Network Adaptation through Side-Tuning Networks

Isaac Perper
Department of EECS
MIT
iperper@mit.edu

Suraj Srinivasan
Department of EECS
MIT
surajs@mit.edu

Abstract

Adapting RL agents to new tasks is a challenging task that often leads to catastrophic forgetting on the original task, or rigidity preventing optimal learning of the new task. Side-tuning networks, first explained in [1], are an additive approach that enables efficient, modular learning to new tasks. In this project, we explored using the side-tuning approach in several OpenAI gym minigrid envrionments to adapt an agent to a new environment without decreasing performance significantly on the first environment. We showed that side-tuning can outperform fine-tuning for these purposes, but that the performance is significantly impacted by finding the right weighting to give the side-network. We were unable to achieve high-success of the agent using a smaller MLP side network, and overall we found that fine-tuning is more sample efficient.

1 Introduction

Humans are able to learn new things across many domains and at different times. Most importantly, humans are able to retain their knowledge of previous tasks even as their learning progresses. For example, a young child does not forget how to walk when it figures out how to hold a spoon. In reinforcement learning (RL), agents are able to learn how to accomplish a wide range of tasks, but policy-generalization and additional task learning by an agent through network adaptation remains challenging [2]. Learning multiple tasks at once is challenging, with some approaches jointly training agents using shared parameters and sharing of learned sub-tasks across agents [3]. However, this can require large amounts of data of both tasks. Many real-world learning tasks do not present all the challenges at once, rather new tasks are encountered one at a time. To this end, something called sequential learning can be done for disparate tasks, where the agent first learns one task (e.g., walking) and then later learns a new task (e.g., holding a spoon).

However, this sequential learning methodology suffers from two main issues. First, a phenomena known as *catastrophic forgetting* occurs – a process whereby the agent is better suited to more recent experiences and thus performs poorly when evaluated on previous environments. In addition, the agent may be hampered by *rigidity*, or the idea that constraints and rules imposed or learned by previous environments inhibit learning with respect to recent environments and thus, generalization is not achieved.

In an effort to mitigate the aforementioned issues with sequential learning, we explore a side-tuning approach developed by Zhang et al [1]. Effectively, side-tuning involves using a successful pre-trained network for some initial environment and freezing the weights of this model. This network is then used in conjunction with a side-network trained on a subsequent environment to allow for network adaptation. The networks work together in an additive fashion as detailed below.

In our project, we look to apply side-tuning to the problem of network adaptation in the context of reinforcement learning. Specifically, in the context of the MiniGrid environment, we explore how side-tuning compares to fine-tuning as an approach to model generalization and attempt to demonstrate a potential reduction in catastrophic forgetting and mitigation of ridigity. Furthermore, we conduct an analysis along other axes of comparison between side-tuning and fine-tuning including sample complexity and the effect of side-network size on performance.

2 Methods

The general approach of side-tuning is demonstrated in Figure 1, which was first implemented in [1]. Whereas fine-tuning focuses on adapting the same network from the previous task to the new task, side-tuning instead adds a entirely new network to the now-frozen output of the previous network.

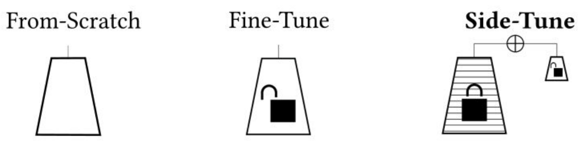

Figure 1: Side-tuning takes an additive approach to network adaptation and generalization. Fine-tuning initializes the network with the pre-trained version from scratch, and proceeds to 'fine-tune' the weights on a new task. Side-tuning freezes the weights from this pre-trained network, and simply adds the output of a new network. Figure adapted from [1].

2.1 Architecture

We call the network that the agent used to learn previous tasks the base-network $B(x)$. As shown in Figure 1, the side-tuning approach takes $B(x)$, and a side network $S(x)$. The combined output of this model for the new target task is represented by $R(x) = B(x) \oplus S(x)$ [1]. The operation $\oplus$ can combine $S(x)$ and $B(x)$ in any linear way, but generally both networks will have the same output dimension, as well as take the same observation from the environment.

