COMPUTATIONAL ECONOMICS

Neural Nets in Excel

David A. Kendrick
Ruben P. Mercado
Hans M. Amman

1 Introduction

Much of economics is about finding optimal variables given parameters that describe human behavior. For example, in the optimal growth model that we solved with Excel in Chapter **??**, page **??** the goal was to find the optimal levels of the consumption and capital stock variables given the parameters of the production and utility functions.

In this chapter we invert this duality. We begin with the observed behavior and attempt to find the parameters that permit the specified relationships to fit the data most closely. Such is the subject matter of econometrics and estimation. However, we are looking at a type of estimation that until recently had not been in the mainstream of econometrics but rather developed in other fields and is now increasingly being used to fit economic relationships—namely neural nets.

Neural network models are suitable for dealing with problems in which the relationships among the variables are not well known. Examples are problems in which information is incomplete or output results are only approximations, as compared to more structured problems handled, for example, with equation-based models. Neural networks are particularly useful for dealing with data sets whose underlying nonlinearities are not known in advance.[1] Among the many possible applications are forecasting and identification of clusters of data attributes.

The example we use here is typical of the applications of neural nets to economics and finance—how best to predict the future prices of a stock.[2] The stock we use is that of the Ford Motor Company, and we attempt to predict its future share price by using the share price of a group of related companies—companies that provide inputs to automobile production and companies that produce competing vehicles.

The central notion of neural net analysis is that we can use a set of observations from the past to predict future relationships. Thus we use the closing price of Ford stock each week over a 14-week period to *train* the model and then use the parameters that emerge from the training to predict the Ford stock price in the 15th and 16th week. This is done in an Excel spreadsheet using the Solver

[1] One of the strengths of neural net methods is that they can approximate any functional shape.

[2] Neural nets are not necessarily a better way to predict stock prices than standard econometric methods; however, stock prices offer a clear and motivating example for many students, so we use that example here.

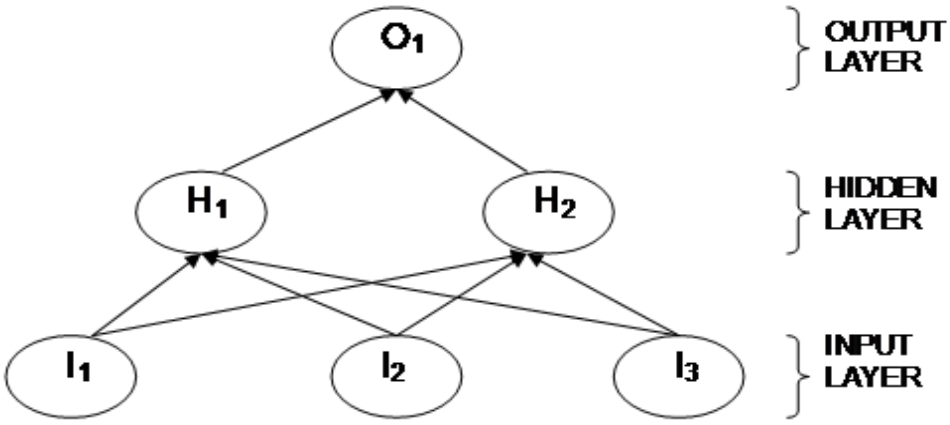
that we first used in the growth model.

The chapter begins with an introduction to neural nets followed by the specification of an automobile stock price model. We then introduce the data that are used in the model, the representation of the model in Excel, and the use of the Excel Solver to find the best parameter values.

2 Neural Net Models

Neural networks (or, more properly, artificial neural networks) are inspired by, or up to a point are analogous to, natural neural networks. They have three basic components: processing elements (called nodes or neurons), an interconnection topology, and a learning scheme. From a computational point of view, a neural network is a parallel distributed processing system. It processes input data through multiple parallel processing elements, which do not store any data or decision results as is done in standard computing. As successive sets of input data are processed, the network processing functions *learn* or *adapt*, assuming specific patterns that reflect the nature of those inputs.

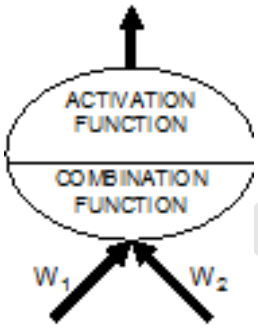
Figure 1: Layers in the neural net



There are many alternative network architectures. Let us look now in more detail at the elements, architecture, and workings of a neural network as shown in Figure 1. This is known as back-propagation or as a feed-forward model, which is the type most commonly used. This is a simple network with one input layer with three neurons, one intermediate layer with two neurons (usually named the *hidden layer*), and one output layer with just one neuron. A key component of the network is the neuron, an elementary processing unit that,

given inputs, generates output. It is composed of two main parts: a combination function and an activation function (Figure 2). The combination function computes the net input to the neuron, usually as a weighted sum of the inputs. The activation function generates output given the net input.

