Learning to play SLITHER.IO with deep reinforcement learning

Joan Creus-Costa & Zhanpei Fang, {jcreus, zhanpei}@stanford.edu
CS229: Machine Learning

Introduction and motivation

This project uses deep reinforcement learning (RL) to train an agent to play the massively multiplayer online game (MMO) SLITHER.IO, in which players attempt to maximize "snake" length by consuming multicolored food pellets while avoiding other players' snakes. The goal of the project was to achieve human-like performance on the game, which presents a good target due to the relative simplicity of game mechanics and the possibility of complex emergent behavior.

Introduced in 2015, deep Q-learning has been applied with great success to a wide variety of gameplay scenarios, such as Atari games Mnih et al. [2015]. By taking in raw image inputs from the online game, we seek to learn an approximation to the Q function and thereby allow us to find a policy.

Dataset and features

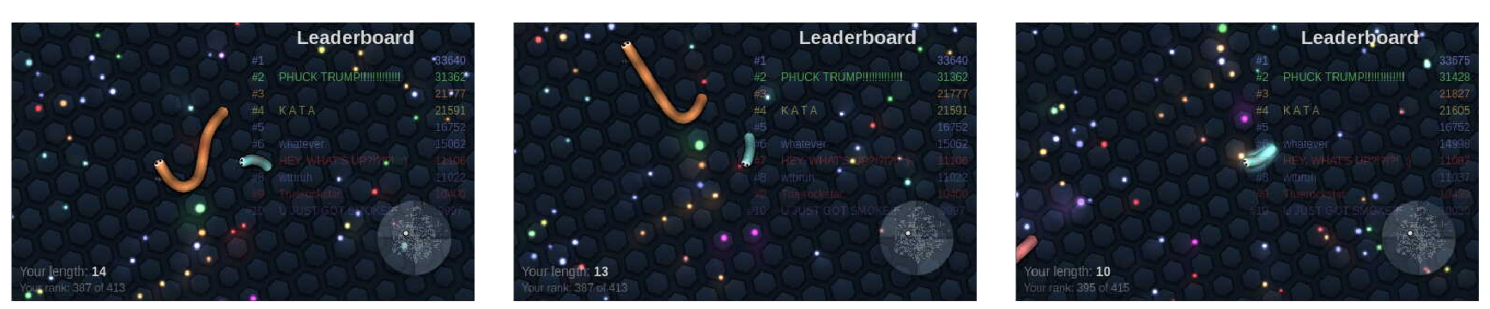
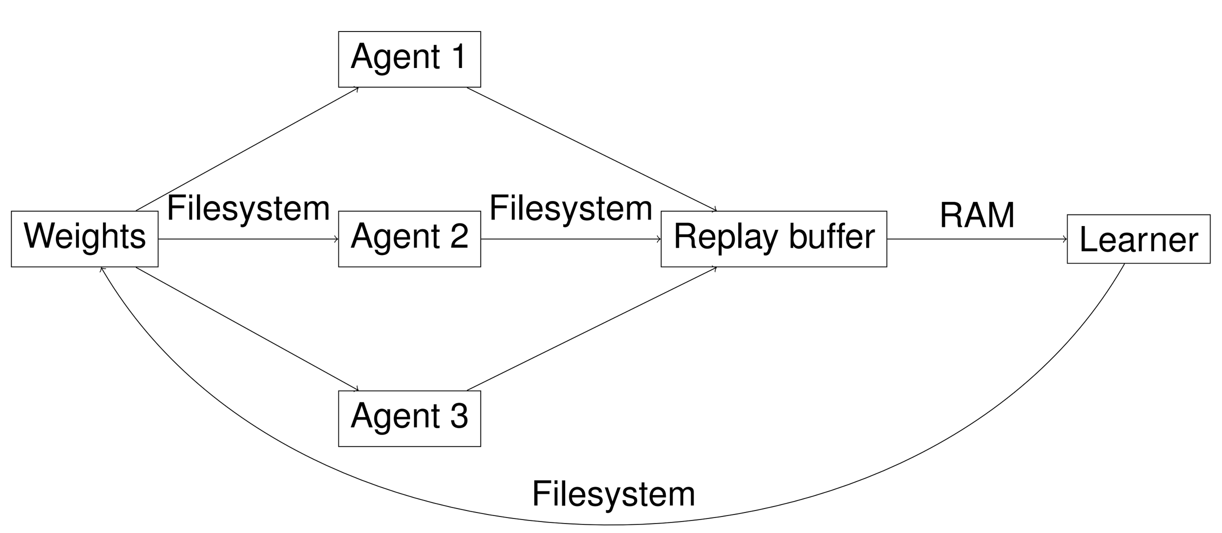


Figure: Sample (cropped) frames from a random agent playing SLITHER.IO.

We collect training data using an OpenAI Universe environment, which interacts with the online game via a simulated screen and virtual keyboard. Our inputs are the raw frames from the remote desktop, and reward signals corresponding to change in score (length of snake).

Pre-processing steps include:

- Crop and downsize image, from $768 \times 1024 \times 3$ pixel RGB image $\rightarrow 150 \times 250 \times 3$ RGB pixels.
- Frame skipping from $60 \rightarrow 5$ frames per second
- 8-bit color range (0–255) $\rightarrow$ floating-point values (0–1).
- Encode difference from previous frame by computing Δ(previous frame $-$ current frame) for each channel, for a total of $150 \times 250 \times 6$ input features.



Model: Deep Q-Learning

Q-learning learns the action-value function $Q(s, a)$ from pixels s:

$$Q^*(s, a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s', a')|s, a\right]$$

Q-learning update at iteration i uses Huber loss: $L_\delta(a) = \frac{a^2}{2}$ if $|a| \leqslant \delta$, $\delta|a| - \frac{\delta^2}{2}$ otherwise.