It has been found that using alpha blending as the operator works well in practice. That is, we can represent the new network output as[1]: $R(x) = (1 - \alpha)B(x) + \alpha S(x)$. α can be a learned parameter, where α may indicate in important of $S(x)$ and $B(x)$ to the relevant task. In Section 4 we evaluate side-tuning with different fixed α to better understand this relationship.

One of the advantages of side-tuning is that the base model can be any model that maps the observation x to outputs, meaning models other than neural networks. However, for this project we only focus on the common scenario of using a pre-trained network for sequential learning. Additionally, the side-network can vary depending on the new task to be learned. In theory, if the new task is similar to the prior task, the side network can be small, since the additional information needed to be learned is small.

Another advantage of side-tuning for sequential learning comes from the additive approach. Each new task can use an additional small side-tuning network, rather than requiring a large entirely new network. The correct side-network for the task could even be automatically chosen, such as detecting a distribution shift in the side-networks' outputs would suggest that a given network may not be a high-confidence estimate for a certain task [1].

2.2 Training

At a high-level, training with side-tuning is as simple as freezing the weights of the base network and training via the gradients in side-network. There can be different strategies to initializing the weights of the side-network. When $S(x)$ has the same structure as $B(x)$ [1] proposes initializing $S(x)$ with the same (now frozen) weights as $B(s)$, but overall finds that various initialization strategies works well. For this project we choose random initialization, since many of our models are different from $B(x)$.

[1]Note that this is the opposite as the definition from [1]. We choose α to be the amount of influence of $S(x)$

3 Experimental Design

We chose to perform our analysis in the family of OpenAI MiniGrid environments [4, 5] as is offers lightweight sparse-reward environments which encompass a variety of tasks. Thereby, we leveraged the presence of different tasks in order to determine the efficacy of side-tuning to enable network adaptation and generalizability. Specifically, we utilized the MiniGrid-Empty environment at different grid sizes (`5x5, 6x6, 8x8`) to provide a baseline; the MiniGrid-Unlock environment for an increased difficulty tasks; and finally, the MiniGrid-DoorKey environment at different grid sizes (`5x5, 6x6`) to ultimately challenge the agent and observe the efficacy of side-tuning (hereon, `Empty`, `Unlock`, and `DoorKey`, respectively).

We began by training agents using PPO on all of the aforementioned environments for 500k steps and observed longitudinal convergence, shown in Fig. 2.

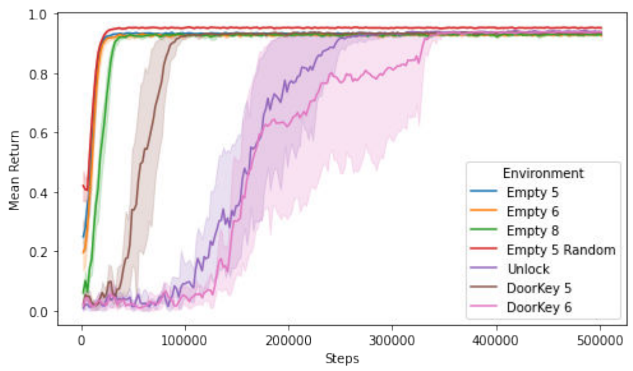

Figure 2: Training curves of PPO on each environment from scratch. The more complicated environments take up to 300,000 steps to converge.

In order to effectively apply side-tuning, we desired to find two environments which would be an effective test of network adaption. Thus, we performed a cross evaluation of the above 7 environments (i.e. 49 total evaluations) and determined the mean return across 100 episodes of evaluation:

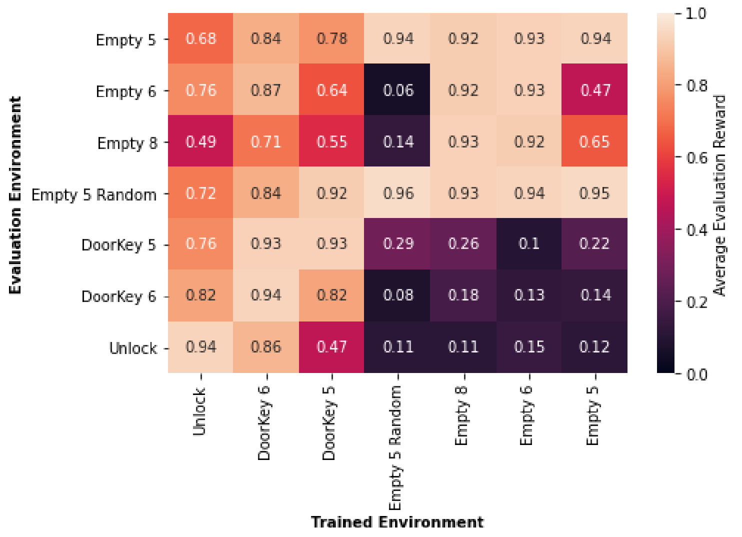

Figure 3: Heatmap of cross evaluation of trained agents on other environments. As expected, the evaluation rewards along the diagonal are the highest, which is where each agent is tested on the environment they were trained on. We also note the the `Empty`-trained agents do not perform well on `DoorKey` or `Unlock` environments.

Using this analysis, we chose the `Empty 5x5` and `Unlock` environments and the associated agents as an experimental subject. As demonstrated in Figure 3, while the respective agents perform well in their own environments, the mean return earned suffers when evaluated on one another. That is, `Empty 5x5` and `Unlock` earn an mean in-environment return of .94 whereas when evaluated on the other environment. On the contrary, when evaluating these agents on the other environment, the mean returns are .68 and .12, respectively. Thus, in Section 4.1 we use these pre-trained networks and environments to determine whether the network trained on `Empty` could be adapted to perform well on `Unlock` using side-tuning. Moreover, we compare these results to that of fine-tuning and examine the relevant sample complexity.

As a follow-on experiment in Section 4.3, we hypothesized as to whether side-tuning could allow for performance improvements within a given environment where PPO failed. That is, given an environment where a PPO agent was unable to converge (i.e. `DoorKey 8x8`), we aimed to see the performance of using a successful agent (i.e `DoorKey 6x6`) and subsequently observing the consequences of fine-tuning or side-tuning this agent on `8x8`.

4 Results

4.1 Impacts on Catastrophic Forgetting

4.1.1 Fine-Tuning

We take the agent trained on `Empty`, and fine-tune the network on the `Unlock` environment. Figure 4(a) shows the evaluation performance of this network on both the original `Empty` environment and the `Unlock` environment.

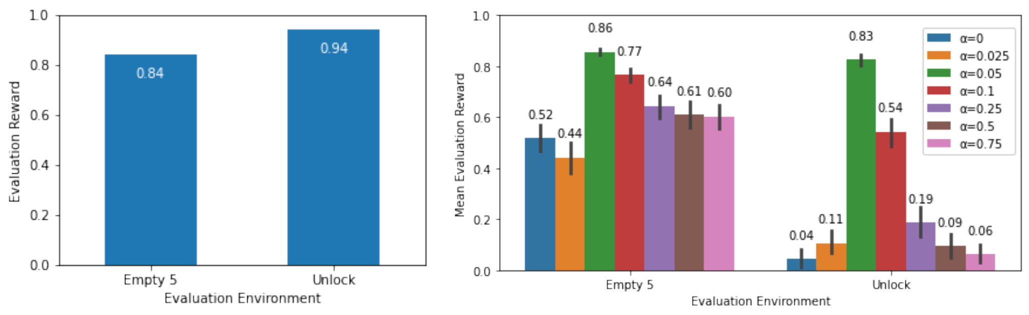

Figure 4: (a) Fine-tuning evaluation reward. Fine-tuning shows catastrophic forgetting in the decreased performance on `Empty` as compared to 0.94 when training from scratch. (b) Side-tuning evaluation reward at different α levels. $\alpha = 0.05$ performs the best and does not suffer from catastrophic loss. However, the `Unlock` performance isn't as good either.

We see that as expected the fine-tuned agent has learned to perform well on the new task (`Unlock`), but it actually decreases in performance on `Empty`, as compared to the results from Figure 3. Although the drop in performance is not extreme, this is still a sign of catastrophic forgetting from the original task (that is, the mean return dropped from .94 to .84). Next, we explore how side-tuning handles this same learning task.