Figure 2: Combination and activation



It is standard procedure to constrain the output of a neuron to be within an interval *(0,1)*. To do so, different functional forms can be used for the activation function, such as logistic functions, sigmoid functions, and so on. Furthermore, a threshold may be used to determine when the neuron will *fire* an output as the activation function yields a value above that threshold. Input layer neurons receive data (*signals*) from outside and in general transmit them to the next layer without processing them. Output layer neurons return data to the outside and are sometimes set to apply their combination functions only.

The learning process of the network consists of choosing values of the weights so as to achieve a desired mapping from inputs to outputs. This is done by feeding the network with a set of inputs, comparing the output (or outputs, in case of having more than one network output) to a known target, computing the corresponding error, and sometimes applying an error function. Then weights are modified to improve the performance. To do this, a variety of methods can be employed, such as Newton or conjugate gradient in Excel, which are discussed later in this chapter.

3 The Automobile Stock Market Model

We begin with the specification of the combination function for the output layer as

$$y_t = \theta_0 + \sum_{j=1}^{q} \theta_j a_{tj} \tag{1}$$

where y_t is the output in period t, a_{tj} is the hidden node value in period t for node j and the θ_j's are parameters. There are q hidden nodes. In our model the y_t variables are the share price of the Ford Motor Company stock in each of the 14 weeks in 1997.

The θ's are among the parameters that we are seeking. Thea_{tj}, which are the values in time period t at hidden node$j$, are given by the expression

$$a_{tj} = S\left(\sum_{i=1}^{q_j} w_{ji} x_{it}\right) \tag{2}$$

where the x_{it} are the inputs at node i in period t. There are q_j inputs at hidden node j. The x_{it} are the share prices of the other companies in our example. The w_{ji} are the parameters at the jth hidden node for the ith input and are the second set of parameters that we want to choose. Thus, in summary, we are given the share prices of the other companies x_{it} and the share price of the Ford stock y_t and are seeking the parameters θ and w that permit our functions to fit the data most closely.

What functions are being used? The first function, in equation (1), is linear and the second function, in equation (1), S, is a sigmoid function, with the mathematical form

$$S(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

One can quickly see by examination that this function evaluated at $z = 0$ is

$$S(0) = \frac{1}{1 + e^{-0}} = \frac{1}{1 + 1} = \frac{1}{2} \tag{4}$$

and that large negative values of z map to near zero, that is,

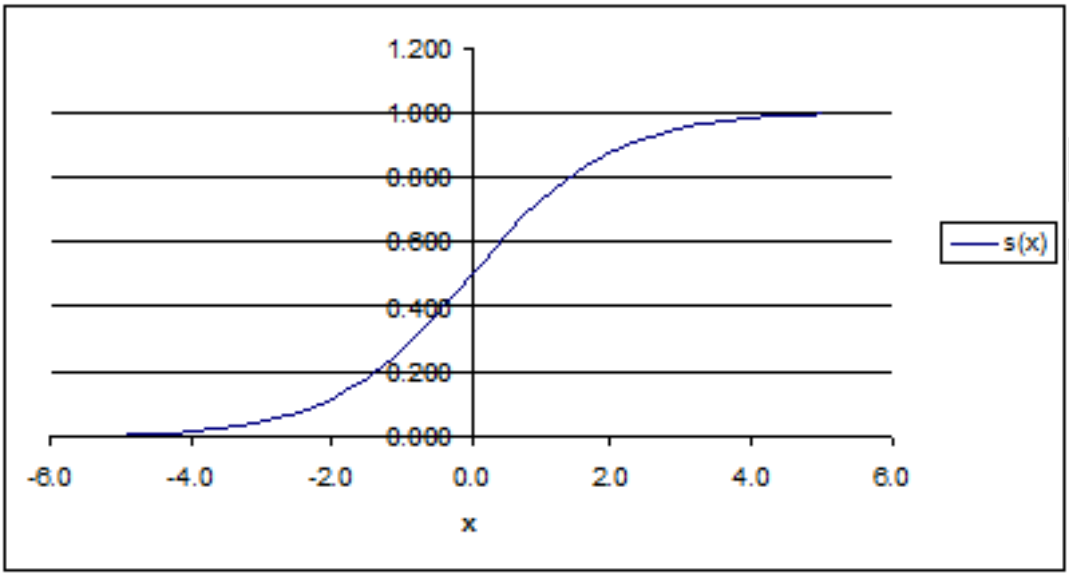
$$S(-5) = \frac{1}{1 + e^{5}} = 0.007 \tag{5}$$

and that large positive values of z map to approximately one, that is,

$$S(5) = \frac{1}{1 + e^{-5}} = 0.993 \tag{6}$$

Thus the function has the shape shown in Figure 3. It is sometimes called the *squasher* and it is quickly apparent why. Given any data set of numbers that range from very large negative numbers to very large positive numbers this function maps those numbers to the zero-to-one interval while maintaining their relative sizes.

Figure 3: Sigmoid function



The example we present here, which was developed by Joe Breedlove. This example contains share prices from the automotive suppliers of Ford in 1997, that is, Bethlehem Steel

- Owen's Glass
- Goodyear Tire and Rubber

and the competing auto makers to Ford, that is,

- Chrysler
- General Motors

to predict the share price of the

- Ford Motor Company

At that time stock prices were quoted as fractions rather than as decimals and the data in the spreadsheet reflect this fact. Ford's suppliers and competitors have changed since 1997, but the example is useful as a starting place for learning about neural nets.

