The Necessity of Separating Control and Logic When Grounding Language using Neuroevolution

Yonatan Bisk (ybisk@cs.utexas.edu)

Department of Computer Science

The University of Texas at Austin

1 University Station C0500,

Austin, TX 78712 USA

Advisor: Risto Miikkulainen

May 13, 2009

Abstract

In this research we analyze the task of evolving a neural network to understand simple English commands. By understand we mean that the final agent will perform tasks and interact with objects in its world as instructed by the experimenter. The lexicon and grammar are kept small in this experiment. This type of work where semantics are based on an agent's perceptions and actions is referred to as language grounding. A substantial literature exists in this area, as the ability to have a robot or computer understand natural language instructions is one major goal of Artificial Intelligence, but very little has been done that grounds language in actions. We discover that grounding in actions requires a separation of logic and motor control.

1 Motivation

Language is a phenomenon which is largely unique to humans. While many animals have complex schemes for communication, none rival the complexity of human language. Seeing as how we evolved from animals, extensive research exists to try and understand the cognitive jump that lead to human language. One fascinating component of this research is its truly interdisciplinary nature. The brain is obviously thought's vessel but neuroscientists are just beginning to isolate brain regions which are necessary for language. Primatologists help us better understand the differences between humans and our primate relatives and psycholinguists explore the brain indirectly by probing what the brain has difficulty interpreting syntactically and semantically. These all assist in our understanding of humans, but studying language alone, linguistics, and its consistent structures across the world is also fundamental to knowing what to look for in the brain.

The problem with this picture of cognitive science may not be immediately clear but there is a missing link. A neuroscientist can explore and understand only parts of the brain that won't permanently damage it. The primatologist can provide incredible detail about a monkey's brain, the psycholinguist can form compelling arguments and theories from indirect evidence and linguists can explain the structure of language. Nobody can answer the question, "How do humans do it?" While no field can answer that question, computer science can provide some insight. Computer science has the unique ability to construct models.

Neural modeling is common place in the neuroscience and psychology literature. Neural networks have enabled us to model parts of the human brain with incredible accuracy and test psychological theories. Computer science lets us do this for any field, by modeling and simulating behavior researchers can gain crucial insight. In this way, computer science, and specifically the use of connectionist networks, artificial neural networks (ANNs), help us unite the interdisciplinary field of cognitive science.

One important property of language is that it can describe new thoughts and paint pictures the

reader has never been exposed to. It follows that this grows out of an ability to abstract simple real world perceptions and actions into higher level concepts. You weren't born knowing the word blue, azul, or bleu because the name is arbitrary, but you did conceive of the concepts. It required your parent pointing while saying blue a few times for the linguistic mapping to emerge. The same holds for most other simple nouns, adjectives and verbs. We refer to word definitions (semantics) learned in this manner as being grounded in the physical objects and actions. Your definition of dog is a collection of different views of different breeds of dog. Despite the fact that we have not seen the same dogs and therefore have not constructed our representations for dog from the same data, when I say I had a blue dog, you have created a mental picture of a fictional creature which is probably relatively similar to what I'm envisioning. This demonstrates the minds ability to combine grounded concepts like those for blue and dog into a new, fictional, entity. By this process, language allows for the creation of conceptual layers.

In reality, humans never see a dog or any object in isolation. When a mother points and says "dog", a child can't be sure she doesn't mean hair, collar or head. We tease out the meaning by hearing these words again in a new context. This begs the question, did we have a representation for dog before hearing the word or did the concept emerge simultaneously?[1] Is the language learning only accomplished because the child had already grounded an internal concept? And if so, how similar is the child's internal representation to that of the mothers. One piece of indirect evidence that this might be the case the human ability to communicate so easily. When the internal states don't match we have great difficulty communicating. For example, trying to teach a child the difference between a horse and a big dog, or trying to match an outfit with a color blind friend.

Language research is very difficult since analyzing high level behaviors in humans is nearly impossible due to the sheer complexity of the brain and that still living human brain dissection is generally frowned upon. Despite this, psychology and neuroscience often spawn new and fascinating theories about the mind.

[1] See Sapir-Whorf hypothesis

Another source of information about language is linguistics. Most language research that comes from within computer science is based on linguistics. Both are fields which revolve primarily around formal systems and the flow of ideas from linguistics into computer science is as old as the field itself. Most recently computer science has been helping linguists analyze large bodies of text to find regularities and correlations while linguists have assisted computer science by supplying information about the syntax of language.

The important difference between this type of language and that explored more generally in psychology is a desire to formalize semantics in order to derive logical meaning from sentences. This tradition was compatible for many years with computer science and it's binary processing methods. As the years progressed, this approach proved less and less useful as we began to try and tackle topics whose logic or interpretation were difficult to derive due to linguistic ambiguity, context and pragmatics. Most modern techniques have addressed this issue with statistical and probabilistic models. Google's machine translation system for example works by aligning two pieces of text and trying to find correlations.

יש לו גלידה	אני אוהב גלידה
He has ice-cream	I love ice-cream

You may have never read Hebrew before but you can tell גלידה is ice-cream due to its presence in both sentences. The problem comes when trying to translate an idiom, something that requires world knowledge, deeper understanding or ambiguous syntax. Here, Google is analyzing the entire internet and achieving barely passable performance, but a human exposed to a fraction of that data can learn to speak (or at least write) fluently. We believe this is because the speaker already knows a language, already has an internal representation for these concepts (or the tools to build them) and must simply learn a new mapping and structure. In other words, the human has the underlying deep semantics where the computer's "understanding" is quite shallow. As any bilingual asked to translate a sentence will tell you, translating directly and conveying the same meaning are often different.

So whether you are incorporating linguistics, neuroscientific or psychological data, the same semantic wall is hit. What is essential to progress in language research is understanding the source and mechanisms underlying deep semantics. The way to do this is grounding language, connecting language and the world. This task must start small but can be grown to handle increasingly difficult language.

2 Overview

This thesis addresses the task of language grounding through the use of neural networks trained using neuroevolution. We find that it is important to appreciate the two sub-components of the task. Logic and Language comprehension as separate from action and motor control. We demonstrate the difficulty in approaching this task without separating the systems. We propose two systems, one for each component and discuss them in future work.

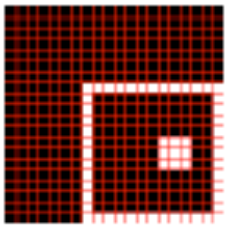
The structure of this thesis will now be to discuss related work (section 3), followed by an explanation of our approach (sections 4 & 5), results (section 7) and a concluding discussion (section 8).

3 Related Work

Grounding as an area of study dates back about 20 years. One of the first pieces of work to understand and define the task comes from [Harnad, 1992]. At this point grounding had yet to leave psychology departments. Early work evolved language within some simulated world or related linguistic descriptions to corresponding images ([Kirby, 2002], [Werner and Dyer, 1992], [Marco and Domenico, 2006]). Grounding of language in perceptual inputs is popular with neuroscientists [Nenov and Dyer, 1994] and computer scientists alike ([Roy, 2005], [Nenov and Dyer, 1993]) but initial work provided little intuition into how to solve the task. Recent work [Chen and Mooney, 2008]

shows how symbols can be matched in annotated video scenes, creating automatic sportscasts describing a simulated robosoccer video sequence. This style of work is young and focuses on mapping to a logical form. Because we approach this problem using genetic algorithms on neural networks, Williams[Williams and Miikkulainen, 2006] who used artificial neural networks to ground descriptions of scenes, is of interest to us.[2]

Williams showed that if a simplified picture could be represented as a matrix of pixels in black and white then those normalized color values could serve as inputs to an ANN and map to a description.



