Recurrent Neural Networks

Mario V. Wüthrich
RiskLab, ETH Zurich

RiskLab®
Switzerland

Block Course “Deep Learning with Actuarial Applications in R”
Swiss Association of Actuaries, Zurich
October 15/16, 2020

Programme SAV Block Course

- Refresher: Generalized Linear Models (THU 9:00-10:00)
- Feed-Forward Neural Networks (THU 12:30-14:00)
- Combined Actuarial Neural Network Models (THU 16:30-17:45)

- Recurrent Neural Networks (FRI 10:30-12:00)
- Discrimination-Free Insurance Pricing (FRI 14:30-15:00)
- Unsupervised Learning Methods (FRI 15:30-16:30)

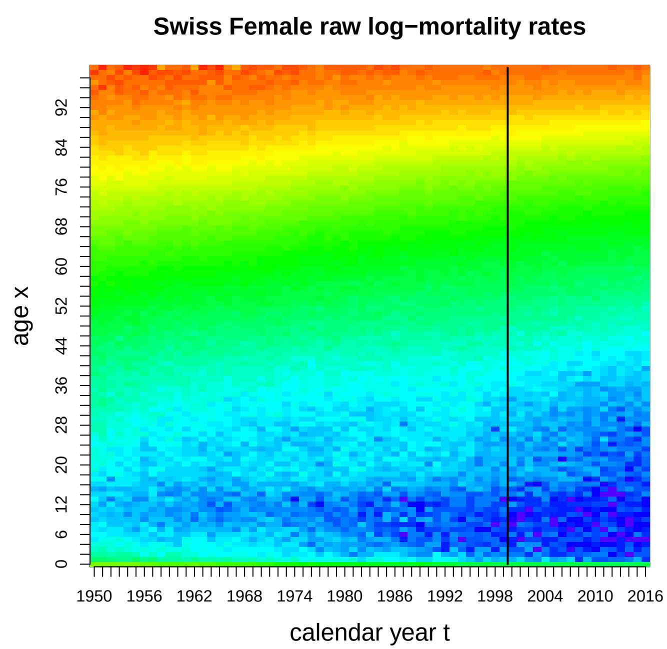
Contents: Recurrent Neural Networks

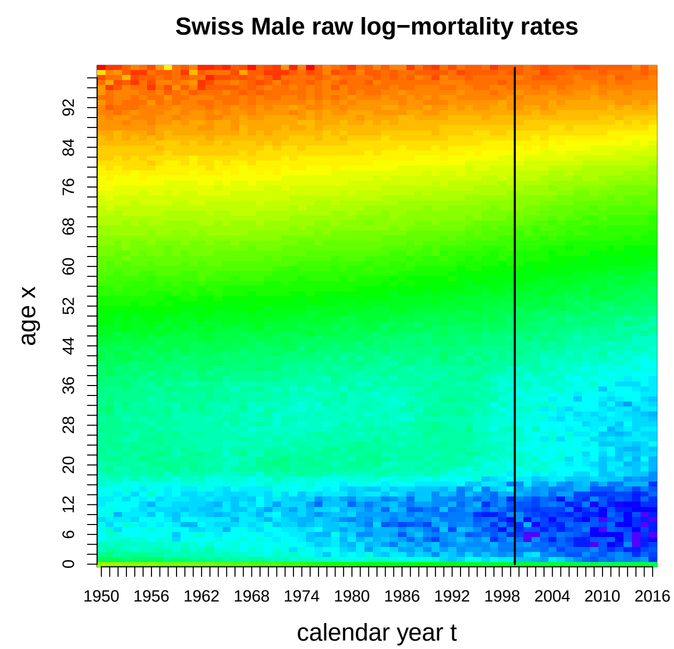
- Lee–Carter (LC) model
- Recurrent neural networks (RNNs)
- Long short-term memory (LSTM) networks
- Gated recurrent unit (GRU) networks
- Recurrent neural networks (RNNs) vs. convolutional neural networks (CNNs)

- **Lee–Carter (LC) Model and Time-Series**

Human Mortality Database (HMD)

```
Classes 'data.table' and 'data.frame':  13400 obs. of  7 variables:
 $ Gender      : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 1 1 1 1 ...
 $ Year        : int  1950 1950 1950 1950 1950 1950 1950 1950 1950 1950 ...
 $ Age         : int  0 1 2 3 4 5 6 7 8 9 ...
 $ Country     : chr  "CHE" "CHE" "CHE" "CHE" ...
 $ imputed_flag: chr  "FALSE" "FALSE" "FALSE" "FALSE" ...
 $ mx          : num  0.02729 0.00305 0.00167 0.00123 0.00101 ...
 $ logmx       : num  -3.6 -5.79 -6.39 -6.7 -6.9 ...
```



