Time-series forecasting of mortality rates using deep learning

Francesca Perla, Ronald Richman, Salvatore Scognamiglio & Mario V. Wüthrich

Taylor & Francis
Taylor & Francis Group

Check for updates

Time-series forecasting of mortality rates using deep learning

Francesca Perla[a], Ronald Richman [b], Salvatore Scognamiglio[a] and Mario V. Wüthrich[c]

[a]Department of Management and Quantitative Sciences, University of Naples 'Parthenope', Napoli, Italy; [b]QED Actuaries and Consultants, Johannesburg, South Africa; [c]RiskLab, Department of Mathematics, ETH Zurich, Zurich, Switzerland

ABSTRACT

The time-series nature of mortality rates lends itself to processing through neural networks that are specialized to deal with sequential data, such as recurrent and convolutional networks. The aim of this work is to show how the structure of the Lee–Carter model can be generalized using a relatively simple shallow convolutional network model, allowing for its components to be evaluated in familiar terms. Although deep networks have been applied successfully in many areas, we find that deep networks do not lead to an enhanced predictive performance in our approach for mortality forecasting, compared to the proposed shallow one. Our model produces highly accurate forecasts on the Human Mortality Database, and, without further modification, generalizes well to the United States Mortality Database.

1. Introduction

Accurate forecasts of mortality rates are key inputs for pricing and reserving models used by actuaries working in life and pension insurance. This is especially the case in times when interest rates are low, implying that the financial results of life insurance companies and pension plans are highly sensitive to mortality rate assumptions. Furthermore, actuaries who work in general insurance may also require accurate forecasts of mortality for specific purposes, e.g. for the valuation of annuities resulting from general insurance claims, such as periodical payment orders in the UK, or workmen's compensation pensions in South Africa. In addition to the use of forecasts for actuarial purposes, demographers require accurate mortality rate forecasts to produce population projections that form the basis of governmental planning and policy setting. Thus, the development of mortality forecasting techniques has been emphasized by both disciplines, usually in the form of statistical models which produce forecasts for single populations. Besides this usual application of mortality forecasting models, both actuaries and demographers may have a need for large-scale mortality forecasts, meaning to say, simultaneous forecasts of the mortality of multiple populations. Within actuarial work, this need arises when assumptions must be set for many different portfolios, each with its own mortality experience, such as a life insurance portfolio of a multinational company which will be comprised of policyholders from multiple countries. Similarly, demographers may be interested in simultaneous population projections of multiple countries, or regions within countries and, thus, may also require large-scale forecasts. Moreover, another motivation for large-scale forecasts is that joint forecasts of mortality rates may lead to more accurate forecasting results for individual countries, see e.g. Li & Lee (2005), and thus large-scale forecasts may be produced even if a limited number of the outputs will be used.

The Lee–Carter (LC) mortality model Lee & Carter (1992) is a fundamental approach to forecasting mortality rates of a single population. The LC model decomposes mortality rates into an age-related static life-table component and a time-related index component using principal component analysis (PCA); we formalize the LC approach to mortality forecasting in the next section. The original LC model was applied directly to forecast mortality rates, whereas later extensions of the LC model by Brouhns et al. (2002) embedded the LC model into a Poisson regression framework, in some instances producing more accurate forecasts than the original LC model. There are numerous extensions of the LC model; in the following, we mention some of them. Renshaw & Haberman (2006) added a cohort effect to the LC model to capture different mortality of different cohorts of lives; Hyndman & Ullah (2007) explore a functional data method using penalized regression splines, and Hainaut & Denuit (2020) adopt the approach of Hyndman & Ullah (2007) to a wavelet-based decomposition enjoying favorable properties in time-series modeling. Shang (2019) extends the classical LC model based on a static PCA to a dynamic PCA regression to model temporal dependence. A different approach to forecasting mortality is the Cairns–Blake–Dowd (CBD) model (Cairns et al. 2006), which proceeds by fitting a parametric mortality model to each year of mortality experience separately, and then extrapolating the coefficients of mortality to future years. For example, in Cairns et al. (2006), mortality at the older ages was assumed to be linear on the logistic scale, and separate logistic models were fit for each year of the mortality experience, providing a time-series of mortality coefficients that could be extrapolated. Some extensions of the CBD model appear in Cairns et al. (2009), who added a quadratic component to the parametric mortality model, as well as a cohort effect. From a computational perspective, whereas Brouhns et al. (2002) and Renshaw & Haberman (2006) utilize non-standard methodologies to fit their extensions of the LC model, Currie (2017) shows how generalized linear and nonlinear models can be used to fit both the LC model and other popular mortality models, Villegas et al. (2018) describe an R package with an interface for fitting generalized age-period-cohort mortality models. While the LC model can be applied to the whole age range, the CBD model is only used to model middle- and old-age adult mortality. Since in this work, we are concerned with the large-scale forecasting of the entire life-table, we focus on the LC model in what follows.

The LC model, along with many other single population mortality models, such as the CBD model, is usually fit in a two-stage procedure: in the first stage, a parsimonious model to predict the life-table for each year is found, and in the second stage, the coefficients of that model are extrapolated using time-series methods to produce forecasts in future years. If the mortality rates of multiple populations are to be forecast, then the LC model may be fit to each population separately. However, since mortality may change at potentially different rates in different periods and for different populations, it becomes necessary to carefully choose the periods in which the models are fit to each population so that reliable forecasts are produced. Thus, quite some element of judgment and manual intervention is required when applying the LC model to multiple populations.

Several multi-population extensions of the LC model have been proposed for forecasting mortality of multiple populations simultaneously, for example, the common age effect (CAE) model of Kleinow (2015), the augmented common factor (ACF) model of Li & Lee (2005) and functional time-series models of Hyndman et al. (2013) and Shang & Haberman (2020). These extensions may make the task of mortality forecasting for multiple populations somewhat easier. However, these extensions were usually intended for forecasting the mortality of similar populations and not for large-scale mortality forecasting, by which we mean the simultaneous production of forecasts for many different and potentially unrelated populations, for example, forecasting the mortality of all populations in the Human Mortality Database (HMD 2020) (Wilmoth & Shkolnikov 2010) simultaneously. Besides this, a comparison of the out of sample forecast accuracy of the LC, ACF and CAE models in Richman & Wüthrich (2020) showed that the multi-population models did not perform fully competitively against the LC model.

In this paper, we explore how large-scale mortality forecasts can be produced using neural networks that are adapted to process data with a time-series structure. By using neural network

architectures that are specialized for time-series forecasting, it is reasonable to expect that better forecasts can be produced than would be possible by using other neural network architectures, such as feed-forward fully connected neural networks (FCNs), which are not specifically intended for time-series data. On the one hand, we consider recurrent neural networks (RNNs), which are a type of neural network architecture that has been designed for processing and forecasting sequential and time-series data. In particular, we consider the application of the long short-term memory (LSTM) (Hochreiter & Schmidhuber 1997) network architecture. On the other hand, we also consider convolutional neural networks (CNNs) which are often used for image recognition, but which have also been successfully applied to time-series analysis (Borovykh et al. 2017). We incorporate both of these network architectures into a network model that is capable of producing large-scale mortality forecasts.

Recently, Richman & Wüthrich (2020) have proposed a multi-population extension of the LC model using FCNs and embedding layers, which are a specialized method of incorporating categorical data into neural networks; see Bengio et al. (2003) in the context of natural language processing (NLP) and Richman (2018) for actuarial applications. To model mortality rates over time the calendar year in which mortality is to be forecast was treated as a continuous input variable into the model. This allowed one for a natural extrapolation to future calendar years using the fitted functional form. In this respect, the model in Richman & Wüthrich (2020) was dissimilar from the LC model which uses time-series techniques to extrapolate and forecast mortality rates. In another recent paper (Nigri et al. 2019), the authors apply RNNs within the usual two-step procedure to fit the LC model, but, instead of forecasting the coefficients with a statistical time-series model, they applied LSTM networks, and found that the resulting forecasts were more accurate and flexible than the classical approach.

In the present paper, we use the forecasts of Richman & Wüthrich (2020) as a benchmark (since those forecasts produced lower prediction errors on the HMD than the other models considered in that study, including the LC, ACF and CAE models) for several different network architectures that directly process time-series of mortality rates to produce forecasts. That is, our proposal here provides a direct forecast approach which does not build on a two-stage procedure.

The rest of the paper is organized as follows. In the next section, we review key concepts of mortality modeling and discuss how the model proposed in this study differs from the LC model. In Section 3, we review network architectures and discuss RNNs, LSTMs and CNNs. In Section 4, we define the models used in this paper, and we illustrate how the model proposed is a locally calibrated version of the LC model. In Section 5, we fit these models to data from the HMD, as well as to data from the United States HMD (University of California 2020). Finally, Section 6 concludes.

2. Mortality modeling and the Lee–Carter model

We are concerned with modeling and forecasting mortality rates $u_{x,t} > 0$ in calendar years t, where x denotes age at the last birthday. We therefore introduce a general modeling framework. This framework contains as special cases the LC, CAE and AFC models. We first extend the mortality rate notation $u_{x,t}^{(i)} > 0$ to different populations indexed by $i \in \mathcal{I}$, where these populations may differ in gender, country, sub-national region or other characteristics. We denote the vector of mortality rates in calendar year t for population i as $\boldsymbol{u}_t^{(i)} = (u_{x,t}^{(i)})_{x_0 \leq x \leq x_1} \in \mathbb{R}_+^{x_1 - x_0 + 1}$, where $x_0 < x_1$ are the minimal and maximal ages considered, respectively. The mortality rates of all ages and calendar years between t_0 and t_1 are denoted by the matrix

$$U^{(i)} = (u_{x,t}^{(i)})_{x_0 \leq x \leq x_1, t_0 \leq t \leq t_1} = (\boldsymbol{u}_{t_0}^{(i)}, \ldots, \boldsymbol{u}_{t_1}^{(i)}) \in \mathbb{R}_+^{(x_1 - x_0 + 1) \times (t_1 - t_0 + 1)}.$$

Mortality modeling is usually concerned with finding good regression functions

$$(t, x, i) \mapsto \log (u_{x,t}^{(i)}). \tag{1}$$

The LC model and multi-population variants thereof can be summarized by the following structural form:

$$\log(u_{x,t}^{(i)}) = a_x^{(i)} + \langle \boldsymbol{b}_x^{(i)}, \boldsymbol{k}_t^{(i)} \rangle, \tag{2}$$

where $a_x^{(i)} \in \mathbb{R}$, $\boldsymbol{b}_x^{(i)}, \boldsymbol{k}_t^{(i)} \in \mathbb{R}^q$, and $\langle \cdot, \cdot \rangle$ denotes the scalar product in $\mathbb{R}^q$. The parameters $a_x^{(i)}$, $\boldsymbol{b}_x^{(i)}$ and $\boldsymbol{k}_t^{(i)}$ can be interpreted as follows:

- $(a_x^{(i)})_x$ is the average force of mortality of population $i \in \mathcal{I}$. It can be thought of an embedding of (x, i) into the one-dimensional Euclidean space, that is,

$$(x, i) \mapsto a_x^{(i)} \in \mathbb{R}. \tag{3}$$

  We come back to embeddings more generally in Section 3.2.
- $(\boldsymbol{k}_t^{(i)})_t$ is the time index that describes the change of force of mortality of population $i \in \mathcal{I}$ over calendar years t. It is obtained by a q-dimensional embedding of (t, i)

$$(t, i) \mapsto \boldsymbol{k}_t^{(i)} \in \mathbb{R}^q. \tag{4}$$

- $(\boldsymbol{b}_x^{(i)})_x$ describes the rate of change of force of mortality broken down to the different ages of population $i \in \mathcal{I}$. It is obtained by another q-dimensional embedding of (x, i)

$$(x, i) \mapsto \boldsymbol{b}_x^{(i)} \in \mathbb{R}^q. \tag{5}$$

By suitably defining the embedding dimension $q \in \mathbb{N}$ and the nature of the dependence on i, one can recover the LC model and variations of this model as follows:

- Classical LC model (Lee & Carter 1992) for individual populations $i \in \mathcal{I}$: choose $q = 1$ and estimate the parameters $a_x^{(i)}, \boldsymbol{b}_x^{(i)}, \boldsymbol{k}_t^{(i)} \in \mathbb{R}$ individually for each population $i \in \mathcal{I}$.
- Increasing q produces a variation on the LC model with multiple time effects, see Renshaw & Haberman (2003).
- CAE model Kleinow (2015): choose change of force of mortality not depending on the chosen population $i \in \mathcal{I}$, i.e. set $\boldsymbol{b}_x^{(i)} = \boldsymbol{b}_x \in \mathbb{R}^q$ for all $i \in \mathcal{I}$. Parameter estimation is done simultaneously over all populations $i \in \mathcal{I}$.
- ACF model Li & Lee (2005): choose $q = 2$, and the two-dimensional parameters have a population-independent and a population-dependent term, i.e. set $\boldsymbol{b}_x^{(i)} = (b_x, b_x^{(i)})' \in \mathbb{R}^2$ and $\boldsymbol{k}_t^{(i)} = (k_t, k_t^{(i)})' \in \mathbb{R}^2$. Parameter estimation is done simultaneously over all populations $i \in \mathcal{I}$.
- The parameters $a_x^{(i)} \in \mathbb{R}$ and $\boldsymbol{b}_x^{(i)} \in \mathbb{R}^q$ in (3) and (5) reflect embeddings of categorical variables (categorical encoding of the continuous age variable x). The Hyndman and Ullah (HU) model (Hyndman & Ullah 2007) replaces this categorical encoding of the age variable x by functional forms building an orthonormal basis, see (2) in Hyndman & Ullah (2007); in their application they use smoothing B-splines for modeling this functional form in the age variable x.

