Lecture Notes for

Deep Learning and Artificial Intelligence

Winter Semester 2024/2025

Value Function Approximation

DBS

Short Comings of the methods so far

So far: All methods work on a discrete state space S.

$\Rightarrow$ A policy π is a table of the form $\{(s_1, a_1), .., (s_{|S|}, a_{|S|})\}$

$\Rightarrow$ If we encounter a new state, we do not know what to do.

$\Rightarrow$ No matter how similar two states are, we learn Q(s,a) independently.

$\Rightarrow$ If |S| is very large,

- we need a lot of memory to store the policy.
- we need large amounts of samples to estimate Q(s,a) for all state-action pairs.

$\Rightarrow$ Previous models for MDPs and Reinforcement learning become infeasible

Some examples

- Number of states for some problems
  - Backgammon: 10^{20}
  - Computer Go: 10^{170}
  - Flying an RC Helicopter: continuous state space

Working with continuous State Spaces

Idea: What if we do not distinguish states but state descriptions,
e.g., feature vectors?

- Depending on the feature space, we can describe an infinite set of states.
- A policy can be described as a function *f* of a continuous state space
  - $\Rightarrow$ f(x,a) = Q(s,a) or f(x) = a
  - $\Rightarrow$ closed-form functions are much more space efficient than tables
- Feature spaces can preserve similarity relation
  ⇒ if we do not have encountered a particular state so far, we can derive a suitable action based on the policy for similar, better-explored states. (generalisation)

Generally: Working on state descriptions allows for flexible agents to be able to cope with unknown situations.

Overview on continuous State Spaces

- Value function approximation (this lecture)
  - Learn a function *f* to predict *U(x(s))* or *Q(x(s),a)*
    (In general, *f* is a regression function of some kind)

- Policy gradient methods: (next lecture)
  - Directly learn a function *f(x(s))* predicting the best action *a* for *x(s)*

- Actor-Critic methods: (next lecture)
  - combine policy functions and value function approximation

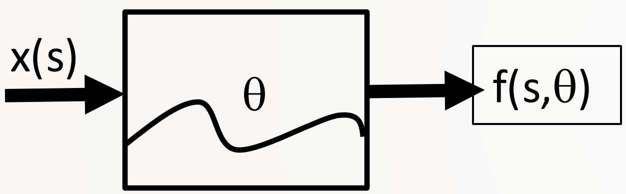
Value function approximation

Given: A mapping *x(s)* describing *s* in IR^d.

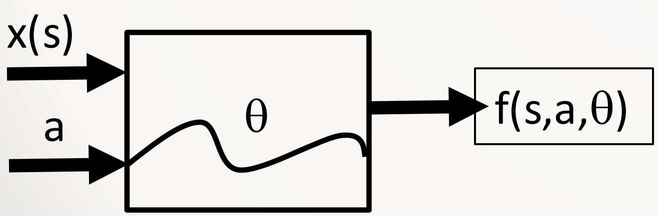
Idea: Learn a function that either describes the utility U(S) or the state-value function Q(S,A).

Options to learn the $\mathrm{f}(s,\theta) \approx U(S)$or $\mathrm{f}(s,a,\theta) \approx Q(S,A)$:

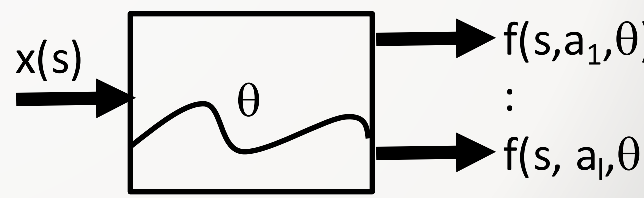
- Approximate U(*S)*



- *Approximate Q(S,A)*



or



Value Function Approximation and Partial Observability

A side-effect of using value function approximation is that we can work on a factor space representing the exact state S or just an observation O.

- factor spaces: often the state can be coded as a set of (independent) parameters:

  Example: position of the agent + state variables of the environment, Stockmarket: recent course development for all traded stocks, …

- Observation spaces: a set of parameters giving us hints about the state.

  Examples: video buffer of a camera, sensor data, player view in a video game,.

$\Rightarrow$ Since $f(x(s),a, \theta)$ is an approximation function works for both settings ($f(x(s),a, \theta)$ can learn to consider belief states)

$\Rightarrow$ **Caution**: Make sure that $x(s)$ is Markov !!!

Mean Squared Value Error

Regardless of how we built our approximation function $f(S,\theta)$, we need a measure for the quality of an approximation:

$$\overline{VE}^{\pi}(\theta) = \mathbb{E}_{\pi,s\sim\mu}[(U_\pi(S) - f(S,\theta))^2]$$
$$= \sum_{s\in S} \mu(S)\,(U_\pi(S) - f(S,\theta))^2$$

where μ is the importance distribution over the state descriptions with $\sum_{s\in S}\mu(S) = 1$.

For example, we can take $\mu(S)$ as the likelihood of being in state s when following π.

