



# Time Series Management

Michele Linardi Ph.D.

[michele.linardi@orange.fr](mailto:michele.linardi@orange.fr)



# Syllabus

- Recap: Time Serie Forecasting and Deep Learning Fundamentals
- Introduction to convolutional neural networks (CNN)
- Implementation of the models with the Keras Library

# Time series data... quick recap

- A **univariate time series** is a sequence of measurements of the same variable collected over time. Most often, the measurements are made at regular time intervals.



<https://www.kaggle.com/code/anushkaml/walmart-time-series-sales-forecasting>

# Time Series Forecasting

- Predict Future (Values) 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
	2018-06-13	?
Future	2018-06-14	?
	2018-06-15	?

# Question to ask prior to forecast

- Can we forecast?
  - How well we understand the factor influencing the future?
  - How much data we have?\*
- How far in the future (horizon) we want to forecast?
- What technique, model should we use ?

# Why Deep Learning models? (1/2)

Deep learning model perform well and better than other methods in many scenarios.

Model	Features	RMSE	MAPE (%)	Run time (s)
FeedForward	Date features + Covariants	<b>10.73</b>	4.20	53.35
AutoARIMA	Baseline	11.00	<b>3.52</b>	20658.95
DeepAR	Date features	11.96	5.99	156.18
FeedForward	Baseline	12.01	3.88	78.47
SeasonalNaive	Baseline	12.47	3.86	<b>1.20</b>
FeedForward	Date features	13.93	3.95	9.37
DeepAR	Baseline	14.01	6.12	856.51
DeepAR	Date features + Covariants	17.79	6.04	781.09
TrivialIdentity	Baseline	18.20	5.09	1.30
NPTS	Baseline	82.45	19.75	25.20

<https://blog.dataiku.com/deep-learning-time-series-forecasting>

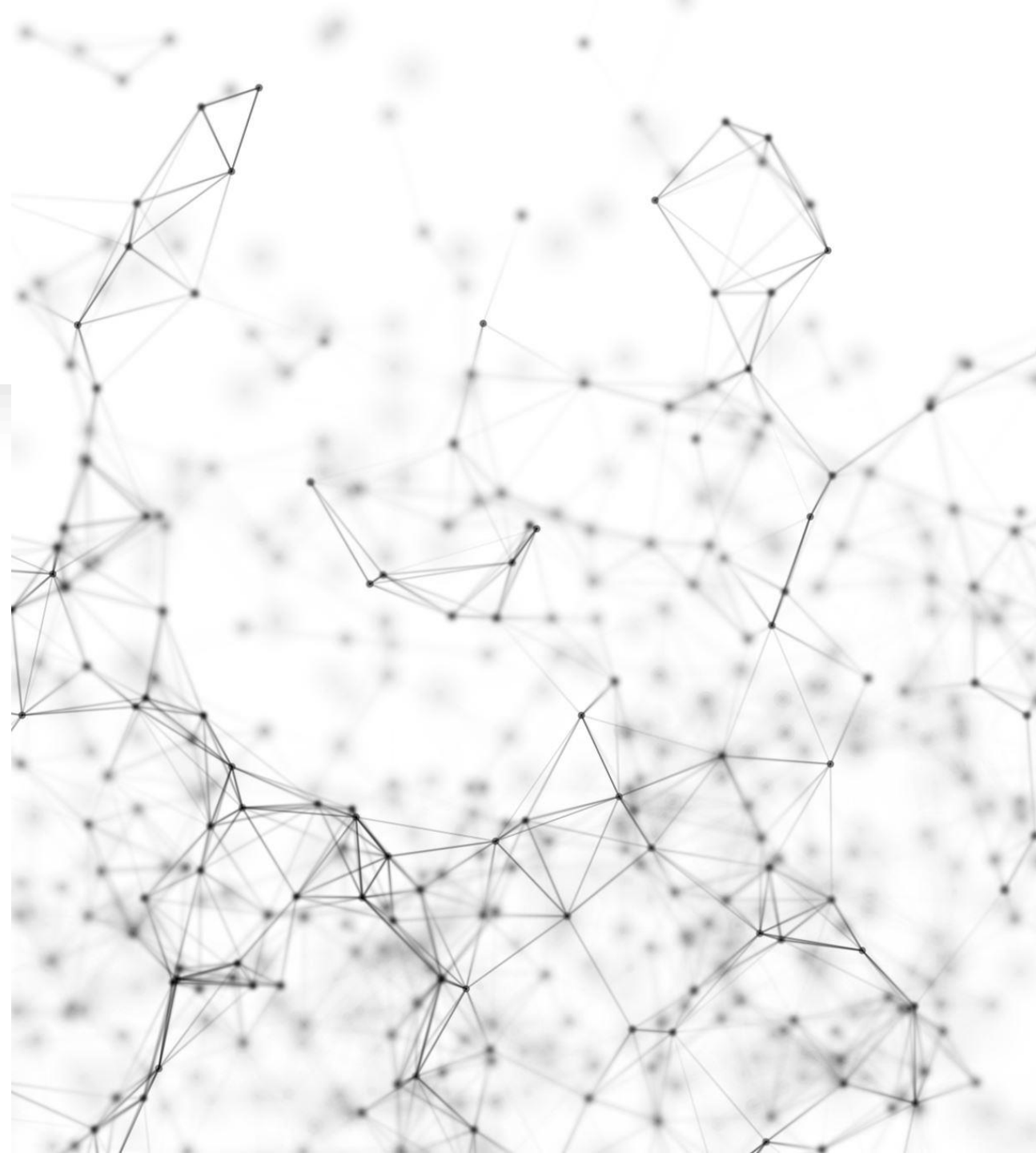
Z. Tang, P.A. Fishwik, [Feedforward Neural Nets as Models for Time Series Forecasting](#), November 1993

