In *Proceedings of the Genetic Evolutionary Computation Conference (GECCO 2004)*

Transfer of Neuroevolved Controllers in Unstable Domains

Faustino J. Gomez and Risto Miikkulainen

Department of Computer Sciences, University of Texas, Austin, TX, 78712 USA

Abstract. In recent years, the evolution of artificial neural networks or *neuroevolution* has brought promising results in solving difficult reinforcement learning problems. But, like standard RL methods, it requires that solutions be discovered in simulation and then be transferred to the real world. To date, transfer has been studied primarily in mobile robot problems that exhibit stability. This paper presents the first study of transfer in an unstable environment. Using the double pole balancing task, we simulate the process of transferring controllers from simulation to the real-world, and show that the appropriate use of noise during evolution can improve transfer significantly by compensating for inaccuracy in the simulator.

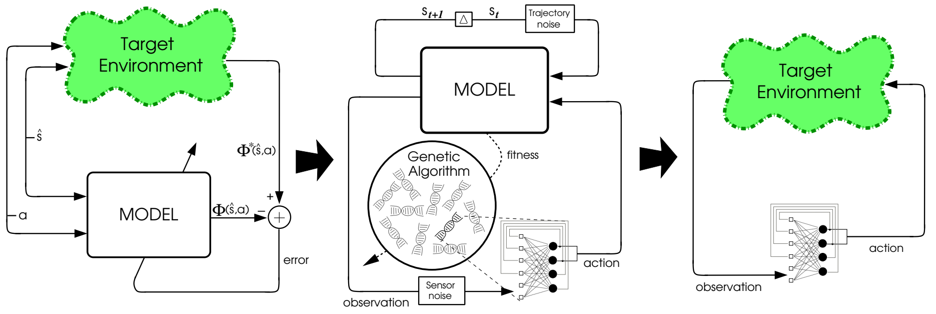
1 Introduction

Neuroevolution (i.e. methods that evolve artificial neural networks) can potentially be used to automatically design controllers for domains such as autonomous robotics and aerospace where designing controllers by more conventional means can be very difficult. Unfortunately, evaluating populations of candidate controllers through direct interaction with real world systems is too time consuming and often dangerous. Instead, controllers must first be evolved in a *simulation environment* that approximates the real-world *target environment*, and then be transferred to that the target environment. In order for transfer to be successful, the controllers must be robust enough to tolerate discrepancies that inevitably exist between the two environments. Otherwise, a controller that performs well in simulation may perform poorly, or not at all, when placed in the real world.

While there is a substantial body of work addressing transfer in the mobile robot domain [1,2,3,4,5,6], almost all transfer experiments have been conducted with robots (e.g. Khepera) and environments that are relatively "transfer friendly." Most significantly, robots like the Khepera are stable: in the absence of a control signal the robot will either stay in its current state or quickly converge to a nearby state. Consequently, a robot can often perform competently in the real world as long as its behavior is preserved qualitatively after transfer. This is not the case with a great many systems of interest such as rockets, aircraft, and chemical plants that are inherently unstable. In such environments, the controller must continuously output a precise control signal to maintain system equilibrium and avoid failure. Therefore, controllers for unstable systems may be less amenable to techniques that have worked for transfer in robots.

In this paper, we study the problem of transferring controllers for unstable, non-linear systems with the aim of providing a more general understanding of the transfer process. This paper represents the first attempt to systematically study the transfer of neuroevolved controllers outside of the mobile robot domain.

The next section describes the methodology used to study transfer, section 3 presents our experimental results, section 4 analyzes the robustness of transferred solutions, and in section 5 we provide some analysis of our results. Sections 6 and 7 contain a broader discussion of transfer and future directions, and our conclusions, respectively.



(a) Learn model (b) Evolve using model (c) Transfer controller

Fig. 1. The model-based neuroevolution approach. (a) Step 1: Learn a model of the target environment; in this case, the double pole balancing system. The model (e.g. a supervised neural network) receives the a state $\widehat{s}$ and an action a as its input and produces a prediction of the next state $\Phi(s, a)$ as its output. The error between the prediction and the correct next state $\Phi^*(s, a)$ is used to improve the prediction. (b) Step 2: Evolve a controller using the model instead of the target environment. The boxes labeled "trajectory noise" and "sensory noise" add uniform noise to the signal passing through them. These noise sources make controllers more robust and facilitate transfer. (c) Step 3: Test the controller on the target environment.

