

Reinforcement Learning

Stephen Scott

(Adapted from Eleanor Quint and Tom Mitchell)

sscott2@unl.edu

- Consider learning to choose actions, e.g.,
 - Robot learning to dock on battery charger
 - Learning to choose actions to optimize factory output
 - Learning to play Backgammon, chess, Go, etc.
- Note several problem characteristics:
 - Delayed reward (thus have problem of **temporal credit assignment**)
 - Opportunity for active exploration (versus exploitation of known good actions)
 - ⇒ Learner has some influence over the training data it sees
 - Possibility that state only **partially observable**

- TD-Gammon (Tesauro, 1995): Learn to play Backgammon
 - Immediate Reward:
 - +100 if win
 - -100 if lose
 - 0 for all other states
 - Trained by playing 1.5 million games against itself
 - Approximately equal to best human player at that time
- Deep Q network for Atari 2600 games (Mnih et al., 2015)
- AlphaGo (Silver et al., 2016) and AlphaGo Zero (Silver et al., 2017): Play Go at world-class level
- Training entails **self play** with reward based on winning or losing each game

- Markov decision processes
- The agent's learning task
- Q learning
- Temporal difference learning
- Deep Q learning
- Example: Learning to play Atari
- Example: AlphaGo
- Summary

Reinforcement Learning Problem

Reinforcement Learning

Stephen Scott

Introduction

MDPs

Q Learning

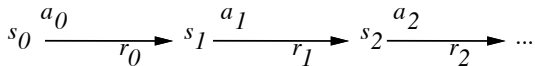
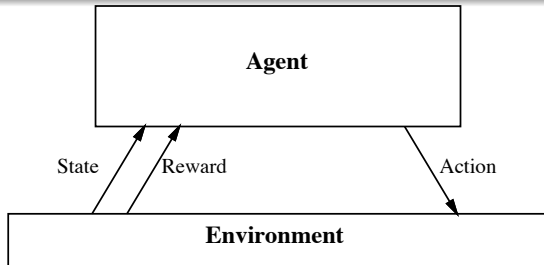
TD Learning

DQN

Atari Example

Go Example

Summary



Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots, \text{ where } 0 \leq \gamma < 1$$

Assume

- Finite set of states S
- Set of actions A
- At each discrete time t agent observes state $s_t \in S$ and chooses action $a_t \in A$
- Then receives immediate reward r_t , and state changes to s_{t+1}
- **Markov assumption:** $s_{t+1} = \delta(s_t, a_t)$ and $r_t = r(s_t, a_t)$
 - I.e., r_t and s_{t+1} depend only on **current** state and action
 - Functions δ and r may be nondeterministic
 - Functions δ and r not necessarily known to agent

Agent's Learning Task

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

Summary

- Execute actions in environment, observe results, and
 - Learn **action policy** $\pi : S \rightarrow A$ that maximizes

$$E [r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots]$$

from any starting state in S

- Here $0 \leq \gamma < 1$ is the **discount factor** for future rewards
- Note something new:
 - Target function is $\pi : S \rightarrow A$
 - But we have no training examples of form $\langle s, a \rangle$
 - Training examples are of form $\langle \langle s, a \rangle, r \rangle$
 - I.e., not told what best action is, instead told reward for executing action a in state s

- First consider deterministic worlds
- For each possible policy π the agent might adopt, we can define **discounted cumulative reward** as

$$V^\pi(s) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i} ,$$

where r_t, r_{t+1}, \dots are generated by following policy π , starting at state s

- Restated, the task is to learn an **optimal policy** π^*

$$\pi^* \equiv \operatorname{argmax}_{\pi} V^\pi(s), \quad (\forall s)$$

Value Function

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

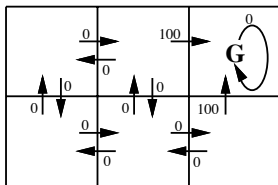
TD Learning

DQN

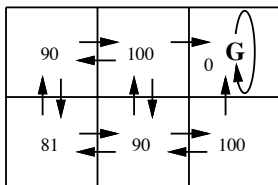
Atari Example

Go Example

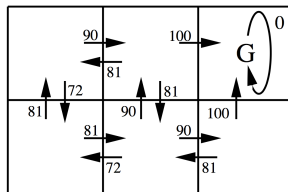
Summary



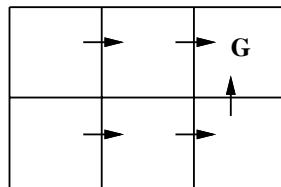
$r(s, a)$ values



$V^*(s)$ values



$Q(s, a)$ values



One optimal policy

What to Learn

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

Summary

- We might try to have agent learn the evaluation function V^{π^*} (which we write as V^*)
- It could then do a lookahead search to choose best action from any state s because

$$\pi^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))] ,$$

i.e., choose action that maximized immediate reward + discounted reward if optimal strategy followed from then on

