Cascade Attribute Network: Decomposing Reinforcement Learning Control Policies using Hierarchical Neural Networks

Haonan Chang * **Zhuo Xu** ** **Masayoshi Tomizuka** **

* *Dept. of Robotics, University of Michigan, Ann Arbor. (e-mail: harveych@umich.edu).*
** *Dept. of Mechanical Engineering, University of California, Berkeley. (e-mail: {zhuoxu,tomizuka}@berkeley.edu)*

Abstract: Reinforcement learning methods have been developed to achieve great success in training control policies in various automation tasks. However, a main challenge of the wider application of reinforcement learning in practical automation is that the training process is hard and the pretrained policy networks are hardly reusable in other similar cases. To address this problem, we propose the cascade attribute network (CAN), which utilizes its hierarchical structure to decompose a complicated control policy in terms of the requirement constraints, which we call attributes, encoded in the control tasks. We validated the effectiveness of our proposed method on two robot control scenarios with various add-on attributes. For some control tasks with more than one add-on attribute attribute, by directly assembling the attribute modules in cascade, the CAN can provide ideal control policies in a zero-shot manner.

Keywords: Reinforcement learning control,Deep Learning, Artificial Intelligence

1. INTRODUCTION

Reinforcement learning (RL) is an artificial intelligence approach that solves for automatic control policies, $\pi(a|s)$, that map the state space inputs s to the control command outputs a. The training of the RL policies is through the interaction of the agent and the environment, which is often modeled using a Markov Decision Process (MDP). The RL has been successful in solving many robotics and automation problems in simulations as well as real-world scenarios (Kahn et al. (2018)(Xu et al. (2020))). However, the wide application of the RL in automation is slowed down by several challenges, and one of the main challenges is the difficulty of RL policy training. The RL policy generally takes a great amount of computation power to train for complicated control tasks, and what makes it worse is that the RL policies are optimized based on certain fixed MDPs, and the knowledge encoded inside an optimized control policy are hard to transfer to other similar tasks. That is, new RL policies have to be trained from scratch even for those control tasks that are very similar to the pretrained task, which leads to a great amount of computation power waste.

Unlike the existing transfer learning approaches (Rusu et al. (2016a)) (Daftry et al. (2016)) that are lack of interpretable transferable features, we propose to decompose the complicated control problems in terms of its consisting requirement constraints, which we call attributes. The conception attributes refer especially to global characteristics or requirements that take effect throughout the task. For example, to solve an autonomous driving problem, we can first decompose the requirements of the task into a base target reaching attribute, an add-on obstacle avoidance and a speed limit attribute. Our methodology includes training an attribute module for each of the attributes and then assembling the attribute modules together to produce the overall policy. To deal with tasks under different attributes, we propose an RL framework called the cascade attribute network (CAN). In CAN, the attribute modules are connected in the cascade series. Our method has two main intriguing advantages:

(1) The decomposed attribute modules are much easier and faster to train compared with the overall control policy.
(2) The attribute related knowledge is encoded in interpretable modules, which can be built up to create versatile policies that can adjust to changes in the control tasks.

2. RELATED WORK AND BACKGROUND

2.1 Related Work

There have been lots of attempts to create versatile intelligence that can adjust to changes in the tasks as well. Transfer learning (Taylor and Stone (2009)) is a key tool that makes the use of previously learned knowledge for the better or faster learning of new knowledge. Rusu et al (Rusu et al. (2016a))(Rusu et al. (2016b)) establish a progressive network, in which newly added columns are laterally connected to previously learned columns for knowledge transfer. Drafty et al (Daftry et al. (2016)) and Braylan et al (Braylan et al. (2016)) also design network architectures for knowledge transfer in MAV control

---

* The first two authors contributed equally to the work.

and video game playing. Barreto et al (Barreto et al. (2017))(Barreto et al. (2019)) propose a scheme based on successor features, a value function describing the dynamics of the environment and a generalized policy improvement over multiple policies. In this way, the method can provide an efficient transferring among different tasks where the reward function is changed but the dynamics of the system remains the same. As for the combinations of transfer learning and imitation learning, Ammar et al (Ammar et al. (2015)) use unsupervised learning to map states for transfer, assuming the existence of distance function between different state spaces. Gupta et al (Gupta et al. (2017)) learn an invariant feature between different dimensional states and use demonstrations to increase the density of the rewards. Our recent works (Xu et al. (2018))(Tang et al. (2019)) on transfer learning utilizes interpretable trajectory as transferrable feature, but the human engineered features lack the versatility to be applied in various robotic control areas.

