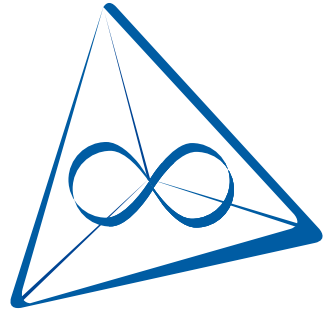
Time series analysis

Predicting, Modeling and Machine Learning

Eckehard Olbrich
Max Planck Institute for Mathematics in the Sciences

Leipzig

06.07.2021



What is time series analysis?

Time series

Possibly vector valued measurements $\mathbf{x}_i$, that are characterised by an one dimensional index, which is usually the time, but could be also a spatial direction.

Tasks

- Characterization of the source of the signal, pattern detection and diagnosis
- Studying dependencies between different observables, e.g. causal inference
- **Prediction, modelling**, and control

Plan for this lecture

1. Prediction and modeling in nonlinear time series analysis
2. Bayesian approaches
3. Machine learning: reservoir computing, Long Short-Term Memory
4. Software packages

Ressources:

- https://personal-homepages.mis.mpg.de/olbrich/ - in particular the the lecture on Modeling and Prediction in "Data analysis and Modeling".
- Kevin McGoff, Sayan Mukherjee, Natesh Pillai, Statistical inference for dynamical systems: A review, Statist. Surv. 9: 209-252 (2015).
- TISEAN 3.0.1 Nonlinear Time Series Analysis package (in C and FORTRAN) by Rainer Hegger, Thomas Schreiber and Hoger Kantz

Prediction and Modeling

Deterministic dynamics

$$\boldsymbol{x}_{n+1} = \boldsymbol{F}(\boldsymbol{x}_n)$$

which reduces to

$$x_{n+1} = F(x_n, \ldots, x_{n-m+1})$$

in the case of delay embedding.

Local methods: Basic idea: Looking for similar events in the past and using their future for prediction.

- Local constant: Using the average as prediction.
- Local linear: Fitting a linear model for similar events, i.e. neighbors in phase space.

Global models: Parameterizing the function $\boldsymbol{F}$ and fitting the parameters.

- Polynomials
- Radial basis functions
- Neural networks

Local constant prediction — *lzo*

We want to predict x_{n+1}.

1. Looking for similar events in the past. Formally, looking for $\boldsymbol{x}_k$ with $|\boldsymbol{x}_n - \boldsymbol{x}_k| \leq \epsilon$, thus $\boldsymbol{x}_k \in \mathcal{U}_\epsilon(\boldsymbol{x}_n)$
2. Then our prediction is

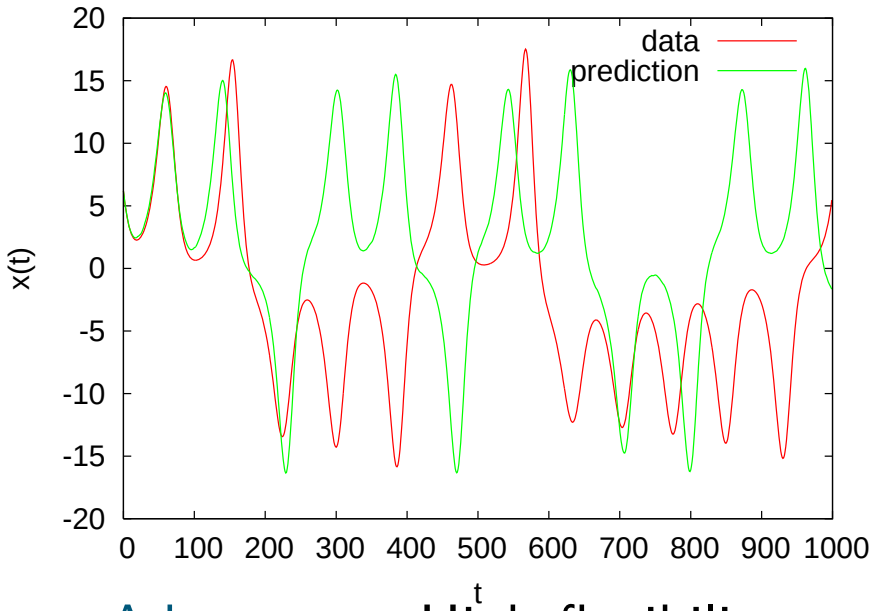
$$\hat{x}_{n+1} = \frac{1}{|\mathcal{U}_\epsilon(\boldsymbol{x}_n)|} \sum_{k:\boldsymbol{x}_k \in \mathcal{U}_\epsilon(\boldsymbol{x}_n)} x_{k+1}$$