In this paper, we try to keep the intuition of the LC model structure as far as possible, but allow for potentially more complex interactions beyond the scalar product in (2) (on the log-scale). These interactions are learned using neural networks. Furthermore, in accordance with our intuition that networks designed to process sequential data might be more efficient than regression models of the form of (1), we replace the mapping in (4) by

$$U^{(i)} \mapsto \boldsymbol{k}_t^{(i)} \in \mathbb{R}^q. \tag{6}$$

In other words, we look to map (matrices of) past raw mortality rates (or rather histories) directly to a vector of time indices to enable forecasting of future mortality rates. We confine ourselves to models

that forecast a single year ahead such that we have $t = t_1 + 1$ in (6), where t_1 is the last observation point matrix $U^{(i)}$. This is important to maintain time causality. In the following sections, we explore models that produce robust estimation of mapping (6).

3. Neural network building blocks

Neural Networks (NNs) provide highly flexible regression models that have recently achieved great success on complex machine learning tasks such as image recognition or text processing, as well as for actuarial modeling. Compared to most other supervised regression models, NNs are distinguished by implementing the paradigm of representation learning (Bengio et al. 2013), meaning to say that NNs automatically construct, layer by layer, new sets of features from the input data to make optimal predictions of target variables. NNs usually consist of a set of nonlinear functions arranged in successive layers and in a way in which the input variables propagate through the layers to the output variables, as we describe next. In most modern NN architectures, the input variables propagate along the network's hidden layers in only one direction: from the input to the output layer (whereas in other types of NNs, the information may propagate in both directions); these former networks are called feed-forward NNs. In this section, we revisit four main types of NN architectures that are often used: feed-forward Fully Connected Networks (FCNs), embedding layers, Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs).

3.1. Feed-forward fully connected neural networks

FCNs are those networks in which each layer is connected to every part of the previous layer. Formally, let $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)' \in \mathcal{X} \subset \mathbb{R}^d$ be a d-dimensional input vector, then a FCN layer with $q \in \mathbb{N}$ hidden units is a function that maps the input $\boldsymbol{x}$ to a new q-dimensional space

$$\boldsymbol{z} : \mathbb{R}^d \to \mathbb{R}^q, \quad \boldsymbol{x} \mapsto \boldsymbol{z}(\boldsymbol{x}) = (z_1(\boldsymbol{x}), z_2(\boldsymbol{x}), \ldots, z_q(\boldsymbol{x}))'. \tag{7}$$

Each new feature component $z_j(\boldsymbol{x})$ is a nonlinear function of $\boldsymbol{x}$

$$\boldsymbol{x} \mapsto z_j(\boldsymbol{x}) = \phi\left(w_{j,0} + \sum_{l=1}^{d} w_{j,l} x_l\right) = \phi(w_{j,0} + \langle \boldsymbol{w}_j, \boldsymbol{x} \rangle), \quad j = 1, \ldots, q, \tag{8}$$

where $\langle \cdot, \cdot \rangle$ denotes the scalar product in $\mathbb{R}^d$, $\phi : \mathbb{R} \to \mathbb{R}$ is a (non linear) activation function (often sigmoid, hyperbolic tangent or rectified linear unit (ReLU) function), and $w_{j,l} \in \mathbb{R}$ represent the parameters of the FCN layer that are learned from the data. In network terminology, the intercepts $w_{j,0}$ are referred to as the biases of the layer, and the regression parameters $\boldsymbol{w}_j = (w_{j,1}, \ldots, w_{j,d})' \in \mathbb{R}^d$ are referred to as the weights. Fully connected requires that all weights $w_{j,l} \neq 0$.

A NN may be composed solely of FCN layers. In a deep FCN multiple FCN layers (7) are composed. Formally, let $m \in \mathbb{N}$ be the number of hidden layers (depth of network), and $q_k \in \mathbb{N}$, for $1 \leq k \leq m$, be a sequence of integers that indicates the dimension of each FCN layer (widths of layers). Then, a deep FCN can be formalized as

$$\boldsymbol{x} \mapsto \boldsymbol{z}^{(m:1)}(\boldsymbol{x}) = \left(\boldsymbol{z}^{(m)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}) \in \mathbb{R}^{q_m}, \tag{9}$$

where all mappings $\boldsymbol{z}^{(k)} : \mathbb{R}^{q_{k-1}} \to \mathbb{R}^{q_k}$ are of structure (7) with weights $W^{(k)} = (\boldsymbol{w}_j^{(k)})_{1 \leq j \leq q_k} \in \mathbb{R}^{q_k \times q_{k-1}}$ and biases $\boldsymbol{w}_0^{(k)} \in \mathbb{R}^{q_k}$, for $1 \leq k \leq m$. Moreover, we initialize input dimension $q_0 = d$ for feature $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$. Note that also activation functions ϕ_k may differ between the layers. Finally, we need to choose an output map g (inverse link function) that maps the activations $\boldsymbol{z}^{(m:1)}(\boldsymbol{x})$ in the last hidden layer to predictions $g(\boldsymbol{z}^{(m:1)}(\boldsymbol{x}))$ for response variables Y.

The performance of a deep FCN depends on the choice of the weights and biases in each layer, that must be properly calibrated or 'trained' to the prediction task. The first step of the training process is to specify a suitable objective function $L(g(z^{(m:1)}(x)), y)$, which measures the quality of the predictions produced by the network $g(z^{(m:1)}(x))$ against the observed values y of the response variable Y. A common example of an objective function is the mean squared error (MSE), its minimization is equivalent to maximum likelihood estimation (MLE) under a (conditional) Gaussian model for Y. NN training is carried out using the Back-Propagation (BP) algorithm where the weights are updated iteratively to step-wise decrease the objective function, with each update of the weights based on the gradient of the loss function. An extensive description of network fitting and the BP algorithm is found in Goodfellow et al. (2016) and Hastie et al. (2005).

3.2. Embedding layers

Insurance data often presents qualitative features expressed on an ordinal or nominal scale (categorical variables). In the context of mortality modeling, an example is the population index $i \in \mathcal{I}$ for which mortality rates are being forecast. Dummy coding or one-hot encoding are popular approaches to deal with categorical variables among statisticians and the machine learning community, respectively. However, when observations present a large number of categorical variables or when the variables have many different levels, these coding schemes produce high dimensional sparse vectors, which often causes computational issues and poor models.

A different technique, originally applied in natural language processing (Bengio et al. 2003), is rather to map categorical variables to low-dimensional vector spaces. These mappings can also be learned using BP for network calibration; see, for example, Guo & Berkhahn (2016). They are trained to have proximity (in the vector space) if the labels are similar for the regression task, and distant otherwise.

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_{n_{\mathcal{P}}}\}$ be the (finite) set of categories (labels or levels) and $n_{\mathcal{P}} = |\mathcal{P}|$ the cardinality of the categorical variable. An embedding layer is a mapping

$$z_{\mathcal{P}} : \mathcal{P} \to \mathbb{R}^{q_{\mathcal{P}}}, \quad p \mapsto z_{\mathcal{P}}(p),$$

where $q_{\mathcal{P}} \in \mathbb{N}$ is a hyper-parameter defining the dimension of the embedding layer for set $\mathcal{P}$. In other words, $z_{\mathcal{P}}(\cdot)$ maps each element of $\mathcal{P}$ to a real-valued $q_{\mathcal{P}}$-dimensional vector being a numerical encoding of the chosen label. Typically, $q_{\mathcal{P}} \ll n_{\mathcal{P}}$, and the number of embedding weights that must be learned during training is given by $n_{\mathcal{P}} q_{\mathcal{P}}$.

3.3. Recurrent neural networks

3.3.1. Plain vanilla recurrent neural networks

RNNs are designed for processing sequential data, where the values computed for each observation depend on previous parts of the sequence. For an example in the context of mortality modeling, FCNs treat consecutive inputs such as (t, x, i) and $(t + 1, x, i)$ fully independently, whereas RNNs would process the second input by maintaining an internal state that is related, in some way, to the information of the first input. This is done by providing additional connections (compared to FCNs) that link the hidden layers of the network cyclically, see Elman (1990). Thus, the values assumed by the hidden layers of the network influence the output of the network in future evaluations of inputs, in theory making RNN architectures a powerful tool for analyzing sequential data such as time-series data.

Let $x_t \in \mathbb{R}^d, 0 \le t \le T$, be multivariate time-series data used as input to predict a response variable Y. An RNN with only one hidden layer ($m = 1$) is a mapping

$$z : \mathbb{R}^{d \times q} \to \mathbb{R}^q, \quad (x_t, z_{t-1}) \mapsto z_t = z(x_t, z_{t-1}) = (z_1(x_t, z_{t-1}), \quad z_2(x_t, z_{t-1}), \ldots, z_q(x_t, z_{t-1}))',$$

where the output at time t of the jth unit $z_{t,j} = z_j(x_t, z_{t-1})$ is given by

$$z_{t,j} = \phi\left(w_{j,0} + \sum_{l=1}^{d} w_{j,l} x_{t,l} + \sum_{k=1}^{q} v_{j,k} z_{t-1,k}\right) = \phi(w_{j,0} + \langle w_j, x_t \rangle + \langle v_j, z_{t-1} \rangle), \quad j = 1, \ldots, q,$$

with $v_{j,k} \in \mathbb{R}$ representing the weights associated with the previous output of the network z_{t-1} (initializing $z_0 = \mathbf{0} \in \mathbb{R}^q$). Compared to (7)–(8) we receive a time-series structure using the previous cell state z_{t-1} as input to predict the next output z_t. Denoting with $W = (w_j)_{1 \leq j \leq q} \in \mathbb{R}^{q \times d}$ and $V = (v_j)_{1 \leq j \leq q} \in \mathbb{R}^{q \times q}$ the weight matrices, and with $w_0 \in \mathbb{R}^q$ the biases.

It is important to remark that parameter matrices W and V are time independent. Training of RNNs is carried out using the Back Propagation Through Time (BPTT) algorithm which, however, presents several problems related to slow convergence and vanishing gradients. To avoid these problems, two more sophisticated proposals for RNN architectures were made: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks, see Hochreiter & Schmidhuber (1997) and Cho et al. (2014). These two architectures specify more complicated ways of updating the state of the network. This paper focuses on the LSTM architecture, which is slightly more complex but often more robust than GRU architectures.

3.3.2. Long short-term memory networks

In addition to the recurrent component z_t discussed above, LSTM networks (Hochreiter & Schmidhuber 1997), are equipped with a so-called 'memory' cell that stores and releases long-term information through a system of sub-networks called gates. Combining these two types of memory cells, LSTM networks have shown excellent performance in different applications involving the processing of sequential data, such as handwriting recognition (Graves & Schmidhuber 2009) and speech recognition (Graves & Schmidhuber 2005). Introducing this memory cell for long-term information can also be motivated by the findings of Yan et al. (2020) proving the existence of long memory in mortality data.

