Jonen, Christian; Meyhöfer, Tamino; Nikolić, Zoran

Neural networks meet least squares Monte Carlo at internal model data

European Actuarial Journal

Provided in Cooperation with:
Springer Nature

Suggested Citation: Jonen, Christian; Meyhöfer, Tamino; Nikolić, Zoran (2022) : Neural networks meet least squares Monte Carlo at internal model data, European Actuarial Journal, ISSN 2190-9741, Springer, Berlin, Heidelberg, Vol. 13, Iss. 1, pp. 399-425, https://doi.org/10.1007/s13385-022-00321-5

European Actuarial Journal (2023) 13:399–425
https://doi.org/10.1007/s13385-022-00321-5

CASE STUDY

Neural networks meet least squares Monte Carlo at internal model data

Christian Jonen[1] · **Tamino Meyhöfer**[1] · **Zoran Nikolić**[2]

Received: 2 September 2021 / Revised: 1 February 2022 / Accepted: 9 June 2022 /
Published online: 8 July 2022

Abstract

In August 2020 we published "Comprehensive Internal Model Data for Three Portfolios" as an outcome of our work for the committee "Actuarial Data Science" of the German Actuarial Association. The data sets include realistic cash-flow models outputs used for proxy modelling of life and health insurers. Using these data, we implement the hitherto most promising model in proxy modeling consisting of ensembles of feed-forward neural networks and compare the results with the *least squares Monte Carlo (LSMC)* polynomial regression. To date, the latter represents—to our best knowledge—the most accurate proxy function productively in use by insurance companies. An additional goal of this publication is a more precise description of "Comprehensive Internal Model Data for Three Portfolios" for other researchers, practitioners and regulators interested in developing *solvency capital requirement (SCR)* proxy models.

Keywords Least squares Monte Carlo · Neural networks · Solvency II · Proxy modeling · Insurance risk management · Cash flow projection models · Risk-neutral valuation · Machine learning

✉ Christian Jonen
christian.jonen@generali.com

Tamino Meyhöfer
tamino.meyhoefer@generali.com

Zoran Nikolić
znikolic@uni-koeln.de

[1] Generali Deutschland AG, Cologne, Germany

[2] Mathematical Institute, University of Cologne, Cologne, Germany

1 Introduction

For years the expectations from the introduction of machine learning methods in insurance companies have been running high in both, academia and industry. This holds even more for the technical areas like actuarial, asset-liability management and risk management.

The progress of machine learning methods in these areas has nevertheless been rather slow, although these methods are readily available and rather easy to implement. A wide variety of open source tools are well-documented and can be easily adapted to insurance applications.[1] A crucial obstacle seems to be the notorious lack of realistic data with which the interested researchers and practitioners may be able to experiment.

Faced with a lack of real-life data, academic research sometimes reverts back to generic data produced by an algorithm. However, generic data are likely to miss the intricacies of real data. Too often this difference between synthetic and realistic data hinders a successful transfer of methods developed among academic researchers into the industry. For complex topics such as the Monte Carlo valuation of insurance companies, realistic data qualitatively differ from synthetically generated data. For instance, realistic data may contain unusual insurance contracts, e.g. with large sums insured or unusual deferment periods. Or they may have a concentration of certain features which are not captured in the synthetic data.

It is not that insurance companies do not possess troves of data. On the contrary, it is sometimes said that actuaries were the pioneers of data science. For decades they have been dealing with huge amounts of data extracting insights from them, see e.g. [4].

The problem seems to be the lack of incentives for actuaries in the industry to anonymize real data in such a way that there are no references to specific policy holders and no information about the company can be extracted from the data. For sure, such changes can be performed for any data set, but they involve effort which is usually not compensated in any way.

When implementing machine learning methods we heavily benefit from open source tools. For instance, Python is a free and open-source programming language; powerful packages such as Tensorflow and Keras are available for everyone to use. We believe that the actuaries should give something back to the larger data science community by providing their problem descriptions, realistic data and best models, such that the community is able to work with and improve them. Our data publication [15] along with this paper is one concrete step into this direction.

1.1 Our data

We have undertaken the task of providing realistic data for the area of proxy modeling in risk management within the Solvency II context. To the best of our

[1] Among numerous internet sources offering tutorials for implementations of machine learning models, we refer the readers to Towards Data Science Inc. [24].

knowledge, so far no data based on a model actually implemented in the industry has been published. For our use case developed for the Actuarial Data Science Committee of the German Actuarial Association we have produced realistic data stemming from life and health *cash flow projection models (CFPMs)* and containing all the complexities of real-life productive models. Data are only scaled and fully anonymized. In the GitHub repository [15] we have uploaded data for two life and one health insurance portfolios, projected 60 years into the future in an actuarial CFPM.

For each of the insurance portfolios our data contain four comprehensive sets of samples with different accuracy. The samples represent the movements of portfolio's *Own Funds (OF)* conditional on the changes of risk factors the company is exposed to. The original purpose of the data concerns the Solvency II regulation, but the same data can also be utilized for various questions with respect to asset-liability management or—more generally—the economic valuation of insurance companies.

The amount of samples we provide as well as the accuracy of actuarial valuations for our samples is higher than what we have seen in most internal models in the industry. We have done this intentionally in order to allow meaningful statistical and convergence analyses.

1.2 Regression models

For years the authors have worked in the area of life and health insurance proxy function modeling. Since 2015 we have been developing and testing various machine learning methods for the risk capital approximation. For a comprehensive description of the framework of risk capital proxy modeling as well as various machine learning approaches that can be used for it, we refer the readers to [18–20].

The regression task refers to the value of a portfolio of insurance policies, which must be evaluated in a Monte Carlo manner using stochastic simulations. An appropriate way of measuring errors for our framework is given in Sect. 3.

Our analyses so far are in line with Krah et al. [19] and indicate that neural networks combined with the hyperparameter tuning and an ensemble approach yield very accurate approximation results. This demonstrates anew the versatility of neural networks and their virtually universal application possibilities.

In the original publication of our data we have presented a rudimentary Jupyter notebook [15] showcasing a simple least squares regression for a given set of polynomial terms as well as a neural network training with a pre-defined architecture of the neural network in terms of e. g. the number of layers, nodes and activation functions used. In this paper we explain these two methods in more detail. Besides that, we disclose our best regression results as an invitation for the actuarial data science community to try to improve them. Besides describing the approach that has resulted in the high approximation quality, we focus on an efficient implementation of neural networks. By doing this, we recognize the fact that the computational resources in companies are limited, even in the times when many companies move such calculations to the cloud.

1.3 Related literature

The *least squares Monte Carlo (LSMC)* framework, which relies on data as ours, is described and formally introduced in Krah et al. [18]. Our polynomial proxy function fitting follows the approach described in this article, that is the adaptive *Akaike information criterion (AIC)*-based algorithm. Our implementation of neural networks follows Krah et al. [19]. The analysis of an efficient approach using less data in the last part of this article is specifically produced for this article.

An overview of several proxy methods can be found in Kopczyk [16]. Neural networks and random forests outperform the linear methods for the data analyzed in this article. The result for neural networks is in line with our results.

The seminal paper for the introduction of the LSMC method is Longstaff and Schwartz [21], where it was proposed for pricing of American options. The transfer to the insurance industry happened some 10 years later; one of the first descriptions of the new application was in Koursaris [17].

Recently, in Castellani et al. [5] several methods were compared and again deep neural networks have demonstrated the best performance.

An application of neural networks for an valuation of variable annuities with guaranteed minimum income benefit with three risk factors is included in Cheridito et al. [6], the example follows the previous work with a classical LSMC solution in Bauer and Ha [2].

1.4 Structure of the article

The remainder of this article is structured as follows: in Sect. 2, we briefly motivate the data analysis problem associated to the *solvency capital requirement (SCR)* calculation, which is the starting point for our use case. The underlying data set of the use case is then described in detail in Sect. 3. This not only serves to better understand the results presented in the following, but should also facilitate application and interpretation by inclined users. Finally, the data set is analyzed using different models, which are briefly introduced in Sect. 4. The numerical results of this analysis and its interpretation are discussed in detail in Sect. 5, and in Sect. 6 we conclude our use case.

2 Solvency II framework

Let us practically motivate our use case by giving a brief introduction to the risk capital calculation according to Solvency II in Sect. 2.1 and describing the functionality of an internal model in Sect. 2.2; in particular, we focus our attention on regression-based Monte Carlo methods in Sect. 2.3.

2.1 Risk capital calculation

It is well known that Solvency II aims at implementing a set of robust solvency rules for insurance companies, which takes the most material risks into account in an adequate way. One of the key concepts is hereby the calculation of the SCR.

According to Article 101 (3) of Solvency II Directive 2009/138/EC (see European Parliament and European Council [8])

"the SCR shall be calibrated so as to ensure that all quantifiable risks to which an insurance or reinsurance undertaking is exposed are taken into account. It shall cover existing business, as well as the new business expected to be written over the following 12 months. With respect to existing business, it shall cover only unexpected losses. It shall correspond to the Value-at-Risk of the basic OF of an insurance or reinsurance undertaking subject to a confidence level of 99.5 % over a 1-year period."