# Why Deep Learning models? (2/2)

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

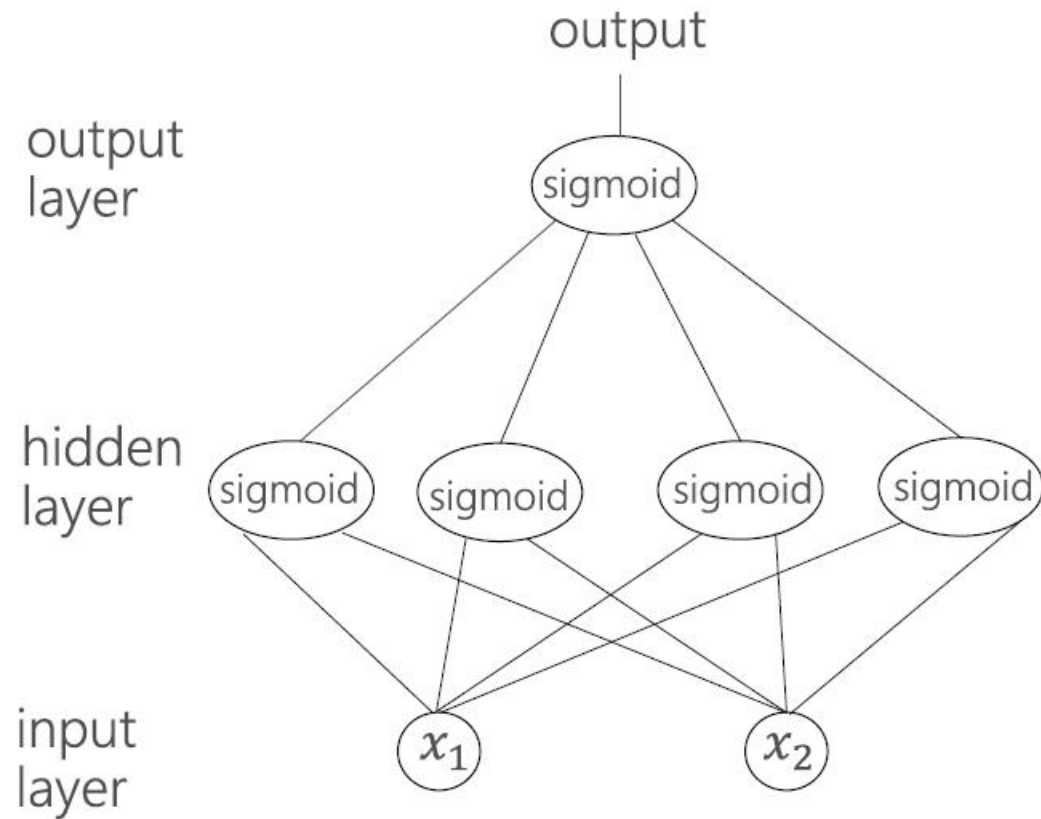
---

# DEEP LEARNING FUNDAMENTALS

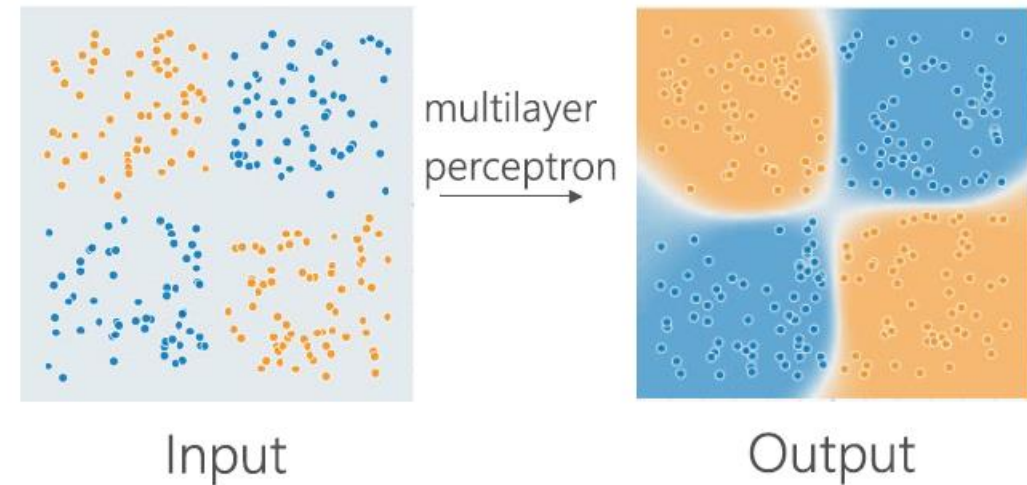




# Multilayer Perceptron



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

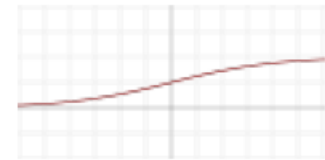


also known as Feed-Forward Neural Network and Deep Neural Network

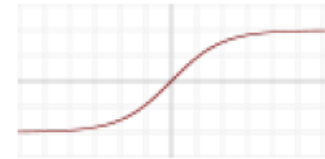
# Activation Function

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

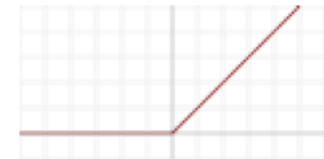
Sigmoid  $f(x') = \sigma(x') = \frac{1}{1+e^{-x'}}$



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

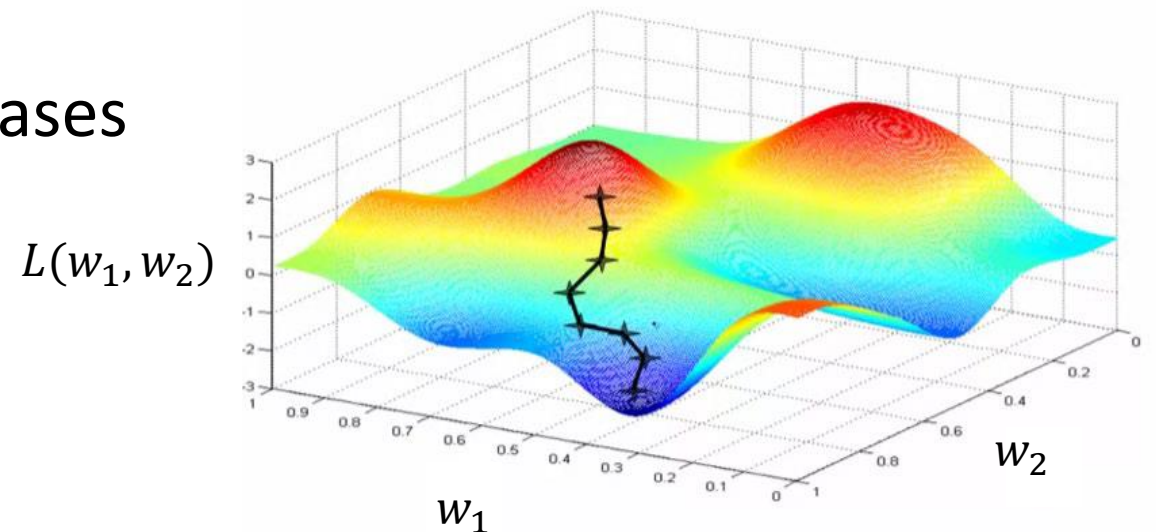


Rectifier linear unit (ReLU)  $f(x') = \begin{cases} 0 & \text{for } x' < 0 \\ x' & \text{for } x' \geq 0 \end{cases}$



# Training: overview

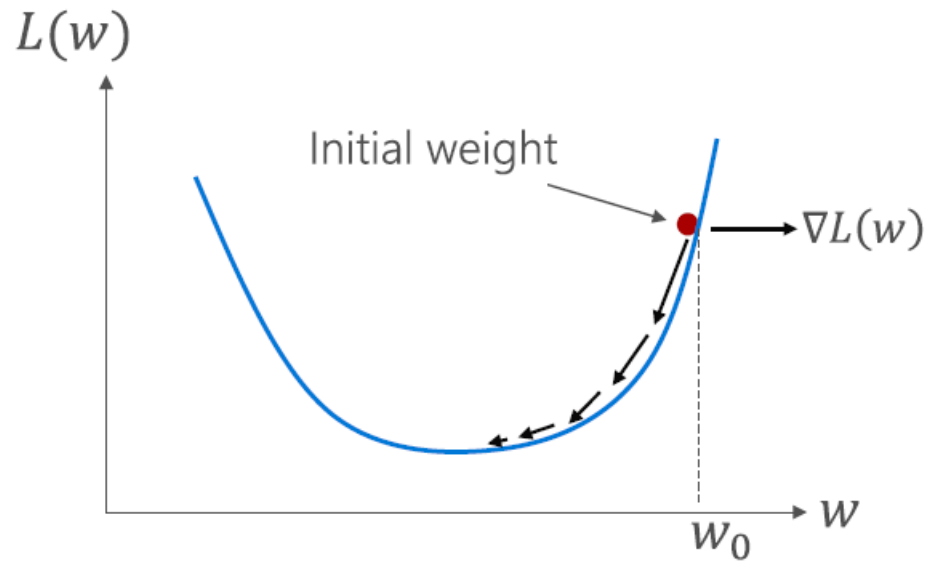
- Main Idea: Use training example to find proper weights and biases.
- Loss function  **$L(\text{Weights}, \text{biases})$**  measure the discrepancy between predicted and true values
- We want to optimize weight and biases to reduce the Loss.



# Optimization (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

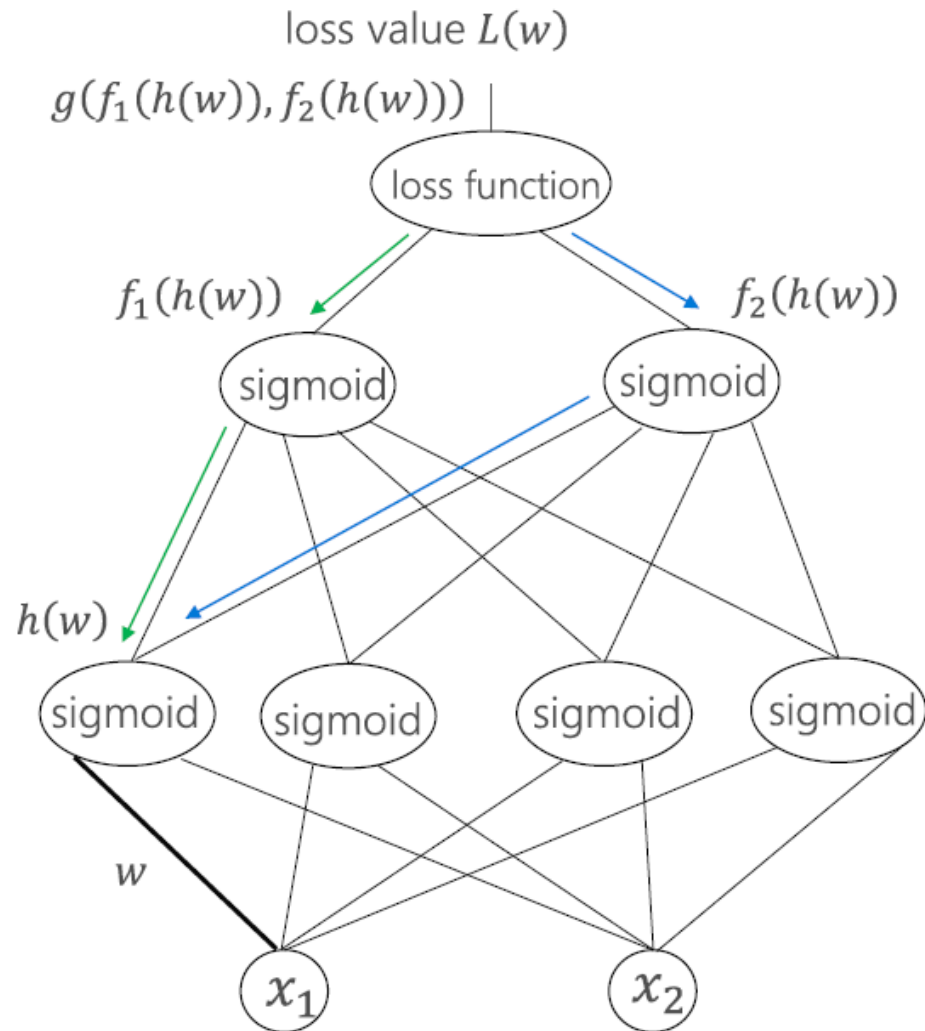
Initialization:  $w = w_0$

While stopping criterion not met:

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

learning rate

# Gradients Computation: Backpropagation



Computation of  $L'(w)$

Chain rule

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

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

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

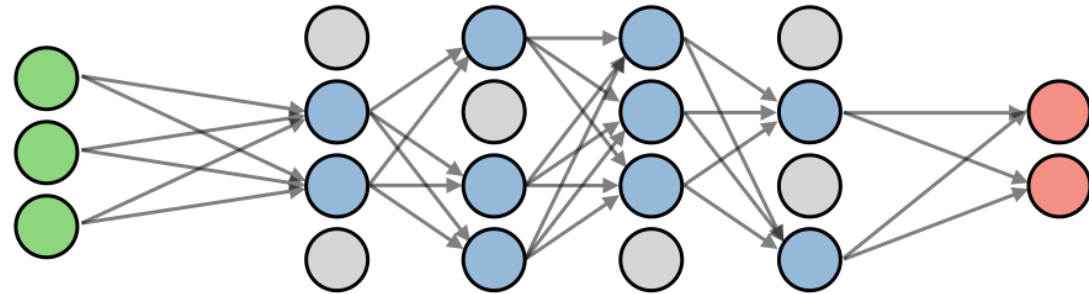
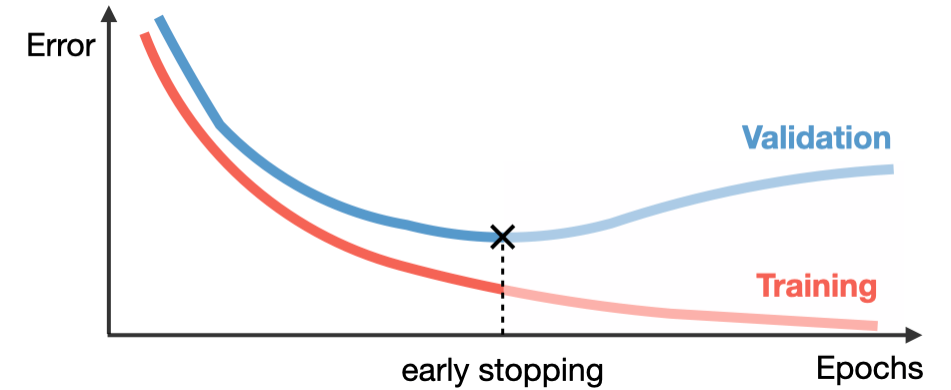
Total derivative

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

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

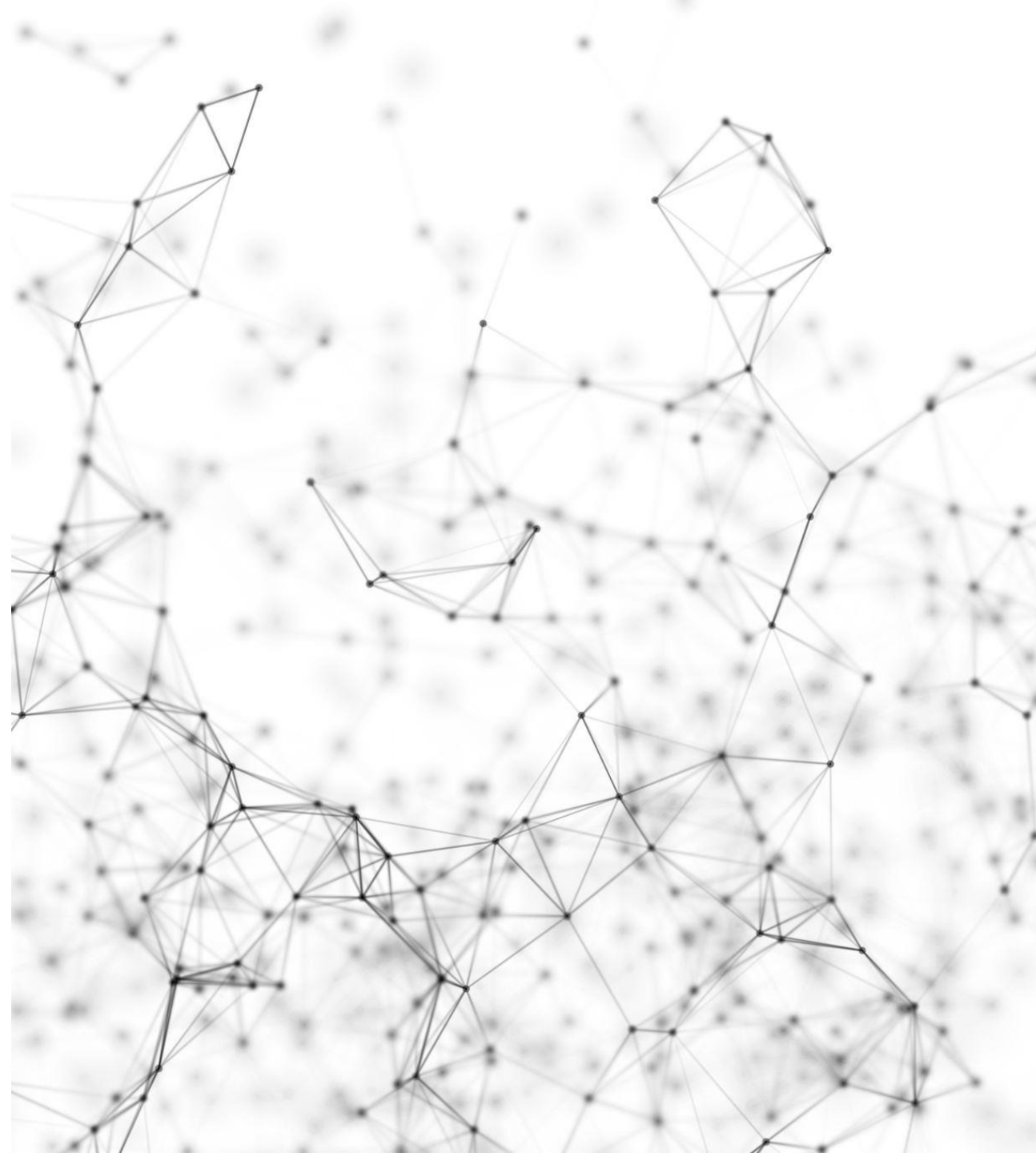
# Training improvement strategy

- **Early stopping**
- Tuning hyperparameters
- Initialization
- Weight regularization
- **Dropout**
- **Batch Normalization**
- Learning rate decay



