Forecasting the Australian Unemployment Rate Using a Gradient Boosted Machine and a Deep Neural Network

Nikki Fitzherbert 13848336

17 June 2020

Abstract

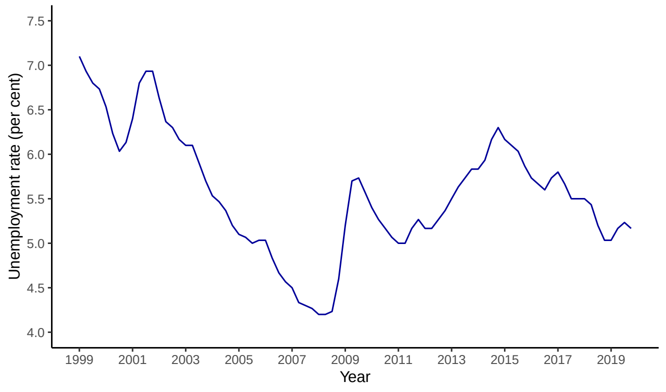
The unemployment rate is one of the most commonly used indicators for understanding labour market conditions in Australia and can provide an important gauge of the level of spare capacity in the economy. Since 1981, some of the primary influences on the unemployment rate have included the 1990s recession, the global financial crisis in 2007-08, the energy and resources boom that ran from about 2003 to mid-2011, and the aging demographic profile of Australia. This analysis transformed the data into a supervised learning problem and modelled the seasonally-adjusted Australian unemployment rate from 1981 to the end of 2014 with a gradient boosted machine (GBM) and a deep neural network, before forecasting the unemployment rate for the next four years. The final results indicated that the GBM did much better at the task than the neural network. Perhaps the most significant improvement to the performance of the neural network could come from using a long short-term memory architecture instead of a standard feed-forward multi-layer perceptron model.

Overview of the Australian unemployment rate between 1999 and 2019

The unemployment rate (UR)[1] is one of the most commonly used indicators for understanding labour market conditions and can provide an important gauge of the level of spare capacity. As with any labour market indicator, the movement of the UR over time generally reflects the interaction between supply and demand in the economy, and between 1999 and 2019, two of the biggest influences on labour market dynamics were one of the largest resources booms in Australian history and demographics.

Up until mid-2008, the UR[2] generally followed a downward trend as the Australian economy recovered from a major economic recession in the early 1990s and the start of a resources boom in about 2003 that would continue for the next 15 years[3] (Figure 1).

Figure 1: Unemployment rate, quarterly seasonally−adjusted estimates, 1999 to 2019



Source: Australian Bureau of Statistics

The first phase of the resources boom saw a rapid rise in Australia's terms of trade along with an appreciation of the nominal exchange rate and a redirection of resources and labour into the mining and mining-related sectors of the economy (Bishop, Kent, Plump and Rayner, 2013). By the September quarter of 2008, the UR had fallen to a low of 4.2 per cent. The decline in the UR abruptly but nevertheless temporarily reversed direction in mid-2008 with the onset of the global financial crisis until mid-2011 when Australia's terms of trade peaked and mining-related investment started to slow (OCE, 2014).

For the next three-and-a-half years, the UR exhibited an upward trend; primarily due to subdued economic activity on the demand side[4] and an ageing population on the supply side. Despite overall economic conditions continuing to remain subdued in 2015 and subsequent years, the UR peaked at 6.3 per cent in the final quarter of 2014 and generally declined through to the end of 2019. This appeared to reflect a combination of wage restraint, which reduced potential job losses, rapidly growing labour-intensive sectors such as Healthcare and Social Assistance, and a continued influx of skilled migrants into labour force (OCE, 2015).

[1]The proportion of the Australian labour force that is unemployed, where an unemployed person is someone of working age who does not have a paid job but is actively looking for work.

[2]For the rest of this paper, unless otherwise specified the UR is in seasonally-adjusted terms.

[3]The completion of the final three liquid natural gas projects - Wheatstone, Ichthys and Prelude - in 2018 largely marked the end of the resources and energy investment boom.

[4]The high nominal exchange rate caused by the resources boom was hurting trade-exposed sectors of the economy and encouraging consumers to import products from overseas rather than purchasing domestically. Furthermore, economic activity in non-resource-related sectors of the economy had failed to pick up as expected as resource-related investment slowed.

Data

The data for this analysis used the seasonally-adjusted UR as the outcome variable and seven economic indicators as the primary features, which included Gross Domestic Product (GDP), Final Consumption Expenditure (FCE), the Terms of Trade (ToT), the Consumer Price Index (CPI), job vacancies and the overall population. Further details about these variables can be found in Appendix B.

	Y	X1	X2	X3	X4	X5	X6	X7
NAs	0.00	0.00	0.00	0.00	0.00	0.00	5.00	2.00
Minimum	4.20	-1.20	-1.00	-0.10	-7.00	28.40	28.40	14923.30
Maximum	11.10	2.30	2.00	1.70	7.60	116.20	242.80	25364.30
Mean	6.86	0.76	0.81	0.79	0.34	74.29	110.47	19485.13
Median	6.20	0.70	0.80	0.70	0.20	72.90	97.15	19028.80
Stdev	1.80	0.52	0.61	0.36	2.46	24.60	58.00	2933.98

Prior to training either model, the data underwent a couple of pre-processing steps:

1. There were a small number of values missing in $X6$ and $X7$. These were imputed using the median of each series in the training set and inserted where applicable into the test set.
2. The data was subset into a training and test set. Due to the time-component, it was inappropriate to randomly split the data. Instead, the training set included all data up to the end of 2014, and the test set consisted of the data from 2015 to 2019.
3. Many machine learning methods are unable to handle time series data, so the dataset was transformed into a form more closely representing a supervised learning problem. This was accomplished by:
   a) Differencing Y, $X5$, $X6$ and $X7$ to remove the presence of the stochastic/deterministic trends apparent in the data (Appendix C)[5].
   b) Initially including a single lag for each feature variable as well as three lags for the outcome variable[6]. The number of lags was kept small based on the theory that the unemployment rate would only depend on recent economic history and to avoid an overparameterised model. The decision on whether to keep those lags would be made during model training.
   c) Extracting the year and quarter from the time stamp and including those in the model.
4. Creating a second normalised version of the data using the min-max method. This was necessary for the neural network as they are sensitive to features on different scales, but not necessary for tree-based models.

[5]For example, tree-based models are unable to extrapolate if the data includes a trend component. This is because these algorithms recursively subset a feature space using binary splitting rules and are therefore unable to make sensible predictions if the values of either the outcome or feature variables in the test set are outside the range encountered in the training set. Differencing also helps reduce the chance of spurious correlation influencing the results.

[6]The additional set of features created excluded the first lag of the first-differenced outcome variable in order to force each model to look for patterns in the data instead of relying on the strong autocorrelation between Y_t and Y_{t-1}.

Application of a Gradient Boosted Machine

Algorithm choice

Three of the most popular supervised machine learning techniques with demonstrated high predictive accuracy are supper vector machines (SVMs), random forests and gradient boosted machines (GBMs). SVMs are a kernel-based method originally developed in the 1990s and based on the idea that data can be separated into two or more classes using a optimal separating hyperplane. The specification of a kernel increases the dimensionality of the enlarged feature space such that a non-linear decision boundary is possible. However, SVMs can be more computationally intensive than tree-based algorithms, require more data pre-processing prior to model training (such as data normalisation and dealing with missing values) and are even less interpretable.

Random forests and GBMs are tree-based ensemble methods. One of the biggest advantages of a tree-based approach over a SVM is the lack of assumptions about the distribution of the data[7]. Random forests build multiple deep decorrelated trees by only allowing the algorithm to consider a subset of all available features at each split, whilst GBMs are based on the idea of a weak learner. That is, each tree is grown sequentially using the residuals from the previous tree to slowly improve the accuracy of the model. GBMs can be prone to overfitting if too many trees are used, but this is controlled through the use of cross-validation in the training process.

For this analysis, a GBM was chosen over a SVM primarily for model interpretability reasons; that is, so the relative importance of individual features could be assessed without having to take into account the underlying data distribution. It was chosen over a random forest to take advantage of the sequential approach to tree building, which reduces both bias and variance in the model, whereas random forests generally work on reducing model variance.

Algorithm hyperparameters

GBMs require careful tuning during model training in order to optimise their predictive power. If done appropriately, the result makes them one of the hardest algorithms to beat in many circumstances. There are five main hyperparameters that need to be specified and tuned prior to testing the model on a separate dataset (Boehmke & Greenwell, 2020):

- Number of trees/boosting iterations ($n.trees$) - As previously identified, if this is too large then overfitting will occur as the algorithm will continue to chase the residuals until it is told to stop. The final value was chosen via five-fold cross-validation during the tuning process.
- Learning rate ($shrinkage$) - determines the contribution of each tree to the final outcome and controls how quickly the algorithm learns. Smaller values make the model more robust and increases its ability to generalise, but requires more trees. Typical values range between 0.001 and 0.3.
- Tree Depth ($interaction.depth$) - Controls the depth of individual trees. Smaller values are computationally efficient but require more trees, and higher values allow the model to capture unique patterns in the data but increase the risk of overfitting. Typical values range between three to eight, but it is not uncommon to see a value of one.

[7] An implicit assumption about data structure is made in SVMs through the kernel choice.

- Minimum number of observations in terminal nodes ($n.minobsinnode$) - Controls the complexity of each tree. Typical values range from five to 15, where higher values can help prevent overfitting.
- Subsample of training observations chosen to build the next tree ($bag.fraction$) - Introduces randomness into the algorithm and trains a stochastic GBM instead of a basic GBM. Typical values range between 0.5 and 0.8.

Training set performance and interpretation

Three different configurations of a GBM model were trained:

1. The first-differences of Y, $X5$, $X6$ and $X7$ plus extracted year and quarter information from the timestamp.
2. Model one plus three lags of the differenced outcome variable.
3. Model two model plus a single lag of each feature.

Each model was assessed relative to each other on three performance metrics: the mean squared error (MSE)[8], mean absolute error (MAE) and mean absolute percentage error (MAPE), and against the results from a naive persistence model. The naive model provided a measure of baseline performance for the GBM algorithm and simply used the value at the current time step as the predicted outcome for the next time step.