This is an example image from [Williams and Miikkulainen, 2006], which learns the scene description:

"Small object inside of large open square"

This work shows impressive results by using SARDNET [James and Miikkulainen, 1995] for handling grammar in simple English sentences. SARDNET is a neural network approach to representing a sentence spatially. SARDNET uses a self-organizing map to represent words in sentences. As the sequence is read in, corresponding neurons are activated and previous words decay. A lock is placed on a node with each use so the second use of a word in the sentence activates a neighboring, semantically similar, neuron, so as to prevent an old value from being overridden by a new instance of the same word. While the implementation is not essential to our work, representing the temporal nature of a sentence in a loss-less spatial representation is essential to processing of grammar. We contribute to the grounding literature by addressing the hypothesis that

[2]The interested reader in language grounding is advised to also look into the work of Deb Roy [Roy, 2005].

neural networks can be used to ground language in actions, mapping language to movements and interactions.

4 Experimental Plan

We propose to create agents with deep semantics by drawing on inspiration from the biological sciences. While the nature of how much structure is required for human minds is hotly debated, we intend to show how a simple neural network can learn basic linguistic structures, specifically in the form of commands, while simultaneously processing simple visual input and motor outputs. We assume the brain can start as a blank slate and develop the necessary structures for language with minimal feedback. In this spirit, agents' brains will be initialized randomly and unstructured. The only feedback is an assessment at the end of an agent's lifetime with a score for performance. No incremental reward will be given during agents' lifetimes. Of particular interest to us will be the difficulty, of learning increasingly challenging linguistic constructs and what network architectures will evolve to handle them. This will shed light on how to build the semantics behind a sentence. We hope these results will be insightful to scientists and linguists alike as they shed light on possible mechanisms for low level language processing.

5 Approach

Artificial Intelligence is a huge field composed of a tremendous number of different philosophies and approaches to learning. Since this research is focused on finding results potentially useful to the psychology and neuroscientific communities, all work will be done with artificial neural networks. Neural networks are arguably the most biologically plausible technique available. Originally modeled after results from probing living neurons, artificial neural networks approximate the type of activation propagation found in biology. Simply speaking, if several neurons fire and

are connected to some inactive neuron, their activation propagates. More details exist to ensure that activations are properly normalized (sigmoid function) and propagate appropriately.

The principle underlying neural networks is that they encode information (specifically a function) in the values of their weights and the links between nodes. Therefore if the desired actions to be performed by the system must be a function (between inputs and output neurons) encoded by the network, learning the appropriate link weights, connections, and number of neurons is of the utmost importance. In our work we learn these values using the genetic algorithm rtNEAT [Stanley et al., 2005].

5.1 Importance of network architecture

The proper construction of the network is essential to success as the number and types of connections determine the complexity of the problem solvable by the network. We will take a moment to explore a classic example of topological importance to help build an intuition for why this is true. In this example we are interested in creating a network to compute the XOR (exclusive or) of two bits, A and B. The following is the truth table for XOR where A and B denote the inputs and E the correct output.

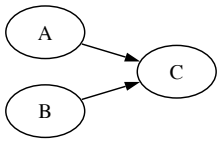
A	B	E
0	0	0
0	1	1
1	0	1
1	1	0

$$(\neg A \wedge B) \vee (A \wedge \neg B)$$

(NOT A AND B) OR (A AND NOT B)

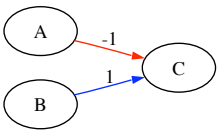
5.1.1 The Logical Explanation

We should quickly recall that XOR can be expressed in predicate logic in terms of AND, OR and NOT. One might naively believe they could represent such a simple expression with the minimal network of three nodes, two inputs and one output like the following.



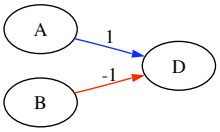
In fact this simple network is only powerful enough to represent AND or OR, though negative links can act as NOTs for the input variables. This is an important point for our work so we will take another moment to address it.

First we will outline an understanding in terms of the logical clauses of XOR. We can create a network for each side of our OR expression above which results in the following two networks, assuming activation thresholds of 1.



$\neg A$	B	C
0	0	0
0	1	1
1	0	0
1	1	0

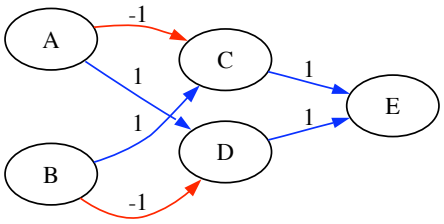
(NOT A AND B)



A	$\neg B$	D
0	0	0
0	1	0
1	0	1
1	1	0

(A AND NOT B)

Given two inputs and two links we can perform each of the two variable binary operations. The same holds true for performing an OR. We therefore combine the above networks, append an OR $(C \vee D \Rightarrow E)$ and the result is the following five neuron network which can compute XOR, a task a smaller network could not. It is important to note that the link allow us to use A and B twice. This solution also assume a propagation threshold of 1.



5.1.2 The Mathematical Explanation

We will now re-examine this same example more formally. First we should note that the activations of neurons are real numbers ($\in \mathbb{R}$) which can only be manipulated in two ways: Multiplication by a link weight (another real number), addition with other inputs. Therefore the number of layers and the number of units per layer limit the type of functions that can be expressed. The neurons' activations (at nodes A and B) get multiplied by link weights connecting them to node C. C then feeds its activation into a sigmoid function of the form $Activation = \frac{1}{1+e^{-input}}$. The shape of this function is seen in Figure 5.1.2.

If we let l_{X_i,Y_i} stand for the link weight between nodes X_i and Y_i then the original three node network can be expressed as:

$$C_{output} = \frac{l_{A,C}}{1+e^{-A}} + \frac{l_{B,C}}{1+e^{-B}}$$

Which allows us to derive the equations in Figure 2.

Therefore, in order for our network to be able to model the XOR function, there must be assignments for $l_{A,C}$ and $l_{B,C}$ such that the four XOR equations produce the desired output (Col. 4

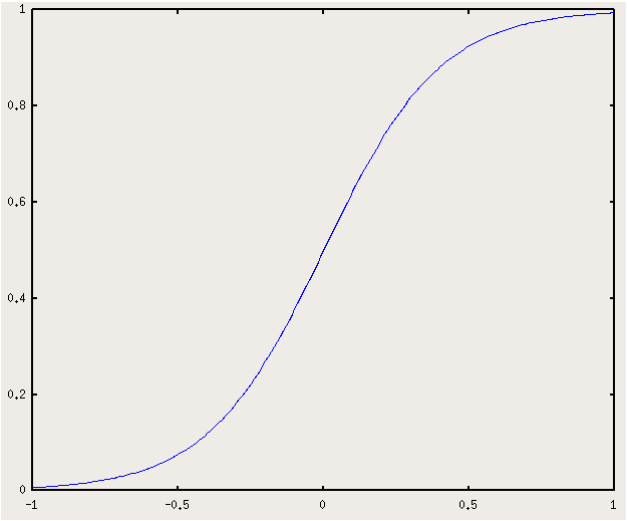


Figure 1: A Sigmoid function as would be used to normalize neuron outputs

A	B	Equation	Desired output
0	0	$\frac{l_{A,C}}{2} + \frac{l_{B,C}}{2}$	0
0	1	$\frac{l_{A,C}}{2} + \frac{1}{1+e^{-1}} l_{B,C}$	1
1	0	$\frac{1}{1+e^{-1}} l_{A,C} + \frac{l_{B,C}}{2}$	1
1	1	$\frac{1}{1+e^{-1}} (l_{A,C} + l_{B,C})$	0

