Learning to Learn from Failures Using Replay

Tao Chen [1] **Pulkit Agrawal** [1]

Abstract

Learning from past mistakes is a quintessential aspect of intelligence. In sequential decision-making, existing meta-learning methods that learn a learning algorithm utilize experience from only a few previous episodes to adapt their policy to new environments and tasks. Such methods must learn to correct their mistakes from highly-correlated sequences of states and actions generated by the same policy's consequent roll-outs during training. Learning from correlated data is known to be problematic and can significantly impact the quality of the learned correction mechanism. We show that this problem can be mitigated by augmenting current systems with an external memory bank that stores a larger and more diverse set of past experiences. Detailed experiments demonstrate that our method outperforms existing meta-learning algorithms on a suite of challenging tasks from raw visual observations. Code and videos are available at: https://sites.google.com/view/learn-from-failures.

1. Introduction

Agents often fail to solve a new decision-making task in their first attempt. While failures do not provide good rewards, they communicate what not to do and often hint at possible solutions. Analyzing past failures to improve the current strategy is vital for adapting and solving new tasks. Because of being memory-less, popular deep reinforcement learning (DRL) algorithms (Mnih et al., 2015; Silver et al., 2017; Lillicrap et al., 2015) cannot exploit previous episodes to adjust their decisions at test time. However, recently proposed methods that learn a learning algorithm (Duan et al., 2016; Mishra et al., 2017) address this shortcoming by processing the history of multiple episodes to select the next action. Such meta-learning algorithms can learn from past failures and have achieved excellent performance on many challenging partially-observable tasks.

In this work, we utilize two observations to construct a modified training procedure for meta-learning methods that learn a learning algorithm. The proposed training scheme leads to substantial performance gains on tasks that require recovery from failures. To understand our contributions, let's first review the training process of existing methods: An agent is provided with a set of tasks to solve. Training happens in trials. In every trial, a training task is randomly sampled, and the agent acts for a fixed number of steps (say T). If the agent completes the task or the episode terminates before T steps, reset is performed, and a new episode starts. A trial typically contains multiple episodes.

Current meta-learning algorithms learn a policy that depends on all previous episodes only in the same trial. Different episodes in a single trial are the outcome of executing the policy network with same weights and are correlated. Correlated self-generated training data results in instability in training. To overcome this shortcoming in the context of meta-learning, we propose having a memory bank for each task that stores episodes across trials. By storing mistakes across trials, such memory banks encourage learning of policies that can recover from a diverse set of failures.

Our second observation is that despite being successful at a task, it is important to preserve memories of past failures. Consider the setup where the agent only utilizes memory of episodes from the same trial. As the agent trains, it will make fewer mistakes in the training tasks. In the extreme case of fitting, the agent may succeed in the first episode of the trial, and as a consequence, the agent's memory will be populated only with successful trajectories. Lack of failures in memory might have the unintended consequence of making the agent forget how to recover from failures by the end of the training. We show that this issue can be mitigated by explicitly storing the most recent failures (possibly from earlier trials) of the agent. In this paper, we used the observations of labeling and storing failures across trials to build a meta-learning algorithm that outperforms the existing state-of-the-art (Mishra et al., 2017). This was achieved by constructing task-specific memory banks that replay trajectories from multiple previous trials of the same

[1]EECS Department, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. Correspondence to: Tao Chen <taochen@mit.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

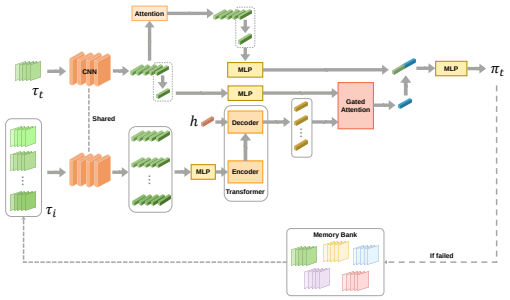


Figure 1. Policy architecture. The policy starts with an empty memory bank. An agent executes the policy in a task. If it fails, the trajectory is added to a memory bank. When the agent performs the task again, the policy adapts the output based on the information from memory. The policy uses the Transformer to extract information from each trajectory and a gated attention mechanism to merge the information of multiple trajectories.

task. We validate our method on 2D and 3D navigation environments, and Procgen environment.

2. Method

Our setup is as follows: An agent is required to solve a set of tasks $\mathcal{M}$. Training is performed in trials. In each trial, the agent executes a fixed number of actions, say T. A trial typically consists of multiple episodes, where each episode refers to the trajectory taken by the agent starting from the initial state until the environment is reset due to either failure or success of the agent or a timeout. During training, suppose the agent encounters the task $\mathcal{M}_i$ for the j^{th} time (i.e, j^{th} trial for task $\mathcal{M}_i$). Let the set of episodes in this trial be $\{\tau_k^{i,j}; k \in K\}$, where K, the total number of episodes, can vary across trials. Let the k^{th} episode be $\tau_k^{i,j} : \left(o_{t^k}^{i,j}, a_{t^k}^{i,j}, r_{t^k}^{i,j}, ..., o_{t^{k+1}-1}^{i,j}, a_{t^{k+1}-1}^{i,j}, r_{t^{k+1}-1}^{i,j}\right)$, where $o_{t^k}^{i,j}, a_{t^k}^{i,j}, r_{t^k}^{i,j}$ represent the observation, action and the reward obtained by the agent at a time step t^k. Further assume that the agent is equipped with $\|\mathcal{M}\|$ memory banks, one for each task: $\mathcal{B}^{\mathcal{M}_i}$. Each memory bank stores a maximum of N trajectories. At every time step in the trial, the agent chooses actions ($a_t^{i,j}$) using the policy,

$$a_t^{i,j} = \pi_\theta(o_t, h_{t-1}^{i,j}, \mathcal{B}^{\mathcal{M}_i}), h_{t-1}^{i,j} = \{o_{1:t-1}^{i,j}, a_{1:t-1}^{i,j}, r_{1:t-1}^{i,j}\} \quad (1)$$

where, $h_{t-1}^{i,j}$ represents the agent's trajectory in the current trial until time $t-1$ and θ represents the policy parameters. The policy π_θ depends on the current observation, the entire history of the current trial ($h_{t-1}^{i,j}$), and the trajectories stored in the memory bank ($\mathcal{B}^{\mathcal{M}_i}$). The policy is optimized to maximize the sum of rewards in the trial.

The memory bank is populated in the following way: Let $f(\tau)$ be a binary function that returns whether the episode τ results in a success ($f(\tau) = 1$) or failure ($f(\tau) = 0$). Let $\mathcal{F}^{i,j} = \{k : f(\tau_k^{i,j}) = 0; k \in [1, K]\}$ be the set of failure episodes in the j^{th} trial. At the end of the trial, the failure episodes $\mathcal{F}^{i,j}$ are added to the memory bank $\mathcal{B}^{\mathcal{M}_i}$. If $\mathcal{B}^{\mathcal{M}_i}$ is out of space, the earliest episodes stored in the memory are dropped. Figure 1 visually illustrates the policy.

Computing Feature Representation of Trajectories in the Memory Bank: We use the *Transformer* to compute the feature representation of each trajectory in the memory bank. Let the trajectories in $\mathcal{B}^{\mathcal{M}_i}$ be $\{\tau_n^i; n \in [1, N]\}$. The transformer consists of an encoder that computes information per time-step of the trajectory and a decoder that aggregates this information.

$$e_{\tau_n^i} = \text{Encoder}(\Phi(\tau_n^i)) \qquad v_{\tau_n^i} = \text{Decoder}(g, e_{\tau_n^i})$$

where Φ is a function (CNN and MLP layers) that merges the information of observation, action, and reward at each time step, $e_{\tau_n^i} \in \mathbb{R}^{L \times E}$ are the embedding vectors for each time step in the trajectory, $g \in \mathbb{R}^E$ is randomly initialized and learned via back-propagation, and $v_{\tau_n^i} \in \mathbb{R}^E$ is the embedding of the entire trajectory.

Aggregating Information Across Trajectories in the Memory Bank: Next, the policy aggregates information from all trajectories in the memory bank $\mathcal{B}^{\mathcal{M}_i}$ using *multi-head self-attention* (Vaswani et al., 2017) with *GRU gating* (Chung et al., 2014; Parisotto et al., 2019). Each trajectory is embedded into a vector of length E as in Section 2, resulting in a set of trajectory vectors $v_{\tau^i} \in \mathbb{R}^{N \times E}$. This set is reduced into a single vector $m_o^i \in \mathbb{R}^E$ that represents the contribution of the memory.

$$m_d^i = \text{MultiHead}(Q, K, V) = \text{MultiHead}(v'_{o_t}, v_{\tau^i}, v_{\tau^i})$$
$$m_o^i = \text{GRU}(v'_{o_t}, m_d^i)$$

where $v'_{o_t} = \text{MLP}(\text{CNN}(o_t))$ is the embedding vector of the current observation.

Encoding Episodic Temporal Information: To encode information from past steps of the episode, we use *multi-head self-attention* mechanism:

$$v_{o_t} = \text{MLP}(\text{MultiHead}(v_{\tau_t}, v_{\tau_t}, v_{\tau_t}) \oplus v_{\tau_t})[-1]$$

where $[-1]$ refers to feature embedding of the current time step in inference, $\oplus$ is the concatenation operator v_{τ_t} denoted is the featurization of current trajectory τ_t. Note that we only encode past trajectory information from the current episode. Therefore, in the k^{th} episode, $h_{t-1}^{i,j} = \{o_{t^k:t-1}^{i,j}, a_{t^k:t-1}^{i,j}, r_{t^k:t-1}^{i,j}\}$ in Equation (1). In experiments without temporal embedding, $v_{o_t} = \text{MLP}(\text{CNN}(o_t))$, which is shown in Figure D.2 in Appendix D.

Memory-conditioned Policy: The final policy π merges information from current trial and memory bank. $\pi = \text{MLP}\left[\text{MLP}(v_{o_t}) \oplus m_o^i\right]$ where m_o^i is the feature representation of the memory $\mathcal{B}^{\mathcal{M}_i}$. If $\mathcal{B}^{\mathcal{M}_i}$ is empty, $m_o^i = \mathbf{0}$.

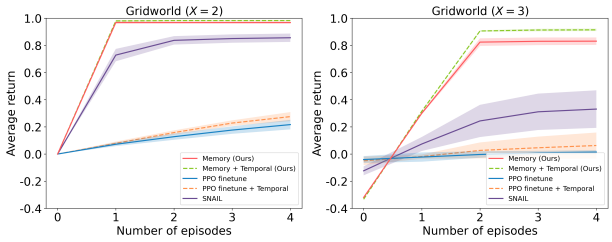
3. Experiments

While our method is fully compatible with reinforcement learning algorithms such as PPO, we use behavior cloning to train all the policies, to speed up the training.

Baselines: We compare our method against the following baselines: **(1)** SNAIL (Mishra et al., 2017) is a meta-learning algorithm that accumulates information across episodes in the same trial as described in Section 2. **(2)** We first trained a memoryless base-policy (*i.e.*, $a_t = \pi_\theta(o_t)$) using expert demonstrations. At test time, we finetune this policy after every episode using PPO (Schulman et al., 2017).

3.1. Gridworld

Environment setup: The gridworld is $W \times W$ in size and has $X + 1$ blocks in different colors (Figure B.1a). The black block represents the agent. One block is the goal. The other $X - 1$ blocks are traps. The agent gets $+1$ reward if it reaches the goal, -1 reward if it reaches any trap, and 0 reward otherwise. The color of the goal and the positions of the blocks vary across tasks. The agent is not provided with the color of the goal block, which makes the environment partially-observable. More experiment details are in the Appendix B.



(a) $X = 2$ (b) $X = 3$

Figure 2. Our methods (*Memory*, *Memory + Temporal*) outperform the *SNAIL* and *PPO finetune* baselines on testing environments in gridworld and miniworld. Mean and standard deviation in the average accumulated reward across 3 seeds are reported.

Evaluation: Our method is evaluated as follows: At the beginning of a trial in test time, the memory bank is empty. The agent acts using its policy, and if an episode fails, the last frame of the episode (*i.e.*, $L = 1$) is added to the memory bank. The agent rolls out five episodes. *SNAIL* is evaluated in the same way, except that in a manner consistent with the original work, all frames from all previous episodes (and not just failures) in the trial are processed to predict the action. For *PPO finetune* baseline, the policy is finetuned with PPO after each episode.

Figure 2 shows that the performance of all methods improves with the number of episodes. Our method (called *Memory*) outperforms *SNAIL* and *PPO finetune* in testing environments. Figures 2a,2b compare performance in environments with one and two traps respectively. While the performance of *SNAIL* reduces significantly with an increase in task complexity, only a modest drop in performance is observed for our method. This suggests that our method is superior to the state-of-the-art for more complex tasks. It can also be seen that *Memory + Temporal* outperforms *Memory*, indicating that additional information obtained from previous steps of the same episode is useful.

Effect of the length L of trajectories in memory: In the previous experiments, only the last time step of each trajectory is stored in the memory. In general, it is not known apriori how many time-steps of an episode should be stored in the memory bank. It is expected that storing longer trajectories can make learning harder. To see the effect of trajectory length L, we trained different versions of our system, each storing a different number of last L steps of the episode. Results in Figures 3a,3b show that the performance of our method is invariant to L. Therefore, our method does not require knowledge of how many steps of the failed episode should be stored in the memory.

Effect of Diversity: Our central hypothesis is that training with diverse failures is a better mechanism for learning to recover from failures. To investigate how critical is diversity, we performed ablations under two conditions: (a) using trials of 80 steps (called normal horizon, `NH`); (b) trials of 300 steps (called long horizon, `LH`). The different variants of our method in these two conditions are:

- `NH{LH}_RESET_DIFF`: To maximize the diversity of trajectories stored in the memory bank, on termination of an episode, the environment is reset to a random task. As the policy gets updated between trials, the failures are more likely to be different when the policy generates them at different training iterations. Hence, the memory bank will have more diverse failures.
- `NH{LH}_RESET_SAME`: An environment is reset to the same task throughout the trial following the setup in (Duan et al., 2016; Mishra et al., 2017). In this case, the diversity in the memory goes down, because the memory has a limited capacity, and trajectories from the same policy are more likely to be similar. If the trial horizon is longer (`LH`), the diversity drops even more as more trajectories will come from the same policy.
- `NH{LH}_SNAIL`: SNAIL with a normal/long trial horizon.

We can see from Figure 3c that using a memory bank can indeed significantly boost the learning speed comparing to SNAIL. We can also see that `NH{LH}_RESET_DIFF` learns much faster than `NH{LH}_RESET_SAME`, which fits well with our hypothesis. The comparison between `NH_RESET_SAME` and `LH_RESET_SAME` shows that longer trial horizon with `RESET_SAME` does lead to significantly slower learning. However, in case of `RESET_DIFF`, the

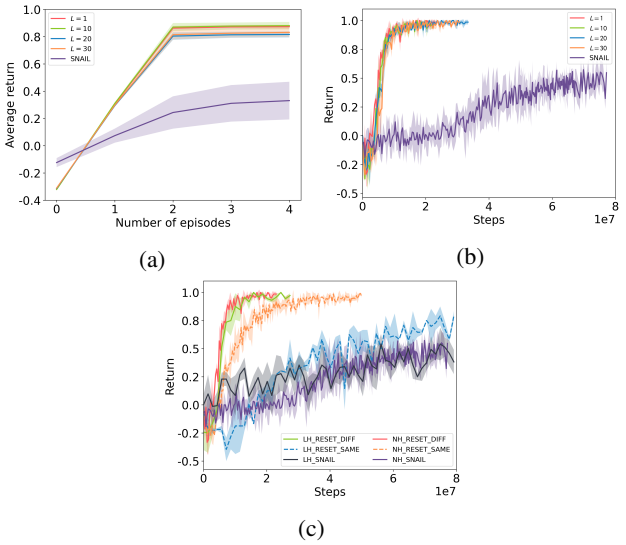


Figure 3. $X = 3$. **(a)**: testing performance for different trajectory length L. **(b)**: learning curves in training for different L. **(c)**: learning curves for different reset mechanisms.

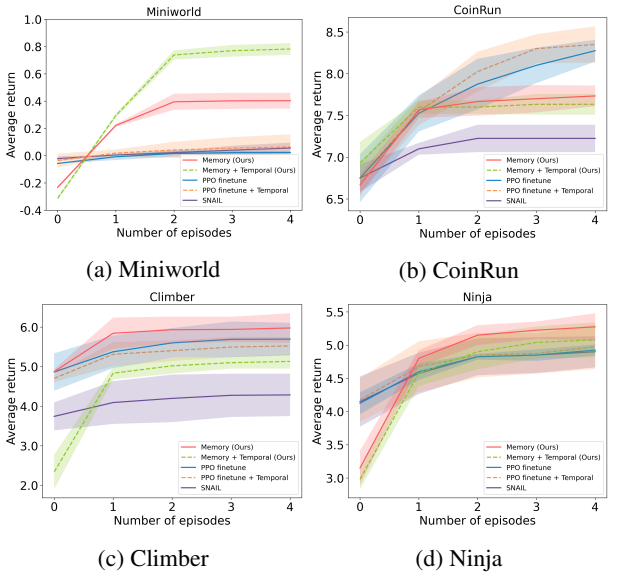


Figure 4. Test-time performance on miniworld and three video games from Procgen. Our method outperforms SNAIL on all the environments.

learning speed is not affected by the trial horizon, because the memory has enough diversity even if T is large. All of the four variants of our method perform better than SNAIL (`NH_SNAIL` and `LH_SNAIL`).

3.2. 3D Miniworld

We also conduct experiments in a 3D navigation environment. The task of the agent is the same as the gridworld environment. There are 3 boxes in the room ($4\text{m}\times4\text{m}$) with different colors. Only one of the boxes gives $+1$ reward, and the other two boxes give -1 reward. The agent's action set is {move forward by 0.3m, turn left by $18°$, turn right by $18°$}. The observation is a first-person view as shown in Figure B.1d. We again use path planning to provide supervision for policy learning. Figure 4a shows that our methods perform much better than the baselines. The advantage of using a memory bank is even more salient in this case. As the agent only has the first-person view of the scene (Figure B.1d), the agent needs intra-episode information to efficiently search and move towards the goal, which is verified by the performance gap between *Memory* and *Memory + Temporal*.

3.3. Procgen

Beyond object collection tasks, we tested our method on a few games (CoinRun, Climber, and Ninja) from the Procgen benchmark (Cobbe et al., 2019). For each video game, all models are trained on 200 levels (tasks) with supervised learning using expert demonstrations, which in turn were collected by training a PPO (Schulman et al., 2017) policy.

We evaluate performance on 100 held-out levels. Figure 4b, 4c, 4d shows that performance of all methods improves with number of testing episodes and our algorithm consistently outperforms *SNAIL*. This suggests that memory-bank and marking failures are very useful additions to the class of meta-learning systems that learn a learning algorithm.

It is interesting to note that in all these games *PPO finetune* is competitive with our method and even outperforms us on CoinRun. We suspect the reason for this extremely good performance is the biases present in these games. For instance, out of 15 possible actions, only two actions *jump* and move-right are sufficient to solve CoinRun. The learned policy exploits this and puts most of the probability mass on these two actions. If the agent fails, its errors can be corrected by interchanging jump and move-right actions at a few critical points. We hypothesize that gradient descent can easily find this solution because negative rewards decrease the probability of actions in the previous rollout, which naturally increases the probability of the only other alternative action in the next rollout.

4. Discussion

In this paper, we apply the idea of replaying failures to the meta-learning setting where an agent learns to avoid making the mistakes that it has made in the past. We show that having a memory bank of the experience across multiple previous trials and explicitly replaying the past failures can significantly improve learning speed as well as the test-time performance.

References

Beaulieu, S., Frati, L., Miconi, T., Lehman, J., Stanley, K. O., Clune, J., and Cheney, N. Learning to continually learn. *arXiv preprint arXiv:2002.09571*, 2020.

Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J. Z., Rae, J., Wierstra, D., and Hassabis, D. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.

Fortunato, M., Tan, M., Faulkner, R., Hansen, S., Badia, A. P., Buttimore, G., Deck, C., Leibo, J. Z., and Blundell, C. Generalization of reinforcement learners with working and episodic memory. In *Advances in Neural Information Processing Systems*, pp. 12448–12457, 2019.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.

Hung, C.-C., Lillicrap, T., Abramson, J., Wu, Y., Mirza, M., Carnevale, F., Ahuja, A., and Wayne, G. Optimizing agent behavior over long time scales by transporting value. *Nature Communications*, 10(1):5223, Nov 2019. ISSN 2041-1723. doi: 10.1038/s41467-019-13073-w. URL https://doi.org/10.1038/s41467-019-13073-w.

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

Koch, G., Zemel, R., and Salakhutdinov, R. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

Lengyel, M. and Dayan, P. Hippocampal contributions to control: the third way. In *Advances in neural information processing systems*, pp. 889–896, 2008.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Nichol, A. and Schulman, J. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2:2, 2018.

Oh, J., Chockalingam, V., Singh, S., and Lee, H. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128*, 2016.

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

Parisotto, E., Song, H. F., Rae, J. W., Pascanu, R., Gulcehre, C., Jayakumar, S. M., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., et al. Stabilizing transformers for reinforcement learning. *arXiv preprint arXiv:1910.06764*, 2019.

Pritzel, A., Uria, B., Srinivasan, S., Badia, A. P., Vinyals, O., Hassabis, D., Wierstra, D., and Blundell, C. Neural episodic control. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2827–2836. JMLR. org, 2017.

Rakelly, K., Zhou, A., Finn, C., Levine, S., and Quillen, D. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pp. 5331–5340, 2019.

Ritter, S., Wang, J. X., Kurth-Nelson, Z., Jayakumar, S. M., Blundell, C., Pascanu, R., and Botvinick, M. Been there, done that: Meta-learning with episodic recall. *arXiv preprint arXiv:1805.09692*, 2018.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pp. 1842–1850, 2016.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Vilalta, R. and Drissi, Y. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pp. 3630–3638, 2016.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Wayne, G., Hung, C.-C., Amos, D., Mirza, M., Ahuja, A., Grabska-Barwinska, A., Rae, J., Mirowski, P., Leibo, J. Z., Santoro, A., et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.