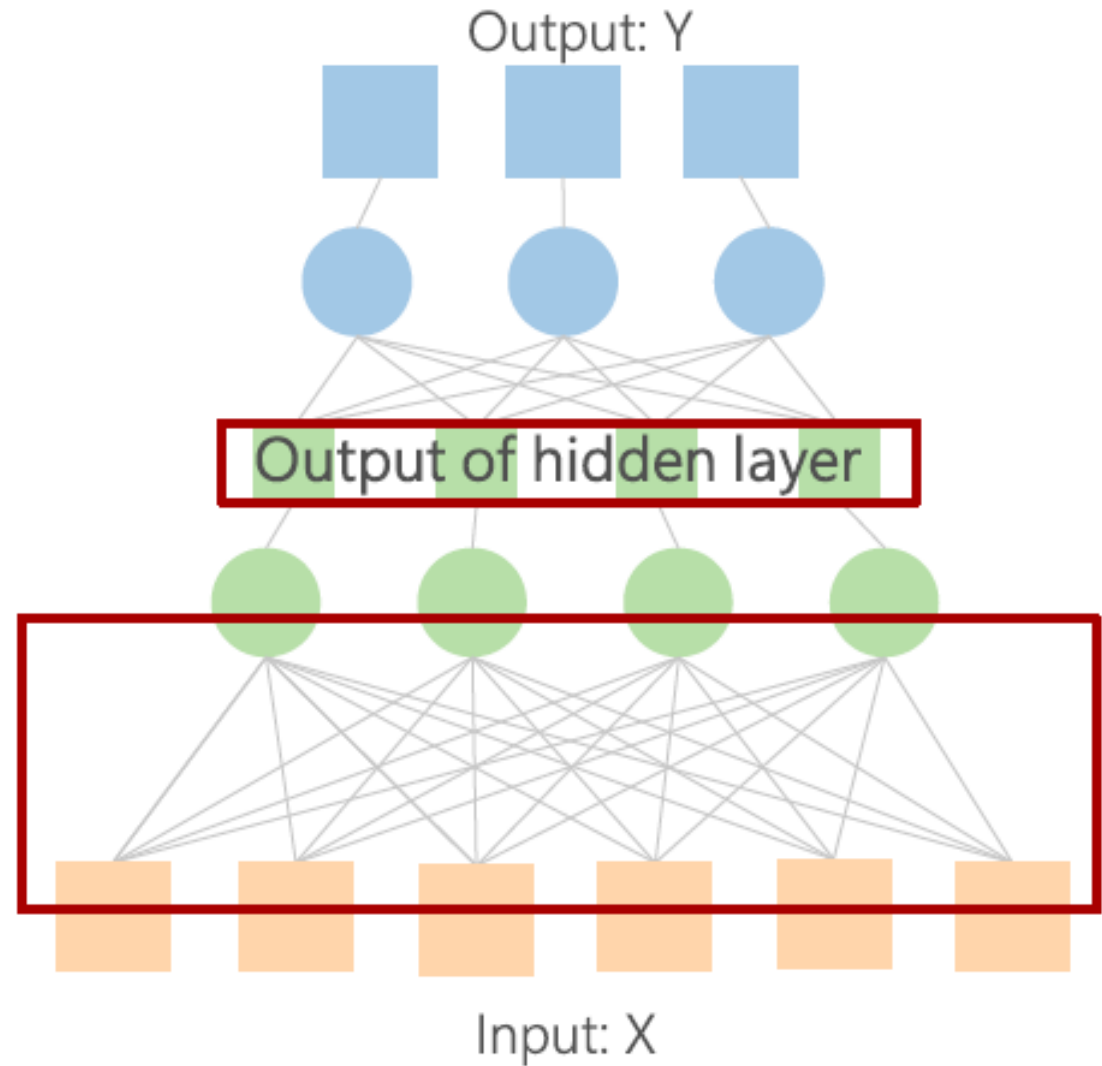
$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

# Convolutional neural networks (CNN) for Time series Forecasting



# Fully connected VS Convolutional Layer

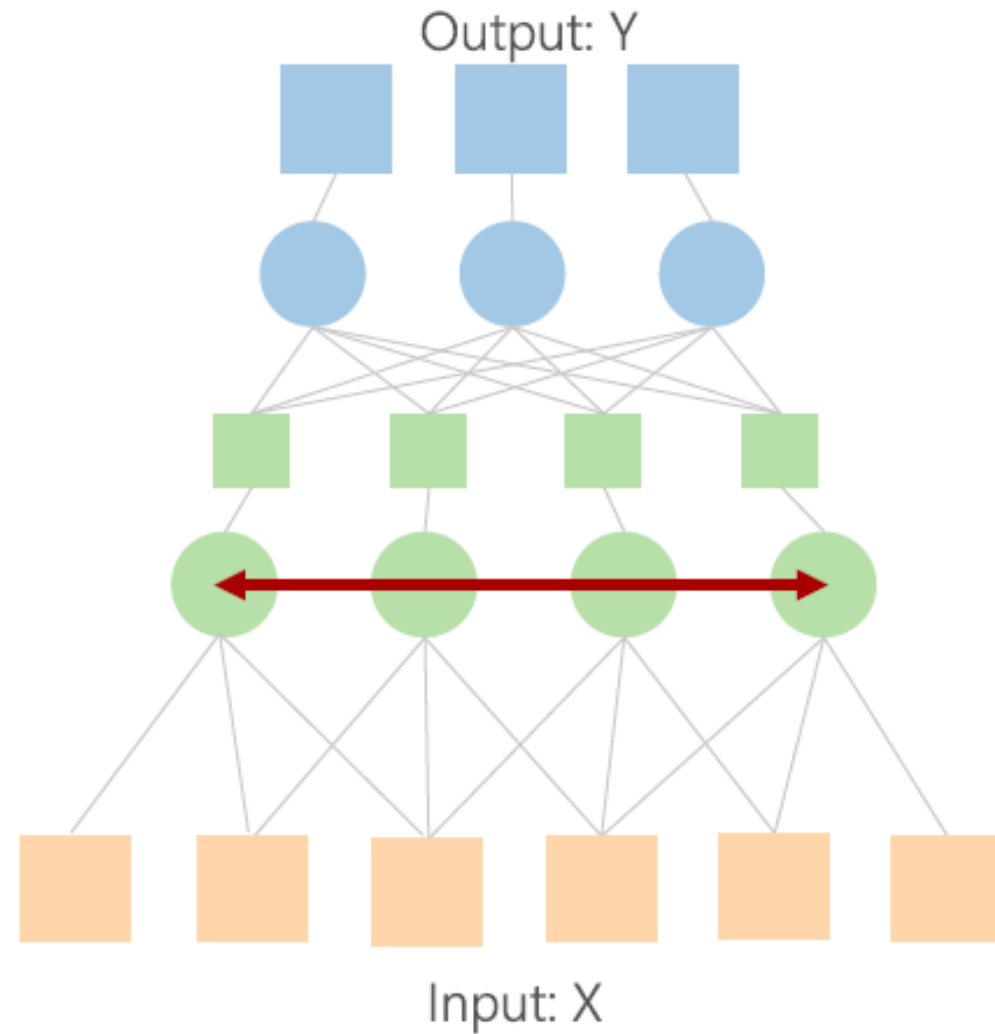
**Fully Connected:** Units in hidden layer are connected to every unit in previous layer.





# Fully connected VS Convolutional Layer

**Convolutional Layer:** Units in hidden operate on a field of the input (the weights are shared across the input)



# 1D Convolution

- Apply a 1D filter to all elements of an input time series.
- Result: SUM of element-wise product