There are other methods seeking to learn a globally general policy: Curriculum learning (CL)(Bengio et al. (2009)) trains a model on a sequence of cognate tasks that get more and more challenging gradually, so as to solve hard tasks that could not be learned from scratch. Florensa et al (Florensa et al. (2017)) apply reverse curriculum generation (RCL) in RL. In the early stage of the training process, the RCL initializes the agent state to be very close to the target state, making the policy very easy to train. They then gradually increase the random level of the initial state as the RL model performs better and better. Sanmit et al (Narvekar and Stone (2019)) formulate the curriculum learning sequence as an MDP process, which can also be learned from experience. These results show that with a good consideration of the nature of the problem, CL can help reinforcement learning have a better convergence speed. Our policy training strategy is inspired by the idea of CL and achieved satisfying robustness for the policies. There are also researches in training modular neural networks, (Devin et al. (2017)) investigates the combinations of multiple robots and tasks, while (Andreas et al. (2017)) investigates the combinations of multiple sequential subtasks. Our work, along with our parallel work (Xu et al. (2019)) different from those works, looks into modularization in a different dimension. We investigate the modularization of attributes, the characteristics or requirements that take effect throughout the whole task.

3. THE CASCADE ATTRIBUTE NETWORKS

3.1 Problem Formulation

We consider an agent with a state-space A performing a class of tasks built up with one base attribute and a series of add-on attributes. We model the control tasks and the attribute in a unified form using MDPs. We denote the attributes using their index $\{0, 1, 2, \ldots\}$, where the 0th attribute is the base attribute, which usually corresponds to the most fundamental goal of the task, such as the target reaching attribute in the autonomous driving task. We define the state space of each attribute to be the minimum state space that fully characterizes the attribute, denoted $S = \{S_0, S_1, S_2, S_3 \ldots\}$. For example, let the base attribute be the target reaching attribute, and the 1st attribute be

the obstacle avoidance attribute. Then S_0 consists of the states of the agent and the target, while S_1 consists of states of the agent and the obstacle, but does not include the states of the target. Using the attribute as elements, one can build various control tasks, such as pure target reaching, target reaching while avoiding an obstacle, target reaching under external force influence, and so on.

Each attribute has an unique reward function as well, denoted $R = \{R_0, R_1, R_2, R_3 \ldots\}$. Each R_i is a function mapping a state action pair to a real number reward, i.e. $R_i : S_i \times A \to \mathbb{R}$.

3.2 Network Architecture

The architecture of the CAN is shown in Fig. 1. The training for a task with more than one attribute is divided into two parts: the training of the base attribute module and the add-on attribute module. In the training phase, first, an RL policy $\pi_0(a_0|s_0)$ is trained to accomplish the goal of the base attribute. The base attribute network takes in $s_0 \in S_0$ and outputs $a_0 \in A$, the reward function of the MDP is given by R_0. This process is a default RL training process.

Consider the control task with more than one attribute, without loss of generality, the 1st add-on attribute module is trained next, in a series of the base attribute module. The 1st attribute module consists of a compensation network and a weighted sum operator. The compensation network is fed with state $s_1 \in S_1$, and action a_0 chosen by π_0. The output of the compensation network is the compensation action a_1^c, which is used to compensate a_0 to produce the overall action a_1. The reward for the MDP is given by $R_0 + R_1$ so that the requirements for both constraints of attributes are satisfied. Since the parameters of the base attribute network are pretrained and fixed, the cascading attribute network would extract the features of the attribute by exploring the new MDP under the guidance of the base policy. It is noted that in the add-on attribute module, the weight of the compensation action a_1^c is initiated to be small and increased over the training time. That is, at the early stage of the training process, mainly a_0 takes effect, while a_1^c gradually gets to influence the overall a_1 as the training goes on. For the other attributes, the training of their attribute networks is the same.

