aNNie Trading

Hacking The Stock Market

−= Evolution of the Ultimate Trading System =−

rewt@annietrading.com

introduction

The Ultimate Trading System has the following properties: self taught, self adjusting, persistent profitability, fully automated. Let's take a look at each one.
− self taught − able to derive a technical trading technique without using a predefined technical indicator.
− self adjusting − able to refine itself periodically to adapt to changing market conditions or to continue the learning process.
− fully automated − no human intervention.
− persistent profitability − remains profitable indefinitely.

Such a system would necessarily employ AI technologies such as artificial neural networks and genetic algorithms. Though there are other technologies that could be used successfully, these happen to be the two that form the basis for the system described in this paper. It is not the technology itself that is the key, it is the proper application of the chosen technology. Whatever it is.

choosing a target

To begin a design we must start with some observable market conditions where there is profit. Choosing a specific target at the outset is necessary to limit the scope of the effort. As layers of complexity are added, so are the number of variables that will need refinement. This increasing number of variables can quickly become unmanageable or even impossible to explore.

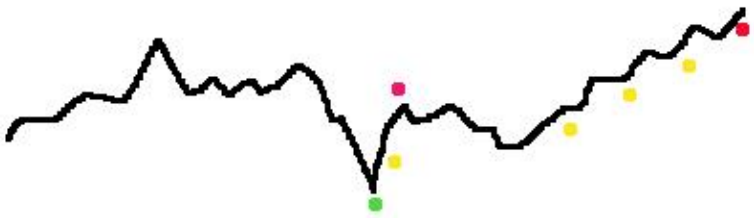
Our observable is that on any given day there are issues with intraday swings as much as 10 or 100 percent. These may follow or run counter to the overall market trend. Such a target would provide for "persistent profitability" that is independent of overall market trends. To further avoid the effects of long term trends we focus on strictly intraday data. To maintain "persistence" we must also develop a system that ignores artifacts that are unique to specific issues. Otherwise we will be idle during periods that those issues are not moving at all.

Another observation is that there are time periods intraday that are more volatile than others. The earliest period is at the open, but these events are primarily interday and overnight news plays. Since our target is intraday swings we will avoid periods of interday trading such as "near the open" and "near the close". And since our aNN system needs some time period of intraday test data before it can begin providing output, we will necessarily begin trading after the open.

We need to refine our target to fit within computational constraints. For a single PC we can run 10s of artificial neural networks (aNN) against hundreds of issues in somewhat real time (seconds) with each aNN using 10s of inputs. Multiple aNNs allows for a set of active black box systems that can learn different market events. The remaining inactive systems serve as a dumping ground for training inputs that just don't seem to fit any of the active systems.

To be a true black box system we need to give the system inputs in a reasonably raw form. For 10s of inputs to represent minutes of test data we need some meta−representation of the most fundamental inputs: bid, ask, last sale, day high, day low and volume. Further, we need to normalize the inputs in some way to attain issue independence and continue down the path of persistent profitability. Reasonable meta computations are averages, ranges and deltas in small time intervals. These kinds of meta inputs allow the system to teach itself which of the the many technical analysis techniques make sense without preconditioning for any one. The system will essentially invent techniques that may or may not relate to any known conventional technical analysis method. If technical traders are driving the market then the system will derive the optimum combination of those techniques that causes our target observable.

To stay within our computational constraints we need to choose a profitable market event that has a handful of "reasons" for a stock to move in our favor. If our inputs cannot be used to derive some number of active subsystems then we would have everything in the dumping ground and the system will learn to "not trade". An example curve will illustrate our chosen target.



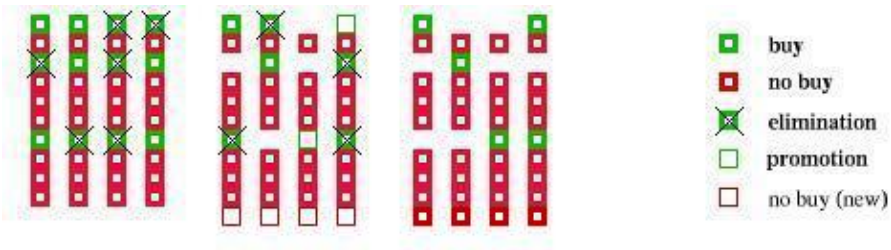
On any trading day there are many issues that have quick reversals followed by quick recovery and a profitable exit. By concentrating on quick actions, we increase the probability that a short test window has enough data for an aNN to work with. There are other areas with profitable entry points but they are "reactive" as in conventional moving average technical analysis. Our preference is to be "predictive" and maximize profit by anticipating the reversal.

