Deep Reinforcement Learning Part 2
Quintin Fettes

April 10, 2018
Outline

- Notation Review
- Double Deep Q Networks [1]
- Dueling Deep Q Networks [2]
- Noisy Networks for Exploration [3]
Markov Decision Processes: Notation Review

<table>
<thead>
<tr>
<th>States</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actions</td>
<td>$A$</td>
</tr>
<tr>
<td>Rewards</td>
<td>$R(s), r$</td>
</tr>
<tr>
<td>Transition/Dynamics Model</td>
<td>$Pr(s'</td>
</tr>
<tr>
<td>Policy</td>
<td>$\pi(s) = Pr(\cdot</td>
</tr>
</tbody>
</table>

- **Bellman Equation**
  - $v_\pi(s) = \mathbb{E} \left[ G_t | S_t = s \right]$  
  - $v_\pi(s) = \mathbb{E} \left[ \sum_{k=0}^{\infty} \gamma^k R(s_{t+k+1}) | S_t = s \right]$  
  - $v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} Pr(s', r|s, a)(r + \gamma v_\pi(s'))$

- **Bellman Optimality Equations**
  - $V_*(s) = \max_a \sum_{s', r} Pr(s', r|s, a)(r + V_*(s'))$
  - $Q_*(s, a) = \sum_{s', r} Pr(s', r|s, a)(r + \max_{a'} Q_*(s', a'))$
Algorithm 1: Deep Q Learning with Experience Replay [5]

1: Initialize Replay Memory D to capacity N
2: Initialize action-value function $Q$ with random weights $\theta$
3: Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
4: for each episode do
5: Initialize sequence $s_1 = x$ and preprocessed sequence $\phi_1 = \phi(s_1)$
6: for $t=1,T$ do
7: Select $a_t$ using an $\epsilon$–greedy policy on $Q(s, \cdot ; \theta)$
8: Execute action $a_t$ and observe reward $r_t$ and next observation $x_{t+1}$
9: Let $\phi_{t+1} = \phi(s_{t+1})$
10: Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
11: Sample a random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
12: 
   
   \[ y_j = \begin{cases} 
   r_j & \text{if } j+1 \text{ is terminal} \\
   r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a' ; \theta^{-}) & \text{else} 
   \end{cases} \]

13: Perform a gradient descent step on $(y_j - Q(\phi_j, a_j ; \theta))^2$ with respect to the network parameters $\theta$
14: Every $C$ steps set $\theta^{-} = \theta$
15: end for
16: end for
Double Deep Q Learning: The Problem

- Approximate Q-Learning Update Rule
  - $\theta \leftarrow \theta + \alpha [r + \gamma \hat{Q}(s', a', \theta^-) - \hat{Q}(s, a; \theta)] \nabla_\theta \hat{Q}(s, a; \theta)$
  - $a' = \max_{a'} Q(s', a'; \theta^-)$

- Motivations
  - Q Learning involves maximization in the construction of its target policy. In other words, a maximum over the estimated next state-action value is used as an estimate of the maximum current state-action value, which can introduce a significant positive bias, or maximization bias.
  - To see why, consider a state, $s$, whose true values, $q(s, a)$, are all zero but whose estimated values, $Q(s, a)$, are uncertain and distributed some above and some below zero. The maximum of the true values is zero, but the maximum of the estimated values is positive; this is maximization bias.
Double Deep Q Learning: The Solution

Example

- $q(s, a) = 0$  $q(s', a') = 0$  $\hat{Q}(s, a; \theta) = q(s, a)$
- $\max_{a'} \hat{Q}(s', a'; \theta^-) > 0$
- $\therefore \hat{Q}(s, a) = r + \gamma \hat{Q}(s', a'; \theta) > q(s, a) = r + \gamma q(s', a')$

Solution

- Double Q Learning!
- Consider instead learning two independent Q functions, $Q_1(S, A)$ and $Q_2(S, A)$
Algorithm 2: Tabular Double Q Learning [4]

