Deep RL applications and challenges
Overview

- Playing Go: AlphaGo, AlphaGo Zero, AlphaZero
- Imitation learning
- Human vs. computer RL
- Curiosity
Playing Go

- Transition model is known and deterministic
- Key challenges: huge state and action space, long sequences, sparse rewards
AlphaGo

- **Policy network:** initialized by supervised learning on human games, improved by policy gradient
- **Value network:** trained to predict outcome of game based on self-play
- Networks are used to guide Monte Carlo tree search (MCTS)

D. Silver et al., *Mastering the Game of Go with Deep Neural Networks and Tree Search*, Nature 529, January 2016
Network architecture

- Input: 19x19x48 feature map
- Architecture of policy networks: 13 layers, 5x5 filter in first layer, 3x3 filters in layers 2-12, 256 feature maps in layers 1-12, 1x1 filters followed by softmax in final layer, ensemble over 8 symmetries

<table>
<thead>
<tr>
<th>Feature</th>
<th># of planes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stone colour</td>
<td>3</td>
<td>Player stone / opponent stone / empty</td>
</tr>
<tr>
<td>Ones</td>
<td>1</td>
<td>A constant plane filled with 1</td>
</tr>
<tr>
<td>Turns since</td>
<td>8</td>
<td>How many turns since a move was played</td>
</tr>
<tr>
<td>Liberties</td>
<td>8</td>
<td>Number of liberties (empty adjacent points)</td>
</tr>
<tr>
<td>Capture size</td>
<td>8</td>
<td>How many opponent stones would be captured</td>
</tr>
<tr>
<td>Self-atari size</td>
<td>8</td>
<td>How many of own stones would be captured</td>
</tr>
<tr>
<td>Liberties after move</td>
<td>8</td>
<td>Number of liberties after this move is played</td>
</tr>
<tr>
<td>Ladder capture</td>
<td>1</td>
<td>Whether a move at this point is a successful ladder capture</td>
</tr>
<tr>
<td>Ladder escape</td>
<td>1</td>
<td>Whether a move at this point is a successful ladder escape</td>
</tr>
<tr>
<td>Sensibleness</td>
<td>1</td>
<td>Whether a move is legal and does not fill its own eyes</td>
</tr>
<tr>
<td>Zeros</td>
<td>1</td>
<td>A constant plane filled with 0</td>
</tr>
<tr>
<td>Player color</td>
<td>1</td>
<td>Whether current player is black</td>
</tr>
</tbody>
</table>

Feature planes used by the policy network (all but last feature) and value network (all features).
Supervised (SL) policy network

• Given position $s$, predict probability distribution over moves $a$
• Trained on 30M positions from human games (KGS Go Server), achieves 57% accuracy on move prediction
• Also train a smaller, faster rollout policy network (24% accuracy)
RL policy network

- Initialize with SL network
- Play current policy network against a randomly selected past snapshot, update parameters using policy gradients
- Use REINFORCE with terminal reward $z = \pm 1$ and baseline $v(s)$ provided by value network:
  \[
  \nabla_\theta J(\theta) \approx (z - v(s_t)) \nabla_\theta \log \pi_\theta(s_t, a_t) \\
  \theta \leftarrow \theta + \eta \nabla_\theta J(\theta)
  \]
- RL network wins against SL network 80% of the time, wins against open-source Pachi Go program 85% of the time
Value network

- Estimate $v(s)$, expected outcome of play starting with position $s$ and following the learned policy for both players
- Train network by minimizing mean squared error (MSE) between actual and predicted outcome
- Training data: 30M positions sampled from different self-play games
Summary of policy and value networks
Monte Carlo Tree Search

- At play time, select moves by MTCS guided by policy and value networks.

