Tutorial: Deep Reinforcement Learning

David Silver, Google DeepMind
Outline

Deep Learning

Reinforcement Learning

Deep Value Functions

Deep Policies

Deep Models
Reinforcement Learning: AI = RL

- RL is a general-purpose framework for artificial intelligence
- We seek a single agent which can solve any human-level task
- The essence of an intelligent agent
- Powerful RL requires powerful representations
Outline

Deep Learning

Reinforcement Learning

Deep Value Functions

Deep Policies

Deep Models
A deep representation is a composition of many functions

\[ x \xrightarrow{w_1} h_1 \xrightarrow{w_2} \ldots \xrightarrow{w_n} h_n \xrightarrow{w_{n+1}} y \]

Its gradient can be backpropagated by the chain rule

\[
\frac{\partial h_1}{\partial x} \leftarrow \frac{\partial h_2}{\partial h_1} \leftarrow \cdots \leftarrow \frac{\partial y}{\partial h_n} \leftarrow \frac{\partial}{\partial y}
\]

\[
\frac{\partial h_1}{\partial w_1} \quad \cdots \quad \frac{\partial h_n}{\partial w_n} \quad \frac{\partial y}{\partial w_{n+1}}
\]
Deep Neural Network

A deep neural network is typically composed of:

- Linear transformations

\[ h_{k+1} = Wh_k \]

- Non-linear activation functions

\[ h_{k+2} = f(h_{k+1}) \]
Weight Sharing

Recurrent neural network shares weights between time-steps

\[ y_1 \rightarrow y_2 \rightarrow \ldots \rightarrow y_n \]

\[ h_0 \xrightarrow{w} h_1 \xrightarrow{w} h_2 \xrightarrow{w} \ldots \xrightarrow{w} h_n \]

\[ x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \]

Convolutional neural network shares weights between local regions

\[ w_1 \xrightarrow{} w_1 \xrightarrow{} w_1 \]

\[ w_2 \xrightarrow{} w_2 \xrightarrow{} w_2 \]

\[ x \rightarrow h_1 \rightarrow h_2 \]
A loss function \( l(y) \) measures goodness of output \( y \), e.g.
- Mean-squared error \( l(y) = ||y^* - y||^2 \)
- Log likelihood \( l(y) = \log \mathbb{P}[y^*|x] \)

The loss is appended to the forward computation:

\[
x \xrightarrow{w_1} h_1 \xrightarrow{w_2} ... \xrightarrow{w_n} h_n \xrightarrow{w_{n+1}} y \xrightarrow{} l(y)
\]

Gradient of loss is appended to the backward computation:

\[
\frac{\partial h_1}{\partial x} \leftarrow \frac{\partial h_2}{\partial h_1} \leftarrow ... \leftarrow \frac{\partial y}{\partial h_n} \leftarrow \frac{\partial l(y)}{\partial y}
\]

\[
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow
\]

\[
\frac{\partial h_1}{\partial w_1} \quad ... \quad \frac{\partial h_n}{\partial w_n} \quad \frac{\partial y}{\partial w_{n+1}}
\]
Stochastic Gradient Descent

- Minimise expected loss $\mathcal{L}(w) = \mathbb{E}_x [l(y)]$
- Follow the gradient of $\mathcal{L}(w)$

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}_x \left[ \frac{\partial l(y)}{\partial w} \right] = \mathbb{E}_x \left( \begin{array}{c} \frac{\partial l(y)}{\partial w^{(1)}} \\ \vdots \\ \frac{\partial l(y)}{\partial w^{(k)}} \end{array} \right)$$

- Adjust $w$ in direction of -ve gradient

$$\Delta w = -\frac{\alpha}{2} \alpha \frac{\partial l(y)}{\partial w}$$

where $\alpha$ is a step-size parameter
Deep Supervised Learning

- Deep neural networks have achieved remarkable success
- Simple ingredients solve supervised learning problems
  - Use deep network as a function approximator
  - Define loss function
  - Optimise parameters end-to-end by SGD
- Scales well with memory/data/computation
- Solves the representation learning problem
- State-of-the-art for images, audio, language, ...
Deep Supervised Learning

