1949

Deep Learning for Time Series Forecasting with Applications to Finance

Thesis for the Degree of Doctor of Philosophy (PhD)

Petneházi Gábor

Supervisor: Dr. Gáll József Mihály

UNIVERSITY OF DEBRECEN
Doctoral Council of Natural Sciences and Information Technology
Doctoral School of Mathematical and Computational Sciences
Debrecen, 2021

Hereby I declare that I prepared this thesis within the Doctoral Council of Natural Sciences and Information Technology, Doctoral School of Mathematical and Computational Sciences, University of Debrecen in order to obtain a PhD Degree in Natural Sciences at the University of Debrecen.

The results published in the thesis are not reported in any other PhD theses.

Debrecen, 2021-06-01

. . . . . .
Petneházi Gábor

Hereby I confirm that Petneházi Gábor candidate conducted his studies with my supervision within the Probability Theory, Statistics and Applied Mathematics Doctoral Program of the Doctoral School of Mathematical and Computational Sciences between 2016 and 2021. The independent studies and research work of the candidate significantly contributed to the results published in the thesis.

I also declare that the results published in the thesis are not reported in any other theses.

I support the acceptance of the thesis.

Debrecen, 2021-06-01

. . . . . .
Dr. Gáll József Mihály
supervisor

Deep Learning for Time Series Forecasting with Applications to Finance

Dissertation submitted in partial fulfilment of the requirements for the doctoral (PhD) degree in mathematics and computing

Written by Petneházi Gábor certified economist

Prepared in the framework of the Doctoral School of Mathematical and Computational Sciences of the University of Debrecen
(Probability Theory, Statistics and Applied Mathematics programme)

Dissertation advisor: Dr. Gáll József Mihály, Associate Professor

The official opponents of the dissertation:

Dr. ....
Dr. ....

The evaluation committee:

chairperson:	Dr.	.....
members:	Dr.	.....
	Dr.	.....
	Dr.	.....
	Dr.	.....

The date of the dissertation defence: .....

Acknowledgements

I am grateful to my supervisor for always guiding me without clipping my wings. Thanks to my wife for the LaTeX template. Special thanks to my little baby for letting me sleep at night.

Contents

Introduction

Knowing the future is one of the main desires of humanity. It is tempting because it is unattainable. This is the reason for my choice of research topic. Predictions can be made in many different ways: from the crystal ball to—well, what is the other end of the scale? I would argue that it is machine learning: the science (and art) of enabling computers to learn from data. I think that machines are much better in forecasting than humans.

The quickly evolving field of machine learning delivers powerful tools for time series forecasting. Certain deep neural network architectures are particularly well-suited to sequential data (such as time series). However, there is a big obstacle: machine learning models (and deep learning models in particular) are hungry for data. While many data science projects involve large amounts of data, time series forecasting is typically not one of them. Only time can generate new data, and we can not urge it. High complexity may do more harm than good without sufficient data—we need to find the right balance. This is what makes deep-learning based time series forecasting an exciting field.

During my years as a PhD student, I have been working on finance-related forecasting problems. Finance offers challenging forecasting problems, together with datasets of considerable size. Furthermore, financial forecasts can be of great practical significance. This does not only mean that our potential findings may be valuable, but also that the field is intensively researched, and that there are very strong benchmarks. All these properties make finance a good area of application for our deep learning-based forecasting efforts.

This thesis consists of two main parts: methods and results. That is, the next chapter covers the relationship between time series forecasting and deep learning, and the one after that presents the results of our empirical

studies.

Publications This section lists the articles on which this thesis is based. Some of them have been published, others are available as manuscripts. *Recurrent Neural Networks for Time Series Forecasting* [Petneházi, 2019b] covers a large part of the methodology that was applied in our empirical studies. *Exploring the predictability of range-based volatility estimators using recurrent neural networks* [Petneházi and Gáll, 2019a] is a comparative study on the forecastability of different volatility estimates. *Volatility Forecasting with 1-dimensional CNNs via transfer learning* [Aradi et al., 2020] explores if jointly trained neural networks can produce competitive volatility forecasts. *Quantile Convolutional Neural Networks for Value at Risk Forecasting* [Petneházi, 2019a] suggests a convolutional neural network-based method for forecasting Value-at-Risk. *Evaluating the Lee-Carter model on Hungarian mortality data* [Petneházi and Gáll, accepted] describes an application of the mortality forecasting Lee-Carter model, while *Mortality rate forecasting: can recurrent neural networks beat the Lee-Carter model?* [Petneházi and Gáll, 2019b] uses recurrent neural networks to produce long-term mortality rate forecasts for several countries of the world.

Chapter 1

Methods

Machine learning is the field that aims to enable computer programs to automatically improve with experience [Mitchell, 1997]. That is, the aim is to build algorithms that can learn from data, without being explicitly programmed. The use of machine learning in time series forecasting is not a novelty. Actually, most quantitative forecasting algorithms are simple applications of machine learning. (Think of autoregressive models, for example.) However, machine learning is a large field, and some parts of it have not yet been fully exploited by time series analysts.

This thesis focuses on the time series forecasting applications of one particular subfield of machine learning: deep neural networks. The following sections describe the basics of deep learning, the most suitable model architectures for time series forecasting, and some important considerations regarding the application of deep learning to time series.

1.1 Deep Neural Networks

Deep learning means machine learning with deep neural networks—the whole field, with all its amazing results, is built upon the relatively simple concept of (artificial) neural networks.

Neural networks are inspired by the brain. A neural network consists of (artificial) neurons, which are connected. The connections transmit signals. The neurons compute some functions of the incoming signals. The whole point is to find appropriate weights for these connected functions, so that the final outputs of the network are close to the desired target values.

A simple neural network is illustrated in Figure 1.1. The circles are the neurons (or nodes, or units). These neurons compute the weighted sum of the input values, and apply some (activation) function to the sum. The formula

$$a = f(Wx + b) \tag{1.1}$$

describes a neural network layer (a: activation vector, x: input values, f: activation function, W: weight matrix, b: bias vector). During training, we aim to find the optimal (that is, loss-minimizing) weights W. It is usually performed using the backpropagation algorithm [Werbos, 1994]. Backpropagation computes the gradients of the loss function with respect to the weights, using the chain rule. Having the gradients, gradient descent (or one of its several variants) can be used to optimize the weights. Backpropagation is mathematically simple and computationally efficient—this makes the seemingly difficult neural networks fairly simple to apply. We have to make several choices (regarding the size of the network, the activation functions, the optimizer, etc.) to build the model, but the training process is simple.

Deep Neural Networks Deep learning uses deep neural networks—the (shallow) neural network in Figure 1.1 has a single hidden layer, a deep neural network has multiple hidden layers.

While the underlying mechanisms are simple, the resulting (deep) networks are usually complex. There are so many weights and connections that it is difficult to understand the reasons behind the network's decisions. Therefore, deep neural networks are usually considered black box algorithms. It is a clear disadvantage, especially for time series forecasting, where the interpretability of the models has a high importance.

Having multiple hidden layers rather than a single large hidden layer (that is, using a deep rather than a shallow network) is advantageous in several respects. For example, it can achieve similar complexity with much fewer parameters, and it can learn hierarchical relationships. Furthermore, there are some special deep neural network architectures that work particularly well for certain types of data.

The interested reader may find a short summary of deep learning in LeCun et al. [2015], a comprehensive introduction in Nielsen [2015], or a very detailed description in Goodfellow et al. [2016].

The neural network architecture displayed in Figure 1.1 is the so-called feedforward neural network, meaning that the signals go forward, and not

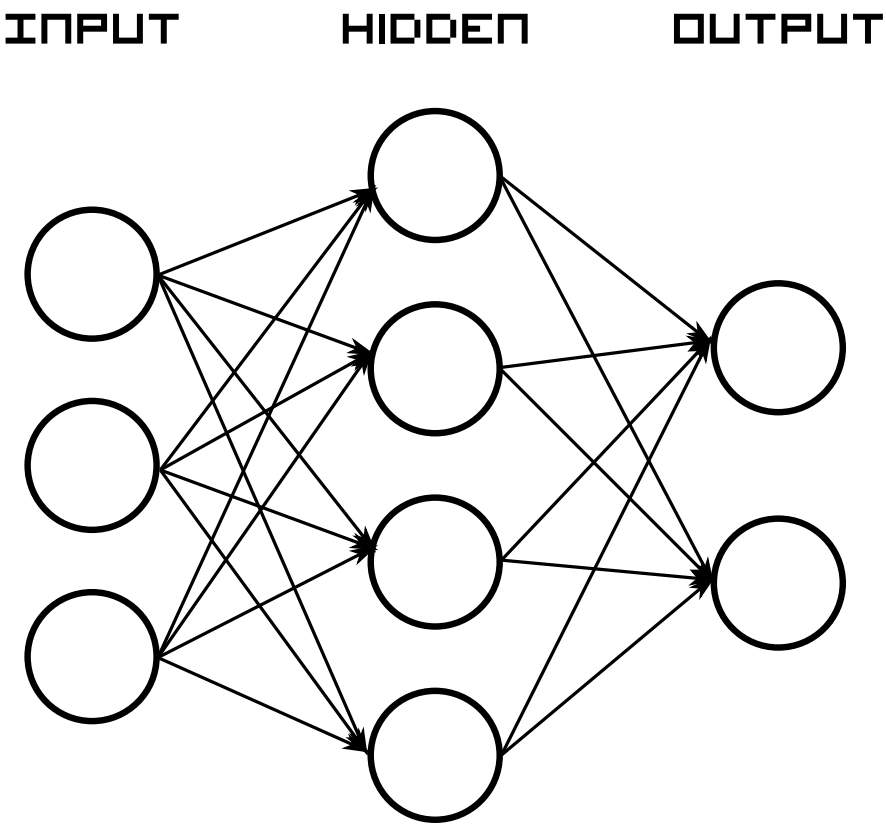


Figure 1.1. (Feedforward) Neural Network

backward. This is the basic model, but there are further architectures, some of which can model time series data especially well. While we could apply feedforward networks to forecasting (for example, in an autoregressive fashion), it is more advantageous to choose a network that is inherently well-suited to time series data. The following sections describe two such networks: recurrent neural networks and convolutional neural networks.

1.1.1 Recurrent Neural Networks

Recurrent neural networks (RNN) are neural networks with temporal connections—with the ability to learn not only from the current input values, but also from previous values returned from the system [Elman, 1990,

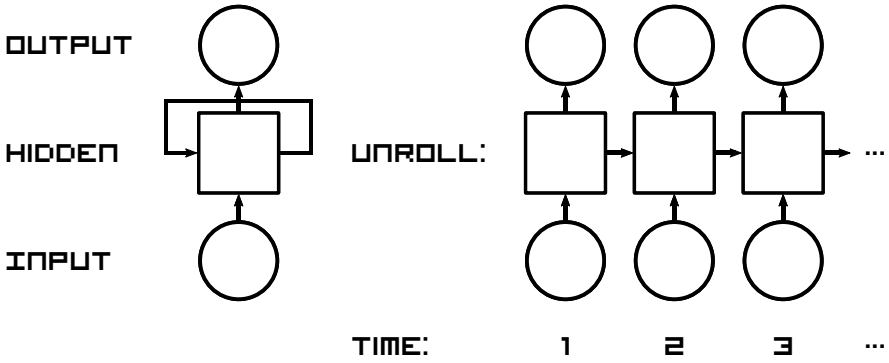


Figure 1.2. (Simple) Recurrent Neural Network

Jordan, 1997]. This temporal nature makes them a good choice for sequence learning problems.

A recurrent neural network architecture is displayed in Figure 1.2. It is not much different from a feedforward network, however, it has some temporal connections. The formula

$$h_t = f(Wx_t + Uh_{t-1} + b) \tag{1.2}$$

shows the maths behind a (simple) recurrent layer (t: time, h: hidden state vector, x: input vector, f: activation function, W, U and b: weights). It uses both the current input x and the previous state h to compute the new state. It is a simple and intuitive structure, which could work well with sequential data. However, in practice, it does not work well. The main problem is that such a simple recurrent neural network can't have a long-term memory. While it could be trained by applying backpropagation through time, we would need a large network (with many temporal connections) in order to have a long lasting memory. These long connections are not beneficial for gradient descent-based optimization techniques—the vanishing gradient problem frequently occurs. So, simple recurrent neural networks can't be trained as easily as feedforward neural networks, and therefore they are rarely applied in practice.

Gated Recurrent Neural Networks There is another (and somewhat more complex) recurrent neural network structure that solves the problem of long-term memory. The core concept is to build simple functions to manage

the content of the memory. We want to write to the memory, read from the memory, and sometimes delete things from the memory. These functions are managed by so-called gates. The gates are continuous functions taking on values from 0 to 1. 1 means that the gate is fully open, 0 means that the gate is fully closed. However, since they are continuous, the gates can be partially open. The gates are also differentiable so that the backpropagation works. This is very important: the usual training procedure can be applied, the gates can be trained together with the whole network. It means that even though the gated structure looks a bit complex, it is as easily applicable as any other neural network.

Essentially all commonly applied recurrent neural networks are gated architectures. The following sections describe two popular recurrent units of this type: Long Short-Term Memory, and Gated Recurrent Unit.

Long Short-Term Memory

Long short-term memory (LSTM) was proposed by Hochreiter and Schmidhuber [1997]. The formulas

$$\begin{aligned} i_t &= sigmoid\left(W_i x_t + U_i h_{t-1} + b_i\right), \\ f_t &= sigmoid\left(W_f x_t + U_f h_{t-1} + b_f\right), \\ o_t &= sigmoid\left(W_o x_t + U_o h_{t-1} + b_o\right), \\ c_t &= f_t \odot c_{t-1} + i_t \odot tanh\left(W_c x_t + U_c h_{t-1} + b_c\right), \\ h_t &= o_t \odot tanh\left(c_t\right) \end{aligned} \tag{1.3}$$

show the equations of LSTM (i: input gate vector, f: forget gate vector, o: output gate vector, c: cell state vector, h: hidden state vector, x: input vector). $\odot$ denotes the element-wise product.

LSTM is an RNN cell with 3 gates. The input (i_t) and the forget (or rather remember) (f_t) gates control the cell state or long-term memory (c_t). The forget gate keeps something from the previous step's cell state, and the input gate adds something to it. The output gate (o_t) produces the hidden state or output vector (h_t), which is a focused memory. That is, it decides what to use immediately from the long-term memory.

The input, output, and forget gates use the logistic activation function

$$sigmoid(x) = \frac{1}{1 + exp(-x)} \tag{1.4}$$

($x \in \mathbb{R}$), which produces values between 0 and 1.

These activations allow for the (continuous) gating mechanism. The computation of the states involves the use of the hyperbolic tangent activation function

$$tanh(x) = \frac{exp(x) - exp(-x)}{exp(x) + exp(-x)} \tag{1.5}$$

($x \in \mathbb{R}$), which produces values between -1 and 1.

Gated Recurrent Unit

Cho et al. [2014] proposed a very similar, yet somewhat simpler architecture: gated recurrent unit (GRU). The formulas

$$\begin{aligned} z_t &= sigmoid\left(W_z x_t + U_z h_{t-1} + b_z\right), \\ r_t &= sigmoid\left(W_r x_t + U_r h_{t-1} + b_r\right), \\ h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot tanh\left(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h\right) \end{aligned} \tag{1.6}$$

show the equations of GRU (z: update gate vector, r: reset gate vector, h: hidden state vector).

GRU has fewer gates and weights than LSTM. There is no output gate, so there is no control over the use of memory. The update gate (z_t) is responsible for both the remaining and new information in the output vector (h_t). The reset gate (r_t) is placed into the candidate activation: it controls the previous values of the hidden state vector when computing the new memory content.

Overall, it is structurally similar to LSTM, but there are some important operational differences, and it is somewhat smaller (and faster). Yet, just by looking at the equations, it is very hard to argue for any one of them. It is difficult to see if either recurrent unit would perform better than the other for the given problem. Chung et al. [2014] empirically evaluated the performance of LSTM and GRU on different sequence modeling tasks and different datasets. They found that both gated units are clearly superior to the simple recurrent unit, but it is diffcult to decide if LSTM or GRU is better.

RNNs for Forecasting

Gers et al. [2002] used LSTM as a ‘pure‘ autoregressive model, and compared it to a feedforward neural network trained on time windows. That is,

the LSTM was given a single input at a time, while the feedforward network saw several successive observations in each step. On time series that can be solved by looking at the past few values, the feedforward network outperformed the long short-term memory. LSTM's main strength is its long-term memory, which was not necessary in these cases. For this reason, the authors suggest to apply LSTM to time series forecasting only when traditional approaches fail.

Guo et al. [2016] used LSTM with an adaptive gradient learning method to forecast streamed data. Fu et al. [2016] applied both LSTM and GRU to a challenging forecasting task that cannot be described by linenar models. Qin et al. [2017] proposed a recurrent neural network width dual-stage attention, that uses an encoder with input attention to select relevant features, and a decoder with temporal attention to learn long-term dependencies. Cinar et al. [2017] used RNN with an extended attention mechanism to forecast univariate and multivariate time series.

Bandara et al. [2020] trained LSTM networks on groups of similar time series specified by clustering techniques. Wang et al. [2019] applied LSTM to probabilistic forecasting using pinball loss. Salinas et al. [2020] proposed an autoregressive recurrent neural network model trained jointly on multiple related time series for producing probabilistic forecasts.

1.1.2 Convolutional Neural Networks

The previously discussed basic (feedforward) network structure is also called fully connected, which means that each neuron in one layer is connected to all neurons in the next layer. Convolutional neural networks (CNN), on the other hand, share weights in different areas of the input data. A convolutional layer takes only a small number of weights, and slides them through the data. This weight-sharing helps avoid overfitting, and it can find the same patterns in different locations, which makes it particularly useful for modeling certain types of data.

While recurrent neural networks share parameters in time, convolutional neural networks share parameters in space. They are most often applied to images. The spatial weight-sharing enables them to identify features regardless of their position (translation invariance), which has led to major breakthroughs in computer vision. Russakovsky et al. [2015] compared human and computer accuracy on a difficult image classification task, and found that humans need significant training to achieve competitive performance

against a state-of-the-art convolutional neural network. The enormous success of CNNs in image processing, however, does not mean that they cannot be applied to other data. While two-dimensional convolutional networks can be applied to images, one-dimensional CNNs can be applied to, for example, time series.

In fact, the very first CNNs were used for modeling temporal data. The so-called time delay neural networks [Lang et al., 1990] were applied to speech recognition. Variables spatially or temporarily nearby are often correlated [LeCun et al., 1995]. This is why extracting local features can be exceptionally useful. Either for images or time series.

Causal Convolutions Time series applications require causal convolutions. It means that the outputs at any point in time should not depend on future inputs. The violation of this requirement would imply a serious data leakage.

Dilated Convolutions Simple convolutional neural networks, just like the recurrent architectures, struggle with old memories. Here we can solve the problem by applying convolutions with holes: dilated convolutions [Yu and Koltun, 2015]. Dilations can make the filter larger by skipping a few steps, that is, by dilating it with zeros. The dilation rate of the l^{th} convolutional layer is usually set to 2^{l-1}, which allows an exponential growth in the receptive field. By increasing the number of layers, we can exponentially increase the size of history used for predictions.

The formula

$$a_i = f\left(b + \sum_{j=0}^{s} w_j x_{i+j}\right) \tag{1.7}$$

describes a one-dimensional convolution, the formula

$$a_i = f\left(b + \sum_{j=0}^{s} w_j x_{i+d \cdot j}\right) \tag{1.8}$$

describes a one-dimensional dilated convolution, the formula

$$a_i = f\left(b + \sum_{j=0}^{s} w_j x_{i-d \cdot j}\right) \tag{1.9}$$

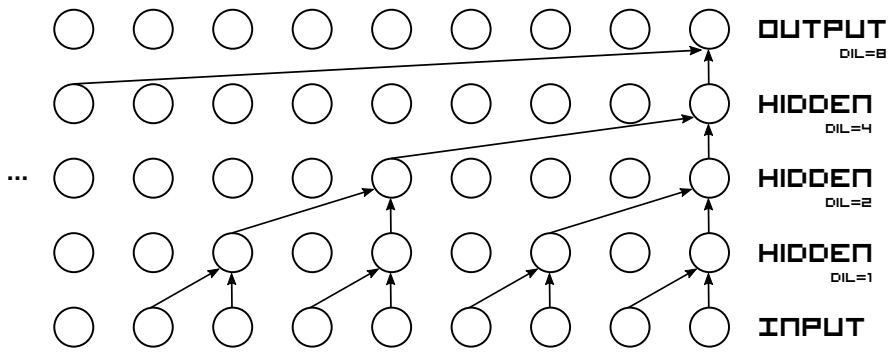


Figure 1.3. Dilated Causal Convolutional Neural Network

describes a one-dimensional dilated causal convolution (a_i: ith activation, x: input vector, s: number of shared weigths; w,b: shared weights and bias; f: activation function; d: dilation rate)—though the exact computation depends on a few parameter choices. A dilated causal convolution is displayed in Figure 1.3.

CNNs for Forecasting

The WaveNet architecture used dilated causal convolutions for generating audio waveforms [Oord et al., 2016]. Although not time series forecasting, it was an excellent solution to a similar sequential learning problem, which soon spilled over. Borovykh et al. [2017] used an adaptation of the WaveNet for time series forecasting, and claims that it is a time-efficient alternative to recurrent neural networks. Chen et al. [2020] used a dilated causal CNN to produce probabilistic forecasts. Wen et al. [2017] used convolutional and recurrent networks to produce multi-step probabilistic forecasts.