Different formulations of LSTM networks have been proposed and we refer the reader to the extended overview of various types of LSTM architectures in Greff et al. (2016). Briefly, the initial version of the LSTM network included input and output gates that control the memory cell; the first gate extracts the relevant information to pass to the memory cell, and the output gate filters the information relevant to predicting the output. In Gers et al. (2000), the authors proposed to insert an additional sub-network, called the forget gate, that erases the obsolete information in the memory cell. Mathematically, denoting respective the weight matrices and bias vectors with $W^{(s)} \in \mathbb{R}^{q \times d}$, $V^{(s)} \in \mathbb{R}^{q \times q}$ and $w_0^{(s)} \in \mathbb{R}^q$ associated with each gate indexed by $s \in \{i, o, f, r\}$, the output z_t at time t of the LSTM layer, given input (x_t, z_{t-1}), is described as follows:

$$f_t = f_t(x_t, z_{t-1}) = \sigma(w_0^{(f)} + \langle W^{(f)}, x_t \rangle + \langle V^{(f)}, z_{t-1} \rangle) \in \mathbb{R}^q, \tag{10}$$

$$i_t = i_t(x_t, z_{t-1}) = \sigma(w_0^{(i)} + \langle W^{(i)}, x_t \rangle + \langle V^{(i)}, z_{t-1} \rangle) \in \mathbb{R}^q, \tag{11}$$

$$o_t = o_t(x_t, z_{t-1}) = \sigma(w_0^{(o)} + \langle W^{(o)}, x_t \rangle + \langle V^{(o)}, z_{t-1} \rangle) \in \mathbb{R}^q, \tag{12}$$

$$r_t = r_t(x_t, z_{t-1}) = \tanh(w_0^{(r)} + \langle W^{(r)}, x_t \rangle + \langle V^{(r)}, z_{t-1} \rangle) \in \mathbb{R}^q, \tag{13}$$

$$c_t = c_t(x_t, z_{t-1}, c_{t-1}) = f_t \odot c_{t-1} + i_t \odot r_t \in \mathbb{R}^q, \tag{14}$$

$$z_t = z_t(x_t, z_{t-1}, c_{t-1}) = o_t \odot \phi(c_t) \in \mathbb{R}^q, \tag{15}$$

where $\odot$ denotes the Hadamard product and σ is the sigmoid activation function. c_t represents the memory cell at time t (with initial value $c_0 = \mathbf{0} \in \mathbb{R}^q$), which is regulated by forget f_t, input i_t and new cell input r_t, respectively. Chronologically, in each time step t, the first gate that acts on the memory cell is the forget gate (10) deleting obsolete information from the memory cell c_{t-1} (14).

Next, the input gate (11) adds new information extracted from the current cell input r_t (13) to update the memory cell (14). Finally, combining the result of the output gate (12) and the updated memory cell c_t, the output value of the LSTM cell is calculated (15).

In most applications of LSTM networks, the network is trained so that the final output of the network z_T matches the response variable Y_T. An alternative is to use the full time-series of outputs $(z_t)_{1 \leq t \leq T}$ as input into a FCN to make predictions.

3.4. Convolutional neural networks

CNNs, proposed in LeCun et al. (1990), consist of a class of networks specialized to process data that has a grid-like or spatial structure such as images. More recently, CNNs have been applied to natural language processing and time-series data, see, for example, Borovykh et al. (2017). Compared to RNNs, CNNs are often significantly faster and easier to train, and thus, CNNs may be considered as a replacement for RNNs in some applications.

CNNs are named due to the use of convolutions to extract new features by convolving the data with a filter, the weights of which are learned during training. Typically, CNNs work on tensors and the number of dimensions to which the convolution operator is applied to is either one-dimensional (for time-series), two-dimensional (for surface structures) or three-dimensional (for spatial structures). The main characteristics differentiating CNNs from FCNs are discussed in Goodfellow et al. (2016). Firstly, whereas FCNs connect the neurons of each layer to all of the neurons in the previous layer, CNNs only connect neurons to a local region of the input array. This reduces significantly the number of parameters that must be learned. Secondly, weights are shared across local regions by applying the same filter to the entire input array. In addition to decreasing the number of parameters further, this also allows for the extraction of similar features in different regions of the input array, we refer to Wiatowski & Bölcskei (2018) for statements about translation invariance in CNNs.

For simplicity, we only describe a one-dimensional CNN which is adequate for mortality forecasting. The main difference to a FCN layer is that the scalar product in (8) is replaced by a convolution. Similar to Section 3.3, we let $x_t \in \mathbb{R}^d$, $0 \leq t \leq T$, be a multivariate time-series of input variables. A 1d-CNN layer with filters $W^{(j)} = (w_{j,1}, w_{j,2}, \ldots, w_{j,m}) \in \mathbb{R}^{d \times m}$ and biases $w_{j,0} \in \mathbb{R}$, for $j = 1, \ldots, q$, is a mapping (we keep the '1' in the following notation to highlight that this is a 1d-CNN layer)

$$z : \mathbb{R}^{1 \times (T+1) \times d} \to \mathbb{R}^{1 \times (T+2-m) \times q}, \quad x = (x_0, \ldots, x_T) \mapsto z(x) = (z_{s,j}(x))_{0 \leq s \leq T+1-m, 1 \leq j \leq q}, \quad (16)$$

with each component given by (the symbol $\star$ illustrates the convolution operator)

$$x \mapsto z_{s,j} = z_{s,j}(x) = \phi(w_{j,0} + (W^{(j)} \star x)_s) = \phi\left(w_{j,0} + \sum_{l=1}^{m} \langle w_{j,l}, x_{m+s-l} \rangle\right). \quad (17)$$

The convolution in (17) is applied locally, note that the filter has a domain of maximal size m, and we apply the same filter over the entire array of input variables x because we always use the same weights and biases in the filters.

The input dimension of one time-series component x_t is d, and the CNN layer provides outputs $z_s(x) = (z_{s,1}(x), \ldots, z_{s,q}(x))' \in \mathbb{R}^q$ for $0 \leq s \leq T+1-m$. Thus, d-dimensional time-series $(x_t)_t$ of length $T+1$ are transformed to a q-dimensional time-series $(z_s)_s$ of length $T+2-m$ by a CNN layer (16). Since we typically apply filters of size $m > 1$, the length of the time series reduces from T to $T+1-m$. If this is an undesired property, padding at the boundaries is applied, we refer to the literature for more details on padding. For higher-dimensional CNN inputs, say, of dimension r, one changes the CNN layer (16) to $z : \mathbb{R}^{r \times (T+1) \times d} \to \mathbb{R}^{r \times (T+2-m) \times q}$ with components $z_{s,j} = z_{s,j}(x) \in \mathbb{R}^r$. In complete analogy to deep FCNs (9), composition of CNN layers provides deep CNNs.

Applications of CNN layers are often followed by so-called pooling layers. These pooling layers can be best understood as being CNN filters that, instead of learning parameters to convolve with

the input, rather perform simple operations such as considering the maximal, minimal or average input component within the domain of the filter. These layers are used to aggregate features of high-dimensional layers to generate smaller dimensional feature structure. Note that, eventually, we try to extract and compress information to make it valuable for predictions, and pooling layers try to extract the most dominant part of information seen within a given domain. For more information, we refer to Goodfellow et al. (2016).

3.5. Summary on network layers

We mention briefly some advantages and disadvantages of the RNN and CNN components for time-series forecasting. RNNs have shown excellent performance in tasks such as time-series forecasting and sequence modeling, however, Richman & Wüthrich (2019) showed that when forecasting Swiss mortality data, RNNs produced highly variable results that were sometimes worse than a LC baseline model. On the other hand, one-dimensional CNNs have recently replaced RNNs in many applications involving sequence modeling, since networks with CNN components are significantly easier to train. In the following section, we mention some of the network structures using RNNs and CNNs that we experimented with. Finally, we mention that due to the flexibility of these models, it is necessary to apply regularization to achieve good predictive performance. In the following, we use dropout (Srivastava et al. 2014) and batch normalization (Ioffe & Szegedy 2015) to regularize the networks, as well using batch normalization to make optimization of networks easier.

4. Neural network regression models for mortality forecasting

4.1. Overview of the network architectures considered

As emphasized in Section 2, we aim at processing directly the recent mortality experience of a population to produce forecasts, see (6). In general terms, to accomplish this, we directly process the mortality experience $U^{(i)}$ of population $i \in \mathcal{I}$ through either a RNN or a CNN. While mortality rates could be forecast directly using these networks, nonetheless, information about the considered population i is also included to the forecasts (which makes it reasonable to use the same network model simultaneously over all populations). Our solution is to combine the output of RNNs or CNNs with population-specific feature information, which we model using the embedding layers for gender and region code.

While this approach to forecasting mortality is appealing, intuitively, nonetheless we found that many solutions that were considered to implement the approach failed to achieve good predictive performance. Before defining the specific solution used in this paper, we briefly give some of the options that were tried (using the process for investigating the performance of a network as discussed in Section 5) and we mention why these approaches might have failed. First, relatively deep networks consisting of multiple RNN or CNN layers were considered to process the mortality experience, but these architectures were too difficult to be successfully calibrated. Second, the output of the RNNs or CNNs was initially processed using several FCN layers, and these networks also did not have good predictive performance due to poor calibration behavior. In both of these cases, processing mortality rates for all ages at once, or for each age at a time, did not make a big difference for performance. Finally, we tried several methods of producing multi-year forecasts with these models.

On the one hand, it is somewhat surprising that highly flexible models were unable to model mortality data along the lines of Equation (6). On the other hand, we found that a much simpler shallow (one hidden layer) network model, which we present next, exhibits good predictive performance on the HMD data sets. Although a rigorous explanation for this has not yet been formulated, nonetheless, it is fair to say that these findings indicate that mortality rate forecasting does not need the highly complex (hierarchical) representations of the input data, which are produced by deep networks, for successful forecasts, perhaps due to the relatively simple manner in which mortality rates have recently

Table 1. LC network mortality forecasting models considered.

Model	
LCCONV	CNN layer according to Listing 1
LCCONV_tanh	– *tanh* activation in CNN layer
LCCONV_relu	– ReLU activation in CNN layer
LCLSTM1	LSTM layer according to Listing 2
LCLSTM1_tanh	– *tanh* activation in LSTM layer
LCLSTM1_relu	– ReLU activation in LSTM layer
LCLSTM2	LSTM layer according to Listing 3
LCLSTM2_tanh	– *tanh* activation in LSTM layer
LCLSTM2_relu	– ReLU activation in LSTM layer

been observed to change over time. In other words, in the absence of major shocks (such as a pandemic) which were not observed until the end of 2019, mortality rates have decreased at a fairly steady rate since the end of the Second World War.

Due to the relatively extensive experiments that were performed on the HMD data to find good models, the extent to which our results would generalize to other data sets became somewhat unclear, as we may have overfit to the HMD data. Thus, as described in the following section, to validate the generalization of our approach, we applied it to the United States Mortality Database (USMD), without changing any parameter settings.

In the following section, we introduce our forecasting model: we define the LC convolutional mortality forecasting model, in short LCCONV. In our model description, we focus on the convolutional model and variants thereof are summarized in Table 1. Furthermore, we provide our code using the R language implementation of Keras (Chollet 2018). Finally, in Section 4.3, we show how the model can be regarded as a type of LC model that preserves the essential structure of (2).

4.2. Defining the Lee–Carter convolutional mortality forecasting model

In Section 2, we considered all population information as being summarized by $i \in \mathcal{I}$. Here, we use a more detailed specification for population consisting of a region and a gender code. These are two static categorical variables that are modeled using embedding layers, see Section 3.2. Let $\mathcal{R} = \{r_1, r_2, \ldots, r_N\}$ be the set of all N regions under investigation and let $\mathcal{G} = \{\texttt{male}, \texttt{female}\}$ be the set of genders, such that the static information can be summarized by $\mathcal{I} = \mathcal{G} \times \mathcal{R}$. Embedding layers map $r \in \mathcal{R}$ and $g \in \mathcal{G}$ to two real-valued vectors

$$z_{\mathcal{R}} : \mathcal{R} \to \mathbb{R}^{n_{\mathcal{R}}}, \quad r \mapsto z_{\mathcal{R}}(r),$$
$$z_{\mathcal{G}} : \mathcal{G} \to \mathbb{R}^{n_{\mathcal{G}}}, \quad g \mapsto z_{\mathcal{G}}(g),$$

where $n_{\mathcal{R}}, n_{\mathcal{G}} \in \mathbb{N}$ are two hyper-parameters determining the embedding dimensions, these embeddings are described on lines 3-6 and 8–11 of Listing 1. In our application, we choose $n_{\mathcal{R}} = n_{\mathcal{G}} = 5$. The choice of $n_{\mathcal{G}} = 5$ for gender could perhaps be questioned on the grounds that gender only has two levels within the data. In preparatory work, we have seen that using a larger dimension for gender embedding improves performance in mortality forecasting, because this allows us for more flexible interactions in gradient descent calibrations between gender, age and region.