The first table below shows the summary performance metrics for the optimal GBM for each specified configuration of features and the performance of the naive model. It clearly indicates that both the first and third configurations outperformed the naive model across all three metrics, but the second configuration performed much worse. Indeed, a plot of the predicted values against the training data showed that it consistently overestimated the values of the UR.

model	MSE	MAE	MAPE	train_time
model 1	0.0178	0.1078	1.6525	155.58 secs
model 2	0.3308	0.5278	8.0658	157.40 secs
model 3	0.0609	0.1963	3.0100	158.05 secs
naive	0.0880	0.2109	2.9118	0.00 secs

The second table shows the relative influence of the top 10 features in each configuration. Relative influence in this instance was an impurity-based measure, where where feature importance is based on the average total reduction of the loss function for a given feature across all trees. It indicated that $X1$ (percentage change in GDP) and $X6$ (number of job vacancies) were very important to predicting the unemployment rate for all three model configurations and many of the other top-five features were shared by two model configurations. The values also indicated that for all three configurations, the first two or three features were much more importance to the model than the remaining seven or more features.

Interestingly, the only two indicators not considered to be as important as the rest were the percentage

[8]MSE was used instead of the square root of the mean squared error in the interest of more easily maintaining consistency and comparability with the neural network performance metrics.

change in government FCE ($X2$) and the first difference of the CPI ($X4$). One possible explanation for the former could be that since it is related to the percentage change in FCE of all sectors ($X3$), each model could be concluding it has no significant additional information to add.

mod1_feature	mod1_value	mod2_feature	mod2_value	mod3_feature	mod3_value
X1	49.586190	X1	45.049625	X1_1	41.513713
diff_X6	17.441382	diff_X6	13.901777	X1	20.966868
X3	9.088328	diffY_2	13.809810	X6_1	5.174087
year	7.906534	X3	6.669991	diff_X6	4.436387
X4	6.403893	year	4.712502	diffY_2	4.045388

Out of the three model configurations, the final model chosen to forecast the unemployment rate on the test set was the first model configuration because it was the most parsimonious and because it performed the best across all three performance metrics.

Test set predictive performance

The true test of a model's performance is on an unseen test set. Here, the test data was the UR from 2015 through to the end of 2019. The number of trees used in the forecast was the number chosen to train the final model for this particular subset of features.

The results on the test set are even better than the training set performance and still beat the naive model, which indicates that the GBM has done well at modelling the UR.

model	MSE	MAE	MAPE
GBM	0.0053	0.0587	1.0699
naive	0.0129	0.0947	1.7242

Application of a Neural Network

Deep neural network architecture

In contrast to machine learning algorithms such as GBMs and SVMs, deep learning algorithms provide a multi-layer approach to data modelling by placing an emphasis on learning successive layers of meaningful representations (Boehmke & Greenwell, 2020). The attraction of deep learning has been the complete automation of one of the most time-consuming but crucial steps in the machine-learning workflow - feature engineering.

At their core, all deep learning neural networks are based on the multi-layer perceptron architecture; otherwise known as a feed-forward deep neural network. These types of neural networks consist of an input layer, one or more hidden layers that model the different relationships between the input features and the outcome variable, and the output layer where the model predictions are produced. The layers are dense or fully connected, which means that all the nodes in each layer are connected together.

The number of hidden layers and nodes within those hidden layers determines the capacity of the model to learn the different features of the data, and therefore also influences the training time of the model. There are no hard rules as to the number of layers and nodes to choose for any particular problem, but too many nodes will lead to overfitting (and similarly too few will lead to underfitting). A common rule of thumb is to start with a number somewhere between the size of the input layer (that is, the number of features) and the output layer. Given the small size of the training set and this rule of thumb, two hidden layers and seven nodes in each layer was chosen as a starting point. Each layer also required an activation function, which would determine the circumstances under which a node would fire a signal to the next layer. With regression problems, a linear function is used for the output layer and it is most common to use a rectified linear unit function for the hidden layers.

Neural networks also require an objective (or loss) function and an optimiser to progressively improve the accuracy of the model across a specified number or iterations or epochs by adjusting the weights across all the node connections. Here, MSE was chosen as the loss function and adam (adaptive moment estimation) was chosen as the optimiser[9].

The final aspect of model training was the use of five-fold cross-validation. This would help determine if the algorithm was overfitting the data and provide more stable performance metrics.

Training set performance and interpretation

Due to the ability of a deep neural network to entirely automate the feature engineering process, only a single model was trained in this instance. It used the seven original features to forecast the UR, with only a single addition - including time stamp information to help the model understand it was working with time series information.

Plots indicated that the average value of each metric across the five cross-validation folds were largely still declining at 1,000 epochs. However, the relatively small rate of decline at that point and the desire to minimise model overfitting meant that the number of epochs was not increased.

A number of different aspects of the model were experimented with in an effort to improve training performance, including changing the batch size, including a dropout layer and changing the learning

[9]An equally valid choice for the optimiser could have been RMSProp (root mean square propagation), and primarily differ in how fast they descend the gradient.

rate. The vast majority of parameter changes appeared to adversely affect training performance, but increasing the number of nodes in each hidden layer slightly improved it, so seventeen nodes were used in the first layer and seven in the second.

The table below shows the summary performance metrics for the neural network compared to the the naive model on the training data. The values indicated that the deep learning model performed much worse than the naive model. It was out by 0.41 percentage points on average compared to the naive model's 0.21 percentage points. Part of the issue could likely have been that the hyperparameters were not been tuned to their optimal levels for the data and/or the model specification was flawed. During the model training process, it was decided that a comprehensive tuning grid search would not be performed as the desktop research performed as part of this analysis had indicated that it could take be very computationally- and time-intensive due to the number of parameters that can be adjusted.

model	MSE	MAE	MAPE	train_time
model 1	0.3381	0.4122	6.0956	145.07 secs
naive	0.0880	0.2109	2.9118	0.00 secs

Test set predictive performance

The deep neural network was unable to improve on its training performance and also performed much worse than the naive model on the unseen test data. The forecasts were out by about 0.24 percentage points on average, whereas the naive model was only out by about 0.09 percentage points on average.

model	MSE	MAE	MAPE
DNN	0.1235	0.2444	4.4370
naive	0.0129	0.0947	1.7242

Impact of changing the number of hidden layers on predictive performance

In order to understand how changing the neural network architecture might impact predictive performance, the number of hidden layers was varied by first dropping it down to a single hidden layer and then increasing it up to three hidden layers. In order to maintain comparability, the number of nodes in each layer was kept at seventeen nodes and predictive performance was assessed using MAE. All other parameters such as batch size, type of optimiser and activation function remained as per the main analysis. In order to reduce computational time, the k-fold cross-validation was replaced by a pseudo-validation split of one-third.

The effect of changing the number of hidden layers made some difference to the point at which the model overfit, with this occurring much sooner for the models with two or three hidden layers (Figure 2). There was also very little change in the training time for each model (all three took just under 12 seconds), although differences may have been more pronounced if the variation in the number of hidden layers had been greater. Finally, there were noticeable improvements in the training set performance as the number of hidden layers increased: MSE dropped from 3.64 to 0.82 and MAE dropped from 1.57 to 0.67.

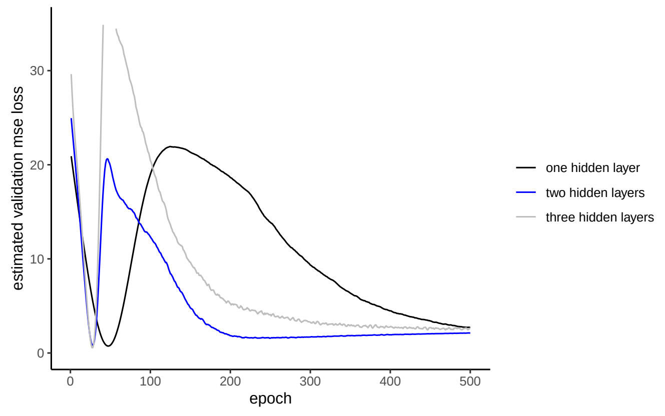


Figure 2: Effect of changing the number of hidden layers on the loss function

Impact of changing the number of nodes on predictive performance

In a similar manner, a small exploration into how changing the number of nodes in the hidden layer might affect predictive performance was undertaken. In order to make it easier to observe any relationship and not be confounded by the possible interaction between the number of hidden layers and the number of nodes in each layer, the network architecture consisted of a single hidden layer. The number of nodes in the hidden layer started at 17, was dropped down to 7 and then increased up to 34. All other parameters except the number of epochs remained as specified in the previous exploration - this was increased to 1,000 to get a better picture of the evolution of the mse loss across the epochs.

The effect of changing the number of nodes in the hidden layer was slightly more pronounced in terms of the point where each model started to overfit the data (Figure 3); the models with higher numbers of nodes overfit much faster than the model with only seven nodes. Again, there was also very little difference in training time between the three models with each taking around 25 seconds to run. Model performance increased as the number of nodes increased, with MSE and MAE decreasing to 0.93 and 0.72 respectively for the 34-node hidden layer.

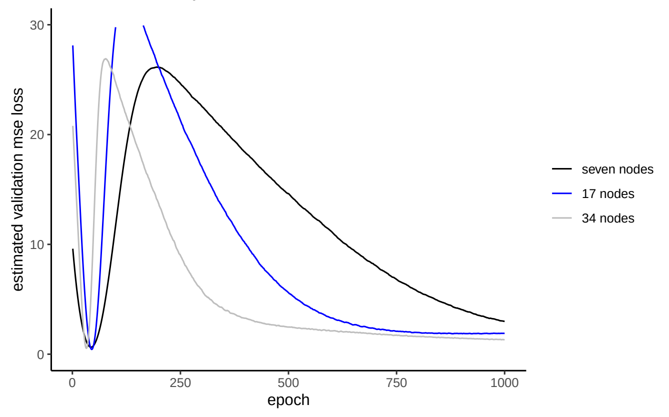


Figure 3: Effect of changing the number of nodes in the hidden layer on the loss function

Algorithm Comparison

Cross-validated accuracy

Both the GBM and neural network models were trained using 5-fold cross-validation. One of the biggest advantages to using cross-validation over a training-validation-test split or other bootstrapping methods is the ability to be more confident about training performance results. It is also an ideal approach when the size of the dataset is small, as it enables much more the data to be retained for training purposes and still produce reliable performance results. However, cross-validation does increase training time as the algorithm runs over the dataset multiple times and can potentially be problematic when used with time series data as the observations are ordered by a specified interval so taking a random sample is not entirely appropriate.

The metric used to assess cross-validation accuracy for both algorithms was squared error. Averaging this metric across the five folds and then across the number of trees (GBM) or the number of epochs (DNN), revealed that cross-validation accuracy was much higher for the GBM compared to the neural network. In fact, the minimum error value for the GBM was 0.03 compared to 0.35 for the neural network.