2 Transfer Methodology

For the purpose of studying transfer, we have chosen double pole balancing as the test domain because it embodies the essential elements of unstable systems while being simple enough to study in depth, and it has been studied extensively, but in simulation only. The problem consists of balancing two poles that are hinged to a cart, riding on a finite stretch of track, by applying a force to the cart at regular intervals. This problem is challenging because the interaction between the poles makes the system highly non-linear and requires very precise control [7].

Figure 1 shows the three steps that constitute the approach used in our experiments. First, the target environment is modeled with a supervised neural network to produce a simulation environment (step 1). Second, the controller is evolved in the simulation environment (step 2). Third, this controller is transferred to the target environment (step 3). In the usual application of this method, the target environment is the physical system, and nothing needs to be known about it as long as its interaction with a controller can be observed. However, in this study, we treat the analytical system (i.e. the differential equations [7] normally used in the literature as a reinforcement learning benchmark) as if it were a real two-pole system (i.e. the target environment), and the simulation environment is an approximation of it. This way it is possible to systematically study conditions for successful transfer in a controlled setting and gain insights that would be difficult to obtain from an actual transfer to a physical system.

Sections 2.1, 2.2, describe in detail steps 1 and 2 of the approach, respectively. Step 3 is presented in section 3.

2.1 Learning the simulation model

While it is possible to develop a tractable mathematical model for some systems, most real world environments are too complex or poorly understood to be captured analytically. A more general approach is to learn a *forward model* using observations of target environment behavior. The model, Φ, approximates the discrete-time state-transition mapping, Φ^*, of the target environment:

$$\widehat{s}_{t+1} = \Phi^*(\widehat{s}_t, a_t), \quad \forall \widehat{s} \in \widehat{S}, a \in A, \tag{1}$$

where $\widehat{S}$ is the set of possible states in the target environment, and $\widehat{s}_{t+1}$ is the state of the system some time in the future if action a is taken in state $\widehat{s}_t$. The function Φ^* is in general unknown and can only be sampled. Using Φ, interaction with the target environment can be simulated by iterating

$$s_{t+1} = \Phi(s_t, \pi(s_t)), \tag{2}$$

where π is the control policy and $s \in S$ are the states of the simulator.

In modeling the two-pole system we followed a standard neural network system identification approach [8]. Figure 1a illustrates the basic procedure for learning the model. The model is represented by a feed-forward neural network (MLP) that is trained on a set of state transition examples obtained from Φ^*. State-action pairs are presented to the model which outputs a prediction $\Phi(\widehat{s}, a)$ of the next state $\Phi^*(\widehat{s}, a)$. The error between the correct next state and the model's prediction is used to improve future predictions using backpropagation.

The MLP allows us to construct a model quickly, using few assumptions about Φ^*, and relatively few examples of correct behavior. Furthermore, when modeling a real system these examples can be obtained from an existing controller (i.e. the controller upon which we are trying to improve).

A training set of 500 example state transitions was generated by randomly sampling the state space, $\widehat{S} \subset \Re^6$, which consists of the cart position and velocity $(x, \dot{x})$, each pole's angle from vertical (θ_1, θ_2), and its angular velocity $(\dot{\theta}_1, \dot{\theta}_2)$, and the action space, $A \subset [-10, 10]$ Newtons, of the two-pole system. For each point $(\widehat{s}^i, a^i)$, $\widehat{s} \in \widehat{S}, a \in A$, the next state $\Phi^*(\widehat{s}^i, a^i)$ was generated 0.02 seconds (i.e. one time-step). The training set is then $\{\ ((\widehat{s}^i, a^i), \Phi^*_\delta(\widehat{s}^i, a^i))\ \}, i = 1..500$.

The accuracy of the model was measured in terms of the average squared error in the model's prediction across a separate test set of 500 examples.

Our model is global in the sense that a single function approximator is used to represent Φ for the entire state space, but it is local in terms of its temporal predictions. If the model is used to make predictions one time-step into the future, then the predictions will be very accurate. However, if the model is iterated (i.e. the output is fed back into the input) it will quickly diverge from the target environment.

