Batch methods. DQN.

Batch Reinforcement Learning

- Gradient methods are appealing:
  - Move a little bit in the downhill direction.
- Not sample-efficient
  - We update our function approximation in the direction of the experience, and then throw the experience away.

Batch Reinforcement Learning

- **Idea**: Find the best fitting value function given the agent's experience.
- The **batch** is the full agent's experience.

Experience

- An **experience** or **replay memory** $\mathcal{D}$ is a collection of tuples (s, a, r).
- The agent chooses randomly a minibatch from $\mathcal{D}$, to replay his experience and update θ.
    - Why? Break correlations from the data.

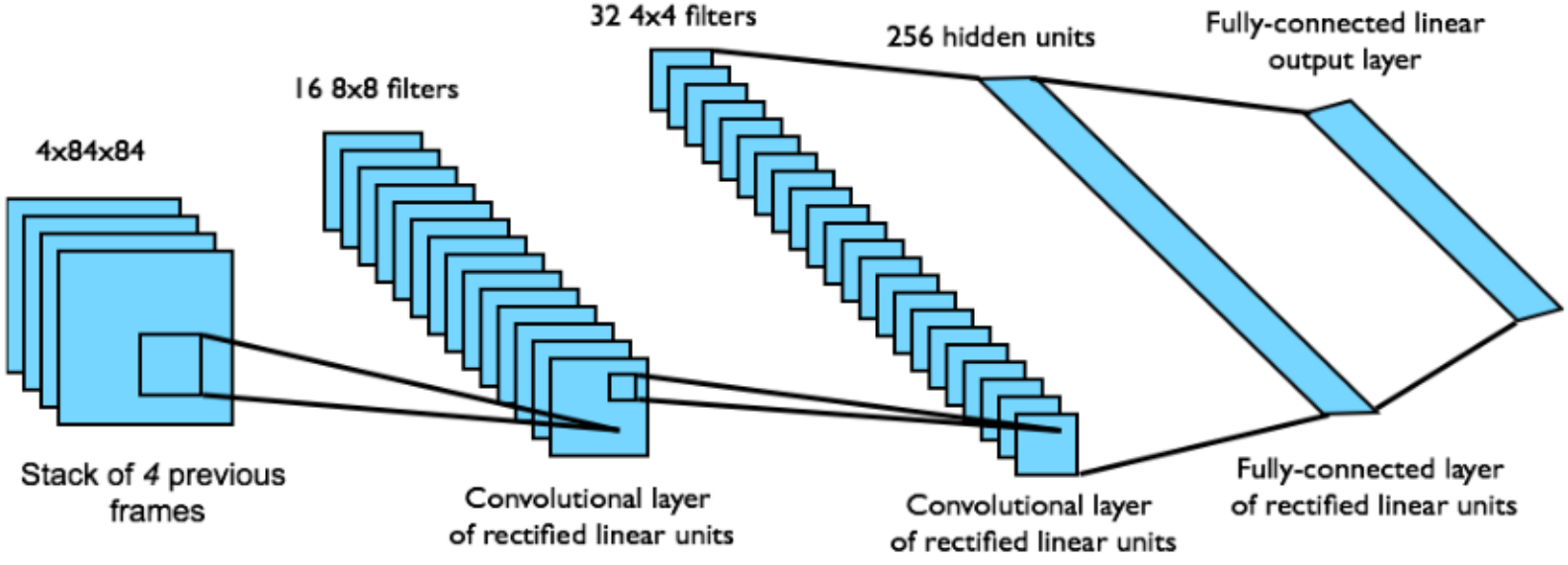
Deep Q-Networks

Experience Replay in Deep Q-Networks (DQN)

- Take action a according to ϵ-greedy policy.
- Store transition s, a, r, s' in replay memory $\mathcal{D}$
- Choose a random sample from $\mathcal{D}$ (minibatch).
- Compute Q-learning target with old, fixed parameters w^-.
- Choose the new parameter w that minimizes the error

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

DQN in Atari games



- Input: stack of raw pixels from last 4 frames.
- Reward is change of score for the step.
- Output: 18 joystick/button positions.
- Training time: 2 weeks on GPU to reach human-level performance.

