PROGRESSIVE REINFORCEMENT LEARNING WITH DISTILLATION FOR MULTI-SKILLED MOTION CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep reinforcement learning has demonstrated increasing capabilities for continuous control problems, including agents that can move with skill and agility through their environment. An open problem in this setting is that of developing good strategies for integrating or merging policies for multiple skills, where each individual skill is a specialist in a specific skill and its associated state distribution. We extend policy distillation methods to the continuous action setting and leverage this technique combine *expert* policies, as evaluated in the domain of simulated bipedal locomotion across different classes of terrain. We also introduce an *input injection* method for augmenting an existing policy network to exploit new input features. Lastly, our method uses transfer learning to assist in the efficient acquisition of new skills. The combination of these methods allows a policy to be incrementally augmented with new skills. We compare our progressive learning and integration via distillation (PLAID) method against two alternative baselines.

1 INTRODUCTION

To-do: Really need to stress that our method with integrating NEW skills, most other methods are just composing many polices without ideas how to best learning a NEW skill

Reinforcement Learning has been successfully used for complex control tasks in recent years. There has been multiple solutions for many difficult control tasks, however, little work has been done to combine learned skills. For *long lived* applications of complex control tasks a learning system may need to acquire and integrate additional skills. This can be seen in motor development of children, where there are a number of stages that are progressed though to learn robust motor skills Adolph and Berger (2007). Continual learning could offer gains in learning speed and allow the development of a single policy capable of performing a wide range of tasks. How to best learn a robust method for combining previously learned skills to create a mutli-skilled controller that is more capable than the individual controllers remains an open question. This is a challenging problem for a number of reasons; scalability, forgetting, how best to combine policies and understand which *expert* to use when.

Distillation refers to the problem of combining the policies of one or more *experts* in order to create one single controller that can perform the tasks of a set of *experts*. *Distillation* treats the problem as a supervised regression problem where the objective is to learn a model that matches the output distributions of all *expert* policies Parisotto et al. (2015); Teh et al. (2017); Rusu et al. (2015). However, what if a new task for which there is no *expert* is given, what is a reasonable method for learning the new task and including this new skill in the repertoire of the agent? One well-known technique in machine learning to significantly improve sample efficiency across similar tasks is to use Transfer Learning (TL) Pan and Yang (2010), which seeks to reuse knowledge learned from solving a previous task to efficiently learn a new task. However, transferring knowledge from previous tasks to new tasks is not always straightforward; there can be *negative transfer*, in some cases, a previously-trained model can take longer to learn a new task via fine-tuning than would a randomly-initialized model Rajendran et al. (2015).

Here we investigate methods to continually learn and integrate new tasks into an existing policy in an efficient and robust fashion. Importantly, to learn these new skills while not *forgetting* how to perform old skills. The main building blocks in this work are policy transfer and multi-task policy *distillation*. First, an existing policy is trained on a new task using TL, TL should increase the speed at which new skills is learned. Then the resulting policy that has learned the new task can be *distilled* with the existing policy, resulting in a new policy that can perform both tasks, with marginal *forgetting*.

In the *distillation* step we can also change the policy model design. For example, after learning to walk on flat and inclined ground we can add state features for the local terrain. Here we introduce the option to modify the *sensory-motor transform*, a model that maps from sensory input to motor or action outputs, by augmenting the policy network to include additional state features. This augmented version of the network will have additional capability to detect subsets of the state space and perform more robust locomotion

planning to travel in complex environments. These multi-skilled policies will then be the basis for further integration of new tasks. The combination of these methods constructs a scalable method to integrate new skills into a single policy.

2 Related Work

Knowledge reuse has been a popular area of machine learning and AI for some time Pan and Yang (2010); Taylor and Stone (2009). There has been a significant amount of work in the area including methods to combine models that have both learned to solve a different task and methods for using information from learning other tasks while learning a new one Teh et al. (2017). Here we outline some of the most relevant work in the area of Deep Reinforcement Learning (DRL) for continuous control environments.

Distillation Recent works have explored the problem of combining multiple *expert* policies in the reinforcement learning setting. A popular approach is to use supervised learning to combine each policy by regression over the action distribution. This approach has been shown to both yield model compression Rusu et al. (2015) and a viable method for multi-task policy transfer Parisotto et al. (2015) on discrete action domains including the Arcade Learning Environment Bellemare et al. (2013). We adopt these techniques and extend them for the case of complex continuous action space tasks and make use of them as building block.

