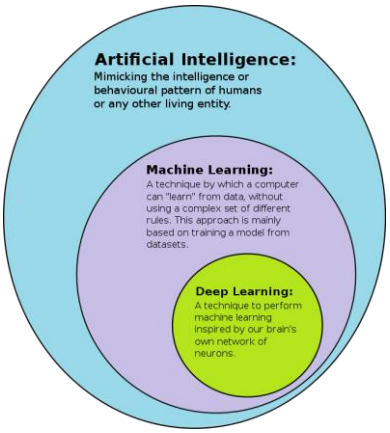
Introducing Nirvana's New Deep Learning Block for OmniTrader®

As presented by Steve Mayo, data scientist and long-time Nirvana customer

Twenty years ago, Nirvana System's pioneering ARM technology accelerated OmniTrader's rise to become the preeminent platform for advanced algorithmic trading. In their latest moonshot, Nirvana has now amplified its technological advantage with the launch of a next-generation ARM tool which will improve their existing technical-trading strategies and even allow users to employ this astronomical advance in building their own. Exalting superlatives exhausted, let's board this rocket and explore Nirvana's new **Deep Learning** strategy block.

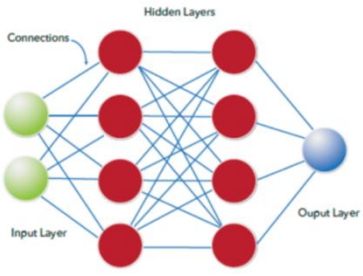
Learning the Lingo



For this discussion, blue text indicates ML lingo, should you want to learn more; orange indicates Nirvana-specific terminology or labels used in the DL block.

Machine learning (ML) refers to programming algorithms employing high-level calculus, linear algebra, and statistics to analyze and draw inferences from a set of data. ML can identify unseen patterns and relationships that other data processing approaches cannot. Euphemistically, we call these sophisticated algorithms Artificial Intelligence (AI) but, so far anyway, AI is not capable of independent thought, arguably a good thing! Deep Learning refers to newer ML architectures that, given enough time and computing power, can learn complex nonlinear relationships in huge datasets having seemingly infinite permutations. Think spaceships and stocks here!

One promising branch of Deep Learning, Neural Networks (NN), are modeled on the human brain. NN's are organized into nodes and internodal connections, like neurons with their interconnecting dendrites. In ML, a node conceptualizes the processing of a signal, and the connectors abstract how signals are passed from node to node across a network.



Nodes are organized into layers with each node in a layer connecting with every node in the following layer. Connections have coefficients, called weights, that amplify or dampen signal transmission. During each iteration through the network, called an epoch, these weights are adjusted such that the network essentially learns how to best respond to incoming data, like how the nervous system learns to respond to sensory stimuli.

Nirvana's Deep Learning block (DL) uses an NN architecture called a Multilayer Perceptron (MLP), a type of feedforward neural network. Feedforward means that information only travels in one direction through the network (i.e., no loops), starting from an Input Layer, through one or more processing layers, called Hidden Layers as their internal values are not exposed, and finally to the Output Layer. The number of nodes in each layer is configurable.

To make inferences about the future, an NN algorithm be trained using a set of calculations called features that it can use to derive numeric observations (samples) about its environment. Nirvana allows you to enter these calculations as OmniScript and calls them inputs. For a

trading system, features are historic timeseries price and volume metrics for stocks, futures, or cryptocurrencies and derivatives thereof such as indicators. Potential other features might include fundamentals, macroeconomic data, search history, chat mentions, news, and consumer sentiment, if you can get such alternative data at a reasonable cost.

The NN algorithm must be run for dozens or even hundreds of epochs. The more observations you provide and the more epochs you run, the more likely the NN will be able to learn something useful. But, as you might expect, more means longer run times and the need for greater computing power. Also, in accordance with the ole garbage-in/garbage-out adage, the quality of the observations and how they is preprocessed is very important, as are the NN's configurable settings, called Hyperparameters.

An NN is classified as a Supervised Learning system because, while training, in addition to a table of observations usually labelled $X_1 \ldots X_n$, the algorithm needs a corresponding column of expected output values, usually labelled as Y. With OmniTrader we have access to large amounts of historic trade data and can easily calculate what the system should learn from it.

