Combining Multiple Inputs in HyperNEAT Mobile Agent Controller

Jan Drchal[1], Ondrej Kapral[1], Jan Koutník[2], and Miroslav Šnorek[1]

[1] Computational Intelligence Group,
Department of Computer Science and Engineering,
Faculty of Electrical Engineering,
Czech Technical University in Prague
+420 224 357 470, fax: +420 224 923 325
{drchaj1|kaprao1|snorek}@fel.cvut.cz,
[2] IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland
+41 58 666 6669, fax: +41 58 666 6661
hkou@idsia.ch,

Abstract. In this paper we present neuro-evolution of neural network controllers for mobile agents in a simulated environment. The controller is obtained through evolution of hypercube encoded weights of recurrent neural networks (HyperNEAT). The simulated agent's goal is to find a target in a shortest time interval. The generated neural network processes three different inputs – surface quality, obstacles and distance to the target. A behavior emerged in agents features ability of driving on roads, obstacle avoidance and provides an efficient way of the target search.

1 Introduction

Exhaustive preprocessing techniques are usually used in design of controllers for artificial agents (robots). Environment sensors such as cameras, radars etc. with possibly high resolution in space and time domain are used and their outputs are utilized to perform the desired task.

Our goal is to generate robotic controllers based on recurrent artificial neural networks trained with evolutionary algorithm. Recurrent neural networks [1] are capable of effective temporal information processing because feedback connections form a short term memory within the networks. Such controllers can express more complex behavior.

There are many options how to transform preprocessed sensory input to actions that the robot performs in order to fulfill goals. Artificial neural networks can play the role of a such controlling system. In artificial neural networks the dimensionality of the sensory input was the obstacle that blocked direct processing of e.g. camera images. To overcome this limitation, we use a hypercube encoding of neural network weights [2], which allows to increase input vector as well as amount of artificial neurons in the networks.

Hypercube encoding allows the large-scale neural networks to be effectively encoded into population of individuals. A single genome size does not grow with

the number of neurons in the network. Similarly, a resolution of the network inputs can be extended without growth of the network genome. This is the property of HyperNEAT algorithm used.

HyperNEAT algorithm was introduced in [2] and [3]. It is an evolutionary algorithm able to evolve large-scale networks utilizing so called generative encoding. HyperNEAT evolves neural networks in a two step process: the NEAT (see below) is used to create networks combining a set of transfer functions into a special function. The transfer functions allow to encode symmetry, imperfect symmetry and repetition with variation. The composed functions are called the Compositional Pattern Producing Networks (CPPNs). In the second step, planned neurons are given spatial coordinates. The previously evolved CPPN is then used to determine synaptic weights between all pairs (or subset of pairs) of neurons. The coordinates of both neurons are fed into the CPPNs inputs, the CPPN then outputs their connection weight. The weight is not expressed if its absolute value is below a given threshold. Such connectivity pattern created by CPPN is called the substrate. The important feature of HyperNEAT substrate is that it can be scaled to higher resolutions approximately preserving its inner structure and function.

NEAT (NeuroEvolution of Augmenting Topologies) [4] is an algorithm originally developed for evolution of both parameters (weights) and topology of artificial neural networks. It was extended to produce the CPPNs in the HyperNEAT algorithm instead of producing the neural networks directly. It works with genomes of variable size. NEAT introduced a concept of historical markings, which are gene labels allowing effective genome alignment in order to facilitate crossover-like operations. Moreover, historical markings are used for computation of a genotypical distance of two individuals. The distance measure is needed by niching evolutionary algorithm, which is a core of the NEAT. Because NEAT evolves networks of different complexity (sizes) niching was found to be necessary for protection of new topology innovations. The important NEAT property is the complexification – it starts with simple networks and gradually adds new neurons and connections. For evolving CPPNs, NEAT was extended to evolve heterogeneous computational units (nodes).

1.1 Related Work

Evolution of artificial neural networks is a robust technique for development of neural systems. Many techniques were developed for evolution of either weights or even a structure of neural networks like e.g. Analog Genetic Encoding [5–7], Continual Evolution Algorithm [8], GNARL for recurrent neural networks [9], Evolino [10] and NeuroEvolution of Augmenting Topologies (NEAT) [4]. The NEAT algorithm became a part of HyperNEAT algorithm as a tool for evolution of CPPNs.

