Deep Neural Network Based Time Series Models

11

11.1 INTRODUCTION

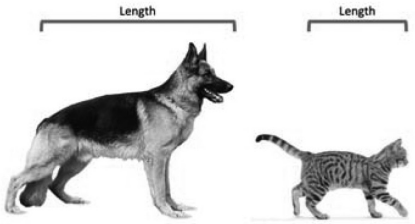
Deep Learning and Artificial Intelligence (AI) have taken the world by storm and have found applications including self-driving cars, virtual assistants such as Siri and Alexa, fraud detection, and image recognition. As we will see, this exciting technology holds tremendous potential for time series analysis as well.

First conceived in 1958 (Rosenblatt, 1958), the neural network is the most fundamental construct used in building "Deep Learning" applications. In this chapter, we will introduce a basic neural network known as the *perceptron* and see how it can be extended to facilitate the analysis of both univariate and multivariate time series data. In the process, we will make use of functions in the **`nnfor`** package to construct and forecast from, potentially "deep", perceptron-based neural networks.

11.2 THE PERCEPTRON

The first neural networks focused on using initial information to choose between two possible outcomes, aka, binary classification. To illustrate, we will first present a simple non-time-series example in which the goal is to classify a particular animal as a dog or a cat, given only the length of each animal from the base of its tail to the tip of its nose. Intuitively, we will first take a sample of n animals and record this metric for each animal and whether it is a dog (coded as a 1) or a cat (coded as a 0). The resulting data, in which we know the length of each animal and its classification, are known as the *training set*.

A neural network approach to this problem is as follows:



	length	animal	coded
1	2.5	dog	1
2	3.4	dog	1
3	1.2	cat	0
4	1.7	cat	0
5	3.9	dog	1
6	2.1	cat	0

For each observation in the training set:

DOI: 10.1201/9781003089070-11

1. The lengths from each animal are normalized by a form of the "minimax" transformation:

$$length_{t_{minimax}} = \frac{length_t - length_{min}}{length_{max} - length_{min}}$$

where $length_t$ is the length of the t[th] animal in the sample, and $length_{min}$ and $length_{max}$ are the smallest and largest lengths of all the animals in the sample (dog or cat). The reasons for this transformation will be described later in the chapter.

2. The normalized length is multiplied by a number w (called a model weight) and then added to a constant (aka: "bias term"),θ, which results in the sum:

$$f = \theta + w*Length$$

3. f is then used as the input to the sigmoid (or "logistic") function (Figure 11.1):

$$g = \frac{1}{1+e^{-f}}$$

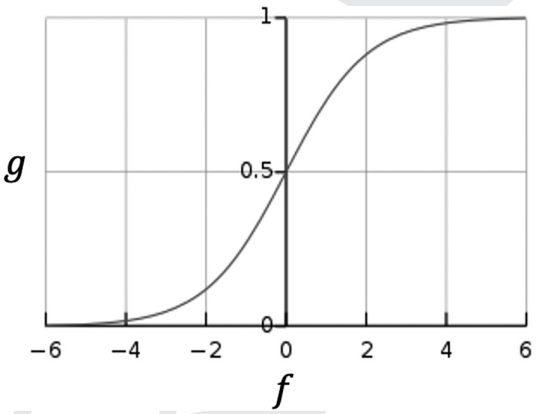


FIGURE 11.1 The logistic (sigmoid) function. This maps a real-valued function f to a value g between 0 and 1 exclusive

This is known as the "activation function", for which we will have many choices. For binary classification, the sigmoid function is a logical choice as it forces an output between zero and one which corresponds to the probability of a "success" (the outcome coded as a one), in this case, a dog.

4. The final step is then to classify the observation as a dog if g is greater than some threshold, commonly .5, and a cat if it is less than or equal to that threshold.
5. An iterative numerical procedure, known as *gradient descent*, is then used to approximate the "best" w and θ that maximize the number of times the model is correct, its measure of accuracy. This can take quite a bit of time and computational power depending on the number of observations and complexity of the model. For this reason, a proxy measure of accuracy that

has been shown to be very efficient is called the "cross-entropy". The cross-entropy measures the disorder that is produced by the model, which will be low if the accuracy is high. Therefore, we call the "cross-entropy" a "loss" function and aim to find the combination of w and θ that will minimize:

$$Cross - Entropy = \sum_{i=1}^{n} - p(x_i)\log\left(q(x_i)\right)$$

For each observation, i, $p(x_i)$ will either be one if x_i is a dog or zero if x_i is a cat, while $q(x_i)$ is the probability that is output from the activation (sigmoid) function. One reason for our normalization step above is because gradient descent has been shown to be more stable and efficient when the input data is normalized.

The algorithm that includes these five steps combined with the gradient descent procedure to minimize the loss function is known as the *perceptron.* The perceptron for the classification problem is illustrated in Figure 11.2 below, and is broken into an input layer with a single input node (Length) and an output layer with a single output node (Dog/Cat).[1] Going forward, we will call these diagrams the *architecture* of the neural network.

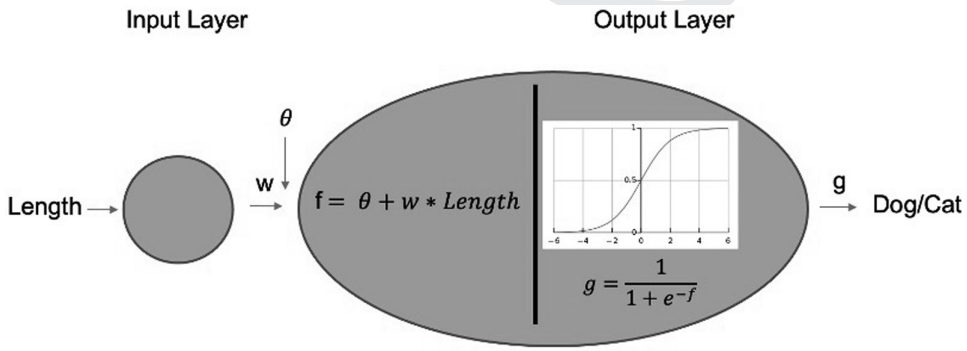


FIGURE 11.2 Architecture of the dog/cat classification perceptron

11.3 THE EXTENDED PERCEPTRON FOR UNIVARIATE TIME SERIES DATA

In this section we will explore how the fundamental MLP framework described above can be extended to model in the time series setting. The MLP analog to the AR(p) class of models will be described first at which point "deeper" neural networks will be investigated. Finally, this section will conclude with an example in which these concepts are applied to the **`AirPassengers`** data.

1 If the loss function for the gradient descent is cross-entropy, then this neural network is equivalent to logistic regression.

11.3.1 A Neural Network Similar to the AR(1)

11.3.1.1 The Architecture

As previously mentioned, the perceptron can be adapted to the time series setting by the choice of the *activation function*. For example, suppose we want to model the West Texas Intermediate crude oil prices and we determine that all useful information about this month's price (X_t) is contained in last month's price (X_{t-1}). Note that in contrast to the neural network described previously, X_t, the price, in this particular case is now a continuous variable rather than binary. For this reason, as seen in Figure 11.3, we can repurpose the perceptron to model time series data by changing the activation function from the sigmoid function to one that will allow for an unbounded response. A commonly used activation function in this setting is the identity function whose output is exactly its input: $y = x$.

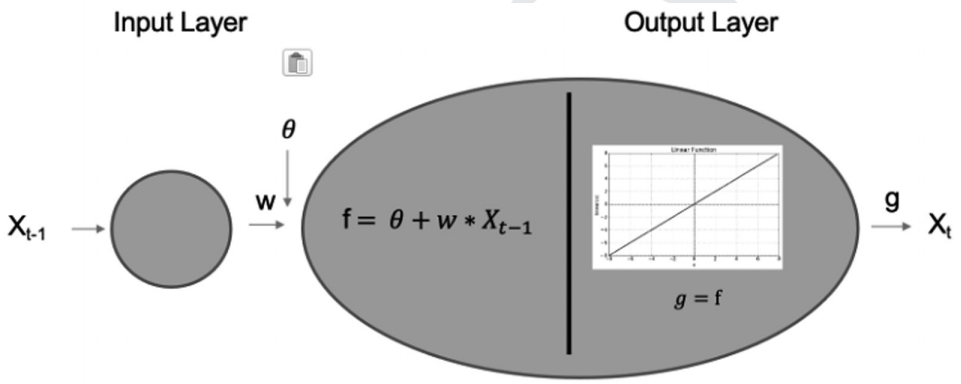


FIGURE 11.3 Architecture of a network similar to the AR(1.

Note that, using the architecture in Figure 11.2, the neural network can be written as

$$X_t = \theta + wX_{t-1} + a_t,$$

which is similar to the AR(1) model,

$$X_t = \mu + \phi_1 X_{t-1} + a_t$$

An additional difference imposed by changing to a continuous response is that of the loss function. We will replace the cross-entropy discussed earlier for classification-based neural networks with the familiar sum of squared error (SSE).[2] The gradient descent algorithm will thus aim to return a set of weights and biases to minimize the SSE.

11.3.1.2 Fitting the MLP

To fit this model in R, we will use the **`mlp`** function in the **`nnfor`** package. The acronym "MLP" stands for multi-layered perceptron, which pertains to potential multiple *hidden layers* that we will discuss later in this chapter. For the model above, we will have only the input and output layers (no hidden

2 Sum of squared residuals

layers: **hd = 0** in the **mlp** function). In this example, we use only one lag (**lags = 1**) and note that the model does not require any pre-differencing (**difforder = 0**) and thus the model can be fit in R using

```
AR1Fit = mlp(wtcrude2020, m = 1, hd = 0, lags = 1, difforder = 0, reps = 5)
```

Figure 11.4 displays a simplified diagram (make that "very simplified"!) of the single input (left) and single output node (right) network illustrated in Figure 11.3. This diagram will be more useful (and involved!) as the architecture becomes more complex. Obtaining this plot is accomplished with the R code

```
plot(AR1Fit)
```

FIGURE 11.4 Visualization of the architecture of the AR(1) type neural network from package **nnfor**

Additionally, summary information (below in 11.1) about the fit model can be obtained with the code

```
AR1Fit
```
[3]

```
MLP fit with 0 hidden node and 5 repetitions.
Univariate lags: (1)
Forecast combined using the median operator.
MSE: 22.2019.
```
(11.1)

The first and second lines of the output in (11.1) summarize the lack of hidden layers (`0 hidden`) and the inclusion of the X_{t-1} term (`Univariate lags: (1)`), and also indicate that five repetitions (`5 repetitions`) were selected to most effectively fit the model. These details are discussed next.

The gradient descent procedure used in estimating the optimal values of w and θ depend on initial weights,[4] or starting values, of w and θ. Depending on the complexity of the neural network, estimates of w may vary considerably based on these starting values. For this reason, we have chosen to fit several neural networks, in this case **reps = 5**, which will be initiated with five different sets of starting values and thus will yield potentially five different estimates of w. (Note that only five repetitions were generated here for illustrative simplicity. In theory, more repetitions will yield better estimates although the increase can also require significantly more time to fit the model; the default is **reps = 20**).[5]

For the West Texas Intermediate Crude example, the five different estimates of θ (the top number in the output) and w (the bottom number in the output) are shown below. They can be accessed using the R command:

```
AR1Fit$net$weights
```

```
             [,1]
[1,] -0.00395087
[2,]  0.98539259
```

3 Note that all output in this chapter was generated with a seed of 1. To reproduce the output and plots in this chapter, **set.seed(1)** will need to be run immediately before any call to the **mlp** function. If the code is run with a different (random) seed, similar results should be expected. Doing so is often a good exercise.

4 https://stats.stackexchange.com/questions/47590/what-are-good-initial-weights-in-a-neural-network

5 In the simple example case presented here, the optimization problem is "convex", meaning that it only has one (global) minimum. For this reason, the gradient descent algorithm will converge to this single value regardless of the starting value. More complex neural networks that include hidden layers will be very non-convex and thus very dependent on the starting values of w and θ.

```
             [,1]
[1,] -0.003922568
[2,]  0.985443097

             [,1]
[1,] -0.003919923
[2,]  0.985462926

             [,1]
[1,] -0.003844282
[2,]  0.985712695

             [,1]
[1,] -0.003926506
[2,]  0.985443707
```

QR 11.1 MLP for the AR(1)

Note also that these values of w are very close to the estimate of $\hat{\phi}_1$ in the AR(1) fit using the Burg estimates ($\hat{\phi}_1$= .987) obtained using the code below.

```
estAR1 = est.ar.wge(wtcrude2020,p = 1, type = "burg")
estAR1$phi
```

```
[1] 0.9865561
```

Interestingly, this is the neural network representation of an AR(1)!

