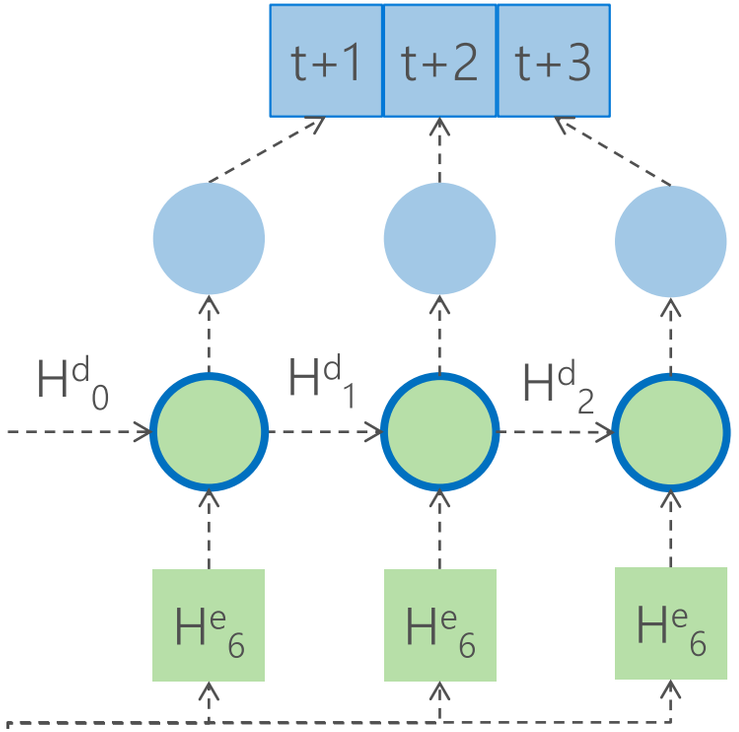
Algorithm	MAPE (Mean Absolute Percentage Error)
ARIMA	4.9
Feedforward Neural Network	4.27
RNN: vector output	3.67

