Chapter 8
Recurrent Neural Networks

Chapter 7 has discussed fully-connected *feed-forward neural* (FN) networks. Feed-forward means that information is passed in a directed acyclic path from the input layer to the output layer. A natural extension is to allow these networks to have cycles. In that case, we call the architecture a *recurrent neural* (RN) network. A RN network architecture is particularly useful for time-series modeling. The discussion on time-series data also links to Sect. 5.8.1 on longitudinal and panel data. RN networks have been introduced in the 1980s, and the two most popular RN network architectures are the long short-term memory (LSTM) architecture proposed by Hochreiter–Schmidhuber [188] and the gated recurrent unit (GRU) architecture introduced by Cho et al. [76]. These two architectures will be described in detail in this chapter.

8.1 Motivation for Recurrent Neural Networks

We start from a deep FN network providing the regression function, see (7.2)–(7.3),

$$\boldsymbol{x} \;\mapsto\; \mu(\boldsymbol{x}) = g^{-1}\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x})\rangle, \tag{8.1}$$

with a composition $\boldsymbol{z}^{(d:1)}$ of d FN layers $\boldsymbol{z}^{(m)}$, $1 \le m \le d$, link function g and with output parameter $\boldsymbol{\beta} \in \mathbb{R}^{q_d+1}$. In principle, we could directly use this FN network architecture for time-series forecasting. We explain here why this is not the best option to deal with time-series data.

Assume we want to predict a random variable Y_{T+1} at time $T \ge 0$ based on the time-series information $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$. This information is assumed to be available at time T for predicting the response Y_{T+1}. The past response information Y_t, $1 \le$

M. V. Wüthrich, M. Merz, *Statistical Foundations of Actuarial Learning and its Applications*, Springer Actuarial, https://doi.org/10.1007/978-3-031-12409-9_8

$t \leq T$, is typically included in $\boldsymbol{x}_t$.[1] Using the above FN network architecture we could directly try to predict Y_{T+1}, based on this past information. Therefore, we define the feature information $\boldsymbol{x}_{0:T} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T)$ and we aim at designing a FN network (8.1) for modeling

$$\boldsymbol{x}_{0:T} \ \mapsto \ \mu_T(\boldsymbol{x}_{0:T}) = \mathbb{E}[Y_{T+1}|\boldsymbol{x}_{0:T}] = \mathbb{E}[Y_{T+1}|\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T].$$

In principle we could work with such an approach, however, it has a couple of severe drawbacks. Obviously, the length of the feature vector $\boldsymbol{x}_{0:T}$ depends on time T, that is, it will grow with every time step. Therefore, the regression function (network architecture) $\boldsymbol{x}_{0:T} \mapsto \mu_T(\boldsymbol{x}_{0:T})$ is time-dependent. Consequently, with this approach we have to fit a network for every T. This deficiency can be circumvented if we assume a Markov property that does not require of carrying forward the whole past history. Assume that it is sufficient to consider a history of a certain length. Choose $\tau \geq 0$ fixed, then, for $T \geq \tau$, we can set for the feature information $\boldsymbol{x}_{T-\tau:T} = (\boldsymbol{x}_{T-\tau}, \ldots, \boldsymbol{x}_T)$, which has a fixed length $\tau + 1 \geq 1$, now. In this situation we could try to design a FN network

$$\boldsymbol{x}_{T-\tau:T} \ \mapsto \ \mu(\boldsymbol{x}_{T-\tau:T}) = \mathbb{E}[Y_{T+1}|\boldsymbol{x}_{T-\tau:T}] = \mathbb{E}[Y_{T+1}|\boldsymbol{x}_{T-\tau}, \ldots, \boldsymbol{x}_T].$$

This network regression function can be chosen independent of T since the relevant history $\boldsymbol{x}_{T-\tau:T}$ always has the same length $\tau + 1$. The time variable T could be used as a feature component in $\boldsymbol{x}_{T-\tau:T}$. The disadvantage of this approach is that such a FN network architecture does not respect the temporal causality. Observe that we feed the past history into the first FN layer

$$\boldsymbol{x}_{T-\tau:T} \ \mapsto \ \boldsymbol{z}^{(1)}(\boldsymbol{x}_{T-\tau:T}) \ \in \ \{1\} \times \mathbb{R}^{q_1}.$$

This operation typically does not respect any topology in the time index of $\boldsymbol{x}_{T-\tau+1:T}$. Thus, the FN network does not recognize that the feature $\boldsymbol{x}_{t-1}$ has been experienced just before the next feature $\boldsymbol{x}_t$. For this reason we are looking for a network architecture that can handle the time-series information in a temporal causal way.

[1] More mathematically speaking, we assume to have a filtration $(\mathcal{A}_t)_{t\geq 0}$ on the probability space $(\Omega, \mathcal{A}, \mathbb{P})$. The basic assumption then is that both sequences $(\boldsymbol{x}_t)_t$ and $(Y_t)_t$ are $(\mathcal{A}_t)_t$-adapted, and we aim at predicting Y_{T+1}, based on the information $\mathcal{A}_T$. In the above case this information $\mathcal{A}_T$ is generated by $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$, where $\boldsymbol{x}_t$ typically includes the observation Y_t. We could also shift the time index in $\boldsymbol{x}_t$ by one time unit, and in that case we would assume that $(\boldsymbol{x}_t)_t$ is previsible w.r.t. the filtration $(\mathcal{A}_t)_t$. We do not consider this shift in time index as it only makes the notation unnecessarily more complicated, but the results remain the same by including the information correspondingly into the features.

8.2 Plain-Vanilla Recurrent Neural Network

8.2.1 Recurrent Neural Network Layer

We explain the basic idea of RN networks in a shallow network architecture, and deep network architectures will be discussed in Sect. 8.2.2, below. We start from the time-series input variable $\boldsymbol{x}_{0:T} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T)$, all components having the same structure $\boldsymbol{x}_t \in \mathcal{X} \subset \{1\} \times \mathbb{R}^{q_0}$, $0 \leq t \leq T$. The aim is to design a network architecture that allows us to predict the random variable Y_{T+1}, based on this time-series information $\boldsymbol{x}_{0:T}$.

The main idea is to feed one component $\boldsymbol{x}_t$ of the time-series $\boldsymbol{x}_{0:T}$ at a time into the network, and at the same time we use the output $\boldsymbol{z}_{t-1}$ of the previous loop as an input for the next loop. This variable $\boldsymbol{z}_{t-1}$ carries forward a memory of the past variables $\boldsymbol{x}_{0:t-1}$. We explain this with a single RN layer having $q_1 \in \mathbb{N}$ neurons. A RN layer is given (recursively) by a mapping, $t \geq 1$,

$$\begin{aligned} \boldsymbol{z}^{(1)} : \{1\} \times \mathbb{R}^{q_0} \times \mathbb{R}^{q_1} &\to \mathbb{R}^{q_1}, \\ (\boldsymbol{x}_t, \boldsymbol{z}_{t-1}) &\mapsto \boldsymbol{z}_t = \boldsymbol{z}^{(1)}(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}), \end{aligned} \tag{8.2}$$

where the RN layer $\boldsymbol{z}^{(1)}$ has the same structure as the FN layer given in (7.5), but based on feature input $(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}) \in \mathcal{X} \times \mathbb{R}^{q_1} \subset \{1\} \times \mathbb{R}^{q_0} \times \mathbb{R}^{q_1}$, and not including an intercept component $\{1\}$ in the output.

More formally, a *RN layer* with activation function ϕ is a mapping

$$\begin{aligned} \boldsymbol{z}^{(1)} : \{1\} \times \mathbb{R}^{q_0} \times \mathbb{R}^{q_1} &\to \mathbb{R}^{q_1} \\ (\boldsymbol{x}, \boldsymbol{z}) &\mapsto \boldsymbol{z}^{(1)}(\boldsymbol{x}, \boldsymbol{z}) = \left(z_1^{(1)}(\boldsymbol{x}, \boldsymbol{z}), \ldots, z_{q_1}^{(1)}(\boldsymbol{x}, \boldsymbol{z})\right)^\top, \end{aligned} \tag{8.3}$$

having neurons, $1 \leq j \leq q_1$,

$$z_j^{(1)}(\boldsymbol{x}, \boldsymbol{z}) = \phi\left(\left\langle \boldsymbol{w}_j^{(1)}, \boldsymbol{x} \right\rangle + \left\langle \boldsymbol{u}_j^{(1)}, \boldsymbol{z} \right\rangle\right), \tag{8.4}$$

for given network weights $\boldsymbol{w}_j^{(1)} \in \mathbb{R}^{q_0+1}$ and $\boldsymbol{u}_j^{(1)} \in \mathbb{R}^{q_1}$.

Thus, the FN layers (7.5)–(7.6) and the RN layers (8.3)–(8.4) are structurally equivalent, only the input $\boldsymbol{x} \in \mathcal{X}$ is adapted to the time-series structure $(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}) \in \mathcal{X} \times \mathbb{R}^{q_1}$. Before giving more interpretation and before explaining how this single RN network structure can be extended to a deep RN network we illustrate this RN layer.

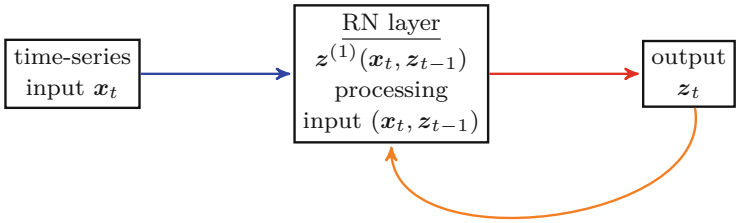

Fig. 8.1 RN layer $z^{(1)}$ processing the input (x_t, z_{t-1})

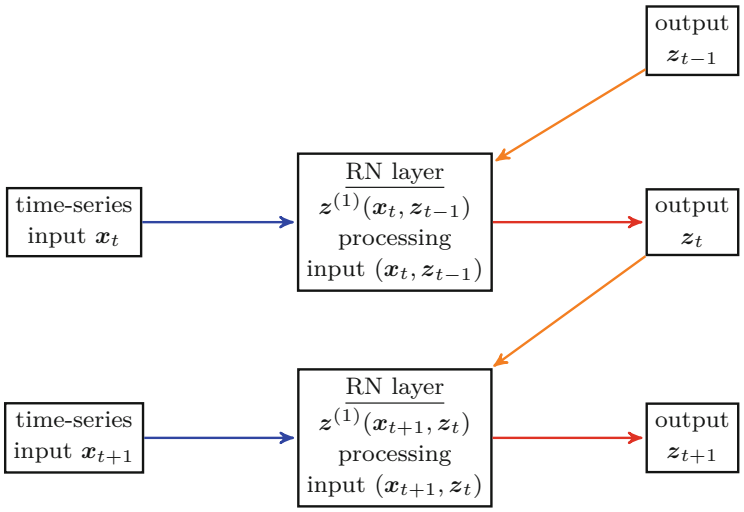

Fig. 8.2 Unfolded representation of RN layer $z^{(1)}$ processing the input (x_t, z_{t-1})

Figure 8.1 shows an RN layer $z^{(1)}$ processing the input (x_t, z_{t-1}), see (8.2). From this graph, the recurrent structure becomes clear since we have a loop (cycle) feeding the output z_t back into the RN layer to process the next input (x_{t+1}, z_t).
Often one depicts the RN architecture in a so-called unfolded way. This is done in Fig. 8.2. Instead of plotting the loop (cycle) as in Fig. 8.1 (orange arrow in the colored version), we unfold this loop by plotting the RN layer multiple times. Note that this RN layer in Fig. 8.2 uses always the same network weights $w_j^{(1)}$ and $u_j^{(1)}$, $1 \le j \le q_1$, for all t. Moreover, the use of the colors of the arrows (in the colored version) in the two figures coincides.

