Deep Q-learning Example

Zhonghao Wang
Contents

• Review: Deep Q-learning
• DQN example: Atari breakout
Deep Q learning

• Train a deep neural network to output Q values:

Source: D. Silver
Deep Q learning

• SARSA update: “nudge” $Q(s,a)$ toward value we observe it to have in the most recent action:

$$Q(s,a) \leftarrow Q(s,a) + \alpha (R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

• Deep Q learning: encourage estimate to match the target by minimizing squared error:

$$L(w) = (R(s) + \gamma \max_{a'} Q(s',a';w) - Q(s,a;w))^2$$
Online Q learning algorithm

• Perform action $a$, get observed tuple: $(s,a,s')$
• Observe: $Q^{local}(s,a) = R(s) + \gamma \max_{a'} Q(s',a';W)$
• Update weights to reduce the error
  \[ L(W) = (Q^{local} - Q(s,a;W))^2 \]
• Gradient:
  \[ \nabla_W L = (Q(s,a;W) - Q^{local})\nabla_W Q \]
• Weight update:
  \[ W \leftarrow W - \eta \nabla_W L \]
• This is called stochastic gradient descent (SGD)
• “Stochastic” because the training sample $(s,a,s')$ was chosen at random by our exploration function
Contents

• Review: Deep Q-learning
• DQN example: Atari breakout
Deep Q learning in Atari

Mnih et al. Human-level control through deep reinforcement learning, Nature 2015
State Representation

Think about the **Breakout** game

- How to define a state?
  - Location of the paddle
  - Location/direction of the ball
  - Presence/absence of each individual brick

Let’s make it more universal!

**Screen pixels**
Deep Q learning in Atari

- Input state $s$ is stack of raw pixels from last 4 frames
- Network architecture and hyperparameters fixed for all games

Mnih et al. Human-level control through deep reinforcement learning, Nature 2015
Function Approximator

- Use a function (with parameters) to approximate the Q-function

\[ Q(s, a; \theta) \approx Q^*(s, a) \]

- Linear
- Non-linear: Q-network
• Think about the **Breakout** game
  • State: screen pixels
    • Image size: $84 \times 84$ (resized)
    • Consecutive 4 images
    • Grayscale with 256 gray levels

\[ \begin{array}{c}
256^{84\times84\times4} \\
\end{array} \text{ rows in the Q-table!} \]
Stability issues with Deep RL

- Naïve Q-learning oscillates or diverges with neural nets
  1. Data is sequential
     - Successive samples are correlated, non-i.i.d.
  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-Network provides a stable solution to deep value-based RL

1. Use experience replay
   - Break correlations in data, bring us back to i.i.d. 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 $\varepsilon$-greedy policy
    (Choose "best" action with probability $1 - \varepsilon$, and selects a random action with probability $\varepsilon$)
  - Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$ (Huge data base to store historical samples)
  - Sample random mini-batch of transitions $(s, a, r, s')$ from $\mathcal{D}$
  - Optimize MSE between Q-network and Q-learning targets, e.g.

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[ \left( r + \gamma \max_{a'} Q(s', a' ; \theta_i) - Q(s, a ; \theta_i) \right)^2 \right]$$
Stable Deep RL(2) : Fixed Target Q-Network

- To avoid oscillations, fix parameters used in Q-learning target
  - Compute Q-learning targets w.r.t. old, fixed parameters $\theta_i^-$
    \[
    r + \gamma \max_{a'} Q(s', a'; \theta_i^-)
    \]
  - Optimize MSE between Q-network and Q-learning targets
    \[
    \mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[ \left( r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]
    \]
  - Periodically update fixed parameters $\theta_i^- \leftarrow \theta_i$
Stable Deep RL(3) : Reward / Value Range

- DQN clips the reward to $[-1, +1]$
- This prevents Q-values from becoming too large
- Ensures gradients are well-conditioned
## Stable Deep RL

**DQN**

<table>
<thead>
<tr>
<th>Game</th>
<th>With replay, with target Q</th>
<th>With replay, without target Q</th>
<th>Without replay, with target Q</th>
<th>Without replay, without target Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakout</td>
<td>316.8</td>
<td>240.7</td>
<td>10.2</td>
<td>3.2</td>
</tr>
<tr>
<td>Enduro</td>
<td>1006.3</td>
<td>831.4</td>
<td>141.9</td>
<td>29.1</td>
</tr>
<tr>
<td>River Raid</td>
<td>7446.6</td>
<td>4102.8</td>
<td>2867.7</td>
<td>1453.0</td>
</tr>
<tr>
<td>Seaquest</td>
<td>2894.4</td>
<td>822.6</td>
<td>1003.0</td>
<td>275.8</td>
</tr>
<tr>
<td>Space Invaders</td>
<td>1088.9</td>
<td>826.3</td>
<td>373.2</td>
<td>302.0</td>
</tr>
</tbody>
</table>
During Training

