Memory Augmented Self-Play

Shagun Sodhani (McGill ID: 260844972, UdeM ID:20111009)
sshagunsodhani@gmail.com

Vardaan Pahuja (McGill ID: 260844959, UdeM ID:20081093)
vardaanpahuja@gmail.com
MILA, Département d'informatique et de recherche opérationnelle
Université de Montréal

Abstract

Self-play [7] is an unsupervised training procedure which enables the reinforcement learning agents to explore the environment without needing any external rewards. We augment the self-play setting by providing an external memory which 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 is able to outperform the agent pretrained in self-play setting.

1 Introduction

In a typical reinforcement learning problem, the training agent has to find trajectories that lead to high rewards while it learns 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. [7] presents an unsupervised training formulation which enables the agent to explore and learn about its environment without using any external rewards. Two versions of the same agent, called as Alice and Bob, are put against each other in a game where Alice proposes a task for Bob to complete and Bob attempts to complete the task. This enables the agents to learn how to efficiently navigate from one state to another. We use the word "self-play paper" to refer to [7] unless specified otherwise.

We extend the formulation of the self-play paper and present a modified approach called as memory augmented self-play. We enable Alice to remember the tasks that she had assigned to Bob previously and use this experience while proposing a task for Bob. We present empirical evidence that agents pre-trained with memory augmented self-play outperforms agents pre-trained with self-play in both Mazebase and Acrobot 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 paper proposes a game between two variants of the same agent (referred to as Alice and Bob) where Alice proposes a task for Bob by taking some actions in the environment. The reward structure is such that Alice's optimal behaviour is to find the simplest task that Bob can not finish and Bob's optimal behaviour is to finish the given task in as less time as possible. In the optimal setting, the tasks proposed by Alice would correspond to a curriculum for training Bob where each new task is just slightly harder than the previous task.

[0]DISCLAIMER: This work is submitted as a course project and not submitted to NIPS'17. Please see Section 7 for details on individual contribution of group members.

One limitation of the self-play formulation is that Alice does not remember 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. 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. This implies that Alice could assign similar tasks to Bob multiple times resulting in wasteful self-play episodes.

We propose to overcome this limitation by providing an explicit memory module to Alice. The new setting is as follows: Alice is randomly initialized in the environment and the task trajectory she generates is conditioned on the memory as well as her starting state and current state. The memory is updated after every self-play episode. This disentangling of memory and policy gradients allows policy gradients to generate more diverse trajectories.

3 Environment Description

We describe the different environments we consider for our experiments.

3.1 Mazebase

Mazebase is a simple 2D game environment introduced in [8]. The environment comprises of 10 different games to train agents on reasoning and planning tasks. The games are played on a 2D rectangular grid with different items scattered across the board. The items include blocks, water, switch, door, push-able blocks etc. The tasks are formulated such that the agent needs to move around the grid and interact with these items to achieve the end goal. The interactions result in different type of intermediate rewards (and penalties) and possibly changes the environment as well. For example, the agent may *press* a switch or *push* a push-able block. The agent is free to move in any of the four directions provided there is no item blocking its path.

For our experiments, we use the **Light Key** game. The setting of the game is as follows: The rectangular grid has a switch and a door (along with other elements like blocks and water). The goal state may or may not be on the same side of the door as the agent. If it is on the same side, the agent should directly navigate to the goal position. In the latter case, it must first reach the switch, perform the *toggle* action on the switch to open the door and then navigate to the goal state. The agent receives a fixed penalty of 0.1 for every action that it takes. It receives an additional penalty of 0.2 if it steps into water. The game ends when the agent reaches the goal state and is awarded a reward of 1.0. This task tests the agent's ability to use if-then reasoning while interacting with the environment.

