risks

MDPI

Article

Least-Squares Monte Carlo for Proxy Modeling in Life Insurance: Neural Networks

Anne-Sophie Krah [1], Zoran Nikolić [2,*] and Ralf Korn [1,3]

1 Department of Mathematics, TU Kaiserslautern, 67653 Kaiserslautern, Germany; anne-sophiekrah@web.de (A.-S.K.); korn@mathematik.uni-kl.de (R.K.)
2 Mathematical Institute, University Cologne, Weyertal 86-90, 50931 Cologne, Germany
3 Department of Financial Mathematics, Fraunhofer ITWM, 67663 Kaiserslautern, Germany
* Correspondence: znikolic@uni-koeln.de

Received: 29 September 2020; Accepted: 28 October 2020; Published: 4 November 2020

Abstract: The least-squares Monte Carlo method has proved to be a suitable approximation technique for the calculation of a life insurer's solvency capital requirements. We suggest to enhance it by the use of a neural network based approach to construct the proxy function that models the insurer's loss with respect to the risk factors the insurance business is exposed to. After giving a mathematical introduction to feed forward neural networks and describing the involved hyperparameters, we apply this popular form of neural networks to a slightly disguised data set from a German life insurer. Thereby, we demonstrate all practical aspects, such as the hyperparameter choice, to obtain our candidate neural networks by bruteforce, the calibration ("training") and validation ("testing") of the neural networks and judging their approximation performance. Compared to adaptive OLS, GLM, GAM and FGLS regression approaches, an ensemble built of the 10 best derived neural networks shows an excellent performance. Through a comparison with the results obtained by every single neural network, we point out the significance of the ensemble-based approach. Lastly, we comment on the interpretability of neural networks compared to polynomials for sensitivity analyses.

Keywords: least-squares Monte Carlo method; proxy modeling; life insurance; Solvency II; neural networks; machine learning; ensemble method

1. Introduction

Solvency Risk Capital

Under Solvency II regulation, insurance companies are required to determine a full probability distribution forecast of the changes in their available capital (AC) over a one-year period. *Solvency Capital Requirement* (SCR) represents the change in AC (usually a loss) which is exceeded in only 0.5% of all one-year scenarios. For life insurers Solvency II regulation poses an enormous computational challenge, since they not only need to compute the change in AC in a large number of one-year scenarios, but for each of these scenarios they have to perform a stochastic valuation of the future cash flows.

If a company does not avoid this computational challenge by opting for the so-called Standard Formula which allows them to estimate SCR without deriving the probability distribution forecast, it usually uses a proxy function for the stochastic valuation of the future cash flows.

However, even when using using proxies, we hear reports from insurers that they need as much computing power for risk management applications—primarily for SCR calculations—as they do for the administration of all insurance policies. Therefore, a large saving potential can be realized if the proxies become more efficient and accurate.

LSMC Proxy Modeling

This is the third in our series of articles on least-squares Monte Carlo (LSMC) proxy modeling of life insurance companies for risk management. We introduced the context of proxy modeling in the first article Krah et al. (2018) by establishing a suitable theoretical framework and by outlining the current state of implementation in the industry. As proxy functions we proposed polynomials, which we built up of their basis functions by an adaptive model choice algorithm, using ordinary least-squares regression. In the second article Krah et al. (2020), we replaced the ordinary least-squares regression by a collection of other regression techniques, which constituted the major ingredient of the machine learning algorithm presented in the calibration part. Correspondingly, the overall setting explained in the LSMC framework of Krah et al. (2018) remained unchanged, but the regression part in the process was modified.

In the outlook of our second article Krah et al. (2020), we suggested additional analyses with other techniques, such as neural networks and tree-based methods. We now partially follow up on this task by showing how artificial neural networks can be trained in the LSMC context and comparing their performance to the other methods tested by us so far. For the numerical results we utilize the same data of a life insurance company with heavy guarantees introduced in Krah et al. (2018).

LSMC Framework under Solvency II

Our starting point is the LSMC framework proposed by Krah et al. (2018) with the goal of deriving the full loss distribution for life insurance companies over a one-year time horizon. The core idea embedded in this framework is to replace the computationally infeasible full Monte Carlo valuation of a company's future cash flows (also called *nested stochastics* or *nested simulation*) by a smaller number of Monte Carlo simulations. A supervised machine learning algorithm is used for a translation of the results of these Monte Carlo simulations into a proxy function of the company's losses with respect to the risk factors it is exposed to.

Hence, instead of performing the expensive task of simulating the cash-flow-projection (CFP) models for the full nested stochastics approach, proxy functions taking the role of the CFP model are used for prediction. For practical applications, it is of course important that the proxy functions can be evaluated in a straightforward and inexpensive way. For instance, as already mentioned above, in Krah et al. (2018) we used polynomials, which are notoriously easy to evaluate. Once an insurance company has a well-calibrated proxy function, it derives its full empirical distribution of losses over the coming year by plugging all real-world scenarios into the proxy function.

Deep Learning

As in our second article Krah et al. (2020), we adopt the LSMC framework from Krah et al. (2018) without any changes, apart from the calibration and validation parts. In particular, we assume the same data sets are available for the regression task. To distinguish the different proxy functions, in the following section we denote the original LSMC approach with polynomial functions as *LSMC-PF*, and the new deep learning approach with neural networks as *LSMC-NN*.

Contrary to the approach in LSMC-PF, as well as in our analyses of the various machine learning methods in the second article (specifically: GLM, GAM, FGLS regression, MARS, kernel regression), the validation points are not only used for out-of-sample validation in LSMC-NN but also for calibration. In LSMC-NN we utilize the *ensemble method*, according to which a selection of "well-fitted" neural networks is averaged in and *ensemble*, that then serves as the final regression function. For the composition of such an ensemble but also for the stopping criterion in the training of the neural networks, we rely on the same out-of-sample validation points that were used to judge the approximation quality of the proxies in LSMC-PF only after calibration. To find "well-fitted" neural networks from the wide range of options, we apply a quasi-random Sobol hyperparameter search algorithm, which is essentially a brute force method.

Related Literature

With the growing popularity of deep neural networks, there are also related applications of neural networks to the problem of calculating the solvency capital requirement (SCR). As examples, we mention the work in Hejazi and Jackson (2017) where an interpolation approach, based on neural networks, is used inside the nested stochastics approach and Castellani et al. (2018), who apply a neural network in an LSMC framework on an artificial data set with seven risk factors. For sensitivity analyses of hyperparameter variations in neural networks, Born (2018) and Schelthoff (2019) perform test calculations based on LSMC data of life and health insurance companies. In Kopczyk (2018), an overview of LSMC and curve fitting methods is given comparing them to machine learning alternatives using a reinsurer's data. A further reference, where neural networks are used inside an LSMC framework is in Frerix (2018), which is based on a much smaller industry type framework than our example and where the neural networks are used to imitate the pathwise valuation of the assets.

2. Calibration and Validation in the LSMC Framework

Before we introduce and apply neural networks as a regression tool in the LSMC framework, we recapitulate and modify some of its central ingredients. We describe the data sets we will use in our numerical experiments, tailor the calibration procedure to an ensemble approach with neural networks and define validation figures, based on which we will compare the approximation quality of the LSMC-NN to the LSMC-PF approach.

2.1. Fitting and Validation Points

Machine learning techniques and in particular neural networks tend to be used in tasks where the examples of the relationship which need to be described (input data) are abundant but not well-structured. In our task of approximating the liabilities of an insurance company, the data are usually well-structured. Moreover, the user has full control over the data, since the CFP models are in itself deterministic tools. As long as the input does not change, a CFP model will always produce identical output.

The necessity to use proxy models arises from the fact that CFP models are expensive to run. Companies often invest significant amounts of money in server farms or book considerable cloud capacities for their Solvency II reporting and company steering. Yet almost no life insurer is able to perform as many runs as would be necessary in the full nested stochastics approach. Since the calculation capacities are limited, the scenario budget has to be cleverly used. For any practical application, at least two sets of scenarios have to be produced:

- *Fitting scenarios*, which should be uniformly distributed within the fitting space, so that they cover the space of real-world scenarios possibly well (in the neural network terminology the *training scenarios);*
- *Validation scenarios*, which are used for an assessment of the goodness-of-fit of the proxy function, and which do not coincide with the aforementioned scenarios (in the neural network terminology the *test scenarios*).

As a rule of thumb, the scenario budget is evenly split between fitting and validation scenarios.

For the fitting scenarios, we strive to get a good coverage of the entire fitting space. The fitting space is a hypercube, defined by the ranges of the risk factors. A uniform coverage of the fitting space with scenarios is important, as the proxy function of the losses is derived based on these scenarios. Insurance companies want to be able to evaluate the proxy function in the entire space of possible real-world scenarios. In the terminology of Krah et al. (2018), given d risk factors the life insurance company is exposed to, the fitting space is a d-dimensional hypercube

$$S_{\text{fit}} = \Pi_{l=1}^{d}[a_l, b_l] \subset \mathbb{R}^d,$$

where the intervals $[a_l, b_l]$, $l = 1, \ldots, d$, indicate the domains of the risk factors.

The risk factors are typically modeled with unbounded distributions, such as normal and lognormal distributions. The endpoints of the intervals $[a_l, b_l]$ are hence chosen such that the probability of a risk factor value being outside of the interval is low. In practical implementations, we have observed 0.1%- and 99.9%-quantiles as values for a_l and b_l. Sometimes the boundaries are additionally increased by a certain amount if the output is strongly dependent on this risk factor. A possible example for such a risk factor is the credit risk. The reason for the extension of the fitting range comes from the expectation that a very improbable value of this risk factor in combination with other risk factors may lead to a loss close to the 99.5%-quantile of the loss distribution forecast, which is defined as the SCR under Solvency II. Further, such an extension stabilizes the modeling on a wider range.

