

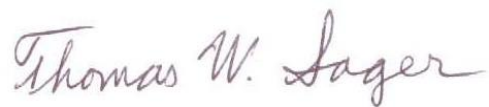
**Modelling Asset Risk, Product Risk and Capital-Asset Ratio in Insurance Industry Using
Feedforward Neural Networks and Recurrent Neural Networks**

Ashutosh Singh

May 5th, 2018

Dear Vicki,

I had the pleasure to advise Ashutosh Singh in his independent study project. His work on modelling risk using neural networks is good and insightful. He has utilized many neural network and statistical modelling techniques and his work is authentic to the best of my knowledge.

A handwritten signature in dark ink, reading "Thomas W. Sager". The script is cursive and fluid, with the first name "Thomas" and last name "Sager" clearly legible.

Thomas W. Sager

A handwritten signature in dark ink, appearing to be "Ashutosh". The signature is stylized and somewhat abstract, with a long horizontal stroke extending to the right.

Ashutosh Singh
As79592

Modelling Asset Risk, Product Risk and Capital-Asset Ratio in Insurance Industry Using Feedforward Neural Networks and Recurrent Neural Networks

Ashutosh Singh

May 6, 2018

Abstract

The research project investigates the application of Neural Networks in risk modeling. The work is based on the partial adjustment model given by Thomas W. Sager and Etti G. Baranoff[1]. The project explores two alternatives to the simultaneous-equation method used by Sager and Baranoff. The research was conducted in two parts. The first part of the research is based on feed-forward neural networks. I use a one-hidden-layer neural network to fit the variables used in the simultaneous-equation method, by Sager and Baranoff. To evaluate the importance of predictor variables, I use the percent change in MSE of reduced models relative to the full model, an approach similar to 'Type II Sum of Squares'[12]. In the second part of the research, I use a Long Short-Term Memory(LSTM) based recurrent neural network to fit the predictor variables. I analyze the capacity of LSTM units to fit time-series data by comparing performances of LSTM models with and without lagged predictors and assessing the importance of the lagged response variable for the former.

1 Introduction

In the paper[1], Sager and Baranoff model product risk, asset risk and capital-asset ratio using the partial-adjustment model in simultaneous-equation mode. They consider a 1st order autoregressive process, with three equations, one each for capital-asset ratio, product risk, and asset risk. Each of the target levels is a function of the exogenous parameters, concurrent values of the other two observable responses and the lagged value of the target level. Authors use an autoregressive two-stage least squares procedure to solve for the coefficients. The equations used are,

$$C_t = \beta_0^C + \beta_C^C C_{t-1} + \beta_A^C A_t + \beta_P^C P_t + \beta_1^C X_{1t} + \dots + \beta_K^C X_{Kt} + \varepsilon_t^C \quad (1)$$

$$A_t = \beta_0^A + \beta_C^A C_t + \beta_A^A A_{t-1} + \beta_P^A P_t + \beta_1^A X_{1t} + \dots + \beta_K^A X_{Kt} + \varepsilon_t^A \quad (2)$$

$$P_t = \beta_0^P + \beta_C^P C_t + \beta_A^P A_t + \beta_P^P P_{t-1} + \beta_1^P X_{1t} + \dots + \beta_K^P X_{Kt} + \varepsilon_t^P \quad (3)$$

Where C_t , A_t and P_t are the t^{th} time-period values of the response variables for capital-asset ratio, asset risk, and product risk, respectively. X_{it} is the t^{th} time-period value of the i^{th} exogenous predictor and β_i^j is the coefficient for the i^{th} predictors corresponding to the j^{th} response variable.

Neural networks have a number of advantages when compared to linear models. Jack Tu states that, "Neural networks offer a number of advantages, including requiring less formal statistical training, ability to implicitly detect complex nonlinear relationships between dependent and independent variables, ability to detect all possible interactions between predictor variables, and the availability of multiple training algorithms"[19]. Neural networks are not just convenient but can also perform better than linear models. According to Sovan and Marc, "The Multiple Regression principle of linear modeling often gives low performance when relationships between variables are nonlinear; this is often the case in ecology; some variables must therefore be transformed. Despite these manipulations, the results often remain disappointing: poor prediction, dependence of residuals on the variable to predict. On the other hand, Neural Networks are nonlinear type models. They do not

necessitate transformation of variables and can give better results. "[11]. Motivated by these advantages of neural networks, I propose two alternatives to the method presented by Sager and Baranoff.

As the first alternative, I propose three feed-forward neural network models based on the equations 1, 2, and 3. For each model, the predictor variables are the variables on the right side of the equation and the variable on the left is the response variable. One of the disadvantages of neural networks is that it is a "Black Box" algorithm[19]. Unlike linear models, it is difficult to estimate the importance of predictors, based on parameters. I propose to overcome this problem by using a method based on calculating the percent change in MSE of the reduced model, compared to the full model. The reduced model is created by removing the predictor variable whose importance is to be assessed. The percent change calculated using this method is proportional to the F statistic, used for estimating predictor importance using an F test.

$$\text{deltaMSE} = \frac{MSE(F) - MSE(R)}{MSE(F)} = \frac{\frac{MSE(R) - MSE(F)}{df(F) - df(R)}}{\frac{MSE(F) \times df(F)}{df(F)}} = \frac{FStatistic}{df(F)} \quad (4)$$

$$\text{PercentChangeinMSE} = \frac{FStatistic}{df(F)} \times 100 \quad (5)$$

Where $MSE(F)$ is the MSE of the full model, $df(F)$ is the degrees of freedom of the full model, $MSE(R)$ is the MSE of the reduced model and $df(R)$ is the degrees of freedom of the reduced model. $df(F) - df(R) = 1$, since the reduced model has one less predictor variable when compared to the full model. Since for neural networks we use batch gradient descent, the batch size is the number of data-points(n). If $n \gg p$, where p is the number of predictor variables, we can write $df(F) \approx n = \text{batchsize}$. Using the discussed approximation, equation 5 can be written as,

$$\text{PercentChangeinMSE} = \frac{FStatistic}{batchsize} \times 100 \quad (6)$$

The reason to not use the F test is to be able to understand the relative importance of predictor variables. Since my primary focus is to achieve a good fit to the training data set, I use a large batch size to achieve a sharp minimum[15]. The large batch size makes the P values of the relevant predictors very small, hence eliminating any possibility for inter-variable comparison.

As the second alternative, I propose three recurrent neural network models, based on LSTM cells. I propose two-versions of the LSTM models. The first version uses the predictor variables used by the feed-forward models and the second version excludes lagged response variables from the predictor variables. To observe the capacity of recurrent neural networks to model time-series variables[10], I compare the performance of the two-versions. I also use the percent change in MSE to evaluate the importance of the predictor variables. I compare the importance of the lagged-response predictors for feed-forward and recurrent neural network models, to see if LSTM models have a reduced importance for lagged response predictors. Reduction in importance is expected as LSTM units take a sequence as input and are capable of accommodating the time-dependence, based on the input sequence, without the lagged predictors (Unsupervised learning)[10].

2 Literature

Neural networks are at the forefront of Machine Learning and Artificial Intelligence research. Neural networks have a lot of variations with new ones being added regularly. In this section, I provide a basic overview of feed-forward neural networks and recurrent neural networks. The text is introductory at best, the reader should consult specific sources for a better understanding of the concepts discussed in the following subsections[2][6] [22].