Remarks 8.1

- The neurons of the RN layer (8.4) have the following structure

$$z_j^{(1)}(x, z) = \phi\left(\langle w_j^{(1)}, x\rangle + \langle u_j^{(1)}, z\rangle\right) = \phi\left(w_{0,j}^{(1)} + \sum_{l=1}^{q_0} w_{l,j}^{(1)} x_l + \sum_{l=1}^{q_1} u_{l,j}^{(1)} z_l\right).$$

The network weights $W^{(1)} = (\boldsymbol{w}_j^{(1)})_{1 \le j \le q_1} \in \mathbb{R}^{(q_0+1) \times q_1}$ include an intercept component $w_{0,j}^{(1)}$ and the network weights $U^{(1)} = (\boldsymbol{u}_j^{(1)})_{1 \le j \le q_1} \in \mathbb{R}^{q_1 \times q_1}$ do not include an intercept component, otherwise we would have a redundancy.

- The RN network architecture generates a new process $(z_t)_t$. This process encodes the part of the past history $(\boldsymbol{x}_{0:t})_t$ which is relevant for forecasting the next step. Thus, $(z_t)_t$ can be interpreted as a (latent) *memory process*, or as the process of learned (relevant) time-series representation giving us $z_t = z_t(\boldsymbol{x}_{0:t})$.
- The same activation function ϕ and the same network weights $(\boldsymbol{w}_j^{(1)})_{1 \le j \le q_1}$ and $(\boldsymbol{u}_j^{(1)})_{1 \le j \le q_1}$ are shared across all time periods $t \ge 0$. This means that we assume a stationary (stochastic) process.
- The upper index $^{(1)}$ indicates the fact that this is the first (and single) RN layer in this example. In this sense, Figs. 8.1 and 8.2 show a shallow RN network. In the next section we are going to discuss deep RN networks, and below we are also going to discuss how the output is modeled, i.e., how the response Y_{T+1} is predicted based on the pre-processed features $(z_t)_{0 \le t \le T} \in \mathbb{R}^{q_1 \times (T+1)}$.

8.2.2 Deep Recurrent Neural Network Architectures

There are many different ways of extending a shallow RN network to a deep RN network. Assume we want to model a RN network of depth $d \ge 2$. A first (obvious) way of receiving a deep RN network architecture is

$$z_t^{[1]} = z^{(1)}\left(\boldsymbol{x}_t, z_{t-1}^{[1]}\right) \in \mathbb{R}^{q_1}, \tag{8.5}$$

$$z_t^{[m]} = z^{(m)}\left(z_t^{[m-1]}, z_{t-1}^{[m]}\right) \in \mathbb{R}^{q_m} \qquad \text{for } 2 \le m \le d, \tag{8.6}$$

where all RN layers $z^{(m)}$, $1 \le m \le d$, are of type (8.3)–(8.4), and additionally we include an intercept component in the RN layers $z^{(m)}$, $2 \le m \le d$. We add the upper indices (in square brackets $[\cdot]$) to the time-series $(z_t^{[m]})_t$ to indicate which RN layer outputs these learned representations (memory processes). In fact, we could also write $z_t^{[m:1]}$ instead of $z_t^{[m]}$, because in $z_t^{[m:1]}$ the feature input $\boldsymbol{x}_{0:t}$ has been processed through m RN layers $z^{(1)}, \ldots, z^{(m)}$. For simplicity, we just use the notation $z_t^{[m]} = z_t^{[m]}(\boldsymbol{x}_{0:t})$.

We are going to use the following abbreviation for a RN layer $m \geq 1$

$$z_t^{[m]} = z^{(m)}\left(z_t^{[m-1]}, z_{t-1}^{[m]}\right) = \phi\left(\left\langle W^{(m)}, z_t^{[m-1]}\right\rangle + \left\langle U^{(m)}, z_{t-1}^{[m]}\right\rangle\right) \in \mathbb{R}^{q_m}, \tag{8.7}$$

where the weights $W^{(m)} = (\boldsymbol{w}_1^{(m)}, \ldots, \boldsymbol{w}_{q_m}^{(m)}) \in \mathbb{R}^{(q_{m-1}+1)\times q_m}$ include the intercept components, and the weights $U^{(m)} = (\boldsymbol{u}_1^{(m)}, \ldots, \boldsymbol{u}_{q_m}^{(m)}) \in \mathbb{R}^{q_m \times q_m}$ do not include any intercept components. The scalar product is understood column-wise in the weight matrices $W^{(m)}$ and $U^{(m)}$, and the activation ϕ is understood component-wise. Moreover, we initialize for the input $z_t^{[0]} = \boldsymbol{x}_t$.

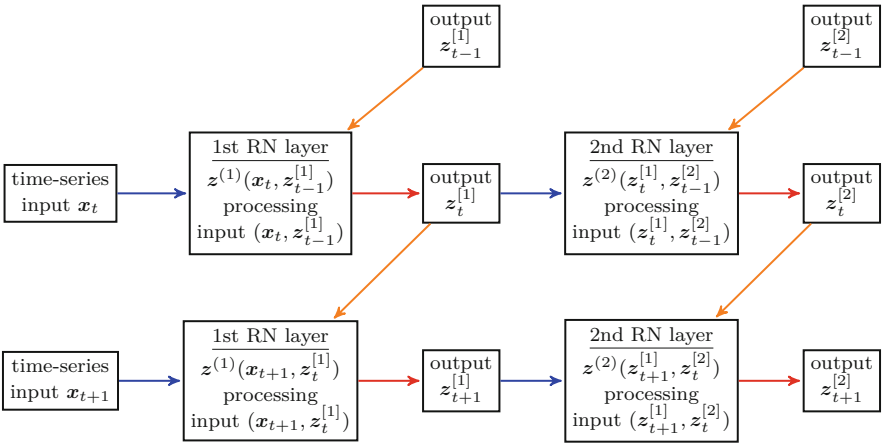

Fig. 8.3 Unfolded representation of a RN network architecture of depth $d = 2$

Figure 8.3 shows the RN network architecture of depth $d = 2$ defined in (8.5)–(8.6). The dimension of the input $z_t^{[0]} = \boldsymbol{x}_t \in \mathcal{X} \subseteq \{1\} \times \mathbb{R}^{q_0}$ is $q_0 + 1$, the first RN layer has q_1 neurons and the second RN layer q_2 neurons. From this graph it becomes clear how a RN network architecture of any depth $d \in \mathbb{N}$ can be constructed (recursively).

Remark 8.2 There are many alternative ways in building deep RN networks. E.g., we can add a loop that connects the output of the second RN layer back to the first one

$$z_t^{[1]} = z^{(1)}\left(\boldsymbol{x}_t, z_{t-1}^{[1]}, z_{t-1}^{[2]}\right),$$
$$z_t^{[2]} = z^{(2)}\left(z_t^{[1]}, z_{t-1}^{[2]}\right),$$

or we can add a skip connection from the input variable $\boldsymbol{x}_t$ to the second RN layer

$$z_t^{[1]} = z^{(1)}\left(\boldsymbol{x}_t, z_{t-1}^{[1]}\right),$$
$$z_t^{[2]} = z^{(2)}\left(\boldsymbol{x}_t, z_t^{[1]}, z_{t-1}^{[2]}\right).$$

We refrain from explicitly studying such RN network variants any further.

8.2.3 *Designing the Network Output*

There remains to explain how to predict the response variable Y_{T+1} based on the pre-processed features (memory processes) $z_T^{[1]}, \ldots, z_T^{[d]}$, outputted by the RN network of depth $d \geq 1$. Typically, only the final output of the last RN layer $z_T^{[d]} = z_T^{[d]}(\boldsymbol{x}_{0:T}) \in \mathbb{R}^{q_d}$ is considered to predict the response Y_{T+1}. We take this output and feed it into a FN network $\bar{z}^{(D:1)} : \{1\} \times \mathbb{R}^{q_d} \to \{1\} \times \mathbb{R}^{\bar{q}_D}$ of depth $D \in \mathbb{N}$ and with FN layers $\bar{z}^{(m)}$, $1 \leq m \leq D$, given by (7.5). Moreover, we choose a strictly monotone and smooth link function g.

This then provides us with the regression function, see (7.7)–(7.8),

$$\boldsymbol{x}_{0:T} \mapsto \mathbb{E}[Y_{T+1}|\boldsymbol{x}_{0:T}] = \mu(\boldsymbol{x}_{0:T}) = g^{-1}\left\langle \boldsymbol{\beta}, \bar{z}^{(D:1)}\left(z_T^{[d]}(\boldsymbol{x}_{0:T})\right)\right\rangle. \tag{8.8}$$

Thus, we first process the time-series features $\boldsymbol{x}_{0:T}$ through a RN network to receive the learned representation $z_T^{[d]}(\boldsymbol{x}_{0:T}) \in \mathbb{R}^{q_d}$ at time T. This learned representation is then used as a feature input to a FN network $\bar{z}^{(D:1)}$ that allows us to predict the response Y_{T+1}. This is illustrated in Fig. 8.4 for depth $d = 1$.

