Introduction to Reinforcement Learning

Lecture 2: Function Approximation & Deep RL

Shimon Whiteson
Dept. of Computer Science
University of Oxford

(based on material from
Rich Sutton & Andrew Barto)

August 28, 2019

Where are we so far? (1)

- MDP planning method that exploits the Bellman equation
- Complexity of value iteration:
  - Per iteration: quadratic in $|S|$ and linear in $|A|$
  - Number of iterations: polynomial in $|S|$ and $\frac{1}{1-\gamma}$
- Efficient considering there are $|A|^{|S|}$ deterministic policies
- But states are usually described using *state features*

$$\mathbf{x}(s) = (x_1(s), x_2(s), \ldots, x_d(s))^\top$$

- *Curse of dimensionality:* $|S|$ is exponential in d
- Missing ingredient is *generalisation*

Where are we so far? (2)

- Model-free RL methods like Q-learning and Sarsa exploit the Bellman equation without needing a model
- Guaranteed to converge to the optimal policy in the limit if:
  1. S and A are finite
  2. $\sum_t \alpha_t^{sa} = \infty$ and $\sum_t (\alpha_t^{sa})^2 < \infty$ (GLIE)
  3. $Var\{R_a^{ss'}\} < \infty$
  4. $\gamma < 1$
- Massively data inefficient
- Missing ingredients:
  - Generalisation
  - Data reuse
  - Smart exploration

Approximate value functions

- Value function parameterised by $\mathbf{w} \in \mathcal{R}^d$ where $d \ll |S|$:

$$\hat{V}(s, \mathbf{w}) \approx V^{\pi}(s)$$

- Formulate objective wrt MSE:

$$\min_{\mathbf{w}} \sum_{s \in S} \mu(s)[V^{\pi}(s) - \hat{V}(s, \mathbf{w})]^2,$$

  where μ is the *on-policy distribution*

- Reduces policy evaluation to an (active, incremental, nonstationary) supervised learning problem

Update rule

- Update using SGD:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{\alpha}{2}\nabla\left[V^\pi(s_t) - \hat{V}(s_t, \mathbf{w}_t)\right]^2 \\ &= \mathbf{w}_t + \alpha\left[V^\pi(s_t) - \hat{V}(s_t, \mathbf{w}_t)\right]\nabla\hat{V}(s_t, \mathbf{w}_t)\end{aligned}$$

- Since $V^\pi(s_t)$ is unknown, use Monte Carlo:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\left[R_t - \hat{V}(s_t, \mathbf{w}_t)\right]\nabla\hat{V}(s_t, \mathbf{w}_t)$$

- Any unbiased *target* like R_t ensures convergence to a local optimum

Semi-gradient TD(0)

- Bootstrapping target:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\left[r_{t+1} + \gamma \hat{V}(s_{t+1}, \mathbf{w}_t) - \hat{V}(s_t, \mathbf{w}_t)\right] \nabla \hat{V}(s_t, \mathbf{w}_t)$$

- *Semi-gradient*: treats the $\mathbf{w}_t$ in the target as a constant
- Converges in linear case
- There are true gradient methods, e.g., *residual gradients* [Baird 1995]:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\left[r_{t+1} + \gamma \hat{V}(s_{t+1}, \mathbf{w}_t) - \hat{V}(s_t, \mathbf{w}_t)\right] (\nabla \hat{V}(s_t, \mathbf{w}_t) - \gamma \nabla \hat{V}(s_{t+1}, \mathbf{w}_t))$$

  or *gradient TD* [Sutton et al. 2009] but these are slow in practice

Linear function approximation (1)

- Let $\mathbf{x}(s) = (x_1(s), x_2(s), \ldots, x_d(s))^\top$ be a feature vector such that

$$\hat{V}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s) = \sum_{i=1}^{d} w_i x_i(s)$$

- The gradient becomes $\nabla \hat{V}(s, \mathbf{w}) = \mathbf{x}(s)$ and TD(0) is:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[r_{t+1} + \gamma \hat{V}(s_{t+1}, \mathbf{w}_t) - \hat{V}(s_t, \mathbf{w}_t) \right] \mathbf{x}(s_t)$$