In order to study the effect of model error on transfer, the weights of the model were saved at five points during training to obtain a set of models with different levels of error: $M_{0.002}, M_{0.005}, M_{0.008}, M_{0.010}, M_{0.025}$, where M_e is a model with error e. The models were tested extensively using trajectories from the target environment to verify that the model error (i.e. the error based on the test set), was a reliable indicator of the relative amount of prediction error that is actually encountered by controllers during evolution. That is, for two models M_x and M_y, $x > y$ implies that the prediction errors of M_x will generally be greater than those of M_y. It is important that this be true so that error measure e can be used to rank the models correctly for the experiments that follow.

2.2 Evolving with the Model

If the simulation environment perfectly replicates the dynamics of the target environment, then a model-evolved controller will behave identically in both settings, and successful transfer is guaranteed. Unfortunately, since such an ideal model is unattainable, the relevant questions are: (1) how do inaccuracies in the model affect transfer and (2) how can successful transfer be made more likely given an imperfect model? To answer these questions we evolved controllers at different levels of model error to study the relationship between model error and transfer, and to find ways in which transfer can be improved.

The controllers were evolved using the Enforced SubPopulations[1](ESP; [9]) neuroevolution method under three conditions:

No noise. The controllers interact with the model according to equation 2, and a controller is evaluated by having it control the double pole system starting in the center of a 4.8 meter track with the short pole (0.1m) at vertical and the long pole (1m) leaning at 4.5 degrees from vertical. Every 0.02 seconds the controller receives the state of the environment and outputs a control force, [-10,10]N, that is applied to the cart. The fitness used to evolve the controllers is the number of time steps before either pole exceeds ± 36 degrees or the cart goes off the edge of the track. The task is considered to be solved when a controller can balance both poles for 100,000 time-steps.

This experiment provides a baseline for measuring how well controllers transfer from simulation to the target environment. Each of the five models

[1] The particular neuroevolution algorithm used is not important in this study and is only mentioned for completeness.

$(M_{0.002}, M_{0.005}, M_{0.008}, M_{0.010}, M_{0.025})$ was used in 20 simulations for a total of 100 simulations.

Sensor noise. The controllers are evolved as above except that their inputs are perturbed by noise. The agent-environment interaction is defined by

$$s_{t+1} = \Phi(s_t, \pi(s_t + v)), \tag{3}$$

where $v \in \Re^6$ is a random vector with components $v(i)$ distributed uniformly over the interval $[-\alpha, \alpha]$. Sensor noise can be interpreted as perturbations in the physical quantities being measured, imperfect measurement of those quantities, or, more generally, a combination of both. This kind of noise has been shown to produce more general and robust evolved controllers in the mobile robot domain [3,10].

In figure 1b, the box labeled "sensor noise" represents this noise source. As in the "no noise" case, 20 simulations were run for each of the five models, this time with three sensor noise levels: 5%, 7%, and 10%, for a total of 300 simulations.

Trajectory noise. In this case, instead of distorting the controllers' sensory input the noise is applied to the dynamics of the simulation model. The agent-environment interaction is defined by

$$s_{t+1} = \Phi(s_t, \pi(s_t)) + w, \tag{4}$$

where $w \in \Re^6$ is a uniformly distributed random vector with $w(i) \in [-\beta, \beta]$. At each state transition the next state is generated by the model and is then perturbed by noise before it is fed back in for the next iteration. Although, a similar effect could be produced by adding noise to the actuators, $s_{t+1} = \Phi(s_t, \pi(s_t) + w)$, equation 4 ensures that the the trajectory of the model remains stochastic even in the absence of a control signal, $\pi(s) = 0$.

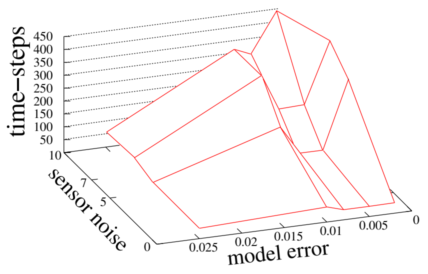
Eight different levels of trajectory noise $\{0.5\%, 0.6\%, \ldots, 1.2\%\}$ were used. As with the sensor noise simulations, 20 simulations were run for each of the five models at each trajectory noise level, for a total of 800 simulations.