Transfer Learning There is limited work done in this area on environments with continuous control tasks. The work in Rajendran et al. (2015) proposes that it may be possible but does not show any examples of this. The concept of appending additional network structure while keeping the previous structure to ensure there is no forgetting has worked well on the Atari games Rusu et al. (2015); Parisotto et al. (2015); Rusu et al. (2016); Chen et al. (2015) Similar work uses methods to reproduce data from all tasks to reduce the possibility of forgetting how to perform previously learned skills e.g, Shin et al. (2017); Li and Hoiem (2016). There is also the challenge of *catastrophic forgetting*, where as new skills are learned old ones are forgotten. Some, recent work seeks to negate this issue using selective learning rates for specific network parameters Kirkpatrick et al. (2017). A different approach to combining policies is to use a hierarchical structure Tessler et al. (2016). In this setting, previously-learned policies are available as options to execute for a policy trained on a new task. However, their approach assumes that the new tasks will be at least a partial composition of previous tasks, and they do not demonstrate reintegration of the newly learned tasks.

Hierarchical RL Another related line of work uses modularity to achieve transfer learning for robotic tasks. This modularity enables the swapping of network modules for different robot types over a similar task that was train with one particular robot Devin et al. (2017). Other works use Hierarchical Reinforcement Learning (HRL) as a method for simplifying a complex motor control problem, defining a decomposition of the overall task into smaller tasks Kulkarni et al. (2016); Heess et al. (2016); Peng et al. (2017) Introducing more modularity to enable transfer between differing task domains is a promising direction of research. Although, these works examines knowledge transfer it does not examine transfer and reintegration of policies for related tasks and associated problems such as *catastrophic forgetting*.

3 Framework

In this section we outline the details of the Reinforcement Learning (RL) framework. We also give an introduction to the concepts of TL and *distillation*.

3.1 Reinforcement Learning

Leveraging the framework of reinforcement learning, we frame the general problem as a Markov Decision Processes (MDP): at each time step t, the world (including the agent) is in a state $s_t \in S$, wherein the agent is able to perform actions $a_t \in A$, sampled from a policy $\pi(s_t, a_t) = p(a_t|s_t)$ and resulting in state $s_{t+1} \in S$ according to transition probabilities $T(s_t, a_t, s_{t+1})$. Performing action a_t from state s_t produces a reward r_t; the expected cumulative reward earned from following some policy π may then be written as:

$$J(\pi) = \mathbb{E}_{r_0,\ldots,r_T}\left[\sum_{t=0}^{T} \gamma^t r_t\right] \tag{1}$$

where T is a possibly-infinite horizon, and γ is the discount factor, capturing the intuition that future rewards become less impactful as they get further in the future.

The agent's goal is to learn an optimal policy, π^*, maximizing $J(\pi)$. If the policy has parameters $\theta_\pi \in \Theta$, then the goal may be reformulated to identify the optimal parameters θ^*_π:

$$\theta^*_\pi = \arg\max_{\theta_\pi} J(\pi(\cdot|\theta_\pi)) \tag{2}$$

We assume that actions follow a Gaussian distribution around a mean determined by the current state, $\mu_{\theta_t}(s_t)$. Thus, Our stochastic policy may be formulated as follows:

$$a_t \sim \pi(a_t \mid s_t, \theta_\pi) = \mathcal{N}(\mu(s_t \mid \theta_\mu), \Sigma) \qquad \Sigma = diag\{\sigma_i^2\} \tag{3}$$

where Σ is a diagonal covariance matrix with entries σ_i^2 on the diagonal, similar to Peng et al. (2017).

To optimize our policy, we employ policy gradient methods, a well-established family of techniques for reinforcement learning Sutton et al. (2000). The gradient of the expected reward with respect to the policy parameters, $\nabla_{\theta_\pi} J(\pi(\cdot|\theta_\pi))$, is given by:

$$\nabla_{\theta_\pi} J(\pi(\cdot|\theta_\pi)) = \int_S d_\theta(s) \int_A \nabla_{\theta_\pi} \log(\pi(a, s|\theta_\pi)) A_\pi(s, a)\, da\, ds \tag{4}$$

where $d_\theta = \int_S \sum_{t=0}^{T} \gamma^t p_0(s_0)(s_0 \to s \mid t, \pi_0)\, ds_0$ is the discounted state distribution, $p_0(s)$ represents the initial state distribution, and $p_0(s_0)(s_0 \to s \mid t, \pi_0)$ models the likelihood of reaching state s by starting at state s_0 and following the policy $\pi(a, s|\theta_\pi)$ for T steps Silver et al. (2014). $A_\pi(s, a)$ represents an advantage function Schulman et al. (2016). In this work, we adapt the Positive Temporal Differnce (PTD) update proposed by Van Hasselt (2012):

$$A_\pi(s_t, a_t) = I\left[\delta_t > 0\right] = \begin{cases} 1, & \delta_t > 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

$$\delta_t = r_t + \gamma V_\pi(s_{t+1}) - V_\pi(s_t) \tag{6}$$

where $V_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t \mid s_0 = s\right]$ is the value function, which gives the expected discounted cumulative reward from following policy π starting in state s. PTD has the benefit of being insensitive to the advantage function scale. Furthermore, limiting policy updates in this way to be only in the direction of actions that have a positive advantage has been found to increase the stability of learning Van Hasselt (2012). Because the true value function is unknown, an approximation $V_\pi(\cdot \mid \theta_v)$ with parameters θ_v can be learned, which can be formulated as a regression problem:

$$\text{minimize} \underset{s_t, r_t, s_{t+1}}{\mathbb{E}} \left[\frac{1}{2}\left(y_t - V_\pi(s \mid \theta_v)\right)^2\right], \qquad y_t = r_t + \gamma V_\pi(s_{t+1} \mid \theta_v) \tag{7}$$