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

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

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

# 1D Convolution

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

5	3	2	7	1	6
---	---	---	---	---	---

X

$w_1$	$w_2$	$w_3$
-------	-------	-------

=

3	4	1	1
---	---	---	---

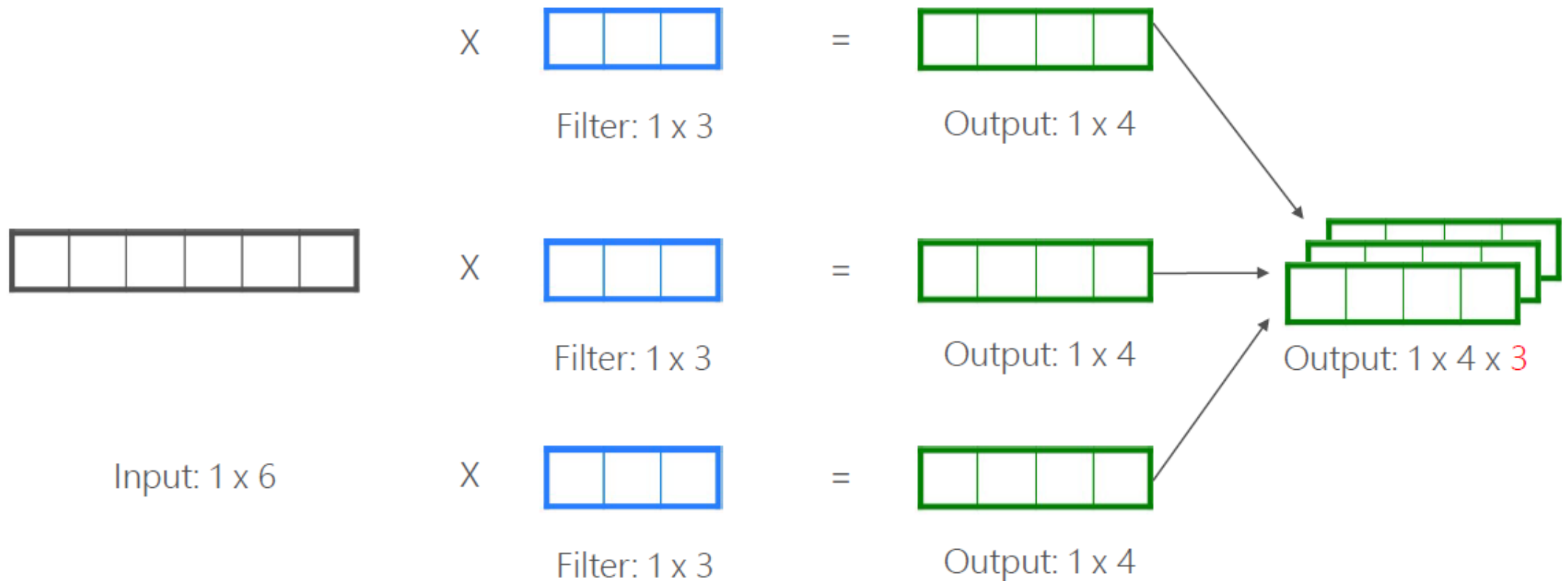
Input: 1 x 6

Filter: 1 x 3

Output: 1 x 4

# 1D Convolution

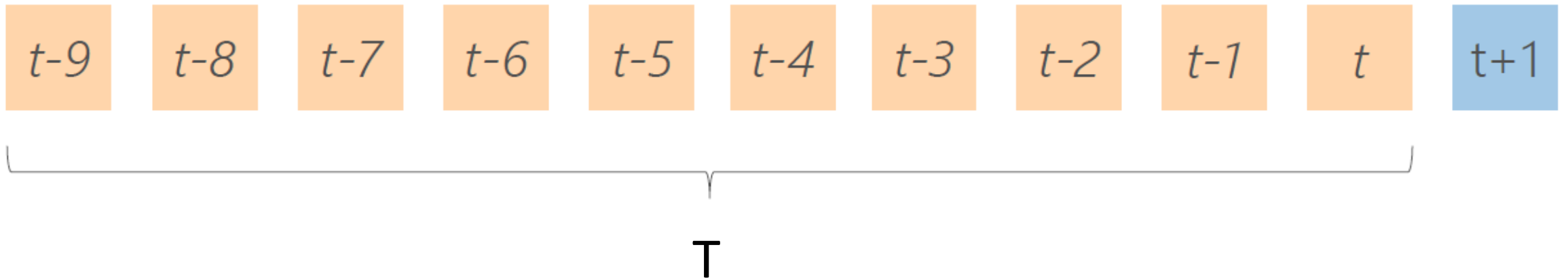
- Apply multiple filters to the input data to detect multiple features.



# Apply CNN to forecasting time series

At time  $t$  :

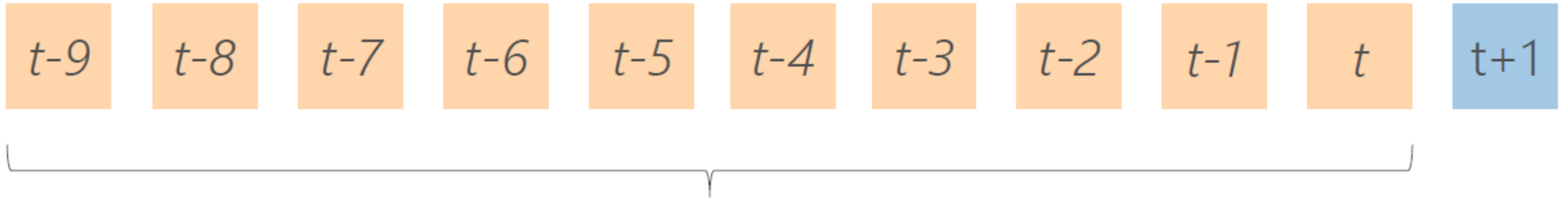
1. predict the value at time  $t+1$
2. Conditioned on the previous  $T$  values of the time series



# Apply CNN to forecasting time series

At time  $t$  :

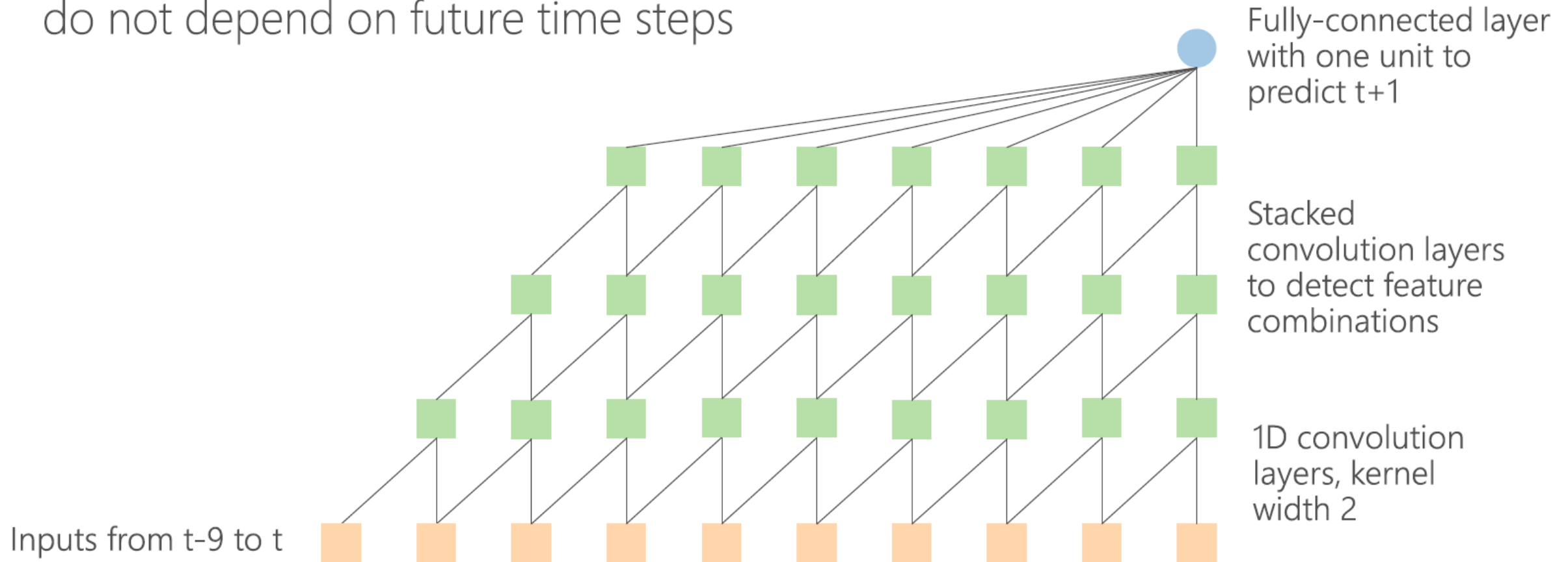
1. predict the value at time  $t+1$
2. Conditioned on the previous  $T$  values of the time series