The environment allows for controlling the dimensions (height and width) of the rectangular grid and the percentage of wall blocks and water blocks. This allows for developing a curriculum based training strategy of the agent. One of the motivations behind the use of self-play is that Alice and Bob would come up with a suitable curriculum for exploring the environment. We did not explicitly generate the mazes to follow a curriculum and used the same setting of these parameters across all the tasks. We always used grids of size 8×8, and the percentage of wall blocks and water blocks was left at their default values. The environment does not impose any limit on the number of permissible steps. The Mazebase paper [8] allowed a maximum of 50 steps and we follow this time limit for the case when the agent is learning on the target task (finding the goal position). In the case of Self-Play task, the number of maximum number of combined steps (t_{max}) is set to 80 as suggested in [7].

3.2 Mountain Car

In this environment, the goal is to drive up to the top of the mountain. However, the momentum of the car is insufficient to reach the goal. The agent must learn to drive back, and build the momentum. The self-play paper uses the Continuous Mountain Car (RLLab) [2] and discretizes its action space into 5 bins. We used REINFORCE [9] with baseline to learn the policy, but the agent doesn't reach the goal in given time, even after training for 100K episodes with the hyper-parameter setting used in the self-play paper. This observation is also mentioned in the self-play paper. We also tried using the Mountain Car (discrete) environment of OpenAI Gym [1] but the agent doesn't converge with policy gradient even in this case. A reward of +1 is given only when the car succeeds in climbing the hill. In case of self-play, we assume that Bob succeeds if $||s_b - s_a|| < 5e-4$, where s_a and s_b are the final states (location and velocity of the car) of Alice and Bob respectively.

3.3 Acrobot

The acrobot system includes two joints and two links, where the joint between the two links is actuated. It is analogous to a gymnast swinging on a high-bar. The goal is to swing the endpoint above the bar by an amount equal to one of the links. As in the mountain-car task, there are three actions, positive torque, negative torque, and no torque, and reward is -1 on all steps, except the time-stamp when goal state is reached. The self-play paper does not consider Acrobot in their experiments, but we additionally consider this environment for our experiments. We modify the number of max. episodes for Acrobot from 500 to 1000, as the agent was rarely able to solve the task in 500 episodes. The value of t_{max} was set to 2000 in this case.

4 Model

We use REINFORCE [9] algorithm with baseline for all the environments. We first describe the architecture of the agents for self-play and then we discuss how this architecture changes for the case of memory augmented self-play. We have a feature extraction network which is a single-layer feed-forward neural network. The input to the network is concatenated tuple of the current state in the current episode and the start state in the current episode (for Alice) or current state in the current episode and target state in the current episode (for Bob). We refer to this as the current episodic tuple. 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, we follow it with the softmax operation and in case of critic, we obtain a scalar, real-valued output.

The architecture of the feature extractor, actor network and the critic network are the same for all the environments. We used ReLU [3] as non-linearity in our feature extraction network.

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. The only exception to this is the case of self-play for Alice where she has one additional "stop" action.

In case of memory augmented self-play, we have the current episodic tuple as before. We fed it to the feature extraction network to obtain the feature vectors. In 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. The case of Bob remains exactly the same as the case of self-play without memory. Now we discuss how the memory module is implemented and updated.

We compared 3 different approaches to model the memory

1. Most Recent - In this setting, we obtain the feature vector corresponding to the last episodic tuple and retain only that in the memory.
2. Last k Recent - This setting generalizes the Most Recent setting where we retain the feature vectors corresponding to the last k episodes and average them to obtain the memory features.
3. LSTM - In this setting, we pass the previous episodic feature vectors through a LSTM [4] and the hidden state of the LSTM gives the feature vector corresponding to the memory.

Based on our experiments, we observed that LSTM based memory model is better than the Most Recent and Last k Recent based approaches. Hence we used LSTM based memory for rest of the experiments.

5 Experiments

We use PyTorch [6] as framework for our implementation. For running the Mazebase experiments, we used mazes of size (8×8). Acrobot and Mountain Car environments were used from OpenAI Gym.

The number of episodes run for Mazebase, Acrobot and Mountain Car were 300K, 30K and 30K respectively. For all the reward curves and tables, we report rewards averaged over last k episodes where k is set to be 5000 for Mazebase and 1000 for both Acrobot and Mountain Car. Using other values of k gives the same trend. The batch size for Mazebase was set to be 256 and that for Acrobot and Mountain Car was 1 (using bigger batch size gives worse results). We used Adam [5] 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 both Acrobot and Mountain Car.