The effect from the suppliers is aggregated into one hidden node and that from the competitors is aggregated into the second hidden node, as shown in Figure 1. So for the example at hand,

$$z_1 = w_{11} * x_1 + w_{12} * x_2 + w_{13} * x_3 \tag{7}$$

and

$$a_{t1} = \frac{1}{1 + e^{-(w11*x1+w12*x2+w13*x3)}} \tag{8}$$

$$z_2 = w_{21} * x_4 + w_{22} * x_5 \tag{9}$$

and

$$a_{t2} = \frac{1}{1 + e^{-(w21*x4+w22*x5)}} \tag{10}$$

In addition,

$$\hat{y}_t = \theta_0 + \theta_1 a_{t1} + \theta_2 a_{t2} \tag{11}$$

Thus the optimization problem in Excel is to find the values of

$$w_{11}, w_{12}, w_{13}, w_{21}, w_{22}, \theta_0, \theta_1, \theta_2 \tag{12}$$

that minimize the square of the separation between the predicted and actual values of the y's, that is,

$$Norm = \sum_{t=1}^{n} (y_t - \hat{y}_t)^2 \tag{13}$$

where n is the number of observations, which for this example is fourteen.

4 The Data

Closing stock prices for Ford and for the three suppliers (Bethlehem, Owen, and Goodyear) and the two competitors (Chrysler and General Motors) for each week in January, February, and March of 1997 were used as shown in Table 4.

As mentioned earlier, at that time stock prices were listed as fractional numbers, rather than as decimal numbers, as is now the case.

Table 1: Share Prices of Ford and Related Companies

Week	Ford	Bethlehem	Owen	Goodyear	Chrysler	GM
Closing	y	x1	x2	x3	x4	x5
Jan 3	$32^{1}/_{2}$	$9^{1}/_{4}$	$42^{1}/_{2}$	$52^{3}/_{8}$	$34^{5}/_{8}$	$57^{7}/_{8}$
Jan 10	$33^{1}/_{2}$	$8^{7}/_{8}$	49	$54^{1}/_{2}$	$35^{3}/_{4}$	$61^{1}/_{8}$
Jan 17	33	9	$48^{5}/_{8}$	55	$34^{3}/_{8}$	$60^{1}/_{8}$
Jan 24	$33^{5}/_{8}$	$8^{5}/_{8}$	$45^{5}/_{8}$	$54^{1}/_{4}$	$35^{1}/_{4}$	$62^{1}/_{2}$
Jan 31	$32^{1}/_{8}$	$8^{3}/_{8}$	$46^{5}/_{8}$	$54^{1}/_{2}$	$34^{7}/_{8}$	59
Feb 7	$32^{1}/_{4}$	$8^{1}/_{4}$	$45^{1}/_{2}$	$52^{1}/_{2}$	$34^{1}/_{8}$	$56^{3}/_{4}$
Feb 14	$32^{3}/_{4}$	$7^{3}/_{4}$	$44^{3}/_{4}$	$53^{5}/_{8}$	$34^{1}/_{2}$	$58^{3}/_{4}$
Feb 21	$33^{1}/_{8}$	$7^{7}/_{8}$	$43^{3}/_{8}$	$53^{3}/_{4}$	$35^{1}/_{8}$	$58^{1}/_{2}$
Feb 28	$32^{7}/_{8}$	$8^{1}/_{4}$	$42^{3}/_{8}$	$52^{3}/_{4}$	34	$57^{7}/_{8}$
Mar 7	$32^{1}/_{4}$	$8^{1}/_{8}$	$42^{5}/_{8}$	$53^{3}/_{8}$	$31^{7}/_{8}$	$56^{5}/_{8}$
Mar 14	$32^{1}/_{8}$	$8^{1}/_{2}$	$42^{1}/_{2}$	$53^{7}/_{8}$	$30^{1}/_{2}$	58
Mar 21	$31^{3}/_{4}$	$8^{1}/_{4}$	$40^{7}/_{8}$	$54^{1}/_{2}$	$30^{1}/_{4}$	57
Mar 27	$30^{7}/_{8}$	$8^{1}/_{2}$	$40^{1}/_{8}$	$54^{1}/_{4}$	$30^{1}/_{4}$	$56^{1}/_{4}$
Mar 31	$31^{3}/_{8}$	$8^{1}/_{4}$	$40\ ^{1}/_{4}$	$52^{3}/_{8}$	30	$55^{3}/_{8}$