$T = 10$  --- Empirically selected

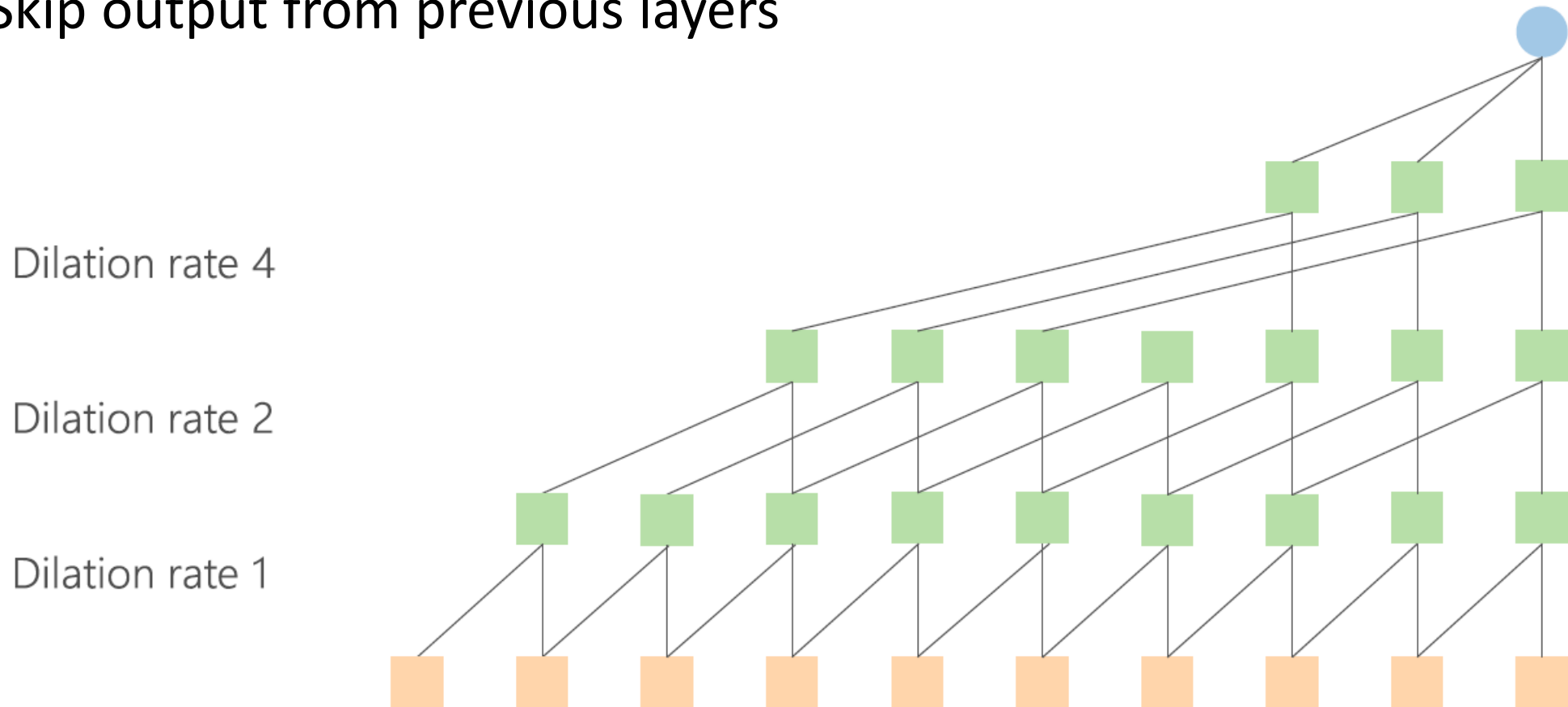
# Apply CNN to forecasting time series

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



# Dilated Convolutions

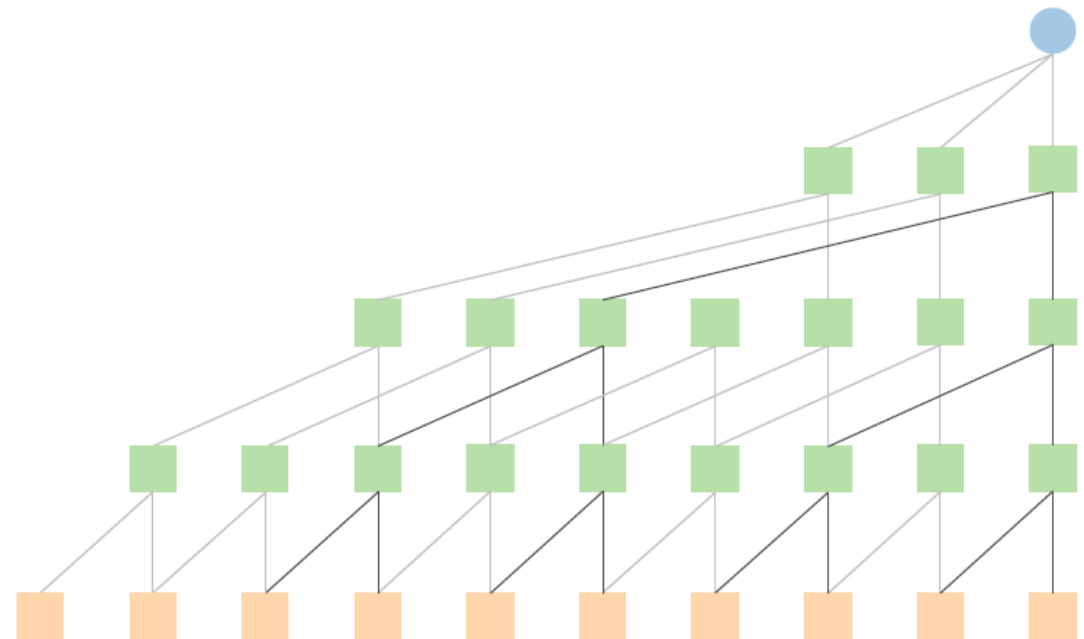
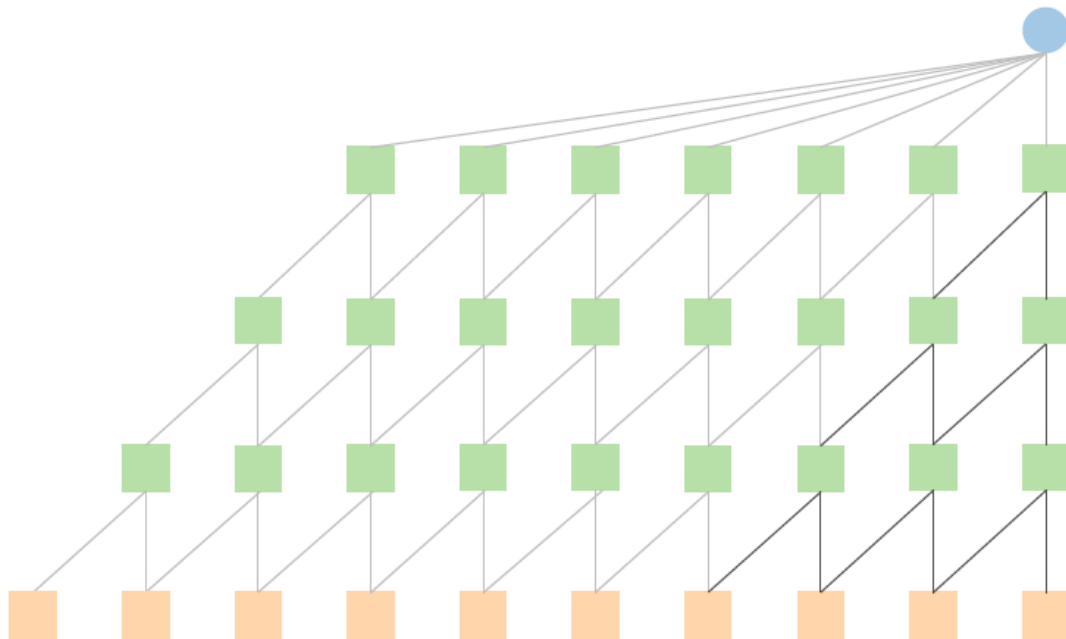
- Skip output from previous layers





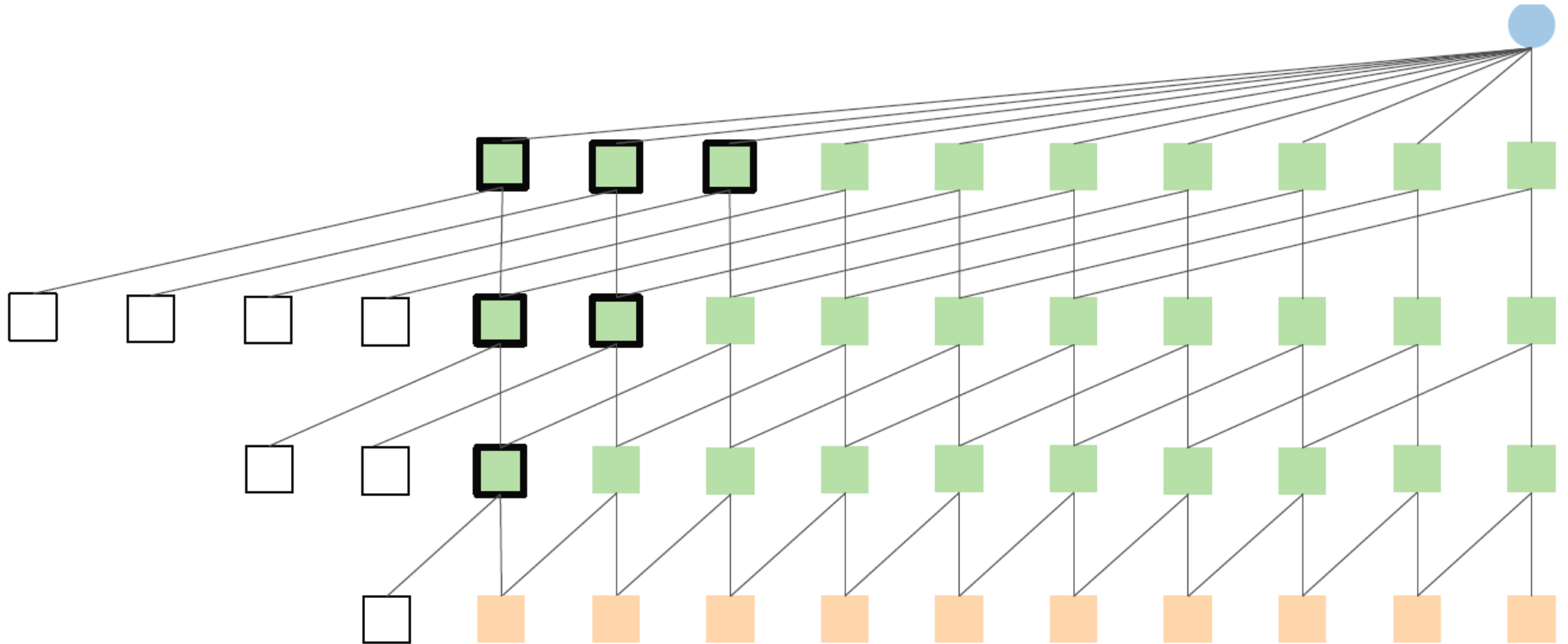
# Dilated Convolutions

- Normal convolution => more connections => more weights to train
- Reach information from more distant values in the time series (non contiguous filter)