The secret sauce

- Two tricks that make DQN work (recall the non-convergence discussion from last lecture):
  - Experience Replay
    - Break correlations with the data.
  - Fixed Q-target
    - For a while, we are improving on the direction of the frozen parameter.

Improvements since the original DQN

Double DQN

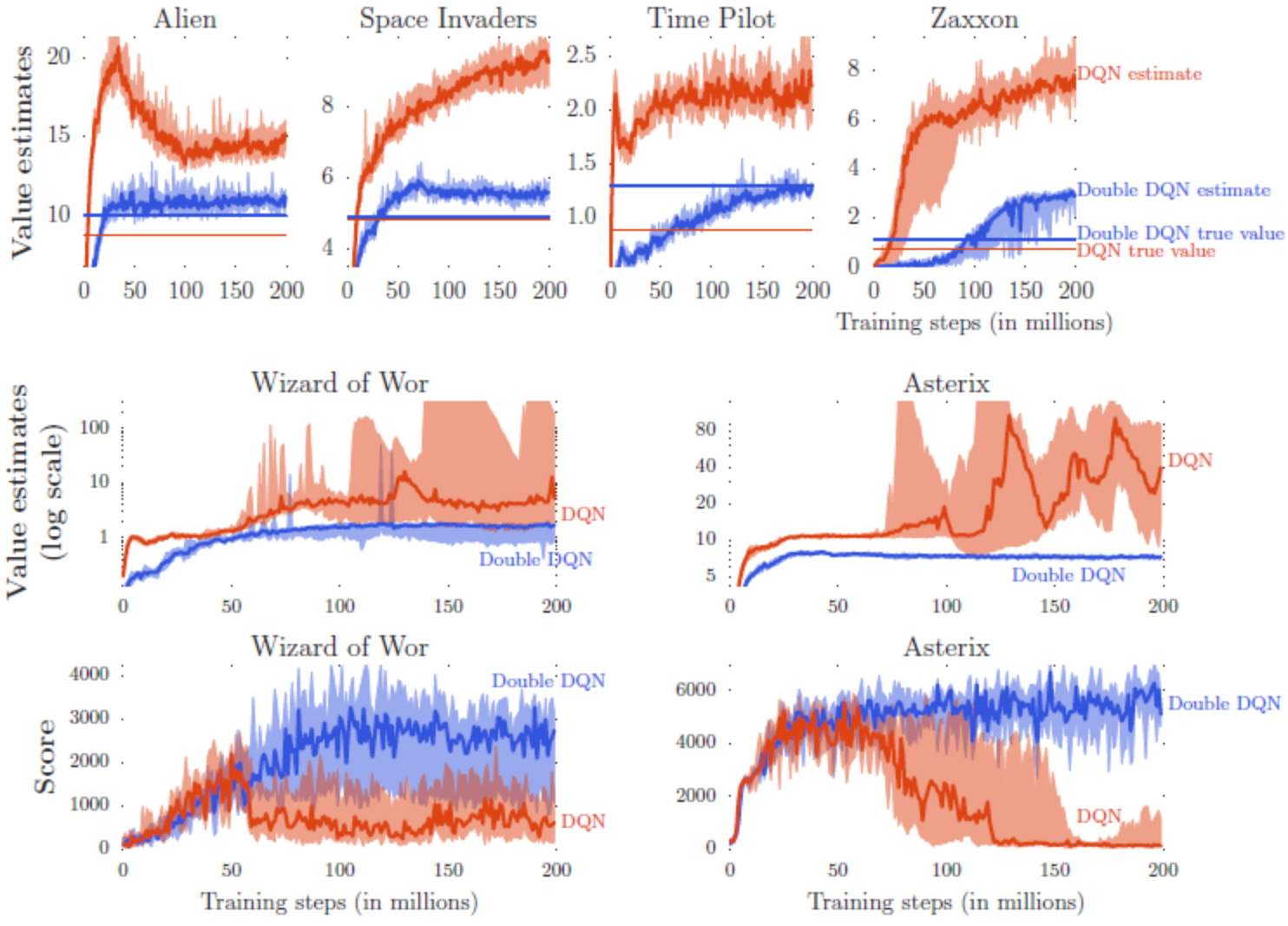
- **Issue**: Overestimation of the actions
- It remains an open problem whether overestimation of the actions is an issue.
    - What can go wrong? Be too optimistic about bad actions.
- Not a "deep learning" issue, the same happens in tabular methods (NIPS 2010).

