Martin Matyášek

Artificial Intelligence Center
Czech Technical University in Prague

October 27, 2016
learning what to do to maximize future reward

general-purpose framework extending sequential decision making when the model of the environment is unknown
Deep Reinforcement Learning

**RL background**

- let’s assume MDP $\langle S, A, P, R, s_0 \rangle$
  - RL deals with situation where the environment model $P$ and $R$ is unknown
  - can be generalized to *Stochastic Games* $\langle S, N, A, P, R \rangle$

- RL agent includes:
  - **policy** $a = \pi(s)$ (deterministic), $\pi(a | s) = P(a | s)$ (stochastic)
  - **value function** $Q^\pi(a | s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s, a]$
    - Bellman Eq.: $Q^\pi(a | s) = \mathbb{E}_{s', a'} [r + \gamma Q^\pi(a' | s') | s, a]$
    - opt. value functions: $Q^*(s, a) = \mathbb{E}_{s'} [r + \gamma \max_{a'} Q^*(s', a') | s, a]$
    - opt. policy: $\pi^*(s) = \arg\max_a Q^*(s, a)$
  - **model** - learned proxy for environment
Deep Reinforcement Learning

RL Types

1. **value-based** RL
   - estimate the opt. value function $Q^*(s, a)$
   - max. value achievable under *any* policy

2. **policy-based** RL
   - search directly for the opt. policy $\pi^*$
   - i.e. policy yielding max. future reward

3. **model-based** RL
   - build a model of the environment
   - plan using this model
Q-learning

**Algorithm 1 Q-learning**

1: initialize the Q-function and V values (arbitrarily)
2: repeat
3: observe the current state $s_t$
4: select action $a_t$ and take it
5: observe the reward $R(s_t, a_t, s_{t+1})$
6: $Q_{t+1}(s_t, a_t) \leftarrow (1 - \alpha_t)Q_t(s_t, a_t) + \alpha_t(R(s_t, a_t, s_{t+1}) + \gamma V_t(s_{t+1}))$
7: $V_{t+1}(s) \leftarrow \max_a Q_t(s, a)$
8: until convergence
Q-learning

- **model-free** method
- **temporal-difference** version:
  \[ Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \]
  
  value based on next state

- converges to \( Q^*, V^* \) iff \( 0 \leq \alpha_t < \infty \), \( \sum_{t=0}^{\infty} \alpha_t = \infty \) and \( \sum_{t=0}^{\infty} \alpha_t^2 < \infty \)
- **zero-sum Stochastic Games**:  
  - cannot simply use \( Q_{i}^{\pi} : S \times A_i \rightarrow \mathbb{R} \) but rather \( Q_{i}^{\pi} : S \times A \rightarrow \mathbb{R} \)  
  - **minimax-Q** converges to NE  
  - **R-max**: converge to \( \epsilon - \text{Nash} \) with prob. \( (1 - \delta) \) in poly. # steps (PAC learn)
Q-Networks

- \( Q^*(s, a) \approx Q(s, a, w) \)
- treat right hand side \( r + \gamma \max_{a'} Q(s', a', w) \) of Bellman’s Eq. as target
- minimize **MSE loss** \( l = (r + \max_{a'} Q(s', a', w) - Q(s, a, w))^2 \) by stochastic gradient descent

---

David Silver, Google DeepMind
Q-learning summary

+ converges to $Q^*$ using table lookup representation
− diverges using NN:
  ■ correlations between samples
  ■ non-stationary targets

! go deep
Deep Reinforcement Learning

Deep Q-Networks (DQN)

- basic approach is called **experience replay**
- idea: remove correlations by building data-set from agent’s experience \( e = (s, a, r, s') \)
  - sample experiences from \( D_t = \{ e_1, e_2, \ldots e_t \} \) and apply update
- deal with non-stationarity by fixing \( w^- \) in
  \[
  l = (r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w))^2
  \]
Deep Reinforcement Learning

Algorithm 2 Deep Q-learning algorithm

1: init. replay memory $D$, init. $Q$ with random weights
2: observe initial state $s$
3: repeat
4: with prob. $\epsilon$ select random $a$, select $a = \text{argmax}_{a'} Q(s, a')$
5: carry out $a$, observe $(r, s')$ and store $(s, a, r, s')$ in $D$
6: sample random transition $(s_s, a_a, r_r, s_s')$ from $D$
7: calculate target for each minibatch transition:
8: if $ss'$ is terminal state then
9: $tt \leftarrow rr$
10: else
11: $tt \leftarrow rr + \gamma \max_{a'} Q(ss', aa')$
12: end if
13: train the Q-network using $(tt - Q(ss, aa))^2$ as loss
14: $s \leftarrow s'$
15: until convergence
DQN in Atari

David Silver, Google DeepMind
Deep Reinforcement Learning

DQN in Atari - setting

- **state**: stack of raw pixels from last 4 frames
- **actions**: 18 joystick/button positions
- **reward**: delta in score
- **learn**: $Q(s, a)$

David Silver, Google DeepMind
Deep Reinforcement Learning

DQN in Atari - results

![Graph showing performance of DQN compared to human-level performance on various Atari games.](image-url)
DQN improvements

