Deep reinforcement learning

Approximate Q-learning to Deep RL

- Feature based approximation requires us to engineer features
  - This is not the way we are doing AI in the mid 2020s!
  - We want to *learn* the features
- So we just replace the feature based approximation with a neural network, and we are done.
- ...
- It is not that simple!

Some challenges and our first solutions

- **Size** of the Q-table
  - our solution was **Q-function approximation**
- Slow convergence due to **sparse rewards**
  - our solution was **bootstrapping**, like in TD-learning
  - we update the V/Q values not directly in function of rewards but from previous V/Q values (rather than waiting for final outcomes)
- The challenge of getting data without already having the policy
  - our solution was **off-policy learning**
- Unfortunately, the solutions create a new problem

The deadly triad

- Richard Sutton and Andrew Barto in "*Reinforcement Learning: An Introduction.*" introduced three components, which, when combined, can lead to **instability** and **divergence** during learning.

1. **Function Approximation**: to estimate value functions or policies.
    - It introduces approximation errors that propagate through the learning process

The deadly triad (cont'd)

2. **Bootstrapping**:
    - Any error in the estimated values is propagated forward in the learning process. Inaccurate bootstrapped estimates can create a feedback loop of errors, especially when combined with function approximation.
3. **Off-Policy Learning**: Learning about one policy (the target policy) while following a different policy (the behavior policy), as seen in algorithms like Q-learning.
    - However, off-policy learning is sensitive to discrepancies between the behavior and target policies, and when combined with function approximation and bootstrapping, it becomes harder to converge to stable solutions.

Why the Deadly Triad Causes Instability

When these three components are combined, they create a situation where errors can accumulate and amplify, often leading to unstable or divergent learning:

- **Error Propagation**: Errors in function approximation get propagated through bootstrapping, causing inaccurate estimates to further distort learning.
- **Divergence**: In off-policy learning, the difference between the behavior and target policy introduces additional **variance**. When this variance combines with approximation errors and bootstrapped updates, it can push the value estimates away from convergence.
- **Overestimation Bias**: In Q-learning, errors in the approximation often lead to overestimation of Q-values. This bias, especially in complex environments with function approximation, can compound over time.

Solutions and Mitigations

Several techniques have been developed to mitigate the effects of the deadly triad:

1. **Experience Replay**: Stores past experiences and samples them randomly to break the correlation between sequential data, stabilizing learning in off-policy methods.
2. **Target Networks**: In DQN, a separate target network is used to provide stable target values during learning, reducing oscillations caused by bootstrapping with function approximation.
3. **Double Q-learning**: Uses two Q-value estimates to reduce overestimation bias and mitigate issues caused by bootstrapping.
4. **Actor-Critic Methods**: In actor-critic approaches, the actor (policy) and critic (value function) are separate, reducing the need for off-policy bootstrapping in some cases.

Q-learning with deep neural network function approx.

- Represent the Q value by a Q-network with weights $\mathbf{w}$

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$

- Optimal Q-values should verify the Bellman equation:

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

- Treat the right hand as a target, and try to minimize the mean squared error loss

$$I(\mathbf{w}) = \left(r + \gamma \max_{a} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w})\right)^2$$

Q-learning with deep neural network function approx.

- Ok, so minimize the MSE loss $I(\mathbf{w})$ and we are done!
- It converges to Q^* when using table lookup representation
- But diverges when using neural networks due to
  - Correlations between samples
  - Non-stationary targets

DQN

- DQN is an algorithm that solves both problems!
- Famous paper shows superhuman performance on a number of Atari games

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533. DOI:10.1038/nature14236 (32000+ citations)

- https://www.youtube.com/watch?v=V1eYniJ0Rnk

DQN experience replay

- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$

$$\mathcal{D} = \{(s_1, a_1, r_2, s_2), (s_2, a_2, r_3, s_3), \ldots (s_t, a_t, r_{t+1}, s_{t+1})\}$$

