Embodied Embeddings for Hyperneat

Collin Cappelle[1], Josh Bongard[1]

[1] The University of Vermont, Burlington, VT 05405
collin.cappelle@uvm.edu

Abstract

A long time goal of evolutionary roboticists is to create ever-increasing lifelike robots which reflect the important aspects of biology in their behavior and form. One way to create such creatures is to use evolutionary algorithms and genotype to phenotype maps which act as proxies for biological development. One such algorithm is HyperNEAT whose use of a substrate which can be viewed as an abstraction of spatial development used by Hox genes. Previous work has looked into answering what effect changing the embedding has on HyperNEAT's efficiency, however no work has been done on the effect of representing different aspects of the agents morphology within the embeddings. We introduce the term *embodied embeddings* to capture the idea of using information from the morphology to dictate the locations of neurons in the substrate. We further compare three embodied embeddings, one which uses the physical structure of the robot and two which use abstract information about the robot's morphology, on an embodied version of the retina task which can be made modular, hierarchical, or a combination of both.

Introduction

The ultimate goal of evolutionary artificial intelligence and automated machine research is to have the complex forms and behaviors of animals reflected in the agents created by algorithms. Modularity, hierarchy, and regularity are important factors to consider when creating agents which reflect the complexities seen in biology (Carroll, 2001; Schlosser and Wagner, 2004; Hartwell et al., 1999; Bongard and Pfeifer, 2001).

In mammalian fetal development, HOX genes are activated at different times in development based on the strength of chemical gradients physically present in the morphology of the organism (Duboule, 1998; Krumlauf, 1994). In one example, Hox genes at the beginning of the genome are activated first near the anterior end of the organism while Hox genes at the end of the genome are activated later in development along the posterior of the organism. This allows development to easily generate symmetry and structure based on a local interaction of chemicals. This type of development helps generate the repeated, hierarchical, and modular structures present in a biological agents body.

Indirect encodings which incorporate development in some manner can help artificially act as development similar to how Hox genes behave during mammalian development(Stanley et al., 2009; Bongard, 2002). Indirect encodings also have the benefit of only needing to optimize a set of parameters existing in a smaller dimension than a counterpart direct encoding. Instead of choosing the weight of each synapse, an indirect encoding instead provides rules or functions for how the weights should be set (Gauci and Stanley, 2010). Evolution then only has to optimize the rules for determining the weights and not the weights themselves meaning indirect encodings can be applied to arbitrarily tasks with arbitrarily large networks without increasing the search space evolution actually exists in. Indirect encodings can also introduce regularity into the phenotype of the agent, similar to what is found in natural agents (Clune et al., 2011). If the rules used by the indirect encoding contain symmetry, it is likely symmetry will be reflected in the final individual. In the effort to create more natural evolved robots being able to codify natural development processes in the generation of said robots is an important step.

HyperNEAT, explained in more detail in the next section, is a direct encoding which acts as a proxy for development for artificial agents. Similar to Hox genes, HyperNEAT takes locality into account in its genotype to phenotype map. We call an *embodied embedding* when the locality present in the substrate is based on some aspect of the morphology of the agent. We provide evidence that different embodied embeddings provide differences based on the objective the robot is tasked to perform.

HyperNEAT

HyperNEAT is a genetic algorithm specialized in creating gradients and patterns along substrates (Stanley et al., 2009). When used on neural networks, Hyperneat uses Compositional Pattern Producing Networks (CPPNs) to determine the synaptic weight between two neurons by taking in the embedded locations of the source and target neuron as input. The output of the CPPN is then used to determine the weight of the synapse. HyperNEAT uses the NEAT algorithm to de-

