Off-policy & control with approximation

Herke van Hoof

Today’s chapter

This week will be a tricky subject - don’t stress out if you don’t understand everything. From next lecture, we’ll start on a new topic, starting a bit easier.

That said, do make use of the recourses (exercises, book, slides, tutorial sessions) to review the material as necessary.

Control with approximation

So far, we've looked at learning v functions with approximation

Let's try the same strategy for learning q functions!

We'll focus on **episodic** tasks
(Continuing tasks requires a couple more modifications)

Episodic semi-gradient Sarsa

As before we can define the semi-gradient of mean value error

$$\left(U_t - \hat{q}\left(S_t, A_t, \mathbf{w}_t\right)\right) \nabla \hat{q}\left(S_t, A_t, \mathbf{w}_t\right)$$

with U a target value, e.g. Sarsa target.

We are still on-policy, so we iterate

- Making a step according to current policy π
- Improve Q^{π} estimate
- Improve π to soft approximation of greedy policy (e.g. ε-greedy)

Episodic semi-gradient Sarsa

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

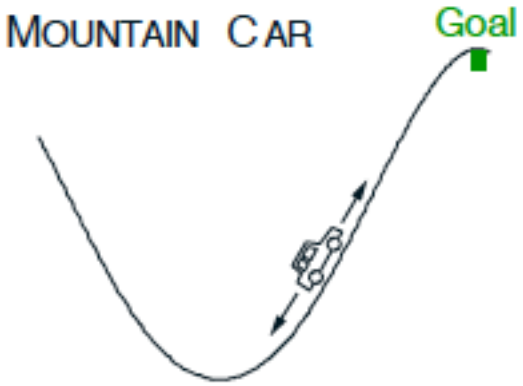
Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

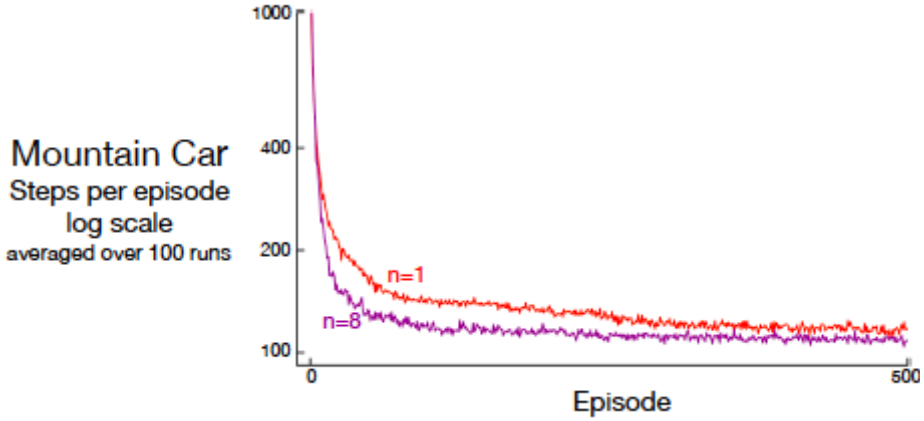
Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy) (Implicit policy update)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

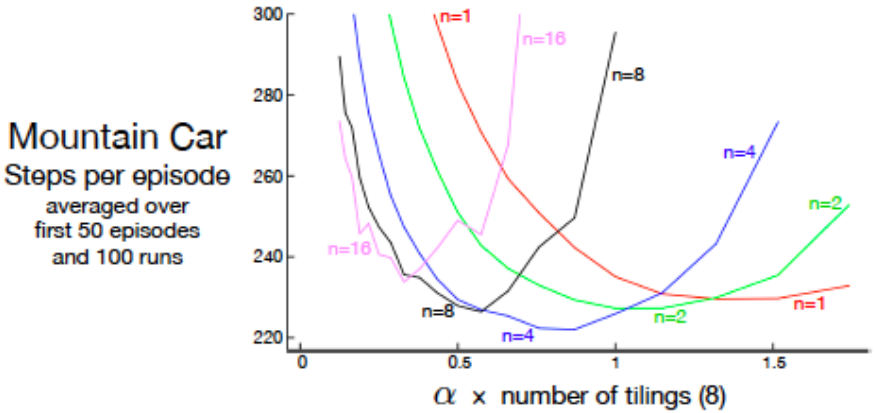
Figures: Sutton & Barto. RL:AI

Episodic semi-gradient Sarsa

As before, we can use n-step Sarsa as well







Figures: Sutton & Barto. RL:AI

Off-policy prediction with approximation

That was easy so far!

How about off-policy?

Remember importance weights $\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

Sample ‘too rare’: high weight

Sample ‘too common’: low weight

Off-policy prediction with approximation

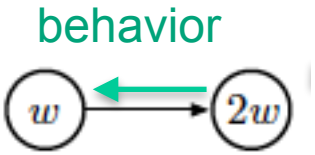
Use importance weight in semi-gradient TD(0)

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \rho_t \underbrace{\left(R_{t+1} + \gamma \hat{v}\left(S_{t+1}, \mathbf{w}_t\right) - \hat{v}\left(S_t, \mathbf{w}_t\right)\right)}_{\delta_t} \nabla \hat{v}\left(S_t, \mathbf{w}_t\right)$$

Let's try it!

Off-policy prediction with approximation

Initial w is 10



On-policy, γ=1

From left to right:

$$w_{t+1} = w_t + \alpha(2w_t - w_t)\nabla\hat{v}$$
$$= w_t + \alpha(2-1)w_t$$

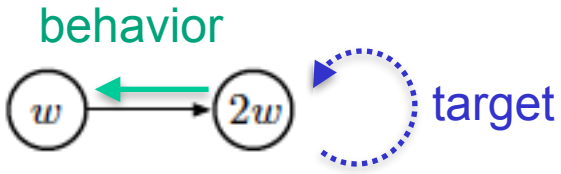
Right to left:

$$w_{t+1} = w_t + \alpha(w_t - 2w_t)\nabla\hat{v}$$
$$= w_t + \alpha(1-2)w_t \cdot 2$$

TDE tends to shrink!

Off-policy prediction with approximation

Initial w is 10



On-policy, γ=1

From left to right:

$$w_{t+1} = w_t + \alpha(2w_t - w_t)\nabla\hat{v}$$
$$= w_t + \alpha(2-1)w_t$$

Right to left:

$$w_{t+1} = w_t + \alpha(w_t - 2w_t)\nabla\hat{v}$$
$$= w_t + \alpha(1-2)w_t \cdot 2$$

TDE tends to shrink!

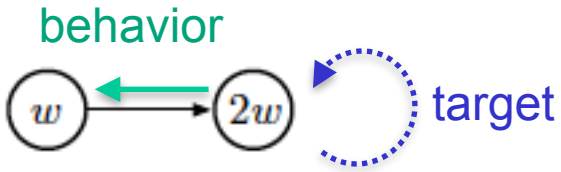
Off-policy, γ=1

From left to right:

Right to left:

Off-policy prediction with approximation

Initial w is 10



On-policy, γ=1

From left to right:

$$w_{t+1} = w_t + \alpha(2w_t - w_t)\nabla\hat{v}$$
$$= w_t + \alpha(2-1)w_t$$

Right to left:

$$w_{t+1} = w_t + \alpha(w_t - 2w_t)\nabla\hat{v}$$
$$= w_t + \alpha(1-2)w_t \cdot 2$$

TDE tends to shrink!

Off-policy, γ=1

From left to right: =1

$$w_{t+1} = w_t + \alpha\rho(2w_t - w_t)\nabla\hat{v}$$
$$= w_t + \alpha(2-1)w_t$$

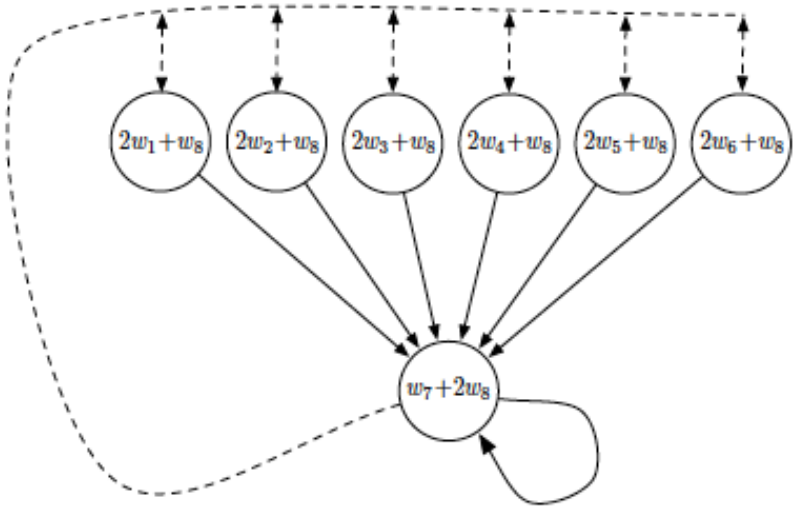
Right to left: =0

$$w_{t+1} = w_t + \alpha\rho(w - 2w)\nabla\hat{v}$$

If ρ<<1, could TDE keep increasing?