---

the core of the system

The idea behind running multiple aNNs is that there is always more than one reason for a reversal. Instead of striving for a single aNN that is the most profitable, most common, or something else, we allow the independent subsystems to choose what works best for the system as a whole. To get started we set some basic criteria for what constitutes a reversal. For example "a 5% drop in 10 minutes followed by a 5% rise in 20 minutes". We can then search through historical data for these events and create test data for desired output equal "1", or buy. At the same time we need to come up with even more data that we can set as a desired output equal "0", or don't buy. Extensive experimentation indicates that you need a number of "0" inputs that is 10s or 100s times the number of "1" inputs for intraday market trading. Experimentation also indicates that the historical data set should cover a 6 month to 2 year period. We want our system to train through a wide variety of market conditions to learn to be independent of overall trends.

To come up with an initial set of aNN subsystems we can jump start all of them with the same test data. Through repeated backtests we can then somewhat evenly distribute the inputs while clustering "like" outputs together. When this system stabilizes we will have aNN subsystems that have either learned unique strategies for our target observable or become one of the dumping ground systems. The graphic illustrates the progression from elimination to shuffling as the subsystems settle on a balanced distribution of clustered subsystems, left to right.



In the end we have a fairly even distribution of inputs and a set of black boxes that have learned how to detect our observed target. Since we have done nothing but provide sample events (and non−events) then we have satisfied the "self taught" property. Training sets for aNNs requires a certain amount of noise to produce good results in testing. Market data is inherently noisy so we don't need to introduce artificial noise. The market noise remains in the training set since we never eliminate training inputs.

---

testing the core

Before continuing with the other elements of the Ultimate Trading System it's important that the core aNN subsystem is a solid performer and is showing profit potential. Introducing more layers of complexity with nothing under the hood leads to disappointing failure. There are already many adjustable parameters to be explored and small adjustments can have a huge impact on performance. We continue after these are reasonably exhausted and the underlying system is where it needs to be.

Let's test what we have under the hood so far. Two complete aNN systems were designed and trained on 15 months of data and then frozen with no further adjustment. We then ran a simple performance test for 18 months that measured percent price increase in the 10 minutes that followed a consolidated "buy" signal.

aNNie threshold	# of signals*	avg mo ve per signal
0.2	6910	0.85%
0.5	1709	1.39%
1.0	369	2.41%

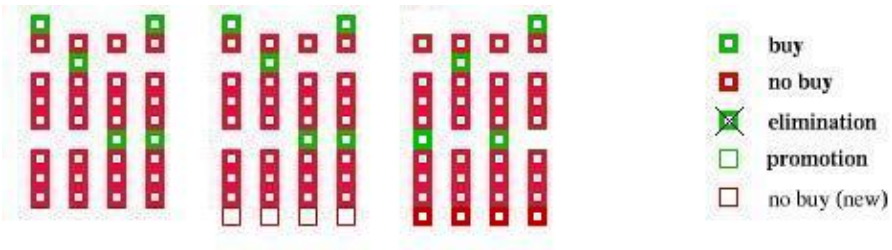
*July 2003 thru Jan 2005

At this point we only want to know if we're on the right track. The results utilize a threshold on the sum of 20 "buy" aNN subsystems (10 aNN subnets for each of the 2 systems). We can see quite clearly that thresholds are correlated with average price moves per trade. This in itself is not a tradable system but verifies that we have goodness under the hood as we continue our design.

---

daily genetic updates

For our particular target entry point, and perhaps any target, simply rerunning the clustering process will yield different sets of aNN subsystems with different test performance. To create a stable overall trading system that we have confidence in, we create 3 complete aNN systems from an initial jumpstart as above and call them "parents". We then begin a genetic update phase where each parent creates a child using new training input from the new day. The final step is to wrap the system in a genetic selection algorithm while exposing it to new market data. In the graphic we show the creation of a new child.



To determine which of the 6 systems will be the parents in future trading days, we need some realistic measure of trading performance. Proper selection incorporates exactly what we want out of The Ultimate Trading System. Examples are things like highest average profit per trade or most profit with one trade per week. Bottom line is that the selection and update strategy should incorporate our trading strategy. We need to develop more of a trading system before we get to the updates.

