British Actuarial Journal (2022), Vol. **27**, e21, pp. 1–57
doi:10.1017/S1357321722000162

SESSIONAL PAPER

Mind the gap – safely incorporating deep learning models into the actuarial toolkit

Ronald Richman

Old Mutual Insure, University of the Witwatersrand, Johannesburg, South Africa
E-mail: ronald.richman@ominsure.co.za, ronaldrichman@gmail.com

Abstract
Deep neural network models have substantial advantages over traditional and machine learning methods that make this class of models particularly promising for adoption by actuaries. Nonetheless, several important aspects of these models have not yet been studied in detail in the actuarial literature: the effect of hyperparameter choice on the accuracy and stability of network predictions, methods for producing uncertainty estimates and the design of deep learning models for explainability. To allow actuaries to incorporate deep learning safely into their toolkits, we review these areas in the context of a deep neural network for forecasting mortality rates.

Keywords: Deep Learning; Actuarial Science; Model Risk Management; Explainable Machine Learning; Quantile Regression; Mortality Modelling

1. Introduction

The maturing field of deep learning, which is the modern approach to designing and fitting neural networks, has achieved remarkable successes in domains such as computer vision, natural language processing and speech recognition (LeCun *et al.*, 2015) and is rapidly advancing the state of the art on so-called "artificial intelligence tasks" (Bengio, 2009), which are those tasks which humans find easy, but have turned out to be difficult for computer systems. These advances have been enabled by the modelling philosophy of representation learning (Bengio *et al.*, 2013; Goodfellow *et al.*, 2016) that is at the core of deep learning, which, in short, can be defined as allowing an algorithm to specify the optimal model structure for a particular task.

Recently, the actuarial community has paid increasing attention to deep learning, with investigation of topics across the spectrum of actuarial practice, such as pricing (Noll *et al.*, 2018; Schelldorfer & Wüthrich, 2019), reserving (Gabrielli & Wüthrich, 2018; Gabrielli *et al.*, 2019; Kuo, 2019; Delong *et al.*, 2020), mortality forecasting (Hainaut, 2018; Richman & Wüthrich, 2019; Nigri *et al.*, 2019; Perla *et al.*, 2020; Schnürch & Korn, 2022), life expectancy forecasting (Nigri *et al.*, 2021; Levantesi *et al.*, 2022) and analysis of telematics data (Gao & Wüthrich, 2018; Gao *et al.*, 2019; Gao *et al.*, 2020). Some of these earlier advances have been surveyed in Richman (2018), who provides an introduction to deep learning geared towards an actuarial audience.

Based on this emerging area of study within actuarial science, it appears that deep learning is an attractive technique for actuaries for (at least) the following reasons. First, the predictions of deep neural networks have been shown to outperform traditional actuarial techniques in pricing (Richman, 2018; Schelldorfer & Wüthrich, 2019), reserving (Gabrielli *et al.*, 2019) and mortality forecasting (Richman & Wüthrich, 2019), when considering out of sample performance metrics. Second, these techniques enable actuaries to analyse new forms of data (Gao *et al.*, 2020), which are potentially beyond the reach of classical techniques, due to the high dimensionality of the data, the high frequency with which the data are received, or both. Third, actuaries are usually familiar with the application of generalised linear models (GLMs) and their understanding of neural networks may be enabled by connecting these techniques to GLMs, see, for example, Wüthrich (2019b), who discusses several manners in which neural networks can be seen as generalising GLMs. Importantly, the manner in which predictions are made by neural networks is much closer to traditional actuarial methods than those based on boosted decision trees, which are an alternative machine learning methodology often applied to tabular data.

While holding significant potential for the actuarial profession, deep learning suffers from a potential shortcoming, in that a quite widely held view is that these models are so-called "black boxes", meaning to say, that it is difficult to understand how these models arrive at a prediction. While this is not a problem in some industries utilising neural networks, for actuaries, who are trained to inspect model structure and fitted coefficients closely, this issue can be worrying. Furthermore, deep neural networks usually provide only a best-estimate output and do not incorporate measures of uncertainty, which are important within actuarial practice. Finally, the results of multiple deep neural networks calibrated on the same data vary due to the random processes used when training these models. Due to these issues, the application of neural network techniques in actuarial work may be considered somewhat risky, and, furthermore, in jurisdictions where regulators, and other stakeholders such as board members, risk managers and auditors, pay close attention to the types of models used by actuaries, the use of deep neural networks might be challenged based on some of these grounds mentioned above. We aim to tackle these issues in this paper, which has, as its goal, to provide actuaries with tools on how to benefit safely from deep learning.

In this paper, we aim to address some of the key modelling issues actuaries should consider when looking to incorporate deep neural networks safely into their work. These issues fall under three main categories:

- Quantitative validation – the impacts of hyperparameter choices on the stability of neural networks applied to typical actuarial problems is studied, and the trade-offs of model variability on the one hand, and improved prediction on the other, are considered.
- Uncertainty quantification – we investigate how deep learning models can be modified to produce explicit measures of uncertainty that may allow actuaries to highlight instances where the model predictions are potentially more unreliable.
- Design considerations – by using suitability designed neural networks that are more amenable to interpretation in traditional actuarial terms, actuaries can provide comfort to their stakeholders that their deep learning models are fit for purpose. Here we consider proposals to design neural networks for explainability.

Relatively few papers within the actuarial literature have focused directly on the topic of how deep learning can be used safely by actuaries. Richman *et al.* (2019) focus on changes to the risk management process necessitated by the use of machine and deep learning models by actuaries and provides guidance on how deep learning models might be challenged from a risk management perspective. Whereas that study focusses more on the risk management of deep neural networks without considering in detail the modifications that might be made to enhance their use for actuarial applications, in this study we consider what modifications to deep neural networks might be appropriate to make their use safer. Wüthrich and Merz (2019) describe a way of structuring neural networks to incorporate traditional actuarial models, thus leading to greater explainability of the model (see also Gabrielli *et al.* (2019) and Schelldorfer & Wüthrich (2019) for applications of the proposal in Wüthrich and Merz (2019)). Gabrielli *et al.* (2019) apply bootstrap techniques to generate Incurred But Not Reported (IBNR) reserve uncertainty measures. Compared to these studies, the contribution of this paper is both to review and add to these approaches to safely utilising deep learning within actuarial work.

The rest of this manuscript is organised as follows. In section 2, we introduce concepts and notation that will be used throughout. In section 3, we use the common actuarial task of reserving for incurred but not reported (IBNR) claims within a tutorial aiming to introduce deep learning in a familiar context. In section 4, we provide a main example of a deep neural network for modelling and forecasting the mortality of national and sub-national populations. In section 5, we study how the performance and variability of the results of calibrated neural networks vary depending on the design of the network. In section 6, we address uncertainty measures for neural networks, based on techniques from the wider machine learning literature. In section 7, we review how neural networks can be designed for greater explainability. Finally, we conclude the paper with avenues for further research into these topics. An appendix to the paper provides a table of acronyms and summary of notation. Throughout, excerpts of code to reproduce the examples are provided and full code is provided on the associated GitHub repository[1].

A reader only interested in an introduction to machine learning applied within actuarial work could read section 3 in isolation. If only a single topic among those discussed in the paper is of interest, the reader could focus on section 4 and then the relevant section from sections 5 and 6.

2. Concepts and Notation

In this paper, we focus on predictive models, which have as their objective to predict an unknown vector of quantities y on the basis of a matrix of other, known variables, X. This definition encompasses many of the models used in actuarial practice, for example, mortality estimation and forecasting, IBNR reserving and pricing models, but does not obviously include the cashflow projection models commonly used for the valuation of life insurance business. The rows of y and X are indexed by $i \in 1 \ldots N$, where N is the number of observations that the actuary has available for parametrising the predictive model. The columns of X are indexed by $j \in 1 \ldots P$, where P is the number of variables available to the actuary to make predictions. Usually, the relevant components of P are determined by the type of problem at hand, for example, in the case of mortality forecasting, the P variables are usually age, gender and year. To improve the quality of the predictions, X is often augmented by transforming the variables in X into new variables using several different techniques (some examples are discretisation of numerical values, the application of spline functions, the combination of categories, and the interaction of different variables) to a new matrix X' indexed by $j \in 1 \ldots P'$. The transformation of X to X' is usually referred to as feature engineering in the machine learning literature. Then, the aim of a predictive model is to produce an accurate estimate $\hat{y}_i$ of the unseen variables y_i on the basis of variables X'_i.

The production of an estimate $\hat{y}_i$ from X_i may be accomplished with many different modelling tools. Traditionally, actuaries have favoured GLMs, which require the explicit specification of the mapping $X' \mapsto \hat{y}$, meaning to say, that the variables from X' that enter the model, and the form in which they do so, are chosen manually by the actuary. As an alternative to GLMs, actuaries have started to consider applying machine learning techniques, which we give a working definition of as those models that automatically produce the mapping $X' \mapsto \hat{y}$. Some popular examples of these techniques are ensembles of decision trees, examples of which are random forests (Breiman, 2001) and gradient boosted trees (Tibshirani, 1996) and elastic-net (Zou & Hastie, 2005) (for a review of these models, see Friedman *et al.* (2009) and for some foundational studies of how machine learning might be applied to actuarial topics, see Wüthrich (2018) and Deprez *et al.* (2017)). Since machine learning models are highly flexible, it is often the case that these models may overfit the data used to parametrise the models, and produce relatively poor predictions on unseen data (indeed, this is also a concern with statistical models and tools such as the Aikaike Information Criterion (AIC) are used to penalise models that are too flexible relative to the gain in performance as measured by the likelihood). Thus, a fundamental procedure when applying machine learning models is to test the performance of these models on unseen data. Also, machine learning techniques often require the choice of hyperparameters, which are usually choices of

[1]https://github.com/RonRichman/Mind-the-Gap

model structure that cannot be calibrated directly from the data, for example, the depth of decision trees used within an ensemble or the extent of the regularisation used when calibrating model coefficients.

If N is relatively large then a popular was of accomplishing this is to partition the N observations into three (disjoint) sets: a training set of size N^{train}, a validation set of size $N^{validation}$ and a testing set of size N^{test}. The models are then parametrised (or trained) on the training set (i.e. using only the observations in the training set of size N^{train}) and the effect of hyperparameter choices on the accuracy of models is tested on the validation set. Finally, the accuracy of the model on unseen data is approximated by testing the model on the N^{test} observations of the test set. If, on the other hand, there are too few observations left to calibrate the models after splitting the observations into disjoint sets, then another more suitable procedure, such as cross-validation, will be used.

Often, the performance of machine learning models is determined by the extent and quality of the feature engineering performed to derive X'. Since feature engineering is a relatively manual and time-consuming process often requiring substantial domain knowledge (for a summary of the arguments against the feature engineering approach, see Richman *et al.* (2019)), a focus of part of the machine learning literature is on automating the production of X' using a technique called representation learning (Bengio *et al.*, 2013). Thus, instead of approximating only the mapping of $X' \mapsto \hat{y}$, algorithms that include a representation learning step first produce a mapping $X \mapsto X'$, where the transformed feature matrix X' is optimised for the task at hand, and then learn the mapping $X' \mapsto \hat{y}$.

A successful implementation of representation learning is deep learning (LeCun *et al.*, 2015; Goodfellow *et al.*, 2016), as measured by the predictive performance of deep learning across many machine learning tasks such as computer vision and natural language processing, as well as on the analysis of tabular data (Guo & Berkhahn, 2016) and time series (Makridakis *et al.*, 2018). Here, our definition of deep learning is that it is the modern approach to designing and fitting neural networks (see, for example, the usage of the phrase deep learning in Goodfellow *et al.* (2016)). In the deep learning approach, machine learning problems are solved by composing together many layers of non-linear functions, which allow complex relationships within the training data to be approximated. The specific form of these functions is usually determined by the application at hand. In the following section we introduce neural networks in the familiar context of IBNR reserving and in the subsequent section, we discuss the deep learning models applied in this paper.

3. Tutorial on Linear Regression, GLMs and Neural Networks

To work towards the definition of neural networks, we start with simple linear regression models and expand that definition to GLMs, after which we define neural networks and the structure of the networks that we focus on in this research. We provide a practical example of the application of these techniques in an actuarial context by fitting GLM and neural network models on the famous claims triangle of Taylor and Ashe (1983) to derive IBNR reserves. This example has been chosen due to its relative simplicity as well as widespread applicability for actuaries.

3.1. Linear Regression and GLMs

A linear regression model can be defined using matrix notation as

$$\hat{y} = b_0 + B_0^T X', \tag{1}$$

where B_0 is a vector of regression parameters and b_0 is a scalar intercept term, both of which are determined from the training data. Note that in the neural network literature, these regression parameters are referred to respectively as the weights and biases of the network. Rewriting equation (1) for a single prediction, $\hat{y}_i = \beta_0 + \sum_{p=1}^{P} \beta_p x'_{i,p}$, it can be seen that an implicit assumption is that predictions can be made as a linear combination of the predictor variables in X'. Usually, it is also assumed that the error terms of this regression are independently and identically distributed (i.i.d) from a Gaussian distribution, i.e. it is assumed that $y_i = \beta_0 + \sum_{p=1}^{P} \beta_p x'_{i,p} + \varepsilon_i$, where $\varepsilon_i \sim N(0, \sigma)$, or equivalently, that $y_i \sim N(\beta_0 + \sum_{p=1}^{P} \beta_p x'_{i,p}, \sigma)$. Closed form formulae are available to fit the parameters of the linear regression, as well as provide uncertainty intervals. If linear regression was to be used as a predictive model, then the parameters could be fit on the training set and out of sample performance assessed on the test set, or an information criterion, such as the AIC could be used to assess likely predictive performance.

GLMs relax, to some extent, both of the assumptions expressed above, firstly, that the predictions $\hat{y}$ are a linear combination of the variables in X' and, secondly, that the error terms are independent and identically distributed (i.i.d) draws from a (conditional) Gaussian distribution. The predictions made using a GLM can be expressed as

$$\eta(\hat{y}) = b_0 + B_0^T X', \tag{2}$$

where η is a link function that is applied so that the assumption of linearity is met. Common examples of link functions in the GLM literature, are the *log* and *logit* functions, which are often used for modelling data using a Poisson or Binomial distribution, respectively. In addition, GLMs are able to use a variety of distributions for the errors, namely those distributions from the exponential family, for example, count data are often modelling with Poisson distribution. Since closed formulae for fitting the parameters of a GLM are usually unavailable, GLMs are fit using Iteratively Reweighted Least Squares (IRLS). For a recent survey of GLMs we refer to Wüthrich (2019b).

Example 3.1. A common activity for actuaries working in non-life insurance is reserving for claims that have occurred, but where the loss is not yet (fully) reported, commonly referred to as IBNR claims. Usually, an array of aggregated claims amounts attributable to a particular accident year is tracked as claims relating to that accident year are reported, producing a so-called reserving triangle. An example of such a triangle from Taylor and Ashe (1983) is shown in Table 1. Commonly, reserving for IBNR claims involves predicting in aggregate the value of claims in periods that have not yet been observed (although some methods involving individual claims are now being developed, see, for a recent example, De Felice and Moriconi (2019)), using methods such as the chain ladder and Bornhuetter-Ferguson techniques, reviewed in detail in Wüthrich and Merz (2012). It is well known that the results of the chain ladder technique can be reproduced using a Poisson GLM regression model, using the covariates of accident and development year to predict the incremental claims in each cell of the triangle (Renshaw & Verrall, 1998).

The code[2] to fit this GLM and produce predictions for future claims development is shown in Listing 1, which also shows how Mack's formulation of the chain ladder method (Mack, 1993) might be applied using the ChainLadder package (Gesmann *et al.*, 2020). Lines 3–4 in the code show how the GLM may be specified by treating each of the accident year and development year covariates as categorical variables using the command as.factor in the GLM formula. Internally within R, these categorical covariates are dummy coded, meaning to say that a separate parameter value is fit for each level of the variables, except for a baseline value of one of the variables which is redundant.

Line 11 of the listing tests whether the estimates of IBNR from the two methods are similar (using the syntax of the data.table package) and running the code shows these differ by an insignificant amount, i.e. both the traditional chain ladder method and the GLM produce the same best-estimate results.

Equation (2) can be written equivalently as

$$\hat{y} = \sigma(b_0 + B_0^T X'), \tag{3}$$

where σ is the inverse of the link function η. Equation (3) is the form more commonly found in the neural network literature, where σ is known as an activation function, and is the inverse of the

[2]To run this example in R, the reader should refer to the file example_IBNR.r on the associated GitHub repository.

```
### Predict using GLM model

glm_model = glm(value~ as.factor(origin) + as.factor(dev)-1,
                family = poisson(link = "log"), data = triangle)

test_model$predictions = glm_model %>%
        predict(newdata = test_model, type = "response")

### Apply Chain-ladder

CL = GenIns %>%
        MackChainLadder()

test_model[calendaryear>10, sum(predictions)] - (CL %>% summary)$Totals[4,1]
```

Listing 1: Code to fit GLM to triangle in Table 1.

GLM link function η. Instead of using IRLS, modern neural network software usually implements gradient descent algorithms to fit models of the form of equation (3).

Example 3.2. In this example, we show how to use the Keras package (Allaire & Chollet, 2017) in R to fit the same GLM discussed in the previous example[3]. On Line 3 of the listing, an input layer with the same number of dimensions as the dummy-coded matrix of covariates is initialised. This is "chained" with a so-called dense layer, which acts as the output of the network and calculates a regression parameter β for each input variable, i.e. the dense layer produces exactly the second part of the right hand side equation (3), $B_0^T X$. Lines 11–17 define the model and compile it with an optimiser and a loss function. In this case, we use the Poisson loss function so that the GLM results will be (closely) reproduced. Data to be fed into the network is derived on Lines 20–26, by dummy coding the categorical variables, resulting in sparse (i.e. most of the values are zero) matrices X' of 19 columns.

Finally, the network is fit using the commands on Lines 30–34. After running the data through the optimisation process about 1,000 times (referred to as epochs), the network converges to a very similar same solution as the GLM, as tested on Line 40.

Example 3.3. relies on techniques that are usually used to fit models to much larger datasets than the IBNR triangle discussed in this section, which require processing the data in batches rather than all at once. Thus, options such as setting the size of each batch passed to the optimiser, which appear to be unnecessary for such a simple example, are really intended for use in a different scenario. Rather than relying on dummy coded inputs to the GLM, in the following example, we illustrate the use of a much more efficient technique.

Example 3.4. Rather than rely on dummy-coded inputs, which even in this simple example creates a large matrix of 19 columns, another option when fitting neural networks is to use a type of look-up table, known as embedding layer (originally used by Bengio *et al.* (2003) in the context of natural language processing, and more recently in state of the art neural network models for tabular data (Guo & Berkhahn, 2016; Howard & Gugger, 2020)). An embedding layer maps an integer, representing a level of a categorical variable, to (a vector of) real values, which is calibrated using the gradient descent algorithm. More formally, if a categorical variable C has N levels, then an embedding layer looks up the value of each of the N levels with reference to a numerical vector $K_n \in \mathbb{R}^k$, $1 \leq n \leq N$, where the k values of K_n are free parameters that are fit together with the rest of the neural network, and where the dimension k is typically much smaller than N. Here, we

[3]To run this example in R, the reader should refer to the file example_IBNR.r on the associated GitHub repository.

Table 1. Claims triangle from Taylor and Ashe (1983)

	Development Year:									
Accident Year:	1	2	3	4	5	6	7	8	9	10
1	357,848	1,124,788	1,735,330	2,218,270	2,745,596	3,319,994	3,466,336	3,606,286	3,833,515	3,901,463
2	352,118	1,236,139	2,170,033	3,353,322	3,799,067	4,120,063	4,647,867	4,914,039	5,339,085	
3	290,507	1,292,306	2,218,525	3,235,179	3,985,995	4,132,918	4,628,910	4,909,315		
4	310,608	1,418,858	2,195,047	3,757,447	4,029,929	4,381,982	4,588,268			
5	443,160	1,136,350	2,128,333	2,897,821	3,402,672	3,873,311				
6	396,132	1,333,217	2,180,715	2,985,752	3,691,712					
7	440,832	1,288,463	2,419,861	3,483,130						
8	359,480	1,421,128	2,864,498							
9	376,686	1,363,294								
10	344,014									

```
### Define the network

input <- layer_input(shape = c(19), dtype = 'float32', name = 'input')

y = input %>%
  layer_dense(units = 1, activation = "exponential", name = 'y',
              use_bias = F)

### Define a Keras model

model <- keras_model(inputs = list(input), outputs = c(y))

adam = optimizer_adam(lr=0.1)

model %>% compile(
  optimizer = adam,
  loss = "poisson")

### Derive data for Keras
train_mat = (model.matrix(~as.factor(origin) + as.factor(dev) - 1, data = triangle)) %>%
  array(dim = c(55,19))
test_mat = (model.matrix(~as.factor(origin) + as.factor(dev) - 1, data = test_model)) %>%
  array(dim = c(test_model[,.N],19))

train_dat = list(input = train_mat, y = triangle$value)
test_dat = list(input = test_mat)

### Train the model

fit = model %>% fit(
  x = train_dat$input,
  y = train_dat$y,
  epochs = 1000,
  batch_size = 55,verbose = 1, shuffle = T)

### Derive predictions and test IBNR

test_model$preds_NN = (model %>% predict(test_dat$input))

test_model[calendaryear>10, sum(preds_NN)] - (CL %>% summary)$Totals[4,1]
```

Listing 2: Keras code to fit GLM to triangle in Table 1.

propose to use a 1-dimensional embedding layer in place of the dummy coded categorical variables, which reproduces the same values of the IBNR, but converges much quicker than the alternative. The major differences between Listings 2 and 3 appear in lines 1–7, where instead of allowing for a numerical input to the networks, Listing 3 sets up integer-valued input layers that feed into embedding layers. The values of the embeddings are added together to produce the output of the network. Figure 1 compares the convergence of the GLM models fit using the code in Listings 2 and 3 and shows that the embedding model converges substantially quicker than the dummy-coded model, reaching a stable loss within about 125 epochs, compared to 500 epochs for the dummy-coded model. While this performance gain is not really relevant in the case of this simple example, when dealing with much larger datasets, this becomes more important.

