Sparse Distributed Memory for Sparse Rewards

Honour's Thesis Defense

Alex Van de Kleut

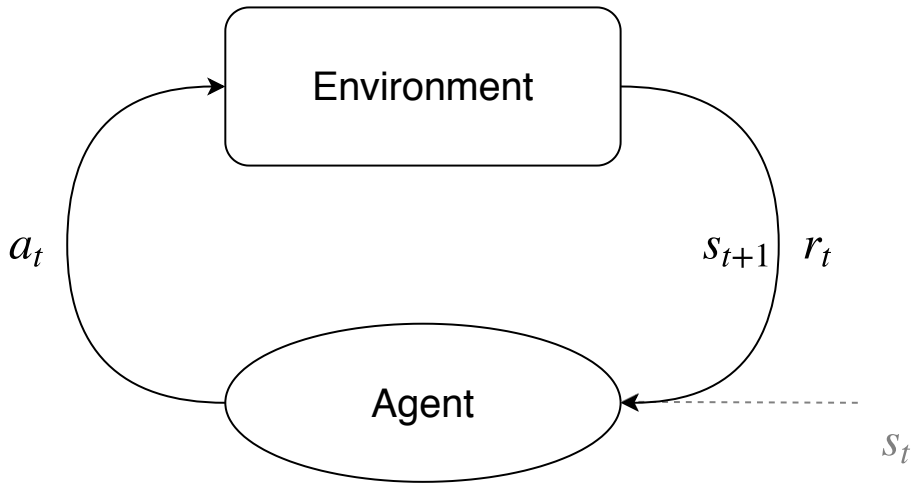
Brock University

May 9, 2019

Reinforcement Learning

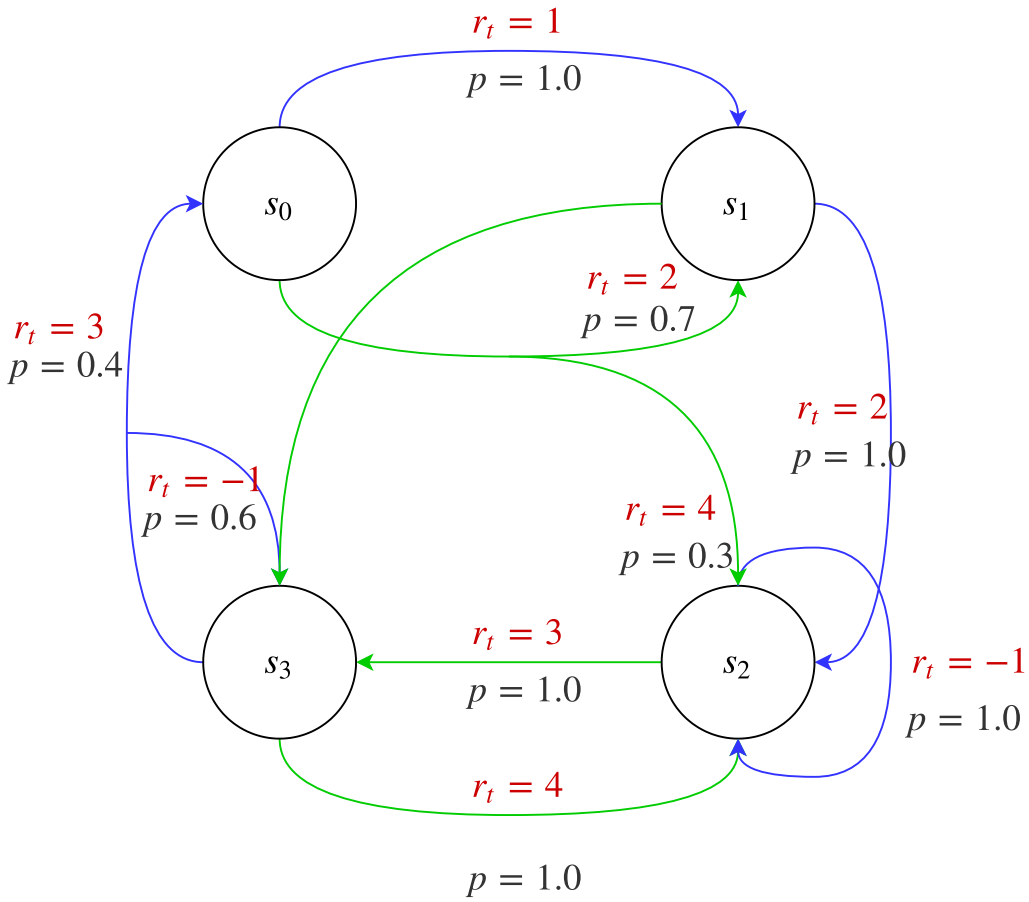
Context:

- Agent
- Environment
- $\langle s_t, a_t, r_t, s_{t+1} \rangle$



Markov Decision Process

- $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$
- $\mathcal{P}(s_t, a_t, s_{t+1}) = \mathbb{P}\left[s_{t+1}|s_t, a_t\right]$
- $\mathcal{R}(s_t, a_t) = \mathbb{E}\left[r_t|s_t, a_t\right]$



Trajectories, Returns, Policies

- Trajectory: $\tau = \langle s_0, a_0, r_0, s_1, a_1, r_1, \dots \rangle$
- Return:
  - $R_t = \sum_{k=t}^{\infty} r_k$
    - ★ Not guaranteed to converge
  - $G_t = \sum_{k=t}^{\infty} \gamma^{t-k} r_k$
    - ★ Guaranteed to converge
- Policy: $\pi : \mathbb{P}[a_t | s_t]$

The Reward Hypothesis

Every action of a rational agent can be thought of as seeking to maximize some cumulative reward signal.

Goal

Learn a policy π that chooses actions $a \sim \pi(\cdot | s_t)$ that maximize G_t

Value, Quality, Q-learning

Value

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[G_t|s_t\right] = \mathbb{E}_{\pi}\left[r_t + \gamma V^{\pi}(s_{t+1})|s_t\right]$$

Quality

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\left[G_t|s_t, a_t\right] = \mathbb{E}_{\pi}\left[r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1})|s_t, a_t\right]$$

In practice, V^{π} and Q^{π} are unknowable, so we approximate it with some functions V and Q.

Q-learning

$$\pi(a_t|a_t) = \begin{cases} 1 & a_t = \arg\max_{a_t} Q(s_t, a_t) \\ 0 & \text{otherwise} \end{cases}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Deep Q-learning

- It is unfeasible to store $Q(s_t, a_t)$ when $\mathcal{S}$ is large or infinite.
- Instead, we use a neural network Q_θ as a function approximator for Q
- Produces a vectorized output for predicted Q over each action
  - Requires discrete $\mathcal{A}$
- Q-network minimizes prediction error between $Q_\theta(s_t, a_t)$ and $r_t + \gamma Q_\theta(s_{t+1}, a_{t+1})$ using **gradient descent**:

Deep Q-learning

$$L(\theta) = \mathbb{E}_{a_t \sim \pi}\left[(y_i - Q_\theta(s_t, a_t))^2\right]$$

$$y_i = \mathbb{E}_{a_{t+1} \sim \pi}\left[r_t + \gamma \max_{a_{t+1}} Q_\theta(s_{t+1}, a_{t+1}) | s_t, a_t\right]$$

$$\theta \leftarrow \theta - \frac{1}{2}\alpha \nabla_\theta L(\theta)$$

Policy Gradient

- Deep Q-learning uses discrete $\mathcal{A}$ and uses a greedy policy.
- Policy gradients model the policy using a set of parameters θ
- Policy gradients directly optimize the policy, allow for stochastic policies, and allow continuous action spaces.

$$\pi_\theta(a_t|s_t) = \mathbb{P}\left[a_t|s_t;\theta\right] \tag{1}$$

Softmax Distribution

Output mapped to probability distribution

$$a_i \rightarrow \frac{e^{a_i}}{\sum_j e^{a_j}}$$

Diagonal Gaussian Distribution

Output treated as mean of continuous action space

$$a_t = \mu_\theta(s_t) + z \odot \sigma$$

Policy Gradient Theorem

We define an objective function $J(\theta)$ that we want our policy to maximize. If π_θ is differentiable, we can use **gradient ascent**:

Policy Gradient Theorem

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{2}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t) \right]$$

Full derivation in thesis.

REINFORCE, Actor-Critic

REINFORCE

Since policy gradient theorem is an expectation, we can sample this estimation to get approximate the gradient.

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$$

Actor-Critic

Monte-carlo methods like REINFORCE subject to high variance.

- Actor: The policy π_θ
- Critic: A deep neural network $\mathcal{S} \rightarrow \mathbb{R}$ that learns to approximate $V(s_t)$ in a manner analogous to deep Q-learning.

