Evolving Bipedal Locomotion through Recurrent Connections

[An Experimental Comparison]

Ethan Eldridge
University of Vermont
33 Colchester Ave
Burlington, Vermont 05405
ejeldrid@uvm.edu

ABSTRACT

In the field of evolutionary robotics, bipedal locomotion is often considered a difficult problem. The nature of this difficulty lies in the inability for a simulated lifeform to balance itself properly. A standard artificial neural network lacks the neccesary complexity required for memory[5], however, through the use of recurrent connections in the neurons of the network, a small amount of memory can be sustained and balance can be more easily ascertained. To test this point, a simulated robot is described in section 2, the controllers for the robot are discussed in section 3, and the task environment and fitness function for the evolutionary algorithm are introduced in section 4. Results and conclusions showing surprising results appear in sections 5 and 7 respectfully,ideas for future works are explored in section 8.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics; I.2.6 [**Artificial Intelligence**]: Learning; I.6 [**Simulation and Modeling**]: Misc

General Terms

Robotics,Recurrent Network,Artificial Neural Networks, Locomotion

Keywords

Evolutionary Robotics, Hill Climber, Recurrent ANN, Bipedal Locomotion

1. INTRODUCTION

In nature, it is very common to find creatures who walk on more than two legs. The reasons for this are multiple, but one of the main points is that balancing on only two legs while moving is a challenging problem for simple beings and sometiems even for humans. In robotics, the ability to balance is usually accomplished in using more complex controllers for the brain of a robot. Typically, the nature of the task requires the ability to remember and be able to act on a previous state of being. One of the many ways to do this is through the use of recurrent artificial neural networks. These are networks that contain what are commonly referred to as self loops, or a connection from a neuron to itself. The difference between a typical self activation in a simple feed forward network and a recurrent node is simple. The recurrent node's self look comes from a previous timestep's activation of the same node. This behavior enables a very primitive form of memory to take place, and even with this slight amount of memory included, the complexity of tasks that a controller can perform increases substantially.

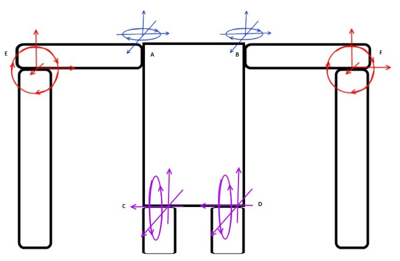


Figure 1: The Robot Body Plan and DOF

2. THE ROBOT

The robot design is based off of a gorilla, while not entirely a biped, the gorilla design is much easier to balance than a completely upright humanoid. This allows for a better comparison between the two networks discussed in the next section. The arms help create a large polygon of support for the robot to aid with it's balance and walking behaviors, also, these long arms enable the robot to traverse the environment far easier than if it was a completely upright biped.

The body plan shown in figure 1 shows the degrees of freedom (DOF) the robot has, each of the joints axis of rotation displayed by the small axes illustrated near that joint. The two shoulder joints A and B rotates about the Z-axis, joints E and F rotate about the Y axis, and C and D rotate around

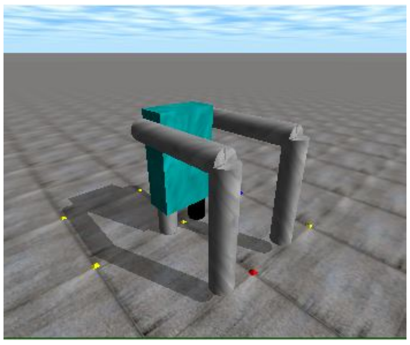


Figure 2: A Screenshot of the robot in an ODE Simulation

the X-axis. The coordinate system is the standard cartesian rule system, where if you were to hold out your right hand and make an L with your index and thumb, while lifting your middle finger orthagonal to the plane created by the index-thumb system, Z is straight up along the middle finger, Y is along the index, and the X axis extends in the direction of the thumb. Each joint rotates with a maximal angle of 60 degrees in either direction allowing for enough freedom to allow the robot to move in a proper fashion.

