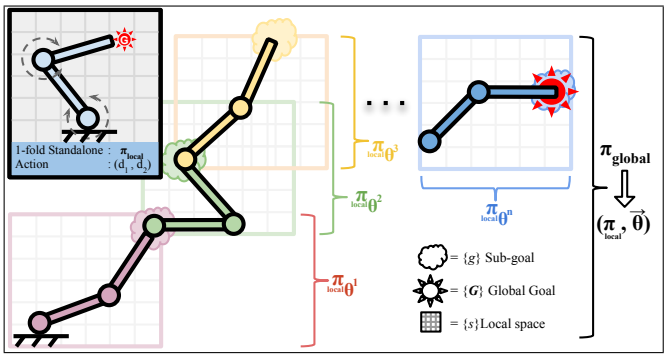
Policy Reuse in Reinforcement Learning for Modular Agents

Sayyed Jaffar Ali Raza and Mingjie Lin
e-mail: jaffar@knights.ucf.edu, mingjie@eecs.ucf.edu

***Abstract*— We present reusable policy method for modular reinforcement learning problem in continuous state space. Our method relies on two-layered learning architecture. The first layer partitions the agent's problem space into *n-folds* sub-agents that are inter-connected with each other with dexterity identical to original problem. It further learns a local control policy for standalone *1-fold* sub-agent. The second layer learns a global policy to reuse 'already learnt' standalone local policy over each *n* sub-agents by sampling local policy with global parameters for each sub-agent—parameterizing local policy independently to approximate non-linear interconnections between sub-agents. We demonstrate our method on simulation example of 12-DOF modular robot that learns maneuver pattern of snake-like gait. We also compare our proposed method against standard single-policy learning methods to benchmark optimality.**

***Keywords*- modular RL, interconnected multi-agents, reusable policy**



Figur1.Global Policy—Reusing 1-fold standalone policy π_{local} with global parameter vector $\vec{\theta}$ for each inter-connected modular agent.

I. Introduction

One of the fundamental objective of artificial intelligence is to achieve optimal behavior for stochastic system in dynamically changing environment. This involves planning to solve complex tasks by observing state of the environment and taking actions based on that observations. Recently reinforcement learning (RL) has shown significant contribution in achieving continuous control for virtual agents in game worlds [1]. The de-facto RL methods allows an agent to interact with the environment and learn by taking temporally extended actions—actions that are returned from a closed-loop policy over one time-step behavior [2]. Using the policy for choosing better actions over time would help an agent to improve its efficiency to solve a problem. However for agents with exceedingly complex state and action space representations—like snake maneuver robot; learning a singular closed-loop policy for such agents cost significant computation power [3].

A number of approaches have been suggested to improve learning in complex environments. One approach is to introduce additional exploration knowledge by imitation to define a positive bias for gradient direction [4] . The user defines preferred sequence of actions with advice rules [5] to be followed in different set of states. Another approach is to accelerate learning new actions by extracting useful macro-actions by correlating commonalities in solutions to previous tasks [6]. The core technique of macro-actions is to reinforce policy to perform action b after action a more frequently if such a pattern had seen more reward in past—thus the system formulates a macro-action $\hat{a}\{b \rightarrow a\} \in A$. This gives an action-to-action mapping coupled with state-to-action mapping. The macro-actions prove to have increase in performance for grid-world problems with a condition of known model representation however for continuous horizons; computing a value function based on values emitting from both pairs (state-action and action-action) is not guaranteed to converge due to large variance in learning time of these pairs [6]. Hierarchical RL on an other hand specially addresses continuous environment spaces by using different abstraction levels to learn task-specific partial policies with computable bounds [7]. The knowledge of partial policies can be used for learning a new task by initializing Q-values for a new episode with previously learned Q-values [8], [9] however the transfer of knowledge requires expert mapping between states and actions, and it is also restricted to learning optimality of previous policies that were learnt partially for generic tasks.