2.1 Feed Forward Neural Networks

The most practical way to visualize a feed-forward neural network is to visualize it using layers and nodes. A layer is a collection of nodes, and nodes are units that perform mathematical operations.

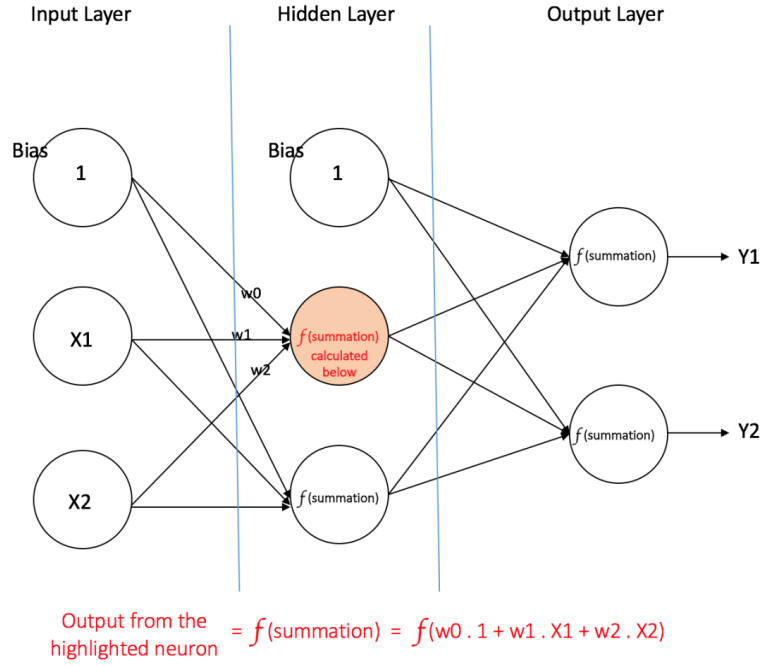


Figure 1: Layers in a feed-forward neural network[20]

The first layer of a feed-forward neural network is the input layer. The input layer consists of the predictors[2]. The number of input nodes is equal to the number of predictors. For the neural network in the Figure 1, the number of input nodes is 2. We rarely feed a vector (single data point) to a neural network. Usually, a batch of data-points is fed to the input layer, which is a matrix of shape $batchsize \times numberofpredictors$.

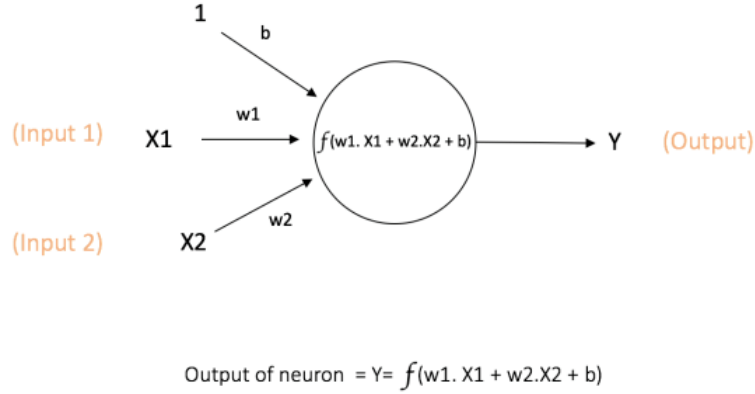


Figure 2: A node in a hidden layer[20]

The next layer is called a hidden layer. It is named so as it does not interact with the input or the output and, in a spatial sense, is hidden between the input and the output layers. The number of nodes in a hidden layer is a hyper-parameter and is determined by the designer. Values from all the data nodes are passed to all the nodes in the hidden layer. Each node has specific weights for every data node. Weights are multiplied by the inputs from the data nodes and are added together with a bias value. An activation function is applied to the outcome of the linear operation. The final output after the activation function acts as the output from the node. The mathematical operation at a node is schematically represented in Figure 3. The bias values are equivalent to the error terms in linear models. Bias is added to avoid over-fitting and provide flexibility during training. Bias provides wiggle room. The activation function is used to introduce non-linearity in the network. Many activation functions are used in practice with the most famous ones being

ReLU, tanh and Sigmoid.

$$y = \text{Sigmoid}(x) = \frac{1}{(1 + e^{-x})} \quad (7)$$

$$y = \tanh(x) \quad (8)$$

$$y = \text{ReLU}(x) = \max(0, x) \quad (9)$$

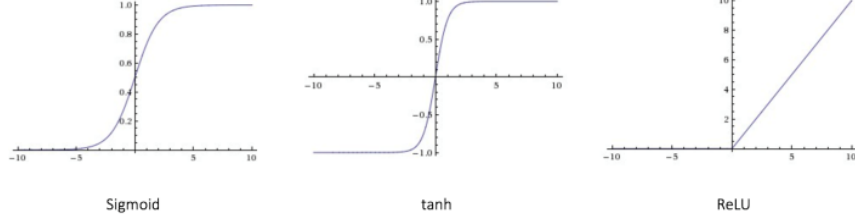


Figure 3: Activation Functions[20]

Multiple hidden layers can be used in sequence. In such a case, a previous layer in the sequence acts as the input layer for the subsequent layer. The output from each node of the preceding hidden layer is passed to every node of the next layer.

The output layer is responsible for producing the response variable and thus, the structure of the output layer depends on the type of the response variable. For categorical response variables, Softmax layer is used. It is called 'Softmax' because it uses the softmax/normalized exponential function. Since the pertinent response variables are continuous, I will not discuss the Softmax layer. Curious readers can consult specific sources[25]. For continuous response variables, the output layer is a linear operation, similar to the linear operations from the hidden layer. The output from all the nodes of the final hidden layer is passed to the output layer. The number of nodes in the output layer is one. The inputs are multiplied by their respective weights and added together with a bias value to produce the final output from the feed forward network.

Mathematically, the feed-forward network could be written as,

$$X_{n0 \times \text{numberpredictors}}^0 \quad (10)$$

$$X_{n0 \times n1}^1 = f((X_{n0 \times \text{numberpredictors}}^0 \times W_{\text{numberpredictors} \times n1}^1) + B_{n1 \times 1}^1) \quad (11)$$

$$X_{n0 \times n2}^2 = f((X_{n0 \times n1}^1 \times W_{n1 \times n2}^2) + B_{n2 \times 1}^2) \quad (12)$$

$$X_{n0 \times n3}^3 = f((X_{n0 \times n2}^2 \times W_{n2 \times n3}^3) + B_{n3 \times 1}^3) \quad (13)$$

.....

$$X_{n0 \times nm}^m = f((X_{n0 \times nm-1}^{m-1} \times W_{nm-1 \times nm}^m) + B_{nm \times 1}^m) \quad (14)$$

$$Y_{n0 \times 1} = (X_{n0 \times nm}^m \times W_{nm \times 1}^y) + B_{1 \times 1}^y \quad (15)$$

Where X^0 is the input matrix, X^i is the output from the i^{th} hidden layer, W^i is the weight matrix for the i^{th} layer, B^i is the bias matrix for the i^{th} layer, Y is the output from the network, m is the total number hidden layer, f is the activation function and the subscript for each matrix is the shape of the matrix.