1: Initialize $Q_1(S, A)$ and $Q_2(S, A)$ randomly
2: Initialize $Q_1(S_{terminal}, \cdot)$ and $Q_2(S_{terminal}, \cdot)$ to 0
3: for each episode do
4:     for $t=1, T$ do
5:         Select $a_t$ using an $\epsilon$-greedy policy on $Q(s, \cdot) + Q_2(s, \cdot)$
6:         Execute action $a_t$ and observe reward $r_t$ and next state $s_{t+1}$
7:         sample probability $p \sim N(0, 1)$
8:         if $p < 0.5$ then
9:             $Q_1(s, a) \leftarrow Q_1(s, a) + \alpha (r_t + \gamma Q_2(s', \arg\max_{a'} Q_1(s', a')) - Q_1(s, a))$
10:            else
11:                $Q_2(s, a) \leftarrow Q_2(s, a) + \alpha (r_t + \gamma Q_1(s', \arg\max_{a'} Q_2(s', a')) - Q_2(s, a))$
12:        end if
13:     end for
14: end for
Algorithm 3: Double Deep Q Learning with Experience Replay [1]

1: Initialize Replay Memory $D$ to capacity $N$
2: Initialize action-value function $Q$ with random weights $\theta$
3: Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
4: for each episode do
5:   Initialize sequence $s_1 = x$ and preprocessed sequence $\phi_1 = \phi(s_1)$
6:   for $t=1,T$ do
7:       Select $a_t$ using an $\epsilon$-greedy policy on $Q(s, \cdot; \theta)$
8:       Execute action $a_t$ and observe reward $r_t$ and next observation $x_{t+1}$
9:       Let $\phi_{t+1} = \phi(s_{t+1})$
10:      Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
11:     Sample a random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
12:     $y_j = \begin{cases} r_j & \text{if $j+1$ is terminal} \\ r_j + \gamma \hat{Q}(\phi_{j+1}, \arg\max_{a'} Q(\phi_{j+1}, a', \theta); \theta^-) & \text{else} \end{cases}$
13:     Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
14:     network parameters $\theta$
15:   Every $C$ steps set $\theta^- = \theta$
16: end for
Double Deep Q Learning: Difference between the tabular algorithm and the approximate algorithm

- Rather than declare two networks and two target networks, Double Deep Q Learning uses only the network and target network already present in Deep Q Learning.
  - Is this a problem? While in theory, the target network $\hat{Q}$ is still correlated with the network $Q$, in practice, it still works to alleviate maximization bias.
  - The reasoning was to make the minimum possible change to DQNs and introduce no computational overhead or added memory complexity.

<table>
<thead>
<tr>
<th></th>
<th>DQN</th>
<th>Double DQN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>93.5%</td>
<td>114.7%</td>
</tr>
<tr>
<td>Mean</td>
<td>241.1%</td>
<td>330.3%</td>
</tr>
</tbody>
</table>

Table: [1]Summary of normalized performance up to 5 minutes of play on 49 games. Results for DQN are from Mnih et al. (2015)
Figure: [1] The top and middle rows show value estimates by DQN (orange) and Double DQN (blue) on six Atari games. The results are obtained by running DQN and Double DQN with 6 different random seeds with the hyper-parameters employed by Mnih et al. (2015). The darker line shows the median over seeds and we average the two extreme values to obtain the shaded area (i.e., 10% and 90% quantiles with linear interpolation). The straight horizontal orange (for DQN) and blue (for Double DQN) lines in the top row are computed by running the corresponding agents after learning concluded, and averaging the actual discounted return obtained from each visited state. These straight lines would match the learning curves at the right side of the plots if there is no bias. The middle row shows the value estimates (in log scale) for two games in which DQNs overoptimism is quite extreme. The bottom row shows the detrimental effect of this on the score achieved by the agent as it is evaluated during training: the scores drop when the overestimations begin. Learning with Double DQN is much more stable.
Motivations

- Traditional Deep Q Networks are single stream, requiring the model to learn the value of an action conditioned on the current state for all possible actions.
- By doing this, significant redundancy is introduced into the model.
- The action-value function, \( Q(s, a) \), can be decomposed such that the agent learns the value of a state once then learns the advantage of each action given the state separately.
- Intuition: In some states, knowing which action to take is of high importance. In others, the choice of action can have little impact on the expected return.
Dueling Deep Q Learning: The Solution