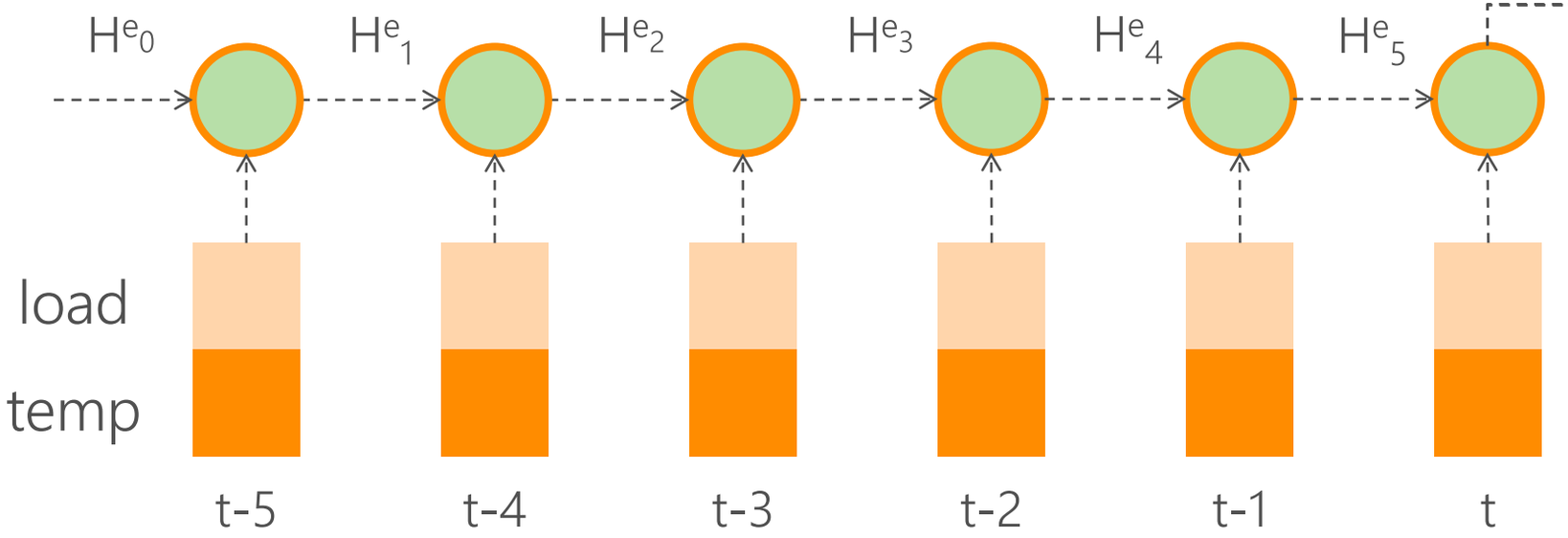
Simple encoder-decoder



= encoder RNN cell

= decoder RNN cell



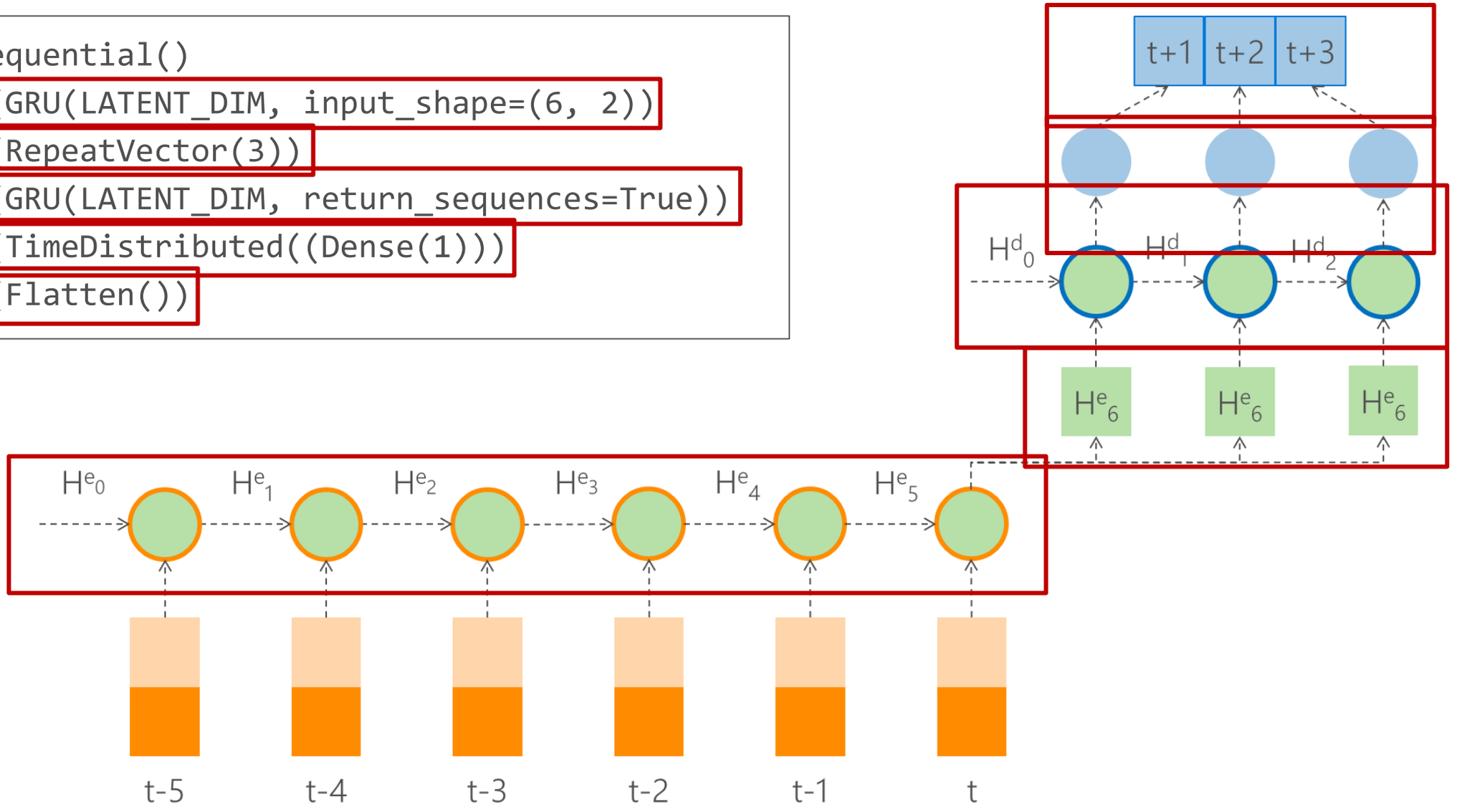