In this paper we propose a modular reinforcement learning method for agents that share analogous structure and can be classified as inter-connected modular architecture. The main learning idea has two layers of abstraction. The first layer evaluates the agent's modularity and decomposes the problem space into symmetrical partitions with a condition of keeping dexterous modularity identical over entire system. The decomposition is said to be done in n-folds where n is the number of sub-modules partitioned over entire system. The first layer then independently learns control over only 1-fold of overall structure as shown in figure 1, the bold boundary box shows the 1-fold agent structure that is equipped with a policy π_{local} learnt at layer 1. We are considering the example of a serpentine (snake-like) robot agent with theoretically unknown number of DOFs and that its geometric structure

978-1-7281-3323-2/19/$31.00 ©2019 IEEE

has potentially bio-inspired gait maneuverability. The policy π_{local} is considered to be well-learnt for the given local space of (1-fold) sub-agent. We consider local policy learning as relatively preliminary for the first sub-agent in the system—such that the very first sub-agent of the system can virtually operate on π_{local} without any need of external parameter tuning. Looking at figure 1 again, we can see the intuitive reuse formation of policy π_{local} over entire agent architecture. Due to restricted inter-connection between each sub-agent, they are subject to geometric constraints and sharing a singular policy would not suffice optimality for every sub-agent in the system. For instance, the local policy π_{local} is learnt with a stationary base-joint grounded all the time for the very first sub-agent, where as rest of the sub-agents experience dynamically floating bases—the end effector of precedent is base to the subsequent as shown in figure 1.

In order to utilize the similar policy for all sub-agents, it needs to adapt parametric changes over entire agent structure. π_{local} can be bounded with a global parameter vector $\vec{\theta}$ that can be used to sample and parameterize local policy before assigning any tasks to sub-agents. Once the first layer finishes partitioning the complex agent into n-folds and trains a standard policy locally for single sub-agent; then the parameter $\vec{\theta}$ learning is done at the second layer. $\vec{\theta}$ is a vector that comprises of unit parameters for each sub-agent such that unit vector $\hat{\theta} \in \vec{\boldsymbol{\theta}}$. The second layer learns a global policy $\pi^* \rightarrow \{\pi_{local}, \vec{\theta}\}$ to use learned parameters in $\vec{\theta}$ and sample the sub-goals for all sub-agents—following a global objective terminal state. The action space $\mathcal{A}$ for global policy learning tend to be a sequence of real valued coordinates that are taken by π_{local} as local goals to each sub-agent. The formation of global actions and parameter learning is elaborated with more detail in later sections. In a nut-shell, we propose a modular policy reuse idea that improves learning time by partitioning dexterous agent into symmetrical sub-modules in continuous state and action space and learn a policy over standalone sub-module as first part of idea. The next and final part of methods learns a guided parameter vector to impose standalone policy repeatedly over all connected sub-modules by building upon independent parameter updates over identical local policy for all sub-agents. We test our proposed method on a $6+6$ DOF serpentine robot in a simulation world and we demonstrate optimal maneuverability control of the agent.

II. Related Work

Effective exploration is one of the main challenges in MDPs. The conventional ϵ-greedy algorithm can be effective, however in large state spaces, they are less effective to explore the full state-action space [10]. For exceedingly large state spaces hierarchical RL has been widely practised, where multiple objectives and temporal abstractions are adopted to facilitate space explorations. For example, in [11] and [12], at each time state s_t, a higher level controller chooses the goal $g_t \in G$ where G is the set of all possible goals currently available for the controller to choose from. These goals provide intrinsic motivations for the agent so that it finishes the overall complex task by choosing a sequence of goals in the right order. Each goal remains active for some amount of time, until a predefined terminal state is reached. These methods are also used for end-to-end reinforcement learning where the agent instead of learning the value functions completely, explores the environment and builds upon policy over experience gained from enivronment exploration [13], [14]