---

trading system parameters

For this system we decide that we want approximately one trade every day or so. We'll let the system decide for itself what works best (fewer trades at high profit vs more trades at a lesser profit). We will also set a threshold on the the number of active and dormant subsystems used for trading. Eliminating signals from the dumping ground has been shown to significantly boost performance. Over time we would expect the daily updates to eventually stomp dumping ground signals to zero, but we can't predict how long it will take for the market to provide all of the necessary inputs. The bottom line will be profit over the course of a 40 day backtest with a fixed entry dollar amount on each trade.

An interesting side note about using data from training periods as backtest data − it is perfectly normal and actually desireable for an aNN to *not* be able to test perfectly against its own training data. A perfect result usually indicates overtraining and will fail miserably when applied to new test data. If it turns out that both are testing well, then the aNN probably reduces to a simple algorithmic solution that didn't need a black box in the first place.

While creating "buy" and "no buy" training data for our original we're also creating other aNN subsystems that have "exit" and "no exit" inputs. There are many ways to deal with exits such as fixed hold time, close at a specified time, build independent "short sell" aNNs, etc. In this system we chose to create 3 separate "exit" systems (30 subnetworks total) to produce exit signals. In addition we have a hard exit time of 3pm so that no positions are held overnight. Each of the 3 aNN systems are variations of a "peak" and are chosen at a point that would produce the exit level that maximizes profit. And further, "exit" training data is only aquired following "buy" training inputs.

We're still not quite there yet. Our genetic selection should also encompass the logic that simulates actual trading. In other words, we need to simulate our own trades and not compute results based on trades that have already occurred. Every little realistic detail that is incorporated into performance testing and genetic feedback helps the system learn about what would occur in actual trading. Examples of bad assumptions are using last sale or best bid for a "buy" entry without confirming that a following trade would have executed it. For our Ultimate Trading System we have logic built in to training and testing that enters best bid and best ask and monitors subsequent trades to see if execution would have occurred. One could argue that our bid could have prompted a better bid and so on and so on, but we can never really solve that problem in simulations. So our simulation is as good as it can be. Here is an example of our trading simulation:

starting aNNie−v2.6 trading sim for 20041229
aNNie2 bids OLAB 3.81 10:42:45 [3.83 3.80x3.95]

```
aNNie2 buys OLAB 3.81 10:42:48 [3.81 3.81x3.95]
aNNie2 asks OLAB 4.07 10:43:20 [3.99 3.81x4.08]
aNNie2 sell OLAB 4.07 from 3.81 10:42:48 at 10:44:13, 6.8% in 1 mins [4.07 3.96x4.07]
finished aNNie−v2.6 trading sim for 20041229
starting aNNie−v2.6 trading sim for 20041231
aNNie2 buys MDII 1.06 09:45:13 [1.04 1.04x1.06]
aNNie2 asks MDII 1.14 09:47:18 [1.15 1.11x1.15]
aNNie2 sell MDII 1.14 from 1.06 09:45:13 at 09:47:29, 7.5% in 2 mins [1.15 1.10x1.15]
aNNie2 buys HOFF 1.98 11:08:53 [1.97 1.97x1.98]
aNNie2 sell HOFF 2.16 from 1.98 11:08:53 at 11:10:42, 9.1% in 1 mins [2.16 2.16x2.17]
finished aNNie−v2.6 trading sim for 20041231
starting aNNie−v2.6 trading sim for 20050103
aNNie2 bids CLTK 1.31 10:16:42 [1.31 1.30x1.40]
aNNie2 buys CLTK 1.31 10:16:43 [1.30 1.30x1.40]
aNNie2 bids CLTK 1.26 10:18:04 [1.27 1.25x1.30]
aNNie2 adds CLTK 1.26 10:18:06 [1.25 1.25x1.28]
aNNie2 bids PRLS 1.21 10:26:36 [1.20 1.20x1.34]
aNNie2 buys PRLS 1.21 10:26:38 [1.20 1.20x1.34]
aNNie2 sell CLTK 1.30 from 1.26 10:18:06 at 10:27:25, 3.2% in 9 mins [1.32 1.30x1.32]
aNNie2 sell CLTK 1.30 from 1.31 10:16:43 at 10:27:25, −0.8% in 10 mins [1.32 1.30x1.32]
aNNie2 sell PRLS 1.32 from 1.21 10:26:38 at 15:07:08, 9.1% in 280 mins [1.32 1.32x1.34]
finished aNNie−v2.6 trading sim for 20050103
```

Recall that we have some number of subsystem clusters that have learned something and others that became the dumping ground. We can pick some number of all subsystems to participate in trading and essentially truncate the system there and filter out any signals from the dumping ground. Rather than do extensive testing we will let genetic updates figure out how best to deal with it. We now have 1 "buy" and 3 "exit" networks for a total of 40 aNN subsystems. We arrived at these numbers to provide for rapid "buy" entries and to create a large pool of "exit" systems for the genetics to choose from upon entry.