Multi-step forecasting using encoder-decoder

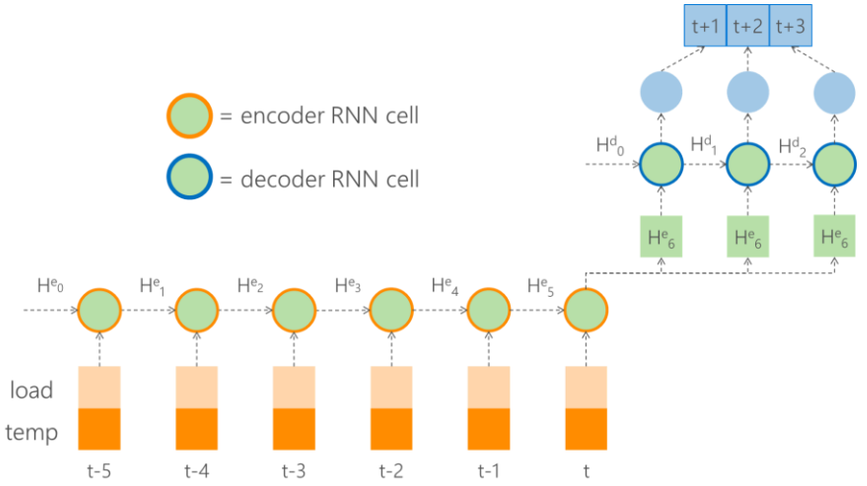
Open 5_multi_step_RNN_encoder_decoder_simple.ipynb

