NEURAL NETWORKS IN INSURANCE

MAGALI-CHEN GOULET

A Thesis
in
The Department
of
Mathematics and Statistics

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Science (Mathematics) at
Concordia University
Montreal, Quebec, Canada

July 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Magali-Chen Goulet

Entitled: Neural Networks in Insurance

and submitted in partial fulfillment of the requirements for the degree of

Master of Science (Mathematics)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

____ Chair
Dr. Frédéric Godin

____ Examiner
Dr. Simone Brugiapaglia

____ Supervisor
Dr. Mélina Mailhot

Approved by ____
Chair of Department or Graduate Program Director

____________2021 ________________________
Dean of Faculty

Abstract

Neural Networks in Insurance

To date, in the insurance industry, the premium for a given risk is based on the expected claim amount since the models used are only meant to calculate a mean response. However, getting more information about the distribution of each single risk would be useful for risk assessment. A method in Neural Networks (NN) called Tractable Approximate Gaussian Inference (TAGI) by Goulet et al. (2020) allows to study each response individually since each output follows its own Normal distribution. The main contributions of this thesis are to make this technique available through an open source package, to apply TAGI in insurance and compare it to other techniques and to study risk measures with it.

Acknowledgments

First, a special thank you to my supervisor, Dr. Mélina Mailhot, not only for her great expertise and guidance through my thesis journey, but for the human being that she is and all the support she gave me.

Secondly, I am sincerely grateful to Dr. James-A. Goulet and Dr. Luong Ha Nguyen for their ideas and support on TAGI.

Moreover, thanks to my closest friends who reminded me that life is not just about work and performance, by bringing me happiness and joyful moments.

Last but not least, I would like to thank my parents, for their unconditional love and everything they have done for me. Thanks for giving me the chance to live the life I have now, and for becoming the person I am.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

In the insurance industry, whether in Life insurance, Group Benefits or in Property and Casualty, the formula to calculate a premium is represented as follows,

$$Premium = E[Claim] + Taxes + Fees,$$

where E[Claim] is the expected claim amount and the fees include a risk loading and a profit margin. Many techniques exist to predict the expected claim amount. It can go from standard methods such as Generalized Linear Models (GLM) to machine learning models. In Group Benefits, it can be calculated as a combination of the client's past experience and trends observed in its industry.

As an insurer, it is useful to calculate the expected claim amount, not only for the premium, but also for risk assessment. If the expected claim amount is too high, the insurer might consider not binding a given policy. However, knowing the distribution of each risk would be even more helpful for risk assessment. For example, for a given policy, the expected claim, calculated from a traditional method, can be considered as acceptable if the amount is not particularly high. The policy would be bound. However, suppose it would be possible for the insurer to also calculate the policy's risk distribution. It can reveal a very high volatility and a non-negligible probability that the claim amount would be higher than the insurer's risk appetite. Then, knowing the risk distribution of any single risk or policy could be used to decline policies that are normally bound or to adjust premiums accordingly.

An industry which could particularly benefit from this new feature would be the Commercial Lines (CL) in Property and Casualty, since they consist in multiple types of risk with less population as in the Personal Lines, which provide auto and house insurance coverages to individuals. CL provide liability and property coverages to businesses. They can be of any types such as restaurants, contractors, retailers, etc. which are all very different risks. Thus, knowing the risk distribution of each single risk would benefit this industry.

To do so, there are already some techniques that exist in the literature such as Quantile Regression (QR) which is commonly used to estimate the median of the response variable. However, its use is not restricted to the median since it can predict any quantiles of the response variable. It is particularly interesting in our case, since insurers are generally worried about the extreme values. Predicting high quantiles of potential claims can provide better risk assessment of a given portfolio of risks, lines of business and/or coverages.

More recently, machine learning is growing in popularity in many fields of study. Many techniques are studied, developed and improved and some of them provide more information about the predicted responses than only their means. Notably, methods from Gradient Boosting (GB) and Neural Networks (NN) achieve that objective. Tractable Approximate Gaussian Inference (TAGI) by Goulet et al. (2020) is extensively reviewed in this thesis, as for all the model's parameters, the predicted responses follow a Normal distribution which, not only answers to the initial purpose of this thesis, but also opens up opportunities for diverse areas of study. Since insurance is our subject of interest, we find interesting to compare general performance of a GB method to TAGI on a medical cost dataset. For its ability to model large amounts of data, efficiency and popularity of use, Extreme Gradient Boosting (XGBoost) by Chen and Guestrin (2016) is the chosen GB method to compare with TAGI.

Moreover, to test the impact of variation of the input variables on a variable of interest, sensitivity analysis is commonly used in pricing and reserving by insurers. Partial derivatives can be used as "local" sensitivity measures to understand the impact of a change on a single input to the model outputs. They can be estimated in TAGI since all model's parameters and states follow a Normal distribution and the outputs are functions of the inputs. Detailed explanations on the first and second derivatives are provided in Section 3 and we also develop the third one.

Another interesting approach to improve risk assessment is to study risk measures. There are different measures that are used to quantify risk from many angles, especially in situations with evidence of randomness. For example, in finance, they can be used to assess volatility of portfolio returns so that an investment strategy can be adjusted accordingly. Popular risk measures are Value at Risk (VaR) and Tail Value at Risk (TVaR). Let X be the random variable of interest. Then, $\mathrm{VaR}_\alpha(X)$ is the quantile risk measure, which represents the minimum value of X that should not be exceeded with probability $\alpha \in [0, 1]$. Insurers are often interested in the potential extreme values for claims. Since they are in the right tail of the claim distribution, risk measures at 95% and 99% α-levels are typically used.

In addition, if interested in the remaining (1-α)% scenario, $\text{TVaR}_\alpha(X)$ is the conditional expectation given that X exceeds $\text{VaR}_\alpha(X)$. An alternative to TVaR is the Range Value at Risk (RVaR), which instead of considering all values starting from $\text{VaR}_\alpha(X)$ up to the end of the X distribution, caps the conditional expectation to $\text{VaR}_{\alpha^+}(X)$, where $0 \leq \alpha < \alpha^+ < 1$. Moreover, the upper bound of those three risk measures, obtained from the Chernoff inequality, is called the Entropic Value at Risk (EVaR). All those measures are of interest in actuarial science, especially in our case to capture the risk of large losses in claims predictive models. Risk measures study was not yet performed on TAGI, so we take that opportunity to test them on TAGI models. We demonstrate layer by layer how modifying the inputs have an impact on the resulting distribution of the outputs and hence, on the risk measures. Moreover, we study the performance of TAGI in prediciting quantiles and compare it to the specialized technique to do so, QR.

To summarize the contributions of this thesis, it is to be noted that TAGI was initially developed in Matlab [2] programming language. We developed a comprehensive package in R. As the major contribution, the package "tagi"[3] contains functions to perform TAGI on any datasets and functions to compute the first and second derivatives (the last one was not implemented anywhere before). Vignettes to guide the users through the package and a reference manual are also provided. The other contributions are all the work around TAGI discussed above such as comparing TAGI's general performance to XGBoost, developing the third partial derivative and studying risk measures on TAGI, which includes comparing it at predicting quantiles with QR.

[2]Available at https://github.com/CivML-PolyMtl/TAGI.

[3]Available at https://github.com/mgoulet847/tagi. Manual is included in Appendix D.

2 Machine Learning

Machines can develop a form of intelligence, which is defined as Artificial Intelligence (AI). Machine Learning (ML), considered a subset of AI, is the field of study where techniques are developed so that machines can learn.

2.1 Common Types of Learning

2.1.1 Supervised Learning

Supervised learning consists in giving labeled data to the model, which contains both the observations in input and their corresponding output. It would be trained using the data provided and its objective is to be able to make predictions given new observations in input which follow the same structure as the training set.

Classification and regression problems can be solved using such technique. A classification problem occurs when the output to predict is a class. Among a possible set of categories, the model assigns one to each observation. A regression problem occurs when the variable to predict is numerical, not categorical. Supervised learning commonly uses neural networks (refer to Section 2.2), random forests, logistic regression, decision trees, etc. Clustering techniques can also be supervised. In k-nearest neighbors clustering, an observation is classified depending on the k categorized observations close to it. Its class would result in the most important that is among its k neighbors. For example, if $k = 5$, then we would look at the five nearest neighbors of the observation of interest. For example, if the exercise consists in assigning a color to a given dot and that the five nearest dots are three red, one blue and one green, then since the majority are red, the dot of interest would be classified as such.

Backtesting can be used to test the predictive models developed using supervised learning. It consists in using real historical data since the response to the input variables are known. For the same inputs, outputs from model are compared to the real results to assess accuracy of the predictions.

2.1.2 Unsupervised Learning

Unsupervised learning consists in giving unlabeled data to the model, which contains observations in input, but not necessarily their outputs. The model's objective is to understand the relationships and patterns between the data.

Association and clustering are types of unsupervised learning problems. Association technique can be used in large datasets to assess relationships between variables. For example, for marketing purposes, companies are interested in clients' behavior. Looking at purchasing histories can lead to interesting associations, such as which products are generally bought together. Clustering consists in assessing patterns between uncategorized data such that clusters are created according to the structure obtained while learning. Different types of clustering techniques exist such as k-means and k-medoids.

In k-means clustering, the number of clusters k is set in advance. Each observation is assigned to one cluster, such that the sum of squares within each cluster is minimized. A centroid, which represents the mean of a cluster, is drawn among the data for each cluster. Each observation is closest to a given centroid, so it is assigned to its corresponding cluster. However, means are sensitive to outliers. An alternative to k-means is k-medoids clustering by Rdusseeun and Kaufman (1987), which, instead of using the centroid, an actual point in the cluster is chosen to be the center of the cluster. That point is called the medoid.

2.1.3 Semi-Supervised Learning

Semi-supervised learning is a hybrid learning type, as a mixture of supervised and unsupervised learning. The model is fed with both labeled and unlabeled data so that it can efficiently learn from the labeled data and still use the unlabeled one. In fact, the model uses the labeled data to make predictions for the unlabeled ones.

Semi-supervised learning is commonly used with image, text and audio data, which are situations where there are generally more unlabeled than labeled data. An example would be for image recognition. Some pictures would be labeled but most of them would not, since it is time-consuming to manually identify or classify pictures. To improve consumers' experience, some insurers thought of speeding up the claim process by developing image recognition. For example, if a client just had a car accident, he can send a picture of his damaged car to the insurer. Then, the algorithm (developed using semi-supervised learning) can immediately determine if the car is a total loss which reduces delays and wait for the claimant to be paid.

2.1.4 Reinforcement Learning

Reinforcement learning consists in rewarding and/or penalizing actions, where a specific objective is defined. There is no training dataset in this framework, which implies that there are no observations. However, a reward function is used to identify and quantify the desirability of actions to be made by an agent which has to learn in an environment. Therefore, given an objective (or a set of objectives), the agent performs an action in its current environment. Depending on how that specific action can be beneficial in achieving the objective, positive or negative feedback is given to the agent, which learns from it. The same action performed in a different environment would not necessarily result in the same feedback as it would depict another situation. Ultimately, a sequence of decisions is optimized to reach the objective, which means that an action should be chosen if it is the most beneficial to the objective in the long-run.

An example would be to make a machine play a game. It could be any games, but as a general example, suppose the objective is to maximize a number of points. Given a specific environment, some actions can result in obtaining different number of points whereas others can make the agent lose some. The agent would then learn to follow a sequence of actions that would result in the highest score. For example, in a given environment, let the immediate reward for choosing Action A and Action B be a and b points respectively, where $a > b$. The chosen action should be the one that maximizes the final score at the end of the game, which is not necessarily the action which results in the immediate highest number of points.

In finance, reinforcement learning can be used to optimize allocation of assets in portfolio management, where the objectives are generally to maximize expected return and minimize financial risk.

2.2 Neural Networks

In insurance, as for many other fields, coming along with advanced technologies, we are now dealing with a tremendous amount of data. A technique well suited to handle a large dataset is Neural Networks (NN) which can be used as a supervised learning technique in loss modeling. The concept is to provide a vast amount of input/output data to train the model so that it can make predictions using new sets of inputs. A very large numbers of covariates can be processed by NN.

NN are part of the ML family but more specifically, it can involve deep learning. As a subset of ML, deep learning algorithms use multiple layers to learn from a vast amount of data. A NN is composed of an input layer, hidden layer(s) and an output layer. All layers contain neurons (also known as units or nodes) which take a set of inputs and transform it into a single number. Detailed explanations are provided in Section 2.2.1. All covariates are entered in the input layer and their values correspond to the neurons of the input layer. The output layer contains the prediction(s) of the model, so the number of neurons corresponds to the number of desired outputs. For example, if one is interested in assessing if a given risk would submit a claim, then there would be only one neuron in the output layer which would correspond to the probability of having a claim. A hidden layer can contain any number of neurons and if a model contains more than one hidden layer, they do not need to contain the same number of neurons.

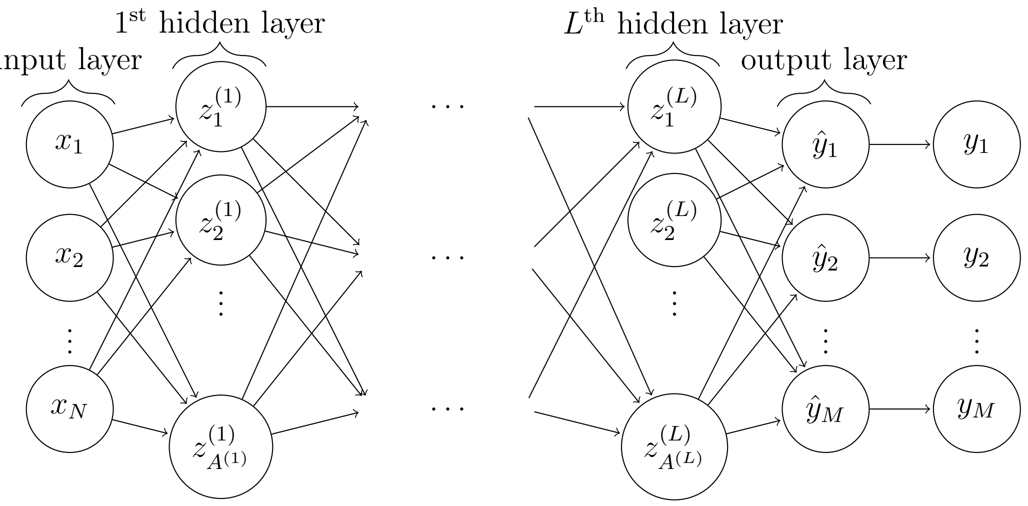


Figure 2.1: Multilayer Neural Network[4]

Figure 2.1 can be referred to for the remaining of Section 2.2. In the illustrated NN, there are N input covariates and M outputs to predict. It has L hidden layers and they can contain different number of hidden units, i.e. there are $A^{(j)}$ hidden units at layer j. For simplicity, we denote A as the number of hidden units in a given layer. Each element of the output layer corresponds to a predicted response.

[4]Adapted from https://davidstutz.de/illustrating-convolutional-neural-networks-in-latex-with-tikz/

2.2.1 General Process

Feedforward [5]

In this section, we will describe the process to go from the input layer to the output one. Suppose the model contains N covariates in input and only one hidden layer. Each neuron from the hidden layer is a weighted combination of the values from the input layer plus bias, as follows,

$$z_i^{(1)} = w_{i1}^{(0)} x_1 + w_{i2}^{(0)} x_2 + ... + w_{iN}^{(0)} x_N + b_i^{(0)}.$$

However, the neurons have to be activated. The value just obtained is then passed into a function σ, called *activation function*, which results in the activated neuron value. In hidden and output layers, activation functions evaluate the worthiness of each neuron (i.e. to which point a neuron's input $z_i^{(j)}$ is relevant to what the model is trying to predict) and most of them scale the neurons between 0 and 1 or between -1 and 1, depending of the function used.

The same process applies to the output layer, but instead of using the input layer in the weighted combination, the activated neurons from the hidden layer are used. Therefore, for the i^{th} activation unit on the j^{th} layer, $a_i^{(j)} = \sigma(z_i^{(j)})$.

More generally, if there are more than one hidden layer, $z_i^{(j+1)}$, the i^{th} neuron on the $(j+1)^{th}$ layer containing A units, is calculated as follows,

[5]From https://youtu.be/jqd3Bj0q2Sc explaining Goulet et al. (2020)

$$z_i^{(j+1)} = w_{i1}^{(j)} a_1^{(j)} + w_{i2}^{(j)} a_2^{(j)} + ... + w_{iA}^{(j)} a_A^{(j)} + b_i^{(j)}, \qquad (2.1)$$

where $i \in \{1, 2, ..., A\}$ and $j \in \{1, 2, ..., L-1\}$. It can be represented graphically by Figure 2.2, as follows,

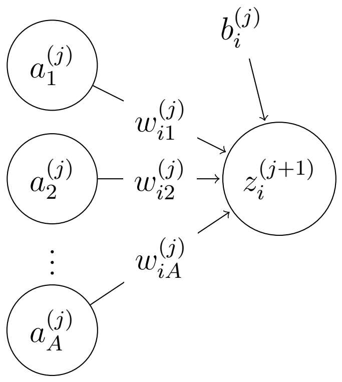


Figure 2.2: Network Graph of $z_i^{(j+1)}$ Calculations

Similarly, from (2.1), the values of the output layer from the last hidden layer L are obtained as follows,

$$g(y_i) = w_{i1}^{(L)} a_1^{(L)} + w_{i2}^{(L)} a_2^{(L)} + ... + w_{iA}^{(L)} a_A^{(L)} + b_i^{(L)}.$$

When there are more than one hidden layer, the activated neurons from the previous layer are used to calculate the activated neurons of the next layer, which is represented as follows,

$$a_i^{(j+1)} = \sigma(z_i^{(j+1)}) = \sigma(w_{i1}^{(j)} a_1^{(j)} + w_{i2}^{(j)} a_2^{(j)} + ... + w_{iA}^{(j)} a_A^{(j)} + b_i^{(j)}). \qquad (2.2)$$

Activation Functions

An article by Nwankpa et al. (2018) presents several activation functions and their current trends in applications in deep learning. The functions presented in this section are from that article. As discussed by Sharma and Sharma (2017), the Rectifided Linear Unit (ReLU) function, which is described in this section, is the most popular activation function and generally achieves better results. The Leaky ReLU (LReLU) is one alternative to ReLU that can improve ReLU's performance.

Linear Activation Functions

A linear activation function is of the form $f(x) = ax + b$, where a and $b \in \mathbb{R}$. Activation is proportional to the weighted sum of the neurons from the previous layer plus the current layer's bias as described in (2.1). The range of the output can be $(-\infty, \infty)$. Weights and biases are still updated during backpropagation, which will be presented and detailed later in Section 2.2.1. However, error does not improve from one iteration to the next, since it would be the same gradient value at each iteration (derivative of a linear function is a constant). Information about which weights would provide a better prediction is therefore not available. Furthermore, for any number of hidden layers in the NN, since a linear function is used, the last layer ends up also being a linear function of the input layer.

Non-Linear Activation Functions

Using non-linear activation functions, information about which weights would provide a better prediction is available, as more complex patterns can be identified from the data.

1. The Sigmoid Function

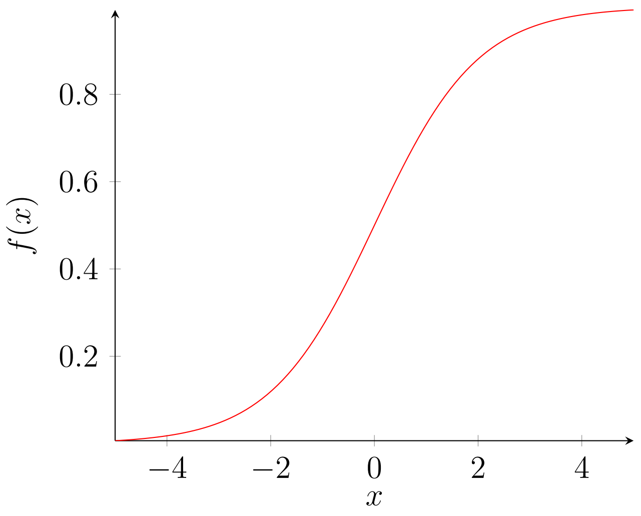


Figure 2.3: Sigmoid Function

The Sigmoid function is shown in Figure 2.3 and is represented as follows,

$$f(x) = \frac{1}{1 + e^{-x}}.$$

Since $f(x) \in (0, 1)$, the Sigmoid function can be used to predict probabilities of a defined event happening in binary classification problems, where the response is either 0 or 1. Given a specific threshold (commonly set as 0.5), if the output value falls above it, the prediction would be 1, whereas if it is below, the prediction would be considered as 0.

2. The Hyperbolic Tangent Function

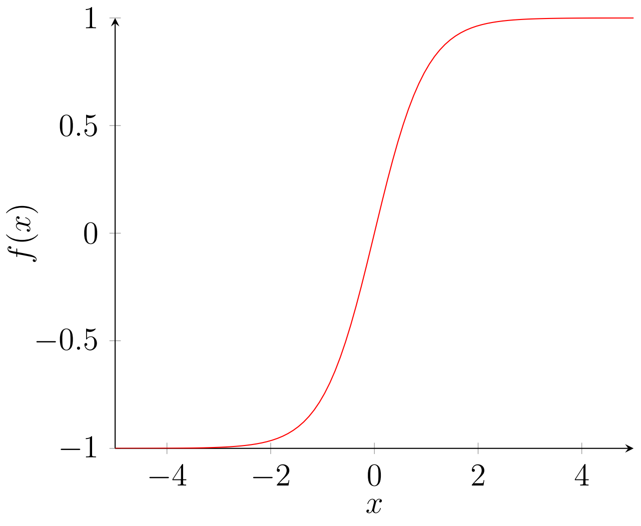


Figure 2.4: Hyperbolic Tangent Function

The Hyperbolic Tangent function is shown in Figure 2.4 and is represented as follows,

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

with $f(x) \in (-1, 1)$. As for the sigmoid function, the Hyperbolic Tangent function is also mainly used in binary classification problems, where the output is between 0 and 1. However, it is smoother than the sigmoid function and it is also centered to zero. Its main advantage over sigmoid is that the activated units are then zero centered which can help in the backpropagation process, which is presented and detailed later in Section 2.2.1.

3. The Softsign Function

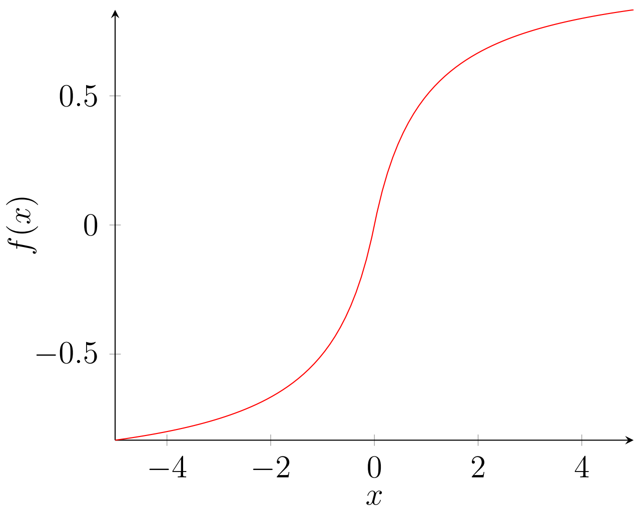


Figure 2.5: Softsign Function

The Softsign function is shown in Figure 2.5 and is represented as follows,

$$f(x) = \frac{x}{|x| + 1},$$

with $f(x) \in (-1, 1)$ as for the Hyperbolic Tangent function, which seems similar looking at the graphs. However, from the performance analysis by Farzad et al. (2019), Softsign, introduced by Turian et al. (2009), produces better results than common functions such as sigmoid and tanh for classification problems using Long Short-Term Memory (LSTM) neural networks. LSTMs, introduced by Hochreiter and Schmidhuber (1997), can process sequences of data such as speech and video.

4. The Rectified Linear Unit function (ReLU) Function

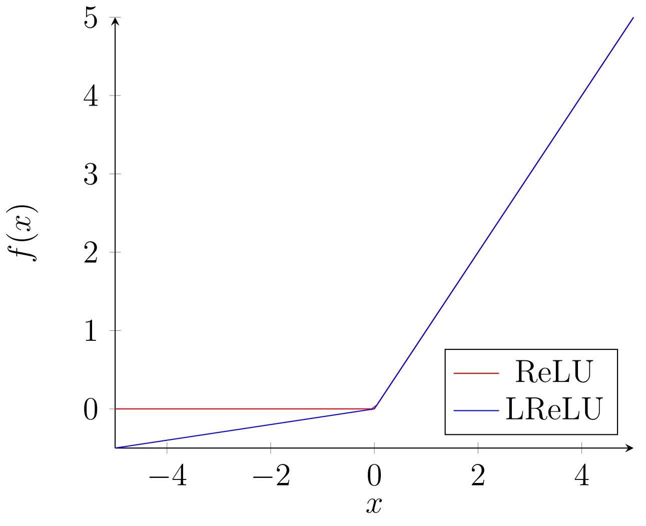


Figure 2.6: ReLU and LReLU

The Rectified Linear Unit function (ReLU) is shown in red in Figure 2.6 and is represented as follows,

$$f(x) = \max(0, x), \tag{2.3}$$

with $f(x) \in [0, \infty)$. This is the activation function that is mostly used. Computational speed is increased when using ReLU compared to other functions. It is a simple function that does not contain exponential nor divisions and neither does its derivative. However, since a neuron with a negative value becomes 0 after activation, it weakens training by causing subsequent neurons to be null and by preventing weight updates.

The Leaky ReLU (LReLU) was proposed as a solution to those drawbacks.

It is shown in blue in Figure 2.6 and is represented as follows,

$$f(x) = \begin{cases} \alpha x, & \text{if } x \leq 0 \\ x, & \text{if } x > 0, \end{cases}$$

where $\alpha \in \mathbb{R}$ is a very small value (typically around 0.01), which is enough to keep some small slope when $x < 0$ to avoid neurons with negative values to become 0 after activation.

5. The Softplus Function

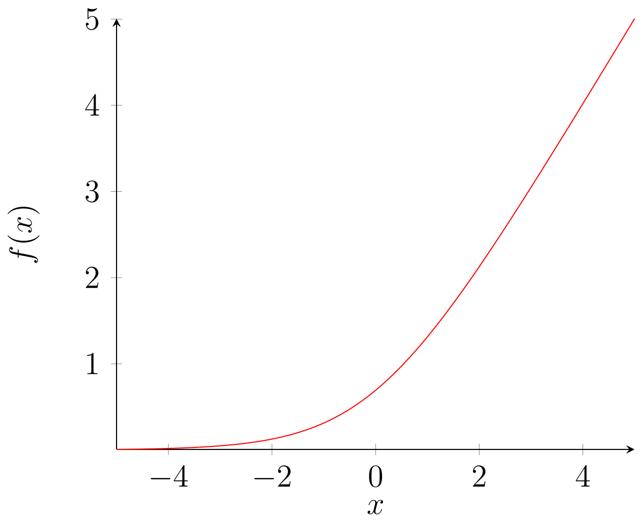


Figure 2.7: Softplus Function

The Softplus function is shown in Figure 2.7 and is represented as follows,

$$f(x) = \ln(1 + e^x),$$

with $f(x) \in [0, \infty)$. It is similar to the ReLU function, in a smoother version. It has been shown by Zheng et al. (2015) that the Softplus function provides better performance with less epochs in training compared to ReLU and Sigmoid functions.

Multivariate Functions

There also exist activation functions, such as the Softmax and the Maxout functions, which do not take a single "x" as input, but a whole vector corresponding to the neurons in the current layer.

The Softmax function is represented as follows,

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}},$$

where i and j are the i^{th} and j^{th} neurons on the current layer. It is used to solve categorical output variable problems. For instance, each $f(x_i) \in [0, 1]$ and their sum is 1, so $f(x_i)$ represents the probability of an observation being in the i^{th} category. The predicted output would then be the category that has the highest probability.

The Maxout function, proposed by Goodfellow et al. (2013), is represented as follows,

$$f(x) = \max(w_1 x + b_1, w_2 x + b_2).$$

It uses two sets (instead of one) of weights and bias to compute the activated units. More generally, we can have $f(x) = \max\limits_i x_i$, where i is the number of sets of parameters. However, using more sets of parameters would increase computational time and resources since it would multiply the number of parameters used in the network by i.

Error Measurement

Now, weights and biases can be first randomly generated but the objective is to optimize them such that the cost function is minimized to improve the accuracy of the model. The cost function results in a number representing how poor the accuracy of the model is by usually taking the prediction errors.

Many types of cost functions exist. Let $\hat{y}_i$ be the predicted value and y_i be its corresponding real value, where $i \in \{1, ..., N\}$. For regression problems, examples are the mean absolute error (MAE), the mean squared error (MSE) and the root mean squared error (RMSE) which are represented as follows,

$$\begin{aligned} MAE &= \frac{1}{N}\sum_{i=1}^{N} |\hat{y}_i - y_i|, \\ MSE &= \frac{1}{N}\sum_{i=1}^{N} (\hat{y}_i - y_i)^2, \\ RMSE &= \sqrt{\frac{1}{N}\sum_{i=1}^{N} (\hat{y}_i - y_i)^2}, \end{aligned}$$

where MAE represents the average absolute difference value between the N predicted estimates and their true value, whereas MSE calculates the average of the squared difference. RMSE is simply the square root of the MSE.

For classification problems, cross entropy is a cost function which is used in two cases. Binary cross entropy is used when there are two possible outcomes and is defined as follows,

$$-\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)].$$

If there are M possible categories, the cross entropy formula is as follows,

$$-\sum_{j=1}^{M}\sum_{i=1}^{N}[y_i \log(\hat{y}_i)].$$

Backpropagation

The cost function is evaluated after the first iteration (using the randomly generated weights and biases). The result is called the cost value. We can see the weights and biases as being parameters of the cost function, since they have an impact on it. Increasing or decreasing any given weight in the model would change the cost value. The same principle applies to biases. Therefore, calculating the partial derivative of each parameter in the model with respect to the cost function is called backpropagation. Once the derivatives are calculated, the algorithm which tries to find the minimum of the cost value function at each iteration is called the gradient descent algorithm. It allows to update the parameters using the partial derivatives (gradients) calculated during backpropagation. However, gradient descent can find stationary points which are not guaranteed to be local minimums, but generally points towards such minimums.

Optimal weights and biases are now found for the current iteration. At each iteration, we update parameters and repeat the process until convergence (when error is below a prescribed tolerance) or when the maximum number of epochs is reached. An epoch represents how many times the model would be trained using the same training set. It is set before starting the training. However, a high epoch value could lead to overfitting.

2.2.2 Literature Review of the Model: Tractable Approximate Gaussian Inference (TAGI)

Goulet et al. (2020) developed a method, called Tractable Approximate Gaussian Inference (TAGI), that calculates the distribution of each predicted response from a NN, which respond to the initial purpose of this thesis. Thus, for a given predicted response, the uncertainty around this prediction is Normal. Let $\boldsymbol{E} \sim \mathcal{N}(0, \boldsymbol{\Sigma_E})$ be the observations errors. In a NN with L hidden layers, where the output layer is the $(L+1)^{th}$, the observed responses $\boldsymbol{Y} \sim \mathcal{N}(\boldsymbol{\mu_Y}, \boldsymbol{\Sigma_Y})$, where $\boldsymbol{\mu_Y} = \boldsymbol{\mu_{Z^{(L+1)}}}$ and $\boldsymbol{\Sigma_Y} = \boldsymbol{\Sigma_{Z^{(L+1)}}} + \boldsymbol{\Sigma_E}$. Furthermore, all elements in TAGI follow a Normal distribution from which we can get the mean and variance for each of them. Moreover, diagonal covariance matrices are used instead of full matrices for computational efficiency purposes. Finally, layer-wise inference of hidden units and parameters is possible using one observation at a time.

In terms of computational efficiency and accuracy, TAGI matches the performance of state-of-the-art NN architectures using backpropagation for both regression and classification problems[6].

[6]Refer to results presented in Goulet et al. (2020).

Before providing detailed explanations on TAGI, some concepts used in that method will be discussed first for better understanding.

Background Information

Bayesian Neural Network

In a Bayesian NN, the weights and biases have probability distributions which is not the case in a standard NN framework, where weights and biases are point estimates that are commonly calculated using maximum-likelihood. The article by Jospin et al. (2020) illustrates the particularities of a Bayesian NN. Learning distributions for weights and biases allow to capture uncertainty around the parameters and outputs, which is a feature that is not possible in standard NN.

Let weights and biases be represented by $\boldsymbol{\theta}$. A prior distribution for the weights and biases, $p(\boldsymbol{\theta})$, is first defined. A common choice of prior distribution is $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ but other distributions can be used. Applying Bayes' Theorem in a NN context to get the posterior distribution is shown as follows,

$$p(\boldsymbol{\theta}|D) = \frac{p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}})} = \frac{p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}} p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta'})p(\boldsymbol{\theta'})\, d\boldsymbol{\theta'}},$$

where $D = \{D_{\boldsymbol{x}}, D_{\boldsymbol{y}}\}$ is the training set. However, computing the posterior is not possible using a standard NN. More precisely, the denominator $\int_{\boldsymbol{\theta}} p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta'})p(\boldsymbol{\theta'})\, d\boldsymbol{\theta'}$ cannot practically be computed as the search space of $\boldsymbol{\theta}$ is too large considering all possible combinations. Approaches exist to approximate the posterior. For example, variational inference is still used nowadays by Blundell et al. (2015) and consists in performing inference with a variational distribution, which is a distribution $q_\phi(\boldsymbol{\theta})$ with known form and parameters $\boldsymbol{\phi}$ that approximates the true $p(\boldsymbol{\theta}|D)$. $q_\phi(\boldsymbol{\theta})$ can follow any known distributions of simple form such as the Normal distribution, in which case $\boldsymbol{\phi}$ would include the mean and variance parameters.

Multivariate Normal Distribution

As described in Johnson et al. (2002), for a n-dimensional random vector $\boldsymbol{X} = (X_1, X_2, ..., X_n)$, $\boldsymbol{X}$ follows a Multivariate Normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_n \end{bmatrix}, \qquad \boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{11} & \dots & \sigma_{1n} \\ \vdots & \ddots & \vdots \\ \sigma_{n1} & \dots & \sigma_{nn} \end{bmatrix},$$

and where $\sigma_{i,j} = cov(X_i, X_j)$ is the covariance between X_i and X_j for $i, j \in \{1, ..., n\}$.

An interesting feature of the Multivariate Gaussian distribution is that $\boldsymbol{X}$ can be partitioned and that all subsets are normally distributed. For example, if $\boldsymbol{X}$ is partitioned in two, then $\boldsymbol{X}_1 \sim \mathcal{N}(\boldsymbol{\mu_1}, \boldsymbol{\Sigma_1})$ and $\boldsymbol{X}_2 \sim \mathcal{N}(\boldsymbol{\mu_2}, \boldsymbol{\Sigma_2})$.

Moreover, the conditional distribution of $\boldsymbol{X}_1$ given that $\boldsymbol{X}_2 = \boldsymbol{x}_2$, $\boldsymbol{X}_1|\boldsymbol{X}_2 = \boldsymbol{x}_2$, follows a Normal distribution with

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) \tag{2.4}$$

and

$$\boldsymbol{\Sigma}_{1|2} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}, \tag{2.5}$$

where $\boldsymbol{\Sigma}_{ij}$ is the covariance matrix of i and j.

