CS 291A: Deep Learning for NLP

Deep Reinforcement Learning (2)

William Wang
UCSB Computer Science
william@cs.ucsb.edu

Slides adapted from V. Chen and D. Silver.

Final Project Presentation Schedule

- Was out on course website.
  - Three sessions.
  - Starting next Thursday.
  - 13 mins presentation (13 slides max) + 2 mins QA.
  - Final report due: 03/23 23:59PM PT via email (ke00@ucsb.edu)

- Report: You must use the ICML 2018 latex style files for writing the report. The final report must be 3-5 pages long including references. It is encouraged to include the following components in your reports (not necessarily this order): abstract, introduction (motivation, task definition, your novel contributions), related work, your technical approach, such as math formulation of the problem, algorithms, theorems (if any), experiments, discussion, and conclusion.

Review

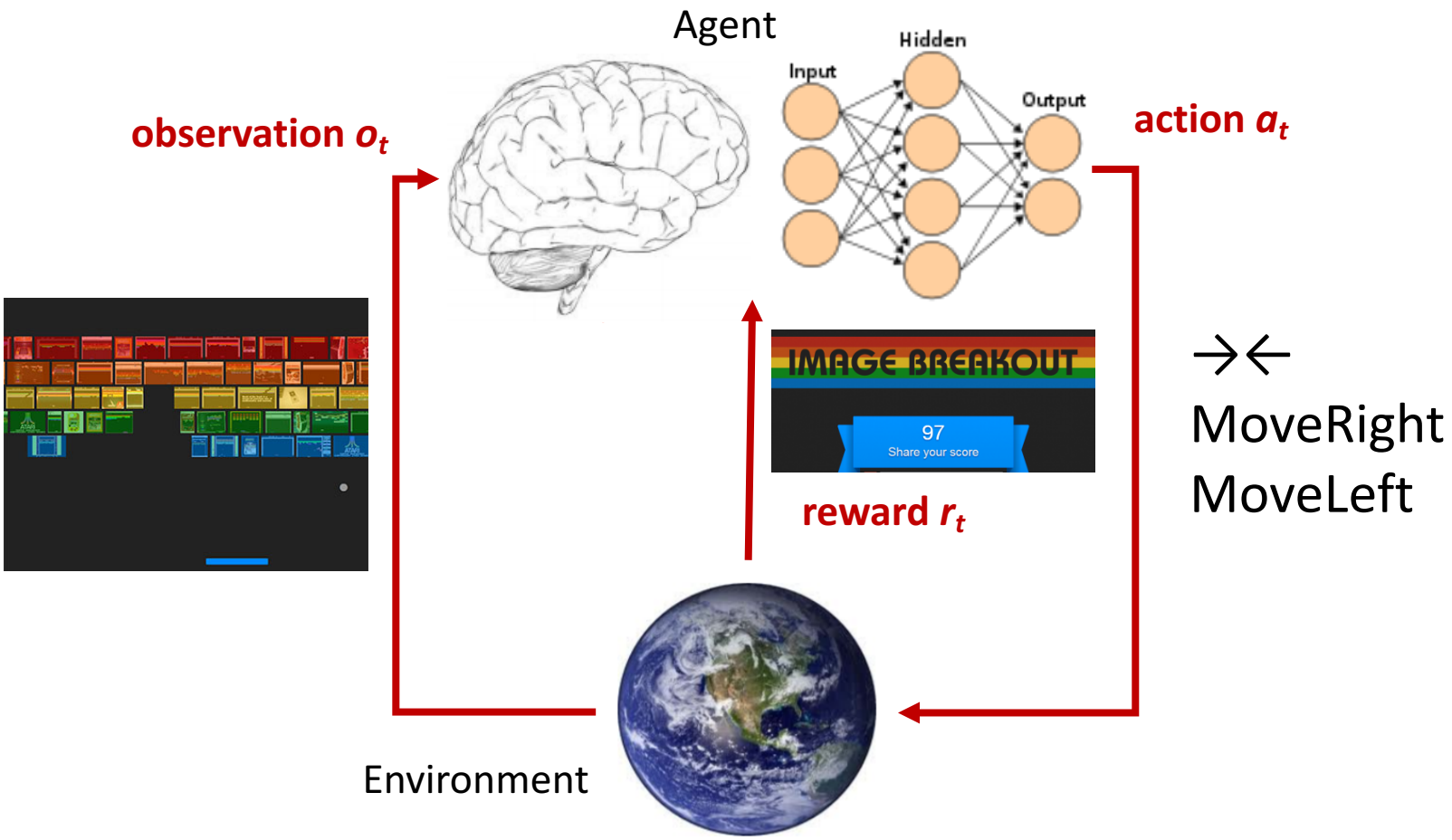
Reinforcement Learning

Reinforcement Learning

- RL is a general purpose framework for **decision making**
  - RL is for an *agent* with the capacity to *act*
  - Each *action* influences the agent's future *state*
  - Success is measured by a scalar *reward* signal