3 Transfer Results

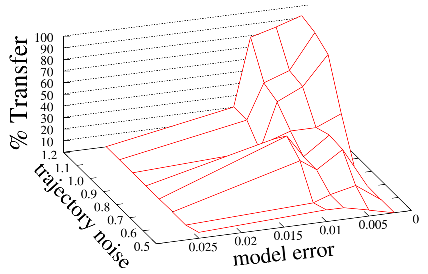
After evolving controllers with the model, the solution found by each simulation was submitted to a single trial in the target environment. Our criteria for successful transfer was whether a controller could perform the same task in the target environment that it performed in simulation. That is, balance the poles for 100,000 time steps using the same setup (i.e. initial state, pole lengths, etc.) used during evolution, but with no noise. Later, in section 4, we apply a more rigorous standard.

Sensor Noise

Figure 2a shows the transfer results for controllers evolved with sensor noise. The plot shows the average number of time-steps that the networks at each level of model error could control the target environment. Successful transfers were



(a) Sensor noise



(b) Trajectory noise

Fig. 2. Transfer results. Plot (a) shows the number of time steps the controllers evolved at different levels of model error and sensor noise could balance the poles after transfer to the target environment. Each curve corresponds to a different level of noise. Sensor noise improves performance slightly but does not produce successful transfers (a transfer was considered successful if the network could control the system for 100,000 time steps). Plot (b) shows the percentage of controllers that successfully transferred to the target environment for different levels of model error and trajectory noise. Lower error (i.e. more accurate local model) and higher trajectory noise produces controllers more likely to transfer.

very rare in this case (occurring only twice in all 400 simulations); in effect, the surface plot shows the performance of networks that did not transfer.

Without noise, all of the transferred controllers failed within 50 time steps i.e. almost immediately. As sensor noise was added, performance improved significantly, especially when model error was low, but controllers were still far from being able to stabilize the target environment. Therefore sensor noise, even at high levels, is not useful for transfer in this kind of domain.

Trajectory Noise

On the other hand, trajectory noise had a much more favorable effect on transfer. Because successes were frequent, a different plot is used. Figure 2b shows the percentage of networks that transferred successfully for different levels of model error and trajectory noise. Each curve corresponds to a different level of trajectory noise, with the "0" curve again indicating transfer without noise. The plot shows that trajectory noise compensates for model error and improves transfer. To achieve very reliable transfer, low model error needs to be combined with higher levels of trajectory noise. The best results were achieved with 1.2% trajectory noise and a model error of 0.002, yielding a successful transfer rate of 91%.

4 Evaluating controller robustness

During evolution a controller can only be exposed to a subset of the conditions that can exist in the real world. Therefore, how will it perform under conditions that differ from those encountered during evolution? In this section, we analyze the quality of transferred controllers in three respects: (1) generalization to novel starting states, (2) resistance to external disturbances, and (3) resistance to changes in the environment. Since only the controllers that were evolved with trajectory noise transferred successfully, this analysis pertains only to those controllers. Also, because different levels of trajectory noise had different transfer rates (figure 2b), additional controllers were evolved at each noise level until a minimum of 20 successfully transferred controllers were obtained for each level.