11.3.1.3 Forecasting

To calculate the forecasts, the **mlp** function uses the five different weights and θs to generate five separate forecasts of West Texas Intermediate Crude price for the next month (month 373: X_{373}). The forecast for this month (horizon **h = 1**) from each of the five model fits can be calculated and accessed with:

```
preds = forecast(AR1Fit,h = 1)
preds$all.mean
```

```
                NN.1       NN.2       NN.3       NN.4       NN.5
Jan 2021    47.16398   47.16486   47.16456   47.16398   47.16455
```

The columns represent the five neural networks that were fit, and the single row contains the respective forecasts for X_{373} (**wtcrude2020**consists of 372 months of data) from each of the five neural networks. Returning to the third line of the output in (11.1), we note that the final forecast is the median of these five individual forecasts (even though the attribute is always called the "mean"): `Forecast combined using the median operator.` This final forecast can be accessed using the following code and output. Note that, internal to the ***nnfor*** package, the attribute is called the "**$mean**" even though for this MLP, the median has been used to combine the forecasts. The value below is indeed the median of the five forecasts.

```
preds$mean # even though this is the median
```

```
          Jan
2021 47.16455
```

Extending this idea to longer horizons, below we see forecasts for a horizon of ten. Again, there is a column for each neural network (for each "repetition") and a row for each of the ten forecasts **(h = 10)**

```
preds = forecast(AR1Fit,h = 10)
preds$all.mean
```

```
                NN.1       NN.2       NN.3       NN.4       NN.5
Jan 2021    47.16398   47.16486   47.16456   47.16398   47.16455
Feb 2021    47.23303   47.23478   47.23418   47.23305   47.23415
Mar 2021    47.30107   47.30368   47.30279   47.30113   47.30274
Apr 2021    47.36812   47.37158   47.37040   47.36824   47.37034
May 2021    47.43418   47.43849   47.43703   47.43439   47.43695
Jun 2021    47.49928   47.50442   47.50269   47.49960   47.50259
Jul 2021    47.56343   47.56940   47.56740   47.56387   47.56727
Aug 2021    47.62665   47.63343   47.63116   47.62722   47.63102
Sep 2021    47.68894   47.69653   47.69400   47.68967   47.69383
Oct 2021    47.75032   47.75871   47.75592   47.75123   47.75573
```

As before, the median of each row becomes the ultimate forecast for each month (check it!):[6]

```
preds$mean #even though this is still the median
```

```
         Jan      Feb      Mar      Apr      May      Jun      Jul      Aug      Sep      Oct
2021 47.16455 47.23415 47.30274 47.37034 47.43695 47.50259 47.56727 47.63102 47.69383 47.75573
```

Note that while the AR(1) and neural network are similar, a difference between the two models is that the forecasts from the neural network are not attracted to the overall sample mean, $\bar{X}_{wtcrude2020} = 47.47493$, as is the case for the stationary AR(1) model.

Recall that in the AR(1) case, the forecasts are calculated with the eventual forecast function from (6.28):

$$\hat{X}_{t_0}(\ell) = \hat{\phi}_1 \hat{X}_{t_0}(\ell - 1) + \bar{x}(1 - \hat{\phi}_1)$$

On the other hand, forecasts from the neural network model are calculated as:

$$\hat{X}_{t_0}(\ell) = \hat{w}_1 \hat{X}_{t_0}(\ell - 1) + \hat{\theta}_1$$

where in most cases, $\bar{x}(1 - \hat{\phi}_1) \neq \hat{\theta}_1$.

This difference is highlighted in Figures 11.5(a)−(c), where the forecasts from both the AR(1) and MLP models are provided (with $h = 300$ to aid in comparison) as well as their respective eventual forecast functions. Figure 1.5(a) displays the forecasts from the end of the **wtcrude2020** series, where the divergence is clear. Given $\hat{\phi}_1 = .987$ and $\bar{x}(1 - \hat{\phi}_1) = 47.47493(1 - .987) = -.004$ for the AR(1), and $\hat{w}_1 \cong .986$ and $\hat{\theta}_1 \cong -.004$ for the MLP, the corresponding eventual forecast functions are:

$$AR(1): \ \hat{X}_{t_{372}}(\ell) = .987\hat{X}_{t_0}(\ell - 1) + .617$$
$$MLP: \quad \hat{X}_{t_{372}}(\ell) = .986\hat{X}_{t_0}(\ell - 1) - .004$$

The difference in the forecasts in Figure 11.5(a) is the result of the discrepancy between .617 and –.004 in the above eventual forecast function.

6 It turns out that the last column (NN5) is always the median in this simple example; this does not have to be the case.

To further highlight this phenomenon, a new forecast origin was chosen to be April 2014 (observation 292). Using this forecast origin, $\hat{\phi}_1 = .992$ and $\bar{x}(1-\hat{\phi}_1) = 45.52196(1-.992) = .364$ for the AR(1) and for the MLP, $\hat{w}_1 \cong .994$ and $\hat{\theta}_1 \cong .002$. These estimates yield the eventual forecast functions below with corresponding plots in Figure 11.5(b).

$$AR(1): \quad \hat{X}_{t_{292}}(\ell) = .992\hat{X}_{t_0}(\ell-1) + .364$$
$$MLP: \quad \hat{X}_{t_{292}}(\ell) = .994\hat{X}_{t_0}(\ell-1) + .002$$

Figure 11.5(c) displays a comparison of the forecasts from a forecast origin of April 2020 (observation 364) in which again, the eventual forecast functions are different, although this time, there is very little difference that can be seen visually.

$$AR(1): \quad \hat{X}_{t_{364}}(\ell) = .987\hat{X}_{t_0}(\ell-1) + .619$$
$$MLP: \quad \hat{X}_{t_{364}}(\ell) = .988\hat{X}_{t_0}(\ell-1) - .004$$

There is an interesting pattern to these forecasts depending on whether the MLP model's forecast origin is below the sample mean, near the sample mean, or greater than the sample mean.

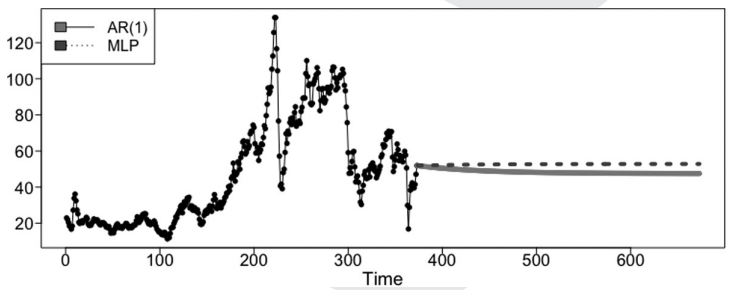


FIGURE 11.5(a) Plot of AR(1) forecasts from the end of the series (January 2021) with a horizon of 300 (solid) versus forecasts with a horizon of 300 from the MLP model with one lag described above (dashed)

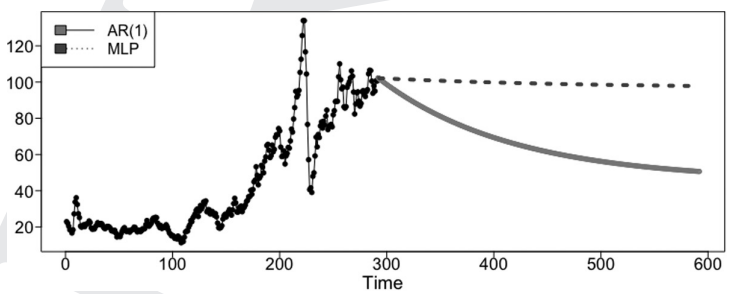


FIGURE 11.5(b) Plot of AR(1) forecasts from the end of the series (April 2014) with a horizon of 300 (solid) versus forecasts with a horizon of 300 from the MLP model with one lag described above (dashed)

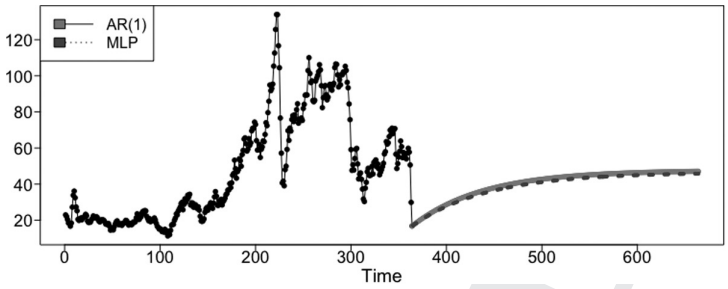


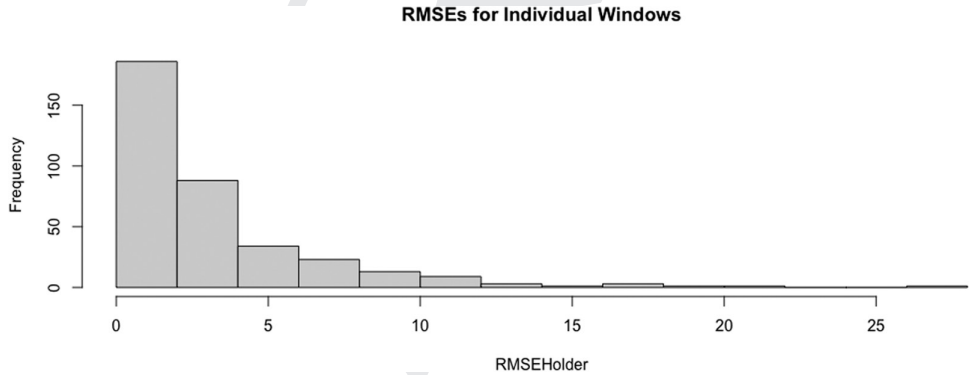
FIGURE 11.5(c) Visualization of a case in which the AR(1) (solid) and MLP (dashed) forecasts are very similar, although closer inspection reveals they are still slightly different

11.3.1.4 Cross Validation Using the Rolling Window RMSE

As in previous chapters, we can produce a rolling window RMSE from this model to assess its fit for a given horizon. In order to do this, we will take the ws and θs from the model we fit with the data and pass these estimates along with the desired horizon to the ***tswge*** function **`roll.win.rmse.nn.wge()`**. The code and output for the rolling window RMSE of the model contained in **`AR1Fit`** with a horizon of one are shown below:

```
rwAR1Fit = roll.win.rmse.nn.wge(wtcrude2020,horizon = 1,fit_model = AR1Fit)
"Please Hold For a Moment, TSWGE is processing the Rolling Window RMSE with 363
windows."
"The Summary Statistics for the Rolling Window RMSE Are:"
   Min.   1st Qu.    Median      Mean    3rd Qu.       Max.
0.01466   0.87807   1.88629   3.13809    3.91681   27.08154
"The Rolling Window RMSE is: 3.138"
```

Similar to the rolling window RMSE functions in previous chapters, this function can use the AR1Fit model to make a forecast for nearly every value in the dataset and then calculate the RMSE since the actual values are known.[7] In this case, since the horizon is one, the function calculates 363 (from the first line of the output above) one-step-ahead forecasts and then takes the average to produce the "Rolling Window RMSE", which in this case, is 3.138. The 5-number summary and a histogram (Figure 11.6) are also available in order to describe the distribution of the RMSEs from these 363 windows.



QR 11.2 Rolling Window for MLP

FIGURE 11.6 Histogram showing the distribution of the 363 RMSEs calculated from the 363 windows constructed from the **`wtcrude2020`** series

7 The **`mlp`** function uses the first eight data values and uses a simple exponential smoother to detect trend and thus decides if a first difference is necessary. For this reason, forecasts for the first eight values are not possible.

Intuitively, it is reasonable to expect the forecasting error to increase as we predict further into the future (as the horizon increases). This intuition is supported by looking at the rolling window RMSE for the horizon of ten discussed earlier:

```
rwAR1Fit = roll.win.rmse.nn.wge(wtcrude2020,horizon = 10,fit_model = AR1Fit)

"Please Hold For a Moment, TSWGE is processing the Rolling Window RMSE with 354
windows."
"The Summary Statistics for the Rolling Window RMSE Are:"
 Min.      1st Qu.    Median      Mean     3rd Qu.      Max.
1.139       3.974     6.524     9.514     11.222     68.515
"The Rolling Window RMSE is: 9.514"
```

Note that given the larger horizon, we can only compute RMSEs for 354 windows and that the corresponding rolling window RMSE has increased from 3.138 to 9.514 dollars.

11.3.2 A Neural Network Similar to AR(*p*): Adding More Lags

More generally, the perceptron-based neural network can be extended to accommodate up to p lags by adding more input nodes as illustrated in Figure 11.7.

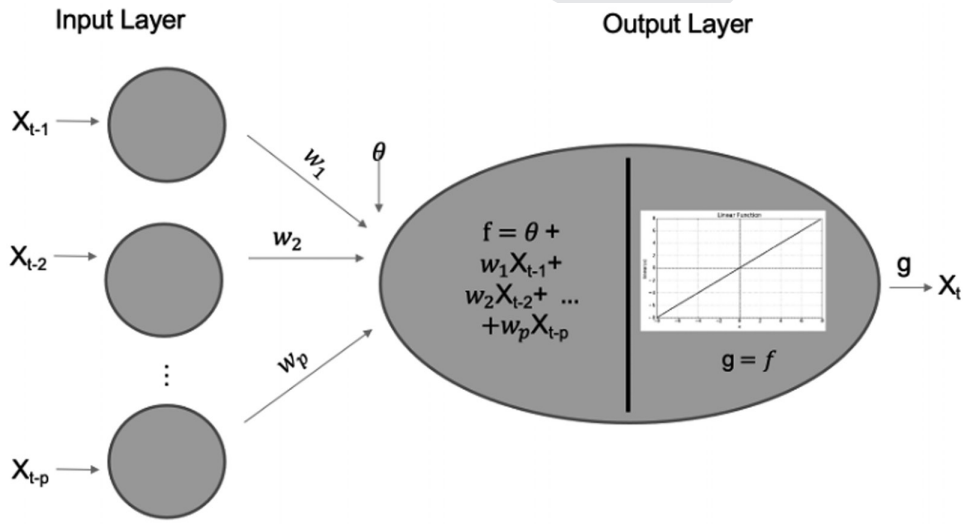


FIGURE 11.7 A Neural Network with input nodes for p lags

To continue the univariate analysis of the West Texas Intermediate Crude oil data, Figure 11.8 shows the architecture for a neural network with inputs for three lags. It will be shown that this is similar, but not equivalent, to the AR(3) fit to these data in Chapter 3.