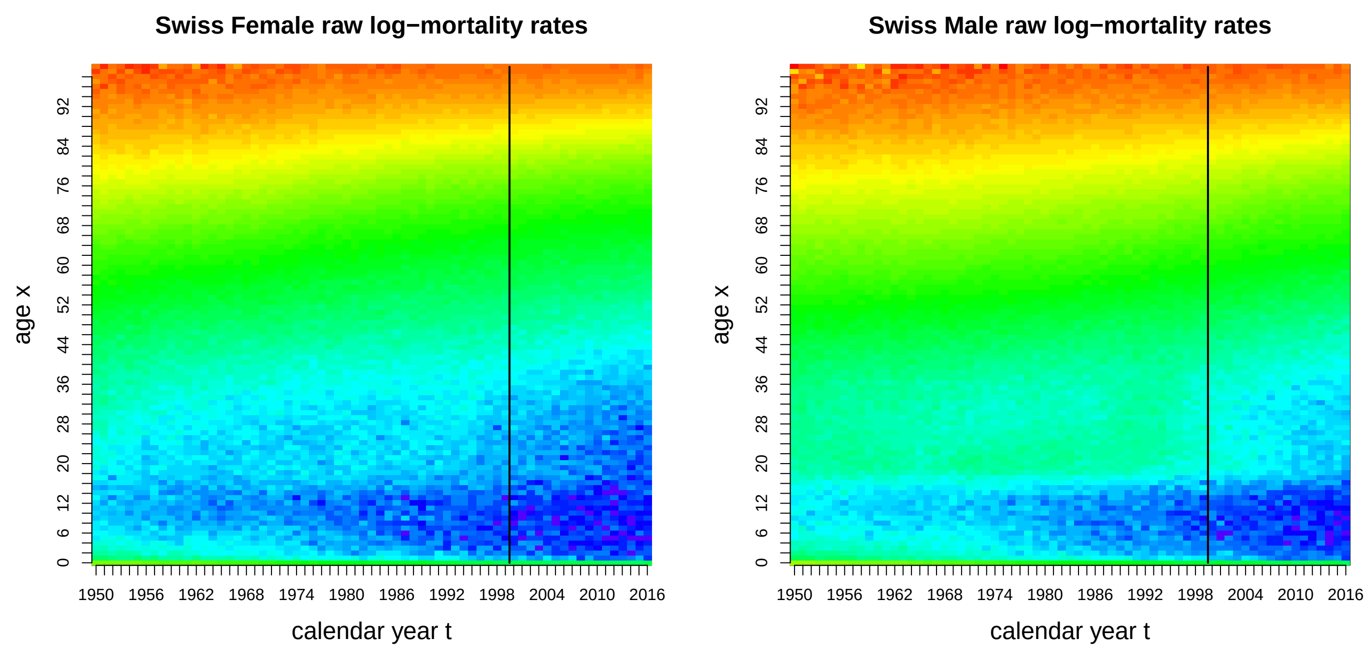


Human Mortality Database (HMD)

- **Aim:** Forecast mortality rates $m_{x,t}^{(i)}$ for ages x, calendar years t and populations i.

- Data available for ages $0 \le x \le 99$ and calendar years $1950 \le t \le 2016$ of 38 countries and 2 genders, i.e., $i = (r, g) \in \mathcal{I} = \mathcal{R} \times \{\texttt{female}, \texttt{male}\}$.

- Learning data $\mathcal{D} = \{1950 \le t \le 1999\}$; test data $\mathcal{T} = \{2000 \le t \le 2016\}$.



Lee–Carter (LC) Model (1992)

- Expected log-mortality rate is modeled by a regression function

$$(x,t,i) \ \mapsto \ \log(m_{x,t}^{(i)}) = a_x^{(i)} + b_x^{(i)} k_t^{(i)},$$

  - ⋆ $a_x^{(i)}$ average force of mortality at age x in population i;
  - ⋆ $k_t^{(i)}$ mortality trend in calendar year t of population i;
  - ⋆ $b_x^{(i)}$ mortality trend broken down to ages x of population i.

  ▷ The inputs (x,i) and (t,i) are treated as categorical variables.
  ▷ We have log-link, but not a GLM.

- 2-stage estimation and forecasting procedure, for each population i individually:

  1. Estimate $a_x^{(i)}$, $k_t^{(i)}$ and $b_x^{(i)}$ with singular value decomposition (SVD).
  2. Forecast by extrapolating estimated time series $(\widehat{k}_t^{(i)})_{t_0 \le t \le t_1}$ to years $t > t_1$.

Lee–Carter 2-Stage Forecasting

- Center the observed log-mortality rates $\log(M_{x,t}^{(i)})$

$$L_{x,t}^{(i)} = \log(M_{x,t}^{(i)}) - \frac{1}{|\mathcal{D}|} \sum_{s \in \mathcal{D}} \log(M_{x,s}^{(i)}).$$

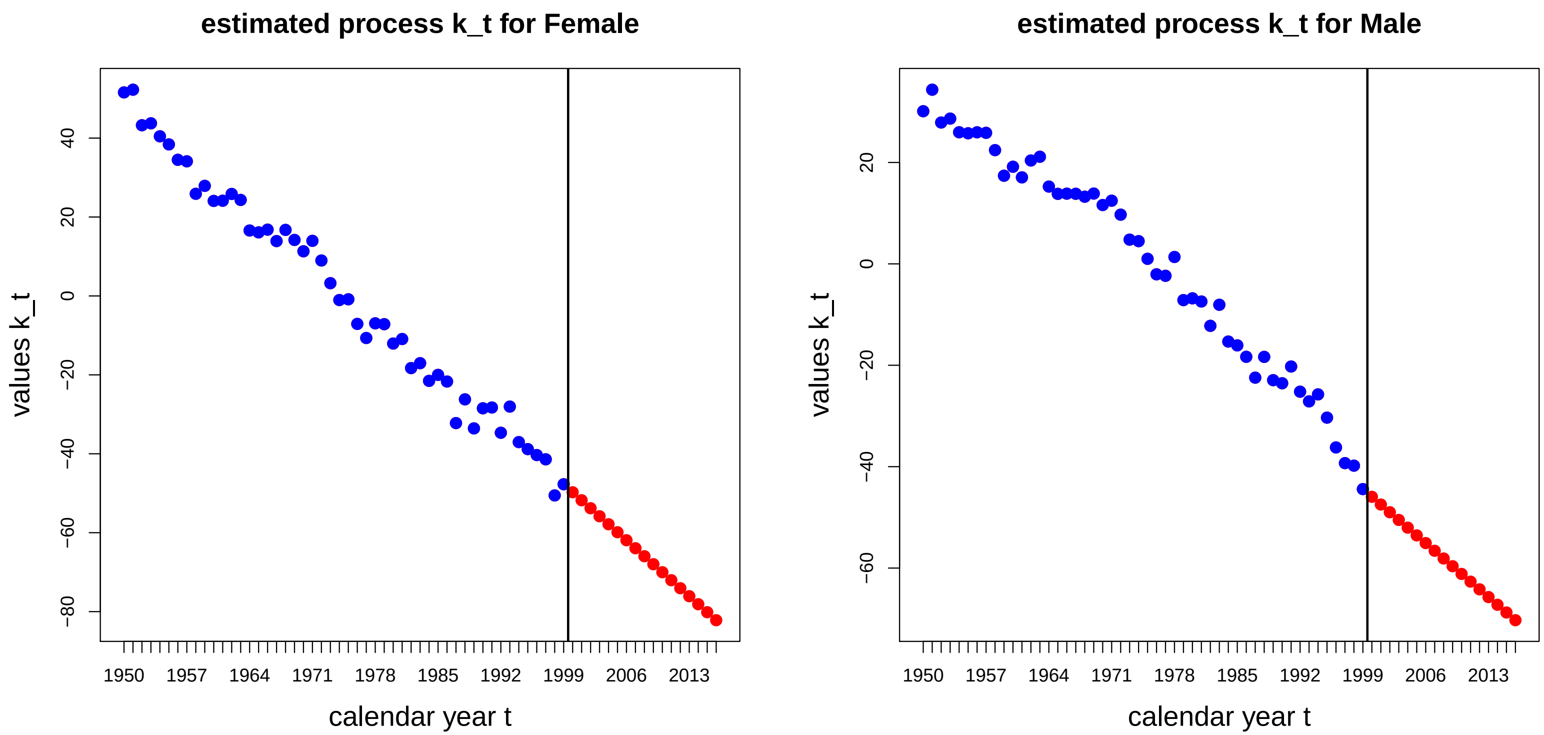
- Find optimal parameter values with SVD (see also PCA chapter)