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

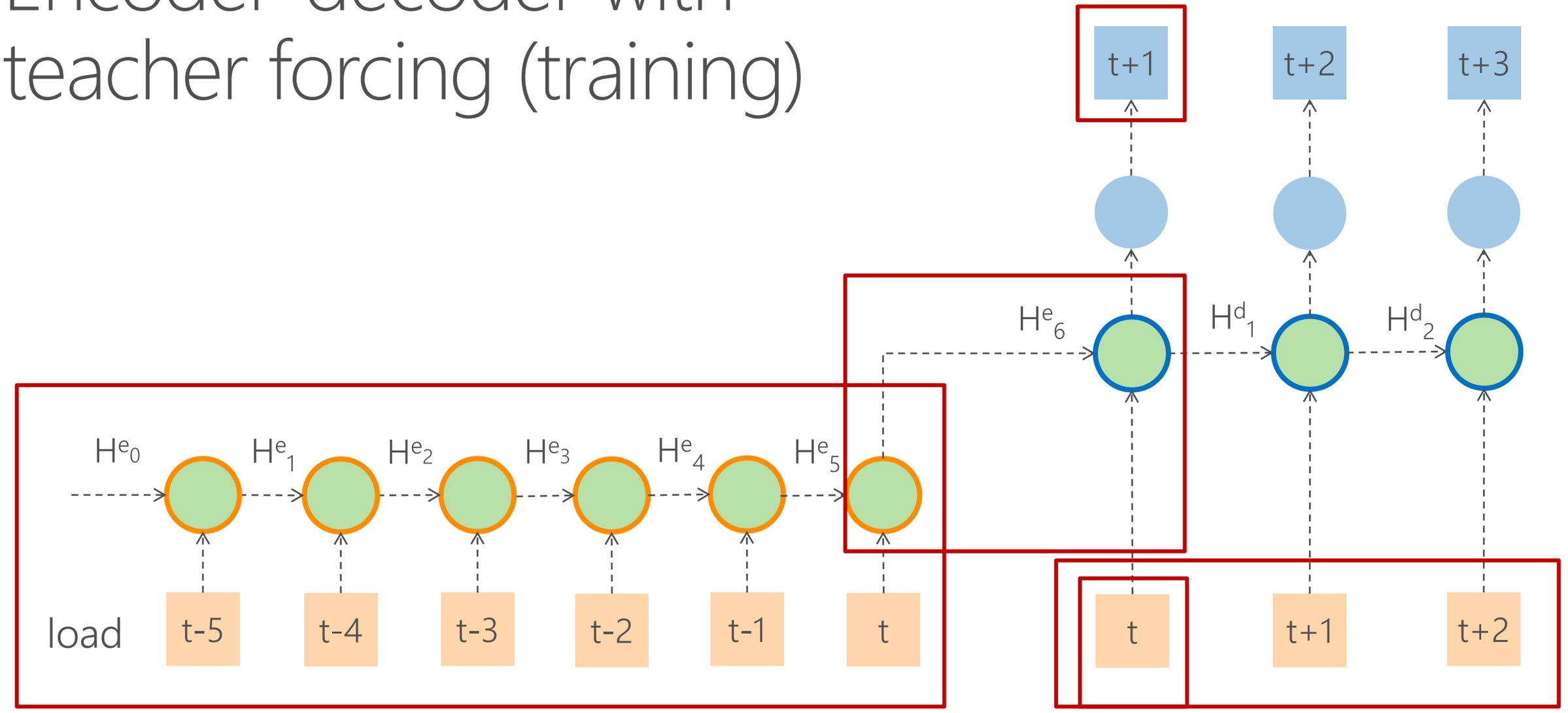


- Fairly simple to implement
- Tries to capture dependencies between forecasted time steps through decoder hidden state
- Slower to train with stacked RNN layers