- Convergence to local optimum $\implies$ convergence to global optimum

Linear function approximation (2)

- But linear semi-gradient TD(0) converges to *TD fixed point* instead
- The update rule can be rearranged, where $\mathbf{x}_t = \mathbf{x}(s_t)$:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\big(r_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t\big)\mathbf{x}_t \\ &= \mathbf{w}_t + \alpha\big(r_{t+1}\mathbf{x}_t - \mathbf{x}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top \mathbf{w}_t\big)\end{aligned}$$

- The expected next weight vector is then:

$$\mathbb{E}[\mathbf{w}_{t+1}|\mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t),$$

where:

$$\mathbf{A} = \mathbb{E}\big[\mathbf{x}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top\big] \quad \text{and} \quad \mathbf{b} = \mathbb{E}[r_{t+1}\mathbf{x}_t]$$

Linear function approximation (3)

- Convergence implies:

$$\begin{aligned} \mathbf{b} - \mathbf{A}\mathbf{w}_{TD} &= \mathbf{0} \\ \mathbf{b} &= \mathbf{A}\mathbf{w}_{TD} \\ \mathbf{w}_{TD} &= \mathbf{A}^{-1}\mathbf{b}, \end{aligned}$$

- Relationship to minimum:

$$\mathrm{MSE}(\mathbf{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\mathbf{w}} \mathrm{MSE}(\mathbf{w})$$

Least squares temporal differences

- Estimate $\mathbf{A}$ and $\mathbf{b}$ directly, not iteratively:

$$\hat{\mathbf{w}}_t = \hat{\mathbf{A}}_t^{-1}\hat{\mathbf{b}}_t,$$

where:

$$\hat{\mathbf{A}} = \sum_{k=0}^{t-1} \mathbf{x}_k(\mathbf{x}_k - \gamma\mathbf{x}_{k+1})^\top + \epsilon\mathbf{I} \quad \text{and} \quad \hat{\mathbf{b}} = \sum_{k=0}^{t-1} r_{k+1}\mathbf{x}_k$$

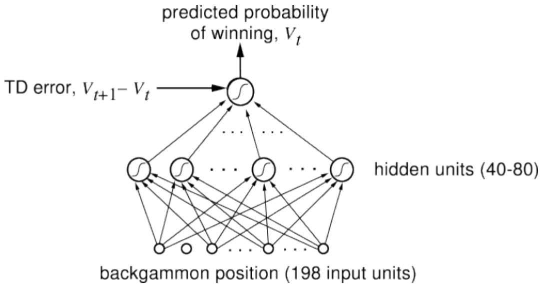
- Cost to compute $\hat{\mathbf{A}}$ and $\hat{\mathbf{b}}$ depends on t unless updated incrementally:

$$\hat{\mathbf{A}}_t = \hat{\mathbf{A}}_{t-1} + \mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top \quad \text{and} \quad \hat{\mathbf{b}}_t = \hat{\mathbf{b}}_{t-1} + r_{t+1}\mathbf{x}_t$$

- Matrix inversion is generally $O(d^3)$ but $\hat{\mathbf{A}}_t$ is a sum of outer products and can be inverted in $O(d^2)$ using the Sherman-Morrison formula

Nonlinear function approximation

- Neural networks represent the value function
- d inputs: $x_1(s), x_2(s), \ldots, x_d(s)$
- Single output estimates $V(s)$
- Early success: TD-Gammon [Tesauro, 1992, 1995, 1996, 2002]
- Uses partial model and evaluates *afterstates*



On-policy semi-gradient control

- Now $\mathbf{w}$ parameterises Q instead of V:

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

- Semi-gradient Sarsa:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\big[r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}, \mathbf{w}_t) - \hat{Q}(s_t, a_t, \mathbf{w}_t)\big]\nabla \hat{Q}(s_t, a_t, \mathbf{w}_t)$$

- Continuous states are fine

- Continuous actions make policy improvement hard

Nonlinear control

- Neural networks represent the value function
- d inputs: $x_1(s), x_2(s), \ldots, x_d(s)$
- $|A|$ outputs: $Q(s, a_1), Q(s, a_2), \ldots, Q(s, a_{|A|})$
- Allows action selection with one forward pass

