Distributed Deep Q-Learning

Hao Yi Ong, Kevin Chavez, and Augustus Hong

CME 323, Stanford University

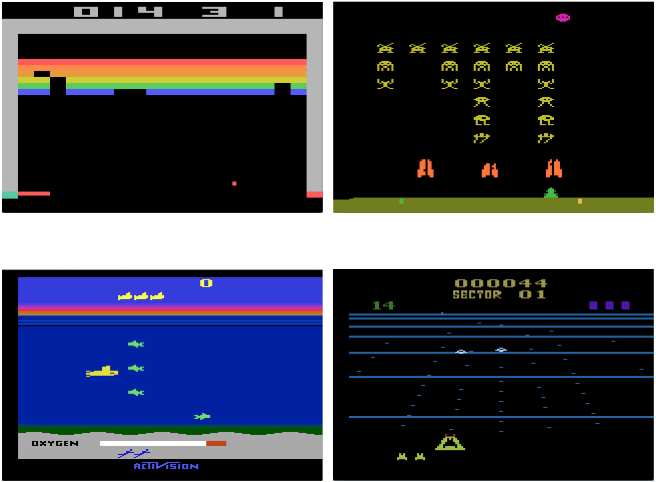
June 3, 2015

Outline

Motivation

- long-standing challenge of reinforcement learning (RL)
  - control with high-dimensional sensory inputs (*e.g.*, vision, speech)
  - shift away from reliance on hand-crafted features
- utilize breakthroughs in deep learning for RL [M^+13, M^+15]
  - extract high-level features from raw sensory data
  - learn better representations than handcrafted features with neural network architectures used in supervised and unsupervised learning
- create fast learning algorithm
  - train efficiently with stochastic gradient descent (SGD)
  - distribute training process to accelerate learning [DCM^+12]

Success with Atari games



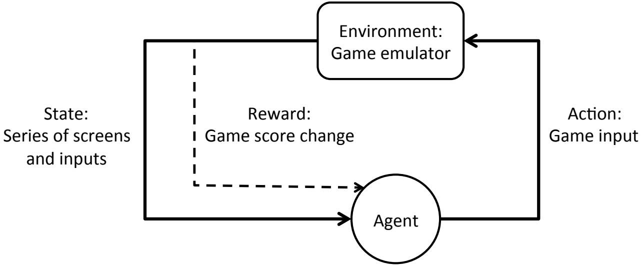
Goals

distributed deep RL algorithm

- robust neural network agent
  - must succeed in challenging test problems
- control policies with high-dimensional sensory input
  - obtain better internal representations than handcrafted features
- fast training algorithm
  - efficiently produce, use, and process training data

Outline

Playing games



objective: learned policy maximizes future rewards

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'},$$

- discount factor γ
- reward change at time t' $r_{t'}$

State-action value function

- basic idea behind RL is to estimate

$$Q^{\star}(s,a) = \max_{\pi} \mathbf{E}\left[R_t \mid s_t = s, a_t = a, \pi\right],$$

where π maps states to actions (or distributions over actions)

- optimal value function obeys Bellman equation

$$Q^{\star}(s,a) = \mathop{\mathbf{E}}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^{\star}(s',a') \mid s,a\right],$$

where $\mathcal{E}$ is the MDP environment

Q-network

- trained by minimizing a sequence of loss functions

$$L^{(i)}\left(\theta^{(i)}\right) = \mathop{\mathbf{E}}_{s,a\sim\rho(\cdot)}\left[\left(y^{(i)} - Q\left(s, a; \theta^{(i)}\right)\right)^2\right],$$

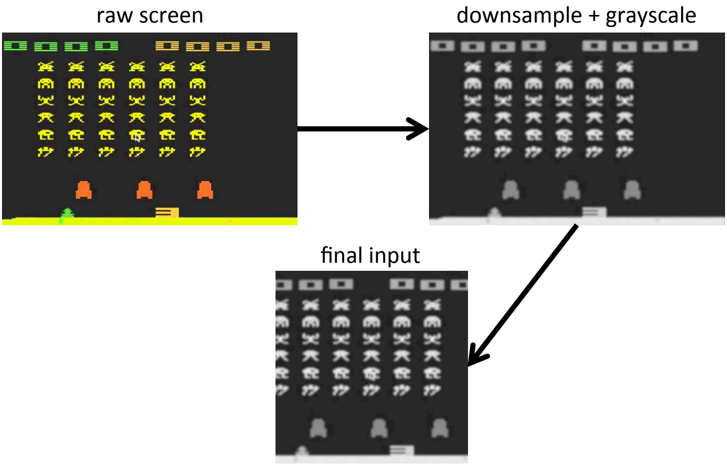
  with

  - iteration number i, ith network parameters $\theta^{(i)}$
  - target $y^{(i)} = \mathbf{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q\left(s', a'; \theta^{(i-1)}\right) \mid s, a\right]$
  - "behavior distribution" (exploration policy) $\rho\left(s, a\right)$

- architecture varies according to application

Outline

Preprocessing



Q-learning

- optimize Q-network loss function via

$$Q\left(s,a\right) := Q\left(s,a\right) + \alpha\left(r + \gamma \max_{a'} Q\left(s',a'\right) - Q\left(s,a\right)\right)$$