Empirical results also show that for cases, where the number of the add-on attributes is small, and different add-on attributes do not entangle with each other, with carefully designed training schemes, the CAN shown in Fig. 2 can be used to provide an ideal control policy. In the CAN shown in Fig. 2, the ith attribute module takes in s_i and a_{i-1}, and outputs a_i that satisfies all the attribute before the ith module. The final output a_j is the overall output that satisfies all the constraints in the constraint array.

4. EXPERIMENTS

4.1 Environmental Setup

The experiments are powered by the MuJoCo physics simulator (Todorov et al. (2012)). The attribute modules
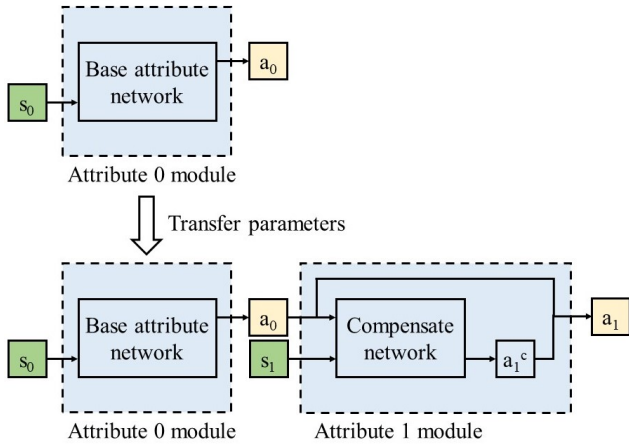
Fig. 1. The training procedure of an attribute module in the CAN: first train the base attribute module, then train the add-on attribute module based on fixed pretrained base module
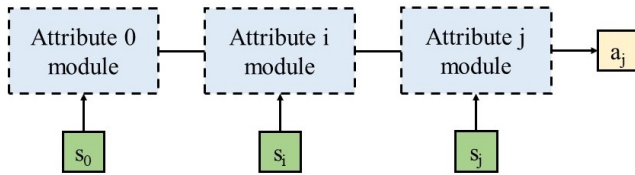

Fig. 2. One kind of usage of the CAN in multi-add-on-attributes tasks, by assembling add-on attribute modules in cascade to the base attribute module, the output action of the last attribute module is the outcome of the overall hierarchical policy network.

in our experiments are all three-layer fully connected networks which output Gaussian distributed stochastic actions, built using TensorFlow. The baseline RL algorithm we use is the PPO (Schulman et al. (2017)) method with GAE (Schulman (2015)) as the advantage estimator.

We evaluate the capability of the CAN on two types of robot scenarios in our experiments. One is a point robot with a 2-dimensional action space, and the other is an articulated robot with a 5-dimensional action space. For the point robot, the state space includes the position and velocity of the robot, and the action vector is the driving force applied to it. For the articulated robot the state spaces are the angle and angular velocity at all the joints, and the x-position and speed of the base of the robot, while the action vector includes the torques at all the joints, and the force applied to the base of the robot. For each robot agent, we implement one base attribute and four different add-on attributes. For each different attribute, we designed a reward function R_i. The attribute settings and reward functions are described as follow:

Reaching (base attribute)   Target reaching task, a common robot task, is defined to be the base attribute for both robot scenarios in our experiment. Given initial configuration and location of the target, the robots aim to to reach the target zone using their end effector. The reward function is shown below:
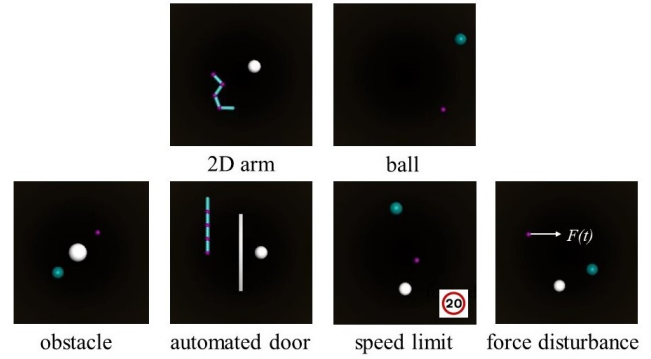

Fig. 3. The images in the top row show the two robot scenarios, in which the agents are performing the base attribute of target reaching. The bottom images show the four add-on attributes.

$$r_b(t) = \begin{cases} 1, \ reaches \ target \\ 0, \ other \ case \end{cases} \quad (4)$$

Obstacle (position phase)   In our obstacle avoidance attribute, a moving circular obstacle is placed in the work area. In our implementation, one obstacle avoidance attribute correspond to one circular obstacle. Nevertheless, many obstacle attributes are allowed to be assembled together to create a more complicated environment. The reward function is represented as following:

$$r_o^i(t) = \begin{cases} -0.3, \ touches \ i^{th} obstacle \\ 0, \ other \ case \end{cases} \quad (5)$$

Automated door (time phase)   The automated door attribute is a time controlled obstacle. This door is closed in the first and will open at some specific time slots. This attribute is challenging to train using RL since it punishes the agent even if it goes to the right direction at a wrong time. The reward function is designed as following:

$$r_d(t) = \begin{cases} -0.01, \ touches \ door \\ 0, \ other \ case \end{cases} \quad (6)$$

Speed limit (velocity phase)   Practical robots always have dynamic constraints on their joints which limit their speeds. The speed limit at time t, $L(t)$, is designed to be a time-variant function as shown in Fig 6. Denote the maximum joint velocity as v_{max}. Reward function can be written as:

$$r_s(t) = -0.3(max(v_{max} - L(t), 0)) \quad (7)$$

Force disturbance (acceleration phase)   In working scenarios, a robot can be influenced by force disturbance or repulsion force on their joints. The force disturbance attribute in our experiment corresponds to a time-invariant force disturbance added to a certain joint of both point robot and articulated robot.

The force disturbance only affects the dynamics function of the robot system, with no additional reward function added.

$$r_f(t) = 0 \quad (8)$$

Algorithm 1 Curriculum Learning

1: $RandomLevel$ = Initial Random Level
2: $\lambda = 1+$ Random Level Increase Rate
3: N = Batch Number
4: $LongTermR = Queue()$
5: **while** $RandomLevel <$ Terminal Random Level **do**
6:     Update the policy using PPO
7:     $Rewards \leftarrow RunEpisode(N)$
8:     $LongTermR.append(Rewards)$
9:     **if** $Average(LongTermR) > Threshold$ **then**
10:         $RandomLevel = RandomLevel \times \lambda$
11:         $Clear(LongTermR)$
12:     **end if**
13: **end while**

4.2 Training Schemes

To guarantee the capacity of the CAN, the attribute modules need to meet two requirements:

(1) The base attribute policies should be robust over the state space, rather than being effective only at the states that are close to the optimal trajectory. This enables the base attribute policies to be instructive when a compensation action is added on the top of it.

(2) The compensation action for a certain attribute should be close to zero if the agent is in a state where this attribute is not active. This property increases the capability of multi-attribute structures.

For the sake of the robustness of the attribute policies, we apply CL to learn a general policy that can accomplish the task starting from any initial state. The CL algorithm first trains a policy with a fixed initial state. As the training goes on, the random level of the initial state is smoothly increased, until the initial state is randomly sampled from the whole state space. The random level is increased only if the policy is capable enough for the current random level, which is reflected by the increase of the episodic reward. The pseudocode for this process is shown in Algorithm 1.