The output from a neural network is an estimate of the response variable and is a function of the weights and the biases. A loss function is used to estimate the accuracy of the output. The loss function is used to train the model. The training process is essentially a search for the weights and biases that result in a desired level of accuracy. Cross-Entropy with Logits[21] and Mean Squared Error(MSE) are used for categorical and continuous response variables, respectively. Mean squared error is the mean of the square of difference of the estimated values and the observed values,

$$MSE = \frac{\sum (Y_i - Y'_i)^2}{n} \quad (16)$$

where Y_i s are the target/observed values, Y_i' s are the estimated values, and n is the number of data-points.

At the beginning of the training process, the weights and the biases are initialized randomly. To attain optimal values for the weights and the biases, gradient-based iterative optimization algorithms are used. A variety of optimization algorithms are used in practice and their discussion is out of the scope of this paper. Interested readers should consult the cited resource for more information on different optimization algorithms[18]. To develop some background for back-propagation, I will discuss the functional aspects of Gradient Descent. Gradient Descent is an optimization algorithm which is used to solve problems that do not have an analytical solution[7]. It is an iterative algorithm and can be mathematically written as,

$$X_{n+1} = X_n - \gamma \times \nabla(X_n) \quad (17)$$

where X_i is the values of the matrix at the i^{th} step, γ is the step size and $\nabla(X_i)$ is the derivative of the matrix at the i^{th} step. Geometrically, it is a process of moving in the opposite direction of the gradient[7]. Step size determines the speed of the descent. The reader can learn about the variations of Gradient Descent from the cited resource[18]. The stated description of gradient descent is easy and intuitive to apply to stand-alone quantities, but cannot be directly applied to the weights and the biases matrices of neural networks. The output from a layer of a network is a function of the output from the previous layer, which in turn is a function of the output from the layer before it. Mathematically, neural network is a composite function and thus the gradient at any layer is related to the gradients of the later layers, by the Chain Rule. To account for this constraint, gradients are calculated in a reverse sequence. The algorithm used to accomplish this task is called back-propagation.

The gradient for any weights or biases matrix is the derivative of the loss with respect to the matrix. The gradient is initialized at the final step and mathematically is the rate of change of loss with respect to the output from the final layer. This gradient is propagated backward through the network using the following algorithm[13]:

$$\delta^N = (z_i^N - y_i) = \frac{\partial L}{\partial z^N} \quad (18)$$

$$\delta^n = ((W^{n+1})^T \delta^{n+1}) \odot f'(z^n) \quad (19)$$

$$\frac{\partial L}{\partial B^n} = \delta^n \quad (20)$$

$$\frac{\partial L}{\partial W^n} = X^{n-1} \delta^n \quad (21)$$

where L is the loss(MSE), W^i and B^i are the weights and the biases matrices, for the i^{th} layer, respectively. N is the total number of layers, f and f' are the activation function and its derivative. $z^n = W^n X^{n-1} + B^n$, and X^n is the output from the n^{th} layer such that, $X^n = f(z^n)$. δ^n is an intermediate quantity, known as the error for the n^{th} layer[13]. \odot is a notation for element wise product, also known as the Hadamard product. The gradients calculated in equation 20 and equation 21 are used to modify the weights and the biases for the n^{th} layer, according to equation 17. The algorithm discussed using equation 18, 19, 20 and 21 is for one iteration. This process is repeated for multiple iterations to calculate the optimal values of weights and biases.

2.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) are "a class of artificial neural network where connections between units form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence"[24]. In case of a time-sequence data, each unit is fed data pertaining to a particular time-period. Each unit receives information in the form of the cell state from the previous unit. The pertinent unit uses the cell state and the input to calculate the output and modifies the cell state. The modified cell state is passed to the next unit and the output is passed

to the output layer. The output from the output layer is the final output for the particular time step. The general operational structure of an RNN could be represented as,

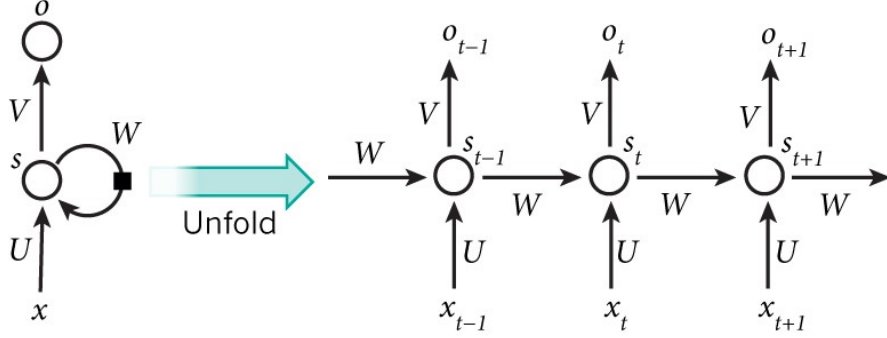


Figure 4: Recurrent Neural Network[3]

where x_i is the input, o_i is the output, and s_i is the cell state for the i^{th} time period.

Long-Short Term Memory (LSTM) units and the Gated Recurrent Units (GRUs) are the two kinds of units that are used to make recurrent neural networks. A gradient based loss minimization technique is used to train the network. Loss functions and optimization techniques similar to the feed-forward networks are used. In the research, I have used LSTM units and will discuss their structure. Any discussion on GRUs is out of the scope of this paper. I will also not discuss the Optimization technique as this a long and complex process and will require a paper of its own. To develop a better understanding, I highly encourage the reader to consult specific material on the topic[3] [9][8][16].

A LSTM unit is composed of a cell and three gates. Each gate can be considered equivalent to a hidden layer in the feed forward network. The output from each gate is activation applied to a linear operation. The gates have specific functions. The forget (so-named because it removes information, thus figuratively “forgetting” it) and input gates decide the information to be removed and added, to generate the next cell, based on the current input, last output, and the current state of the cell. The estimated output from a cell is calculated using the output gate and the current state of the cell[23]. An LSTM unit can be mathematically written as,

$$I_t = \sigma(W_i(X_t, h_{t-1}) + B_i) \quad (22)$$

$$F_t = \sigma(W_f(X_t, h_{t-1}) + B_f) \quad (23)$$

$$O_t = \sigma(W_o(X_t, h_{t-1}) + B_o) \quad (24)$$

$$h_t = O_t \odot \tanh(C_t) \quad (25)$$

$$C'_t = \tanh(W_u(X_t, h_{t-1})) \quad (26)$$

$$C_{t+1} = F_t \odot C_t + I_t \odot C'_t \quad (27)$$

where I_t , F_t , and O_t are input, forget and output gates for the t^{th} time-period, respectively. C_t and h_t are the cell state and the output for the t^{th} time-period. C'_t is an intermediate quantity known as the update value. It is the information that is added to cell using the update gate. W s and B s are the weights and the biases matrices with the subscripts i , u and o pertaining to input, update and output gates, respectively. σ and \tanh are the sigmoid and the hyperbolic tangent activation functions. h_t is passed to the output layer, to calculate the output for the t^{th} time-period. The output from the output layer is the estimate of the response variable for the t^{th} time period and is used to calculate the loss and initiate the gradient for back-propagation. An LSTM cell can be schematically represented by the figure 5

