

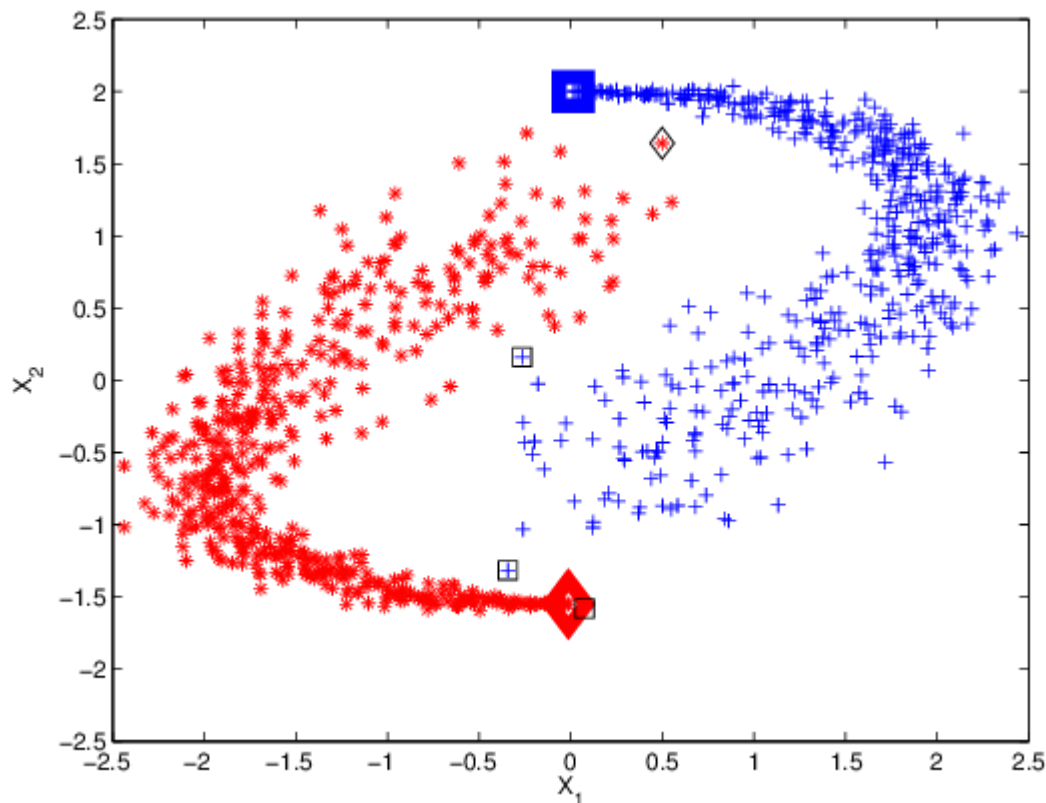
The time dimension: from linear to deep predictors

«The future is just a step ahead»



The iid assumption

- In most ML tasks we assume data to be i.i.d.
 - Independent (no time dependency among consecutive samples)
 - Identical pdf
- We weaken –at least– the independent assumption



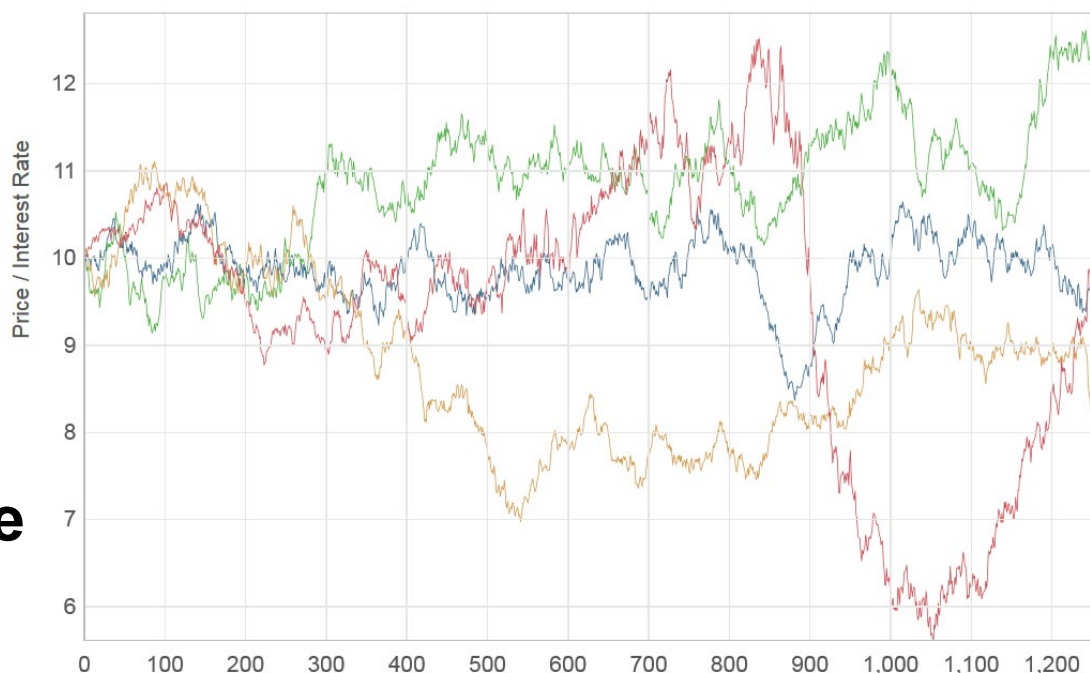
Time series

- A discrete **time series process** X is a discrete time stochastic process, i.e., a sequence of random variables (rv):

$$X = \{X_t \in \mathbb{R}^d, \quad t = 0, 1, 2, \dots\}$$

- An observed time series is a realization of the stochastic process

If $d=1$ the time series is **univariate**, **multivariate** otherwise



Univariate Time series

- Given the stochastic process:

$$X = \{X_t \in \mathbb{R}^d, \quad t = 0, 1, 2, \dots\}$$

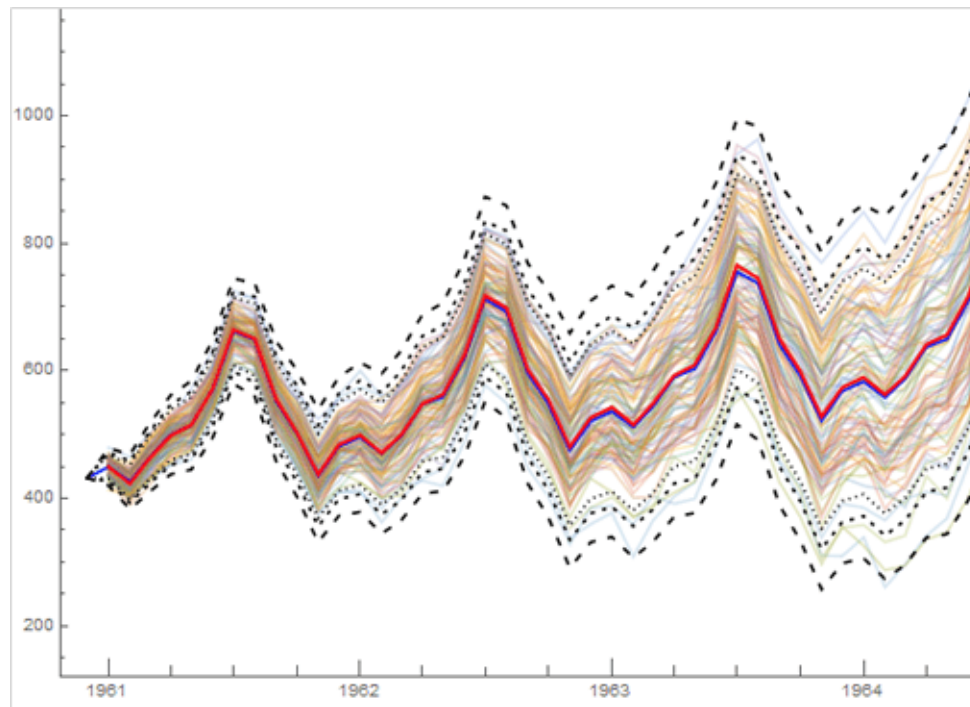
- Its mean function is defined as

$$\mu_X(t) = \mathbb{E}[X_t] \quad t \in T$$

- Its covariance function is:

$$\gamma_X(r, s) = \text{Cov}(X_r, X_s) \quad r, s \in T$$

(when time $r=s$ we have the variance at time r)



Stationarity and time invariance

- **Stationarity**

- We say that a data generating process is stationary when generated data are realizations of a random variable whose distribution does not change with time

- **Weak stationarity**

- Only the first two moments of the random variable do not change with time

Stationarity and time invariance

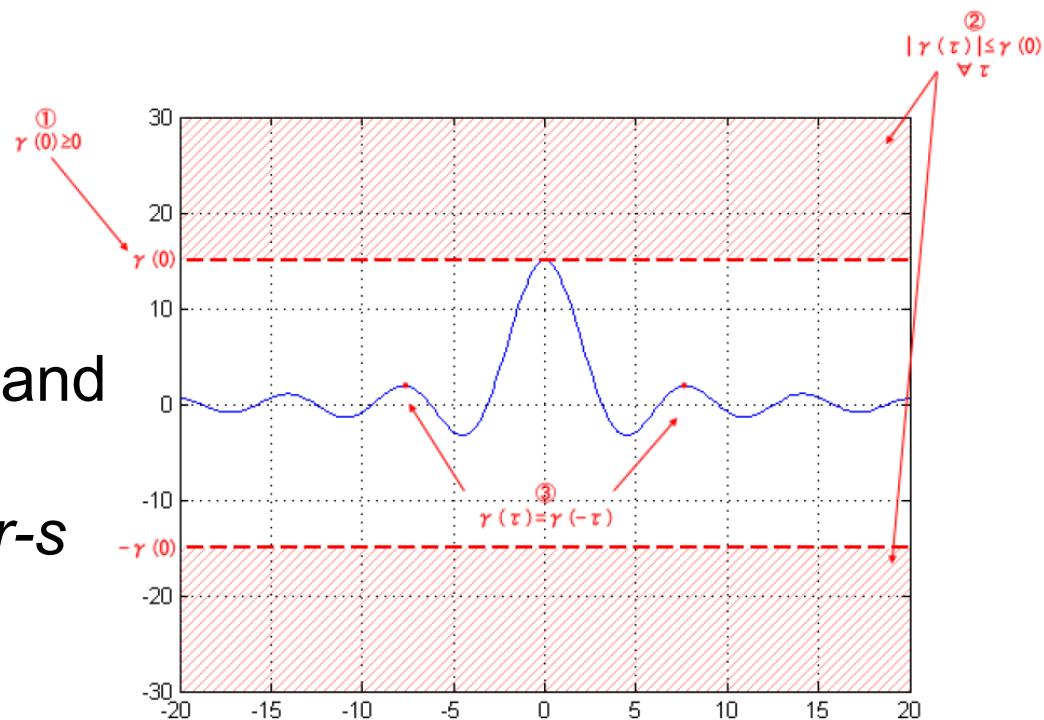
■ Time invariance (signal processing community)