Common types of function approximations

- Generally, any regression/prediction function can be used (usually, we will require a continuous return to model the Utility)
- Common methods:
  - Linear predictors
  - Neural networks
  - Decision trees
  - Regression with Fourier/Wavelet bases
  - ..
- However: Reinforcement learning is tricky because:
  - experience is non-stationary: Q(S,A) constantly changes when using TD learning
  - experience is not independently drawn: the observation from a single episode is usually highly correlated

  ⇒ data is not identically and independently distributed (non-IID data)

Value Function Approx. with SGD

Goal: Given policy π and $U_\pi(S)$ find θ minimising the loss function $L^\pi\,(\theta)$.
Note: We won't have $U_\pi(S)$ but only R(S) later on.

For example, consider $L_{X,Y}(\theta)$ is mean square loss:

$$L^\pi\,(\theta) = \mathbb{E}_\pi[(U_\pi(S) - f(S,\theta))^2]$$

Computing the gradient, we get

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta L^\pi\,(\theta) = \alpha\mathbb{E}_\pi\big[\big(U_\pi(S) - f(S,\theta)\big)\nabla_\theta f(S,\theta)\big]$$

- with SGD we sample the gradient: $\alpha\big(U_\pi(S) - f(S,\theta)\big)\nabla_\theta f(S,\theta)$ and incrementally update to minimize the loss $L^\pi\,(\theta)$.

- the expected update is equal to the full gradient update

Linear Prediction Functions

A simple function approximation would be the linear function.

- Linear Functions over x(S) $\in \mathbb{R}^d$ where θ is a weight vector w:

$$f(x(S), \mathrm{W}) = \mathrm{x(S)^T W} = \sum_{j=1}^{n} x(S)_j^T w_j$$

- Loss function:

$$L(W) = E[(U(s) - \mathrm{x(s)^T W})^2]$$

- Stochastic Gradient Descent on L(w):

$$\nabla W \ f(x(s), \mathrm{W}) = \mathrm{x(s)}$$

$$-\frac{1}{2}\nabla L(\theta) = \big(U(s) - f(x(s), \theta)\big)x(\mathrm{s})$$

$$\Delta\theta = \alpha\big(U(s) - f(x(s), \theta)\big)x(\mathrm{s})$$

$$\text{update} = step - size \times prediction\ error\ \times feature\ vector$$

Table Lookup Features

- Table lookups can be considered as a special case of linear value function approximation
- Use a lookup table of the of the following form:

$$x^{table}(S) = \begin{pmatrix} 1(S = s_1) \\ \vdots \\ 1(S = s_n) \end{pmatrix}$$

- Parameter vector *w* gives us the value of each state:

$$f(x(S), w) = \begin{pmatrix} 1(S = s_1) \\ \vdots \\ 1(S = s_n) \end{pmatrix}^T \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}$$

Incremental Prediction algorithms

- In practice, we do not have the utility $U_\pi(S)$ but only R(S)

$\Rightarrow$ We have to employ a target for $U_\pi(S)$ as in the last lecture

Prediction based on value function approximation:

- For MC, the target is the complete return G_t

$$\Delta\theta = \alpha\big(G_t - f(x(S_t),\theta)\big)\nabla_\theta f(x(S_t),\theta)$$

- For TD, the target is the TD target $R_{t+1} + \gamma f(x(S_{t+1}),\theta)$

$$\Delta\theta = \alpha\big(R_{t+1} + \gamma f(x(S_{t+1}),\theta) - f(x(S_t),\theta)\big)\nabla_\theta f(x(S_t),\theta)$$

- For TD(λ), the target is the λ-return G_t^λ

$$\Delta\theta = \alpha\left(G_t^\lambda - f(x(S_t),\theta)\right)\nabla_\theta f(x(S_t),\theta)$$

Caution: For TD and TD(λ) the target depends on θ

$\Rightarrow$ TD and TD(λ) are semi-gradient methods because the gradient is only computed w.r.t. $f(x(S_t),\theta)$, but not for the target functions.

MC with value function approximation

- Return G_t is an unbiased, noisy sample of true value $U(S)$
- Applying supervised learning to known experience is viable:
  $(x(S_1), G_1), (x(S_2), G_2), \ldots (x(S_T), G_T)$

- For example, linear Monte-Carlo policy evaluation:

$$\begin{aligned}\Delta\theta &= \alpha\big(G_t - f(x(S_t), \theta)\big)\nabla_\theta f(x(S_t), \theta) \\ &= \alpha\big(G_t - f(x(S_t), \theta)\big) \cdot x(S_t)\end{aligned}$$

- Monte-Carlo evaluation converges to a local optimum of the loss function even when using non-linear value function approximation
- note: this does not imply convergence towards the true q-values

TD with value function approximation

- The TD-target is a biased sample of the true value $U(S)$
- Applying supervised learning is still possible but training data looks like:

$\big(x(S_1), R_1 + \gamma f(x(S_2), \theta)\big), \big(x(S_1), R_2 + \gamma f(x(S_3), \theta)\big), \ldots (x(S_{T-1}), R_T)$

- For example, linear TD(0) policy evaluation:

$$\begin{aligned}\Delta\theta &= \alpha\big(R_t + \gamma f(x(S_{t+1}), \theta) - f(x(S_t), \theta)\big)\nabla_\theta f(x(S_t), \theta) \\ &= \alpha\big(R_t + \gamma f(x(S_{t+1}), \theta) - f(x(S_t), \theta)\big) \cdot x(S_t) \\ &= \alpha\delta \cdot x(S_t)\end{aligned}$$