To get us close to our "one trade every day or so" goal, we first find a threshold for the 10 "buy" subnetworks that gives us just over 40 trades in backtesting the prior 40 days. Then we adjust the threshold for the 30 "exit" subnetworks to maximize profit. To force the exit nets to learn something, the profit is scaled by the threshold. The scaling factor is something like 10% per 0.1 threshold level. Otherwise the genetics can waste time exploring buys while ignoring exits. So for an exit threshold of 0.5 to be used, profit must be 50% more than the profit at the 0.1 threshold.

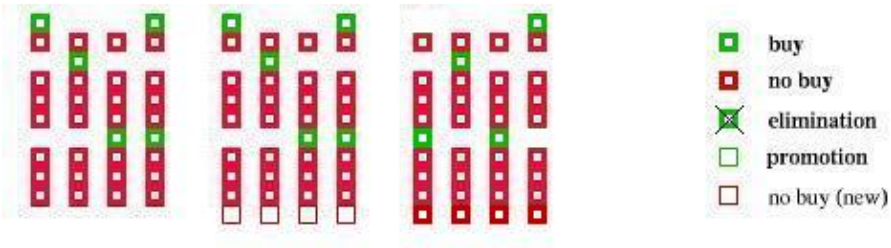
Of the 40 total we decide that 10 will be active and the rest dormant. To determine the top ten we rank the subsystems (including "buy" and "exit") in order of their average profit per trade weighted by their signal level. For the 40 total trades in our backtest there can be signals combinations that cause entries and exits as well as other complex situations. Rather than explore all the possibilities we just let the genetic selection process work out the details over time.

---

genetic selection

We now have our system simulator so let's get back to a genetic selection method. We need to perform gentle updates and allow the system time to stabilize. Adding "no buy" inputs for every single output signal that isn't perfect. We also don't want the parents to be accumulating days of data that created only minute improvements in trading performance. Without gentle updates the system would learn to "not trade". Our goal here is to give the system a few months of new data and expose it to a wide variety of market conditions while stomping out really bad outputs.

So let's start at day one. We choose the one with the best backtest performance to be "best parent". At the end of day one, we will create a child from each of the parents that incorporates new training data for the current day. Here's the old figure for child creation:



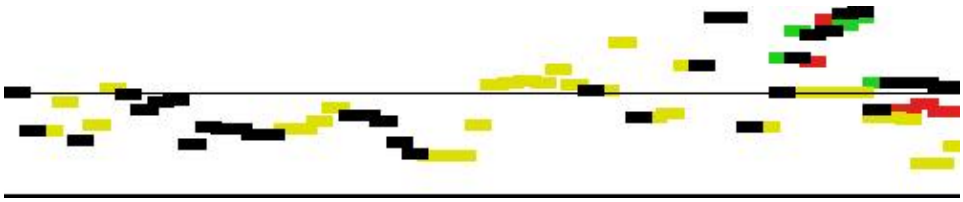
We then backtest all 6 systems through our complete trading simulation to arrive at total profit for each over last 40 day period. We then compare 5 systems to the previous best parent. If none of them are 10% better, then we stick with our current best system. If another parent beats our previous best by 10% then it becomes our best parent for tomorrow. If a child beast all of the parents by 10% then the child replaces the weakest parent and becomes the new best parent. In time, all of the parents will have accumulated similar updates and have similar performance. At that point we have the Ultimate Trading System. Genetic selection continues into the future to continuously fill in any gaps that may still be present in our training data.

It's important to watch how the system is shaping up in the first months of learning. There will be periods when a parent system gets a few lucky trades that are simply too good to continue long term. For example, a handful of 50% gains with no exit that beats out a more stable system with a dozen 5% gains. A lucky best parent will eventually buy and hold for a significant loss and either create a better child or be replaced by another system. As we see this occur we want to ensure that the other parents haven't been lead down the lucky path so that the system as a whole can get back to reality. This is why we require a modest improvement of 10% before replacing an existing best parent.

