Memory Augmented Self-Play

Shagun Sodhani [1] [*]   **Vardaan Pahuja** [1] [*]

Abstract

Self-play (Sukhbaatar et al., 2017) is an unsupervised training procedure which enables the reinforcement learning agents to explore the environment without requiring any external rewards. We augment the self-play setting by providing an external memory where the agent can store experience from the previous tasks. This enables the agent to come up with more diverse self-play tasks resulting in faster exploration of the environment. The agent pretrained in the memory augmented self-play setting easily outperforms the agent pretrained in no-memory self-play setting.

1. Introduction

In a typical reinforcement learning task, the training agent has to find trajectories that lead to high rewards while learning to interact with the environment. Learning a good policy could require a large number of episodes as the agent has no understanding of the environment and only a small fraction of agent's interactions are useful for updating its policy, making these approaches sample inefficient. (Sukhbaatar et al., 2017) presents an unsupervised training formulation, called as self-play, which enables the agent to explore and interact with its environment without using any external rewards. Two versions of the same agent, called as Alice and Bob, play a game where Alice proposes a task for Bob to perform. The task requires Bob to navigate through the environment and interact with different items thus enabling the agent to learn how to efficiently transition from one state to another. We use the word *self-play* to refer to the technique proposed by (Sukhbaatar et al., 2017).

We extend the *self-play* formulation and present a modified approach called as *memory augmented self-play*. The key idea is to enable Alice to remember the tasks that she has assigned to Bob previously and use this experience while proposing a new task for Bob. We evaluate our approach

on both discrete and continuous environments and present empirical evidence that the agents pre-trained with *memory augmented self-play* outperform the agents pre-trained with *self-play* for different tasks. We also show that *memory augmented self-play* enables Alice to explore more parts of the environment than just *self-play*.

2. Motivation

Self-play proposes a game between two variants of the same agent (referred to as Alice and Bob) where Alice assigns a task to Bob by performing some actions in the environment. The reward structure is such that the optimal behaviour for Alice is to come up with the simplest task that Bob can not finish and the optimal behaviour for Bob is to finish the given task as quickly as possible. Under the optimal setting, the series of tasks proposed by Alice would be such that each task is just slightly harder than the previous task. This would correspond to a curriculum for training Bob how to navigate through the environment and how to interact with the items in the environment. In literature, curriculum learning (Bengio et al., 2009) has been shown to boost the speed of convergence as well as enhance the generalization performance in the domain of supervised learning. In (Florensa et al., 2017), the agent learns to reach the goal from a set of start states increasingly far from the goal, which is another form of curriculum learning. The curriculum of start states depends on the performance of the agent at each step of the training process. Using this approach results in a significant performance gain in fine-grained robotic manipulation tasks.

While *self-play* has been demonstrated to be an effective technique for unsupervised pre-training of the learning agents, there is a scope for improvement in some aspects of its formulation. Even though Alice is required to come up with a curriculum of tasks for Bob, she has no means of remembering what tasks she has already assigned to Bob. The reward signal tells her how hard the current task was for Bob but she has no explicit means of remembering this. This implies that Alice could assign similar tasks to Bob multiple times resulting in wasteful self-play episodes. Intuitively, the policy gradient algorithm has to learn how to remember the history of the previous tasks and how to use this history for generating a new task trajectory.

---
[*]Equal contribution [1]MILA, Université de Montréal
.   Correspondence to: Shagun Sodhani <sshagun-soddhani@gmail.com>, Vardaan Pahuja <vardaan-pahuja@gmail.com>.

We propose to overcome this limitation by providing an explicit memory module to Alice. In the *self-play* setting, Alice is randomly initialized in the environment at the start of each episode and her actions are conditioned on her start state (for that episode) and her current state. The sequence of these actions generates the task for Bob. In the memory augmented setting, the actions that Alice take are conditioned on the memory as well as the start state and the current state in the episode. The memory is updated after every *self-play* episode. This disentangling of memory and policy gradients allows policy gradients to generate more diverse trajectories which, in turn, help the agent to train faster.