- trains optimal policy using "behavior policy" (off-policy)
  - learns policy $\pi^{\star}\left(s\right) = \mathrm{argmax}_a\, Q\left(s,a;\theta\right)$
  - uses an ϵ-greedy strategy (behavior policy) for state-space exploration

Experience replay

a kind of short-term memory

- store agent's experiences at each time step

$$e_t = (s_t, a_t, r_t, s_{t+1})$$

- experiences form a replay memory dataset

$$\mathcal{D} = \{e_1, \ldots, e_N\},$$

  where N is the fixed memory capacity

- execute Q-learning updates with samples of experience

$$e \sim \mathcal{D}$$

Serial deep Q-learning

given replay memory $\mathcal{D}$ with capacity N

initialize Q-networks Q, $\hat{Q}$ with same random weights θ

repeat until timeout

initialize frame sequence $s_1 = \{x_1\}$ and preprocessed state $\phi_1 = \phi\left(s_1\right)$

for $t = 1, \ldots, T$

1. select action $a_t = \begin{cases} \max_a Q\left(\phi\left(s_t\right), a; \theta\right) & \text{w.p. } 1 - \epsilon \\ \text{random action} & \text{otherwise} \end{cases}$
2. execute action a_t and observe reward r_t and frame x_{t+1}
3. append $s_{t+1} = \left(s_t, a_t, x_{t+1}\right)$ and preprocess $\phi_{t+1} = \phi\left(s_{t+1}\right)$
4. store experience $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in $\mathcal{D}$
5. uniformly sample minibatch $\left(\phi_j, a_j, r_j, \phi_{j+1}\right) \sim \mathcal{D}$
6. set $y_j = \begin{cases} r_j & \text{if } \phi_{j+1} \text{ terminal} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta\right) & \text{otherwise} \end{cases}$
7. perform gradient descent step for Q on minibatch
8. every C steps reset $\hat{Q} = Q$

Outline

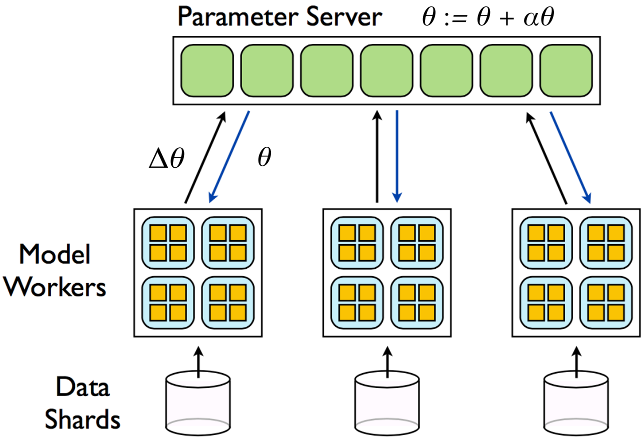
Model parallelism

for each Q-network

- partition model across CPUs/GPUs
  - up to availability of CPU/GPU resources
  - uses Caffe deep learning framework
- **[[Kevin/Hao Yi: How does caffe use CPU/GPU resources? How does complexity scale for implementation? Answers question of how our algorithm scale for model.]]**

Data parallelism

downpour SGD: generic asynchronous distributed SGD



Implementation

- data shards are generated locally on each model worker in real-time
  - data is stored independently for each worker
  - since game emulation is simple, generating data is fast
  - simple fault tolerance approach: regenerate data if worker dies
- algorithm scales very well with data
  - since data lives locally on workers, no data is sent

Implementation

- bottleneck is parameter update time on parameter server
  - *e.g.*, if parameter server gradient update takes 2 ms, then we can only do up to 500 updates per second (using buffers, etc.)
- trade-off between parallel updates and model staleness
  - because worker is likely using a stale model, the updates are “noisy” and not of the same quality as in serial implementation

Implementation

communication pattern

- one-to-all and all-to-one, but asynchronous for every minibatch
- like multiple asynchronous all-reduces

Outline

Evaluation

Snake

- parameters
  - snake length grows with number of apples eaten
  - one apple at any time, regenerated once eaten
  - $n \times n$ array, with walled-off world
  - want to maximize score, equal to snake length
- complexity
  - four possible states for each cell: $\{$empty, head, body, apple$\}$
  - state space cardinality is $O\left(n^8\right)$
  - four possible actions: $\{$north, south, east, west$\}$

Results

Outline

Summary

References

- Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al.
  Large scale distributed deep networks.
  In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.
- V. Mnih et al.
  Playing Atari with deep reinforcement learning.
  arXiv preprint arXiv:1312.5602, 2013.
- V. Mnih et al.
  Human-level control through deep reinforcement learning.
  Nature, 518(7540):529–533, 2015.

Theoretical complications

deep learning algorithms require

- huge training datasets
- independence between samples
- fixed underlying data distribution

Deep Q-learning

avoids theoretical complications

- greater data efficiency
  - each experience potentially used in many weight udpates
- reduce correlations between samples
  - randomizing samples breaks correlations from consecutive samples
- experience replay averages behavior distribution over states
  - smooths out learning
  - avoids oscillations or divergence in gradient descent