HyperNEAT algorithm was already applied to control artificial robots in a food gathering problem [2]. A robot with a set of range-finder sensors is controlled to approach the food. It was shown that HyperNEAT is able to evolve very large neural networks with more than eight million connections. A very interesting

property of the HyperNEAT is the ability to change a resolution of the substrate. For example 11×11 grid was resized to 55×55 while preserving the underlying neural network function. In the food gathering experiment the inputs indicating whether the food is in a particular direction were arranged parallel or concentric with the robot body. Each sensor was geometrically linked with an effector, which drives the robot.

Our approach differs in the organization of the input sensors, which are arranged in polar rays having particular angular and distance resolution. The sensors are sensitive to color of the surface and in fact represent a camera with arbitrary pixel resolution.

In [11] HyperNEAT algorithm was applied in a very efficient way so that each agent shares a portion of the substrate and neural network. The neural network splits to local areas in the substrate geometrically but all agents share a single substrate. This can be exploited in agents' cooperative behavior.

In [12] it is shown that robots can complete common goals with a minimum information coming from sensors. The robots are controlled by evolved feed-forward neural networks.

In [13] we shown that the HyperNEAT is capable to generate neural networks that can keep the agents stay and drive on roads. Further more, we replaced the NEAT in HyperNEAT with Genetic Programming [14] with comparable results.

In our approach, we reduced an effort typically required to build hardware robotic platforms such as described in [12]. We moved directly to a simulation to concentrate on development of the robot's control algorithms. First, we created a simulation environment described in Section 2.1. The environment allows a rapid development and an experimentation with simulated robots.

This paper is organized as follows. In the next section a simulation environment and a robot setup is described. Section 3 describes the experimental results. A final section concludes the paper.

2 Experimental Setup

2.1 Simulation Environment

Experiments were performed in a simulation environment called ViVAE (Visual Vector Agent Environment) featuring easy design of simulation scenarios in a SVG vector format [13]. There are two types of surfaces in the simulation (a road and a grass) with different frictions. The grass has a friction 5 times higher than the road. Additionally, solid unmovable and movable objects can be placed into the simulation environment. In the current experiments, we used the fixed objects only.

ViVAE supports number of different agents equipped with various sensors for surfaces and other objects in the scenario.

ViVAE allows easy snapshoting of the whole simulation into a sequence of SVG frames. All agents can be tracked and their tracks recorded as a SVG path displayed as a simulation result, see Figure 2.