3.2. Neural Networks

Neural networks can be seen as generalisations of GLMs in the following manner: a neural network is a multi-layered machine learning model that learns a new representation of the predictor variables X. This new representation, which we call Z^L where L is the number of intermediate

```
### Define the network

ay <- layer_input(shape = c(1), dtype = 'int32', name = 'ay')
ay_fact = ay %>% layer_embedding(input_dim = 20, output_dim = 1) %>% layer_flatten()

dy <- layer_input(shape = c(1), dtype = 'int32', name = 'dy')
dy_fact = dy %>% layer_embedding(input_dim = 20, output_dim = 1) %>% layer_flatten()

y = list(ay_fact,dy_fact) %>% layer_add() %>%
  layer_dense(units = 1, activation = "exponential", name = 'y',
              use_bias = F, weights = list(array(c(1), dim = c(1,1))), trainable = F)

### Define a Keras model

model <- keras_model(inputs = list(ay,dy), outputs = c(y))

adam = optimizer_adam(lr=0.1)

model %>% compile(
  optimizer = adam,
  loss = "poisson")

### Derive data for Keras
train_dat = list(input = list(ay = triangle$origin, dy = triangle$dev), y = triangle$value)
test_dat = list(input = list(ay = test_model$origin, dy = test_model$dev))

### Train the model

fit_embed = model %>% fit(
  x = train_dat$input,
  y = train_dat$y,
  epochs = 1000,
  batch_size = 55,verbose = 1, shuffle = T)
```

Listing 3: Keras code to fit GLM to triangle in Table 1 using 1-dimensional embedding layers.

layers of the network, is then used within a GLM to make predictions. More formally, we define a deep feed-forward fully connected network with L intermediate layers as

$$\mathbf{Z}^1 = \sigma_0(c_0 + B_0' X), \tag{4}$$

$$\mathbf{Z}^2 = \sigma_1(c_1 + B_1' Z^1) \tag{5}$$

$$\vdots \tag{6}$$

$$\mathbf{Z}^L = \sigma_{L-1}(c_{L-1} + B_{L-1}' Z^{L-1}) \tag{7}$$

$$\hat{y} = \sigma_L(c_L + B_L' Z^L), \tag{8}$$

where, for $l \in \{1 \ldots L\}$, Z^l are the intermediate layers of the network, B_l are weight matrices containing regression parameters, c_l are intercepts, and σ_l are (non-linear) activation functions of the neural network which are applied component-wise to the elements of each vector. Note that whereas a vector of regression parameters was used for the linear regressions and GLMs, the intermediate layers of the neural network may contain several variables, and a separate row of parameters is required to derive each of these variables, i.e. B_l may be a matrix of parameters. In neural network terminology, the intermediate variables in each layer of the network are referred to as "neurons". We define a network as "deep" if L is at least 3; for L equal to 2 we call the network "shallow" and for L equal to 1, the network is nothing more than a GLM. This type of network is referred to as a "feed-forward" network since the input data travels in only one direction through the network (other types of neural network with recurrent connections can also be defined, see the

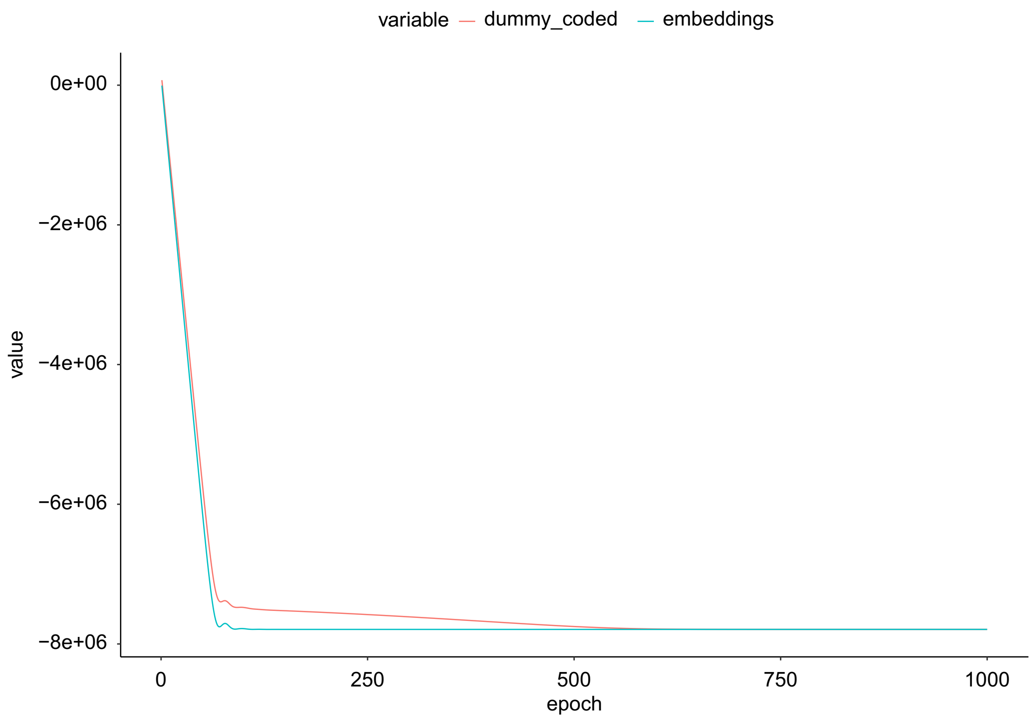

Figure 1. Comparison of the convergence of the GLM models fit using the Keras package in Listings 2 and 3.

later chapters of Goodfellow *et al.* (2016) for an overview). Equation (7) represents the last intermediate layer of the network, which acts as a "new" input to the final layer of the network in equation (8). Looking only at this last equation, it can be seen that it has the same form as the GLM in equation (3), i.e. the network can be interpreted as a two-stage model that first derives a new representation of the input data X by learning a mapping $X \mapsto Z^L$, which is then used in the second stage as inputs into a GLM.

The choice of the activation functions, σ_l, is often recommended in the machine learning literature to be the rectified linear unit (ReLu), which is nothing more than the function

$$\sigma_l(x) = max(0, x). \tag{9}$$

However, in Richman & Wüthrich (2019), and in other places, we have found that the use of the hyperbolic tangent activation functions, defined as

$$\sigma_l(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{10}$$

often provides better predictive performance than ReLU activations. In section 5.3, we also test the exponential linear unit (ELU) (Clevert *et al.*, 2016), which is defined as

$$\sigma_l(x, \alpha) = \begin{cases} x & if\ x > 0 \\ \alpha(e^x - 1) & if\ x \le 0 \end{cases}. \tag{11}$$

Although we have covered only one of the major types of deep neural networks – feed-forward networks – in this section, we note that the formulation of deep networks as GLMs is quite general. For example, most computer vision models replace equations (4)–(7) with features derived using convolutional neural networks, but maintain the final layer of the network which functions as a

```
### Define the network

ay <- layer_input(shape = c(1), dtype = 'int32', name = 'ay')
ay_fact = ay %>% layer_embedding(input_dim = 20, output_dim = 2) %>% layer_flatten()

dy <- layer_input(shape = c(1), dtype = 'int32', name = 'dy')
dy_fact = dy %>% layer_embedding(input_dim = 20, output_dim = 2) %>% layer_flatten()

y = list(ay_fact,dy_fact) %>% layer_concatenate() %>%
  layer_dense(units = 4, activation = "tanh") %>%
  layer_dense(units = 1, activation = "sigmoid", name = 'y')

### Define a Keras model

model <- keras_model(inputs = list(ay,dy), outputs = c(y))

adam = optimizer_adam(lr=0.1)

model %>% compile(
  optimizer = adam,
  loss = "poisson")

### Derive data for Keras
train_dat = list(input = list(ay = triangle$origin, dy = triangle$dev),
        y = (triangle$value - min(triangle$value))/(max(triangle$value) - min(triangle$value)))

test_dat = list(input = list(ay = test_model$origin, dy = test_model$dev))

### Train the model

fit_embed = model %>% fit(
  x = train_dat$input,
  y = train_dat$y,
  epochs = 1000,
  batch_size = 55,verbose = 1, shuffle = T)
```

Listing 4: Keras code to fit a neural network to triangle in Table 1 using 2-dimensional embedding layers and a single intermediate layer.

GLM, see, for example, He *et al.* (2016). More importantly, for actuarial applications, categorical covariates are usually treated using embedding layers (as shown in Example 3.3), which involve a simple modification to equation (4), replacing the input matrix X with a new matrix X', where the parameters describing the categorical variables are learned during the calibration of the network. Finally, the final layer of the network does not necessarily need to be a GLM learned together with the neural network, for example, Wüthrich (2019a) applies a GLM to the features in the last layer of a neural network to achieve a bias correction and Guo and Berkhahn (2016) use features learned by a neural network in a variety of other machine learning algorithm, such as random forests. In the following example, we provide an example of fitting a neural network model to the claims triangle.

Example 3.5. Defining a more richly parametrised neural network than the models shown previously is almost trivial using Keras. As a first step, on Lines 3–7 of Listing 4, the dimension of the embeddings is increased from 1 to 2, and as a second step, on Lines 9–11, an intermediate dense layer has been added, allowing for some representation learning to occur. This model provides a much closer fit to the data than the GLM did, as shown in Figure 2. Thus, one conclusion from this toy example is that neural network models are capable of fitting data much more closely than traditional actuarial models.

Despite this closer fit, we should be wary of uncritically accepting results from a relatively complicated model fit to relatively few data points (the triangle only contains 55 points). Ideally, we

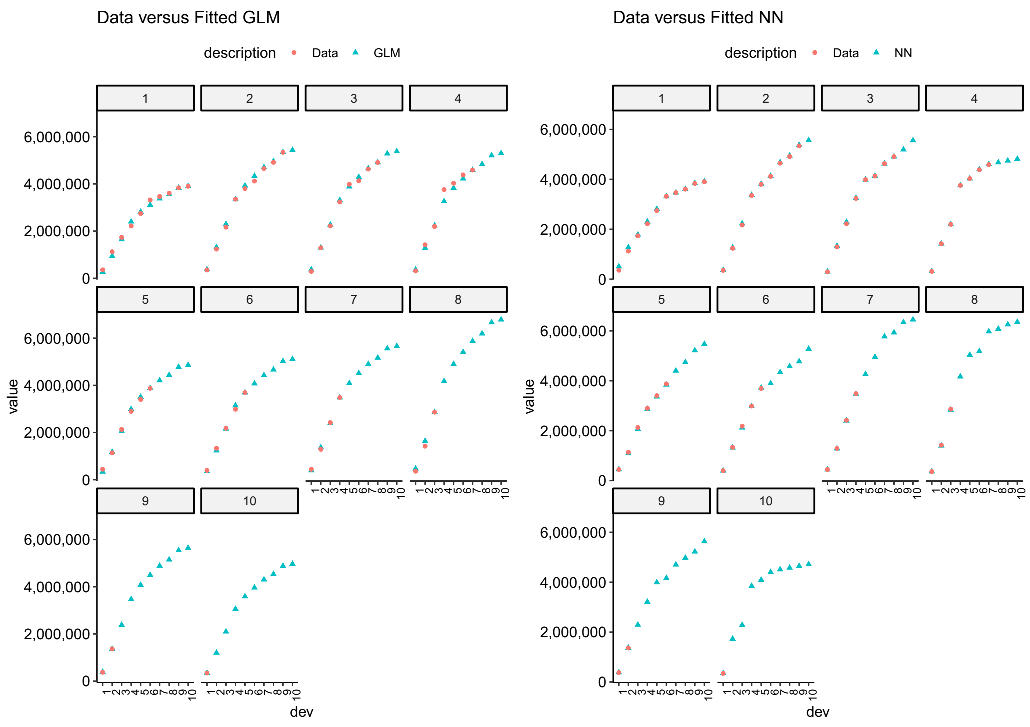

Figure 2. Comparison of the fits of the GLM and neural network models to the data shown in Table 1.

should assess generalisation performance of all of the models using a technique such as cross-validation. If performance was found to be poor, then a less complex model could be considered, or the model could be regularised using some of the techniques discussed at the end of this section. Only then would we be able to draw some conclusions as to the predictive power of the models. Nonetheless, we refrain from performing this analysis here, but turn our focus to more complex models in the next section.

3.3. Conclusion

We do not intend to provide a full introduction to neural networks in this work, since this has been done elsewhere and we rather wish to focus on the issues described in the introduction, but we nonetheless note briefly several practical issues that should be considered when neural network models are used[4].

In equation (4), we have defined the input to the neural network as X and not the transformed feature matrix X'. Ideally, we would like for the network to learn all of the transformations of X necessary to achieve an optimal result. In practice, this is usually only possible if the correct architecture has been specified for the network, for example, using an embedding layer for categorical covariates. Thus, instead of performing feature engineering, when fitting neural networks, the modeller is instead required to consider the somewhat different task of *architecture specification*. Nonetheless, when providing inputs to a neural network, a particular recommendation is to scale

[4]For more detail on these issues we refer to Richman (2018) in an actuarial context, Goodfellow *et al.* (2016) in a general machine learning context and for best practices in designing deep neural networks for a range of applications including computer vision and natural language processing, to Howard and Gugger (2020).

all of the variables with X, as well as the output y, to lie within the interval $[0, 1]$, since this step is usually necessary for the network to converge (another option is to standardise the input variables by subtracting the (sample) mean and dividing by the standard deviation). When specifying the architecture of a network, one may also get good performance through using a shallow network (i.e. when $L = 2$), if the data are small, though usually deep networks outperform shallow networks. Similarly, increasing the number of neurons in each layer usually leads to better performance but it is usually more efficient to increase the depth of the layers rather than the width of each layer. While assessing training set performance (as we have done in this chapter) is important to ensure that a model has enough representational capacity to fit the data, more important are assessments of predictive performance on the validation and test set. In the example of the reserving triangle used in this section, this could have been accomplished by leaving the most recent diagonal of the triangle (in other words, the claims in the most recent calendar period) out of the training set and validating the performance of the neural network on this diagonal. Generally, designing and parametrising models that fit the training data well is not sufficient to produce good performance on the validation and test sets, thus, regularisation techniques are often applied when using deep neural networks. One of the most popular of these techniques is dropout (Srivastava *et al.*, 2014), which randomly leaves out neurons from the network while it is being trained. Another technique, originally introduced to enable the training of deep networks for computer vision, is batch normalisation (Ioffe & Szegedy, 2015) which normalises the values of the neurons in each layer to lie between $[0, 1]$, that has been shown to have a regularising effect on the predictions of neural networks, as well as making the networks easier to calibrate (see, e.g. Luo *et al.* (2018)). Other options for regularising a network are including $L1$ and $L2$ penalties on the weights of the network.

We consider the implementation of some of these best practices on the models in the next section, and the effect of modifying the hyperparameters of neural networks in the subsequent section.

4. Example Applications of Deep Neural Networks

In the rest of this study, we focus on an example network of the form of equations (4)–(8), with the addition of embedding layers for categorical variables (as described in Example 3.3), that we apply within the context of mortality forecasting. We describe this application in the current section and then use the model to demonstrate the techniques in the rest of the paper.

4.1. Mortality Forecasting

Mortality rates are a fundamental input into many actuarial tasks such as the pricing and valuation of life insurance portfolios, pensions and social security schemes. With the introduction of risk-based capital and own-risk assessment requirements in many jurisdictions around the world (such as in the Solvency II regime in Europe), modelling the uncertainty around current and future mortality rates has also become important. Concurrent with these developments is the availability of databases containing high quality mortality data, measured contemporaneously, for many populations (such as in the Human Mortality Database (HMD) (Human Mortality Database, 2020)). Other advances include the provision of more detailed mortality information, such as cause of death information (Danilova *et al.*, 2020) or mortality rates reported for different levels of socio-economic factors (Cairns *et al.*, 2019). These new data sources pose some challenges for modellers who may be used to applying mortality models to simpler data (such as the famous Lee Carter model (Lee & Carter, 1992) which is applied to single-population mortality data). Recently, several studies have applied deep neural networks to model and forecast the mortality rates appearing in the HMD: Richman & Wüthrich (2019) apply a deep neural network with embeddings to model the mortality of 76 populations simultaneously and show that their model outperforms more traditional multi-population mortality models and Nigri *et al.* (2019) apply recurrent neural networks to forecast the time element of the Lee Carter model (in other words, k_t) and show that their approach outperforms the ARIMA models usually used for making these forecasts. Finally, Perla *et al.* (2020) apply convolutional and recurrent neural networks to model the time series of mortality rates in the HMD and the United States Human Mortality Database (USMD) and find that these types of neural networks, which are specialised to process time series data, outperform standard feed forward neural networks.

In this study, we consider the case of forecasting mortality data for both national and sub-national populations simultaneously. In the case when sub-national data are not available, then we only forecast mortality at the country level, whereas if sub-national data are available (such as for the USA, Japan, Canada and Australia (Ishii *et al.*, 2020; Payeur *et al.*, 2020; Romo, 2020; Wilmoth *et al.*, 2020)), then we also produce forecasts for each region within the overall country.

We define $\mu_{x,t}^{p,g}$ as the force of mortality for (sub-)population p for those of gender g aged x in year t. Here, we consider P different populations, comprised of those living in a particular region or sub-national region of a country. In this study, we consider 41 different countries, and, for 4 of these (namely, the United States, Canada, Japan and Australia), we consider 117 different sub-regions, such that $p \in [1, 158]$. We restrict this study to focus on forecasting the mortality rates of males and females in the age range 0–99. Data covering the years 1950–2017 were downloaded from the HMD (Human Mortality Database, 2020), and the four associated databases covering the specific countries mentioned above[5]. The data up until the year 2000 is used as a training set, and from 2000 to 2017 is used as the testing set. These data consist of three categorical variables indicating country, sub-national region and gender, and two numerical variables, age and year. For the purpose of modelling these data, we treat the age variable, x, as a categorical variable as well. All of these categorical variables are mapped to d-dimensional embeddings, where d is 5 for age and gender and 10 for country and sub-national region. Similarly to Richman and Wüthrich (2019), the year variable, t enters the model as a numerical variable. Thus, a 31-dimensional feature vector, $feature_{x,t}^{p,g}$ is fed into the neural network, which consists of 5 layers of 128 neurons, with hyperbolic tangent activations (this network was found in Richman and Wüthrich (2019) to perform well on the HMD data set, and better than an equivalent network with ReLU activations, see also section 5.3 below), defined as follows:

$$\mathbf{Z}^1 = \sigma_0(c_0 + B_0' feature_{x,t}^{p,g}), \tag{12}$$

$$\mathbf{Z}^2 = \sigma_1(c_1 + B_1' Z^1) \tag{13}$$

$$\vdots \tag{14}$$

$$\mathbf{Z}^4 = \sigma_3(c_3 + B_3'(Z^3, feature_{x,t}^{p,g}))) \tag{15}$$

$$\mathbf{Z}^5 = \sigma_4(c_4 + B_4' Z^4) \tag{16}$$

$$\hat{y} = \sigma_5(c_5 + B_5' Z^5) \tag{17}$$

While similar to the equations defining the neural network in the previous section, the layer of the network defined by equation (15) also contains a so-called skip connection between the feature vector and the fourth layer of the network. Skip connections have been shown to improve the training of deep neural networks used for computer vision, see, for example, He *et al.* (2016), and have been used in several actuarial applications of deep learning (Richman & Wüthrich, 2019; Schelldorfer & Wüthrich, 2019; Wüthrich & Merz, 2019). The code to produce this network

[5]To derive the data used in this example in R, the reader should refer to the file munge_HMD.r on the associated GitHub repository.