3.2 Policy Distillation

When there exist many agents that have solved/mastered different tasks we may want to combine the skills of these different *experts* into a single *expert* agent. This process is referred to as *distillation*. *Distillation* does not necessarily produce an optimal mix of the given experts but instead tries to produce an *expert* that best matches the action distributions produced by the *experts*. This can be preferred because this method can function independent of the separate reward functions used when training each *expert*. Here we also favour *distillation*, as the method should scale better to the number of tasks or *experts* that are being combined.

3.3 Transfer Learning

Given an *expert* that has solved/mastered a task we want to reuse the knowledge the *expert* has in order to learn a new task. This problem falls in the area of *Transfer Learning* Pan and Yang (2010). When considering the distribution of states the existing *expert* is skilled at solving, (D_{ω_i} the *source* distribution) it may be advantageous to start learning the new, target task ω_{i+1} with *target* distribution $D_{\omega_{i+1}}$ using assistance from the existing *expert*. The agent learning how to solve the *target* task with domain $D_{\omega_{i+1}}$ is sometimes referred to as the *student*. When the *expert* is used to assist the *student* in learning the target task and can be referred to as the *teacher*. The success of these methods rely on the assumption that there is some overlap between the D_{ω_i} and $D_{\omega_{i+1}}$ state distributions.

4 PROGRESSIVE LEARNING AND INTEGRATION

In this section we detail our proposed learning framework for continual policy transfer and *distillation* (Progressive Learning and Integration via Distillation (PLAiD)).

In the acquisition (TL) step, we are interested in learning a new task ω_{i+1}. Here we assume the task to be somewhat similar to previous tasks ω_i such that transfer can be beneficial. The more skilled an agent is the more likely it will have some knowledge that will assist in learning a new task. We adopt the straightforward transfer learning strategy of using an existing policy network and fine-tuning it to the new task. Since we are not interested in retaining any of our previous skills in this step, we can update this policy without concern for forgetting.

In the integration (*distillation*) step, we are interested in combining all past skills $(\pi_0, \ldots, \pi_i)$ with the newly acquired one π_{i+1}. Traditional approaches have commonly used policy regression where the tuples for the regression are generated by running trajectories of the expert policy on its task. Training the *student* on these trajectories does not always result in robust behaviour. This poor behaviour is because the *student* experiences a different state distribution than the *expert* during evaluation. To compensate for this state distribution difference sections of the trajectories should be generated by the *student*, this allows the *expert* to suggest behaviour that will pull the *student* state distribution closer to the *expert*'s. This is a common problem in creating a model to reproduce a given distribution of trajectories Ross et al. (2010); Bengio et al. (2015); Martinez et al. (2017); Lamb et al. (2016). We use the DAgger algorithm and demonstrate that for our locomotion problems, it is crucial to the robustness of distilled policies Parisotto et al. (2015).As our reinforcement learning algorithms is an actor-critic method, we also perform regression on the critic by fitting both in the same step.

4.1 HIGH LEVEL EXPERIMENT DESIGN

For the results presented in this work, the range of tasks to be solved share a similar action space and state space, as our main focus is to demonstrate continual learning between related tasks. However, the conceptual framework allows for simple extensions that would permit differing action or state spaces, since we can use a new network model as the *student* policy in the *distillation* step. In Figure 1 a flow diagram of PLAiD, showing how new skills are learned and integrated into a new policy, is given.