Generalization

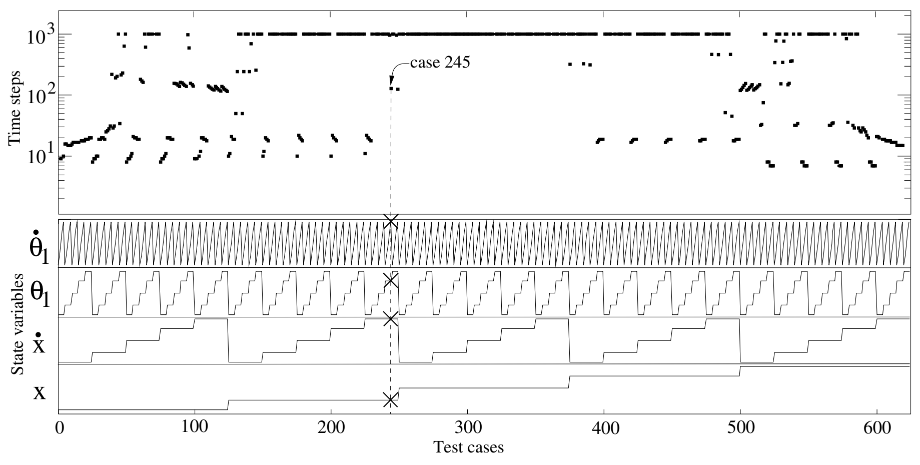
To measure how trajectory noise affects the performance of transferred controllers in across a broad range of initial states, 625 test cases were generated by allowing the state variables $x, \dot{x}, \theta_1$, and $\dot{\theta}_1$ to take on the values: 0.05, 0.25, 0.5, 0.75, 0.95, scaled to the appropriate range for each variable ($5^4 = 625$). These ranges were ± 2.16m for x, ± 1.35m/s for $\dot{x}$, ± 3.6 deg for θ_1, and ± 8.6 deg/s for $\dot{\theta}_1$. A controller is awarded a point for each of the 625 different initial states from which it is able to control the system for 1000 time steps. This test, first introduced in [11], has become a standard for evaluating the generality of solutions in pole balancing. A high score indicates that a solution has competence in a wide area of the state space.

Figure 3a is a visualization of a controller's performance on this test. Each dot in the upper half of the graph identifies the number of time steps the poles could be balanced for a particular start state, i.e. test case. Each test case is denoted by a unique setting of the four state variables $x, \dot{x}, \theta_1, \dot{\theta}_1$ in the lower half of the graph (θ_2 and $\dot{\theta}_2$ were always set to zero). Drawing a vertical line through the graph at a given dot gives the state variable values for that case. For example, the balance time for case 245 ($x = -1.080$, $\dot{x} = 1.215$, $\theta_1 = 0.031$, $\dot{\theta}_1 = 0.135$; the labeled point in the figure) is 129 time steps for this particular controller, which solved 353 of the 625 cases.

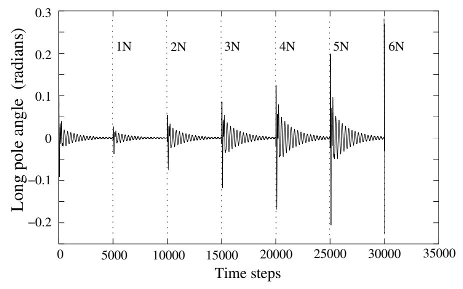
Figure 4a shows the quantitative results for the generalization test. Solutions evolved with more trajectory noise generalize to a larger number of novel initial states. Recall that the fitness function used here simply measures the number of time steps the poles stay balanced. It is therefore almost devoid of domain knowledge, and places no restriction (bias) on the control strategy. Still, the use of trajectory noise produces solutions that generalize to a large number of cases in the target environment that were not experienced in the simulation environment.

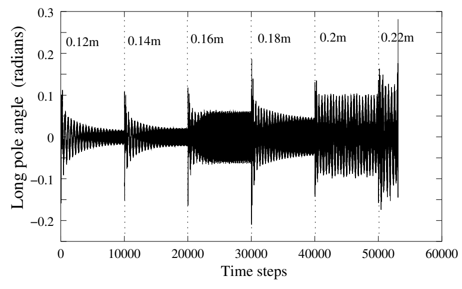
External Disturbances

The generalization test measures how well networks behave across a large region of the state space. Another important question is: how well will these solutions operate in "unprotected" environments where external disturbances, not modeled in simulation, are present? To answer this question, the networks were subjected to external forces that simulate the effect of wind gusts buffeting the cart-pole system. Each network was started in the center of the track with the long pole at 4.5 degrees (the small pole was vertical). After every 5,000 time steps, a force was applied to the cart for 2 time steps (0.04 sec). The magnitude of this pulse was started at 1 Newton and increased by 1 Newton on each pulse.