The self-play paper had modified the Mountain Car environment (in terms of reward structure and action space). Where ever possible, we made reasonable assumptions to perform the experiments. With these assumptions, the agent did not learn to perform the task in the Mountain Car setting. Instead, we reported the performance on the Acrobot task (which was not considered by the self-play paper). The self-play percentage used for Mountain Car in the self-play paper is 1%, which might be insufficient to see any significant performance boost with self-play.

The values of average episodic reward for some regularly sampled points are shown in Tables 3, 4 and 5.

5.1 Results

For the mazebase task, the trend of average reward in target task with the number of episodes is shown in Figure 1. The key observation is that memory augmented self-play is the clear winner, outperforming both self-play and no self-play setting. Moreover, the trend in the curves suggests that the gap between the performance of memory augmented self-play and self-play rises steeply with the number of episodes, while the gap between self-play and no self-play remains near constant. This suggests that using memory provides additional benefits over and above using self-play. We also observe that self-play out-performs the no self-play setting. This was observed in the the self-play paper and our experiments validate that observation.

For the Acrobot task, we observe that the memory augmented agent performs the best. Even though both the self-play and no self-play variants perform poorly, the no-selfplay version is slightly better. We expect that augmenting Alice with memory would enable her to explore more parts of the environment as compared to the self-play setting. To validate this hypothesis, we track the list of all the start and end state that Alice encounters during training. Even though the start states are chosen randomly, the end states are determined by policy gradient. We embed all the states into a 2-dimensional space using PCA as shown in Figure 2 and plot the line segments connecting the start and the end states for each episode. 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 (self-play with memory augmentation), which is a $5\times$ increase. This affirms our hypothesis that memory augmented self-play enables Alice to explore the environment more. It also indicates that memory helps in evolving a curriculum where Alice can learn to progressively give more difficult tasks to Bob.

Table 2: Comparison of different approaches on mazebase task

Setting	**Average Reward (after 100K steps)**	**Average Reward (after 200K steps)**	**Average Reward (after 300K steps)**
no self-play	-4.7106	- 4.1786	- 3.1484
self-play	-4.6475	-4.1003	-3.080
memory-augmented self-play	-4.5449	-3.65448	-2.7444