Advantage, Generalized Advantage Estimation

The Q^{π_θ} term in the policy gradient can have high variance. We can subtract a baseline $b(s)$ without changing the expectation.

Advantage

$$A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$$

Generalized Advantage Estimation

$$\hat{A}_t = \sum_{i=0}^{\infty} (\gamma\lambda)^i \delta^{\pi_\theta}_{t+i} \qquad \delta^{\pi_\theta}_t = r_t + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)$$

rather than two sets of parameters to approximate $Q^{\pi_\theta}(s_t, a_t)$ and $V^{\pi_\theta}(s_t)$, we use one: V_ϕ. Then V_ϕ we update ϕ to minimize

$$d_t(\phi) = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$$

and update θ to maximize $\hat{A}_t$.

Proximal Policy Optimization (PPO)

PPO

Modified objective function based on direction of policy updates.

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\rho(\theta) A^{\pi_\theta}(s, a)\right]$$

where

$$\rho(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$$

When $\rho(\theta) > 1$ then we increased the likelihood of $\pi_\theta(a_t|s_t)$.

	$\rho(\theta) > 1$	$\rho(\theta) < 1$
$A^{\pi_\theta}(s, a) > 0$	Good (clip)	Bad (roll back)
$A^{\pi_\theta}(s, a) < 0$	Bad (roll back)	Good (clip)

- Clipping means policy updates are conservative
- Allows us to do multiple epochs of policy updates on same batch of training experience (faster learning)

Sparse Rewards

- RL works best when rewards are **dense** (mostly nonzero).
- RL struggles when rewards are **sparse**
- Requires **intrinsic motivation** - method for agent to reward itself
  - What is rational behaviour without a clear goal?
  - Exploration, curiosity, etc.
  - Design intrinsic reward to encourage these behaviours to increase likelihood of finding sparse extrinsic rewards.

Count-Based

Methods for keeping track of how often certain states have been visited, with reward inversely proportional to visitation frequency.

Forward-Dynamics

Try to predict next state given current state and action, with reward proportional to prediction error. Subject to noisy tv problem.

Random Network Distillation

Novel approach to intrinsic motivation that achieves state-of-the-art performance on Montezuma's Revenge, a sparse reward environment.

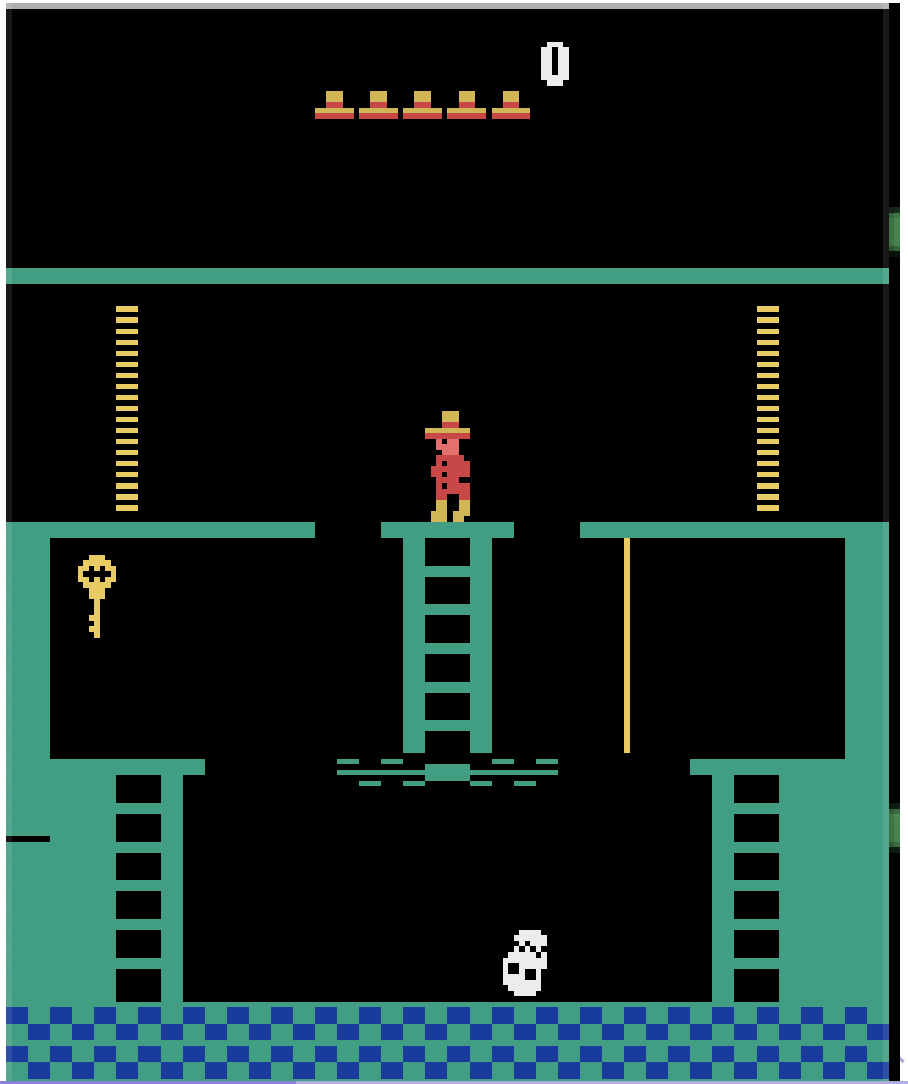
RND

Two neural networks

- Feature network: $\mathcal{S} \rightarrow \mathbb{R}^k$
- Predictor network: same architecture as feature network with different initialization. Learns to predict output of feature network.

Commonly visited state will have lower prediction error. Reward proportional to prediction error.

Poor sample efficiency! Takes 1.6×10^9 frames to achieve SOTA

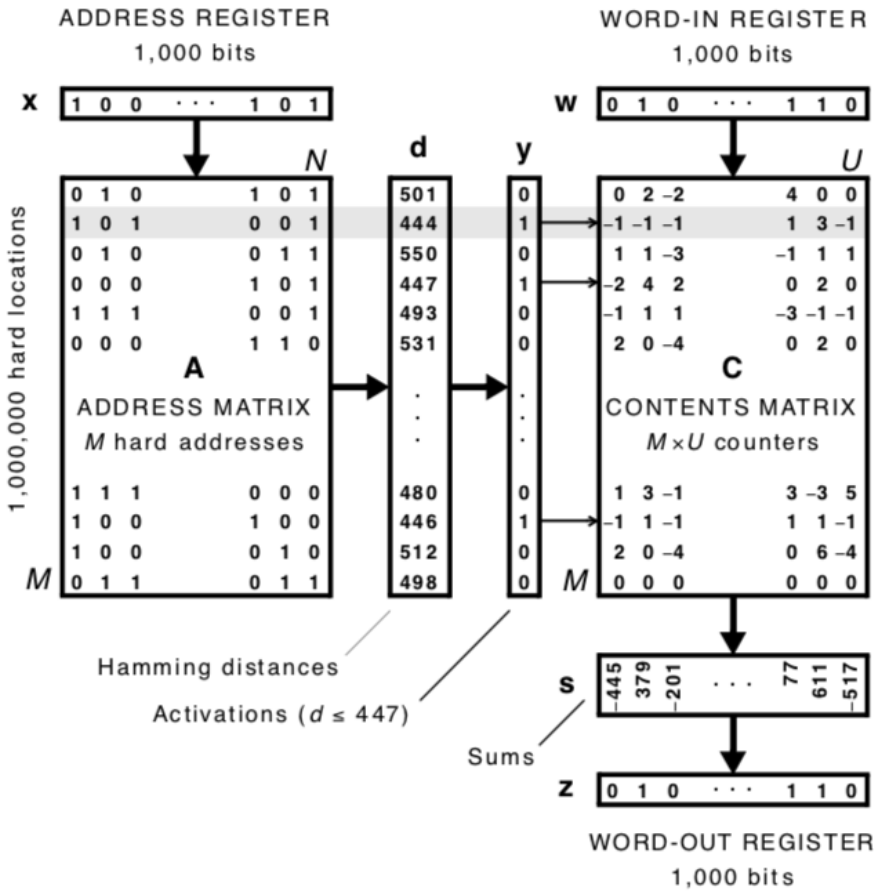


Sparse Distributed Memory

How can we have an intrinsic reward signal like RND that works immediately?

SDM

- Computer memory for long ($>$ 100-bit) words
- Small subset of address space implemented in hardware
- Reading and writing distributed to addresses 'close' to the one specified
- **autoassociative memory**

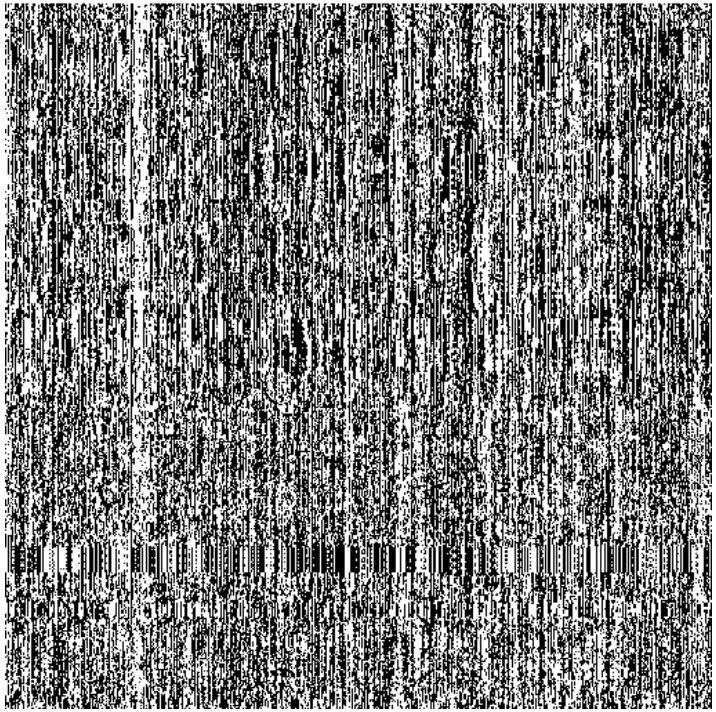


Hashing

Data written to/read from memory is hashed state: $\text{Hash} : \mathcal{S} \to \{0,1\}^N$

- We want hashes for similar states to be similar.
- We can use a neural network with the same architecture as the feature network from RND.
- If that network's output is o, then we can define the following hash:

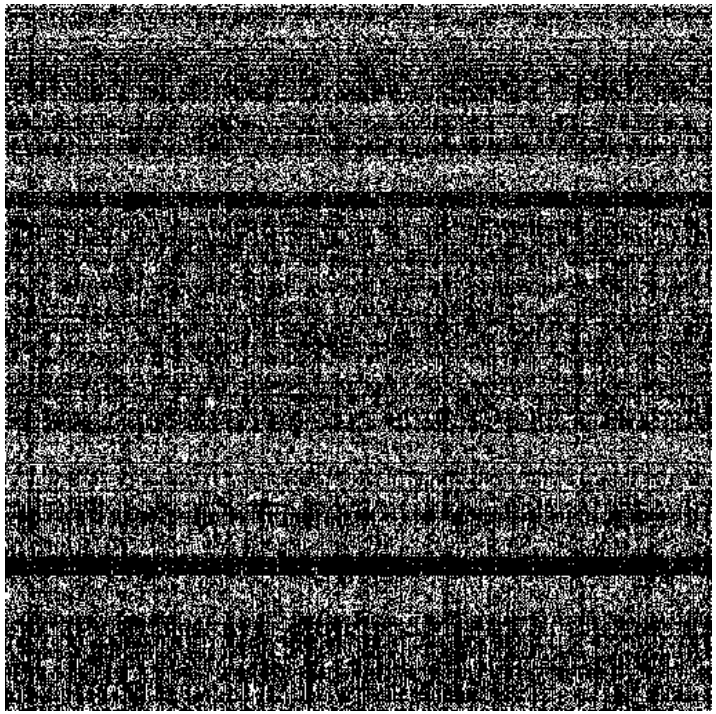
$$\text{Hash}(s_t)_i = \begin{cases} 1, & o_i \geq \bar{o} \\ 0, & o_i < \bar{o} \end{cases} \tag{3}$$



Proof of Concept

- Agent is in state s_t and computes a hash $\mathbf{w}_t$ of the state
- Agent checks memory at address $\mathbf{w}_t$ and retrieves stored binary vector $\mathbf{z}_t$
- Agent computes intrinsic reward by taking scaled hamming distance between $\mathbf{w}_t$ and $\mathbf{z}_t$
- Agent stores $\mathbf{w}_t$ to memory