3. THE BRAIN

3.1

3.2 Memoryless and Feedforward

The memoryless neural network controller is a typical feed forward neural network. Each input neuron has a set of synaptic weights that connect it to an output neuron. The sum of these weights acting on an output neuron are multiplied by the value of the input neuron (its activation value), and then passed through a sigmoidal function to normalize it before it is used. This operation is reflected in equation (1)

$$a_i = \sigma(\sum_{1}^{n} w_{ij} a_j) \tag{1}$$

where a_i stands for the activation of any neuron i and w_{ij} denotes the synaptic weight from neuron i to neuron j, and in graphically in figure 3.

These types of neural networks allow for complicated behavior with a small number of nodes. However, if the same signal is given as input, the same output signal is always given, while this may seem obvious and trivial to state, this behavior limits the ability of the robot to process a given command that requires doing something based on a previous action. In case of balance, knowing your previous body state determines the next action in order to maintain an upright posture. Also, these types of state driven ideas allow for walking with coordinated leg movements and balancing with coordinated arm,leg, and body movements. the power of even a simple feedforward network is strong. Allowing for a controller that can successfully perform in a simple task environment given the correct training regiminent.

3.3 Recurrent

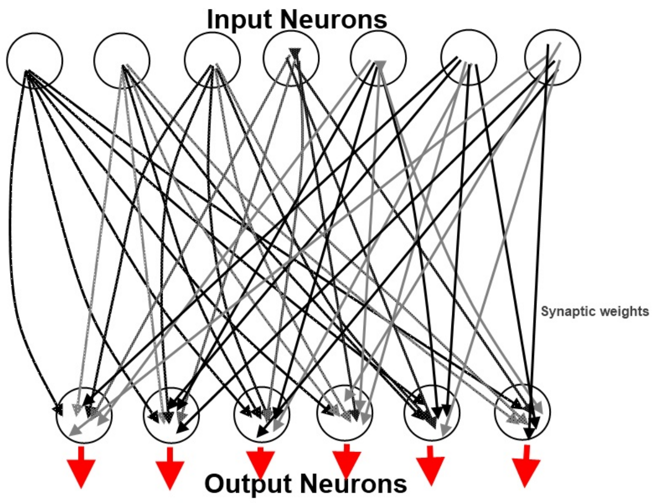


Figure 3: The Simple Feedforward Network

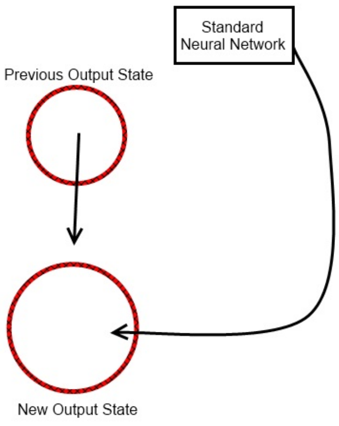


Figure 4: A Sample Recurrent Connection

In order to maintain balance, some sort of memory must be maintained and the previous state of a robots body configuration must be represented somehow. To introduce this quality into our simulation, an artificial neural network with recurrent connections is introduced. The network used is very simple, the output nodes are connected to themselves in such a way that the previous activation state of the output neuron has a direct affect on the new output. This is illustrated in figure 4, and by equation (2) where a_n denotes the activation of the output node at any timestep n.

$$a_n = a_n + a_{n-1} a_n \tag{2}$$

These self connections allow for versatile operations to be performed by the robot being controlled by the network, and our hypothesis is that such added beneficial connections will enable a controller to maintain it's balance and climb a set of short stairs with less difficulty than a controller without recurrent connections.