1.2 Deep Time Series Forecasting

The following subsections discuss some important considerations regarding the application of deep learning to time series forecasting. Full-featured time series forecasting has several requirements which are not usually fulfilled by machine learning.

1.2.1 Uncertainty

Unlike traditional time series forecasting, machine learning (more precisely, supervised learning) usually aims to produce point predictions only. Deep learning models, for example, optimize a predefined loss function—the aim is to find values closest to the target (where the distance between the predicted and true values is measured by the chosen loss function). In this case we try to predict a single point of the distribution, typically, the mean.

In case of time series forecasting, producing a single best guess is usually just part of the solution. The future always holds uncertainty, and our forecasts should somehow reflect this uncertainty. This is why the package should include interval forecasts, too.

Prediction intervals contain future values with a predefined probability. That is, they do not give a best guess for the future value, but a highly probable range. Prediction intervals can make forecasts more meaningful for time series professionals. The problem is, unlike some traditional time series models, most machine learning and deep learning models do not produce prediction intervals. They operate as black boxes—we don't really know how they produce their predictions, and we don't know either how confident they are.

Although our deep learning algorithms, by default, produce point predictions only, they may be extended to produce interval forecasts, too. The remainder of this section describes two possible methods: bootstrapping and quantile regression.

Bootstrapping

There is a simple yet computationally intensive method to produce prediction intervals for literally any regression algorithm: bootstrapping. It was proposed by Efron et al. [1979] for estimating the quality of estimates. We take samples with replacement from the dataset, and make a separate forecast using each sample. The variability of these forecasts gives a measure of the confidence of the estimates.

Efron and Tibshirani [1986] argue that the bootstrap has nice properties for estimating standard errors, and its computational intensity is less and less a problem as computation is getting cheaper and faster. Also, bootstrap aggregating (bagging) [Breiman, 1996] is an ensemble method [Dietterich,

2002] which can help stabilize the predictions

$$\hat{y}_i = \frac{1}{n_{run}} \sum_{run=1}^{n_{run}} \hat{y}_i^{run}. \tag{1.10}$$

We use the following notation: $\hat{y}_i^{run}$ is the estimate of the ith obervation (y_i) with the runth bootstrap sample, $\hat{y}_i$ is the bagged estimate of y_i, and n_{run} is the number of bootstrap samples.

Most time series are not too long (small data), which makes bootstrapping affordable and makes bagging reasonable. Thus, for time series forecasting problems, the use of bootstrapping seems perfectly applicable, using any learning algorithm (including neural networks).

Bootstrapping was used for producing confidence and prediction intervals for neural networks [Paass, 1993, Heskes, 1997, Carney et al., 1999, Khosravi et al., 2011].

Confidence intervals measure how accurately we can approximate the true regression, that is, the mean of the target distribution. The center of the interval is the mean of the bootstrap estimates (1.10). It is the bagged estimator. Using the variance ($\sigma^2_{\hat{y}_i}$) of these estimates

$$\sigma^2_{\hat{y}_i} = \frac{1}{n_{run} - 1} \sum_{run=1}^{n_{run}} (\hat{y}_i^{run} - \hat{y}_i)^2, \tag{1.11}$$

and an appropriate $conf$-quantile [t_{conf}, $conf \in (0.5, 1)$] of the t-distribution (with degrees of freedom equal to the number of bootstrap samples, n_{run}, see Heskes [1997]), we can compute the confidence interval

$$CI_i = [\hat{y}_i - t_{conf} \cdot \sigma_{\hat{y}_i},\ \hat{y}_i + t_{conf} \cdot \sigma_{\hat{y}_i}]. \tag{1.12}$$

Prediction intervals measure how well we can approximate the target values. They are more useful, but they are also more difficult to estimate. We need to estimate the noise variance ($\sigma^2_{\hat{\epsilon}}$) of the regresssion

$$\sigma^2_{\hat{\epsilon}} \simeq E[(y - \hat{y})^2] - \sigma^2_{\hat{y}}. \tag{1.13}$$

The true values are denoted by y.

Thus, we need to train a separate learning model to predict the remaining residual (r_i) of observation i

$$r_i^2 = \max((y_i - \hat{y}_i)^2 - \sigma^2_{\hat{y}_i}, 0). \tag{1.14}$$

This noise is assumed more or less Gaussian [Heskes, 1997]. The model can be trained on a separate validation set, or on the observations randomly left out of bootstrap samples. The output activation is exponential, so that we only predict positive variances, and the loss function is the negative log-likelihood [Heskes, 1997, Khosravi et al., 2011]

$$L = \frac{1}{2}\sum_{i=1}^{n}(ln(\sigma_{\hat{\epsilon}_i}^2) + \frac{r_i^2}{\sigma_{\hat{\epsilon}_i}^2}). \tag{1.15}$$

The 2 variances are added together

$$\sigma_i^2 = \sigma_{\hat{y}_i}^2 + \sigma_{\hat{\epsilon}_i}^2, \tag{1.16}$$

which finally yields the prediction intervals

$$PI_i = [\hat{y}_i - t_{conf} \cdot \sigma_i,\ \hat{y}_i + t_{conf} \cdot \sigma_i]. \tag{1.17}$$

Quantile Regression

Another possible approach to producing interval forecasts is to directly estimate quantiles of the distribution. Machine learning algorithms typically predict the mean. However, we may be interested in other points of the distribution as well—for example, quantiles. By predicting not the mean, but the median, we can produce more robust forecasts. (That is, a few extreme values cannot drag it away as easily.) The median may be a better representation of a typical value than the mean—but it is still just a single value. We can get a much better description of the distribution by forecasting different quantiles. We can do this, using quantile regression [Koenker and Hallock, 2001]. Sometimes we are not particularly curious about the expectation, but rather about an optimistic or pessimistic scenario—in such cases, we can forecast upper or lower quantiles, accordingly. Or, we can compute the difference between the two to get interval forecasts—with much less computation than using, for example, bootstrapping.

In deep learning, we can predict the mean by using the mean squared error loss function

$$\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}. \tag{1.18}$$

We can use the mean absolute error loss

$$\frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n} \tag{1.19}$$

to estimate the median.

For predicting arbitrary θ-quantiles ($\theta \in [0, 1]$), we can use the so-called tilted (or pinball) loss function

$$\theta \sum_{y_i \geq \hat{y}_i} (y_i - \hat{y}_i) + (\theta - 1) \sum_{y_i < \hat{y}_i} (y_i - \hat{y}_i). \tag{1.20}$$

That is, applying deep learning models to quantile regression is just a matter of choosing loss function.

1.2.2 Feature Engineering

This section covers two main components of feature engineering for machine learning: creating features and evaluating features. The first one means constructing input variables that the model can easily learn from—it is for improving the model. The second one means measuring the importance of the features—it is for understanding the model.

Feature Creation

Feature engineering is about producing input features for the machine learning model, so that it can learn conveniently. The truth is, deep neural networks do not reqiure much feature engineering.

On the one hand, Domingos [2012] argues that feature engineering is key to the success of machine learning projects. If the output is a complex function of the (raw) features, the model may not be able to learn it. For this reason, feature construction is usually very important, and data processing often takes more time than doing machine learning.

On the other hand, Bengio et al. [2007] argue that the goal of machine learning is to produce methods that can solve highly complex tasks, and this can be achieved with algorithms that can learn high complexity with minimal human intervention. The authors claim that deep architectures can progressively combine lower level features into more abstract representations.

Deep neural networks require less feature engineering than most machine learning algorithms. Still, some computed features may prove useful, especially in a time series forecasting context.

Probably the most simple and self-evident feature engineering step is to turn our model into autoregression. That is, to use lagged values of the output as input variables. While recurrent neural networks and convolutional neural networks have an inherent ability to learn sequentiality, it may still be advantageous to use the past few steps' values as inputs.

While traditional time series forecasting methods usually aim to remove the trend, and fit a model to the detrended time series, machine learning models can learn the trend during model fitting. For example, we can learn a (linear) trend by producing an input variable of equidistant increasing values representing the passage of time. We can also engineer features to model seasonality. We have (at least) two options. We can encode different periods using individual dummy variables (one-hot-encoding). It is simple and easily applicable, but it has its drawbacks. The more periods we have the more variables we need (e.g. encoding the day of year could be pretty inconvenient). Also, it cannot grab the cyclical nature of seasonality (i.e., the distances of the periods). Another popular method encodes seasonality into continuous, cyclical variables. We can transform the variables using the $sine$

$$\tilde{x_i} = \sin\left(\frac{2 \cdot \pi \cdot x_i}{\max(x)}\right) \tag{1.21}$$

and the $cosine$

$$\tilde{x_i} = \cos\left(\frac{2 \cdot \pi \cdot x_i}{\max(x)}\right) \tag{1.22}$$

($x_i \in \mathbb{R}, max(x) \neq 0$) transformations—we should use both to have each period uniquely represented. These two simple features, together, can nicely represent any seasonal variation (hour of day, day of week, week of year, etc.). (These features are most applicable when there is considerable seasonality in the time series. Since our applications are mostly limited to finance, where there is typically no strong seasonality, we do not use such constructed variables.)

Dummy variables can also be used to indicate important events (e.g., holidays) or conditions.

Feature Scaling

In practice, it is a common requirement to scale the features before feeding them to the algorithm. (Not only the computed, but all features.) It makes the training process smoother.

Two popular feature scaling methods are min-max normalization

$$\tilde{x_i} = \frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{1.23}$$

(i.e., to range [0, 1]), and standardization

$$\tilde{x_i} = \frac{x_i - mean(x)}{sd(x)} \tag{1.24}$$

(i.e., zero mean, unit variance).

Feature Importances

Machine learning is primarily focused on out-of-sample prediction performance, unlike traditional statistical and time series methods, which put more emphasis on explaining variables. Furthermore, some machine learning models (including neural networks) are quite difficult to interpret. So, even if we had the willingness, it would not be easy to understand the inner mechanisms of deep learning algorithms.

Time series analysts aim for interpretable predictions—for deep learning, this is a challenging task. We know how the network works, yet we don't know what it does. We can't understand all the weights and connections.

We can get at least a partial understanding of the model by analyzing feature importances. There are several possible measures of variable importance (see, e.g., Gevrey et al. [2003] or Olden et al. [2004]).

There is a simple and generally applicable method called mean decrease accuracy (also called permutation accuracy). It is most often used with random forests [Breiman, 2001], but it works with any machine learning algorithm. We permute each variable, one by one, and measure the decrease in accuracy resulting from each random permutation. The feature whose permutation leads to the largest decrease in accuracy, is the most important one. It is a simple method that can make neural networks' predictions somewhat easier to interpret, however, it is not perfect. Features are shuffled one-by-one. If there are highly correlated features, permuting one of them may not lead to as high drop in accuracy as it would deserve. Furthermore, when we encode a single piece of information into multiple features (for example, seasonality), it may be difficult to interpret the importances.

For regression problems, we can replace the accuracy score with R^2. It is preferable to compute the accuracies on a validation set, rather than the training set, so that the importances measure the features' contribution to the quality of out-of-sample predictions. For example, it may be advantageous to compute feature importances with bootstrapping: in each bootstrap run, we get a new validation set (the left-out observations), and we can compute (and aggregate) the importance scores on a new set of randomly shuffled values, thereby obtaining more reliable estimates.

1.2.3 Data Extensions

The main problem of deep learning-based time series forecasting is data scarcity. Limited data is very often an issue in machine learning and, in partcular, deep learning. Domingos [2012] argues, as a rule of thumb, that a dumb algorithm with lots of data is better than a clever algorithm with a modest amount of data. The data should get the job done. Ideally, we can gather more data so that we can afford more powerful learning algorithms. However, time series are not ideal in this respect. We can't speed up time, we can't just make more observations.

Data Augmentation

The problem is not specific to time series forecasting. In computer vision, limited data is often alleviated by data augmentation. It means increasing the dataset by applying random modifications. That is, the algorithm is given a slightly transformed image, which looks a little different, but still shows the same thing. Image augmentation can range from basic geometric transformations to neural style transfer and more [Shorten and Khoshgoftaar, 2019].

Similarly, time series can be augmented. Wen et al. [2020] produced a systematic review of time series data augmentation methods for deep learning. We may apply simple methods, such as injecting Guassian noise, spikes, or trends. Or, we can use advanced techniques, such as decomposing the time series, and recombining the augmented components. Wen et al. [2020] applied basic augmentation methods to forecasting using different deep learning models and datasets. Overall, the performance of the methods was promising, however, certain data/model pairs produced negative results.

Transfer Learning

Another possible approach to extending data is, simply, using multiple datasets. This can be accomplished by using tranfer learning: training the model on one problem, and transferring the knowledge to another problem. This is also best illustrated with examples from computer vision. The reason for the success of (deep) convolutional neural networks is that they can learn hierarchically: the first layers learn simple shapes, and later layers learn more and more complex patterns. This hierarchical structure allows for a straightforward implementation of transfer learning: grab a pre-trained network (trained on a huge dataset), freeze the initial layers, and re-train the final layers for the specific problem at hand. It is very useful with limited data: even though we may have just a few examples, we can learn the omnipresent patterns from a much larger set of images.

Though transfer learning may apply to time series as well, it seems less evident and less studied. But it can work. As an early example, Laptev et al. [2018] applied transfer learning to forecasting, and achieved dramatic accuracy improvement. The authors also claim that transfer learning can cut costs—it may be computationally unsustainable to train separate models for a large number of time series, but with transfer learning, we only need one.

[We chose to use a (kind of) transfer learning approach to increase our datasets in our various experiments with deep learning-based forecasting. Though, in a strict sense, what we did was not transfer learning. Transfer learning (as previously described) typically trains a model, freezes the earlier, and re-trains the later layes. We found that re-training the final layers is not necessarily necessary. That is, when we forecast time series with sufficiently similar behavior, it may be enough to jointly train the network on multiple time series to achieve competitive performance.]

1.2.4 Forecast Horizons

Producing forecasts one-step-ahead may be a straightforward task. Making long-term, multi-step predictions can be much more difficult. Bontempi et al. [2012] suggests 3 strategies for producing multi-step-ahead forecasts with machine learning: the recursive strategy, the direct strategy, and the multiple output strategy. The so-called recursive strategy makes one-step-ahead forecasts, and uses the forecasted values recursively to produce further predictions. It means that some forecasts are based on previous fore-

casts, and for this reason the errors may accumulate. The direct strategy trains a separate model for each time step in the future that we aim to predict. In this case, forecasts for different time steps are independent from each other. The multiple output strategy produces a vector as an output, that is, it produces forecasts for all time steps at once.

(In most of our empirical studies, we produced one-step-ahead forecasts only. When we made multi-step-ahead forecasts, we used the recursive strategy.)

1.2.5 Evaluation

Model evaluation is a key step in any machine learning project. When we train a learning model, we want to make sure that it will work on new examples. In case of time series forecasting, we want to make sure that it will work in the future.

Validation Sets

The performance of (supervised) machine learning models is usually evaluated with cross validation. Cross validation partitions the data into training and validation sets multiple times. That is, it randomly sets aside a proportion of the data for validating a model which was trained on the rest of the data. This validation is performed multiple times (on different validation sets), and the results are combined. Cross validation gives a good measure of the out-of-sample performance of the algorithms. However, applying cross validation to time series data is problematic due to the possible dependencies. Several modified cross validation methods have been proposed for time series, and, according to Bergmeir and Benítez [2012], cross validation can work just as well as last block evaluation (that is, validating the forecasts on the last part of the data, which is a common choice). They performed a thorough empirical study on stationary time series, and found that (even standard) cross validation can produce more robust error measures than last block evaluation. Yet, the authors only suggest the use of cross validation with adequate control for stationarity.

(In our empirical studies, we chose to validate our forecasts on the last part of the available time series. However, we tried to make our evaluations more robust by producing forecasts for multiple instances, and, in some cases, by re-running the analyzes on a completely separate time period.

Also, we tried training forecasting models on bootstrapped subsequences of the time series, and using the observations that were not sampled by the bootstrap as a validation set. While it is not perfectly correct, as we may use later observations to train the model that predicts previous observations, we found that it can be a useful addition to last block evaluation.)

Evaluation Metrics

Model performance can be evaluated from different perspectives. Whether we perform regression or classification, we can choose from different metrics to quantify the quality of the prediction. In most cases it is preferable to use several metrics at once.

Value Forecasts In general, time series forecasting can be formulated as a regression problem: we are predicting continuous values. Mean squared error is often used in neural networks as loss function—since it is the objective, it is a must-have for evaluation as well. Usually, its square root, RMSE (root mean squared error)

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{1.25}$$

is reported. MAE (mean absolute error)

$$MAE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{1.26}$$

is also often reported, which is less sensitive to outliers, and may be easier to interpret. We could also use MedAE (median absolute error)

$$MedAE(y, \hat{y}) = median(|y_1 - \hat{y}_1|, \ldots, |y_n - \hat{y}_n|), \tag{1.27}$$

which is even more robust to outliers. R^2 (coefficient of determination)

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}. \tag{1.28}$$

is the proportion of variance explained. Unlike the just discussed error measures, in case of R squared, higher values are preferred. It usually takes

values between 0 and 1, which is a convenient property. (It takes 1, when the forecasts have no error, it takes negative values when we do worse than always predicting the mean of the target values.) We can also use percentage errors, such as MAPE (mean absolute percentage error)

$$MAPE(y, \hat{y}) = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{1.29}$$

or SMAPE (symmetric mean absolute percentage error)

$$SMAPE(y, \hat{y}) = \frac{100}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2}, \tag{1.30}$$

which are relative errors in a sense, and are popular for forecasting problems.

Direction Forecasts Forecasting can be transformed into a binary classification problem, when we focus solely on the direction of changes (up or down). Thus, we can always use classification metrics to make the evaluation more detailed. Accuracy is simply the proportion of correct predictions

$$accuracy(\tilde{y}, \tilde{\hat{y}}) = \frac{1}{n} \sum_{i=1}^{n} 1_{\tilde{y}_i = \tilde{\hat{y}}_i}. \tag{1.31}$$

($\tilde{y}$ and $\tilde{\hat{y}}$ indicate the directions of changes of the true values, and the predicted directions of change.) Accuracy may be misleading when the classes are imbalanced. (If the values of the time series increase in the vast majority of the cases, always predicting an upward movement produces a high accuracy, even though the forecasts may be completely worthless.) In such cases we may use precision

$$precision(\tilde{y}, \tilde{\hat{y}}) = \frac{\sum_{i=1}^{n} 1_{\tilde{y}_i = 1 \; and \; \tilde{\hat{y}}_i = 1}}{\sum_{i=1}^{n} 1_{\tilde{\hat{y}}_i = 1}} \tag{1.32}$$

and recall

$$recall(\tilde{y}, \tilde{\hat{y}}) = \frac{\sum_{i=1}^{n} 1_{\tilde{y}_i = 1 \; and \; \tilde{\hat{y}}_i = 1}}{\sum_{i=1}^{n} 1_{\tilde{y}_i = 1}}. \tag{1.33}$$

Precision is the fraction of our positive predictions that are actually positive. Recall is the fraction of actually positive outputs that we predicted correctly. F1 score is the harmonic mean of precision and recall

$$F1(\tilde{y}, \tilde{\hat{y}}) = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{1.34}$$

In addition to these metrics, a confusion matrix is often produced to summarize the correct predictions and the two types of errors in an easy-to-understand format.

Interval Forecasts The indicators discussed so far evaluate point forecasts, prediction intervals require another set of measures. Khosravi et al. [2011] claims that prediction intervals are frequently assessed by coverage probabilities only, disregarding the width of the intervals, which may be misleading—we should use further metrics. Prediction interval coverage probability (PICP)

$$PICP = \frac{1}{n} \sum_{i=1}^{n} 1_{y_i \in [L_i, U_i]} \tag{1.35}$$

is the proportion of observations that fall into the interval. (L_i and U_i are the lower and upper bound of the ith prediction interval.) The value of PICP should be very close to the (chosen) nominal confidence level. Mean prediction interval width (MPIW)

$$MPIW = \frac{1}{n} \sum_{i=1}^{n} (U_i - L_i), \tag{1.36}$$

as the name implies, measures the width of the interval. Normalized mean predition interval width (NMPIW)

$$NMPIW = \frac{MPIW}{R} \tag{1.37}$$

is the MPIW divided by the range (R) of target values. Coverage width-based criterion (CWC)

$$CWC = NMPIW(1 + 1_{PICP<\mu} exp(-\eta(PICP - \mu))) \tag{1.38}$$

is a combined metric evaluating both coverage and width. It takes the value of NMPIW when the PICP is above the nominal confidence level

(μ), and it takes larger values when PICP is below its target value. (η is a hyperparameter—following Khosravi et al. [2011], we set its value to 50.) The lower CWC value the better.

Quantile Forecasts Quantile forecasting can be considered a special case of interval forecasting: one-sided intervals. Christoffersen [1998] produced a framework for evaluating conditional interval forecasts using likelihood ratio tests, and applied it to Value-at-Risk forecasts. Christoffersen [1998] proposed three tests: an unconditional coverage test, an independence test, and a joint test of both independence and coverage. We should test both the correct coverage (i.e., the right proportion of observations fall into the interval), and the serial independence (i.e., the violations do not cluster). These tests are often applied, however, Engle and Manganelli [1999] claim that these are not sufficient to evaluate a quantile forecasting model. Engle and Manganelli [1999] proposed the Dynamic Quantile (DQ) test. (Let's call the θ-quantile forecast "$-VaR_t$" for now, since these tests were proposed for the specific task of Value-at-Risk forecasting, which is described in more detail later in the Applications chapter.) The DQ test uses a Hit variable

$$Hit_t = I(y_t < -VaR_t) - \theta \tag{1.39}$$

that takes $1 - \theta$ when the observation exceeds the VaR-threshold (that is, when the true value is below the forecasted quantile), otherwise it takes $-\theta$.