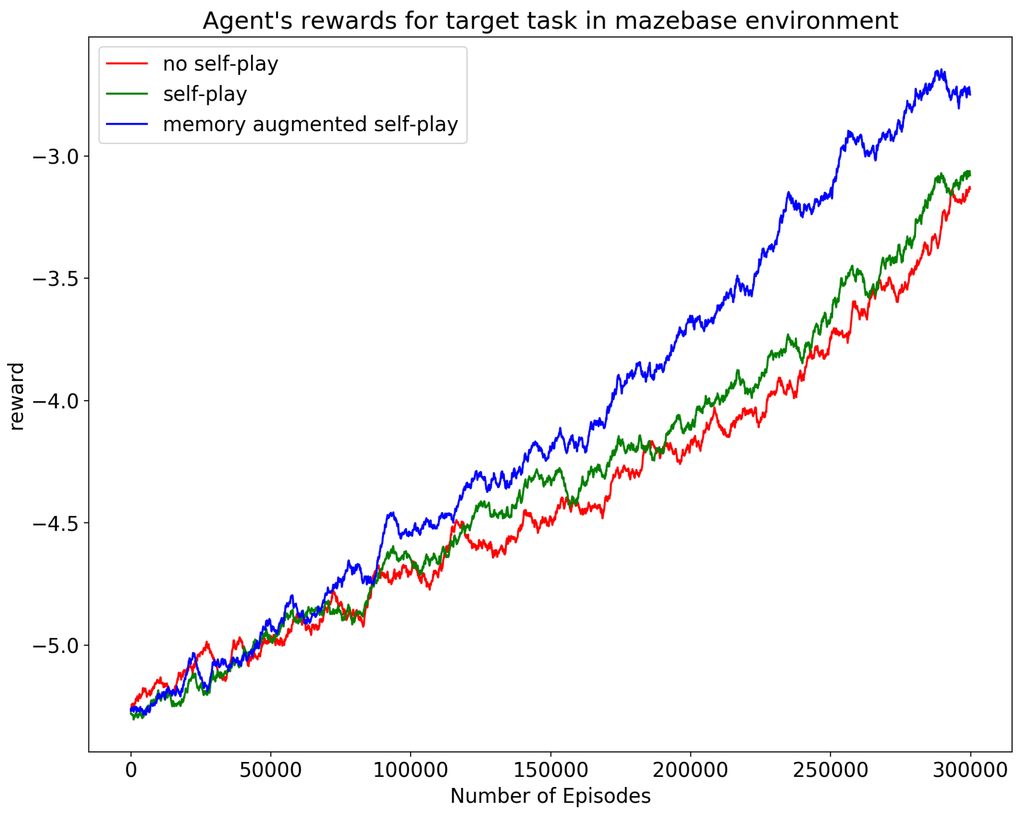

Figure 1: Average Episodic Rewards for Mazebase (no self-play)

Table 3: Comparison of different approaches on Acrobot task

Setting	**Average Reward (after 5K steps)**	**Average Reward (after 10K steps)**	**Average Reward (after 20K steps)**	**Average Reward (after 30K steps)**
no self-play	-998.62	-995.29	-1000.0	-994.52
self-play	-999.92	-1000.0	-1000.0	-1000.0
memory-augmented self-play	-920.86	-997.52	-897.05	-970.04

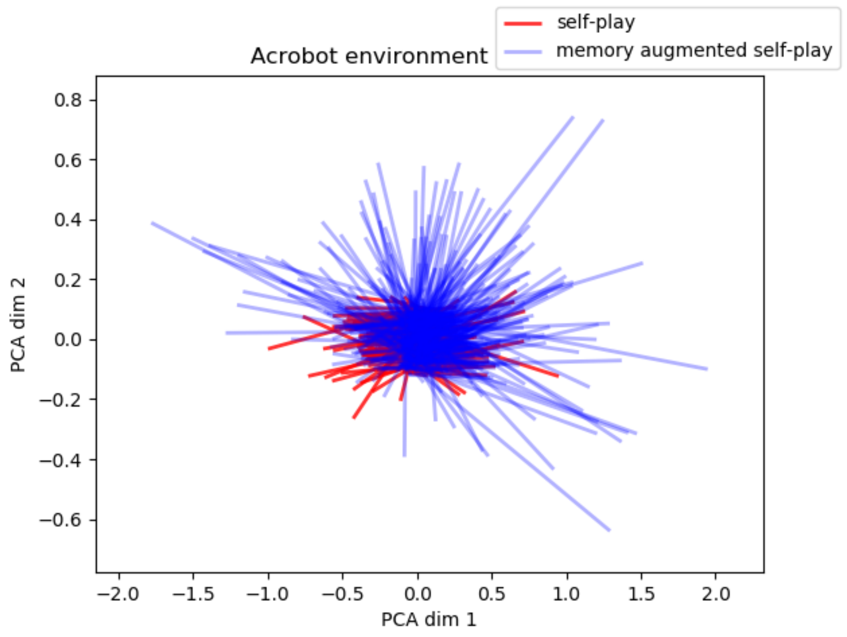

Figure 2: Plot of start and end states in 2D with and without memory augmentation