Solution
- Rewrite the action value function as $Q(s, a) = V(s) + A(s, a)$
- Introduce an architecture, the dueling architecture, which explicitly separates the representation of the state values, $V(S)$, and the action advantages, $A(s, a)$.
- $Q^\pi(s, a) = V^\pi(s) + A^\pi(s, a)$
  - $V^\pi(s) = \mathbb{E}_{a \sim \pi(s)} [Q^\pi(s, a)]$
  - It follows that: $\mathbb{E}_{a \sim \pi(s)} [A^\pi(s, a)] = 0$
Dueling Architecture

Figure: [2] A popular single stream Q-network (top) and the dueling Q-network (bottom). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action; the green output module implements equation (9) to combine them. Both networks output Q-values for each action.
Dueling Deep Q Learning: Implementation

- We can use exactly the same algorithm as Deep Q Networks with Experience Replay. The only difference is the network architecture.

- Let $\theta$ be parameters shared between the value estimation and advantage estimation, let $\beta$ be parameters unique to the value function estimation, and let $\alpha$ be the parameters unique to the advantage function estimation.

- Option 1: $Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$
  - An issue with this equation is that it’s unidentifiable. That is, given $Q$, we cannot recover $V$ and $A$ uniquely. This leads to poor practical performance.
Dueling Deep Q Learning: Implementation

- Option 2:
  \[ Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left[ A(s, a; \theta, \alpha) - \max_{a' \in A} A(s, a'; \theta, \alpha) \right] \]
  
  - Now, for \( a^* = \arg \max_{a \in A} Q(s, a; \theta, \alpha, \beta) = \arg \max_{a \in A} A(s, a; \theta, \alpha) \), we obtain \( Q(s, a^*; \theta, \alpha, \beta) = V(s; \theta, \beta) \).
  - In other words, \( V(s; \theta, \beta) \) provides an estimate of the value function, while \( A(s, a'; \theta, \alpha) \) provides an estimate of the advantage function.

- Option 3:
  \[ Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left[ A(s, a; \theta, \alpha) - \frac{1}{|A|} \sum_{a' \in A} A(s, a'; \theta, \alpha) \right] \]
  
  - Here, we lose the original semantic of \( V \) and \( A \).
  - On the other hand, the advantages only need to change as fast as the mean; in option 2, the advantages must compensate for any change to the optimal action’s advantage.
  - The original paper’s results were published using option 3.
Dueling Deep Q Learning: Convergence as the number of actions increases

**Figure:** [2] (a) The corridor environment. The star marks the starting state. The redness of a state signifies the reward the agent receives upon arrival. The game terminates upon reaching either reward state. The agents actions are going up, down, left, right and no action. Plots (b), (c) and (d) shows squared error for policy evaluation with 5, 10, and 20 actions on a log-log scale. The dueling network (Duel) consistently outperforms a conventional single-stream network (Single), with the performance gap increasing with the number of actions.
## Dueling Deep Q Learning: Results

<table>
<thead>
<tr>
<th></th>
<th>30 no op</th>
<th>Human Start</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Median</td>
</tr>
<tr>
<td>Prior Duel Clip</td>
<td>591.9%</td>
<td>172.1%</td>
</tr>
<tr>
<td>Prior Single</td>
<td>434.6%</td>
<td>123.7%</td>
</tr>
<tr>
<td>Duel Clip</td>
<td>373.1%</td>
<td>151.5%</td>
</tr>
<tr>
<td>Single Clip</td>
<td>341.3%</td>
<td>132.6%</td>
</tr>
<tr>
<td>Single</td>
<td>307.3%</td>
<td>117.8%</td>
</tr>
<tr>
<td>Nature DQN</td>
<td>227.9%</td>
<td>79.1%</td>
</tr>
</tbody>
</table>

**Table:** [2] Mean and median scores across all 57 Atari games, measured in percentages of human performance.
Noisy Networks for Exploration: The Problem

- **Explore Vs. Exploit**
  - At any time step, there is at least one action whose estimated value is the greatest; We call selecting this action exploiting. If instead one of the nongreedy actions, we say the agent is exploring; this allows the agent to improve its estimate of the value of these nongreedy actions.
  - Exploitation will allow the agent to maximize its return in the short term; however, more exploration may allow the agent to better estimate the true value function and lead to a greater long term return.
  - Exploration vs. Exploitation is a central conflict in Reinforcement Learning

