Time Series Modelling and Forecasting with Applications in R

Prof. Sudeep Bapat

Shailesh J. Mehta School of Management

Indian Institute of Technology Bombay

Department of Applied Statistics/Finance

Week 12

Lecture 59: Neural Networks for Time Series

Thank you. Hello all, welcome to this course on time series modeling and forecasting using R. Now, we are almost through with the entire course. We just have a couple more sessions to go. The broad idea about this week was machine learning aspects in time series. So, we started with a bit of an introduction in the first session. We gave a flavor of different ML techniques used in practice, along with some applications and integrations of each one of them with time series applications, examples, and so on. Then, in the second session this week, we talked about three ML techniques: regression, support vector machines, random forests, etc.

And in the very last session, we talked about an IDR called K-nearest neighbors, applications, limitations, and advantages of KNN, right? And right towards the end, I gave you a very short flavor of a classification ML model called Naive Bayes. So, now is the time to discuss briefly, yet in detail, about the IDR—or rather, the integration of neural networks and time series, okay? So, again, the heading has two important aspects built into it.

The first is neural networks, the second is time series, and then how you integrate the two and what exactly are some of the models pertaining to or having an integration of neural networks and time series, etc. So, again, I thought of putting some slides as introductory slides to describe generally about neural networks. So, if you are not very confident about what neural networks mean, right. These days, there is a huge hype about all these terminologies, right. Deep learning, neural networks, AI, ML, etc.

So, we will try to understand exactly what you mean by neural networks outside of a time series context, and then, of course, we will stitch the two things together, okay. So, as always, we'll start with a small introduction about the general idea of a neural network.

So, firstly, all neural networks are artificial. Artificial neural networks are nothing but forecasting methods based on simple mathematical models of the brain. Secondly, they allow for complex nonlinear relationships between the response variable and its predictors.

So, again, when it comes to neural networks, of course, they are capable of handling much more complex nonlinearity aspects or patterns compared to some traditional ML methods. Hence, all these neural network ideas fall under the umbrella of deep learning. So here, in front of you, you can see a small image where, let's say, you have the nucleus. This is an image of what goes inside a typical brain. So, let's say in the center you have the nucleus, and then you have all these dendrites, and this is the axon; these are boutons, etc.

But essentially, the idea is, let's say, how you pass information through all the neurons in the brain. So, from the nucleus, let's say if you generate some information, and then how the information is passed through all the layers in the brain and through all the neurons in the brain. So, if you try to mimic the exact same thing using some mathematical models, these are called neural networks. So, again, just to recap one more time, neural networks are nothing but some forecasting methods based on simple mathematical models of a typical brain. And secondly, they allow for complex nonlinear relationships between the response variable and its predictors.

So, sort of pick this image from this source and if you want some more information, then one can actually visit the source here. Alright. Okay, so secondly we will talk about what exactly goes inside the let us say neurons and how exactly is the information passed and then how can we connect the two things together. So, on one hand you have the architecture of the brain, on the other hand you have the actual mathematical model. So, how do you stitch the two things together and then how are kind of these two things sort of similar.

So, we will discuss all these things now. So, the second idea is what exactly is the architecture of the neural network. So, a neural network can be thought of as a network of neurons. So, let's say you have a strong network or a rather widespread network of neurons which are organized in multiple layers, right? So, let's say you have all these multiple layers and then the information has to be passed through all these layers, right?

And then imagine again that you see inside somebody's brain and then you're thinking that the information is getting passed through all the neurons originating from the Again,

if you go back a slide originating from the nucleus and let us say different sets of information are being passed through all these layers inside the brain until it reaches the output, okay. So, again a neural network can be thought of as a network of neurons which are organized in different layers. The predictors or inputs form the bottom layer and the forecasts or outputs form the top layer, okay. So, all these bottom layers which are the layer's

below the very last one. So, these are called as bottom layers. So, all the bottom layers form the structure of the predictors or inputs and the top layers. So, top layers are nothing but the last layers are nothing but the forecasts or outputs. There may also be some intermediate layers called as hidden neurons.

So, there may be some hidden layers like you have drawn here and all these layers may be hidden. So you essentially don't know that how many layers are there and then information is being passed through how many layers in between from the input layer and the output layer. So the very bottom layer is exactly where all the inputs are being passed, something like that. And this is the output layer. So the output is being pre-processed and then coming out of the output layer.

And even in between the input layer and the output layer, there could be multiple hidden layers or hidden neurons. So essentially, this entire exercise of this entire idea loosely mimic the way our brains solve the problem. Let us say by taking in inputs, processing them and generating an output. So the entire processing is done inside the hidden layers between the input and the output layer. So let us say imagine that the information is passing through the input layer, then the pre-processing or the processing is being done and the output is generated from the output layer.

