risks

MDPI

Article

DeepTriangle: A Deep Learning Approach to Loss Reserving

Kevin Kuo

Kasa AI, 3040 78th Ave SE #1271, Mercer Island, WA 98040, USA; kevin@kasa.ai

Received: 15 August 2019; Accepted: 12 September 2019; Published: 16 September 2019

Abstract: We propose a novel approach for loss reserving based on deep neural networks. The approach allows for joint modeling of paid losses and claims outstanding, and incorporation of heterogeneous inputs. We validate the models on loss reserving data across lines of business, and show that they improve on the predictive accuracy of existing stochastic methods. The models require minimal feature engineering and expert input, and can be automated to produce forecasts more frequently than manual workflows.

Keywords: loss reserving; machine learning; neural networks

1. Introduction

In the loss reserving exercise for property and casualty insurers, actuaries are concerned with forecasting future payments due to claims. Accurately estimating these payments is important from the perspectives of various stakeholders in the insurance industry. For the management of the insurer, the estimates of unpaid claims inform decisions in underwriting, pricing, and strategy. For the investors, loss reserves, and transactions related to them, are essential components in the balance sheet and income statement of the insurer. In addition, for the regulators, accurate loss reserves are needed to appropriately understand the financial soundness of the insurer.

There can be time lags both for reporting of claims, where the insurer is not notified of a loss until long after it has occurred, and for final development of claims, where payments continue long after the loss has been reported. Also, the amounts of claims are uncertain before they have fully developed. These factors contribute to the difficulty of the loss reserving problem, for which extensive literature exists and active research is being done. We refer the reader to England and Verrall (2002) for a survey of the problem and existing techniques.

Deep learning has garnered increasing interest in recent years due to successful applications in many fields (LeCun et al. 2015) and has recently made its way into the loss reserving literature. Wüthrich (2018b) augments the traditional chain ladder method with neural networks to incorporate claims features, Gabrielli and Wüthrich (2018) use neural networks to syntheisze claims data, and Gabrielli et al. (2018) and Gabrielli (2019) embed classical parametric loss reserving models into neural networks. More specifically, the development in Gabrielli et al. (2018) and Gabrielli (2019) proposes initializing a neural network so that, before training, it corresponds exactly to a classical model, such as the over-dispersed Poisson model. The training iterations then adjust the weights of the neural network to minimize the prediction errors, which can be interpreted as a boosting procedure.

In developing our framework, which we call DeepTriangle[1], we also draw inspiration from the existing stochastic reserving literature. Works that propose using data in addition to paid losses include

[1] A portmanteau of *deep learning* and *loss development triangle*.

Quarg and Mack (2004), which uses incurred losses, and Miranda et al. (2012), which incorporates claim count information. Moving beyond a single homogeneous portfolio, (Avanzi et al. (2016) considers the dependencies among lines of business within an insurer's portfolio, while Peremans et al. (2018) proposes a robust general multivariate chain ladder approach to accommodate outliers. There is also a category of models, referred to as state space or adaptive models, that allow parameters to evolve recursively in time as more data is observed (Chukhrova and Johannssen 2017). This iterative updating mechanism is similar in spirit to the continuous updating of neural network weights during model deployment.

The approach that we develop differs from existing works in many ways, and has the following advantages. First, it enables joint modeling of paid losses and claims outstanding for multiple companies simultaneously in a single model. In fact, the architecture can also accommodate arbitrary additional inputs, such as claim count data and economic indicators, should they be available to the modeler. Second, it requires no manual input during model updates or forecasting, which means that predictions can be generated more frequently than traditional processes, and, in turn, allows management to react to changes in the portfolio sooner.

The rest of the paper is organized as follows: Section 2 provides a brief overview of neural network terminology, Section 3 discusses the dataset used and introduces the proposed neural network architecture, Section 4 defines the performance metrics we use to benchmark our models and discuss the results, and Section 5 concludes.

2. Neural Network Preliminaries

For comprehensive treatments of neural network mechanics and implementation, we refer the reader to Goodfellow et al. (2016) and Chollet and Allaire (2018). A more actuarially oriented discussion can be found in Wuthrich and Buser (2019). To establish common terminology used in this paper, we present a brief overview in this section.

We motivate the discussion by considering an example feedforward network with fully connected layers represented in Figure 1, where the goal is to predict an output y from input x. The intermediate values, known as hidden layers and represented by $h_j^{[l]}$, try to transform the input data into representations that successively become more useful at predicting the output. The nodes in the figure are computed, for each layer $l = 1, \dots, L$, as

$$h_j^{[l]} = g^{[l]}(z_j^{[l]}), \tag{1}$$

where

$$z_j^{[l]} = w_j^{[l]T} h^{[l-1]} + b_j^{[l]}, \tag{2}$$

for $j = 1, \dots, n^{[l]}$. In these equations, a superscript $[l]$ denotes association with the layer l, a subscript j denotes association with the j-th component of the layer, of which there are $n^{[l]}$. The $g^{[l]}$ $(l = 1, \dots, L)$ are called activation functions, whose values $h^{[l]}$ are known as activations. The vectors $w_j^{[l]}$ and scalars $b_j^{[l]}$ are known as weights and biases, respectively, and together represent the parameters of the neural network, which are learned during training.