$$\underset{(b_x^{(i)})_x, (k_t^{(i)})_t}{\arg\min} \sum_{t,x} \left(L_{x,t}^{(i)} - b_x^{(i)} k_t^{(i)} \right)^2,$$

under side constraint for identifiability $\sum_x \widehat{b}_x^{(i)} = 1$; and $\sum_{t \in \mathcal{D}} \widehat{k}_t^{(i)} = 0$.

- Extrapolate time series $(\widehat{k}_t^{(i)})_{t \in \mathcal{D}}$ using a random walk with drift.
- A random walk with drift often works surprisingly well.

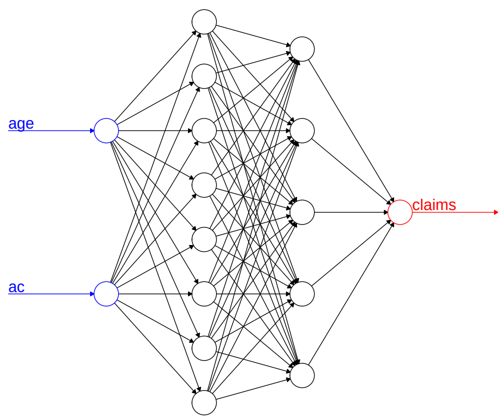
Lee–Carter Forecast for Switzerland



	in-sample MSE		out-of-sample MSE	
	female	male	female	male
LC model with SVD	3.7573	8.8110	0.6045	1.8152

- **Recurrent Neural Networks (RNNs)**

Recap: Feed-Forward Neural Networks (FNNs)



- Deep FNN mapping

$$\boldsymbol{x} \;\mapsto\; \mu = \mathbb{E}[Y] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) \right\rangle.$$

- **Goal:** Use time series input $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$ to predict output Y.
- The input of this FNN grows whenever we have a new observation $\boldsymbol{x}_t \in \mathbb{R}^{\tau_0}$.
- This FNN does **not** respect time series (causality) structure.

Plain-Vanilla Recurrent Neural Network (RNN)

- Define a recursive structure using a single RNN layer (upper index$^{(1)}$)

$$\boldsymbol{z}^{(1)} : \mathbb{R}^{\tau_0 \times \tau_1} \to \mathbb{R}^{\tau_1}, \qquad (\boldsymbol{x}_t, \boldsymbol{z}_{t-1}) \mapsto \boldsymbol{z}_t = \boldsymbol{z}^{(1)}(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}).$$

- The RNN layer is given by

$$\begin{aligned} \boldsymbol{z}_t &= \boldsymbol{z}^{(1)}(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}) \\ &= \left(\phi\left(\langle \boldsymbol{w}_1^{(1)}, \boldsymbol{x}_t\rangle + \langle \boldsymbol{u}_1^{(1)}, \boldsymbol{z}_{t-1}\rangle\right), \ldots, \phi\left(\langle \boldsymbol{w}_{\tau_1}^{(1)}, \boldsymbol{x}_t\rangle + \langle \boldsymbol{u}_{\tau_1}^{(1)}, \boldsymbol{z}_{t-1}\rangle\right)\right)^\top, \end{aligned}$$

where the individual neurons $1 \le j \le \tau_1$ are modeled by

$$\phi\left(\langle \boldsymbol{w}_j^{(1)}, \boldsymbol{x}_t\rangle + \langle \boldsymbol{u}_j^{(1)}, \boldsymbol{z}_{t-1}\rangle\right) = \phi\left(w_{j,0}^{(1)} + \sum_{l=1}^{\tau_0} w_{j,l}^{(1)} x_{t,l} + \sum_{l=1}^{\tau_1} u_{j,l}^{(1)} z_{t-1,l}\right).$$

- This RNN has one hidden layer with upper index$^{(1)}$ that is visited T times.

Remarks on RNNs

- Lower index t in $\boldsymbol{z}_t = \boldsymbol{z}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}\right)$ is time and upper index$^{(1)}$ is the hidden layer.

- This gives time series structure:

$$\cdots \quad \mapsto \quad \boldsymbol{z}_t = \boldsymbol{z}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}\right) \quad \mapsto \quad \boldsymbol{z}_{t+1} = \boldsymbol{z}^{(1)}\left(\boldsymbol{x}_{t+1}, \boldsymbol{z}_t\right) \quad \mapsto \quad \cdots$$

- Network weights $(\boldsymbol{w}_1^{(1)}, \ldots, \boldsymbol{w}_{\tau_1}^{(1)})^\top \in \mathbb{R}^{\tau_1 \times (\tau_0+1)}$ and $(\boldsymbol{u}_1^{(1)}, \ldots, \boldsymbol{u}_{\tau_1}^{(1)})^\top \in \mathbb{R}^{\tau_1 \times \tau_1}$ are time independent (are shared across every t-loop).