Table 2. Test set mean squared error (multiplied by 10^4) using the LC and neural network models, national and sub-national populations

	Model	type	Average MSE	Median MSE	Best Performance
1	LC_SVD	National	5.55	2.48	5
2	LC_SVD	Sub-National	22.08	1.48	20
3	DEEP	National	2.38	1.31	71
4	DEEP	Sub-National	20.49	0.78	216

appears in Listing 5. We train the network 10 times and use the average prediction of the networks in the years 2000–2017 for comparison to the actual mortality rates.

We compare the results of this model to Lee-Carter (LC) (Lee & Carter, 1992) models fit to each country, sub-national region and gender separately[6]. The LC model is defined as

$$\mu_{x,t} = a_x + b_x k_t, \tag{18}$$

where a_x is the average log mortality rate at age x in the period, b_x is the rate of mortality improvement and k_t is the value of the mortality improvement index in year t. The LC models were fit using the singular value decomposition (SVD) of the matrix of centred log mortality rates and the time index, k_t was forecast using a random walk with drift.

Table 2 shows the average and median mean squared error (MSE) achieved by the neural network and LC models on the test set[7]. It can be seen that, both for the neural network and LC models, the average MSE is significantly lower for national populations than sub-national populations, whereas the median MSE is lower for sub-national populations as compared to national populations, indicating that in some instances of the sub-national data, the models perform quite poorly and produce high MSE values, whereas the models perform well on most of the sub-national populations. The neural network achieves substantially better performance than the LC model on all metrics considered here, with lower MSE scores on 71 out of 76 national populations and 216 out of 236 sub-national populations. Figure 3 shows that both the LC and neural network models have similar patterns of performance across the different countries, with the worst performance (at the right of each panel of the figure) for Australian and Canadian sub-national regions.

Remarks 4.1. In this study, following common practice in fitting neural networks, we use the Adam optimiser to calibrate the neural networks (Kingma & Ba, 2014). Recently, several studies of speeding up the training of neural networks through modifying the learning rate of the optimiser have appeared (Loshchilov & Hutter, 2019; Smith, 2017). Here, we apply a simple version of the restart technique, by training the network for 50 epochs, then restarting the optimiser and training for another 50 epochs. We show below that this improves out of sample predictive accuracy.

4.2. Conclusions

We have presented a neural network-based mortality forecasting model in this section. This mortality model treats the time dimension as an input into the regression, similar to the IBNR

[6]We utilize the LC model as a standard against which to compare the deep neural networks since this is suitable for forecasting the lifetable at all ages, whereas other popular mortality forecasting models often focus only on old age mortality.

[7]To derive the results used in this example in R, the reader should refer to the file HMD model.r on the associated GitHub repository.

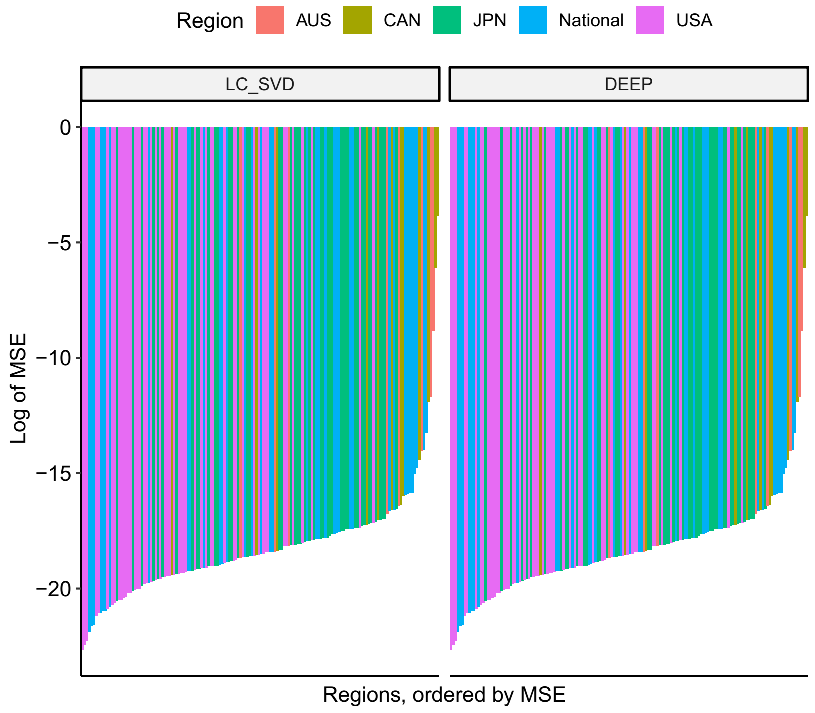

Figure 3. Logarithm of out of sample MSE for each region for which a forecast has been made. Lower values (i.e. more negative) indicate better performance. Plots are colour-coded for national populations, and for the country in which sub-national forecasts have been made.

reserving model in the previous section that projects future claims amounts to be reported in the future based on accident and development year variables that are input into the model.

Of course, there are other types of models that are of interest to actuaries, such as models that do not make forecasts ahead in time, but rather make predictions at the same point in time, for example, general insurance pricing models. Other models, for example, the reserving model of Kuo (2019) or the time series model of Perla *et al.* (2020), use past observations of loss amounts or mortality rates directly to produce forecasts. The analysis presented in the following sections is likely to be equally applicable to these types of models.

5. Variability of Neural Network Results

In this section, we investigate how neural network predictions vary each time the network is trained. On the one hand, predictions that change with each training run are problematic to explain to stakeholders in the modelling process and are not suitable for most actuarial work due to the lack of reproducibility. On the other hand, by averaging the results of multiple training runs of neural networks, it is often found that better predictive performance can be achieved, compared to the predictive performance of a single neural network. The stability of neural network predictions at the portfolio and policy level has been studied in Richman and Wüthrich (2020), who call these aggregated network results the "nagging predictor", in the spirit of the "bagging" (bootstrap aggregating) predictor of Breiman (1996). The phenomenon of unstable neural network results has also been discussed in several places in the actuarial literature (Richman *et al.*, 2019; Richman & Wüthrich, 2019; Schelldorfer & Wüthrich, 2019) and in the wider deep learning literature (Guo & Berkhahn, 2016).

Here, we investigate how the stability of neural network predictions varies as different hyper-parameter choices are made for different parts of the networks. Whereas increasing the variability of predictions may be undesirable if only a single neural network has been fit, it is likely the case that averaging the predictions of somewhat more variable networks will lead to better results than the averaged predictions of more stable neural networks. Using the example neural network presented in section 4, we study the effect of changing each of the components of the networks on the stability of the results, as measured by the standard deviation of the MSE on the test set data. By understanding these effects on this example network, actuaries can draw parallels to their own modelling examples and either look to minimise the variability of a single neural network, or design networks with an optimal amount of variability to produce averaged results that are as predictive as possible.

Since training neural networks is a random process, partly due random batches of data being fed to the network during training and the application of dropout, which randomly sets some neurons of the network to zero, an initial hypothesis is that smaller batches and higher dropout rates will lead to increased randomness of the neural network predictions. We test this hypothesis and explore other options as we discuss these components of the networks in the following sub-sections:

- dimension of the intermediate layers;
- dimension of the embedding and convolutional layers;
- activation function of the intermediate layers;
- application of batch normalisation;
- depth of the network;
- drop-out rates;
- size of batches; and
- learning rate, restarts and optimiser.

To carry out these experiments, ten training runs of each modified neural network were performed. To investigate the stability of the neural network predictors, the average test set mean squared error (MSE) and standard deviation of the MSE over these ten runs were recorded. These estimates, which are summary statistics calculated over the ten training runs, are shown in the first table in each of the following sub-sections. Also, the MSE of the averaged predictions of the ten runs on the test set was recorded, as well as the number of times that the model beat the LC model. These estimates, which are estimated first by averaging the predictions, then calculating the MSE, are shown in the second table in each of the following sub-sections.

5.1. Dimension of Intermediate Layers

An inspection of the number of parameters in the mortality forecasting model shows that the majority of the parameters are in the intermediate fully connected layers of the network. Here, we consider the effect of varying the number of neurons in each intermediate layer (in other words, the width of each layer) from 32 to 1,024 in multiples of 2. Intermediate layers with 32 neurons probably do not have enough representational capacity to fit the mortality data well, whereas 1,024 neurons may overfit the training data (despite the regularisation of the network using dropout and batch normalisation). It is unclear what the effect of varying the neurons in each layer will be on the stability of the results of the network.

Table 3 shows that the baseline model discussed in section 4.1 with intermediate layers of 128 neurons achieves the lowest standard deviation of MSE, and both the narrower and wider models are less stable. Two of the networks with wider layers (of size 256 and 512 neurons) achieve better performance than the baseline and the narrower networks. However, Table 4 shows that despite the baseline network attaining the lowest standard deviation of the MSE, two of the wider networks (with 256 and 512 neurons in each layer) outperform the baseline network, as measured

Table 3. Average and Standard Deviation of MSE over ten runs of the mortality forecasting model on the test set (multiplied by 10^4), with the width of the intermediate layers being varied as noted in the description column. Results sorted by Standard Deviation of MSE

	Description	Average MSE	Std Dev of MSE
1	Baseline	12.55	0.18
2	Width: 256	12.47	0.26
3	Width: 512	12.43	0.28
4	Width: 1,024	12.67	0.33
5	Width: 32	13.16	0.35
6	Width: 64	13.01	0.40

Table 4. MSE of the average prediction of ten runs of the mortality forecasting model on the test set (multiplied by 10^4) and number of times that the LC model is beaten, with the width of the intermediate layers being varied as noted in the description column. Results sorted by MSE

	Description	MSE	Best Performance over Populations
1	Width: 512	12.18	296
2	Width: 256	12.27	288
3	Baseline	12.44	287
4	Width: 1,024	12.45	279
5	Width: 64	12.87	247
6	Width: 32	13.02	235

both by the MSE and outperformance of the LC model. Nonetheless, the widest network tested, of 1,024 neurons, performs slightly worse than the baseline. These findings can be interpreted intuitively: the ensemble predictions of the networks benefit from some increase in "diversity" of the predictions (as measured by standard deviation of the MSE), compared to the baseline model, however, too much variability eventually hurts performance (see also Wilson and Izmailov (2020) who make a similar argument). Moreover, the ensemble results benefit only when the networks contributing predictions perform well in of themselves, as can be seen from the relatively poor performance of the narrower networks. Thus, one can conclude that maximising the stability of neural network solutions may in fact not be a good strategy in practice, if the networks will be used in an ensemble.

We also comment that traditional actuarial and statistical modelling often seeks to minimise the number of parameters in a model (i.e. to find a parsimonious model). In this case, we find that a model with a very large number of parameters exhibits excellent predictive performance, suggesting that the traditional bias towards smaller models should be interrogated carefully when using deep neural networks especially when these are regularised.

5.2. Dimension of the Embedding Layers

Here, we test the impact of varying the dimensions of the embedding layers in the mortality forecasting model. The baseline mortality forecasting model used embeddings of 5 dimensions for age and gender and of 10 dimensions for country and sub-national region. In addition, the second last layer of the network contained a skip-layer connecting the deeper layers of the network directly to

Table 5. Average and Standard Deviation of MSE of ten runs of the mortality forecasting model on the test set (multiplied by 10^4), with the dimension of the embedding layers being varied as noted in the description column. Results sorted by Standard Deviation of MSE

	Description	Average MSE	Std Dev of MSE
1	Embedding: 10 dim no-skip	12.66	0.09
2	Embedding: 10 dim	12.43	0.17
3	Baseline	12.55	0.18
4	Embedding: 5 dim	12.49	0.21
5	Embedding: 20 dim	12.49	0.23
6	Embedding: 2 dim	13.78	0.70

Table 6. MSE of the average prediction of ten runs of the mortality forecasting model on the test set (multiplied by 10^4) and number of times that the LC model is beaten, with the dimension of the embedding layers being varied as noted in the description column. Results sorted by MSE

	Description	Average MSE	Best Performance over Populations
1	Embedding: 10 dim	12.26	295
2	Embedding: 20 dim	12.34	298
3	Embedding: 5 dim	12.35	294
4	Baseline	12.44	287
5	Embedding: 10 dim no-skip	12.53	274
6	Embedding: 2 dim	13.32	224

the embeddings. The impact of varying the dimensions of the embeddings between 2 and 20 dimensions, as well as removing the skip-connection is tested in this section.

Table 5 shows that the lowest standard deviation is achieved by the model with no skip connections; however, the performance of this network is significantly worse than the rest of the models, i.e. here we replicate the finding of Richman and Wüthrich (2019) that adding a skip connection improves performance of neural networks for mortality forecasting. Increasing the dimension of the embedding layers for age and gender to 10 dimensions produces the best performance among the networks tested, as well as the best performance of the ensemble, as measured by the MSE (shown in Table 6). The best performance versus the baseline LC model is produced by networks with embeddings of 20 dimensions. Interestingly, the performance of the network is enhanced by having embeddings with the same number of dimensions (whether these are 5 or 10). One can conclude that minimising standard deviation of the network solutions is not a good strategy from the perspective of maximising the performance of the averaged predictions. Another conclusion, similar to the previous section, is that more highly parametrised networks seem to perform better on this data set.

5.3. Activation Function of Intermediate Layers

The mortality forecasting network utilised hyperbolic tangent (tanh) activation functions, based on prior experience with the HMD data in Richman and Wüthrich (2019), where it was found that networks with tanh activations outperformed networks with ReLU activations (which were defined in equation (9)). On the other hand, the machine learning literature often recommends the use of ReLu activation functions. Since ReLU units may output 0, it is fair to hypothesise that

Table 7. Average and Standard Deviation of MSE of ten runs of the mortality forecasting model on the test set (multiplied by 10^4), with the activation function of the intermediate layers being varied as noted in the description column. Results sorted by Standard Deviation of MSE

	Description	Average MSE	Std Dev of MSE
1	Baseline	12.55	0.18
2	Activation: ReLu	13.23	0.50
3	Activation: ELU	13.00	0.51

Table 8. MSE of the average prediction of ten runs of the mortality forecasting model on the test set (multiplied by 10^4) and number of times that the LC model is beaten, with the activation function of the intermediate layers being varied as noted in the description column. Results sorted by MSE

	Description	Average MSE	Best Performance over Populations
1	Baseline	12.44	287
2	Activation: ELU	12.74	259
3	Activation: ReLu	13.01	231

predictions based on ReLU networks will be more unstable than those from tanh networks. We also test ELU units, defined in equation (11), which do not output zero, but rather a value that depends both on the input to the ELU unit and a learned parameter. Whereas we expect ELU units to produce less variation than ReLU units, nonetheless, for the same amount of regularisation, the performance on test data is likely to be a little worse, given the extra parameters in ELU networks.

Somewhat surprisingly, the performance of both the ReLu and ELU networks is significantly worse than the tanh networks across all metrics (see Tables 7 and 8), although the ELU network appears to perform better than the ReLu network. Notably, this is despite the application of batch normalisation and dropout within the networks. It could be argued that the choice of parameter initialisation schemes, learning rates and optimisers is not optimal for these alternative activation functions (and indeed, no further effort to tune these choices was made), nonetheless, these results reinforce that tanh networks should usually be considered for actuarial modelling.

5.4. Application of Batch Normalisation

Batch normalisation (Ioffe & Szegedy, 2015) is a standard technique that is applied both to make the training of deep networks easier, as well as for its effect as a regulariser, see, for example, Luo *et al.* (2018). This technique acts by standardising (subtracting the mean and dividing by the standard deviation) the network activations to have a zero mean and a standard deviation of unity. Batch normalisation was applied in two places in the mortality forecasting networks, as can be seen in Listing 5: firstly, after the embedding layers, and secondly, between each dense layer in the network. Here, we experiment with removing all batch normalisation from the network, and then, only from the embedding layers.

We note that since tanh activations are constrained to the range $(-1, 1)$, it might be thought that applying batch normalisation will have little beneficial effect. It can be seen in Table 9 that this is not the case and that the performance of the network is improved by applying batch normalisation both to the embedding layer and the dense layers of the network. However, the standard deviation of the MSE increases as batch normalisation is added to more parts of the network. Based on the results in Table 10, it appears that batch normalisation should be applied both after embedding and dense layers of deep neural networks. Furthermore, although batch normalisation

Table 9. Average and Standard Deviation of MSE of ten runs of the mortality forecasting model on the test set (multiplied by 10^4), with the application of batch normalization being varied as noted in the description column. Results sorted by Standard Deviation of MSE

	Description	Average MSE	Std Dev of MSE
1	Batchnorm: none	13.04	0.12
2	Batchnorm: not on embedding	12.75	0.14
3	Baseline	12.55	0.18

Table 10. MSE of the average prediction of ten runs of the mortality forecasting model on the test set (multiplied by 10^4) and number of times that the LC model is beaten, with the application of batch normalization being varied as noted in the description column. Results sorted by MSE

	Description	Average MSE	Best Performance over Populations
1	Baseline	12.44	287
2	Batchnorm: not on embedding	12.65	272
3	Batchnorm: none	12.97	233

Table 11. Average and Standard Deviation of MSE of ten runs of the mortality forecasting model on the test set (multiplied by 10^4), with the depth of the network being varied as noted in the description column. Results sorted by Standard Deviation of MSE

	Description	Average MSE	Std Dev of MSE
1	Baseline	12.55	0.18
2	Depth: 10 layers	12.70	0.27
3	Depth: 15 layers	13.20	0.55

is usually associated with ReLu activation functions, we see that nonetheless, there is a beneficial effect even on tanh networks.

5.5. Depth of the Network

The mortality forecasting model is an example of a relatively deep network, compared to many usual applications of neural networks to tabular data. In this section, we experiment with making the network even deeper, by adding 5 and 10 more layers to the network. It is well known that deeper networks are harder to optimise, thus, it is fair to speculate that increased depth may result in worse results.

The results in Tables 11 and 12 show that increasing the depth of the networks results in worse and more variable performance than the baseline network. While this section shows that adding depth does not result in improved performance, nonetheless, other ways of building deeper networks could be explored, for example, using the fully connected architecture of Huang *et al.* (2017) or by using the ResNet architecture of He *et al.* (2016).

5.6. Dropout Rates

Here, we test whether modifying the dropout rate within the mortality forecasting network might produce more optimal results. The baseline network applies dropout to the embedding layers with a chance of a neuron being dropped set to 4%, and to the intermediate layers at 8%, and here we vary these percentages (in the description column of Tables 13 and 14, the first percentage refers to

Table 12. MSE of the average prediction of ten runs of the mortality forecasting model on the test set (multiplied by 10^4) and number of times that the LC model is beaten, with the depth of the network being varied as noted in the description column. Results sorted by MSE

	Description	Average MSE	Best Performance over Populations
1	Baseline	12.44	287
2	Depth: 10 layers	12.53	243
3	Depth: 15 layers	12.96	189

Table 13. Average and Standard Deviation of MSE of ten runs of the mortality forecasting model on the test set (multiplied by 10^4), with the application of drop out being varied as noted in the description column. Results sorted by Standard Deviation of MSE

	Description	Average MSE	Std Dev of MSE
1	Dropout: 4%/15%	12.63	0.10
2	Dropout: 4%/25%	12.73	0.13
3	Baseline	12.55	0.18
4	Dropout: 4%/50%	13.57	0.33
5	Dropout: 8%/5%	13.06	0.36
6	Dropout: 4%/50% last layer	13.72	0.91

Table 14. MSE of the average prediction of ten runs of the mortality forecasting model on the test set (multiplied by 10^4) and number of times that the LC model is beaten, with the application of drop out being varied as noted in the description column. Results sorted by MSE

	Description	Average MSE	Best Performance over Populations
1	Baseline	12.44	287
2	Dropout: 4%/15%	12.57	284
3	Dropout: 4%/25%	12.64	276
4	Dropout: 8%/5%	12.92	225
5	Dropout: 4%/50% last layer	13.19	213
6	Dropout: 4%/50%	13.50	159

the dropout on the embedding layer and the second to dropout applied to the intermediate layers). We also experiment with a network that only applies dropout to the last layer of the network.

Somewhat unintuitively, Table 13 shows that increasing the dropout rate of the intermediate layers acts to reduce the standard deviation of the MSE. Increasing the dropout rate applied to the embedding layer greatly reduces performance and increases the standard deviation, as does applying dropout only to the last layer.

5.7. Learning Rate, Restart and Batch Size

Here, we address hyperparameter choices relating to the optimisation of the network. The performance of neural networks is often closely linked to the learning rate used in the optimiser, as well as to the batch size used in the optimisation process. In addition to these choices, as discussed in section 4, we employ a "restart" of the optimiser when fitting the mortality forecasting network.