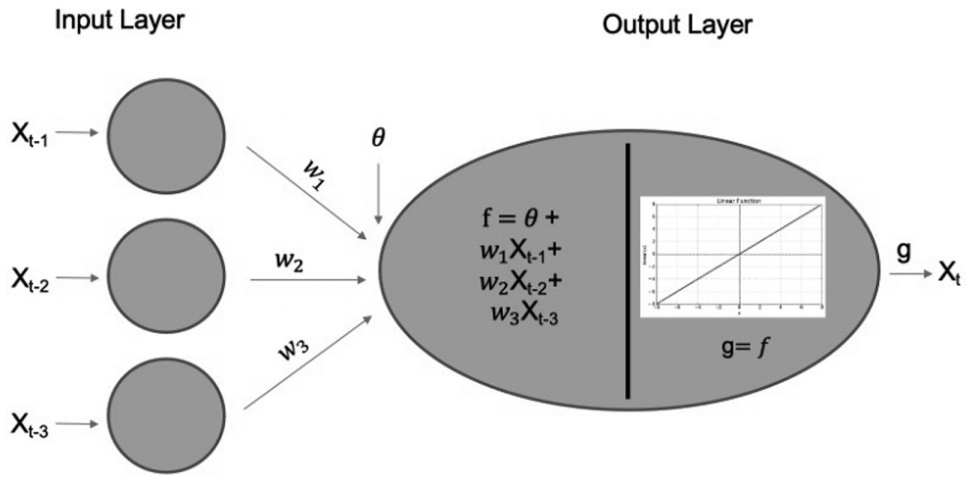


FIGURE 11.8 A Neural Network with input nodes for three lags. This is analogous to an AR(3) model

The **mlp** function will allow for the inclusion of specific lags in the model. In order to include X_{t-1}, X_{t-2} and X_{t-3} in the model, we must specify **lags = c(1,2,3)**. However, by default, the **mlp** function will automatically select from these lags only those it deems useful (using cross-validation). To override this, the analyst must set the **sel.lag** argument to FALSE (**sel.lag = FALSE)**. Below, see the resulting fit and architecture diagram of this AR(3)-like neural network.

```
AR3Fit = mlp(ts(wtcrude2020),hd = 0, lags = c(1,2,3), difforder = 0, reps = 5,
sel.lag = FALSE)

AR3Fit

MLP fit with 0 hidden node and 5 repetitions.
Univariate lags: (1,2,3)
Forecast combined using the median operator.
MSE: 18.8756.

plot(AR3Fit)
```

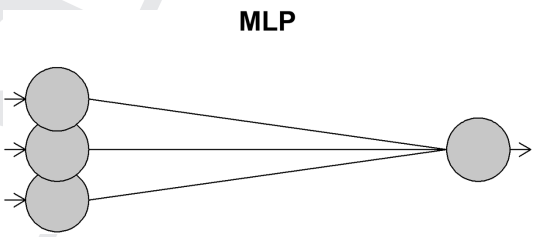


FIGURE 11.9 Architecture of the MLP equivalent of the AR3

As an indication of better fit, note that the MSE for one-step-ahead forecasts on the training data has reduced from 22.2 in the model including only X_{t-1}, to 18.9 when adding X_{t-2} and X_{t-3}. While the one-step forecasts may be overly optimistic and should be used with caution, the apples-to-apples comparison here is useful.

As a quick aside, observe what occurs if **mlp** is used to automatically select the lags from X_{t-1}, X_{t-2} and X_{t-3}.

```
AutoFit = mlp(ts(wtcrude2020),hd = 0, lags = c(1,2,3), difforder = 0, reps = 5,
sel.lag = TRUE)

AutoFit

MLP fit with 0 hidden node and 5 repetitions.
Univariate lags: (1,2)
Forecast combined using the median operator.
MSE: 18.8279.

plot(AutoFit)
```

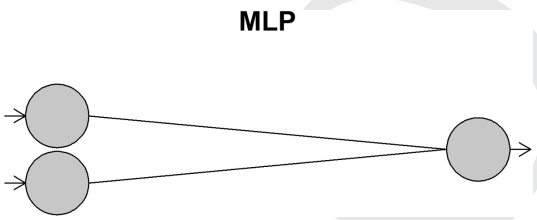


FIGURE 11.10 Architecture of the MLP equivalent of the AR3

Internally, the **mlp** function fit a model with all combinations of the three lags (c(1), c(1,2), c(1,3), c(2,3) and c(1,2,3)) and returned the model that included only X_{t-1} and X_{t-2}, which results in a one-step-ahead MSE of 18.8279. Figure 11.10 above reflects this architecture.

Forecasting from these models is performed as before and forecasts are provided below for reference:

```
preds = forecast(AR3Fit,h = 10)
preds$mean

Time Series:
Start = 373
End = 382
Frequency = 1
     t+1      t+2      t+3      t+4      t+5      t+6      t+7      t+8      t+9     t+10
49.28015 50.24284 50.63960 50.79640 50.84923 50.85729 50.84636 50.82766 50.80604 50.78361

preds = forecast(AutoFit,h = 10)
preds$mean

Time Series:
Start = 373
End = 382
Frequency = 1
     t+1      t+2      t+3      t+4      t+5      t+6      t+7      t+8      t+9     t+10
49.30093 50.16587 50.49673 50.61529 50.64973 50.65107 50.63677 50.61640 50.59460 50.57501
```

Cross-validation for the horizon of ten is analogous (below) as well, and demonstrates that the RMSEs are close but are slightly smaller for the lag 2 (**Autofit**) neural network model (Lag 3 Fit: 10.172 versus Lag 2 Fit: 10.099):

```
rwAR3Fit = roll.win.rmse.nn.wge(wtcrude2020,horizon = 10, fit_model = AR3Fit)

"Please Hold For a Moment, TSWGE is processing the Rolling Window RMSE with 354
windows."
"The Summary Statistics for the Rolling Window RMSE Are:"

 Min.       1st Qu.   Median     Mean    3rd Qu.    Max.
1.220        5.021    7.380    10.172    12.080   62.094
"The Rolling Window RMSE is: 10.172"
```

```
rwAutoFit = roll.win.rmse.nn.wge(wtcrude2020,horizon = 10,fit_model= AutoFit)

"Please Hold For a Moment, TSWGE is processing the Rolling Window RMSE with 354
windows."
"The Summary Statistics for the Rolling Window RMSE Are:"

 Min.       1st Qu.   Median     Mean    3rd Qu.    Max.
1.305        4.984    7.243    10.099    12.021   62.298
"The Rolling Window RMSE is: 10.099"
```

11.3.3 A *Deeper* Neural Network: Adding a Hidden Layer

We are now in a position to highlight one of the principal differences between neural network models and the linear models we have studied thus far. Remember that "MLP" stands for "multi-layered perceptron" and "multi-layered" refers to the inclusion of a "hidden layer" (that has previously been excluded). This "hidden layer" is inserted between the input and output layer and may be composed of one or more "hidden nodes". While we have technically investigated a few MLP architectures to this point, many in the data science field feel that the term "neural network" implies one or more hidden layers.[8,9]

For simplicity, consider the previous neural network model with X_{t-1}, X_{t-2}, and X_{t-3} described earlier except this time a single hidden layer is included, which is composed of two hidden nodes. This architecture is illustrated in Figure 11.11.

8 Research has shown that analyzing time series with neural networks usually does not benefit beyond the addition of a single hidden layer. See https://kourentzes.com/forecasting/2019/01/16/tutorial-for-the-nnfor-r-package/

9 The above MLPs were included to introduce the vocabulary, architecture, and functionality of neural networks using a familiar model: the AR(p). As we have seen above, these MLP architectures produce different forecasts and thus are not AR(p) models. If an AR(p) was thought to be the most useful model, which is often the case, they should be implemented using methods such as those found in ***tswge;*** forecasting will potentially be much faster and will yield the appropriate forecasts.

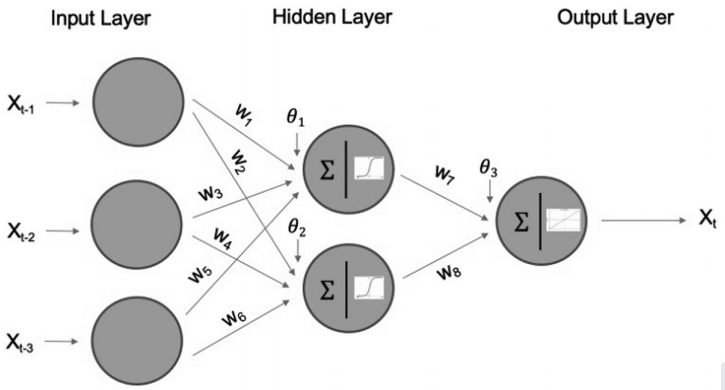


FIGURE 11.11 Architecture with three input nodes in the input layer, two hidden nodes in the hidden layer with logistic activation functions, and a single output node in the output layer with the identity activation function

Although the architecture of this MLP is relatively simple, there is a lot going on here! Note first that each input node is "connected" via a weight to every hidden node and the Σ in each hidden node indicates that the inputs, weights, and constants are combined in the same way as described before. For example, the combination of these inputs and parameters facilitated by the first (top) hidden node is displayed in Figure 11.12a. The result is obtained similarly for the second hidden node (Figure 11.12b).

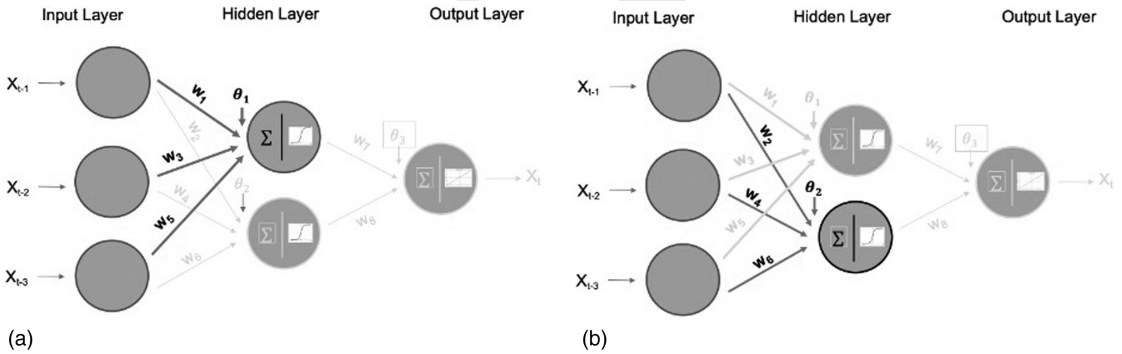


FIGURE 11.12 Illustration of the calculations involved when adding a hidden layer. The functional form for (a) f_1 and (b) f_2 are included below each plot

The resulting sum for each hidden node is then acted upon by its corresponding activation function. Figure 11.13 is an enhanced view of the first hidden node in Figure 1.12(a) and reveals that the **`mlp`** function uses the logistic function for the hidden nodes, in contrast to the identity function used for the output node, which allows for the modeling of nonlinear relationships between X_{t-1}, X_{t-2} and X_{t-3}.

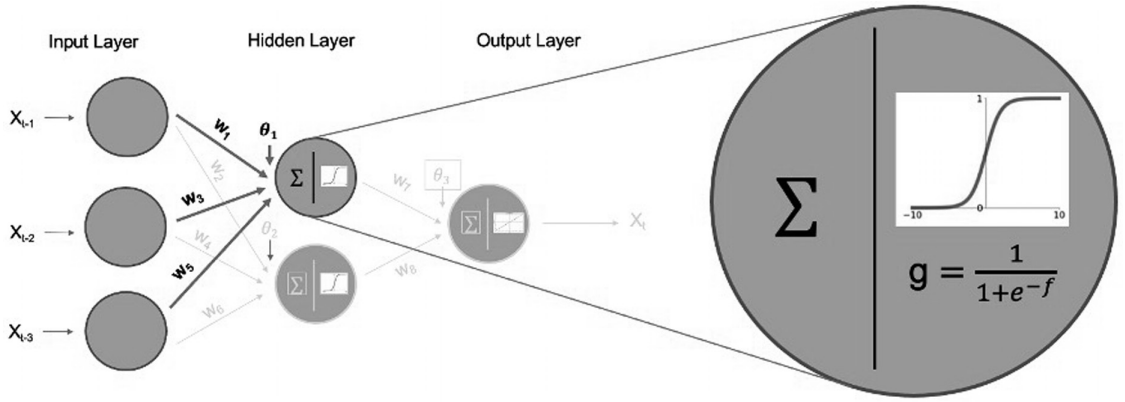


FIGURE 11.13 Close-up of the first hidden node in the single hidden layer highlights the logistic (sigmoid) activation function that maps a real number to a number between 0 and 1

The corresponding function including both hidden nodes is:

$$w_7 g(f_1) + w_8 g(f_2) + \theta_3 = X_t,$$

where the w_is and θ_js are again estimated using gradient descent.

The added complexity from the additional weights and the non-linear activation functions degrade the interpretability of these types of models and has even led to these types of models being labeled as "black box"; however, in some settings, they provide more accurate forecasts.