Figure 4: GBM algorithm cross−validated accuracy

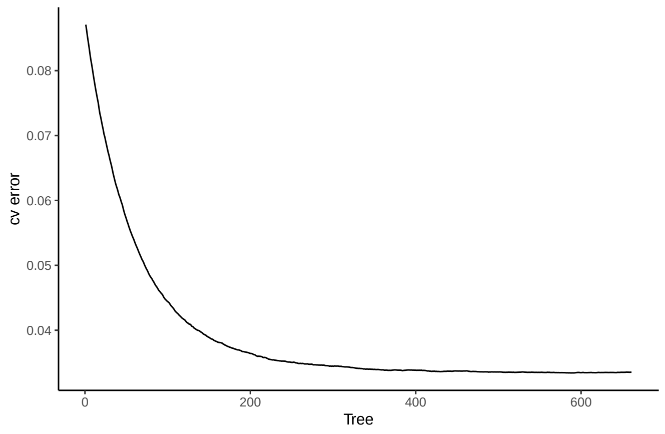
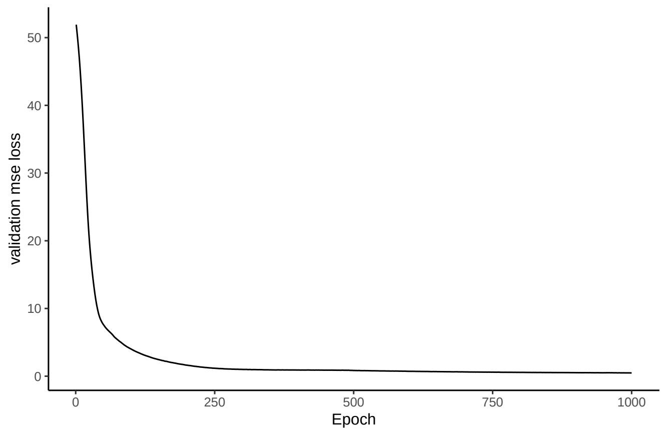


Figure 5: DNN algorithm cross−validated accuracy

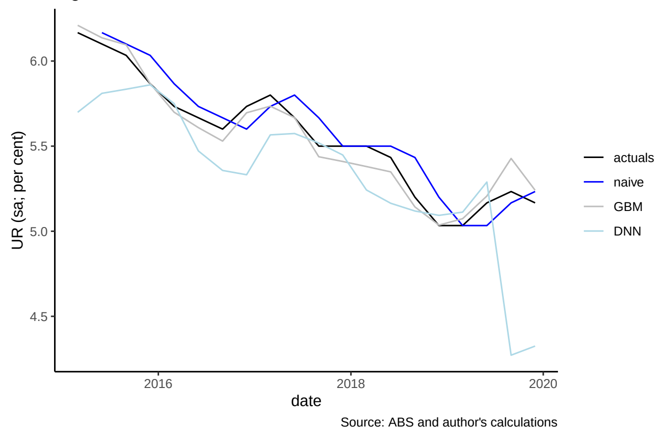


Predictive performance

The summary table detailing the predictive performance for the final model of each algorithm indicates clearly that the GBM was the better algorithm against both the training and test data. It was also able to outperform the naive persistence model, which the neural network was unable to do in its current configuration.

model	train_MSE	train_MAE	train_MAPE	test_MSE	test_MAE	test_MAPE
GBM	0.0178	0.1078	1.6525	0.0053	0.0587	1.0699
DNN	0.3381	0.4122	6.0956	0.1235	0.2444	4.4370
naive	0.0880	0.2109	2.9118	0.0129	0.0947	1.7242

Figure 6: Test data forecasts, 2015 to 2020



Computation time

The time measurements in the table indicated that the training time for even a simple deep neural network model on a small dataset was much longer than what it took to train a GBM on the same data. Indeed the time required to train the neural network was only slightly shorter than what it took to train and tune the GBM.

It was also clear that the hyperparameter tuning of a machine learning model takes much longer than the initial training, and is influenced by the size of the input data and the size of the tuning grid it is presented with (that is, more tuning parameters and more values to search across means an exponentially increasing number of combinations to compare and hence increased tuning time). There were some indications from the desktop research that tuning a deep neural network can take a long time, which is consistent with the difference in training time between the GBM and the neural network here.

It should also be noted that in this case, the significantly longer training time did not correspond with increased predictive performance, which can make the trade-off worthwhile.

model	train_time	tune_time	total_time
GBM	5.79 secs	2.5 mins	155.58 secs
DNN	145.07 secs	NA	NA
naive	0.00 secs	NA	0.00 secs

Algorithm interpretability

GBMs and neural networks are both classed as black-box algorithms. This means that interpreting model results and the decision rules behind those results can be difficult if not impossible. The trade-off is the powerful prediction performance (otherwise known as the accuracy-interpretability trade-off). Boosted trees have a slight advantage over neural networks in that it is slightly easier to produce outputs such as feature importance plots and variable partial dependencies. In the case of economic data it is often just as important to be able to understand variable interactions as it is to produce accurate forecasts as economic indicators are often used to make policy decisions and adjust macro- and microeconomic parameters.

Final algorithm choice

All the results consistently indicated that with the current model specifications, the GBM algorithm did a better choice at modelling and forecasting the Australian seasonally-adjusted UR rate between 1981 and 2020. In summary, the GBM had better cross-validated accuracy, better predictive performance and was able to outperform the naive persistence model. In addition, it was less time-intensive to train and was somewhat more interpretable than the neural network, which is generally important for economic data.

Suggestions for future research

The background research and subsequent modelling and analysis contained in this paper highlighted a number of areas of further research that could help to improve the accuracy of the modelling and or forecasting of either or model machine learning algorithm.

1. Given the tendency of the unemployment rate to reflect economic conditions, additional features such as the wage price index or other measures or wage growth in Australia, or various stock market indices such as the ASX200 could be added as additional features.
2. Using classical time series forecasting methods to assist in the feature engineering of variables for the GBM or any other shallow machine learning method. ARIMA and VAR methods for examples have been designed to work with some of the unique characteristics of time series data as they are particularly good at describing the dynamic behaviour of economic and financial data.
3. Long short term memory networks are a specific type of recurrent neural network that is capable of learning long-term dependencies. As such, they are well-suited to processing and making predictions based on time series data. It is strongly suspected that if a LSTM network architecture was used to model the UR instead of a basic feed-forward deep neural network then model performance would have significantly improved and rivalled that of the GBM.

Conclusion

A GBM and a deep neural network were used to model the quarterly Australian seasonally-adjusted UR between 1981 and 2014 and then forecast the next four years of data using seven economic indicators. A GBM is an example of a machine learning method unable to understand the concept of time, so the data was transformed into a supervised learning problem by differencing any variables that looked to have a stochastic or deterministic trend and including year and quarter information. Additional lags of the features and outcome variable were also considered but ultimately discarded as they offered no significant improvement in performance. Due to the ability of a neural network to completely automate the feature engineering required for shallow machine learning methods such as the GBM, similar transformations were not performed. Performance of the two algorithms was assessed using three accuracy metrics and a comparison against a naive persistence model. Although all the results consistently indicated that the GBM model had better performance in this instance, the author strongly suspected that the poor performance of the neural network was primarily due to a flawed algorithm architecture and a well-designed LSTM network would have rivalled the GBM.

Reference List

Australian Bureau of Statistics. (2005). Time series analysis: The basics. Retrieved from https://www.abs.gov.au/websitedbs/D3310114.nsf/home/Time+Series+Analysis:+The+Basics

Australian Bureau of Statistics. (2014). 6105.0 - Australian labour market statistics, July 2014: How does the ABS measure unemployment. Retrieved from https://www.abs.gov.au/AUSSTATS/abs@.nsf/Latestproducts/6105.0Feature%20Article53July%202014?opendocument&tabname=Summary&prodno=6105.0&issue=July%202014&num=&view=.

Australian Bureau of Statistics. (2016). 5216.0 - Australian system of national accounts: Concepts, sources and methods, 2015. Retrieved from https://www.abs.gov.au/AUSSTATS/abs@.nsf/Lookup/5216.0Glossary12015?OpenDocument

Bishop, J., Kent, C., Plumb, M., & Rayner, V. (2013). The resources boom and the Australian economy: A sectoral analysis. Bulletin - March Quarter 2013, 39-50. Retrieved from https://www.rba.gov.au/publications/bulletin/2013/mar/5.html#fn0

Boehmke, B. & Greenwell, B. (2020). *Hands-On Machine Learning with R*. CRC Press. Retrieved from: https://bradleyboehmke.github.io/HOML/

Office of the Chief Economist. (2014). *Australian Industry Report 2014*. Canberra: DIIS. Retrieved from https://www.industry.gov.au/data-and-publications/australian-industry-report-2014

Appendix

Appendix A - Glossary

- **Chain volume measure (cvm)** - Measures value quantities by using prices in a base period that are annually updated and then linked together to provide a time series. Considered by the ABS to be a more accurate reflection of volume changes over time than current or constant price estimates.
- **Consumer Price Index (CPI)** - Measures changes in the price of a 'basket' of goods and services, which account for a high proportion of expenditure by the CPI population group. In Australia, the CPI is measured on a quarterly basis.
- **Estimated resident population (ERP)** - A count of the population based on where they usually reside at the time of measurement. It includes all people, regardless of nationality or citizenship who are living in Australia at the time.
- **Final consumption expenditure (FCE)** - Net expenditure on goods and services by public authorities that does not result in the creation of fixed assets or inventories or the acquisition of land and existing buildings or second-hand assets (ABS, 2016).
- **Gross Domestic Product (GDP)** - The primary indicator of measuring the overall health of a country's economy. It represents the total dollar value of all final goods and services produced over a specific time period.
- **Seasonally-adjusted data** - Data from which the effects of systematic and calendar-related time series influences have been removed. Examples of such influences include weather fluctuations representative of a specific season, Christmas and other stable holidays and the start and end of school term. Seasonal effects can conceal the true underlying movement in a series as well as any non-seasonal characteristics, which is why it is usually removed prior to analysis (ABS, 2005).
- **Spurious regression** - When one or more features that are highly serially correlated with the outcome variable but may or may not actually be related are regressed on that outcome variable. This often leads to the model indicating the presence of a non-existent relationship and other incorrect conclusions.
- **Stationarity** - A time series with constant statistical properties over time; that is, its mean, variance, autocorrelation etc are constant between subsequent time periods. Many classicial statistical forecasting techniques are based on the assumption that time series data can be rendered approximately stationary through the use of mathematical transformations.
- **Terms of Trade (ToT)** - Represents the relationship between export and import prices.
- **Trend estimate** - The *'long-term' movement in a time series without calendar and irregular effects, and is a reflection of the underlying level. It is the result of influences such as population growth, price, inflation and general economic changes*.
- **Unemployment rate** - The proportion of the labour force that is unemployed. An unemployed person in Australia is someone of working age who did not work for at least one hour in the reference week for that month's Labour Force Survey, but was actively looking for work during the previous four weeks and was able to start in the reference week (ABS, 2014).