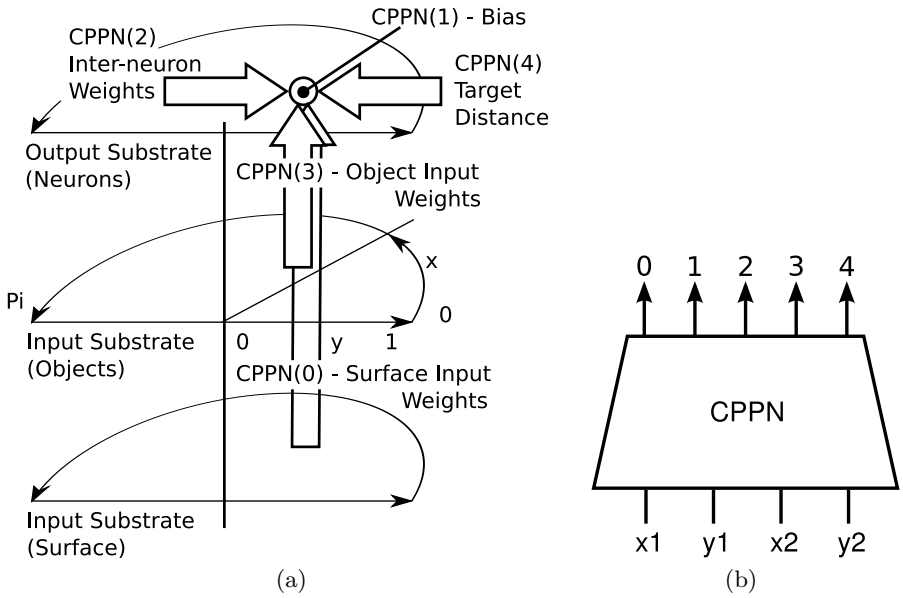

Fig. 1. Organization of the HyperNEAT substrate. There are three distinct substrates used (a) and the CPPN has 5 outputs (b). CPPN(0) output is a weight between input surface substrate and a neuron in the upper substrate. Second, CPPN(1) output is used as bias for neurons in the upper substrate. For a bias calculation the third and the fourth CPPN inputs are set to 0. Third, CPPN(2) output represents connection weights among neurons in the upper substrate. Fourth, CPPN(3) output represents connection weights between input object substrate and neurons in the upper substrate. Last, CPPN(4) output is used for weights between distance to target input variable and neurons in the upper substrate. In this case, first two CPPN inputs are set to 0.

2.2 Agent Setup

The agent is driven by two simulated wheels and is equipped with sensors of three different types. The controlling neural network is organized in a single layer of possibly fully interconnected perceptron (global) type neurons (neurons compute biased scalar product, which is transformed by a bipolar logistic sigmoidal function). Steering angle is proportional to an inverse actual speed of the robot.

The sensors as well as the neural network are spread in a substrate. Neurons and sensors are addressed with polar coordinates, see Figure 1. Two of the neurons in the output substrate are dedicated to control acceleration of the wheels.

During a simulation, an agent is controlled by a neural network controller constructed using HyperNEAT. The neural network neurons and connections are mapped into three substrates. The CPPN has 5 outputs. Three outputs are used for obtaining weights among the neurons and between neurons and inputs from the input substrates (CPPN outputs 0, 2 and 3). One CPPN output is used to set up neurons biases (CPPN output 1). The last CPPN output (4) determines a weight of a connection between distance to target input and particular neuron in the neurons substrate.

The substrate resolution was chosen to be 5 polar rays of 3 sensors in both input layers and 3×3 neurons in the layer of neurons.

2.3 HyperNEAT setup

We have used our own implementation of the HyperNEAT algorithm. The NEAT part resembles Stanley's original implementation. The HyperNEAT extension is inspired mainly by the David D'Ambrosio's HyperSharpNEAT[1]. Table 1 shows CPPN node functions.

Table 1. CPPN node functions

Name	**Equation**
Bipolar Sigmoid	$\frac{2}{1+e^{-4.9\,x}} - 1$
Linear	x
Gaussian	$e^{-2.5\,x^2}$
Absolute value	$\|x\|$
Sine	$sin(x)$
Cosine	$cos(x)$

[1] Both Stanley's original NEAT implementation and D'Ambrosio's HyperSharpNEAT can be found on http://www.cs.ucf.edu/ kstanley.

The parameter settings are summarized in Table 2. Note, that we have extended the original set of constants which determine the genotype distance between two individuals (C_1, C_2 and C_3) by the new constant C_{ACT}. The constant C_{ACT} was added due to the fact that, unlike in classic NEAT, we evolve networks (CPPNs) with heterogeneous nodes. C_{ACT} multiplies the number of not matching output nodes of aligned link genes. The CPPN output nodes were limited to bipolar sigmoidal functions in order to constrain the output.

Table 2. HyperNEAT parameters

Parameter	**Value**
population size	100
CPPN weights amplitude	3.0
CPPN output amplitude	1.0
controller network weights amplitude	3.0
distance threshold	15.0
distance C_1	2.0
distance C_2	2.0
distance C_3	0.5
distance C_{ACT}	1.0
mating probability	0.75
add link mutation probability	0.3
add node mutation probability	0.1
elitism per species	5%

3 Experimental Results

Experimental results described in Section 3.1 were intended to learn the agents to drive on roads instead of grass surface, which has 5 times greater friction than a road.

3.1 On Road Driving

In this experiment the agent controller used three substrates (surface input, neurons and biases) only. The scenario contained no obstacles. And there were 5 agents in the simulation performing concurrently. The agents had no particular

target to find. Instead, the agents were trained to gain a maximum average speed in the simulation, according to the following fitness function:

$$f_1 = \frac{distanceTraveled}{simulationSteps + 1} \tag{1}$$

Fitness function f_1 is an average speed of the simulated robots. The 1 is added to prevent a division by zero. The speed is a meaningless variable (number of pixels per a simulation step) but can be computed in a straightforward way and is suitably proportional.

