Distributed Deep Q-Learning

Hao Yi Ong, Kevin Chavez, and Augustus Hong

CME 323, Stanford University

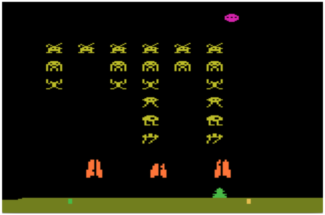
June 3, 2015

Outline

Motivation

- long-standing challenge of 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
  - extract high-level features from raw sensory data
  - learn better representations than handcrafted features with neural network architectures used in supervised and unsupervised learning
  - train efficiently with stochastic gradient descent

Playing Atari



Deep reinforcement learning

- connecting RL with deep neural networks
  - reinforcement learning approach to obtain control policies
  - use high-dimensional sensory input
- practical considerations
  - differences between supervised/unsupervised deep learning and RL
  - algorithm must learn like a human player

Theoretical complications

- deep learning usually requires huge hand-labeled training datasets
  - sparse, noisy, and delayed reward signal in RL
  - delay of $\sim 10^3$ time steps between actions and resulting rewards
  - *cf.* direct association between inputs and targets in supervised learning

Theoretical complications

most deep learning algorithms assume

- independence between data samples
  - sequences of highly correlated states in RL problems
- fixed underlying data distribution
  - distribution changes as RL algorithm learns new behaviors

Emulating human learning

- neural network not provided
  - game-specific information
  - hand-designed visual features
  - internal state of emulator
- generalizing across games with the same
  - network architecture
  - training algorithm hyperparameters

Goals

first deep RL model

- single game-agnostic, robust neural network agent
  - must succeed in various Atari 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

Prior work

TD-gammon [Tes95]

- backgammon-playing program learned by RL and self-play
- model-free RL algorithm similar to Q-learning
- approximated state value function with multilayer perceptron with one hidden layer
- fared poorly in chess, Go, and checkers

Prior work

neural fitted Q-learning [Rie05]

- minimizes sequence of loss functions (ℓ_2-norm)
- success with purely visual input by first using deep autoencoders to learn reduced task representation
- most similar to current paper

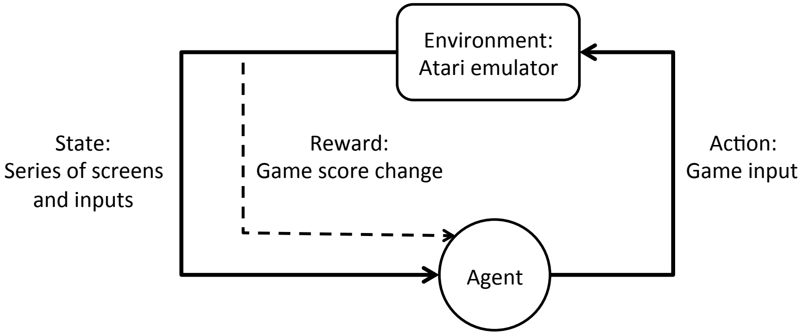
Prior work

other works include

- Q-learning with experience replay, simple neural network, and reduced visual inputs [Lin93]
- restricted Boltzmann machines as value function or policy approximators [SH04, HST12]
- evolutionary architecture used to evolve separate neural networks representing strategies for different games [HMS13]

Outline

Playing Atari



Playing Atari

- **state** sequence of game screens and inputs
  - impossible to capture current situation from only the current screen
  - we refer to sequences and states interchangeably
- **objective** learned policy maximizes discounted future rewards

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

with

  - game termination time step T
  - discount factor γ
  - change in reward at time t' $r_{t'}$

State-action value function

- basic idea behind RL is to estimate

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

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

- optimal value function obeys Bellman equation

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

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

Value approximation

- typically, a linear function approximator is used to estimate $Q^\star$

$$Q\left(s, a; \theta\right) \approx Q^\star\left(s, a\right),$$

  which is parameterized by θ

- we introduce the Q-network
  - nonlinear neural network state-action value function approximator
  - "Q" for Q-learning