Hierarchical or layered reinforcement learning is broadly studied topic in field of artificial intelligence. It is widely used for learning and optimizing policies for environments that cannot be modeled into discrete representations and are also infinitely huge in terms of model dimensions. The crucial part in modeling policy is adapting actions based on dynamic primitives. Policy based methods such as splines gradients or often called vanilla policy gradients algorithms are popular in discrete space behavioral policy modeling due to their trajectory centric representation of actions [15]. These methods are widely seen to solve complex tasks like bi-pedal robot walking [16] or learning primitive motor angles to control joints [17]; also these vanilla policy gradient methods are relatively easy to implement and adapt in real-world as well. However such policies are limited to small-scale horizons that are often discretized with extended granularity resolution [10], [18]. Besides this, other notable approximation techniques [19]–[22] also involve Bayesian optimization of cost functions by formulating a conventional transform function type approach towards specific gradient trajectories with their covariance already estimated. Such approaches convert the sparsity of rewards into probability map given that rewards are of finite and discounted nature. Next step is to take account of calculated covariance and apply traversal algorithm and update policy in that direction. The algorithm takes current and predicted probabilities and calculates the Bayesian optimality recursively. When it comes to integrating more "moving variables" for decision making and taking actions in continuous horizon – recent works intent to use deeper and larger systems that can simultaneously perform end-to-end control for a wide range of task in parallel [13], [20]. Such methods normally are build over networked systems like convolutional neural networks (CNN) with extended array of parameters that are trained using a guided policy search method [23] that represent the policy learning as supervised approach guided by the trajectory [13], [24]. Such methods are also seen to solve real-world problems like putting a cap on bottle [13] or stacking Lego blocks [15] Talking about hierarchies, researchers also consider RL problems as modular task and claim that every problem is actually composed of concurrent sub-problems with matter of abstraction levels [3]. This setup is very similar to classical RL, except that an extra layer of abstraction is defined on the set of actions, so that there are specific actions for each of the goals. Different approaches to hierarchical RL result in variants on this overall approach, choosing different trade-offs in flexibility, training speed, and other properties [25]. Our approach in this paper is similar to these hierarchical methods in the sense that multiple levels of reinforcement learning is employed, but differ in that

our main objective is to make our overall control learning to be more scalable and more efficient. As such, our main focus is to reuse locally learned policy over modular system and achieve admissible optimality over entire system. We use the underlying methods of deep Q-learning to learn local policy and further deploy it with parametric assignments.

III. Learning Model

In our work We extend existing hierarchical frameworks [7], [26] which are based on expert behavioral demonstration at different abstraction levels for learning optimal trajectories for control a robot agent. In this section we introduce formal reinforcement learning methods for value iteration and policy optimization for local learning of π_{local} and through local behavioral experiences we derive global formal method of connected multi-agents. We consider an infinite-horizon environment with memoryless attributes such that a state tuple is an MDP with (S, A, P, R) can use parameterized function approximators [27] as a value function V_i to store the Q-value of given (currently observing) state and generalize that for unseen (expected) state value V_{i+1}. The parameters for such function approximator can be tuned to achieve optimality – updating values over continuous time.

$$\begin{aligned} V_{i+1}(S_i) &= V_i(S_i) + \alpha * \delta_i \approx V_\pi(S) \\ \delta_i &= r_i + \gamma(V_{i+1}(S_{i+1}) - V_i(S_i)) \end{aligned} \tag{1}$$

The value function V states simply estimates the expected maximized value given in certain state s_i under some policy π. In other words it uses a recursive approximation to calculate all possible values for given state s_i – not state (s_i, a) pairs like in Q-Function.

$$\begin{aligned} V(s(t))^\pi = E(r(t) + \gamma r(t+1) + \\ \gamma^2 r(t+2) + \quad ... + \gamma^{\mathrm{i}} r(t+i)) \end{aligned} \tag{2}$$

From dimensionality point of view, it is out of scope to calculate Q values for all possible combinatorial (s_i, a) tuples because state and action pairs can go infinitely long. Short hand for value function can be written as equation (3).

$$V(s(t))^* = max_{a'} Q(s, a') \tag{3}$$

In order to optimize the value function estimation we need to have tuning parameters to adjust the approximations accordingly that can optimally generate highest Q value possible. Let θ_V be the parameter we need to optimize our value function approximator then using equation (1).

$$\theta_{V_{i+1}} = \theta_{V_i} + \alpha * \delta_i \frac{\partial V_i(S_i)}{\partial \theta_{V_i}} \tag{4}$$

Expanding $V_i(S_i)$ by partial derivative of (2) and that is basically expected sum of cumulative rewards.Since in next steps we want global reward as function of local values we [illegible] parameterized sum of differential local rewards over existing parameter θ and generating a uniform slope from the policy of sequence of local policies for adjusting gradient of global policy. Our local policy is based on an MDP that