The validation scenarios are generally chosen from the d-dimensional hypercube, constituting the fitting space, as they should measure the accuracy of the proxy function in the area that was used for the regression. Usually, the validation scenarios must not coincide with the fitting scenarios since they should help in preventing an overfitting. For more details on which paradigms exist for the selection of suitable validation scenarios, see Section 3.3.1 of Krah et al. (2018).

To obtain the *fitting* and *validation values* (in the neural network terminology the *training* and *test values*), the Monte Carlo simulations of the CFP model are carried out with respect to the fitting and validation scenarios. While the scenarios are the realizations of the explanatory variables in the regression or training, the values denote the realizations of the response variable, such as the AC or best estimate liability (BEL), which we aim to find a proxy function for. In the Numerical Experiments section below, we will only derive proxy functions for the best estimate liability. For more information on which risk factors are typically taken into account as explanatory variables, how exactly the resulting *fitting* and *validation points* (in the neural network terminology the *training* and *test data sets*) are obtained from simulation, which different properties they should have to suit well the calibration or validation needs, and on how to ensure these properties, see Sections 3.1 and 3.3 of Krah et al. (2018).

2.2. Nested Stochastics and Capital Region Points

For the numerical example we analyze in our series of articles, we have furthermore produced two additional sets of scenarios which in practical implementations cannot exist at the time of the regression:

- *Nested stochastics scenarios*, which constitute the scenarios from the real-world distribution forecast with the highest losses;
- A subset of the nested stochastics scenarios, which we call *capital region scenarios*, which include the scenarios that are close to the scenario leading to the SCR.

The nested stochastics scenarios in our example include the scenarios causing the highest 5% of the losses, whereas the capital region scenarios lead to the highest 0.3–0.7% losses. The reason for this choice is that the SCR is defined as the loss caused by at most 0.5% of all scenarios. This means that the capital region scenarios lead to losses similar to the SCR.

Why do we claim above that these scenario sets cannot exist before the proxy function has been derived? In order to be able to identify the highest 5% of the losses, one must obtain the loss distribution. However, as the nested stochastics approach is not computationally feasible, a proxy function itself must be evaluated. How do we identify these scenarios? We fit a traditional ordinary least-squares polynomial proxy function with the help of the adaptive model choice algorithm as described in Krah et al. (2018). After having obtained this proxy function we were able to evaluate it at the real-world scenarios to derive the full loss distribution.

As explained in Section 5.2 of Krah et al. (2018), after performing the CFP model valuations for the highest 5% of the losses, it was clear that the order of losses was largely preserved with the ordinary least-squares polynomial. Hence, the assumption that, for example, the chosen capital region scenarios indeed represented the 0.3–0.7% highest losses, or losses very close to that, was shown to be valid.

Similar to the fitting and validation points, we obtain the *nested stochastics* and *capital region points* from carrying out the Monte Carlo simulations of the CFP model with respect to the nested stochastics and capital region scenarios. For additional details on the properties of these sets, see Section 2.3.1 of Krah et al. (2020).

2.3. Calibration Procedure

In Section 3.2 of Krah et al. (2018) we explain in detail a calibration procedure relying on adaptive forward step wise regression with polynomials and the so-called principle of marginality. For instance, this approach has also been applied by Bettels et al. (2014) in the LSMC context, but it can be used in any high-dimensional variable selection applications. Alternative adaptive variants, based on the methodology of Hocking (1976), which are only suited for applications with at maximum seven risk factors, can be found—for example, in Teuguia et al. (2014). Non-adaptive variants, which are also only applicable in settings with few risk factors, have been employed—for example, by Bauer and Ha (2015).

In Krah et al. (2020), we have systematically extended the adaptive forward step wise approach to other linear regression alternatives, preserving the five major components of the algorithm which we quote here: (1) a set of allowed basis function types for the proxy function, (2) a regression method, (3) a model selection criterion, (4) a candidate term update principle, and (5) the number of steps per iteration and the directions of the algorithm. For instance, we repeat the state-of-the-art choices of components (1)–(5) in the insurance industry:

1. As the function types for the basis functions (1), let only monomials be permitted.
2. Let the regression method (2) be ordinary least-squares (OLS) regression.
3. Let the model selection criterion (3) be Akaike information criterion (AIC) from Akaike (1973).
4. Let the set of candidate terms (4) be updated by the so-called principle of marginality.
5. Let only one basis function be added to the proxy function at each iteration (5), and do not allow removal of polynomial terms (adaptive forward step wise selection).

In Krah et al. (2020), we studied variations to the state-of-the-art choices that included generalized linear models (GLMs) by Nelder and Wedderburn (1972), generalized additive models (GAMs) by Hastie and Tibshirani (1986) and Hastie and Tibshirani (1990), feasible generalized least-squares (FGLS) regression, multivariate adaptive regression splines (MARS) by Friedman (1991), and kernel regression by Watson (1964) and Nadaraya (1964).

Once we move to the non-linear neural networks as regression functions, the first component of the calibration algorithm remains unchanged, that is, a set of allowed basis function types for the proxy function has to be chosen.

In the framework of neural networks, however, we do not increase the complexity of the models successively in an adaptive algorithm until the sweet spot between under- and overfitting (bias-variance tradeoff) is reached. In other words, the procedure does not start with a neural network consisting of, for example, just one hidden layer and one node in this layer, and does not continue by adding nodes and layers to construct more and more complex solutions, which finally become sufficiently flexible to approximate the complexity of the underlying CFP model.

Instead, in the LSMC-NN framework, the calibration algorithm consists of the following five steps (see Section 3 for an introduction into the neural network terminology used below):

1. Decide that neural networks are used as basis function types for the proxy function.
2. Select (randomly) N architectures and calibration procedures for neural networks (choices of *hyperparameters*).
3. Define stopping criteria and train the neural networks (i.e., change their weights using the backpropagation algorithm) according to the specifications in the previous step.
4. Identify the neural networks with the best overall performance on the validation points.
5. Build an ensemble of the P best neural networks (effectively by averaging their predictions).

How do we apply the fitting and validation points in the context of neural networks? For the ordinary least-squares regression with polynomial basis functions, only the fitting points are used for the regression. Therefore the resulting polynomial is a function of only the fitting points, meaning the validation points have no impact on the polynomial structure or coefficients. They are used for an assessment of the suitability of the proxy function only after calibration. It is a matter of governance within the company to decide on what happens if the validation indicates a poor approximation quality of the derived proxy function. It is possible to examine whether the polynomial overfits or underfits the data and to manually amend the terms. Another possibility is to use a different stopping criterion for the adaptive algorithm.

Contrary to that, when we derive neural networks as proxy functions, we propose to define stopping criteria in Step 3 of the calibration algorithm which depend on the validation points. Additionally, we rely on the validation points in subsequent Step 4 for the assessment of whether a neural network performs well. Therefore, the validation points are strictly speaking no out-of-sample points in the choice of the final model. However, in the training part of Step 3 before stopping occurs, only the fitting points are used. Further, it would be possible as well to use the fitting points instead of the validation points for the stopping criteria. In this case, the validation points would not be required in Step 3. However, test calculations have shown that the neural networks resulting from such an approach have a worse approximation quality.

It is also conceivable to allocate a certain number of validation points for a true out-of-sample validation. However, in our numerical experiment, only 51 validation points are available so that no split of this set appears advantageous.

2.4. Validation Procedure

Finding sound and feasible regression methods for the LSMC proxy function calibration is the primary objective of our article series. As in Krah et al. (2020), we employ several validation figures for judging the accuracy of the proxy function. This time, we measure the validation performance of each proxy function on five instead of three different validation sets but calculate again the same five validation figures per set, as in Krah et al. (2020).

The first three validation sets are a validation set, a nested stochastics set and a capital region set. These sets essentially consist of the validation, nested stochastics and capital region points specified in Sections 2.1 and 2.2. Further, they are exactly the sets from Section 4.1.1 of Krah et al. (2020), where the validation set was referred to as the Sobol set due to its predominant composition of 26 Sobol points. As already mentioned above, unlike the validation set, the nested stochastics and capital region sets are not feasible validation sets in the LSMC routine in day-to-day business, as they require massive computational capacities. However, they can be regarded as the natural benchmark for any LSMC-based method and are thus very valuable for this analysis. For more details, see Section 5.2 of Krah et al. (2018).

The last two validation sets are subsets of the nested stochastics and capital region sets. They are obtained from these sets by excluding all points with scenarios lying outside of the fitting space. While, from the nested stochastics set, 107 of 1624 points are removed as a result of this transformation, from the capital region set, 24 of 128 points are removed. The validation figures computed with respect to these data sets demonstrate how the proxy functions perform in settings where extrapolation is not required.

The five validation figures, which we report in our numerical experiments, comprise three normalized mean absolute errors (MAEs) and two mean errors (MEs), that is, means of residuals. The smaller the normalized MAEs are, the better the proxy function approximates the response variable. The normalized MAEs serve only as meaningful indicators as long as the proxy functions do not become too precise. This is a consequence of the Monte Carlo error afflicting each valuation with the CFP model. Furthermore, The MEs should preferably be close to zero. This follows from the fact that they indicate systematic deviations of the proxy functions from the validation values.

Summing up, the first three reported validation figures measure how well the proxy function reflects the response variable in the CFP model, whereas the latter two address the approximation effects on the SCR—compare Section 3.4.1 of Krah et al. (2018).