Remarks 8.3

- From the graph in Fig. 8.4 it also becomes apparent that we can consider different insurance policies $1 \leq i \leq n$ having different lengths of the corresponding histories $\boldsymbol{x}_{i,T-\tau_i:T} \in \mathbb{R}^{(q_0+1)\times(\tau_i+1)}$, $\tau_i \in \{0, \ldots, T\}$. The stationarity assumption allows us to enter the network in Fig. 8.4 at any time $T - \tau_i$. The RN network encodes this history into a learned feature $z_T^{[1]}(\boldsymbol{x}_{i,T-\tau_i:T})$ which is then decoded by the FN network $\bar{z}^{(D:1)}$ to forecast $Y_{i,T+1}$.
- If there is additional insurance policy dependent feature information $\widetilde{\boldsymbol{x}}_i$ that is not of a time-series structure, we can concatenate the feature information $(z_T^{[d]}(\boldsymbol{x}_{i,0:T}), \widetilde{\boldsymbol{x}}_i)$ which then enters the FN network (8.8).

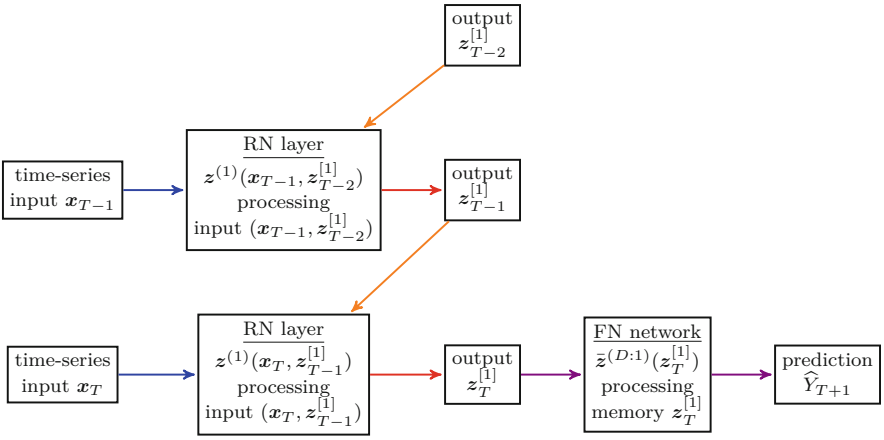

Fig. 8.4 Forecasting the response Y_{T+1} using a RN network (8.8) based on a single RN layer $d = 1$ and on a FN network of depth D

There remains to fit this network architecture having d RN layers and D FN layers to the available data. The RN layers involve the network weights $W^{(m)} \in \mathbb{R}^{(q_{m-1}+1)\times q_m}$ and $U^{(m)} \in \mathbb{R}^{q_m \times q_m}$, for $1 \le m \le d$, and the FN layers involve the network weights $(\bar{\boldsymbol{w}}_j^{(m)})_{1\le j\le \bar{q}_m} \in \mathbb{R}^{(\bar{q}_{m-1}+1)\times \bar{q}_m}$, for $1 \le m \le D$, and with $\bar{q}_0 = q_d$. Moreover, we have an output parameter $\boldsymbol{\beta} \in \mathbb{R}^{\bar{q}_D+1}$. The fitting is again done by a gradient descent algorithm minimizing the corresponding objective function.

Assume we have independent (in i) data $(Y_{i,T+1}, \boldsymbol{x}_{i,0:T}, v_{i,T+1})$ of the cases $1 \le i \le n$. We then assume that the responses $Y_{i,T+1}$ can be modeled by a fixed member of the EDF having unit deviance $\mathfrak{d}$. We consider the deviance loss function, see (4.9),

$$\boldsymbol{\vartheta} \mapsto \mathfrak{D}(\boldsymbol{Y}_{T+1}, \boldsymbol{\vartheta}) = \frac{1}{n}\sum_{i=1}^{n} \frac{v_{i,T+1}}{\varphi}\, \mathfrak{d}\Big(Y_{i,T+1}, \mu_{\boldsymbol{\vartheta}}(\boldsymbol{x}_{i,0:T})\Big), \tag{8.9}$$

for the observations $\boldsymbol{Y}_{T+1} = (Y_{1,T+1}, \ldots, Y_{n,T+1})^\top$, and where $\boldsymbol{\vartheta}$ collects all the RN and FN network weights/parameters of the regression function (8.8). This model can now be fitted using a variant of the gradient descent algorithm. The variant uses back-propagation through time (BPTT) which is an adaption of the back-propagation method to calculate the gradient w.r.t. the network parameter $\boldsymbol{\vartheta}$.

8.2.4 *Time-Distributed Layer*

There is a special feature in RN network modeling which is called a *time-distributed layer*. Observe from Fig. 8.4 that the deviance loss function (8.9) only focuses on the

final observation $Y_{i,T+1}$. However, the stationarity assumption allows us to output and study any (previous) observation $Y_{i,t+1}$, $0 \le t \le T$. A time-distributed layer considers applying the deep FN network (8.8) *simultaneously* at all time points $0 \le t \le T$; simultaneously meaning that we use the same FN network weights for all t. The latter is justified under the assumption of having stationarity.

This then provides us with the regressions

$$\boldsymbol{x}_{0:t} \;\mapsto\; \mathbb{E}[Y_{t+1}|\boldsymbol{x}_{0:t}] = \mu(\boldsymbol{x}_{0:t}) = g^{-1}\Big\langle \boldsymbol{\beta}, \bar{z}^{(D:1)}\Big(z_t^{[d]}(\boldsymbol{x}_{0:t})\Big)\Big\rangle \qquad \text{for all } t \ge 0. \tag{8.10}$$

Figure 8.5 illustrates a time-distributed output where we predict $(Y_{t+1})_t$ based on the history $(\boldsymbol{x}_{0:t})_t$, and we always apply the same FN network $\bar{z}^{(D:1)}$ to the memory $z_t^{[1]} = z_t^{[1]}(\boldsymbol{x}_{0:t})$.

A time-distributed layer changes the fitting procedure. Instead of considering the objective function (8.9) for the final observation $Y_{i,T+1}$, we now include all observations $\boldsymbol{Y} = (Y_{i,t+1})_{0\le t\le T, 1\le i\le n}$ into the objective function. This results in studying the deviance loss function

$$\boldsymbol{\vartheta} \;\mapsto\; \mathfrak{D}(\boldsymbol{Y}, \boldsymbol{\vartheta}) \;=\; \frac{1}{n}\sum_{i=1}^{n} \frac{1}{T+1}\sum_{t=0}^{T} \frac{v_{i,t+1}}{\varphi}\, \mathfrak{d}\Big(Y_{i,t+1}, \mu_{\boldsymbol{\vartheta}}(\boldsymbol{x}_{i,0:t})\Big). \tag{8.11}$$

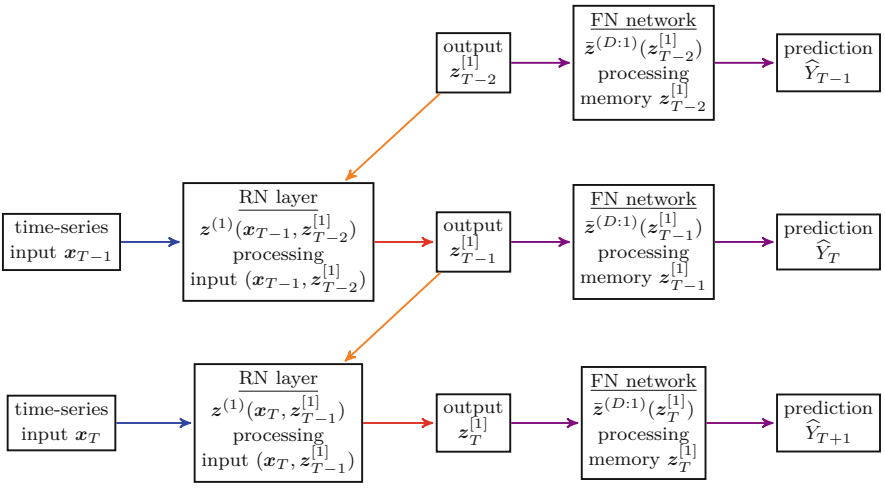

Fig. 8.5 Forecasting $(Y_{t+1})_t$ using a RN network (8.10) based on a single RN layer $d = 1$ and using a time-distributed FN layer for the outputs

Note that this can easily be adapted if the different cases $1 \le i \le n$ have different lengths in their histories. An example is provided in Listing 10.8, below.

8.3 Special Recurrent Neural Networks

In the plain-vanilla RN networks introduced above we have defined the memory processes $(z_t^{[m]})_{t\ge 0}$, $1 \le m \le d$, which encode the information history $(\boldsymbol{x}_t)_{t\ge 0}$ through different RN layers in a temporal causal way. This is naturally done through the use of a time-series structure as illustrated, e.g., in Fig. 8.5. There are more specific RN network architectures that allow the memory processes to be of a long memory or a short memory type. In this section, we present the two most popular architectures that pay a special attention to the memory storage. This is the long short-term memory (LSTM) architecture introduced by Hochreiter–Schmidhuber [188] and the gated recurrent unit (GRU) architecture proposed by Cho et al. [76].

8.3.1 Long Short-Term Memory Network

The LSTM network of Hochreiter–Schmidhuber [188] is the most commonly used RN network architecture. The LSTM network uses simultaneously three different activation functions for different purposes, the sigmoid and hyperbolic tangent activation functions, respectively,

$$\phi_\sigma(x) = \frac{1}{1+e^{-x}} \in (0,1) \qquad \text{and} \qquad \phi_{\tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \in (-1,1),$$

and a general activation function $\phi : \mathbb{R} \to \mathbb{R}$, see also Table 7.1.

The LSTM network relies on several RN layers that are of the same structure as the plain-vanilla RN layer given in (8.7). We start by defining three different so-called *gates* that all have the RN layer structure (8.7). These three gates are used to model the memory cell of the LSTM network. Choose a layer index $m \ge 1$ and assume that $z_t^{[m-1]}$ is modeled by the previous layer $m-1$; for $m = 1$ we initialize $z_t^{[0]} = \boldsymbol{x}_t$. The three gates are then defined as follows, set $t \ge 1$:

- The *forget gate* models the loss of memory rate

$$\boldsymbol{f}_t^{[m]} = \boldsymbol{f}^{(m)}\left(z_t^{[m-1]}, z_{t-1}^{[m]}\right) = \phi_\sigma^f\left(\left\langle W_f^{(m)}, z_t^{[m-1]}\right\rangle + \left\langle U_f^{(m)}, z_{t-1}^{[m]}\right\rangle\right) \in (0,1)^{q_m},$$

  with the network weights $W_f^{(m)} \in \mathbb{R}^{(q_{m-1}+1)\times q_m}$ and $U_f^{(m)} \in \mathbb{R}^{q_m \times q_m}$, and with the sigmoid activation function $\phi_\sigma^f = \phi_\sigma$, we also refer to (8.7).