Appendix B - Dataset

The dataset used for this research was based from a small subset of indicators published by the Australian Bureau of Statistics (ABS). It contains quarterly time series data from June 1981 to December 2019.

The response variable (Y) was the quarterly seasonally-adjusted unemployment rate. The ABS publishes the bulk of its labour force data on a monthly basis, but national account, inflation and demographic indicators are all published on a quarterly basis so it was necessary to change the frequency of the outcome variable.

Seven ABS indicators were used to model and forecast the unemployment rate:

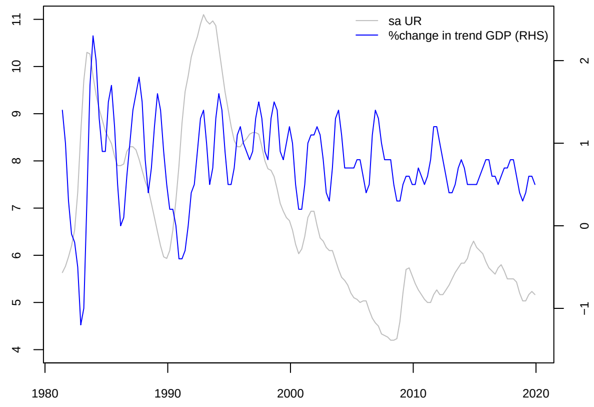
- Trend estimates of the percentage change in chain volume GDP ($X1$)
- Trend estimates of the percentage change in chain volume general government FCE ($X2$)
- Trend estimates of the percentage change in the chain volume FCE of all sectors ($X3$)
- Trend estimates of the percentage change in the ToT index ($X4$)
- All groups CPI ($X5$)
- Number of job vacancies (measured in thousands; $X6$)
- Estimated Resident Population of Australia (measured in millions; $X7$).

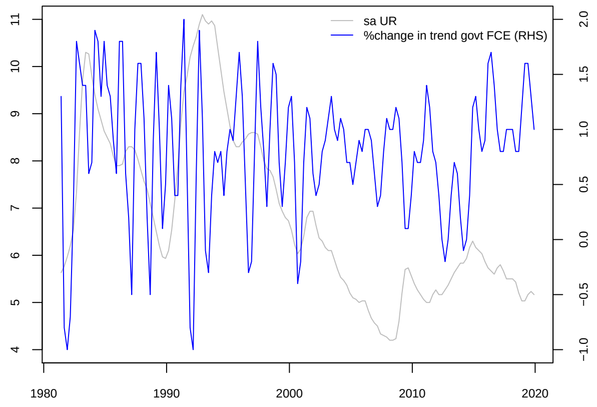
After the data pre-processing steps outlined in the *Data* section, the descriptive statistics for the entire dataset were as per the table below:

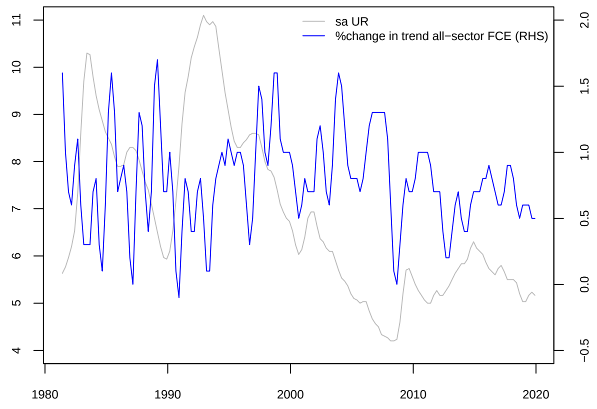
	Y	X1	X2	X3	X4	X5	X6	X7
NAs	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Minimum	-0.47	-1.20	-1.00	-0.10	-7.00	-0.30	-94.80	-6791.90
Maximum	1.27	2.30	2.00	1.70	7.60	2.70	63.10	132.80
Mean	0.00	0.76	0.80	0.79	0.34	0.57	1.26	23.70
Median	-0.07	0.70	0.80	0.70	0.15	0.50	1.70	62.45
Stdev	0.28	0.52	0.62	0.36	2.47	0.41	10.34	553.31

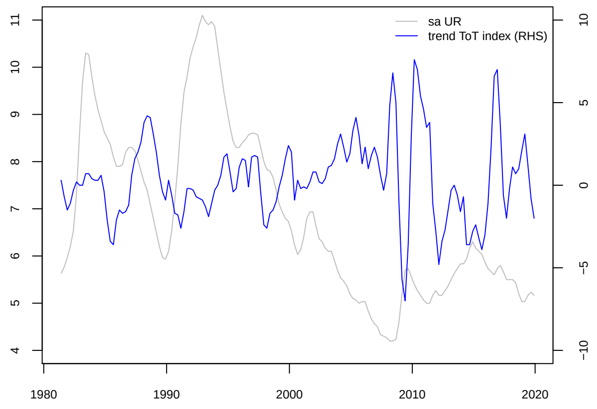
Appendix C - Plots of each feature against the unemployment rate

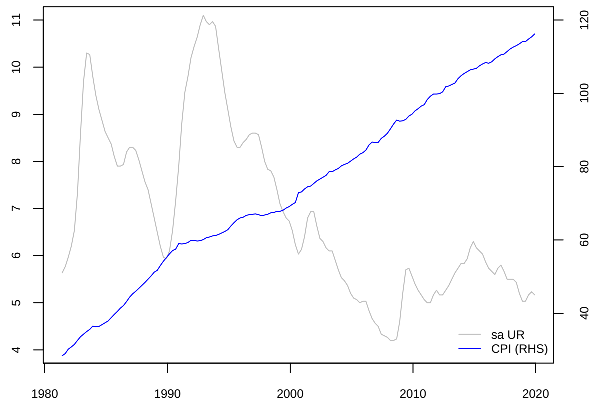
The following seven figures depict each of the seven original features plotted against the UR.

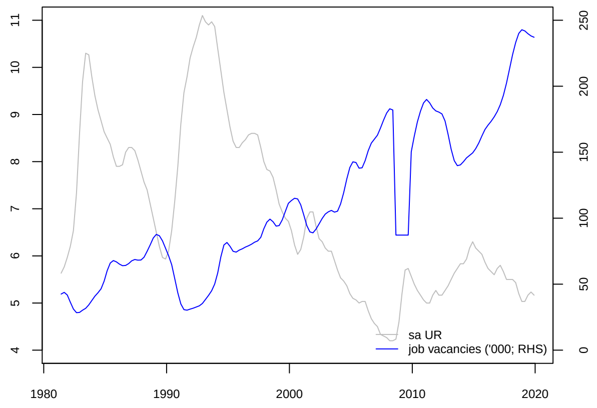


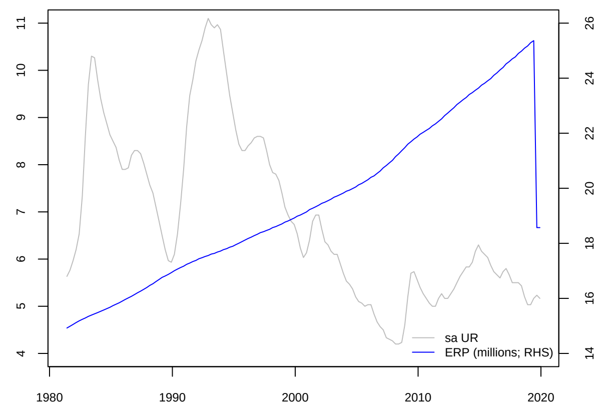




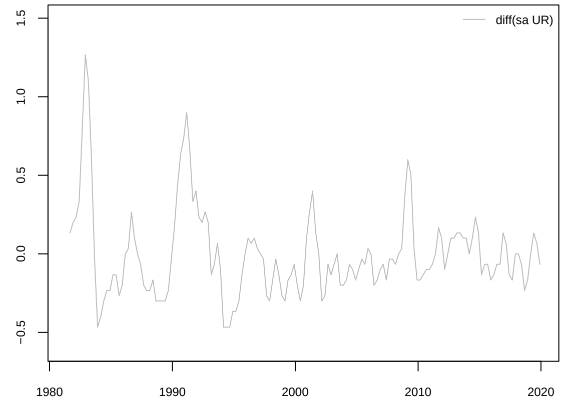


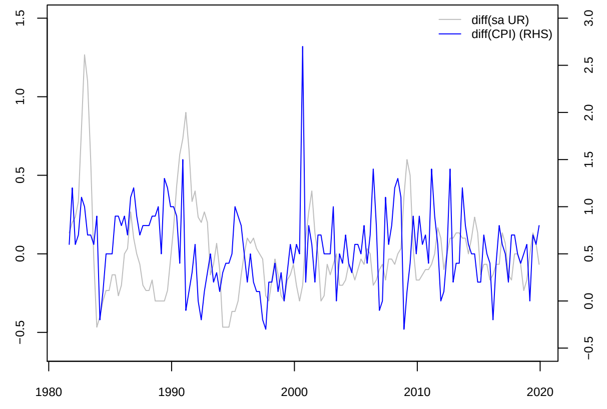


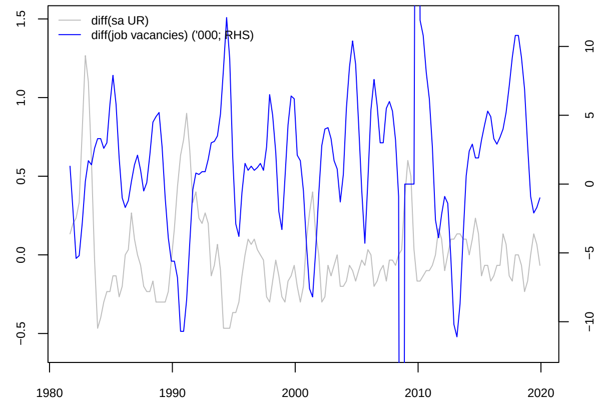


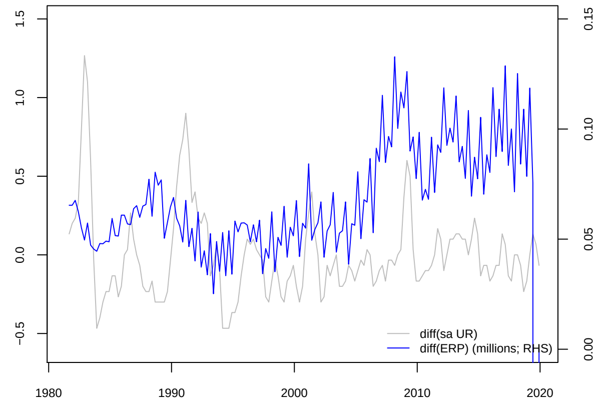


The following four figures depict the UR, CPI, job vacancies and ERP after taking first differences to remove the trend in each series.









