An individual model for claims reserving based on Bayesian neural networks

Gabriele Pittarello [1]
Gian Paolo Clemente [2]
Diego Zappa [2]

[1]La Sapienza, Università di Roma
[2]Università Cattolica del Sacro Cuore

September 27, 2022

Abstract

Following the approach in Wüthrich (2018), we propose a new approach for individual claims reserving and we show how individual development factors can be modelled as the prediction target of a system of Bayesian neural networks. This approach allows to take into account the complete information on policyholders available to the insurance company and to provide a new application of Bayesian neural networks to obtain a stochastic claims reserve. This contribution will show a case study that compares the individual chain ladder approach and the Bayesian neural networks model.

Individual claims reserving, Chain ladder, Bayesian neural networks, Machine learning

1 Introduction

While life insurance mainly concerns contracts with large duration and immediate settlement, non-life insurance contracts have short duration and delayed resolution. For this reason, it is essential to fairly quantify liabilities providing estimates that are as close as possible to the actual future realization of the company outflows and modelling the stochastic process that determines their future development. The computational aspects of this task are strictly linked to the accounting criteria that determines their fairness, so to the timing of the insurance policy. Non-life insurance contracts cover the policyholder from losses arising from predetermined events that might happen in a period of time that is called coverage period. In case some of these predetermined events occurs, we call accident date the year in which the event was triggered. The payment date is conversely the date in

which the insurance company paid the policyholder. Before the settlement date in which the claim is closed, there might be multiple payments. Notionally, we group insurance claims as follows:

- *Reported But Not Setteled claims (RBNS)* as those claims for which one or more payments were recorded. Further claims develop in time, larger the amounts paid until closure are expected to be.
- *Reported But Not Paid claims (RBNP)* as those claims that were already reported at measurement date but no payment is yet occurred.
- *Incurred But Not Reported claims (IBNR)* as those claims that already occurred at measurement date but the underwriter is unaware of.

In standard claims reserving, practitioners aggregate individual policyholders data into specific data structures called triangles to model the underlying process with some well known reserving technique. In individual claims reserving, data are disentangled and treated as supervised learning problem. The peculiar form of a reserving data set forces the practitioner to deal with truncated time-series. A detailed description is provided in the following sections.

Several authors started surveying the possibilities of machine learning models as a solution to improve claims reserving. In [12], the author investigates how actuarial science may adapt and evolve in the coming years to incorporate deep learning models and provides background on machine learning and deep learning for practitioners. The paper surveys emerging applications of artificial intelligence in actuarial science, including claims reserving.

The work in [13] surveys recent developments in machine learning models against micro-level models for loss reserving. The two families are compared and assessed in terms of potential future development. The discussion is focused on their relative merits and the factors governing the choice between them and the standard actuarial models.

The authors of [3] use a neural networks approach to simulate individual claims reporting and cashflow patterns that can be deployed to support future back-testing of reserving methods. The simulation algorithm is parameterised using individual claims records and it explicitly considers the possibility of recoveries and reopened claims with a full history of 12 years of development. The conclusion is that the neural network algorithm creates simulated datasets which exhibit similar behavior to real data. In [4], the authors embed classical actuarial regression models into neural network architectures. The models in [4] are efficiently fitted and bootstrap methods for prediction uncertainty are explored. The starting point of the neural network calibration is the classical actuarial model. The theoretical framework in [4] is applied to a cross-classified over-dispersed Poisson model. Afterwards the benefit of improving the model via neural networks is demonstrated. Similarly, the actuarial technique in [3] takes into account both claim counts and claim amounts with two separate overdispersed Poisson models. The result is a boosting machine that allows for mutual learning of claim counts and claim amounts beyond the two individual (overdispersed) Poisson models.

In [11] the author uses neural networks to model both risk price and ultimate claims on undeveloped policy and claims level data. Lastly, it concludes granular claims reserving can outperform portfolio approaches on larger datasets. It is worth mentioning that this project discusses that choice of initialization and optimizer can be important to model fit.

In [14] the authors develop regression models and postulate distributions which can be used in practice to describe the joint development process of individual claim payments and claim incurred. The RBNS modelling is regarded with information on risk factors and allocations. It is provided a joint distribution of paid and incurred claims.