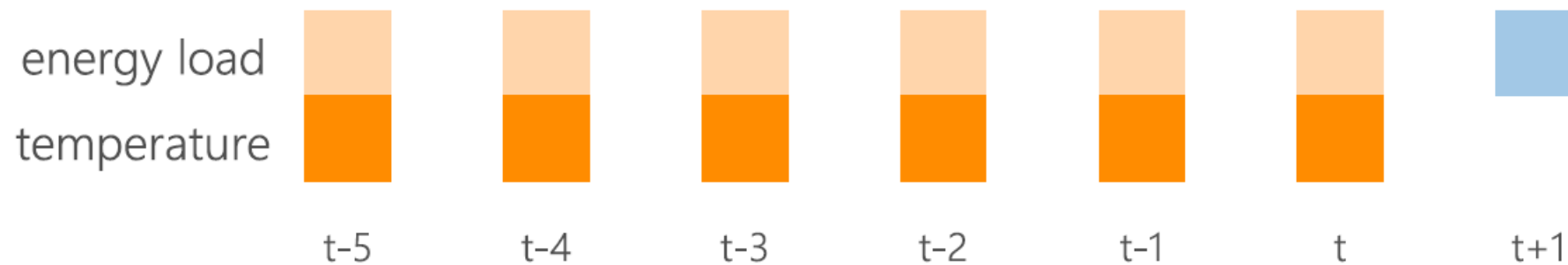
# Causal padding

- Padding is necessary to keep the right output dimensions



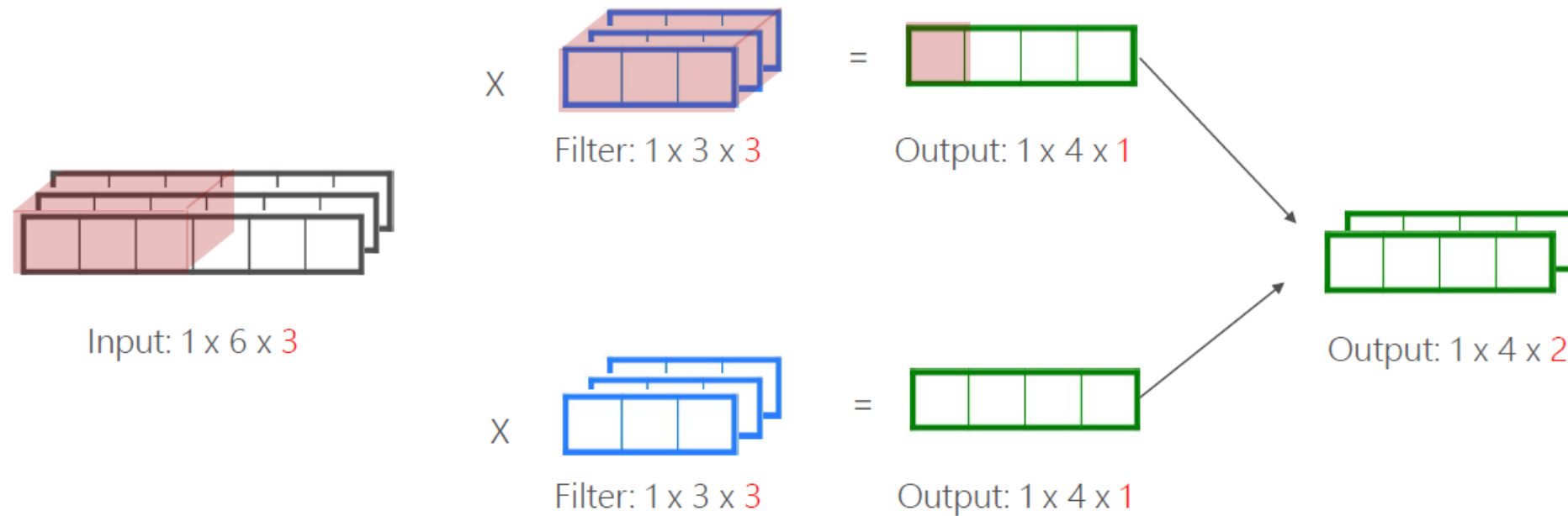
# Multivariate Time Series Case

- Time series with more than 1 variable evolving along time (aka covariates)
- Example

