Double Q-Learning

SOTA2 - Group C

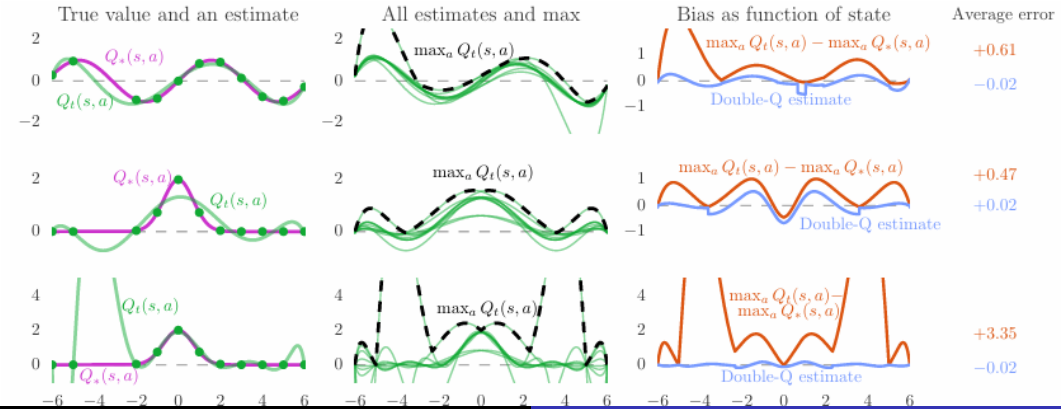
Anton Kesy, Étienne Muser, Katharina Schindler, Lukas Fehrenbacher, Nico Ruschmann

Offenburg University of Applied Sciences

WS 2024/2025

Problem with Q-Learning

- Verwendung desselben Netzwerks für Policy estimation und Action Selection führt zu Überschätzung des Wertes einer Aktion
- Obere Schranke der Überschätzung ist abhängig von der Anzahl der Aktionen



Double Q-Learning: Hauptfunktion

- **Ziel**: Reduktion der Überschätzung von Q-Werten, die bei standardmäßigem Q-Learning auftreten.
- **Ansatz**:
  1. **Trennung von Aktionsauswahl und Bewertung**:
     - Primäres Netzwerk (θ): Auswahl der Aktion $arg\max$).
     - Sekundäres Netzwerk $theta'$): Bewertung des Werts der ausgewählten Aktion.
  2. **Aktualisierung des Zielwerts (Target)**:
     $Y_t^{\text{DoubleQ}} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta), \theta')$
- **Effekte**:
  - Verringerung von Verzerrungen durch Überschätzung.
  - Stabileres und genaueres Lernen.
  - Besonders nützlich bei verrauschten oder ungenauen Wertschätzungen, z. B. in Deep Reinforcement Learning.
- **Kernidee**: Separate Netzwerke zur Entkopplung von Aktionsermittlung und -bewertung.

Double DQN (DDQN)

- **Ziel**: Reduzierung der Überschätzung von Aktionswerten im Vergleich zu DQN
- **Ansatz**
  1. **Aktionsauswahl**: Das Online-Netzwerk wählt die Aktion mit dem höchsten Q-Wert im nächsten Zustand aus, um die beste Aktion zu bestimmen.
  2. **Aktionsbewertung**: Das Target-Netzwerk bewertet den Wert der zuvor ausgewählten besten Aktion im nächsten Zustand.
  3. **Zielwertberechnung**: Der Zielwert wird durch Addition des sofortigen Rewards und des vom Target-Netzwerk bewerteten Werts der nächsten Aktion berechnet.
  4. **Online-Netzwerk Update**: Die Gewichte des Online-Netzwerks werden angepasst, um die Differenz zwischen der aktuellen Schätzung und dem Zielwert zu reduzieren
  5. **Periodisches Target-Netzwerk Update**: Die Gewichte des Target-Netzwerks werden periodisch mit den Gewichten des Online-Netzwerks aktualisiert

Zusammenfassend verwendet DDQN das Online-Netzwerk für die Aktionsauswahl und das Target-Netzwerk zur Aktionsbewertung, um die Überschätzung von Aktionswerten zu reduzieren, was zu einem stabileren Lernprozess führt

Explain DDQ Implementation (1/2)

1. **Two Networks**:
   - **Q-online**: Predicts best actions
   - **Q-target**: Stabilizes training by estimating future rewards (frozen during training)
2. **Experience Replay**:
   - stores and samples past experiences to break correlation
3. **TD Estimate**:
   - (Q_{online}(s, a)): Current Q-value for the chosen action
4. **TD Target**:
   - (TD$\{target\} = r +$ $Q\{$target$\}$(s', a')): Combines reward and future Q-value
5. **Loss and Update**:
   - Loss = (| TD$\{estimate\}$ - $TD\{$target$\}$ |)
   - Update Q-online with backpropagation

Explain DDQ Implementation (1/2)

6. **Q-target Sync**:
   - Periodically copy weights from Q-online to Q-target
7. **Learning**:
   - Train Q-online every few steps, sync Q-target periodically
8. **Save Progress**:
   - Save the network's state regularly

Code Example

- mario_rl_tutorial - Learn
- share feature generator across Q-Online and Q-Target

```
def __build_cnn(self, c, output_dim):
    return nn.Sequential(
        nn.Conv2d(in_channels=c, out_channels=32, kernel_size=8, stride=4),
        nn.ReLU(),
        nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2),
        nn.ReLU(),
        nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1),
        nn.ReLU(),
        nn.Flatten(),
        nn.Linear(3136, 512),
        nn.ReLU(),
        nn.Linear(512, output_dim),
    )
```

TD Estimate & TD Target

```python
def td_estimate(self, state, action):
    current_Q = self.net(state, model="online")[
        np.arange(0, self.batch_size), action
    ]  # Q_online(s,a)
    return current_Q

def td_target(self, reward, next_state, done):
    next_state_Q = self.net(next_state, model="online")
    best_action = torch.argmax(next_state_Q, axis=1)
    next_Q = self.net(next_state, model="target")[
        np.arange(0, self.batch_size), best_action
    ]
    return (reward + (1 - done.float()) * self.gamma * next_Q).float()
```

Demo

- mario_rl_tutorial.html
- working fork