Let us review the formulae for the validation figures by writing the absolute value as $|\cdot|$ and denoting with L the number of points in the data set. Then we can express the MAE of the proxy function $\widehat{f}\left(x^i\right)$ evaluated at the validation scenarios x^i versus the response values y^i as $\frac{1}{L}\sum_{i=1}^{L}\left|y^i-\widehat{f}\left(x^i\right)\right|$. After normalizing the MAE with respect to the mean of the absolute values of the response variable or the market value of assets, that is, $\frac{1}{L}\sum_{i=1}^{L}\left|d^i\right|$ with $d^i\in\left\{y^i,a^i\right\}$, we obtain the first two validation figures, which are

$$\text{mae}=\frac{\sum_{i=1}^{L}\left|y^i-\widehat{f}\left(x^i\right)\right|}{\sum_{i=1}^{L}\left|d^i\right|}. \tag{1}$$

In the following section, we will refer to (1) with $d^i=y^i$ as the MAE with respect to the relative metric, and to (1) with $d^i=a^i$ (where a^i is the market value of the assets in run i) as the MAE with respect to the asset metric.

The ME is given by

$$\text{res}=\frac{1}{L}\sum_{i=1}^{L}\left(y^i-\widehat{f}\left(x^i\right)\right). \tag{2}$$

The validation value corresponding to the base scenario x^0 is denoted by y^0. In this scenario, no risk factor has an effect on the response variable. In analogy to (1), we introduce the third MAE using the relative metric y^i by the following

$$\text{mae}^0=\frac{\sum_{i=1}^{L}\left|\left(y^i-y^0\right)-\left(\widehat{f}\left(x^i\right)-\widehat{f}\left(x^0\right)\right)\right|}{\sum_{i=1}^{L}\left|y^i-y^0\right|}. \tag{3}$$

The mean of the corresponding residuals is given by

$$\text{res}^0=\frac{1}{L}\sum_{i=1}^{L}\left(\left(y^i-y^0\right)-\left(\widehat{f}\left(x^i\right)-\widehat{f}\left(x^0\right)\right)\right). \tag{4}$$

Finally, let us define the base residual by

$$\text{res}^{\text{base}}=y^0-\widehat{f}\left(x^0\right)=\text{res}-\text{res}^0. \tag{5}$$

It can easily be extracted from (2) and (4), and can therefore be used as a substitute for (4), depending on personal taste.

Analogously to Krah et al. (2020), we will output validation Equation (1) using both the relative and the asset metric, and Equations (2)–(4). Equations (3) and (4) are evaluated with respect to a base value having a very low standard error based on 16,000 inner simulations. The corresponding values in Section 4 are v.mae^0, v.res^0 for the 51 validation points, ns.mae^0, ns.res^0 for the two nested stochastics sets, and cr.mae^0, cr.res^0 for the two capital region sets. As noted in Section 4.1.2 of Krah et al. (2020), where we reported v.res^0 with respect to the base value resulting from only 1000 inner simulations, these values can easily be transformed into each other by adjusting them by the difference of $y^0_{16,000}-y^0_{1000}=14.4$ which the two base values $y^0_{16,000}=14,661.1$ and $y^0_{1000}=14,646.7$ incur. The base residual (5) is just (2) minus (4), hence we will not explicitly state it here.

3. Neural Networks in Proxy Modeling

As we use neural networks to identify the proxy function in the LSMC-NN approach in this article, we are going to sketch their major characteristics in this section to keep the article self-contained.

3.1. A Short Introduction to Neural Networks

While in the literature on neural networks they are typically associated with attractive network graphs and a language that differs from the standard mathematical one, we will concentrate on introducing them in a conventional style.

The first ingredient that distinguishes neural networks from linear regression is the so-called activation function.

Definition 1. *A Lipschitz continuous monotonic function $\phi : \mathbb{R} \to \mathbb{R}$ is called an activation function.*

Typical examples of activation functions include $f(x) = 1/(1 - e^{-x})$ (the logistic or *sigmoid* function), $f(x) = \tanh(x)$, $f(x) = x^+$ (the *ReLU* (rectified linear unit) function) or $f(x) = x^+ - \alpha x^-$ with $\alpha > 0$ (*parametric ReLU* or *Leaky ReLU* with parameter α). They are all non-linear, but differ in their value ranges and degrees of smoothness.

A neural network typically consists of an iterated concatenation of the activation function with affine linear functions. In this standard form it is called a *multilayer perceptron* (see also Wiese et al. (2020) for the definition below).

Definition 2 (Multilayer perceptron)**.** *Let $L, N_0, ..., N_{L+1} \in \mathbb{N}$, ϕ be an activation function, Θ an Euclidean vector space and for any $l \in \{1, ..., L+1\}$ let $a_l : \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$ be an affine mapping. A function $f : \mathbb{R}^{N_0} \to \mathbb{R}^{N_{L+1}}$ defined by*

$$f(x, \theta) = a_{L+1} \circ f_L \circ \cdots \circ f_1(x),$$

where $\circ$ denotes the concatenation, $f_l = \phi \circ a_l$ for all $l \in \{1, ..., L\}$ and ϕ being applied component-wise, is called a multilayer perceptron (MLP) *or a* feed forward neural network (FNN) *with L hidden layers. Here, N_0 represents the input dimension, N_{L+1} the output dimension, $N_1, ..., N_L$ the hidden dimensions and a_{L+1} the mapping onto the output layer. Furthermore, for any $l \in \{1, \ldots, L+1\}$ the function a_l takes the form*

$$a_l(x) = W(l)x + b(l)$$

for some weight matrix $W(l) \in \mathbb{R}^{N_l \times N_{l-1}}$ and bias $b(l) \in \mathbb{R}^{N_l}$. The parameters of the MLP are denoted by

$$\theta := (W(1), \ldots, W(L+1), b(1), \ldots, b(L+1)) \in \Theta.$$

Before we give a graphical illustration of this definition, let us look at its mathematical implications. If we use an FNN as a nonlinear regression tool, then:

- The function $a_l(x)$ is a linear combination of its input variables as in linear regression, the activation function $\phi(.)$ determines the local degree of non-linearity when applied to components of $a_l(x)$;
- The number L that equals the number of compositions of the activation function with the linear inputs $a_l(x)$ determines the global degree of non-linearity that the FNN can achieve (it measures *how deep* the FNN is);
- The numbers N_l determine the size of the linear combination of the local non-linearity at level l;
- The final step to the output variables is a linear combination of the output from the concatenation of the functions before. It ensures that linear regression is a special case of FNN (i.e., $L = 0$).

Of course, one can also use different activation functions ϕ_l for different levels $l \in \{1, \ldots, L\}$ in the definition. Further, an additional function for transforming the output to a given range of values

can be applied to obtain the output (think of a transformation to $[0,1]$ or even $\{0,1\}$ if the classification of input is our aim, but think also of just another activation function).

The usual graphical interpretation of Definition 2 is given by Figure 1.

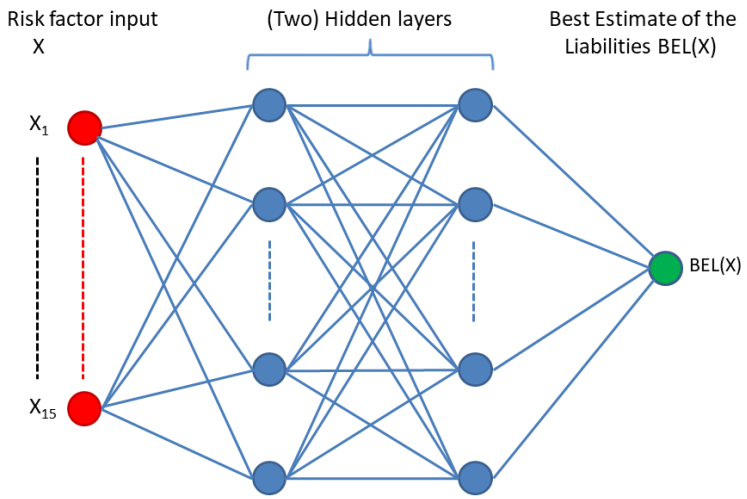

Figure 1. Schematic picture of a fully connected feed forward neural network with two hidden layers illustrating the mapping of risk factor input X to the best estimate of the liabilities $BEL(X)$.

Here, we present a neural network that is similar to those that we are using in our application in Section 4 . The so-called *layers of neurons* (i.e., the vertical collections of circles) start from left to right with the input layer that consists of the $N_0 = 15$ red neurons which are identified with the input variables $X_1, \dots, X_{N_0}$, the risk factors in our application. The right most, green output layer contains the $N_{L+1} = 1$ output neuron(s) which represent the output variable(s), in our case the best estimate of the liabilities $BEL(X)$ as a function of the risk factors X. The layers in between are called *hidden layers* as they are neither input nor output and are therefore never observed. The graph is assumed to be a directed graph—i.e., the lines between the neurons move forward from left to right. They indicate that the variable in the starting neuron enters the function $\phi(a_{l_i}(.))$, $l_i \in 1, ..N_l$, in the neuron at the end of the arrow. As every neuron from a previous layer at level j is connected to all neurons in the following layer at level $j+1$, $j = 0, \dots, L$ and as all lines point to the right, we say that we have a *fully connected forward neural network*. In this article we only consider such FNN and do not refer to other types of neural networks.

Note, in particular, the enormous amount of free parameters $\theta \in \Theta$ in Definition 2 that have to be calibrated to available input and output data pairs $(x^i, y^i) \in \mathbb{R}^{N_0+N_{L+1}}$, $i = 1, \dots, N$ by minimizing a suitable error measure such as the usual least-squares errors. The amount of variables on the one hand adds a lot of flexibility to fit the observed data well, but also bears the danger of overfitting. This can result in bad predictions of the output of not yet observed input—i.e., in a bad generalization behavior.

