Time Series and Sequence Learning

Lecture 9 – Recurrent Neural Networks

Johan Alenlöv, Linköping University

2021-09-28

Summary of Lecture 8

Summary of Lecture 8: Calculating the Log-Likelihood

- In the joint-smoothing distribution, the normalizing constant is the **Likelihood** of the model.
- When running the **particle filter** we are able to estimate this likelihood in the following way

$$L(y_{1:n}) \approx \prod_{t=1}^{n} \left(\frac{1}{N} \sum_{i=1}^{N} \omega_t^i \right).$$

- Or the **log-likelihood**

$$\ell(y_{1:n}) \approx \sum_{t=1}^{n} \left[\log \left(\sum_{i=1}^{N} \omega_t^i \right) - \log(N) \right]$$

- Note that ω_t^i should be the **unnormalized** weights!
- Typically the **log-likelihood** is calculated within the particle filter and updated each iteration.

Summary of Lecture 8: EM-Algorithm

- Assuming that the model belongs to the **exponential family** the EM-algorithm was reduced to
  - **E-step:** Calculate the **smoothed** sum of sufficient statistics,

$$\begin{aligned}
\mathbf{T}_1 &= \mathbb{E}[\mathbf{T}_q^1(\alpha_1) \mid y_{1:n}] && \text{Initial distribution} \\
\mathbf{T}_2 &= \sum_{t=2}^{n} \mathbb{E}[\mathbf{T}_q(\alpha_t, \alpha_{t-1}) \mid y_{1:n}] && \text{State transition} \\
\mathbf{T}_3 &= \sum_{t=1}^{n} \mathbb{E}[\mathbf{T}_g(\alpha_t, y_t) \mid y_{1:n}] && \text{Observation density}
\end{aligned}$$

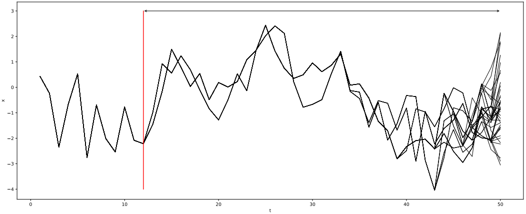
  - **M-step:** Maximize the expression,

$$\underbrace{\mathbf{n}_\mathbf{q}^1(\theta) \cdot \mathbf{T}_1 - A_q^1(\theta)}_{\text{Initial distribution}} + \underbrace{\mathbf{n}_\mathbf{q}(\theta) \cdot \mathbf{T}_2 - A_q(\theta)}_{\text{State transition}} + \underbrace{\mathbf{n}_\mathbf{g}(\theta) \cdot \mathbf{T}_3 - A_g(\theta)}_{\text{Observation density}}$$

- Requires the **smoothing** distribution

Summnary of Lecture 8: Fixed-Lag Smoothing

- Due to the resampling of the particle filter the trajectories **collapse**.



- Instead approximate using **fixed-lag smoothing**,

$$\mathbb{E}[h(\alpha_t) \mid y_{1:n}] \approx \mathbb{E}[h(\alpha_t) \mid y_{1:t+l}] \approx \sum_{i=1}^{N} \frac{\omega_{t+l}^{i}}{\Omega_{t+l}} h(\alpha_t^i)$$

- The lag l has to be set beforehand.

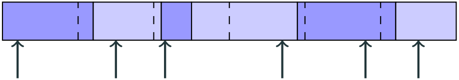
Summary of Lecture 8: Adaptive Resampling

- **Effective sample size** (ESS),

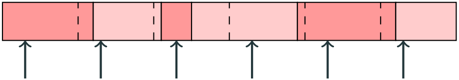
$$\mathrm{ESS}_t = \frac{\left(\sum_{i=1}^{N} \omega_t^i\right)^2}{\sum_{i=1}^{N} (\omega_t^i)^2},$$

  can be used to measure if resampling is necessary.
- If all weights are **equal** then $\mathrm{ESS} = N$.
- If all weights except one is zero then $\mathrm{ESS} = 1$.
- Set a **threshold** N_{ESS} and only resample if $\mathrm{ESS} < N_{\mathrm{ESS}}$.
- If no resampling happens the weights should be updated as in the SIS algorithm.

- In the basic algorithm **multinomial resampling** is used, (`np.random.choice`).
- There are many alternatives that can be used,
    - **Residual resampling:** We set the number of offspring for particle i to $\lfloor N\omega_t^i \rfloor$, then the final ones are set randomly.
    - **Stratified resampling:** Sample one value in each section independently,



    - **Systematic resampling:** Sample one value in each section using same offset,



Aim and outline

Aim:

- Show how Recurrent Neural Networks (RNNs) can be used for time series prediction.
- Provide a formal connection between SSMs and RNNs.

Outline:

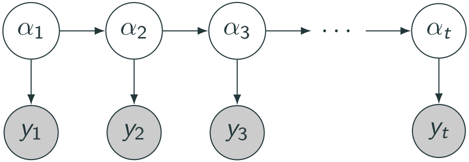
1. Linear Gaussian state space models revisited
   - State transformations
   - The innovation form
2. A nonlinear generalization — Recurrent Neural Networks
3. Training RNNs: Different approaches to mini-batching

Linear Gaussian state space models revisited

Linear state space models

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

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



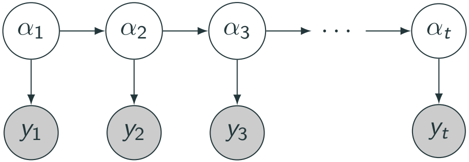
Limitation: The next state α_{t+1} as well as the observation y_t depend linearly on the current state α_t.

The model flexibility is limited.