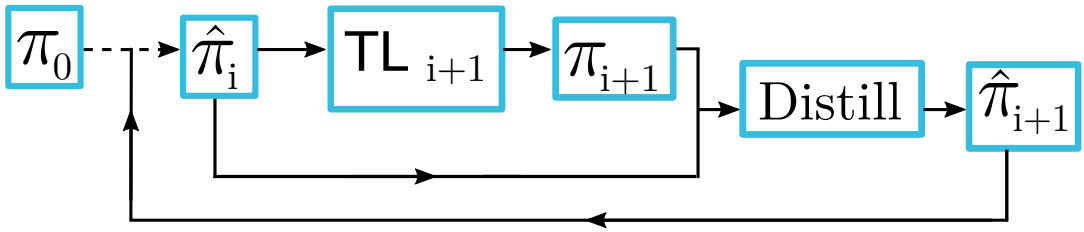
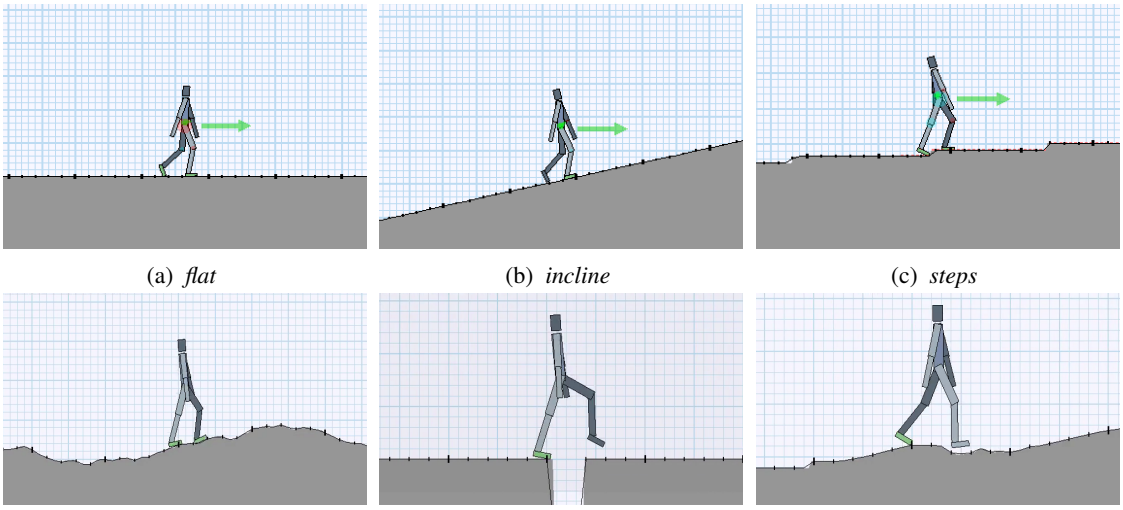


Figure 1: A flow diagram for PLAiD that can continuously integrating new skills. Given an initially random (π_0) or *expert* policy $\hat{\pi}_i$ on D_i, perform transfer learning (TL) starting from that policy to master a new skill, producing a new policy π_{i+1} that is an *expert* on D_{i+1}. After learning the new skill, policies $\hat{\pi}_i$ and π_{i+1} are *distilled* into a new policy that has the capabilities of both, $\hat{\pi}_i$ and π_{i+1}, i.e. an *expert* in $D_i \bigcup D_{i+1}$. During *distillation* it is possible to alter the design of the policy model, possibly reducing its size. This new distilled policy $\hat{\pi}_{i+1}$ is used as the initial policy $\hat{\pi}_i$ for the next skill to be learned.

5 RESULTS

The method in this work is evaluated on a humanoid robotic simulation. Here a bipedal character is trained to navigate multiple types of terrain, including *flat* and *incline*, shown in Figure 2a and Figure 2b as well as *steps* (Figure 2c), *slopes* (Figure 2d), *gaps* (Figure 2e) and a combination of all the terrains *mixed* (Figure 2f). The environment used is similar to the environments used in Peng and van de Panne (2016) where the goal is to learn the pd-targets to establish a robust walking gait. We add additional terrain types to this problem as well. An example of a biped trained to walk on an *steps* is shown in Figure 2c.

In this experiment, our set of tasks consists of To-do: 5 different terrains that a 2D humanoid walker learns to traverse. The details of the learning tasks as well as the reinforcement learning algorithm closely follow those outlined in Peng and van de Panne (2016) for *flat* terrain only. The goal in these tasks is to maintain a consistent forward velocity traversing various terrains, we further use a supplied motion capture clip of a natural human walking gait on flat ground. The 2D humanoid receives as input both a character and terrain state representation. A detailed description of the experimental setup is included in Section: 8.4. The tasks



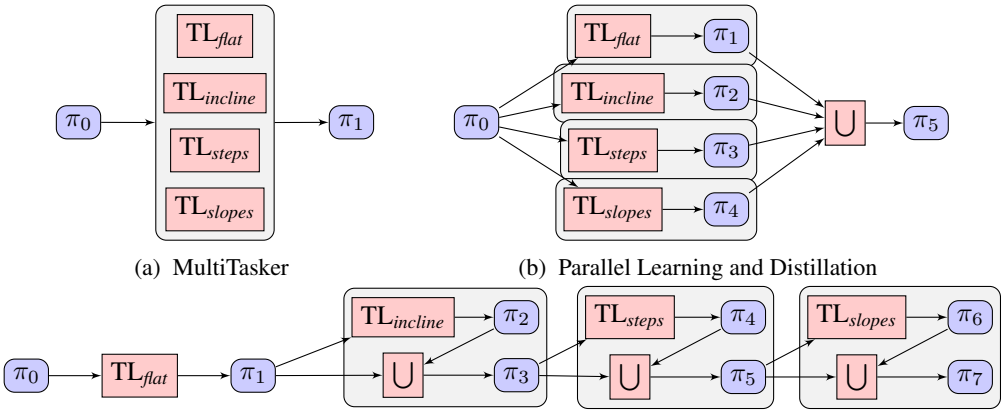
(a) *flat* (b) *incline* (c) *steps*

(d) *slopes* (e) *gaps* (f) *mixed*

Figure 2: The environments used to evaluate PLAiD.

are presented to the continual learner sequentially and the goal is to progressively learn to traverse all terrain types.