For the matrix of mortality inputs to the network, we define $U^{(i)}$ to cover ages 0–99 and 10 years of data, i.e. we choose $x_0 = 0, x_1 = 99, T = 9, t_1 = t_0 + T$ and $d = x_1 - x_0 + 1 = 100$, this provides us with observations (note that we transpose to bring dimensions in line with CNN layers, see (16), and we add a time index t_0 to illustrate the start of the observation period)

$$U_{t_0}^{(i)} = (u_{x,t}^{(i)})'_{x_0 \leq x \leq x_1, t_0 \leq t \leq t_1 = t_0 + T} = (\boldsymbol{u}_{t_0}^{(i)}, \ldots, \boldsymbol{u}_{t_1}^{(i)})' \in \mathbb{R}_+^{1 \times (T+1) \times d} = \mathbb{R}_+^{1 \times 10 \times 100}.$$

Listing 1 Deep neural network model LCCONV with a 1d-CNN layer.

```
1   rates <- layer_input(shape = c(10,100), dtype = 'float32', name = 'rates')
2   #
3   Country <- layer_input(shape = c(1), dtype = 'int32', name = 'Country')
4   Country_embed=Country %>%
5       layer_embedding(input_dim = N, output_dim = 5) %>%
6       layer_flatten(name= 'Country_embed')
7   #
8   Gender <- layer_input(shape = c(1), dtype = 'int32', name = 'Gender')
9   Gender_embed = Gender %>%
10      layer_embedding(input_dim = 2, output_dim = 5) %>%
11      layer_flatten(name= 'Gender_embed')
12  #
13  conv = rates %>% layer_conv_1d(filter = 32, kernel_size = 3,
14                          activation = 'linear', padding = "causal") %>%
15      layer_max_pooling_1d(pool_size = 2) %>%
16      layer_batch_normalization() %>%
17      layer_dropout(rate = 0.35) %>%
18      layer_flatten()
19  #
20  decoded = conv %>% list(Country_embed, Gender_embed)%>%
21      layer_concatenate() %>%
22      layer_dropout(rate = 0.4) %>%
23      layer_dense(units = 100, activation = 'sigmoid') %>%
24      layer_reshape(c(1,100), name = 'forecast_rates')
25  #
26  model <- keras_model(inputs = list(rates, Country, Gender), outputs = c(decoded))
```

As discussed above, the matrix $U_{t_0}^{(i)}$ is processed using network layers specifically designed to process sequential data via RNNs and CNNs. For illustration we choose the CNN version in this section. This requires two hyper-parameters; we choose the number of filters $q = 32$ with filter sizes $m = 3$. A 1d-CNN with q filters $W^{(j)} = (\boldsymbol{w}_{j,1}, \boldsymbol{w}_{j,2}, \boldsymbol{w}_{j,m=3}) \in \mathbb{R}^{d \times m}$ and biases $w_{j,0} \in \mathbb{R}$, for $j = 1, \ldots, q$, maps the observations $U_{t_0}^{(i)}$ into a new feature space

$$\boldsymbol{z}^{(1)} : \mathbb{R}^{1 \times (T+1) \times d} \to \mathbb{R}^{1 \times (T+1) \times q}, \quad U_{t_0}^{(i)} \mapsto \boldsymbol{z}^{(1)} = \boldsymbol{z}^{(1)}(U_{t_0}^{(i)}),$$

according to a CNN layer given as in (16–17). This is exactly described by lines 13–14 of Listing 1 (using padding). Next, a 1d max-pooling layer of size 2 is applied to the output of the CNN layer $\boldsymbol{z}^{(1)}(U_{t_0}^{(i)})$, see line 15 of Listing 1. This 1d max-pooling layer extracts the maximum of neighboring pairs of components in $\boldsymbol{z}^{(1)}(U_{t_0}^{(i)})$. This step halves the size of the features

$$\boldsymbol{z}^{(2)} : \mathbb{R}^{(T+1) \times q} \to \mathbb{R}^{\lceil (T+1)/2 \rceil \times q}, \quad \boldsymbol{z}^{(1)}(U_{t_0}^{(i)}) \mapsto \boldsymbol{z}^{(2)}(\boldsymbol{z}^{(1)}(U_{t_0}^{(i)})) = (\boldsymbol{z}^{(2)} \circ \boldsymbol{z}^{(1)})(U_{t_0}^{(i)}).$$

Then, batch normalization and dropout are applied (the latter only acts during model training), see lines 16–17 of Listing 1, and then on line 18 we reshape the resulting array of $(\boldsymbol{z}^{(2)} \circ \boldsymbol{z}^{(1)})(U_{t_0}^{(i)})$ to a vector $\boldsymbol{z}_f = \boldsymbol{z}_f(U_{t_0}^{(i)}) \in \mathbb{R}^{n_f}$ obtained by concatenating all elements of $(\boldsymbol{z}^{(2)} \circ \boldsymbol{z}^{(1)})(U_{t_0}^{(i)})$ into a single vector. As a result the mortality history $U_{t_0}^{(i)} \in \mathbb{R}_+^{(T+1) \times d}$ is *encoded* in a vector $\boldsymbol{z}_f(U_{t_0}^{(i)})$ of dimension $n_f = q \lceil (T+1)/2 \rceil$.

Finally, all feature information related to region $z_{\mathcal{R}}(r)$, to gender $z_{\mathcal{G}}(g)$ and to mortality history $\boldsymbol{z}_f(U_{t_0}^{(i)})$ is combined to create a new representation $\boldsymbol{z}(r, g, U_{t_0}^{(i)}) = (z_{\mathcal{R}}(r), z_{\mathcal{G}}(g), \boldsymbol{z}_f(U_{t_0}^{(i)})) \in \mathbb{R}^{n_{\mathcal{R}} + n_{\mathcal{G}} + n_f}$ of the data, this is done on lines 20–21 of Listing 1.

Mortality rates are then calculated by *decoding* the information $\boldsymbol{z}(r, g, U_{t_0}^{(i)})$ using a multivariate generalized linear model (GLM). For this purpose we need to explain how we choose the response

variables. We define the response variable by applying the MinMaxScaler to the log-mortality rates such that they are within $[0, 1]$, thus, we set

$$y_{t_0+T+1}^{(i)} = \frac{\log(u_{t_0+T+1}^{(i)}) - y_0}{y_1 - y_0} \in [0, 1]^d,$$

where $y_0 = \min_{\{x,t_0,i\}} \log(u_{x,t_0+T+1}^{(i)})$ and $y_1 = \max_{\{x,t_0,i\}} \log(u_{x,t_0+T+1}^{(i)})$ are the minimum and maximum log-mortality rates observed over the entire training data. These responses are then predicted by the following multivariate GLM (using a FCN layer), see lines 20–24 of Listing 1,

$$(r, g, U_{t_0}^{(i)}) \mapsto \widehat{y}_{t_0+T+1}^{(i)} = \widehat{y}(r, g, U_{t_0}^{(i)}) = \sigma(w_0 + \langle W, z(r, g, U_{t_0}^{(i)}) \rangle) \in [0, 1]^d, \tag{18}$$

where $w_0 \in \mathbb{R}^d$ is the vector of the intercepts, $W \in \mathbb{R}^{d \times (n_{\mathcal{R}} + n_{\mathcal{G}} + n_f)}$ is the weight matrix, and all operations are understood component-wise. The latter can also be re-written in the following unscaled version:

$$\log(\widehat{u}_{t_0+T+1}^{(i)}) = y_0 + (y_1 - y_0)\widehat{y}_{t_0+T+1}^{(i)} = y_0 + (y_1 - y_0)\sigma(w_0 + \langle W, z(r, g, U_{t_0}^{(i)}) \rangle) \in \mathbb{R}^d. \tag{19}$$

Finally, we discuss briefly how the model is used to produce multi-year forecasts. Since the model requires the matrix of mortality rates $U_{t_0}^{(i)}$ as an input, we need to calculate multi-year forecasts recursively. In particular, if we forecast year $t_0 + T + 1 + \tau = t_1 + 1 + \tau$ for $\tau > 0$ we use mortality rates

$$\widehat{U}_{t_0+\tau}^{(i)} = (u_{t_0+\tau}^{(i)}, \dots, u_{t_1}^{(i)}, \widehat{u}_{t_1+1}^{(i)}, \dots, \widehat{u}_{t_1+\tau}^{(i)})' \in \mathbb{R}_+^{1 \times (T+1) \times d}$$

as feature information, where the former entries have been observed at time t_1, and the latter entries are recursively obtained predicted values after time t_1.

4.3. Interpreting the Lee–Carter convolutional mortality forecasting model

We interpret this model as a variation of the LC concept in (2). We focus on a single age component x in (19) with static population information $i = (r, g)$

$$\widehat{y}_{x,t_0+T+1}^{(r,g)} = \sigma(w_{x,0} + \langle W_x, z(r, g, U_{t_0}^{(i)}) \rangle). \tag{20}$$

We first decouple the scalar product in (20) based on the weight $W_x = (W_x^{\mathcal{R}}, W_x^{\mathcal{G}}, W_x^f) \in \mathbb{R}^{n_{\mathcal{R}} + n_{\mathcal{G}} + n_f}$. All population and time-specific information is contained in

$$z(r, g, U_{t_0}^{(i)}) = (z_{\mathcal{R}}(r), z_{\mathcal{G}}(g), z_f(U_{t_0}^{(i)})) \in \mathbb{R}^{n_{\mathcal{R}} + n_{\mathcal{G}} + n_f}.$$

This allows us to rewrite formula (20) as follows:

$$\sigma^{-1}(\widehat{y}_{x,t_0+T+1}^{(r,g)}) = w_{x,0} + \langle W_x^{\mathcal{R}}, z_{\mathcal{R}}(r) \rangle + \langle W_x^{\mathcal{G}}, z_{\mathcal{G}}(g) \rangle + \langle W_x^f, z_f(U_{t_0}^{(i)}) \rangle. \tag{21}$$

We can now give the right interpretation in the light of LC concept (2). The first three terms $w_{x,0} + \langle W_x^{\mathcal{R}}, z_{\mathcal{R}}(r) \rangle + \langle W_x^{\mathcal{G}}, z_{\mathcal{G}}(g) \rangle$ play the role of the $a_x^{(i)}$ parameters of the LC model, see (2). We have a population independent force of mortality $w_{x,0}$ which is corrected by population specific terms $\langle W_x^{\mathcal{R}}, z_{\mathcal{R}}(r) \rangle$ and $\langle W_x^{\mathcal{G}}, z_{\mathcal{G}}(g) \rangle$.

Time dependence comes in through the last term. Similar to the term $\langle b_x^{(i)}, k_t^{(i)} \rangle$ in the LC model, we have a population independent vector W_x^f playing the role of $b_x^{(i)}$, and the change of force of mortality $k_t^{(i)}$ is encoded in the time-dependent vector $z_f(U_{t_0}^{(i)})$. Concluding, we can assign to all our terms a LC interpretation; the main difference of our approach is that we directly encode mortality

inputs $U_{t_0}^{(i)}$ through a CNN layer to receive time dependence. This exactly illustrates our initial motivation (6) in receiving a change of force of mortality $k_t^{(i)}$ directly from feature engineering the raw mortality inputs $U_{t_0}^{(i)}$.

