CONTINUAL AND MULTI-TASK REINFORCEMENT LEARNING WITH SHARED EPISODIC MEMORY

Artyom Y. Sorokin
Moscow Institute of Physics and Technology
Dolgoprudny, Russia
griver29@gmail.com

Mikhail S. Burtsev
Moscow Institute of Physics and Technology
Dolgoprudny, Russia
burcev.ms@mipt.ru

ABSTRACT

Episodic memory plays an important role in the behavior of animals and humans. It allows the accumulation of information about current state of the environment in a task-agnostic way. This episodic representation can be later accessed by downstream tasks in order to make their execution more efficient. In this work, we introduce the neural architecture with shared episodic memory (SEM) for learning and the sequential execution of multiple tasks. We explicitly split the encoding of episodic memory and task-specific memory into separate recurrent sub-networks. An agent augmented with SEM was able to effectively reuse episodic knowledge collected during other tasks to improve its policy on a current task in the Taxi problem. Repeated use of episodic representation in continual learning experiments facilitated acquisition of novel skills in the same environment.

1 INTRODUCTION

Humans and other animals use episodic memory to adapt quickly in complex environments (Kumaran et al., 2016; Lake et al., 2017; McClelland et al., 1995). A striking feature of animal behaviour is ability to achieve several different goals in the same environment. Both of these features of adaptive behavior are being actively studied in the field of reinforcement learning (Rusu et al., 2015; Chaplot et al., 2017; Teh et al., 2017; Oh et al., 2016; Parisotto & Salakhutdinov, 2017; Santoro et al., 2018; Schwarz et al., 2018). However, the majority of the studies consider them in isolation, focusing either on episodic memory (Blundell et al., 2016; Pritzel et al., 2017; Santoro et al., 2018) or on learning several policies for achieving different goals (Frans et al., 2017; Dosovitskiy & Koltun, 2016; Tessler et al., 2017; Schwarz et al., 2018). Yet, the content of the episodic memory can be useful not only for a single task, but for completing multiple consecutive tasks, in addition to the general acquisition of new skills. For example, one can imagine a robotic home assistant instructed to retrieve a certain object. If the robot has encountered this object during a past house cleaning and recalls it, then this memory can greatly facilitate locating the requested object.

In this work, we propose a deep neural architecture able to store the episodic representation of an environment to improve the solution of multi-task problems and facilitate continual learning of new skills.

2 RELATED WORK

One of the most popular general approaches to multi-task learning is to train an agent on all tasks simultaneously. This method is called batch multi-task learning (Chen & Liu, 2016). In the field of deep reinforcement learning, it is often coupled with the weight sharing technique. In that case sub-task networks share part of their layers and the representation of the agent's current task is often fed in as additional input to the network (Florensa et al., 2017; Dosovitskiy & Koltun, 2016; Kaplan et al., 2017). Weight sharing allows to generalize experience over tasks and facilitates the learning of individual tasks (Taylor & Stone, 2011). A remarkable extension of this approach is the representation of sub-tasks with a descriptive system (Denil et al., 2017). Previous work on neuroevolution in a multi-task stochastic environment (Lakhman & Burtsev, 2013) demonstrated that agents evolve representation for episodic memory and use it in behavior. Several studies have been done on the

mapping of natural language task descriptions into sequences of actions (Chaplot et al., 2017; Misra et al., 2017). Another work in that area (Kaplan et al., 2017) used a sequence of instructions to guide the agent in Montezuma's Revenge game. However, the majority of current research in the field is focused on the isolated execution of sub-tasks, ignoring the transfer of episodic memory between sub-tasks.

In the past couple of years, numerous works have been completed on adding memory to deep RL architectures (Mnih et al., 2016; Hausknecht & Stone, 2015). One direction of research is to improve the agent's ability to store relevant memory about the state of environment (Parisotto & Salakhutdinov, 2017; Oh et al., 2016; Santoro et al., 2018). Alternatively, memories about recent state transitions can be used to facilitate rapid learning (Blundell et al., 2016; Pritzel et al., 2017).