Application in TAGI [7]

Recall that (2.1) provides the values of the neurons before activation of the layer $j+1$ using the activated units of layer j, with $A^{(j)}$ and $A^{(j+1)}$ as the number of neurons in layers j and $j+1$ respectively. The equation can be rewritten in matrix form as $\boldsymbol{Z} = \boldsymbol{W}\boldsymbol{A} + \boldsymbol{B}$, where $\boldsymbol{Z}$ is a $A^{(j+1)}$-vector representing the units to calculate, $\boldsymbol{W}$ is the weight $(A^{(j+1)} \times A^{(j)})$-matrix, $\boldsymbol{A}$ is a $A^{(j)}$-vector representing the activated units from the previous layer and $\boldsymbol{B}$ is a $A^{(j+1)}$-vector representing the bias.

Now, let $\boldsymbol{A} \sim \mathcal{N}(\boldsymbol{\mu_A}, \boldsymbol{\Sigma_A})$, $\boldsymbol{B} \sim \mathcal{N}(\boldsymbol{\mu_B}, \boldsymbol{\Sigma_B})$ and $\boldsymbol{W} \in \mathbb{R}^{A^{(j+1)} \times A^{(j)}}$. Using partitioning, the joint probability distribution $f(\boldsymbol{z}, \boldsymbol{a})$ is $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu_Z} \\ \boldsymbol{\mu_A} \end{bmatrix} \text{ and } \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma_Z} & \boldsymbol{\Sigma_{ZA}} \\ \boldsymbol{\Sigma_{AZ}} & \boldsymbol{\Sigma_A} \end{bmatrix},$$

where using basic properties of matrices we can obtain

$$\boldsymbol{\mu_Z} = \boldsymbol{W}\boldsymbol{\mu_A} + \boldsymbol{\mu_B},$$

$$\boldsymbol{\Sigma_Z} = \boldsymbol{W}\boldsymbol{\Sigma_A}\boldsymbol{W}^T + \boldsymbol{\Sigma_B},$$

$$\boldsymbol{\Sigma_{ZA}} = \boldsymbol{\Sigma_A}\boldsymbol{W}^T \text{ and } \boldsymbol{\Sigma_{AZ}} = \boldsymbol{\Sigma_{ZA}^T}.$$

We can also get the conditional distribution of $\boldsymbol{A}|\boldsymbol{Z} = \boldsymbol{z}$ which would be $\mathcal{N}(\boldsymbol{\mu_{A|z}}, \boldsymbol{\Sigma_{A|z}})$, where, using (2.4) and (2.5),

[7]From https://youtu.be/jqd3Bj0q2Sc explaining Goulet et al. (2020)

$$\boldsymbol{\mu_{A|z}} = \boldsymbol{\mu_A} + \boldsymbol{\Sigma_{AZ}}\boldsymbol{\Sigma_Z^{-1}}(\boldsymbol{z} - \boldsymbol{\mu_Z}),$$

$$\boldsymbol{\Sigma_{A|z}} = \boldsymbol{\Sigma_A} - \boldsymbol{\Sigma_{AZ}}\boldsymbol{\Sigma_Z^{-1}}\boldsymbol{\Sigma_{ZA}}.$$

Now, it can be used in Bayesian NN, where weights also follow a probability distribution. Let $\boldsymbol{W} \sim \mathcal{N}(\boldsymbol{\mu_W}, \boldsymbol{\Sigma_W})$. Since there are now three random variables, the joint probability distribution $f(\boldsymbol{z}, \boldsymbol{a}, \boldsymbol{w})$ is of interest. However, that distribution cannot be Multivariate Gaussian because of $\boldsymbol{WA}$. Multiplying two Gaussian distributions does not result in a Gaussian joint distribution, but in a Chi-Square distribution. Therefore, the closed-form with (2.4) and (2.5) cannot be used to get the conditional distribution, which corresponds to the posterior probability distribution $p(\boldsymbol{\theta}|D)$.

Product of Two Gaussian Random Variables

Suppose that $X \sim \mathcal{N}(\mu_X, \sigma_X^2)$ and $Y \sim \mathcal{N}(\mu_Y, \sigma_Y^2)$, where the two Gaussian random variables X and Y are independent. We can write the product XY as follows,

$$XY = \tfrac{1}{4}((X+Y)^2 - (X-Y)^2)) = (\tfrac{X+Y}{2})^2 - (\tfrac{X-Y}{2})^2,$$

which follows a generalized Chi-Square distribution since it is a linear combination of noncentral Chi-Square distributions with one degree of freedom each.

If we are interested in the distribution of the sum of product of two Gaussian random variables, it is now equivalent to find the distribution of the sum of Chi-Square random variables. Suppose there are n independent Chi-Square random variables (i.e. n products of two Gaussian random variables) denoted as $X_1, X_2, ..., X_n$ with mean μ and variance σ^2. Central limit theorem

(CTL) states that, as $n \to \infty$, their normalized sum converges to a Normal distribution.

2.2.2.1 Concepts and Propositions in TAGI

We would like to approximate the distribution of WA by a Normal distribution, where $W \sim \mathcal{N}(\mu_W, \sigma_W^2)$ and $A \sim \mathcal{N}(\mu_A, \sigma_A^2)$. In the case discussed in Section 2.2.2, the sum of many product terms of two Normal distributions is used to calculate the hidden units as illustrated in (2.1). As a solution, the authors considered Gaussian multiplication approximation (GMA), which they define as "the approximation of the probability density function (PDF) for any product term X_iX_j by a Gaussian whose first two moments are defined by" (2.6) to (2.9), which are as follows,

$$\mathbb{E}[X_1X_2] = \mu_1\mu_2 + \mathrm{cov}(X_1, X_2), \tag{2.6}$$

$$\mathrm{cov}(X_3, X_1X_2) = \mathrm{cov}(X_1, X_3)\mu_2 + \mathrm{cov}(X_2, X_3)\mu_1, \tag{2.7}$$

$$\begin{aligned}\mathrm{cov}(X_1X_2, X_3X_4) = {} & \mathrm{cov}(X_1, X_3)\mathrm{cov}(X_2, X_4) \\ & + \mathrm{cov}(X_1, X_4)\mathrm{cov}(X_2, X_3) \\ & + \mathrm{cov}(X_1, X_3)\mu_2\mu_4 + \mathrm{cov}(X_1, X_4)\mu_2\mu_3 \\ & + \mathrm{cov}(X_2, X_3)\mu_1\mu_4 + \mathrm{cov}(X_2, X_4)\mu_1\mu_3,\end{aligned} \tag{2.8}$$

$$\begin{aligned}\mathrm{var}(X_1X_2) = {} & \sigma_1^2\sigma_2^2 + \mathrm{cov}(X_1, X_2)^2 + 2\mathrm{cov}(X_1, X_2)\mu_1\mu_2 \\ & + \sigma_1^2\mu_2^2 + \sigma_2^2\mu_1^2.\end{aligned} \tag{2.9}$$

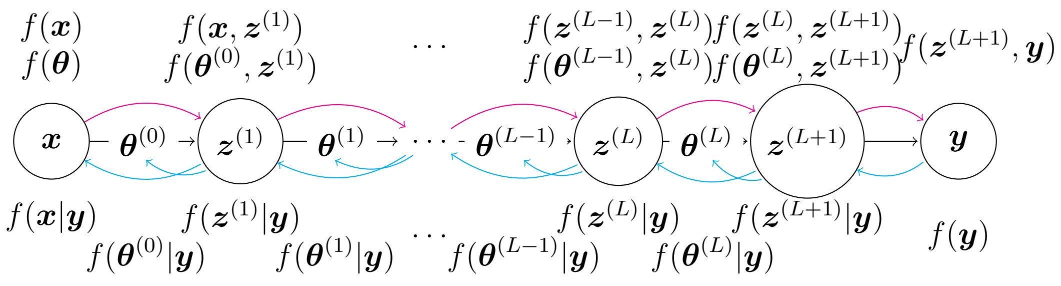
As the number of product terms increases, the distribution of the sum of those product terms (resulting from GMA) tends to follow a Normal distribution, as expected from CTL. (2.6) to (2.9) are valid when the random variables are Normal and were derived using moment generating functions[8]. They will be used in Section 3.

Now that the hidden units can be calculated using GMA, we would like to carry their uncertainty in the activation phase. However, applying a non-linear activation function to a random variable does not propagate its uncertainty. For its low computational cost, performance and usability with common activation functions described in Section 2.2.1, the authors propose a linearization procedure $\tilde{\sigma}(\cdot)$, where (2.2) is approximated as follows,

$$a = \sigma(z) \approx \tilde{\sigma}(z) = \sigma(\mu_Z) + \frac{\partial \sigma(\mu_Z)}{\partial z}(z - \mu_Z), \tag{2.10}$$

where the linearization is done at μ_Z.

The authors suggest to perform tractable recursive layer-wise inference which is possible using the inherent conditional independence concept, where $\boldsymbol{Z}^{(j-1)}$ is statistically independent of $\boldsymbol{Z}^{(j+1)}$ given $\boldsymbol{z}^{(j)}$. Tractable recursive layer-wise inference is represented as follows,



$f(\boldsymbol{x})$
$f(\boldsymbol{\theta})$
$f(\boldsymbol{x}, \boldsymbol{z}^{(1)})$
$f(\boldsymbol{\theta}^{(0)}, \boldsymbol{z}^{(1)})$
$\cdots$
$f(\boldsymbol{z}^{(L-1)}, \boldsymbol{z}^{(L)})f(\boldsymbol{z}^{(L)}, \boldsymbol{z}^{(L+1)})$
$f(\boldsymbol{\theta}^{(L-1)}, \boldsymbol{z}^{(L)})f(\boldsymbol{\theta}^{(L)}, \boldsymbol{z}^{(L+1)})$
$f(\boldsymbol{z}^{(L+1)}, \boldsymbol{y})$
$\boldsymbol{x}$
$\boldsymbol{\theta}^{(0)}$
$\boldsymbol{z}^{(1)}$
$\boldsymbol{\theta}^{(1)}$
$\cdots$
$\boldsymbol{\theta}^{(L-1)}$
$\boldsymbol{z}^{(L)}$
$\boldsymbol{\theta}^{(L)}$
$\boldsymbol{z}^{(L+1)}$
$\boldsymbol{y}$
$f(\boldsymbol{x}|\boldsymbol{y})$
$f(\boldsymbol{\theta}^{(0)}|\boldsymbol{y})$
$f(\boldsymbol{z}^{(1)}|\boldsymbol{y})$
$f(\boldsymbol{\theta}^{(1)}|\boldsymbol{y})$
$\cdots$
$f(\boldsymbol{\theta}^{(L-1)}|\boldsymbol{y})$
$f(\boldsymbol{z}^{(L)}|\boldsymbol{y})$
$f(\boldsymbol{\theta}^{(L)}|\boldsymbol{y})$
$f(\boldsymbol{z}^{(L+1)}|\boldsymbol{y})$
$f(\boldsymbol{y})$

Figure 2.8: Tractable Recursive Layer-Wise Inference[9]

[8] Please refer to Appendix A in Goulet et al. (2020) for more details.

For two given consecutive layers, let $\{\boldsymbol{\theta}, \boldsymbol{Z}\} = \{\boldsymbol{\theta}^{(j)}, \boldsymbol{Z}^{(j)}\}$ and $\{\boldsymbol{\theta}^{+}, \boldsymbol{Z}^{(+)}\} = \{\boldsymbol{\theta}^{(j+1)}, \boldsymbol{Z}^{(j+1)}\}$ represent the sets of parameters and hidden units for the current layer and the next one respectively. Using the Rauch-Tung-Striebel recursive procedure by Rauch et al. (1965), $f(\boldsymbol{\theta}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}_{\boldsymbol{\theta}|\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}|\boldsymbol{y}})$, where

$$\boldsymbol{\mu}_{\boldsymbol{\theta}|\boldsymbol{y}} = \boldsymbol{\mu}_{\boldsymbol{\theta}} + \boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{\mu}_{\boldsymbol{Z}^{+}|\boldsymbol{y}} - \boldsymbol{\mu}_{\boldsymbol{Z}^{+}}), \tag{2.11}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{\theta}|\boldsymbol{y}} = \boldsymbol{\Sigma}_{\boldsymbol{\theta}} + \boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{\Sigma}_{\boldsymbol{Z}^{+}|\boldsymbol{y}} - \boldsymbol{\Sigma}_{\boldsymbol{Z}^{+}})\boldsymbol{J}_{\boldsymbol{\theta}}^{\boldsymbol{T}}, \tag{2.12}$$

$$\boldsymbol{J}_{\boldsymbol{\theta}} = \boldsymbol{\Sigma}_{\boldsymbol{\theta}\boldsymbol{Z}^{+}}\boldsymbol{\Sigma}_{\boldsymbol{Z}^{+}}^{\boldsymbol{-1}}, \tag{2.13}$$

and where $f(\boldsymbol{z}|\boldsymbol{y})$ is $\mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{Z}|\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{Z}|\boldsymbol{y}})$ with

$$\boldsymbol{\mu}_{\boldsymbol{Z}|\boldsymbol{y}} = \boldsymbol{\mu}_{\boldsymbol{Z}} + \boldsymbol{J}_{\boldsymbol{Z}}(\boldsymbol{\mu}_{\boldsymbol{Z}^{+}|\boldsymbol{y}} - \boldsymbol{\mu}_{\boldsymbol{Z}^{+}}), \tag{2.14}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{Z}|\boldsymbol{y}} = \boldsymbol{\Sigma}_{\boldsymbol{Z}} + \boldsymbol{J}_{\boldsymbol{Z}}(\boldsymbol{\Sigma}_{\boldsymbol{Z}^{+}|\boldsymbol{y}} - \boldsymbol{\Sigma}_{\boldsymbol{Z}^{+}})\boldsymbol{J}_{\boldsymbol{Z}}^{\boldsymbol{T}}, \tag{2.15}$$

$$\boldsymbol{J}_{\boldsymbol{Z}} = \boldsymbol{\Sigma}_{\boldsymbol{Z}\boldsymbol{Z}^{+}}\boldsymbol{\Sigma}_{\boldsymbol{Z}^{+}}^{\boldsymbol{-1}}. \tag{2.16}$$

Therefore, for a model with L hidden layers, parameters are not updated using backpropagation, but through the inference procedure using (2.11) to (2.16), which is summarized as follows,

[9] Adapted version of Figure 6(b) in Goulet et al. (2020)

Algorithm 2.1 Inference Procedure

```
1: for ($j$ in $(L+1):0$)
2: Calculate and store $\boldsymbol{\mu}_{\boldsymbol{Z}^{(j)}|\boldsymbol{y}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{Z}^{(j)}|\boldsymbol{y}}$
3: Calculate $\boldsymbol{\mu}_{\boldsymbol{\theta}^{(j)}|\boldsymbol{y}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}^{(j)}|\boldsymbol{y}}$
4: Update parameters' distribution $\boldsymbol{\theta}^{(j)}$ using $\boldsymbol{\mu}_{\boldsymbol{\theta}^{(j)}|\boldsymbol{y}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}^{(j)}|\boldsymbol{y}}$
```

2.3 Gradient Boosting

Another popular supervised ML technique is Gradient Boosting (GB), which is mainly known for its predictive accuracy. In this section, details and explanations on this technique and one of its famous implementation, Extreme Gradient Boosting (XGBoost), are provided. In auto insurance, GB trees, which are described in Section 2.3.1, are used for loss cost modeling and prediction by Guelman (2012) and XGBoost for fraud detection by Dhieb et al. (2019).

2.3.1 Gradient Boosting Algorithm

First, boosting is a technique converting weak learners into strong ones, where weak learners are defined as learners that are slightly better at predictions than randomly guessing. Using decisions trees, an initial one is generated based on the initial data and then subsequent ones are grown from previous trees, which makes boosting a sequential process. The objective is to improve the accuracy of predictions from trees to trees since each tree is improved based on its previous version.

Since decision trees are an important concept used in GB, we first provide detailed explanations about them. However, please note that they consist in a classification and regression technique that was developed independently of GB. Decision trees can be solely used, but they are included in some GB algorithms.

Decision Trees

Classification and Regression Trees (CART), introduced by Breiman et al. (1984), are a supervised learning technique that can be used to predict categorical and numerical responses respectively.

For example, let a and b be continuous variables and c be a discrete variable which can take three values ($\{c_1, c_2, c_3\}$). A decision tree trying to predict a binary variable y can be as follows,

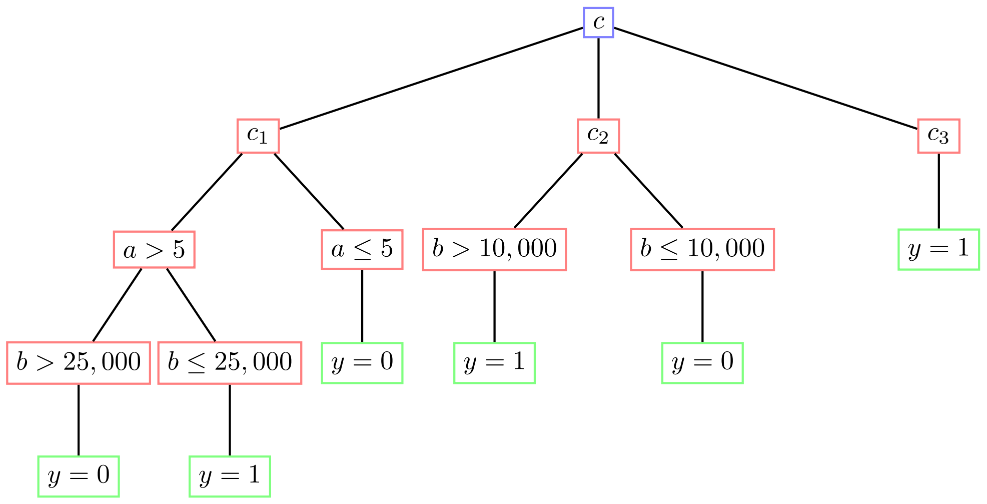


Figure 2.9: Classification Tree Example

A decision tree must be read from its root at the top to the bottom and is composed of nodes, which are subsets of previous nodes. In Figure 2.9, the root node is in blue and the red nodes are called decision nodes. Each node has two or more branches which represent a split among the possible values at that node. For example, the split at the c_1 node is done with the variable a, whereas it is done with variable b at the c_2 node and no split is observed at the c_3 node. At each node, the resulting split provides the most accurate prediction of the response variable. At the c_1 node, given the observations that are in the c_1 category, the clearest distinction between those which $y = 0$ to $y = 1$ occurs with a split at $a = 5$. For the c_2 category, it occurs when the data is divided between whether b is below or above 10,000.

When a node cannot be split into further nodes (reasons to stop the splitting nodes process are explained later in the current section), it leads to the leaf node (represented in green in Figure 2.9). It is where a decision about the prediction is made. In the example, the y prediction is assessed given the number of observations with $y = 0$ and $y = 1$ at the end of each tree path. Since this is a binary problem, if the majority of the observations have $y = 0$, then the leaf would be $y = 0$ (the same principle applies if the majority is $y = 1$). If there are more than 2 possible categories, than the leaf would take the value of the most important resulting category among the observations. In the case of a regression tree, where the response variable is numerical, the leafs correspond to the mean value of the response variable among the observations at the last node.

Splitting at Nodes

Now, we provide further explanations on when and how the splitting process is stopped. First, if all observations in a node have the same response (i.e.

if they all belong to the same category to predict), then there is no need to split them, since it is a pure node. Second, the splitting process stops if there is no more input variables to consider at that point. Third, there are some parameters which can be set when building a tree such as maximum depth, minimum number of training observations in each leaf, minimum number of training observations for a node split, maximum number of leaves and maximum variables to consider for a split. The depth of a tree is defined as the length of the longest path from the root to a leaf. Setting a maximum depth stops the splitting process for every paths that attain the specified length. Maximum depth along with the other parameters allow to control the size of the tree.

At each node, the best possible split is evaluated based on a criterion. Two commonly used criteria are the Gini index by Breiman et al. (1984) and entropy by Quinlan (1986), which are calculated as follows,

$$\begin{aligned} Gini &= 1 - \sum_{j=1}^{K} p_j^2, \\ Entropy &= -\sum_{j=1}^{K} p_j log_2 p_j, \end{aligned}$$

where there are K possible categories to the response variable and p_j is the proportion of observations in class j among the observations at that node.

Both indices can be used to calculate the information gain which is the measure that determines which variable is split at a node and at which threshold.

At a given node, there is the set S of observations, so let $l(S)$ be the impurity measure at that node (calculated either with Gini or Entropy). To get the informational gain for an attribute A, we compute the gain $G(S, A)$ as follows,

$$G(S, A) = l(S) - \sum_{i=1}^{C} p_c l(c),$$

where $l(c)$ is the impurity measure at potential child node c among C possible number of child nodes and p_c is the corresponding proportion of observations that fit in that child node. The information gain is calculated for any attribute A. Therefore, the split at a node corresponds to the one that maximizes the information gain.

Advantages and Disadvantages

The advantages of using CART are that the models are easy to understand and interpret. Data preparation is relatively easy since decision trees handle categorical and numerical data, they are not too sensible to extreme data/outliers and they can work with missing data. They also implicitly contribute to feature selection and detect interactions between variables.

However, those models are less accurate when they are used for classification problems with many categories. Also, if one category prevails over the data, it can create biased trees, so it is recommended to use a balanced training set. Furthermore, those models are subject to overfitting by creating very complex trees if parameters are not well controlled, such as the depth and minimal leaf size parameters.

Gradient Boosting with Trees

Let $\boldsymbol{x} = \{x_1, ..., x_p\}$ be a set of p covariates and $\rho^{(m)} > 0$ be a multiplier which acts as a learning rate. With a training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ and M iterations, the following is a simplified version of the gradient tree boosting algorithm for regression.

Algorithm 2.2 Simplified Gradient Tree Boosting Algorithm for Regression

```
1: Build a initial tree based on all data in the training set, where the variable
   to predict is $\hat{y}_i$.
2: for ($m$ in $1 : M$)
3: Calculate the residuals: $r_i^{(m)} = y_i - \hat{y}_i$
4: Take a random sample from remaining observations to build a tree $\hat{r}_i^{(m)}$
   from which the variable to predict is the residuals $r_i^{(m)}$'s using the $\boldsymbol{x}_i$'s
   as covariates.
5: Update the model by combining that tree with the others with multipli-
   ers: $\hat{y}_i^{(m)} = \hat{y}_i^{(m-1)} + \rho^{(m)}\hat{r}_i^{(m)}$
```

Gradient Boosting: General Case

In every GB problems, the objective is to minimize the loss function. Taking the derivative of the loss function with respect to the predictions would give insights on how the predictions can be adjusted to maximize the loss. Since the objective is to minimize it, gradient descent defined in Section 2.2.1 is used to get the negative gradient $-\left[\frac{\partial L(y_i, F(\boldsymbol{x}_i)}{\partial F(\boldsymbol{x}_i)}\right]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})}$. Thus, for a loss function L which includes both the real value y and its corresponding prediction F, the negative of the derivative of that function with respect to F

is called a pseudo-residual. An example with the squared error loss function is illustrated in Algorithm 2.4.

With a training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, M iterations and a differientiable loss function $L(y, F)$, the generic gradient boosting algorithm discussed in Friedman (2001) for regression is as follows,

Algorithm 2.3 Generic Gradient Boosting Algorithm for Regression

```
1: Initialize the model: $F_0(\boldsymbol{x}) = \underset{\rho}{\text{argmin}} \sum_{i=1}^n L(y_i, \rho)$.
2: for ($m$ in $1:M$)
3: Calculate the pseudo-residuals: $\tilde{y}_i = -[\frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})}$ for $i = 1, ..., n$
4: Fit a base learner $h(\boldsymbol{x}_i; \boldsymbol{a})$ to the pseudo-residuals using the training set $\{(\boldsymbol{x}_i, \tilde{y}_i)\}_{i=1}^n$ and compute multiplier: $(\boldsymbol{a}_m, \rho_m) = \underset{\boldsymbol{a},\rho}{\text{argmin}} \sum_{i=1}^n L(y_i, F_{m-1}(\boldsymbol{x}_i) + \rho h(\boldsymbol{x}_i; \boldsymbol{a}_m))$
5: Update the model: $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m)$
```

where $h(\boldsymbol{x}; \boldsymbol{a})$ is a parameterized function of the covariates $\boldsymbol{x}$ with parameters $\boldsymbol{a} = \{a_1, a_2...\}$. In the case of trees, $h(\boldsymbol{x}; \boldsymbol{a})$ would be an individual tree and $\boldsymbol{a} = \{a_1, a_2...\}$ would be its structure and components (split of variables and thresholds and terminal node means).

A common loss criterion is the least-squares (LS), where the objective is to minimize the sum of squares of the error terms. The squared error loss function is $L(y, F) = \frac{(y-F)^2}{2}$ and the pseudo-residuals are the error terms, $\tilde{y}_i = y_i - F_{m-1}(\boldsymbol{x})$. The generic algorithm is modified as follows:

Algorithm 2.4 Gradient Boosting Algorithm for Least-Squares Regression

```
1: Initialize the model: $F_0(\boldsymbol{x}) = \bar{y}$.
2: for ($m$ in $1 : M$)
3: Calculate the pseudo-residuals: $\tilde{y}_i = y_i - F_{m-1}(\boldsymbol{x})$ for $i = 1, ..., n$
4: Fit a base learner $h(\boldsymbol{x}_i; \boldsymbol{a})$ to the pseudo-residuals using the training set
   $\{(\boldsymbol{x}_i, \tilde{y}_i)\}_{i=1}^n$ and compute the multiplier: $(\boldsymbol{a}_m, \rho_m) = \underset{\boldsymbol{a},\rho}{\operatorname{argmin}} \sum_{i=1}^n [\tilde{y}_i -$
   $\rho h(\boldsymbol{x}_i; \boldsymbol{a})]^2$
5: Update the model: $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m)$
```

Advantages and Disadvantages

The most important feature of GB is the prediction accuracy that can be reached. It also shares some of the decision tree advantages. No data imputation is required since it handles missing values. Categorical and numerical variables are also treated by those models.

However, outliers in the data have a negative effect on the model training. Since the model trains on errors, there is a high risk of overfitting. Also, GB models are computationally expensive in the number of generated trees over all iterations and in the search of the best features that minimize the loss at each step of the processes. Lastly, this type of model is not easy to interpret, especially when comparing to decision trees which are interpretable simply by looking at the trees with their nodes and branches illustrating the possible paths to the predictions.

2.3.2 Extreme Gradient Boosting

Extreme Gradient Boosting (XGBoost), developed by Chen and Guestrin (2016), is a specific implementation of GB which contains improved features, from which computational speed may be the most important one. It is available in many programming languages such as R and Python. As in standard GB, the first-order gradients on the loss function are used. XGBoost also computes the second-order gradients which provides additional information on how to minimize the loss function. Moreover, L1 and L2 regularizations are implemented which improves the models.

L1 and L2 Regularizations

L1 regularization corresponds to the Lasso (Least Absolute Shrinkage and Selection Operator) regression. It adds a penalty term to the loss function using the absolute value of weights. For example, let $\boldsymbol{x} = \{x_1, ..., x_p\}$ be a set of p covariates, β_j be the weight associated with the j^{th} covariate and $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be the training set. If the loss function is represented by $\sum_{i=1}^n (Y_i - \sum_{j=1}^p X_{ij}\beta_j)^2$, then $\lambda \sum_{j=1}^p |\beta_j|$ is added to the function, where $\lambda > 0$ is the regularization parameter. Using gradient descent, differentiating the loss function with respect to the weights allows to adjust them and to remove the less important features by shrinking their weight to zero. L1 regularization is useful for feature selection.

L2 regularization corresponds to the Ridge regression. It adds a penalty term to the loss function by squaring the weights. For example, if the loss function is represented by $\sum_{i=1}^n (Y_i - \sum_{j=1}^p X_{ij}\beta_j)^2$, then $\lambda \sum_{j=1}^p \beta_j^2$ is added to the function. L2 regularization keeps all the variables, estimates their importance and penalizes those that are not important, which contribute in reducing overfitting in the models.

Computational Speed

In GB, the splitting step can be time consuming considering that the algorithm evaluates the information gain of every possible features. It can result in an extensive phase if there are thousands of them. There is a histogram-based algorithm in XGBoost which allows, for all possible features in a node, to split the data into bins which reduces the time spent on searching for a split value.

Parameters

The package "xbgoost" by Chen et al. (2020) offers many features. Hyper-parameters and parameters values can be specified in inputs which prevents overfitting and reduces the number of computations. There are two types of learners, called boosters, which are tree booster (gbtree) or linear booster (gblinear). The first one generates trees whereas the second one uses linear functions. Gbtree performs better than gblinear when there are non-linear relationships in the data and/or when interactions are unknown. Both can be used for classification and regression problems. Known for its predictive accuracy, XGBoost generally well fits training data compared to many other methods. For example, using gblinear as booster is different than using linear regression and can provide better accuracy. XGBoost uses coordinate descent (optimization of one parameter at a time) instead of standard gradient descent (optimization of all parameters at once). However, XGBoost is more prone to overfitting than linear regression. To avoid overfitting, XGBoost uses a learning rate parameter to slow down learning at each iteration. For both boosters, the maximum number of iterations and the use of L1 and/or L2 regularizations can be set. Specific to the tree booster, parameters specific to trees, such as maximum depth, number of observations and/or features supplied to a tree, are available.

Moreover, a choice of metrics to evaluate the model's accuracy is available. For regression problems, the default error function is Root Mean Square Error (RMSE), but Mean Absolute Error (MAE) is also available. For classification problems, the default error function is the binary classification error rate which is the proportion of failed predictions. Multiclass classification error rate, negative log-likelihood and multiclass logloss are also available.

Multiclass classification error rate is also available, which is the same as binary classification error but for cases where there are more than two possible classes.

Negative Log-Likelihood

Since the objective is to improve the model accuracy from an iteration to the next, we want to increase likelihood. Let the model have only two possible classes for the response variable. It can be represented as a binary classification problem with $y_i = 1$ if the prediction is accurate for observation i. Otherwise, $y_i = 0$. The likelihood $L(\theta)$ of a Bernoulli distribution for n observations is represented as follows,

$$L(\theta) = \prod_{i=1}^{n} \theta_i^{y_i}(1-\theta_i)^{1-y_i},$$

where θ_i is the probability calculated by the model of predicting the appropriate class for observation i. To make the optimization of the likelihood easier, log-likelihood (LL) is often used, since it easier to work with sums than products. Log-likelihood, $l(\theta)$, is represented as follows,

$$l(\theta) = \sum_{i=1}^{n} [y_i \log \theta_i + (1 - y_i) \log(1 - \theta_i)].$$

Maximizing LL is equivalent to maximizing likelihood since the logarithmic function is strictly increasing. However, error (or loss) functions should be minimized. We can use Negative log-likelihood (NLL), which is the negative of the log-likelihood, $-l(\theta)$, since minimizing NLL is equivalent to maximizing LL.

Multiclass Log Loss

Multiclass log loss is the same as NLL, but for problems where there are more than two classes. For a classification problem with m possible categories, multiclass log loss is represented as follows,

$$-l(\theta) = -\sum_{i=1}^{n} \sum_{j=1}^{m} y_{ij} \log \theta_{ij},$$

where y_{ij} is the binary indicator that assesses if class j is the appropriate one for observation i and θ_{ij} is the corresponding probability.

2.4 Comparaison between TAGI and XGBoost

We apply TAGI and XGBoost to an insurance dataset by Lantz (2013). It contains 1,338 observations with seven variables and no missing data. Table 2.1 provides a description of the variables, as follows,

Variable	Type	Description
Age	Numerical	Age of the insured
Sex	Categorical	Gender of the insured: male or female
BMI	Numerical	Body Mass Index (kg/m^2) of the insured
Children	Numerical	Number of children covered as dependents
Smoker	Categorical	Smoking status: "yes" or "no"
Region	Categorical	Insured's residential US area: northeast, southeast, southwest, northwest.
Charges	Numerical	The response variable: Medical costs

Table 2.1: Description of Medical Cost Personal Dataset

The variable to predict is the medical costs ("charges"). Since TAGI does not take directly categorical nor string variables as inputs, "sex" and "smoker" variables are transformed into binary variables based on whether the insured is a female and a smoker respectively. For "region" variable, we use one-hot encoding, which means that we transform each possible value into a binary variable. With that data preparation step, there are now nine covariates.

For TAGI, models are trained over 40 epochs and one observation at the time (i.e. batch size is one). 80% of the observations are used for the training set. Data is normalized. The activation function used is the ReLU defined by (2.3). The initial prior distribution of the biases is $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma_B})$, where $\boldsymbol{\Sigma_B} = 0.01 \times \boldsymbol{I}$ and the initial prior distribution of the weights is $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma_W})$, where $\boldsymbol{\Sigma_W}$ is obtained by multiplying a factor 0.25 to the Xavier's approach by Glorot and Bengio (2010). Xavier's approach consists in initializing the weights as $\mathcal{N}(0, \frac{1}{n^{(j-1)}})$ or as $\mathcal{N}(0, \frac{2}{n^{(j-1)}+n^{(j)}})$, where $n^{(j-1)}$ and $n^{(j)}$ represent the number of units in the previous and current layers respectively. The models have one hidden layer of 50 and 100 units respectively, as they are common models to use. We generated 20 of each, which means that they are all trained using a different training set which was randomly selected among data for each model.

For the Extreme Gradient Boosting (XGBoost) technique, as it is a regression problem, we test the tree and linear boosters, described in Section 2.3.2, with their default parameters, except for the number of rounds which corresponds to the number of boosting iterations. For both techniques, we try with 40 rounds (as it can be comparable to the 40 epochs used in TAGI) and with optimization of the number of rounds using the inbuilt xgb.cv function based on the RMSE, which is defined in Section 2.2.1. Therefore, the optimized number of rounds is the round which minimizes RMSE at validation step, so the parameters at that specific round are used on the testing set. The average RMSE and its variance for each type of model (since for each of them, 20 models were generated using different training sets) is shown in Table 2.2.

Model	Root Mean Square Error (RMSE)
TAGI with L = 50	4,904.40 ± 576.34
TAGI with L = 100	4,607.99 ± 358.08
gbtree with nrounds=40	5011.28 ± 389.45
gbtree with optimized nrounds	4,781.26 ± 385.45
gblinear with nrounds=40	6,397.32 ± 348.49
gblinear with optimized nrounds	6,047.67 ± 457.74

Table 2.2: Medical Cost Personal Dataset Results with XGBoost

Comparing the two TAGI models, not surprisingly, the model with the hidden units containing 100 units performs better, since the model is more complex with more hidden units. However, a model with too many hidden units or layers can result in overfitting. In our case, it did not seem to have happened since it improved RMSE.

Comparing XGBoost models, gblinear performs very poorly compared to gbtree, which indicates that there are non-linear relationships in the data. Compared to TAGI, using gbtree with the same number of iterations, TAGI results in a smaller RMSE, but with optimized number of rounds, gbtree performs better than the model with 50 hidden units, but does not outperform the model with 100 hidden units.

3 Partial Derivatives [10]

Sensitivity analysis can be used when there exists uncertainty in a given model from any fields of study, such as finance, biology, meteorological, engineering, etc. Sensitivity analysis consists in studying the relationships between inputs and outputs by testing how changes to inputs affect model outputs.