Remark 4.1:　• The right-hand side of (21) is linear in $z_{\mathcal{R}}(r)$, $z_{\mathcal{G}}(g)$ and $z_f(U_{t_0}^{(i)})$. We show later that introducing nonlinear functions, here, generally degrades the predictive performance, especially when ReLU activations are used. This is likely due to the relative simplicity of how mortality rates change over time.
- The standard LC model cannot be fit with standard GLM techniques since the time varying parameters k_t of the LC model are not observed in the data, in other words, k_t are latent parameters that must first be estimated using, for example, principal component analysis. In the network model presented here, generalized k_t parameters $z_f(U_{t_0}^{(i)})$ are first determined by the CNN layer of the network. More precisely, we learn a representation of the mortality input $U_{t_0}^{(i)}$. This learned representation is then directly used in GLM (18).
- In the introduction, we noted that the LC model, and many other mortality forecasting models, are usually fit in two stages. First, the data is transformed into a parsimonious representation a_x, b_x and k_t, and then the parameters k_t are extrapolated using time-series methods. Because of the two-stage process, if a change in the mortality experience is observed, for example in the rate of mortality improvement, then both stages of the fitting process must be re-performed. This must be done on data that has been (subjectively) selected such that the model fit to these data will reflect the expected future mortality experience correctly. On the other hand, in common with most modern network models, which apply the paradigm of representation learning, the network model presented here combines the data transformation and forecasting steps in a single step. Therefore, as new data is observed, this can be processed through the network model (without refitting the model) and an updated forecast will be produced that takes into account any changes in the mortality experience automatically (at least to some extent). The fitted network thus produces a locally calibrated forecast each time it processes a new set of mortality data, meaning to say, the forecasts are calibrated based on mortality rates in a given small window.
- Similar to the LC model, regression function (21) can be seen in relation to the HU model (Hyndman & Ullah 2007). As described in Section 2, the HU model replaces the PCA methodology of LC by a functional version thereof where the age variable x is modeled through basis functions (we drop the error terms here and we fix on single population i)

$$\log(u_{x,t}^{(i)}) \approx \mu^{(i)}(x) + \langle \boldsymbol{\psi}(x), k_t^{(i)} \rangle,$$

where $\boldsymbol{\psi} = (\psi_1, \ldots, \psi_q)$ denotes a set of orthonormal basis functions. The main differences of the HU model to (21) are that (1) we directly process mortality observations $U_{t_0}^{(i)}$ to receive $z_f(U_{t_0}^{(i)})$ playing the role of $k_t^{(i)}$, (2) we use a categorical encoding of the age variable through W_x^f whereas the HU model uses the functional paradigm using the orthonormal basis $\boldsymbol{\psi}$. Our network approach could mimic a functional age dependence, too, this would require that the age variable x is used as input to the network. The specific functional form will then be determined through (compositions of) the chosen activation functions.

4.4. Lee–Carter network mortality forecasting models

In Section 4.2, we have explicitly introduced one of the models that we are going to study numerically in the next section. We call this base model LCCONV. In the numerical part, we are going to consider a couple of variants of this base model LCCONV. We briefly introduce these variants in the present section.

We consider two variants of model LCCONV, which both replace the CNN layer on line 13 of Listing 1 by LSTM layers. The key difference between these variants relates to how the LSTM layer is applied, which is whether the LSTM produces a single output z_{t_1} representing the prediction at the last time step, t_1, or if an output is produced for each time step t where $t_0 \leq t \leq t_1$, as mentioned at the end of Section 3.3.

In more detail, in the first variant (called LCLSTM1) only the last activation z_{t_1} of dimension 128 is used as input into the next layer. This is probably the most common way of using recurrent neural networks and is described in Listing 2 in the appendix. The second variant (called LCLSTM2) processes the mortality rates using a LSTM layer with fewer units (32) but it produces an output for all the intermediate time steps. To achieve this, we use the built-in functionality within the LSTM command in Keras (see Listing 3 in the appendix). Similar to the LCCONV model, the output of this layer $z = (z_{t_0}, \ldots, z_{t_1})' \in \mathbb{R}^{10 \times 32}$ is a matrix and a 1D max-pooling layer can be employed to reduce the size of the temporal dimension of this matrix. As suggested in Goodfellow et al. (2016), the size reduction induced by pooling layers can also result in improved statistical efficiency and a reduced number of parameters, especially when a fully connected layer follows a convolutional layer. This is exactly the case of the LCCONV and LCLSTM2 models, since the next layer in the model is a fully connected layer, thus, the max-pooling layer saves 16,000 ($32 \times 5 \times 100$) parameters. Since the output of the LCLSTM1 model is a vector $z_{t_1} \in \mathbb{R}^{1 \times 128}$, a max-pooling layer, which would operate on the first dimension of the vector, cannot be applied. Finally, in both variants of the LSTM models, both batch normalization and dropout are used.

Further variants that we are considering are *tanh* and ReLU activations in the CNN and LSTM layers, respectively, instead of the linear function. The models considered are summarized in Table 1.

5. Numerical analysis

This section presents our results of applying the models of Table 1 to data from the Human Mortality Database (HMD) and the United States Mortality Database (USMD).

Listing 2 Deep neural network model LCLSTM1 with a LSTM layer.

```
1   rates <- layer_input(shape = c(10,100), dtype = 'float32', name = 'rates')
2   #
3   Country <- layer_input(shape = c(1), dtype = 'int32', name = 'Country')
4   Country_embed=Country %>%
5       layer_embedding(input_dim = N, output_dim = 5) %>%
6       layer_flatten(name= 'Country_embed')
7   #
8   Gender <- layer_input(shape = c(1), dtype = 'int32', name = 'Gender')
9   Gender_embed = Gender %>%
10      layer_embedding(input_dim = 2, output_dim = 5) %>%
11      layer_flatten(name= 'Gender_embed')
12  #
13  LSTM1 = rates %>% layer_lstm(units = 128, activation = "linear",
14                  recurrent_activation = "tanh", return_sequences = F) %>%
15      layer_batch_normalization() %>%
16      layer_dropout(rate = 0.35)
17  #
18  decoded = LSTM1 %>% list(Country_embed, Gender_embed)%>%
19      layer_concatenate() %>%
20      layer_dropout(rate = 0.4) %>%
21      layer_dense(units = 100, activation = 'sigmoid') %>%
22      layer_reshape(c(1,100), name = 'forecast_rates')
23  #
24  model <- keras_model(inputs = list(rates, Country, Gender), outputs = c(decoded))
```

Listing 3 Deep neural network model LCLSTM2 with a LSTM layer.

```
1   rates <- layer_input(shape = c(10,100), dtype = 'float32', name = 'rates')
2   #
3   Country <- layer_input(shape = c(1), dtype = 'int32', name = 'Country')
4   Country_embed=Country %>%
5       layer_embedding(input_dim = N, output_dim = 5) %>%
6       layer_flatten(name= 'Country_embed')
7   #
8   Gender <- layer_input(shape = c(1), dtype = 'int32', name = 'Gender')
9   Gender_embed = Gender %>%
10      layer_embedding(input_dim = 2, output_dim = 5) %>%
11      layer_flatten(name= 'Gender_embed')
12  #
13  LSTM2 = rates %>% layer_lstm(units = 32, activation = "linear",
14                      recurrent_activation = "tanh", return_sequences = T) %>%
15      layer_max_pooling_1d(pool_size  =2) %>%
16      layer_batch_normalization() %>%
17      layer_dropout(rate  =0.35) %>%
18      layer_flatten()
19  #
20  decoded = LSTM2 %>% list(Country_embed, Gender_embed)%>%
21      layer_concatenate() %>%
22      layer_dropout(rate = 0.4) %>%
23      layer_dense(units = 100, activation = 'sigmoid') %>%
24      layer_reshape(c(1,100), name = 'forecast_rates')
25  #
26  model <- keras_model(inputs = list(rates, Country, Gender), outputs = c(decoded))
```

5.1. Human Mortality Database

Similarly to Richman & Wüthrich (2020), the first part of the data analysis was carried out using data of all countries of the Human Mortality Database from 1950 onward.[1] Let $\mathcal{T} = \{t \in \mathbb{N} : 1950 \leq t \leq 2016\}$ denote the calendar years of available data. We set 1999 to be the final observation year. Our aim is to find the model that best forecasts, out of sample, the mortality rates from 2000 onward. In this setting, data were partitioned in a training data set containing the mortality rates of years in $\mathcal{T}_{\text{train}} = \{t \in \mathcal{T} : t \leq 1999\}$, and a testing data set $\mathcal{T}_{\text{test}} = \{t \in \mathcal{T} : t > 1999\}$ for out of sample model performance analysis.

All data entering the networks was pre-processed as follows: mortality rates recorded as zero or missing were imputed using the average rate at that age across all countries for that gender in that year, and ages $x > 99$ were omitted due to the excess variability of the mortality rates of the oldest ages in the HMD. Only male and female populations of countries for which more than 10 years of data before 2000 are available were considered in the model, producing a set $\mathcal{I}$ of populations by gender and region of size $|\mathcal{I}| = 76$.

While we have specified the models in the previous section, various parts of these models might be tweaked to achieve optimal performance, for example, the choice of activation functions, swapping CNN for LSTM layers, number of hidden units, dropout rates, etc. Since we frame the problem that 1999 is the final year in which mortality is observed year, the following years (from 2000 onward) are not available for tuning the model architecture and parameters. Accordingly, the search for the optimal model should be carried out using only observations for years in 1950–1999, i.e. in $\mathcal{T}_{\text{train}}$. In order to find the architecture with the highest out of sample accuracy, we replicate the procedure in Richman & Wüthrich (2020).

In Round 1 of fitting the models, the set $\mathcal{T}_{\text{train}}$ was further partitioned in $\mathcal{T}_{\text{train}}^{(1)} = \{t \in \mathcal{T}_{\text{train}} : 1950 \leq t < 1991\}$ and $\mathcal{T}_{\text{test}}^{(1)} = \{t \in \mathcal{T}_{\text{train}} : 1991 \leq t \leq 1999\}$: all architectures are trained on years

---

[1] A complete list of the countries can be found in the Appendix of Richman & Wüthrich (2020).

Table 2. Round 1 of fitting the deep networks on the HMD data (1950–1990): validation and test set MSEs of all 9 deep neural architectures described in Table 1 averaged over ten training runs; MSE values are in 10^{-4}.

Model	Val_loss	Test_loss
LCCONV	2.15	4.19
LCCONV_tanh	2.12	4.21
LCCONV_relu	2.86	4.70
LCLSTM1	2.31	5.31
LCLSTM1_tanh	2.44	5.80
LCLSTM1_relu	3.25	4.73
LCLSTM2	2.13	4.20
LCLSTM2_tanh	2.13	4.32
LCLSTM2_relu	4.15	5.36

in $\mathcal{T}_{\text{train}}^{(1)}$ and performance was assessed on $\mathcal{T}_{\text{test}}^{(1)}$. The error on $\mathcal{T}_{\text{test}}^{(1)}$ represents a measure of the out of sample or out-time accuracy and can be used as a criterion for the best network selection. In Round 2, the best network was re-trained on $\mathcal{T}_{\text{train}}$ and future mortality rates for years in $\mathcal{T}_{\text{test}}$ were predicted. Throughout, we also rely on measuring the out of sample error on a validation set, taken as 5% of each training set. The model producing the lowest error on each training set was taken as the candidate for measuring the error on the test set $\mathcal{T}_{\text{test}}$.

All 9 networks of Table 1 are trained to predict mortality rates of the next year using data of $T + 1 = 10$ previous years. In this setting, predictions can be done only for years in $\{1960, \ldots, 1999\}$ since there is not enough input data for the first 10 years of the training set (which starts in 1950). In addition, knowing that the length of the mortality rate history recorded in the HMD varies from country to country, the data were further filtered selecting for each forecasting year in $\mathcal{T}_{\text{train}}$ only populations for which the previous 10 years of data are available. Table A1 in the appendix lists the number of populations considered for each forecasting year $t \in \mathcal{T}_{\text{train}}$.

In Round 1, all networks were trained on years in $\mathcal{T}_{\text{train}}^{(1)}$ and the performance was assessed on years in $\mathcal{T}_{\text{test}}^{(1)}$. In this stage, the training set includes 1986 sequences: 1886 (around 95%) were used as training set and 100 examples (the remaining 5%) were used as validation set. Fitting was carried out using the Adam optimizer, see Kingma & Ba (2014), with the parameter values taken at the defaults for 2000 epochs. The weight configuration with the lowest validation loss was selected and used to make out of sample predictions. Since network results can vary among training attempts due to the randomness of some elements of the training process (i.e. the random selection of batches of training data, dropout, the initial value of optimization algorithm and others), 10 different runs were carried out for each network and the results obtained in Round 1 were summarized in Table 2 and Figure 1. Table 2 lists the average validation and testing loss for all networks on ten different runs and Figure 1 shows boxplots of the results. Note that we use mean square error (MSE) as objective function, and that these results are in 10^{-4}.