- Deep neural networks have achieved remarkable success
- Simple ingredients solve supervised learning problems
  - Use deep network as a function approximator
  - Define loss function
  - Optimise parameters end-to-end by SGD
- Scales well with memory/data/computation
- Solves the representation learning problem
- State-of-the-art for images, audio, language, ...
- Can we follow the same recipe for RL?
Outline

Deep Learning

Reinforcement Learning

Deep Value Functions

Deep Policies

Deep Models
Policies and Value Functions

- **Policy** $\pi$ is a behaviour function selecting actions given states $a = \pi(s)$

- **Value function** $Q^\pi(s, a)$ is expected total reward from state $s$ and action $a$ under policy $\pi$

  $$Q^\pi(s, a) = \mathbb{E} \left[ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a \right]$$

  “How good is action $a$ in state $s$?”
Approaches To Reinforcement Learning

Policy-based RL
- Search directly for the optimal policy $\pi^*$
- This is the policy achieving maximum future reward

Value-based RL
- Estimate the optimal value function $Q^*(s, a)$
- This is the maximum value achievable under any policy
Approaches To Reinforcement Learning

Policy-based RL
- Search directly for the optimal policy $\pi^*$
- This is the policy achieving maximum future reward

Value-based RL
- Estimate the optimal value function $Q^*(s, a)$
- This is the maximum value achievable under any policy

Model-based RL
- Build a transition model of the environment
- Plan (e.g. by lookahead) using model
Deep Reinforcement Learning

- Can we apply deep learning to RL?
- Use deep network to represent value function / policy / model
- Optimise value function / policy / model end-to-end
- Using stochastic gradient descent
Outline

Deep Learning

Reinforcement Learning

Deep Value Functions

Deep Policies

Deep Models
Bellman Equation

- Bellman expectation equation unrolls value function $Q^\pi$

$$Q^\pi(s, a) = \mathbb{E} \left[ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \mid s, a \right]$$

$$= \mathbb{E}_{s', a'} \left[ r + \gamma Q^\pi(s', a') \mid s, a \right]$$
Bellman Equation

- Bellman expectation equation unrolls value function $Q^\pi$

\[
Q^\pi(s, a) = \mathbb{E} \left[ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a \right] \\
= \mathbb{E}_{s', a'} \left[ r + \gamma Q^\pi(s', a') \mid s, a \right]
\]

- Bellman optimality equation unrolls optimal value function $Q^*$

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

- **Bellman expectation equation** unrolls value function $Q^\pi$
  \[
  Q^\pi(s, a) = \mathbb{E} \left[ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a \right] = \mathbb{E}_{s', a'} \left[ r + \gamma Q^\pi(s', a') \mid s, a \right]
  \]

- **Bellman optimality equation** unrolls optimal value function $Q^*$
  \[
  Q^*(s, a) = \mathbb{E}_{s'} \left[ r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]
  \]

- **Policy iteration algorithms** solve Bellman expectation equation
  \[
  Q_{i+1}(s, a) = \mathbb{E}_{s'} \left[ r + \gamma Q_i(s', a') \mid s, a \right]
  \]

- **Value iteration algorithms** solve Bellman optimality equation
  \[
  Q_{i+1}(s, a) = \mathbb{E}_{s', a'} \left[ r + \gamma \max_{a'} Q_i(s', a') \mid s, a \right]
  \]
Policy Iteration with Non-Linear Sarsa