Appendix D - Main report R Code

Overview of the Australian unemployment rate

```r
library(readxl)
library(tidyverse)
library(scales)

# data import
col_names <- data.frame(date = "date",
                        read_excel("AUS_Data.xlsx",
                                   n_max = 1,
                                   col_names = FALSE),
                        stringsAsFactors = FALSE)
data <- read_excel("AUS_Data.xlsx", skip = 2, col_names = FALSE)
colnames(data) <- col_names
data <- mutate_if(data, is.character, as.numeric)
data$date <- as.Date(data$date, format = "%Y-%m-%d")

# plotting the UR
xmin = as.Date("1999-03-01", format = "%Y-%m-%d")

data %>%
  subset(date >= "1999-03-01") %>%
  ggplot(aes(x = date, y = Y)) +
  geom_line(colour = "#000099") +
  scale_y_continuous(limits = c(4, 7.5), breaks = seq(4, 7.5, 0.5)) +
  scale_x_date(breaks = function(x) seq.Date(from = xmin,
                                             to = max(data$date),
                                             by = "2 years"),
               labels = date_format("%Y", tz = "AEST")) +
  labs(x = "Year",
       y = "Unemployment rate (per cent)",
       title = "Figure 1: Unemployment rate, quarterly seasonally-adjusted
       estimates, 1999 to 2019",
       caption = "Source: Australian Bureau of Statistics") +
  theme_classic()
```

Data

```r
data_nd <- data %>% dplyr::select(-date)

library(fBasics)
sum_tbl <- round(basicStats(data_nd)[c("NAs","Minimum", "Maximum", "Mean",
                                       "Median", "Stdev"),], digits = 2)
knitr::kable(sum_tbl)
```

```r
### Data Pre-Processing
## Missing values
```

```r
# imputation on what will be the training data
sum(is.na(data))
colSums(is.na(data))

train <- data %>%
  subset(date < "2015-03-01")

colSums(is.na(train))
X6_med <- median(train$X6, na.rm = TRUE)
data$X6 <- replace_na(data$X6, X6_med)

# replacement on what will be the test data
test <- data %>%
  subset(date >= "2015-03-01")

colSums(is.na(test))
X7_med <- median(train$X7, na.rm = TRUE)
data$X7 <- replace_na(data$X7, X7_med)

## Feature engineering
# extracting year and quarter from time stamp
library(lubridate)

year <- year(as.POSIXct(data$date))
quarter <- quarter(as.POSIXct(data$date))

# taking first differences of non-stationary variables
diff_Y <- c(NA, diff(data$Y))
diff_X5 <- c(NA, diff(data$X5))
diff_X6 <- c(NA, diff(data$X6))
diff_X7 <- c(NA, diff(data$X7))

# creating lags of all variables as additional features
diffY_2 <- dplyr::lag(diff_Y, n = 2)
diffY_3 <- dplyr::lag(diff_Y, n = 3)
diffY_4 <- dplyr::lag(diff_Y, n = 4)

X1_1 <- dplyr::lag(data$X1)
X2_1 <- dplyr::lag(data$X2)
X3_1 <- dplyr::lag(data$X3)
X4_1 <- dplyr::lag(data$X4)

X5_1 <- dplyr::lag(diff_X5)
X6_1 <- dplyr::lag(diff_X6)
X7_1 <- dplyr::lag(diff_X7)
```

```r
add_vars <- data.frame(diff_Y = diff_Y,
                       diffY_2 = diffY_2,
                       diffY_3 = diffY_3,
                       diffY_4 = diffY_4,
                       X1_1 = X1_1,
                       X2_1 = X2_1,
                       X3_1 = X3_1,
                       X4_1 = X4_1,
                       X5_1 = X5_1,
                       X6_1 = X6_1,
                       X7_1 = X7_1,
                       diff_X5 = diff_X5,
                       diff_X6 = diff_X6,
                       diff_X7 = diff_X7,
                       year,
                       quarter)

# adding new features onto dataset
data_orig <- data
data <- cbind(data_orig, add_vars)

## Splitting data into training and test sets
train <- data %>%
  subset(date < "2015-03-01")

test <- data %>%
  subset(date >= "2015-03-01")

## Feature-wise normalisation using min-max method
# only applies to neural network as tree-based models don't require such
#  a transformation
nn_train1 <- train %>%
  select(Y, X1:X7, year, quarter)

nn_train2 <- train %>%
  select(diff_Y, X1:X4, diff_X5:diff_X7, year, quarter)
nn_train2 <- nn_train2[2:nrow(nn_train2),]

nn_test1 <- test %>%
  select(Y, X1:X7, year, quarter)

nn_test2 <- test %>%
  select(diff_Y, X1:X4, diff_X5:diff_X7, year, quarter)

library(caret)
train_stats1 <- preProcess(nn_train1[2:10], method = "range")
```

```r
norm_train1 <- predict(train_stats1, nn_train1[2:10])
norm_test1 <- predict(train_stats1, nn_test1[2:10])

train_stats2 <- preProcess(nn_train2[2:10], method = "range")
norm_train2 <- predict(train_stats2, nn_train2[2:10])
norm_test2 <- predict(train_stats2, nn_test2[2:10])
```