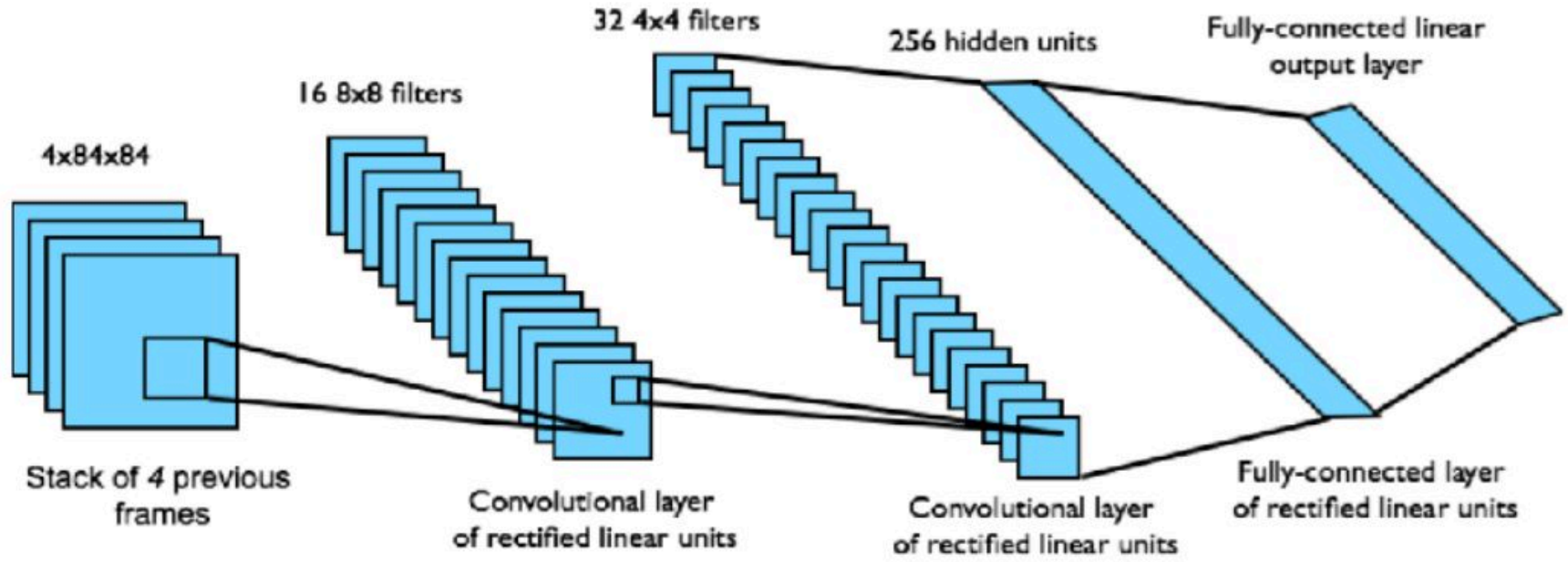
- Sample a **random mini-batch** of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets with respect to old, fixed parameters $\mathbf{w}^-$
- Optimize the mean squared error between the Q-network and Q-learning targets

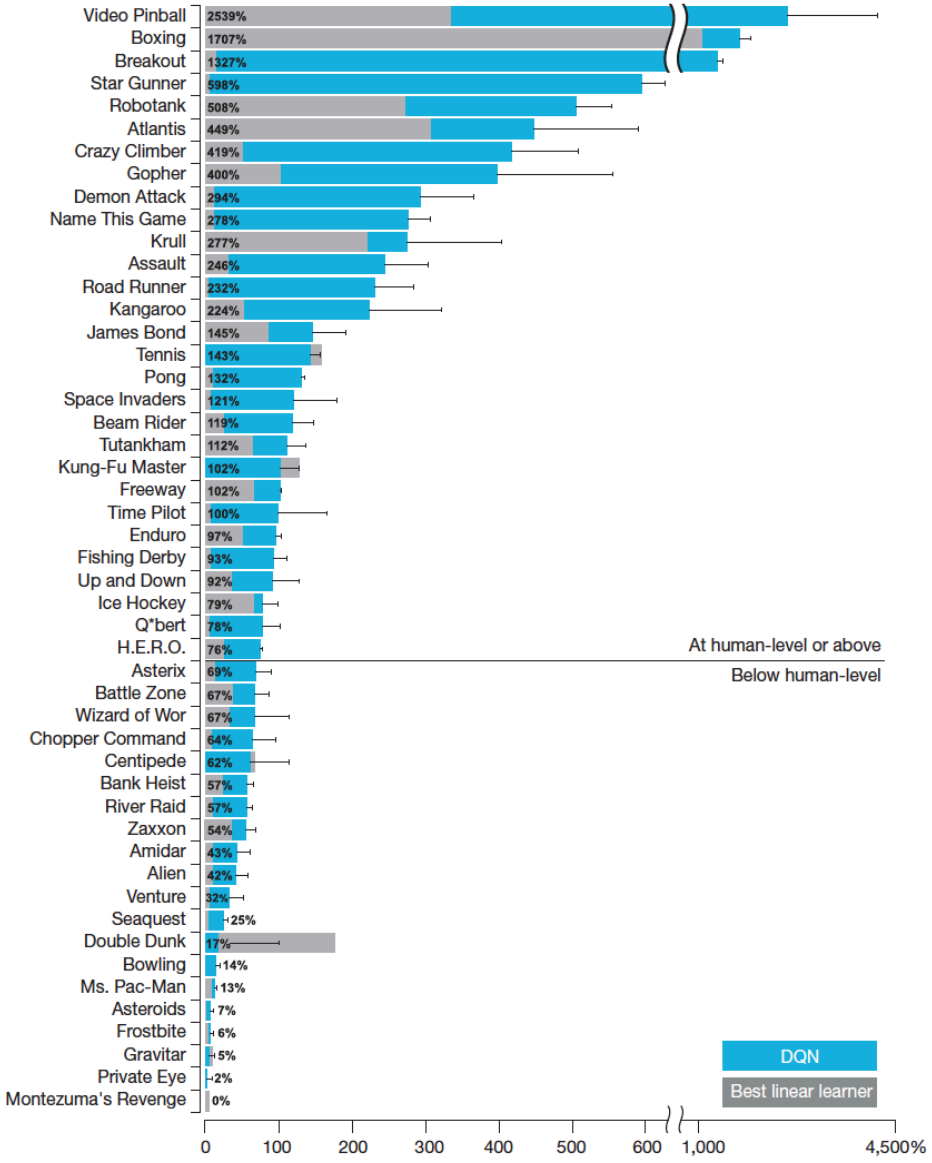
$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2 \right]$$