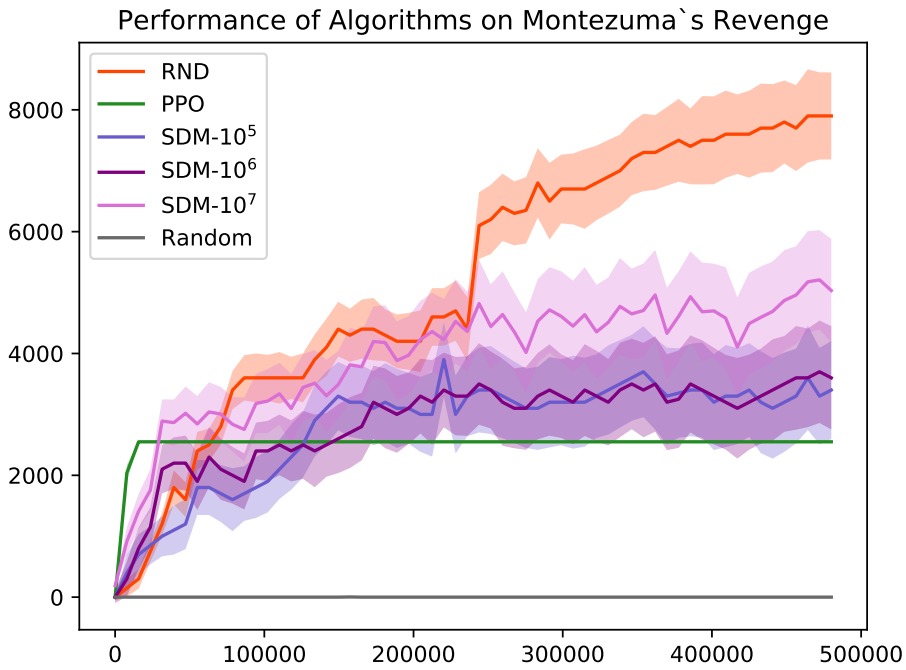
$$r_t^I = \frac{d_H(\mathbf{w}_t, \mathbf{z}_t)}{N}$$



Pseudocode

$N \leftarrow$ number of rollouts
$N_{\text{opt}} \leftarrow$ number of optimization epochs
$K \leftarrow$ length of rollout
$t = 0$
sample s_0 from $d(s)$
for $i = 1$ to N **do**:
    for $j = 1$ to K **do**:
        sample $a_t \sim \pi_\theta(s_t)$
        sample s_{t+1}, r_t^E from $\mathcal{P}, \mathcal{R}$
        normalize state $s_t \leftarrow \frac{s_t - \mu}{\sigma}$
        compute $\mathbf{w} = \text{Hash}(s_t)$
        compute intrinsic reward r_t^I
        normalize intrinsic reward
        store s_t, a_t, r_t^E, r_t^I to optimization batch B_i
        $t = t + 1$
    compute A_E and A_I for batch B_i using GAE
    compute total advantage $A = c_E A_E + c_I A_I$
    for $j = 1$ to N_{opt} **do**:
        optimize policy using PPO

Results



M	Cumulative Reward	RND-Normalized	PPO-Normalized	Random-Normalized
10^5	3607 ± 651	45.82%	141.45%	13035%
10^6	3553 ± 428	45.15%	139.33%	17765%
10^7	4251 ± 399	54.01%	166.71%	21255%
RND	7871 ± 676			
PPO	2550 ± 0			
Random	20 ± 40			

Results

- SDM improves performance over PPO baselines (significant improvement, $p < 0.05$)
- SDM competitive with RND up to around halfway through training
  - RND finds room with easy access to many other rooms
- Some increase in performance associated with increasing memory size to $M = 10^7$
- All methods perform significantly better than random
- Overall: SDM plays at amateur human level

Discussion

- SDM improvement over PPO demonstrates that using intrinsic reward can improve exploration and increase likelihood of finding sparse extrinsic rewards
- Rate at which intrinsic reward dissipates is much faster for SDM than for RND
  - Originally the motivation for this technique, actually hampered the agent's learning
  - Intrinsic rewards become sparse quickly for common states
  - No extrinsic *or* intrinsic rewards
- Intrinsic reward disappears completely around $300,000$ parameter updates: agent is learning using only PPO at this point

Future Work

1. Compare RND to SDM in a complex state space (such as a 3D environment)
   - Possible that RND networks take too long to learn since inputs vary significantly more than with Montezuma's Revenge
   - Instanteneity of SDM becomes more important here
2. Write schedule that writes only when states are novel rather than every time step
   - Reduce rate of reward evaporation
3. Used a discrete action space environment here:
   - Extend intrinsic reward methods to deep Q-learning rather than just policy gradient methods?

Conclusion

- Reinforcement learning struggles when rewards are sparse
- SDM improves performance in Montezuma's Revenge over baseline performance by introducing memory-based intrinsic reward
- SDM's performance does not achieve state-of-the-art