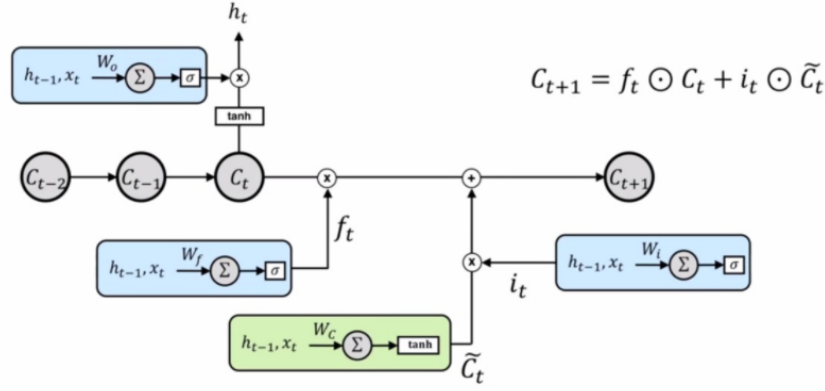


Figure 5: An LSTM Cell

3 Data

The dataset was obtained from the NAIC database of insurers' annual statements from 1993-1997. It has a total of 35904 data-points with 250 variables. The dataset is a longitudinal dataset. For most of the companies, the dataset contains multiple entries, corresponding to different years. Each company has a unique key, mentioned in the 'CODE' column

3.1 Variable Selection

I select the variables that were used by Sager and Baranoff in their work[1]. My selection is justified by the argument of 'Expert Opinion'.

- ProdHRRisk: It is the measure of product risk, It the ratio of health writing to the total writing.
- Atotal: Total assets of the company.
- pRegARisk: It is the measure of regulatory asset risk. It is used to approximate the asset risk.
- Capital-asset ratio: It is the ratio of the total capital to the total asset of the insurer.
- Ntype: It is an indicator for governance structure, with 1 indicating a stock company and 0 indicating other types – typically mutual (policyholder-owned).
- Ngroup: An indicator for whether the insurer is a member of a group of affiliated companies.
- RetOnCap: ratio of total income to capital.
- LogATotal: Logarithm of total asset.
- RBCratio: A measure of capital coverage of investment risks – the smaller this measure, the more exposed the company to regulatory intervention.
- year95: A binary indicator that captures general industry specific changes in 1995, relevant to the study, is 1 for data from 1995, and is 0 for other years.
- year96: A binary indicator that captures general industry specific changes in 1996, relevant to the study, is 1 for data from 1996, and is 0 for other years.
- year97: A binary indicator that captures general industry specific changes in 1997, relevant to the study, is 1 for data from 1997, and is 0 for other years.
- lagProdHRRisk: Lagged ProdHRRisk, lagged by a time period of 1 year.
- lagpRegARisk: Lagged RegARisk, lagged by a time period of 1 year.
- lagCAratio: Lagged Capital-Asset ratio, lagged by a time period of 1 year.

	Quantity	Variable	Mean	SD	Median
1	Product Risk	ProdHRisk	0.27332	0.3722	0.0556
2	Total Assets	Atotal	2.96E+09	9697.7	91804.9
3	Asset Risk	pRegARisk	2.58E-02	0.0411	0.0107
4	Capital-Asset Ratio	Rtotcap/Atotal	0.35648	0.2695	0.2247
5	Governance Structure	Ntype	0.9146	0.2966	1
6	Member of a Group	Ngroup	0.7157	0.4401	0
7	Return on Capital	RetOnCap	-0.0204	0.0424	1.9576
8	Logarithm of Asset	LogATotal	18.35	2.5199	18.3352
9	Regulatory Pressure	RBCratio	5574	87320	371

Table 1: Summary Statistics

3.2 Data Cleaning

The dataset had missing values and outliers. I removed the rows with missing values for any of the selected fifteen predictor variables. I also removed rows with extreme outliers. The dataset was reduced to 17500 rows from the initial 35904 rows.

3.3 Data Visualization

The cleaned data has the statistics shown in Table 1,

I use a correlation plot for visual inspection of the correlations between variables. It is clear

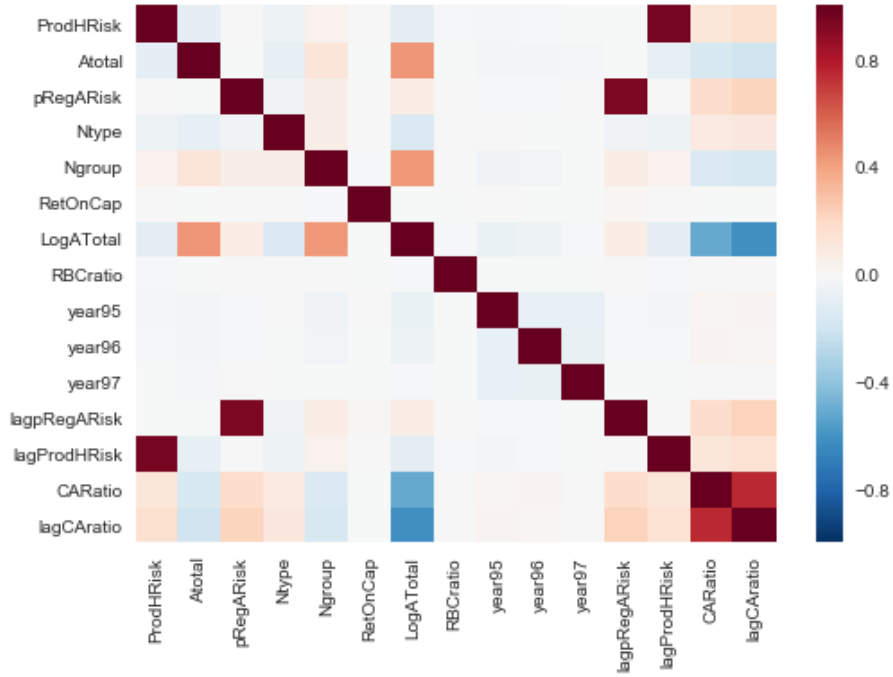


Figure 6: Correlation between variables[]

from Figure 6 that the predictors for product risk, asset risk, and capital-asset ratio have high correlation with their respective lagged predictors. The correlation Table 2 shows the values for these correlations. Asset risk and product risk have correlations higher than 0.9 with their respective lagged predictors. This corroborates the assumption of time-series behavior of these responses. Regardless of the high correlations, I continue with the modeling exercise without any remedial action as neural networks are adept at dealing with multicollinearity[4].

	ProdHRisk	pRegARisk	CARatio
lagProdHRisk	0.960088	0.002596	0.121839
lagpRegARisk	0.000021	0.941193	0.180168
lagCARatio	0.158134	0.222392	0.750428

Table 2: Correlations between Response and lagged response variables

3.4 Data Preprocessing

Feed-forward neural networks and recurrent neural networks take input in different formats. I used Python and Numpy to write functions that feed data in required formats[17]. Feed-forward neural networks take input in the form of a matrix. The rows of the matrix correspond to data-points and the columns correspond to predictor variables. The number of rows is equal to the batch size being used for the optimization process. The batch size is a hyper parameter. The dataset is separated into predictor variables (Xs) and the response variable (Y). The response variable is either product risk, asset risk or capital-asset ratio, depending on the model. The remaining fourteen variables are used as predictor variables. Predictor variables are scaled using the pre-processing package from the scikit-learn library. The response variable and the predictor variables are combined together. The dataset is randomly divided in a ratio of 1:1:4 into validation, test, and training datasets, respectively. For each iteration of the optimization process, the feed function randomly selects a subset, of the specified batch size, from the training dataset, and feeds it to the neural network.

