2023

An Exploration of Machine Learning & Artificial Neural Networks to Predict the Unemployment Rate in Australia

TOTAL PAGES: 12

JAYDEN DZIERBICKI

Contents

Abstract/Executive Summary

The Australian economy has undergone various macroeconomic events in the past 21 years, impacting the unemployment rate, which is a crucial metric used to guide policy decisions and monitored by the ABS. This study examines the potential of machine learning and neural networks to predict and model unemployment by incorporating macroeconomic indicators, such GDP, terms of trade, CPI, job vacancies, ERP, general government consumption expenditure, and all sectors consumption expenditure. To assess performance, the root mean squared error was used as a metric, and residual plots were examined to compare the performance of these models against a basic naive model. The multivariate adaptive spline regression model (MARS) demonstrated superior performance compared to random forest and boosting models. However, the test RSME was slightly higher than the cross-validation RSME, likely due to modelling the COVID-19 pandemic's impact on unemployment. In addition, a basic neural network with two hidden layers outperformed the MARS model with a lower test RSME; though interestingly had a worse cross validation RMSE. Through increasing the complexity of our neural network to include more neurons we were able to reduce the cross validation RMSE, though MARS still outperformed in this respect. The predictive capability of these models can be utilized to predict unemployment during the waiting period for ABS unemployment reports to be published. However, it is essential to note that unpredictable events such as natural disasters, as observed during COVID-19, could impact the models' predictive power, as they are only as reliable as the training data.

Introduction - Australia Unemployment: 1999 and 2020

The Australian Bureau of Statistics (ABS) is Australia's national statistical agency which is responsible for collecting employment statistic such as the unemployment rate, one of many employment statistics. The unemployment rate is a key indicator of the labour market performance, providing a snapshot of the available labour supply at a particular time. The survey is conducted on about 26,000 dwellings, with responses from around 52,000 people every month, ensuring it forms a representative sample of the Australia population, with respondents being asked self-guided questions (How the ABS Measures Unemployment, 2022). There are various limitations of the unemployment rate, with the most obvious being underemployment, with the true unemployment rate tending to be larger than the actual unemployment rate.

The Australian government often relies on the unemployment rate to shape policy decisions, especially during election cycles (Cockburn et al., 2022; Visentin et al., 2022). High unemployment rates have even been associated with election outcomes (Leigh & McLeish, 2009), which can lead to a change in government. Over the past 21 years, the Australian labour market has experienced various macroeconomic events that have influenced the unemployment rate, which has averaged around 5.6% during this period. After the recession of the 1990s, the labour market started to recover in the early 2000s, with the unemployment rate dropping to around 4.1%. However, the global financial crisis (GFC) of 2007-2008 had a significant impact on the Australian economy, causing the unemployment rate to rise to 5.75% (RBA, 2010). During such economic events, the unemployment rate is often linked to various macroeconomic indicators, such as GDP, with declining GDP leading to an increase in unemployment, as seen during recessions or the GFC (Higgens, 2011). The Covid-19 pandemic in 2019-2020 caused a sharp increase in unemployment to 7.1%, but unlike traditional recessions, the movement of other indicators such as GDP and house prices did not follow a typical recession pattern (Owen, 2022). This paper aims to predict unemployment using various macroeconomic indicators such as GDP, terms of trade, CPI, job vacancies, ERP, general government consumption expenditure, and all sectors consumption expenditure, by training two models on data from 1981 to 2018. A grid search approachwas used to tune the hyperparameters of the models. However, we acknowledge that the

models may perform poorly during events such as the Covid-19 pandemic, which deviates from normal economic conditions. Therefore, the ABS survey will still be valuable and necessary for measuring unemployment during such events.

Data

Overview

Data was obtained from the JCU MA5832 content page and downloaded from the assessment 3 folder on 14th February 2023, labled "AUS_Data.xlsx"; additional data labelled "Dec2020.xlsx" was downloaded on 23rd February 2023 and the row associated with December 2020 was manually pasted to "AUS_Data.xlsx" to include in the analysis. The data was saved locally and imported into R for data pre-processing. The dataset provided is aggregated and collected from the ABS which contains data provided quarterly from June 1981 to September 2020, which includes the Australian unemployment rate as well as various macroeconomic indicators which are described in table 1. There was a total of 9 inputs and 158 observations, relatively small data set. We observed some extreme values and shifts coinciding with the Covid-19 pandemic for GDP and final consumption expenditure; though these appeared to be correct (ABS, Job vacancies, Australia, November 2022; Massimo et al., 2022). In addition we utilised the lubridate package to split our period into month and year via the month() and year() function respectively, allowing us to capture any time-related pattern or trends based as we anticipate seasonal influence on the unemployment rate (Pettinger, ND).

Missing values

Utilizing the sapply() function we produced a column wise summary to detect any missing observations in our data set. We observed that both Job Vacancies and ERP contained a total of 5 missing observations each, we then called on the ts() function and plot() function to visually inspect the missing observations. We observed missing observations for Job Vacancies around the period of the GFC 2008-2009; whilst missing observations for ERP around the period of the covid-19 pandemic 2019-2020. We attempted to locate both original data before attempting to impute any missing observations and were able to locate a data set from the ABS for ERP.

Estimated Resident Population X7: Data was obtained from the ABS directly (ABS, *National, state and territory population, June 2022*) and saved locally as "310101.xlsx". Upon further inspection we observed that ERP in the original data frame was reported in the thousands when it should have been in the hundreds in line with ABS reporting. We elected to remove ERP and appended the correct figures from the "Data1" sheet and renamed it X7_2 (ERP) linking the data by period onto our data frame.

Job Vacancies Imputation X6: We were unable to locate any data directly from the ABS, utilising domain knowledge we assumed that we would expect a decrease in Job vacancies due to the GFC which occurred during this period. Utilizing the imputeTS package we deemed the na_interpolation() function appropriate which produced values we considered appropriate based on the conditions at the time, which we confirmed visually.

Distribution, Outliers & Summary statistics

We observe that many of our variables have various distribution types; with the data containing evidence of potential outliers evident from the boxplots (figure 6 appendix). Given that many variables are not normally distributed we decided to explore for outliers utilising the IQR approach as opposed to the z-score approach which assumes gaussian distribution which we did not observe with figure 6; we observed many possible outliers based on this approach but assumed these to be true observations and as such did not omit them as they represent extreme economic events

such as recessions, and could be valuable to keep them in the dataset to train the model; we confirmed some of the extreme data points by consulting source data and external sources as a sanity test (ABS, Job vacancies, Australia, November 2022; Massimo et al., 2022). Including such extreme events can help the model to capture the full range of variation in the data and produce more robust predictions. Table 1 suggests that between the whole data set (1981 and 2020) the unemployment rate in Australia has had a mean of 6.85% and a standard deviation of 1.78%, with a minimum value of 4.10% and a maximum value of 11.13%; this highlights the dynamic nature of unemployment, and other macroeconomic indicators which experience variation in the spread suggesting the importance of retaining the whole data set for the model to train on; highlighting that the economy is dynamic.

Table 1: Summary statistics of numeric variables, demonstrating a wide spread of many macroeconomic indicator. Refer to appendix corresponding boxplot summary

Variable full name	R-name	Mean	SD	Median	Min	Max	Missing Values Imputed
Unemployment rate	Y	6.85	1.78	6.26	4.10	11.13	NA
GDP %	X1	0.74	0.99	0.70	-7.00	3.40	NA
General government ; Final consumption expenditure: Percentage	X2	0.85	1.66	1.00	-4.60	7.5	NA
All sectors ; Final consumption expenditure: Percentage	X3	0.77	1.11	0.80	-8.30	5.90	NA
Terms of trade: Index - Percentage	X4	0.35	2.95	0.30	-8.10	13.20	NA
CPI (all group)	X5	75.34	25.16	73.90	28.40	117.20	NA
Job Vacancies (000)	X6	114	57.31	103.0	26.80	259.2	Yes, imputed utilising imputeTS package
Estimated Resident Population (000)	X7_2	19715	3103.70	19225	14923	25655	Yes, all values replaced with updated ABS data

Training and Test data set

At this stage the data was separated into two data frames, one titled 'train' and the other titled 'test'. The split for train was applied via the filter() function on data points occurring prior to March 2018 whilst test included March 2018 and onwards, the train and test data frames contained 147 and 12 observations respectively and each with 10 variables summarised in table 2. In addition, we elected to remove "period" from both test and train via the select() function as we would capture the cyclical nature of unemployment from the engineered feature of month and year. In addition, for our neural network model we created two new matrixes based on the training and test data frames discussed above, we then hot-encoded the month and year into a dummy variable utilising the dummyVars() function from the caret package; we then scaled all numeric variables (excluding binary variables) for our neural network model. In addition, we decided to train our models on all available data which includes from 1981 onwards to capture the various economic events which have occurred throughout history and not limit us from 1999 only; whilst not shown here or discussed any further we observed an increase in RSME/MSE when we excluded data points prior to 1999 in the training data as another justification for retaining the whole data set based on the models tested.

Table 2: Summary description of variables both, with the inclusion of engineered features such as month and year. Additional data processing steps might occur at a later stage depending on underlying model used.