- The *input gate* models the memory update rate

$$\boldsymbol{i}_t^{[m]} = \boldsymbol{i}^{(m)}\left(\boldsymbol{z}_t^{[m-1]}, \boldsymbol{z}_{t-1}^{[m]}\right) = \phi_\sigma^i\left(\left\langle W_i^{(m)}, \boldsymbol{z}_t^{[m-1]}\right\rangle + \left\langle U_i^{(m)}, \boldsymbol{z}_{t-1}^{[m]}\right\rangle\right) \in (0,1)^{q_m},$$

  with the network weights $W_i^{(m)} \in \mathbb{R}^{(q_{m-1}+1)\times q_m}$ and $U_i^{(m)} \in \mathbb{R}^{q_m \times q_m}$, and with the sigmoid activation function $\phi_\sigma^i = \phi_\sigma$.
- The *output gate* models the release of memory information rate

$$\boldsymbol{o}_t^{[m]} = \boldsymbol{o}^{(m)}\left(\boldsymbol{z}_t^{[m-1]}, \boldsymbol{z}_{t-1}^{[m]}\right) = \phi_\sigma^o\left(\left\langle W_o^{(m)}, \boldsymbol{z}_t^{[m-1]}\right\rangle + \left\langle U_o^{(m)}, \boldsymbol{z}_{t-1}^{[m]}\right\rangle\right) \in (0,1)^{q_m}, \tag{8.12}$$

  with the network weights $W_o^{(m)} \in \mathbb{R}^{(q_{m-1}+1)\times q_m}$ and $U_o^{(m)} \in \mathbb{R}^{q_m \times q_m}$, and with the sigmoid activation function $\phi_\sigma^o = \phi_\sigma$.

These gates have outputs in $(0, 1)$, and they determine the relative amount of memory that is updated and released in each step. The so-called *cell state process* $(\boldsymbol{c}_t^{[m]})_t$ is used to store the relevant memory. Given $\boldsymbol{z}_t^{[m-1]}$, $\boldsymbol{z}_{t-1}^{[m]}$ and $\boldsymbol{c}_{t-1}^{[m]}$, the updated cell state is defined by

$$\begin{aligned}\boldsymbol{c}_t^{[m]} &= \boldsymbol{c}^{(m)}\left(\boldsymbol{z}_t^{[m-1]}, \boldsymbol{z}_{t-1}^{[m]}, \boldsymbol{c}_{t-1}^{[m]}\right) \\ &= \boldsymbol{f}_t^{[m]} \odot \boldsymbol{c}_{t-1}^{[m]} + \boldsymbol{i}_t^{[m]} \odot \phi_{\tanh}\left(\left\langle W_c^{(m)}, \boldsymbol{z}_t^{[m-1]}\right\rangle + \left\langle U_c^{(m)}, \boldsymbol{z}_{t-1}^{[m]}\right\rangle\right) \in \mathbb{R}^{q_m},\end{aligned} \tag{8.13}$$

with the network weights $W_c^{(m)} \in \mathbb{R}^{(q_{m-1}+1)\times q_m}$ and $U_c^{(m)} \in \mathbb{R}^{q_m \times q_m}$, and $\odot$ denotes the Hadamard product. This defines how the memory (cell state) is updated and passed forward using the forget and the input gates $\boldsymbol{f}_t^{[m]}$ and $\boldsymbol{i}_t^{[m]}$, respectively.

The neuron activations $\boldsymbol{z}_t^{[m]}$ are updated, given $\boldsymbol{z}_t^{[m-1]}$, $\boldsymbol{z}_{t-1}^{[m]}$ and $\boldsymbol{c}_t^{[m]}$, by

$$\boldsymbol{z}_t^{[m]} = \boldsymbol{z}^{(m)}\left(\boldsymbol{z}_t^{[m-1]}, \boldsymbol{z}_{t-1}^{[m]}, \boldsymbol{c}_t^{[m]}\right) = \boldsymbol{o}_t^{[m]} \odot \phi\left(\boldsymbol{c}_t^{[m]}\right) \in \mathbb{R}^{q_m}, \tag{8.14}$$

with the cell state $\boldsymbol{c}_t^{[m]}$ given in (8.13) and the output gate $\boldsymbol{o}_t^{[m]}$ defined in (8.12). Figure 8.6[2] shows a LSTM cell (8.13)–(8.14) which includes four RN layers (8.7) for the forget gate $\boldsymbol{f}^{(m)}$, the input gate $\boldsymbol{i}^{(m)}$, the output gate $\boldsymbol{o}^{(m)}$ and in the cell state update (8.13). These RN layers are combined using the Hadamard product $\odot$ resulting in the updated cell state $\boldsymbol{c}_t^{[m]}$ and the learned representation $\boldsymbol{z}_t^{[m]}$ both being functions of the inputs $\boldsymbol{x}_{0:t}$.

[2] This figure is based on colah's blog explaining LSTMs https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

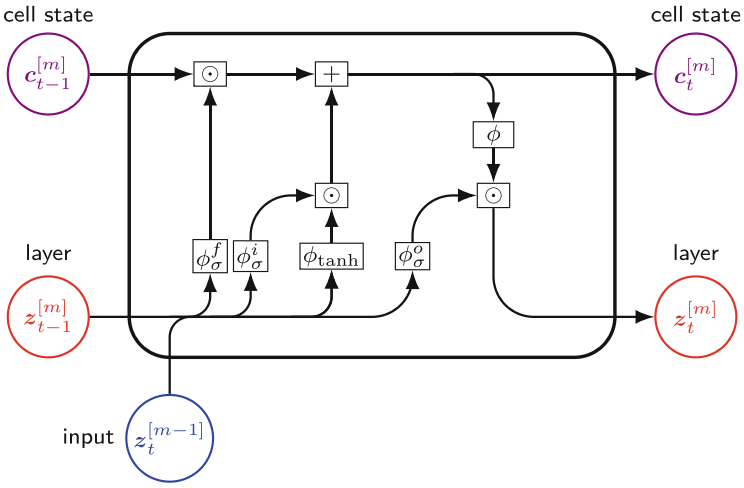

Fig. 8.6 LSTM cell $z^{(m)}$ with forget gate ϕ_σ^f, input gate ϕ_σ^i and output gate ϕ_σ^o

Below, we are going to summarize the LSTM cell update (8.13)–(8.14) as follows

$$\left(z_t^{[m-1]}, z_{t-1}^{[m]}, \boldsymbol{c}_{t-1}^{[m]}\right) \mapsto \left(z_t^{[m]}, \boldsymbol{c}_t^{[m]}\right) = z^{\mathrm{LSTM}(m)}\left(z_t^{[m-1]}, z_{t-1}^{[m]}, \boldsymbol{c}_{t-1}^{[m]}\right). \tag{8.15}$$

The update (8.15) involves the eight network weight matrices $W_f^{(m)}$, $W_i^{(m)}$, $W_o^{(m)}$, $W_c^{(m)} \in \mathbb{R}^{(q_{m-1}+1)\times q_m}$ and $U_f^{(m)}$, $U_i^{(m)}$, $U_o^{(m)}$, $U_c^{(m)} \in \mathbb{R}^{q_m \times q_m}$. Altogether we have $4(q_{m-1}+1+q_m)q_m$ network parameters in each LSTM cell $1 \le m \le d$. These are learned with the gradient descent method. Moreover, we need to initialize the LSTM cell update (8.15). From the previous layer $m-1$ we have the input $z_t^{[m-1]}$ which we initialize as $z_t^{[0]} = \boldsymbol{x}_t$ for $m = 1$ and $t \ge 0$. The initial states $z_0^{[m]}$ and $\boldsymbol{c}_0^{[m]}$ are usually set to zero.

8.3.2 *Gated Recurrent Unit Network*

The LSTM architecture of the previous section seems quite complex and involves many parameters. Cho et al. [76] have introduced the GRU architecture that is simpler and uses less parameters, but has similar properties. The GRU architecture uses two gates that are defined as follows for $t \ge 1$, see also (8.7):

- The *reset gate* models the memory reset rate

$$\boldsymbol{r}_t^{[m]} = \boldsymbol{r}^{(m)}\left(z_t^{[m-1]}, z_{t-1}^{[m]}\right) = \phi_\sigma^r\left(\left\langle W_r^{(m)}, z_t^{[m-1]}\right\rangle + \left\langle U_r^{(m)}, z_{t-1}^{[m]}\right\rangle\right) \in (0,1)^{q_m},$$

  with the network weights $W_r^{(m)} \in \mathbb{R}^{(q_{m-1}+1)\times q_m}$ and $U_r^{(m)} \in \mathbb{R}^{q_m \times q_m}$, and with the sigmoid activation function $\phi_\sigma^r = \phi_\sigma$.
- The *update gate* models the memory update rate

$$\boldsymbol{u}_t^{[m]} = \boldsymbol{u}^{(m)}\left(z_t^{[m-1]}, z_{t-1}^{[m]}\right) = \phi_\sigma^u\left(\left\langle W_u^{(m)}, z_t^{[m-1]}\right\rangle + \left\langle U_u^{(m)}, z_{t-1}^{[m]}\right\rangle\right) \in (0,1)^{q_m},$$

  with the network weights $W_u^{(m)} \in \mathbb{R}^{(q_{m-1}+1)\times q_m}$ and $U_u^{(m)} \in \mathbb{R}^{q_m \times q_m}$, and with the sigmoid activation function $\phi_\sigma^u = \phi_\sigma$.

