Reinforcement Learning

Timothy Chou Charlie Tong Vincent Zhuang

April 19, 2016

Table of Contents

1. Reinforcement Learning
   - Introduction to RL.
   - Markov Decision Processes.
   - RL Objective and Methods.
2. Q-Learning
   - Algorithm
   - Example
   - Guarantees
3. Deep Q-Learning on Atari
   - Atari Learning Environment
   - Deep Learning
   - Tricks

What is Reinforcement Learning?

RL: general framework for online decision making given partial and delayed rewards

- learner is an agent that performs actions
- actions influence the state of the environment
- environment returns reward as feedback

Generalization of the Multi-Armed Bandit problem

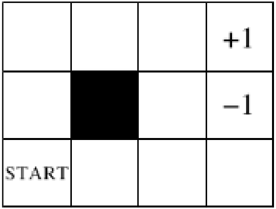
Markov Decision Processes (MDP)

Models the environment that we are trying to learn

Tuple (S, A, P_a, R, γ)

- S the set of states (not necessarily finite)
- A the set of actions (not necessarily finite)
- $P_a(s, s')$ the transition probability kernel
- $R : S \times A \to \mathbb{R}$ the reward function
- $\gamma \in (0, 1)$ the discount factor

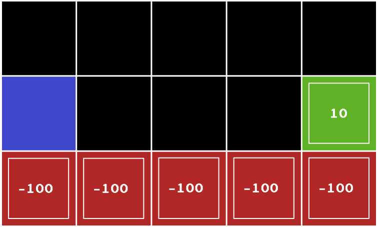
GridWorld MDP Example



- States: each cell of the grid is a state
- Actions: move N, S, E, W, or stationary (can't move off grid or into wall)
- Transitions: Deterministic, move into cell in action direction
- Rewards: 1 or -1 in special spots, 0 otherwise

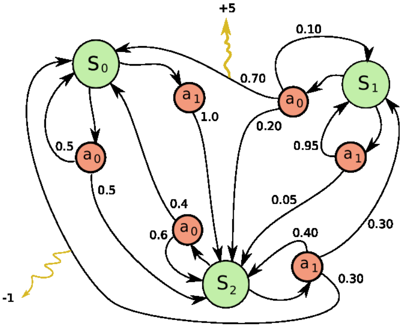
Simulation . . .

Another GridWorld Example



- States: each cell of the grid is a state
- Actions: move N, S, E, W (can't move off grid or into wall)
- Transitions: Deterministic, move into cell in action direction. Any move from 10 or -100 transitions to Start.
- Rewards: 10 or -100 moving out of special spots, 0 otherwise

MDP Overview Example



- Three states $S = \{S_0, S_1, S_2\}$.
- Two actions for each states $A = \{a_0, a_1\}$.
- Probabilistic transitions P_a.
- Rewards defined by $R : S \times A \to \mathbb{R}$.

Markov Property

- Markov Decision Processes (MDP) are very similar to Markov chains. An important property is the **Markov Property**.
- **Markov Property**: Set of possible actions and probability of transitions does not depend on the sequence of events that preceded it. In other words, the system is *memoryless*.
- Sometimes not completely satisfied, but approximation is good enough.

Episodic vs Continuing RL

- Two classes of RL problems:
- Episodic problems are separated by terminations and restarting, such as losing in a game and having to start over.
- Continuing problems are single-episode and continue forever, such as a personalized home assistance robot.

Objective

- Pick the actions that lead to the best future reward
- ”best” $\longleftrightarrow$ maximize expected future discounted return:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{t' \geq t} \gamma^{t'-t} r_{t'}$$

- Discount factor $\gamma \in (0, 1)$
  - avoids infinite return
  - encodes uncertainty about future rewards
  - encodes bias towards immediate rewards

Using a discount factor γ is only one way of capturing this.

Policy and Value

- Policy: $\pi : S \to \mathcal{P}(A)$ - given a state, the probability distribution of the action the agent will choose
- Value: $Q^\pi(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$ - given some policy π, the expected future reward under some state and action

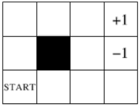
---

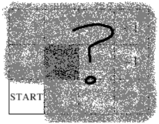
- Compare to the MAB definitions:
  - Policy: Pick an action a_i. For example, UCB1 can be used to determine what action to pick.
  - Value: The expected reward μ_i associated with each action.