The work in [2] demonstrates that a system of neural networks provides more improvement over chain-ladder for long tail lines of business or if the line of business is not homogeneous. The authors also observed neural networks are better under very restrictive conditions, such as using transformed pattern which requires non-zero paid values that may not apply to practical reserving analysis.

The model in [7] is an individual claims forecasting framework that relies on bayesian mixture density networks for claims analytics tasks. By incorporating claims information from different data sources, it allows to produce multi-period cash flow forecasts and different scenarios of future payment patterns generation. The modeling framework is tested on a case study.

The model proposed in [15] embeds individual features into the traditional Chain-Ladder model, while IBNR claims are modelled on aggregate. RBNS claims are predicted as the prediction target of multilayer perceptron. On the other hand, while stochastic claims reserving assess both process uncertainty and model uncertainty, multilayer perceptrons do not provide an estimate of the variability which is necessary for capital requirement purposes. The novel of this work is to provide the first bayesian neural networks implementation of an individual one-period model based on the individual Chain-Ladder that allows to assess the individual reserves.

The paper is structured as follows: in section 2, we will introduce the triangular data structure and the deterministic chain ladder model. In section 3, we will provide the general framework of our individual claims reserving model. In section 4 we will define the individual chain ladder model as in [15] and our stochastic individual chain ladder model. In section 5, we show the simulated data set that we simulate from [5] to implement the individual chain ladder. In the last section, we will show the mean predictions of the Bayesian neural networks model. In the Appendix A, additional results are provided. In the Appendix B, we summarise main computational aspects.

2 Claims reserving

Within this section we introduce the deterministic chain ladder model, which is the foundation of this work, see [8]. In order to do so, we show the main notation of the aggregate reserving framework as the starting point for individual reserving models. Often, in non-life insurance individual data are represented as data structures called run-off triangles:

$$S^{(n)} = \{C_{i,j} : i + j \leq T\},$$

where $C_{i,j}$ represents the cumulative sum of the individual payments of the insurance company for accident period i at development period j, with $i = 0, \ldots, T$, $j = 0, \ldots, T$, where T is the triangle dimension or evaluation date. We show a non-life insurance triangle in Figure 3.

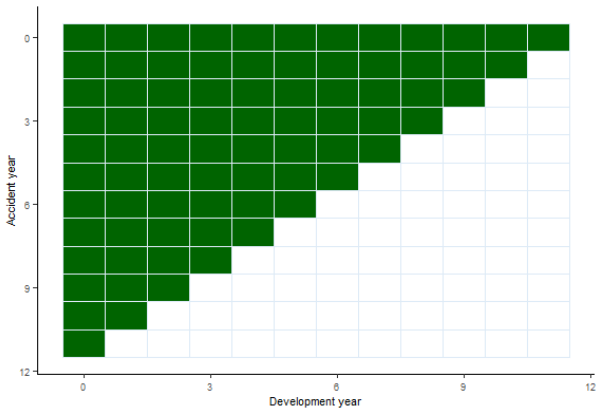

Figure 1: Run-off triangle.

The chain ladder algorithm describes the development of cumulative claims as:

$$C_{i,j} = C_{i;j-1} \cdot f_{j-1}$$

where $f_0, \ldots, f_{T-1}$ are the so-called development factors. By means of a cumulative product it is possible to obtain the ultimate cost for each accident period as:

$$C_{i,T} = C_{i,T-i} \prod_{j=T-i}^{T-1} f_j$$

For each accident year the claims reserve can be computed as:

$$R_i = C_{i,T} - C_{i,T-i}$$

The claims reserve is given as the sum of the reserves for each accident year.

3 Modelling framework

In individual claims reserving, each individual $k = 1, \ldots, N$ is described by the vector of features $\mathbf{x_k} \in \mathcal{R}^p$ and the time series of individual cumulative payments $\mathbf{C_{i,k}} =$

$\{C_{i,1,k}, \ldots, C_{i,T,k}\}$. In this project we model the chain-ladder development factors in a similar fashion as it was previously done in [15].