- Linear TD(0) converges (close) to global optimum

TD(λ) with value function approximation

- The λ-return is also a biased sample sample of true value $U(S)$
- Applying supervised learning is to training data of the form:

  $\left(x(S_1), G_1^\lambda\right), \left(x(S_2), G_2^\lambda\right), .., \left(x(S_{T-1}), G_{T-1}^\lambda\right)$

- Forward view of linear TD(λ):

$$\begin{aligned}\Delta\theta &= \alpha\left(G_1^\lambda - f(x(S_t), \theta)\right) \nabla_\theta f(x(S_t), \theta) \\ &= \alpha\left(G_1^\lambda - f(x(S_t), \theta)\right) \cdot x(S_t)\end{aligned}$$

- Backward view of linear TD(λ):

$$\delta_t = R_{t+1} + \gamma f(x(S_{t+1}), \theta) - f(x(S_t), \theta)$$

$$E_t = \gamma\lambda E_{t-1} + x(S_t)$$

$$= \alpha\delta_t \cdot E_t$$

Convergence of Prediction Methods

On/Off policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
On-Policy	TD	✓	✓	✗
On-Policy	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
Off-Policy	TD	✓	✗	✗
Off-Policy	TD(λ)	✓	✗	✗

Control and Value Function Approximation

- To apply policy iteration, we again have to switch to state-value functions Q(S,A)

- Basic idea for on-policy learning:
  - approximate q_π with a function q(x(S),a,θ):
    $\hat{q}(S, A, \theta) \approx q_\pi(S, A)$
  - employ $\varepsilon - greedy$ policy improvement

Caution:

- It is not necessary to approximate $q_\pi(S, A)$ very accurately. Instead, we take a step into improving $\hat{q}(S, A, \theta)$ and then adjust the policy.
- Using function approximation is not guaranteed to converge against $q_\pi(S, A)$. Since $\hat{q}(S, A, \theta)$ is a regression function, it is not guaranteed that the model can describe the real $q_\pi(S, A)$ for all (S,A).

Control and Value Function Approximation

To learn a reasonably close $\hat{q}(S, A, \theta)$, we can:

- Minimize the mean square error between the approximation $\hat{q}(S, A, \theta)$ and the true action value $q_\pi(S, A)$:

$$L(\theta) = \mathbb{E}_{\pi, s\sim\mu}\left[\left(q_\pi(s, a) - \hat{q}(s, a, \theta)\right)^2\right]$$

- Optimization via SGD:

$$-\frac{1}{2}\nabla_\theta L(\theta) = \left(q_\pi(S, A) - \hat{q}(S, A, \theta)\right)\nabla_\theta \hat{q}(S, A, \theta)$$

$$\Delta\theta = \alpha\left(q_\pi(S, A) - \hat{q}(S, A, \theta)\right)\nabla_\theta \hat{q}(S, A, \theta)$$

Control with Linear Value Functions

- State-action are modelled as a feature vector:

$$x(S,A) = \begin{pmatrix} x_1(S,A) \\ \vdots \\ x_n(S,A) \end{pmatrix}$$

- Represent action-value function by linear combination of features

$$\hat{q}(S\ ,A,w) = \sum_{j=1}^{n} x_j(S,A)w_j$$

- With the SGD update:

$$-\frac{1}{2}\nabla_w L(w) = \left(q_\pi(S,A) - (x(S,A)^T w)\right)\nabla_w (x(S,A)^T w\ )$$

$$= \left(q_\pi(S,A) - (x(S,A)^T w)\right)w$$

$$\Delta w = \alpha\left(q_\pi(S,A) - \hat{q}(S,A,w)\right)x(S,A)$$

Incremental Control Algorithms

similar to prediction but substitute $q_\pi(S, A)$:

- For MC, the target is the complete return G_t

$$\Delta\theta = \alpha\big(G_t - \hat{q}(S_t, A_t, \theta)\big)\nabla_\theta \hat{q}(S_t, A_t, \theta)$$

- For TD, the target is the TD target $R_{t+1} + \gamma\hat{q}(S_t, A_t, \theta)$

$$\Delta\theta = \alpha\big(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \theta) - \hat{q}(S_t, A_t, \theta)\big)\nabla_\theta \hat{q}(S_t, A_t, \theta)$$