The expected value of Hit is 0. Hit should be uncorrelated with (at least) its own lagged values, the forecasted Value-at-Risk, and a constant. The DQ test fits a linear regression

$$Hit = X\delta + u \tag{1.40}$$

(X is a $t \times q$ matrix, where t is the numer of time steps, and q is the number of input variables).

The input variables include a constant, a few lags of Hit, and the VaR estimate. (The test can easily be extended, using further explanatory variables, for example, dummy variables indicating the years.) The null hypothesis is that the parameters of the regression are zero ($H_0 : \delta = 0$), which can be tested with an asymptotically chi-square distributed test statistic

$$\frac{Hit' X[X'X]^{-1}X'Hit}{\theta(1-\theta)} \overset{a}{\sim} \chi^2(q) \tag{1.41}$$

under H_0 ($\overset{a}{\sim}$ denotes asymptotic distribution).

That is, to make sure that the hits (violations) are unbiased and uncorrelated, we need to test if the regression has no explanatory power.

Towards Applications The following chapter presents our empirical studies in financial forecasting. These applications build heavily on the methodology described above. In most studies, we use either convolutional or recurrent neural networks to produce forecasts for the future. Transfer learning is applied in various forecasting problems (either risk or mortality forecasting), since the available datasets are rather small. Quantile regression is applied to Value-at-Risk forecasting. We use bootstrapping and mean decrease accuracy to validate the stylized properties of volatility. We use recursive multi-step forecasting to produce long-term mortality predictions. For more details of the applications, read the next chapter.

Chapter 2

Results

The following sections describe our empirical applications.

Software We built most of our forecasting algorithms in Python [Van Rossum and Drake Jr, 1995]. It is a high-level, general-purpose programming language, that is commonly used in machine learning projects. One of the biggest advantages of Python is that it has a huge ecosystem of packages, which facilitates work.

All our neural networks were implemented in Keras [Chollet et al., 2015], which is an open-source neural network library built upon the TensorFlow [Abadi et al., 2015] machine learning platform. Keras includes implementations of popular neural networks—feedforward, convolutional, and recurrent architectures.

We used the scikit-learn machine learning library [Pedregosa et al., 2011] to build alternative machine learning models, and to evaluate predictions. Data preprocessing was performed using the Pandas package [Wes McKinney, 2010]. Our figures were produced using the matplotlib plotting library [Hunter, 2007].

The R programming language [R Core Team, 2014] was also used.

All the deep learning-based methods were implemented in Python, and most benchmark models, too. The Lee-Carter mortality rate forecasting model was implemented in R.

What was missing? While machine learning-based forecasting has a considerable literature, the application of deep learning is still in its infancy.

Recurrent and convolutional networks have gone through considerable improvement in recent years, and the achievements have not yet been exploited. Financial forecasting (and our areas of application, risk forecasting and mortality forecasting, in particular), primarily relies on simpler forecasting techniques. Hybrid methods are also often applied. However, fully deep learning-based approaches are rather rarely used. As powerful sequential learning algorithms are increasingly available, and the potential of transfer learning for time series forecasting is better explored, we have research opportunities that would not have existed a few years ago. We applied state-of-the-art algorithms to areas of great practical importance, and proved the methods with comparisons to traditional models—this may be the major contribution of our research efforts. This field of research is very quickly evolving, so whatever I state today as a missing piece in the literature, may lose its novelty tomorrow. For example, when we applied a supposedly novel approach to mortality rate forecasting (as it is described later in this chapter), Richman and Wuthrich [2019] delivered a very similar approach at about the same time.

Areas of Application The following sections describe our applications of deep learning-based time series forecasting. The applications fall into two main fields: financial risk forecasting and mortality rate forecasting. The two fields are weakly related, since both are used in financial practice. The choice of application areas was motivated by the nature of available data and the practical importance of the potential results. Each field provides fairly reliable historical datasets. The frequency of observations (and so the size of available data) differs in the two fields, which allowed us to produce forecasts in various data environments. Good forecasts, in either field, are of great importance, which was also a major consideration in our choice.

2.1 Risk Forecasting

Unsurprisingly, finance is a popular area of application for time series forecasting. There are excellent datasets and there are huge rewards for correct forecasts.

Knowing the future prices of financial assets can be priceless. However, such a precious knowledge cannot be easy to acquire. The efficient-market hypothesis [Fama, 1970] states that prices fully reflect all available

information. From a forecasting point of view, perfect market efficiency would imply a guaranteed failure. It is clear that markets are not perfectly efficient—signs of inefficiency (that is, predictability that allows for excess returns) are often found. Malkiel [2003] argues that despite the fact that predictable patterns may be found, the investment opportunities are not robust. Arbitrage opportunities self-destruct. Easy money is picked up quickly.

Though the recent progress of data science may deliver new opportunities and better results to asset price forecasting, we have chosen an easier task: forecasting the risk of financial instruments.

The most general measure of risk is volatility (the degree of variation of returns). It is most often estimated as the standard deviation of logarithmic returns. Volatility measures the overall variation of returns, we may need further metrics to quantify downside risk, that is, only the negative outcomes. Value-at-Risk (which is a small quantile of the return distribution) is a common choice for that. Volatility and Value-at-Risk were the subject of our risk-forecasting studies.

2.1.1 Directional Forecasts of Range-Based Volatility Estimates

Volatility (or the dispersion of prices) is not directly observable—we need to use estimates. The standard deviation of daily returns is a very popular choice. It is fairly straightforward, easy to compute, freely available, however, it is just a rough estimate. By using just a single observation each day, we disregard much of the price information. We could make an improvement by using intraday trading data, but it is not as easy to obtain. Range-based volatility estimation may be the golden mean: using 4 obervations per day (open, high, low and close prices). These values have been recorded for decades, for lots of stocks, and are freely and easily available.

Range-based volatility estimators are surprisingly rarely applied. Our goal was to make a comparison between these undeservedly unpopular range-based esimators and the popular raw (that is, close-only) volatility estimator in terms of forecastability.

Stylized Facts Volatility is forecastable. There are several well documented facts supporting this claim [Engle and Patton, 2001]. Volatility is persistent: large moves and small moves cluster. It is mean reverting: there seems to be a normal level to which it returns after high or low volatility

periods. Positive and negative innovations can have different impacts: equity returns and volatility were found to be negatively correlated. Volatility might be influenced by exogenous variables (e.g., other assets, announcements). All these properties suggest that it is predictable—predictable but not easy to predict. An ideal task for deep learning-based forecasting algorithms.

Forecasting Volatility Volatility might be forecasted using various methods. Poon and Granger [2003] produced an extensive review of volatility forecasting models, summarizing 93 papers: historical volatility models, ARCH-class models, stochastic volatility models, and so on. However, neural networks-based models were excluded from the study, even though they have various notable applications in the field.

Malliaris and Salchenberger [1996] used neural networks to forecast implied volatilities. Donaldson and Kamstra [1996] combined time series forecasts of volatility using neural networks. Roh [2007] proposed a hybrid volatility forecasting model with neural networks and time series models. Dunis and Huang [2002] used recurrent neural networks and model combination. Xiong et al. [2015] applied LSTM to forecast volatility using Google domestic trends data.

Volatility Estimators Stock prices (S_t) are usually assumed to follow a geometric Brownian motion, which satisfies $dS_t = \mu S_t dt + \sigma S_t dW_t$, where W_t is a Brownian motion or Wiener process, μ is the drift, and σ is the volatility. The aim is to estimate σ.

The close-to-close volatility estimator

$$\sigma = \sqrt{F}\sqrt{\frac{\sum_{t=1}^{N}(ln(\frac{C_t}{C_{t-1}}) - \overline{ln(\frac{C_t}{C_{t-1}})})^2}{N-1}} \tag{2.1}$$

uses only the daily closing prices. It is the standard deviation of daily logarithmic returns. C_t is the closing price of day t, N is the number of days used to make the estimate, and F is used to scale the estimate to an arbitrary time unit (typically, a year). It is simple and intuitive, but limited. We could make much more precise estimates by using intraday observations, but those are rarely available. However, we still have the often neglected open (O_t), high (H_t), and low (L_t) values of the daily stock prices.

Parkinson [1980] proposed a volatility estimator

$$\sigma_P = \sqrt{F}\sqrt{\frac{\frac{1}{4\,ln(2)}\sum_{t=1}^{N}(ln(\frac{H_t}{L_t}))^2}{N}} \tag{2.2}$$

using daily high and low prices. Parkinson [1980] claims that this so-called extreme event method is far superior to the traditional (close-to-close) estimate.

Garman and Klass [1980] used all four values (open, high, low, and close) to make a volatility estimate

$$\sigma_{GK} = \sqrt{F}\sqrt{\frac{\sum_{t=1}^{N} 0.5(ln(\frac{H_t}{L_t}))^2 - (2\,ln(2)-1)(ln(\frac{C_t}{O_t}))^2}{N}}. \tag{2.3}$$

The proposed estimator achieved considerably higher efficiency relative to the close-to-close estimator.

Rogers and Satchell [1991] proposed yet another formula

$$\sigma_{RS} = \sqrt{F}\sqrt{\frac{\sum_{t=1}^{N} ln(\frac{H_t}{O_t})(ln(\frac{H_t}{O_t}) - ln(\frac{C_t}{O_t})) + ln(\frac{L_t}{O_t})(ln(\frac{L_t}{O_t}) - ln(\frac{C_t}{O_t}))}{N}}, \tag{2.4}$$

which is unbiased even when there is a nonzero drift. Rogers et al. [1994] compared this estimate to the one proposed by Garman and Klass [1980] and found that it is only superior when there is a drift in the data.

Yang and Zhang [2000] proposed another drift-independent formula

$$\begin{gathered}\sigma_{YZ} = \sqrt{F}\times \\ \sqrt{\frac{\sum_{t=1}^{N}(ln(\frac{O_t}{C_{t-1}}) - \overline{ln(\frac{O_t}{C_{t-1}})})^2}{N-1} + \frac{k\sum_{t=1}^{N}(ln(\frac{C_t}{O_t}) - \overline{ln(\frac{C_t}{O_t})})^2}{N-1} + (1-k)V_{RS}}, \\ k = \frac{0.34}{1.34 + \frac{N+1}{N-1}}, \\ V_{RS} = \frac{\sum_{t=1}^{N} ln(\frac{H_t}{O_t})(ln(\frac{H_t}{O_t}) - ln(\frac{C_t}{O_t})) + ln(\frac{L_t}{O_t})(ln(\frac{L_t}{O_t}) - ln(\frac{C_t}{O_t}))}{N}.\end{gathered} \tag{2.5}$$

It can deal with opening jumps, and it is claimed to have the smallest variance among estimators with similar properties.

Figure 2.1 displays volatility estimates of the Dow Jones Industrial Average stock market index for a sample period.

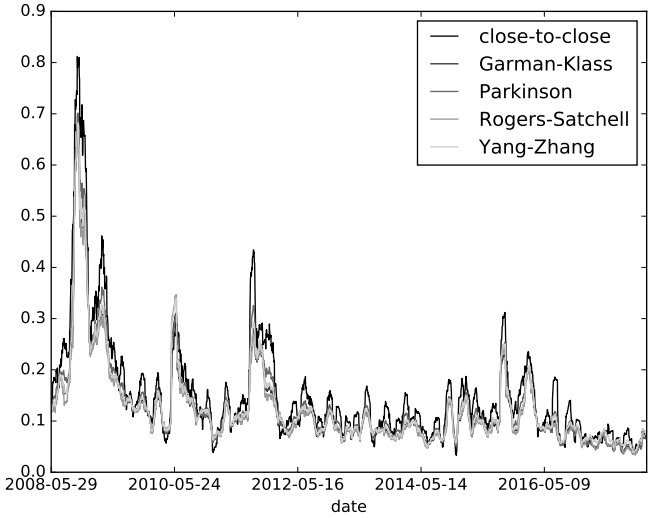


Figure 2.1. Volatility estimates for the DJIA index in the observed period

Shu and Zhang [2006] performed a simulation study, and found that all these estimators work well when asset prices follow a continuous geometric Brownian motion. With opening jumps or large drifts, however, there may be significant differences. Shu and Zhang [2006] claim that range-based volatility estimates are fairly robust against microstructure noise, and that they are quite close to integrated variances computed from high frequency intraday trading data. These findings support the use of range-based estimators.

Data Our forecasting study used the price history of constituents of the Dow Jones Industrial Average index. The dataset was obtained from Yahoo Finance[1]. We used 29 of the 30 constituents, since one stock did not have a long enough history.

Volatility estimates were computed using a sliding window of 21 days, that is, a trading month of time.

We produced a binary variable to indicate the direction of the daily changes of the volatility estimates, with 1s indicating upward movements.

[1] https://finance.yahoo.com/

We used this indicator variable to produce directional forecasts.

We used the first 70% of the available data as a training set, and the remaining 30% as a test set. That is, the first 70% of the data was used to train the neural network models, while 30% (about the last 3 years of time) was used to make predictions and to evaluate them.

Neural Network Architecture We used the same recurrent neural network architecture throughout our experiments. The hyperparameters were not specifically optimized, and we applied the same network to each stock under study.

Since the daily time series of volatility estimates are very small data for deep neural networks, we considered it reasonable to build rather small neural network. A 2-layer LSTM was applied with 10 units in each layer. The last LSTM layer was followed by a dense layer with a single unit with sigmoid activation function. We used a dropout [Srivastava et al., 2014] of .3 on the non-recurrent connections of the network. The loss function was binary cross entropy for the directional forecasts, and mean squared error for the value forecasts. (We built separate neural networks for the binary and the continuous forecasting problems.)

The volatility series were unrolled for 10 steps, and these 10-step sequences were fed to the learning algorithm in batches of 32 elements. The learning rate was set to .001. The training process ran for 300 epochs.

Results We produced one-step ahead forecasts for the different volatility estimates. We defined 2 problems: predicting only the directions of volatility changes, and predicting the numeric values of the volatility estimates. The results for the two problems are reported separately.

Direction Forecasts Our first experiment aimed to forecast the directions of daily volatility changes. That is, we did not use the volatility estimates, but only a binary variable indicating upward movements of the volatility. A series of zeros and ones—this was our entire dataset.

The last 3 years of the 10-year dataset was used as an out-of-sample test set. We made one-step-ahead forecasts, that is, we predicted only the next change at each step.

A separate neural network was trained for each stock. We chose 4 evaluation metrics for this binary forecasting problem: accuracy, precision, recall,

and F1 score. The results were averaged over all stocks.

	Accuracy		Precision		Recall		F1	
	mean	std	mean	std	mean	std	mean	std
close-to-close	0.51	0.02	0.52	0.07	0.31	0.25	0.33	0.17
Garman-Klass	**0.57**	0.03	**0.63**	0.05	0.30	0.10	0.40	0.10
Parkinson	**0.57**	0.02	**0.63**	0.04	**0.32**	0.09	**0.42**	0.08
Rogers-Satchell	0.55	0.03	0.61	0.04	0.26	0.10	0.35	0.10
Yang-Zhang	**0.57**	0.02	**0.63**	0.05	0.29	0.09	0.39	0.08

Table 2.1. Evaluation metrics for one-day-ahead direction-of-change forecasts

Table 2.1 shows the results of this first experiment. The close-to-close estimator produced an accuracy just above .5, while all other estimators achieved .55 or higher. Although the accuracy of the range-based estimators seemed promising, F1 scores were poor. F1 is the harmonic mean of precision and recall. Precision is the fraction of true upward movements among our predicted upward movements. Recall is the fraction of true upward movements that we predicted correctly. Table 2.1 shows that the recall of the algorithm was very poor—it had difficulties identifying upward movements in the volatility estimates. The algorithm was downward-biased. This is an undesirable property, since upward and downward changes are about equally likely. (Unless it would be more important to identify drops than rises, but it is clearly not the case with volatility forecasting.) In order to increase recall, we modified the threshold of the binary classification.

	Accuracy		Precision		Recall		F1	
	mean	std	mean	std	mean	std	mean	std
close-to-close	0.50	0.02	0.50	0.01	**0.88**	0.16	**0.63**	0.06
Garman-Klass	**0.58**	0.03	0.59	0.03	0.49	0.09	0.53	0.06
Parkinson	**0.58**	0.02	**0.60**	0.03	0.46	0.09	0.51	0.06
Rogers-Satchell	0.57	0.02	0.58	0.03	0.45	0.10	0.50	0.07
Yang-Zhang	**0.58**	0.02	0.59	0.03	0.47	0.10	0.52	0.07

Table 2.2. Evaluation metrics for predictions with .45 probability threshold

We changed the default probability threshold from .5 to .45, in order to get more postive outcomes. Table 2.2 summarizes the results. The recall values increased, as expected, and so did the F1 scores. Producing more ups did not cause a decrease in overall accuracy—all range-based estimators

produced higher accuracies with this setting.

In order to optimize the results further, we tried to exclude uncertain decisions. Since the neural network does not produce a binary outcome, but a probability, we can modify the threshold value so that it makes a choice only when it has enough confidence in the decision (that is, when the probability is far from .5). We chose to exclude predictions between .4 and .5 probability, and classify above .5 predictions as upward and below 0.4 predictions as downward changes. (The thresholds were chosen arbitrarily, based on our previous findings.)

	Accuracy		Precision		Recall		F1	
	mean	std	mean	std	mean	std	mean	std
close-to-close	0.51	0.11	0.51	0.11	**0.93**	0.20	**0.66**	0.13
Garman-Klass	0.61	0.03	**0.63**	0.04	0.50	0.11	0.55	0.07
Parkinson	0.61	0.03	**0.63**	0.05	0.45	0.11	0.51	0.08
Rogers-Satchell	0.59	0.03	0.62	0.05	0.46	0.15	0.52	0.12
Yang-Zhang	**0.62**	0.03	**0.63**	0.04	0.48	0.12	0.54	0.09

Table 2.3. Evaluation metrics for confident predictions (P>.5 or P<.4)

As displayed in Table 2.4, this new strategy made considerably fewer predictions, but in these cases it achieved about 60% accuracy for the range-based estimators, while the close-to-close estimator remained around 50%.

close-to-close	Garman-Klass	Parkinson	Rogers-Satchell	Yang-Zhang
0.28	0.66	0.67	0.54	0.64

Table 2.4. Average proportions of confidently predicted directions

Table 2.4 shows the proportions of cases when the forecasing algorithm made a decision. It seems that lower accuracies came with lower confidence. Our forecasting algorithm made fewer predictions for the hard-to-predict close-to-close estimator.

Our results show that it may be easier to forecast the directions of changes in range-based volatility estimates, than those in the raw close-to-close estimate. Even though all estimates move fairly close together (in case of the DJIA index, as displayed in Figure 2.1, all pairwise correlations are above .95), we found a remarkable difference in the forecastability of the direction of changes.

Value Forecasts Our second experiment aimed to forecast the values of the volatility—not only the directions but the numeric estimates. That is, we replaced the binary forecasting problem with a continuous one.

Table 2.5 shows the results of this regression problem. RMSE (root mean squared error), and SMAPE (symmetric mean absolute percentage error) were used as evaluation metrics—we measured both absolute and relative errors.

These forecasts show less visible differences in predictability. The close-to-close estimates produced the largest SMAPE and second largest RMSE, so it still seemed that the range-based estimates are more predictable.

	RMSE		SMAPE	
	mean	std	mean	std
close-to-close	0.0185	0.0047	7.03	1.66
Garman-Klass	0.0141	0.0055	5.14	1.76
Parkinson	**0.0131**	0.0043	**5.01**	1.27
Rogers-Satchell	0.0155	0.0070	5.07	1.84
Yang-Zhang	0.0194	0.0046	5.28	0.99

Table 2.5. Evaluation metrics for value forecasts of the volatility estimates

Forecasts With Enhanced Estimators We applied a few reasonable enhacements to the estimators, and reproduced the forecasts.

We used adjusted closing prices in the volatility formulas, so that we account for dividends and stock splits that happen after-hours.

There are slight differences in what the different estimators estimate exactly. While (2.1) and (2.5) produce estimates over the whole day; (2.2), (2.3) and (2.4) consider the trading period only. So, the previously given volatility estimators should be modified so that they are comparable [Molnár, 2012]. We have implemented these modifications.

Tables 2.6 and 2.7 show the results. The directional forecasts show similar but less clear patterns compared to the original estimates. However, the values of our enhanced range-based estimates seemed essentially unpredictable as most of them produced about 10 times higher average errors than the close-to-close.

	Accuracy		Precision		Recall		F1	
	mean	std	mean	std	mean	std	mean	std
close-to-close	0.51	0.02	**0.50**	0.01	**0.83**	0.09	**0.63**	0.02
Garman-Klass	0.51	0.04	0.45	0.03	0.55	0.21	0.48	0.10
Parkinson	0**.56**	0.06	0.44	0.04	0.42	0.20	0.41	0.11
Rogers-Satchell	0.54	0.04	0.45	0.03	0.47	0.16	0.45	0.08
Yang-Zhang	0.50	0.02	**0.50**	0.02	0.82	0.11	0.62	0.04

Table 2.6. Evaluation metrics for one-day-ahead direction-of-change forecasts of the modified volatility estimates (.45 threshold)

	RMSE		SMAPE	
	mean	std	mean	std
close-to-close	**0.0183**	0.0044	**6.80**	0.99
Garman-Klass	0.2786	0.1009	44.18	6.88
Parkinson	0.4190	0.1722	50.85	8.80
Rogers-Satchell	0.3815	0.1511	48.06	9.38
Yang-Zhang	0.0276	0.0070	11.88	3.64

Table 2.7. Evaluation metrics for value forecasts of the modified volatility estimates

Summary and Conclusions Since the volatility of financial asset prices is claimed to be more or less predictable, we aimed to compare the predictability of different range-based volatility estimates using a long short-term memory recurrent neural network.

