# Variable Hidden Layer Sizing in Elman Recurrent Neuro-Evolution

**Khosrow Kaikhah and Ryan Garlick**
Department of Computer Science
Southwest Texas State University
San Marcos, TX 78666
kk02@swt.edu, ryang@seas.smu.edu

## Abstract

The relationship between the size of the hidden layer in a neural network and performance in a particular domain is currently an open research issue. Often, the number of neurons in the hidden layer is chosen empirically and subsequently fixed for the training of the network. Fixing the size of the hidden layer limits an inherent strength of neural networks – the ability to generalize experiences from one situation to another, to adapt to new situations, and to overcome the "brittleness" often associated with traditional artificial intelligence techniques. This paper proposes an evolutionary algorithm to search for network sizes along with weights and connections between neurons. The size of the networks simply becomes another search parameter for the evolutionary algorithm.

This research builds upon the neuro-evolution tool SANE, developed by David Moriarty. SANE evolves neurons and networks simultaneously, and is modified in this work in several ways, including varying the hidden layer size, and evolving Elman recurrent neural networks for non-Markovian tasks. These modifications allow the evolution of better performing and more consistent networks, and do so more efficiently and faster.

SANE, modified with variable network sizing, learns to play modified casino blackjack and develops a successful card counting strategy. The contributions of this research are up to 8.34% performance increases over fixed hidden layer size models while reducing hidden layer processing time by almost 10%, and a faster, more autonomous approach to the scaling of neuro-evolutionary techniques to solving larger and more difficult problems.

## 1. Introduction

Neural networks have proven effective in a range of pattern recognition and association problems, and generalize well to new situations, often overcoming the "brittleness" of some other traditional artificial intelligence methods. Evolutionary or genetic algorithms have been applied to training neural networks, with the neuro-evolution approach significant in its ability to discover difficult, counter-intuitive strategies [2]. Evolutionary algorithms represent a candidate solution as a chromosome, and these potential solutions are evaluated and the operations of crossover and mutation are performed on them in a hill-climbing search for better solutions. A critical aspect of evolutionary algorithms is maintaining diversity in the population, thus preventing the algorithm from falling into a local optimum and converging too early upon a sub-optimal solution.

The hybrid neural and genetic approach takes advantage of the strengths of both. By training neural networks with evolutionary algorithms, feedback from the environment does not have to occur at every network output, and a decision strategy can be based upon the evaluation of an entire series of decisions.

Often, neural networks are evolved with a fixed size, with the genetic algorithm adjusting connection weights to optimize network performance. Choosing the correct size or number of neurons in the hidden layer for a neural network is problem dependent, and is currently an open research issue. Commonly, networks are tested using different sized models, and size is chosen empirically. This paper presents a new approach to neuro-evolution, treating the size of the network as another parameter in the evolutionary algorithm. This approach allows the network to grow in response to shifts in the problem, or more efficiently form a smaller network if this solution is more appropriate.

This paper demonstrates a new method of efficiently automating the search for appropriate network size, with performance and efficiency increases due to increases in network population diversity, creation of Elman recurrent neural networks with SANE, and direction toward the goal of a more autonomous learning system.

The remainder of this paper is organized as follows: Chapter 2 introduces neuro-evolution and a precursor to the SANE system. Chapter 3 is devoted to SANE, with Chapter 4 addressing modifications to SANE to vary the hidden layer size and permit Elman recurrent networks. Chapter 5 discusses the application of SANE to blackjack. Chapter 6 gives the results of blackjack experiments on these recurrent models. Chapter 7 presents some brief comments on the experiments, with a conclusion and future work in Chapter 8.

## 2. Neuro-Evolution Models

A primary advantage of hybrid neuro-evolution searches over more traditional gradient-descent searches is the ability to implement reinforcement learning rather than supervised learning. Supervised learning methods (such as backpropagation) require a smooth, continuously differentiable activation function from which gradient information can be derived for the backpropagation of error signals for every iteration of the network. For training purposes, the network must receive feedback as to its performance after every output. In many domains, this output may not come until a sequence of events has occurred. Training a neural network using backpropagation or other supervised learning methods to perform a sequential decision task requires a determination of which specific decisions were responsible for any errors based upon an evaluation of a series of such decisions. This is Minsky’s credit assignment problem, and is at the center of many AI problems [1].

Reinforcement learning addresses the credit assignment problem by assigning a performance measure to an entire system, even after several decisions have been made. A poker player, for example, could be evaluated after several hands have been played, and not by a statistical analysis of precisely the correct decision to take another card or stay at every possible decision point. In poker, this analysis could be an estimation only, as the other player’s cards and the cards not yet dealt are not known. In many harder tasks, this information is very difficult or impossible to obtain from the environment, and reinforcement learning becomes the natural choice for evaluating performance and selecting favorable agents.

An additional advantage of not having to compute gradients for backpropagation is that recurrent neural networks can be evolved without requiring a more complex and computationally expensive algorithm [2]. Supervised learning in recurrent networks can be performed. However, existing algorithms are complex and difficult to extrapolate to new neural models. In neuro-evolution, a network may be created, evolved by an EA, and evaluated, without regard for whether the network is feedforward or recurrent.

### 2.1. Marker Based Encoding

Several authors have contributed to neuro-evolution research in the direction of automating the size of the network. Fullmer and Miikkulainen [3] explored marker based encoding of neural networks. Using this strategy, networks are encoded in a single circular chromosome, with start and end markers indicating the beginning and end of neurons in the network. Weight and connection information is encoded within the start and end markers, and the networks are recurrent. The marker-based encoding is unique in that the interpretation of each allele is independent of its locus in the chromosome. Each position is used in such a way that produces the maximum benefit for the network. In the crossover operator, neurons may be added or taken away, and connections may feed back to other neurons or even to themselves. This encoding is very loose and dynamic, and the idea of growing a network to fit the problem at hand was used extensively in this research.