To guarantee the second requirement, an extra loss term that punishes the magnitude of the compensative action, $l_i^c \propto -\|a_i^c\|^2$, is added to the reward function so as to reduce $\|a_i^c\|$ when attribute i is not active.

4.3 CAN Performance

The first set of experiments tests the capability of the CAN to learn attributes and assemble learned attributes. We first train the base attribute module using the baseline RL algorithm with CL and then use the cascading modules to decompose the different add-on attributes based on the pretrained base module. In the actor-critic RL training using PPO, the maximum episodes is 10000. Both the actor network and critic network are trained using the Adam optimizer, with a batch number of 256, an initial learning rate of 0.0001, and are updated for 20 times in each training iteration. After training the two robot scenarios and all the four add-on attributes for each scenario, tasks. Therefore, the results show that the add-on attributes can be successfully added to the base attribute using the CAN. Fig. 4. shows the random level training log for the four add-on attribute modules in the point mass
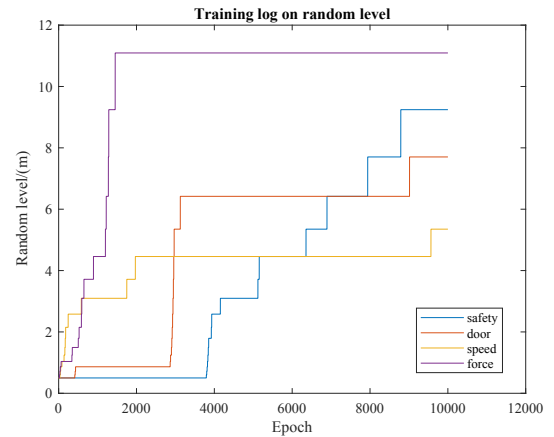

Fig. 4. Training log on random level for the point mass robot with the four add-on attributes. Because we are using CL training techniques, the random-level of the environment is increasing during the training. The higher the random-level is, the more general the final policy will be.

scenario, we ran 10 test episodes for the 8 combination tasks and received 10/10 success rate for all the tasks and Fig. 5 and Fig. 6 show the example episodes of the performing different attributes combinations.

We also show cases where the CAN can assemble more than one add-on attribute modules. Concretely, we first train the obstacle attribute module based on the point mass robot. Then we connect two identically parameterized obstacle attribute modules in series following the base attribute module, each handling one obstacle ball. The CAN structure is the same as the one shown in Fig. 2.. Fig. 7. shows two examples of the CAN zero shooting (Zero shooting means testing on a new task without extra training) the task where the moving point robot reaches the target while avoiding two obstacles simultaneously. Our perspective is not learning a universally effective policy for any control task, but utilizing the decomposition of the policy to reuse pre-learned knowledge, so as to achieve easier and faster training compared to training a new policy from scratch for new control task (as shown in Section IV-C).

4.4 Comparison with Baseline RL Methods

We compare the capability and efficiency of the CAN and the baseline RL on an obstacle task for the point mass robot. The CAN is trained with CL, with the base attribute pretrained and fixed. For the baseline RL, the task is trained from scratch. For baseline RL trained with RCL, we let the initial state be very close to the target in the early stage of the training phase. Using RCL, the baseline RL could gain positive reward in the early stage. The challenge would be whether the RL algorithm can maintain a high reward level as the random level increases.