Although we focus on the problem of being presented with tasks sequentially there exist other methods for learning a multi-skilled character. We considered 3 overall integration methods for learning multiple skills. The first being a MultiTasker: controller that trys to learn multiple tasks at the same time (MultiTasker), where a number of skills are learned at the same time. It has been shown that learning may tasks together can be faster than learning each on separately Parisotto et al. (2015). The curriculum for using this method is shown in Figure 3a were during a single RL simulation time is spent learning each task. It is also possible to learn each separately but in parallel and then combine the resulting policies Figure 3b. We attempted to evaluate this method as well but we found that learning many skills from scratch was challenging, we were only able to get fair results for the *flat* task. Also, when a *new* task is learned with this model it would occur outside of the original parallel learning, leading to a more sequential method. The last version learns each task sequentially using TL from the previous, most skilled policy Figure 3c. This method works well for both combining learned skills and learning new skills.



(a) MultiTasker (b) Parallel Learning and Distillation

(c) PLAiD

Figure 3: Outlines the curriculum learning process used in this evaluation. The red circle with a $\bigcup$ in it denotes a *distillation* step that combines policies. Each gray box denotes one iteration of learning a new policy. (a) is a parallel version of our method. This did not work because we could only get good results on the *flat* terrain. (b) Is similar to the MultiTasker in that all tasks are being trained for at the same time. (c) Is PLAiD where we can take advantage of TL to assist with learning new tasks.

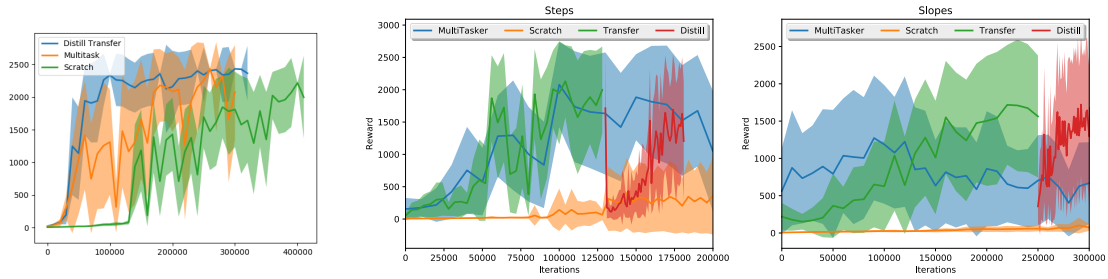
We evaluate our approach against two baselines. Firstly, we compare the above learning curriculum from learning new tasks in PLAiD with learning new tasks from Scratch: randomly initilized controller (Scratch). This will demonstrate that knowledge from previous tasks can be effectively transferred after *distillation*

steps. Second, we compare to the MultiTasker, to demonstrate that iterated *distillation* is effective for retention of learned skills.

5.1 TRANSFER LEARNING

The *pd-biped* is trained to produce a walking motion on flat ground (*flat*). We then initialize training on a similar task with slanted ground at random inclines (*incline*), and compare it to training a controller with two other methods. The Scratch version is trained to learn the task from a randomly initialized policy. The *distiller* version initializes its policy with one that was trained to walk on *flat* ground. The MultiTasker version initialized its policy from one trained to walk on *flat* ground but also alternates between *flat* and *incline* during training. The MultiTasker version is designed to evaluate the affects of adding skills to an agent while training on the old skills to prevent forgetting. This is in contrast to the PLAiD method that will integrate skills together after each new skill is learned.

In Figure 4a PLAiD is compared to the two baselines for *flat* + *incline*. The Scratch method learns the slowest as it is given no information about how to perform similar skills. The MultiTasker method is initialized from a controller that was trained to walk on flat ground. This controller has to learn multiple tasks together, which complicates the learning process as simulation for each task is split across the training process and the overall RL task can be challenging. This is in contrast to using PLAiD that will integrate skills together after each new skill is learned. However, it has been shown that training across many tasks at once can increase overall learning speed and potentially reduce forgetting Teh et al. (2017). We can see this in Figure 4b where the MultiTasker is learning the new task (*steps*) with similar speed. However, after adding more tasks, we can see in Figure 4c the MultiTasker beginning to struggle with learning many tasks at the same time. On the other hand PLAiD learns the new task faster and is able to integrate the new skill required to solve the task robustly. While training, the MultiTasker splits its time across the number of tasks equally. If we only consider the time taken to learn the new task the MultiTasker can learn quicker. However, as we will show later PLAiD will begin to out perform the MultiTasker as the number of tasks increases.