- We say that a process is time invariant when its outputs do not explicitly depend on time

$$y(t) = a_1(e^{t_0-t})y(t-1) + a_2y(t-2) + \eta, \eta = \bar{N}(0, \sigma^2)$$

■ Stationarity (system identification community)

- A stochastic process is weakly stationary if its expectation is constant and the covariance matrix depends on the time $\tau=r-s$ difference only



White Noise

- A **white noise** stochastic process is a sequence of independent random variables with zero mean and σ^2 variance

$$\mu_X(t) = \mathbb{E}[X_t] = 0 \quad \forall t \in T$$

$$\gamma_X(\tau) = \begin{cases} \sigma^2 & \text{if } \tau = 0 \\ 0 & \text{otherwise} \end{cases}$$

- Unrelated at different time steps, as such unpredictable

Linear time invariant models (LTI)

- Hierarchical predictive LTI models for time series
 - Autoregressive (AR)
 - Moving Average (MA)
- a combination of the above:
 - Autoregressive Moving Average (ARMA)
 - Autoregressive Integrated Moving Average (ARIMA)

AR(p): Autoregressive predictive models

- A stationary time series is said to be autoregressive of order p if:

$$y(t) = \phi_1 x(t-1) + \phi_2 x(t-2) + \cdots + \phi_p x(t-p) + \eta(t)$$

$$y(t) = x(t) \qquad \eta(t) \sim WN(0, \sigma^2)$$

- Predictive model family

$$\hat{y}(t) = \sum_{i=1}^p \phi_i g_i(t)$$

- Where:

- $\Theta = (\phi_1, \dots, \phi_p)$ is the vector of parameters
- $g_i(t) = x(t-i)$ are the components of the regressor vector

AR(p): Autoregressive predictive models

Consider the delay operator

$$x(t-1) = z^{-1}x(t)$$

And apply it to

$$y(t) = \phi_1 x(t-1) + \phi_2 x(t-2) + \cdots + \phi_p x(t-p) + \eta(t)$$

$$y(t) = x(t)$$

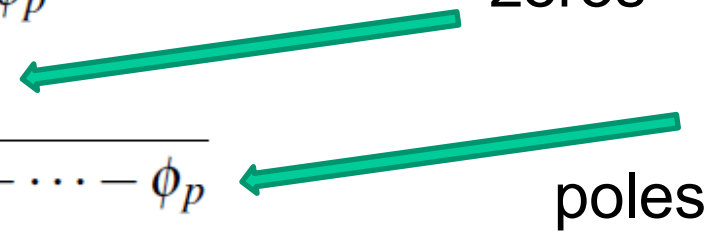
after manipulations

$$x(t) = \phi_1 z^{-1}x(t) + \cdots + \phi_p z^{-p}x(t) + \eta(t)$$

$$x(t) = (\phi_1 z^{-1} + \cdots + \phi_p z^{-p})x(t) + \eta(t)$$

AR(p): Autoregressive predictive models

From which we derive the *transfer function*

$$x(t) = \frac{z^p}{z^p - \phi_1 z^{p-1} - \dots - \phi_p} \eta(t)$$
$$H(z) = \frac{1}{A(z)} = \frac{z^p}{z^p - \phi_1 z^{p-1} - \dots - \phi_p}$$


zeros

poles

Comments:

- p zeros and p poles
- System BIBO stability requires each pole to have a magnitude within the unit circle; system identification (learning) must be carried out carefully to guarantee stability

MA(q): Moving Average predictive models

- A stationary time series is said to be a moving average of order q if :

$$y(t) = \theta_1 \eta(t-1) + \theta_2 \eta(t-2) + \cdots + \theta_q \eta(t-q) + \eta(t)$$

$$\eta(t) \sim WN(0, \sigma^2)$$

- Where:

- $\Theta = (\theta_1, \dots, \theta_q)$
- Regressor components are terms $f_j(t) = \eta(t-j)$

The predictive model family is $\hat{y}(t) = \sum_{j=1}^q \theta_j f_j(t)$

MA(q): Moving Average predictive models

- By applying the delay operator,

$$y(t) = (1 + \theta_1 z^{-1} + \dots + \theta_q z^{-q}) \eta(t)$$

$$y(t) = \frac{z^q + \theta_1 z^{q-1} + \dots + \theta_q}{z^q} \eta(t)$$

and the transfer function becomes

$$H(z) = C(z) = \frac{z^q + \theta_1 z^{q-1} + \dots + \theta_q}{z^q}$$

- A MA predictor is always BIBO stable

ARMA(p,q): Autoregressive Moving Average predictive models

- ARMA: Combination of AR(p) and MA(q)

$$y(t) = \eta(t) + \sum_{i=1}^p \phi_i x(t-i) + \sum_{j=1}^q \theta_j \eta(t-j)$$

- Using the already defined regressors for both the lagged variables of the dependent variable and the error terms:

$$\hat{y}(t) = \sum_{i=1}^p \phi_i g_i(t) + \sum_{j=1}^q \theta_j f_j(t)$$

- Vector of model's parameters:

$$\Theta = (\phi_1, \dots, \phi_p, \theta_1, \dots, \theta_q)$$

ARMA(p,q): Autoregressive predictive Moving Average models

- Since ARMA contains both AR(p) and MA(q) contributions

$$y(t) = \eta(t) + \sum_{i=1}^p \phi_i x(t-i) + \sum_{j=1}^q \theta_j \eta(t-j)$$

The transfer function is

$$H(z) = \frac{C(z)}{A(z)}$$

And BIBO stability depends on the AR part only

ARIMA(p,d,q): Autoregressive Integrated Moving Average predictive models

- All previous predictive models work under the stationarity assumption.
- ARIMA generalizes ARMA models through *differentiation* to describe non stationary time series (e.g., stock market prices)
- Differentiation eliminates trends and seasonalities
 - E.g., $x'(t) = x(t) - x(t - 1)$
 - Or long term seasonalities

$$x'(t) = x(t) - x(t - d)$$

ARIMA(p,d,q): Autoregressive Integrated Moving Average predictive models

- The model builds a stationary time series by finite differentiation

$$\hat{y}(t) = \sum_{i=1}^p \phi_i g_i(t) + \sum_{j=1}^q \theta_j f_j(t)$$

$$g_i(t) = x(t - i) - x(t - i - d)$$

and we get an ARMA model in the transformed variable

What about the predictor?

- Given system model

$$y(t) = \eta(t) + \sum_{i=1}^p \phi_i y(t-i) + \sum_{j=1}^q \theta_j \eta(t-j)$$

- The single optimal step-ahead predictor becomes

$$\hat{y}(t) = (1 - H^{-1}) y(t)$$

Examples

Power load Forecast: smart grids

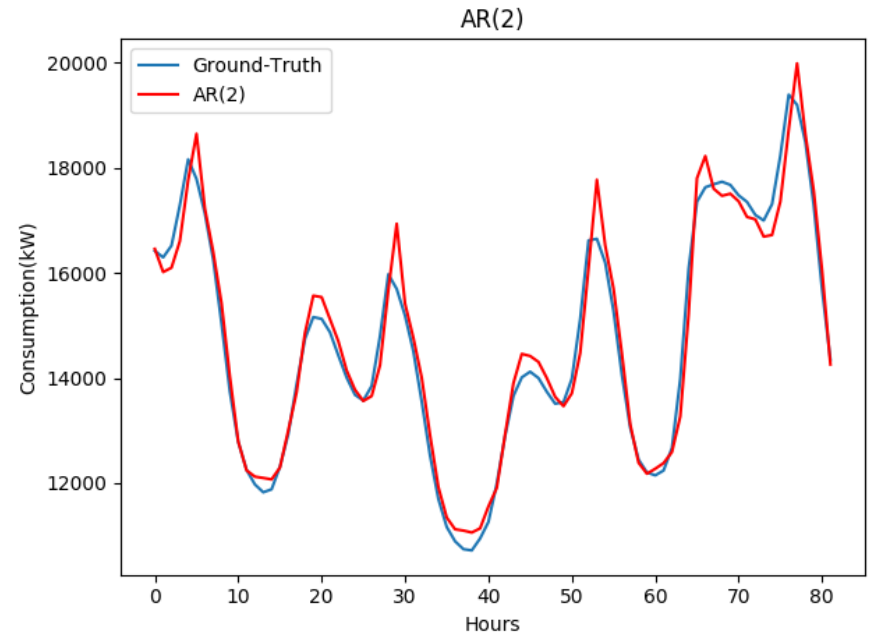
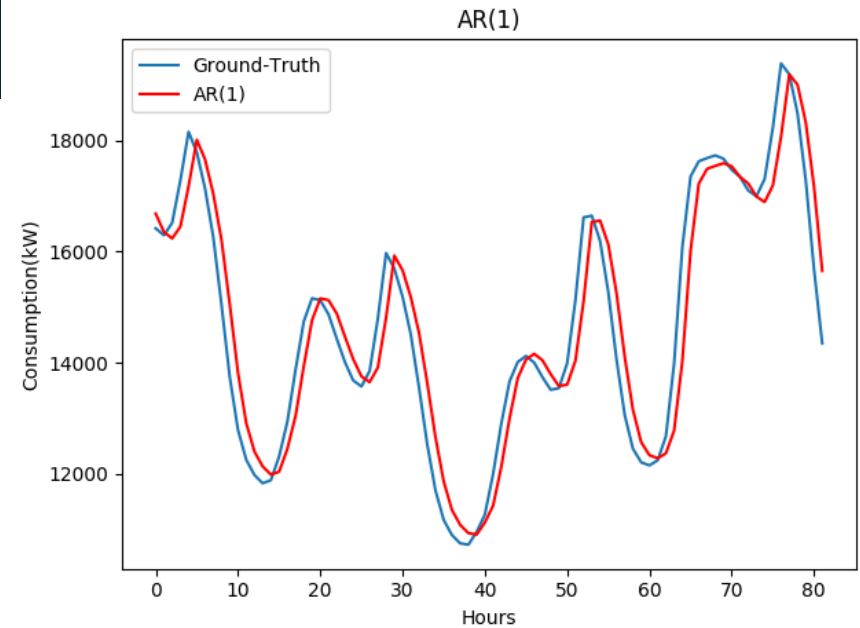
1 step ahead forecast:

- AR(1):

$$\hat{y}(t) = \phi_1 x(t-1)$$

- AR(2):

$$\hat{y}(t) = \phi_1 x(t-1) + \phi_2 x(t-2)$$



Example

Power load forecast

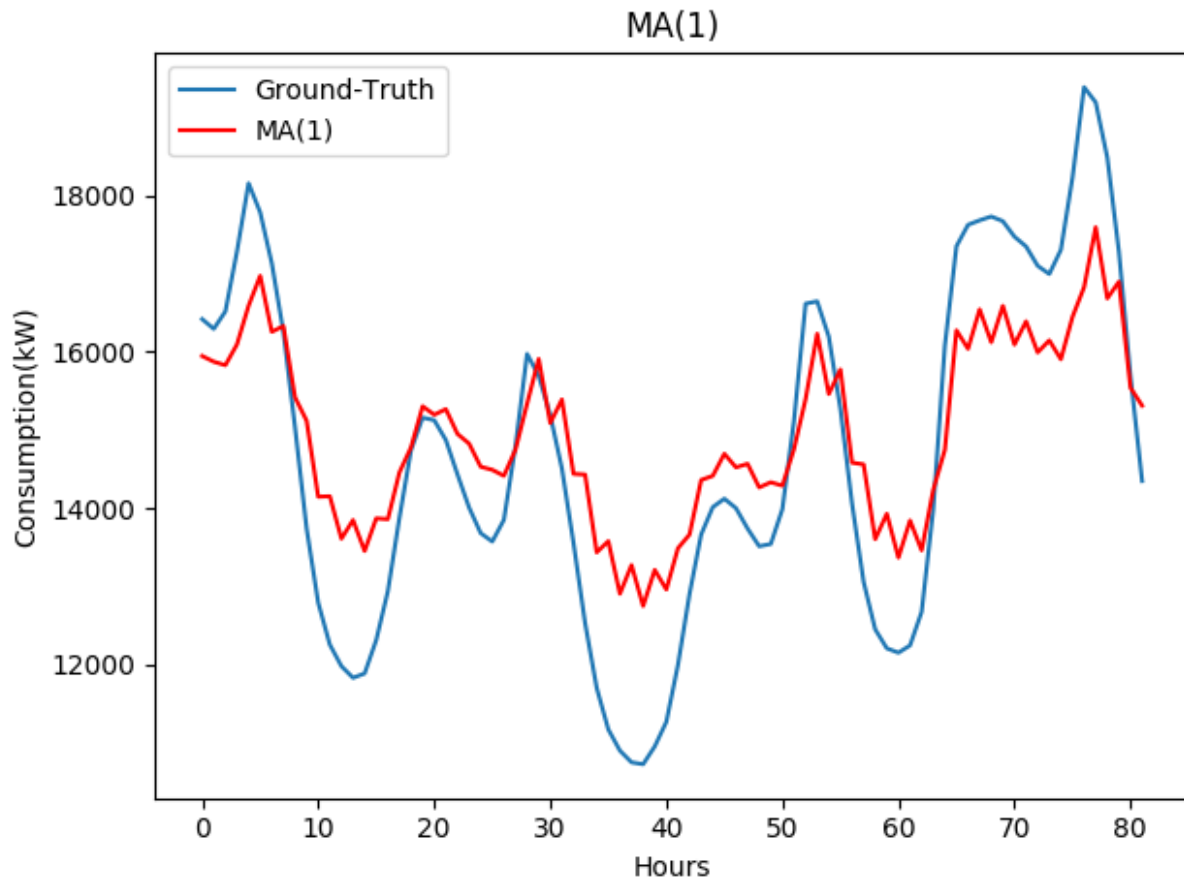
Define the residual

$$\hat{\eta}(t) = y(t) - \hat{y}(t)$$

1 step ahead forecasts:

- MA(1):

$$\hat{y}(t) = \theta_1 \hat{\eta}(t-1)$$

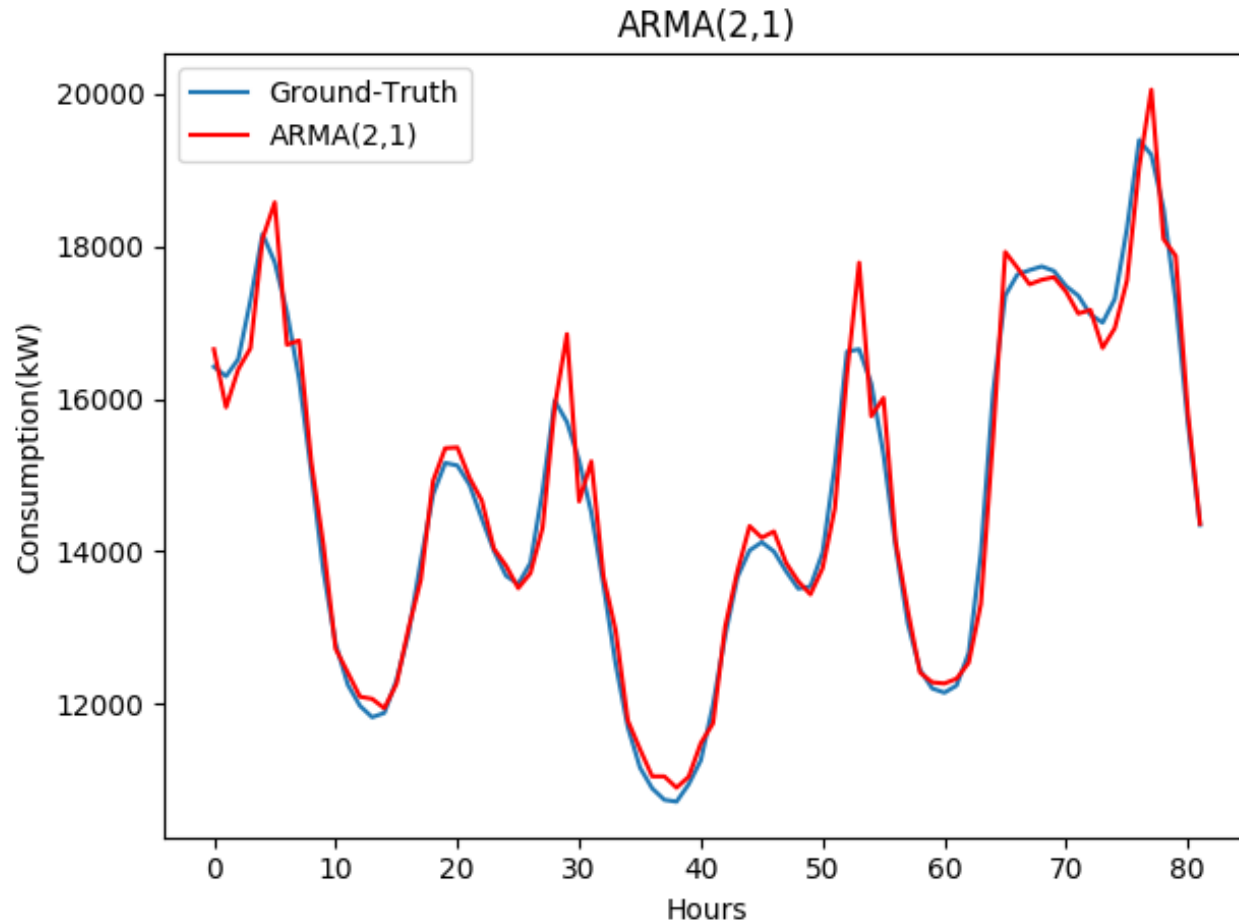


Examples

Power load forecast

1 step ahead forecast:

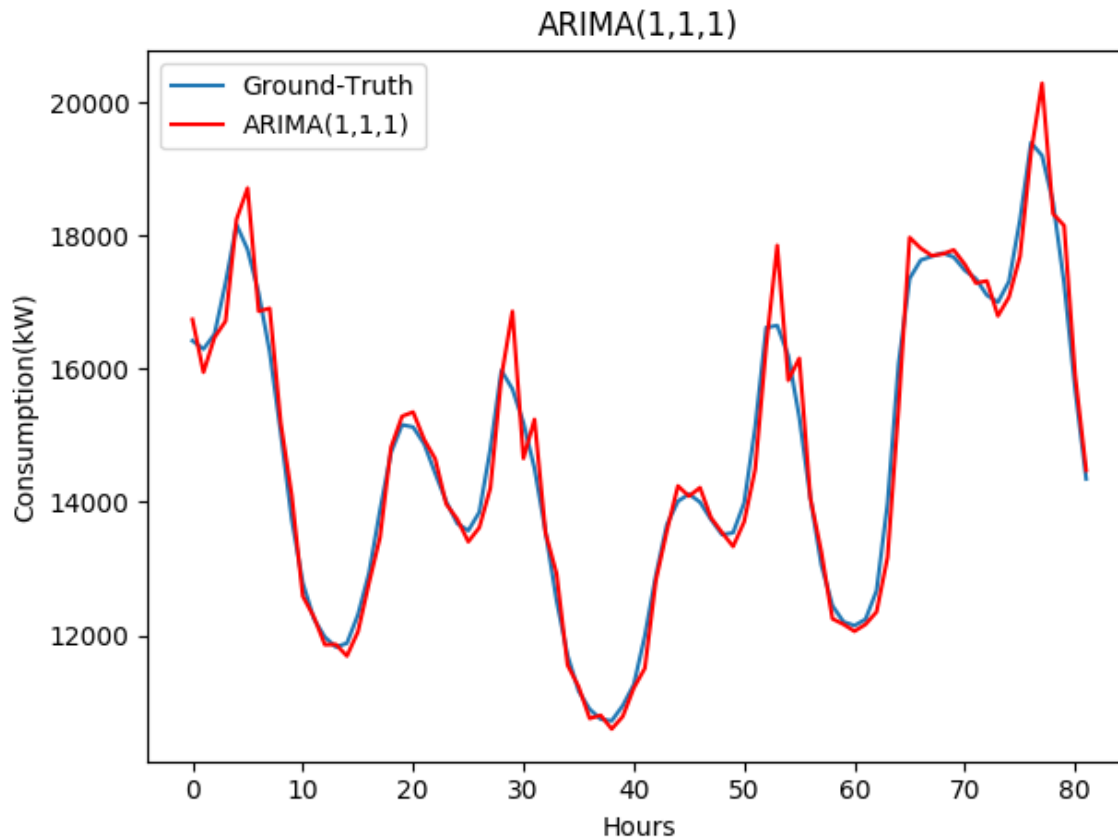
- ARMA(2,1): $\hat{y}(t) = \phi_1 x(t-1) + \phi_2 x(t-2) + \theta_1 \hat{\eta}(t-1)$