3. Model

We use REINFORCE (Williams, 1992) algorithm with baseline for all the experiments. We first describe the agent architecture for *self-play* and then we discuss the modifications made for *memory augmented self-play*. We use a feature extraction network which is a single-layer feedforward neural network. During an epsiode, at any step, the input to the network is a concatenated tuple of the current state in the current episode and the start state in the current episode (for Alice) or the current state in the current episode and target state in the current episode (for Bob). We refer to this tuple as the *current episodic tuple* as the input. The network produces a feature vector of size *feature-dim* using the *current episodic tuple*. This feature vector is fed as an input to both the actor and the critic networks. The actor and critic networks are also single-layer feed-forward networks. In case of actor, the feed-forward network is followed by the softmax operator and in case of critic, we obtain a real-valued, scalar output.

The architecture of the feature extractor, actor network and the critic network are the same for all the environments. We used ReLU (Glorot et al., 2011) as the activation function.

Table 1. Value of *feature-dim* for different settings

Environment	Alice (self-play)	Alice (Memory)	Bob
Mazebase	50	100	50
Acrobot	10	20	10

Table 1 summarizes the value of *feature-dim* for the different settings. In all the cases, the size of the softmax layer is given by the number of possible actions in the environment except in the case of self-play episodes for Alice where she has an additional "stop" action.

In the case of *memory augmented self-play*, we use the *current episodic tuple* as before to obtain the feature vectors. In the case of Alice, these features are concatenated with the features coming from the memory and the concatenated features are fed into the actor and the critic networks. Since

Bob does not use the external memory, the architecture for Bob remains exactly the same as before. Now we discuss how the memory module is implemented and updated.

We compared 3 different approaches to model the memory

1. Last Episode Memory - In this setting, we obtain the feature vector corresponding to the *last episodic tuple* and retain only that in the memory.

2. Last k Episode Memory - This setting generalizes the Last Episode Memory setting where we retain the feature vectors corresponding to the last k episodes and average them to obtain the memory features.

3. LSTM Episode Memory - In this setting, we pass the previous episodic feature vectors through a LSTM (Hochreiter & Schmidhuber, 1997) and the hidden state of the LSTM gives the feature vector corresponding to the memory.

Based on our experiments, we observed that LSTM Episode Memory performs better than the other two variants, so we used LSTM based memory for rest of the experiments.

4. Experiments

We use PyTorch (0.3.1)(Paszke et al., 2017) as the framework for our implementation. The code for this implementation is available at the link https://github.com/shagunsodhani/memory-augmented-self-play. We evaluate our proposed approach on Mazebase (Sukhbaatar et al., 2015) (grid world environment) and Acrobot. Mazebase is a discrete environment with discrete actions while Acrobot is a continuous environment with discrete actions. For running the Mazebase experiments, we used mazes of size (8×8). The OpenAI Gym (Brockman et al., 2016) version of Acrobot environment was used for our experiments. For Mazebase, different models were trained till the current episodic reward saturated and for Acrobot, the experiments were run for 50K episodes. For all the reward curves and tables, we report rewards averaged over last k episodes where k is set to be 10000 for Mazebase and 2000 for Acrobot. Using other values of k gives the same trend. The batch size for Mazebase was set to be 256 and that for Acrobot was 1 (using bigger batch size gives worse results). We used Adam (Kingma & Ba, 2014) optimizer (with learning rate of 0.001) and policy-gradient algorithm with baseline to train the agents. For self-play based experiments, we interleaved the training of self-play task and target task with one batch of self-play training followed by n batches of training with the target task. n is set to be 4 for Mazebase and 100 for Acrobot. For Mazebase, we set we set the maximum number of steps per episode, in the

target task, to be 50, while the maximum number of steps, in the *self-play* episode, is set to 80. In case of Acrobot task, we set the maximum number of steps per episode to 1000, while the maximum number of steps per *self-play* episode is set to 2000.