In principle, the Solvency II framework requires the derivation of the full loss distribution of the available OF, in order to derive its correct Value-at-Risk. This particularly does not only involve a market consistent calculation of the economic balance sheet items at the valuation date (so-called *base case* in the following) but also its re-evaluation for each possible scenario at the risk horizon (1 year within Solvency II). Most insurance companies avoid this enormous effort by applying the standard formula approach to calculate the SCR. But the largest life insurers usually stick to the original Solvency II requirement and develop a full-scale internal model which allows them to calculate the economic balance sheet for thousands of 1-year scenarios. It is important to note that a company-specific internal model needs to be approved by the regulator. Due to its complexity the management needs to decide whether the effort for running an internal model is justified considering costs and benefits. A clear advantage hereby comes from the fact that an internal model is able to take specific risks of the company into account as it is built upon an adequate methodology and a well-chosen calibration process.

In the following we focus our attention on internal models.

2.2 Functionality of an internal model

To start with, let us consider the functionality of an internal model for calculating the SCR. As motivated above, the SCR derivation relies on the re-evaluation of the economic balance sheet in multiple risk scenarios as displayed in Fig. 1.

As introduced in Bauer and Ha [2] technically these risk scenarios can be thought of as the realization of a multi-dimensional state process under the physical measure (aka real-world measure). This state process represents the uncertain development of influencing factors (risk factors) of the company's assets and liabilities. These risks are typically split into classes such as market, credit or underwriting risks; operational risk might be a further risk class. The market consistent re-evaluation of balance sheet items is then formally associated with the derivation of the conditional expectation of future cash flows under the risk-neutral measure. The liability

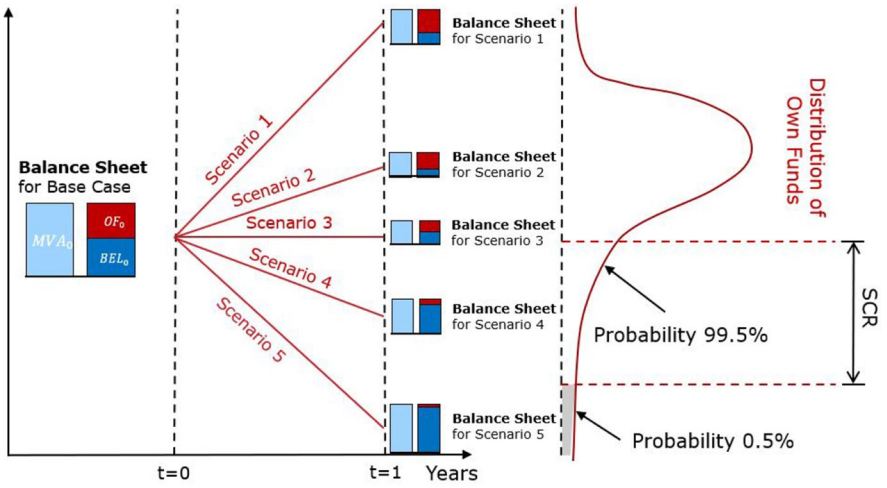

Fig. 1 Determination of OF distribution at risk horizon by simulating several scenarios

items of the economic balance sheet of life and health insurers are driven by options, asymmetries and complex dependencies which do not allow for a simple closed-form solution. Therefore, the valuation is typically carried out by involving Monte Carlo methods based on an actuarial CFPM, which determines the distribution of future cash flows along risk-neutral valuation paths between all stakeholders, like policy holders, company's employees and shareholders.

As illustrated in Fig. 2 a logical approach is to use a simulation within simulation technique; this nested simulation approach is also often applied in the field of financial mathematics, see e.g. Glasserman [11] or Andersen and Broadie [1]. This requires the generation of a large number of risk scenarios (also called *outer scenarios*) under the real-world measure; in doing so, the dependencies between the various risk factors have to be reflected in an appropriate way. In each of those outer scenarios the re-evaluation of the balance sheet is then carried out by another simulation based on risk-neutral valuation scenarios (also called *inner scenarios*) that are fed into the CFPM. Statistically, the more simulations are generated the more accurate the approximation of the OF distribution is. This approach is attractive due to its simplicity, but leads to exponentially growing computational efforts and is therefore expensive; moreover, high storage capacities are necessary. Thus, we move on with more efficient methods which are preferred in industry.

2.3 Regression-based approaches

The state-of-the-art answer to avoid the time-consuming nested simulations approach are regression-based methods; these methods need significantly lower storage and computational time. There has already been a few publications which have dealt with this problem, see for instance Bettels et al. [3], Hejazi and Jackson [14],

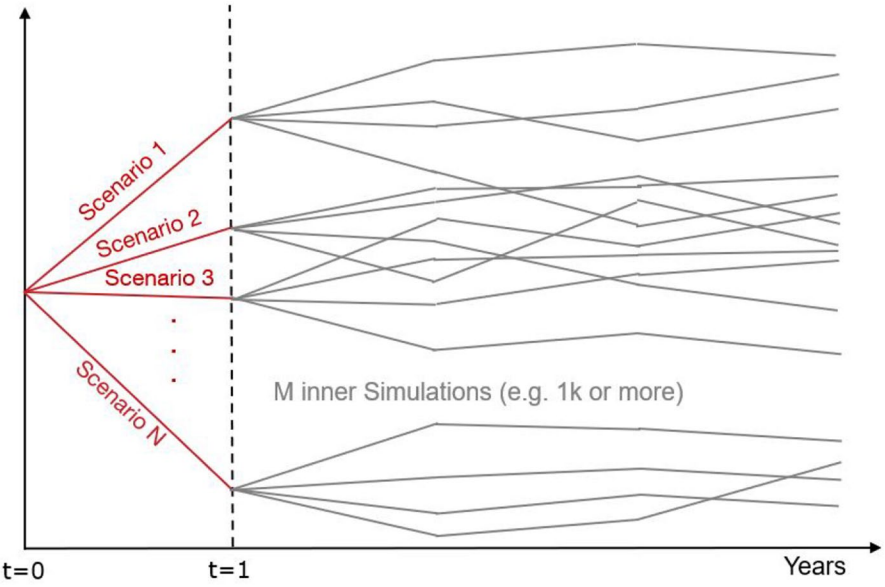

Fig. 2 Launching simulations within simulations to approximate balance sheet items for several scenarios at risk horizon

Nikolić et al. [22], Fernandez-Arjona [9], Krah et al. [19, 20].[2] The key idea is to define an appropriate fitting space, denoted by $I_1 \times \cdots \times I_D$ in the following, which is given by the intervals $I_d \ni RF_d$ for the corresponding D risk factors $RF_1, \ldots, RF_D$. Then, only a low number of inner scenarios are generated for each outer scenario from the cube $I_1 \times \cdots \times I_D$. The inaccurate mean of the low number of inner scenarios leads to an inaccurate but unbiased fitting point for each outer scenario, which is interpreted as a rough estimation of a balance sheet position. In the following we focus on the balance sheet item OF.[3]

Based on this approach we receive a point cloud of OF estimations, such that the ultimate task is to find a sound functional relationship between the risk factors and OF. Let us call f this functional relationship which represents the approximation $\widehat{OF}$ of the true OF. Then, for $n = 1, \ldots, N$, denoting the estimated OF samples and the simulated risk factor samples (outer scenarios) by OF_n and $RF_{1n}, \ldots, RF_{Dn}$, respectively, we are able to determine the function f via least squares, i.e. minimizing

$$\sum_{n=1}^{N} (OF_n - f(RF_{1n}, \ldots, RF_{Dn}))^2. \tag{1}$$

An exemplary visualisation of an OF surface for two risk factors is shown in Fig. 3.

[2] In the context of pricing American-style options a well-working regression-based Monte Carlo method has been proposed by Longstaff and Schwartz [21].