For $l = 1$, we define the previous layer activations as the input, so that the calculation for the first hidden layer becomes

$$h_j^{[1]} = g^{[1]}(w_j^{[1]T} x + b_j^{[1]}). \tag{3}$$

Also, for the output layer $l = L$, we compute the prediction

$$\hat{y} = h_j^{[L]} = g^{[L]}(w_j^{[L]T} h^{[L-1]} + b_j^{[L]}). \tag{4}$$

We can then think of a neural network as a sequence of function compositions $f = f_L \circ f_{L-1} \circ \dots \circ f_1$ parameterized as $f(x; W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]})$. Here, it should be mentioned that the $g^{[l]}$

($l = 1, \ldots, L$) are chosen to be nonlinear, except for possibly in the output layer. These nonlinearities are key to the success of neural networks, because otherwise we would have a trivial composition of linear models.

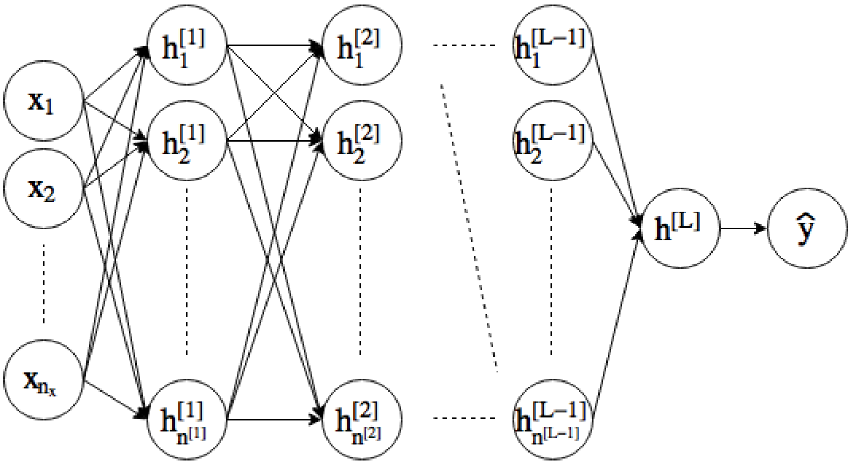

Figure 1. Feedforward neural network.

Each neural network model is specified with a specific loss function, which is used to measure how close the model predictions are to the actual values. During model training, the parameters discussed above are iteratively updated in order to minimize the loss function. Each update of the parameters typically involves only a subset, or mini-batch, of the training data, and one complete pass through the training data, which includes many updates, is known as an epoch. Training a neural network often requires many passes through the data.

3. Data and Model Architecture

In this section, we discuss the dataset used for our experiments and the proposed model architecture.

3.1. Data Source

We use the National Association of Insurance Commissioners (NAIC) Schedule P triangles (Meyers and Shi 2011). The dataset corresponds to claims from accident years 1988–1997, with development experience of 10 years for each accident year. In Schedule P data, the data is aggregated into accident year-development year records. The procedure for constructing the dataset is detailed in Meyers (2015).

Following Meyers (2015), we restrict ourselves to a subset of the data which covers four lines of business (commercial auto, private personal auto, workers' compensation, and other liability) and 50 companies in each line of business. This is done to facilitate comparison to existing results.

We use the following variables from the dataset in our study: line of business, company code, accident year, development lag, incurred loss, cumulative paid loss, and net earned premium. Claims outstanding, for the purpose of this study, is derived as incurred loss less cumulative paid loss. The company code is a categorical variable that denotes which insurer the records are associated with.

3.2. Training/Testing Setup

Let indices $1 \leq i \leq I$ denote accident years and $1 \leq j \leq J$ denote development years under consideration. Also, let $\{P_{i,j}\}$ and $\{OS_{i,j}\}$ denote the *incremental* paid losses and the *total* claims outstanding, or case reserves, respectively.

Then, at the end of calendar year I, we have access to the observed data

$$\{P_{i,j} : i = 1, \ldots, I; j = 1, \ldots, I - i + 1\} \tag{5}$$

and

$$\{OS_{i,j} : i = 1, \ldots, I; j = 1, \ldots, I - i + 1\}. \tag{6}$$

Assume that we are interested in development through the Ith development year; in other words, we only forecast through the eldest maturity in the available data. The goal then is to obtain predictions for future values $\{\widehat{P}_{i,j} : i = 2, \ldots, I; j = i + 1, \ldots, I\}$ and $\{\widehat{OS}_{i,j} : i = 2, \ldots, I; j = i + 1, \ldots, I\}$. We can then determine ultimate losses (UL) for each accident year $i = 1, \ldots, I$ by calculating

$$\widehat{UL}_i = \left(\sum_{j=1}^{I-i+1} P_{i,j} \right) + \left(\sum_{j=I-i+2}^{I} \widehat{P}_{i,j} \right). \tag{7}$$

In our case, data as of year end 1997 is used for training. We then evaluate predictive performance on the development year 10 cumulative paid losses.