- **Double DQN** removes bias caused by $\max_a Q(\cdot)$
  - current QN - $w$ used to select actions
  - old QN - $w^-$ used to evaluate actions

- **Prioritized Replay** weight experience according to DQN error (stored in PQ)

- **Duelling Network** split Q-Network into:
  - action-independent *value* function
  - action-dependent *advantage* function
Deep Reinforcement Learning

General Reinforcement Learning Architecture (GORILA)
Deep Reinforcement Learning

Deep Policy Network

- parametrize the policy $\pi$ by a DNN and use SGD to optimize weights $u$
  - $\pi(a \mid s, u)$ or $\pi(s, u)$
  - $\max_u L(u) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r_t \mid \pi(\cdot, u) \right]$

- policy gradients:
  - $\frac{\partial L(u)}{\partial u} = \mathbb{E} \left[ \frac{\partial \log \pi(a \mid s, u)}{\partial u} Q^\pi(s, a) \right]$ for stochastic policy $\pi(a \mid s, u)$
  - $\frac{\partial L(u)}{\partial u} = \mathbb{E} \left[ \frac{\partial Q^\pi(s, a)}{\partial a} \frac{\partial a}{\partial u} \right]$ for deterministic policy $a = \pi(s)$ where $a$ is cont. and $Q$ diff.

- variations: *Actor-Critic alg.*, A3C, *Fictitious Self-Play*
Game Theory 101

normal-form game \( G = (\mathcal{N}, \mathcal{A}, u) \)
- \( \mathcal{N} = \{1, 2, \ldots, n\} \) - players
- \( \mathcal{A} = \times_{i \in \mathcal{N}} A_i \) - pure strategies (actions)
- \( \Pi = \times_i \Pi_i = \times_i \Delta(A_i) \) - mixed strategies
- \( u = (u_1, \ldots, u_n), \ u_i : \mathcal{A} \to \mathbb{R} \) - utilities

best response

\[ BR_i(\pi_{-i}) = \{ \pi_i \in \Pi_i \mid \forall \pi'_i \neq \pi_i : u_i(\pi_i, \pi_{-i}) \geq u_i(\pi'_i, \pi_{-i}) \} \]

Nash equilibrium

\[ NE(G) = \{ \pi \in \Pi \mid \forall i \in \mathcal{N} : \pi_i \in BR_i(\pi_{-i}) \} \]
Fictitious Self Play (FSP)

- **fictitious play (FP)**
  1. initialize beliefs about the opponent’s strategy
  2. play a best response to the assessed strategy of the opponent
  3. observe the opponent’s actual play and update beliefs accordingly, *goto 2*

- **fictitious self play (FSP)**
  - DQN with experience replay learns “BR” to opponent policies
  - policy network learns an average of BRs

\[
\frac{\partial l}{\partial w} = \frac{\partial \log \pi_w(a | s)}{\partial w}
\]

- actions a sample mix of policy network and best response
Deep RL Summary

- RL steps in when the model of the environment is unknown
- NN employed when the state space is too large
- RL sets the learning objective, NN then approximates:
  - value function $V^* \approx V_w$, $Q^* \approx Q_w$
  - policy $\pi^* \approx \pi_u$
- use DNN to deal with convergence and correlations

What about some applications?
MvM History

Man vs Machine:

1992 backgammon  *Tesauro* very close to top human experts
1996 chess  Kasparov loses 2.5-3.5 vs *Deep Blue*
1997 othello  Logistello vs Murakami 6-0
2007 checkers  solved
2008 poker  Polaris wins vs poker pros

- 2015 - Heads-up limit Texas hold’em solved
Deep Reinforcement Learning

Go
The Rules of Go

(a) capture

(b) territory

start  empty board
move   place one stone (of your color)
goal   surround
win    control more than half of the board
Go in Theory

- 2-players (just me and the opponent)
- zero-sum game (win for me = loss for opponent)
- perfect information
- finite (the game rules ensure this)

NE exists and we can search for it:
  - minimax
  - alpha-beta
  - negascout
there is $\mathcal{O}(b^d)$ game states where $b \approx 250$ and $d \approx 150$
Monte Carlo Tree Search (MCTS):

- popular heuristic search algorithm for game play
- Monte Carlo rollouts to estimate $v(s) \approx v^*(s)$ (reduces $d$)
- sampling actions from $p(a \mid s)$ reduces $b$
  - MC rollouts search to max. depth without branching at all
MCTS

