Our ongoing advanced series on AI trading attempts to improve a RSI-based trend following system.

APPLYING MACHINE LEARNING SYSTEMS TO TRADING

by Murray A. Ruggiero Jr.

During the past year, we have covered many different machine learning methods and discussed how they can be used in trading. Now it's time to discuss real trading applications. (See the resources box at the end of this article for multiple sources on the basics of machine learning.)

We will start with a familiar concept that has proven itself to be a reliable target for these models: Predicting traditional technical indicators. We'll walk through this process, step by step. Our goal is to create an improved version of our regular system, not a perfect version.

The system we will attempt to improve is fairly simple: It uses the Relative Strength Index (RSI) as a trend-following system that buys when RSI surpasses a set parameter and sells when it falls below its sell parameter. The rules are:

```
Sub RSIClassicX(SLen,BuyTrig,SellTrig)
Dim RSIVal As BarArray
RSIVal=RSIClassicX(Close,SLen)
If RSIVal>BuyTrig Then Buy("",1,0,Market,Day)
If RSIVal<SellTrig Then Sell("",1,0,Market,Day)
If RSIVal<50 And MarketPosition=1 Then
    ExitLong("","",1,0,Market,Day)
If RSIVal>50 And MarketPosition=-1 Then
    ExitShort("","",1,0,Market,Day)
End Sub
```

For our base results, we will run this basic RSI system on crude oil from Jan. 4, 1984, through the present. We are only interested in the results and do not presume this will perform as a standalone system; therefore, we are not deducting any slippage and commission. For our parameters, we settle on 20, 50 and 50. This set doesn't make the most money (net profit of $157,250 over the test period), but testing suggests it's robust; nearby values perform as well if not better.

Our next step is to take into consideration the performance if we know the value of the RSI up to five days into the future. "Future cast" (left) includes the results from two days late to five days early. If the earlier results perform better, then we know that a neural network that can accurately predict the indicator will be extremely valuable.

Clearly, knowing RSI perfectly one to five days into the future is powerful. It makes 700% to 800% more than the base case. While we can't expect our neural network to forecast perfectly, even a modestly accurate prediction performance should result in a nice bump in performance.

ADDRESSING LAG

RSI has been in the public domain since June 1978 when *Futures* magazine (then called *Commodities*) published an article by J. Welles Wilder that introduced it. However, there are some slight variations to its calculation. To remove some of the lag in the indicator we need to study the code behind it (see "Behind the scenes," right).

Looking at the code, these are the three major components of creating the indicator that can be used to create our prediction:

```
UpAvg = UpSum / MyRange
DownAvg = DownSum / MyRange
RSIClassicX = 100 * UpAvg / (UpAvg +
    DownAvg)
```

FUTURE CAST

The RSI system goes from run-of-the-mill to exceptional if only we could predict the future and know the value of the indicator several days ahead of time.

RSI day shift	Net profit
-2	$136,520
-1	$143,920
0	$157,250
1	$1,057,850
2	$1,205,110
3	$1,287,920
4	$1,254,220
5	$1,261,740

Source: TradersStudio

We will begin testing with random weights and use gradient descent to find a local minimum. Note that each run can produce different results due to the starting with random weights. We need to train and test multiple times and save the best networks to use for out-of-sample testing. If we are retraining every 100 bars we will test out-of-sample for 50 bars and move to real trading for 50 bars. We will trade with the best network based on performance of training set error and the out-of-sample period.

For our analysis, we will use TradersStudio Turbo with Neural Genius. Our goal is to show how we can improve performance of the base system. We will use the components used to calculate RSI and create an RSI function that returns multiple values. This is the call for the new function:

```
RSIClassicplus(Price As BarArray, Length, byref
    OupAvg,byref ODnAvg)
```

Here is the code that demonstrates how this neural network script works:

```
RSIVal=RSIClassicplus(Close,SLen,UpVal,DnVal)
UpValBArray=UpVal
DnValBArray=DnVal

Input1=RSIVal[LookAhead]
Input2=RSIVal[LookAhead+1]
Input3=RSIVal[LookAhead+3]
Input4=RSIVal[LookAhead+5]

Input5=UpValBArray[LookAhead]
Input6=UpValBArray[LookAhead+1]
Input7=UpValBArray[LookAhead+3]
Input8=UpValBArray[LookAhead+5]

Input9=DnValBArray[LookAhead]
Input10=DnValBArray[LookAhead+1]
Input11=DnValBArray[LookAhead+3]
Input12=DnValBArray[LookAhead+5]

Output=RSIVal
```