Like us, they learn to recognize patterns, but they do this by training on labeled datasets. So, what exactly are the inputs here? Again, the inputs are some labeled datasets. So, they try to mimic the labeled datasets, gain some information from the labeled dataset, pre-process or rather process the labeled dataset using the hidden neurons or the hidden architecture, and then provide the output. So, hopefully, if somebody is not very confident about neural networks, one should again read up, let us say, a small elementary book on that or see some videos on that.

But broadly speaking, this is the idea that there are some mathematical models inside the hidden layers that are capable of processing the inputs and producing the outputs. So, a very famous neural network is called a perceptron. So, we will briefly discuss what a

perceptron is. So, the network of nodes sends signals in one direction, and hence it is called the feedforward network. So, what do you mean by feedforward?

So, the information only goes in one direction, from an input to all the hidden layers, and then to the output. Something like this is then called the feedforward network. And the figure below depicts a neuron connected with n other neurons and thus receives n inputs. Let us say x1, x2, x3, x4 up to xn. So, this entire architecture is, in fact, called the perceptron.

So, a perceptron is a very elementary kind of neural network which contains a feed-forward network where information is passed in only one direction. And let us say you have all these sets of inputs, let us say n inputs: x1, x2, x3 up to xn. And then there has to be some sort of weight function. So, again, do not worry. So, we have all these ingredients one by one.

So, we will talk about them one by one. So, let us say inputs, then you have some weight functions given by Ws. Then you take some sort of sum. Then you have some activation function and eventually the output. So, these are all the individual ingredients which sort of process the inputs and produce the output.

So, using the inputs, you bring in some weight functions, then you take the sum or rather the weighted sum, then you involve some activation function of some sort. And once you translate or transform using the activation function, then you are capable of producing the output. But since this is a very classical sort of neural network, it is called a perceptron. So, the entire architecture seen here is called a perceptron. Now, the second aspect, again, if you go back a slide, then after the input layer, you must implement some sort of weight functions to all the input layers.

So, the second aspect of the second ingredient is the weights. So, we start with an example. So, let us say you bike to work. So, you have to go to work using a bicycle, and you have two factors to make your decision to go to work. So, whether you should go to work or not is based on two factors.

The first factor is the weather. So, the weather must not be bad, of course. And the second factor is that it must be a weekday. So, you will not go to work on a weekend. So, you have to focus on, or rather control for, these two factors.

So, the weather factor and then the weekday factor. So, the inputs have to be binary. So, let us propose the conditions as yes-or-no questions. Again, in this case, we can easily do that. So, if the weather is fine, then 1 for yes, 0 for no.

And similarly, if it is a weekday, 1 for yes, 0 for no. So basically, binary coding is applied to each and every factor here. So, if it's good weather, then 1. If it's bad weather, then 0. Similarly, if it's a weekday, it's 1.

And if it's a weekend, it's 0. We cannot tell the neural network these conditions; it must learn them for itself. So what happens inside the neural network—we cannot specifically tell the neural network what condition it is. So based on some labeled inputs or a labeled dataset, the neural network has to understand by itself what weight functions one should choose. And essentially, the neural network does this with something called weights.

So again, just to repeat, in the context of this example, the neural network has to decide which factor, whether the first factor, which is weather, or the second factor, which is weekday, should be given more weight. And again, it depends on which factor is important. So if the weather is more important than the weekday, then the weight function for weather should be higher, and vice versa.

So if somebody thinks that being a weekday is more important than the weather, then it should give more weight to the weekday. So essentially, one has to balance the weights for each of the features or each of the inputs. So a higher weight means the neural network considers that input more important compared to the other inputs. And how does it do this? So, it will learn this by training on the data.

So, whatever labeled datasets or labeled features you are giving or inputting into the network, the neural network is capable of understanding the weight functions from the training data itself. Does this make sense so far? So, the second ingredient is weights. And then, how do you train the perceptron? So, training the perceptron.

So, input vectors from a training set are presented to the perceptron one after the other. And weights are modified according to the following equation. So, now we are slowly entering into some technicalities here. So again, we do not worry. So, we will not throw in a lot of notations here.

$$W(i) = W(i) + a * g'(\text{sum of inputs}) * (T - A) * P(i)$$

But just to understand the mathematical idea behind what goes inside training the perceptron, right? So, for all inputs I, this is my W I. So, W I depends on the current W I plus A into G prime of the sum of inputs. This is G prime of the sum of inputs into T minus A into P of I. Now, what do you mean by each and every terminology here? So, firstly, G prime.