Table 15. Average and Standard Deviation of MSE of ten runs of the mortality forecasting model on the test set (multiplied by 10^4), with the optimization parameters being varied as noted in the description column. Results sorted by Standard Deviation of MSE

	Description	Average MSE	Std Dev of MSE
1	Baseline	12.55	0.18
2	Optimization: half batch size	12.39	0.21
3	Optimization: double LR	12.36	0.24
4	Optimization: no restart	12.98	0.44
5	Optimization: half LR	13.34	0.73
6	Optimization: double batch size	13.32	0.75

Table 16. MSE of the average prediction of ten runs of the mortality forecasting model on the test set (multiplied by 10^4) and number of times that the LC model is beaten, with the optimization parameters being varied as noted in the description column. Results sorted by MSE

	Description	Average MSE	Best Performance over Populations
1	Optimization: double LR	12.23	300
2	Optimization: half batch size	12.26	299
3	Baseline	12.44	287
4	Optimization: no restart	12.79	249
5	Optimization: double batch size	13.02	210
6	Optimization: half LR	13.04	202

Here we experiment with different choices relating to the optimisation process and show these results in Tables 15 and 16.

The results of Table 16 are similar to those of Table 4, in that small increases in standard deviation results in better performance of the network, whereas larger increases result in degraded performance. Significantly better performance than the baseline can be achieved by doubling the learning rate or halving the batch size. However, halving the learning rate and doubling the batch size, as well as removing the restarts, harms the performance of the network significantly.

5.8. Summary

When considering the mortality forecasting model, we have found that modest increases in the standard deviation of the predictions from the model have often led to better performance of the averaged prediction of the networks. In Figure 4, we compare the performance of the averaged predictions against the standard deviation of the individual predictions. It can be seen that the baseline model is quite close to optimal, but benefits from changes that increase the standard deviation only marginally. Models that produce predictions that are either too stable or too unstable tend to produce worse predictions when averaging over multiple runs.

In this section, we have investigated how different choices in neural network architecture can influence the variability of the predictions made and the accuracy of the averaged predictions. This has been done by varying a single element of the models at a time. On the other hand, it might be found that varying two or more elements of the model simultaneously could lead to improved

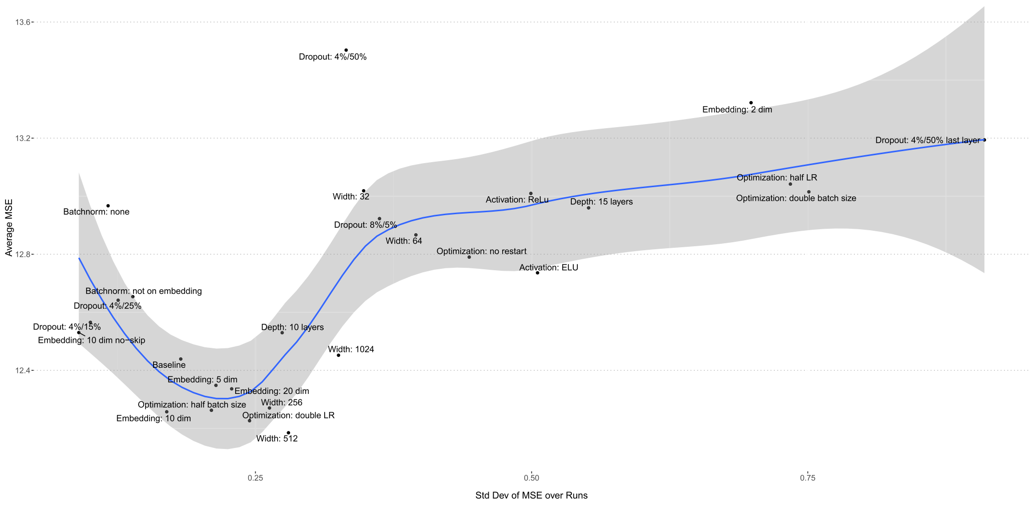

Figure 4. Comparison of the performance of the averaged neural network predictions on the test set, measured by MSE, compared to the standard deviation of the MSE produced by the individual predictions over different training runs.

performance and different conclusions. For example, it was shown above that the performance of the mortality forecasting model with layers of 1,024 neurons is worse than the model with layers of 512 neurons. However, if the dropout rate was increased, it might be found that the wider model is in fact more optimal. Nonetheless, the conclusion from this section is that the most predictive neural network results are those that are averaged over multiple training runs of models that produce predictions with variability that is neither too extreme nor too minimal.

6. Uncertainty Estimation

In this section, we discuss how the neural network presented in section 4 can be extended from producing only a best estimate to producing also estimates of uncertainty around the predictions. A paradigm for estimating uncertainty within the actuarial literature is the estimation of uncertainty of IBNR reserve estimates, which is usually estimated by considering two types of uncertainty that may affect estimates from models: parameter uncertainty and process variance, see, for example, section 7 in England and Verrall (2002). The former type of uncertainty relates to the uncertainty in model parameters and has often been quantified in IBNR reserving using the bootstrap, for an early example, see England (2002). The latter type of uncertainty relates to the random variation of forecast quantities around their best estimate prediction. Recently, there has been some interest in quantifying the accuracy of traditional measures of uncertainty used in IBNR reserving, see Gabrielli and Wüthrich (2019), who find that, in the case of the chain-ladder method, the estimate of the MSE of prediction of the IBNR underestimates the true uncertainty for small and medium size portfolios.

On the other hand, the machine learning literature, which has as its focus the task of prediction, has focussed less on uncertainty arising from parameter error, and more on estimating directly the conditional quantiles of predictions, see, for example, the quantile random forest method of Meinshausen (2006) or the more general study of Takeuchi *et al.* (2006).

Uncertainty of predictions has been used by actuaries both as a way of quantifying and communicating how certain actuarial projections are, as well as to estimate the capital requirements of financial services companies (for a discussion of the link between these two objectives in IBNR reserving, see England *et al.* (2019)). In addition, we see uncertainty estimation as playing another role, which is to enable actuaries to investigate which model predictions are more uncertain, and take corrective action, for example, by increasing margins in areas of higher uncertainty or considering other risk management techniques such as reinsurance.

Here, we adapt two techniques from the machine learning literature to estimate prediction uncertainty in the context of mortality modelling. These techniques are the non-parametric quantile regression approach using the so-called "pinball" loss Takeuchi *et al.* (2006) and the recent parametric approach of deep ensembles (Lakshminarayanan *et al.*, 2017). Both of these techniques have achieved considerable success in estimating the uncertainty of predictions from deep learning models (Ovadia *et al.*, 2019; Smyl, 2020; Wilson & Izmailov, 2020); the latter is often cited as the best performing technique in the deep learning literature, see, for example (Mukhoti *et al.*, 2021).

First, we provide definitions that are only used in this section, following which we discuss the application of these techniques.

6.1. Definitions

Here we are concerned with estimating upper and lower bounds of the predictions made by the deep learning models presented above. These bounds are often represented using quantiles. Above, we have defined that the usual aim of a predictive model is to produce an accurate estimate $\hat{y}_i$ of unseen variables y_i on the basis of variables X_i'. We note that unseen y_i are random variables usually following an unknown distribution. In addition to predicting a best estimate of y_i we are also interested in how y_i is distributed. Here, we focus on estimating the quantiles of the distribution of y_i, denoted as y_i^τ, where y_i^τ is the smallest quantity such that $P(y_i < y_i^\tau) = \tau$. We now consider that for the purpose of uncertainty estimation, the actuary has chosen an upper and lower quantile of the distribution of y_i that must be estimated, denoted as τ_u and τ_l respectively (other measurements of the distribution of y_i might be of interest, but here we have chosen quantiles of the unknown distribution since these are often used in insurance regulation, for example, the use of Value-at-Risk in Solvency II, which depends on nothing more than the quantile of an insurer's loss distribution). If the quantiles are calibrated correctly, then the upper and lower bounds $y_i^{\tau_u}$ and $y_i^{\tau_l}$ form a predictive interval into which observations of y_i should fall with probability $P(y_i^{\tau_l} \leq y_i \leq y_i^{\tau_u}) = \tau_u - \tau_l$. In practice, we do not know the actual quantiles of y_i, and therefore we try to make predictions of the lower and upper quantiles, $\hat{y}_i^{\tau_l}$ and $\hat{y}_i^{\tau_u}$, conditional on the known variables X_i'. To measure the accuracy of the predicted quantiles, we calculate the empirical coverage of the predictive interval formed by $[\hat{y}_i^{\tau_l}; \hat{y}_i^{\tau_u}]$ using the following estimator evaluated on test set data:

$$\frac{\sum_{i=1}^{N_{test}} 1_{y_i \in [\hat{y}_i^{\tau_l}; \hat{y}_i^{\tau_u}]}}{N_{test}}.$$

If the predicted quantiles are perfectly accurate, then the coverage should equal $\tau_u - \tau_l$. We estimate the quantiles by suitably adapting the deep learning models presented above for this task, using the two techniques we present next and consider the accuracy of the coverage on unseen data, in other words, on the test set for each model.

6.1.1. Quantile regression via the pinball loss

Quantile regression techniques aim to estimate the quantiles of the distribution of y_i directly, in other words, the model directly outputs the quantiles and does not rely on making distributional assumptions for y_i. We achieve this by modifying the loss function used when training the deep learning models from the MSE loss to the so-called pinball loss, which is defined as

$$L(y_i, \hat{y}_i, \tau) = \begin{cases} \tau(y_i - \hat{y}_i) & \text{if } y_i - \hat{y}_i \geq 0 \\ (\tau - 1)(y_i - \hat{y}_i) & \text{if } y_i - \hat{y}_i < 0 \end{cases}$$

and where τ is the selected quantile that the deep network is optimised to predict. Since we wish to derive a confidence band composed of two quantiles, we use a network that produces two outputs, which are the upper and lower percentiles τ_l and τ_u. An implementation of the pinball loss function in R appears in Listing 6.

6.1.2. Deep ensembles

The deep ensemble technique adds an estimate of the variance of each observation to the neural network, on the assumption that each observation i is Gaussian distributed, in other words, this technique applies a type of heteroscedastic Gaussian regression where both the expected value and variance of observation i are jointly predicted. To apply the technique, several of this same type of network are fit to the data and the resulting expected values and variances are averaged to derive the "ensemble" predictions for these quantities. Predictive intervals are then derived using the Gaussian parameters for each observation i, thus, different from the quantile regression approach above, the deep ensemble approach relies on distributional assumptions. The loss function is derived by minimising the negative log likelihood of each observation according to the Gaussian assumption (see Nix and Weigend (1994) for the derivation):

$$L(\hat{y}_i) = \frac{(y_i - \hat{y}_i)^2}{\sigma_i^2} + \frac{\log(\sigma_i^2)}{2}. \tag{19}$$

Inspecting equation (19) shows that the first-term estimates the MSE of the prediction with reference to the actual observation, scaled by the variance, thus, the network can reduce the contribution of an observation i to the loss by increasing the variance. The second term acts as a penalty on observations with large estimated variances.

An implementation of the deep ensemble loss function in R appears in Listing 7. The first function in the listing is used to derive the expected value prediction and depends on the predicted variance. This corresponds to equation (19), except for the addition of a small constant number ε to each variance term to precent division by, or taking the logarithm of, zero. The second function is used to ensure that the predicted variance of the network is unbiased by calculating the empirical variance of the observations and calculating the MSE of the empirical variance compared to the predicted variance.

6.2. Uncertainty in the Mortality Forecasting Model

The mortality forecasting model was adapted to apply each of the pinball loss and the deep ensemble approaches separately. A simple way to adapt the models for this is to modify only the last layer of the network, in other words, keep the design of the network the same as it is when trying to forecast a best estimate and modify only Line 51 in Listing 5. Listing 8 shows how this can be done for each of the pinball loss and deep ensemble methods, by adding outputs to the network. For the deep ensemble method, these outputs are the predicted mean and variance for observation i (see Lines 8–16 of the Listing), and for the pinball loss method, these outputs are the lower and upper quantiles, in addition to the mean forecast of mortality (see Lines 30–37 of the Listing).

On the other hand, one might suppose that the representation learned by the network to produce a best estimate forecast is not ideally suited to predicting uncertainty and should be modified to produce a representation that is more suitable. This could be achieved, for example, by adding layers to the network that modify the representation to be more suitable for the task of estimating the quantiles. Listing 9 contains code for an example model using the pinball loss that implements this idea. The model is the same as that presented in Listing 5 up to line 18. After this point, the

Table 17. Empirical coverage of the 2.5%–97.5% confidence interval derived using ten runs of the uncertainty prediction models on the test set. Results sorted by deviation from the targeted 5% coverage

	Description	Data Exceeding 97.5%	Data Exceeding 2.5%	Coverage	Delta
1	Pinball Loss, ReLu branches	0.022	0.026	0.048	0.002
2	Pinball Loss, tanh branches	0.021	0.026	0.048	0.002
3	Deep Ensemble	0.026	0.029	0.055	0.005
4	Deep Ensemble, ReLu branches	0.012	0.030	0.043	0.007
5	Pinball Loss	0.017	0.025	0.042	0.008
6	Deep Ensemble, tanh branches	0.014	0.027	0.041	0.009

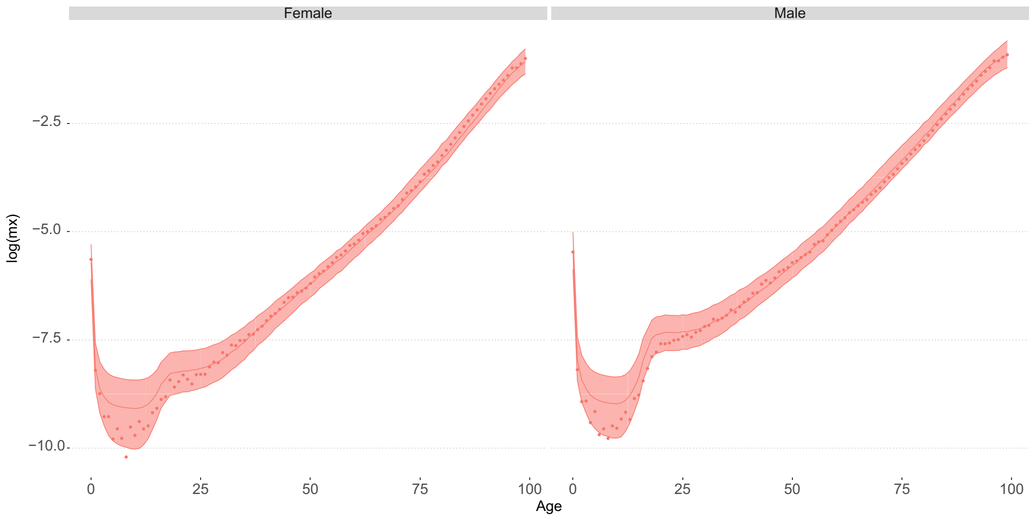

Figure 5. Confidence bands for projected UK mortality rates using the pinball loss model with ReLu branches, 2016, males and females.

model “branches” into three separate paths: one to estimate the mean outcome and two more for each of the lower and upper quantiles respectively. Similar modifications were made to the mortality forecasting model to implement the deep ensemble approach, except in this case, only two branches are needed[8]. Each model was fit ten times on the training data set and the predicted 2.5% and 97.5% quantiles on the test set were averaged over the ten runs to provide a final predicted quantile for each model type. Table 17 shows the coverage of the models on the test set for the following variations of the model: deep ensemble and pinball loss models without branches, and deep ensemble and pinball loss models with branches, where, for each model, we test two versions that use either tanh and ReLu activations.

The table shows that all variations of the models achieve relatively good performance, with the branch variations of the pinball loss model producing an empirical coverage of 4.8% and lower and upper quantiles that are both quite accurate. The next most accurate model is the deep ensemble model without branches, followed by the rest of the models. Thus, based on these results, it

[8]Model code for the pinball loss and the deep ensemble approach in R can be found in the files HMD model pinball.r and HMD model deep ensemble.r on the associated GitHub repository.

Table 18. MSE of the average prediction of ten runs of the uncertainty forecasting models on the test set (multiplied by 10^4) and number of times that the LC model is beaten, with the particular model choice being varied as noted in the description column. Results sorted by number of times each forecasting model beat the LC model

	Description	Average MSE	Best Performance over Populations
1	Pinball Loss	12.385	278
2	Pinball Loss, ReLu branches	12.481	285
3	Pinball Loss, tanh branches	12.488	289
4	Deep Ensemble, tanh branches	12.642	291
5	Deep Ensemble	12.644	294
6	Deep Ensemble, ReLu branches	12.492	297

seems that both the deep ensemble and pinball loss approaches produce reasonably accurate predictive intervals for mortality forecasting, and that adding "branching" layers to the network leads to improved uncertainty quantification when applying the pinball loss. In Figure 5, we show some example confidence bands from the pinball loss model with ReLu branches for actual and projected mortality rates in the UK in 2016. The figure shows that the confidence bands are adapted well to the shape of the underlying mortality rates, with the greatest uncertainty at the teenage and young adult years, and lower uncertainty at ages older than these. Comparing the actual mortality rates to the expected shows that, at the younger ages, mortality rates were generally lower than expected, whereas at the older ages, the outcomes were closer to the forecasts. In this case, at least, the greater uncertainty of the models at the younger ages corresponds well with the greatest deviations from the model forecasts.

Next, we consider the impact of including uncertainty predictions on the models' forecasting performance. Table 18 shows that compared to the baseline model performance of a test set MSE of 12.44 and exceeding the LC model's forecasting performance on 287 populations (see the baseline model results in Table 3), all of the uncertainty forecasting models perform quite well, with the variations of the deep ensemble models exceeding the performance of the LC model by a greater margin than the baseline models, and with the pinball loss model producing a lower average MSE score than the baseline model. Thus, some performance gains on best estimate forecasting can be achieved by augmenting best estimate forecasting models to predict measures of uncertainty.

We now consider to what extent the intervals produced by the models accord with intuition: it would be expected that countries with larger populations would have more stable mortality experience leading to narrower confidence intervals, and, similarly, that forecasts for subnational populations will have wider confidence intervals than those for national populations. Also, we would expect that confidence intervals projected far into the future should have wider confidence intervals than projections made for time periods in the short-term future. We illustrate the model performance in these cases graphically in Figure 6, which compares the performance of the pinball loss model with ReLu branches forecasting mortality rates for the USA and Iceland in 2016. It can be seen that for the USA, with a large population and thus, stable mortality rates, the confidence bands are quite narrow, whereas for Iceland, with a very small population, the bands are wider at almost every age. Similarly, Figure 7 shows that the model produces wider confidence bands for the sub-national populations of Australia compared with the national population. Finally, Figure 8 shows that the confidence bands for UK mortality in 2016 are slightly wider than those in 2001 at most ages. This is confirmed in Figure 9, which displays the size of the confidence band at each age, measured by adding together the ratio of the upper confidence band to the best estimate forecasts, and the inverse of this ratio for the lower band. It can be seen that the confidence band for

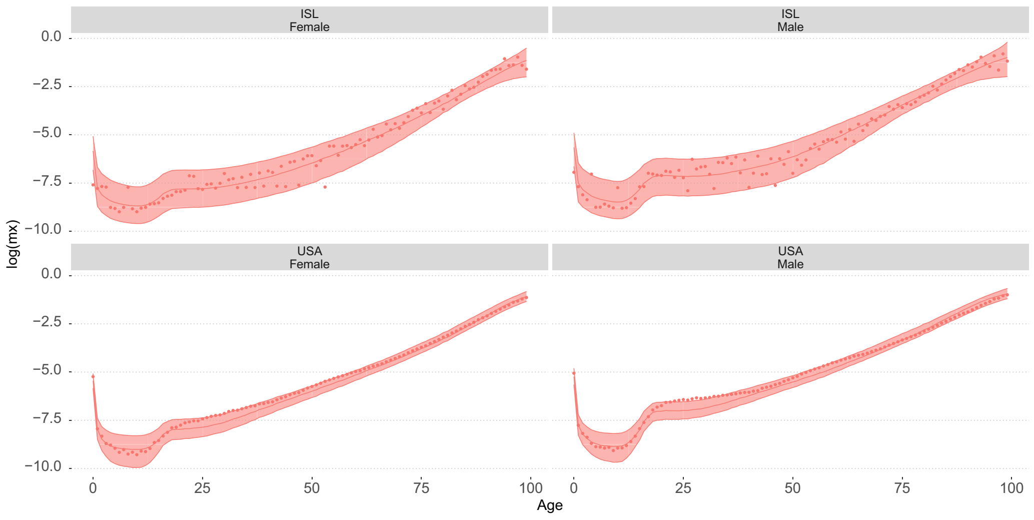

Figure 6. Confidence bands for projected mortality rates in the USA and Iceland using the pinball loss model with ReLu branches, 2016, males and females.

