Time Series and Sequence Learning

Lecture 11 – Course summary

Johan Alenlöv, Linköping University

2020-10-06

Summary of Lecture 10

Training RNNs

Learning an RNN amounts to minimizing a loss function, e.g.,

$$L(\theta) = \sum_{t=1}^{n} \left\{ y_t - \hat{y}_{t|t-1}(\theta) \right\}^2 .$$

Mini-batching not straightforward if the data consists of a **single, long time series** (large n).

[illegible]

Training RNNs

Possible approaches:

1. Do nothing
   - (Non-stochastic) Gradient descent using BPTT on the full data $\Rightarrow O(n)$ computations per gradient update.

Training RNNs

Possible approaches:

1. Do nothing
   - (Non-stochastic) Gradient descent using BPTT on the full data $\Rightarrow O(n)$ computations per gradient update.
2. Split the data into shorter sequences that are assumed to be independent
   - Deterministically — simple but can make unwanted boundary effects more pronounced
   - Randomly — smooths out boundary effects
   - With warm-up — let the hidden state warm up for a few time steps in each window to further mitigate boundary effects

Training RNNs

Possible approaches:

1. Do nothing
   - (Non-stochastic) Gradient descent using BPTT on the full data $\Rightarrow O(n)$ computations per gradient update.
2. Split the data into shorter sequences that are assumed to be independent
   - Deterministically — simple but can make unwanted boundary effects more pronounced
   - Randomly — smooths out boundary effects
   - With warm-up — let the hidden state warm up for a few time steps in each window to further mitigate boundary effects
3. Split the data with "statefulness" between sequences
   - Better respects the temporal dependencies between windows
   - Requires processing windows in order (non-randomly) which can result in systematic errors

RNNs for long-range dependencies

RNNs can capture **long-range temporal dependencies** by aggregating information in the hidden state.

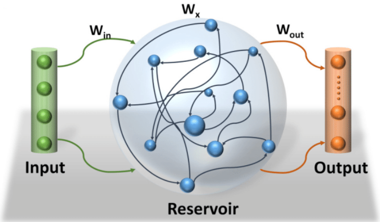
RNNs for long-range dependencies

RNNs can capture **long-range temporal dependencies** by aggregating information in the hidden state.

In practice: Challenging due to,

- Vanishing gradients (operating in a "stable regime")
- Exploding gradients (operating in an "unstable regime")

Echo State Networks



Adapted from DOI: 10.3389/fnins.2015.00502 under license CC4.0.

Echo State Networks:

- ▲ No learnable parameters in the dynamic part of the model $\Rightarrow$ no vanishing/exploding gradients!
- ▲ Extremely simple and fast to train
- ▼ Requires a large reservoir (high-dimensional $\mathbf{h}_t$) to be efficient.
- ▲ Can be used to initialize fully trainable RNNs.

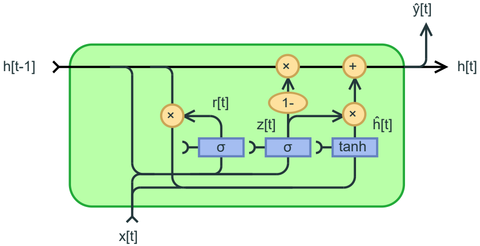
Gated RNNs

Gated RNNs: Allow the dynamic mapping $\mathbf{h}_t = H_\theta(\mathbf{h}_{t-1}, y_{t-1})$ to be

1. **learnable**, but
2. **carefully designed.**

Specifically, use **gating mechanisms** to enable gradients to propagate through time without vanishing or exploding.

ex) Gated Recurrent Unit

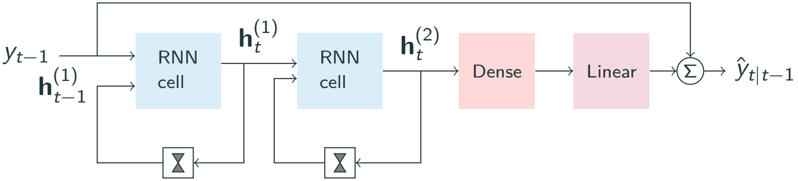


Stacked RNNs

We can build more complex (deep) models by stacking additional neural network blocks at each time step:

$$\mathbf{h}_t = H_{\theta}(\mathbf{h}_{t-1}, y_{t-1}),$$
$$\hat{y}_{t|t-1} = O_{\theta}(\mathbf{h}_t, y_{t-1}).$$

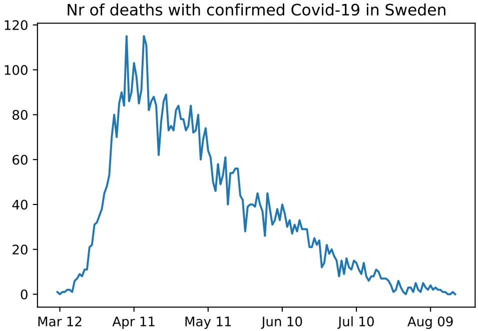
ex) Adding a second layer of RNN cells, and a densely connected layer for the output mapping



Summary of the course

What is a time series?

- Observations (data) are collected over time
- Observations are typically **temporally dependent**



Data from https://www.folkhalsomyndigheten.se/

Application domains

Time series data is everywhere — our world is inherently dynamic!

Application domains:

1. Climatology *(e.g., GMSL data from labs 1 & 2)*
2. Epidemiology *(e.g., covid data from lab 3)*
3. Astronomy *(e.g., sunspot data from lab 4)*
4. Econometrics
5. Audio signal processing
6. Robotics
7. ...

Prediction and inference

Why analyze time series data?

Prediction: By constructing **a model** we can predict (or **forecast**) future values.

ex) *How many persons in Östergötland will be infected with covid-19 one month from now?*

Prediction and inference

Why analyze time series data?

Prediction: By constructing **a model** we can predict (or **forecast**) future values.

ex) *How many persons in Östergötland will be infected with covid-19 one month from now?*

Inference: The model can help us to understand the underlying process that generates the data.

ex) *What is the basic reproduction number of covid-19?*

Time series analysis

Different approaches to time series analysis:

	Time domain	Frequency domain
Linear	**AR**, ARIMA **Linear SSM** Impulse response ...	Spectral analysis Transfer functions Wavelets ...
Nonlinear	**Nonlinear AR** **Nonlinear SSM** **RNN, TCN** ...	Volterra series Nonlinear freq resp Describing functions ...

Time series analysis

Categorization of **time domain** methods:

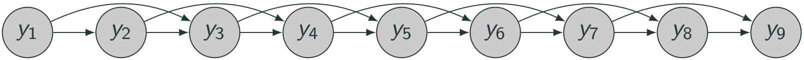
	Auto-regressive	State-space
Linear	Linear AR ARIMA	LGSS Structural TS
Nonlinear	Nonlinear AR TCN	Nonlinear SSM RNN

Auto-regressive models

Auto-regressive models: Current observation y_t depends on past data $y_{t-1}, \dots y_{t-p}$ through an **explicit functional relationship.**

- **Auto** — on itself
- **Regression** — regress the current value on the past

ex) AR(2):



Auto-regressive models

Can use **standard (linear or nonlinear) regression** models with input given by,

$$\phi_t = \begin{pmatrix} y_{t-1} & \dots & y_{t-p} \end{pmatrix}^\mathsf{T}$$

- **Linear AR(p)** $\iff$ linear regression,

$$\begin{aligned} y_t &= \theta^\mathsf{T} \phi_t + \varepsilon_t \\ &= a_1 y_{t-1} + a_2 y_{t-2} + \cdots + a_p y_{t-p} + \varepsilon_t. \end{aligned}$$

- **Nonlinear AR(p)** $\iff$ non-linear regression

$$y_t = f_\theta(\phi_t) + \varepsilon_t.$$

TCN

Temporal Convolutional Network

Temporal Convolutional Network: Construct the regression function f_θ using **one-dimensional, causal convolutions.**

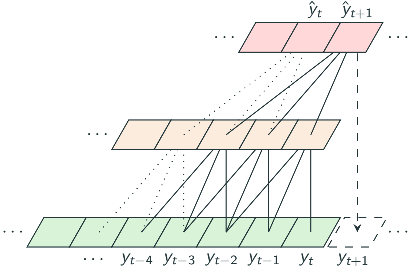
ex) 2-layer TCN:

$$h_t^{(0)} = y_t,$$

$$h_t^{(1)} = \sigma(W^{(1)} H_t^{(0)} + b^{(1)}),$$

$$y_{t+1} = W^{(2)} H_t^{(1)} + b^{(2)} + \varepsilon_{t+1},$$

with $\varepsilon_{t+1} \sim N(0, \sigma_\varepsilon^2)$.



Temporal Convolutional Network

Temporal Convolutional Network: Construct the regression function f_θ using **one-dimensional, causal convolutions.**

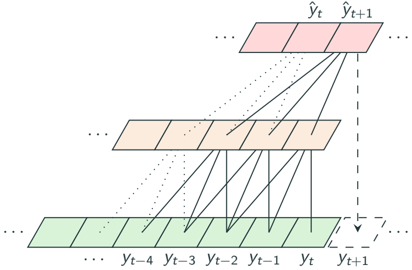
ex) 2-layer TCN:

$$h_t^{(0)} = y_t,$$

$$h_t^{(1)} = \sigma(W^{(1)} H_t^{(0)} + b^{(1)}),$$

$$y_{t+1} = W^{(2)} H_t^{(1)} + b^{(2)} + \varepsilon_{t+1},$$

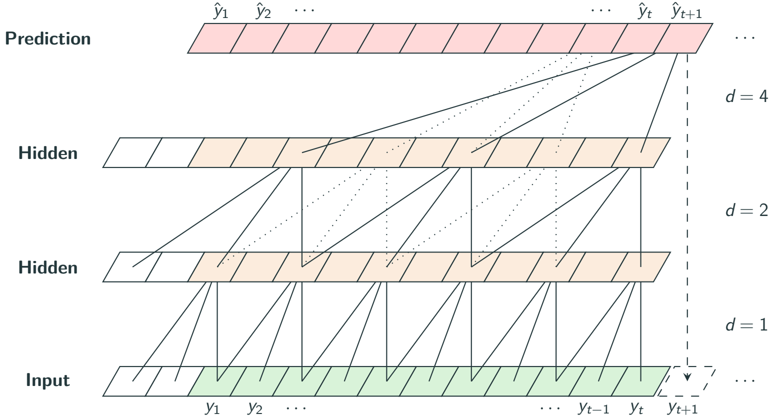
with $\varepsilon_{t+1} \sim N(0, \sigma_\varepsilon^2)$.



- Highly structured special case of a NAR(p) model.
- Large $p \implies$ large receptive field

TCN with dilated convolutions

By using **dilated convolutions** we can increase receptive field **exponentially** with depth.



Time series analysis

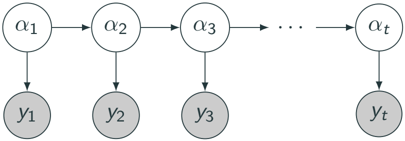
Categorization of **time domain** methods:

	Auto-regressive	State-space
Linear	Linear AR ARIMA	LGSS Structural TS
Nonlinear	Nonlinear AR TCN	Nonlinear SSM RNN

State-space models

State-space models: The observations y_t depends on an **unobserved** state-process α_t.

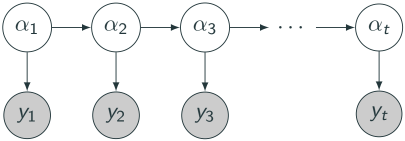
The state-process is a Markov chain and conditioned on the state-process the observations are independent.



State-space models

State-space models: The observations y_t depends on an **unobserved** state-process α_t.

The state-process is a Markov chain and conditioned on the state-process the observations are independent.



State inference: A key element is computing the **conditional distributions** of the states α_t conditioned on the observations $y_{1:s}$.

$t > s$: Prediction $\qquad$ $t = s$: Filtering $\qquad$ $t < s$: Smoothing

Linear and Gaussian state-space models

Def. A **Linear Gaussian State-Space (LGSS)** model is given by:

$$
\begin{aligned}
\alpha_t &= T\alpha_{t-1} + R\eta_t, & \eta_t &\sim \mathcal{N}(0, Q),\\
y_t &= Z\alpha_t + \varepsilon_t & \varepsilon_t &\sim \mathcal{N}(0, \sigma_\epsilon^2),
\end{aligned}
$$

and initial distribution $\alpha_1 \sim \mathcal{N}(a_1, P_1)$.

Linear and Gaussian state-space models

Def. A **Linear Gaussian State-Space (LGSS)** model is given by:

$$\begin{aligned} \alpha_t &= T\alpha_{t-1} + R\eta_t, & \eta_t &\sim \mathcal{N}(0, Q), \\ y_t &= Z\alpha_t + \varepsilon_t & \varepsilon_t &\sim \mathcal{N}(0, \sigma_\epsilon^2), \end{aligned}$$

and initial distribution $\alpha_1 \sim \mathcal{N}(a_1, P_1)$.

Thm. For an LGSS model, $p(\alpha_t \,|\, y_{1:s}) = \mathcal{N}(\alpha_t \,|\, \hat{\alpha}_{t|s}, P_{t|s})$.

- **Kalman filter** finds the filter and predictive distributions.
- **Kalman smoother** performs smoothing by an additional backward pass, given the predictive distributions.
- Easily handles **missing observations!**

Auto-regressive model in state space form

State space formulation of AR model: The AR(p) model,

$$y_t = \sum_{j=1}^{p} a_j y_{t-j} + \eta_t,$$

can equivalently be expressed in **state space form** as

$$\alpha_t = \begin{bmatrix} a_1 & a_2 & \cdots & a_{p-1} & a_p \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \alpha_{t-1} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \eta_t,$$

$$y_t = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \end{bmatrix} \alpha_t$$

A scalar AR(p) model can be written as a vector-valued AR(1) model!

Non-linear and/or non-Gaussian state-space models

Def. A **General State-Space** model is given by:

$$\alpha_t \,|\, \alpha_{t-1} \sim q(\alpha_t \,|\, \alpha_{t-1}),$$
$$y_t \,|\, \alpha_t \sim g(y_t \,|\, \alpha_t),$$

and initial distribution $\alpha_1 \sim q(\alpha_1)$.

Non-linear and/or non-Gaussian state-space models

Def. A **General State-Space** model is given by:

$$\alpha_t \,|\, \alpha_{t-1} \sim q(\alpha_t \,|\, \alpha_{t-1}),$$
$$y_t \,|\, \alpha_t \sim g(y_t \,|\, \alpha_t),$$

and initial distribution $\alpha_1 \sim q(\alpha_1)$.

In general there is **no exact solution**, we are forced to estimate.

- The **Bootstrap particle filter** estimates the **filter** and **predictor** distributions using a weighted sample.
- **Particles** mimic the behaviour of the hidden states α_t.
- **Weights** are used to correct for the observations.
- **Resampling** is necessary to avoid weight degeneracy.

Calculating the likelihood

The **log-likelihood** $\ell(y_{1:n}) = \log p(y_{1:n})$ can be calculated **exactly** using the Kalman filter for LGSS models, and **estimated** using the bootstrap particle filter for general SSMs.

- **Kalman filter:** F_t and $\hat{y}_{t\,|\,t-1}$ are given by the algorithm, and

$$\ell(y_{1:n}) = -\frac{n}{2}\log 2\pi - \frac{1}{2}\sum_{t=1}^{n}\left(\log F_t + \frac{(y_t - \hat{y}_{t\,|\,t-1})^2}{F_t}\right)$$

- **Bootstrap particle filter:** ω_t^i are the **unnormalized** weights given by the algorithm, and

$$\hat{\ell}(y_{1:n}) = \sum_{t=1}^{n}\log\left(\frac{1}{N}\sum_{i=1}^{N}\omega_t^i\right)$$

Expectation-Maximization

Expectation maximization: Algorithm for maximum likelihood learning of model parameters θ in models containing **latent random variables.**

Alternate two steps until convergence. Given current parameter θ_k:

- **E-step:** Calculate $\mathcal{Q}(\theta, \theta_k) = \mathbb{E}[\log p_\theta(\alpha_{1:n}, y_{1:n}) \mid y_{1:n}, \theta_k]$
- **M-step:** Set $\theta_{k+1} = \arg\max_\theta \mathcal{Q}(\theta, \theta_k)$.