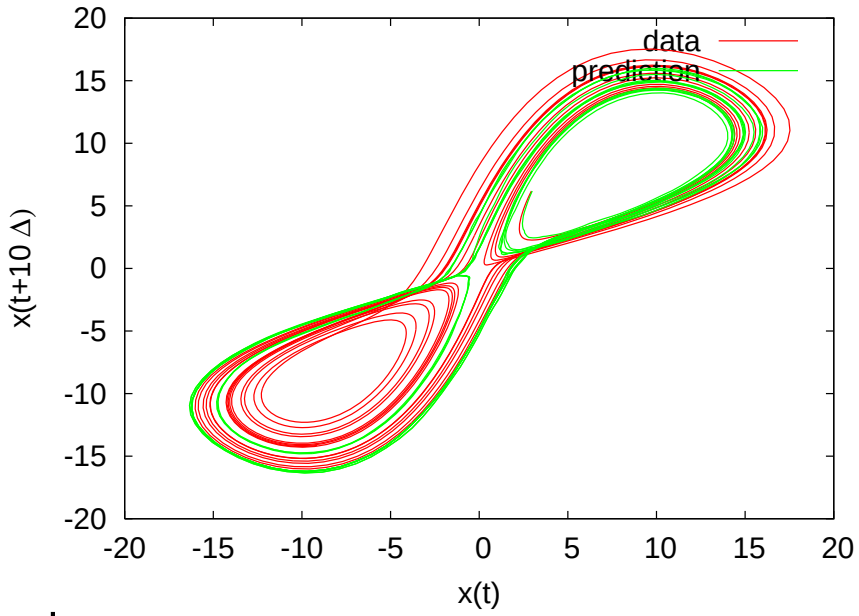
Parameter:

- Embedding parameter: delay d and embedding dimension m
- Minimal number of neighbours and/or neighbourhood size

Local constant prediction

Lorenz system, $m = 5$, $d = 5$





Advantage: High flexibility and robustness.

Disadvantage: Bad approximation of $\boldsymbol{F}(\boldsymbol{x}_n)$.

- Large bias at the boundaries.
- Dynamics more regular than the original one.

Optimal number of neighbours: trade-off between bias and variance of $\hat{\boldsymbol{F}}$

Local linear prediction — *lfo*

We want to predict x_{n+1}.

1. Looking for similar events in the past. Formally, looking for $\boldsymbol{x}_k$ with $|\boldsymbol{x}_n - \boldsymbol{x}_k| \leq \epsilon$, thus $\boldsymbol{x}_k \in \mathcal{U}_\epsilon(\boldsymbol{x}_n)$
2. Now our prediction is not the average of the futures of the similar from the past, but made by a linear model of these events with

$$\hat{x}_{n+1} = \boldsymbol{A}_n \boldsymbol{x}_n + \boldsymbol{b}_n$$

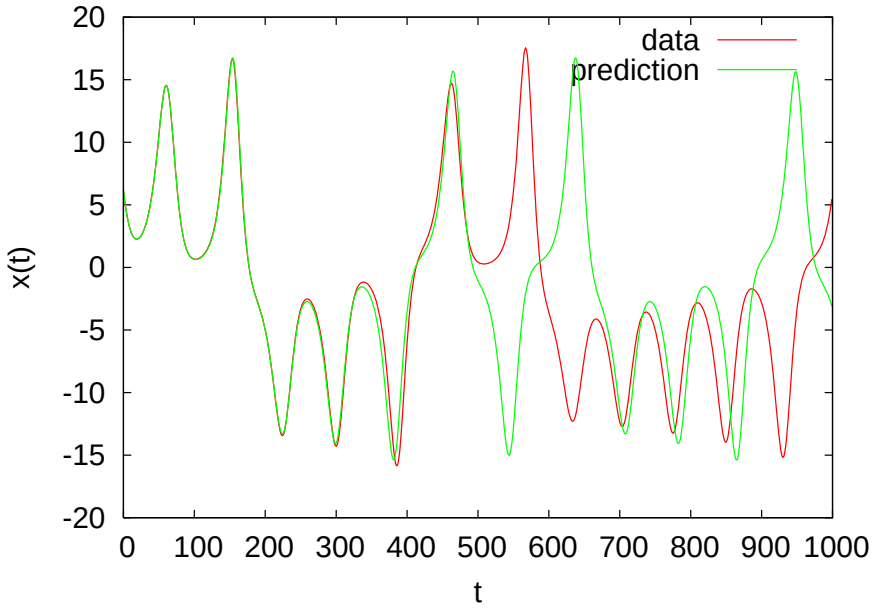
   being the optimal linear predictor for the x_{k+1}.

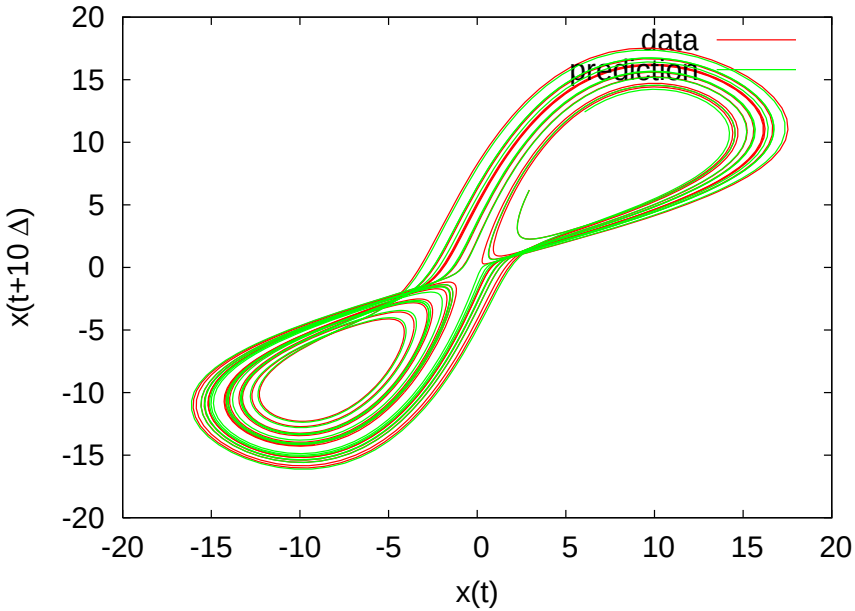
Parameter:

- Embedding parameter: delay d and embedding dimension m
- Minimal number of neighbours and/or neighbourhood size

Local linear prediction

Lorenz system, $m = 5$, $d = 5$





Advantage: High flexibility, better model than local constant model

Disadvantage: Less robust than local constant, in particular with noisy data

Global models — Polynoms

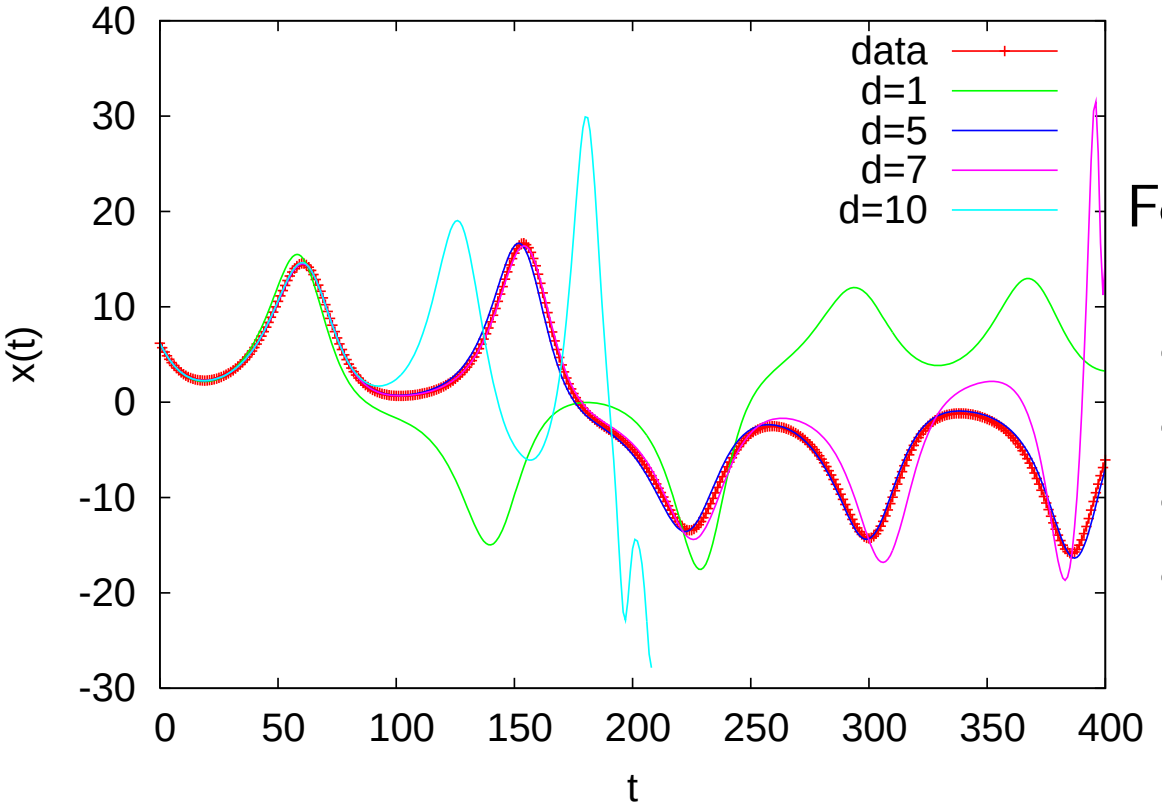
$$\boldsymbol{F}(\boldsymbol{x}) = \sum_{i_1,\ldots,i_m} a_{i_1 i_2 \ldots i_m} x_n^{i_1} x_{n-1}^{i_2} \ldots x_{n-m+1}^{i_m}$$

with the sum going over all m-tupel $(i_1, \ldots, i_m)$ with $\sum_{k=1}^{m} i_k \leq p$.
Parameter:

- Embedding parameter: delay d and embedding dimension m
- Order of the polynom p. $p = 1$ corresponds to a linear model.

Polynomial models - Example

Lorenz system: $m = 5$, $p = 3$



Forecast errors:

$d = 1$	1.153983e-04
$d = 5$	7.063427e-05
$d = 7$	6.116494e-05
$d = 10$	7.397461e-05

Advantage: Easy to interprete (at least in some cases)

Disadvantage: Very often unstable — trajectory becomes unbounded, explosion of number of parameters

Global models — Radial Basis Functions

$$F(\boldsymbol{x}) = a_0 + \sum_{k=l}^{p} a_k \Phi(||\boldsymbol{x} - \boldsymbol{y}_i||)$$

- Functions $\Phi(r)$ are bellshaped, i.e. maximal at $r = 0$ and rapidly decaying towards zero with increasing r.
- Number and width of the functions Φ being fixed, the estimation of the a_k is a linear problem and could be estimated using least squares.
- rbf in TISEAN uses Gaussians with the standard deviation of the data

Least square estimation (LSE)

How do we fit the models?