- Represent value function by \textit{Q-network} with weights \( w \)

\[
Q(s, a, w) \approx Q^\pi(s, a)
\]
Policy Iteration with Non-Linear Sarsa

- Represent value function by Q-network with weights $w$

$$Q(s, a, w) \approx Q^\pi(s, a)$$

- Define objective function by mean-squared error in Q-values

$$\mathcal{L}(w) = \mathbb{E} \left[ \left( r + \gamma Q(s', a', w) - Q(s, a, w) \right)^2 \right]$$
Policy Iteration with Non-Linear Sarsa

- Represent value function by Q-network with weights $w$

$$Q(s, a, w) \approx Q^\pi(s, a)$$

- Define objective function by mean-squared error in Q-values

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

- Leading to the following Sarsa gradient

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

- Optimise objective end-to-end by SGD, using $\frac{\partial \mathcal{L}(w)}{\partial w}$
Value Iteration with Non-Linear Q-Learning

- Represent value function by deep Q-network with weights $w$
  
  $Q(s, a, w) \approx Q^\pi(s, a)$

- Define objective function by mean-squared error in Q-values
  
  $L(w) = \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \right]$

- Leading to the following Q-learning gradient
  
  $\frac{\partial L(w)}{\partial w} = \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right) \frac{\partial Q(s, a, w)}{\partial w} \right]$

- Optimise objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$
Example: TD Gammon
Self-Play Non-Linear Sarsa

- Initialised with random weights
- Trained by games of self-play
- Using non-linear Sarsa with afterstate value function

\[ Q(s, a, w) = \mathbb{E} \left[ V(s', w) \right] \]

- Greedy policy improvement (no exploration)
- Algorithm converged in practice (not true for other games)
Self-Play Non-Linear Sarsa

- initialised with random weights
- trained by games of self-play
- using non-linear Sarsa with afterstate value function
  \[ Q(s, a, w) = \mathbb{E} [V(s', w)] \]
- greedy policy improvement (no exploration)
- algorithm converged in practice (not true for other games)
- TD Gammon defeated world champion Luigi Villa 7-1 (Tesauro, 1992)
New TD-Gammon Results

Performance of TD nets with no expert knowledge

- 10 hidden units
- 20 hidden units
- 40 hidden units
- 80 hidden units

expected points per game vs. number of self-play training games
Stability Issues with Deep RL

Naive Q-learning oscillates or diverges with neural nets

1. Data is sequential
   ▶ Successive samples are correlated, non-iid

2. Policy changes rapidly with slight changes to Q-values
   ▶ Policy may oscillate
   ▶ Distribution of data can swing from one extreme to another

3. Scale of rewards and Q-values is unknown
   ▶ Naive Q-learning gradients can be large unstable when backpropagated
Deep Q-Networks

DQN provides a stable solution to deep value-based RL

1. Use experience replay
   - Break correlations in data, bring us back to iid setting
   - Learn from all past policies
   - Using off-policy Q-learning

2. Freeze target Q-network
   - Avoid oscillations
   - Break correlations between Q-network and target

3. Clip rewards or normalize network adaptively to sensible range
   - Robust gradients
Stable Deep RL (1): Experience Replay

To remove correlations, build data-set from agent’s own experience

- Take action $a_t$ according to $\epsilon$-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions $(s, a, r, s')$ from $\mathcal{D}$
- Optimise MSE between Q-network and Q-learning targets, e.g.

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \right]$$
To avoid oscillations, fix parameters used in Q-learning target

- Compute Q-learning targets w.r.t. old, fixed parameters $w^-$

  $$r + \gamma \max_{a'} Q(s', a', w^-)$$

- Optimise MSE between Q-network and Q-learning targets

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

- Periodically update fixed parameters $w^- \leftarrow w$
Reinforcement Learning in Atari
DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels $s$
- Input state $s$ is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step

Network architecture and hyperparameters fixed across all games [Mnih et al.]
DQN Results in Atari
DQN Demo
How much does DQN help?

