JIRSS (2022)
Vol. 21, No. 01, pp 81-103
DOI: 10.22034/JIRSS.2022.704621

Transformer Self-Attention Network for Forecasting Mortality Rates

Amin Roshani [1], **Muhyiddin Izadi** [1], **Baha-Eldin Khaledi** [2]

[1] Department of Statistics, Razi University, Kermanshah, Iran.
[2] Department of Applied Statistics and Research Methods, University of Northern Colorado, Greeley, CO 80636, USA.

Received: 15/04/2022, Accepted: 21/12/2022, Published online: 03/05/2023

Abstract. The transformer network is a deep learning architecture that uses self-attention mechanisms to capture the long-term dependencies of a sequential data. The Poisson-Lee-Carter model, introduced to predict mortality rate, includes the factors of age and the calendar year, which is a time-dependent component. In this paper, we use the transformer to predict the time-dependent component in the Poisson-Lee-Carter model. We use the real mortality data set of some countries to compare the mortality rate prediction performance of the transformer with that of the long short-term memory (LSTM) neural network, the classic ARIMA time series model and simple exponential smoothing method. The results show that the transformer dominates or is comparable to the LSTM, ARIMA and simple exponential smoothing method.

Keywords. Auto-Regressive Integrated Moving Average, Human Mortality Database, Long Short-Term Memory, Mean Absolute Percentage Error, Poisson-Lee-Carter Mortality Model, Recurrent Neural Network, Simple Exponential Smoothing, Time Series

Amin Roshani (roshani.amin@gmail.com)
Corresponding Author: Muhyiddin Izadi (izadi_552@yahoo.com)
Baha-Eldin Khaledi (bahaedin.khaledi@unco.edu)

Prediction.

MSC: 62P05, 62-08.

1 Introduction

Predicting future mortality rates enables governments and demographic information organizations such as social security, insurance companies, etc., to plan for the future more accurately. Therefore, mortality rate models have garnered increased attention and many researchers have attempted to introduce and develop mortality rate models with more accurate predictions. Some well-known mortality rate models in the literature are:

Lee-Carter (LC) (Lee and Carter, 1992), Poisson LC (PLC) (Brouhns et al., 2002), RH (Renshaw and Haberman, 2006), augmented common factor (Li and Lee, 2005), Poisson common factor (Li, 2013), Bayesian Poisson log-bilinear (Antonio et al., 2015) and Bayesian Poisson common factor with overdispersion (Roshani et al., 2022). Among these, the PLC model remains a popular and widely used model. For a comprehensive review of mortality rate modellings, one can see Hunt and Blake (2021).

Suppose that $D_{x,t}, m_{x,t}$ and $E_{x,t}$ are the number of deaths, central death rate and central exposure-to-risk, respectively, for age x in year t. The PLC model is

$$\begin{aligned} D_{x,t} &\sim \mathbb{P}\text{oisson}(E_{x,t}\, m_{x,t}), \\ \log m_{x,t} &= \alpha_x + \beta_x \kappa_t, \qquad x \in \mathcal{X}, t \in \mathcal{T}, \end{aligned} \tag{1.1}$$

where $\mathcal{X} = \{x_i \mid i = 1, 2, \ldots, A\}$, A is the number of age groups, $\mathcal{T} = \{t_i \mid i = 1, 2, \ldots, T\}$, α_x measures the age effect, κ_t is a time-dependent parameter and β_x is the age-related parameter which measures the marginal effect of κ_t over time on the log mortality rate. The constraints $\sum_{i=1}^{A} \beta_{x_i} = 1$ and $\sum_{i=1}^{T} \kappa_{t_i} = 0$ are taken into account to ensure model identification (see Brouhns et al., 2002). In this model and the other introduced mortality rate models, after estimating the parameters α_x's, β_x's and κ_t's, an auto-regressive integrated moving average (ARIMA) time series model is fitted to $\hat{\kappa}_t$, $t \in \mathcal{T}$, to predict the future mortality rates, where $\hat{\kappa}_t$ is the estimate of κ_t, $t \in \mathcal{T}$.

Machine learning algorithms, particularly neural networks, have recently gained popularity as a tool in some applications such as natural language processing (NLP), computer vision, automatic speech recognition, social network filtering, and medical diagnosis. Various neural network techniques were developed, each used for a specific application; convolutional, perceptron, recurrent, and transformer are just a few examples. Recurrent neural networks (RNNs) retain past or historical information to predict future values. This property has led to their application in the analysis of sequential data such as time-series data. Some examples of sequential data are a series of data points; Audio, video, text, biomedical data such as EEG signals or DNA sequence data and financial data such as stock prices.

RNNs such as LSTM (introduced in Section A.1) (Hochreiter and Schmidhuber, 1997) and Gated recurrent unit (GRU) (Chung et al., 2014) have been recently used in mortality rate models to predict the calendar year effect. Richman and Wüthrich (2019) predicted Swiss female and male mortality rates using LSTM and GRU architectures to predict the mortality rate and compared them with the LC model. Nigri et al. (2019) introduced a new approach based on LSTM architecture to predict the time-dependent component, κ_t, in the LC model. They applied the technique to data from six countries separately for males and females. Perla et al. (2021) generalized the LC model using a simple convolutional network model as well as an LSTM network. Choi (2021) proposed the 6-parameter model and used LSTM to forecast time-dependent factors in conjunction with traditional time series methods such as vector autoregression (VAR).

The transformer (introduced in Section A.2) is a deep learning architecture that was first proposed for NLP by a group of Google researchers (Vaswani et al., 2017). This architecture relies on encoder-decoder attention mechanisms rather than recurrent layers. Transformers avoids the vanishing gradient problem that plagues RNNs (Pascanu et al., 2013) and use self-attention mechanisms to capture the long-term dependencies. Transformers are much faster to train and easier to parallelize (Géron, 2019).

Recently, transformers have been used to predict time series. Wu et al. (2020) used the transformer network for forecasting influenza-like illness and show that it outperforms the LSTM and Seq2Seq models. Farsani and Pazouki (2021) used the transformer network to provide more accurate time series prediction over longer time intervals.

To the best of our knowledge, time-series transformer architecture have not been used in the mortality rate modelling in the literature. Therefore, the rest of this paper is organized as follows. In Section 2, we use the transformer architecture to predict the time-dependent component in the PLC mortality rate model. Using the mortality data of several countries, we compare the transformer with the LSTM, the best ARIMA time series model (Box et al., 2015), and the best simple exponential smoothing method denoted by SES (Hyndman et al., 2008) for predicting the time-dependent parameter.