Application of a GBM

```r
### Model training
## Naive persistence model
start_naive <- Sys.time()

naive_pred <- dplyr::lag(train$Y)
naive_df <- data.frame(date = train$date,
                       Y = train$Y,
                       Y_pred = naive_pred)

end_naive <- Sys.time()
time_naive <- end_naive - start_naive
```

```r
## Model 1: First differences plus time stamp features
# model specification
diff_train <- train %>%
  select(date, diff_Y, X1:X4, diff_X5:diff_X7, year:quarter)
diff_train <- diff_train[2:nrow(diff_train),]

# training a basic model
library(gbm)

start_gbm1 <- Sys.time()

set.seed(2020)
diff_gbm <- gbm(diff_Y ~.-date, data = diff_train,
                distribution = "gaussian",
                n.trees = 2000,         # set to an arbitrarily high number
                shrinkage = 0.1,        # default
                interaction.depth = 3, # default is 1
                n.minobsinnode = 10,    # default
                bag.fraction = 0.5,     # default
                cv.folds = 5)

diff_gbm
RMSE_loc <- which.min(diff_gbm$cv.error)
RMSE <- sqrt(min(diff_gbm$cv.error))
gbm.perf(diff_gbm, method = "cv", plot.it = FALSE)
```

```r
# tuning the learning rate
start_tunegbm1 <- Sys.time()

gbm_grid <- expand.grid(
  shrinkage = c(0.001, 0.005, 0.01, 0.05, 0.1, 0.3),
  RMSE = NA,
  trees = NA,
  time = NA)

for(i in seq_len(nrow(gbm_grid))) {
  set.seed(2020)
  train_time <- system.time({
    gbm_tune <- gbm(
      diff_Y ~.-date, data = diff_train,
      distribution = "gaussian",
      n.trees = 1000, # set somewhere above the optimal number indicated by the
                      #  first run
      shrinkage = gbm_grid$shrinkage[i],
      interaction.depth = 3,
      n.minobsinnode = 10,
      bag.fraction = 0.5,
      cv.folds = 5)
  })

  gbm_grid$RMSE[i] <- sqrt(min(gbm_tune$cv.error))
  gbm_grid$trees[i] <- which.min(gbm_tune$cv.error)
  gbm_grid$time[i] <- train_time[["elapsed"]]

}

gbm_grid %>%
  dplyr::arrange(RMSE) %>%
  head(20)

# tuning the remaining parameters
gbm_grid <- expand.grid(
  n.trees = 661,
  shrinkage = 0.01,
  interaction.depth = c(1, 3, 5, 7, 8),
  n.minobsinnode = c(5, 10, 15),
  bag.fraction = c(0.5, 0.65, 0.8))

gbm_fit <- function(n.trees, shrinkage, interaction.depth, n.minobsinnode,
                    bag.fraction) {
  set.seed(2020)
  gbm_tune <- gbm(
    diff_Y ~.-date, data = diff_train,
    distribution = "gaussian",
    n.trees = n.trees,
    shrinkage = shrinkage,
    interaction.depth =  interaction.depth,
    n.minobsinnode = n.minobsinnode,
    bag.fraction = bag.fraction,
    cv.folds = 5
  )
  sqrt(min(gbm_tune$cv.error))
}

gbm_grid$RMSE <- purrr::pmap_dbl(
  gbm_grid,
  ~ gbm_fit(
    n.trees = ..1,
    shrinkage =  ..2,
    interaction.depth = ..3,
    n.minobsinnode = ..4,
    bag.fraction = ..5
  )
)

hyper_params1 <- gbm_grid %>%
  dplyr::arrange(RMSE) %>%
  head(20)

end_tunegbm1 <- Sys.time()

# training the final model
n.trees <- hyper_params1[[1,1]]
shrinkage <- hyper_params1[[1,2]]
interaction.depth <- hyper_params1[[1,3]]
n.minobsinnode <- hyper_params1[[1,4]]
bag.fraction <- hyper_params1[[1,5]]

set.seed(2020)
gbm1 <- gbm(diff_Y ~.-date, data = diff_train,
            distribution = "gaussian",
            n.trees = n.trees,
            shrinkage = shrinkage,
            interaction.depth = interaction.depth,
            n.minobsinnode = n.minobsinnode,
            bag.fraction = bag.fraction,
            cv.folds = 5)
gbm1
```

```r
summary(gbm1, plot = FALSE) # produces feature importance output

end_gbm1 <- Sys.time()

totaltime_gbm1 <- end_gbm1 - start_gbm1
tunetime_gbm1 <- end_tunegbm1 - start_tunegbm1
traintime_gbm1 <- totaltime_gbm1 - tunetime_gbm1
```

```r
### Model 2: Model 1 plus outcome variable lags
# model specification
difflag_train <- train %>%
  select(date, diff_Y, diffY_2:diffY_4, X1:X4, diff_X5:diff_X7, year:quarter)
difflag_train <- difflag_train[6:nrow(difflag_train),]

# training a basic model
start_gbm2 <- Sys.time()

set.seed(2020)
difflag_gbm <- gbm(diff_Y ~.-date, data = difflag_train,
               distribution = "gaussian",
               n.trees = 2000,        # set to an arbitrarily high number
               shrinkage = 0.1,       # default
               interaction.depth = 3, # default is 1
               n.minobsinnode = 10,   # default
               bag.fraction = 0.5,    # default
               cv.folds = 5)

difflag_gbm
RMSE_loc <- which.min(difflag_gbm$cv.error)
RMSE <- sqrt(min(difflag_gbm$cv.error))
gbm.perf(difflag_gbm, method = "cv", plot.it = FALSE)

# tuning the learning rate
start_tunegbm2 <- Sys.time()

gbm_grid <- expand.grid(
  shrinkage = c(0.001, 0.005, 0.01, 0.05, 0.1, 0.3),
  RMSE = NA,
  trees = NA,
  time = NA)

for(i in seq_len(nrow(gbm_grid))) {
  set.seed(2020)
  train_time <- system.time({
    gbm_tune <- gbm(
      diff_Y ~.-date, data = difflag_train,
      distribution = "gaussian",
      n.trees = 1000, # set somewhere above the optimal number indicated by the
                      #  first run
      shrinkage = gbm_grid$shrinkage[i],
      interaction.depth = 3,
      n.minobsinnode = 10,
      bag.fraction = 0.5,
      cv.folds = 5)
  })

  gbm_grid$RMSE[i] <- sqrt(min(gbm_tune$cv.error))
  gbm_grid$trees[i] <- which.min(gbm_tune$cv.error)
  gbm_grid$time[i] <- train_time[["elapsed"]]

}

gbm_grid %>%
  dplyr::arrange(RMSE) %>%
  head(20)

# tuning the remaining parameters
gbm_grid <- expand.grid(
  n.trees = 706,
  shrinkage = 0.05,
  interaction.depth = c(1, 3, 5, 7, 8),
  n.minobsinnode = c(5, 10, 15),
  bag.fraction = c(0.5, 0.65, 0.8))

gbm_fit <- function(n.trees, shrinkage, interaction.depth, n.minobsinnode,
                    bag.fraction) {
  set.seed(2020)
  gbm_tune <- gbm(
    diff_Y ~.-date, data = difflag_train,
    distribution = "gaussian",
    n.trees = n.trees,
    shrinkage = shrinkage,
    interaction.depth =  interaction.depth,
    n.minobsinnode = n.minobsinnode,
    bag.fraction = bag.fraction,
    cv.folds = 5
  )
  sqrt(min(gbm_tune$cv.error))
}

gbm_grid$RMSE <- purrr::pmap_dbl(
  gbm_grid,
  ~ gbm_fit(
    n.trees = ..1,
    shrinkage =  ..2,
    interaction.depth = ..3,
    n.minobsinnode = ..4,
    bag.fraction = ..5
  )
)

hyper_params2 <- gbm_grid %>%
  dplyr::arrange(RMSE) %>%
  head(20)

end_tunegbm2 <- Sys.time()

# training the final model
n.trees <- hyper_params2[[1,1]]
shrinkage <- hyper_params2[[1,2]]
interaction.depth <- hyper_params2[[1,3]]
n.minobsinnode <- hyper_params2[[1,4]]
bag.fraction <- hyper_params2[[1,5]]

set.seed(2020)
gbm2 <- gbm(diff_Y ~.-date, data = difflag_train,
            distribution = "gaussian",
            n.trees = n.trees,
            shrinkage = shrinkage,
            interaction.depth = interaction.depth,
            n.minobsinnode = n.minobsinnode,
            bag.fraction = bag.fraction,
            cv.folds = 5)
gbm2
summary(gbm2, plot = FALSE) # produces feature importance output

end_gbm2 <- Sys.time()

totaltime_gbm2 <- end_gbm2 - start_gbm2
tunetime_gbm2 <- end_tunegbm2 - start_tunegbm2
traintime_gbm2 <- totaltime_gbm2 - tunetime_gbm2
```

```
### Model 3: Kitchen sink (Model 2 plus one lag of all features except
#    time stamp features)
# model specification
ks_train <- train %>%
  select(date, diff_Y, diffY_2:diffY_4, X1:X4, diff_X5:diff_X7, X1_1:X7_1,
         year:quarter)
```

```
ks_train <- ks_train[6:nrow(ks_train),]

# training a basic model
start_gbm3 <- Sys.time()

set.seed(2020)
ks_gbm <- gbm(diff_Y ~.-date, data = ks_train,
              distribution = "gaussian",
              n.trees = 2000,         # set to an arbitrarily high number
              shrinkage = 0.1,        # default
              interaction.depth = 3, # default is 1
              n.minobsinnode = 10,    # default
              bag.fraction = 0.5,     # default
              cv.folds = 5)

ks_gbm
RMSE_loc <- which.min(ks_gbm$cv.error)
RMSE <- sqrt(min(ks_gbm$cv.error))
gbm.perf(ks_gbm, method = "cv", plot.it = FALSE)

# tuning the learning rate
start_tunegbm3 <- Sys.time()

gbm_grid <- expand.grid(
  shrinkage = c(0.001, 0.005, 0.01, 0.05, 0.1, 0.3),
  RMSE = NA,
  trees = NA,
  time = NA)

for(i in seq_len(nrow(gbm_grid))) {
  set.seed(2020)
  train_time <- system.time({
    gbm_tune <- gbm(
      diff_Y ~.-date, data = ks_train,
      distribution = "gaussian",
      n.trees = 1000, # set somewhere above the optimal number indicated by the
                      #  first run
      shrinkage = gbm_grid$shrinkage[i],
      interaction.depth = 3,
      n.minobsinnode = 10,
      bag.fraction = 0.5,
      cv.folds = 5)
  })

  gbm_grid$RMSE[i] <- sqrt(min(gbm_tune$cv.error))
  gbm_grid$trees[i] <- which.min(gbm_tune$cv.error)
```

```r
  gbm_grid$time[i] <- train_time[["elapsed"]]

}

gbm_grid %>%
  dplyr::arrange(RMSE) %>%
  head(20)

# tuning the remaining parameters
gbm_grid <- expand.grid(
  n.trees = 972,
  shrinkage = 0.05,
  interaction.depth = c(1, 3, 5, 7, 8),
  n.minobsinnode = c(5, 10, 15),
  bag.fraction = c(0.5, 0.65, 0.8))

gbm_fit <- function(n.trees, shrinkage, interaction.depth, n.minobsinnode,
                    bag.fraction) {
  set.seed(2020)
  gbm_tune <- gbm(
    diff_Y ~.-date, data = ks_train,
    distribution = "gaussian",
    n.trees = n.trees,
    shrinkage = shrinkage,
    interaction.depth =  interaction.depth,
    n.minobsinnode = n.minobsinnode,
    bag.fraction = bag.fraction,
    cv.folds = 5
  )
  sqrt(min(gbm_tune$cv.error))
}

gbm_grid$RMSE <- purrr::pmap_dbl(
  gbm_grid,
  ~ gbm_fit(
    n.trees = ..1,
    shrinkage =  ..2,
    interaction.depth = ..3,
    n.minobsinnode = ..4,
    bag.fraction = ..5
  )
)

hyper_params3 <- gbm_grid %>%
  dplyr::arrange(RMSE) %>%
  head(20)
```

```r
end_tunegbm3 <- Sys.time()

# training the final model
n.trees <- hyper_params3[[1,1]]
shrinkage <- hyper_params3[[1,2]]
interaction.depth <- hyper_params3[[1,3]]
n.minobsinnode <- hyper_params3[[1,4]]
bag.fraction <- hyper_params3[[1,5]]

set.seed(2020)
gbm3 <- gbm(diff_Y ~.-date, data = ks_train,
            distribution = "gaussian",
            n.trees = n.trees,
            shrinkage = shrinkage,
            interaction.depth = interaction.depth,
            n.minobsinnode = n.minobsinnode,
            bag.fraction = bag.fraction,
            cv.folds = 5)
gbm3
summary(gbm3, plot = FALSE) # produces feature importance output

end_gbm3 <- Sys.time()

totaltime_gbm3 <- end_gbm3 - start_gbm3
tunetime_gbm3 <- end_tunegbm3 - start_tunegbm3
traintime_gbm3 <- totaltime_gbm3 - tunetime_gbm3
```

```r
# reverse engineering the transformation for training set performance
train_gbm <- data.frame(date = train$date,
                        Y = train$Y,
                        naive_pred = naive_df$Y_pred,
                        diff_Y = train$diff_Y,
                        diff1_pred = c(NA, gbm1$cv.fitted),
                        diff2_pred = c(rep(NA, 5), gbm2$cv.fitted),
                        diff3_pred = c(rep(NA, 5), gbm3$cv.fitted),
                        gbm1_pred = c(NA, gbm1$cv.fitted) + train$Y,
                        gbm2_pred = c(rep(NA, 5), gbm2$cv.fitted) + train$Y,
                        gbm3_pred = c(rep(NA, 5), gbm3$cv.fitted) + train$Y)

naive_error <- train_gbm$Y[2:135] - train_gbm$naive_pred[2:135]
MSE_naive <- mean(naive_error^2)
MAE_naive <- mean(abs(naive_error))
MAPE_naive <- mean(abs(naive_error / train_gbm$Y[2:135]) * 100)

gbm1_error <- train_gbm$Y[2:135] - train_gbm$gbm1_pred[2:135]
MSE_gbm1 <- mean(gbm1_error^2)
```

```r
MAE_gbm1 <- mean(abs(gbm1_error))
MAPE_gbm1 <- mean(abs(gbm1_error / train_gbm$Y[2:135]) * 100)

gbm2_error <- train_gbm$Y[6:135] - train_gbm$gbm2_pred[6:135]
MSE_gbm2 <- mean(gbm2_error^2)
MAE_gbm2 <- mean(abs(gbm2_error))
MAPE_gbm2 <- mean(abs(gbm2_error / train_gbm$Y[6:135]) * 100)

gbm3_error <- train_gbm$Y[6:135] - train_gbm$gbm3_pred[6:135]
MSE_gbm3 <- mean(gbm3_error^2)
MAE_gbm3 <- mean(abs(gbm3_error))
MAPE_gbm3 <- mean(abs(gbm3_error / train_gbm$Y[6:135]) * 100)
```

```r
gbm_train_perf <- data.frame(model = c("model 1", "model 2", "model 3", "naive"),
                             MSE = round(c(MSE_gbm1, MSE_gbm2, MSE_gbm3,
                                           MSE_naive), 4),
                             MAE = round(c(MAE_gbm1, MAE_gbm2, MAE_gbm3,
                                           MAE_naive), 4),
                             MAPE = round(c(MAPE_gbm1, MAPE_gbm2, MAPE_gbm3,
                                            MAPE_naive), 4),
                             train_time = round(c(totaltime_gbm1, totaltime_gbm2,
                                                  totaltime_gbm3, time_naive), 2))
knitr::kable(gbm_train_perf)
```

```r
library(vip)

rel_inf1 <- data.frame(gbm1_feature = head(vi(gbm1)[,1], 5),
                       gbm1_value = head(vi(gbm1)[,2], 5))

rel_inf2 <- data.frame(gbm1_feature = head(vi(gbm2)[,1], 5),
                       gbm1_value = head(vi(gbm2)[,2], 5))

rel_inf3 <- data.frame(gbm1_feature = head(vi(gbm3)[,1], 5),
                       gbm1_value = head(vi(gbm3)[,2], 5))

gbm_vi <- data.frame(rel_inf1, rel_inf2, rel_inf3)
colnames(gbm_vi) <- c("mod1_feature", "mod1_value", "mod2_feature", "mod2_value",
                      "mod3_feature", "mod3_value")

knitr::kable(gbm_vi)
```

```r
diff_test <- test %>%
  select(diff_Y, X1:X4, diff_X5:diff_X7, year:quarter)

gbm_fc <- predict(gbm1, newdata = diff_test, n.trees = hyper_params1[[1,1]])

# naive persistence model forecasts
```

```r
naive_fc <- dplyr::lag(test$Y)
naive_tdf <- data.frame(date = test$date,
                        Y = test$Y,
                        Y_fc = naive_fc)

test_gbm <- data.frame(date = test$date,
                       Y = test$Y,
                       naive_fc = naive_tdf$Y_fc,
                       diff_Y = test$diff_Y,
                       diffgbm_fc = gbm_fc,
                       gbm_fc = gbm_fc + test$Y)

naive_terror <- test_gbm$Y[2:20] - test_gbm$naive_fc[2:20]
tMSE_naive <- mean(naive_terror^2)
tMAE_naive <- mean(abs(naive_terror))
tMAPE_naive <- mean(abs(naive_terror / test_gbm$Y[2:20]) * 100)

gbm_terror <- test_gbm$Y - test_gbm$gbm_fc
tMSE_gbm <- mean(gbm_terror^2)
tMAE_gbm <- mean(abs(gbm_terror))
tMAPE_gbm <- mean(abs(gbm_terror / test_gbm$Y) * 100)
```

```r
gbm_test_perf <- data.frame(model = c("GBM", "naive"),
                            MSE = round(c(tMSE_gbm, tMSE_naive), 4),
                            MAE = round(c(tMAE_gbm, tMAE_naive), 4),
                            MAPE = round(c(tMAPE_gbm, tMAPE_naive), 4))
knitr::kable(gbm_test_perf)
```