The resulting comparison of the reward and random level are shown in Fig. 8, with the maximum training iterations exceeding 120000. It is shown that the baseline RL using CL barely learns anything. This is because the reward is too sparse and the agent is consistently receiving punish-
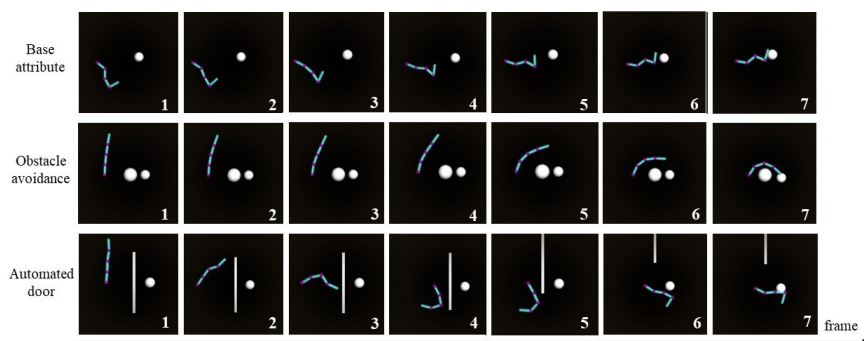
Fig. 5. Three example episodes for the articulated robot: target reaching, target reaching while avoiding an obstacle, or an automated door.
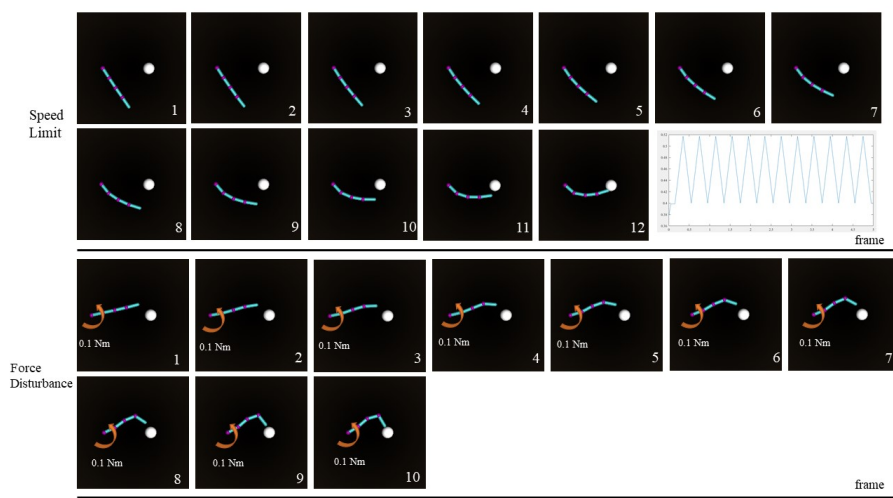

Fig. 6. Two example episodes for the articulated robot: target reaching under the speed limit, or force disturbance. Because of the limit of speed and force disturbance in the actuator, these two tasks take a longer time to finish compared to the three previous tasks. The speed limit and external force are also visualized.
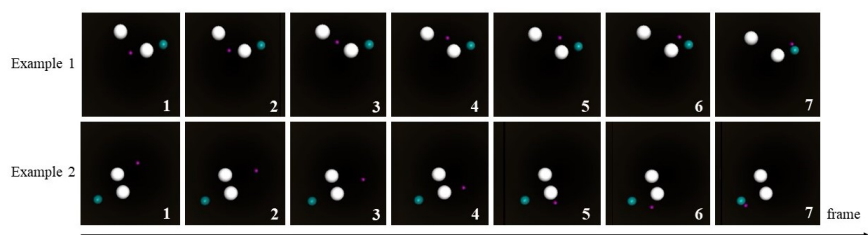

Fig. 7. Moving point robot (the pink ball) reaches the target (the green ball) while avoiding two obstacles simultaneously. Note that this task is never seen by the CAN. The obstacle attribute module is trained only once, and two identically parameterized obstacle attribute modules corresponding two different obstacles are assembled together to zero shoots the new task.