5. Results

5.1. Mazebase

Figure 1 shows how the trend in the value of current episodic reward (computed as a running average over last 10000 episodes) changes with the number of episodes for different tasks. We plot the different curves to the point of convergence and hence the three curves have different span along x-axis. The values of current episodic reward for some regularly sampled points are shown in Table 2. The key observation is that *memory augmented self-play* consistently performs better than both *self-play* and *no self-play* setting. Even though all the models converge towards the same reward value, using *memory augmented self-play* increases the speed of convergence. All the curves were plotted using the average value after running the models for 5 different seeds. Figures 4, 5, and 6 (in the appendix) show the corresponding plots individually, along with the band of standard deviation. We also observe that *self-play* (without memory) performs better than the no self-play setting. This was observed in the *self-play* paper and our experiments validate that observation.

5.2. Acrobot

Figure 2 shows the comparison of trend of current episodic reward (computed as a running average over last 2000 episodes) for the Acrobot task. All the curves were plotted using the average value after running the models for 3 seeds. Figures 7, 8 and 9 (in the appendix) show the trend of current episodic reward (averaged over last 2000 episodes) with the number of episodes for the Acrobot task in *memory augmented self-play*, *self-play* and *no self-play*, settings respectively. The values of average episodic reward for some regularly sampled points are shown in Table 3. For the Acrobot task, we observe that the memory augmented agent performs the best with the peak reward of -568.80. Even though both the *self-play* and *no self-play* variants perform poorly, the *self-play* version is comparatively better with the peak reward of -611.72, as compared to -798.21 obtained with *no self-play*.

One of our motivations behind adding memory was that having an explicit memory would allow Alice to remember what states she has already visited. This would enable her to explore more parts of the environment as compared to the *self-play* setting. To validate this hypothesis, we perform a simple analysis. We compile the list of all the start and

the end states that Alice encounters during training. Even though the start states are chosen randomly, the end states are determined by the actions she takes. We embed all the states into a 2-dimensional space using PCA and plot the line segments connecting the start and end states for each episode. The resulting plot is shown in Figure 3. We observe that using LSTM memory results in a wider distribution of end state points as compared to the case of *self-play* with no memory. The mean euclidean distance between start and end points (in PCA space) increases from 0.0192 (self-play without memory) to 0.1079 (*memory augmented self-play*), a $5\times$ improvement. This affirms our hypothesis that *memory augmented self-play* enables Alice to explore more parts of the environment and enables her to come up with more diverse set of tasks for Bob to train on.

Table 2. Comparison of trend of current episodic reward (averaged over last 10000 episodes) for different approaches on Mazebase Task

No. of episodes	No self-play	self-play	memory-augmented self-play
100K	-4.841	-4.650	-4.580
200K	-4.419	-4.253	-3.899
300K	-3.861	-3.686	-3.130
400K	-3.357	-3.265	-2.758
500K	-3.076	-2.942	-2.541
600K	-2.782	-2.802	-2.528
700K	-2.669	-2.564	-2.516

Table 3. Comparison of trend of current episodic reward (averaged over last 2000 episodes) for different approaches on Acrobot task

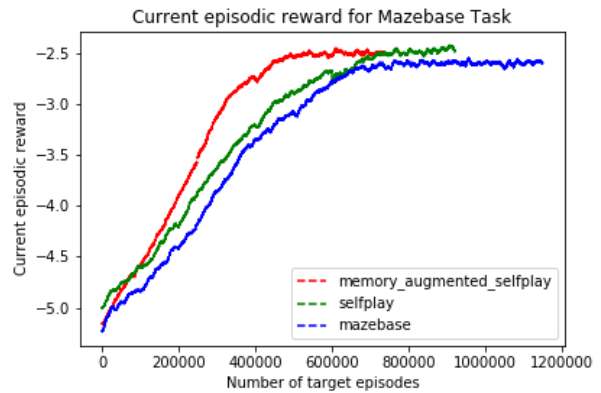| No. of episodes	No self-play	self-play	memory-augmented self-play
10K	-826.31	-778.74	-678.14
20K	-986.14	-712.65	-778.31
30K	-999.24	-949.59	-924.37
40K	-999.98	-999.79	-992.29
50K	-1000	-1000	-996.77

Figure 1. Figure showing the current episodic reward for the Mazebase Task using different strategies.