2016 is wider than that for 2000 at most ages, and, in particular, is wider at the younger and middle ages, whereas the bands are of similar magnitudes at the older ages. Since the younger ages exhibit substantially larger variability in mortality rates than the older ages, this behaviour is to some extent understandable. Moreover, forcing the confidence bands to expand with time at every age would produce bands that are less calibrated than the examples shown here. We return to this point at the end of this section. In summary, the results of applying the proposed methods for quantifying uncertainty generally accord well with intuition, and the models have produced confidence bands that adapt reasonably to the specifics of each case being forecast.

6.3. Summary

Two relatively simple techniques to modify deep neural networks to produce uncertainty estimates have been presented in this section. Notably, neither of these approaches require multiple simulations, as is the case with bootstrapping, and can be implemented directly within Keras, without the need for other packages. This section showed that the predictive intervals produced using the deep ensemble and pinball loss approaches are quite well calibrated empirically and slightly improve the best estimate forecasting performance of the networks. Importantly, neither of these approaches account for parameter error within the networks, meaning that it is likely the predictive intervals produced in this section are too narrow, if these would be used for setting capital requirements, among other actuarial uses. On the other hand, widening the intervals to account for parameter error would lead to worse empirical coverage than is shown in Table 18, thus, for applications where good empirical coverage of predictive intervals is necessary, these techniques should be considered.

7. Designing Networks for Explainability

In this section, we discuss how neural network models can be designed to allow for greater explainability. On the one hand, the complex functional structure of deep neural networks (see section 4.1) allows for highly accurate predictions to be made at the cost that the effect of

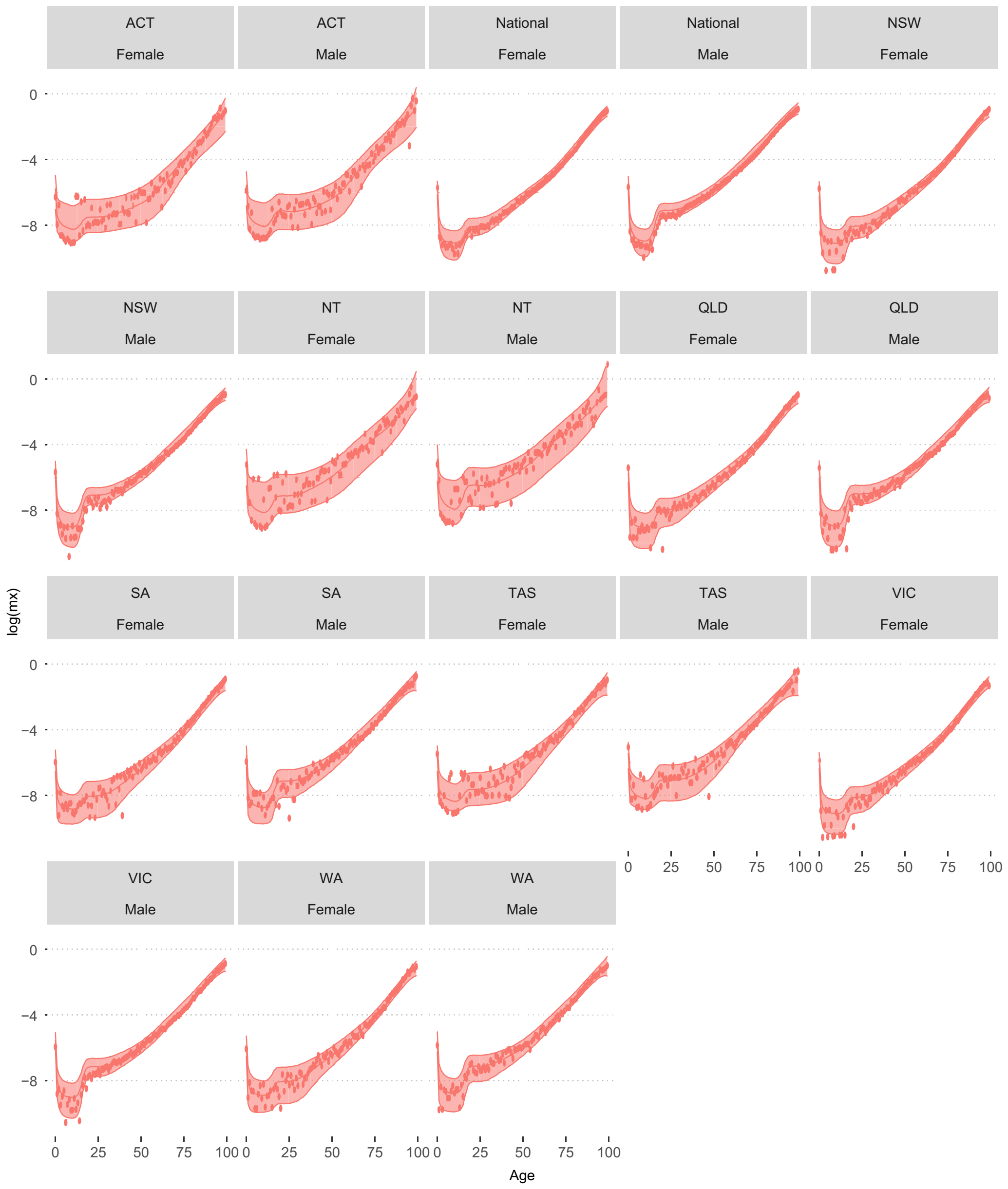

Figure 7. Confidence bands for projected mortality rates in Australia and its sub-national territories using the pinball loss model with ReLu branches, 2014, males and females.

each input variable on the predictions is difficult to quantify (mainly because feature engineering is performed implicitly by the neural network). On the other hand, for linear models and GLMs, the effect of each input variable is easy to quantify, but extensive feature engineering becomes necessary to attain good predictive performance. In this section, we discuss how deep neural networks can be designed to enable greater explainability, by which we mean to say, that the effect of the input variables on the predictions of the network can be easily quantified and understood.

This section is organised as follows. In the first sub-section, we discuss the Combined Actuarial Neural Network (CANN) proposal of Wüthrich and Merz (2019) and show how it can be

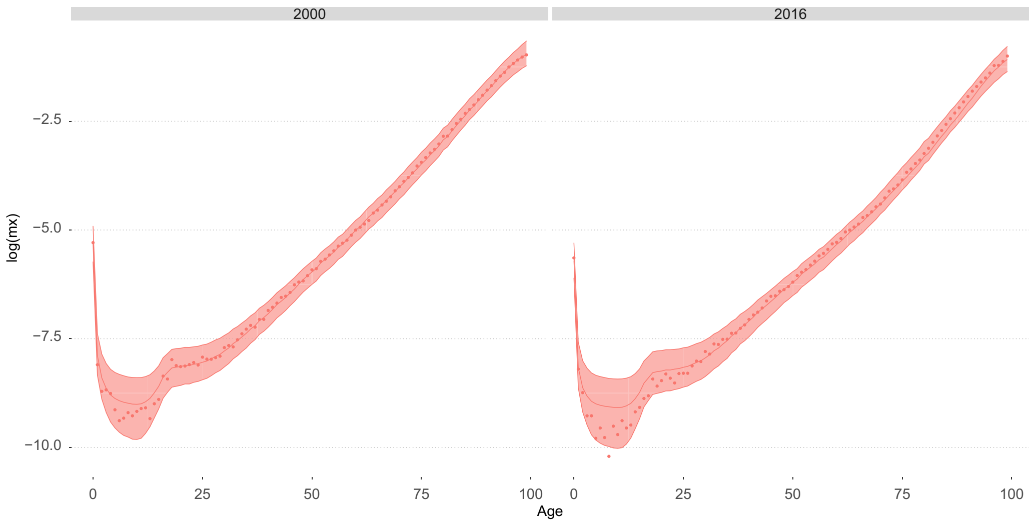

Figure 8. Confidence bands for projected mortality rates in the UK using the pinball loss model with ReLu branches, 2000 and 2016, females.

implemented for the mortality forecasting example. Using this implementation, we consider the linear and non-linear contributions of the input variables on the mortality forecasts produced with the neural networks. In the next section, we define an explainable neural network (XNN) model (Vaughan *et al*., 2018; Agarwal *et al*., 2020), extend the formulation to include embeddings, and discuss how to combine the CANN and XNN models into a single model, which we call the combined actuarial explainable neural network (CAXNN) model. Following this section, based on the insights into the model gained in these first sub-sections, we reproduce the performance of the ensemble of mortality forecasting networks using the technique of model distillation (Hinton *et al*., 2015).

7.1. Combined Actuarial Neural Network

Wüthrich and Merz (2019) describe a general approach to combining classical actuarial models, usually based on GLMs, together with neural networks, which has been applied in the context of actuarial pricing and reserving models (Gabrielli, 2019; Gabrielli *et al*., 2019; Gao *et al*., 2020; Schelldorfer & Wüthrich, 2019). The basic idea is to use a neural network to enhance the predictions made by a simple actuarial model. This is done by embedding a GLM within a neural network, such that both the GLM and neural network use the same covariates X for the prediction y. The output of the GLM component is used directly to predict the outcomes y, i.e. the GLM is directly connected to the output, whereas the neural network component first augments the covariates via representation learning to X', before predicting the outcomes. One advantage of the CANN approach is faster calibration of the combined model, if the linear component of the model is pre-calibrated using maximum likelihood estimation (MLE). Moreover, directly connecting the covariates input into a network to the output has been studied in Van Der Smagt and Hirzinger (2012), who find that this approach allows networks to be calibrated faster.

Here, we focus on a different advantage of the CANN approach, which is that predictions made using the linear component of a CANN model are easily explainable, in that the prediction is a simple linear combination of the covariates X. The magnitude of the non-linear component of the predictions can then be compared to the explainable component and the covariates X, allowing the user of the model to understand further the contribution of the non-linear component.

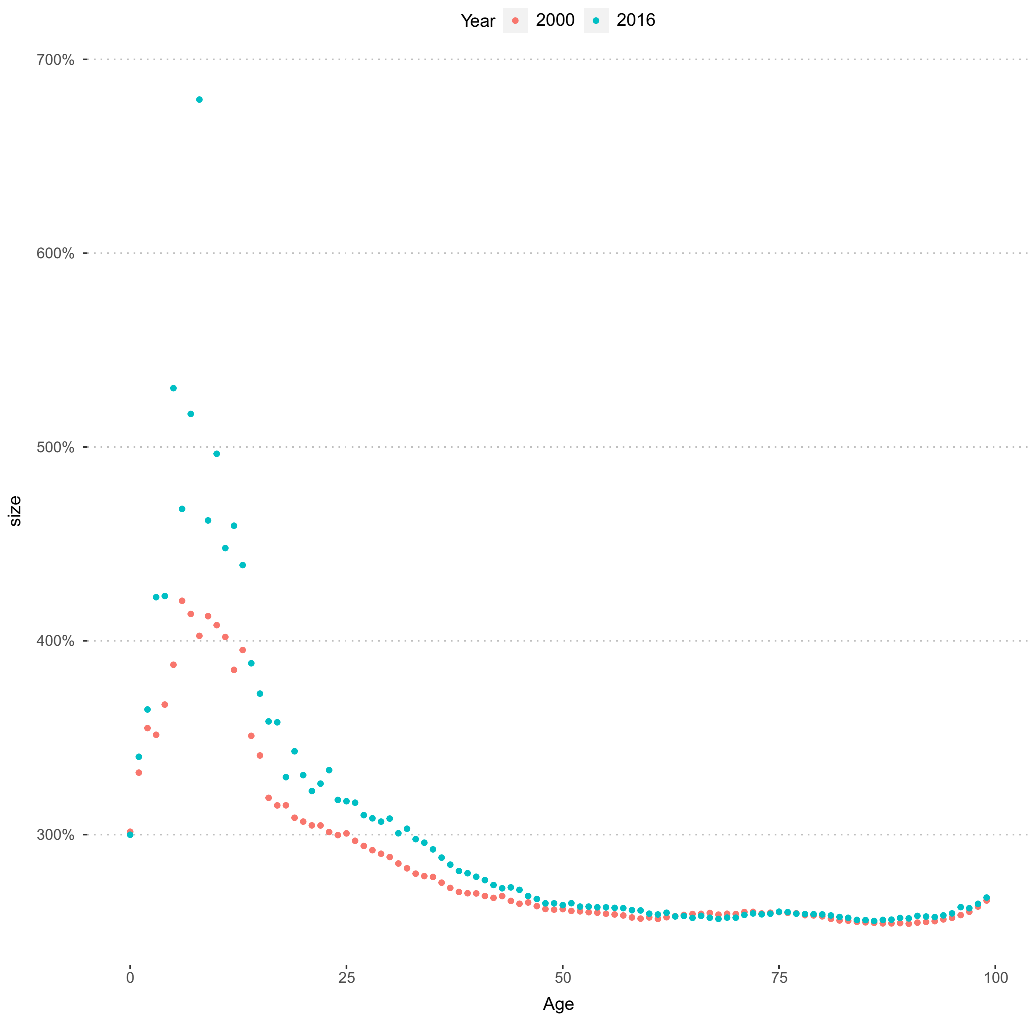

Figure 9. Size of the confidence bands for projected mortality rates in the UK using the pinball loss model with ReLu branches, 2000 and 2016, females. Size was estimated as described in the text.

We define a CANN version of the mortality model presented in section 4.1 as follows:

$$\mathbf{Z}^1 = \sigma_0(c_0 + B_0' feature_{x,t}^{p,g}) \tag{21}$$

$$\mathbf{Z}^2 = \sigma_1(c_1 + B_1' Z^1) \tag{21}$$

$$\vdots \tag{22}$$

$$\mathbf{Z}^4 = \sigma_3(c_3 + B_3' Z^3) \tag{23}$$

$$\mathbf{Z}^5 = \sigma_4(c_4 + B_4'(Z^4, feature_{x,t}^{p,g})) \tag{24}$$

$$\hat{y} = \sigma_5(c_5 + B_5' feature_{x,t}^{p,g} + B_6' Z^5). \tag{25}$$

Table 19. Test set mean squared error (multiplied by 10^4) using the LC and CANN models, national and sub-national populations

	Model	type	Average MSE	Median MSE	Best Performance
1	LC_SVD	National	5.55	2.48	10
2	LC_SVD	Sub-National	22.08	1.48	17
5	CANN	National	2.46	1.26	66
6	CANN	Sub-National	20.37	0.82	219

Comparing these equations to those presented before, it can be seen that the major change is to include the feature vector $feature_{x,t}^{p,g}$ directly in the output of the network, so that a skip connection to the covariates now links X directly to the output of the network, y. Thus, on the canonical scale (before the application of the link function σ_5), the predictions of the model are comprised of a linear component $B_5' feature_{x,t}^{p,g}$ added to the output of the neural network $B_6' Z^5$, which we refer to as the non-linear component of the CANN model in what follows. We remark that the skip connection in the neural network part of the model is maintained, see equation (25), as this leads to better performance of the model compared to the case when this is removed.

In this application of the CANN approach, we do not pre-calibrate the linear component of the model using MLE (as is suggested in Wüthrich and Merz (2019)), but rather calibrate both components of the CANN model simultaneously. Listing 10 shows the code used to fit the model. Different from the model presented previously, we now pass the time covariate through a smaller network so that the time effect on the outcome can be non-linear, even in the linear component of the network. This smaller network outputs 5 new variables, equal to the dimension of the embeddings for Age and Gender. This is shown on Lines 24–27 of the listing. We note that allowing a network focussing on only a single variable within the predictor variables X is discussed in more detail in the following parts of this section. The major change compared with the model presented earlier is on Line 58 of the listing, where the skip connection containing the feature vector appears. 10 versions of the model were fit using the same data and process as described section 4.1. The results of the model on the test set are shown in Table 19. The CANN model achieves similar performance to the DEEP model, performing slightly better on forecasts at the sub-national level, and slightly worse for forecasts at the national level.

We now proceed to explore the linear and non-linear components of the network by selecting one of the networks used to produce the forecasts and exploring the weights (regression coefficients) of the last layer. The left pane of Figure 10 shows the magnitude of these weights for the linear and non-linear components of the CANN model. The linear weights vary over a far greater range than the non-linear weights, which are confined to a small range close to zero. The right pane shows that the weights for the Age, Gender and Year embeddings are substantially larger than those for the Country and Subnational region embeddings. Thus, relatively large weight is being placed within the CANN model on each element of the linear component, and smaller weight on each component of the non-linear component (although there are more of the latter).

The effect size for each level of the Year, Gender, Age and Country variables are shown in Figure 11. It can be seen that the effects are quite easily interpretable: the values for Year show a (non-linear) decreasing trend, signifying mortality improvements with time, and the Age variable follows the shape of a life-table. Males are shown to have higher mortality than females and, finally, the country effects match intuition quite well. Thus, the linear component of the CANN model can be interpreted directly in familiar terms and could be used directly to construct an approximate life-table.

Remarks 7.1. Including the embeddings in the last layer of the network, see equation (25), has the effect of imposing a constraint within the model that a linear projection of the embeddings into a

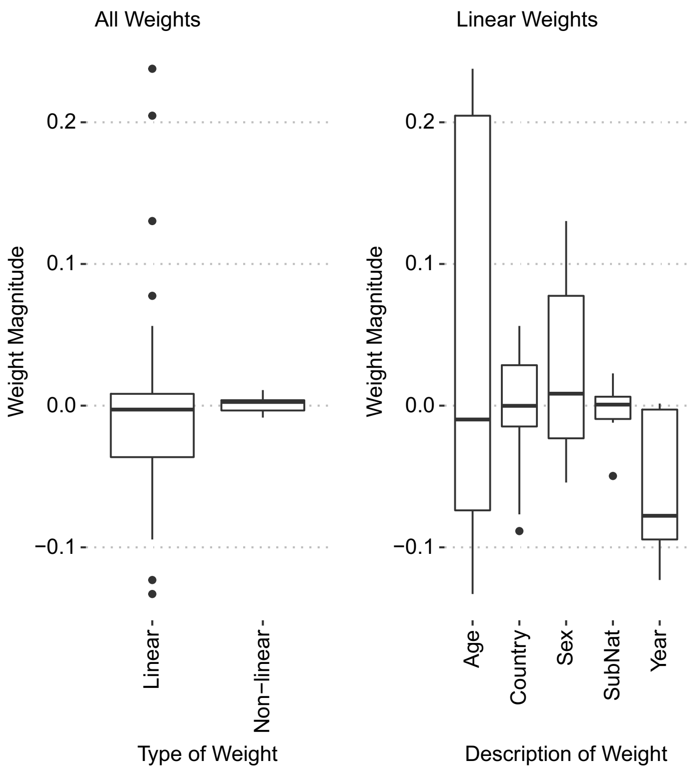

Figure 10. Magnitude of weights in last layer of CANN model for HMD data. Left pane shows weights for the linear and non-linear parts of the network and right pane shows only weights for the non-linear component.

single dimension must have a direct interpretation in terms of the output of the neural network. This linear projection is learned by the network, see the linear coefficients B_6 in 25, and is shown in Figure 11. Whereas a principal component analysis has been used to explain learned embeddings in a relative manner, see, for example Perla *et al.* (2020), the approach shown here does not require PCA, and, moreover, allows each level of the embedding to be interpreted in absolute terms.

We now turn to the non-linear component of the CANN model. Figure 12 compares the non-linear component of the CANN model to the linear component for a 5% sample of predictions on the test set. For middle-aged and older age mortality, the non-linear component is quite small relative to the linear component, whereas for child, young adult and oldest ages, the non-linear component is much more significant. Thus, we can explain the workings of the CANN model that the majority of the corrections made to the linear part of the model relate to accurately forecasting mortality for these ages, whereas for middle- and older- aged mortality, a linear model performs quite well.

A more detailed view of these non-linear corrections is shown in Figure 13, which shows these separately for different countries and sub-national regions, for each gender, over time. Immediately observable is that the extent of the non-linear component varies with all of the factors within the model, in other words, it varies in quite a complex manner by Country, Sub-national region, Year and Gender. Thus, the non-linear component can be understood in terms of interaction effects between these inputs to the CANN model that are implicitly learned by the neural network and added to the linear output to produce predictions from the model.