(a) (b) *steps* (c) *slopes*

Figure 4: TL comparison over each of the environments. (a) Shows the benefit of using TL when learning a new task/skill, *incline*, when the controller has some knowledge of *flat*. (b) TL for both PLAiD and MultiTasker is similar (c) PLAiD is showing faster learning after adding an additional skill. The *distiller* initializes its policy with the most recently learned *expert*. The MultiTasker is also initialized from the most recent *expert* but alternates between each environment during training. The learning for PLAiD is split into two steps, first the TL part in green followed by the *distillation* part in red. Using TL assists in the learning of new tasks.

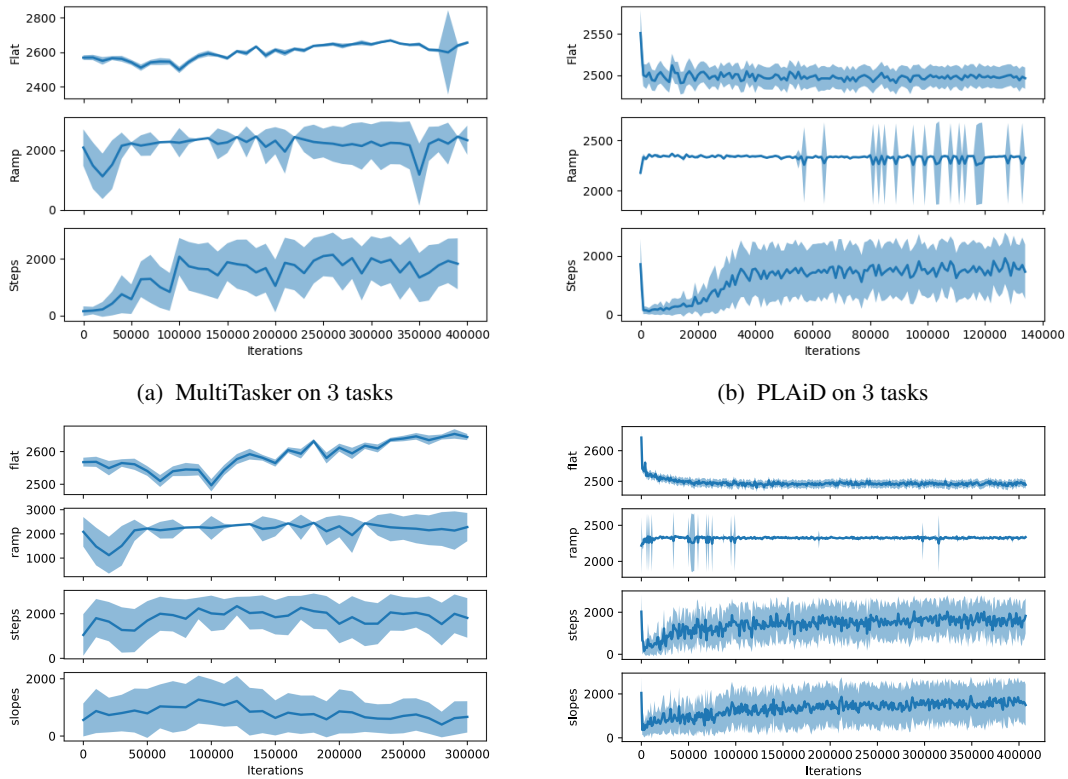
5.2 FEATURE INJECTION

An appealing property of using *distillation* in PLAiD is that the combined policy model need not resemble that of the individual *expert* controllers. Two different *experts* lacking state features and trained without a local map of the terrain can be combined into a single policy that has new state features for the terrain. These new terrain features assist the agent in identifying which task domain it is operating in.

In our framework we wish to initialize the *distillation* step with the previous policy. We introduce the idea of *feature injection* for this purpose. We augment a policy with additional input features while allowing it to retain its original functional behaviour in a way similar to Chen et al. (2015). This is achieved by adding additional inputs to the neural network and initializing the weights and biases associated with these new features to 0. this approach is demonstrated in the third step of the experiment (π_3 in Figure 3c) by combining the blind *flat* and *incline expert* policies via distillation.

5.3 DISTILLING MULTIPLE POLICIES

Learning multiple tasks at the same time my help the agent learn tasks quicker but this method may not scale with respect to the number of tasks. When training the MultiTasker over two or even three tasks (Figure 5a) the method shows improvement, however when learning a fourth task the method struggles, as shown in Figure 5c. Using PLAiD to combine the skills of many policies appears to scale better with respect to the number of skills being integrated. This is likely because distillation is a semi-supervised method which is more stable than the un-supervised RL solution. This can be seen in Figure 5b and 5d where PLAiD combines the skills faster, and can find higher value policies in practice.



(a) MultiTasker on 3 tasks

(b) PLAiD on 3 tasks

(c) MultiTasker on 4 tasks

(d) PLAiD on 4 tasks

Figure 5: These figures show the average reward a particular policy achieves over a number of tasks. After learning an *expert* for *flat* + *incline* a new *steps* task is trained. Figure (a) shows the performance for the MultiTasker and figure (b) for the *distiller*. The *distiller* learns the combined tasks fast, however the MultiTasker achieves marginally better average reward over the tasks Figures (c,d) show the performance of an *expert* on *flat* + *incline* + *steps* trained to learn the new task *slopes* for the MultiTasker (c) and *distiller* (d) (π_7 from Figure 3c). After including another task (*slopes*) the MultiTasker struggles to achieve its previous performance and the *distiller* combines the tasks gracefully.