For example, returning to the analysis of the West Texas Crude Oil data, the following R output and Figure 11.14 reveal that the inclusion of a single hidden layer with two nodes (**`hd = 2`**) has reduced the MSE from 18.8279 to 17.6459.

```
DeepFit = mlp(ts(wtcrude2020),hd = 2, lags = c(1,2,3), difforder = 0, reps = 5,
sel.lag = FALSE)

DeepFit

MLP fit with 2 hidden nodes and 5 repetitions.
Univariate lags: (1,2,3)
Forecast combined using the median operator.
MSE: 17.6459.

plot(DeepFit)
```

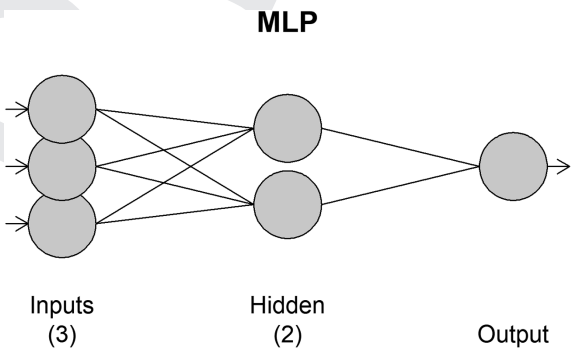


FIGURE 11.14 Architecture of the "DeepFit" MLP. There are three input nodes corresponding to the three lags, two hidden nodes in the hidden layer and an output node

However, it is very important to remember that since the loss function of this MLP is the SSE, which is in the numerator of the MSE itself, *the MSE will always decrease with the inclusion of hidden layers*. A more useful measure of model improvement will be the rolling window RMSE with a horizon larger than 1. Calculating forecasts and the rolling window RMSE is the same as it was with the simpler models discussed above.

```
preds = forecast(DeepFit,h = 10)
round(preds$mean, 2)
```

```
Time Series:
Start = 373
End = 382
Frequency = 1

   t+1    t+2    t+3    t+4    t+5    t+6    t+7    t+8    t+9   t+10
 49.19  50.12  50.58  50.87  51.12  51.35  51.59  51.83  52.07  52.30
```

```
rwDeepFit = roll.win.rmse.nn.wge(wtcrude2020,horizon = 10,fit_model= DeepFit)
```

```
"Please Hold For a Moment, TSWGE is processing the Rolling Window RMSE with 354
windows."
"The Summary Statistics for the Rolling Window RMSE Are:"

  Min.     1st Qu.    Median      Mean   3rd Qu.      Max.
0.6294      3.8252    5.9459    9.0319   12.5727   31.2432

"The Rolling Window RMSE is: 9.032"
```

We note that the MLP with a single hidden layer with two hidden nodes (DeepFit) achieves a lower rolling window RMSE for a horizon of 10, as seen in Table 11.1, than the MLPs with the AR-type architectures fit above.

TABLE 11.1 Comparison of the 4 MLP architectures fit to the wtcrude2020 data.

MODEL	*rwRMSE*
AR1Fit	9.514
AR3Fit	10.172
AutoFit	10.099
DeepFit	9.032

This indicates that there is evidence to suggest that there may be non-linear relationships in the data and that the hidden layers may be useful in modeling those relationships.

11.3.3.1 Differences and Seasonal "Dummies"

In neural network terminology, the number of lags, hidden layers, hidden nodes per hidden layer, choice of activation function, etc. are called *hyperparameters*. Incidentally, two additional hyperparameters previously studied, differencing and adding dummy (indicator) variables to model seasonal effects, are also useful in MLP-based time series models. We will make use of the **`AirPassenger`** data to illustrate how to set these additional hyperparameters.

Key Points:

hyperparameter – "A hyperparameter is a parameter that is set before the learning process begins. These parameters are tunable and can directly affect how well a model trains."[10]

Examples of hyperparameters include number of hidden layers, number of hidden nodes, number of differences, presence of seasonal indicator variables, choice of activation function, number of lags, and many more (both a blessing and a curse!)

Recall the monthly airline passenger data in the AirPassenger dataset displayed below for convenience in Figure 11.15 a-c.

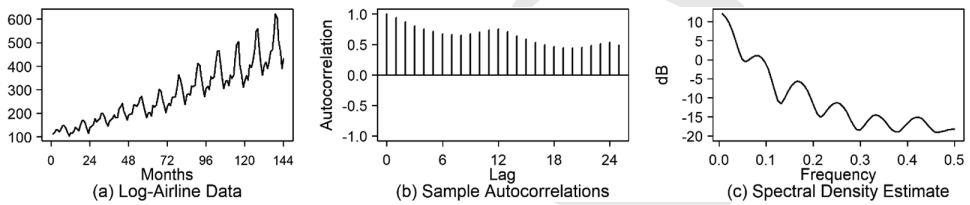


FIGURE 11.15 (a) The number of airline passengers from 1949 to 1960, (b) the sample autocorrelations of the airline passenger data, (c) the spectral density of the airline passenger data

Domain knowledge of air traffic patterns and visual inspection of the plots in Figure 11.15 provide overwhelming evidence of these data being generated from a non-stationary process. The data clearly exhibit strong evidence of positive trend, which is reflected in the slowly damping sample autocorrelations and the peak at zero in the spectral density. Additionally, the realization, sample autocorrelations, and sample spectral density paired with our experience provide equally strong evidence of seasonal behavior with a period of 12. Specifically, sample autocorrelations reveal a sinusoidal pattern with a peak at lag 12, and the spectral density shows a peak at .083 (1/12) with four other peaks that are consistent with a $(1\text{-}B^{12})$ factor. These characteristics support the conclusion that the data exhibit non-stationary seasonal behavior. Previously, in Chapter 7, we modeled these data with a model containing a (1-B) and $(1\text{-}B^{12})$. We called this an "airline model".

We also analyze these data with an MLP by specifying **`difforder = c(1,12)`**. The following code fits this model by allowing for five hidden nodes (the default) and the automatic selection of the number of lags (also the default). Note that the MLP model will also fit seasonal dummy/indicator variables by default. We will discuss this option later; for now, so we can focus on the differencing, we will turn off the seasonal indicator variable option (**`allow.det.season = FALSE`**).

```
fit.1.12.H5 = mlp(AirPassengers,difforder = c(1,12),allow.det.season = FALSE)
fit.1.12.H5

MLP fit with 5 hidden nodes and 20 repetitions.
Series modeled in differences: D1D12.
Univariate lags: (1,3,4,7,9)
Forecast combined using the median operator.
```

10 https://deepai.org/machine-learning-glossary-and-terms/hyperparameter

```
MSE: 53.5823.
```

The code below and Figure 11.16 displays the architecture of this model:

```
plot(fit.1.12.H5)
```

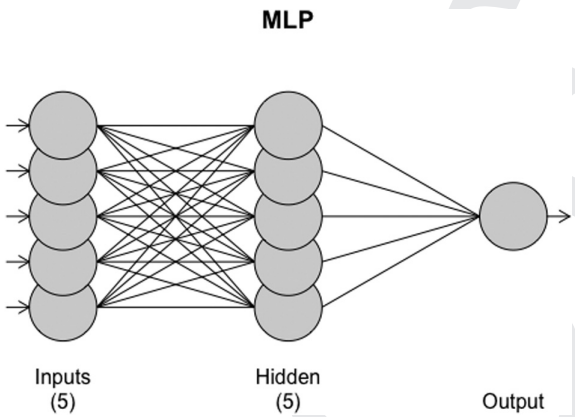


FIGURE 11.16 Architecture of the MLP to fit the AirPassenger data with a first difference and a twelfth difference. This architecture has five input nodes, one hidden layer with five hidden nodes, and a single output layer

As before, we can see from the output and the plot that lags 1,3,4,7 and 9 make up the inputs and that the architecture includes a single hidden layer with five hidden nodes. However, we also now see in the second line of the output that the first and twelfth difference were also included (`Series modeled in differences: D1D12.`). While we can see that the MSE for one-step-ahead forecasts is 53.5823, assuming we wanted to evalulate this model's performance with a horizon of 36 as in Chapters 2 and 6, we would issue the following commands to calculate the rolling window RMSE:

```
rwfit1.12.H5 = roll.win.rmse.nn.wge(AirPassengers,horizon = 36, fit.1.12.H5)
```

```
"Please Hold For a Moment, TSWGE is processing the Rolling Window RMSE with 86
windows."
"The Summary Statistics for the Rolling Window RMSE Are:"

  Min.     1st Qu.    Median      Mean   3rd Qu.      Max.
 12.32       27.39     34.74     36.47     43.97     68.11
"The Rolling Window RMSE is: 36.474"
```

We will use this model's rolling window RMSE = 36.474 as a baseline and fit a few competing models (architectures). We will fit the first of these competing models by using the **`mlp`** function to autoselect the difference(s). This can be done by simply *not* specificying the **`difforder`**:

```
fit.1.H5 = mlp(AirPassengers,allow.det.season = FALSE)
fit.1.H5
```

```
MLP fit with 5 hidden nodes and 20 repetitions.
Series modelled in differences: D1.
Univariate lags: (1,2,3,4,5,6,7,8,9,10,11,12)
Forecast combined using the median operator.
MSE: 20.6619.
```

We see that the function has automatically selected a model with a single difference and lags for each of the previous 12 observations, which reduced the one-step-ahead MSE to 20.6619. Again, being cautious when using the in-sample MSE, the code below shows that the rolling window RMSE for this model with

a horizon of 36 has been reduced to 12.015. This is strong evidence that the first difference is a useful filter in modeling these data!

```
rwfit.1.H5 = roll.win.rmse.nn.wge(AirPassengers,horizon = 36, fit.1.H5)
```

```
"Please Hold For a Moment, TSWGE is processing the Rolling Window RMSE with 94
windows."
"The Summary Statistics for the Rolling Window RMSE Are:"

 Min.   1st Qu.   Median     Mean  3rd Qu.     Max.
8.355    10.682   11.797   12.015   13.458   15.611
"The Rolling Window RMSE is: 12.015"
```

As mentioned earlier, the **mlp** function will also allow for seasonal indicator variables to be added to the model. If we set **allow.det.season = TRUE,** the **mlp** function will automatically add seasonal indicator variables that correspond to the frequency of the series stored in the frequency attribute of the **ts** object. Recall that the **AirPassenger ts** object has a frequency of 12 (displayed below) and thus 11 deterministic seasonal indicator (dummy) variables will be added to the model. Note that **difforder** and **lags** are also not specified and therefore these will be automatically selected. The code to fit this MLP is below and the architecture is illustrated in Figure 11.17.

```
frequency(AirPassengers)
```

```
[1] 12
```

```
fitSeasonal = mlp(AirPassengers,allow.det.season = TRUE)
# same as fitSeasonal = mlp(AirPassengers)
```

```
fitSeasonal
```

```
MLP fit with 5 hidden nodes and 20 repetitions.
Series modelled in differences: D1.
Univariate lags: (1,2,3,4,5,6,7,8,10,12)
Deterministic seasonal dummies included.
Forecast combined using the median operator.
MSE: 8.7509.
```

```
plot(fitSeasonal)
```

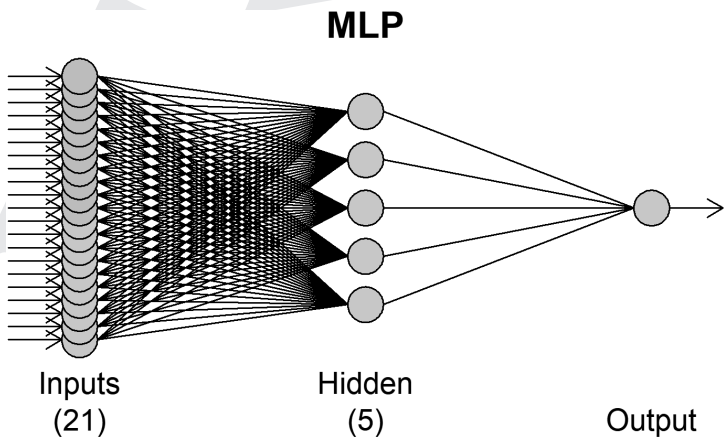


FIGURE 11.17 Architecture of the MLP to fit the AirPassenger data, with a first difference and seasonal indicator (dummy) variables

Again, we can see that five hidden nodes, a first difference, ten lags (9 and 11 were not included), and seasonal indicators (dummies) were included in the final model. Moreover, with a one-step-ahead MSE of 8.7509, we can see that there is evidence that this model is an improvement over the previous two. However, since we are interested in forecasting the next three years, we will evaluate how this model performs with a horizon of 36.

```
rwfitSeasonal = roll.win.rmse.nn.wge(AirPassengers,h = 36, fit_model = fitSeasonal)
"Please Hold For a Moment, TSWGE is processing the Rolling Window RMSE with 1
windows."
"Seasonal MLP models will be evaluated only on the last window."
"The Summary Statistics for the Rolling Window RMSE Are:"
 Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
11.58     11.58     11.58     11.58     11.58     11.58
"The Rolling Window RMSE is: 11.581"
```

Due to the methods the **mlp** function uses to estimate the seasonal indicator variables, note the second line of output from the **roll.win.rmse.nn.wge** call above: "*Seasonal MLP models will be evaluated only on the last window*". In this case, the last window is the last three years of data, which provides some evidence of improved fit with a *single* window RMSE of 11.581. Although the MLP with the seasonal indicators has the lowest RMSE, it is also based on a single window and this is much more dependent on the window on which it is calculated. Closer inspection of the rolling window code for the D1 model with 12 lags reveals that the median of the 94 rolling window RMSEs generated for this model (11.797) is greater than the single window RMSE generated for the the seasonal model (11.581). Since more than half of the rolling window RMSEs from the D1 with 12 lags model are greater than the RMSE of the last window of the seasonal MLP, we will obtain our final forecasts of the next 36 months with the seasonal indicator MLP model. These forecasts are visualized in Figure 11.18 below.

