NEAT for Large-Scale Reinforcement Learning through Evolutionary Feature Learning and Policy Gradient Search

Yiming Peng, Gang Chen, Harman Singh, and Mengjie Zhang
Victoria University of Wellington
Wellington, New Zealand
yiming.peng@ecs.vuw.ac.nz;aaron.chen@ecs.vuw.ac.nz

ABSTRACT

NeuroEvolution of Augmenting Topology (NEAT) is one of the most successful algorithms for solving traditional reinforcement learning (RL) tasks such as pole-balancing. However, the algorithm faces serious challenges while tackling problems with large state spaces, particularly the Atari game playing tasks. This is due to the major flaw that NEAT aims at evolving a single neural network (NN) that must be able to simultaneously extract high-level state features and select action outputs. However such complicated NNs cannot be easily evolved directly through NEAT. To address this issue, we propose a new reinforcement learning scheme based on NEAT with two key technical advancements: (1) a new three-stage learning scheme is introduced to clearly separate feature learning and policy learning to allow effective knowledge sharing and learning across multiple agents; (2) various policy gradient search algorithms can be seamlessly integrated with NEAT for training policy networks with deep structures to achieve effective and sample efficient RL. Experiments on several Atari games confirm that our new learning scheme can be more effective and has higher sample efficiency than NEAT and three state-of-the-art algorithms from the most recent RL literature.

CCS CONCEPTS

• **Computing methodologies → Reinforcement learning**; **Neural networks**;

KEYWORDS

Reinforcement Learning, NeuroEvolution, NEAT, Feature Learning, Policy Learning, Policy Gradient Search

ACM Reference Format:
Yiming Peng, Gang Chen, Harman Singh, and Mengjie Zhang. 2018. NEAT for Large-Scale Reinforcement Learning through Evolutionary Feature Learning and Policy Gradient Search. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan,* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205536

1 INTRODUCTION

As an important machine learning technology that improves the decision making ability of intelligent agent via learning from direct interactions between itself and an unknown environment, Reinforcement Learning (RL) has attracted remarkable attentions from both academia and industry. It has great application prospects in domains such as robotics, psychology, cognitive science, economics, neuroscience, and human-level intelligent game play [6, 14–17, 24].

NeuroEvolution of Augmenting Topology (NEAT) is an algorithm capable of evolving both weights and topologies of Neural Networks (NNs) and has already been proven to be a successful RL technology for classic control tasks, such as pole-balancing [22]. The advantages of using NEAT for RL can be summarized as follows: 1) NEAT can search directly in the policy space (i.e., the solution space) without requiring any indirect inferences from value functions, and 2) NEAT can work effectively on problems where gradients do not exist or are hard to calculate; and 3) NEAT can also mitigate the potential influence of initial network topology configurations on final learning performance since it has the tendency to evolve more useful topologies.

However, when applied to solving large-scale RL problems, NEAT must evolve highly sophisticated NNs from the initial simplest structure with no hidden neurons and connections. Apparently, this task is extremely difficult and sample inefficient [20].

Although huge research efforts have been made to improve the capability of NEAT for solving problems with large state spaces, existing technologies still have room for improvement. For example, HyperNEAT uses an indirect encoding method to evolve large NNs but was found to perform poorly on problems with high-level and geometrically unrelated features [10].

As far as we know, a majority of existing NeuroEvolution methods focused primarily on evolving policy networks directly from raw state inputs and therefore share the same limitations as NEAT and NEAT+Q. To address this important issue, a recent approach has made a first attempt to split feature learning from policy learning, resulting in a new algorithm called NEAT+AC [16]. NEAT+AC evolves a population of learning agents, each contains both a feature network and a separate policy network which transforms high-level state features produced by the feature network linearly to action outputs. In NEAT+AC, NEAT is used to evolve the feature network only. In comparison to a network that maps raw state inputs directly to actions, such feature networks are expected to have much simpler structures.

Despite of clear performance advantages of NEAT+AC on classic control problems, NEAT+AC still has limited effectiveness for large-scale RL. Specifically, NEAT+AC relies on a traditional Regular Actor-Critic (RAC) RL algorithm [3] which often fails to learn reliably with non-linear and more powerful policy networks. Moreover, feature learning and policy learning are heavily mingled in a single process, preventing easy sharing of learned knowledge (e.g. policy networks) across multiple agents.

Inspired by our understanding of NEAT and its variations, this paper aims to propose a new learning scheme that presents two main advantages over NEAT and NEAT+AC: 1) in the new learning scheme, feature learning is clearly separated from policy learning to allow effective knowledge sharing and learning among a population of agents, and 2) policy improvement can be achieved by adopting various cutting-edge algorithms for policy gradient search (PGS), making the new learning scheme particularly suitable for training policy networks with deep structures.

To achieve our research goal, the learning scheme is designed to include three separate stages. In the first stage, an interim policy network is pre-trained to provide an appropriate foundation for feature learning. In the second stage, the trained policy network is shared over all learning agents to enhance knowledge reuse. Isolated completely from policy learning, NEAT is further utilized in this stage by every agent to evolve useful feature networks, the best of which will be subsequently adopted in the last learning stage. Finally, with the help of a state-of-the-art PGS algorithm (e.g. A2C [7] and Trust Region Policy Optimization [17]), the policy network is trained again in line with the chosen feature network in the third stage.

Since our new learning scheme is characterized by its seamless integration of NEAT and cutting-edge PGS algorithms, it will be referred to as NEAT+PGS in the remaining of this paper. In comparison to NEAT+AC and other NEAT based RL approaches, NEAT+PGS is highly competent at tackling large-scale learning problems such as Atari game playing tasks. Particularly, with strong feature learning capability, the dimensionality of feature inputs to policy networks can be substantially reduced in NEAT+PGS, further facilitating effective and sample efficient policy training, as confirmed by our in-depth sample efficiency analysis.