Figure 2: XOR Equations for a three node network

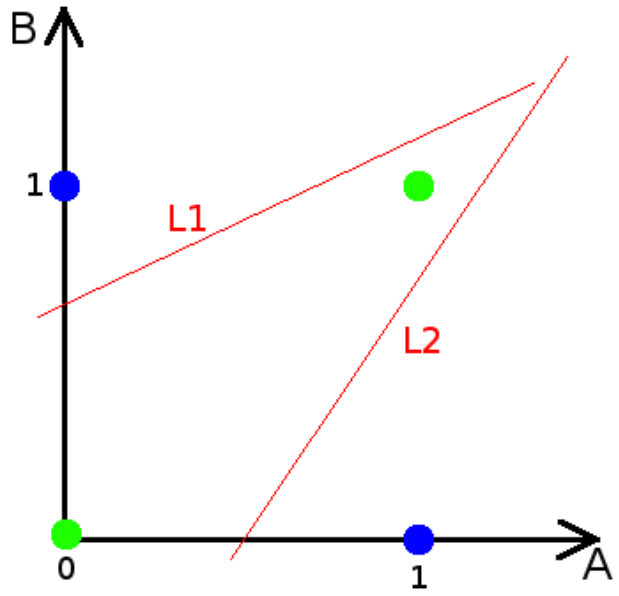


Figure 3: XOR is not linearly separable

) given the inputs in the first two columns. We demonstrate this is not the case by solving three of the equations to produce a contradiction. Equation one simplifies to:

$$l_{A,C} = -l_{B,C}$$

This allows us to solve equation two for:

$$l_{B,C} = (\frac{1}{1+e^{-1}} - \frac{1}{2})^{-1}$$

Finally, using these values for $l_{A,C}$ and $l_{B,C}$, equation three reduces to -1 which does not match the desired XOR output.

The basic idea here is that because XOR is not linearly separable, you cannot draw a line to divide the ones and zeros (see Figure 3) and therefore cannot choose coefficients $l_{A,C}$ and $l_{B,C}$ which let this three node network model XOR. We can also visualize this in 3D. First we must imagine a smooth[3] function which climbs to one at points $(0,1)$ and $(1,0)$ and falls to zero at $(0,0)$ and $(1,1)$, similar in shape to the product function for $x, y \in [-1,1]$ (Figure 4).

[3]We say smooth here, because even though XOR is not a continuously defined function, neural networks model which don't have sharp activation cut-offs model continuous functions.

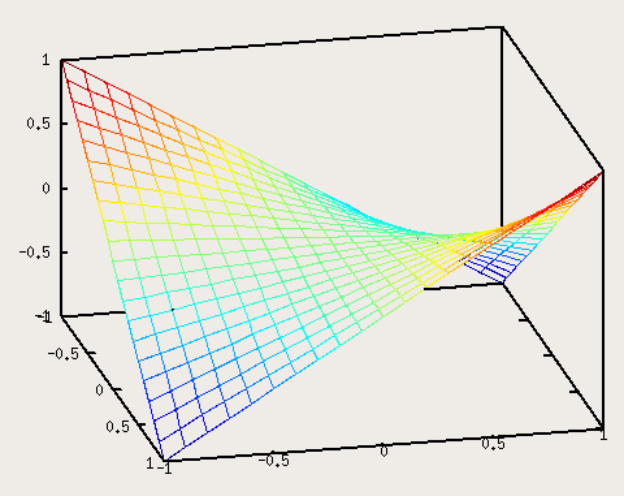


Figure 4: $z = x * y$ where $x, y \in [-1, 1]$

We can then investigate the possible shape of C_{output} from before in terms of $A, B \in [0, 1]$ (shown in Figure 5). As we can see, the function dips and cannot curve back up, so solving two of the four equations will provide a function which works for three out of the four desired outputs. The only parameter not yet discussed is a constant factor which can be multiplied by the sigmoid input. This exaggerates the curvature (Figure 6) of the exponential but the function will not bend back on itself. We need a higher order function to achieve that curvature.

It should be relatively straight forward to see how these equations can be extended to networks of arbitrary size. In particular we discussed a five node network in the previous section which we claimed could solve XOR. The equations are as follows.

$$C_{output} = \frac{l_{A,C}}{1+e^{-A}} + \frac{l_{B,C}}{1+e^{-A}}$$
$$D_{output} = \frac{l_{A,D}}{1+e^{-A}} + \frac{l_{B,D}}{1+e^{-A}}$$
$$E_{output} = l_{C,E} * C_{output} + l_{D,E} * D_{output}$$

This now provides sufficient flexibility for us to solve the XOR problem. By setting the appropriate link weights the E_{output} equation will take the form form of the XOR.

I hope this provides at least the basic intuition that choosing the correct network topology is essential to solving our task. This also provides a very concrete example of a neural network

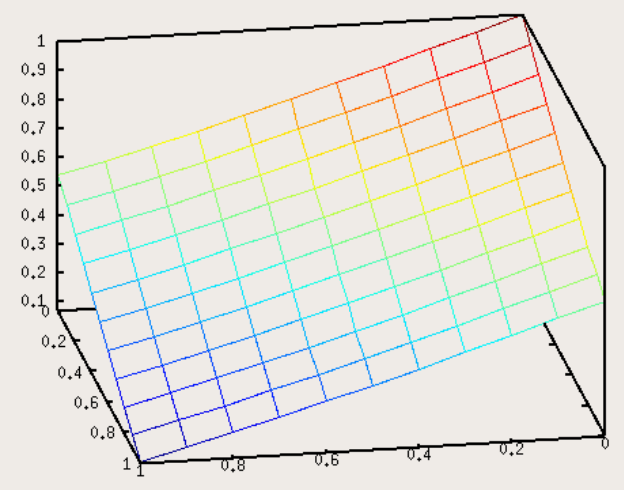


Figure 5

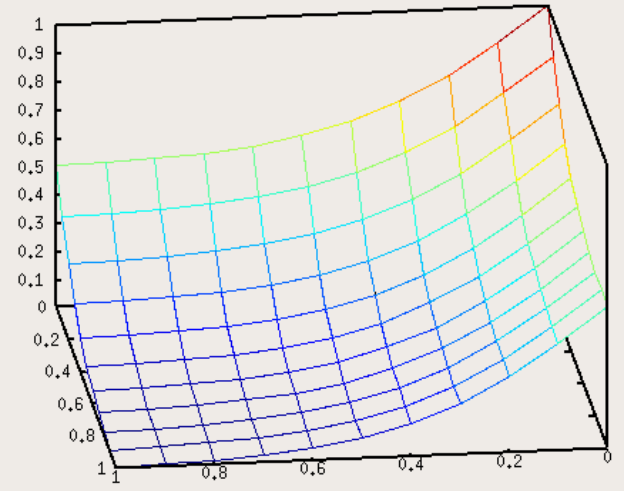
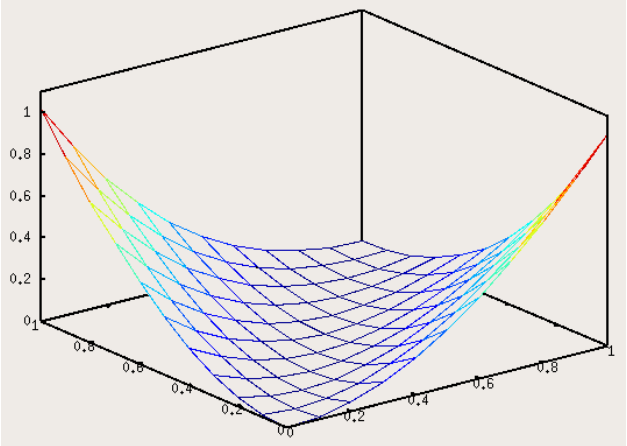


Figure 6



representing a simple function. We will return to XOR later in this work.