The chain-ladder fundamental relation in the individual framework then becomes:

$$ {}_kC_{i,j} = {}_kf_{j-1} \cdot {}_kC_{i,j-1} \tag{1}$$

Since our model is a one-period model, we can construct the individual development factors in the upper triangle and fit a Bayesian neural network for each development period. This is because at the evaluation date T we do not observe the full data. We then exploit Bayesian neural networks together with the chain-ladder algorithm to forecast the value of the reserve. Indeed, for each development period we will observe the training set:

$$ D_j = \left\{(x_1, {}_1f_1), \ldots, (x_{N_j}, {}_{Nj}f_j)\right\} \tag{2}$$

where N_j indicates the number of individuals in the development period j for which the information on the upper triangle is available.

4 Model framework

Bayesian neural networks are a flexible structure that allows practitioners to account for both model uncertainty and process uncertainty. We introduce baysian neural networks for individual claims reserving in three steps. We first introduce the notation we use for multilayer perceptrons in the individual chain-ladder framework. In a second place, we explain how is it possible to account for process uncertainty within our estimates by properly changing the loss function. Lastly, we show the novelty of variational inference applied to neural networks weights to properly account for model uncertainty: multilayer perceptrons only provide a point estimate of model weights. Within our framework, model weights become a distribution.

4.1 Individual chain-ladder

In the individual chain-ladder approach in [15], development factors are modeled as the prediction target of a neural network with one hidden layer with q neurons. Consider the prediction problem for the individual development factor in the duration j. The regression target ${}_k\,\hat{f}_j$, for the $k-th$ individual with x_k features is expressed as:

$$ {}_k\,\hat{f}_j = \exp\left(\mathbf{z}^T\mathbf{w}^{(2)}\right) \tag{3}$$

The hidden structure is described with the vector of q neurons:

$$ \mathbf{z} = (1, \ldots, z_q)\,, $$

where each neuron is expressed as $z_l = \tanh\left(\mathbf{x}^T\mathbf{w}_l^{(1)}\right)$.

The connection from the $p-$dimensional feature input $\mathbf{x} = (1, \ldots, x_p)$ to the neurons in the hidden layer is specified with the weights $\mathbf{w}_l^{(1)} = \left(w_{0l}^{(1)}, w_{1l}^{(1)}, \ldots, w_{pl}^{(1)}\right)'$.

The output is connected to the neurons via the vector of weights $\mathbf{w}^{(2)} = \left(w_0^{(2)}, w_1^{(2)}, \ldots, w_q^{(2)}\right)'$.

In the optimization phase, the optimal set of parameter is obtained by minimizing a loss function $L(\mathbf{w}^{(1)}, \mathbf{w}^{(2)})$. In [15] the authors adopt a weighted version of the mean squared error, see the paper for more details. During the model training phase the neural networks parameters are optimized on the data. Several algorithms were introduced in the literature, see [6].

4.1.1 Zero claims features

A problem arises for those individuals with a zero-payment on the diagonal. The authors in [15] propose a separate aggregate model for those claimants. In a similar fashion we propose a solution that acts on the individual level. We are now able to define a new set:

$$D_j^* = \left\{(x_1, 0), \ldots, (x_{N_j^*}, 0)\right\} \tag{4}$$

where N_j^* represents the number of individuals with no payment record in development year j. We now obtain:

$$C_{i,j}^* = \sum_{k \in D_j^*, i=i} {}_k\, C_{i,j-1} \tag{5}$$

We can then compute the proportions λ_i with $i = 0, \ldots, T-1$:

$$\lambda_i = \frac{\sum_{s=1}^{i-1} C_{s,T-i+1}^*}{\sum_{s=1}^{i-1} C_{s,T-i}} \tag{6}$$

We then split the $C_{i,T-i}$ amount among those claimants with zero payment on the diagonal and treat them in the Bayesian neural network model.

4.2 Bayesian neural networks

In the following section, we explain how to derive the Bayesian neural networks model that we use in this paper from the multilayer perceptron. In a first place, we choose a variational inference approach to quantify uncertainty over the model weights. Indeed, the **Bayes by Backprop** algorithm presented in [1], will be adapted to our framework. Lastly, we will state a probabilistic assumption to model development factors in a stochastic framework. In order to provide a clear explanation, we consider the connection between the input j and the node l on the first hidden layer for the duration j.