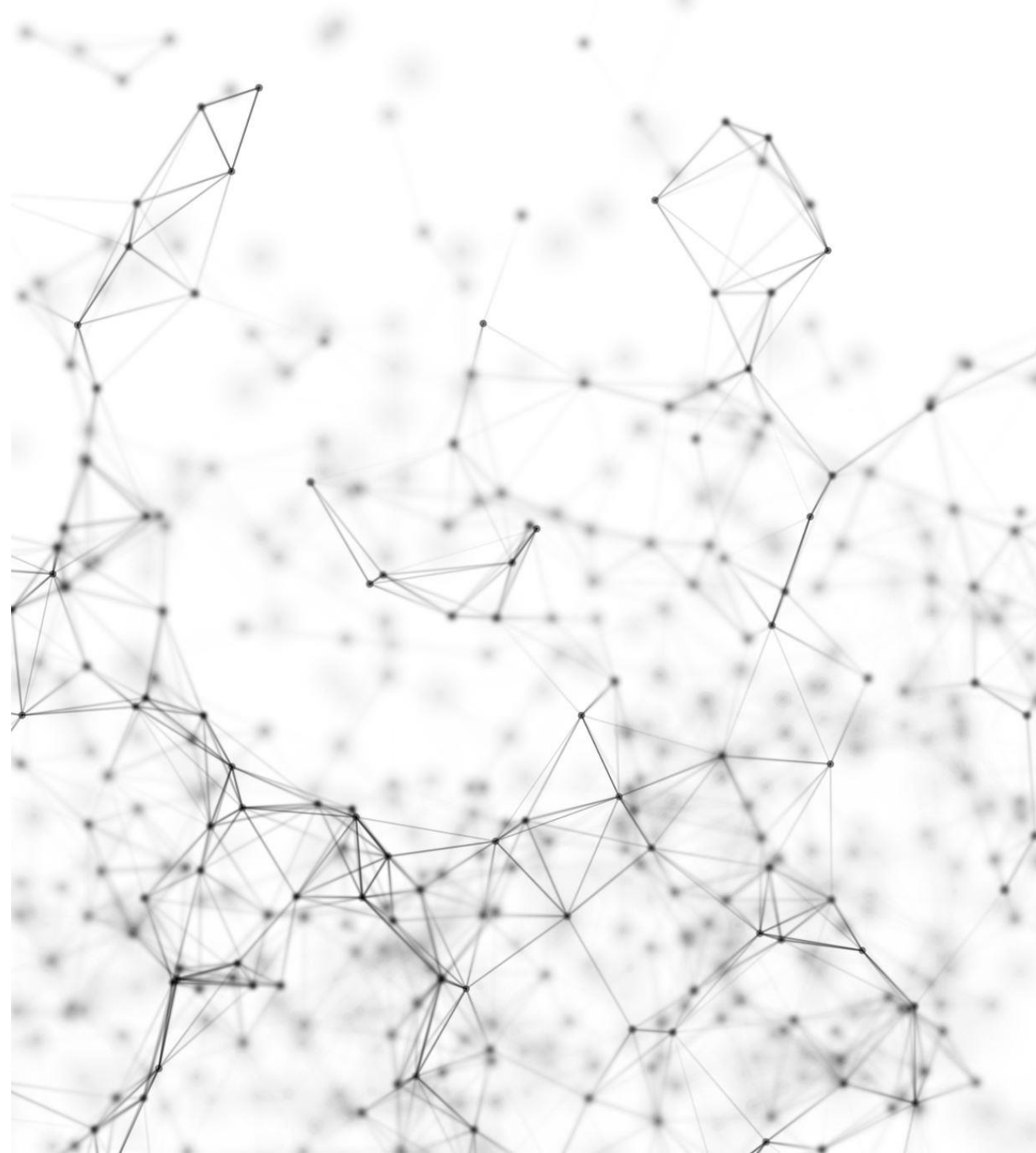


# Multivariate Time Series

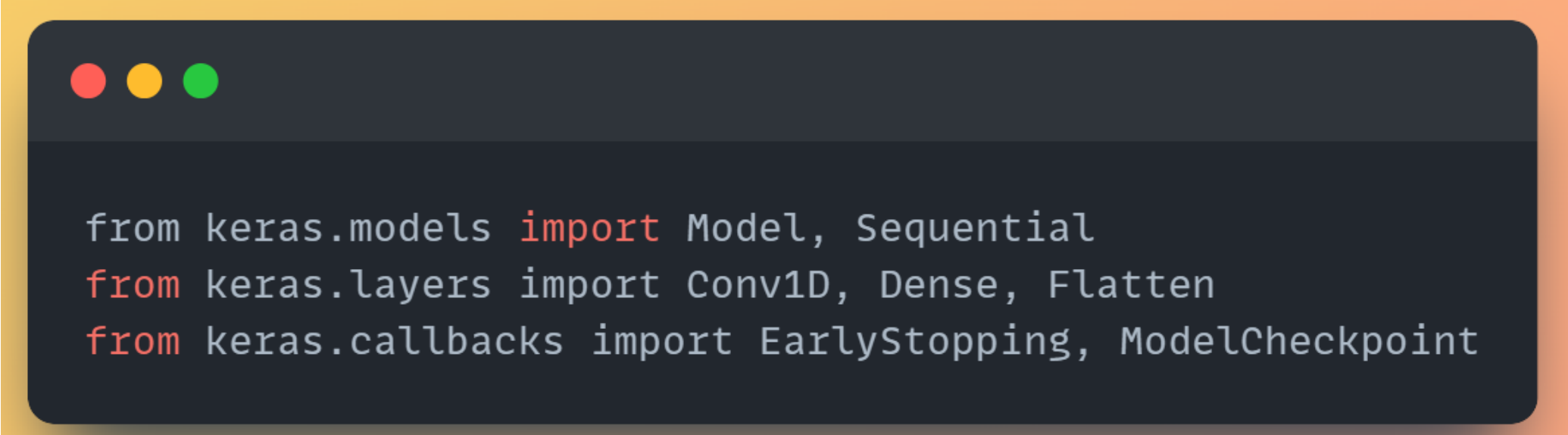
- Feature extraction with multiple filters (per dimension)



# Convolutional neural networks (CNN) with Keras




# Imports the KERAS modules (NN components)



```
from keras.models import Model, Sequential
from keras.layers import Conv1D, Dense, Flatten
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

# KERAS – 1D convolutional filter

TIME SERIES PAST VALUE WINDOW (T)




```
model = Sequential()
model.add(Conv1D(FILTERS_NUMBER, kernel_size=KERNEL_SIZE,
                 padding='causal', strides=1, activation='relu',
                 dilation_rate=1, input_shape=(T, 1)))
```

[https://keras.io/api/layers/convolution\\_layers/convolution1d/](https://keras.io/api/layers/convolution_layers/convolution1d/)

# KERAS – 1D convolutional filter

# TIME SERIES variable (1 = univariate case)



```
model = Sequential()
model.add(Conv1D(FILTERS_NUMBER, kernel_size=KERNEL_SIZE,
                 padding='causal', strides=1, activation='relu',
                 dilation_rate=1, input_shape=(T, 1)))
```

[https://keras.io/api/layers/convolution\\_layers/convolution1d/](https://keras.io/api/layers/convolution_layers/convolution1d/)



Do not forget...flatten + dense layer for the final prediction



```
model.add(Flatten())  
model.add(Dense(HORIZON, activation='linear'))
```



Number of predictions

[https://keras.io/api/layers/reshaping\\_layers/flatten/](https://keras.io/api/layers/reshaping_layers/flatten/)

[https://keras.io/api/layers/core\\_layers/dense/](https://keras.io/api/layers/core_layers/dense/)

# Early stopping + best model save + training

```
model.compile(optimizer='Adam', loss='mse')

earlystop = EarlyStopping(monitor='val_loss', min_delta=0, patience=5)

best_val = ModelCheckpoint('model_{epoch:02d}.keras', save_best_only=True, mode='min')

history = model.fit(X_train,
                    y_train,
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS,
                    validation_data=(X_valid, y_valid),
                    callbacks=[earlystop, best_val],
                    verbose=1)
```

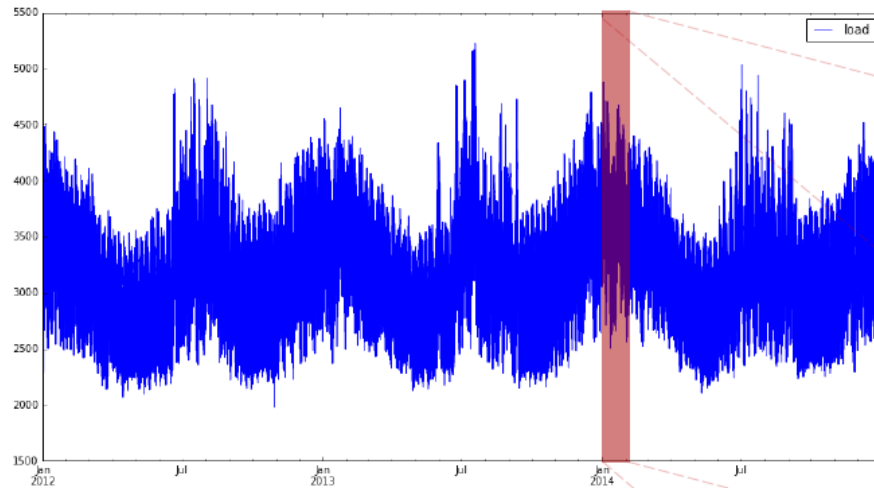
[https://keras.io/api/models/model\\_training\\_apis/](https://keras.io/api/models/model_training_apis/)  
[https://keras.io/api/callbacks/early\\_stopping/](https://keras.io/api/callbacks/early_stopping/)

A black and white photograph of a wooden floor with white chalk markings. The markings include the letters 'FF' and the number '0'. The text 'Presentation du TD' is overlaid in the center.

# Presentation du TD

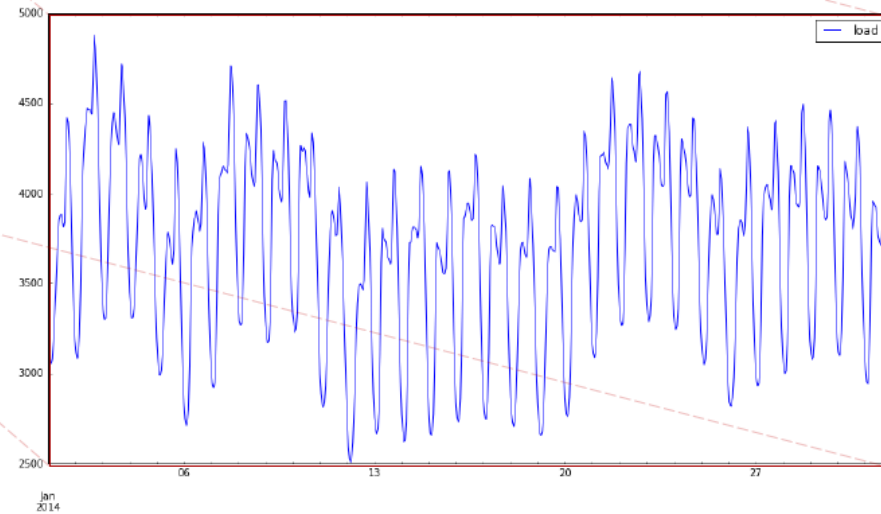
# Data

## 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.



# Notebook

- Open the file (Python Notebook):

**TD\_CNN\_dilated\_TS\_FORECASTING.ipynb**

- Instruction are contained in the notebook



# References

- <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks>
- **DeepLearningForTimeSeriesForecasting (Microsoft)**  
<https://github.com/Azure/DeepLearningForTimeSeriesForecasting>
- **Ben Auffarth** Machine Learning for Time-Series with Python  
<https://github.com/PacktPublishing/Machine-Learning-for-Time-Series-with-Python/tree/main/chapter10>
- Playground to visualize and test Deep Networks : <https://playground.tensorflow.org>
- Visualization CNN Filter and hyperparameters : <https://ezyang.github.io/convolution-visualizer/>