Big three: action, state, reward

Agent and Environment



Major Components in an RL Agent

- An RL agent may include one or more of these components
  - **Policy**: agent's behavior function
  - **Value function**: how good is each state and/or action
  - **Model**: agent's representation of the environment

Reinforcement Learning Approach

- Policy-based RL
  - Search directly for optimal policy π^*

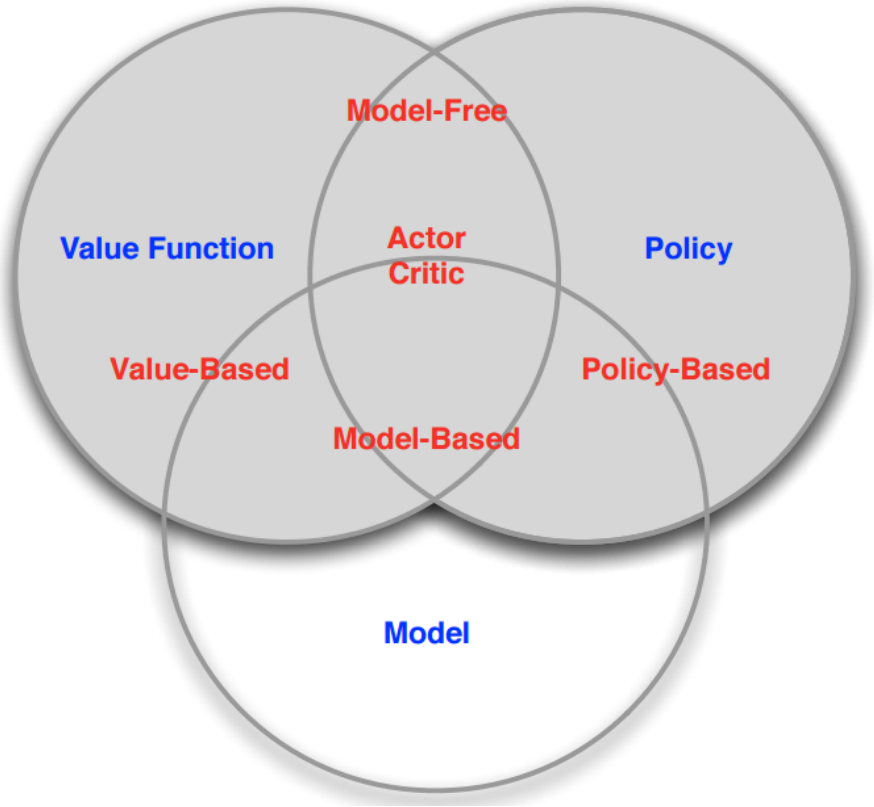
π^* is the policy achieving maximum future reward

- Value-based RL
  - Estimate the optimal value function $Q^*(s, a)$

$Q^*(s, a)$ is maximum value achievable under any policy

- Model-based RL
  - Build a model of the environment
  - Plan (e.g. by lookahead) using model

RL Agent Taxonomy



Deep Reinforcement Learning

- Idea: deep learning for reinforcement learning
  - Use deep neural networks to represent
    - Value function
    - Policy
    - Model
  - Optimize loss function by SGD