4.2.1 Weights uncertainty

Define $P(w_{jl}^{(1)}|D_j)$ as the posterior distribution of the weights given the data D_j. The exact computation of the model weights via the Bayes rule is possible but in practice it is intractable, [10].
We use variational inference to approximate the posterior distribution to the known distribution form $g(w_{jl}^{(1)}|\theta)$, where θ represents the distribution parameters. Within our framework we assume:

$$g(w_{jl}^{(1)}) \sim N(\mu, \sigma), \;\; \theta = (\mu, \sigma).$$

To approximate, we want the Kullback-Leibler distance distance between the weights posterior and the known functional form $g(w_{jl}^{(1)}|\theta)$ to be as small as possible.

The Kullback-Leibler distance between the posterior distribution $P(w_{jl}^{(1)}|D_j)$ and $g(w_{jl}|\theta)$ is defined as:

$$D_{KL}(g(w_{jl}^{(1)}|\theta)||P(w_{jl}^{(1)}|D_j)) = \int g(w_{jl}^{(1)}|\theta) \log \left(\frac{g(w_{jl}^{(1)}|\theta)}{P(w_{jl}^{(1)}|D_j))} \right) \mathrm{d}w_{jl}^{(1)} \tag{7}$$

During the optimization phase, we want to minimize the Kullback-Leibler distance between the posterior distribution and $g(w_{jl}^{(1)}|\theta)$ in θ. This leads us to the following optimization problem:

$$\arg\min_{\theta} D_{KL}(g(w_{jl}^{(1)}|\theta)||P(w_{jl}^{(1)}|D_j))$$
$$\arg\min_{\theta} \int g(w_{jl}^{(1)}|\theta) \log \left(\frac{g(w_{jl}^{(1)}|\theta)}{P(w_{jl}^{(1)}|D_j))} \right) \mathrm{d}w_{jl}^{(1)}$$

The integral in 7 be written as:

$$\arg\min_{\theta} \log P(D_j) + D_{KL}(g(w_{jl}^{(1)}|\theta)||P(w_{jl}^{(1)})) - \mathbb{E}_{g(w_{jl}^{(1)}|\theta)}(\log P(D_j|w_{jl}^{(1)}))$$

We call $P(w_{jl}^{(1)})$ the weights prior. Since $\log P(D_j)$ is constant, the previous trivially leads to the minimization of the following quantity:

$$\arg\min_{\theta} D_{KL}(g(w_{jl}^{(1)}|\theta)||P(w_{jl}^{(1)})) - \mathbb{E}_{g(w_{jl}^{(1)}|\theta)}(\log P(D_j|w_{jl}^{(1)})) \tag{8}$$

Two different terms with a clear mathematical interpretation appear in equation 8.

- $D_{KL}(g(w_{jl}^{(1)}|\theta)||P(w_{jl}^{(1)}))$ term is called *complexity cost*: it depends on θ and the prior $P(w_{jl}^{(1)})$.

- Conversely, the expectation $\mathbb{E}_{g(w_{jl}^{(1)}|\theta)}(\log P(D_j|w_{jl}^{(1)}))$ is called the *likelihood cost*, as it depends on θ and the data only.

The direct minimization of the cost function in 8 is again unfeasible. Following the novel in [1], we implement the **Bayes by Backprop** algorithm that is commonly used for variational inference on the weights. Equation 8 can be rewritten as:

$$\arg\min_{\theta} \mathbb{E}_{g(w_{jl}^{(1)}|\theta)}(\log g(w_{jl}^{(1)}|\theta) - \log P(D_j|w_{jl}^{(1)}) - \log P(w_{jl}^{(1)})).$$

Define $t(w_{jl}^{(1)}, \theta) = \log g(w_{jl}^{(1)}|\theta) - \log P(D_j|w_{jl}^{(1)}) - \log P(w_{jl}^{(1)})$ and a learning rate η. We adopt the **Bayes by Backprop** algorithm for weights learning handled in [9]:

1: Sample $\epsilon_i \sim N(0, 1)$.
2: Let $w_i = \mu + \sigma\epsilon_i$.
3: Calculate the gradients with respect to the parameters μ and the σ.