Goals. The overall goal of this paper is to develop a new learning scheme, called NEAT+PGS, that seamlessly integrates NEAT based feature learning and PGS, to address challenging RL problems with large state spaces. Here, we aim to achieve the following objectives through the development of NEAT+PGS.

(1) To design a three-stage learning scheme for accommodating NEAT based automated feature learning and PGS based effective policy learning.

(2) To evaluate the effectiveness of NEAT+PGS in comparison to NEAT and three state-of-the-art algorithms on large-scale problems.

(3) To analyze the sample efficiency of NEAT+PGS algorithms against NEAT and state-of-the-art PGS algorithms.

2 REINFORCEMENT LEARNING AND RELATED WORK

This section starts with the conceptual context of RL, followed by a description on NEAT algorithm for RL.

2.1 Reinforcement Learning

RL is a learning paradigm where an agent learns from interactions between an unknown environment and itself [24]. The interaction can be described as follows. The agent observes a state $\vec{s}_t$ and takes an action a_t at time step t. Subsequently it makes an transition and observes a new state $\vec{s}_{t+1}$, meanwhile it receives a numerical reward feedback r_t from the environment. Such an observed state transition is treated as a sample. Often, RL is modeled as a Markov Decision Process (MDP) containing a state space $\mathcal{S}$, a discrete action space $\mathcal{A}$, a transition probability function $\mathcal{P}$, a reward function $\mathcal{R}$ and a discount factor γ.

RL aims to seek an optimal policy that can maximize long-term cumulative reward in an infinite horizon. Similar to many existing works, this paper particularly considers the stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Accordingly, the long-term cumulative reward, which an agent can achieve while following any policy π, is defined as

$$J^{\pi} = V^{\pi}(\vec{s}) = \mathrm{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t | \vec{s}_0 = \vec{s}], \tag{1}$$

where V^{π} is called state-value function. It represents the expected cumulative rewards starting from any state $\vec{s}_0$ by following the policy π. Alternatively, we can define the action-value function Q^{π} to represent the expected cumulative rewards of taking action a in state $\vec{s}$, i.e.,

$$Q^{\pi}(\vec{s}, a) = \mathrm{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t | \vec{s}_0 = \vec{s}, a_0 = a]. \tag{2}$$

Accordingly, we can established the goal of RL which is to learn the optimal policy, i.e.,

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, V^{\pi}(\vec{s}). \tag{3}$$

2.2 NeuroEvolution of Augmenting Topology

NEAT is an evolutionary approach towards learning flexible NN-based solutions for various machine learning problems including RL [22]. It has gained prominent successes on classic RL tasks such as double pole balancing [21] and robotic control [23]. In comparison to other Neural Evolution methods, NEAT possesses the following technical strengths:

(1) *Topology evolution*: NEAT starts with a population of simplest networks where no hidden neurons or connections are given. The topology is incrementally augmented via two mutation operators, adding nodes and adding links. In this way, NEAT tends to find an NN with a minimum number of weights and a suitably complex topology for a given problem.

(2) *Innovation number*: NEAT encodes its genome (i.e., a NN) in a direct way where each genome is defined by a group of connection genes. Each connection gene is specified by one in-node gene, one out-node gene, the weight of the connection, an enable indicator that determines whether the connection gene is expressed, and an innovation number that helps identify corresponding genes for crossover. By using the simple innovation number, the crossover operation can be straightforwardly performed without expensive comparisons on topological similarities.

(3) *Speciation*: NEAT also speciates its population so that an individual mainly competes within its own speciation rather than within the entire population. In doing so, NEAT can prevent any topological innovations generated by individuals within their own niches from overriding other niches of the population.

Thanks to the above algorithmic designs, NEAT can be particularly effective for feature learning. Firstly, owing to its capability of evolving NN topologies from the most primitive forms, NEAT can maintain structural simplicity that help reduce output variances [1]. Secondly, by simply avoiding the connections between some nodes, NEAT can easily filter those dimensions of the state space that are not useful for decision making [28].

2.3 Related Work

The literature shows us plenty of endeavors on enhancing the ability of NEAT for scalable RL. Here, we will discuss several typical examples that can be considered as either new variations of NEAT or hybrid methods involving NEAT.

One well-known variation of NEAT is HyperNEAT [20] which was designed to evolve large NNs indirectly by exploiting inherent geometric properties of the learning problems. Unfortunately, extensive investigations of HyperNEAT suggest that this method may perform much worse than NEAT on many large-scale problems including Atari games, especially in the presence of high-level features [10]. In addition, NEAT was found vulnerable to the fracture issue, i.e. the mapping from states to optimal actions is highly discontinuous [12]. The same issue is proven to be even more challenging for HyperNEAT in [26].

Frustrated by the inherent limitations of NEAT, researchers started to consider enhancing NEAT by combining it with mainstream gradient-based RL methods, resulting in several hybrid approaches. One typical example is NEAT+Q [28] which aims to improve NEAT with the aid of Q-learning. However, its ability to evolve sufficiently complicated Q-networks, as demanded by difficult RL problems, remains questionable. Another recent development was reported in [16] where the NEAT+AC algorithm was introduced. On one hand, NEAT+AC clearly demonstrates the importance of separating feature learning from policy learning on low-dimension control tasks. On the other hand, as mentioned in the introduction, the algorithm may face huge difficulty with large state spaces. This is due to the fact that NEAT+AC uses simple Regular Actor-Critic (RAC) RL algorithm that cannot train deep policy networks effectively as well as the constraint for all agents to learn independently which forbids constructive knowledge sharing.