To avoid overfitting, one divides the available data in a training set to calibrate the parameters and a test set which is used to check if the resulting FNN still works on data that have not been used for calibration. As we dealt with the same aspect already in Krah et al. (2018, 2020), we do not go into further detail here.

3.2. Neural Networks in Action—Choice of Hyperparameters

A neural network is a very powerful tool that requires a careful handling.

While the choice of the parameters $\theta \in \Theta$ has the purpose to obtain the best possible performance of the FNN with respect to fitting the training data and to generalize on the test data set, a lot of decisions have to be made before the actual training and validation process.

All these decisions are summarized under the *choice of hyperparameters*. Hyperparameters determine the form of the neural network (its *architecture*) and the procedure to calibrate the parameters $\theta \in \Theta$ to the training data and further steps on the way to the final form of the network.

The list of hyperparameters that define the architecture of a neural network includes the choice of

- The number of hidden layers L in the neural network;
- The number of nodes N_l in each layer;
- The activation function(s) in the hidden layers;
- The output activation function.

For general neural networks, there are many other features that determine the plan according to what a network can be built. We refer to the usual literature on neural networks for a survey on the further forms (see Goodfellow et al. (2016)).

After having fixed the form of the FNN, we have to calibrate the parameter vector $\theta \in \Theta$ of the FNN. Due to the non-linear activation functions, it is clear that there is no closed analytical form for the vector θ^* that attains the global minimum of the error criterion. Thus, numerical methods of global minimization have to be applied. They are often variants of gradient-based descent methods. However, due to the size of the network, already calculating the gradient of the objective function is not at all easy. The famous backpropagation algorithm and its modern variants (see Goodfellow et al. (2016)) at least allows for an efficient calculation of the gradient. As, however, this can already be too costly, the stochastic gradient method as a kind of Monte Carlo estimate of the gradient is typically used in such cases. The iterative process of performing the calibration is referred to as *training the network*.

Since there is no standard way to perform the calibration task, we have to, for example, decide on the following tasks that are also referred to as hyperparameters:

- Choice of the numerical *optimizer* (typically a suitable version of the gradient descent method);
- Control the step size (the *learning rate*) in each iteration of the calibration to find a compromise between speed of descent and not missing the global minimum by a too large step;
- Possibly look only at a subset of the network (control the *dropout rate*) to make the calculation times feasible—i.e., we restrict the search space of the direction of descent and keep a lot of weights in the calibration constant per step;
- Choice of the (random) weight initialization technique (the *initializer*) by which the starting weights for the training are set;
- Possibly train the network only with respect to randomly shuffled subsets (i.e., *batches*) of the training data in each iteration to accelerate the calculation of the gradient (which is taken to the extreme by minibatches, having *batch sizes* of 1).

There are many more possible features that can be found in, for example, Goodfellow et al. (2016). Actually, in our case, only the number N_0 of the input parameters (i.e., $N_0 = 15$ risk factors of the company, cf. (Krah et al. 2020)) and the dimension $N_{L+1} = 1$ of the output layer are given.

There are various, often heuristic strategies for hyperparameter optimization that can be seen as educated guesses. Popular methods to search the hyperparameter space are random search, quasi-random search (such as using a Sobol sequence) and grid search. There are also more sophisticated methods, such as Bayesian search approaches, that use the performance from already tested hyperparameters. As there is no clearly preferred approach, hyperparameter optimization is an active area of research.

3.3. Neural Networks as a Team—The Ensemble Approach

Ensemble methods can be motivated by the concept of swarm intelligence, by the law of large numbers or by combining local approximation methods. These seemingly unrelated justifications are connected to different ensemble methods. One can train the same neural network on different data sets, which might lead to locally optimal approximations and then use a (possibly) weighted average as the actual global approximation. One can also train nets with different hyperparameters on the same data and finally average the best performing ones for the global approximation. The idea of swarm intelligence comes into the game when there will be majority of votes among different networks.

We will, in the Numerical Experiments section below, use a combination of a quasi-random hyperparameter search via a Sobol sequence for the training of a large amount of FNN, and then built an ensemble by the average over the best performing networks.

4. Numerical Experiments

We apply the LSMC-NN approach, as described in Section 2.3, based on the data sets presented in Sections 2.1 and 2.2. A direct comparison of the validation figures from Section 2.4 for LSMC-NN versus LSMC-PF (cf. Section 4.2 of (Krah et al. 2020)) reveals the superiority of the LSMC-NN approach in this numerical example. Further, we motivate the use of the ensemble method by comparisons of the derived ensemble to the best performing single neural networks.

We produce the fitting points by running the CFP model for N = 25,000 fitting scenarios, with each of them entailing two inner simulations. The (Sobol) validation set is produced based on $L = 51$ validation scenarios with 1000 inner simulations, where the 51 scenarios comprise 26 Sobol scenarios, one base scenario, 15 risk scenarios, where only one risk factor is changed from its base value, and finally 9 scenarios representing the capital region scenarios in the previous year risk capital calculations. The nested stochastics set represents the highest 5% real-world losses and is based on $L = 1638$ outer scenarios with, respectively, 4000 inner simulations. From the 1638 real-world scenarios, 14 show extreme stresses far beyond the boundaries of the fitting space. For this reason, they are excluded from the analysis. The capital region set is based on the $L = 129$ nested stochastics points, which correspond to the solvency capital requirement (=99.5% highest loss) as well as the 64 losses above and below this scenario (=99.3% to 99.7% highest losses). From this set, one point is excluded.

4.1. Neural Networks

4.1.1. Settings

As we have noted above, we utilize the ensemble method, combined with a quasi-random Sobol hyperparameter search procedure, for our neural network training. In one session, we train $N = 300$ neural networks with different hyperparameters and random seeds and select the $P = 10$ best performing ones to build the ensemble. As the criterion for early stopping, we evaluate the mean squared error (MSE) with respect to the validation set. The maximum number of epochs in each training is set to 1600 as we have observed that this is sufficient in our numerical experiment. Only 9 of 300 trained neural networks reach 1600 epochs, and the average number of epochs across all neural networks is 418.

For the practical implementation, we employed the Keras deep learning library in Python—see (Keras team 2020)—which runs on top of the TensorFlow framework—see (TensorFlow team 2020).

Returning to our application, we list below the hyperparameters we vary in the FNN:

1. Number of hidden layers: between 2 and 10;
2. Number of neurons per layer: between 16 and 128 (constant for all layers);
3. Activation function: *Sigmoid*, *ReLU*, *LeakyReLU* (constant for all neurons) with additional parameter LReLUAlpha chosen between 0 and 0.1;
4. Output activation function : *Sigmoid*, *Linear*;

5. Optimizer: *Nadam*, *Adam*, *Adamax*;
6. Learning rate: between 0.0005 and 0.005;
7. Dropout rate: between 0 and 0.4;
8. Initializer: *GlorotUniform*, *RandomNormal*, *RandomUniform*;
9. Batch size: 100, 200, 400, 800, and 1600.

The names of the activation functions, optimizers and initializers refer to the corresponding functions in Keras, see (Keras team 2020).

The choice of the activation functions is, on the one hand, motivated by the fact that the ReLU-function and variants of it are similar to the final payments of financial option contracts. However, as the option prices before maturity are typically smooth functions in their input parameters (i.e., the suitable risk variables), the choice of a smooth activation function, such as the sigmoid function, can also be justified.

Table 1 displays the combinations of hyperparameters which belong to the 10 best performing neural networks in terms of the early stopping criterion.

Table 1. Hyperparameter combinations producing the 10 best performing neural networks.

ID	Hidden Layers	Neurons per Layer	Activation Function	Activation in Output	Optimizer	Learning Rate	Dropout Rate	Initializer	Batch Size
NN1	4	80	Sigmoid	Linear	Nadam	0.003488281	0.003125	GlorotUniform	1600
NN2	6	67	Sigmoid	Linear	Nadam	0.004929688	0.00625	GlorotUniform	400
NN3	3	114	Sigmoid	Sigmoid	Adamax	0.0044375	0.05	GlorotUniform	800
NN4	6	95	Sigmoid	Linear	Adam	0.002662109	0.0390625	GlorotUniform	400
NN5	4	63	Sigmoid	Sigmoid	Adamax	0.001976563	0.06875	RandomNormal	400
NN6	5	44	Sigmoid	Linear	Adam	0.003356445	0.00078125	RandomUniform	1600
NN7	6	101	Sigmoid	Sigmoid	Adamax	0.004332031	0.028125	RandomNormal	400
NN8	6	126	Sigmoid	Linear	Adamax	0.003365234	0.0515625	GlorotUniform	1600
NN9	3	113	Sigmoid	Sigmoid	Adamax	0.003426758	0.14453125	GlorotUniform	800
NN10	5	104	Sigmoid	Sigmoid	Adam	0.001220703	0.1046875	RandomNormal	400

Since it is not our task to replicate, for example, the annual cash flows of the CFP models, it appears sufficient to use solely FNNs and to not extend our analyses to other forms of neural networks. For replication tasks in insurance risk management, focusing on the cash flows themselves and not only their present values, such as in our application, for example, within the asset liability management or company planning, recurrent neural networks are expected to be better suited. See Kiermayer and Weiß (2019) for an application of recurrent networks as a substitute of K-means clustering for grouping insurance contracts.