So, G prime is nothing but the derivative of the activation function. Now, again, we will soon see what you mean by activation function. This is one ingredient which is still remaining. A is called the learning rate. W is a weight vector.

We have seen in the previous slide what you mean by W. So, W is a weight vector. P is the input vector. So, whatever features you are throwing inside the network. Capital T is the correct output that the perceptron should have known, and capital A is the output given by the perceptron. So, imagine that you are training the perceptron by some features, by some inputs, and the perceptron is throwing some output, right?

So capital T is the output that is expected. Capital A is the output which is given by the perceptron. So this sort of tells us the error, right? So how much is the difference between the expected output and the given output? So T minus A. And then P is nothing but the input vector.

So you multiply by the input features, T minus A. And G prime is nothing but the derivative of the activation function. and then plus the weight function. So this is essentially the mathematical model which goes inside training the perceptron. So this is more like a training or rather going back and forth sort of an exercise. So you keep on training the perceptron, trying to improve the accuracy, again go back, try to correct for the errors, again train the perceptron, until the final output has a very high accuracy.

So, in general, the entire idea about any ML technique or deep learning technique, neural network technique, it considers lot of training which goes inside the particular model. So, you have to correct the errors, you have to go back and forth, you try to correct the errors, again retrain, get the output, find the accuracy. If you still think that the accuracy is not good enough, then you again go back, again reweigh the options, try to correct for all the parameters there, again retrain, again get the output and so on. So, it is sort of an iterative procedure that goes inside any of the model there.

So, hopefully, the idea about training the perceptron is clear—what are the ingredients that go into training any perceptron. Now, the third ingredient and the most important one

is the activation function. So, what do you mean by an activation function? So, activation functions are nothing but mathematical equations or formulas that determine the output of a neural network layer given its inputs. So, an activation function is nothing but some function that transforms the inputs or the mathematics behind the inputs and then gives some desired output.

They introduce some non-linearity to the model, enabling it to learn complex patterns and relationships in the dataset. And again, don't worry—we'll discuss a couple of examples of widely applicable activation functions. The first one is called the sigmoid activation function. The sigmoid activation function is useful for binary classification problems. And what exactly is a sigmoid function?

$$f(x) = \frac{1}{1 + e^{-x}}$$

So, this is the sigmoid function. It's 1 divided by 1 plus e to the power of minus x. Here, x stands for the input. So, whatever input or feature you have, or the quantity of the input, you basically replace that value here. You find out the activation function, which is f of x, and then you approach the output.

But why do you require the activation function? You require the activation function to translate all the inputs into desired outputs. Second, the most important or widely applicable activation function is the tanh activation function. So, in the hidden layer where the output needs to be zero-centered, this is the activation function. So, f of x is e to the x minus e to the minus x divided by e to the x plus e to the minus x. So, these two are somewhat similar if you find resemblance between a sigmoid function and a tanh function.

So, I will say a tanh function is nothing but a slightly varied version of the sigmoid function. Now, what would happen if you do not have any hidden layers in the network? Then what? So, if there are no hidden layers in the model, the model reduces to what? So, can you try to answer this question?

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

So, let us say if this is the architecture. So, you have four inputs. So, the input layer and you do not have any hidden layers. You only have an output layer. So, based on some sets of inputs, if you are producing one single output, then the model reduces to

And the answer is regression. Okay. So, if you do not have any hidden layers, the model essentially reduces to a regression sort of idea. Okay. Because again, you can think that all these are individual predictors.

So, x1, x2, x3, x4, then you have some sort of mathematical model that goes inside this, which connects all the inputs together and tries to estimate the betas and throws one single output. So, since you do not have any hidden layers in between, the problem reduces to translating the inputs that you have, which are nothing but predictors, and then constructing that model and throwing the single output. So, the answer here is nothing but regression. So, if you do not have any hidden layers, then a neural network reduces to a regression problem. And then, a similar extension of that is what would happen if you bring in a hidden layer in between.

So, this is exactly how hidden layers are visualized. So, you have this input layer here. Now again, assuming that you have four inputs—1, 2, 3, 4—and then, before you throw an output, you have this hidden layer in between. And remember one thing: these are called nodes, right? So, each circle in any of the layers is called a node, by the way.

So, the hidden layer here contains three nodes. And if you see in this architecture, each node in the hidden layer receives inputs from all four inputs. So, can you see that? So, you have arrows from all four inputs pointing toward any node in the hidden layer. And this is a typical extension of what we saw earlier, a slide back, where you didn't have any hidden layers.