Examples

1 step ahead forecast:

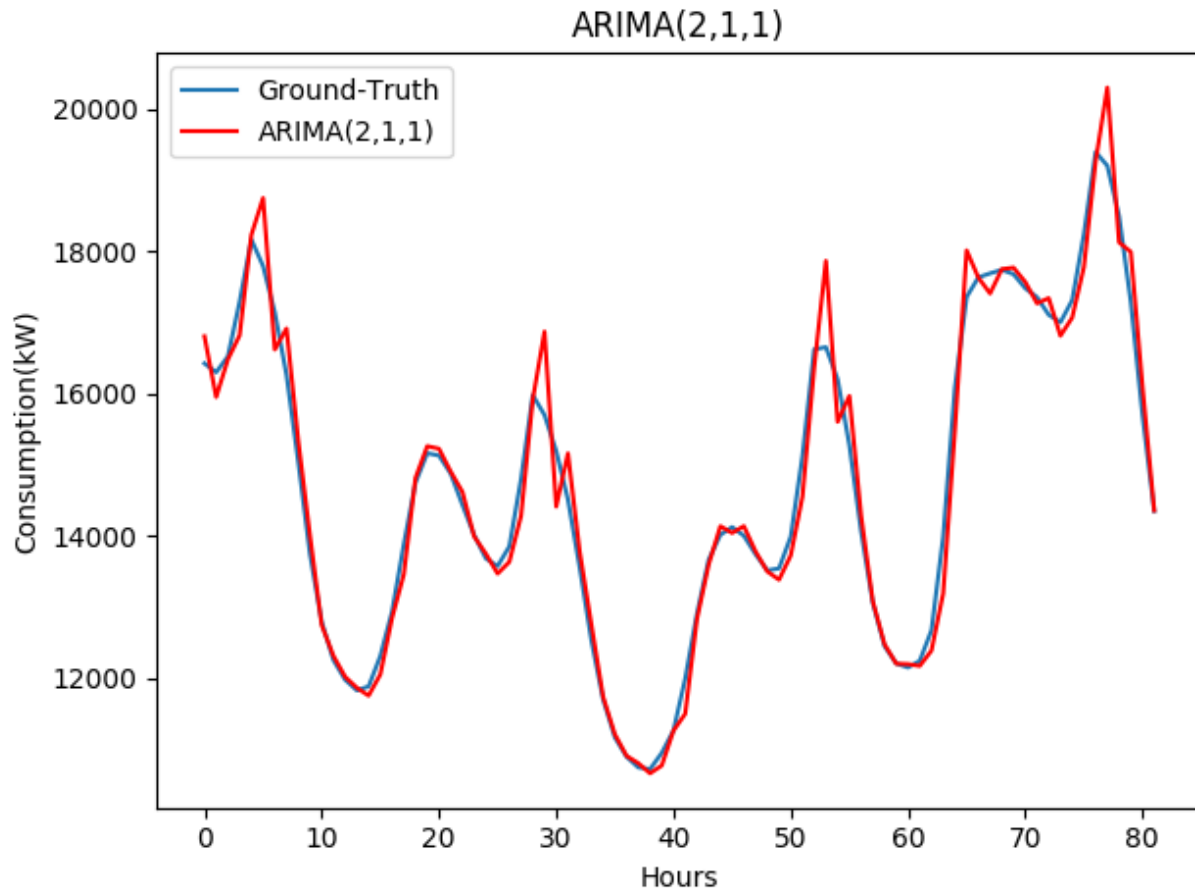
- ARIMA(1,1,1): $x'(t) = x(t) - x(t-1)$
 $\hat{y}(t) = \phi_1 x'(t) + \theta_1 \hat{\eta}(t-1)$



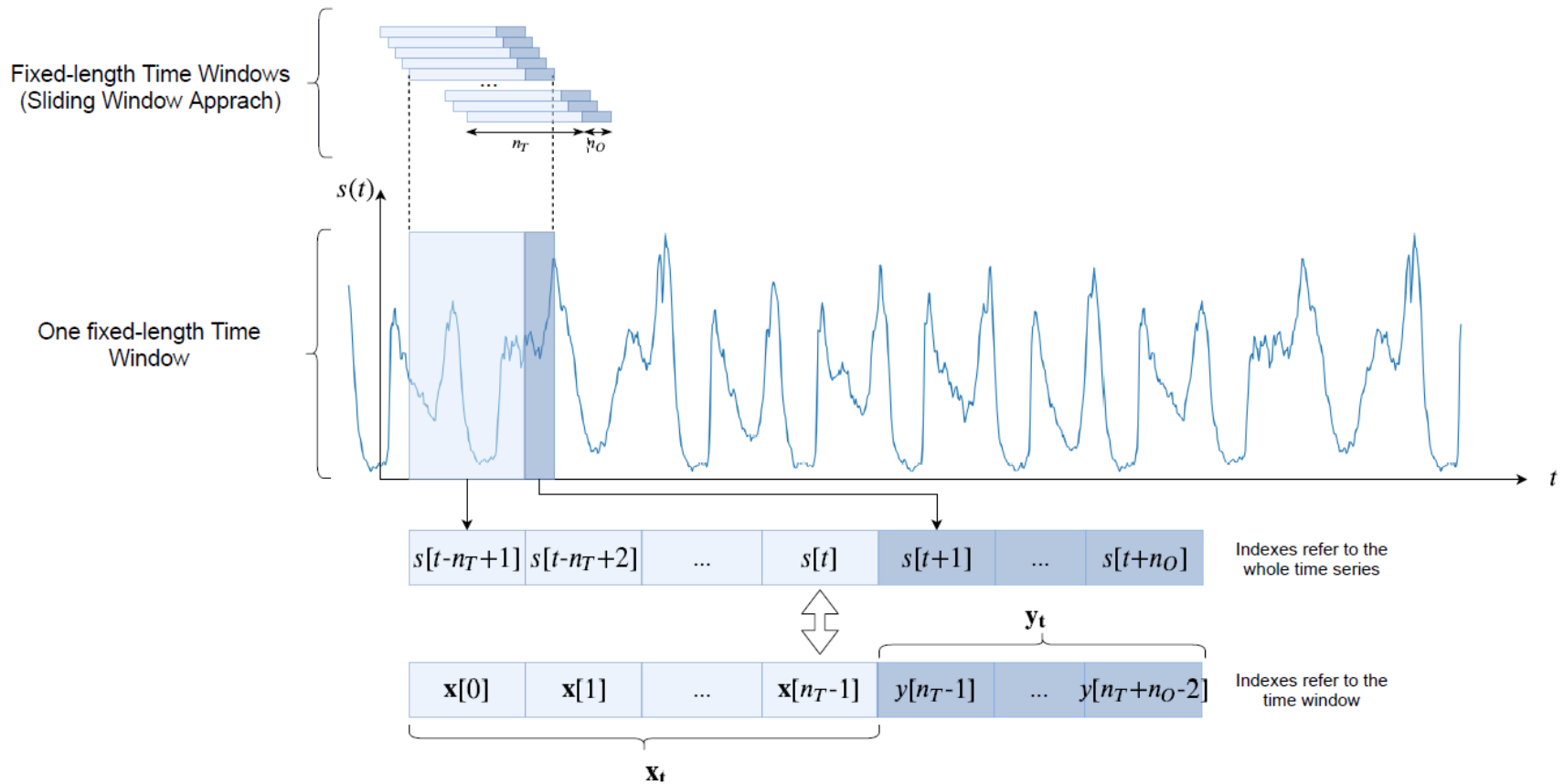
Examples

1 step ahead forecast:

- ARIMA(2,1,1):
$$x'(t) = x(t) - x(t-1)$$
$$\hat{y}(t) = \phi_1 x'(t) + \phi_2 x'(t-1) + \theta_1 \hat{\eta}(t-1)$$



A supervised problem



In the sliding window approach, input data are presented as a regressor vector composed of **fixed time-lagged data** associated with a **window size** of length n_T which slides over the time series. Given this fixed length view of past values, a predictor f aims at forecasting the next n_O values of the time series.

How to estimate ARMA parameters?

- ARMA(p,q) has p+q unknown parameters. Then, we have the noise variance (+1)

$$\Theta = (\phi_1, \dots, \phi_p, \theta_1, \dots, \theta_q, \sigma^2)$$

- How to estimate them?
 - LMS
 - Maximum Likelihood Estimation (MLE)
 - Yule-Walker Estimation

Estimating ARMA parameters

With LMS

- “One shot” solution only for ARMA(p,0) i.e., AR(p)

$$y(t) = \eta(t) + \sum_{i=1}^p \phi_i g_i(t)$$

- We can estimate parameters via LMS (following the classic procedure)

- For ARMA(p,q) with $q > 0$

$$y(t) = \eta(t) + \sum_{i=1}^p \phi_i g_i(t) + \sum_{j=1}^q \theta_j f_j(t)$$

- We cannot estimate with LMS as we cannot build matrices X and Y (lagged errors are unobservables)
- What can we do here?