RL vs. Bandits

- Reinforcement learning is an extension of bandit problems.
- Standard stochastic MAB problem $\longleftrightarrow$ single-state MDP.
- Contextual bandits can model state, but not transitions
- **Key point:** RL utilizes the entire MDP (S, A, P_a, R, γ). RL can account for delayed rewards and can learn to "traverse" the MDP states.
- No regret analysis for RL (too difficult, hard to generalize). MAB is more constrained, so it is easier to analyze and bound.

Model-based vs. Model-free RL





Model-based approaches assume information about the environment

Do we know the MDP (in particular its transition probabilities)?

- Yes: can solve MDP exactly using dynamic programming/value iteration
- No: try to learn the MDP (e.g. E^3 algorithm[1])

Model-free: learn a policy in absence of a model

- We will focus on model-free approaches

[1] Kearns and Singh (1998)

Model-free approaches

Optimize either value or policy directly - or both!

- Value-based:
  - Optimize value function
  - Policy is implicit
- Policy-based:
  - Optimize policy directly
- Value and policy based:
  - Actor-critic[2]

We will mostly consider value-based approaches.

[2] Konda and Tsitsiklis 2003

Value-based RL

- Define optimal value function to be the best payoff among all possible policies:

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

  Recall π are the policies and Q^{π} are the value functions.
- Value-based approaches: learn optimal value function
- Simple to derive a target policy from optimal value function

Exploration vs. Exploitation in RL

- Important concept for both RL and MAB
- Relevant in learning stage
- Fundamental tradeoff: agent should explore enough to discover a good policy, but should not sacrifice too much reward in the process
- ϵ-greedy strategy:
  Pick the ‘optimal’ strategy with probability $1 - \epsilon$, and select a random action with probability ϵ.

- Recall that the value function is defined as

$$Q^{\pi}(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

- Recall that we can solve the RL problem by learning the optimal value function

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

Bellman equation

- Suppose action a leads to state s'. We can expand the value function recursively:

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

- Solve using value iteration:

$$Q^{\pi}_{i+1}(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^{\pi}_{i}(s', a')|s, a]$$

Approximating the expectation

- If we know the MDP's transition probabilities, we can just write out the expectation:

$$Q(s,a) = \sum_{s'} p_{ss'}(r + \gamma \max_{a'} Q(s',a'))$$

- Q-learning approximates this expectation with a single-sample iterative update (like in SGD)

- Iteratively solve for optimal action-value function Q^* using Bellman equation updates

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t[r_t + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)]$$

  for learning rate α_t

- Intuition for value iteration algorithms: a la gradient descent, iterative updates (hopefully) lead to desired convergence

Target vs. training policy

We distinguish between action selection policies during training and test time.

- Training policy: balance exploration and exploitation such as
  - ϵ-greedy (most commonly used)
  - Softmax

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

- Target policy: pick the best possible action (highest Q-value) every time

Q-learning algorithm

```
Init Q(s, a) = 0 ∀(s, a) in S × A
while not converged do
    t += 1
    pick and do action a_t according to current policy (e.g. ε-greedy)
    receive reward r_t
    observe new state s'
    update Q(s_t, a_t) = Q(s_t, a_t) + α_t [r_t + γ max_{a'} Q(s', a') − Q(s_t, a_t)]
end while
```

Init $Q(s,a) = 0 \forall (s,a) in S \times A$
while not converged do
  $t += 1$
  pick and do action a_t according to current policy (e.g. ϵ-greedy)
  receive reward r_t
  observe new state s'
  update $Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t [r_t + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)]$
end while

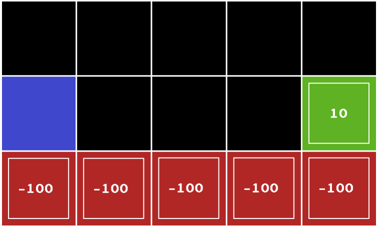
On-policy vs. off-policy algorithm

- Q-learning is an **off-policy** algorithm
    - learned Q function approximates Q^* independent of policy being used
- **On-policy** algorithms perform updates that depend on the policy, such as SARSA:

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha_t[r_t + \gamma Q(s_{t+1}, a_{t+1})]$$

- Convergence properties dependent on policy

Q-learning GridWorld Example



- States: each cell of the grid is a state
- Actions: move N, S, E, W (can't move off grid or into wall)
- Transitions: Deterministic, move into cell in action direction. Any move from 10 or -100 transitions to Start.
- Rewards: 10 or -100 moving out of special spots, 0 otherwise

Q-learning GridWorld Details

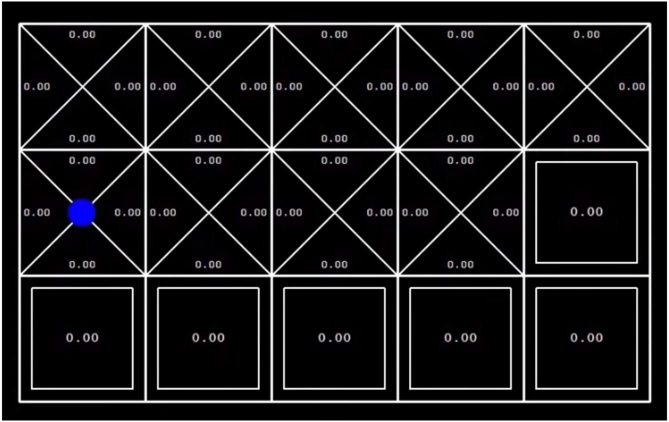
Recall Bellman equation update

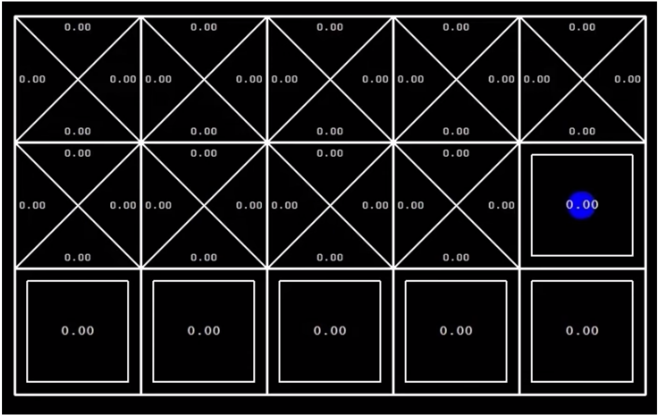
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t [r_t + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)]$$

We have

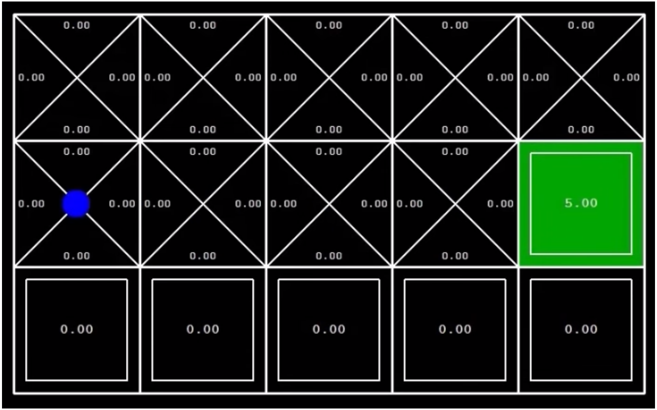
- $\alpha = 0.5$ (for fast updates; usually much smaller)
- $\gamma = 1$