- E.g., $V^*(bot. \text{ ctr.}) = 0 + \gamma 100 + \gamma^2 0 + \gamma^3 0 + \dots = 90$
- A problem:
 - This works well if agent knows $\delta : S \times A \rightarrow S$, and $r : S \times A \rightarrow \mathbb{R}$
 - But when it doesn't, it can't choose actions this way

- Define new function very similar to V^* :

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

i.e., $Q(s, a)$ = total discounted reward if action a taken in state s and optimal choices made from then on

- If agent learns Q , it can choose optimal action even without knowing δ

$$\begin{aligned} \pi^*(s) &= \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))] \\ &= \operatorname{argmax}_a Q(s, a) \end{aligned}$$

- Q is the evaluation function the agent will learn

Training Rule to Learn Q

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

 Q Learning

TD Learning

DQN

Atari Example

Go Example

Summary

- Note Q and V^* closely related:

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

- Which allows us to write Q recursively as

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

- Let \hat{Q} denote learner's current approximation to Q ;
consider training rule

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a') ,$$

where s' is the state resulting from applying action a in state s

Q Learning for Deterministic Worlds

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

Summary

- For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Observe current state s
- Do forever:
 - Select an action a (greedily or probabilistically) and execute it
 - Receive immediate reward r
 - Observe the new state s'
 - Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$
 - $s \leftarrow s'$
- Note that actions not taken and states not seen don't get explicit updates (might need to generalize)

Updating \hat{Q}

Reinforcement Learning

Stephen Scott

Introduction

MDPs

Q Learning

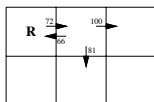
TD Learning

DQN

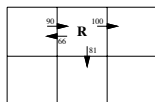
Atari Example

Go Example

Summary



Initial state: s_1



Next state: s_2

$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &= 0 + 0.9 \max\{66, 81, 100\} \\ &= 90\end{aligned}$$

Can show via induction on n that if rewards non-negative and \hat{Q} s initially 0, then

$$(\forall s, a, n) \quad \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)$$

and

$$(\forall s, a, n) \quad 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$$

- \hat{Q} **converges to** Q : Consider case of deterministic world where each $\langle s, a \rangle$ is visited infinitely often
- **Proof:** Define a **full interval** to be an interval during which each $\langle s, a \rangle$ is visited. Will show that during each full interval the largest error in \hat{Q} table is reduced by factor of γ
- Let \hat{Q}_n be table after n updates, and Δ_n be the maximum error in \hat{Q}_n ; i.e.,

$$\Delta_n = \max_{s,a} |\hat{Q}_n(s, a) - Q(s, a)|$$

- Let $s' = \delta(s, a)$

- For any table entry $\hat{Q}_n(s, a)$ updated on iteration $n + 1$, error in the revised estimate $\hat{Q}_{n+1}(s, a)$ is

$$\begin{aligned}
 |\hat{Q}_{n+1}(s, a) - Q(s, a)| &= |(r + \gamma \max_{a'} \hat{Q}_n(s', a')) - (r + \gamma \max_{a'} Q(s', a'))| \\
 &= \gamma |\max_{a'} \hat{Q}_n(s', a') - \max_{a'} Q(s', a')| \\
 (*) &\leq \gamma \max_{a'} |\hat{Q}_n(s', a') - Q(s', a')| \\
 (**) &\leq \gamma \max_{s'', a'} |\hat{Q}_n(s'', a') - Q(s'', a')| \\
 &= \gamma \Delta_n
 \end{aligned}$$

(*) works since $|\max_a f_1(a) - \max_a f_2(a)| \leq \max_a |f_1(a) - f_2(a)|$

(**) works since max will not decrease

- Also, $\hat{Q}_0(s, a)$ and $Q(s, a)$ are both bounded $\forall s, a$
 $\Rightarrow \Delta_0$ bounded
- Thus after k full intervals, error $\leq \gamma^k \Delta_0$
- Finally, each $\langle s, a \rangle$ visited infinitely often \Rightarrow number of intervals infinite, so $\Delta_n \rightarrow 0$ as $n \rightarrow \infty$

Nondeterministic Case

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

Summary

- What if reward and next state are non-deterministic?
- We redefine V, Q by taking expected values:

$$\begin{aligned} V^\pi(s) &\equiv \mathbf{E} [r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots] \\ &= \mathbf{E} \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \right] \end{aligned}$$

$$\begin{aligned} Q(s, a) &\equiv \mathbf{E} [r(s, a) + \gamma V^*(\delta(s, a))] \\ &= \mathbf{E} [r(s, a)] + \gamma \mathbf{E} [V^*(\delta(s, a))] \\ &= \mathbf{E} [r(s, a)] + \gamma \sum_{s'} P(s' | s, a) V^*(s') \\ &= \mathbf{E} [r(s, a)] + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s', a') \end{aligned}$$

