NERL: Neural-Network Emulation of Reinforcement Learners

Will Schneider, Chris Magnano and Kevin Roberts

Abstract

Neural networks are one of the most useful machine learning techniques, but typically require substantial data sets of input/output pairs. In this paper we define NERL, a method of training NNs on Reinforcement learner agents. We trained an approximate RL agent on the Space Invaders game in the Arcade Learning Environment (ALE) and on simulated robots in Pyrobot. We then trained NNs on the input given to the RL agent and the action chosen by the RL policy. In ALE, we compared three different methods of feature generation, ranging from a raw visual field to hard coded game-specific features. We found that in ALE, NNs were very good at mimicking the RL agent, and in one instance were able to outperform the RL agent in terms of score. The light foraging task proved too chaotic for NERL as the dual training compounded the noise in the environment. NERL shows promise as a technique to easily generate multiple machine learning agents without requiring a large amount of labeled training data.

1 Introduction

In recent years, there has been an explosion of learning methods for robotic agents, each with their own strengths and weaknesses. Some methods, such as reinforcement learning (RL), have the ability to learn while interacting with the environment. RL uses delayed feedback but can have trouble generalizing to new situations and performing complex, multi-step actions which require a memory of previous behavior [1]. Other methods, such as artificial neural networks (NNs), are able to generalize and perform more complex actions but require constant feedback and large amounts of training data.

Another important factor in choosing a learning model is the level of domain engineering necessary for that task. Model-free RL systems [1] simply require an overall reward function to maximize. NN training methods such as back-propagation [3], however, are supervised and thus require labeled data. For tasks such as classification, labeled data can be easily found. For agent-based tasks which rely on a diverse set of behaviors, labeled data can be extremely time-consuming and difficult to generate. Often, a domain expert is needed to manually generate this data, which can be a stumbling block for experiments involving neural networks.

Methods such as evolutionary algorithms seek to circumvent the necessity of labeled data by changing networks at random and then testing them against a fitness function [10]. However, these training methods struggle with deceptive problems [6] and can require large amounts of time and memory to run [10].

A method which could translate information gained from an RL agent to an NN agent could solve the problems of domain engineering and generalization, combining the strengths of each method. This task of transferring knowledge has additional applications in adversarial machine [**?**] learning and legacy systems. In the task of adversarial machine learning, an agent may be attempting to determine the behavior of a spam filter or other security device, and thus needs to use the actions of that security device as labeled training data. Often a slow, dense legacy system is swapped out for a faster and more user-friendly trained system. However, it is desirable for the new system to emulate the behavior of the old system.

While RLs have the ability to quickly learn sub-problems and require little domain engineering, NNs may posses a greater ability to apply knowledge in new or noisier situations. We propose using an RL agent to generate labeled training data for an NN. Our training pipeline involves training an RL agent on a task, outputting the RL agent's actions as labeled data, and training a neural network on that labeled data. Thus, the NN will be learning to emulate the RL agent. We call this method Neural network Emulation of Reinforcement Learners, or NERL. We hypothesize that NERL will outperform RL agents, and that NERL will require a lesser degree of domain engineering with regards to feature generation than an RL agent.

The rest of the paper is laid out as follows: We present background information and existing research on RLs and NNs, as well as past work on hybrid approaches. We then outline the parameters of our experiment and document and discuss our results. We complete by exploring possible future directions for our research.

2 Background

2.1 Reinforcement Learning

Reinforcement learning (RL) is a machine learning technique where the agent learns an action to perform for every state in the environment. To determine whether or not a state is good to be in, the agent is given a reward for that state. Rewards can be either positive or negative. Most states typically have very little reward, so the agent will move around the environment randomly. However, once it reaches a positive or negative reward state, it will learn that the state which lead to that state is also associated with that reward.

A common method of RL is Q-Learning where each state and its current reward is stored in a table. However, in an environment with lots of states, such as one where both the RL agent and other agents are moving, the agent is unlikely to visit a certain state enough times in order to assign it a specific reward. For more complex environments we use RL methods which instead approximate their current state by extracting certain features from it. Q learning is another method of RL learning which creates a table where each arrangement of features is then given its own reward. However, calculating this table can quickly become intractable in large environments.