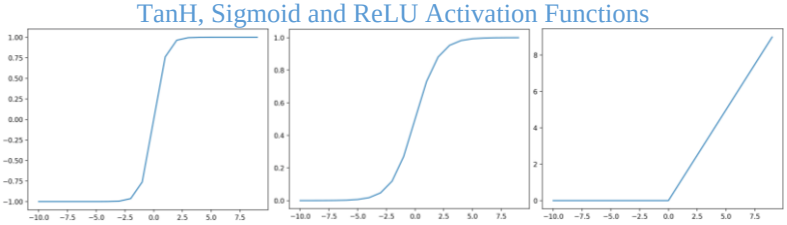
Essentially, an NN is a giant multiple regression, the ole $Y = b + W_1X_1 + W_2X_2 + \ldots W_nX_n$ equation from basic statistics, but non-linear and without the extreme curve fit. After training, you can give an NN a new row of X's, say today's prices, but now without the Y you want it to predict.

How does an MLP Learn?

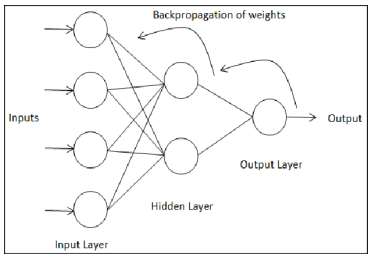
During each epoch, the set of observations (X's) for a time increment are passed by the input nodes to the first processing layer. Based on the weights, and what is called a non-linear Activation or Transfer Function, some nodes in a processing layer will get activated and pass their signals to nodes in the next layer; conversely, other nodes may block their signals.

The activation function maps inputs to an s-shaped curve that squashes the signal's dimension. Often, it is a Hyperbolic Tangent (Tanh) function that forces signal values to between -1 and 1.

TanH, Sigmoid and ReLU Activation Functions

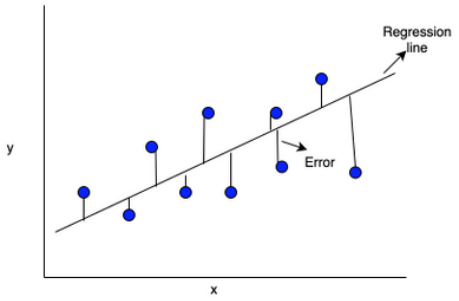


Or, you can use the Sigmoid (logistic) function which is similar but forces the signal to between 0 and 1. ReLU is a popular newer function that may be added to the DL in a future update; in theory, ReLU avoids the situation where an NN stops learning after it gets stuck at a saddle point, i.e., where the curves are changing direction.

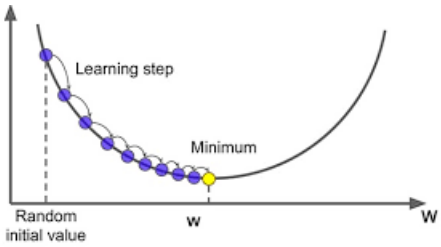


With each iteration, a methodology called backpropagation is used to adjust the weights with the goal of minimizing the error between what the network calculated as the predicted output and the true Y value. Technically, an NN checks the error on every output node, however the DL block needs only one. We say that an ML model is in convergence when the loss has been minimized, or at least when it is close to minimum.

This predicted-vs-actual comparison is based on a calculation called Mean Square Error (MSE). It resembles how a spreadsheet calculates a best-fit regression: a line is randomly plotted through the datapoints, then the distance between each point and this line is measured and squared and summed across the points to get the error rate. The line is then adjusted a bit, and the process repeats until the error is minimized. In Nirvana's DL module, the MSE is inverted and graphed as the Certainty percentage.