Inspired by NEAT+AC, in particular its use of NEAT for evolving feature networks, we will develop NEAT+PGS as a general learning scheme in this paper. NEAT+PGS improves both NEAT and NEAT+AC by completely separating feature learning from policy learning to promote direct sharing of trained policy networks among all learning agents. It also facilitates the use of state-of-the-art PGS algorithms in a more sample efficient manner. HyperNEAT could be possibly used to evolve feature networks too. However it is less useful on the benchmark problems studied in this paper, i.e. RAM-based Atari games where inputs are a sequence of 128-bit integers with no geometrical relationships [10].

3 POLICY GRADIENT SEARCH FOR REINFORCEMENT LEARNING

Policy Gradient Search (PGS) is a family of important RL algorithms where policy updating follows the direction of gradients estimated by the long-term cumulative rewards with respect to policy parameters [3, 11, 16, 17, 25] as shown below,

$$\Delta\vec{\theta} \propto \nabla_{\vec{\theta}} J(\vec{\theta}). \tag{4}$$

Note that, the gradient $\nabla_{\vec{\theta}} J(\vec{\theta})$, which is hard to directly compute in reality, is often estimated via different means which give rise to a blossom of various PGS algorithms. Though our NEAT+PGS learning scheme can support majority of PGS methods in theory, in this paper we decide to consider several most popularly studied algorithms.

3.1 Actor-Critic Policy Gradient Search

Actor-Critic Policy Gradient Search (AC-PGS) algorithms [6] maintain a parametric policy $\pi^{\vec{\theta}}(a|\vec{s})$ and an estimated parametric value function $V^{\vec{\omega}}(\vec{s})$. The policy parameters and value function parameters are updated by following the gradient estimation shown as follows,

$$\nabla_{\vec{\theta}} J(\vec{\theta}) = \mathrm{E}_t[\nabla \log \pi^{\vec{\theta}}(a_t|\vec{s}_t) A_t^{\vec{\theta},\vec{\omega}}(\vec{s}_t, a_t)], \tag{5}$$

where $A_t^{\vec{\theta},\vec{\omega}}(\vec{s}_t, a_t)] = r_{t+1} + r_t V^{\vec{\omega}}(\vec{s}_{t+1}) - V^{\vec{\omega}}(\vec{s}_t)$ is an estimation of the advantage function at timestep t. Here, the expectation E_t is approximated through averaging over a batch of samples (e.g., a single sample or a complete trajectory).

One of the most recent algorithm in called Advantage Actor-Critic algorithm called A2C [7], which has been chosen in this paper because of its competitive performance on playing Atari games reported in [19]. In addition, as reported in [7], A2C has shown proven better performance in comparison to its asynchronous version, i.e, A3C [14], when being used in single CPU environment.

3.2 Expectation Maximization based Policy Gradient Search

The key idea of Expectation Maximization based Policy Gradient Search (EM-PGS) is to construct lower bound for the policy, and instead of maximizing J in (1), the lower bound is expected to be optimized during learning [6]. In line with this idea, EM-PGS derives a lower bound shown below,

$$\log J(\vec{\theta}) \geq \mathcal{L}^{\vec{\theta}}(\vec{\theta}') = \mathrm{E}_{p^{\vec{\theta}'}(\tau)}[R(\tau) \log p^{\vec{\theta}}(\tau)], \tag{6}$$

where τ represents any trajectory, $\vec{\theta}'$ is the old policy parameter (i.e., sampling policy) whereas $\vec{\theta}'$ is the new policy parameter, $p^{\vec{\theta}}(\tau)$ and $p^{\vec{\theta}'}(\tau)$ are the probabilities of generating the trajectory τ following the new policy $\pi^{\vec{\theta}}$ and $\pi^{\vec{\theta}'}$ respectively.

Based on (6), if $\vec{\theta}'$ and $\vec{\theta}$ are close in the limit, we can have the policy gradient estimation by differentiating the lower bound with respect to the policy parameters $\vec{\theta}'$, i.e.,

$$\nabla_{\vec{\theta}} J(\vec{\theta}) = \lim_{\vec{\theta} \to \vec{\theta}'} \nabla_{\vec{\theta}} \mathcal{L}^{\vec{\theta}}(\vec{\theta}') = \mathrm{E}_{p^{\vec{\theta}'}}[\sum_{t=0}^{T-1} Q_t^{\pi}(\vec{s}_t, a_t) \nabla_{\vec{\theta}} \log \pi^{\vec{\theta}}(\vec{s}_t, a_t)] \tag{7}$$

In this paper, we consider a typical EM-PGS algorithm called Policy learning by weighting exploration with the returns (POWER) [11], because it has shown proven effectiveness on many practical applications like robotic controls. However, POWER cannot train NNs directly, thus instead of using POWER, we are actually using a

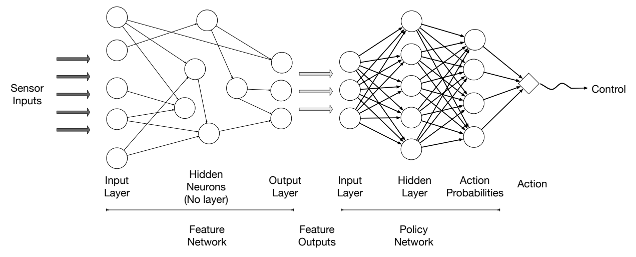


Figure 1: An Overview on the NN Architecture of NEAT+PGS

gradient variation of POWER. This adaptation uses a new formulation never presented in the literature, we experimentally found that it can achieve better performance compared to original POWER. For expression simplicity, we still use the abbreviation "POWER" representing the actual "POWER-gradient-variant".