- Q learning generalizes to nondeterministic worlds
- Alter training rule to

$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n) \hat{Q}_{n-1}(s, a) + \alpha_n [r + \gamma \max_{a'} \hat{Q}_{n-1}(s', a')]$$

where

$$\alpha_n = \frac{1}{1 + \text{visits}_n(s, a)}$$

- Can still prove convergence of \hat{Q} to Q , with this and other forms of α_n (Watkins and Dayan, 1992)

- Q learning: reduce error between successive Q estimates
- **Q estimate using one-step time difference:**

$$Q^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$$

- **Why not two steps?**

$$Q^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a)$$

- **Or n ?**

$$Q^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \cdots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a)$$

- **Blend all of these** ($0 \leq \lambda \leq 1$):

$$\begin{aligned} Q^\lambda(s_t, a_t) &\equiv (1 - \lambda) \left[Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \dots \right] \\ &= r_t + \gamma \left[(1 - \lambda) \max_a \hat{Q}(s_{t+1}, a) + \lambda Q^\lambda(s_{t+1}, a_{t+1}) \right] \end{aligned}$$

- TD(λ) algorithm uses above training rule
 - Sometimes converges faster than Q learning
 - Converges for learning V^* for any $0 \leq \lambda \leq 1$ (Dayan, 1992)
 - Tesauro's TD-Gammon uses this algorithm

- Convergence proofs assume that $\hat{Q}(s, a)$ represented **exactly**
 - E.g., as an array
- How well does this scale to real problems?
- What can we do about it?

- We already have machinery to approximate functions based on labeled samples
- Search for a **deep Q network** (DQN) to implement function Q_{θ} approximating Q
- Each training instance is $\langle s, a \rangle$ with label $y(s, a) = r + \gamma \max_{a'} Q_{\theta}(s', a')$
 - I.e., take action a in state s , get reward r and observe new state s'
 - Then use Q_{θ} to compute label $y(s, a)$ and update as usual
- Convergence proofs break, but get scalability to large state space

DQN Example: Playing Atari (Mnih et al., 2015)

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

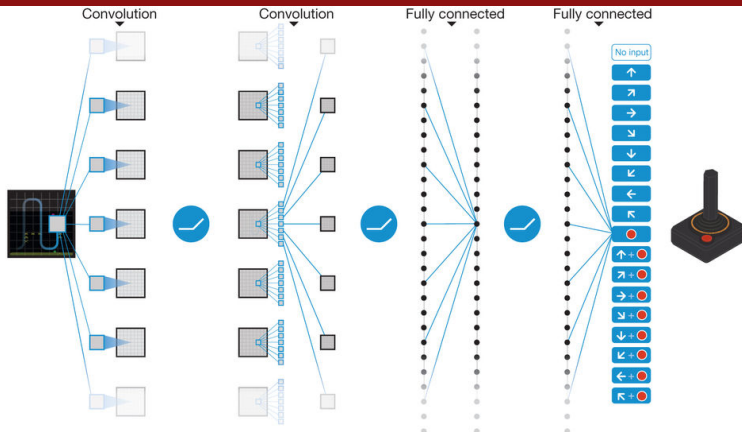
Go Example

Summary

- Applied same architecture and hyperparameters to 49 Atari 2600 games
 - System learned effective policy for each, very different, game
 - No game-specific modifications
- State description consists of raw input from emulator
- Frames rescaled to 84×84 , single channel
- Each state is sequence of four most recent frames
- Rather than take s and a as inputs, network takes s and gives prediction of $Q(s, a)$ for all a as outputs
- Clipped positive rewards to $+1$ and negative to -1
- Evaluated each policy's performance against professional human tester

DQN Example: Playing Atari (Mnih et al., 2015)

Architecture



- **Input:** $84 \times 84 \times 4$, 3 convolutional layers, two dense
- Conv: 32 20×20 , 64 16×16 , 64 12×12
- 512 units in dense layers
- 18 outputs: Output i is estimate of $Q(s, a_i)$

DQN Example: Playing Atari (Mnih et al., 2015)

Training

Reinforcement Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

Summary

- Reward signal at time t : $+1$ if score increased, -1 if decreased, 0 otherwise
- Action in game selected via **ϵ -greedy policy**: With probability ϵ choose action u.a.r., with probability $(1 - \epsilon)$ choose $\arg\max_a Q_{\theta}(s, a)$
- Chosen action a_t run in emulator, which returns reward r_t and next frame for state s_{t+1}
- Update:

$$\theta_{t+1} = \theta_t + \alpha \left[r_t + \gamma \max_{a'} Q_{\theta_t}(s_{t+1}, a') - Q_{\theta_t}(s_t, a_t) \right] \nabla Q_{\theta_t}(s_t, a_t)$$