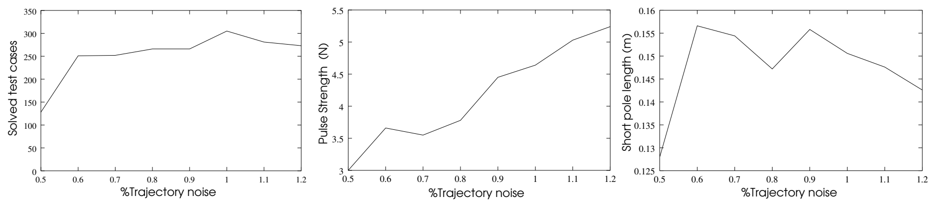
(a) Generalization





(b) External Disturbances (c) Changing Environments

Fig. 3. Examples of controller behavior on the robustness tests. The plots quantify the performance of a particular controller evolved with 1.0% trajectory noise on the three robustness tests. In plot (a) the lower half of the plot shows the values of $x, \dot{x}, \theta_1$, and $\dot{\theta}_1$ for each of the 625 cases. The upper half shows the number of time steps the poles were balanced starting from each case. This controller solved 353 of the cases. Plot (b) shows the trajectory of the long pole angle for a disturbance test. Each vertical dotted line marks the onset of an external pulse of the shown intensity (in Newtons). The controller is able to recover from disturbances of up to 5 Newtons, but fails when the force reaches 6 Newtons. Plot (c) shows the trajectory of the long pole angle for a changing environment test. Each vertical dotted line marks each lengthening of the short pole. The controller is able to balance the poles using increasingly wide oscillations as the short pole is lengthened. However, at 0.22 meters (i.e. almost double the original length) the system becomes unstable.



(a) Generalization (b) External Disturbances (c) Changing Environments

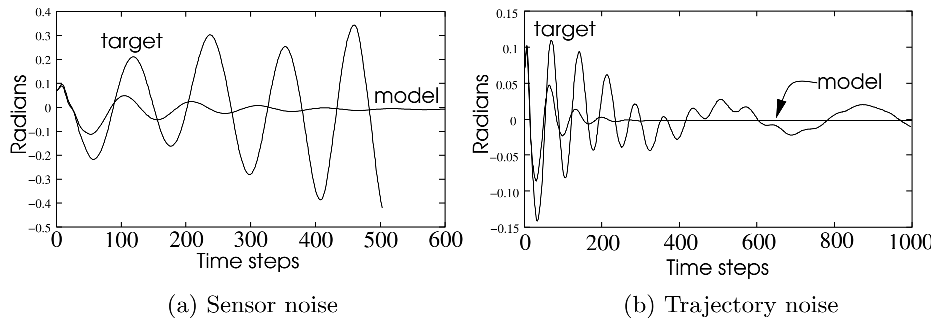
Fig. 4. Robustness results. The plots show the performance of successfully transferred controllers evolved at different levels of trajectory noise on the three different tests for robustness. Plot (a) shows the number of generalization test cases, on average, solved by the controllers. In plot (b), the y-axis is the average force in Newtons that a network could withstand. In plot (c), the y-axis is the average short pole length that could be balanced for 10,000 time steps. High levels of trajectory noise generally yield more robust behavior in transferred controllers.

Figure 3b shows the angle of the long pole for a typical disturbance test. In the first 5,000 time steps the controller stabilizes the system from its unstable initial state. After the first pulse hits, the system recovers rapidly. As the pulse is strengthened, the controller takes longer to recover, until the force becomes too large causing the system to enter into an unrecoverable state. Figure 4b shows the average maximum force that could be handled for each level of trajectory noise. Above 1.1% noise controllers could withstand an average pulse of over 5 Newtons. This magnitude of disturbance is very large: it is more than half of the maximum force that can be exerted by the controller. Higher levels of trajectory noise lead to networks that are less affected by external forces.

Changes to the Environment

An interesting and convenient property of the double pole system is that it is more difficult to control as the poles assume similar lengths. By lengthening the short pole during testing, we can measure how well the controllers can cope with changes to the dynamics of the environment such as those that occur in real systems through mechanical wear, damage, and modifications. For this test, each network was started controlling the system with a short pole length of 0.12 meters, 0.02 meters longer than the length used during evolution. If after 10,000 time steps the trial had not ended in failure, the cart-pole system was restarted from the same initial state but with the short pole elongated by another 0.02 meters. This cycle was repeated until the network failed to control the system for 10,000 time steps. A network's score was the short pole length that it could successfully control for 10,000 time steps. Figure 3c shows the behavior of the long pole during one of these tests.