4.1.2. Brute Force Helps

By training 300 networks with different combinations of hyperparameters and at the same time varying the random seed for the initialization of the weights (and other subtleties within Keras and TensorFlow), we have produced a wide range of results. The box plots in Figure A1 display the MAE (without normalization), MSE and ME of the 300 resulting neural networks with respect to the validation set. An important insight from these training runs is that there is no clear correlation between the hyperparameter values and goodness-of-fit. While most columns in Table 1 reflect this insight very well by covering wide parts of the hyperparameter search space, the activation function column includes, by chance, only Sigmoid. However, LeakyReLU and ReLU, which first appear at positions 14 and 28, are not far behind.

For apparently similar hyperparameter combinations, very different results are produced. To challenge this observation, we have taken it to the extreme by training, in another session, 300 networks based on the same hyperparameter combinations and varying only the random seed. The box plots in Figure A2 not only confirm this observation, they also suggest that, besides the global minimum, which is found in only approximately 10% of the test runs, there can be further local minima which might be attained in the majority of the runs. By using the hyperparameter combinations of the

best produced single neural network from the first training session (NN1 in Table 1), we reveal the potential numerical instability of the top results and hence their high dependence on the random seed. In practice, it is thus necessary to control the seed for reproducibility.

As we have already seen by looking at Table 1, the converse observation holds as well: the neural networks, which lead to very good or very poor results, do not seem to point to a specific hyperparameter or a combination thereof, which steers the result. Hence, it is not helpful to restrict the hyperparameter search space. This is especially true when taking into account new training sessions after CFP model updates where no hyperparameter combination has yet proven its suitability in combination with a certain random seed.

4.1.3. Results

Following up on our results in Krah et al. (2018, 2020), we describe here in detail the comparison of the ensemble of the 10 best performing neural networks to the best ordinary least-squares (OLS) polynomial. Table 2 reports the corresponding validation Equation (1), where we set $d^i \in \{y^i, a^i\}$, and (2)–(4) with respect to the validation, nested stochastics and capital region set for the OLS polynomial in the first row and the ensemble in the fifth row. For a comparative analysis, this table depicts, in addition, the figures for the best found GLM, GAM and FGLS polynomial from Krah et al. (2020). Similar to the OLS polynomial, these proxy models were obtained by suitable applications within an adaptive machine learning algorithm. We forego to state the numbers for the best derived MARS and kernel regression models, as these performed significantly worse than the aforementioned methods. Table 3 repeats the figures of Table 2 for the validation set and contrasts them now to the figures for the reduced nested stochastics and capital region sets, which are obtained from the original sets by the exclusion of the points with scenarios lying outside of the fitting space. Moreover, for the ensemble of neural networks, we present in Tables 4 and 5 the aforementioned figures once again and additionally the 10 best performing single neural networks that serve as the building blocks of the ensemble.

For a graphical comparison of the 10 best performing neural networks, their ensemble and the OLS polynomial, we plot their one-dimensional curves with respect to the risk factors X_1 and X_8, respectively, in Figures A3 and A4. Additionally, we provide two-dimensional curves of the ensemble with respect to risk factors X_1 and X_8 as well as X_8 and X_9, respectively, in Figures A5 and A6.

4.1.4. Neural Networks Outperform the Adaptive Method with Polynomials and Co

As the major result from our numerical experiments, we show that the LSMC-NN approach outperforms the LSMC-PF approach. As already stated above, in Tables 2 and 3, we report besides the best OLS regression and neural network results also the results obtained by the best adaptive GLM, GAM and FGLS approaches. To keep the analysis concise, we will not go into the details of these additional machine learning methods, though. For an extensive discussion of these results, as well as of the rather poor results generated by the MARS algorithm and adaptive kernel regression approaches, we refer the reader to Section 4 of Krah et al. (2020).

The neural networks were chosen according to the MSE on the validation set. This needs to be considered when reviewing the results in Table 2: The listed figures for the validation set represent for the polynomials, GLMs and GAMs unseen data, whereas for the neural networks they are part of the selection process.

We emphasize another important remark regarding Table 2: The very low MEs of, for example, the OLS polynomial in the capital region—i.e., cr.res $= 0.8$, and on the nested stochastics set, i.e., ns.res$^0 = -1.6$, must be interpreted as random results. As we indicate in Section 5.2 of Krah et al. (2018), the polynomial of the own funds significantly underestimates the base value, whilst being quite accurate in the capital region. For losses larger than in the capital region, the polynomial overestimates the simulation results of the own funds, whereas it underestimates them for losses smaller than in the capital region. For the polynomial of the best estimate liability, the converse is true—see Section 4.2 of Krah et al. (2020).

For this article, we have additionally amended our nested stochastics and capital region sets by removing all points with scenarios lying outside of the fitting space. By excluding the extrapolation effects in this way, we reduce the random effects in the comparison of the polynomials, GLM, GAM and neural networks. The fitting space does not cover all possible realizations of the risk factors as it is by definition bounded, whereas the distributions of the risk factors are unbounded. This necessarily leads to some one-year real-world scenarios lying outside of the fitting space. Since the fitting points do not provide any information about the behavior of the CFP model outside of the fitting space, for a purely technical comparison, it is advisable to remove these points from the statistics we use to judge the goodness-of-fit. By doing so, we have to beware of putting zero weights on the extrapolation performance of the proxy functions in the statistics. The accuracy of a proxy model outside of the fitting space is essentially random.

Table 2. Validation figures for the best OLS and FGLS polynomials, GLM and GAM, derived in Krah et al. (2020), and the ensemble of 10 best performing neural networks. MAEs in %.

v.mae	v.mae^a	v.res	v.mae^0	v.res^0	ns.mae	ns.mae^a	ns.res	ns.mae^0	ns.res^0	cr.mae	cr.mae^a	cr.res	cr.mae^0	cr.res^0
300–886 AIC OLS polynomial with 225 terms														
0.194	0.186	−8.7	5.329	19.9	0.268	0.259	−30.2	4.200	−1.6	0.168	0.165	0.8	5.007	29.4
300–886 AIC GLM with inverse gaussian random component, $\frac{1}{\mu^2}$ link and 251 terms														
0.174	0.166	−12.4	4.067	11.1	0.193	0.186	−14.6	3.833	8.8	0.188	0.184	17.3	6.266	40.8
150–443 LOO-CV GAM with thin plate regression splines, normal random component, identity link and 151 terms														
0.212	0.203	−9.8	5.837	22.4	0.230	0.223	−24.3	3.575	7.9	0.173	0.170	8.3	6.337	40.4
300–886 AIC FGLS polynomial with 14 multiplicative heteroscedasticity variance modeling terms and 259 terms														
0.172	0.165	−14.4	3.868	−4.0	0.134	0.129	−2.1	3.504	8.2	0.214	0.210	28.2	6.063	38.6
Ensemble of the best 10 neural networks														
0.091	0.087	−1.8	2.583	−9.8	0.079	0.077	4.7	2.490	−3.4	0.145	0.142	15.9	3.018	7.9

In Table 3, this becomes apparent: The best OLS polynomial does not anymore virtually perfectly match the capital region scenarios—cf. 0.8 in Table 2 versus −10.3 in Table 3. After the removal of the points with the scenarios outside of the fitting space, the ensemble of neural networks performs better than the OLS polynomial in all metrics we consider here.

Table 3. Validation figures for the best OLS and FGLS polynomials, GLM and GAM derived in Krah et al. (2020), and the ensemble of 10 best performing neural networks after the exclusion of the points with scenarios outside of the fitting space. MAEs in %.

v.mae	v.mae^a	v.res	v.mae^0	v.res^0	ns.mae	ns.mae^a	ns.res	ns.mae^0	ns.res^0	cr.mae	cr.mae^a	cr.res	cr.mae^0	cr.res^0
300–886 AIC OLS polynomial with 225 terms														
0.194	0.186	−8.7	5.329	19.9	0.253	0.244	−34.6	3.608	−6.0	0.124	0.121	−10.3	4.159	18.4
300–886 AIC GLM with inverse gaussian random component, $\frac{1}{\mu^2}$ link and 251 terms														
0.174	0.166	−12.4	4.067	11.1	0.169	0.163	−19.6	3.046	3.8	0.107	0.105	2.5	4.942	26.0
150–443 LOO-CV GAM with thin plate regression splines, normal random component, identity link and 151 terms														
0.212	0.203	−9.8	5.837	22.4	0.220	0.212	−28.3	3.031	3.8	0.120	0.117	−3.6	5.609	28.6
300–886 AIC FGLS polynomial with 14 multiplicative heteroscedasticity variance modeling terms and 259 terms														
0.172	0.165	−14.4	3.868	−4.0	0.114	0.110	−5.8	2.961	4.6	0.144	0.141	17.6	5.341	27.9
Ensemble of the best 10 neural networks														
0.091	0.087	−1.8	2.583	−9.8	0.067	0.065	3.1	2.292	−5.0	0.091	0.090	7.7	2.350	−0.4

It is worth noting that the near perfect SCR match of the ensemble of 10 best performing neural networks in metric cr.res^0 is an outcome of the offsetting of two deviations: 8.1 in the base value and 7.7 in the average over all capital region scenario—i.e., $7.7 - 8.1 = -0.4$. In the case of the OLS polynomial, the corresponding deviations are −28.7 in the base value and −10.3 for the average in the capital region—i.e., $-10.3 - (-28.7) = 18.4$.

4.1.5. Ensemble Beats Single Networks

Another important result is that the ensemble method indeed helps in achieving an improved approximation quality compared to the more parsimonious alternative of choosing a single neural network.

By taking a look at Table 4, we see that the ensemble scores second place in terms of v.mae/v.mae^a and cr.mae/cr.mae^a and first place in terms of ns.mae/ns.mae^a. Further, it scores third place in terms of v.mae^0 and second place in terms of ns.mae^0 and cr.mae^0. In terms of the ME metrics, the ensemble shows a decent performance.