The neuron activations $z_t^{[m]}$ are updated, given $z_t^{[m-1]}$ and $z_{t-1}^{[m]}$, by

$$\begin{aligned} z_t^{[m]} &= z^{(m)}\left(z_t^{[m-1]}, z_{t-1}^{[m]}\right) \\ &= \boldsymbol{r}_t^{[m]} \odot z_{t-1}^{[m]} + (\mathbf{1} - \boldsymbol{r}_t^{[m]}) \odot \phi\left(\langle W^{(m)}, z_t^{[m-1]}\rangle + \boldsymbol{u}_t^{[m]} \odot \langle U^{(m)}, z_{t-1}^{[m]}\rangle\right) \in \mathbb{R}^{q_m}, \end{aligned} \tag{8.16}$$

with the network weights $W^{(m)} \in \mathbb{R}^{(q_{m-1}+1)\times q_m}$ and $U^{(m)} \in \mathbb{R}^{q_m \times q_m}$, and for a general activation function ϕ.
The GRU and the LSTM architectures are similar, the former using less parameters because we do not explicitly model the cell state process. For an illustration of a GRU cell we refer to Fig. 8.7. In the sequel we focus on the LSTM architecture;

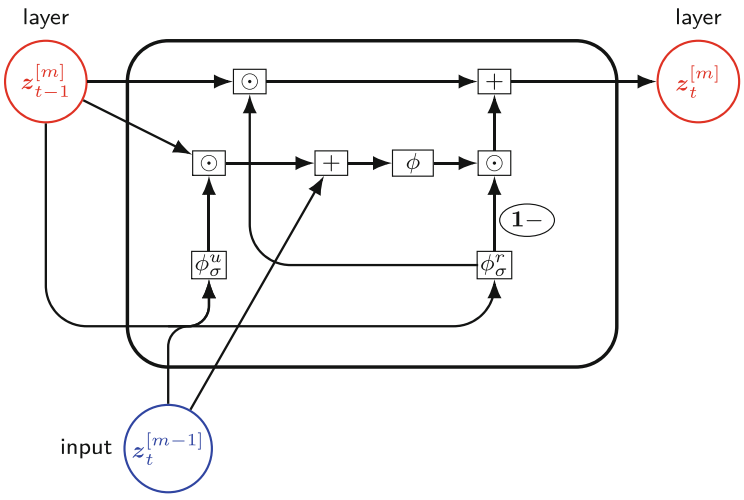

Fig. 8.7 GRU cell $z^{(m)}$ with reset gate ϕ_σ^r and update gate ϕ_σ^u

though the GRU architecture is simpler and has less parameters, it is less robust in fitting.

8.4 Lab: Mortality Forecasting with RN Networks

8.4.1 Lee–Carter Model, Revisited

The mortality data has a natural time-series structure, and for this reason mortality forecasting is an obvious problem that can be studied within RN networks. For instance, the LC mortality model (7.63) involves a stochastic process $(k_t)_t$ that needs to be extrapolated into the future. This extrapolation problem can be done in different ways. The original proposal of Lee and Carter [238] has been to analyze ARIMA time-series models, and to use standard statistical tools, Lee and Carter found that the random walk with drift gives a good stochastic description of the time index process $(k_t)_t$. Nigri et al. [286] proposed to fit a LSTM network to this stochastic process, this approach is also studied in Lindholm–Palmborg [252] where an efficient use of the mortality data for network fitting is discussed. These approaches still rely on the classical LC calibration using the SVD of Sect. 7.5.4, and the LSTM network is (only) used to extrapolate the LC time index process $(k_t)_t$.

More generally, one can design a RN network architecture that directly processes the raw mortality data $M_{x,t} = D_{x,t}/e_{x,t}$, not specifically relying on the LC structure. This has been done in Richman–Wüthrich [316] using a FN network architecture, in Perla et al. [301] using a RN network and a convolutional neural (CN) network architecture, and in Schürch–Korn [330] extending this analysis to the study of prediction uncertainty using bootstrapping. A similar CN network approach has been taken by Wang et al. [375] interpreting the raw mortality data of Fig. 7.32 as an image.

Lee–Carter Mortality Model: Random Walk with Drift Extrapolation

We revisit the LC mortality model [238] presented in Sect. 7.5.4. The LC log-mortality rate is assumed to have the following structure, see (7.63),

$$\log(\mu_{x,t}^{(p)}) = a_x^{(p)} + b_x^{(p)} k_t^{(p)},$$

for the ages $x_0 \le x \le x_1$ and for the calendar years $t \in \mathcal{T}$. We now add the upper indices $^{(p)}$ to consider different populations p. The SVD gives us the estimates $\widehat{a}_x^{(p)}$, $\widehat{k}_t^{(p)}$ and $\widehat{b}_x^{(p)}$ based on the observed centered raw log-mortality rates, see Sect. 7.5.4. The SVD is applied to each population p separately, i.e., there is no interaction between the different populations. This approach allows us to fit a separate log-mortality surface estimate $(\log(\widehat{\mu}_{x,t}^{(p)}))_{x_0 \le x \le x_1; t \in \mathcal{T}}$ to each population p. Figure 7.33

shows an example for two populations p, namely, for Swiss females and for Swiss males.

The mortality forecasting requires to extrapolate the time index processes $(\widehat{k}_t^{(p)})_{t\in\mathcal{T}}$ beyond the latest observed calendar year $t_1 = \max\{\mathcal{T}\}$. As mentioned in Lee–Carter [238] a random walk with drift provides a suitable model for modeling $(\widehat{k}_t^{(p)})_{t\geq 0}$ for many populations p, see Fig. 7.35 for the Swiss population. Assume that

$$\widehat{k}_{t+1}^{(p)} = \widehat{k}_t^{(p)} + \varepsilon_{t+1}^{(p)} \qquad t \geq 0, \tag{8.17}$$

with $\varepsilon_t^{(p)} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\delta_p, \sigma_p^2)$, $t \geq 1$, having drift $\delta_p \in \mathbb{R}$ and variance $\sigma_p^2 > 0$.

Model assumption (8.17) allows us to estimate the (constant) drift δ_p with MLE. For observations $(\widehat{k}_t^{(p)})_{t\in\mathcal{T}}$ we receive the log-likelihood function

$$\delta_p \;\mapsto\; \ell_{(\widehat{k}_t^{(p)})_{t\in\mathcal{T}}}(\delta_p) = \sum_{t=t_0+1}^{t_1} -\log(\sqrt{2\pi}\sigma_p) - \frac{1}{2\sigma_p^2}\left(\widehat{k}_t^{(p)} - \widehat{k}_{t-1}^{(p)} - \delta_p\right)^2,$$

with first observed calendar year $t_0 = \min\{\mathcal{T}\}$. The MLE is given by

$$\widehat{\delta}_p^{\text{MLE}} = \frac{\widehat{k}_{t_1}^{(p)} - \widehat{k}_{t_0}^{(p)}}{t_1 - t_0}. \tag{8.18}$$

This allows us to forecast the time index process for $t > t_1$ by

$$\widehat{k}_t^{(p)} = \widehat{k}_{t_1}^{(p)} + (t - t_1)\widehat{\delta}_p^{\text{MLE}}.$$

We explore this extrapolation for different Western European countries from the HMD [195]. We consider separately females and males of the countries {AUT, BE, CH, ESP, FRA, ITA, NL, POR}, thus, we choose $2 \cdot 8 = 16$ different populations p. For these countries we have observations for the ages $0 = x_0 \leq x \leq x_1 = 99$ and for the calendar years $1950 \leq t \leq 2018$.[3] For the following analysis we choose $\mathcal{T} = \{t_0 \leq t \leq t_1\} = \{1950 \leq t \leq 2003\}$, thus, we fit the models on 54 years of mortality history. This fitted models are then extrapolated to the calendar years $2004 \leq t \leq 2018$. These 15 calendar years from 2004 to 2018 allow us to perform an out-of-sample evaluation because we have the observations $M_{x,t}^{(p)} = D_{x,t}^{(p)}/e_{x,t}^{(p)}$ for these years from the HMD [195].

Figure 8.8 shows the estimated time index process $(\widehat{k}_t^{(p)})_{t\in\mathcal{T}}$ to the left of the dotted lines, and to the right of the dotted lines we have the random walk with drift extrapolation $(\widehat{k}_t^{(p)})_{t>t_1}$. The general observation is that, indeed, the random walk with drift seems to be a suitable model for $(\widehat{k}_t^{(p)})_t$. Moreover, there is a huge

[3] We exclude Germany from this consideration of (continental) Western European countries because the German mortality history is shorter due to the reunification in 1990.

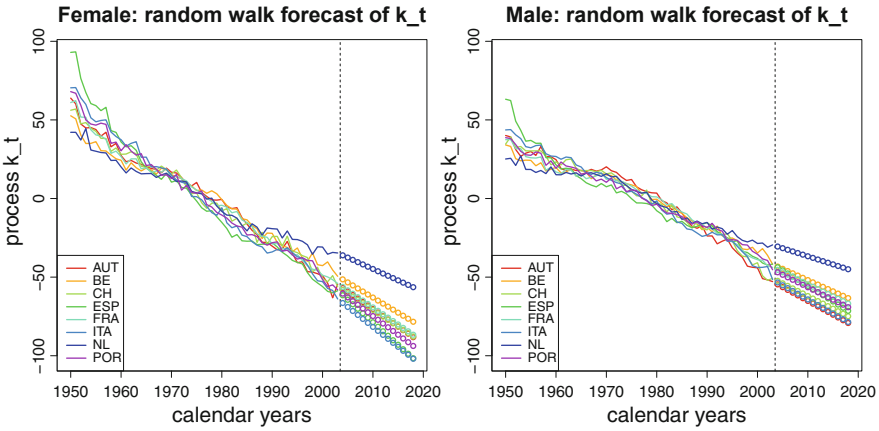

Fig. 8.8 Random walk with drift extrapolation of the time index process $(\widehat{k}_t)_t$ for different countries and genders; the y-scale is the same in both plots

similarity between the different countries, only with the Netherlands (NL) being somewhat an outlier.

Remarks 8.4

- For Fig. 8.8 we did not explore any fine-tuning, for instance, the estimation of the drift δ_p is very sensitive in the selection of the time span $\mathcal{T}$. ESP has the biggest negative drift estimate, but this is partially caused by the corresponding observations in the calendar years between 1950 and 1960, see Fig. 8.8, which may no longer be relevant for a decline in mortality in the new millennium.
- For all countries, the females have a bigger negative drift than the males (the y-scale in both plots is the same). Moreover, note that we use the normalization $\sum_{x=x_0}^{x_1} \widehat{b}_x^{(p)} = 1$ and $\sum_{t\in\mathcal{T}} \widehat{k}_t^{(p)} = 0$, see (7.65). This normalization is discussed and questioned in many publications as the extrapolation becomes dependent on these choices; see De Jong et al. [90] and the references therein, who propose different identification schemes.
- Another issue is an age coherence in forecasting, meaning that for long term forecasts the mortality rates across the different ages should not diverge, see Li et al. [250], Li–Lu [248] and Gao–Shi [153] and the references therein.
- There are many modifications and extensions of the LC model, we just mention a few of them. Brouhns et al. [56] embed the LC model into a Poisson modeling framework which provides a proper stochastic model for mortality modeling. Renshaw–Haberman [308] extend the one-factor LC model to a multifactor model, and in Renshaw–Haberman [309] a cohort effect is added. Hyndman–Ullah [197] and Hainaut–Denuit [179] explore a functional data method and a wavelet-based decomposition, respectively. The static PCA can be adopted to a dynamic PCA version, see Shang [333], and a long memory behavior in the time-series is studied in Yan et al. [395].