Application of a Neural Network

```r
## Model: Baseline plus time stamp variables
# model specification
library(keras)

train1_data <- as.matrix(norm_train1)
train1_y <- as.matrix(nn_train1$Y)

build_model <- function() {
  nn1 <- keras_model_sequential() %>%
    layer_dense(units = 17, activation = "relu",
                input_shape = dim(train1_data)[[2]]) %>%
    layer_dense(units = 7, activation = "relu") %>%
    layer_dense(units = 1)

nn1 %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae", "mape", "mse"))
}

# model training with k-fold cross-validation
start_nn1 <- Sys.time()

k <- 5
indices <- sample(1:nrow(train1_data))
folds <- cut(indices, breaks = k, labels = FALSE)
num_epochs <- 1000
all_mae_histories <- NULL
all_mape_histories <- NULL
all_mse_histories <- NULL

for (i in 1:k) {
  cat("processing fold #", i, "\n")

  # preparing validation data: data from partition k
  val_indices <- which(folds == i, arr.ind = TRUE)
  val_data <- train1_data[val_indices,]
  val_targets <- train1_y[val_indices]

  # preparing validation data: data from all other partitions
  partial_train_data <- train1_data[-val_indices,]
  partial_train_targets <- train1_y[-val_indices]

  # building model
  nn1 <- build_model()

  # training model
  history <- nn1 %>% fit (
    train1_data, train1_y,
    validation_data = list(val_data, val_targets),
    epochs = num_epochs, batch_size = 32, verbose = FALSE,
  )
  mae_history <- history$metrics$val_mae
  mape_history <- history$metrics$val_mape
  mse_history <- history$metrics$val_mse

  all_mae_histories <- rbind(all_mae_histories, mae_history)
  all_mape_histories <- rbind(all_mape_histories, mape_history)
  all_mse_histories <- rbind(all_mse_histories, mse_history)
}

# averaging per-epoch metric scores for all folds
average_mae_history <- data.frame(
```

```r
  epoch = seq(1:ncol(all_mae_histories)),
  validation_mae = apply(all_mae_histories, 2, mean)
)

average_mape_history <- data.frame(
  epoch = seq(1:ncol(all_mape_histories)),
  validation_mape = apply(all_mape_histories, 2, mean)
)

average_mse_history <- data.frame(
  epoch = seq(1:ncol(all_mse_histories)),
  validation_mse = apply(all_mse_histories, 2, mean)
)

# plotting validation scores
library(ggplot2)
mse_plot1 <- average_mse_history %>%
  ggplot(aes(x = epoch, y = validation_mse)) +
  geom_line()

mse_min1 <- which.min(average_mse_history$validation_mse)
mse1 <- average_mse_history$validation_mse[mse_min1]

# training the final model
final_nn1 <- build_model()
final_nn1_mod <- fit(final_nn1, train1_data, train1_y,
                     epochs = 1000, batch_size = 32, verbose = FALSE)

nn1_pred <- predict(final_nn1, train1_data)

end_nn1 <- Sys.time()
time_nn1 <- end_nn1 - start_nn1

# test set performance
test1_data <- as.matrix(norm_test1)
test1_y <- as.matrix(nn_test1$Y)

test_perf <- evaluate(final_nn1, test1_data, test1_y)
test_perf

nn_fc <- predict(final_nn1, test1_data)

tMSE_nn = as.numeric(test_perf[1])
tMAE_nn = as.numeric(test_perf[2])
tMAPE_nn = as.numeric(test_perf[3])
```

```r
# training set performance
train_nn <- data.frame(date = train$date,
                       Y = train$Y,
                       naive_pred = naive_df$Y_pred,
                       nn1_pred = nn1_pred)

naive_error <- train_nn$Y[2:135] - train_nn$naive_pred[2:135]
MSE_naive <- mean(naive_error^2)
MAE_naive <- mean(abs(naive_error))
MAPE_naive <- mean(abs(naive_error / train_nn$Y[2:135]) * 100)

nn1_error <- train_nn$Y - train_nn$nn1_pred
MSE_nn1 <- mean(nn1_error^2)
MAE_nn1 <- mean(abs(nn1_error))
MAPE_nn1 <- mean(abs(nn1_error / train_nn$Y) * 100)
```

```r
nn_train_perf <- data.frame(model = c("model 1", "naive"),
                            MSE = round(c(MSE_nn1, MSE_naive), 4),
                            MAE = round(c(MAE_nn1, MAE_naive), 4),
                            MAPE = round(c(MAPE_nn1, MAPE_naive), 4),
                            train_time = round(c(time_nn1, time_naive), 2))
knitr::kable(nn_train_perf)
```

```r
nn_test_perf <- data.frame(model = c("DNN", "naive"),
                           MSE = round(c(tMSE_nn, tMSE_naive), 4),
                           MAE = round(c(tMAE_nn, tMAE_naive), 4),
                           MAPE = round(c(tMAPE_nn, tMAPE_naive), 4))
knitr::kable(nn_test_perf)
```

```r
## Model: Baseline - two hidden layers
# model specification
exp1_start <- Sys.time()

nn_exp1 <- keras_model_sequential() %>%
  layer_dense(units = 17, activation = "relu",
              input_shape = dim(train1_data)[[2]]) %>%
  layer_dense(units = 17, activation = "relu") %>%
  layer_dense(units = 1)

nn_exp1 %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae"))

# model training
history1 <- fit(nn_exp1, train1_data, train1_y, validation_split = 1/3,
                epochs = 500, batch_size = 32,
                  verbose = FALSE)
hist_plot1 <- plot(history1)

exp1_data <- data.frame(x = c(1:history1$params$epochs),
                         y = history1$metrics$val_loss)
plot_exp1 <- ggplot(exp1_data, aes(x = x, y = y)) +
  geom_line() +
  xlab("epoch") +
  ylab("estimated validation mse loss")

overfit_epoch1 <- which.min(history1$metrics$val_loss)

exp1_pred <- predict(nn_exp1, train1_data)
exp1_error <- train1_y - exp1_pred
MSE_exp1 <- mean(exp1_error^2)
MAE_exp1 <- mean(abs(exp1_error))

exp1_end <- Sys.time()
exp1_time <- exp1_end - exp1_start
```

```r
## Model: One hidden layer
# model specification
exp2_start <- Sys.time()

nn_exp2 <- keras_model_sequential() %>%
  layer_dense(units = 17, activation = "relu",
              input_shape = dim(train1_data)[[2]]) %>%
  layer_dense(units = 1)

nn_exp2 %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae"))

# model training
history2 <- fit(nn_exp2, train1_data, train1_y, validation_split = 1/3,
                epochs = 500, batch_size = 32,
                 verbose = FALSE)
hist_plot2 <- plot(history2)

exp2_data <- data.frame(x = c(1:history2$params$epochs),
                         y = history2$metrics$val_loss)
plot_exp2 <- ggplot(exp2_data, aes(x = x, y = y)) +
  geom_line() +
  xlab("epoch") +
  ylab("estimated validation mse loss")
```

```r
overfit_epoch2 <- which.min(history2$metrics$val_loss)

exp2_pred <- predict(nn_exp2, train1_data)
exp2_error <- train1_y - exp2_pred
MSE_exp2 <- mean(exp2_error^2)
MAE_exp2 <- mean(abs(exp2_error))

exp2_end <- Sys.time()
exp2_time <- exp2_end - exp2_start
```

```r
## Model: Three hidden layers
# model specification
exp3_start <- Sys.time()

nn_exp3 <- keras_model_sequential() %>%
  layer_dense(units = 17, activation = "relu",
              input_shape = dim(train1_data)[[2]]) %>%
  layer_dense(units = 17, activation = "relu") %>%
  layer_dense(units = 17, activation = "relu") %>%
  layer_dense(units = 1)

nn_exp3 %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae"))

# model training
history3 <- fit(nn_exp3, train1_data, train1_y, validation_split = 1/3,
                epochs = 500, batch_size = 32,
                 verbose = FALSE)
hist_plot3 <- plot(history3)

exp3_data <- data.frame(x = c(1:history3$params$epochs),
                        y = history3$metrics$val_loss)
plot_exp2 <- ggplot(exp3_data, aes(x = x, y = y)) +
  geom_line() +
  xlab("epoch") +
  ylab("estimated validation mse loss")

overfit_epoch3 <- which.min(history3$metrics$val_loss)

exp3_pred <- predict(nn_exp3, train1_data)
exp3_error <- train1_y - exp3_pred
MSE_exp3 <- mean(exp3_error^2)
MAE_exp3 <- mean(abs(exp3_error))
```

```r
exp3_end <- Sys.time()
exp3_time <- exp3_end - exp3_start
```

```r
exp_data_metrics <- data.frame(exp2_loss = history2$metrics$val_loss,
                               exp2_mae = history2$metrics$val_mae,
                               exp1_loss = history1$metrics$val_loss,
                               exp1_mae = history1$metrics$val_mae,
                               exp3_loss = history3$metrics$val_loss,
                               exp3_mae = history3$metrics$val_mae)
metric_means <- apply(exp_data_metrics, 2, mean)

exp_data_pred <- data.frame(exp2_MSE = MSE_exp2,
                            exp2_MAE = MAE_exp2,
                            exp1_MSE = MSE_exp1,
                            exp1_MAE = MAE_exp1,
                            exp3_MSE = MSE_exp3,
                            exp4_MAE = MAE_exp3)

library(reshape2)

exp_data <- data.frame(epoch = exp1_data$x,
                       mse_1hd = exp2_data$y,
                       mse_2hd = exp1_data$y,
                       mse_3hd = exp3_data$y)
exp_data <- melt(exp_data, id.vars = "epoch")

exp_data %>%
  ggplot() +
  geom_line(aes(x = epoch, y = value, color = variable)) +
  scale_colour_manual(name = NULL,
                        labels = c("one hidden layer", "two hidden layers",
                                   "three hidden layers"),
                        values = c("black", "blue", "grey")) +
  scale_y_continuous(limits = c(0, 35), breaks = seq(0, 35, 10)) +
  labs(x = "epoch",
       y = "estimated validation mse loss",
       title = "Figure 2: Effect of changing the number of hidden
       layers on the loss function") +
  theme_classic()
```

```r
## Model: Baseline - one hidden layer with 17 nodes
exp4_start <- Sys.time()

nn_exp4 <- keras_model_sequential() %>%
  layer_dense(units = 17, activation = "relu",
              input_shape = dim(train1_data)[[2]]) %>%
  layer_dense(units = 1)
```

```r
nn_exp4 %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae"))

# model training
history4 <- fit(nn_exp4, train1_data, train1_y, validation_split = 1/3,
                epochs = 1000, batch_size = 32,
                 verbose = FALSE)
hist_plot4 <- plot(history4)

exp4_data <- data.frame(x = c(1:history4$params$epochs),
                        y = history4$metrics$val_loss)
plot_exp4 <- ggplot(exp4_data, aes(x = x, y = y)) +
  geom_line() +
  xlab("epoch") +
  ylab("estimated validation mse loss")

overfit_epoch4 <- which.min(history4$metrics$val_loss)

exp4_pred <- predict(nn_exp4, train1_data)
exp4_error <- train1_y - exp4_pred
MSE_exp4 <- mean(exp4_error^2)
MAE_exp4 <- mean(abs(exp4_error))

exp4_end <- Sys.time()
exp4_time <- exp4_end - exp4_start
```

```r
## Model: One hidden layer with seven nodes
exp5_start <- Sys.time()

nn_exp5 <- keras_model_sequential() %>%
  layer_dense(units = 7, activation = "relu",
              input_shape = dim(train1_data)[[2]]) %>%
  layer_dense(units = 1)

nn_exp5 %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae"))

# model training
history5 <- fit(nn_exp5, train1_data, train1_y, validation_split = 1/3,
                epochs = 1000, batch_size = 32,
                 verbose = FALSE)
hist_plot5 <- plot(history5)
```

```r
exp5_data <- data.frame(x = c(1:history5$params$epochs),
                        y = history5$metrics$val_loss)
plot_exp5 <- ggplot(exp5_data, aes(x = x, y = y)) +
  geom_line() +
  xlab("epoch") +
  ylab("estimated validation mse loss")

overfit_epoch5 <- which.min(history5$metrics$val_loss)

exp5_pred <- predict(nn_exp5, train1_data)
exp5_error <- train1_y - exp5_pred
MSE_exp5 <- mean(exp5_error^2)
MAE_exp5 <- mean(abs(exp5_error))

exp5_end <- Sys.time()
exp5_time <- exp5_end - exp5_start
```

```r
## Model: One hidden layer with 34 nodes
exp6_start <- Sys.time()

nn_exp6 <- keras_model_sequential() %>%
  layer_dense(units = 34, activation = "relu",
              input_shape = dim(train1_data)[[2]]) %>%
  layer_dense(units = 1)

nn_exp6 %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae"))

# model training
history6 <- fit(nn_exp6, train1_data, train1_y, validation_split = 1/3,
                epochs = 1000, batch_size = 32,
                 verbose = FALSE)
hist_plot6 <- plot(history6)

exp6_data <- data.frame(x = c(1:history6$params$epochs),
                        y = history6$metrics$val_loss)
plot_exp6 <- ggplot(exp6_data, aes(x = x, y = y)) +
  geom_line() +
  xlab("epoch") +
  ylab("estimated validation mse loss")

overfit_epoch6 <- which.min(history6$metrics$val_loss)

exp6_pred <- predict(nn_exp6, train1_data)
```

```r
exp6_error <- train1_y - exp6_pred
MSE_exp6 <- mean(exp6_error^2)
MAE_exp6 <- mean(abs(exp6_error))

exp6_end <- Sys.time()
exp6_time <- exp6_end - exp6_start
```

```r
exp_data_metrics2 <- data.frame(exp5_loss = history5$metrics$val_loss,
                                exp5_mae = history5$metrics$val_mae,
                                exp4_loss = history4$metrics$val_loss,
                                exp4_mae = history4$metrics$val_mae,
                                exp6_loss = history6$metrics$val_loss,
                                exp6_mae = history6$metrics$val_mae)
metric_means2 <- apply(exp_data_metrics2, 2, mean)

exp_data_pred2 <- data.frame(exp5_MSE = MSE_exp5,
                             exp5_MAE = MAE_exp5,
                             exp4_MSE = MSE_exp4,
                             exp4_MAE = MAE_exp4,
                             exp6_MSE = MSE_exp6,
                             exp6_MAE = MAE_exp6)

exp_data2 <- data.frame(epoch = exp4_data$x,
                        mse_17n = exp5_data$y,
                        mse_7n = exp4_data$y,
                        mse_34n = exp6_data$y)
exp_data2 <- melt(exp_data2, id.vars = "epoch")

exp_data2 %>%
  ggplot() +
  geom_line(aes(x = epoch, y = value, color = variable)) +
  scale_colour_manual(name = NULL,
                      labels = c( "seven nodes", "17 nodes", "34 nodes"),
                      values = c("black", "blue", "grey")) +
  scale_y_continuous(limits = c(0, 30), breaks = seq(0, 30, 10)) +
  labs(x = "epoch",
       y = "estimated validation mse loss",
       title = "Figure 3: Effect of changing the number of nodes in
       the hidden layer on the loss function") +
  theme_classic()
```

Algorithm Comparison

```r
gbm_cv <- data.frame(num_trees = 1:gbm1$n.trees,
                     value = gbm1$cv.error)
min_gbm <- min(gbm_cv$value)
```

```r
gbm_cv %>%
  ggplot() +
  geom_line(aes(x = num_trees, y = value)) +
  labs(x = "Tree",
       y = "cv error",
       title = "Figure 4: GBM algorithm cross-validated accuracy") +
  theme_classic()

nn_cv <- average_mse_history
min_cv <- min(nn_cv$validation_mse)

nn_cv %>%
  ggplot() +
  geom_line(aes(x = epoch, y = validation_mse)) +
  labs(x = "Epoch",
       y = "validation mse loss",
       title = "Figure 5: DNN algorithm cross-validated accuracy") +
  theme_classic()
```

```r
pred_perf_summary <- data.frame(model = c("GBM", "DNN", "naive"),
                                train_MSE = round(c(MSE_gbm1, MSE_nn1, MSE_naive), 4),
                                train_MAE = round(c(MAE_gbm1, MAE_nn1, MAE_naive), 4),
                                train_MAPE = round(c(MAPE_gbm1, MAPE_nn1, MAPE_naive), 4),
                                test_MSE = round(c(tMSE_gbm, tMSE_nn, tMSE_naive), 4),
                                test_MAE = round(c(tMAE_gbm, tMAE_nn, tMAE_naive), 4),
                                test_MAPE = round(c(tMAPE_gbm, tMAPE_nn, tMAPE_naive), 4))
knitr::kable(pred_perf_summary)

test_fcs <- data.frame(date = test$date,
                       Y = test$Y,
                       naive_fc = naive_tdf$Y_fc,
                       gbm_fc = gbm_fc + test$Y,
                       nn_fc = nn_fc)
test_fcs2 <- melt(test_fcs, id.vars = "date")

pred_plot <- test_fcs2 %>%
  ggplot() +
  geom_line(aes(x = date, y = value, color = variable)) +
  scale_colour_manual(name = NULL,
                      labels = c("actuals", "naive", "GBM", "DNN"),
                      values = c("black", "blue", "grey", "light blue")) +
  labs(x = "date",
       y = "UR (sa; per cent)",
       title = "Figure 6: Test data forecasts, 2015 to 2020",
       caption = "Source: ABS and author's calculations") +
  theme_classic()
```

```r
pred_plot
```

```r
comp_time <- data.frame(model = c("GBM", "DNN", "naive"),
                        train_time = round(c(traintime_gbm1, time_nn1, time_naive), 2),
                        tune_time = round(c(tunetime_gbm1, "", ""), 2),
                        total_time = round(c(totaltime_gbm1, "", time_naive), 2))
knitr::kable(comp_time)
```