- We have an auto-regressive structure of order 1 in $(\boldsymbol{z}_t)_t$ summarizing the past history; this structure also resembles a state-space model.

- There are different ways in designing RNNs with multiple hidden layers. We give examples of two hidden layers, i.e. depth $d = 2$.

Variants with 2 Hidden RNN Layers

- 1st variant of a two-hidden layer RNN:

$$\begin{aligned} \boldsymbol{z}_t^{(1)} &= \boldsymbol{z}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}^{(1)}\right), \\ \boldsymbol{z}_t^{(2)} &= \boldsymbol{z}^{(2)}\left(\boldsymbol{z}_t^{(1)}, \boldsymbol{z}_{t-1}^{(2)}\right). \end{aligned}$$

- 2nd variant of a two-hidden layer RNN:

$$\begin{aligned} \boldsymbol{z}_t^{(1)} &= \boldsymbol{z}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}^{(1)}, \boldsymbol{z}_{t-1}^{(2)}\right), \\ \boldsymbol{z}_t^{(2)} &= \boldsymbol{z}^{(2)}\left(\boldsymbol{z}_t^{(1)}, \boldsymbol{z}_{t-1}^{(2)}\right). \end{aligned}$$

- 3rd variant of a two-hidden layer RNN:

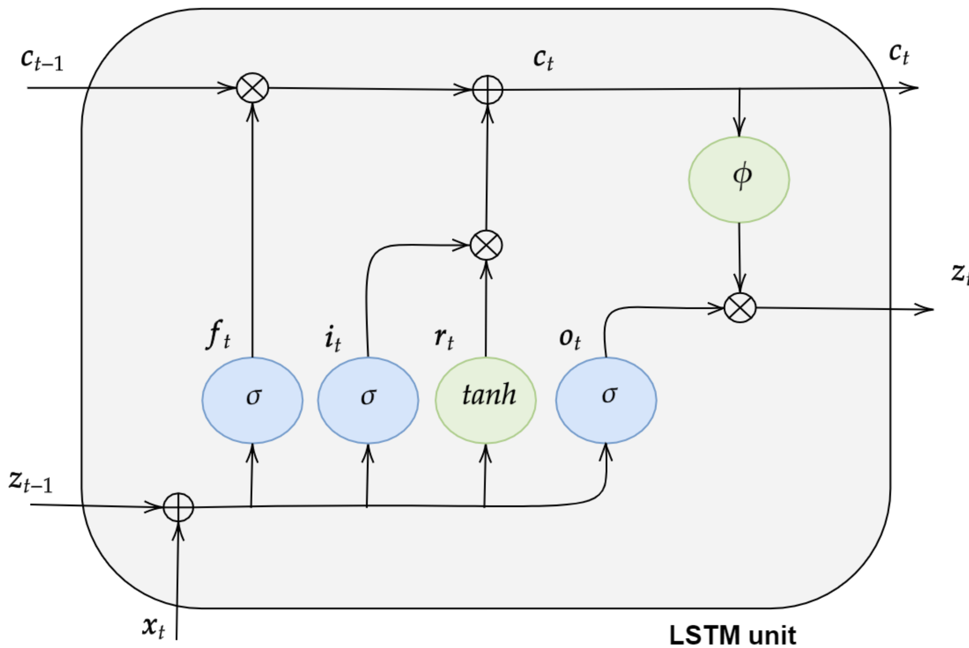
$$\begin{aligned} \boldsymbol{z}_t^{(1)} &= \boldsymbol{z}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}^{(1)}, \boldsymbol{z}_{t-1}^{(2)}\right), \\ \boldsymbol{z}_t^{(2)} &= \boldsymbol{z}^{(2)}\left(\boldsymbol{x}_t, \boldsymbol{z}_t^{(1)}, \boldsymbol{z}_{t-1}^{(2)}\right). \end{aligned}$$

- **Long Short-Term Memory (LSTM) Networks**

Long Short-Term Memory (LSTM) Networks

- The above plain-vanilla RNN architecture is of auto-regressive type of order 1.

- Long short-term memory (LSTM) networks were introduced by Hochreiter–Schmidhuber (1997): design a RNN architecture that can store information for "longer" by using a so-called memory cell c_t.



LSTM Layer: The 3 Gates

- **Forget Gate** (loss of memory rate):

$$\boldsymbol{f}_t = \boldsymbol{f}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}\right) = \phi_\sigma\left(\langle W_f, \boldsymbol{x}_t\rangle + \langle U_f, \boldsymbol{z}_{t-1}\rangle\right) \;\in\; (0,1)^{\tau_1}.$$

- **Input Gate** (memory update rate):

$$\boldsymbol{i}_t = \boldsymbol{i}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}\right) = \phi_\sigma\left(\langle W_i, \boldsymbol{x}_t\rangle + \langle U_i, \boldsymbol{z}_{t-1}\rangle\right) \;\in\; (0,1)^{\tau_1}.$$

- **Output Gate** (release of memory information rate):

$$\boldsymbol{o}_t = \boldsymbol{o}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}\right) = \phi_\sigma\left(\langle W_o, \boldsymbol{x}_t\rangle + \langle U_o, \boldsymbol{z}_{t-1}\rangle\right) \;\in\; (0,1)^{\tau_1}.$$

- Network weights are given by $W_f^\top, W_i^\top, W_o^\top \in \mathbb{R}^{\tau_1 \times (\tau_0+1)}$ (including an intercept), $U_f^\top, U_i^\top, U_o^\top \in \mathbb{R}^{\tau_1 \times \tau_1}$ (excluding an intercept).

LSTM Layer: The Memory Cell

- The above gates determine the release and update of the memory cell $\boldsymbol{c}_t$.