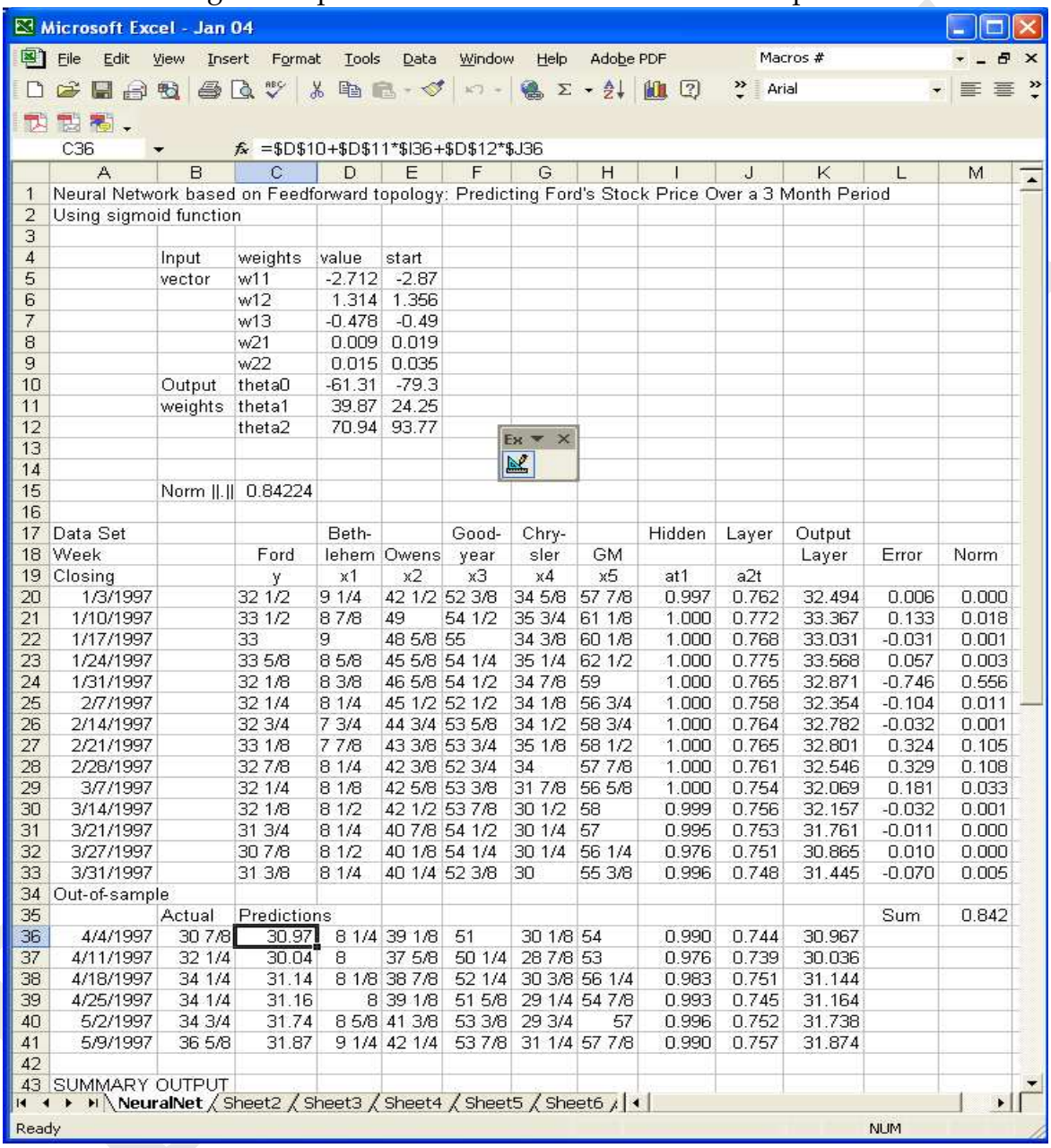
5 The Model Representation in Excel

Here we follow the representation of a neural net in Excel developed by Hans Amman and combine this with Joe Breedlove's model of Ford share prices. The input file for Excel for this example can be obtained from the book web site. Once you have downloaded the file you can begin by opening it in Excel as is shown in Figure 5.

Skip down to the section on the data set beginning in line 17 and note that there are fourteen observations consisting of the weekly closing share price y for Ford shares and the five inputs `x1` through `x5` for the other stocks. These observations are aggregated using the sigmoid function into the hidden layers `at1` and `at2` using a formula like

```
at1 = 1 / (1 + Exp(-(D20*D5 + E20*D6 + F20*D7)))
```