Approximate Q-learning, the RL method used in the light foraging experiment (see section 4.2) solves this problem through creating weights for each input feature. The value of a certain state-action pair is then evaluated by the sum of the current features, each multiplied by its weight. SARSA learning [1], the RL method used in the arcade experiments (see section 4.1), is extremely similar to approximate Q-learning but uses an additional state-action pair in its evaluation function, allowing it to have increased memory of previous actions.

2.2 Neural Networks

Artificial neural networks (NN) are a well-known and powerful supervised machine learning technique which have been employed in a variety of situations and tasks. An NN is made of an input layer, which is given the sensor data, a number of hidden layers, and an output layer. Features are fed into the input nodes. The nodes then propagate information through the hidden layers to the output layer. Given enough sets of labeled data the NN will then be able to generalize to new inputs and generate a correct output by learning different weights to give connections to different nodes. We trained NNs using back propagation (BP) [3], which uses gradient decent to propagate error back from the output nodes to the input nodes, changing connection weights. Over multiple epochs, or runs through the training data, the correct connection weights are learned to properly map inputs to outputs. While powerful and theoretically able to approximate any function when at least one hidden layer is present, NNs require large amounts of labeled training data.

3 Previous Work

3.1 Arcade Learning Environment

For a number of our experiments we used the Arcade Learning Environment (ALE), a framework built on top of an Atari 2600 emulator which allows for AI agents to play a wide variety of Atari games. Bellamare et al [2] trained a Reinforcement Learner for 10000 iterations on 50 different games. They used a number of different feature encodings, most of which involved representing the input to the learning agent as the pixels on the screen. The actions the agent could take corresponded to the outputs on the Atari controller: moving the joystick in one of 8 directions or keeping it still, or pressing the controller button, resulting in 18 total possible actions. Some actions, for instance, moving the joystick "UP" in Space Invaders, would have no effect given the nature of the game.

ALE was designed to give learning agents an experience playing Atari 2600 games as close to a human experience as possible. There is no information immediately available to any agent about the rules or objects in a given game, only the actual visual screen a human would see from a top-down perspective. This raw visual task has been proven to be very difficult for many learning agents when they are not given more game-specific information [2]. The Atari games themselves, stored in ROM files, are not included with ALE and are available from a separate website [1].

3.2 Other Work

While there has been some previous work done integrating reinforcement learning techniques into training NNs, there haven't been attempts to directly train a NN on the output of an RL agent. Some methods [7] create an online learning method for NNs which directly incorporate rewards as error into NN parameter learning. Most attempts to make more general NN agent training methods which do not require labeled data focus on artificial evolution or other alternate training methods [10].

4 Experiments

4.1 Space Invaders

We chose to run our first experiment on Space Invaders in the ALE. Space Invaders is a game where the player controls a space fighter which can move back and forth along the bottom of the screen and fire bullets at a grid of slowly advancing enemies who also fire back. The player can also take

[1]atarimania.com

cover behind a row of bunkers which will slowly be eroded by enemy fire. The player scores points for destroying enemies, and being hit three times will result in a game over. After a random time interval, a special UFO enemy will fly across the top of the screen. Successfully destroying this will award additional points.

4.1.1 Feature Generation

One major challenge in learning to play a general video game is feature generation in a raw visual environment. We used three different methods of feature encoding in Space Invaders to see which gave us the most interesting behaviors in the RL agent and the trained NN.

4.1.2 Basic Features

This was the most straightforward feature generation, simply giving the learner an approximation of a raw visual field as input. The raw image was down-sampled by a factor of 10 from 160x210 to 16x21, so that each block represents one 10x10 grid of features. For each block, 8 binary features were encoded, each representing the presence or absence of one of 8 colors in that block. For example, if in a certain block there existed the intersection of black background, a grey bullet, and a green alien all three corresponding binary features would be set to 1 for that block. This feature encoding was used in Ballamare et al [2].

This resulted in a total of 2688 binary features as input to the reinforcement learner and neural network. One advantage of this method is that it can be applied generically to any game, as it is completely unsupervised and lacks and domain engineering. However, with no further information the learner must learn to account for large amounts of noise. For instance, the actual numerical representation of the score on the screen is represented in this set of features.

4.1.3 Black and White Features

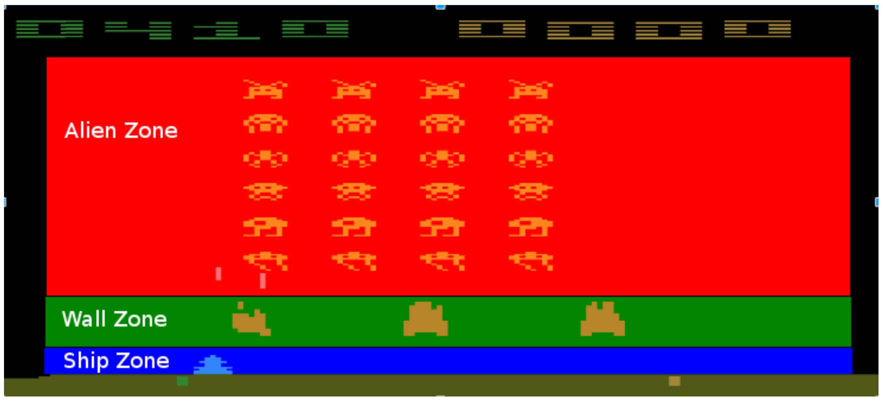

Figure 1: The above figure demonstrates the different zones or domains of the different types of objects in Space Invaders. The Aliens will only every exist in the red zone, the walls in the green zone, and the ship in the blue zone. This means that the specific color should matter less than the presence or absence of any color that is not black.

In Space Invaders, all objects are against a black background. Additionally, all objects except bullets remain in a specified zone as shown in figure 1. This means that simply the presence or absence of any color could encode enough information for the agent, as with the exception of bullets zone is enough to determine what kind of object is represented. Therefore, we created a set of black and white features, one for each 10x10 block, each of which is a binary feature that encodes the presence or absence of any color that is not black in that block. This resulted in the smaller set of 338 features.

4.1.4 Domain Specific Features

A third feature encoding we attempted was a simple domain-engineered set of features. This set of features is constructed from the position of the ship. There are three sets of 21 features, representing vertical bars going in-front of the ship 10 pixels to the left and right of the center. Each feature encodes, on a single vertical line, a scaled distance to the nearest object its group represents. We encoded the nearest alien pixels, wall pixels, and bullet pixels this way for a total of 63 features.

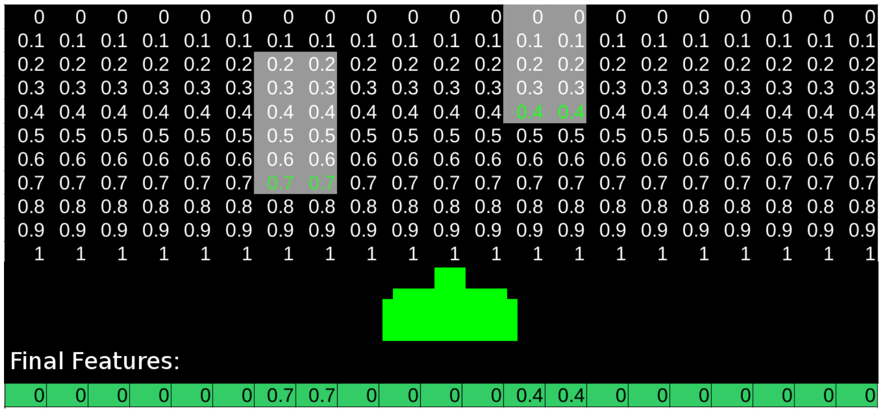

Figure 2: An example of how the 21 bullet features are generated for the domain specific features. Background is shown in black, the ship is represented by the large neon green object, and the final feature vector for the bullet features are shown at the bottom in the green bar.

Figure 2 shows an example of the bullet feature. The bullet feature explores a 21x10 grid of pixels in-front of the ship. In this example there are two bullets, shown in grey, within the ship's field of view. The closest bullet pixels in each vertical bar, shown in green text, are translated into final feature values which are shown at the bottom of the figure. This process is identical for the feature groups of aliens and walls, except that the grid extends vertically to cover the entire target zone from the ship as shown in figure 1. These features are much more informative and specific than the other two methods, but are entirely domain specific.

4.2 Light Foraging

The second experiment was a food-foraging task. The goal of this experiment was to see if the NERL framework could be applied to a noisy and more realistic robotics environment. Robotics simulations were performed in PyRobot[2] with a simulated TK Pioneer robot. The environment consists of an open space with 20 randomly placed "food" pellets. These pellets emit light, allowing

[2]http://code.google.com/p/pyrobot/

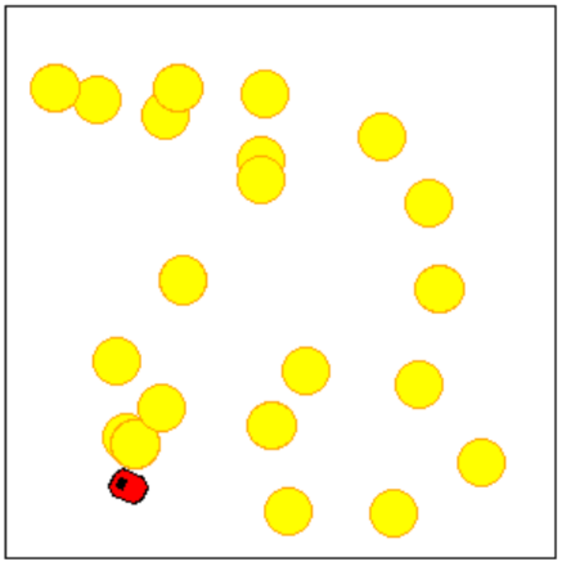

Figure 3: A randomly generated instance of the light foraging environment

the robot to sense them from a distance. Each robot began with an energy level of 20. Each time step removed 0.3 energy while eating a food pellet gave the robot 15 energy.

The robot controller in this experiment was an Approximate Q-Learner. The features of the environment were two light sensors, the current energy level of the robot, and a stall sensor. The robot was given positive reward for eating food and having high energy, while was given negative reward for having low energy and stalling. The robots outputs were its two wheels, allowing it to control its speed and rotation. The RL agent was trained for 10000 iterations, the last 100 of which were used to train the NN agent. Due to the extremely skewed action distribution of the RL agent, we also trained the NN agent on the normalized output of the training data, so that all actions were represented an equal number of times.

5 Results

Here we present the results of how the RL agents performed in terms of reward, and then how well the neural network mimicked the RL agent's actions and how much reward the NN's actions would accrue. While what we were mainly interested in was how well the NN performed, we believe it is also important to understand how well the RL agent is a useful baseline performance, and it is also important to examine the behavior of the RL agent itself.

5.1 Space Invaders

5.1.1 RL agent

The RL agents all reached a basic level of convergence within the 10000 iterations. The basic feature set saw a spike in learning around iteration 150, with a maximum reward of 1080. The black and white feature encoding got between 200 and 300 reward, with slightly higher peaks in later iterations and a maximum reward of 1000. The hard coded feature RL agent converged at a lower reward around iteration 650, but also had the highest maximum reward of three with 1345. There was a large amount of score variance between iterations, though overall trends are apparent in figure 5.

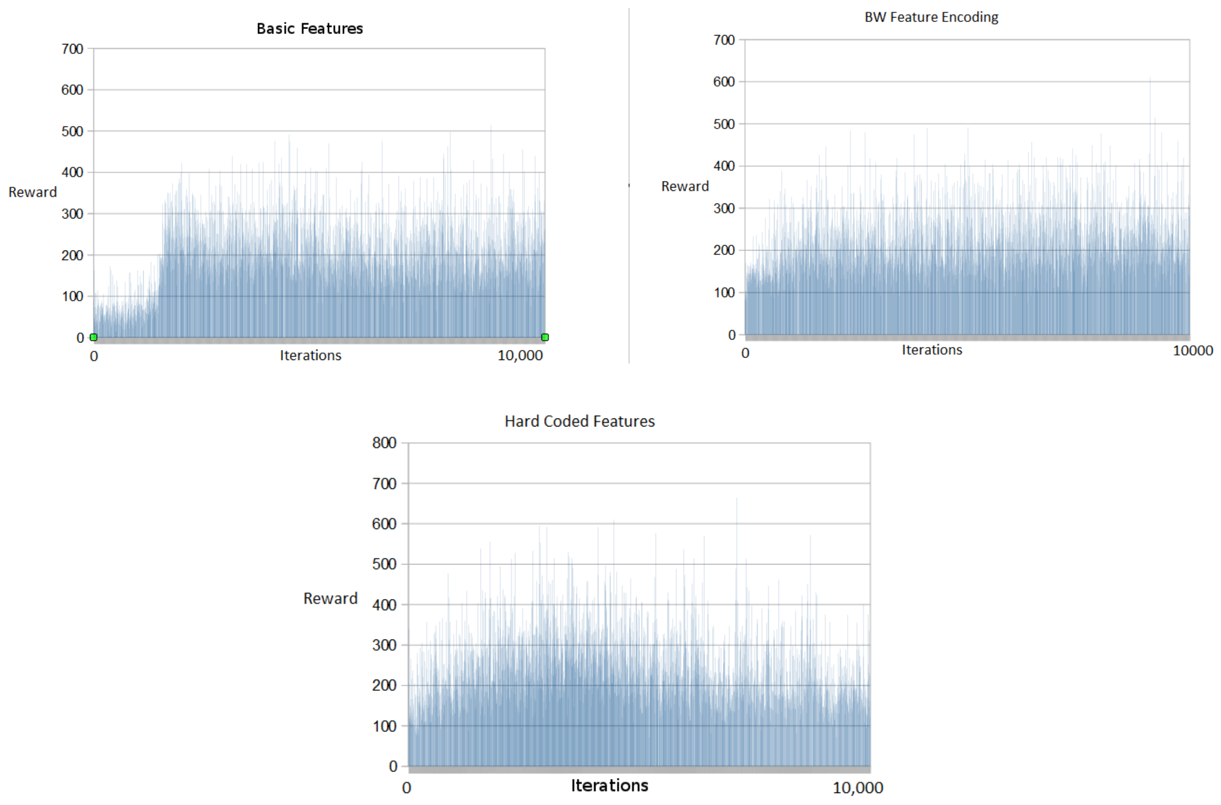

Figure 4: Rewards for the RL agent with the different feature encodings averaged over three runs

5.1.2 Neural Network

Three NN agents were trained, one for each of the different feature encodings. Each NN was trained on a given feature set as input and what action the RL agent took as output. Each NN was trained for either 50 generations or until it reached 2% accuracy, that is, until its behavior matched the RL agent at least 98% of the time. None of the three agents required the total 50 generations to reach 2% accuracy. Given more generations, the NNs oscillated between 2% and 1.5% accuracy without increase in score. There was no significant difference in emulation accuracy between different feature encodings.

In figure 6 we see that on average the NN trained with the basic set of 2688 features as input got the highest score. On the basic feature encoding and the BW feature encoding, the NN agents learned to perform better than their RL counterparts.

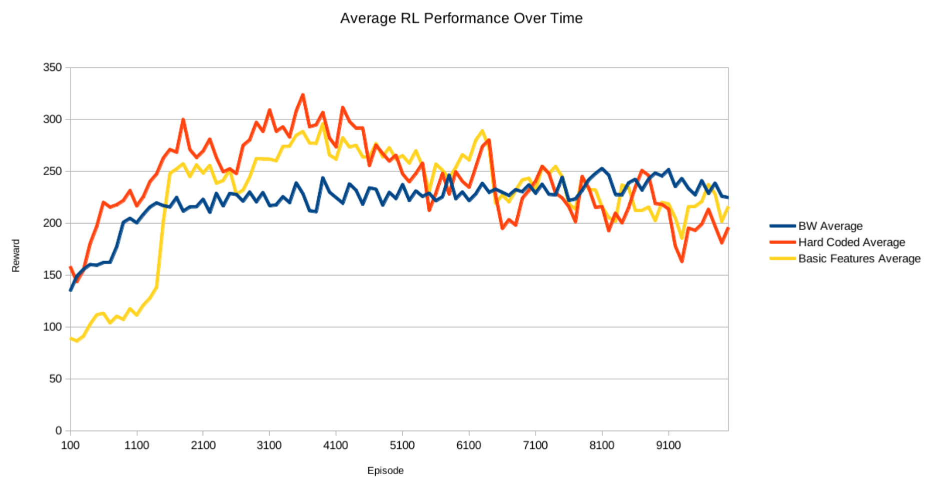

Figure 5: Average Performances of RL agents on Space Invaders with 3 different feature encodings. Each point represents the average of the 100 previous episodes. Each agent