Partial derivatives can be used as "local" sensitivity measures to understand the impact of a change on a single input to the model outputs. Let $\boldsymbol{X} = (X_1, X_2, ..., X_n)$ be the vector of inputs for a given model. Denote the output $Y = g(\boldsymbol{X})$ as a function of its input factors. The partial derivative of Y with respect to X_i is represented by $\dfrac{\partial g(\boldsymbol{X})}{\partial X_i}$, where $i \in \{1, 2, ..., n\}$.

However, using partial derivatives as sensitivity measures does not fully capture the interactions and dependence between input variables. This drawback is addressed in a new technique called cascade sensitivity by Pesenti et al. (2020).

Since NN can be used as a predictive model, partial derivatives can be calculated. Recall that outputs are linked to inputs through neurons in hidden layers. Details are provided through the following example of a NN with two hidden layers.

[10]In collaboration with J.-A. Goulet and L. H. Nguyen.

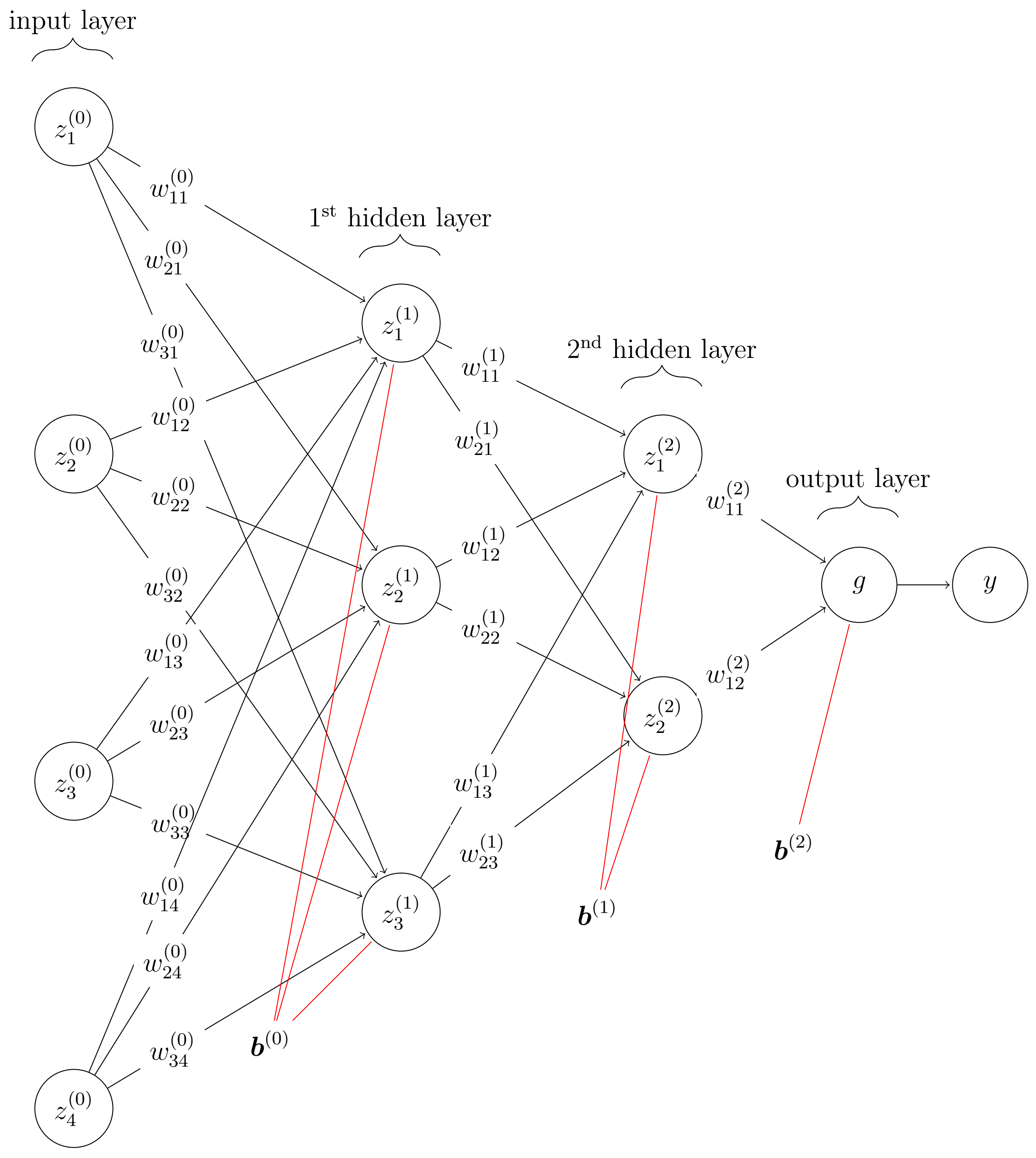


Figure 3.1: Neural Network with Two Hidden Layers

In Figure 3.1, neurons are not activated. From (2.2), the $a_i^{(j)}$'s terms below are obtained from $a_i^{(j+1)} = \sigma(z_i^{(j+1)})$. Defining step by step (i.e. layer per layer) how g is a function of the input layer, we first start from the output layer,

$$g = w_{11}^{(2)} a_1^{(2)} + w_{12}^{(2)} a_2^{(2)} + b^{(2)}, \tag{3.1}$$

where, using (2.2),

$$a_1^{(2)} = \sigma(z_1^{(2)}) = \sigma(w_{11}^{(1)} a_1^{(1)} + w_{12}^{(1)} a_2^{(1)} + w_{13}^{(1)} a_3^{(1)} + b_1^{(1)}) \tag{3.2}$$

$$a_2^{(2)} = \sigma(z_2^{(2)}) = \sigma(w_{21}^{(1)} a_1^{(1)} + w_{22}^{(1)} a_2^{(1)} + w_{23}^{(1)} a_3^{(1)} + b_2^{(1)}). \tag{3.3}$$

The activated units from layer 1 are functions of the input layer and are as follows,

$$a_1^{(1)} = \sigma(z_1^{(1)}) = \sigma(w_{11}^{(0)} a_1^{(0)} + w_{12}^{(0)} a_2^{(0)} + w_{13}^{(0)} a_3^{(0)} + w_{14}^{(0)} a_4^{(0)} + b_1^{(0)}) \tag{3.4}$$

$$a_2^{(1)} = \sigma(z_2^{(1)}) = \sigma(w_{21}^{(0)} a_1^{(0)} + w_{22}^{(0)} a_2^{(0)} + w_{23}^{(0)} a_3^{(0)} + w_{24}^{(0)} a_4^{(0)} + b_2^{(0)}) \tag{3.5}$$

$$a_3^{(1)} = \sigma(z_3^{(1)}) = \sigma(w_{31}^{(0)} a_1^{(0)} + w_{32}^{(0)} a_2^{(0)} + w_{33}^{(0)} a_3^{(0)} + w_{34}^{(0)} a_4^{(0)} + b_3^{(0)}). \tag{3.6}$$

Please note that normally, neurons from the input layer (layer 0) are not activated (i.e. $a_i^{(0)} = z_i^{(0)}$). However, if one would use a given layer of a model to feed another NN, then the input layer of the second NN would be activated, which is the case using convolutional neural networks. For generalization purposes, partial derivatives are demonstrated with activation of the input layer.

3.1 First Derivative

Since g is not directly a function of the inputs $\boldsymbol{z^{(0)}}$, the chain rule for the first derivative is used. Let $g = f(u)$ and $u = y(x)$, then

$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial u}\frac{\partial u}{\partial x}.$$

Derivative of g can be evaluated with respect to any nodes in the NN, including variables from input layer. Going through the process above to find the derivative of g with respect to the first input $z_1^{(0)}$, $\dfrac{\partial g}{\partial z_1^{(0)}}$ is as follows,

$$\begin{aligned}\frac{\partial g}{\partial z_1^{(0)}} &= \frac{\partial}{\partial z_1^{(0)}}\left(w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + b^{(2)}\right)\\ &= \frac{\partial g}{\partial a_1^{(2)}}\frac{\partial a_1^{(2)}}{\partial z_1^{(0)}} + \frac{\partial g}{\partial a_2^{(2)}}\frac{\partial a_2^{(2)}}{\partial z_1^{(0)}}\\ &= w_{11}^{(2)}\frac{\partial a_1^{(2)}}{\partial z_1^{(0)}} + w_{12}^{(2)}\frac{\partial a_2^{(2)}}{\partial z_1^{(0)}}.\end{aligned} \tag{3·7}$$

Derivatives of $a_1^{(2)}$ and $a_2^{(2)}$ can be expressed using layer 1 terms as follows,

$$\begin{aligned}
\frac{\partial a_1^{(2)}}{\partial z_1^{(0)}} &= \frac{\partial}{\partial z_1^{(0)}} \left(\sigma(z_1^{(2)})\right) \\
&= \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial z_1^{(0)}} \\
&= \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \left[\frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} \frac{\partial a_1^{(1)}}{\partial z_1^{(0)}} + \frac{\partial z_1^{(2)}}{\partial a_2^{(1)}} \frac{\partial a_2^{(1)}}{\partial z_1^{(0)}} + \frac{\partial z_1^{(2)}}{\partial a_3^{(1)}} \frac{\partial a_3^{(1)}}{\partial z_1^{(0)}} \right] \\
&= \sigma'(z_1^{(2)}) \left[w_{11}^{(1)} \frac{\partial a_1^{(1)}}{\partial z_1^{(0)}} + w_{12}^{(1)} \frac{\partial a_2^{(1)}}{\partial z_1^{(0)}} + w_{13}^{(1)} \frac{\partial a_3^{(1)}}{\partial z_1^{(0)}} \right], \\
\frac{\partial a_2^{(2)}}{\partial z_1^{(0)}} &= \frac{\partial}{\partial z_1^{(0)}} \left(\sigma(z_2^{(2)})\right) \\
&\dots \\
&= \sigma'(z_2^{(2)}) \left[w_{21}^{(1)} \frac{\partial a_1^{(1)}}{\partial z_1^{(0)}} + w_{22}^{(1)} \frac{\partial a_2^{(1)}}{\partial z_1^{(0)}} + w_{23}^{(1)} \frac{\partial a_3^{(1)}}{\partial z_1^{(0)}} \right].
\end{aligned}$$

Derivatives of $a_1^{(1)}$, $a_2^{(1)}$ and $a_3^{(1)}$ can be expressed using input layer terms as follows,

$$\begin{aligned}
\frac{\partial a_1^{(1)}}{\partial z_1^{(0)}} &= \frac{\partial}{\partial z_1^{(0)}} \left(\sigma(z_1^{(1)})\right) \\
&= \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \frac{\partial z_1^{(1)}}{\partial z_1^{(0)}} \\
&= \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \frac{\partial z_1^{(1)}}{\partial a_1^{(0)}} \frac{\partial a_1^{(0)}}{\partial z_1^{(0)}} \\
&= \sigma'(z_1^{(1)}) w_{11}^{(0)} \sigma'(z_1^{(0)}), \\
\frac{\partial a_2^{(1)}}{\partial z_1^{(0)}} &= \frac{\partial}{\partial z_1^{(0)}} \left(\sigma(z_2^{(1)})\right) \\
&= \sigma'(z_2^{(1)}) w_{21}^{(0)} \sigma'(z_1^{(0)}), \\
\frac{\partial a_3^{(1)}}{\partial z_1^{(0)}} &= \frac{\partial}{\partial z_1^{(0)}} \left(\sigma(z_3^{(1)})\right) \\
&= \sigma'(z_3^{(1)}) w_{31}^{(0)} \sigma'(z_1^{(0)}).
\end{aligned}$$

Combining all derived elements above in (3.7) results as follows,

$$\begin{aligned}
\frac{\partial g}{\partial z_1^{(0)}} &= \frac{\partial g}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \left[\frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \frac{\partial z_1^{(1)}}{\partial a_1^{(0)}} \frac{\partial a_1^{(0)}}{\partial z_1^{(0)}} + \frac{\partial z_1^{(2)}}{\partial a_2^{(1)}} \frac{\partial a_2^{(1)}}{\partial z_2^{(1)}} \frac{\partial z_2^{(1)}}{\partial a_1^{(0)}} \frac{\partial a_1^{(0)}}{\partial z_1^{(0)}} \right. \\
&\quad \left. + \frac{\partial z_1^{(2)}}{\partial a_3^{(1)}} \frac{\partial a_3^{(1)}}{\partial z_3^{(1)}} \frac{\partial z_3^{(1)}}{\partial a_1^{(0)}} \frac{\partial a_1^{(0)}}{\partial z_1^{(0)}} \right] + \frac{\partial g}{\partial a_2^{(2)}} \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \left[\frac{\partial z_2^{(2)}}{\partial a_1^{(1)}} \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \frac{\partial z_1^{(1)}}{\partial a_1^{(0)}} \frac{\partial a_1^{(0)}}{\partial z_1^{(0)}} \right. \\
&\quad \left. + \frac{\partial z_2^{(2)}}{\partial a_2^{(1)}} \frac{\partial a_2^{(1)}}{\partial z_2^{(1)}} \frac{\partial z_2^{(1)}}{\partial a_1^{(0)}} \frac{\partial a_1^{(0)}}{\partial z_1^{(0)}} + \frac{\partial z_2^{(2)}}{\partial a_3^{(1)}} \frac{\partial a_3^{(1)}}{\partial z_3^{(1)}} \frac{\partial z_3^{(1)}}{\partial a_1^{(0)}} \frac{\partial a_1^{(0)}}{\partial z_1^{(0)}} \right] \\
&= w_{11}^{(2)} \sigma' \left(z_1^{(2)} \right) \left[w_{11}^{(1)} \sigma' \left(z_1^{(1)} \right) w_{11}^{(0)} \sigma'(z_1^{(0)}) + w_{12}^{(1)} \sigma' \left(z_2^{(1)} \right) w_{21}^{(0)} \sigma'(z_1^{(0)}) \right. \\
&\quad \left. + w_{13}^{(1)} \sigma' \left(z_3^{(1)} \right) w_{31}^{(0)} \sigma'(z_1^{(0)}) \right] + w_{12}^{(2)} \sigma' \left(z_2^{(2)} \right) \left[w_{21}^{(1)} \sigma' \left(z_1^{(1)} \right) w_{11}^{(0)} \sigma'(z_1^{(0)}) \right. \\
&\quad \left. + w_{22}^{(1)} \sigma' \left(z_2^{(1)} \right) w_{21}^{(0)} \sigma'(z_1^{(0)}) + w_{23}^{(1)} \sigma' \left(z_3^{(1)} \right) w_{31}^{(0)} \sigma'(z_1^{(0)}) \right].
\end{aligned} \tag{3.8}$$

The same process can be applied to obtain the first derivatives of g with respect to the other input variables $\frac{\partial g}{\partial z_2^{(0)}}$, $\frac{\partial g}{\partial z_3^{(0)}}$ and $\frac{\partial g}{\partial z_4^{(0)}}$. The matrix representation is as follows,

$$
\begin{aligned}
\frac{\partial g}{\partial \mathbf{z}^{(0)}} \quad &= \quad \underbrace{\overbrace{\begin{bmatrix} \sigma'(z_1^{(0)}) \\ \sigma'(z_2^{(0)}) \\ \sigma'(z_3^{(0)}) \\ \sigma'(z_4^{(0)}) \end{bmatrix} \odot \begin{bmatrix} w_{11}^{(0)} & w_{21}^{(0)} & w_{31}^{(0)} \\ w_{12}^{(0)} & w_{22}^{(0)} & w_{32}^{(0)} \\ w_{13}^{(0)} & w_{23}^{(0)} & w_{33}^{(0)} \\ w_{14}^{(0)} & w_{24}^{(0)} & w_{34}^{(0)} \end{bmatrix}}^{\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{z}^{(0)}}}}_{\text{Layer L}-2} \\
&\times \quad \underbrace{\overbrace{\begin{bmatrix} \sigma'\left(z_1^{(1)}\right) \\ \sigma'\left(z_2^{(1)}\right) \\ \sigma'\left(z_3^{(1)}\right) \end{bmatrix} \odot \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} \end{bmatrix}}^{\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(1)}}}}_{\text{Layer L}-1} \\
&\times \quad \underbrace{\overbrace{\begin{bmatrix} \sigma'(z_1^{(2)}) \\ \sigma'(z_2^{(2)}) \end{bmatrix} \odot \begin{bmatrix} w_{11}^{(2)} \\ w_{12}^{(2)} \end{bmatrix}}^{\frac{\partial g}{\partial \mathbf{z}^{(2)}}}}_{\text{Layer L}} \\
&= \quad \left[\sigma'(\mathbf{z}^{(0)}) \odot \mathbf{W}^{(0)}\right] \times \left[\sigma'(\mathbf{z}^{(1)}) \odot \mathbf{W}^{(1)}\right] \times \left[\sigma'(\mathbf{z}^{(2)}) \odot \mathbf{W}^{(2)}\right],
\end{aligned} \tag{3.9}
$$

where $\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(0)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{z}^{(0)}}$. Since the network is fully connected, which means that all nodes and weights from consecutive layers are connected, there is no matrix dimension issues when multiplying matrices, element wise or not.

Recall that in TAGI, every parameters and nodes follow Normal distributions

$\mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}_{\boldsymbol{\theta}|\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}|\boldsymbol{y}})$ and $\mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{Z}}, \boldsymbol{\Sigma}_{\boldsymbol{Z}})$, respectively. Therefore, the mean of the derivative of g can be obtained. It is represented by taking the expectation of a product of six terms. The detailed explanations of g derivative calculations for a NN with two hidden layers are provided in Appendix A.

Adding layers results in more and longer product terms to sum, but the approach remains the same. Let V represents weights and nodes in the NN. For a model with L hidden layers,

$$\begin{aligned}
&\mathbb{E}\left[V_1 V_2 ... V_{2L-1} V_{2L} V_{2L+1} V_{2(L+1)}\right] \\
&= \mathbb{E}\left[V_1 V_2 ... V_{2L-1} V_{2L}\right] \mathbb{E}\left[V_{2L+1} V_{2(L+1)}\right] \\
&\quad + \mathbb{E}\left[V_1 V_2 ... V_{2(L-1)-1} V_{2(L-1)}\right] \operatorname{cov}(V_{2L-1} V_{2L}, V_{2L+1} V_{2(L+1)}).
\end{aligned} \tag{3.10}$$

3.2 Second Derivative

Again, we use the chain rule to compute the next order derivative. Let $g = f(u)$ and $u = y(x)$, then

$$\frac{\partial^2 g}{(\partial x)^2} = \frac{\partial^2 g}{(\partial u)^2} \left(\frac{\partial u}{\partial x}\right)^2 + \frac{\partial g}{\partial u} \frac{\partial^2 u}{(\partial x)^2}. \tag{3.11}$$

Continuing with the example of a NN with two hidden layers, $u = \boldsymbol{z}^{(2)}$ and $x = \boldsymbol{z}^{(0)}$ (the inputs). Recall that $\boldsymbol{z}^{(2)}$ are functions which can be expressed in terms of $\boldsymbol{z}^{(0)}$. Using the chain rule, the second derivative of g can be expressed as follows,

$$\frac{\partial^2 g}{(\partial \boldsymbol{z}^{(0)})^2} = \frac{\partial^2 g}{(\partial \boldsymbol{z}^{(2)})^2} \left(\frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{z}^{(0)}}\right)^2 + \frac{\partial g}{\partial \boldsymbol{z}^{(2)}} \frac{\partial^2 \boldsymbol{z}^{(2)}}{(\partial \boldsymbol{z}^{(0)})^2}. \tag{3.12}$$

The second derivative of g with respect to the first input variable $z_1^{(0)}$, $\dfrac{\partial^2 g}{(\partial z_1^{(0)})^2}$, is derived from (3.8) and is equal to

$$\begin{aligned}
&\frac{\partial g}{\partial a_1^{(2)}}\,\frac{\partial^2 a_1^{(2)}}{(\partial z_1^{(2)})^2}\left[\frac{\partial z_1^{(2)}}{\partial a_1^{(1)}}\,\frac{\partial a_1^{(1)}}{\partial z_1^{(1)}}\,\frac{\partial z_1^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial a_1^{(0)}}{\partial z_1^{(0)}}+\frac{\partial z_1^{(2)}}{\partial a_2^{(1)}}\,\frac{\partial a_2^{(1)}}{\partial z_2^{(1)}}\,\frac{\partial z_2^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial a_1^{(0)}}{\partial z_1^{(0)}}\right.\\
&\quad\left.+\frac{\partial z_1^{(2)}}{\partial a_3^{(1)}}\,\frac{\partial a_3^{(1)}}{\partial z_3^{(1)}}\,\frac{\partial z_3^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial a_1^{(0)}}{\partial z_1^{(0)}}\right]^2+\frac{\partial g}{\partial a_1^{(2)}}\,\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\left[\frac{\partial z_1^{(2)}}{\partial a_1^{(1)}}\,\frac{\partial^2 a_1^{(1)}}{(\partial z_1^{(1)})^2}\left(\frac{\partial z_1^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial a_1^{(0)}}{\partial z_1^{(0)}}\right)^2\right.\\
&\quad\left.+\frac{\partial z_1^{(2)}}{\partial a_2^{(1)}}\,\frac{\partial^2 a_2^{(1)}}{(\partial z_2^{(1)})^2}\left(\frac{\partial z_2^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial a_1^{(0)}}{\partial z_1^{(0)}}\right)^2+\frac{\partial z_1^{(2)}}{\partial a_3^{(1)}}\,\frac{\partial^2 a_3^{(1)}}{(\partial z_3^{(1)})^2}\left(\frac{\partial z_3^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial a_1^{(0)}}{\partial z_1^{(0)}}\right)^2\right]\\
&\quad+\frac{\partial g}{\partial a_1^{(2)}}\,\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\left[\frac{\partial z_1^{(2)}}{\partial a_1^{(1)}}\,\frac{\partial a_1^{(1)}}{\partial z_1^{(1)}}\,\frac{\partial z_1^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial^2 a_1^{(0)}}{(\partial z_1^{(0)})^2}+\frac{\partial z_1^{(2)}}{\partial a_2^{(1)}}\,\frac{\partial a_2^{(1)}}{\partial z_2^{(1)}}\,\frac{\partial z_2^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial^2 a_1^{(0)}}{(\partial z_1^{(0)})^2}\right.\\
&\quad\left.+\frac{\partial z_1^{(2)}}{\partial a_3^{(1)}}\,\frac{\partial a_3^{(1)}}{\partial z_3^{(1)}}\,\frac{\partial z_3^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial^2 a_1^{(0)}}{(\partial z_1^{(0)})^2}\right]+\frac{\partial g}{\partial a_2^{(2)}}\,\frac{\partial^2 a_2^{(2)}}{(\partial z_2^{(2)})^2}\left[\frac{\partial z_2^{(2)}}{\partial a_1^{(1)}}\,\frac{\partial a_1^{(1)}}{\partial z_1^{(1)}}\,\frac{\partial z_1^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial a_1^{(0)}}{\partial z_1^{(0)}}\right.\\
&\quad\left.+\frac{\partial z_2^{(2)}}{\partial a_2^{(1)}}\,\frac{\partial a_2^{(1)}}{\partial z_2^{(1)}}\,\frac{\partial z_2^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial a_1^{(0)}}{\partial z_1^{(0)}}\frac{\partial z_2^{(2)}}{\partial a_3^{(1)}}\,\frac{\partial a_3^{(1)}}{\partial z_3^{(1)}}\,\frac{\partial z_3^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial a_1^{(0)}}{\partial z_1^{(0)}}\right]^2\\
&\quad+\frac{\partial g}{\partial a_2^{(2)}}\,\frac{\partial a_2^{(2)}}{\partial z_2^{(2)}}\left[\frac{\partial z_2^{(2)}}{\partial a_1^{(1)}}\,\frac{\partial^2 a_1^{(1)}}{(\partial z_1^{(1)})^2}\left(\frac{\partial z_1^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial a_1^{(0)}}{\partial z_1^{(0)}}\right)^2\right.\\
&\quad\left.+\frac{\partial z_2^{(2)}}{\partial a_2^{(1)}}\,\frac{\partial^2 a_2^{(1)}}{(\partial z_2^{(1)})^2}\left(\frac{\partial z_2^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial a_1^{(0)}}{\partial z_1^{(0)}}\right)^2+\frac{\partial z_2^{(2)}}{\partial a_3^{(1)}}\,\frac{\partial^2 a_3^{(1)}}{(\partial z_3^{(1)})^2}\left(\frac{\partial z_3^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial a_1^{(0)}}{\partial z_1^{(0)}}\right)^2\right]\\
&\quad+\frac{\partial g}{\partial a_2^{(2)}}\,\frac{\partial a_2^{(2)}}{\partial z_2^{(2)}}\left[\frac{\partial z_2^{(2)}}{\partial a_1^{(1)}}\,\frac{\partial a_1^{(1)}}{\partial z_1^{(1)}}\,\frac{\partial z_1^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial^2 a_1^{(0)}}{(\partial z_1^{(0)})^2}\right.\\
&\quad\left.+\frac{\partial z_2^{(2)}}{\partial a_2^{(1)}}\,\frac{\partial a_2^{(1)}}{\partial z_2^{(1)}}\,\frac{\partial z_2^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial^2 a_1^{(0)}}{(\partial z_1^{(0)})^2}+\frac{\partial z_2^{(2)}}{\partial a_3^{(1)}}\,\frac{\partial a_3^{(1)}}{\partial z_3^{(1)}}\,\frac{\partial z_3^{(1)}}{\partial a_1^{(0)}}\,\frac{\partial^2 a_1^{(0)}}{(\partial z_1^{(0)})^2}\right].
\end{aligned} \tag{3.13}$$

Replacing all the derivative terms,

$$
\begin{aligned}
&\frac{\partial^2 g}{(\partial z_1^{(0)})^2} \\
&= w_{11}^{(2)}\sigma''(z_1^{(2)})\left\{w_{11}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)}) + w_{12}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)})\right. \\
&\quad \left. + w_{13}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma'(z_1^{(0)})\right\}^2 + w_{11}^{(2)}\sigma'(z_1^{(2)})\left\{w_{11}^{(1)}\sigma''(z_1^{(1)})\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]^2\right. \\
&\quad \left. + w_{12}^{(1)}\sigma''(z_2^{(1)})\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]^2 + w_{13}^{(1)}\sigma''(z_3^{(1)})\left[w_{31}^{(0)}\sigma'(z_1^{(0)})\right]^2\right\} \\
&\quad + w_{11}^{(2)}\sigma'(z_1^{(2)})\left\{w_{11}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma''(z_1^{(0)}) + w_{12}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma''(z_1^{(0)})\right. \\
&\quad \left. + w_{13}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma''(z_1^{(0)})\right\} + w_{12}^{(2)}\sigma''(z_2^{(2)})\left\{w_{21}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)})\right. \\
&\quad \left. + w_{22}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)}) + w_{23}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma'(z_1^{(0)})\right\}^2 \\
&\quad + w_{12}^{(2)}\sigma'(z_2^{(2)})\left\{w_{21}^{(1)}\sigma''(z_1^{(1)})\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]^2 + w_{22}^{(1)}\sigma''(z_2^{(1)})\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]^2\right. \\
&\quad \left. + w_{23}^{(1)}\sigma''(z_3^{(1)})\left[w_{31}^{(0)}\sigma'(z_1^{(0)})\right]^2\right\} + w_{12}^{(2)}\sigma'(z_2^{(2)})\left\{w_{21}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma''(z_1^{(0)})\right. \\
&\quad \left. + w_{22}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma''(z_1^{(0)}) + w_{23}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma''(z_1^{(0)})\right\}.
\end{aligned}
\tag{3.14}
$$

The same process can be applied to obtain the second derivatives of g with respect to the other input variables $\dfrac{\partial^2 g}{(\partial z_2^{(0)})^2}$, $\dfrac{\partial^2 g}{(\partial z_3^{(0)})^2}$ and $\dfrac{\partial^2 g}{(\partial z_4^{(0)})^2}$. From (3.12), $\dfrac{\partial g}{\partial \boldsymbol{z}^{(2)}}$ and $\dfrac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{z}^{(0)}}$ are already shown in matrix form in (3.9). The remaining terms, $\dfrac{\partial^2 g}{(\partial \mathbf{z}^{(2)})^2}$ and $\dfrac{\partial^2 \mathbf{z}^{(2)}}{(\partial \mathbf{z}^{(0)})^2}$, are represented in matrix form as follows,

$$\frac{\partial^2 g}{(\partial \mathbf{z}^{(2)})^2} = \underbrace{\begin{bmatrix} \sigma''(z_1^{(2)}) \\ \sigma''(z_2^{(2)}) \end{bmatrix} \odot \begin{bmatrix} w_{11}^{(2)} \\ w_{12}^{(2)} \end{bmatrix}}_{\text{Layer L}}, \tag{3.15}$$

$$\frac{\partial^2 \mathbf{z}^{(2)}}{(\partial \mathbf{z}^{(0)})^2} = \underbrace{\left\{ \begin{bmatrix} \sigma'(z_1^{(0)}) \\ \sigma'(z_2^{(0)}) \\ \sigma'(z_3^{(0)}) \\ \sigma'(z_4^{(0)}) \end{bmatrix} \odot \begin{bmatrix} w_{11}^{(0)} & w_{21}^{(0)} & w_{31}^{(0)} \\ w_{12}^{(0)} & w_{22}^{(0)} & w_{32}^{(0)} \\ w_{13}^{(0)} & w_{23}^{(0)} & w_{33}^{(0)} \\ w_{14}^{(0)} & w_{24}^{(0)} & w_{34}^{(0)} \end{bmatrix} \right\}^{\odot 2}}_{\text{Layer L}-2}$$

$$\times \underbrace{\begin{bmatrix} \sigma''\left(z_1^{(1)}\right) \\ \sigma''\left(z_2^{(1)}\right) \\ \sigma''\left(z_3^{(1)}\right) \end{bmatrix} \odot \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} \end{bmatrix}}_{\text{Layer L}-1} \tag{3.16}$$

$$+ \underbrace{\begin{bmatrix} \sigma''(z_1^{(0)}) \\ \sigma''(z_2^{(0)}) \\ \sigma''(z_3^{(0)}) \\ \sigma''(z_4^{(0)}) \end{bmatrix} \odot \begin{bmatrix} w_{11}^{(0)} & w_{21}^{(0)} & w_{31}^{(0)} \\ w_{12}^{(0)} & w_{22}^{(0)} & w_{32}^{(0)} \\ w_{13}^{(0)} & w_{23}^{(0)} & w_{33}^{(0)} \\ w_{14}^{(0)} & w_{24}^{(0)} & w_{34}^{(0)} \end{bmatrix}}_{\text{Layer L}-2}$$

$$\times \underbrace{\begin{bmatrix} \sigma'\left(z_1^{(1)}\right) \\ \sigma'\left(z_2^{(1)}\right) \\ \sigma'\left(z_3^{(1)}\right) \end{bmatrix} \odot \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} \end{bmatrix}}_{\text{Layer L}-1}.$$

The detailed explanations of g second derivative calculations for a NN with two hidden layers are provided in Appendix B.

3.2.1 Generalization

Adding hidden layers to the model results in adding sequence types, which are different combinations of product terms from all hidden layers and input layer. As seen in a model with two hidden layers, the sequences follow some patterns because of the chain rule for first and second order derivatives. The observed patterns are:

1. A sequence will follow the first derivative process (multiplying two terms of first order per layer) until there is a first pair of second order.
2. At only one layer per sequence, there is a pair of second order terms and that position is unique among all sequences. Also, among all sequences, pairs of second order terms fill all positions.
3. Following a pair of second order terms are four terms of first order thereafter.

Table B.1 is modified to illustrate the general case for a model with L hidden layers which is shown in Table 3.1.

	Layer L	Layer $L-1$	$\cdots$	Layer 1	Layer 0
1	2×2^{nd} order	4×1^{st} order	$\cdots$	4×1^{st} order	4×1^{st} order
2	2×1^{st} order	2×2^{nd} order	$\cdots$	4×1^{st} order	4×1^{st} order
$\vdots$	2×1^{st} order	2×1^{st} order	$\cdots$	4×1^{st} order	4×1^{st} order
L	2×1^{st} order	2×1^{st} order	$\cdots$	2×2^{nd} order	4×1^{st} order
$L+1$	2×1^{st} order	2×1^{st} order	$\cdots$	2×1^{st} order	2×2^{nd} order

Table 3.1: Sequence Types of Product Terms

Some combinations are not covered yet in Section 3.2, since they do not exist in the context of a NN with two hidden layers. Illustrating a model with three hidden layers as follows,

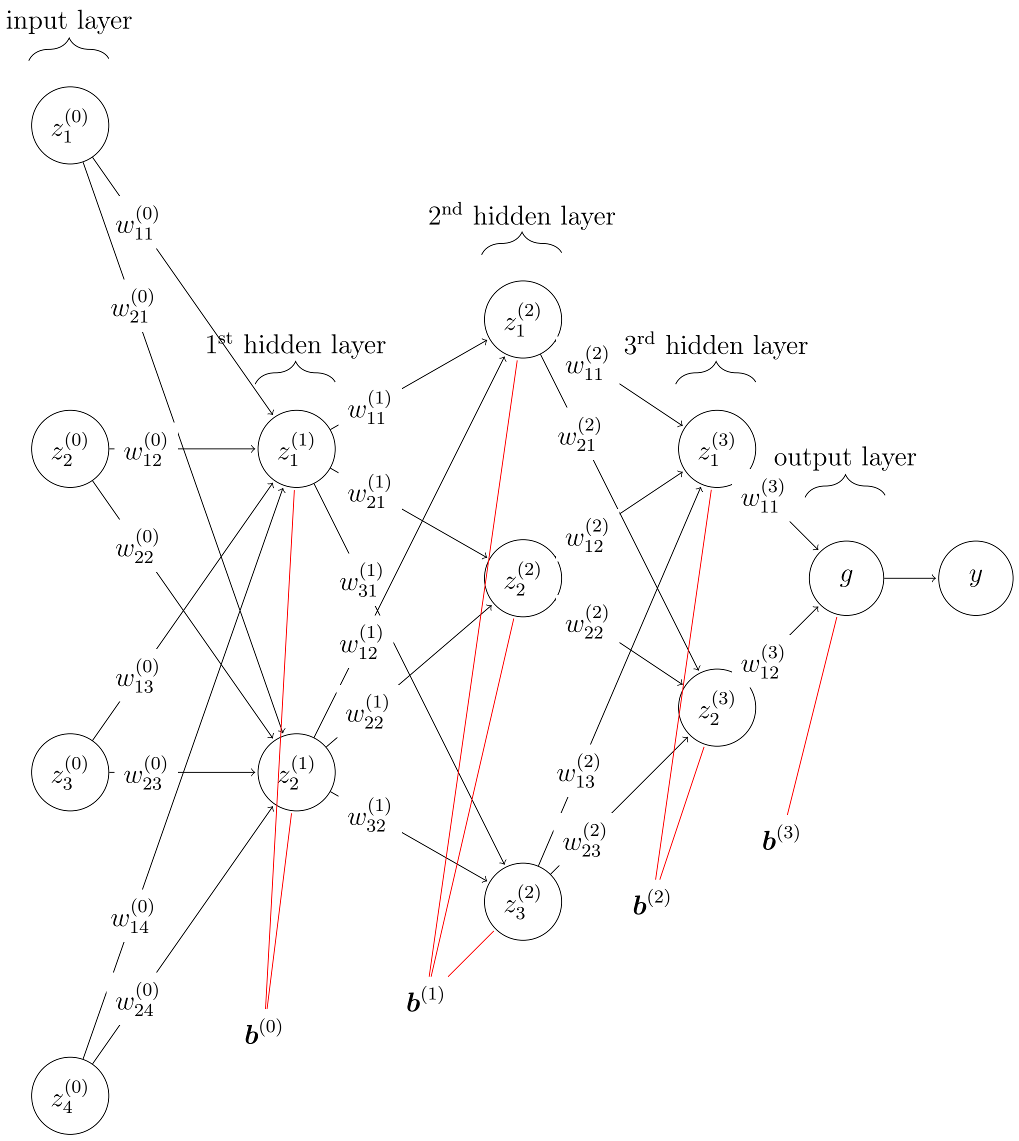


Figure 3.2: Neural Network with Three Hidden Layers

In the model shown in Figure 3.2, the second derivative of g, using the chain rule, can be expressed as follows,

$$\frac{\partial^2 g}{(\partial \boldsymbol{z}^{(0)})^2} = \frac{\partial^2 g}{(\partial \boldsymbol{z}^{(3)})^2}\left(\frac{\partial \boldsymbol{z}^{(3)}}{\partial \boldsymbol{z}^{(0)}}\right)^2 + \frac{\partial g}{\partial \boldsymbol{z}^{(3)}}\frac{\partial^2 \boldsymbol{z}^{(3)}}{(\partial \boldsymbol{z}^{(0)})^2}, \tag{3.17}$$

where, considering only the above derivative with respect to the first input variable, $\frac{\partial^2 g}{(\partial \boldsymbol{z}^{(3)})^2}\left(\frac{\partial \boldsymbol{z}^{(3)}}{\partial z_1^{(0)}}\right)^2$ is equal to

$$\begin{aligned}
&w_{11}^{(3)}\sigma''(z_1^{(3)})\\
&\{w_{11}^{(2)}\sigma'(z_1^{(2)})w_{11}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)}) + w_{11}^{(2)}\sigma'(z_1^{(2)})w_{12}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)})\\
&+ w_{12}^{(2)}\sigma'(z_2^{(2)})w_{21}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)}) + w_{12}^{(2)}\sigma'(z_2^{(2)})w_{22}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)})\\
&+ w_{13}^{(2)}\sigma'(z_3^{(2)})w_{31}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)}) + w_{13}^{(2)}\sigma'(z_3^{(2)})w_{32}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)})\}^2\\
&+ w_{12}^{(2)}\sigma''(z_2^{(2)})\\
&\{w_{21}^{(2)}\sigma'(z_1^{(2)})w_{11}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)}) + w_{21}^{(2)}\sigma'(z_1^{(2)})w_{12}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)})\\
&+ w_{22}^{(2)}\sigma'(z_2^{(2)})w_{21}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)}) + w_{22}^{(2)}\sigma'(z_2^{(2)})w_{22}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)})\\
&+ w_{23}^{(2)}\sigma'(z_3^{(2)})w_{31}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)}) + w_{23}^{(2)}\sigma'(z_3^{(2)})w_{32}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)})\}^2.
\end{aligned} \tag{3.18}$$

Evaluating the second derivative of g considering the model with three hidden layers follows the same process as for the model with two hidden layers for layers 2 and 3 (refer to layers 1 and 2 respectively in Section B.1). However, layer 1 is treated differently, since once the expression above is developed, expectations of products from layer 1 shown in Table 3.2 are to be calculated.