These authors did pioneering work in representing neural structures in an evolutionary algorithm, and applied the work to an object recognition task requiring exploration and discrimination of objects in a simulated environment. Tests confirmed that agents were able to discriminate objects in an environment even when memory was required.

Moriarty and Miikkulainen [4] continued the marker based encoding strategy by applying neuro-evolution to the domain of Othello play. Othello is played on an 8 x 8 board with pieces black on one side and white on the other. Players may only move in unoccupied squares and must linearly surround the opponent’s piece or pieces horizontally, diagonally, or vertically. After doing so, the player flips the opponent’s pieces to her color and play continues. The game ends when there are no legal moves for either player, in which case the player with the most pieces of her color wins. Tournament level players have developed a mobility strategy based on actually maintaining a low piece count, but holding strategic positions and forcing the opponent to make poor moves, surrendering good positions.

Moriarty and Miikkulainen used the power of the marker based encoding strategy and refined its representation of the network. Only hidden layer neurons are represented in the chromosome, with connections to the output layer specifically encoded in the connection information. In the earlier version, output nodes were explicitly defined.

The network was pitted against a random player, a minmax search with $\alpha$-$\beta$ pruning, and finally against themselves. With enough evolution (typically 24 hours on an IBM RS6000 25T workstation), the network defeated all three. According to these authors, after 2000 generations, the networks were employing a beginning mobility strategy.

## 3. SANE

This paper presents an extension of the strength of the marker-based system and the work of David Moriarty. Moriarty developed SANE, a novel neuro-evolution tool that evolves neurons and networks simultaneously. This co-evolution of neurons and networks is an effort to maintain population diversity and encourage neurons to specialize or optimize one aspect of the problem and connect with other neurons that optimize another part of the problem.

SANE encodes network and individual neuron information in chromosomes. While the specific information to be encoded in the chromosome is domain dependent, the approach used in this paper is to encode connections and weights for hidden layer neurons.

Each neuron in the hidden layer is encoded in such a manner, and networks are formed of groups of such structures. These chromosomes are then subjected to the crossover and mutation operators in a search for a globally optimal solution.

The neuro-evolution approach is significant because of its ability to discover difficult, often counter-intuitive strategies [2]. Neuro-evolution also allows a flexibility in the encoding of the network, and allows the evolution of more complex behavior because of its ability to train under reinforcement learning.

### 3.1. SANE Implementation

SANE is short for Symbiotic, Adaptive, Neuro-Evolution. Almost every existing neuro-evolution tool evolves network structures [5], but SANE is unique in that it uses an evolutionary algorithm to evolve neurons *and* network ‘blueprints’. SANE evolves partial solutions to problems in neurons, combines the neurons into networks, and evolves the best network structures.

Sane encodes weight and connection information for each neuron in the neuron population. These neurons are then combined and formed into networks. The networks are evaluated in some domain, and the neurons are rated based on the best networks in which the neurons participated. This neuron level

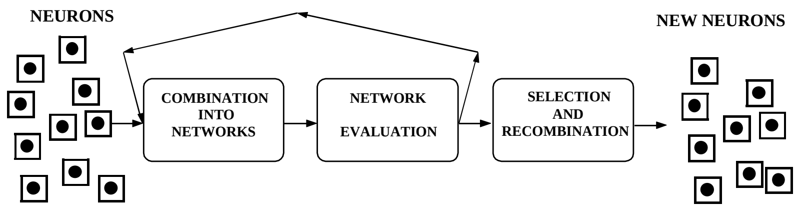

**Figure 1 - Neuron level SANE operation. Reproduced from Moriarty (1997.)**

operation of SANE is shown in Figure 1, reproduced from Moriarty [2]. Figure 1 depicts one generation of neuron level evolution. The best neurons are selected based on the network level evaluation of the networks that those neurons participated in. When the best neurons are selected, crossover and mutation are performed to recombine neurons, which replace the lowest scoring individuals in the population and form the new neuron population for the next generation.

The neurons are evolved in the context of the other neurons in the population. This strategy allows the neurons to rely on other neuron 'specializations' that form in the population, and helps prevent premature convergence of the population. Moriarty defines symbiotic in the SANE acronym as symbiotic evolution in which "individuals explicitly cooperate with each other and rely on the presence of other individuals for survival." [2].

A layer of neural network blueprints is also evolved on top of the neuron population, with network blueprints maintained as a separate population. These blueprints are collections of neurons grouped together to form a hidden layer of a neural network. Since the number of input neurons and output neurons are fixed in a particular environment in SANE, an entire network can be defined by the hidden layer neurons and their weighted connections to the input and output layer. The blueprint population is evaluated, and the crossover and mutation operations are performed on this network population as the genetic search for the best network progresses.

Each member of the network blueprint population specifies a number of pointers to members of the neuron population equal to the hidden layer size. Neurons are combined systematically based on past performance, and are thus grouped in network structures with neurons that perform well together.

In traditional network evolution, the evolutionary search focuses on a single, dominant individual, and can often converge prematurely on local optima. Networks that perform well are bred with other networks that perform well, and the population of networks often becomes very homogeneous, which decreases population diversity and discourages alternative and possibly higher scoring approaches to the network architecture.