5.2 Light Foraging

The RL agent did not manage to learn a complex behavior. It typically moved forward, curving very slightly toward the nearest light. The average RL reward can been seen in figure 7. When training on raw data, the NN was able to achieve approximately 80% accuracy on the training data. However, this result is deceptive as about 80% of the RL agent's actions were moving forward. On the normalized training data, the NN agent was only able to attain 20% accuracy, which is more representative of its performance. Both NN agents performed worse than the RL agent on the light foraging task, as seen in figure 7.

6 Discussion

6.1 Space Invaders

The RL agents were all able to develop some level of complex behavior when playing Space Invaders. One surprising result what the relative equivalence in performance between the three feature encoding types for RL agents.

The NN agent was able to quickly learn to emulate the RL agent in the deterministic ALE environment with a very high degree of accuracy. This shows that it is possible for the NN agent successfully mimic another learner. Applications of this emulation only goal, such as adversarial machine learning or legacy systems, are a possible future direction.

Our results in the Space Invaders environment show that the NN agent was able to very quickly learn to mimic the RL agent in all the feature encodings. Additionally, in both the basic and BW feature encodings the NN agent actually outperformed the RL agent. Interestingly, the larger the feature space, the more the NN agent outperformed the RL agent. This shows the the NN agent was able to take the noisy RL agent actions and learn a better mapping of actions to difficult feature spaces.

It is important to note that, though this is an impressive result which is the average of 500

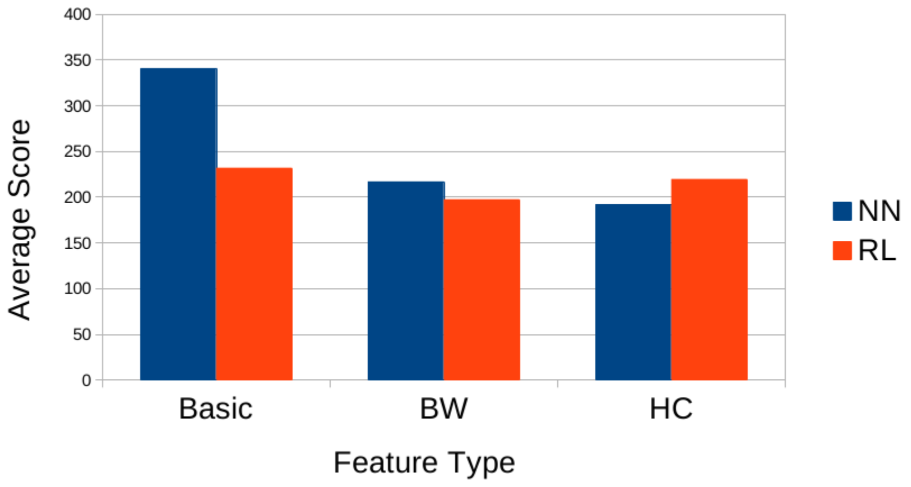

Figure 6: A comparison between the average game score of the RL agent and the NN agent given a particular feature set

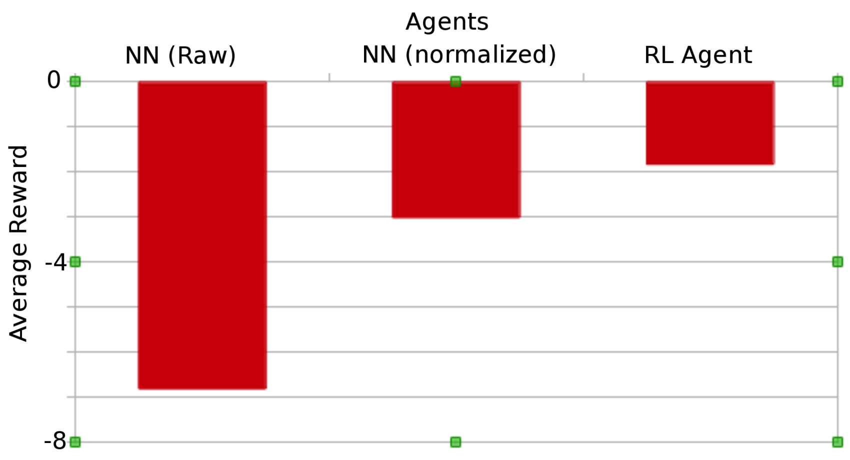

Figure 7: Average reward for the simulated robotic light foraging task for neural networks trained with NERL on raw data, neural networks trained with NERL on normalized data, and an RL agent. More negative scores performed worse.