4. THE TASK ENVIRONMENT

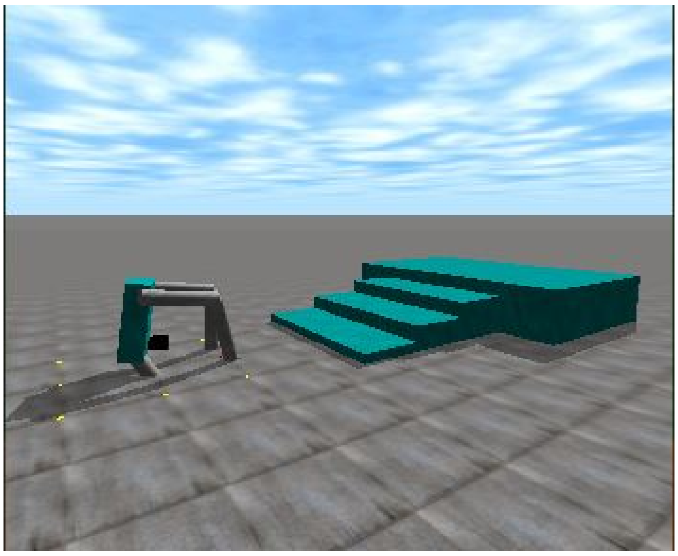


Figure 5: The robots task environment

To test the hypothesis presented in the previous section, we implement our robot design in the Open Dynamics Engine (ODE)[1]. To test the ability for our gorilla robot to balance, we create an environment that requires the robot to move not only forward, but upwards as well. The simplest way to do this is with a ramp, but in an attempt to showcase the recurrent networks benefits over the memoryless network, we use stairs. You can see this clearly in figure 5.

In order to traverse this environment the robot requires a suitable fitness function that will select for both forward displacement and climbing the steps. This can be done by rewarding for x and z displacement and penalizing for distance from the y axis. The penalty term must be included if the robot is to stay on the stairs while climbing. Mathematically, this is represented as

$$F = x_{dist} + z_{dist} - (0 - abs(y_{dist})) \tag{3}$$

by using these fitness functions and the task environment we can test our hypothesis.

During the experiment each controller was evolved for 1000 generations using a serial hillcimber evolutionary algorithm with a variable mutation rate. The mutation rate m started at 0.6 and after 10% of the total generations lowered to 0.4, and for each hundred generations decreases by 0.1 until after 500 generations when the mutation rate drops sharply to a more standard 0.05 rate of change. The reasoning behind this piecewise mutation rate is to aid the hillclimbers search diversity. If the mutation rate was fixed at the typical 5% value the entire time, we would probably have to evolve for far more than one thousand generations to see a significant change from the random neural network controllers at the start of evolution and what we hope to be good controllers at the end. by diversifying the mutation rate we allow our search to cover a larger portion of the possible search space and then zero in on a more optimal point. This is a powerful concept that can allow a hillclimber to behave in a slightly more intelligent way.

5. RESULTS

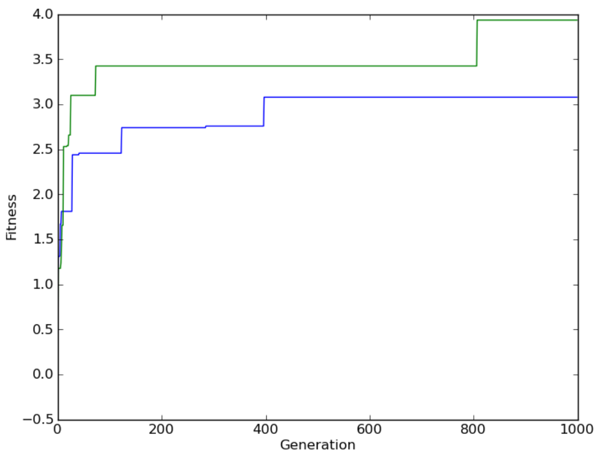


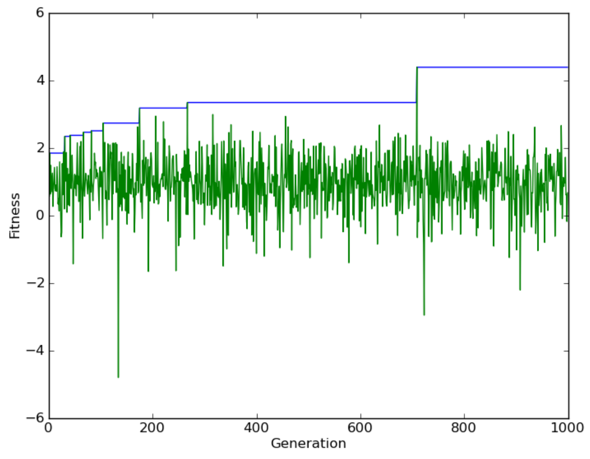
Figure 6: The Feed Forward Network outperforming the Recurrent Network after 1000 generations, Blue - Recurrent Network Fitness, Green - Simple Feedforward Network Fitness