Table 4. Validation figures for the ensemble and the 10 best performing neural networks. MAEs in %.

ID	v.mae	v.mae^a	v.res	v.mae^0	v.res^0	ns.mae	ns.mae^a	ns.res	ns.mae^0	ns.res^0	cr.mae	cr.mae^a	cr.res	cr.mae^0	cr.res^0
Ens.	0.091	0.087	−1.8	2.583	−9.8	0.079	0.077	4.7	2.490	−3.4	0.145	0.142	15.9	3.018	7.9
NN1	0.084	0.081	−0.7	2.308	−6.2	0.107	0.103	14.2	2.393	8.7	0.180	0.176	25.6	3.452	20.1
NN2	0.092	0.088	0.6	2.458	−5.5	0.124	0.120	17.1	2.645	11.0	0.191	0.187	27.4	3.616	21.3
NN3	0.095	0.091	−0.4	3.100	−14.6	0.085	0.083	6.8	3.014	−7.3	0.154	0.150	17.1	3.304	2.9
NN4	0.108	0.103	−5.5	3.868	−21.6	0.092	0.089	5.8	3.405	−10.3	0.169	0.166	19.4	3.296	3.3
NN5	0.105	0.100	−0.1	2.665	−7.2	0.108	0.104	5.9	2.989	−1.2	0.161	0.158	15.9	3.407	8.8
NN6	0.109	0.104	−0.6	2.705	−1.2	0.092	0.089	−4.2	2.627	−4.9	0.124	0.121	8.2	2.799	7.5
NN7	0.103	0.098	−7.0	3.182	−15.0	0.090	0.087	−0.8	3.090	−8.8	0.148	0.145	10.9	3.344	2.8
NN8	0.102	0.098	−0.8	3.229	−14.4	0.094	0.091	4.1	3.400	−9.5	0.151	0.148	12.6	3.510	−1.0
NN9	0.119	0.114	−2.0	3.100	−7.9	0.107	0.103	2.1	3.213	−3.7	0.158	0.155	13.8	3.511	8.0
NN10	0.126	0.120	0.0	3.161	7.0	0.107	0.103	6.0	3.358	13.0	0.182	0.178	21.1	4.732	28.1

Only network NN1 trumps the ensemble in terms of v.mae/v.mae^a with 0.084/0.081 (in %) versus 0.091/0.087 (in %). Since NN1 was chosen as the network with the lowest validation deviation, it is not surprising that, in terms of the other metrics on the validation set, NN1 performs better as well. However, in terms of ns.mae/ns.mae^a and cr.mae/cr.mae^a, NN1 scores only ninth out of eleven. On the capital region set, this is an even worse performance than what the OLS polynomial achieves. In terms of the ME metrics on the nested stochastics and capital region set, NN1 shows a comparatively poor performance as well. Now, consider NN6 scoring first, in terms of cr.mae/cr.mae^a. It scores seventh in terms of v.mae/v.mae^a and fourth in terms of ns.mae/ns.mae^a. Further, it performs very well in terms of the ME metrics as it approximates the base value very well.

Table 5 confirms these results with respect to the reduced nested stochastics and capital region sets.

Table 5. Validation figures for the ensemble and the 10 best performing neural networks after the exclusion of the points with scenarios outside of the fitting space. MAEs in %.

ID	v.mae	v.mae^a	v.res	v.mae^0	v.res^0	ns.mae	ns.mae^a	ns.res	ns.mae^0	ns.res^0	cr.mae	cr.mae^a	cr.res	cr.mae^0	cr.res^0
Ens.	0.091	0.087	−1.8	2.583	−9.8	0.067	0.065	3.1	2.292	−5.0	0.091	0.090	7.7	2.350	−0.4
NN1	0.084	0.081	−0.7	2.308	−6.2	0.096	0.093	13.0	2.184	7.5	0.137	0.134	19.4	3.011	13.9
NN2	0.092	0.088	0.6	2.458	−5.5	0.112	0.108	15.8	2.402	9.7	0.143	0.140	20.3	3.056	14.3
NN3	0.095	0.091	−0.4	3.100	−14.6	0.073	0.070	5.3	2.869	−8.8	0.093	0.092	7.7	2.743	−6.5
NN4	0.108	0.103	−5.5	3.868	−21.6	0.079	0.076	4.0	3.303	−12.2	0.110	0.108	10.5	2.743	−5.6
NN5	0.105	0.100	−0.1	2.665	−7.2	0.094	0.091	3.9	2.760	−3.2	0.101	0.099	6.3	2.659	−0.8
NN6	0.109	0.104	−0.6	2.705	−1.2	0.083	0.080	−5.6	2.483	−6.3	0.087	0.085	2.9	2.386	2.2
NN7	0.103	0.098	−7.0	3.182	−15.0	0.079	0.077	−2.3	2.989	−10.4	0.096	0.094	2.1	2.838	−5.9
NN8	0.102	0.098	−0.8	3.229	−14.4	0.083	0.080	2.5	3.312	−11.1	0.098	0.096	3.6	3.199	−10.0
NN9	0.119	0.114	−2.0	3.100	−7.9	0.095	0.091	0.5	3.044	−5.4	0.101	0.099	4.1	2.826	−1.8
NN10	0.126	0.120	0.0	3.161	7.0	0.091	0.088	3.8	3.034	10.8	0.117	0.115	10.8	3.870	17.7

Hence, overall our ensemble with $P = 10$ unifies the advantages of the rather NN1 alike and rather NN6 alike networks. Of course, this dichotomy shall only be regarded as being illustrative as a high goodness-of-fit of a proxy function depends on many dimensions in our actuarial application. To build a decent ensemble, it is crucial to set the number P of best networks it will be composed of neither too small nor too large. Thereby, P should be set sufficiently large to ensure enough variation

(NN1 versus NN6) and balance of opposing effects. At the same time, P should be set sufficiently small to prevent the calibration procedure from selecting unnecessarily poorly performing networks into the ensemble. Since the frequency of "well-suited" networks produced by a training session depends on the total number of trained networks (here: $N = 300$), P should be set in accordance with total number N.

4.1.6. Sensitivity Analysis and Interpretability

The question of interpretability of results of algorithms, in particular of machine learning methods, is currently a hot topic. It should be possible to tell the reasons of the decisions (in case of a classification method) or of the exact value (in case of, for example, a regression type prediction method) given by the numerical method. While, on the one hand, this topic arises from a public fear of being controlled by algorithms and artificial intelligence, it is, on the other hand, an argument often raised against the highly non-linear nature of neural networks. In particular, one has to understand that there are two fundamentally differing philosophical reasons behind the question of interpretability. While public fear is focused on algorithmic decisions in general, the non-linearity of neural networks is a mathematical concern.

Certainly, polynomials, as in the regression approach in Krah et al. (2018), are easier to understand than the repeated concatenation of non-linear and affine linear functions in an FNN, as given in Definition 2. However, in our application, this is only a formal advantage. To see this, note the proxy function given in Table A.3 in Krah et al. (2020).

Of course, a sensitivity analysis with regard to the different risk factors X_i can be performed. If we, for example, take risk factor X_8 (the shock on the market value of bonds) and set all the other risk factors to 0, we obtain a polynomial of degree 4 in X_8. However, the risk factor X_8 is included in 88 of the 225 terms of the polynomial. It is by no means clear how, for any practical purpose, one can interpret the implications of these 88 terms other than by evaluating the polynomial in interesting scenarios, like for instance in the capital region. However, this can also be done in the case of neural networks.

To illustrate this type of analysis, we show in Figures A3 and A4 one-dimensional plots that explain the behavior of the proxy function as a polynomial in X_1 (the movement of the risk-free interest rate) and X_8, written with an explicit formula, respectively. The evaluation of a neural network in one dimension—e.g., for the risk factors X_1 and X_8, respectively—is as easy as it is for a polynomial. One can see in the plot, with respect to X_1, that the 10 best derived neural networks, the ensemble network and the polynomial, are close inside of the fitting range (which is $[-3.34, 3.34]$ with the endpoints being the 0.1% and 99.9% quantiles of the univariate distribution) and that they start to differ outside of it. The plot with respect to the most influential risk factor X_8 demonstrates a slightly different pattern. Here, the range $[-0.19, 0.19]$ corresponds to the interval from the 0.1% to the 99.9% quantile of the univariate distribution but the fitting range is chosen to cover the wider range $[-0.30, 0.30]$ for the reasons stated in Section 2.1. While the neural networks and the polynomial behave very similarly on $[-0.19, 0.19]$ (as they do on $[-3.34, 3.34]$ with respect to X_1), they already start to drift apart beyond that interval, even though there is still some way to go before the endpoints of the fitting range are reached. At the lower boundary, even the different neural networks drift apart. This result demonstrates, impressively, how important it is to set the endpoints of the fitting space carefully.

To illustrate the behavior of the ensemble method with regard to the joint variation of the two variables X_1 and X_8, we have plotted the corresponding surface in Figure A5. Note, in particular, the twist of the surface when both variables are jointly negative. A different joint behavior between X_8 and X_9 (the shock on lapse rates) can be inferred from Figure A6. Here, it seems that X_8 and X_9 do not have strong interactions in the CFP model.

As a consequence of the above analyses and arguments, in our application, one can take on the view that with regard to interpretability the advantage of having a complicated polynomial as

in Krah et al. (2020) is not a big one compared to having a neural network that can be dealt with numerically with nearly the same amount of work.

5. Conclusions

In the third in our series of articles, we have moved beyond linear models in our attempts to build proxy functions for insurance risk capital calculations. In light of many accomplishments in various applications achieved with neural networks, we have decided to test neural networks as regression functions in our risk capital approximation task.