Estimating ARMA parameters

With MLE

- Maximum Likelihood Estimation

Assumptions:

- The stochastic process X is gaussian and noise terms are i.i.d gaussian
- Data are centered (subtract the sample mean to match the assumption)

- Given the joint pdf for the ARMA model is:

$$f_{\Theta}(X_1, \dots, X_p)$$

- We look for those parameters that maximize the likelihood

$$\hat{\Theta}_{MLE} = \arg \max_{\Theta} L(\Theta|X) = \arg \max_{\Theta} f_{\Theta}(X_1, \dots, X_p)$$

Estimating ARMA parameters

- More in detail

$$\begin{aligned} L(\Theta|X) &= f_{\Theta}(X_1, \dots, X_p) \\ &= \frac{1}{\sqrt{(2\pi)^{\frac{p}{2}} \det(\Sigma(\Theta))}} e^{-\frac{1}{2} X^T \Sigma(\Theta)^{-1} X} \end{aligned}$$

- $\Sigma(\Theta)$ is the covariance matrix of the parameters
- Take the log Likelihood and solve

$$\frac{\partial \log(L(\Theta|X))}{\partial \Theta} = 0$$

- to determine the parameters estimate

Linear time invariant models (LTI)

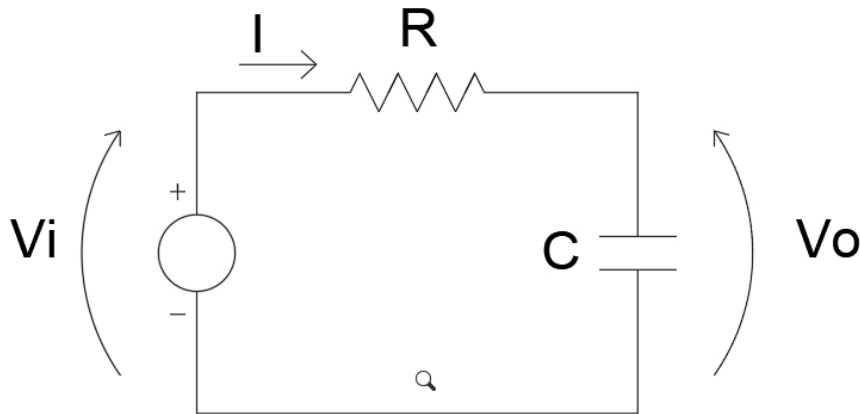
- LTI models extended to include Exogenous variables (system identification):
 - Autoregressive Exogenous (ARX)
 - Autoregressive Moving Average Exogenous (ARMAX)
- Many other LTI model hierarchies exist
- For instance, given the causal, stationary, ARMAX system model

$$y(t) = \eta(t) + \sum_{i=1}^p \phi_i y(t-i) + \sum_{j=1}^q \theta_j \eta(t-j) + \sum_{k=1}^r \psi_k u(t-k)$$

- Where $u(t)$ is the eXogenous variable

Linear time invariant models (LTI)

- Many physical systems can naturally be described with ARX models



$$\frac{dV_o}{dt} + \frac{V_o}{RC} = \frac{V_i}{RC}$$

$$V_o(0) = 0$$

What about the predictor?

- The optimal single step-ahead predictor becomes

$$\hat{y}(t) = (1 - H^{-1})y(t) + H^{-1}Gu(t)$$

- where

$$G(z) = \frac{B(z)}{A(z)} \quad B(z) = \frac{\psi_1 z^{r-1} - \psi_2 z^{r-2} - \dots - \psi_r}{z^r}$$

- We can advantageously extend LTI to the nonlinear case

Time lags and model selection, and Sampling rate. Discussion

Feedforward Neural Networks

- *Autoregressive nonlinear predictive models* use past observations to forecast

$$\hat{y}(t) = f_{\Theta}(x(t-1), \dots, x(t-p))$$

With $y(t) = x(t)$ and f_{Θ} a nonlinear function.

- To stick to the notation generally used in the literature consider the FFNN system model

$$\hat{y}(t) = f_{\Theta}(x(t), \dots, x(t-p-1))$$

with $y(t) = x(t+1)$ and f_{Θ} the non linear NN function.

- **Conceptually, nothing has changed**

Recurrent Neural Networks

- RNNs are an extension of feed forward neural networks for sequence learning.
- Differently from FFNNs, in RNNs the hidden states are given their own dynamics and the network presents its own memory.

Recurrent Neural Networks

- RNN:

$$h(t) = f_{\Theta}(h(t-1), x(t))$$

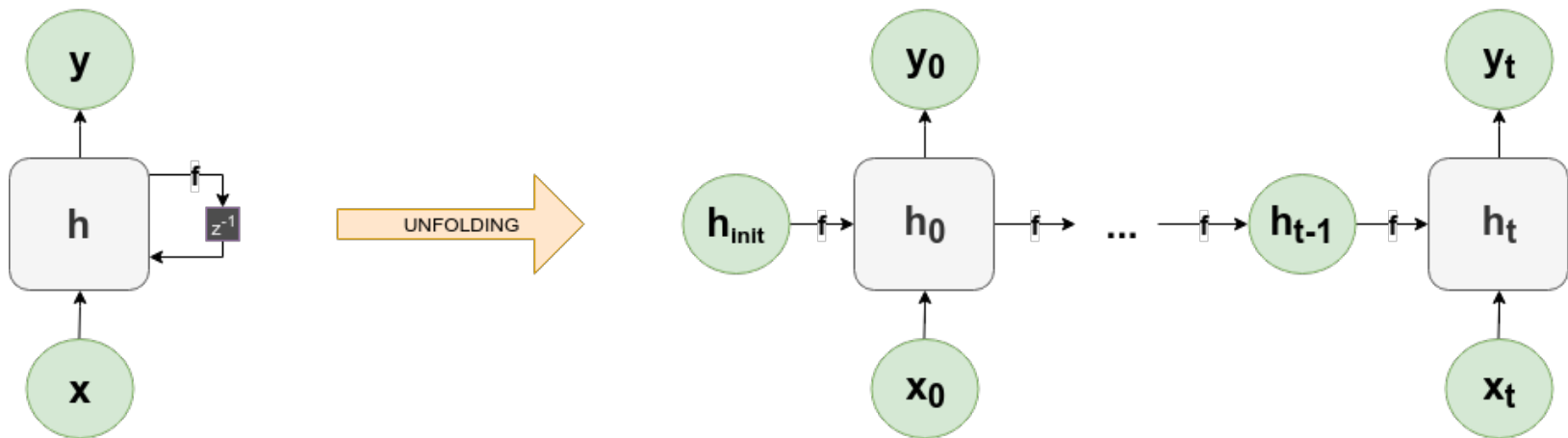
$$h(t) = \tanh(h(t-1)W_{rnn} + x(t)U_{rnn} + b_{rnn}(t))$$

$$\hat{y}(t) = h(t)V_{rnn}$$

- $\Theta_{rnn} = [W_{rnn}, U_{rnn}, V_{rnn}]$ are the parameters of network
- $h(t)$ is the state of the network at time t : a (generally lossy) summary of the past events.

RNNs – Learning procedure

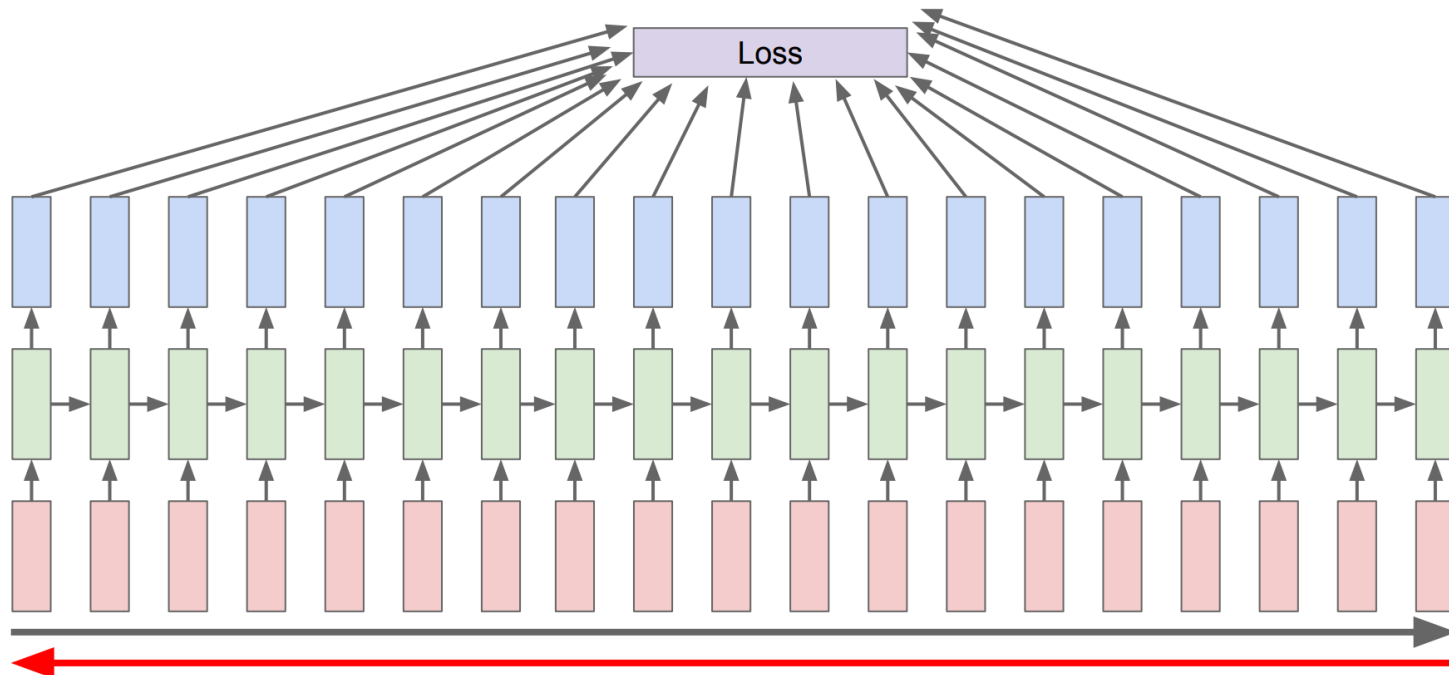
- RNN: weighted, directed, cyclic computational graph with a delay operator.