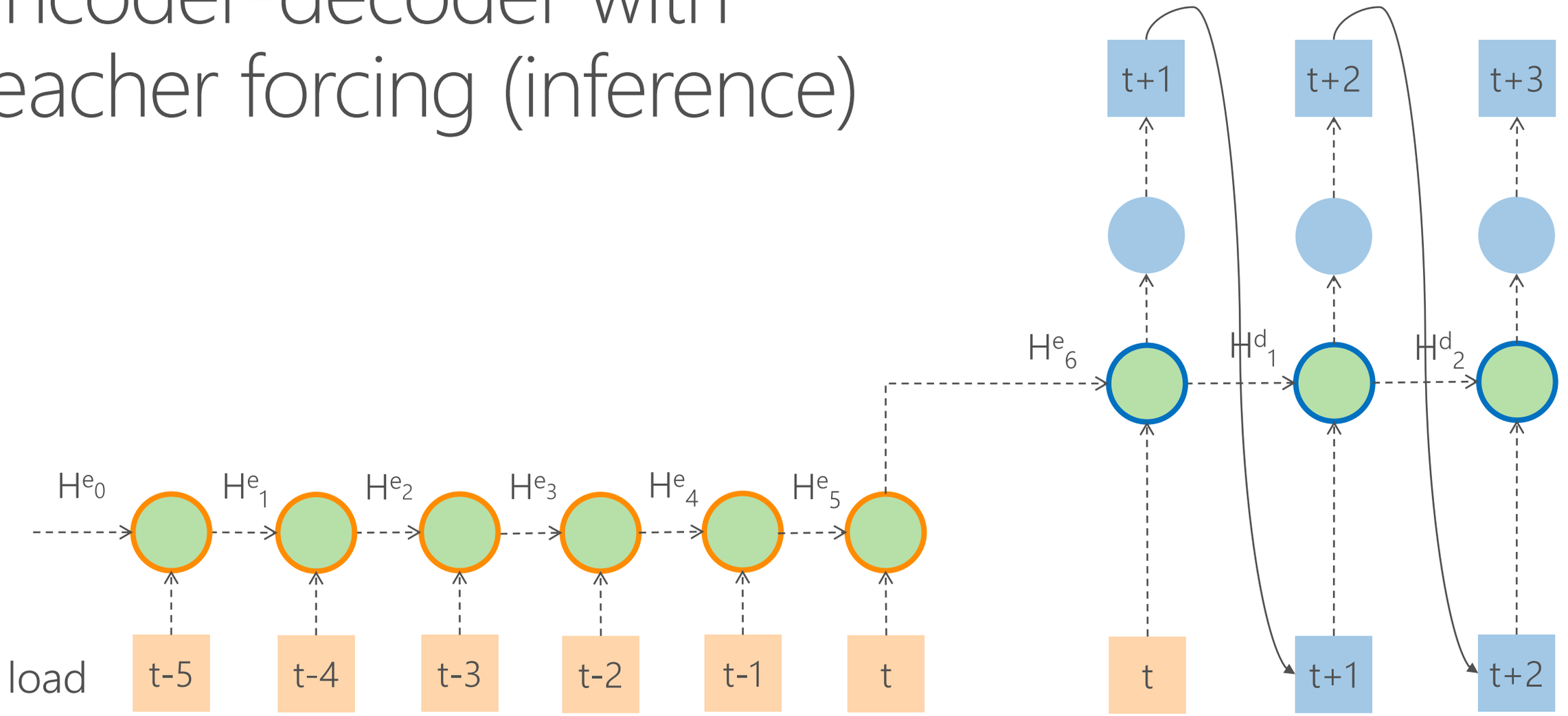
Performance comparison

Algorithm	MAPE (Mean Absolute Percentage Error)
ARIMA	4.9
Feedforward Neural Network	4.27
RNN: vector output	3.67
RNN: simple encoder-decoder	3.58

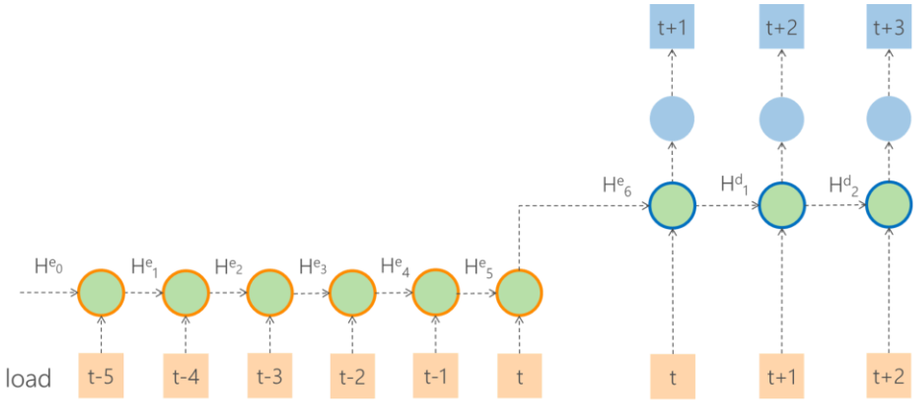
Encoder-decoder with teacher forcing (training)



Encoder-decoder with teacher forcing (inference)