Case	Expectations
1	$\mathbb{E}\left[\left(w_{11}^{(1)}\sigma'(z_1^{(1)})\right)^2\right]$, $\mathbb{E}\left[\left(w_{12}^{(1)}\sigma'(z_2^{(1)})\right)^2\right]$, $\mathbb{E}\left[\left(w_{21}^{(1)}\sigma'(z_1^{(1)})\right)^2\right]$ $\mathbb{E}\left[\left(w_{22}^{(1)}\sigma'(z_2^{(1)})\right)^2\right]$, $\mathbb{E}\left[\left(w_{31}^{(1)}\sigma'(z_1^{(1)})\right)^2\right]$, $\mathbb{E}\left[\left(w_{32}^{(1)}\sigma'(z_2^{(1)})\right)^2\right]$
2(a)	$\mathbb{E}\left[w_{11}^{(1)}\sigma'(z_1^{(1)})w_{12}^{(1)}\sigma'(z_2^{(1)})\right]$, $\mathbb{E}\left[w_{21}^{(1)}\sigma'(z_1^{(1)})w_{22}^{(1)}\sigma'(z_2^{(1)})\right]$, $\mathbb{E}\left[w_{31}^{(1)}\sigma'(z_1^{(1)})w_{32}^{(1)}\sigma'(z_2^{(1)})\right]$
2(b)	$\mathbb{E}\left[w_{11}^{(1)}\sigma'(z_1^{(1)})w_{22}^{(1)}\sigma'(z_2^{(1)})\right]$, $\mathbb{E}\left[w_{11}^{(1)}\sigma'(z_1^{(1)})w_{32}^{(1)}\sigma'(z_2^{(1)})\right]$, $\mathbb{E}\left[w_{12}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(1)}\sigma'(z_1^{(1)})\right]$, $\mathbb{E}\left[w_{12}^{(1)}\sigma'(z_2^{(1)})w_{31}^{(1)}\sigma'(z_1^{(1)})\right]$, $\mathbb{E}\left[w_{21}^{(1)}\sigma'(z_1^{(1)})w_{32}^{(1)}\sigma'(z_2^{(1)})\right]$, $\mathbb{E}\left[w_{22}^{(1)}\sigma'(z_2^{(1)})w_{31}^{(1)}\sigma'(z_1^{(1)})\right]$
3	$\mathbb{E}\left[w_{11}^{(1)}w_{21}^{(1)}\sigma'(z_1^{(1)})^2\right]$, $\mathbb{E}\left[w_{11}^{(1)}w_{31}^{(1)}\sigma'(z_1^{(1)})^2\right]$, $\mathbb{E}\left[w_{12}^{(1)}w_{22}^{(1)}\sigma'(z_2^{(1)})^2\right]$, $\mathbb{E}\left[w_{12}^{(1)}w_{32}^{(1)}\sigma'(z_2^{(1)})^2\right]$ $\mathbb{E}\left[w_{21}^{(1)}w_{31}^{(1)}\sigma'(z_1^{(1)})^2\right]$, $\mathbb{E}\left[w_{22}^{(1)}w_{32}^{(1)}\sigma'(z_2^{(1)})^2\right]$

Table 3.2: Expectations at Layer 1 for a Three Hidden Layers Model

Let $V_7V_8V_9V_{10}$ be the four order derivative terms from layer 1. The cases are explained as follows,

1. $V_7V_8 = V_9V_{10}$: Same node and weight are multiplied.

2. $V_7 \neq V_9$ and $V_8 \neq V_{10}$: Not same node nor weight are multiplied.

   (a) Weights point toward the same next layer node.

   (b) Weights do not point toward the same next layer node.

3. $V_7 \neq V_9$ and $V_8 = V_{10}$: Same node but not same weight are multiplied.

Case 2(b) does not exist in the model with two hidden layers, so when calculating expectations, $\mathrm{cov}(V_7V_8, V_9V_{10}) = 0$. Moreover, it is the first time that all cases exist together. Cases 1 and 2(a) arise in layer 1 and Cases 1 and 3 in layer 0 in Section B.1, but never the three of them at the same time in all the calculation process from Section 3.2.

In a given sequence, let j be the layer where there is the pair of second order terms, then Case 2(b) can be observed in layers 1 to $j-2$ inclusively, where $j \in \{3, ...L\}$. Ultimately, Case 2(b) has an impact on the covariance between the next and current layers, the number of possible covariance terms to consider increases. Therefore, $(B.6)$ cannot automatically be simplified as before.

3.3 Third Derivative

To obtain the next order derivative, the chain rule for third derivative is used. Let $g = f(u)$ and $u = y(x)$, then

$$\frac{\partial^3 g}{(\partial x)^3} = \frac{\partial^3 g}{(\partial u)^3}\left(\frac{\partial u}{\partial x}\right)^3 + 3\frac{\partial^2 g}{(\partial u)^2}\frac{\partial u}{\partial x}\frac{\partial^2 u}{(\partial x)^2} + \frac{\partial g}{\partial u}\frac{\partial^3 u}{(\partial x)^3}. \tag{3.19}$$

Continuing with the example of a NN with two hidden layers, $u = \boldsymbol{z}^{(2)}$ and $x = \boldsymbol{z}^{(0)}$ (the inputs). Recall that $\boldsymbol{z}^{(2)}$ are functions that can be expressed in terms of $\boldsymbol{z}^{(0)}$. Using the chain rule, the third derivative of g can be expressed as follows,

$$\frac{\partial^3 g}{(\partial \boldsymbol{z}^{(0)})^3} = \frac{\partial^3 g}{(\partial \boldsymbol{z}^{(2)})^3}\left(\frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{z}^{(0)}}\right)^3 + 3\frac{\partial^2 g}{(\partial \boldsymbol{z}^{(2)})^2}\frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{z}^{(0)}}\frac{\partial^2 \boldsymbol{z}^{(2)}}{(\partial \boldsymbol{z}^{(0)})^2} + \frac{\partial g}{\partial \boldsymbol{z}^{(2)}}\frac{\partial^3 \boldsymbol{z}^{(2)}}{(\partial \boldsymbol{z}^{(0)})^3}. \tag{3.20}$$

$\dfrac{\partial g}{\partial \boldsymbol{z}^{(2)}}$ and $\dfrac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{z}^{(0)}}$ are already shown in matrix form in (3.9). For the second derivative terms, $\dfrac{\partial^2 g}{(\partial \mathbf{z}^{(2)})^2}$ and $\dfrac{\partial^2 \mathbf{z}^{(2)}}{(\partial \mathbf{z}^{(0)})^2}$ are shown in (3.15) and (3.16) respectively. From (3.20), the remaining terms, $\dfrac{\partial^3 g}{(\partial \mathbf{z}^{(2)})^3}$ and $\dfrac{\partial^3 \mathbf{z}^{(2)}}{(\partial \mathbf{z}^{(0)})^3}$, are represented in matrix form as follows,

$$\frac{\partial^3 g}{(\partial \mathbf{z}^{(2)})^3} = \underbrace{\begin{bmatrix} \sigma'''(z_1^{(2)}) \\ \sigma'''(z_2^{(2)}) \end{bmatrix} \odot \begin{bmatrix} w_{11}^{(2)} \\ w_{12}^{(2)} \end{bmatrix}}_{\text{Layer L}}, \tag{3.21}$$

$$\begin{aligned}\frac{\partial^3 \mathbf{z}^{(2)}}{(\partial \mathbf{z}^{(0)})^3} = & \underbrace{\left\{ \begin{bmatrix} \sigma'(z_1^{(0)}) \\ \sigma'(z_2^{(0)}) \\ \sigma'(z_3^{(0)}) \\ \sigma'(z_4^{(0)}) \end{bmatrix} \odot \begin{bmatrix} w_{11}^{(0)} & w_{21}^{(0)} & w_{31}^{(0)} \\ w_{12}^{(0)} & w_{22}^{(0)} & w_{32}^{(0)} \\ w_{13}^{(0)} & w_{23}^{(0)} & w_{33}^{(0)} \\ w_{14}^{(0)} & w_{24}^{(0)} & w_{34}^{(0)} \end{bmatrix} \right\}^{\odot 3}}_{\text{Layer L}-2} \\ & \times \underbrace{\begin{bmatrix} \sigma'''\left(z_1^{(1)}\right) \\ \sigma'''\left(z_2^{(1)}\right) \\ \sigma'''\left(z_3^{(1)}\right) \end{bmatrix} \odot \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} \end{bmatrix}}_{\text{Layer L}-1}\end{aligned}$$

$$+\quad \underbrace{3\left\{\begin{bmatrix}\sigma'(z_1^{(0)})\\ \sigma'(z_2^{(0)})\\ \sigma'(z_3^{(0)})\\ \sigma'(z_4^{(0)})\end{bmatrix}\odot\begin{bmatrix}\sigma''(z_1^{(0)})\\ \sigma''(z_2^{(0)})\\ \sigma''(z_3^{(0)})\\ \sigma''(z_4^{(0)})\end{bmatrix}\odot\begin{bmatrix}w_{11}^{(0)} & w_{21}^{(0)} & w_{31}^{(0)}\\ w_{12}^{(0)} & w_{22}^{(0)} & w_{32}^{(0)}\\ w_{13}^{(0)} & w_{23}^{(0)} & w_{33}^{(0)}\\ w_{14}^{(0)} & w_{24}^{(0)} & w_{34}^{(0)}\end{bmatrix}^{\odot 2}\right\}}_{\text{Layer L}-2}$$

$$\times\quad \underbrace{\begin{bmatrix}\sigma''\left(z_1^{(1)}\right)\\ \sigma''\left(z_2^{(1)}\right)\\ \sigma''\left(z_3^{(1)}\right)\end{bmatrix}\odot\begin{bmatrix}w_{11}^{(1)} & w_{21}^{(1)}\\ w_{12}^{(1)} & w_{22}^{(1)}\\ w_{13}^{(1)} & w_{23}^{(1)}\end{bmatrix}}_{\text{Layer L}-1}$$

$$+\quad \underbrace{\begin{bmatrix}\sigma'''(z_1^{(0)})\\ \sigma'''(z_2^{(0)})\\ \sigma'''(z_3^{(0)})\\ \sigma'''(z_4^{(0)})\end{bmatrix}\odot\begin{bmatrix}w_{11}^{(0)} & w_{21}^{(0)} & w_{31}^{(0)}\\ w_{12}^{(0)} & w_{22}^{(0)} & w_{32}^{(0)}\\ w_{13}^{(0)} & w_{23}^{(0)} & w_{33}^{(0)}\\ w_{14}^{(0)} & w_{24}^{(0)} & w_{34}^{(0)}\end{bmatrix}}_{\text{Layer L}-2}$$

$$\times\quad \underbrace{\begin{bmatrix}\sigma'\left(z_1^{(1)}\right)\\ \sigma'\left(z_2^{(1)}\right)\\ \sigma'\left(z_3^{(1)}\right)\end{bmatrix}\odot\begin{bmatrix}w_{11}^{(1)} & w_{21}^{(1)}\\ w_{12}^{(1)} & w_{22}^{(1)}\\ w_{13}^{(1)} & w_{23}^{(1)}\end{bmatrix}}_{\text{Layer L}-1}. \tag{3.22}$$

From (3.14), developing the equation for $\dfrac{\partial^3 g}{(\partial z_1^{(0)})^3}$ results as follows,

$$
\begin{aligned}
&= w_{11}^{(2)}\sigma'''(z_1^{(2)})\left\{w_{11}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)}) + w_{12}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)})\right.\\
&\quad \left.+\ w_{13}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma'(z_1^{(0)})\right\}^3 + 3w_{11}^{(2)}\sigma''(z_1^{(2)})\left\{w_{11}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)})\right.\\
&\quad \left.+\ w_{12}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)}) + w_{13}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma'(z_1^{(0)})\right\}\\
&\quad \left\{w_{11}^{(1)}\sigma''(z_1^{(1)})\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]^2 + w_{12}^{(1)}\sigma''(z_2^{(1)})\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]^2\right.\\
&\quad +\ w_{13}^{(1)}\sigma''(z_3^{(1)})\left[w_{31}^{(0)}\sigma'(z_1^{(0)})\right]^2 + w_{11}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma''(z_1^{(0)})\\
&\quad \left.+\ w_{12}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma''(z_1^{(0)}) + w_{13}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma''(z_1^{(0)})\right\}\\
&\quad + w_{11}^{(2)}\sigma'(z_1^{(2)})\left[w_{11}^{(1)}\sigma'''(z_1^{(1)})\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]^3 + w_{12}^{(1)}\sigma'''(z_2^{(1)})\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]^3\right.\\
&\quad +\ w_{13}^{(1)}\sigma'''(z_3^{(1)})\left[w_{31}^{(0)}\sigma'(z_1^{(0)})\right]^3 + 3\left\{w_{11}^{(1)}\sigma'(z_1^{(1)})[w_{11}^{(0)}]^2\sigma'(z_1^{(0)})\sigma''(z_1^{(0)})\right.\\
&\quad \left.+\ w_{12}^{(1)}\sigma'(z_2^{(1)})[w_{21}^{(0)}]^2\sigma'(z_1^{(0)})\sigma''(z_1^{(0)}) + w_{13}^{(1)}\sigma'(z_3^{(1)})[w_{31}^{(0)}]^2\sigma'(z_1^{(0)})\sigma''(z_1^{(0)})\right\}\\
&\quad +\ w_{11}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'''(z_1^{(0)}) + w_{12}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'''(z_1^{(0)})\\
&\quad \left.+\ w_{13}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma'''(z_1^{(0)})\right] + w_{12}^{(2)}\sigma'''(z_2^{(2)})\left\{w_{21}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)})\right.\\
&\quad \left.+\ w_{22}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)}) + w_{23}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma'(z_1^{(0)})\right\}^3\\
&\quad + 3w_{12}^{(2)}\sigma''(z_2^{(2)})\left\{w_{21}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)}) + w_{22}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)})\right.\\
&\quad \left.+\ w_{23}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma'(z_1^{(0)})\right\}\left\{w_{21}^{(1)}\sigma''(z_1^{(1)})\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]^2\right.\\
&\quad +\ w_{22}^{(1)}\sigma''(z_2^{(1)})\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]^2 + w_{23}^{(1)}\sigma''(z_3^{(1)})\left[w_{31}^{(0)}\sigma'(z_1^{(0)})\right]^2\\
&\quad +\ w_{21}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma''(z_1^{(0)}) + w_{22}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma''(z_1^{(0)})\\
&\quad \left.+\ w_{23}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma''(z_1^{(0)})\right\} + w_{12}^{(2)}\sigma'(z_2^{(2)})\left[w_{21}^{(1)}\sigma'''(z_1^{(1)})\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]^3\right.\\
&\quad +\ w_{22}^{(1)}\sigma'''(z_2^{(1)})\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]^3 + w_{23}^{(1)}\sigma'''(z_3^{(1)})\left[w_{31}^{(0)}\sigma'(z_1^{(0)})\right]^3
\end{aligned}
$$

$$
\begin{aligned}
&+ 3\Big\{w^{(1)}_{21}\sigma'(z^{(1)}_1)[w^{(0)}_{11}]^2\sigma'(z^{(0)}_1)\sigma''(z^{(0)}_1) + w^{(1)}_{22}\sigma'(z^{(1)}_2)[w^{(0)}_{21}]^2\sigma'(z^{(0)}_1)\sigma''(z^{(0)}_1)\\
&+ w^{(1)}_{23}\sigma'(z^{(1)}_3)[w^{(0)}_{31}]^2\sigma'(z^{(0)}_1)\sigma''(z^{(0)}_1)\Big\} + w^{(1)}_{21}\sigma'(z^{(1)}_1)w^{(0)}_{11}\sigma'''(z^{(0)}_1)\\
&+ w^{(1)}_{22}\sigma'(z^{(1)}_2)w^{(0)}_{21}\sigma'''(z^{(0)}_1) + w^{(1)}_{23}\sigma'(z^{(1)}_3)w^{(0)}_{31}\sigma'''(z^{(0)}_1)\Big]
\end{aligned}
\tag{3.23}
$$

Since the development of the third derivative is considerably longer than the second derivative, the detailed explanations about how to compute the expectations and the covariance terms are not provided. However, the logic behind those calculations remains the same as for the first and second derivatives. Consequently, (A.2) and (3.10) can always be used to compute expectations of products of consecutive or same layers and can be adapted for the number of terms to multiply together. Moreover, (A.3) can be referred to for covariance terms.

4 Risk Measures

In insurance and in finance, risk measures are used to quantify risk in situations with evidence of randomness. For example, in finance, risk measures are used to assess volatility of portfolio returns and in actuarial science, to capture the risk of large losses in claims predictive models.

4.1 Properties of Risk Measures

From Artzner et al. (1999), a risk measure is coherent if it satisfies translation invariance, subadditivity, positive homogeneity and monotonicity, which are defined below.

Let ρ be a risk measure and X and Y be random variables. Then, $\rho(X)$ is the risk measure ρ evaluated for X distribution and $\rho(Y)$ for Y.

1. Translation invariance: If $a \in \mathbb{R}$, then $\rho(X + a) = \rho(X) + a$
2. Subadditivity: $\rho(X + Y) \leq \rho(X) + \rho(Y)$
3. Positive homogeneity: If $\alpha \geq 0$, then $\rho(\alpha X) = \alpha\rho(X)$
4. Monotocity: If $X \leq Y$, then $\rho(Y) \geq \rho(X)$

4.2 Common Risk Measures In Actuarial Science

The most popular risk measures used in actuarial science are Value at Risk (VaR) and Tail Value at Risk (TVaR).

Let X be a random variable and F_X be its cumulative distribution function. For $0 \leq \alpha < 1$, VaR_α is the α-th quantile defined as follows,

$$\mathrm{VaR}_\alpha(X) = \inf\{x \in \mathbb{R}, F_X(x) \geq \alpha\}, \tag{4.1}$$

which represents the minimum value of X that should not be exceeded with probability α. Insurers are often interested in the potential extreme values for claims. Since they are in the right tail of the claim distribution, VaR_{95} could be an interesting measure to calculate, which corresponds to the claim cost that would not be exceeded 95% of the time. Now, if we are interested in the expected claim amount of the remaining 5%, we should calculate TVaR_{95}, where TVaR_α is defined for a continuous random variable X as follows,

$$\begin{aligned}\mathrm{TVaR}_\alpha(X) &= \mathbb{E}\left[X | X > \mathrm{VaR}_\alpha(X)\right] \\ &= \frac{1}{1-\alpha}\int_\alpha^1 \mathrm{VaR}_u(X)\, du. \end{aligned} \tag{4.2}$$

Now, Range Value at Risk (RVaR) is similar to TVaR with the difference that instead of taking the expectation of the domain at all points bigger than a given quantile VaR_α, it considers an upper bound that is set at VaR_{α^+}, where $0 \leq \alpha < \alpha^+ < 1$, as follows,

$$\begin{aligned}\mathrm{RVaR}_{\alpha,\alpha^+}(X) &= \mathbb{E}\left[X | \mathrm{VaR}_\alpha < X < \mathrm{VaR}_{\alpha^+}\right] \\ &= \frac{1}{\alpha^+ - \alpha}\int_\alpha^{\alpha^+} \mathrm{VaR}_u(X)\, du. \end{aligned} \tag{4.3}$$

If $\alpha = \alpha^+$, it would be equivalent to calculate VaR_α. Moreover, if $\alpha^+ = 1$, then we get TVaR_α. RVaR is an interesting measure to calculate in insurance if we are interested in the expected claim amount over a specific range of loss probabilities.

The Entropic Value at Risk (EVaR) and expectiles are two risk measures that can be considered as alternatives to VaR and TVaR.

Let $M_X(z)$ be the moment-generating function of X which exists for all $z \geq 0$. For $0 \leq \alpha < 1$, EVaR_α is defined as follows,

$$\text{EVaR}_\alpha(X) = \inf_{z<0} \left\{ z^{-1} \ln \left(\frac{M_X(z)}{1-\alpha} \right) \right\}. \tag{4·4}$$

For the same α, EVaR is the tightest upper bound to VaR and TVaR obtained from the Chernoff inequality, where $\text{VaR}(X) \leq \text{TVaR}(X) \leq \text{EVaR}(X)$. For any constant a, the Chernoff inequality by Chernoff et al. (1952) is defined as follows,

$$\text{P}(X \geq a) \leq e^{-za} M_X(z).$$

EVaR was introduced by Ahmadi-Javid (2012). The author showed that EVaR can be used to efficiently solve stochastic problems which are computationally intractable when using TVaR. In the literature, VaR and TVaR are often used in portfolio optimization. Ahmadi-Javid and Fallah-Tafti (2019) showed that EVaR approach performs similarly in terms of computational efficiency as TVaR approach and outperforms it as the sample size increases. Moreover, Firouzi and Luong (2014) applied EVaR to a portfolio optimization problem since that risk measure yielded an explicit formula for the objective function, which was not the case with VaR and TVaR, where numerical approximations were needed.

Now, expectiles $e_\alpha(X)$, introduced by Newey and Powell (1987), are defined as follows,

$$e_\alpha(X) = \underset{x\in\mathbb{R}}{\operatorname{argmin}}\ \alpha\mathbb{E}\left[(X-x)_+^2\right] + (1-\alpha)\mathbb{E}\left[(X-x)_-^2\right],$$

where $\alpha \in (0,1)$. If $\dfrac{\mathbb{E}\left[(X-x)_+^2\right]}{\mathbb{E}\left[(X-x)_-^2\right]} = \dfrac{1-\alpha}{\alpha}$, then x is the α-expectile for the distribution X. In finance and in insurance, this quantity can be interpreted as the amount of money to add in order to satisfy a given gain-loss ratio. Note that $e_{0.5}(X) = \mathbb{E}[X]$.

To conclude, TVaR, EVaR and RVaR are coherent risk measures as they satisfy all properties listed earlier. As stated by Bellini et al. (2014), $e_\alpha(X)$ is a coherent risk measure if $\alpha \geq 0.5$. Hovewer, VaR is not coherent since it does not satisfy the subadditivity property. Coherence is important since a portfolio can contain different types of assets or insurance products. Applying a given risk measure individually to the different assets or products should be consistent to applying it at the portfolio level. For example, for the subadditivity property, the risk of the entire portfolio should be smaller than the sum of the risk of each individual elements.

A risk measure for which verification and comparison of predictive performance of risk models can be achieved using a scoring function is considered elicitable. It is known that VaR and expectiles are elicitable, whereas TVaR is not. Since elicitability allows for backtesting, it can be interesting to use an elicitable risk measure even if it is not coherent.

4.3 Risk Measures in TAGI

In TAGI, all elements in a NN (weights, biases, nodes and outputs) follow a Normal distribution which capture the uncertainty around them. TAGI outputs a mean μ and a variance σ^2 for each predicted output. Therefore, it is possible to study risk measures applied to each predicted output. In addition, with all the usual transformations from activation functions in NN, the input variables being normalized and the outputs being denormalized at the end of the process, it is interesting to show if the risk measures still satisfy their properties, such as positive homogeneity, even after such transformations.

Since we are interested in risk measures applied to outputs that follow a Normal distribution, from (4.1) to (4.4), there exist closed-form formulas for VaR, TVaR, RVaR and EVaR which are calculated as follows,

$$\mathrm{VaR}_\alpha(X) = \mu + \sigma\Phi^{-1}(\alpha), \tag{4.5}$$

$$\mathrm{TVaR}_\alpha(X) = \mu + \sigma\frac{\phi[\Phi^{-1}(\alpha)]}{1-\alpha}, \tag{4.6}$$

$$\mathrm{RVaR}_{\alpha,\alpha^+}(X) = \mu + \sigma\frac{\phi[\Phi^{-1}(\alpha)] - \phi[\Phi^{-1}(\alpha^+)]}{\alpha^+ - \alpha}, \tag{4.7}$$

$$\mathrm{EVaR}_\alpha(X) = \mu + \sigma\sqrt{2\ln(1-\alpha)}, \tag{4.8}$$

where $\phi(x)$ and $\Phi(x)$ denote the probability and cumulative distribution functions, respectively, of a standard Normal distribution.

4.3.1 Simple Experiment with Risk Measures in TAGI

Consider a simple case where there are n input variables $X_1, X_2, ..., X_n$ and the response variable is $Y = \sum_{i=1}^{n} X_i$. Let $Y_1 = \sum_{i=1}^{n} X_i$ and $Y_2 = \sum_{i=1}^{n} cX_i = cY_1$. Because of the positive homogeneity property of risk measures, if $c \geq 0$, then for an homogeneous risk measure ρ, $\rho(Y_2) = c\rho(Y_1)$. We are interested to see if it is also the case when using $\{cX_i\}_{i=1}^{n}$ as inputs in TAGI.

To calculate VaR, TVaR, RVaR and EVaR for the outputs in TAGI, we first need to derive μ and σ which are used in (4.5) to (4.8). Let $A^{(j)}$ be the number of units in layer j and L be the number of hidden layers in a NN with ReLU as the activation function. We first need to standardize the input variables $\{X_i\}_{i=1}^{n}$. Let $S_i = \dfrac{X_i - \mu_i}{\sigma_i}$. We would be interested in cS_i.

To calculate the mean and variance, (2.2), (2.6) and (2.9) are used. With ReLU, using the linearization procedure from (2.10), the i^{th} activated unit of layer j is as follows,

$$\begin{aligned}
a_i^{(j)} &= \sigma(z_i^{(j)}) \\
&\approx \tilde{\sigma}(z_i^{(j)}) \\
&= \sigma(\mathbb{E}[z_i^{(j)}]) + \frac{\partial \sigma(\mathbb{E}[z_i^{(j)}])}{\partial z_i^{(j)}} \cancelto{0}{(z_i^{(j)} - \mathbb{E}[z_i^{(j)}])} \\
&= \max(0, \mathbb{E}[z_i^{(j)}]),
\end{aligned} \tag{4.9}$$

since linearization is done at $\mathbb{E}[z_i^{(j)}]$.

The mean of the i^{th} activated unit of layer j is as follows,

$$
\begin{aligned}
\mathbb{E}[a_i^{(j)}] &= \mathbb{E}[\max\left(0, \mathbb{E}[z_i^{(j)}]\right)] \\
&= \max\left(0, \mathbb{E}[z_i^{(j)}]\right).
\end{aligned}
\tag{4.10}
$$

The variance of the i^{th} activated unit of layer j can be found using Taylor series approximation and is as follows,

$$
\begin{aligned}
V(a_i^{(j)}) &= \left(\frac{\partial\sigma(\mathbb{E}[z_i^{(j)}])}{\partial z_i^{(j)}}\right)^2 V(z_i^{(j)}) \\
&= \mathbb{1}_{z_i^{(j)}>0} V(z_i^{(j)}).
\end{aligned}
\tag{4.11}
$$

Layer 1

For the first hidden layer, the mean and variance of the i^{th} hidden unit are as follows,

$$
\begin{aligned}
\mathbb{E}[z_i^{(1)}] &= \sum_{l=1}^{A^{(0)}} \mathbb{E}[w_{il}^{(0)}]\mathbb{E}[cS_l] + \mathbb{E}[b_i^{(0)}] \\
&= c\sum_{l=1}^{A^{(0)}} \mathbb{E}[w_{il}^{(0)}]S_l + \mathbb{E}[b_i^{(0)}],
\end{aligned}
\tag{4.12}
$$

$$
\begin{aligned}
V(z_i^{(1)}) &= \sum_{l=1}^{A^{(0)}} \left(V(w_{il}^{(0)})\cancelto{0}{V(cS_l)} + V(w_{il}^{(0)})\mathbb{E}[cS_l]^2 + \mathbb{E}[w_{il}^{(0)}]^2\cancelto{0}{V(cS_l)}\right) + V(b_i^{(0)}) \\
&= c^2\sum_{l=1}^{A^{(0)}} V(w_{il}^{(0)})S_l^2 + V(b_i^{(0)}).
\end{aligned}
\tag{4.13}
$$

From (4.10) and (4.11) and using (4.12) and (4.13), the mean and variance of the i^{th} activated unit are as follows,

$$\mathbb{E}[a_i^{(1)}] = \max\left(0, c\sum_{l=1}^{A^{(0)}} \mathbb{E}[w_{il}^{(0)}]S_l + \mathbb{E}[b_i^{(0)}]\right), \tag{4.14}$$

$$V(a_i^{(1)}) = \mathbb{1}_{z_i^{(1)}>0}\left\{c^2\sum_{l=1}^{A^{(0)}} V(w_{il}^{(0)})S_l^2 + V(b_i^{(0)})\right\}. \tag{4.15}$$

Layer 2

From the first hidden layer, (4.14) and (4.15) are used to calculate the mean and variance of the i^{th} hidden unit for a second layer, which are as follows,

$$\begin{aligned}\mathbb{E}[z_i^{(2)}] &= \sum_{l=1}^{A^{(1)}} \mathbb{E}[w_{il}^{(1)}]\mathbb{E}[a_l^{(1)}] + \mathbb{E}[b_i^{(1)}] \\ &= \sum_{l=1}^{A^{(1)}} \mathbb{E}[w_{il}^{(1)}]\left(\max(0, c\sum_{k=1}^{A^{(0)}} \mathbb{E}[w_{lk}^{(0)}]S_k + \mathbb{E}[b_l^{(0)}])\right) + \mathbb{E}[b_i^{(1)}],\end{aligned} \tag{4.16}$$

$$\begin{aligned}V(z_i^{(2)}) &= \sum_{l=1}^{A^{(1)}} \Big(V(w_{il}^{(1)})V(a_l^{(1)}) + V(w_{il}^{(1)})\mathbb{E}[a_l^{(1)}]^2 + \mathbb{E}[w_{il}^{(1)}]^2 V(a_l^{(1)})\Big) + V(b_i^{(1)}) \\ &= \sum_{l=1}^{A^{(1)}} \Bigg[\mathbb{1}_{z_l^{(1)}>0}\left\{c^2\sum_{k=1}^{A^{(0)}} V(w_{lk}^{(0)})S_k^2 + V(b_l^{(0)})\right\}\Big(V(w_{il}^{(1)}) + \mathbb{E}[w_{il}^{(1)}]^2\Big) \\ &\quad + V(w_{il}^{(1)})\max\left(0, c\sum_{k=1}^{A^{(0)}} \mathbb{E}[w_{lk}^{(0)}]S_k + \mathbb{E}[b_l^{(0)}]\right)^2\Bigg] + V(b_i^{(1)}).\end{aligned} \tag{4.17}$$

From (4.10) and (4.11) and using (4.16) and (4.17), the mean and variance of the i^{th} activated unit are as follows,

$$\mathbb{E}[a_i^{(2)}] = \max(0, \sum_{l=1}^{A^{(1)}} \mathbb{E}[w_{il}^{(1)}] \left(\max(0, c \sum_{k=1}^{A^{(0)}} \mathbb{E}[w_{lk}^{(0)}] S_k + \mathbb{E}[b_l^{(0)}]) \right) + \mathbb{E}[b_i^{(1)}]),$$

$$\begin{aligned} V(a_i^{(2)}) = \mathbb{1}_{z_i^{(2)}>0} \Bigg\{ \sum_{l=1}^{A^{(1)}} \Bigg[\mathbb{1}_{z_l^{(1)}>0} \left\{ c^2 \sum_{k=1}^{A^{(0)}} V(w_{lk}^{(0)}) S_k^2 + V(b_l^{(1)}) \right\} \\ \Big(V(w_{il}^{(1)}) + \mathbb{E}[w_{il}^{(1)}]^2 \Big) + V(w_{il}^{(1)}) \max \left(0, c \sum_{k=1}^{A^{(0)}} \mathbb{E}[w_{lk}^{(0)}] S_k + \mathbb{E}[b_l^{(0)}] \right)^2 \Bigg] \\ + V(b_i^{(1)}) \Bigg\}. \end{aligned}$$