- The model

$$\hat{x}_{n+1} = \sum_i a_i f_i(\boldsymbol{x}_n) \qquad \boldsymbol{x}_n = x_n, x_{n-1}, \ldots, x_{n-m+1}$$

- Root mean square error

$$\mathrm{RMSE} = \sqrt{\frac{1}{N-m} \sum_{n=m+1}^{N} (\hat{x}_n - x_n)^2}$$

- Minimizing the RMSE gives for each i one equation of the form

$$0 = \frac{1}{N-m} \sum_{n=m+1}^{N} (x_n - \sum_k a_k f_k(\boldsymbol{x}_{n-1})) f_i(\boldsymbol{x}_{n-1})$$

- System of linear equations for the a_i

Maximum Likelihood (ML)

- Likelihood: $\mathrm{L} = p(x_N, x_{N-1}, \ldots, x_2, x_1 | a_1, \ldots, a_K)$
- Likelihood from an explicit assumption about the distribution of the errors:

$$x_{n+1} = \sum_i a_i f(\boldsymbol{x}_n) + \epsilon_{n+1}$$

$$L = \prod_{n=m+1}^{N} p(\epsilon_n)$$

$$L = \prod_{n=m+1}^{N} p(x_n - \sum_i a_i f_i(\boldsymbol{x}_{n-1}))$$

- Maximum likelihood: We want to find the parameters of our model, that maximize the likelihood of observing our data.

Equivalence between LSE and ML

- Log-Likelihood

$$\log L = \sum_{n=m+1}^{N} \log p(\epsilon_n)$$

- If the errors are Gaussian

$$p(\epsilon) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\epsilon^2}{2\sigma^2}}$$

$$\log p(\epsilon_n) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(x_n - \sum_i a_i f_i(\boldsymbol{x}_{n-1}))^2$$

maximizing the log likelihood is equivalent to minimizing the root mean square error.

Increasing the number of terms in the function approximation will increase the likelihood on the given data. But, it might not be better on new data. What can be done about it?

- **Cross-validation**: Dividing your data in training and test data. Fitting the model on the training data and evaluating the likelihood on the test data. Repeat the procedure with different partitions in training and test data and average. Problem. Not feasible if you have very small data sets.
- **Model costs** (regularization): Include in your cost function not only the prediction error, but also a term that gets larger for larger models. For instance, the Akaike Information Criterion (AIC) adds a cost term to the log likelihood that is proportional to the number of parameters divided by the number of data points.
- Conceptionally appealing is **minimal description length (MDL)** principle by Rissanen: The more parameters the model has the more bits are needed to ecode the model, but the less bits are needed for encoding the residuals. The best model is the model which leads to the shortest encoding of the data.

Some general remarks — Modelling

- Prediction $\neq$ Modelling. Minimizing the one-step prediction error leads usually to good short term predictions, but not to nessecarily to good models (in particular for chaotic systems).
- The model might be
  - Unstable, i.e. the trajectory diverges (*polynom*).
  - Converge to an attractor at different positions in the phase space.
  - Model shows qualitatively different behavior than the data, e.g. periodic instead of chaotic (*lzo-run*)
- One possibility: minimizing n-step prediction errors instead of only the 1-step prediction error. Minimizing the n-step prediction error is, however, already a **non-linear** problem. Thus there are computational problems and also problems of existence and uniqueness.
- Nonlinear state space models (dynamical + measurement noise)

$$x_{n+1} = F(\boldsymbol{x}_n) + \xi_n \qquad y_n = x_n + \epsilon_n$$

Bayesian inference

- Simplified notation:

$$\begin{aligned} \text{data}: \quad & p(\textbf{data}) := p(x_1, \ldots, x_N) \\ \text{parameter}: \quad & p(\boldsymbol{a}) := p(a_1, \ldots, a_K) \\ \text{Likelihood}: \quad & p(\textbf{data}|\boldsymbol{a}) \end{aligned}$$

- Aim is to estimate the posterior

$$p(\boldsymbol{a}|\textbf{data}) = \frac{p(\textbf{data}|\boldsymbol{a})p(\boldsymbol{a})}{p(\textbf{data})}$$

- Prior distribution should encode prior knowledge about the problem
- For model estimation one could use the maximum a posteriori probability (MAP) estimate, the mean or the median of the posterior.

Advantages

- Prior distribution could act as a regularization term
- Posterior provides not only an estimate of the model parameters but also about its uncertainty

Disadvantages

- Higher computational costs
- Inference of the full posterior often not feasible
- No clear procedure to determine the prior

Choosing the prior

- In case of an uniform prior MAP is equivalent to ML.
- Often conjugate priors are used. They allow for a closed form expression of the posterior: the posterior has the same algebraic form as the prior, but different parameters. For instance, the Gaussian distribution is self conjugate. The problem is, that the conjugate prior might be hard to justify as representing "prior knowledge".
- On the other side "prior knowledge" is also not well defined and some people want to use "uninformed" or "objective" priors.
- In the case of time series analysis we can paramterize the same model differently. Therefore, in order to encode the same prior knowledge, different prior distributions have to be used for different parametrizations.

