Introduction to Deep Reinforcement Learning

Yen-Chen Wu

2015/12/11
Outline

• Reinforcement Learning
• Markov Decision Process
• How to Solve MDPs
  – DP
  – MC
  – TD
  – Q-learning (DQN)
• Paper Review
REINFORCEMENT LEARNING
Branches of Machine Learning
What makes different?

• There is no supervisor, only a reward signal
• Feedback is delayed, not instantaneous
• Time really matters (sequential, non i.i.d data)
• Agent’s actions affect the subsequent data it receives
Goal:
Maximize Cumulative Reward

- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
Agent & Environment

→←↑↓
Defense
Attack
Jump

observation

Input
Hidden
Output

reward

$O_t$

$A_t$

$R_t$
Markov Processes
Markov Reward Processes
Markov Decision Processes

MARKOV DECISION PROCESS
Markov Process

Example: Student Markov Chain
Markov Reward Processes

Example: Student MRP
Markov Decision Process

Example: Student MDP
Markov Decision Process (MDP)

- $S$: finite set of states (observations)
- $A$: finite set of actions
- $P$: transition probability
- $R$: immediate reward
- $\gamma$: discount factor

Goal:
- Choose policy $\pi$
- Maximize expected return: $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$
Dynamic Programming
Monte-Carlo
Temporal-Difference
Q-Learning

HOW TO SOLVE MDP
Model-based

• Dynamic Programming
  – Evaluate policy
  – Update policy

Right: A simple Gridworld solved with a Dynamic Programming. Very exciting. Head over to the GridWorld: DP demo to play with the GridWorld environment and policy iteration.
Model Free

- Unknown Transition Probability & Reward
- MC vs TD
Model Free:
Q-learning

• Instead of tabular
• optimal action-value **function** (Q-learning)

\[- Q^*(s,a) = \max_{\pi} \mathbb{E}_t[R_t | s_t = s, a_t = a, \pi] \]

• Bellman equation

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

• Basic idea : iterative update (lack of generalization)
• In practical : function approximator
  • Linear ?
  • Using DNN !
DEEP Q-NETWORK (DQN)
Video

• https://www.youtube.com/watch?v=LJ4oCb6u7kk
Deep Q-Network

- compute Q-values for all actions

Input: 84x84x4

- Convolves 32 filters of 8x8 with stride 4
- Convolves 64 filters of 4x4 with stride 2
- Convolves 64 filters of 3x3 with stride 1

Full-connected 512 nodes

Output a node for each action

Convolutional layers followed by fully connected layers with 512 nodes.
Update DQN

• Loss function

\[
L_i(\theta_i) = \mathbb{E}_{s,a,r} \left[ (\mathbb{E}_{s'} [y|s,a] - Q(s,a; \theta_i))^2 \right]
\]

\[
= \mathbb{E}_{s,a,r,s'} \left[ (y - Q(s,a; \theta_i))^2 \right] + \mathbb{E}_{s,a,r} [\nabla_{s'} [y]]
\]

• Gradient

\[
\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'} \left[ \left( r + \gamma \max_{a'} Q(s',a'; \theta_i) - Q(s,a; \theta_i) \right) \nabla_{\theta_i} Q(s,a; \theta_i) \right]
\]
Two Technique

• Experience Replay
  – Experience
  – Pooled Mer: \( e_t = (s_t, a_t, r_t, s_{t+1}) \)
    • Data efficiency: \( D_t = \{e_1, \ldots, e_t\} \)
    • Avoid correlation between samples (variance between batches)
    • Off–policy is suitable for Q-learning
  – Random sampled mini-batch

<table>
<thead>
<tr>
<th>Example</th>
<th>Learn the value of...</th>
<th>Pros &amp; Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-policy</td>
<td>SARSA</td>
<td>policy being carried out by the agent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Off-policy</td>
<td>DQN</td>
<td>optimal policy independently of the agent's actions</td>
</tr>
</tbody>
</table>
PAPER REVIEW
Paper list

• Massively Parallel Methods for Deep Reinforcement Learning
• Continuous control with deep reinforcement learning
• Deep Reinforcement Learning with Double Q-learning
• Policy Distillation
• Dueling Network Architectures for Deep Reinforcement Learning
• Multiagent Cooperation and Competition with Deep Reinforcement Learning
Massively Parallel Methods for Deep Reinforcement Learning
Arun Nair
arXiv:1507.04296

- **Parallel acting:** generate new interactions
- **Distributed replay memory:** save interactions
- **Parallel learning:** compute gradients from replayed interactions
- **Distributed neural network:** update network from gradients
DDPG (Deterministic Policy Gradient)

- DDAC (Deep Deterministic Actor-Critic)

Continuous control with deep reinforcement learning
Timothy P. Lillicrap
arXiv:1509.02971
https://goo.gl/J4PIAz
Double Q-learning

\[ Y_t^{DQN} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t^-) . \]

\[ Y_t^{DoubleQ} \equiv R_{t+1} + \gamma Q(S_{t+1}, \text{argmax}_a Q(S_{t+1}, a; \theta_t); \theta'_t) . \]
Policy Distillation

- Soft target

Figure 3: Performance of multi-task agents with identical network architecture and size, relative to respective single-task DQN teachers. A detailed results table is given in Appendix B.
Dueling Network

Figure 2: See, attend and drive: Value and advantage saliency maps on the Enduro game for a trained dueling architecture. The value stream learns to pay attention to the road. The advantage stream learns to pay attention only when there are cars immediately in front, so as to avoid collisions.
Multiagent

The agents manage to hit the ball a few times per round.