Expectation-Maximization

Expectation maximization: Algorithm for maximum likelihood learning of model parameters θ in models containing **latent random variables.**

Alternate two steps until convergence. Given current parameter θ_k:

- **E-step:** Calculate $\mathcal{Q}(\theta, \theta_k) = \mathbb{E}[\log p_\theta(\alpha_{1:n}, y_{1:n}) \mid y_{1:n}, \theta_k]$
- **M-step:** Set $\theta_{k+1} = \arg\max_\theta \mathcal{Q}(\theta, \theta_k)$.

Properties:

- If the model beongs to the **exponential family** we only need the **smoothed sufficient statistics**.
- $\mathcal{Q}(\theta, \theta_k)$ requires the **smoothing distribution**.
- **Disturbance smoother** or **Kalman smoother** gives exact solution for linear Gaussian state-space models.
- For general state-space models we need **particle smoothers**.

Recurrent Neural Networks provide an alternative nonlinear generalization of the state space model,

$$\begin{aligned} \mathbf{h}_t &= H_\theta(\mathbf{h}_{t-1}, y_{t-1}), \\ y_t &= O_\theta(\mathbf{h}_t, y_{t-1}) + \nu_t, \qquad \nu_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_\nu^2). \end{aligned}$$

for arbitrary (parameterized) nonlinear functions H_θ and O_θ.

Recurrent Neural Networks

Recurrent Neural Networks provide an alternative nonlinear generalization of the state space model,

$$\begin{aligned} \mathbf{h}_t &= H_\theta(\mathbf{h}_{t-1}, y_{t-1}), \\ y_t &= O_\theta(\mathbf{h}_t, y_{t-1}) + \nu_t, \qquad \nu_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_\nu^2). \end{aligned}$$

for arbitrary (parameterized) nonlinear functions H_θ and O_θ.

- Modeling the state transition as a **conditionally deterministic** mapping removes the need for (approximate) **state inference**.

Recurrent Neural Networks

Recurrent Neural Networks provide an alternative nonlinear generalization of the state space model,

$$\begin{aligned} \mathbf{h}_t &= H_{\theta}(\mathbf{h}_{t-1}, y_{t-1}), \\ y_t &= O_{\theta}(\mathbf{h}_t, y_{t-1}) + \nu_t, \qquad \nu_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_\nu^2). \end{aligned}$$

for arbitrary (parameterized) nonlinear functions H_θ and O_θ.

- Modeling the state transition as a **conditionally deterministic** mapping removes the need for (approximate) **state inference**.
- Enables easier **gradient-based learning** of parameters by standard back-propagation (through time).

Black or gray?

The models that we have worked with can further be grouped into:

Black-box: Generic models that are learnt "entirely from data".

- AR, NAR, TCN
- Fully parameterized LGSS models
- RNN

Black or gray?

The models that we have worked with can further be grouped into:

Black-box: Generic models that are learnt "entirely from data".

- AR, NAR, TCN
- Fully parameterized LGSS models
- RNN

Gray-box: Domain-specific knowledge is used to define the **model structure**, but it still contains unknown and learnable parameters.

Black or gray?

The models that we have worked with can further be grouped into:

Black-box: Generic models that are learnt "entirely from data".

- AR, NAR, TCN
- Fully parameterized LGSS models
- RNN

Gray-box: Domain-specific knowledge is used to define the **model structure**, but it still contains unknown and learnable parameters.

Different shades of gray...

- General state space models, tailored to the application at hand (cf. SEIR model from lab 3)
- Structured state-space models (e.g., trend and seasonality)

Two final remarks

Try Simple Things First!

- Simple AR models and LGSS models can be very useful for time series prediction.

Two final remarks

Try Simple Things First!

- Simple AR models and LGSS models can be very useful for time series prediction.

Validate your model!

- If the data consists of a single time series, use the first part for training and the latter part for validation.

Thank you for attending the course!