iterations for multiple runs, there was too much variation of the RL agent between episodes that it cannot be ruled out that this result is not the product of noise. It would likely take at least 50 trials of each learning method to be able to conclusively detect trends in the data. Due to time and computation constraints, this was not possible. ALE was able to be run in parallel on a single machine, which limited the number of trials possible to run in only a few weeks.

6.2 Light Foraging

The light foraging environment proved too noisy and difficult for the RL agent to effectively learn a complex behavioral strategy. We believe that this is because of the lack of recurrence in the RL agent model. Though SARSA learning does store two state-action pairs, and thus has some limited level of memory, light sensors are so sporadic that more general averages over time are needed to determine the direction light is coming in. In addition, though positive reward states were given to the agent these positive states were always followed by more negative reward states as the agent began to lose energy again. Therefore, these positive reward states would have been less positive in the SARSA encoding.

It is likely that NERL could be applied to a more complex and noisy space, as RL agents have been shown to be successful in these kind of environments [9]. However, in the light foraging experiment the RL agent was not successful enough for the NN agent to be able to learn any meaningful behaviors. The 20% accuracy in emulation for the normalized NN agent shows the chaotic nature of the RL learner when accounting for its repeat behaviors.

7 Conclusion

While our results were not entirely conclusive, we believe that they show that NERL has the potential to be an effective technique. Under two of the three Space Invaders feature encodings the evolved NN vastly outperformed the RL agent it was trained on, which is very promising. Also, that the NN was still able to have 2% accuracy in copying the RL agent shows that NNs can learn the behavior that a RL learns quickly. While none of our RL learners developed exceptionally complex strategies, as long as the environment wasn't too noisy (as in the case of the light foraging environment) we see that NNs are very effective at emulating RL agents and may surpass those RL agents in large environments.

7.1 Future Work

In this paper we only used NERL on very similar environments to those it was trained on. Though there is some proof of generalization in that NERL became more successful as the feature space became larger and less domain specific, it would be interesting to create a more general game solver in ALE and then evaluate how NERL performs compared to a normal RL agent on novel games. Additionally, it would be useful to perform experiments on more games. Due to time constraints, we chose to only focus on Space Invaders so we could have the opportunity to create multiple game-specific feature encodings. However, another direction would be so see if a similar setup can be explored in other Atari games.

The light foraging experiment can be explored in two directions. An alternate RL agent could be used which is designed for noisy robotics environments. Additionally, alternate feature encodings which perform more averaging over time could provide a cleaner feature set to the RL learner and allow approximate Q learning to be successful in the noisy environment. However, we only tried one particular hardcoding of the NN so either an evolving topology or deep learning might help solve some of the noise problems here. Overall we believe that although methods like NEAT can also evolve solutions without labeled training data, NERL deserves further investigation as a method to generate multiple machine learning agents.

References

[1] Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 1998.

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *arXiv preprint arXiv:1207.4708*, 2012.

[3] Scott E Fahlman. An empirical study of learning speed in back-propagation networks. 1988.

[4] Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. A neuroevolution approach to general atari game playing. 2013.

[5] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011.

[6] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[7] Chin-Teng Lin and CS George Lee. Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems. *Fuzzy Systems, IEEE Transactions on*, 2(1):46–63, 1994.

[8] Yavar Naddaf. Game-independent ai agents for playing atari 2600 console games. Master's thesis, University of Alberta, Edmonton, Alberta, 2010.

[9] William D Smart and Leslie Pack Kaelbling. Effective reinforcement learning for mobile robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, pages 3404–3410. IEEE, 2002.

[10] Kenneth O Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *J. Artif. Intell. Res.(JAIR)*, 21:63–100, 2004.