Figure 4c shows the average short pole length versus trajectory noise. For low noise levels (≈0.5%), the networks adapt only to relatively small changes (0.03m, or 30%). At and above 0.6%, networks tolerate up to as much as a 57% increase (to 0.157m) on average. These are very large changes because not only do the dynamics change, but the system also becomes harder and harder to control with each extension of the short pole. Trajectory noise prepares controllers for conditions that can deviate significantly from those to which they were exposed during evolution, and, consequently, produces high-performance solutions that can better endure the rigors of operating in the real world.



(a) Sensor noise (b) Trajectory noise

Fig. 5. Comparison of controller behavior before and after transfer. The graphs show typical behavior (long pole angle) of a network evolved with sensor noise (a) versus a network evolved with trajectory noise (b), when controlling either the model or the target environment. With sensor noise (a), the trajectories coincide initially, but after about 50 time steps the target environment becomes unstable, although the model is quickly damped into equilibrium. With trajectory noise (b), the trajectories also do not coincide after about 50 time steps but the network is still able to control the target environment because it has been evolved to cope with a range of deviations from the model trajectory.

5 Analysis of transfer results

Whenever a behavior is learned in one environment some adaptation is necessary to preserve an acceptable level of performance in another, different environment. In a sense, using noise pre-adapts the agent to a range of possible deviations from the simulation model. It is not surprising that controllers evolved without noise did not transfer, even when model error was low. But why is there such a disparity between the sensor and trajectory noise results?

Let us take a typical solution evolved with sensor noise and use it to control the simulation environment *without* sensor noise. The resulting behavior is shown by the "model" curve in figure 5a. Sensor noise forces evolution to select against strategies (such as swinging the poles back and forth) that are too precarious when the state of the environment is uncertain. Consequently, high-performance solutions quickly stabilize the environment by dampening oscillations. This behavior, however, does not help networks control the target environment. Because the target environment reacts differently from the model, a policy that would stabilize the simulation environment can soon cause the target environment to diverge and become unstable (the "target" curve in figure 5a).

This result differs from the experience of many researchers (e.g. [3,10]) that have used sensor noise to make robots more robust and facilitate transfer. Unlike robot navigation tasks, where small inaccuracies in actuator values do not affect the transferred behavior qualitatively, the pole balancing domain requires very precise control and is much less forgiving because it is inherently unstable; a sequence of poor or even imprecise actions can quickly lead to failure.

The success of controllers evolved with trajectory noise can be explained by looking at the space of state trajectories that are possible by making noisy state transitions. If we let $\{u_i\}$ and $\{l_i\}$ be the sequence of states that form the upper and lower bound on the possible state trajectories for a given policy π, such that

$$u_{i+1} = \max_{s_i \in [l_i, u_i]} \|\Phi(s_i, \pi(s_i)) + \overline{w}\|, \tag{5}$$

$$l_{i+1} = \min_{s_i \in [l_i, u_i]} \|\Phi(s_i, \pi(s_i)) - \overline{w}\|, \tag{6}$$

where $\overline{w} = \text{argmax}_w \|w\|, \quad s_1 = l_1 = u_1$, then every state $\|l_i\| \leq \|s\| \leq \|u_i\|$ can be visited by some sequence of state transitions generated by equation 4. State sequences $\{u_i\}$ and $\{l_i\}$ form a trajectory envelope for a particular controller.

All of the trajectories that can be followed from an initial state s_1 will fall inside this envelope. Although each network evaluation involves only a single trajectory of finite length, the number of state transitions in a successful trial (100,000) is large enough that it represents a relatively dense sampling of the trajectory space. So the controller is effectively trained to respond to a range of possible transitions at each state, much like the principle underlying robust control theory where a controller is designed not for a single system but for a whole family of systems that is likely to contain the actual system of interest.