4.1.2 α-Variation

Side-tuning is dependent on the alpha-blending parameter α, so we perform a manual parameter search across different α-levels before drawing conclusion on side-tuning performance. Ideally, α would be learned parameter, but we did not implement this learning in the scope of the project. Fig.4(b) shows the evaluation performance of `Empty`-trained agent after being side-tuned on the `Unlock` environment. The side-network for these results was `ConvNet-1`, which is further explained in Section 4.2.

We find that an α-parameter of 0.05 performs the best. Additionally, we note that the new agent achieves evaluation scores of .86 and .83 on environments `Empty` (original) and `Unlock` (new task), respectively. This shows that with the correct α, side-tuning has less catastrophic loss but worse new-task performance than fine-tuning.

4.2 Network-Variation

One potential goal of side-tuning is to enable to use of multiple lightweight side-networks for different tasks. Thus, we explore the performance of side-tuning with different side-network architectures.

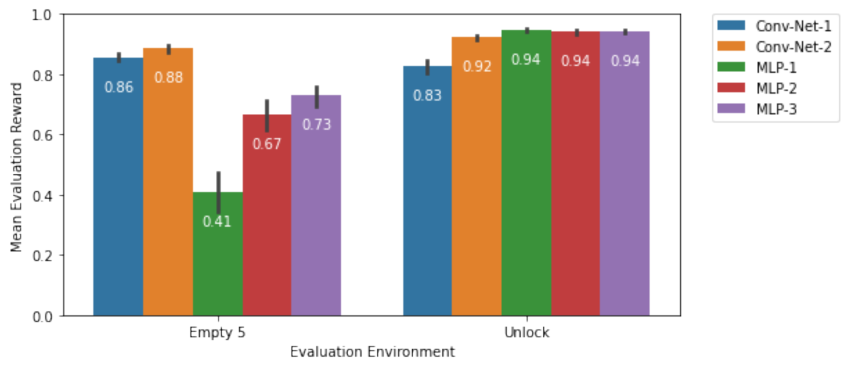

Figure 5: Evaluation reward of different side-tuning architectures. The `ConvNet` side networks perform the best overall compared to `MLP` side networks.

We use $\alpha = 0.05$ for each of these networks. Figure 5 shows the evaluation performance on `Empty` and `Unlock` environments.

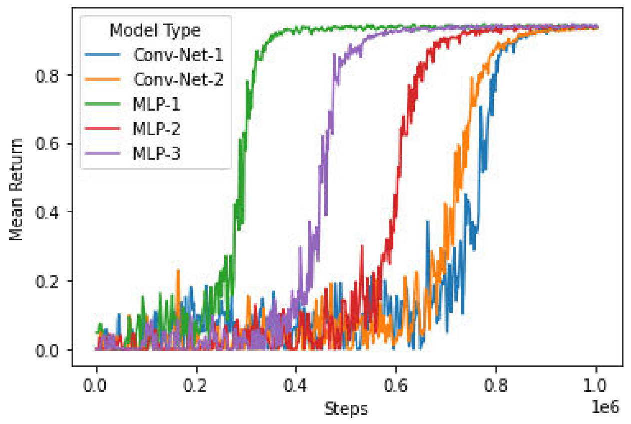

Figure 6: Network variation training curves. The `ConvNets` take much longer to converge due to network complexity, but this may also mean the `MLPs` maybe be over-simplistic as side-networks

The two `ConvNet` architectures, which are similar to the base-network perform the best on the original `Empty` environment while still performing almost as well as the `MLP` architectures on `Unlock` environment. This may be because the base network was a `ConvNet` architecture trained on `Empty`, such that as similar architecture will more easily handle the original task as well. However, another explanation could be the network complexity is larger with the `ConvNets`, so the side-network is simply learning more than the `MLPs` do. This may be noticeable from the training curves shown in Figure 6, which indicate that the `ConvNet` side-networks take significantly longer to converge. What's even more surprising the the side-networks take longer to converge than the original agent's took to learn the same `Unlock` environment from scratch in Figure 6. This is further discussed in Section 5.

4.3 Regime-Specific Performance Improvement

We briefly document the results of our experiment to test the hypothesis of whether side-tuning could be used to achieve convergence in environments where a PPO agent initially failed. Training was performed for 500k steps and with respect to side-tuning, the most optimal network described in 4.2 and $\alpha = .05$ were used.