3.3 Trust Region Policy Optimization

Similar to EM-PGS, Trust Region Policy Optimization (TRPO) [17] is to approximate a lower bound on the performance improvement via a surrogate objective function $\mathcal{L}^{\vec{\theta}'}(\vec{\theta})$, and then maximize the lower bound to achieve performance improvements. The surrogate objective function is given below,

$$\begin{aligned} &\text{maximize}_{\vec{\theta}}[\mathcal{L}^{\vec{\theta}'}(\vec{\theta})] \\ &\text{subject to } \mathrm{D}_{KL}(\pi^{\vec{\theta}'}||\pi^{\vec{\theta}}) \leq \delta \end{aligned} \tag{8}$$

where

$$\mathcal{L}^{\vec{\theta}'}(\vec{\theta}) = \eta(\vec{\theta}') + \mathrm{E}_{\rho^{\vec{\theta}}(\vec{s})}\mathrm{E}_{a\sim\pi^{\vec{s}}}A_{\vec{\theta}'}(\vec{s},a), \tag{9}$$

with $\mathrm{D}_{KL}(\pi^{\vec{\theta}'}||\pi^{\vec{\theta}})]$ is the expected KL divergence between the old policy parameter $\vec{\theta}'$ and the new policy parameter $\vec{\theta}$, $\rho^{\vec{\theta}}(\vec{s})$ is state visiting frequency which is approximated by using $\rho^{\vec{\theta}'}(\vec{s})$. Note, $\mathrm{D}_{KL}(\pi^{\vec{\theta}'}||\pi^{\vec{\theta}})]$ is also known as the Trust Region which guarantees policy parameter updating to be reasonable small (i.e., "be trusted") so as to stabilize policy learning. In addition, in order to reduce the variance of policy gradient estimation for effective RL, we also employ a recent improvement of TRPO, Generalized Advantage Estimation (GAE), proposed in [18]. TRPO is a prominent PGS approach that has been successfully applied to many difficult RL problems like playing Atari games [17].

In fact, the field of investigations on PGS has been more and more active, resulting in an explosion on appearances of new PGS algorithms such as ACER [27], ACKTR [29], PPO [19]. A comprehensive review list covering all these algorithms are thus beyond our scope, more importantly, they are fundamentally similar in practice to the ones presented here. This leaves our room for future investigation on their potential use in NEAT+PGS.

4 POLICY GRADIENT SEARCH WITH NEAT BASED FEATURE LEARNING

This section presents our new NEAT+PGS learning scheme in detail. We firstly provide an overview of the three-stage learning scheme. Next, we give detailed algorithmic descriptions of all key functions involved.

4.1 An Overview of NEAT+PGS

NEAT+PGS is proposed with the aim of achieving three design objectives: O1) feature learning and policy learning should be completely separated and supported by their own learning techniques; O2) the knowledge obtained through learning, in particular the trained policy network, must be shared across all learning agents to avoid unnecessary wastage of training samples; and O3) policy network training should be realized with the flexibility of using arbitrary PGS algorithms.

4.1.1 A Neural Network Architecture. Figure 1 depicts the NN architecture to be adopted by NEAT+PGS for RL. As shown, the architecture features a sequential combination of one feature network and one policy network. At any time, the raw observation from the learning environment (i.e. current memory status of an Atari game) is provided first as the state input to the feature network which subsequently produces a group of high-level features to be further fed into the policy network. The policy network then produces stochastic action selection decisions (i.e. the probability of selecting each optional action) that an agent can use to determine one action which will be performed finally in the learning environment, resulting in a sampled transition to a new environment state. An instant reward as described in Section 2.1 is also generated as the immediate feedback for continued RL. In NEAT+PGS, the policy network has a fixed topology which will be determined before learning starts. The feature network, on the other hand, has the flexibility for its topology to be evolved by NEAT.

4.1.2 A Three-Stage Design for Feature and Policy Learning. To meet all of our design objectives, NEAT+PGS introduces three consecutive learning stages as briefly explained below:

(1) *Interim Policy Search Stage*: This is the first learning stage with the purpose of quickly pre-training a single interim policy network. This trained policy network establishes preliminary discrimination over all alternative actions that can be performed at any environment state based on a randomly created feature network. It presents a stable starting point and guides subsequent evolution of feature networks in the second learning stage. Without pre-training the policy network, all high-level features produced by an evolved feature network cannot impose a clear preference of performing any desirable actions. The overall RL performance will be affected as a consequence.

(2) *NEAT based Feature Learning Stage*: This is the second learning stage, at the beginning of which the pre-trained interim policy network is distributed to a group of learning agents, each of which maintains a separate feature network. The population of such feature networks is further evolved through NEAT to produce eventually one feature network, which can achieve the best learning performance upon using it together with the interim policy network. The high-level features created by this feature network is hence considered suitable for tackling the RL problem.

(3) *Policy Gradient Search Stage*: This is the third learning stage. At this stage, based on the best feature network evolved by NEAT, the policy network is trained again using a state-of-the-art PGS algorithm that is expected to achieve clearly better performance than both NEAT and PGS algorithms when the entire learning process finishes.