- The LC model is fitted to each population p separately, without exploring any common structure across the populations. There are many multi-population extensions that try to learn common structure across different populations. We mention the common age effect (CAE) model of Kleinow [218], the augmented common factor (ACF) model of Li–Lee [249] and the functional time-series models of Hyndman et al. [196] and Shang–Haberman [334]. A direct multi-population extension of the SVD matrix decomposition of the LC model is obtained by the tensor decomposition approaches of Russolillo et al. [325] and Dong et al. [110].

Lee–Carter Mortality Model: LSTM Extrapolation

Our aim here is to replace the individual random walk with drift extrapolations (8.17) by a common extrapolation across all considered populations p. For this we design a LSTM architecture. A second observation is that the increments $\varepsilon_t^{(p)} = \widehat{k}_t^{(p)} - \widehat{k}_{t-1}^{(p)}$ have an average empirical auto-correlation (for lag 1) of -0.33. This clearly questions the Gaussian i.i.d. assumption in (8.17).

We first discuss the available data and we construct the input data. We have the time-series observations $(\widehat{k}_t^{(p)})_{t\in\mathcal{T}}$, and the population index $p = (c, g)$ has two categorical labels c for country and g for gender. We are going to use two-dimensional embedding layers for these two categorical variables, see (7.31) for embedding layers. The time-series observations $(\widehat{k}_t^{(p)})_{t\in\mathcal{T}}$ will be pre-processed such that we do not simultaneously feed the entire time-series into the LSTM layer, but we divide them into shorter time-series. We will directly forecast the increments $\varepsilon_t^{(p)} = \widehat{k}_t^{(p)} - \widehat{k}_{t-1}^{(p)}$ and not the time index process $(\widehat{k}_t^{(p)})_{t\geq t_0}$; in extrapolations with drift it is easier to forecast the increments with the networks. We choose a *lookback period* of $\tau = 3$ calendar years, and we aim at predicting the response $Y_t = \varepsilon_t^{(p)}$ based on the time-series features $\boldsymbol{x}_{t-\tau:t-1} = (\varepsilon_{t-\tau}^{(p)}, \ldots, \varepsilon_{t-1}^{(p)})^\top \in \mathbb{R}^\tau$. This provides us with the following data structure for each population $p = (c, g)$:

year	country	gender	feature $\boldsymbol{x}_{t-\tau:t-1}$	Y_t
$t_0+\tau+1$	c	g	$\varepsilon_{t_0+1}^{(p)} \cdots \varepsilon_{t_0+\tau}^{(p)}$	$\varepsilon_{t_0+\tau+1}^{(p)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots \quad \vdots$	$\vdots$
t	c	g	$\varepsilon_{t-\tau}^{(p)} \cdots \varepsilon_{t-1}^{(p)}$	$\varepsilon_t^{(p)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots \quad \vdots$	$\vdots$
t_1	c	g	$\varepsilon_{t_1-\tau}^{(p)} \cdots \varepsilon_{t_1-1}^{(p)}$	$\varepsilon_{t_1}^{(p)}$

(8.19)

Thus, each observation $Y_t = \varepsilon_t^{(p)}$ is equipped with the feature information $(t, c, g, \boldsymbol{x}_{t-\tau:t-1})$. As discussed in Lindholm–Palmborg [252], one should highlight that there is a dependence across t, since we have a diagonal cohort structure in the

features and the observations $(\boldsymbol{x}_{t-\tau:t-1}, Y_t)$. Usually, this dependence is not harmful in stochastic gradient descent fitting.

Listing 8.1 LSTM architecture example

```
1  TS       = layer_input(shape=c(lookback,1), dtype='float32', name='TS')
2  Country  = layer_input(shape=c(1), dtype='int32',   name='Country')
3  Gender   = layer_input(shape=c(1), dtype='int32',   name='Gender')
4  Time     = layer_input(shape=c(1), dtype='float32', name='Time')
5  #
6  CountryEmb = Country %>%
7   layer_embedding(input_dim=8,output_dim=2,input_length=1,name='CountryEmb') %>%
8   layer_flatten(name='Country_flat')
9  #
10 GenderEmb = Gender %>%
11  layer_embedding(input_dim=2,output_dim=2,input_length=1,name='GenderEmb') %>%
12  layer_flatten(name='Gender_flat')
13 #
14 LSTM = TS %>%
15  layer_lstm(units=15,activation='tanh',recurrent_activation='sigmoid',
16                 name='LSTM')
17 #
18 Output = list(LSTM,CountryEmb,GenderEmb,Time) %>% layer_concatenate() %>%
19  layer_dense(units=10, activation='tanh',  name='FNLayer') %>%
20  layer_dense(units=1, activation='linear', name='Network')
21 #
22 model = keras_model(inputs = list(TS, Country, Gender, Time),
23                     outputs = c(Output))
```

In Fig. 8.9 we plot the LSTM architecture used to forecast $\varepsilon_t^{(p)}$ for $t > t_1$, and Listing 8.1 gives the corresponding R code. We process the time-series $\boldsymbol{x}_{t-\tau:t-1}$ through a LSTM cell, see lines 14–16 of Listing 8.1. We choose a shallow LSTM network ($d = 1$) and therefore drop the upper index $m = 1$ in (8.15), but we add an upper index $^{[\text{LSTM}]}$ to highlight the output of the LSTM cell. This gives us the

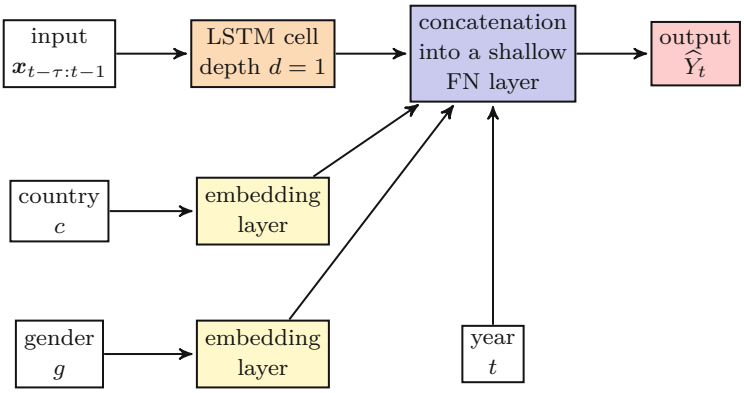

Fig. 8.9 LSTM architecture used to forecast $\varepsilon_t^{(p)}$ for $t > t_1$

LSTM cell updates for $t - \tau \leq s \leq t - 1$

$$\left(\boldsymbol{x}_s, \boldsymbol{z}_{s-1}^{[\mathrm{LSTM}]}, \boldsymbol{c}_{s-1}\right) \;\mapsto\; \left(\boldsymbol{z}_s^{[\mathrm{LSTM}]}, \boldsymbol{c}_s\right) = \boldsymbol{z}^{\mathrm{LSTM}}\left(\boldsymbol{x}_s, \boldsymbol{z}_{s-1}^{[\mathrm{LSTM}]}, \boldsymbol{c}_{s-1}\right).$$

This LSTM recursion to process the time-series $\boldsymbol{x}_{t-\tau:t-1}$ gives us the LSTM output $\boldsymbol{z}_{t-1}^{[\mathrm{LSTM}]} \in \mathbb{R}^{q_1}$, and it involves $4(q_0 + 1 + q_1)q_1 = 4(2 + 15)15 = 1'020$ network parameters for the input dimension $q_0 = 1$ and the output dimension $q_1 = 15$.

For the categorical country code c and the binary gender g we choose two-dimensional embedding layers, see (7.31),

$$c \mapsto \boldsymbol{e}^{\mathrm{C}}(c) \in \mathbb{R}^2 \qquad \text{and} \qquad g \mapsto \boldsymbol{e}^{\mathrm{G}}(g) \in \mathbb{R}^2,$$

these embedding maps give us $2(8 + 2) = 20$ embedding weights. Finally, we concatenate the LSTM output $\boldsymbol{z}_{t-1}^{[\mathrm{LSTM}]} \in \mathbb{R}^{15}$, the embeddings $\boldsymbol{e}^{\mathrm{C}}(c), \boldsymbol{e}^{\mathrm{G}}(g) \in \mathbb{R}^2$ and the continuous calendar year variable $t \in \mathbb{R}$ and process this vector through a shallow FN network with $q_2 = 10$ neurons, see lines 18–20 of Listing 8.1. This FN layer gives us $(q_1 + 2 + 2 + 1 + 1)q_2 = (15 + 2 + 2 + 1 + 1)10 = 210$ parameters. Together with the output parameter of dimension $q_2 + 1 = 11$, we receive 1'261 network parameters to be fitted, which seems quite a lot.

To fit this model we have $8 \cdot 2 = 16$ populations, and for each population we have the observations $\widehat{k}_t^{(p)}$ for the calendar years $1950 \leq t \leq 2003$. Considering the increments $\varepsilon_t^{(p)}$ and a lookback period of $\tau = 3$ calendar years gives us $2003 - 1950 - \tau = 50$ observations, rows in (8.19), per population p, thus, we have in total 800 observations. For the gradient descent fitting and the early stopping we choose a training to validation split of 8 : 2. As loss function we choose the squared error loss function. This implicitly implies that we assume that the increments $Y_t = \varepsilon_t^{(p)}$ are Gaussian distributed, or in other words, minimizing the squared error loss function means maximizing the Gaussian log-likelihood function. We then fit this model to the data using early stopping as described in (7.27). We analyze this fitted model. Figure 8.10 provides the learned embeddings for the country codes c. These learned embeddings have some similarity with the European map.

The final step is the extrapolation $\widehat{k}_t$, $t > t_1$. These updates need to be done recursively. We initialize for $t = t_1 + 1$ the time-series feature

$$\boldsymbol{x}_{t_1+1-\tau:t_1} = (\varepsilon_{t_1+1-\tau}^{(p)}, \ldots, \varepsilon_{t_1}^{(p)})^\top \in \mathbb{R}^\tau. \tag{8.20}$$

Using the feature information $(t_1 + 1, c, g, \boldsymbol{x}_{t_1+1-\tau:t_1})$ allows us to forecast the next increment $Y_{t_1+1} = \varepsilon_{t_1+1}^{(p)}$ by $\widehat{Y}_{t_1+1}$, using the fitted LSTM architecture of Fig. 8.9. Thus, this LSTM network allows us to perform a *one-period-ahead forecast* to receive

$$\widehat{k}_{t_1+1} = \widehat{k}_{t_1} + \widehat{Y}_{t_1+1}. \tag{8.21}$$

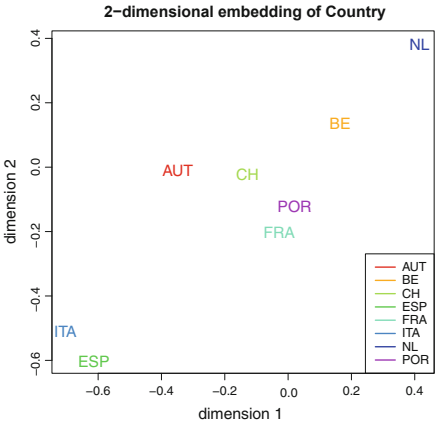

Fig. 8.10 Learned country embeddings for forecasting $(\widehat{k}_t)_t$

This update (8.21) needs to be iterated recursively. For the next period $t = t_1 + 2$ we set for the time-series feature

$$\boldsymbol{x}_{t_1+2-\tau:t_1+1} = (\varepsilon^{(p)}_{t_1+2-\tau}, \ldots, \varepsilon^{(p)}_{t_1}, \widehat{Y}_{t_1+1})^\top \in \mathbb{R}^\tau, \tag{8.22}$$

which gives us the next predictions $\widehat{Y}_{t_1+2}$ and $\widehat{k}_{t_1+2}$, etc.
In Fig. 8.11 we present the extrapolation of $(\varepsilon^{(p)}_t)_t$ for Belgium females and males. The blue curve shows the observed increments $(\varepsilon^{(p)}_t)_{1951\le t\le 2003}$ and the LSTM fitted (in-sample) values $(\widehat{Y}_t)_{1954\le t\le 2003}$ are in red color. Firstly, we observe a negative correlation (zig-zag behavior) in both the blue observations $(\varepsilon^{(p)}_t)_{1951\le t\le 2003}$ and in their red estimated means $(\widehat{Y}_t)_{1954\le t\le 2003}$. Thus, the LSTM finds this negative correlation (and it does not propose i.i.d. residuals). Secondly, the volatility in the

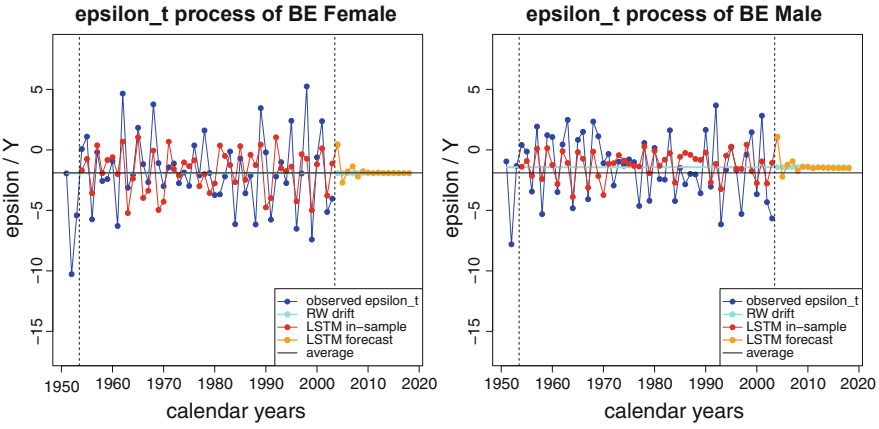

Fig. 8.11 LSTM network extrapolation $(\widehat{Y}_t)_{t>t_1}$ for Belgium (BE) females and males

red curve is smaller than in the blue curve, the former relates to expected values and the latter to observations of the random variables (which should be more volatile). The light-blue color shows the random walk with drift extrapolation (which is just a horizontal straight line at level $\widehat{\delta}_p^{\mathrm{MLE}}$, see (8.18)). The orange color shows the LSTM extrapolation using the recursive one-period-ahead updates (8.20)–(8.22), which has a zig-zag behavior that vanishes over time. This vanishing behavior is critical and is going to be discussed next.

There is one issue with this recursive one-period-ahead updating algorithm. This updating algorithm is not fully consistent in how the data is being used. The original LSTM architecture calibration is based on the feature components $\varepsilon_t^{(p)}$, see (8.20). Since these increments are not known for the later periods $t > t_1$, we replace their unknown values by the predictors, see (8.22). The subtle point here is that the predictors are on the level of expected values, and not on the level of random variables. Thus, $\widehat{Y}_t$ is typically less volatile than $\varepsilon_t^{(p)}$, but in (8.22) we pretend that we can use these predictors as a one-to-one replacement. A more consistent way would be to simulate/bootstrap $\varepsilon_t^{(p)}$ from $\mathcal{N}(\widehat{Y}_t, \sigma^2)$ so that the extrapolation receives the same volatility as the original process. For simplicity we refrain from doing so, but Fig. 8.11 indicates that this would be a necessary step because the volatility in the orange curve is going to vanish after the calendar year 2003, i.e., the zig-zag behavior vanishes, which is clearly not appropriate.

The LSTM extrapolation of $(\widehat{k}_t)_t$ is shown in Fig. 8.12. We observe quite some similarity to the random walk with drift extrapolation in Fig. 8.8, and, indeed, the random walk with drift seems to work very well (though the auto-correlation has not been specified correctly). Note that Fig. 8.8 is based on the individual extrapolations in p, whereas in Fig. 8.12 we have a common model for all populations.

Table 8.1 shows how often one model outperforms the other one (out-of-sample on calendar years $2004 \leq t \leq 2018$ and per gender). On the male populations of

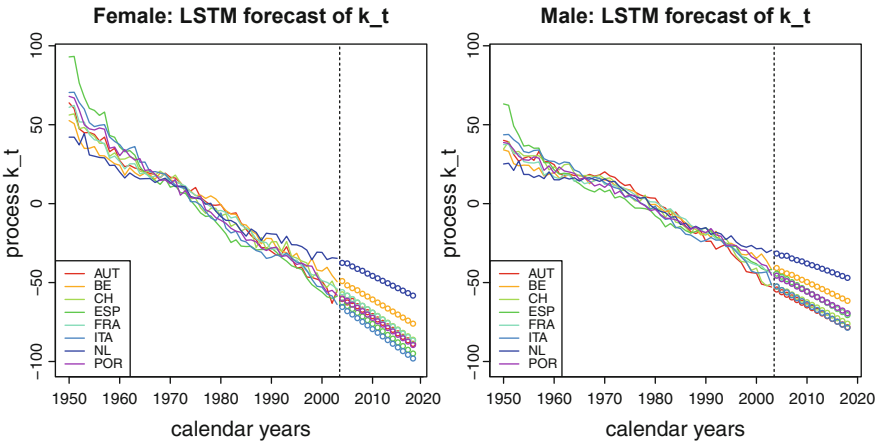

Fig. 8.12 LSTM network extrapolation of $(\widehat{k}_t)_t$ for different countries and genders

Table 8.1 Comparison of the out-of-sample mean squared error losses for the calendar years $2004 \leq t \leq 2018$: the numbers show how often one approach outperforms the other one on each gender

	Female	Male
Random walk with drift	5/8	4/8
LSTM architecture	3/8	4/8

the 8 European countries both models outperform the other one 4 times, whereas for the female population the random walk with drift gives 5 times the better out-of-sample prediction. Of course, this seems disappointing for the LSTM approach. This observation is quite common, namely, that the deep learning approach outperforms the classical methods on complex problems. However, on simple problems, as the one here, we should go for a classical (simpler) model like a random walk with drift or an ARIMA model.

8.4.2 Direct LSTM Mortality Forecasting

The previous section has been relying on the LC mortality model and only the extrapolation of the time-series $(\widehat{k}_t)_t$ has been based on a RN network architecture. In this section we aim at directly processing the raw mortality rates $M_{x,t} = D_{x,t}/e_{x,t}$ through a network, thus, we perform the representation learning directly on the raw data. We therefore use a simplified version of the network architecture proposed in Perla et al. [301].

As input to the network we use the raw mortality rates $M_{x,t}$. We choose a lookback period of $\tau = 5$ years and we define the time-series feature information to forecast the mortality in calendar year t by

$$\boldsymbol{x}_{t-\tau:t-1} = (\boldsymbol{x}_{t-\tau}, \ldots, \boldsymbol{x}_{t-1}) = \left(M_{x,s}\right)_{x_0 \leq x \leq x_1, t-\tau \leq s \leq t-1} \in \mathbb{R}^{(x_1-x_0+1)\times\tau} = \mathbb{R}^{100\times 5}. \tag{8.23}$$

Thus, we directly process the raw mortality rates (simultaneously for all ages x) through the network architecture; in the corresponding R code we need to input the transposed features $\boldsymbol{x}_{t-\tau:t-1}^{\top} \in \mathbb{R}^{5\times 100}$, see line 1 of Listing 8.2.

We choose a shallow LSTM network ($d = 1$) and drop the upper index $m = 1$ in (8.15). This gives us the LSTM cell updates for $t - \tau \leq s \leq t - 1$

$$\left(\boldsymbol{x}_s, \boldsymbol{z}_{s-1}^{[\mathrm{LSTM}]}, \boldsymbol{c}_{s-1}\right) \mapsto \left(\boldsymbol{z}_s^{[\mathrm{LSTM}]}, \boldsymbol{c}_s\right) = \boldsymbol{z}^{\mathrm{LSTM}}\left(\boldsymbol{x}_s, \boldsymbol{z}_{s-1}^{[\mathrm{LSTM}]}, \boldsymbol{c}_{s-1}\right).$$

This LSTM recursion to process the time-series $\boldsymbol{x}_{t-\tau:t-1}$ gives us the LSTM output $\boldsymbol{z}_{t-1}^{[\mathrm{LSTM}]} \in \mathbb{R}^{q_1}$, see lines 14–15 of Listing 8.2. It involves $4(q_0 + 1 + q_1)q_1 = 4(100 + 1 + 20)20 = 9'680$ network parameters for the input dimension $q_0 = 100$

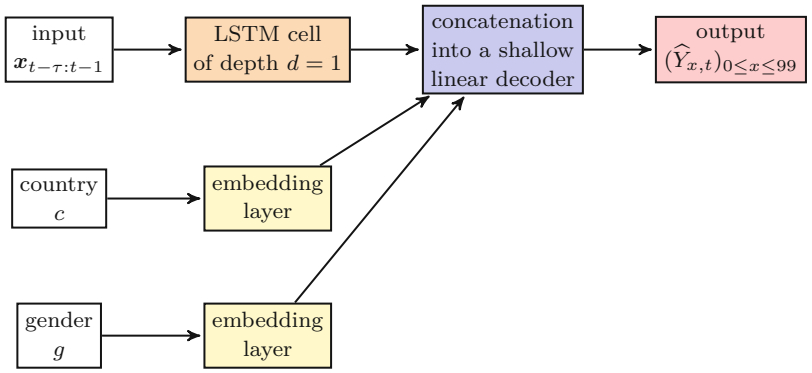

Fig. 8.13 LSTM architecture used to process the raw mortality rates $(M_{x,t})_{x,t}$

and the output dimension $q_1 = 20$. Many statisticians would probably stop at this point with this approach, as it seems highly over-parametrized. Let's see what we get.