[3] In this use case we directly focus on approximating the OF, but the OF might also be implicitly calculated by approximating the assets and the *Best Estimate of Liabilities (BEL).*

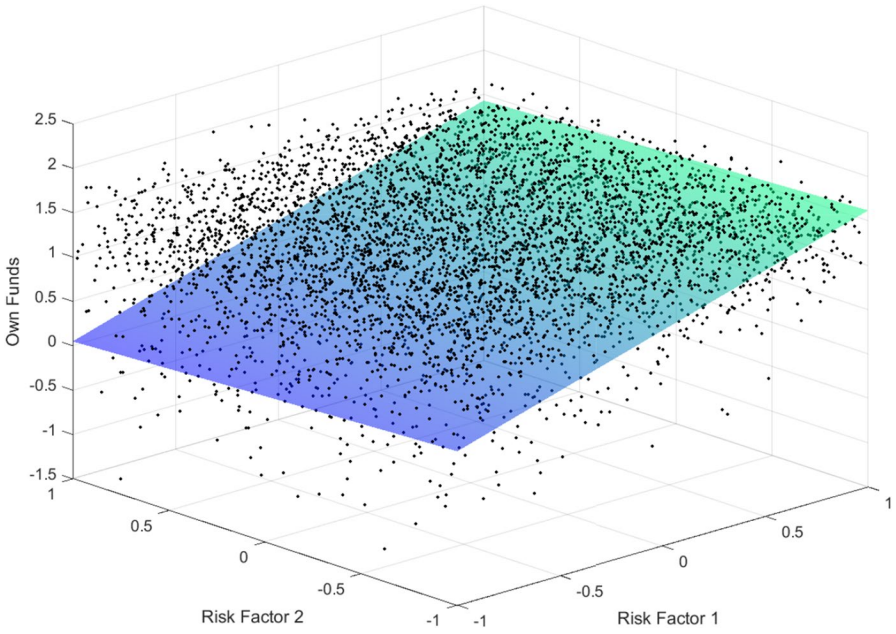

Fig. 3 OF polynomial function plotted for two risk factors, derived by ordinary least squares and the basis functions $\{1, \{RF_d\}_{d=1}^{D}, \{RF_d^2\}_{d=1}^{D}\}$; black points denote the first 5k fitting scenarios for Portfolio 1 of the data set [15], explained in Sect. 3

The focus of this use case is on the application of neural networks for regression. By doing so, we want to challenge the ordinary least squares approach—working with polynomials—and, based on our comprehensive data set, an adequate comparative analysis is guaranteed. In the sense of open data science our data set can be used by the interested readers for performing own meaningful studies.

3 Data set

As indicated above, our central contribution consists in providing comprehensive samples of risk model data for three insurance portfolios. In the following we exactly describe what data sets are included in our publication [15].

The data sets are based on the logic of the regression approach outlined above. They provide combinations of risk scenarios and the matching OF values for three different insurance portfolios, two life and one health insurance portfolio. These portfolios are purely fictitious, albeit realistically compiled. The OF values have been derived as Monte Carlo estimates based on risk neutral projections of CFPMs. These take into account the development of assets and liabilities, their mutual interaction via defined management rules and regulatory requirements of the German insurance market along the capital market paths of a sophisticated economic scenario generator.

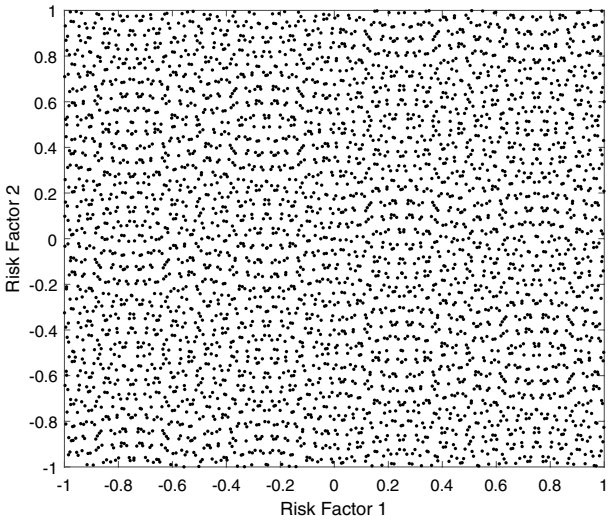

Fig. 4 Evenly filled fitting space $I_1 \times \cdots \times I_D$; here as an example with $D = 2$ for the first 4k realisations of the risk factors 1 and 2 from the training data set of Portfolio 1, see Sect. 3 and Jonen et al. [15]

Each of the Monte Carlo values corresponds to a specific risk scenario which is represented by the realizations of relevant risk factors $RF_1, \ldots, RF_D, D = 13$ for Portfolios 1 and 2, $D = 12$ for Portfolio 3 (e.g. interest rate level, equity, underwriting stresses, and so on). The choices of the number N and distribution of (outer) risk scenarios and the number M of (inner) risk neutral projections based on which the Monte Carlo estimates are derived, vary with the different types of data sets explained in the following.

For the training of the proxy function f, the *fitting data set* (also named *training data set*) provides rather imprecise Monte Carlo evaluations of the balance sheet items, based on the average of only two inner (risk-neutral) projections of a CFPM, but from a large number ($N = 2^{15}$) of risk scenarios. An evenly filled space, as illustrated in Fig. 4, is considered to be advantageous for the task of proxy function training. This regular distribution of scenarios is ensured by deriving them from a Sobol sequence on the hypercube $I_1 \times \cdots \times I_D$. The intervals $I_j, \ldots, I_D$ are chosen such that they cover a large part, for instance more than 99.95%, of each risk factor distribution. In particular, the scenario selection of the training set does not contain any further information about the distribution characteristics. This choice is motivated by the need of a uniform quality of the predictions over a wide range of possible risk factor representations in order to allow for an as much as possible unbiased evaluation of all risk scenarios.

The *out-of-sample (OOS) validation set* consisting of 256 scenarios can be used to check the quality of the selected proxy function f. The values shown are based on a risk-neutral evaluation of the model described above over a thousand simulations. Thus, these represent significantly more precise estimates of the balance sheet item, matching the respective external scenario, than in the case of the fitting scenarios. A high quality proxy function should be able to predict the precisely simulated values.

Statistics such as the mean of the relative (absolute) deviations between the predicted values (calculated by the proxy function) and the values determined according to the Monte Carlo estimations are suitable for checking the quality; this quality check is the so called *OOS validation*. It should be noted that the observed discrepancy is on the one hand due to the inaccuracy of the selected proxy function, and on the other to the inaccuracy of the Monte Carlo simulation. In order to better classify the latter, the data set for the OOS validation additionally contains the standard error for each scenario as a statistical measure of the inaccuracy of the underlying estimator for the mean value, e.g. by allowing the derivation of suitable confidence intervals.

While the OOS validation points here are evenly distributed in the risk factor space, our third data set called *SCR region data set* represents an alternative OOS validation package that specifically focuses on a certain subset of the risk factor space. The repository again contains output variables, risk factors and standard errors per scenario. The SCR region represents the subset of the risk scenarios whose losses are close to the 99.5th percentile of the loss distribution relevant for the SCR. For Portfolios 1 and 2 we provide 129 SCR region scenarios, for Portfolio 3 there are 50. However, for Portfolio 3 we have identified one outlier which is excluded from further analyses in this use case.[4]

Hence, the selection of scenarios in the SCR region depends on the results of a previously performed regression and proxy function analysis, as this is the only way to determine the overall distribution of own funds and the relevant loss region. In this respect, the SCR region set primarily represents a validation for the initially selected proxy function approach. For the SCR region derivation we have used a standard LSMC polynomial proxy function. Since the quality of this proxy is good, the SCR region should be largely stable with regards to other proxy functions of sufficiently high quality, see Krah et al. [18], Figs. 2 and 3. Due to the importance of the SCR region, the requirements for the accuracy of any proxy functions in the SCR region are particularly high. In order to reduce a distortion due to Monte Carlo errors as much as possible, the OF for scenarios of the SCR region have been evaluated with four thousand simulations each. They have therefore a significantly lower standard error than in the usual OOS validation points.

To simplify the interpretation of the results, the fitting space as well as the output variable have been standardized in the provided data set in such a way that the OF of the baseline scenario are for all portfolios equal to 1. The term baseline scenario refers to the unstressed scenario in which all risk factors take the value 0. The reference value, that has been used for the scaling, has been derived on 16 thousand simulations. Corresponding to its high relevance—the SCR corresponds to the 1 in 200 years loss which is measured as the difference of the corresponding adverse scenario (SCR region) with respect to the baseline—the statistical precision again has been increased with respect to all other evaluations mentioned above.

Summarizing, we have produced four sets of data by running the following numbers of simulations:

[4] Scenario no. 14 in the SCR region data set.

- $32{,}768 \cdot 2 = 65{,}536$ simulations for the fitting set,
- $256 \cdot 1{,}000 = 256{,}000$ simulations for the OOS validation data,
- $129 \cdot 4{,}000 = 516{,}000$ for Portfolios 1 and 2 and $50 \cdot 4{,}000 = 200{,}000$ for Portfolio 3 simulations for the SCR region data set,
- 16, 000 simulations for the base.

As it can be seen, the resulting number of 853,536 simulations (537,536 for Portfolio 3) is enormous and beyond calculation capabilities of most insurance companies. Indeed, companies would arguably consider running a light version of nested stochastics if they were able to cope with this amount of simulations. This makes our data sets convenient objects for testing and convergence analyses.

4 Models

In our use case [15] we focus the attention on two methods for deriving adequate functions and apply these methods on the underlying data set.

A milestone in statistics was reached by Gauss at the end of the eighteenth century when he invented the least squares method. In research and practice several applications are based on this regression type, both, for linear regression but also for more complex tasks.