Walkthrough: Initial state



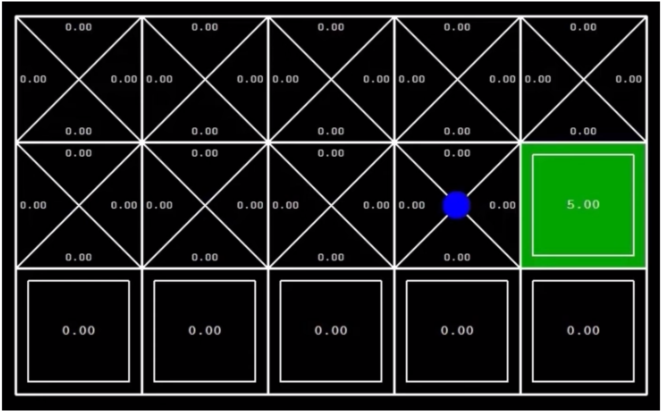


Let’s say the agent keeps on moving right until he reaches the exit

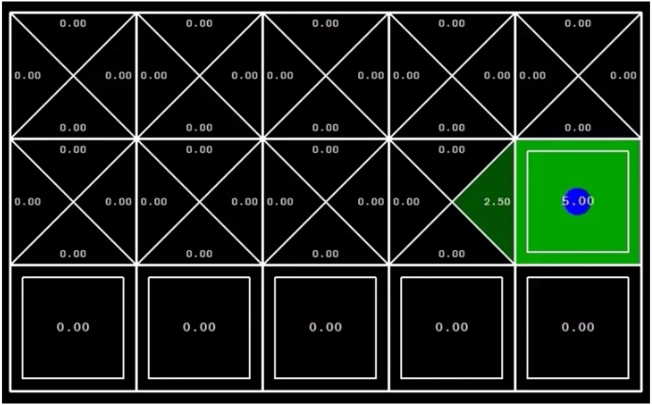


$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t[r_t + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)]$$

$$Q(s^*, a) = 0 + 0.5[10 + 0 - 0] = 5$$

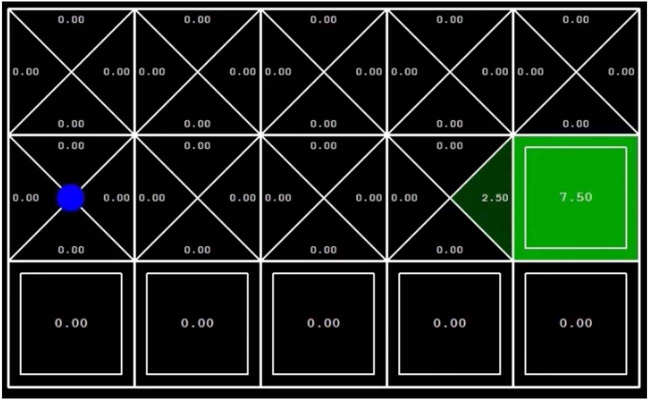


What happens if we reach the exit again?



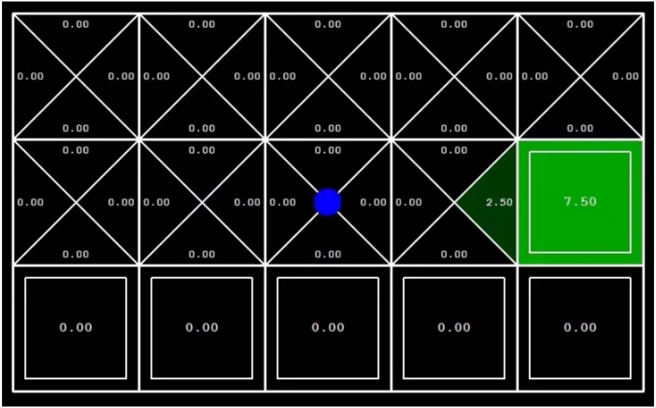
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t[r_t + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)]$$

$$Q(s, a = E) = 0 + 0.5[0 + 5 - 0] = 2.5$$



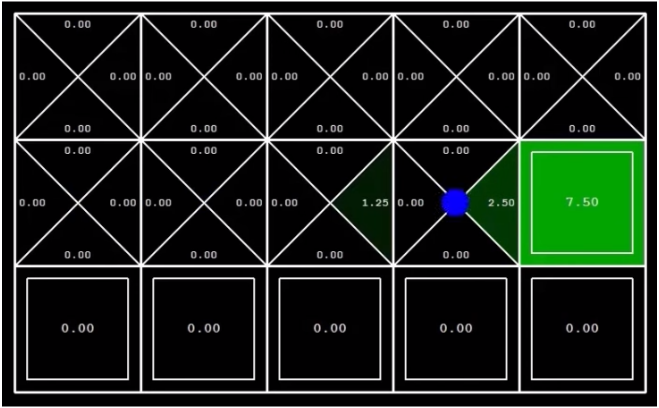
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t [r_t + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)]$$

