MAGNets: Micro-Architectured Group Neural Networks

AAAI Track

Sumanta Dey
IIT Kharagpur
Kharagpur, India
sumanta.dey@iitkgp.ac.in

Briti Gangopadhyay
IIT Kharagpur
Kharagpur, India
briti_gangopadhyay@iitkgp.ac.in

Pallab Dasgupta
Synopsys Inc
San Francisco, USA
pallabd@synopsys.com

Soumyajit Dey
IIT Kharagpur
Kharagpur, India
soumya@cse.iitkgp.ac.in

ABSTRACT

Reinforcement Learning (RL) algorithms have successfully achieved human-like performances in complex environments like games, autonomous vehicles, and industrial robots. However, the Deep Neural Networks (DNNs) used to approximate large Deep Reinforcement Learning (DRL) policies are resource-hungry and opaque. This limits the applicability of DRL in safety-critical applications running on resource-constrained platforms. On the other hand, on inspecting the design of most multi-output safety critical embedded control applications, it may be observed that such systems often derive each output based on some artifacts, which are, in turn, derived from input variables. Given such dependencies of internal system states on inputs, one may argue that each of these derived artifacts can be approximated by a smaller network in a multi-network DRL setting. In this work, we propose Micro Architecture Group Neural Networks (MAGNets) that can distill the learning of a large DRL network into multiple small neural networks. Using several OpenAI Gym environments, we show that existing verification tools can be used to verify the output of MAGNets while preserving the performance of a large neural policy. We also report our gains in network compactness, which directly impacts the execution latency and applicability in edge devices.

KEYWORDS

Reinforcement Learning; Edge Machine Learning; Edge Computation; Knowledge Distillation

ACM Reference Format:
Sumanta Dey, Briti Gangopadhyay, Pallab Dasgupta, and Soumyajit Dey. 2024. MAGNets: Micro-Architectured Group Neural Networks: AAAI Track. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024,* IFAAMAS, 9 pages.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

1 INTRODUCTION

Reinforcement Learning (RL) is a feedback-oriented machine learning method, a feature that makes it a natural choice for self-learning of control policies. RL estimates an optimal state-to-action mapping for an agent by exploring an environment following Markov properties with reward feedback. However, classical RL techniques like Q-Learning [31] do not scale for real-world control problems working on large state-action spaces. Deep Reinforcement Learning (DRL) addresses scalability using Deep Neural Networks (DNNs) as policy approximators. DRL has successfully learned control policies often outperforming human beings in game-playing [28] and has shown promise in deployable technologies like autonomous driving [18], unmanned autonomous vehicle control [19], traffic intersection management [21], power-grid control [32] etc. However, two primary concerns that restrict the use of DNNs on real-time edge devices are as follows. Firstly, embedded edge devices are often resource-constrained in terms of memory and computation power. Though DRL training algorithms have been subject to significant research and are well understood, choosing the right neural network architecture still remains an art, often resulting in overly complex networks with many parameters [5] requiring large storage space and high computation power. Secondly, DNNs are known for their opaque structure with non-explainable outputs [25] and are, therefore, difficult to verify against safety specifications.

Safety critical systems are often designed to satisfy inductive invariants, essentially asserting that the system never takes a transition from a state, $\vec{x}$, to a state, $N(\vec{x})$, if $\vec{x}$ satisfies a safety precondition, $P(\vec{x})$, and the next state, $N(\vec{x})$, satisfies a post-condition, $Q(N(\vec{x}))$ modeling unsafe states. In DRL, the function N is condensed in a neural network. Therefore the verification problem (as detailed in [1]) is to check the validity of:

$$\forall \vec{x} \; (P(\vec{x}) \Rightarrow \neg Q(N(\vec{x}))) \tag{1}$$

The main challenge here is to keep the network N small enough for verification. Recently, a multitude of methods and tools have been developed for neural network verification [1, 14–16] and output range prediction [8]. However, all of them suffer from scalability limitations, rarely going beyond 300-400 neurons in size and limited in handling only specific types of activation functions [15]. In that way, neural network size is a critical determinant for verifying DRL policies. We provide an approach for distilling a DRL network into multiple verifiable networks of small sizes. We show this succeeds

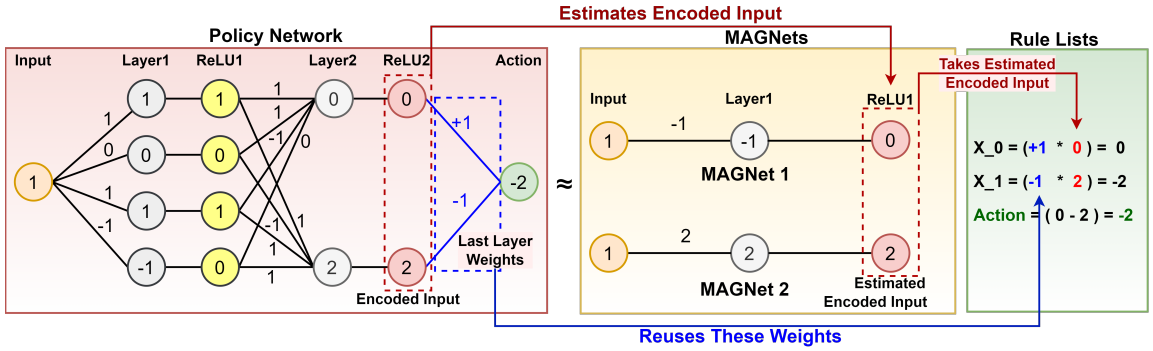

Figure 1: A toy example illustrating policy network to Rule List conversion using MAGNets.

in many control applications where a limited number of derived artifacts decide all control outputs.