Output Layer

For the output layer, the mean of the response can be expressed as follows,

$$\begin{aligned} \mathbb{E}[g(Y_2^*)] = \sum_{l=1}^{A^{(L)}} \Bigg[\mathbb{E}[w_{1l}^{(L)}] \cdots \max\Big(0, \sum_{k=1}^{A^{(1)}} \mathbb{E}[w_{tk}^{(1)}] \Big(\max(0, c \sum_{m=1}^{A^{(0)}} \mathbb{E}[w_{km}^{(0)}] S_m \\ + \mathbb{E}[b_k^{(0)}]) \Big) + \mathbb{E}[b_t^{(1)}] \Big) \cdots \Bigg] + \mathbb{E}[b_1^{(L)}], \end{aligned} \tag{4.18}$$

where Y_2^* represents the response variable prior to the denormalization step. However, note that the mean and variance of the activated units of the last hidden layer L can be expressed in a generalized form, as follows,

$$f^{(L)}(f^{(L-1)}(\cdots f^{(2)}(f^{(1)}(c\boldsymbol{S}))\cdots)), \tag{4.19}$$

where for the mean, at layer j, (4.19) is rewritten as follows,

$$m^{(j)}(\boldsymbol{a}) = \max\left(0, \sum_{l=1}^{A^{(j-1)}} \mathbb{E}[w_{il}^{(j-1)}]\mathbb{E}[a_l^{(j-1)}] + \mathbb{E}[b_i^{(j-1)}]\right), \tag{4.20}$$

where i is the i^{th} activated units of layer j. For the variance, at layer j, (4.19) is rewritten as follows,

$$\begin{aligned} v^{(j)}(\boldsymbol{a}) = \mathbb{1}_{z_i^{(j)}>0}\Bigg\{ \sum_{l=1}^{A^{(j-1)}} \Big(& V(w_{il}^{(j-1)})V(a_l^{(j-1)}) + V(w_{il}^{(j-1)})\mathbb{E}[a_l^{(j-1)}]^2 \\ & +\mathbb{E}[w_{il}^{(j-1)}]^2 V(a_l^{(j-1)})\Big) + V(b_i^{(j-1)})\Bigg\}. \end{aligned} \tag{4.21}$$

Now, because there is no activation at the output layer, expressing the mean and the variance of the response using (4.20) and (4.21) results as follows,

$$\begin{aligned} \mathbb{E}[g(Y_2^*)] &= \sum_{l=1}^{A^{(L)}} \mathbb{E}[w_{1l}^{(L)}]m^{(L)}(m^{(L-1)}(\cdots m^{(2)}(m^{(1)}(c\boldsymbol{S}))\cdots)) + \mathbb{E}[b_1^{(L)}], \\ V(g(Y_2^*)) &= \sum_{l=1}^{A^{(L)}} \Big(v^{(L)}(v^{(L-1)}(\cdots v^{(2)}(v^{(1)}(c\boldsymbol{S}))\cdots)) \Big(V(w_{1l}^{(L)}) + \mathbb{E}[w_{1l}^{(L-1)}]^2\Big) \\ &\quad +V(w_{1l}^{(L)})(m^{(L)}(m^{(L-1)}(\cdots m^{(2)}(m^{(1)}(c\boldsymbol{S}))\cdots)))^2\Big) + V(b_1^{(L)}). \end{aligned}$$

Now, even before the denormalization step, when using $\{cX_i\}_{i=1}^n$ as inputs, we see that the positive homogeneity property of risk measure cannot be followed because of the bias terms which are not multiplied by c. For example, in the simple case where $L = 1$, the mean and variance of Y_2^* are as follows,

$$\begin{aligned}
\mathbb{E}[g(Y_2^*)] &= \sum_{l=1}^{A^{(1)}} \mathbb{E}[w_{1l}^{(1)}] \left(\max(0, c\sum_{k=1}^{A^{(0)}} \mathbb{E}[w_{lk}^{(0)}]S_k + \mathbb{E}[b_l^{(0)}]) \right) + \mathbb{E}[b_1^{(1)}], \\
V(g(Y_2^*)) &= \sum_{l=1}^{A^{(1)}} \left[\mathbb{1}_{z_l^{(1)}>0} \left\{ c^2 \sum_{k=1}^{A^{(0)}} V(w_{lk}^{(0)})S_k^2 + V(b_l^{(1)}) \right\} \left(V(w_{il}^{(1)}) + \mathbb{E}[w_{1l}^{(1)}]^2 \right) \right. \\
&\quad \left. + V(w_{1l}^{(1)}) \max\left(0, c\sum_{k=1}^{A^{(0)}} \mathbb{E}[w_{lk}^{(0)}]S_k + \mathbb{E}[b_l^{(0)}]\right)^2 \right] + V(b_1^{(1)}),
\end{aligned}$$

whereas the mean and variance of Y_1^* are as follows,

$$\begin{aligned}
\mathbb{E}[g(Y_1^*)] &= \sum_{l=1}^{A^{(1)}} \mathbb{E}[w_{1l}^{(1)}] \left(\max(0, \sum_{k=1}^{A^{(0)}} \mathbb{E}[w_{lk}^{(0)}]S_k + \mathbb{E}[b_l^{(0)}]) \right) + \mathbb{E}[b_1^{(1)}], \\
V(g(Y_1^*)) &= \sum_{l=1}^{A^{(1)}} \left[\mathbb{1}_{z_l^{(1)}>0} \left\{ \sum_{k=1}^{A^{(0)}} V(w_{lk}^{(0)})S_k^2 + V(b_l^{(1)}) \right\} \left(V(w_{il}^{(1)}) + \mathbb{E}[w_{1l}^{(1)}]^2 \right) \right. \\
&\quad \left. + V(w_{1l}^{(1)}) \max\left(0, \sum_{k=1}^{A^{(0)}} \mathbb{E}[w_{lk}^{(0)}]S_k + \mathbb{E}[b_l^{(0)}]\right)^2 \right] + V(b_1^{(1)}).
\end{aligned}$$

One sees that $\rho(g(Y_2^*))$ does not equal to $c\rho(g(Y_1^*))$. Moreover, in the case where for $z_l^{(1)}$, a hidden unit l in layer 1, $\sum_{k=1}^{A^{(0)}} \mathbb{E}[w_{lk}^{(0)}]S_k$ and $\mathbb{E}[b_l^{(0)}]$ are of opposite signs (i.e. one term is negative and the other is positive), then multiplying $\sum_{k=1}^{A^{(0)}} \mathbb{E}[w_{lk}^{(0)}]S_k$ by $c > 0$ can result in the original $z_l^{(1)}$ to change sign, which can completely change the outcome at activation using ReLU. With more than one hidden layer, such modifications to the first layer units when using $\{cX_i\}_{i=1}^n$ as inputs create a chain reaction, since units in layer j are calculated based on units from layer $j-1$.

From (4.5) to (4.8), VaR, TVaR, RVaR and EVaR are as follows,

$$\begin{aligned}
\mathrm{VaR}_\alpha(g(Y_2^*)) &= \mathbb{E}[g(Y_2^*)] + \sqrt{V(g(Y_2^*))}\Phi^{-1}(\alpha) \\
&\neq c\mathbb{E}[g(Y_1^*)] + c\sqrt{V(g(Y_1^*))}\Phi^{-1}(\alpha) = c\mathrm{VaR}_\alpha(g(Y_1^*)), \\
\mathrm{TVaR}_\alpha(g(Y_2^*)) &= \mathbb{E}[g(Y_2^*)] + \sqrt{V(g(Y_2^*))}\frac{\phi[\Phi^{-1}(\alpha)]}{1-\alpha} \\
&\neq c\mathbb{E}[g(Y_1^*)] + c\sqrt{V(g(Y_1^*))}\frac{\phi[\Phi^{-1}(\alpha)]}{1-\alpha} = c\mathrm{TVaR}_\alpha(g(Y_1^*)), \\
\mathrm{RVaR}_{\alpha,\alpha^+}(g(Y_2^*)) &= \mathbb{E}[g(Y_2^*)] + \sqrt{V(g(Y_2^*))}\frac{\phi[\Phi^{-1}(\alpha)] - \phi[\Phi^{-1}(\alpha^+)]}{\alpha^+ - \alpha} \\
&\neq c\mathbb{E}[g(Y_1^*)] + c\sqrt{V(g(Y_1^*))}\frac{\phi[\Phi^{-1}(\alpha)] - \phi[\Phi^{-1}(\alpha^+)]}{\alpha^+ - \alpha} \\
&= c\mathrm{RVaR}_{\alpha,\alpha^+}(g(Y_1^*)), \\
\mathrm{EVaR}_\alpha(g(Y_2^*)) &= \mathbb{E}[g(Y_2^*)] + \sqrt{V(g(Y_2^*))}\sqrt{2\ln(1-\alpha)} \\
&\neq c\mathbb{E}[g(Y_1^*)] + c\sqrt{V(g(Y_1^*))}\sqrt{2\ln(1-\alpha)} = c\mathrm{EVaR}_\alpha(g(Y_1^*)).
\end{aligned}$$

Furthermore, the mean and variance of the response variable from the training set are used at the denormalization step as follows,

$$\begin{aligned}
\mathbb{E}[g(Y_2)] &= \mathbb{E}[g(Y_2^*)]\sqrt{V(Y_{train})} + \mathbb{E}[Y_{train}], \\
V(g(Y_2)) &= V(g(Y_2^*))V(Y_{train}).
\end{aligned}$$

Since, in theory, we do not know the underlying function predicted by the NN (i.e. in our simple example, training is done such that weights and biases are optimized to replicate $Y = \sum_{i=1}^{n} X_i$, but the model does not explicitly calculates $Y = \sum_{i=1}^{n} X_i$), we cannot use cY_{train} for denormalization. Thus, $\rho(g(Y_2)) \neq c\rho(g(Y_1))$ when using $\{cX_i\}_{i=1}^{n}$ as inputs.

4.3.2 Precipitations Example

Daily previsions on precipitations for many geographic locations from 24 experts are available from CRIM (2021) and the real data is from Muñoz-Sabater (2019). Both previsions and real data are available from 1981 up to early 2021, so we can train and test models during that period of time. Moreover, daily previsions are available up to 2100 for almost all experts, so we can predict precipitations in the future using NN trained on the 40 years of previsions and precipitations we have. All experiments are done at the same coordinates and the units are milliliters.

To continue exploring risk measures with TAGI, we show how that technique performs in predicting quantiles. Since TAGI performs well when there is few observations, we consider modeling on an annual basis. To compare with models that contain more data, we also show models that are on a monthly basis. Even though real data for early 2021 is available, we use up to December 31^{st}, 2020 data to calculate risk measures that are based on complete years. We built models for the 50^{th}, 60^{th}, 70^{th}, 80^{th}, 90^{th}, 95^{th} and 100^{th} quantiles, as well as for the mean. Lower quantiles are not shown since they consist in very low values (approaching zero, if not zero), which is not a case of interest. For a given model, the input variables are the quantile of each of the 24 experts and the output variable is the real data's quantile on the desired time span basis. In other words, for a given year and for each expert, we have approximately 365 daily previsions, so we can calculate the quantiles from those values for each expert. The same principle applies for the monthly models, where instead of having approximately 365 daily precipitations, the quantiles are calculated over 30 values. We use the same method to calculate the observed precipitations quantiles. For example, the VaR_{100} annual model is fed by 40 observations which each contains the maximum daily prevision over a year from each expert. That model predicts the maximum daily precipitations in any given years.

As for the insurance example, we use a NN with one hidden layer of 50 neurons with ReLU activation function. There are 40 epochs and one split. Testing is done on 20% of the observations, which are out-of-sample. For different quantiles and for the mean, the mean-square errors (MSE) and its bias-variance decomposition as well as the root-mean-square errors (RMSE) for the testing set are as follows,

	Annual				Monthly			
	Bias^2	$V(\hat{Y})$	MSE	RMSE	$Bias^2$	$V(\hat{Y})$	MSE	RMSE
VaR$_{100}$	114.6	136.6	250.2	15.8	162.1	77.8	239.9	15.5
VaR$_{95}$	5.6	0.5	6.1	2.5	48.3	28.7	77.0	8.8
VaR$_{90}$	1.1	3.9	5.0	2.2	26.2	13.6	39.7	6.3
VaR$_{80}$	0.8	0.1	0.9	1.0	10.6	4.5	15.0	3.9
VaR$_{70}$	0.1	0.0	0.2	0.4	2.9	1.4	4.3	2.1
VaR$_{60}$	0.0	0.0	0.1	0.2	0.9	0.5	1.4	1.2
VaR$_{50}$	0.0	0.0	0.0	0.1	0.3	0.2	0.5	0.7
Mean	0.1	0.0	0.1	0.4	1.5	1.0	2.5	1.6

Table 4.1: Quantile Models' Bias^2, Variance, MSE and RMSE

The annual models achieve better accuracy than the monthly ones, except when predicting the maximum. High $V(\hat{Y})$ can be an evidence of overfitting, which seems to be the case for the annual model predicting the maximum. High Bias^2 can be an evidence of underfitting, which seems to be the case in monthly models. Monthly risk measures are calculated on around 30 observations (which is 12 times less than with annual measures) so it is more likely to diverge from the real quantity. Moreover, for both time spans, accuracy improves as quantiles decrease, which is not surprising as extreme values are generally more difficult to predict than values close to the mean. However, the annual models perform well for high quantiles, achieving low RMSE levels, which are only met starting at the 70^{th} quantiles for the monthly models.

Now, it is interesting to compare performance with a different method. Since our focus is on risk measures, quantile regression is an appropriate method for comparison purposes. Instead of estimating the mean of the response variable (as for least-squares regression), it estimates its quantiles. It is commonly used to predict the median, as an alternative to linear regression.

Previously, the x^{th} quantile from each expert was used to predict the real x^{th} quantile. In quantile regression, the response variable's quantiles are predicted using a single set of input variables. Therefore, let the mean predictions from each expert be used as inputs to both methods to predict all chosen quantiles. In other words, for a given year and for each expert, we have approximately 365 daily previsions, so we can calculate the mean from those values for each expert. The same principle applies for the monthly models, where instead of having approximately 365 daily precipitations, the mean is calculated over 30 values. For NN, there is a different model for each of the quantiles to predict, but each is trained using the same mean predictions. Both approaches are built on 80% of the data. RMSE is calculated on the remaining 20% using the real observed quantiles calculated previously. For the same quantiles shown in Table 4.1, the results for both methods are as follows,

	Annual		Monthly	
	QR	NN	QR	NN
VaR_{100}	40.6	12.0	23.0	15.0
VaR_{95}	13.1	2.4	12.4	8.5
VaR_{90}	6.8	1.4	7.6	6.7
VaR_{80}	1.6	0.8	3.5	4.0
VaR_{70}	1.1	0.4	2.1	2.1
VaR_{60}	2.2	0.2	2.3	1.2
VaR_{50}	2.7	0.1	2.5	0.7

Table 4.2: RMSE under Quantile Regression (QR) and Neural Network (NN)

According to Table 4.2, for both annual and monthly time spans, NN models generally perform better than quantile regression, particularly in predicting extreme values. NN learns from each real response's quantile whereas QR's estimates are only based from responses' mean. However, the difference is less pronounced at quantiles closer to the median.

Regarding NN only, as mentioned previously, the difference between experiments in Table 4.1 and Table 4.2 is that the first one uses the x^{th} quantile from each expert as inputs whereas the second one uses the mean to predict the real x^{th} quantile. The same conclusions are drawn for the last experiment, which are that annual models perform better than monthly ones (especially for extreme values) and that accuracy improves as quantiles decrease. Moreover, RMSE at each quantile for both time span models are similar.

5 Conclusion

The initial purpose of this thesis was to explore techniques which predict output risk distribution (not only mean response) and apply them to insurance premium pricing to improve risk assessment. Quantile regression (QR) is a well-known method for doing so by predicting response's quantiles. More recently, several machine learning techniques are studied, developed and improved and some of them provide more information about the predicted responses. Notably, methods from Gradient Boosting (GB) and Neural Networks (NN) achieve that objective. Tractable Approximate Gaussian Inference (TAGI) by Goulet et al. (2020) is extensively reviewed in this thesis, as for all the model's parameters, the predicted responses follow a Normal distribution which, not only answers to the initial purpose of this thesis, but also opens up opportunities for diverse areas of application.

Thus, it was interesting to compare general performance of TAGI to a GB technique on an insurance dataset. Extreme Gradient Boosting (XGBoost) by Chen and Guestrin (2016) was chosen for its ability to model large amounts of data, efficiency and popularity of use. From Table 2.2, for similar parameters, TAGI performs generally better than XGBoost, where results are more comparable when tree booster is used rather then the linear one.

Furthermore, since all model's parameters and states follow a Normal distribution and also because the outputs are functions of the inputs, partial derivatives, which are commonly used in sensitivity analysis, can be estimated using TAGI. Detailed explanations on the first and second derivatives are provided in Section 3 as well as the formulas for the third one which are developed and shown.

Moreover, risk measures study was not yet performed on TAGI and since the outputs follow a Normal distribution, analyzing such measures on them is possible. In a simple experiment, we demonstrate layer by layer how modifying the inputs have an impact on the resulting distribution of the response and hence, on the risk measures. We also use TAGI to build models to predict quantiles of daily total precipitations on annual and monthly time spans, from which we compare to QR's performance. As shown in Table 4.2, NN models generally performs better than QR, particularly when predicting extreme values. However, the difference is less pronounced at quantiles closer to the median.

Lastly, TAGI was initially developed in Matlab[11] programming language. We developed a comprehensive package in R. As the major contribution from this thesis, the package "tagi"[12] contains functions to perform TAGI on any datasets and functions to compute the first and second derivatives. Vignettes to guide the users through the package and a reference manual are also provided. The other contributions are all the work around TAGI discussed previously such as comparing TAGI's general performance to XG-Boost, developing the third partial derivative and studying risk measures on TAGI which implies testing and comparing it at predicting quantiles.

[11] Available at https://github.com/CivML-PolyMtl/TAGI.

[12] Available at https://github.com/mgoulet847/tagi. Manual is included in Appendix D.

To conclude, further research can include:

1. Test TAGI models' accuracy and performance at predicting extreme values. With climate change, more natural disasters are expected, which impacts the insurance industry.

2. Develop a premium pricing model using TAGI based on insurers' real (and comprehensive) data and then use backtesting to assess the quality of the model. It would be beneficial for the insurance industry to explore different techniques than the traditional generalized linear models (GLM) for example.

3. Develop and implement higher order partial derivatives. The complexity and the number of calculations to perform, such as expectations of products of variables and covariance terms, increase at each order derivative, which also increases programming challenges and computation time substantially. Derivatives are used in sensitivity analysis but also in engineering problems, where the first derivative corresponds to speed, the second derivative to acceleration and the third derivative to the rate at which acceleration changes.

References

A. Ahmadi-Javid. Entropic value-at-risk: A new coherent risk measure. *Journal of Optimization Theory and Applications*, 155(3):1105–1123, 2012.

A. Ahmadi-Javid and M. Fallah-Tafti. Portfolio optimization with entropic value-at-risk. *European Journal of Operational Research*, 279(1):225–241, 2019.

P. Artzner, F. Delbaen, J.-M. Eber, and D. Heath. Coherent measures of risk. *Mathematical finance*, 9(3):203–228, 1999.

F. Bellini, B. Klar, A. Müller, and E. R. Gianin. Generalized quantiles as risk measures. *Insurance: Mathematics and Economics*, 54:41–48, 2014.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, M. Li, J. Xie, M. Lin, Y. Geng, and Y. Li. *xgboost: Extreme Gradient Boosting*, 2020. URL `https://CRAN.R-project.org/package=xgboost`. R package version 1.2.0.1.

H. Chernoff et al. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952.

CRIM. Daily total precipitation at (-73.6,45.5), 2021. Retrieved May 26, 2021, from `https://climatedata.ca/`.

N. Dhieb, H. Ghazzai, H. Besbes, and Y. Massoud. Extreme gradient boosting machine learning algorithm for safe auto insurance operations. In *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–5. IEEE, 2019.

A. Farzad, H. Mashayekhi, and H. Hassanpour. A comparative performance analysis of different activation functions in lstm networks for classification. *Neural Computing and Applications*, 31(7):2507–2521, 2019.

H. O. Firouzi and A. Luong. Optimal portfolio problem using entropic value at risk: When the underlying distribution is non-elliptical. *arXiv preprint arXiv:1406.7040*, 2014.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *International conference on machine learning*, pages 1319–1327. PMLR, 2013.

J.-A. Goulet, L. H. Nguyen, and S. Amiri. Tractable approximate Gaussian inference for bayesian neural networks. *arXiv preprint arXiv:2004.09281*, 2020.

L. Guelman. Gradient boosting trees for auto insurance loss cost modeling and prediction. *Expert Systems with Applications*, 39(3):3659–3667, 2012.

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

R. A. Johnson, D. W. Wichern, et al. *Applied multivariate statistical analysis*, volume 5. Prentice hall Upper Saddle River, NJ, 2002.

L. V. Jospin, W. Buntine, F. Boussaid, H. Laga, and M. Bennamoun. Hands-on bayesian neural networks–a tutorial for deep learning users. *arXiv preprint arXiv:2007.06823*, 2020.

B. Lantz. *Machine learning with R.* Packt publishing ltd, 2013.

J. Muñoz-Sabater. Era5-land hourly data from 1981 to present, 2019. Copernicus Climate Change Service (C3S) Climate Data Store (CDS). (Accessed on 26-05-2021), 10.24381/cds.e2161bac.

W. K. Newey and J. L. Powell. Asymmetric least squares estimation and testing. *Econometrica: Journal of the Econometric Society*, pages 819–847, 1987.

C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

S. M. Pesenti, P. Millossovich, and A. Tsanakas. Cascade sensitivity measures. *Available at SSRN 3270839*, 2020.

J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

H. E. Rauch, F. Tung, and C. T. Striebel. Maximum likelihood estimates of linear dynamic systems. *AIAA journal*, 3(8):1445–1450, 1965.

L. Rdusseeun and P. Kaufman. Clustering by means of medoids, 1987.

S. Sharma and S. Sharma. Activation functions in neural networks. *Towards Data Science*, 6(12):310–316, 2017.

J. Turian, J. Bergstra, and Y. Bengio. Quadratic features and deep architectures for chunking. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 245–248, 2009.

H. Zheng, Z. Yang, W. Liu, J. Liang, and Y. Li. Improving deep neural networks using softplus units. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–4. IEEE, 2015.

Appendices

A First Derivative with Two Hidden Layers

Let $\mathbb{E}\left[V_1V_2V_3V_4V_5V_6\right]$, where $V_1 = \mathbf{W}^{(2)}$, $V_2 = \sigma'(\mathbf{z}^{(2)})$, $V_3 = \mathbf{W}^{(1)}$, $V_4 = \sigma'(\mathbf{z}^{(1)})$, $V_5 = \mathbf{W}^{(0)}$ and $V_6 = \sigma'(\mathbf{z}^{(0)})$. For implementation, we proceed by layer, starting from the last hidden layer.

In layer 2, there are two expectations to compute, which are $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)\right]$ and $\mathbb{E}\left[w_{12}^{(2)}\sigma'\left(z_2^{(2)}\right)\right]$. They can be calculated as follows,

$$\mathbb{E}\left[V_1V_2\right] = \mathbb{E}\left[V_1\right]\mathbb{E}\left[V_2\right] + \cancelto{0}{\operatorname{cov}(V_1, V_2)}, \qquad (A.1)$$

where V_1 and V_2 are independent because there is no dependence between a given node and its weights.

In layer 1, there are six expectations to compute, which are
$\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)w_{11}^{(1)}\sigma'\left(z_1^{(1)}\right)\right]$, $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)w_{12}^{(1)}\sigma'\left(z_2^{(1)}\right)\right]$,
$\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)w_{13}^{(1)}\sigma'\left(z_3^{(1)}\right)\right]$, $\mathbb{E}\left[w_{12}^{(2)}\sigma'\left(z_2^{(2)}\right)w_{21}^{(1)}\sigma'\left(z_1^{(1)}\right)\right]$,
$\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_2^{(2)}\right)w_{22}^{(1)}\sigma'\left(z_2^{(1)}\right)\right]$ and $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_2^{(2)}\right)w_{23}^{(1)}\sigma'\left(z_3^{(1)}\right)\right]$.

They can be calculated as follows,

$$
\begin{aligned}
\mathbb{E}\left[V_1V_2V_3V_4\right] &= \mathbb{E}\left[V_1V_2\right]\mathbb{E}\left[V_3V_4\right] + \mathrm{cov}(V_1V_2, V_3V_4) \\
&= \left(\mathbb{E}\left[V_1\right]\mathbb{E}\left[V_2\right] + \cancelto{0}{\mathrm{cov}(V_1, V_2)}\right) \\
&\quad \times \left(\mathbb{E}\left[V_3\right]\mathbb{E}\left[V_4\right] + \cancelto{0}{\mathrm{cov}(V_3, V_4)}\right) \\
&\quad + \mathrm{cov}(V_1V_2, V_3V_4) \\
&= \mathbb{E}\left[V_1\right]\mathbb{E}\left[V_2\right]\mathbb{E}\left[V_3\right]\mathbb{E}\left[V_4\right] + \mathrm{cov}(V_1V_2, V_3V_4).
\end{aligned} \tag{A.2}
$$

Using (2.7) in (2.8) to rewrite $\mathrm{cov}(V_1V_2, V_3V_4)$ as follows,

$$
\begin{aligned}
\mathrm{cov}(V_1V_2, V_3V_4) &= \mathrm{cov}(V_1, V_3)\mathrm{cov}(V_2, V_4) + \mathrm{cov}(V_1, V_4)\mathrm{cov}(V_2, V_3) \\
&\quad + \mathrm{cov}(V_1, V_3)\mu_2\mu_4 + \mathrm{cov}(V_1, V_4)\mu_2\mu_3 \\
&\quad + \mathrm{cov}(V_2, V_3)\mu_1\mu_4 + \mathrm{cov}(V_2, V_4)\mu_1\mu_3 \\
&= \mathrm{cov}(V_1, V_3)\mathrm{cov}(V_2, V_4) + \mathrm{cov}(V_1, V_4)\mathrm{cov}(V_2, V_3) \\
&\quad + \mu_2\left(\mathrm{cov}(V_1, V_3)\mu_4 + \mathrm{cov}(V_1, V_4)\mu_3\right) \\
&\quad + \mu_1\left(\mathrm{cov}(V_2, V_3)\mu_4 + \mathrm{cov}(V_2, V_4)\mu_3\right) \\
&= \mathrm{cov}(V_1, V_3)\mathrm{cov}(V_2, V_4) + \mathrm{cov}(V_1, V_4)\mathrm{cov}(V_2, V_3) \\
&\quad + \mu_2\mathrm{cov}(V_1, V_3V_4) + \mu_1\mathrm{cov}(V_2, V_3V_4).
\end{aligned} \tag{A.3}
$$

Applying $(A.3)$ to $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)w_{11}^{(1)}\sigma'\left(z_1^{(1)}\right)\right]$ to demonstrate $\mathrm{cov}(V_1V_2, V_3V_4)$ in $(A.2)$,

$$
\begin{aligned}
\mathrm{cov}(V_1V_2, V_3V_4) &= \cancelto{0}{\mathrm{cov}\left(w_{11}^{(2)}, w_{11}^{(1)})\right)}\mathrm{cov}\left(\sigma'(z_1^{(2)}), \sigma'(z_1^{(1)}))\right) \\
&\quad + \cancelto{0}{\mathrm{cov}\left(w_{11}^{(2)}, \sigma'(z_1^{(1)}))\right)}\mathrm{cov}\left(\sigma'(z_1^{(2)}), w_{11}^{(1)})\right) \\
&\quad + \mathbb{E}\left[w_{11}^{(2)}\right]\ \mathrm{cov}\left(\sigma'(z_1^{(2)}), w_{11}^{(1)}\sigma'(z_1^{(1)})\right) \\
&\quad + \mathbb{E}\left[\sigma'(z_1^{(2)})\right]\ \cancelto{0}{\mathrm{cov}\left(w_{11}^{(2)}, w_{11}^{(1)}\sigma'(z_1^{(1)})\right)} \\
&= \mathbb{E}\left[w_{11}^{(2)}\right]\ \mathrm{cov}(d_1^+, w_{11}^{(1)}d_1).
\end{aligned}
$$

Now, covariance calculation between derivatives depends on the activation function that is used. Formulas for sigmoid, tanh and ReLU functions are detailed in Appendix C. For generalization purposes, covariance can be rewritten as follows,

$$
\mathrm{cov}(V_1V_2, V_3V_4) = \mathbb{E}\left[\mathbf{W}^+\right]\ \mathrm{cov}(\mathbf{d}^+, \mathbf{W}\mathbf{d}). \tag{A.4}
$$

Then, remains the calculations in the input layer to get $\dfrac{\partial g}{\partial z_1^{(0)}}$ from (3.8).

$$
\begin{aligned}
\mathbb{E}\left[V_1V_2V_3V_4V_5V_6\right] &= \mathbb{E}\left[V_1V_2V_3V_4\right]\mathbb{E}\left[V_5V_6\right] + \mathrm{cov}(V_1V_2V_3V_4, V_5V_6) \\
&= \mathbb{E}\left[V_1V_2V_3V_4\right]\mathbb{E}\left[V_5V_6\right] \\
&\quad + \mathbb{E}\left[V_1V_2\right]\mathrm{cov}(V_3V_4, V_5V_6) \\
&\quad + \mathbb{E}\left[V_3V_4\right]\cancelto{0}{\mathrm{cov}(V_1V_2, V_5V_6)} \\
&= \mathbb{E}\left[V_1V_2V_3V_4\right]\mathbb{E}\left[V_5V_6\right] \\
&\quad + \mathbb{E}\left[V_1V_2\right]\mathrm{cov}(V_3V_4, V_5V_6),
\end{aligned} \tag{A.5}
$$

where $\mathbb{E}\left[V_5 V_6\right]$ and $\mathrm{cov}(V_3 V_4, V_5 V_6)$ can be calculated using $(A.1)$ and $(A.4)$ respectively. Note that $V_1 V_2$ (from layer 2) and $V_5 V_6$ (from layer 0) are independent because of the inherent conditional independence of hidden units between layers $\boldsymbol{Z}^{(j-1)} \perp\!\!\!\perp \boldsymbol{Z}^{(j+1)} | \boldsymbol{z}^{(j)}$.

B Second Derivative with Two Hidden Layers

As for the first derivative of g, expectations of products terms must be computed. With two hidden layers, there are three possible types of sequence of product terms, one from $\dfrac{\partial^2 g}{(\partial \boldsymbol{z}^{(2)})^2}\left(\dfrac{\partial \boldsymbol{z}^{(2)}}{\partial z_1^{(0)}}\right)^2$ and two from $\dfrac{\partial g}{\partial \boldsymbol{z}^{(2)}}\dfrac{\partial^2 \boldsymbol{z}^{(2)}}{(\partial z_1^{(0)})^2}$, which are shown in Table B.1.

	Layer 2	Layer 1	Layer 0
1	2 × 2nd order	4 × 1st order	4 × 1st order
2	2 × 1st order	2 × 2nd order	4 × 1st order
3	2 × 1st order	2 × 1st order	2 × 2nd order

Table B.1: Sequence Types of Product Terms

B.1 Sequence Type 1

We first start with $\dfrac{\partial^2 g}{(\partial \boldsymbol{z}^{(2)})^2}\left(\dfrac{\partial \boldsymbol{z}^{(2)}}{\partial z_1^{(0)}}\right)^2$ from (3.14) which is equal to

$$\begin{aligned}
& w_{11}^{(2)}\sigma''(z_1^{(2)}) \\
& \left\{w_{11}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)}) + w_{12}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)}) + w_{13}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma'(z_1^{(0)})\right\}^2 \\
& + w_{12}^{(2)}\sigma''(z_2^{(2)}) \\
& \left\{w_{21}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma'(z_1^{(0)}) + w_{22}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma'(z_1^{(0)}) + w_{23}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma'(z_1^{(0)})\right\}^2 .
\end{aligned}$$

Implementation is, as for the first derivative, done by layer from the last hidden layer to the input layer.

In layer 2, there are two expectations to compute which are $\mathbb{E}\left[w_{11}^{(2)}\sigma''\left(z_1^{(2)}\right)\right]$ and $\mathbb{E}\left[w_{12}^{(2)}\sigma''\left(z_2^{(2)}\right)\right]$. They can be calculated using $(A.1)$.

In layer 1, developing $\left(\dfrac{\partial z_1^{(2)}}{\partial z_1^{(0)}}\right)^2$ results in multiplying $w_{11}^{(2)}\sigma''(z_1^{(2)})$ to

$$\begin{aligned}&\left[w_{11}^{(1)}\sigma'(z_1^{(1)})\right]^2\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]^2 + 2w_{11}^{(1)}\sigma'(z_1^{(1)})w_{12}^{(1)}\sigma'(z_2^{(1)})w_{11}^{(0)}w_{21}^{(0)}\sigma'(z_1^{(0)})^2\\&+2w_{11}^{(1)}\sigma'(z_1^{(1)})w_{13}^{(1)}\sigma'(z_3^{(1)})w_{11}^{(0)}w_{31}^{(0)}\sigma'(z_1^{(0)})^2+\left[w_{12}^{(1)}\sigma'(z_2^{(1)})\right]^2\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]^2\\&+2w_{12}^{(1)}\sigma'(z_2^{(1)})w_{13}^{(1)}\sigma'(z_3^{(1)})w_{21}^{(0)}w_{31}^{(0)}\sigma'(z_1^{(0)})^2+\left[w_{13}^{(1)}\sigma'(z_3^{(1)})\right]^2\left[w_{31}^{(0)}\sigma'(z_1^{(0)})\right]^2,\end{aligned} \tag{B.1}$$

and developing $\left(\dfrac{\partial z_2^{(2)}}{\partial z_1^{(0)}}\right)^2$ results in multiplying $w_{12}^{(2)}\sigma''(z_2^{(2)})$ to

$$\begin{aligned}&\left[w_{21}^{(1)}\sigma'(z_1^{(1)})\right]^2\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]^2 + 2w_{21}^{(1)}\sigma'(z_1^{(1)})w_{22}^{(1)}\sigma'(z_2^{(1)})w_{11}^{(0)}w_{21}^{(0)}\sigma'(z_1^{(0)})^2\\&+2w_{21}^{(1)}\sigma'(z_1^{(1)})w_{23}^{(1)}\sigma'(z_3^{(1)})w_{11}^{(0)}w_{31}^{(0)}\sigma'(z_1^{(0)})^2+\left[w_{22}^{(1)}\sigma'(z_2^{(1)})\right]^2\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]^2\\&+2w_{22}^{(1)}\sigma'(z_2^{(1)})w_{23}^{(1)}\sigma'(z_3^{(1)})w_{21}^{(0)}w_{31}^{(0)}\sigma'(z_1^{(0)})^2+\left[w_{23}^{(1)}\sigma'(z_3^{(1)})\right]^2\left[w_{31}^{(0)}\sigma'(z_1^{(0)})\right]^2.\end{aligned} \tag{B.2}$$

Note that the layer 1 weights that are multiplied together point to the same node in layer 2. For example, $w_{11}^{(1)}$ and $w_{21}^{(1)}$, which point to $z_1^{(2)}$ and $z_2^{(2)}$ respectively, are never multiplied. Expectations of product terms to compute in layer 1 are shown in Table B.2.