5.2 Genetic Algorithms and rtNEAT

Most interesting problems do not have known optimal answers and so discovering the weights of the network and the topology must be done by a learning algorithm. It should also be clear that the learning method must change both link weights and network topology. One approach to searching the space of possible network configurations is genetic algorithms (GA) which evolve a solution. Rather than speak in generalities I will explain the procedure used in this research, a specific GA called rtNEAT or real time Neuro-Evolving Augmented Topologies. rtNEAT is a very powerful technique for evolving agents which has been demonstrated empirically to outperform many other learning methods. As with many techniques, when given a neural network which consists of neurons, links and weights, we will describe the structure as a genome. As seen in Figure 7 this genome can be read to produce a phenotype (the actual physical network).

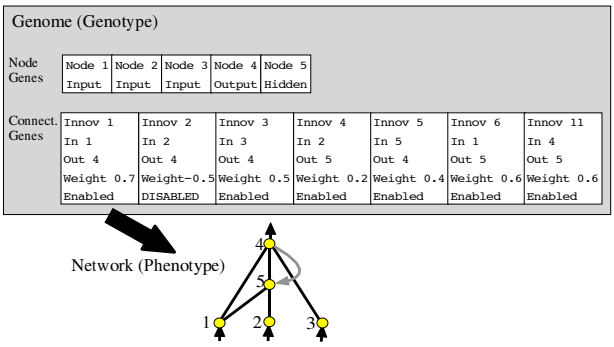


Figure 7: **A NEAT genotype to phenotype mapping.** A genotype is depicted that produces the shown phenotype. The genotype can have arbitrary length, and thereby represent arbitrarily complex networks. Innovation numbers,which allow NEAT to identify which genes match up between different genomes, are shown on top of each gene. [Stanley et al., 2005]

This list representation provides us a simple way of describing and manipulating a network.

The network in this case will take in values of neurons 1, 2 and 3 and will output from neuron 4. Since we are training the network to perform a specific task we can evaluate the output of neuron 4 compared to our expected or desired output.

In this training procedure we will generate 50 random networks and evaluate their performance. We can easily imagine that given a population of 50 networks which differ slightly, at least one network will provide an output more closely related to what we would like. We can then isolate this network from the population and use it as the template for future generations. We will define some probability of adding a connection, adding a neuron or changing a weight and use these probabilities to slightly randomize our new population of networks before repeating the process. Two types of structural mutation are demonstrated in Figure 8.

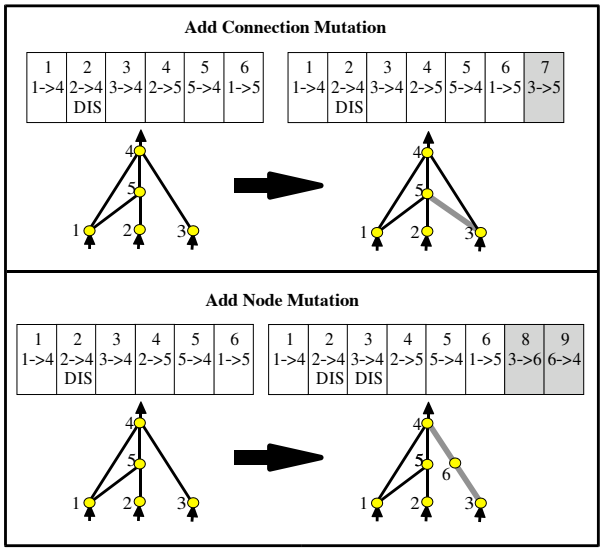


Figure 8: **Structural mutation in NEAT.** The weights and the node genes are not shown for simplicity. A new connection or a new node is added to the network by adding connection genes to the genome. Assuming the node is added after the connection, the genes would be assigned innovation numbers 7, 8, and 9, as the figure illustrates. NEAT can keep an implicit history of the origin of every gene in the population, allowing matching genes to be identified even in different genome structures.[Stanley et al., 2005]

Additionally, as with biological evolution, two or more mutations might emerge which provide equal but different benefit to the population. NEAT's power comes from allowing for two or more species to evolve simultaneously and randomly recombine. In general topological recombination processes are non-trivial due to the changing network architectures. Figure 9 demonstrates how two distinct topologies can be crossed to create a new child. The key insight is that the innovation numbers indicating when the gene was added, allow for the genomes to be aligned without topological analysis. Typically one of the two parent genomes is more successful than the other so the child inherits solely from the one with better fitness. In this case two networks share some set of "good" genes which will be passed on and the new innovations of the parents will be mixed.

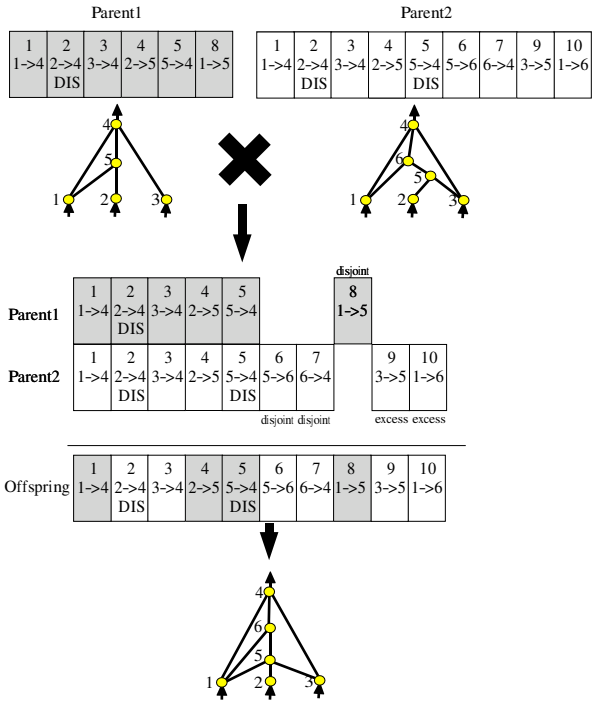


Figure 9: **Matching up genomes for different network topologies using innovation numbers.** [Stanley et al., 2005]

5.3 Experimental Setup

As discussed previously, the primary goal of this work is evolving agents which understand basic language with deep semantics. In order to do this, agents, like with human children, must be trained incrementally. Agents will require a simplified artificial world in which they can build their semantic representations and there must be a mechanism for the experimenter to interact with them. Lastly, in order for an agent to learn a linguistic mapping it will need to be exposed to the words and targets in several different contexts in order to disambiguate the terms.

We will accomplish this by having agents perform several distinct tasks. An initial task is chosen randomly and agents are trained on it until the champion performs with greater than 75% accuracy. At this point a new task is randomly added to the repertoire. Tasks to be added are drawn randomly from a universe of tasks $\mathcal{T}$ and with replacement. All tasks consist of approaching objects. Depending on the task these objects have different shapes (square or triangle) and colors (blue or red). The agent must learn to approach the correct object as determined by the words they are being recited and will be penalized based on their distance from the correct target. The following is an explanation of each component of the experiment in further detail before diving into the experiment and results.

5.3.1 Environment

For the experiments agents exist in a virtual two dimensional world. Objects of various shapes and colors are randomly placed on the edge of a 100 unit circle centered on the agent. The environment is recreated with new objects between every task evaluation. There is only one agent in each virtual world and they are free to move about it continuously.

All agents output two values which correspond to forward and yaw speed. All outputs are normalized to the range $[0, 1]$ so a forward output activation greater than 0.5 moves the agent forward, while less than 0.5 indicated reverse. The same holds for the yaw or angular rotation

speed. In this way, the agent can express movement continuously over the domain with only two outputs. These action conventions are also common in other areas of robotics.