- The memory cell $(\boldsymbol{c}_t)_t$, called *cell state process*, is defined by

$$\begin{aligned} \boldsymbol{c}_t &= \boldsymbol{c}^{(1)}(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}, \boldsymbol{c}_{t-1}) \\ &= \boldsymbol{f}_t \otimes \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \otimes \phi_{\tanh}(\langle W_c, \boldsymbol{x}_t \rangle + \langle U_c, \boldsymbol{z}_{t-1} \rangle) \ \in \ \mathbb{R}^{\tau_1}, \end{aligned}$$

  for weights $W_c^\top \in \mathbb{R}^{\tau_1 \times (\tau_0+1)}$ (incl. intercept), $U_c^\top \in \mathbb{R}^{\tau_1 \times \tau_1}$ (excl. intercept), and Hadamard product $\otimes$ (element-wise product).

- Finally, define the updated neuron activation, given $\boldsymbol{c}_{t-1}$ and $\boldsymbol{z}_{t-1}$, by

$$\boldsymbol{z}_t = \boldsymbol{z}^{(1)}(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}, \boldsymbol{c}_{t-1}) = \boldsymbol{o}_t \otimes \phi(\boldsymbol{c}_t) \ \in \ \mathbb{R}^{\tau_1}.$$

- This is one LSTM layer indicated by the upper index$^{(1)}$.

Outputs and Time-Distributed Layers

- The LSTM produces a latent variable $\boldsymbol{z}_T$, based on time series input $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$.

- LSTM prediction: choose link function g and set

$$(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \ \mapsto \ \mu_{T+1} = \mathbb{E}\left[Y_{T+1}\right] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}_T \right\rangle.$$

- Network weights $W_f^\top, W_i^\top, W_o^\top, W_c^\top \in \mathbb{R}^{\tau_1 \times (\tau_0+1)}$, $U_f^\top, U_i^\top, U_o^\top, U_c^\top \in \mathbb{R}^{\tau_1 \times \tau_1}$ and $\boldsymbol{\beta} \in \mathbb{R}^{(\tau_1+1)\times \dim(Y_{T+1})}$. All time t-independent.

- The LSTM produces a latent time series $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T$. A so-called *time-distributed layer* outputs all of them such that we can fit

$$(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t) \ \mapsto \ \mu_{t+1} = \mathbb{E}\left[Y_{t+1}\right] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}_t \right\rangle,$$

using the same output filter $g^{-1}\langle \boldsymbol{\beta}, \cdot \rangle$ for all $t = 1, \ldots, T$.

- **Code LSTM Layers and Networks**

R Code for Single LSTM Layer Architecture

```
T     <- 10     # length of time series x_1,...,x_T
tau0 <- 3       # dimension of inputs x_t
tau1 <- 5       # dimension of the neurons z_t and cell states c_t

Input <- layer_input(shape=c(T,tau0), dtype='float32', name='Input')

Output = Input %>%
 layer_lstm(units=tau1,activation='tanh',recurrent_activation='tanh',name='LSTM1')%>%
 layer_dense(units=1, activation='exponential', name="Output")

model <- keras_model(inputs = list(Input), outputs = c(Output))
```

```
Layer (type)                     Output Shape                  Param #
======================================================================
Input (InputLayer)               (None, 10, 3)                 0
______________________________________________________________________
LSTM1 (LSTM)                     (None, 5)                     180
______________________________________________________________________
Output (Dense)                   (None, 1)                     6
======================================================================
Total params: 186
Trainable params: 186
Non-trainable params: 0
```

R Code for LSTM Time-Distribution

```
Output = Input %>%
  layer_lstm(units=tau1,activation='tanh',recurrent_activation='tanh',
                                          return_sequences=TRUE, name='LSTM1') %>%
  time_distributed( layer_dense(units=1,activation='exponential',
                                                    name="Output"), name='TD')
```

```
Layer (type)                    Output Shape                    Param #
======================================================================
Input (InputLayer)              (None, 10, 3)                   0
______________________________________________________________________
LSTM1 (LSTM)                    (None, 10, 5)                   180
______________________________________________________________________
TD (TimeDistributed)            (None, 10, 1)                   6
======================================================================
Total params: 186
Trainable params: 186
Non-trainable params: 0
```

R Code for Deep LSTMs

```
tau2 <- 4

Output = Input %>%
  layer_lstm(units=tau1,activation='tanh',recurrent_activation='tanh',
                                          return_sequences=TRUE, name='LSTM1') %>%
  layer_lstm(units=tau2,activation='tanh',recurrent_activation='tanh',name='LSTM2')%
  layer_dense(units=1, activation='exponential', name="Output")
```

```
Layer (type)                     Output Shape                  Param #
==========================================================================
Input (InputLayer)               (None, 10, 3)                 0
--------------------------------------------------------------------------
LSTM1 (LSTM)                     (None, 10, 5)                 180
--------------------------------------------------------------------------
LSTM2 (LSTM)                     (None, 4)                     160
--------------------------------------------------------------------------
Output (Dense)                   (None, 1)                     5
==========================================================================
Total params: 345
Trainable params: 345
Non-trainable params: 0
```

- **Gated Recurrent Unit (GRU) Networks**

Gated Recurrent Unit (GRU) Networks

- A shortcoming of LSTMs is their complexity.
- Gated recurrent unit (GRU) networks were introduced by Cho et al. (2014).
- They should share similar properties as LSTMs but based on less parameters.

GRU Layer

- **Reset gate**:

$$\boldsymbol{r}_t = \boldsymbol{r}^{(1)}(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}) = \phi_\sigma(\langle W_r, \boldsymbol{x}_t \rangle + \langle U_r, \boldsymbol{z}_{t-1} \rangle) \;\in\; (0,1)^{\tau_1}.$$

- **Update gate**:

$$\boldsymbol{u}_t = \boldsymbol{u}^{(1)}(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}) = \phi_\sigma(\langle W_u, \boldsymbol{x}_t \rangle + \langle U_u, \boldsymbol{z}_{t-1} \rangle) \;\in\; (0,1)^{\tau_1}.$$

- **Latent time series $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T$:**

$$\boldsymbol{z}_t = \boldsymbol{z}^{(1)}(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}) = \boldsymbol{r}_t \otimes \boldsymbol{z}_{t-1} + (\boldsymbol{1} - \boldsymbol{r}_t) \otimes \phi(\langle W_z, \boldsymbol{x}_t \rangle + \boldsymbol{u}_t \circ \langle U_z, \boldsymbol{z}_{t-1} \rangle) \in \mathbb{R}^{\tau_1},$$

  thus, we consider a credibility weighted average for the update of $\boldsymbol{z}_t$, this can also be understood as a skip connection.