Related to $w_{11}^{(2)}\sigma''\left(z_1^{(2)}\right)$	Related to $w_{12}^{(2)}\sigma''\left(z_1^{(2)}\right)$
$\mathbb{E}\left[w_{11}^{(2)}\sigma''\left(z_1^{(2)}\right)\left[w_{11}^{(1)}\sigma'(z_1^{(1)})\right]^2\right]$	$\mathbb{E}\left[w_{12}^{(2)}\sigma''\left(z_2^{(2)}\right)\left[w_{21}^{(1)}\sigma'(z_1^{(1)})\right]^2\right]$
$\mathbb{E}\left[w_{11}^{(2)}\sigma''\left(z_1^{(2)}\right)w_{11}^{(1)}\sigma'(z_1^{(1)})w_{12}^{(1)}\sigma'(z_2^{(1)})\right]$	$\mathbb{E}\left[w_{12}^{(2)}\sigma''\left(z_2^{(2)}\right)w_{21}^{(1)}\sigma'(z_1^{(1)})w_{22}^{(1)}\sigma'(z_2^{(1)})\right]$
$\mathbb{E}\left[w_{11}^{(2)}\sigma''\left(z_1^{(2)}\right)w_{11}^{(1)}\sigma'(z_1^{(1)})w_{13}^{(1)}\sigma'(z_3^{(1)})\right]$	$\mathbb{E}\left[w_{12}^{(2)}\sigma''\left(z_2^{(2)}\right)w_{21}^{(1)}\sigma'(z_1^{(1)})w_{23}^{(1)}\sigma'(z_3^{(1)})\right]$
$\mathbb{E}\left[w_{11}^{(2)}\sigma''\left(z_1^{(2)}\right)\left[w_{12}^{(1)}\sigma'(z_2^{(1)})\right]^2\right]$	$\mathbb{E}\left[w_{12}^{(2)}\sigma''\left(z_2^{(2)}\right)\left[w_{22}^{(1)}\sigma'(z_2^{(1)})\right]^2\right]$
$\mathbb{E}\left[w_{11}^{(2)}\sigma''\left(z_1^{(2)}\right)w_{12}^{(1)}\sigma'(z_2^{(1)})w_{13}^{(1)}\sigma'(z_3^{(1)})\right]$	$\mathbb{E}\left[w_{12}^{(2)}\sigma''\left(z_2^{(2)}\right)w_{22}^{(1)}\sigma'(z_2^{(1)})w_{23}^{(1)}\sigma'(z_3^{(1)})\right]$
$\mathbb{E}\left[w_{11}^{(2)}\sigma''\left(z_1^{(2)}\right)\left[w_{23}^{(1)}\sigma'(z_3^{(1)})\right]^2\right]$	$\mathbb{E}\left[w_{12}^{(2)}\sigma''\left(z_2^{(2)}\right)\left[w_{23}^{(1)}\sigma'(z_3^{(1)})\right]^2\right]$

Table B.2: Sequence Type 1 at Layer 1

Let $V_3V_4V_5V_6$ be the four first order derivative terms from layer 1. Then, $(A.5)$ can be rewritten for the expectation of the product of layers 2 and 1 terms as follows,

$$\begin{aligned}\mathbb{E}\left[V_1V_2V_3V_4V_5V_6\right] &= \mathbb{E}\left[V_1V_2\right]\mathbb{E}\left[V_3V_4V_5V_6\right] + \operatorname{cov}(V_1V_2, V_3V_4V_5V_6)\\ &= \mathbb{E}\left[V_1V_2\right]\mathbb{E}\left[V_3V_4V_5V_6\right] + \mathbb{E}\left[V_3V_4\right]\operatorname{cov}(V_1V_2, V_5V_6)\\ &\quad + \mathbb{E}\left[V_5V_6\right]\operatorname{cov}(V_1V_2, V_3V_4).\end{aligned}$$

$(A.2)$ can be used to calculate $\mathbb{E}\left[V_3V_4V_5V_6\right]$. However, $\operatorname{cov}(V_3V_4, V_5V_6)$ is not like in $(A.4)$ since now, the covariance is between terms from the same layer. There exists a theoretical covariance between nodes from the same layer, but it becomes negligible as the number of nodes increases, so it is not considered in TAGI. Since a hidden layer usually contains many neurons, we assume that covariance is null for simplicity. Therefore, if $V_3V_4 \neq V_5V_6$, $\text{cov}(V_3V_4, V_5V_6) = 0$, but if $V_3V_4 = V_5V_6$,

$$\begin{aligned}\text{cov}(V_3V_4, V_5V_6) &= \text{V}(V_3V_4)\\ &= \sigma_3^2\sigma_4^2 + \cancelto{0}{\text{cov}(V_3, V_4)^2} + 2\cancelto{0}{\text{cov}(V_3, V_4)}\mathbb{E}[V_3]\,\mathbb{E}[V_4]\\ &\quad + \sigma_3^2\mathbb{E}[V_4]^2 + \sigma_4^2\mathbb{E}[V_3]^2\\ &= \sigma_W^2\sigma_d^2 + \sigma_W^2\mathbb{E}[V_d]^2 + \sigma_d^2\mathbb{E}[V_W]^2,\end{aligned} \tag{B.3}$$

from (2.9). Note that expectation and variance for weights are already provided from the TAGI model and those for the derivatives are found in Appendix C since they depend on the activation function used.

It remains to calculate $\text{cov}(V_1V_2, V_3V_4)$ and $\text{cov}(V_1V_2, V_5V_6)$. For those, $(A.4)$ is modified such that

$$\text{cov}(V_1V_2, V_3V_4) = \mathbb{E}\left[\mathbf{W}^+\right]\,\text{cov}(\mathbf{dd}^+, \mathbf{Wd}), \tag{B.4}$$

since the term from next layer is a second order derivative instead of first. Formulas for sigmoid, tanh and ReLU functions are detailed in Appendix C.

Now, the last calculations for the sequence type are done in layer 0. Four terms from layer 0 are to be multiplied within the expectations shown in Table B.2. For the exact combinations, refer to $(B.1)$ and $(B.2)$.

Let $V_7V_8V_9V_{10}$ be the four first order derivative terms from layer 0. Then, the expectation of the product of layers 2, 1 and 0 terms is as follows,

$$
\begin{aligned}
\mathbb{E}[V_1V_2V_3V_4V_5V_6V_7V_8V_9V_{10}] &= \mathbb{E}[V_1V_2V_3V_4V_5V_6]\,\mathbb{E}[V_7V_8V_9V_{10}] \\
&\quad + \operatorname{cov}(V_1V_2V_3V_4V_5V_6, V_7V_8V_9V_{10}) \\
&= \mathbb{E}[V_1V_2V_3V_4V_5V_6]\,\mathbb{E}[V_7V_8V_9V_{10}] \\
&\quad + \mathbb{E}[V_1V_2]\operatorname{cov}(V_3V_4V_5V_6, V_7V_8V_9V_{10}) \qquad (B.5) \\
&\quad + \mathbb{E}[V_3V_4V_5V_6]\,\cancelto{0}{\operatorname{cov}(V_1V_2, V_7V_8V_9V_{10})} \\
&= \mathbb{E}[V_1V_2V_3V_4V_5V_6]\,\mathbb{E}[V_7V_8V_9V_{10}] \\
&\quad + \mathbb{E}[V_1V_2]\operatorname{cov}(V_3V_4V_5V_6, V_7V_8V_9V_{10}).
\end{aligned}
$$

Note that, as in $(A.5)$, V_1V_2 (from layer 2) and $V_7V_8V_9V_{10}$ (from layer 0) are independent because of the inherent conditional independence of hidden units between layers $\boldsymbol{Z}^{(j-1)} \perp\!\!\!\perp \boldsymbol{Z}^{(j+1)}|\boldsymbol{z}^{(j)}$. $(A.2)$ can be used to calculate $\mathbb{E}[V_7V_8V_9V_{10}]$. However, $\operatorname{cov}(V_7V_8, V_9V_{10})$ is not like in $(A.4)$ since now, the covariance is between terms from the same layer and $V_8 = V_{10}$. Therefore, if $V_7 = V_9$ (i.e. $V_7V_8 = V_9V_{10}$), $(B.3)$ can be used, but if $V_7 \neq V_9$,

$$
\begin{aligned}
\operatorname{cov}(V_7V_8, V_9V_{10}) &= \operatorname{cov}(V_7V_8, V_9V_8) \\
&= \cancelto{0}{\operatorname{cov}(V_7, V_9)}\operatorname{cov}(V_8, V_8) + \cancelto{0}{\operatorname{cov}(V_7, V_8)}\cancelto{0}{\operatorname{cov}(V_8, V_9)} \\
&\quad + \mathbb{E}[V_7]\operatorname{cov}(V_8, V_9V_8) + \mathbb{E}[V_8]\,\cancelto{0}{\operatorname{cov}(V_7, V_9V_8)} \\
&= \mathbb{E}[V_7]\left(\mathbb{E}[V_9]\operatorname{cov}(V_8, V_8) + \mathbb{E}[V_8]\,\cancelto{0}{\operatorname{cov}(V_8, V_9)}\right) \\
&= \mathbb{E}[V_7]\,\mathbb{E}[V_9]\,\mathrm{V}(V_8) \\
&= \mathbb{E}[V_{W_7}]\,\mathbb{E}[V_{W_9}]\,\mathrm{V}(V_d),
\end{aligned}
$$

using $(A.3)$. It only remains to calculate $\mathrm{cov}(V_3V_4V_5V_6, V_7V_8V_9V_{10})$. Using $(A.3)$,

$$\begin{aligned}
&\mathrm{cov}(V_3V_4V_5V_6, V_7V_8V_9V_{10})\\
&= \mathrm{cov}(V_3V_4, V_7V_8)\mathrm{cov}(V_5V_6, V_9V_{10}) + \mathrm{cov}(V_3V_4, V_9V_{10})\mathrm{cov}(V_5V_6, V_7V_8)\\
&\quad + \mathbb{E}\left[V_3V_4\right]\mathrm{cov}(V_5V_6, V_7V_8V_9V_{10}) + \mathbb{E}\left[V_5V_6\right]\mathrm{cov}(V_3V_4, V_7V_8V_9V_{10})\\
&= \mathrm{cov}(V_3V_4, V_7V_8)\mathrm{cov}(V_5V_6, V_9V_{10}) + \mathrm{cov}(V_3V_4, V_9V_{10})\mathrm{cov}(V_5V_6, V_7V_8)\\
&\quad + \mathbb{E}\left[V_3V_4\right]\left(\mathbb{E}\left[V_7V_8\right]\mathrm{cov}(V_5V_6, V_9V_{10}) + \mathbb{E}\left[V_9V_{10}\right]\mathrm{cov}(V_5V_6, V_7V_8)\right)\\
&\quad + \mathbb{E}\left[V_5V_6\right]\left(\mathbb{E}\left[V_7V_8\right]\mathrm{cov}(V_3V_4, V_9V_{10}) + \mathbb{E}\left[V_9V_{10}\right]\mathrm{cov}(V_3V_4, V_7V_8)\right),
\end{aligned} \tag{B.6}$$

where $\mathrm{cov}(V_3V_4, V_7V_8)$, $\mathrm{cov}(V_5V_6, V_9V_{10})$, $\mathrm{cov}(V_3V_4, V_9V_{10})$ and $\mathrm{cov}(V_5V_6, V_7V_8)$ can be calculated using $(A.2)$. From the possible combinations of products of terms from layers 1 and 0, there are only two possible types:

1. V_7V_8 is only connected to V_3V_4 and V_9V_{10} to V_5V_6 (i.e. same layer 0 node, but are not connected to the same layer 1 node)
2. $V_3V_4 = V_5V_6$ and $V_7V_8 = V_9V_{10}$, where V_7V_8 is connected to V_3V_4

In both cases, the node from layer 0 is always the same such that $V_8 = V_{10}$.

For Case 1, if the two terms from layer 1 and the two from layer 0 are not connected in Figure 3.1 (i.e. if the weight from layer 0 does not point toward the layer 1 node), their covariance is equal to 0. Taking for example $\mathbb{E}\left[w_{11}^{(1)}\sigma'(z_1^{(1)})w_{12}^{(1)}\sigma'(z_2^{(1)})w_{11}^{(0)}w_{21}^{(0)}\sigma'(z_1^{(0)})^2\right]$ and evaluating the required covariance $\mathrm{cov}(w_{11}^{(1)}\sigma'(z_1^{(1)})w_{12}^{(1)}\sigma'(z_2^{(1)}), w_{11}^{(0)}w_{21}^{(0)}\sigma'(z_1^{(0)})^2)$ using $(B.6)$,

$$
\begin{aligned}
&\mathrm{cov}(w_{11}^{(1)}\sigma'(z_1^{(1)})w_{12}^{(1)}\sigma'(z_2^{(1)}), w_{11}^{(0)}w_{21}^{(0)}\sigma'(z_1^{(0)})^2)\\
&= \mathrm{cov}(w_{11}^{(1)}\sigma'(z_1^{(1)}), w_{11}^{(0)}\sigma'(z_1^{(0)}))\mathrm{cov}(w_{12}^{(1)}\sigma'(z_2^{(1)}), w_{21}^{(0)}\sigma'(z_1^{(0)}))\\
&\quad + \cancelto{0}{\mathrm{cov}((w_{11}^{(1)}\sigma'(z_1^{(1)}), w_{21}^{(0)}\sigma'(z_1^{(0)}))}\cancelto{0}{\mathrm{cov}(w_{12}^{(1)}\sigma'(z_2^{(1)}), w_{11}^{(0)}\sigma'(z_1^{(0)}))}\\
&\quad + \mathbb{E}\left[w_{11}^{(1)}\sigma'(z_1^{(1)})\right]\mathbb{E}\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]\mathrm{cov}(w_{12}^{(1)}\sigma'(z_2^{(1)}), w_{21}^{(0)}\sigma'(z_1^{(0)}))\\
&\quad + \mathbb{E}\left[w_{11}^{(1)}\sigma'(z_1^{(1)})\right]\mathbb{E}\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]\cancelto{0}{\mathrm{cov}(w_{12}^{(1)}\sigma'(z_2^{(1)}), w_{11}^{(0)}\sigma'(z_1^{(0)}))}\\
&\quad + \mathbb{E}\left[w_{12}^{(1)}\sigma'(z_2^{(1)})\right]\mathbb{E}\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]\cancelto{0}{\mathrm{cov}(w_{11}^{(1)}\sigma'(z_1^{(1)}), w_{21}^{(0)}\sigma'(z_1^{(0)}))}\\
&\quad + \mathbb{E}\left[w_{12}^{(1)}\sigma'(z_2^{(1)})\right]\mathbb{E}\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]\mathrm{cov}(w_{11}^{(1)}\sigma'(z_1^{(1)}), w_{11}^{(0)}\sigma'(z_1^{(0)}))\\
&= \mathrm{cov}(w_{11}^{(1)}\sigma'(z_1^{(1)}), w_{11}^{(0)}\sigma'(z_1^{(0)}))\mathrm{cov}(w_{12}^{(1)}\sigma'(z_2^{(1)}), w_{21}^{(0)}\sigma'(z_1^{(0)}))\\
&\quad + \mathbb{E}\left[w_{11}^{(1)}\sigma'(z_1^{(1)})\right]\mathbb{E}\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]\mathrm{cov}(w_{12}^{(1)}\sigma'(z_2^{(1)}), w_{21}^{(0)}\sigma'(z_1^{(0)}))\\
&\quad + \mathbb{E}\left[w_{12}^{(1)}\sigma'(z_2^{(1)})\right]\mathbb{E}\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]\mathrm{cov}(w_{11}^{(1)}\sigma'(z_1^{(1)}), w_{11}^{(0)}\sigma'(z_1^{(0)})).
\end{aligned}
$$

In Case 1, $(B.6)$ simplifies to

$$
\begin{aligned}
\mathrm{cov}(V_3V_4V_5V_6, V_7V_8V_9V_{10}) &= \mathrm{cov}(V_3V_4, V_7V_8)\mathrm{cov}(V_5V_6, V_9V_{10})\\
&\quad + \mathbb{E}\left[V_3V_4\right]\mathbb{E}\left[V_7V_8\right]\mathrm{cov}(V_5V_6, V_9V_{10})\\
&\quad + \mathbb{E}\left[V_5V_6\right]\mathbb{E}\left[V_9V_{10}\right]\mathrm{cov}(V_3V_4, V_7V_8),
\end{aligned}
$$

since $\mathrm{cov}(V_3V_4, V_9V_{10}) = \mathrm{cov}(V_5V_6, V_7V_8) = 0$.

In Case 2, $V_3V_4 = V_5V_6$ and $V_7V_8 = V_9V_{10}$. Also, V_7V_8 is connected to V_3V_4 so $\mathrm{cov}(V_3V_4, V_7V_8) \neq 0$. $(B.6)$ simplifies to

$$\begin{aligned}
\mathrm{cov}(V_3V_4V_5V_6, V_7V_8V_9V_{10}) &= \mathrm{cov}((V_3V_4)^2, (V_7V_8)^2) \\
&= 2\mathrm{cov}(V_3V_4, V_7V_8)^2 \\
&\quad + 4\mathbb{E}\left[V_3V_4\right]\mathbb{E}\left[V_7V_8\right]\mathrm{cov}(V_3V_4, V_7V_8).
\end{aligned}$$

B.2 Sequence Type 2

The second sequence type is the first sum in $\dfrac{\partial g}{\partial \boldsymbol{z}^{(2)}}\dfrac{\partial^2 \boldsymbol{z}^{(2)}}{(\partial z_1^{(0)})^2}$, which is, from (3.14), equal to

$$\begin{aligned}
&w_{11}^{(2)}\sigma'(z_1^{(2)})\left\{w_{11}^{(1)}\sigma''(z_1^{(1)})\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]^2 + w_{12}^{(1)}\sigma''(z_2^{(1)})\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]^2\right. \\
&\left.+w_{13}^{(1)}\sigma''(z_3^{(1)})\left[w_{31}^{(0)}\sigma'(z_1^{(0)})\right]^2\right\} + w_{12}^{(2)}\sigma'(z_2^{(2)})\left\{w_{21}^{(1)}\sigma''(z_1^{(1)})\left[w_{11}^{(0)}\sigma'(z_1^{(0)})\right]^2\right. \\
&\left.+w_{22}^{(1)}\sigma''(z_2^{(1)})\left[w_{21}^{(0)}\sigma'(z_1^{(0)})\right]^2 + w_{23}^{(1)}\sigma''(z_3^{(1)})\left[w_{31}^{(0)}\sigma'(z_1^{(0)})\right]^2\right\}.
\end{aligned}$$

In layer 2, the same expectations as in the first derivative process (refer to Section 3.1), $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)\right]$ and $\mathbb{E}\left[w_{12}^{(2)}\sigma'\left(z_2^{(2)}\right)\right]$, are to be calculated using $(A.1)$.

In layer 1, there are six expectations to compute from which the only difference with the first derivative process is that the products of layer 1 are now of second order derivative (instead of first). Expectations to calculate are now $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)w_{11}^{(1)}\sigma''\left(z_1^{(1)}\right)\right]$, $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)w_{12}^{(1)}\sigma''\left(z_2^{(1)}\right)\right]$, $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)w_{13}^{(1)}\sigma''\left(z_3^{(1)}\right)\right]$, $\mathbb{E}\left[w_{12}^{(2)}\sigma'\left(z_2^{(2)}\right)w_{21}^{(1)}\sigma''\left(z_1^{(1)}\right)\right]$, $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_2^{(2)}\right)w_{22}^{(1)}\sigma''\left(z_2^{(1)}\right)\right]$ and $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_2^{(2)}\right)w_{23}^{(1)}\sigma''\left(z_3^{(1)}\right)\right]$. They can, as in the first derivative process, be calculated using $(A.2)$, but to calculate $\mathrm{cov}(V_1V_2, V_3V_4)$, $(A.4)$ is modified as follows,

$$\text{cov}(V_1V_2, V_3V_4) = \mathbb{E}\left[\mathbf{W}^+\right] \text{ cov}(\mathbf{d}^+, \mathbf{Wdd}), \qquad (B.7)$$

since the term from layer 1 is a second order derivative instead of first. Formulas for sigmoid, tanh and ReLU functions are detailed in Appendix C.

Now at layer 0, let $V_5V_6V_7V_8$ be the four order derivative terms from that layer. Similar to $(B.5)$,

$$\begin{aligned}\mathbb{E}\left[V_1V_2V_3V_4V_5V_6V_7V_8\right] = {} & \mathbb{E}\left[V_1V_2V_3V_4\right]\mathbb{E}\left[V_5V_6V_7V_8\right] \\ & + \text{cov}(V_1V_2V_3V_4, V_5V_6V_7V_8).\end{aligned}$$

From the possible combinations of products of terms from layers 0 and 1, there is only one possible type:

1. $V_5V_6 = V_7V_8$, where V_5V_6 is connected to V_3V_4

From $(A.2)$ and $(B.3)$, $\mathbb{E}\left[V_5V_6V_7V_8\right]$ is as follows,

$$\begin{aligned}\mathbb{E}\left[V_5V_6V_7V_8\right] &= \mathbb{E}\left[(V_5V_6)^2\right] \\ &= \mathbb{E}\left[V_5\right]^2\mathbb{E}\left[V_6\right]^2 + \text{Var}(V_5)\text{Var}(V_6) \\ &\quad + \text{Var}(V_5)\mathbb{E}\left[V_6\right]^2 + \text{Var}(V_6)\mathbb{E}\left[V_5\right]^2.\end{aligned}$$

Also, similar to $(B.6)$ and using $(A.3)$,

$$
\begin{aligned}
&\mathrm{cov}(V_1V_2V_3V_4, V_5V_6V_7V_8)\\
&= \cancelto{0}{\mathrm{cov}(V_1V_2, V_5V_6)}\mathrm{cov}(V_3V_4, V_7V_8) + \cancelto{0}{\mathrm{cov}(V_1V_2, V_7V_8)}\mathrm{cov}(V_3V_4, V_5V_6)\\
&\quad + \mathbb{E}\left[V_1V_2\right]\mathrm{cov}(V_3V_4, V_5V_6V_7V_8) + \mathbb{E}\left[V_3V_4\right]\cancelto{0}{\mathrm{cov}(V_1V_2, V_5V_6V_7V_8)}\\
&= \mathbb{E}\left[V_1V_2\right]\left(\mathbb{E}\left[V_5V_6\right]\mathrm{cov}(V_3V_4, V_7V_8) + \mathbb{E}\left[V_7V_8\right]\mathrm{cov}(V_3V_4, V_5V_6)\right)\\
&= 2\mathbb{E}\left[V_1V_2\right]\mathbb{E}\left[V_5V_6\right]\mathrm{cov}(V_3V_4, V_5V_6).
\end{aligned}
$$

Note that, as in $(A.5)$ and $(B.5)$, V_1V_2 (from layer 2) is independent of V_5V_6 and V_7V_8 (from layer 0) because of the inherent conditional independence of hidden units between layers $\boldsymbol{Z}^{(j-1)} \perp\!\!\!\perp \boldsymbol{Z}^{(j+1)}|\boldsymbol{z}^{(j)}$. $(A.2)$ can be used to calculate $\mathbb{E}\left[V_7V_8V_9V_{10}\right]$. $\mathrm{cov}(V_3V_4, V_5V_6)$ can be directly calculated using $(B.4)$.

B.3 Sequence Type 3

The last sequence type is the second sum in $\dfrac{\partial g}{\partial \boldsymbol{z}^{(2)}}\dfrac{\partial^2 \boldsymbol{z}^{(2)}}{(\partial z_1^{(0)})^2}$ which is, from (3.14), equal to

$$
\begin{aligned}
&w_{11}^{(2)}\sigma'(z_1^{(2)})\\
&\left\{w_{11}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma''(z_1^{(0)}) + w_{12}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma''(z_1^{(0)}) + w_{13}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma''(z_1^{(0)})\right\}\\
&+ w_{12}^{(2)}\sigma'(z_2^{(2)})\\
&\left\{w_{21}^{(1)}\sigma'(z_1^{(1)})w_{11}^{(0)}\sigma''(z_1^{(0)}) + w_{22}^{(1)}\sigma'(z_2^{(1)})w_{21}^{(0)}\sigma''(z_1^{(0)}) + w_{23}^{(1)}\sigma'(z_3^{(1)})w_{31}^{(0)}\sigma''(z_1^{(0)})\right\}.
\end{aligned}
$$

For both layers 2 and 1, the same process as for the first derivative (refer to Section 3.1) is to be applied since the terms to multiply are the same for those layers. The difference in process only comes at layer 0. Therefore, starting with layer 2, the same expectations as in the first derivative process, $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)\right]$ and $\mathbb{E}\left[w_{12}^{(2)}\sigma'\left(z_2^{(2)}\right)\right]$, are to be calculated using $(A.1)$. In layer 1, the six same expectations, which are $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)w_{11}^{(1)}\sigma'\left(z_1^{(1)}\right)\right]$, $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)w_{12}^{(1)}\sigma'\left(z_2^{(1)}\right)\right]$, $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)w_{13}^{(1)}\sigma'\left(z_3^{(1)}\right)\right]$, $\mathbb{E}\left[w_{12}^{(2)}\sigma'\left(z_2^{(2)}\right)w_{21}^{(1)}\sigma'\left(z_1^{(1)}\right)\right]$, $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_2^{(2)}\right)w_{22}^{(1)}\sigma'\left(z_2^{(1)}\right)\right]$ and $\mathbb{E}\left[w_{11}^{(2)}\sigma'\left(z_2^{(2)}\right)w_{23}^{(1)}\sigma'\left(z_3^{(1)}\right)\right]$, are to be calculated using $(A.2)$ and $\mathrm{cov}(V_1V_2, V_3V_4)$ using $(A.4)$.

Now, at layer 0, $\mathbb{E}\left[V_1V_2V_3V_4V_5V_6\right]$ can be computed using $(A.5)$, but $\mathrm{cov}(V_3V_4, V_5V_6)$ is calculated using $(B.7)$.

C Activation's Derivative $a(\mathbf{z})$[13]

$$\begin{aligned}
\mu_a &= a(\mu_z) \\
\Sigma_a &= \mathbf{J}_a \Sigma_z \mathbf{J}_a^{\intercal} \\
\mathbf{J}_a(\mu_z) &= \mu_a \left[1 - \mu_a\right]
\end{aligned}$$

C.1 Sigmoid($\mathbf{z}$)

$$a = \frac{1}{1 + \exp(-\mathbf{z})}$$

C.1.1 First Derivative

$$\begin{aligned}
d &= a(\mathbf{z}) \left[1 - a(\mathbf{z})\right] \\
\mu_d &= \mu_a \left(1 - \mu_a\right) - \Sigma_a \\
\Sigma_d &= \Sigma_a \left[2\Sigma_a + 4\mu_a^2 - 4\mu_a + 1\right]
\end{aligned}$$

$$\begin{aligned}
\text{cov}(\mathbf{d}^+, \mathbf{W}\mathbf{d}) &= \text{cov}(\mathbf{d}^+, \mathbf{W})\mu_d + \text{cov}(\mathbf{d}^+, \mathbf{d})\mu_W \\
\text{cov}(\mathbf{d}^+, \mathbf{W}) &= \left[1 - 2\mu_{a^+}\right] \text{cov}(\mathbf{a}^+, \mathbf{W}) \\
\text{cov}(\mathbf{a}^+, \mathbf{W}) &= \mathbf{J}_{a^+} \text{cov}(\mathbf{W}, \mathbf{W})\mu_a \\
\text{cov}(\mathbf{d}^+, \mathbf{d}) &= \text{cov}(\mathbf{a}^+, \mathbf{a}) - 2\text{cov}(\mathbf{a}^+, \mathbf{a})\mu_{\mathbf{a}} - 2\text{cov}(\mathbf{a}^+, \mathbf{a})\mu_{\mathbf{a}^+} \\
&+ 2\text{cov}(\mathbf{a}^+, \mathbf{a})^2 + 4\text{cov}(\mathbf{a}^+, \mathbf{a})\mu_{a^+}\mu_a \\
\text{cov}(\mathbf{a}^+, \mathbf{a}) &= \mathbf{J}_{a^+} \text{cov}(\mathbf{a}, \mathbf{a})\mu_W
\end{aligned}$$

[13]In collaboration with J.-A. Goulet and L. H. Nguyen.

$$\begin{aligned}
\operatorname{cov}(\mathbf{d}^+, \mathbf{z}) &= [1 - 2\mu_{a^+}]\operatorname{cov}(\mathbf{a}^+, \mathbf{z}) \\
\operatorname{cov}(\mathbf{a}^+, \mathbf{z}) &= \mathbf{J}_{a^+}\mathbf{J}_a\operatorname{cov}(\mathbf{z}, \mathbf{z})\mu_W \\
\operatorname{cov}(\mathbf{d}, \mathbf{z}) &= [1 - 2\mu_a]\operatorname{cov}(\mathbf{a}, \mathbf{z}) \\
\operatorname{cov}(\mathbf{a}, \mathbf{z}) &= \mathbf{J}_a\operatorname{cov}(\mathbf{z}, \mathbf{z})
\end{aligned}$$

The covariance between d^+ and d is detailed following

$$\begin{aligned}
\operatorname{cov}(\mathbf{d}^+, \mathbf{d}) &= \operatorname{cov}(\mathbf{a}^+(1-\mathbf{a}^+), \mathbf{a}(1-\mathbf{a})) \\
&= \operatorname{cov}(\mathbf{a}^+ - (\mathbf{a}^+)^2, \mathbf{a} - \mathbf{a}^2) \\
&= \operatorname{cov}(\mathbf{a}^+, \mathbf{a}) - \operatorname{cov}(\mathbf{a}^+, \mathbf{a}^2) - \operatorname{cov}((\mathbf{a}^+)^2, \mathbf{a}) + \operatorname{cov}((\mathbf{a}^+)^2, \mathbf{a}^2) \\
&= \operatorname{cov}(\mathbf{a}^+, \mathbf{a}) - 2\operatorname{cov}(\mathbf{a}^+, \mathbf{a})\mu_{\mathbf{a}} - 2\operatorname{cov}(\mathbf{a}^+, \mathbf{a})\mu_{\mathbf{a}^+} \\
&+ 2\operatorname{cov}(\mathbf{a}^+, \mathbf{a})^2 + 4\operatorname{cov}(\mathbf{a}^+, \mathbf{a})\mu_{a^+}\mu_a,
\end{aligned}$$

where

$$\begin{aligned}
\operatorname{cov}((\mathbf{a}^+)^2, \mathbf{a}^2) &= \operatorname{cov}(\mathbf{a}^+\mathbf{a}^+, \mathbf{a}\mathbf{a}) \text{ see Equation 5 in Goulet et al. (2020)} \\
&= 2\operatorname{cov}(\mathbf{a}^+, \mathbf{a})^2 + 4\operatorname{cov}(\mathbf{a}^+, \mathbf{a})\mu_{a^+}\mu_a \\
\operatorname{cov}(\mathbf{a}^+, \mathbf{a}^2) &= 2\operatorname{cov}(\mathbf{a}^+, \mathbf{a})\mu_{\mathbf{a}} \text{ see Equation 4 in Goulet et al. (2020)} \\
\operatorname{cov}((\mathbf{a}^+)^2, \mathbf{a}) &= 2\operatorname{cov}(\mathbf{a}^+, \mathbf{a})\mu_{\mathbf{a}^+}
\end{aligned}$$

C.1.2 Second Derivative

$$\begin{aligned}
dd &= a(\mathbf{z})\,[1-a(\mathbf{z})]\,[1-2a(\mathbf{z})] \\
\mathrm{cov}(d, 1-2a(\mathbf{z})) &= 4\Sigma_a\mu_a - 2\Sigma_a \\
\mu_{dd} &= \mu_d(1-2\mu_a) + \mathrm{cov}(d, 1-2a(\mathbf{z})) \\
\Sigma_{dd} &= 4\Sigma_d\Sigma_a + \mathrm{cov}(d, 1-2a(\mathbf{z}))^2 \\
&+ 2\mathrm{cov}(d, 1-2a(\mathbf{z}))\mu_d(1-2\mu_a) + \Sigma_d(1-2\mu_a)^2 \\
&+ 4\Sigma_a\mu_d^2
\end{aligned}$$

$$\begin{aligned}
\mathrm{cov}(\mathbf{dd}^+, (\mathbf{Wd})^2) &= 2\mathrm{cov}(\mathbf{dd}^+, \mathbf{Wd})\mu_{Wd} \\
\mathrm{cov}(\mathbf{dd}^+, \mathbf{Wd}) &= \mathrm{cov}(\mathbf{dd}^+, \mathbf{W})\mu_d + \mathrm{cov}(\mathbf{dd}^+, \mathbf{d})\mu_W \\
\mathrm{cov}(\mathbf{dd}^+, \mathbf{W}) &= \mathrm{cov}(\mathbf{d}^+, \mathbf{W}) - 2\mathrm{cov}(\mathbf{d}^+, \mathbf{W})\mu_{a^+} - 2\mathrm{cov}(\mathbf{a}^+, \mathbf{W})\mu_{d^+} \\
\mathrm{cov}(\mathbf{dd}^+, \mathbf{d}) &= \mathrm{cov}(\mathbf{d}^+(1-2\mathbf{a}^+), \mathbf{d}) \\
&= \mathrm{cov}(\mathbf{d}^+, \mathbf{d}) - 2\mathrm{cov}(\mathbf{d}^+, \mathbf{d})\mu_{a^+} - 2\mathrm{cov}(\mathbf{a}^+, \mathbf{d})\mu_{d^+} \\
\mathrm{cov}(\mathbf{a}^+, \mathbf{d}) &= \mathrm{cov}(\mathbf{a}^+, \mathbf{a})\,[1-2\mu_a]
\end{aligned}$$

$$\begin{aligned}
\mathrm{cov}(\mathbf{d}^+, \mathbf{Wdd}) &= \mathrm{cov}(\mathbf{d}^+, \mathbf{W})\mu_{dd} + \mathrm{cov}(\mathbf{d}^+, \mathbf{dd})\mu_W \\
\mathrm{cov}(\mathbf{d}^+, \mathbf{dd}) &= \mathrm{cov}(\mathbf{d}^+, \mathbf{d}) - 2\mathrm{cov}(\mathbf{d}^+, \mathbf{d})\mu_a - 2\mathrm{cov}(\mathbf{d}^+, \mathbf{a})\mu_d \\
\mathrm{cov}(\mathbf{d}^+, \mathbf{a}) &= \mathrm{cov}(\mathbf{a}^+, \mathbf{a})\,[1-2\mu_{a^+}]
\end{aligned}$$

C.2 Tanh(z)

$$a = \tanh(\mathbf{z})$$

C.2.1 First Derivative

$$\begin{aligned} d &= 1 - a(\mathbf{z})^2 \\ \mu_d &= 1 - \mu_a^2 - \Sigma_a \\ \Sigma_d &= 2\Sigma_a\left(\Sigma_a + 2\mu_a^2\right) \end{aligned}$$

$$\begin{aligned} \mathrm{cov}(\mathbf{d}^+, \mathbf{W}\mathbf{d}) &= \mathrm{cov}(\mathbf{d}^+, \mathbf{W})\mu_d + \mathrm{cov}(\mathbf{d}^+, \mathbf{d})\mu_W \\ \mathrm{cov}(\mathbf{d}^+, \mathbf{W}) &= -2\mu_{a^+}\mathrm{cov}(\mathbf{a}^+, \mathbf{W}) \\ \mathrm{cov}(\mathbf{a}^+, \mathbf{W}) &= \mathbf{J}_{a^+}\mathrm{cov}(\mathbf{W}, \mathbf{W})\mu_a \\ \mathrm{cov}(\mathbf{d}^+, \mathbf{d}) &= 2\mathrm{cov}(\mathbf{a}^+, \mathbf{a})^2 + 4\mathrm{cov}(\mathbf{a}^+, \mathbf{a})\mu_{a^+}\mu_a \\ \mathrm{cov}(\mathbf{a}^+, \mathbf{a}) &= \mathbf{J}_{a^+}\mathrm{cov}(\mathbf{a}, \mathbf{a})\mu_W \end{aligned}$$

$$\begin{aligned} \mathrm{cov}(\mathbf{d}^+, \mathbf{z}) &= -2\mu_{a^+}\mathrm{cov}(\mathbf{a}^+, \mathbf{z}) \\ \mathrm{cov}(\mathbf{a}^+, \mathbf{z}) &= \mathbf{J}_{a^+}\mathbf{J}_a\mathrm{cov}(\mathbf{z}, \mathbf{z})\mu_W \\ \mathrm{cov}(\mathbf{d}, \mathbf{z}) &= -2\mu_a\mathrm{cov}(\mathbf{a}, \mathbf{z}) \\ \mathrm{cov}(\mathbf{a}, \mathbf{z}) &= \mathbf{J}_a\mathrm{cov}(\mathbf{z}, \mathbf{z}) \end{aligned}$$

The covariance between d^+ and d is detailed following

$$\begin{aligned} \mathrm{cov}(\mathbf{d}^+, \mathbf{d}) &= \mathrm{cov}(1 - (\mathbf{a}^+)^2, 1 - (\mathbf{a})^2) \\ &= \mathrm{cov}((\mathbf{a}^+)^2, (\mathbf{a})^2) \\ &= \mathrm{cov}(\mathbf{a}^+\mathbf{a}^+, \mathbf{a}\mathbf{a}) \\ &= 2\mathrm{cov}(\mathbf{a}^+, \mathbf{a})^2 + 4\mathrm{cov}(\mathbf{a}^+, \mathbf{a})\mu_{a^+}\mu_a \end{aligned}$$

C.2.2 Second Derivative

$$\begin{aligned}
dd &= -2a(\mathbf{z})\,[1 - a(\mathbf{z})^2] \\
\mathrm{cov}(d, -2a(\mathbf{z})) &= 4\Sigma_a \mu_a \\
\mu_{dd} &= -2\mu_d \mu_a + 4\Sigma_a \mu_a \\
\Sigma_{dd} &= 4\Sigma_d \Sigma_a + \mathrm{cov}(d, -2a(\mathbf{z}))^2 - 4\mathrm{cov}(d, -2a(\mathbf{z}))\mu_d \mu_a \\
&+ 4\Sigma_d \mu_a^2 + 4\Sigma_a \mu_d^2
\end{aligned}$$

$$\begin{aligned}
\mathrm{cov}(\mathbf{dd}^+, (\mathbf{Wd})^2) &= 2\mathrm{cov}(\mathbf{dd}^+, \mathbf{Wd})\mu_{Wd} \\
\mathrm{cov}(\mathbf{dd}^+, \mathbf{Wd}) &= \mathrm{cov}(\mathbf{dd}^+, \mathbf{W})\mu_d + \mathrm{cov}(\mathbf{dd}^+, \mathbf{d})\mu_W \\
\mathrm{cov}(\mathbf{dd}^+, \mathbf{W}) &= -2\mathrm{cov}(\mathbf{a}^+, \mathbf{W})\mu_{d^+} - 2\mathrm{cov}(\mathbf{d}^+, \mathbf{W})\mu_{a^+} \\
\mathrm{cov}(\mathbf{dd}^+, \mathbf{d}) &= -2\mathrm{cov}(\mathbf{a}^+, \mathbf{d})\mu_{d^+} - 2\mathrm{cov}(\mathbf{d}^+, \mathbf{d})\mu_{a^+} \\
\mathrm{cov}(\mathbf{a}^+, \mathbf{d}) &= -2\mathrm{cov}(\mathbf{a}^+, \mathbf{a})\mu_a
\end{aligned}$$

$$\begin{aligned}
\mathrm{cov}(\mathbf{d}^+, \mathbf{Wdd}) &= \mathrm{cov}(\mathbf{d}^+, \mathbf{W})\mu_{dd} + \mathrm{cov}(\mathbf{d}^+, \mathbf{dd})\mu_W \\
\mathrm{cov}(\mathbf{d}^+, \mathbf{dd}) &= -2\mathrm{cov}(\mathbf{d}^+, \mathbf{a})\mu_d - 2\mathrm{cov}(\mathbf{d}^+, \mathbf{d})\mu_a \\
\mathrm{cov}(\mathbf{d}^+, \mathbf{a}) &= -2\mathrm{cov}(\mathbf{a}^+, \mathbf{a})\mu_{a^+}
\end{aligned}$$

C.3 ReLU(z)

$$a = \begin{cases} \mathbf{z} & if \quad \mathbf{z} > 0 \\ 0 & if \quad \mathbf{z} \leq 0 \end{cases}$$

C.3.1 First Derivative

$$d = \begin{cases} 1 \quad if \quad \mathbf{z} > 0 \\ 0 \quad if \quad \mathbf{z} \leq 0 \end{cases}$$

D Package Manual

Package ‘tagi’

July 23, 2021

Title Tractable Approximate Gaussian Inference in Neural Networks

Version 0.0.0.9000

Author Magali-Chen Goulet [aut, cre],
Mélina Mailhot [aut],
James-Alexandre Goulet [aut],
Luong-Ha Nguyen [aut]

Maintainer Magali-Chen Goulet <mag_goul@live.concordia.ca>

Description In this package, we implement the Tractable Approximate Gaussian Inference (TAGI)
is a method developped by Goulet et al. (2020),
used in Bayesian neural networks.