5.3.2 Perception Sensors

Agents need to be able to sense objects in their environment. Since we are primarily interested in language learning we have provided agents with sensors which encode angular information by calculating the angular difference between an agent's heading and the vector which connects the agent to a given target. We calculate an offset in $(-\pi, \pi)$ which we map to $(0, 1)$. So $Activation = \frac{\pi - \alpha}{2 * \pi}$ which keeps the value between zero and one.

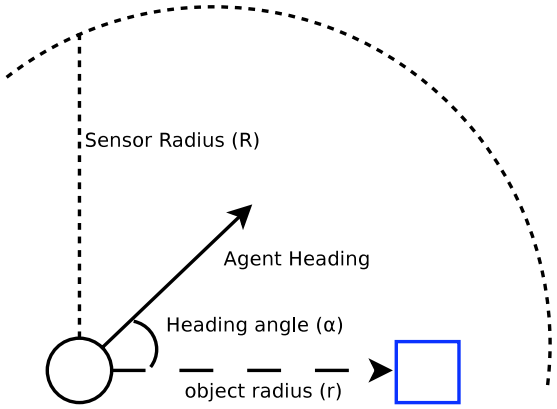


Figure 10: Agent sees a target in the distance

There is a known problem within the community called binding which motivates the decision to have sensors map to objects rather than shapes and colors. When humans are confronted by an object the color can be recognized as an intrinsic property. If instead a separate sensor were used for color and shape, the presence of a Red Square and Blue Triangle would be indistinguishable from a Blue Square and Red Triangle. Our results will show comparative performance with sensors which do not solve the binding problem. In the case of four objects (two red and two blue), there will be one sensor dedicated exclusively to each of the four objects in the environment (blue square,

red square, blue triangle, and red triangle). The ramifications of using a different visual scheme will be investigated later in greater detail.

5.3.3 Word Sensors

While early experiments were done with sophisticated linguistic encodings, due to the difficulty of the task we simply represent each object with a single bit for our current work. This uniquely identifies every target.

$$< BlueSquare, RedSquare, RedTriangle, BlueTriangle >$$

The word sensors will be inputs to the agents where the experimenter can encode directions. One input will be present for each word. A naive construction would simply have binary values for the components. Our experiment will contain four words and therefore four inputs. This activation is constant throughout the agent's lifetime.

Our focus, grammar, poses lots of interesting issues. It comes later in child learning, many people are quite bad at it and it appears to be processed in a different part of the brain than the lexicon[4]. While we won't comment on these high level findings in our simplified task, grammar poses new problems here as well. Grammar is dependent on a temporal encoding of language. "The dog bit the man" is different and perhaps more likely than "the man bit the dog." There is a temporal frame, in this case defined by the word *bit* which specifies interchangeable slots before and after the verb. We will use a simple grammatical structure, *"from-to"* to represent tasks like "go from the red square to the blue triangle." To represent ordering we will decay the first word's activation. So given the input vector defined above, the sentence "blue square red triangle" could be represented as $< 0.5, 0.0, 1.0, 0.0 >$ and so "red triangle blue square" as $< 1.0, 0.0, 0.5, 0.0 >$.

[4]For more details see work on the differences between Wernicke and Broca's areas

5.3.4 Rewards For Training

The penalty system is an essential component to training. We want to prevent any inappropriate bias when training and thus choose our reward scheme to simply be inversely proportional to the agent's distance from the target

$$R = [2 * \mathcal{D} - dist(Agent.position, Target)]^2$$

where $\mathcal{D}$ corresponds to the maximum distance an agent can travel. In this way the reward grows as the agent approaches the target but does not discriminate based on the direction of the approach. The basis for squaring the reward is simply an empirical result that training proceeds more quickly when the differences in rewards are exaggerated. For assistance visualizing reference Figure 11.

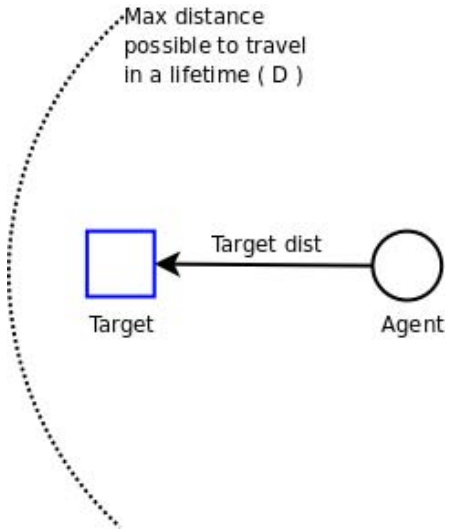


Figure 11: Agent approaching target

For a given experiment there are a universe of possible tasks $\mathcal{T}$ composed of all the sentences that can be presented to an agent. All agents start off with one task drawn randomly from $\mathcal{T}$. As learning progresses additional tasks are drawn with replacement from $\mathcal{T}$ and appended to the training list $\vec{T}$. Additionally, every task is evaluated five times. A new task is added to $\vec{T}$ when over

75% of runs in $\vec{T}$ are completely successfully. We denote the reward of a single run by $r_{t,i}$ where t is a task and i the current run between one and five. We can therefore express a lifetime reward as:

$$Reward_{Lifetime} = \sum_{t \in \vec{T}} \sum_{i=1}^{5} r_{t,i}$$

Since the task is learning a simple grammar, the agents must progress towards an initial target before changing direction and approaching a second one. We only train agents to go between two different targets. This is encoded in the reward function by adding a second term.

$$Reward = (2 * \mathcal{D} - [dist(Agent, T_1) + dist(T_1, T_2)])^2$$

The equation collapses back to the original equation after the first target has been reached.

The lifetime of an agent, the time given to solve the task, is set to slightly longer than the minimum required to reach the two targets needed to solve the task. Targets are located on the edge of a 100 unit radius circle. In the event that the two objects referenced by the grammar are on opposite sides of the environment, an agent might have to travel up to a maximum of 300, three times the radius, units during a lifetime. Since agents can move with a forward speed of 2.5 units per time step, they should theoretically be able to solve the task in $\frac{300}{2.5} = 120$ time steps. In order to give time for exploration we arbitrarily decided to provide a 25% slack, so $Lifetime_{Total} = 1.25 * \frac{300}{2.5} = 150$. The decision to keep the lifetime short will be evaluated later.

the actions are simply two outputs from the network, one interpreted as a forward speed and the other as a rotation speed. The outputs are in the range zero to one and they are mapped to the range negative one to one. This value is then multiplied by a forward or rotation speed. These precise values are not important as long as they are help constant throughout training and testing.

6 Learning Theory

Lastly, before diving into the results of the experiment it is important to understand the types of pitfalls that can be present in machine learning tasks. As with all machine learning algorithms, and

perhaps all things in life, we are always in some state, have some set of possible actions we can make and will receive different rewards depending. By choosing to postpone work for a romantic evening or spend the night out, our net happiness can vary wildly. As we progress through life we hone our ability to approximate that reward, the happiness payoff. We learn which classes need less attention and that a romantic evening is almost always a good idea if you enjoy staying in a relationship. But we also know that the truth is far more nuanced and that the type of evening is important or that getting work done can be very beneficial, so we think ourselves clever when we figure out a way to accomplish both with some novel approach (not bringing a laptop to the restaurant). The same is true of artificially intelligent systems. They are trying to discover the best actions based on experience and make wise decisions in the future. In order to do this they want to discover the optimal function which maps their current state to actions which will be beneficial.

