A Simple Neural Attentive Meta-Learner

Nikhil Mishra[*†‡] **Mostafa Rohaninejad**[*†] **Xi Chen**[†] **Pieter Abbeel**[†]

† UC Berkeley, ‡ OpenAI

Abstract

Deep neural networks tend to struggle when data is scarce or when they need to adapt quickly to a changing task. In response, *meta-learning* proposes training a *meta-learner* on a distribution of similar tasks, in the hopes of generalization to novel but related tasks by learning a high-level strategy that exploits commonalities in task structure. However, many recent meta-learning approaches are extensively hand-designed, either using architectures specialized to a particular application, or hard-coding algorithmic components that constrain how the meta-learner solves the task. We propose a class of simple and generic meta-learner architectures that use a novel combination of temporal convolutions and soft attention; the former to aggregate information from past experience and the latter to pinpoint specific pieces of information. In the most extensive set of meta-learning experiments to date, we evaluate the resulting Simple Neural AttentIve Learner (or SNAIL) on several heavily-benchmarked tasks. On all tasks, in both supervised and reinforcement learning, SNAIL attains state-of-the-art performance by significant margins.

1 Introduction

The key principle motivating SNAIL is simplicity and versatility: a meta-learner should be universally applicable to domains in both supervised and reinforcement learning. It should be generic and expressive enough to learn an optimal strategy, rather than having the strategy already built-in. Prior work [11] has formalized meta-learning as a sequence-to-sequence problem and attempted to tackle it with traditional RNN architectures. Despite the generality of such approaches, the bottleneck seems to be in the recurrent architecture's ability to internalize and refer to past experience.

van den Oord et al. [16] introduced a class of architectures that generate sequential data (in their case, audio) by performing dilated 1D-convolutions over the temporal dimension. These temporal convolutions (TC) are causal, so that the generated values at the next timestep are only influenced by past timesteps and not future ones. Compared to traditional RNNs, they offer more direct, high-bandwidth access to past information, allowing them to perform more sophisticated computation over a temporal context of fixed size. However, to scale to long sequences, the dilation rates generally increase exponentially, so that the required number of layers scales logarithmically with the sequence length. Hence, they have coarser access to inputs that are further back in time; their bounded capacity and positional dependence can be undesirable in a meta-learner, which should be able to fully utilize increasingly large amounts of experience.

In contrast, soft attention (in particular, the style used by Vaswani et al. [17]) allows a model to pinpoint a specific piece of information from a potentially infinitely-large context. It treats the context as an unordered key-value store which it can query based on the content of each element. However, the lack of positional dependence can also be undesirable, especially in reinforcement learning, where the observations, actions, and rewards are intrinsically sequential.

Despite their individual shortcomings, temporal convolutions and attention complement each other: while the former provide high-bandwidth access at the expense of finite context size, the latter can

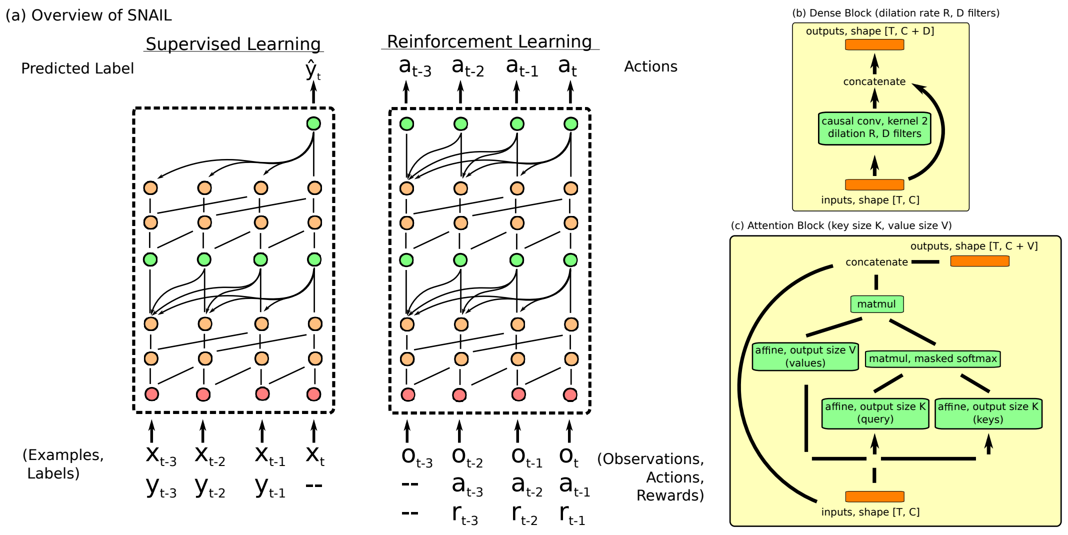
[*] denotes equal contribution, authors are listed in alphabetical order.