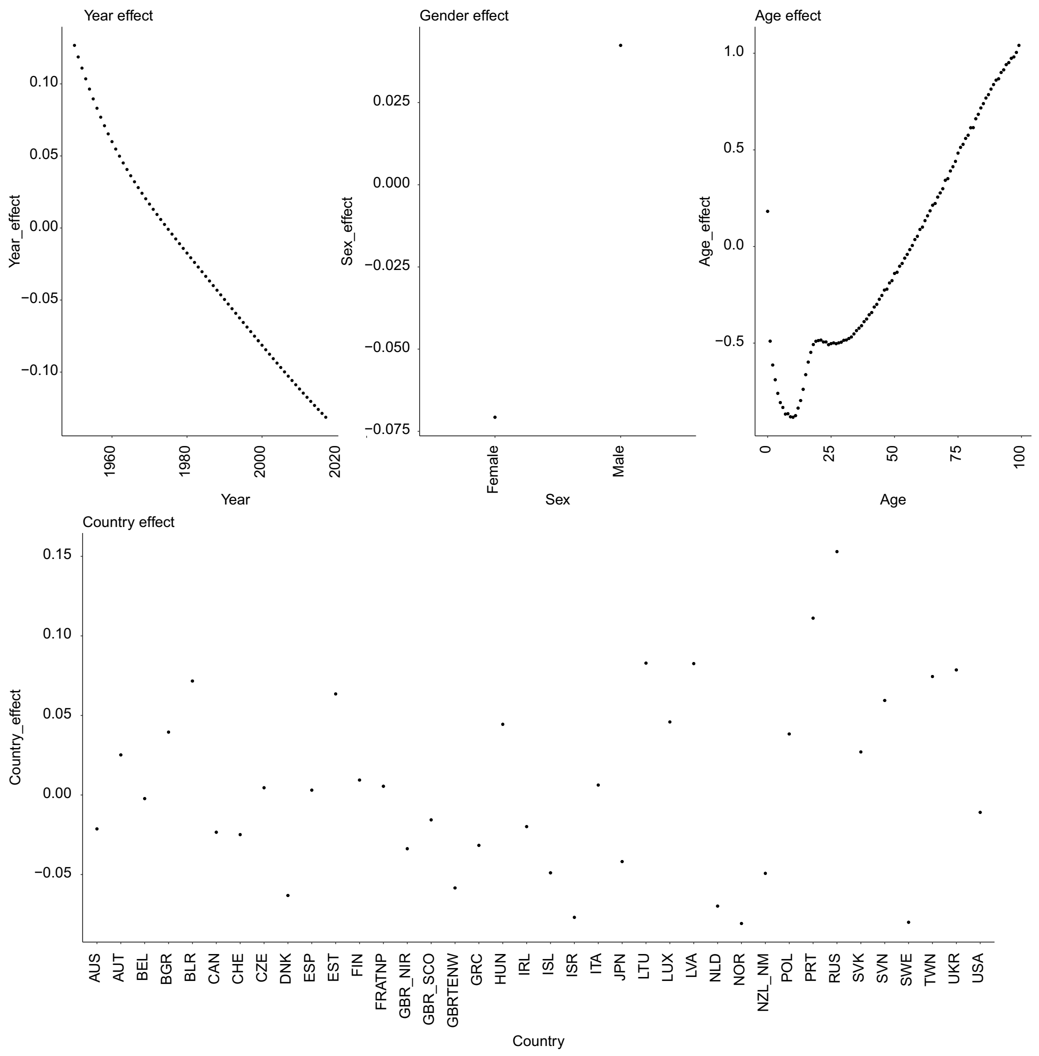

Figure 11. Effect sizes for each level of the Year (top lhs), Gender (top middle), Age (top rhs) and Country variables (bottom), derived using the CANN model.

Notably, however, the exploration of the linear and non-linear components of the network has been performed for only a single example training run, whereas to produce predictions using this network, an average prediction from many networks would be used. Of course, the predictions from each of these networks could be inspected but besides for being a tedious exercise, each network might produce predictions using a different representation, leading to difficulties in gaining an understanding the average result. For this reason, in what follows, we fit a new surrogate neural network model to reproduce the average predictions of the networks trained in this section. In the deep learning literature, this is called model distillation (Hinton *et al.*, 2015), and has been applied in Richman and Wüthrich (2020) in the context of non-life pricing. The non-linear component of the single surrogate model is designed to be explainable and is based on the insights into the non-linear component of the network gained in this section, thus leading to a fully explainable model, while taking advantage of representation learning to specify the exact form of the model.

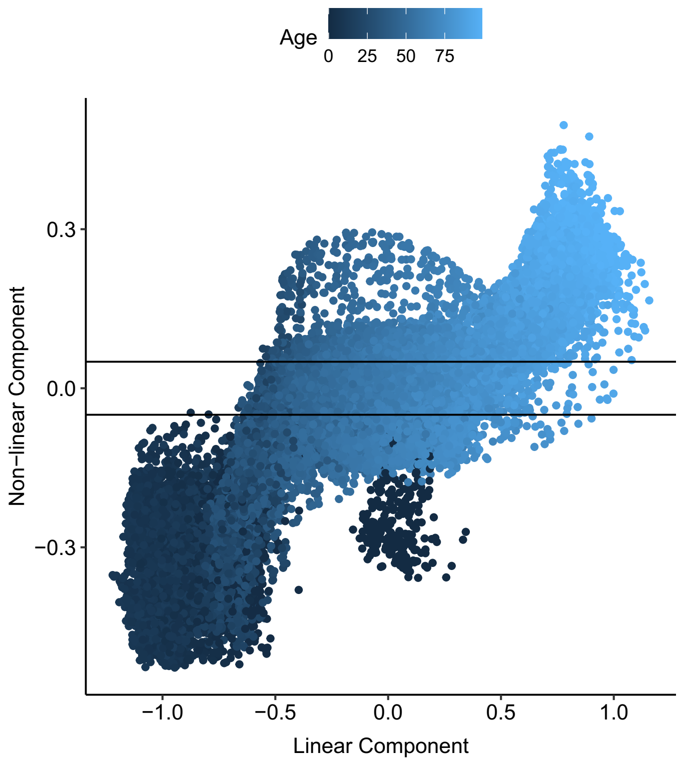

Figure 12. Comparison of linear and non-linear effects from the CANN model, coloured according to age. Horizontal lines indicate a non-linear effect of 0.05 and −0.05, respectively. 5% sample of the test set predictions.

7.2. Explainable Neural Networks

The non-linear component fit the CANN model was shown in the previous section to depend in a complex manner on all of the inputs to the neural network, which can be understood as the interaction effects between these variables. However, it is still difficult to isolate and explain the contribution of each interaction effect, which, as shown in Figure 12 can modify the predictions of the linear component of the model significantly. On the one hand, if the non-linear effect could be decomposed into the interaction effects of particular variables, then the explainability of the model would be increased. On the other hand, the exact form of these interaction effects appears to be quite difficult to specify explicitly, thus building a traditional model to reproduce these does not appear viable. Here we focus on solving this problem by using neural networks to model each potential interaction on a stand-alone basis, in other words, using representation learning to specify the interactions implicitly but constraining the input to each network so that the output of these networks is explainable as a combination of only a limited number of variables.

Two quite similar proposals have appeared in the deep learning literature in Vaughan *et al.* (2018) and Agarwal *et al.* (2020), who, respectively, refer to these types of models as Explainable Neural Networks (XNNs) or Neural Additive Models (NAMs); in what follows we focus on the former since it appeared earlier in the literature. XNNs are a type of additive index model defined as

$$\hat{y}_i = \mu + \gamma_1 f_1(x_{i,1}) + \gamma_2 f_2(x_{i,2}) + \ldots + \gamma_P f_P(x_{i,P}), \tag{26}$$

where f_p, $1 \le p \le P$, are functions of individual covariates $x_{i,p}$ which are learned using neural networks, μ and γ_p, are the coefficients of the final layer of the model, and P is the number of covariates

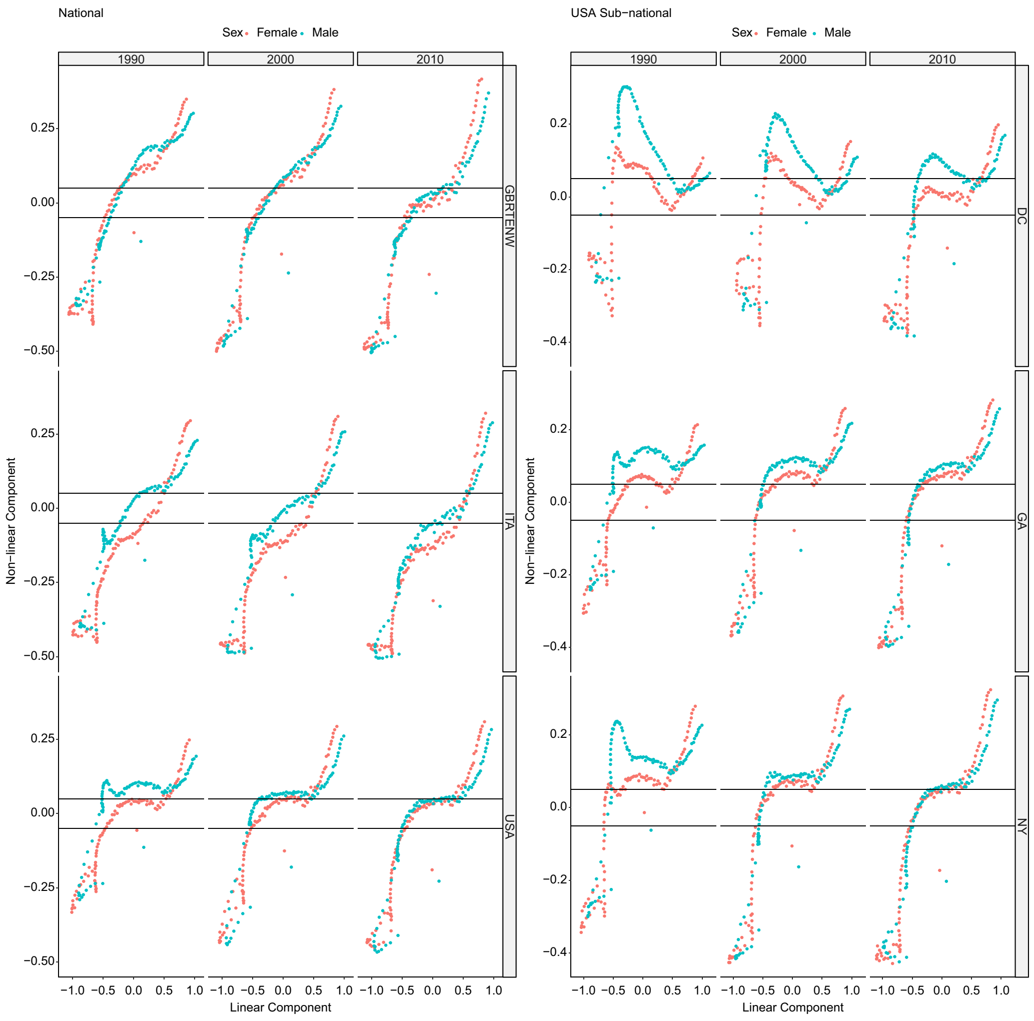

Figure 13. Comparison of linear and non-linear effects from the CANN model, coloured according to age. Horizontal lines indicate a non-linear effect of 0.05 and −0.05, respectively. 5% sample of the test set predictions.

entering the model. Comparing 7.7 to 3.4, the effect of each covariate that enters the model on the prediction can be isolated in the XNN, whereas this cannot be done in the traditional neural network. Both Vaughan *et al.* (2018) and Agarwal *et al.* (2020) discuss extensions to these models that could result in improved predictive performance in some cases, but are not necessary for the application that follows.

XNNs can be adapted for use in the mortality forecasting example by extending these models to incorporate categorical variables and to allow for interactions between different variables. The first of these adaptations consists simply of using embedding layers for each categorical variable that are input into the neural networks that define each function $f_p(.)$. The second adaption consists of passing multiple variables into each function $f_p(.)$, where these interaction effects need to be chosen using judgement. To encourage the network to learn interaction effects, we use a simple "crossing" layer (see Wang *et al.* (2017) for an extended discussion and application of a similar idea), which can we formulate as

$$\mathbf{Z}^1 = \sigma_0(c_0 + B_0'X') \tag{27}$$

$$\mathbf{Z}^2 = e^{\mathbf{Z}^1}, \tag{28}$$

where X' is the vector containing the embeddings for each categorical variable entering the network and the exponential operator in equation (28) is applied element-wise to the output of the neural network $\mathbf{Z}^1$. The idea of the crossing layer is to learn multiplicative representations of the input vector X'. We found that adding this crossing layer significantly improves the predictive performance of the XNN.

In the next section, will illustrate this process for mortality forecasting model.

7.3. The Combined Actuarial Explainable Neural Network

In this section, we build on the previous CANN decomposition of the output of a deep network into a linear and non-linear component, by modelling the non-linear component of the output using an XNN to derive explicitly the interaction effects among the predictor variables X. We call this combined model a Combined Actuarial Explainable Neural Network (CAXNN). The linear component of the model is specified in the same way as shown in equation (25). For the non-linear component, we consider the analysis in section 7.1 which showed that the non-linear component of the network varies in a complex manner depending on all of the inputs to the CANN network. To allow for these non-linear effects, we specify a XNN which models several of the second- and third- order interactions of the inputs to the network in an explicit manner. Since the effects of each interaction can be considered in isolation, the CAXNN model is more explainable than the CANN model considered earlier.

The CAXNN model is used here as a surrogate model to explain the average predictions of the 10 CANN networks fit in section 7.1 by "distilling" the predictions of these networks using a different model. Notably, we fit the CAXNN model on the average predictions of the 10 CANN models on both the training set (years less than or equal to 1999) and testing set (years greater than 1999), since the predictions $\hat{y}$ on the test set are available from the model even if the outcomes y are not.

The CAXNN model is fit in two steps. First, a neural network consisting only of the linear component of the CAXNN model is calibrated; this part of the model is structured in exactly the same manner as the linear component of the CANN model discussed in section 7.1. The code for this model is shown in Listing 11, which defines a smaller subnetwork for the time variable and embeddings for each categorical variable. Table 20 shows the results of this model on the test set. Compared to the results of the CANN model shown in Table 19, this linear model performs surprisingly well, achieving substantially better performance than the LC model for sub-national populations. This indicates that, to some extent, a linear distillation to explain the predictions of an ensemble of neural networks may be sufficient.

Remarks 7.2. Whereas the CAXNN model shown here relies on calibrating a stand-alone linear model, other sources for the linear component of the model might be considered. For example, the embeddings from the CANN model discussed in section 7.1 could be used. Although we do not show these results here in detail, experiments using embeddings from a single CANN model produced a CAXNN model that exceeded the performance shown in Table 21.

In a second step, the rest of the CAXNN model, i.e. the non-linear component is calibrated. To do this, the weights of the linear component of the CAXNN model are set equal to those derived in the previous step where model with only a linear model was calibrated. These weights are held fixed at these values throughout the second step of training the CAXNN network, meaning to say, that only the weights of the non-linear component are calibrated. The code for setting these weights of the linear component of the CAXNN model is shown in Listing 12.

Table 20. Test set mean squared error (multiplied by 10^4) using the LC and linear component of the CAXNN model, national and sub-national populations

	Model	type	Average MSE	Median MSE	Best Performance
1	LC_SVD	National	5.55	2.48	35
2	LC_SVD	Sub-National	22.08	1.48	43
3	Linear	National	4.28	3.07	41
4	Linear	Sub-National	20.94	0.95	193

Table 21. Test set mean squared error (multiplied by 10^4) using the LC and full CAXNN model, national and sub-national populations

	Model	type	Average MSE	Median MSE	Best Performance
1	LC_SVD	National	5.55	2.48	9
2	LC_SVD	Sub-National	22.08	1.48	21
3	DEEP	National	2.67	1.46	67
4	DEEP	Sub-National	20.60	0.90	215

The non-linear component is based on the XNN introduced in section 7.2 and consists of a simple three layer neural network, which operates by first attempting to learn interactions using a "crossing" layer (see equation (28)) and then passing the input data and the output of the "crossing" layer through two more fully connected layers, the second of which is a skip connection. The function used for this purpose is shown in Listing 13.

Finally, the rest of the CAXNN network is shown in Listing 14. A number of different interaction effects between the predictor variables comprising X are manually specified, based on the insights from section 7.1. Finally, the linear and non-linear components of the CAXNN model are combined, see Lines 54–55 of the listing.

The results of fitting the CAXNN network are shown in Table 21. Comparing these to the linear component of the model, we see that performance has improved for both national and sub-national forecasts, particularly for the latter. Quite remarkably, the performance of a single CAXNN network is only slightly worse than the average prediction of the 10 basic networks shown in Table 2 and the average performance of the 10 CANN networks shown in Table 19.

Figure 14 shows the linear component of the CAXNN model, which are comparable to those in Figure 11, and are, likewise, quite easily interpretable in an intuitive manner. To compare the relative magnitude of the contribution of the linear and non-linear components of the CAXNN model, a sample of 15% of the effect sizes in the training and testing sets was taken, and a box-plot of these effects estimated. This is shown in Figure 15. By far, the effects with largest magnitude are the age effects in the linear component of the network, which is expected, since mortality rates vary by several orders of magnitude with age in modern life-tables. Similarly, the other linear effect sizes (for Gender, Year and Country) are significant. It can be seen that the interaction effects are of a similar magnitude, and that the interactions between the year variable and the other categorical variables have the largest impact on the predictions.

Next, we consider how the interaction effects might be interpreted. An intuitive example is the effect size for the interaction between Year, Age and Gender shown in Figure 16. The CAXNN model makes relatively large adjustments to the linear model for younger females and middle-aged males, as well as at the older ages for both genders. The figure shows that the size of these adjustments varies with year, indicating that, to enhance predictive accuracy, the CAXNN model

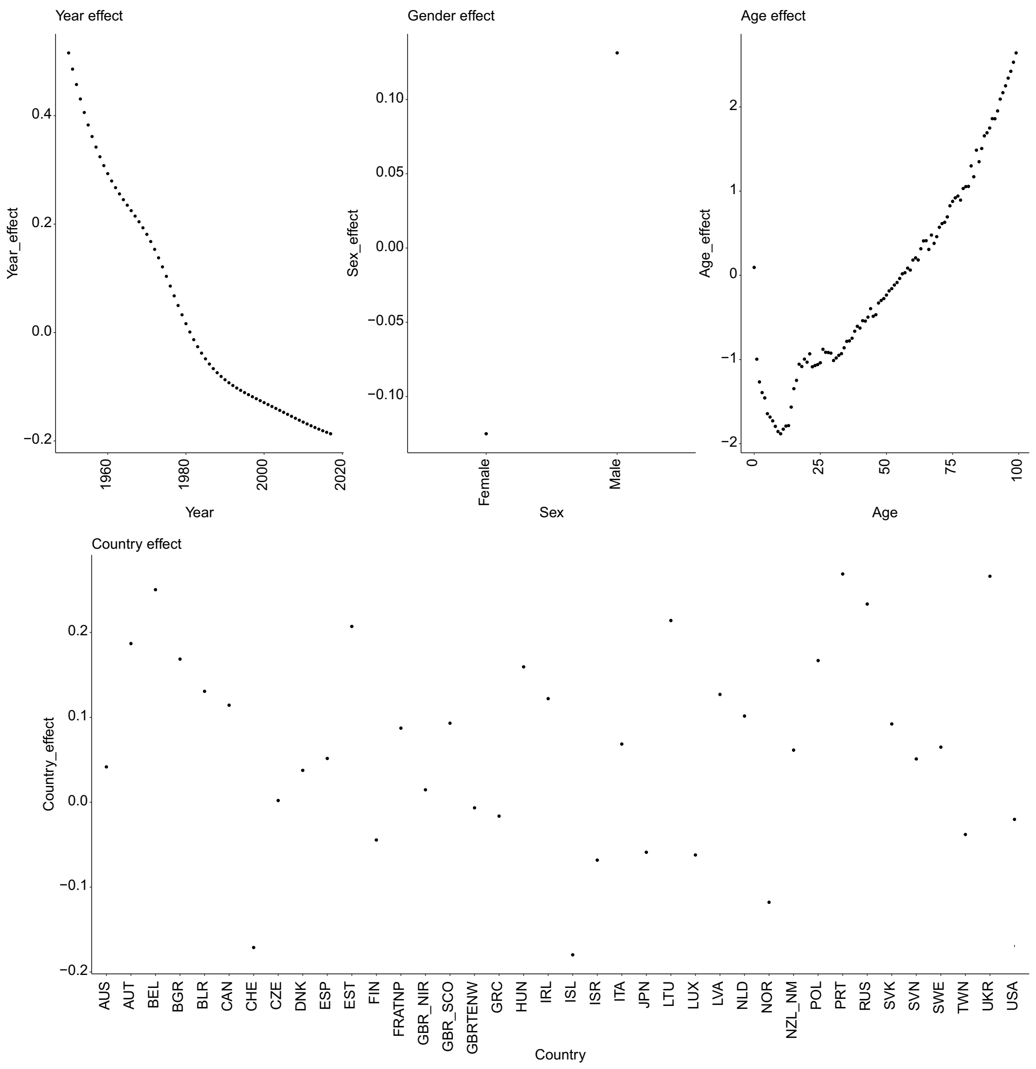

Figure 14. Effect sizes for each level of the Year (top lhs), Gender (top middle), Age (top rhs) and Country variables (bottom), derived using the CAXNN model.

allows for different rates of mortality improvement for each age and gender. Similarly, Figure 17 shows the interaction effects between Country and Year, showing that the model has learned a different baseline level of mortality for each country that then improves at a rate of mortality improvement specific to that Country.