Off-policy prediction with approximation

Yes, the TDE can keep increasing

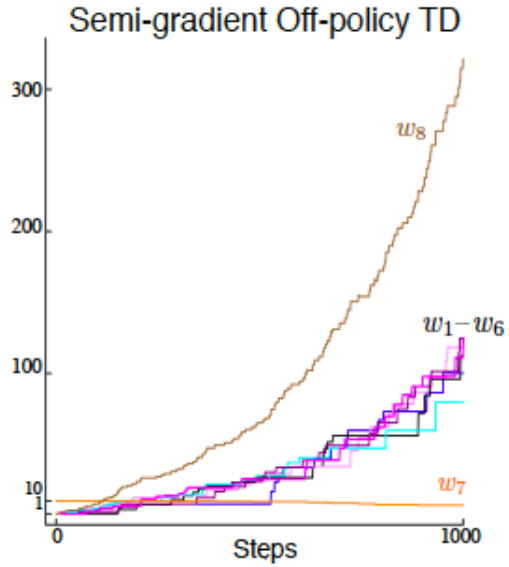


$\pi(\text{solid}|\cdot) = 1$

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

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

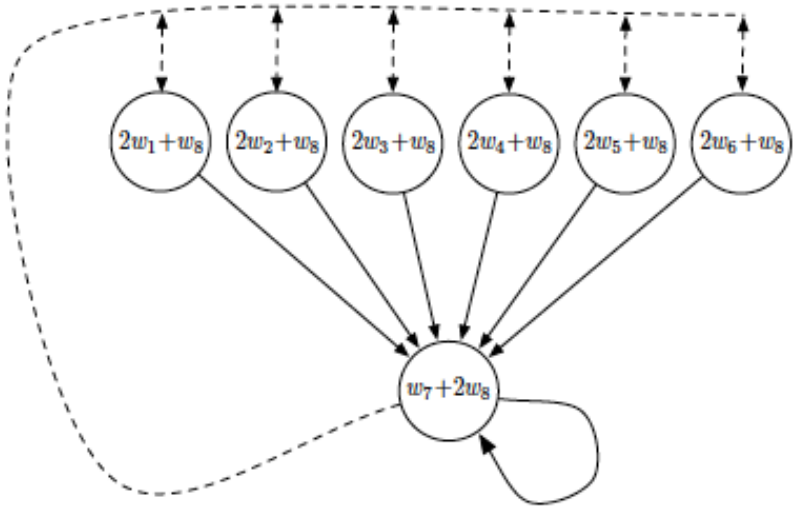
$\gamma = 0.99$



Figures: Sutton & Barto. RL:AI
Reinforcement Learning

Off-policy prediction with approximation

Yes, the TDE can keep increasing

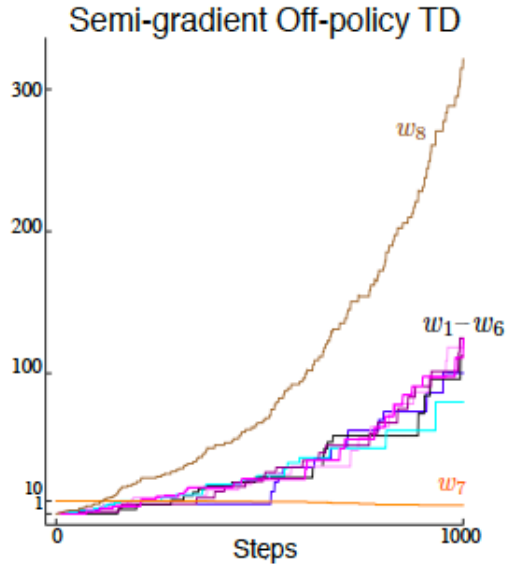


$\pi(\text{solid}|\cdot) = 1$

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

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

$\gamma = 0.99$



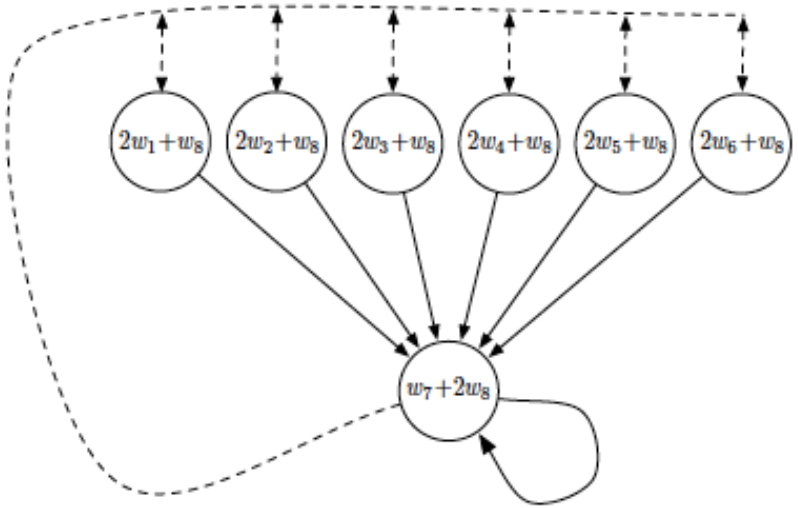
Effect is not because

- V not representable
- Random effects
- Dependent features

Figures: Sutton & Barto. RL:AI

Off-policy prediction with approximation

Yes, the TDE can keep increasing

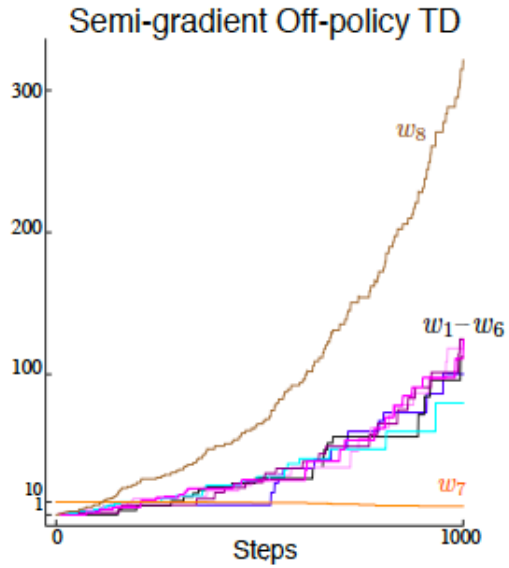


$\pi(\text{solid}|\cdot) = 1$

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

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

$\gamma = 0.99$



Effect is not because

- V not representable
- Random effects
- Dependent features

But 'deadly triad':

- Function approximation
- *Semi-gradient* bootstrapping
- Off-policy training

Figures: Sutton & Barto. RL:AI

Off-policy prediction with approximation

Can we make semi-gradients work?

Get rid of function approximation?

- Cannot scale to large / continuous domains

Get rid of bootstrapping?

- MC is much slower!

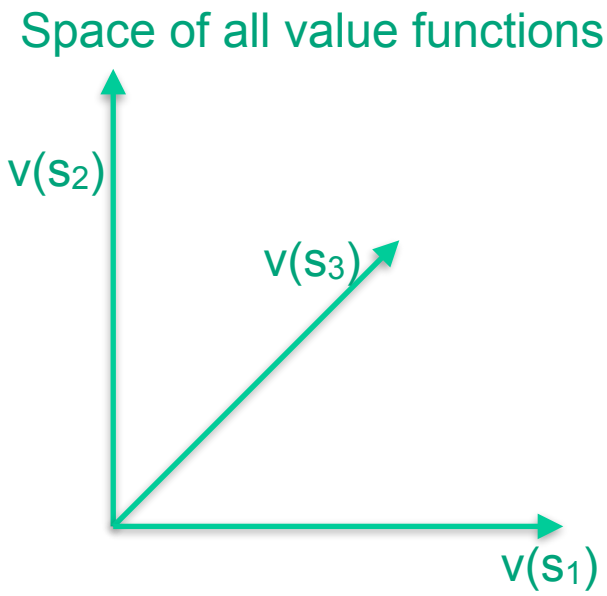
Get rid of off-policy learning?

- Possible, but…
- We would like to learn different policies at same time (e.g. running, walking, standing, all from same behavior)
- Also: reuse old data, learn from recorded data, …

Alternatives to semi-gradients?

3 states

True value
($v(s_1)$, $v(s_2)$, $v(s_3)$)
Represented as point
in 3D space



Approximated value
$\hat{v}=w_1x_1(s) + w_2x_2(s)$

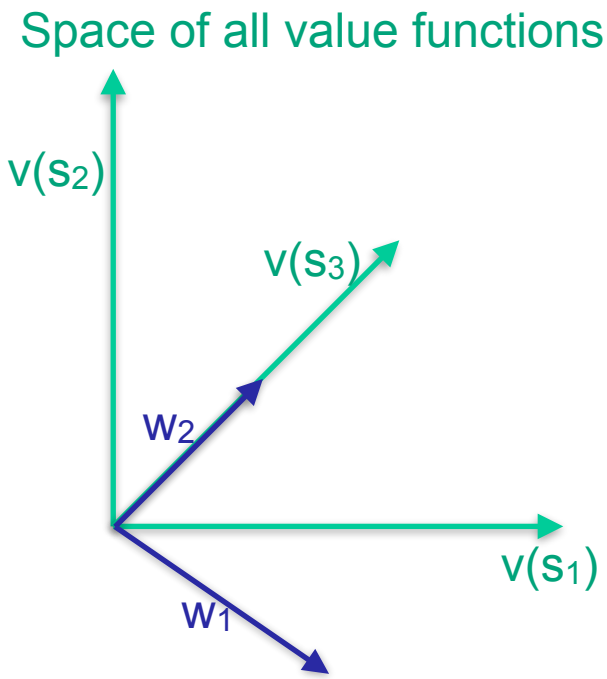
- $x(s_1)=[1,0]$
- $x(s_2)=[-0.5,0]$
- $x(s_3)=[0,1]$

Represent as point in 2D subspace

Alternatives to semi-gradients?