$$Q(s, a = E) = 5 + 0.5[10 + 0 - 5] = 7.5$$



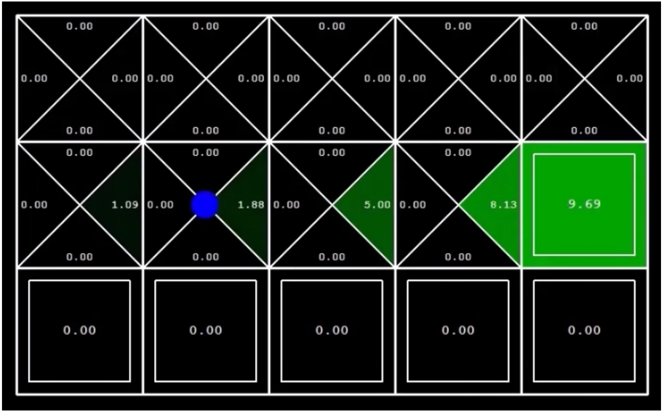
What happens if we keep on going east?

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t[r_t + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)]$$

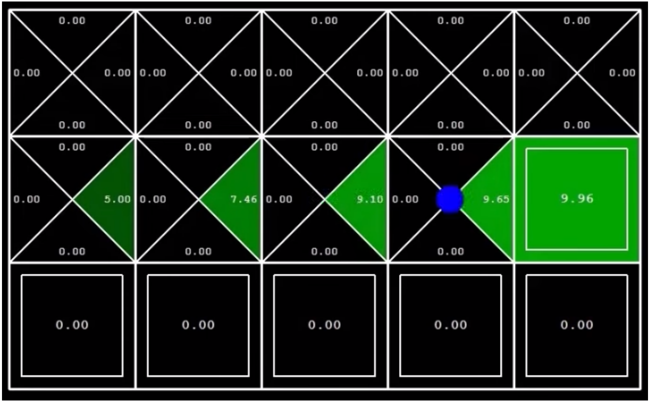


$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t [r_t + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)]$$

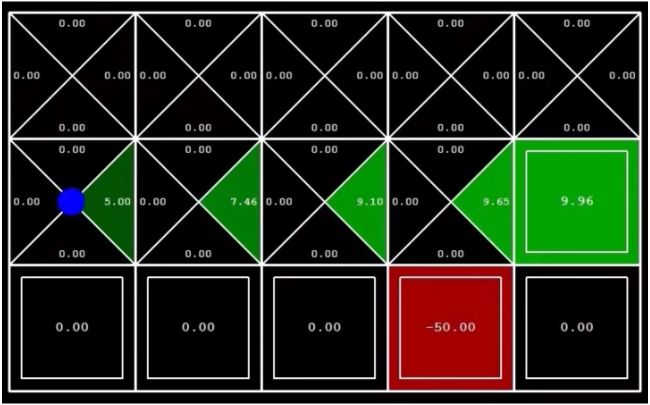
$$Q(s, a = E) = 0 + 0.5[0 + 2.5 - 0] = 1.25$$



After going only east for several episodes

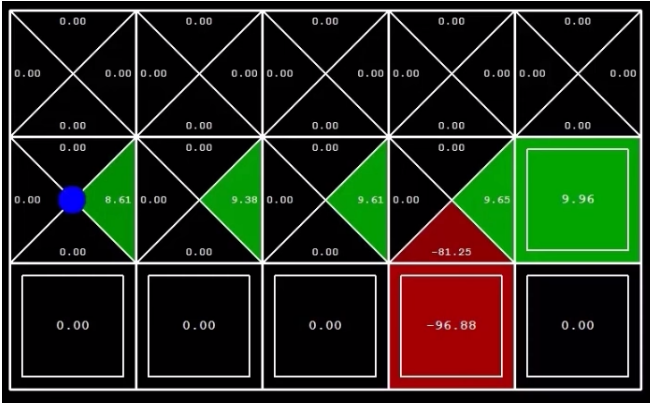


What if we go south?