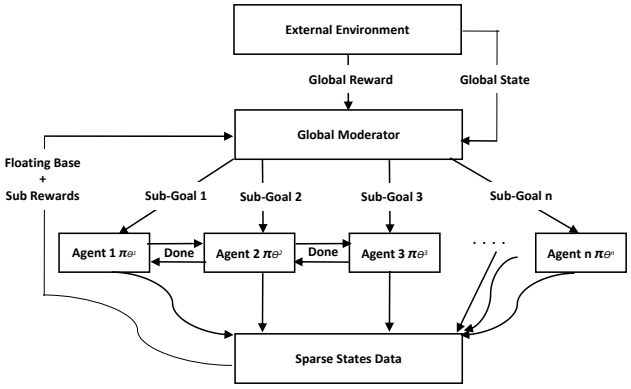


Figure2.Workflow of sub-agent interaction with global controller that receives observation from external environment and emits sub-goals.

generates max value for (s, a). We can generalize reward at each time step as equation (5) where γ is the discount factor and eewrite (2) as sum of expected local reward as (6)

$$\begin{aligned} \mathbf{R} &= r_i + r_{i+1} + r_{i+2} \cdots + r_{i+n} \\ \Rightarrow \mathbf{R} &= \sum_{i=0}^{n=maxSteps} \gamma^i r_i \end{aligned} \tag{5}$$

$$V(s(t))^\pi = \mathbb{E}\left(\sum_{i=1}^{n=maxSteps} \gamma^{\mathrm{i}} r(i) \right) \tag{6}$$

Assuming we have a well trained policy for local agents that generates maximum expected value for action selection every time. We can then further simplify equation (6) as following

$$V(s(t))^\pi = \max_{(s_t, a, s_{t+1})} r(s, a, s_{t+1}) \tag{7}$$

Since we need to formulate a global policy that observes local rewards in order to generate optimal sub-goals for each agent and collectively yields to obtain global objective as well. However when an agent X_i transitions to specified sub-goal, then agent X_{i+1} and all succeeding agents should adapt to it's positioning continuously and each agent X_m should get its sub-goals adjusted after each step of preceding agent X_{m-1} until the last agent's end effector positions precisely on the goal or objective position. This adaptive relationship can be investigated by observing each agents' combinatorial rewards in such fashion that each step should be taken with most choosing optimal combination of local rewards—global policy needs to generate combination of sub-goals that yield maximum reward distributed non-linearly among each agent and yield towards achieving the global objective.

$$\underbrace{\frac{\partial V(S(t))^\pi}{\partial step\#}}_{\text{global}} = \mathbb{E}\left(\underbrace{\frac{\partial V(s(t))^\pi}{\partial step\#_{agent1}}}_{\text{agent1}} + \underbrace{\frac{\partial V(s(t))^\pi}{\partial step\#_{agent2}}}_{\text{agent2}} + ... \right) \tag{8}$$

We can then compute global value function as expected sum of local value functions and use any policy optimization method to learn global approximation function for obtaining a

combination of maximum expected local rewards for each local agent.The global moderator needs to take steps in direction $\nabla_\theta V$. This gradient can be written as (8)

$$\nabla V(\pi_{param=\theta}) = \mathop{\mathbb{E}}_{(s,a)\in S,A}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)] \quad (9)$$

Consider N agents with policies $\pi_i \ \forall \ i = 1..N$ parameterized by $\theta_i \ \forall \ i = 1..N$. Each agent performs sequence of actions such that a single state-action representation grows as $Q_i^\pi(s, \hat{a_1}, \hat{a_2}, ...\hat{a_N})$ $S \times A \rightarrow [0,1]$, $\hat{a} \in \vec{A}$ The deterministic objective gradient function in eq: (9) can be rewritten with expanded Q function.

$$\nabla V(\pi_{param=\theta_i}) = \mathop{\mathbb{E}}_{\substack{(s,a) \\ a\in BUFFER}} [\nabla_{\theta_i} \log \pi_{\theta_i}(a|s) Q^\pi(s,a)]$$

$$BUFFER \rightarrow (s, s_t, \hat{a}, (r_1, r_2, ..., r_N)$$

$$\Rightarrow \mathop{Q}_{UPDATE} \rightarrow \sum_i r_i + \gamma Q_i^\pi \quad (10)$$