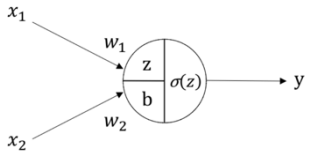


In an NN as in many other ML architectures, that increment-of-adjustment between epochs usually employs an optimization algorithm called Gradient Descent configured via the Learning Rate hyperparameter. Basically, this means that early adjustments are bigger than later ones as the error rate approaches its minimum. Nirvana politely hides this complexity but still allows the user to select between an Adaptive Learning Rate, with a specified momentum that employs gradient descent, or a Constant Learning Rate with a user-specified increment.



An Example Manual Calculation

To expand upon this explanation, let's manually calculate a single node. In the accompanying node diagram, X_1 and X_2 are the observations, W_1 and W_2 are the associated weights, Z holds the weighted input variable, b (bias) is the y-intercept which we will assume is zero, $\sigma(Z)$ holds the sigmoid activation variable, and Y is the resulting signal that will be sent to the next layer.



Let's assume we are analyzing a single stock, say with RSI(14) and MACD(12,26) as inputs. A requirement of any well-designed NN is that the inputs need to be scaled so they are comparable across the X columns. The DL block scales automatically but here, I have manually scaled my fictious inputs to between 0 and 1 using the min-max scaler equation: (x – min(x)) / (max(x) – min(x)). The calculation of single epoch is shown in this table.

X_1 RSI	**X_2** MACD	**W_1**	**W_2**	**Weighted Input Formula** $z = X_1W_1 + X_2W_2 + b$	**Sigmoid Activation Formula** $\sigma(z) = 1/(1+\exp^{-Z})$	**Passed Signal**
0.1	0.3	0.6	0.2	Z = 0.1 * 0.6 + 0.3 * 0.2 + 0 = 0.12	$\sigma(Z) = 1/(1+\exp^{-0.12}) = 0.530$	0.530

In an actual MLP, there would be more of these nodes, each doing a similar calculation. Using math too complex to explain here, backpropagation and gradient descent optimization are next used to update the weights and bias across all the nodes to finish the epoch. The process repeats until a minimum error (or maximum certainty in the DL) is reached. More correctly, the cycle repeats until Convergence is achieved, which we'll discuss momentarily.

Application to an Algorithmic Trading System

Taking this example further, let's consider how this process is implemented in a trading system. First, as mentioned above, we need to compare the final output to the actual Y values that the MLP precomputed for our dataset. This gives us the MSE for performing the optimization process, which Nirvana inverts and graphs as Certainty. The network output is labeled with Ŷ and called Yhat; in statistics, a circumflex (the hat) above a variable indicates a predicted value.

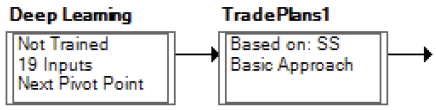
X_1 RSI	X_2 MACD	Output Yhat	True Y	MSE
0.7	0.1	0.450	0.540	0.0912

Second, what we are really after here is a trade marker, something that can be graphed as a buy/sell chevron. Thankfully, Nirvana does this tedious conversion for us in the background. In the DL block, we simply specify our desired Target Measurement Calculation, say Profit-Per-Trade (PPT), as well as a Target, say the exit defined by the strategy's trade plan or a fixed number of bars. Based on these settings, the Yhat values will be translated into a Score and used to plot a buy or sell marker on the stock graph. Over a series of bars, scores will span a range, with a high and low value. We'll see this when we get to Input Pruning.

Having summarized the underlying technology, what follows now is a near-exhaustive overview of how to configure the DL Block, with a few suggested best practices. *The red circles in the text map back to the interface screenshots at the article's end.*

Using the DL Block Within a Strategy 1

In the Strategy editor, the DL can be the first block and thereby used as a signal generator. Minimally, this option requires a TradePlan block, but you can always add other downstream blocks to your strategy such as the Vote and Filter blocks.



Alternatively, the DL block can be used as a filter. The DL block works remarkably well in this configuration, even with lower Certainty levels. In testing, adding a DL block as a filter improves most strategies but be sure to run a few tests as it could cause the strategy to be overfit.

Best Practice: As always when developing a strategy, you should use a backtest and forwardtest to evaluate the generalizability of your strategy for the given list, then test other similar lists and/or alternative date ranges to see if the performance is reasonably consistent. The DL block will use only the backtest data for building the network. Before going live, you will want to drop the backtest and run the final DL over the full set of data, just DON'T continue to tweak with the forwardtest data – that's an insidious bias and likely to result in non-generalizability.

Moving forward, as market conditions change, the DL block may require periodic retraining. If you are using dynamic networks, OmniTrader will periodically set the DL to retrain when you run the ToDo list. Again, avoid the wicked temptation to tweak in the forwardtest!

The Target Settings 2

The Calculation toggle lets you pick from several options for how the DL block will generate the Y output values and essentially what will be the loss function to be optimized.

- **Profit per Trade** – the average percentage return per trade.
- **APR** – The overall annualized percentage return.
- **Direction** –the exit price being up or down relative to the entry price, regardless of return.
- **Signal to Noise Ratio** – the ratio of desired signal relative to the statistical noise (where the DL couldn't discriminate), such that higher mean better signal quality; essentially, optimizing to the Certainty Score.
- **Profit vs. Excursion** – the return divided by Maximum Adverse Excursion (MAE), essentially the drawdown. This is a risk-adjusted return metric using the TradePlan's exit (unlike Next Pivot Point).
- **Signal vs. Wilders Risk** – based on Wells Wilder's Average True Range (ATR) indicator, this is basically an on-the-fly risk-adjusted return metric in a situation where calculation of longer-term metrics, like Calmar or Sharp, is not feasible.
- **OmniLanguage Formula** – presents text boxes for entering a custom long/short functions.

For the Target, as an enhancement to the traditional approach of having an NN algorithm optimize to a calculated metric such as the 1-day return, Nirvana's DL block can optimize to the output of an entire strategy! This is a significant advantage that I've not seen elsewhere and, speaking from experience, is a grueling challenge to program outside of OT! You simply set the Target to Strategy's Exits. With this setting, the TradePlan determines the trade's entry and exit which are then used to calculate the target Y value, in contrast to a fixed N-Bar or Next-Pivot Point exit determined without respect to the TradePlan. Other blocks such as Vote and Filter are disregarded so that the DL will be able to train on the unfiltered signal stream.

Unlike with a TradePlan, it is perfectly acceptable to leave Next Pivot Point as the target when creating a mechanical strategy for automated trading. (In the TradePlan block, the Next Pivot Point can't be determined until after it occurs so it's only useful for discretionary trading.)

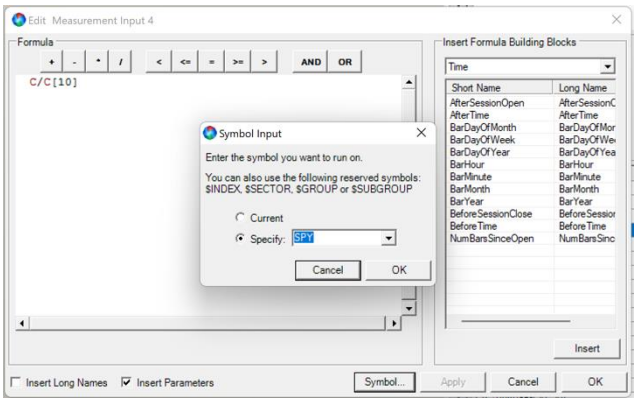
The N-bars target is intuitive and useful. You just enter the number of bars after which the target metric will be measured.

Best Practice: When building a new strategy, I once thought it best to *initially* start with Next Pivot Point or N-Bars as my optimization target. My thinking was this would allow me to refine my inputs so that I get the best possible entries without worrying about the exits. However, I've discovered that later switching to Strategy's Exits is a really big change and a network tuned to one target setting often doesn't work well for another. Now, my recommendation is to start with the target setting most appropriate for your trading style. For example, I may choose Strategy's Exits when building a trend-following system, maybe N-Bar for a Return-to-Mean (RTM) approach, and perhaps Next Pivot Point for a breakout of wave-trading strategy.

The Input Grid and Feature Engineering 3

Feature Engineering is a tedious and time-consuming step in ML. With OmniScript and Nirvana's huge library of indicators, systems, and technical & fundamental metrics, this task is greatly simplified. Still, it takes some domain knowledge and much trial & error to engineer a highly performant set of inputs.

In the Input Grid, click the Add button to open a window where you can specify the input. It can be a measurement (equation) which can include indicators and price/volume/fundamental metrics, a system such as BullBear(20,20), or a measurement that resolves to a Boolean (true/false) value such as C/C[1]>1.1. Also, an input can be associated with a specific symbol, say SPY, or with the index, group, sector, or sub-sector of the current symbol, as the DL steps down the Focus List. If running a list of symbols, you should normalize so inputs are comparable across symbols, ex., HHV/(HHV-LLV).



Once saved, you can click selectors in the grid to change the Timeframe for an input. Current refers to the timeframe setting from the Period Type setting (usually in the toolbar at the bottom of the OT interface), say Daily, while, say +1, indicates the next higher-domain timeframe, say Weekly.

The Signal Column can be used to limit inputs to only longs or shorts. This is quite useful in conjunction with input analysis, which we will discuss shortly.

The Network Training Settings 4

You can run the DL for long-only, short-only or both signals. Something I think may be unique to Nirvana is the ability to run the DL against a list of symbols, generating a network that is Global across those securities.

This creates a tradeoff. Individual networks will have fewer observations, and I find it easier to do feature selection on a single stock, or at least a list of correlated stocks. Conversely, I find it difficult to create good networks that include lots of non-correlated stocks in a list such as the S&P100. Yet, diversification often does improve overall strategies if you can get high Certainty.

The DL can also use a Dynamic network design, meaning the DL will compile separate networks at each increment of time, say quarterly, using the number of bars specified by the user up to the end of each collection period. This has the benefit of enhancing generalizability but with the tradeoff of a much longer run time.

With similar benefit to generalizability, the DL can use Cluster networks, creating what in ML is called a Committee Machine. The DL generates a collection of identically sized networks, combining their independent certainty scores for an improved prediction. As you would expect, this also multiplies the run time.

The Data Preprocessing Settings 5

Warmup provides a way for the user to explicitly specify when data collection should start. For example, you may want to set this number higher when using indicators with long lags.

Binning is only used on continuous measurements when performing Data Mining. It groups a feature's output into the stated number of smaller divisions. In data science, binning is generally used to reduce the effect of minor observation errors particularly when the amount of data is small. That's not usually a problem with price and indicator data (as they are continuous and voluminous), but it might be helpful for inputs generating discrete data.

The remaining settings determine how observations are scaled. Remember an NN needs the X values to be comparable in dimension.

The Neural Network Settings 6

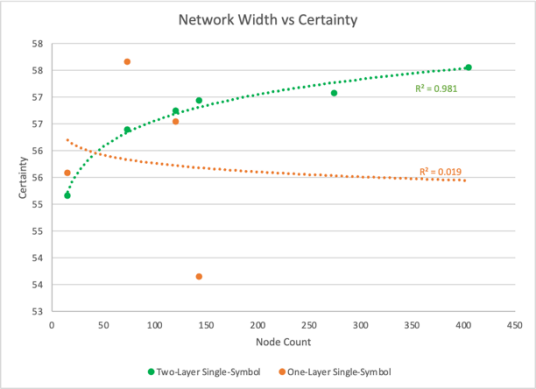
The Show Settings selector is intuitive but easy to overlook; if building a network for both longs and shorts, you need to flip this switch and adjust this group of setting independently.

Neurons per Layer should be a comma-separated string on numbers, for example "80, 60" would specify 80 nodes in the first hidden layer and 60 in the second. The network width (#nodes/layer) and network depth (#layers) can make a big difference so test a few options.

In the ML world, regularization is a technique used to avoid letting a network overfit. In the DL, regularization is implemented using dropout. Check the Dropout box then enter a comma-separated set of number that align with the Neurons Per Layer entry. For our "80, 60" example above, you might enter "0.2, 0.15" to randomly drop 20% of the nodes in the first layer and 15% in the second. Regularization is routinely used in ML even if already using cross-validation as randomly removing some of the nodes forces the network to learn subtle relationships it might not otherwise recognize.

Max Samples will limit the number of observations to be used by the network. One reason might be to shorten processing, but generally leave this as a very high number, say "100000". After a run, the training summary on the initial dialog shows the Number of Collected Samples.

Best Practice: As shown in the accompany graph from one of my own experiments, a two-layer network is far more predictable (better R-Squared) when increasing the node count than an equally sized one-layer network. Also, note how certainty in this two-layer network seems to increase logarithmically with the number of nodes.



I haven't yet tested equally sized three- or four-layer networks. Why? In theory, a two-layer (excluding the input & output layers) network should be able to address any non-linear data relationships, but it may require hundreds of nodes. A network with 20 symbols and 5collecting 6500 samples across 20 inputs theoretically might need about 750 nodes,

say split, “413, 344” over two layers (I usually set the second layer to around 80% of the first). However, three or four layers might be more efficient so you might try several permutations. Of note, the DL block currently only supports about 250 total nodes (depending on the number of samples) albeit that limitation should be removed in a forthcoming software release.

The Convergence Settings 7

Max Iterations sets a limit on the allowed number of epochs. Unless you are doing a quick test of some hyperparameter change, you generally want to set this number high, say 10000, and let the DL reach convergence, usually with far fewer epochs.

You can toggle between two modes for how the DL will determine convergence:

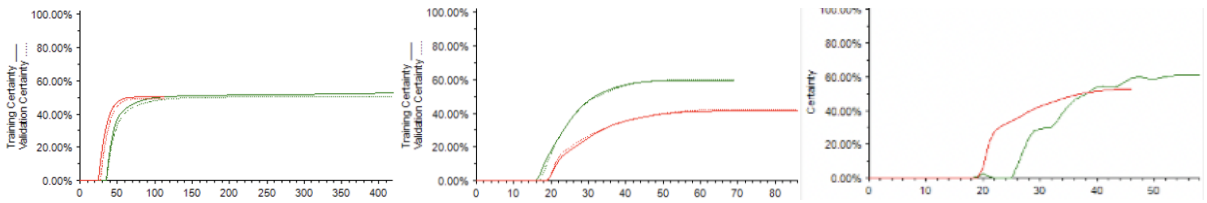
- **Use Training Samples with low/medium/high precision** – this mode looks for the change between successive epochs to be below a fixed percentage of the Certainty Score (actually, it probably uses the MSE directly). Those exact percentages are not exposed, so I usually start with the low setting for the shorter run time while engineering features, and then increase the precision in latter runs.
- **Reserve __% of Samples for Cross-Validation** – this mode reserves the stated percentage of samples to run a separate network and plot a second Certainty line. In this instance, the network is in convergence when the test network and the cross-validation network have both converged.

The Reserve __% of Samples to Calculate Generalization Certainty setting is like cross-validation in that it trains another network using the specified subset of samples. The difference is it gets reported separately on the summary window so the user can assess generalizability.

Best Practice: You always want the certainty graph to show convergence, meaning the curve levels off indicating the MSE is approaching its minimum. All the accompanying examples below show convergence, but the middle graph shows certainty for shorts is significantly lower (higher MSE) than for longs; time to revisit inputs. The third example suggests the training precision was set too low or there were insufficient samples or epochs. Try altering hyperparameters.

It is important to understand the difference between performance of the DL versus performance of the overall strategy. When using the DL as the *signal generator*, I try to get certainty above 60%, 75-85% is better; such high performance is less important when used as a *filter block*. The overall strategy might still show a profitable simulation with a lower accuracy rate but, in such case, the strategy’s performance is due to the downstream signal modifier blocks (filters, tradeplan, etc.) more so than the DL; it raises generalizability concerns. Conversely, a high certainty, say above 90-95%, suggests the network has overfit, which again means the performance probably won’t be generalizable.