The ordinary least squares method for proxy derivation has been implemented for regulatory reporting purposes in a number of life and health insurers in Europe. In our opinion it represents the most mathematically sound regression approach of all the methods found today in the industry.

Nowadays, artificial intelligence is on everybody's lips and researchers, practitioners but also politicians are talking about a field with significant potential. Machine learning can help to better understand and solve complex problems. Thus, in our study we concentrate on investigating neural networks as regression functions or rather an ensemble of neural networks. The LSMC framework from Krah et al. [18] is mostly preserved in this approach and the main difference between both approaches is the substitution of polynomial functions by neural networks, as outlined in the following.

4.1 Ordinary least squares

Coming back to the regression problem (1) a simple, yet powerful model function f for approximating the OF is given by a linear combination of basis functions $\{\phi_m(\cdot)\}_{m=1}^{M}$ with coefficients $a_m \in \mathbb{R}$:

$$f(RF_1, \ldots, RF_D) = \sum_{m=1}^{M} a_m \phi_m(RF_1, \ldots, RF_D).$$

For what concerns the basis functions, we are free in the selection but polynomials or radial functions have shown well-working approximation quality; using e.g. monomials as basis functions allows for a direct interpretation of the functional shape with respect to several risk factors, whereas the constant term is the base case value and the residual term shows the stress behavior for any evaluated scenario. Then, in order to determine the function f the corresponding optimization problem can be solved by numerically stable methods such as the QR or singular value decomposition, see Golub and van Loan [12]. The advantage of this approach is clearly its simplicity, good results interpretation and an easy implementation as a number of standard software packages provide solvers for deriving the coefficients of the assumed proxy model. This approach goes under the name LSMC, called *LSMC-Pol* in the following, where *Pol* stands for polynomial functions.

4.2 Neural networks

Lately the neural networks have become ubiquitous in the scientific and practical publications. The attempts to describe neural networks in mathematical terms have proved to be somehow tedious. This comes primarily from the notational complexity, not from a conceptual difficulty in explaining neural networks as mathematical functions.

Often the natural model, that is the brain, is used as a motivation for an introduction of neural networks. Although there are some striking analogies between "natural" learning and its "machine" counterpart,[5] this perspective is not really useful for a mathematical understanding of the underlying functions and algorithms.

We do not intend to fully introduce the concept of neural networks. Instead, we clarify the terminology, which we employ, and focus our attention on the features, which are relevant for our application of neural networks.

Let $N_0, \dots, N_{L+1} \in \mathbb{N}$. Following Definition 1 and Definition 2 in Krah et al. [19], a *fully connected feed forward neural network* with $L \in \mathbb{N}$ *hidden layers* is a function $f : \mathbb{R}^{N_0} \to \mathbb{R}^{N_{L+1}}$ defined as

$$f = (\Phi_{L+1} \circ a_{L+1}) \circ (\Phi_L \circ a_L) \circ \cdots \circ (\Phi_1 \circ a_1),$$

where $\circ$ denotes the concatenation, and

$$a_l : \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}, l \in \{1, \dots, L+1\},$$

are affine mappings represented by matrices W_l of dimension $N_{l-1} \times N_l$ and vectors $b_l \in \mathbb{R}^{N_l}$. N_0 is the *input dimension*, N_{L+1} the *output dimension* and N_l the *number of neurons* or *nodes* in the hidden layer l. For each $l \in \{1, \dots, L+1\}$, the functions Φ_l are defined as

$$\Phi_l : \mathbb{R}^{N_l} \to \mathbb{R}^{N_l},$$
$$\Phi_l(z_1, \dots, z_{N_l}) = (\phi_l(z_1), \dots, \phi_l(z_{N_l})),$$

[5] E.g.: The wisdom of crowds, "slow" and "quick" learners, learning by heart etc.

with real-valued functions $\phi_l : \mathbb{R} \to \mathbb{R}$, which are called *activation functions* of the neural network and for which it is required to be monotone and Lipschitz continuous (see Definition 1 in Krah et al. [19][6]). In our use case all $\phi_l, l \in \{1, \dots, L\}$, are equal and denoted by ϕ. Only the function ϕ_{L+1}, which is called *output activation function*, may differ from ϕ. Moreover, in our application we also set $N_1 = \cdots = N_L$, that is the number of neurons in each hidden layer is the same.

In the numerical results in Sect. 5 we use the following activation functions:

- *Sigmoid*: $\phi(z) = 1/(1 + exp(-z))$,
- *Rectified linear unit (ReLU)*: $\phi(z) = \max(0, z)$,
- *Leaky ReLU*: $\phi(z) = \max(\lambda \cdot z, z)$ with $0 < \lambda < 1$,
- *Linear*: $\phi(z) = z$.

The functions a_l with $l \in \{1, \dots, L+1\}$ are affine transformations between the layers of the neural network and the parameters of the affine transformations are *weights* between the nodes of the layers. The change of these weights by an algorithm constitutes the actual *training* for given input and output data.

For instance, for Portfolio 1 with 13 risk factors $RF_1, \dots, RF_{13}$ (denoted by x), the input dimension is $N_0 = 13$ and the first affine transformation is

$$\begin{aligned} a_1 &: \mathbb{R}^{13} \to \mathbb{R}^{N_1}, \\ a_1(x) &= W_1 x + b_1, \end{aligned}$$

with a $13 \times N_1$ matrix W_1 and a vector $b_1 \in \mathbb{R}^{N_1}$. Hence, a_1 maps the 13-dimensional space of risk factors to the first layer consisting of N_1 neurons.

As already noted above, in our applications $N_1 = \cdots = N_L$ and $N_{L+1} = 1$ as we work with the same number of nodes for all hidden layers and our data have only one output variable.

The last affine transformation takes the form $a_{L+1} : \mathbb{R}^{N_L} \to \mathbb{R}$, so it maps the outputs of the nodes in the last hidden layer to a real number. After a transformation with the output activation function ϕ_{L+1} we obtain an estimate for the Own Funds $\widehat{OF}(x) = f(x) \in \mathbb{R}$, where the input x represents the input vector of the risk factors $RF_1, \dots, RF_D$.

Notify that the vector x is 13-dimensional for Portfolios 1 and 2, and 12-dimensional for Portfolio 3. For a more thorough description of the risk factors which may represent the input data, we refer to Table 1 in Krah et al. [18]. We do not disclose the exact meaning of the input data, i.e. what each risk factor represents, as we do not want to reveal details about exact implementations in the industry. Our application with neural networks is illustrated in Fig. 5.

For a more comprehensive introduction to neural networks, also including many other possible architectures, we refer the readers to Goodfellow et al. [13] or Efron and Hastie [7]. In this paper we only refer to fully connected feed forward neural networks.

[6] Recently, the notion of activation functions has been extended to non-monotone Lipschitz continuous functions, e.g. by Google, see Prajit Ramachandran [23]. However, this has no relevance for our use case as we only use monotone activation functions.

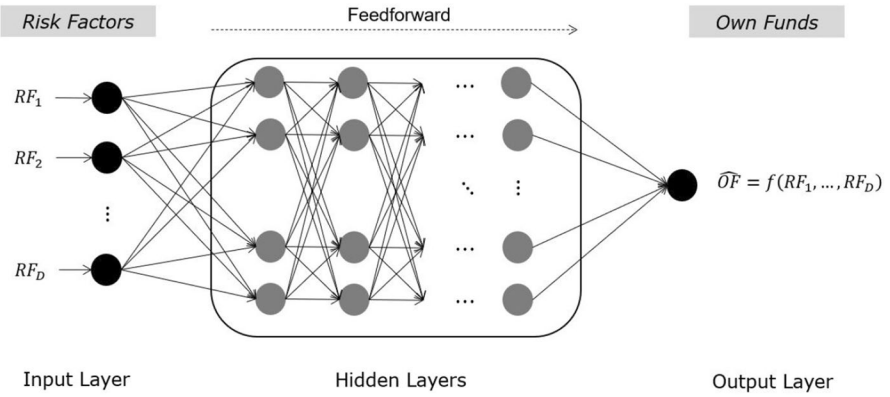

Fig. 5 Neural network for the underlying use case, where the input layer is given by D risk factors $RF_1, \ldots, RF_D$ and the output layer presents the approximation $\widehat{OF}$ of true OF

4.2.1 Hyperparameters

Our application of neural networks is based on varying the hyperparameters of neural networks and utilizing the ensemble method.

What are hyperparameters? A neural network is parameterized by the weights of the connections between its nodes. The word *parameters* when speaking about neural networks refers to these weights. The *hyperparameters* define the architecture and settings which are not changed during the training of a neural network with the backpropagation method. Hence, the task of the chosen optimization algorithm is to determine the weights for a given set of hyperparameters. Note that the underlying problem is nonlinear and thus iterative solvers have to be applied to find the optimal weights.

In the next section we list the hyperparameters which we have varied in our experiments.