It has been demonstrated that recurrent neural networks (RNNs) are capable of meta-learning (Thrun & Pratt, 1998; Santoro et al., 2016). Meta-learning in this case typically refers to the interaction of two learning processes. Slow adaptation when the weights of the neural network gradually learn persistent regularities in the environment. And fast dynamics of the recurrent network to adapt for rapid changes in the environment. Recently, this approach was extended to the RL setting Wang et al. (2016); Duan et al. (2016). In another work (Peng et al., 2018), a similar training technique helped to transfer a policy learned in simulation to a physical robot. While current focus of meta-RL with recurrent architectures is mainly on the adaptation of one policy to different variations of the environment we will consider a joint adaptation of several goal-oriented policies via shared memory.

3 Shared Episodic Memory for Multi-Task Reinforcement Learning

In this work, we use two ideas to facilitate the transfer of useful episodic representation between multiple sub-task policies. The first is an introduction of two separate recurrent sub-networks (1) for the environment and (2) for task-specific memories. The second is to use meta-learning setting (Duan et al., 2016; Wang et al., 2016; Frans et al., 2017) to optimize the agent over a series of tasks in the same environment.

Traditionally in multi-task reinforcement learning setting, a new task is selected at the beginning of each episode so that one episode corresponds to one task. An agent then simultaneously interacts with several instances of the environment and updates policy using samples collected for different tasks. This procedure is ineffective in storing a representation for more than one task. To make episodic memory useful we train a multi-task agent in a setting similar to the one used in meta-RL (Wang et al., 2016; Duan et al., 2016).

In our study, training consisted of episodes lasting T steps. For every episode, the environment was modified to some extent, i.e. locations of walls, targets, or objects. At the beginning of an episode, a task was randomly selected. If the task was completed by the agent, a new task was activated, but the state of the environment remained the same. Upon task completion, the agent received a reward and a "completion" signal shared among all tasks. The agent optimized the cumulative reward for all T steps.

Thus, the more tasks an agent completed in the available time budge more reward it received. This training mode encourages the agent's neural network not only to learn suitable policies for tasks but also to share between them a memory about the state of the environment.

To train the agent, we used the Parallel Advantage Actor Critic (*A2C*) algorithm (Mnih et al., 2016; Clemente et al., 2017). Our proposed network architecture with *shared episodic memory* (*SEM-A2C*) is presented in Figure 1. Instead of LSTM layer of the recurrent A2C, we introduce separate memory sub-networks for the environment state and task. At each step t, the network receives the current observation o_t, and the task identifier g_t. Observation o_t is processed by the observation encoder E^{obs}, and identifier g_t by task embedder E^{task}. In our experiments, E^{obs} is a two-layered convolutional network, and E^{task} is an embedding matrix that stores trainable task embeddings in its rows.

The core of the proposed architecture consists of RNN^{sem} and RNN^{tsm} recurrent sub-networks. RNN^{sem} takes observation embedding $E^{obs}(o_t)$ and returns its hidden state h_t^{sem}, which is reset to zero values only at the end of each episode. RNN^{tsm} takes the same input as RNN^{sem} as well

as h_t^{sem} and task embedding v_{g_t}. Unlike RNN^{sem} the hidden state h_t^{tsm} of the RNN^{tsm} is reset after the completion of the current task. The idea is that RNN^{sem} is responsible for capturing and storing a task-agnostic representation of the environment state, and RNN^{tsm} encodes a task specific representation. In contrast, RNN^{sem} has no knowledge of the current task, but is continuously updated over a longer period, which would correspond to several tasks in the batch multi-task setting.

In our experiments, we use a single LSTM layer (Hochreiter & Schmidhuber, 1997) for RNN^{sem}. For RNN^{tsm}, we use a factorized LSTM layer (F-LSTM) to generate weights of RNN^{tsm} on the fly with the task embedding vector v_{g_t} and the multiplication of three smaller matrices (see suppl. A.1).

Outputs of RNN^{tsm} and RNN^{sem} are concatenated and fed to three separate heads implemented as fully-connected layers. The first two are standard actor-critic heads where F^{val} predicts the state value function and F^{pol} generates the probabilities of the actions. The last head F^{comp} predicts the probability of task completion (see suppl. A.2).