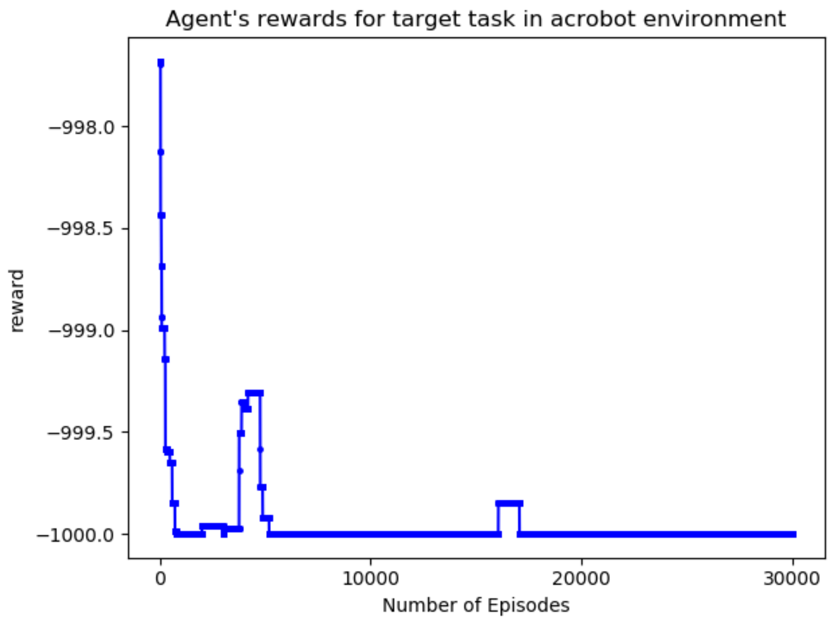

Figure 3: Average Episodic Rewards for Acrobot (self-play)

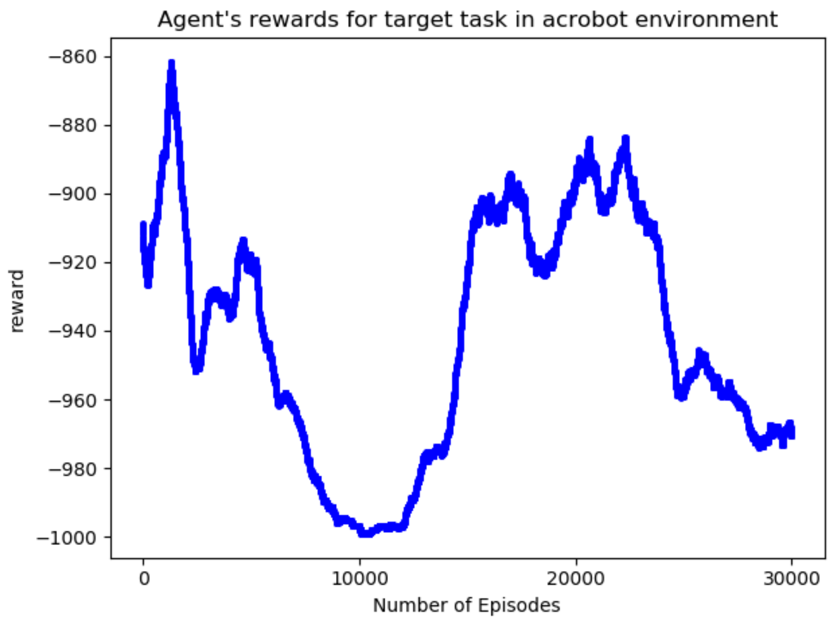

Figure 4: Average Episodic Rewards for Acrobot (memory augmented self-play)