5 Numerical results

We now move on to the quantitative part of our use case in which we want to challenge the current industry standard least-squares approach based on polynomials (LSMC-Pol) by neural networks (called *LSMC-NN*) for approximating the OF at the risk horizon. As we are interested in receiving high accuracy on the one hand and having in mind the practical limitations given by complexity and computational time on the other hand, we implement a step-wise approach. Hereby, for LSMC-NN we go from a very complex modeling approach to a smart way of implementation in order to tackle the trade-off between accuracy and complexity as well as provide the reader a guideline for a practical application of neural networks.

5.1 Hyperparameter tuning (first attempt)

In the course of the preparation and publication of our data we have developed various machine learning models. The most promising models have been developed by neural networks. After testing various options we have defined a rich set of 300 quasi-randomly selected combinations of hyperparameters[7]:

(i) Architecture of the neural network:

- Number of hidden layers: 2–10,
- Number of nodes in each layer: 16–128 (constant across all layers),
- Activation functions (hidden layers): Sigmoid, ReLU, Leaky ReLU with $\lambda \in (0, 0.1)$,
- Output activation functions: Linear, Sigmoid.

(ii) Optimization algorithm:

- Solver: *Adam, Adamax, Nadam*,
- *Batch size*: discrete values $\{100, 200, 400, 800, 1{,}600\}$,
- *Drop out rate* in the range of (0, 0.4),
- *Learning rate* in the range of (0.0005, 0.005),
- *Initializer*: *Random Normal, Random Uniform, Glorot Uniform.*

In total we have performed more than 800 different pre-training tests with neural networks. Even after keeping all but one hyperparameters fixed, the variation of the results when varying the remaining hyperparameter was considerably high. Most results did not show a completely clear dependency of the approximation quality (measured in sum of errors and sum of squared errors for OOS validation and SCR region data) from the changes in a hyperparameter. For instance, we did not observe that increasing the drop out rate always leads to a better result or vice versa. There were some tendencies, which we explain below when we define the efficient set of hyperparameters.

For each of the hyperparameters listed above we have derived the sets from which to choose the hyperparameter values (e.g. from 2 to 10 for the number of hidden layers) by performing these pre-training tests in which we have allowed much larger ranges for all hyperparameters. To give an example, in this pre-training stage we have trained neural networks with the number of hidden layers from 1 until 11. After completing the training we have plotted the out-of-sample results depending only on one hyperparameter, i.e. here depending only on the number of hidden layers. The results have shown that 1 hidden layer leads to notably lower accuracy and the same holds for 11 hidden layers. As a consequence we have selected [2, 10] as the range for the number of hidden layers. Furthermore, in each training we have allowed a new random seed. Since the training of neural networks converges towards a local minimum we should always keep in mind that another set of initial weights for the

[7] For a more detailed description of these hyperparameters, we refer the readers to Towards Data Science Inc. [24] and Geron [10].

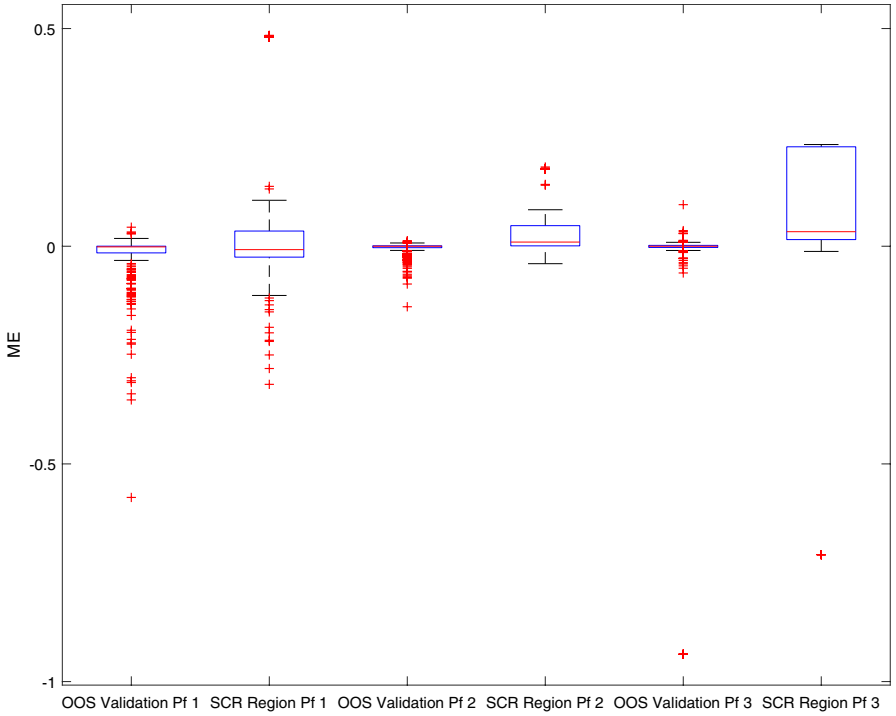

Fig. 6 Box plots for the mean error resulting from 300 calculated nets for all validation points and Portfolios 1, 2 and 3 (denoted by *Pf1*, *Pf2* and *Pf3*, respectively)

optimization method might lead to another model. We take this into account by working with randomly selected values for the initial weights.

5.2 The benchmark approach—ordinary least squares

Working with LSMC-Pol the toolbox is much simpler. Instead of varying several hyperparameters we employ the forward adaptive step-wise algorithm which gradually builds up a polynomial adding terms of monomial basis functions. For the model selection as well as the stopping criterion we use the AIC where we restrict the number of admissible terms to the number of 150 and the maximum polynomial degree to eight for both, the single and cross-terms, albeit the highest degree chosen by the algorithm was six. For a detailed explanation of the procedure, we refer the readers to Krah et al. [20]. We refer to the results derived by this benchmark technique as *LSMC-Pol2* in the following. For the sake of comparability, we additionally report results from an even simpler model, namely a simple model consisting only of linear terms RF_d, denoted by *LSMC-Pol1*.

With reference to the approach outlined above it is worth noting that one might try deriving LSMC proxy functions with higher quality, but we think that the goodness of the selected polynomials is satisfactory. In particular, we do not want to

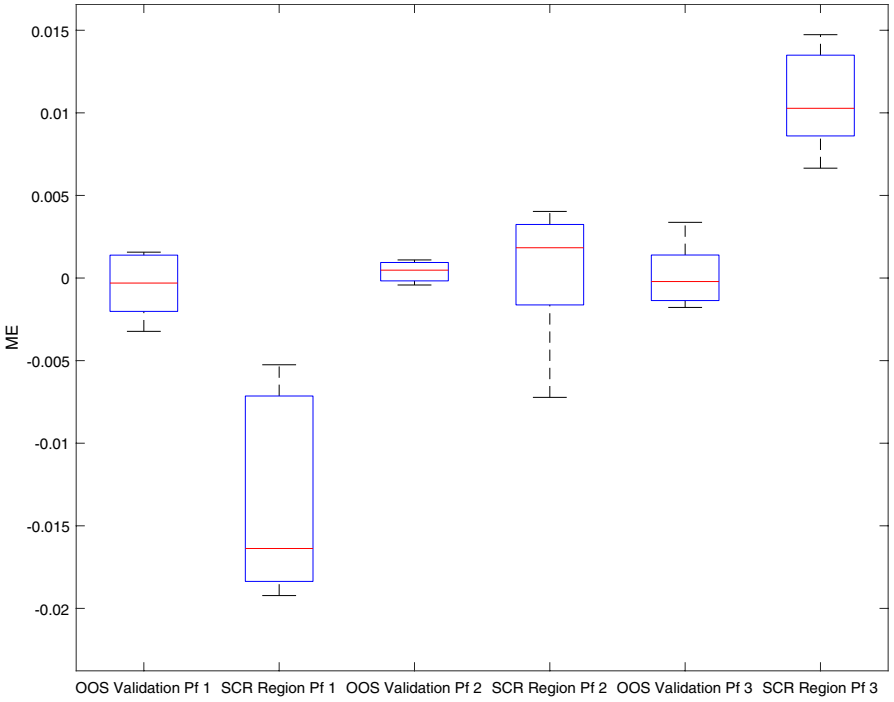

Fig. 7 Box plots for the mean error resulting from the "Top 10" of 300 nets for all validation points and portfolios

explore the limits of the LSMC approach. Rather, we investigate whether an alternative technique can reliably beat the goodness of an easy-to-implement existing approach, such as the LSMC-Pol approach.

5.3 Comparison LSMC-Pol vs. LSMC-NN

Let us focus on the results of our first trial, in which we run a number of neural networks based on the previously mentioned comprehensive set of hyperparameters.

For all three portfolios, Fig. 6 shows box plots of the *mean errors (ME)* calculated for the OOS validation and SCR region points.[8] We clearly see a huge variety of accuracy: driven by the choice of hyperparameters we yield results with high approximation quality, but also models with a poor goodness of fit; some outliers can be observed for each validation set. This observation is in line with our expectation as the hyperparameter choice might be more art than science.