Figure 4: Spreadsheet for neural nets with stock prices



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Neural Network based on Feedforward topology: Predicting Ford's Stock Price Over a 3 Month Period												
2	Using sigmoid function												
3													
4		Input	weights	value	start								
5		vector	w11	-2.712	-2.87								
6			w12	1.314	1.356								
7			w13	-0.478	-0.49								
8			w21	0.009	0.019								
9			w22	0.015	0.035								
10		Output	theta0	-61.31	-79.3								
11		weights	theta1	39.87	24.25								
12			theta2	70.94	93.77								
13													
14													
15		Norm ||.||	0.84224										
16													
17	Data Set			Beth-		Good-	Chry-		Hidden	Layer	Output		
18	Week		Ford	lehem	Owens	year	sler	GM			Layer	Error	Norm
19	Closing		y	x1	x2	x3	x4	x5	at1	a2t			
20	1/3/1997		32 1/2	9 1/4	42 1/2	52 3/8	34 5/8	57 7/8	0.997	0.762	32.494	0.006	0.000
21	1/10/1997		33 1/2	8 7/8	49	54 1/2	35 3/4	61 1/8	1.000	0.772	33.367	0.133	0.018
22	1/17/1997		33	9	48 5/8	55	34 3/8	60 1/8	1.000	0.768	33.031	-0.031	0.001
23	1/24/1997		33 5/8	8 5/8	45 5/8	54 1/4	35 1/4	62 1/2	1.000	0.775	33.568	0.057	0.003
24	1/31/1997		32 1/8	8 3/8	46 5/8	54 1/2	34 7/8	59	1.000	0.765	32.871	-0.746	0.556
25	2/7/1997		32 1/4	8 1/4	45 1/2	52 1/2	34 1/8	56 3/4	1.000	0.758	32.354	-0.104	0.011
26	2/14/1997		32 3/4	7 3/4	44 3/4	53 5/8	34 1/2	58 3/4	1.000	0.764	32.782	-0.032	0.001
27	2/21/1997		33 1/8	7 7/8	43 3/8	53 3/4	35 1/8	58 1/2	1.000	0.765	32.801	0.324	0.105
28	2/28/1997		32 7/8	8 1/4	42 3/8	52 3/4	34	57 7/8	1.000	0.761	32.546	0.329	0.108
29	3/7/1997		32 1/4	8 1/8	42 5/8	53 3/8	31 7/8	56 5/8	1.000	0.754	32.069	0.181	0.033
30	3/14/1997		32 1/8	8 1/2	42 1/2	53 7/8	30 1/2	58	0.999	0.756	32.157	-0.032	0.001
31	3/21/1997		31 3/4	8 1/4	40 7/8	54 1/2	30 1/4	57	0.995	0.753	31.761	-0.011	0.000
32	3/27/1997		30 7/8	8 1/2	40 1/8	54 1/4	30 1/4	56 1/4	0.976	0.751	30.865	0.010	0.000
33	3/31/1997		31 3/8	8 1/4	40 1/4	52 3/8	30	55 3/8	0.996	0.748	31.445	-0.070	0.005
34	Out-of-sample												
35		Actual	Predictions									Sum	0.842
36	4/4/1997	30 7/8	30.97	8 1/4	39 1/8	51	30 1/8	54	0.990	0.744	30.967		
37	4/11/1997	32 1/4	30.04	8	37 5/8	50 1/4	28 7/8	53	0.976	0.739	30.036		
38	4/18/1997	34 1/4	31.14	8 1/8	38 7/8	52 1/4	30 3/8	56 1/4	0.983	0.751	31.144		
39	4/25/1997	34 1/4	31.16	8	39 1/8	51 5/8	29 1/4	54 7/8	0.993	0.745	31.164		
40	5/2/1997	34 3/4	31.74	8 5/8	41 3/8	53 3/8	29 3/4	57	0.996	0.752	31.738		
41	5/9/1997	36 5/8	31.87	9 1/4	42 1/4	53 7/8	31 1/4	57 7/8	0.990	0.757	31.874		
42													
43	SUMMARY OUTPUT												

where the D5, D6, and D7 are weights that are to be solved for and the D20, E20, and F20 are the observations x1, x2, and x3. You can see this formula in the spreadsheet by selecting the I20 cell and then looking at the expression in the formula bar at the top of the spreadsheet. Alternatively, you can see all of the formulas in the spreadsheet by selecting Tools:Options:Views and then checking the Formula box.

Now back to the Data Set section of the spreadsheet. Check the column at2 and you will find that it is similar to the column at1 except that it uses input data for x4 and x5 to compute the second of the two hidden layer values.

Consider next the Output Layer column, which is computed using an expression of the form

```
Output = theta0 + theta1 * at1 + theta2 * at2
```

where the thetas are weights that are computed in the optimization and are shown in the section on Output weights near the top of the spreadsheet.

Next look at the Error column in the Data Set section of the spreadsheet. This column is simply the difference

```
Error = y -- Output Layer
```

and the Norm column is the square of the elements in the Error column. The elements in the Norm column are summed up in cell M35 at the bottom of the column.

Now we are ready for the optimization problem. It is seen by selecting Tools:Solve at which point the dialog box shown in Figure 6 should appear.[3] This dialog box indicates that the optimization problem is to minimize the value in cell C15 (which on inspection is set equal to M35, which in turn is the sum of the elements in the Norm column).

As was discussed earlier, the Excel Solver uses nonlinear optimization methods (Newton method or conjugate gradient method—see Appendix B). The optimization is done by changing the elements in cells D5 through D12 until the minimum of the function is obtained. These cells are shown in Table 5 below beginning with the number *–2.712* and going down the value column to the element *70.94*.

The column to the right labeled start shows the numbers that were originally used when searching for the optimal parameters. They are not used in

[3] In case the dialog box does not appear, see Appendix A.

Figure 5: The Solver dialog box.