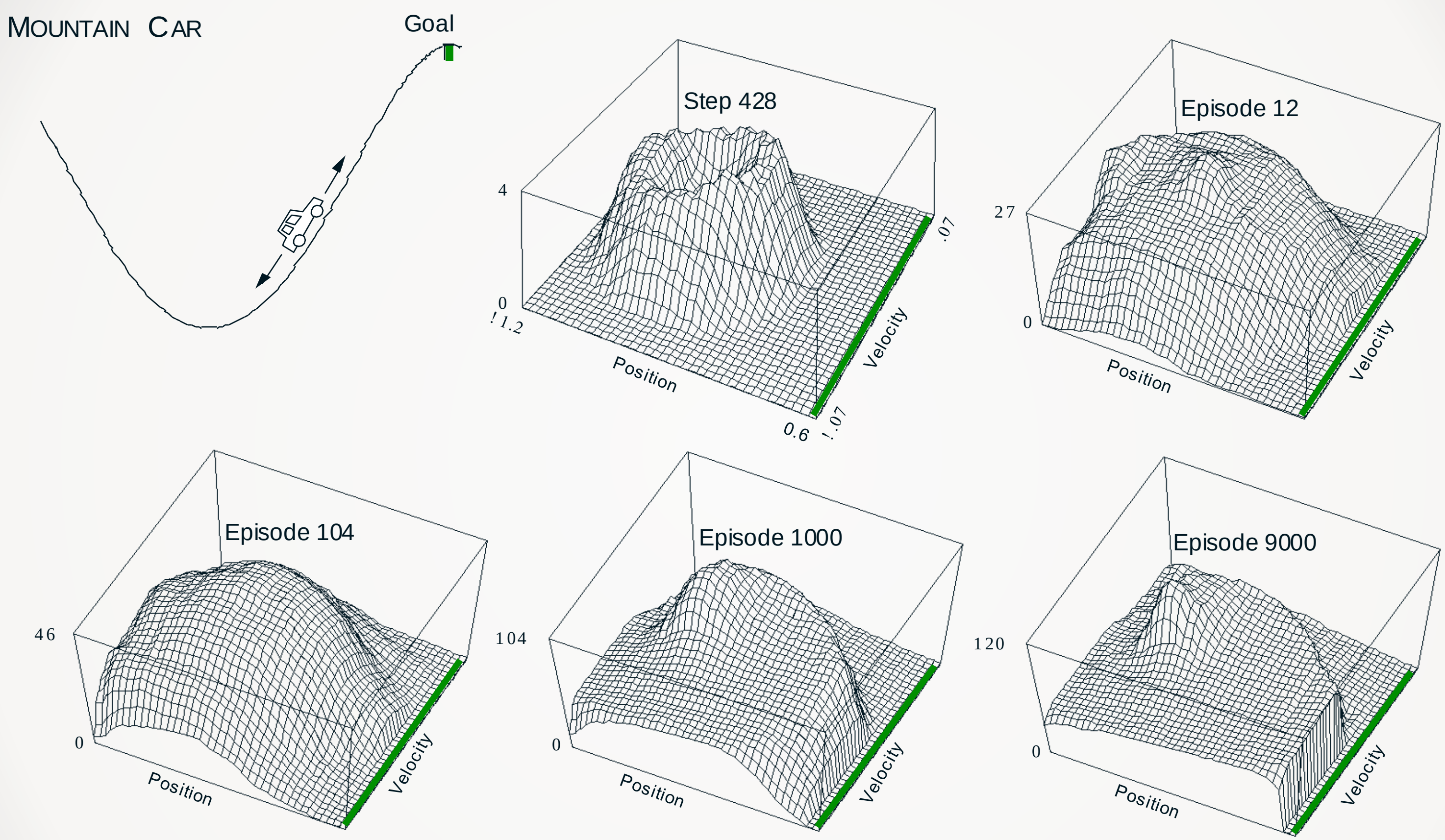
- For TD(λ), the target is the action-value λ-return q_t^λ:
  - Forward view

$$\Delta\theta = \alpha\left(q_t^\lambda - \hat{q}(S_t, A_t, \theta)\right)\nabla_\theta \hat{q}(S_t, A_t, \theta)$$

  - Backward view:

$$\delta_t = R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \theta) - \hat{q}(S_t, A_t, \theta)$$
$$E_t = \lambda\gamma E_{t-1} + \nabla_\theta \hat{q}(S_t, A_t, \theta)$$
$$\Delta\theta = \alpha\delta_t E_t$$

Example: Mountain Car



R. S. Sutton, A. G. Barto: **Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning),** The MIT Press; Auflage: 2., 2018, page 245, Fig 10.1