Examples of the Certainty Graph 10

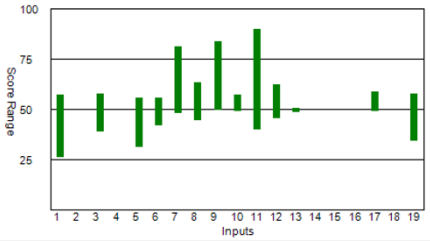


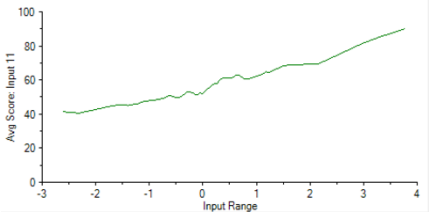
Input Analysis and Pruning 8

After clicking the Analyze button, you can prune (delete) inputs that don't meet certain score thresholds. Pruning non-performant inputs *may* improve the DL's performance (% Certainty) and make the DL run faster. Be aware that an input may function differently for longs than for shorts so be sure to analyze under both settings. Notably, the Input Table has a column labeled Signal where you can limit an input to only longs or shorts.

HOWEVER, this approach to feature engineering is crude. Undoubtably, there will be interactions between inputs that these graphs don't capture. Ultimately, the best way to test the suitability of a particular input is to repeatedly re-run the network to compare the scores with and without that input. Obviously, this is tedious and something you should only do after all other parameters have been finalized, and then only if you have the time to kill. In theory at least, the Filter Settings can achieve much of the same benefit as pruning.

Best Practice: I strive to create inputs, which on the Network Range Graph, show bars that extend above 60% or more, and ideally don't drop much below 50%. Within that group, I may then check the Input Sensitivity Graph on any that drop below 50% then keep only inputs where the line sloops upward or downward as typically found with tall bars. Such a demonstrated correlation between a change in the input value to a change in the score suggests that the input is effective as a predictor. A flat line suggests the input isn't predictive, at least not in isolation.





In the accompanying snippets from one of my runs while building a strategy, the long-side inputs with their top score below 60% were candidates for deleting. Input #9 is ideal while in comparison #11 is somewhat questionable; I therefore ran the Input Sensitivity graph to verify that input #11 looks to be predictive and then ran a copy of my DL where I pruned inputs 1, 3, 10, 13 and 17. I kept 5 and 6 (for both sides) because they had sloping input ranges and were intuitive features. Composite certainty for longs went down slightly from 64.6% to 63.9 but I got an improvement by a factor of 6.5X in the overall strategy's profit-per-trade, presumably by eliminating bad trades rather than more accurate signal generation. So, I kept trying.

The Filter Settings 9

For each timepoint with its set of observations (the X's), the output (Y) from the DL is a numeric Score, quantifying the accuracy of the network's prediction in context of the specified target. Filter settings allow you to specify a minimum threshold, independently for longs and shorts, for passing the score to the next block in the strategy. You can use Nirvana's Strategy Wizard to optimize these settings.