$$\Delta_\mu = \frac{\partial t(w_{jl}^{(1)}, \theta)}{\partial w_i} + \frac{\partial t(w_{jl}^{(1)}, \theta)}{\partial \mu} \qquad \Delta_\sigma = \frac{\partial t(w_{jl}^{(1)}, \theta)}{\partial w_{jl}^{(1)}}\sigma + \frac{\partial t(w_{jl}^{(1)}, \theta)}{\partial \sigma}$$

4: The gradients provided an updating criterion:

$$\mu \leftarrow \mu - \eta \cdot \Delta_\mu$$

$$\sigma \leftarrow \sigma - \eta \cdot \Delta_\sigma$$

4.2.2 Process uncertainty

In order to model our development factors in a probabilistic framework, we assume the development factors to be normal distributed. This may not seem straightforward but we impose it so that we are able to model recoveries within our payments history.

$$_k f_j \sim N\left(_k \mu_j, _k \overset{2}{\sigma}_j\right) \tag{9}$$

The minimization problem is now reduced to the minimization of a likelihood function. This is the second structural change from the neural network architecture we introduced in the previous sections.

5 Data

The data were simulated with the algorithm in [5]. It is a data set with 5003216 observations, 7 features over 12 development years. We display in Table 1 the features available for each individual that will be adopted in this paper.

Feature	**Description**
ClNr	policy identifier
LoB	Line of business, categorical with 4 levels
cc	claim code, categorical with 53 levels
AY	year of claim accident, integer with values from 1994 to 2005
AQ	year quarter of the claim accident, integer with values from 1 to 4
inj_part	year quarter of the claim accident, categorical with 99 levels

Table 1: Individual observation features

The available individual information allows us to visualize the information for the individuals by line of business. In Figure 2 we show the average claim severity for the individuals.

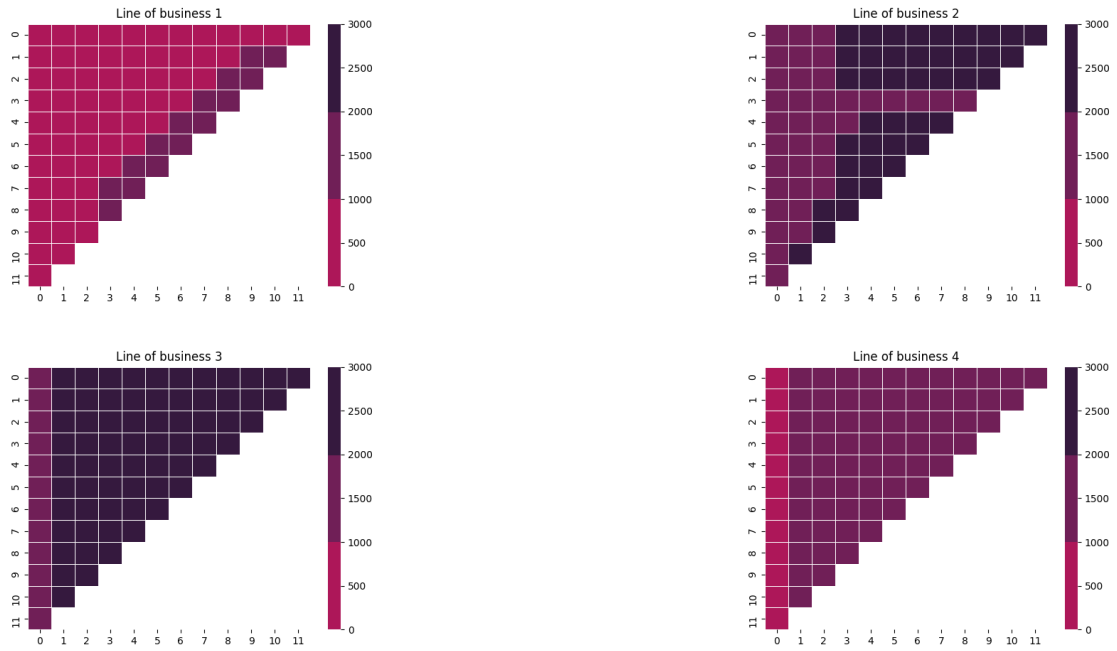

Figure 2: Heat map of the individual average severity for each accident year displayed by line of business.