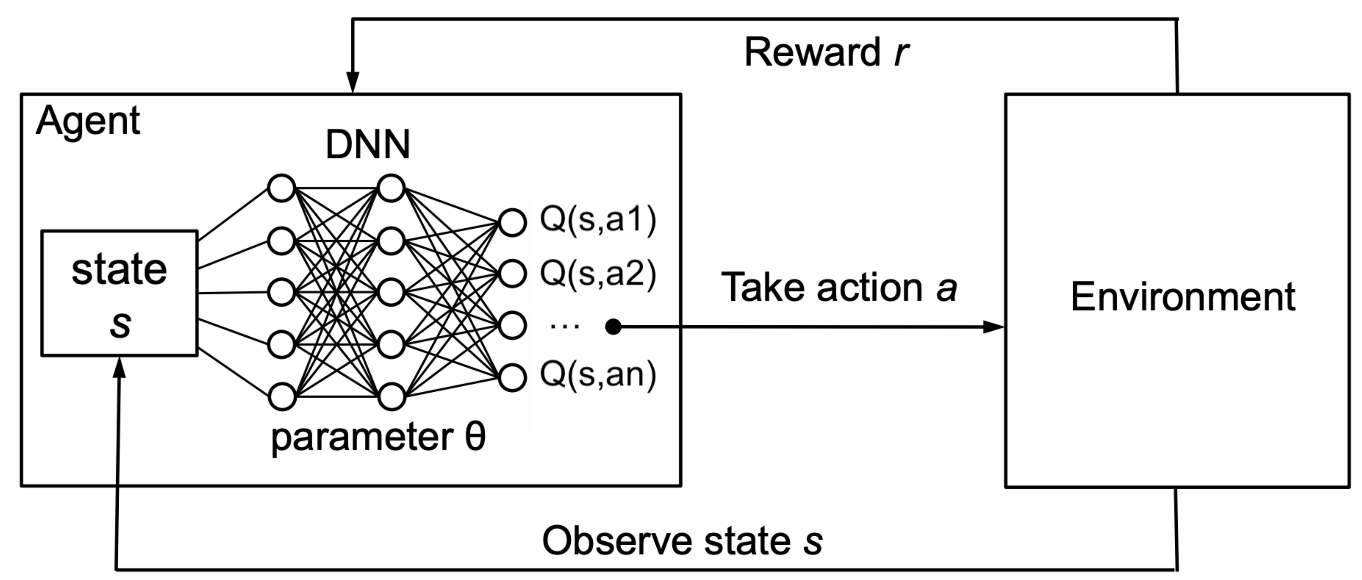


Figure: Deep-Q network for reinforcement learning. Modified from Mao et al. [2016].

A variety of tricks were necessary to be able to learn a policy:

- Learn from human demonstrations to faciliate training, as in [Hester et al., 2018]. We collected around 50 demonstrations to pretrain the network. We add the following term to the cost function
$$L_E(Q) = \max_{a \in A}\left[Q(s, a) + l(a_e, a)\right] - Q(s, a_E)$$
where $l(a_e, a)$ is a margin function that's 0 when the actions are equal.
- Reward shaping disincentivizes ending the episode [Ng et al., 1999]:
$$r'_t = \begin{cases} -10 & T - t \leqslant 10 \\ r_t & \text{otherwise} \end{cases}$$
- Prioritize experience replay to sample transitions with or near a reward to compensate for sparsity of rewards and mitigate instability.

Results

Model	Median score*	Average reward
Random policy	3^{+1}_{-0}	0.08
Human†	145^{+36}_{-38}	0.68
No human demonstrations, ϵ-greedy, $K = 1.5 \times 10^5$ batches	17^{+1}_{-8}	0.10
Pretrain on human demonstrations, $K_1 = 1.5 \times 10^4$	11^{+6}_{-3}	0.14
$\cdots$ followed by $K_2 = 3 \times 10^4$ collected batches	54^{+16}_{-22}	0.26

* Estimated 68% confidence interval using the bootstrap; using median instead of mean to avoid bias from outliers. † Human performance was tested in the simulated environment instead of the website, and was therefore worse than normal play due to the environment limitations (low resolution, low frame-rate).

- Our best model performs significantly better than a random policy, but noticeably worse than a human benchmark.
- Training was complicated and unstable; used $\alpha_{\text{Adam}} = 10^{-4}$, $B = 128$, $\gamma = 0.95$. Higher γ or α resulted in worse policies—we suspect due to more difficult convergence.

Discussion

- Realtime Internet playing introduces significant bottleneck to training. Combined with the complexity of the game, this results in high difficulty in training a better-than-random policy.
- Using human demonstrations results in a tremendous speedup in training as well as in a much better model. 15,000 pretraining steps yields a better model than 150,000 ϵ-greedy steps; combining both yields the best model in 45,000 iterations.
- Reward shaping and prioritized replay also helped get most of the (expensive) transitions to acquire. Using PyTorch's CUDA integration allows for significantly faster training.
- Limited model capacity since we only use a single delta frame, and implement vanilla DQN.

Future work

- Switch to better RL methods, given that more training with the current model did not yield significant improvements; such as recurrent network architecture, or actor-critic models or Rainbow agents which have also shown promise in similar environments.
- Use cloud infrastructure to allow for faster training to avoid the limitations of the network-based game. Use asynchronous methods (e.g. A3C) to distribute parameters.

Selected references:

- T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, Horgan, et al. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018
- V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, et al. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*, 2013
- A. Paszke, S. Gross, S. Chintala, and G. Chanan. PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration. *PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration*, 6, 2017
- G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, et al. OpenAI Universe, 2016
- A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999
- Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015