CS-566 Deep Reinforcement Learning

Stable Deep Value-Based Learning

Nazar Khan
Department of Computer Science
University of the Punjab

Stable Deep Value-Based Learning

From Instability to Convergence

- Early concerns about convergence discouraged research in **deep reinforcement learning (DRL)** for years.
- Researchers focused on **Linear function approximators** — more stable, with convergence guarantees.
- Yet, work on **convergent deep RL** continued:
  - Neural fitted Q-learning[1]
  - Actor-critic variants[2,3]
  - Early deep TD learning[4,5]

---

[1] Riedmiller, 'Neural fitted Q iteration—first experiences with a data efficient neural reinforcement learning method'.

[2] Bhatnagar et al., 'Convergent temporal-difference learning with arbitrary smooth function approximation'.

[3] Maei et al., 'Toward off-policy learning control with function approximation'.

[4] Sallans and Hinton, 'Reinforcement learning with factored states and actions'.

[5] Heess, Silver, and Teh, 'Actor-critic reinforcement learning with energy-based policies'.

The Breakthrough: Deep Q-Networks (DQN)

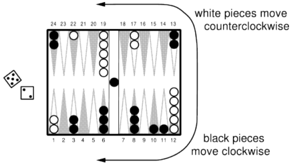
Reviving Deep RL

- **Mnih et al.**[6] showed that:
  - Stable and convergent training **is possible** with deep networks.
  - Even on complex domains (e.g., **Atari 2600**).
- Triggered renewed exploration into:
  - Conditions enabling convergence
  - Techniques to overcome the **deadly triad**:
    1. Function approximation
    2. Bootstrapping
    3. Off-policy learning
- Led to **stability-enhancing methods** (e.g., replay buffers, target networks).

[6] Volodymyr Mnih et al. 'Human-level control through deep reinforcement learning'. In: *Nature* 518.7540 (2015), pp. 529–533.

Early Signs of Stable Learning
The Case of TD-Gammon

- In the late 1980s–1990s, **Gerald Tesauro** developed:
  - **Neurogammon**: Supervised learning from expert Backgammon games[7].
  - **TD-Gammon**: Reinforcement learning from **self-play** using temporal-difference updates[8].
- Achieved stable learning with:
  - A **shallow network** (1 hidden layer)
  - Raw board input + heuristic features
  - TD-style value updates (like Q-learning)



[7] Tesauro, 'Neurogammon wins Computer Olympiad'.
[8] Tesauro, 'Temporal difference learning and TD-Gammon'.

Beyond Backgammon
Limits of Early Success

- Attempts to reproduce TD-Gammon's success:
  - Checkers[9]
  - Go[10,11]
- These efforts largely **failed** to achieve stable learning.
- Hypothesis: **Backgammon's randomness** (dice rolls) may have:
  - Improved exploration
  - Smoothed the value landscape
- For years, it was believed that Backgammon was a **special case**.

[9] Kumar Chellapilla and David B Fogel. 'Evolving neural networks to play checkers without relying on expert knowledge'. In: *IEEE Transactions on Neural Networks* 10.6 (1999), pp. 1382–1391.

[10] Ilya Sutskever and Vinod Nair. 'Mimicking Go experts with convolutional neural networks'. In: *International Conf. on Artificial Neural Networks*. Springer. 2008, pp. 101–110.

[11] Christopher Clark and Amos Storkey. 'Teaching deep convolutional neural networks to play Go. arXiv preprint'. In: *arXiv preprint arXiv:1412.3409* 1 (2014).

Deep RL Matures

Towards Stable and Generalizable Learning

- Later work confirmed that stability is achievable in deep RL:
  - **Atari**: DQN and successors[12]
  - **Go**: AlphaGo and AlphaZero[13]
  - **Continuous control**: Deep actor-critic methods[14]
- **Stable training and generalization** are possible with:
  - Target networks
  - Experience replay
  - Regularization and diversity methods
- Ongoing research aims to further understand and enhance:
  - Convergence properties
  - Diversity and representation learning

[12] Volodymyr Mnih et al. 'Human-level control through deep reinforcement learning'. In: *Nature* 518.7540 (2015), pp. 529–533.