Q-network

- trained by minimizing a sequence of loss functions

$$L_i\left(\theta_i\right) = \underset{s,a\sim\rho(\cdot)}{\mathbf{E}}\left[\left(y_i - Q\left(s,a;\theta_i\right)\right)^2\right],$$

  with

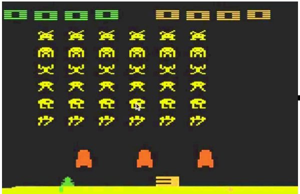
  - iteration number i
  - target $y_i = \mathbf{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q\left(s',a';\theta_{i-1}\right) \mid s,a\right]$
  - "behavior distribution" (exploration policy) $\rho\left(s,a\right)$

- architecture varies according to application

Outline

Preprocessing

raw screen:
210 x 160 pixels, 128 colors



downsample + grayscale:
110 x 84 pixels, 8 colors

final input:
84 x 84 pixels, 8 colors

Brief review

network architecture utilizes

- rectified nonlinearity
- convolutional neural network

Activation function

- common neural units
  - sigmoid unit

$$\frac{1}{1+\exp(-x)}$$

  - tanh unit

$$\tanh(x)$$

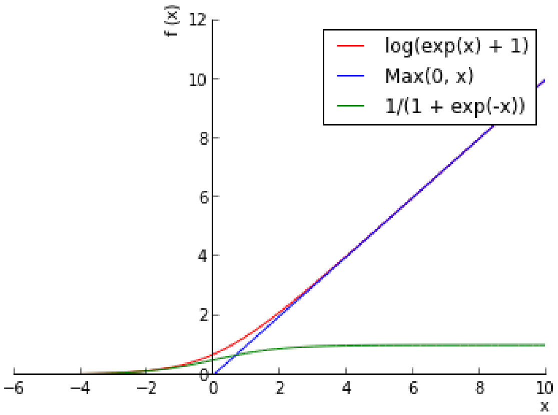
- rectified nonlinearities
  - positive part (used here)

$$\max(0,x) = (x)_+$$

  - softplus

$$\log(1+\exp(x))$$

Activation function

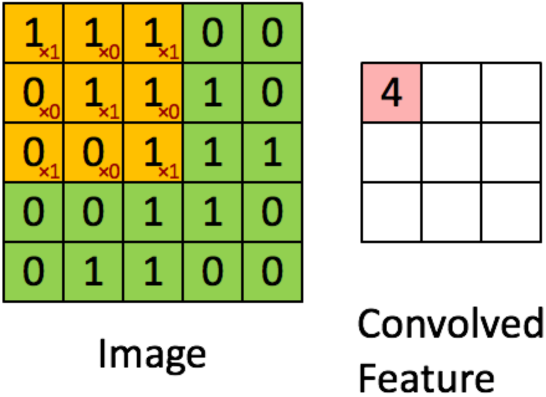


Rectifier nonlinearity

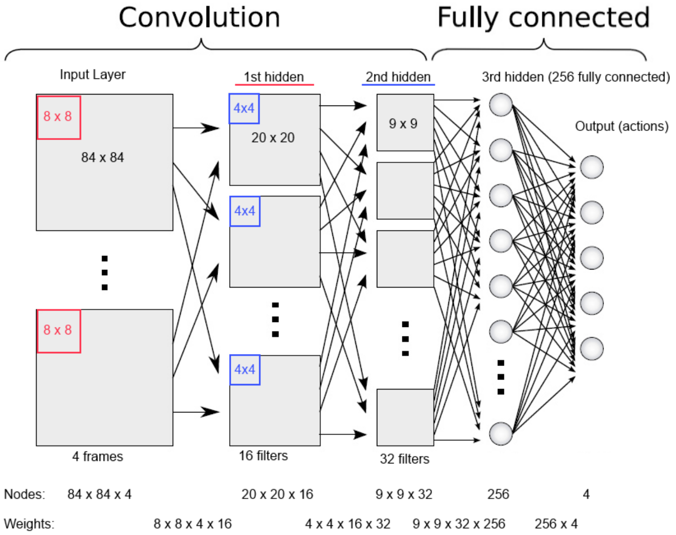
- major differences
  - range of sigmoid/tanh $[0, 1]$ vs. $[0, \infty]$ for rectifier units
  - gradient of sigmoid and tanh vanishes as x increases/decreases
- advantages of rectifier units [J$^+$08, NH10]
  - positive part induces sparsity in hidden units (think ℓ_1-norm)
  - no vanishing gradient problem
  - can model real/integer valued inputs

Convolutional neural network

- biologically-inspired by the visual cortex
- CNN example: single layer, single frame to single filter, stride = 1



Network architecture



Stochastic gradient descent

- optimize Q-network loss function by gradient descent

$$Q\left(s,a;\theta\right) := Q\left(s,a;\theta\right) + \alpha \nabla_{\theta} Q\left(s,a;\theta\right),$$

  with

  - learning rate α

- for computational expedience

  - update weights after every time step
  - avoid computing full expectations
  - replace with single samples from ρ and $\mathcal{E}$

Q-learning

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

- model free RL
  - avoids estimating $\mathcal{E}$
- off-policy
  - learns policy $a = \operatorname{argmax}_a Q(s,a;\theta)$
  - uses behavior distribution selected by an ϵ-greedy strategy

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

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

Deep Q-learning

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

initialize Q-network with random weights θ

repeat until timeout

  initialize frame sequence $s_1 = \{x_1\}$ and preprocessed state $\phi_1 = \phi(s_1)$
  for $t = 1, \ldots, T$

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

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

Outline

Implementation

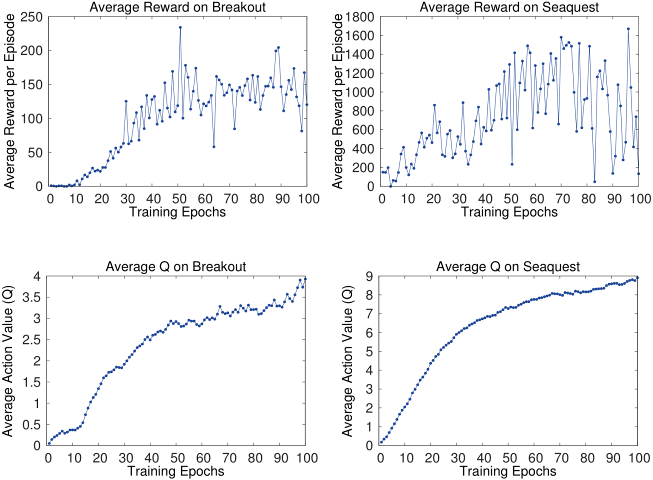
- experiment on seven Atari 2600 games
  - Beam Rider, Breakout, Enduro, Pong, Q*bert, Seaquest, and Space Invaders
- same algorithm
  - network architecture
  - learning algorithm
  - training hyperparameters
- modified reward structure
  - all positive/negative rewards mapped to $+1/-1$

Implementation

- minibatch stochastic gradient descent with RMSProp speedup
  - divide learning rate by a running average of the magnitudes of recent gradients
- ϵ-greedy exploration with simulated annealing
  - linear decrease of ϵ from 1 to 0.1 for first million frames
  - fixed value of 0.1 thereafter
- experience gain with frame-skipping
  - trained for 10 million frames and memory capacity of one million
  - observe every 4th frame (3rd for Space Invaders—because lasers!)

Training and stability

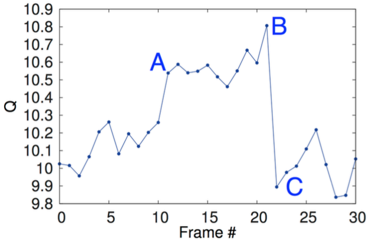
- evolution of average Q suggests convergence



Visualizing the value function

- notice how shooting an enemy increases Q

A B C



Evaluation

- Sarsa [B$^+$13]
  - Sarsa algorithm to learn linear policies
  - uses different handcrafted feature sets for Atari games
- Contingency [BVB12]
  - same basic approach as Sarsa
  - augmented feature sets with learned representation of parts of the screen under agent's control
- both approaches incorporate significant problem knowledge
  - background subtraction
  - treating colors indicating different objects as different inputs

Evaluation

HNeat [S^+13]

- HNeat Best uses a handcrafted object detector
- HNeat Pixel uses an 8-color channel representation of the emulator as an object label map
- relies heavily on finding a deterministic sequence of states that represents a successful game exploit
  - unlikely to generalize to random perturbations in general gameplay
  - compare only against best score

Results

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	-20.4	157	110	179
Sarsa	996	5.2	129	-19	614	665	271
Contingency	1743	6	159	-17	960	723	268
DQN	4092	**168**	**470**	**20**	1952	1705	581
Human	**7456**	31	368	-3	**18900**	**28010**	**3690**
HNeat Best	3616	52	106	19	1800	920	**1720**
HNeat Pixel	1332	4	91	-16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

- beat expert on Breakout, Enduro, and Pong
- relatively close to human performance on Beam Rider
- lost to expert on Q*bert, Seaquest, and Space Invaders
  - probably require strategies with long time scales

Outline

Summary

- introduced novel deep learning model for RL
- demonstrated ability to master difficult game control policies with pixel input
- extended Q-learning with stochastic minibatch updates and experience replay
- acquired by Google for $400 million #win #adtech #googleworlddomination

Extension: prioritized sweeping

- uniform sampling of replay memory does not differentiate important experiences
- always overwrites memory with recent transitions due to finite memory size
- improvement could emphasize experiences from which we can learn the most with prioritized sweeping

Some issues

- lack of theoretical convergence guarantees
- lack of hardware/software implementation details
- lack of computational time estimates (some hints available in recorded presentation)
- switching between British and American English, grammatical and spelling errors, colloquial phrases, and inconsistent figures

Replicating DeepMind

- implementation attempts
  - https://github.com/kristjankorjus/Replicating-DeepMind/
  - https://github.com/spragunr/deep_q_rl/
- reading list
  - http://deeplearning.net/reading-list/

References

- M. Bellemare et al.
  The arcade learning environment: An evaluation platform for general agents.
  Journal of Artificial Intelligence Research, 47:253–279, 2013.
- M. Bellemare, J. Veness, and M. Bowling.
  Investigating contingency awareness using Atari 2600 games.
  AAAI, 2012.
- M. Hausknecht, R. Miikkulainen, and P. Stone.
  A neuro-evolution approach to general Atari game playing.
  IEEE Transactions on Computational Intelligence and AI in Games, 6(4):355–366, 2013.
- N. Heess, D. Silver, and Y. W. Teh.
  Actor-critic reinforcement learning with energy-based policies.
  European Workshop on Reinforcement Learning, page 43, 2012.

References

- K. Jarrett et al.
  What is the best multi-stage architecture for object recognition?
  In *Proc. International Conference on Computer Vision and Pattern Recognition*, Kyoto, Japan, September 2008.
- Long-Ji Lin.
  Reinforcement learning for robots using neural networks.
  PhD thesis, Carnegie Mellon University, 1993.
- V. Nair and G. E. Hinton.
  Rectified linear units improve restricted Boltzmann machines.
  In *Proc. International Conference on Machine Learning*, 2010.
- M. Riedmiller.
  Neural fitted Q-iteration—first experiences with a data efficient neural reinforcement learning method.
  Machine Learning: ECML 2005, pages 317–328, 2005.

References

- G. Stathopoulos et al.
  A hierarchical time-splitting approach for solving finite-time optimal control problems.
  In *European Control Conf.*, 2013.
- Brian Sallans and Geoffrey E. Hinton.
  Reinforcement learning with factored states and actions.
  Journal of Machine Learning Research, 5:1063–1088, 2004.
- G. Tesauro.
  Temporal difference learning and TD-gammon.
  Communications of ACM, 38(3):58–68, 1995.