The results show that neural networks indeed perform very well overall and in particular in the task of capturing the changes of the underlying variables. We observe a remarkable improvement in approximation quality, in terms of the most important metric in the risk management applications, namely the SCR. We make the full use of two approaches which, thankfully, can be operated on neural networks:

- Quasi-random hyperparameter search;
- Building an ensemble of the best performing models.

These approaches achieve a very satisfactory result for the observed company.

In recent machine learning competitions, two methods stand out as the most successful ones: ensembles of neural networks and gradient boosting with decision trees. We have found that the former method indeed performs very well in our context. In a recent thesis by Geller (2020), the gradient boosting method was tested in a very similar setting to the one in our tests. Interestingly, even after performing an extensive hyperparameter optimization within the gradient boosting models, the resulting functions still performed worse than neural networks.

Moreover, we conclude from our analyses that the neural networks applied in the proposed LSMC-NN framework outperform all other methods we have tested in Krah et al. (2018, 2020). If it is confirmed that the gradient-boosted decision trees do not match the performance of the ensembles of neural networks, then—based on our tests—there is a clear winner in the contest for the best regression function for proxy modeling in life insurance risk management.

Finally, keeping in mind our recommendation for neural networks, we have also commented on the increasingly important field of interpretability of the results obtained with machine learning methods. The proxy models are used for the determination of the regulatory risk capital and hence have real impact on life insurers. Even more, the risk exposure and the solvency ratio are important indicators for investors and capital markets. Thus, understanding the implications of the changes in risk factors for the SCR is essential.

We believe that our arguments and examples in Section 4.1.6 have shown that, in our application of calculating the SCR by the LSMC approach, analyzing a polynomial with numerous terms (including mixed monomials) does not fundamentally differ from analyzing a suitable single neural network or an ensemble network. On top of this, neural networks have an additional advantage compared to polynomials: While polynomials of higher degrees tend to explode beyond the fitting range, neural networks do not necessarily show this tendency.

With the fast evolution of applications of neural networks, their various different types and also the huge range of freely available software, we believe that this article is mostly a starting point for encouraging more work on the application of neural networks in SCR calculation.

Author Contributions: Conceptualization, A.-S.K., Z.N. and R.K.; Formal analysis, A.-S.K., Z.N. and R.K.; Investigation, A.-S.K. and Z.N.; Methodology, A.-S.K., Z.N. and R.K.; Project administration, A.-S.K., Z.N. and R.K.; Resources, Z.N.; Software, Z.N.; Supervision, R.K.; Validation, A.-S.K. and R.K.; Visualization, A.-S.K.; Writing—original draft, Z.N. and R.K.; Writing—review and editing, A.-S.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

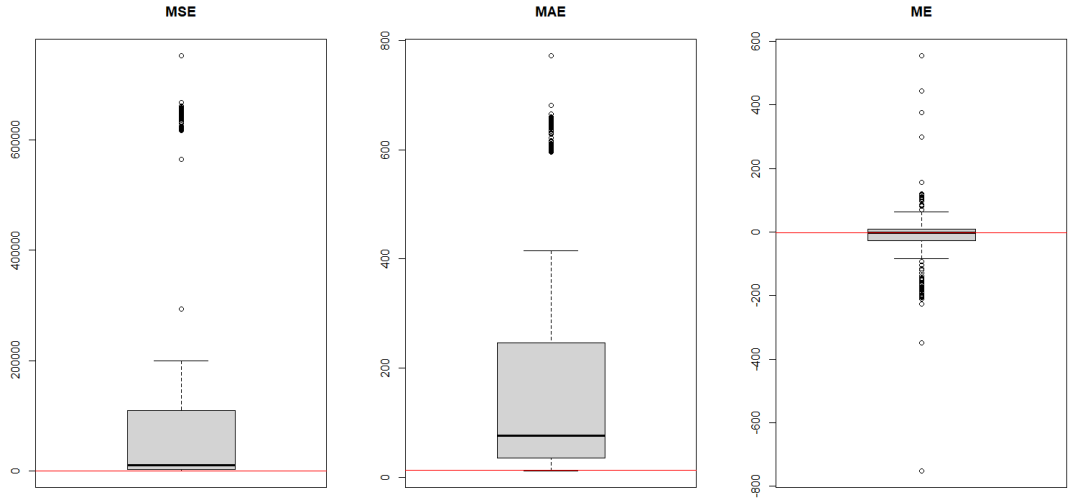

Figure A1. Box plots displaying the MSE, MAE and ME of 300 fully trained neural networks using different hyperparameters. MSE, MAE and ME evaluated with respect to the (Sobol) validation set. Figures of the ensemble of 10 best performing neural networks added in red.

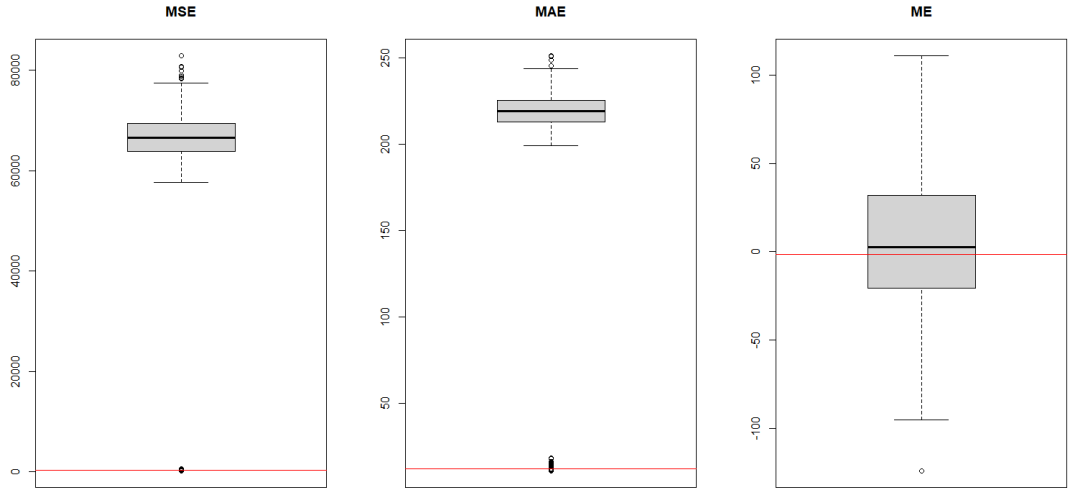

Figure A2. Box plots displaying the MSE, MAE and ME of 300 fully trained neural networks using the same hyperparameters. MSE, MAE and ME evaluated with respect to the (Sobol) validation set. Figures of the ensemble of 10 best performing neural networks added in red.

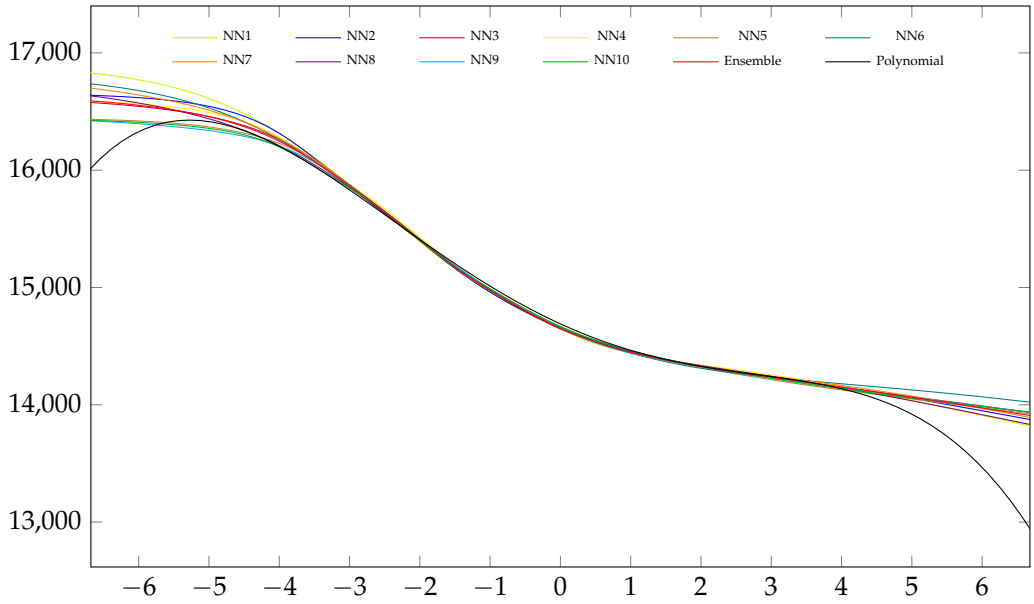

Figure A3. One-dimensional curves of the 10 best performing neural networks, their ensemble and the polynomial with respect to X_1.