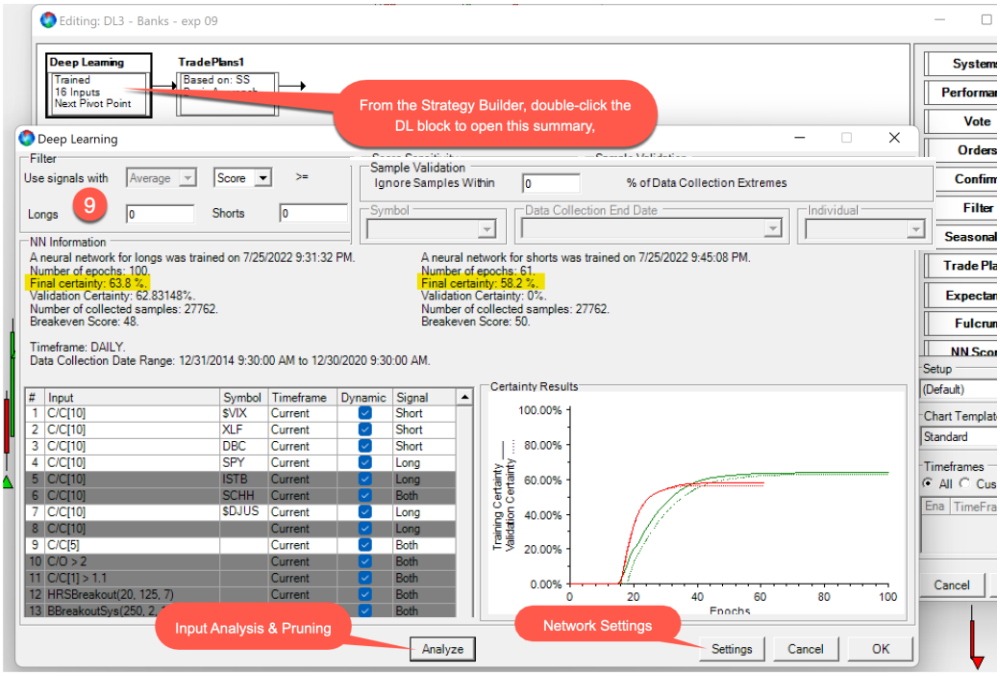
Concluding Thoughts

Alluding again to my auspicious allegory in the introduction, creating an unguided rocket booster that can land safely back on earth takes a lot more than just training a sophisticated deep learning algorithm. Likewise, a well-trained network alone isn't going to conquer the stock market. But as Elon might say, it's a highly promising path to take. And, as my favorite business school professor would add, you only must be 1% better to reap tremendous rewards.

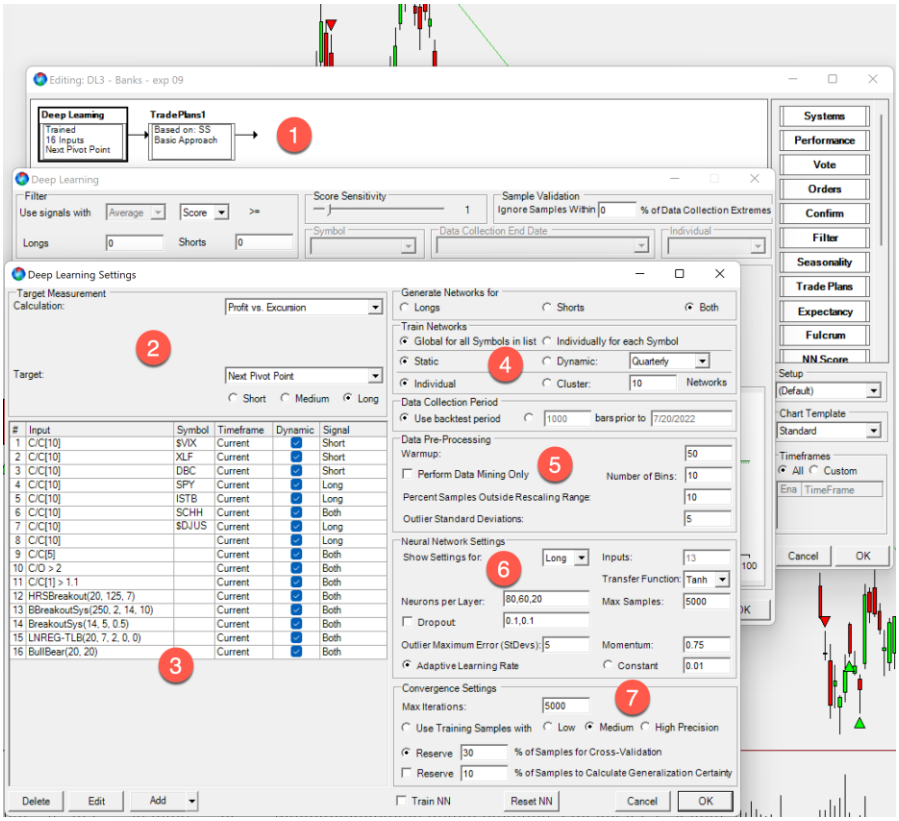
As you may be thinking after reading after this lengthy treatise, building a deep learning system for algorithmic trading can feel like rocket science, even with the wonderful head-start provided by Nirvana's new DL tool. But, sometimes, it's easy. The accompanying example shows a 15% profit improvement for a DL-enhanced strategy (purple line). This sadly un-televised success involved nothing more than pasting a quickly trained DL filter block into an older strategy. A nice reward for only an hour's work!

Here's to hoping I've now impressed Elon enough to give me a few shares of SpaceX.

The Summary Window



The Settings Window



The Analyze Window