$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t [r_t + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)]$$

$$Q(s, a) = 0 + 0.5[-100 + 0 - 0] = -50$$


0.00
0.00
0.00
0.00
8.61
9.38
9.61
9.65
9.96
-81.25
-96.88

Recall that update is greedily optimistic:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t[r_t + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)]$$

Q-learning Convergence

Two major assumptions:

i. Every state is visited infinitely often

ii. Learning rate α_t satisfies

$$\sum_{t=1}^{\infty} \alpha_t = \infty \qquad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Theorem

Q-learning converges to the optimal action-value function $Q^*(s,a)$ with probability 1 given i. and ii.

Proof: use stochastic approximation ideas.

Proof Sketch

Lemma

A random iterative process $\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x)F_t(x)$ *convergences to zero w.p.1 under the following assumptions:*

i. $\sum_{t=1}^{\infty} \alpha_t = \infty \qquad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$

ii. $||\mathbb{E}[F_t(x)|\mathcal{F}_t]||_W \leq \gamma ||\Delta_t||_W$ *for* $\gamma \in (0, 1)$

iii. $Var[F_t(x)|\mathcal{F}_t] \leq C(1 + ||\Delta_t||_W^2)$ *for some constant* C

- x denotes state.
- drop dependence on state for clarity
- $\|\cdot\|_W$ denotes some weighted max norm - can just analyze for sup norm

Applying the lemma

Rewrite Bellman equation update:

$$Q_{t+1}(s_t, a_t) = (1 - \alpha_t)Q_t(s_t, a_t) + \alpha_t(r_t + \gamma \max_{a'} Q_t(s_{t+1}, a'))$$

Subtract $Q^*(s_t, a_t)$ from both sides:

$$\begin{aligned} Q_{t+1}(s_t, a_t) - Q^*(s_t, a_t) &= (1 - \alpha_t)(Q_t(s_t, a_t) - Q^*(s_t, a_t)) \\ &\quad + \alpha_t(r_t + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q^*(s_t, a_t)) \\ \Delta_{t+1} &= (1 - \alpha_t)\Delta_t + \alpha_t F_t \end{aligned}$$

Proof boils doing to showing that requirements 2 and 3 of the lemma are satisfied

- First follows from fact that value iteration update F_t is a contraction mapping.
- Second follows by expanding and noting that rewards are bounded.
- See [2] for details.

Function Approximation

- Vanilla Q-learning for finite MDPs stores values in a lookup table
- Obviously intractable for large or continuous MDPs
- However, we can replace this with a function approximator
- Find some model Q with parameters θ s.t.

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

  - Linear models
  - Gaussian processes
  - Neural networks

1. Reinforcement Learning
   - Introduction to RL.
   - Markov Decision Processes.
   - RL Objective and Methods.

2. Q-Learning
   - Algorithm
   - Example
   - Guarantees

3. Deep Q-Learning on Atari
   - Atari Learning Environment
   - Deep Learning
   - Tricks

Deep Q-Learning

- Approximates the value function using a deep network.
  - Non-linear function approximator
- Approximate the value function $Q(s, a, w) \approx Q^{\pi}(s, a)$
- Objective function is mean-squared error of Q-values

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

- Train using gradient descent

$$\nabla \mathcal{L} = \mathbb{E}\left[\left(r + \gamma_{a'} Q(s', a', w) - Q(s, a, w)\right) \nabla Q(s, a, w)\right]$$

Atari

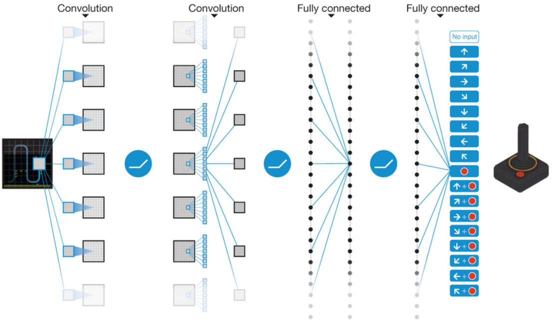
Arcade Learning Environment (ALE): pixel-level games

- Receive as input a 210x160 image with 128 colors and current score
- Action is any of the 18 buttons/joy stick movements
  - Actions unlabeled (ie no specification for up button)
- Still largely unsolved (even after DQN!)