Interpretability is one of the key requirements for safety certification. An interesting line of work uses neural policy search to construct policies in interpretable forms like conventional programs [30] [22] [7] or decision trees [3] [10] [20]. Existing approaches towards generating program policies rely on designing a program grammar, on which the quality of the generated program depends. Constructing an optimal grammar requires in-depth knowledge of the underlying systems. On the other hand, a small feed-forward neural network with ReLU activation can be translated into a conventional program. In our framework, by factoring large DRL networks into multiple small networks using knowledge distillation [2] [12], we are able to provide interpretable codes for each control output.

In this paper, we propose the *Micro Architecture Group Neural Networks* (**MAGNets**) framework[1]. We break the neural structure into smaller networks, representing derived artifacts responsible for determining the control outputs. The smaller neural networks and a set of rules (Rule List) that define each control output as a function of the derived artifacts jointly represent the RL policy. Due to the small size of neural networks, each component can be verified using available verification tools, and the policy can be converted into a conventional program. The main contributions of the proposed **MAGNets** framework are as follows:

- We show that a set of small neural networks (MAGNets) can learn an intermediate representation or encoding for the input states.
- We show that a DRL policy can be converted into an interpretable Rule List, namely a collection of linear equations that consider MAGNets as variables and calculate the final control outputs (action values). Each rule can be further expanded along with the MAGNets into conventional programs.
- We show that the final policy consisting of MAGNets and the Rule List can be verified using existing verification tools like Sherlock [8].
- We provide results on several OpenAI Gym environments and illustrate that the MAGNets policies are, on average, 9 times smaller in terms of parameter count than the original network.
- We provide results showing a MAGNets policy is much more suitable for edge devices, considering its lower latency and memory footprint than the original DRL policy.

It is important to note here that the primary focus of this work is on obtaining compact DRL-based policies that are better suited for resource-constrained edge devices. Due to platform constraints, the large policy network is problematic for deployment as shown in [29][23]. Edge-aware on-device machine learning also has benefits like latency, privacy, connectivity, size, and power consumption [23]. For example, a large policy network for a very lightweight drone is not suitable due to power and size constraints. Then, the policy network should run in an edge server and rely on communication links for every simple and frequent control decision. The above issues motivate our work to obtain stipped-down NN-based policies with similar performance. The compaction of the policy network indirectly contributes to verifiability as we scale down to a size that formal/semi-formal tools can handle.

2 THE MAGNETS FRAMEWORK

Interpretable models like linear regression models and decision trees are easy to verify [4]; however, simple linear mapping between states and action might not always hold for a control policy. Large neural networks have higher learning capacities, capturing relationships beyond linear correlations. However, they are hard to verify, even utilizing state-of-the-art verification methodologies. Therefore, the framework aims to generate a set of distilled networks smaller in structure and parameters than the original DRL policy network, which can be verified using existing verification tools.

We propose **M**icro **A**rchitecture **G**roup **N**eural **Net**works Framework or MAGNets Framework to address the learning capacity vs. verification trade-off by replacing a large policy network with multiple interpretable Rule List or linear equations. However, rather than directly using the input states as parameters for the linear equations, we use an intermediate encoded representation or the *latent representation* of the input states. The intermediate representation is captured using multiple smaller neural networks.

We start with a handcrafted motivating example to elucidate our methodology, as shown in Figure 1. The policy network converts

[1]Codebase and appendix: https://github.com/sumantasunny/MAGNets.git

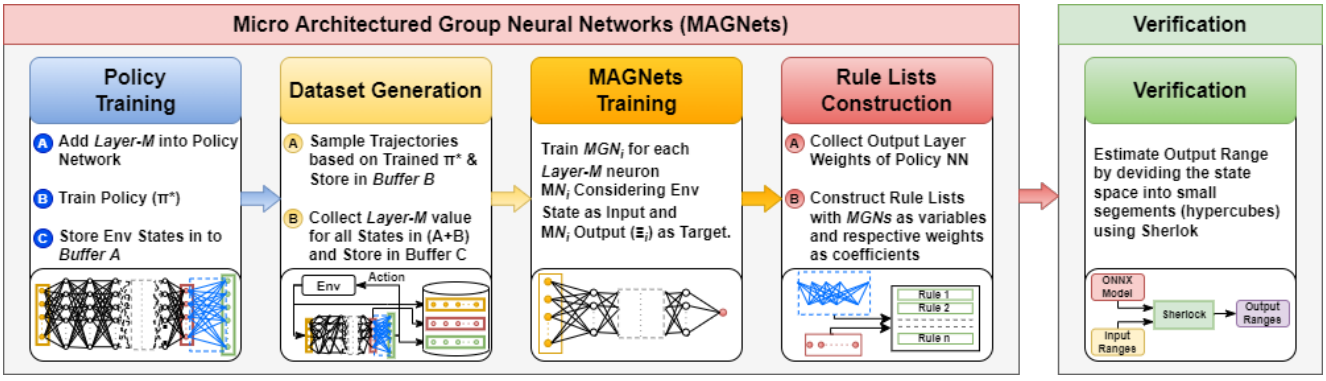

Figure 2: Overall Process Flow of MAGNets Framework

the input value (1) into an encoded feature space $(0, 2)$. This encoded representation can be captured with two simpler networks containing only one neuron each, abstracting out the multiple neurons and connections between layers 1 and 2. The final output can then be obtained using a set of linear equations called a Rule List, where the encoded input $(0, 2)$ is multiplied by the weights of the last layer $[1, -1]$ of the original policy network. The MAGNets framework aims to find such structurally simple networks that mimic the original policy.