We dare not make many conclusions, since we got different results using the basic and improved estimators.

It seems that the range-based estimates can be forecasted to some degree. The different range-based estimates generated forecasts of similar quality, and they outdid the raw close-to-close estimates—at least, using the original volatility estimators. Once we applied a few reasonable modifications to make them more precise and comparable, we got less clear and sometimes contradictory results.

Our forecasting model was not optimized, since we did not aim to measure the degree of predictability, but to make a comparison between the different estimates. We used small data which is not too good for our neural network-based forecasting. Future research could explore volatility estimates computed from intraday (high frequency) trading data.

2.1.2 Volatility Forecasting with Transfer Learning

Our study on range-based volatility estimates used separate neural networks to forecast every single series of estimated volatility. Since we used daily price observations, and most stocks had no more than a few decades of history, we had very small data. Neural networks have lots of parameters, and are very flexible, hence using small datasets is not beneficial. The trained models can be very unstable. Averaging over multiple models could bring some stability, but it would be much better to exploit the full potential of these powerful algorithms.

There is a pretty straightforward way to increase the size of the dataset: we could use multiple stocks to train a single model. It may sound strange at first, but this idea enjoys great popularity, and has a wide range of applications in certain areas of machine learning.

Sirignano and Cont [2019] used LSTMs to forecast high frequency trading data. They found that a universal model (trained jointly on all stocks) outperforms asset-specific models. This hold even for assets which are not part of the training dataset. The authors claim that it is evidence of a universal and stationary price formation mechanism.

If we can find generality in asset prices, we must find it in volatilites as well. The previously described stylized facts state that there are quite some general properties of asset volatilities. It suggests that knowledge gained from the volatility history of one asset could help predict the future volatility of another. Following this conjecture, we built a general volatility forecasting model using a deep neural network.

Data For this experiment, we downloaded the 10-year (2009-2018) daily price history of hundreds of assets. We chose to explore the constituents of the S&P 500 stock market index. The price data was obtained from Quandl[2].

We used the standard close-to-close volatility estimator (2.1), since it is the most commonly used formula. That is, we computed 21-day moving standard deviations of the daily logarithmic returns. These volatility estimates were annualized (multiplied by the square root of 252).

Having removed stocks with too many missing observations, 440 volatility time series remained. We split each one to 64-step overlapping subseries. These 64-step sequences were fed to the algorithm as inputs.

[2]https://www.quandl.com/data/WIKI

The subsequent, 65th value was used as the target. The dataset was jointly standardized, that is, we used the mean and standard deviation of the entire dataset to scale the values.

We randomly chose the following 10 stocks to evaluate our forecasts:

- Fidelity National Information Services, Inc. (FIS)
- Branch Banking and Trust Company (BBT)
- Regeneron Pharmaceuticals, Inc. (REGN)
- Thermo Fisher Scientific Inc. (TMO)
- Entergy Corporation (ETR)
- Agilent Technologies, Inc. (A)
- Deere & Company (DE)
- Kimberly-Clark Corporation (KMB)
- Bank of America Corporation (BAC)
- International Paper Company (IP)

Neural Network Architecture We used a one-dimensional dilated causal convolutional neural network. One-dimensional, since we have a single time dimension. Dilated, so that our model can learn from distant points in time. And causal, that is, it always learns from the past.

The network has 6 dilated convolutional layers with exponentially increasing dilation rates. (The l^{th} layer's dilation rate is 2^{l-1}.) Each layer has 8 filters with a kernel size of 2. We used the relu activation function in each layer.

The model was trained with the $\mathrm{adadelta}$ [Zeiler [2012]] optimizer, for 300 epochs.

Results We used the 10 randomly selected stocks to test our forecasting models. Two different convolutional neural networks were applied to forecast the chosen volatility series. The first one learns from the volatility history of the chosen stock only. We call this the *individual model*. The other one learns from all stocks' data, except the chosen 10 stocks—it learns from

more than 400 times as much data, but it knows nothing about the asset that it aims to forecast. We call this the *joint model*. We also used a simple ARIMA model as a benchmark.

We used the first 70% of the available 10-year data to train the models. The rest was used to evaluate the forecasts. One-step-ahead predictions were produced: we only forecasted the volatility estimate of the following day.

Table 2.18 displays the results. We used two metrics (root mean squared error, and symmetric mean absolute percentage error) to evaluate the value forecasts, and two metrics (accuracy, and F1 score) to evaluate the direction forecasts. The results were averaged over the randomly chosen 10 stocks.

The individual CNNs' performance was rather poor. Our neural network is probably way too complex to be trained with a single stock's data. A single time series is insufficient to find proper values for the parameters of the network. The weak forecasting performance is no surprise.

We used an automatically parameterized autoregressive integrated moving average model as a benchmark. The order of differencing was chosen with successive KPSS tests [Kwiatkowski et al., 1992], then grid search was used to choose the number of autoregressive and moving average terms (between 0 and 3) based on AIC [Akaike, 1974]. This method was strongly inspired by the automatic ARIMA model in the forecast package of R [Hyndman et al., 2007].

The ARIMA model did not deliver very accurate forecasts either. The regression metrics (RMSE, SMAPE) were somewhat lower than in case of the individual CNN, but classification metrics (accuracy, F1) are even more discouraging.

The joint CNN model produced the best result according to both regression and classification metrics. It is remarkable, since we did not optimize the network's hyperparameters. While we applied a popular automatic procedure to find reasonably optimal parameters for the ARIMA model, we chose the parameters of the CNN model arbitrarily. We could have used a similar grid search-based method to optimize the size and parameters of the neural network, too. However, we thought that an unoptimized model could deliver more general and more reliable results.

The unaveraged root mean squared errors are displayed in Figure 2.2. The unaveraged accuracy scores are displayed in Figure 2.3. We should prefer lighter cells in the former plot, and darker cells in the latter. The figures show that the results are fairly consistent—the joint convolution net-

	CNN Individual	ARIMA	CNN Joint
Value Forecasts			
RMSE	0.0283	0.0261	0.0154
SMAPE	9.6324	4.9468	3.9358
Direction Forecasts			
Accuracy	0.5333	0.5161	0.6262
F1	0.5379	0.4182	0.6835

Table 2.8. Evaluation metrics averaged over stocks

work performed best for almost all stocks, regarding both metrics.

The mean difference between the RMSE scores are 0.01285 (CNN Individual – CNN Joint), and 0.0107 (ARIMA – CNN Joint). While the assumptions underlying a paired sample t-test are not necessarily satisfied, we report some statistics that describe the significance of the differences. The standard errors are approximately 0.0018, and 0.0042. The t-value for the difference between the individual CNN errors and jointly trained CNN errors is 7.1113, while the t-value for the difference between the ARIMA and joint CNN is 2.5657. In case of accuracies, the same differences are t-values of –8.4483, and –9.538, respectively. Though the assumption of independence may not be satisfied, and we report the t-values for illustrative purposes only, we think that they are fairly indicative of the superior performance of the transfer learning approach.

Summary and Conclusions Since asset prices (and, especially, the volatility of price changes) seem to have general formation mechanisms, we built a jointly trained neural network to forecast volatilities.

We used a one-dimensional causal dilated convolutional neural network. When trained individually, it did not produce good results. However, trained jointly on hundreds of volatility series, the model clearly outperformed the individually trained network and the ARIMA benchmark.

The transfer learning approach took advantage of the increased data, and worked well, even though it was tested on stocks which were not part of the training set. It delivered clearly and consistently better predictions for both the value and the direction forecasts.

The results suggest that transfer learning can contribute to volatility forecasting. Daily price observations do not produce enough data for ap-

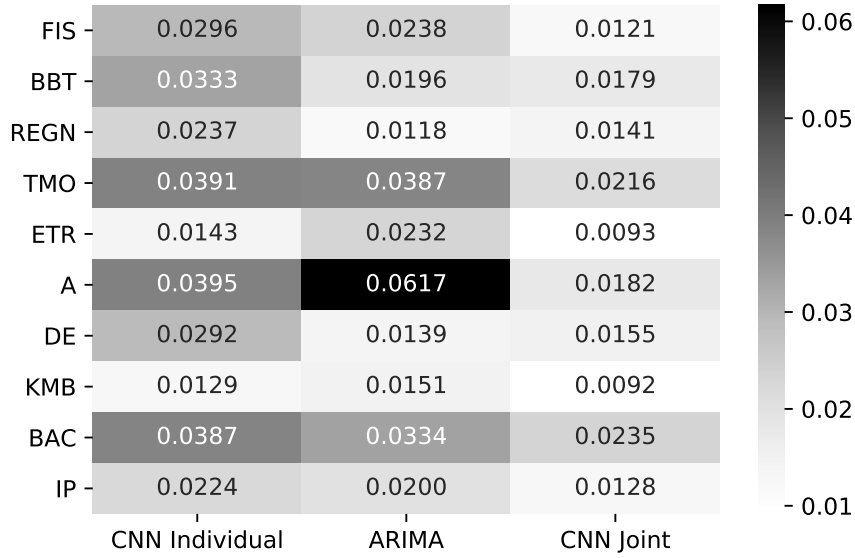


Figure 2.2. RMSE scores of different volatility forecasts for the 10 randomly chosen stocks

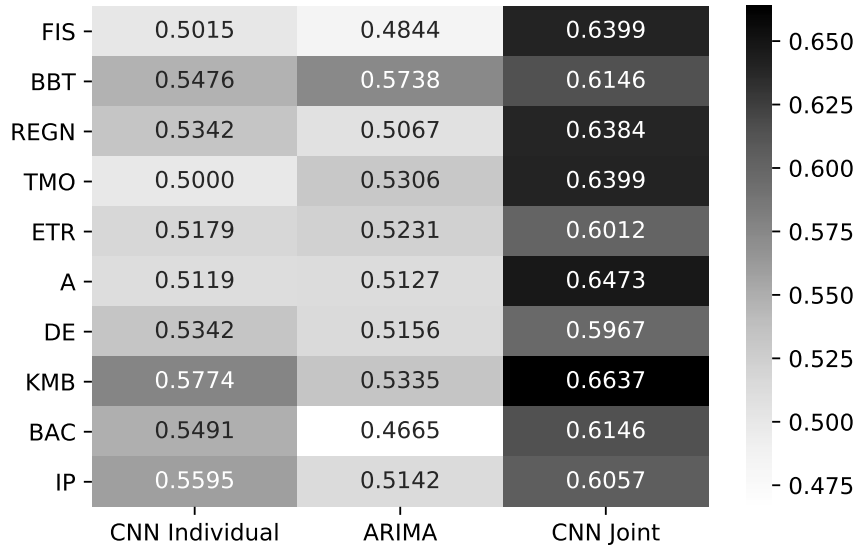


Figure 2.3. Accuracy scores of different volatility forecasts for the 10 randomly chosen stocks

plying complex machine learning models. However, using the proposed approach, it seems probable that deep learning can keep up with traditional approaches in volatility forecasting.

2.1.3 Inspecting the Stylized Facts of Volatility

All our volatility forecasting studies were built on the claim that there are certain general properties that make volatilities more predictable than, for example, returns. In the following, we describe an experiment, where we focused less on forecast accuracy and more on interpretation. Based on the stylized facts, we've built a forecasting model using multiple input features, with the primary purpose of measuring the features' contribution to the model's performance.

Our goal is not only to make predictions, but also to investigate the presence of the well-known properties of volatility. We aim to measure what features can help predict the future values of volatility.

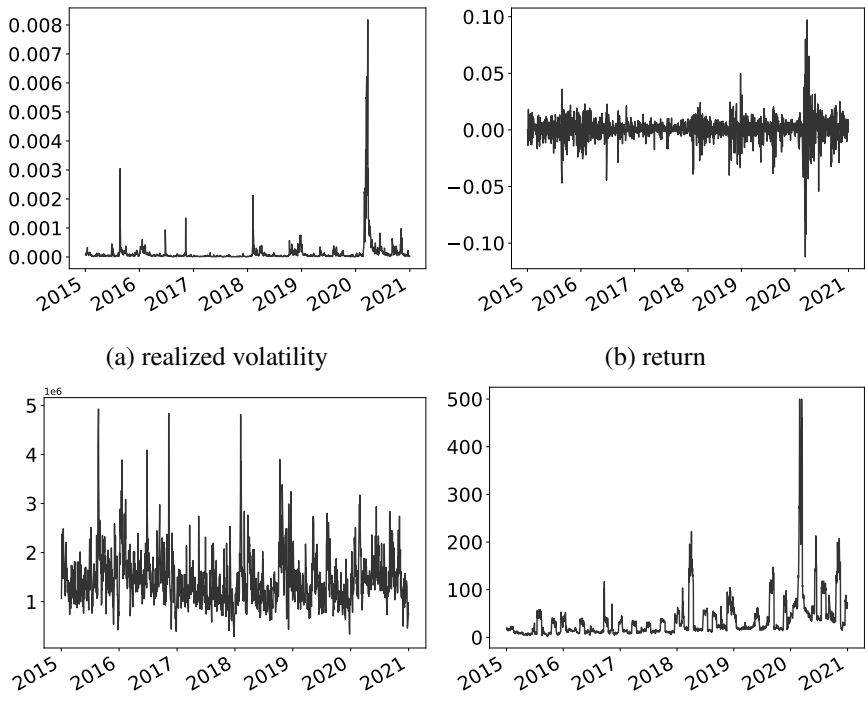
Data We aim to forecast realized volatilities computed from intraday price observations. So far, we have used volatilities computed from daily price observations—using high frequency data, we can get better estimates. Realized volatility is the sum of realized squared returns over a fixed time interval. Andersen and Teräsvirta [2009] claims that, although in theory we could get arbitrarily good estimates by increasing the sampling frequency, ultra-high frequency sampling can bias realized volatility estimates due to market microstructure noise (price rounding, bid-ask bounces, etc.). Andersen et al. [2001] used 5-minute observations for actively traded exchange rates, as a reasonable compromise. We, too, use 5-minute prices here to compute the realized volatility of the stock market index futures contract under study.

We chose to forecast the volatility of the E-mini S&P 500 futures contract. Andersen et al. [2018] analyzed intraday variations, and claim that the choice of E-mini is advantageous for various reasons. It is traded through a single electronic trading system, which operates almost continuously during weekdays, and it has a large daily turnover, and is extremely liquid. We computed volatility as the daily sum of squared 5-minute logarithmic returns.

Different features are applied to forecast the one-day-ahead volatility of the E-mini S&P 500. We use the (current) volatility itself as an input variable. We also use the daily return, the daily trading volume, and an external variable: the Google Trends of the keyword "s&p 500". These time series are plotted in Figure 2.4.

Google Trends are normalized search volumes for different keywords, as published officially by the search engine. Search trends quantify the time-varying interest of people in different topics. We have downloaded the Google Trends of the keyword "s&p 500". Though it is not a specific product or company, the search interest in the S&P 500 may be a good proxy for the overall interest in the stock market. Indeed, we may see in Figure (2.4d) that there was a remarkable jump in search interest during the early spread of the coronavirus.

Weekends and trading holidays were excluded from the dataset. A logarithmic transformation was applied to the target variable, in order to make it less skewed (Figure 2.5). All features were scaled using min-max normalization (1.23).



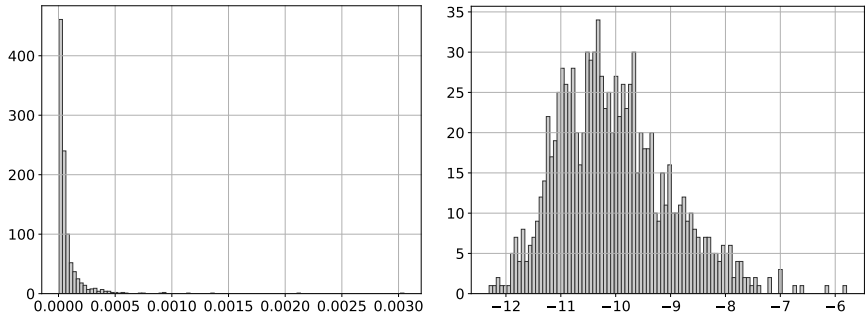
(a) realized volatility

(b) return

(c) volume

(d) Google Trends "s&p 500"

Figure 2.4. Input features used in the (realized) volatility forecasting neural network



(a) Original values

(b) Log-transformed values

Figure 2.5. Histograms of the output variable (i.e., realized volatility) in the train period

Neural Network Architecture The network consists of one LSTM layer with 4 units, followed by a dense layer with one unit and a linear activation. The model is trained on 16-step sequences (given in 32-element batches), which are randomly sampled during bootstrapping. The loss is mean squared error, optimized with the Adam optimizer. The training lasts for 300 epochs.

Results We run the LSTM multiple (=100) times on bootstrap sampled subsets of the volatility series.

Producing bootstrap samples is advantageous in, at least, three aspects. First, we can make an ensemble forecast by aggregating the predictions of the different bootstrap samples. Second, we can produce prediction intervals, that is, we can measure the uncertainty in our forecasts. Third, we can use the bootstrap left-out observations for evaluation purposes, and compute permutation-based feature importances.

Point Predictions The point predictions are not very accurate. Our forecasts did not manage to follow the volatility jump during the first wave of COVID-19. It is not surprising, since it was unprecedented—the maximum volatility in the training period was nowhere near close.

Table 2.9 shows that the bagged predictions produced lower errors than the averaged errors of the individual forecasts, especially in the test period.

We have computed directional accuracies for the test period (Table 2.10). These too are better for the bagged predictions.

	individual forecasts	bagged forecasts
validation set		
RMSE	0.000127	0.000127
SMAPE	36.416580	35.681638
test set		
RMSE	0.000702	0.000528
SMAPE	45.609503	43.161287

Table 2.9. Evaluation of the point predictions

	individual forecasts	bagged forecasts
accuracy	0.650271	0.665971
precision	0.643829	0.654206
recall	0.595265	0.619469
F1	0.614486	0.636364

Table 2.10. Directional accuracies in the test period

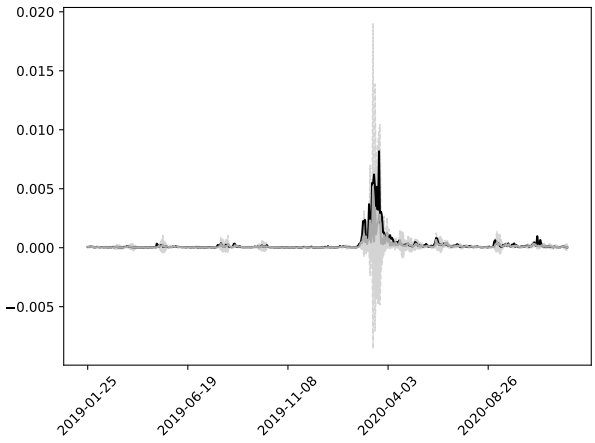


Figure 2.6. Interval forecasts of the realized volatility of E-mini S&P 500 futures

Interval Predictions In Figure 2.6, the solid black line shows the true volatility, the solid grey line shows the bagged forecast, and the shaded gray area shows the prediction interval. We produced 90% prediction intervals assuming that the noise is more or less normally distributed—it was clearly not a reasonable assumption. The lower end of the symmetric interval is often too low. (Negative values are not very informative. Constructing better, and preferably asymmetric, prediction intervals may be the subject of further research.) The coverage (PICP) of the interval in the test period is 89.375% (Table 2.11), which is fairly close to the target value. The intervals are considerably wider in turbulent periods of the stock market.

Feature Importances The primary purpose of this analysis was to examine the existence of the stylized properties of volatility. It seems that

PICP	MPIW	NMPIW	CWC
0.893750	0.000647	0.079123	0.187270

Table 2.11. Evaluation of the interval predictions for the test period

we can produce more or less accurate forecasts of the volatility by using a flexible learning model and input variables that are claimed to be related to volatility. But we also need to measure the contribution of the different input variables to the overall forecasting performance. It would be easy for simple models, but it is very difficult for deep neural networks. Still, there is one thing we can do: quantifying feature importances.

Variable importances can be considered an ultimately simplified model interpretation. Though we don't know what exactly the features do, at least, we know how much.

Mean decrease accuracy (that is, the decline in performance due to random permutation of input features) is an easy-to-implement importance measurement method. We have computed mean decrease accuracy in each bootstrap run, with both value (R^2) and direction ($accuracy$) forecasts on both the bootstrap left-out validation set and the separate test set (Figure 2.7).

In each case, feature importances were normalized (divided by the sum). The order of importance is the same, with one exception: the value forecasts in the test set expressed quite different behavior. The Google Trends data seems to be misleading in the test period (at least, considering the value forecasts). The results of the 4 computations are summarized in Figure 2.8.

Not surprisingly, realized volatility is the most important predictor. That is, the previous values of the output contribute the most to the forecasting performance. Return also seems to be an important predictor of volatility. Volume does not seem to have that much importance, though it got positive scores in three out of four cases. The Google Trends data seems to be not only useless, but harmful.

Summary and Conclusions We produced one-step-ahead point and interval forecasts of the realized volatility of the E-mini S&P 500 futures contract using multiple input variables in a recurrent neural network, with the aim of validating the existence of the well known stylized facts of volatility.