provide pinpoint access over an unbounded context. Hence, we construct SNAIL by combining the two: we use temporal convolutions to produce the context over which we use a causal attention operation. By interleaving TC layers with causal attention layers, SNAIL can have high-bandwidth access over its past experience *without* constraints on the amount of experience it can effectively use. By using attention at multiple stages within a model that is trained end-to-end, SNAIL can learn what pieces of information to pick out from the experience it gathers, as well as a feature representation that is amenable to doing so easily.

We compose SNAIL architectures using a few primary building blocks. Figure 1 provides an illustration, as well as an overview of SNAIL.

- A *dense block* applies a single causal 1D-convolution with dilation rate R and D filters (we used kernel size 2 in all experiments), and then concatenates the result with its input.
- A *TC block* consists of a series of dense blocks whose dilation rates increase exponentially until their receptive field exceeds the desired sequence length.
- A *attention block* performs a single key-value lookup; we style this operation after the self-attention mechanism proposed by Vaswani et al. [17].

Figure 1: (a) An overview of SNAIL. The same class of model architectures, which combine TC (orange) with attention (green) can be applied to both supervised and reinforcement learning. (b) A dense block performs a temporal convolution and concatenates the input and output. (c) An attention block performs a causal key-value lookup.


(a) Overview of SNAIL
Supervised Learning
Reinforcement Learning
Predicted Label
$\hat{y}_t$
a_{t-3} a_{t-2} a_{t-1} a_t
Actions
(Examples, Labels)
x_{t-3} x_{t-2} x_{t-1} x_t
y_{t-3} y_{t-2} y_{t-1} --
o_{t-3} o_{t-2} o_{t-1} o_t
-- a_{t-3} a_{t-2} a_{t-1}
-- r_{t-3} r_{t-2} r_{t-1}
(Observations, Actions, Rewards)
(b) Dense Block (dilation rate R, D filters)
outputs, shape [T, C + D]
concatenate
causal conv, kernel 2 dilation R, D filters
inputs, shape [T, C]
(c) Attention Block (key size K, value size V)
outputs, shape [T, C + V]
concatenate
matmul
affine, output size V (values)
matmul, masked softmax
affine, output size K (query)
affine, output size K (keys)
inputs, shape [T, C]

In supervised settings, SNAIL receives as input a sequence of example-label pairs $(x_1, y_1), \ldots, (x_{t-1}, y_{t-1})$ for timesteps $1, \ldots, t-1$, followed by an unlabeled example $(x_t, -)$. It then outputs its prediction for x_t based on the previous labeled examples it has seen.

In reinforcement-learning settings, it receives a sequence of observation-action-reward tuples $(o_1, -, -), \ldots, (o_t, a_{t-1}, r_{t-1})$. At each time t, it outputs a distribution over actions a_t based on the current observation o_t as well as previous observations, actions, and rewards. Crucially, following existing work in meta-RL [1, 19], we preserve the internal state of a SNAIL across episode boundaries, which allows it to have memory that spans multiple episodes.

2 Results and Discussion

We evaluated SNAIL on a number of meta-learning domains introduced by prior work[1]; SNAIL achieved state-of-the-art performance on all domains, outperforming methods that are hand-tailored to a particular domain, or rely on built-in algorithmic priors. Below, we summarize the different benchmarks; for more details we refer the reader to Appendix A:

- Few-shot classification: the Omniglot and mini-Imagenet datasets are the standard benchmarks in supervised meta-learning. In Table 2, we report 1-shot and 5-shot accuracies.

[1]Some video results can be found at `https://sites.google.com/view/snail-iclr-2018/`.