Taking the mean of the models with the lowest *mean squared error (MSE)* for the OOS validaton set might be applied to get a higher approximation quality and to solve the curse of hyperparameters choice. By doing so, Fig. 7 shows the box plots

[8] Note that for Portfolio 3 we have excluded the outlier scenario no. 14 in the SCR region points.

Table 1 ME and MAE for Portfolios 1, 2 and 3 calculated by LSMC-Pol and LSMC-NN

			Approach		
			LSMC-Pol1	LSMC-Pol2	NN Ensemble
Portfolio 1	ME	OOS validation	− 0.0059	− 0.0010	− 0.0004
		SCR region	− 0.0717	− 0.0427	− 0.0139
	MAE	OOS validation	0.1064	0.0182	0.0112
		SCR region	0.0748	0.0431	0.0176
Portfolio 2	ME	OOS validation	0.0046	0.0054	0.0005
		SCR region	− 0.0077	0.0077	0.0004
	MAE	OOS validation	0.0241	0.0113	0.0072
		SCR region	0.0186	0.0124	0.0053
Portfolio 3	ME	OOS validation	0.0021	0.0004	0.0001
		SCR region	0.0233	0.0117	0.0105
	MAE	OOS validation	0.0382	0.0208	0.0181
		SCR region	0.0258	0.0217	0.0187

for the ten best neural networks, measured regarding the MSE of the OOS validation set, out of the 300 calculated models. We observe tighter box plots without outliers. While the tighter box plots on the OOS validation sets are a logical result of taking out nets with low accuracy (measured as MSE on the very same set), the results also show that by doing this we are able to reduce the bias of the SCR estimate and in particular reduce the prediction error on unseen data.

Following, for our final comparison with the LSMC-Pol approach we take the average of these best 10 nets, which is based on the idea of the ensemble technique for weak learners. As can be seen in Table 1 we get a high approximation quality for our NN ensemble; this is true for all Portfolios. In particular, we see that LSMC-NN is superior to LSMC-Pol, both in terms of ME and *mean absolute errors (MAE)*. While the MAE gives evidence for a lowered variability of errors, also the ME can be of particular interest, especially for the SCR region set. After all, in the sense of the original task—the SCR calculation—we are able to receive a lower bias using LSMC-NN. Fairly enough, we should also take the computational effort into account as the calculation of the comprehensive set of 300 different combinations of hyperparameters is much more expensive than deriving a well-working polynomial with a good approximation quality. Therefore, we go on by considering some further techniques to make the LSMC-NN approach more efficient, which is, in particular, important for applications in practice.

5.4 Optimizations of the procedure and feasibility concerns

By acknowledging that the amount of data we provide, as outlined in Sect. 3, exceeds the computational resources of most insurance companies reporting under

Table 2 Overview of the settings tested with the aim of reducing the computational burden in the practical applications

Step	HypCho	NoHypComb	Early	EnsCho	NoFit
I	Wide	300	Vali	Vali	32,768
II	Efficient	300	Vali	Vali	32,768
III	Efficient	90 (a), 30 (b)	Vali	Vali	32,768
IV	Efficient	90 (a), 30 (b)	Fitting	Vali	32,768
V	Efficient	90 (a), 30 (b)	Fitting	Fitting	32,768
VI	Efficient	90 (a), 30 (b)	Fitting	Fitting	16,384
VII	Efficient	90 (a), 30 (b)	Fitting	Fitting	8192

Solvency II we intend to show in this chapter how good proxy functions can be developed using just a fraction of the available data.

In order to facilitate an approach which may actually be feasible for most insurance companies, we have trained neural networks in the following seven settings. The ultimate target is that we arrive at Step VI and Step VII to settings which are actually computationally feasible for most companies. These seven steps can be found in Table 2.

The columns in Table 2 denote different settings we have varied in our numerical experiments: *HypCho* is the possible set for hyperparameters. With *Wide* we allow a broad range of hyperparameters, e.g. number of layers from 2 to 10 as outlined above. The *Efficient* choice means that prior to the final run we have analyzed which configurations lead on average to lower errors and allow only them in the hyperparameter search space; here we go into detail below. *NoHypComb* is the number of calculated hyperparameter combinations. *Early* denotes the data set that is used for the early stopping within the neural network training. If *Vali* is chosen, the MSE on the OOS validation data is monitored during training, and, if it does not decrease in the last 200 epochs, the training of the neural network stops; we have chosen 200 epochs as the maximum number, whereas one is free to adapt this threshold. *Fitting* means that the fitting data set is used to monitor the MSE and stop the training. *EnsCho* shows which data set is used when deciding which 10 neural networks are the best and should be used in the ensemble. Finally, *NoFit* refers to the number of fitting points used for the neural network training.

Summing up all seven steps, we can represent the settings in the following list, in which we describe the full setting for Step I and mention for each subsequent step the feature that has changed:

I Use 300 widely chosen hyperparameter combinations, ensemble of 10 best neural networks, validation data for early stopping, validation data for ensemble building,
II Choose efficiently hyperparameter combinations,
III Choose only 90 (a) and 30 (b) efficient hyperparameter combinations,
IV Use fitting data for early stopping,
V Use fitting data for ensemble building,
VI Use only 16,384 fitting points,

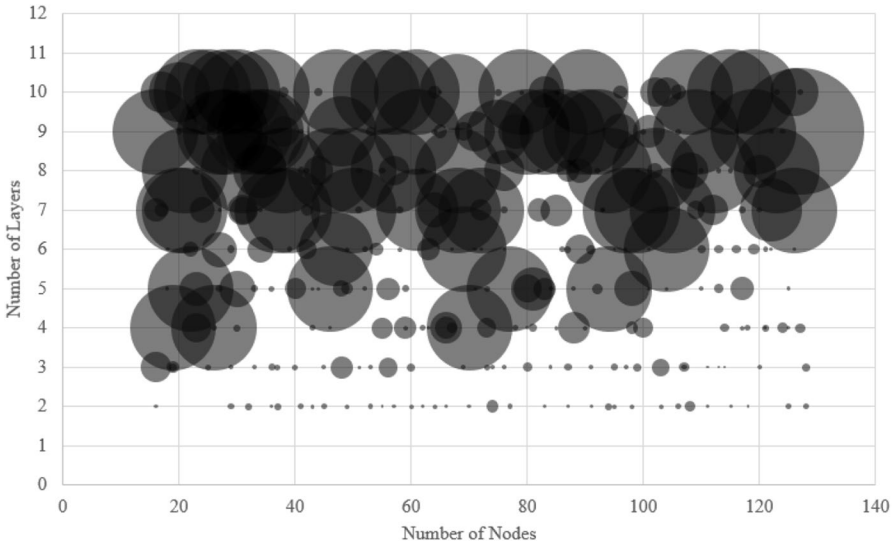

Fig. 8 MSE for different numbers of layers and nodes (the size of a bubble corresponds to the error)

VII Use only 8192.

In the following we focus our attention on measuring the quality of results by reducing complexity as previously outlined.

5.5 Comparison of neural networks

In order to investigate the potential loss of accuracy by moving from Step I to Step VII in the following we concentrate on Portfolio 1 and both error metrics, ME and MAE, for the OOS validation and SCR region data sets. In order to ensure a fair comparison between LSMC-Pol and LSMC-NN for Step VII, we have also derived LSMC-Pol1 and LSMC-Pol2 based on 8192 fitting points, denoted by *LSMC-Pol1 8k* and *LSMC-Pol2 8k*.

As mentioned above, to execute Step II we have to decide which values for hyperparameters can be seen as efficient. For that we have analyzed the MSE with respect to changes in several hyperparameters with the goal of identifying whether some hyperparameter values are favorable in terms of good approximations. In Fig. 8 we clearly see there is a significant impact of the number of layers on the magnitude of errors. Neural networks with a low number of layers tend to produce smaller errors and with an increasing number of layers the errors systematically become larger; a small error cannot be guaranteed for four or more layers. The reason for this may lie in the vanishing gradient effect. To underline this, let us consider the mean of the MSE for an increasing number of hidden layers and the chosen activation functions

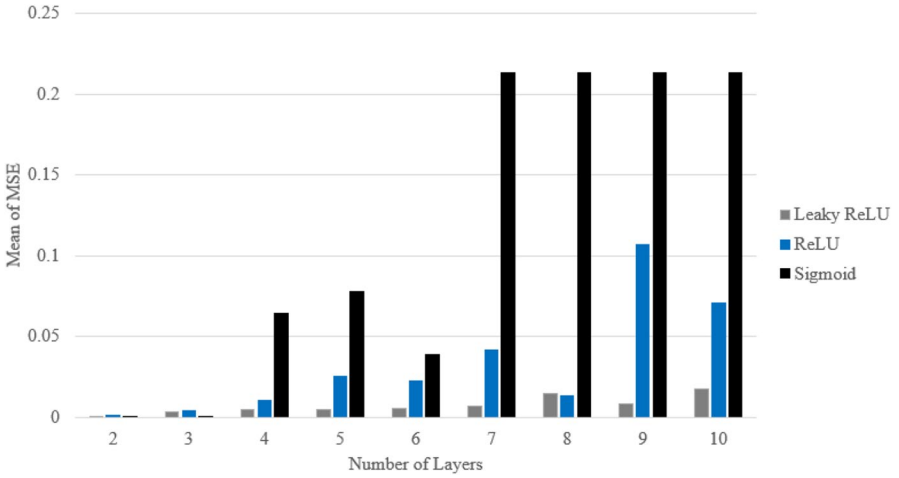

Fig. 9 Mean of MSE for different numbers of layers separately for each activation function