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

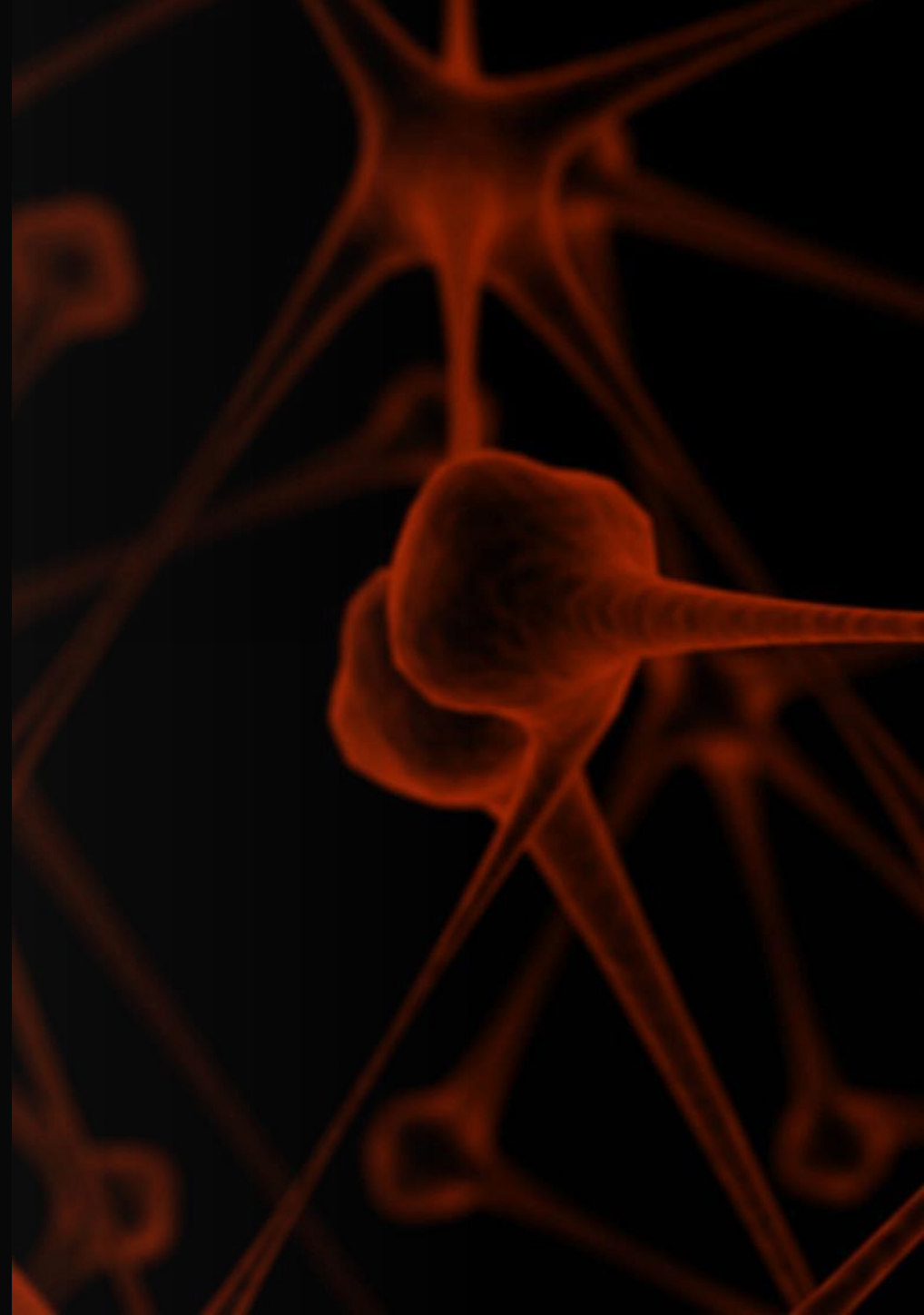
Performance comparison

Algorithm	MAPE (Mean Absolute Percentage Error)
ARIMA	4.9
Feedforward Neural Network	4.27
RNN: vector output	3.67
RNN: simple encoder-decoder	3.58
RNN: encoder-decoder with teacher forcing	4.57

Hands-on Quiz

Open Quiz_multi_step_RNN_encoder_decoder.ipynb

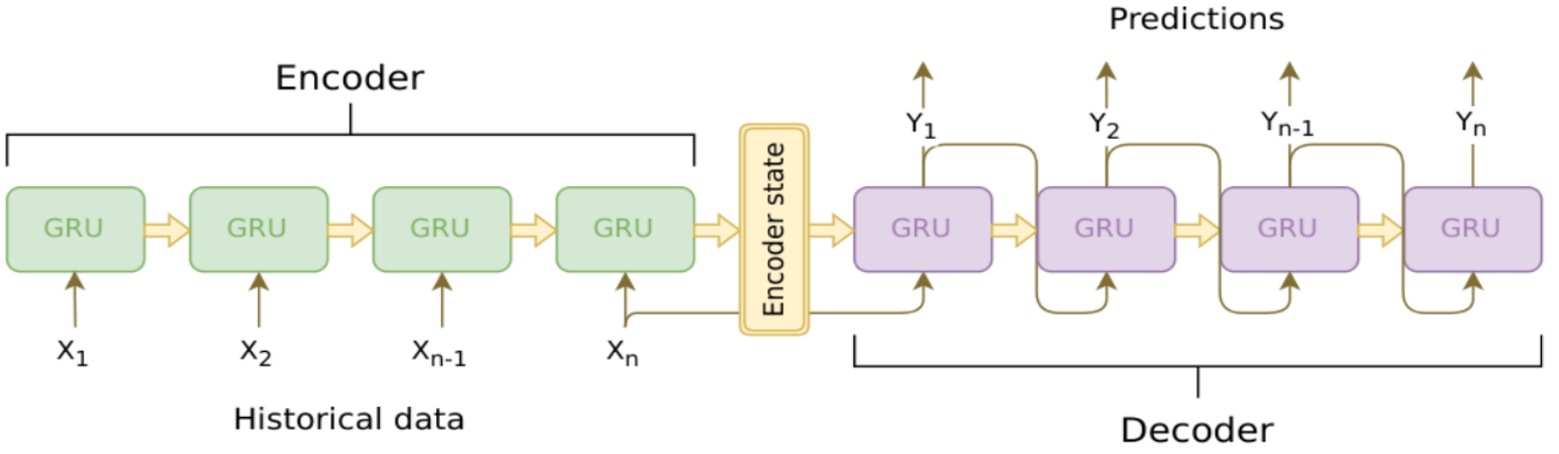
Successful RNN models



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:

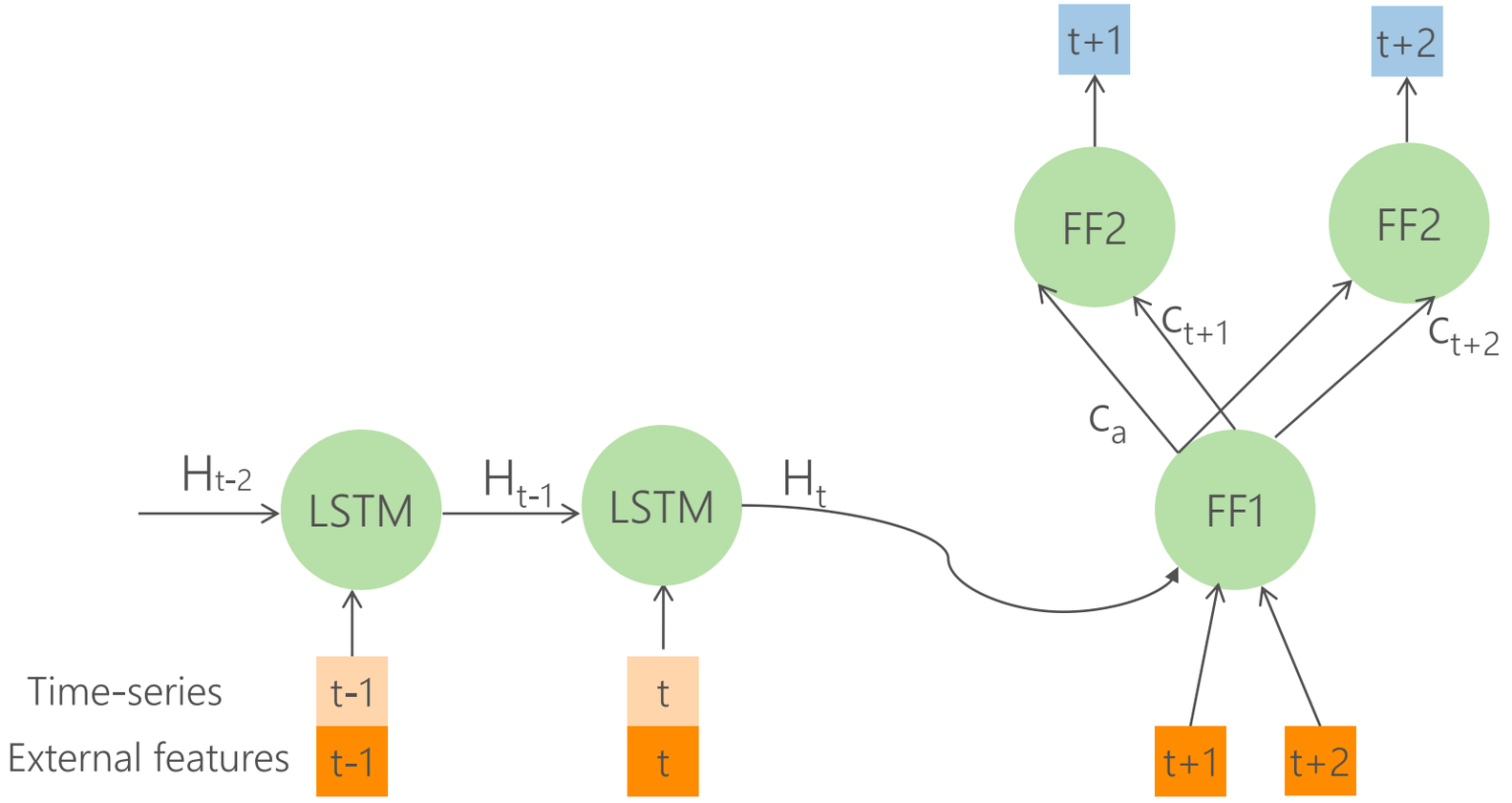
$$x_t \rightarrow (x_t, x_{t\text{-quarter}}, x_{t\text{-year}})$$

- 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

GEFCom 2014 competition

Energy load and price forecasting in New England

Best model (2017): LSTM encoder + feed-forward neural network arxiv.org/abs/1711.11053