- Multi-armed bandit problems [1, 19]: feature a stateless task involving exploration and exploitation. The meta-learner interacts with K arms (whose reward distributions are unknown) for N timesteps. As an oracle, we consider the Gittins index [3], which is optimal as $N \to \infty$. In Table 2, we report the average reward per episode for different N, K.
- Tabular MDPs [1, 19]: we generate random MDPs following Duan et al. [1], and allow a meta-learner to interact with each for N episodes. In Table 2, we report performance normalized by that of value iteration (optimal if the MDP parameters are known).
- Continuous control [2]: a simulated robot must adapt its locomotion to run in a random direction or at a random goal velocity. In Figure 2, we show test-time adaptation curves on the different task distributions introduced by Finn et al. [2]. SNAIL learns to identify the task and perform optimally within a few timesteps.
- Visual navigation [1, 19]: an agent must navigate random mazes to find a randomly-located goal, using first-person visual observations. It interacts with each maze for two episodes, so an optimal agent should explore the maze on the first episode, and then go directly to the goal on the second. In Figure 3, we show sample observations and maze layouts, as well as emergence of the optimal strategy. In Table 2, we report the average time to find the goal on unseen mazes, and explore generalization to larger mazes.

In the bandit and MDP domains, there exist human-designed algorithms with optimality guarantees. Despite the relative lack of task structure, these domains let us evaluate the optimality of a meta-learned algorithm. However, the true utility of a meta-learner is that it can learn an algorithm specialized to a particular task distribution. We evaluate this in the visual navigation and continuous control domains, where there is significant task structure to exploit, but no optimal algorithms.

In the RL domains (all but few-shot classification), we benchmark SNAIL against the following meta-learners. All were trained using TRPO with GAE [12, 13].

- An LSTM-based meta-learner, as concurrently proposed by Duan et al. [1], Wang et al. [19]. We refer to this method as “LSTM" in the subsequent tables and figures.
- MAML, the method introduced by Finn et al. [2]. It trains the initial parameters of a policy to achieve maximal performance after one (policy) gradient update on a new task.

The subsequent tables summarize our results for each domain. We highlight the best-performing method, as well as any with overlapping confidence intervals (sometimes omitted for brevity).

Method	5-Way Omniglot		20-Way Omniglot		5-Way mini-Imagenet	
	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
Santoro et al. [11]	82.8%	94.9%	–	–	–	–
Koch [6]	97.3%	98.4%	88.2%	97.0%	–	–
Vinyals et al. [18]	98.1%	98.9%	93.8%	98.5%	43.6%	55.3%
Finn et al. [2]	**98.7%**	**99.9%**	95.8%	98.9%	48.7%	63.1%
Snell et al. [14]	97.4%	99.3%	96.0%	98.9%	49.42%	68.20%
Munkhdalai and Yu [8]	**98.9%**	–	97.0%	–	49.21%	–
SNAIL, Ours	**99.07%**	**99.78%**	**97.64%**	**99.36%**	**55.71%**	**68.88%**

Table 1: Few-shot classification: 5-way Omniglot, 20-way Omniglot, and 5-way mini-Imagenet.

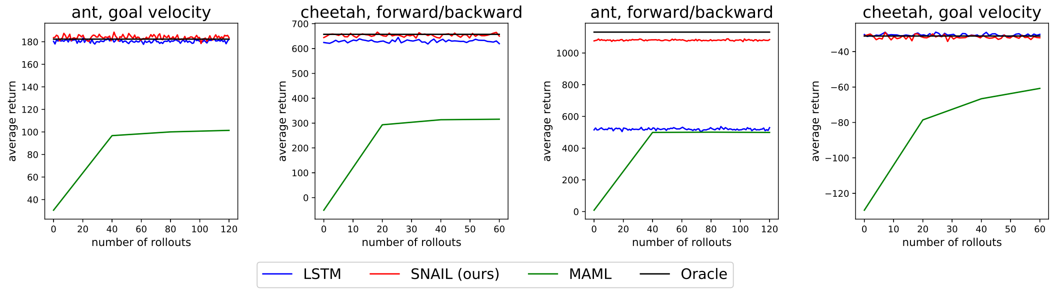


Figure 2: Test-time adaptation curves on simulated locomotion tasks. As an oracle, we compute the average performance of a policy trained on individual tasks sampled from each distribution.

Setup (N, K)	Gittins (optimal as $N \to \infty$)	Random	LSTM	MAML	SNAIL (ours)
$10, 5$	**6.6**	5.0	**6.7**	6.5 ± 0.1	$\mathbf{6.6 \pm 0.1}$
$10, 10$	**6.6**	5.0	**6.7**	$\mathbf{6.6 \pm 0.1}$	$\mathbf{6.7 \pm 0.1}$
$10, 50$	6.5	5.1	**6.8**	$\mathbf{6.6 \pm 0.1}$	$\mathbf{6.7 \pm 0.1}$
$100, 5$	**78.3**	49.9	**78.7**	67.1 ± 1.1	$\mathbf{79.1 \pm 1.0}$
$100, 10$	**82.8**	49.9	**83.5**	70.1 ± 0.6	$\mathbf{83.5 \pm 0.8}$
$100, 50$	**85.2**	49.8	**84.9**	70.3 ± 0.4	$\mathbf{85.1 \pm 0.6}$
$500, 5$	**405.8**	249.8	401.5	–	$\mathbf{408.1 \pm 4.9}$
$500, 10$	**437.8**	249.0	**432.5**	–	$\mathbf{432.4 \pm 3.5}$
$500, 50$	**463.7**	249.6	438.9	–	442.6 ± 2.5
$1000, 50$	**944.1**	499.8	847.43	–	889.8 ± 5.6

Table 2: Average reward per episode on a range of multi-armed bandit problems. We found that training MAML was computationally infeasible for $N = 500, 1000$.

N	Random	ϵ-greedy	PSRL	OPSRL	UCRL2	LSTM	MAML	SNAIL (ours)
10	0.482	0.640	0.665	0.694	0.706	0.752	0.563	$\mathbf{0.766 \pm 0.001}$
25	0.482	0.727	0.788	0.819	0.817	0.859	0.591	$\mathbf{0.862 \pm 0.001}$
50	0.481	0.793	0.871	0.897	0.885	0.902	–	$\mathbf{0.908 \pm 0.003}$
75	0.482	0.831	0.910	**0.931**	0.917	0.918	–	$\mathbf{0.930 \pm 0.002}$
100	0.481	0.857	0.934	**0.951**	0.936	0.922	–	0.941 ± 0.003

Table 3: Average normalized reward on randomly-generated MDPs. Similarly to the bandit problems, we were unable to successfully train MAML for $N = 50, 75, 100$. We compare the meta-learners to several human-designed algorithms (described in Appendix A).

Method	Small Maze Episode 1	Small Maze Episode 2	Large Maze Episode 1	Large Maze Episode 2
Random	188.6 ± 3.5	187.7 ± 3.5	420.2 ± 1.2	420.8 ± 1.2
LSTM	52.4 ± 1.3	39.1 ± 0.9	180.1 ± 6.0	150.6 ± 5.9
SNAIL (ours)	$\mathbf{50.3 \pm 0.3}$	$\mathbf{34.8 \pm 0.2}$	$\mathbf{140.5 \pm 4.2}$	$\mathbf{105.9 \pm 2.4}$

Table 4: Average time to find the goal on each episode in the small and large mazes. We didn't train MAML on this benchmark, due to the scalability challenges encountered during bandits and MDPs.

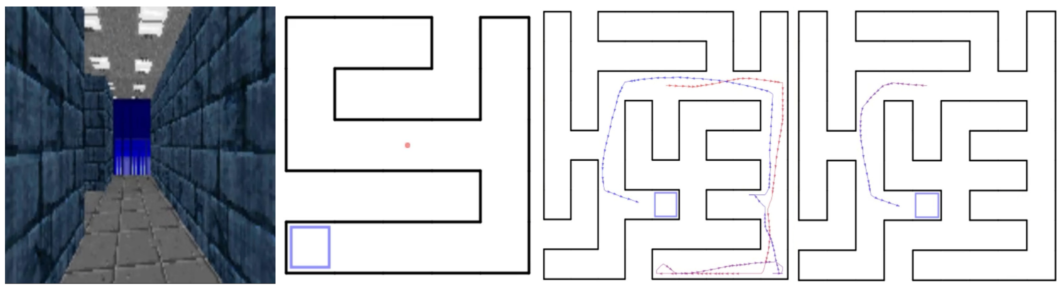


Figure 3: From left to right: (a) A (higher-resolution) example of the observations the agent receives. (b) An example of the mazes used for training (goal shown in blue). (c) The movement of the SNAIL on its first episode in a larger maze, exploring the maze until it finds the goal. (d) The SNAIL's path during its second episode in the same maze as (c). Remembering the goal location, it navigates there directly on the second episode. Maps like in (b), (c), (d) are used for visualization but not available to the agent. In (c), (d), the color progression from red to blue indicates the passage of time (red earlier).

References

[1] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[2] Chelsea Finn, Pieter Abbeel, and Sergy Levine. Model-agnostic meta learning. *International Conference on Machine Learning (ICML)*, 2017.

[3] J.C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1979.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[5] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 2010.

[6] Gregory Koch. *Siamese neural networks for one-shot image recognition*. PhD thesis, University of Toronto, 2015.

[7] Brenden M Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B Tenenbaum. One shot learning of simple visual concepts. In *CogSci*, 2011.

[8] Tsendsuren Munkhdalai and Hong Yu. Meta networks. *International Conference on Machine Learning (ICML)*, 2017.

[9] Ian Osband and Benjamin Van Roy. Why is posterior sampling better than optimism for reinforcement learning. *International Conference on Machine Learning (ICML)*, 2017.

[10] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations (ICLR)*, 2017.

[11] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning (ICML)*, 2016.

[12] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. *International Conference on Machine Learning (ICML)*, 2015.

[13] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *International Conference on Learning Representations (ICLR)*, 2016.

[14] Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*, 2017.

[15] Malcolm Strens. A bayesian framework for reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2000.

[16] Aaron van den Oord, Sander Dieleman, Heig Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[17] Ashish Vaswani, Noah Shazeer, Jakob Uszkoreit, Llion Jones, Aidan Gomez N., Lukas Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[18] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[19] 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.

Appendix

A Further Experiment Details

A.1 Few-Shot Classification

In the few-shot classification setting, we wish to classify data points into N classes when we only have a small number (K) of labeled examples per class. The Omniglot and mini-ImageNet datasets for few-shot image classification are the standard benchmarks in supervised meta-learning.

Introduced by Lake et al. [7], Omniglot consists of black-and-white images of handwritten characters gathered from 50 languages, for a total of 1632 different classes with 20 instances per class. Like prior works, we downsampled the images to 28×28 and randomly selected 1200 classes for training and 432 for testing. We performed the same data augmentation proposed by Santoro et al. [11], forming new classes by rotating each member of an existing class by a multiple of 90 degrees.

Mini-ImageNet is a more difficult benchmark; a subset of the well-known ImageNet dataset, it consists of 84×84 color images from 100 different classes with 600 instances per class. We used the split released by Ravi and Larochelle [10] and used by a number of other works, with 64 classes for training, 16 for validation, and 20 for testing.

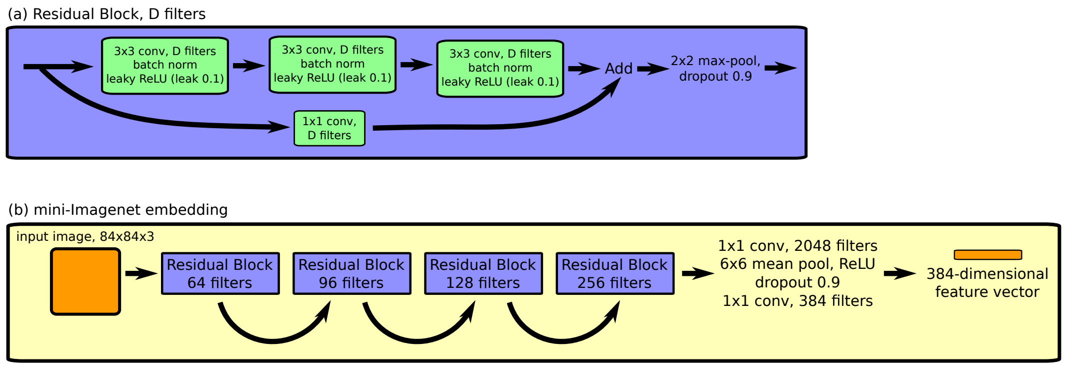
To evaluate a SNAIL on the N-way, K-shot problem, we sample N classes from the overall dataset and K examples of each class. We then feed the corresponding NK example-label pairs to the SNAIL in a random order, followed by a new, unlabeled example from one of the N classes. We report the average accuracy on this last, $(NK + 1)$-th timestep.

Using the building blocks defined in Section 1, we can concisely decribe the SNAIL architecture we used. For the N-way, K-shot problem, the sequence length is $T = NK + 1$, and we used the following for both Omniglot and mini-Imagenet. AttentionBlock(64, 32), TCBlock(T, 128), AttentionBlock(256, 128), TCBlock(T, 128), AttentionBlock(512, 256), followed by a final 1×1 convolution with N filters.

For the Omniglot dataset, we used the same embedding network architecture as all prior works, which repeat the following block four times { 3x3 conv (64 channels), batch norm, ReLU, 2x2 max pool }, and then apply a single fully-connected layer to output a 64-dimensional feature vector.

For mini-Imagenet, existing gradient-descent-based methods [10, 2], which update their model's weights during testing, used the same network structure as the Omniglot network but reduced the number of channels to 32, in spite of the significantly-increased complexity of the images. We found that this shallow embedding network did not make adequate use of SNAIL's expressive capacity, and opted to to use a deeper embedding network to prevent underfitting. Illustrated in Figure 4, our embedding was a smaller version of the ResNet [4] architectures commonly used for the full Imagenet dataset.

Figure 4: (a) A residual block within our mini-Imagenet embedding. (b) The embedding, a smaller version of ResNet [4], uses several of the residual blocks depicted in (a).



A.2 Multi-Armed Bandits

In our bandit experiments (styled after Duan et al. [1]), each of K arms gives rewards according to a Bernoulli distribution whose parameter $p \in [0, 1]$ is chosen randomly at the start of each episode of length N. At each timestep, the meta-learner receives previous timestep's reward, along with a one-hot encoding of the corresponding arm selected. It outputs a discrete probability distribution over the K arms; the selected arm is determined by sampling from this distribution.

The oracle we considered, the Gittins index [3], is the Bayes optimal solution in the discounted, infinite horizon setting. Since it is only optimal as $N \to \infty$, a meta-learner can outperform it for smaller N by choosing to exploit sooner.

For the N-timestep bandit problem, we used the following SNAIL architecture: First, we applied a fully-connected layer with 32 outputs that was shared between the policy and value function. Then the policy used: TCBlock(N, 32), TCBlock(N, 32), AttentionBlock(32, 32). The value function used: TCBlock(N, 16), TCBlock(N, 16), AttentionBlock(16, 16).

A.3 Tabular MDPs

In our tabular MDP experiments (also following Duan et al. [1]), each MDP had 10 states and 5 actions (both discrete); the reward for each (state, action)-pair followed a normal distribution with unit variance where the mean was sampled from $\mathcal{N}(1, 1)$, and the transitions are sampled from a flat Dirichlet distribution (the latter is a commonly used prior in Bayesian RL) with random parameters. We allowed the meta-learners to interact with an MDP for N episodes of length 10. As input, they received one-hot encodings of the current state and previous action, the previous reward received, and a binary flag indicating termination of the current episode.

In addition to a random agent, we consider the follow human-designed algorithms as baselines:

- PSRL [15]: a Bayesian method that estimate the belief over the current MDP parameters. At the start of each of the N episodes, it samples an MDP from the current posterior, and acts according to the optimal policy for the rest of the episode.
- OPSRL [9]: an optimistic variant of PSRL.
- UCRL2 [5]: uses an extended value iteration procedure to compute an optimistic MDP under the current belief.
- ϵ-greedy: with probability $1 - \epsilon$, act optimally against the MAP estimate according to the current posterior (which is updated once per episode).

The oracle we considered, value iteration, is optimal, when the MDP parameters (reward function, transition probabilities) are known; thus, its performance provides an upper bound on that of any algorithm, whether human-designed or meta-learned (which do not receive the MDP parameters). Note that since our episodes were truncated at 10 timesteps, we only ran value iteration for 10 iterations. The performance for each algorithm that we report in Table 2 is the average reward per episode, as a fraction of the reward achieved by value iteration. As an algorithm is allowed to interact with an MDP for more episodes, its normalized performance should approach 1, as the algorithm learns more about the current MDP.

We used the same SNAIL architecture as for bandits. Below, in Figure 5, we show learning curves of SNAIL and LSTM.

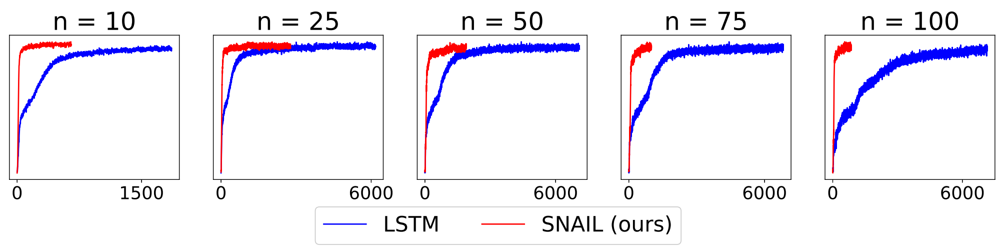


Figure 5: Learning curves of SNAIL (red) and LSTM (blue) on the random MDP task for different values of N. The horizontal axis is the TRPO iteration, and the vertical is average reward.

A.4 Continuous Control

We considered the set of tasks introduced by Finn et al. [2], in which two simulated robots (a planar cheetah and a 3D-quadruped ant) have to run in a particular direction or at a specified velocity (the direction or velocity are chosen randomly and not told to the agent). For each of these four task distributions ($\{$ant, cheetah$\} \times \{$goal velocity, goal direction$\}$), Finn et al. [2] trained a policy to maximize its performance after one policy gradient update using 20 episodes (40 for ant), of 200 timesteps each, on a newly sampled task.

In the goal direction experiments, the reward is the magnitude of the robot's velocity in either the forward or backward direction, and in the goal velocity experiments, the reward is the negative absolute value between its current forward velocity and the goal. The observations are the robot's joint angles and velocities, and the actions are its joint torques.

Since SNAIL and LSTM do not update their parameters at test time (instead incorporating experience through their hidden state), they receive as input the previous action, previous reward, and an episode-termination flag in addition to the current observation. We found that two episodes of interaction was sufficient for these meta-learners to adapt to a task, and that unrolling them for longer did not improve performance.

For all task distributions, the total trajectory length was $T = 400$ (2 episodes of 200 timesteps each). We used the same architecture (shared between policy and value function) for all tasks: two fully-connected layers of size 256 with tanh nonlinearities, AttentionBlock(32, 32), TCBlock(T, 16), TCBlock(T, 16), AttentionBlock(32, 32). Then the policy and value function applied separate fully-connected layers to produce the requisite output dimensionalities.

A.5 Visual Navigation

Both Duan et al. [1] and Wang et al. [19] consider the task of visual navigation, where the agent must find a target in a maze using only visual inputs. The former used randomly-generated mazes and target positions, while the latter used a fixed maze and only four different target positions. Hence, we evaluated SNAIL on the former, more challenging task. The observations the agent receives are 30×40 first-person images, and the actions it can take are $\{$step forward, turn slightly left, turn slightly right$\}$. We constructed a training dataset and two test datasets (unseen mazes of the same and larger size, respectively), each with 1000 mazes. The agents were allowed to interact with each maze for 2 episodes, with episode length 250 (1000 in the larger mazes). The starting and goal locations were chosen randomly for each trial but remained fixed within each pair of episodes. The agents received rewards of +1 for reaching the target (which resulted in the episode terminating), -0.01 at each timestep, to encourage it to reach the goal faster, and -0.001 for hitting the wall.

We preprocessed the images using the same convolutional architecture as Duan et al. [1]: two layers with $\{$kernel size 5×5, 16 filters, stride 2, ReLU nonlinearity$\}$, whose output is then flattened and then passed to a fully-connected layer to produce a feature vector of size 256.

The total trajectory length was $T = 500$ (2 episodes of 250 timesteps each). For the policy, we used: TCBlock(T, 32), AttentionBlock(16, 16), TCBlock(T, 32), AttentionBlock(16, 16). For the value function we used: TCBlock(T, 16), TCBlock(T, 16).

A.6 Additional Reinforcement Learning Hyperparameters

As discussed in Section 2, we trained all policies using trust-region policy optimization with generalized advantage estimation (TRPO with GAE, Schulman et al. [12, 13]). The hyperparameters are listed in Table 7. For multi-armed bandits, tabular MDPs, and visual navigation, we used the same hyperparamters as Duan et al. [1] to make our results directly comparable; additional tuning could potentially improve SNAIL's performance.

Hyperparameter	Multi-armed Bandits	Tabular MDPs	Continuous Control	Visual Navigation
Batch Size (timesteps)	250K	250K	50K	50K
Discount	0.99	0.99	0.99	0.99
GAE λ	0.3	0.3	0.97	0.99
Mean KL	0.01	0.01	0.01	0.01

Table 5: The TRPO + GAE hyperparameters we used in our RL experiments.