License GPL (>= 3)

URL https://github.com/mgoulet847/tagi

BugReports https://github.com/mgoulet847/tagi

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

Suggests knitr,
rmarkdown,
mvtnorm,
randtoolbox,
xgboost

VignetteBuilder knitr

Imports matlab,
stats

Depends R (>= 2.10)

R topics documented:

`activationFunIndex` *Assign ID to activation functions*

Description

This function assigns an ID number depending on the type of activation function.

Usage

```
activationFunIndex(funName)
```

Arguments

`funName`	Type of activation function: "tanh", "sigm", "cdf", "relu" or "softplus"

Value

An ID number which corresponds to:

- 1 if `funName` is "tanh"
- 2 if `funName` is "sigm"
- 3 if `funName` is "cdf"
- 4 if `funName` is "relu"
- 5 if `funName` is "softplus"

`backwardHiddenStateUpdate`

Backward hidden states update

Description

This function updates hidden units from responses to input data. It updates $\mu_{Z|y}$ and $\Sigma_{Z|y}$ from the $Z|y$ distribution for a given layer.

Usage

```
backwardHiddenStateUpdate(mz, Sz, mzF, SzF, SzB, Czz, mzB, idx)
```

Arguments

`mz`	Mean vector of units for the current layer μ_Z
`Sz`	Covariance matrix of units for the current layer Σ_Z
`mzF`	Mean vector of units for the next layer μ_{Z^+}
`SzF`	Covariance matrix of units for the next layer Σ_{Z^+}
`SzB`	Covariance matrix of units for the next layer given y $\Sigma_{Z^+|y}$
`Czz`	Covariance matrix between units of previous and current layers Σ_{ZZ^+}
`mzB`	Mean vector of units for the next layer given y $\mu_{Z^+|y}$
`idx`	Indices for the hidden state update step of the current layer

Details

$f(\boldsymbol{z}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu_{Z|y}}, \boldsymbol{\Sigma_{Z|y}})$ where

$$\boldsymbol{\mu_{Z|y}} = \boldsymbol{\mu_Z} + \boldsymbol{J_Z}(\boldsymbol{\mu_{Z^+|y}} - \boldsymbol{\mu_{Z^+}})$$

$$\boldsymbol{\Sigma_{Z|y}} = \boldsymbol{\Sigma_Z} + \boldsymbol{J_Z}(\boldsymbol{\Sigma_{Z^+|y}} - \boldsymbol{\Sigma_{Z^+}})\boldsymbol{J_Z^T}$$

$$\boldsymbol{J_Z} = \boldsymbol{\Sigma_{ZZ^+}}\boldsymbol{\Sigma_{Z^+}^{-1}}$$

Value

- Mean vector of units for the current layer given y $\mu_{Z|y}$
- Covariance matrix of units for the current layer given y $\Sigma_{Z|y}$

backwardParameterUpdate

Backward parameters update

Description

This function updates parameters from responses to input data. It updates $\mu_{\theta|y}$ and $\Sigma_{\theta|y}$ from the $\theta|y$ distribution for a given layer.

Usage

```
backwardParameterUpdate(mp, Sp, mzF, SzF, SzB, Czp, mzB, idx)
```

Arguments

mp	Mean vector of parameters for the current layer μ_θ
Sp	Covariance matrix of parameters for the current layer Σ_θ
mzF	Mean vector of units for the next layer μ_{Z^+}
SzF	Covariance matrix of units for the next layer Σ_{Z^+}
SzB	Covariance matrix of units for the next layer given y $\Sigma_{Z^+\|y}$
Czp	Covariance matrix between units and parameters for the current layer $\Sigma_{\theta Z^+}$
mzB	Mean vector of units for the next layer given y $\mu_{Z^+\|y}$
idx	Indices for the parameter update step of the current layer

Details

$f(\boldsymbol{\theta}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu_{\theta|y}}, \boldsymbol{\Sigma_{\theta|y}})$ where

$\boldsymbol{\mu_{\theta|y}} = \boldsymbol{\mu_\theta} + \boldsymbol{J_\theta}(\boldsymbol{\mu_{Z^+|y}} - \boldsymbol{\mu_{Z^+}})$

$\boldsymbol{\Sigma_{\theta|y}} = \boldsymbol{\Sigma_\theta} + \boldsymbol{J_\theta}(\boldsymbol{\Sigma_{Z^+|y}} - \boldsymbol{\Sigma_{Z^+}})\boldsymbol{J_\theta^T}$

$\boldsymbol{J_\theta} = \boldsymbol{\Sigma_{\theta Z^+}}\boldsymbol{\Sigma_{Z^+}^{-1}}$

Value

- Mean vector of parameters for the current layer given y $\mu_{\theta|y}$
- Covariance matrix of parameters for the current layer given y $\Sigma_{\theta|y}$

`batchDerivative` *One iteration of the TAGI with derivative calculations*

Description

This function goes through one learning iteration of the neural network model using TAGI with derivative calculations.

Usage

```
batchDerivative(NN, theta, normStat, states, x, Sx, y, dlayer)
```

Arguments

`NN`	Lists the structure of the neural network
`theta`	List of parameters
`normStat`	Normalized statistics
`states`	List of states
`x`	Input data
`Sx`	Variance of input data
`y`	Response data
`dlayer`	Layer from which derivatives will be in respect to

Value

- List of parameters
- List of normalized statistics
- Mean of predicted responses
- Variance of predicted responses
- Mean of first derivative of predicted responses
- Variance of first derivative of predicted responses
- Covariance between derivatives and inputs
- Mean of second derivative of predicted responses

BH *Price of 506 Boston houses.*

Description

This dataset was originally from the StatLib archive. It contains the price and other attributes of 506 Boston houses.

Usage

```
BH
```

Format

A data frame with 506 rows and 14 variables:

CRIM per capita crime rate by town

ZN proportion of residential land zoned for lots over 25,000 sq.ft.

INDUS proportion of non-retail business acres per town

CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)

NOX nitric oxides concentration (parts per 10 million)

RM average number of rooms per dwelling

AGE proportion of owner-occupied units built prior to 1940

DIS weighted distances to five Boston employment centres

RAD index of accessibility to radial highways

TAX full-value property-tax rate per $10,000

PTRATIO pupil-teacher ratio by town

B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town

LSTAT % lower status of the population

MEDV median value of owner-occupied homes in $1000's

Details

The dataset from the TAGI repository was used for comparison purposes, but the original dataset was published by Harrison, D. and Rubinfeld, D.L.

Source

https://github.com/CivML-PolyMtl/TAGI/blob/master/BostonHousing/data/BostonHousing.mat

References

http://lib.stat.cmu.edu/datasets/boston

Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978.

buildCzp *Reformat covariance matrix between units and parameters*

Description

This function properly reformats covariance matrix between units and parameters $\Sigma_{Z\theta}$ for the update step.

Usage

```
buildCzp(Czw, Czb, currenthiddenUnit, prevhiddenUnit, batchSize)
```

Arguments

`Czw`	Covariance matrix between units and weights for the current layer
`Czb`	Covariance matrix between units and baises for the current layer
`currenthiddenUnit`	Number of units in the current layer
`prevhiddenUnit`	Number of units in the previous layer
`batchSize`	Number of observations trained at the same time

Value

Reformatted covariance matrix between units and parameters

buildCzz *Reformat covariance matrix between units of the previous and current layers*

Description

This function properly reformats covariance matrix between units of the previous and current layers Σ_{ZZ^+} for the update step.

Usage

```
buildCzz(Czz, currenthiddenUnit, prevhiddenUnit, batchSize)
```

Arguments

`Czz`	Covariance matrix between units of the previous and current layers
`currenthiddenUnit`	Number of units in the current layer
`prevhiddenUnit`	Number of units in the previous layer
`batchSize`	Number of observations trained at the same time

Value

Reformatted covariance matrix between of the previous and current layers

`catParameters` *Concatenate parameters*

Description

Combines in a single column vector each parameter for all layers.

Usage

```
catParameters(mw, Sw, mb, Sb, mwx, Swx, mbx, Sbx)
```

Arguments

`mw`	Mean of weights for the current layer
`Sw`	Covariance of weights for the current layer
`mb`	Mean of biases for the current layer
`Sb`	Covariance of biases for the current layer
`...`	Other parameters

Value

- Mean vector of weights for the current layer
- Covariance vector of weights for the current layer
- Mean vector of biases for the current layer
- Covariance vector of biases for the current layer

`compressParameters` *Compress parameters*

Description

Put together parameters into a list of parameters.

Usage

```
compressParameters(mw, Sw, mb, Sb, mwx, Swx, mbx, Sbx)
```

Arguments

`mw`	Mean vector of weights for the current layer
`Sw`	Covariance vector of weights for the current layer
`mb`	Mean vector of biases for the current layer
`Sb`	Covariance vector of biases for the current layer
`...`	Other parameters

Value

List of parameters

compressStates — *Compress states*

Description

Put together states into a list of states.

Usage

```
compressStates(mz, Sz, ma, Sa, J, mdxs, Sdxs, mxs, Sxs)
```

Arguments

`mz`	Mean of units for each layer
`Sz`	Covariance of units for each layer
`ma`	Mean of activated units for each layer
`Sa`	Covariance of activated units for each layer
`J`	Jacobian
`...`	Other parameters

Value

List of states

computeError — *Compute error*

Description

This function calculates the Root Mean Square Error (RMSE). It takes as input two vectors (or matrices) with one containing the real y's and the other the predicted y's from the model.

Usage

```
computeError(y, ypred)
```

Arguments

`y`	Response data
`ypred`	Mean of predicted responses

Value

RMSE for the given data

`covariance` *Indices for covariances in the neural network*

Description

This function assigns indices for all covariance elements in the neural network.

Usage

```
covariance(NN)
```

Arguments

`NN`	Lists the structure of the neural network

Value

NN with new elements:

- Indices (weights and activation units) for deterministic matrix F * μ_{WA} for each layer
- Bias indices for deterministic matrix F * μ_B for each layer
- Indices (weights and activation units) for deterministic matrix F * Σ_{ZWA} for each layer
- Indices for the parameter update step for each layer
- Indices for the hidden state update step for each layer
- Indices (weights and activation units) for deterministic matrix F * $\Sigma_{WA\theta}$ for each layer
- Indices for activation unit for each layer
- Bias indices for deterministic matrix F * Σ_B for each layer

---

`covarianceCzp` *Covariance matrices between units and parameters*

Description

This function calculate the covariance matrices between units and parameters Σ_{ZW} and Σ_{ZB} for a given layer.

Usage

```
covarianceCzp(ma, Sp, idxFCwwa, idxFCb)
```

Arguments

`ma`	Mean vector of activation units from previous layer μ_A
`Sp`	Covariance matrix of parameters for the current layer Σ_θ
`idxFCwwa`	Indices for weights and for activation units for the current and previous layers respectively
`idxFCb`	Indices for biases of the current layer

Value

- Covariance matrix between units and biases for the current layer Σ_{ZB}
- Covariance matrix between units and weights for the current layer Σ_{ZW}

`covarianceCzz` *Covariance matrix between units of the previous and current layers*

Description

This function calculate the covariance matrix between units of the previous and current layers Σ_{ZZ+} for a given layer.

Usage

```
covarianceCzz(mp, Sz, J, idxCawa)
```

Arguments

`mp`	Mean vector of parameters for the current layer μ_θ
`Sz`	Covariance matrix of units for the current layer Σ_Z
`J`	Jacobian matrix evaluated at μ_Z
`idxCawa`	Indices for weights and for activation units for the current and previous layers respectively

Value

Covariance matrix between units of previous and current layers Σ_{ZZ+}

`covarianceSa` *Calculate variance of activated units*

Description

This function uses lineratization to estimate the covariance matrix of activation units Σ_A.

Usage

```
covarianceSa(J, Sz)
```

Arguments

`J`	Jacobian matrix evaluated at μ_Z
`Sz`	Covariance matrix of units for the current layer Σ_Z

Value

The activation units covariance matrix Σ_A

`covarianceSz` *Covariance matrix of units*

Description

This function calculate the covariance matrix of the units Σ_Z for a given layer.

Usage

```
covarianceSz(mp, ma, Sp, Sa, idxFSwaF, idxFSwaFb)
```

Arguments

`mp`	Mean vector of parameters for the current layer
`ma`	Mean vector of activation units from previous layer
`Sp`	Covariance matrix of parameters for the current layer
`Sa`	Covariance matrix of activation units from previous layer
`idxFSwaF`	Indices for weights and for activation units for the current and previous layers respectively
`idxFSwaFb`	Indices for biases of the current layer

Value

Covariance matrix of units for the current layer Σ_Z

`covdx` *Covariance between derivatives and hidden states*

Description

This function calculates covariance between derivatives and hidden states. It is not related to the derivative calculation process. It could be used infer Z (hidden states) with the constraint that the derivative of g with respect to Z equals 0.

Usage

```
covdx(mwo, mw, mdgo2, mpdi, mdgoe, Cdozi, Cdizi, ni, no, no2, B)
```

Arguments

`mwo`	Mean vector of weights for the next layer
`mw`	Mean vector of weights for the current layer
`mdgo2`	Mean vector of product of derivatives in second next layer
`mpdi`	Mean vector of first derivative product wd of current layer
`mdgoe`	Mean of product of derivatives at each node in next layer
`Cdozi`	Covariance between derivative (next) and hidden (current) layers
`Cdizi`	Covariance between derivative and hidden layers (same layer)
`ni`	Number of units in current layer
`no`	Number of units in next layer
`no2`	Number of units in second next layer
`B`	Batch size

Value

Covariance between derivative and hidden states

`createDevCellarray` *States initialization (unit matrices)*

Description

Initiliazes neural network derivative states at 1.

Usage

```
createDevCellarray(nodes, numlayers, B, rB)
```

Arguments

`nodes`	Vector which contains the number of nodes at each layer
`numlayers`	Number of layers in the neural network
`B`	Batch size
`rB`	Number of times batch size is repeated

Value

Unit matrices for each layer

`createInitCellwithArray` *Initialization (matrix of lists)*

Description

Initializes a matrix containing lists.

Usage

```
createInitCellwithArray(numlayers)
```

Arguments

`numlayers`	Number of layers in the neural network

Value

Matrix containing empty lists

`createStateCellarray` *States initialization (zero-matrices)*

Description

Initiliazes neural network states at 0.

Usage

```
createStateCellarray(nodes, numlayers, B, rB)
```

Arguments

`nodes`	Vector which contains the number of nodes at each layer
`numlayers`	Number of layers in the neural network
`B`	Batch size
`rB`	Number of times batch size is repeated

Value

Zero-matrices for each layer

`denormalize` *Denormalize data*

Description

This function denormalizes response data processed by the neural network.

Usage

```
denormalize(yn, syn, myntrain, syntrain)
```

Arguments

`yn`	Predicted responses
`syn`	Variance of the predicted responses
`myntrain`	Mean vector of responses from training set
`syntrain`	Variance vector of responses from training set

Value

- Mean of denormalized predicted responses
- Variance of denormalized predicted responses

`derivative` *Derivative calculation*

Description

This function does derivative calculations.

Usage

```
derivative(NN, theta, states, mda, Sda, mdda, Sdda, dlayer)
```

Arguments

`NN`	Lists the structure of the neural network
`theta`	List of parameters
`states`	List of states
`mda`	Mean vectors of activation units' first derivative
`Sda`	Covariance matrices of activation units' first derivative
`mdda`	Mean vectors of activation units' second derivative
`Sdda`	Covariance matrices of activation units' second derivative
`dlayer`	layer from which derivatives will be in respect to

Value

- Mean of first derivative of predicted responses
- Variance of first derivative of predicted responses
- Covariance between derivatives and inputs
- Mean of second derivative of predicted responses

`extractParameters` *Extract parameters*

Description

Extract parameters from list of parameters.

Usage

```
extractParameters(theta)
```

Arguments

`theta`	List of parameters

Value

- Mean vector of weights for the current layer
- Covariance vector of weights for the current layer
- Mean vector of biases for the current layer
- Covariance vector of biases for the current layer

`extractStates` *Extract states*

Description

Extract states from list of states.

Usage

```
extractStates(states)
```

Arguments

`states`	List of states

Value

- Mean of units for each layer
- Covariance of units for each layer
- Mean of activated units for each layer
- Covariance of activated units for each layer
- Jacobian

`fcCombinaisonDnode` *Combination of products of first derivative (iterations on nodes)*

Description

This function calculates mean of combination of products of first derivatives (wd)*(wd). Each node is multiplied to another node in the same layer (including itself). Their weights are both pointing to the same node in the next layer.

Usage

```
fcCombinaisonDnode(mpdi, mw, Sw, mda, Sda, ni, no, B)
```

Arguments

`mpdi`	Mean vector of first derivative product wd of current layer
`mw`	Mean vector of weights for the current layer
`Sw`	Covariance of weights for the current layer
`mda`	Mean vector of activation units' first derivative from current layer
`Sda`	Covariance of activation units' first derivative from current layer
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

Mean array of combination of products of first derivatives

fcCombinaisonDweight *Combination of products of first derivative (iterations on weights)*

Description

This function calculates mean of combination of products of first derivatives (wd)*(wd). Each weight is multiplied to another weight (including itself) from the same node. Each node is multiplied to the same node (in the same layer).

Usage

```
fcCombinaisonDweight(mpdi, mw, Sw, mda, Sda, ni, no, B)
```

Arguments

`mpdi`	Mean vector of first derivative product wd of current layer
`mw`	Mean vector of weights for the current layer
`Sw`	Covariance of weights for the current layer
`mda`	Mean vector of activation units' first derivative from current layer
`Sda`	Covariance of the activation units' first derivative from current layer
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

Mean array of combination of products of first derivatives

fcCombinaisonDweightNode *Combination of squared products of first derivative*

Description

This function calculates mean of squared products of first derivatives (wd)^2. Every products (weight times node) from current layer are considered which results in a (B*ni x no)-matrix.

Usage

```
fcCombinaisonDweightNode(mpdi, mw, Sw, mda, Sda, ni, no, B)
```

Arguments

`mpdi`	Mean vector of first derivative product wd of current layer
`mw`	Mean vector of weights for the current layer
`Sw`	Covariance of weights for the current layer
`mda`	Mean vector of activation units' first derivative from current layer
`Sda`	Covariance of activation units' first derivative from current layer
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

Mean matrix of squared products of first derivatives

`fcCombinaisonDweightNodeAll` *All possible combinations of products of first derivatives*

Description

This function calculates mean of products of first derivatives wd*wd. Since both weight and node are iterated over all products, every products (weight times node) from current layer are considered which results in a (Bni x no x noni)-array. I.e. each dimension of the array represents a single product being multiplied to all other possible products from current layer. Order is as followed: w11d1, w12d2, w13d3, ..., w1nidni, w21d1, w22d2, ..., w2nidni, ... wno1d1, ..., wnonidni

Usage

```
fcCombinaisonDweightNodeAll(mpdi, mpdin, mpdiw, ni, no, B)
```

Arguments

`mpdi`	Mean vector of first derivative product wd of current layer
`mpdin`	Mean array of combination of products of first derivatives (iterations on nodes)
`mpdiw`	Mean array of combination of products of first derivatives (iterations on weights)
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

Mean array of combination of products of first derivatives

`fcCovaddddddw` *Covariance between first and second derivatives from consecutive layers*

Description

This function calculates covariance between weights and second derivatives and covariance between first and second derivatives from consecutive layers.

Usage

```
fcCovaddddddw(mao, mai, mdao, Caoai, Cdodi, Caow, Cdow, acto, acti, ni, no, B)
```

Arguments

`mao`	Mean vector of activation units from next layer
`mai`	Mean vector of activation units from current layer
`mdao`	Mean vector of activation units' first derivative from next layer
`Caoai`	Covariance between activation units from current and next layers
`Cdodi`	Covariance between first derivatives from current and next layers
`Caow`	Covariance between activation units and weights
`Cdow`	Covariance between derivatives and weights
`acto`	Activation function index for next layer defined by `activationFunIndex`
`acti`	Activation function index for current layer defined by `activationFunIndex`
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

- Covariance between first and second derivatives from consecutive layers
- Covariance between second derivatives from next layer and weights

`fcCovawaa` *Covariance between activation units and weights*

Description

This function calculates covariance between activation units and weights and covariance between activation units from consecutive layers.

Usage

```
fcCovawaa(mw, Sw, Jo, mai, Sai, ni, no, B)
```

Arguments

`mw`	Mean vector of weights for the current layer
`Sw`	Covariance of weights for the current layer
`Jo`	Jacobian of next layer
`mai`	Mean vector of activation units from current layer
`Sai`	Covariance of activation units from current layer
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

- Covariance between activation units and weights
- Covariance between activation units from current and next layers

`fcCovaz` *Covariance between activation and hidden units*

Description

This function calculates covariance between activation and hidden units.

Usage

```
fcCovaz(Jo, J, Sz, mw, ni, no, B)
```

Arguments

`Jo`	Jacobian of next layer
`J`	Jacobian of current layer
`Sz`	Covariance of units from current layer
`mw`	Mean vector of weights for the current layer
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

- Covariance between activation and hidden layers (same layer)
- Covariance between activation (next) and hidden (current) layers

fcCovdaddd — *Covariance between first and second derivatives from consecutive layers*

Description

This function calculates covariance between activation units and first derivatives and covariance between first and second derivatives from consecutive layers.

Usage

```
fcCovdaddd(mao, mai, mdai, Caoai, Cdodi, acti, ni, no, B)
```

Arguments

`mao`	Mean vector of activation units from next layer
`mai`	Mean vector of activation units from current layer
`mdai`	Mean vector of activation units' first derivative from current layer
`Caoai`	Covariance between activation units from current and next layers
`Cdodi`	Covariance between derivatives from current and next layers
`acti`	Activation function index for current layer defined by `activationFunIndex`
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

- Covariance between first derivatives from next layer and activation units from current layer
- Covariance between first and second derivatives from consecutive layers

fcCovDlayer — *Covariance between products of derivatives and weights*

Description

This function calculates covariance between products of derivatives and weights from consecutive layers.

Usage

```
fcCovDlayer(mdgo2, mwo, Cdowdi, ni, no, no2, B)
```

Arguments

`mdgo2`	Mean vector of product of derivatives in second next layer
`mwo`	Mean vector of weights for the next layer
`Cdowdi`	Covariance between derivatives and weights times derivatives
`ni`	Number of units in current layer
`no`	Number of units in next layer
`no2`	Number of units in second next layer
`B`	Batch size

Value

Covariance between weights times derivatives from consecutive layers

---

`fcCovdwd` *Covariance between derivatives and weights*derivatives*

Description

This function calculates covariance between derivatives and weights and covariance between derivatives from consecutive layers.

Usage

```
fcCovdwd(md, mw, Cdow, Cdodi, ni, no, B)
```

Arguments

`md`	Mean vector of derivatives
`mw`	Mean vector of weights for the current layer
`Cdow`	Covariance between derivatives and weights
`Cdodi`	Covariance between derivatives from current and next layers
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

Covariance between derivatives and weights times derivatives

fcCovdwddd — *Covariance between derivatives and weights*

Description

This function calculates covariance between derivatives and weights and covariance between derivatives from consecutive layers.

Usage

```
fcCovdwddd(mao, Sao, mai, Sai, Caow, Caoai, acto, acti, ni, no, B)
```

Arguments

`mao`	Mean vector of activation units from next layer
`Sao`	Covariance of activation units from next layer
`mai`	Mean vector of activation units from current layer
`Sai`	Covariance of activation units from current layer
`Caow`	Covariance between activation units and weights
`Caoai`	Covariance between activation units from current and next layers
`acto`	Activation function index for next layer defined by `activationFunIndex`
`acti`	Activation function index for current layer defined by `activationFunIndex`
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

- Covariance between derivatives and weights
- Covariance between derivatives from current and next layers

fcCovdz — *Covariance between derivatives and hidden Units*

Description

This function calculates covariance between derivatives and hidden units.

Usage

```
fcCovdz(mao, mai, Caizi, Caozi, acto, acti, ni, no, B)
```

Arguments

`mao`	Mean vector of activation units from next layer
`mai`	Mean vector of activation units from current layer
`Caizi`	covariance between activation and hidden layers (same layer)
`Caozi`	covariance between activation (next layer) and hidden (current) layers
`acto`	Activation function index for next layer defined by `activationFunIndex`
`acti`	Activation function index for current layer defined by `activationFunIndex`
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

- Covariance between derivative (next) and hidden (current) layers
- Covariance between derivative and hidden layers (same layer)

`fcCovwdo2wdiwdi` *Covariance between products in (same) next and current layers*

Description

This function calculates covariance cov(wdo^2,wdiwdi) of weights times derivatives products terms when there are two products in both next and current layers. The product fom next layer is the same squared.

Usage

```
fcCovwdo2wdiwdi(mpdo, Cwdowdiwdi)
```

Arguments

`mpdo`	Mean vector of first derivative product wd of next layer
`Cwdowdiwdi`	Covariance cov(wdo,wdi*wdi) of weights times derivatives products terms when there are one product in next layer and two in current

Value

Covariance cov(wdo^2,wdi*wdi) of weights times derivatives products terms when there are two products in both next and current layers

fcCovwdowdi2 — *Covariance between next layer product and current layer squared product*

Description

This function calculates covariance cov(wdo,(wdi)^2) of weights times derivatives products terms when there are one product in next layer and two squared in current.

Usage

```
fcCovwdowdi2(mpdi, Cdgoddgik)
```

Arguments

`mpdi`	Mean vector of first derivative product wd of current layer
`Cdgoddgik`	Covariance between weights times derivatives from consecutive layers

Value

Covariance cov(wdo,(wdi)^2) of weights times derivatives products terms when there is one product in next layer and two squared in current

fcCovwdowdiwdi — *Covariance between next layer product and current layer multiplied products*

Description

This function calculates covariance cov(wdo,wdiwdi) of weights times derivatives products terms when there are one product in next layer and two in current.

Usage

```
fcCovwdowdiwdi(mpdi, Cdgoddgik, ni, no, B)
```

Arguments

`mpdi`	Mean vector of first derivative product wd of current layer
`Cdgoddgik`	Covariance between weights times derivatives from consecutive layers
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

Covariance cov(wdo,wdi*wdi) of weights times derivatives products terms when there is one product in next layer and two in current

`fcCwdowdowdiwdi`	*Covariance between next layer multiplied products and current layer multiplied products (minimum 3 hidden layers)*

Description

This function calculates covariance cov(wdowdo,wdiwdi) where all terms can be different. It is used when there are at least 3 hidden layers and second next layer is a product of only two terms (wdo2).

Usage

```
fcCwdowdowdiwdi(mpdi, mpdo, Cdgodgi, acti, ni, no, no2, B)
```

Arguments

`mpdi`	Mean vector of first derivative product wd of current layer
`mpdo`	Mean vector of first derivative product wd of next layer
`Cdgodgi`	Covariance between weights times derivatives from consecutive layers
`acti`	Activation function index for current layer defined by `activationFunIndex`
`ni`	Number of units in current layer
`no`	Number of units in next layer
`no2`	Number of units in second next layer
`B`	Batch size

Value

Covariance cov(wdowdo,wdiwdi) where all terms can be different

`fcCwdowdowdiwdi_4hl`	*Covariance between next layer multiplied products and current layer multiplied products (minimum 4 hidden layers)*

Description

This function calculates covariance cov(wdowdo,wdiwdi) where all terms can be different. It is used when there are at least 4 hidden layers.

Usage

```
fcCwdowdowdiwdi_4hl(mpdi, mpdo, mdgo2, Cdgodgi, acti, ni, no, no2, B)
```

Arguments

`mpdi`	Mean vector of first derivative product wd of current layer
`mpdo`	Mean vector of first derivative product wd of next layer
`mdgo2`	Mean vector of product of derivatives in second next layer
`Cdgodgi`	Covariance between weights times derivatives from consecutive layers
`acti`	Activation function index for current layer defined by `activationFunIndex`
`ni`	Number of units in current layer
`no`	Number of units in next layer
`no2`	Number of units in second next layer
`B`	Batch size

Value

Covariance cov(wdowdo,wdiwdi) where all terms can be different

---

`fcCwdowdowwdi2` *Covariance between next layer multiplied products and current layer multiplied products (same derivative)*

Description

This function calculates covariance cov(wdowdo,wdiwdi) where the di terms are the same, when next second layer involves only a product term (wddo2).

Usage

```
fcCwdowdowwdi2(mpdi, mpdo, Cdgodgi, acti, ni, no, no2, B)
```

Arguments

`mpdi`	Mean vector of first derivative product wd of current layer
`mpdo`	Mean vector of first derivative product wd of next layer
`Cdgodgi`	Covariance between weights times derivatives from consecutive layers
`acti`	Activation function index for current layer defined by `activationFunIndex`
`ni`	Number of units in current layer
`no`	Number of units in next layer
`no2`	Number of units in second next layer
`B`	Batch size

Value

Covariance cov(wdowdo,wdiwdi) where the di terms are the same fcCwdowdowwdi2_3hl *Covariance between next layer multiplied products and current layer multiplied products (same derivative, minimum 3 hidden layers)*

Description

This function calculates covariance cov(wdowdo,wdiwdi) where the di terms are the same when next second layer involves multiplied terms (wdo2wdo2). It is used when there are at least 3 hidden layers.

Usage

```
fcCwdowdowwdi2_3hl(mpdi, mpdo, mdgo2, Cdgodgi, acti, ni, no, no2, B)
```

Arguments

mpdi	Mean vector of first derivative product wd of current layer
mpdo	Mean vector of first derivative product wd of next layer
mdgo2	Mean vector of product of derivatives in second next layer
Cdgodgi	Covariance between weights times derivatives from consecutive layers
acti	Activation function index for current layer defined by activationFunIndex
ni	Number of units in current layer
no	Number of units in next layer
no2	Number of units in second next layer
B	Batch size

Value

Covariance cov(wdowdo,wdiwdi) where the di terms are the same

fcDerivative *Derivatives for fully connected layers*

Description

This function calculates mean and variance of derivatives and covariance of derivative and input layers.