Parameters are updated in SEM-A2C in the same way as in A2C (Clemente et al., 2017) and A3C (Mnih et al., 2016) algorithms. To learn the task completion prediction $\hat{d}_t$ we use cross-entropy loss.

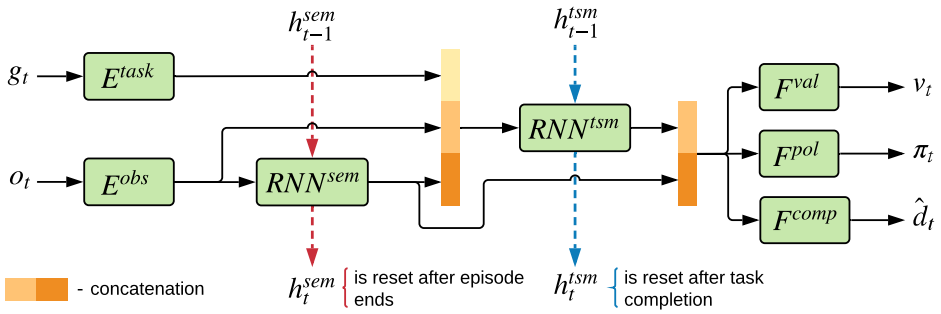


Figure 1: A diagram of the SEM-A2C network architecture. The green blocks are trainable modules. Dashed blue and red lines indicate the flow of information through the hidden states of the recurrent memory modules over time. SEM-A2C encodes current observation o_t and adds it to episodic memory RNN^{sem}. The task specific memory RNN^{tsm} is updated with the embeddings of task ids g_t and o_t as well as the content of episodic memory h_t^{sem}. Finally, episodic and task specific memories are fed to policy F^{pol}, value F^{val} and task completion F^{comp} heads.

4 Experiments

We studied SEM-A2C performance in randomized grid-worlds implemented on top of the Mazebase engine (Sukhbaatar et al., 2015). For multi-task experiments, (sect. 4.1) an agent was trained for the sub-tasks of the Taxi problem (Dietterich, 2000): reach passenger (*Reach*(P)), pick up passenger (*Pickup*(P)), reach destination (*Reach*(D)), and drop off passenger (*Dropoff*(P)). For continual learning experiments (sect. 4.2), we added a new cargo object on the map and three associated sub-tasks.

The map was filled with randomly placed walls and ponds. To make the problem harder an agent can only see objects in a small 7x7 cell grid which surrounds it, and has no direct information about the coordinates of the passenger, cargo and destination (see Figure 2).

In our study, we consider sub-tasks as separate goals for the agent. If the agent completes its current sub-task, the next sub-task is selected from those which may follow logically in the current state. We place the agent and every pickable object at a new location on the map after completion of every two or three sub-tasks. However, the map and location of the target remain unchanged throughout all T steps of the episode. Thus, it is beneficial for the agent to transfer acquired knowledge about the structure of the maze and location of the target between sub-tasks to facilitate their completion.

4.1 MULTI-TASK LEARNING

As a baseline we used a batch multi-task A2C (*Multitask-A2C*) with weight sharing and task parametrization (Chaplot et al., 2017; Denil et al., 2017; Florensa et al., 2017; Dosovitskiy & Koltun, 2016). It received the same input as SEM-A2C including g_t, but instead of separate RNN^{tsm} and RNN^{sem} sub-networks, it has a single LSTM layer with approximately the same number of parameters. Multitask-A2C is trained via batch multi-task learning, where each episode corresponds to one task.

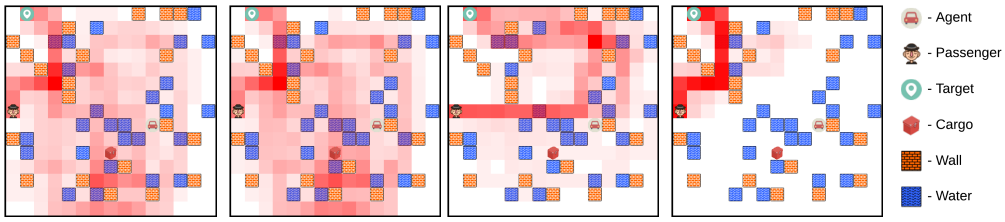
To evaluate the utility of episodic memory shared between the learned policies, we tested SEM-A2C and Multitask-A2C on the full taxi problem, where each agent had to perform the following sequence of tasks: *Reach*(P), *Pickup*(P), *Reach*(D), *Dropoff*(P). After dropping off a passenger at the target location a new one was spawned at a random location on the map. Each algorithm was previously trained for 80 million steps.