TABLE 11.2 Rolling Window RMSEs for Three Models Fit to the Airline Data

MODEL	*rwRMSE*	*NUMBER OF WINDOWS*
"Airline Model" (D1 and D12 Included with 5 Lags)	36.474	86
D1 with 12 Lags	12.051	94
Seasonal Indicators, D1 and 10 Lags	11.581	1

The final step is to forecast the next 36 months of passengers using the D1 with 12 Lags model. We will now use *all* the data to make these predictions.

```
t = 1:180
fitSeasonal = mlp(AirPassengers,allow.det.season = TRUE)
finalFit = forecast(fitSeasonal, h = 36)
plot(t[1:144], AirPassengers, type = "l", xli = c(0,185), ylim = c(100, 800),
xlab = "Time")
lines(t[145:180], finalFit$mean,lty = 3, lwd = 3)
```

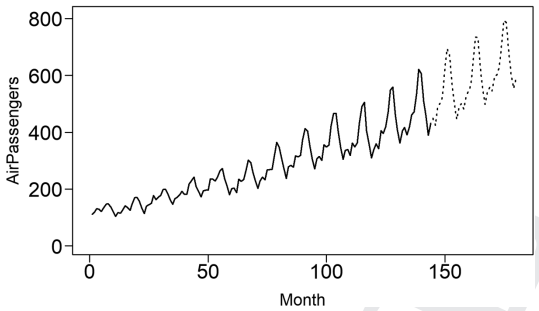


FIGURE 11.18 Final forecasts of the next three years of the **AirPassenger** series from the MLP with seasonal indicator variables

11.4 THE EXTENDED PERCEPTRON FOR MULTIVARIATE TIME SERIES DATA

11.4.1 Forecasting Melanoma Using Sunspots

11.4.1.1 Architecture

Multivariate time series data are easily accomodated by the multilayered perceptron neural network model. Consider the 37 years of melanoma and sunspot data (**`melanoma2.0`**) studied in Chapter 10. Figure 11.19 is an MLP architecture designed to model melanoma at time *t* based on melanoma at time *t-1* and sunpot at time *t-1*.

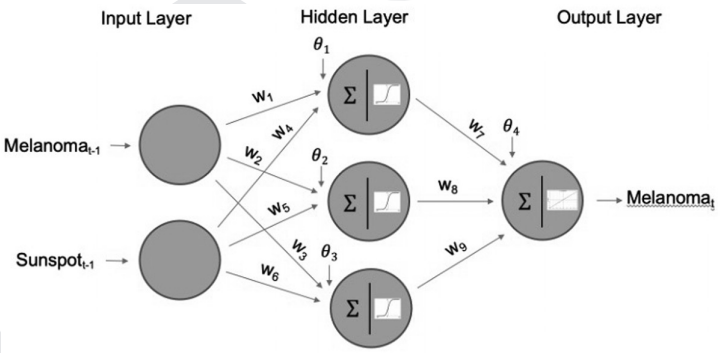


FIGURE 11.19 MLP architecture that models the melanoma count at time t (Melanoma$_t$) given the melanoma and sunspot counts at the previous observation (Melanoma$_{t-1}$ and Sunspot$_{t-1}$, respectively)

11.4.1.2 Fitting the Baseline Model

We will use this model as a baseline and assess it for the purpose of forecasting the next eight years of melanoma counts. Below, a training set is created with the first 29 years of data while the last eight years of data are reserved as a test set for cross-validation. The training data are then used to fit the

MLP architecture in Figure 11.19. Note that the sunspot data are entered into the model through the **mlp** function's **xreg** argument, which requires a data frame.

```
SMtrain = melanoma2.0[1:29,]
SMtest = melanoma2.0[30:37,]

SMtrainDF = data.frame(Sunspot = ts(SMtrain$sunspot))

fit.mlp1 = mlp(ts(SMtrain$melanoma), lags = 1,reps = 20,comb = "median",
xreg = SMtrainDF, xreg.lags = 1, allow.det.season = FALSE, hd = 3, difforder = 0,
sel.lag = FALSE)
```

The details of the model can be viewed as usual below.

```
fit.mlp1
```

```
MLP fit with 3 hidden nodes and 20 repetitions.
Univariate lags: (1)
1 regressor included.
- Regressor 1 lags: (1)
Forecast combined using the median operator.
MSE: 0.1492.
```

The description clearly notes that the MLP had two input nodes (melanoma at lag 1 and sunspot at lag 1), three hidden nodes, and resulted in a one-step-ahead forecast MSE of .1492. A plot of the architecture (Figure 11.20) is provided as usual and is consistent with the architecture in Figure 11.19.

```
plot(fit.mlp1)
```

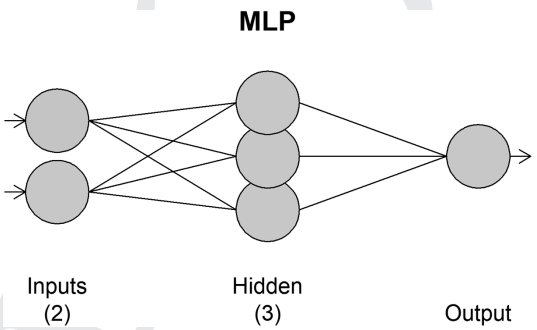


FIGURE 11.20 fit.mlp model architecture diagram

11.4.1.3 Forecasting Future Sunspot Data for Predicting Future Melanoma

Next, the rolling window RMSE for the MLP is not implemented for the multivariate case, so we will evaluate this model based on the RMSE of the last eight observations (effectively, the last window of a rolling window RMSE). To do so, as is often the case with multivarite time series, it is not realistic that we would know the sunspot number for future years; thus, the sunspot data for the last eight years of the series must be forecast. We will do this with a simple univariate MLP. The code and sunspot forecasts are below, as well as a plot of the forecasts in Figure 11.21.

```
fit.mlp.SP = mlp(ts(SMtrainDF$Sunspot),reps = 20, comb = "median")
fore.mlp.SP = forecast(fit.mlp.SP, h = 8)
fore.mlp.SP
```

```
Point            Forecast
30                53.63687
31               161.85283
32               228.67077
33               220.15203
34               204.40068
35               143.16898
36                95.58033
37                62.55229
```

```
plot(fore.mlp.SP)
```

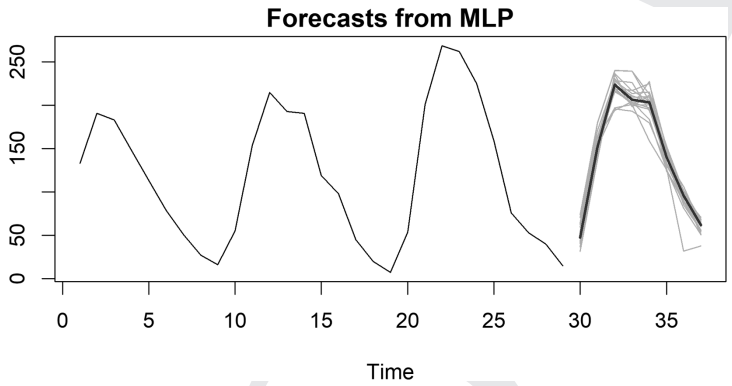


FIGURE 11.21 Plot displaying the 20 forecasts for each time period and their median (bold)

It is important to note here when forecasting observations 30−37, it is realistic to assume that we know the actual sunspots for observations 1–29. For this reason, the sunspot forecasts for observations 30–37 must be appended to the assumed known sunspots in the training set and then built into a data frame. Based on the output, it is clear the forecasts start at observation 30, since the known values will be whole numbers and will have decimal values of 0 while the forecasted values from 30−37 (the last column) contain decimals.

```
SMDF_fore = data.frame(Sunspot = ts(fore.mlp.SP$mean))
SMDF = data.frame(Sunspot = ts(c(SMtrainDF$Sunspot,SMDF_fore$Sunspot)))
```

```
  Sunspot          10 55.00000       20  54.00000       30  53.63687
1 133.00000        11 154.00000      21 201.00000       31 161.85283
2 191.00000        12 215.00000      22 269.00000       32 228.67077
3 183.00000        13 193.00000      23 262.00000       33 220.15203
4 148.00000        14 191.00000      24 225.00000       34 204.40068
5 113.00000        15 119.00000      25 159.00000       35 143.16898
6  79.00000        16  98.00000      26  76.00000       36  95.58033
7  51.00000        17  45.00000      27  53.00000       37  62.55229
8  27.00000        18  20.00000      28  40.00000
9  16.00000        19   7.00000      29  15.00000
```

11.4.1.4 Forecasting the Last Eight Years of Melanoma

The last eight years of melanoma data can now be forecasted. The code and forecasts are below and a plot of the forecasts is in Figure 11.22.

```
fore.mlp = forecast(fit.mlp1, h = 8, xreg = SMDF)
fore.mlp
```

```
Point        Forecast
30           4.058019
31           3.985312
32           3.765170
33           3.857733
34           3.873401
35           3.826858
36           3.645963
37           3.583501
```

```
plot(fore.mlp)
```

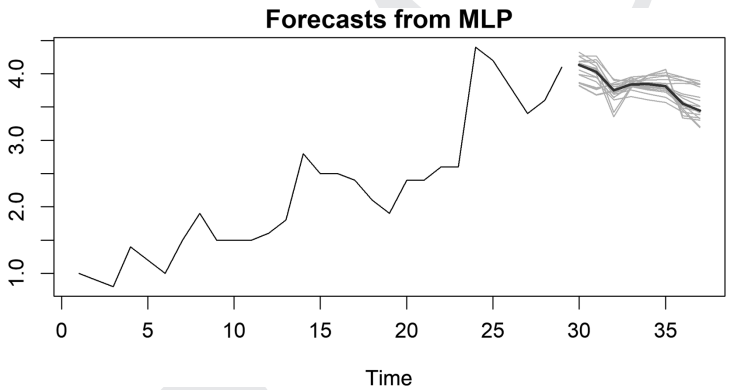


FIGURE 11.22 Melanoma forecasts from fit.mlp1 with sunspot forecasts from fit.mlp.SP

(1) Assessing the Baseline Model on the Last Eight Years of Melanoma

The forecasts seen in Figure 11.21 leave quite a bit to be desired since they do not appear to reflect the strong evidence of an upward trend in melanoma counts. Nevertheless, an RMSE can be calculated for the last eight melanoma values of the series, which results in the baseline RMSE of .90852:

```
RMSE = sqrt(mean((SMtest$melanoma - fore.mlp$mean)^2))
RMSE
```

```
0.9955416
```

11.4.1.5 Fitting a Competing Model

Remember that a useful VAR model based on these data in Chapter 10 made use of the melanoma and sunspot data at lag four, so there is hope that this model may have room for improvement. The code below allows the **mlp** function to autoselect the lags and differencing order so as to try and improve the baseline model above:

```
fit.mlp2 = mlp(ts(SMtrain$melanoma), reps = 20,comb = "median", xreg = SMtrainDF,
allow.det.season = FALSE)
```

The resulting model displayed in Figure 11.23 includes a first difference of the melanoma series, which contains melanoma and sunspot data up to lag three and a hidden layer with five nodes. The MSE is not a comparable measure of fit in this case since the data have been differenced, meaning that the resulting MSE (.001) is calculated on the differenced data, rather than the raw data.

```
fit.mlp2
```

```
MLP fit with 5 hidden nodes and 20 repetitions.
Series modelled in differences: D1.
Univariate lags: (1,2,3)
1 regressor included.
- Regressor 1 lags: (3,4)
Forecast combined using the median operator.
MSE: 0.001.
```

```
plot(fit.mlp2)
```

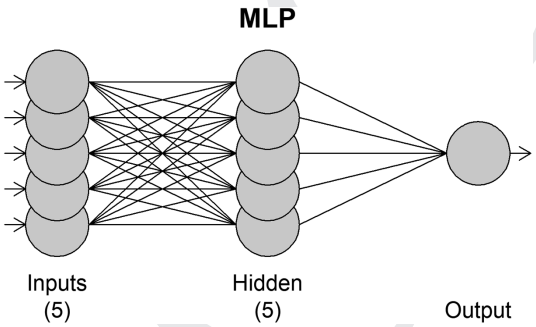


FIGURE 11.23 fit.mlp2 model description and architecture diagram

11.4.1.6 Assessing the Competing Model on the Last Eight Years of Melanoma Data

Forecasts are provided below and visual evidence that this is a more useful model than our baseline is provided in Figure 11.24. The forecasts now appear to be reflecting the trend, and possibly a seasonal component in the melanoma series.

```
fore.mlp = forecast(fit.mlp2, h = 8, xreg = SMDF)
fore.mlp
```

```
30              4.145854
31              4.208989
32              4.191491
33              4.403321
34              5.568593
35              5.317146
36              5.170883
37              4.848404
```

```
plot(fore.mlp)
```

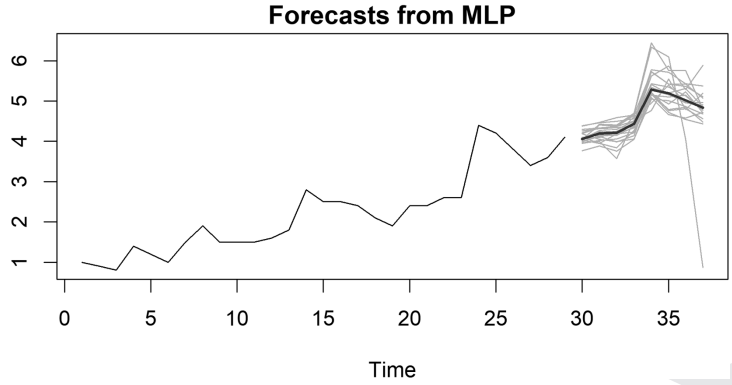


FIGURE 11.24 Melanoma forecasts and a plot summarizing their calculation from model fit.mlp2

Finally, the RMSE of the last eight values is calculated below. The RMSE of .6118 provides evidence that this model is a considerable improvement over the baseline model (RMSE .9955). In fact, this RMSE is also evidence of an improvement over the VAR model fit to this data in Chapter 10 (RMSE = 0.7870964)

```
RMSE = sqrt(mean((SMtest$melanoma - fore.mlp$mean)^2))
RMSE
```

```
0.6117655
```