Semi-gradient Sarsa method with tile-coding

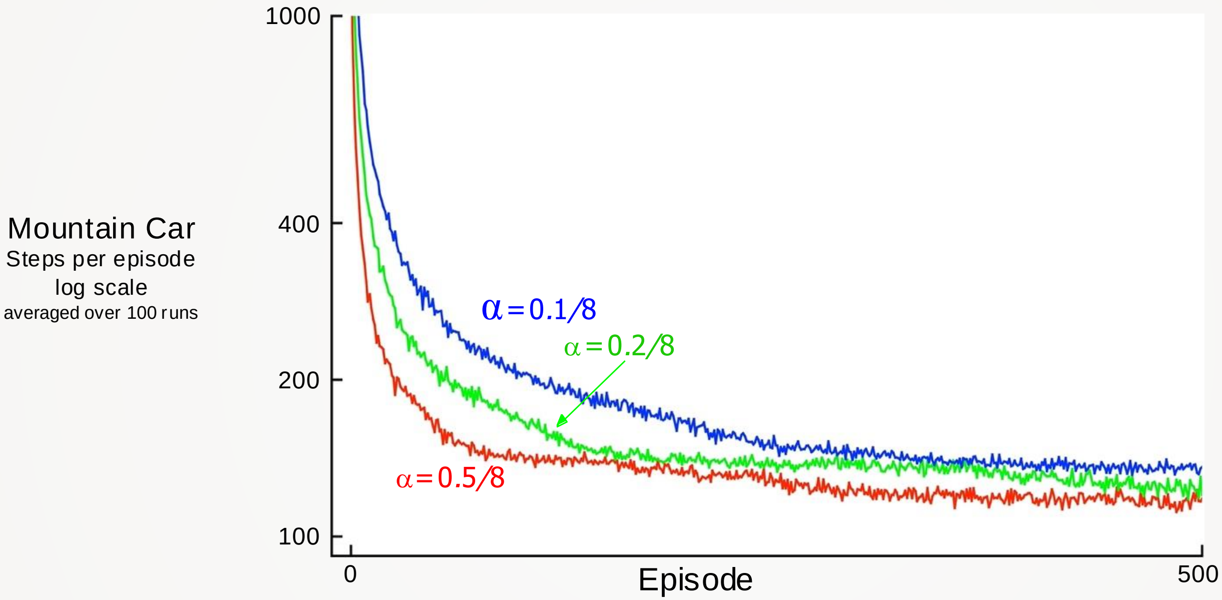


Figure 10.2: Mountain Car learning curves for the semi-gradient Sarsa method with tile-coding function approximation and ϵ-greedy action selection.

Control and Convergence

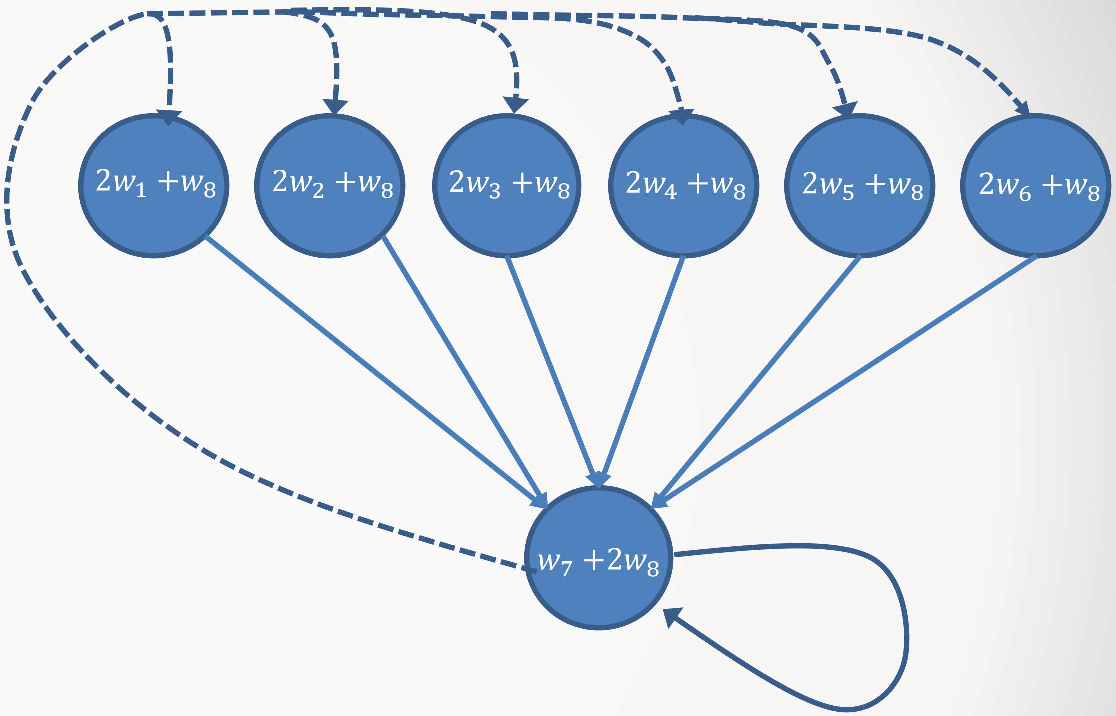
- Convergence to the minimal error between Q(S,A,θ) and $q_{\pi}(S, A)$ is problematic.
- Generally, convergence is problematic if we employ:
  - Value Function Approximation
  - Bootstrapping
  - Off-Policy Learning

  (Deadly Triad)

$\Rightarrow$ For these cases, updates might even increase the error.

Baird's Counterexample

- episodic MDP with 7 states and 2 actions:
  - Dashed: go to any of the upper states with 1/6
  - Solid: go to lower state 100 %
- reward is always 0
  - $\Rightarrow$ true value functions = 0
- γ=0.999
- behavioural policy b:
  - b$(dashed|\cdot)$ =6/7
  - b$(solid|\cdot)$ =1/7
- target policy:
  - $\pi(solid|\cdot)$ =1.0



feature vectors:

$x(1)=(2,0,0,0,0,0,0,1)^T$

$x(2)=(0,2,0,0,0,0,0,1)^T$

…

$x(7)=(0,0,0,0,0,0,1,2)^T$

Applying semi-gradient TD(0) makes the weights diverge into infinity but switching to on-policy makes the TD(0) converge.

Making gradient methods converge

- TD is not a full GD approach
- Idea: Compute the complete gradient over.
  - Straight-forward the error function is smoothed rather then optimized
- Gradient TD[1] and empathic TD follow the true gradient of projected Bellman error and therefore do not diverge.