We show the quality of the CAXNN approximation to the ensemble of CANN models in Figure 18, for two examples: Males in the United States and Males in the Tokyo region of Japan, both in 2010. It can be seen that the CAXNN model closely approximates the average prediction of the CANN models at almost every age, with the largest deviations of the approximation at the young adult ages around 23–25, which themselves are quite insignificant. Thus, the CAXNN model has successfully approximated the ensemble predictions of the CANN models.

Finally, in Figure 19 we consider the key drivers of the difference in mortality predictions between Males and Females aged 50 in the United Kingdom and Georgia, United States in the

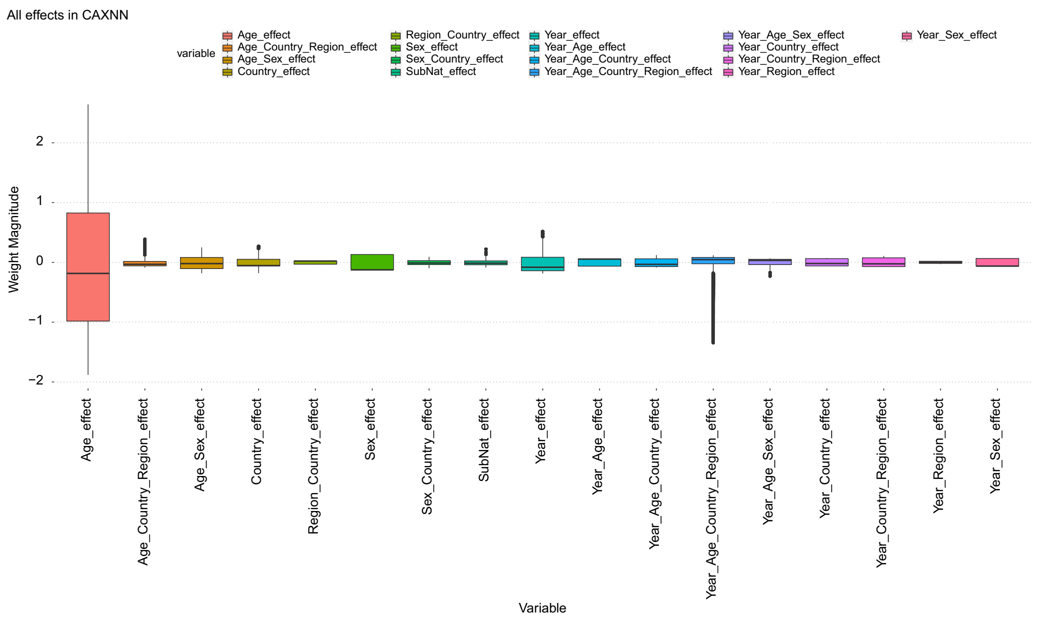

Figure 15. Effect sizes from the CAXNN model for a 15% sample of the training and testing data, all effects included.

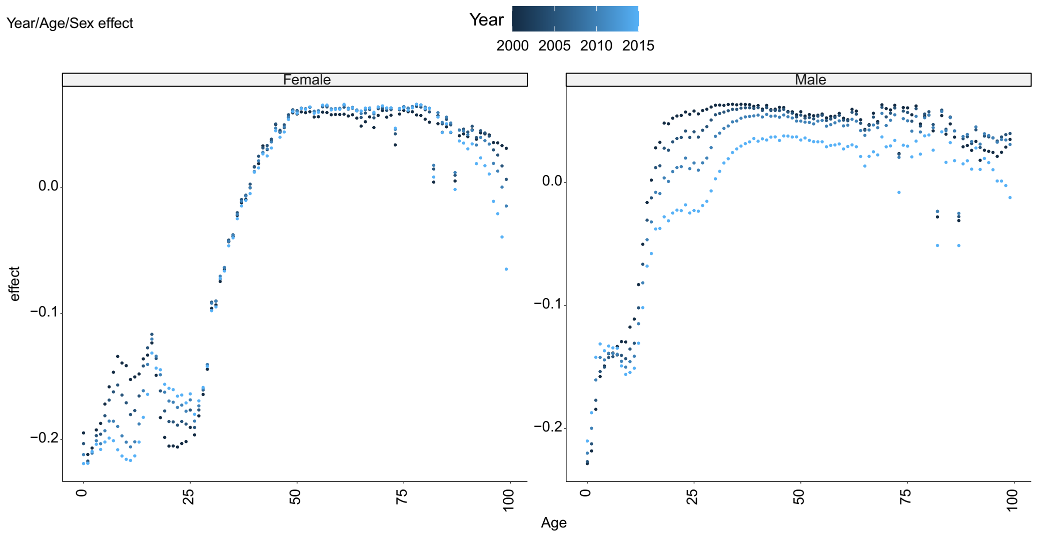

Figure 16. Interaction effect between Year, Age and Gender in the CAXNN model.

year 2015. Several characteristics are shared across all of these predictions, which are the effect of the age and year variables. The plots illustrate that, for these predictions, the country and region variables, as well as the interactions of these variables with the other variables in the model, are familiarly responsible for the different mortality predictions made by the model. Among the most important of these are the effect of the sub-national region, which leads to higher predictions for the state of Georgia and the interaction between the year of the forecasts, and the country and

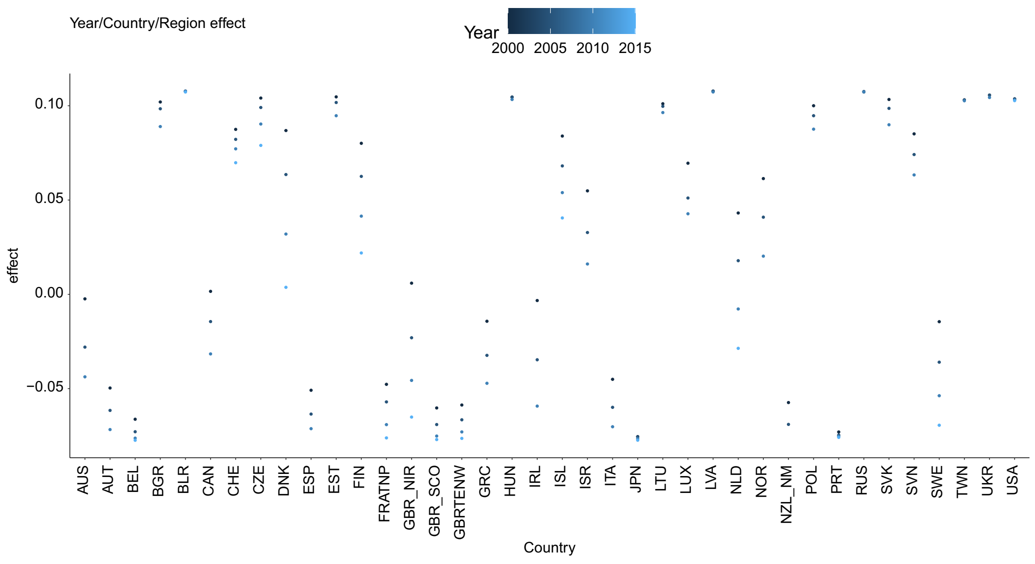

Figure 17. Interaction effect between Year and Country in the CAXNN model.

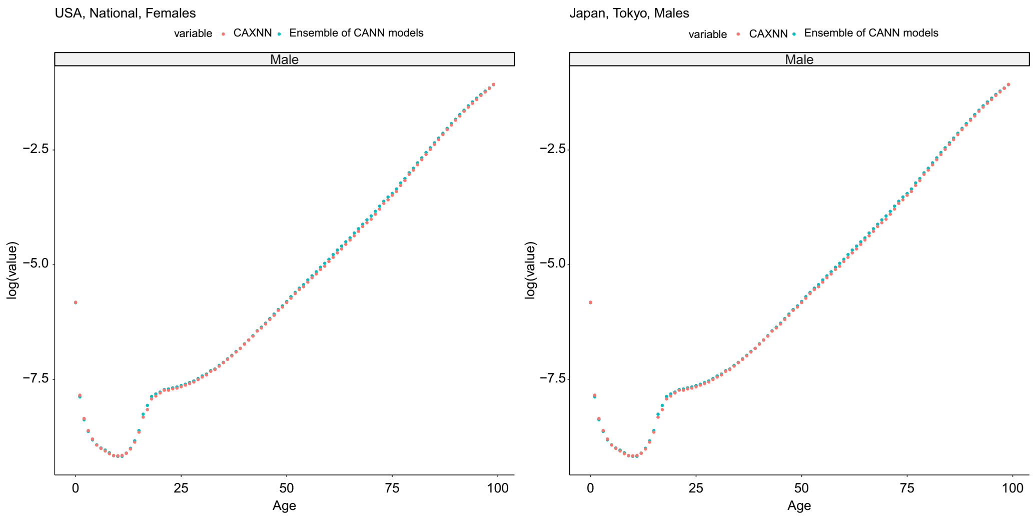

Figure 18. Comparison of the CAXNN approximation to the average prediction from the CANN models for Females in the United States and for Males in the Tokyo region of Japan, in 2010.

region for which these are being made: in the United Kingdom, this effect is quite negative, contributing to a lower mortality prediction, whereas for the State of Georgia in the United States, this is quite positive, leading to higher mortality predictions.

7.4. Summary

Two model designs that enable greater interpretability of neural network predictions were presented in this section. The CANN model predictions can be decomposed into linear and

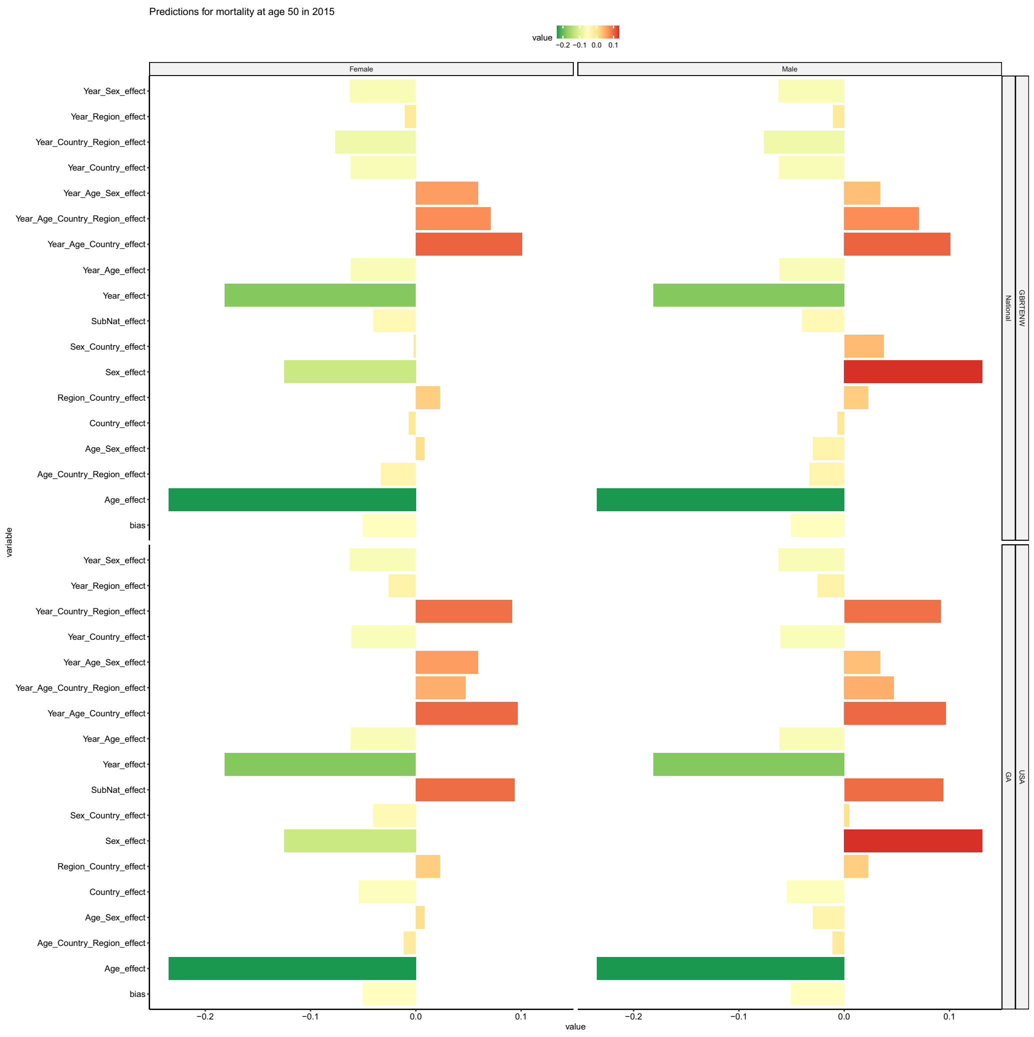

Figure 19. Breakdown of the contributions to the mortality predictions for Males and Females aged 50 in the year 2015, in the United Kingdom, and Georgia, United States.

non-linear parts. The former are easily explainable when effect sizes are considered, whereas the latter are quite amenable to analysis in a graphical manner. The CAXNN model builds upon this to enable easier interpretation of the non-linear effects and was shown to be quite predictive when used to distil the average predictions arising from 10 different CANN models. Since the CAXNN model essentially produces a linear model in the last layer of the network, the effect of each component of the predictor variables on the predictions of the model can easily be isolated and investigated.

8. Conclusions

This research has investigated three aspects of neural networks that may impact their adoption into the toolkit of actuaries: the variability of results, producing uncertainty intervals and designing these models for explainability. A relatively complex mortality forecasting example was used throughout to investigate these issues.

In examining the variability of results produced with neural network models, it was shown that the accuracy of ensemble predictions of networks suffers if the underlying networks have too little or too much variability. The analysis presented here focussed on modifying single aspects of the design and fitting of neural network models; future research could consider a more complex investigation where more than one aspect is modified at a time.

Two methods were used to extend the neural network model to quantify the uncertainty of the predictions using predictive intervals: deep ensembles and networks trained using the pinball loss. Similar to the results of applying these methods in other contexts, both of these methods produced empirically well-calibrated predictive intervals as measured on unseen data. It was observed that if parameter error was to be included within this type of analysis, then the intervals would likely be less well calibrated. Future research should consider how to incorporate parameter risk appropriately into the predictive intervals produced by neural networks.

To enhance the explainability of neural network predictions, the example mortality forecasting model was successively modified to produce predictions based on combinations of linear and relatively low order interaction effects. This model was shown to be highly accurate in reproducing the average predictions of an ensemble of models, with the output of the model being directly amenable to interpretation. Some subjective judgement was utilised to specify the interaction effects of the non-linear component of the model. Future work could consider how best to automate the selection of these interaction effects, perhaps using post-hoc model interpretability methods. Another useful extension would be to apply an explainable model to produce uncertainty metrics, thus leading to more insight into the role of each variable in producing predictive intervals.

The investigations within this research should be applicable to other types of actuarial modelling, for example, non-life pricing models. As more complex neural networks, such as recurrent or convolutional networks, begin to be applied for actuarial modelling tasks, future research should also address how these models can best be modified for the safe use by actuaries.

Acknowledgements. The author thanks the AFIR-ERM section of the International Actuarial Association for a grant which supported this research and the reviewers of the paper for their comments which helped to improve the manuscript.

References

Agarwal, R., Frosst, N., Zhang, X., Caruana, R. & Hinton, G. E. (2020). Neural additive models: interpretable machine learning with neural nets. arXiv.

Allaire, J. J. & Chollet, F. (2017). *R Interface to "Kerns".* RStudio, Google.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends® in Machine Learning*, **2**(1), 1–127.

Bengio, Y., Courville, A. & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **35**(8), 1798–1828.

Bengio, Y., Ducharme, R., Vincent, P. & Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, **3**(6), 1137–1155.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, **24**(2), 123–140.

Breiman, L. (2001). Random forests. *Machine Learning*, **45**(1), 5–32.

Cairns, A. J., Kallestrup-Lamb, M., Rosenskjold, C., Blake, D. & Dowd, K. (2019). Modelling socio-economic differences in mortality using a new affluence index. *ASTIN Bulletin*, **49**(3), 555–590.

Clevert, D. A., Unterthiner, T. & Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (ELUs), in *4th International Conference on Learning Representations, ICLR 2016 – Conference Track Proceedings.*

Danilova, I., Mesle, F., Jdanov, D., Pechholdova, M., Jasilionis, D., Shkolnikov, V. M. & Vallin, J. (2020). *HCD The Human Cause-of-Death Database.*

De Felice, M. & Moriconi, F. (2019). Claim watching and individual claims reserving using classification and regression trees. *Risks*, **7**(4), 102.

Delong, L., Lindholm, M. & Wüthrich, M. V. (2020). Collective reserving using individual claims data. SSRN Electronic Journal, 2022(1), 1–28.

Deprez, P., Shevchenko, P. V. & Wüthrich, M. V. (2017). Machine learning techniques for mortality modeling. *European Actuarial Journal*, **7**(2), 337–352.

England, P. (2002). Addendum to "Analytic and bootstrap estimates of prediction errors in claims reserving". *Insurance: Mathematics and Economics*, **31**(3), 461–466.

England, P. & Verrall, R. J. (2002). Stochastic claims reserving in general insurance. *British Actuarial Journal*, **8**(03), 443–518.

England, P., Verrall, R. J. & Wüthrich, M. V. (2019). On the lifetime and one-year views of reserve risk, with application to IFRS 17 and Solvency II risk margins. *Insurance: Mathematics and Economics*, **85**, 74–88.

Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, **29**(5), 1189–1232.

Friedman, J., Hastie, T. & Tibshirani, R. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (4th ed.), Vol. **27**(2). New York: Springer New York.

Gabrielli, A. (2019). A neural network boosted double over-dispersed poisson claims reserving model. SSRN Electronic Journal.

Gabrielli, A., Richman, R. & Wüthrich, M. V. (2019). Neural network embedding of the over-dispersed Poisson reserving model. Scandinavian Actuarial Journal.

Gabrielli, A. & Wüthrich, M. (2018). An individual claims history simulation machine. *Risks*, **6**(2), 29.

Gabrielli, A. & Wüthrich, M. V. (2019). Back-testing the chain-ladder method. *Annals of Actuarial Science*, **13**(2), 334–359.

Gao, G., Meng, S. & Wüthrich, M. V. (2019). Claims frequency modeling using telematics car driving data. *Scandinavian Actuarial Journal*, **2019**(2), 143–162.

Gao, G., Wang, H. & Wüthrich, M. V. (2020). Boosting poisson regression models with telematics car driving data. SSRN.

Gao, G. & Wüthrich, M. V. (2018). Feature extraction from telematics car driving heatmaps. *European Actuarial Journal*, **8**(2), 383–406.

Gesmann, M., Murphy, D., Zhang, W., Carrato, A., Crupi, G. & Wüthrich, M. V. (2020). *Statistical Methods and Models for Claims Reserving in General Insurance [R package ChainLadder version 0.2.11]. Comprehensive R Archive* Network (CRAN).

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

Guo, C. & Berkhahn, F. (2016). Entity embeddings of categorical variables. arXiv, *arXiv:1604*.

Hainaut, D. (2018). A neural-network analyzer for mortality forecast. *ASTIN Bulletin*, **48**(2), 481–508.

He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep residual learning for image recognition, in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, December 2016, pp. 770–778.

Hinton, G., Vinyals, O. & Dean, J. (2015). *Distilling the Knowledge in a Neural* Network, 1–9.

Howard, J. & Gugger, S. (2020). Fastai: A layered API for deep learning. *Information*, **11**(2), 108.

Huang, G., Liu, Z., Van Der Maaten, L. & Weinberger, K. Q. (2017). Densely connected convolutional networks, in *Proceedings – 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, January 2017, pp. 2261–2269.

Human Mortality Database. (2020). University of California, Berkeley (USA), and Max Planck Institute for Demographic Research (Germany).

Ioffe, S. & Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift, in *32nd International Conference on Machine Learning, ICML 2015*, vol. 1, pp. 448–456.

Ishii, F., Hayashi, R., Izumida, N., Yamamoto, K., Beppu, M. & Korekawa, Y. (2020). The Japanese Mortality Database — National Institute of Population and Social Security Research.

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

Kuo, K. (2019). DeepTriangle: A deep learning approach to loss reserving. *Risks*, **7**(3), 97.

Lakshminarayanan, B., Pritzel, A. & Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles, in *Advances in Neural Information Processing Systems*, December 2017, pp. 6403–6414.

LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep learning. *Nature*, **521**(7553), 436–444.

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

Levantesi, S., Nigri, A. & Piscopo, G. (2022). Clustering-based simultaneous forecasting of life expectancy time series through long-short term memory neural networks. *International Journal of Approximate Reasoning*, **140**, 282–297.

Loshchilov, I. & Hutter, F. (2019). SGDR: Stochastic gradient descent with warm restarts, in *5th International Conference on Learning Representations, ICLR 2017 – Conference Track Proceedings*.

Luo, P., Wang, X., Shao, W. & Peng, Z. (2018). Towards understanding regularization in batch normalization, In *7th International Conference on Learning Representations, ICLR 2019.* International Conference on Learning Representations, ICLR.

Mack, T. (1993). Distribution-free Calculation of the standard error of chain ladder reserve estimates. *ASTIN Bulletin*, **23**(2), 213–225.

Makridakis, S., Spiliotis, E. & Assimakopoulos, V. (2018). The M4 Competition: results, findings, conclusion and way forward. *International Journal of Forecasting*, **34**(4), 802808.

Meinshausen, N. (2006). Quantile regression forests. *Journal of Machine Learning Research*, **7**, 983–999.

Mukhoti, J., Kirsch, A., van Amersfoort, J., Torr, P. H. S. & Gal, Y. (2021). Deterministic neural networks with appropriate inductive biases capture epistemic and aleatoric uncertainty. arXiv.

Nigri, A., Levantesi, S. & Marino, M. (2021). Life expectancy and lifespan disparity forecasting: a long short-term memory approach. *Scandinavian Actuarial Journal*, **2021**(2), 110–133.

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

Nix, D. A. & Weigend, A. S. (1994). Estimating the mean and variance of the target probability distribution, In IEEE *International Conference on Neural Networks – Conference Proceedings*, vol. 1, pp. 55–60. IEEE.

Noll, A., Salzmann, R. & Wüthrich, M. V. (2018). Case study: French motor third-party liability claims. *SSRN Electronic Journal*, **2018**(17 June).

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., … Snoek, J. (2019). Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift.

Payeur, F. F., Chouinard, P., Bourbeau, R. & Ouellette, N. (2020). *CHMD Canadian Human Mortality Database.*

Perla, F., Richman, R., Scognamiglio, S. & Wüthrich, M. V. (2020). Time-series forecasting of mortality rates using deep learning. SSRN Electronic Journal.

Renshaw, A. & Verrall, R. (1998). A stochastic model underlying the chain-ladder technique. *British Actuarial Journal*, **4**(4), 903–923.

Richman, R. (2018). AI in actuarial science. SSRN Electronic Journal.

Richman, R., von Rummell, N. & Wüthrich, M. V. (2019). Believing the bot – Model risk in the era of deep learning. SSRN Electronic Journal.

Richman, R. & Wüthrich, M. V. (2019). A neural network extension of the Lee-Carter model to multiple populations. *Annals of Actuarial Science*, **15**(2), 346–366.

Richman, R. & Wüthrich, M. V. (2020). Nagging Predictors. *Risks*, **8**(3), 83.

Romo, V. C. (2020). Australian Human Mortality Database — ANU School of Demography.

Schelldorfer, J. & Wüthrich, M. V. (2019). Nesting Classical actuarial models into neural networks. SSRN Electronic Journal.

Schnürch, S. & Korn, R. (2022). Point and Interval Forecasts of Death Rates Using Neural Networks. *ASTIN Bulletin*, **52**(1), 333–360.

Smith, L. N. (2017). Cyclical learning rates for training neural networks, in *Proceedings – 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*, pp. 464–472. Institute of Electrical and Electronics Engineers Inc.

Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, **36**(1), 75–85.

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

Takeuchi, I., Le, Q. V., Sears, T. D. & Smola, A. J. (2006). Nonparametric quantile estimation. *Journal of Machine Learning Research*, **7**, 1231–1264.

Taylor, G. C. & Ashe, F. R. (1983). Second moments of estimates of outstanding claims. *Journal of Econometrics*, **23**(1), 37–61.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, **58**(1), 267–288.

Van Der Smagt, P. & Hirzinger, G. (2012). *Solving the Ill-Conditioning in Neural Network Learning.* Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7700 LECTU, pp. 191–203.

Vaughan, J., Sudjianto, A., Brahimi, E., Chen, J. & Nair, V. N. (2018). Explainable neural networks based on additive index models. arXiv.

Wang, R., Fu, B., Fu, G. & Wang, M. (2017). Deep & cross network for ad click predictions. arXiv.

Wilmoth, J., Barbieri, M., Winant, C., Coche, A., Boe, C., Andreeva, M. & Yang, L. (2020). *United States Mortality Database.*

Wilson, A. G. & Izmailov, P. (2020). Bayesian Deep Learning and a Probabilistic Perspective of Generalization. arXiv.

Wüthrich, M. V. (2018). Machine learning in individual claims reserving. *Scandinavian Actuarial Journal*, **2018**(6), 465–480.

Wüthrich, M. V. (2019a). Bias regularization in neural network models for general insurance pricing. *European Actuarial Journal*, 1–24.

Wüthrich, M. V. (2019b). From Generalized Linear Models to Neural Networks, and Back. SSRN Electronic Journal.

Wüthrich, M. V. & Merz, M. (2012). *Stochastic Claims Reserving Methods in Insurance* (Vol. **435**). John Wiley & Sons.

Wüthrich, M. V. & Merz, M. (2019). Yes, we CANN! *ASTIN Bulletin: The Journal of the IAA*, **49**(1), 1–3.

Zou, H. & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, **67**(2), 301–320.

A Abbreviations and Notation

Tables 22 and 23 provide a summary of abbreviations used in the text.

Table 22. Summary of abbreviations and definitions used in the text

Abbreviation	Definition
AIC	Aikaike Information Criterion
ARIMA	Autoregressive Integrated Moving Average
CANN	Combined Actuarial Neural Network
CAXNN	Combined Actuarial eXplainable Neural Network
ELU	Exponential Linear Unit
GLM	Generalized Linear Model
HMD	Human Mortality Database
i.i.d	independent and identically distributed
IBNR	Incurred But Not Reported
IRLS	Iteratively Reweighted Least Squares
LC	Lee-Carter
MLE	Maximum Likelihood Estimation
MSE	Mean Squared Error
NAM	Neural Additive Model
ReLu	Recitifed Linear unit
tanh	Hyperbolic tan function
USMD	United States Human Mortality Database
XNN	eXplainable Neural Network

Table 23. Summary of notation and definitions used in the text

Notation	Definition
X	Predictor variables
X'	Predictor variables after feature engineering
$X_{i,j}$	Predictor variable j for observation i
y	Target/unknown variable
$\hat{y}$	Estimated target variable
N	Number of observations
P	Number of predictor variables
b	Intercept/bias
B	Regression coefficients/weights
T	Transpose
σ	Activation function
Z	Neurons in hidden layer of network
τ	Quantile of target variable distribution
τ_u	Upper quantile
τ_l	Lower quantile
y^{τ_u}	Estimated upper quantile of target variable
y^{τ_l}	Estimated lower quantile of target variable

```
Year <- layer_input(shape = c(1), dtype = 'float32', name = 'Year')
Age <- layer_input(shape = c(1), dtype = 'int32', name = 'Age')
Country <- layer_input(shape = c(1), dtype = 'int32', name = 'Country')
Sex <- layer_input(shape = c(1), dtype = 'int32', name = 'Sex')
SubNat <- layer_input(shape = c(1), dtype = 'int32', name = 'SubNat')

Age_embed = Age %>%
  layer_embedding(input_dim = 100, output_dim = 5,input_length = 1, name = 'Age_embed') %>%
  keras::layer_flatten()

Sex_embed = Sex %>%
  layer_embedding(input_dim = 2, output_dim = 5,input_length = 1, name = 'Sex_embed') %>%
  keras::layer_flatten()

Country_embed = Country %>%
  layer_embedding(input_dim = country_dim, output_dim = 10,input_length = 1, name = 'Country_embed') %>%
  keras::layer_flatten()

SubNat_embed = SubNat %>%
  layer_embedding(input_dim = region_dim, output_dim = 10,input_length = 1, name = 'SubNat_embed') %>%
  keras::layer_flatten()

feats <- list(Year,Age_embed,Sex_embed,Country_embed,SubNat_embed) %>%
  layer_concatenate() %>%
  layer_batch_normalization() %>%
  layer_dropout(rate = 0.04)

middle=feats%>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%

  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%

  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%

  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05)

main_output =

  layer_concatenate(list(feats, middle)) %>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'main_output')

model <- keras_model(inputs = c(Year,Age,Country,Sex,SubNat),
  outputs = c(main_output))
```

Listing 5: Keras code to fit a neural network to the national and sub-national mortality data.

```
pinball_loss = function(y_true, y_pred){
  tau=0.025
  err = y_true - y_pred
  k_mean(k_maximum(tau * err, (tau - 1) * err), axis=-1)
}
```

Listing 6: Keras code for the pinball loss function to estimate the 0.95 quantile.

```
deep_ensemble_loss <- function(sigma) {
  loss_fun = function(y, y_hat) {
    0.5*k_log(sigma + k_epsilon()) + 0.5*k_square(y - y_hat) / (sigma + k_epsilon())
  }
  return(loss_fun)
}

bias_loss = function(y, y_hat) {
  dat_sd = k_var(y, axis = 1)
  pred_sd = k_mean(y_hat, axis=1)
  return(loss_mean_squared_error(dat_sd, pred_sd))
}
```

Listing 7: Keras code for the Deep Ensemble method.

```
### Modify last layer of model to apply Deep Ensembles

last = layer_concatenate(list(feats, middle)) %>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05)

main_output = last  %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'main_output')

var_output = last  %>%
  layer_dense(units = 1, activation = 'softplus', name = 'var_output')

model <- keras_model(
  inputs = c(Year,Age,Country,Sex,SubNat),
  outputs = c(main_output, var_output ))

model %>% compile(
  optimizer = adam,
  loss = list(deep_ensemble_loss(var_output), bias_loss),
  experimental_run_tf_function=FALSE )

### Modify last layer of model to apply Pinball Loss

last = layer_concatenate(list(feats, middle)) %>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05)

main_output = last  %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'main_output')

lower_output = last  %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'lower_output')

upper_output = last  %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'upper_output')

model <- keras_model(
  inputs = c(Year,Age,Country,Sex,SubNat),
  outputs = c(main_output, lower_output,upper_output))

pinball_loss_down = function(y_true, y_pred){
  tau=0.025
  err = y_true - y_pred
  k_mean(k_maximum(tau * err, (tau - 1) * err), axis=-1)
}

pinball_loss_up = function(y_true, y_pred){
  tau=0.975
  err = y_true - y_pred
  k_mean(k_maximum(tau * err, (tau - 1) * err), axis=-1)
}

model %>% compile(
  optimizer = adam,
  loss = list("mse",pinball_loss_down,pinball_loss_up),
  experimental_run_tf_function=FALSE)
```

Listing 8: Keras code to modify the mortality forecasting model to estimate uncertainty.

```
middle=feats%>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%

  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%

  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%

  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05)

last = layer_concatenate(list(feats, middle)) %>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05)

main_output = last  %>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'main_output')

lower_output = last  %>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'lower_output')

upper_output = last  %>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'upper_output')

model <- keras_model(
  inputs = c(Year,Age,Country,Sex,SubNat),
  outputs = c(main_output, lower_output,upper_output))
```

Listing 9: Keras code to fit a neural network to the national and sub-national mortality data that estimates uncertainty using the pinball loss and \branches”.

```
############### Build embedding layers
Year <- layer_input(shape = c(1), dtype = 'float32', name = 'Year')
Age <- layer_input(shape = c(1), dtype = 'int32', name = 'Age')
Country <- layer_input(shape = c(1), dtype = 'int32', name = 'Country')
Sex <- layer_input(shape = c(1), dtype = 'int32', name = 'Sex')
SubNat <- layer_input(shape = c(1), dtype = 'int32', name = 'SubNat')

Age_embed = Age %>%
  layer_embedding(input_dim = 100, output_dim = 5,input_length = 1, name = 'Age_embed') %>%
  keras::layer_flatten()

Sex_embed = Sex %>%
  layer_embedding(input_dim = 2, output_dim = 5,input_length = 1, name = 'Sex_embed') %>%
  keras::layer_flatten()

Country_embed = Country %>%
  layer_embedding(input_dim = country_dim, output_dim = 10,input_length = 1, name = 'Country_embed') %>%
  keras::layer_flatten()

SubNat_embed = SubNat %>%
  layer_embedding(input_dim = region_dim, output_dim = 10,input_length = 1, name = 'SubNat_embed') %>%
  keras::layer_flatten()

Year_net = Year %>%
  layer_dense(units = 32, activation = "tanh") %>%
  layer_dense(units = 32, activation = "tanh") %>%
  layer_dense(units = 5, activation = "tanh")

feats <- list(Year_net,Age_embed,Sex_embed,Country_embed,SubNat_embed) %>%
  layer_concatenate() %>%
  layer_batch_normalization()

middle=feats%>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%

  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%

  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%

  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05)

middle = list(feats, middle)%>%
  layer_concatenate() %>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05)

# skip connection

main_output = layer_concatenate(list(feats, middle))  %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'main_output')

model <- keras_model(
  inputs = c(Year,Age,Country,Sex,SubNat),
  outputs = c(main_output))
```

Listing 10: Keras code to fit a CANN model to the national and sub-national mortality data.

```
Year <- layer_input(shape = c(1), dtype = 'float32', name = 'Year')
Age <- layer_input(shape = c(1), dtype = 'int32', name = 'Age')
Country <- layer_input(shape = c(1), dtype = 'int32', name = 'Country')
Sex <- layer_input(shape = c(1), dtype = 'int32', name = 'Sex')
SubNat <- layer_input(shape = c(1), dtype = 'int32', name = 'SubNat')

Age_embed = Age %>%
  layer_embedding(input_dim = 100, output_dim = 5,input_length = 1, name = 'Age_embed') %>%
  keras::layer_flatten()

Sex_embed = Sex %>%
  layer_embedding(input_dim = 2, output_dim = 5,input_length = 1, name = 'Sex_embed') %>%
  keras::layer_flatten()

Country_embed = Country %>%
  layer_embedding(input_dim = country_dim, output_dim = 10,input_length = 1, name = 'Country_embed') %>%
  keras::layer_flatten()

SubNat_embed = SubNat %>%
  layer_embedding(input_dim = region_dim, output_dim = 10,input_length = 1, name = 'SubNat_embed') %>%
  keras::layer_flatten()

feats <- list(Year,Age_embed,Sex_embed,Country_embed,SubNat_embed) %>% layer_concatenate() %>% layer_batch_normalization() %>% layer_dropout(rate = 0.04)

middle=feats%>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%

  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%

  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%

  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05)

last = layer_concatenate(list(feats, middle)) %>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05)

main_output = last  %>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'main_output')

lower_output = last  %>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'lower_output')

upper_output = last  %>%
  layer_dense(units = 128, activation = 'tanh') %>%
  layer_batch_normalization() %>%
  layer_dropout(0.05) %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'upper_output')

model <- keras_model(
  inputs = c(Year,Age,Country,Sex,SubNat),
  outputs = c(main_output, lower_output,upper_output))
```

Listing 11: Keras code to fit the linear component of a CAXNN model to the average predictions of the CANN networks.

```
#### linear
Age_embed_linear = Age %>%
  layer_embedding(input_dim = 100, output_dim = 5,input_length = 1,
                  name = 'Age_embed_linear',
                  weights = model_weights$layers[[8]] %>% get_weights(),
                  trainable = F) %>%
  keras::layer_flatten()

Sex_embed_linear = Sex %>%
  layer_embedding(input_dim = 2, output_dim = 5,input_length = 1,
                  name = 'Sex_embed_linear',
                  weights = model_weights$layers[[9]] %>% get_weights(),
                  trainable = F)  %>%
  keras::layer_flatten()

Country_embed_linear = Country %>%
  layer_embedding(input_dim = country_dim, output_dim = 10,input_length = 1,
                  name = 'Country_embed_linear',
                  weights = model_weights$layers[[10]] %>% get_weights(),
                  trainable = F)  %>%
  keras::layer_flatten()

SubNat_embed_linear = SubNat %>%
  layer_embedding(input_dim = region_dim, output_dim = 10,input_length = 1,
                  name = 'SubNat_embed_linear',
                  weights = model_weights$layers[[11]] %>% get_weights(),
                  trainable = F)  %>%
  keras::layer_flatten()

Year_net_linear = Year %>%
  layer_dense(units = 32, activation = "tanh",
              weights = model_weights$layers[[2]] %>% get_weights(), trainable = F) %>%
  layer_dense(units = 32, activation = "tanh",
              weights = model_weights$layers[[7]] %>% get_weights(), trainable = F) %>%
  layer_dense(units = 5, activation = "tanh",
              weights = model_weights$layers[[12]] %>% get_weights(), trainable = F)
```

Listing 12: Keras code to fit the full CAXNN model to the average predictions of the CANN networks: linear weights transferred from previously trained model.

```
skip_net = function(dat){

  ### concatenate input layers
  concat_dat = dat %>% layer_concatenate()

  ### cross layer
  cross = concat_dat %>%
    layer_dense(25, activation = "exponential")

  ### first dense layer
  temp = list(concat_dat, cross) %>%
    layer_concatenate() %>%
    layer_dense(units =128, activation = "tanh") %>%
    layer_batch_normalization()

  ### skip connection and second dense layer
  out = list(temp, concat_dat, cross) %>% layer_concatenate() %>%
    layer_dense(units =128, activation = "tanh") %>%
    layer_batch_normalization() %>%
    layer_dense(units = 1, activation = "tanh")

  ### return output
  return(list(concat = concat_dat, temp = temp, out = out))

}
```

Listing 13: Keras code to define the network used in the XNN.

```
#### non-linear embeddings
Age_embed = Age %>%
  layer_embedding(input_dim = 100, output_dim = 5,input_length = 1,
  name = 'Age_embed') %>%
  keras::layer_flatten()

Sex_embed = Sex %>%
  layer_embedding(input_dim = 2, output_dim = 5,input_length = 1,
  name = 'Sex_embed')  %>%
  keras::layer_flatten()

Country_embed = Country %>%
  layer_embedding(input_dim = country_dim, output_dim = 10,input_length = 1,
  name = 'Country_embed')  %>%
  keras::layer_flatten()

SubNat_embed = SubNat %>%
  layer_embedding(input_dim = region_dim, output_dim = 10,input_length = 1,
  name = 'SubNat_embed')  %>%
  keras::layer_flatten()

Year_net = Year %>%
  layer_dense(units = 32, activation = "tanh") %>%
  layer_dense(units = 32, activation = "tanh") %>%
  layer_dense(units = 5, activation = "tanh")

feats <- list(Year_net_linear, Age_embed_linear,Sex_embed_linear,
Country_embed_linear,SubNat_embed_linear) %>%
  layer_concatenate() %>%
  layer_batch_normalization()

Age_Sex                  = list(Sex_embed,Age_embed) %>% skip_net()
Year_Sex                 = list(Year_net,Age_embed) %>% skip_net()
Year_Country             = list(Year_net,Country_embed) %>% skip_net()
Year_Region              = list(Year_net,SubNat_embed) %>% skip_net()
Region_Country           = list(SubNat_embed,Country_embed) %>% skip_net()
Sex_Country              = list(Sex_embed, SubNat_embed,Country_embed) %>% skip_net()
Year_Age                 = list(Year_net,Age_embed) %>% skip_net()
Year_Age_Sex             = list(Year_net,Age_embed,Sex_embed) %>% skip_net()
Year_Age_Country         = list(Year_net,Age_embed,Country_embed) %>% skip_net()
Year_Age_Country_Region = list(Year_net,Age_embed,Country_embed,SubNat_embed) %>% skip_net()
Year_Country_Region      = list(Year_net,Country_embed,SubNat_embed) %>% skip_net()
Age_Country_Region       = list(Age_embed,Country_embed,SubNat_embed) %>% skip_net()

middle = list(Age_Sex$out, Year_Sex$out, Year_Country$out,Region_Country$out,
              Sex_Country$out,Year_Age$out, Year_Age_Sex$out,Year_Age_Country$out,
              Year_Age_Country_Region$out, Year_Country_Region$out, Year_Region$out,
              Age_Country_Region$out) %>%
  layer_concatenate() %>%
  layer_batch_normalization()

# skip connection

main_output = layer_concatenate(list(feats, middle))  %>%
  layer_dense(units = 1, activation = 'sigmoid', name = 'main_output')

model <- keras_model(
  inputs = c(Year,Age,Country,Sex,SubNat),
  outputs = c(main_output))
```

Listing 14: Keras code to fit the full CAXNN model to the average predictions of the CANN networks.