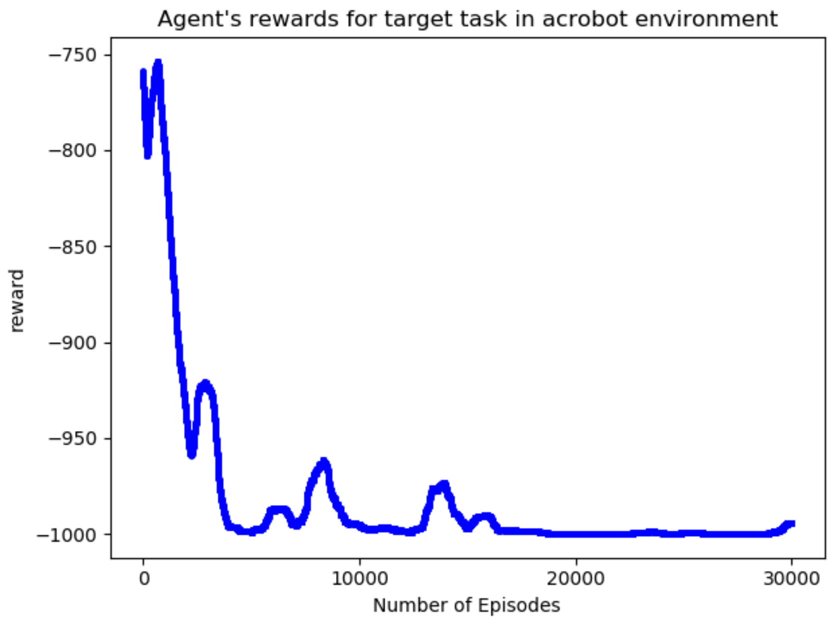

Figure 5: Average Episodic Rewards for Acrobot (no self-play)

6 Gaps in the paper

6.1 Difficulties

6.2 Size of Mazebase Environment

Mazebase supports environments of size ranging from (3×3) to (10×10). The self-play paper does not specify what size is used to train or evaluate the the agent or in what proportion are the mazes of different sizes used. We perform all the experiments on mazes of size (8×8).

6.2.1 Use of Mountain Car in RLLab

The self-play paper usew the continuous Mountain Car environment from RLLab. But the reward structure which they follow is to give zero reward at all time-stamps except at the goal state where the reward is +1. This differs from the reward structure inherently present in RLLab Mountain Car. We tried to learn the policy on this environment by discretizing the action space and using either reward structure, but found no success as the agent hardly learns anything. Instead, we report the performance on the Acrobot task (which is not considered by the self-play paper).

6.2.2 Limiting the number of steps that Alice can take during self-play

The original self-play formulation allows Alice to decide how many actions she wants to perform during self-play. We observed that in the initial phase of self-play training, Alice would use all the permitted time t_{max} and would not give Bob any chance to make a move. This situation limits Bob's ability to explore the environment. To avoid this situation, we restrict the number of time-stamps Alice is allowed to play to $(t_{max} - 1)$ thereby forcing Alice to give at least one move to Bob.

6.2.3 Adjusting the goal condition for Mountain Car during self-play

The self-play paper mentions that in the case of self-play, Bob succeeds if $||s_b - s_a|| < 0.2$, where s_a and s_b are the final states (location and velocity of the car) of Alice and Bob respectively. We noticed that this goal condition was achieved trivially (without Bob having to take any steps at all). We instead used $||s_b - s_a|| < 5e - 4$ as the termination condition.

6.2.4 Optimizer

The self-play paper proposed the use of RMSProp with a learning rate of 0.003. In our initial experiments, we found that Adam optimizer (with learning rate of 0.001) was outperforming RMSProp for the Mazebase environment and hence we used Adam for all our experiments.

6.3 Assumptions

1. For the task of Acrobot, we assume that Bob succeeds if $||s_b - s_a|| < 0.1$, where s_a and s_b are the final states of Alice and Bob respectively.
2. Due to lack of massive computational resources, we were able to run our experiments only for a limited number of episodes which is an order of magnitude less than the ones reported in the self-play paper.

6.4 Conclusion and Future Work

Based on the experimental observations, we make the following conclusions:

1. self-play is an effective strategy for the unsupervised training of the learning agent. The agent learns to navigate between different states and can re-use this learning in the same environment for different tasks.
2. Our proposal to augment the agent (Alice) with memory empirically helps the agent to out perform self-play trained agent on the target task.
3. Within different memory formulations, use of LSTM outperforms the other memory baselines.