For the categorical country code c and the binary gender g we choose two one-dimensional embeddings, see (7.31),

$$c \mapsto \boldsymbol{e}^{\mathrm{C}}(c) \in \mathbb{R} \qquad \text{and} \qquad g \mapsto \boldsymbol{e}^{\mathrm{G}}(g) \in \mathbb{R}. \tag{8.24}$$

These embeddings give us $8 + 2 = 10$ embedding weights. Figure 8.13 shows the LSTM cell in orange color and the embeddings in yellow color (in the colored version).

The LSTM output and the two embeddings are then concatenated to a learned representation $\boldsymbol{z}_{t-1} = (\boldsymbol{z}_{t-1}^{[\mathrm{LSTM}]}, \boldsymbol{e}^{\mathrm{C}}(c), \boldsymbol{e}^{\mathrm{G}}(g))^\top \in \mathbb{R}^{q_1 \times 1 \times 1} = \mathbb{R}^{22}$. The 22-dimensional learned representation $\boldsymbol{z}_{t-1}$ *encodes* the 500-dimensional input $\boldsymbol{x}_{t-\tau:t-1} \in \mathbb{R}^{100\times 5}$ and the two categorical variables c and g. The last step is to *decode* this representation $\boldsymbol{z}_{t-1} \in \mathbb{R}^{22}$ to predict the log-mortality rates $(Y_{x,t})_{0\le x\le 99} = (\log M_{x,t})_{0\le x\le 99} \in \mathbb{R}^{100}$, simultaneously for all ages x. This decoding is obtained by the code on lines 17–19 of Listing 8.2; this reads as

$$\boldsymbol{z}_{t-1} \;\mapsto\; \left(\beta_x^0 + \beta_x^C \boldsymbol{e}^{\mathrm{C}}(c) + \beta_x^G \boldsymbol{e}^{\mathrm{G}}(g) + \left\langle \boldsymbol{\beta}_x, \boldsymbol{z}_{t-1}^{[\mathrm{LSTM}]} \right\rangle\right)_{0\le x\le 99}. \tag{8.25}$$

This decoding involves another $(1 + 22)100 = 2'300$ parameters $(\beta_x^0, \beta_x^G, \beta_x^C, \boldsymbol{\beta}_x)_{0\le x\le 99}$. Thus, altogether this LSTM network has $r = 11'990$ parameters.

Summarizing: the above architecture follows the philosophy of the auto-encoder of Sect. 7.5. A high-dimensional observation $(\boldsymbol{x}_{t-\tau:t-1}, c, g)$ is encoded to a low-dimensional bottleneck activation $\boldsymbol{z}_{t-1} \in \mathbb{R}^{22}$, which is then decoded by (8.25) to give the forecast $(\widehat{Y}_{x,t})_{0\le x\le 99}$ for the log-mortality rates. It is not precisely an auto-encoder because the response is different from the input, as we forecast the log-mortality rates in the next calendar year t based on the information $\boldsymbol{z}_{t-1}$ that

Listing 8.2 LSTM architecture to directly process the raw mortality rates $(M_{x,t})_{x,t}$

```
1  TS       = layer_input(shape=c(lookback,100), dtype='float32', name='TS')
2  Country  = layer_input(shape=c(1), dtype='int32',   name='Country')
3  Gender   = layer_input(shape=c(1), dtype='int32',   name='Gender')
4  Time     = layer_input(shape=c(1), dtype='float32', name='Time')
5  #
6  CountryEmb = Country %>%
7   layer_embedding(input_dim=8,output_dim=1,input_length=1,name='CountryEmb') %>%
8   layer_flatten(name='Country_flat')
9  #
10 GenderEmb = Gender %>%
11  layer_embedding(input_dim=2,output_dim=1,input_length=1,name='GenderEmb') %>%
12  layer_flatten(name='Gender_flat')
13 #
14 LSTM = TS %>%
15  layer_lstm(units=20,activation='linear',recurrent_activation='sigmoid',
16                 name='LSTM')
17 #
18 Output = list(LSTM,CountryEmb,GenderEmb) %>% layer_concatenate() %>%
19  layer_dense(units=100, activation='linear',  name='scalarproduct') %>%
20  layer_reshape(c(1,100), name = 'Output')
21 #
22 model = keras_model(inputs = list(TS, Country, Gender),
23                     outputs = c(Output))
```

is available at the end of the previous calendar year $t-1$. In contrast to the LC mortality model, we no longer rely on the two-step approach by first fitting the parameters with a SVD, and performing a random walk with drift extrapolation. This encoder-decoder network performs both steps simultaneously.

We fit this network architecture to the available data. We have $r = 11'990$ network parameters. Based on a lookback period of $\tau = 5$ years, we have $2003 - 1950 - \tau + 1 = 49$ observations per population $p = (c, g)$. Thus, we have in total 784 observations $\left(\boldsymbol{x}_{t-\tau:t-1}, c, g, (Y_{x,t})_{0\le x\le 99}\right)$. We fit this network using the `nadam` version of the gradient descent algorithm. We choose a training to validation split of 8 : 2 and we explore 10'000 gradient descent epochs. A crucial observation is that the algorithm converges rather slowly and it does not show any signs of over-fitting, i.e., there is no strong need for the early stopping. This seems surprising because we have 11'990 parameters and only 784 observations. There are a couple of important ingredients that make this work. The features and observations themselves are high-dimensional, the low-dimensional encoding (compression) leads to a natural regularization, Moreover, this is combined with linear activation functions, see lines 15 and 19 of Listing 8.2. The gradient descent fitting has a certain inertness, and it seems that high-dimensional problems on comparably smooth high-dimensional data do not over-fit to individual components because the gradients are not very sensitive in the individual partial derivatives (in high dimensions). These high-dimensional approaches only work if we have sufficiently many populations across which we can learn, here we have 16 populations, Perla et al. [301] even use 76 populations.

Since every gradient descent fit still involves several elements of randomness, we consider the nagging predictor (7.44), averaging over 10 fitted networks, see

Table 8.2 Comparison of the out-of-sample mean squared losses for the calendar years $2004 \leq t \leq 2018$; the figures are in 10^{-4}

	LC female	LSTM female	LC male	LSTM male
Austria AUT	0.765	**0.312**	2.527	**1.169**
Belgium BE	0.371	**0.311**	2.835	**0.960**
Switzerland CH	0.654	**0.478**	1.609	**1.134**
Spain ESP	1.446	**0.514**	1.742	**0.245**
France FRA	**0.175**	1.684	**0.333**	0.363
Italy ITA	**0.179**	0.330	0.874	**0.320**
The Netherlands NL	0.426	**0.315**	1.978	**0.601**
Portugal POR	2.097	**0.464**	1.848	**1.239**

Sect. 7.4.4. The out-of-sample prediction results on the calendar years 2004 to 2018, i.e., $t > t_1 = 2004$, are presented in Table 8.2. These results verify the appropriateness of this LSTM approach. It outperforms the LC model on the female population in 6 out of 8 cases and on the male population on 7 out of 8 cases, only for the French population this LSTM approach seems to have some difficulties (compared to the LC model). Note that these are out-of-sample figures because the LSTM has only been fitted on the data prior to 2004. Moreover, we did not pre-process the raw mortality rates $M_{x,t}$, $t \leq 2003$, and the prediction is done recursively in a one-period-ahead prediction approach, we also refer to (8.22). A more detailed analysis of the results shows that the LC and the LSTM approaches have a rather similar behavior for females. For males the LSTM prediction clearly outperforms the LC model prediction, this out-performance is across different ages x and different calendar years $t \geq 2004$.

The advantage of this LSTM approach is that we can directly predict by processing the raw data. The disadvantage compared to the LC approach is that the LSTM network approach is more complex and more time-consuming. Moreover, unlike in the LC approach, we cannot (easily) assess the prediction uncertainty. In the LC approach the prediction uncertainty is obtained from assessing the uncertainty in the extrapolation and the uncertainty in the parameter estimates, e.g., using a bootstrap. The LSTM approach is not sufficiently robust (at least not on our data) to provide any reasonable uncertainty estimates.

We close this section and example by analyzing the functional form of the decoder (8.25). We observe that this decoder has much similarity with the LC model assumption (7.63)

$$\begin{aligned} \widehat{Y}_{x,t} &= \beta_x^0 + \beta_x^C \boldsymbol{e}^{\mathrm{C}}(c) + \beta_x^G \boldsymbol{e}^{\mathrm{G}}(g) + \left\langle \boldsymbol{\beta}_x, \boldsymbol{z}_{t-1}^{[\mathrm{LSTM}]} \right\rangle, \\ \log(\mu_{x,t}^{(p)}) &= a_x^{(p)} + b_x^{(p)} k_t^{(p)}. \end{aligned}$$

The LC model considers the average force of mortality $a_x^{(p)} \in \mathbb{R}$ for each population $p = (c, g)$ and each age x; the LSTM architecture has the same term $\beta_x^0 + \beta_x^C \boldsymbol{e}^{\mathrm{C}}(c) +$

$\beta_x^G \boldsymbol{e}^{\mathrm{G}}(g)$. In the LC model, the change of force of mortality is considered by a population-dependent term $b_x^{(p)} k_t^{(p)}$, whereas the LSTM architecture has a term $\langle \boldsymbol{\beta}_x, \boldsymbol{z}_{t-1}^{[\mathrm{LSTM}]} \rangle$. This latter term is also population-dependent because the LSTM cell directly processes the raw mortality data $M_{x,t}$ coming from the different populations p. Note that this is the only time-t-dependent term in the LSTM architecture. We conclude that the main difference between these two forecast approaches is how the past mortality observations are processed. Apart from that the general structure is the same.