The results are presented in the Table 1. The table shows the number of steps required to complete tasks *Reach*(P) and *Reach*(D) depending on the order in which the tasks were performed in the episode. As can be seen from the table, SEM-A2C but not the Multitask-A2C baseline, manages to optimize the solution of *Reach*(D) sub-task. After the agent with shared episodic memory discovers the target it can deliver future passengers to the same location 40% faster. Due to episodic memory SEM-A2C solves the full Taxi problem with 20% less steps than Multitask-A2C.

Figure 2 shows the difference between SEM-A2C and Multitask-A2C agents on a random fixed map. SEM-A2C drastically improves its policy by utilizing the experience obtained by performing previous tasks in this episode (Figures 2c, 2d). Conversely, the Multitask-A2C Baseline did not learn to account its experience from the previous tasks performed in the same environment (Figures 2a, 2b).

Table 1: Episodic memory improves performance on the Taxi problem. Table shows the number of steps to complete sub-tasks over an episode on the 15x15 map. Values are averaged over 500 runs.

Appearance of sub-task	Reach Passenger sub-task		Reach Target sub-task	
in an episode	SEM-A2C	Multitask-A2C	SEM-A2C	Multitask-A2C
1^{st}	22.67	23.61	24.26	25.38
2^{nd}	22.38	21.34	15.04	26.41
3^{d}	25.53	21.84	15.19	25.33
4^{th}	25.14	24.48	16.09	25.21
5^{th}	25.5	22.40	14.80	24.02



(a) Multitask-A2C, before visiting target (b) Multitask-A2C, after visiting target (c) SEM-A2C, before visiting target (d) SEM-A2C, after visiting target

Figure 2: The heatmap shows the relative frequency of the agent visiting each location during the sub-task of carrying the passenger to the target location(*Reach*(D)) for Multitask-A2C and SEM-A2C. Each heatmap represents results averaged over 50 independent runs. The car icon shows location of the agent on the map at the beginning of the episode. Panels (a) and (c) show the behavior of agents that have never visited the target location before receiving the Reach(D) task. Panels (b) and (d) show the behavior of agents which visited the target location during the previous tasks in the episode.

4.2 CONTINUAL LEARNING

To study continual learning, we added a new cargo object to the map, as well as three associated tasks: reach cargo (*Reach*(C)), pickup cargo (*Pickup*(C)), deliver and drop off cargo at the target location (*Deliver*(C)). This experiment consisted of two stages. We first pre-trained our model and two new baselines together on 4 taxi sub-tasks and a new *Reach*(C) sub-task. The *Reach*(C) was included into pre-training to teach the E^{obs} sub-network to recognitize the cargo object on the map. The learning procedure in this stage was the same as in the multi-task experiment. In the cargo delivery training stage, we added *Pickup*(C) and *Deliver*(C) sub-tasks and fine-tuned the output layers and task embedding E^{task} of pre-trained models for 5×10^6 steps, keeping all other layers frozen.

To test the utility of the explicit division between task-agnostic episodic memory and recurrent task-dependent policies, we used two baselines:

1. **Baseline (concat)** has the same architecture as Multitask-A2C, but uses the same learning procedure as SEM-A2C (see sec. 3).
2. **Baseline (factorized)** is identical to the previous baseline. However, rather than concatenating the task embedding with the LSTM input, we use the same factorization as in RNN^{tsm} module.

Figure 3 shows learning curves for new tasks during the fine-tuning stage. Baseline (concat) did not succeed in learning both new tasks. Baseline (factorized) was able to fully learn only the simpler *Pickup*(C) task. On the other hand, SEM-A2C managed to learn both tasks in one million steps.