<table>
<thead>
<tr>
<th></th>
<th>Q-learning</th>
<th>Q-learning + Target Q</th>
<th>Q-learning + Replay</th>
<th>Q-learning + Replay + Target Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakout</td>
<td>3</td>
<td>10</td>
<td>241</td>
<td>317</td>
</tr>
<tr>
<td>Enduro</td>
<td>29</td>
<td>142</td>
<td>831</td>
<td>1006</td>
</tr>
<tr>
<td>River Raid</td>
<td>1453</td>
<td>2868</td>
<td>4103</td>
<td>7447</td>
</tr>
<tr>
<td>Seaquest</td>
<td>276</td>
<td>1003</td>
<td>823</td>
<td>2894</td>
</tr>
<tr>
<td>Space Invaders</td>
<td>302</td>
<td>373</td>
<td>826</td>
<td>1089</td>
</tr>
</tbody>
</table>
Stable Deep RL (3): Reward/Value Range

- DQN clips the rewards to $[-1, +1]$
- This prevents Q-values from becoming too large
- Ensures gradients are well-conditioned
Stable Deep RL (3): Reward/Value Range

- DQN clips the rewards to \([-1, +1]\)
- This prevents Q-values from becoming too large
- Ensures gradients are well-conditioned
- Can’t tell difference between small and large rewards
- Better approach: normalise network output
- e.g. via batch normalisation
Demo: Normalized DQN in PacMan
Outline

Deep Learning

Reinforcement Learning

Deep Value Functions

Deep Policies

Deep Models
Policy Gradient for Continuous Actions

- Represent policy by deep network $a = \pi(s, u)$ with weights $u$
- Define objective function as total discounted reward
  $$J(u) = \mathbb{E} \left[ r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots \right]$$
- Optimise objective end-to-end by SGD
- i.e. Adjust policy parameters $u$ to achieve more reward
Deterministic Policy Gradient

The gradient of the policy is given by

\[
\frac{\partial J(u)}{\partial u} = \mathbb{E}_s \left[ \frac{\partial Q^\pi(s, a)}{\partial u} \right]
\]

\[
= \mathbb{E}_s \left[ \frac{\partial Q^\pi(s, a)}{\partial a} \frac{\partial \pi(s, u)}{\partial u} \right]
\]

Policy gradient is the direction that most improves \( Q \)
Deterministic Actor-Critic

Use two networks

- **Actor** is a policy $\pi(s, u)$ with parameters $u$

  $$s \xrightarrow{u_1} \ldots \xrightarrow{u_n} a$$

- **Critic** is value function $Q(s, a, w)$ with parameters $w$

  $$s, a \xrightarrow{w_1} \ldots \xrightarrow{w_n} Q$$

- **Critic** provides loss function for actor

  $$s \xrightarrow{u_1} \ldots \xrightarrow{u_n} a \xrightarrow{w_1} \ldots \xrightarrow{w_n} Q$$

- Gradient backpropagates from critic into actor

  $$\frac{\partial a}{\partial u} \leftarrow \ldots \leftarrow \frac{\partial Q}{\partial a} \leftarrow \ldots \leftarrow$$
Deterministic Actor-Critic: Learning Rule

- **Critic** estimates value of current policy by Q-learning
  \[
  \frac{\partial L(w)}{\partial w} = \mathbb{E} \left[ \left( r + \gamma Q(s', \pi(s'), w) - Q(s, a, w) \right) \frac{\partial Q(s, a, w)}{\partial w} \right]
  \]

- **Actor** updates policy in direction that improves \( Q \)
  \[
  \frac{\partial J(u)}{\partial u} = \mathbb{E}_s \left[ \frac{\partial Q(s, a, w)}{\partial a} \frac{\partial \pi(s, u)}{\partial u} \right]
  \]
Deterministic Deep Policy Gradient (DDPG)

- Naive actor-critic oscillates or diverges with neural nets
- DDPG provides a stable solution
Deterministic Deep Policy Gradient (DDPG)