- **$\epsilon$-greedy methods: the most common approach**
  - At every timestep, $t$, the agent selects the action $a_t$ s.t. $a_t = \arg \max_a Q(s, a)$ with probability $1 - \epsilon$
  - With probability $\epsilon$ the agent selects a random action instead
  - With engineered annealing schedules, it can be very effective
  - However, the method seems ill-informed. Can we do better?
Noisy Networks for Exploration: The Solution

- Consider a normal linear layer in a Neural Network:
  - $Y = \theta X + b$

- The noisy parameters, $\theta$, are represented as $\theta \overset{\text{def}}{=} \mu^w + \sigma^w \odot \epsilon^w$

- Thus, we can replace a linear layer with a noisy linear layer such that:
  - $Y \overset{\text{def}}{=} (\mu^w + \sigma^w \odot \epsilon^w)X + (\mu^b + \sigma^b \odot \epsilon^b)$
  - $\mu^w, \sigma^w, \epsilon^w \in \mathbb{R}^{S_l \times S_{l-1}}$
  - $\mu^b, \sigma^b, \epsilon^b \in \mathbb{R}^{S_l \times 1}$
  - $\zeta \overset{\text{def}}{=} \{\mu, \Sigma\}$ is a vector of learnable parameters where $\mu \overset{\text{def}}{=} \{\mu^w, \mu^b\}$ and $\Sigma \overset{\text{def}}{=} \{\sigma^w, \sigma^b\}$
  - $\epsilon^w$ and $\epsilon^b$ are randomly sampled, zero mean noise matrices with fixed statistics
Noisy Networks for Exploration: The Solution

- The usual loss of the neural network is wrapped by expectation over the noise, $\epsilon$
  - $L(\zeta) = \mathbb{E}[L(\theta)]$
  - $L(\zeta) = \mathbb{E} \left[ \mathbb{E}_{s,a,r,s' \sim D}[r + \gamma \max_{a' \in A} Q(s', a', \epsilon', \zeta^-) - Q(s, a, \epsilon, \zeta)]^2 \right]$
  - $L(\zeta) = [r + \gamma \max_{a' \in A} Q(s', a', \epsilon', \zeta^-) - Q(s, a, \epsilon, \zeta)]^2$

- Finally the loss gradient of the loss can be approximated as:
  - $\nabla L(\zeta) = \nabla \mathbb{E}[L(\theta)] = \mathbb{E}[\nabla_{\mu, \Sigma} L(\mu + \Sigma \odot \epsilon)]$
  - $\nabla L(\zeta) \approx \nabla_{\mu, \Sigma} L(\mu + \Sigma \odot \epsilon')$
    - Where we use monte carlo approximates, taking a single sample $\epsilon$ at each step of optimization

- Now, instead of selecting actions according to an $\epsilon$-greedy policy, the agent can act greedily according to a network using noisy linear layers in place of all linear layers.
Figure: [3] Comparison of the learning curves of NoisyNet agent versus the baseline according to the median human normalised score.
## Noisy Networks for Exploration: Results

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>NoisyNet</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Median</td>
<td>Mean</td>
</tr>
<tr>
<td>DQN</td>
<td>319%</td>
<td>83%</td>
<td>379%</td>
</tr>
<tr>
<td>Dueling</td>
<td>524%</td>
<td>132%</td>
<td>633%</td>
</tr>
<tr>
<td>A3C</td>
<td>293%</td>
<td>80%</td>
<td>347%</td>
</tr>
</tbody>
</table>

**Table:** [3] Comparison between the baseline DQN, Dueling and A3C and their NoisyNet version in terms of median and mean human-normalised scores. We report on the last column the percentage improvement on the baseline in terms of median human-normalised score.
Noisy Networks for Exploration: Why is this better?

- Custom Exploration strategies vs. fixed exploration strategies in $\epsilon$-greedy methods
- Automatic annealing schedule
- Deep Exploration: more weight can be added to the noise upon reaching new regions of the state space
Questions
References