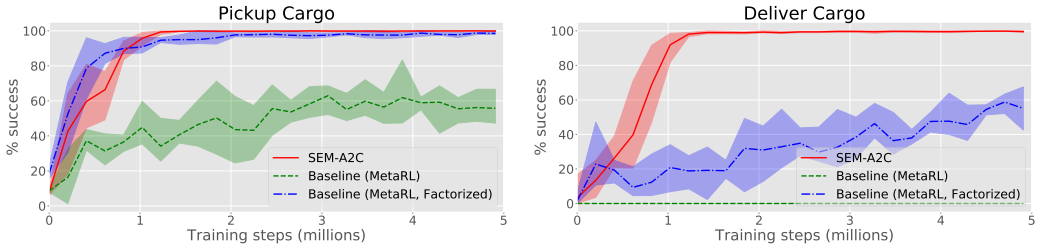


Figure 3: **Left**: Success rate for the *Pickup*(C) task during the fine-tuning stage. **Right**: Success rate for the *Deliver*(C) task during the fine-tuning stage. Each curve is averaged over 6 different runs. The shaded area corresponds to the minimum and maximum scores achieved during these runs.

5 CONCLUSIONS

Episodic memory helps to solve a significant number of tasks in the real world. Yet in spite of recent progress in the fields of deep reinforcement learning and meta-learning, the question about effective reuse of episodic memory is still open. We proposed and studied a deep neural architecture with shared episodic memory for multi-task problems (SEM-A2C).

The results of our experiments on the Taxi problem demonstrate that our proposed architecture is able to effectively learn how to store and use episodic representation in order to more quickly deliver a passenger. SEM-A2C displayed a better performance compared to alternative deep architectures included into the study. We also found that task agnostic episodic memory facilitates acquisition of novel skills for extra tasks in the same environment. Another important result is ability of SEM-A2C to learn sub-task completion. This opens possibility for more autonomous execution of hierarchical tasks by robotic and virtual agents, as a sub-task completion signal can activate the following task in a high level preprogrammed sequence.

We use A2C (Mnih et al., 2016; Clemente et al., 2017) as a starting point for our modifications and baselines. However, out proposed modifications do not rely on the unique properties of the A2C and can be applied to other general-purpose RL algorithms (PPO, DRQN, etc).

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their suggestions and comments. This work was supported by National Technology Initiative and PAO Sberbank project ID 0000000007417F630002.

REFERENCES

Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. *arXiv preprint arXiv:1706.07230*, 2017.

Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(3):1–145, 2016.

Alfredo V Clemente, Humberto N Castejón, and Arjun Chandra. Efficient parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1705.04862*, 2017.

Misha Denil, Sergio Gómez Colmenarejo, Serkan Cabi, David Saxton, and Nando de Freitas. Programmable agents. *arXiv preprint arXiv:1706.06383*, 2017.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*, 2016.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl$\hat{2}$: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*, 2017.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 7(1), 2015.

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

Russell Kaplan, Christopher Sauer, and Alexander Sosa. Beating atari with natural language guided reinforcement learning. *arXiv preprint arXiv:1704.05539*, 2017.

Dharshan Kumaran, Demis Hassabis, and James L McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20 (7):512–534, 2016.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

Konstantin Lakhman and Mikhail Burtsev. Neuroevolution results in emergence of short-term memory in multi-goal environment. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pp. 703–710. ACM, 2013.

James L McClelland, Bruce L McNaughton, and Randall C O'reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. *arXiv preprint arXiv:1704.08795*, 2017.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128*, 2016.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8. IEEE, 2018.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. *arXiv preprint arXiv:1703.01988*, 2017.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 7310–7321, 2018.

Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370*, 2018.

Sainbayar Sukhbaatar, Arthur Szlam, Gabriel Synnaeve, Soumith Chintala, and Rob Fergus. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*, 2015.

Matthew E Taylor and Peter Stone. An introduction to intertask transfer for reinforcement learning. *Ai Magazine*, 32(1):15, 2011.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 4496–4506, 2017.

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, volume 3, pp. 6, 2017.

Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pp. 3–17. Springer, 1998.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

A SUPPLEMENTARY MATERIALS FOR "EPISODIC MEMORY FOR MULTI-TASK AND CONTINUAL REINFORCEMENT LEARNING"

A.1 FACTORIZED LSTM LAYER

At each time-step, the LSTM layer computes it's output vector h_t and the cell state c_t, given the previous vector h_{t-1}, the previous cell state c_{t-1}, and the current observation x_t:

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} T \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}, \tag{1}$$

$$c_t = f \odot c_{t-1} + i \odot g, \tag{2}$$

$$h_t = o \odot tanh(c_t), \tag{3}$$

here T is an affine transformation $T = W * [x_t, h_{t-1}] + b$ and (i, f, o, g) are LSTM gates (see (Hochreiter & Schmidhuber, 1997)).

For RNN^{tsm} module LSTM weights W_{g_t} for task g_t are computed as a product of three matrices:

$$W_{g_t} = W_1 \operatorname{diag}(v_{g_t}) W_2, \tag{4}$$

here v_{g_t} is an embedding vector of the current task g_t. Weights W_1 and W_2 are shared across all tasks, while task embeddings are trained for each task. This technique allows storing large weight matrices for each task to be avoided. Additionally, the weights factorization significantly increases sensitivity of the RNN^{tsm} module to different task embeddings. This in turn leads to improved exploration. As shown in Figure 3 the baseline with the factorized LSTM layer outperforms the baseline with regular LSTM layer that gets task embeddings as a part of it's input.

A.2 SEM-A2C ARCHTECTURE

The following equations describe the forward dynamics of the SEM-A2C network (see fig.1):

$$\hat{o}_t = [E^{obs}(o_t), d_{t-1}, a_{t-1}], \tag{5}$$

$$h_t^{sem} = RNN^{sem}(h_{t-1}^{sem}, \hat{o}_t), \tag{6}$$

$$\hat{h}_t^{tsm} = (1 - d_{t-1}) h_{t-1}^{tsm}, \tag{7}$$

$$h_t^{tsm} = RNN^{tsm}(\hat{h}_t^{tsm}, E^{task}(g_t), [\hat{o}_t, h_t^{sem}]), \tag{8}$$

$$\pi_t = SoftMax(F^{pol}(h_t^{sem}, h_t^{tsm})), \tag{9}$$

$$v_t = F^{val}(h_t^{sem}, h_t^{tsm}), \tag{10}$$

$$\hat{d}_t = F^{comp}(h_t^{sem}, h_t^{tsm}). \tag{11}$$

Training of SEM-A2C is performed in the same way as in A2C (Clemente et al., 2017) and A3C (Mnih et al., 2016) algorithms. To learn the task completion prediction $\hat{d}_t$ we use cross-entropy loss.

A.3 ENVIRONMENT SETUP

We studied the proposed model at the modified Taxi problem (Dietterich, 2000). The Taxi problem is a clear example of a task with a hierarchy of sub-tasks. The goal of this problem is to maneuver a taxi in order to reach a passenger placed in a random location, and then to pick up and deliver the passenger to a target location. Thus the main task of delivering a passenger from an initial to a target location, is divided into 4 separate sub-tasks:

1. Reach (P): to get to the passenger on the map;
2. Pickup (P): to put the passenger in the car;
3. Reach (D): to transport the passenger to the target location;

4. Dropoff (P): to disembark the passenger.

In the original Taxi game, a map on which the agent operates was fixed throughout the training. In our implementation, the map is created randomly for each new episode. On the map, 10% of the blocks are filled with impassable terrain (walls) and another 10% contain difficult terrain (water). The starting positions of the agent, the passenger and the target are chosen randomly. In order to increase the difficulty of the problem, the agent is only permitted to view the small 7 by 7 area of cells surrounding it. The agent does not possess knowledge of the initial coordinates of the passenger and the target. It must independently find them on the map. For each action, the agent receives a penalty of -0.1, and any action performed on difficult terrain the penalty increases to -0.3. After sub-task completion, the agent receives a reward equal to 1. The length of one training episode is 400 steps. Our implementation is built on top of the MazeBase engine (Sukhbaatar et al., 2015).