[13] David Silver et al. 'Mastering the game of Go without human knowledge'. In: *Nature* 550.7676 (2017), p. 354.

[14] Nicolas Heess, David Silver, and Yee Whye Teh. 'Actor-critic reinforcement learning with energy-based policies'. In: *European Workshop on Reinforcement Learning*. 2013, pp. 45–58.

Deep Q-Networks

- The DQN algorithm[15] achieves **stable and convergent** training on complex domains using
  - experience replay, and
  - infrequent weight updates.

Focus of DQN

The original focus of DQN is on two things.

1. breaking correlations between subsequent states, and
2. slowing down changes to parameters to improve stability.

[15] Volodymyr Mnih et al. 'Human-level control through deep reinforcement learning'. In: *Nature* 518.7540 (2015), pp. 529–533.

Why are correlated states bad?

- Sequential agent-environment interactions create **highly correlated** training samples
- The network might be trained on too many samples of a certain kind or in a certain area.
- Other parts of the state space will remain under-explored.

We can reduce correlation – and the local minima they cause – by adding a small amount of supervised learning.

Experience Replay[19]

- To break correlations and to create a more diverse set of training examples, DQN uses *experience replay*.
- Introduces a *replay buffer*[16] – a cache of previously explored states.
- Randomly samples training states from the replay buffer.
- Biologically inspired[17].
- **Stores the last N examples in the replay memory, and samples uniformly when performing updates.**
- A typical[18] number for N is 10^6.

[16] Long-Ji Lin. 'Self-improving reactive agents based on reinforcement learning, planning and teaching'. In: *Machine Learning* 8.3-4 (1992), pp. 293–321.

[17] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. 'Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory.' In: *Psychological Review* 102.3 (1995), p. 419.

[18] Shangtong Zhang and Richard S Sutton. 'A deeper look at experience replay'. In: *arXiv preprint arXiv:1712.01275* (2017).

[19] Volodymyr Mnih et al. 'Playing Atari with deep reinforcement learning'. In: *arXiv preprint arXiv:1312.5602* (2013).

Experience Replay

- Training becomes more dynamic and diverse compared to learning from the most recent state.
- Increases independence of subsequent training examples since next state to be trained on is no longer a direct successor of the current state.
- **More coverage** since it spreads out the learning over more previously seen states.
- **Less correlation** since it samples randomly from previous experiences.

Experience replay $\Longrightarrow$ off-policy learning

Experience replay is a form of off-policy learning, since the target parameters are different from those used to generate the sample.

Infrequent Weight Updates[20]

- **After every n updates, the network Q_θ is cloned to obtain a *target network* Q_{θ^-}, which is used for generating the targets for the following n updates to Q_θ.**
- Weights θ^- of the target network change n-times slower than weights θ of the behavior policy.

Benefit of stable targets

Stable Q-targets lead to

1. reduced divergence,
2. reduced oscillations, and
3. more stable parameters θ.

[20] Volodymyr Mnih et al. 'Human-level control through deep reinforcement learning'. In: *Nature* 518.7540 (2015), pp. 529–533.

DQN Pseudocode

```
def dqn:
    initialize replay_buffer empty
    initialize Q network with random weights
    initialize Qt target network with random weights
    set s = s0
    while not convergence:
        # DQN in Atari uses preprocessing; not shown
        epsilon-greedy select action a in argmax(Q(s,a)) #
            action selection depends on Q (moving target)
        sx,reward = execute action in environment
        append (s,a,r,sx) to buffer
        sample minibatch from buffer # break temporal
            correlation
        take target batch R (when terminal) or Qt
        do gradient descent step on Q # loss function uses
            target Qt network
```

Stable Baselines

- The environment is only half of the RL experiment.
- We also need an agent algorithm to learn the policy.
- OpenAI provided implementations of many RL algorithms, called **Baselines**[21].
- **Stable Baselines**[22] contains improved implementations of OpenAI's algorithms with more documentation[23] and other features.
- Almost all algorithms in this course can be found in Stable Baselines.

```
pip install stable-baselines
```