- Trained with RMSProp, mini-batch size of 32

DQN Example: Playing Atari (Mnih et al., 2015)

Modifications

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

Summary

Deep RL systems can be unstable or divergent, so Mnih:

- 1 Used **experience replay**: Rather than train on consecutive tuples, tuple (s_t, a_t, r_t, s_{t+1}) from game play added to **replay memory**
 - Replay memory sampled u.a.r. for training mini-batches
 - Independent instances in mini-batches reduces correlations in training data
 - Trained **off-policy** (policy trained is not the one choosing actions in game)
- 2 Used separate **target network** $\tilde{\theta}$ to generate labels:

$$\theta_{t+1} = \theta_t + \alpha \left[r_t + \gamma \max_{a'} Q_{\tilde{\theta}_t}(s_{t+1}, a') - Q_{\theta_t}(s_t, a_t) \right] \nabla Q_{\theta_t}(s_t, a_t)$$

Copied θ into $\tilde{\theta}$ every C updates

- 3 **Clipped** error term $[r_t + \dots - Q_{\theta_t}(s_t, a_t)]$ to $[-1, 1]$

DQN Example: Playing Atari (Mnih et al., 2015)

Pseudocode

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ε select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

DQN Example: Playing Atari (Mnih et al., 2015)

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

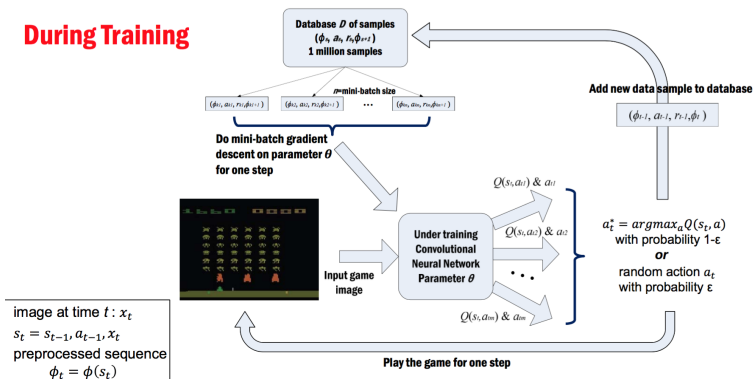
DQN

Atari Example

Go Example

Summary

During Training



DQN Example: Playing Atari (Mnih et al., 2015)

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

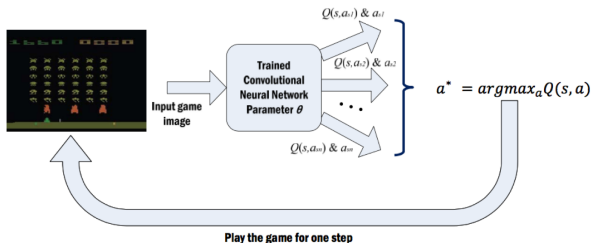
DQN

Atari Example

Go Example

Summary

After Training



DQN Example: Playing Atari (Mnih et al., 2015)

Results

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

Summary

- Trained on each game for 50 million frames, no transfer learning
- Testing: Averaged final score over 30 sessions/game
- Measured performance of DQN RL and linear learner RL (with custom features) vs. human player:

$$100(\text{RL} - \text{random}) / (\text{human} - \text{random})$$
 - I.e., human=100%, random=0%
- DQN outperformed linear learner on all but 6 games, outperformed human on 22, and comparable to human on 7
- Shortcoming: Performance poor (near random) when long-term planning required, e.g., Montezuma's revenge

DQN Example: Playing Atari (Mnih et al., 2015)

Results

Reinforcement Learning

Stephen Scott

Introduction

MDPs

Q Learning

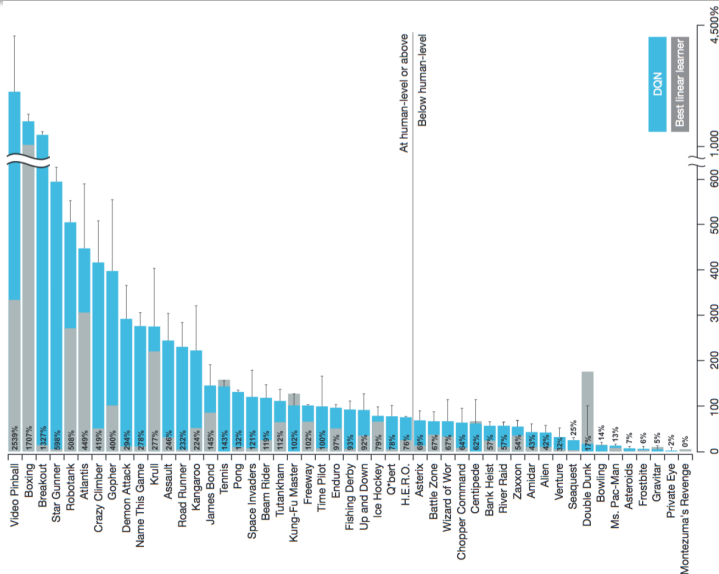
TD Learning

DQN

Atari Example

Go Example

Summary



DQN Example: Playing Atari (Mnih et al., 2015)