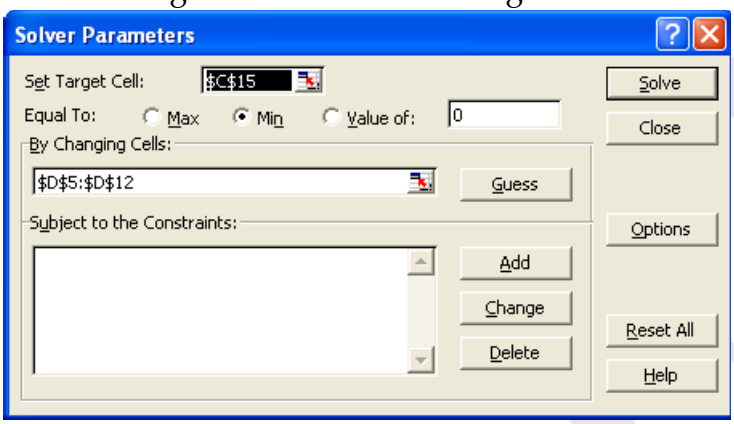


Table 2: Parameters

Input	weights	value	start
vector	w11	–2.712	–2.87
	w12	1.314	1.356
	w13	–0.478	–0.49
	w21	0.009	0.019
	w22	0.015	0.035
output	theta0	–61.31	–79.3
weights	theta1	39.87	24.25
	theta2	70.94	93.77

the present calculations but are stored there only to indicate which starting values were used. In fact each time the model is solved the numbers in the value column are used as the starting point and an effort is made to find values that decrease the norm. For a first experiment you might try changing some of the elements in the value column, selecting `Tools:Solver` and then clicking on the Solve button to solve the optimization problem and see if the parameters return to the original values or converge to some others that have either a smaller or a larger norm.

A point of caution: at times the solution procedure converges to a result with a higher norm because neural net estimation problems are sometimes characterized by nonconvexities and may have local optimal solutions that are not the same as the global optimal solution. Sometimes the number of local solutions may be very large. Thus in Excel it may be advisable to use a number of different starting values in order to check for global convergence. When there are many local optima, global optimization algorithms such as genetic algorithms can be used to perform global exploration of the solution space [see Chapters 11 and 12 or Goldberg (1989)].

In addition, you can experiment by changing some data elements in the y and x columns either in an arbitrary manner or by looking up the share prices for these companies in another time period and seeing whether the parameter values have remained the same.

Finally the spread sheet contains some forecast in the section called Predictions. These predictions are made for 6 weeks after the last week for which data were collected to *fit* or *train* the model. Look at the formulas for cells `B36` and `C36` that are shown in Table 5.

Table 3: Predictions

Out-of-sample		
	Actual	Predictions
4/4/1997	30 7/8	30.97
4/11/1997	32 1/4	30.04
4/18/1997	34 1/4	31.14
4/25/1997	34 1/4	31.16
5/2/1997	34 3/4	31.74
5/9/1997	36 5/8	31.87

If you select the cell just beneath the Predictions label you see that the predictions use an expression such as

```
= D10 + D11*I36 + D12*J36
```

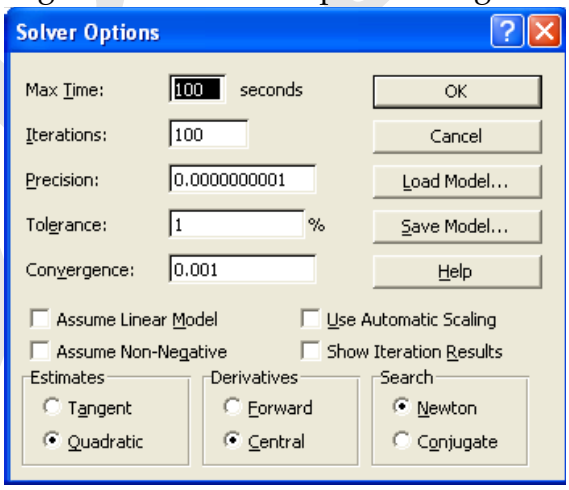
that translates to

```
Output = theta0 + theta1 * at1 + theta2 * at2
```

Note in particular that these predictions are done from *out of sample* data, that is, the data that are used to fit the model are not used to make the predictions. Rather some elements of the sample are reserved to test the model after it is fitted to a subset of the data.

There is one other topic that needs to be mentioned about the Excel Solver. Select `Tools:Solver:Options` and the dialog box shown in Figure reffig305 appears. You can use this dialog box to control the number of iterations that the Solver will use in trying to achieve convergence. Keep the number of iterations low when you are first working with a new data set and then if convergence is not being achieved raise this number as necessary. Moreover, a convergence value of *0.001* is probably close enough for most of the work you do, but you may require a looser convergence by lowering this setting to *0.01* in order to obtain convergence in *100* iterations. On the other hand, you may want to keep the convergence value at *0.001* and increase the number of iterations.

Figure 6: The Solver option dialog box.



Probably the most important element in the Solver options dialog box is Use Automatic Scaling. In many neural net data sets the various series may be of

very different magnitudes. For example, you might have an unemployment series with numbers of the size of *0.04* and a consumption series with numbers like *625*. In such a case it is wise to check the automatic scaling option. If you do this, the Solver will automatically scale all of your series so that they are of roughly the same magnitude and thereby increase the probability that the Solver will be able to find an optimal set of parameter estimates.

