Adaptive Cognition:
Using Genetic Algorithms to Forecast
Financial Markets

Prepared by: Animesh Koratana
Northview High School — 10th grade

RATIONALE

In the field of artificial intelligence, genetic algorithms have long prevailed as search heuristics utilized to gain a maximal reach within large search spaces. This reach was obtained using the process of natural selection which is observed in nature. Mimicry of these natural processes allow a genetic algorithm to converge quickly at a locally optimum answer within an extremely large search spaces. Unlike artificial neural networks (ANNs), designed to function like neurons in the brain, these algorithms utilize the concepts of natural selection to determine the best solution for a problem. As a result, genetic algorithms are great contenders to optimize sets of parameters to obtain a desired feedback measure.

Financial markets are often considered highly volatile entities. Although this is true to some degree, it can be safely assumed that at any given point, if a specific stock is backtracked and its parameters are observed and evolved to an optimum measure, there will be some level of intelligence that can be offered. However, it has been observed that no single parameter affects a days stock price directly. This observation only heightens need to employ genetic algorithms to offer multivariable optimization that wouldn’t be computationally possible using brute-force methods. Genetic algorithms’ efficiency in quickly making its way through large search spaces has been proved with repeated study — and this is the very reason genetic algorithms serve to be perfect contenders for this combinational optimization problem.
PROBLEM & HYPOTHESIS
Traditionally, prediction engines used to the forecast financial markets have employed deep-learning or feed-forward neural networks to optimize weights and derive an optimal prediction. The fallacy in these solutions lies in neural networks' weighty activation functions and permutationally many parameterization functions. These parameters require thousands of training epochs before a back propagation entity can completely optimize a single function. The neural network has be locally optimized at each layer and node before a parameter is optimized and this process has to be iterated through for each parameter. Naturally, with an increasing number of input parameters, the search space for a completely optimized network increases permutationally — and neural networks are not designed to optimize for large search spaces consisting of many input parameters. In the case of a new pattern, neural networks have been found to be slow to react and reclassify weights for the same parameters disregarding the addition of new parameters.

In an effort to create a highly adaptable entity it is necessary to employ more evolutionary based concepts. This approach reduces dimensionality of the feature space and enhances the generalizability of the classifier in parameterization. The difficulty in employing these methods lies in encoding the genetic algorithm to optimize the feature discretization of each input and the aggregation the discretized inputs to collectively make a prediction. More than anything, the implementation of an evolutionary based system will be an encoding problem rather than a parameterization problem like in a neural network’s initialization. Through a system of repeated dispersion, analysis, and aggregation a genetic algorithm will be able to predict the futures of any given financial market with superior adaptability and more efficient parameterization. A light-weight algorithm such as this will be able to adapt to new patterns witnessed in the market much more quickly and ultimately make more viable decisions on the futures of financial markets.

SYNOPSIS
We have laid out a foundation upon which a projective population of lightweight neural networks will make independent decisions on any selected financial market. A set of oscillators (each with thresholds, weights, and direction) will allow the chromosome to react to different variations in its input parameters. Each neural network will be representative of a trading philosophy and will be optimized for a singular prediction function. The maintenance of such a static prediction function throughout the population will bring the optimization to the individual weights, thresholds, and directions for each oscillator. With each added parameter, the overall complexity permutationally increases and this sacrifices adaptability of the population in its simulated evolution. To avoid the problem of overfitting, a hybrid of the gradient descent algorithm commonly used to optimize weights in deep learning algorithms known as backpropagation will be used as an evolutionary strategy. This will effectively serve as a feedback mechanism and allow the weights to be minutely varied after each generation.

The input parameters that are being used are: 50 day moving average, momentum, beta, covariance, on balance volume, and senti-
ment. All of the input parameters except sentiment are technical analysis indicators that can be mathematically derived from quantitative data.

After relevant content from Google News and Twitter will be extracted from the internet, a recursive neural tensor network will apply what Pitler calls, an “exact, graph-based algorithm for non-projective parsing with higher order features” (2014). This algorithm will reference the Stanford NLP Tree Bank to calculate a dependency tree for each sentence from the content. This dependency tree will outline the contextual relationships between different words within the sentence. Therefore, a sentence can then be manipulated as an idea rather than a set of words and hierarchical importance can be given to different parts or clauses of the sentence. An overall sentiment score will be calculated for each sentence and will be aggregated up to a topic level while being normalized in every layer of aggregation to minimize the effect of writing style. Multiple topics will be run in parallel pertaining to the sector, industry, and actual stock of the company; each topic’s unweighted variance with the close prices will serve as a normalizing factor in the final sentiment score.

Using the defined six input parameters, each chromosome will predict a vector including magnitude and direction of the change in close price for the next reading frame. The genetic algorithm will optimize weights in trading philosophies and still maintain diversity in the chromosomes through bi-linear crossover and creep mutation. This diversity will propagate adaptability in the population.

GENETIC ALGORITHMS

A genetic algorithm (GA) is a search technique used in computing to find exact or locally optimum solutions for a defined optimization problem. Genetic algorithms are a particular class of evolutionary computation that uses techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover. In a population of animals or plants, a new individual is generated by the crossover of the genetic information of two parents. A genetic algorithm finds the potential solution to a specific problem as a simple chromosome like data structure so as to preserve the critical information.

Chromosome

An individual chromosome in this scenario could colloquially be defined as a trading philosophy. A trading philosophy would consist of a set of oscillators for a set of predefined and provided inputs and reactions for each of these values.

For this experiment, an oscillator would be defined as:

- Weight \((w)\) - where \(0.0<w<1.0\) and determines importance of the variable in decision making
- Threshold \((t)\) - a real number that pertains to its respective indicator and acts as a cutoff to discretize a relative high or low
- Direction \((d)\) - that the chromosome favors for the direction of that variable

The chromosome would also consist of a summatory threshold which would function similar to an oscillator, but would instead react to a weighted average of the weights in any given instance of time. The reaction of this summatory
threshold would then allow the chromosome to decide the future direction of the stock and later make a prediction.

**Figure 1 - Chromosome Matrix Representation**

\[
\text{chromosome}_i = \begin{bmatrix}
    v_1 \\
    v_2 \\
    v_n
\end{bmatrix}
\begin{bmatrix}
    w_1 & t_1 & d_1 \\
    w_2 & t_2 & d_2 \\
    w_n & t_n & d_n
\end{bmatrix}
\]

**Bilinear Crossover**

The crossover is a genetic operator used to vary chromosome gene structure where gene information is interchanged between selected parents by selecting two points in the gene structure of each parent. This ensures the conglomeration of genetic material perpetuating the search for an optimal solution within a population. For the sake of crossover, a chromosome will be represented as a \( n \times 3 \) matrix of \( n \) oscillators. Each oscillator will contain a threshold \( t \) (a cutoff), a respective weight \( w \) (an importance), and a direction to favor \( d \) (is a higher value better). The method of crossover remains fairly constant regardless of the problem and scope. Crossover is achieved by first selecting a crossover point within a pair of defined and unique organisms \( P_1 \) and \( P_2 \) (which are the equivalent of parents for the crossed over parent). The chromosomes are then split at the selected crossover point. The second half of \( P_2 \) \((P_2H_2)\) is then appended to the first half of \( P_1 \) \((P_1H_1)\) to make one child chromosome \((C_1)\). The second child \((C_2)\) is made by appending the second half of \( P_1 \) \((P_1H_2)\) to the first half of \( P_2 \) \((P_2H_1)\). There are bound to be duplicate nodes in each of the halves, so, in that case it is necessary to take a random node from the graph that isn’t present in either of the halves of the chromosomes that are to be crossed over. Below is an example of two parent chromosomes being crossed over:

**Figure 2 - Crossover - note: can be initiated from any division within the matrix as long as both parents are split along the same axis**

**Mutation**

During the progression of a genetic algorithm, the population can hit a local optima (or extrema). Nature copes for this local optima by adding random genetic diversity to the population set “every-so-often” with the help of mutation. The creep mutation used works by adding a small value to each gene with probability \( p \). The selection method used to select the population is roulette wheel selection. In this method the fitness assigned to each individual is used for the selection process. This fitness is used to associate a probability selection with each individual. Where \( f_i \) is the fitness of the \( i^{th} \) individual and \( N \) is the population size. This can be given as below:

**Figure 3 - Creep Mutation Function (Liang, 2008)**

\[
P_i = \frac{f_i}{\sum_{j=1}^{N} f_j}
\]
Inverse Linear Roulette Selection

In every generation, a selection agent comes to play which sifts out the fit chromosomes from the unfit chromosomes. The selection agent “kills off” a user specified percentage of organisms in the population. However, it is under the discretion of the selection agent in determining which chromosomes to kill. The selection agent in this scenario will determine fitness of chromosomes in an individual chromosome both by examining percentage error and if the correct direction was predicted. The agent will initially classify each entity into two groups of “correct direction predicted” and a ‘wrong direction predicted’. 30% of the chromosomes from the first group will be selected through inverse linear roulette wheel selection and 20% of the wrong direction predicted group will be chosen using the same selection method. In inverse linear roulette selection each individual is giving the probability $P_i$ of being selected. $P_i$ is linearly proportional to the fitness of the organism. The probability of each organism being selected decreases with higher weight making weight inversely proportional to the probability of being selected. The algorithm to determine probability for any certain organism can be seen in the equation below in which $f_i$ represents the fitness function of for the organism $i$. This method allows for the maintenance of a higher genetic diversity in the fact that the top chromosome doesn’t necessarily have a 100% chance of survival year over year and the most expensive or most unfit chromosome doesn’t necessarily have a 100% chance of death year over year. This opens the door to new crossovers that may lead to better paths and a further reach in the undeterministic sample space.

Figure 4 - Inverse Linear Roulette Selection function (Kordon, 2004)

$\frac{1}{N-1} \times \left(1 - \frac{f_i}{\sum_{j \in \text{Population}} f_j}\right)$

Prediction

The process of incorporating different aspects of a trading philosophy to result in a predicted close price will involve a multitude of layers of dispersion, aggregation, and later summation to produce a single output: similar to an ANN’s system of propagating an output. In an effort to keep the system and population adaptable to the slightest changes in market, it will be necessary to perceive each chromosome as completely independent, lightweight feed-forward neural network. The neural network would consist of a single function allowing the weights to be optimized to fit the particular function. The maintenance of a single function neural network chromosome would open the door for the entirety of a population to parameterize its weights to optimize the philosophy (weights and thresholds inclusive) rather than the functions to get there. This approach will therefore ensure adaptability in the populations—not necessarily an individual chromosome’s decision making process. A highly adaptable solution such as this would result in a problem known as overfitting, which specifically occurs in noisy data situations. It has been observed that excessive training results in decreased generalization. Instead of finding general
properties of the different input patterns that match to a certain output, the training brings the network closer to each of the given input patterns (Kordon, 2004). This results in less tolerance in dealing with new patterns. To negate the effect of overfitting and focus the progression of the population on a global search, a system of using backpropagation as an evolutionary mechanism will be used.

**Backpropagation as an Evolutionary Strategy**

Backpropagation is essentially a gradient descent algorithm. It’s commonly used in the application of deep learning artificial neural networks and feedforward neural networks. However, its ultimate goal is to improve the performance of the entity by reducing its error along a gradient by modifying weights. Therefore, a modified backpropagation agent to a chromosome could be applied as an evolutionary strategy to make minute changes generation over generation. In backpropagation, error is expressed by the root-mean-square error (RMS), which can be calculated by:

\[
E = \frac{1}{2} \sum_p \| t_p - o_p \|^2
\]

The error \(E\) is the half the sum of the geometric averages of the difference between projected target \(t\) and the actual output \(o\) vector over all patterns \(p\). For each generation, rather than epoch, the weights \(w\) will be adjusted towards the direction of maximum decrease. The magnitude of this change will be scaled by a learning rate lambda (D. Whitley). Therefore, a new weight can be calculated using a simple addition or subtraction methods. Movement along the derivative of the sigmoid will no longer be necessary because the chromosome will no longer utilize hidden nodes to augment the parameterized input nodes.

\[
\nabla E = \left( \frac{\delta E}{\delta w_1}, \frac{\delta E}{\delta w_2}, \ldots, \frac{\delta E}{\delta w_n} \right)
\]

Using this approach to evolutionary strategies will allow targeted control and optimization of weights and will therefore minimize the overfitting of the chromosome to a particular dataset.

**Feature Calculation and Discretization: Technical Analysis**

Representative indicators through technical analysis of a particular ticker will be provided to each chromosome to allow for reactions and ultimately predictions to be calculated. Each chromosome will have trained reactions for each discretized parameter. The parameters will remain consistent through each chromosome. Each of the selected technical indicators are representative of different aspects of a stock. It is necessary to minimize the number of indicators for each chromosome to keep complexity in decision-making minuscule and to maximize the
parameterized reaction for each value. Therefore, each indicator must be carefully chosen and be representative of a different aspect of a market.

**50 DAY MOVING AVERAGE**

A 50 day moving average is an indicator in technical analysis that helps smooth out price action by filtering out the “noise” from random price fluctuations. A moving average (MA) is a trend-following or lagging indicator because it is based on past prices. A simple moving average (SMA) can be calculated by:

\[
SMA = \frac{p_M + p_{M-1} + \cdots + p_{M-(n-1)}}{n}
\]

**MOMENTUM**

Momentum is a simple technical analysis indicator. It is the measurement of the speed or velocity of price changes and is the difference between today’s closing price and the close \(N\) days ago. Momentum can define volatility and may be instrumental in favoring more modest trading philosophies or chromosomes. Momentum can be calculated through:

\[
momentum = close_{today} - close_{N \text{ days ago}}
\]

**BETA**

Beta is a measure of the volatility, or systematic risk, of a security or a portfolio in comparison to the market as a whole (S&P index). The beta coefficient is often used in capital asset pricing model and in unison with alpha to assess risk. And you can think of beta as the tendency of a security’s returns to respond to swings in the market (Pressly, 1989). A beta of 1 indicates that the security’s price will move with the market. A beta of less than 1 means that the security will be less volatile than the market. A beta of greater than 1 indicates that the security's price will be more volatile than the market. Beta is calculated using regression analysis. Simply, the beta is quotient of the covariance and variance. The following equation calculates beta between two matrices \(i\) and \(j\).

\[
\beta = \frac{\sum_{i,j<i} a_i a_j \sigma(X_i, X_j)}{\sigma^2}
\]

**COVARIANCE**

A measure of the degree to which returns on two risky assets move in tandem (S&P index with the close prices of the ticker). A positive covariance means that asset returns move together. A negative covariance means returns move inversely. Covariance can be used to make decisions based off of data that may not completely pertain to the specific ticker. It weighs the information aggregated pertaining to other sources. \(q_{jk}\) is an estimate of the covariance between the \(j^{th}\) variable and the \(k^{th}\) variable and can be calculated by:

\[
q_{jk} = \frac{1}{N - 1} \sum_{i=1}^{N} (x_{ij} - \bar{x}_j) (x_{ik} - \bar{x}_k)
\]
ON BALANCE VOLUME

On balance volume (OBV) is another momentum indicator that uses volume flow to predict changes in stock price (Pai, 2005). On balance volume is an indicator that is affected when mutual funds or other large entities make large trades for any reason. Differences in on balance volumes can provide insight into futures of a stock in the short term and the long term. A higher OBV may also be an indicator of a more volatile ticker. OBV can be calculated by:

Figure 12 - OBV Calculation Function (Pai, 2005)

\[ OBV = \text{Previous OBV} + \text{Current Period Volume} \]

SENTIMENT

Calculating sentiment as a parameter is another nondeterministic problem that will require the aggregation, parsing, and analysis of natural language in large quantities. Sentiment analysis lies in the domain of natural language processing. To accurately process data from articles, tweets, and other unstructured data, a multitude of steps to factor in different structures in language. Initially, for each ticker, 4 separate sentiment topics will be created, each pertaining to different areas of a stock:

- Google News Stock Topic - recent google news articles pertaining to the stock
- Twitter Stock Topic - consisting of recent tweets people made about the particular company
- Google News Sector Topic - recent google news articles pertaining to the company’s sector
- Google News Industry Topic - recent google news articles pertaining to the company’s industry

After aggregating the respective sector and industry of the ticker, it will be necessary to find pertaining URL’s for the articles and/or the respective tweets. After scraping the content from all of these content pointers, the processing of the unstructured content can begin.

Traditional approaches to sentiment analysis matches keywords in an article to a vocabulary consisting of words and its respective strength and sentiment. A weighted mean of the sentiment scores for each of the keywords makes up the sentiment score of the unstructured entity. However, this linear approach fails to account for the modularity of language and favors colloquial speech over more complex language. The “keyword” approach negates the relationships between words and language structures that can contextually change the overall meaning of a sentence. Therefore, to best analyze sentiment of an entity, scores must be calculated in the frame of a sentence rather than a word. This approach will allow for a more holistic approach to sentiment analysis.

In order to capture the compositional effects with higher accuracy, we propose a new sub-model called the Recursive Neural Tensor Network (RNTN) (Sutskever, 2009); they represent a phrase through word vectors and a parse tree and then compute vectors for higher nodes in the tree using the same tensor-based composition function (Zettlemoyer and Collins, 2005). Variants of this idea use more complex frequencies such as how often a word appears in a certain syntactic context (Satta and Kehlmann,
Each sentence will be mapped to a logical form using the Stanford NLP treebank.

**Sentence Dependency Parsing**

A sentence's logical form will consist of an inter-dependent tree of tokenized words. This model will compute compositional vector representations for phrases of variable length and syntactic type (Socher, 2013). The vector’s error to propagate the RNTN can be defined as:

Figure 12 - Vector Error Propagation Function in RNTN (Klein and Manning, 2003)

\[ E(\theta) = \sum_i \sum_j t^i_j \log y^i_j + \lambda ||\theta||^2 \]

The derivative for the weights of the softmax classifier are standard and simply sum up from each node’s error. We define \( x^i \) to be the vector at node \( i \). We skip the standard derivative for \( W_s \).

Each node "backpropagates" its error through to the recursively used weights \( V, W \) (Socher, 2013). This method of backpropagation is independent of the GA’s backpropagation. Let \( \delta^i,s \in R_{d \times 1} \) be the softmax error vector at node \( i \) (Socher, 2013):

\[ \delta^i,s = (W_s^T (y^i - t^i)) \odot f'(x^i) \]

**HIGH LEVEL NEGATION**

These vectors can be reorganized to form relational trees of tokenized words. With this relational tree, it will be possible to negate portions of a sentence in context relative to another portion of a sentence. For example, in a sentence consisting of two independent clauses separated by a “however,” a dominance should be attributed to sentiment of the clause following the
conjunction. Variations in natural language like these minutely vary the overall meaning of each sentence but may affect the overall sentiment of a particular sentence much more. The keyword dependent system would be heavily skewed in the presence of sarcasm and other complex linguistic systems.

A relational tree constructed of the tokenized words would also allow for negation based on hierarchy of the word. Words with a higher relative location in the sentiment tree would be assigned higher weights in determining the overall sentiment of the sentence.

The sentiment score for each topic will be calculated from the bottom up from each sentence. Article’s total sentiment score would be the normalized and weighted average of the sentences that constitute the article. Each topic will then be normalized and weighted relative to the variance between the close price of the ticker and the unweighted sentiment score of the topic. Therefore the equation to calculate the sentiment of a topic would be:

\[
S_{\text{topic}} = \sigma^2 \sum_{i=0}^{n} \left( \frac{W_i}{\|\delta_{i,w}\|^2} \right) \]

**PREDICTION ALGORITHM**

The overall equation for a predicted close price is the following where \( f \) is the frame that is being predicted, \( o \) is the number of oscillators, and \( w_o \) is the parameterized weight attributed to the oscillator. The direction (to add or subtract from the previous frame’s close) is determined by the threshold of the chromosome. A prediction will effectively be a vector determining magnitude and direction of the change in close price of the next reading frame.

\[
predicted_f = \text{close}_{f-1} \pm \sum_{i=0}^{o} \left( \frac{w_i}{o} \right)\]

The weight \( w \) is determined by reactions (thresholds) to the actual values of the technical analysis variables. This lightweight equation allows the entire population to adapt to optimize its reaction weights to minimize percentage error in prediction of close prices.

**ALGORITHM**

1. Initialize population of \( n \) number of random chromosomes.
2. Make predictions
3. Go to next reading frame
4. Aggregate data and calculate technical analysis variables
5. Aggregate non-obsolete sentiment content from twitter and google news
6. Calculate percent error
7. Run natural selection agent
8. Randomly mutate a chromosome in the population
9. Sort population by percent error and determine top chromosome.
10. Use the top chromosome to make a prediction about the next reading frame’s close price.
1. RMS Error:

2. Vector Weight Error:

3. Weight Change:

Initialize population of \(n\) (a million) chromosomes

Get data to feed population:
- S&P Index
- Close Price
- Open Price
- Trading Volume

{Yahoo Finance YQL API}

Calculate Technical Analysis Variables:
- Moving Average
- Beta
- Covariance
- Momentum
- On Balance Volume

Calculate Sentiment

Feed Calculated and Discretized Data To Chromosomes

Make Predictions

Calculate Error

Backpropagate Weights

1. RMS Error:

2. Vector Weight Error:

3. Weight Change:

Pick Top Chromosome to Predict Next Frame (R Price)

"Kill off" Unfit Chromosomes (with high error)

Crossover & Mutate

Aggregate Data From Twitter and Google News for Ticker, Sector & Industry

Annotate & Parse Dependencies

Calculate Weighted Sentiment Score

Calculate final Score (s) by weighing each sentiment score with the covariance between price and score

MovingAverage = \(\alpha\)

Beta = \(\beta\)

Covariance = \(c\)

Momentum = \(m\)

OnBalanceVolume = \(v\)

Sentiment = \(s\)

\[ w_{0, f} t_{0} d_{i} \]

\[ w_{1, f} t_{1} d_{i} \]

\[ w_{2, f} t_{2} d_{i} \]

\[ w_{3, f} t_{3} d_{i} \]

NOTE: An Oscillator can be defined as a 1x3 array pertaining to a particular variable consisting of a weight \(w_0\), threshold \(t_0\), and direction \(d_0\).
EXPERIMENTS

Experiments will be conducted to test the responsiveness of the population to pattern changes. This responsiveness or adaptability will be compared to a traditional convolutional deep learning neural nets (CDNN) developed by the Neuroph project (The libraries and code to implement that can be found at: [http://neuroph.sourceforge.net/tutorials/StockMarketPrediction-Tutorial.html](http://neuroph.sourceforge.net/tutorials/StockMarketPrediction-Tutorial.html)). Different time frame lengths will be tested to find the optimal time frame which best utilizes the sensitivity of the genetic algorithm to pattern changes.

The experiments will be set up as a comparison to a traditional deep learning neural network that takes in the same 6 parameters, including sentiment. The population in each trial will be a randomly initialized population of 1,000,000 unique chromosomes. The experiments will be repeated for four different tickers: AAPL, CAMP, XOM, and MBLY. These four tickers are representative of different sectors and also have different dynamics in volatility and risk. A combination of the following three experiments will give insight into the true adaptability, convergence, and pattern learning abilities of the algorithm.

**Convergence**

This experiment will test the speed in convergence or training of the genetic algorithm based system. A converged population will be determined when the variance in error is maintained under 3% for 10 cycles. Each frame will be defined as 1 day initially and therefore a higher percent error than with low latency trading will be acceptable. The data set that will be used will be 5 years of stock information aggregated through the Yahoo YQL data tables for the tickers AAPL and XOM. The speed in convergence to the non convex optima will be recorded for 10 different trials for each ticker.

**Adaptability**

This experiment will test the adaptability of the population to minute variations in the stock market. The two systems will be run concurrently in low latency situations where each frame is 5 minutes long. The number of cycles needed in convergence after relatively large changes will be recorded. One cycle in a genetic algorithm will be one generation and one cycle in the neural network will be one epoch. Each system will be given 20 cycles to train its chromosomes or nodes respectively before data will start being collected. In essence, this experiment will test the adaptability of the genetic algorithm based system of modular parametrization compared to the functional parametrization in neural networks. The experiment will be repeated with varying time frames: 1 month, and 3 months.

**Pattern Fitting**

This experiment will test the system's ability to react to patterns that it has never seen before. Each system will be initialized on the data from September 9, 2008 (1 week from the financial crash). Each system's weights for each of the indicators and variables will be monitored.

---

1 A type of feed-forward artificial neural network where the individual neurons are tiled in such a way that they respond to overlapping regions in the visual field. They are commonly used to analyze patterns and therefore are applied to stock predicting algorithms.
through the crash along with the overall percent error. The 2008 crash was a pattern that the world had never witnessed, and therefore, no entity could have been trained to react to such a pattern before. The results of this experiment will give an insight into the true adaptability of a GA based algorithm in optimizing trading decisions.

MODEL ANALYSIS

The following categories are disparate conclusions that can offer insights into targeted aspects of the GA:

Non-convex Convergence & Adaptation

In a system where fitness is not static, the gradient upon which a population must descend is a volatile entity. Where the frame was a day, the CDNN took 81 epochs to converge whereas the GA based system converged in 19 generations.

Figure 17 - Convergence Speeds when Frame = One Day

<table>
<thead>
<tr>
<th>Model</th>
<th>$\mu_{error}$ (%)</th>
<th>$\sigma^2_{error}$ (%)</th>
<th>Cycles in Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDNN</td>
<td>0.1</td>
<td>1.4</td>
<td>81</td>
</tr>
<tr>
<td>GA</td>
<td>$3.72 \times 10^{-3}$</td>
<td>$1.42 \times 10^{-2}$</td>
<td>19</td>
</tr>
</tbody>
</table>

Figure 18 - Average error (%) in varying Frame Lengths

<table>
<thead>
<tr>
<th>Model</th>
<th>5 minutes</th>
<th>1 day</th>
<th>1 month</th>
<th>3 months</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDNN</td>
<td>0.1</td>
<td>1.4</td>
<td>7.9</td>
<td>11.4</td>
</tr>
<tr>
<td>GA</td>
<td>$3.72 \times 10^{-3}$</td>
<td>$1.4 \times 10^{-2}$</td>
<td>4.2</td>
<td>10.1</td>
</tr>
</tbody>
</table>

Although convergence can be modeled after several factors, it is worth noticing the significant difference in convergence speed of the GA based algorithm over the CDNN. The gradient can never be simplified down to a single vector at any particular point, and therefore error propagation modeled in figure 19 would be erratic without the normalization of the figure.

Figure 19 - Convergence of GA vs. CDNN Over Multiple Cycles

The GA's based system seemed to be most receptive to current patterns in the context of a few frames. This effect can likely be attributed to the basic nature of genetic algorithms where all importance is given to fitness in that particular time frame. Although in a convolutional deep learning neural network importance is given to learning to a degree, the stratification of patterns is given preference. And this stratification is likely what causes the lesser magnitude of change in error when the frame is increased. The speed of convergence also acted as a constant of complete re-convergence (c)—meaning the system would refactor all of its weights for every time c number of frames would run.

A similar pattern was observed in low latency trading scenarios. For every c frames, the weights would become completely refactored to fit the changes of that instance. Since in low latency situations, the frame of pattern recognition does not exceed the frame of a few hours, the chromosomes were quick to adapt. In this context, the significantly lower percent error in GAs
can be attributed to the backpropagation entity optimizing the weights in real time. In trading, time is always of the essence, and the creation of new genetic material into which patterns are encoded takes many generations. Backpropagation introduced relevant genetic material to minutely tweak the output vector of each chromosome. This increased the context sensitivity of each chromosome by a factor of 2.2 (learning rate, lambda) to prevent the overfitting of the chromosomes to a situation.

**Pattern Propagation**

No backpropagation entity can possibly train reactions for any new pattern. So in the event of any pattern, it is a matter of adaptation and learning rate. In a neural network, learning rate is usually parametrized as a number; however, in the context of a GA, the learning rate is directly proportional to the genetic diversity and ultimately population size. Therefore, a GA based weight propagation system is inherently more adaptable than any functional neural network.

An experiment with the current system initialized in the context of 1 week before the 2008 stock market crash highlights this forte in adaptability. At the point of the crash, both systems were completely converged.

**Figure 20 - Post Crash System Statistics**

- Frame is 5 minutes

<table>
<thead>
<tr>
<th>Model</th>
<th>$\mu_{\text{error}}$ (%)</th>
<th>$\sigma^2_{\text{error}}$ (%)</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDNN</td>
<td>22.7%</td>
<td>6.7%</td>
<td>103</td>
</tr>
<tr>
<td>GA</td>
<td>0.84</td>
<td>2.1</td>
<td>8</td>
</tr>
</tbody>
</table>

**Figure 22 - Weight Refactoring During Hours of Crash in CDNN (September 16, 2008)**

---

Model $\mu_{\text{error}}$($\%$) $\sigma^2_{\text{error}}$($\%$) Convergence

| CDNN | 22.7% | 6.7% | 103 |
| GA   | 0.84  | 2.1  | 8   |

**Figure 22 - Weight Refactoring During Hours Of Crash in GA (September 16, 2008)**
A deliberate and “human-like” shift in weight can be witnessed in the GA based method. For example, the importance given to support indicators such as the moving average dropped consistently through 4:00 PM. At the same time, weights for volatility indexes rose and converged at a point. These changes would be expected in a human trading philosophy, but the modification of individual weights in reaction to a new pattern is difficult in the case of a CDNN. The CDNN reacted much slower to this new pattern. Its initial reaction was to reduce the weights for all of the parameters in order to gradually converge at a higher optima. Therefore, the CDNN was not able to react by the end of the day. The targeted modification of weights in the GA in new situations is due to the back propagation entity serving as an evolutionary strategy and the optimization of the population to a single activation function. Together, these two characteristics of our GA based system maintained an inherent state of lesser complexity.

CONCLUSION & DISCUSSION

It is necessary to realize that the approach GAs take in optimization is by definition more adaptable than that of a neural network. A GA will prioritize speed in convergence over accuracy whereas a neural network will prioritize accuracy in parametrization. The model that was created in this experiment can be classified as a highly adaptable entity best suited for low latency trading in volatile markets. A common theme emerges when comparing the effects of these various genetic operators in the large scheme of non-deterministic problems—genetic diversity. The operators that maintained genetic diversity prevailed over the ones that conglomerated to the locally apt answer. This highlights one of the key principles of genetic algorithms; the focus generation over generation isn’t the top vector — its the population. In a finite sample space, the initial population represents a negligible yet tangible fraction of the sample space. The intent of any genetic algorithm is to take that initial population and extend its reaches to the other areas of the sample space and explore it. It is easy to facilitate a seemingly correct answer and tweak it to make it a little better using evolutionary strategies — but in a game of enormous numbers, it is more than likely that a magnitude of a better answer is available. Further study would include increasing the contextual sensitivity of the algorithm. Contextual sensitivity is the differentiating factor between neural networks and GA based systems. Using back propagation as an evolutionary strategy in this scenario helped increase the GA’s sensitivity to patterns within its convergence frame. So far, context sensitivity has been static and directly proportional to the convergence frame of the model. If these two factors were to be separated and sensitivity were to be determined in a change sensitive basis, the reach of the GA based system would be extrapolated tremendously.

The near future entails experimentation in oscillators and making the model make decisions off of different trends. No financial market is independent of other factors in its proximity and adding a correlational ability will allow the model to analyze repercussions of changes across financial markets. In this experiment, correlation was analyzed between the market and the S&P price. A system of adding more dependencies while maintaining adaptability will have to be implemented.
REFERENCES


I. Sutskever, R. Salakhutdinov, and J. B. Tenenbaum. 2009. Modelling relational data using Bayesian clustered tensor factorization. In NIPS.