Value-Based Deep RL

Estimate How Good Each State and/or Action is

Piazza Poll: Value Function Representation

- Value functions can be represented by a *lookup table*

$$Q(s, a) \quad \forall s, a$$

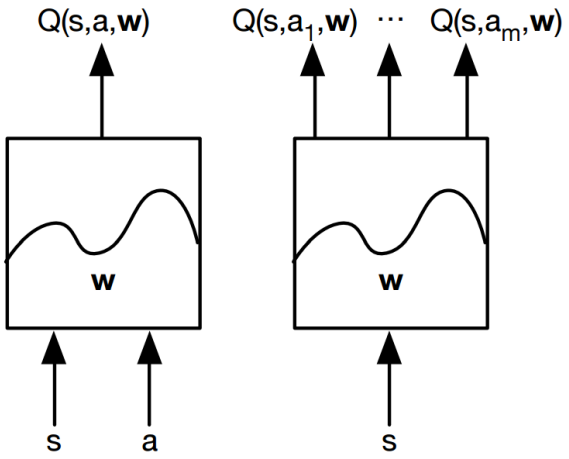
  - What are the main issues with the lookup table approach?
  - (There could be multiple answers)

Value Function Approximation

- Value functions are represented by a *lookup table*

$$Q(s,a) \quad \forall s,a$$

  - too many states and/or actions to store
  - not able to learn the value of each entry individually
- Values can be estimated with *function approximation*

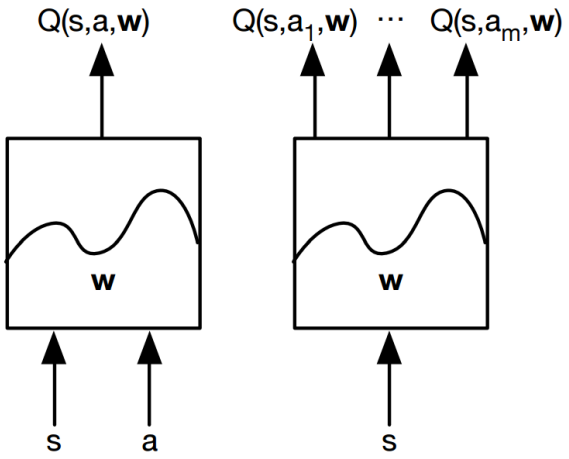


Q-Networks

- Q-networks represent value functions with weights w

$$Q(s, a, w) \approx Q^*(s, a)$$

  - generalize from seen states to unseen states
  - update parameter w for function approximation



Q-Learning

- Goal: estimate optimal Q-values
  - Optimal Q-values obey a Bellman equation

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

learning target

  - *Value iteration* algorithms solve the Bellman equation

$$Q_{i+1}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q_i(s',a') \mid s,a\right]$$

Deep Q-Networks (DQN)

- Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^*(s, a)$$

- Objective is to minimize MSE loss by SGD
  - Starts with initial state s, takes a, gets r, and sees s'
  - but we want w to give us better Q early in the game.

$$L(w) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right)^2\right]$$

- Leading to the following Q-learning gradient

$$\frac{\partial L(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

Piazza Quiz: Issues with training deep Q-Networks (DQN) (multiple choice)?

- Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^*(s, a)$$

- Objective is to minimize MSE loss by SGD
  - Starts with initial state s, takes a, gets r, and sees s'
  - but we want w to give us better Q early in the game.

$$L(w) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right)^2\right]$$

- Leading to the following Q-learning gradient

$$\frac{\partial L(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

Issues with training deep Q-Networks (DQN) (multiple choice)?

- Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^*(s, a)$$

- Objective is to minimize MSE loss by SGD
  - Starts with initial state s, takes a, gets r, and sees s'
  - but we want w to give us better Q early in the game.

$$L(w) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right)^2\right]$$

- Leading to the following Q-learning gradient

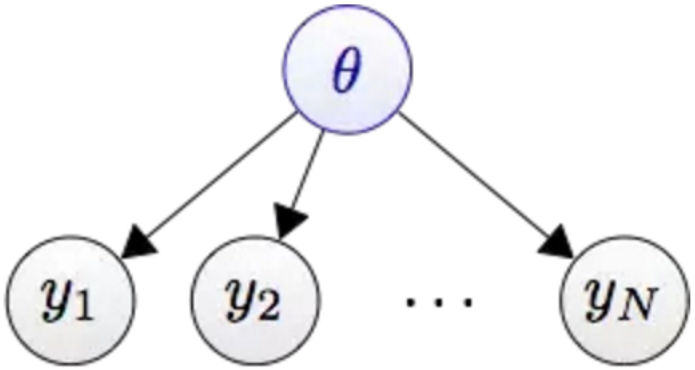
$$\frac{\partial L(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

Issue: naïve Q-learning oscillates or diverges using NN due to:
1) correlations between samples 2) non-stationary targets

Stability Issues with Deep RL

- Naive Q-learning oscillates or diverges with neural nets
  1. Data is sequential
     - Successive samples are correlated, non-iid (independent and identically distributed)
  2. Policy changes rapidly with slight changes to Q-values
     - Policy may oscillate
     - Distribution of data can swing from one extreme to another
  3. Scale of rewards and Q-values is unknown
     - Naive Q-learning gradients can be unstable when backpropagated

Correlations between examples



We are performing deep Q-learning like linear regression,
but the response variable (Q-values) are strongly correlated.

