Investigation of Evolving Neural Networks through Augmenting Typologies

Described Sections:

1. Motivation(Page: )

◦ Neat Short()

2. Concerns and design decisions()

3. Structure

4. Process()

5. Proposed Structure and implementation

6. Finished Tests and comparisons

Limitations

Future and ongoing work

Works cited

Motivation:

There are many different machine learning models out in the world today, and expanding on the different techniques used is important in growing an understanding and capability to work with and advance different machine learning techniques.

This is an in depth investigation of NEAT (Neurological Evolving Augmenting Typologies).

NEAT Short:

NEAT stands for Neurological Evolution of Augmenting Typologies. And uses genetics and natural selection to gradually build the best neural network.

• Progressive evolution of a network for both weights, and topology could save time of people trying to decide on the topology of networks for a particular problem.(Gruau et al. 1996) Although Enforced sub populations(ESP)(Gomex and miikkulainen,1999) proved, it could be done X5 faster, by simply restarting with a random number of hidden neurons whenever it became stuck.

◦ ”attempt to prove Gruau right, if done right the evolving structure along with connection weights can significantly enhance the performance of NE(Neurological Evolution)”(Kenneth O. Stanley and Risto Miikkulainen (2002)).

◦ (Kenneth O. Stanley and Risto Miikkulainen (2002)) address

▪ Is there a genetic representation that allows disparate typologies to cross over in a meaningful way.

▪ Question: How can topological innovation that needs a few generations to be optimized be protected so that it does not disappear from the population prematurely?

Answer:

▪ Question: How can topology be minimized through evolution without the need for a specially contrived fitness function that measures complexity?

As described here are a few of the described/Proposed advantages of NEAT Network evolution:

• Employing a principled method of crossover of different typologies

• Protecting structural innovation using specialization

• Incrementally growing from a minimal structure

Further references(Link:http://www.asimovinstitute.org/neural-network-zoo-prequel-cells-layers/) describe different Neural Network cells such as:

• Recurrent weights(Grouped by color),

• Recurrent Cell(previous iteration)

• Feedforward cell(Basic cell)

• GRU Cell

• LSTM Cell

Being able to dynamically use NEAT and different machine learning methods, there should be a general structure between each machine learning methods, in order to easily implement and compare each model.

According to (Kenneth O. Stanley and Risto Miikkulainen (2002).), protecting innovations with a specialization is an attempt to prevent genes from prematurely dying out without more investigation.

Competing conventions(2.2):

• One main problem with Competing for conventions problem(Montana and Davis, 1989; Schaffer et al., 1992).

• Permutations problem(Radcliffe, 1993).

Concerns and design decisions:

Some concerns with building the NEAT Typology described by Kenneth O. Stanley and Risto Miikkulainen (2002). "Evolving Neural Networks Through Augmenting Topologies."

Missing descriptions, and processes:

•

In order to keep this package flexible and capable of being used in the comparison between other machine learning models, here are some prescribed structure decisions made to ensure easy maintenance and functionality:

Concerns with creating common simulation/interface structure:

1. Different simulation types:

• N players with turns.

• N players in real-time.

2. Different input/output structures and displays.

• A real-time strategy game might contain a 2D grid of the world, a spherical globe, or some other grid representation(Hexagonal grid from Civilization).

• Additionally, a turn-based game might have intermediate steps where the player(s) might perform specific actions within an iterative step cycle: for example, in the game: Settlers of Catan there are different phases: Gather resources, Trade, build, and buy development cards.

• Shops showing ‘hidden’ in-game menus, and resources(ie, items) allowing players to buy and sell items and allowing for different tasks to be performed only within menus.

• Other games include a re-setup including building a deck of trading cards around a specific playstyle or strategy. This might be the majority of the thought process and training to build the most effective deck around a given playstyle or composition.

Problem: What about regularized testing for several ‘maps’/scenarios?

Example: let’s say we are training a model to play super Mario, and want to generate many different maps to create a more regularized model to play super Mario? Well not only do we need to allow for the creation

Solution: Create a ‘List’ storing each seed for every level loaded; no need to keep an original copy.