We are using RSI sampled, as well as the UpValue and DownValue component from RSI. When developing a neural network model we are predicting future values. To achieve this, we shift the input data back the same number of bars we are predicting into the

BEHIND THE SCENES

Here's the code for the version of RSI that we are using.

```
Function RSIClassicX(Price As BarArray, Length) As BarArray
    Dim Counter As BarArray
    Dim DownAmt As BarArray
    Dim UpAmt As BarArray
    Dim UpSum As BarArray
    Dim DownSum As BarArray
    Dim UpAvg As BarArray
    Dim DownAvg As BarArray
    Dim MyRange As BarArray
    If BarNumber=FirstBar Then
        Counter = 0
        DownAmt = 0
        UpAmt = 0
        UpSum = 0
        DownSum = 0
        UpAvg = 0
        DownAvg = 0
        MyRange = 0
    End If
    MyRange = Length
    UpSum = 0
    DownSum = 0
    For Counter = 0 To MyRange-1
        UpAmt = Price[Counter] - Price[Counter + 1]
        If UpAmt >= 0 Then
            DownAmt = 0
        Else
            DownAmt = -UpAmt
            UpAmt = 0
        End If
        UpSum = UpSum + UpAmt
    Next
    UpAvg = UpSum / MyRange
    DownAvg = DownSum / MyRange
    If UpAvg+DownAvg <> 0 Then
        RSIClassicX = 100 * UpAvg / (UpAvg + DownAvg)
    Else
        RSIClassicX = 0
    End If
End Function
```

Source: TradersStudio

TRIAL COMPARISON

We ran the model 10 times during the data set. We can get an idea of how receptive the base model is to improvement.

Parameters	Net Profit	Trades	Win%	AveTrade	W/L	DD	PF
1	$224,530	632	40.51	$355.27	2.12	($55,670)	1.47
2	$180,390	670	38.36	$269.24	2.14	($36,210)	1.35
3	$224,390	638	40.6	$351.71	2.1	($38,980)	1.46
4	$229,370	636	38.68	$360.64	2.28	($42,310)	1.47
5	$217,010	640	41.25	$339.08	2.02	($47,250)	1.45
6	$254,990	644	40.22	$395.95	2.28	($31,990)	1.57
7	$225,510	644	40.37	$350.17	2.11	($40,050)	1.46
8	$206,810	676	40.24	$305.93	2.03	($41,230)	1.4
9	$237,030	658	41.64	$360.23	2.06	($44,680)	1.49
10	$232,790	614	38.93	$379.14	2.3	($34,790)	1.49

future; that is how we create our training set.

Next, let's review the code that initializes the neural network and sets the variables that allow us to retrain every 100 bars:

```
FirstTraining = 500
Dim trainonthisbar
Dim Retrainbar
Retrainbar=100
If BarNumber=FirstBar Then
        NeuralGenius_Initialize()
End If
trainonthisbar = 0
If BarNumber - FirstBar = FirstTraining Then
        trainonthisbar = 1
End If
Dim X
Dim Val
Val=((BarNumber-FirstBar)/(Retrainbar))
If BarNumber-FirstBar>FirstTraining+Retrainbar
   Then
        If CInt(Val)=Val Then
            trainonthisbar = 1
        End If
End If
```

We also set the parameters for the neural network. This logic set **trainonthisbar** to 1 every 100 bars. This way we can retrain every 100 bars. We can change **Retrainbar** or set it from an input so we can test using different retrain periods. Now we need to load the data into the neural network and then train:

```
BP_NetSetParameters(9,1250,0.25,0.1,1,0)
```

This command initializes a backpropagation neural network with nine hidden nodes, runs it for 1,250 epochs with a learning rate of 0.25, a momentum of 0.1, normalizing the variations between their maximum and minimum value (1: normalize mean +/- deviation within the function range), and by default we start with type 0 normalization.