- Use stochastic gradient descent

How was this applied for Atari

- Input observation is stack of raw pixels from the last 4 frames
- Output is $Q(s, a)$ for 18 joystick / button positions
- Reward is change of score for that step





Thoughts about the performance

- Superhuman performance on many of the games.
- Number of updates is usually in the range of 10s of millions
  - Much less efficient learning than humans!
- And what is going on with Montezuma's revenge?

Montezuma's revenge

- You need to get the key before you open the door!

Challenges in Montezuma's revenge

- Challenging exploration:
  - Epsilon greedy or random exploration is unhelpful: it is unlikely that you will pick up the key and then go to the door through random movement
- Long distance between action and reward
  - A key picked up might only be used several rooms later
- Costly negative rewards
  - Falling or touching an enemy lead to loss of life and starting over.

Deep RL developments

- Many new algorithms had been developed to try to improve on DQN
  - Double Q-learning for fighting maximization bias
  - Prioritized experience replay
  - Dueling Q-networks
  - Stochastic nets for exploration instead of ϵ-greedy

Q-maximization bias

- In general, Q-learning tends to overestimate the Q-values, even if ultimately converges to the right value
- Consider a state for which the correct Q value for all actions is Q(s,a)=0
- Our estimates are uncertain, some are negative and some are positive
- In standard Q-learning, the update rule is as follows:

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

- The use of $\max_{a'} Q(s',a')$ introduces a bias because it's more likely to pick a Q-value that has been overestimated. This overestimation can accumulate over time, leading to suboptimal policies.

Double Q-learning

- Reduces the overestimation bias
- Maintaining **two separate Q-value estimates**, referred to as Q_A and Q_B.
- The algorithm alternates between updating Q_A and Q_B as follows:
- When updating Q_A:
  - The action a' that maximizes Q_A in the next state s' is chosen: $a' = \arg\max_a Q_A(s', a)$.
  - However, the target value is calculated using Q_B instead of Q_A:

$$Q_A(s, a) \leftarrow Q_A(s, a) + \alpha \left(r + \gamma Q_B(s', a') - Q_A(s, a) \right)$$

- When updating Q_B, do the reverse.

Benefits of Double Q-learning

- **Reduced Overestimation Bias**: By decoupling the action selection and evaluation processes, Double Q-learning leads to more accurate Q-value estimates.
- **Improved Stability and Convergence**: Reducing overestimation makes learning more stable, especially in environments with noisy rewards or complex state-action spaces.
- **Better Performance in Practice**: Double Q-learning has been shown to perform better than traditional Q-learning in many settings, especially when combined with function approximators (like neural networks) in environments with high variance.

Double Q-learning in Deep Reinforcement Learning (Double DQN)

In deep reinforcement learning, Double Q-learning is applied in **Double Deep Q-Networks (Double DQN)**. Here, two neural networks are maintained:

1. A **main Q-network** for action selection.
2. A **target Q-network** for action evaluation.

Double DQN addresses overestimation bias in the same way as Double Q-learning but adapts it for use with deep neural networks, significantly improving stability and performance.

Deep RL summary

- There are more things to discuss that we cannot cover in this class.
- Policy gradients (eg. Proximal Policy Optimization PPO)
  - Used in reinforcement learning from human feedback, one of the algorithms used in LLM alignment
- Actor-critic algorithms (eg. Soft Actor Critic SAC)
  - Used for continuous control tasks in robotics. Significant successes in simulation.
- Many research challenges remain.