Actions sequences for a local agent in any given state is $\pi_i(s) \rightarrow \hat{a_i}$ given that π_i is well trained locally; $Q^* : s \in S \times \hat{a} \in \hat{A}$ which gives global policy expression $\pi^*(S)$ and value function $V^*(S)$

$$\pi^*(S) = \mathop{argmax}_{\vec{A}} Q^*(S, \vec{A})$$
$$V^*(S) = \mathop{max}_{\vec{A}} Q^*(S, \vec{A}) \quad \forall s \in \mathbf{S}$$
$$V^*(S) = \mathop{max}_{\vec{A}} [R(s \in S, \vec{A}) \ + \ \gamma \sum \ P(s^{'} \in S) \mid (s, \vec{A})]$$
$$\Rightarrow \mathop{max}_{\vec{A}} \sum_{S,A} \mathcal{P} \cdot \mathcal{R} \quad (11)$$

Partial derivatives with respect to the shared parameter θ for value functions in and logarithmic transition probability gives simplified expression for value function that can be iterated recursively in place of classic bellman function reducing effort by utilizing localized probability distributions over shared θ parameters

$$\because P(a_i) = a_i \log(\theta_i) + (1 - a_i) \log(1 - \theta_i)$$

for simplicity, assume all agents choose parameter $\theta_i$then for single $agent_i$ update:

$$\frac{\partial}{\partial \theta_i{}^i} V_{\theta_i}^i = \mathop{R}_{(S,\vec{A})} \cdot \frac{\partial}{\partial \theta_i} \left[P(\hat{a_1}, \hat{a_2}, \ldots, \hat{a_N} \right]$$

$$\frac{\partial}{\partial \theta_i{}^i} V_{\theta_i}^i = \mathop{R}_{(S,\vec{A})} \cdot \frac{\partial}{\partial \theta_i} \left[\frac{\vec{a}}{\theta_i} - \frac{(1-\hat{a})}{(1-\theta_i)} \right]$$

The global moderator generates sub-goals for local agents from local agents in episodic mode. Once an episode is finished the environment generates rewards as well – global moderator correlates local rewards with global objective (in terms of reward) and takes decision: to restart sub-goal assignment randomly or follow same trajectory if that trajectory does help approaching global convergence. Policy gradient method buffers all type of experiences, either a success or failure and then prioritize the most rewarding experiences from the buffer. Prioritizing rewards can be calculated by measuring temporal difference error. The critic computes the TD target value (FILLER) and then uses the temporal difference value to compute the loss *L* over the batch size *n* We can formulate actor and critic models by using neural networks. The weights and biases of critic network will be updated by backpropagating the error computed from loss function *L*. The weight updates for actor network will be formulated by computing the gradient of the value function (output of critic network) and the output of actor itself, that is actually a distribution over action bounds.

IV. Experiments

We carry out experiments over a hyper-redundant serpentine robot arm with 12-DOF maneuvering capabilities. The arm resembles maneuvering properties of an octopus tentacles. We performed experiments against Deterministic Proximal Policy Optimization (DPPO) method and Asynchronous Advantage Actor-Critic (A3C) baselines. Figure 3 shows running reward trend that is initially over-motivating due to high exploration rate but it settles down above zero ensuring that global predictate always receives positive value. We tested our idea against standard policy learning methods, our method performs better against baselines in figure 4. Though DPPO performs initially better due to its alternative surrogate update but as the exploration rate reduces, it seems to slow down as well.