6 Data application

6.1 Cross-validation

In order to fit the system of Bayesian neural networks on the data, the most appropriate architecture requires to be selected on the data. Working with big data may require a considerable amount of time for a considerable set of combinations of possible hyperparameter choices. The hyper-parameter choices were tested in terms of mean squared error of prediction, mean absolute error and prediction accuracy on the validation data. See 2 for the hyper-parameters sets tested in this phase. In order to select the correct architecture the hyperparameter sets where compared by using the Tensorboard powered by Tensorflow, see [9]. Using the Tensorboard allows to select the most appropriate parameters combination in a compact way.

Hyper-parameter	**Set**
Number of units, hidden layers	$16, 20, 32$
Number of hidden layers	$1, 2$
Optimizer	adam, RMSprop
Activation functions	relu,tanh, exponential,linear
Learning rate	$0.001, 0.01$

Table 2: Hyper-parameter selected during CV.

6.2 Claims reserving with Bayesian neural networks

Within this section we implement a case study that compares the mean results obtained with our Bayesian neural networks algorithm and the results of the individual chain ladder in [15]. A similar analysis by line of business can be found in the appendix. We prove the predictions consistency by providing similar results to those obtained from the model in [15]. As long as the approach in [15] does not use an individual model for claims with zero diagonal payments we limit our case study to those claims with positive payment at the evaluation date. Table 3 considers the error margin with respect to the real value of the reserve. In general, the Bayesian neural networks model seems to better perform in terms of final predicted amount while the individual chain-Ladder model seem to provide a smaller mean absolute error. For sake of interpretability, we consider the $+0.1\%$ relative error on the reserve: it means that the Bayesian Neural Network model overestimated the true amount of the reserve by $+0.1\%$. The mean absolute error was again computed with respect to the real data from the simulation engine.

Model	Relative error on the ultimate cost	Relative error on the reserve	Mean Absolute Error (MAE)
Bayesian Neural Networks	+0.1%	+1.2%	364.516
Chain-Ladder Neural Networks	+1.0%	+9.1%	294.698

Table 3: In the first two columns we display the relative errors on the reserve estimates according to the different approaches. The last columns shows the mean absolute error. The results were rounded to the first three decimal points.

In Table 4 some summary statistics on the predictions for the full data set are displayed. The Bayesian neural network model describes the distribution better than the individual chain ladder model. The objective of Table 4 is to provide sketch of the payments distribution according to the different models in order to understand to what extent they catch the process behavior.

Data	Mean individual ultimate cost	0.05 ultimate cost quantile	median	0.995 ultimate cost quantile
Bayesian Neural Networks predictions	2262.418	55.053	460.229	53301.001
Chain-Ladder Neural Network predictions	2281.342	69.059	470.192	53768.506
Real data	2259.567	64.0	425.0	56367.0

Table 4: Mean individual predictions according to different methodologies. The results were rounded to the first three decimal points.

In Figure 3 the ultimate cost predictions are compared with the ultimate cost actual individual values. The results seem consistent as the predictions lay in the points cloud.

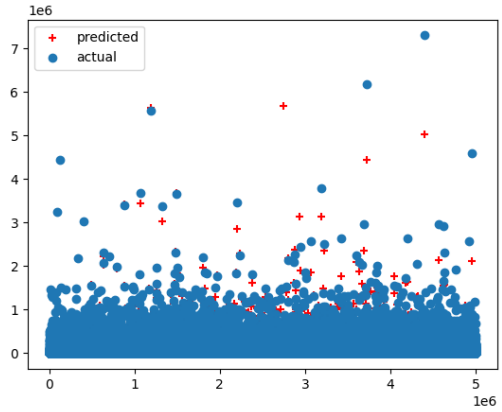

Figure 3: Graphical representation of the ultimate cost predictions for the ICRBNN.

Table 5 shows the prediction errors in terms of relative amount between Bayesian Neural Networks and Individual Chain-Ladder. While the Bayesian Neural Network model outperforms on aggregate and on the older accident years the Individual Chain-Ladder provides better results on the most recent accident years. For sake of interpretability, consider the individual chain ladder 0.108 error in accident year 1: it means that the individual chain-ladder model overestimates the accident year 1 amount by 10.8%.