---

genetics in action

Let's take a look at the first 3 months of genetics in action (60 trading days). In the following graphic we have zero profit at the bottom and a thin line for 40 day profit equal to the amount entered per trade. In color we have the 3 parents and in black the best parent. When you see color preceding black there was a change in best parent. Things to note are that we are hovering below the 100% line for a month or so. And if you use your imagination you can see a general upward trend.

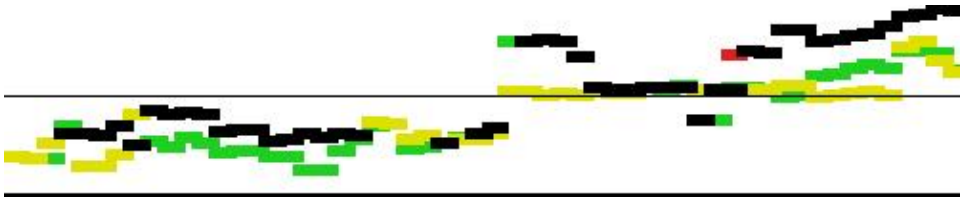


Note that the graphic is a look backward into the genetic processes. The lack of colors present in the first 2 months shows that all parents became part of the same family tree after that point. The fairly wide distribution of the 3 parent colors in the last weeks tells us that each parent tree is still behaving quite differently. There is more to learn.

Earlier we measured the performance of our underlying aNN system to ensure that there was goodness under the hood. This needs to occur as each of the subsequent processing layers are incorporated into the design. This way we can explore a small number of variables and minimize their dependencies. Not described here

are several trading system designs that used stop loss and other conventional technical exit strategiess. At the last layer of genetic feedback there also were several iterations of experimentation with different parent numbers and child promotion techniques. The most recent refinement, for example, was the use of subnet truncation which led to an almost 2x performance increase. The initial training genetic curves without truncation were essentially capped at the 100% line shown above.

With an unlimited number of tweakable parameters and limited processing power, we use the test results to feed back new adjustments and strategies. In our 3 month curve, for example, we can see a few downward trends followed by erratic correction. If we can find the root of the problem then it may be wise to make some adjustments and try again. And so we do.



We have incorporated the following changes:

(before/after)
− (limited/variable) number of trading subnets
− (limited/variable) number of trades
− (tuned/fixed) entry and exit thresholds

The subnet adjustment gives the system more flexibility in choosing how to trade (more trades at less per−trade profit vs less trades at higher per−trade profit). As a result, the system chose to increase the number of trades for less avg trade profit during periods of low market volatility (fewer trading opportunities). The fixed entry and exit thresholds precondition the training with an "answer" based on almost all prior implementations and removes one of the lessons that needed to be learned.

The bottom line is a smoother, more profitable learning period. The single large correction created parents that choose to trade twice as much as the earlier versions. The system as a whole is still on the right track...

---

pulling the trigger

At some point you have to be comfortable with your system and pull the trigger with real money. The *comfort factor* for this system was a combination of:
− the time required to pursue the next performance tweak
− the simulated profit of the current snapshot

With a full year of genetic updates and simulated profit of 2−3 times commission, a suitable level of comfort was achieved.

Real trading brings with it the need to incorporate commission into the equation. If the reason isn't obvious consider that 100 2% profit trades makes the same money as 200 1% profit trades and then take out some fixed commission for entry and exit. Ideally these details would be determined early and incorporated into the learning process.

In the early days of live trading it was useful to have an active system in order to synchronize with a real trading account. For long−term profitability, however, it is necessary to apply commission to the model. A 5

second signal delay was also incorporated into the simulator to approximate aNN computation time and order entry delay. With these in place, the end−of−day simulation almost exactly replicates the live trading for the day.

It should be noted that these kinds of system tweaks are solely to produce a replica of live trading and performance. It should not to be confused with the perpetual system tweaks that human traders must make to their technical trading systems to remain profitable. For our system, those things are done automatically as long as our simulator is accurate.

With these last simulator tweaks in place, the system updated itself in a few trading days to make about 5 times fewer trades in a given time period. After about 3 months we are satisfied that simulator is almost an exact replica of the real executed trades. With everything fully automated there is nothing left to do but sit back and watch.

−− Last updated Jan 13, 2006 −−