Usage

```
fcDerivative(
  mw,
  Sw,
  mwo,
  Jo,
  J,
  mao,
  Sao,
  mai,
  Sai,
  Szi,
  mdai,
  Sdai,
  mdgo,
  mdgoe,
  Sdgo,
  mdgo2,
  acto,
  acti,
  ni,
  no,
  no2,
  B
)
```

Arguments

`mw`	Mean vector of weights for the current layer
`Sw`	Covariance of weights for the current layer
`mwo`	Mean vector of weights for the next layer
`Jo`	Jacobian of next layer
`J`	Jacobian of current layer
`mao`	Mean vector of activation units from next layer
`Sao`	Covariance of activation units from next layer
`mai`	Mean vector of activation units from current layer
`Sai`	Covariance of activation units from current layer
`Szi`	Covariance of units from current layer
`mdai`	Mean vector of activation units' first derivative from current layer
`Sdai`	Covariance of activation units' first derivative from current layer
`mdgo`	Mean vector of product of derivatives in next layer
`mdgoe`	Mean of product of derivatives at each node in next layer
`Sdgo`	Variance of first derivatives in next layer
`mdgo2`	Mean vector of product of derivatives in second next layer
`acto`	Activation function index for next layer defined by `activationFunIndex`
`acti`	Activation function index for current layer defined by `activationFunIndex`
`ni`	Number of units in current layer
`no`	Number of units in next layer
`no2`	Number of units in second next layer
`B`	Batch size

Value

Mean vector of first derivatives

Covariance matrix of first derivatives

Covariance matrix of first derivative and input layer

Covariance between activation units and weights

Covariance between activation units from current and next layers

Covariance between first derivatives and weights

Covariance between first derivatives from current and next layers

Covariance between first derivatives from next layer and weights times derivatives from current layer

`fcDerivative2` *Second derivatives for fully connected layers*

Description

This function calculates mean of product of derivatives, when new product term involves second derivatives (wdd).

Usage

```
fcDerivative2(
  mw,
  mwo,
  mao,
  mai,
  mdai,
  mddai,
  mpddi,
  mdgo,
  mdgo2,
  Caoai,
  Cdow,
  Cdodi,
  acti,
  ni,
  no,
  no2,
  B
)
```

Arguments

`mw`	Mean vector of weights for the current layer
`mwo`	Mean vector of weights for the next layer
`mao`	Mean vector of activation units from next layer
`mai`	Mean vector of activation units from current layer

mdai	Mean vector of activation units' first derivative from current layer
mddai	Mean vector of activation units' second derivative from current layer
mpddi	Mean vector of second derivative product wdd of current layer
mdgo	Mean vector of product of derivatives in next layer
mdgo2	Mean vector of product of derivatives in second next layer
Caoai	Covariance between activation units from current and next layers
Cdow	Covariance between first derivatives and weights
Cdodi	Covariance between first derivatives from current and next layers
acti	Activation function index for current layer defined by `activationFunIndex`
ni	Number of units in current layer
no	Number of units in next layer
no2	Number of units in second next layer
B	Batch size

Value

Mean of product terms for second derivative calculations

---

fcDerivative3 — *Products of first derivatives multiplied to second derivative for fully connected layers*

Description

This function calculates mean of product of derivatives, when new product term involves product of two first derivatives (wdwd) from the same layer multiplied to second derivatives (wdd) from next layer.

Usage

```
fcDerivative3(
  mw,
  Sw,
  mwo,
  mao,
  mai,
  mdao,
  mdai,
  Sdai,
  mpdi,
  mdgo,
  mdgo2,
  Caow,
  Caoai,
  Cdow,
  Cdodi,
  acto,
  acti,
  ni,
  no,
  no2,
  B,
  dlayer
)
```

Arguments

mw	Mean vector of weights for the current layer
Sw	Covariance of weights for the current layer
mwo	Mean vector of weights for the next layer
mao	Mean vector of activation units from next layer
mai	Mean vector of activation units from current layer
mdao	Mean vector of activation units' first derivative from next layer
mdai	Mean vector of activation units' first derivative from current layer
Sdai	Covariance of activation units' first derivative from current layer
mpdi	Mean vector of first derivative product wd of current layer
mdgo	Mean vector of product of derivatives in next layer
mdgo2	Mean vector of product of derivatives in second next layer
Caow	Covariance between activation units and weights
Caoai	Covariance between activation units from current and next layers
Cdow	Covariance between first derivatives and weights
Cdodi	Covariance between first derivatives from current and next layers
acto	Activation function index for next layer defined by `activationFunIndex`
acti	Activation function index for current layer defined by `activationFunIndex`
ni	Number of units in current layer
no	Number of units in next layer
no2	Number of units in second next layer
B	Batch size
dlayer	TRUE if derivatives will be in respect to current layer

Value

Mean of product terms for second derivative calculations

fcDerivative4 — *Products of first derivatives multiplied to products of first derivatives for fully connected layers*

Description

This function calculates mean of product of derivatives, when new product term involves product of two first derivatives (wdwd) from the same layer multiplied to product of two first derivatives (wdwd) from next layer, when second next layer is second derivatives (wdd).

Usage

```
fcDerivative4(
  mw,
  Sw,
  mwo,
  mao,
  mai,
  mdao,
  mdai,
  Sdai,
  mpdo,
  mpdi,
  mdgo,
  mdgo2,
  Cdowdi,
  acto,
  acti,
  ni,
  no,
  no2,
  B,
  dlayer
)
```

Arguments

`mw`	Mean vector of weights for the current layer
`Sw`	Covariance of weights for the current layer
`mwo`	Mean vector of weights for the next layer
`mao`	Mean vector of activation units from next layer
`mai`	Mean vector of activation units from current layer
`mdao`	Mean vector of activation units' first derivative from next layer
`mdai`	Mean vector of activation units' first derivative from current layer
`Sdai`	Covariance of activation units' first derivative from current layer
`mpdo`	Mean vector of first derivative product wd of next layer
`mpdi`	Mean vector of first derivative product wd of current layer
`mdgo`	Mean vector of product of derivatives in next layer

mdgo2	Mean vector of product of derivatives in second next layer
Cdowdi	Covariance between first derivatives from next layer and weights times derivatives from current layer
acto	Activation function index for next layer defined by activationFunIndex
acti	Activation function index for current layer defined by activationFunIndex
ni	Number of units in current layer
no	Number of units in next layer
no2	Number of units in second next layer
B	Batch size
dlayer	TRUE if derivatives will be in respect to current layer

Value

Mean of product terms for second derivative calculations

---

fcDerivative5 *Products of first derivatives multiplied to products of first derivatives (not only last layer) for fully connected layers*

Description

This function calculates mean of product of derivatives, when new product term involves product of two first derivatives (wdwd) from the same layer multiplied to product of two first derivatives (wdwd) from next and second next layers.

Usage

```
fcDerivative5(
  mw,
  Sw,
  mwo,
  mao,
  mai,
  mdao,
  mdai,
  Sdai,
  mpdo,
  mpdi,
  mdgo,
  mdgo2,
  Cdowdi,
  acto,
  acti,
  ni,
  no,
  no2,
  B,
  dlayer
)
```

Arguments

`mw`	Mean vector of weights for the current layer
`Sw`	Covariance of weights for the current layer
`mwo`	Mean vector of weights for the next layer
`mao`	Mean vector of activation units from next layer
`mai`	Mean vector of activation units from current layer
`mdao`	Mean vector of activation units' first derivative from next layer
`mdai`	Mean vector of activation units' first derivative from current layer
`Sdai`	Covariance of activation units' first derivative from current layer
`mpdo`	Mean vector of first derivative product wd of next layer
`mpdi`	Mean vector of first derivative product wd of current layer
`mdgo`	Mean vector of product of derivatives in next layer
`mdgo2`	Mean vector of product of derivatives in second next layer
`Cdowdi`	Covariance between derivatives from next layer and weights times derivatives from current layer
`acto`	Activation function index for next layer defined by `activationFunIndex`
`acti`	Activation function index for current layer defined by `activationFunIndex`
`ni`	Number of units in current layer
`no`	Number of units in next layer
`no2`	Number of units in second next layer
`B`	Batch size
`dlayer`	TRUE if derivatives will be in respect to current layer

Value

Mean of product terms for second derivative calculations

`fcHiddenStateBackwardPass`

Backpropagation (states' deltas) for fully connected layers (many observations)

Description

This function calculates units' deltas at a given layer when using more than one observation at the time.

Usage

```
fcHiddenStateBackwardPass(Sz, Sxs, J, mw, deltaM, deltaS, ni, no, B, rB)
```

Arguments

`Sz`	Covariance of units from current layer
`Sxs`	Null by default (not used yet)
`J`	Jacobian of current layer
`mw`	Mean vector of weights for the current layer
`deltaM`	Delta of mean vector of the next layer units given y $\mu_Z|y$
`deltaS`	Delta of covariance matrix of the next layer units given y $\Sigma_Z|y$
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size
`rB`	Number of times batch size is repeated

Value

- Delta of mean vector of current layer units given y $\mu_Z|y$
- Delta of covariance matrix of current layer units given y $\Sigma_Z|y$

`fcHiddenStateBackwardPassB1`

Backpropagation (states' deltas) for fully connected layers (one observation)

Description

This function calculates units' deltas at a given layer when using one observation at the time.

Usage

```
fcHiddenStateBackwardPassB1(Sz, Sxs, J, mw, deltaM, deltaS, ni, no)
```

Arguments

`Sz`	Covariance of the units from current layer
`Sxs`	Null by default (not used yet)
`J`	Jacobian of current layer
`mw`	Mean vector of weights for the current layer
`deltaM`	Delta of mean vector of next layer units given y $\mu_Z|y$
`deltaS`	Delta of covariance matrix of next layer units given y $\Sigma_Z|y$
`ni`	Number of units in current layer
`no`	Number of units in next layer

Value

- Delta of mean vector of current layer units given y $\mu_Z|y$
- Delta of covariance matrix of current layer units given y $\Sigma_Z|y$

fcMeanDlayer2array — *Mean of weights times derivatives products terms ((wdo*wdo) x (wwdi^2))*

Description

This function calculates mean of weights times derivatives products terms when adding two of those products from current layer to already calculated expectation that ended with one such product of next layer (i.e. (wdowdo) x (wwdi^2)). Mean terms are in array format. Once added, rows need to be summed to aggregate expectations by node*weight combinations of current layer.

Usage

```
fcMeanDlayer2array(mpdi2w, mdgo, Cwdowdowwdi2, ni, no, B)
```

Arguments

`mpdi2w`	Combination of products of first derivative of current layer (wd)*(wd) (iterations on weights on the same node)
`mdgo`	Mean of product of derivatives in next layer
`Cwdowdowwdi2`	Covariance cov(wdowdo,wdiwdi) of weights times derivatives products terms, where the di terms are the same
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

Mean of weights times derivatives products terms

fcMeanDlayer2row — *Mean of weights times derivatives products terms squared (wdo x (wdi*wdi))*

Description

This function calculates mean of weights times derivatives products terms when adding two of those products from current layer to already calculated expectation that ended with one such product of next layer (i.e. wdo x (wdiwdi)). Mean terms are in array format. Once added, rows need to be summed to aggregate expectations by node*node combinations of current layer.

Usage

```
fcMeanDlayer2row(mpdi, mpdi2, mdgo, Cwdowdiwdi, ni, no, no2, B)
```

Arguments

`mpdi`	Mean vector of first derivative product wd of current layer
`mpdi2`	Mean array of combination of products of first derivatives
`mdgo`	Mean vector of product of derivatives in next layer
`Cwdowdiwdi`	Covariance cov(wdo,(wdi*wdi)) of weights times derivatives products terms when there is one product in next layer and two in current
`ni`	Number of units in current layer
`no`	Number of units in next layer
`no2`	Number of units in second next layer
`B`	Batch size

Value

Mean of weights times derivatives products terms

`fcMeanVar` *Mean and covariance vectors of units (many observations)*

Description

This function calculate the mean vector of units μ_Z and the covariance matrix of the units $\Sigma_Z$for a given layer.

Usage

```
fcMeanVar(mz, Sz, mw, Sw, mb, Sb, ma, Sa, ni, no, B, rB)
```

Arguments

`mw`	Mean vector of weights for the current layer
`Sw`	Covariance of weights for the current layer
`mb`	Mean vector of biases for the current layer
`Sb`	Covariance of biases for the current layer
`ma`	Mean vector of activation units from previous layer
`Sa`	Covariance of activation units from previous layer
`ni`	Number of units in previous layer
`no`	Number of units in current layer
`B`	Batch size
`rB`	Number of times batch size is repeated

Value

- Mean vector of units for the current layer μ_Z
- Covariance matrix of units for the current layer Σ_Z

fcMeanVarB1 *Mean and covariance vectors of units (one observation)*

Description

This function calculate the mean vector of units μ_Z and the covariance matrix of the units $\Sigma_Z$for a given layer.

Usage

```
fcMeanVarB1(mw, Sw, mb, Sb, ma, Sa, ni, no)
```

Arguments

mw	Mean vector of weights for the current layer
Sw	Covariance of weights for the current layer
mb	Mean vector of biases for the current layer
Sb	Covariance of biases for the current layer
ma	Mean vector of activation units from previous layer
Sa	Covariance of activation units from previous layer
ni	Number of units in previous layer
no	Number of units in current layer

Value

- Mean vector of units for the current layer μ_Z
- Covariance matrix of units for the current layer Σ_Z

fcMeanVarDlayer *Mean and variance of weights times derivatives products terms*

Description

This function calculates mean and variance of weights times derivatives products terms.

Usage

```
fcMeanVarDlayer(mx, Sx, my, mye, Sy, Cxy, ni, no, no2, B)
```

Arguments

`mx`	Mean vector of inputs
`Sx`	Variance of inputs
`my`	Mean vector of outputs
`mye`	Mean derivatives at each node in next layer
`Sy`	Variance of outputs
`Cxy`	Covariance between inputs and outputs
`ni`	Number of units in current layer
`no`	Number of units in next layer
`no2`	Number of units in second next layer
`B`	Batch size

Value

- Mean of weights times derivatives products terms
- Covariance between weights times derivatives products terms

`fcMeanVarDnode` *Mean and covariance of derivatives*

Description

This function calculates the mean vector and the covariance matrix for derivatives.

Usage

```
fcMeanVarDnode(mw, Sw, mda, Sda, ni, no, B)
```

Arguments

`mw`	Mean vector of weights for the current layer
`Sw`	Covariance of weights for the current layer
`mda`	Mean vector of activation units' derivative from current layer
`Sda`	Covariance of activation units' derivative from current layer
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size

Value

- Mean vector of derivatives
- Covariance matrix of derivatives

fcParameterBackwardPass *Backpropagation (parameters' deltas) for fully connected layers (many observations)*

Description

This function calculates parameters' deltas at a given layer when using more than one observation at the time.

Usage

```
fcParameterBackwardPass(
  deltaMw,
  deltaSw,
  deltaMb,
  deltaSb,
  Sw,
  Sb,
  ma,
  deltaMr,
  deltaSr,
  ni,
  no,
  B,
  rB
)
```

Arguments

`deltaMw`	next layer delta of mean vector of weights given y $\mu_\theta|y$
`deltaSw`	next layer delta of covariance matrix of weights given y $\Sigma_\theta|y$
`deltaMb`	next layer delta of mean vector of biases given y $\mu_\theta|y$
`deltaSb`	next layer delta of covariance matrix of biases given y $\Sigma_\theta|y$
`Sw`	Covariance of weights for the current layer
`Sb`	Covariance of biases for the current layer
`ma`	Mean vector of activation units for the current layer
`deltaMr`	Delta of mean vector of next layer units given y $\mu_Z|y$
`deltaSr`	Delta of covariance matrix of next layer units given y $\Sigma_Z|y$
`ni`	Number of units in current layer
`no`	Number of units in next layer
`B`	Batch size
`rB`	Number of times batch size is repeated

Value

- Delta of mean vector of weights given y $\mu_{\theta}|y$
- Delta of covariance matrix of weights given y $\Sigma_{\theta}|y$
- Delta of mean vector of biases given y $\mu_{\theta}|y$
- Delta of covariance matrix of biases given y $\Sigma_{\theta}|y$

---

fcParameterBackwardPassB1

Backpropagation (parameters' deltas) for fully connected layers (one observation)

Description

This function calculates parameters' deltas at a given layer when using one observation at the time.

Usage

```
fcParameterBackwardPassB1(Sw, Sb, ma, deltaMr, deltaSr, ni, no)
```

Arguments

`Sw`	Covariance of weights for the current layer
`Sb`	Covariance of biaises for the current layer
`ma`	Mean vector of activation units for the current layer
`deltaMr`	Delta of mean vector of next layer units given y $\mu_{Z}|y$
`deltaSr`	Delta of covariance matrix of next layer units given y $\Sigma_{Z}|y$
`ni`	Number of units in current layer
`no`	Number of units in next layer

Value

- Delta of mean vector of weights given y $\mu_{\theta}|y$
- Delta of covariance matrix of weights given y $\Sigma_{\theta}|y$
- Delta of mean vector of biases given y $\mu_{\theta}|y$
- Delta of covariance matrix of biaises given y $\Sigma_{\theta}|y$

feedBackward *Backpropagation*

Description

This function feeds the neural network backward from responses to input data.

Usage

```
feedBackward(NN, mp, Sp, mz, Sz, Czw, Czb, Czz, y)
```

Arguments

NN	Lists the structure of the neural network
mp	Mean vectors of parameters for each layer μ_θ
Sp	Covariance matrices of parameters for each layer Σ_θ
mz	Mean vectors of units for each layer μ_Z
Sz	Covariance matrices of units for each layer Σ_Z
Czw	Covariance matrices between units and weights for each layer Σ_{ZW}
Czb	Covariance matrices between units and biases for each layer Σ_{ZB}
Czz	Covariance matrices between previous and current units for each layer Σ_{ZZ^+}
y	Response data

Value

- Updated mean vectors of parameters for each layer μ_θ
- Updated covariance matrices of parameters for each layer Σ_θ

See Also

backwardhiddenStateUpdate, backwardParameterUpdate, forwardhiddenStateUpdate

feedForward *Forward uncertainty propagation*

Description

This function feeds the neural network forward from input data to responses.

Usage

```
feedForward(NN, x, mp, Sp)
```

Arguments

`NN`	Lists the structure of the neural network
`x`	Input data
`mp`	Mean vectors of parameters for each layer μ_θ
`Sp`	Covariance matrices of parameters for each layer Σ_θ

Value

- Mean vectors of units for each layer μ_Z
- Covariance matrices of units for each layer Σ_Z
- Covariance matrices between units and weights for each layer Σ_{ZW}
- Covariance matrices between units and biases for each layer Σ_{ZB}
- Covariance matrices between previous and current units for each layer Σ_{ZZ+}

`feedForwardPass` *Forward uncertainty propagation for derivative calculation*

Description

This function feeds the neural network forward from input data to responses and considers components required for derivative calculations.

Usage

```
feedForwardPass(NN, theta, states)
```

Arguments

`NN`	Lists the structure of the neural network
`theta`	List of parameters
`states`	List of states

Value

- Updated states
- Mean vectors of activation units' first derivative
- Covariance matrices of activation units' first derivative
- Mean vectors of activation units' second derivative
- Covariance matrices of activation units' second derivative

`forwardHiddenStateUpdate` *Last hidden layer states update*

Description

This function updates last hidden layer units using responses. It updates $\mu_{Z^{(0)}|y}$ and $\Sigma_{Z^{(0)}|y}$ from the $Z^{(0)}|y$ distribution.

Usage

```
forwardHiddenStateUpdate(mz, Sz, mzF, SzF, Cyz, y)
```

Arguments

`mz`	Mean vector of units for the last hidden layer $\mu_{X^{(0)}}$
`Sz`	Covariance matrix of units for the last hidden layer $\Sigma_{Z^{(0)}}$
`mzF`	Mean vector of units for the output layer μ_y
`SzF`	Covariance matrix of tunits for the output layer Σ_y
`Cyz`	Covariance matrix between last hidden layer units and responses $\Sigma_{YZ^{(0)}}$
`y`	Response data

Details

$f(\boldsymbol{z}^{(0)}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{z}^{(0)}; \boldsymbol{\mu}_{\boldsymbol{Z}^{(0)}|\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{Z}^{(0)}|\boldsymbol{y}})$ where

$\boldsymbol{\mu}_{\boldsymbol{Z}^{(0)}|\boldsymbol{y}} = \boldsymbol{\mu}_{\boldsymbol{Z}^{(0)}} + \boldsymbol{\Sigma}_{\boldsymbol{YZ}^{(0)}}^{\boldsymbol{T}} \boldsymbol{\Sigma}_{\boldsymbol{Y}}^{-1} (\boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{Y}})$

$\boldsymbol{\Sigma}_{\boldsymbol{Z}^{(0)}|\boldsymbol{y}} = \boldsymbol{\Sigma}_{\boldsymbol{Z}^{(0)}} - \boldsymbol{\Sigma}_{\boldsymbol{YZ}^{(0)}}^{\boldsymbol{T}} \boldsymbol{\Sigma}_{\boldsymbol{Y}}^{-1} \boldsymbol{\Sigma}_{\boldsymbol{YZ}^{(0)}}$

Value

- Mean vector of last hidden layer units given y $\mu_{Z^{(0)}|y}$
- Covariance matrix of last hidden layer units given y $\Sigma_{Z^{(0)}|y}$

`globalParameterUpdate` *Backpropagation (parameters update)*

Description

This function updates parameters.

Usage

```
globalParameterUpdate(theta, deltaTheta)
```

Arguments

`theta`	List of parameters
`deltaTheta`	Parameters' deltas (mean and covariance for each)

Value

List of updated parameters

`hiddenStateBackwardPass` *Backpropagation (states' deltas)*

Description

This function calculates states' deltas.

Usage

```
hiddenStateBackwardPass(NN, theta, states, y, Sy, udIdx)
```

Arguments

`NN`	Lists the structure of the neural network
`theta`	List of parameters
`states`	List of states
`y`	Response data
`Sy`	Variance of responses
`udIdx`	Specific update IDs

Value

- Delta of mean vector of units given y $\mu_Z|y$ at all layers
- Delta of covariance matrix of units given y $\Sigma_Z|y$ at all layers

`initialization`	*Network initialization*

Description

Verify and add components to the neural network structure.

Usage

```
initialization(NN)
```

Arguments

`NN`	Lists the structure of the neural network

Value

- NN with all required components
- States of all required elements to perform TAGI

`initialization_net`	*Network initialization*

Description

Verify and add components to the neural network structure.

Usage

```
initialization_net(NN)
```

Arguments

`NN`	Lists the structure of the neural network

Value

NN with all required components

`initializeInputs` *Input initialization*

Description

Initializes neural network inputs.

Usage

```
initializeInputs(states, mz0, Sz0, ma0, Sa0, J0, mdxs0, Sdxs0, mxs0, Sxs0, xsc)
```

Arguments

`states`	States of the neural network
`mz0`	Input data
`Sz0`	Variance of input data
`ma0`	Activated input data
`Sa0`	Variance of activated input data
`J0`	Jacobian
`...`	Other parameters

Value

States of the neural network

`initializeStates` *States initialization*

Description

Initiliazes neural network states.

Usage

```
initializeStates(nodes, B, rB, xsc)
```

Arguments

`nodes`	Vector which contains the number of nodes at each layer
`B`	Batch size
`rB`	Number of times batch size is repeated

Value

States of the neural network

`initializeWeightBias` *Weights and biases initialization*

Description

This function initializes the first weights and biases of the neural network.

Usage

```
initializeWeightBias(NN)
```

Arguments

`NN`	Lists the structure of the neural network

Value

- Initial mean vectors of parameters for each layer
- Initial covariance matrices of parameters for each layer

`initializeWeightBiasD` *Weights and biases initialization for calculating derivatives*

Description

This function initializes the first weights and biases of the neural network.

Usage

```
initializeWeightBiasD(NN)
```

Arguments

`NN`	Lists the structure of the neural network

Value

All parameters required in the neural network to perform derivative calculations

`innovationVector` *Last hidden layer states' deltas update*

Description

This function updates hidden layer units' deltas using next hidden layer' deltas. It updates $\mu_{Z|y}$ and $\Sigma_{Z|y}$ from the $Z|y$ distribution.

Usage

```
innovationVector(SzF, dMz, dSz)
```

Arguments

`SzF`	Covariance matrix of units for the next layer Σ_y
`dMz`	Delta of mean vector of units for the next hidden layer μ_Z
`dSz`	Delta of covariance matrix of units for the next hidden layer Σ_Z

Details

$f(\boldsymbol{z}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu_{Z|y}}, \boldsymbol{\Sigma_{Z|y}})$ where

$\boldsymbol{\mu_{Z|y}} = \boldsymbol{\mu_Z} + \boldsymbol{J_Z}(\boldsymbol{\mu_{Z^+|y}} - \boldsymbol{\mu_{Z^+}})$

$\boldsymbol{\Sigma_{Z|y}} = \boldsymbol{\Sigma_Z} + \boldsymbol{J_Z}(\boldsymbol{\Sigma_{Z^+|y}} - \boldsymbol{\Sigma_{Z^+}})\boldsymbol{J_Z^T}$

$\boldsymbol{J_Z} = \boldsymbol{\Sigma_{ZZ^+}}\boldsymbol{\Sigma_{Z^+}^{-1}}$

Value

- Delta of mean vector of current hidden layer units given y $\mu_Z|y$
- Delta of covariance matrix of current hidden layer units given y $\Sigma_Z|y$

`layerEncoder` *Layer encoder*

Description

Add layer encoder to the neural network structure.

Usage

```
layerEncoder(NN)
```

Arguments

`NN`	Lists the structure of the neural network

Value

NN with layer encoder

`loglik` *Compute log-likelihood*

Description

This function calculates the log-likelihood (LL). It takes as input three vectors (or matrices) with one containing the real y's, one with the predicted y's from the model and the last one with the variance of the y's.

Usage

```
loglik(y, ypred, Vpred)
```

Arguments

`y`	Response data
`ypred`	Mean of predicted responses
`Vpred`	Variance of the predicted responses

Value

LL for the given data

`meanA` *Calculate mean of activated units*

Description

This function uses lineratization to estimate the activation units mean vector μ_A and the Jacobian matrix evaluated at μ_Z.

Usage

```
meanA(z, mz, funIdx)
```

Arguments

`z`	Vector of units for the current layer
`mz`	Mean vector of units for the current layer μ_Z
`funIdx`	Activation function index defined by `activationFunIndex`

Value

A list which contains the activation units mean vector μ_A and the Jacobian matrix evaluated at μ_Z

meanMz *Mean vector of units*

Description

This function calculate the mean vector of units μ_Z for a given layer.

Usage

```
meanMz(mp, ma, idxFmwa, idxFmwab)
```

Arguments

`mp`	Mean vector of parameters for the current layer
`ma`	Mean vector of activation units from previous layer
`idxFmwa`	Indices for weights and for activation units for the current and previous layers respectively
`idxFmwab`	Indices for biases of the current layer

Value

Mean vector of units for the current layer μ_Z

meanVar *Mean, Jacobian and variance of activated units*

Description

This function returns mean vector μ_A, Jacobian matrix evaluated at μ_Z and covariance matrix of activation units Σ_A.

Usage

```
meanVar(z, mz, Sz, funIdx)
```

Arguments

`z`	Vector of units for the current layer
`mz`	Mean vector of units for the current layer μ_Z
`Sz`	Covariance matrix of units for the current layer Σ_Z
`funIdx`	Activation function index defined by `activationFunIndex`

Value

- Mean vector of activation units for the current layer μ_A
- Covariance matrix activation units for the current layer Σ_A
- Jacobian matrix evaluated at μ_Z

`meanVarDev` *Mean and variance of activated units for derivatives*

Description

This function calculates mean vector and covariance matrix of activation units' derivatives.

Usage

```
meanVarDev(mz, Sz, funIdx, bound)
```

Arguments

`mz`	Mean vector of units for the current layer μ_Z
`Sz`	Covariance matrix of units for the current layer Σ_Z
`funIdx`	Activation function index defined by `activationFunIndex`
`bound`	If layer is bound

Value

- Mean vector of activation units' first derivative
- Covariance matrix of activation units' first derivative
- Mean vector activation units' second derivative
- Covariance matrix activation units' second derivative

`MedicalCost` *Medical Cost of 1,338 insureds.*

Description

A dataset containing the medical costs ("charges") and other attributes of 1,338 insureds.

Usage

```
MedicalCost
```

Format

A data frame with 1,338 rows and 10 variables:

age age of the insured
sex gender of the insured, binary (if female)
BMI Body Mass Index of the insured
children number of children covered as dependents
smoker smoking status, binary (if the insured smokes)

region: northeast	binary (if the insured lives in that region)
region: southeast	binary (if the insured lives in that region)
region: southwest	binary (if the insured lives in that region)
region: northwest	binary (if the insured lives in that region)
charges	medical costs, in US dollars

Details

The original dataset contains 7 variables, but one-hot encoding was used on the "region" categorical variable. It is a dataset that was used in a Kaggle competition.

Source

`https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/insurance.csv`

`network` *One iteration of the Tractable Approximate Gaussian Inference (TAGI)*

Description

This function goes through one learning iteration of the neural network model using TAGI.

Usage

```
network(NN, mp, Sp, x, y)
```

Arguments

`NN`	Lists the structure of the neural network
`mp`	Mean vector of parameters for each layer μ_θ
`Sp`	Covariance matrix of parameters for each layer Σ_θ
`x`	Input data
`y`	Response data

Value

- Updated mean vector of parameters for each layer μ_θ
- Updated covariance matrix of parameters for each layer Σ_θ
- Mean of predicted responses
- Variance of the predicted responses

`normalize` *Normalize data*

Description

This function normalizes data before entering the neural network.

Usage

```
normalize(xtrain, ytrain, xtest, ytest)
```

Arguments

`xtrain`	Training set of input variables
`ytrain`	Training set of responses
`xtest`	Testing set of input variables
`ytest`	Testing set of responses

Value

- Normalized training set of input variables
- Normalized training set of responses
- Normalized testing set of input variables
- Normalized testing set of responses
- Mean vector of input variables from training set
- Covariance matrix of input variables from training set
- Mean vector of responses from training set
- Covariance matrix of responses from training set

`parameterBackwardPass` *Backpropagation (parameters' deltas)*

Description

This function calculates parameter's deltas.

Usage

```
parameterBackwardPass(NN, theta, states, deltaM, deltaS)
```

Arguments

`NN`	Lists the structure of the neural network
`theta`	List of parameters
`states`	List of states
`deltaM`	Delta of mean vector of units given y $\mu_Z|y$ at all layers
`deltaS`	Delta of covariance matrix of units given y $\Sigma_Z|y$ at all layers

Value

Parameters' deltas (mean and covariance for each)

`parameters` *Indices for biases and weights*

Description

This function assigns indices for all weights and biases in the neural network.

Usage

```
parameters(NN)
```

Arguments

`NN`	List that contains the structure of the neural network

Details

Bias indices are assigned from 1 to the maximum number of biases for a given layer. Then, weight indices start where bias indices end plus one until all weights are assigned an indice. The number of weights for a given layer is the number of units in the previous layer times the number of units in the current one.

For example, if there are 10 units in the previous layer and 50 in the current one, then there would be 50 biases and 500 weights in the current layer. The bias indices would be from 1 to 50 and weight IDs from 51 to 550.

Value

NN with three new elements, each of size (number of layers -1) :

- Weight indices for each layer
- Bias indices for each layer
- Combined weight and bias indices for each layer

`regression` *Regression problem*

Description

This function trains neural network models to solve a regression problem.

Usage

```
regression(NN, x, y, trainIdx, testIdx)
```

Arguments

`NN`	Lists the structure of the neural network
`x`	Input data
`y`	Response data
`trainIdx`	Observations IDs that are assigned to the training set
`testIdx`	Observations IDs that are assigned to the testing set

Value

- Mean vector of parameters for each layer μ_θ
- Covariance matrix of parameters for each layer Σ_θ
- RMSE and LL metrics for each network models created
- Training time of each neural network models created
- Mean of predicted responses
- Variance of the predicted responses

`runBatchDerivative` *Result of the TAGI with derivative calculations*

Description

This function returns the resulting derivatives from the neural network model using TAGI.

Usage

```
runBatchDerivative(NN, xtrain, ytrain, xtest, ytest)
```

Arguments

`NN`	Lists the structure of the neural network
`xtrain`	Training set of input variables
`ytrain`	Training set of responses
`xtest`	Testing set of input variables
`ytest`	Testing set of responses

Value

- Mean of predicted responses
- Variance of the predicted responses
- Mean of first derivative of predicted responses
- Mean of second derivative of predicted responses

`split` *Split data*

Description

This function splits data into training and test sets.

Usage

```
split(x, y, ratio)
```

Arguments

`x`	Input data
`y`	Response data
`ratio`	Training ratio

Value

- Training set of input variables
- Training set of responses
- Testing set of input variables
- Testing set of responses

`ToyExample.x_obs` *Inputs used in training part for 1D toy problem*

Description

The orignal dataset represents a 1D regression problem from Hernández-Lobato & Adams (2015): $y = x^3 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 9)$ and $x \in [-4, 4]$. In this dataset, x and y are normalized.

Usage

```
ToyExample.x_obs
```

Format

A data frame with 20 rows and 1 variable x

Details

The dataset generated with the seed from the TAGI repository was used for comparison purposes.

Source

`https://github.com/CivML-PolyMtl/TAGI/blob/master/ToyExample/ToyExample_1D.m`

References

Hernández-Lobato, J. M., & Adams, R. "Probabilistic backpropagation for scalable learning of bayesian neural networks." International Conference on Machine Learning. 2015.

`ToyExample.x_val` *Inputs used in validation part for 1D toy problem*

Description

The orignal dataset represents a 1D regression problem from Hernández-Lobato & Adams (2015): $y = x^3 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 9)$ and $x \in [-4, 4]$. In this dataset, x and y are normalized.

Usage

```
ToyExample.x_val
```

Format

A data frame with 20 rows and 1 variable x

Details

The dataset generated with the seed from the TAGI repository was used for comparison purposes.

Source

https://github.com/CivML-PolyMtl/TAGI/blob/master/ToyExample/ToyExample_1D.m

References

Hernández-Lobato, J. M., & Adams, R. "Probabilistic backpropagation for scalable learning of bayesian neural networks." International Conference on Machine Learning. 2015.

`ToyExample.y_obs` *Responses used in training part for 1D toy problem*

Description

The orignal dataset represents a 1D regression problem from Hernández-Lobato & Adams (2015): $y = x^3 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 9)$ and $x \in [-4, 4]$. In this dataset, x and y are normalized.

Usage

```
ToyExample.y_obs
```

Format

A data frame with 20 rows and 1 variable y

Details

The dataset generated with the seed from the TAGI repository was used for comparison purposes.

Source

https://github.com/CivML-PolyMtl/TAGI/blob/master/ToyExample/ToyExample_1D.m

References

Hernández-Lobato, J. M., & Adams, R. "Probabilistic backpropagation for scalable learning of bayesian neural networks." International Conference on Machine Learning. 2015.

`ToyExample.y_val` *Responses used in validation part for 1D toy problem*

Description

The orignal dataset represents a 1D regression problem from Hernández-Lobato & Adams (2015): $y = x^3 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 9)$ and $x \in [-4, 4]$. In this dataset, x and y are normalized.

Usage

```
ToyExample.y_val
```

Format

A data frame with 20 rows and 1 variable y

Details

The dataset generated with the seed from the TAGI repository was used for comparison purposes.

Source

`https://github.com/CivML-PolyMtl/TAGI/blob/master/ToyExample/ToyExample_1D.m`

References

Hernández-Lobato, J. M., & Adams, R. "Probabilistic backpropagation for scalable learning of bayesian neural networks." International Conference on Machine Learning. 2015.

Index