Breakout

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

Summary

Breakout example: <https://youtu.be/TmPfTpjtdgg>

- One of the most complex board games humans have
- Checkers has about 10^{18} distinct states, Backgammon: 10^{20} , Chess: 10^{47} , **Go**: 10^{170}
 - Number of atoms in the universe around 10^{81}
 - Another issue: Difficult to quantify goodness of a board configuration
- **AlphaGo**: Used RL and human knowledge to defeat professional player
- **AlphaGo Zero**: Improved on AlphaGo without human knowledge
- **AlphaZero**: Generalized to chess and shogi with general RL

AlphaGo (Silver et al., 2016)

Overview

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

AlphaGo

AlphaGo Zero

AlphaZero

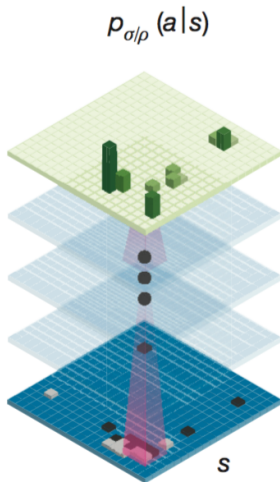
Summary

- **Input:** $19 \times 19 \times 48$ image stack representing player's and opponent's board positions, number of opponent's stones that could be captured there, etc.
- **Training**
 - **Supervised learning** (classification) of policy networks p_π and p_σ based on expert moves for states
 - **Transfer learning** from p_σ to policy network p_ρ
 - **Reinforcement learning** to refine p_ρ via **policy gradient** and **self-play**
 - **Regression** to learn value network v_θ
- **Live play**
 - Uses these networks in **Monte Carlo tree search** to choose actions during games
- 99.8% winning rate vs other Go programs and defeated human Go champion 5-0

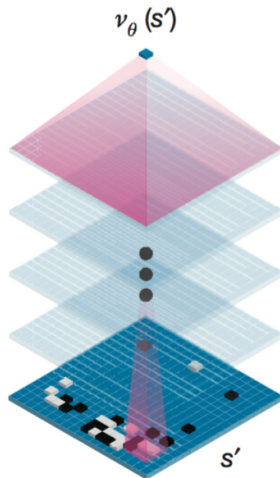
AlphaGo (Silver et al., 2016)

Overview

Policy network



Value network



Reinforcement Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

AlphaGo

AlphaGo Zero

AlphaZero

Summary

AlphaGo (Silver et al., 2016)

Supervised Learning

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

AlphaGo

AlphaGo Zero

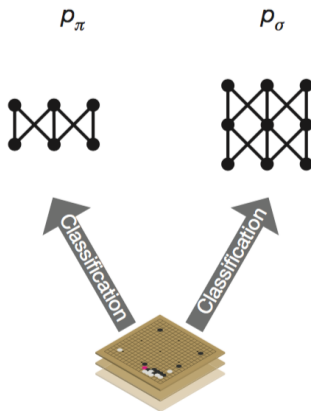
AlphaZero

Summary

- **Supervised learning** of policies p_π and p_σ
- Board positions from **KGS Go Server**, labels are experts' moves
- p_σ is full network (accuracy 57%, 3ms/move), p_π is simpler (accuracy 24% $2\mu\text{s/move}$)

Rollout policy

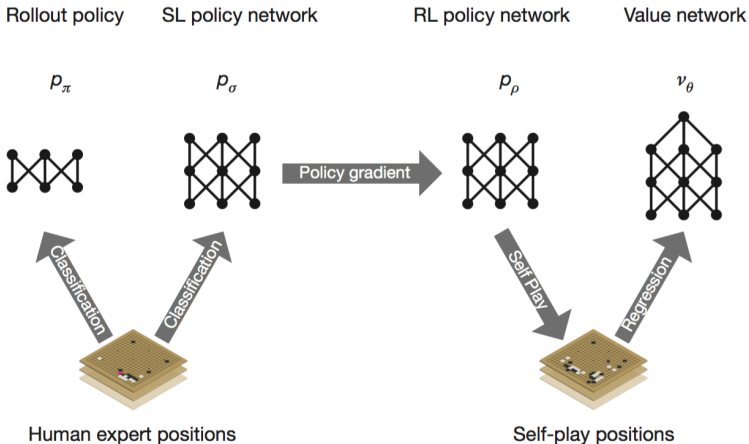
SL policy network



AlphaGo (Silver et al., 2016)