3.3. Response and Predictor Variables

In DeepTriangle, each training sample is associated with an accident year-development year pair, which we refer to thereinafter as a *cell*. The response for the sample associated with accident year i and development year j is the sequence

$$(Y_{i,j}, Y_{i,j+1}, \ldots, Y_{i,I-i+1}), \tag{8}$$

where each $Y_{i,j} = (P_{i,j}/NPE_i, OS_{i,j}/NPE_i)$, and NPE_i denotes the net earned premium for accident year i. Working with loss ratios makes training more tractable by normalizing values into a similar scale.

The predictor for the sample contains two components. The first component is the observed history as of the end of the calendar year associated with the cell:

$$(Y_{i,1}, Y_{i,2}, \ldots, Y_{i,j-1}). \tag{9}$$

In other words, for each accident year and at each evaluation date for which we have data, we attempt to predict future development of the accident year's paid losses and claims outstanding based on the observed history as of that date. While we are ultimately interested in $P_{i,j}$, the paid losses, we include claims outstanding as an auxiliary output of the model. We elaborate on the reasoning behind this approach in the next section.

The second component of the predictor is the company identifier associated with the experience. Because we include experience from multiple companies in each training iteration, we need a way to differentiate the data from different companies. We discuss handling of the company identifier in more detail in the next section.

3.4. Model Architecture

As shown in Figure 2, DeepTriangle is a multi-task network (Caruana 1997) using a sequence-to-sequence architecture (Srivastava et al. 2015; Sutskever et al. 2014) with two prediction goals: paid loss and claims outstanding. We construct one model for each line of business and each model is trained on data from multiple companies.

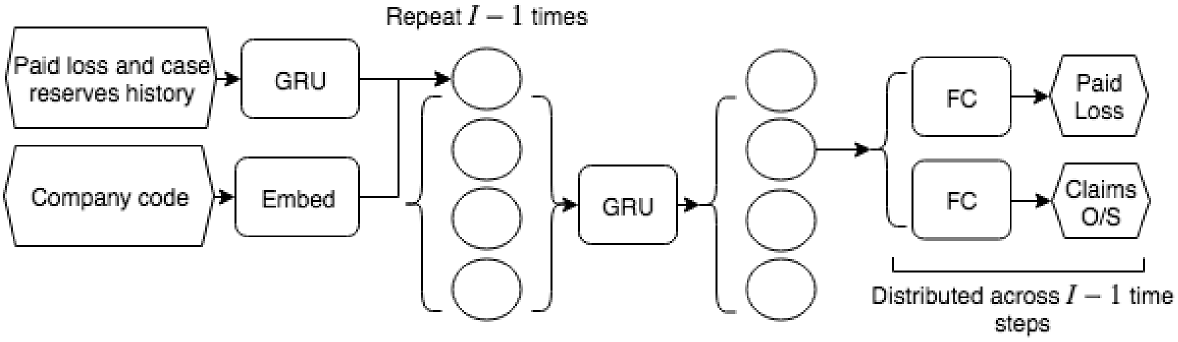
Repeat $I-1$ times
Paid loss and case reserves history
Company code
GRU
Embed
GRU
FC
FC
Paid Loss
Claims O/S
Distributed across $I-1$ time steps

Figure 2. DeepTriangle architecture. *Embed* denotes embedding layer, *GRU* denotes gated recurrent unit, *FC* denotes fully connected layer.

3.4.1. Multi-Task Learning

Since the two target quantities, paid loss and claims outstanding, are related, we expect to obtain better performance by jointly training than predicting each quantity independently. While Caruana (1997) contains detailed discourse on the specific mechanisms of multi-task learning, we provide some heuristics on why it may improve predictions: by using the response data for claims outstanding, we are effectively increasing the training data size since we are providing more signals to the learning algorithm; there may be hidden features, useful for predicting paid losses, that are more easily learned by trying to predict claims outstanding; also, by trying to predict claims outstanding during training, we are imposing a bias towards neural network weight configurations which perform that task well, which lessens the likelihood of arriving at a model that overfits to random noise.