Algorithm 1 NEAT+PGS

```
Require: an MDP $\langle \mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \gamma \rangle$, $n_1$: the number of interim policy search iterations, $n_2$: the number of feature learning iterations (generations), $n_3$: the number of policy gradient search iterations, $\vec{\theta}_0$: the randomly initial policy parameters, $p$: the population size, $P[]$: a population of NNs
    Initialization:
1:  $\vec{\theta} \leftarrow \vec{\theta}_0$
2:  $P[] \leftarrow$ INIT_POPULATION($\mathbb{S}, \mathbb{A}, p$)
    Interim Policy Search:
3:  $\phi_0 \leftarrow$ RANDOM($P[]$)
4:  $\vec{\theta} \leftarrow$ PGS($\vec{\theta}, \phi_0, n_1$)    ▷ See Algorithm 2
    NEAT based Feature Learning:
5:  $\phi^* \leftarrow$ EVOLVE($\vec{\theta}, P[], p, n_2, m_n, m_l, m_w$)    ▷ See Algorithm 3
    Policy Gradient Search:
6:  $\vec{\theta}^* \leftarrow$ PGS($\vec{\theta}, \phi^*, n_3$)    ▷ See Algorithm 2
7: return $\vec{\theta}^*, \phi^*$
```

Obviously, the three-stage design of NEAT+PGS enables complete separation of feature learning and policy learning, thereby realizing design objective O1 mentioned previously. Meanwhile, all feature networks evolved by NEAT in the second learning stage share the same goal of enhancing the effectiveness of the same interim policy network. Therefore, by breaking the independence constraint among learning agents in NEAT+AC, objective O2 is fulfilled in NEAT+PGS. Moreover, in the third learning stage, we directly treat high-level features produced by the trained feature network as the state inputs to the policy network. This approach permits arbitrary PGS algorithms to be employed for continued training of the policy network and gives our learning scheme the highest flexibility of using many cutting-edge RL technologies, as required by objective O3.

4.2 Algorithmic Description of NEAT+PGS

In this subsection, we present the high-level algorithmic description of NEAT+GPS in Algorithm 1, followed by detailed algorithmic descriptions of each learning stage in Algorithms 2, 3. In Algorithm 2, we present a general description of PGS algorithms which will be employed to train policy networks in the interim policy search stage as well as the policy gradient search stage. In Algorithm 3, we adopt the standard implementation of NEAT [22] for evolving feature networks.

5 EMPIRICAL ANALYSIS

This section provides an empirical analysis of NEAT+PGS. We first discuss the general experiment design in 5.1, including testing algorithms, benchmark problems, and experiment setups such as network topology and hyper-parameter settings. Next, we present learning performance results obtained on six benchmark problems in 5.2. Lastly, we compare the sample efficiency of NEAT+PGS, NEAT and several PGS algorithms in 5.3.

5.1 Experiment Design

Algorithm 2 Policy Gradient Search

```
Require: an MDP $\langle \mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \gamma \rangle$, $n$: the number of learning iterations, $T$: the horizon, $l$: maximum length for one trajectory, $M$: the batch size, $\mathcal{T}$: sample repository
1: function PGS($\vec{\theta}$, $\phi$, $n$)
2:     for $i = 1, 2, ..., n$ do
3:         for $t = 1, 2, ..., T/l$ do
4:             $\mathcal{T} \leftarrow$ ROLLOUT($\vec{\theta}, \phi$) ▷ Store $T$ samples into sample repository. See [11].
5:         end for
6:         Update Policy Parameters $\vec{\theta}$ based on the principle of chosen PGS algorithm with a batch of $M$ samples from $\mathcal{T}$.
7:     end for
8:     return $\vec{\theta}^*$    ▷ Return a learned policy
9: end function
```

Algorithm 3 NEAT Feature Learning

```
Require: $P$: a population of learning agents, $p$: population size, $m_n$: add node mutation rate, $m_l$: add link mutation rate, $m_w$: weight mutation rate
Ensure: $P$: a reproduced population of learning agents
1: function EVOLVE($\vec{\theta}$, $P$, $p$, $n_2$, $m_n$, $m_l$, $m_w$)
2:     for $i = 1, 2, ..., n_2$ do
3:         $P'[] \leftarrow$ new array of size $p$
4:         for $k = 1, 2, ..., p$ do
5:             $P'[k] \leftarrow$ BREED_NET($P[]$)
6:             if RANDOM() $< c$ then
7:                 $P'[k], P'[k'] \leftarrow$ SELECT($P'[]$)
8:                 $P'[k+1] \leftarrow$ CROSS_OVER($P'[k], P'[k']$)
9:             if RANDOM() $< m_n$ then ADD_NOTE($P'[k]$)
10:            if RANDOM() $< m_l$ then ADD_LINK($P'[k]$)
11:            if RANDOM() $< m_w$ then WEIGHTS_MUTATE($P'[k].\phi$)
12:            $P'[k].fitness \leftarrow$ ROLLOUT($\vec{\theta}, P'[k]$)
13:        end for
14:    end for
15:    return $P'[k^*]$    ▷ Return the fittest feature network
16: end function
```

5.1.1 Testing Algorithms. In this paper, we choose seven algorithms for evaluation. Firstly, as NEAT+PGS is an improvement to NEAT, NEAT hence is chosen as a baseline for comparison. Besides, HyperNEAT is not taken into account as its unreliable performance on RAM-based Atari Games in comparison to NEAT reported in [10]. Next, in order to examine the effectiveness of NEAT+PGS, we select three PGS algorithms, including A2C, POWER and TRPO, for policy training in NEAT+PGS, resulting in three new algorithms: NEAT+A2C, NEAT+POWER, NEAT+TRPO. Note that, NEAT+Q can be achieved by adopting our scheme of combining NEAT feature network and Q learning network. We do not include NEAT+Q in experiments, because of two reasons: 1) Deep Q learning is reported to perform poorly contrast to the above cutting-edge PGS methods in literature [7, 17]. 2) In comparison to Q learning, feature learning is more influential in PGS methods because feature network is shared in both critic and actor networks. Thus, to investigate the effectiveness of feature learning, we do not need to specifically study single critic learning like Q learning as it is a part of most PGS methods. In addition, NEAT+AC [16] is not necessary to be included in experiments, as it cannot be used to train NNs with deep structures thus not suitable for our experiments. Other than this, we use A2C which can be viewed as an enhanced version of RAC used in NEAT+AC.