We found that historical values of volatility and return are, indeed, good

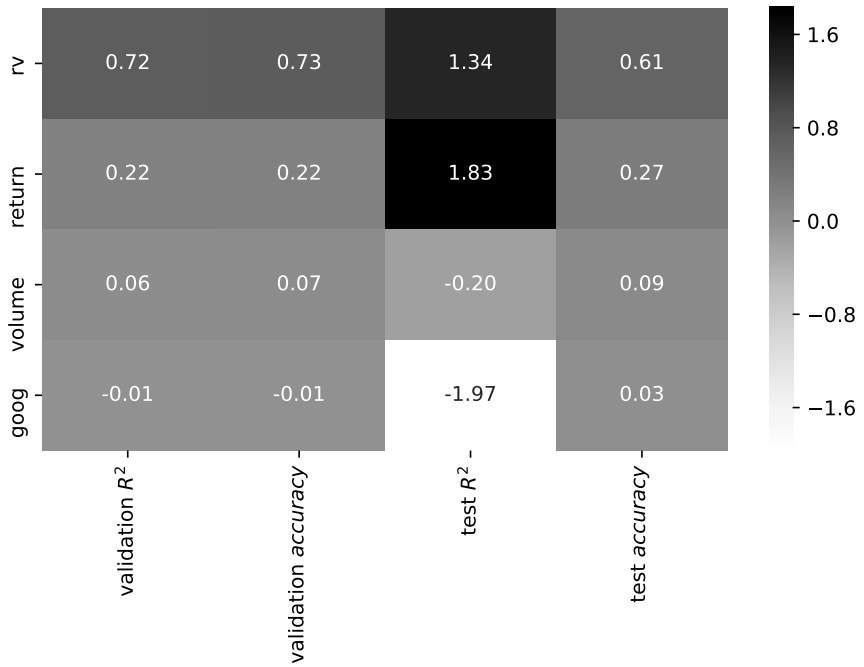


Figure 2.7. Feature importances computed on different data subsets (validation and test) and with different evaluation metrics (accuracy and R^2)

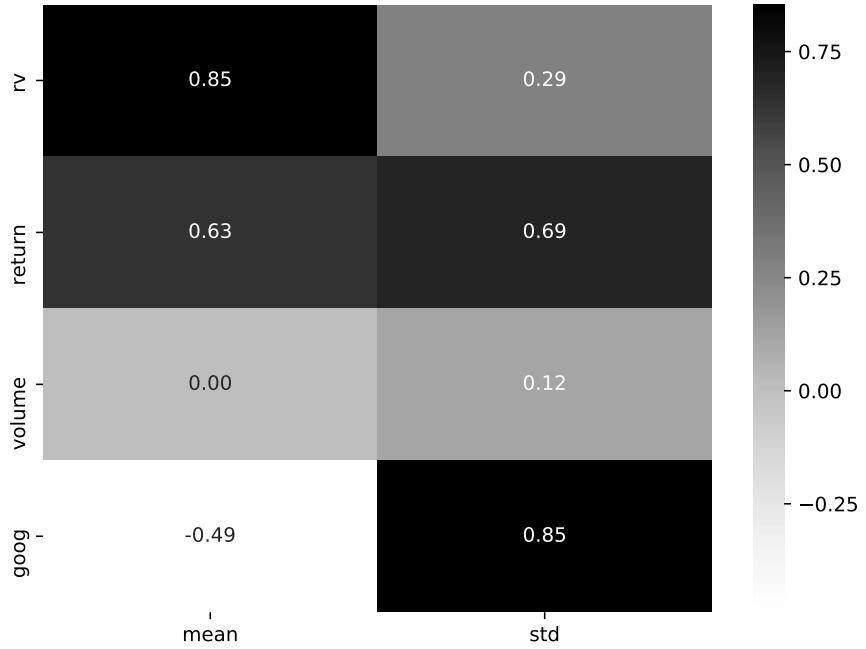


Figure 2.8. Mean and standard deviation of the different feature importance computations

predictors. Volume does not seem to be as important. The external variable that we used (the search interest in S&P 500) happened to have an adverse effect on our forecasting model.

2.1.4 Value-at-Risk Forecasting With Quantile Convolutional Neural Networks

It is natural to use return volatility as a measure of risk. However, we can (and should) have further definitions of risk. One possible objection to volatility is that it measures the variability of price movements in both directions. However, we may be more interested in the risk regarding just one direction. The so-called downside risk is the risk associated with losses only. In the following, we are going to focus on one possible measure of downside risk: Value-at-Risk.

Value-at-Risk Value-at-Risk estimates a realistically worst-case scenario: something that is so bad, that anything worse happens with a probability less than a (specified) small value [Shin, 2010]. It is a possible, but not very probable worst case. VaR is a quantile of the loss distribution. Though VaR is a bad return (i.e., a loss), it is usually reported as a positive number.

Mathematically, the Value-at-Risk

$$VaR_\theta(X) = F^{-1}_{-X}(1-\theta) = -\inf\{x \in \mathbb{R} : \theta \leq F_X(x)\} \tag{2.6}$$

of a return distribution X is the quantile function of $-X$ at the confidence level $1-\theta$.

Forecasting Value-at-Risk There are different ways to estimate and forecast Value-at-Risk. For example, we can estimate (and forecast) return variance, and compute the quantiles with some distributional assumptions. Traditional volatility forecasting models, such as ARCH/GARCH [Engle, 2001], are often applied. However, methods that require distributional assumptions have a drawback: they require distributional assumptions. Wrong assumptions can lead to poor estimates. One possible improvement is to relax the assumptions, so that we can choose any distribution [Hull and White, 1998]. Or, we can get rid of the obligation to choose a distribution, and use a method that works anyway—for example, quantile regression.

The CAViaR model of Engle and Manganelli [1999] used quantile regression to produce Value-at-Risk forecasts. Taylor [2000] used a quantile regression neural network. Xu et al. [2016] applied a quantile autoregression neural network (QARNN). Yan et al. [2018] used long short-term memory to produce forecasts of VaR.

ARCH-type models In the following, ARCH-type volatility forecasting models are briefly described. A detailed discussion of the topic is not intended, the interested reader may follow the references. Autoregressive conditional heteroscedasticity (ARCH) models are commonly used to model and forecast financial volatility. ARCH processes were proposed by Engle [1982]. An $ARCH(q)$ models the time series of variance (σ_t^2) as a function of the past q squared errors (e^2)

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i e_{t-i}^2, \tag{2.7}$$

where ω, and α_i are parameters. The error is assumed to follow a zero-mean normal distribution. Generalized autoregressive conditional heteroskedasticity models generalize ARCH models—they assume that the error variance follows an autoregressive moving average model. A $GARCH(p, q)$ [Bollerslev, 1986] models the variance with q lags of the squared error terms and p lags of the variance itself

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i e_{t-i}^2 + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2, \tag{2.8}$$

where ω, α_i, and β_i are parameters.

ARCH and GARCH models are well described in Brockwell et al. [2016]. There are many further ARCH-type models, however, Hansen and Lunde [2005] compared 330 models in terms of their out-of-sample forecast performance, and found no evidence that more sophisticated models outperform the popular $GARCH(1, 1)$ model.

ARCH-type models can be used to produce VaR-forecasts. Kuester et al. [2006] compared the one-step-ahead out-of-sample forecast performance of several models. They found that the great majority of GARCH models outperform unconditional models, while they found none of the CAViaR models to work very well. For this reason, we used a GARCH model, too, as a benchmark to the quantile regression-based forecasting techniques.

Data The dataset for this study was obtained from Kaggle[3]. It includes daily price observations for stocks listed on NASDAQ, NYSE, and AMEX. 100 randomly chosen stocks were analyzed using a 10-year period (from 2009-01-01 to 2018-12-31), using about the last 3 years as a test set. Value-at-Risk was forecasted from the daily logarithmic returns.

Neural Network Architecture A one-dimensional dilated causal convolutional neural network is applied with a quantile loss (1.20). (The value of θ is set to 0.05, 0.01, and 0.001 for the different VaR confidence levels.) The network has causal convolutional layers with exponentially increasing dilation rates. Each one has 8 filters with a kernel size of 2. The causal convolutional layers are followed by another convolutional layer with only 1 filter with a kernel size of 1. The causal convolutional layers use $relu$ activation, while the final convolutional layer uses linear activation.

The inputs to the network are 128-step sequences of daily logarithmic returns, while the outputs are the same sequences shifted by 1 step. The data was standardized using the mean and standard deviation of the entire dataset.

The $adadelta$ [Zeiler, 2012] optimizer was used, and the models were trained for 128 epochs.

Results The VaR-forecasts of the proposed neural network are compared to 3 benchmark forecasts: a constant quantile estimate, a linear quantile regression (an autoregressive model with 4 lagged input variables), and a GARCH(1,1) model with normal distribution. Two differently trained versions of the convolutional neural network were compared: one trained on one stock at a time, and another trained on all available training data. Each model was used to produce VaR predictions with 3 different confidence levels (95%, 99%, and 99.9%).

The Dynamic Quantile (DQ) test is used to evaluate the forecasts, using a constant, 3 lags of the Hit variable (1.39), and the current VaR forecast as input variables. (The DQ test is described in the Evaluation section of the Methods chapter.)

The results are shown in Tables 2.12, 2.13, and 2.14. Except for the 99.9% confidence rate, the rejection rates of the DQ test with two different

[3]https://www.kaggle.com/qks1lver/amex-nyse-nasdaq-stock-histories

significance levels (0.01 and 0.05) for the 100 stocks in the study, are reported, together with the averages, medians, and standard deviations of the realized VaR exceedances, and the average forecasted VaR values. (The rejection rate of the DQ test means the proportion of stocks for which the null hypothesis of the test was rejected at the given significance level.)

	Exceedances			DQ Test Rejections		VaR
	Mean	Median	SD	0.01	0.05	Mean
Constant	0.0399	0.0358	0.0291	0.5300	0.6600	0.0727
GARCH	0.0347	0.0338	0.0134	0.2100	0.4100	0.0739
QR	0.0409	0.0344	0.0293	0.6200	0.7200	0.0689
QCNN	0.1269	0.1245	0.0362	0.7400	0.8000	0.0423
Joint QCNN	0.0433	0.0444	0.0101	0.0500	0.1600	0.0583

Table 2.12. 95% VaR forecasts

	Exceedances			DQ Test Rejections		VaR
	Mean	Median	SD	0.01	0.05	Mean
Constant	0.0084	0.0061	0.0117	0.3800	0.4100	0.1935
GARCH	0.0133	0.0119	0.0066	0.1400	0.2000	0.1039
QR	0.0097	0.0066	0.0125	0.5400	0.5900	0.1852
QCNN	0.0599	0.0576	0.0280	0.9500	0.9600	0.0676
Joint QCNN	0.0115	0.0119	0.0056	0.0900	0.1600	0.1023

Table 2.13. 99% VaR forecasts

	Exceedances			VaR
	Mean	Median	SD	Mean
Constant	0.0023	0.0007	0.0052	0.3764
GARCH	0.0060	0.0053	0.0041	0.1375
QR	0.0041	0.0013	0.0075	0.3630
QCNN	0.0249	0.0212	0.0186	0.1079
Joint QCNN	0.0023	0.0013	0.0026	0.2062

Table 2.14. 99.9% VaR forecasts

The quantile convolutional neural network trained on individual stocks did not perform well. It produced way too large VaR exceedance rates. The other methods delivered exceedance rates close to the targeted values. In most cases, the jointly trained convolutional network managed to produce reasonably accurate exceedance rates with lower average VaR values than

other methods. It is a desirable property, since it may allow for more efficient risk management. The DQ test is rejected for fewer stocks in case of the jointly trained network than the benchmark models, and the exceedance rates has the least variation for our proposed model. All these results show that the one-dimensional dilated causal quantile convolutional neural network can produce competitive forecasts of Value-at-Risk.

Summary and Conclusions A new method was proposed for Value-at-Risk forecasting. It is a one-dimensional convolutional neural network trained to forecast quantiles. When trained jointly on multiple stocks' return series, it produced better forecasts than the benchmarks.

The results show that convolutional neural networks can make good quantile forecasts, and that, taking advantage of transfer learning, deep neural networks can contribute to VaR-forecasting.

2.1.5 Summary of Risk Forecasting Studies

We used deep learning to produce value and direction forecasts of various estimates of the unobservable volatility. We used our forecast to compare the predictability of the different estimated volatility series. We also used a transfer learning approach to train a volatility forecasting model on multiple assets' data, and have shown that it can improve upon asset-specific models. We tried to lighten up the black box of deep learning, and study the stylized facts of volatility. We proposed a new method for Value-at-Risk forecasting, using deep (transfer) learning and quantile regression, and we got fairly promising results.

2.2 Mortality Forecasting

In a strict sense, the prediction of mortality rates is not a financial forecasting problem, however it is closely related, since it has important actuarial applications.

In case of financial risk forecasting, we had difficulties applying deep learning models to datasets with daily observations. With mortality forecasting, we have an even more difficult case. We have no better than annual observations, and (for most countries) mortality rates have been recorded for less than a century. That is, considering a single series of mortality rates,

we have very small data—not something that allows for using deep learning models. However, we have many similar series. Female and male mortality rates are recorded separately, and they are broken down by age groups. That is, we have mortality rate time series for all ages (from 0 to 110+ years) and for both sexes. And we have data from various countries. In this respect, it is not that small data. We may use a joint deep learning approach—like we did with stock volatilities or VaR—if all these mortality rate series show sufficiently similar behavior. If joint training works, deep learning methods may be competitive, otherwise there is no reason to apply neural networks to mortality rate forecasting.

2.2.1 Lee-Carter Model

The Lee-Carter model [Lee and Carter, 1992] is a standard long-term mortality rate forecasting method that uses age-specific mortality rates to forecast age-specific mortality rates. That is, it forecasts a matrix of mortality data—in a sense, this itself is a joint forecasting approach. Yet, it is simple and intuitive.

The Lee-Carter model estimates the logarithm of the mortality rate $m_{x,t}$ of age group x at time t

$$ln(m_{x,t}) = a_x + b_x k_t + e_{x,t}. \tag{2.9}$$

The vector of average age-specific mortality rates is denoted by a_x, k_t is a mortality trend, that is, the general change of mortality in time, b_x measures how the mortalities of different age groups change with changes of the mortality trend, while $e_{x,t}$ is the corresponding error term. In order to have a unique solution, we need two constraints: $\sum_x b_x = 1$ and $\sum_t k_t = 0$.

We can formulate this as a least squares problem, that is, we need to minimize the squared error between the true log-mortality rates and our estimates

$$\sum_x \sum_t \left[ln(m_{x,t}) - a_x - b_x \cdot k_t\right]^2. \tag{2.10}$$

The estimates of a_x are the mortality rates averaged over time

$$\hat{a}_x = \frac{1}{T} \sum_t ln(m_{x,t}). \tag{2.11}$$

The estimates of b_x

$$\hat{b}_x = \frac{U_{x,1}}{\sum_x U_{x,1}}, \tag{2.12}$$

and the estimates of k_t

$$\hat{k}_t = V_{t,1} D_{1,1} \sum_x U_{x,1} \tag{2.13}$$

are obtained from the singular value decomposition (SVD) of the centered log-mortality matrix $M_{x,t} = ln(m_{x,t}) - \hat{a}$.

SVD factorizes the (centered) log-mortality rates in the form of $M = UDV^T$. The original Lee-Carter model uses only the largest singular value ($D_{1,1}$) to estimate the parameters. Time series forecasting methods can be used to produce estimates of the future k_t values. ARIMA models are often used to make these forecasts. The original Lee-Carter article [Lee and Carter, 1992] proposed a random walk with drift model. Once we have predictions of k_t for the future, we can use them together with the a_x and b_x values to forecast mortality rates

$$\hat{m}_{x,T+t} = exp(\hat{a}_x + \hat{b}_x \cdot \hat{k}_{T+t}),\ t = 1, 2, \ldots. \tag{2.14}$$

The Lee-Carter model was developed for US mortality rates, but it is now used for many countries.

Applying the Lee-Carter model to Hungarian Mortality Rates While the above-discussed Lee-Carter model is popular and widely applied, it does not necessarily produce uniformly high quality forecasts for all countries. The original (and fairly simple) approach may prove to be insufficient for countries that experienced some kind of mortality shock. This is the case for post-socialist countries, and especially for Hungary, where mortality rates expressed strange historical patterns. Large political changes impose great costs on the society (even when the change is positive)—the former communist countries experienced a mortality crisis that claimed 10 million lives between 1990 and 2000 due to acute stress experienced by the weakest parts of the population [Cornia and Paniccià, 2000]. The crisis was relatively quickly solved in central European countries. In Hungary, male life expectancies decreased since the mid 1960s. The trajectory of Hungarian mortality statistics deviated from that of most developed countries, where life expectancy typically increased 0.2-0.3 year annually in that

period [Bálint and Kovács, 2015]. Other East-Central European countries experienced stagnating or moderately increasing life expectancies. That is, Hungarian mortality rates are a special case, which is well-documented in mortality forecasting literature. For example, Scheiring et al. [2018] produced a systemic review of 29 articles on the social determinants of the puzzling mortality history of Hungary.

Baran et al. [2007] used the Lee-Carter model to produce forecasts of Hungarian mortality rates. They found that the original model is hardly applicable, however, a few modifications allow for successful application: using only post-1989 data and a higher order generalization of the Lee-Carter model, they managed to produce reasonable forecasts. We tried to reproduce the forecasts to see if they worked and if the modifications are still necessary, and to produce new forecasts for the future.

The higher order Lee-Carter estimates are produced by keeping more than just one singular value from the SVD [Booth et al., 2001]. That is, we model not one but multiple mortality trends (k_t)

$$ln(m_{x,t}) = a_x + b_x^{(1)}k_t^{(1)} + b_x^{(2)}k_t^{(2)} + b_x^{(3)}k_t^{(3)} + e_{x,t}. \tag{2.15}$$

These different trends can be forecasted separately, using time series methods. By using multiple trends, we allow for more flexibility, and we may expect it to work better for peculiar mortality rates than the original model.

In case of Baran et al. [2007], the original model resulted in increasing morality rate estimates for middle aged men. Even the higher order models produced increasing estimates, which was not reasonable for a well developing country. Therefore, Baran et al. [2007] chose to drop all mortality rate observations before 1989, and used the remaining few years to fit the model. The resulting model produced more reasonable forecasts.

We used the (now available) longer period for two purposes: to evaluate the accuracy of the previous forecasts, and to produce new forecasts for the future. The dataset was downloaded from the Human Mortality Database (HMD), mortality.org [University of California, Berkeley (USA), and Max Planck Institute for Demographic Research (Germany)]. HMD collects original mortality data from national statistical offices, with the aim to provide researchers easy access to comparable mortality statistics. Hence, the dataset we obtained from mortality.org relies on the same source of data as Baran et al. [2007], the Hungarian Central Statistical Office.

The results of the evaluation are displayed in Table 2.15, the figures

of the forecasts are available in Figure 2.9. The errors justify the use of the reduced dataset—the model that did not use data from the socialist era produced clearly more accurate forecasts.

Metric	1950-2003 model	1989-2003 model
RMSE	0.0071	0.0040
SMAPE	24.35	17.47
MedAE	0.0058	0.0028

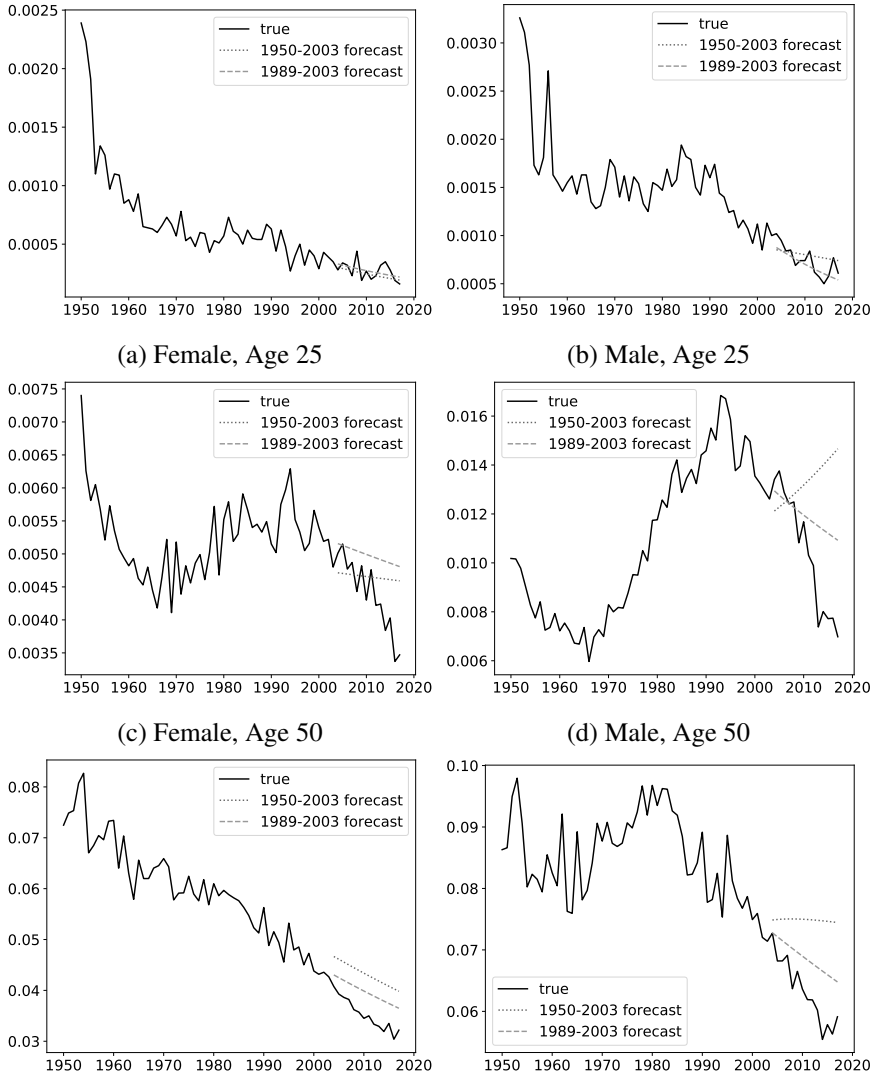
Table 2.15. Evaluation metrics for 2004-2017, averaged over all ages (root mean squared error, symmetric mean absolute percentage error, median absolute error)

Thus, we only used mortality rates from 1989 to refit the model, but we had twice as much data as Baran et al. [2007]. Figures of the Lee-Carter parameters are displayed in Figures 2.10 and 2.11. We used the Box-Jenkins method to find time series models to forecast the mortality trends (k_t). We used the Ljung-Box test [Ljung and Box, 1978] to test if the k_t values exhibit serial correlation, the results are shown in Table 2.16. The autocorrelation and partial autocorrelation plots are displayed in Figure 2.12.