The results show that the mortality rate prediction performance of the transformer outperforms or is comparable to those of the LSTM, the best ARIMA model, and the best SES technique. We explain the transformer and LSTM neural networks in Appendix A.

2 Methodology and Data Analysis

In this section, we first describe the data, then we explain the methodology. We use the male and female mortality rates from Japan, Australia, Sweden, Italy, France, Switzerland, Austria, Norway, Denmark, Canada, and the United States. The source of the data is the Human Mortality Database (HMD[1]). We consider the ages from 0 to 89, so A = 90. The time period selected for each country begins in 1950 and ends depending on the information in the HMD. We use the data from 1950 to 2000 as the train data ($\mathcal{T}_{\text{train}}$) and from 2001 onward as the test data ($\mathcal{T}_{\text{test}}$) which is shown in Table 1.

Table 1: Total, training and testing set by country

Country	$\mathcal{T}$	$\mathcal{T}_{\text{train}}$	$\mathcal{T}_{\text{test}}$
ITA	1950–2017	1950–2000	2001-2017
SWE	1950–2018	1950–2000	2001-2018
FRA	1950–2018	1950–2000	2001-2018
CHE	1950–2018	1950–2000	2001-2018
AUT	1950–2017	1950–2000	2001-2017
NOR	1950–2018	1950–2000	2001-2018
DEN	1950–2019	1950–2000	2001-2019
USA	1950–2017	1950–2000	2001-2017
CAN	1950–2016	1950–2000	2001-2016
JPN	1950–2018	1950–2000	2001-2018
AUS	1950–2018	1950–2000	2001-2018

Using `StMoMo` (Villegas et al., 2018) package in `R` software (R Core Team, 2021), we fit the PLC model to the train data for male and female groups of each country and estimate the parameters a_x, b_x, and κ_t denoted by $\hat{a}_x$, $\hat{b}_x$, and $\hat{\kappa}_t$, respectively. The estimated parameters $\hat{\kappa}_t$, $t \in \mathcal{T}_{\text{train}}$, are considered as the input data for the transformer, LSTM, ARIMA and SES, to predict future values κ_t, $t \in \mathcal{T}_{\text{test}}$. We apply the `auto.arima`

[1]www.mortality.org

and ses functions in the `forecast` package in R (Hyndman and Khandakar, 2008) to find the best ARIMA and SES. The parameters of ARIMA model for all the countries are determined and included in Table 2.

Table 2: Best ARIMA model for each country and gender.

Country	Male	Female
ITA	ARIMA(0,2,3)	ARIMA(0,1,1) with drift
SWE	ARIMA(2,2,2)	ARIMA(0,1,1) with drift
FRA	ARIMA(0,1,1) with drift	ARIMA(0,1,1) with drift
CHE	ARIMA(0,2,2)	ARIMA(0,1,1) with drift
AUT	ARIMA(1,2,1)	ARIMA(1,1,0) with drift
NOR	ARIMA(1,2,1)	ARIMA(0,1,1) with drift
DEN	ARIMA(1,1,0) with drift	ARIMA(1,1,0) with drift
USA	ARIMA(0,2,1)	ARIMA(0,1,0) with drift
CAN	ARIMA(0,2,1)	ARIMA(0,1,0) with drift
JPN	ARIMA(0,1,1) with drift	ARIMA(0,1,1) with drift
AUS	ARIMA(0,2,2)	ARIMA(1,1,0) with drift

For the LSTM and the transformer networks, we divide the training data into two sets, a reduced training data set (80% of the training data) and a validation data set (20% of the training data). Because, the random initial values of the learning parameters provide random prediction, we apply each network 50 times on the reduced training data for various selections of hyper-parameters and predict κ_t, for $t \in \mathcal{T}_{\text{validation}}$. Then, we calculate the average MSE based on the validation data set for the selected hyper-parameters. The hyper-parameters with the minimum average MSE are chosen. Next, we use the network with optimum hyper-parameters to predict κ_t, $t \in \mathcal{T}_{\text{test}}$, using the whole training data for 50 times. Now, we use the average of predicted κ_t's in the PLC model to predict the mortality rate, $m_{x,t}$, for $t \in \mathcal{T}_{\text{test}}$, denoted by $\hat{m}_{x,t}$. Finally, we compute the mean absolute percentage error (MAPE) measure defined below, that is used to compare our transformer results with those of LSTM, ARIMA and SES.

It is worth to mention that, for the transformer and LSTM to have a high accurate prediction, we make the time series stationary and then re-scale it to $[0, 1]$ (Brownlee, 2017, p. 87).

The MAPE which is a common model selection measure of accuracy is used to compare the above four models in predicting mortality rates. This criterion is defined as

$$\text{MAPE} = \frac{1}{N} \sum_{(x,t)\in\mathcal{X}\times\mathcal{T}_{\text{test}}} \left|\frac{\log(\hat{m}_{x,t}) - \log(m_{x,t})}{\log(m_{x,t})}\right|,$$

where N is the cardinality of $\mathcal{X} \times \mathcal{T}_{\text{test}}$.

For all countries, we use a single hidden layer and the sigmoid activation function for LSTM architecture. Other hyperparameters such as the number of neurons, the learning rate, the length of input samples and the number of epochs[2], depend on the countries (Table 3). In transformer architecture, we use the Rectified Linear Unit (ReLU) activation function in Feed-Forward layers for both encoder and decoder blocks. The number of encoder and decoder blocks and the number of heads in multi-head attention are set to be one. The input length of the encoder and decoder, the learning rate, the number of epochs, and other hyperparameters, are explained in Section A.2, depend on the countries (Table 4).

We use the open-source python libraries `PyTorch` (Paszke et al., 2019) and `Keras` (Chollet et al., 2015) to apply the transformer and LSTM networks. In addition, to use the output of these libraries in `R`, we used `Reticulate` package (Ushey et al., 2021).