So, once we add a hidden layer or intermediate layer with hidden neurons, the neural network becomes nonlinear, of course. And why is it called a hidden layer? Because you don't know what exactly is going on inside this layer. You just assume that there is some hidden layer; you throw in the inputs, which is the input layer. There is some complex nonlinear mathematical model which goes inside this hidden layer and is capable of producing an output, basically.

And such an extension or such an architecture is called a multi-layer feed-forward network. So, obviously, why multi-layer? Because it contains more than one layer. So, it contains several hidden layers as well. Now, remember one thing: there need not be a single set of hidden layers; there could be multiple also.

So, for example, there could be more hidden layers here. So, there could be hidden layer 1, hidden layer 2, hidden layer 3, etc. So, the more hidden layers there are, the more

complicated the entire neural network becomes. But the more important aspect here is that each layer of nodes receives inputs from the previous layers. And the outputs of the nodes in one layer are nothing but inputs to the next layer.

So, can you see that? So, outputs coming from the input layer are nothing but inputs to the hidden layer. And again, the outputs from the hidden layer go to the output layer and produce a single output. So, the inputs to each node are combined using a weighted linear combination. So, there has to be some mathematical formula or technique to combine the information coming from, let us say, the input layer or the hidden layer, etc.

So hopefully, the general architecture should be clear as to what you mean by layers, what you mean by hidden layers, how the information passes through the layers, etc. Now we will talk about one very famous model, and this is where we integrate neural networks and auto-regression or time series. So, what would happen if you have time series data? With time series data, lagged values of the time series can be used as inputs to your neural network. So, let us say if you want to predict y_t, then you have easy access to, let us say, its own lag values.

So, y t minus 1, y t minus 2, y t minus 3, etcetera. So, why not use all the lag values as inputs to your neural network, just as we use lag values in a linear auto-regression model? So, when you create, let us say, an AR model, even there you have, let us say, all these along with the phi coefficients and a random error, right? So, remember the AR model structure. Similarly, here the idea is that we use all the lag values as inputs to the in-between layers of a neural network.

So, hence we call this neural network autoregression, or the NNAR model. So, NNAR stands for neural network autoregression model. And here, assuming—so, we are assuming a very simple case—that we only consider a feed-forward network. So, what do you mean by that? So, information passes in one direction only, and assuming it only has one hidden layer.

So, there is an input layer, one hidden layer, and an output layer. And we use the notation NNAR p comma k to indicate that there are p lag inputs. So, p stands for you having p lag inputs. So, y t minus 1, y t minus 2, up to y t minus p, let us say, in general, and k nodes in the hidden layer. So, the second argument here stands for how many nodes are there in the hidden layer.

So, for example, can you answer this? So, what do you think this stands for? So, NNAR 9 comma 5. So, in general, an NNAR 9 comma 5 model is nothing but a neural network with the last 9 observations as user inputs for forecasting the output y t. It has 5 neurons in the hidden layer.

So, let us say you have all these input layers, and the hidden layer only has 5 neurons. So, 1, 2, 3, 4, 5, and then you have the output layer. So, information is passed from the input layer to all the—this is the architecture you have—and again from the hidden layer to the output layer, which produces the output. So, in the same way, can you try to answer this question now? So, an NNAR P comma 0 model,

is equivalent to which model? So, what do you think? So, again, NNAR P comma 0 means you are using P lag values as inputs, and you do not have any hidden layer here because the second argument is 0. So, you have 0 nodes in the hidden layer. So, essentially, this model is nothing but an ARMA P comma 0.

So, essentially this model is nothing but ARMA P comma 0 or rather ARIMA P comma 0 comma 0. So, an NNAR p comma 0 model reduces to nothing but ARMA p comma 0 model. So, hopefully I have tried to explain a very elementary sort of a connection between neural network and time series by bringing the idea of a NNAR p comma k model. Now, there is one more ingredient rather one important aspect which has to be controlled when it comes to neural networks which is called as batch iterations and epoch. So, what do you mean by that?

So, what do you mean by batch? What are iterations? What do you mean by epoch? So, we will try to understand this one by one. So, what happens in let us say if you have a very big data set, one cannot pass the entire data set in a neural network at once.

So, what can we do? So, we have to divide the entire data set into small parts which are called as batches. So, we cannot pass the entire data set into a neural network at once. So, we have to divide the data set into number of batches or smaller sets or parts. Hence, the batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters.

So, batch size. So, how many observations in one batch? How many batches should there be? So, on is a hyperparameter that have to be controlled before updating the internal model parameters. And then what do you mean by iterations?