On/Off policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
On-Policy	TD	✓	✓	✗
On-Policy	Gradient TD	✓	✓	✓
Off-Policy	MC	✓	✓	✓
Off-Policy	TD	✓	✗	✗
Off-Policy	Gradient TD	✓	✓	✓

[1] D. Silver: **Gradient Temporal Difference Networks** *Proceedings of the Tenth European Workshop on Reinforcement Learning*, PMLR 24:117-130, 2013.
[2] Richard S. Sutton, A. Rupam Mahmood, and Martha White: **An emphatic approach to the problem of off-policy temporal-difference learning,** J. Mach. Learn. Res. 17, 1 (January 2016), 2603–2631

Convergence of Control Methods

Algorithm	Table Lookup	Linear	Non-Linear
MC Control	✓	(✓)	✗
SARSA	✓	(✓)	✗
Q-Learning	✓	✗	✗
Gradient Q-Learning	✓	✓	✗

(✓)= jumps around the near optimal value function

Batch Methods

Utilization of experience is rather bad with GD methods.

⇒ Batch Reinforcement Learning: Find the best fitting value function for the given experience
("training data")

⇒ Using an example only once for making one step might be a waste of experience

Least Squares Prediction

- Given the value function approximation $\mathrm{f}(s, \theta) \approx U(S)$
- The experience $\mathcal{D}$ is given by a set of state-value pairs

$$\mathcal{D} = \{\langle s_1, U^{\pi}(S_1) \rangle, \ldots, \langle s_T, U^{\pi}(s_T) \rangle\}$$

We want to find the parameters θ to provide the value function approximation $\mathrm{f}(s, \theta)$ with the best fit on $\mathcal{D}$.

$\Rightarrow$ The least squares method fits the θ to minimize the sum-squared error between $\mathrm{f}(s, \theta)$and $U(S)$.

$$LS_{\mathcal{D}}(\theta) = \sum_{t=1}^{T} (U^{\pi}(s_t) - f(s, \theta))^2$$
$$\cong \mathbb{E}_{\mathcal{D}}[(U^{\pi}(s_t) - f(s, \theta))^2]$$

SGD with Experience Replay

Given experience $\mathcal{D}$ is given by a set of state-value pairs

$$\mathcal{D} = \{\langle s_1, U^\pi(S_1) \rangle, .., \langle s_T, U^\pi(s_T) \rangle\}$$

Repeat:

- sample state-value pair from $\mathcal{D}$:

$$\left\langle\langle s, U^\pi(s) \rangle\right\rangle \sim \mathcal{D}$$

- apply SGD update on parameter θ:

$$\Delta\theta = \alpha\big(U^\pi(s) - f(s,\theta)\big)\nabla_\theta f(s,\theta)$$

$\Rightarrow$ converges to least squares solution

$$\theta^\pi = arg\max_\theta LS_{\mathcal{D}}(\theta)$$

Experience Replay in Deep Q-Networks (DQN)

- DQN applies deep learning to off-policy, non-linear, TD-target reinforcement learning. (danger of instability)
- using experience replay and fixed Q-targets often achieves stable convergence.
- experience replay:
  - observed transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$.
  - by sampling independently from $\mathcal{D}$ episodes are decoupled

- idea of fixed Q-targets:
  - Q-learning targets in the experience replay are all generated w.r.t. "old", fixed parameters θ^-.
  - Q-targets are independent from θ in f(s, θ) which is updated

Prioritized Experience Replay

idea: Sample experience based on its impact to learning.

- not every piece of experience in the buffer is equally valuable
  - if (s,a,r,s') or similar instances are already well-covered by the function approximation, gradients are small (TD-error is small)
  - instances with larger TD-error are more valuable
- increase the likelihood for being sampled into a batch based on the TD error.
- Caution: batches still have to cover various state-actions and need to keep performing on the well-covered area as well

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

DQN Algorithm

Repeat:

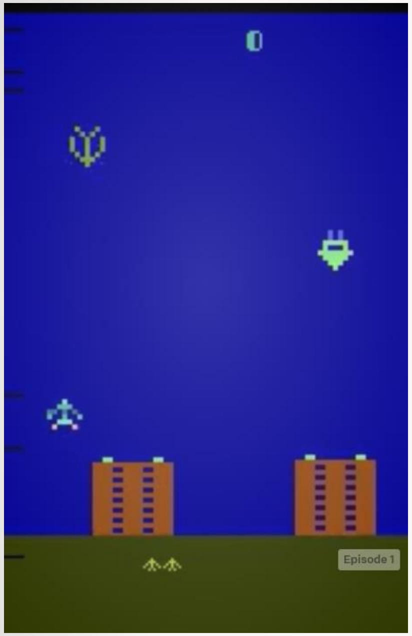
- Take action a_t according to ε-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay buffer $\mathcal{D}$
- Sample random mini-batch of transition (s,a,r,s') from $\mathcal{D}$
- Compute the Q-learning targets w.r.t. fixed θ^-
- Optimize MSE between Q-network and Q-learning targets:

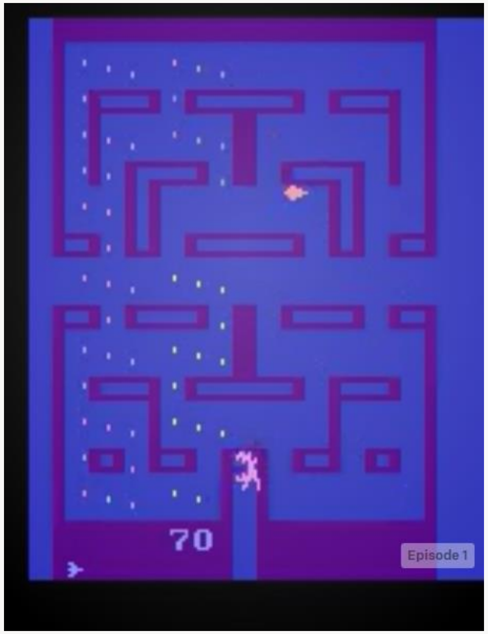
$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}}\left[\left(r + \gamma \max_{a'} f(s', a'; \theta^-) - f(s, a, \theta)\right)^2\right]$$

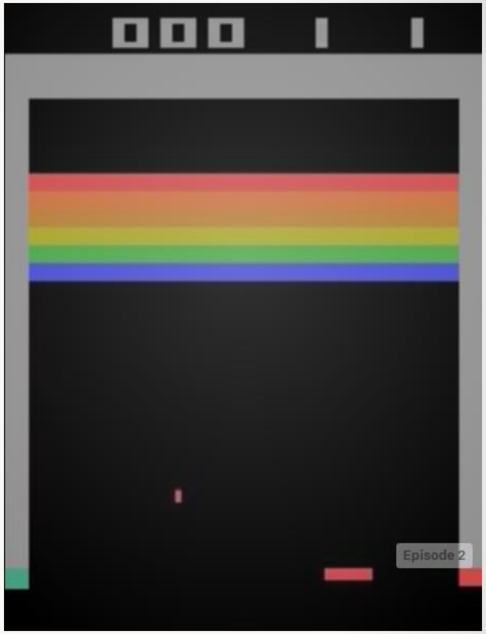
by SGD

Example: DQN on Atari games

https://gym.openai.com/envs/#atari



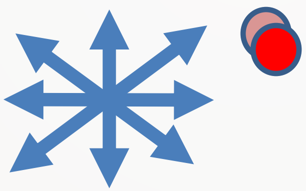




DQN in Atari

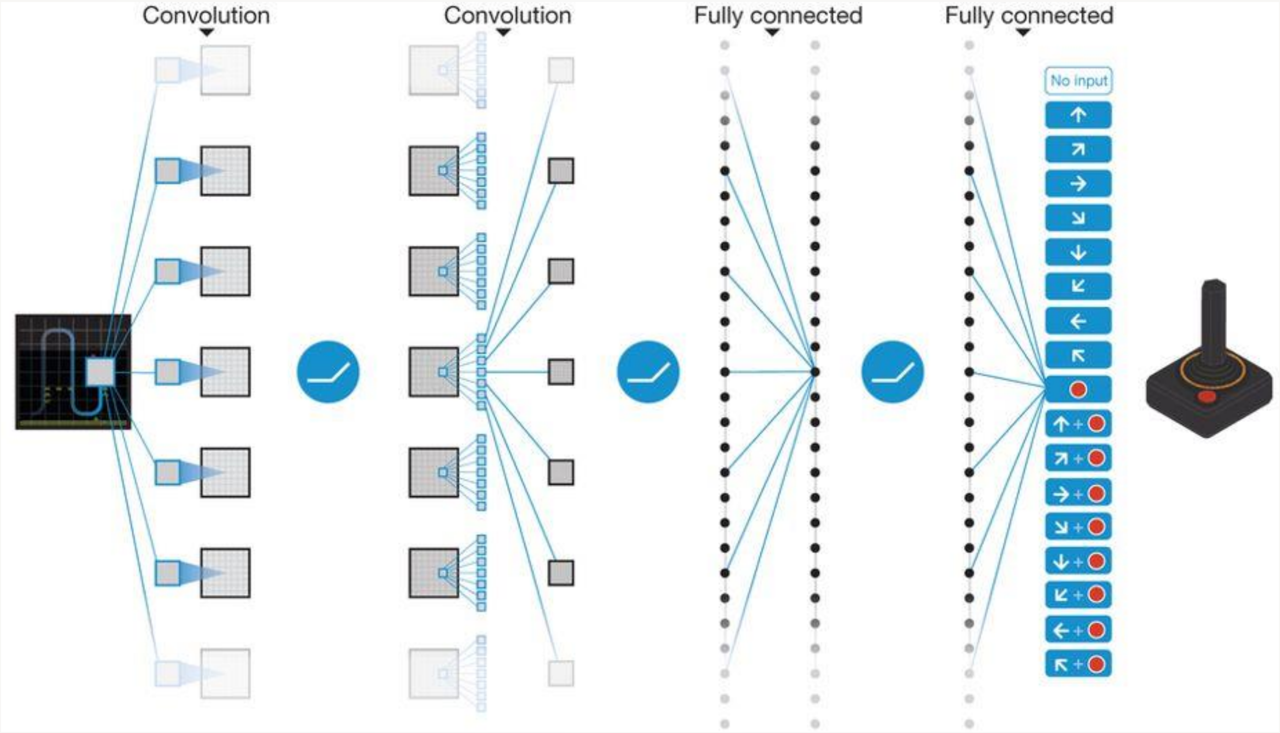
Idea: Use one network architecture to learn multiple computer games on the video buffer as input.