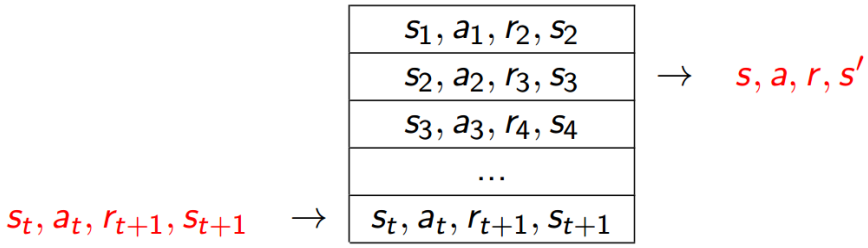
Stable Solutions for DQN

- DQN provides a stable solutions to deep value-based RL
  1. Use experience replay
     - Break correlations in data, bring us back to iid setting
     - Learn from all past policies
  2. Freeze target Q-network
     - Avoid oscillation
     - Break correlations between Q-network and target
  3. Clip rewards or normalize network adaptively to sensible range
     - Robust gradients

Stable Solution 1: Experience Replay

- To remove correlations, build a dataset from agent's experience
  - Take action at according to ϵ-greedy policy
  - Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
  - Sample random mini-batch of transitions (s, a, r, s') from D

small prob for exploration



  - Optimize MSE between Q-network and Q-learning targets

$$L(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right)^2\right]$$

Stable Solution 2: Fixed Target Q-Network

- To avoid oscillations, fix parameters used in Q-learning target
  - Compute Q-learning targets w.r.t. old, fixed parameters w^-

$$r + \gamma \max_{a'} Q(s', a', w^-)$$

  - Optimize MSE between Q-network and Q-learning targets

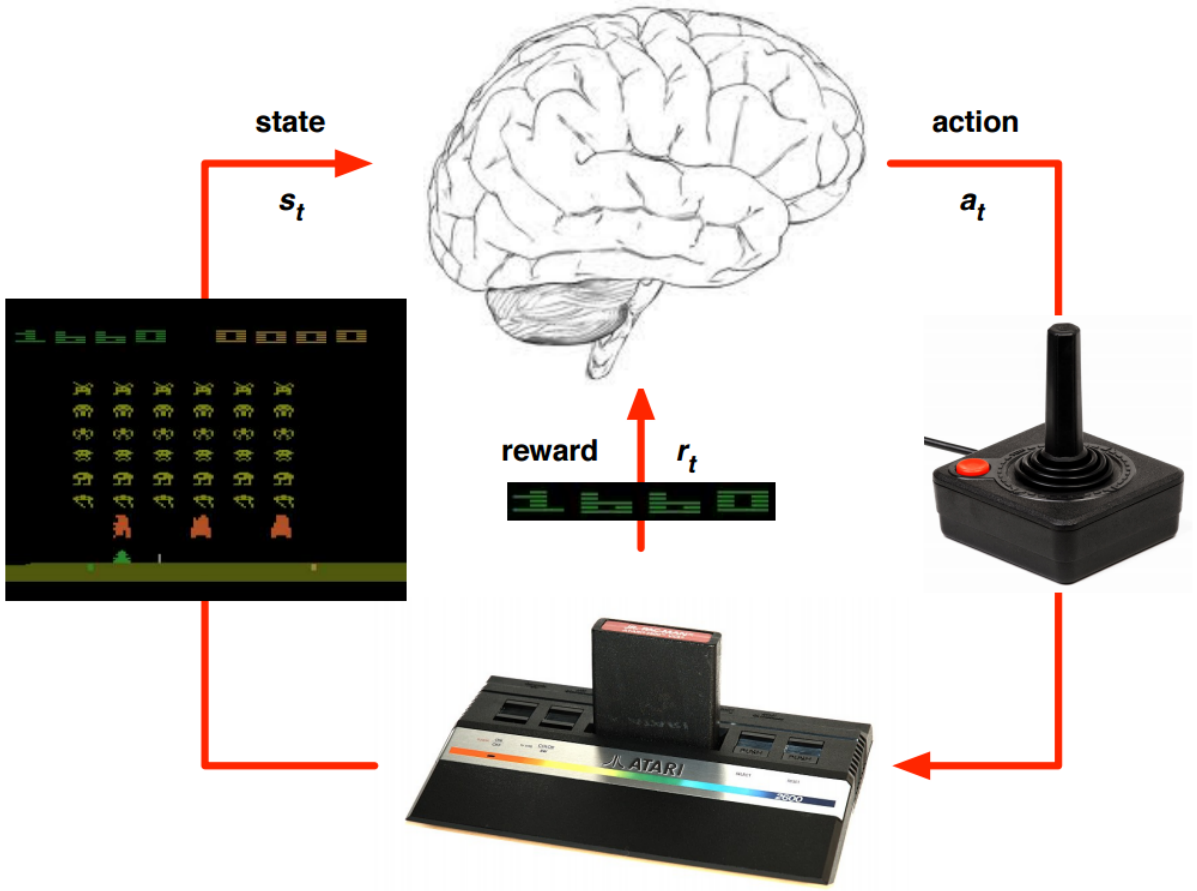
$$L(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w)\right)^2\right]$$