RNNs take tensors as input. A tensor is a sequence of matrices and each matrix is similar to the matrices fed to the feed-forward network. The matrix sequence is time based. If the first matrix contains data for fifty companies for the first time-period ($t = 1$), then the second matrix contains data for the same companies, arranged in the same order, for the second time-period ($t = 2$). The number of matrices in the tensor is equal to the number of maximum time-steps in the network. The companies in the dataset have entries for different number of years. The number of years vary from two to eighteen. To make uniform matrices, I pad the data by adding data-points that have zeros for all the predictors and the response variable. Consider an example, if I want to make the tensor using companies A and B that have data available for 3 and 5 years, respectively. I will make a tensor with 5 matrices having two rows each. For the last two matrices, rows corresponding to A will have zeros for all the predictor variables and also for the response variable. Padding doesn't impact the training process as the gradient is zero for the padded rows. I wrote a feed function[17] that randomly selects companies using the 'CODE' variable. The number of companies selected is equal to the batch size and is a hyper parameter. All the entries for the selected companies are scaled and arranged in matrices, according to their time-steps. The number of rows is equal to the batch size and zeros are added for companies that have data for less than eighteen time-steps. The tensor is fed to the optimization process .

4 Model

In this section, I discuss the specific details of the models. Neural networks need a fair amount of trial-based decisions to decide the hyper-parameters. The final parameters have been decided by testing manually a range of possible combinations. I recommend to the readers, who are unfamiliar with such an approach, to go through this section with an open mind.

4.1 Feed-Forward Neural Network

The hyper-parameters to be determined for a feed forward neural network include the number of layers, size of each layer, activation function, optimizer and the batch size. In my model,

- I use a one hidden layer network with 32 nodes. Increasing the number of nodes or the number of layers results in a bigger MSE value.
- I use a large batch size of 14000 to achieve a sharp minimum.
- I use Adagrad Optimizer because it is the only optimizer that results in a converging loss. Other optimizers such as Gradient Descent give a non-converging and sometimes an increasing loss trend.

- I use a learning rate of 0.5 for no particular reason. A range of learning rates have optimal performance. Learning rate is equivalent to γ in equation 17.
- I run the optimization process for 10000 iterations

I use the above architecture to model product risk, asset risk and capital-asset ratio in three different models.

4.2 Recurrent Neural Network

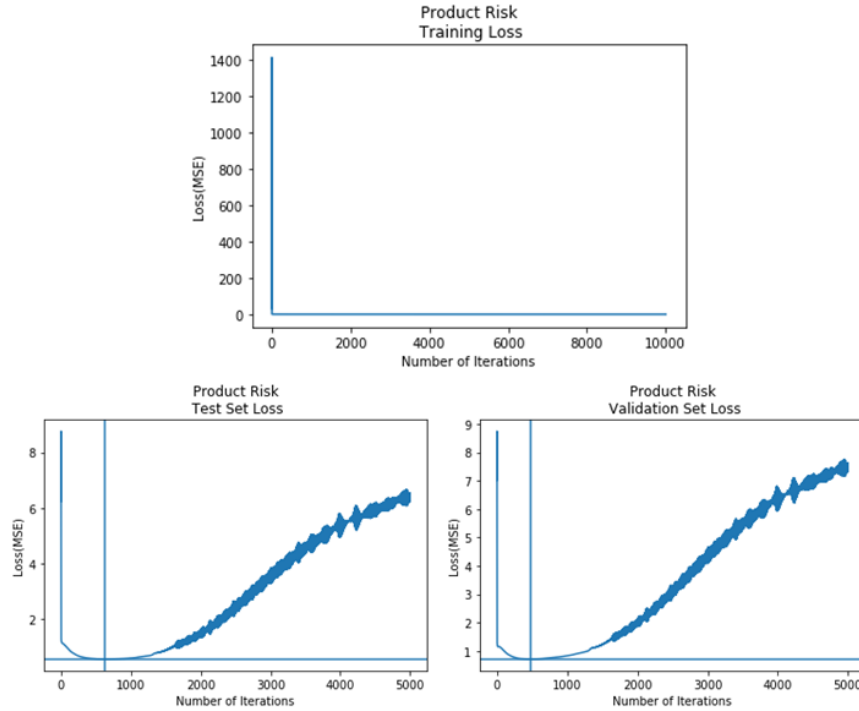
I use an LSTM cell with 32 nodes and a batch size of 150. An increase in batch size causes no additional reduction in MSE but increases the training time considerably. For the optimization process of 10000 iterations, I use Adagrad optimizer with a learning rate of 0.5. I use the same parameters for both versions of the three LSTM models.

5 Results

In this section, I present the results of the modeling exercise. A comprehensive interpretation of the results reported in this section, is presented in the next section.

5.1 Feed-Forward Network

During the optimization process, the loss (Mean Squared Error) for the product risk model, in a few hundred iterations, converges to a value of 8×10^{-3} . An extended run for 10000 thousand iterations results in little change in the training loss. Minimum MSE of 0.0077924528 is obtained for the training dataset.



[H]

Figure 7: Loss Curves: Product Risk Feedforward Model

Test and validation losses behave identically. They decrease to a minimum value and then start to increase. The increase is caused by over fitting the model on the training dataset. The minimum MSEs achieved on the test and validation sets are 0.55793589 and 0.71244967, respectively.

Description	Minimum Loss(MSE)	Number of Iterations
Linear Model (Reference)	0.370526	-
Training Loss	0.0077924528	100000
Test Loss	0.55793589	630
Validation Loss	0.71244967	472

Table 3: Loss comparison: Product Risk Feedforward Model

The percent change in MSE is the largest for the reduced¹ model without the lagged predictor of product risk.

Predictor	Reduce Model MSE	Full Model MSE	Percent Change in MSE
lagProdHRisk	0.107844661	0.007792453	1283.962955
RBCratio	0.012480779	0.007792453	60.16496115
year95	0.011791273	0.007792453	51.31657904
RetOnCap	0.010685591	0.007792453	37.12743951
Ntype	0.010150982	0.007792453	30.26683973
Atotal	0.009942177	0.007792453	27.58725725
CARatio	0.009844301	0.007792453	26.33122783
lagpRegARisk	0.009480093	0.007792453	21.65736442
LogATotal	0.009445637	0.007792453	21.21519042
year97	0.008958207	0.007792453	14.96003543
year96	0.008550015	0.007792453	9.721746406
lagCARatio	0.008155325	0.007792453	4.656646749
pRegARisk	0.008145252	0.007792453	4.527449945
Ngroup	0.007833122	0.007792453	0.521884457