Off-policy function approximation

- Naive off-policy semi-gradient TD(0):

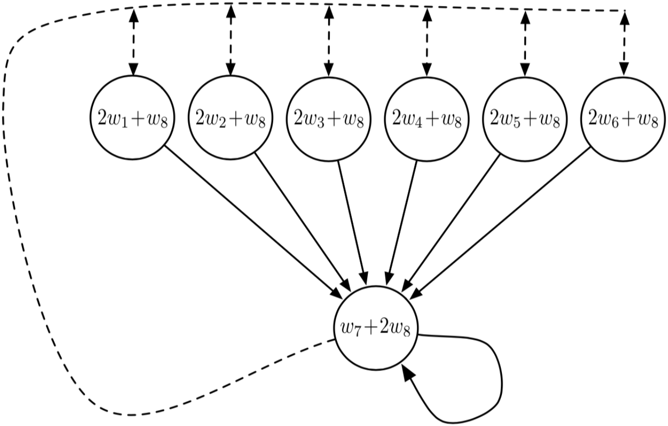
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \frac{\pi(s_t, a_t)}{\pi'(s_t, a_t)} \left[r_{t+1} + \gamma \hat{V}(s_{t+1}, \mathbf{w}_t) - \hat{V}(s_t, \mathbf{w}_t) \right] \nabla \hat{V}(s_t, \mathbf{w}_t)$$

- Semi-gradient Q:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a, \mathbf{w}_t) - \hat{Q}(s_t, a_t, \mathbf{w}_t) \right] \nabla \hat{Q}(s_t, a_t, \mathbf{w}_t)$$

- Both known to be vulnerable to divergence

Baird's counterexample [1995]



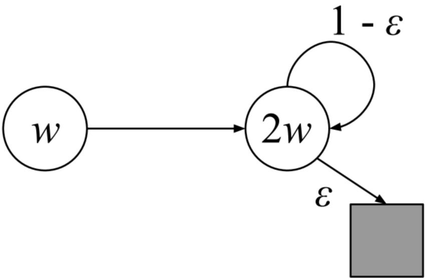
$\pi(\text{solid}|\cdot) = 1$

$b(\text{dashed}|\cdot) = 6/7$

$b(\text{solid}|\cdot) = 1/7$

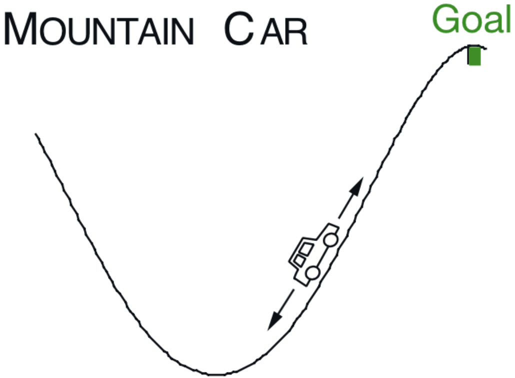
$\gamma = 0.99$

Tsitsiklis & Van Roy counterexample [1997]



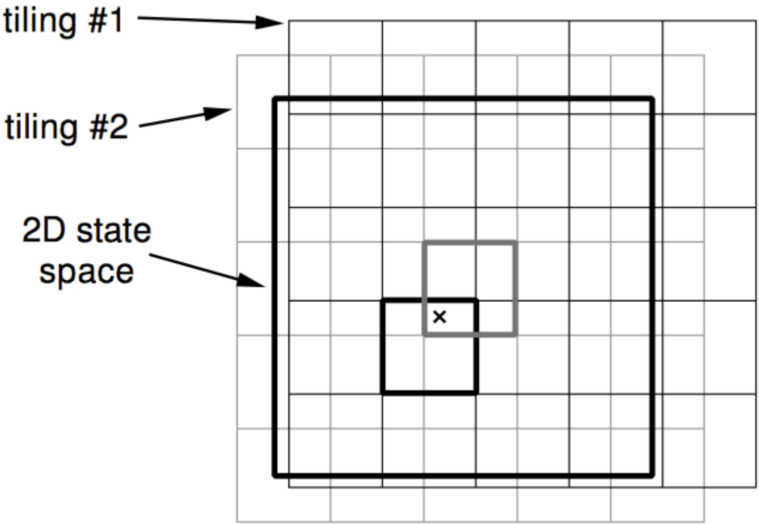
- $V(s) = w\phi(s)$, where $\phi(s_i) = i$
- $\forall i, R(s_i) = 0 \implies w^* = 0$
- Only update s_1:
  - $\Delta w \propto \gamma 2w - w$
  - $\gamma > 0.5 \implies$ divergence
- Even uniform updates of s_1 and $s_2 \implies$ divergence for large γ