3.4.2. Sequential Input Processing

For handling the time series of paid losses and claims outstanding, we use gated recurrent units (GRU) (Chung et al. 2014), which is a type of recurrent neural network (RNN) building block that is appropriate for sequential data. A graphical representation of a GRU is shown in Figure 3, and the associated equations are as follows[2]:

$$\tilde{h}^{<t>} = \tanh(W_h[\Gamma_r h^{<t-1>}, x^{<t>}] + b_h) \tag{10}$$

$$\Gamma_r^{<t>} = \sigma(W_r[h^{<t-1>}, x^{<t>}] + b_r) \tag{11}$$

$$\Gamma_u^{<t>} = \sigma(W_u[h^{<t-1>}, x^{<t>}] + b_u) \tag{12}$$

$$h^{<t>} = \Gamma_u^{<t>}\tilde{h}^{<t>} + (1-\Gamma_u^{<t>})h^{<t-1>}. \tag{13}$$

Here, $h^{<t>}$ and $x^{<t>}$ represent the activation and input values, respectively, at time t, and σ denotes the logistic sigmoid function defined as

$$\sigma(x) = \frac{1}{1+\exp(-x)}. \tag{14}$$

W_h, W_r, W_u, b_h, b_r, and b_u are the appropriately sized weight matrices and biases to be learned. Intuitively, the activations $h^{<t>}$ provide a way for the network to maintain state and "remember" values from early values of the input sequence. The values $\tilde{h}^{<t>}$ can be thought of as candidates to replace the current state, and $\Gamma_u^{<t>}$ determines the weighting between the previous state and the candidate state. We remark that although the GRU (and RNN in general) may seem opaque at first,

[2] Note the use of angle brackets to index position in a sequence rather than layers in a feedforward neural network as in Section 2.

they contain sequential instructions for updating weights just like vanilla feedforward neural networks (and can in fact be interpreted as such (Goodfellow et al. 2016)).

We first encode the sequential predictor with a GRU to obtain a summary encoding of the historical values. We then repeat the output $I - 1$ times before passing them to a decoder GRU that outputs its hidden state for each time step. The factor $I - 1$ is chosen here because for the Ith accident year, we need to forecast $I - 1$ timesteps into the future. For both the encoder and decoder GRU modules, we use 128 hidden units and a dropout rate of 0.2. Here, dropout refers to the regime where, during training, at each iteration, we randomly set the output of the hidden units to zero with a specified probability, in order to reduce overfitting (Srivastava et al. 2014). Intuitively, dropout accomplishes this by approximating an ensemble of sub-networks that can be constructed by removing some hidden units.

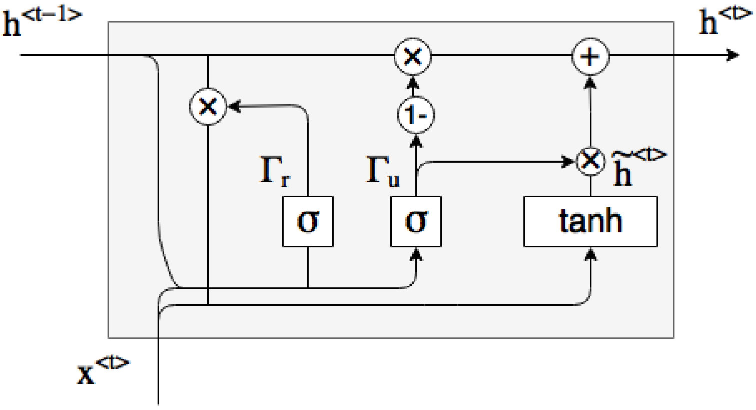

Figure 3. Gated recurrent unit.

3.4.3. Company Code Embeddings

The company code input is first passed to an embedding layer. In this process, each company is mapped to a fixed length vector in $\mathbb{R}^k$, where k is a hyperparameter. In our case, we choose k = number of levels − 1 = 49, as recommended in Guo and Berkhahn (2016). In other words, each company is represented by a vector in $\mathbb{R}^{49}$. This mapping mechanism is part of the neural network and hence is learned during the training of the network, instead of in a separate data preprocessing step, so the learned numerical representations are optimized for predicted the future paid losses. Companies that are similar in the context of our claims forecasting problem are mapped to vectors that are close to each other in terms of Euclidean distance. Intuitively, one can think of this representation as a proxy for characteristics of the companies, such as size of book and case reserving philosophy. Categorical embedding is a common technique in deep learning that has been successfully applied to recommendation systems (Cheng et al. 2016) and retail sales prediction (Guo and Berkhahn 2016). In the actuarial science literature, Richman and Wuthrich (2018) use embedding layers to capture characteristics of regions in mortality forecasting, while Gabrielli et al. (2018) apply them to lines of business factors in loss reserving.

3.4.4. Fully Connected Layers and Outputs

Each timestep of the decoded sequence from the GRU decoder is then concatenated with the company embedding output. The concatenated values are then passed to two subnetworks of fully connected layers, each of which shares weights across the timesteps. The two subnetworks correspond to the paid loss and case outstanding predictions, respectively, and each consists of a hidden layer of 64 units with a dropout rate of 0.2, followed by an output layer of 1 unit to represent the paid loss or claims outstanding at a time step.