In our experiments, we track learning performances in terms of average long-term cumulative rewards per episode that an algorithm can obtain after being trained on every 10,000 samples. In total, each algorithm has been trained for 10,000,000 samples. For NEAT as a population-based algorithm, we have put a counter to track the number of samples used for fitness evaluation. As long as the counter reaches a multiple of 10,000, we will select the best known NN in the current population, and its testing performance will be recorded. Regarding NEAT+PGS, we use a total of 10,000 samples in the interim policy search stage. In fact, we have experimentally found that more samples for pre-training the interim policy will not lead to better performance. Moreover, the purpose of this stage is only to let the interim policy distinguish different actions. Thus, we do not require a great number of samples here. Furthermore, in the NEAT base feature learning stage, we allow exact 2,000,000 samples be used for evolving good feature networks. Lastly, we will consume another 7,990,000 samples to train policy network in the policy gradient search stage.

5.1.2 Arcade Learning Environment. In the RL literature, ALE [2] is a set of well-known and highly challenging benchmark problems [2, 8, 10, 14, 15, 17]. Hence in this paper, we also adopt ALE to help us evaluate NEAT+PGS and other competing algorithms.

ALE provides two types of Atari Games with the key difference on state representations: one is RAM-based games where states are represented as 128-bit integers stored in memory, the other is IMAGE-based games where states are represented as video frames captured directly from the games. In this paper, we choose RAM-based games, because they are more suitable for NEAT. For IMAGE-based games, Convolution Neural Networks (CNNs) are the natural choices that are difficult to be evolved directly by NEAT. However, we are aware of some recent efforts being made to develop such NEAT based algorithms capable of evolving Deep CNNs, for example, DeepNEAT [13]. Nonetheless, only preliminary results have been obtained and further refinement of DeepNEAT is required. Thus DeepNEAT may not be usable for our study at this moment. In addition, 128 dimensions of raw state inputs are considered sufficiently large for our empirical study.

5.1.3 Experiment Setup. The NN topologies of policy and feature networks for competing algorithms are given in Table 1. The table does not specify NEAT's topology which is actually automatically evolved. In addition, the topology of feature network in NEAT+PGS is also expected to be automatically evolved by NEAT. The reason of using evolved topology other than fixed topology is that, there are few references can be found on discussion about how to determine the topology for feature networks or how much impacts different topologies would have. In fact, these questions are beyond our research scope which are not specificallyconsidered in this paper. All feature networks are configured to output 32-dimension high-level features. We have examined three opinions: 32, 64 and 128. They do not appear to have any significant impact on final performance. PGS algorithms employed a single NN with 32 by 32 hidden neurons, the first hidden layer and the second layer can be considered as the feature network and policy network respectively. Note that, for all networks, we use the same activation function "RELU" that can help reduce likelihood of vanishing gradients [9].

Network Topology	Layer	NEAT+A2C	NEAT+POWER	NEAT+TRPO	A2C	POWER	TRPO
Feature Network Structure	Input	128	128	128	128	128	128
	Hidden	Evolved by NEAT			32	32	32
	Output	32	32	32	32	32	32
Policy Network Structure	Input	32	32	32	32	32	32
	Hidden	32	32	32	32	32	32
	Output	Number of Actions					

Table 1: Topology Setups for Policy Networks and Feature Networks.

Hyper-Parameters	A2C	POWER	TRPO
policy step size (α)	2.5×10^{-6}	2.5×10^{-6}	2.5×10^{-6}
discount factor (γ)	0.99	0.99	0.99
max size of sample repository	40,000	40,000	40,000
batch size (M)	10,000	10,000	10,000
max trajectory length (l)	10,000	10,000	10,000
interim policy training iterations (n_1)	1	1	1
policy gradient search iterations (n_3)	1,000	1,000	1,000
conjugate gradient iterations	10	-	-
GAE factor (λ)	0.95	-	-
optimization epoch (ϵ)	5	5	5

Table 2: Hyper-parameter settings for PGS algorithms

HyperParameters	NEAT
population size (p)	100
max generations (n_2)	100
add node rate (m_n)	0.02
add connection rate (m_l)	0.5
weight init mean	0.0
weight init std	1.0
weight mutate rate (m_w)	0.46
weight mutate power	0.825
activation function	relu
crossover rate (c)	0.1

Table 3: Hyper-parameter settings for NEAT

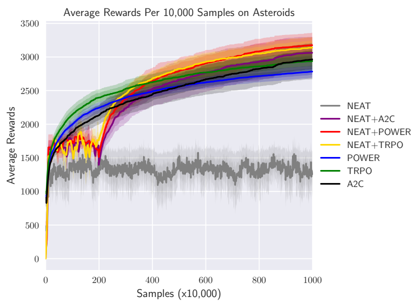
Hyper-parameter settings for all algorithms have been specified in Table 2. For all PGS algorithms, we follow the same settings in [17]. For NEAT, we follow the setting of a work published on GitHub [1] which has applied NEAT to some RAM-based Atari games with reasonably good performance.

Regarding the implementations, we adopt OpenAI-GYM [2][5] that implements ALE, NEAT-python [3] that implements NEAT, and rllab [4] [8] that implements PGS algorithms including A2C, POWER and TRPO.