Figure from Chaslot (2006)
the strongest Go programs are based on MCTS
enhanced by policies that are trained to predict human expert moves
- early rules hand-made
- later ML based on simple features (lin. comb. of inputs)

knowledge learned:
(i) fast (simple) knowledge used for move selection in simulation (rollout policy)
(ii) slower (better) knowledge used for move ordering in tree search (SL policy)
2016: Lee Sedol vs. AlphaGo
<table>
<thead>
<tr>
<th>name</th>
<th>Lee Sedol</th>
<th>AlphaGo</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>33</td>
<td>2</td>
</tr>
<tr>
<td>rank</td>
<td>9 dan prof.</td>
<td>none</td>
</tr>
<tr>
<td>titles</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>power</td>
<td>1 brain</td>
<td>1200 CPU and 200 GPU</td>
</tr>
<tr>
<td>results</td>
<td>loss, loss, loss, <strong>win</strong>, loss, <strong>win</strong>, loss</td>
<td><strong>win</strong>, <strong>win</strong>, <strong>win</strong>, loss, <strong>win</strong></td>
</tr>
<tr>
<td>experience</td>
<td>c · 1k games</td>
<td>c · 1M self-play games</td>
</tr>
</tbody>
</table>
How?
Deep Reinforcement Learning

Science

At last — a computer program that can beat a champion Go player

ALL SYSTEMS GO
Deep Reinforcement Learning

AlphaGo Design

- normal search - MCTS
- simulation (rollout) policy - relatively normal
- supervised learning (SL) policy from master games - improved in details, more data
- RL from self-play for value network
- RL from self-play for policy network
Convolutional Neural Network
Value Network
Deep Reinforcement Learning

Policy Network

Move probabilities

Position

\[ p_\sigma(a|s) \]
Reducing $d$ with Value Network
Reducing $d$ with Value Network
Reducing $b$ with Policy Network
Deep RL in AlphaGo

Rollout policy

SL policy network

RL policy network

Value network

$p_\pi$

$p_\sigma$

$p_\rho$

$v_\theta$

Neural network

Policy gradient

Classification

Human expert positions

Classification

Self-play positions

Regression

Data

Human expert positions

Self-play positions
Deep Reinforcement Learning

## SL of Policy Networks

- **network**: 12 layer convolutional NN
- **training data**: 30M position from human experts (KGS 5+ dan)
- **training alg.**: max. likelihood by SGD

\[
\Delta \sigma \propto \frac{\partial \log p_{\sigma}(a | s)}{\partial \sigma}
\]

- **training time**: 4 weeks on 50 GPUs (Google Cloud)
- **results**: 57% accuracy on held out test data
  - state-of-the-art was 44%
Deep Reinforcement Learning

**RL of Policy Networks**

- **network**: 12 layer convolutional NN
- **training data**: games of self-play between policy networks
- **training alg.**: max. wins \( z \) by policy gradient RL

\[
\Delta \rho \propto \frac{\partial \log p_\rho(a | s)}{\partial \rho} z
\]

- **training time**: 1 week on 50 GPUs (Google Cloud)
- **results**: 80% vs. SL network \( p_\sigma \)
  - raw network \( \sim 3 \) amateur dan

---

**Diagram**:
- **Human expert positions**
- **Supervised Learning policy network**
- **Reinforcement Learning policy network**
- **Self-play data**
- **Value network**

---

Martin Matyášek  
VPD, 2016
Deep Reinforcement Learning

RL of Value Networks

network 12 layer convolutional NN
training data 30M games of self-play
idea

\[ v^p(s) = \mathbb{E}[Z_t \mid s_t = s, a_t, ..., T \sim \rho] \]

\[ v_\theta(s) \approx v^p_\rho(s) \approx v^*(s) \]

training alg. min. MSE by SGD

\[ \Delta \theta \propto \frac{\partial v_\theta(s)}{\partial \theta} (z - v_\theta(s)) \]

training time 1 week on 50 GPUs (Google Cloud)
results first strong position eval. function

Human expert positions
Supervised Learning policy network
Reinforcement Learning policy network
Self-play data
Value network

Classification Self Play Self Play Regression
MCTS in AlphaGo

- Each edge stores:
  - Action value: \( Q(s, a) \)
  - Visit count: \( N(s, a) \)
  - Prior prob.: \( P(s, a) \) (initialized to \( P(s, a) = p_\sigma(a | s) \))

- At each step \( t \) we select in state \( s_t \):
  \[
  a_t = \arg\max_a (Q(s_t, a) + u(s_t, a))
  \]

- \( u(s, a) \propto \frac{P(s, a)}{1 + N(s, a)} \) decays with repeated visits (encourages exploration)

- Leaf node \( s_L \) is evaluated in two different ways:
  \[
  V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_L
  \]
  1. By value network \( v_\theta(s_L) \)
  2. By outcome \( z_L \sim^* p_\pi \)
MCTS in AlphaGo: selection

\[ P \text{ prior probability} \]
\[ Q \text{ action value} \]
\[ u(P) \propto \frac{P}{N} \]
MCTS in AlphaGo: expansion

\[ p_\sigma \] Policy network

\[ P \] prior probability
MCTS in AlphaGo: evaluation

\[ v_\theta \] Value network
MCTS in AlphaGo: rollout

\[ v_\theta \]
\[ r \]

Value network
Game scorer
MCTS in AlphaGo: backup

$Q$  Action value

$\nu_\theta$  Value network

$r$  Game scorer
AlphaGo: results
Deep Reinforcement Learning

References


- Silver, David. *AlphaGo slides*. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Resources_files/AlphaGo_IJCAI.pdf