Often times in computer science we visualize this problem as a high dimensional space in which all these variables are accounted for. Imagine that your happiness in life could be plotted in the third dimension and your simplified world consisted of two actions we will plot on the X and Y axis. The X axis will represent the amount of energy you put into love life and the Y the amount of energy into work. We can now imagine there is a hill in front of us covered in fog so only local features can be known. If we put no energy into either we have no happiness and as we put more energy in we get more happiness out. The trick is to find some place in the middle where this hill is at a peak. Occasionally when on the hill we will misstep and lose happiness, requiring us to reposition and try again but in the process learning more about the shape of the hill.

This perhaps contrived sounding example is precisely what our neural networks are doing here. All learning problems can be phrased in this framework. The benefit that the neural network has over us is that it can try several actions at the same time and choose the most successful. This is the purpose of the population size we described earlier. Given that the network is somewhere on the hill, it can make 50 choices and choose whichever gets it furthest up the hill. There is a catch. We have been naively assuming that the hill is smooth, there is no reason this should be the case.

In fact, in almost all things it's not.

If you work a lot with no attention to love life you probably have a pretty good amount of wealth and happiness. You will in fact be at the top of **a** hill if not **the** hill. It will be difficult to leave this hill because spending time away from work may decrease profits, productivity and happiness initially. Despite this, by doing so can you reach the global maxima rather than the local one you've found. The problem then becomes, if changing your actions decreases your short term happiness and you have no guarantee it will increase your long term happiness, why change?

Another problem you might be faced with is an inability to figure out how to go up. You haven't reached the highest happiness peak but every decision you make seems to land you in the same place. You can imagine in this case it is as if you were walking around the hill. The situation keeps changing, but your happiness isn't growing or shrinking.

These are both huge issues in computer science and they are unsolved. There are approaches but no answers. As we complicate our task we must be mindful of local maxima that might exist we didn't anticipate and be weary of slowing progress. One particular type of slowing which our system is prone to is due to network size. We discussed earlier ways in which a network can grow. As the network grows the number of variables that can be changed (links added, weights changed) grows as well. If we are only trying 50 changes per generation our search for the optimal solution necessarily slows.

So we are now faced with the dilemma: How do we know if training has stopped at a local maxima and not just slowed? The answer to that will hopefully be analyzing action traces and evolved network topologies. In other words, a qualitative analysis of the results compared with what were desired actions.

7 Results

Previous work indicates that agents can tell if there is an object in front of them, and perhaps appreciate something about the layout of the objects, a baseline is therefore proposed to distinguish an agent that understands something about the linguistic task versus simply the layout of the objects. These calculations are based on an agent which simply tries to cover as much area over the circle as possible, a base-runner. The agent will proceed to the first seen object, and then to the next closest, repeating this until their lifetime runs out. Given that the radius of the circle of targets is 100 and their lifetime 150, an agent can theoretically cover the 100 unit radius distance plus 275 units on the edge of the circle. This means that the likelihood of their encountering any given object is approximately 44% or $\frac{275}{2*\pi*100}$. Therefore, the probability that both of the correct targets appear in that 275 unit stretch and in the correct ordering is simply $.44^2 * \frac{1}{2}$ or 9.68%. The presence of other targets along the path is unimportant. Alternatively if the agent takes the shortest path between two objects more ground can be covered. If we simplify and assume objects are approximately located on the corners of a square we can calculate that an agent can reach 75% of targets, which yields a baseline performance of $\frac{.75^2}{2}$ or 28.13%.

The following provide results from 130 hours of training. The primary results show agent performance with sensors which solve the binding problem for them and with the limited lifetime detailed above. To test the validity of these choices, these results are presented alongside the identical experiments with simpler sensors and longer lifetimes, as were discovered previously.

The first important point to see here is that neither of the first two columns perform better than the baseline.

The third column results for long life are troubling at first blush. They greatly outperform baseline performance and that of bound sensors with a short lifetime. The first tip off that something might be awry comes from investigating the genome numbers at the bottom of the table. The genome number helps trace the number of generations that transpired. This value is updated with

Task	Simple sensors	Bound sensors	Long life
RS ⇒ BT	12.30	29.70	68.94
RS ⇒ BS	8.90	33.60	68.02
RS ⇒ RT	9.80	15.00	81.64
BS ⇒ BT	3.20	33.30	65.70
BS ⇒ RS	7.00	32.40	70.06
BS ⇒ RT	3.40	9.40	67.56
RT ⇒ BT	12.50	19.50	66.66
RT ⇒ BS	4.00	7.90	68.14
RT ⇒ RS	8.80	18.30	84.92
BT ⇒ BS	1.60	34.8	66.42
BT ⇒ RS	10.50	30.60	69.38
BT ⇒ RT	4.90	19.50	66.74
μ	7.24	23.67	70.35
σ	3.59	9.43	5.95
gnm #	6780	12274	2245

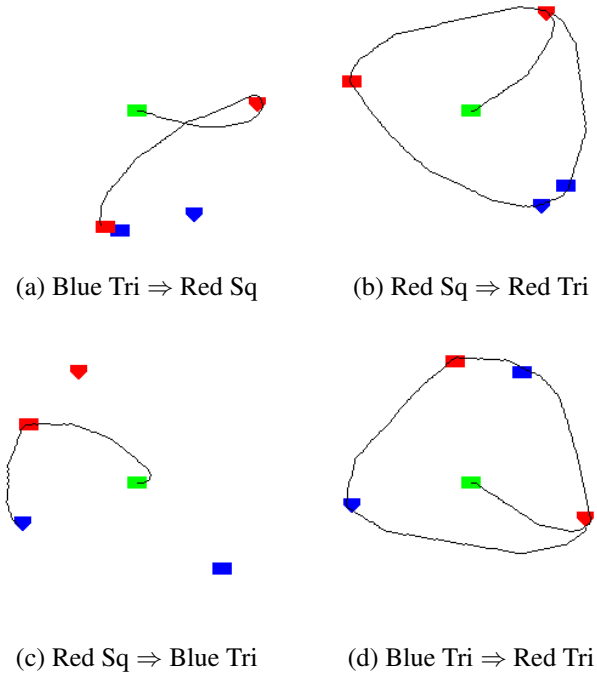
Figure 12: Results from one training run of each column. The numbers are all percentages denoting the percentage of successfully completed tasks. The first two columns were run with lifetimes of 150 timesteps while the third column was allowed 500 time steps.

every evolutionary step. Therefore, if evolution has stagnated the value will remain constant or grow very slowly. When during training, the task has been solved the champion of the previous generation will oftentimes remain champion, leaving the genome number unchanged. The long life agent appears to "solve" the task in a fraction of the time spent by the others. This result indicates that a solution was easy to discover. To investigate further a new baseline base-runner baseline needs to be calculated given $Lifetime_{Total} = 500$ and traces of the agents actions need to be analyzed.

$$Lifetime_{Total} = 500 \Rightarrow \mathcal{D} = 1250$$

so the agent can traverse the circle $\frac{1250}{2*\pi*100}$ or 1.98 times.