Rectified linear unit (ReLU) (Nair and Hinton 2010), defined as

$$x \mapsto \max(0, x), \tag{15}$$

is used as the activation function (which we denote by g in Section 2) for all fully connected layers, including both of the output layers. We remark that this choice of output activation implies we only predict nonnegative cash flows, i.e., no recoveries. This assumption is reasonable for the dataset we use in our experiments, but may be modified to accommodate other use cases.

3.5. Deployment Considerations

While one may not have access to the latest experience data of competitors, the company code predictor can be used to incorporate data from companies within a group insurer. During training, the relationships among the companies are inferred based on historical development behavior. This approach provides an automated and objective alternative to manually aggregating, or clustering, the data based on knowledge of the degree of homogeneity among the companies.

If new companies join the portfolio, or if the companies and associated claims are reorganized, one would modify the embedding input size to accommodate the new codes, leaving the rest of the architecture unchanged, then refit the model. The network would then assign embedding vectors to the new companies.

Since the model outputs predictions for each triangle cell, one can calculate the traditional age-to-age, or loss development, factors (LDF) using the model forecasts. Having a familiar output may enable easier integration of DeepTriangle into existing actuarial workflows.

Insurers often have access to richer information than is available in regulatory filings, which underlies the experiments in this paper. For example, in addition to paid and incurred losses, one may include claim count triangles so that the model can also learn from, and predict, frequency information.

4. Experiments

We now describe the performance metrics for benchmarking the models and training details, then discuss the results.

4.1. Evaluation Metrics

We aim to produce scalar metrics to evaluate the performance of the model on each line of business. To this end, for each company and each line of business, we calculate the actual and predicted ultimate losses as of development year 10, for all accident years combined, then compute the root mean squared percentage error (RMSPE) and mean absolute percentage error (MAPE) over companies in each line of business. Percentage errors are used in order to have unit-free measures for comparing across companies with vastly different sizes of portfolios. Formally, if $\mathcal{C}_l$ is the set of companies in line of business l,

$$MAPE_l = \frac{1}{|\mathcal{C}_l|} \sum_{C \in \mathcal{C}_l} \left| \frac{\widehat{UL}_C - UL_C}{UL_C} \right|, \tag{16}$$

and

$$RMSPE_l = \sqrt{\frac{1}{|\mathcal{C}_l|} \sum_{C \in \mathcal{C}_l} \left(\frac{\widehat{UL}_C - UL_C)}{UL_C} \right)^2} \tag{17}$$

where $\widehat{UL}_C$ and UL_C are the predicted and actual cumulative ultimate losses, respectively, for company C.

An alternative approach for evaluation could involve weighting the company results by the associated earned premium or using dollar amounts. However, due to the distribution of company sizes in the dataset, the weights would concentrate on a handful of companies. Hence, to obtain a more balanced evaluation, we choose to report the unweighted percentage-based measures outlined above. We note that the evaluation of reserving models is an ongoing area of research; and refer the reader to Martinek (2019) for a recent analysis.

4.2. Implementation and Training

The loss function is computed as the average over the forecasted time steps of the mean squared error of the predictions. The losses for the outputs are then averaged to obtain the network loss. Formally, for the sample associated with cell (i, j), we can write the per-sample loss as

$$\frac{1}{I-i+1-(j-1)} \sum_{k=j}^{I-i+1} \frac{(\widehat{P_{i,k}} - P_{i,k})^2 + (\widehat{OS_{i,k}} - OS_{i,k})^2}{2}. \tag{18}$$

For optimization, we use the AMSGRAD (Reddi et al. 2018) variant of ADAM with a learning rate of 0.0005. We train each neural network for a maximum of 1000 epochs with the following early stopping scheme: if the loss on the validation set does not improve over a 200-epoch window, we terminate training and revert back to the weights on the epoch with the lowest validation loss. The validation set used in the early stopping criterion is defined to be the subset of the training data that becomes available after calendar year 1995. For each line of business, we create an ensemble of 100 models, each trained with the same architecture but different random weight initialization. This is done to reduce the variance inherent in the randomness associated with neural networks.

We implement DeepTriangle using the keras R package (Chollet et al. 2017) and TensorFlow (Abadi et al. 2015), which are open source software for developing neural network models. Code for producing the experiment results is available online.[3]

4.3. Results and Discussion

In Table 1 we tabulate the out-of-time performance of DeepTriangle against other models: the Mack chain-ladder model (Mack 1993), the bootstrap ODP model (England and Verrall 2002), an AutoML model, and a selection of Bayesian Markov chain Monte Carlo (MCMC) models from Meyers (2015) including the correlated incremental trend (CIT) and leveled incremental trend (LIT) models. For the stochastic models, we use the means of the predictive distributions as the point estimates to which we compare the actual outcomes. For DeepTriangle, we report the averaged predictions from the ensembles.

Table 1. Performance comparison of various models. DeepTriangle and AutoML are abbreviated to DT and ML, respectively. The best metric for each line of business is in bold.