6 Discussion and Future Work

The transfer experiments revealed that a relatively accurate model alone may not be sufficient to ensure transfer when the target environment is inherently unstable. Transforming the deterministic model into a stochastic one by injecting trajectory noise, however, improved the transferred controllers significantly. They were able to cope with a wide variety of conditions that where not present during evolution. These experiments demonstrate how such transfer can be achieved in principle. For a large class of problems involving unstable systems this technique should prove useful.

Although these experiments have focused on direct transfer, in domains where unsuccessful controllers can be tested a simple post-transfer search can be used to find a successful controller. We found that many of the controllers that did not transfer in our trajectory noise experiments were "near-misses" that could be adapted to the target environment quite easily through local random search in the weight space. In domains where testing is not feasible, it may be possible to determine the stability of the controller through analytical tools such as those developed by Suykens [12] and Kretchmar [13] for robust neurocontrol. Only controllers that pass a stability test would then be allowed to interact with the environment, thereby reducing the chance of failure.

The pole balancing problem studied in this paper is a good surrogate for challenging problems in the real world. In ongoing work, we are applying neuroevolution to the active guidance of finless (i.e. unstable) sounding rockets, a problem with dynamics similar to that of the inverted pendulum. The techniques described in this paper will be used to transfer controllers to the actual rocket for launch. Similar applications should be possible in other domains, leading to more accurate and robust non-linear control in the real world.

7 Conclusion

The transfer experiments demonstrated that combining an empirical model with trajectory noise can make transfer possible even in unstable environments. Our results will hopefully provide a better understanding of the conditions necessary to make neuroevolved controllers sufficiently robust, and ultimately allow neuroevolution to be a viable controller design method for real-world systems.

References

1. Nolfi, S., Floreano, D.: Evolutionary Robotics. MIT Press, Cambridge (2000)
2. Ficici, S.G., Watson, R.A., Pollack, J.B.: Embodied evolution: A response to challenges in evolutionary robotics. In Wyatt, J.L., Demiris, J., eds.: Eighth European Workshop on Learning Robots. (1999) 14–22
3. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: The use of simulation in evolutionary robotics. In: Proceedings of the Third European Conference on Artificial Life, Springer-Verlag (1995)
4. Miglino, O., Lund, H.H., Nolfi, S.: Evolving mobile robots in simulated and real environments. Artificial Life **2** (1995) 417–434
5. Chavas, J., Corne, C., Horvai, P., Kodjabachian, J., Meyer, J.A.: Incremental evolution of neural controllers for robust obstacle-avoidance in khepera. In: EvoRobots. (1998) 227–247
6. Smith, T.M.C.: Blurred vision: Simulation-reality transfer of a visually guided robot. In: EvoRobots. (1998) 152–164
7. Wieland, A.P.: Evolving controls for unstable systems. In Touretzky, D.S., Elman, J.L., Sejnowski, T.J., Hinton, G.E., eds.: Connectionist Models: Proceedings of the 1990 Summer School. San Francisco, CA: Morgan Kaufmann (1990) 91–102
8. Barto, A.G.: Connectionist learning for control. In 3rd, W.T.M., Sutton, R.S., Werbos, P.J., eds.: Neural Networks for Control. MIT Press, Cambridge, MA (1990) 5–58
9. Gomez, F.J.: Robust Nonlinear Control through Neuroevolution. PhD thesis, Department of Computer Sciences, The University of Texas at Austin (2003)
10. Reynolds, C.W.: Evolution of obstacle avoidance behaviour: using noise to promote robust solutions. In Kenneth E. Kinnear, J., ed.: Advances in Genetic Programming. MIT Press (1994)
11. Dominic, S., Das, R., Whitley, D., Anderson, C.: Genetic reinforcement learning for neural networks. In: Proceedings of the International Joint Conference on Neural Networks (Seattle, WA), Piscataway, NJ: IEEE (1991) 71–76
12. Suykens, J., Moor, B.D., Vandewalle, J.: Stabilizing neural controllers: a case study for swinging up a double inverted pendulum. In: International Symposium on Nonlinear Theory and its Application (NOLTA'93). (1993) 411–414
13. Kretchmar, R.M.: A Synthesis of Reinforcement Learning and Robust Control Theory. PhD thesis, Department of Computer Science, Colorado State University, Fort Collins, Colorado (2000)