Model	ay0	ay1	ay2	ay3	ay4	ay5	ay6	ay7	ay8	ay9	ay10	
Bayesian Neural Networks predictions	0.0	0.004	−0.07	0.616	−0.325	−0.21	0.064	0.046	−0.111	−0.132	0.137	0.634
Chain-Ladder Neural Network predictions	0.0	0.108	0.43	−1.458	0.368	0.382	0.237	0.132	−0.007	−0.008	0.084	0.323

Table 5: Relative prediction errors according to the different approaches for each accident year. The results were rounded to the first three decimal points.

7 Conclusions

This paper shows a novel application of an existing algorithm to the claims reserving framework. Within this case study we showed that this approach was able to replicate the individual chain ladder in [15]. Further research will regard the development of an extensive analysis to allow practitioners to assess the prediction uncertainty. By doing so we will be able to access process and model uncertainty of the claims reserve. While the process uncertainty is coming from the distributive assumption on the output layer, the model uncertainty is arising from the distribution weights.

References

[1] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015.

[2] Bor Harej Salma Jamal Roman Gächter et. al. Individual claim development with machine learning. *ASTIN working party report*, 2017.

[3] Andrea Gabrielli. A neural network boosted double over-dispersed poisson claims reserving model. *Scandinavian Actuarial Journal*, 2019.

[4] Andrea Gabrielli, Ronald Richman, and Mario V. Wüthrich. Neural network embedding of the over-dispersed poisson reserving model. 2018.

[5] Andrea Gabrielli and Mario V. Wüthrich. An individual claims history simulation machine. 2018.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[7] Kevin Kuo. Deeptriangle: A deep learning approach to loss reserving. *Risks*, 2019.

[8] Thomas Mack. Distribution-free calculation of the standard error of chain ladder reserve estimates. *ASTIN Bulletin: The Journal of the IAA*, 23(2):213–225, 1993.

[9] Paul Barham Martín Abadi, Ashish Agarwal et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[10] Beate Sick Oliver Dürr and Elvis Murina. *Probabilistic Deep Learning with Python, Keras and TensorFlow Probability*. Manning, 2020.

[11] Jacky H. L. Poon. Penalising unexplainability in neural networks for predicting payments per claim incurred. *Risks*, 2019.

[12] Ronald Richman. Ai in actuarial science. *Annals of Actuarial Science*, 2018.

[13] Greg Taylor. Loss reserving models: Granular and machine learning forms. *Risks*, 2019.

[14] Mario Wüthrich and Lukasz Delong. Regression models for the joint development of individual payments and claim incurred. 2020.

[15] Mario V. Wüthrich. Neural networks applied to chain–ladder reserving. *European Actuarial Journal*, 8:407–436, 2018.

A Case study by lines of business

The following appendix compares the two different machine learning models performances in the different lines of business. In general, while the individual chain Ladder is better performing in the line of business 4, the bayesian neural networks model seem to be more consistent on the other lines. The results were rounded to the first three decimal points.

Model	Relative error on the ultimate cost	Relative error on the reserve	Mean Absolute Error (MAE)
Bayesian Neural Networks	+0.05%	+0.411%	220.617
Chain-Ladder Neural Networks	+0.377%	+3.11%	161.549

Table 6: Line of business 1, relative prediction errors and MAE.

Data	Mean individual ultimate cost	0.05 ultimate cost quantile	median	0.995 ultimate cost quantile
Bayesian Neural Networks predictions	1140.787	41.63	321.266	25555.908
Chain-Ladder Neural Network predictions	1144.515	54	327.961	25599.786
Real data	1140.219	51	300	27637.61

Table 7: Line of business 1, process description capabilities.

Model	ay0	ay1	ay2	ay3	ay4	ay5	ay6	ay7	ay8	ay9	ay10	
Bayesian Neural Networks predictions	0.0	0.0	−0.13	0.435	0.131	−0.252	−0.009	−0.054	−0.039	−0.055	0.039	0.258
Chain-Ladder Neural Network predictions	0.0	0.149	0.54	−1.28	−0.257	0.234	0.056	−0.026	−0.038	0.01	−0.006	0.048

Table 8: Line of business 1, prediction errors in the different accident years.