[21] https://github.com/openai/baselines
[22] https://github.com/hill-a/stable-baselines
[23] https://stable-baselines.readthedocs.io/en/master/

The Breakout Game

Learning Breakout using a DQN Agent using Stable Baselines

```
from stable_baselines.common.atari_wrappers import make_atari
from stable_baselines.deepq.policies import MlpPolicy,
   CnnPolicy
from stable_baselines import DQN

env = make_atari('BreakoutNoFrameskip-v4')

model = DQN(CnnPolicy, env, verbose=1)
model.learn(total_timesteps=25000)

obs = env.reset()
while True:
    action, _states = model.predict(obs)
    obs, rewards, dones, info = env.step(action)
    env.render()
```

DQN Extensions

- DQN results spawned many refinements.
- Some in setting targets.
- Some in selecting training samples.
- Some in network architecture.
- Some in handling stochasticity.

Double Deep Q-Learning (DDQN)

- Deep Q-Learning may overestimate action values due to the max operation.
- DQN target is computed as

$$y^{\text{DQN}} = r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta^-)$$

- We can rewrite it as

$$y^{\text{DQN}} = r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta^-); \theta^-)$$

- Same set of weights θ^- is used twice for action *selection* and *evaluation*.

Double Deep Q-Learning (DDQN)

- Double Deep Q-Learning[24]
  - uses the Q-Network θ to choose the action, but
  - uses the separate target Q-Network θ^- to evaluate the action.
- Double DQN target is computed as

$$y^{\text{DDQN}} = r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta); \theta^-)$$

- Reduces overestimation caused by the max operator.
- Action that maximizes $Q(s_{t+1}, a; \theta)$ might not maximize $Q(s_{t+1}, a; \theta^-)$.

On 49 Atari games, DDQN achieved

- about twice the average score of DQN with the same hyperparameters, and
- four times the average DQN score with tuned hyperparameters.

[24] Hado Van Hasselt, Arthur Guez, and David Silver. 'Deep Reinforcement Learning with Double Q-Learning'. In: *AAAI*. vol. 2. Phoenix, AZ. 2016, p. 5.

Prioritized Experience Replay (PEX)

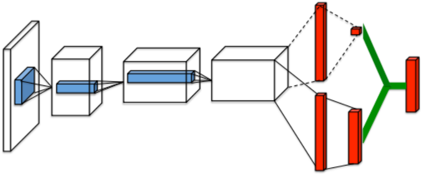
- Instead of sampling uniformly from the replay buffer, use prioritized[25] sampling.
- Probability of picking i-th sample from the buffer is

$$p_i = \frac{|e_i|^a}{\sum_{j=1}^{|B|} |e_j|^a}$$

  where e_i is the TD-error for experience i and $|B|$ is the size of the replay buffer.
- *Sample with higher TD-error will have higher probability of being replayed.*
- Parameter a controls the amount of prioritization ($a = 0 \implies$ uniform sampling).

[25] Tom Schaul et al. 'Prioritized experience replay'. In: *International Conference on Learning Representations*. 2016.

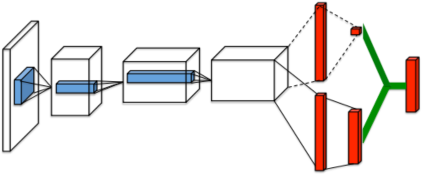
Dueling Double Deep Q-Learning[26] (DDDQN)



- Instead of directly estimating $Q(s, a)$, network outputs 2 things:
  1. State Value: $V(s)$ – how good it is to be in state s.
  2. Advantage: $A(s, a)$ – how much better action a is compared to others in state s.

[26] Ziyu Wang et al. 'Dueling network architectures for deep reinforcement learning'. In: *International Conference on Machine Learning*. 2016, pp. 1995–2003.

Dueling Double Deep Q-Learning (DDDQN)



- The Q-value is then combined as

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha) \right)$$

  where θ denotes parameters of the *shared backbone*, α denotes parameters of the *advantage stream*, and β denotes parameters of the *value stream*.