Transfer Learning

Transfer learning of p_σ to p_ρ (same arch., copy parameters)



AlphaGo (Silver et al., 2016)

Reinforcement Learning

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

AlphaGo

AlphaGo Zero

AlphaZero

Summary

- Trained p_ρ via play against $p_{\tilde{\rho}}$ (randomly selected earlier version of p_ρ)
- For state s_t , **terminal reward** $z_t = +1$ if game ultimately won from s_t and -1 otherwise
- Note p_ρ does not compute value of actions like *Q*-learning does
 - It **directly** implements a policy that outputs a_t given s_t
 - Use **policy gradient** method to train:
 - If agent chooses action a_t in state s_t and ultimately wins 90% of the time, what should happen to $p_\rho(a_t | s_t)$?
 - How can we make that happen?

AlphaGo (Silver et al., 2016)

Policy Gradient

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

 Q Learning

TD Learning

DQN

Atari Example

Go Example

AlphaGo

AlphaGo Zero

AlphaZero

Summary

- **REINFORCE: REward Increment = Nonnegative Factor times Offset Reinforcement times Characteristic Eligibility**
- Perform **gradient ascent** to increase probability of actions that on average lead to greater rewards:

$$\Delta \rho_j = \alpha(r - b_s) \frac{\partial \log p_{\rho}(a | s)}{\partial \rho_j} ,$$

α is learning rate, r is reward, a is action taken in state s , and b_s is **reinforcement baseline** (independent of a)

- b keeps expected update same but reduces variance
- E.g., if all actions from s good, b_s helps differentiate
- Common choice: $b_s = \hat{v}(s)$ = estimated value of s

AlphaGo (Silver et al., 2016)

Policy Gradient

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

AlphaGo

AlphaGo Zero

AlphaZero

Summary

- AlphaGo uses REINFORCE with baseline $b_s = v_{\theta}(s)$, $r = z_t$, and sums over all game steps $t = 1, \dots, T$
- Average updates over games $i = 1, \dots, n$

$$\Delta \rho = \frac{\alpha}{n} \sum_{i=1}^n \sum_{t=1}^{T^i} (z_t^i - v_{\theta}(s_t^i)) \nabla_{\rho} \log p_{\rho}(a_t^i | s_t^i)$$

AlphaGo (Silver et al., 2016)

Value Learning

Reinforcement Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

AlphaGo

AlphaGo Zero

AlphaZero

Summary

- $v_{\theta}(s)$ approximates $v^{p_{\rho}}(s)$ = value of s under policy p_{ρ}
- Regression problem on state-outcome pairs (s, z)
- Train with MSE
- Analogous to experience replay, mitigated overfitting by drawing each instance from a unique self-play game:
 - 1 Choose time step U uniformly from $\{1, \dots, 450\}$
 - 2 Play moves $t = 1, \dots, U$ from p_{σ}
 - 3 Choose move a_U uniformly
 - 4 Play moves $t = U + 1, \dots, T$ from p_{ρ}
 - 5 Instance (s_{U+1}, z_{U+1}) added to train set

AlphaGo (Silver et al., 2016)

Live Play

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

AlphaGo

AlphaGo Zero

AlphaZero

Summary

- Now, we're ready for live play
- Rather than exclusively using p_ρ or v_θ to determine actions, will instead base action choice on a **rollout algorithm**
- Use the functions learned to simulate game play from state s forward in time ("rolling it out") and computing statistics about the outcome
- Repeat as much as time limit allows, then choose most favorable action
 - ⇒ **Monte Carlo Tree Search** (MCTS)

AlphaGo (Silver et al., 2016)

Monte Carlo Tree Search

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

AlphaGo

AlphaGo Zero

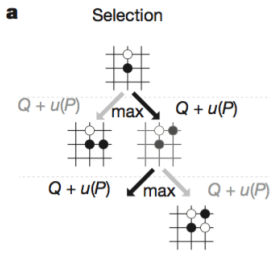
AlphaZero

Summary

- Given current state s , MCTS runs four operations:
 - Selection:** Given a tree rooted at s , follow **tree policy** to traverse and select a leaf node
 - Expansion:** Expand selected leaf by adding children
 - Evaluation (simulation):** Perform rollout to end of game
 - Use p_π to speed up this part
 - Backup:** Use rollout results to update action values of tree
- Each tree edge $((s, a)$ pair) has statistics:
 - Prior probability** $P(s, a)$
 - Action values** $W_v(s, a)$ and $W_r(s, a)$
 - Value counts** $N_v(s, a)$ and $N_r(s, a)$
 - Mean action value** $Q(s, a)$
- After many parallel simulations, choose action maximizing $N_v(s, a)$

AlphaGo (Silver et al., 2016)