Several observations can be made. The best test set results are produced by LCCONV. Adding nonlinear activations does not improve test set performance. Whereas the CNN layer with *tanh* activations (LCCONV_tanh) is only marginally worse than the linear version (LCCONV), adding ReLU activations produces much worse performance. This is also the case for the LCLSTM2 models, whereas the LCLSTM1 models benefit from ReLU nonlinearities. Furthermore, Figure 1 shows that the results of the ReLU activations are highly variable. Compared to the CNN layers, the LSTM layers are much more variable, although the linear LCLSTM2 model performs in average about as well as the LCCONV model. LCLSTM1 model and the, respectively, nonlinear variants seem to be less accurate than LCCONV and LCLSTM2 models. We conclude from this that the LCCONV results seem to be the most optimal with the lowest average out of sample error by a small margin and less variable results than the others.
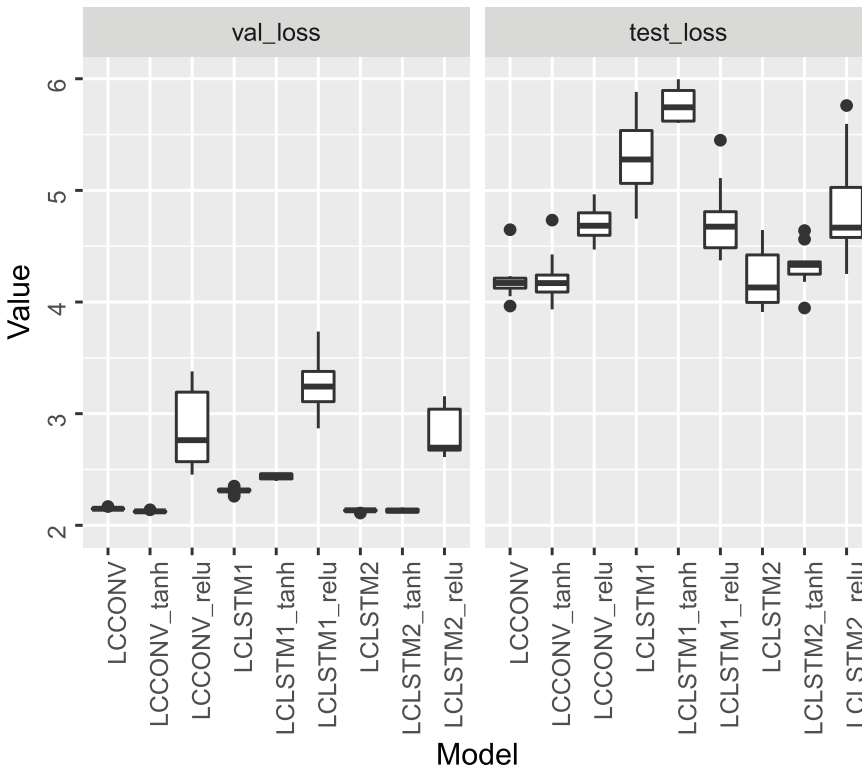
Figure 1. Boxplot of the results of Round 1 of the networks described in Table 1 for 10 runs; left shows validation losses on data from 1950 to 1990, right shows losses on test data from 1991 to 2000.

To explore the role of the pooling layer in the performance of the network, we also test in Round 1 two variants of the LCCONV and LCLSTM2 models, that exclude the max-pooling layer. We observed slightly worse performance for both models as measured by the MSE, which was $4.28 \cdot 10^{-4}$ for the LCCONV model and $4.24 \cdot 10^{-4}$ for the LCLSTM2 model, however, the number of parameters increases significantly by approximately 16,000, which is a significant proportion of the total model parameters, as shown in Table 5. Since the max pooling layer has a beneficial effect on model performance and size, these two variants of the models will not be considered further in the next sections.

Following the procedure for selecting the networks specified above, in Round 2, theoretically only model LCCONV should be considered. However, we wish to confirm in this work that the proposed model selection strategy is valid, at least for mortality forecasting. Therefore, all 9 networks are fit on the full training $\mathcal{T}_{train}$ and their out of sample accuracy was assessed on $\mathcal{T}_{test}$. In this stage, the training set includes 2662 sequences of which 134 (5%) were employed to validate the models.

Results are shown in Table 3 that reports, for each network, the average test set MSE on 10 different runs, the MSE of the ensemble predictions obtained by averaging the 10 predictions of each model fit and, finally, the number of populations in which the ensemble MSE is lower than the LC prediction. The results of the deep fully connected ReLU network proposed in Richman & Wüthrich (2020), named DEEP5 and denoted by DEEP in this work, is also considered, since it represents a comparably strong benchmark.

Some interesting observations can be made. First, there is not a perfect match between the ranking of model performance resulting from the test loss of Table 2 and the test loss of Table 3. Nonetheless, three of the four models with good results on the period 91–99, namely LCCONV, LCLSTM2_tanh and LCLSTM2, seem to perform quite well on the data from the years 2000–2016. Thus, it can be

Table 3. Round 2 of fitting the deep networks on the HMD data (1950–1999): results of all 9 deep network architectures of Table 1, average test loss over ten training runs, ensemble MSE and number of populations in which each network beats the LC model; MSE values are in 10^{-4}.

Model	Test_loss	Ensemble MSE	Number populations
LCCONV	2.27	2.24	75/76
LCCONV_tanh	2.62	2.58	61/76
LCCONV_relu	3.26	3.10	57/76
LCLSTM1	2.86	2.54	69/76
LCLSTM1_tanh	3.32	3.03	58/76
LCLSTM1_relu	3.33	3.25	52/76
LCLSTM2	2.43	2.32	74/76
LCLSTM2_tanh	2.36	2.27	75/76
LCLSTM2_relu	3.44	3.11	56/76
DEEP	2.83	2.53	67/76
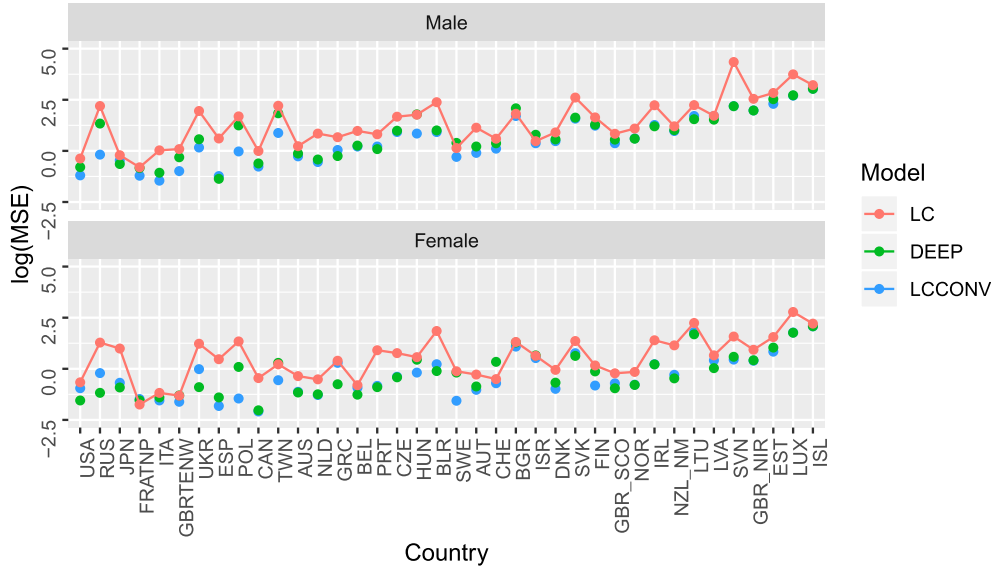
Figure 2. Test set performance of LCCONV, DEEP and the LC models for years in 2000–2016 in different populations ranked by population size; MSE values are in 10^{-4} and on log-scale.

confirmed that the two-round approach of fitting networks produced results which indicate the most optimal model for the forecasting of unseen data. Second, the ensemble MSE on 10 runs is systematically lower than the average MSE which suggests, in line with results in other applications of deep learning, that a further improvement can be achieved by averaging the results of different runs.

Finally, LCCONV produces the lowest ensemble MSE closely followed by the LCLSTM2_tanh model. LCCONV and LCLSTM2_tanh ensemble beat the LC prediction in almost all of the populations considered (75/76). In addition, these models beat the DEEP model of Richman & Wüthrich (2020) by a significant margin, meaning that the motivating premise of the paper is correct: neural network models designed for directly processing sequential data can significantly outperform more general network architectures applied to sequential data.

For the rest of this section, we focus on the LCCONV results, given its strong performance. Figure 2 compares the performance of LCCONV and DEEP against the LC model in all countries under investigation distinguishing by gender.

Figure 3. Comparison errors among LCCONV, DEEP and the LC model on HMD data for years in 2000–2016 for different ages and genders; MSE values are in 10^{-4} and on log-scale.

Countries are sorted by population size in 2008, the median year of $\mathcal{T}_{\text{test}}$, and the logarithmic scale was employed to improve the readability of this plot. We note that the error produced by the LC model seems to increase when population size decreases: small MSE values can be observed in countries with large populations such as USA and Russia whereas large MSEs occur in less populated countries such as Luxembourg and Iceland. We believe that this is related to the law of large numbers, which makes volatility in morality rates smaller for bigger countries, which acts positively in our model calibrations. Both network models show the same pattern, however, with lower MSE values than for the LC model. The only population in which the LC model produces the most accurate prediction is for the French female population. A comparison between the LCCONV and DEEP models shows that the LCCONV model produces a lower MSE on male populations of large size while the performances are quite similar in other countries. On the other hand, there is no clear ranking among these two models on female populations.

Figure 3 compares the same models, however, it analyses the errors by age for both genders using the logarithmic scale. All three models produce error shapes that look like the shape of a mortality table; this can be understood by considering that the variability of mortality rates will increase as mortality rates increase or due to increasing estimation error as the population at each age decreases. For both genders, the LCCONV model produces the lowest error curve, the DEEP model produces a slightly higher one and the LC model has the worst performance. The error curves for males are more distant from each other than those of females, indicating that more significant improvements are obtained for males. For males, the benchmark DEEP model beats the LC model with lower MSEs in all ages but LCCONV produces even better performance, beating the LC model in 100/100 ages and the DEEP model in 97/100 ages. Specifically, the improvement produced by LCCONV appears to be significant for most ages (15–90). For females, we observe that LCCONV beats the LC model on 95/100 ages and the DEEP model on 79/100 ages. In this case, the LCCONV model performs better than both models at the middle ages, whereas the DEEP and LCCONV models perform similarly for childhood and old ages.

Figure 4 plots the average residuals of the LC, DEEP and LCCONV models for all ages and (test set) years, distinguishing by gender. The residuals are calculated as the observed values minus predictions, scaled by the estimated standard deviation of the actual values calculated at each age and for each gender separately. The red areas in Figure 4 indicate overestimation of mortality rates, while blue areas indicate an underestimation. Again, we observe that the improvement produced by LCCONV for males is quite well distributed among ages while it especially concerns females in the middle ages.

Overall, DEEP produces smaller residuals than the LC model. As highlighted in Richman & Wüthrich (2020), the DEEP model seems to learn interactions between age and year, thus, reducing
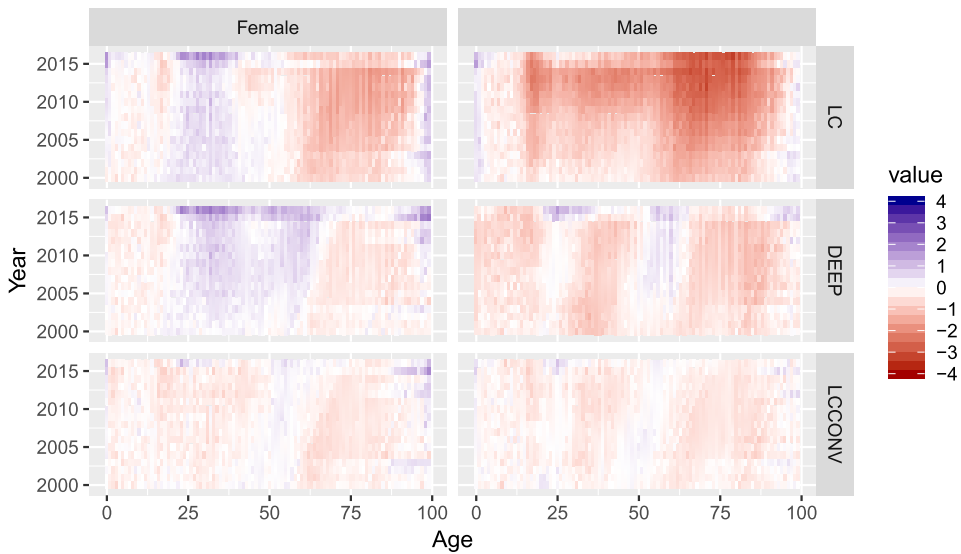
Figure 4. Residuals produced by each of the models, for each gender, year and age separately, averaged over the countries in the HMD.

the cohort effect in the residuals. The LCCONV model further improves the forecasting, exhibiting even smaller residuals, in particular, for males at middle ages in the whole period 2000–2016, and it also appears to reduce the cohort effect seen in the LC residuals. The cohort effects appearing in the LC residuals have been well studied and extensions of the LC model to incorporate a cohort effect have been presented, for example, in the Renshaw and Haberman (RH) model (Renshaw & Haberman 2006). Although models incorporating a cohort effect may fit the data of some countries more closely than the LC model, nonetheless, there are well-known difficulties in fitting the RH model (Cairns et al. 2009, Villegas et al. 2018), thus making the model less suitable for large-scale forecasting.[2] Also, since the LCCONV model we propose does not explicitly incorporate a cohort effect, limiting the comparison to models that present this same characteristic appears reasonable. We discuss the possibility of explicitly including a cohort effect in the LCCONV model in the conclusion to the paper.

