Time series forecasting based on Artificial Intelligence algorithms and their application to economic and financial time series Master's Thesis

Master in Computer Engineering and Mathematics

Alexander Lagos

June 2020

Abstract

The following work seeks to evaluate traditional methodologies of time series forecasting compared to the newest artificial intelligence algorithms. Economic and financial time series are used in order to evaluate the effectiveness of these procedures so that this has an economic and practical significance.

Keywords: Neural Networks, Time Series, Forecasting

Resumen

El siguiente trabajo busca evaluar las metodologías tradicionales de predicción de series de tiempo en comparación con los algoritmos de inteligencia artificial más nuevos. Se utilizan series de tiempo económicas y financieras. con el fin de evaluar la efectividad de estos procedimientos para que esto tenga un significado económico y práctico.

Contents

List of Figures **4**

1 Introduction **5**
- 1.1 Objectives 6

2 Description of Work Done **10**
- 2.1 Classical Methods 12
- 2.2 Deep learning Methods 18
  - 2.2.1 Multi-layer perceptrons 20
  - 2.2.2 Recurrent Neural Networks 25
  - 2.2.3 Gate Recurrent Units 29
  - 2.2.4 Convolutional Neural Networks 30

3 Experimental results **34**
- 3.1 Arima Model Selection 37
- 3.2 Deep Learning Forecast 45

4 Conclusions **73**
- 4.1 Proposals for future work 75

5 References **76**

List of Figures

1 Employment and S&P 500 data from august 1978 to September 2019 11
2 Employment data from august 1978 to September 2019 11
3 Box-Jenkins Scheme. Source: researchgate.net 12
4 Employment data from august 1978 to September 2019 13
5 S&P 500 data from august 1978 to September 2019 14
6 First difference of Employment data 16
7 First difference of S&P 500 data 16
8 k-fold intuition. Source: Github 19
9 Basic Multi-layer perceptron. Source: Slideplayer.com 21
10 Dropout layer intuition. Source:[13] 23
11 RNN structures. Source: towardsdatascience.com 25
12 GRU structures. Source: data-blogger.com 29
13 An example of CNN architecture. Source:[30] 31
14 Relationship between S&P 500 and Labor Data 34
15 Employment Forecast Arima(1,1,2) 43
16 Employment Forecast Arima(1,1,2) 44
17 K-fold for RNN in Employment series 46
18 K-fold for RNN in S&P 500 series 47
19 K-fold for MLP in Employment series 47
20 K-fold for MLP in S&P 500 series 48
21 K-fold GRU for Employment series 49
22 K-fold GRU for S&P 500 series 49
23 K-fold CNN for Employment series 50
24 K-fold CNN for S&P 500 series 51
25 Final RNN structure 52
26 Final RNN structure 52
27 Final RNN Forecast for Employment series 53
28 Final RNN structure 54
29 Final RNN structure 54
30 Final RNN Forecast for S&P 500 series 55
31 Final MLP structure 56
32 Final MLP structure 56
33 Final MLP Forecast for Employment series 57
34 Final MLP structure 58
35 Final MLP structure 58
36 Final MLP Forecast for S&P 500 series 59
37 Final GRU structure 60
38 Final GRU structure 60

39 Final GRU Forecast for Employment series 61
40 Final GRU structure 62
41 Final GRU structure 63
42 Final GRU Forecast for S&P 500 series 63
43 Final CNN structure 64
44 Final CNN structure 65
45 Final CNN Forecast for Employment series 66
46 Final CNN structure 67
47 Final CNN structure 68
48 Final CNN Forecast for S&P 500 series 69

1 Introduction

The purpose of this work is to evaluate the efficiency of traditional methods as opposed to artificial intelligence algorithms in the forecasting of financial and economic time series. The importance of this topic has multiple ramifications in the field of economics and finance. Finance and economics have dealt with uncertainty for many years and the inability of economists to predict economic recessions has led to major errors in economic policy that have had high costs for society as a whole. On the other hand, financial markets are by nature random, and are affected by bad decisions of economic policy. The ability to improve the algorithms used to forecast the evolution of the economic series will have an important impact in decision making by firms and central banks. This should lead to a better allocation of resources, which will allow an improvement in general well-being.

Most prediction methods used in economics are based on recursive methods that extrapolate past results to the future. However the accuracy of these methods decreases considerably as the forecast horizon increases. The need to find new mathematical methods that adapt more efficiently to the trajectory of the time series, and that are able to handle a greater amount of information, offer an opportunity to artificial intelligence algorithms as possible solution that could increase the probability of success in the forecasting of time series.
Although traditional statistical time series methods perform well, many have inherent limitations. First, without expertise it is possible to misspecify the functional form relating the independent and dependent variables and fail to make the necessary data transformations. Second, outliers can lead to biased estimates of model parameters.
In addition, time series models are often linear and thus may not capture nonlinear behavior. Many have argued[14] that neural networks can overcome or, at least, be less subject to these limitations.

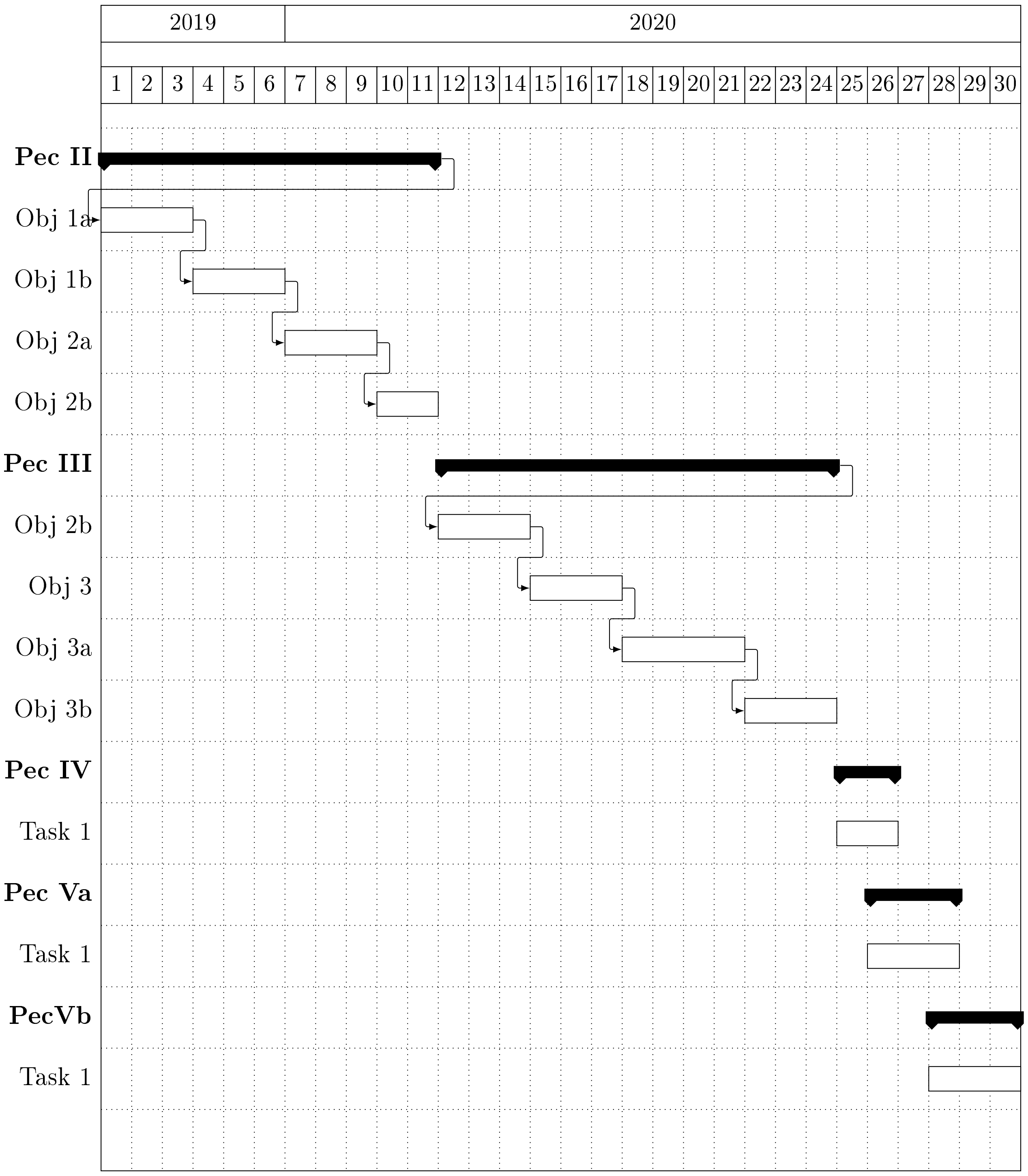
Some traditional statistical time series methods have inherent limitations due to the way in which the models are estimated. When many kinds of traditional statistical time series models are estimated, human interaction and evaluation are required. Also, many traditional statistical methods do not learn incrementally as new data arrive; instead, they must be re-estimated periodically. It has been claimed[15] that neural networks can also overcome these problems.

What this work raises is the requirement to both evaluate classical methods and use their results as a baseline when evaluating deep learning methods for time series forecasting in order demonstrate that their added complexity is adding skill to the forecast.

1.1 Objectives

- Implement classical methods for predicting time series.
  - 1a)Apply ARIMA models to the forecasting of economic and financial time series
- Implement a method for forecasting time series based on neural networks.
  - 2a)Learn about the different artificial intelligence algorithms that can be used to forecast economic and financial series.
  - 2b)Create forecasting neural networks based on recurrent, CNN, GRU and MLP structures.
- Compare the efficiency of traditional algorithms vs artificial intelligence algorithms in the forecasting of financial and economics time series.
  - 3a)Analyze the differences between economic and financial time series through the results obtained with artificial intelligence algorithms.
  - 3b)Reflect on the practical ability of artificial intelligence algorithms to be used as an economic analysis tool.

Project Timing



In accordance with the initially planned times, the times necessary to carry out the PEC are met, however, there are changes in the objectives and in the series initially proposed, which are detailed:

- Economic time series is changed from GDP to Employment series, this because GDP was only available quarterly.
- Initially the objective "Apply Simple Exponential Smoothing and holt Winter's to the forecasting of economic and financial time series." was raised. However, its application was considered redundant and did not add much value to the results that were sought, so it was removed.
- Initially out of sample forecast were made for ARIMA models, however for making it comparable with built in method "predict" , from TensorFlow-Keras, forecast was reduced to one step ahead for all models.

According to PEC I the following Risk and mitigation actions were proposed:

Risk 1) Results of the neural network do not exceed the classical methods

Contingency plan $\rightarrowtail$ choose another neural network

Risk 2) Computational cost of network training is excessive (too late)

Contingency plan$\rightarrowtail$ request access to UOC servers or use Google servers.

Risk 3) Results doesn't have economic or financial meaning

Contingency plan $\rightarrowtail$ conclusions will be oriented to the mathematical and algorithmic aspects

Table 1: Risk Classification

	Probbility	**Gravity**
1	High	Low
2	Low	High
3	High	Low

However none of these risks were materialized.
For risk 1, although the results of Neural Network Models did not exceed the results of the ARIMA models, they are considered very close and generally quite good and promising. Likewise, the main structures for time series based on Neural Networks were proposed, allowing for a broader comparative base.
For risk 2, although the cost of training the Neural Networks was high, it was considered manageable, and training sets could be generated, from which it was possible to draw the necessary conclusions.
For risk 3, it was possible to find an economic sense to the results when the results were separated between economic and financial series. Likewise, in the conclusions, neural networks are proposed as promising instruments for economic analysis in the coming years.

2 Description of Work Done

The Data

For the current project a sample of two series will be used that start from 08/01/1978 to 09/01/2019 of data collected from the St Louis Federal Reserve Bank homepage[16]. This data contains time series of the United States economy and US markets, the name of the different time series used in this study are shown in Table 1

Table 2: Names and types of the two data sets used

Data Set	**Type**
S&P 500	Financial
NonFarm Payrolls	Economic

The first one is the S&P 500 which is stock market index that tracks the price of the stocks of 500 large capitalization U.S. companies. It represents the stock market's performance by reporting the risks and returns of the biggest companies. Investors use it as the benchmark of the overall market, to which all other investments are compared.
The second index is the nonfarm payrolls report of the U.S. bureau of labor statistics. Nonfarm payrolls are a summation of payroll jobs available within the nonfarm payrolls classification as designated by the Bureau of Labor Statistics. The monthly nonfarm payrolls statistic is a measure of new payrolls added by private and government entities in the U.S. This time series is widely used in economic and financial analysis because of the importance it has to market participants, in the prediction of changes in economic growth. The Gross Domestic Product measures the value of economic activity within a country.

We begin the analysis by the comparison of the economic time series of the employment data, and financial time series. This time series is expected to be less volatile than financial time series.

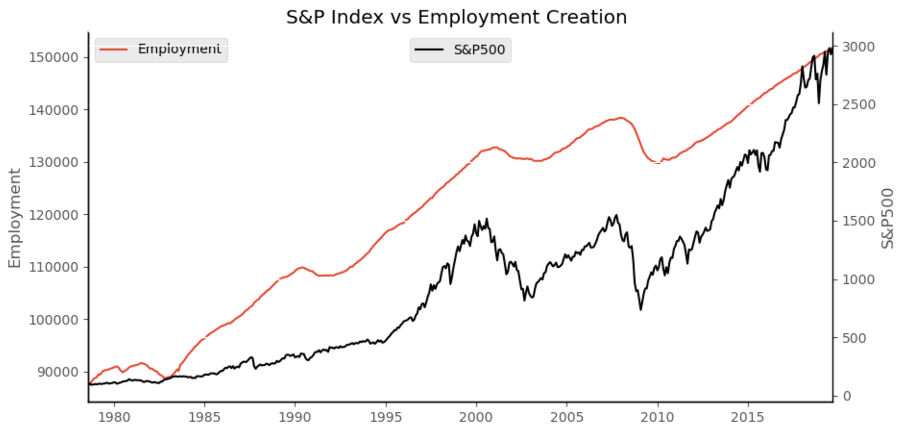


Figure 1: Employment and S&P 500 data from august 1978 to September 2019

Employment is expected to be a cyclical variable and so its related to the dynamics of GDP. We can check its difference in Volatility with the S&P 500 index by estimating its annual Volatility.

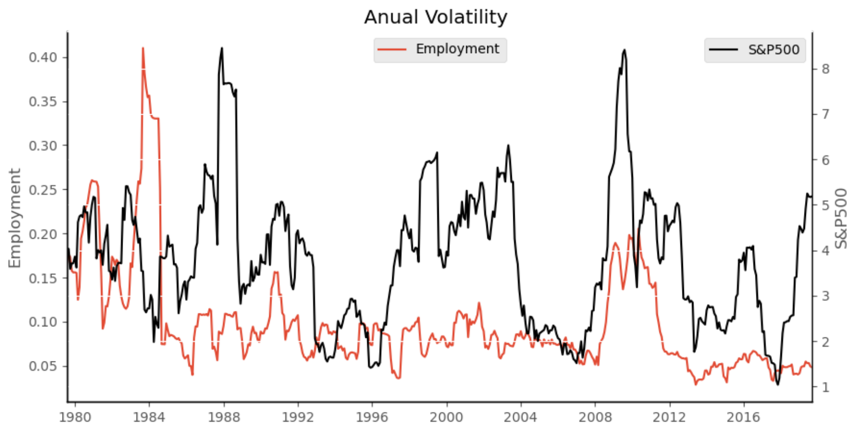


Figure 2: Employment data from august 1978 to September 2019

In general its expected that financial time series are more volatile and have less predictability than economic time series. As seen in the figure number two, the volatility of the S&P 500 is much higher than the volatility of the employment series.

2.1 Classical Methods

For both time series classical methods have been implemented following the box Jenkins methodology[17].

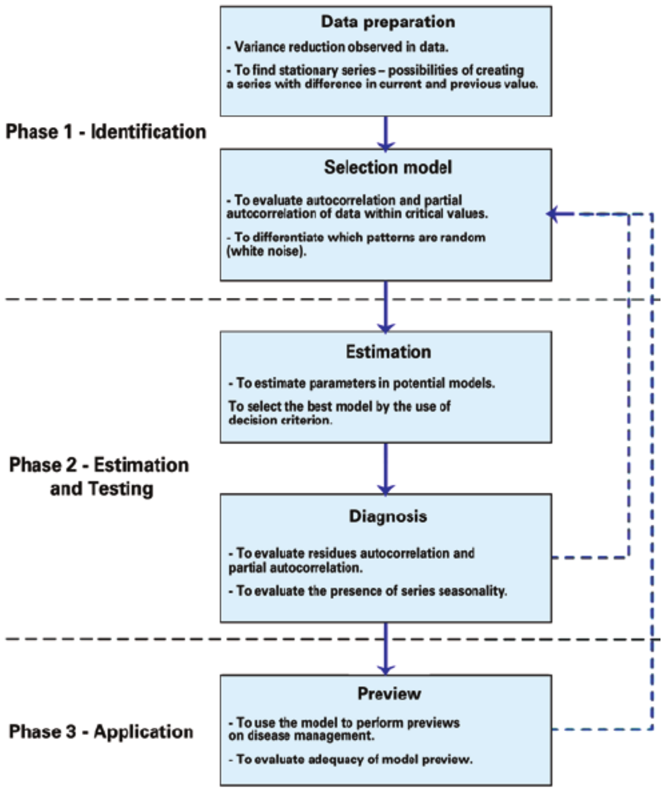

Data preparation
- Variance reduction observed in data.
- To find stationary series – possibilities of creating a series with difference in current and previous value.
Phase 1 - Identification
Selection model
- To evaluate autocorrelation and partial autocorrelation of data within critical values.
- To differentiate which patterns are random (white noise).
Estimation
- To estimate parameters in potential models.
To select the best model by the use of decision criterion.
Phase 2 - Estimation and Testing
Diagnosis
- To evaluate residues autocorrelation and partial autocorrelation.
- To evaluate the presence of series seasonality.
Preview
- To use the model to perform previews on disease management.
- To evaluate adequacy of model preview.
Phase 3 - Application

Figure 3: Box-Jenkins Scheme. Source: researchgate.net

Box - Jenkins Analysis refers to a systematic method of identifying, fitting, checking, and using integrated autoregressive, moving average (ARIMA) time series models. The method is appropriate for time series of medium to long length (at least 50 observations). The Box-Jenkins method refers to the iterative application of the following three steps:

- Identification. Using plots of the data, autocorrelations, partial autocorrelations, and other information, a class of simple ARIMA models is selected. This amounts to estimating appropriate values for p, d, and q.

- Estimation. The phis and thetas of the selected model are estimated using maximum likelihood techniques, backcasting, as outlined in Box-Jenkins [17].

- Diagnostic Checking. The fitted model is checked for inadequacies by considering the autocorrelations of the residual series (the series of residual, or error, values).

These steps are applied iteratively until step three does not produce any improvement in the model[19]

As shown in figure 4 we can separate the series in its irregular component and its trend component[18].

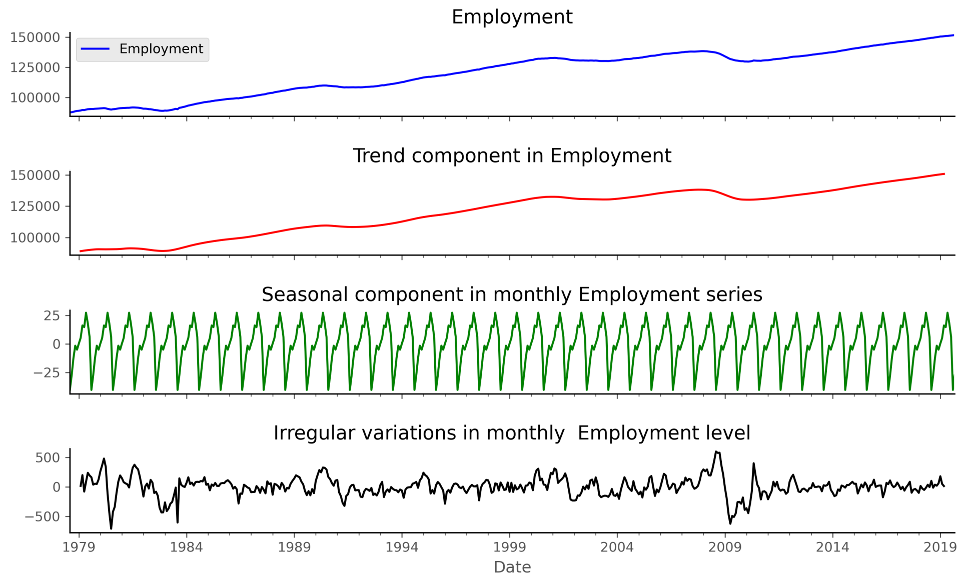


Figure 4: Employment data from august 1978 to September 2019

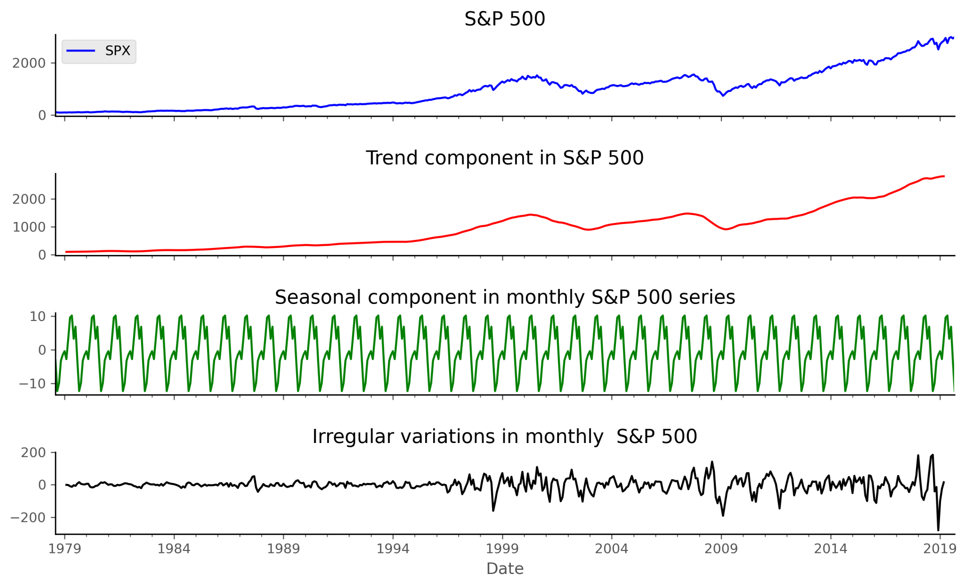


Figure 5: S&P 500 data from august 1978 to September 2019

For both methods we used the classical analysis of time series models that is, basic ARIMA modelling[18]. For the purpose of this analysis, al code was written in python for doing the following steps:

- Stationarity testing.
- Selection of different ARIMA models based on information criteria and the mean squared error.
- Evaluation of selected models with a train set and a test set.
- Evaluation of forecast made with Mean Squared Error(MSE) and R-squared.

For Stationarity the following test were used:

- LjungBox Test: The null hypothesis of the Ljung-Box test[20] is that the autocorrelations (for the chosen lags) in the population from which the sample is taken are all zero. If p-value < 0.051: we can reject the null hypothesis assuming a 5% chance of making a mistake. So we can assume that our values are showing dependence on each other. If p-value > 0.051: we don't have enough statistical evidence to reject the null hypothesis. So we can not assume that your values are dependent. This could mean that our values are dependent anyway or it can mean that our values are independent. But we are not proving any specific possibility, what our test actually said is that we can not assert the dependence of the values, neither can we assert the independence of the values.

- Dickey fuller test[23]: According to the Augmented Dickey-Fuller(ADF) test, in the presence of autocorrelation, the first-order differences x' t of the original series can be expressed as a linear regression model of the previous time index and first-order differences up to a lag of m times indices. The linear regression on x'.

- Evaluation of the behavior of mean and variance along different sample paths.

For model selection the following criteria have been used:

- "Akaike Info Criterion: Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models. Thus, AIC provides a means for model selection. AIC estimates the relative amount of information lost by a given model: the less information a model loses, the higher the quality of that model. In estimating the amount of information lost by a model, AIC deals with the trade-off between the goodness of fit of the model and the simplicity of the model" (wikipedia 2020).

- "Bayesian Info Criterion: When fitting models, it is possible to increase the likelihood by adding parameters, but doing so may result in over-fitting. The BIC resolves this problem by introducing a penalty term for the number of parameters in the model. The penalty term is larger in BIC than in AIC. Though BIC is always higher than AIC, lower the value of these two measures, better the model" (wikipedia 2020).

- Mean Squared Error (MSE): Measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value. The MSE is a measure of the quality of an estimator—it is always non-negative, and values closer to zero are better.

- Graphical analysis of the PACF and the ACF plots for the differenced and original series.

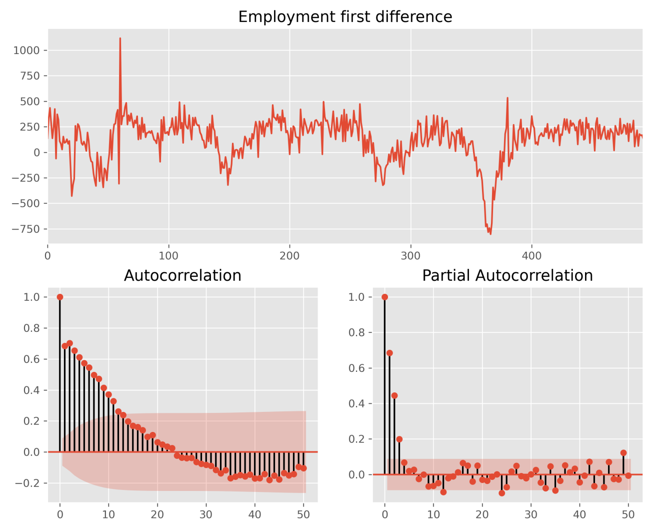


Figure 6: First difference of Employment data

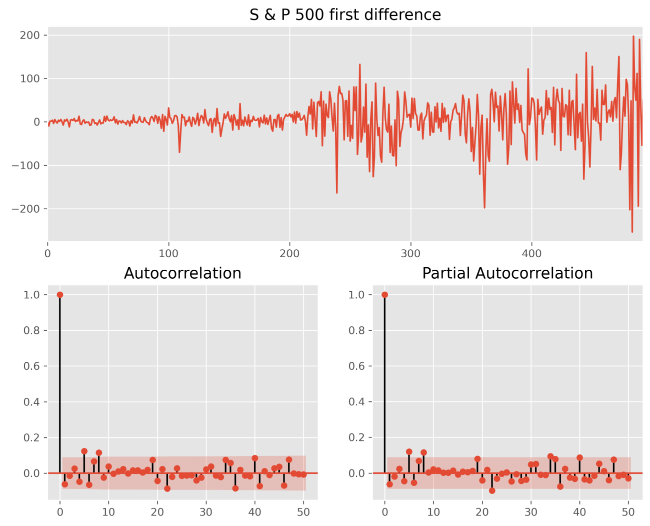


Figure 7: First difference of S&P 500 data

For the forecasting evaluation, out of sample and in sample forecast were made, taking 80% of the series for training and 20% for testing.

Finally all of in sample forecast performance was stored with its MSE and R-squared to be compared with the results produced by different Neural Networks architectures.

2.2 Deep learning Methods

For the purpose of this work a univariate forecast will be made using several types of architecture of neural networks. Four types of structures will be used:

- Multi-layer Perceptron's
- Recurrent Neural Networks
- Gate Recurrent Units
- Convolutional Neural Networks

For all of this methods the selection on basic hyperparameters has been done through a GridSearchCV and using TensorFlow Keras Tuner[24], were the following parameters were optimized:

- Number of units
- Neuron activation function
- Initial weights
- Network Optimizer
- Learning rate
- Batch size and number of epochs
- Dropout

For the number of units the following interval of values was used:
min_value=32,max_value=512, step=32, default=32

For the neuron activation function the following parameters were used:
'softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hardsigmoid', 'linear'

For the initial weights the following option were available:
'uniform', 'lecununiform', 'normal', 'zero', 'glorotnormal', 'glorotuniform', 'henormal', 'heuniform'

For the optimizer selection the following options were available:
'SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam'

For the learning rate the following parameters were used:
learn_rate = [1e-2, 1e-3, 1e-4, 1e-5]
For the Batch size and epochs the following parameters were used:
batch_size = [16,32,64,128]
epochs = [10, 50, 100]
For the dropout the following parameters were used:
min=0,max=0.5, step=0.05.

All this parameters were evaluated through a grid search looking for the minimization of the val loss. This grid search method was evaluated multiple times looking for consistency in the results obtained Once the optimal parameters were found a K-fold with time series split is used to evaluate the performance and evolution of the loss, the val_loss and the mse.

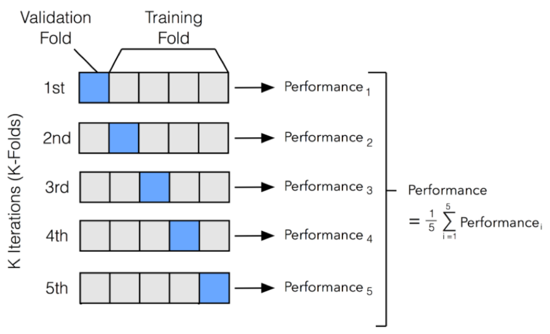


Figure 8: k-fold intuition. Source: Github

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure For the purpose of this work 5 splits have been used. Each split gives us a set of results that are then averaged to get a final result of performance indicators.
It is important to emphasize the a time series split was used for this process. Classical cross-validation techniques such as KFold and ShuffleSplit assume the samples are independent and identically distributed, and would result in unreasonable correlation between training and testing instances (yielding poor estimates of generalisation error) on time series data. Therefore, it is very important to evaluate our model for time series data on the "future"

observations least like those that are used to train the model. To achieve this, one solution is provided by TimeSeriesSplit. TimeSeriesSplit is a variation of k-fold which returns first folds as train set and the th fold as test set. Note that unlike standard cross-validation methods, successive training sets are supersets of those that come before them. Also, it adds all surplus data to the first training partition, which is always used to train the model[25]. Once this step is validated a in sample forecast is made by using 80% of the set for training and 20% for test. Final performance is measured through the mean squared error and R-squared. This two measures are used to compare different neural network architectures and classical methods.

Now that the general step for the selection of parameters has been explained, we will proceed to explain the peculiarities of the different structures used in this work. There are several bibliographical resources related to this point that have been consulted [2,3,4,5,6,7,8,9,10,11].

2.2.1 Multi-layer perceptrons

Avishek[12] defined in 2017 an MLP as follows:

> "An MLP consists of three components: an input layer, a number of hidden layers, and an output layer. An input layer represents a vector of regressors or input features , for example, observations from preceding p points in time $\{x_{t-1}, x_{t-2} .... x_{t-p}\}$.
>
> The input features are fed to a hidden layer that has n neurons, each of which applies a linear transformation and a nonlinear activation to the input features. The output of a neuron is $g_i = h(w_i x + b_i)$, where w_i and b_i are the weights and bias of the linear transformation and h is a nonlinear activation function. The nonlinear activation function enables the neural network to model complex non-linearities of the underlying relations between the regressors and the target variable. Popularly, popularly although there are many activation functions h can be the sigmoid function
>
> $$\frac{1}{1 - e^z}$$
>
> This function squashes any real number to the interval $[0, 1]$. Due to this property, the sigmoid function is used to generate binary

class probabilities and hence is commonly used in classification model.

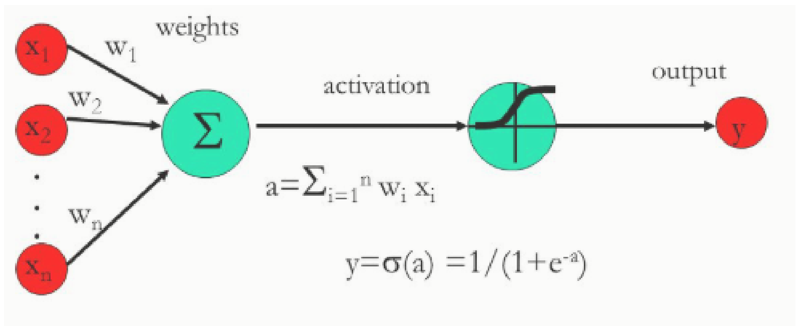


Figure 9: Basic Multi-layer perceptron. Source: Slideplayer.com

In case of a single hidden layer neural network, the output from each neuron is passed to the output layer, which applies a linear transformation and an activation function to generate the prediction of the target variable, which in case of time series forecasting, is the predicted value of the series at the t^{ht} point in time."

Description Multi-layer Structure Used

First the data is scaled using the MinMaxScaler from sklearn libraries[26]. Preprocessing the data in this way makes the gradient descent algorithm to work better, the next line show how this preprocessing was done in python:

```
scaler = MinMaxScaler(feature_range=(0, 1))
df['scaled_EM']=scaler.fit_transform(np.array(df['Employment'])
     .reshape(-1, 1))
```

Then the data is split in two parts, a train set and a test set. Here as mentioned earlier a 80% training and a 20% test is used. This process generate the following shapes of data:

Shape of train: (396, 2)
Shape of test: (98, 2)

Then regressors are generated, and a target variable, for train and test. For the time series forecasting model, the past seven months of observations are used to predict for the next month, this is equivalent to and AR(7) model. The train and test sets are generated:

```
Shape of train: (396, 2)
Shape of test: (98, 2)
Shape of train arrays: (388, 7) (388,)
Shape of train arrays: (90, 7) (90,)
```

This process will be used in all network structures in this document.

The MLP network Structure

The input layer is declared with shape (None, 7) and of type float32.

```
input_layer = Input(shape=(7,), dtype='float32')
```

Then Dense layers are declared with the activation obtained in the tuning process also n number of units are selected in function of minimizing the val_loss, all this models are created using the keras functional api[1], here default values are used for the exposition. Three dense layers are used in the employment series and four are used in the S&P 500:

```
dense1 = Dense(32,activation='relu')(input_layer)
dense2 = Dense(32, activation='relu')(dense1)
dense3 = Dense(32, activation='relu')(dense2)
```

A dense layer is a classic fully connected neural network layer : each input node is connected to each output node.

After that a $n\%$ dropout layer is used.

```
 dropout_layer = Dropout(0.2)(dense3)
```

Dropout of $n\%$ is used to dropout $n\%$ of randomly selected input features. Srivastava[13] in 2014 defined dropout as:

> "The term “dropout” refers to dropping out units (hidden and visible) in a neural network. "Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase. In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks"

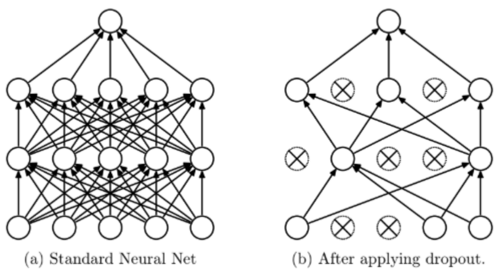


Figure 10: Dropout layer intuition. Source:[13]

Finally, the output layer gives prediction for the next month employment or S&P 500 data.

```
output_layer = Dense(1, activation='relu')(dropout_layer)
```

Model Compilation

The input and output layers are packed in a model function. Were *mse* is used to minimize the loss function. For this an optimization algorithm is used. The optimization algorithm is again selected from the process of tuning the hyperparameters choosing the one that generates the smallest val_loss.

```
optimizer = keras.optimizers.Adam(learning_rate=0.001, rho=0.9)
ts_model = Model(inputs=input_layer, outputs=output_layer)
ts_model.compile(loss='mean_squared_error', optimizer=optimizer)
ts_model.summary()
```

2.2.2 Recurrent Neural Networks

The problem with MLP is that it doesn't consider the sequential nature of the time series data where observations have correlation with each other. The correlation in a time series can also be interpreted as the memory that the series carries over itself. The RNN its said to solve or, at least alleviate this problem by keeping some information about the past data. More specifically Venkatachalam [27] defined in 2019, an RNN as follows:

> "Recurrent Neural Network remembers the past and it's decisions are influenced by what it has learnt from the past. While RNNs learn similarly while training, in addition, they remember things learnt from prior input(s) while generating output(s). It's part of the network. RNNs can take one or more input vectors and produce one or more output vectors and the output(s) are influenced not just by weights applied on inputs like a regular NN, but also by a "hidden" state vector representing the context based on prior input(s)/output(s). So, the same input could produce a different output depending on previous inputs in the series".

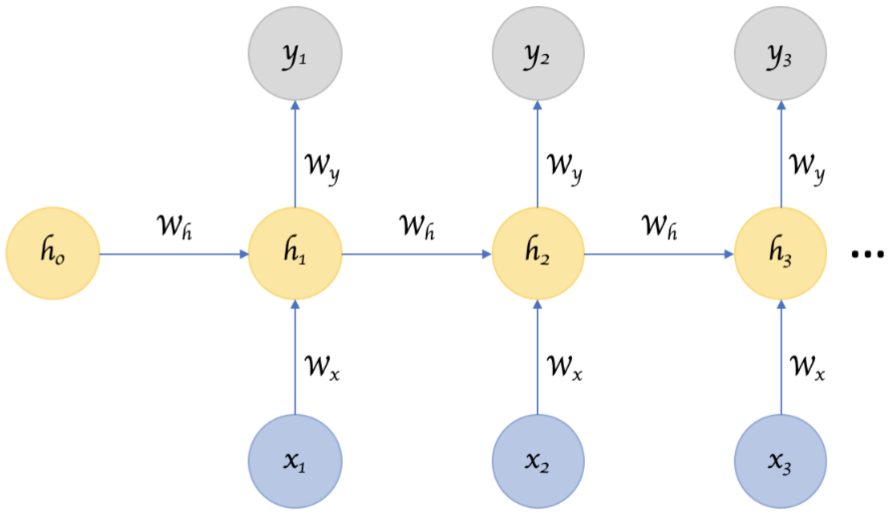


Figure 11: RNN structures. Source: towardsdatascience.com

A RNN can be used to develop a time series forecasting model where the input series $\{x_{t-1}, ..., x_{t-p}\}$ is fed to the RNN and the output from the last timestep is the prediction. RNNs are notoriously difficult to be trained.

The Vanishing Gradient Problem

According to Missinglink.ai 2016[28]

> "This is one of the most significant challenges for RNNs performance. In practice, the architecture of RNNs restricts its long-term memory capabilities, which are limited to only remembering a few sequences at a time. Consequently, the memory of RNNs is only useful for shorter sequences and short time-periods ".

The vanishing gradient problem restricts the memory capabilities of traditional RNNs adding too many time-steps increases the chance of facing a gradient problem and losing information when you use backpropagation.
As a result, RNNs have difficulty in learning long-range dependencies. For time series forecasting, going too many timesteps back in the past would be problematic. To address this problem, Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU), which are special types of RNNs are used.

Long Range Short Term Memory

Sciaga[27] in 2018 mentions:

> "One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame"

.
"Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. LSTMs are designed to overcome the vanishing gradient problem and allow them to retain information for longer periods compared to traditional RNNs. LSTMs can maintain a constant error, which allows them to continue learning over numerous time-steps and backpropagate through time and layers. LSTMs use gated cells to store information outside the regular flow of the RNN. With these cells, the network can manipulate the information in many ways, including storing information in the cells and reading from them. The cells are individually capable of making decisions regarding the information and can execute these decisions by opening or closing the gates. LSTM contains three gates: an input gate, an output gate and a forget gate. At each

iteration, the three gates try to remember when and how much the information in the memory cell should be updated. These three gates form a path of remembering the long-term dependency of the system"(Missinglink.ai[28] 2016).
The ability to retain information for a long period of time gives LSTM the edge over traditional RNNs in these tasks.

Description of RNN Used

Again we generate regressors and targets, X_train, X_val, y_train and y_val. X_train, and X_val. However, the input to RNN layers must be of shape (number of samples, number of timesteps, number of features per timestep).

Shape of train arrays: (388, 7) (388,)
Shape of train arrays: (90, 7) (90,)
Shape of 3D arrays: (388, 7, 1) (90, 7, 1)

we use here 7 month time steps and 1 feature per time step. Employment series use 1 lstm layer and S&P 500 uses 2 lstm layers.

```
input_layer = Input(shape=(7, 1), dtype='float32')
lstm_layer = LSTM(64,input_shape=(7,return_sequences=False) (input_layer)
```

The LSTM layer has seven timesteps, which is the same as the number of historical observations taken to make the next-month prediction of employment and S&P 500. Only the last timestep of the LSTM returns an output. The number of units again are defined in function of the val_loss.

Next, the LSTM's output is passed to a dropout layer that randomly drops $n\%$ of the input before passing to the output layer, which has a single hidden neuron:

```
dropout_layer = Dropout(n)(lstm_layer)
output_layer = Dense(1, activation='relu', kernel_initializer='glorot_uniform')
(dropout_layer)
ts_model = Model(inputs=input_layer, outputs=output_layer)
optimizer = keras.optimizers.Nadam(lr=0.001)
ts_model.compile(loss='mse', optimizer=optimizer)
ts_model.summary()
```

Finally, all the layers are wrapped in a tf.keras.models.model and trained for N number of epochs to minimize MSE using an optimizer.

2.2.3 Gate Recurrent Units

"The key distinction between regular RNNs and GRUs is that the latter support gating of the hidden state. This means that there are dedicated mechanisms for when a hidden state should be updated and also when it should be reset"(dive into deep learning 2020). The different gates of a GRU are as described below:

- "Update Gate(z_t): It determines how much of the past knowledge needs to be passed along into the future. It is analogous to the Output Gate in an LSTM recurrent unit.
- Reset Gate(r_t): It determines how much of the past knowledge to forget. It is analogous to the combination of the Input Gate and the Forget Gate in an LSTM recurrent unit.
- Current Memory Gate($\overline{h}_t$): It is often overlooked during a typical discussion on Gated Recurrent Unit Network. It is incorporated into the Reset Gate just like the Input Modulation Gate is a sub-part of the Input Gate and is used to introduce some non-linearity into the input and to also make the input Zero-mean. Another reason to make it a sub-part of the Reset gate is to reduce the effect that previous information has on the current information that is being passed into the future"(Geekforgeeks 2020).

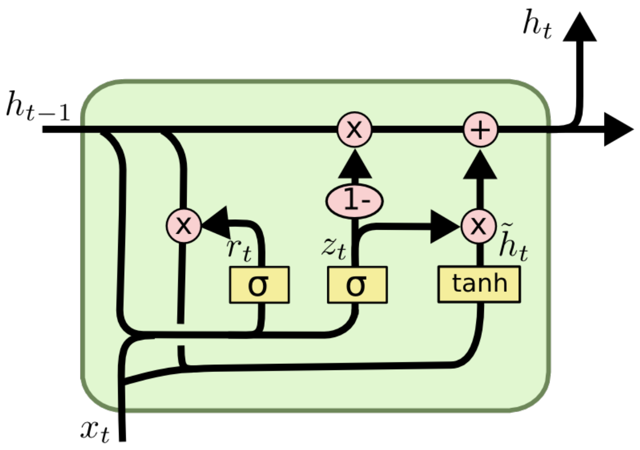


Figure 12: GRU structures. Source: data-blogger.com

Similarly to the LSTM unit, GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cell.

Description of GRU Used

It has been used two stacked GRU layers, for the employment series and 4 stacked layers for the S&P series.

```
input_layer = Input(shape=(7, 1), dtype='float32')

gru_layer1 = GRU(64, input_shape=(7, 1),
return_sequences=True)(input_layer)
gru_layer2 = GRU(32, input_shape=(7,64), return_sequences=False)(gru_layer1)
dropout_layer = Dropout(0.2)(gru_layer2)
output_layer = Dense(1, activation='linear')(dropout_layer)
ts_model = Model(inputs=input_layer, outputs=output_layer)
ts_model.compile(loss='mse', optimizer='adam', metrics=['mae', 'accuracy'])
ts_model.summary()
```

The first GRU takes a sequential input from the preceding input layer. Each timestep of the first GRU returns a n dimensional feature vector as output. This sequence is passed as input to the next GRU layer. The second GRU layer returns output only from the last timestep. The neural network is trained to minimize the MSE loss using the best optimizer selected in the grid process and the keras tuner implementation.

2.2.4 Convolutional Neural Networks

As described by Saha[30] in 2018

> "A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better".

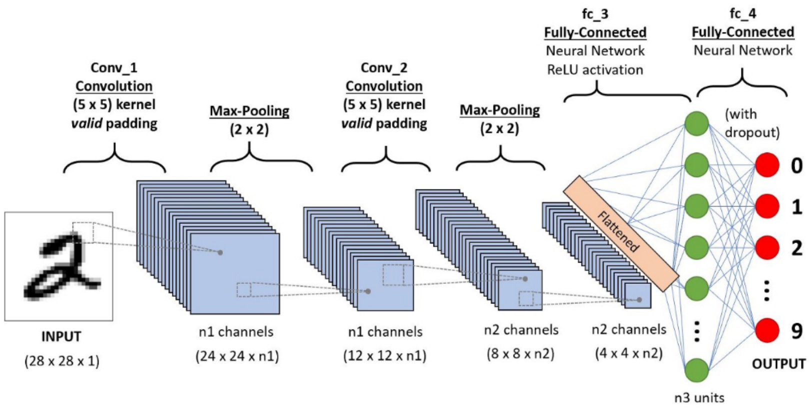


Figure 13: An example of CNN architecture. Source:[30]

The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

CNN for time series Analysis

"Although traditionally developed for two-dimensional image data, CNNs can be used to model univariate time series forecasting

> problems. Univariate time series are datasets comprised of a single series of observations with a temporal ordering and a model is required to learn from the series of past observations to predict the next value in the sequence.
>
> 1D convolution layers can be used to develop time series forecasting models. A time series having 1 x m observations is like an image of dimension *p*, which has a height of a single pixel. In this case, 1D convolution can be applied as a special case of 2D convolution using a 1 x 3 filter. Additionally, the filter is moved only along the horizontal direction by strides length of 1 x 8 time units. The approach of using a 1 x 3 convolution filter is equivalent to training several local autoregressive models of order three. These local models generate features over short-term subsets of the input time series. When an average pooling layer is used after the 1D convolution layer, it creates moving averages over the feature map generated by the preceding convolution layer. Furthermore, several 1D convolution and pooling layers, when stacked with each other, give a powerful way of extracting features from the original time series. Thus, using CNNs proves to be effective when dealing with complex, nonlinear time series such as audio waves, speech, and so on"

(Avishek[12] 2017).

Description of the ConvNet Used

Define input layer

```
input_layer = Input(shape=(7, 1), dtype='float32')
```

A ZeroPadding1D layer is added after the input layer to add zeros at the beginning and end of each series.

> "Zero padding is a technique that allows us to preserve the original input size. This is something that we specify on a per-convolutional layer basis. With each convolutional layer, just as we define how many filters to have and the size of the filters, we can also specify whether or not to use padding." (Deeplizard[31] 2020).

```
zeropadding_layer = ZeroPadding1D(padding=1)(input_layer)
```

This ensures that the convolution layer does not reduce the dimension of the output sequences.

```
conv1D_layer = Conv1D(64,3,strides=1,use_bias=True)(zeropadding_layer)
```

then a conv1d layer is added, this has 64 filters with a 1d convolution windows of length 3 that shifts in strides of 1.CNNs share the same characteristics and follow the same approach, no matter if it is 1D, 2D or 3D. The key difference is the dimensionality of the input data and how the feature detector (or filter) slides across the data.(Ackermann 2018)

```
avgpooling_layer = AveragePooling1D(pool_size=3, strides=1)(conv1_layer)
```

AveragePooling1D is added next to downsample the input by taking average over three timesteps with stride of one timestep. The average pooling in this case can be thought of as taking moving averages over a rolling window of three time units.

> "An average pooling layer performs down-sampling by dividing the input into rectangular pooling regions and computing the average values of each region. Pooling layers follow the convolutional layers for down-sampling, hence, reducing the number of connections to the following layers" (MathWorks 2020).

```
flatten_layer = Flatten()(avgpooling_layer)
```

The flatten layer reshapes the data received from the pooling layer and pass the data to the dropout layer, so that it is finally processed

```
dropout_layer = Dropout(0.2)(flatten_layer)
output_layer = Dense(1, activation='linear',
kernel_initializer='he_uniform')(dropout_layer)
ts_model = Model(inputs=input_layer, outputs=output_layer)
optimizer = keras.optimizers.adadelta(lr=0.001)
ts_model.compile(loss='mean_absolute_error', optimizer=optimizer, metrics=[
                  'mse', 'accuracy'])   SGD(lr=0.001, decay=1e-5))
ts_model.summary()
```

finally an optimizer is used to minimize the Mean Absolute Error.

3 Experimental results

The data Set

The data set used consist of 494 monthly observations of a financial and economic series ranging form August 1978 to September 2019. Both series are positive correlated as measured by its correlation Coefficient of 0.94.

Table 3: Descriptive Statistics

Statistic	S&P 500	Employment
Average	958.81	120584.85
STD Deviation	735.55	18456.72
Asimetry	0.79	-0.33
Kurtosis	-0.05	-1.15
Max	2980.38	151722.00
Min	93.15	87483.00
Coeficient of Variation	0.77	0.15
Number of Observations	494	494

However as explained initially this series differ greatly in is volatility and trend component which is a usual characteristic difference between financial and economic time series, this can be appreciated in the big difference between the coefficient of variation.

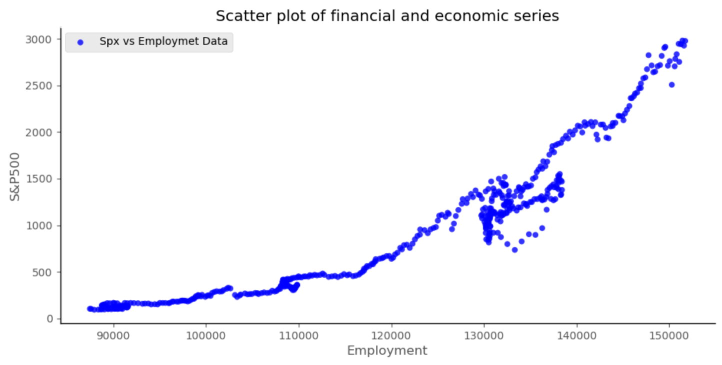


Figure 14: Relationship between S&P 500 and Labor Data

As the above figure shows, they have a very strong positive correlation. This has a very intuitive economic explanation. Job creation is strongly correlated with the strength of expansions in GDP. The strength of GDP is associated with an improvement in company profits. The value of a stock index is based on the value of discounted earnings.

Table 4: Evolution of variance across time

S&P 500		Employment Data	
Mean 1	Mean 2	Mean 1	Mean 2
368.35	1549.28	104,708.93	136,460.77
Variance 1	Variance 2	Variance 1	Variance 2
73,170.32	309,399.44	136,946,210.00	38,885,859.18

However we can see this data is not stationary and we can check this by start looking at how the variance and mean behaves across different samples.

When analyzed the ljung-box test for both time series we have the following results:

Table 5: LjungBox Test

Up to lag 20	Employment	S&P 500
Original	Rejected	Rejected
First Difference	Rejected	Accepted

we can reject the null hypothesis that autocorrelations up to lag k equal zero (that is, the data values are random and independent up to a certain number of lags–in this case 20) If p-value < 0.051: we can reject the null hypothesis assuming a 5% chance of making a mistake. So we can assume that our values are showing dependence on each other. If p-value > 0.051: we don't have enough statistical evidence to reject the null hypothesis. When we reject we can suspect that autocorrelations for one or more lags might be significantly different from zero, indicating the values are not random and independent over time, which is one of the assumptions of box Jenkins model selection process.

Table 6: Dicket fuller test

Statistic	S&P 500	Employment Data
Dickey Fuller Test	1.1630	-0.4380
p-val	0.9957	0.9035
1% Critical Value	-3.4439	-3.4437
5% Critical Value	-2.8675	-2.8674
10% Critical Value	-2.5699	-2.5699

The p-value is obtained is greater than significance level of 0.05 and the ADF statistic is higher than any of the critical values. Clearly, there is no reason to reject the null hypothesis. So, the time series is in fact non-stationary. The null hypothesis that the model of order 1 without a constant offset and no deterministic trend contains a unit root is no rejected at the 0.5 percent level based on the Dickey-Fuller T test. Based on both of this statistics we can conclude that this series are non stationary. So at least one level of integration has been used when applied autoregressive models to the data.

3.1 Arima Model Selection

As discussed earlier based on Box-Jenkins methodology model selection criteria would be based on AIC and BIC criteria.
The results obtained for the information criteria are detailed below:

Table 7: Employment

AR	I	MA	AIC	AR	I	MA	BIC
1	2	1	6220.29	1	2	1	6237.08
0	2	2	6220.58	0	2	2	6237.37
1	1	2	6220.90	0	2	1	6239.62
2	1	1	6221.33	1	1	2	6241.90
2	2	1	6222.27	2	1	1	6242.33
1	2	2	6222.27	2	2	1	6243.26
2	1	2	6222.88	1	2	2	6243.26
2	2	2	6224.29	2	2	0	6244.10
0	2	1	6227.02	1	1	1	6246.81
2	2	0	6227.30	2	1	2	6248.08
1	1	1	6230.01	2	2	2	6249.48
2	1	0	6240.91	2	1	0	6257.71
1	2	0	6255.57	1	2	0	6268.16
1	1	0	6345.91	1	1	0	6358.51
0	1	2	6405.75	0	1	2	6422.55
0	2	0	6415.98	0	2	0	6424.38
0	1	1	6503.63	0	1	1	6516.23
0	1	0	6654.50	0	1	0	6662.90

Table 8: S&P 500

AR	I	MA	AIC	AR	I	MA	BIC
2	2	2	5176.93	0	2	1	5194.33
1	2	2	5178.94	0	1	0	5195.29
1	2	1	5181.44	1	2	1	5198.23
0	2	1	5181.73	0	1	1	5199.52
2	1	1	5182.58	1	1	0	5199.58
2	2	1	5183.11	1	2	2	5199.94
0	1	0	5186.88	2	2	2	5202.12
0	1	1	5186.92	2	1	1	5203.59
1	1	0	5186.98	2	2	1	5204.10
2	1	2	5187.45	2	1	0	5205.61
2	1	0	5188.81	0	1	2	5205.64
0	1	2	5188.84	1	1	1	5205.67
1	1	1	5188.87	2	1	2	5212.66
2	2	0	5323.14	2	2	0	5339.94
1	2	0	5393.46	1	2	0	5406.05
0	2	0	5547.61	0	2	0	5556.01

"The Akaike Information Criterion, or AIC for short, is a method for scoring and selecting a model. The AIC statistic is defined for logistic regression as follows

$$AIC = -2/N * LL + 2 * k/N \tag{1}$$

Where N is the number of examples in the training dataset, LL is the log-likelihood of the model on the training dataset, and k is the number of parameters in the model.

The score, as defined above, is minimized, e.g. the model with the lowest AIC is selected.

The Bayesian Information Criterion, or BIC for short, is a method for scoring and selecting a model.
The BIC statistic is calculated for logistic regression as follows:

$$BIC = -2 * LL + log(N) * k \tag{2}$$

Where log() has the base-e called the natural logarithm, LL is the log-likelihood of the model, N is the number of examples in the training dataset, and k is the number of parameters in the model".(Brownlee 2019)

The score as defined above is minimized, e.g. the model with the lowest BIC is selected.

Also Based on the MSE an ordering of the best results has been made for the ARIMA structure.

Table 9: Employment by MSE

AR	I	MA	MSE
2	0	0	1418.88
1	0	1	5353.54
1	0	0	5468.89
1	1	1	6901.84
3	1	1	6966.74
1	1	2	6968.81
2	1	1	6970.64
2	1	2	6978.39
0	2	1	7028.00
0	2	2	7085.63
3	1	0	7098.83
1	2	1	7101.38
3	2	1	7101.55
1	2	2	7106.31
2	2	1	7106.81
2	2	2	7114.08
3	2	2	7114.18
3	2	0	7290.39
2	0	1	7292.03
2	0	2	7396.76
3	0	1	7406.54
3	0	2	7414.39
2	2	0	7443.04
2	1	0	7847.19
1	2	0	8579.66
3	0	0	8664.69
1	0	2	9156.59
1	1	0	9207.34
0	1	2	9669.75
0	1	1	9999.81
0	1	0	11737.66
0	2	0	11763.05
0	0	1	153362312.91
0	0	0	606265627.10

Table 10: S&P 500 by MSE

AR	I	MA	MSE
0	2	0	3156.95
0	2	1	5412.30
0	1	0	5465.33
1	2	1	5490.42
1	2	2	5523.07
1	1	0	5532.54
0	1	1	5539.38
2	2	2	5556.57
2	2	1	5582.42
2	1	0	5619.85
1	0	0	5625.65
0	1	2	5627.19
3	1	2	5640.58
3	2	1	5669.49
3	1	0	5728.55
3	1	1	5750.01
2	1	1	5762.10
3	0	0	5801.42
2	0	0	6191.91
3	2	0	7758.11
2	2	0	7929.49
1	2	0	8850.41
0	0	1	482556.86
0	0	0	1820059.59

Based on the results obtained above the following model specifications were tested:

Table 11: Arima models selected

Employment			S &P 500		
AR	I	MA	AR	I	MA
2	0	0	0	2	1
1	2	1	2	2	2
0	2	2	1	2	2
1	1	2	1	2	1

This lead us to the following results:

Table 12: Employment Series

Order	MSE	RMSE	R2
Arima(1,1,2)	6876.2711	82.9232	0.9998
Arima(0,2,2)	6964.3443	83.4526	0.9997
Arima (1,2,1)	7001.8839	83.6772	0.9997
Arima(2,0,0)	11395.5151	106.74977	0.9996
Average	**8059.5036**	**89.2007**	**0.9997**

For the employment series we can see that the best model is of the order (1,1,2). We can plot the results as follows:

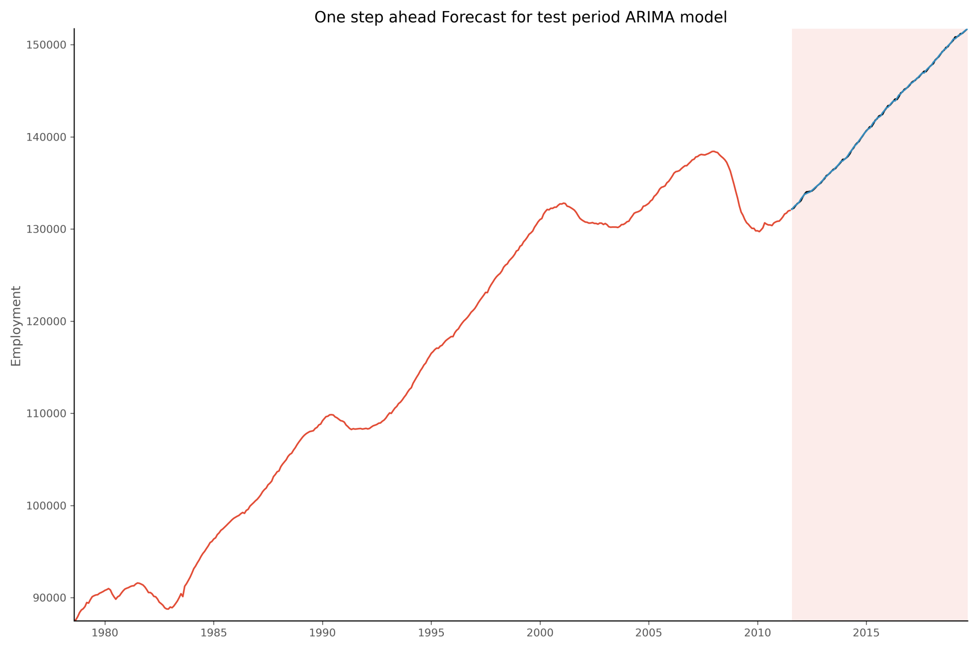


Figure 15: Employment Forecast Arima(1,1,2)

For the S&P500 series we get the following results:

Table 13: S&P 500 Series

Order	MSE	RMSE	R2
Arima(0,2,1)	5457.0651	73.8719	0.9787
Arima(1,2,1)	5537.5709	74.4148	0.9894
Arima(1,2,2)	5567.4915	74.6156	0.9893
Arima(2,2,2)	5587.7679	74.7513	0.9782
Average	**5537.4738**	**74.4134**	**0.9839**

For the S&P 500 series we can see that the best model is of the order (0,2,1). We can plot the results as follows:

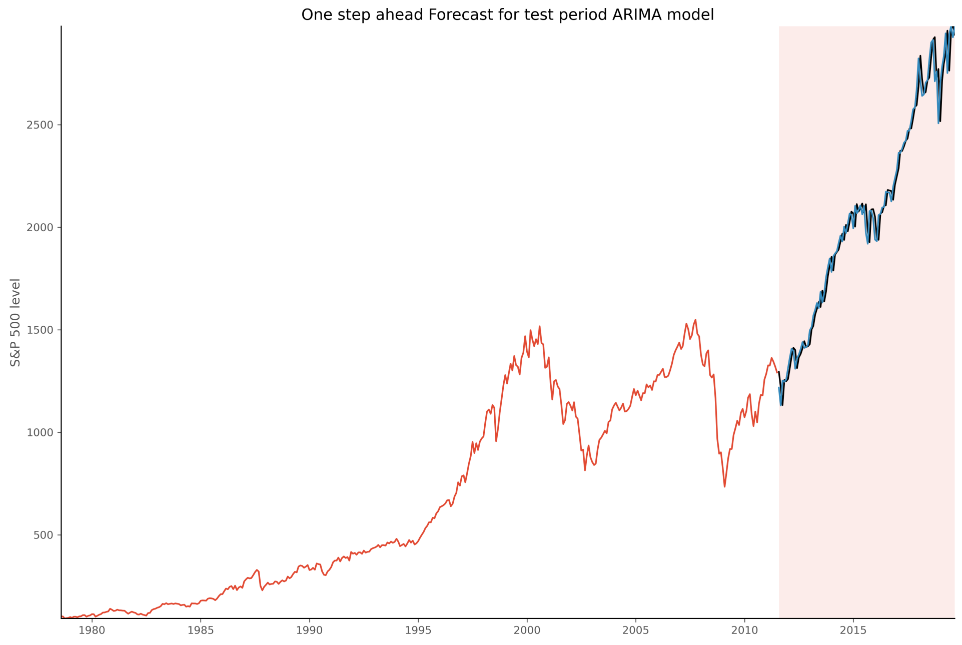


Figure 16: Employment Forecast Arima(1,1,2)

3.2 Deep Learning Forecast

GridSearch Results

As stated before one of the methods for the tuning of hyperparameter was a grid search. This was done for the following parameters:

- Neuron activation Function
- Initial weights
- Optimizer Selection
- Learning rate and momentum
- Batch size and number of epochs

The results are as follow:

Table 14: Hyperparameters for employment series

	RNN_EM	MLP_EM	GRU_EM	CNN_EM
Activation	linear	linear	linear	linear
Initial Weigths	glorot uniform	glorot uniform	normal	normal
Optimizer	Adam	Nadam	Nadam	Adam
Learning Rate	0.0001	0.001	0.01	0.01
Momentum	0	0	0	0
Batch size	10	10	60	60
Epochs	100	10	100	100

Table 15: Hyperparameters for S&P 500 series

	RNN_SPX	MLP_SPX	GRU_SPX	CNN_SPX
Activation	relu	linear	linear	linear
Initial Weigths	normal	glorot uniform	glorot uniform	normal
Optimizer	Adam	Nadam	RMSprop	SGD
Learning Rate	0.0001	0.001	0.001	0.01
Momentum	0	0	0	0
Batch size	10	10	128	60
Epochs	100	10	100	100

This yielded the following k-fold results:

Table 16: RNN K-fold Result for Employment series

	RNN_EM
Average_loss	0.01409
Average_val_loss	0.00278
Average R2	0.93470
Average MSE_level	1,937,755.18

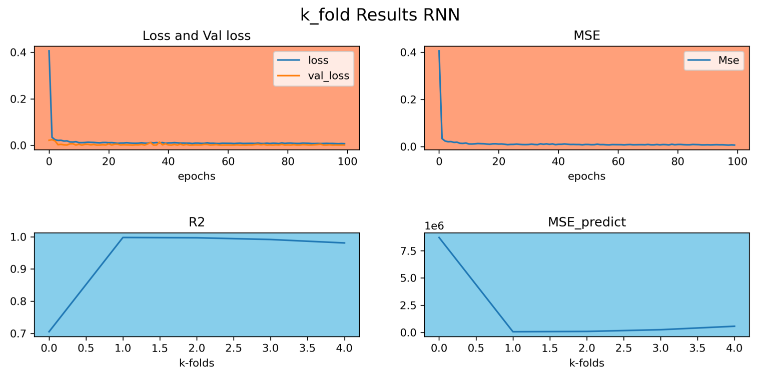


Figure 17: K-fold for RNN in Employment series

Table 17: RNN for S&P 500 series

	RNN_SPX
Average_loss	0.06130
Average_val_loss	0.07627
Average R2	0.82660
Average MSE_level	37,522.58

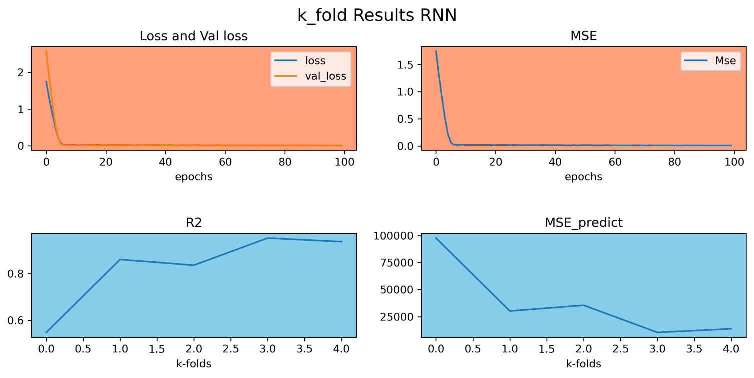


Figure 18: K-fold for RNN in S&P 500 series

Table 18: MLP for Employment series

	MLP_EM
Average_loss	0.20139
Average_val_loss	0.02452
Average R2	0.92323
Average MSE_level	2,270,390.32

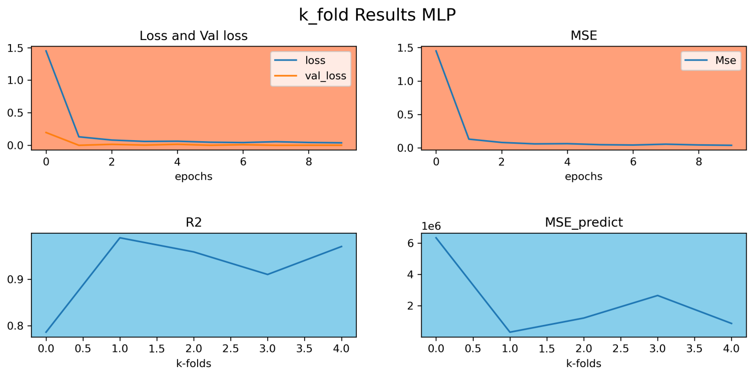


Figure 19: K-fold for MLP in Employment series

Table 19: MLP for S&P series

	MLP_SPX
Average_loss	0.09238
Average_val_loss	0.01849
Average R2	0.91230
Average MSE_level	18,974.14

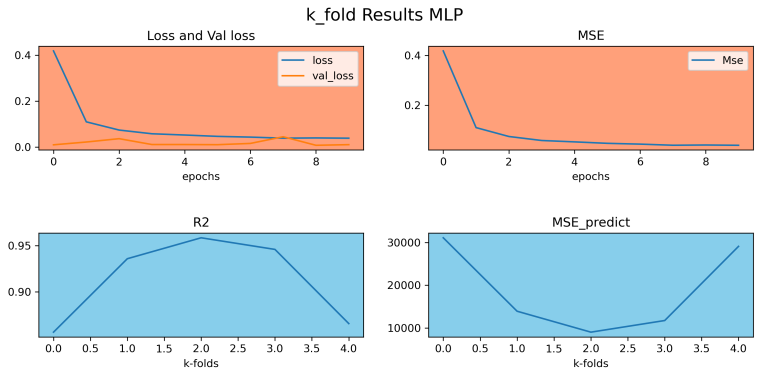


Figure 20: K-fold for MLP in S&P 500 series

Table 20: GRU for EM series

	GRU_EM
Average_loss	0.01993
Average_val_loss	0.00399
Average R2	0.95656
Average MSE_level	1,374,085.00

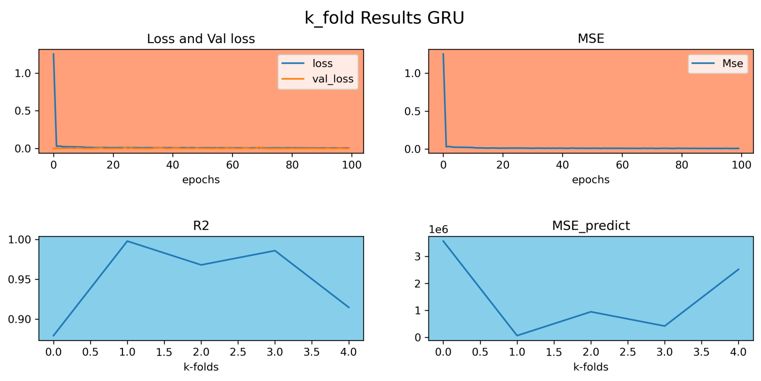


Figure 21: K-fold GRU for Employment series

Table 21: GRU for SPX series

	GRU_SPX
Average_loss	0.04272
Average_val_loss	0.07479
Average R2	0.84200
Average MSE_level	34,199.11

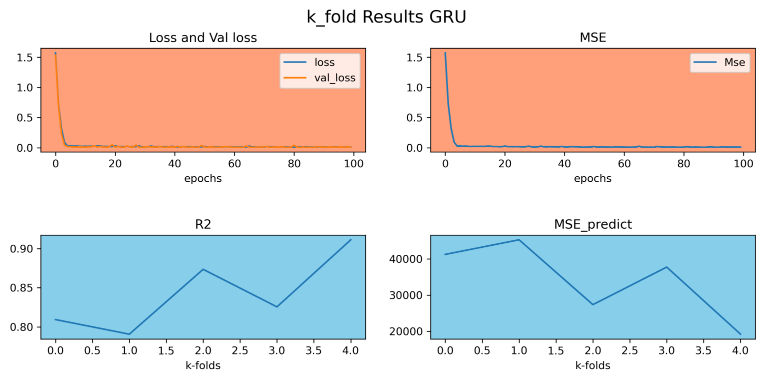


Figure 22: K-fold GRU for S&P 500 series

Table 22: CNN for EM series

	CNN_EM
Average_loss	0.06600
Average_val_loss	0.05844
Average R2	0.97644
Average MSE_level	696,668.03

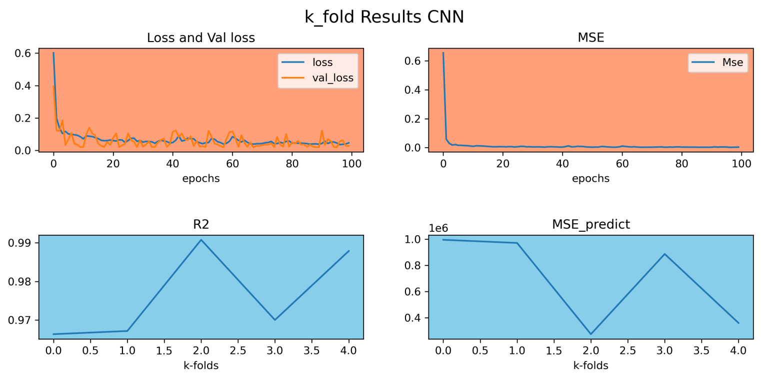


Figure 23: K-fold CNN for Employment series

Table 23: CNN for SPX

	CNN_SPX
Average_loss	0.08110
Average_val_loss	0.09155
Average R2	0.92617
Average MSE_level	15,985.03

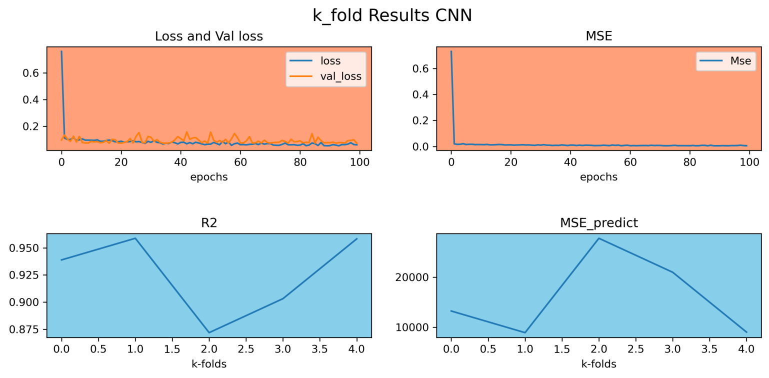


Figure 24: K-fold CNN for S&P 500 series

Keras tuner results

Additionally tensorflow keras tuner was used for hyperparameter tuning yielding very different results than previous k-folds. We show the results with keras tuner:

Recurrent Neural Network

For the employment series the final RNN structure used:

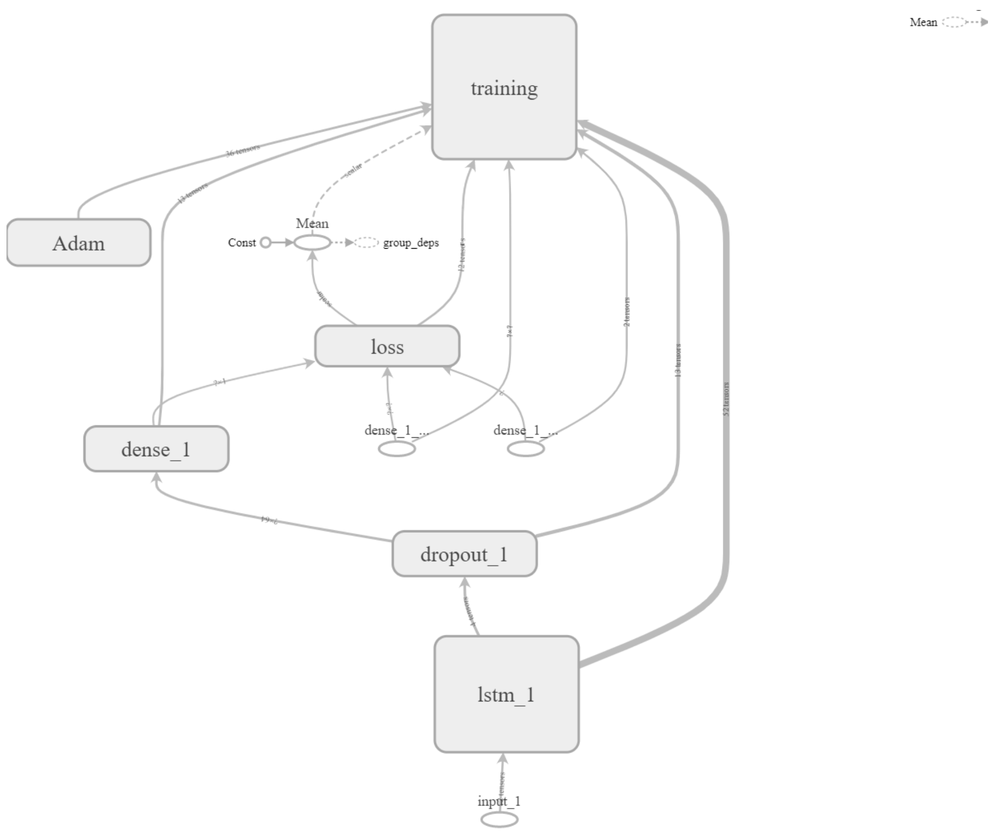


Figure 25: Final RNN structure

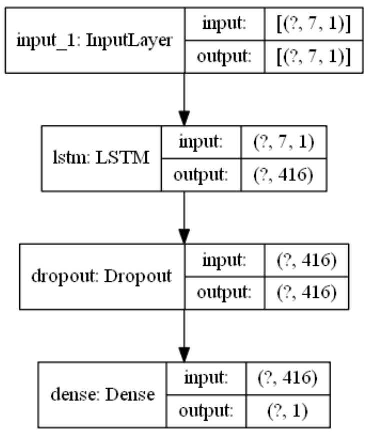


Figure 26: Final RNN structure

With hyperparameters:

Table 24: Hyperparameters for Employment LSTM

Initializer:	he_uniform
activation:	softplus
batch_size:	32
dropout_2:	0.45
epochs:	50
learning_rate:	0.01
optimizer:	Adam
units:	416

This generates the following forecast:

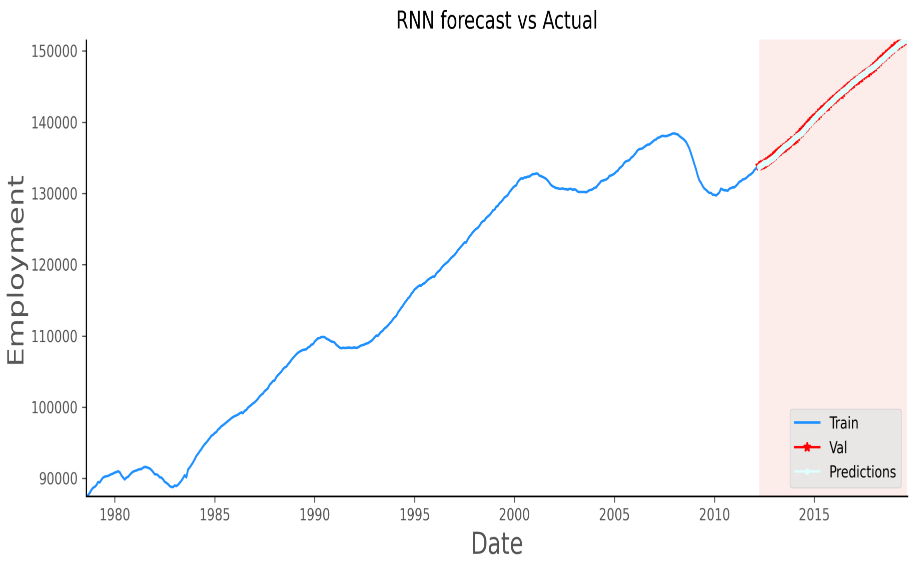


Figure 27: Final RNN Forecast for Employment series

with the following statistics:

Table 25: RNN for Employment

R2	0.9996
MSE	12157.61

For the S&P500 series we have:

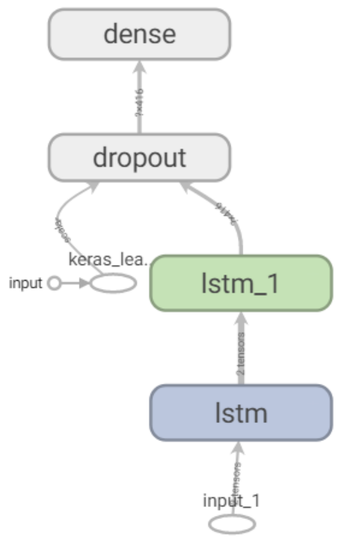


Figure 28: Final RNN structure

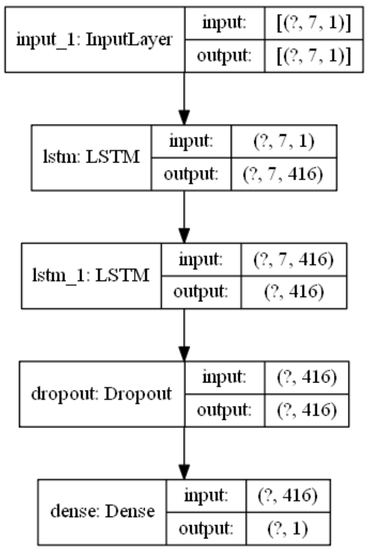


Figure 29: Final RNN structure

This LSTM has the following hyperparameters:

Table 26: Hyperparameters for S&P LSTM

Initializer:	he_uniform
activation:	softsign
batch_size:	32
dropout:	0.2
epochs:	10
learning_rate:	0.01
optimizer:	Adamax
units:	480

This generates the following forecast:

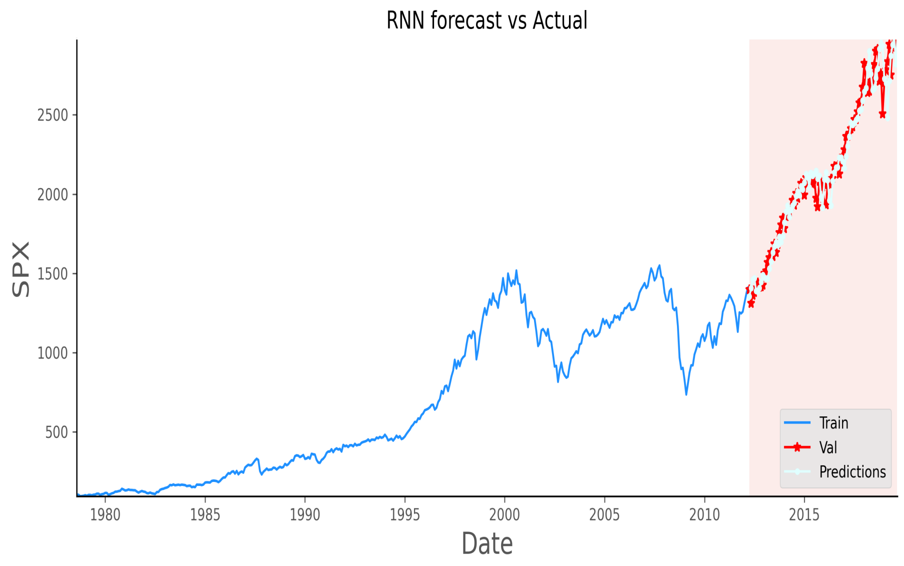


Figure 30: Final RNN Forecast for S&P 500 series

With the following Statistic:

Table 27: RNN for S&P series

R2	0.9456
MSE	11787.75

Multi-layer Perceptron

The structure used for the MLP in the Employment series is:

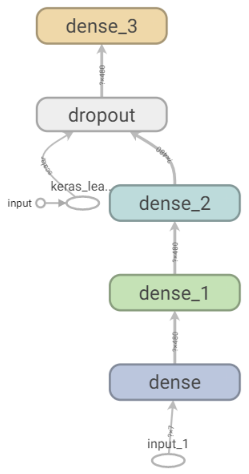


Figure 31: Final MLP structure

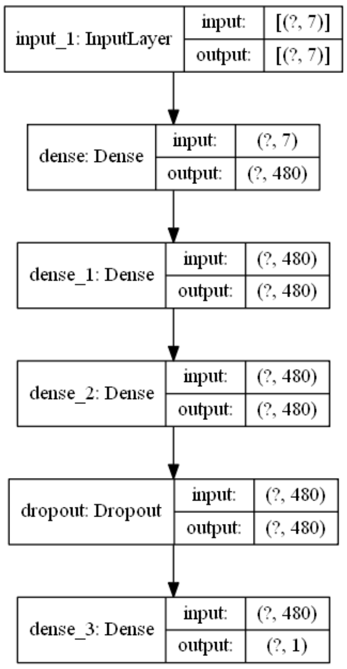


Figure 32: Final MLP structure

The following hyperparameters were used:

Table 28: Hyperparameter for Employment MLP

Initializer:	uniform
activation:	sigmoid
batch_size:	16
dropout:	0.45
epochs:	50
learning_rate:	0.001
optimizer:	Adam
units:	480

This generates the following forecast:

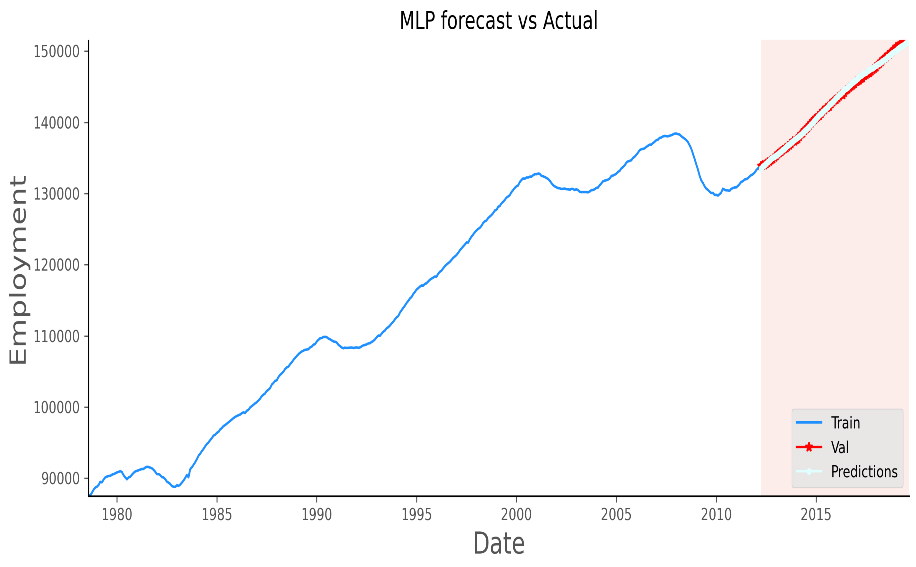


Figure 33: Final MLP Forecast for Employment series

which gives the following statistics:

Table 29: MLP for Employment series

R2	0.9982
MSE	54631.05

The MLP for the S&P 500 series has the following structure:

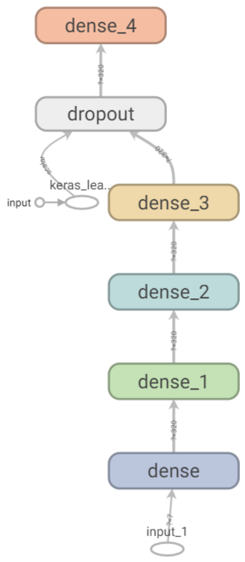


Figure 34: Final MLP structure

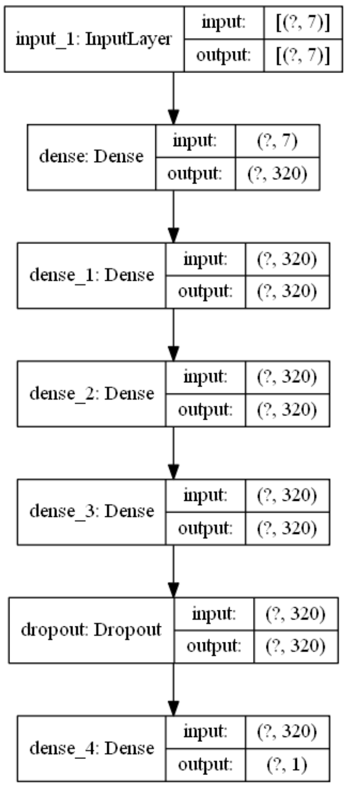

input_1: InputLayer | input: | [(?, 7)] | output: | [(?, 7)]
dense: Dense | input: | (?, 7) | output: | (?, 320)
dense_1: Dense | input: | (?, 320) | output: | (?, 320)
dense_2: Dense | input: | (?, 320) | output: | (?, 320)
dense_3: Dense | input: | (?, 320) | output: | (?, 320)
dropout: Dropout | input: | (?, 320) | output: | (?, 320)
dense_4: Dense | input: | (?, 320) | output: | (?, 1)

Figure 35: Final MLP structure

The following hyperparameters were used:

Table 30: Hyperparameters for S&P MLP

Initializer:	he_uniform
activation:	softplus
batch_size:	128
dropout:	0.5
epochs:	10
learning_rate:	0.0001
optimizer:	Adam
units:	320

This generates the following forecast:

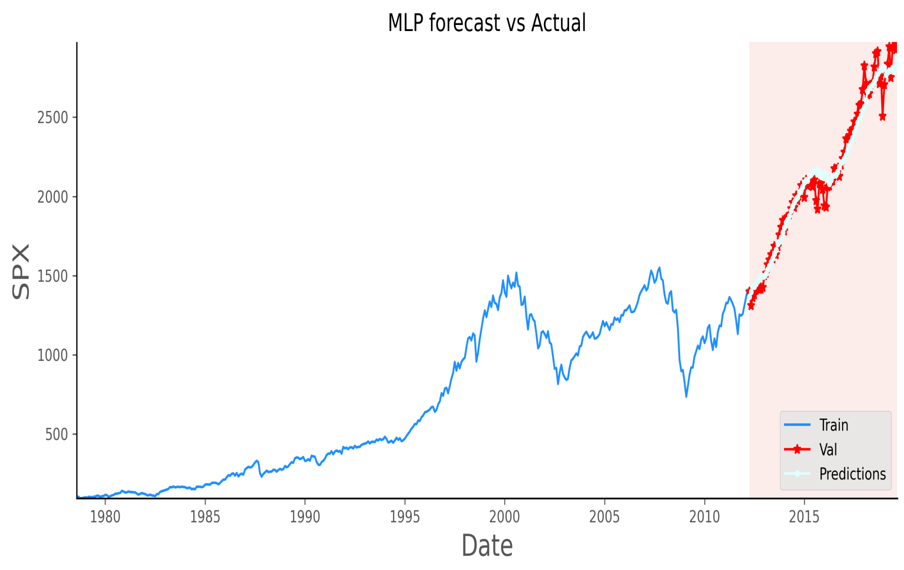


Figure 36: Final MLP Forecast for S&P 500 series

This generates the following statistics:

Table 31: MLP for S&P series

R2	0.9662
MSE	7322.57

Gate Recurrent Units

The structure used for the GRU for the Employment series is:

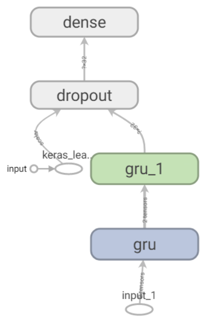


Figure 37: Final GRU structure

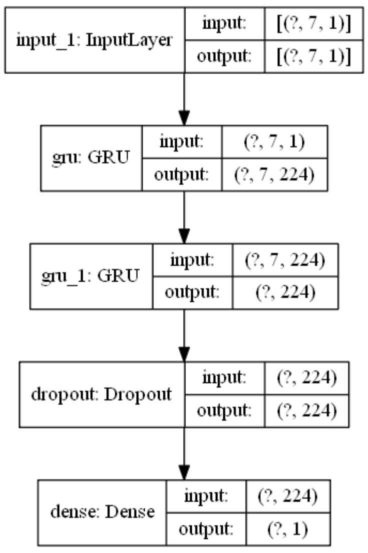


Figure 38: Final GRU structure

This generates the following forecast:

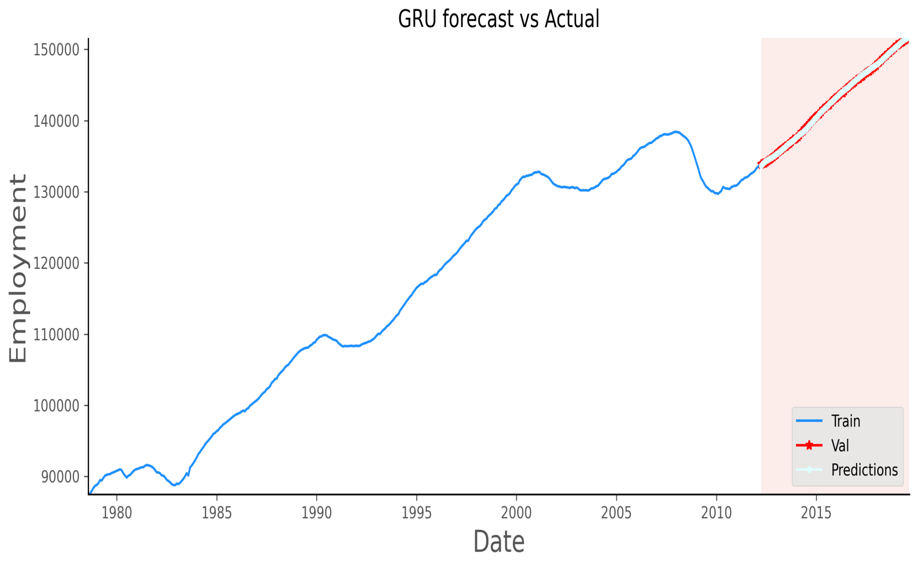


Figure 39: Final GRU Forecast for Employment series

The following hyperparameters were used:

Table 32: Hyperparameters for Employment GRU

Initializer:	uniform
activation:	softsign
batch_size:	16
dropout:	0.2
epochs:	10
learning_rate:	1.00E-05
optimizer:	Adam
units:	224

This generates the following statistics:

Table 33: GRU for Employment

R2	0.9994
MSE	18559.20

The structure used for the S&P series is:

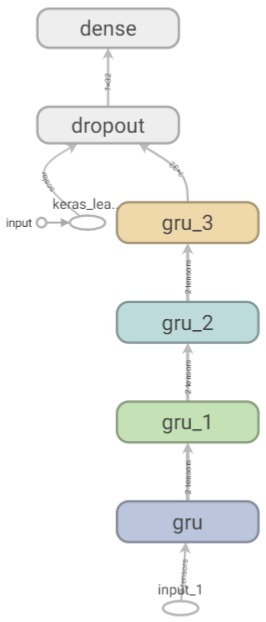


Figure 40: Final GRU structure

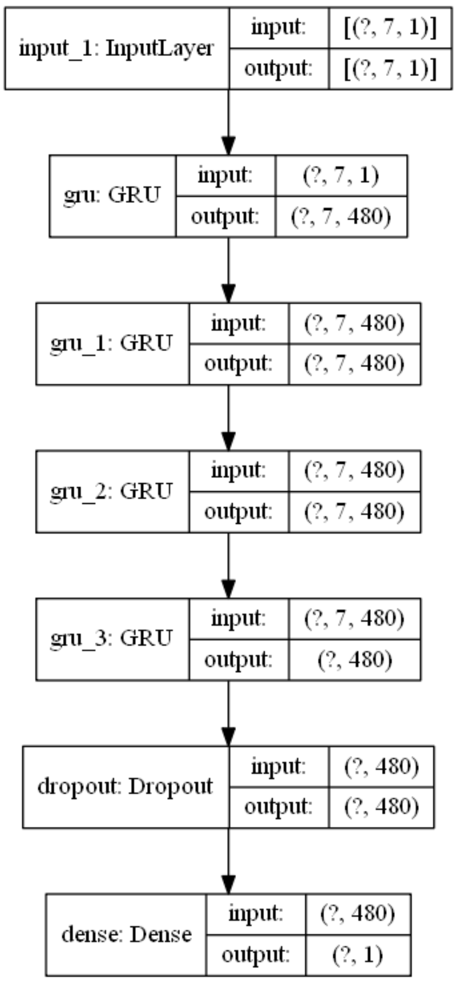


Figure 41: Final GRU structure

The following hyperparameters were used:

Table 34: Hyperparameters for S&P GRU

Initializer:	he_normal
activation:	softsign
batch_size:	64
dropout:	0.1
epochs:	10
learning_rate:	1.00E-05
optimizer:	Adamax
units:	480

This generates the following forecast:

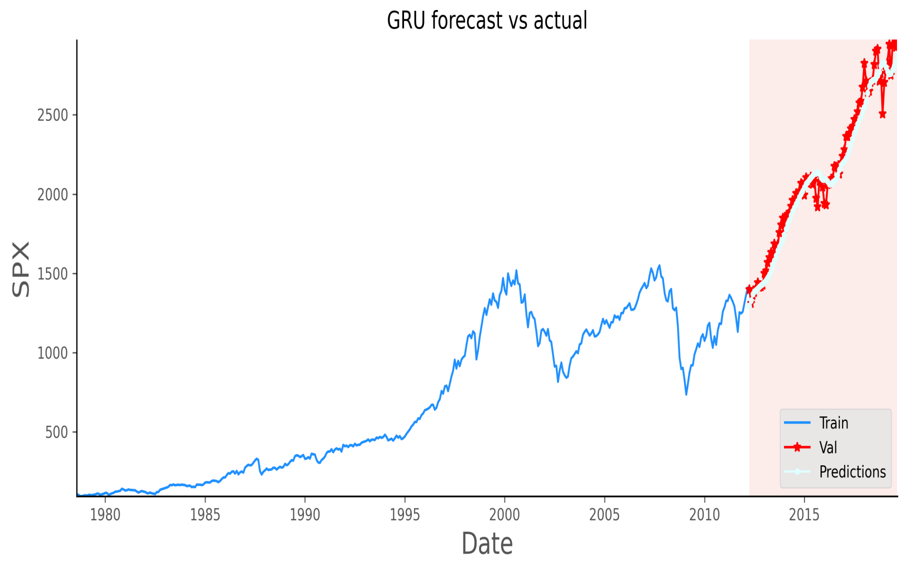


Figure 42: Final GRU Forecast for S&P 500 series

This generate the following statistics:

Table 35: GRU for S&P 500

R2	0.958
MSE	9098.26

Convolutional Neural Networks

The structure used for the Employment series is:

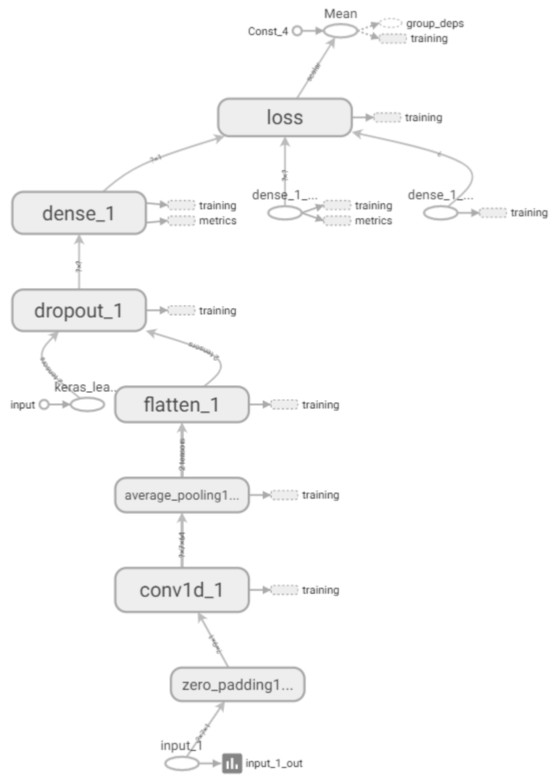


Figure 43: Final CNN structure

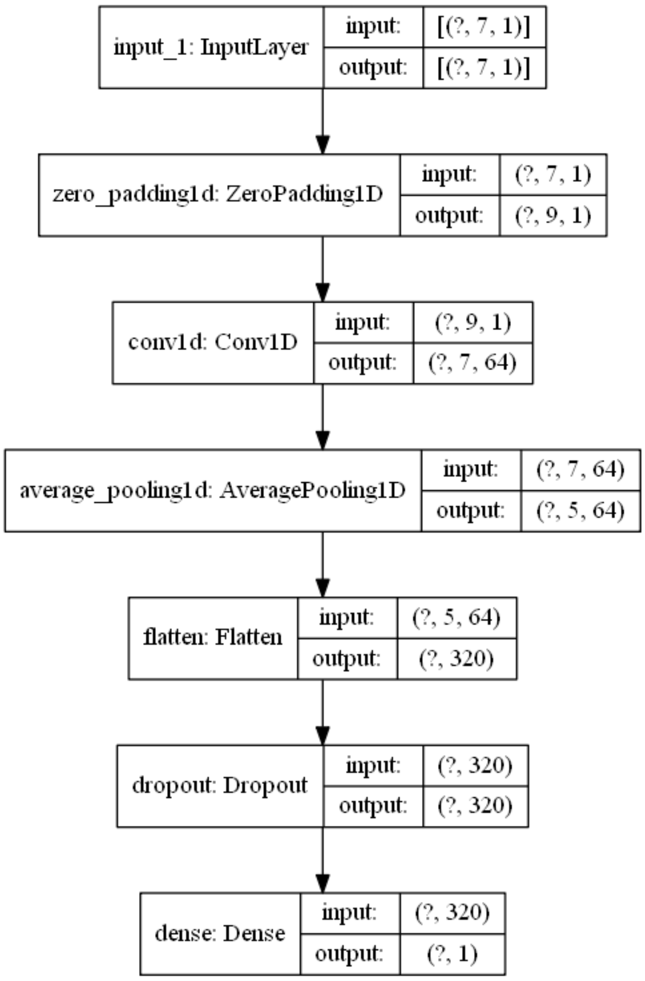


Figure 44: Final CNN structure

The following hyperparameters were used:

Table 36: Hyperparameters for the Employment CNN

Initializer:	he_uniform
activation:	relu
batch_size:	64
dropout:	0.3
epochs:	100
learning_rate:	0.001
optimizer:	SGD
units:	64

This generates the following forecast:

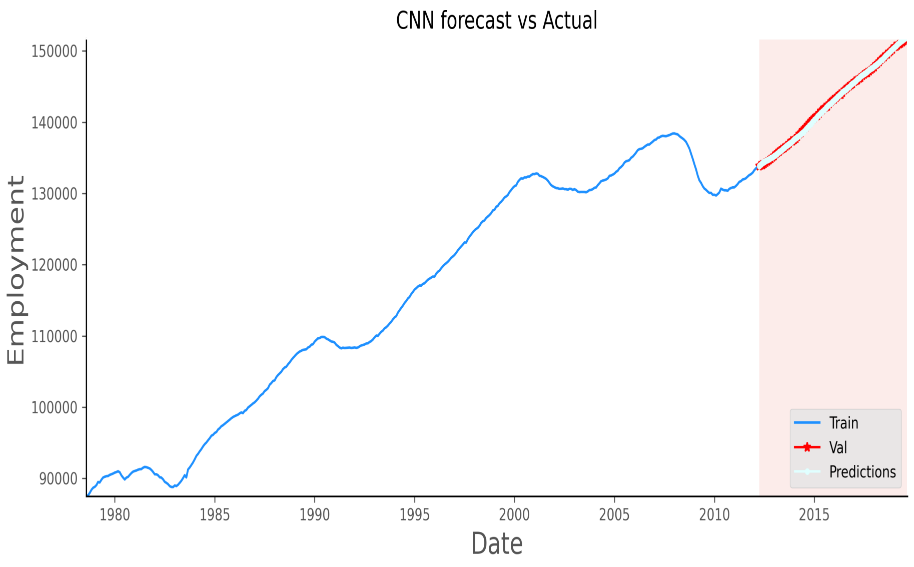


Figure 45: Final CNN Forecast for Employment series

This generates the followings statistics:

Table 37: CNN for Employment series

R2	0.9991
MSE	26803.42

The final structure for S&P series was:

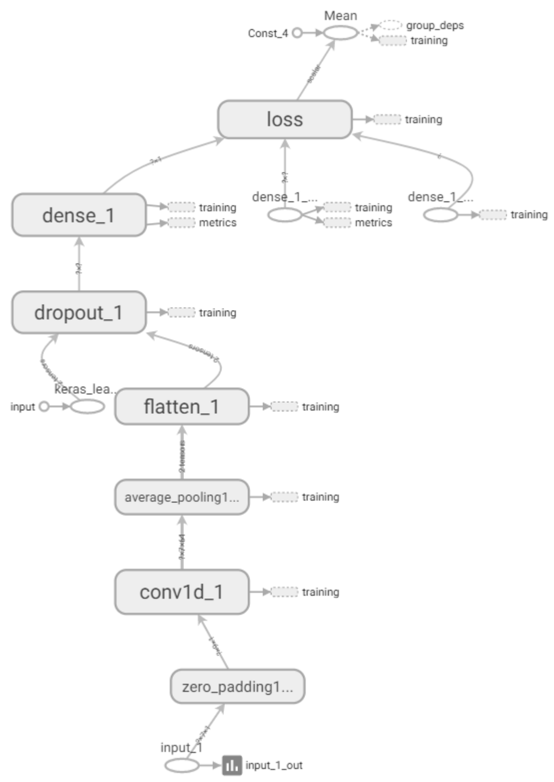


Figure 46: Final CNN structure

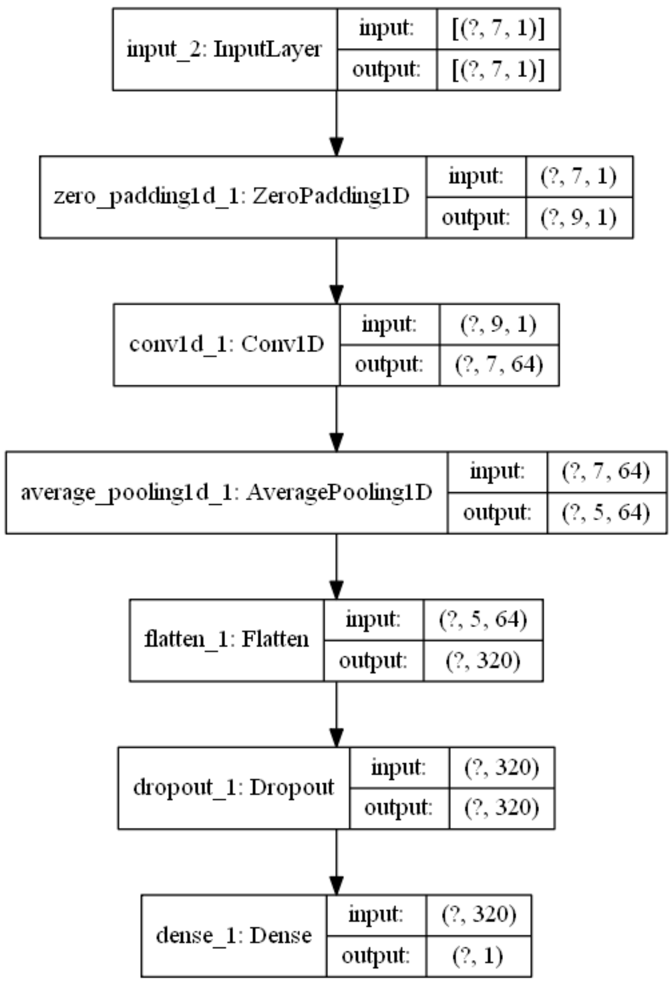


Figure 47: Final CNN structure

The following hyperparameters were used:

Table 38: Hyperparameters for the S&P CNN

Initializer:	glorot_uniform
activation:	relu
batch_size:	128
dropout:	0.15
epochs:	100
learning_rate:	0.001
optimizer:	SGD
units:	64

This generates the following forecast:

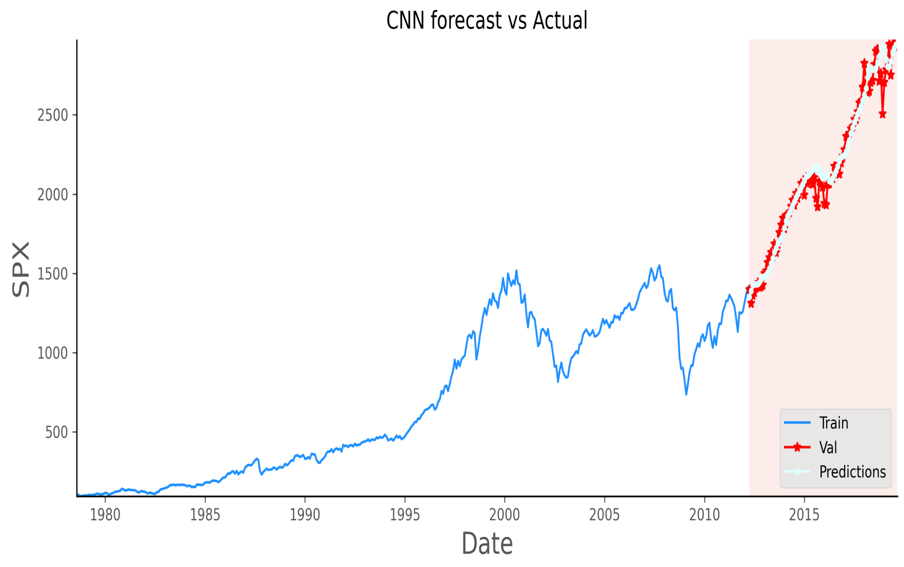


Figure 48: Final CNN Forecast for S&P 500 series

This generates the followings statistics:

Table 39: CNN for Employment series

R2	0.9623
MSE	8162.00

In summary the hyperparameters used based on Keras tuner selection are:

Table 40: Summary of Hyperparameters

	RNN_EM	MLP_EM	GRU_EM	CNN_EM
Initializer	he_uniform	uniform	uniform	he_uniform
Activation	softplus	sigmoid	softsign	relu
Batch_size	32	16	16	64
Dropout	0.45	0.45	0.2	0.3
Epochs	50	50	10	100
Learning Rate	0.01	0.001	1.00E-05	0.001
Optimizer	Adam	Adam	Adam	SGD
Units	416	480	224	64
	RNN_SPX	MLP_SPX	GRU_SPX	CNN_SPX
Initializer	he_uniform	he_uniform	he_normal	glorot_uniform
Activation	softsign	softplus	softsign	relu
Batch_size	32	128	64	128
Dropout	0.2	0.5	0.1	0.15
Epochs	10	10	10	100
Learning Rate	0.01	0.0001	1.00E-05	0.001
Optimizer	Adamax	Adam	Adamax	SGD
Units	480	320	480	64

Table 41: Summary of results

	RNN_EM	MLP_EM	GRU_EM	CNN_EM	Average
R2	0.9996	0.9982	0.9994	0.9991	**0.999075**
MSE	12157.61	54631.05	18559.2	26803.42	**28037.82**
	RNN_SPX	MLP_SPX	GRU_SPX	CNN_SPX	Average
R2	0.9456	0.9662	0.958	0.9623	**0.958025**
MSE	11787.75	7322.57	9098.26	8162	**9092.645**

This results must be compared with ARIMA results that were previously exposed

Table 42: Employment Series

Order	Mse	Rmse	R2
Arima(1,1,2)	6876.2711	82.9232	0.9998
Arima(0,2,2)	6964.3443	83.4526	0.9997
Arima (1,2,1)	7001.8839	83.6772	0.9997
Arima(2,0,0)	11395.5151	106.74977	0.9996
Average	**8059.5036**	**89.2006925**	**0.9997**

Table 43: S&P 500 Series

Order	Mse	Rmse	R2
Arima(0,2,1)	5457.0651	73.8719	0.9787
Arima(1,2,1)	5537.5709	74.4148	0.9894
Arima(1,2,2)	5567.4915	74.6156	0.9893
Arima(2,2,2)	5587.7679	74.7513	0.9782
Average	**5537.4738**	**74.4134**	**0.9839**

Table 44: Final Comparison among models

Model	Average R2	Average MSE
Employment NN Model	0.9991	28,037.82
Employment Arima Model	0.9997	8,059.50
Difference	**-0.0006**	**19978.31**
	Average R2	Average MSE
S&P 500 NN Model	0.9580	9,092.65
S&P 500 ARIMA Model	0.9839	5,537.47
Difference	**-0.0259**	**3555.17**

As it can be observed, although are very close, none of the results found to date based on neural networks, can exceed the results obtained by the ARIMA models. The average R2 score for ARIMA model in the case of the employment series is 0.9997 with an MSE of 8059.50, this generates a big difference when compared with a 0.9990 for Neural network structures with average MSE of 28037.82, been this MSE 3.48 times larger the the average ARIMA MSE. However this should be understood as an average and it is notable that there are other results that closely approximate the results of the ARIMA models. For example, the fact that for the employment series, the recurrent models like LSTM and GRU are the ones that have give the best results must be rescued. Table 44 shows how the differences are reduced when we separate only the recurrent models.

Table 45: Recurrent NN results for Employment

	R2	MSE
Recurrent Models	0.9995	15358.41
ARIMA	0.9997	8059.50
Difference	**0.0002**	**7298.90**

When we compare the models for the S&P 500 series it is possible to observe that average R2 score for ARIMA models is 0.9839 while the for neural networks we can see an average of 0.9580, we observe a difference of -0.0259. The MSE for the ARIMA model is 5537.47 while for the Neural Network models is 9092.65, that is 1.64 times the MSE of ARIMA models, this difference is much smaller, than the difference presented in the employment series. Again, although there are some models that are quite close, none of them can exceed the results obtained for ARIMA modelling. However we can still see a very good performance for the MLP and the CNN models. Table 45 show how the differences are reduced when ARIMA models are compared with the average of this two models.

Table 46: CNN-MLP Average result for S&P 500

	R2	MSE
Average CNN - MLP	0.9643	7742.29
ARIMA	0.9839	5537.47
Difference	**-0.0196**	**-2204.81**

When comparing results between financial and economic time series, as expected in both cases it can be seen that financial time series are more difficult to forecast than economic time series. This is reflected in the lower level of R2. Table 46 show the differences between financial and economic series for both ARIMA and Neural Network models, this difference is larger between the neural networks models with a 0.0411.

Table 47: Difference between Economic and Financial Series

Employment NN Model R2	S&P 500 NN Model	Difference
0.9991	0.9580	**0.0411**
Employment Arima Model R2	S&P 500 ARIMA Model	Difference
0.9997	0.9839	**0.0158**

Additionally, it is important to add that the effort made both from the computational point of view and analytical work for the neural networks is much greater in contrast to the basic ARIMA analysis. Effort that under these results would not be offset by a better forecasting capacity.

4 Conclusions

Throughout this document a tour of different neural network architectures has been made in order to evaluate the forecasting capacity that these have when applied to economic and financial time series. According to the order of the objectives set, simple but powerful ARIMA forecasting methods were developed. In which the box-jenkins methodology was applied to determine which models were more in line with the time series analyzed. Based on this results, the 4 best models were determined for each series and their prediction capacity measured through the R-squared and the mean squared error were determined, this served to form the basis of comparison on which neural networks were valued.
After this process in objective 2, an investigation of the main neural network structures was carried out, through which forecasts for time series could be made. This lead to the use of RNN, GRU, MLP and CNN structures which were developed in accordance with objective 2b.
Finally in accordance with objective 3, through the comparative results tables, the structures of the neural networks were analyzed in opposition to the results obtained under the ARIMA models and the conclusions were determined. Here we could see the differences between the average R-squared and the mean of mean squared error for each series. For the employment series, it was the recurrent models that gave the best results with an R2 of 0.9995. For the S&P 500 series it was the MLP and CNN models that showed the best results with an R2 of 0.9643.
However, none of the Neural Network results developed here could overcome the results of the ARIMA models. Being the results by the ARIMA models the ones that obtained better score.
Although neural network models are promising, under the parameters established in this study, they cannot overcome the results of the basic ARIMA models.
Additionally, at this point and throughout this document as expected in objective 3a, the differences between the economic and financial series have been compared, both from the point of view of their volatility structure and within their ability to be forecasted. This was demonstrated when analyzing the differences between the average of the R2 for the two types of series. As expected the financial series due to their more dynamic and random nature tend to be more difficult to be forecasted by both methods showing a smaller R2.

The conclusion reached in this document is that based on the amount of effort required to work a neural network, the capacity of these algorithms

for forecasting time series does not exceed the classical methods, which with less effort and less intensive use of CPU achieve results equal or superior in all cases. However, it is important to add that within the large number of hyperparameters that can be tuned there could be some combination that could approach the results obtained by ARIMA models. However, due to time and GPU limitations, it was only possible to estimate some possible combinations.
Likewise, even with the results obtained, the great capacity of neural networks to learn about the trajectory of a time series and show forecasting results that in general terms are considered quite good is evident. When we reflect on the ability of these algorithms to contribute to the economic analysis and especially to the new economic mathematics, it is considered that these are very powerful and promising instruments, and it's not discarded, the possibility that with a greater training effort or perhaps other combinations of hyperparameters the results can be as good as the traditional ARIMA models. Although the results under the applied models do not succeed in surpassing the traditional classical models, it must be clear that the world of deep learning opens up endless possibilities for new methods of quantitative analysis. Therefore, what this document aims to conclude is not that neural networks are not sufficient to compete with classic ARIMA models, but rather, that the world of deep learning opens up endless alternatives for the analysis of time series that have hardly start. In this way, the learning obtained in this work was invaluable from the point of view of exposing some of the possible alternatives offered by the world of deep learning to economic and financial modeling. The incorporation of artificial intelligence algorithms into economic analysis is a relatively new topic and to date there is little literature on it. However, its considered that the combination of deep learning methods for forecasting exogenous variables, together with reinforcement learning algorithms for the optimization of the trajectories of macroeconomic variables in economic systems, will be a powerful tool for many economists in the years to come.
What has been sought with this exercise is to contribute to the development and use of deep learning methods in economic and financial analysis through examples applied to series that in practice are very relevant and have been very difficult to forecast by economists all over the world.

It is concluded therefore that although it could be that the deep learning methods still do not surpass the classical methods. It will be a matter of time until developments such as those made in this document open up new architectures that are better adapted to solve economic and financial problems.

4.1 Proposals for future work

Among the many possibilities that remain to be tested would be to use different timesteps in the input so that other recurrent structures are formulated different than the ar(7) used in this work. Likewise, for the purposes of this document, the tuner used to do hyperparameter search was random search however, Hyperband and bayesian optimization are also available. Additionally, within the analyzed structures, you could use other combinations of layers that could give better results.
Also, relatively short series were used for this work. It is considered valuable to carry out a similar exercise with series with a greater number of observations. For example, the series of financial instruments have the property of fractality and therefore preserve their structure regardless of scale. Therefore, series as small as 1-minute on the S&P 500 can be created, this would open up the possibility to generate data as large as desired for testing and training.
For this work, comparisons based on one step ahead forecasts have been made. However, in practice many times it is possible to find the need to make forecasts for more than one period onwards. So one point that is of my personal interest, is the ability of neural networks to make out-of-sample forecasts. This is a topic that I would like to later work on personal investigations.

5 References

[1] Francois Chollet *Deep Learning with Python* 2018: Manning Publications

[2] Pedro Coelho *Building Machine Learning Systems with Python* 2018: PacktPublishing.

[3] Santanu Pattanayak *Pro Deep Learning with TensorFlow* 2017: Apress.

[4] Rodolfo Bonnin *Building Machine Learning Projects with TensorFlow* 2016: PacktPublishing

[5] Martin T.Hagan *Neural Network Design*

[6] Michael Nielsen *Neural Network and Deep Learning*

[7] MathWorks *Econometric toolbox User guide*

[8] Yves Hilpisch *Python for finance* 2015: O'Reilly Media

[9] James Ma Weiming *Mastering Python for finance* 2015: PacktPublishing

[10] Aileen Nielsen *Practical Time Series Analysis* 2015: PacktPublishing

[11] Econpy.org *Python Programming for Economist* 2017

[12] Avishek Pal *Practical Time Series Analysis* 2017: PacktPublishing

[13] Srivastava Nitish *Dropout: A simple Way to prevent Neural Networks from overfitting* 2014: Journal of Machine Learning

[14] Anton Olenink *What are neural networks not good at?* 2019: Big Data& Society

[15] Tim Hill *Neural Network Models for Time Series Forecast* 1996: Managment Science

[16] FRED *Economic Research Federal Reserve Bank of St Louis* https://fred.stlouisfed.org/

[17] George E.P Box, Gwilym Jenkins *Time series analysis forecasting and control* 2015: Wiley

[18] James D Hamilton *Time Series Analysis* 1994: Princeton University Press

[19] NCSS *the Box-Jenkins Method* 2020: NCSS statistical software

[20] G.M. Ljung, G.E.P. Box *On a measure of lack of fit in time series models* 1978: Biometrika

[21] Ian Dewancker *A Stratified Analysis of Bayesian Optimization Methods* 2016: SIGOPT.com

[22] Lisha Li *Hyperband: A novel Bandit-Based Approach to Hyperparameter Optimization* 2018: Journal of Machine Learning

[23] David A Dickey, Wayne A. Fuller *Distribution of the Estimators for Autoregressive Time Series with a Unit Root* 1976: Journal of American Statistical Association

[24] Keras Tuner *Keras Tuner Documentation* 2020: Github -https://keras-team.github.io/keras-tuner/

[25] Scikit learn *Cross-validation: evaluating estimator performance* 2020: https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

[26] Scikit learn *sklearn.preprocessing.MinMaxScaler* 2020: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[27] Scikit learn *Recurrent Neural Networks* 2019: https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce

[28] MissingLink *Deep Learning Long Short-Term Memory (LSTM) Networks: What You Should Remember* 2016: https://missinglink.ai/guides/neural-network-concepts/deep-learning-long-short-term-memory-lstm-networks-remember/

[29] Sciaga Progamisty *DRNN (Recurrent Neural Networks and LSTM (Long short-term memory)* 2018: http://sciagaprogramisty.blogspot.com/2018/04/rnn-recurrent-neural-networks-and-lstm.html

[30] Sumit Saha *Convolutional Neural Networks -the ELI5 way* 2018: towards data science https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[31] Deeplizard *Zero Padding in Convolutional Neural Networks Explained* 2020: https://deeplizard.com/learn/video/qSTv_m-KFk0