A traditional approach to maintaining population diversity has been to increase the mutation rate. This approach injects new genetic material into the population, but only rarely produces better individuals, and follows no specific heuristic to improve performance. A better approach, introduced by Kenneth DeJong, has spawned many similar versions [6]. In DeJong style crowding, when two chromosomes are crossed-over, the children become new individual genotypes. These new children replace the members of the population most similar to them. This preserves more varied members of the population, and improves overall diversity. More powerful techniques, including those that identify chromosomes that contribute to low scoring solutions are available. However, these techniques are costly and add CPU time to a system already very computationally expensive. An approach such as SANE is ideal - building diversity pressures into the system while requiring little or no additional processing.

By evolving neurons, which are partial solutions to the problem to be solved by the resulting networks, SANE automatically maintains diversity in the population [2]. SANE restricts the scope of each individual to a single neuron, with each neuron optimizing a particular sub-task of the network [2]. If one neuron is a member of one or more particularly high scoring networks, its genetic material will begin to permeate throughout the neuron population. In that case, networks evolve that contain several copies of this neuron. These networks will rarely perform well, as difficult tasks often require several different types or “specializations” of neurons. This poor performance will garner a low fitness rating, and lower the chance that the dominant neuron will reproduce in subsequent generations, thus restoring diversity to the population.

This is one of the major contributions of SANE over previous neuro-evolution tools and is one of its major strengths. Although EAs are inherently stochastic techniques, effectively and intelligently guiding evolution toward global optima is the main goal of current trends in evolutionary algorithms. A primary advantage of EAs over gradient descent methods is the search is not inherently biased toward a locally optimal solution. However, EAs differ from purely random sampling algorithms due to their ability to direct the search toward relatively “prospective” regions in the search space [7].

### 3.2. SANE Results

SANE achieves very good results in sequential decision tasks. It has been applied to a number of domains, including the game of Go [8]. It has been used to evolve a network for controlling a robotic arm [2], balancing an inverted pendulum [2], balancing 2 inverted pendulums [9], and capturing simulated prey [10]. In almost every simulation, SANE has been shown to evolve networks more quickly, keep a more diverse population of neurons and networks, and outperform other neuro-evolution strategies.

Richards, et al. applied SANE to the game of Go. Computers have had limited success in the game of Go. Despite its simple gameplay, Go is deceptively hard to master. Black and white stones are alternately placed on a board until both players mutually decide the game is over and pass, at which time the score is calculated and a winner determined.

Go is largely pattern based, which makes it particularly suited for implementation by a neural network. SANE was used to evolve a feedforward network with two input neurons and one output neuron for each board position. Input neuron one is fed a boolean value indicating the presence or absence of a black stone, and input neuron two represents a white stone. The output neurons are fuzzy values indicating a range of relative ‘goodness’ of a move to a particular board position.

SANE achieved quite good results in evolving Go playing networks. SANE was able to defeat a publicly available Go program called Wally, developed by Bill Newman, on small boards. SANE was able to defeat Wally up to a 9 x 9 board, but took 5 days of CPU time. The authors estimated the time to evolve a successful network on a full sized 19x19 board at over a year.

An important conclusion of that experiment was an insight into neural networks and evolutionary algorithms. SANE evolved to defeat deterministic opponents quite quickly, but “…learned little about playing Go and only learned what was necessary to win against that particular opponent.” [8]. When 10% non-determinism was applied to the Wally opponent in the form of random legal moves, SANE actually required more generations to defeat the opponent. Richards, et al. concluded that SANE was finding holes in the deterministic opponent’s strategy, but actually learning Go strategy against the non-deterministic opponent. The non-determinism of blackjack makes the experiment in this paper a more interesting test.

### 3.3. Modifications to SANE

SANE 2.0 has been modified by several people. The primary modifications involve sub-populations, recurrency, and delta-coding. This research uses the sub-population modification of Faustino Gomez and Risto Miikkulainen, and introduces hidden layer growth, and Elman recurrency (a variant of the recurrency introduced by Gomez).

In unmodified SANE, the neurons are in one large population, and a network may be made of neurons from the entire population. As Moriarty [2] showed, in the advanced stages of evolution, instead of converging to a single individual as a standard evolutionary algorithm would, the neuron population forms groups of individuals (neurons) that perform "specialized functions in the target behavior." These neurons specialize to perform a specific feature of the task, combining into networks to form effective solutions to the entire problem.

Sub-population modifications split the neuron population into sub-populations. A sub-population is maintained for each neuron that may be in a hidden layer, and neurons are only replaced in a network from this respective sub-population. This is in an effort to circumscribe the "species" which evolve in advanced stages of SANE evolution, and thus speed up the evolutionary process. This modification also allows each neuron to be evaluated on how well it performs in the context of the other neurons. Neuron specialization, which is hopefully contained in each sub-population, is not hindered or contaminated by recombination across specialization or sub-population.

Sub-populations also increase the performance and allow for more effective creation of recurrent networks. The effectiveness of a neuron is more critically dependent on the neurons to which it is connected in a recurrent network. The specialization of neurons in each sub-population allows recurrent neurons to rely more upon the type of neuron to which they are connected, and the performance of a recurrent network is boosted [10].

## 4. Variable Hidden Layer Sizing

Varying the size of the hidden layer in a neural network is achieved by varying the number of neurons in the hidden layer. Since the input and output neurons have semantics in the environment, the size of the input and output layers are usually fixed in a neural network implementation.

The optimal size of the hidden layer in a neural network has been the topic of much debate and is still very much an open research issue. A common heuristic has been "a harder problem requires a larger hidden layer size", but the number of neurons in the hidden layer of the network is often left to guesswork, or trying several sizes until acceptable results are achieved empirically.

There have traditionally been three approaches to attempting to automate the hidden layer size in a network. One may build a large network and prune it, start with a small network and add to it as needed, or start with a 'sufficient' size, and add or subtract and retrain. Finding an appropriate size for the hidden layer is important. If the hidden layer is too small, the network may not have sufficient ability to encode parameters of the problem. Hidden layers that are too large may require much longer to train and can require unnecessary processing.

This paper proposes a new solution working in conjunction with the genetic selection inherent in the training and creation of networks created with SANE. The size of a single hidden layer becomes another parameter in the genetic search for weights and connections, and networks are evolved with hidden layer size as a genotype along with weight and connection information. Network size is another trait of the individuals in the network population.

### 4.1. Motivation

Varying the hidden layer size creates a more autonomous learning system and eliminates some of the guesswork associated with finding the proper hidden layer size, thus decreasing development time.

Variable hidden layer sizes (VHLS) are also motivated by the earlier work of Moriarty, Miikkulainen, and Fullmer, who developed the marker based encoding strategy. They achieved good results by allowing recurrent networks to assume any size necessary. A strength of reinforcement learning when combined with EAs is the ability to vary network parameters and architecture easily while selecting those individuals that perform the best, regardless of size.

An additional motivation for VHLS is the ability to explore larger networks that may be required to solve a particular problem, and automating the empirical search for the best network size. Complex problems may require larger hidden layers. The ability to evolve a population to more closely match this requirement is an important consideration in any learning system. Alternately, if a smaller hidden layer size can provide better performance, processing will be saved. This important benefit is discussed more fully in Chapter 6.

As mentioned earlier, maintaining population diversity is critical to the effective performance of any evolutionary algorithm. By forcing variable sized networks, a measure of diversity is introduced into the network population. Varying the number of neurons in the hidden layer makes the population of networks more diverse. Network "blueprints" not only explore different combinations of neurons, but different quantities as well. Adding and removing neuron specializations dynamically increases the dimensionality of the network evolution. Diversity is increased in the network population by evolving networks that combine different numbers of neurons (or specializations.)

## 4.2. VHLS Implementation

It is important to note that varying the hidden layer size does not inherently give the networks more power. Networks with hidden layers from 10 to 20 neurons are no more effective than networks with a fixed size of 20 neurons, since the fixed network may evolve zero valued weights for the 10 to 0 extra neurons. In addition, the fixed network may not evolve connections to the extra neurons at all, and effectively becomes a network with fewer hidden layer neurons. This could be seen as the traditional approach of starting with a large network and pruning 'useless' connections.

Pruning is often accomplished by searching for nodes whose associated connection weights have very small magnitude, or running a lesion study to find connections whose existence does not significantly affect network outputs. If $\partial o/\partial w$ is negligible for a given node, where o is the output and w is the weight for a connection, this node may be pruned.

The advantages of dynamic, evolutionary hidden layer sizing are the elimination of searches for 'prunable' nodes, increases in network population diversity, implicit elimination of excess nodes, extensibility to $\Delta$-coding (discussed in Chapter 8), and performance increases for the experiments in this study.

By including various sized networks in the population of candidate solutions, networks are more efficiently sized for the task at hand. Allowing a larger network to evolve zero weights or connections to certain neurons slows the search. Allowing a hidden layer size genotype in the genetic representation of the neuron forces networks to explore different sizes, as networks rarely evolve all zero weights and connections for a neuron. A population of various sized networks also aggregates to fewer CPU cycles than a population of networks that are all at the upper limit of this size range. Evaluating the fitness of a network with a smaller hidden layer size (and later applying that network to the task) requires less CPU time and storage than a larger sized network.

Networks evolved under variable sizing are

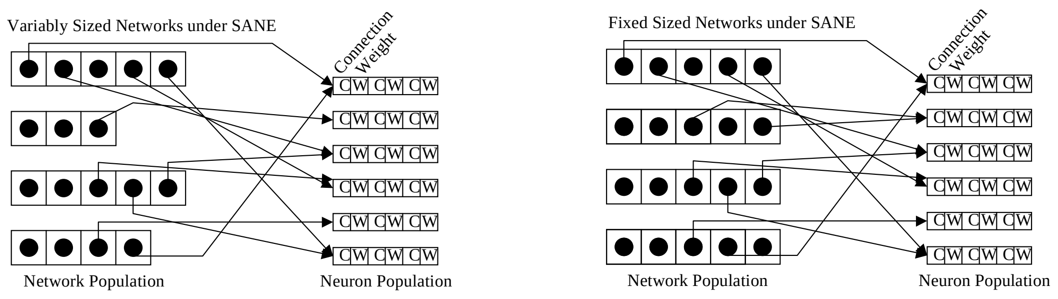

**Figure 2 – Variable and fixed (standard SANE) network sizing**

possible with a large fixed hidden layer model, but specifically removing a neuron is rarely explored by a fixed network architecture. That is to say, a neuron in the hidden layer rarely evolves with all zero connections and weights to other neurons under a fixed architecture. Varying the size of the hidden layer forces this evolution, and increases the dimensionality of the search, not only exploring different combinations of specializations, but different quantities as well. By maintaining a population of networks with different sizes, the network population is more diverse. This diversity helps prevent pre-mature convergence. Figure 2 shows standard and variable network sizing under SANE. The figure demonstrates the network level representation of individual networks as groups of pointers to members of the neuron population. Variable sized networks are composed of different numbers of pointers to the neuron population.

With variable hidden layer sizing, the equation for each output neuron becomes:

$$f\Big(\sum_{i=1}^{n}\big(\sum_{j=1}^{q} w_j\big)x_i\Big)$$

$x_i$ is the output of hidden layer neuron i, and w is the weight matrix for connections between hidden layer and output neurons. The only difference between this equation and the equation for a standard output neuron is that n may now assume variable values. The innermost summation is present because SANE allows the evolution of multiple connections (q) between 2 distinct neurons in adjacent layers.

The enhancements provided by variable hidden layer sizing are similar to those introduced by enforced sub-populations. Sub-populations form in SANE after several generations, due to neuron specialization [2]. By forcing sub-populations, however, the formation of sub-populations is speeded and performance improves, particularly in recurrent networks. Including those features from the start that evolve naturally gives the system a "head start" and allows the evolutionary search to focus on optimal solutions rather than forming specializations first and then optimizing. Having variable sized networks, the evolutionary algorithm eliminates the search for possibly beneficial null connections in a larger network.

Variable hidden layer models require slight modifications to the crossover and mutation operators found in the outer loop evolutionary algorithm of SANE. Since the network sizes are initialized randomly, the crossover operator often performs crossover between two networks of different sizes. Networks are initialized to a random size between two user-defined numbers. A minimum size and maximum size are included to refine searches, as very broad ranges of size require a very large and often unfeasible network population to achieve good results. The following equation generally produced the best results for the tests in this research, although this is domain dependent.

$$\mathbf{5 \leq (Max\_Net\_Size - Min\_Net\_Size) \leq 10}$$

A crossover point is selected to be somewhere between the start and the end of the shorter network chromosome, and crossover is performed as usual. One child assumes the size of the shorter length parent and one assumes the length of the larger parent, as shown in Figure 3. Other crossover techniques were tried, such as picking an independent crossover point on each parent. Performing crossover and producing radically different sized children seemed to introduce too much diversity and led to poor performance, however.

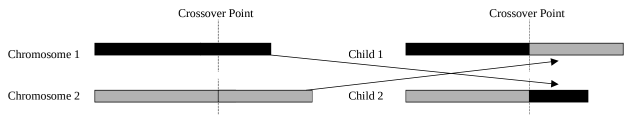

**Figure 3 - Crossover operator**

The mutation operator is also modified slightly to explore larger networks. Instead of traditional mutation, in which a bit is flipped, or a connection or weight value in a chromosome is randomly altered, mutation in variable hidden layer sizing was performed by adding a neuron to each chromosome (if the length of the chromosome is less than Max_Net_Size). This operator is performed on a user-defined

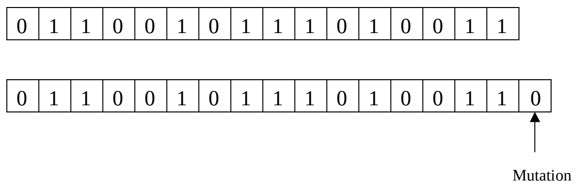

**Figure 4 - The mutation operator**

percentage of the network population per generation to further explore the search space, as shown in Figure 4. The diagram illustrates a bit string representation for simplicity, but in SANE, a complete single neuron encoding including weights and connections is added to the end of a network chromosome during mutation.

### 4.3. Using VHLS in Elman Recurrent Networks

Elman [11] explored recurrent neural networks that provide the system with memory. This is done in the context of giving the system "dynamic properties that are responsive to temporal sequences." These networks include a time parameter, which necessitated a new network model for representing inputs to the system in previous time steps. Elman added context units to a standard feedforward neural architecture as shown in Figure 5. These context units function similarly to input units, but receive their input from the output of the hidden layer in the previous iteration.

Elman notes that in feedforward networks, the hidden units develop internal representations of input patterns and recode those patterns to produce the correct output for any given input. The hidden units in the Elman model thus have the dual task of mapping both an external input and the previous internal state saved in the context units. The internal representations that develop thus have an implicit temporal property [11].

In order to provide the network with short-term memory and give the network the ability to define the problem domain in simpler terms, tests were run with an Elman recurrent neural network. The short-term memory gives

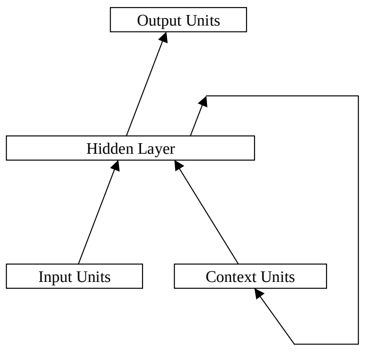

**Figure 5 - Simplified Elman network**

recurrent networks information from previous iterations. In our case, this provides the information about previous cards played.

In SANE, the recurrent network is free to ignore this information and evolve zero weights for the feedback connections, provided the information does not improve the performance of the networks. Hence, the network functions as a feedforward network. However, the previous state information provided by recurrent networks is essential in non-Markovian tasks, where recurrent networks provide significant feedback for decomposing difficult tasks.

The number of examples needed to train a neural network to learn a function increases roughly exponentially with the input dimension [12]. Game playing typically requires at least two input neurons per game square, and most interesting problems have a high input dimension. Recurrent networks can decompose a high dimensional function into many lower dimensional functions connected in a feedback loop, and reduce the difficulty of the problem [13].

## 5. Applying Neuro-Evolution to Blackjack

SANE was modified with sub-populations, hidden layer growth, and Elman recurrency, and applied to the game of blackjack. The environment is a partially observable Markov decision problem. Blackjack provides a unique test, as recurrent networks are applied to the task with the opportunity to evolve into more complex agents, taking advantage of previous state information. By using knowledge of cards played in previous hands, the network can gather more information from the environment, or in a more formal sense, the network can make the environment more accessible. Russell and Norvig [14] describe one property of environments as accessible vs. inaccessible. Accessible environments provide an agent with a complete state of the environment. Despite increases in accessibility through information of previously played cards from recurrent connections, the environment still remains inaccessible because there are some cards the network will never see, and thus some uncertainty in the environment. This problem is interesting because making the problem easier or more accessible can be a goal of the network evolution, by evolving useful recurrent connections. Blackjack is also the only casino game in which a player can gain a statistical edge over the house by using information from previous hands [15].

For this paper, blackjack was played with a single deck, with standard rules. Pair splitting, insurance, and doubling down were not allowed. The player and the dealer were initially dealt two cards, with the player aware of one of the dealer's cards (the up card). The network was aware of the total of its (the player's) hand, and the dealer up card. The network is then activated, and can decide to hit or stand. Hitting gives another card, with the goal of reaching 21. Cards are worth their face value, with 10s, Kings, Queens, and Jacks worth 10 points. An ace is worth 1 or 11, and a player with a hand containing an ace has an option of using the ace as 1 or 11 (if using the ace as an 11 does not make the total more than 21). A hand with this option is referred to as 'soft'. For example, a hand consisting of {A,5} is a soft 16, because hitting and receiving a Jack for {A,5,J} is still 16, although now it is a hard 16. The network (player) wins if it has a higher point total than the dealer, without going over 21 (busting). An initial deal of a 10 value card and an ace is an automatic victory for the player (assuming the dealer does not also have 21), and is referred to as a 'natural.' Ties in blackjack are referred to as a 'push', and the player's bet is returned.

A diagram of the actual network implementation is given in Figure 6. The networks consist of 41 input neurons for the player's point total (separate neurons for hard and soft totals) and the dealer's up card. For example, a player receiving {10,6} with a dealer up card of {8} activates the hard 16 input neuron and the dealer's 8th neuron. The networks have 4 output neurons, 2 for hit and stand. If the hit neuron's output is higher, the network hits, and vice versa. Networks were outfitted with two additional output neurons for raising or lowering the bet on the next hand.

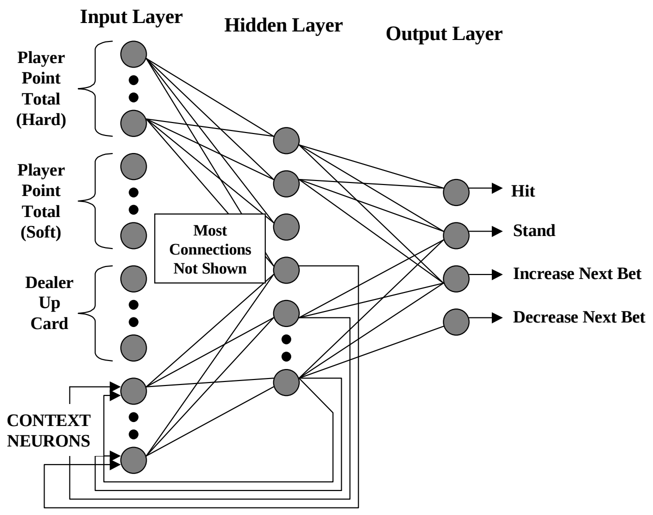

**Figure 6 - Blackjack network structure**

By varying bets, the network can influence its monetary outcome based on the additional information and accessibility from recurrent connections and weights conveying previous decision information. From a domain specific standpoint, this can be thought of as counting cards, or changing present behavior based on previous states and previous cards played.

Card counting is a strategy employed by blackjack professionals to significantly improve the player's odds. Simply stated, the more 10 value cards remaining in the deck, the more favorable future hands will be to the player. Similarly, if all of the face cards and 10s are dealt out early, later hands will favor the dealer. A recurrent network that increases its bets when the deck becomes favorable demonstrates an effective use of the additional information provided by the feedback connections. Advanced blackjack players usually try to track individual cards, whereas the network employs a slightly different strategy of using the point total from previous hands to influence present decisions.

| TEST PARAMETER | fixed | variable |
|---|---|---|
| Decks of play per network eval. | 35 | 35 |
| Number of decks (shue size) | 1 | 1 |
| Hidden layer size | 25 | 20-25 |
| # of Context Neurons | 20 | 20 |
| Network population size | 140 | 140 |
| Neuron population size | 5000 | 5000 |
| Sub-population size | 200 | 200 |
| Adding neuron mutation rate | 2% | 2% |

**Figure 7 - Blackjack test parameters**

The inputs of the model depend on the hidden layer activation of the previous iteration, as well as card total of the player and the dealer up card. Test parameters are given in Figure 7. Most parameters were empirically discovered based on preliminary trials. As an interesting corollary, very early generations evolved networks employing the "always stand" strategy, a very beginning and ineffective strategy found in some human players, and also known as the "dealer bust" strategy [15].

These parameters were established as producing networks that emulated known blackjack strategy and avoided the "always stand" strategy. Providing a sufficiently large neuron population size was the most important factor in producing intelligent network play.

There are widely available blackjack tables, which indicate the correct hit/stand decision for each possible point total in a player's hand, based upon the dealer's up card. No previously dealt cards are considered in the tables. The dealer in blackjack has no choices – the dealer must hit a 16 or below, and must stay on 17 or higher. In these experiments, the dealer does hit a soft 17, as in some casinos. This was the only dealer rule variant introduced in the experiment, to make play slightly harder for the network. Since the dealer's down card is revealed after the network has made a decision, the network is never aware of the dealer's down card, which is not standard in normal blackjack play. This makes keeping track of unplayed cards more difficult for the network.

One test evolved networks with a variable hidden layer size, and one test evolved networks with a fixed hidden layer size. The fixed model had 25 neurons in the hidden layer, while the variable model could have 20 to 25 neurons in the hidden layer. With a 25 neuron fixed hidden layer, the fixed model could evolve all of the networks the variable model could. Evolving zero weights for the connections to extra neurons effectively makes the fixed model the same size as a smaller network. The fixed model was just as powerful as the flexible model, spending more time optimizing its fixed network structure rather than finding the optimal network size. By comparison, the variable sized networks had more built-in population diversity in terms of the network structures, but had to spend generations exploring appropriate network size as well as finding appropriate weights.

The network could learn the remaining contents of the deck and use this information to increase the bet on the next hand when the deck becomes 'favorable' (more high cards left in the deck), or lower the bet when the deck is 'unfavorable' (more non-10 value cards remaining in the deck.) In this sense, if the system evolves networks that take advantage of this additional information, the problem becomes more accessible – that is, more information from the environment is available to the agent, and performance will improve.

Both network architectures were evaluated based upon the mean of the amount of money at the end of 35 decks of blackjack, and the percentage of correct hit/stand decisions made by the network, as defined by known blackjack tables. The network player started with a bankroll of 100 units, and could dynamically determine the next bet in the range of 1 unit to 5 units. Performance based on an average of money remaining and mathematically correct decisions was deemed appropriate. This measure was used to balance the 'real-life' goal of competitive blackjack – to make money, while preventing lucky high bets on the part of the recurrent network by requiring that half of performance be based on the correct decisions according to a blackjack table. Initial tests were conducted exclusively with money remaining as the performance measure. While many networks made intelligent decisions, some ineffective but lucky "always stand" networks performed well on shuffles that were more forgiving of this strategy.

## 6. Experimental Results

Overall, blackjack networks both fixed and variable in hidden layer size, performed well by making intelligent decisions and evolved a strategy similar to a player utilizing the blackjack tables. Given the rules of the game used for these tests, with no doubling down or pair splitting, no insurance, and dealer hitting soft 17, the 'house edge' was 3.28% [15]. When the best network in the entire testing series was run over 20 decks of test play, the network had $116, after starting with $100. Due to the house edge and an average of 8 hands per deck, a non card-counting player playing exactly according to blackjack tables should have only $94.75. It is important to note that this best network was evolved with variable hidden layer sizing.

All tests consisted of trials until statistical significance of scores was established at or beyond a 95% probability of significant difference. Due to extremely computationally expensive tests, SANE was run for 200 generations, and the score achieved at generation 200 taken as the score for the best network on that trial. The average generation is the generation at which the highest scoring network was evolved. Between this generation (and up to 200 generations, when the test was halted), no higher scoring networks were evolved. Although interesting for observation, the generation measures had a very high standard deviation and are not statistically significant.

### 6.1. VHLS Results in Recurrent Networks

Figure 8 shows the results of the trials. A detailed analysis of actual blackjack play by the networks indicated that to a certain extent, networks were using previous decisions to improve performance. Only a small number of networks varied their bets, however. This means that most networks used feedback information to refine the hit/stand decision rather than attempting to bet more when the deck was favorable. The networks that did modify their betting strategy did so very successfully. This was a very advanced trait and evolved in 2% of network tests. Networks that did vary bets had an average 64% success rate. That is, when a network decided to raise its next bet, it won the next hand an average of 64% of the time. A non card-counting player following the blackjack table, as defined in this test, would have won only 47.72% of the time.

| **Recurrent Blackjack Tests** | | | |
|---|---|---|---|
| **Variable Hidden Layer Size** | | **Fixed Hidden Layer Size** | |
| Average Size | 22.641 | Average Size | 25.000 |
| Average Score | 78.104 | Average Score | 72.092 |
| Average Generation | 95.333 | Average Generation | 84.729 |
| Standard Dev. of Score | 10.386 | Standard Dev. of Score | 16.183 |

**Figure 8 - Test results**

Trends in the test results demonstrated that VHLS provides a higher score and more consistency in the form of lower standard deviation of score. VHLS models performed better, and had an average hidden layer size of 22.641 instead of 25. Thus, VHLS provided an average reduction of 9.436% in CPU time for hidden layer processing. The resulting network thus performed 8.339% better and required almost 10% fewer CPU cycles under VHLS.

## 7. Comments

This research has demonstrated a new modification to the SANE neuro-evolution tool and established the effectiveness of evolving Elman recurrent networks. In addition, performance enhancements in the form of more consistent network evolution, and higher score increases were also achieved. This conclusion was confirmed over tests in the domain of partially observable Markov decision problems.

Allowing the evolutionary algorithm to modify the number of hidden layer neurons in the networks increased the average scores over the domain of blackjack. The networks, with variable hidden layer size, were higher scoring, more consistent in performance (lower standard deviation of scores), and reduced CPU time for hidden layer processing by an average of 9.436%.

This research extended the domain of SANE applications to partially observable Markov decision problems. SANE evolved a network capable of predicting cards in future blackjack hands and evolved to make it's environment more accessible by using previous state information.

It has only been recently that neuro-evolution has evolved the power to solve non-Markovian problems [9]. SANE has demonstrated effectiveness in these domains and varying the size of the hidden layer has improved performance and created higher performing networks with lower computational requirements.

For the domain of blackjack play, recurrent networks displayed predictive abilities. Elman recurrent networks were shown to be an effective and efficient addition to SANE, as they require little modification to the internals of the system. A distinction is simply made in the connection of a neuron to indicate a connection to a context neuron. Context neurons then function as input units, and SANE can be applied to many new domains.

This research has also made some headway into creating neuro-evolution models capable of "scaling up" to larger and more complex domains. Tests in this paper have confirmed that growing or varying the hidden layer size is an effective technique for creating larger neural models, and may improve performance, efficiency, and automation of network sizing for many domains.

Although SANE with the hidden layer size modifications did evolve better networks more efficiently, the standard version provided acceptable results. Hidden layer size evolution may be necessary for acceptable performance, when combined with Δ-coding on non-Markovian tasks, and for the esoteric ideal of creating truly automated learning systems.

## 8. Future Work and Conclusion

The goal of dynamic construction of neural networks supports a reduction in development time and is toward, in the general sense, a more autonomous learning system [16]. Automating the selection of hidden layer size augments this goal and simply becomes another search criterion for the evolutionary algorithm. For the domains in this research, varying the hidden layer size has been shown to improve the score of the network and provide a direction toward that autonomy.

Future directions for neuro-evolution research include refining and modifying the very effective SANE model and adding functionality and applicability to newer and more difficult classes of problems. An interesting area of future research is in augmenting the work of Gomez and Miikkulainen in incrementally evolving behaviors. Networks under incremental evolution are not evolved from a random population of neurons and networks, but rather start evolution by building upon previously evolved decision strategies.

The idea of incremental evolution and Δ-coding is to start with simpler tasks and evolve more sophisticated behavior on top of the existing knowledge. Starting with smaller goals, more complex behavior can generally be evolved than starting from scratch.

Delta-coding was not included in the experiments in this paper, but merits discussion due to its importance in hidden layer growth and future work. Originally included in SANE by Faustino Gomez, Delta-coding is a method developed by Whitley et al. [5]. The concept of Delta-coding is to search the neighborhood around the best solution found so far.

Delta-coding is similar to evolving a network population, arriving at the highest scoring network, and then starting over, using this best network as a starting point. Gomez and Miikkulainen [10] showed that Delta-coding can be used to implement incremental evolution by successively evolving more complex prey capture behavior.

After many generations, the population of neurons will become more homogeneous, and the evolutionary algorithm will perform poorly or fail to find a global optimum. When the neuron sub-population has reached a minimum diversity (defined by the designer), the best network chromosome so far is saved. New sub-populations are then initialized with Δ-values representing small differences in the connection weights for each neuron in the best network found so far. Thus, each neuron in the best network has a specific sub-population of neuron Delta-chromosomes designed to improve this neuron specifically. Delta values are added to the connection weights in the best solution and the resulting chromosomes are termed Δ-chromosomes. Those Delta-chromosomes that improve performance are kept and bred.

In each step of this incremental evolution, the difficulty of the task increases and the network requires more power. Varying or growing the hidden layer size may provide additional power to the network evolving more difficult decision strategies.

Delta-coding and hidden layer size variation are together significant in incremental evolution, since they are both methods that can be used when a population has converged. Delta-coding has been shown to be effective in incremental evolution. Hidden layer growth could be combined with Delta-coding to provide more power to networks attempting to achieve higher scores in difficult, non-Markovian tasks. Delta-coding increases the diversity of the candidate solutions, and hidden layer growth increases the dimensionality of the solution. Exploring this combination for solving non-Markovian tasks is an interesting consideration for future research.

Another future area for exploration is on-line learning. For experiments in this paper, network weights and connections were not modified after training. Growing the hidden layer size under on-line learning is another interesting investigation, as networks that learn on-line must be more adaptable and robust. Input to the network during on-line learning may be outside the range anticipated by the designer's training inputs. As a result, networks using on-line learning must be able to adapt to these changes. Variable hidden layer sizing could help in this adaptability.

## References

1. M. Minsky, "Steps Toward Artificial Intelligence," *Computers and Thought*, edited by E. Feigenbaum and J. Feldman, McGraw-Hill, 406-450, 1963.

2. D. Moriarty, "Symbiotic Evolution of Neural Networks in Sequential Decision Tasks," Dept. Of Computer Science, University of Texas at Austin, Ph.D. Dissertation, 1997.

3. B. Fullmer and R. Miikkulainen, "Using Marker-Based Genetic Encoding of Neural Networks to Evolve Finite-State Behavior," in *Proceedings of the First European Conference on Artificial Life*, 1992, pp. 255-262.

4. D. Moriarty and R. Miikkulainen, "Discovering Complex Othello Strategies Through Evolutionary Neural Networks," *Connection Science*, 7(3), pp. 195-209, 1995.

5. D. Whitley, K. Mathias, and P. Fitzhorn, "Delta-coding: An iterative search strategy for genetic algorithms, in *Proceedings of the Fourth International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann, 1991.

6. M. A. Potter and K. De Jong, "Evolving Neural Networks with Collaborative Species," Navy Center for Applied Research in Artificial Intelligence, Technical Report, 1996.

7. L. Patnaik and S. Mandavilli, "Adaptation in Genetic Algorithms," in *Genetic Algorithms for Pattern Recognition*, CRC Press, 1996.

8. N. Richards, D. Moriarty, and R. Miikkulainen, "Evolving neural Networks to Play Go," in *Applied Intelligence*, Vol. 8, Issue 1, 1998.

9. F. Gomez and R. Miikkulainen, "Solving Non-Markovian Control Tasks with Neuro-Evolution," Submitted to the International Conference on Machine Learning, 1998.

10. F. Gomez and R. Miikkulainen, "Incremental Evolution of Complex General

Behavior," *Adaptive Behavior*, Vol. 5, pp. 317-342, 1997.

11. J.L. Elman, "Finding Structure in Time," *Cognitive Science* 14, pp. 179-211, 1990.

12. E. Baum and D. Haussler, "What size network gives valid generalization?," *Neural Computation*, 1(1), pp.151-160, 1994.

13. M. Jones, "Using Recurrent Networks for Dimensionality Reduction," Massachusetts Institute of Technology, AI Technical Report 1396, 1992.

14. S. Russell and P. Norvig, *Artificial Intelligence, A Modern Approach*, Prentice-Hall, 1994.

15. L. Humble and C. Cooper. *The World's Greatest Blackjack Book*, Doubleday, 1980.

16. S. Romaniuk, "Learning to Learn with Evolutionary Growth Perceptrons," in *Genetic Algorithms for Pattern Recognition*, CRC Press, 1996.