It is therefore almost guaranteed for an agent to solve the task. Investigating agent traces verifies that in fact the agents have discovered this. We see this in Figure 13. The left column shows traces left by agents with short lifetimes while the right side are agents with long lives. The appeal of this



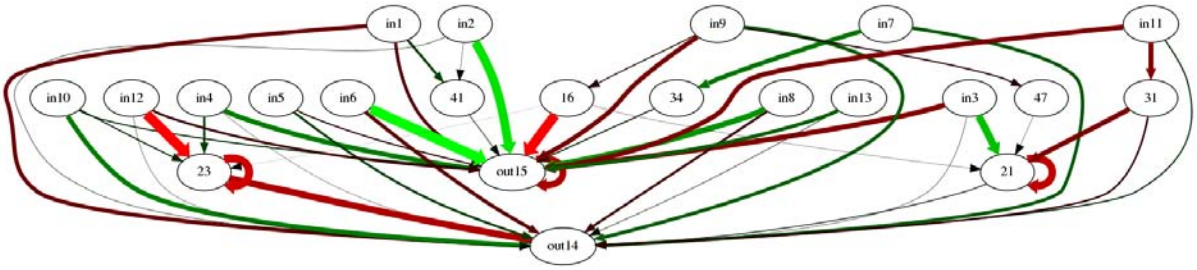
(a) Blue Tri ⇒ Red Sq (b) Red Sq ⇒ Red Tri

(c) Red Sq ⇒ Blue Tri (d) Blue Tri ⇒ Red Tri

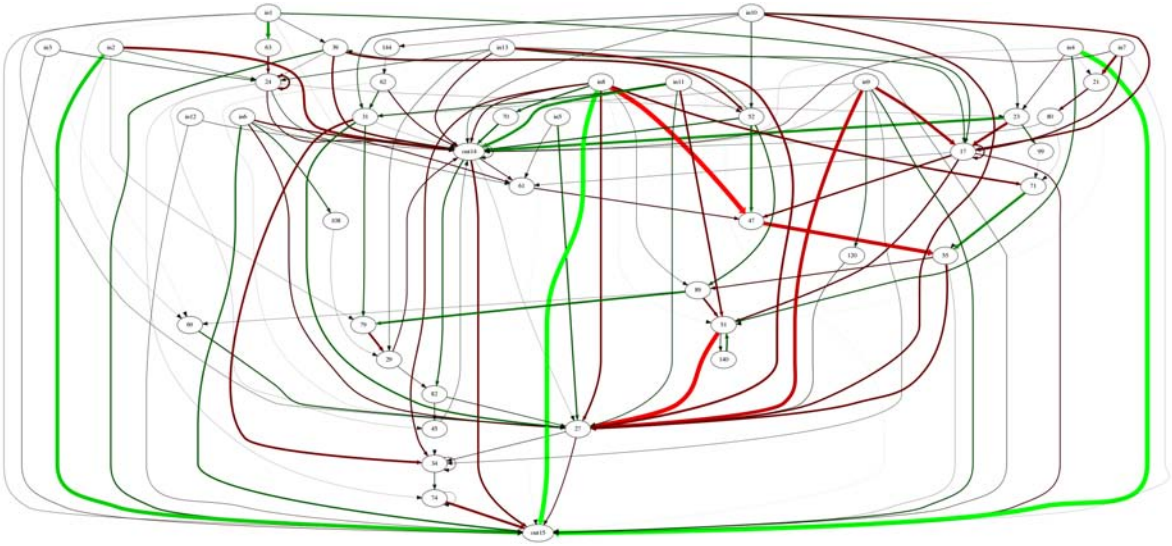
Figure 13: **A**gent Traces In these diagrams we see the differences in the search strategy of the two agent types. The left column are agents with short lifetimes and the right have long lives.

strategy appears to be very strong. This was verified by changing the reward function to include a term which benefits short lifetimes. Despite this, agents would not leave their local maxima to find the global optima. An additional draw to this strategy is its simplicity.

In Figure 14, we present the two champion brains evaluated. The weights continue to change with no improvement and random complexity added in the case of 14b which indicates it is searching for a solution while that of 14a has converged. In both cases, basic memory has evolved, seen by the neurons with self-links, but the strong red arrows on the recurrent links indicate a strong negative value of fast decay.



(a) Long Lifetime



(b) Short Lifetime

Figure 14: Champion brains with short and long lifetimes

8 Discussion and Future Work

It is clear there are challenges affiliated with training neural networks and then challenges which are related to the nature of the task. We have only scratched the surface here of what needs to be researched in this area. We have learned about the presence of several pitfalls that need to be avoided and have insight into the importance of the vision binding problem.

To properly understand why the system is failing we need to analyze the nature of the task further. To do so we will describe a grammar-free simplification. Let us assume there are only two objects present in a discrete world. Furthermore let us assume that the targets can only be in two places, left and right of the agent. In our thought experiment an agent simply need be told which

target to approach and where one of them is located. For example, if the agent is told to go to target A and that it is on the left, he simply needs to respond with the action "Left." We will now outline the four cases:

Word	A_{Loc}	Output
A	Left	Left
A	Right	Right
B	Left	Right
B	Right	Left

This truth table corresponds directly to the XOR function. If that is not immediately obvious you need only substitute 0 for A and 0 for Left. Given this truth table we can now draw a Karnaugh map to visualize the logic necessary to solve the task.

	Target A	**Target B**
Left	L	R
Right	R	L

As a quick refresher, with a Karnaugh map you would like to find adjacent like terms as they allow for simplification. In the case of diagonals like we have here this is not possible. This means that every logical clause must be learned separately. We found that NEAT required nearly a thousand generations to properly learn this simple setup.

This training was performed in logical isolation. The system was just learning XOR. This leads us to wonder, if placed in a simulated world would they still solve the task. We raise this concern because it seems to correlate with the results we saw earlier. In an actual target locating task an agent can perform a search and oftentimes perform with sufficient accuracy that there is no evolutionary pressure to discover a more sophisticated logic. In other words, as discussed earlier the system finds itself in a local optima.

NEAT's speciation is specifically designed to assist this problem by allowing species to evolve different paths for some period of time before requiring them to measure up with the current champion. In this time they can theoretically explore different approaches to solving the problem and potentially find the global optima. In the case of XOR a species could theoretically learn the deeper logic in a reasonable period of time and prevent being thrown out during evaluations.

By contrast we shall now analyze the four target extension of our toy example. If we place four targets in a ring (North, South, East and West) and an agent in the center, by having the targets maintain their relative positions an identical setup presents itself, one in which we only need to provide the position of one target to discover the relative locations of the others. Here an agent must reply with a desired cardinal direction rather than the Left-Right of before. This setup produces the following Karnaugh map.

	A	**B**	**C**	**D**
N	N	W	S	E
E	E	N	W	S
S	S	E	N	W
W	W	S	E	N

Again we see the intense diaganolization. We stopped training NEAT after 300,000 generations were unsuccessful at solving the task. We should note that this XOR style diagonal logic is nearly identical to that of a multiplexor. Early on we saw the type of function that needed to be modeled for solving the XOR. If we encode the cardinal directions and A through D as the binary numbers zero through three in order then there are two output bits which have the following Karnaugh maps.

	A	**B**	**C**	**D**
N	0	0	1	1
E	0	1	1	0
S	1	1	0	0
W	1	0	0	1

	A	B	C	D
N	0	1	0	1
E	1	0	1	0
S	0	1	0	1
W	1	0	1	0

We can again try to visualize the hills and valleys that these would create and it becomes apparent that several more layers of the network or embeddings of sigmoid functions would be necessary to capture this curvature.

It therefore seems that NEAT (at least in its present state) has great difficulty discovering this logic in isolation and cannot be expected to find it during simulation.

An alternative approach to learning logic utilizes the cascade correlation algorithm [SE Falhman, 1990]. Cascade correlation works by making directed structural changes which build a network hierarchically. The network is trained to minimize error before fixing the link weights and introducing a new node. The effect of fixing these component structures is to create features which group various inputs. The intuition should therefore follow from our earlier discussion of XOR. Because the new node is trained to minimize error it can learn one of the logical sub-components of XOR. This value will then be fixed, allowing for a second structure to emerge in parallel or on top of the existing one. In the end a tiered system emerges which can correlate with the underlying logic. Cascade correlation is a supervised algorithm but the structural mutations can be applied to NEAT [Kohl and Miikkulainen, 2009].