Table 4: Importance of Predictors: Product Risk Feedforward Model

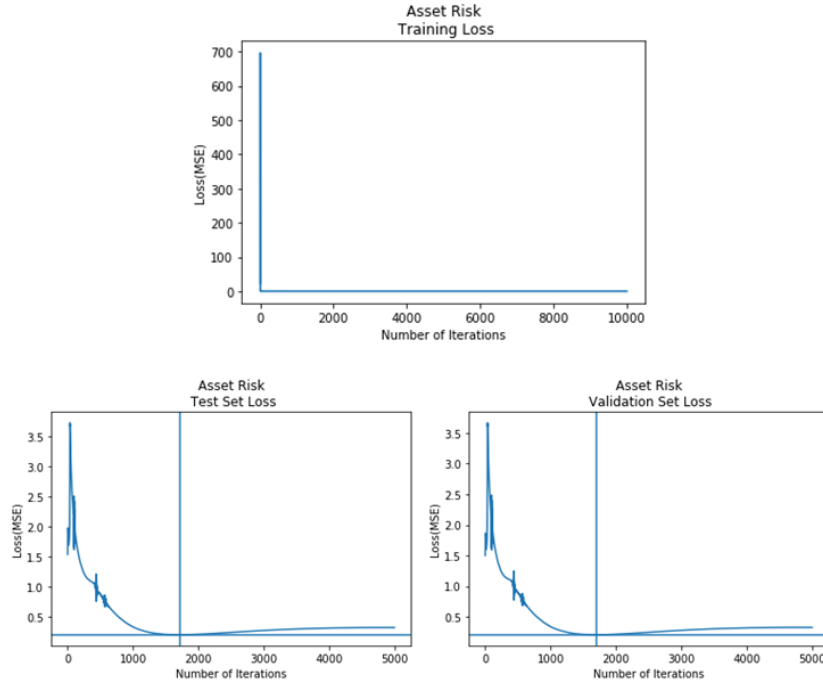
For the asset risk model, the loss (Mean Squared Error) converges to a range of 4×10^{-4} , in a few hundred iterations. An extended run for 10000 thousand iterations results in little change in the training loss. A minimum MSE of 0.00035161469 is obtained for the training dataset. The percent change in MSE is largest for the reduced model without the lagged predictor of Asset Risk, followed by models without the predictors for Return on Capital and Total Asset, respectively.

Description	Minimum Loss(MSE)	Number of Iterations
Linear Model (Reference)	0.0409973	-
Training Loss	0.00035161469	100000
Test Loss	0.19583209	1713
Validation Loss	0.20566596	1703

Table 5: Loss Comparison: Asset Risk Feedforward Model

Test and Validation losses behave identically. They decrease to a minimum value and then start to increase. The increase is caused by over fitting the model on the training dataset. The minimum MSEs achieved on the test and the validation datasets are 0.19583209 and 0.20566596, respectively.

¹reduced model excludes the predictor whose importance is being assessed.



[H]

Figure 8: Loss Curves: Asset Risk Feedforward Model

Predictor	Reduce Model MSE	Full Model MSE	Percent Change in MSE
lagpRegARisk	0.002460134	0.000351615	599.6674968
RetOnCap	0.001437996	0.000351615	308.969375
Atotal	0.001365024	0.000351615	288.2158621
RBCratio	0.000862782	0.000351615	145.3769892
year96	0.000769441	0.000351615	118.830769
lagProdHRisk	0.000599341	0.000351615	70.45403023
Ngroup	0.000577697	0.000351615	64.29832895
year97	0.000529086	0.000351615	50.47313296
year95	0.000526865	0.000351615	49.84154957
CARatio	0.000486126	0.000351615	38.25541248
LogATotal	0.000430181	0.000351615	22.3443793
lagCARatio	0.000421553	0.000351615	19.89057966
Ntype	0.000381949	0.000351615	8.627278911
ProdHRisk	0.000357414	0.000351615	1.649330976

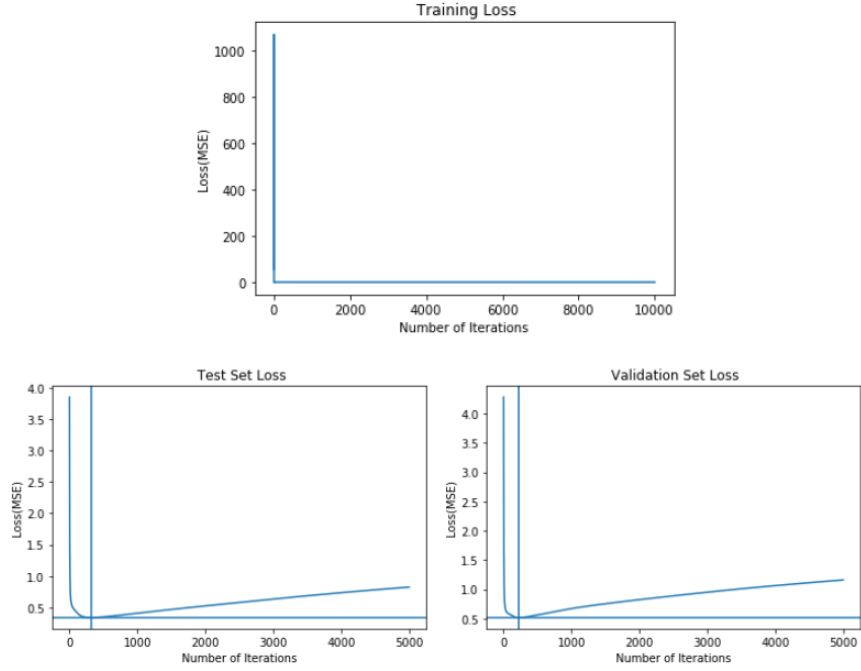
Table 6: Importance of Predictors: Asset Risk Feedforward Model

For the capital-asset ratio model, the loss (Mean Squared Error) converges to a range of 7×10^{-2} , in a few hundred iterations. An extended run for 10000 thousand iterations results in little change in the training loss. The minimum MSE of 0.061007324 is achieved for the training dataset. The percent change in MSE is the largest for the reduced model without the lagged predictor of capital-asset ratio.

Description	Minimum Loss(MSE)	Number of Iterations
Linear Model (Reference)	0.2132114	-
Training Loss	0.061007324	100000
Test Loss	0.34082574	326
Validation Loss	0.51742101	229

Table 7: Loss comparison-Capital-Asset ratio feed-forward model

Test and Validation losses behave identically. They decrease to a minimum value and then



[H]

Figure 9: Loss Curve: Capital-Asset Ratio Feedforward Model

start to increase. The increase is caused by over fitting the model on the training dataset. The minimum MSEs achieved on the test and the validation datasets are 0.34082574 and 0.51742101, respectively

Predictor	Reduce Model MSE	Full Model MSE	Percent Change in MSE
lagCARatio	0.08860933	0.061007324	45.24375795
lagpRegARisk	0.062851518	0.061007324	3.022905906
year96	0.062627532	0.061007324	2.655759823
Ntype	0.062517002	0.061007324	2.474584855
LogATotal	0.062434126	0.061007324	2.33873887
RBCratio	0.062116772	0.061007324	1.818548868
RetOnCap	0.062105391	0.061007324	1.799893731
ProdHRisk	0.062054645	0.061007324	1.716713554
Ngroup	0.062030938	0.061007324	1.677854285
Atotal	0.061848912	0.061007324	1.37948683
year97	0.061628871	0.061007324	1.018807185
pRegARisk	0.061586052	0.061007324	0.94862053
lagProdHRisk	0.061460014	0.061007324	0.742025662
year95	0.061017387	0.061007324	0.016494741

Table 8: Importance of Predictors: Capital-Asset Ratio Feedforward Model

5.2 Recurrent Neural Network Model

The losses for the Product risk models with and without the lagged predictor, converges to similar values. A minimum loss of 0.052094657 and 0.062157836 is achieved in 8851 and 9396 iterations, for models with and without the lagged predictor, respectively.