Main challenges:

- Input is very high-dimensional (vision in the form of pixels)
- Long-term planning is difficult (delay between action and reward)

Convolutional Neural Networks

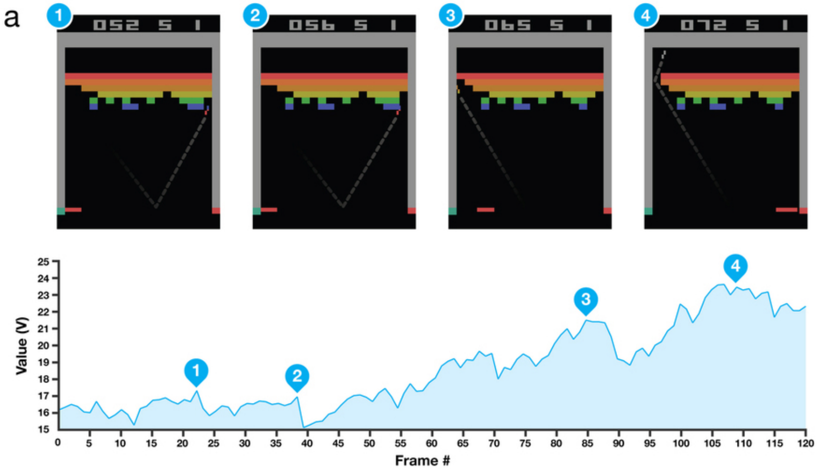


- Convolutional filters mirror the way we see
  - Same filter applies through sliding window across image
  - substantially decreases number of weights needed
- Subsampling of results
  - Take average or max of sliding window
  - translational invariance
- End with fully connected layers

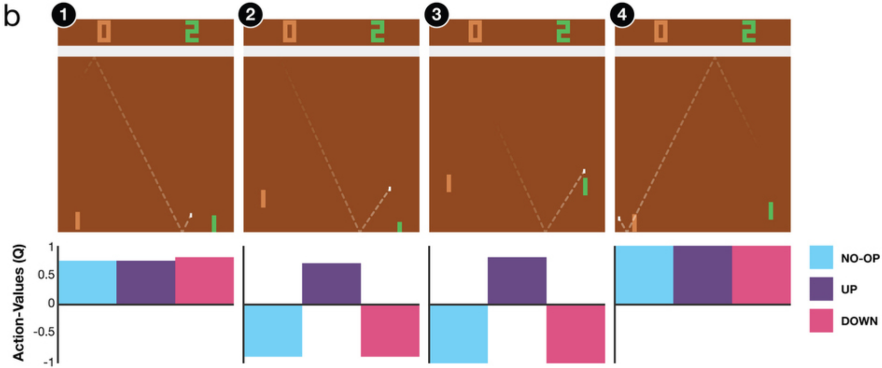
Preprocessing

- CNN on raw CMYK data
  - Pre-processed images by downscaling from 210x160 to 110x84 then cropping to 84x84
  - Max of two frames used to account for flickering
  - Extracted solely Y (luminance) channel
  - Final fully-connected layer to separate output units for each action
- Action selected every k frames for faster training

Q-network Example



Q-network Example



Atari-specific problems

Training deep RL networks directly leads to bad performance

- Adjacent training samples are clearly correlated
- Break correlations
  - experience replay
- Unstable gradients from unknown reward scale
  - clip rewards
- Oscillation from policy and Q-network changing
  - Fix Q-network

Experience Replay

- Build dataset from agent's own experience
  - Store last N transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
  - At each iteration, sample random mini-batch $U(\mathcal{D})$ of transitions from $\mathcal{D}$
  - Recall Bellman equation $Q(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s', a') | s, a\right]$
  - Target $y = r + \gamma \max_{a'} Q(s', a', w)$

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

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

Reward clipping

- Clip rewards to $\{-1, 1\}$
  - Keeps Q-values small
  - Can use same gradient descent parameters
  - Can't tell difference between small and large rewards

Q-network Stability

- Fix Q-network every C updates to a target network $\hat{Q}$
  - Denote saved weights $\hat{w}$
- Use $\hat{Q}$ to generate Q-learning targets y
- Less likely to have oscillations between y and Q changes