Figure 2 shows a final solution which was found in a generation 324. The trajectories are smooth. Moreover, robots learned to drive on a one side of the road to avoid mutual collisions. The complete experiment is described in [13].

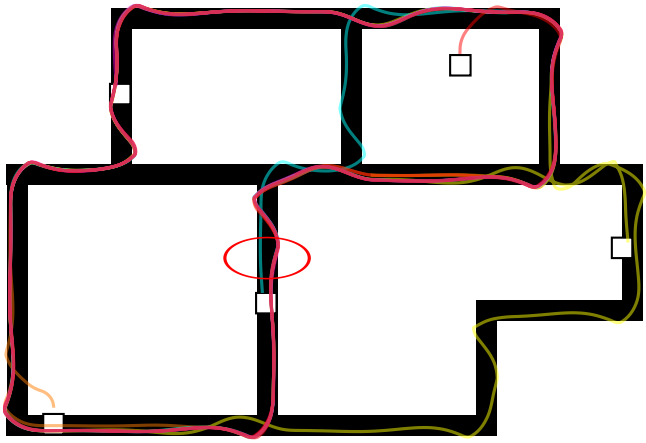

Fig. 2. Trajectories of the robots controlled by a neural network found in a generation 324. The trajectories are smooth. The robots learned how to drive on a one side of the road (as emphasized by the red ellipse).

3.2 Obstacle Avoidance

In this experiment the agents were equipped with an additional substrate for connections generated by a CPPN output number 3. The last CPPN output controls weights of connections to input containing sign of the actual target distance difference. The fitness function is the following one:

$$f_1 = \frac{distanceTraveled}{simulationSteps + 1} \left(1 - \frac{targetDistance}{initialTargetDistance}\right) \tag{2}$$

The fitness from the previous experiment is multiplied by a relative distance to the target. Agents that find the target faster are preferred to those, which drive on road but do not approach the target.

The controller performances obtained in the evolution are depicted in Figure 3. Agent trajectories evolved in generations 2, 30, 209 and 300 are depicted in a single scenario. We can see how the path was precised between generations 200 and 300. The final controller controls the agent motion to be straight on straight roads.

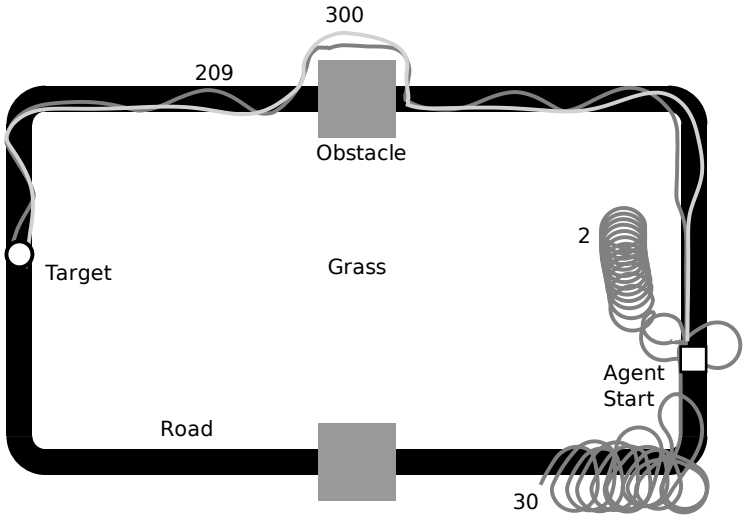

Fig. 3. Obstacle avoidance. This figure shows how an obstacle avoidance emerges during an evolution. There are trajectories of an agent controlled by the best controller found in a particular generation. We can see that after 300 generations of the evolution run, the agent can successfully drive around the obstacle and return to the road to reach the target. The path improved in comparison to generation 209 in which the controller moves the agent periodically from one side of the road to the other one.

The trained controller is capable of the generalization as can be seen in Figure 4. The agent follows a border of an obstacle, returns to the road and continues the ride.

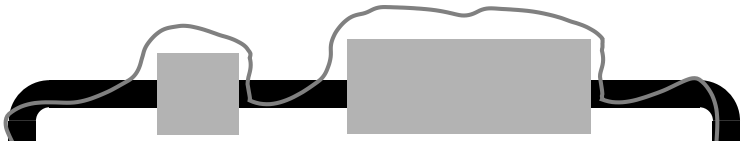

Fig. 4. Generalization Example. The agent trajectory is depicted. The agent follows border of an obstacle and returns to the road afterwards.

4 Conclusion

The aim of the experiments was to verify whether the HyperNEAT trained neural network can learn to control the agent based on multiple inputs. The presented experiments show that the HyperNEAT trained neural network controller can process multiple inputs and utilize them to drive the agent in order to maximize its fitness during the evolution. We used two input layers (substrates). One layer represented surfaces, the second layer represented solid unmovable obstacle sensors. An additional neural network input contains a relative difference in actual distance to the target. Beyond previous experiments, the agents are capable to bypass solid obstacles and drive in a direction to the target. The fitness was the agent average speed multiplied by a relative distance to the target reached. The agents learned to follow the roads in a direction to the target. The agents trails were improved during the evolution to be more smooth and straight on straight roads.

Further experiments should discover dependencies of the controller capabilities on a density of the input substrates as well as a coevolution among agents in a population, possibly in a different more complex task than approaching a single target.

5 Acknowledgement

This work has been supported by the research program "Transdisciplinary Research in the Area of Biomedical Engineering II" (MSM6840770012) sponsored by the Ministry of Education, Youth and Sports of the Czech Republic and partially by Humanobs EU Project (#231453).

References

1. Elman, J.L.: Finding structure in time. Cognitive Science **14**(2) (1990) 179–211
2. D'Ambrosio, D.B., Stanley, K.O.: A novel generative encoding for exploiting neural network sensor and output geometry. In: GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, New York, NY, USA, ACM (2007) 974–981
3. Gauci, J., Stanley, K.: Generating large-scale neural networks through discovering geometric regularities. In: GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, New York, NY, USA, ACM (2007) 997–1004
4. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation **10** (2002) 99–127
5. Mattiussi, C.: Evolutionary synthesis of analog networks. PhD thesis, EPFL, Lausanne (2005)
6. Dürr, P., Mattiussi, C., Floreano, D.: Neuroevolution with Analog Genetic Encoding. In: Parallel Problem Solving from Nature - PPSN iX. Volume 9 of Lecture Notes in Computer Science. (2006) 671–680

7. Dürr, P., Mattiussi, C., Soltoggio, A., Floreano, D.: Evolvability of Neuromodulated Learning for Robots. In: The 2008 ECSIS Symposium on Learning and Adaptive Behavior in Robotic Systems, Los Alamitos, CA, IEEE Computer Society (2008) 41–46
8. Buk, Z., Šnorek, M.: Hybrid evolution of heterogeneous neural networks. In: Artificial Neural Networks - ICANN 2008. Volume 5163., Springer Berlin / Heidelberg (2008) 426–434
9. Angeline, P.J., Saunders, G.M., Pollack, J.B.: An evolutionary algorithm that constructs recurrent neural networks. IEEE Transactions on Neural Networks **5** (1993) 54–65
10. Schmidhuber, J., Wierstra, D., Gagliolo, M., Gomez, F.: Training recurrent networks by evolino. Neural computation **19**(3) (March 2007) 757–779
11. D'Ambrosio, D.B., Stanley, K.O.: Generative encoding for multiagent learning. In: GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, New York, NY, USA, ACM (2008) 819–826
12. Waibel, M.: Evolution of Cooperation in Artificial Ants. PhD thesis, EPFL (2007)
13. Drchal, J., Koutník, J., Šnorek, M.: HyperNEAT controlled robots learn to drive on roads in simulated environment. In: Accepted to IEEE Congress on Evolutionary Computation (CEC 2009). (2009)
14. Buk, Z., Koutník, J., Šnorek, M.: NEAT in HyperNEAT substituted with genetic programming. In: Accepted to International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2009). (2009)