Figure 3: Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value $Q$, plus a bonus $u(P)$ that depends on a stored prior probability $P$ for that edge. b, The leaf node may be expanded; the new node is processed once by the policy network $p_\pi$ and the output probabilities are stored as prior probabilities $P$ for each action. c, At the end of a simulation, the leaf node is evaluated in two ways: using the value network $v_\theta$ and by running a rollout to the end of the game with the fast rollout policy $p_\pi$, then computing the winner with function $r$. d, Action values $Q$ are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.
Figure 4 | Tournament evaluation of AlphaGo. a, Results of a tournament between different Go programs (see Extended Data Tables 6–11). Each program used approximately 5 s computation time per move. To provide a greater challenge to AlphaGo, some programs (pale upper bars) were given four handicap stones (that is, free moves at the start of every game) against all opponents. Programs were evaluated on an Elo scale\(^{37}\): a 230 point gap corresponds to a 79% probability of winning, which roughly corresponds to one amateur dan rank advantage on KGS\(^{38}\); an approximate correspondence to human ranks is also shown, horizontal lines show KGS ranks achieved online by that program. Games against the human European champion Fan Hui were also included; these games used longer time controls. 95% confidence intervals are shown. b, Performance of AlphaGo, on a single machine, for different combinations of components. The version solely using the policy network does not perform any search. c, Scalability study of MCTS in AlphaGo with search threads and GPUs, using asynchronous search (light blue) or distributed search (dark blue), for 2 s per move.
AlphaGo vs. Lee Sedol

• March 2016 (after publication of Nature paper): DeepMind’s AlphaGo system beats world champion Lee Sedol 4-1
AlphaGo Zero

• A fancier architecture (ResNets with BatchNorm)
• No hand-crafted features used as input
• Train a single network to simultaneously predict value and policy
• Training is done entirely by RL with self-play, starting with random moves
• MCTS is used inside the RL loop, not outside

D. Silver et al., Mastering the Game of Go without Human Knowledge, Nature 550, October 2017
https://deepmind.com/blog/alphago-zero-learning-scratch/
**RL loop**

- Given a position $s$, use current network to predict move probabilities $p(a|s)$ and value $v(s)$
- Run MCTS guided by the network to obtain refined move probabilities $\pi$ and final value $z$
- Update network parameters to make $p(a|s)$ and $v(s)$ closer to $\pi$ and $z$ (MSE loss on $v$ and cross-entropy on $p$)
Results
Entire human chess knowledge learned and surpassed by DeepMind's AlphaZero in four hours

https://www.telegraph.co.uk/science/2017/12/06/entire-human-chess-knowledge-learned-surpassed-deepminds-alphazero/
AlphaZero

D. Silver et al., Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, arXiv 2017
RL challenges

• What if the task is too hard to bootstrap from random performance?
• What if rewards are too sparse?
• What if a reward function is hard to formulate?
• Why do RL agents learn so much more slowly than humans?
Imitation learning

- In some applications, you cannot bootstrap yourself from random policies
  - High-dimensional state and action spaces where most random trajectories fail miserably
  - Expensive to evaluate policies in the physical world, especially in cases of failure
- **Solution**: learn to imitate sample trajectories or demonstrations
  - This is also helpful when there is no natural reward formulation
Learning visuomotor policies

- **Underlying state** $s$: true object position, robot configuration
- **Observations** $o$: image pixels

Two-part approach:
- Learn *guiding policy* $\pi(a|s)$ using trajectory-centric RL and control techniques
- Learn *visuomotor policy* $\pi(a|o)$ by imitating $\pi(a|s)$

S. Levine et al. *End-to-end training of deep visuomotor policies*. JMLR 2016
Learning visuomotor policies

Neural network architecture
Learning visuomotor policies

Overview video, training video

S. Levine et al. End-to-end training of deep visuomotor policies. JMLR 2016
RL challenges: Curiosity

• How to deal with sparse or non-existent rewards?

(a) learn to explore on Level-1  (b) explore faster on Level-2

Video

RL challenges: Sample complexity

- Why do RL agents take so long to learn compared to humans?

RL challenges: Sample complexity

Performance of humans
RL challenges: Sample complexity

- RL agent is relatively unaffected by most transformations that hinder humans
Challenges for RL and deep learning

- Training faster, generalizing from one task to another
- Dealing with changing environments, multiple agents
- Planning
- Reasoning
- Integrating memory, knowledge