How might these different steps/ turns be represented within a given set system and require minimal change and modification?

3. Different Possible moves and output spaces.

• Real-Time strategy games contain several different moves for every possible position in the game:

◦ Move Unit to position

◦ Create building at position

* Games contain special moves for players in special roles, including games such as:
  + Poker

• Given Functions associated with simulation:

◦ Simulate Pregame (Function pointer)

◦ Intermediate Steps (Functional decorator pattern)

◦ Simulate Endgame (Function pointer?)

Concerns regarding the structure and future structural and functional additions:

• Functionality,

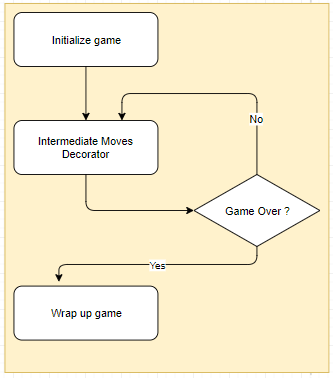
◦ A chance for different Hyper Parameters, (IE: internal Network Size, Network type).

◦ Chance to have backpropagation, and maintain existing states to speed up BP, and include further functionality including Batch Normalization.

◦

◦ Chance for different edge correspondence,

• instead of each node representing an individual Node, they might blocks, with different



Structure

Items

Genes

Gene structure contains the Innovation number associated with the mutation.

• NodeGene\_t

◦ Innovation Number allows for each gene to be aligned up during crossover.

◦ Node Number

◦ Disabled Flag

• EdgeGene\_t

◦ Innovation Number allows for each gene to be aligned up during crossover.

◦ In/Out Nodes of the individual edge

◦ Disabled Flag

Innovation Number

https://miro.medium.com/max/700/0\*Kze4g6cLA3maofxq.png

Process

Concerns addressed by paper:

* Problem: Representations will not necessarily matchup: Some genomes can have different sizes. Other times, genes in the exact positions on different chromosomes may represent completely different traits. Additionally, genes expressing the same trait may appear at different positions on different chromosomes. How can these complications be resolved?

Solution: Homology: two genes are homologous if they are alleles of the same trait. For example, in E. coli, in a process called synapsis, a special protein called RecA lines up homologous genes between two genomes before crossover().

To answer this NEAT is the historical origin of tow genes is direct evidence of homology if the genes share the same origin. Therefore, NEAT performs artificial synapsis based on historical markings, allowing it to add now structure without losing track of which gene is which over the course of a simulation.” Kenneth O. Stanley and Risto Miikkulainen (2002).

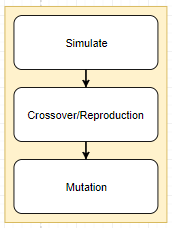
UML Control flow:

Species:

Problem: The problem with evolution programs is there is a limitation of unique solutions and unique specialization.

Solution:

Here is the general control flow of NEAT and many other Evolving simulations:



Steps:

1. Simulation/Selection

2. Crossover/Reproduction

3. Mutation

These steps iterate until either a specified level of performance/fitness has been achieved or

1. Selection

Select different Parents to propagate,

NEAT emphasizes having less evolved species within the environment to create new effective genes.

pseudo-code:

2. Crossover/Reproduction

+Align the two genes up against each other showing the different , because the networks have corresponding

https://miro.medium.com/max/700/0\*AlgDyVgxlUCOlE0B.png

https://miro.medium.com/max/700/0\*N4j\_sl8M05G6pXZV.png

A crossover takes two parents and “Crosses” them where we use a Dominant/Recessive chance to pass on different genetics to children nodes. So, for example here is some pseudo-code:

* If (Gene in both parents and differ slightly): Select the gene from the parent with the higher fitness score.
* Else: Child Gets Gene.

3. Mutation

A mutation has the chance to either create a new network structure or change a gene. Many times, this hurts the fitness for the network to decrease. For example, adding a new node introduces a non-linearity where there was none before; where the network needs to rebalance the different weights for each edge before finding a better fitness value. Protecting this innovation is important in order to try new structures before calling them off completely.

The system GNARL addresses this by adding a nonfunctional structure. A node is added to a genome without any connections, in the hopes that it the future some useful connections will develop. However, nonfunctional structures may never end up connecting and end up adding unnecessary complexity and memory.

Structure mutations:

* Add connection, a single new connection gene with a random weight is added connecting two previously unconnected nodes.
* “Add node, an existing node is split, and the new node placed where the old connection used to be. The old connection is disabled, and two new connections are added to the genome.”

In order to attempt combating this, GNARL proposes

• Create/Remove NodeGene\_t:

◦ Create: requires taking existing Edge and splitting in two.

◦ Remove: Simply remove Node from list

◦

• Increase/Decrease weight in Gene:

◦ Set Disabled flag to TRUE.

◦

Explicit fitness sharing, organisms in the same species must share the fitness of their Neche. There fore a species cannot afford to become too big even if many of its organisms perform well. This is designed to allow for diversity and prevent any one species from taking over.

This highlights the idea NEAT presents that innovation must be protected through specialization.

**Distance between species in a system: D~**

The final goal of the system, then, is to perform the search for a solution as efficiently as possible

C1,C2,C3 - Hyper parameters for Distance priorities  
E - a Linear combination of the number of excesses  
N- Number of Genes in the larger genome  
D - Disjoint genes  
W(Hat) - average weight differences of matching genes

Where: N is the number of genes in the larger genome, no and normalizes for genome size(N can be set to 1 if both genomes are small, and consist of fewer than 20 genes)

D~ allows us to measure the distance between species and the compatibility threshold./

+ “Each existing species is represented by a random genome inside the species from the previous generation. A given Ge in the current generation.”

The fitness for an organism is calculated according to its distance d~ from every other organism j in the population:

F’ = f\_(i) / (Sum(sh(d~(I,j))))

https://miro.medium.com/max/446/0\*hk8JqrWFbRiG04L2.jpg

Add new node steps:

1. If no edges exist, create one from input to output

2. Select an existing edge, and create a node from

**XOR and bias?**

“because XOR is not linearly separable, a neural network requires hidden units to solve it. The two inputs must be combined at some hidden unit, as opposed to only at the output node, because there is no function of very a linear combination of the inputs that can separate the inputs into the proper classes. These structural requirements make COR for adding new nodes might be too destructive to allow new nodes to get into the population.”

Pole balancing as a benchmark task:

*Pole Balancing,* Benchmark is a built simulation of a network attempting to balance a pole for as long as possible and is good to demonstrate the effectiveness of NEAT compared to others.

As explained “there are many potential problems that don’t contain known solutions, a ‘pole’ balancing domain for comparison is good as it ”

**Ablations** setup: can have a significant detrimental effect on performance, potentially to the point where the system cannot solve the task at all.

Limitations:

+ The current implementation of NEAT doesn’t allow for the large scaling of larger networks and doesn’t use more conventional techniques of machine learning such as Back Propagation.

* Potential proposal: create some mediation allowing for both backpropagation with larger networks and more advanced learning like Relu. This would require overcoming many different challenges, including but not limited to:
  + Searching Large search spaces without supervised learning. One solution could be an implementation of:
    - Alpha Go Zero()
    - World Model Learning model()
    - Reinforcement Learning()

Future Work:

+Take NEAT and modify it for larger neural networks, so each ‘Node’ is a Block of neurons; each network is trained with backpropagation.

+ Investigate other abstract machine learning methods, including the Neurological Turing machine().

+ Investigate/ create a method to protect edge weight values that are ‘well’ optimized?

Building:

1. Documentation
2. Python
3. C++/C with Open MP(Parallelized threads)

Terms:

GNARL: GeNeralization Acquisition of recurrent links, commenting that the…” the prospect of evolving connectionist networks with crossover appears limited in general.”

TWEANNs : Topology and weight evolving artificial neural networks. Note: These are different from NEAT allows for more complex structures than TEANN network structures.

CE: Cellular Encoding

Works Cited:

http://www.asimovinstitute.org/author/fjodorvanveen/

http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

Kenneth O. Stanley and Risto Miikkulainen (2002). "Evolving Neural Networks Through Augmenting Topologies". Evolutionary Computation 10 (2): 99-127