Monte Carlo Tree Search: Selection



- Before reaching leaf state, choose action

$$a_t = \operatorname{argmax}_a (Q(s_t, a) + u(s_t, a)) ,$$

where

$$u(s, a) = cP(s, a) \frac{\sqrt{\sum_b N_r(s, b)}}{1 + N_r(s, a)}$$

- I.e., if (s_t, a_t) has been evaluated a lot relative to other actions from s_t , then $N_r(s_t, a_t)$ is large and a_t is evaluated mainly by Q
- Otherwise, exploration is encouraged
- To avoid all searches choosing same actions: When (s_t, a_t) chosen, update stats as if n_{vl} games lost

$$N_r(s_t, a_t) = N_r(s_t, a_t) + n_{vl}$$

$$W_r(s_t, a_t) = W_r(s_t, a_t) - n_{vl}$$

AlphaGo (Silver et al., 2016)

Monte Carlo Tree Search: Expansion

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

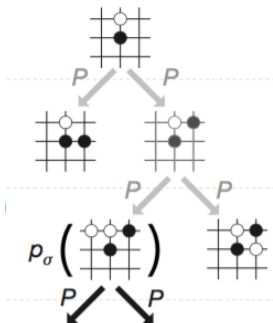
AlphaGo

AlphaGo Zero

AlphaZero

Summary

b Expansion



- If $N_r(s, a) > n_{thr}$, expand next state s' in tree

$$[N_v(s', a) = N_r(s' a) = 0, W_v(s' a) = W_r(s', a) = 0, P(s', a) = p_{\sigma}(a | s')]$$

AlphaGo (Silver et al., 2016)

Monte Carlo Tree Search: Evaluation

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

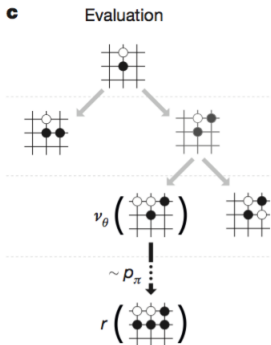
Go Example

AlphaGo

AlphaGo Zero

AlphaZero

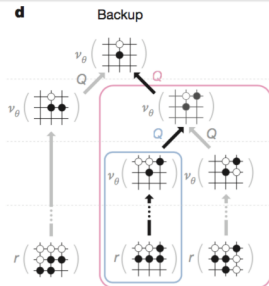
Summary



- Expand from leaf s_L until game ends
- At each time $t \geq L$, **each player** chooses $a_t \sim p_\pi$
- At game's end, compute $z_t = \pm 1$ for all t

AlphaGo (Silver et al., 2016)

Monte Carlo Tree Search: Backup



At end of simulated game, update statistics for all steps $t \leq L$

1 Undo virtual loss and update z :

$$N_r(s_t, a_t) = N_r(s_t, a_t) - n_{vl} + 1$$

$$W_r(s_t, a_t) = W_r(s_t, a_t) + n_{vl} + z_t$$

2 After leaf evaluation done:

$$N_v(s_t, a_t) = N_v(s_t, a_t) + 1$$

$$W_v(s_t, a_t) = W_v(s_t, a_t) + v_\theta(s_L)$$

3 Take weighted average for final action value:

$$Q(s, a) = (1 - \lambda) \left(\frac{W_v(s, a)}{N_v(s, a)} \right) + \lambda \left(\frac{W_r(s, a)}{N_r(s, a)} \right)$$

AlphaGo Zero (Silver et al., 2017)

Overview

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

AlphaGo

AlphaGo Zero

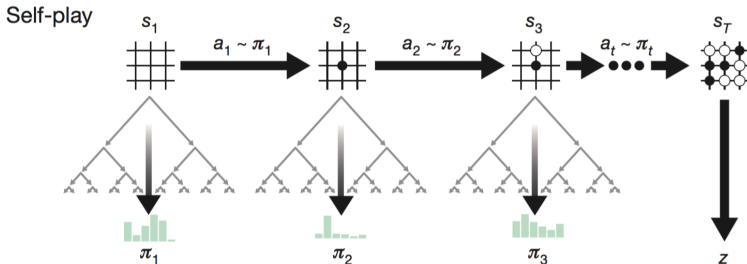
AlphaZero

Summary

- The “Zero” refers to zero human knowledge
- No supervised training from KGS Go data
 - Trained only via RL in self-play
 - Trained a single network $(p, v) = f_{\theta}$ for both policy and value
- Integrated MCTS into training as well as live play
 - Folded lookahead search into training loop
 - Did not rollout to end of game
- Input: $19 \times 19 \times 17$ image stack:
 - Eight of 17 binary planes indicate locations of player's stones the past 8 time steps
 - Eight of 17 binary planes indicate locations of opponent's stones the past 8 time steps
 - Final plane indicates color to play
- Discovered new Go knowledge during self-play, including previously unknown tactics