In order to check the robustness of the argued accuracy, we further compare LC, DEEP and LCCONV considering different error measures and age ranges. Table 4 lists the performances in terms of MSE, MAE (Mean Absolute Error) and MAPE (Mean Absolute Percentage Error) of these 3 models in the age ranges 0–99 and 55–90 for both genders. The LCCONV model produces the best performance on males and females for all considered error measures in the full age range (0–99) as well as in the shorter age range (55–90). Interestingly, the relative gain produced by the LCCONV on the LC is larger when considering the 55–90 range for both genders (for example, the relative gain on males, in terms of MAE, is 46.82% in the full range 0-99 and becomes 64.13% considering the shortened range 55–90).

Finally, it is useful to investigate the number of parameters that must be learned in all the networks under investigation. A lower number of weights simplifies the model and speeds up the training process. Table 5 lists the number of weights for all 9 networks. It must be noted that LCCONV, LCCONV_tanh and LCCONV_relu have different activation functions but they share the same number of weights. Since this is true for the other configurations as well, Table 5 provides a single item for each group of networks. Of all the models, LCLSTM1 involves the largest number of parameters

---

[2] In some initial experiments with the RH model on the HMD data, we found that this model often does not converge, and when it does, the forecasting performance is only better than the LC model for a limited number of countries.

Table 4. MSE, MAE and MAPE of LC, DEEP and LCCONV in the age ranges 0–99 and 55–90 for each gender; MSE and MAE values are in 10^{-4}.

Range	Variable	MSE		MAE		MAPE	
		Female	Male	Female	Male	Female	Male
0–99	LC	2.7512	8.1808	53.6300	89.5677	23.7294%	31.5151%
	DEEP	1.3183	3.7506	32.3518	58.9379	18.2453%	21.2134%
	LCCONV	1.2625	3.2239	30.1143	47.6310	17.6264%	15.4263%
55–90	LC	1.0326	2.7575	58.4269	110.9377	16.3799%	23.0486%
	DEEP	0.2739	1.0010	28.4942	58.9209	8.3352%	10.1734%
	LCCONV	0.2156	0.5289	25.1887	39.7923	7.5622%	6.9878%

Table 5. Number of parameters of all 9 networks under investigation.

Model	Number of parameters
LCCONV	27,120
LCLSTM1	131,920
LCLSTM2	35,512
DEEP	73,676

(131,920) while DEEP is second (73,676). The LCLSTM2 model (35,512) and the LCCONV (27,120) model further reduce the number of network weights. It is also important to note that, especially on GPUs, the LCCONV model is significantly faster to train than the LCLSTM models.

5.2. United States Mortality Database

In this subsection, we discuss fitting the models discussed above to the data in the United States Mortality Database (USMD) to validate how well our approach generalizes across different data sets. One goal is to ensure that the models presented above have not overfit to the HMD data (although this is somewhat unlikely given the relative simplicity of the model). We also seek to understand if the insights into activation functions and variability of results observed above persist across data sets.

The USMD provides data at various levels of aggregation; here we use the mortality data at a state level, thus, the USMD provides mortality rates over the period $\mathcal{T}^{US} = \{t \in \mathcal{T} : 1959 \leq t \leq 2017\}$ for 51 regions (namely, the 50 states and the District of Columbia) distinguished by gender, leading to a total of 102 populations. Similarly to the previous subsection, 1999 was taken as the final observation year and data were partitioned in training $\mathcal{T}^{US}_{train} = \{t \in \mathcal{T} : t \leq 1999\}$, and testing sets $\mathcal{T}^{US}_{test} = \{t \in \mathcal{T} : t > 1999\}$.

All 9 networks discussed above were directly trained on mortality data of years in $\mathcal{T}^{US}_{train}$ and the performance was assessed in $\mathcal{T}^{US}_{test}$. In this case, the training set includes 3162 sequences and 159 (5%) examples of these were used as validation set. The same optimization hyper-parameters were employed and ten runs for each model were carried out. Table 6 summarizes the results obtained.

Most importantly, it is observed that the LCCONV model performs very well on the USMD data without any changes to architecture or hyper-parameters, thus, it appears that this model does not overfit to the HMD data. Furthermore, some of the other results obtained on the HMD data hold also on the USMD data. The LCCONV, LCCONV_tanh, LCLSTM2 and LCLSTM2_tanh work well on the US data: the average MSE over ten runs of each of these models is lower than the MSE produced by the LC model (which is 1.12×10^{-4}) and the performance further improves considering the MSE of ensemble predictions. In this case, the LCLSTM2 model produces the lowest ensemble MSE on US mortality data by a small margin over the LCCONV model. Similar to the conclusions above, the ReLU networks do not perform well.

Figure 5 compares logged MSEs obtained by LCCONV and LC distinguishing by population. In this case, countries are ordered by the LC performance and the only one in which LCCONV has a

Table 6. Fitting the deep networks on USMD data (1950–1999): results of all 9 network architectures investigated, averaged test loss over ten training runs, ensemble MSE and number of populations in which each network beats LC model; MSE values are in 10^{-4}.

| Model	Test loss	Ensemble MSE	Number of populations
LCCONV	0.50	0.49	101/102
LCCONV_tanh	0.55	0.54	100/102
LCCONV_relu	0.74	0.73	86/102
LCLSTM1	0.63	0.58	98/102
LCLSTM1_tanh	0.66	0.53	99/102
LCLSTM1_relu	1.20	1.10	52/102
LCLSTM2	0.50	0.46	102/102
LCLSTM2_tanh	0.53	0.50	101/102
LCLSTM2_relu	1.09	0.96	62/102
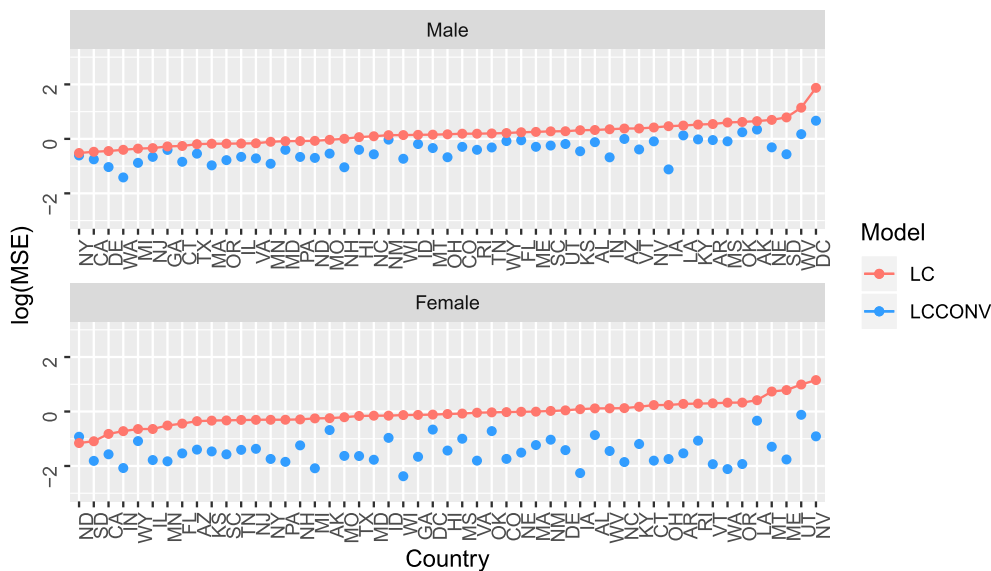
Figure 5. Comparison between the LCCONV and the LC model on USMD data for years in 2000–2016 for different populations; MSE values are in 10^{-4} and on the log-scale.

larger MSE is the North Dakota female population. The figure also shows that the LCCONV model improves predictions more for females than for males, which is the opposite of the finding discussed above for the HMD data. A clear explanation for this was not found, and we leave the matter for future exploration and discussion.

Figure 6 compares performance of LCCONV and LC distinguishing by age. The logged MSEs produced by the LC model is generally higher than the one of the LCCONV model except for ages in 22–26 where LC seems to work particularly well.

We conclude this section by discussing two interpretations of the LCCONV network. As mentioned in Section 4, the parameters of the last layer of the LCCONV model can be interpreted as a generalized LC model, see (21). In particular, the bias terms of the output layer $w_0 = (w_{0,j})_{1 \leq j \leq d} \in \mathbb{R}^d$ can be seen as playing the role of a population independent estimate of mortality parameter a_x similar to the LC model. Figure 7 plots these terms, scaled to lie in range [0, 1]. As would be expected, the bias terms have the familiar shape of a life table, and are smooth, reflecting the large amount of data that the network has been trained on, thus, we have the classical age one-dimensional embedding that is usually explored under LC.
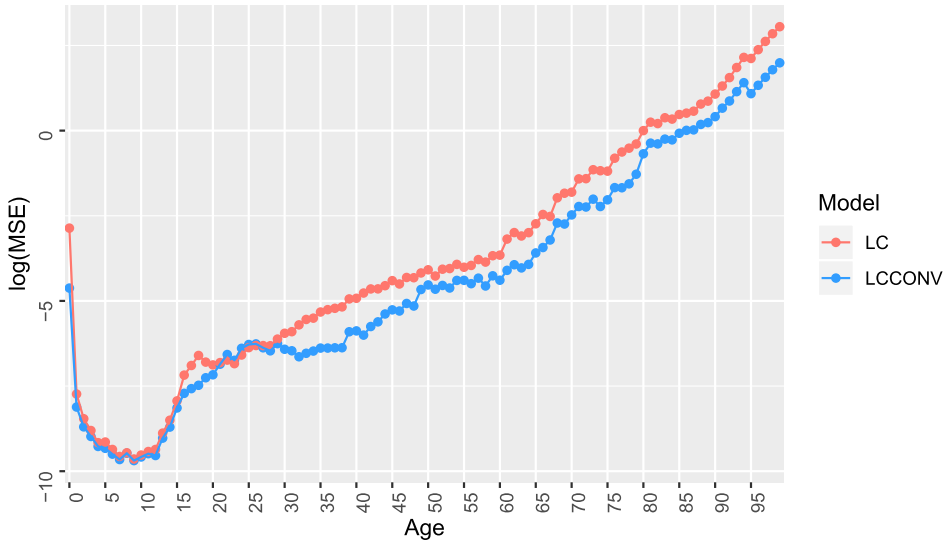
Figure 6. Comparison between the LCCONV and the LC model on USMD data for years in 2000–2016 for different ages; MSE values are in 10^{-4} and on the log-scale.
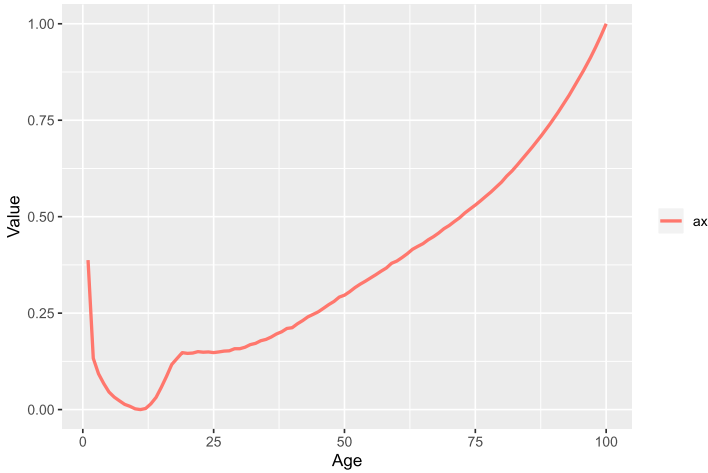
Figure 7. Bias terms $w_0 = (w_{0,j})_{1 \le j \le d}$ of the first of the ten LCCONV models fit to the data in the USMD database scaled to range $[0, 1]$.

We analyze the parameters of the embedding layer related to the regional effect $z_{\mathcal{R}}(r)$ in the LCCONV model applied to the USMD data. Figure 8 plots the state level parameters after reducing the dimension from 5 dimensions to 2 dimensions using principal component analysis. Each point stands for one of the 51 regions represented by their first two principal components.

A K-means cluster analysis was performed on the new features for $K = 4$, and interesting groups can be observed. The first cluster includes countries located in the south of US (Arizona, Florida, New Mexico, Texas) and US countries geographically distant from the US zone such as Alaska and Hawaii. The second cluster groups countries in the southeast zone (Louisiana, Georgia, South Carolina, Mississippi and Alabama) while the third one includes the central (Colorado, Kansas, Utah and Missouri) and the northwest (Washington, Oregon, Montana and North Dakota) regions. The last cluster groups countries in the northeast of US (Pennsylvania, Connecticut, Massachusetts, Virginia
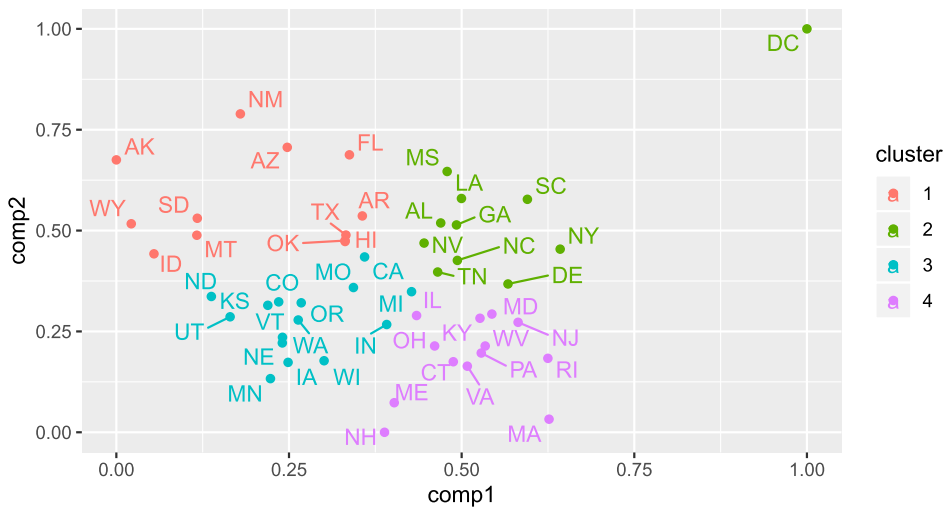
Figure 8. Region embedding vector $z_{\mathcal{R}}(r)$ from one of the LCCONV models, visualization considers the first two principal components, and clustered by ($K = 4$)-means clustering.

and West Virginia). This analysis highlights that the network has learned similarities and dissimilarities among mortality structures of different US countries and it seems reasonable to suggest that neighboring states present similar evolutions of mortality rates.

6. Discussion and conclusion

We have motivated why a neural network model designed to process sequential data might provide better forecasting performance than a more general model, and shown that, indeed, processing raw mortality data using a simple convolutional network model combined with embedding layers produces excellent performance on large-scale mortality forecasting problems. We have shown that the model concept generalizes across two large mortality databases, thus, it is reasonable to expect that this type of model would also produce a good performance on other mortality data sets. An interesting avenue for future research is whether the model presented in this paper would perform well on smaller populations than those investigated in this paper, for example, for forecasting mortality rates for portfolios of insurance products. The model that has been proposed has a useful interpretation within the general structure of the well-known LC model. Although the focus has been on this convolutional model, we have also shown how LSTM networks might be adapted to process mortality data successfully. An interesting question is to consider what other applications this network structure might be suitable for, and how well the structure will generalize to other forecasting areas. Since mortality varies relatively smoothly over time, it may be the case that this model could be used to forecast other demographic variables that behave in a similar fashion, for example, fertility rates. Returning to the analysis of mortality, an interesting extension of the model presented in this paper would be to attempt to include cohort effects similar to those used in the model of Renshaw & Haberman (2006), which could feasibly be incorporated into the model by adding a cohort term to Equation (21). Another path might be to consider mortality modeling paradigms other than the LC model, for example, how might mortality laws such as the Heligman–Pollard model be applied within a neural network framework. Whereas we have focused on forecast accuracy in this work, coherent forecasts of the mortality of related populations are also of interest, see, for example, Li & Lee (2005) and Hyndman et al. (2013). An interesting issue to understand in future research is how coherence can be enforced within the time series models presented in this paper, which produces

forecasts using time series that may themselves not be coherent. Finally, the issue of forecast uncertainty should be addressed in future research, since applying the techniques usually used in statistical literature is not straightforward. For example, applying the bootstrap is complicated by needing to average over the results of several neural networks to achieve stability and some theoretical developments on the appropriate manner of doing this in each bootstrap simulation are needed before the technique can be applied. Moreover, since fitting the neural networks used in this study is computationally demanding, the many simulations required by the bootstrap appear unfeasible to derive forecast uncertainty, thus, the bootstrap technique would need to be modified if it was to be applied in this scenario. Other approaches to deriving forecast uncertainty might involve the investigation of techniques popular in the wider machine learning literature, such as the pinball loss of Takeuchi et al. (2006).

Acknowledgments

Disclosure statement

No potential conflict of interest was reported by the author(s).

ORCID

Ronald Richman http://orcid.org/0000-0002-0441-8354

References

Bengio Y., Courville A. & Vincent P. (2013). Representation learning: a review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **35**(8), 798–1828.

Bengio Y., Ducharme R., Vincent P. & Jauvin C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research* **3**(Feb), 1137–1155.

Borovykh A., Bohte S. & Oosterlee C. (2017). Conditional time series forecasting with convolutional neural networks. arXiv:1703.04691.

Brouhns N., Denuit M. & Vermunt J. K. (2002). A Poisson log-bilinear regression approach to the construction of projected lifetables. *Insurance: Mathematics and Economics* **31**, 373–393.

Cairns A. J., Blake D. & Dowd K. (2006). A two-factor model for stochastic mortality with parameter uncertainty: theory and calibration. *Journal of Risk & Insurance* **73**(4), 687–718.

Cairns A. J., Blake D., Dowd K., Coughlan G. D., Epstein D., Ong A. & Balevich I. (2009). A quantitative comparison of stochastic mortality models using data from England and Wales and the United States. *North American Actuarial Journal* **13**(1), 1–53.

Cho K., Van Merrienboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H. & Bengio Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv:1406.1078.

Chollet F. (2018). Keras: the Python deep learning library. *Astrophysics Source Code Library*.

Currie I. D. (2017). On fitting generalized linear and non-linear models of mortality. *Scandinavian Actuarial Journal* **4**, 356–383.

Elman J. L. (1990). Finding structure in time. *Cognitive Science* **14**(2), 179–211.

Gers F. A., Schmidhuber J. & Cummins F. (2000). Learning to forget: continual prediction with LSTM. *Neural Computation* **12**(10), 2451–2471.

Goodfellow I., Bengio Y. & Courville A. (2016). *Deep learning*. Cambridge, MA: MIT Press.

Graves A. & Schmidhuber J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* **18**(5–6), 602-–610.

Graves A. & Schmidhuber J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. *Advances in Neural Information Processing Systems* **21**(NIPS 2008), 545–552.

Greff K., Srivastava R. K., Koutník J., Steunebrink B. R. & Schmidhuber J. (2016). LSTM: a search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems* **28**(10), 2222–2232.

Guo C. & Berkhahn F. (2016). Entity embeddings of categorical variables. arXiv:1604.06737.

Hainaut D. & Denuit M. (2020). Wavelet-based feature extraction for mortality projection. *ASTIN Bulletin* **50**(3), 675–707.

Hastie T., Tibshirani R., Friedman J. & Franklin J. (2005). The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer* **27**(2), 83–85.

Hochreiter S. & Schmidhuber J. (1997). Long short-term memory. *Neural Computation* **9**(8), 1735–1780.

Human Mortality Database. (2020). University of California, Berkeley (USA), and Max Planck Institute for Demographic Research (Germany). Available at https://www.mortality.org/Public/CitationGuidelines.php.

Hyndman R. J., Booth H. & Yasmeen F. (2013). Coherent mortality forecasting: the product-ratio method with functional time series models. *Demography* **50**, 261–283.

Hyndman R. J. & Ullah M. S. (2007). Robust forecasting of mortality and fertility rates: a functional data approach. *Computational Statistics & Data Analysis* **51**(10), 4942–4956.

Ioffe S. & Szegedy C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. CoRR, abs/1502.03167.

Kingma D. P. & Ba J. (2014). Adam: a method for stochastic optimization. arXiv:1412.6980.

Kleinow T. (2015). A common age effect model for the mortality of multiple populations. *Insurance: Mathematics and Economics* **63**, 147–152.

LeCun Y., Boser B. E., Denker J. S., Henderson D., Howard R. E., Hubbard W. E. & Jackel L. D. (1990). Handwritten digit recognition with a back-propagation network. *Neural Information Processing Systems* **2**(NIPS 1989), 396–404.

Lee R. D. & Carter L. R. (1992). Modeling and forecasting us mortality. *Journal of the American Statistical Association* **87**(419), 659–671.

Li N. & Lee R. (2005). Coherent mortality forecasts for a group of populations: an extension of the Lee–Carter method. *Demography* **42**(3), 575–594.

Nigri A., Levantesi S., Marino M., Scognamiglio S. & Perla F. (2019). A deep learning integrated Lee–Carter model. *Risks* **7**(1), 33.

Renshaw A. E. & Haberman S. (2003). Lee-Carter mortality forecasting with age-specific enhancement. *Insurance: Mathematics and Economics* **33**(2), 255–272.

Renshaw A. E. & Haberman S. (2006). A cohort-based extension to the Lee–Carter model for mortality reduction factors. *Insurance: Mathematics and Economics* **38**(3), 556–570.

Richman R. (2018). AI in actuarial science. *SSRN Manuscript* ID 3218082.

Richman R. & Wüthrich M. V. (2019). Lee and Carter go machine learning. *SSRN Manuscript* ID 3441030.

Richman R. & Wüthrich M. V. (2020). A neural network extension of the Lee–Carter model to multiple populations. *Annals of Actuarial Science*, 1–21, to appear.

Shang H. L. (2019). Dynamic principal component regression: application to age-specific mortality forecasting. *ASTIN Bulletin* **49**(3), 619–645.

Shang H. L. & Haberman S. (2020). Forecasting multiple functional time series in a group structure: an application to mortality. *ASTIN Bulletin* **50**(2), 357–379.

Srivastava N., Hinton G., Krizhevsky A., Sutskever I. & Salakhutdinov R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* **15**(1), 1929–1958.

Takeuchi I., Le Q. V., Sears T. & Smola A. (2006). Nonparametric quantile estimation. *Journal of Machine Learning Research* **67**, 1231–1264.

University of California, Berkeley (USA) (2020). United States Mortality DataBase. usa.mortality.org. Accessed on 2020-03-30.

Villegas A. M., Millossovich P. & Kaishev V. K. (2018). StMoMo: stochastic mortality modeling in R. *Journal of Statistical Software* **84**, 1–32.

Wiatowski T. & Bölcskei H. (2018). A mathematical theory of deep convolutional neural networks for feature extraction. *IEEE Transactions on Information Theory* **64**(3), 1845–1866.

Yan H., Peters G. W. & Chan J. S. K. (2020). Multivariate long-memory cohort mortality models. *ASTIN Bulletin* **50**(1), 223–263.

Appendix

Table A1. Number of populations for each year.

Forecast year	Number of populations	Forecast year	Number of populations
1960	52	1980	70
1961	52	1981	70
1962	52	1982	70
1963	52	1983	70
1964	52	1984	70
1965	52	1985	70
1966	52	1986	70
1967	52	1987	70
1968	54	1988	70
1969	66	1989	70
1970	68	1990	70
1971	68	1991	72
1972	68	1992	72
1973	68	1993	76
1974	68	1994	76
1975	68	1995	76
1976	68	1996	76
1977	68	1997	76
1978	68	1998	76
1979	68	1999	76