- End-to-end learning of Q(s,a) from pixels *s*
- Input state *s* is stack of raw pixels from last 4 frames
  (a single frame would not be Markov!!)
- Actions: 18 Joystik/button combnation (9 directions + 2 button states)



- Reward change in score for the step (most Atari games had general scores constantly rewarding actions)

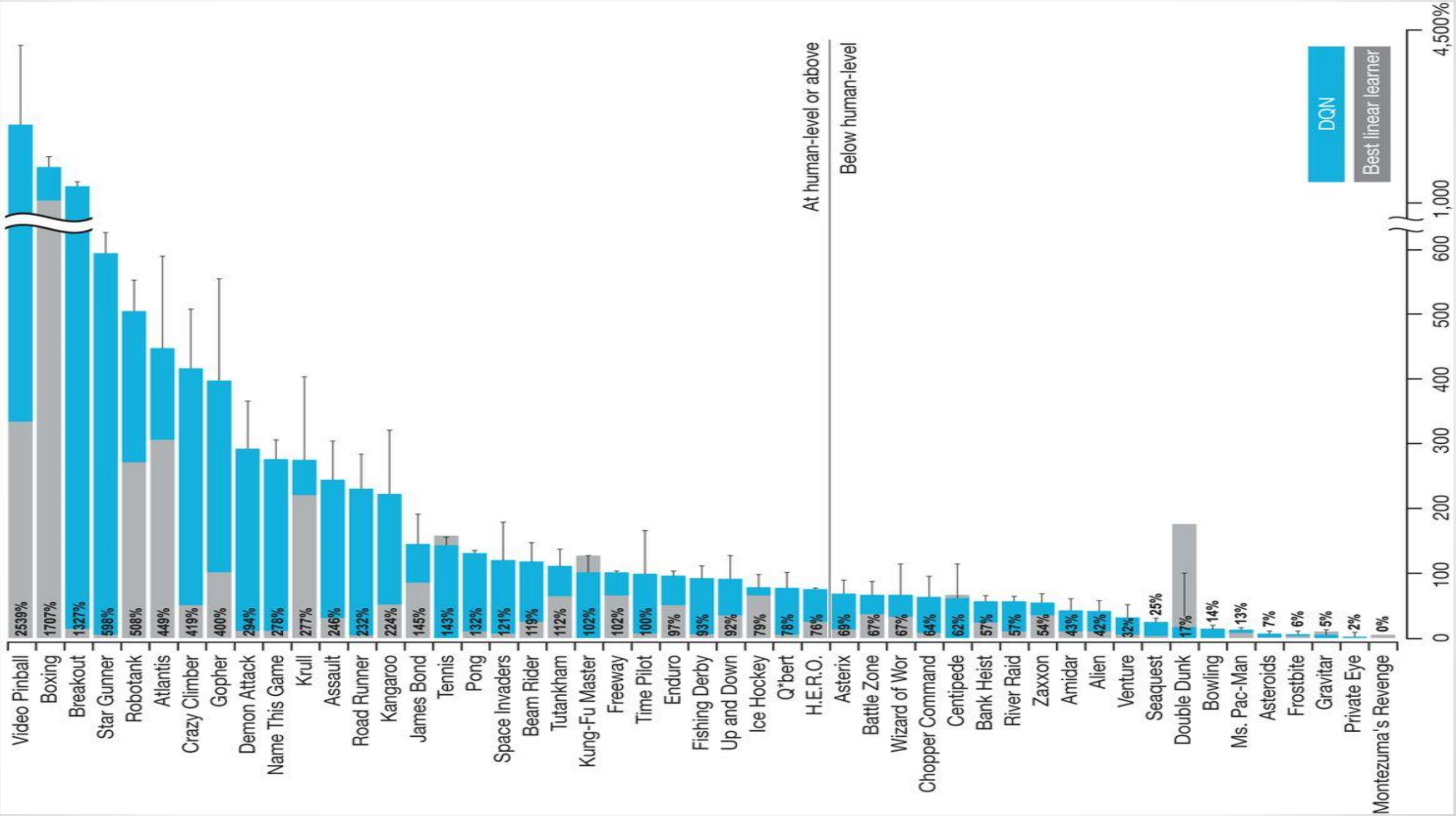
DQN on Atari Games Network Architecture



https://media.nature.com/lw926/nature-assets/nature/journal/v518/n7540/images/nature14236-f1.jpg

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529--533.

DQN Results in Atari



Advantages replay buffer and fixed targets

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99

Lecture notes D. Silver: Introduction to Reinforcement Learning (https://www.davidsilver.uk/wp-content/uploads/2020/03/FA.pdf)

Lecture 6: Function Approximation (Slide 41)

Literature

- Lecture notes D. Silver: Introduction to Reinforcement Learning (https://www.davidsilver.uk/wp-content/uploads/2020/03/FA.pdf)

- S. Russel, P. Norvig: Artificial Intelligence: A modern Approach, Pearson, 3rd edition, 2016

- R. S. Sutton, A. G. Barto: **Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning),** The MIT Press; Auflage: 2., 2018

- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529--533.