Mountain car



- Boyan & Moore [1995] showed Q-learning's failure with nonlinear FA
- Sutton [1996] succeeded with Sarsa with linear tile coding

Tile coding



Deadly triad [Sutton & Barto 2018]

1. Function approximation
2. Bootstrapping
3. Off-policy learning

Are all three essential?

Deadly triad [Sutton & Barto 2018]

1. Function approximation
2. Bootstrapping
3. Off-policy learning

Are all three essential?

Not in the triad:

1. Control
2. Learning
3. Nonlinearity

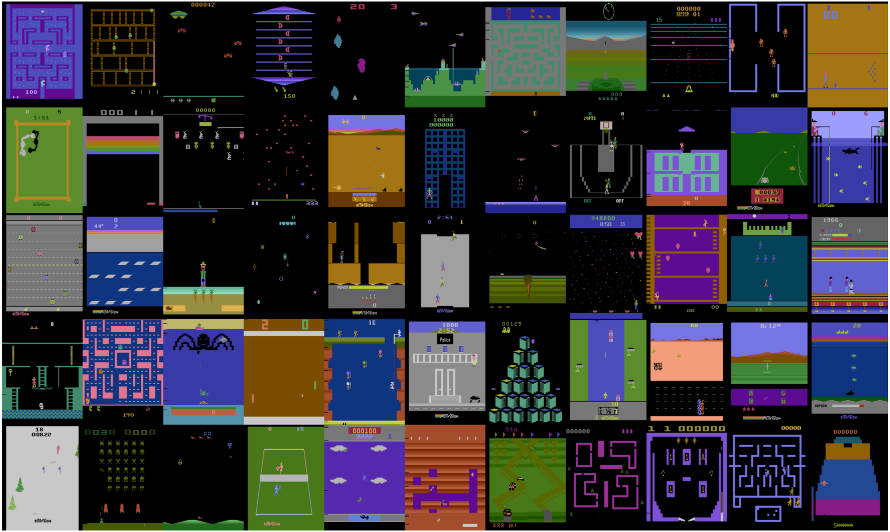
Experience replay [Lin 1992]

- All methods discussed so far (except LSTD) are sample inefficient
- Binning the data after one use is madness
- Experience replay stores samples $d_t = (s_t, a_t, r_{t+1}, s_{t+1})$
- Repeatedly replays them to the agent
- More computation but fewer samples

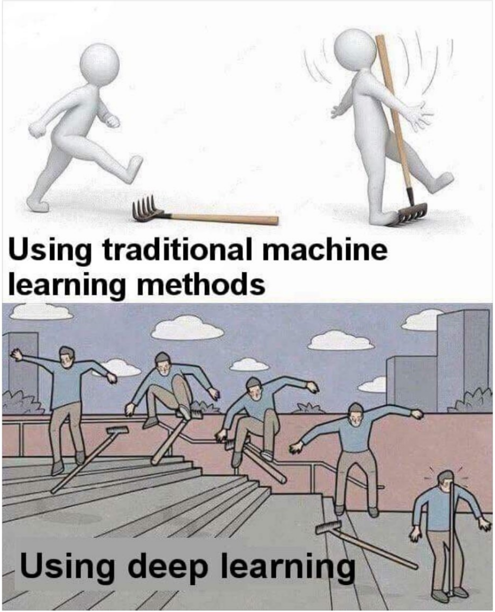
(Neural) fitted Q-iteration [Riedmiller 2005] [Ernst et al. 2005]

- Store all samples as in experience replay
- Initialise $\mathbf{w}$
- For $i = 0, 1, \ldots$
  - For each d_t, construct target $y_t^i = r_{t+1} + \gamma \max_a \hat{Q}(s_t, a_t, \mathbf{w})$
  - For $j = 0, 1, \ldots$
    - Sample a datapoint d_t
    - $\mathbf{w} \leftarrow \mathbf{w} + \alpha [y_t^i - \hat{Q}(s_t, a_t, \mathbf{w})] \nabla \hat{Q}(s_t, a_t, \mathbf{w})$
- Targets remain fixed during inner loop

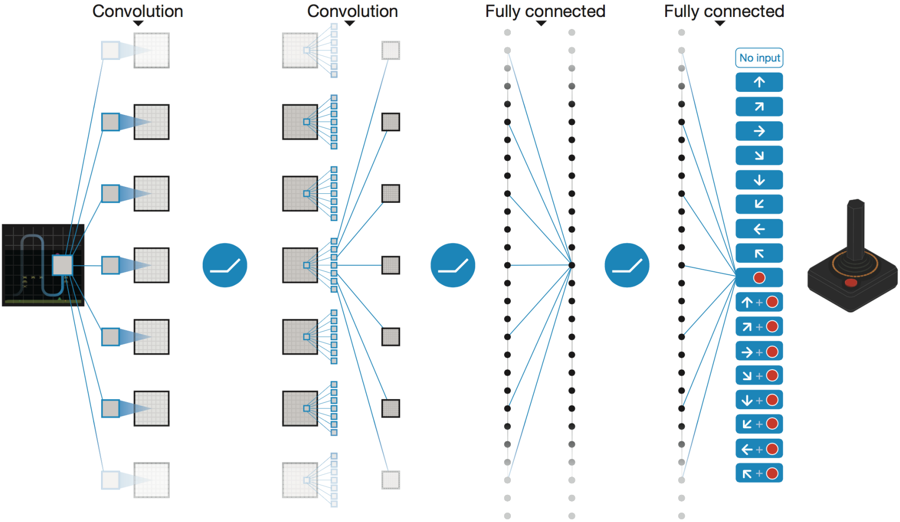
Atari learning environment



Deep reinforcement learning



DQN [Mnih et al. 2015]



DQN [Mnih et al. 2015]

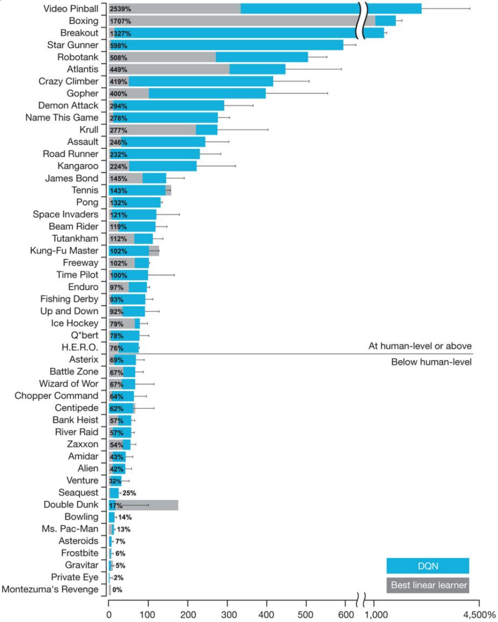
- DQN update rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a, \mathbf{w}^-) - \hat{Q}(s_t, a_t, \mathbf{w}_t) \right] \nabla \hat{Q}(s_t, a_t, \mathbf{w}_t)$$

  where $\mathbf{w}^-$ are the weights of a frozen *target network*

- Every k updates: $\mathbf{w}^- \leftarrow \mathbf{w}_t$
- Yields a cheap approximation to NFQ
- Gradients estimated from mini-batches
- Mini-batches randomly sampled via experience replay

DQN results


Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

Rainbow [Hessel et al. 2017]

- Double Q-learning [van Hasselt et al. 2015]
- Prioritised replay [Schaul et al. 2015]
- Duelling networks [Wang et al. 2016]
- Multi-step targets [Sutton 1988]
- Distributional RL [Bellemare et al. 2017]
- Noisy nets [Fortunato et al. 2017]

Double DQN [van Hasselt et al. 2015]

- Q-learning takes max of noisy Q estimate: yields bias
- Instead separate estimation from maximisation
- Note that:

$$\max_a \hat{Q}(s_{t+1}, a, \mathbf{w}_t) = \hat{Q}(s_{t+1}, \arg\max_a \hat{Q}(s_{t+1}, a, \mathbf{w}_t), \mathbf{w}_t)$$

- Double Q-learning uses two independent sets of weights:

$$\hat{Q}(s_{t+1}, \arg\max_a \hat{Q}(s_{t+1}, a, \mathbf{w}_t), \mathbf{w}'_t)$$

- Double DQN uses target network, yielding update target:

$$r_{t+1} + \gamma \hat{Q}(s_{t+1}, \arg\max_a \hat{Q}(s_{t+1}, a, \mathbf{w}_t), \mathbf{w}^-)$$

- Why not swap $\mathbf{w}_t$ and $\mathbf{w}^-$?

Prioritised replay [Hessel et al. 2017]

- Prioritised sweeping [Moore & Atkeson 1993]
  - Model-based RL
  - Efficient planning upon model updates
  - Starting from updated state, put tree of predecessors in priority queue
  - Priority is magnitude of update, i.e., TD error
- Prioritised replay [Schaul et al. 2015] extends to model-free RL
  - Sample transitions from replay buffer with probability based on last encountered absolute TD error:

$$p_t \propto \left| r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a, \mathbf{w}^-) - \hat{Q}(s_t, a_t, \mathbf{w_t}) \right|^\omega$$

  - New transitions have maximal priority
  - Can inappropriately favour stochastic transitions

Duelling networks [Wang et al. 2016]

- *Advantage function* compares given action to expected action:

$$A(s, a) = Q(s, a) - V(s)$$

- Could represent $Q(s, a)$ as sum of two parts:

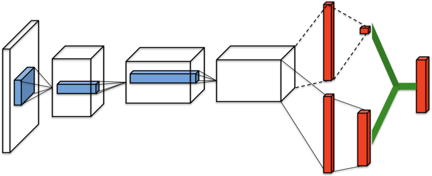
$$\hat{Q}(s, a) = \hat{V}(s) + \hat{A}(s, a)$$

- To improve *identifiability* force advantage of a^* to be zero:

$$\hat{Q}(s, a) = \hat{V}(s) + \hat{A}(s, a) - max_{a'} \hat{A}(s, a')$$

- More stable to use average instead of max:

$$\hat{Q}(s, a) = \hat{V}(s) + \hat{A}(s, a) - \frac{1}{|A|} \sum_{a'} \hat{A}(s, a')$$



Multi-step targets [Sutton 1988]

- The n-step return is:

$$R_t^n = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1}$$

- Multi-step DQN target:

$$R_t^n + \gamma^n \max_a \hat{Q}(s_{t+1}, a, \mathbf{w}^-)$$

- Is this on-policy or off-policy?

Distributional RL [Bellemare et al. 2017]

- Distributional RL learns the distribution of returns instead of the expected returns
- Represent distribution with probability masses placed at discrete support points
- Return distribution satisfies as variant of the Bellman equation
- TD error becomes a KL divergence
- Models aleatoric, not epistemic, uncertainty
- Why does it work? [Imani & White 2018]

Noisy nets [Fortunato et al. 2017]

- Replace linear layer $\mathbf{b} + \mathbf{W}\mathbf{x}$ with:

$$\mathbf{b} + \mathbf{W}\mathbf{x} + (\mathbf{b}_{noisy} \odot \epsilon^{b} + \mathbf{W}_{noisy} \odot \epsilon^{w})\mathbf{x})$$

  where ϵ^b and ϵ^w are random variables, e.g., Gaussian and $\odot$ denotes element-wise product

- Over time network can learn to ignore noisy stream

- Rate differs across search space

- Automatic state-conditional annealing of exploration