So, iterations is nothing but the number of batches needed to complete one full cycle. And then one full cycle is called as one epoch. So, essentially one epoch is when an entire data set is passed forward and backward through the neural network only once. So, once you exhaust all the observations that are there in the entire data set, then that is what is a one epoch. And usually what people do is they have they sort of pass the entire data set number of times.

So they have number of different epochs or let us say 100 epochs or 500 epochs and why exactly. So let us say you run the entire data set of course by dividing them into batches and form all the epochs and you run the entire neural network exercise 100 times or 500 times or 1000 times and then you take the average. And why exactly? Because doing the entire exercise just once sort of cannot give a very clear picture, right? I mean, since you do not know what is going inside the neural network, it is completely black box.

So, it is obviously a very good idea to sort of repeat the entire exercise number of times and take the average. So, let us say if you repeat the entire exercise 100 times then you have 100 epochs or if you repeat the entire exercise let us say 500 times it is called as 500 epochs and so on. So, again this is a very important idea when it comes to neural network that once you divide the data set into number of smaller parts called as batches and you run the batches and once you exhaust all the batches or the entire data set it is called as one epoch. Now, of course, use of more than one epoch, why exactly? So, since we are using a limited data set and to optimize the learning, we are using a gradient descent which is an iterative process.

So, updating the weights with single pass or one epoch is not enough. And then obviously, one epoch leads to underfitting. So, if you only use one epoch or pass the entire set of observations only once, then it is a rather underfitting. As the number of epochs increases, a greater number of times the weight are changed in the neural network and the curve goes from underfitting to optimal to overfitting. So, again the number of EPOS that one should run the entire network for is again a hyper parameter which has to be balanced somewhere.

So, one should not again go too overboard. So, one should balance or strike a balance between underfitting and overfitting. Now, towards the end of this session we will see a very briefly a practical application in R and of course the next session is entirely practical. So we have in front of you, we have USD INR exchange rate from let us say 3rd April 23 to 29th March of 24. And this is exactly how the exchange rate begins.

So, on April—or rather, 3rd April 2023—the exchange rate was close to 83.5, and then it sort of dropped down to 82 and so on and so forth. So, the USD-INR exchange rate over this timeline. And now we will see if we can fit some sort of an NNAR model on that. So, first, the ACF plot. So, the ACF plot is rather non-stationary.

So, what should be done? So, one should either transform, take the difference, and so on. So, since all these spikes are significant and away from the bounds, we can safely conclude that this series is highly non-stationary. And, of course, one can use the augmented Dickey-Fuller test to test that. So, again, all these have been covered well back in the course.

So, again, if you are not very confident, you should go back and try to revise. So, here, if you see the p-value is more than 0.05, hence we cannot reject the null. And the null hypothesis is that the series is not stationary. So, what do you do? I can difference the plot once.

So, the time series plot of the exchange rate difference then and now essentially shows that the stationarity has improved from the earlier plot. So, again, if you run the ADF test on the difference plot now, the p-value is within the range. So, 0.01 is less than 0.05, let us say. So, one can actually reject the null and conclude that the series is stationary. So, now then what?

So, then we can implement, or rather, put forward an NNAR model with one hidden layer. So, let us say you have one hidden layer in between, and which model are we training the dataset on? So, we will train the dataset on the NNAR 1,1 model. So, again, what is my 1,1 model?

We will use only one past value as input. So, y t minus 1 only, and in the hidden layer, there will only be one So, let us say if you have the input layer, then you have the hidden layer having one node, and then the output layer, something like that. And then you take the average of 20 such networks. So, we are taking 20 epochs here.

So, this number means 20 epochs. So, you run the entire exercise 20 times, take the average of that, each of which is a 1-1-1 network as specified here with 4 weight options. So, one can choose the weight options, and this entire exercise has been done in R, by the way. And here, you can see that it throws you some estimates. So, sigma square is estimated as 0.01438, etc.

And these are the forecasts along with the accuracy measures. So, here you can see that the forecasts are not very good, right? So, by the way, the forecast is given by this horizontal blue line, which is obviously not matching the pattern of the exchange rate. So, we can improve.

But I have only included this exercise just to make you all understand how to implement the NNAR in R exactly on a practical dataset. And along with here, we can see, let us say, how much is the root mean square error, or how much is the mean square error, MAE, all those values, okay? So, hopefully, the idea of neural networks is clear, and the integration of neural networks to time series is clear, and then we discussed one very important model called NNAR, and then the idea about, let us say, how to implement it on a practical application in R is clear. Thank you.