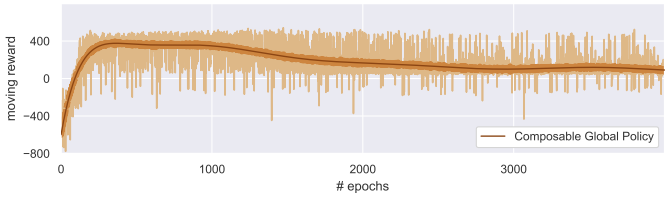


Figure 3. The global reward floats above zero to ensure constant intrinsic reward from sub-agents coupled with global objective signal

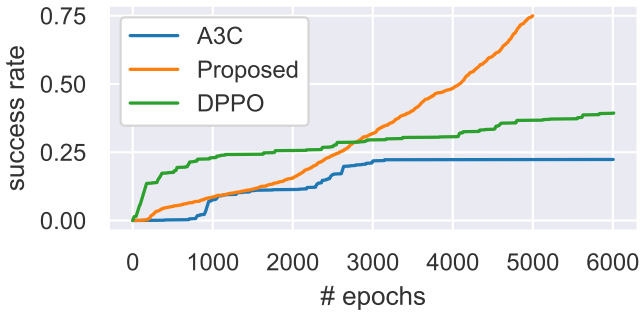


Figure4. Overall success rate of our proposed method vs. baselines

V. Conclusion

Modular reinforcement learning problems can be solved as model free methods. Our proposed method introduces reusable policy method that contributes towards scale-able policy learning. We have examined the performance of our method to learn a control policy for an agent comprised of complex geometry. Our method primarily delivers optimal results in shorter epoch counts ensuring that the global policy is locally guided by the 1-fold standalone policy. Secondly, the proposed method achieves substantial reduction in exploration effort of total state space—it explores the standalone space for local control and then the system only learns parameter vector to correlate previous learnt space over subsequent sub-agent spaces. Hence our results also present a comparison against standard methods to demonstrate improved success rate.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[2] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, 2000.

[3] Ö. Şimşek, A. P. Wolfe, and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning," in *Proceedings of the 22nd international conference on Machine learning*, pp. 816–823, ACM, 2005.

[4] S. Thrun and A. Schwartz, "Finding structure in reinforcement learning," in *Advances in neural information processing systems*, pp. 385–392, 1995.

[5] R. Maclin, J. Shavlik, L. Torrey, T. Walker, and E. Wild, "Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression," in *AAAI*, pp. 819–824, 2005.

[6] M. Pickett and A. G. Barto, "Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning," in *ICML*, vol. 19, pp. 506–513, 2002.

[7] J. Z. Kolter, P. Abbeel, and A. Y. Ng, "Hierarchical apprenticeship learning with application to quadruped locomotion," in *Advances in Neural Information Processing Systems*, pp. 769–776, 2008.

[8] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 2169–2176, IEEE, 2017.

[9] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *arXiv preprint arXiv:1702.08165*, 2017.

[10] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, "Composable deep reinforcement learning for robotic manipulation," *arXiv preprint arXiv:1803.06773*, 2018.

[11] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in neural information processing systems*, pp. 3675–3683, 2016.

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[13] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[14] [illegible] and M. Mateas, "On the difficulty of modular reinforcement learning for real-world partial programming," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 21, p. 318, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[15] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[16] P. Kormushev, B. Ugurlu, S. Calinon, N. G. Tsagarakis, and D. G. Caldwell, "Bipedal walking energy minimization by reinforcement learning with evolving policy parameterization," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 318–324, IEEE, 2011.

[17] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in neural information processing systems*, pp. 1547–1554, 2003.

[18] C. Guestrin, M. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *ICML*, vol. 2, pp. 227–234, Citeseer, 2002.

[19] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: Scaling up," in *Machine Learning Proceedings 1995*, pp. 362–370, Elsevier, 1995.

[20] E. Tzeng, C. Devin, J. Hoffman, C. Finn, X. Peng, S. Levine, K. Saenko, and T. Darrell, "Towards adapting deep visuomotor representations from simulated to real environments," *CoRR, abs/1511.07111*, 2015.

[21] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[22] P. Henry, C. Vollmer, B. Ferris, and D. Fox, "Learning to navigate through crowded environments," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 981–986, IEEE, 2010.

[23] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, pp. 1–9, 2013.

[24] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine, "Combining model-based and model-free updates for trajectory-centric reinforcement learning," *arXiv preprint arXiv:1703.03078*, 2017.

[25] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," *arXiv preprint arXiv:1710.09767*, 2017.

[26] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver, and D. P. Wierstra, "Continuous control with deep reinforcement learning," Jan. 26 2017. US Patent App. 15/217,758.