- Unfolded RNN: weighted, directed, acyclic computational graph. It's a FFNN but the weight matrices are constrained to have the same value in all layer replicas

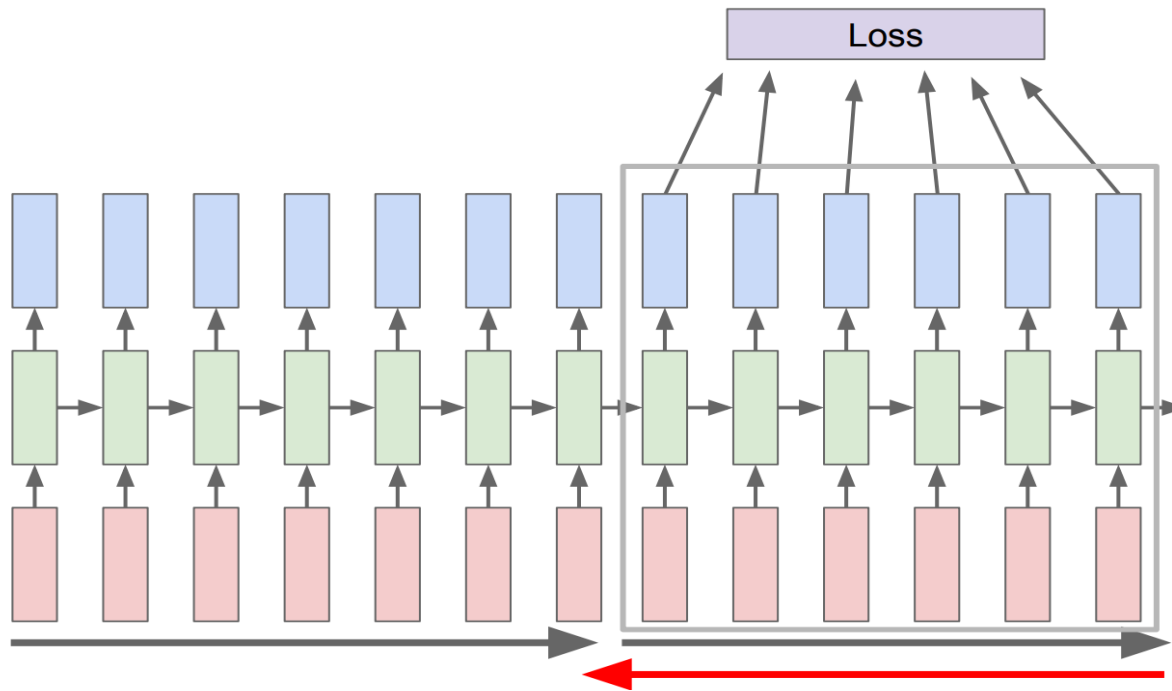
RNNs - Learning

- We know that learning is based on **Backpropagation Through Time (BPTT)**: forward the entire sequence, compute the (final) loss, back-propagate the error through the entire sequence.



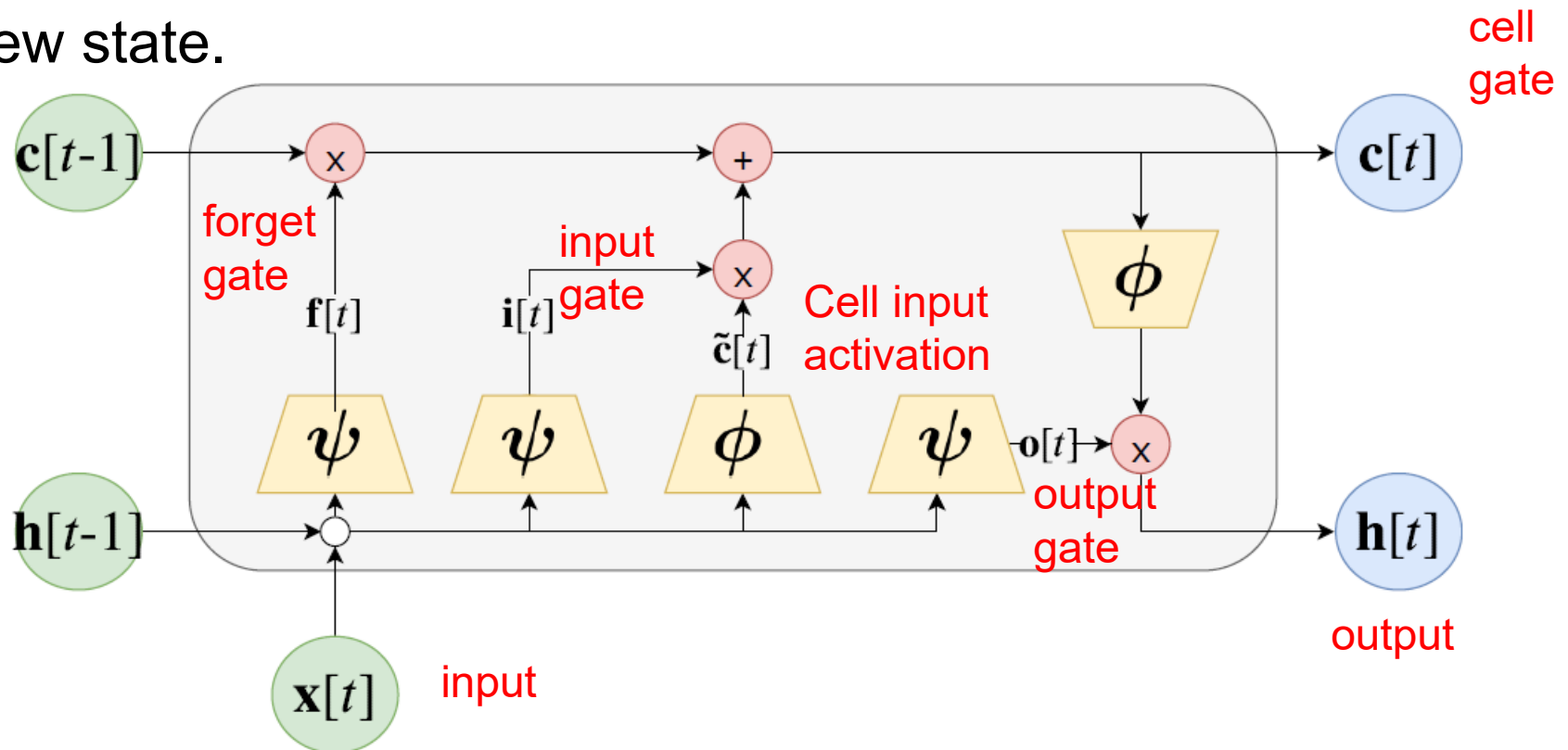
RNNs - Learning

- In practice we use **Truncated BPTT**: the unfolding is truncated after a finite number of timesteps.



Long Short-Term Memory

- RNNs with LSTM aim at dealing with the gradient-related problems of standard RNNs.
- Differently from standard RNN cells, the inner state of the network is a linear combination of the old state and the new state.




Long Short-Term Memory

- RNNs with LSTM cells can be described as:

- $h(t) = f_{\Theta_{LSTM}}(h(t-1), c(t-1), x(t))$

- $c(t) = \mathbf{f}(t) \odot c(t-1) + \mathbf{i}(t) \odot \tilde{c}(t)$
 - $h(t) = \mathbf{o}(t) \odot \tanh(c(t))$

Hadamard
product



- Gates $\mathbf{i}(t)$, $\mathbf{f}(t)$, $\mathbf{o}(t)$ have the form

- $\sigma(h(t-1)W + x(t)U)$

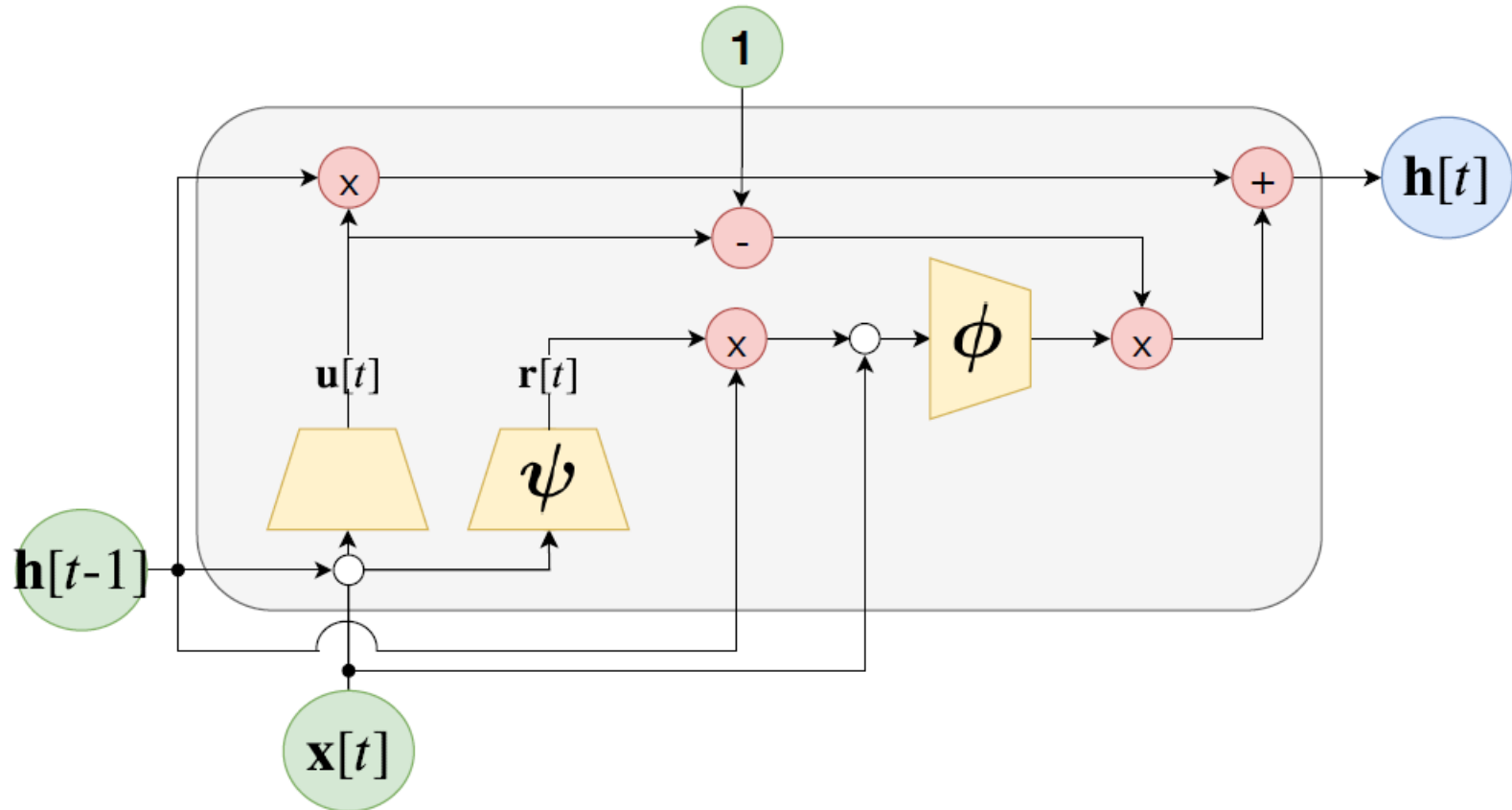
with the respective parameters, while

- $\tilde{c}(t) = \tanh(h(t-1)W_c + x(t)U_c)$

- $\Theta_{LSTM} = [W_f, W_i, W_o, W_c, U_f, U_i, U_o, U_c]$ are the parameters of the network

Gated Recurrent Units

- GRUs are a variant of LSTM where forget and input gates merge.
- The cell is left with update $\mathbf{u}(t)$ and reset $\mathbf{r}(t)$ gates.



Gated Recurrent Units

- GRUs can be described as:

- $h(t) = f_{\Theta_{GRU}}(h(t-1), x(t))$

- $$\left[\begin{aligned} &\tilde{h}(t) = \tanh\left((\mathbf{r}(t) \odot h(t-1))W_c + x(t)U_c\right) \\ &h(t) = \mathbf{u}(t) \odot h(t-1) + (1 - \mathbf{u}(t)) \odot \tilde{h}(t) \end{aligned} \right.$$

- The gates $\mathbf{u}(t), \mathbf{r}(t)$ have the form

- $\sigma(h(t-1)W + x(t)U)$

with the respective parameters

- $\Theta_{GRU} = [W_u, W_r, W_c, U_u, U_r, U_c]$ are the parameters of the network

Deep RNN

- An unfolded RNN is already a deep model given its multiple non-linear processing layer
- However, further depth can be introduced by stacking recurrent layers on top of each other.

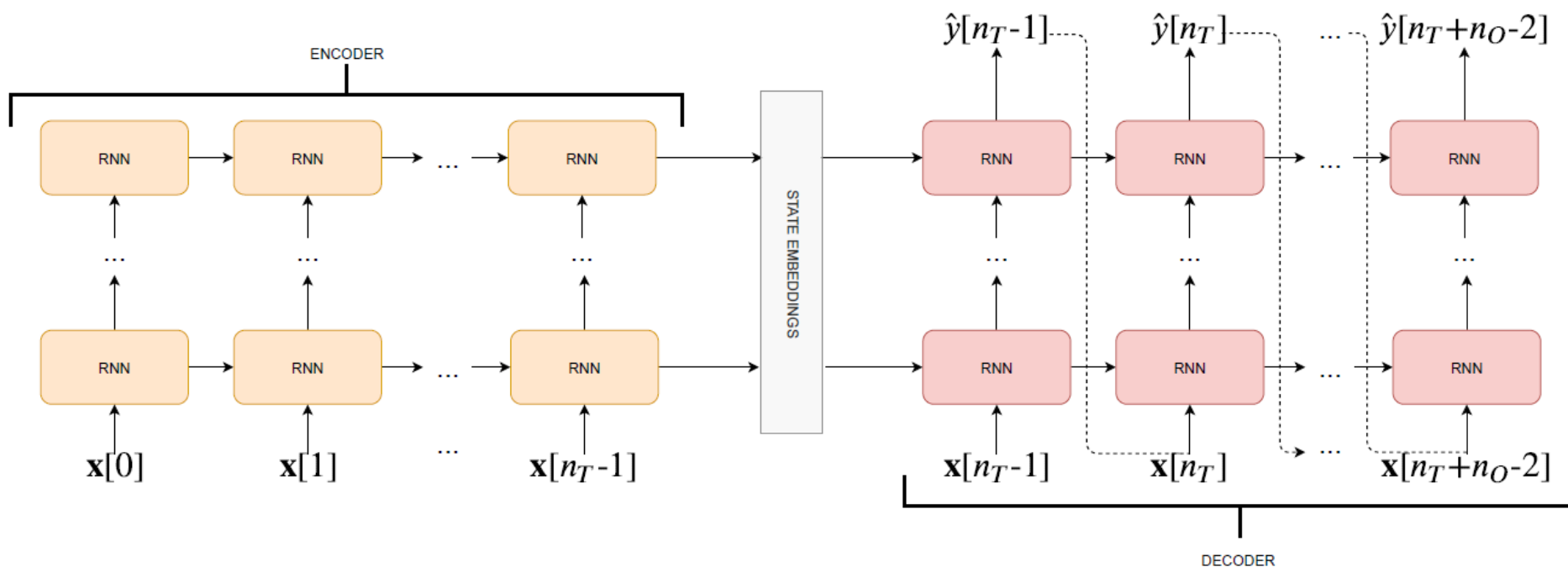
$$h_{\ell}(t) = f_{\Theta}(h_{\ell}(t-1), h_{\ell-1}(t)) \quad \ell = 1, 2, \dots, L$$

Where

- $h_{\ell}(t)$ is the hidden state at time t for layer ℓ
- $h_0(t) = x(t)$

Sequence to Sequence

- Seq2seq (or encoder-decoder) models were designed to produce output sequences of arbitrary length
- Two networks are employed: an encoder and a decoder.



Sequence to Sequence

- The encoder reads one input at time and generates a fixed dimensional vector representation of it

$$c = f_{\Theta_f}(x(t))$$

c is called *context* and is a function of the last hidden state of the encoder.

- The decoder will learn how to produce the output sequence given the context vector

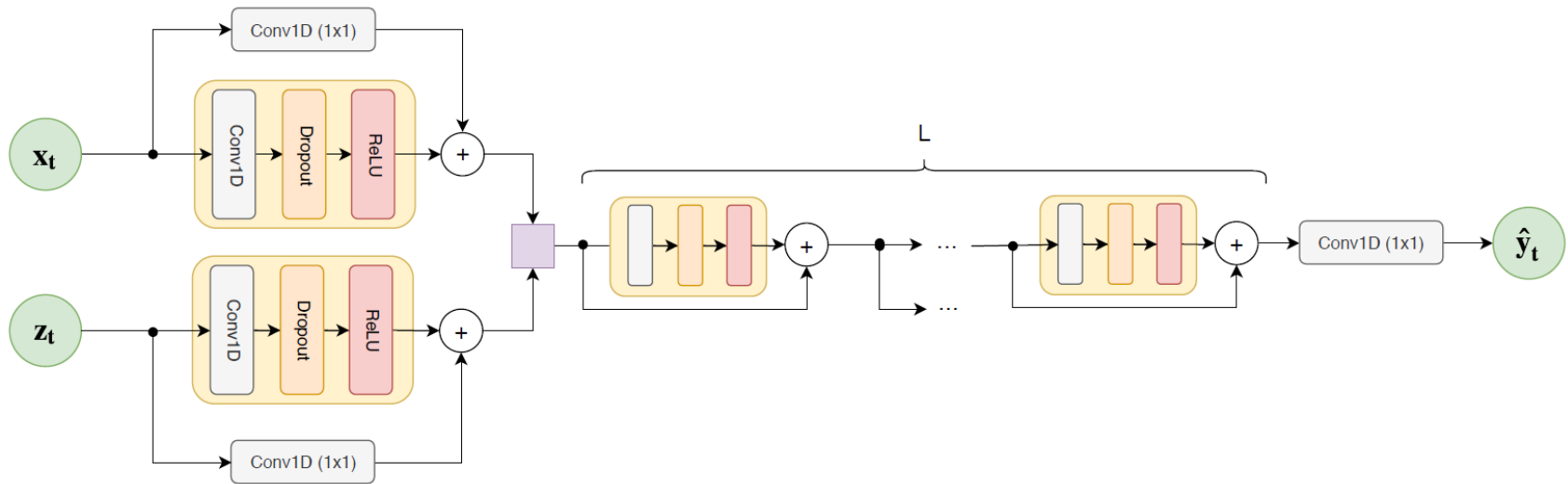
$$\hat{y}(t) = g_{\Theta_g}(c)$$

Convolutional Neural Networks

These networks take advantage of

1. **local connectivity**
2. **parameter sharing**: the same kernel is used for each location. This allows to reduce the number of parameters to be learned.
3. **translation equivariance**: the network is robust to input shifts.

CNNs for time-series forecasting



- Temporal Convolutional Networks (**TCNs**) are autoregressive convolutional networks and process sequences of arbitrary length to output a sequence of the same length.
- In the above TCN NARX, \mathbf{x}_t is the vector of past time series values (e.g., power load), \mathbf{z}_t is the vector of exogenous variables (i.e., temperature, weather forecast), $\hat{\mathbf{y}}_t$ is the output vector.

CNNs for time-series forecasting

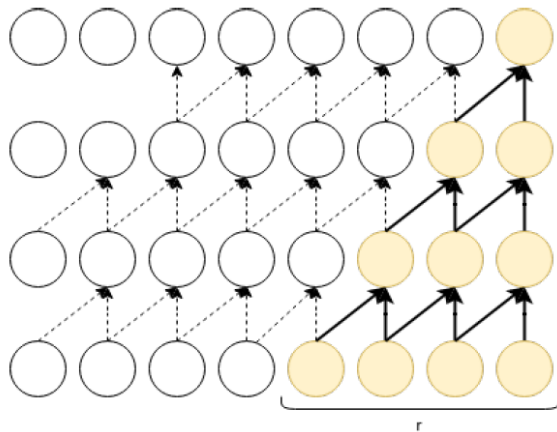


Figure A

Figure A shows a 3-layers CNN with **causal convolution**: the estimated value at time t depends only on past samples. The **receptive field** r , i.e. the number of input neurons to which the filter is applied, is 4.

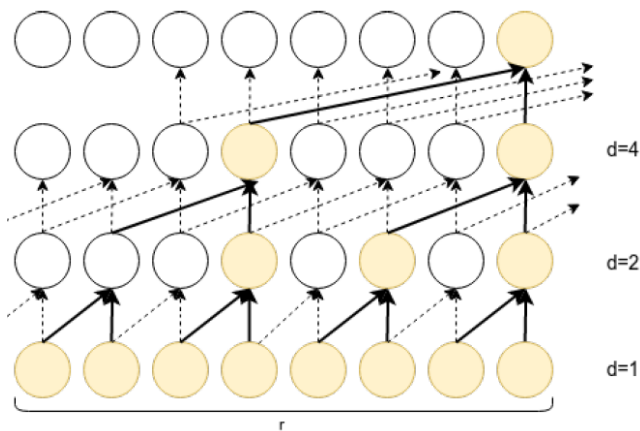


Figure B

In case of **dilated causal convolutions** (Figure B), the receptive field is increased to learn long-term dependencies in the time-series.

Multi-Step Prediction strategies

- **Recursive:** a single model is trained to perform a one-step ahead forecast given the input sequence. Subsequently, the forecasted output is recursively fed back and considered to be the correct one. The output vector is built using the predicted n_o scalars.
- **Direct:** design n_o independent predictors f_k , $k = 1, \dots, n_o$, each of which providing a forecast at time $t + k$. Each predictor f_k outputs a scalar value, but the input vector is the same to all the predictors.
- **MIMO** (Multiple input - Multiple output): a single predictor f is trained to forecast a whole output sequence of length n_o in one-shot. Differently from the others, the output of the predictor is not a scalar but a vector.

Figures of merit

The figures of merit mostly used in the prediction task are

$$\text{MAE} = \frac{1}{N} \sum_{i=0}^{N-1} \frac{1}{n_O} \sum_{t=0}^{n_O-1} | \hat{y}_i[t] - y_i[t] |$$

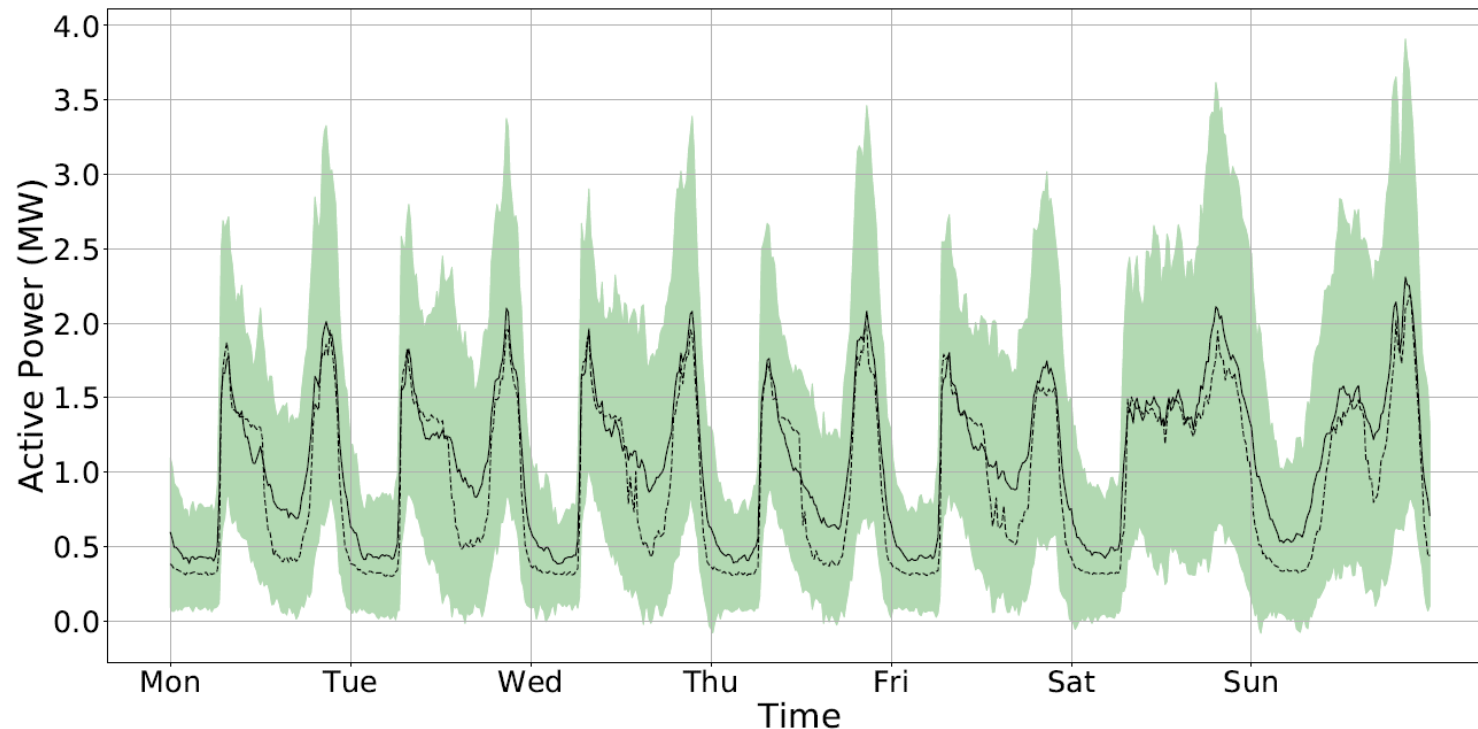
$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} \frac{1}{n_O} \sum_{t=0}^{n_O-1} (\hat{y}_i[t] - y_i[t])^2}$$

$$\text{NRMSE}_{\%} = \frac{\text{RMSE}}{y_{\max} - y_{\min}} \cdot 100$$

where N is the number of testing samples, $y_i[t]$ and $\hat{y}_i[t]$ are the real and estimated output value at time t for sample i , y_{\max} and y_{\min} are the maximum and minimum target value of the training dataset (dynamics).

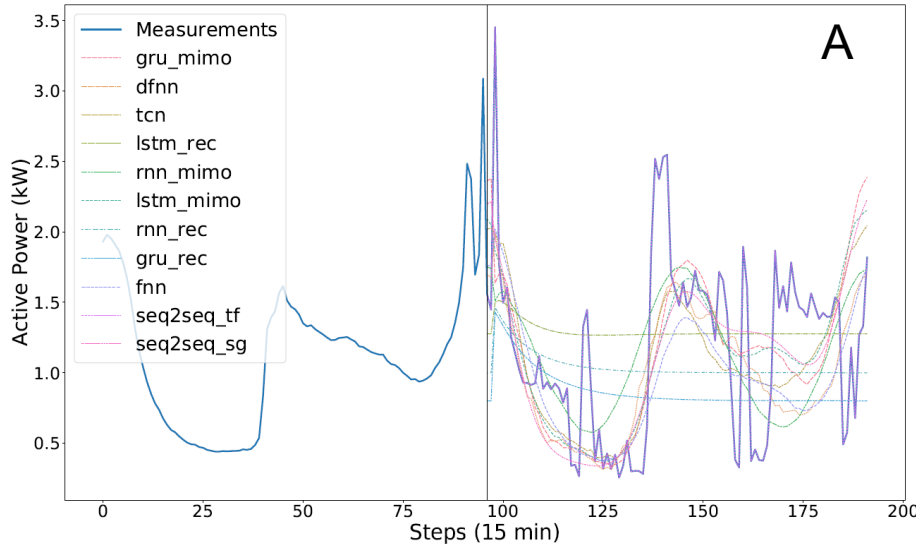
Some experiments

- Household electric power consumption (single house).

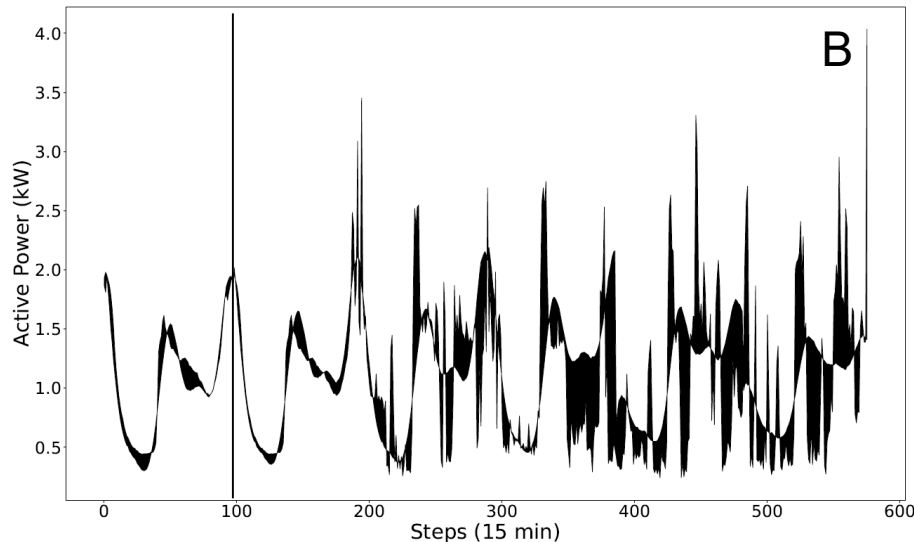


- There is high variance over time; the average signal is not appropriate to build the predictor.

Real data experiments



A) The crystal ball predicting the future does not exist.



B) the difference between the real value and the predicted one: the prediction error **amplifies** over time.

Dealing with large multivariate time series. Discussion

Data streams and graph streams

- In others, we derive graphs from signals