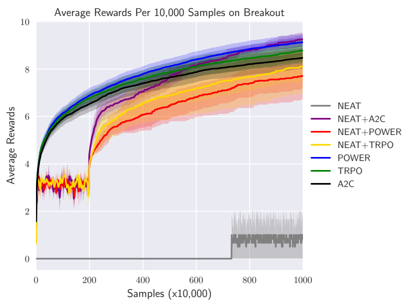
5.2 Evaluations on Learning Effectiveness

We depict learning curves for all seven algorithms on six different Atari games in Figure 2. We can see that NEAT+PGS (including NEAT+A2C, NEAT+POWER, NEAT+TRPO) is generally more effective than other competing algorithms especially NEAT on all six

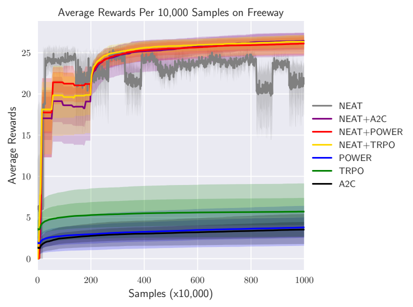
[1]https://github.com/HackerHouseYT/OpenAI-NEAT
[2]https://github.com/openai/gym
[3]https://github.com/CodeReclaimers/neat-python
[4]https://github.com/rll/rllab



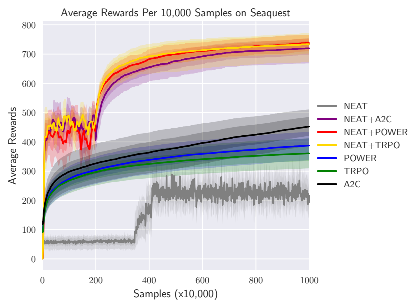
(a) Asteroids



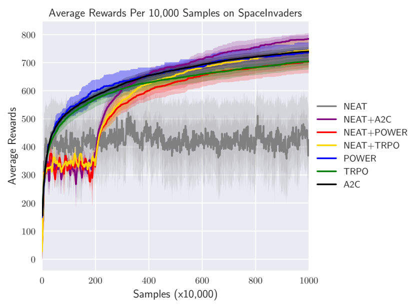
(b) Breakout



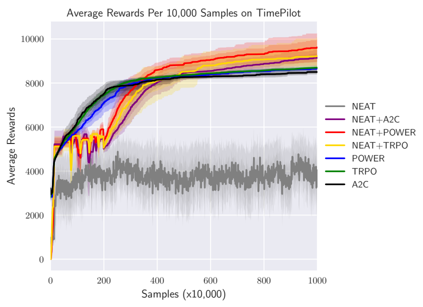
(c) Freeway



(d) Seaquest



(e) SpaceInvaders



(f) TimePilot

Figure 2: Average rewards per 10,000 samples obtained by NEAT, NEAT+A2C, NEAT+POWER, NEAT+TRPO, A2C, POWER and TRPO on six Atari games, including Asteroids, Breakout, Freeway, Seaquest, SpaceInvaders, and TimePilot. (As being highlighted with red color, for NEAT+PGS, the NEAT based feature learning stage stops at 2,000,000 samples.

Algorithm	Asteroids	Breakout	Freeway	Seaquest	SpaceInvaders	TimePilot
NEAT+A2C	246.44 ±30.05	0.69 ±0.04	**2.39 ±0.27**	**62.75 ±5.96**	**61.99 ±3.03**	756.26 ±51.60
NEAT+POWER	**257.79 ±24.42**	0.59 ±0.10	**2.42 ±0.18**	**63.95 ±4.81**	57.03 ±4.95	**800.66 ±62.90**
NEAT+TRPO	**258.38 ±20.80**	0.62 ±0.10	**2.43 ±0.22**	**64.96 ±6.21**	59.37 ±4.43	773.14 ±77.21
NEAT	131.82 ±47.85	0.02 ±0.03	2.28 ±0.25	16.33 ±5.96	42.04 ±22.62	370.95 ±205.36
A2C	246.52 ±28.37	**0.73 ±0.11**	0.30 ±0.32	37.17 ±15.80	**64.62 ±4.96**	**779.89 ±29.44**
POWER	245.75 ±21.10	**0.78 ±0.09**	0.33 ±0.55	33.63 ±13.37	**64.83 ±4.58**	778.77 ±23.58
TRPO	**259.85 ±20.80**	**0.76 ±0.10**	0.54 ±0.61	32.08 ±13.18	61.96 ±5.74	**789.64 ±26.32**

Table 4: Sample Efficiency comparison of NEAT+PGS against NEAT, A2C, POWER and TRPO.

Atari games. In particular, NEAT+PGS apparently outperforms all competing algorithms on Seaquest. On Asteroids and TimePilot, after using the trained feature networks after 2,000,000 samples, NEAT+PGS can clearly learn faster than PGS algorithms that use predefined policy networks, and eventually surpasses these algorithms.

Interestingly, we have also found that, on Freeway, the cutting-edge PGS algorithms have all failed. This result is agreeable with recent findings in [19]. On the other hand, NEAT+PGS manages to perform well and eventually surpass NEAT. This evidence further shows the importance of using NEAT for feature learning. In addition, NEAT+PGS algorithms performed competitively as PGS algorithms on Breakout. For NEAT+PGS, we found this is because that NEAT cannot find useful feature networks for the problem for only learning over 2,000,000 samples. In fact, NEAT itself also performed poorly on this game for learning 10,000,000 samples. Even though, NEAT+PGS can still manage to achieve similar or even slightly better performance than PGS, indicating the strong resilience of NEAT+PGS on low quality feature networks.

Note that, we are aware of that NEAT can actually achieve state-of-the-art performance on some Atari games as reported in [4, 10]. However, the purpose of developing NEAT+PGS is not to achieve the best scores on different games but to understand the effectiveness of feature learning.