3 states

True value
($v(s_1)$, $v(s_2)$, $v(s_3)$)
Represented as point
in 3D space



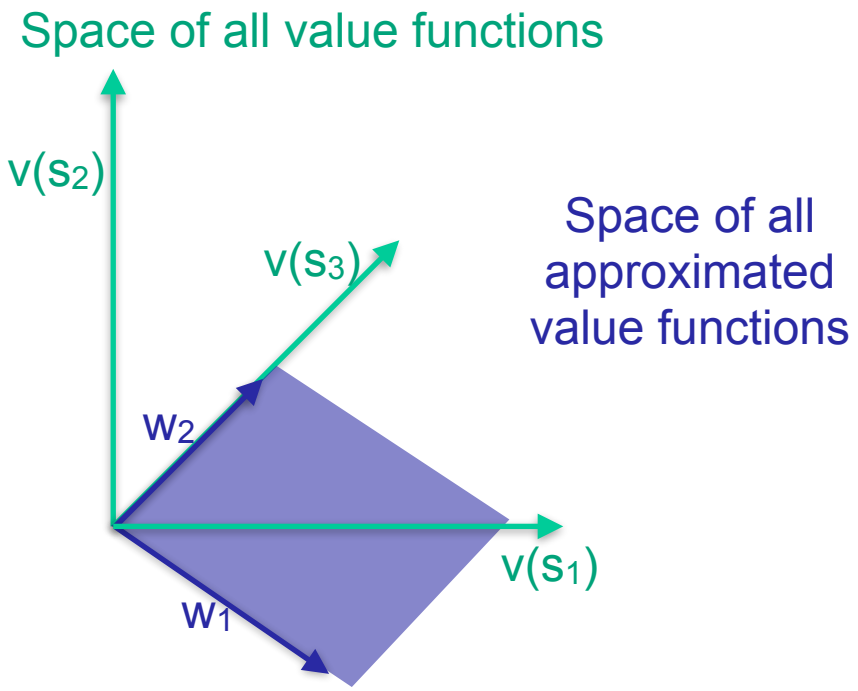
Approximated value
$\hat{v}=w_1x_1(s) + w_2x_2(s)$

- $x(s_1)=[1,0]$
- $x(s_2)=[-0.5,0]$
- $x(s_3)=[0,1]$

Alternatives to semi-gradients?

3 states

True value
$(v(s_1), v(s_2), v(s_3))$
Represented as point
in 3D space



Approximated value
$\hat{v}=w_1x_1(s) + w_2x_2(s)$

- $x(s_1)=[1,0]$
- $x(s_2)=[-0.5,0]$
- $x(s_3)=[0,1]$

Represent as point in 2D subspace

Alternatives to semi-gradients?

Without approximation, could keep applying Bellman operator

$$B_\pi v_{\mathbf{w}} = v_{\mathbf{w}} + \overline{\delta}_{\mathbf{w}}$$

$$\overline{\delta}_{\mathbf{w}}(s) = \mathbb{E}_\pi[\underbrace{R_{t+1} + \gamma v_{\mathbf{w}}\left(S_{t+1}\right) - v_{\mathbf{w}}\left(S_t\right)}_{\delta_{\mathbf{w}}(S_t, A_t, S_{t+1})} | S_t = s, A_t \sim \pi]$$

Figure: Sutton & Barto. RL:AI

Alternatives to semi-gradients?

Without approximation, could keep applying Bellman operator

$$B_\pi v_{\mathbf{w}} = v_{\mathbf{w}} + \overline{\delta}_{\mathbf{w}}$$

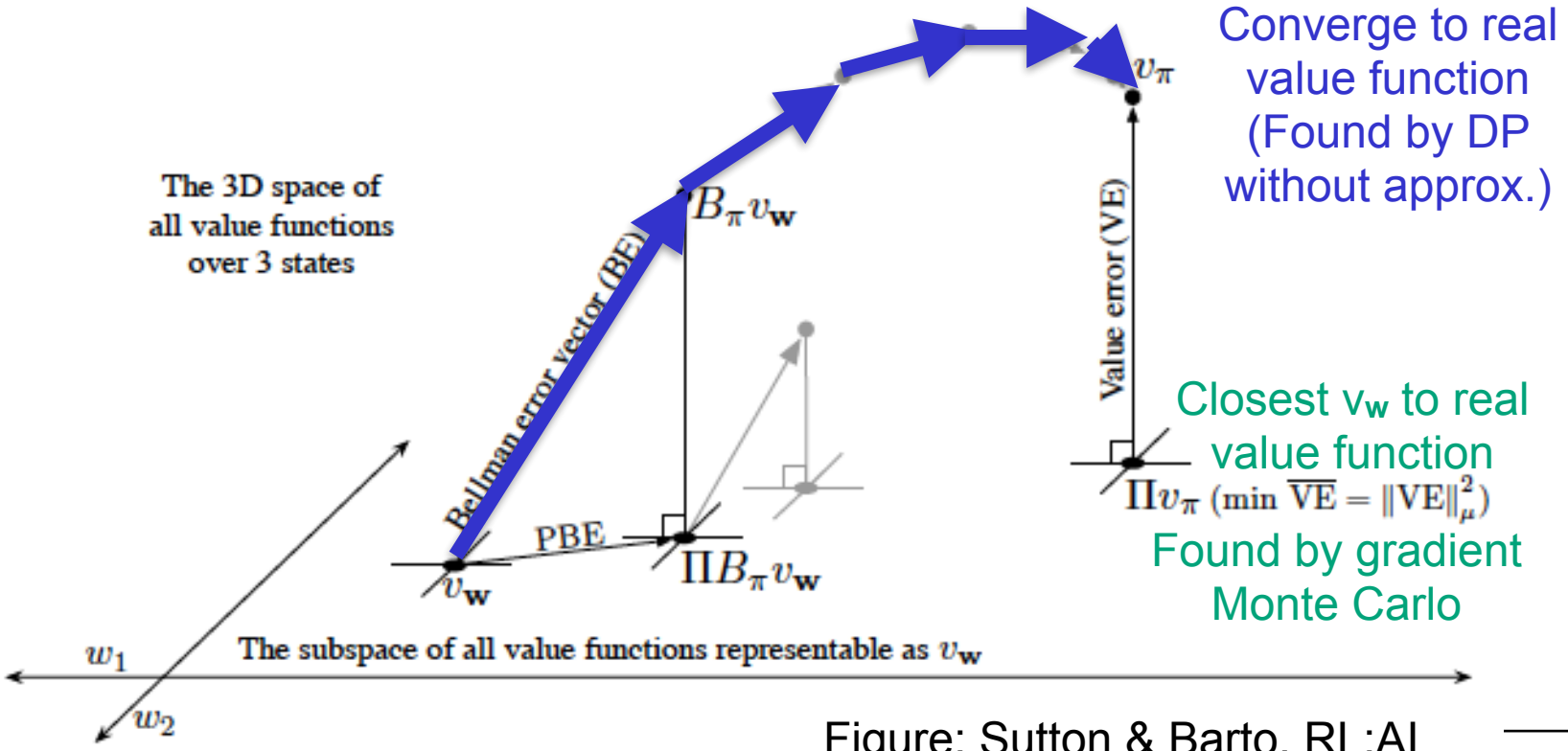
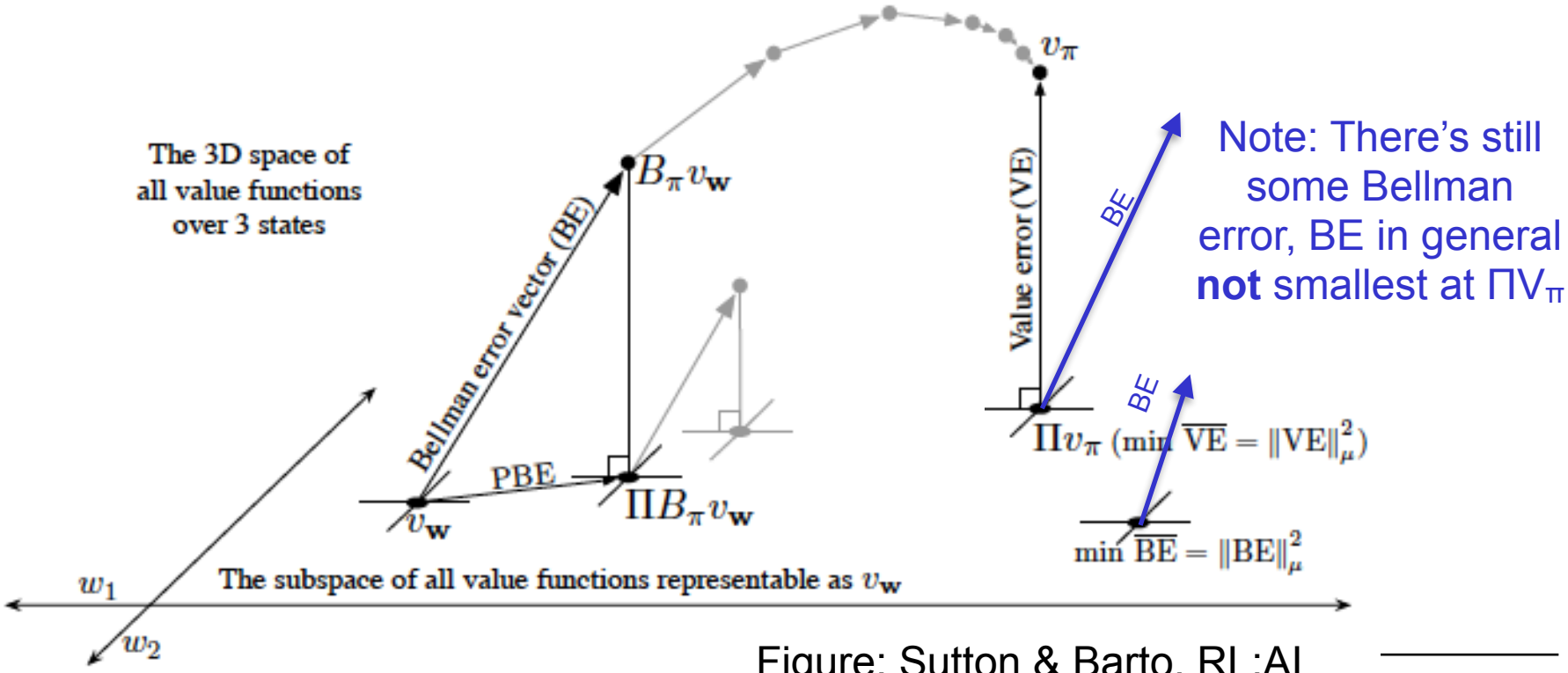


Figure: Sutton & Barto. RL:AI

Alternatives to semi-gradients?

Without approximation, could keep applying Bellman operator

$$B_\pi v_{\mathbf{w}} = v_{\mathbf{w}} + \overline{\delta}_{\mathbf{w}}$$



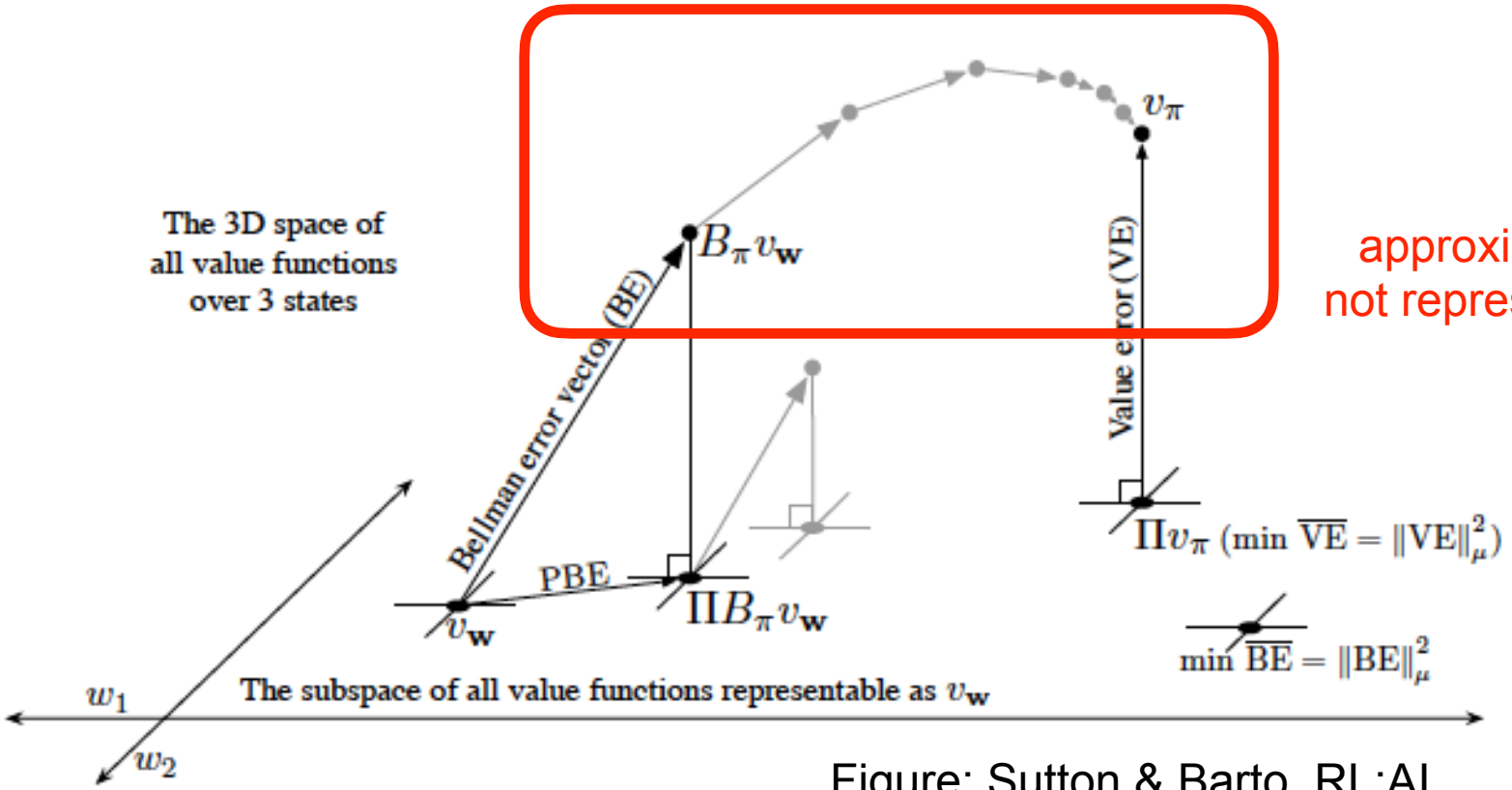
Note: There's still some Bellman error, BE in general **not** smallest at ΠV_π

Figure: Sutton & Barto. RL:AI

Alternatives to semi-gradients?

Without approximation, could keep applying Bellman operator

$$B_\pi v_{\mathbf{w}} = v_{\mathbf{w}} + \overline{\delta}_{\mathbf{w}}$$



approximation: not representable!

Figure: Sutton & Barto. RL:AI

Alternatives to semi-gradients?

Update according to Bellman error?

$$B_\pi v_{\mathbf{w}} = v_{\mathbf{w}} + \overline{\delta}_{\mathbf{w}}$$

Π: Closest alternative to desired update

In norm:

$$\|\Delta v\|^2_\mu \doteq \sum_{s \in S} \mu(s) \Delta v(s)^2$$

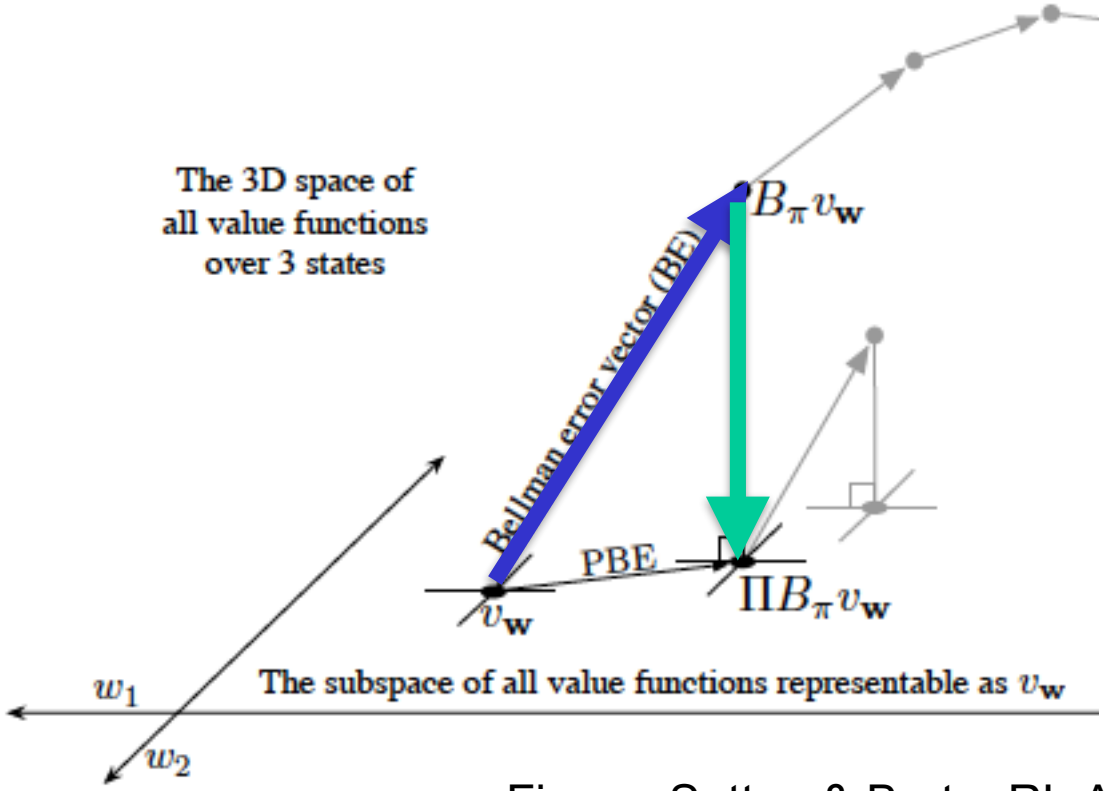


Figure: Sutton & Barto. RL:AI

Alternatives to semi-gradients?

With approximation, need to stay within representable space

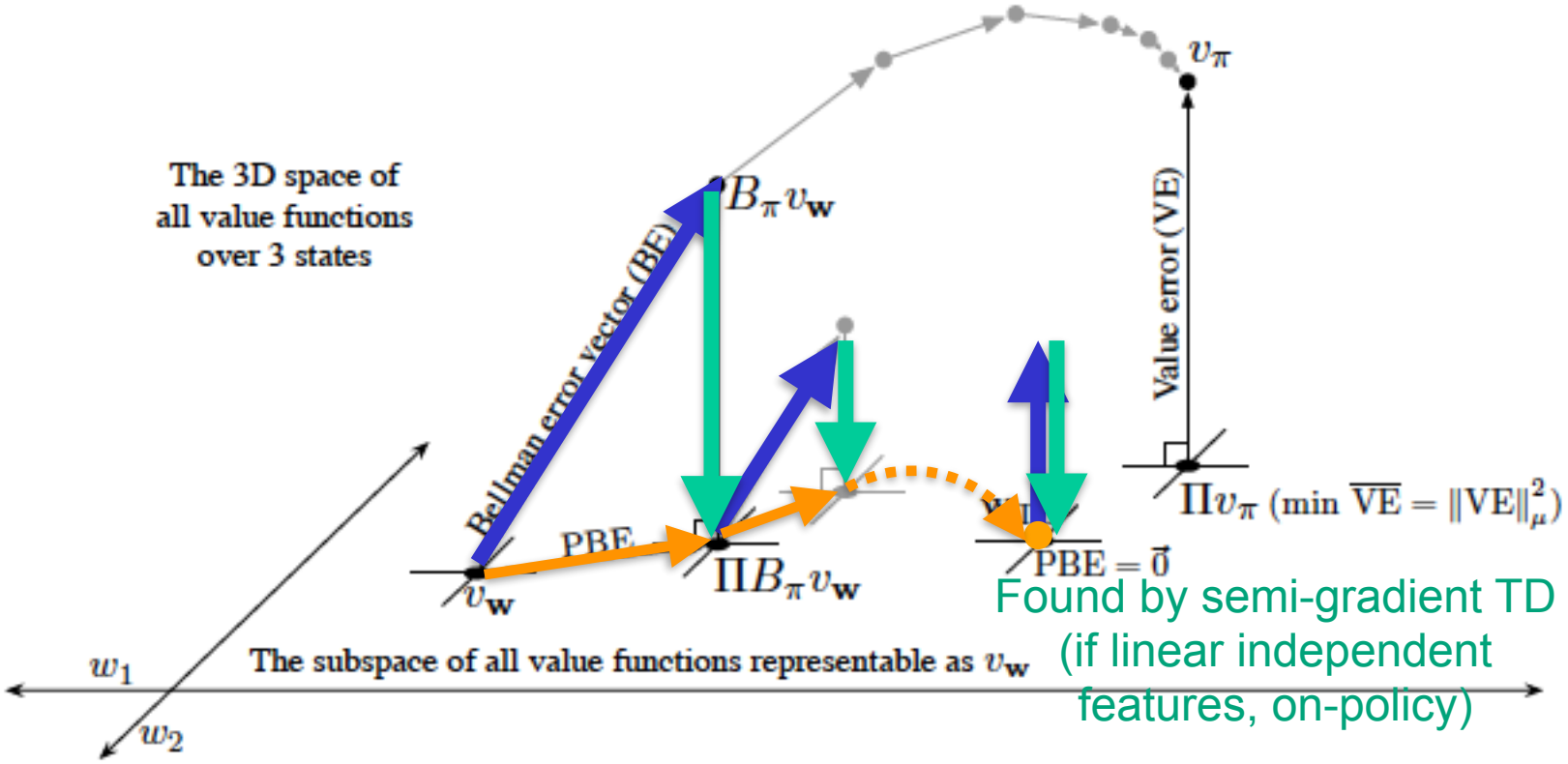
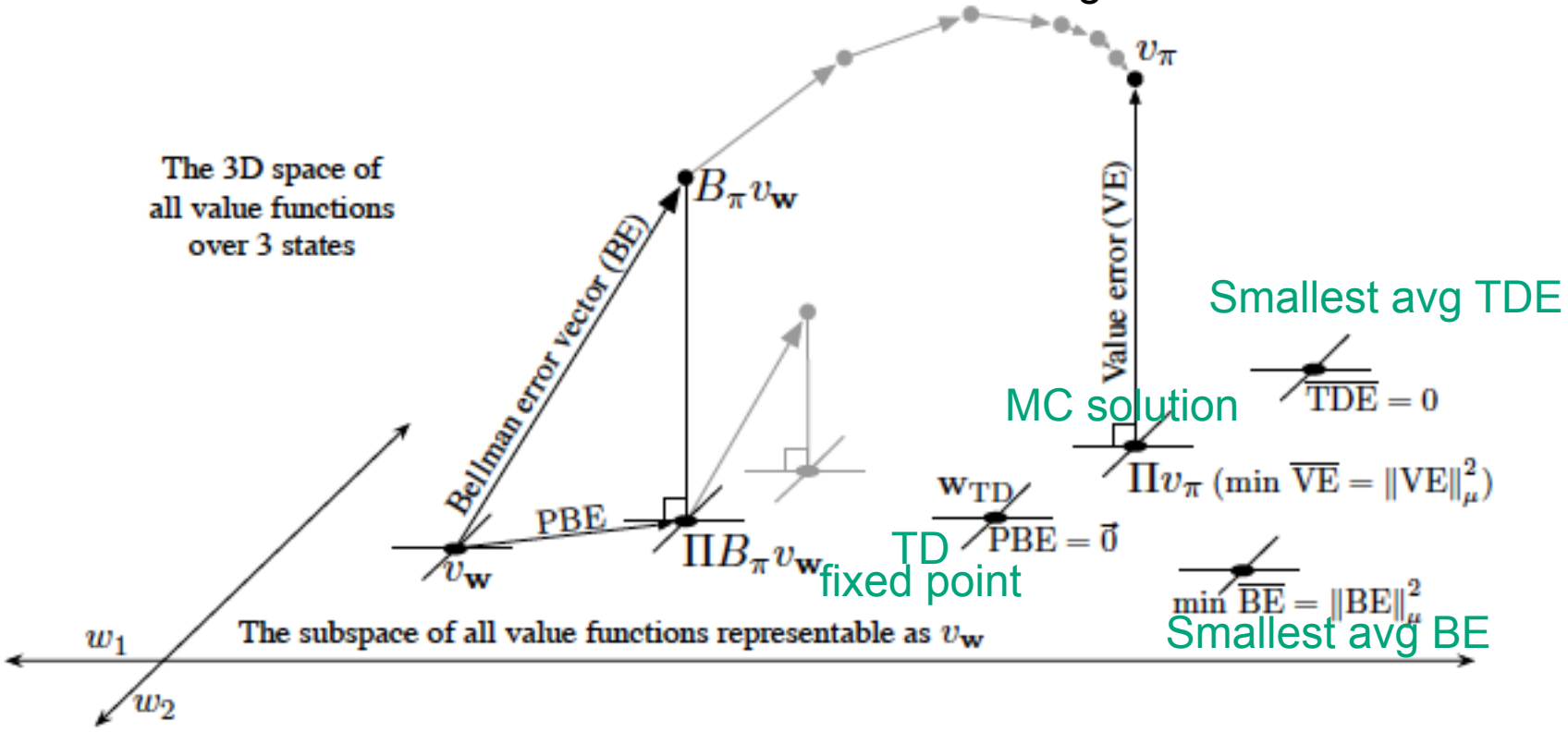


Figure: Sutton & Barto. RL:AI

Alternatives to semi-gradients?

With approximation, need to stay within representable space

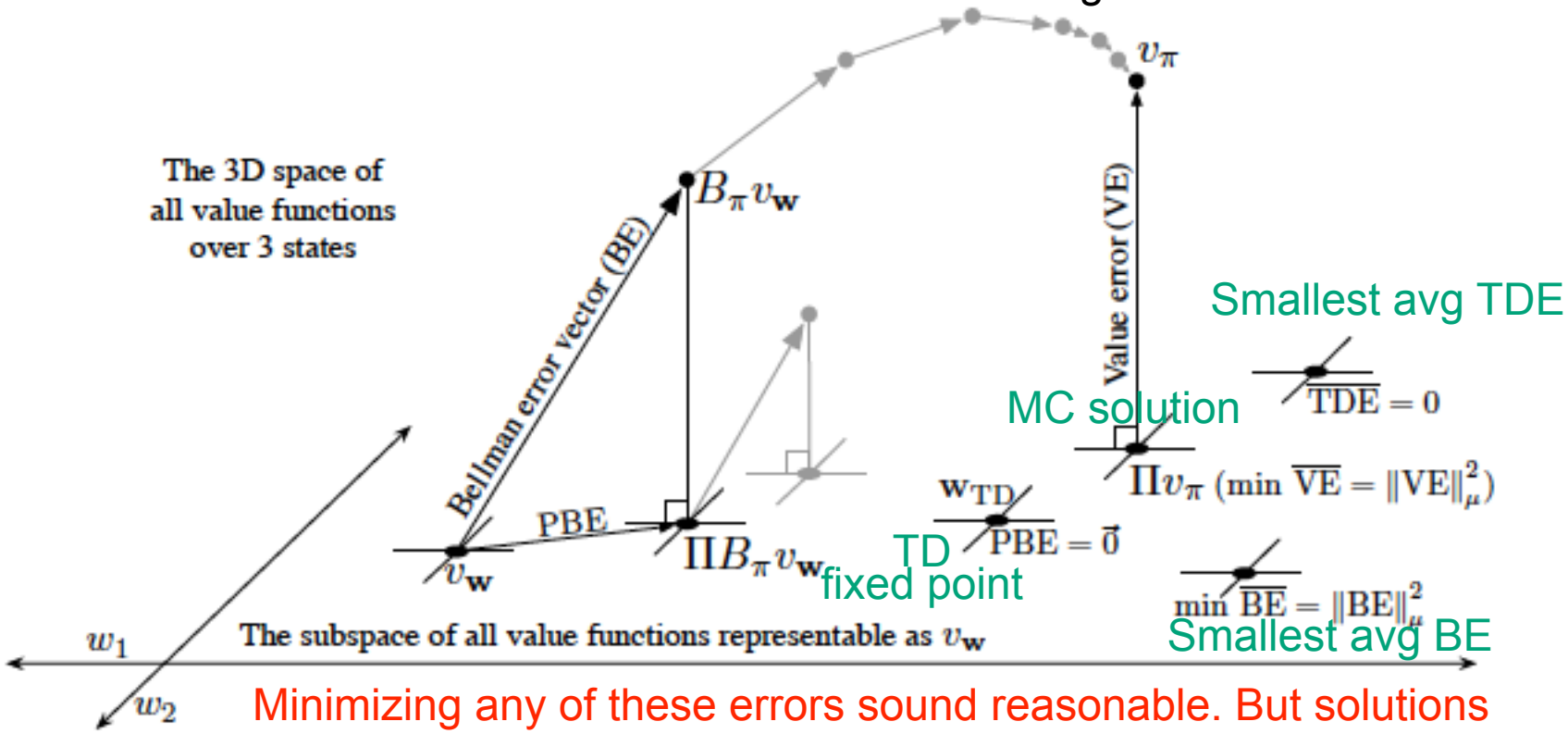
Figure: Sutton & Barto. RL:AI



Alternatives to semi-gradients?

With approximation, need to stay within representable space

Figure: Sutton & Barto. RL:AI



Minimizing any of these errors sound reasonable. But solutions are different in general! Which should we choose?

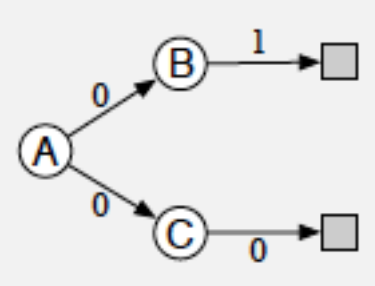
Minimize mean squared TD error?

$$\begin{aligned}\overline{\mathrm{TDE}}(\mathbf{w}) &= \sum_{s \in S} \mu(s) \mathbb{E}\left[\delta_t^2 | S_t = s, A_t \sim \pi\right] \\ &= \sum_{s \in S} \mu(s) \mathbb{E}\left[\rho_t \delta_t^2 | S_t = s, A_t \sim b\right] \\ &= \mathbb{E}_b\left[\rho_t \delta_t^2\right].\end{aligned}$$

We can estimate the real gradient of the TDE from data!

How does it perform?

Minimize mean squared TD error?



A-split example

- What value minizes TD error at A?
- Now, what value minimises TD error at B, C?

Figure: Sutton & Barto. RL:AI

Minimize mean squared Bellman error?

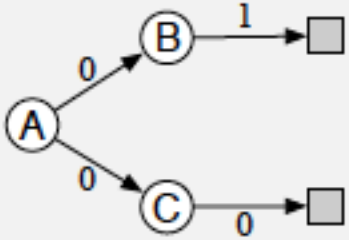
Let's try something else

$$\overline{\mathrm{BE}}(\mathbf{w}) = \left\|\overline{\delta}_{\mathbf{w}}\right\|_{\mu}^{2}$$

$$\overline{\delta}_{\mathbf{w}}(s) = \mathbb{E}_{\pi}[\underbrace{R_{t+1} + \gamma v_{\mathbf{w}}\left(S_{t+1}\right) - v_{\mathbf{w}}\left(S_{t}\right)}_{\delta_{\mathbf{w}}(S_t, A_t, S_{t+1})} | S_t = s, A_t \sim \pi]$$

Least Bellman error is acceptable solution in A-split problem

However, the gradient cannot be estimated from data only!



Mean squared projected Bellman error?

PBE last chance for bootstrapping!

- Linear approximation: PBE=0 at w_{TD}
- On-policy, semi-gradient finds w_{TD}
- LSTD find TD fixed point
- Find TD fixed point off-policy?

  Let's minimize PBE used gradient descent algorithm

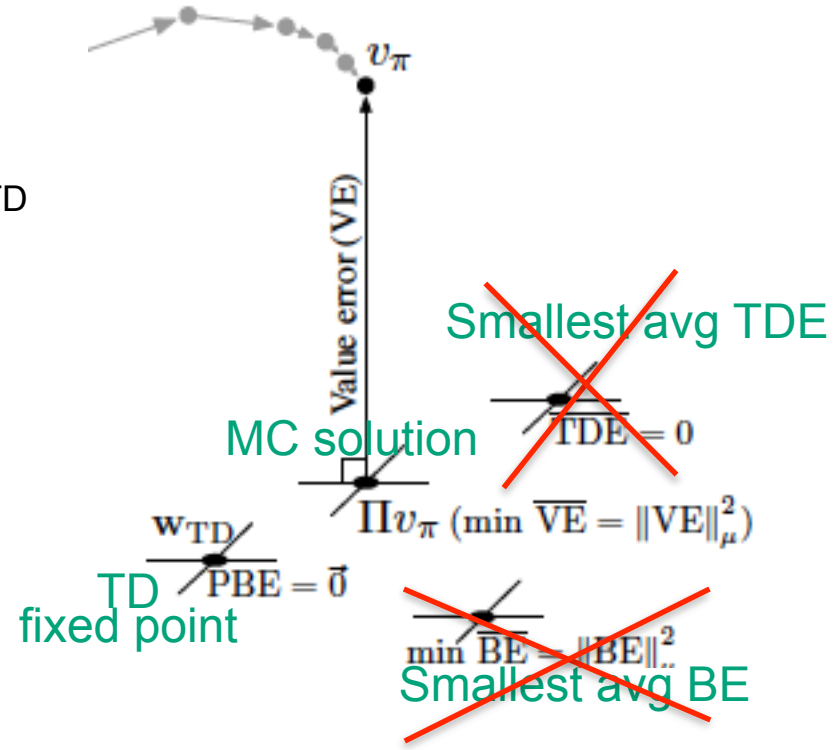


Figure: Sutton & Barto. RL:AI

Mean squared projected Bellman error?

$$\mathbf{D} = \begin{bmatrix} \mu(s_1) & 0 & \dots & 0 \\ 0 & \mu(s_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mu(s_n) \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} x_1(s_1) & \dots & x_m(s_1) \\ x_1(s_2) & \dots & x_m(s_2) \\ \vdots & \ddots & \vdots \\ x_1(s_n) & \dots & x_m(s_n) \end{bmatrix}$$

$$\begin{aligned} \overline{\mathrm{PBE}}(\mathbf{w}) &= \left\| \Pi \bar{\delta}_{\mathbf{w}} \right\|_{\mu}^2 \\ &= \left(\Pi \bar{\delta}_{\mathbf{w}} \right)^\top \mathbf{D} \Pi \bar{\delta}_{\mathbf{w}} \\ &\vdots \\ &= \left(\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}} \right)^\top \left(\mathbf{X}^\top \mathbf{D} \mathbf{X} \right)^{-1} \left(\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}} \right) \end{aligned}$$

Mean squared projected Bellman error?

$$\nabla \overline{\mathrm{PBE}} = \nabla \left(\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}}\right)^\top \left(\mathbf{X}^\top \mathbf{D} \mathbf{X}\right)^{-1} \left(\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}}\right)$$
$$= 2 \left(\nabla \mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}}\right)^\top \left(\mathbf{X}^\top \mathbf{D} \mathbf{X}\right)^{-1} \left(\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}}\right)$$

Not used to multivariate calculus? Compare to scalar case:

$$\nabla f(\boldsymbol{w}) c f(\boldsymbol{w}) = 2(\nabla f(\boldsymbol{w})) c f(\boldsymbol{w})$$

(using the product rule)

(the 'matrix cookbook' gives a thorough explanation, https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf)

Mean squared projected Bellman error?

$$\nabla \overline{\mathrm{PBE}}(\mathbf{w}) = 2\nabla \left[\mathbf{X}^\top \mathbf{D}\bar{\delta}_{\mathbf{w}}\right]^\top \left(\mathbf{X}^\top \mathbf{D}\mathbf{X}\right)^{-1} \left(\mathbf{X}^\top \mathbf{D}\bar{\delta}_{\mathbf{w}}\right)$$

Mean squared projected Bellman error?

$$\nabla \overline{\text{PBE}}(\mathbf{w}) = 2\nabla \left[\mathbf{X}^\top \mathbf{D}\bar{\delta}_\mathbf{w}\right]^\top \left(\mathbf{X}^\top \mathbf{D}\mathbf{X}\right)^{-1} \left(\mathbf{X}^\top \mathbf{D}\bar{\delta}_\mathbf{w}\right)$$

$$= \sum_s \mu(s)\mathbf{x}(s)\bar{\delta}_\mathbf{w}(s)$$

$$= \mathbb{E}_{s_t}\mathbf{x}_t\mathbb{E}_{a_t\sim\pi}\mathbb{E}_{s_{t+1}}\delta_t(s_t, a_t, s_{t+1})$$

$$= \mathbb{E}_{s_t}\mathbf{x}_t\mathbb{E}_{a_t\sim b}\rho_t\mathbb{E}_{s_{t+1}}\delta_t(s_t, a_t, s_{t+1})$$

$$= \mathbb{E}\left[\rho_t\delta_t\mathbf{x}_t\right]$$

$$\nabla\mathbb{E}\left[\rho_t\delta_t\mathbf{x}_t\right]^\top = \mathbb{E}\left[\rho_t\nabla\delta_t^\top\mathbf{x}_t^\top\right]$$

$$= \mathbb{E}\left[\rho_t\nabla\left(R_{t+1} + \gamma\mathbf{w}^\top\mathbf{x}_{t+1} - \mathbf{w}^\top\mathbf{x}_t\right)^\top\mathbf{x}_t^\top\right]$$

$$= \mathbb{E}\left[\rho_t\left(\gamma\mathbf{x}_{t+1} - \mathbf{x}_t\right)\mathbf{x}_t^\top\right]$$

Mean squared projected Bellman error?

$$\nabla \overline{\mathrm{PBE}}(\mathbf{w}) = 2\nabla \left[\mathbf{X}^\top \mathbf{D}\bar{\delta}_\mathbf{w}\right]^\top \left(\mathbf{X}^\top \mathbf{D}\mathbf{X}\right)^{-1} \left(\mathbf{X}^\top \mathbf{D}\bar{\delta}_\mathbf{w}\right)$$

$$= \mathbb{E}\left[\rho_t \delta_t \mathbf{x}_t\right]$$

$$\mathbf{X}^\top \mathbf{D}\mathbf{X} = \sum_s \mu(s)\mathbf{x}_s \mathbf{x}_s^\top = \mathbb{E}\left[\mathbf{x}_t \mathbf{x}_t^\top\right]$$

$$= \mathbb{E}\left[\rho_t \left(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t\right) \mathbf{x}_t^\top\right]$$

Can we just plug in a new samples (s,a,s') tuple in each of these?

Mean squared projected Bellman error?

$$\nabla \overline{\mathrm{PBE}}(\mathbf{w}) = 2\mathbb{E}\left[\rho_t \left(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t\right) \mathbf{x}_t^\top\right] \mathbb{E}\left[\mathbf{x}_t \mathbf{x}_t^\top\right]^{-1} \mathbb{E}\left[\rho_t \delta_t \mathbf{x}_t\right]$$

The first and last factor are dependent - both depend on x_{t+1}

Why a problem? Example: consider r.v. $Y \in -1,1$ uniform

$$\mathbb{E}[Y]\mathbb{E}[Y] \quad =? \qquad \sum_{i=1}^{N} Y_i Y_i \quad =?$$

Mean squared projected Bellman error?

$$\nabla \overline{\mathrm{PBE}}(\mathbf{w}) = 2\mathbb{E}\left[\rho_t \left(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t\right) \mathbf{x}_t^\top\right] \mathbb{E}\left[\mathbf{x}_t \mathbf{x}_t^\top\right]^{-1} \mathbb{E}\left[\rho_t \delta_t \mathbf{x}_t\right]$$

The first and last factor are dependent - both depend on x_{t+1}

Learn some factors from all data, only plug in sample for rest

Mean squared projected Bellman error?

$$\nabla \overline{\mathrm{PBE}}(\mathbf{w}) = 2\mathbb{E}\left[\rho_t \left(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t\right) \mathbf{x}_t^\top\right] \underbrace{\mathbb{E}\left[\mathbf{x}_t \mathbf{x}_t^\top\right]^{-1} \mathbb{E}\left[\rho_t \delta_t \mathbf{x}_t\right]}_{\mathbf{v}}$$

The first and last factor are dependent - both depend on x_{t+1}

Learn some factors from all data, only plug in sample for rest

Mean squared projected Bellman error?

$$\nabla\overline{\mathrm{PBE}}(\mathbf{w}) = 2\mathbb{E}\left[\rho_t\left(\gamma\mathbf{x}_{t+1} - \mathbf{x}_t\right)\mathbf{x}_t^\top\right]\underbrace{\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^\top\right]^{-1}\mathbb{E}\left[\rho_t\delta_t\mathbf{x}_t\right]}_{\mathbf{v}}$$

The first and last factor are dependent - both depend on x_{t+1}

Learn some factors from all data, only plug in sample for rest

v is the solution to a least squares problem - for such problems there is a SGD algorithm: $\mathbf{v}_{t+1} \doteq \mathbf{v}_t + \beta\rho_t\left(\delta_t - \mathbf{v}_t^\top\mathbf{x}_t\right)\mathbf{x}_t$

Plug in the parts:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \alpha/2\nabla\overline{\mathrm{PBE}}\left(\mathbf{w}_t\right)\\ &\approx \mathbf{w}_t + \alpha\mathbb{E}\left[\rho_t\left(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\right)\mathbf{x}_t^\top\right]\mathbf{v}_t\\ &\approx \mathbf{w}_t + \alpha\rho_t\left(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\right)\mathbf{x}_t^\top\mathbf{v}_t\end{aligned}$$

Mean squared projected Bellman error?

This is GTD2, a gradient-TD method

- Converges to 0 mean squared projected Bellman error (linear features)
- With non-linear features, converges to local optimum
- There are other, more sophisticated gradient-TD methods

- Cost?
- Additional step size parameter for learning **v** (‘larger’ than α)
- Need to store and update two parameter vectors

Convergence with function approximation

	Tabular On/Off	Linear on **	Nonlinear on	Linear off **	Nonlinear off
Gradient MC *					
Semi-gradient TD *			No C!	No C!	No C!
Gradient TD *					
LSTD			N.A.		N.A.

* with appropriate step-size schedule

** if features independent, single solution

Convergence with function approximation

	Tabular On/Off	Linear on **	Nonlinear on	Linear off **	Nonlinear off
Gradient MC *					
Semi-gradient TD *				No C!	No C!
Gradient TD *					
LSTD			N.A.		N.A.

Local or global convergence?

* with appropriate step-size schedule

** if features independent, single solution

Convergence with function approximation

	Tabular On/Off	Linear on **	Nonlinear on	Linear off **	Nonlinear off
Gradient MC *	global	global	non-linear: local	global	non-linear: local
Semi-gradient TD *			No C!	No C!	No C!
Gradient TD *					
LSTD			N.A.		N.A.

* with appropriate step-size schedule

** if features independent, single solution

Convergence with function approximation

	Tabular On/Off	Linear on **	Nonlinear on	Linear off **	Nonlinear off
Gradient MC *					
Semi-gradient TD *			!	No C!	No C!
Gradient TD *					
LSTD			N.A.		N.A.

Convergence to minimum of which error?

* with appropriate step-size schedule

** if features independent, single solution

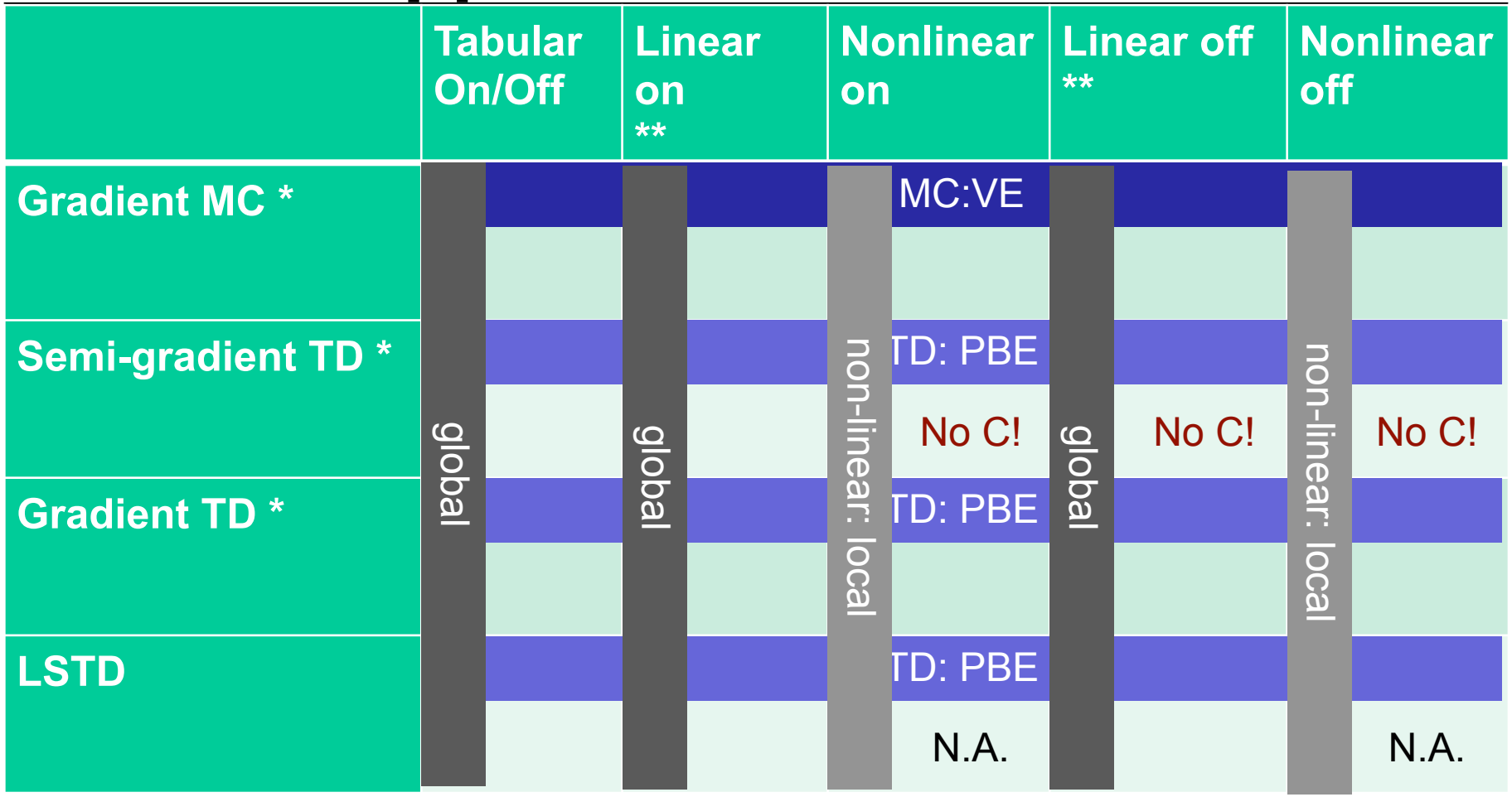
Convergence with function approximation

	Tabular On/Off	Linear on **	Nonlinear on	Linear off **	Nonlinear off
Gradient MC *	MC:VE				
Semi-gradient TD *	TD: PBE		No C!	No C!	No C!
Gradient TD *	TD: PBE				
LSTD	TD: PBE		N.A.		N.A.

* with appropriate step-size schedule

** if features independent, single solution

Convergence with function approximation



* with appropriate step-size schedule

** if features independent, single solution

Deep Q networks

The famous algorithm for learning to play Atari games: Deep Q networks (DQN)

Illustrates some of the challenges in doing control with (non-linear) function approximation

Figure: Deepmind.com

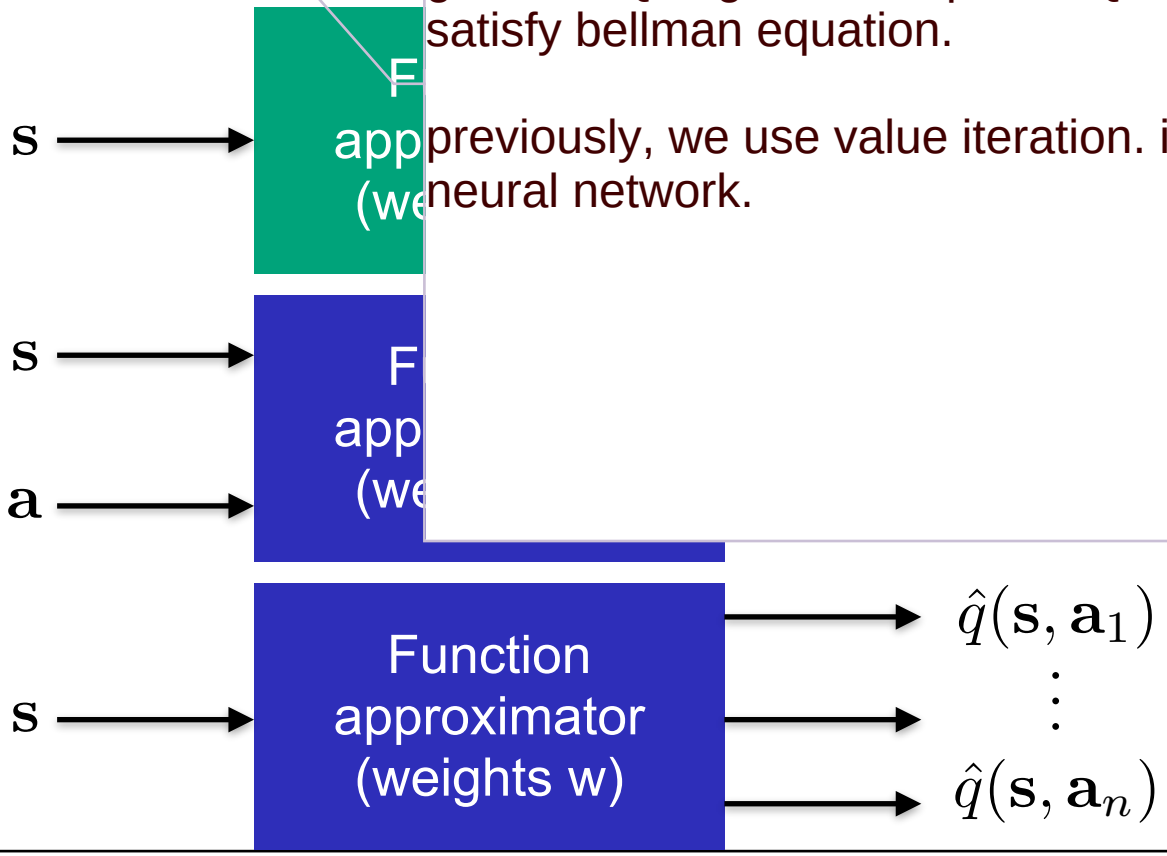
Deep Q n

combine deep neural network with Q function
loss = output q values computed from network- target Q value
goal: mini loss.
this loss is computed for each state in the env. for each env, we compute q value of all possible actions at this state.

goal of DQN: generate optimal Q func. This Q func must satisfy bellman equation.

previously, we use value iteration. in DQN, we use deep neural network.