Intuition

Faster learning of $V(s)$ when actions don't matter.
The two streams *duel* to produce the final Q-values.

Dueling Double Deep Q-Learning (DDDQN)

- Combines tricks from DQN and DDQN.
- *Experience replay* and *target networks* from standard DQN.
- *Double DQN target and update rule* to reduce overestimation.
- Training target:

$$y^{\text{DDDQN}} = y^{\text{DDQN}}$$

  and the network estimates $Q(s, a)$ via $V(s)$ and $A(s, a)$.
- More stable, faster, and robust learning.

Distributional Deep Q-Learning
Motivation

- Standard DQN estimates **expected return:**

$$Q(s, a) = \mathbb{E}[G_t \mid s_t = s, a_t = a]$$

- This collapses uncertainty by ignoring the variability of outcomes.

Example

- Two actions might have the same expected reward $Q(s, a_i) = Q(s, a_j) = 5$.
- But one is always 5, while the other alternates between 0 and 10.
- DQN treats both actions as identical, which can limit performance.

Key Idea

Model the **entire distribution** of returns, not just its mean.

Distributional Reinforcement Learning
Core Idea

- Predict distribution $Z(s, a)$ over returns[27].

$$Q(s, a) = \mathbb{E}[Z(s, a)]$$

- Each action's value is a random variable.
- The Bellman operator now acts on distributions:

$$\mathcal{T} Z(s, a) = R(s, a) + \gamma Z(s', a')$$

- $a' = \arg\max_{a'} \mathbb{E}[Z(s', a')]$

[27] Marc G Bellemare, Will Dabney, and Rémi Munos. 'A distributional perspective on reinforcement learning'. In: *International Conference on Machine Learning*. 2017, pp. 449–458.

C51 Algorithm Overview
Bellemare et al. (2017)

1. **Discretize return space:**

$$z_i = v_{\min} + i\frac{v_{\max} - v_{\min}}{50}, \quad i = 0, \ldots, 50$$

2. **Network output:** 51 categorical probabilities ($p_i = p(z_i)$)

$$p(s, a) = [p_0, p_1, \ldots, p_{50}]$$

3. **Bellman target:**

$$TZ(s, a) = r + \gamma Z(s', a^*)$$

4. **Projection:** project target distribution back to fixed supports.
5. **Loss:** minimize KL-divergence

$$\mathcal{L} = D_{KL}\big(p(\cdot|s, a) \parallel \Pi TZ(\cdot|s, a)\big)$$

C51: Intuition and Summary

- Learns a **histogram** of possible future returns.
- Captures uncertainty and risk.
- Each Q-value is computed as

$$Q(s, a) = \sum_i z_i p_i(s, a)$$

- Provides richer learning signals $\Rightarrow$ faster convergence.

Benefits

- Models uncertainty and multi-modal outcomes
- Stabilizes training

Summary Table
DQN vs. Distributional DQN

Aspect	DQN	Distributional DQN (C51)
Output type	Scalar $Q(s, a)$	Probability distribution $p_i(s, a)$
Target	Expected return	Distribution of returns
Learning signal	Mean-squared TD error	KL divergence between distributions
Stability	Moderate	Higher (richer gradients)
Captures risk?	No	Yes

Noisy DQN

- Another distributional method is noisy DQN[28].
- Noisy DQN makes network layers stochastic by adding noise to the weights.
- Noise is controlled by learnable parameters.
- Noise induces randomness in the agent's policy, which increases exploration.

[28] Meire Fortunato et al. 'Noisy networks for exploration'. In: *International Conference on Learning Representations*. 2018.

Summary

- Deep Q-Learning suffers from instability and divergence due to the moving-targets problem.
- Correlated states introduce further inefficiency.
- The DQN paper used
  - frozen target networks to reduce the moving-targets issue, and
  - replay buffers to break temporal correlations.
- Spawned numerous extensions to achieve greater stability, speed, and robustness.
  - Double DQN to reduce overestimation from max operator.
  - PEX to learn more from samples with high TD error.
  - Dueling Double DQN for faster learning of $V(s)$ when actions don't matter.
  - Distributional DQN for exploiting uncertainty of returns.