11.4.1.7 Forecasting the Next Eight Years of Melanoma

Now that the final model has been identified, the MLP will need to be refit using *all the data* (**fit.mlp3**) and then melanoma forecasts for the next eight years can be calcuated and visualized. The code to perform these steps is shown below, and the output that summarizes the model is in Figure 11.25, while the forecasts are displayed in Figure 11.26.

```
#refit model using all the data
fit.mlp3 = mlp(ts(melanoma2.0$melanoma), reps = 20,comb = "median", xreg = data.
frame(Sunspot = melanoma2.0$sunspot), allow.det.season = FALSE)

fit.mlp3
```

```
MLP fit with 5 hidden nodes and 20 repetitions.
Series modelled in differences: D1.
Univariate lags: (1,2,3)
1 regressor included.
- Regressor 1 lags: (3,4)
Forecast combined using the median operator.
MSE: 0.0046.
```

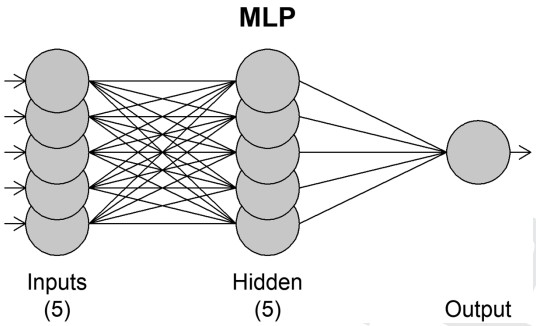


FIGURE 11.25 Architecture of the final MLP fit using the full dataset

```
#forecast the next 8 years of sunspot to use in the forecasts
fit.mlp.SP = mlp(ts(melanoma2.0$sunspot),reps = 50, comb = "median")
fore.mlp.SP = forecast(fit.mlp.SP, h = 8)

#put the sunpsots in a data frame so they can be input in mlp()
SMDF_fore = data.frame(Sunspot = ts(fore.mlp.SP$mean))
SMDF = data.frame(Sunspot = ts(c(melanoma2.0$sunspot,SMDF_fore$Sunspot)))

#calculate and visualize the forecasts
fore.mlp = forecast(fit.mlp3, h = 8, xreg = SMDF)
fore.mlp
```

```
Point           Forecast
38              5.202662
39              5.104457
40              5.527732
41              5.260273
42              5.534467
43              5.674302
44              5.717306
45              5.987818
```

QR 11.3 Combining Forecasts with Mean vs. Median

```
plot(fore.mlp)
```

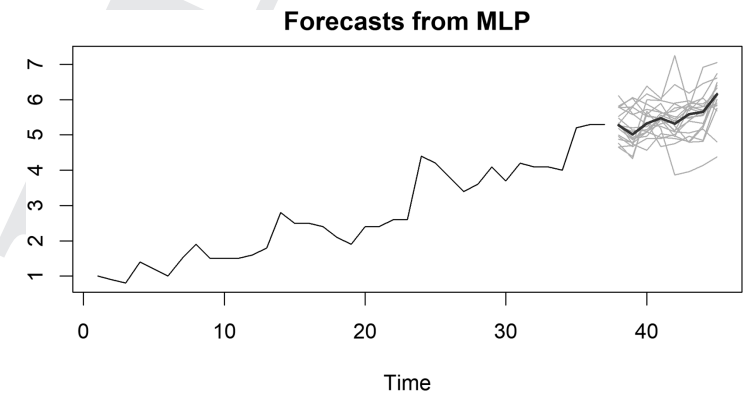


FIGURE 11.26 Forecasts of the next eight years (1980–1987) from the MLP model

11.4.2 Forecasting Cardiac Mortality Using Temperature and Particulates

In this section, we will return to the problem of modeling the weekly cardiac mortality in LA from 1970 to 1979 with temperature and particulates as explanatory variables. Our eventual goal will be to forecast a year in advance; we will compare two candidate models based on their fit (RMSE) to the last year (52 weeks) of the series. Since it is not realistic to know the temperature and particulate counts in the future, these will need to first be forecast for later use in our candidate MLP models aimed at forecasting cardiac mortality. After we have forecasted the next year of our explanatory variables (temperature and particulates) we will first fit a model that addresses the periodic behavior in the data through correlatons in the data. Second, a competing model will be fit that explicitly models this seasonal behavior using deterministic seasonal indicator (dummy) variables. These models will be compared using the RMSE of the fit of the cardiac mortiality of the *last* 52 weeks of the series; then, the final model will be used to forecast the cardiac mortality for the *next* 52 weeks.

11.4.2.1 General Architecture

Both models considered in this section will contain temperature and particulates as explanatory variables, while only the second model will use seasonal indicator variables to model the mean of each week explicitly. Figure 11.27 displays the general architecture employed in both models. Note that there are potentially many input nodes associated with the lags of the two explanatory variables, temperature, and particulates. Note, for simplicity, there is only one arrow drawn between each explanatory variable's input nodes and the hidden nodes. These arrows represent the fact that there is really a unique arrow connecting each node to each hidden layer to form a "fully conncected neural network". Also note that the asterisk next to the "Seasonal Indicators" group of nodes indicates that they are present only in the second model which addresses the periodic behavior with 51 indicator variables (51 additional input nodes.).

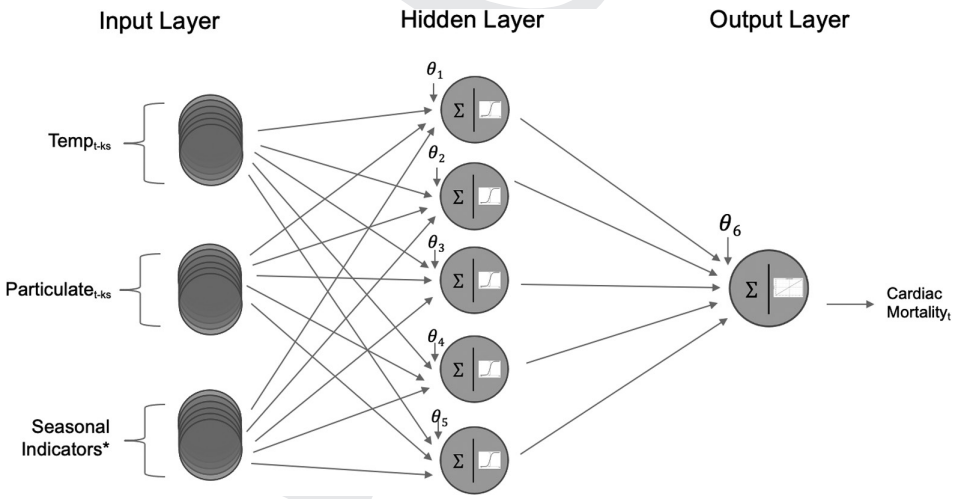


FIGURE 11.27 General archictecure of the MLPs used to model cardiac mortality.
* The seasonal indicator variables will only be present in the second model.

11.4.2.2 Train / Test Split

A useful first step in forecasting is to divide the dataset into a training and test set. Since the RMSE will be calculated on the last 52 weeks of the dataset, as was done in Chapter 10, the training set will consist of the data between 1970 and the first 40 weeks of 1978 and the test set will consist of the last 52 weeks of the dataset (the last 12 weeks of 1978 and the first 40 weeks of 1970).

```
cardiacTrain = window(cardiac, start = c(1970,1), end = c(1978,40))
cardiacTest = window(cardiac, start = c(1978,41), end = c(1979,40))
```

11.4.2.3 Forecasting Covariates: Temperature and Particulates

Unlike the VAR models we fit in Chapter 10 which forecast the explanatory variables simultaneously with the response, MLP models require the explanatory variables to be forecast in advance if the future values are not known ahead of time. Temperature and particulates are not known in advance, so we will use univariate MLPs to forecast these values for the weeks in the year 1979; these forecasts will be used in the calculation of the RMSE. Later, we will forecast the temperature and particulate values for the weeks of 1980 to calculate our final forecasts.

(1) Temperature

It is intuitive that temperature would have seasonal behavior; the seasonal indicator variables which we have previously fit are available in the MLP model. In this case, 51 (the frequency of the **ts** object minus one) indicator variables corresponding to the week of the year are automatically added if **allow.det.season** is set to **TRUE**, **det.type** is set to **"bin"** and if their inclusion reduces the overall one-step-ahead MSE.[11] The code to fit the MLP is below and the output (11.2), architecture (Figure 11.28(a) and a plot of the forecasts (Figure 11.28(b) follow. The output suggests that sufficient fit of the temperature series was achieved using only 16 of the lags from 1 to 52; note that the 51 seasonal indicator variables were left out.

```
# Temperature
fit.mlp.temp = mlp(ts(cardiacTrain[,"tempr"],frequency = 52),reps = 50,
difforder = 0, comb = "median", allow.det.season = TRUE, det.type = "bin")
plot(fit.mlp.temp)
fore.mlp.temp = forecast(fit.mlp.temp, h = 52)
```

```
MLP fit with 5 hidden nodes and 50 repetitions.
Univariate lags: (1,2,4,8,9,12,13,22,28,35,41,42,47,49,50,52)
Forecast combined using the median operator.
MSE: 10.342.
```

(11.2)

```
plot(fore.mlp.temp)
```

11 In addition to the binary indicator variables we used in Chapter 10 (**det.type = "bin"**), the mlp function allows for a sin/cos based method as well (**det.type = "trg"**).[11] See page 12 and 13 of https://kourentzes.com/forecasting/wp-content/uploads/2016/07/Barrow-Kourentzes-2016-impact-special-days.pdf for more information.

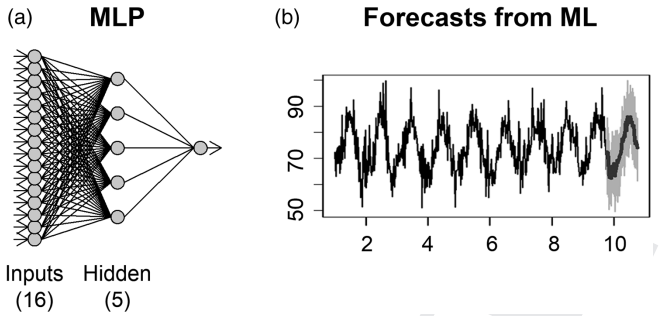


FIGURE 11.28 (a) Architecture of the MLP fit for the temperature series (b)Plot of the forecasts of the last 52 weeks of the temperture data

(2) Particulates

A univariate MLP was used to model the particulates as well. The code to fit this model is below and the output (11.3), architecture (Figure 11.29(a) and a plot of the forecasts (Figure 11.29(b) follow. Similar to forecasting the temperatures, a model with seasonal indicators was not deemed to be as useful as a model with only 14 lags (11.3).

```
# Particulates
fit.mlp.part = mlp(ts(cardiacTrain[,"part"],frequency = 52),reps = 50,
difforder = 0, comb = "median", allow.det.season = TRUE, det.type = "bin")
```

```
MLP fit with 5 hidden nodes and 50 repetitions.
Univariate lags: (2,3,4,6,12,14,17,28,29,39,43,50,51,52)                    (11.3)
Forecast combined using the median operator.
MSE: 29.8761.
```

```
plot(fit.mlp.part)

fore.mlp.part = forecast(fit.mlp.part, h = 52)
plot(fore.mlp.part)
```

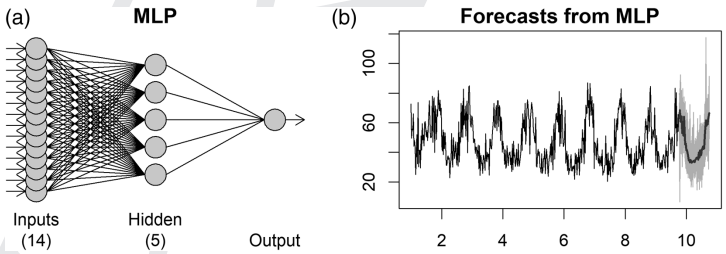


FIGURE 11.29 (a) Architecture of the MLP fit for the particulate series (b) Plot of the forecasts of the last 52 weeks of the particulates series (last 12 weeks of 1978 and first 40 weeks of 1979)

11.4.2.4 Model Without Seasonal Indicator Variables

Now that the temperatures and particulates have been forecast for the last year of the series, models can now be fit to forecast cardiac mortality. The first candidate model considered does not have the option of adding seasonal indicators to the MLP. The code and output below show that a model with a first

difference, 60 lagged variables between temperature, particulates and cardicac mortality, and 5 hidden layers was selected. Figure 11.30(a) and (b) show the selected architecture of the model and a plot of the final forecasts respectively.

```
cardiacDF_xreg = data.frame(temp = ts(cardiacTrain[,"tempr"]),
part = ts(c(cardiacTrain[,"part"])))

fit.mlp.cmort = mlp(ts(cardiacTrain[,"cmort"], frequency = 52),reps = 50,
comb = "median",xreg = cardiacDF_xreg, allow.det.season = FALSE)
fit.mlp.cmort
```

```
MLP fit with 5 hidden nodes and 50 repetitions.
Series modelled in differences: D1.
Univariate lags: (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,
24,25,26,27,28,29,30,31,32)
2 regressors included.
- Regressor 1 lags: (1,4,14,16,17,18,20,23,27,30,38,44,50,52)
- Regressor 2 lags: (1,7,9,11,16,20,21,26,33,34,44,48,49,50)
Forecast combined using the median operator.
MSE: 0.0621.
```

```
plot(fit.mlp.cmort)
```

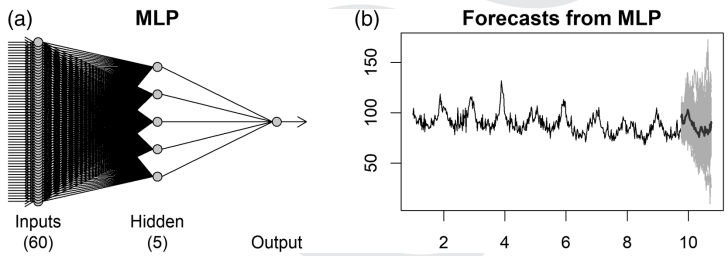


FIGURE 11.30 (a) The architecture of the MLP for cardiac mortality fit.mlp.cmort. (b) the forecasts for the next 52 weeks with the median in bold

Recall that the model above was fit using the data in **`cardiacTrain`**, which contains only the data from 1970 to the first 40 weeks of 1978. The next step is to evaluate this model by assessing its performance on the hold-out set (the last 52 weeks of the dataset). A realistic scenario would be that we know the temperatures and particulates through week 40 of 1978 but would need to use forecast values to predict cardiac mortalities in the remaining 52 weeks. The code below creates a data frame that reflects this structure; the forecast temperatures forecasted earlier (**`fore.mlp.temp$mean`**) are appended to the known temparatures (**`cardiacTrain[,"tempr"]`**) in a column called "**`temp`**", and the same is done for the particulates in a column called "**`part`**".

```
CMDF_fore = data.frame(temp = ts(c(cardiacTrain[,"tempr"],fore.mlp.temp$mean)),
part = ts(c(cardiacTrain[,"part"],fore.mlp.part$mean)))
```

The final step is to forecast cardiac mortalities for the last 52 weeks of the series (displayed in Figure 11.30(b)) and assess the model by calculating the RMSE of the forecasts using the code below.

```
fore.mlp.cmort = forecast(fit.mlp.cmort, h = 52, xreg = CMDF_fore)
plot(fore.mlp.cmort)
RMSE = sqrt(mean((cardiacTest[,"cmort"][1:52] - fore.mlp.cmort$mean)^2))
RMSE
```

```
7.00397
```

The RMSE for the last 52 weeks of the series was found to be 7.00397, which will be compared to the RMSE of a model that allows for seasonal indicator variables; this model will be fit and assessed next.

11.4.2.5 Model With Seasonal Indicator Variables

The code below fits a model that allows for seasonal indicator variables (**`allow.det.season = TRUE, det.type = "bin"`**). The code and output below (11.4) and the diagram in Figure 11.31(a) indicate that a model with a first difference, 31 total lags for cardiac mortality, 15 total lags for temperature, 15 total lags for particulates, and 51 seasonal indicator variables was selected based on the data in the training set (**`cardiacTrain`**).

```
cardiacDF_xreg = data.frame(temp = ts(cardiacTrain[,"tempr"]), part =
ts(c(cardiacTrain[,"part"])))

fit.mlp.cmortS = mlp(ts(cardiacTrain[,"cmort"],frequency = 52),reps = 50,comb
= "median",xreg = cardiacDF_xreg, allow.det.season = TRUE, det.type = "bin")
fit.mlp.cmortS
```

```
MLP fit with 5 hidden nodes and 50 repetitions.
Series modelled in differences: D1.
Univariate lags: (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,23,24,
26,31,33,46,47,48,49,51)                                                    (11.4)
2 regressors included.
- Regressor 1 lags: (1,4,5,14,16,17,20,23,27,30,31,38,44,46,50)
- Regressor 2 lags: (2,7,16,18,20,21,26,29,31,32,33,34,44,49,50)
Deterministic seasonal dummies included.
Forecast combined using the median operator.
MSE: 1e-04.
```

```
plot(fit.mlp.cmortS)
```

QR 11.4 MLP with Seasonal Variables

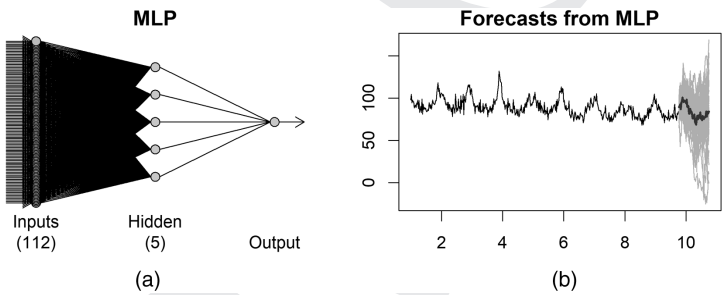


FIG 11.31 (a) The architecture of the MLP for cardiac mortality fit.mlp.cmortS (b) the forecasts for the next 52 weeks with the median in bold

As before, a realistic scenario when forecasting the last 52 weeks of cardiac mortality is that the temperatures and particulates for 1970 through the first 40 weeks of 1978 are known, and that these values would need to be forecast for the last 52 weeks (the test set). For this reason, the *forecast* temperatures and particulates for the last 52 weeks of the dataset are appended to the *known* temperatures and particulates for the training set, and then stored in a dataframe.

```
CMDF_fore = data.frame(temp =
ts(c(cardiacTrain[,"tempr"],fore.mlp.temp$mean)), part =
ts(c(cardiacTrain[,"part"],fore.mlp.part$mean)))
```

These explanatory variables (**CMDF_fore**) are then passed with the model (**fit.mlp.cmortS**) to the **forecast** function which forecasts the cardiac mortalities for the last 52 weeks. Finally, the model is assessed by calculating the RMSE of the 52 forecasts, which was found to to be 6.440517. The code for the visualization of the forecasts and calculation of the RMSE for the forecasts of the last 52 weeks is below. Figure 11.29(b) is a plot of these forecasts.

```
fore.mlp.cmortS = forecast(fit.mlp.cmortS, h = 52, xreg = CMDF_fore)
plot(fore.mlp.cmortS)
RMSE = sqrt(mean((cardiacTest[,"cmort"][1:52] - fore.mlp.cmortS$mean)^2))
RMSE
```

```
6.440517
```

11.5 AN "ENSEMBLE" MODEL

An "ensemble" model is one that is a combination of two or more individual models. For instance, taking the median forecast for each time point from the 50 repetitions of the MLP (technically, 50 different MLPs) is an example of an ensemble model. Forecasts from both univariate and multivariate models can be "ensembled". In this section, an ensemble of a VAR and a multivariate MLP model is considered.

Figure 11.32(a) displays the 52 forecasts for the VAR with p = 2 (including trend and seasonality) that was studied in Chapter 10 and had an RMSE of 6.38. It will be important to note that there is evidence that this model consistently underestimates the cardiac mortality for a large portion of the test set, especially from weeks 466 and later. On the other hand, Figure 11.32(b) displays the forecasts for the previously fit MLP with seasonality (solid)−note that, with the exception of weeks 484−486, these forecasts tend to be greater than the VAR forecasts (dashed). While the MLP model had an RMSE of 6.44, it stands to reason that averaging the forecasts from these two models may yield forecasts that outperform both the individual RMSEs.

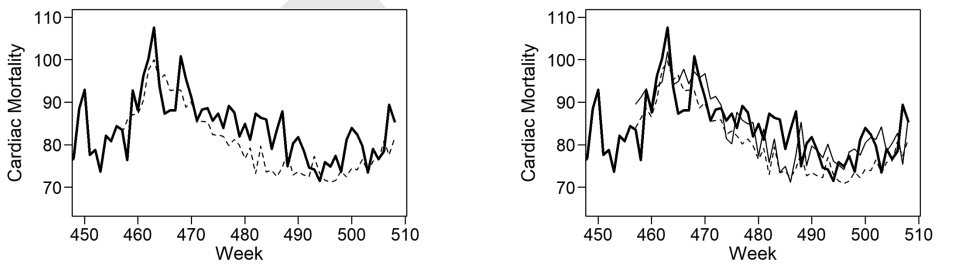


FIGURE 11.32 (a) Forecasts from VAR(2) with Trend and Seasonality (b) Forecasts from MLP (solid) versus VAR(7) With Seasonality (dashed)

The VAR model with p = 2 (including trend and seasonality) is fit again below for completeness, and the forecasts are stored in the object **preds2S**.

```
CMortVAR2S = VAR(cardiacTrain, season = 52, type = "both", p = 2)
preds2S=predict(CMortVAR2S,n.ahead=52)
```

Next, we recall that the forecasts from the MLP with seasonality were stored in the object **fore.mlp.cmortS$mean.** The code below shows the calculation of a very simple ensemble in which the forecasts from these two models are averaged.

```
ensemble = (preds2S$fcst$cmort[,1] + fore.mlp.cmortS$mean)/2
```

Voilà! Figure 11.33 displays the forecasts produced by the code below and yields an RMSE of 6.058764, which is lower than the RMSE from either of the individual VAR and MLP forecasts.

```
#Plot
plot(seq(1,508,1), cardiac[,"cmort"], type = "l",xlim = c(450,508), xlab =
"Time", ylab = "Cardiac Mortality", main = "52 Week Cardiac Mortality Forecast
From A VAR/MLP Ensemble")
lines(seq(457,508,1), ensemble, type = "l", lwd = 4, col = "green")
```

```
RMSEENSEMBLE = sqrt(mean((cardiacTest[,"cmort"] - ensemble)^2))
RMSEENSEMBLE
```

```
6.058764
```

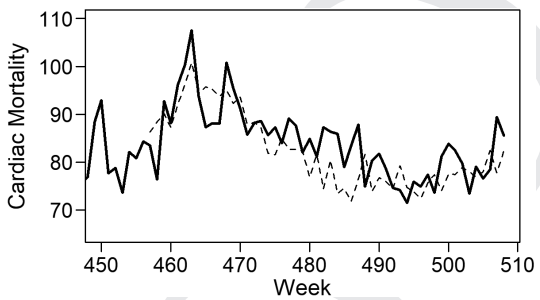


FIGURE 11.33 Forecasts from the ensemble model (mean of VAR and MLP models)

11.5.1 Final Forecasts for the Next Fifty-Two Weeks

Including our analysis in Chapter 10, we have fit several VAR and MLP models to the cardiac mortality data with the goal of making forecasts through the next year. Table 11.2 below summarizes the results with respect to the RMSE of the forecasts on the hold-out set (**`cardiacTest`**.)

TABLE 11.3 Table of VAR, MLP, and Ensemble Performance with a 52-week Horizon.

MODEL	*RMSE*
VAR(7) with Trend	5.44
VAR(2) with Trend and Seasonality	6.38
MLP with seasonality	6.44
Ensemble: VAR(2) with Trend/Seasonality and MLP with Seasonality	6.05

The VAR(7) with trend from Chapter 10 achieved the lowest RMSE (RMSE = 5.44) on the 52-week hold-out set, outperforming even the ensemble model (RMSE = 6.05) by a notable margin. For this reason, the VAR(7) with trend was selected as the model to provide the forecasts of the next 52 weeks. For convenience and reference, the plot with these forecasts and 95% prediction intervals can be seen in Figure 11.34.

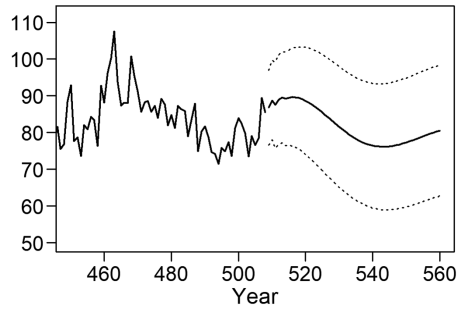


FIGURE 11.34 Forecasts and 95% prediction intervals for the weekly cardiac mortalty rates of LA county for the last 12 weeks of 1979 and first 40 weeks of 1980 from the VAR model with p= 7, including trend

11.5.2 Final Forecasts for the Next Three Years (Longer Term Forecasts)

Remember, since there are no seasonality terms, the forecasts from the VAR(7) with trend model will damp to the trend line, and will thus not forecast the strong seasonal behavior evident in the cardiac mortality in the long term. This is illustrated in Figure 11.35, in which the VAR(7) with trend has been used to forecast the next three years (156 weeks) of cardiac mortalities. The RMSE for these 156 forecasts was found to be 6.66.

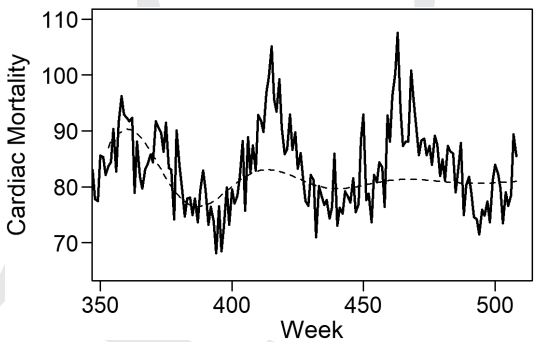


FIGURE 11.35 Three years of forecasts from the VAR(7) with trend. Note the slow, but eventual convergence to the mean / trend line

The VAR(2) with trend and seasonality (Figure 11.36(a)) and the MLP with seasonality (Figure 11.36(b)) were also assessed with respect to the 156-week forecasts, with RMSEs 6.39 and 5.88, respectively. Note that both of these models are not only more visually "satisfying" than the VAR(7) with trend, but have lower RMSEs as well.

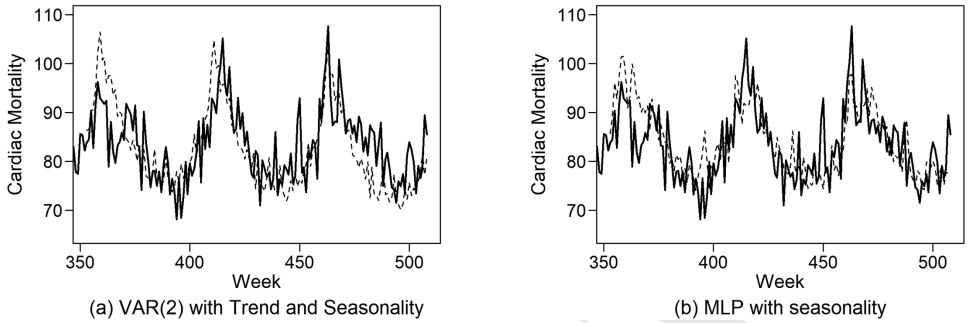


FIGURE 11.36 (a) Three-year forecast using a VAR(2) with trend and seasonality, (b) Three-year forecasts using a MLP model with seasonality

It is visually clear that the MLP forecasts provide closer overall fit to the actual values. We next use the VAR forecasts to construct an ensemble model (Figure 11.37). Averaging the forecasts from these seasonal models results in the ensemble model achieving the lowest RMSE of all the models tested: 5.65. Table 11.4 summarizes these findings. The code for creating Figures 11.36 and 11.37 is included in Appendix 11B.

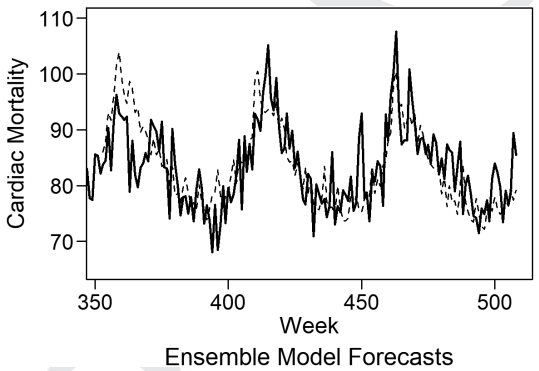


FIGURE 11.37 Three years of cardiac mortality forecasts from the ensemble model (mean of the VAR and MLP forecasts)

TABLE 11.4 Table of VAR, MLP and Ensemble Performance on a 3-year (156-week) Horizon.

MODEL	*RMSE*
VAR(7) with Trend	6.66
VAR(2) with Trend and Seasonality	6.34
MLP with Seasonality	5.89
Ensemble: VAR(2) with Trend/Seasonality and MLP with Seasonality	5.65

11.6 CONCLUDING REMARKS

In this chapter, multilayered perceptron (MLP) models were studied with respect to their forecasting of time series data. It has been shown in this chapter, as well as in other areas of research, that MLPs have the *potential* to obtain "better" forecasts with respect to other modeling technologies. This potential stems from the MLP's ability to include hidden layers and various activation functions that allow for the modeling of non-linear relationships that may be missed by other methods. This is certainly a noteworty potential advantage of MLP models. This advantage, on the other hand, often comes at the cost of significantly longer training times, lengthy hyperparameter tuning, lack of dependable confidence intervals, and black box models that lack interpretablity. Many time series applications can tolerate these costs and, in the coming years, increases in computational power as well as new research developments and innovations will almost certainly mitigate some of these concerns.

It is important to underline that currently, the advantage of the MLP model is the *potential* to obtain "better" forecasts. We have witnessed an example of an MLP model outperforming the VAR in forecasting melanoma occurence. Alternatively, a VAR model was shown to outperform the MLP in the 52-week forecast of cardiac mortality data. Interestingly, an ensemble of the MLP and the VAR was shown to outperform the individual MLP *and* VAR models for 3-year forecasts of cardiac mortality! The take-away is that the time series analyst should make room in their toolbox for MLP, VAR, MLR with correlated errors, ARIMA, ARMA, Holt-Winters and any or all of the many additional models that are available and will become available in the future.

APPENDIX 11A

TSWGE FUNCTION

(a) **roll.win.rmse.nn.wge(series,horizon = 1,fit_model)**

creates as many "windows" as is possible with the data and calculates an RMSE for each window. The resulting "rolling window RMSE" is the average of the individual RMSEs from each window.

series =original realization
horizon = the number of observations to be forecast beyond the forecast origin.
fit_model = the mlp object (model) to be evaluated. This model will have been fit before the call to this function.

CRAN Package *nnfor* The above call statement indicates the parameters available in function mlp. See the **mlp** help for more information by typing **help(mlp)** in R.

```
mlp(y, m = frequency(y), hd = NULL, reps = 20, comb = c("median",
  "mean", "mode"), lags = NULL, keep = NULL, difforder = NULL,
  outplot = c(FALSE, TRUE), sel.lag = c(TRUE, FALSE),
  allow.det.season = c(TRUE, FALSE), det.type = c("auto", "bin",
  "trg"), xreg = NULL, xreg.lags = NULL, xreg.keep = NULL,
  hd.auto.type = c("set", "valid", "cv", "elm"), hd.max = NULL,
  model = NULL, retrain = c(FALSE, TRUE))
```

APPENDIX 11B

The following is code used to create Figures 11.36 and 11.37.

```
# VAR / MLP / Ensemble Long Term Forecast: 3 years = 156 Weeks
#
# Code for Figure 11.36 l
#
library(astsa)
data(lap) #original cardiac mortality data from schumway
cardiac = window(lap[,c("cmort","tempr","part")],start = c(1970,1))
cardiacTrain = cardiac[1:352,]
cardiacTest = cardiac[353:508,]
seed = 1
#forecast temp and particles
set.seed(seed)
#temp
fit.mlp.temp=mlp(ts(cardiacTrain[,"tempr"],frequency=52),reps=50,difforder=0,
comb = "median", det.type = "bin")
plot(fit.mlp.temp)
fore.mlp.temp = forecast(fit.mlp.temp, h = 156)
plot(fore.mlp.temp)
plot(ts(cardiacTrain[,"tempr"],frequency = 52))
plot(fore.mlp.temp$mean)
set.seed(seed)
#particles
fit.mlp.part=mlp(ts(cardiacTrain[,"part"],frequency=52),reps=50,difforder=0,
comb = "median", det.type= "bin")
plot(fit.mlp.part)
fore.mlp.part = forecast(fit.mlp.part, h = 156)
plot(fore.mlp.part)
#package them up in data frame. Use the ones you know AND the ones you have to
forecast
CMDF_fore = data.frame(Week = ts(seq(1,508,1)),temp = ts(c(cardiacTrain[,"t
empr"],fore.mlp.temp$mean)), part = ts(c(cardiacTrain[,"part"],fore.mlp.
part$mean)))
CMDF_foreNP = data.frame(Week = ts(seq(1,508,1)), temp = ts(c(cardiacTrain[,"t
empr"],fore.mlp.temp$mean)))
CMDF_foreNT = data.frame(Week = ts(seq(1,508,1)), part =
ts(c(cardiacTrain[,"part"],fore.mlp.part$mean)))
CMDF_foreNW = data.frame(temp = ts(c(cardiacTrain[,"tempr"],fore.mlp.
temp$mean),frequency = 52), part = ts(c(cardiacTrain[,"part"],fore.mlp.
part$mean),frequency = 52))
CMDF_fore = data.frame(temp = ts(c(cardiacTrain[,"tempr"],fore.mlp.temp$mean)),
part = ts(c(cardiacTrain[,"part"],fore.mlp.part$mean)),Week = ts(seq(1,508,1)))
#CMDF_fore
set.seed(seed)
#forecast cmort using mlp with forecasted xreg (don't need to forecast week.)
cardiacDF_xreg = data.frame(Week = ts(seq(1,352,1)),temp =
ts(cardiacTrain[,"tempr"]), part = ts(c(cardiacTrain[,"part"])))
cardiacDF_xregNP = data.frame(Week = ts(seq(1,352,1)), temp = ts(cardiacTrain
[,"tempr"]))
cardiacDF_xregNT = data.frame(Week = ts(seq(1,352,1)), part = ts(c(cardiacTra
in[,"part"])))
```

```
cardiacDF_xregNW = data.frame(temp = ts(cardiacTrain[,"tempr"], frequency =
52), part = ts(c(cardiacTrain[,"part"]),frequency = 52))
cardiacDF_xreg = data.frame(temp = ts(cardiacTrain[,"tempr"]), part = ts(c(car
diacTrain[,"part"])), Week = ts(seq(1,352,1)))
fit.mlp.cmort = mlp(ts(cardiacTrain[,"cmort"], frequency = 52),reps = 50,
difforder = 0, comb = "median",xreg = cardiacDF_xregNW, allow.det.season =
TRUE, det.type = "bin")
fit.mlp.cmort
plot(fit.mlp.cmort)
fore.mlp.cmort = forecast(fit.mlp.cmort, h = 156, xreg = CMDF_foreNW)
plot(fore.mlp.cmort)
RMSE = sqrt(mean((cardiacTest[,"cmort"] - fore.mlp.cmort$mean)^2))
RMSE
#Plot
t=1:508
plot(t[348:508]seq(1,508,1), cardiac[,"cmort"], type = "l",xlim = c(0,510),
ylim = c(0,130), xlab = "Time", ylab = "Cardiac Mortality", main = "52 Week
Cardiac Mortality Forecast")
lines(seq(353,508,1), fore.mlp.cmort$mean, type = "l", col = "red")
#
# Code for Figure 11.37
#
#Ensemble
#VAR p = 7 non seasonal
CMortVAR7 = VAR(cardiacTrain, type = "both", p = 7) #p = 2 from SBC
preds7=predict(CMortVAR7,n.ahead=156)
RMSEVAR7 = sqrt(mean((cardiacTest[,"cmort"] - preds7$fcst$cmort[,1])^2))
RMSEVAR7
#VAR p = 2 seasonal
CMortVAR2S = VAR(cardiacTrain, season = 52, type = "both", p = 2) #p = 2 from
SBC
preds2S=predict(CMortVAR2S,n.ahead=156)
ensemble = (preds2S$fcst$cmort[,1] + fore.mlp.cmort$mean)/2
#Plot
plot(seq(1,508,1), cardiac[,"cmort"], type = "l",xlim = c(350,508), ylim =
c(70,110), xlab = "Time", ylab = "Cardiac Mortality", main = "52 Week Cardiac
Mortality Forecast From A VAR/MLP Ensemble")
lines(seq(353,508,1), ensemble, type = "l", lwd = 4, col = "green")
lines(seq(353,508,1),preds2S$fcst$cmort[,1] , type = "l", lwd = 2, lty = 2, col
= "red")
lines(seq(353,508,1),fore.mlp.cmort$mean , type = "l", lwd = 2, lty = 4, col =
"blue")
lines(seq(353,508,1),preds7$fcst$cmort[,1] , type = "l", lwd = 2, lty = 2, col
= "purple")
RMSEVAR7 = sqrt(mean((cardiacTest[,"cmort"] - preds7$fcst$cmort[,1])^2))
RMSEVAR7
RMSEVAR2S = sqrt(mean((cardiacTest[,"cmort"] - preds2S$fcst$cmort[,1])^2))
RMSEVAR2S
RMSEMLP = sqrt(mean((cardiacTest[,"cmort"] - fore.mlp.cmort$mean)^2))
RMSEMLP
RMSEENSEMBLE = sqrt(mean((cardiacTest[,"cmort"] - ensemble)^2))
RMSEENSEMBLE
```

CHAPTER 11 PROBLEMS

1. Compare the "best" ARMA or ARIMA model with the best MLP model for forecasting the next 12 observations of the global temperature data (**`global.temp`**).
2. Compare the "best" ARMA or ARIMA model with the best MLP model for forecasting the next 12 observations from the Ozona data (**`ozona`**).
3. Pick a stock and download the last three years of closing prices for this stock. Use this dataset to compare the "best" ARMA or ARIMA model with the best MLP model for forecasting the next day's stock price. "Best" here will simply mean whether your model correctly predicted the price to go up or down with respect to the previous day's close. Create a clear report or presentation that explains how you approached this problem as well as your results.
4. Fit an MLP with 1 input node, no hidden layers and 1 output node to the **`wtcrude2020`** data and forecast a horizon of 300. Match this result to that of this chapter. Next select two forecast origins below the mean and two that are above the mean and calculate and plot the forecasts for a horizon of 300. What do you notice?
5. Do your best to model the cardiac mortality (**`cmort`**) with a univariate MLP model. Compare this model with the multivariate model we fit in this chapter.
6. Note that the variable "time" was never input into the melanoma / sunspot models. This is because the trend behavior is addressed with the first difference of the response (**`melanoma`**). Fit an MLP model that does not take the first difference and explicitly addresses the possible trend in the melanoma count by adding a "year" variable to the MLP as an explanatory variable (in addition to the **`sunspot`** explanatory variable). Note that you will need to add this variable to the data frame (**`melanoma2.0`**). How do the results compare with the first difference approach performed in the text?
7. Note that the variable "time" was never input into the cardiac mortality models. This is because the trend behavior is addressed using the first difference of the response. Fit an MLP model that does not take the first difference and explicitly addresses the possible trend in the cardiac mortality by adding a "week" variable to the MLP as an explanatory variable. How do the results compare with the first difference approach performed in the text? Note, you will need to add this variable to the **`cardiac`** data frame.
8. With respect to the cardiac mortality study, repeat the short term (52-week) ensemble calculation and analysis using a different seed. Compare your results with the results from the text (**`set.seed(1)`**). Repeat this process several times using different seeds.
9. With respect to the cardiac mortality study, repeat the longer term (156-week) ensemble calculation and analysis using a different seed. Compare your results with the results from the text (**`set.seed(1)`**). Also, repeat this process several times using different seeds.
10. Assess the fit of the MLP on the **`AirPassengers`** data with and without the seasonal dummy variables. Make the initial assessment based on visualizations of the forecasts for the last three years versus the actual values. Also, compare the results using the RMSEs for forecasts of the last 36 months of the series.

Mini Research Project

There is some research that suggests that the *tanh* activation function is better in some sense that the *sigmoid* activation function.

https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d

Find out how to change the hidden layer activation functions from the default (sigmoid) to the tanh function (this may take some significant research into the **`mlp`** function and the function it calls). Then think of a way to evaluate whether it is truly "better". Create a report or presentation that explains your findings!