We found that in fact cascade NEAT was effective at learning the diagonal maps provided above. Despite this, Cascade-NEAT does not solve our general problem. We discovered that while Cascade-NEAT had worked very well for the binary maps or binary multiplexor, the performance seemed to wane when more continuous values were presented. This can be seen in Figure 8. Specifically we created a four input mux with a four bit coding signal. We then trained networks where the MUX inputs ranged between zero to one in increments of one eighth and one sixteenth.

	XOR	4 Targets	Mux 8	Mux 16
NEAT	98.4	84.94	83.92	83
Cascade-NEAT	100.00	93.95	90.0	88.4

Figure 15: Here we show the performance of NEAT and Cascade-NEAT on three related tasks. The first two are the direct truth tables discussed previously. The third and fourth are extensions of the 4 Target case to a slightly more continuous domain. Here we created a 4-4 Mux in which the words were binary but the input lines were all possible combinations of four inputs chosen $\frac{0}{8}$ to $\frac{8}{8}$ and $\frac{0}{16}$ to $\frac{16}{16}$ respectively.

Cascade-NEAT's performance might eventually reach 100% as in the case of the simpler XOR but we had to terminate training after 12 hours due to practical constraints. This produced the performances previously referenced.

While there are still some drawbacks to Cascade-NEAT it does seem to perform very well compared to regular NEAT in the more logical component of this task. Due to NEAT's performance in other continuous domains its possible that a joint approach would be ideal for solving this task.

We would point future researchers to literature interested in merging cascade correlation and evolutionary approaches ([Kohl and Miikkulainen, 2009], [Tulai and Oppacher, 2002], [Potter and Jong, 2000]). It is possible that some of the issues raised here could be dealt with better by neurons which multiplied activations.

Ideally, if an appropriate learning algorithm were created for this task, then the future future direction of this work is to move the visual system to one more akin to the black and white grid of [Williams and Miikkulainen, 2006] which was discussed at the beginning. In this domain the binding problem will have a different structure as individual "pixels" or grid points will implicitly encode color while drawing out a shape. This will hopefully interface with a more complex system like SARDNET [James and Miikkulainen, 1995] for handing grammar and language. A system of that nature would be the first complete system that takes in visual input very much akin to humans, represents grammar in an elegant spatial network inspired by human memory and produce fluid actions. Thankfully we have been able to provide here the building blocks for this future research and have discovered some of the many difficulties of learning this task.

8.1 Mirror neurons

Much of the insight into shared internal categories comes from a blossoming area of neurology and psychology which investigates mirror neurons. Mirror neurons are a part of our brain which imitates others actions as though they were our own. These neurons were discovered when researchers suspected that certain neurons in a monkeys' brain were affiliated with the actions of reaching for an object. Sure enough the neurons fired whenever a monkey reached for it, but then a strange accidental discovery was made. If the human researcher reached for the same object then the same neurons in the monkey's brain fired. The neurons controlling the monkey's actions were active when the human performed the same task. These results were replicated for more complex tasks. This led some researchers to believe that monkey's (and humans) model the state of others around them internally. Additional results showed that hearing the sound made when performing a task also triggers the same activation as performing said task. This begs the question, would hearing the task described activate these neurons in humans? In other words, do our linguistic representations, at least for simple actions, map all the way down to low level neuron structures devoted to modeling the actions of others? If so, it would make sense why language works as well as it does. We leave this question open for the reader to ponder. This brand new research also motivates our interest in using neural networks to perform these tasks. Neural networks help test the plausibility of using small networks to model others and eventually build up to more cognitively difficult tasks. ([Fischer and Zwaan, 2008], [Rizzolatti and Arbib, 1998], [Hurford, 2002], [Kohler et al., 2002])

9 Acknowledgments

Would like to extend thank-yous to Igor Karpov whose assistance was essential to this thesis and all work leading up to it. Ken Stanley for the use of his diagrams for explaining rtNEAT. Nate Kohl for his assistance with cascade correlation. And the thesis review committee Dana Ballard and Ray Mooney with a special thanks to Ray for his role as second reader.

References

[Chen and Mooney, 2008] Chen, D. L. and Mooney, R. J. (2008). Learning to sportscast: a test of grounded language acquisition. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 128–135, New York, NY, USA. ACM.

[Fischer and Zwaan, 2008] Fischer, M. H. and Zwaan, R. A. (2008). Embodied language: A review of the role of motor system in language comprehension. *The Quarterly Journal of Experimental Psychology*.

[Harnad, 1992] Harnad, S. (1992). Grounding symbols in the analog world with neural nets: A hybrid model. *Special Issue on "Connectionism versus Symbolism"*.

[Hurford, 2002] Hurford, J. R. (2002). Language beyond our grasp: What mirror neurons can, and cannot, do for language evolution. *The Evolution of Communication Systems: A Comparative Approach*, pages 1–16.

[James and Miikkulainen, 1995] James, D. L. and Miikkulainen, R. (1995). Sardnet: A self-organizing feature map for sequences. *Advances in Neural Information Processing Systems*, 7:577584.

[Kirby, 2002] Kirby, S. (2002). Natural language from artificial life. *Artificial Life 8*.

[Kohl and Miikkulainen, 2009] Kohl, N. and Miikkulainen, R. (2009). Evolving neural networks for strategic decision-making problems. *Neural Networks, Special issue on Goal-Directed Neural Systems*.

[Kohler et al., 2002] Kohler, E., Keysers, C., Umilta, M., and Fogassi, L. (2002). Hearing sounds, understanding actions: action representation in mirror neurons. *Science*.

[Marco and Domenico, 2006] Marco, M. and Domenico, P. (2006). The emergence of language: how to simulate it. In C. Lyon, C. N. and Cangelosi, A., editors, *Emergence and Evolution of Linguistic Communication*. Berlin: Springer Verlag.

[Nenov and Dyer, 1993] Nenov, V. I. and Dyer, M. G. (1993). Perceptually grounded language learning: Part I—a neural network architecture for robust sequence association. *Connection Science*, 5:115–138.

[Nenov and Dyer, 1994] Nenov, V. I. and Dyer, M. G. (1994). Perceptually grounded language learning: Part II—DETE: A neural/procedural model. *Connection Science*, 6:3–41.

[Potter and Jong, 2000] Potter, M. A. and Jong, K. A. D. (2000). Cooperative coeveolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*.

[Rizzolatti and Arbib, 1998] Rizzolatti, G. and Arbib, M. A. (1998). Language within our grasp. *Trends in Neurosciences*, 21(5):188 – 194.

[Roy, 2005] Roy, D. (2005). Semiotic schemas: A framework for grounding language in action and perception. *Artificial Intelligence*, 167(1-2):170–205.

[SE Falhman, 1990] SE Falhman, C. L. (1990). The cascade-correlation learning architecture. *Advances in neural information processing Systems*.

[Stanley et al., 2005] Stanley, K. O., Bryant, B. D., and Miikkulainen, R. (2005). Real-time neuroevolution in the nero video game. *IEEE Trans. Evolutionary Computation*, 9(6):653–668.

[Tulai and Oppacher, 2002] Tulai, A. F. and Oppacher, F. (2002). Combining competitive and cooperative coevolution for training cascade neural networks. *In Proceedings of the Genetic and Evolutionary Computation Conference*, pages 618–625.

[Werner and Dyer, 1992] Werner, G. and Dyer, M. (1992). Evolution of communication in artificial organisms. In Langton, C., Taylor, C., Farmer, D., and Rasmussen, S., editors, *Artificial Life II*, pages 659–687, Redwood City, CA. Addison-Wesley Pub.

[Williams and Miikkulainen, 2006] Williams, P. and Miikkulainen, R. (2006). Grounding language in descriptions of scenes. *In Proceedings of the 28th Annual Conference of the Cognitive Science Society (COGSCI-06, Vancouver, Canada). Hillsdale, NJ: Erlbaum.*