Forgetting There are some indications that *distillation* is hindering during the initial few training steps. We are initializing the network used in the *distillation* step with the most recently learning policy after TL. The large change in the initial state distribution from the previous seen distribution during TL could be causing larger gradients to appear, disrupting some of the structure learned during the TL step, shown in Figure 5b and 5d. There also might not exist a smooth transition in policy space between the newly learning policy and the previous learned policy distribution.

6 DISCUSSION

Transfer Learning: For our transfer learning results we could be over fitting. The initial *expert* could have been over trained for the particular task it was learning. Making it more challenging for the policy to learning a new task, resulting in *negative transfer*. As we are using an actor-critic learning method we also studied the possibility of using the value functions for TL as well. We did not discover any empirical evidence that this assisted the learning process. When transferring to a new task the state distribution has changed and the reward function maybe be completely different. This makes it unlikely the value function

will be accurate on this new task. Also, value functions are in general easier and faster to learn than policies, implying that it is less important to transfer knowledge from these function approximations. We also find that helpfulness of TL depends on not only the task difficulty task but the rewards as well. Two tasks may overlap in state space but the area they overlap could be easily to reachable. In this case TL may not give significant benefit because the overall RL problem is easy. The greatest benefit is gained from TL when the state space that overlaps for two tasks is difficult to reach and in that difficult to reach area is where the highest rewards are achieved.

To-do: I wonder if the distillation also provides some form of bias reduction that helps generalization and transfer learning...

MultiTasker vs PLAiD The MultiTasker may be able to produce a policy that has higher overall average reward but in practice constraints can keep the method from combining skills gracefully. If the reward functions are different between tasks the MultiTasker can favour the task with higher rewards, as this tasks may receive higher advantage. It is also a non-trivial task to normalize the reward functions for each task in order to combine them. We have shown that the *distiller* can scale better to the number of tasks than the MultiTasker. The tasks used in this analysis could be considered too similar, with respect to state space and reward functions. We believe PLAiD would further outperform the MultiTasker if the tasks were more difficult and the reward functions dissimilar.

6.1 Limitations

After learning many new tasks the previous tasks may not receive a large enough potion of the *distillation* training process to preserve the *experts* skill well enough. How best to chose which data should be trained on next to best preserve the behaviour of *experts* is a general problem with multi-task learning. *Distillation* treats all tasks equally independent of their reward. This can result in very low value tasks receiving potentially more distribution than desired and high value tasks recieving not enough. We haven't needed the use a *one-hot* vector to indicate what task to the agent is performing. We want the agent to be able to recognize which task it is being given but we do realize that some tasks could be too similar to differentiate between. For example, walking vs jogging on flat ground.

6.2 Future Work

It could be helpful to perform some kind of task prioritization during the distillation step. This could assist the agent with forgetting or help relearn tasks. Here we use the Mean Squared Error (MSE) to pull the distributions of the *student* policies in line with the *expert* polices for *distillation*. It could be more advantageous to use a better metric for distance between the policies. Previous methods have used KL Divergence in the discrete action space domain where the value function, in this case Deep Q-Netowrk (DQN), encodes the policy. This could improve the TL after *distillation* step. In this work we are not focusing on producing the best policy from a *mixture of experts*, but instead matching the distributions from a number of *experts*. The difference is subtle but in practice it can be too difficult to balance many reward functions. It could also be beneficial to use some kind of KL penalty while performing *distillation*. Something similar to the work in Teh et al. (2017), that will help keep the policy from shifting too much / to fast during training.

7 Conclusion

Using PLAiD we gain the benefits of *distillation* as a supervised method to integrate policies together. This being more efficient then learning all skills together, and is a favourable method for integrating many skills together. Ideally, the more skilled an agent is the easier it should be to learn new skills.

References

Adolph, K. E. and Berger, S. E. (2007). *Motor Development*. John Wiley and Sons, Inc.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 1171–1179. Curran Associates, Inc.

Chen, T., Goodfellow, I., and Shlens, J. (2015). Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*.

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 *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2169–2176. IEEE.

Heess, N., Wayne, G., Tassa, Y., Lillicrap, T. P., Riedmiller, M. A., and Silver, D. (2016). Learning and transfer of modulated locomotor controllers. *CoRR*, abs/1610.05182.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems 29*, pages 3675–3683.

Lamb, A. M., ALIAS PARTH GOYAL, A. G., Zhang, Y., Zhang, S., Courville, A. C., and Bengio, Y. (2016). Professor forcing: A new algorithm for training recurrent networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 4601–4609. Curran Associates, Inc.