There are several interesting directions to extend this work. We have limited our experiments to only "repeat" tasks for now. We would like to explore the "reset" and other, more complex tasks. It would also be interesting to explore how Alice uses the memory module - more specifically which episodes does she decide to drop from the memory. This would be helpful to understand if the hypothesis that Alice builds a curriculum for Bob actually holds and if it does, can we extract that curriculum. The memory module that we have used is quite standard and could eventually become a bottle-neck. It would be interesting to investigate if we could provide Alice with write-able differential memory. This would allow her to choose states that she wants to persist across episodes unlike the current setting where only the start and the end states are persisted across episodes.

6.5 References

1. The code for Acrobot and Mountain Car environments was originally sourced from OpenAI Gym [1] github repository and modified suitably for compatibility with our interface.

7 Contribution of each member

We thank the authors of Facebook for making the code of Mazebase open-source, which helped us in reproducing the experiments for our project. Vardaan Pahuja implemented the interface for Acrobot and Mountain Car task and the LSTM memory code. Shagun implemented the different self-play modes (memory-augmented self-play, vanilla self-play), policy gradient algorithms (REINFORCE with baseline), interface for the mazebase environment and other training infrastructure). The code is available at https://github.com/rllabmcgill/final-project-memory-augmented-self-play

References

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

[2] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. *ArXiv e-prints*, April 2016.

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

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

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

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

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

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

[9] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.

8 Appendix

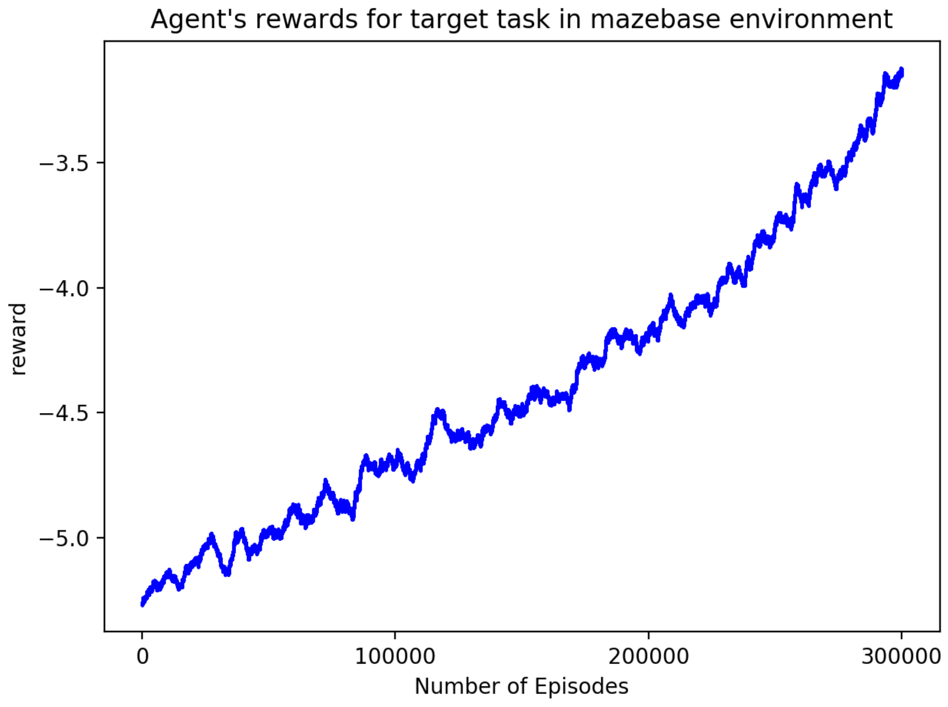

Figure 6: Average Episodic Rewards for Mazebase (no self-play)