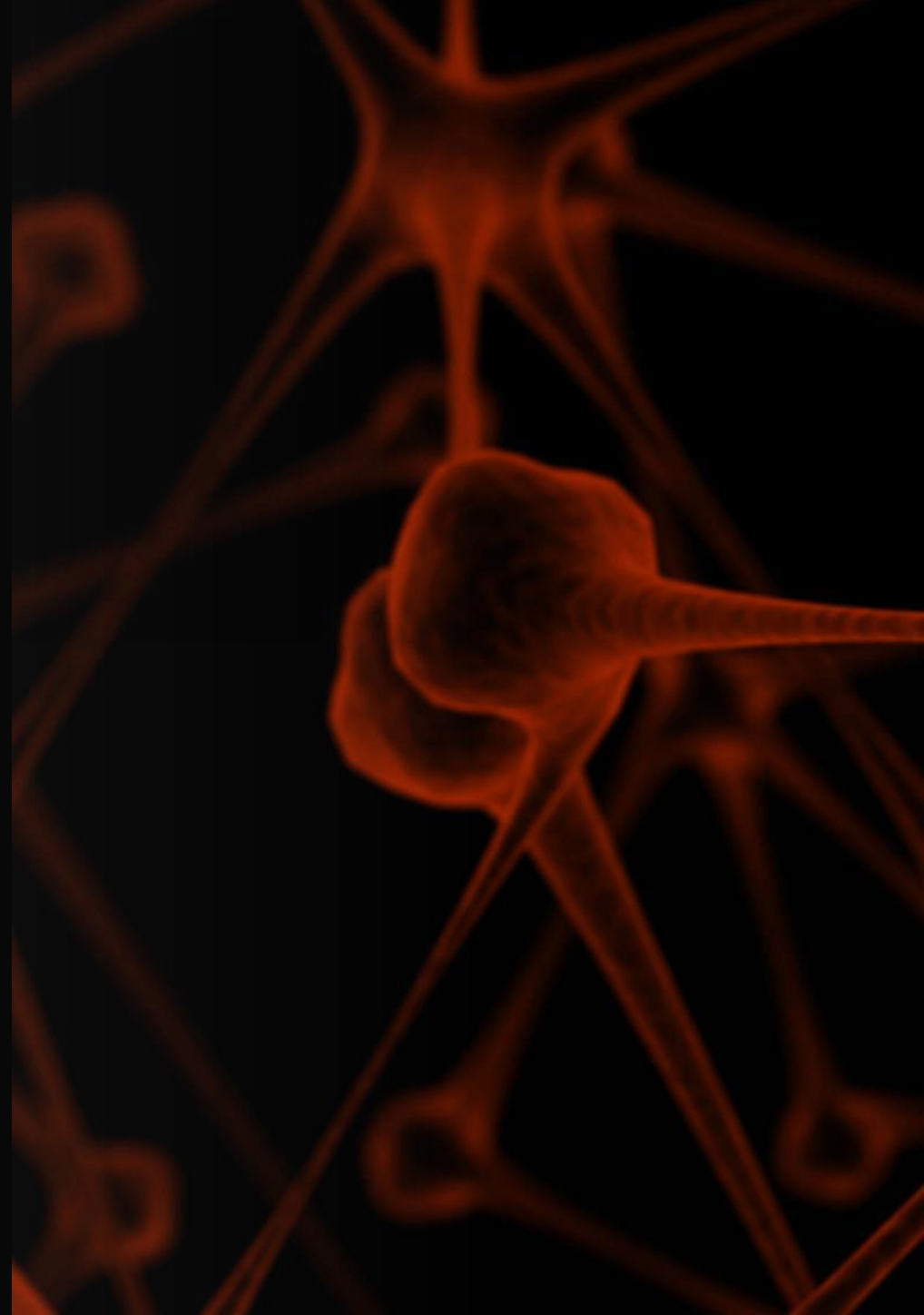
c_a - horizon-agnostic context

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

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 RNN models with Keras
  URL: https://aka.ms/rnntutorial

- 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

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

Key Takeaways

- RNNs are very powerful models, sometimes RNNs are the most accurate models.
- RNNs share many techniques/tricks with feed-forward neural networks
- Tuning hyperparameters and other tricks are important for creating an accurate model

Q & A