- Network weights are given by $W_r^\top, W_u^\top, W_z^\top \in \mathbb{R}^{\tau_1 \times (\tau_0+1)}$ (including an intercept), $U_r^\top, U_u^\top, U_z^\top \in \mathbb{R}^{\tau_1 \times \tau_1}$ (excluding an intercept).

R Code for Single GRU Layer Architecture

```r
T     <- 10     # length of time series x_1,...,x_T
tau0 <- 3       # dimension of inputs x_t
tau1 <- 5       # dimension of the neurons z_t and cell states c_t

Input <- layer_input(shape=c(T,tau0), dtype='float32', name='Input')

Output = Input %>%
 layer_gru(units=tau1,activation='tanh',recurrent_activation='tanh',name='GRU1')%>%
 layer_dense(units=1, activation='exponential', name="Output")

model <- keras_model(inputs = list(Input), outputs = c(Output))
```

```
Layer (type)                     Output Shape                  Param #
======================================================================
Input (InputLayer)               (None, 10, 3)                 0
----------------------------------------------------------------------
GRU1 (GRU)                       (None, 5)                     135
----------------------------------------------------------------------
Output (Dense)                   (None, 1)                     6
======================================================================
Total params: 141
Trainable params: 141
Non-trainable params: 0
```

- **Example: Mortality Modeling**

Toy Example of LSTMs and GRUs

- Consider raw Swiss female log-mortality rates $\log(M_{x,t})$ for calendar years $1990, \ldots, 2001$ and ages $0 \leq x \leq 99$.

- Set $T = 10$ and $\tau_0 = 3$. Define for ages $1 \leq x \leq 98$ and years $1 \leq t \leq T$ features

$$\boldsymbol{x}_{x,t} = \Big(\log(M_{x-1,1999-(T-t)}), \log(M_{x,1999-(T-t)}), \log(M_{x+1,1999-(T-t)})\Big)^\top \in \mathbb{R}^{\tau_0},$$

and observations

$$Y_{x,T+1} = \log(M_{x,2000}) \;=\; \log(M_{x,1999-(T-(T+1))}).$$

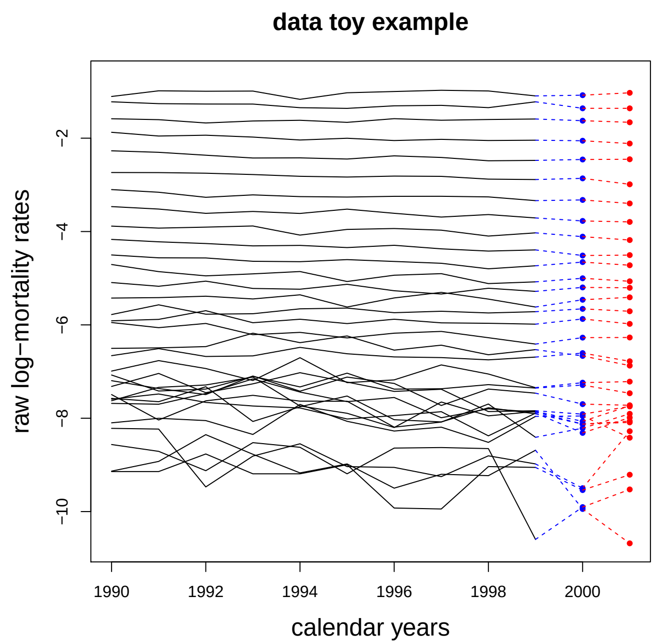
- Based on these definitions, choose training data

$$\mathcal{D} = \{(\boldsymbol{x}_{x,1}, \ldots, \boldsymbol{x}_{x,T}; Y_{x,T+1});\ 1 \leq x \leq 98\}.$$

Thus, we have 98 training samples.

Toy Example of LSTMs and GRUs

- Consider ages $(x-1, x, x+1)$ simultaneously in $\boldsymbol{x}_{x,t}$ to smooth inputs over neighboring ages to predict the central mortality rate $Y_{x,T+1}$.



⋆ black lines: explanatory variables $(\boldsymbol{x}_{x,t})_{1 \le t \le T}$ (input data)

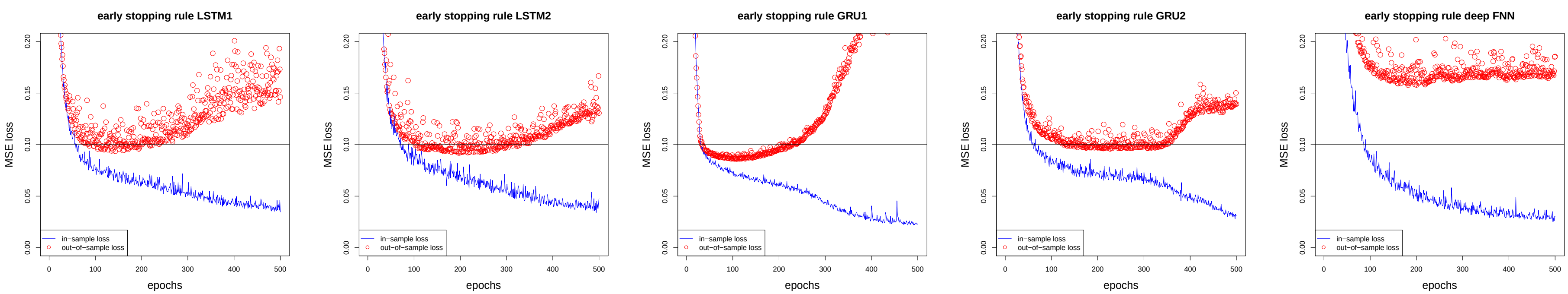
⋆ blue dots: response variables $Y_{x,T+1}$ (for training)

⋆ test data $\mathcal{T} = \{(\boldsymbol{x}_{x,2}, \ldots, \boldsymbol{x}_{x,T+1}; Y_{x,T+2});\ 1 \le x \le 98\}$ (shifted data); or alternatively $\mathcal{T}_+ = \{(\boldsymbol{x}_{x,1}, \ldots, \boldsymbol{x}_{x,T+1}; Y_{x,T+2});\ 1 \le x \le 98\}$ (expanded data)

Toy Example of LSTMs and GRUs

- Pre-process all variables $x_{x,t}$ with MinMaxScaler to domain $[-1, 1]$.