Example: Autoregressive model

$$x_n = \sum_{i=k}^{p} a_k x_{n-k} + \epsilon_n$$

The parameters a_i can be transformed into freqencies and damping constants of stochastically driven harmonic oscillators:

$$z^p - \sum_{k=1}^{p} a_k z^{p-k} = \prod_{k=1}^{p} (z - z_k) \qquad z_k = r_k e^{i\phi_k}$$

$$\text{frequencies} \qquad f_k = \frac{\phi_k}{2\pi\Delta} \qquad \Delta \quad \text{is the sampling time}$$

$$\text{damping} \qquad \gamma_k = \frac{1}{\tau_k} = -\Delta^{-1} \log r_k$$

In order to be stable, the z_k have to be in the unit circle. But what about the ditribution?
On the other side, often exponential priors are used for the a_i.

Machine Learning

- There seems to be an ongoing debate about the difference of "statistical inference" and "machine learninng". But, a core point seems to be that machine learning is mainly concerned about predictions and less about interpretable models.
- In this sense also many parts of time series analysis is more on the side of machine learning than on the side of statistical modeling or inference
- Practically, machine learning, and in particular deep learning, is working with artifical neural networks.
- Thus, we will look at two architectures that are also used in a time series context: "reservoir computing" or "liquid state machines" and "Long Short-term memory" (LSTM).

Neural networks

- Input layer $\boldsymbol{x}$, output layer $\boldsymbol{y}$
- One layer neural network:

$$y_i = F(\sum_j w_{ij} x_j)$$

$$\boldsymbol{y} = F(\boldsymbol{W}\boldsymbol{x})$$

  with F being for instance a sigmoid function $F(x) = \tanh(b + x)$

- Multilayer feedforward ($(m-1)$ hidden layer):

$$\boldsymbol{y} = F(\boldsymbol{W}_m \dots F(\boldsymbol{W}_2 F(\boldsymbol{W}_1 \boldsymbol{x}))$$

- Neural networks are universal function approximators (e.g. Hornik 1981)

Recurrent neural networks

- Feed forward networks are representations for a function between the input and the output.
- Recurrent neural networks contain not only feed forward, but also feedback connections.
- Lets call the state of the hidden nodes $\boldsymbol{z}$ and they are connected accordig to a weighted adjacency matrix

$$\boldsymbol{z}_{n+1} = F(\boldsymbol{A}\boldsymbol{z}_n + \boldsymbol{W}\boldsymbol{x}_n)$$

- Lets call the state of the hidden nodes $\boldsymbol{z}$ and they are connected accordig to a weighted adjacency matrix

$$\boldsymbol{z}_{n+1} = F(\boldsymbol{A}\boldsymbol{z}_n + \boldsymbol{W}\boldsymbol{x}_n)$$

- Echo state networks (Jaeger 2001) or liquid state machine (Maass et al. 2002)
  - A is a sparse random matrix
  - $\boldsymbol{z}$ is the reservoir state
  - There is an additional output layer

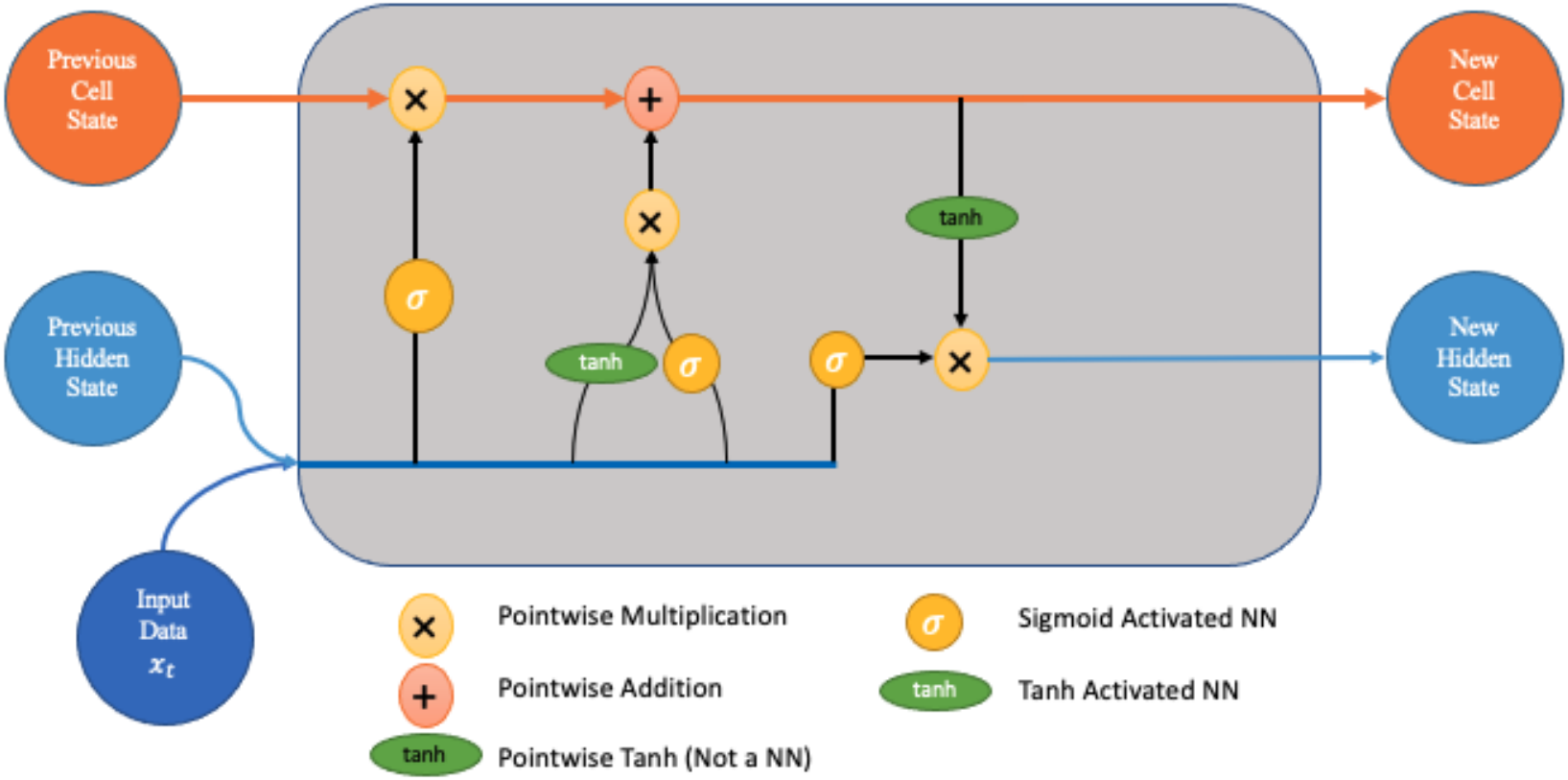
$$\boldsymbol{y}_n = F(\boldsymbol{W}^{out}\boldsymbol{z}_n)$$