  - Periodically update fixed parameters $w^- \leftarrow w$

Stable Solution 3: Reward / Value Range

- To avoid oscillations, control the reward / value range
  - DQN clips the rewards to [−1, +1]
    - Prevents too large Q-values
    - Ensures gradients are well-conditioned

Deep RL in Atari Games

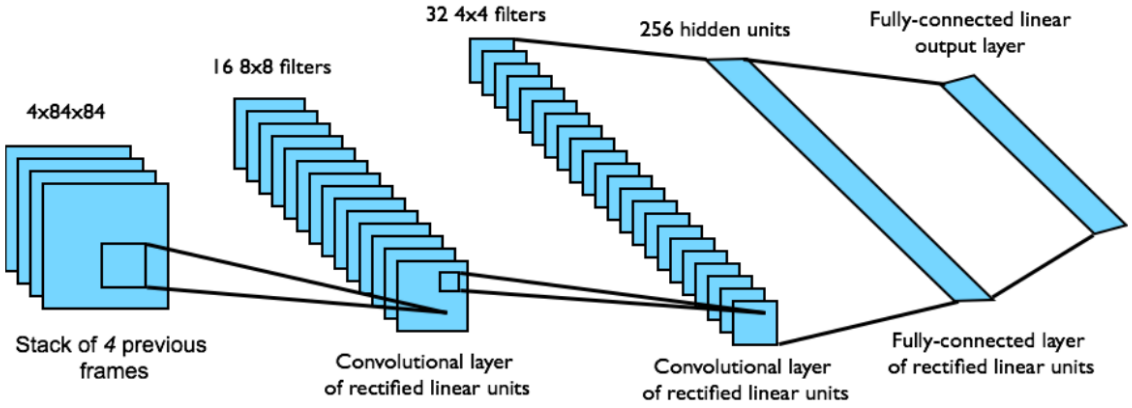


DQN in Atari

- Goal: end-to-end learning of values $Q(s, a)$ from pixels

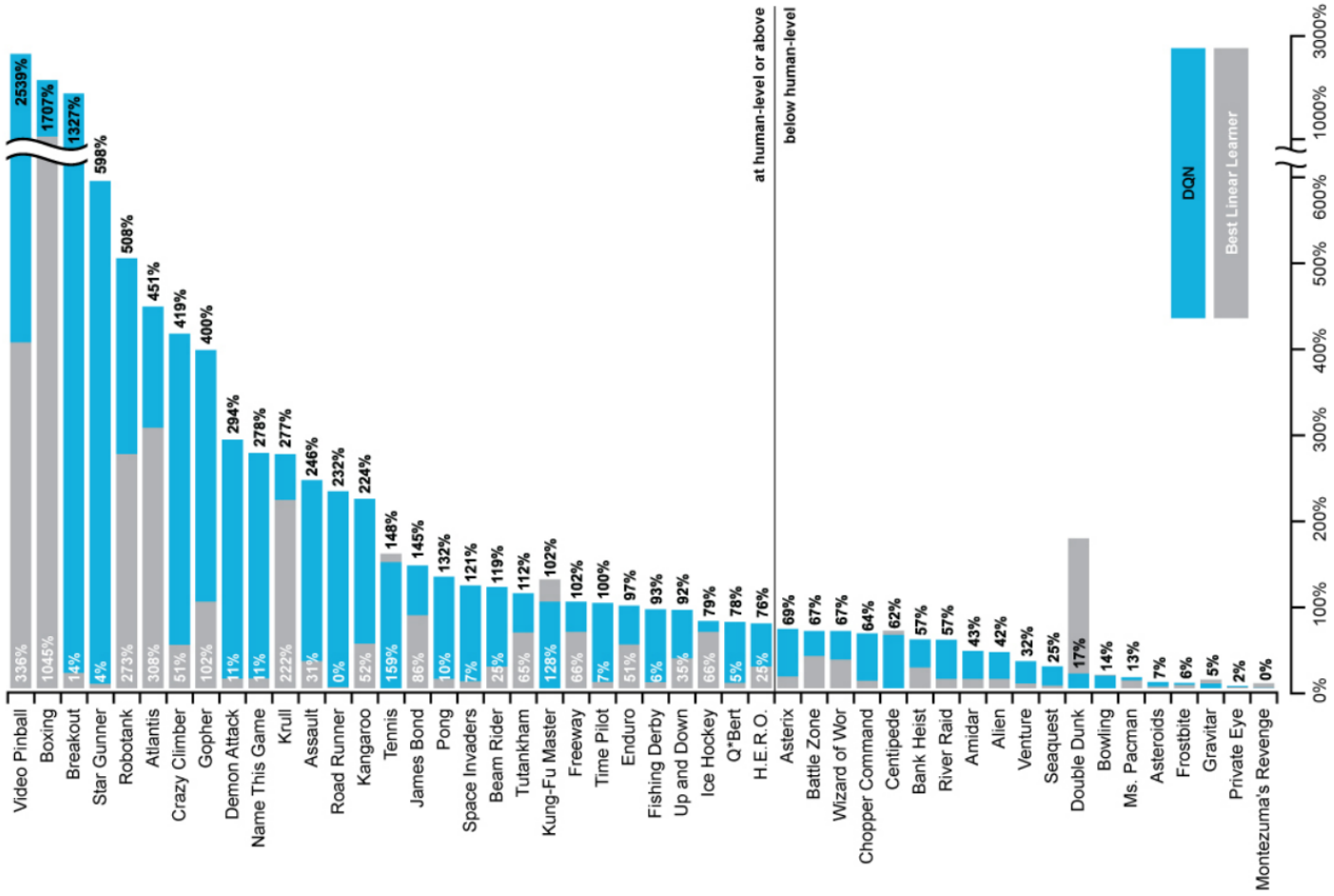
$$L(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s', a', w^{-}) - Q(s, a, w)\right)^2\right]$$

  - Input: state is stack of raw pixels from last 4 frames
  - Output: $Q(s, a)$ for all joystick/button positions a
  - Reward is the score change for that step



link code

DQN in Atari



link code

Other Improvements: Double DQN

- Nature DQN

$$L(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w)\right)^2\right]$$

- Double DQN: remove upward bias caused by $\max_a Q(s, a, w)$
  - Current Q-network w is used to select actions
  - Older Q-network w^- is used to evaluate actions

$$L(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma Q(s', \arg\max_{a'} Q(s', a', w), w^-) - Q(s, a, w)\right)^2\right]$$

Other Improvements: Prioritized Replay

- Prioritized Replay: weight experience based on surprise
  - Store experience in priority queue according to DQN error

$$\left| r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w) \right|$$

Other Improvements: Dueling Network

- Dueling Network: split Q-network into two channels

$$Q(s,a) = V(s,v) + A(s,a,w)$$

  - Action-independent value function $V(s,v)$
    - Value function estimates how good the state is
  - Action-dependent advantage function $A(s,a,w)$
    - Advantage function estimates the additional benefit

Policy-Based Deep RL

Estimate How Good An Agent's Behavior is

Deep Policy Networks

- Represent policy by deep network with weights u

$$a = \pi(a \mid s, u) \qquad a = \pi(s, u)$$

stochastic policy — deterministic policy

- Objective is to maximize total discounted reward by SGD

$$L(u) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots \mid \pi(\cdot, u)\right]$$