Going nonlinear

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

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



Limitation: Filtering and smoothing distributions, as well as the one-step predictive pdf $p(y_t \mid y_{1:t-1})$, lack closed form expressions.

Learning and state inference becomes challenging.

Innovation form

Linear state space model:

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

Innovation form. There exists an **equivalent** representation,

$$\mathbf{h}_t = W\mathbf{h}_{t-1} + Uy_{t-1},$$
$$y_t = C\mathbf{h}_t + \nu_t, \qquad \nu_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_\nu^2).$$

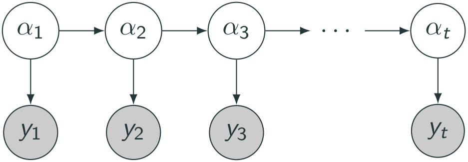
(Assuming stationarity for simplicity.)

Proof. Let $\mathbf{h}_t = \hat{\alpha}_{t|t-1}$, the Kalman predictive mean.

Innovation form

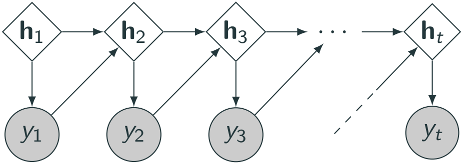
Original form:

$$\alpha_t = T\alpha_{t-1} + R\eta_t,$$
$$y_t = Z\alpha_t + \varepsilon_t.$$



Innovation form:

$$\mathbf{h}_t = W\mathbf{h}_{t-1} + Uy_{t-1},$$
$$y_t = C\mathbf{h}_t + \nu_t.$$



The hidden state variable $\mathbf{h}_t$ can be **deterministically and recursively computed** from the data.

Going nonlinear

Doesn't this look suspiciously similar to an MLP...?

$$\mathbf{h}_t = W\mathbf{h}_{t-1} + Uy_{t-1},$$
$$y_t = C\mathbf{h}_t + \nu_t,$$

for some **nonlinear activation function** $\sigma(\cdot)$.

This is a simple **Recurrent Neural Network (RNN).**

Referred to as a *Jordan–Elman network.*

Recurrent neural networks

Parameterized model

In the RNN we view the weight matrices and bias vectors as learnable parameters:

$$\mathbf{h}_t = \sigma(W\mathbf{h}_{t-1} + Uy_{t-1} + b),$$
$$y_t = C\mathbf{h}_t + c + \nu_t,$$

with $\theta = \{W, U, b, C, c\}$.

The parameters are the same for all time steps ("weight sharing").

Learning the parameters

We train the model by minimizing the negative log-likelihood,

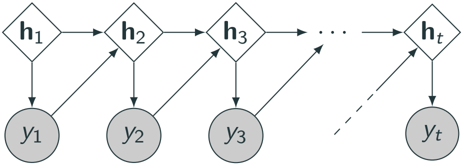
$$L(\theta) = -\sum_{t=1}^{n} \log p_\theta(y_t \mid y_{1:t-1}),$$

using gradient-based numerical optimization.

The fact that there is **no state noise** means that we can compute

$$p_\theta(y_t \mid y_{1:t-1}) = N(y_t \mid C\mathbf{h}_t + c, \sigma_\nu^2),$$

for all $t = 1, \ldots, n$ by a single forward pass through the model.



Back-propagation through time

The gradient of the loss function is given by

$$\nabla_\theta L(\theta) = -\sum_{t=1}^{n} \nabla_\theta \log p_\theta(y_t \mid y_{1:t-1}) = \sum_{t=1}^{n} \nabla_\theta \left\{ y_t - \hat{y}_{t|t-1}(\theta) \right\}^2$$

where

$$\begin{aligned}
\hat{y}_{t|t-1}(\theta) &= C\mathbf{h}_t + c \\
&= C\sigma(W\mathbf{h}_{t-1} + Uy_{t-1} + b) + c \\
&= C\sigma\left(W\sigma(W\mathbf{h}_{t-2} + Uy_{t-2} + b) + Uy_{t-1} + b\right) + c \\
&= \dots
\end{aligned}$$

This can be computed using the chain rule of differentiation, propagating information from $t = 1$ to $t = n$ and then back again.

$\Longrightarrow$ **Back-propagation through time.**

A (more) general RNN model

RNNs are not restricted to the simple networks discussed above.

A generalization of the Jordan-Elman network is,

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

for arbitrary (parameterized) nonlinear functions H_θ and O_θ.

- This is a nonlinear state-space model with output feedback and **without state noise.**
- As before, the one-step prediction can be computed by a forward propagation

$$p_\theta(y_t \mid y_{1:t-1}) = \mathcal{N}(y_t \mid O_\theta(\mathbf{h}_t, y_{t-1}), \sigma_\nu^2).$$

Residual connection in the output

Practical detail: In time series applications, the observations $\{y_t\}_{t \geq 0}$ are often **slowly varying** with time,

$$y_t \approx y_{t-1}.$$

Idea: Add an **explicit skip connection** in the output equation.

$$\mathbf{h}_t = H_\theta(\mathbf{h}_{t-1}, y_{t-1}),$$
$$y_t = y_{t-1} + O_\theta(\mathbf{h}_t, y_{t-1}) + \nu_t.$$

In practice, a simple way to accomplish this is to define $\tilde{y}_t = y_t - y_{t-1}$ as the target value used at time t.

Summary Lecture 9

- For a **state-space model** there are many representations giving the same distribution of the data.
- A special such case is the **innovation** form, where the **same noise** is used in both state and observation process.
- An **RNN** is a network on the innovation form.
    - The parameters are the same for all time steps, **weight sharing**.
    - Learn the parameters by minimizing the negative log-likelihood.
    - The gradient can be calculated using back-propagation through time.
- More general structure are possible.