AlphaGo Zero (Silver et al., 2017)

Self-Play



- Play games against self, choosing actions $a_t \sim \pi_t$ via MCTS
- Outcome of game recorded as $z = \pm 1$

AlphaGo Zero (Silver et al., 2017)

Training

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

Q Learning

TD Learning

DQN

Atari Example

Go Example

AlphaGo

AlphaGo Zero

AlphaZero

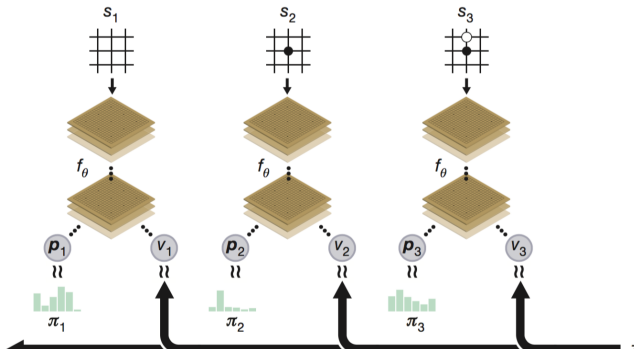
Summary

- Training is a form of **policy iteration**: Alternating between
 - **Policy evaluation**: Estimating value v of policy p
 - **Policy improvement**: Improving policy wrt v
- Use MCTS to map NN policy p to search policy π
- Self-play outcomes inform updates to v

AlphaGo Zero (Silver et al., 2017)

Training

Neural network training

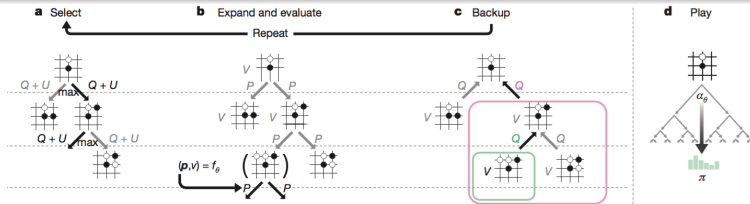


- State s_t 's targets are distribution π_t and reward z_t
- Update network using loss function

$$\underbrace{\text{sq loss}}_{(z_t - v(s_t))^2} - \underbrace{\text{CE}}_{\pi_t^\top \log p_t} + \underbrace{\text{regularizer}}_{+c\|\theta\|^2}$$

AlphaGo Zero (Silver et al., 2017)

MCTS



- MCTS similar to that of AlphaGo, but drop N_r and W_r since no rollout: $[N(s, a), W(s, a), Q(s, a), P(s, a)]$

- Select:** same as before, but $u(s, a)$ uses N instead of N_r
- Expand + evaluate:** f_θ compute value $v(s)$ (modulo symmetry) for backup instead of rollout to game end
- Backup:** same as before, but no N_r or W_r
- Play policy:** $\pi(a | s_0) = N(s_0, a)^{1/\tau} / \sum_b N(s_0, b)^{1/\tau}$ (τ controls exploration)

AlphaZero (Silver et al., 2017b)

Reinforcement
Learning

Stephen Scott

Introduction

MDPs

 Q Learning

TD Learning

DQN

Atari Example

Go Example

AlphaGo
AlphaGo Zero

AlphaZero

Summary

- AlphaGo Zero's approach applied to chess and shogi
- Same use of $(p, v) = f_{\theta(s)}$ and MCTS
- Go-specific parts removed + other generalizations
- No game-specific hyperparameter tuning
 - Similar framework as Atari

Game	White	Black	Win	Draw	Loss
Chess	<i>AlphaZero</i>	<i>Stockfish</i>	25	25	0
	<i>Stockfish</i>	<i>AlphaZero</i>	3	47	0
Shogi	<i>AlphaZero</i>	<i>Elmo</i>	43	2	5
	<i>Elmo</i>	<i>AlphaZero</i>	47	0	3
Go	<i>AlphaZero</i>	<i>AG0 3-day</i>	31	–	19
	<i>AG0 3-day</i>	<i>AlphaZero</i>	29	–	21

Table 1: Tournament evaluation of *AlphaZero* in chess, shogi, and Go, as games won, drawn or lost from *AlphaZero*'s perspective, in 100 game matches against *Stockfish*, *Elmo*, and the previously published *AlphaGo Zero* after 3 days of training. Each program was given 1 minute of thinking time per move.

- Reinforcement learning trains an **agent** to perform **actions** in an **environment**
 - Goal is to find a **policy** to maximize **expected discounted cumulative reward**
 - Modeled as a **Markov decision process**
- **Q learning** is a process that learns a **Q function** that estimates total discounted reward if action a taken in state s and optimal choices from then on
- The Q function is often approximated by an ANN
- **Policy gradient** directly updates the policy to increase the probability of actions that lead to greater rewards