Policy Gradient

- The gradient of a stochastic policy $\pi(a \mid s, u)$ is given by

$$\frac{\partial L(u)}{\partial u} = \mathbb{E}_s \left[\frac{\partial \log \pi(a \mid s, u)}{\partial u} Q^{\pi}(s, a) \right]$$

- The gradient of a deterministic policy $\pi(s, u)$ is given by

$$\frac{\partial L(u)}{\partial u} = \mathbb{E}_s \left[\frac{\partial Q^{\pi}(s, a)}{\partial a} \frac{\partial a}{\partial u} \right] \qquad a = \pi(s, u)$$

Actor-Critic (Value-Based + Policy-Based)

- Estimate value function $Q(s, a, w) \approx Q^{\pi}(s, a)$
- Update policy parameters u by SGD
  - Stochastic policy

$$\frac{\partial L(u)}{\partial u} = \mathbb{E}_s \left[\frac{\partial \log \pi(a \mid s, u)}{\partial u} Q(s, a, w) \right]$$

  - Deterministic policy

$$\frac{\partial L(u)}{\partial u} = \mathbb{E}_s \left[\frac{\partial Q(s, a, w)}{\partial a} \frac{\partial a}{\partial u} \right]$$

Deterministic Deep Actor-Critic

- Deep deterministic policy gradient (DDPG) is the continuous analogue of DQN
  - Experience replay: build dataset from agent's experience
  - Critic estimates value of current policy by DQN

$$L(w) = \mathbb{E}\left[\left(r + \gamma Q(s', \pi(s', u^-), w^-) - Q(s, a, w)\right)^2\right]$$

$$\frac{\partial L(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma Q(s', \pi(s', u^-), w^-) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

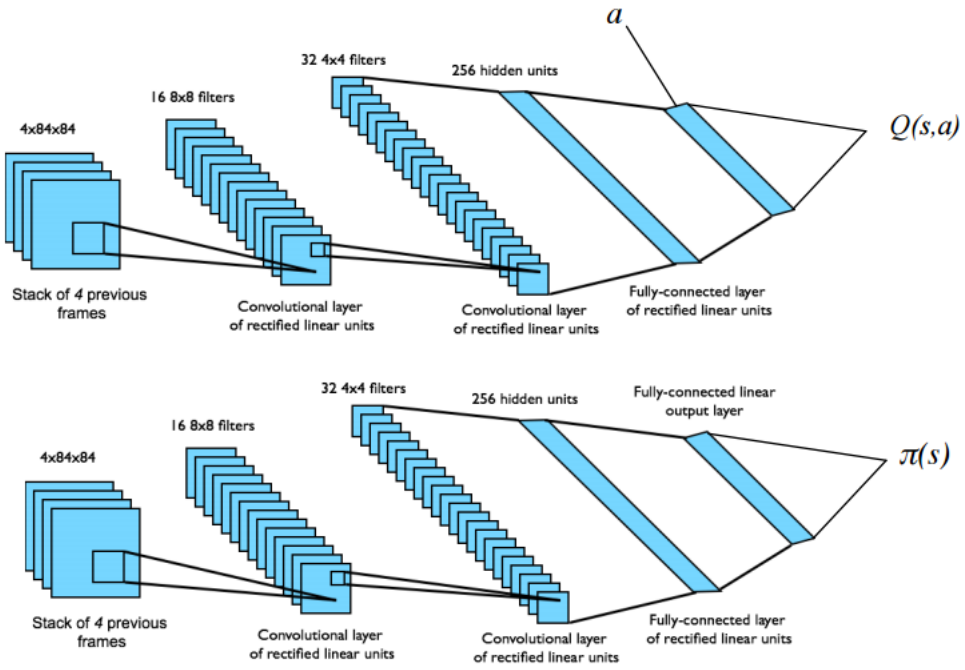
  - Actor updates policy in direction that improves Q

$$\frac{\partial L(u)}{\partial u} = \mathbb{E}\left[\frac{\partial Q(s, a, w)}{\partial a} \frac{\partial a}{\partial u}\right]$$