	$k^{(1)}_{male}$	$k^{(2)}_{male}$	$k^{(3)}_{male}$	$k^{(1)}_{female}$	$k^{(2)}_{female}$	$k^{(3)}_{female}$
Q	97.224	7.4648	15.72	102.49	5.072	6.7178
$p-value$	$2.22e^{-16}$	0.681	0.1079	$< 2.2e^{-16}$	0.8863	0.7518

Table 2.16. Ljung-Box test results

In case of $k^{(2)}$ and $k^{(3)}$, we could not reject the null hypothesis that the data are independently distributed. That is, the higher order trends can be considered white noises for both female and male mortality rates. It means that we can model Hungarian mortality rates with a single trend, so we can use the original Lee-Carter model, there is no need for the extension. Furthermore, we found that $k^{(1)}_{female}$ can be described by an ARIMA(0,1,0) with drift—the same model that was originally used in the Lee-Carter model for the United States. However, we found an ARIMA(1,1,0) with drift model more suitable for female mortality rates. Having performed Ljung-Box test for independence, and Shapiro-Wilk test for normality, we can assume that the residuals of both models behave as white noise. The residuals' normal QQ plots are displayed in Figure 2.13. The fitted time series models are the



(a) Female, Age 25

(b) Male, Age 25

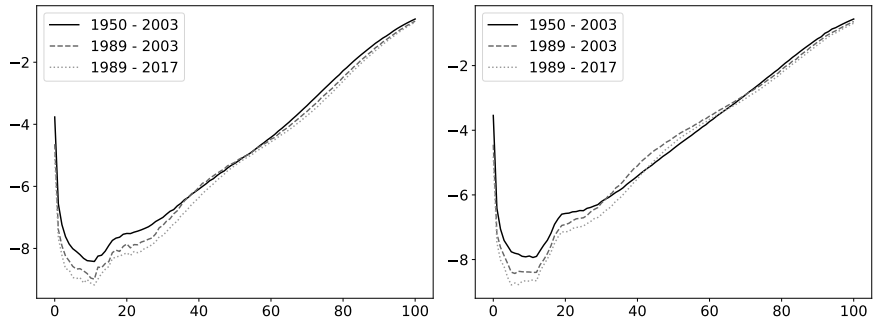
(c) Female, Age 50

(d) Male, Age 50

(e) Female, Age 75

(f) Male, Age 75

Figure 2.9. Forecasts of the Lee-Carter model for Hungary in the 2004-2017 period



(a) Female (b) Male

Figure 2.10. Age specific mortality from the Lee-Carter model (a_x)

following (standard errors in parentheses):

- $k_t^{female} = -2.4842\ (0.4004) + k_{t-1}^{female} - 0.7975\ (0.1421) \times (k_{t-1}^{female} - k_{t-2}^{female}) + \delta_t^{female}$,
- $k_t^{male} = -2.5842\ (0.6348) + k_{t-1}^{male} + \delta_t^{male}$.

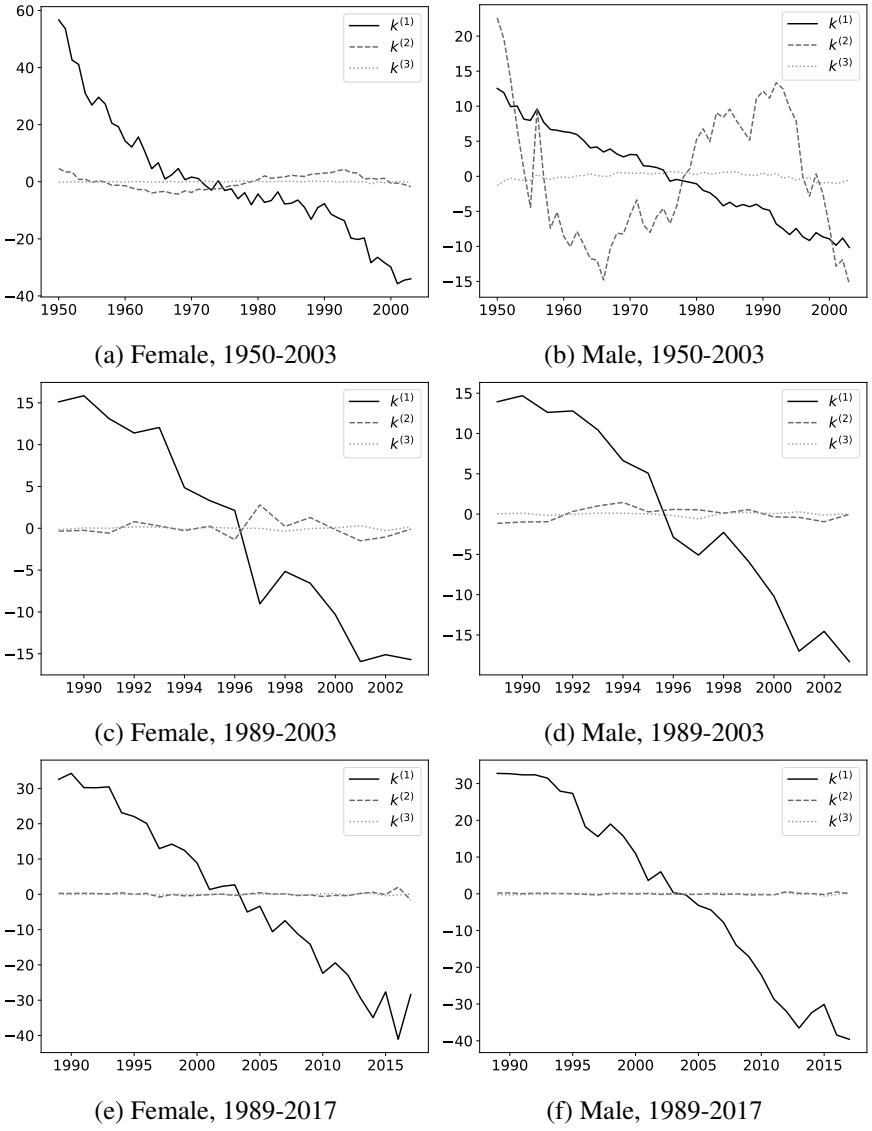
Mortality rate forecasts for the 2018-2047 period are displayed in Figure 2.14.

The main findings of our experiments with the Lee-Carter model are that we confirmed that using only the period after the regime change is beneficial for model fitting, and that we have shown that the original Lee-Carter model is becoming applicable as mortality rates are normalizing.

Applying the Lee-Carter model required us to reduce the dataset size. We also tried the other direction: increasing the dataset (and model complexity, as well). This approach is described in the following section.

2.2.2 Mortality Rate Forecasting with Recurrent Neural Networks

We performed a large-scale mortality rate forecasting study using neural networks. The time series of mortality rates are very short—this is why simple approaches (such as the Lee-Carter model) are often preferred. However, if there is some generality in mortality rates, we may apply some kind of



(a) Female, 1950-2003

(b) Male, 1950-2003

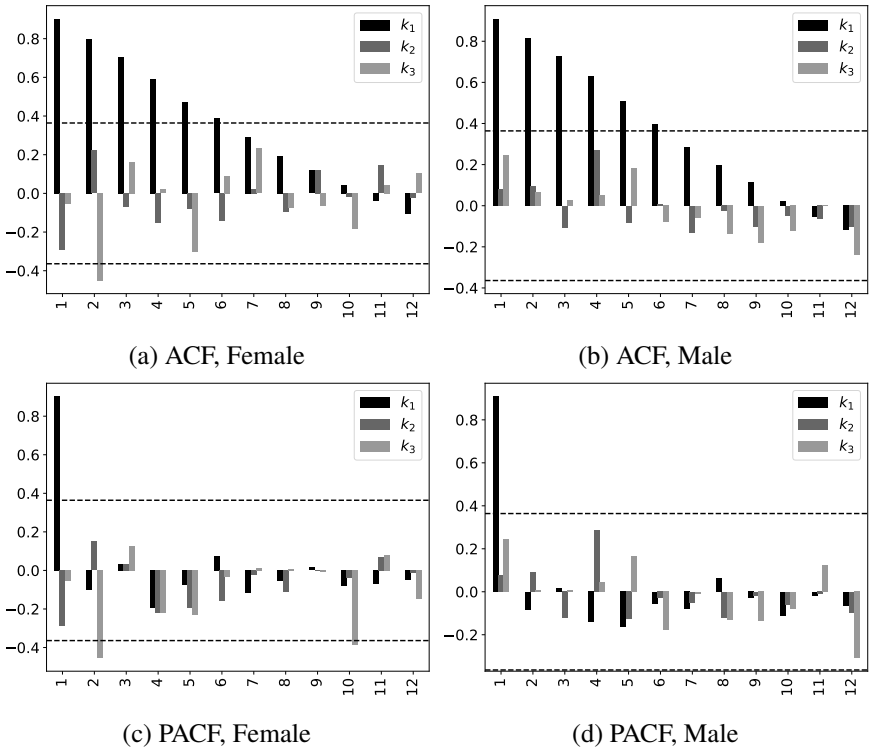
(c) Female, 1989-2003

(d) Male, 1989-2003

(e) Female, 1989-2017

(f) Male, 1989-2017

Figure 2.11. Mortality changes over time from the Lee-Carter model ($k_t^{(i)}$)



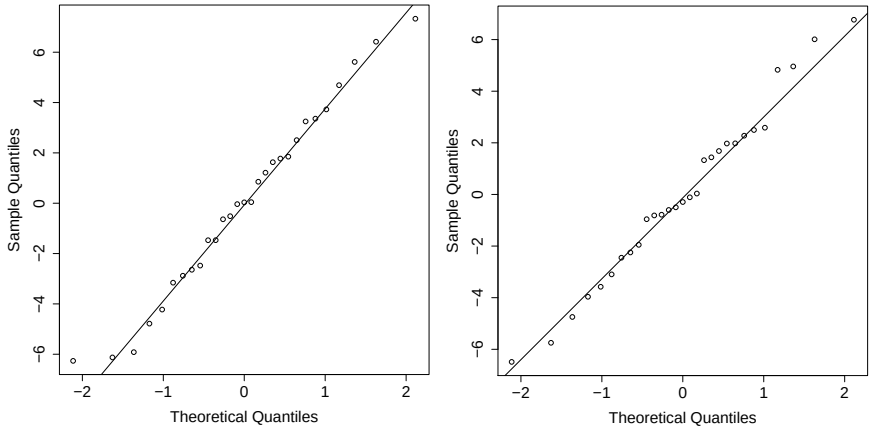
(a) ACF, Female

(b) ACF, Male

(c) PACF, Female

(d) PACF, Male

Figure 2.12. ACF and PACF plots of the mortality trends



(a) $k^{(1)}_{female}$ (b) $k^{(1)}_{male}$

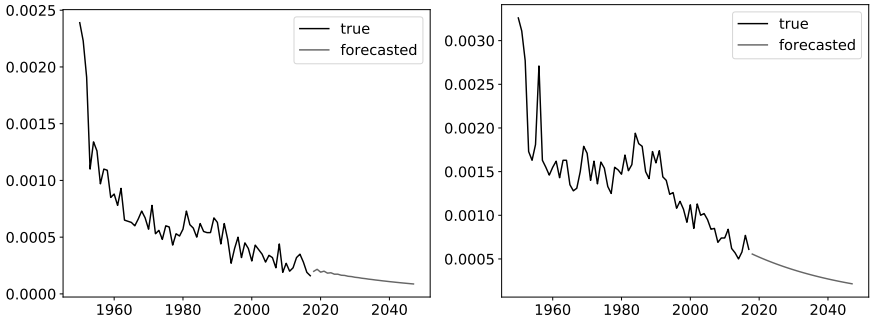
Figure 2.13. Normal QQ plots of residuals

transfer learning, and use a larger and more diverse dataset to fit flexible models.

We used a (long short-term memory) recurrent neural network architecture to forecast mortality rates in different countries, and compared its performance to the Lee-Carter model. We used the last 10 years of available mortality data to evaluate the forecast performance.

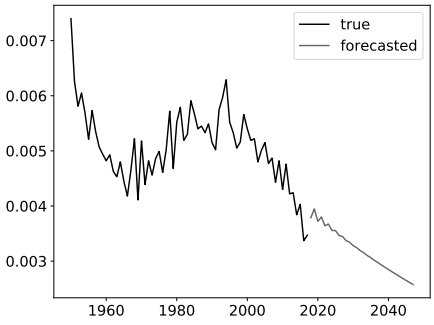
Jointly Training Recurrent Neural Networks Mortality rates are observed annually, this is why the available time series are very short—in many cases, less than 100 observations. There is no reason to apply sophisticated machine learning models to such small data. In order to do so, we need to transfer knowledge from other time series.

Essentially, the Lee-Carter model can itself be considered as an example of transfer learning. (At least, with a little exaggeration.) It assumes that there is a general trend in mortality rates that can be used to forecast the mortality rates of different ages. We obtain knowledge of the mortality trend from the history of an entire population, and transfer that knowledge to the individual time series of mortality rates by age. The Lee-Carter model assumes that there is some general trend in the mortality rates in different age groups. We need a similar assumption of generality, in order to apply a

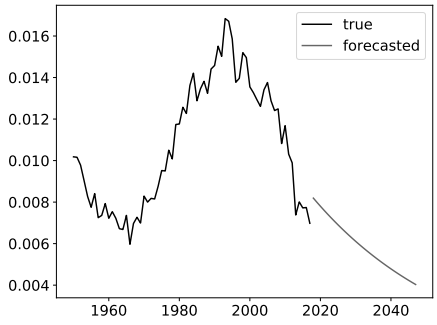


(a) Female, Age 25

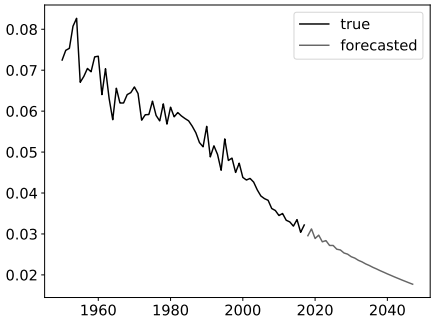
(b) Male, Age 25



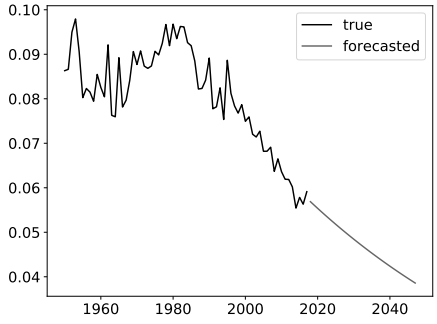
(c) Female, Age 50



(d) Male, Age 50



(e) Female, Age 75



(f) Male, Age 75

Figure 2.14. Forecasts for future Hungarian mortality rates from the re-fitted Lee-Carter model

jointly trained machine learning model.

The Lee-Carter model learns a common trend across different ages. This is a well-proven idea, but we can go way further. We can train an even more general mortality rate forecasting model. We could use not only observations from different age groups, but also from different sexes. Female and male mortality rates show similar patterns, but may have experienced different shocks, so knowledge extracted from one may help forecasting the other. We could also use mortality rates from different countries. It seems possible that we could learn from the mortality rate history of one nation to forecast that of another.

We trained long short-term memory networks purely on mortality rates. That is, we did not use country, sex or age specific data or indicators, only the time series of mortality rates.

Neural Network Architecture As in most of our studies, we did not optimize the hyperparameters of the neural network, we just aimed to find a resonable size and reasonable settings. We used a one-layer neural network with 8 LSTM units, followed by a dense layer with a single unit with linear activation. The mortality rate series were unrolled to 16 timesteps, and were fed to the algorithm in 128-item batches. The cell states and hidden states of the network were reinitiated in each batch.

The dataset was standardized (the overall mean was subtracted from each value, and the differences were divided by the overall standard deviation of the training set). The network was trained with the Adam optimizer [Kingma and Ba, 2014] with a learning rate of 0.001, for 300 epochs.

We produced 10-year recursive forecasts: single-year forecasts were fed back to the network to generate the subsequent prediction.

Data The datasets for the study were obtained from mortality.org [University of California, Berkeley (USA), and Max Planck Institute for Demographic Research (Germany)]. We downloaded historical mortality rates for 40 countries, however, we had to exclude 5 countries with too little data or too many missing observations. We used all available mortality rates of the 35 remaining countries. The length of available mortality history varies greatly from country to country. Some countries have no more than a few decades of recorded mortality rates, while others have observations back from the 19th century. The most recent observations may also differ, since

the datasets of different countries are updated at different times. The countries and the time periods of the available mortality rates are displayed in Table 2.17. We used all but the last 10 years of available mortality rates for training the models, the last 10 years were used for evaluation.

Results We produced multi-step out-of-sample forecasts for the 10-year evaluation period for each country, with different forecasting models: our proposed LSTM model (trained on different subsets of the data), and the benchmark Lee-Carter model.

We used various evaluation metrics: root mean squared error (RMSE), mean absolute error (MAE), median absolute error (MedAE), symmetric mean absolute percentage error (SMAPE), mean error (ME). We aimed for a thorough comparison.

Our first LSTM-based forecasting attempts outperformed the benchmark Lee-Carter model in terms of absolute errors (e.g., RMSE), but clearly underperformed it in terms of relative errors (SMAPE). The reason is that the LSTM produced poor forecasts in the low-mortality (i.e., young) age groups. Since the network was trained with mean squared error loss function, it disregarded the age groups that could not contribute much to the overall loss, even with relatively high errors. In order to avoid this undesirable behavior, we transformed the target variable, so that the network forecasts not the mortality rates, but their natural logarithms. (Some mortality series contained zero values, so we clipped the values at a lower limit of 1e-12.) The Lee-Carter model also models log-mortality rates, so this transformation makes the models even more comparable. The evaluation was performed on the inverse (i.e., exponential) transformed values, that is, the true and forecasted mortality rates.

The results of the evaluation are displayed in Table 2.18. Heatmaps of the errors are displayed in Figure 2.15 (by age), and Figure 2.16 (by country).

All LSTM models were trained using data from all age groups. Our first models used training data from only one country. That is, separate models were trained for each country. (Each forecasting model learned only from the mortality history of the given country.) These models are called LSTM Country. The Lee-Carter model was trained separately for each country, using all available data.

On average, LSTM Country outperformed the Lee-Carter model ac-

country code	first year	last year
aus	1921	2014
aut	1947	2017
bgr	1947	2010
blr	1959	2016
can	1921	2011
che	1876	2016
cze	1950	2016
dnk	1835	2016
esp	1908	2016
est	1959	2017
fin	1878	2015
fra	1816	2016
gbr	1922	2016
grc	1981	2013
hun	1950	2017
irl	1950	2014
isl	1838	2016
isr	1983	2016
ita	1872	2014
jpn	1947	2016
ltu	1959	2017
lux	1960	2014
lva	1959	2017
nld	1850	2016
nor	1846	2014
nzl	1948	2013
pol	1958	2016
prt	1940	2015
rus	1959	2014
svk	1950	2017
svn	1983	2017
swe	1751	2017
twn	1970	2014
ukr	1959	2013
usa	1933	2016

Table 2.17. Countries and time periods used in the neural network-based mortality forecasting study. The countries are identified by the same country codes at mortality.org.

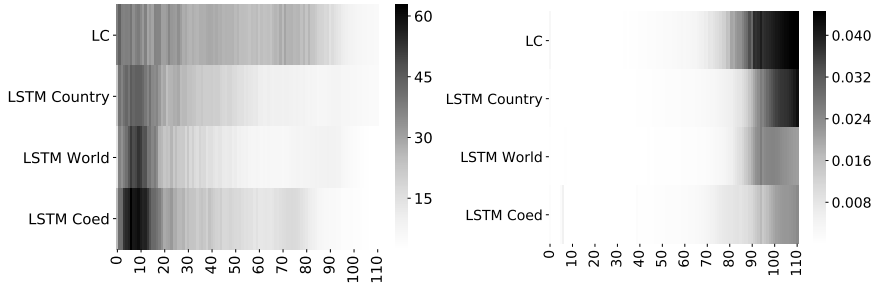
cording to all metrics. The mean error is positive for both the LSTM and the LC model, so it seems that the forecasts are slightly positively biased. (This bias, too, is larger for the LC model.) The LSTM produced lower RMSE values in all age groups, and in 27 countries (77% of all countries). Even though we modeled log-transformed values, the SMAPE still remained lower for the younger age groups. However, it also applies to the Lee-Carter model, as is shown in Figure 2.15. The LSTM produced better SMAPE values for 97 ages (87% of all), and for 33 countries (94%).

We have computed the correlations between the error (RMSE) and the length of available mortality rate history (years) in the given country. In case, of the LSTM model, we found a negative correlation (-0.25), while in case of the LC model, we found a positive correlation (0.15). Though these values are not particularly large in absolute terms, it suggests that the neural network works better for countries with more recorded mortality rates (i.e., with more data). It is not surprising, since neural networks require large amounts of training data—preferably, much more than what was available in these experiments. This conjecture leads us to two conclusions. First, whether or not the LSTM outperforms the LC in this study, the former holds more promise for the future (at least, for the distant future). Second, it seems reasonable to try expanding the dataset (for example, by jointly training the model on data from multiple countries).

We trained a Long Short-Term Memory model on all countries' data (called LSTM World). It was a bold move. By jointly training the model on multiple countries' data, we can substantially increase the size of the training set, which is clearly preferable. However, it makes our dataset more heterogeneous. Even though mortality rates show similar patterns in different countries, they are also highly dependent on local conditions (such as the political environment). We couldn't know in advance if it would improve the model. The differences in the countries could mess the model up and lead to poor predictions, or the increased dataset together with LSTM's flexibility could extract better patterns and deliver better forecasts. The latter proved to be true.

LSTM World produced better errors than either the LC or the LSTM Country model. (Except for the mean error, which was lower for LSTM Country.) In terms of RMSE, LSTM World outperformed LSTM Country in 28 countries (80%), and 103 age groups (93%).

For training our final model, we have expanded the dataset even further:



(a) SMAPE (b) RMSE

Figure 2.15. Errors by age from different forecasing models

beyond the total, we used the female and male mortality rates too. This model is called LSTM Coed. (Still, forecasts were only produced for the total population, so that they are comparable to the previous predictions.) This improved most errors (compared to LSTM World), however, it produced higher relative errors (the SMAPE is higher than that of the LSTM Country model). According to RMSE, LSTM Coed outperformed LSTM World in 26 countries (74%), but only in 24 ages (22%).

metric	LC	LSTM Country	LSTM World	LSTM Coed
RMSE	0.0115	0.0076	0.0058	0.0055
MAE	0.0109	0.0069	0.0051	0.0047
MedAE	0.0108	0.0067	0.0049	0.0045
SMAPE	24.85	18.02	15.82	20.76
ME	0.0085	0.0027	0.0037	0.0026

Table 2.18. Averaged evaluation metrics

Some example forecasts are displayed in Figure 2.17, for 2 arbitrarily chosen countries, and 3 arbitrarily chosen ages. The forecasts typically follow similar directions, but there are some notable differences and patterns. One of the example countries is the United States—the country for which the Lee-Carter model was originally developed. While it may not be evident from the plotted forecasts, Figure 2.16 shows that the LC model worked relatively better in the United States than in most countries. It suggests that, although it is widely applied, the Lee-Carter model may not necessarily be

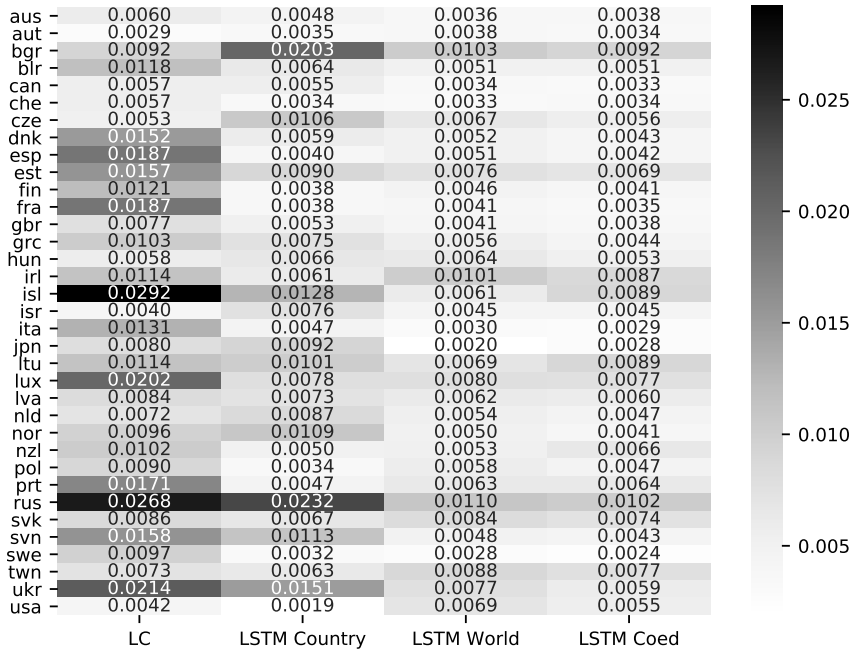


Figure 2.16. RMSE by country from different forecasing models

just as competitive everywhere. The other example is (the above discussed) Hungary, with its strangely evolving mortality rates. Figure 2.17c clearly shows how the Lee-Carter model failed for Hungary's middle-age population, while the LSTM models produced fairly reasonable forecasts.

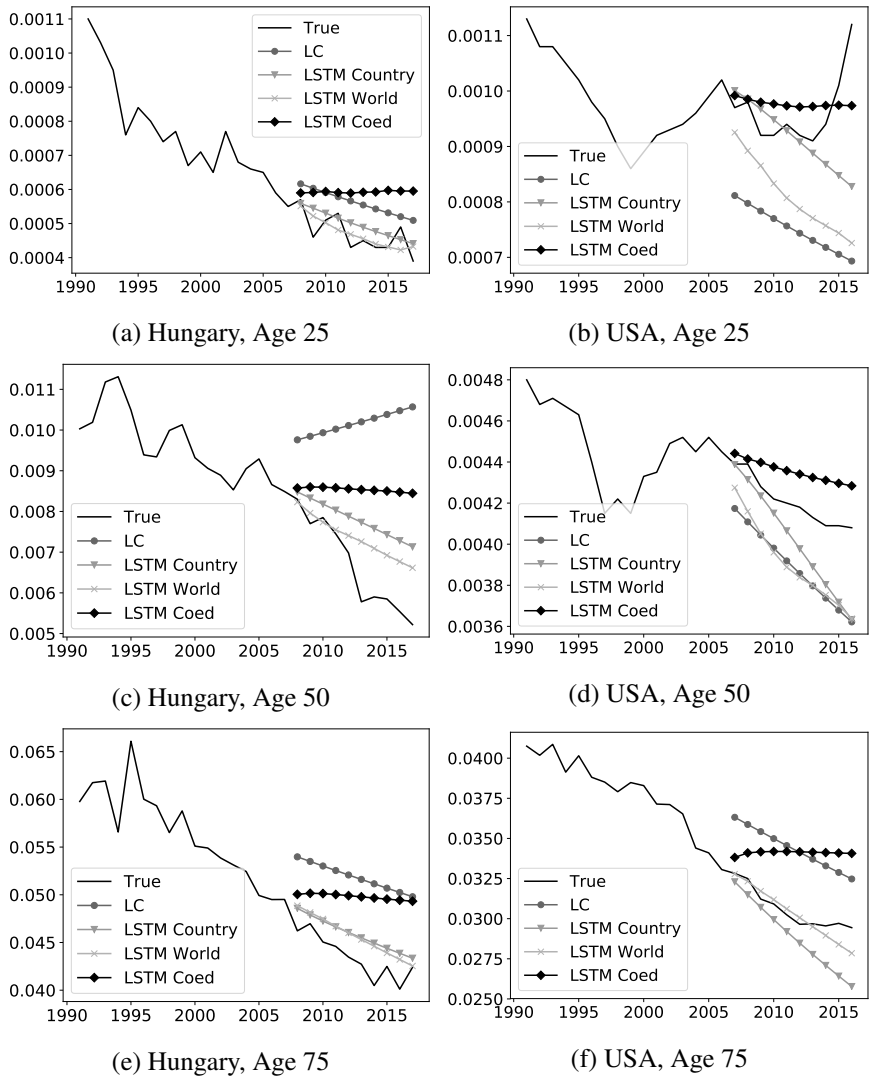
Improving the Benchmark Our results, so far, have shown that the LSTM model can outperform the Lee-Carter model fairly consistently, particularly when it is trained jointly on multiple countries' mortality rate history. However, we only used the original (basic) Lee-Carter model as a benchmark. It has several extensions and modifications, which may work better in certain cases. In order to better evaluate the competitiveness of our recurrent neural networks, we tried to improve the benchmark Lee-Carter model.

One possible extension of the model is to keep multiple trends—it was discussed in the previous section, see (2.15). The original Lee-Carter model keeps only the largest singular value from the SVD to find a single trend. This is a strong simplification: while the model is fairly interpretable and intuitive, it ignores some information—and in some cases, it may ignore too much information. By keeping multiple trends, we allow for a better fit, which may be beneficial in some cases. Naturally, when we build a model with multiple trends, we need multiple separate time series models to forecast them.

We chose to use a third order Lee-Carter model, that is, we used 3 trends.

Another modification (and possible improvement) is the choice of time series model. The original Lee-Carter model [Lee and Carter, 1992] used a random walk with drift model to forecast k_t. It may be the best choice for forecasting the mortality trend of the United States, but it is not necessarily the best choice for all countries. It seems reasonable to revise the choice of time series model for each country. Furthermore, when we extend the Lee-Carter model to account for multiple trends, we need to find appropriate time series models for the new components as well.

Though we constrained our time series models to the ARIMA-family, it seemed a tedious work to manually optimize the parameters for each k_t series in our study. We used an automated method to choose the model parameters [Hyndman et al., 2007]. It is available in the forecast package of R. This method uses successive KPSS tests for choosing the order of differencing, and a step-wise algorithm to choose the number of AR and



(a) Hungary, Age 25

(b) USA, Age 25

(c) Hungary, Age 50

(d) USA, Age 50

(e) Hungary, Age 75

(f) USA, Age 75

Figure 2.17. Example forecasts for two arbitrarily chosen countries

MA terms based on AIC. The chosen ARIMA models are available in Table 2.19.

We used 3 extensions of the Lee-Carter model. The first one uses auto ARIMA (LC Auto). The second one uses multiple trends (LC Higher). The third one uses both (LC Auto Higher).

Table 2.20 shows the performance of the improved benchmark models. The automatic ARIMA models brought only slight improvements to the errors. However, the third order model performed considerably better than the basic Lee-Carter model. (It still underperforms most LSTM models, but the difference is much smaller.)

Summary and Conclusions We applied a Long Short-Term Memory recurrent neural network to mortality rate forecasting, and compared its performance to the well-known and well-proven Lee-Carter model. Applying LSTMs to mortality rate forecasting is not a straightforward task, since we have very small datasets. Hence we have trained our recurrent neural networks jointly on multiple time series: the mortality rates of different ages, different sexes, and different countries.

We produced forecasts for 35 countries and 111 age groups using a rather simple neural network architecture, and evaluated the predictions on the last 10 years of available mortality rates. The LSTM models made more accurate forecasts than the Lee-Carter model, especially when trained globally. (That is, jointly trained on multiple countries' data.) Using the separate mortality rates of the 2 sexes did not bring a clear additional improvement. Our LSTM models had less advantage over some extensions of the Lee-Carter model.

Our study has a few shortcomings. We only considered the out-of-sample forecasting performance of the models, ignoring model complexity. While the basic Lee-Carter model underperformed the neural networks in accuracy, its simplicity and interpretability is a clear (but unmeasured) benefit. Another shortcoming is that we used a single forecast horizon (10 years). The models may perform better or worse in the more distant future.

Our approach could be improved in different ways. We used a simple recurrent neural network with LSTM units, with no systematic hyperparameter optimization. More sophisticated recurrent neural network architectures could be applied, and convolutional networks could also be used in a similar manner. Another (im)possibility for improvement is using more data. (Even

country	LC			LC Higher								
	k			k_1			k_2			k_3		
	AR	I	MA	AR	I	MA	AR	I	MA	AR	I	MA
aus	0	1	1	0	1	1	1	2	1	1	0	0
aut	0	1	0	0	1	0	0	2	2	1	0	0
bgr	0	2	3	0	2	3	0	1	1	1	0	0
blr	0	1	0	0	1	0	1	0	0	1	0	0
can	0	1	0	0	1	0	1	2	1	2	0	0
che	0	1	1	0	1	1	0	1	1	1	1	3
cze	0	1	0	0	1	0	2	2	1	1	0	0
dnk	1	1	1	1	1	1	0	1	1	1	1	1
esp	1	1	0	1	1	0	0	1	1	1	0	0
est	0	1	3	0	1	3	1	1	0	0	1	0
fin	0	1	0	0	1	0	0	1	1	1	1	1
fra	1	1	1	1	1	1	1	1	1	0	1	1
gbr	2	1	2	2	1	2	1	2	1	2	0	2
grc	1	1	0	1	1	0	0	0	0	0	0	0
hun	1	1	2	1	1	2	1	1	1	1	2	1
irl	2	1	0	2	1	0	0	2	1	2	0	0
isl	2	1	0	2	1	0	2	1	1	0	0	0
isr	0	1	0	0	1	0	0	0	0	0	0	0
ita	0	1	0	0	1	0	3	0	2	1	1	1
jpn	0	2	3	0	2	3	2	2	1	2	0	1
ltu	0	1	1	0	1	1	2	0	1	1	0	1
lux	2	1	0	2	1	0	1	2	1	0	0	0
lva	0	1	0	0	1	0	2	0	0	3	0	2
nld	1	1	1	1	1	1	0	1	2	0	1	1
nor	1	1	0	1	1	0	0	1	1	0	1	1
nzl	1	1	0	1	1	0	1	1	0	2	0	0
pol	1	1	0	1	1	0	0	1	0	2	0	0
prt	2	1	2	2	1	2	0	2	1	1	1	0
rus	0	1	1	0	1	1	1	0	2	1	0	0
svk	0	1	0	0	1	0	0	1	0	0	0	0
svn	0	1	0	0	1	0	0	1	0	0	0	0
swe	1	1	2	1	1	2	3	1	3	2	1	1
twn	0	1	0	0	1	0	1	0	0	1	0	0
ukr	0	1	0	0	1	0	2	0	1	1	0	0
usa	1	1	0	1	1	0	0	2	1	2	1	2

Table 2.19. Auto ARIMA parameters

metric	LC	LC Auto	LC Higher	LC Auto Higher
RMSE	0.0115	0.0113	0.0078	0.0079
MAE	0.0109	0.0107	0.0071	0.0072
MedAE	0.0108	0.0107	0.0070	0.0071
SMAPE	24.85	24.85	17.28	17.97
ME	0.0085	0.0083	0.0033	0.0035

Table 2.20. Averaged evaluation metrics (Lee-Carter variants)

our largest mortality dataset is very small for neural networks.) However, it is not easy, since we used the largest available mortality database. So we should wait, and, sadly, die, in order that neural networks can make very accurate mortality rate predictions for our descendants.

2.2.3 Summary of Mortality Forecasting Studies

We evaluated the Lee-Carter model's forecasting performance with Hungary's (rather strange) mortality rates, and we used the algorithm to produce new forecasts for the future. We found that the previously proposed modifications to the model were reasonable, and led to better results than the original model. We also found that in recent years, the original Lee-Carter model is getting better applicable. In a larger-scale study, we proposed a deep learning-based forecasing model, and compared its out-of-sample forecast performance to that of the Lee-Carter model. We found that using a jointly trained learning model, and mortality data from multiple countries, the deep learning-approach can deliver superior results, even having considered extensions of the Lee-Carter model.

Summary

The aim of this thesis was to combine methods from time series analysis and deep learning, retaining the benefits of both.

Time Series Forecasting with Deep Learning The first chapter describes the methodology of deep learning-based time series forecasting. The first part of the chapter covers the mechanisms behind deep neural networks together with two architectures that are particularly well-suited to time series forecasting (recurrent and convolutional networks). The second part of the chapter discusses the most common requirements of time series forecasting projects, and some practices that can help deep learning meet the expectations.

Neural Networks The term ”deep learning” covers machine learning methods using multi-layer neural networks. Two particular neural network architectures have inherent ability to learn sequential data. Recurrent neural networks (RNN) build memory, and use their previous states to make predictions. Convolutional neural networks (CNN) use slided shared weights to find local patterns. Gated recurrent units (LSTM, GRU), and dilated causal convolutional neural networks have delivered state-of-the-art solutions for various sequential learning problems, but are not yet fully exploited for time series forecasting.

Deep Time Series Forecasting Traditional machine learning fails to satisfy certain requirements of time series forecasting. One such requirement is the quantification of uncertainty. The single goal of supervised learning is to make accurate point predictions, to get as close to the target values as possible. Accordingly, deep neural networks typically produce

point predictions only, without any measure of confidence. This deficiency might be remedied by using bootstrapping or quantile regression to produce prediction intervals. Another unmet requirement is interpretability—neural networks are particularly difficult to interpret. While a complete understanding of the networks' operation seems unattainable, the importance of input features can be quantified, and may be quite valuable. The size of available time series data is often a difficulty, since deep learning algorithms need large amounts of data. This might be addressed by using transfer learning, that is, by acquiring knowledge from multiple time series. Time series forecasting also has some peculiar requirements regarding feature engineering and model evaluation. All these considerations are discussed in the first chapter, and are applied in the second.

Applications The second chapter describes our applications of the methodology. Our studies can be divided into two main categories: financial risk forecasting, and human mortality rate forecasting.

Directional Forecasts of Range-Based Volatility Estimates Volatility is unobservable, so we need to estimate it somehow. Range-based volatility estimates are rarely used despite their advantageous properties. Since, in theory, stock price volatility is forecastable, we aimed to compare the degree to which the different volatility estimates can be predicted. We primarily aimed to make and compare directional forecasts. We used a neural network with long short-term memory units to forecast whether the volatility estimates increase or decrease from one day to the other. Having analyzed 29 frequently traded stocks, and having compared the forecasts on a 3-year period, we found that the directions of range-based volatility changes are clearly more predictable than those of the close-to-close volatility. However, considering value forecasts and some modified estimators, the differences in predictability are less clear.

Volatility Forecasting with Transfer Learning Stock prices express common properties, which allows for building learning algorithms that can learn multiple assets' price history. We aimed to study if a jointly trained neural network can produce better volatility forecasts than the individually trained models. (Here we defined volatility as a moving standard deviation

of the returns.) We used a one-dimensional dilated causal convolutional neural network that was trained on the volatility history of hundreds of stocks, but not the stocks that we aimed to forecast. That is, the forecasting model did not know anything about the assets that it was applied to forecast. This jointly trained model was compared to individually trained convolutional neural networks and ARIMA models. Our proposed transfer learning approach produced clearly better results than the benchmark models, considering either value or direction forecasts. The results show that jointly trained neural networks can work very well for volatility forecasting, which allows for much better applications of deep learning in the field.

Inspecting the Stylized Facts of Volatility The stylized facts claim that volatility is related to returns, volumes, and even to certain external variables. We tried to investigate if the claimed relationships exist. We used a long short-term memory neural network with multiple explanatory variables. The models were trained on bootstrap sampled subsets of the time series. Bootstrapping allowed us to produce more stable forecasts, to measure the uncertainty of the predictions, and to quantify feature importances. We found that volatility and return are important predictors of future volatility. Trading volume had less importance in our forecasting model.

Value-at-Risk Forecasting With Quantile Convolutional Neural Networks We used a jointly trained one-dimensional dilated causal convolutional neural network to forecast Value-at-Risk. (Value-at-Risk is a quantile of the return distribution: it is a loss that is only exceeded with a predefined small probability.) We used the pinball loss function to forecast quantiles using the raw returns as input variables. We produced one-day Value-at-Risk forecasts for 100 US stocks with 3 different confidence levels. Several benchmark models were used: individually trained convolutional neural networks, linear quantile regressions, GARCH models, and constant quantile estimates. While various methods produced fairly accurate exceedance rates, the proposed quantile convolutional neural network typically delivered lower Value-at-Risk estimates. The dynamic quantile test also showed that the QCNN's forecasts were superior to the baseline models.

Mortality Rate Forecasting with Recurrent Neural Networks We applied the jointly trained deep learning approach to mortality rate forecasting, as well. Since the recorded time series of mortality rates are short, and generally long-term forecasts are required, it is a challenging task. While the individual time series are short, there are multiple similar time series, and for this reason, even traditional approaches use multiple series for model training. The widely applied Lee-Carter model aims to find a general mortality trend across different age groups. We proposed a recurrent neural network-based forecasting model that, similarly, learns from the mortality rates of different ages, but can also be extended to learn from the mortality history of both sexes or different countries. We produced 10-year recursive forecasts for 35 countries, and compared the predictions of the Lee-Carter model and the neural networks using various regression metrics. The recurrent neural network expressed superior forecasting performance, especially when it was trained on all countries' data. The proposed model outperformed the basic Lee-Carter model by a large margin. Some extended Lee-Carter models produced more competitive, but still worse forecasts than most neural networks. The results show that, even though mortality rate datasets are small, complex deep learning models can produce valuable forecasts.

Conclusions The aim of our research was to bridge the gap between deep learning and time series forecasting, and to prove the methods with useful applications. We produced various financial forecasting studies using recurrent and convolutional neural networks. Deep learning-based forecasting is not a novelty, but it is not very common either, since there are some difficulties regarding, for example, the size of data or the interpretability of the models. We tried to show that these difficulties can be overcome, and we managed to deliver competitive (or, at least, promising) forecasts for various tasks and datasets. We hope that the models and methods that we used will be useful, and that we have managed to contribute to this interesting field of research.

Összefoglaló (Summary in Hungarian)

A disszertáció és az azt megelőző kutatómunka célja az volt, hogy egyesítse az idősorelemzés és a mélytanulás módszereit, megtartva mindkét terület előnyeit.