The MAGNets Framework consists of four phases: namely, Policy Training, Dataset Generation, MAGNets Training, and finally Rule List Generation. Figure 2 depicts the process flow of our MAGNets framework. We describe each phase in detail as follows:

2.1 Policy Training

We consider an RL policy training problem instance where the policy is trained using PPO (Proximal Policy Optimization) [27]. To capture the intermediate encoded representation of the input, one extra fully connected layer is added between the last hidden layer and the output layer of the policy network, termed Layer-M, as shown in Figure 3. Since the number of MAGNets depends on the number of neurons in Layer-M, we aim to keep this layer's neurons as small as possible without degrading performance. In our study, we manually tune the size (number of neurons) and activation function of Layer-M. We assume an optimal policy π^* is obtained post-training. In buffer A, we store the states that the RL agent visits during training.

Essentially, Layer-M is introduced to reduce the number of MAGNets required for environments having large action spaces. For environments having small action spaces, Layer-M may be excluded and the action logit values can be directly used to train the MAGNets. We present the generic approach in the paper as sometimes we have to consider a slightly larger Layer-M size than the action space size to perform similarly to the original PPO network.

2.2 Dataset Generation

After the policy training phase, we sample multiple trajectories based on the final trained policy π^* with respect to a starting state distribution μ. We consider an environment state $(\vec{x})$ and values of Layer-M neurons as a single data item represented in terms of tuple

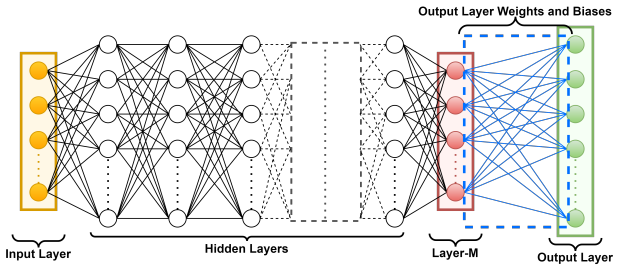

Figure 3: Modified Policy Network with Layer-M

$\langle \vec{x}, \Xi \rangle$, where, $\Xi = \langle \Xi_1 \dots \Xi_m \rangle$ depicts the encoded representation of the input state $\vec{x}$. The i-th neuron output is denoted as follows.

$$\Xi_i \leftarrow \pi^*(\vec{x})\Big|_{LayerM}^{i}$$

These data items are collected for all states of the sampled trajectories and are stored in buffer B. In a similar way, we calculate the Ξ values for all the states present in buffer A, collected during training using π^*. We merged these two buffers A and B to construct the training dataset C.

2.3 MAGNets Construction and Training

We build and train m number of single regression neural networks (MAGNets) where m is the number of neurons in Layer-M. Our primary aim is to construct decomposed networks with fewer parameters. Therefore, we consider a small network for each MAGNet with substantially fewer hidden layers and neurons, experimentally determined, compared to the original policy. The input size of each MAGNet is equivalent to the cardinality of the environment state while it has a single output unit. We consider ReLU as the activation function for all the neurons as it can be represented with only two linear segments aiding verification.

Given the overall training dataset C, the i-th MAGNet is trained on data pairs of the form $\langle \vec{x}, \Xi_i \rangle, \forall i \in \{1 \dots m\}$. This is because the i^{th} MAGNet takes $\vec{x}$ as input and produces the value of i^{th} neuron of the Layer-M (Ξ_i) as the target output. An illustrative depiction is shown in Figure 4 (highlighted in yellow). Therefore, each MAGNet (MGN_i) is trained to estimate the respective index of the encoding

of state, and combining all the MAGNets output will produce the estimated encoded state given as follows.

$$\hat{\Xi} \leftarrow \Big[MGN_i(\vec{x})\Big]_{i=1\ldots m}$$

2.4 Rule List Generation

Given an input $\vec{x}$, our network of MAGNets produces the estimated encoded state $\hat{\Xi}$ as output. We use the weights and biases between Layer-M and output layers of π^* to find a linear mapping from the encoded state $\hat{\Xi}$ to the set of rules *Rule List*. The evaluation of a rule $r_i \in$ *Rule List* produces the logit value of an action $a_i \in \mathcal{A}$. Here, $\mathcal{A}$ represents the RL agent's action set. The final Rule List can be mathematically represented as follows.

$$Rule\ List : \Big[r\Big]_{n\times 1} \leftarrow Weights\Big|_{LayerM} \times \hat{\Xi} + Biases\Big|_{LayerM}$$

Here $Weights\Big|_{LayerM}$ and $Biases\Big|_{LayerM}$ represent the weights and biases of Layer-M of the modified policy network. Figure 4 depicts the overall construction process of the Rule List.

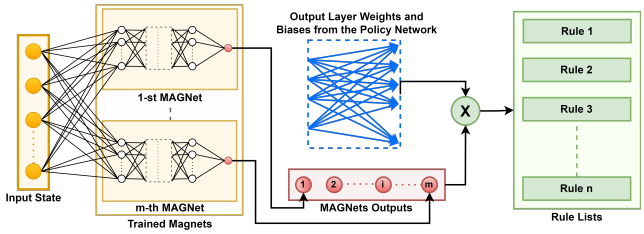

Figure 4: Rule List Construction

2.5 Verification Strategy

As discussed in Section 1, we consider that a policy can be verified if we can correctly compute that the output range of $N(\vec{x})$ (from Eq. 1) is safe. In this work, we use Sherlock [8] for statically computing the output range of individual MAGNets. The calculated values are then used to predict the output ranges of the rules, using which we can identify possible action values for chosen input ranges.

The output range estimated by Sherlock is guaranteed to be tight for a set of input ranges forming a polyhedron ([8], Theorem 1.1). Therefore, we consider standard region-based abstraction on the inputs to improve the scalability of the verification process. The state is defined over an n-dimensional hyperspace. We divide the hyperspace by splitting each environment state variable into multiple range segments based on a chosen granularity. The n-dimensional ranges form hypercubes containing a subset of the state space. Then, we use Sherlock to predict the output ranges of the policy (see Figure 5a).

In the verification pass, the choice of input state space abstraction granularity controls the trade-off between the time required for verification and the chance of obtaining deterministic actions (for discrete action space) or more coarse-grained action ranges (for continuous action space) with safety verification. For example, consider an input space x comprising two variables v_1, v_2 and

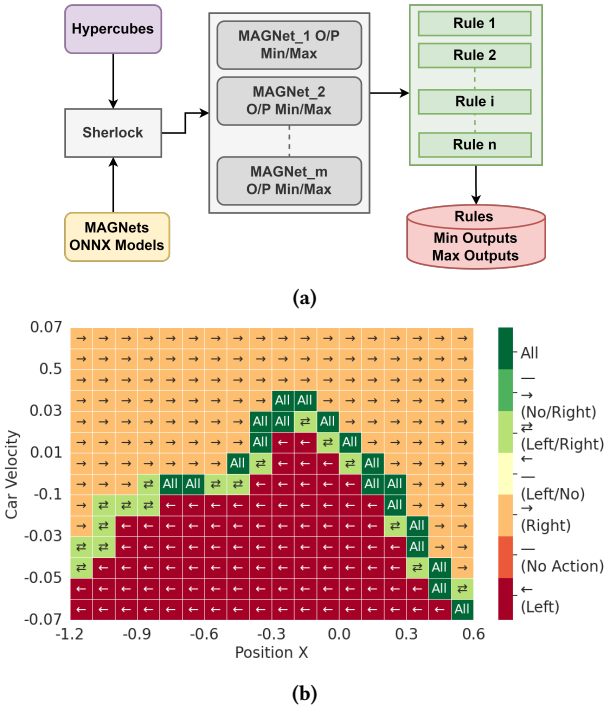

Figure 5: (a) MAGNets verification strategy using Sherlock. (b) Verification result of MountainCar-v0 MAGNets policy using Sherlock.

two actions $[a, b]$. Assume $N(x) = a$ is a deterministic *discrete* action when $v_1 \in [0, 2] \wedge v_2 \in [0, 3]$ while $N(x) = b$ when $v_1 \in [3, 5] \wedge v_2 \in [4, 8]$. However, if we consider input ranges at a coarser range, i.e., $v_1 \in [0, 5] \wedge v_2 \in [0, 8]$, the action estimate by range analysis would be nondeterministic, since $N(x) = [a, b]$. For coarser input ranges, the number of hypercubes will be smaller, requiring less time for verification, while the opposite can be said for fine-grained input ranges that lead to more precise output estimates.

For example, we consider the "MountainCar-v0" environment taken from OpenAI Gym Environments [6]. In Figure 5b, we plot the action(s) predicted by Sherlock for each hypercube of input range combinations for the "MountainCar-v0" environment's MAGNets policy. MountainCar-v0 has two observation variables, i.e., the position of the car along the x-axis (POS-X) with a domain range of (-1.2 to 0.6) and the velocity of the car (VEL) with a domain range of (-0.07 to 0.07). We consider a granularity of 0.1 for the axis POS-X and 0.01 for the axis VEL, which therefore divides the state space into (18×14) or 252 two-dimensional hypercubes (essentially rectangles in this case). The environment has three discrete actions: 'Left' (←), 'Right' (→), and 'No Action' (–); others are the combinations of such actions as shown in Figure 5b. We can observe that with the above granularity of value ranges, a few hypercubes have multiple actions. Those hypercubes can be split into multiple smaller hypercubes until each hypercube has only one predicted action. This aids in understanding whether the action

Table 1: Neurons and parameter counts of various policies for different Gym environments.

Environments	Original Policy		Pruned Policy		MAGNet Policy	
	Neurons Count	Param Count	Neurons Count	% of PPO Param used	Neurons/ MAGNet	% of PPO Param used
CartPole-v1	64	1282	64	90	4	3.43
Acrobot-v1	128	4803	128	90	16	3.23
BipedalWalker-v3	128	6020	128	70	16	20.47
LunarLander-v2	128	4996	128	55	8	6.08
LunarLanderCont-v2	256	17922	256	50	8	1.10
MountainCar-v0	128	4547	128	65	16	10.82
MountainCarCont-v0	256	17025	256	35	16	0.62
Ant-v2	128	6472	128	70	8	15.95
HalfCheetah-v2	128	5702	128	95	8	8.56
Hopper-v2	128	5123	128	75	16	15.23
Humanoid-v2	128	29393	128	75	8	43.90
InvertedDblPendlm-v2	128	4993	128	80	8	4.39
InvertedPendulum-v2	128	4545	128	35	4	0.84
Reacher-v2	128	5058	128	95	6	3.32
Swimmer-v2	128	4866	128	90	8	5.43
Walker2d-v2	128	5702	128	85	16	18.94

is safe for each hypercube. The detailed verification strategy and results for multiple environments are given in Appendix B.1.

3 EMPIRICAL STUDIES

We conduct various experiments on different OpenAI Gym [6] environments with both continuous and discrete action spaces to provide empirical support to the following claims:

- *Compaction:* Our MAGNets framework can capture the learned policies with much fewer parameters and neurons than the original policy network.
- *Interpretability:* The compaction attained leads to a simple program-based representation of the learned policies. The encoded states, as derived, have simple linear relations with output actions. The above transformations aid the interpretability of the learned policy.
- *Verification:* Each MAGNet used by our MAGNets framework is sufficiently small ($\leq$ 16 neurons) so that it can be verified using existing verification methodologies.

3.1 Baselines

We consider three baselines to compare with the MAGNets framework:

- *Original Policy Network*: We consider the original policy network as a baseline to derive a (size, performance) comparison with our MAGNets framework.
- *Static Pruned Policy Network*: In this approach, the original policy network is sparsified by replacing the weights that have very small values with zeros such that the pruned network gives approximately similar returns as the original policy network [11]. We use it as a baseline to demonstrate size comparison with our MAGNets framework for similar performance.
- *Dagger-based Cloned Policy Network*: We use Dagger-based policy cloning or imitation learning method [24] to clone the original policy network into a single MAGNet-sized neural network. Using a Dagger policy with an equivalent amount of neurons as combined MAGNets results in a network with a large number of parameters. Our motivation is to retain verifiability, assuming the system becomes unwieldy for verification beyond MAGNet-sized networks. We consider this baseline to demonstrate the performance improvements in terms of the reward of using a group of small neural networks over a single small neural network.

3.2 Evaluation Metrics

We consider the following metrics to compare our MAGNets Framework with the baselines.

- *Average Episodic Rewards*: The performance of the RL algorithms is generally represented using the cumulative reward the RL Agents accumulate in an episode. Therefore, we consider the average episodic reward over 100 test episodes a metric.
- *Neurons Count*: This metric is calculated by the total number of neurons in hidden layers of each neural network. For our MAGNets framework, we can calculate this value by multiplying the total number of MAGNets by the number of nodes present in each MAGNet.
- *Parameters Count*: This metric counts the number of weights and biases present in each neural network. We sum up the number of parameters of the MAGNets corresponding to non-zero Layer-M nodes to calculate the total number of parameters.

Among these three metrics, the first two metrics are used to evaluate the performance of the underlying RL policy. At the same time,

Table 2: Avg. Episodic Reward ± Std. Deviation of the baselines and MAGNets policy over 100 episodes and four seeds for Gym environments, and the environments' input and output size.

Environments	State Space Size	Action Space Size	Layer-M Size	MAGNets Count	Average Rewards			
					Original Policy	Pruned Policy	Dagger Policy	MAGNets Policy
CartPole-v1	4	2	2	2	500 ± 0	500 ± 0	97 ± 18	500 ± 0
Acrobot-v1	6	3	1	1	−61.6 ± 0.2	−62.2 ± 0.6	−70 ± 3	−76.8 ± 0.9
BipedalWalker-v3	24	4	4	4	294 ± 10	256 ± 14	−14 ± 74	281 ± 8
LunarLander-v2	8	4	4	4	280 ± 1	246 ± 8	247 ± 7	260 ± 8
LunarLanderCont-v2	8	2	3	3	251 ± 2	244 ± 6	−43 ± 24	231 ± 3
MountainCar-v0	2	3	5	4	−100 ± 0.7	−101 ± 0.4	−200 ± 0	−99 ± 0.6
MountainCarCont-v0	2	1	1	1	93.3 ± 0	94.1 ± 0.1	0 ± 0	89.2 ± 0.4
Ant-v2	27	8	7	6	4533 ± 29	4344 ± 53	882 ± 7	4287 ± 20
HalfCheetah-v2	17	6	4	4	4807 ± 21	4813 ± 25	−92 ± 0	4570 ± 58
Hopper-v2	11	3	4	4	1285 ± 7	1297 ± 37	837 ± 3	1303 ± 29
Humanoid-v2	376	17	9	8	803 ± 19	741 ± 11	290 ± 1	862 ± 14
InvertedDblPendlm-v2	11	1	4	3	9336 ± 46	9358 ± 0	840 ± 46	9356 ± 1
InvertedPendulum-v2	4	1	4	2	1000 ± 0	1000 ± 0	1000 ± 0	1000 ± 0
Reacher-v2	11	2	4	3	−5.5 ± 0.3	−5.7 ± 0.3	−7 ± 0.2	−5.1 ± 0.2
Swimmer-v2	8	2	4	4	132 ± 0	129 ± 0	68 ± 0	122 ± 0
Walker2d-v2	17	6	4	4	2091 ± 25	2162 ± 118	477 ± 0	1898 ± 98

the number of neurons is used to determine the verifiability of the neural network. Finally, the parameter count is used to define the compactness of the learned policy.

3.3 Experiments on Gym Environments

Table 1 shows the size of the Neural Networks in terms of the number of neurons and the number of parameters of our MAGNets framework against the baselines for various environments taken from MuJoCo, Classic Control, and Box2D groups of environments from OpenAI Gym [6]. We observe that the number of neurons for each MAGNet is very small compared to the original policy network and the pruned policy network for all the environments. For environments like "CartPole-v1" and "InvertedPendulum-v2", the number of neurons is only 6.25%, and for "LunarLanderContinuous-v2" it is only 3.13% of the original policy network. On average, the neuron count of each MAGNet varies between four to sixteen (for comparable performance) or "7.7%" of original policy networks. Similar trends can be seen in the case of parameter count. Each MAGNet has a substantially lesser number of parameters than the original policy network, with around "2.66%", and "Humanoid-v2," which has the highest "11%" (due to its large input space size). However, when all the MAGNets of the Rule List are considered, then it averages around "11.16%" with "Humanoid-v2" having the highest percentage of parameters "43.9%" (due to its large input space size) and "MountainCarContinuous-v0" having the lowest percentage of parameters "0.62%" compared to the respective original policy network.

In Table 2, we give each environment's state space and action space size along with Layer-M size and MAGNets counts. During MAGNets training, we do not train any MAGNet for the zero neurons in Layer-M. Therefore, with the column heading 'MAGNets count,' we mean the number of zero neurons in Layer-M. Table 2 shows the performance in terms of Average Episodic Reward (over 100 episodes) ± Standard Deviation over four different seeds for the same Gym environments of Table 1. Our MAGNets framework gives similar rewards as the original PPO policies (with ±5%) for most of the environments. For a few environments, it goes up to $\pm(7\ to\ 9)\%$ except for the "Acrobot-v1" environment. In the "Acrobot-v1" environment, the MAGNets framework gives "25%" less reward than the original PPO and pruned policies. Our MAGNets Policy also works better regarding rewards than the Dagger-based cloned policy for most of the Gym environments except the "Acrobot-v1" environment, where the Dagger-based clone policy achieves better rewards, but the difference is not too big. For the "InvertedPendulum-v2" environment, both collect the same rewards. However, the MAGNets policy achieves much better rewards for the rest of the environments than the Dagger-based cloned policy. The results from Table 1 and Table 2 give the empirical evidence of our first claim, "Compaction." Another important observation from Table 2 is the number of MAGNets required (MAGNets Count column) for MAGNets policy is about the same as the Action Space Size for all the environments except for the "Humanoid-v2" environment. For this environment, the action space size is large (seventeen), but the number of required MAGNets counts (eight and four, respectively) is much smaller.

Sometimes, we consider a slightly larger Layer-M size than the action space size to perform similarly to the original PPO network. In the case of InvertedPendulum-v2, a MAGNet-sized network trained using DAGGER achieves similar performance as the MAGNet framework with two MAGNets. However, in some of those cases, the MAGNets policy performs better than the similar sized logits-based

Table 3: PyTorch Profiling. Times in μs and memories in Kb.

Env Name	PPO Time (CPU+GPU)	MAGNets Time (CPU+GPU)	PPO Memory	MAGNets Memory
MountainCar Continuous-v0	(906+1191)	(120+118)	2017.28	109.38
LunarLander Continuous-v2	(993+1325)	(189+316)	2037.76	230.47

Table 4: Varying Layer-M Size while MAGNets Size is Fixed

Env Name	Layer-M Size	Magnet Size	Reward PPO	Reward MAGNets
Ant-v2	4	4x4	4482.52	2286.3
	10	4x4	4482.52	3966.27
	14	4x4	4482.52	**4636.13**
Hopper-v2	2	8x8	1286	199.46
	3	8x8	1286	274
	4	8x8	12864	**1273.8**
HalfCheetah-v2	1	4x4	4876.75	825.03
	2	4x4	4876.75	4419.72
	4	4x4	4876.75	**4614.9**

policy. For example, in the case of InvertedDblPendlm-v2, the MAGNet policy (with three MAGNet) achieves far better performance than a MAGNet-sized network trained using DAGGER. This justifies using "Layer-M" instead of the output values in our framework.

In Table 3, we shown statistics from the PyTorch profiler for both PPO and MAGNets executed in our server. From the results, it is clear that the MAGNets-based policies have a far better advantage in execution time and a much lesser memory footprint (10-20 times faster).

3.4 Varying Layer-M Size, Fixed MAGNets Size

We have conducted a few experiments to test the sensitivity of our framework for the Layer-M Size hyperparameter. In Table 4 we provide performance comparison between PPO Policy and MAGNets Policy while varying the size of Layer-M, keeping the size of individual MAGNets fixed. The fixed MAGNet size is provided in column 3 of Table 4. As the Layer-M size grows, the MAGNets policy learns more and more rewards for "Ant-v2" and "Hopper-v2", but the gains get plateaued for "HalfCheetah-v2".

3.5 Varying MAGNets Size, Fixed Layer-M Size

We have also conducted a few experiments to test the sensitivity of our framework for the MAGNets size hyperparameter. In Table 5 we vary the size of MAGNets while keeping the Layer-M size fixed and provide performance comparison between PPO Policy and MAGNets Policy. It is evident (from the rewards in bold-face) that with suitable choice of MAGNets size performance similar to PPO policy can be obtained.

In general, Table 4 and Table 5 also demonstrates the trade-off between the choice of network dimensions and rewards obtained. PPO-like rewards are attained with larger Layer-M or larger MAGNets size, even though the networks are still much smaller in comparison with PPO. Choosing the suitable Layer-M and MAGNets

Table 5: Effect of varying MAGNets Size while Layer-M Fixed

Env Name	Layer-M Size	Magnet Size	Reward PPO	Reward MAGNets
Acrobot-v1	1	6x6	-61.74	-78.86
	1	8x8	-61.74	-76.82
	1	10x10	-61.74	**-70.34**
BipedalWalker-v3	4	4x4	298.85	222.04
	4	6x6	298.85	276.23
	4	8x8	298.85	**280**
LunarLander-v2	4	2x2	276.14	170.59
	4	3x3	276.14	180.84
	4	4x4	276.14	**260**

sizes requires tuning to achieve satisfactory rewards on a target platform. This is enabled by the proposed method.

3.6 Edge Devices Compatibility

Given the portability advantage of MAGNet policies, we gather execution statistics for the same using Edge platforms like Arduino Uno R3 and Nvidia Jetson Nano. For such cases, Table 6 shows the average time and memory taken by the Original PPO Policy and MAGNets Policy for each iteration during testing in "LunarLanderContinuous-v2" and "MountainCarContinuous-v0" environments. Due to memory constraints, PPO policies for both environments failed to run in Arduino Uno. This shows that the MAGNets-based policies are advantageous in terms of both execution time (100 times faster) and memory footprint. Hence, our policy networks are suitable for low-power lightweight platforms that are useful in various Cyber Physical Systems (CPS) applications.

3.7 Rule List Representation

As discussed in section 2.4, the policy outputs can be represented using a simple set of rules. For example, the rules of the "MountainCar-v0" environment are shown below.

$$\begin{aligned} r1 =& 0 + (-1.261 \times M_2) + (-1.258 \times M_3) + \\ & (1.243 \times M_4) + (-1.206 \times M_5) + 0.524 \\ r2 =& 0 + (-0.650 \times M_2) + (-0.575 \times M_3) + \\ & (-0.525 \times M_4) + (0.283 \times M_5) + 0.060 \\ r3 =& 0 + (1.290 \times M_2) + (1.010 \times M_3) + \\ & (-0.925 \times M_4) + (0.923 \times M_5) - 0.654 \end{aligned}$$

Each M_i is a relatively simple NN-based mapping from the environment state $\vec{x}$ to encoded state Ξ_i. In this example, M_1 happens to be a zero neuron; hence, the respective term is omitted. Each rule $(r_1,\ r_2,\ r_3)$ returns the logit values of the corresponding action $(a_0,\ a_1,\ a_2)$. Since "MountainCar-v0" is a discrete-action environment, we consider the $argmax(r1,\ r2,\ r3)$ value as the action of the MAGNets policy. In general, one can thus argue that the Rule List representation can be easily expanded into a conventional program block, as Appendix A.1 shows. This feature makes MAGNets policies interpretable, similar to program representations synthesized for NNs in [22, 30].

Table 6: Average Time and Memory taken by Original PPO Policy and MAGNets Policy for each iteration during testing in LunarLanderContinuous and MountainCarContinuous environments.

Environment	Policy	Edge Device	Arduino Uno R3		Jetson Nano
		Iterations	Average Time (ms)	Memory (bytes)	Average Time(ms)
LunarLander Continuous	PPO Policy	100	NA	NA	337
		1000	NA	NA	335
		10000	NA	NA	351
	MAGNets Policy	100	3.07	751	4
		1000	3.06	751	5
		10000	3.06	751	4
MountainCar Continuous	PPO Policy	100	NA	NA	333
		1000	NA	NA	332
		10000	NA	NA	330
	MAGNets Policy	100	0.96	1327	2
		1000	0.94	1327	2
		10000	0.95	1327	2

4 RELATED WORKS

DNN Pruning and Knowledge Distillation: Pruning [9] and knowledge distillation methods [2], [12] are used to compress a trained DNN. In contrast to our approach, this method only focuses on reducing the parameters but does not consider reducing the neurons. The knowledge distillation-based method in [2] trains a shallow neural network that mimics the original deep neural network behaviors. However, the number of neurons ranges from 8k to 400k. The authors in [12] propose to train multiple neural networks or an ensemble of models and then distill the knowledge of the ensemble of models into a single compact model. This helps to achieve ensemble-like accuracy but with a compact single model. Compared to our method, we distilled the knowledge of a bigger neural network into multiple compact neural networks and then considered all the neural networks together to estimate the output.
Imitation Learning: Behavior Cloning/Imitation Learning [26] based methods are used in RL to reduce the sample complexity via expert advice to achieve the intended behavior. The dataset aggregation or Dagger [24] algorithm proposed an iterative approach that samples new trajectories based on a weighted average of expert policy and target policy. However, it uses expert policy to determine the actions of the states present in the trajectories. These new trajectories and actions aggregated with the previous trajectories are used to update the target policy. Compared to this, in our framework, rather than directly mimicking the expert policy using a single neural network, we consider learning the intermediate representation of an input state by the expert policy using multiple compact neural networks.
DNN Verification: Currently, several tools customized for DNN verification are available. One such pioneer tool is Reluplex [15], an SMT solver customized for verifying DNNs with only ReLU activation function for all neurons. The authors in [1] use Marabou [16], another SMT-based DNN verifier in their whiRL 2.0 algorithm for DNN verification. On the other hand, [8] uses Mixed Integer Linear Programming (MILP) to calculate the output range of a DNN for a given input range. The authors in [14] [13] propose the Verisig tool that uses a hybrid system for safety properties for closed-loop systems that use DNN as a controller.

All the above tools suffer from scalability with the growing size (in terms of neurons) of the neural networks. However, MAGNets use small neural networks with ReLU as activation functions; therefore, these tools can be effectively used for verification. Also, as a MAGNets policy can be converted into a conventional program, program verification tools [17] can be used for verification.

5 CONCLUSION

We evaluate our framework on different Gym environments and present a comparative study with the original policy network, randomly pruned policy network, and Dagger-based cloned policy. We show that the MAGNets framework achieves similar performance with fewer parameters. In summary, our framework provides the following advantages.

- *Convert policies suitable for edge devices*: A large DRL policy has many interconnected dependencies that impede data parallel high-performance computation. On the other hand, as each MAGNet is lightweight and independent, they can be executed in parallel on embedded GPGPU platforms while avoiding synchronization barrier overheads, thus providing energy, latency, and memory resource benefits.
- *Build verifiable and interpretable safety-critical policies*: Empirically, we establish MAGNets as a collection of Lightweight NNs that approximate the original policy network, a property that makes them verifiable and interpretable. Moreover, this structural decomposition can aid in determining the regions where the learned policy fails to ensure safety.

This work opens two important future avenues. The MAGNet size and state space granularity for verification can be treated as hyperparameters and tuned using optimization frameworks. The structure of MAGNets can be iteratively refined based on the verification results.

REFERENCES

[1] Guy Amir, Michael Schapira, and Guy Katz. 2021. Towards Scalable Verification of Deep Reinforcement Learning. *2021 Formal Methods in Computer Aided Design (FMCAD)* (2021), 193–203.

[2] Jimmy Ba and Rich Caruana. 2014. Do Deep Nets Really Need to be Deep?. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2014/file/ea8fcd92d59581717e06eb187f10666d-Paper.pdf

[3] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. 2018. Verifiable Reinforcement Learning via Policy Extraction. In *Neural Information Processing Systems*.

[4] Andrew Bell, Ian Solano-Kamaiko, Oded Nov, and Julia Stoyanovich. 2022. It's Just Not That Simple: An Empirical Study of the Accuracy-Explainability Trade-off in Machine Learning for Public Policy. In *2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul, Republic of Korea) *(FAccT '22)*. Association for Computing Machinery, New York, NY, USA, 248–266. https://doi.org/10.1145/3531146.3533090

[5] Yoshua Bengio et al. 2009. Learning deep architectures for AI. *Foundations and trends® in Machine Learning* 2, 1 (2009), 1–127.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *ArXiv* abs/1606.01540 (2016). https://api.semanticscholar.org/CorpusID:16099293

[7] Rudy Bunel, Matthew J. Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. 2018. Leveraging Grammar and Reinforcement Learning for Neural Program Synthesis. *ArXiv* abs/1805.04276 (2018).

[8] Souradeep Dutta, Xin Chen, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2019. Sherlock - A Tool for Verification of Neural Network Feedback Systems: Demo Abstract. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control* (Montreal, Quebec, Canada) *(HSCC '19)*. Association for Computing Machinery, New York, NY, USA, 262–263. https://doi.org/10.1145/3302504.3313351

[9] Jonathan Frankle and Michael Carbin. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=rJl-b3RcF7

[10] Abhiroop Ghosh, Yashesh Dhebar, Ritam Guha, Kalyanmoy Deb, Subramanya Nageshrao, Ling Zhu, Eric Tseng, and Dimitar Filev. 2021. Interpretable AI Agent Through Nonlinear Decision Trees for Lane Change Problem. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. 01–08. https://doi.org/10.1109/SSCI50451.2021.9659552

[11] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *arXiv: Computer Vision and Pattern Recognition* (2015).

[12] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *ArXiv* abs/1503.02531 (2015).

[13] Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. 2019. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 169–178.

[14] Radoslav Ivanov, James Weimer, Oleg Sokolsky, and Insup Lee. 2018. Demo: verisig - verifying safety properties of hybrid systems with neural network controllers. *Proceedings of the Workshop on Design Automation for CPS and IoT* (2018).

[15] Guy Katz, Clark W. Barrett, David L. Dill, Kyle D. Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *International Conference on Computer Aided Verification*.

[16] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Computer Aided Verification*, Isil Dillig and Serdar Tasiran (Eds.). Springer International Publishing, Cham, 443–452.

[17] James C. King. 1976. Symbolic Execution and Program Testing. *Commun. ACM* 19, 7 (jul 1976), 385–394. https://doi.org/10.1145/360248.360252

[18] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. 2022. Deep Reinforcement Learning for Autonomous Driving: A Survey. *IEEE Transactions on Intelligent Transportation Systems* 23, 6 (2022), 4909–4926. https://doi.org/10.1109/TITS.2021.3054625

[19] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. 2018. Reinforcement Learning for UAV Attitude Control. *ACM Transactions on Cyber-Physical Systems* 3 (2018), 1 – 21.

[20] Stephanie Milani, Zhicheng Zhang, Nicholay Topin, Zheyuan Ryan Shi, Charles Kamhoua, Evangelos E Papalexakis, and Fei Fang. 2023. MAVIPER: Learning Decision Tree Policies for Interpretable Multi-Agent Reinforcement Learning. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2022, Grenoble, France, September 19–23, 2022, Proceedings, Part IV*. Springer, 251–266.

[21] Ananya Paul, Krishnendu Bera, Devtanu Misra, Sattwik Barua, Saurabh Singh, Nishu Nishant Kumar, and Sulata Mitra. 2021. Intelligent Traffic Signal Management Using DRL for a Real-Time Road Network in ITS. In *2021 Thirteenth International Conference on Contemporary Computing (IC3-2021)* (Noida, India) *(IC3 '21)*. Association for Computing Machinery, New York, NY, USA, 417–425. https://doi.org/10.1145/3474124.3474187

[22] Wenjie Qiu and He Zhu. 2022. Programmatic Reinforcement Learning without Oracles. In *International Conference on Learning Representations*.

[23] Partha Pratim Ray. 2022. A review on TinyML: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences* 34, 4 (2022), 1595–1623. https://doi.org/10.1016/j.jksuci.2021.11.019

[24] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. 2010. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *International Conference on Artificial Intelligence and Statistics*.

[25] Iqbal H Sarker. 2021. Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science* 2, 6 (2021), 420.

[26] Stefan Schaal. 1999. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences* 3, 6 (1999), 233–242.

[27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *ArXiv* abs/1707.06347 (2017).

[28] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[29] Filip Svoboda, David Nunes, Milad Alizadeh, Russel Daries, Rui Luo, Akhil Mathur, Sourav Bhattacharya, Jorge Sa Silva, and Nicholas Donald Lane. 2021. Resource Efficient Deep Reinforcement Learning for Acutely Constrained Tiny{ML} Devices. In *Research Symposium on Tiny Machine Learning*. https://openreview.net/forum?id=_vo8DFo9iuB

[30] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. 2018. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*. PMLR, 5045–5054.

[31] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (01 May 1992), 279–292. https://doi.org/10.1007/BF00992698

[32] Zidong Zhang, Dongxia Zhang, and Robert C. Qiu. 2020. Deep reinforcement learning for power system applications: An overview. *CSEE Journal of Power and Energy Systems* 6, 1 (2020), 213–225. https://doi.org/10.17775/CSEEJPES.2019.00920