- Use shallow LSTM1, deep LSTM2 as above, and corresponding GRU1, GRU2, and deep FNN; GDM: blue is in-sample, red is out-of-sample



	# param.	epochs	run time	in-sample loss	out-of-sample loss
LSTM1	186	150	8 sec	0.0655	0.0936
LSTM2	345	200	15 sec	0.0603	0.0918
GRU1	141	100	5 sec	0.0671	0.0860
GRU2	260	200	14 sec	0.0651	0.0958
deep FNN	184	200	5 sec	0.0485	0.1577

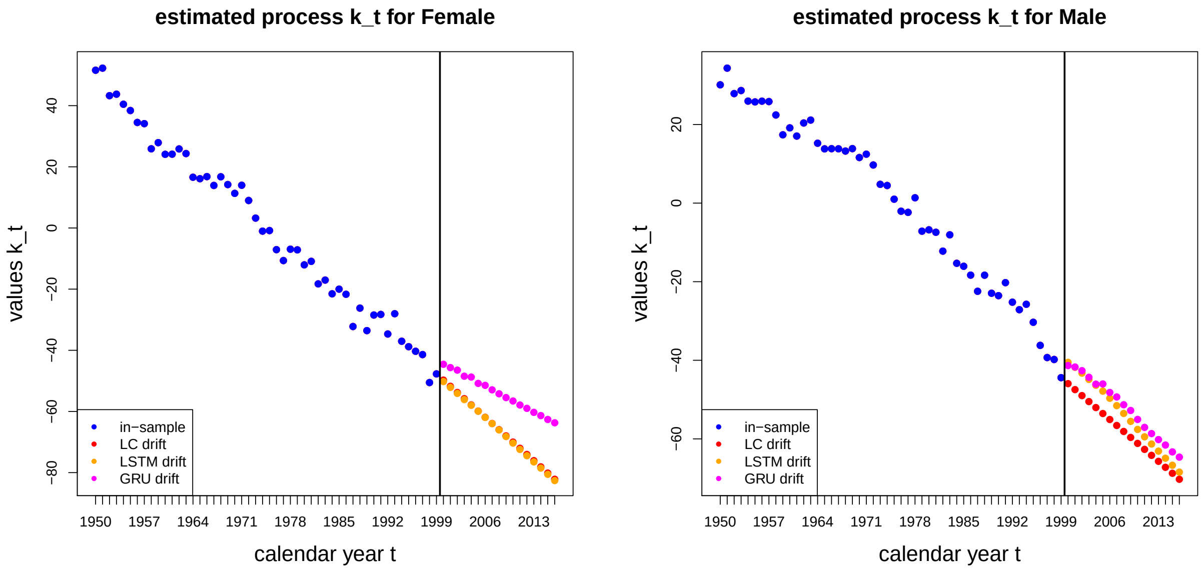
- In general: LSTMs seem more robust than GRUs.

Hyper-Parameters in LSTMs

	# param.	epochs	run time	in-sample	out-of-sample
base case:					
LSTM1 ($T = 10$, $\tau_0 = 3$, $\tau_1 = 5$)	186	150	8 sec	0.0655	0.0936
LSTM1 ($T = 10$, $\tau_0 = 1$, $\tau_1 = 5$)	146	100	5 sec	0.0681	0.0994
LSTM1 ($T = 10$, $\tau_0 = 5$, $\tau_1 = 5$)	226	150	15 sec	0.0572	0.0795
LSTM1 ($T = 5$, $\tau_0 = 3$, $\tau_1 = 5$)	186	100	4 sec	0.0753	0.1028
LSTM1 ($T = 20$, $\tau_0 = 3$, $\tau_1 = 5$)	186	200	16 sec	0.0678	0.0914
LSTM1 ($T = 10$, $\tau_0 = 3$, $\tau_1 = 3$)	88	200	10 sec	0.0614	0.1077
LSTM1 ($T = 10$, $\tau_0 = 3$, $\tau_1 = 10$)	571	100	5 sec	0.0667	0.0962

Application to Swiss Mortality Data

	in-sample		out-of-sample		run times	
	female	male	female	male	female	male
LSTM3 ($T = 10$, $(\tau_0, \tau_1, \tau_2, \tau_3) = (5, 20, 15, 10)$)	2.5222	6.9458	0.3566	1.3507	225s	203s
GRU3 ($T = 10$, $(\tau_0, \tau_1, \tau_2, \tau_3) = (5, 20, 15, 10)$)	2.8370	7.0907	0.4788	1.2435	185s	198s
LC model with SVD	3.7573	8.8110	0.6045	1.8152	–	–