Idősor-előrejelzés mélytanulással Az első fejezet a mélytanulás-alapú idősor-előrejelzés módszertanát ismerteti. A fejezet első része bemutatja a mélytanulás mögött meghúzódó mechanizmusokat, illetve két architektúrát, melyek különösen jól alkalmazhatók idősorok előrejelzésére (rekurrens és konvolúciós hálózatok). A fejezet második része az idősor-előrejelzési projektek leggyakoribb elvárásait tárgyalja, illetve olyan eljárásokat, melyek segítségével a mélytanulási módszerek jobban megfelelhetnek a követelményeknek.

Neurális hálózatok A "mélytanulás" elnevezés olyan gépi tanulási módszereket takar, melyek több rétegű neurális hálózatokat használnak. Van két olyan speciális hálózattípus, melyek eredendően jól tudnak modellezni szekvenciális adatokat. A rekurrens neurális hálózatok (RNN) memóriát építenek, és a korábbi státuszaikat is használják döntéseik meghozatalához. A konvolúciós neurális hálózatok (CNN) csúsztatott megosztott súlyokat használnak a helyi mintázatok felismeréséhez. A kapuk által vezérelt rekurrens egységek (LSTM, GRU) és a dilatált kauzális konvolúciós neurális hálózatok versenyképes eredményeket szállítottak különféle szekvenciális tanulási feladatok esetén, azonban az idősor-előrejelzés területén még nem lettek teljes mértékben kiaknázva a módszerekben rejlő lehetőségek.

Mély idősor-előrejelzés A hagyományos gépi tanulási megoldások tipikusan nem felelnek meg az idősor-előrejelzés bizonyos elvárásainak. Az egyik ilyen elvárás a bizonytalanság számszerűsítése. A felügyelt tanulás egyetlen célja az, hogy pontos pont-predikciókat készítsen, vagyis hogy a becslések minél közelebb legyenek a célzott értékekhez. Ennek megfelelően a mély neurális hálózatok általában csak pontbecsléseket készítenek anélkül, hogy számszerűsítenék a döntéseikben rejlő bizonytalanságot. Ez a hiányosság kezelhető bootstrapping vagy kvantilis regresszió alkalmazásával, mely módszerek segítségével intervallum előrejelzéseket készíthetünk. Egy további probléma az értelmezhetőség: a neurális hálózatok különösen nehezen interpretálhatók. Habár elérhetetlen célnak tűnik az, hogy tökéletesen megértsük a hálózatok működését, a bemeneti változók fontosságát tudjuk számszerűsíteni, és ez már önmagában sokat javíthat az értelmezhetőségen. Az elérhető adatok mennyisége szintén egy gyakori probléma, hiszen a mélytanuló algoritmusok tanításához nagy mennyiségű adatra van szükség. Itt az átviteli tanulás nyújthat segítséget, vagyis egy olyan eljárás, ami egyszerre több idősor adataiból képes tanulni. Az idősor-előrejelzésnek vannak további sajátosságai, melyeket figyelembe kell vennünk például a változók tervezése vagy éppen az eredmények értékelése során. Ezeket a szempontokat tárgyalja a disszertáció első és alkalmazza annak második fejezete.

Alkalmazások A második fejezet azokat az elemzéseinket mutatja be, melyekben az előzőekben ismertetett módszertant alkalmaztuk. Az elemzések két kategóriába sorolhatók: pénzügyi kockázat előrejelzése, és mortalitási ráták előrejelzése.

Terjedelemalapú volatilitás-becslések változási irányának előrejelzése A részvényárfolyamok volatilitása (elméletileg) előrejelezhető. A volatilitás azonban nem megfigyelhető, valamilyen módon becsülnünk kell. A terjedelemalapú volatilitás-becsléseket ritkán alkalmazzák, előnyös tulajdonságaik ellenére. Azt vizsgáltuk, hogy a volatilitás különböző becslései milyen mértékben igazolják az előrejelezhetőséget. Elsősorban a változások irányának előrejelzésére fókuszáltunk. Egy neurális hálózatot alkalmaztunk LSTM egységekkel annak előrejelzésére, hogy a volatilitás-becslések értéke növekszik vagy éppen csökken egyik napról a másikra. 29 gyakran kereskedett részvényhez

készítettünk előrejelzéseket egy 3 éves időszakra, és azt találtuk, hogy a terjedelemalapú becslések jobban előrejelezhetők mint a klasszikus, csak záró árfolyamokat használó becslés. Azonban (észszerűen) módosított becsléseket vizsgálva vagy a volatilitás értékét előrejelezve már sokkal kevésbé tapasztaltuk ezeket az eltéréseket.

Volatilitás-előrejelzés átviteli tanulással A részvényárfolyamok mutatnak olyan közös tulajdonságokat, melyek alapján érdemes lehet több részvény árfolyamtörténetét együttesen használni algoritmusok tanításához. Azt tanulmányoztuk, hogy egy együttesen tanított neurális hálózat képes lehet-e jobb eredményeket szállítani mint az egyedileg tanított modellek. (Itt a volatilitást a hozamok egy mozgó szórásával becsültük.) Egy dimenziós dilatált kauzális konvolúciós neurális hálózatot tanítottunk több száz részvény volatilitás-történetét felhasználva, kihagyva azonban azokat a részvényeket, melyek előrejelzésével próbálkoztunk. Vagyis az előrejelző algoritmus a tanítás során nem is találkozott azokkal a részvényekkel, melyeket előrejelezni volt hivatott. Ezt az együttesen tanított modellt vetettük össze egy egyedileg tanított konvolúciós hálózattal és egy ARIMA modellel. Az átviteli tanulásos megközelítés egyértelműen jobb eredményeket hozott mint az összehasonlításul használt módszerek, mind a volatilitás irányának, mind a volatilitás értékének előrejelzésében. Az eredmények azt mutatják, hogy az együttesen tanított neurális hálózatok jó volatilitás-előrejelzéseket készíthetnek, ez pedig lényegesen bővítheti a mélytanulás alkalmazási lehetőségeit a területen.

A volatilitás stilizált tényeinek vizsgálata A stilizált tények szerint a volatilitás kapcsolatos a hozamokkal, kereskedési volumenekkel, és bizonyos külső tényezőkkel is. Azt próbáltuk vizsgálni, hogy az állítólagos kapcsolatok valóban léteznek-e. LSTM hálózatot használtunk, több bemeneti változóval. A modelleket bootstrap mintavételezett részein tanítottuk a volatilitás idősorának. A bootstrapping segítségével stabilabb előrejelzéseket tudtunk készíteni, mérni tudtuk a predikciók bizonytalanságát, és számszerűsíteni tudtuk a változók fontosságát. Úgy találtuk, hogy a volatilitás és a hozam fontos előrejelzője a jövőbeli volatilitásnak. A kereskedés volumenét kevésbé találtuk fontos tényezőnek a modellben.

Value-at-Risk előrejelzése kvantilis konvolúciós neurális hálózatokkal Egy együttesen tanított egy dimenziós dilatált kauzális konvolúciós neurális hálózatot használtunk a Value-at-Risk előrejelzésére. (A Value-at-Risk a hozameloszlás egy kvantilise: egy olyan veszteség, aminél nagyobb veszteség csak egy előre megadott alacsony valószínűséggel következik be.) A "flipper" veszteségfüggvényt használtuk, hogy előrejelezhessük a vizsgált kvantilist, a nyers hozamokat használva bemeneti változóként. Egy napos Value-at-Risk előrejelzéseket készítettünk 100 egyesült államokbeli részvényhez, 3 különböző konfidencia szinttel. Többféle előrejelzéssel versenyeztettük a javasolt modellt: használtunk lineáris kvantilis regressziót, egy GARCH modellt, egy konstans kvantilis becslést és egy egyedileg tanított konvolúciós hálózatot is. Bár több modell is meglehetősen pontos túllépési arányokat produkált, a javasolt kvantilis konvolúciós neurális hálózat tipikusan kisebb Value-at-Risk értékeket produkált. A dinamikus kvantilis teszt szintén azt mutatta, hogy az együttesen tanított hálózatunk jobb előrejelzéseket produkált mint a többi vizsgált modell.

Mortalitási ráták előrejelzése rekurrens neurális hálózatokkal Az együttesen tanított mélytanulási megközelítésünket alkalmaztuk mortalitási ráták előrejelzésére is. Ez egy nehéz feladat, hiszen a nyilvántartott mortalitási ráták idősorai rövidek, és általában hosszútávú előrejelzések készítése szükséges. Habár az egyedi idősorok rövidek, több hasonló idősor áll rendelkezésre, így már hagyományosnak mondható módszerek is használnak egyszerre több idősort a modellek illesztéséhez. A széles körben alkalmazott Lee-Carter modell egy általános mortalitási trendet keres az adatokban, ami érvényes minden korcsoportra. Mi egy rekurrens neurális hálózatot használtunk, ami hasonlóképpen több korcsoport mortalitási adataiból tanul, de kiterjeszthető a két nem, illetve különböző országok adatainak használatára is. 10 éves rekurzív előrejelzéseket készítettünk 35 országra, és összehasonlítottuk a Lee-Carter modell és a neurális hálózat predikcióit többféle mérőszám használatával. A rekurrens neurális hálózat jobbnak bizonyult, különösen amikor az összes ország adatain tanítottuk. A javasolt modell jelentősen jobban teljesített az alap Lee-Carter modellnél. A Lee-Carter modell néhány kibővített változata már versenyképesebbnek bizonyult, de még azok is alulmúlták a legtöbb neurális hálózatunkat. Az eredmények azt mutatják, hogy bár a mortalitási ráták

kevés adatot képeznek, a komplex mélytanulási módszerek sikerrel alkalmazhatók és hasznos előrejelzéseket készíthetnek.

Következtetések Kutatásaink célja az volt, hogy áthidaljuk a mélytanulás és az idősor-előrejelzés területei között lévő szakadékot, és hogy a módszerek működőképességét hasznos alkalmazásokkal igazoljuk. Különféle pénzügyi előrejelzéseket készítettünk rekurrens és konvolúciós neurális hálózatok alkalmazásával. A mélytanulás-alapú előrejelzés nem újdonság, de nem is nagyon elterjedt, hiszen vannak hátráltató tényezők, mint például az adatmennyiség vagy éppen a modellek értelmezhetősége. Azt próbáltuk megmutatni, hogy ezek a nehézségek orvosolhatók, és sikerült versenyképes (vagy legalábbis biztató) eredményeket szállítanunk különböző előrejelzési feladatokhoz. Reméljük, hogy a modellek és módszerek hasznosnak fognak bizonyulni, és hogy sikerült hozzájárulnunk ennek az érdekes kutatási területnek a fejlődéséhez.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Hirotugu Akaike. A new look at the statistical model identification. In *Selected Papers of Hirotugu Akaike*, pages 215–222. Springer, 1974.

Torben G Andersen and Timo Teräsvirta. Realized volatility. In *Handbook of financial time series*, pages 555–575. Springer, 2009.

Torben G Andersen, Tim Bollerslev, Francis X Diebold, and Paul Labys. The distribution of realized exchange rate volatility. *Journal of the American statistical association*, 96(453):42–55, 2001.

Torben G Andersen, Oleg Bondarenko, Albert S Kyle, and Anna A Obizhaeva. Intraday trading invariance in the e-mini s&p 500 futures market. *Anna A., Intraday Trading Invariance in the E-Mini S&P*, 500, 2018.

Bernadett Aradi, Gábor Petneházi, and József Gáll. Volatility forecasting with 1-dimensional cnns via transfer learning. *arXiv preprint arXiv:2009.05508*, 2020.

Lajos Bálint and Katalin Kovács. Mortality. *Demographic Portrait of Hungary*, 2015.

Kasun Bandara, Christoph Bergmeir, and Slawek Smyl. Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. *Expert Systems with Applications*, 140: 112896, 2020.

Sándor Baran, József Gáll, Márton Ispány, and Gyula Pap. Forecasting hungarian mortality rates using the Lee-Carter method. *Acta Oeconomica*, 57(1):21–34, 2007.

Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.

Christoph Bergmeir and José M Benítez. On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191:192–213, 2012.

Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, 31(3):307–327, 1986.

Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. Machine learning strategies for time series forecasting. In *European business intelligence summer school*, pages 62–77. Springer, 2012.

Heather Booth, JH Maindonald, Len Smith, et al. Age-time interactions in mortality projection: Applying Lee-Carter to australia. *Working Papers in Demography*, (85), 2001.

Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Peter J Brockwell, Peter J Brockwell, Richard A Davis, and Richard A Davis. *Introduction to time series and forecasting*. Springer, 2016.

John G Carney, Pádraig Cunningham, and Umesh Bhagwan. Confidence and prediction intervals for neural network ensembles. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on*, volume 2, pages 1215–1218. IEEE, 1999.

Yitian Chen, Yanfei Kang, Yixiong Chen, and Zizhuo Wang. Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing*, 2020.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

François Chollet et al. Keras, 2015.

Peter F Christoffersen. Evaluating interval forecasts. *International economic review*, pages 841–862, 1998.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Yagmur Gizem Cinar, Hamid Mirisaee, Parantapa Goswami, Eric Gaussier, Ali Aït-Bachir, and Vadim Strijov. Position-based content attention for time series forecasting with sequence-to-sequence rnns. In *International conference on neural information processing*, pages 533–544. Springer, 2017.

Giovanni Andrea Cornia and Renato Paniccià. *The mortality crisis in transitional economies*. OUP Oxford, 2000.

T Dietterich. Ensemble learning. the handbook of brain theory and neural networks. In *:*. MIT Press, MA, 2002.

Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

R Glen Donaldson and Mark Kamstra. Forecast combining with neural networks. *Journal of Forecasting*, 15(1):49–61, 1996.

Christian L Dunis and Xuehuan Huang. Forecasting and trading currency volatility: An application of recurrent neural regression and model combination. *Journal of forecasting*, 21(5):317–354, 2002.

B Efron et al. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

Bradley Efron and Robert Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical science*, pages 54–75, 1986.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Robert Engle. Garch 101: The use of arch/garch models in applied econometrics. *Journal of economic perspectives*, 15(4):157–168, 2001.

Robert F Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the econometric society*, pages 987–1007, 1982.

Robert F Engle and Simone Manganelli. Caviar: conditional value at risk by quantile regression. Technical report, National Bureau of Economic Research, 1999.

Robert F Engle and Andrew J Patton. What good is a volatility model? *QUANTITATIVE FINANCE*, 1:237–245, 2001.

Eugene F Fama. Efficient capital markets: A review of theory and empirical work. *The journal of Finance*, 25(2):383–417, 1970.

Rui Fu, Zuo Zhang, and Li Li. Using lstm and gru neural network methods for traffic flow prediction. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 324–328. IEEE, 2016.

Mark B Garman and Michael J Klass. On the estimation of security price volatilities from historical data. *Journal of Business*, pages 67–78, 1980.

Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *Neural Nets WIRN Vietri-01*, pages 193–200. Springer, 2002.

Muriel Gevrey, Ioannis Dimopoulos, and Sovan Lek. Review and comparison of methods to study the contribution of variables in artificial neural network models. *Ecological modelling*, 160(3):249–264, 2003.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Tian Guo, Zhao Xu, Xin Yao, Haifeng Chen, Karl Aberer, and Koichi Funaya. Robust online time series prediction with recurrent neural networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 816–825. Ieee, 2016.

Peter R Hansen and Asger Lunde. A forecast comparison of volatility models: does anything beat a garch (1, 1)? *Journal of applied econometrics*, 20(7):873–889, 2005.

Tom Heskes. Practical confidence and prediction intervals. In *Advances in neural information processing systems*, pages 176–182, 1997.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

John Hull and Alan White. Value at risk when daily changes in market variables are not normally distributed. *Journal of derivatives*, 5:9–19, 1998.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

Rob J Hyndman, Yeasmin Khandakar, et al. *Automatic time series for forecasting: the forecast package for R*. Number 6/07. Monash University, Department of Econometrics and Business Statistics, 2007.

Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997.

Abbas Khosravi, Saeid Nahavandi, Doug Creighton, and Amir F Atiya. Comprehensive review of neural network-based prediction intervals and new advances. *IEEE Transactions on neural networks*, 22(9):1341–1356, 2011.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Roger Koenker and Kevin F Hallock. Quantile regression. *Journal of economic perspectives*, 15(4):143–156, 2001.

Keith Kuester, Stefan Mittnik, and Marc S Paolella. Value-at-risk prediction: A comparison of alternative strategies. *Journal of Financial Econometrics*, 4(1):53–89, 2006.

Denis Kwiatkowski, Peter CB Phillips, Peter Schmidt, and Yongcheol Shin. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of econometrics*, 54(1-3):159–178, 1992.

Kevin J Lang, Alex H Waibel, and Geoffrey E Hinton. A time-delay neural network architecture for isolated word recognition. *Neural networks*, 3 (1):23–43, 1990.

Nikolay Laptev, Jiafan Yu, and Ram Rajagopal. Reconstruction and regression loss for time-series transfer learning. In *Proceedings of the Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) and the 4th Workshop on the Mining and LEarning from Time Series (MiLeTS), London, UK*, volume 20, 2018.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

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

Greta M Ljung and George EP Box. On a measure of lack of fit in time series models. *Biometrika*, 65(2):297–303, 1978.

Burton G Malkiel. The efficient market hypothesis and its critics. *Journal of economic perspectives*, 17(1):59–82, 2003.

Mary Malliaris and Linda Salchenberger. Using neural networks to forecast the s&p 100 implied volatility. *Neurocomputing*, 10(2):183–195, 1996.

Tom M Mitchell. *Machine Learning*. Mcgraw-Hill Science/Engineering/Math, 1997.

Peter Molnár. Properties of range-based volatility estimators. *International Review of Financial Analysis*, 23:20–29, 2012.

Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, 2015.

Julian D Olden, Michael K Joy, and Russell G Death. An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. *Ecological modelling*, 178(3-4):389–397, 2004.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Gerhard Paass. Assessing and improving neural network predictions by the bootstrap algorithm. In *Advances in Neural Information Processing Systems*, pages 196–203, 1993.

Michael Parkinson. The extreme value method for estimating the variance of the rate of return. *Journal of Business*, pages 61–65, 1980.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Gábor Petneházi. Quantile convolutional neural networks for value at risk forecasting. *arXiv preprint arXiv:1908.07978*, 2019a.

Gábor Petneházi. Recurrent neural networks for time series forecasting. *arXiv preprint arXiv:1901.00069*, 2019b.

Gábor Petneházi and József Gáll. Exploring the predictability of range-based volatility estimators using recurrent neural networks. *Intelligent Systems in Accounting, Finance and Management*, 26(3):109–116, 2019a.

Gábor Petneházi and József Gáll. Mortality rate forecasting: can recurrent neural networks beat the Lee-Carter model? *arXiv preprint arXiv:1909.05501*, 2019b.

Gábor Petneházi and József Gáll. Evaluating the Lee-Carter model on hungarian mortality data. *Acta Oeconomica*, accepted.

Ser-Huang Poon and Clive WJ Granger. Forecasting volatility in financial markets: A review. *Journal of economic literature*, 41(2):478–539, 2003.

Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL http://www.R-project.org/. [Downloaded 2019-04-01].

Ronald Richman and Mario V Wuthrich. Lee and carter go machine learning: Recurrent neural networks. *Available at SSRN 3441030*, 2019.

L Christopher G Rogers and Stephen E Satchell. Estimating variance from high, low and closing prices. *The Annals of Applied Probability*, pages 504–512, 1991.

Leonard CG Rogers, Stephen E Satchell, and Y Yoon. Estimating the volatility of stock prices: a comparison of methods that use high and low prices. *Applied Financial Economics*, 4(3):241–247, 1994.

Tae Hyup Roh. Forecasting the volatility of stock price index. *Expert Systems with Applications*, 33(4):916–922, 2007.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

Gábor Scheiring, Darja Irdam, and Lawrence King. The wounds of post-socialism: a systematic review of the social determinants of mortality in hungary. *Journal of Contemporary Central and eastern europe*, 26(1): 1–31, 2018.

Hyun Song Shin. *Risk and liquidity*. Oxford University Press, 2010.

Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

Jinghong Shu and Jin E Zhang. Testing range estimators of historical volatility. *Journal of Futures Markets*, 26(3):297–313, 2006.

Justin Sirignano and Rama Cont. Universal features of price formation in financial markets: perspectives from deep learning. *Quantitative Finance*, pages 1–11, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15 (1):1929–1958, 2014.

James W Taylor. A quantile regression neural network approach to estimating the conditional density of multiperiod returns. *Journal of Forecasting*, 19(4):299–311, 2000.

University of California, Berkeley (USA), and Max Planck Institute for Demographic Research (Germany). Human mortality database. URL https://www.mortality.org. [Downloaded 2019-04-01].

Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

Yi Wang, Dahua Gan, Mingyang Sun, Ning Zhang, Zongxiang Lu, and Chongqing Kang. Probabilistic individual load forecasting using pinball loss guided lstm. *Applied Energy*, 235:10–20, 2019.

Qingsong Wen, Liang Sun, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*, 2020.

Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*, 2017.

Paul John Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*, volume 1. John Wiley & Sons, 1994.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

Ruoxuan Xiong, Eric P Nichols, and Yuan Shen. Deep learning stock volatility with google domestic trends. *arXiv preprint arXiv:1512.04916*, 2015.

Qifa Xu, Xi Liu, Cuixia Jiang, and Keming Yu. Quantile autoregression neural network model with applications to evaluating value at risk. *Applied Soft Computing*, 49:1–12, 2016.

Xing Yan, Weizhong Zhang, Lin Ma, Wei Liu, and Qi Wu. Parsimonious quantile regression of financial asset tail dynamics via sequential learning. In *Advances in Neural Information Processing Systems*, pages 1575–1585, 2018.

Dennis Yang and Qiang Zhang. Drift-independent volatility estimation based on high, low, open, and close prices. *The Journal of Business*, 73(3):477–492, 2000.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.