Model	Relative error on the ultimate cost	Relative error on the reserve	Mean Absolute Error (MAE)
Bayesian Neural Networks	+0.824%	+11.094%	657.362
Chain-Ladder Neural Networks	+1.549%	+20.847%	574.839

Table 9: Line of business 2, relative prediction errors and MAE.

Data	Mean individual ultimate cost	0.05 ultimate cost quantile	median	0.995 ultimate cost quantile
Bayesian Neural Networks predictions	5342.369	199.0	1393.718	140777.479
Chain-Ladder Neural Network predictions	5380.763	219.61	1407.292	142976.67
Real data	5298.695	209.0	1323.0	144487.32

Table 10: Line of business 2, process description capabilities.

Model	ay0	ay1	ay2	ay3	ay4	ay5	ay6	ay7	ay8	ay9	ay10	
Bayesian Neural Networks predictions	0.0	0.042	−0.432	0.001	0.027	0.795	0.283	0.107	0.022	−0.07	0.138	−5.49
Chain-Ladder Neural Network predictions	0.0	0.237	−1.828	−0.169	−0.858	4.092	0.703	0.21	0.074	0.045	0.061	−2.875

Table 11: Line of business 2, prediction errors in the different accident years.

Model	Relative error on the ultimate cost	Relative error on the reserve	Mean Absolute Error (MAE)
Bayesian Neural Networks	+0.923%	+11.905%	816.381
Chain-Ladder Neural Networks	+1.856%	+23.952%	733.804

Table 12: Line of business 3, relative prediction errors and MAE.

Data	Mean individual ultimate cost	0.05 ultimate cost quantile	median	0.995 ultimate cost quantile
Bayesian Neural Networks predictions	6500.793	242.839	1671.583	164073.411
Chain-Ladder Neural Network predictions	6560.924	267.88	1699.292	166182.945
Real data	6441.367	253.0	1578.0	171955.6

Table 13: Line of business 3, process description capabilities.

Model	ay0	ay1	ay2	ay3	ay4	ay5	ay6	ay7	ay8	ay9	ay10	
Bayesian Neural Networks predictions	0.0	0.005	−0.032	−0.041	−0.086	−0.22	0.069	−0.965	−0.469	−0.275	−22.535	−0.935
Chain-Ladder Neural Network predictions	0.0	0.048	0.167	0.249	0.172	0.54	0.663	−6.263	−0.073	0.096	−19.902	−0.624

Table 14: Line of business 3, prediction errors in the different accident years.

Model	Relative error on the ultimate cost	Relative error on the reserve	Mean Absolute Error (MAE)
Bayesian Neural Networks	−1.079%	−7.333%	347.185
Chain-Ladder Neural Networks	+0.164%	+1.118%	271.285

Table 15: Line of business 4, relative prediction errors and MAE.

Data	Mean individual ultimate cost	0.05 ultimate cost quantile	median	0.995 ultimate cost quantile
Bayesian Neural Networks predictions	1737.461	59.074	412.883	41396.901
Chain-Ladder Neural Network predictions	1759.298	75.49	426.377	41940.318
Real data	1756.409	70.0	379.0	46127.64

Table 16: Line of business 4, process description capabilities.

Model	ay0	ay1	ay2	ay3	ay4	ay5	ay6	ay7	ay8	ay9	ay10	
Bayesian Neural Networks predictions	0.0	−0.021	−0.196	−0.621	−0.288	−0.262	0.052	0.124	−0.189	−0.167	0.013	−0.06
Chain-Ladder Neural Network predictions	0.0	0.101	0.194	0.328	0.051	0.127	0.102	0.128	−0.032	−0.069	−0.012	−0.047

Table 17: Line of business 4, prediction errors in the different accident years.

B Computational notes

The computation for this project where performed in Python 3.9.5. See Table 18 for the package versions used in this project.

Package	**Version**
logging	0.5.1.2
numpy	1.21.2
pandas	1.3.5
tensorflow	2.5.0
sklearn	0.24.2
tensorflow_probability	0.13.0

Table 18: Main packages versions for the project computation.