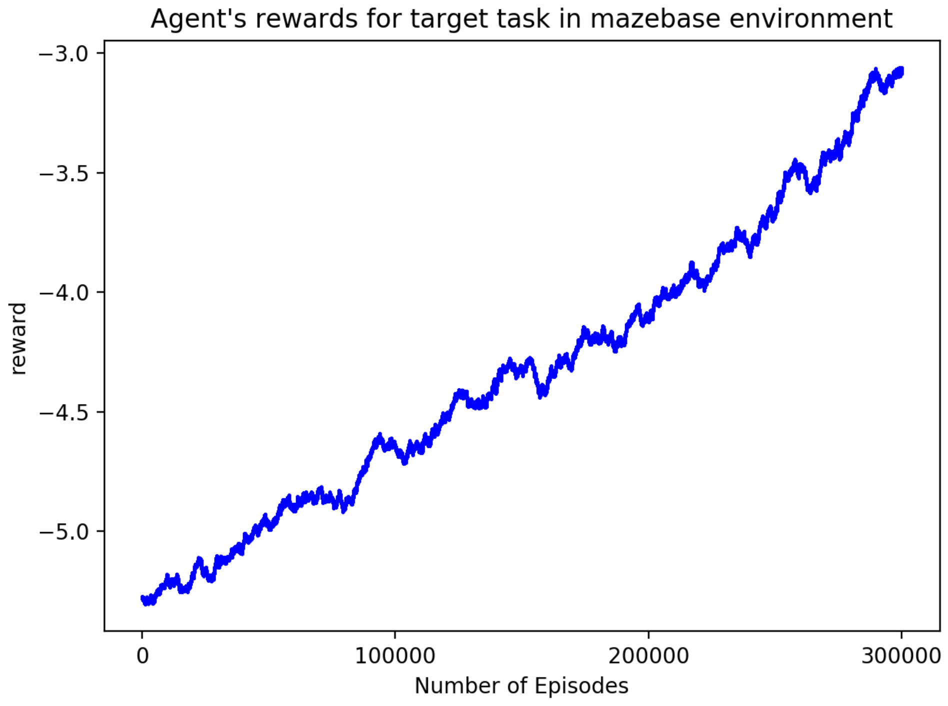

Figure 7: Average Episodic Rewards for Mazebase (self-play)

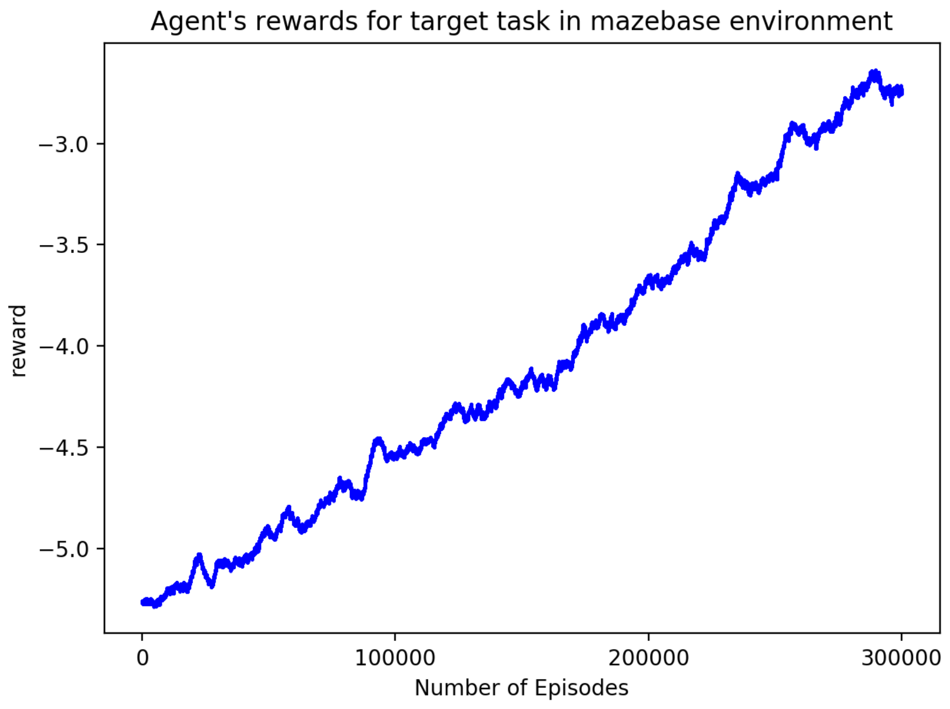

Figure 8: Average Episodic Rewards for Mazebase (memory augmented self-play)