Database $D$ of samples
$(\phi_t, a_t, r_t, \phi_{t+1})$
1 million samples

$m$=mini-batch size

$(\phi_{t1}, a_{t1}, r_{t1}, \phi_{t+1})$
$(\phi_{t2}, a_{t2}, r_{t2}, \phi_{t+1})$
$\cdots$
$(\phi_{tm}, a_{tm}, r_{tm}, \phi_{tm+1})$

Do mini-batch gradient
descent on parameter $\theta$
for one step

Under training
Convolutional
Neural Network
Parameter $\theta$

Input game image

$Q(s_t, a_{t1})$ & $a_{t1}$
$Q(s_t, a_{t2})$ & $a_{t2}$
$\cdots$
$Q(s_t, a_{tm})$ & $a_{tm}$

$a^*_t = \arg\max_a Q(s_t, a)$
with probability $1-\epsilon$

or
random action $a_t$
with probability $\epsilon$

Play the game for one step

Image at time $t$: $x_t$
$s_t = s_{t-1}, a_{t-1}, x_t$
preprocessed sequence
$\phi_t = \phi(s_t)$
After Training

Play the game for one step

\[ a^* = \arg \max_a Q(s, a) \]
<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minibatch size</td>
<td>32</td>
<td>Number of training cases over which each stochastic gradient descent (SGD) update is computed.</td>
</tr>
<tr>
<td>replay memory size</td>
<td>1000000</td>
<td>SGD updates are sampled from this number of most recent frames.</td>
</tr>
<tr>
<td>agent history length</td>
<td>4</td>
<td>The number of most recent frames experienced by the agent that are given as input to the Q network.</td>
</tr>
<tr>
<td>target network update frequency</td>
<td>10000</td>
<td>The frequency (measured in the number of parameter updates) with which the target network is updated (this corresponds to the parameter $C$ from Algorithm 1).</td>
</tr>
<tr>
<td>discount factor</td>
<td>0.99</td>
<td>Discount factor gamma used in the Q-learning update.</td>
</tr>
<tr>
<td>action repeat</td>
<td>4</td>
<td>Repeat each action selected by the agent this many times. Using a value of 4 results in the agent seeing only every 4th input frame.</td>
</tr>
<tr>
<td>update frequency</td>
<td>4</td>
<td>The number of actions selected by the agent between successive SGD updates. Using a value of 4 results in the agent selecting 4 actions between each pair of successive updates.</td>
</tr>
<tr>
<td>learning rate</td>
<td>0.00025</td>
<td>The learning rate used by RMSProp.</td>
</tr>
<tr>
<td>gradient momentum</td>
<td>0.95</td>
<td>Gradient momentum used by RMSProp.</td>
</tr>
<tr>
<td>squared gradient momentum</td>
<td>0.95</td>
<td>Squared gradient (denominator) momentum used by RMSProp.</td>
</tr>
<tr>
<td>min squared gradient</td>
<td>0.01</td>
<td>Constant added to the squared gradient in the denominator of the RMSProp update.</td>
</tr>
<tr>
<td>initial exploration</td>
<td>1</td>
<td>Initial value of $\epsilon$ in $\epsilon$-greedy exploration.</td>
</tr>
<tr>
<td>final exploration</td>
<td>0.1</td>
<td>Final value of $\epsilon$ in $\epsilon$-greedy exploration.</td>
</tr>
<tr>
<td>final exploration frame</td>
<td>1000000</td>
<td>The number of frames over which the initial value of $\epsilon$ is linearly annealed to its final value.</td>
</tr>
<tr>
<td>replay start size</td>
<td>50000</td>
<td>A uniform random policy is run for this number of frames before learning starts and the resulting experience is used to populate the replay memory.</td>
</tr>
<tr>
<td>no-op max</td>
<td>30</td>
<td>Maximum number of “do nothing” actions to be performed by the agent at the start of an episode.</td>
</tr>
</tbody>
</table>
Thanks!