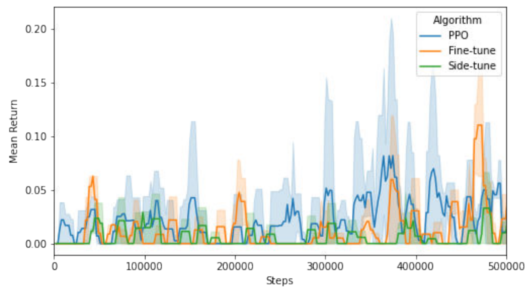

Figure 7: Training Curves for the a vanilla PPO agent on `DoorKey 8x8`, a fine-tuned `DoorKey 6x6` agent, and a side-tuned `DoorKey 6x6` agent

Here, as demonstrated , we observe that none of the methodologies produced an agent which was able to succeed and achieve convergence on `DoorKey 8x8`. This result has particularly interesting implications on the usefulness of side-tuning methods. That is, we hypothesize that if the benefits of side-tuning with respect to generalizing some network from environment A to B are only achieved when a vanilla PPO agent can succeed, then it gives rise to the notion of simply training different policy heads for each of the necessary environments. This corresponds to a more hierarchical approach whereby a supervisor network decides which of its child networks to pass the task at hand to in order to derive a policy. However, we note that further testing and experimentation with this idea can lead to more evidenced conclusions.

4.4 Sample Efficiency

Finally, we considered the implications of side-tuning on the sample complexity (i.e. the number of samples observed by the algorithm before convergence, where we defined a threshold of mean return > 0.9) of training when compared to a fine-tuned agent.

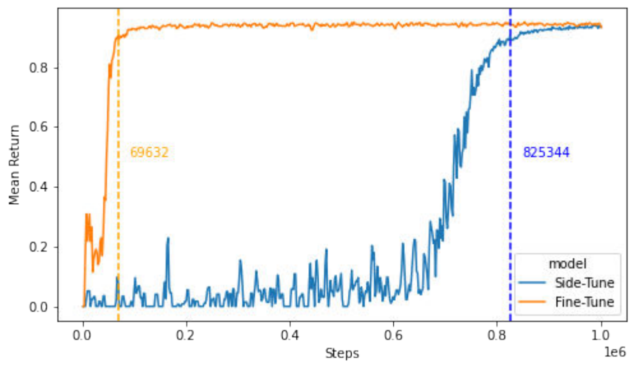

Figure 8: Side-tuning demonstrates much higher sample complexity in comparison to fine-tuning

As revealed by the above figure, the sample complexity of a fine-tuning approach is far less than that of side-tuning. Based on this result, we conclude that in contexts where there may not be a lot of data to guide network adaptation, then side-tuning may be an ineffective approach.

5 Conclusion

In this project, we determine the efficacy of side-tuning networks as a method for network adaptation. Using a variety of MiniGrid environments which contain different tasks, we perform experiments to understand the relative performance of side-tuning in comparison to fine-tuning.

Specifically, we found that the claim of Zhang et al. that side-tuned networks reduce catastrophic forgetting in the context of sequential learning to hold, although the improvement in our trials was only about 5% over fine-tuning. Thus, further experimentation with more complex task changes is needed. Additionally, further exploration of network architectures is warranted, given the surprisingly low performance of the `MLP` networks. Learning α through gradient descent rather than our naive parameter search α may resolve both these issues. Side-tuning did not provide any performance improvements in cases where fine-tuning and from-scratch training also both failed. Lastly, the sample complexity of side-tuning was unexpected, and more analysis is required to draw meaningful conclusions with that.

References

[1] Jeffrey O Zhang, Alexander Sax, Amir Zamir, Leonidas Guibas, and Jitendra Malik. Side-tuning: A baseline for network adaptation via additive side networks, 2020.

[2] Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization, 2020.

[3] Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning, 2016.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

6 Contributions

Both authors worked together in developing the idea and experimentation protocol. Isaac Perper focused more on determining how side-tuning decreases catastrophic forgetting and network variation. Suraj Srinivasan carried out the experiments relating to α-variation and regime-specific performance improvement. Both authors contributed equally to the production of this report.