We apply these four methods on the mortality data of the countries displayed in Table 1. The computed MAPE for the test data is presented in Table 5. For males, the MAPE of the transformer is less than that of the LSTM, SES and ARIMA for all the countries except Italy and Norway. It is worth to mention that the MAPE result for Norway using transformer is less than LSTM and SES and is comparable with that of ARIMA. That is, the transformer network provides more accurate predictions than the LSTM, SES and ARIMA. For the female group, the results obtained by transformer, LSTM, SES and ARIMA are comparable as presented in Table 5. We plot the mortality rate predictions for ages 20, 40, 60 and 80, along with actual values for Sweden based on the transformer, LSTM, ARIMA and SES method in Figure 1. Finally, in Tables 6, 7, 8 and 9, we present 2022 prediction of mortality rate for ages of 20, 40, 60 and 80 for males and females, respectively.

After obtaining the optimum parameters, the prediction running time of LSTM and Transformer algorithms are given in Table 10. We see that the running time for Transformer in considerably shorter than LSTM. The system that we used is Google Collaboration Platform with 2-core Intel® Xeon® 2.20GHz CPU and 13.6 GB RAM.

[2]One epoch is when all samples in the training dataset are processed once, and the network weights are updated.

Table 3: Hyperparameters of LSTM network for each country and gender.

Country	Gender	LSTM hyperparameters			
		length of input	number of neurons	learning rate	epoch
ITA	Male	8	1	0.0001	200
	Female	4	4	0.01	200
SWE	Male	4	1	0.0001	200
	Female	16	16	0.01	200
FRA	Male	8	8	0.01	200
	Female	2	32	0.001	200
CHE	Male	4	1	0.0001	200
	Female	1	8	0.001	200
AUT	Male	16	8	0.01	200
	Female	1	1	0.0001	200
NOR	Male	32	1	0.01	200
	Female	32	1	0.01	200
DEN	Male	1	1	0.0001	200
	Female	32	2	0.01	200
USA	Male	16	1	0.0001	200
	Female	32	1	0.01	200
CAN	Male	16	32	0.01	200
	Female	16	8	0.0001	200
JPN	Male	32	32	0.0001	200
	Female	4	32	0.01	200
AUS	Male	32	8	0.01	200
	Female	16	1	0.01	200

Table 4: Hyperparameters of transformer network for each country and gender.

Country	Gender	Transformer hyperparameters								
		encoder length	decoder length	d	d_k	number of heads	number of encoder layers	number of decoder layers	learning rate	epoch
ITA	Male	32	8	10	5	1	1	1	0.0001	200
	Female	2	2	128	64	1	1	1	0.0001	400
SWE	Male	32	32	128	64	1	1	1	0.01	200
	Female	8	8	128	64	1	1	1	0.01	200
FRA	Male	32	2	10	5	1	1	1	0.0001	400
	Female	2	2	10	5	1	1	1	0.0001	400
CHE	Male	32	32	10	5	1	1	1	0.01	200
	Female	2	2	10	5	1	1	1	0.0001	400
AUT	Male	32	2	10	5	1	1	1	0.0001	400
	Female	32	4	10	5	1	1	1	0.0001	400
NOR	Male	2	2	12	6	1	1	1	0.01	200
	Female	16	16	10	5	1	1	1	0.01	200
DEN	Male	16	8	128	64	1	1	1	0.0001	200
	Female	8	2	10	5	1	1	1	0.01	200
USA	Male	16	16	10	5	1	1	1	0.0001	400
	Female	32	4	10	5	1	1	1	0.0001	400
CAN	Male	16	2	10	5	1	1	1	0.0001	400
	Female	32	2	128	64	1	1	1	0.0001	400
JPN	Male	32	2	10	5	1	1	1	0.0001	400
	Female	8	2	128	64	1	1	1	0.01	200
AUS	Male	2	2	10	5	1	1	1	0.01	200
	Female	16	2	10	5	1	1	1	0.0001	400

Table 5: MAPE of ARIMA, SES, LSTM and Transformer for each country and gender.

	Male				Female			
Country	ARIMA	SES	LSTM	Transformer	ARIMA	SES	LSTM	Transformer
ITA	4.5669	4.9164	4.4717	4.5284	1.9021	2.6465	2.2168	1.9528
SWE	4.1184	4.2207	4.0705	4.0049	2.5426	2.4253	2.4831	2.4334
FRA	3.8747	3.3165	3.175	2.9727	2.4898	2.4910	2.4876	2.4845
CHE	4.2215	4.3584	4.7111	4.0611	3.1667	3.1890	3.1691	3.1919
AUT	3.8224	3.8980	3.8639	3.7405	3.1895	3.3819	3.2133	3.3196
NOR	5.656	11.6106	5.9546	5.6641	3.0841	3.0346	3.1913	3.0327
DEN	7.005	6.9791	6.9546	6.6331	5.298	5.0316	5.1574	5.4565
USA	2.5338	2.5351	2.5343	2.4508	1.7678	2.1363	2.1783	2.1910
CAN	3.4104	3.7195	3.3526	3.2759	1.5878	1.6255	1.5996	1.5747
JPN	3.1593	3.0090	2.9173	2.9162	7.6805	7.8694	7.4117	7.5783
AUS	3.9564	5.4907	5.5608	3.6050	2.1823	3.8583	2.1837	2.2083

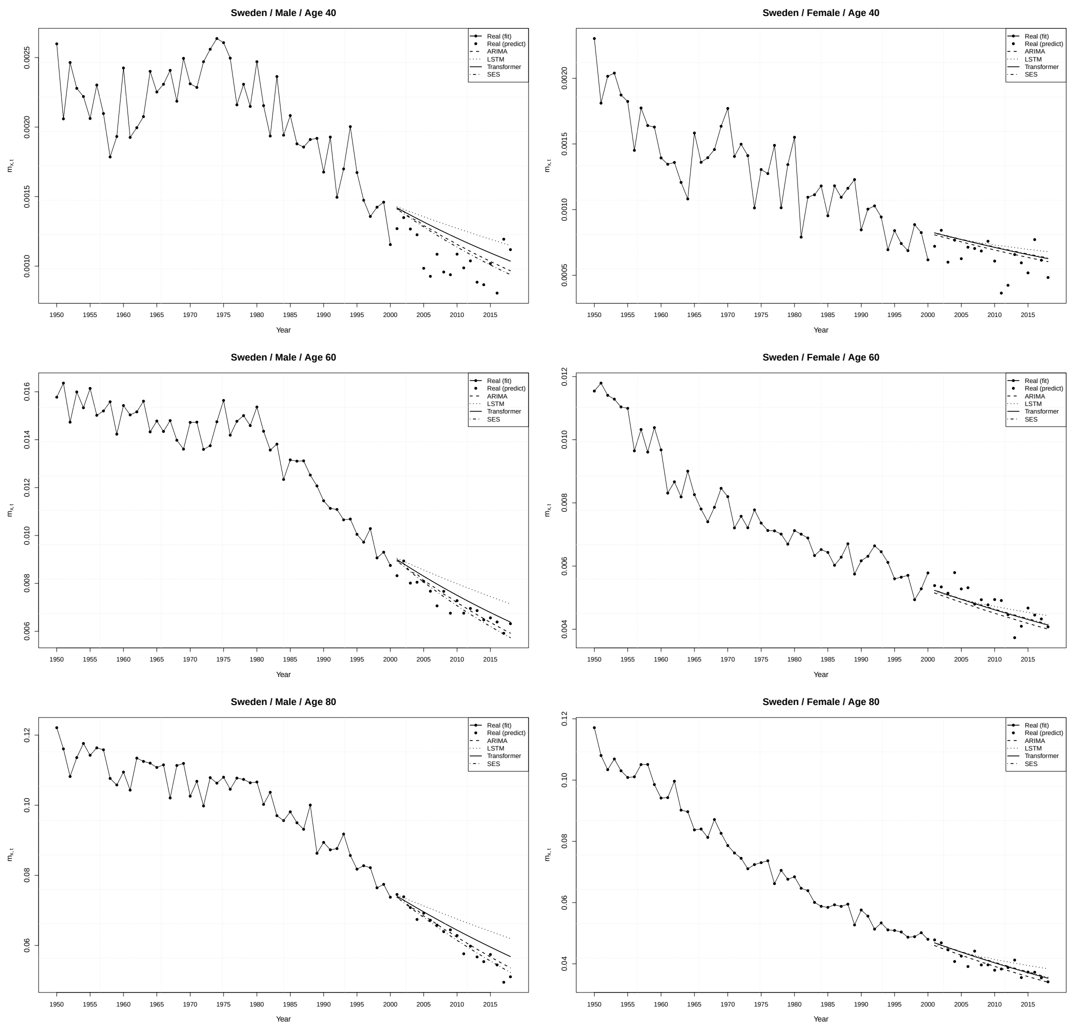

Figure 1: Plots of fitted and projected mortality rates for Swedish male and female groups (The left side for males and the right side for females) with ages 20, 40 and 85 accompanied by observed crude mortality rates.

Table 6: Mortality rate for Male in year 2022.

Country	Model	Age			
		20	40	60	80
ITA	ARIMA	0.0004535	0.0008014	0.0060383	0.0501608
	SES	0.0004522	0.0007986	0.0060191	0.0500402
	LSTM	0.0004399	0.0007729	0.0058398	0.0489068
	Transformer	0.0004384	0.0007698	0.0058183	0.0487702
SWE	ARIMA	0.0004308	0.0008134	0.005490	0.049991
	SES	0.0004673	0.0008848	0.0059591	0.0533261
	LSTM	0.0004398	0.0008311	0.0056063	0.0508236
	Transformer	0.0004191	0.0007906	0.0053401	0.0489129
FRA	ARIMA	0.0007042	0.001521	0.0084967	0.0447384
	SES	0.0006996	0.0015103	0.0084366	0.0443904
	LSTM	0.0006894	0.0014867	0.0083031	0.0436179
	Transformer	0.0007116	0.0015383	0.0085944	0.0453044
CHE	ARIMA	0.0004955	0.0007738	0.0050696	0.0427874
	SES	0.0004942	0.0007718	0.0050561	0.0426957
	LSTM	0.000495	0.000773	0.0050645	0.0427528
	Transformer	0.0004992	0.0007793	0.0051074	0.0430435
AUT	ARIMA	0.0006134	0.0010305	0.0078522	0.0523279
	SES	0.0006169	0.0010363	0.0078897	0.0525628
	LSTM	0.0006066	0.0010192	0.0077792	0.0518701
	Transformer	0.0006047	0.0010159	0.0077579	0.0517367
NOR	ARIMA	0.0005491	0.0008312	0.0052029	0.0490369
	SES	0.0005497	0.0008324	0.0052109	0.0490897
	LSTM	0.0005443	0.0008226	0.0051431	0.0486408
	Transformer	0.0005588	0.0008487	0.0053244	0.049837

Table 7: (continued) Mortality rate for Male in year 2022.

Country	Model	Age			
		20	40	60	80
DEN	ARIMA	0.0003637	0.0011184	0.0079847	0.055705
	SES	0.0003233	0.0010284	0.0073781	0.0519341
	LSTM	0.0003807	0.0011554	0.0082333	0.0572407
	Transformer	0.0003667	0.0011249	0.008028	0.0559724
USA	ARIMA	0.0011283	0.0021099	0.0100346	0.0578829
	SES	0.0011869	0.0022258	0.0108314	0.0614882
	LSTM	0.0012018	0.0022553	0.0110372	0.0624101
	Transformer	0.0011955	0.0022429	0.0109505	0.0620221
CAN	ARIMA	0.0005949	0.001029	0.0061699	0.0496926
	SES	0.0006219	0.0010683	0.0064465	0.0511498
	LSTM	0.0006242	0.0010716	0.0064697	0.051271
	Transformer	0.0006232	0.0010701	0.0064592	0.0512164
JPN	ARIMA	0.0003193	0.0008715	0.0061243	0.0447223
	SES	0.0003243	0.0008826	0.0061898	0.045197
	LSTM	0.0003236	0.000881	0.00618	0.0451257
	Transformer	0.0003241	0.0008822	0.006187	0.0451769
AUS	ARIMA	0.0005728	0.0011092	0.005228	0.0456955
	SES	0.0005272	0.0010455	0.0047616	0.042778
	LSTM	0.0005681	0.0011027	0.0051797	0.0453968
	Transformer	0.0005339	0.001055	0.0048297	0.0432095

Table 8: Mortality rate for Female in year 2022.

Country	Model	Age			
		20	40	60	80
ITA	ARIMA	0.0001249	0.0004599	0.0032093	0.0293615
	SES	0.0001309	0.0004786	0.0033178	0.0304109
	LSTM	0.0001253	0.000461	0.0032158	0.0294238
	Transformer	0.0001271	0.0004669	0.0032502	0.0297567
SWE	ARIMA	0.0002055	0.0005423	0.004045	0.0329319
	SES	0.0002095	0.0005565	0.0041251	0.0337557
	LSTM	0.0002075	0.0005496	0.0040861	0.0333542
	Transformer	0.0002087	0.0005536	0.0041087	0.0335868
FRA	ARIMA	0.0002307	0.0007129	0.0035631	0.025043
	SES	0.0002339	0.0007221	0.0036071	0.0254237
	LSTM	0.0002327	0.0007187	0.0035908	0.0252822
	Transformer	0.000236	0.0007284	0.0036372	0.025685
CHE	ARIMA	0.0002203	0.0004982	0.003211	0.0271815
	SES	0.0002218	0.0005024	0.0032373	0.0274295
	LSTM	0.000221	0.0005002	0.0032235	0.0272998
	Transformer	0.0002203	0.000498	0.0032103	0.027175
AUT	ARIMA	0.0001978	0.000521	0.0040394	0.0342367
	SES	0.0002278	0.0006104	0.0045304	0.0390488
	LSTM	0.0002145	0.0005707	0.0043151	0.0369286
	Transformer	0.0002085	0.0005527	0.0042161	0.0359585
NOR	ARIMA	0.0002516	0.0005876	0.0041719	0.0328564
	SES	0.00025	0.0005805	0.0041307	0.0324245
	LSTM	0.0002495	0.0005782	0.0041176	0.0322881
	Transformer	0.0002508	0.0005841	0.0041516	0.0326431

Table 9: (continued) Mortality rate for Female in year 2022.

Country	Model	Age			
		20	40	60	80
DEN	ARIMA	0.0001744	0.0007002	0.0058506	0.0367189
	SES	0.0001552	0.0006194	0.0054182	0.0329184
	LSTM	0.0001808	0.0007271	0.0059907	0.037976
	Transformer	0.0001725	0.0006921	0.0058081	0.03634
USA	ARIMA	0.0003773	0.001134	0.0063689	0.0393065
	SES	0.0004007	0.0012136	0.006777	0.0419853
	LSTM	0.0003968	0.0012004	0.0067091	0.0415388
	Transformer	0.0004007	0.0012138	0.0067778	0.041991
CAN	ARIMA	0.0002452	0.0006503	0.0043027	0.0315134
	SES	0.0002491	0.0006616	0.0043704	0.03199
	LSTM	0.0002473	0.0006564	0.0043389	0.0317683
	Transformer	0.0002441	0.0006468	0.0042818	0.0313665
JPN	ARIMA	0.0000901	0.000421	0.0025267	0.0219111
	SES	0.0000953	0.0.000438	0.0026139	0.0227017
	LSTM	0.0000882	0.0004145	0.0024934	0.0216091
	Transformer	0.0000874	0.0004118	0.0024793	0.0214822
AUS	ARIMA	0.0002616	0.0005848	0.0033139	0.0287985
	SES	0.0002383	0.0005187	0.0029183	0.0257003
	LSTM	0.0002637	0.000591	0.0033508	0.0290852
	Transformer	0.0002426	0.0005307	0.0029901	0.0262656

Table 10: The runtime (in seconds) of LSTM and Transformer algorithms by gender.

Country	Male		Female	
	LSTM	Transformer	LSTM	Transformer
ITA	1107.42	82.84	1098.58	269.09
SWE	1005.33	696.11	1639.98	387.42
FRA	1155.77	168.98	1147.67	127.34
CHE	1020.31	99.55	908.50	143.26
AUT	1313.89	166.64	948.79	174.49
NOR	913.22	76.92	972.49	101.34
DEN	915.73	267.19	977.69	80.58
USA	1147.59	190.81	976.91	149.87
CAN	1483.96	177.68	1141.49	394.43
JPN	1162.95	178.85	1088.29	356.84
AUS	1029.15	76.96	1208.58	165.05

Acknowledgements

The authors would like to thank the reviewers for their valuable comments and suggestions, which definitely improved the quality and presentation of the paper.

References

Antonio, K., Bardoutsos, A., and Ouburg, W. (2015), Bayesian poisson log-bilinear models for mortality projections with multiple populations. *European Actuarial Journal*, **5**(2), 245-281.

Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. (2015), *Time series analysis: forecasting and control*. New York: John Wiley & Sons.

Brouhns, N., Denuit, M., and Vermunt, J. K. (2002), A poisson log-bilinear regression approach to the construction of projected lifetables. *Insurance: Mathematics and economics*, **31**(3), 373-393.

Brownlee, J. (2017), *Long short-term memory networks with python: develop sequence prediction models with deep learning*. Machine Learning Mastery.

Choi, J. (2021), 6-parametric factor model with long short-term memory. *Communications for Statistical Applications and Methods*, **28**(5), 521–536.

Chollet, F. et al. (2015), Keras. https://github.com/fchollet/keras.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014), Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555*.

Farsani, R. M., and Pazouki, E. (2021)., A transformer self-attention model for time series forecasting. *Journal of Electrical and Computer Engineering Innovations (JECEI)*, **9**(1), 1-10.

Géron, A. (2019), *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

Glorot, X., and Bengio, Y. (2010), Understanding the difficulty of training deep feedforward neural networks. *In Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 249-256.

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

Hunt, A., and Blake, D. (2021), On the structure and classification of mortality models. *North American Actuarial Journal*, **25**(1), 215-234.

Hyndman, R. J., Koehler, A. B., Ord, J. K., and Snyder, R. D. (2008), *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media.

Hyndman, R. J., and Khandakar, Y. (2008), Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, **26**(3), 1-22.

Lee, R. D., and Carter, L. R. (1992), Modeling and forecasting us mortality. *Journal of the American statistical association*, **87**(419), 659-671.

Li, J. (2013), A poisson common factor model for projecting mortality and life expectancy jointly for females and males. *Population studies*, **67**(1), 111-126.

Li, N., and Lee, R. (2005), Coherent mortality forecasts for a group of populations: An extension of the lee-carter method. *Demography*, **42**(3), 575-594.

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

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *In International conference on machine learning*, PMLR, 1310-1318.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, Steiner, B., Fang, L., Bai, J. and Chintala, S. (2019), Pytorch: An imperative style, high-performance deep learning library. *In Advances in Neural Information Processing Systems*, Wallach, H., Larochelle, H., Beygelzimer, A., dAlché-Buc, F., Fox, E. and Garnett, R. (Eds.). Curran Associates, Inc., 8024-8035.

Perla, F., Richman, R., Scognamiglio, S. and Wüthrich, M. V. (2021), Time-series forecasting of mortality rates using deep learning. *Scandinavian Actuarial Journal*, 1-27.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing.

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

Richman, R. and Wüthrich, M. V. (2019), Lee and carter go machine learning: Recurrent neural networks. *Available at SSRN 3441030*.

Roshani, A., Izadi, M. and Khaledi, B. (2022), Bayesian poisson common factor model with overdispersion for mortality forecasting in multiple populations. *Submitted*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986), Learning representations by back-propagating errors. *nature*, **323**(6088), 533-536.

Ushey, K. and Allaire, J. and Tang, Y. (2021). reticulate: Interface to Python. *R package version 1.22*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. and Polosukhin, I. (2017), Attention is all you need. *In Advances in neural information processing systems*, 5998-6008.

Villegas, A. M., Kaishev, V. K., and Millossovich, P. (2018), StMoMo: An R package for stochastic mortality modeling. *Journal of Statistical Software*, **84**(3), 1-38.

Wu, N., Green, B., Ben, X., and O'Banion, S. (2020), Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv:2001.08317*.

A Appendix

Let $\{\kappa_t\}_{t=1}^{T}$ be an observed sequential time series data. The aim is to predict κ_t for $t = T + 1, \ldots$, using the LSTM and the transformer networks. Input and output data are required for supervised learning methods in machine learning. Therefore, we need to divide the training data into two parts as displayed in Table 11.

Table 11: Divide the training data into two parts input and output.

Input	Output
$\kappa_1, \kappa_2, \ldots, \kappa_\ell$	$\kappa_{\ell+1}$
$\kappa_2, \kappa_3, \ldots, \kappa_{\ell+1}$	$\kappa_{\ell+2}$
$\vdots$	$\vdots$
$\kappa_{T-\ell}, \kappa_{T-\ell+1}, \ldots, \kappa_{T-1}$	κ_T

The optimum hyperparameter ℓ, the size of the input data, is obtained using MSE criteria. The sample size from the above method is $T - \ell$.

A.1 LSTM

Recurrent neural networks can remember a lot of information about the past and use it to predict the future more accurately. This property is used to analyse sequential data such as time series. RNNs developed in the 1980s (Rumelhart et al., 1986) and have recently become popular due to increasing computing power. RNNs have a hidden state (or memory) and loop to store the output for a given input which is again used as inputs in the next time step. In other words, RNNs consist of a recursive loop that allows information gained from previous time step.

RNNs are comprised of several *cells* connected in a series across a time axis. Figure 2 illustrates a simple RNN architecture with one hidden layer. The right side of the figure shows the unfolding of the network through time. At time step t, the RNN's cell receive input $\mathbf{x}_t$ as well as the previous hidden state, $\mathbf{h}_{t-1}$, update the current hidden state, $\mathbf{h}_t$, and eventually produce output $\tilde{\mathbf{y}}_t$.

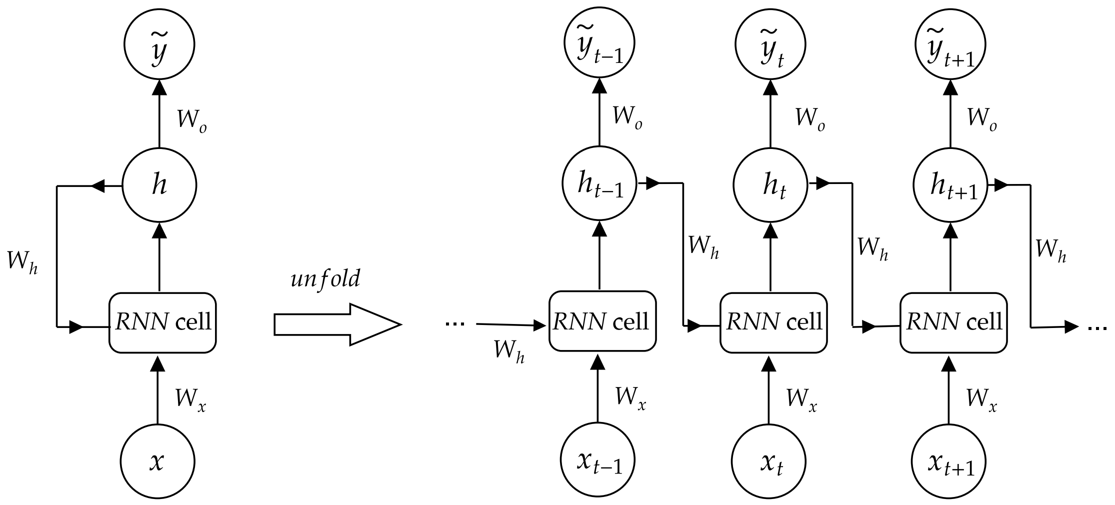

Figure 2: RNN architecture in two forms, folded (left side) and unfolded across time (right side).

The RNN network's hidden state and output in time step t are specified as follows:

$$\begin{aligned}\mathbf{h}_t &= f\left(\mathbf{W}_x\mathbf{x}_t + \mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{b}_h\right),\\ \tilde{\mathbf{y}}_t &= f\left(\mathbf{W}_o\mathbf{h}_t + \mathbf{b}_o\right),\end{aligned}$$

where,

- $\mathbf{x}_t$ is the input vector of size n_I.
- $\tilde{\mathbf{y}}_t$ is the output vector of size n_o.
- $\mathbf{h}_t$ is the hidden state of size n_h.
- $\mathbf{W}_x$ is the $n_h \times n_I$ matrix of connection weights for the inputs.
- $\mathbf{W}_h$ is the $n_h \times n_h$ matrix of connection weights for the previous hidden states.
- $\mathbf{W}_o$ is the $n_o \times n_h$ matrix of connection weights for the current hidden states.
- $\mathbf{b}_h$ is the bias vector of size n_h.
- $\mathbf{b}_o$ is the bias vector of size n_o.

- $f(.)$ is an activation function such as hyperbolic tangent or ReLU.

In general, training means a process that a model learns the optimal parameters with a training set by minimizing a given error function which depends on trainable parameters. Training is composed of three steps, forward propagation, backpropagation and parameter update. At each time step t, forward propagation of RNN updates values of the state $\mathbf{h}_t$, the output $\tilde{\mathbf{y}}_t$ and the corresponding error

$$E_t = \sum_{i=1}^{n_o} \left\{ (\tilde{\mathbf{y}}_t)_i - (\mathbf{y}_t)_i \right\}^2 ,$$

with respect to the input $\mathbf{x}_t$ and the target $\mathbf{y}_t$, where $(\mathbf{y}_t)_i$ is the ith component of the vector $\mathbf{y}_t$. Note that the parameters of weights $\mathbf{W}_x, \mathbf{W}_h, \mathbf{W}_o$ and biases $\mathbf{b}_h, \mathbf{b}_o$ remain unchanged during the forward propagation. The initial state $\mathbf{h}_0$ and the initial values of weights and biases are required to compute the first state $\mathbf{h}_1$ and consequently the output value $\tilde{\mathbf{y}}_\mathbf{1}$. The initial state and biases are usually set to zero, and the initial weights are determined by the Glorot uniform initializer (Glorot and Bengio, 2010). In backpropagation step, the gradients (partial derivatives) of the error function

$$E = \frac{1}{T} \sum_{t=1}^{T} E_t = \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{n_o} \left\{ (\tilde{\mathbf{y}}_t)_i - (\mathbf{y}_t)_i \right\}^2$$

are computed with respect to the learning parameters $\mathbf{W}_x, \mathbf{W}_h, \mathbf{W}_o, \mathbf{b}_h$ and $\mathbf{b}_o$. The chain rule is applied to compute gradients and then used to updates parameters with learning rate γ. For example, for parameter $\mathbf{W}_x$,

$$\mathbf{W}_\mathbf{x} \longleftarrow \mathbf{W}_x - \gamma \frac{\partial E}{\partial \mathbf{W}_x}.$$

Hochreiter and Schmidhuber (1997) introduced LSTM networks which are special cases of RNNs. They are proficient in considering long-term dependencies of a sequential data. The diagram of the unfolded LSTM network across time is shown in Figure 3. The cell state, $\mathbf{C}_t$, conveys the processed information so far to the next cell, which acts like state $\mathbf{h}_t$ in simple RNN. As the cell state does not have any activation functions, it is less influenced by the vanishing or exploding gradient caused by the product of partial derivatives in the learning procedure.

The forget gate, input gate, and output gate are the three gates that make up the LSTM cell. A sigmoid layer plus a point-wise multiplication operation make up the

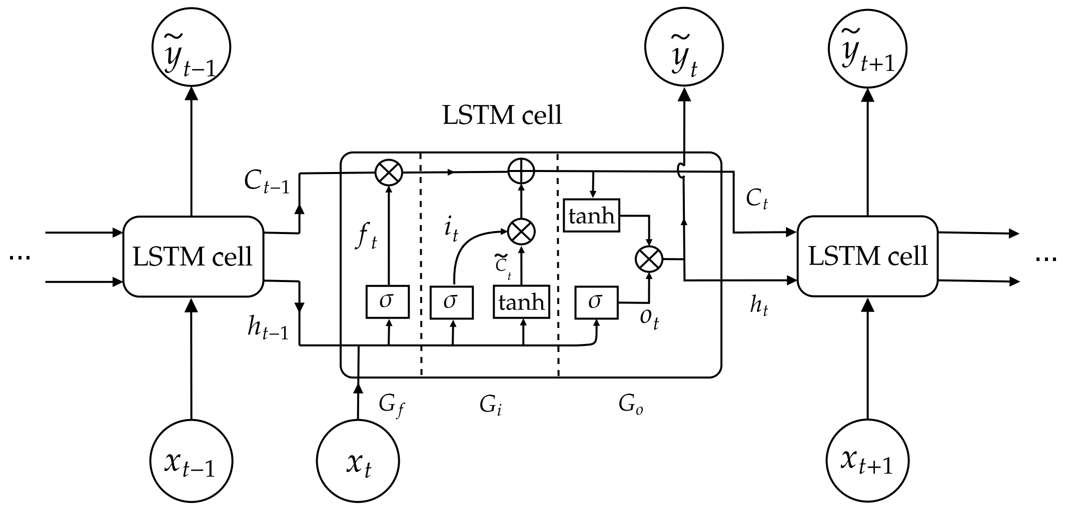

Figure 3: LSTM diagram. The gates are separated by a vertical dashed line.

gates. The sigmoid layer generates numbers between 0 and 1 that indicate how much of each component should be permitted to pass.

The gates in the time step t are specified as follows:

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_{xf}^\top \mathbf{x}_t + \mathbf{W}_{hf}^\top \mathbf{h}_{t-1} + \mathbf{b}_f\right), \qquad \text{(forget gate)}$$

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_{xi}^\top \mathbf{x}_t + \mathbf{W}_{hi}^\top \mathbf{h}_{t-1} + \mathbf{b}_i\right),$$

$$\tilde{\mathbf{C}}_t = \tanh\left(\mathbf{W}_{xc}^\top \mathbf{x}_t + \mathbf{W}_{hc}^\top \mathbf{h}_{t-1} + \mathbf{b}_c\right), \qquad \text{(input gate)}$$

$$\mathbf{C}_t = \mathbf{f}_t \otimes \mathbf{C}_{t-1} + \mathbf{i}_t \otimes \tilde{\mathbf{C}}_t,$$

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_{xo}^\top \mathbf{x}_t + \mathbf{W}_{ho}^\top \mathbf{h}_{t-1} + \mathbf{b}_o\right),$$

$$\tilde{\mathbf{y}}_t = \mathbf{h}_t = \mathbf{o}_t \otimes \tanh(\mathbf{C}_t). \qquad \text{(output gate)}$$

where

- $\mathbf{x}_t$ is the input vector of size n_I.
- $\tilde{\mathbf{y}}_t$ is the output vector of size n_o.
- $\mathbf{W}_{xf}$, $\mathbf{W}_{xi}$, $\mathbf{W}_{xc}$, and $\mathbf{W}_{xo}$ are the $n_I \times n_h$ matrices of connection weights for the inputs in the forget gate, input gate, candidate state, and output gate, respectively.

- $\mathbf{W}_{hf}$, $\mathbf{W}_{hi}$, $\mathbf{W}_{hc}$, and $\mathbf{W}_{ho}$ are the $n_h \times n_h$ matrices of connection weights for previous hidden states in the forget gate, input gate, candidate state, and output gate, respectively.
- $\mathbf{W}_p$ is the $n_h \times n_o$ matrix of connection weights for the current hidden states.
- $\mathbf{b}_f$, $\mathbf{b}_i$, $\mathbf{b}_c$, and $\mathbf{b}_o$ are the bias vectors of size n_h.
- $\mathbf{b}_p$ is the bias vector of size n_o.
- $\otimes$ stands for the element-wise product operator.

A.2 Transformer

Vaswani et al. (2017) presented the transformer network for NLP for the first time. The transformer is a deep learning architecture that avoids the vanishing gradient problem that plagues RNNs (Pascanu et al., 2013) and uses self-attention mechanisms to capture the long-term dependencies. The transformer has an encoder-decoder structure. Both the encoder and the decoder use stacked self-attention and point-wise completely connected layers, as shown in Figure 4. The decoder receives previous outputs and the encoded input from the encoder to generate the output. Because the transformer network has been proposed initially for translation tasks, the input of the encoder and decoder section is a series of words.

Thus, it needs an embedding layer to convert the words into numbers. In this paper, the values of a time-series data, which are numbers, are given to the transformer network. Thus, the embedding layer is removed in the encoder and decoder sections. In sequential data, an element's position is essential in predicting future values. Thus, the sequence is given to a positional encoding layer before feeding the encoder and decoder sections. Followed by the original paper (Vaswani et al., 2017), we use the sinusoidal function in this layer.

Encoder: The first stage is the multi-head self-attention block. The output position information from positional encoding is added to the features before feeding the multi-head self-attention mechanism. The basic idea behind self-attention is to develop an attention mechanism that allows any element in a sequence to attend to any other. In a self-attention mechanism, three different copies of each input are created by multiplying with three weight matrices, *query*, *key*, and *value*, learned through the training process.

The attention value from element i to element j is based on the dot product attention which is defined by $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$, where d_k is the hidden dimensionality for queries $Q \in R^{d \times d_k}$ and keys $K \in R^{d \times d_k}$ and softmax is the well known softmax function. Instead of performing single self-attention, multi-head attention, i.e., multiple different query, key, and value triples on the same sequence, performs to capture multiple various aspects of sequence elements. Suppose that m head attentions apply on the same sequence, then the heads are concatenated and combined with a final weight matrix as follows

$$\text{Multihead}(Q, K, V) = \text{Concat}(\text{Head}_1, \ldots, \text{Head}_m)W^o,$$

where

$$\text{Head}_i = \text{Attention}(Q_i, K_i, V_i), \quad i = 1, \ldots, m.$$

The next layer is an add and normalization layer. In this layer, the output vector of the multi-head attention block is added to the original input of the encoder section, denoted by $\mathbf{x}$. Then, the layer normalization of the vector $\mathbf{x}$ is given by

$$\text{LayerNorm}(\mathbf{x}) = \gamma \frac{\mathbf{x} - \mu}{\sigma} + \beta,$$

where μ and σ are the mean and standard deviation of elements of $\mathbf{x}$, respectively. The scale parameter γ and bias parameter β are learned through the training process. In the next stage, a fully connected feed-forward network (FFN) is applied to each position with a ReLU activation function. This layer consists of two linear transformations as follows:

$$\text{FFN}(x) = \max\{0, xW_1 + b_1\}W_2 + b_2$$

where W_1, W_2, b_1 and b_2 are learned parameters. The FFN is followed by an add and norm layer, which is the final stage of the encoder section. The output of the encoder is fed to the decoder.

Decoder: Similar to the encoder section, after the positional encoding, the first layer is multi-head attention, followed by an add and norm layer. The second multi-head attention takes the output of the encoder block and makes the linear transformation's key and value in the self-attention mechanism. The third linear transformation, i.e., the query, is made from the output of the add and norm layer. The decoder section is finished by an add and norm layer, a fully connected feed-forward network with a ReLu activation function, and another add and norm layer.

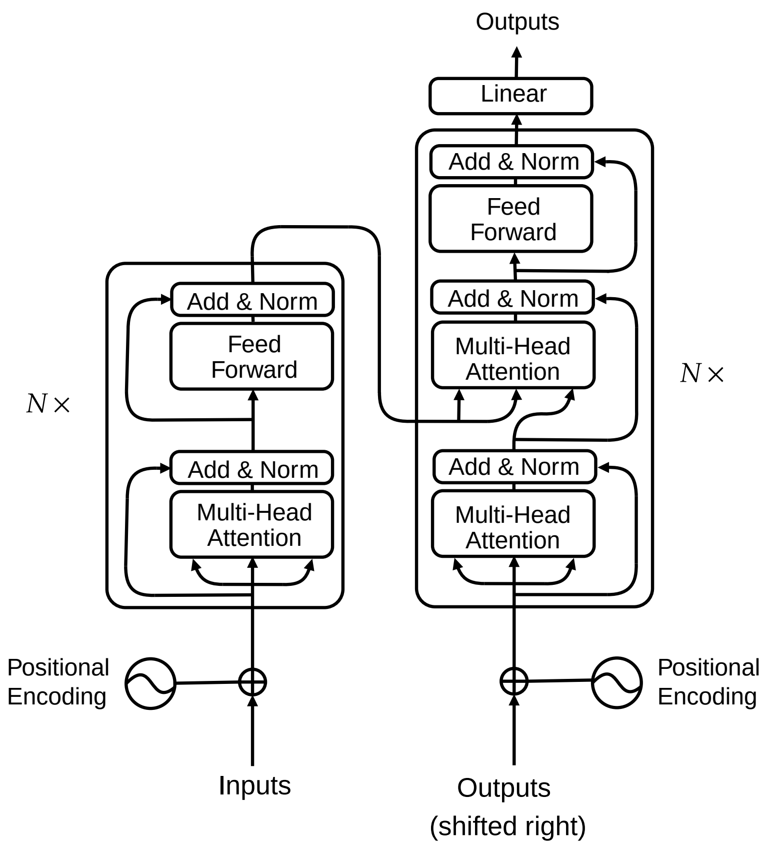

Figure 4: Transformer Architecture.