Comparing Inputs to Evolutionary Algorithms for Maze Pathfinding

Annie Zhao and Owen Kephart[1]

[1]Computer Science, Swarthmore College, Swarthmore, PA 19081

December 18, 2017

Abstract

This paper explores trade offs between varying compression schemes on visual sensory inputs and neuroevolution of augmenting topologies (NEAT) fitness functions for efficiently navigating through a visually distinct maze. We examined 128x128x4 image inputs scaled down to 4x4x4 inputs using an autoencoder network and rescaling. While the autoencoder performed well, we do not recommend that it be used as input into a NEAT network as it is difficult to reclaim the important information out of the encoded representation without the decoder. We suspect that simplifying the sensor inputs manually is generally superior to neural encodings. We also experiment with objective-based, novelty-based and hybridized fitness functions for NEAT but did not obtain significant results. We could not create a map that hit the sweet spot of difficulty to easily observe a difference between our experimental methods.

1 Introduction

Navigating mazes is a classically difficult autonomous learning task. Mazes are often used to test for intelligence and recall in animal species as the process of efficiently pathing through a maze requires both exploration and, once the goal has been found, repeated exploitation. For robotic agents, recent attempts to solve this problem have found success across a variety of machine learning domains. Initial efforts utilized straightforward algorithms, such as a Modified Wall-Following Navigation algorithm [1]. Researchers found that while genetic algorithms generally outperformed their benchmarks of depth-first search and breadth-first search in finding the shortest path to the goal [2]. The difference in performance between the genetic algorithms and iterative approaches became increasingly significant as the sizes of the mazes increased. Furthermore, successful maze navigation behavior has been achieved outside of simulation. In a study conducted by researches at the University of Freisburg, a successor-feature-based deep reinforcement learning algorithm with an ability "to transfer knowledge from previously mastered navigation tasks to new problem instances" managed not only to speed up existing deep learning techniques but also solve simple, visually similar mazes in a live environment [3].

The primary inspiration, however, for our project is the work on maze navigation done by Google's Deepmind Lab team. Earlier this year, the team designed and performed well on the task using "goal-driven reinforcement learning ... with auxiliary depth prediction and loop closure classification tasks" [4]. This project did well in two domains: 1) developed a robust method of processing visual input and 2) created a multi-layered reinforcement learning system for an agent to recall past runs and then act optimally. We use the same 3D simulation engine to create our own visually distinct mazes. In the first domain, we examine

different ways of encoding sensor input through rescaling or dimensionality reduction via an autoencoder network. In the second domain, we test the performance of evolutionary algorithms using neuroevolution of augmenting topologies (NEAT) with either pure objective based fitness, novelty based fitness or a combination of both fitness functions.

2 Experimental Setup

2.1 Environment

We used a simulation environment built on top of the Quake 3 engine, Deepmind Lab. We built five different maze levels for this environment using levels generated from hand-crafted text files. By defining an entity layer and a variation layer, Fig 1, we can design the architecture of the level, spawn position, goal position and the colors of the walls within the maze. The goal, shown in Fig 2, is a fruit-like object surrounded by red walls that gives a reward and ends the agent's run when it has been reached. For our experimental runs, each maze had a set spawn point and set goal location, with only a single goal. Each maze was designed such that they would not be visually uniform to prevent two different locations from looking too similar. The text maps were compiled into a .pk3 format for use by the engine.

The agent has 7 possible actions within the environment, each specified by a value in an action vector: {look left/right, look up/down, move left/right, move forwards/backwards, fire their weapon, jump or crouch}. At each step, or frame, the agent behaves based off of the values in given in the action vector. For our experiments, we restricted the agent's movement set to be left/right and forwards/backwards by modifying only two action vector values and leaving the rest to be zero.

Figure 1: Maze 4 entity and variation layers.

2.2 Autoencoder

In order to collect training data for our autoencoder, we utilized a special collector agent. Each agent is be intialized with a random turn value between 10 and 30 degrees and rotate in place. A 128x128x4 image was saved as an RGBA PNG every 20 steps. As a result, the depth field of a given frame from our environment is visually represented by the transparency values in the alpha channel. As the learning environment has a special spawn animation, we avoided taking images from the first 50 steps of a run. We repeatedly ran this agent within the mazes using variable spawn location and each run had a default length of 200 steps.

We created several variations of an autoencoder that reduced the size of our RGBD input images. The final version was a 22-layer autoencoder that narrowed down to a bottleneck of size 4x4x4 before rexpanding back out to 128x128x4 over 2000 epochs. The encoder half of the network was composed of 6 convolutional and 5 max pooling layers, each with a 2x2 filter. The decoder half of the network reversed the convolutional layers of the encoding process and

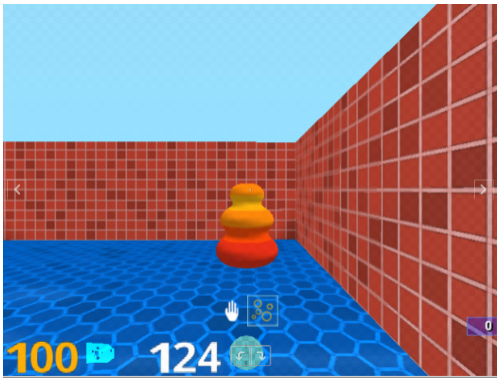


Figure 2: Goal within maze, surrounded by red walls.

replaced the max pooling layers with 2x2 filter upsampling layers. As this was a fairly large network for an autoencoder, the training time for this network took approximately 14 hours on a training set of 3000 collected RGBD images. Some simple keras graph surgery allowed us to remove the decoder half of the network and extract the layers of the autoencoder only up until the bottleneck for use in our experimental agents.

2.3 Pathfinding

We created three varying forms of visual sensor input for our agent, all of size 4x4x4. The autoencoded image, as mentioned in the previous section, a scaled down image, using bilinear interpolation and an image composed of random data for our baseline comparison.

We ran trials on three different evolutionary algorithms, objective-based NEAT, novelty-based NEAT, and a hybrid algorithm. Fitness-based NEAT minimizes distance from goal and focused on objective but has a weakness of getting stuck in local minima. We scaled the distance to the goal to be from between 0 to 1 for the objective fitness function. Then, if the goal is reached, the total score for that run would be $10 - (timetoreach)$. Meanwhile, novelty-based NEAT maximizes the uniqueness of the path but is unfocused and had no guarantee of moving towards the goal. Finally, we hybridized the two fitness functions using a linear combination using a tunable parameter α in

$$\alpha \cdot noveltyfitness + (1 - \alpha) \cdot objectivefitness$$

For our experiments, we set $\alpha = 0.5$ for a ratio of 50% objective and 50% novelty fitness. For each of these three evolutionary algorithms, we tried the three visual inputs for a total of nine experimental runs per map. We conducted 10 evolutionary runs per experiment for 30 generations each with a population size of 20 individuals. It took approximately 30 mins per run for total computation time of around 5 hours per experiment.

We found that maze 4, as shown in Fig 1, was consistently solvable for each of the 9 experiments, including even the random input agent. As a result, we decided to run the same set of experiments on maze 2, shown in Fig 3, with a more difficult path to the goal.

2.4 Hardware

We used several machines to train our autoencoder and run our maze experiments. Each had 32GB of RAM, Intel(R) Xeon(R) CPU and a Quadro M1000M GPU. The autoencoder ran on the GPUs while the maze computation was on the processors.

Figure 3: Maze 2 entity and variation layers, with a more difficult path towards goal.

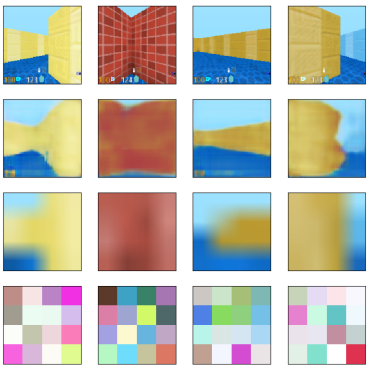


Figure 4: Visual comparison between original image (first row), image passed through autoencoder (second row), image scaled down and back up using bilinear interpolation (third row), and a random image (last row).

3 Results

Overall, our results produced by our autoencoder were very encouraging, and showed our output was significantly more similar to the input than a simple downscaling/upscaling procedure. For our pathfinding, we did not see a significant difference in performance based on the algorithm used, and perfomance was similar between networks with scaled image sensor inputs and autoencoded image sensor inputs, and lower for random sensor inputs.

3.1 Autoencoder

Our initial autoencoder compressed the data in the input image down to an 8x8x4 image. The model for this autoencoder converged after approximately 2000 epochs, and achieved a validation loss of .4038 for our metric, binary cross entropy. However, this bottleneck proved to be too large to work with efficiently in NEAT framework, and so we switched to an even smaller 4x4x4 bottleneck. This model also converged after around 2000 epochs, achieving a validation loss of .4113 (lower is better). While the difference in loss is relatively small, there were some minor subjective quality differences.

However, these subjective differences are small compared to the differences between the different methods, shown in Fig 4. Note how the images passed through the autoencoder preserve a large amount of data about angles and edges that an image that is simply scaled down cannot.

Comparison Image	Mean	Std
Original	0.394	.034
Autoencoded	0.411	.034
Scaled	0.436	.033
Random	0.831	.065

Table 1: Comparison of image loss values for different compression schemes (N=1000)

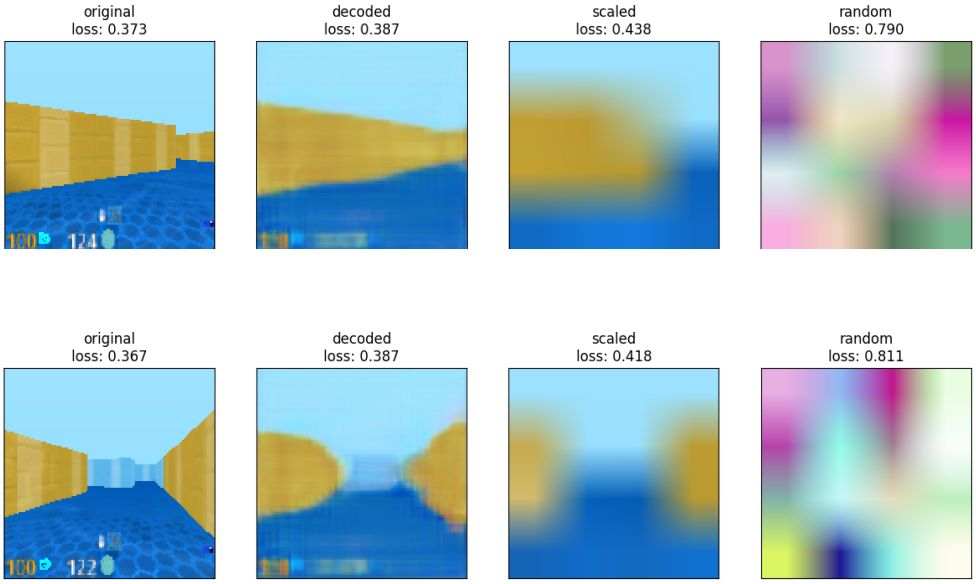


Figure 5: Loss metrics for different compression methods with two different input images.

In terms of quantitative differences, we randomly selected an input image from the test set 1000 times, and found the binary cross entropy of this entropy compared with four other images – the same exact image, the image after being passed through the autoencoder, a rescaled image, and a random image (scaled up from a 4x4x4 original image). From this, we calculated the mean and standard deviation of each off these distributions (Table 1). We found with high confidence ($p < .01$) that each of these distributions was statistically different from the others, indicating that using the autoencoder is a better method for minimizing this metric than scaling the image down and then up, which is in turn better than simply generating a random image.

Binary cross entropy is generally a hard metric to visualize, but in Fig 5, you can see how smaller loss values correspond to images that look more similar to the original. Note that even finding the cross entropy of an image with itself is not guaranteed to return zero.

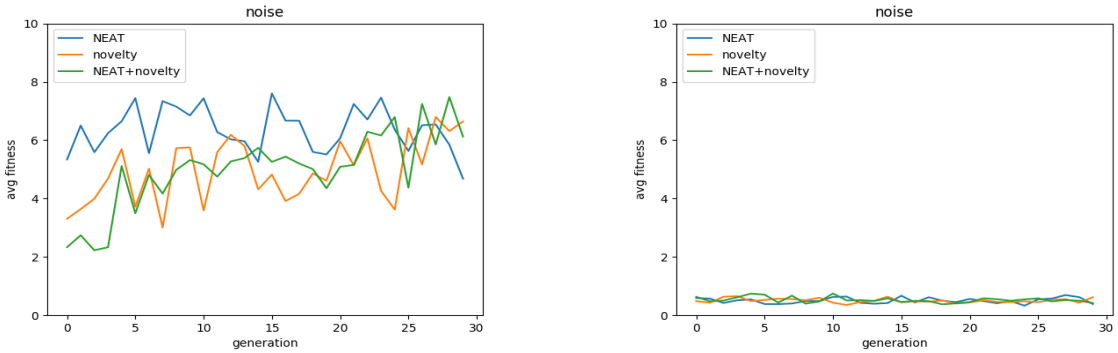
3.2 Pathfinding

For our metric of average of maximum fitness at each generation, no statistically significant difference was observed between scaled input and autoencoded inputs across all algorithms.

In addition, the algorithm itself did not seem to matter. On the hard map, we were not able to generate an agent that could solve the maze within the 30 generations, regardless of our parameters.

While on the easy map, the agent with random sensor inputs was able to fairly consistently find a solution, by virtue of it having inconsistent inputs, it was not able to iterate upon this and retain this good solution for future generations. Thus its average maximum fitness does not grow over time as it does for the other two input types, which are deterministic.

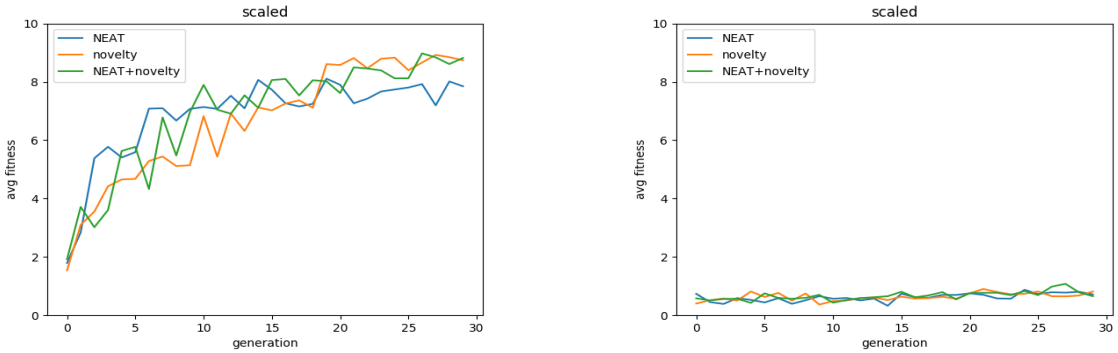
On the hard map, we see a very minor upward trend for the non-random sensor inputs, which aligns with our expectations. These results are plotted in Fig 6, Fig 7, and Fig 8.



(a) Easy map

(b) Hard map

Figure 6: Average over 10 runs of the maximum fitness for each generation for algorithms with random input sensors



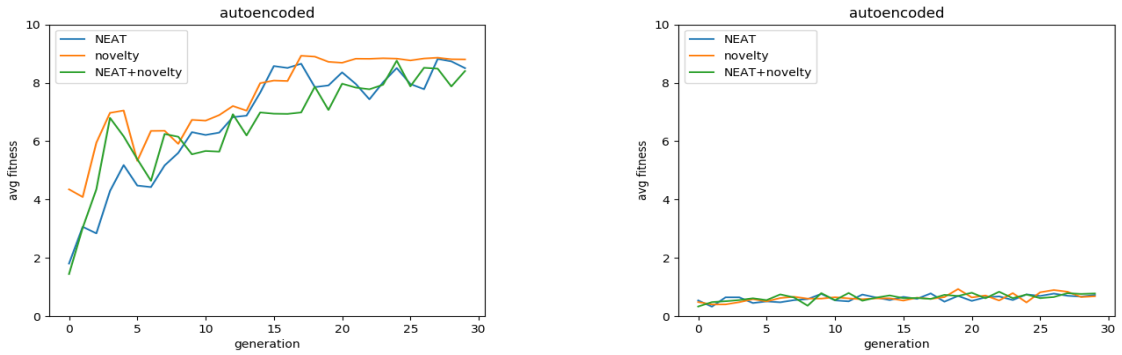
(a) Easy map

(b) Hard map

Figure 7: Average over 10 runs of the maximum fitness for each generation for algorithms with scaled input sensors

4 Discussion

In general, we did not find statistically significant differences between most of the studied parameters. None of the three genetic algorithms that we experimented with were shown to behave significantly differently from each other on either of the maps that we experimented



(a) Easy map

(b) Hard map

Figure 8: Average over 10 runs of the maximum fitness for each generation for algorithms with autoencoded input sensors

with (the easy and hard mazes). For the hard maze, it was fairly obvious that the algorithms were simply incapable of solving the maze in the relatively low number of generations that we gave it. No run ever made it all the way to the end of the maze. It would be interesting to see what would happen if we let the algorithms run for many more generations, but the tests took an extremely long time to run as it was, and there was no guarantee that the robot would ever find the goal.

For the easy mazes, we believe that most of the successful solves were simply a result of random variation in genomes happening to get the robot to the end of the maze, rather than a controlled series of explorations over time, or iterative improvements on past genomes that got close to the goal. Having 64 sensors as input to the learning algorithm likely overwhelmed it, and in such a small number of generations it was unlikely for it to discover a way to make use of all of the information it was being flooded with.

We were originally surprised at the similarity of the results of the scaled image inputs and the autoencoded image inputs in the case of the easy map. However, on further reflection, this is to be expected. The NEAT networks that we were generating were likely not picking up on any real qualities of the input data. However, it was guaranteed that for identical input images, each of these methods would output the same sensor readings. Thus all the learning algorithm had to do was stumble upon some combination of parameters that managed to get it to the goal, and from there it could just copy this solution and have it spread throughout the population (as this solution would be guaranteed to have a much higher fitness than any other genomes).

This thinking is reinforced by the observation that the network that worked with noise as an input consistently found

5 Conclusion

While the autoencoder that we trained was definitely interesting, and had a fairly high performance, we do not recommend that it be used as input into a NEAT network. It is incredibly difficult to reclaim the important information out of the encoded representation without the weights that are stored in the decoding half of the network. While we were not able to create a map that hit the sweet spot of difficulty such that we could observe a difference between the non-random methods, intuition gained from this process suggests that you would be better off simplifying the sensor inputs manually.

In general, the simple NEAT implementation seems somewhat ill-equipped for a complex task in a 3D environment if you are not able to dedicate large computing resources to its training.

6 Acknowledgements

We'd like to thank Professor Lisa Meeden for all of her help, both in lecture and in lab. We went through many shifts in goals with this project and she was invaluable in helping us narrow in on what we found to be most interesting.

Jeff Knerr was extremely helpful in the experimentation phase of our project, and aided us in installing the necessary software across all of the Computer Science department lab machines.

References

[1] Fawaz Y. Annaz. *A Mobile Robot Solving a Virtual Maze Environment* 2012. University of Nottingham (Malaysia Campus), Department of Electrical Electronic Engineering, 43500 Semenyih, Malaysia. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.885.9214rep=rep1type=pdf.

[2] Anton Jonasson and Simon Westerlind. *Genetic algorithms in mazes.* 2016. Examensarbete Teknik, Inom Grundniv, 15 Hp, Stockholm, Sverige. http://www.diva-portal.se/smash/get/diva2:927325/FULLTEXT01.pdf.

[3] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker and Wolfram Burgard *Deep Reinforcement Learning with Successor Features for Navigation across Similar Environments.* 2017. University of Freiburg, Institute of Computer Science, 79110 Freiburg, Germany. https://arxiv.org/pdf/1612.05533.

[4] Piotr Mirowski, Razvan Pascanu et Al. *Learning To Navigate In Complex Environments.* 2017. DeepMind London, UK. https://arxiv.org/abs/1611.03673.