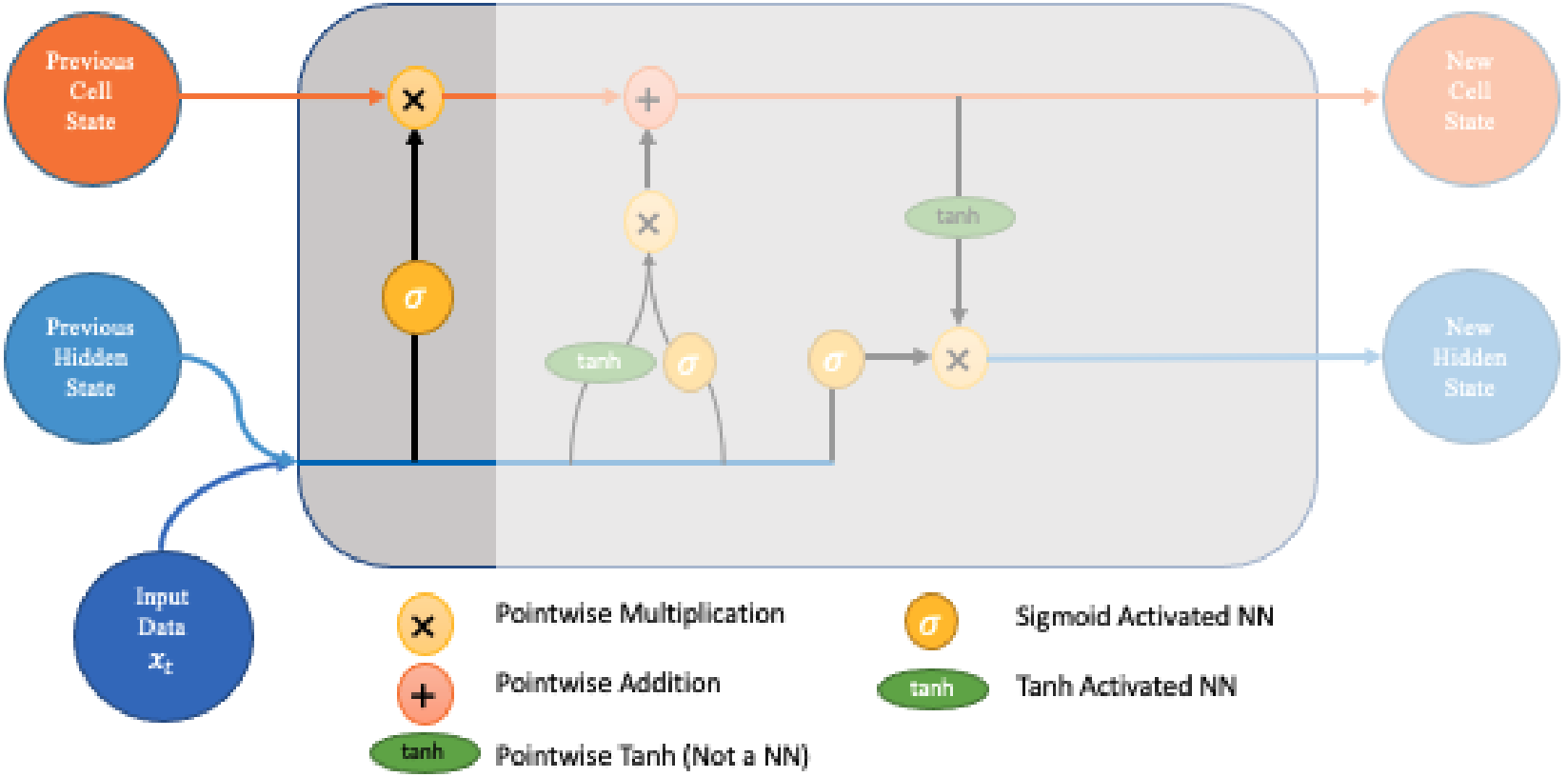
  - Only the weights of $\boldsymbol{W}^{out}$ are learned.
  - Echo state property: the reservoir will asymptotically "forget" all information from initial conditions.
  - Time series prediction: $y_n = x_{n+1}$. Open loop training, closed loop prediction.