6 Experiments

There are two kinds of experiments that come to mind with this spreadsheet. As discussed earlier, at the simplest level you can change the data in the `y` and `x` columns and see how the weights and predictions change. You could even use some kind of data of your own for doing this. Some students who have a greater interest in professional sports than in the stock market have used offensive and defensive statistics from basketball teams to predict the point spread in playoffs.

Moreover, you can change the number of input series `x1` through `x5` by adding series such as `x6` and `x7` for other automotive companies such as Toyota and Honda. However, this is somewhat harder to do than the experiments discussed previously as it involves making changes in the formulas in the spreadsheet. On the other hand, it is a very good way to really learn how a neural net is represented and solved in a spreadsheet.

7 Further Reading

Sargent (1993) provides an introduction to neural nets. Garson (1998) presents an introduction to the use neural networks in the social sciences and a systematic coverage of the subject. Beltratti, Margarita, and Terna (1996) also present an introduction to neural networks and develop a variety of models for economic and financial modeling.

References

Beltratti, A. S. and Terna, M. P.: 1996, *Neural Networks for Economic and Financial Modeling*, International Thompson Computer Press.

Garson, G. D.: 1998, *Neural Networks: An Introductory Guide for Social Scientists*, New Technologies for Social Research series, SAGE Publications Ltd, London, United Kingdom.

Judd, K. L.: 1998, *Numerical methods in economics*, MIT Press, Cambridge, Massachussets.

Miranda, M. J. and Fackler, P. L.: 2002, *Applied Computational Economics and Finance*, MIT Press, Cambridge, Massachussets.

Sargent, T. J.: 1993, *Bounded rationality in macroeconomics*, Clarendon Press, Oxford, United Kingdom.

Appendix

A Running the Solver in Excel

Download the file for the growth model or for the neural net from the book web site. Your version of Excel may not have the Solver option available by default. To check this look for the Solver option on the Tools menu. If you do not find it, click on `Add-Ins`, check Solver Add-in, and click OK. Then look in the Tools menu again.

B Nonlinear Optimization Solvers

Solving nonlinear optimization problems usually requires the use of numerical methods. In general, those methods consist of a *smart* trial-and-error algorithm that is a finite sequence of computational steps designed to look for convergence to a solution. There is a variety of algorithms to solve nonlinear problems. Some of them are global methods, in the sense that they perform a parallel exploration of many regions of the optimization space, for example, the genetic algorithm. Other are local, as they tend to focus on the exploration of a particular region of the optimization space. We introduce here two of the most popular local methods—the gradient and the Newton methods—used by the solvers in Excel, **GAMS**, and **MATLAB**, but before introducing them, we give with a simple example.

Suppose that we are trying to find the maximum of a nonlinear function

$$y = f(x) \tag{B-1}$$

such as the one represented in Figure 7.

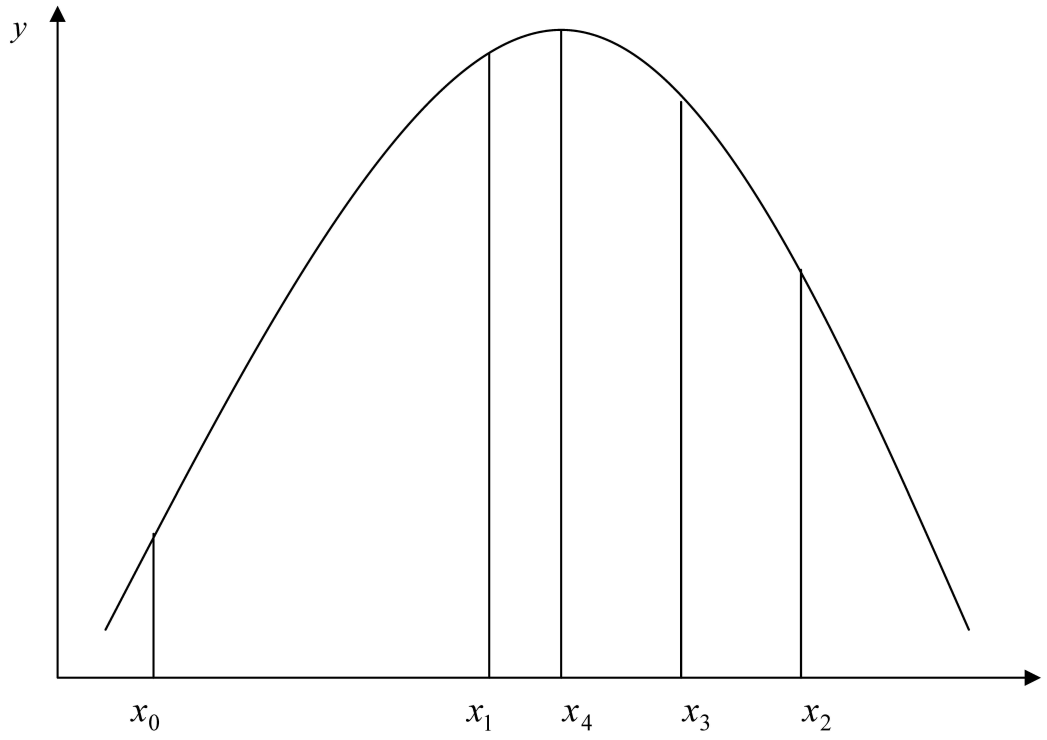
A simple and very rudimentary algorithm to find the solution might be as follows: We choose an arbitrary initial value for x, such as x_0 in Figure 7, and compute the corresponding $y_0 = f(x_0)$. We then increase that value by a constant magnitude h (we name this magnitude the *search step*) that we also choose in an arbitrary way. For the new value of x, that is x_1, we compute the corresponding value of