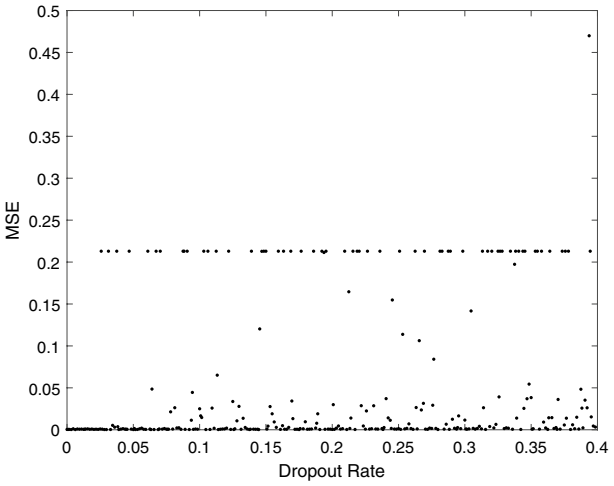

Fig. 10 MSE for several dropout rates

(in the inner nodes) in Fig. 9. We observe that the Sigmoid function systematically leads to higher errors for more layers.[9]

The activation functions ReLU and Leaky ReLU also produce higher errors with an increasing number of layers, albeit their errors are much lower. By performing this analysis for all hyperparameters it has turned out that some hyperparameter values have systematically led to strong approximation results (in the sense of small MSE values measured for the OOS validation set). On the other hand, different

[9] This is a known effect for Sigmoid function, see https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.

values of some hyperparameters such as the optimization method, learning rate or dropout rate have not given any insight regarding the approximation quality, see e.g. Fig. 10.

Following, to determine an efficient set of hyperparameters we have focused on the most promising values. For Portfolio 1, this procedure has led to the following search space:

(i) Architecture of the neural network:

- Number of hidden layers: 2–3,
- Number of nodes in each layer: 16–128 (constant across all layers),
- Activation functions (inner nodes): Leaky ReLU with $\lambda \in (0, 0.1)$,
- Activation functions (output): Sigmoid.

(ii) Optimization algorithm:

- Solver: Adam,
- Batch size: discrete values $\{100, 400\}$,
- Dropout rate in the range of (0, 0.025),
- Learning rate in the range of (0.0005, 0.001),
- Initializer: Random Uniform.

It seems to be natural to automatize such an efficient hyperparameter search. Figures 11 and 12 plot the ME and MAE, respectively, for polynomials—with good statistical properties—and neural networks resulting from all steps.

At first glance it can be highlighted that the neural networks have outperformed the LSMC-Pol approach for all steps regarding the MAE for the OOS validation data set, apart from Step VII, where the MAE is lower for the polynomial Pol2; furthermore, the polynomial Pol2 shows good results for OOS validation data if the ME is the error metric. Considering the approximation quality for the SCR region data set we have observed a significant improvement by applying neural networks; this is true for both error metrics, MAE and ME. Using an efficient hyperparameter search approach could further reduce the ME from -1.39×10^{-2} to -0.83×10^{-2} for the SCR region points, whereas the quality for the OOS validation data set has slightly declined. Even if we reduce the number of hyperparameter sets from 90 resp. to 30 we have still achieved strong results. In many machine learning applications the initial data set is split into a training data set and a test data set. As mentioned above we are in the quite comfortable situation to have a rich set of validation points. A more realistic case in line with practice is thus to use also the fitting set for early stopping and the ensemble building. Moving from Step III to Step IV does not give any rise of concerns with respect to approximation quality as the errors have still been on a moderately low level. This has also held for the ensemble where a choice of the best 10 nets based on the fitting data rather than OOS validation data has still given very good results. As a last step we have reduced the number of fitting points and we also see here that the neural networks have clearly outperformed the LSMC-Pol approach for the SCR region data set. Let us finally note that the LSMC-Pol proxies largely

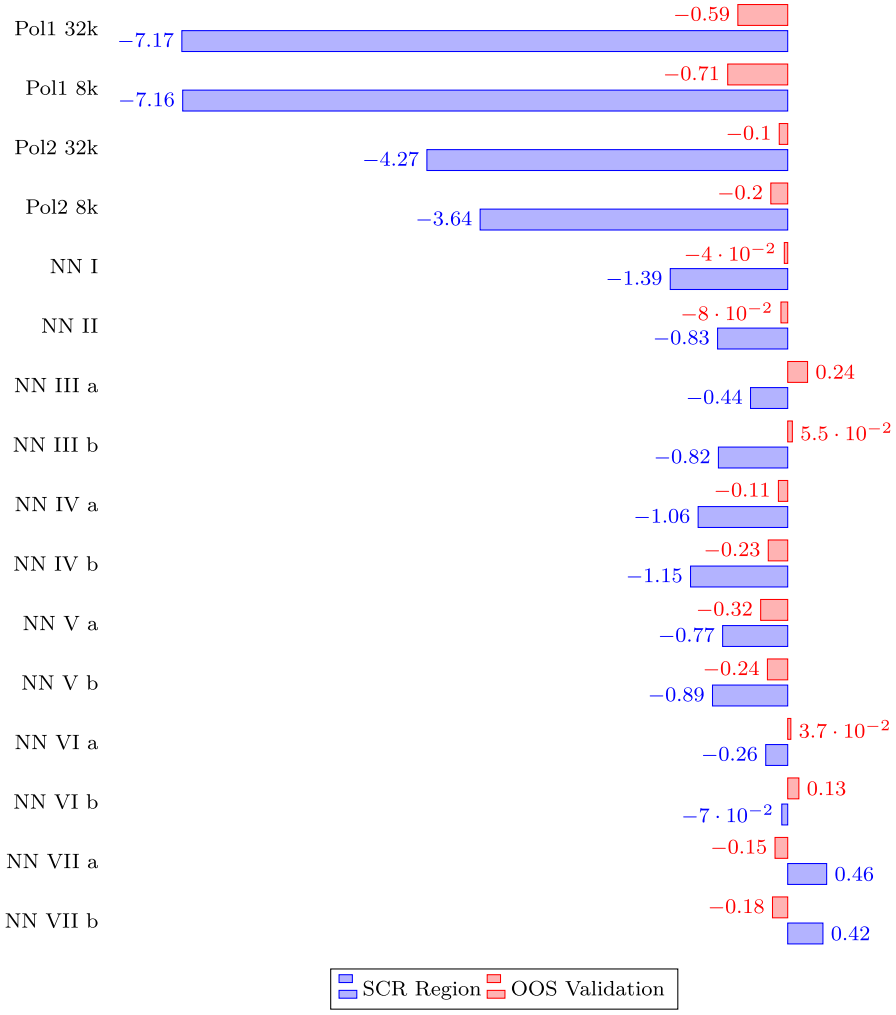

Fig. 11 ME (in 10^{-2}) calculated for LSMC-Pol and LSMC-NN with several steps

maintain the goodness of fit by reducing the number of fitting points from 32k to 8k, i.e. Pol1 32k results are comparable to Pol1 8k results; the same holds for Pol2.

Following the described procedure from Step I to Step VII we expect that similar results may be achieved for Portfolios 2 and 3. We invite the interested reader to perform the procedure with her or his best choice of hyperparameters.

To sum up, we can conclude that even if we reduce the number of training data points in our use case, we still get good results, outperforming the LSMC-Pol approach. Implementing an efficient hyperparameter search algorithm leads to less computational time and strong approximation quality, which is an important insight for practice.