- Naive actor-critic oscillates or diverges with neural nets
- DDPG provides a stable solution

1. Use experience replay for both actor and critic
2. Freeze target network to avoid oscillations

\[
\frac{\partial L(w)}{\partial w} = \mathbb{E}_{s,a,r,s' \sim D} \left[ \left( r + \gamma Q(s', \pi(s', u^-), w^-) - Q(s, a, w) \right) \frac{\partial Q(s, a, w)}{\partial w} \right]
\]

\[
\frac{\partial J(u)}{\partial u} = \mathbb{E}_{s,a,r,s' \sim D} \left[ \frac{\partial Q(s, a, w)}{\partial a} \frac{\partial \pi(s, u)}{\partial u} \right]
\]
DDPG for Continuous Control

- End-to-end learning of control policy from raw pixels $s$
- Input state $s$ is stack of raw pixels from last 4 frames
- Two separate convnets are used for $Q$ and $\pi$
- Physics are simulated in MuJoCo
DDPG Demo
Model-Based RL

Learn a transition model of the environment

\[ p(r, s' \mid s, a) \]

Plan using the transition model

- e.g. Lookahead using transition model to find optimal actions
Deep Models

- Represent transition model $p(r, s' | s, a)$ by deep network
- Define objective function measuring goodness of model
- e.g. number of bits to reconstruct next state (Gregor et al.)
- Optimise objective by SGD
DARN Demo
Challenges of Model-Based RL

Compounding errors

- Errors in the transition model compound over the trajectory
- By the end of a long trajectory, rewards can be totally wrong
- Model-based RL has failed (so far) in Atari
Challenges of Model-Based RL

Compounding errors
▶ Errors in the transition model compound over the trajectory
▶ By the end of a long trajectory, rewards can be totally wrong
▶ Model-based RL has failed (so far) in Atari

Deep networks of value/policy can “plan” implicitly
▶ Each layer of network performs arbitrary computational step
▶ $n$-layer network can “lookahead” $n$ steps
▶ Are transition models required at all?
Deep Learning in Go

Monte-Carlo search

- Monte-Carlo search (MCTS) simulates future trajectories
- Builds large lookahead search tree with millions of positions
- State-of-the-art 19 × 19 Go programs use MCTS
- e.g. First strong Go program MoGo

(Gelly et al.)
**Deep Learning in Go**

**Monte-Carlo search**
- Monte-Carlo search (MCTS) simulates future trajectories
- Builds large lookahead search tree with millions of positions
- State-of-the-art 19 × 19 Go programs use MCTS
- e.g. First strong Go program *MoGo* 
  (Gelly et al.)

**Convolutional Networks**
- 12-layer convnet trained to predict expert moves
- Raw convnet (looking at 1 position, no search at all)
- Equals performance of *MoGo* with $10^5$ position search tree
  (Maddison et al.)

<table>
<thead>
<tr>
<th>Program</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human 6-dan</td>
<td>~ 52%</td>
</tr>
<tr>
<td>12-Layer ConvNet</td>
<td>55%</td>
</tr>
<tr>
<td>8-Layer ConvNet*</td>
<td>44%</td>
</tr>
<tr>
<td>Prior state-of-the-art</td>
<td>31-39%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program</th>
<th>Winning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>GnuGo</td>
<td>97%</td>
</tr>
<tr>
<td>MoGo (100k)</td>
<td>46%</td>
</tr>
<tr>
<td>Pachi (10k)</td>
<td>47%</td>
</tr>
<tr>
<td>Pachi (100k)</td>
<td>11%</td>
</tr>
</tbody>
</table>

*Clarke & Storkey*
Conclusion

- RL provides a general-purpose framework for AI
- RL problems can be solved by end-to-end deep learning
- A single agent can now solve many challenging tasks
- Reinforcement learning + deep learning = AI
Questions?

“The only stupid question is the one you never asked” - Rich Sutton