Variable full name	R-name	Datatype/R Coercion	Description	Comment
Unemployment rate	Y	numeric	representing the percentage of unemployed individuals in Australia.	
GDP %	X1	Numeric/as.numeric()	representing the growth rate of Australia's Gross Domestic Product (GDP)	Imported as character, converted to numeric manually.
General government ; Final consumption expenditure: Percentage	X2	numeric	representing the percentage of final consumption expenditure by the general government in Australia.	
All sectors ; Final consumption expenditure: Percentage	X3	numeric	representing the percentage of final consumption expenditure by all sectors in Australia.	
Terms of trade: Index - Percentage	X4	numeric	representing the percentage change in the terms of trade index in Australia.	
CPI (all group)	X5	numeric	representing the percentage change in the Consumer Price Index (CPI) for all groups in Australia.	
Job Vacancies (000)	X6	numeric	representing the estimated number of job vacancies in Australia	Imputed variables around time of GFC
Estimated Resident Population (000)	X7_2	numeric	representing the estimated resident population in Australia,	Replaced X7, obtained from ABS due to errors in original supplied data.
Month	Month_lag	Ordinal categorical/as.factor()	Month derived from period, converted to factor with 3 levels	Manually derived from period, allowing us to capture potential cyclical nature.

Year	Year	Ordinal categorical/as.factor	Year derived from period, converted to factor with multiple lelevels	

Method

We conducted two types of supervised learning, using a MARS model and a neural network model, both using the training data described above. These algorithms were implemented using R and RStudio (Version 4.2.2) on an x86_64-w64-mingw32 platform. To ensure reproducibility, we used the set.seed() function with a seed value of 123. We assumed that all the data was correct, despite some minor imputations made for Job vacancies and ERP as previously discussed. We also assumed that any outliers in the data represented extreme economic conditions, which have been documented in the literature and should therefore be retained (ABS, Job vacancies, Australia, November 2022; Massimo et al., 2022). Our goal was to test our ability to predict unemployment based on various current metrics, which we summarize with a simple equation for illustrative purposes. We did not explore the possibility of forecasting unemployment, although lagging the variables and removing the unlagged variables could be used instead.

$$Unemployment_t = B_1 X1_{1t} + \cdots + B_n X_{nt} + error_t$$

Since our primary objective was to develop a predictive model, we elected to utilise the root mean squared error (RMSE) as our metric of choice when evaluating the performance of our model, which we can define formally with the following equation:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - Ypredicted_i)^2}$$

In addition, we compared all our model's performance against a naïve persistence model. The naïve model simply assumes that there is no change in unemployment rate from one period to the next. This is a very basic and naïve model; but serves an important purpose as it allows us to compare a reference point on our more advanced models (Nair, 2022). We defined our naive model with the following simple equation, which resulted in a validation MSE of 0.6122222 (RSME 0.7824463) as a baseline for our reference, which can be explained utilising the following equation:

$$Unemployment_{t+1} = Unemployment_t$$

Analysis and Investigation of Machine Learning Methods

Model selection

We evaluated various supervised machine learning models, including boosting, random forest, and multivariate adaptive regression splines (MARS). To select the best model, we reviewed each model's strengths, limitations, and assumptions in light of the specific characteristics of our dataset, such as its size, complexity, and variable types. Although the models shared many similarities, we decided to run three "out of the box" models with default settings in R and evaluate the residuals and mean squared error (MSE)/RMSE before selecting the desired model.

Based on our analysis, we chose the MARS model because it had the lowest test RMSE and the second-lowest train RMSE, as shown in Table 3. Although the random forest had a smaller train RMSE, its test MSE was only marginally better, and it displayed evidence of systematic bias in the training residual plots (figure 7 appendix). In contrast, the MARS training residual plot showed improved randomness with little obvious pattern or structure. However, all models appeared to be overfitting, as evidenced by the difference between the high test MSE and low train MSE, which could be associated with low bias and high variance. Nevertheless, all models, except boosting, outperformed the naïve model. We also conducted basic grid searching to optimize hyperparameters for both random forest and boosting, but this did not improve the results or residuals compared to the "out of the box" models.

While the "out of the box" approach can reduce computational processing time and help select the best model, it has one obvious flaw; the best model may require hyperparameter tuning initially. This highlights a major limitation in selecting which machine learning model to use and tune. The aspect of computational processing time becomes more evident with big data.

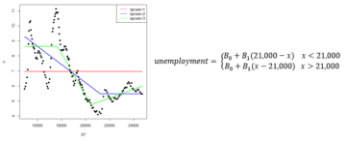
Table 3: Performance of out of bag models comparing test and train RSME, both Random Forest and Boosting performed poorly with a test RMSE larger than the naïve model. We selected the MARS model due to performance at this stage.

Metrics	Random Forest	MARS	Boosting
Train MSE | RMSE	0.02870791 | 0.1694341	0.04736039 | 0.2176244	0.08096076 | 0.284536
Test MSE | RMSE	0.6030101 | 0.7765373	0.4132105 | 0.6428145	1.723489 | 1.312817
Comment on train residual plot	Residual plot is not random, upward trend from left to right. All residuals when unemployment less than 5% are negative.	Residual plot is random, some possible outliers. All residuals when unemployment less then 5% are negative	Many residuals are below 0 indicating possible bias in the model
Comment on test/train MSE	Model could be overfitting to training data. Low train and high test MSE suggest model not generalizing well to unseen data	Model could be overfitting to training data. Low train and high test MSE suggest model might not be generalising well to unseen data	Smallest difference, though test MSE almost identical to naïve model.
Run time	0.03983212 secs	0.00947094 secs	0.006984949 secs
R-Package used	randomForest	earth	gbm

The MARS Model (NB Marker: This was covered in R-Demo week 3/Topic 3)

We discovered that the MARS model is suitable for complex non-linear regression problems, and it has been used to predict and forecast various models such as unemployment, energy demand, and sales in the literature (Lu et al., 2012; Katris, 2019). These models involve time-related and non-complex linear relationships, which are also present in our data. The MARS model, first introduced by Fridman (1991), is a nonparametric statistical method that employs a divide and conquer strategy by partitioning the training dataset into separate piecewise linear segments, or splines, with various gradients. It is similar to CART in that it uses recursive portioning to divide the data into subsets based on predictor variables, but it differs in that it uses basis functions like hinge functions and constant functions to model nonlinear relationships between predictor and response variables within each subset, allowing it to capture more complex nonlinear relationships than CART. The points where the pieces connect are called knots, and they mark the end of one region of the data and the beginning of another, which is achieved by minimizing some loss function.

A simple 2-dimensional model can be used to illustrate the MARS model's functioning, where it models unemployment on estimated population. The model starts by searching for the point where two different linear relationships between unemployment and estimated population achieve the smallest error through a hinge function, which is known as a single knot and corresponds to a single hinge function. Figure 1 shows a simple 2-dimensional demonstration of the MARS model and an equation with a single knot corresponding to the blue line. Once the first knot is found, the model continues to search for a second knot. If successful, it results in three linear models for Y (unemployment), as shown below, with the following hinge functions h(x-17,000) and h(x-20,500) approximately. The number of knots is an essential hyperparameter that must be tuned to avoid overfitting or underfitting our model, which we will discuss below.



$$unemployment = \begin{cases} B_0 + B_1(21{,}000 - x) & x < 21{,}000 \\ B_0 + B_1(x - 21{,}000) & x > 21{,}000 \end{cases}$$

Figure 1: Simple 2-dimensional demonstration of MARS model and an equation with a single knot corresponding to the blue line.

Hyper-parameters

To optimize the MARS model, it is essential to tune two critical parameters: the maximum degree of interactions and the number of terms in the final model. Grid search can be used to find the best combination that minimizes a loss function such as RSME. A grid search of 30 different

interaction complexities and 10 evenly spaced values for nprune is a good starting point, as there is little benefit in assessing greater than third-degree interaction (Greenwell & Boehmke, 2020). We used the caret package with the earth method and the previously described train data set, which took 38 seconds to run. The optimal combinations, first-degree interaction with 23 retained terms, were obtained from the first grid search. To further improve the model, we conducted a second grid search ranging from 10 to 23 and obtained the optimal combination of nprune 21 with first-degree interaction. The final optimal model had a cross-validation RSME of 0.3548613 with a total run time of 61 seconds, including both iterations of the grid search. Other hyperparameters may be tuned, but for the MARS model, we only focused on the two parameters discussed above and highlights a common challenge in machine learning as to selecting what needs optimising.

Report on Performance and Interpretation of obtained MARS Model on Training Data

Table 4 presents various metrics that evaluate the performance of our MARS model on the training dataset. We note that the training RMSE of 0.2176244 is lower than the cross-validation RMSE of 0.3548613, indicating a possible overfitting issue, which is a common problem in machine learning. However, the difference between the two errors is relatively small, which suggests that the model may not be significantly overfitting. Nonetheless, we did not explore this further. The cross-validation RMSE is also lower than the RMSE on our naïve model, indicating that the MARS model is performing relatively well on new data. One benefit of the MARS model is our ability to interpret variable importance; we reviewed the top five variables determined by the MARS model to be the most important predictors of unemployment, job vacancies and CPI were not surprising, but we expected GDP to be included. The inclusion of 1983 and 1999 as important variables for the hinge function came as a surprise, and when plotted visually, these periods were associated with an increase in unemployment. Although these variables are important for predicting unemployment based on the training data, they may not be representative of the current state of the economy. As such, the model may be overfitting to these variables and perform poorly when presented with new data from different periods. This highlights the importance of feature selection in machine learning, and it may be a reason why we hypothesize an increase in the repeated 10-fold cross-validation RMSE. Therefore, we need to further investigate alternative approaches.

Table 4: Performance of tuned MARS model on training data with cross valdation accuracy included. We observe that the model perfoms relativly well on the training data. In addition, cross validation accuracy was less than the naïve model RMSE suggesting the model is able to adapt to unseen data. Possible evidence of overfitting.

Metric	MARS Model (nprune = 20, degree = 1)
RMSE	0.2176244
Repeated 10-fold- Cross-Validation RMSE	0.3256355

Whilst the residual plot shows randomness, we do observe some possible outliers which could warrant further investigation, though we did not explore this further and mark it as a limitation of the model. We also observe that as unemployment increases the residual plot appears to become narrower which could suggest improved accuracy in predicting unemployment when unemployment is higher.

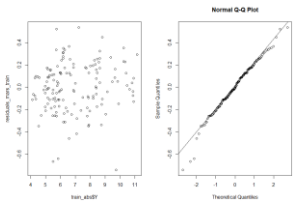


Figure 2: Residual plot predicted against actual on training data set. We observe evidence of randomness with possibility of outliers. As unemployment increase the residual plot becomes narrower suggesting possible improved accuracy as distribution around 0 still appears random.

Report On Performance and Interpretation of obtained MARS Model on Validation Data

We have compiled several metrics in table 5 in relation to the performance of our MARS model on the validation data set. What we observe is that the validation RMSE of 0. 6428145 is higher than the cross-validation RMSE of 0.3548613. This suggests that our model did not perform as well on the validation data as it did on other unseen data. However, the model's performance on the other unseen data was comparatively better. Going forward we would suggest acquiring new data for the period 2021 to 2023 to see how it performs on new unseen data as the validation data set only contained 12 observations which may not be sufficient to accurately assess the model's performance; a common approach is to use a 80:20 ratio for training and validation respectively, which would result in 31 observations instead of the 12 we used based on the current sample size. Ideally going forward we would recommend increasing the overall sample size to include observations from 2020 to 2023 and applying the commonly used ratio. In addition, we highlighted previously that the importance of 1983 and 1999 may lead to overfitting and could be a reason why our model performed poorly on the validation data set as these periods are not associated with the current state of the economy.

Table 5: Performance of tuned MARS model on validation data with cross valdation accuracy included. We observe that the model perfomrses worse on the validation data then it did on cross validation data, both metrics provided on utilising unseen data. We hyptohisis this could be due to extreme condions caused by covid-19 pandaic as the model was not trained on pandamic like conditions.

Metric	MARS Model (nprune = 20, degree = 1)
RMSE	0.6428145
Repeated 10-fold- Cross-Validation RMSE	0.3256355

We analysed the residuals of the validation data set (figure 3) and identified two observations that were underestimated, predicting unemployment at around 5% for quarter three and four of 2020 when in actuality it should have been in excess of 6-7%. We attribute this deviation to the impact of the Covid-19 pandemic and suggest the need for additional post-Covid-19 data to more accurately evaluate the model's ability, as the pandemic represents a deviation from the normal economic conditions. Furthermore, unforeseen events like Covid-19 can impact the model's performance, highlighting the need for caution and the inclusion of more diverse data sets. Despite high unemployment during the pandemic, there were still instances of positive GDP growth and high job vacancies due to government policies and interventions, as well as strong house price growth. While not shown here, we were able to improve the RMSE of the validation data when we developed a lag MARS model by lagging all predictors by one time period. This resulted in a revised RMSE of 0.6076795 on validation data. We propose to refine the model further by including lagged variables and feature selection to reduce overfitting, such as the removal of the year variable. This illustrates the complex nature of developing a machine learning model, where the accuracy of the model is dependent on the training data and the presence of unforeseen events like the Covid-19 pandemic can adversely impact the model's performance, despite a relatively low cross validation RMSE initially obtained.

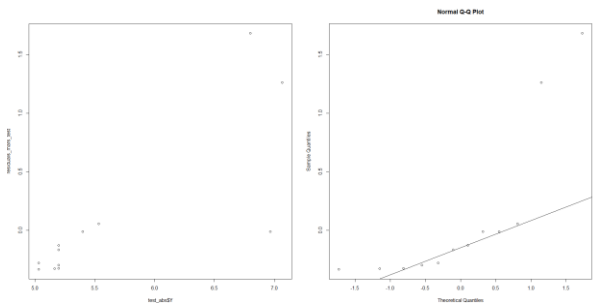


Figure 3: We observe based on the residual plot on validation data that the MARS model potentially underestimates when unemployment exceeds 6.5%

Analysis and Investigation of Neural Network

For this inquiry, we opted to utilize the neuralnet package in R instead of the keras package, as it offers a straightforward approach to comprehending the fundamental principles of neural networks (NN) before delving into more sophisticated packages like keras that are more suited for advanced users with experience in NN models; in addition we experienced connectivity issues with the keras package which would make it work intermittently. The NN model has evolved to encompass a wide range of models and learning techniques that are inspired by the functioning of the human brain. To grasp the workings of NN, we can draw an analogy with the human brain cells, referred to as neurons, which form a complex interconnected network that transmits electrical signals to each other to process information. The NN is comprised of artificial neurons that aim to solve a diverse range of practical problems, such as image and text recognition, among other applications. These artificial neurons, also known as nodes, form a basic vanilla neural network that comprises interconnected artificial neurons in three layers. These three layers include an (1) input layer where information from the external environment enters the NN and is transferred to the next layer, (2) the hidden layer that receives input from the input layer (or other hidden layers in complex networks), analyses the output from the previous layer, processes it further, and passes it on to the next layer, and (3) the output layer that provides the final result and prediction of the NN. For a regression problem, we have only one output layer. The structure of a basic NN is illustrated in Figure 4.

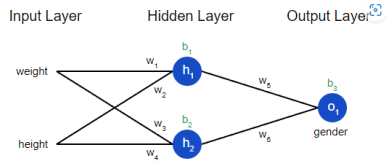


Figure 4: A simple neural network with 1 hidden layer for illustrative purposes.

Backpropagation is a critical technique in neural networks that involves adjusting the weights of the network based on the error rate (loss) from the previous epoch. The weights control the strength of the connections between neurons and determine how much each neuron contributes to the final output of the network. The purpose of backpropagation is to increase the generalization ability of the model and reduce overfitting. In regression neural networks, the sum-of-squared errors (SSE) is commonly used as a measure of fit. During forward propagation, information flows from the input node through the hidden layers and finally out the output node. After calculating the SSE, the algorithm will undertake backpropagation to fine-tune the weights to minimize the SSE. Gradient descent is often used to determine the optimal weights. However, this approach is vulnerable to issues such as vanishing gradients, overfitting, and local minima, which can lead to problems when attempting to update the weights (Hastie et al., 2017); again, highlighting the challenges faced with hyper-parameters when dealing with machine learning. With keras offering a range of solutions to overcome issues of overfitting (Sagar, 2019) through better control of hyper-parameters.

Report the performance(s) and interpretation(s) of the produced NN models on the training dataset.

When setting up a neural network, it is important to determine the appropriate number of neurons and hidden layers. Research suggests that many problems can be solved using just one or two hidden layers, with additional layers adding unnecessary complexity and computation time (Mic, 2015; M, 2011). Therefore, we limited our analysis to one and two hidden layers. We attempted to perform a grid search with three hidden layers, but it was computationally expensive and could not be completed in a timely manner. To determine the optimal number of neurons in each layer, we limited the number of neurons in each hidden layer to be less than the number of input features (Mic,

2015), we will come back to this point later. We compared the validation RMSE values for each model and selected the one with the smallest test RMSE. We achieved this by performing a grid search with the number of neurons ranging from 1 to 51 and recording both the validation and training RMSE. We selected the number of neurons that resulted in the lowest test RMSE as the optimal choice. After comparing the results, we elected to proceed with the two hidden layer neuron model since it scored better on the validation RMSE than the optimized one hidden layer neuron model and our naïve model as summarised in table 6.

Table 6: Performance of neural network model in R, comparing one hidden layer agaisnt two hidden layer across various metrics such as validtion, train and cross validation RMSE. All models indicate potential overfifting.

	One hidden layer	Two hidden layer
Number of neurons in layer 1	17	49
Number of neurons in layer 2	-	18
Validation RMSE	0.9868	0.3497
Train RMSE	0.00155	0.00129
Run Time	9.34 seconds	30.57
Cross validation RMSE	1.2377	1.1148

Furthermore, upon reviewing the residual plot of the training data, we noticed that the residuals were extremely close to zero, indicating that the model may be overfitting as depicted in figure 4. Additionally, we observed that as the unemployment rate increases, the spread also increases, which implies that the model might be under or overestimating unemployment levels when they are above 10%. Nevertheless, this disparity is negligible in absolute terms, with the residual being only 0.01% off, which is much narrower than what we observed in the MARS model. Both models produced high root mean square error (RMSE) values during 10-fold cross-validation, which were considerably higher than the training and validation RMSE values. This could suggest that the model has poor generalization capabilities for unseen data. We executed the 10-fold cross-validation by creating a custom loop that divided the data into 10 equally sized folds and computed the average RMSE across all folds. In conclusion, it appears that the model is overfitting on the validation data, as evidenced by the low validation RMSE and relatively high training RMSE and high cross-validation RMSE which was worse than our naïve model. Furthermore, the narrow spread of the residual plot indicates overfitting on the training data and issues occurring when unemployment is about 10% which isn't being captured correctly.

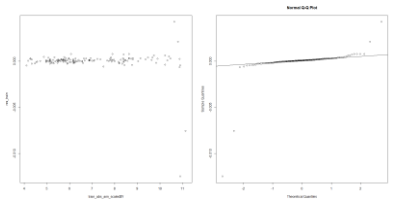


Figure 4: Comparing residual plot of neural network with two hidden layers, each hidden layer comprising of 49 and 18 neurons on training data

Regarding the interpretation of the NN model, it is a more opaque approach. While there are several ways to extract variable importance, like the MARS model, such methods require additional processing steps which we did not have the time to explore. Additionally, interpreting the weights of the NN model proved challenging, as the weight matrix contained 3470 rows. However, we did observe that inputs associated with the year had large magnitude weights, indicating a relatively stronger connection with the first hidden layer. Unfortunately, interpreting beyond this observation was unfeasible given the size of the weight matrix, making it hard to generalize why overfitting occurred. One possible explanation for the overfitting could be the strong connection between the input and hidden layer 1, though again the structure of the NN makes it challenging to say why.

Discuss the predictive performance of the model on the test dataset (March 2018 to December 2020)

According to the results presented in table 6, the two-hidden layer NN performed better than the naïve model in predicting the unemployment rate on the validation dataset, as indicated by its relatively low validation RMSE. However, the high CV RMSE, which was significantly worse than that of the naïve model, raises concerns about the model's ability to generalise well to new data, and supports the idea that the model is overfitting to the training data. In an ideal scenario, we would have utilised a cross-validation method to select the optimal hyper-parameters, such as the number of neurons and hidden layers, through a loop, and selected the model with the lowest CV RMSE. However, due to time constraints, we were unable to do so, which highlights the trade-off between processing time and accuracy in machine learning. Furthermore, we attempted to use the keras package, but faced intermittent connectivity issues, which made it difficult to explore this option using a more advance package, highlighting the importance of utilising the correct tools for any machine learning problem.

When we plotted the residuals of the validation data set, we observed that the model tends to overestimate the prediction when the unemployment rate is less than 6% and underestimate it when it is greater than 6.5%, as shown in figure 5. This could suggest that the model is not capturing the true relationship between the input variables and the target variable in those regions. As evidenced by the low train RMSE and high validation RMSE, this could be due to the model's poor generalisation to unseen data, which is also reflected in the cross-validation accuracy. As we previously hypothesised, this could be due to the inclusion of years in the model, which may have a significant impact on the state of the economy.

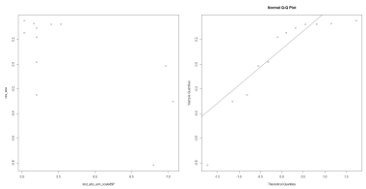


Figure 5: Comparing residual plot of neural network with two hidden layers, each hidden layer comprising of 49 and 18 neurons on validation data

Vary the number of hidden layers in the model 4(a). Explore the impacts of the change on the prediction performance of the model.

We conducted a grid search to identify the optimal number of neurons and hidden layers for our neural network model. The results are presented in table 6, where we can see that increasing the number of hidden layers while comparing two optimized models led to improved performance across all metrics, except for computational processing time. However, it is worth noting that adding more hidden layers does not necessarily mean better accuracy, as it can result in longer training and optimization times, as cited by Cansiz (2020). According to Sachdev (2022), there are several ways to determine the optimal number of hidden layers. If the data is linearly separable, no hidden layers may be necessary. For less complex data with fewer dimensions or features, 1-2 hidden layers may be sufficient. However, for data with a high number of dimensions or features, more than 3 hidden layers may be optimal. In our case, due to the size of the training data, we decided that exploring 3 or more hidden layers was not feasible due to computational processing time, especially through employing a grid search approach. Furthermore, we were unable to undertake a more in-depth grid search due to time constraints and limited resources. It is important to note that we would have liked to use cross-validation to select the optimal number of neurons and hidden layers, but we were unable to do so

within a reasonable timeframe. In summary, while increasing the number of hidden layers can improve the accuracy of the model which we seen, it also results in longer processing times. Ultimately, the optimal number of hidden layers is dependent on the complexity of the data and available resources. One possible issue of too many layers is that the model will become complex, which could result in overfitting (Sagar, 2019).

Vary the number of neurons in each layer in the model 4(a). Explore the impacts of the change on the prediction performance of the model

It is worth noting that there is no clear consensus on the optimal number of neurons to use in a neural network. While some sources suggest that the number of neurons should be less than the number of input features, others show figures where the number of neurons exceeds the number of features (Mic, 2015). Blindly following rules or discussion boards may lead to suboptimal results. To investigate the impact of the number of neurons on our model's RMSE, we extended our grid search to include up to 100 neurons instead of the initial 51 and present the best results in Table 7. We found that the one hidden layer model did not improve, but the two-hidden layer model showed a decrease in validation RMSE, indicating an improvement in performance in this example. However, increasing the number of neurons resulted in a much longer processing time of almost 3 hours, highlighting the trade-offs and complexities involved in selecting optimal hyper-parameters in machine learning. Additionally, some studies suggest that increasing the number of neurons does not necessarily improve performance and highlights a decision of selecting a cut-off in machine learning given resources available (Cansiz, 2020). One possible issue of too many neurons in a layer is that the model will become complex, which could result in overfitting (Sagar, 2019). Though in this example we have seen an increase in train RMSE which indicates less overfitting relative to table 6, and a reduction in validation RMSE which suggest the model is performing better on unseen data compared to table 6; this was also validated through a decrease in CV RMSE from 1.14 to 1.10. This highlights the complex process of selecting optimal number of neurons in each layer, and that there is no clear-cut answer as to the optimal number of neurons.

Table 7: Performance of neural network model in R, comparing one hidden layer agaisnt two hidden layer across various metrics such as validtion, train and cross validation RMSE. In this example our grid search compared up to 100 neurons in each hidden layer as opposed to limiting to 51 as seen in table 6.

	One hidden layer	Two hidden layers
Number of neurons in layer 1	17	4
Number of neuros in layer 2	-	85
Validation RMSE	0.986948	0.2701716
Train RMSE	0.001558822	0.01671
Run time	17 seconds	2.41 hours
Cross Validation RMSE	Refer table 6	1.1079

Comparison and Suggestion

There was an obvious winner in respect to processing time, that being the MARS model which only took 61 seconds to undertake a grid search approach across the various hyper-parameters, whilst the best NN model took a total of almost 3 hours when we expended the number of neurons in the hidden layers. It was also evident from the training residual plots that the NN model was heavily overfitting; and lacked randomness in the residual plot which we did not observe as much with the MARS model. In addition, the CV accuracy of MARS was much lower than NN, 0.32 vs 1.10 respectively; this suggests that the MARS model performed better in relation to unseen data, though the NN model performed better on our validation data with a lower RMSE of 0.27 vs MARS 0.64. We did raise concerns that the validation data set was simply not large enough as it did not represent an 80:20 split which is common in ML. The interpretation of the models is vastly different; we can extract hinge functions & variable importance with ease in the MARS model, though the NN model would require additional data steps to derive figures such as variable importance which are just not as straight forward; the NN is a much black box approach. Though with NN we observe the weights associated

between the layers such as the input layer and first hidden layer; we observed year tending to have a large absolute weight which represents a strong connection between the input and first hidden layer, though beyond the first hidden layer it becomes more complex with 3470 weights calculated in total. One thing we noted in our study was the use of lagged variables to predict unemployment which yielded improved RMSE in the MARS model, decreasing the optimised RMSE from 0.64 to 0.60; whilst a relatively small reduction we attempted to see if this effect translated into the NN model with improvements. The use of lagged variables in NN has been reported to lead to reduction in RMSE (Surakhi et al., 2021) though in our case resulted in a slight increase in RMSE once optimised resulting in validation RMSE of 0.354 vs 0.27; suggesting it made our NN model worse. This highlights that developing a ML algorithm is complex and requires significant resources both time and computationally, especially if the goal is to derive a well performing product as opposed to a sub-optimal product. When attempting to refine the model we observed that the gains at times can only be marginally improved, which depending on the context might no be worth the time to investigate. In addition, attempting to refine the model can also yield worse results highlighting the complex nature of machine learning which we observed with the use of lagged variables in NN; just because it worked for one model does not mean it will improve all models as we seen when we removed year. If time persisted and keras worked without API issues we would attempt the keras package to overcome some of the limitations of the NN model we produced, such as exploring various other hyper-parameters which keras offer. Though, we found through increasing the complexity of our NN model by the inclusion of more neurons that we obtained an improved validation RMSE and increased our train RMSE suggesting an improvement to overfitting which was validated by a decrease in CV RMSE for our NN when we attempted to increase the number of neurons. Overall, the NN is a much more complex black box model relative to our MARS model.

Conclusion & Lessons learned

Overall, both the MARS and NN models outperformed our naïve model RMSE in relation to the validation data, but we had concerns about overfitting and the insufficient size of our validation data sample to generalize about the models' performance. We observed evidence of overfitting, particularly in the NN model, which was apparent in the training residual plots. One limitation we encountered was the impact of the COVID-19 pandemic on our macroeconomic indicators. This event caused deviations from the usual trends seen during a recession, such as high unemployment and low GDP; whilst GDP dropped to -7% during Covid-19 it was quickly followed by 3% growth with high levels of unemployment. Therefore, the use of surveys to measure unemployment and other factors is crucial; and the limitations of any machine learning model must be considered. The accuracy of any machine learning model is only as good as the training data, and not all events can be captured in the training data as we live in a dynamic world. This means models will need to be periodically updated and assessed to ensure relevance, or different models employed based on different events, or the use of caveats around models developed. Furthermore, we demonstrated throughout the comparison and suggestion sections that feature engineering and selection can lead to improved results. No machine learning model is perfect, and a combination of various models or approaches may be necessary to achieve the best results for a particular problem; we observed previously through feature engineering such as lagging variables we could obtain an improved RMSE for our MARS model, though this did not translate to the NN model. Overall, we learned the importance of (1) considering a models interpretability, (2) trade-off between improved accuracy and computational processing time, (3) assessing the relevance of our models in a dynamic world, and (4) critical thinking of what could be the root cause of issues such as overfitting and attempting to overcome some of these. These are critical factors to consider when building and implementing machine learning models in real-world scenarios.

Appendix & R Code

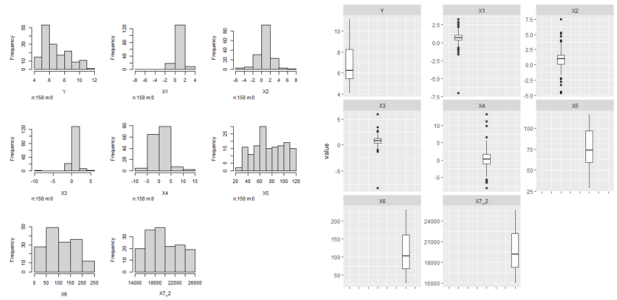


Figure 6: Distribution of variables, we observe data is of various distribution types and many outliers evident by boxplot

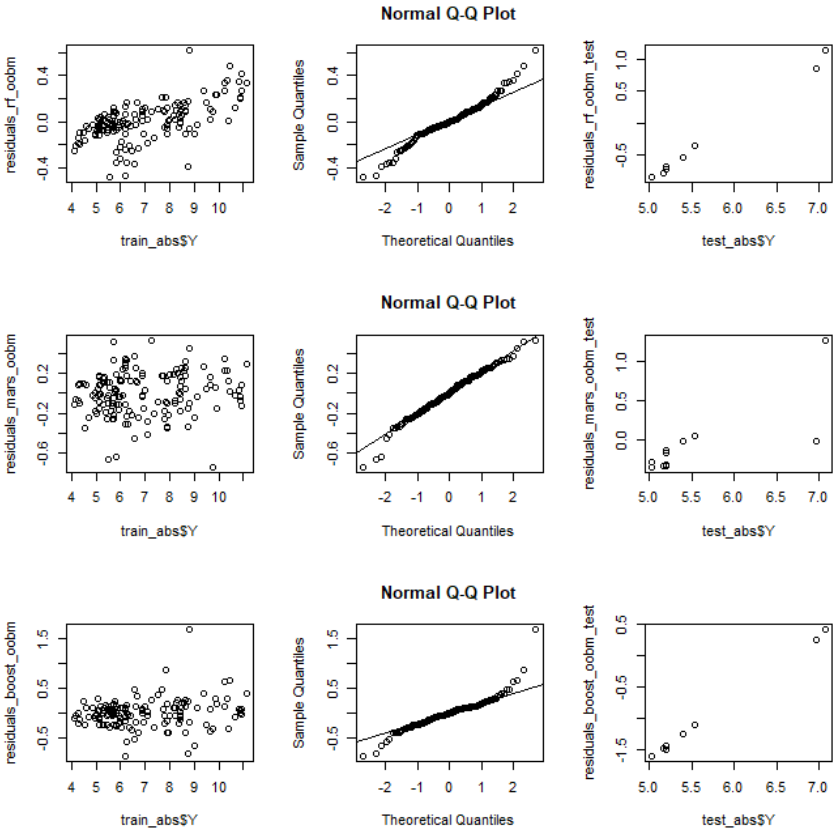


Figure 5: Comparison of out of box models, comping random forest (top), mars (middle) and boosting (bottom) with evidence of randomness observed in the MARS model .

```
#==============================================================================

# The purpose of this script is to explore unemplyoment is Australia over the

# last 20 years. The goal of the script is to compare 2 models ability to

# detect unemployment in Australia through the use of histoic data
```

```
#============================================================================
#
# Load in packages ============================================================
library(readxl) # Read excel files
library(dplyr) # Data wrangling
library(lubridate)
library(ggplot2) # For plot
library(gridExtra)
library(imputeTS) # Impute TS data
library(neuralnet) # NN modeling
library(Hmisc) # Multiple historgrams



# Load in data set and define column types in data import=======================
abs <- read_excel("AUS_Data.xlsx",
                  col_types = c("date", "numeric", "text",
                                "numeric", "numeric", "numeric",
                                "numeric", "numeric", "numeric"))
# Data column name cleaning
abs <- abs[-1,]
abs$period <- abs$...1
abs$...1 <- NULL


str(abs) # Ensure data in correct format, X1 laoded as character


abs <- abs %>% # Convrt X1 to numeric
  mutate(X1 = as.numeric(X1))


str(abs) # Confrim above works
summary(abs)
```

```
# We will also metric for our analysis to the period 1998. We will split date
# out into month and year as seasonality could be captured by doing so which
# might not be captured otherwise.
abs_clean_full <- as.data.frame(abs) %>%
  mutate(month = as.factor(lubridate::month(period)),
         year = as.factor(lubridate::year(period)))
(str(abs_clean_full))

# We will also only retain data from after 1999, as for if we should keep it
# or remove it for ML model it can depend on the context of the research question.
# Does the data before 1999 contain information that could help explain patters?
# Additionally, more data points mean a bigger training sample for the model
abs_clean_1999 <- abs_clean_full %>%
  filter(period >= "1999-01-01")

# Data Wrangling================================================================
# - Timeseries plot
# - MIssing observations
# - Summary stats
```

```
# We will not impute missing values yet, as this data is generally seasonal we

# deem it immature without actually having an understating of what is going on.



# Plot data: Both full and restricted to see how historically before

# 1999 things might have been

#

# Things which stand out from TS plot: - We observe missing observations

#                       - Unemployment was much higher between 1980 and 1990

#                       - Around 2019 we observe the following (possibly due to covid):

#                               - Increase Unemployment (Y)

#                               - Reduction/spike in GDP (X2)

#                               - Reduction/spike in All sectors; Final consumption expenditure %
(X3)

#                               - Reduction/spike job vacancies

#                               -*We will need to confirm this this is correct or error in data, as
based on

#                                 prior knowledge GDP looks a little odd

#                       - Around 2007+ we observe impact of GFC, increase in unemployment,
pssobile reduction in

#                        job vacancies, though appears to be NA values. In addition, spike observed
with covid for

#                        some variables is not observed in GFC eluding to further investigation of ABS
data before

#                        full machine learning model to be implemented.

#

# Overall, there is clear evidence of cyclical occurrence in the data set which correspondence to
'shocks' in the system

# due to external factors such as Covid-19 and the GFC. Based on the simple TS() plot we do observe
some issues which

# will require consulting the literature/external sources to validate the data points and correct
handling on NA values.

plot(ts(abs_clean_full[,-c(9,10,11)], start=c(1980,1), frequency=4))

plot(ts(abs_clean_full[,c(7,8)], start=c(1980,1), frequency=4))
```

```
# Many ML models are limited by missing observations, and missing observations==
# must be treated. We observe that X6 and X7 both contain 5 missing observations
# which should be addressed:
#                   -The missing values for X6 appear to correspond to GFC
#                   -The missing values for X7 appear to correspond to just prior to Covid-19
#
# The implications of this is that if we use mean/mediam imputation then it
# might not take into account the real impact of these events which tend to
# see a reduction in these variables, as such we should keep this in mind
# as we might need to attempt to source the missing values instead.
(na_count <- sapply(abs, function(y) sum(length(which(is.na(y))))) )



# Things to consider before going forward=====================================
# We need to do the following: - why do we see a sudden dip and spike for some data points in covid?
#                   - Some of the NA values correspond to world events, as such how should we impute to ensure
#                     we capture this? As it is a deviation from the normal.
#                   - Explore impact on dates/months/years on ML models as this is an area for improvement


# Data for GDP appears correct for Q2/Q3 2020: https://www.focus-economics.com/countries/australia/news/gdp/lifting-of-restrictions-amid-massive-fiscal-and-monetary-stimulus
# Data for job vacanices appears correct: https://www.abs.gov.au/statistics/labour/jobs/job-vacancies-australia/latest-release


# Impute missing values=======================================================
# Will add in a flag


# Manual Imputation
```

```
# Impute X7

# Obtained data from: https://www.abs.gov.au/statistics/people/population/national-state-and-territory-population/latest-release

abs_eer_x7 <- read_excel("310101.xlsx", sheet = "Data1", skip = 9) %>%

  select(X7_2 = A2133251W, # This is our EER value, based on series ID

         period = `Series ID`) # This is period due to skipping the first 9 rows


abs_impute_1 <- left_join(abs_clean_full, abs_eer_x7) %>%  # Left join as we don't want all observations from abs_eer_x7

  select(-X7) # Remove X7 as we now have X7_2 with updated figures


# Impute X6


# Use R package to impute too missing observations

abs_impute_2 <- na_interpolation(abs_impute_1)

plot(ts(abs_impute_2[,c(7,10)], start=c(1980,1), frequency=4))

ggplot_na_distribution(abs_impute_2$X6)

plot(ts(abs_impute_2[,c(7,8)], start=c(1980,1), frequency=4))
```

```
# BASELINE MODEL=================================================================
# Very simple persistence model, naive model assumes that the value of the
# variable being predicted will be the same as its most recent observed value.
# It does not take into account any other facots or varibles that could affect the
# outcome.
# Convert the date column to a time series object
unemployment_ts <- ts(abs_impute_2$Y, start = c(1981, 2), frequency = 4)

# Split the data into training and testing sets
train <- window(unemployment_ts, end = c(2017, 4))
test <- window(unemployment_ts, start = c(2018, 1))

# Create a persistence model
persistence_model <- tail(train, 1)

# Generate predictions for the test set
predictions <- rep(persistence_model, length(test))

# Calculate the mean squared error
mse <- mean((test - predictions)^2)

# Print the MSE
sqrt(mse)
```

```
# Data exploitation on whole data set=========================================

# Correlation and multicolineariy testing

# Assuming your data frame is named 'df' and the response variable is named 'Y'

par(mfrow=c(3,3))  # Set the plot layout to a 3x3 grid

for (i in 2:10) {  # Assuming you have 8 predictor variables in your data frame

  plot(abs_impute_2[,i], abs_impute_2$Y, main = paste0("Scatterplot of Y vs ",
colnames(abs_impute_2)[i]))

}


# Outliers

abs_impute_2_numeric <- abs_impute_2 %>%

  select(-period, -month, -year) # Remove the non-numeric varibles


# Histograme of all inputs

hist.data.frame(abs_impute_2_numeric)


# Boxplots of all inputs

# Melt the data into long format

library(reshape2)

abs_impute_2_melt <- melt(abs_impute_2_numeric)


# Load required packages
```

```
library(ggplot2)
library(ggpubr)

# Create the boxplot with ggplot
ggplot(abs_impute_2_melt, aes(x = variable, y = value)) +
  geom_boxplot() +
  facet_wrap(~variable, scales = "free_y") +
  xlab("") +
  theme(axis.text.x = element_blank())

# Summary stats
summary(abs_impute_2_numeric)
sd_table <- abs_impute_2_numeric %>% # SD value
  summarise_at(vars(Y:X7_2), sd)

# Split data into test/train==================================================
# At this stage we split into test/train and did not scale as this could depend
# on which model we select.
```

```
# Further manipulations can be applied to test/train at a later stage depending
# on specific model
#
# We decdide to train on the full data set at this stage



# Whilst not shown here, we see improvement in RMSE validation if we lag variables
# MARS Still best model based on lag. This is included in disccusion as part of
# ways to improve the model to account for time effect.

# If we want to lag varibiles & tune
#abs_impute_2$X1_lag <- lag(abs_impute_2$X1)
#abs_impute_2$X2_lag <- lag(abs_impute_2$X2)
#abs_impute_2$X3_lag <- lag(abs_impute_2$X3)
#abs_impute_2$X4_lag <- lag(abs_impute_2$X4)
#abs_impute_2$X5_lag <- lag(abs_impute_2$X5)
#abs_impute_2$X6_lag <- lag(abs_impute_2$X6)
#abs_impute_2$X7_lag <- lag(abs_impute_2$X7_2)
#abs_impute_2$month_lag <- lag(abs_impute_2$month)
#abs_impute_2 <- abs_impute_2[-1,]
#abs_impute_2 <- abs_impute_2 %>%
#  select(-X1, -X2, -X3, -X4, -X5, -X6, -X7_2, -month)



train_abs <- abs_impute_2 %>%
  filter(period < "2018-03-01") %>%
  select(-period)
```

```
dim(train_abs) # 147, 10

test_abs <- abs_impute_2 %>%
  filter(period >= "2018-03-01") %>%
  select(-period)



dim(test_abs) # 11, 10
```

```
# Machine learning model============================================================

# As dealing with a regresion problem wich varies in space and time

# we elected to utilize a regression tree model, which we discussed in wk3.

# Possible models include:

# - Basic regression tree

# - PRIM-Bump hunting

# - Multivariate Adaptive Regresion Spline: This did quite well - was mentioned in R demonstration wk3

# - Bagging

# - Random Forest: This did poorly

# - Boosted trees: EXPLORE


# We decide to compare the following models
```

```
# Random forest

# MARS

# Boosting

# Comment on residual plots

# Comment on test/train MSE

# Out of box model: random forest model=======================================

library(randomForest)

set.seed(123)




start_time <- Sys.time()

# Fit a random forest model to training data

random_forest_oobm <- randomForest(Y ~., data = train_abs)

(end_time_rf <-Sys.time() - start_time ) # 0.03983212 secs

# Calculate the predicted values on the training data

yhat_train_oobm <- predict(random_forest_oobm, newdata = train_abs )

# Calculate the residuals on the training data

residuals_rf_oobm <- train_abs$Y - yhat_train_oobm

plot(train_abs$Y,  residuals_rf_oobm) # Plot resdiuals of rf oobm

qqnorm(residuals_rf_oobm)

qqline(residuals_rf_oobm)

# Calculate the training MSE

(mse_train_oobm <- mean(residuals_rf_oobm^2)) #0.02870791

(sqrt(mse_train_oobm))# 0.1694341
```

```
# Calculate the predicted values on the test data
yhat_test_oobm <- predict(random_forest_oobm, newdata = test_abs)

# Calculate the residuals on the training data
residuals_rf_oobm_test <- test_abs$Y - yhat_test_oobm
plot(test_abs$Y, residuals_rf_oobm_test) # Plot resdiuals of rf oobm

# Calculate the training MSE
(mse_test_oobm <- mean(residuals_rf_oobm_test^2) ) #0.6030101
(sqrt(mse_test_oobm)) # RMSE  0.7765373

# Out of box model: MARS=====================================================
library(earth)
set.seed(123)

# Fit a MARS model to training data
start_time <- Sys.time()
mars_oobm <- earth(Y ~., data = train_abs )
(end_time_mars <-Sys.time() - start_time )

# Calculate the predicted values on the training data
yhat_train_mars_oobm <- predict(mars_oobm, newdata = train_abs )

# Calculate the residuals on the training data
residuals_mars_oobm <- train_abs$Y - yhat_train_mars_oobm
plot(train_abs$Y,residuals_mars_oobm) # Much disperse then RF
qqnorm(residuals_mars_oobm)
qqline(residuals_mars_oobm)

# Calculate the training MSE
(mse_train_oobm <- mean(residuals_mars_oobm^2))  # 0.04736039
```

```
(sqrt(mse_train_oobm))  #0.4401467

# Calculate the predicted values on the test data
yhat_test_mars_oobm <- predict(mars_oobm, newdata = test_abs  )

# Calculate the residuals on the training data
residuals_mars_oobm_test <- test_abs$Y - yhat_test_mars_oobm
plot(test_abs$Y, residuals_mars_oobm_test) # Plot resdiuals of rf oobm

# Calculate the training MSE
(mse_test_oobm <- mean(residuals_mars_oobm_test^2) ) #0.4132105
(sqrt(mse_test_oobm)) # RMSE 0.6428145

# Out of box model: Boosting==================================================
library(gbm)
set.seed(123)

# Fit a Boosting model to training data
start_time <- Sys.time()
# Fit a random forest model to training data
boost_oobm <- gbm(Y ~., data = train_abs )
(end_time_boost <-Sys.time() - start_time ) # 0.006984949 secs


# Calculate the predicted values on the training data
yhat_train_boost_oobm <- predict(boost_oobm, newdata = train_abs )

# Calculate the residuals on the training data
residuals_boost_oobm <- train_abs$Y - yhat_train_boost_oobm
plot(train_abs$Y, residuals_boost_oobm) # Plot residuals of boosting oobm
qqnorm(residuals_boost_oobm)
```

```
qqline(residuals_boost_oobm)

# Calculate the training MSE
(mse_train_boost_oobm <- mean(residuals_boost_oobm^2)) # 0.08096076
(sqrt(mse_train_boost_oobm)) # RMSE 0.284536

# Calculate the predicted values on the test data
yhat_test_boost_oobm <- predict(boost_oobm, newdata = test_abs   )

# Calculate the residuals on the test data
residuals_boost_oobm_test <- test_abs$Y - yhat_test_boost_oobm
plot(test_abs$Y,residuals_boost_oobm_test) # Plot residuals of boosting oobm

# Calculate the test MSE
(mse_test_boost_oobm <- mean(residuals_boost_oobm_test^2)) #1.723489
(sqrt(mse_test_boost_oobm)) # RMSE




# We will now optimize our MARS MODEL========================================
# What is a MARS model? How can we describe it itnutivly? We can
# plot it out and put some points around it:



# Fit a MARS model with nprune = 1
# Will produce linear model which minimizes loss function
mars_model_1 <- earth(Y ~ X7_2, data = train_abs, nprune = 1)
summary(mars_model_1) # 1 of 7 terms, only intercept

# Fit a MARS model with nprune = 2
```

```
# Will produce linear model which minimizes loss function
mars_model_2 <- earth(Y ~ X7_2, data = train_abs, nprune = 2)
summary(mars_model_2) # 2 of 7 terms, only intercept/1hinge function



# Fit a MARS model with nprune = 3
# Will produce linear model which minimizes loss function
mars_model_3 <- earth(Y ~ X7_2, data = train_abs, nprune = 3)
summary(mars_model_3) # 3 of 7 terms, only intercept/2xhinge function

# This plot will demo how it looks visually
# Create a scatter plot of the data
plot(train_abs$X7_2, train_abs$Y, pch=16, xlab="X7", ylab="Y")

# Add lines to the plot for the MARS models
x_vals <- seq(min(train_abs$X7_2), max(train_abs$X7_2), length.out=100)
lines(x_vals, predict(mars_model_1, newdata=data.frame(X7_2=x_vals)), col="red", lwd=2)
lines(x_vals, predict(mars_model_2, newdata=data.frame(X7_2=x_vals)), col="blue", lwd=2)
lines(x_vals, predict(mars_model_3, newdata=data.frame(X7_2=x_vals)), col="green", lwd=2)

# Add a legend to the plot
legend("topright", legend=c("nprune=1", "nprune=2", "nprune=3"),
       col=c("red", "blue", "green"), lty=1)



# TUNE MARS MODEL=================================================================
# Tuning: We have two parameters, the degree of interactions and the
# number of retained terms, we will undertake a grid search to find the optimal
# number of cominations of these hyperparamters that minmise the prediction error

library(caret)
```

```
set.seed(123)
start_time <- Sys.time()
# create a tuning grid
hyper_grid <- expand.grid(
  degree = 1:3,
  nprune = seq(2, 100, length.out = 10) %>% floor()
)
tuned_mars <- train(
  x = subset(train_abs, select = -Y),
  y = train_abs$Y,
  method = "earth",
  metric = "RMSE",
  trControl = trainControl(method = "repeatedcv", number = 10, repeats = 3),
  tuneGrid = hyper_grid
)

end_time <- Sys.time() - start_time # Time difference of 38 seconds
plot(tuned_mars)
tuned_mars$bestTune # nprune = 23, degree = 1\
tuned_mars$results %>%
  filter(nprune == tuned_mars$bestTune$nprune, degree == tuned_mars$bestTune$degree)


# Can we further improve our model by tweaking nprune, we set degree as 1
set.seed(123)
start_time <- Sys.time()
# create a tuning grid
hyper_grid_2 <- expand.grid(
  degree = 1:3,
  nprune = 2:23 %>% floor()
```

```
)
tuned_mars_2 <- train(
  x = subset(train_abs, select = -Y),
  y = train_abs$Y,
  method = "earth",
  metric = "RMSE",
  trControl = trainControl(method = "repeatedcv", number = 10, repeats = 3),
  tuneGrid = hyper_grid_2
)

end_time <- Sys.time() - start_time # Time difference of 23 seconds
plot(tuned_mars_2)
tuned_mars_2$bestTune # nprune = 20, degree = 1\
tuned_mars_2$results %>%
  filter(nprune == tuned_mars_2$bestTune$nprune, degree == tuned_mars_2$bestTune$degree) # CV RMSE 0.3252662
varImp(tuned_mars_2)

# Extract some input to interpet model
set.seed(123)
tuned_mars_2 <- earth(Y ~., data = train_abs, nprune = 20, degree = 1 )
#tuned_mars_2 <- earth(Y ~., data = train_abs, nprune = 11, degree = 1 ) # Without year in model
summary(tuned_mars_2) # Provide summary/coefiencets of model

# Calculate the predicted values on the training data
yhat_train_mars <- predict(tuned_mars_2, newdata = train_abs)

# Calculate the residuals on the training data
residuals_mars_train <- train_abs$Y - yhat_train_mars
par(mfrow = c(1,2))
plot(train_abs$Y, residuals_mars_train) # Plot residuals of boosting oobm
```

```
qqnorm(residuals_mars_train)
qqline(residuals_mars_train)

# Calculate the training MSE
(mse_train_mars <- mean(residuals_mars_train^2))
(sqrt(mse_train_mars))

# Calculate the predicted values on the test data
yhat_test_mars <- predict(tuned_mars_2, newdata = test_abs)


# Calculate the residuals on the test data
residuals_mars_test <- test_abs$Y - yhat_test_mars
par(mfrow = c(1,2))
plot(test_abs$Y, residuals_mars_test) # Plot residuals of boosting oobm
qqnorm(residuals_mars_test)
qqline(residuals_mars_test)

# Calculate the test MSE
(mse_test_mars <- mean(residuals_mars_test^2)) # 0.1783143, same as baseline actually
(sqrt(mse_test_mars))

# Now lets plot everything, and put it on a graph and see how Y compares yhat==
# Create data frame of know, predicted agaisnt time
predict_all <- as.numeric(predict(tuned_mars_2, newdata = abs_impute_2))
model_plot <- abs_impute_2 %>%
  select(period, Y) %>%
  mutate(period = as.Date(period))
model_plot <- bind_cols(model_plot, Y_hat = predict_all)
```

```
# Plot, highlight train/test data with solid line
ggplot(model_plot, aes(x = period)) +
  geom_line(aes(y = Y, color = "Y")) +
  geom_line(aes(y = Y_hat, color = "Yhat")) +
  scale_color_manual(values = c("Y" = "blue", "Yhat" = "red")) +
  labs(x = "period", y = "Value") +
  geom_vline(xintercept = as.Date("2018-03-01"), linetype="solid",
             color = "black", size=1.5) +
  ggtitle("Predictive Performance of MARS Model")




# Cross validation accuracy MARS================================================
# create a tuning grid

# Insert our tuned hyper-parameters
set.seed(123)
hyper_grid <- expand.grid(
  degree = 1,
  nprune = 20
)
cv_mars_model <- train(
  x = subset(train_abs, select = -Y),
  y = train_abs$Y,
  method = "earth",
  metric = "RMSE",
  trControl = trainControl(method = "repeatedcv", number = 10, repeats = 3),
  tuneGrid = hyper_grid
```

```
)
print(cv_mars_model) # 0.3838806
```

```
# ANN Model=========================================================================

# Data prep
library(caret)

# ANN/Deep learning requires data to be scaled:
# We do not want to scale month/year/Y we will hot encode these
cols_to_scale <- c("X1", "X2", "X3", "X4", "X5" ,"X6", "X7_2")
#cols_to_scale <- c("X1_lag", "X2_lag", "X3_lag", "X4_lag", "X5_lag" ,"X6_lag", "X7_lag") # If we lag
cols_not_to_scale <- c("Y", "month" , "year")
#cols_not_to_scale <- c("Y", "month_lag" ) #, "year") # IF we lag

# Calculate the mean and standard deviation of the numerical variables in the training set
```

```
train_mean <- apply(train_abs[, cols_to_scale], 2, mean)

train_sd <- apply(train_abs[, cols_to_scale], 2, sd)


# Scale the numerical variables in the training set using the mean and standard deviation from the training set

train_abs_ann_scaled <- scale(train_abs[, cols_to_scale], center = train_mean, scale = train_sd)


# Scale the numerical variables in the test set using the mean and standard deviation from the training set

test_abs_ann_scaled<- scale(test_abs[, cols_to_scale], center = train_mean, scale = train_sd)


# Add back columns non scaled data

train_abs_ann_scaled <- cbind(train_abs_ann_scaled[, cols_to_scale], train_abs[, cols_not_to_scale]) #%>% select(-year)

test_abs_ann_scaled <- cbind(test_abs_ann_scaled[, 1:7], test_abs[, cols_not_to_scale]) # %>% select(-year)



dummy_test <- dummyVars( " ~ .", data = test_abs_ann_scaled)

test_abs_ann_scaled <- data.frame(predict(dummy_test, newdata = test_abs_ann_scaled))


dummy_train <- dummyVars(" ~ .", data = train_abs_ann_scaled)

train_abs_ann_scaled <- data.frame(predict(dummy_train, newdata = train_abs_ann_scaled))



# https://www.r-bloggers.com/2015/09/fitting-a-neural-network-in-r-neuralnet-package/


# We will compare 3 simple neural networks, 1 hidden layer, 2 hidden layer and 3 hidden layer NB: Unable to do 3 due to taking 24+ hours

# we will select the most optimized one via a grid search approach which loops

# 1:50 based on the number of varibles. This should answer all the requred points

# of the NN question. We have variations in number of nodes, as well as variation
```

```
# in number of hidden networks.
# We use set seed for reproducatiblity.



# Two hidden layer model: Varying the number of neurons in each layer
start_time <- Sys.time()
set.seed(123)
loop <- 1:100 # 1:50 What if we vary the number of neurons?
loop2 <- 1:100 # 1:50
mse1 <- data.frame(Layer_1_neuron = numeric(),
                   Layer_2_neuron = numeric(),
                   Test_MSE = numeric())
formula <- as.formula(paste("Y ~", paste(colnames(train_abs_ann_scaled[-8]), collapse = " + ")))
for (i in loop){
  for (j in loop2){
    set.seed(123)
    tryCatch({
      nn <- neuralnet(formula, data=train_abs_ann_scaled, hidden=c(i, j))
      yhat <- predict(nn, test_abs_ann_scaled)
      res <- yhat - test_abs_ann_scaled$Y
      mse <- mean(res^2)

      yhattrain <- predict(nn, train_abs_ann_scaled)
      restrain <- yhattrain - train_abs_ann_scaled$Y
      msetrain <- mean(restrain^2)
      output <- data.frame(Layer_1_neuron = i,
                           Layer_2_neuron = j,
                           Test_MSE = mse,
                           Train_MSE = msetrain)
      mse1 <- rbind(mse1, output)
      print(paste0("N1 ", i, " N2 ", j, " Test MSE ", mse, " Train MSE ", msetrain))
```

```
   }, error = function(e){
     print(paste0("Error: ", e$message)) # We anticipate some issues 0
   })
  }
}
end_time <- Sys.time() - start_time # 30 mins

# Find row with the lowest Test_MSE value
min_row <- which.min(mse1$Test_MSE)

# Extract the corresponding values of Layer_1_neuron, Layer_2_neuron, and Test_MSE
(best_result <- mse1[min_row, ]) # L1: 49, L2: 18, Test MSE 0.122355

# NB: Remove year, best result is L1, L2, Test MSE




# One hidden layer model
start_time <- Sys.time()
set.seed(123)
loop <- 1:100 #ncol(train_abs_ann_scaled) - 1
mse2 <- data.frame(Layer_1_neuron = numeric(),
                   Test_MSE = numeric())
formula <- as.formula(paste("Y ~", paste(colnames(train_abs_ann_scaled[-8]), collapse = " + ")))
for (i in loop){
  set.seed(123)
  tryCatch({
    nn <- neuralnet(formula, data=train_abs_ann_scaled, hidden=c(i))
    yhat <- predict(nn, test_abs_ann_scaled)
    res <- yhat - test_abs_ann_scaled$Y
    mse <- mean(res^2)
```

```
    yhattrain <- predict(nn, train_abs_ann_scaled)
    restrain <- yhattrain - train_abs_ann_scaled$Y
    msetrain <- mean(restrain^2)
    output <- data.frame(Layer_1_neuron = i,
                         Test_MSE = mse,
                         Train_MSE = msetrain)
    print(paste0("N1 ", i, ", Train MSE ", mse, " Test MSE ", msetrain))
    mse2 <- rbind(mse2, output)
  }, error = function(e){
    print(paste0("Error: ", e$message)) # We anticipate some issues 0
  })
}
(end_time <- Sys.time() - start_time) # 9 seconds
# Find row with the lowest Test_MSE value
min_row <- which.min(mse2$Test_MSE)

# Extract the corresponding values of Layer_1_neuron, Layer_2_neuron, and Test_MSE
(best_result <- mse2[min_row, ] )# L1: 17, Test MSE 0.9740996



# We will procced with analysis on 2 hidden layer NN with======================
set.seed(123)
formula <- as.formula(paste("Y ~", paste(colnames(train_abs_ann_scaled[-8]), collapse = " + ")))
nn_two_final <- neuralnet(formula, data=train_abs_ann_scaled, hidden=c(49, 18))
#nn_model <- neuralnet(formula, data = train_abs_ann_scaled, hidden = c(33, 96)) # Optimised 2
hidden layer without year

# Plot residuals train data
yhat_train <- predict(nn_two_final, newdata = train_abs_ann_scaled)
res_train <- yhat_train - train_abs_ann_scaled$Y
par(mfrow = c(1,2))
```

```
plot(train_abs_ann_scaled$Y , res_train)
qqnorm(res_train)
qqline(res_train)
(mse_train_nn <- mean(res_train^2))
(sqrt(mse_train_nn))

# Plot residuals test data
yhat_test <- predict(nn_two_final, newdata = test_abs_ann_scaled)
res_test <- yhat_test - test_abs_ann_scaled$Y
par(mfrow = c(1,2))
plot(test_abs_ann_scaled$Y , res_test)
qqnorm(res_test)
qqline(res_test)
(mse_test_nn <- mean(res_test^2))
(sqrt(mse_test_nn))




# Cross validation NN based on above==========================================
# We observe large RMSE with all input varibles including year. If we remove year
# then we hyptohisis the model will be less prone to overfitting. As such
# when we removed year we obtained an improved CV RMSE
k <- 10

# Split the data into k equal-sized folds
fold_indices <- cut(seq(1, nrow(train_abs_ann_scaled)), breaks = k, labels = FALSE)

# Initialize a list to store the cross-validation results
cv_results <- list()
```

```
# Loop through each fold and train the model
for (i in 1:k) {
  formula <- as.formula(paste("Y ~", paste(colnames(train_abs_ann_scaled[-8]), collapse = " + ")))
  set.seed(123)
  # Get the indices of the current fold
  test_indices <- which(fold_indices == i)
  train_indices <- which(fold_indices != i)

  # Extract the training and test data for the current fold
  train_data <- train_abs_ann_scaled[train_indices, ]
  test_data <- train_abs_ann_scaled[test_indices, ]

  # Train the model on the training data
  #nn_model <- neuralnet(formula, data = train_data, hidden = c(17)) # Optimised 1 hidden layer
  nn_model <- neuralnet(formula, data = train_data, hidden = c(49, 18)) # Optimised 2 hidden layer with 51 neuron search
  #nn_model <- neuralnet(formula, data = train_data, hidden = c(4, 85)) # Optimised 2 hidden layer with 100 neuron search



  # Make predictions on the test data
  yhat <- predict(nn_model, test_data)

  residuals <- yhat - test_data$Y
  mse <- mean(residuals^2)
  rmse <- sqrt(mse)

  # Add the cross-validation metric to the results list
  cv_results[[i]] <- rmse
}
```

```
# Compute the average cross-validation metric across all folds
avg_cv_metric <- mean(unlist(cv_results))

# Print the cross-validation results
print(paste0("Average cross-validation metric: ", avg_cv_metric))



# End code: NOTE: We have eddited the code to optimise different variations
# such as removing year, lagging etc. We have hashed (#) them out as needed/not
# to compare and contrast some points.
```

References

ABS (2022) HOW THE ABS MEASURES UNEMPLOYMENT, Australian Bureau of Statistics. Available at: https://www.abs.gov.au/ausstats/abs@.nsf/Lookup/by%20Subject/6102.0.55.001~Feb%202018~Main%20Features~4.%20How%20the%20ABS%20Measures%20Unemployment~50#:~:text=ABS%20measures%20unemployment%20by%20collecting,and%20indigenous%20communities%20throughout%20Australia (Accessed: February 14, 2023)

Cansiz, S. (2020) The math behind training neural networks, Medium. Towards Data Science. Available at: https://towardsdatascience.com/adventure-of-the-neurons-theory-behind-the-neural-networks-5d19c594ca16 (Accessed: February 28, 2023).

Higgens, P. (no date) GDP growth, the unemployment rate, and Okun's Law. Available at: https://www.atlantafed.org/-/media/Documents/regional-economy/econsouth/11q3fedatissue.pdf (Accessed: January 1, 2011).

Job vacancies, Australia, November 2022 (no date) Australian Bureau of Statistics. Available at: https://www.abs.gov.au/statistics/labour/jobs/job-vacancies-australia/latest-release (Accessed: February 22, 2023).

Katris, C. (2019) "Prediction of unemployment rates with time series and Machine Learning Techniques," Computational Economics, 55(2), pp. 673–706. Available at: https://doi.org/10.1007/s10614-019-09908-9.

Kumar, S. (2022) 4 techniques to handle missing values in time series data, Medium. Towards Data Science. Available at: https://towardsdatascience.com/4-techniques-to-handle-missing-values-in-time-series-data-c3568589b5a8 (Accessed: February 17, 2023).

LEIGH, A.N.D.R.E.W. and MCLEISH, M.A.R.K. (2009) "Are state elections affected by the National Economy? Evidence from Australia," Economic Record, 85(269), pp. 210–222. Available at: https://doi.org/10.1111/j.1475-4932.2009.00549.x.

Lu, C.-J., Lee, T.-S. and Lian, C.-M. (2012) "Sales forecasting for computer wholesalers: A comparison of multivariate adaptive regression splines and artificial neural networks," Decision Support Systems, 54(1), pp. 584–596. Available at: https://doi.org/10.1016/j.dss.2012.08.006.

Massimo, Massimo and FocusEconomics (2022) Lifting of restrictions amid massive fiscal and monetary stimulus prompt rebound in Q3, FocusEconomics. Available at: https://www.focus-economics.com/countries/australia/news/gdp/australia-lifting-of-restrictions-amid-massive-fiscal-and-monetary-stimulus (Accessed: February 22, 2023).

M, J. (1957) How to choose the number of hidden layers and nodes in a feedforward neural network? Available at: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw (Accessed: February 28, 2023).

Mic (2015) Selecting the number of neurons in the hidden layer of a neural network: R-bloggers, R. Available at: https://www.r-bloggers.com/2015/09/selecting-the-number-of-neurons-in-the-hidden-layer-of-a-neural-network/#:~:text=The%20rules%20of%20thumb,number%20is%20greater%20than%201). (Accessed: February 28, 2023).

National, state and territory population, June 2022 (no date) Australian Bureau of Statistics. Available at: https://www.abs.gov.au/statistics/people/population/national-state-and-territory-population/latest-release (Accessed: February 22, 2023)

Nair, A. (2022) Baseline models: Your guide for model building, Medium. Towards Data Science. Available at: https://towardsdatascience.com/baseline-models-your-guide-for-model-building-1ec3aa244b8d (Accessed: February 20, 2023)

Owen, E. (2022) Two Years on: Six ways covid-19 has shaped the housing market, CoreLogic Australia. Available at: https://www.corelogic.com.au/news-research/news/2022/two-years-on-six-ways-covid-19-has-shaped-the-housing-market#:~:text=Despite%20an%20initial%20dip%2C%20housing,the%20onset%20of%20COVID%2D19 (Accessed: February 27, 2023).

Pettinger, T. (no date) Seasonal unemployment, Economics Help. Available at: https://www.economicshelp.org/blog/glossary/seasonal-unemployment/ (Accessed: February 17, 2023).

RBA (2010) The labour market during the 2008–2009 downturn: Bulletin – March 2010, Reserve Bank of Australia. Reserve Bank of Australia. Available at: https://www.rba.gov.au/publications/bulletin/2010/mar/1.html (Accessed: February 17, 2023).

RBA (2011) The Australian labour market in the 2000s: The quiet decade: Conference – 2011, Reserve Bank of Australia. Reserve Bank of Australia. Available at: https://www.rba.gov.au/publications/confs/2011/borland.html (Accessed: February 17, 2023).

Sachdev, H.S. (no date) Choosing number of hidden layers and number of hidden neurons in neural networks, LinkedIn. Available at: https://www.linkedin.com/pulse/choosing-number-hidden-layers-neurons-neural-networks-sachdev/ (Accessed: February 28, 2023).

Surakhi, O. et al. (2021) “Time-lag selection for time-series forecasting using neural network and heuristic algorithm,” Electronics, 10(20), p. 2518. Available at: https://doi.org/10.3390/electronics10202518.

Visentin, L., Wright, S. and Curtis, K. (2022) 'I made a mistake': Albanese stumbles on unemployment rate and cash rate, The Sydney Morning Herald. The Sydney Morning Herald. Available at: https://www.smh.com.au/politics/federal/i-m-not-sure-what-it-is-anthony-albanese-stumbles-on-unemployment-rate-and-cash-rate-20220411-p5aci2.html (Accessed: FCockburn, G., Jervis-Bardy, D. and Dennett, H. (2022) 'not just luck': Frydenberg Cheers Steady 4 per cent unemployment rate, The Canberra Times. The Canberra Times. Available at: https://www.canberratimes.com.au/story/7699832/not-just-luck-frydenberg-cheers-steady-4-per-cent-unemployment-rate/ (Accessed: February 17, 2023). ebruary 17, 2023).