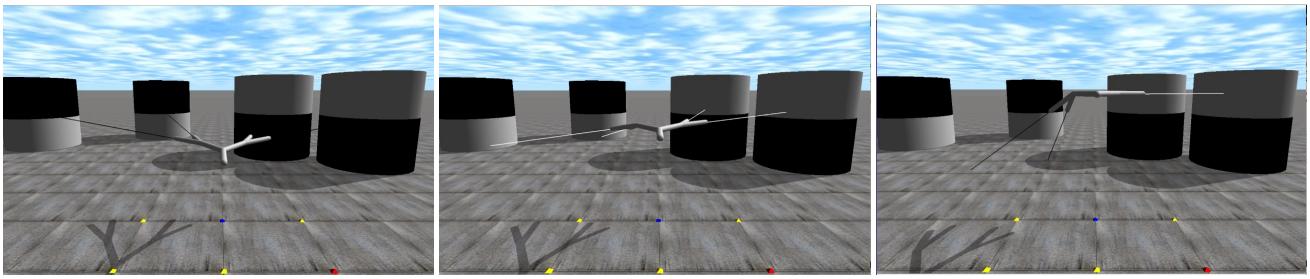

Figure 1: A depth 2 robot in an environment consisting of two Far cylinders on the left and two Near cylinders on the right. The robot is tasked with a local and a global objective. The local objective consists of pointing at the white portions of the cylinders while the global objective consists of moving its root node upwards because there are an even number of each type of cylinder. The initial position of the robot is shown on the left. At the midpoint of evaluation (middle image) the robot is correctly pointing at the correct portions of the cylinders. Rays coming out of the robots leaves are purely graphical to help indicate where the robot is pointing. Towards the end of simulation (right) the robot has completed the global task at the cost of half of the local task.

termine the structure and topology of the CPPN. The initial population for NEAT is a simplistic network directly connecting inputs to outputs with randomly determined weights. Over evolutionary time the network is made more complex by adding nodes and edges to the network, as well as changing the weights of the edges. The activation functions of the nodes are chosen from a predetermined list of functions which can be chosen to reflect the relevant regularity needed in the specific task at hand. NEAT further uses diversity preservation techniques which give new genotypes a chance to optimize to the task before removing them from the population due to fitness.

HyperNEAT has been shown to be effective in a wide range of domains which require regularity from determining weights in neural networks, to painting images, and determining an agent's morphology (DAmbrosio et al., 2014).

There are many variants and modifications one can make to HyperNEAT. The one we use throughout the experiments in this paper is HyperNEAT with Link Expression Output (HyperNEAT-LEO). HyperNEAT-LEO uses two outputs to the CPPN, one to determine the weight of the synapse in question and one to determine whether the synapse is expressed or not in the final neural network (Verbancsics and Stanley, 2011). By allowing evolution to control whether the synapse is expressed, HyperNEAT-LEO can control the final topology of the produced neural network and explicitly dictate the presence of modules. To further imbue the concept of modularity and locality for HyperNEAT-LEO, the initial population of CPPNs can be seeded with Gaussian nodes which are activated only when input neurons are close to eachother in their embedded substrate. Thus synapses are initially much more likely to be expressed if the neurons are close in embedded space.

This type of control over synaptic expression can be useful in tasks in which modularity is necessary (Verbancsics and Stanley, 2011). One such task is the Retina Task detailed later in the paper. So far the majority of tasks which use HyperNEAT-LEO have been disembodied networks in which modularity is critical or necessary to correct completion of the task (Verbancsics and Stanley, 2011; Huizinga et al., 2014). In this work we apply HyperNEAT-LEO to an embodied agent evolved in task environments that are modular and hierarchical.

Substrate Analysis

Clune et al. (2009) showed that the configuration of the substrate can impact fitness and efficiency when using HyperNEAT. The authors applied different embodied embeddings for a quadruped. In one experiment, the embeddings differed by the dimension of the geometric representation. The authors tested 1,2, and 3-d representations for the neural network controlling the quadruped, the idea being that the 3-d representation more encompasses the actual symmetries and structure present in the physical robot. They showed that the 2 and 3-d representations resulted in similar performance meaning that this increase in dimensionality did not help or hinder HyperNEAT's ability to evolve a walking gait in the quadruped. Their work did not use information about the abstract structure and relationships between components in the morphology in determining the embedding.

Further work in the impact on of embedding locations has been focused on evolvable-substrate HyperNEAT (ES-HyperNEAT) which evolves the location of neurons in the substrate as well as the CPPN (Risi et al., 2010; Risi and Stanley, 2011). While these methods have shown to have been effective on benchmark problems, when the designer of the substrate is given a body in which to physically place neurons, it allows the morphology to dictate the structure of the controller.

Retina Task

The retina task is an inherently hierarchical and modular task. Two retinas, on the left and right, are fed into a neural network. The modular aspect of the retina task is to distinguish whether each retina contains a target pattern or not. The hierarchical aspect of the retina task is to then take whether each retina is a target pattern or not as a logical input and compute a function on that input, like NAND.