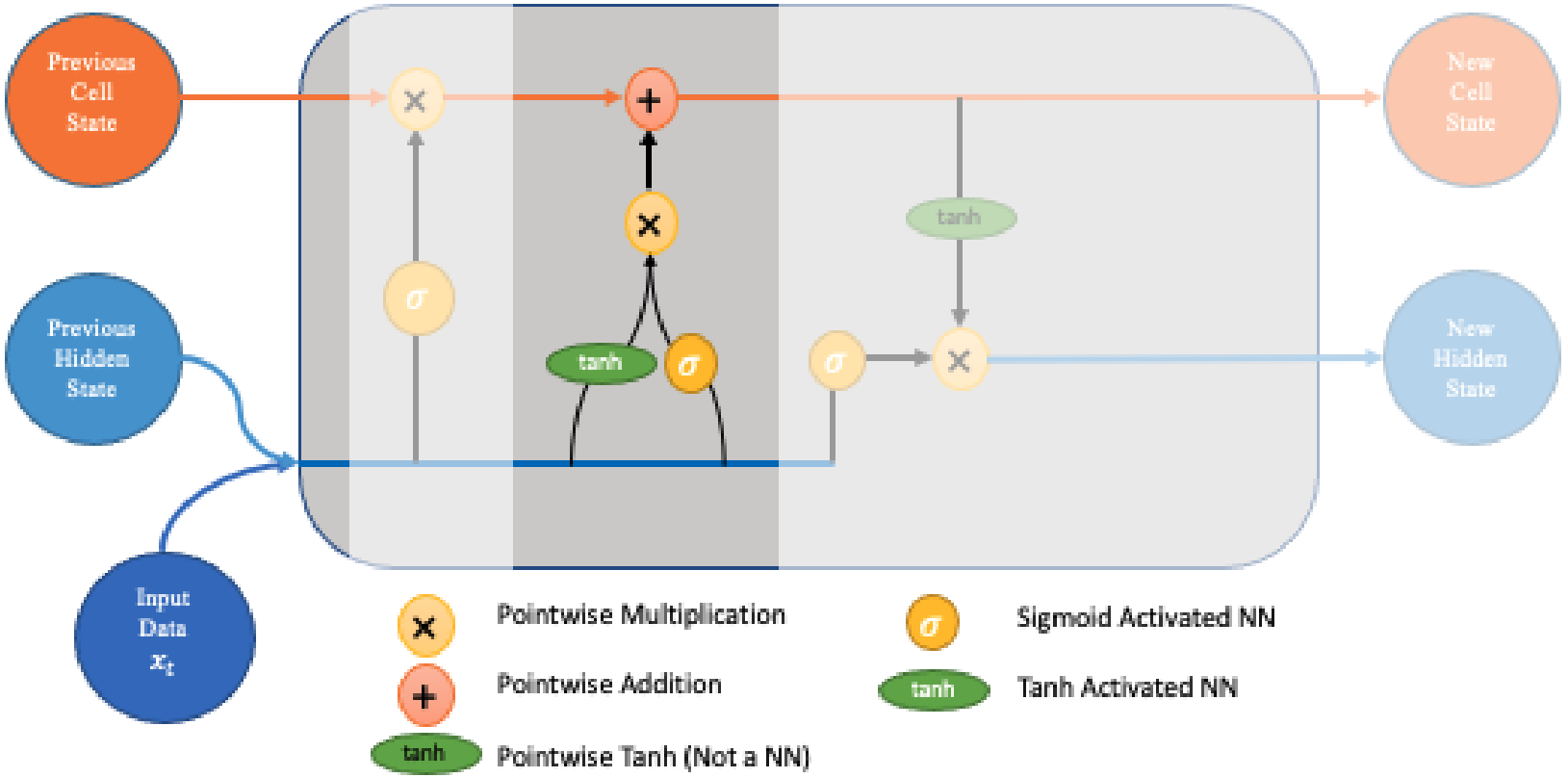
Long Short-term memory (LSTM)