Li, Z. and Hoiem, D. (2016). Learning without forgetting. *CoRR*, abs/1606.09282.

Martinez, J., Black, M. J., and Romero, J. (2017). On human motion prediction using recurrent neural networks. *CoRR*, abs/1705.02445.

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

Parisotto, E., Ba, J. L., and Salakhutdinov, R. (2015). Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*.

Peng, X. B., Berseth, G., Yin, K., and Van De Panne, M. (2017). Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41.

Peng, X. B. and van de Panne, M. (2016). Learning locomotion skills using deeprl: Does the choice of action space matter? *CoRR*, abs/1611.01055.

Pouget, A. and Snyder, L. H. (2000). Computational approaches to sensorimotor transformations. *Nature Neuroscience*, 3 Suppl(Supp):1192–8.

Rajendran, J., Lakshminarayanan, A. S., Khapra, M. M., Prasanna, P., and Ravindran, B. (2015). Attend, adapt and transfer: Attentive deep architecture for adaptive transfer from multiple sources in the same domain. *arXiv preprint arXiv:1510.02879*.

Ross, S., Gordon, G. J., and Bagnell, J. A. (2010). No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686.

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2015). Policy distillation. *arXiv preprint arXiv:1511.06295*.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive Neural Networks. *arXiv*.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR 2016)*.

Shin, H., Lee, J. K., Kim, J., and Kim, J. (2017). Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *ICML*.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

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

Teh, Y. W., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. *arXiv preprint arXiv:1707.04175*.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., and Mannor, S. (2016). A Deep Hierarchical Approach to Lifelong Learning in Minecraft. *arXiv*, pages 1–6.

Van Hasselt, H. (2012). Reinforcement learning in continuous state and action spaces. In *Reinforcement Learning*, pages 207–251. Springer.

8 Appendix

8.1 Network Models

We used two different Network models for the experiments in this paper. The first model is a *blind* model that does not have any terrain features. The *blind* policy is a Neural Network with 2 hidden layers (512×256) with ReLU activations. The output layer of the policy network has linear activations. The network used for the value function has the same design except there is 1 output on the final layer. This design is used for the *flat* and *incline* tasks.

We augment the *blind* network design by adding features for terrain to create an agent with *sight*. This network with *terrain features* has a single convolution layer with 8 filters of width 3. This constitutional layer is followed by a dense layer of 32 units. The dense layer is then concatenated twice, once along each of the original two hidden layers in the *blind* version of the policy.

8.2 Hyper Parameters

The policy network models a Gaussian distribution by outputting a state dependant mean. We use a state independent standard deviation.

Each policy uses an epsilon-greedy, fixed standard deviation Gaussian for action exploration. For all of our experiments we fix the standard deviation to 0.1 and linearly anneal epsilon from 0.2 to 0.1 in 100,000 iterations and leave it from that point on. For the learning we use SGD with momentum with the same settings for all experiments. For DAGGER we gradually anneal the probability of selecting an expert action from 1 to 0 over 10000 iterations.

8.3 More details on terrain Injection

8.4 More details on training

For reinforcement learning, a time limit cap of 100 seconds is used.

To-do: Should discuss the quarks of hyper parameter changes for TL. For example, how to initialize the exploration rate.

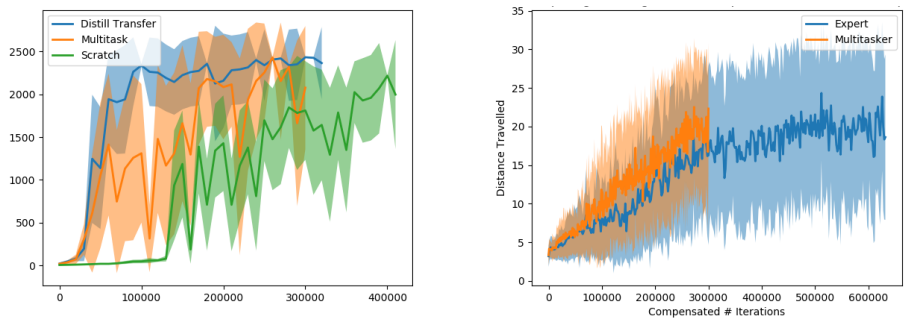
To-do: Which network to we initialize the distillation from?

How does *pd-biped* work details on reward function

Terrain Types To-do: Need Videos and images for each terrain type, *gaps*, *steps*, *slopes* and *flat*.

To-do: filmstrip of mixed terrain

8.5 More??



(a) (b)

Figure 6: (a) Shows the benefit of using TL when learning a new task/skill, *incline*, when the controller has some knowledge of *flat*. The *distiller* initializes its policy with *flat expert*. The MultiTasker is also initialized from the *flat expert* but alternates between *flat* and *incline* during training. (b) Shows that the MultiTasker can learn faster on *steps*, *flat* and *incline* than PLAiD learning the single task with TL does. To-do: Need policy evaluation comparison for with and without DAGGER. To-do: DUmping this here for now