Based on these findings, we can conclude that NEAT+PGS is a generally effective approach. In NEAT+PGS, feature learning and policy learning are mutually supplementary processes. Particularly, even one process fails on a problem, the other process can cope and ensure eventual good performance. When both processes are effective, NEAT+PGS can achieve clearly better performance.

5.3 Analysis on Sample Efficiency

Sample efficiency can be understood as the learning speed of an algorithm which is measured in terms of number of samples used for achieving good performance. In view of this, we choose the following learning speed metric to measure sample efficiency, i.e.,

$$\text{Score} = \sum_{t=0}^{1000} \frac{1}{10000} R_t, \tag{10}$$

where t is the tracking point (i.e., 10,000 samples), R_t is the average total reward obtained after learning 10,000 samples across different trials. With the metric, higher learning speed implies higher sample efficiency. Note that, the metric can be intuitively interpreted as the area under the curve.

In Table 4, we present the sample efficiency comparisons measured by using (10), and top three results for each game are highlighted. Evidently, NEAT+PGS algorithms achieved highlighted results in most cases. For cases on Freeway and Seaquest, the superiority of NEAT+PGS to PGS algorithms is observably significant. For other cases, we still have NEAT+PGS achieve the competitive results to those PGS baselines, even on Breakout. These findings indicate that NEAT+PGS can be more sample efficient than NEAT and PGS algorithms on RAM-based Atari games.

6 CONCLUSIONS AND FUTURE WORK

Motivated by the limitations of NEAT and NEAT+AC for large-scale RL problems, we have developed a new learning scheme called NEAT+PGS that seamlessly integrates NEAT based feature learning and policy gradient search. In comparison to previous works, this scheme has been shown two advantages. First, a clear separation between feature learning and policy learning is achieved to achieve effective knowledge sharing and learning among agents so that the sample efficiency can be improved. Second, a promising way of integrating NEAT with various cutting-edge PGS algorithms is developed to enable training policy networks with deep structures so that large-scale RL problems can be effectively addressed. The experimental results also confirmed these advantages found in NEAT+PGS.

In the future, we will further investigate the reliability of NEAT+PGS to hyper-parameter settings. It is also interesting to investigate the effectiveness of using other suitable evolutionary methods for training feature networks, possible methods can be HyperNEAT, Evolutionary Strategy, Covariance Matrix Adaptation Evolutionary Strategy and so forth. These methods may be more suitable for IMAGE-based Atari games rather than RAM-based Atari games. Additionally, we can explore possibilities of using other advanced PGS algorithms such as ACER, ACKTR, and PPO. By using these aforementioned techniques, we are expecting to tackle IMAGE-based Atari games.

REFERENCES

[1] David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. 2017. The Shattered Gradients Problem: If resnets are the answer, then what is the question? *arXiv preprint arXiv:1702.08591* (2017).

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.

[3] Shalabh Bhatnagar, Richard S Sutton, Mohammad Ghavamzadeh, and Mark Lee. 2009. Natural actor–critic algorithms. *Automatica* 45, 11 (2009), 2471–2482.

[4] Alexander Braylan, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen. 2015. Frame Skip Is a Powerful Parameter for Learning to Play Atari. In *AAAI Workshop: Learning for General Competency in Video Games*.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *arXiv* (June 2016). arXiv:cs.LG/1606.01540v1

[6] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. 2013. A survey on policy search for robotics. *Foundations and Trends® in Robotics* 2, 1–2 (2013), 1–142.

[7] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2017. OpenAI Baselines. https://github.com/openai/baselines.

[8] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. 2016. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*. 1329–1338.

[9] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 315–323.

[10] Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. 2014. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 4 (2014), 355–366.

[11] Jens Kober and Jan R Peters. 2009. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*. 849–856.

[12] Nate Kohl and Risto Miikkulainen. 2009. Evolving neural networks for strategic decision-making problems. *Neural Networks* 22, 3 (2009), 326–337.

[13] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. 2017. Evolving Deep Neural Networks. *arXiv* (March 2017), arXiv:1703.00548. arXiv:cs.NE/1703.00548

[14] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[16] Yiming Peng, Chen Gang, Mengjie Zhang, and Yi Mei. 2017. Effective Policy Gradient Search for Reinforcement Learning Through NEAT Based Feature Extraction. *SEAL* 10593, 8 (2017), 473–485.

[17] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International Conference on Machine Learning*. 1889–1897.

[18] John Schulman, Philipp Moritz, Sergey Levine, Michael I Jordan, and Pieter Abbeel. 2015. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv* cs.LG (2015).

[19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. (July 2017). arXiv:1707.06347

[20] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life* 15, 2 (2009), 185–212.

[21] Kenneth O Stanley and Risto Miikkulainen. 2002. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 569–577.

[22] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation* 10, 2 (2002), 99–127.

[23] Kenneth O. Stanley and Risto Miikkulainen. 2004. Competitive Coevolution Through Evolutionary Complexification. *Journal of Artificial Intelligence Research* 21, 1 (Feb. 2004), 63–100. http://dl.acm.org/citation.cfm?id=1622467.1622471

[24] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[25] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.

[26] Thomas G van den Berg and Shimon Whiteson. 2013. Critical factors in the performance of HyperNEAT. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 759–766.

[27] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. 2016. Sample Efficient Actor-Critic with Experience Replay. *arXiv* (Nov. 2016), arXiv:1611.01224. arXiv:cs.LG/1611.01224

[28] Shimon Whiteson, Peter Stone, Kenneth O Stanley, Risto Miikkulainen, and Nate Kohl. 2005. Automatic feature selection in neuroevolution. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM, 1225–1232.

[29] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. 2017. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. In *Advances in neural information processing systems*. 5285–5294.