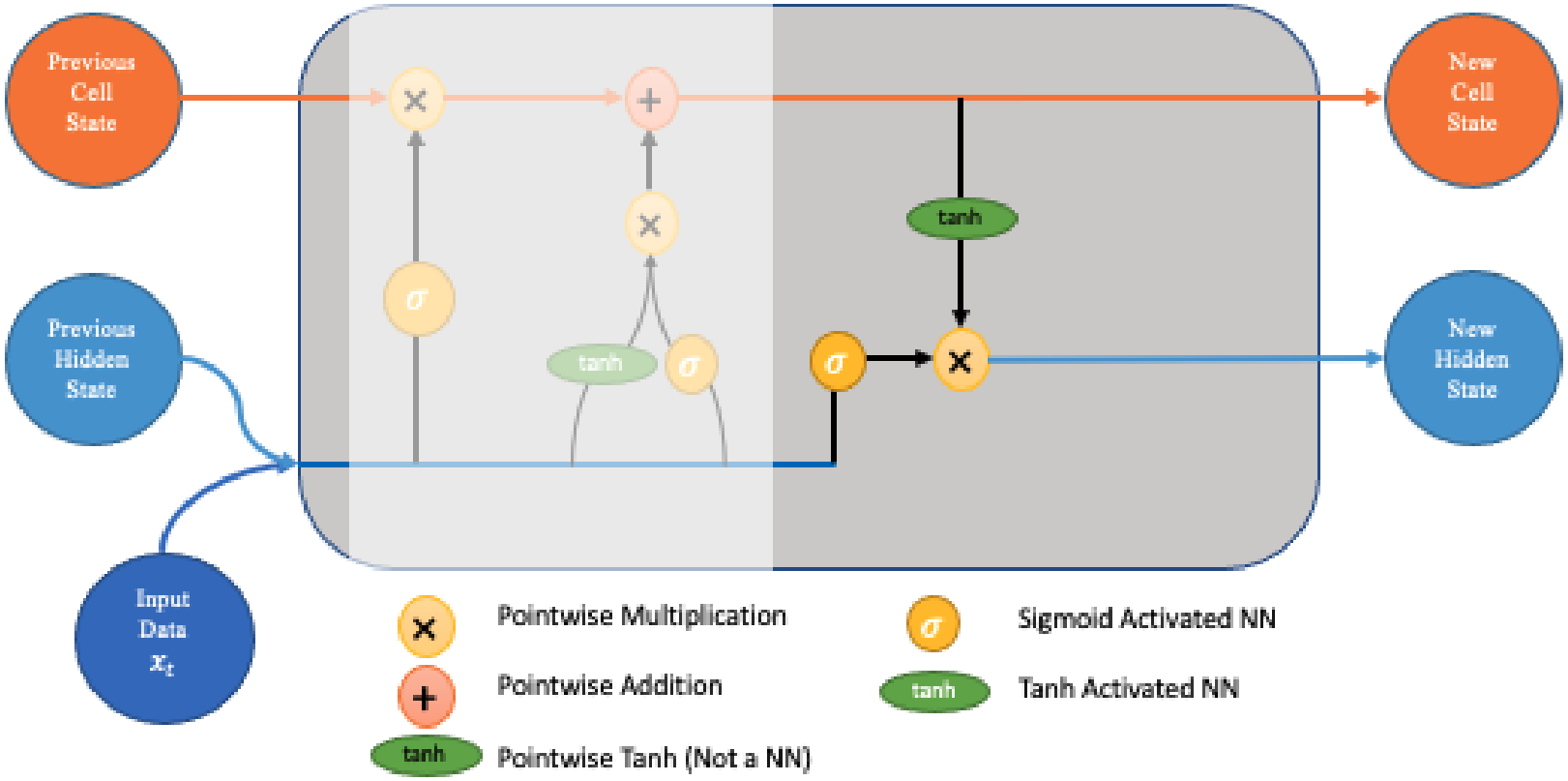
- Recurrent neural network (RNN) for processing sequences of data (text, speech, time series)
- Designed to overcome the "vanishing gradient problem" that is encountered when training RNNs with backpropagation
- Ingredients:
  - Cell state for long-term memory
  - Input, output an forget gate
  - How does it work?

Long Short-term memory (LSTM)




Previous Cell State
Previous Hidden State
Input Data x_t
New Cell State
New Hidden State
tanh
tanh
Pointwise Multiplication
Pointwise Addition
Pointwise Tanh (Not a NN)
Sigmoid Activated NN
Tanh Activated NN


Previous Cell State
Previous Hidden State
Input Data x_t
New Cell State
New Hidden State
tanh
σ
Pointwise Multiplication
Pointwise Addition
Pointwise Tanh (Not a NN)
Sigmoid Activated NN
Tanh Activated NN


Previous Cell State
Previous Hidden State
Input Data x_t
New Cell State
New Hidden State
tanh
tanh
Pointwise Multiplication
Pointwise Addition
Pointwise Tanh (Not a NN)
Sigmoid Activated NN
Tanh Activated NN

Long Short-term memory (LSTM)

- Recurrent neural network (RNN) for processing sequences of data (text, speech, time series)
- Designed to overcome the "vanishing gradient problem" that is encountered when training RNNs with backpropagation
- Ingredients:
  - Cell state for long-term memory
  - Input, output an forget gate
  - How does it work?
  - Time series predictions are created by a read-out layer on the hidden state similar to the echo state network

Software

- Linear time series analysis (spectral analysis, linear models) are available in all standard software packages, e.g. in matlab
- Nonlinear time series analysis
  - TISEAN - works with unix/linux and needs a FOTRAN compiler, difficult with Windows
  - Julia software library Dynamical Systems.jl
  - Java Information Dynamics Toolkit
  - Tigramite - Causal discovery for time series datasets - a phython package by Jakob Runge