Line of Business	Mack	ODP	CIT	LIT	ML	DT
MAPE						
Commercial Auto	0.060	0.217	0.052	0.052	0.068	**0.043**
Other Liability	0.134	0.223	0.165	0.152	0.142	**0.109**
Private Passenger Auto	0.038	0.039	0.038	0.040	0.036	**0.025**
Workers' Compensation	0.053	0.105	0.054	0.054	0.067	**0.046**
RMSPE						
Commercial Auto	0.080	0.822	0.076	0.074	0.096	**0.057**
Other Liability	0.202	0.477	0.220	0.209	0.181	**0.150**
Private Passenger Auto	0.061	0.063	0.057	0.060	0.059	**0.039**
Workers' Compensation	0.079	0.368	0.080	0.080	0.099	**0.067**

[3] https://github.com/kasaai/deeptriangle.

The AutoML model is developed by automatically searching over a set of common machine learning techniques. In the implementation we use, it trains and cross-validates a random forest, an extremely randomized forest, a random grid of gradient boosting machines, a random grid of deep feedforward neural networks, and stacked ensembles thereof (The H2O.ai team 2018). Details of these algorithms can be found in Friedman et al. (2001). Because the machine learning techniques produce scalar outputs, we use an iterative forecasting scheme where the prediction for a timestep is used in the predictor for the next timestep.

We see that DeepTriangle improves the performance of the popular chain ladder and ODP models, common machine learning models, and Bayesian stochastic models.

In addition to aggregated results for all companies, we also investigate qualitatively the ability of DeepTriangle to learn development patterns of individual companies. Figures 4 and 5 show the paid loss development and claims outstanding development for the commercial auto line of Company 1767 and the workers' compensation line of Company 337, respectively. We see that the model captures the development patterns for Company 1767 reasonably well. However, it is unsuccessful in forecasting the deteriorating loss ratios for Company 337's workers' compensation book.

We do not study uncertainty estimates in this paper nor interpret the forecasts as posterior predictive distributions; rather, they are included to reflect the stochastic nature of optimizing neural networks. We note that others have exploited randomness in weight initialization in producing predictive distributions (Lakshminarayanan et al. 2017), and further research could study the applicability of these techniques to reserve variability.

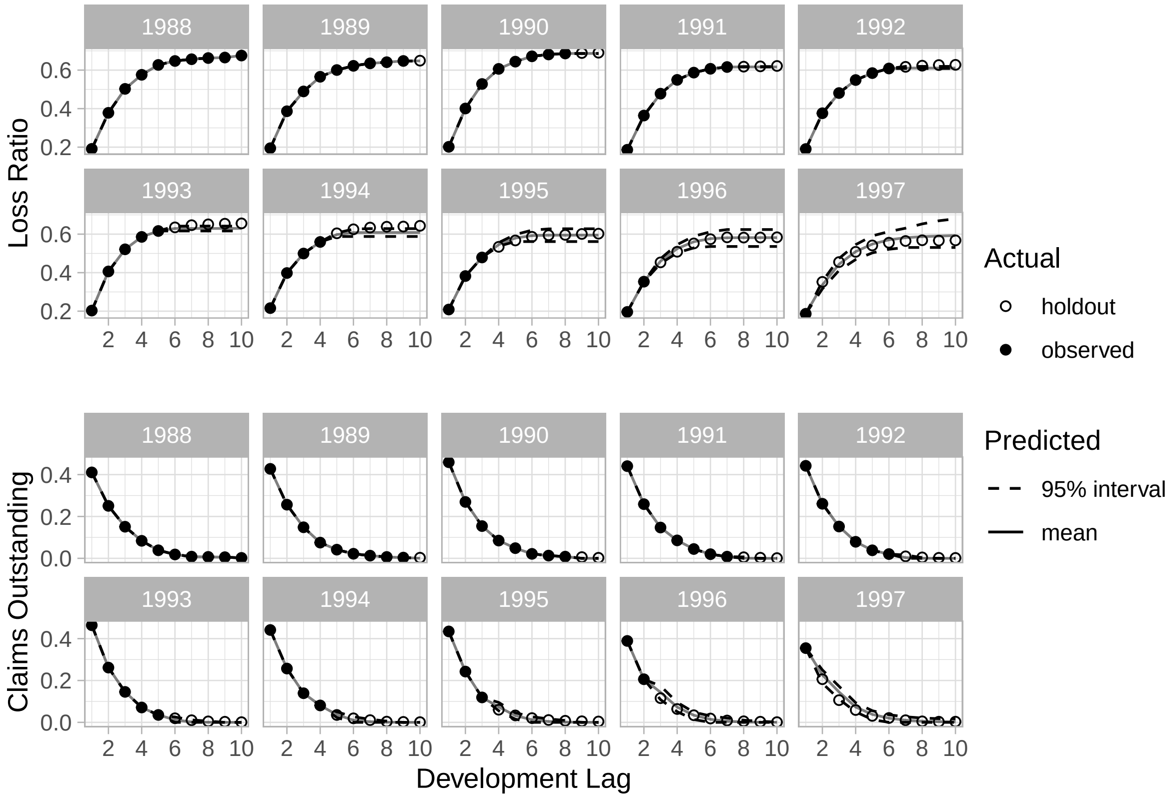

Figure 4. Development by accident year for Company 1767, commercial auto.