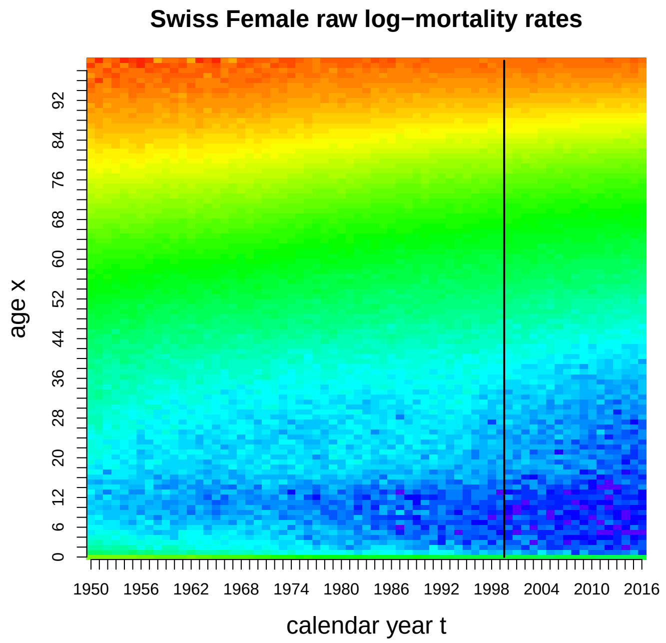


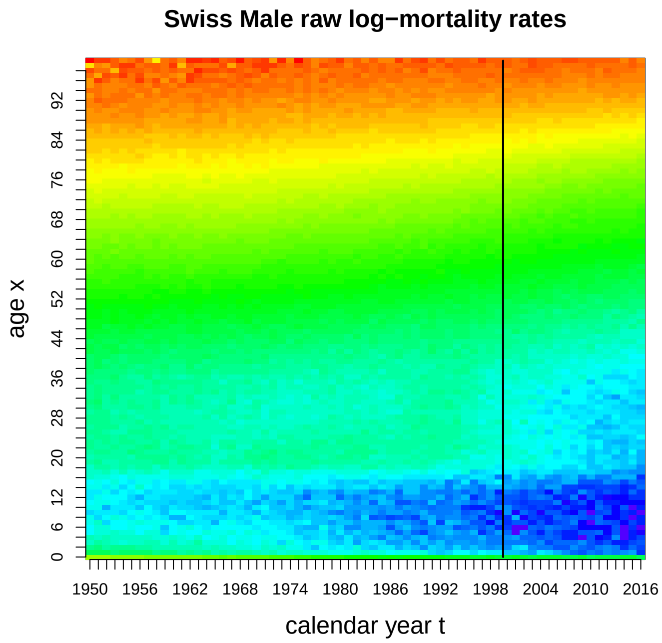
- Main difficulty: Robustness of results.
- More stability by simultaneous multi-population modeling.
- For more sophisticated models see Perla et al. (2020).

- **RNNs vs. Convolutional Neural Networks (CNNs)**

RNNs vs. Convolutional Neural Networks (CNNs)

- RNNs respect time series structures.
- Convolutional neural networks (CNNs) respect local spatial structure.
- Intuitively, for CNNs we move small windows (filters) over the images to discover similar structure at different locations in the images.





Convolutional Neural Networks (CNNs)

- CNNs have been introduced in Fukushima (1980).
- CNNs used for image and speech recognition, natural language processing (NLP), and in many other fields, for references see our tutorial Meier–Wüthrich (2020).
- Main feature of CNNs is translation invariance, see Wiatowski–Bölcskei (2018).

Convolutional Layer: Sketch of Structure

A convolution layer (we consider a two-dimensional image here and a single filter)

$$\boldsymbol{z}^{(m)}: \mathbb{R}^{n_1^{(m-1)} \times n_2^{(m-1)}} \to \mathbb{R}^{n_1^{(m)} \times n_2^{(m)}}, \quad \boldsymbol{x} \mapsto \begin{pmatrix} z_{1,1}^{(m)}(\boldsymbol{x}) & \cdots & z_{1,n_2^{(m)}}^{(m)}(\boldsymbol{x}) \\ \vdots & & \vdots \\ z_{n_1^{(m)},1}^{(m)}(\boldsymbol{x}) & \cdots & z_{n_1^{(m)},n_2^{(m)}}^{(m)}(\boldsymbol{x}) \end{pmatrix},$$

with (local) filter/window having filter size $f_1^{(m)}$ and $f_2^{(m)}$

$$\boldsymbol{x} \mapsto z_{i_1,i_2}^{(m)}(\boldsymbol{x}) = \phi\left(w_{0,0}^{(m)} + \sum_{j_1=1}^{f_1^{(m)}} \sum_{j_2=1}^{f_2^{(m)}} w_{j_1,j_2}^{(m)} x_{i_1+j_1-1,i_2+j_2-1} \right).$$

- ⋆ In our tutorial we add activation ϕ only later (after batch normalization).
- ⋆ We illustrate a single filter, multiple filters are used to extract different features.
- ⋆ Multiple filters require three-dimensional inputs in deep CNNs.
- ⋆ Pooling layers, flatten layers and so-called padding is used.

References

- Cho, van Merrienboer, Gulcehre, Bahdanau, Bougares, Schwenk, Bengio (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv*:1406.1078.
- Efron, Hastie (2016). Computer Age Statistical Inference: Algorithms, Evidence, and Data Science. Cambridge UP.
- Fukushima (1980). Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36/4, 193-202.
- Goodfellow, Bengio, Courville (2016). Deep Learning. MIT Press.
- Hastie, Tibshirani, Friedman (2009). The Elements of Statistical Learning. Springer.
- Hochreiter, Schmidhuber (1997). Long short-term memory. Neural Computation 9/8, 1735-1780.
- Lee, Carter (1992). Modeling and forecasting US mortality. Journal American Statistical Association 87/419, 659-671.
- Meier, Wüthrich (2020). Convolutional neural network case studies: (1) anomalies in mortality rates (2) image recognition. SSRN 3656210.
- Perla, Richman, Scognamiglio, Wüthrich (2020). Time-series forecasting of mortality rates using deep learning. SSRN 3595426.
- Richman, Wüthrich (2019). Lee and Carter go machine learning. SSRN 3441030.
- Richman, Wüthrich (2020). A neural network extension of the Lee–Carter model to multiple populations. Annals of Actuarial Science.
- Smyl (2019). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. International Journal of Forecasting.
- Wiatowski, Bölcskei (2018). A mathematical theory of deep convolutional neural networks for feature extraction. IEEE Transactions on Information Theory 64/3, 1845-1866.
- Wilmoth, Shkolnikov (2010). Human Mortality Database. University of California.
- Wüthrich, Buser (2016). Data Analytics for Non-Life Insurance Pricing. SSRN 2870308, Version September 10, 2020.