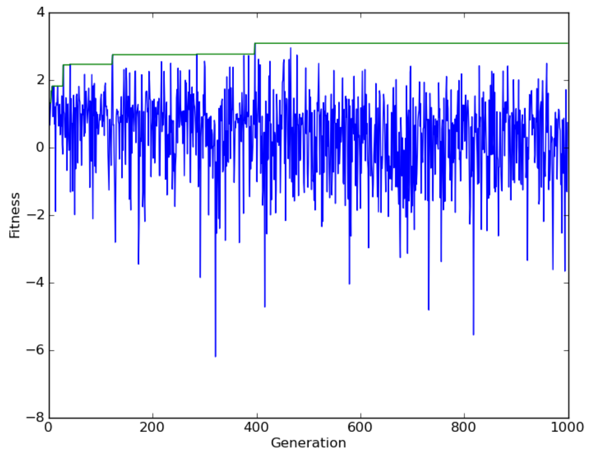
Despite the intuitive hypothesis that acting on a previous state would result in a better balance and overall fitness, through experimentation, this is not the case. The feedforward network not only outperformed the recurrent network, but also evolved faster as well. As shown in figure 6, the feedforward network outperforms the recurrent network from the start. This happened on almost all runs in the beginning, but even if the recurrent network started better, the simpler feedforward network overtook the other controller in all cases.

Investigating this curiousity, we found that the robot wasn't developing a gait, or trying to evolve balance, but instead performing a small amount of gymnastics. As shown in [2] the robot flips over itself and lands on one of the stairs. This isn't what we were evolving for, but no matter the number of times the simulations were evolved, this behavior developed in all feedforward controllers, so it seems that the fitness function is being exploited by the loopholes of the ODE simulation.

In the Recurrent Connection Controllers, we see a slightly less drastic strategy, most controllers have trouble coordinating the affect of the recurrent nodes and the current state and either stand still, or fall over this behavior is shown clearly in [3]. However, some controllers after significant evolution showed some promise. An example of this is displayed in [4], you can see from 0:01 - 0:16 there is a promising gait developing, but unfortunately, this was the end of an evolutionary run and no more work was done to evolve these weights as of yet.

As a comparison between a sample run of 1000 generations, figure 7 shows the parent and child fitness of the evolutionary run over time. As one can see from observing the Y-axis, the simpler network beats the recurrent network by a solid 1 in fitness. This is significant, especially when observing the behaviors of both controllers [2][4]. The diversity and trend of each child also is interesting to note.





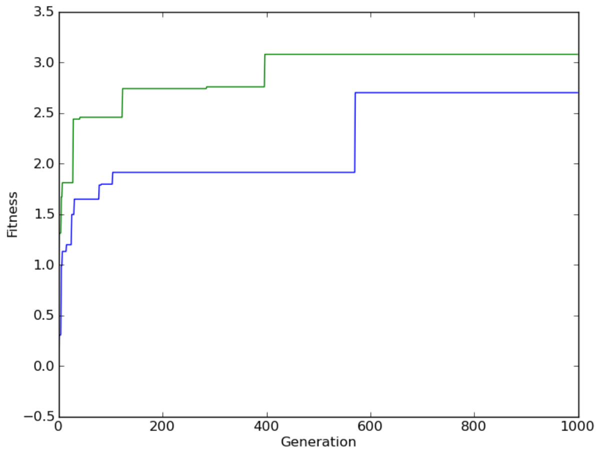


Figure 7: Top- Feedforward network, blue as best fitness, green as child fitness. Middle - Recurrent network, green as best fitness and blue as child fitness. Bottom - Another evolutionary run where the simpler controllers beat the recurrent one. Recurrent shown in blue and Feedforward as green.

Feedforward children, in general don't drop drastically in fitness, only three times in the top figure of figure 7. The recurrent network on the other hand shows a large amount of child fitness dropping off, even near the end of the total number of generations. This is curious because the mutation rate drops off drastically at this time; one would expect to see more controllers performing close to their parents fitness, but this is not the case.

Since controllers using recurrent connections seemed to require more time to evolve, a series of tests with a maximum generation count of 2000,3000, and 10000 was performed. As the trend showed, the feedforward network performed with it's usual flipping routine. the recurrent networks became better, but none of them managed to accomplish the tasks the feedforward networks were able to, and the significant increase to 10000 generations didn't aid the recurrent connections more than increasing to 2000 did.