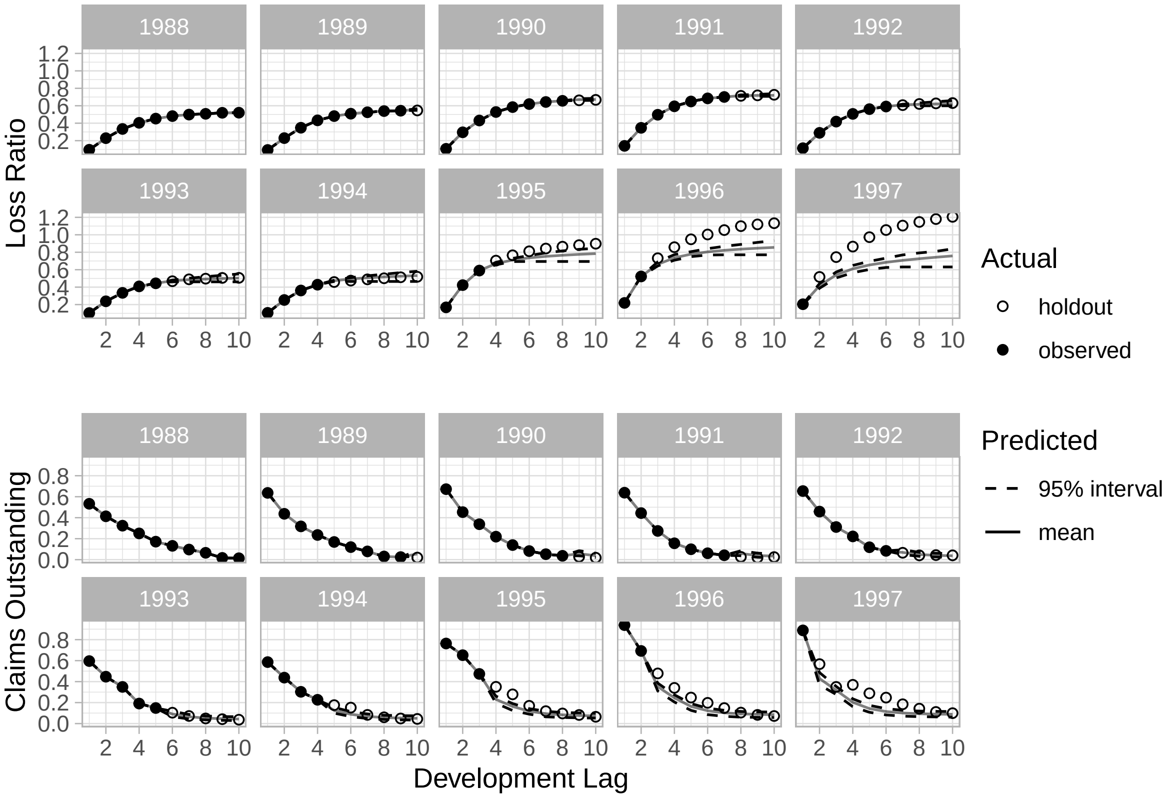

Figure 5. Development by accident year for Company 337, workers' compensation.

5. Conclusions

We introduce DeepTriangle, a deep learning framework for forecasting paid losses. Our models are able to attain performance comparable, by our metrics, to modern stochastic reserving techniques, without expert input. This means that one can automate model updating and report production at the desired frequency (although we note that, as with any automated machine learning system, a process involving expert review should be implemented). By using neural networks, we can incorporate multiple heterogeneous inputs and train on multiple objectives simultaneously, and also allow customization of models based on available data. To summarize, this framework maintains accuracy while providing automatability and extensibility.

We analyze an aggregated dataset with limited features in this paper because it is publicly available and well studied, but one can extend DeepTriangle to incorporate additional data, such as claim counts.

Deep neural networks can be designed to extend recent efforts, such as Wüthrich (2018a), on applying machine learning to claims level reserving. They can also be designed to incorporate additional features that are not handled well by traditional machine learning algorithms, such as claims adjusters' notes from free text fields and images.

While this study focuses on prediction of point estimates, future extensions may include outputting distributions in order to address reserve variability.

Funding: This research received no external funding.

Acknowledgments: We thank Sigrid Keydana, Ronald Richman, the anonymous reviewers, and the volunteers on the Casualty Actuarial Society Committee on Reserves (CASCOR) who helped to improve the paper through helpful comments and discussions.

Conflicts of Interest: The author declares no conflict of interest.

References

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, and et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. *arXiv* arXiv:1603.04467.