Figure 2. Figure showing the current episodic reward for the Acrobot Task using different strategies.
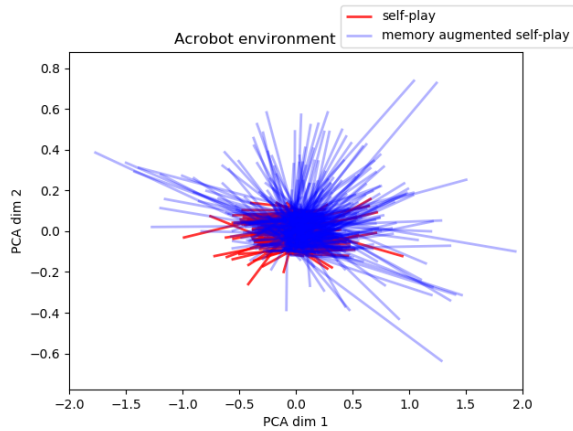

Figure 3. Plot of start and end states in 2D with and without memory augmentation

5.3. Conclusion

We propose to augment the *self-play* setting by providing a memory module to Alice so that she can remember the tasks she has assigned to Bob so far. We perform experiments on both discrete and continuous environments and provide empirical evidence that using *memory augmented self-play* enables the agent to come up with more diverse self-play tasks. This leads to faster exploration of the environment and the agent pretrained in the *memory augmented self-play* takes lesser time to train.

There are several interesting directions to extend this work. We have limited our experiments to use of a relatively simpler memory module. Further, we store just the start and the end state of each episode. A straight forward extension would be the have a hierarchical memory model where one level of memory is used to persist states across an episode and the second level of memory is used to maintain informa-

tion across episodes. Another interesting extension would be to provide Alice with a write-able differential memory. This would allow her to choose states that she wants to persist across episodes and may lead to a more efficient utilization of the memory.

References

Bengio, Yoshua, Louradour, Jérôme, Collobert, Ronan, and Weston, Jason. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48. ACM, 2009.

Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. Reverse Curriculum Generation for Reinforcement Learning. *ArXiv e-prints*, July 2017.

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

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Paszke, Adam, Gross, Sam, Chintala, Soumith, and Chanan, Gregory. Pytorch, 2017.

Sukhbaatar, S., Szlam, A., Synnaeve, G., Chintala, S., and Fergus, R. MazeBase: A Sandbox for Learning from Games. *ArXiv e-prints*, November 2015.

Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., and Fergus, R. Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. *ArXiv e-prints*, March 2017.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pp. 5–32. Springer, 1992.
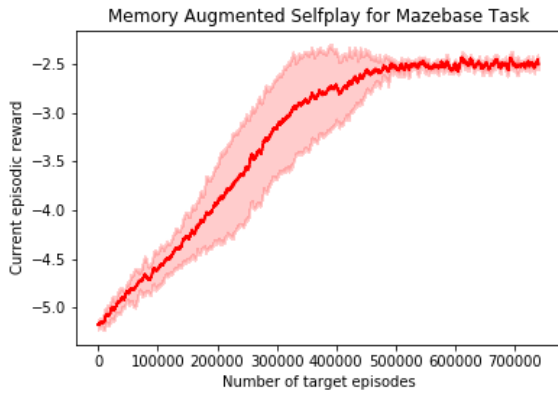
6. Appendix

Figure 4. Figure showing the current episodic reward for the Mazebase Task using Memory Augmented SelfPlay.
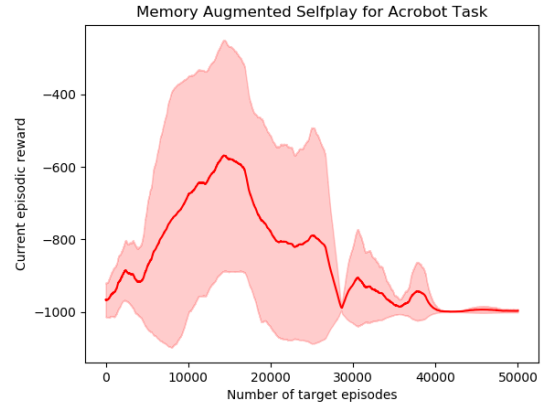