ment from the obstacle and falls into a local minimum of purely avoiding the obstacle and ignoring the target. For the baseline RL using RCL, in the early stage, the average discounted reward in an episode is high as expected. But as the random level rises, the performance of the baseline RL with RCL drops. Therefore, the random level increases slowly as the training goes on. The CAN, on the other hand, is able to overcome the misleading punishments from the obstacle. As a result, the random level of the CAN rises rapidly, and the CAN achieves terminal random level more than 10 times faster than the baseline.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we propose the attribute learning and present the advantages of using this novel method to decompose the control policies of complicated control tasks. The RL framework we propose, the CAN, uses cascading attribute modules structure to model the characteristics of the attributes. The attribute modules are trained with the guidance of the pretrained base attribute module. We validate the effectiveness of the CAN of decomposing and assembling attributes and show the advantages of the CAN in solving complicated tasks compared to the baseline
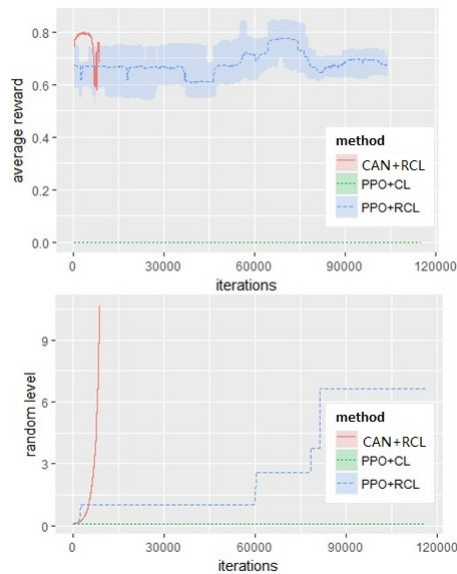
Fig. 8. Comparison between the performance of the CAN and the baseline RL (PPO) in the training phase.

RL. Our ongoing future work includes applying the CAN method on more advanced simulation scenarios such as autonomous driving and using a real 2D articulated robot similar to the one in the simulation experiment to verify the capability of the CAN in more general and complicated settings.

REFERENCES

Ammar, H.B., Eaton, E., Ruvolo, P., and Taylor, M.E. (2015). Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Andreas, J., Klein, D., and Levine, S. (2017). Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 166–175. JMLR. org.

Barreto, A., Borsa, D., Quan, J., Schaul, T., Silver, D., Hessel, M., Mankowitz, D., Žídek, A., and Munos, R. (2019). Transfer in deep reinforcement learning using successor features and generalised policy improvement. *arXiv preprint arXiv:1901.10964*.

Barreto, A., Dabney, W., Munos, R., Hunt, J.J., Schaul, T., van Hasselt, H.P., and Silver, D. (2017). Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, 4055–4065.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48. ACM.

Braylan, A., Hollenbeck, M., Meyerson, E., and Miikkulainen, R. (2016). Reuse of neural modules for general video game playing. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Daftry, S., Bagnell, J.A., and Hebert, M. (2016). Learning transferable policies for monocular reactive mav control. In *International Symposium on Experimental Robotics*, 3–11. Springer.

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

Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. (2017). Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*.

Gupta, A., Devin, C., Liu, Y., Abbeel, P., and Levine, S. (2017). Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*.

Kahn, G., Villaflor, A., Ding, B., Abbeel, P., and Levine, S. (2018). Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 1–8. IEEE.

Narvekar, S. and Stone, P. (2019). Learning curriculum policies for reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 25–33. International Foundation for Autonomous Agents and Multiagent Systems.

Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016a). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Rusu, A.A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. (2016b). Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*.

Schulman, J. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Tang, C., Xu, Z., and Tomizuka, M. (2019). Disturbance-observer-based tracking controller for neural network driving policy transfer. *IEEE Transactions on Intelligent Transportation Systems*.

Taylor, M.E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul), 1633–1685.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. IEEE.

Xu, Z., Chang, H., Tang, C., Liu, C., and Tomizuka, M. (2019). Toward modularization of neural network autonomous driving policy using parallel attribute networks. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, 1400–1407. IEEE.

Xu, Z., Chen, J., and Tomizuka, M. (2020). Guided policy search model-based reinforcement learning for urban autonomous driving. *arXiv preprint arXiv:2005.03076*.

Xu, Z., Tang, C., and Tomizuka, M. (2018). Zero-shot deep reinforcement learning driving policy transfer for autonomous vehicles based on robust control. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2865–2871. IEEE.