Critic provides loss function for actor

DDPG in Simulated Physics

- Goal: end-to-end learning of control policy from pixels
  - Input: state is stack of raw pixels from last 4 frames
  - Output: two separate CNNs for Q and π



Model-Based Deep RL

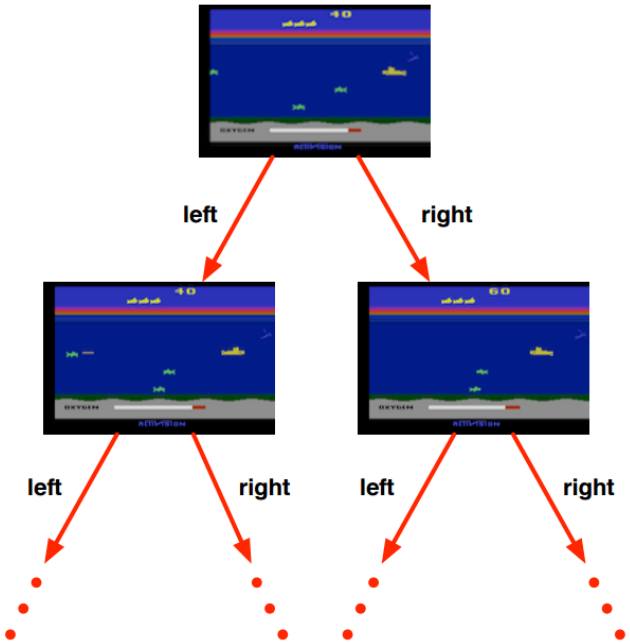
Agent's Representation of the Environment

Model-Based Deep RL

- Goal: learn a transition model of the environment and plan based on the transition model

$$p(r, s' \mid s, a)$$

Objective is to maximize the measured goodness of model



Model-based deep RL is challenging, and so far has failed in Atari

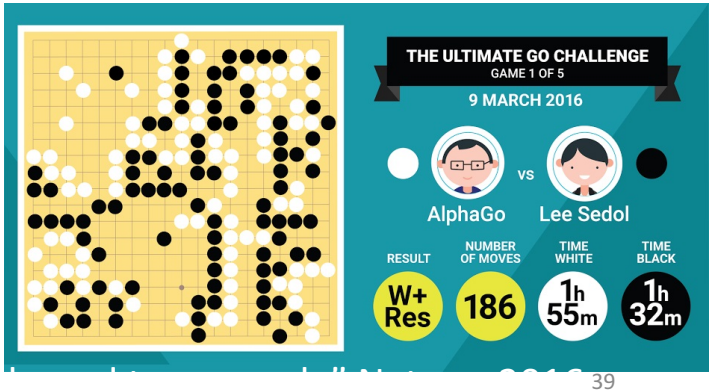
Issues for Model-Based Deep RL

- Compounding errors
  - Errors in the transition model compound over the trajectory
  - A long trajectory may result in totally wrong rewards
- Deep networks of value/policy can “plan” implicitly
  - Each layer of network performs arbitrary computational step
  - n-layer network can “lookahead” n steps

Model-Based Deep RL in Go

- Monte-Carlo tree search (MCTS)
  - MCTS simulates future trajectories
  - Builds large lookahead search tree with millions of positions
  - State-of-the-art Go programs use MCTS
- Convolutional Networks
  - 12-layer CNN trained to predict expert moves
    - Raw CNN (looking at 1 position, no search at all) equals performance of MoGo with 105 position search tree

1st strong Go program



https://deepmind.com/alphago/

OpenAI Universe

- Software platform for measuring and training an AI's general intelligence via the OpenAI gym environment

Concluding Remarks

- RL is a general purpose framework for **decision making** under interactions between agent and environment
- An RL agent may include one or more of these components
  - Policy: agent's behavior function
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment
- RL problems can be solved by end-to-end deep learning
- Reinforcement Learning + Deep Learning = AI

Group Discussion: Challenges for DRL on NLP problems?

- 1) What are some NLP applications that DRL might work?
- 2) DRL was proven to be effective in Atari games and Go: both games have well-defined rules. For NLP problems, we might not have well-defined evaluation, so what can we do?
- 3) To generate words for natural language generation problems, the action space seems large (vocabulary). So how should we design the action space? What could be actions in DRL4NLP apps?

References

- Course materials by David Silver: http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html
- ICLR 2015 Tutorial: http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf
- ICML 2016 Tutorial: http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf