Lindsey Sample

COGS 319

Final Project

GridBot: A comparison of ANNs in Dynamic and Static Worlds

Introduction

Is there an optimal architecture for neuro-controllers? How do brain structures evolve with respect to the behaviors that they support? What are the tradeoffs between efficiency and full access to input information? Why are there different looking brains across species? What made them evolve that way? These are central questions in both Cognitive Science and Neuroscience, but it is difficult to address them given the complexity of animals. We can use models to examine hypotheses regarding brain architectures and the effect on fitness that variations can have. **Presently, we are interested in examining the differences in neural architectures that evolve in dynamic versus static environments in evolved artificial neural networks (ANNs).** Eventually, we hope to expand this project to include a comparison of ANNs and Weight Agnostic Neural Networks (WANNs) evolution facing conditions (Gaier & Ha, 2019; Stanley & Miikkulainen, 2002).

Often, Cognitive Scientists have used neural networks to model the brain of biological organisms. The standard neural net features a suitable architecture of nodes, and requires the development of appropriate weights between nodes that yield optimal output or performance. When exploring architectures of neurocontroller, a common method of describing and analyzing architecture is to examine modularity and sparsity (Cappelle, Bernatskiy, Livingston, Livingston, & Bongard, 2016). This attempts to quantify the degree to which the system is divided into modules, or is more fully-connected. Clune, Mouret, & Lipson (2013), define modular networks as those which "contain highly connected clusters of nodes that are sparsely connected to nodes in other clusters." Another way to conceptualize modularity is

to examine how discrete or integrated communication is within the network between input (sensors) and output (motors).

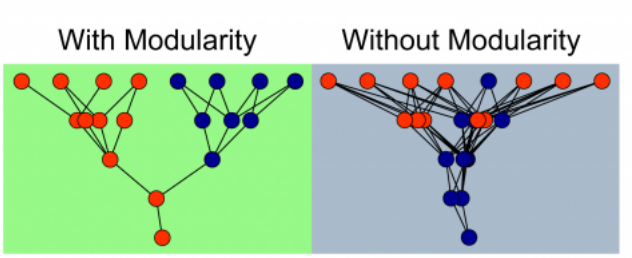

Figure 1. Depiction of ANNs with modularity and without modularity (Huizinga, Mouret, & Clune, 2014).

Modularity is believed to evolve from selection pressure **to reduce connection costs** (Clune et al., 2013). It is important to note the distinction that modularity does not evolve from selection pressure on performance alone, but specifically on pressure to reduce connection costs (Clune et al., 2013; Livingston et al., 2016).

The modularity of nervous systems is hypothesized to enhance evolutionary adaptation of a population by allowing selection to target regions separately. Previous research suggests modular networks are favorable for handling evolutionary goals. Livingston et al. (2016), suggests modularity can allow rapid response to environmental change, specialization without loss of useful sub-functions, and avoidance of "catastrophic forgetting."

Kashtan and Alon (2005) found that the evolutionary emergence of modularity was related to the environment faced and evolutionary goal experienced. They found that when goals were repeatedly switched (with common subgoals), the networks rapidly evolved to satisfy the different goals with only a few rewiring changes. They claim that these evolutionary forces favor modularity for its structural simplicity and ability to rapidly adapt (Kashtan and Alton, 2005). They found that when the varying goals

contained no common sub-goals, modular structures did not evolve and that adaptation was very slow (since evolution was essentially starting from scratch each time there was a change).

The majority of research on this topic points to the claim that cognitive architectures develop with respect to their specific environments, or the tasks required of them. Modularity evolves with respect to pressure to reduce connection costs in environments, and is a strategy for easily evolving to changing environments. This leads us to the hypothesis that modularity will be favored in dynamic environments, more-so than in static environments. We have developed a model to test this hypothesis.

We will test this hypothesis with a model featuring a simulated robot called GridBot. This model will compare the evolution of ANNs in a dynamic environment, and a static environment. GridBot‘s specific task is to traverse a GridWorld that features a light source and obstacles, which vary, in order to change what is expected of GridBot in the changing environment condition, and are stagnant in the stable condition. GridBot features simulated versions of two photoresistor sensors (LDRs), two infrared sensors (IRs), and one bumper, and navigates GridWorld autonomously, making movement decisions based on the values inputted and computed in its ANN. We will analyze and compare the best performing emergent architectures in the static and dynamic conditions.

Methods

GridBot and GridWorld. Presently, we develop a simulated world to emulate a toy world for a simulated bot to traverse. This bot, called GridBot, has two light dependent resistors (LDRs), two infared sensors (IRs), and one bumper on its front. It can traverse a grid world by moving forward or backward, or by turning to the left or right. Each step, GridBot can make one move: stepping, or turning. GridBot can make 100 moves in a trial, and move backwards, move forwards, turn clockwise, or turn counterclockwise on any given move. Each move is determined probabilistically based on its ANN, featuring 5 input nodes corresponding to its 5 sensors, and 4 output nodes, corresponding to its 4 move options.

GridWorld is a grid, composed of a 10x10 (x,y) matrix (100 squares) in the stable condition, and a randomly determined NxN matrix (also equal to 100 squares) in the dynamic condition (4x25, 25x4, 10x10, 20x5, or 5x20). Every square within the grid contains a value of light that can be collected by entering that square, and a binary value that indicates if there is an obstacle in that square. In the static condition, there are 5 obstacles that form a square in the center of the GridWorld. In the dynamic condition, there are 5 randomly placed obstacle squares. GridBot cannot step into a square that an obstacle inhabits, and cannot collect light from these squares. Light values are highest near the source, and fall off in a gradient (255*e^(-distance*.5)) across the GridWorld. In the static condition, the light source resides at the top of the world (5,5). In the dynamic condition, the light source is in the center top as well, but this depends on which dynamic grid the bot is randomly in (see **Figure 2)**.

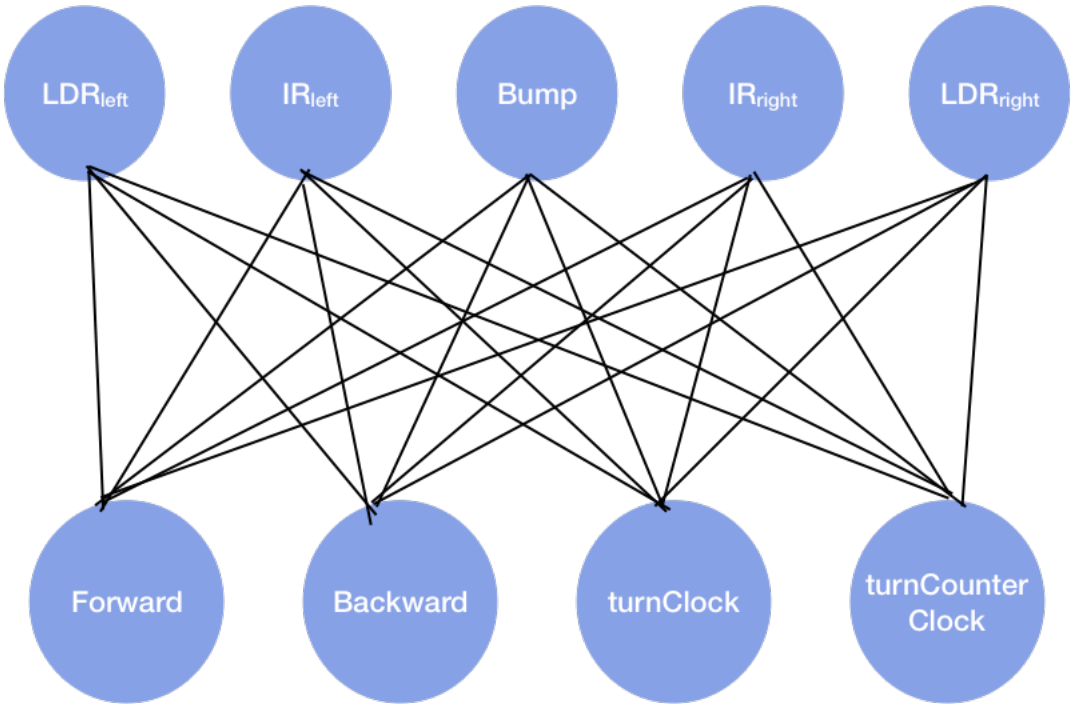

Figure 2. Depiction of GridBot's ANN

ANNs. Each GridBot has a weight matrix, where weights can take on values of -1, -.5, 0, .5, and 1. On each turn, each output receives a value that is the dot product of the weight matrix (or the bot's genome), and the input. The outputs are also scaled using a sigmoid function (1 / (1 + x^(-sum(input*weight[,i])) and divided by the sum of outputs (to appropriately scale them for probabilistically choosing between them), and an output node is probabilistically chosen. This action is then taken, sensors are updated, and the process repeats for 100 moves. At the end of a trial, a GridBot has the amount of light it has collected accumulated.

Sensors and Action. IRs are calculated by searching the four spaces in front of the bot as depicted in **Figure 3**. The spaces are searched in order, and if an object is detected in the space, the IR value becomes the distance between the bot's current location in the grid and the obstacle.

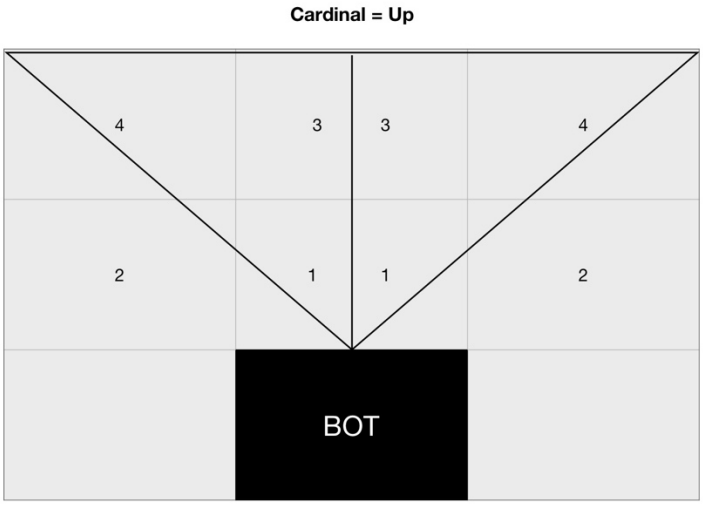

Figure 3. Specifications for LDR sensors (right and left).

LDRs are calculated by taking the average of the space the bot is currently in and the 3 adjacent spaces (depicted in **Figure 4**). Each LDR assumes the light value of the space it is sensing according to the specifications described.

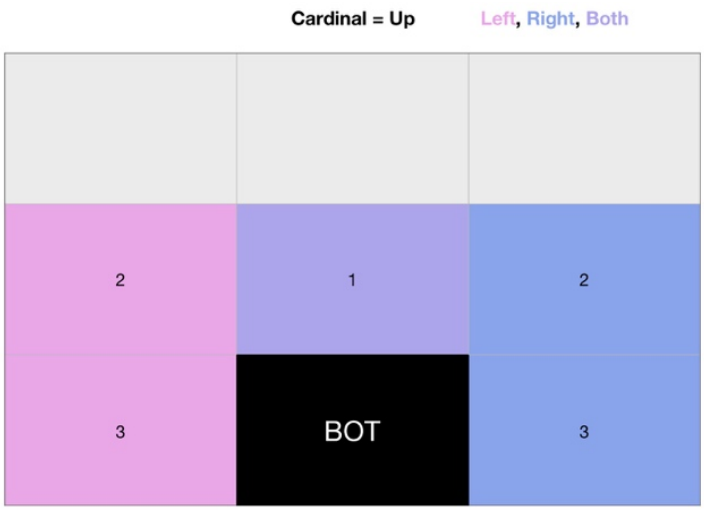

Figure 4. Specifications for LDR sensors.

GridBot also features a bumper, that returns 1 when there is an object directly in front of where GridBot is facing, and 0 when the space is unoccupied.

Selection. GridBot survives by harvesting light. Each move, GridBot's light counter is updated by adding the light it has collected by moving into a new square, and taking the mean of the light sensed in the square it is in, averaged with the 3 adjacent squares to the right (for the right sensor) and left (for the left sensor). The total light collected over a trial (100 moves) will serve as that bot's fitness, where the most fit bots collect the most light.

The evolutionary algorithm uses a roulette style wheel algorithm to select the next generation of GridBots based on relative fitness of the previous generation. This evolutionary algorithm is taken from Haddow & Tufte (1999), and depicted in **Figure 5.** In this example, individual number 1 has 20% percent of the roulette wheel whereas individual number 4 has only 7% percent of the wheel. As such, individual 1 is more likely to be selected than individual number 7. The selection mechanism "spins the wheel" or in our case, uses the built-in sample function (with replacement) to select selects individuals. The wheel has to be 10 times to select 10 individuals to retain the size of the population in the new generation.

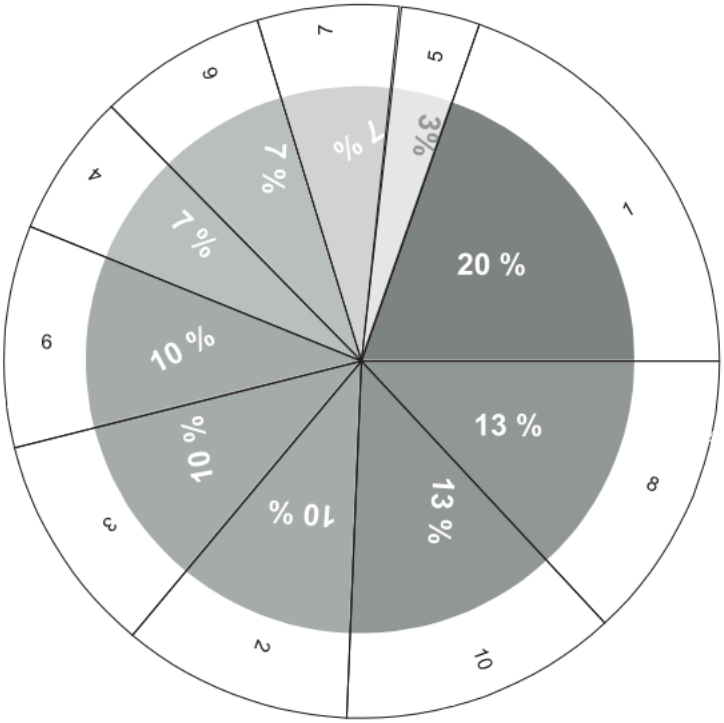

Figure 5. Roulette style wheel for selection, based on relative fitness (Haddow & Tufte, 1999).

The corresponding bot numbers are selected, and each bot's weight matrix is mutated at a rate of 5%. This means 1/20 weights are changed at random (from the list of possible values: -1, -.5, 0, .5, and 1). The experiment is run for 100 generations, with 10 bots per generation, and 100 moves per bot, per trial. The experiment is run in the dynamic and simulated condition.

Code. This model is implemented in R using RStudio. The code for this model can be accessed on the public GitHub repository: https://github.com/lpsample/LSample_COGS319. The complete model can be found in the file entitled: "GridBot Model.Rmd." The functional modularity calculations, visuals for this report, and all previous drafts can be found in the repository as well.

Functional Modularity. Functional Modularity (FM) is calculated to compare different ANNs of bots. FM is determined as any of the sets of the minimum number of edges needed to connect one sensor input to one motor output. For the common three-layer NN, the total number of possible circuits in the NN is the product of the number of nodes in each layer. The following approach was adapted with the guidance of John Long and Ken Livingston in 2018.

Our measurement for node-specific functional modularity, F_{ns}, is for each node, and is calculated as follows:

$$F_{ns} = 1 - \left\{ \left[\frac{\sum(|w_i|)}{\sum(|w_{max}|)} \right] \left(\frac{N_p - N_a}{N_p - 1} \right) \right\}$$

where, for all circuits in which the node is involved w_i is the weight of each edge; w_{max} is the maximum possible weight of each of those edges; N_p is the number of possible circuits for that node; and N_a is the actual number of circuits for that node. For a three-layer neural network, the N_p for any input node is the product of the number of hidden layer nodes and the number of output layer nodes; for any output node the N_p is the product of the number of hidden layer nodes and the number of output layer nodes. A circuit is defined as a path from an input node to an output node.

For the purposes of our model, $N_p = 4$ for any input node and $N_p = 5$ for any output node. We will take the average of every node's F_{ns} to determine the overall functional modularity for that network.

Functional modularity is one of many measures by which we can examine the differences between ANNs, and while we are using this measure for this project due to time constraint, we hope to expand our analysis in the future.

Results and Analysis

Basic Analysis. Our results yielded two weight matrices depicted below in **Tables 1 and 2**, and in a visualization in **Figure 6**. The ANNs featured some similarities, and some differences. First, the static condition's ANN featured 3 connection weights of 0, while the dynamic condition featured 7. In terms of our hypothesis, this supports the theory that dynamic conditions produce more modular architectures.

Static Condition

	Forward	**Backwards**	**TurnClock**	**TurnCounter**
leftLDR	0.5	-1	0	-1
leftIR	-0.5	0	0	0.5
Bumper	1	0.5	1	-0.5
rightIR	-1	1	1	-0.5
rightLDR	-0.5	-0.5	-0.5	-1

Dynamic Condition

	Forward	**Backwards**	**TurnClock**	**TurnCounter**
leftLDR	1	-1	0	-0.5
leftIR	-1	-0.5	0	0
Bumper	0	1	1	1
rightIR	-1	1	-1	0
rightLDR	0	0.5	-0.5	0

Tables 1 and 2. Best Performing Evolved Weight's for GridBot in Static (top) and Dynamic (bottom) Conditions

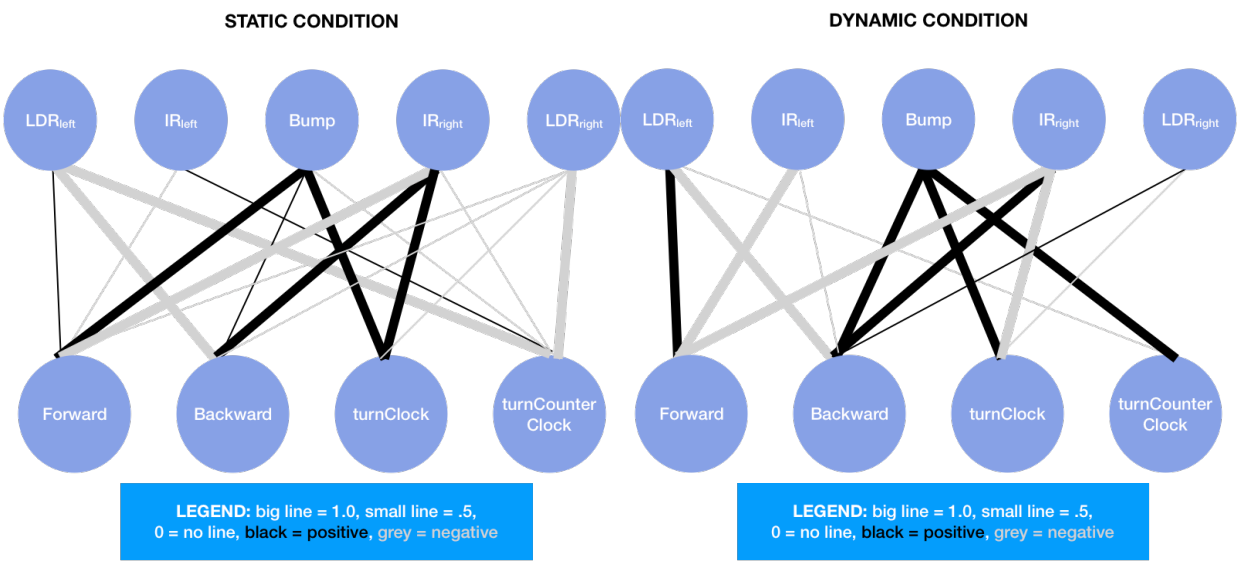

Figure 6. Visualization of GridBot's Evolved, Best Performing Static and Dynamic ANNs

The ANNs both feature similar numbers of strong connections (weight = 1.0), but different numbers of weak connections (see **Table 3**):

	Strong (1.0)	Weak (0.5)	Excitatory (+)	Inhibitory (-)	Zero (0.0)
Static	8	8	7	10	3
Dynamic	9	4	5	7	7

Table 3. Breakdown of connection types in Static and Dynamic conditions

The ANNs had a few trends in common. Both models featured an excitatory connection between the left LDR and the forwards move, and no connection to the turn-clockwise move. Both also featured inhibitory connections between the left LDR and the backwards and turn-counterclockwise functions, despite their unique magnitudes. In the left IR, both feature inhibitory connections with the forward move, and no connection to backward. For the Bumper, both feature excitatory connections for the backwards node, and

strong excitatory connections for the clockwise node. For the right IR, both condition's weights feature a strong inhibitory connection for the forward move and strong excitatory connection for the backwards move. Both feature strong connections for the clockwise move but in opposite directions. Finally, for the right LDR, both feature a weak connection in opposite directions for the backwards node, and a weak negative connection for the clockwise node.

In both conditions, the only excitatory connection to the forwards node comes from the left LDR. We might have expected a connection to emerge from the right LDR as well, and next steps might include looking into why this did not occur. The bumper excited move backwards in both conditions, and at least one turn action, which logically, we could expect to occur. All IRs in both conditions featured inhibitory connections towards the forward node, which could be expected as well.

Functional Modularity. We found that the static condition had an average node-specific functional modularity of 0.856, where the dynamic condition had an average F_{ns} of 0.609 (see **Table 4)**. We are unfortunately unable to determine the appropriate statistical analysis of these values to determine their significance. We hope to conclude that these values confirm a difference in modularity that evolves between static and dynamic environments.

STATIC	INPUT NODES FNS		OUTPUT NODES FNS	
LEFTLDR	0.5	**Forward**	1	
LEFTIR	1	**Backward**	1	
BUMPER	1	**TurnClock**	0.25	
RIGHTIR	1	**TurnCounter**	1	
RIGHTLDR	1			**Total**
AVERAGE	0.9		0.8125	0.85625

DYNAMIC				
LEFTLDR	0.83333333	**Forward**	0.5	
LEFTIR	0	**Backward**	1	
BUMPER	0	**TurnClock**	0.75	
RIGHTIR	0.66666667	**TurnCounter**	0.625	
RIGHTLDR	1			**Total**
AVERAGE	0.5		0.71875	0.609375

Table 4. Node-Specific Functional Modularity Calculations by Node and Averaged for Static (top) and Dynamic (bottom) Conditions.

Next Steps and Reflection

Next Steps. Immediate next steps include reconfiguring the model to easily report more information on what occurred in our best performing trials, which world yielded higher overall fitness, and implementing better analyses and statistical tests that tell us more information about the reproducibility and magnitude of the evolutionary findings. For example, if we ran this many times, would our weights come out the same? This will perhaps lead us to augment our sample size and trials run.

Longer term next steps include implementing a WANN to compare to the results of these ANNs in the same experiment, and finding new (better) ways to compare and analyze the resulting neural architectures.

Reflection. This project was a very cool opportunity to explore a curiosity of mine and undertake a large coding operation. While I started very early and scheduled weekly appointments to get help, I still ran out of time to fully complete the project well. Specifically with respect to analysis, I wish I had planned out my analysis better up-front. I spent most of my time jumping into the coding, which did require a lot of planning and time as well, but even with a running model, without proper means of analysis, a model's

purpose cannot be well fulfilled. This was my biggest learning from this project. I truly ran out of time to get help with deciding which statistical tests needed to be run, and this would have been better to decide up-front.

My second learning was in code planning. Though I had a well-planned experimental design, my code design was figured out as I went, which resulted in many mistakes. For example, we used R, which does not allow global variables, and required us to pass in and out of functions a list that contained "global variables." But, during the run of the experiment, these variables got tossed aside and the only information outputted became the most-fit weight matrices. I wish I had done a better job of anticipating all of the data I would want to analyze, like the grid, which would be interesting to analyze and compare bot paths. Likewise, I wish I had accumulated more data on the dynamic condition's path, in terms of how many "randomly sized grids" were of each type.

I am very interested in understanding the brain, how individuals' brains differ and cause those individuals to differ, and how these differences evolve. I am increasingly curious and excited about this project, and very much enjoyed the freedom and lack of limitations that a modeling approach offers. (especially with respect to sample size!). I am far more optimistic and excited about the use of models in scientific inquiry as a result of what I have learned throughout this semester and from this project. Additionally, I hope to continue this project through my thesis, and one day publish this work.

References

Cappelle, C. K., Bernatskiy, A., Livingston, K., Livingston, N., & Bongard, J. (2016). Morphological modularity can enable the evolution of robot behavior to scale linearly with the number of environmental features. *Frontiers Robotics AI*, *3*(OCT), 1–10. https://doi.org/10.3389/frobt.2016.00059

Clune, J., Mouret, J. B., & Lipson, H. (2013). Summary of the evolutionary origins of modularity. *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*

Companion, 23. https://doi.org/10.1145/2464576.2464596

Gaier, A., & Ha, D. (2019). *Weight Agnostic Neural Networks*. (NeurIPS), 1–19. Retrieved from http://arxiv.org/abs/1906.04358

Haddow, P., & Tufte, G. (1999). Evolving a robot controller in hardware. *Proc. of the Norwegian Computer Science Conference*, (November), 141–150. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.27.7078&rep=rep1&type=pdf

Huizinga, J., Mouret, J. B., & Clune, J. (2014). Evolving neural networks that are both modular and regular: Hyperneat plus the connection cost technique. *GECCO 2014 - Proceedings of the 2014 Genetic and Evolutionary Computation Conference*, 697–704. https://doi.org/10.1145/2576768.2598232

Livingston, N., Bernatskiy, A., Livingston, K., Smith, M. L., Schwarz, J., Bongard, J. C., … Long, J. H. (2016). Modularity and sparsity: Evolution of neural net controllers in physically embodied robots. *Frontiers Robotics AI*, *3*(DEC), 1–16. https://doi.org/10.3389/frobt.2016.00075

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, *10*(2), 99–127. https://doi.org/10.1162/106365602320169811