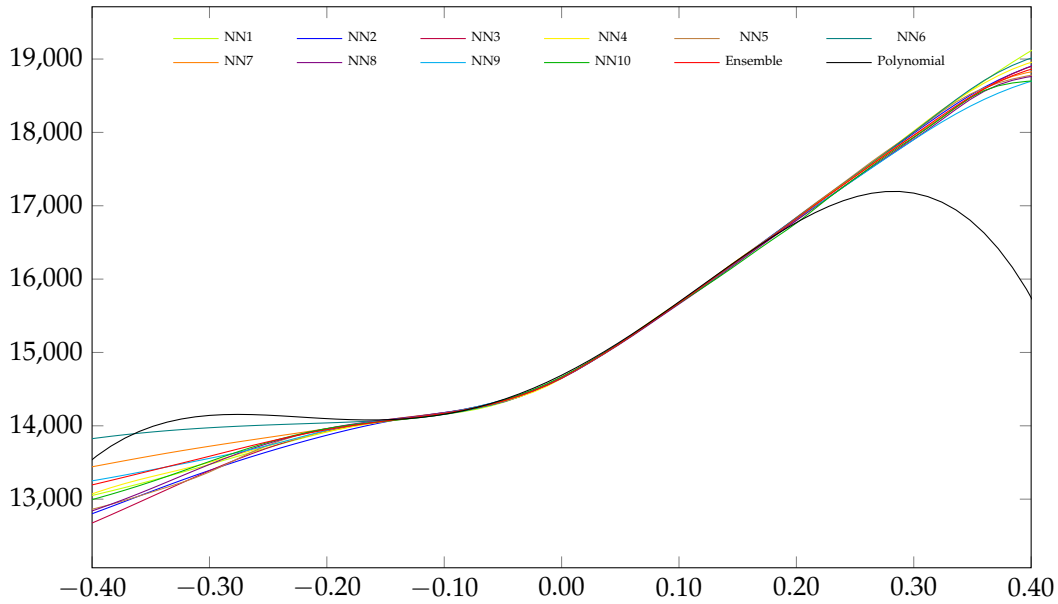

Figure A4. One-dimensional curves of the 10 best performing neural networks, their ensemble and the polynomial with respect to X_8.

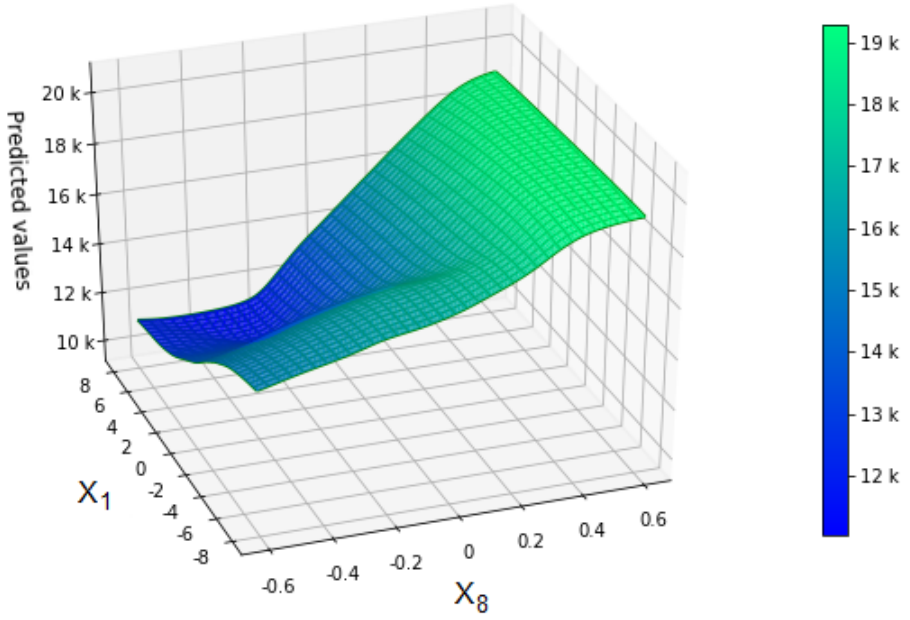

Figure A5. Two-dimensional curve of the ensemble of 10 best performing neural networks with respect to X_1 and X_8.

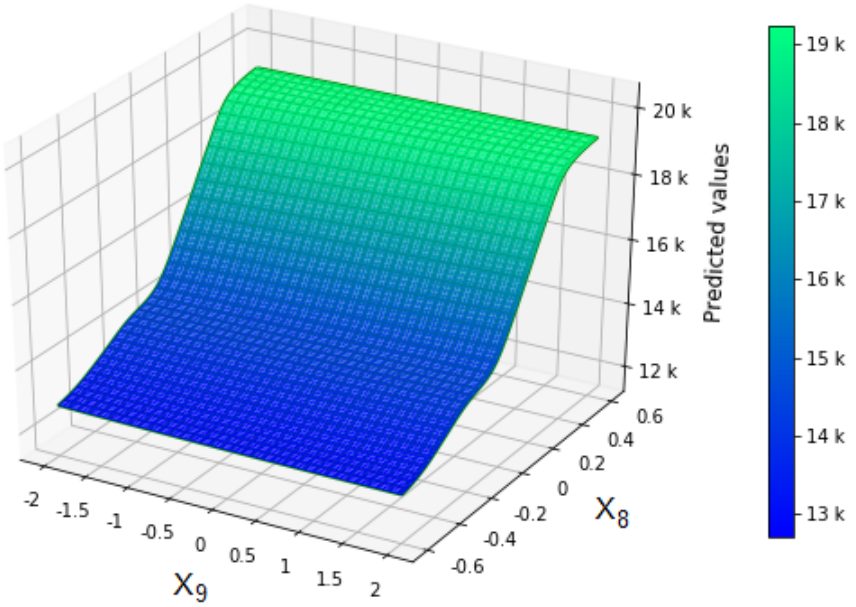

Figure A6. Two-dimensional curve of the ensemble of 10 best performing neural networks with respect to X_8 and X_9.

References

Akaike, Hirotogu. 1973. Information theory and an extension of the maximum likelihood principle. Paper presented at 2nd International Symposium on Information Theory, Tsahkadsor, Armenia, September 2–8; pp. 267–81.

Bauer, Daniel, and Hongjun Ha. 2015. A least-squares Monte Carlo approach to the calculation of capital requirements. Paper presented at World Risk and Insurance Economics Congress, Munich, Germany, August 2–6.

Bettels, Christian, Johannes Fabrega, and Christian Weiß. 2014. Anwendung von Least Squares Monte Carlo (LSMC) im Solvency-II-Kontext—Teil 1. *Der Aktuar* 2: 85–91.

Born, Rudolf. 2018. Künstliche Neuronale Netze im Risikomanagement. Master's thesis, Universität zu Köln, Köln, Germany.

Castellani, Gilberto, Ugo Fiore, Zelda Marino, Luca Passalacqua, Francesca Perla, Salvatore Scognamiglio, and Paolo Zanetti. 2018. An Investigation of Machine Learning Approaches in the Solvency II Valuation Framework. Available online: http://dx.doi.org/10.2139/ssrn.3303296 (accessed on 14 August 2019).

Frerix, S. P. H. M. 2018. Efficient Estimation of the Solvency Capital Requirement Using Neural Networks. Master's thesis, University of Amsterdam, Amsterdam, The Netherlands.

Friedman, Jerome H. 1991. Multivariate adaptive regression splines (with discussion). *The Annals of Statistics* 19: 1–141. [CrossRef]

Geller, Ina. 2020. Gradient Boosting with Regression Trees for Solvency Capital Requirements. Master's thesis, Universität zu Köln, Köln, Germany.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. Available online: http://www.deeplearningbook.org (accessed on 16 September 2020).

Hastie, Trevor, and Robert Tibshirani. 1986. Generalized additive models. *Statistical Science* 1: 297–318. [CrossRef]

Hastie, Trevor, and Robert Tibshirani. 1990. *Generalized Additive Models*. London: Chapman & Hall.

Hejazi, Seyed A., and Kenneth R. Jackson. 2017. Efficient valuation of scr via a neural network approach. *Journal of Computational and Applied Mathematics* 313: 427–39. [CrossRef]

Hocking, R. R. 1976. The analysis and selection of variables in linear regression. *Biometrics* 32: 1–49. [CrossRef]

Keras team. 2020. *Keras: Deep Learning API*. Available online: https://github.com/keras-team/keras (accessed on 18 September 2020).

Kiermayer, Mark, and Christian Weiß. 2019. Grouping of Contracts in Insurance Using Neural Networks. *arXiv* arXiv:1912.09964.

Kopczyk, Dawid. 2018. Proxy Modeling in Life Insurance Companies with the Use of Machine Learning Algorithms. Working Paper. Available online: http://dx.doi.org/10.2139/ssrn.3396481 (accessed on 29 July 2019).

Krah, Anne-Sophie, Zoran Nikolić, and Ralf Korn. 2018. A least-squares Monte Carlo framework in proxy modeling of life insurance companies. *Risks* 6: 62.

Krah, Anne-Sophie, Zoran Nikolić, and Ralf Korn. 2020. Machine learning in least-squares Monte Carlo proxy modeling of life insurance companies. *Risks* 8: 21. [CrossRef]

Nadaraya, Elizbar A. 1964. On estimating regression. *Theory of Probability and Its Applications* 9: 141–42. [CrossRef]

Nelder, John A., and Robert W. M. Wedderburn. 1972. Generalized linear models. *Journal of the Royal Statistical Society, Series A* 135: 370–84. [CrossRef]

Schelthoff, Tom. 2019. Machine Learning Methods as Alternatives to the Least Squares Monte Carlo Model for Calculating the Solvency Capital Requirement of Life and Health Insurance Companies. Master's thesis, Universität zu Köln, Köln, Germany.

TensorFlow team. 2020. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Available online: https://www.tensorflow.org (accessed on 17 September 2020).

Teuguia, Oberlain N., Jiaen Ren, and Frédéric Planchet. 2014. *Internal Model in Life Insurance: Application of Least Squares Monte Carlo in Risk Assessment*. Technical Report. Lyon, France: Laboratoire de Sciences Actuarielle et Financière.

Watson, Geoffrey S. 1964. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A* 26: 359–72.

Wiese, Magnus, Robert Knobloch, Ralf Korn, and Peter Kretschmer. 2020. Quant gans: Deep generation of financial time series. *Quantitative Finance* 20: 1419–40. [CrossRef]