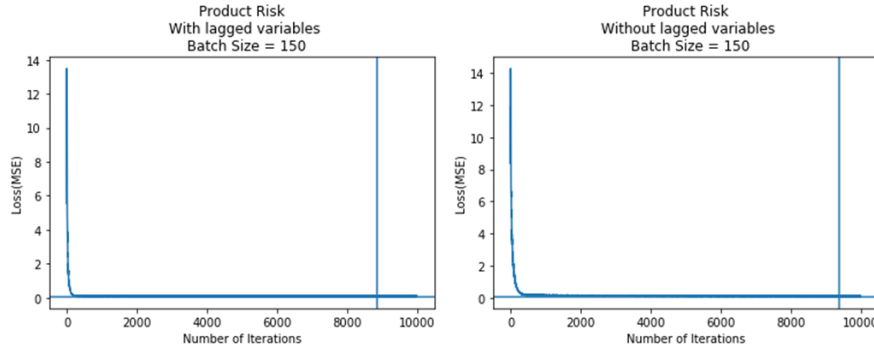


Figure 10: Loss Comparison of Product Risk models including and excluding lagged predictors

The losses for Asset risk models with and without the lagged predictor, converges to similar values. A minimum loss of 0.001920539 and 0.002519771 is achieved in 9892 and 9737 iterations, for models with and without the lagged predictor, respectively.

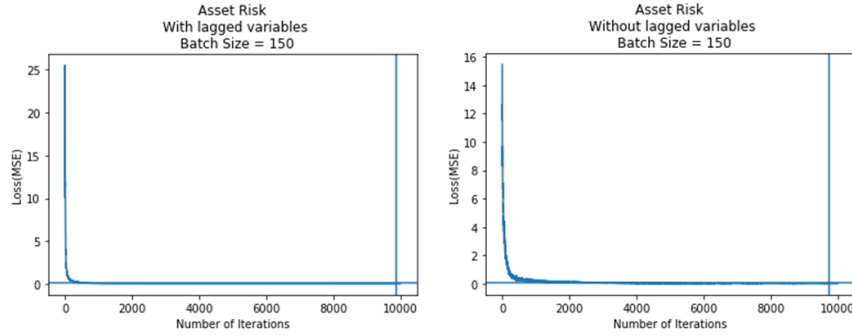


Figure 11: Loss Comparison of Asset Risk models including and excluding lagged predictors

The losses for Capital-Asset Ratio models with and without the lagged predictor, converges to similar values. A minimum loss of 0.058613993 and 0.061472513 is achieved in 9208 and 7680 iterations, for models with and without the lagged predictor, respectively.

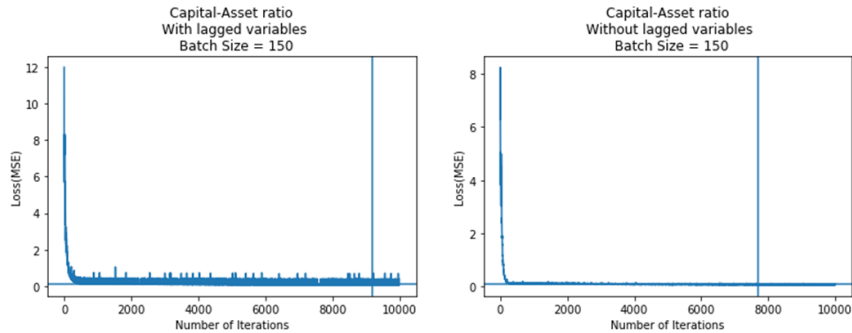


Figure 12: Loss Comparison of Capital-Asset Ratio models including and excluding lagged predictors

Model	With Lagged Predictors		Without Lagged Predictors	
	Minimum Loss	Iterations	Minimum Loss	Iterations
Product Risk	0.052094657	8851	0.062157836	9396
Asset Risk	0.001920539	9892	0.002519771	9737
Capital-Asset Ratio	0.058613993	9208	0.061472513	7680

Table 9: Summary of training loss for models with and without lagged predictors

Recurrent neural networks are infamous for high computational requirements. 10000 iterations for a single model takes 71 minutes on my hardware. Due to hardware limitations, I could only test the predictor importance for one model. I choose the Asset risk model as it has 4 important predictors for the feed-forward model. This provides an opportunity to observe the change in the importance of the lagged response variable, compared to other important predictors.

Predictor	Reduce Model MSE	Full Model MSE	Percent Change in MSE
RetOnCap	0.004196823	0.001920539	118.5231668
RBCratio	0.003955975	0.001920539	105.9825592
Atotal	0.003681707	0.001920539	91.70176289
lagpRegARisk	0.003238688	0.001920539	68.63430561
Ngroup	0.002834274	0.001920539	47.5770309
Ntype	0.002265493	0.001920539	17.96130773
ProdHRisk	0.002248705	0.001920539	17.08720265
lagCARatio	0.002217888	0.001920539	15.4825755
year95	0.002169678	0.001920539	12.97236671
year97	0.002151802	0.001920539	12.04154785
year96	0.002016781	0.001920539	5.0111993
lagProdHRisk	0.001983566	0.001920539	3.281741598
LogATotal	0.001978485	0.001920539	3.017165559
CARatio	0.001975269	0.001920539	2.849704831

Table 10: Importance of Predictors: LSTM based Capital-Asset Ratio Model

The lagged predictor for the asset risk is the fourth most important predictor in the LSTM asset risk model. There is a significant reduction in the importance and the percentage change of MSE, compared to the feed-forward asset risk model.

6 Discussion

The work done by Sager and Baranoff is statistically robust and uses the three equations to jointly describe a single structural model. For example, the same asset risk variable that appears on the left-hand side as the Y variable in one equation also appears as an X variable in the other two equations. When estimated simultaneously, as by Sager and Baranoff, the estimates of the asset risk response variables (Y) in one equation would be the same values of the asset risk variables, when used as predictors (X) in the two other equations. Similar remarks apply to the other two response variables. In my work, I estimate the three equations separately and focus on reduction of variance. Separate estimation induces bias and inconsistency in the context of the simultaneous structural model. However, my objective is to investigate possible applications of neural networks. I explore the possibility of using neural networks to achieve improved precision of estimation in the structural model. The first step in this is to investigate whether estimation of the separate structural equations can be improved. Integration of neural network estimation into the complete simultaneous framework is very complex and was a larger project than could be essayed for this report. The next section discusses what would be involved in the implementation of neural network estimation in the full simultaneous context. Certainly, if my project had found inferior estimation of the separate equations, then there would be little hope for improving simultaneous estimation through neural networks. The fact that I achieved meaningfully improved estimation of the separate equations lends encouragement to the extension of my project to the full simultaneous context. Though the comparison is therefore not fair, considering the low losses achieved by the neural network models, I am confident that modifications (discussed in the next section) to account for

bias and inconsistency can lead to an improvement over the linear estimation methods of the simultaneous equations model. Neural network models achieve MSEs that are much smaller than the linear model. The validation and the test losses have the characteristic convex shape. The losses decrease and then increase. The increase is a result of over fitting on the training dataset. For making predictions from new data, the parameters from the iteration with the minimum test/validation loss should be used. The test and the validation sets are chosen at random. From the identical behavior of test and validation losses, I conclude that the models are stable and can generalize to new data. Analyzing the predictor importance tables, I find that, for the three feed-forward models, lagged response variables are the most important variables. The importance was expected as Sager and Baranoff discuss these time dependencies and use an auto-regressive model in their paper.[1] Importance of the lagged response for the asset risk model and the capital-asset ratio model outweighs the importance of other predictors by an order of magnitude. The reduced model without the lagged predictor for product risk has an MSE that is 12.83 times of the model with the lagged predictor. RBCratio, the second most important predictor, only changes the MSE by 60 percent. A possible explanation can be that most of the companies follow similar underwriting standards for their products and thus the product risk for a company mostly depends on the market standards and past trends. The importance of 'capital coverage of investment risk' can be explained by the reason that a company with low capital coverage of investment risk would prefer lower product risk. The change in MSE for the lagged response in the capital-asset ratio model is 40 percent. This can be explained by the fact that capital-asset ratio is regulated and thus depends more on the requirement and industry standards. All other predictors change the MSE by less than 4 percent and are practically insignificant. Predictor importance table for the asset risk model provides interesting insights. The lagged response is the most important predictor; removing it increases the MSE to 5 times that of the full model. Unlike the other two models, this model has other predictors that have importance in range of the importance of the lagged response. The predictors for return on capital, total asset, and capital coverage of investment risk changed the MSE by 308, 288 and 145 percent respectively. Possible interpretations for the importance of these predictors can be:

- companies with more assets prefer more asset risk.
- companies with lower capital coverage of investment risk are more prone to regulatory intervention and thus would take less risk on their assets.
- Companies earning more return on their capital take more risk on their assets.

LSTM models perform less adequately than the feed-forward models, but have lower MSEs than the linear models. A comparison between the models and without lagged predictors shows a comparable performance by both the versions. The models with lagged predictors performed marginally better than the models without them. This could be attributed to limited data and computing power. RNNs are infamous for high data and computational requirements. Niklas Donges states, "RNN's are relatively old, like many other deep learning algorithms. They were initially created in the 1980's, but can only show their real potential since a few years, because of the increase in available computational power, the massive amounts of data that we have nowadays and the invention of LSTM in the 1990's"[14]. The 'computational requirement' argument is further supported by the fact that minimum MSEs are achieved at the end of the training process and a weak trend is present which can lead to further reduction, if training is continued.

The predictor importance table for the LSTM based asset risk model shows a reduced importance of the lagged response variable. Lagged response is the fourth most important response variable for this model. This contrasts with the importance of the lagged response in the feed-forward asset risk model. This reduction is caused because RNNs can account for time dependencies using the sequence of the input data. The information from previous time periods is passed to future time periods using the state of the cells(LSTM) that make up the recurrent network and the need for variables providing information from previous time-periods is eliminated.[5]

7 Conclusion

From my research I conclude that feed-forward and recurrent neural network models have smaller losses than the linear model used by Sager and Baranoff in their work[1]. A summary of MSEs for

the different models investigated during the study is given by Table 11.

Model	Linear Model	Feedforward	LSTM	LSTM without lag resp.
Product Risk	0.370526	0.007792453	0.052094657	0.062157836
Asset Risk	0.0409973	0.000351615	0.001920539	0.002519771
Capital-Asset Ratio	0.2132114	0.061007324	0.058613993	0.061472513

Table 11: Performance(MSE) Summary

Predictor importance estimated using change in MSE gives a fair sense of importance, but linear models with their robust statistical tests do a better job at assessing importance of predictors. From the second part of the research I conclude that LSTM based RNN models can be used to model quantities with time dependencies.

As noted in the preceding section, the MSE of the linear model is not directly comparable with the MSE's of the neural network models, on account of the linear model equations being estimated simultaneously and the neural net models being estimated separately. To tackle bias and inconsistency introduced by separate estimation, the NNet model can be modified to follow a two-stage procedure, similar to what was used by Sager and Baranoff. In the proposed modification, the first stage uses NNets to estimate the first-stage model. This is a separate equation estimation of the reduced form of each separate equation. The reduced form expresses each of the three Y's as a function of the other non-Y predictors. Having estimated the Y's in this manner, these estimates are then substituted into the right-hand side of the three structural equations, which are then estimated separately in the second stage. This two-stage approach can avoid the bias and inconsistency of separate equation estimation. Another extension of this work could be the use bi-directional RNNs and addition of attention mechanism. Investigation of a sequence to sequence model for making predictions for a sequence of future time-periods can be another possible extension to this work.

References

- [1] Etti G Baranoff and Thomas W Sager. The relations among asset risk, product risk, and capital in the life insurance industry. *Journal of banking & finance*, 26(6):1181–1197, 2002.
- [2] George Bebis and Michael Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31, 1994.
- [3] DENNY BRITZ. Recurrent neural networks tutorial, part 1 – introduction to rnns, 2015. [Online; accessed 28-April-2018].
- [4] James W Denton. How good are neural networks for causal forecasting? *The Journal of Business Forecasting*, 14(2):17, 1995.
- [5] Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *Neural Nets WIRN Vietri-01*, pages 193–200. Springer, 2002.
- [6] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.
- [7] Jason Brownlee . Gradient descent for machine learning, 2016. [Online; accessed 28-April-2018].
- [8] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.
- [9] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.
- [10] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

- [11] Sovan Lek, Marc Delacoste, Philippe Baran, Ioannis Dimopoulos, Jacques Lauga, and Stéphane Aulagnier. Application of neural networks to modelling nonlinear relationships in ecology. *Ecological modelling*, 90(1):39–52, 1996.
- [12] Charlotte H Mason and William D Perreault Jr. Collinearity, power, and interpretation of multiple regression analysis. *Journal of marketing research*, pages 268–280, 1991.
- [13] Michael Nielsen . How the backpropagation algorithm works, 2017. [Online; accessed 28-April-2018].
- [14] Niklas Donges. Recurrent neural networks and lstm, 2018. [Online; accessed 28-April-2018].
- [15] Vivak Patel. The impact of local geometry and batch size on the convergence and divergence of stochastic gradient descent. *arXiv preprint arXiv:1709.04718*, 2017.
- [16] Fernando J. Pineda. Generalization of back-propagation to recurrent neural networks. *Phys. Rev. Lett.*, 59:2229–2232, Nov 1987.
- [17] Ashutosh Singh. Risk modeling using neural networks git hub code, 2018.
- [18] Jan Snyman. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*, volume 97. Springer Science & Business Media, 2005.
- [19] Jack V Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology*, 49(11):1225–1231, 1996.
- [20] ujjwalkarn. A quick introduction to neural networks, 2016. [Online; accessed 28-April-2018].
- [21] Wikipedia contributors. Cross entropy — Wikipedia, the free encyclopedia, 2018. [Online; accessed 28-April-2018].
- [22] Wikipedia contributors. Long short-term memory — Wikipedia, the free encyclopedia, 2018. [Online; accessed 28-April-2018].
- [23] Wikipedia contributors. Long short-term memory — Wikipedia, the free encyclopedia, 2018. [Online; accessed 28-April-2018].
- [24] Wikipedia contributors. Recurrent neural network — Wikipedia, the free encyclopedia, 2018. [Online; accessed 28-April-2018].
- [25] Wikipedia contributors. Softmax function — Wikipedia, the free encyclopedia, 2018. [Online; accessed 28-April-2018].