Double DQN

- Current network is used to select actions.
- Older network is used to evaluate actions.
- Error to minimize is:

$$\sum_{s,a,r,s'} \left(r + \gamma Q(s', \text{argmax}_{a'} Q(s', a', w), w^{-}) - Q(s, a, w)\right)^2$$

Double DQN: Did it help?



Prioritised replay

- State transitions can be more or less surprising, irrelevant or even not relevant for the current agent level.
- Replay transitions with high expected learning progress

- Store the experience in a priority queue, depending to the DQN error

$$|r + \gamma \max_{a'}(s', a', w^-) - Q(s, a, w)|$$

- Some noise in the selection needed to reduce bias and loss of diversity.
- Similar results as in the DQN paper, but faster.

Duelling network

- Split the Q-network into two channels

$$Q(s, a) = V(s, u) + A(s, a, w).$$

- More efficient learning, because the updates of the value function V do not depend on the action.

Duelling network

Architecture

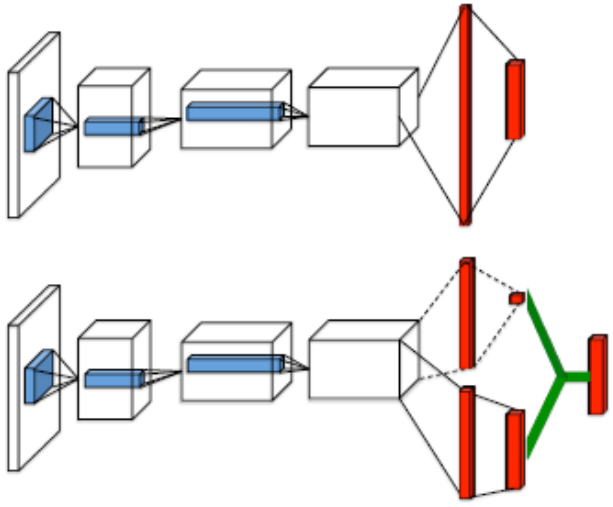


Figure 1. A popular single stream Q-network (**top**) and the dueling Q-network (**bottom**). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action; the green output module implements equation (9) to combine them. Both networks output Q-values for each action.

Demo

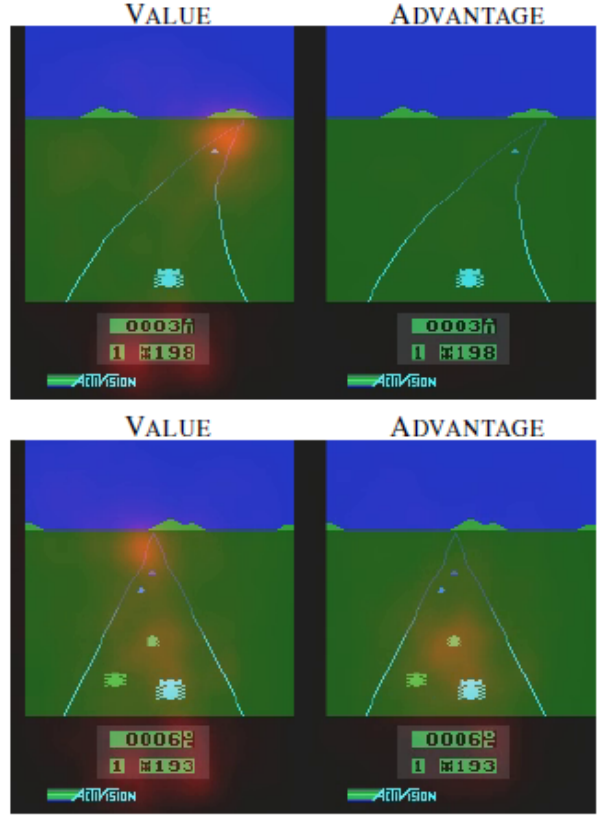


Figure 2. See, attend and drive: Value and advantage saliency maps (red-tinted overlay) on the Atari game Enduro, for a trained dueling architecture. The value stream learns to pay attention to the road. The advantage stream learns to pay attention only when there are cars immediately in front, so as to avoid collisions.