Figure 7. Figure showing the current episodic reward for the Acrobot Task using Memory Augmented SelfPlay.
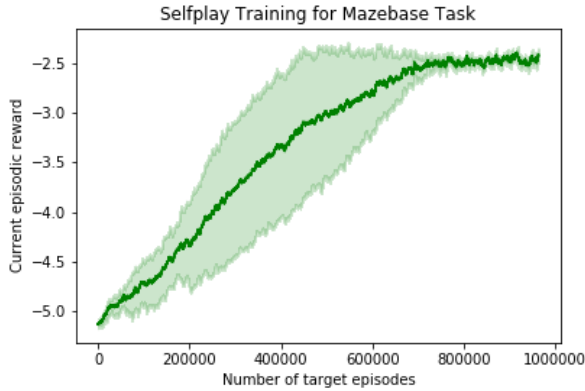

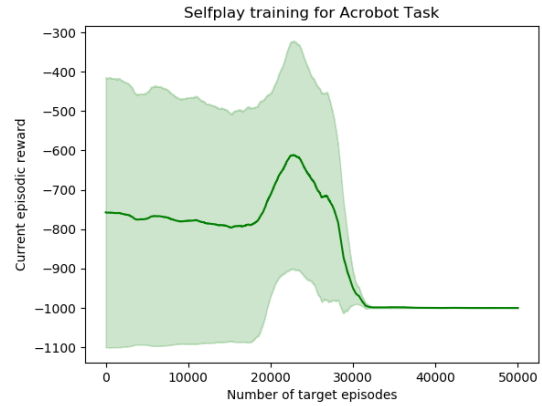Figure 5. Figure showing the current episodic reward for the Mazebase Task using SelfPlay(no memory).

Figure 8. Figure showing the current episodic reward for the Acrobot Task using SelfPlay(no memory).
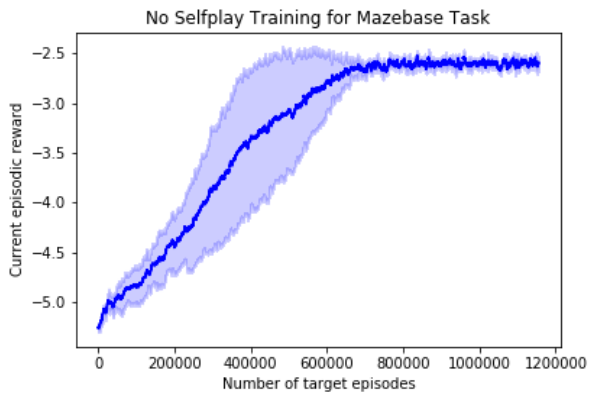

Figure 6. Figure showing the current episodic reward for the Mazebase Task without using SelfPlay.
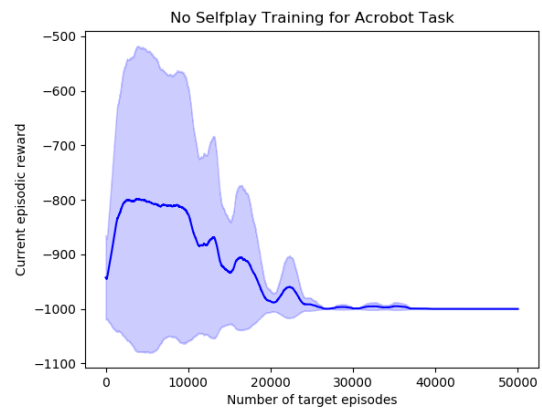

Figure 9. Figure showing the current episodic reward for the Acrobot Task without using SelfPlay.