6. DISCUSSION

6.1 Reasons for Counter Intuitivity

Given that each controller was only allowed 1000 generations in all initial experiments, it makes sense that the recurrent solutions do poorly. Given the added complexity of dealing with the recurrent networks, more evolution would likely benefit the controllers greatly. However, even given this fact, my hopes were for the piecewise mutation rate diversifying the search in such a way that the added complexity of the controller could be overcome. But this turned out not to be the case, even when two or three times the number of generations were allowed for continued evolution.

Despite not being able to do the task, the level of balance shown by the recurrently controled networks was far greater than that of the feedforward controllers. [4] clearly shows the beginnings of a gait being created, and whats more, it's very similar to what one might expect from the animal the robot is modeled after. If parameters of the physics engine were set differently, that controller might evolve to better model a gorilla.

6.2 Difficulties and Initial Configurations

The body plan presented in section 2 is the best of the multiple designs tried for the robot. The first robot had not only Z-axis rotation in the shoulder joints[1], but also a rotation about the X-axis[2]. However, due to difficulties with the ODE simulation this body plan was scrapped. The other difference that persisted through the first attempt and the second body plan[3] is the initial placement of the upper arms. While the final setup has them positioned in front of the body, the second and first setup had the upper arms straight out to the sides of the body, and much shorter legs. Longer legs were added to allow the robot to stand up when it fell, displace easier, and because the shorter legs had almost no affect on the forward locomotion and many controllers simply toppled over, or found a position with an axis of support and stood for it's lifespan.

Another difficulty which persisted from the beginning of implementation towards the ends of the final experiments was the unreliability of using file IO between programs. In

[1]joints A and B Figure 1
[2]the same rotation as the legs
[3]the third body plan is the one presented in this paper.

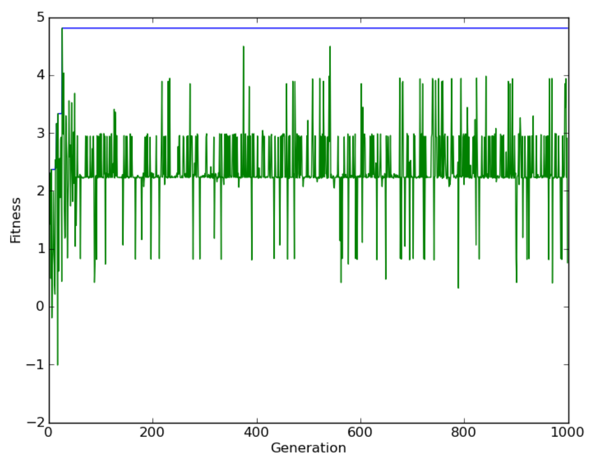


Figure 8: The Blue line is best fitness and the green is the child fitness. Notice how the perturbation stems from a single horizontal line, this is caused by the error in implementation.

the initial implementation of the experiment, python code would create the neural network, and then output these weights to a file that would then be read in by the ODE Simulation running in C++. Unfortunately, timing these two programs without any threads, or knowledge thereof resulted in numerous moments when 984 generations would pass after waiting an hour, and then a runtime error in python would result in all data being lost. After a multitude of such irritations, all code was converted into a single C++ program except for those responsible for creating plots and images. The increase in both speed and reliability resulted in the ability to perform experiments at a significantly higher rate; this increase in productivity allowed for multiple runs with a higher generation count of 2000 for both types of controllers.

At first, a third and fourth neural network were also included in testing data, but to boil the experiment down to a more fundamental level, networks using hidden nodes, and networks using both hidden nodes and recurrent connections were removed from the work presented here. Also, multiple parameters of the ODE simulation had to be more finely tuned, for example, the arms of the gorilla, at first, were far too heavy for the body and forces to support, and controllers had little room but to simply fall over, once the density of the arms was lowered, the controllers were able to find much more suitable controllers for the task environment. An error that persisted and caused much of my data to become invalid was the implementation of the recurrent neurons. This is done by storing the output of the neuron in another array. These values are then used to modify the next outputs, however, once a simulation is done for a controller, these weights *must* be flushed and reset back to their iniatialized bias value. This bias value is an arbituary value that is constant for all controllers, this error was discovered by a combination of observing figure 8 and that the exact same weights passed to the simulation resulted in radically different behavior. In figure 8, green lines indicate the fitness of the children who are perturbed from the parent whose fitness is represented by the blue line. The green values in the figure show a trend of always stemming from a single horizontal line. This line is the visual result of only refreshing a single neuron, thus created five neurons whose values persisted throughout controllers. This persitence was disasterous to the project and set back days of work due to invalid data.

