The Performance of Symbolic Regression based on Deep Double Q-Networks in an Atari 2600 Environment

Jay Hari

Received September 30, 2024
Accepted February 25, 2025
Electronic access March 15, 2025

The field of Neurosymbolic AI, a field which reconciles symbolic regression with neural architectures, is a developing domain which holds much promise. However, it is imperative that both Neural Networks and Symbolic AI are be compatible for the success of Neurosymbolic AI. Compatibility between symbolic and neural networks would be defined as lossless conversion between both types of algorithm. In this study, we test the effectiveness of conversion from Deep Double Q-Networks to symbolic equations, using a number of modifications to increase the effectiveness of the original neural network. We used the RAM information of Atari 2600 environments, as the data fed into the networks directly corresponds with important game variables. We find that, while many modifications made to the traditional Deep Q-Network (DQN) structure, an architecture related to Reinforcement Learning, greatly increase its efficiency in finding a good policy for a given environment, and while RAM remains a viable source of training information as opposed to direct graphical data, symbolic regression algorithms based on genetic programming principles are largely unfit for representing RAM-based DQNs in the form of equations. We suggest further research to be directed into other forms of symbolic regression other than genetic programming, because these alternative methods may be more apt at converting DQNs to symbolic equations.

1 Introduction

Reinforcement Learning is a field of Machine Learning which mainly deals with exploratory learning in order to find optimal policies to problems it is faced[1]. One of the more popular applications of Reinforcement Learning, Deep Q-Networks, uses neural architectures to produce a non-linear approximation of a value function, and is very popular in many applicable situations, such as recommendation systems, industrial applications, and robotics[2]. Deep Q-Networks are a neural variation of Classical Q-learning, which keep a running table of Q-values to map out the expected reward of any given action in any given situation[3]. In contrast to the relatively newer paradigm of neural machine learning which Reinforcement Learning is a part of, Symbolic AI is a much older and currently underutilized field of AI[4]. Symbolic AI tends to focus on higher-level logical systems, unlike neural architectures. In other words, Symbolic AI derives mathematical equations to solve a given problem. As an example of a subfield of Symbolic AI, Symbolic Regression is analogous to Q-Learning: it is a hallmark technique that commonly defines the field itself[5]. Symbolic Regression has been shown to be useful in cases such as derivation of Newtonian Physics rules, so its utility value cannot be overstated[6]. In recent times, though, Symbolic Regression seems to have fallen out of favor in preference for neural approaches, such as Deep Q-Learning. However, it has been acknowledged recently that neural networks lack the abstract logical capabilities of symbolic algorithms and the interpretability that comes from these algorithms[4]. To this end, the field of Neurosymbolic AI has arisen, which seeks to reconcile both paradigms into a superior hybrid architecture. This new field would allow not only for superior capabilities in artificial intelligence, but also for increased interpretability of these algorithms: performance of neurosymbolic architectures has been reported to be "on par or above benchmarks"[4]. In fact, recent studies have shown neurosymbolic integrations in Large Language Models to enhance reasoning capabilities[7]. Additionally, conversion from pixel-based deep learning models to symbolic equations has been successfully achieved[8]. Neurosymbolic AI has a wide range of possible applications where the versatility of neural networks needs to match our understandings of what they're doing. For example, neurosymbolic AI could be used to automate medical diagnosis, not just by diagnosing off of signs, but also by explaining how different signs and their severities indicate different conditions. Neurosymbolic AI could also be leveraged in the field of autonomous transportation, where a number of factors are better computed neurally (such as the dynamics of turning), and others are best adapted symbolically (such as road signage or lights). The issue of uninterpretability, often dubbed the 'Black Box Problem', is preventive of the abilities of researchers and regulators to comprehend the reasoning behind the actions taken by a given neural network. To deal with this issue in solely neural networks, conversion techniques to first-order symbolic equations have been tested and found to outperform Deep Q-Networks (DQNs) in dynamic

and random environments[9]. Symbolic expressions derived from Deep Learning Networks also generalize "much better than the very. . . neural network[s] [they were]

extracted from"[6]. At this point, DQN methods have been tested extensively on Atari games in a simulated Arcade Learning Environment (ALE)[10]. Q-Learning has been shown to outperform humans in Breakout, Enduro, and Pong, while outperforming other models and underperforming against humans in games that exist over a longer timescale, such as Q*bert, Beamrider, Seaquest, and Space Invaders[11]. Additionally, more complex variations of DQNs, such as Double DQNs (DDQNs), have been proven to perform more accurately than DQNs on Atari games[12]. The combination of superior abstraction capabilities that symbolic models possess, combined with the universal function approximation capabilities of neural networks (NNs) suggest that the symbolic models extracted from NNs can be expected to obtain superior performance. While many of these networks successfully manage to train on pixel data, there exist other sources of input data that can act as feature inputs to a given model, namely RAM. The usage of RAM information confers symbolic benefits, as, unlike pixel data, RAM information is representative of semantic meaning, which would prove useful to the interpretability of symbolic equations that use it as an input variable. Interestingly, Neural Networks trained on the RAM information of Atari games have been shown to perform on par with, or sometimes superior to screen-based models and mixed-input algorithms alike[13]. Given the advantages discussed in the prior discussion, as well as converting them to symbolic regressions, we seek to test the hypothesis that the derivation of symbolic models from DQNs trained on RAM information will be more efficient and provide better performance, as compared to simply training DQNs on RAM or acquiring symbolic models from DQNs training on pixel data. This study seeks to investigate the effectiveness of conversion from Deep Double Q-Networks to Symbolic equations through genetic algorithms. We hypothesize that derivation of symbolic equations from DDQNs will be effective, but the generated equations will be highly complex and inefficient due to the nuances of genetic algorithms. If successful, this study would result in compact, interpretable symbolic equations which perform well when tested in an Arcade Learning Environment.

2 Results

2.1 Derived Symbolic Equations

All equations are simplified where possible, and constants are rounded to two decimal places. Here are the highest-scoring symbolic equations on average for Pong. All of these equations were derived from the Vanilla DQN.

1.

$$a = \text{sgn}(x_6 + x_7) \times (\text{greater}(x_3 - x_5, x_6 + 0.45) - \sin(x_3 - 0.11))$$

• x_3 is the Y coordinate of the ball, x_5 is the change in the Y coordinate of the agent's paddle, x_6 is the change in the X coordinate of the ball, and x_7 is the change in the Y coordinate of the ball.

2.

$$a = \Big(x_5 - \Big(\big(\text{sgn}(x_7) + \text{sgn}(x_6)\big) \times \big(\text{greater}(x_3 - x_5, \frac{x_6}{1.36} + 0.45) - x_0\big)\Big) \times \big(\text{greater}(\sin^2(x_2), (x_7 + x_3) - 0.20) - 0.80\big)\Big) + x_5$$

• x_0 is the Y coordinate of the computer paddle, x_2 is the X coordinate of the ball, x_3 is the Y coordinate of the ball, x_5 is the change in the Y coordinate of the agent's paddle, x_6 is the change in the X coordinate of the ball, and x_7 is the change in the Y coordinate of the ball.

3.

$$a = x_5 - \Big(\sin\big(\text{greater}(x_3 - x_5, x_6 + 0.44) - x_0\big) \times \big(\text{sgn}(x_7) + \text{sgn}(x_6 + \tanh(x_7))\big) \times \big(-0.45 + \big(\text{greater}(x, \sin((x_3 + x_7) - \tanh(0.20))) - x_2\big)\big)\Big)$$

• x_2 is the X coordinate of the ball, x_3 is the Y coordinate of the ball, x_5 is the change in the Y coordinate of the agent's paddle, x_6 is the change in the X coordinate of the ball, and x_7 is the change in the Y coordinate of the ball.

Here are the highest-scoring symbolic equations on average for Seaquest, all calculated with a Pseudo-Rainbow DDQN. There are too many x-variables to list for Seaquest, but any $x_n \quad \forall \quad 14 \leq n \leq 27$ happens to be the delta of a RAM variable.

1. $a = \text{sgn}(x_6 + x_7) \times (\text{greater}(x_3 - x_5, x_6 + 0.45) - \sin(x_3 - 0.11))$

• x_3 is the Y coordinate of the agent, x_5 is the X coordinate of one enemy missile or diver, x_6 is the direction of the agent, and x_7 is the direction of the agent's missile.

2.

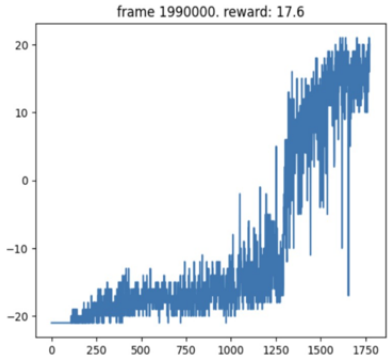
$$a = \frac{\text{greater}(x_2, x_6) - \left(\frac{x_0}{-0.73}\right)^2 + 0.57}{\sqrt{x_3}}$$

• x_0 is the X value of one enemy, x_3 is the Y coordinate of the agent, x_6 is the direction of the agent, and x_2 is the change in the oxygen meter value.

3.

$$a = \frac{\text{greater}(x_4, x_6) - \left(\frac{x_0}{0.75}\right)^2 + 0.57}{\sqrt{x_3}}$$

• x_0 is the X value of one enemy, x_3 is the Y coordinate of the agent, x_6 is the direction of the agent, and x_4 is the change in the oxygen meter value.



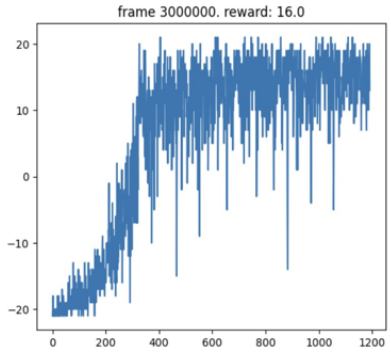


Fig. 1 Learning Curves for Pong

4.

$$a = \frac{\text{greater}(x_4, \sqrt{x_6}) - \left(\frac{x_0}{0.75}\right)^2 + 0.57}{\sqrt{x_3}}$$

• x_0 is the X value of one enemy, x_3 is the Y coordinate of the agent, x_6 is the direction of the agent, and x_4 is the change in the oxygen meter value.

3 Discussion

The first observation made over the course of these experiments was the utility value of Frameskip Decay. As seen in Table 1, not only did the DDQN with Linear Frameskip Decay from 30 to 4 outperform the DDQN with Noise, it also outperformed the DDQN which utilized a Dueling Architecture. Furthermore, when Exponential Discretized Frameskip Decay was used alongside other beneficial additions, such as Prioritized Experience Replay, Dueling Architecture, and Noise, including γ annealing, the model reached convergence in approximately 66.3% of the number of frames that Vanilla DDQN required, as demonstrated in Figure 1. Based on the mean score performance of each individual modification to the DDQN, the Dueling Network architecture proved to be the most successful, followed by Linear Frameskip Decay, Noisy Networks, and finally Frameskip Annealing (Table 1). This performance informed the traits selected for the Pseudo-Rainbow DDQN. However, the symbolic regression module did not perform nearly as well as expected. The equations derived for Pong, shown in Section 2.1, have very low average scores, around -21. Interestingly, the best symbolic policies, attaining a maximum score of -20, were obtained from the Vanilla DDQN, instead of the pseudo-Rainbow DDQN. We hypothesize that this discrepancy is due to the relatively simpler policy that the simpler Vanilla DDQN came up with, although random seeding for the symbolic regressor could also be a factor. In Seaquest as well, results were not ideal. We found that the derived Symbolic Equations (average score: 20.75) performed essentially on par with a policy of random action (average score: 25.8). This demonstrates the ineffectiveness of the symbolic equations as a conversion method from DQNs, even when acting in an environment with relatively less sparse reward systems.

In both experiments, there was a slight correlation between equation complexity and peformance (r=0.84). However, performance started to level off in the higher-complexity equations at around an average score of 112 in Seaquest. The highest-scoring policies that were discovered still performed abysmally compared to the DQNs they were derived from. Overall, when compared to Symbolic Regression as derived from reinforcement learning algorithms in a simpler environment, the derived symbolic equations did not perform to a similar degree [14]. Kamienny & Lamprier's method found symbolic regression to maximize reward within the simpler Cartpole environment, whereas our study found symbolic regression unable to make much progress in relatively more complex environments [14]. As a corollary to that idea, the derived symbolic equations are still not very interpretable, and the intention behind every operation performed within them is ambiguous. As such, this study demonstrates that symbolic regression is not an effective method to use for interpretable conversions of Deep Q-Networks, at least in an Atari 2600 environment.

4 Limitations

Given the nature of this study, some manual optimizations had to made which limited the effectiveness of both the produced DDQNs and the symbolic equations, but allowed for quicker training. We isolated four separate RAM values when training the model on Pong, as explained in Table 2. Ideally, a DDQN model would be trained on all 128 RAM values and be allowed to filter out the irrelevant values naturally, but this was simply not feasible for our experiments, as it would require much more processing time and resources. An additional benefit of allowing unnecessary values to filter out on their own is that it would allow for the symbolic regression module to train only on necessary variables, giving insight into what each RAM value

Table 1 Performance of different DDQN algorithms on Pong for 1 million frames. Note: We use the term Pseudo-Rainbow DDQN to describe a DDQN model which utilizes Frameskip Decay, Prioritized Experience Replay, Noisy Networks, and Dueling Networks, due to their proven effectiveness here.

Model	Mean Score	Highscore
Vanilla DDQN	14.6	21
DDQN with Linear Frameskip Decay from 30 to 41	-8.2	17
DDQN with Linear Frameskip Annealing from 4 to 301	-17.1	13
DDQN with Noisy Networks1	-12.6	10
DDQN with Dueling Architecure1	-6.2	9
Pseudo-Rainbow DDQN	**17.5**	21
Vanilla DDQN-Derived Equation2	**-20.6**	**-20**
Pseudo-Rainbow DDQN-Derived Equation2	-21	-21

may mean in the context of any given game. This would be achieved by providing the entirety of RAM data to the DDQN and the symbolic regressor, and allotting ample training time for the relevant RAM values to be located by both modules. Furthermore, training symbolic regression models on games such as Pong and Seaquest can be challenging, as these games contain sparse rewards, which makes model performance more subject to exploration and renders some of the learning experiences useless, as well as creating an increased necessity for exploration to encounter rewards. If a game with dense rewards, such as Ms. Pacman, was utilized, it would possibly be more successful in training and validation. Also, the highest-level algorithm we leverage, the Pseudo-Rainbow DDQN, is not perfect. Compared to the high-performing Rainbow DQN [15], our model lacks a key feature – distributional networks. If a full Rainbow DQN implementation was to be used, then performance would likely improve. The Rainbow DQN was not implemented in full, because of the number of independent modifications it contains: using all of the modifications would slow down training, and they are not guaranteed to improve performance with novel modifications such as Linear Frameskip Decay and Annealing. We used only Noisy Networks, Dueling Architecture, and Prioritized Experience replay due to their importance in the Rainbow DQN model compared with the other modifications.

4.1 Symbolic Regression

The symbolic regression module itself is not infallible either, though. Because PySR [16] works on the principles of genetic programming, it is susceptible to premature convergence, which typically results in suboptimal solutions. This problem could potentially be rectified by the usage of other non-GP symbolic regression modules such as DSR and AIFeynman, although these methods all have their own flaws [17]. While DSR and AIFeynman, both of which use neural networks, are limited by the issue of overfitting, neither of them can fall short of accuracy indefinitely in a way similar to the occurrence of premature convergence [17]. Another possible limitation to the model was the policy derived by the DDQN. For both Pong and Seaquest, differing gameplay policies exist, i.e. more aggressive playstyles and defensive ones. Each of these different strategies may be characterized by a unique symbolic equation, some of which may be more complex than others. It is then possible that, through pure chance, the DDQNs adopted highly complex, or even overfitted, policies which were difficult for the symbolic regressor to match. Additionally, Pong and Seaquest, despite being relatively simple environments, have somewhat large observation and action spaces (See Section 5.1), which stymies the usefulness of genetically-based symbolic regressors. Even efforts to curtail this issue, mentioned in Section 5.2, still leave a significantly large number of features for the regressor to work with, signaling that genetic programming-based symbolic regressors may be better suited to simpler environments, such as those mentioned by Cranmer et al. [6]. Ultimately, it will depend upon the use case to determine which symbolic regression module is best used.

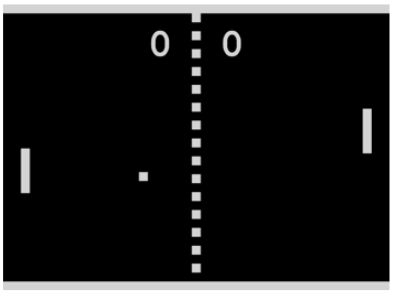
5 Materials and Methods

5.1 Arcade Learning Environment

We utilized OpenAI Gymnasium to use a suitable Arcade Learning Environment [10]. This module allowed us to interface with a number of Atari games to train deep reinforcement models on, of which we chose Pong and Seaquest [18]. Both of these games were chosen for their relative simplicities: Pong is a comparatively basic game, while Seaquest requires more advanced control.

5.1.1 Pong

In Pong, a paddle is controlled in order to deflect a ball into an opposing paddle's goal. Motion is vertical, and there is an action space of cardinality $|A| = 6$, meaning that there are 6 possible actions to be taken in any situation. The game runs on 128 bytes of RAM, and is considered to be a very simple game. We passed 8 specific values from the memory into the DDQN, four distinct



(a) Pong.

(b) Seaquest.

Fig. 2 The two Atari 2600 games we focus on

features, as well as their rates of change over the past frame in order to account for temporal differences. These variables were the y-value of the opponent's paddle, the y-value of the agent's paddle, and the coordinates of the ball. More on these choices in Section 4. See Figure 2a. This game was chosen due to its relatively simple set of input variables and action space. Out of the 128 bytes of RAM data, only four of them were used and the action space, while consisting of a larger cardinality, ultimately comprises just three possible actions: moving up, moving down, and staying still. This game was chosen in order to provide a simple environment to the DDQN in the hopes that it would train quickly, and also that the symbolic regressor would derive a noncomplex equation.

5.1.2 Seaquest

Seaquest is a game in which the a submarine must navigate around a the ocean to collect loose divers while avoiding sharks and managing to surface for air. The player may move in any of four directions at a constant speed, and they may also fire torpedoes. This game also runs on 128 bytes of RAM, and is considered more difficult than Pong due to the multidirectional control, extra commands, and the larger number of dynamic obstacles. It resides within an action space with a cardinality of $|A| = 18$, meaning that 18 different actions can be taken. Since a game of Seaquest could theoretically run forever with limited to no action on the part of the agent, we implemented an 8,000 frame time frame for the agent to act, which would be followed by a timeout. See Figure 2b. Compared to Pong, Seaquest contains a much higher action space and many more relevant RAM variables. This game was chosen to test the boundaries of the DDQN-symbolic equation conversion method, and to compare the method's performance in both simple and complex environments.

5.2 Symbolic Regression

We use the symbolic regression module PySR for its practicality and versatility, as well as its recency [16]. The module takes a genetic programming approach to developing symbolic representations and uses multiple populations of equations to develop, allowing diversity in the gene pool and final results [16]. As such, we used this package for all symbolic equation derivation. It is worth noting that other symbolic regression algorithms may produce different or more accurate equations, but genetic programming proved to be the most accessible. For reference, 10 out of the 14 models included on SRBench, a benchmark for symbolic regression algorithms by La Cava et al. [17], leveraged genetic programming. We derived the examples that the regressor module was trained on through the replay buffer we used. First, we had attempted to train the regressor to predict the expected value of any action based on the input state, for example [state1, state2, ..., staten, action]. However, we found that this approach gave the regressor too many input features to train on, leading to slow training, and eventually unremarkable results. In light of this realization, we switched the method of symbolic regression around. We provided the current state as the input features, and instructed the regressor to predict the ideal action under that circumstance. This approach, helped along by a few other optimizations, performed better than the former method, partly because of the observation space limitation. We found that the action space provided by OpenAI Gymnasium typically contained redundant commands. For example, while Pong had an action space cardinality of $|A| = 6$, there were truly only 3 actions: move up, move down, and do nothing. As such, we decided to limit the output features of the symbolic regressor by collapsing the entire action space of Pong into 3 separate values, and spacing them evenly on the interval [-1, 1]. We found that this modification also allowed the regressor to converge more quickly on a more optimal result.

Additionally, we implemented one last feature to attempt to find the most optimal solutions. After training, we evaluated every single one of the solutions that PySR outputted (The PySR regressor usually outputs one final pick based on accuracy, but maintains a running "Hall of Fame" for the most accurate solutions with different complexities). The logic behind this

Fig. 3 Main Study Design

Table 2 Denotations of each individual feature input for Pong to the DDQNs, as utilized by the derived symbolic equations. a is the action taken in any given circumstance.

Data Point	Original	Δ
Y coordinate of computer paddle	x_0	x_4
Y coordinate of agent's paddle	x_1	x_5
X coordinate of ball	x_2	x_6
Y coordinate of ball	x_3	x_7

decision was that while the equation with the least accuracy will match the best

DQN decisions, the equation that is most optimal may be slightly off from the DQN, even if it was trained to convergence. The parameters passed into the PySR regressor are reproduced in Table 3.

5.3 Reinforcement Learning

Reinforcement learning is a paradigm of optimal control in the field of Machine Learning which seeks to train models using a sort of trial-and-error process [1]. While this field is not entirely neural in nature, branches of it such as Deep Reinforcement Learning have proven to be highly useful in applicable cases [1].

5.4 Classical Q-Learning

Q-Learning, as defined classically, is a Reinforcement Learning algorithm which maintains a table of Q-values for each action it considers for every given state it is presented, where each Q-value represents how rewarding any given action would be in any given state [3]. These Q-values update according to the Bellman Equation:

Maximum predicted reward, given new state and all possible actions:

$$\text{NewQ}(s,a) = Q(s,a) + \alpha \left[R(s,a) + \gamma \max Q(s',a') - Q(s,a)\right]$$

Where: - NewQ(s,a) is the new Q-value. - $R(s,a)$ is the reward. - γ is the discount rate.

These calculations are mitigated with Deep Q-Learning, which replaces the Q-table with a Neural Network to determine Q-values for each action based on a given state [2].

5.5 Deep Q-Learning

We utilized a Deep Double Q-Network (DDQN) algorithm for our practical experiments. The DDQN consists of a DQN with an additional neural network involved to prevent overestimation [19]. We chose the DDQN for its high stability and reliability in Reinforcement Learning situations, and because of its proven accuracy in an Atari 2600 Arcade Learning Environment as compared to single DQNs [12]. The DDQN was programmed to the specifications of van Hasselt et al. [19]. We utilized a model with three hidden layers of 128 neurons each, which corresponded to RAM inputs. These layers used Rectified Linear Units (ReLU) as activation functions [20] All DDQN models were trained on Tesla T4 or RTX 4070 GPUs in every experiment.

Table 3 The parameters which were passed into the PySR Regressor module.

Argument	Value
Processors	4
Populations	8
Cycles per Iteration	500
Iterations	10000000
Binary Operators	$[+,,,]$
Unary Operators	[sin, relu, exp, tanh, log, sqrt, sign, square]
Early Stop Condition	loss $< 1 \times 10^{-3}$
Timeout	18000s
Batching	True
Batch Size	128
turbo	True
bumper	True
Select k Features	9
Model Selection	"accuracy"
Max Size	50
Max Depth	10
Progress	TRUE

5.5.1 Prioritized Experience Replay

We utilized Prioritized Experience Replay to the specifications of Schaul et al. [21] in order to build on the replay buffer we utilized. Replay buffers serve the purpose of storing valuable experiences so that the DDQN can train on them. This implementation included proportional-based ranking, as well as the same α and β_0 values as specified in the paper, 0.6 and 0.4, respectively [21]. We annealed β_0 from its original value to 1.0, also as described by the paper. Additionally, we specified that ε = 1e-5. This allows the most helpful experiences to be trained on the most, alongside downweighting to prevent overtraining. In every experiment, with or without Prioritized Experience Replay, a buffer size of 10,000 was used. We created a new replay buffer when testing each network and filled it with experiences from testing, so that variations during training would have no effect.

5.5.2 Noisy Networks

In our implementation of Noisy Networks[22], we utilized the PyTorch Reinforcement Learning package, which suited our purposes well enough. When using this with Dueling Networks (Section 5.5.3)[23], we placed noise on only the

advantage and value streams, while keeping all prior layers unaltered. In essence, this feature offers an alternative to epsilon-greedy exploration, but gives the added benefit of training the model to denoise. The use of introducing noise, or small errors, in input information is to train the network to discern the relevant information from the noise to enhance its decision-making capabilities.

5.5.3 Dueling Network Architecture

We implemented Dueling Networks in a straightforward manner, as according to the specifications of Wang et al.[23]. Dueling networks prove to be beneficial by allowing the DDQN to come up with two independently correlated variables, rather than a single Q-value, which allows for faster and more simple backpropagation.The dueling network architecture introduces another neural network to the model which predicts the advantage of each action; the Q-value from the first network and the advantage are then combined together to predict the next best possible action[23].

5.5.4 Frameskip Decay and Annealing

We tested both Decay and Annealing of the frameskip parameter in the OpenAI Gym library between the values of 8 and 30 in order to test for any noticeable effects that could be incorporated into the Pseudo-Rainbow DDQN. The intention behind testing these features was to replicate the training of the human reflex[13]. One can think of this concept as a tennis player who gets better at the sport. At first, they only see the ball coming at them, but they begin to notice the subtleties of the ball as they get better, such as the spin it has and its trajectory. In a similar fashion, the frameskip decay method trains the model on basic details, while slowly increasing the granularity of detail the DDQN takes in to smoothen training. We also tested annealing of the frameskip variable to test the inverse theory. In general, we used linear decay and annealing to change the frameskip parameter over the training lifetime. In our original experiment testing the performance of frameskip decay and annealing, we used linear decay of the frameskip variable from the intial value of 30 to the final value of 4, or the reverse. However, we have discovered that this basic implementation prevents the model from properly taking time to adjust to each separate frameskip change, producing instability in results. As a solution to this problem, we created a set path of decay over four values: [30, 15, 8, 4]. For each time the frameskip approximately halved, we doubled the amount of time the model trained for, up to a total of 2,000,000 frames in our final Pseudo-Rainbow DDQN. Additionally, in order to create healthy downweighting in the earlier stages, where results were less accurate, we set the weight decay parameter, γ, to $0.9916 \approx 0.851$, and set $\gamma = \sqrt{\gamma}$ every time the frameskip value halved. This adaptive modification of the γ variable served to keep the loss of the value function more stable as the frameskip parameter changed. We describe this method as Discretized Exponential Frameskip Decay.

5.5.5 Technical Details

We used a variety of specialized tricks to keep the network from becoming too unstable during training. We first scaled the input RAM values from the range [0, 255] to [0, 1]. Additionally, we used Perturbed Rewards[24], which similarly constrain rewards from their original range to the range [-1, 1]. Both of these modifications served to keep the network from handling inputs which could be too noisy, and which could contribute to instability during training and poor results. When not testing Frameskip Annealing, the DDQN was using a Frameskip parameter of 4. The specific values used as bounds for frameskip decay and annealing, 30 and 8, were chosen due to their standard use in Atari-based reinforcement learning studies. For almost all experiments2, DDQN models were trained over 1 million frames at a learning rate of 0.0005 with an Adam optimizer. This learning rate parameter was determined after setting an initial learning rate of 0.0001 for initial experiments and adjusting according to the performance of the network. Based on Table 1, it can be deduced that this learning rate was quite relevant to model accuracy.

5.6 RAM Preprocessing

Due to the size and complexity of the entire RAM data provided for both Pong and Seaquest, the amount of RAM inputs to the DDQNs were limited to a few key values. These important RAM values, along with what they represented were found from "Unsupervised State Representation Learning in Atari" by Anand et al.[25]. The implications of this decision are evaluated in Section 4.

6 Concluding Statements

As the field of neurosymbolic AI grows, reconciliations between symbolic AI and neural methods will become increasingly important to ensure that the best of both worlds can be utilized in future applications using neurosymbolic architectures. Part of this reconciliation is the development of conversion methods between both neural algorithms and symbolic ones. As such, this study sought to explore the effectiveness of these conversion methods to test if there was any clear benefit to RL-derived Symbolic Equations. We conclude that, at least in an environment containing sufficiently large action/observation spaces, Symbolic Regression fails to accurately and efficiently find optimal solutions or even direct replications of Deep Q-Networks. We find many possible factors that this failure can be attributed to,

but we believe that the most relevant detail is the usage of evolutionary symbolic regression, which is dependent on random chance and is known to be slow. We believe further research would benefit from being directed towards more effective symbolic regression methods, which should be able to rectify a number of the issues that have led to these results. For example, a comparison of the effectiveness of different symbolic regression methods on deriving policies from DQNs may be beneficial to discovering more efficient and accurate symbolic equations, and the effectiveness of these different methods should be tested in environments of different reward densities to find their limits.

7 Acknowledgments

The author thanks Alex Goodall of Imperial College for his guidance and assistance in the production of this research paper.

References

1 Y. Li, *Deep Reinforcement Learning: An Overview*, 2018, https://arxiv.org/abs/1701.07274.

2 V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu *et al.*, *Human-level control through deep reinforcement learning*, 2015.

3 C. J. Watkins and P. Dayan, *Q-learning*, 1992.

4 K. Hamilton *et al.*, *Is neuro-symbolic AI meeting its promises in natural language processing? A structured review*, 2022, http://dx.doi.org/10.3233/SW-223228.

5 J. R. Koza, *Genetic programming as a means for programming computers by natural selection*, 1994.

6 M. Cranmer *et al.*, *Discovering Symbolic Models from Deep Learning with Inductive Biases*, 2020, https://arxiv.org/abs/2006.11287.

7 H. Xiong *et al.*, *Converging Paradigms: The Synergy of Symbolic and Connectionist AI in LLM-Empowered Autonomous*, 2024, https://arxiv.org/abs/2407.08516.

8 N. Grandien, Q. Delfosse and K. Kersting, *Interpretable end-to-end Neurosymbolic Reinforcement Learning agents*, 2024, https://arxiv.org/abs/2410.14371.

9 M. Garnelo, K. Arulkumaran and M. Shanahan, *Towards Deep Symbolic Reinforcement Learning*, 2016, https://arxiv.org/abs/1609.05518.

10 M. G. Bellemare *et al.*, *The Arcade Learning Environment: An Evaluation Platform for General Agents*, 2013, http://dx.doi.org/10.1613/jair.3912.

11 V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves *et al.*, *Playing Atari with Deep Reinforcement Learning*, 2013, https://arxiv.org/abs/1312.5602.

12 F. Moreno-Vera, *Performing Deep Recurrent Double Q-Learning for Atari Games*, 2019, https://arxiv.org/abs/1908.06040.

13 J. Sygnowski and H. Michalewski, *Learning from the memory of Atari 2600*, 2016, https://arxiv.org/abs/1605.01335.

14 P.-A. Kamienny *et al.*, *Symbolic-Model-Based Reinforcement Learning*, 2022.

15 M. Hessel *et al.*, *Rainbow: Combining Improvements in Deep Reinforcement Learning*, 2017, https://arxiv.org/abs/1710.02298.

16 M. Cranmer, *Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl*, 2023, https://arxiv.org/abs/2305.01582.

17 W. L. Cava *et al.*, *Contemporary Symbolic Regression Methods and their Relative Performance*, 2021, https://arxiv.org/abs/2107.14351.

18 G. Brockman *et al.*, *OpenAI Gym*, 2016, https://arxiv.org/abs/1606.01540.

19 H. V. Hasselt, A. Guez and D. Silver, *Deep reinforcement learning with double q-learning*, 2016.

20 V. Nair and G. E. Hinton, *Rectified linear units improve restricted boltzmann machines*, 2010.

21 T. Schaul *et al.*, *Prioritized Experience Replay*, 2016, https://arxiv.org/abs/1511.05952.

22 M. Fortunato *et al.*, *Noisy Networks for Exploration*, 2019, https://arxiv.org/abs/1706.10295.

23 Z. Wang *et al.*, *Dueling Network Architectures for Deep Reinforcement Learning*, 2016, https://arxiv.org/abs/1511.06581.

24 J. Wang, Y. Liu and B. Li, *Reinforcement Learning with Perturbed Rewards*, 2020, https://arxiv.org/abs/1810.01032.

25 A. Anand *et al.*, *Unsupervised State Representation Learning in Atari*, 2020, https://arxiv.org/abs/1906.08226.