Avanzi, Benjamin, Greg Taylor, Phuong Anh Vu, and Bernard Wong. 2016. Stochastic loss reserving with dependence: A flexible multivariate tweedie approach. *Insurance: Mathematics and Economics* 71: 63–78. [CrossRef]

Caruana, Rich. 1997. Multitask learning. *Machine Learning* 28: 41–75. [CrossRef]

Cheng, Heng-Tze, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, Hemal Shah, Levent Koc, Jeremiah Harmsen, and et al. 2016. Wide & deep learning for recommender systems. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems—DLRS 2016, Boston, MA, USA, September 15. [CrossRef]

Chollet, Francois, and Joseph J. Allaire. 2018. *Deep Learning with R*. Shelter Island: Manning Publications.

Chollet, François, and Joseph J. Allaire. 2017. R Interface to Keras. Available online: https://github.com/rstudio/keras (accessed on 7 September 2019).

Chukhrova, Nataliya, and Arne Johannssen. 2017. State space models and the kalman-filter in stochastic claims reserving: Forecasting, filtering and smoothing. *Risks* 5: 30. [CrossRef]

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

England, Peter D., and Richard J. Verrall. 2002. Stochastic claims reserving in general insurance. *British Actuarial Journal* 8: 443–518. [CrossRef]

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. New York: Springer.

Gabrielli, Andrea. 2019. A Neural Network Boosted Double Over-Dispersed Poisson Claims Reserving Model. Available online: https://ssrn.com/abstract=3365517 (accessed on 15 September 2019).

Gabrielli, Andrea, Ronald Richman, and Mario V. Wuthrich. 2018. Neural Network Embedding of the Over-Dispersed Poisson Reserving Model. Available online: https://ssrn.com/abstract=3288454 (accessed on 15 September 2019).

Gabrielli, Andrea, and Mario V. Wüthrich. 2018. An individual claims history simulation machine. *Risks* 6: 29. [CrossRef]

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. Cambridge: MIT Press.

Guo, Cheng, and Felix Berkhahn. 2016. Entity embeddings of categorical variables. *arXiv* arXiv:1604.06737.

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In Proceedings of the Advances in Neural Information Processing Systems 30, Long Beach, CA, USA, December 4–9.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521: 436. [CrossRef] [PubMed]

Mack, Thomas. 1993. Distribution-free calculation of the standard error of chain ladder reserve estimates. *ASTIN Bulletin* 23: 213–25. [CrossRef]

Martinek, László. 2019. Analysis of stochastic reserving models by means of naic claims data. *Risks* 7: 62. [CrossRef]

Meyers, Glenn. 2015. *Stochastic Loss Reserving Using Bayesian MCMC Models*. Arlington: Casualty Actuarial Society.

Meyers, Glenn, and Peng Shi. 2011. Loss Reserving Data Pulled from NAIC Schedule p. Available online: http://www.casact.org/research/index.cfm?fa=loss_reserves_data (accessed on 7 September 2019).

Miranda, María Dolores Martínez, Jens Perch Nielsen, and Richard Verrall. 2012. Double chain ladder. *ASTIN Bulletin: The Journal of the IAA* 42: 59–76.

Nair, Vinod, and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning, Haifa, Israel, June 21–24.

Peremans, Kris, Stefan Van Aelst, and Tim Verdonck. 2018. A robust general multivariate chain ladder method. *Risks* 6: 108. [CrossRef]

Quarg, Gerhard, and Thomas Mack. 2004. Munich chain ladder. *Blätter der DGVFM* 26: 597–630. [CrossRef]

Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar. 2018. On the convergence of adam and beyond. In Proceedings of the 6th International Conference on Learning Representations, Vancouver, BC, Canada, April 30–May 3.

Richman, Ronald, and Mario V. Wuthrich. 2018. A Neural Network Extension of the Lee-Carter Model to Multiple Populations. Available online: https://ssrn.com/abstract=3270877 (accessed on 7 September 2019).

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15: 1929–58.

Srivastava, Nitish, Elman Mansimov, and Ruslan Salakhutdinov. 2015. Unsupervised learning of video representations using LSTMs. *arXiv* arXiv:1502.04681.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In Proceedings of the Advances in Neural Information Processing Systems 27, Montreal, QC, Canada, December 8–13.

The H2O.ai team. 2018. *h2o: R Interface for H2O*. R Package Version 3.20.0.8. Mountain View: H2O.ai.

Wüthrich, Mario V. 2018a. Machine learning in individual claims reserving. *Scandinavian Actuarial Journal* 1–16. [CrossRef]

Wüthrich, Mario V. 2018b. Neural networks applied to chain–ladder reserving. *European Actuarial Journal* 8: 407–36. [CrossRef]

Wuthrich, Mario V., and Christoph Buser. 2019. Data analytics for non-life insurance pricing. *Swiss Finance Institute Research Paper*. doi:10.2139/ssrn.2870308. [CrossRef]