The training process creates the data set for training by building the array and then setting each variable using the NS_AddSignal function. We then create the output variable and train, printing the necessary information to use the neural network in our trading system. We use the same rules, but now just the predicted values:

```
If PreRSI>BuyLev Then Buy("",1,0,Market,Day)
        If PreRSI<SellLev Then
   Sell("",1,0,Market,Day)
        If PreRSI<50 And MarketPosition=1 Then
   ExitLong("","",1,0,Market,Day)
        If PreRSI>50 And MarketPosition=-1
   Then ExitShort("","",1,0,Market,Day)
End If
```

PREDICTING RSI

To begin testing our model, we use the same parameters used for the original 20-period RSI with triggers at 50. We will attempt to predict its value one to two days in the future.

We can optimize several variables within the neural network when we are doing this. First, there is the number of hidden nodes that will range from 50% of the number of inputs to 100% of the number of inputs. We can also optimize the retraining window, as well as the size of the training window.

Because neural networks start with random weights, each time we train we get different results. When we develop commercial systems employing neural networks, we run each training window 10 times and then select the best network from that training to use for the new window. We save the best network for each window so we can reproduce the backtested results for the system. "Trial comparison" (above) shows the results for running the model 10 times over the complete data set. The average across these 10 trials is $223,000, which is significantly better than the acceptably robust results without the neural network. Also, note that these are

walk-forward results; this performance is being realized on data the neural network filter has not seen.

We don't have to select the best network for each training period. We could take the best two to five networks and average the outputs; this will stabilize the results more. While instability in runs is something you need to watch, that isn't the case here. The standard deviation of these runs is less than 10%. This is more stable than even nearby parameters without the neural network, where changes from 18-24 for the RSI length resulted in large changes in net profit. This illustrates another tenet of this analysis: Optimization is almost always safer and more robust if done on the neural network filter rather than the core technical logic itself.

This is the first cut at predicting RSI. Here are some ways we could build a better model:

1. Sample further back. Currently we are taking a sample five bars back for our three inputs. We could expand that to seven to 10 bars. When you increase your lookback, you spread out your sampling and often improve out-of-sample results.
2. Add inputs that are rate-of-change some of the existing inputs. This works better when we use infinite response inputs, such as those generated with exponential moving averages versus simple moving averages.
3. Use a shorter version of the indicator, such as a half-cycle version, and generate inputs from that.
4. The biggest problems with predictions result from when the prediction period is during a turning point. This is why turning point technology, such as intermarket analysis and cycle analysis can improve performance; it gives the system a defense at its most vulnerable times.
5. Add volatility as an input, both historical as well as implied volatility.
6. Add post processing. We can post process by tracking errors in forecasts and trying to correct for them. We could even use a second neural network to predict errors, which could be used to improve our forecasts
7. In this type of model, we are using the neural network as a nonlinear regression model. We can also use support vector machines using regression models to develop this further as long as we keep the number of inputs low, about a dozen or less. This example is on the larger side of something we would train using the Support Vector Machine.

RESOURCES

Still not solid on machine-learning basics? Consider the following resources.

- Amazon.com/Clever-Algorithms-Nature-Inspired-Programming-Recipes/dp/1446785068/
- Machinelearningmastery.com/master-machine-learning-algorithms/
- Machinelearningmastery.com/machine-learning-with-python/
- Machinelearningmastery.com/machine-learning-with-r/
- Systematicinvestor.wordpress.com/systematic-investor-toolbox/
- Gbeced.github.io/pyalgotrade/

Source: TradersStudio

Since the mid-1990s, we have been covering the promise of advanced cybernetic trading methods in this column. It's now time for trading technology to deliver on the promises of the past quarter century. Advances in computer hardware, software and the unlimited storage and processing power of the cloud provide the tools we need to tap into the true power of market analysis. This doesn't mean the past couple decades have been for naught; we have used this time wisely, coming to grips with the methodologies that work best on all time frames, whether that is high-frequency tick data or monthly moves.

The natural progression of this movement is a fully automated trading solution, from creation to execution. These artificial trading intelligence systems will excel at what human traders have struggled with for centuries, and the vast majority of individual traders will be unable to compete. Those who adapt, however, will thrive. This work, shared here for the past 25 years, has built up the base of knowledge you need to exploit the nuanced relationship between technology and trading, and profit in this brave new world. Δ

Murray A. Ruggiero Jr. is the author of "Cybernetic Trading Strategies."