7. CONCLUSIONS

The hypothesis, that a recurrently connected network could outperform a simple feedforward network, as shown by the data presented in previous sections, was disproven by the experiments. Numerous reasons exist for why this happened, but the most likely include: faulty implementation, an unsuitable number of generations for each controller to evolve, or the sheer size of the search space is simply too large for even the piecewise mutation rate method to scour. The most likely of these is length of generations for the controllers to evolve. The sophistication of the behavior of these controllers is greater than that of the feedforward networks. In future works, these more complex behaviors could help create more robust controllers to a changing environment.

8. FUTURE WORK

In future work many of the concepts implemented in this paper can be improved on, mutation rate choice, the evolutionary algorithm, the networks compared, and the use of scaffolding. Also, the task environment is completely static in all the experimental runs, but a scaffolding technique was originally intended to be used. A difficulty variable in the code modifys where the stairs are relative to the origin, the least difficult setting is the one used in all experiments in this paper, however if more work was put into this project in the future the setting could be modified as a controller completed each difficulty level. In this scaffolding fashion, a more robust controller could be evolved. Another possibility could be to include more vertical stairs or modify the steepness of the stairs in another difficulty sense. This idea could perhaps extend to evolving controllers with the ability to climb objects.

Plans for the future include implementing a fitness proportionate mutation rate instead of a piecewise mutation rate, this could be done as follows. The best fitness would be calculated as the X position of the last stair and the Z position of an upright robot standing on that stair. This fitness would be the numerator in the mutation rate equation, with the actual fitness of any controller as the denominator. This modifier would act on the maximal mutation rate to shrink or increase the mutation rate as a function of the fitness of a controller, allowing for a narrower search space as the controllers became better. This have an affect similar to the piecewise mutation function, but suboptimal solutions could be hard to break out of as a result of the lack of rediversification caused by such a function.

Only simple feedforward networks and a very basic recurrent network were compared in this paper, but code for hidden nodes and combinations of simple,hidden, and reccurent nodes is in place. This paper seeks to showcase the differences between the recurrent connection design shown in this paper versus the original feedforward network, and clearly shows the levels of sophistication from the recurrent networks versus the fast evolutionary pace of the feedforward networks. In later works, hidden nodes that would act in a similar fashion to the human ear drum may be implemented and other sensors taken from human anatomy.

There is no doubt that the body shapes the way the brain percieves stimuli, just as there is no doubt that the brain shapes the way the body itself moves. This complex dueling system is something that nature has been evolving for years, that our simulations create controllers that mimic nature without being aware of it[4], is impressive in itself and show the power of using any ANN as a controller for a robot.

9. ACKNOWLEDGMENTS

The author would like to thank Professor Dr. Joshua Bongard for a fantastic and interesting class. Jack Christoforo and Garth Fritz for staying awake with me in the Computer Science Laboratory running tests until dawn to try to get as much data as possible.

10. REFERENCES

[1] Russell Smith *Open Dynamics Engine* www.ode.org 2001

[2] Ethan Eldridge *Video Capture - Typical Feedforward* http://www.youtube.com/watch?v=72YZ5vmEKEc 2011

[3] Ethan Eldridge *Video Capture - Typical Recurrent Network Behavior* http://www.youtube.com/watch?v=8ISqRisSSGY

[4] Ethan Eldridge *Video Capture - A promising evolved recurrent networks behavior http://www.youtube.com/watch?v=6u3pRC_1OjM*

[5] Joshua Bongard *Lecture 5 - Neural Networks* http://www.cs.uvm.edu/j̃bongard/2011_EvolRob/CS295_395_Lecture5_NeuralNetworks.pdf

[6] Eric D. Vaughan and Ezequiel Di Paolo and Inman R. Harvey *The Evolution of Control and Adaptation in a 3D Powered Passive Dynamic* In Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems, ALIFE'9 2004 139–145 MIT Press Walker

[4]The terminology is rather loose here, computers are not aware of anything, which makes this statement even more mind boggling