$$y_1 = f(x_1) = f(x_0 + h) \tag{B-2}$$

and compare this value to the one obtained in the previous step. We continue to do this as long as the differences between two successive values of y are positive (negative for a minimization problem). As soon as we compute a difference with a negative sign (in Figure F.1 this would correspond to x_2), we reverse the direction of the search. We begin to move in the opposite direction along x (i.e., subtracting h from x) and use a smaller value for h than the one we were using while we moved in the opposite direction. We continue like this until we again find a difference between two successive values of y that is negative, at which point we again reverse the direction of the search and further reduce the size of h, and so on. We stop when the difference between two successive values of y falls below a preestablished tolerance limit.

The gradient and the Newton methods are iterative like the one just presented.

Figure 7: A nonlinear function.



However, they exploit local information about the form of the function; that is, they use the function's derivatives. To illustrate this we change to a multivariate example. In this case we use the following equation to obtain each new value of the vector x:

$$x_{n+1} = x_n + h\Delta x \tag{B-3}$$

where h is the search step—now always a positive value—and Δx is the direction of change, which, as we will see, is determined by the function's derivatives.

The gradient method uses the first derivative or gradient, which gives us information about how the function changes in the neighborhood of a given point. Its basic framework is the well-known first-order Taylor approximation,

$$f(x_{n+1}) \cong f(x_n) + h\nabla f(x_n)\Delta x \tag{B-4}$$

where $\nabla f(x_n)$ is the gradient vector. Note that since h is supposed to be positive, the best direction of motion is

$$\Delta x = \nabla f(x_n) \tag{B-5}$$

for a maximization problem, since

$$f(x_{n+1}) \cong f(x_n) + h\nabla(f(x_n))^2 > f(x_n) \tag{B-6}$$

In addition, for a minimization problem

$$\Delta x = -\nabla f(x_0) \tag{B-7}$$

since

$$f(x_{n+1}) \cong f(x_n) - h\nabla(f(x_n))^2 < f(x_n) \tag{B-8}$$

The basic framework of the Newton method is the second-order Taylor approximation

$$f(x_{n+1}) \cong f(x_n) + h\nabla f(x_n)\Delta x + \frac{h}{2}\Delta x' H(x_n)\Delta x \tag{B-9}$$

where $H(x_0)$ is the second-order derivative or Hessian, which tells us how the slope of the function changes in a neighborhood of a given point.

Assuming the Taylor expansion of a function f is a good global approximation of that function, we approximate the optimum value of f by optimizing its

Taylor expansion. In our case, this is equivalent to saying that to determine the best direction of motion Δx we have to optimize the expression (f.9). Differentiating (f.9) with respect to Δx, setting the result equal to zero, and solving for Δx, we obtain

$$\Delta x = -\frac{\nabla f(x_n)}{H(x_n)} \tag{B-10}$$

which is the best direction of motion for the Newton method.

Sometimes iterative methods like the ones presented here do not converge to a solution after a finite number of iterations. This problem can be overcome by changing the maximum number of iterations, or the size of the search step, or the tolerance limit, or the initial value of the search. Most solvers allow you to change these parameters.

Note also, as is the general case for numerical methods dealing with nonlinear optimization problems, that if there is more than one local optimum we will find only one of them. Thus, we never know for sure if the optimum we reached was a local or a global one. A rough way of dealing with this difficulty is to solve the problem providing the algorithm with alternative initial values of the search.

In this appendix we presented three numerical methods of increasing complexity. Of course, the more complex ones make use of more information, thus reducing, in general, the number of steps needed to achieve convergence. However, those steps become more complex, since they require the computation of a gradient or a Hessian. Then there are trade-offs to be evaluated when choosing a solution method.

There are additional methods for solving nonlinear problems numerically—for example, the conjugate gradient method, the penalty function method, and the sequential quadratic programming—a number of which extend, combine, or mimic the ones introduced here. For a comprehensive presentation refer to Judd (1998) and Miranda and Fackler (2002). The Excel solver uses a conjugate gradient method or a Newton method. **GAMS** uses a variety of methods, depending on what you choose or have set up as the default nonlinear solver. The **MATLAB** solver used in Chapter 7 and invoked by the fmincon function uses a sequential quadratic programming method. For details on the specific methods used by **Excel**, **GAMS**, and **MATLAB** refer to their corresponding user's and solver's manuals.