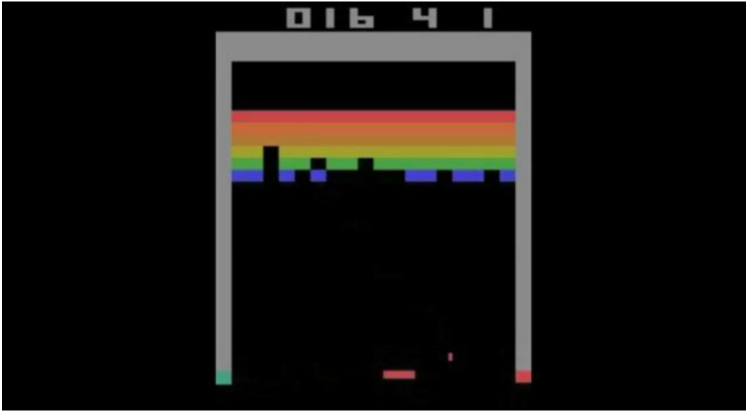
$$\nabla_w = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s', a', \hat{w}] - Q(s, a, w)\right) \nabla_w Q(s, a, w)\right]$$

initialize replay memory $\mathcal{D}$
initialize action-value Q randomly
for episode = $1, M$ **do**
  initialize sequence s_1 and preprocessed sequence ϕ_1
  for $t = 1, T$
  select random action a_t with probability ϵ
  else select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ **do**
    execute action a_t in emulator and observe reward r_t and image x_{t+1}
    store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    set $y_j = r_j$ for terminal ϕ_{j+1} and
    $y_j = r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta)$ for non-terminal ϕ_{j+1}
    perform gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

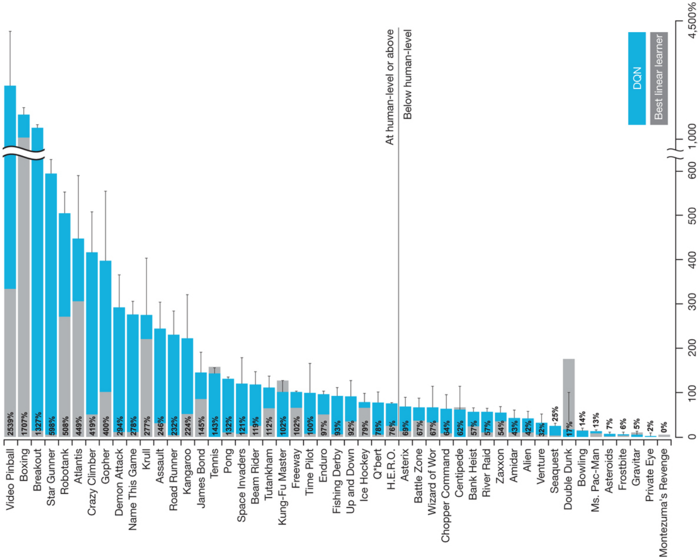
Example

Water World

Example



DQN results


DQN
Best linear learner
At human-level or above
Below human-level
0
100
200
300
400
500
600
1,000
4,500%
Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%

Long-term Planning

- Performs poorly in games requiring long-term planning
- Low probability of finding exact sequence of events with $\epsilon -$ greedy
  - Sequence of *n* exact events is found with probability exponential to *n*
- Q-network has no memory state
- DQRN tries to remedy this with LSTM layer replacing fully connected layer
  - Partially successful on long term games

Breakout trained for 24 hours on Titan X

References

- Hausknecht M., Stone P. (2015). Deep Recurrent Q-Learning for Partially Observable MDPs.arXiv preprint arXiv:1507.06527.
- Jaakkola, T., Jordan, M. I., & Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6), 1185-1201.
- Melo, F. S. (2001). Convergence of Q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep.*
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.
- Mnih V., Kavukcouglu, K., Silver D., Rusu A., Veness J., Bellemare M., Graves A., Riedmiller M., Fidjeland A., Ostrovski G., Petersen S., Beattie C., Sadik A., Antonoglou I., King H., Kumaran D., Wierstra D., Legg S., Hassabis D. Human-level control through deep reinforcement learning. Nature.
- Sutton, R. S., & Barto, A. G. (2011). *Reinforcement learning: An introduction.* The MIT Press (March 1998).