The original goal of the retina task was to show that modularly varying goals causes evolution to generate modular networks whereas fixed goals tend to generate nonmodular networks (Kashtan and Alon, 2005). In their initial paper Kashtan and Alon (2005) used direct encodings to construct the neural networks topology and synaptic weights. By changing the logical function the network needed to compute periodically, they were able to generate networks which exhibited left-right modularity.

Clune et al. (2010) then used the same retina task using HyperNEAT to specify the neural network. They found that HyperNEAT performed much more poorly than the direct encodings used previously as well as finding that the solutions produced by HyperNEAT were not modular.

In response, Verbancsics and Stanley (2011) showed that by using Hyperneat-LEO modular solutions were found to the retina task when the initial population of Central Pattern Producing Networks (CPPNs) used by HyperNEAT-LEO were seeded with an explicit concept of locality. This work further showed that other versions of HyperNEAT including dynamic threshold and LEO without seeding were less effective in generating modular networks which were fit to the required task.

Further modularity has been shown to be evolved by using connection cost along with HyperNEAT-LEO (Huizinga et al., 2014). The authors evolved networks on a variety of variants of the disembodied retina task.

Every work using the retina task has done so in a disembodied way. We present an embodied version of the retina task in which the robot must physically move in order to respond correctly to what it senses in the environment. This movement then changes how the robot perceives its environment causing its sensation of the environment to change. In this manner seemingly modular tasks can have extremely effective non-modular solutions. In future work we plan to examine how adding connection cost can aid modularity in the embodied retina task presented later in the paper.

Methods

Robot Construction

The robot was a planar tree structure consisting of an actuated root and 2^d actuated leaves where d is the depth of the tree. Hence, at $d = 2$ there are two actuated leaves (Fig. 2). Each leaf consisted of a distance sensor pointing out from the tree into the environment and a motor which actuates a hinge joint connecting the leaf to its parent branch. The hinge joint allowed the leaf to rotate up and down with respect to its parent branch. The root consisted of a motor which actuated a linear joint moving the entire tree along the z-axis (up and down). Joint ranges in the leaves were limited to $\pm\frac{\pi}{4}$ from their starting position and the root node could move ± 1 units from its starting position. Each branch was $1/2$ unit long with the root base starting at the point $(0, 0, 1.5)$

$$n_i^{(t)} = \sigma\left(n_i^{(t-1)} + \sum_{j \in J} w_{ji} n_j^{(t-1)}\right) \quad (1)$$

Neuron activation in the robot was controlled using Equation 1. The value of the i^{th} neuron, n_i, at time step t was equal to the value of n_i in the previous time step plus the sum of the incoming synaptic weights multiplied by the corresponding neuron's value. The σ in the Eq. (1) is the hyperbolic tangent function.

Embodied Retina Task

The goal of the embodied retina task is to perceive objects in the environment and react accordingly. Similar to the its disembodied counterpart the embodied retina task requires aspects of modularity and hierarchy in the controller of the agent.

The task environment consisted of cylinders placed along a semi circle four units (Near) and six units (Far) away from the origin. Near cylinders were white on top, black on the bottom and far cylinders were colored black on top and white on bottom. The cylinders were placed such that each leaf of the robot was pointing at the middle of its corresponding cylinder (Fig. 1).

There were two objectives for each robot: local and global. The local objective was to have leaves point at the white region of their corresponding cylinders. The global objective was to determine if there was an even or odd number of Near cylinders. The robot had to respond by moving the root, and thus the whole tree, up if there was an even number and down if there was an odd number of Near cylinders.

For example, given a depth $d = 1$ robot and the the environment {Near, Near}, the robot should move its root up while the leaves of the robot point up towards the tops of the cylinders. Further, given the environment {Near, Far}, the global solution will be to move the root down, while the local solutions will be to point to the top of the left (Near) cylinder and the bottom of the right (Far) cylinder.

The number of cylinders in the environment is dependent on the number of leaves in the tree which is further dependent on the depth of the tree. Specifically, the number of cylinders, n, is $n = 2^d$. Each cylinder had two variants. Thus, at depth $d = 1$ there are two cylinders meaning there are $2^2 = 4$ total environments for the robot to be evaluated

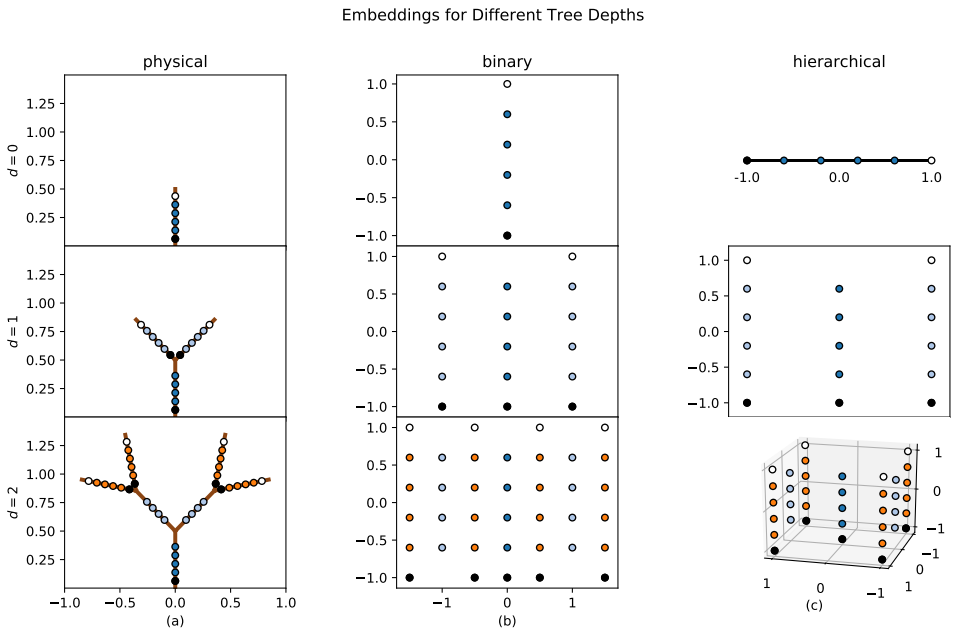

Figure 2: The embeddings shown at three different depths of the tree. White neurons indicate the distance sensor neurons. Sensor neurons are placed in the leaves of the tree and point outward in the direction of the leaf they are contained in. Black neurons indicate motor neurons and are placed in the leaves and the root of the tree. The remaining neurons are hidden neurons colored according to their depth for ease of comparison across embeddings. Each branch consisted of four hidden neurons. Both the physical **(a)** and binary **(b)** embeddings exist in 2D-space regardless of the depth of the tree and produce valid embeddings (i.e. no overlapping neurons) for arbitrary depths of the tree. The dimension of the hierarchical embedding **(c)** grows as the size of the tree grows. Note at $d = 1$ the hierarchical and binary embedding are exactly the same.

in. For depth $d = 2$ there are four cylinders giving $4^2 = 16$ total environments for the robot to be evaluated in.

$$\text{eval}(c_e, t) = \begin{cases} 1 & \text{if pointing at white region of} \\ & \quad c_e \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$

$$g_e = |z_{\text{target}} - z_{\text{root}}| \tag{2}$$

$$\ell_e = 1 - \frac{\sum_{t=T/2}^{T} \sum_{c_e \in C_e} \text{eval}(c_e, t)}{\ell_{e,\max}} \tag{3}$$

$$Err(\alpha, E) = \sum_{e \in E} (\alpha g_e + (1-\alpha)\ell_e)^2 \tag{4}$$

The global objective, calculated in Eq. (2) as g_e, is primarily hierarchical. g_e is the absolute difference between the target z-location (1.5 if there is an even number of each cylinder, 0.5 otherwise) and the ending z-location of the root at the final time step. A human designer would possibly create a network where information would flow from the leaves to the root where the robot would then aggregate the information to create the correct response.

The local objective, calculated in Eq. (3) is primarily modular. ℓ_e is found by assessing whether the robot is looking at the white portion of the cylinders in the last half of the evaluation. By expanding out evaluation to the final half of simulation we allow evolution to create a more steady gradient to the optimal solution. At each time step during evaluation, the robot is given a point if it is correctly looking at the white portion of the cylinder and a 0 if it is not. These points are then summed for each cylinder and normalized between $[0, 1]$ and subtracted from 1 to give the error. In correctly assessing a cylinder, each leaf would benefit from having an isolated module which determines whether to move the leaf branch up or down as appropriate.

The overall objective function is a mean squared error consisting of a weighted sum of the error from the two tasks. From these two objectives we explored three tasks determined by Eq. (4) using $\alpha = \{0.0, 0.5, 1.0\}$. These values correspond to focusing on only the local objective, both objectives combined, and only the global objective, respectively.

Embodied Network Embeddings

Each embedding consisted of four hidden neurons per physical tree branch. The embeddings are embodied because each neuron is placed using information from the morphology of

the robot. The physical embedding uses the actual (x, y) positioning of the robot in its initial state whereas both the binary and hierarchical embeddings use information about the branches location in the abstract tree structure. Each embedding consisted of a number of motor and sensor neurons dependent on the depth of the tree considered. Each leaf branch contained one sensor neuron and one motor neuron. The root contained a single motor neuron. Thus for $d = 1$ there where $3 * 4 = 12$ hidden neurons 2 sensor neurons and 3 motor neurons giving 17 total neurons. For $d = 2$ there where $7 * 4 = 28$ hidden neurons 4 sensor neurons and 5 motor neurons giving 37 total neurons.

The path of a branch is determined by whether the branch is a left or right child or an ancestor. Thus, the path of each branch is a list with length d, the depth of the tree. The elements of this list are chosen from $\{-1, 0, 1\}$. A -1 indicates the branch is a left child, $+1$ indicates a right child, and 0 indicates the branch is an ancestor to branches in that depth. For example, given a $d = 2$ tree, the leftmost leaf is $[-1, -1]$ because it is left child of the left child of the root. In contrast root's path is $[0, 0]$ because it is the ancestor of both depth one and two branches.

In the physical embedding, neurons were placed along the branches of the physical robot. Thus each neuron had a physical location which corresponded to the robots initial starting position in physical (x, y) space. The physical embedding is in two-dimensional space, regardless of the the depth of the robot.

In the binary embedding, the neurons were placed according to the location of its corresponding branch in the overall tree structure. The x position of the neurons are placed using information about the path and depth of the current branch. Neurons $x = \sum_{i=1^d} p_i * \frac{1}{i}$ where p_i is the i^{th} index of the path p. This results in neurons in branches of left children being placed to the left of neurons in the parent branches and, conversely, neurons in right child branches are to the right of their parents. The neurons in the root branch are located at $x = 0$. The y coordinate of the binary embedding was chosen to be linear spacing between $[-1, +1]$. Motors were placed at $y = -1$, sensors at $y = +1$ and hidden neurons were linearly spaced in between. In branches without sensors or motors, hidden neurons were still placed as if they existed, meaning hidden neurons from each branch shared y coordinates. This embedding was chosen because it encompasses information about the morphology of the tree, specifically left-right symmetry, while also being extensible to different depths of tree morphologies and remaining in a 2-d embedding. The embedding is shown in more detail in Figure 2b.

The hierarchical embedding (Fig. 2c) is similar to the binary embedding in that it uses information about the branches location to determine the position of the embedded neurons. However, instead of placing neurons in a 2-d embedding, the dimension hierarchical embedding grows with the depth of the tree, specifically dim $= d + 1$. Each new depth of the tree is a new dimension for the embedding with variations in the positioning at that depth, according to the path of the branch, corresponding to location in that dimension in the embedding. For example, if a branches path in a depth 2 tree is [-1, 1], the corresponding (x, y) embedding for neurons in that branch are $(-1, 1)$ with the z coordinate being determined by the same linear interpolation between $[-1, +1]$ as seen previously. In general, the first d coordinates of the hierarchical embedding are determined by the path with the final coordinate being determined by the linear interpolation. This means at $d = 1$ the binary and hierarchical embeddings are exactly the same. Further, the distance between in neurons in the hierarchical embedding is reflective of the path distance of the branches within the tree. This means a child is closer to its parent than its sibling and branches with the same parent are closer than branches with different parents.

	bin/hier	physical
$\alpha = 0.0$	**0.0152****	0.0185
$\alpha = 0.5$	**0.0208**	0.0245
$\alpha = 1.0$	**0.125**	0.140

Table 1: Depth 1 average minimum fitness at generation 1000. Bolded values indicates minimum across row. **** indicates $p < 0.0001$ according to Mann-Whitney U test.

	binary	physical	hierarchical
$\alpha = 0.0$	0.0600	0.0675	**0.0548**
$\alpha = 0.5$	0.0699	0.0632	**0.0588** *
$\alpha = 1.0$	0.227	0.239	**0.222**

Table 2: Depth 2 average minimum fitness at generation 1000. Bold values indicate minimum across row. $*$ indicates significance at $p < 0.05$ according to Kruskall-Wallis H-Test.

Experimental Parameters

Robots were simulated using Pyrosim, a python interface for Open Dynamics Engine [1]. We used the same parameters for HyperNEAT as in Verbancsics and Stanley (2011) with the exception of changing the population size to 100 due to the computational cost of simulation. Robots were evaluated for 100 time steps in each environment.

The initial population of CPPNs were seeded as in Verbancsics and Stanley (2011). Because we considered two dimensional substrates, two Gaussian nodes were used and the bias was connected to the LEO output of the CPPN with a -2. This seed means that two neurons which are close together in the embedded xy-plane are much more likely to be connected by HyperNEAT.

[1] https://ccappelle.github.io/pyrosim/

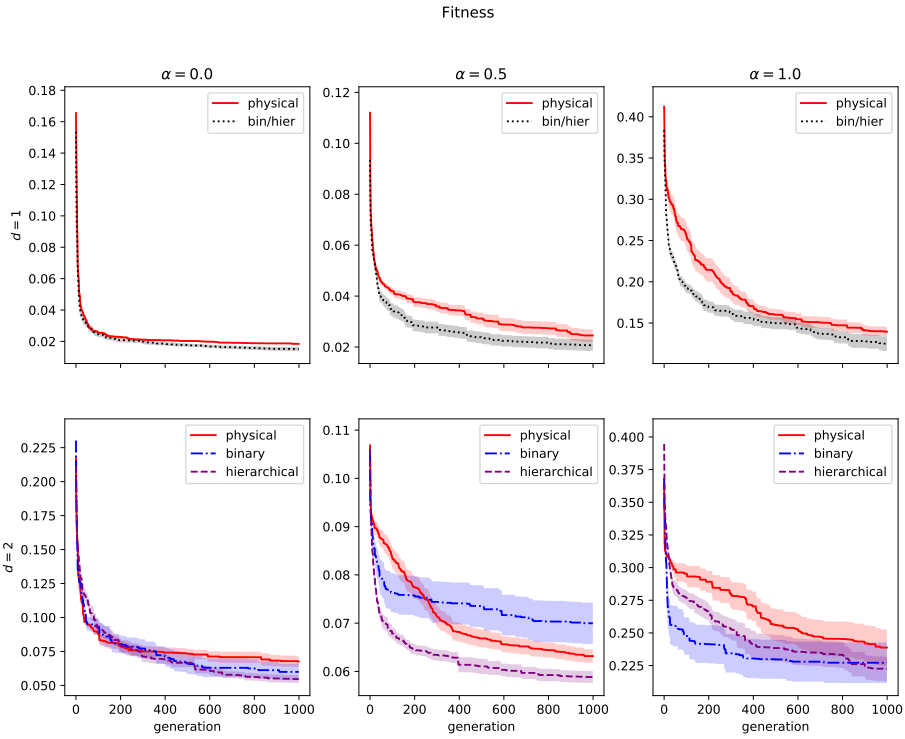

Figure 3: The average minimum error for the three embeddings in the six experiments performed over generational time. The top row contains the average minimum error for depth 1 trees with four total test environments. The bottom row contains the average minimum error for depth 2 trees with 16 total test environments. Each column indicates a different α parameter. α is used to tune the error from the local objective, $\alpha = 0.0$, to the global objective ,$\alpha = 1.0$, as dictated by Equation 4. Shaded regions indicate $\pm$ SEM. Only the $d = 1, \alpha = 0.0$ and $d = 2, \alpha = 0.5$ results are significant with $p < 0.05$ according to the Kuskal-Wallis H-test.

Results

We ran 30 trials of HyperNEAT-LEO for both the physical and binary embedding for $\alpha \in \{0.0, 0.5, 1.0\}$ with $d \in \{1, 2\}$ for 1000 generations each. The results are presented in Figure 3. Amongst the six experiments, two resulted in population in a significant difference in ending average error. In the $d = 2, \alpha = 0.5$ experiment, the physical embedding proved to have significantly better fitness after 1000 generations ($p < 0.05$). Conversely, in the $d = 2, \alpha = 1.0$ experiment, the binary embedding proved to be significantly better error than the physical embedding ($p < 0.05$). All other experiments provided statistically similar ending fitness values for both embeddings.

For every value of α in the Depth 1 experiments, the binary/hierarchical embedding performed better than the physical embedding over 1000 generations however only the $\alpha = 0.0$ results are significant. Complete reporting on ending error of Depth 1 experiments is located in Table 1.

For every value of α in the Depth 2 experiments, the hierarchical embedding performed better than both the physical and binary embeddings over 1000 generations however only the $\alpha = 0.5$ results are significant.

Modularity in the form of network modularity was not present in any of the ending champion networks, every network was completely connected. Regularity was found to be present and varied between the different encodings as seen in Figure 4.

Discussion

The location of neurons embedded in the substrate used by HyperNEAT is known to have a difference in the efficiency of evolution to optimize to both embodied and disembodied tasks (Clune et al., 2009). By using different *embodied embeddings*, HyperNEAT is able to set connection weights using locality and gradients, present in the morphology, which may stress the importance of certain desirable traits in robot controllers such as hierarchy and modularity. Figure 4 shows a clear difference in the types of patterns that are more common given a physical embedding, one where the robots morphology is directly reflected in the location of the neurons,

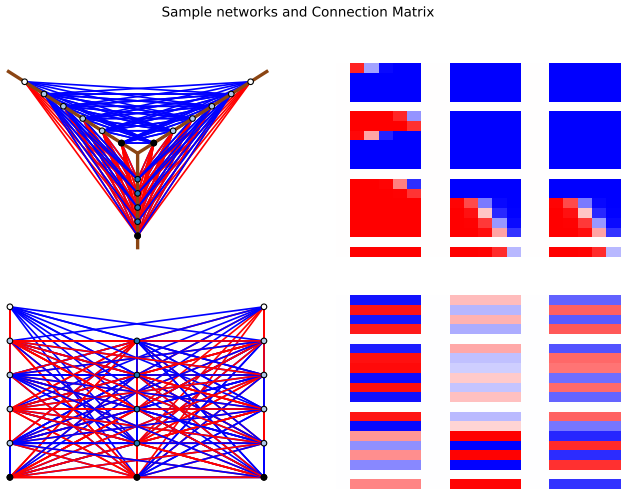

Figure 4: The best run champions from the $\alpha = 0.5, d = 1$ experiment. The physical embedding is shown on the top row and the binary/hierarchical embedding is shown on the bottom row. The left column shows how the network is placed on the embedding and the right column is the same network in adjacency matrix form. Red connections indicate negative synaptic weights while blue indicates positive weights and the alpha of the connection indicates the magnitude of the weight. The adjacency matrix (right) helps show how the positioning of nodes in the embedding impacts the type of connection structure which occurs. The white separations are sensors which cannot be connected to by synapses. The separations help further distinguish the neurons within each branch and how they connect to the neurons in other branches

compared to a different type of embodied embedding which uses information about the abstract concept of the structure of the morphology.

Figure 3 shows the hierarchical embedding was able to perform better than other embeddingns on a complex task which had both global and local objectives. The increases in dimensionality of the hierarchical embedding helped it compared to the similar binary embedding which uses the same concept of a branches path to determine neuron location but restricts the embedding to two dimensions. These differences are important because it gives an indication as to the nature of the relationship between the morphology of the robot, the structure of the task at hand, and the embedding used.

One important aspect of HyperNEAT is that it is resolution independent meaning it is likely that solutions found are able to scale in a predictable manner (Gauci and Stanley, 2010). Here we give some insight into how embodied embeddings may be able to be scaled effectively. In either embedding presented in this work, more hidden neurons can easily be placed by using linear spacing between the end points set by the sensor and motor neurons. In the physical embedding this takes place near the physical (x, y) position of the tips of each branch while in the binary embedding this occurs at $x \in \{-1, +1\}$. Further resolution increases can occur for this robot by increasing the depth of the tree. In other robots this can be thought of as adding different components or sensors and incorporating them into the substrate as prescribed by the emobodied embedding plan. Figure 3 shows that while increasing the depth had a impact on the overall error achieved it is important to note that the number of environments between $d = 1$ and $d = 2$ was squared. The increase in depth helped elucidate potential problems with each embedding at these higher depth dimensions.

The disembodied retina task is known to be a benchmark in order to create modularity in evolved neural networks (Kashtan and Alon, 2005; Clune et al., 2010). There are many potential reasons as to why modularity did not form in the experiments performed. One could be that there are plenty of perfectly acceptable non-modular controllers in this task even though, to a human designer, the task seems separable and modular. Another reason could be that there was not enough pressure for modularity to form. Kashtan and Alon (2005) only consistently found modularity when the global task the network needed to compute was changed over the course of evolution. This changing every few generations pressured evolution to separate the network into left and right halves. It is possible that for this task, in order for modularity to be present, one would need to change the global objective periodically.

Another way to further constrain connectivity is to explicitly choose for solutions which have less connections

through applying a connection cost function. The simplest function to represent cost simply counts the total number of connections present in the network, however, given the tree structure of the robot, we can direct evolution towards hierarchical solutions by assigning each neuron a physical corresponding branch and computing the path length from the branch of one neuron to the branch of the other. In this manner we could select for controllers which explicitly use a hierarchical structure.

Lastly, it is possible modularity did not occur because this is an embodied task in which speed of movement may play a factor. The more heavily connected a motor neuron is the more likely it is that it will actuate with a higher magnitude velocity. This higher velocity can help the robot achieve its target position more quickly resulting in higher fitness.

Conclusion

In this work we presented the term *embodied embedding* and presented two ways in which it could be performed. One simply took the physical morphology of the robot in space to inform the construction of the substrate. The other used the abstract notion of the robot's structure to create the embedding. Both were able to perform in tasks, that to a human, seem modular and hierarchical. We showed that differences in these embeddings can cause differences in the evolvability of the robots. We further gave insight into the patterns of connections created by certain embeddings and how they can create different networks to complete the same task.

In the future we would also like to investigate if ES-Hyperneat chooses hierarchical embeddings for hierarchical tasks and whether connection cost could help produce modularity in the embodied retina task.

Acknowledgements

This work was supported by National Science Foundation award INSPIRE-1344227. The computational resources provided by the UVMs Vermont Advanced Computing Core (VACC) are gratefully acknowledged.

References

Bongard, J. (2002). Evolving modular genetic regulatory networks. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1872–1877. IEEE.

Bongard, J. C. and Pfeifer, R. (2001). Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 829–836. Morgan Kaufmann Publishers Inc.

Carroll, S. B. (2001). Chance and necessity: the evolution of morphological complexity and diversity. *Nature*, 409(6823):1102.

Clune, J., Beckmann, B. E., McKinley, P. K., and Ofria, C. (2010). Investigating whether hyperneat produces modular neural networks. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 635–642. ACM.

Clune, J., Ofria, C., and Pennock, R. T. (2009). The sensitivity of hyperneat to different geometric representations of a problem. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 675–682. ACM.

Clune, J., Stanley, K. O., Pennock, R. T., and Ofria, C. (2011). On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 15(3):346–367.

Duboule, D. (1998). Vertebrate hox gene regulation: clustering and/or colinearity? *Current opinion in genetics & development*, 8(5):514–518.

DAmbrosio, D. B., Gauci, J., and Stanley, K. O. (2014). Hyperneat: The first five years. In *Growing adaptive machines*, pages 159–185. Springer.

Gauci, J. and Stanley, K. O. (2010). Indirect encoding of neural networks for scalable go. In *International Conference on Parallel Problem Solving from Nature*, pages 354–363. Springer.

Hartwell, L. H., Hopfield, J. J., Leibler, S., and Murray, A. W. (1999). From molecular to modular cell biology. *Nature*, 402(6761supp):C47.

Huizinga, J., Clune, J., and Mouret, J.-B. (2014). Evolving neural networks that are both modular and regular: Hyperneat plus the connection cost technique. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 697–704. ACM.

Kashtan, N. and Alon, U. (2005). Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences of the United States of America*, 102(39):13773–13778.

Krumlauf, R. (1994). Hox genes in vertebrate development. *Cell*, 78(2):191–201.

Risi, S., Lehman, J., and Stanley, K. O. (2010). Evolving the placement and density of neurons in the hyperneat substrate. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 563–570. ACM.

Risi, S. and Stanley, K. O. (2011). Enhancing es-hyperneat to evolve more complex regular neural networks. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1539–1546. ACM.

Schlosser, G. and Wagner, G. P. (2004). *Modularity in development and evolution*. University of Chicago Press.

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212.

Verbancsics, P. and Stanley, K. O. (2011). Constraining connectivity to encourage modularity in hyperneat. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1483–1490. ACM.