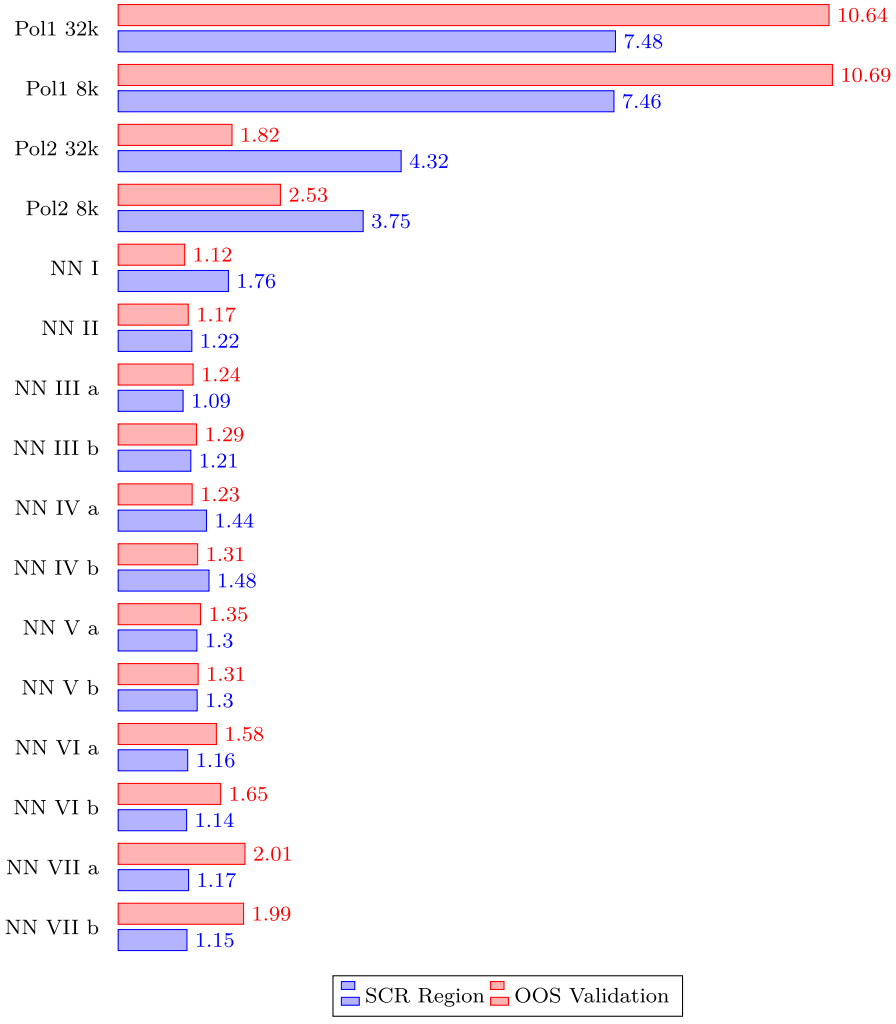

Fig. 12 MAE (in 10^{-2}) calculated for LSMC-Pol and LSMC-NN with several steps

5.6 Further practical considerations

In addition to a purely quantitative evaluation of the different machine learning techniques as carried out above, selected qualitative aspects can also be taken into account when assessing the practical suitability of the various methods. These aspects include the smoothness of risk profiles and the ability to provide a meaningful extrapolation, i.e. a reasonable behavior of the derived proxy functions beyond the training area.

In the present case, the former aspect arises as a requirement from the economic interpretation of the risk factors and their effect on the selected insurance portfolios. The latter aspect arises as a general problem in the context of the risk evaluation

on the basis of proxy functions, that are necessarily derived on a bounded training space that does not cover the entire domain of the potentially unbounded underlying statistical distribution of the risk factors.

In the LSMC-Pol approach, depending on the choice of the basis functions, some conclusions on these two aspects can already be drawn on the basis of analytical considerations. In the case of generalized proxy functions that are derived via alternative machine learning techniques, at least a visual or rather empirical inspection of selected risk profiles is possible. In the context of the investigations presented here, no obvious undesired behaviors related to the above mentioned aspects have been detected. The extent to which the extrapolation delivers meaningful results would have to be checked in each individual case on the basis of an extended OOS validation carried out on a further enlarged data set. In principle, however, the results found in this investigation allow the conclusion that neural networks are able to depict a reasonable behavior beyond the training space. This can also be seen as an advantage over other alternative machine learning techniques such as tree-based methods, which cannot provide any meaningful extrapolation by construction due to the way how the risk factor space is partitioned.

6 Conclusion

In this paper we close the void and report the best proxy function results which we have obtained in the work with our data. We do not provide the entire code, as portions of this code are being used by some companies. But we have provided enough ingredients for a complete comprehension and replication of our approach.

We are convinced to have shown that there is a clear improvement in the quality of proxy functions when using neural networks compared to the current state of the art in the industry, namely the polynomial proxy functions. The implementation may for many appear surprisingly easy, as most of the complex training algorithms are already efficiently implemented in the freely available libraries.

The flexibility and accuracy of neural network models open the door to a variety of further applications, ranging from asset-liability management to product management as well as to IFRS sensitivities and forecasts.

We look forward to seeing the results of interested researchers and practitioners. It is for sure conceivable if not even likely that tree-based methods like random forests or boosted trees may show a comparable degree of accuracy as the ensembles of neural networks we have constructed and presented in this paper. An advantage of the tree-based models is their somewhat better explainability. In our analyses we have made first attempts to fit various tree-based models, using a grid search approach for a series of hyperparameters. The results have, however, never matched those obtained with neural networks and presented in this paper.

Acknowledgements The authors would like to thank Julia Jagdhuber for various runs of the neural network code used in the article.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Andersen L, Broadie M (2004) A primal-dual simulation algorithm for pricing multi-dimensional American options. Manag Sci 50(9):1222–1234. https://doi.org/10.1287/mnsc.1040.0258
2. Bauer D, Ha H (2015) A least-squares Monte Carlo approach to the calculation of capital requirements. The World Risk and Insurance Economics Congress, Munich
3. Bettels C, Fabrega J, Weiß C (2014) Anwendung von Least Squares Monte Carlo (LSMC) im Solvency-II-Kontext-Teil 1. Der Aktuar 2:85–91
4. British Actuarial Journal (2018) What data science means for the future of the actuarial profession: abstract of the London discussion. Br Actuar J 23:E16. https://doi.org/10.1017/S1357321718000053
5. Castellani G, Fiore U, Marino Z, Passalacqua L, Perla F, Scognamiglio S, Zanetti P (2021) Machine learning in nested simulations under actuarial uncertainty. Math Stat Methods Actuar Sci Finance
6. Cheridito P, Ery J, Wüthrich MV (2020) Assessing asset-liability risk with neural networks. Risks 8(1). https://doi.org/10.3390/risks8010016. https://www.mdpi.com/2227-9091/8/1/16
7. Efron B, Hastie T (2016) Computer age statistical inference, 1st edn. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9781316576533
8. European Parliament, European Council (2009) Directive 2009/138/EC on the taking-up and pursuit of the business of insurance and reinsurance (Solvency II). Directive, articles 112–127
9. Fernandez-Arjona L (2021) A neural network model for solvency calculations in life insurance. Ann Actuar Sci 15(2):259–275. https://doi.org/10.1017/S1748499520000330
10. Geron A (2019) Hands-on machine learning with Scikit-learn, Keras and TensorFlows, 2nd edn. O'Reilly
11. Glasserman P (2004) Monte Carlo methods in financial engineering, 1st edn. Springer, Berlin
12. Golub G, van Loan C (1996) Matrix computations, 3rd edn. Johns Hopkins University Press, Baltimore
13. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press. http://www.deeplearningbook.org
14. Hejazi SA, Jackson KR (2017) Efficient valuation of SCR via a neural network approach. J Comput Appl Math 313:427–439
15. Jonen C, Meyhöfer T, Nikolić Z (2020) Comprehensive internal model data for three portfolios. https://github.com/DeutscheAktuarvereinigung/insurance_scr_data
16. Kopczyk D (2018) Proxy modeling in life insurance companies with the use of machine learning algorithms, Working Paper. https://ssrn.com/abstract=3396481
17. Koursaris A (2011) A least squares Monte Carlo approach to liability proxy modeling and capital calculation. Technical Notes Barrie+Hibbert. https://www.yumpu.com/en/document/view/52508 52/a-least-squares-monte-carlo-approach-to-liability-barrie-hibbert
18. Krah AS, Nikolić Z, Korn R (2018) A least-squares Monte Carlo for proxy modeling in life insurance. Risks 6(2). https://doi.org/10.3390/risks6020062

19. Krah AS, Nikolić Z, Korn R (2020) Least-squares Monte Carlo for proxy modeling in life insurance: neural networks. Risks 8(4). https://doi.org/10.3390/risks8040116
20. Krah AS, Nikolić Z, Korn R (2020) Machine learning in least-squares Monte Carlo proxy modeling of life insurance companies. Risks 8(1). https://doi.org/10.3390/risks8010021
21. Longstaff FA, Schwartz ES (2001) Valuing American options by simulation: a simple least-squares approach. Rev Financ Stud 14(1):113–147
22. Nikolić Z, Jonen C, Zhu C (2017) Robust regression technique in LSMC proxy modeling. Der Aktuar 1:8–16
23. Prajit Ramachandran QVL Barret Zoph (2017) TensorFlow: large-scale machine learning on heterogeneous systems. https://arxiv.org/pdf/1710.05941v1.pdf
24. Towards Data Science Inc (2021) Towards data science. https://towardsdatascience.com/