DQN approxim
to approximatin



(basically, a mapping is learned from states to lookup tables over actions)

Deep Q networks

Linear approximation has nice properties, but task-specific features need to be designed

DQN: Instead, use a multi-layer neural network. Idea is that early layers can learn features

Thus, the network can directly take relatively ‘raw’ input without task-specific preprocessing

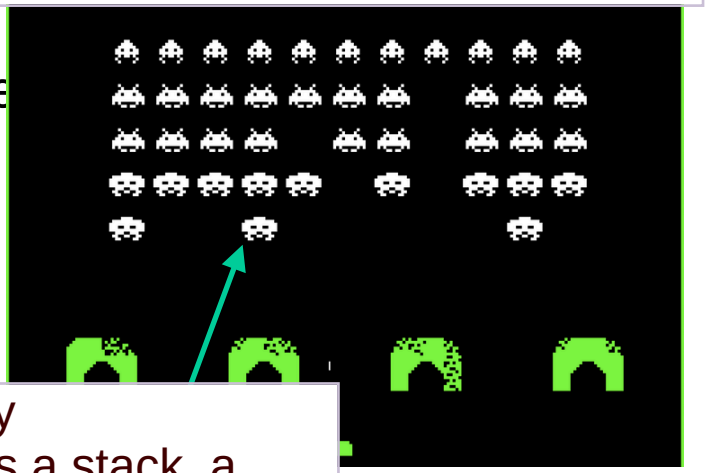
Deep Q networks

lec
https://canvas.uva.nl/courses/25598/pages/lecture-6-last-years-recording?module_item_id=1024064

book 16.5

Pre-processing (Task-agnostic)

- Downscale image and convert to grayscale
- Stack frames
- Reward = increment in score



1 convert rgb to gray scale, in order to reduce memory
2 use a few consecutive 时间上连续的 frame image as a stack. a stack tells: is a ball moving up or down, how fast a ball is. a stack describe a state. a state is an input to the network

DQN is composed by: conv layer, relu layer, fully connected layer

if 4 possible actions: go up, down, left, right
then output of DQN is q(s, a=up), q(s,a=down),q(s, a=left),q(s,a=right)

Q learning is off policy, so DQN is also off policy

t or right?

hianmag.com

Deep Q networks

Most layers in DQN are convolutional layers:

- Same filters are used at each location in the image
- Reduces #weights by only connecting neurons locally
- Further reduces #weights by sharing parameters at each location
- Imposes translation equivariance

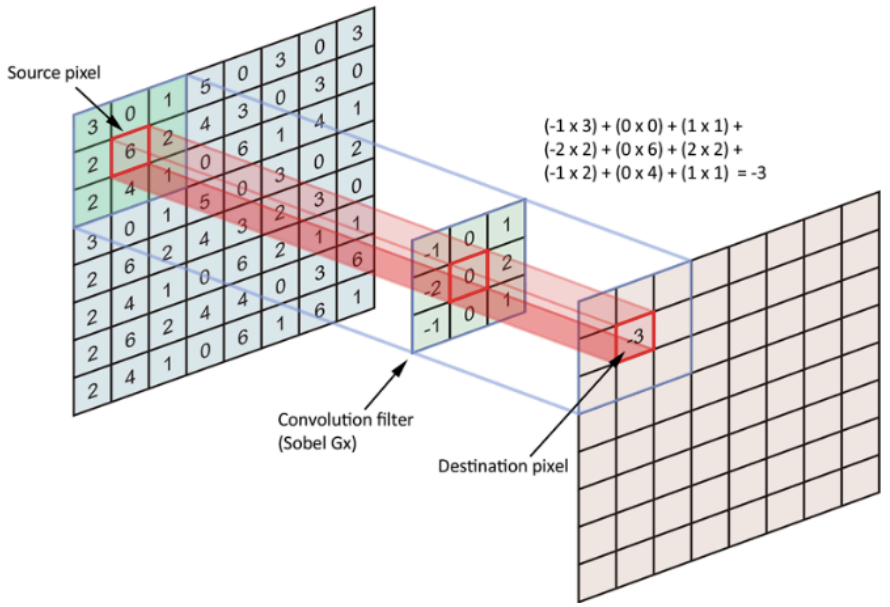


Figure: towardsdatascience.com

Deep Q networks

Convolutional layers and fully connected layers of ReLU units

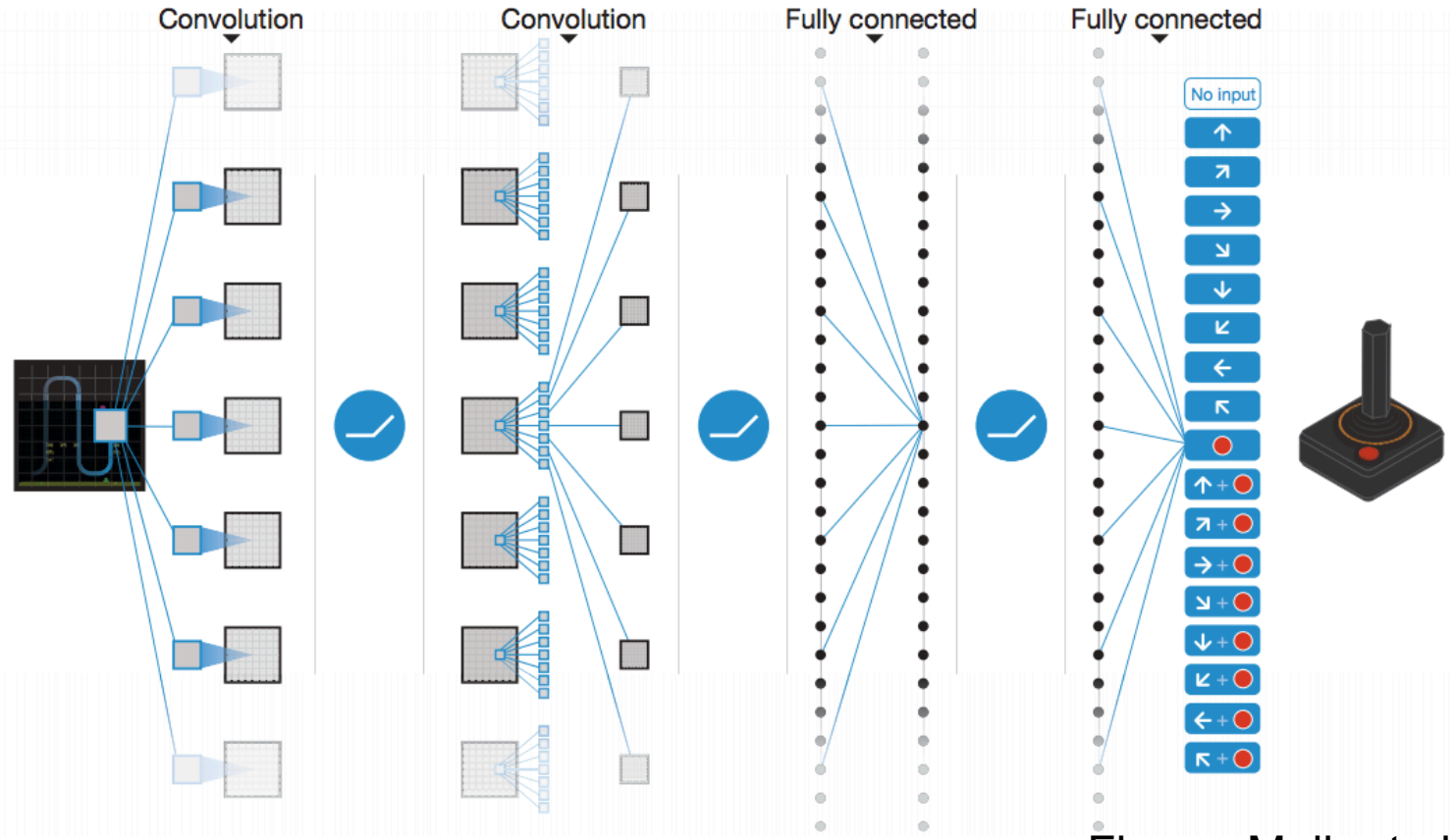


Figure: Mnih et al., 2015

Deep Q networks

In regular TD-learning, each transition used once. Inefficient

Also, subsequent transitions are highly **correlated**

Break correlation using experience replay

- Store experiences in a buffer
- At every time step, train on a batch of experiences from the buffer, not the most recent one
- Makes sure the network can't 'overfit' to recent experience only

Are we learning on-policy or off-policy?

off policy

Deep Q networks

Learning with semi-gradient version of q-learning

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \max_a \hat{q}(S', a, \mathbf{w}) - \hat{q}\left(S_t, a, \mathbf{w}\right) \right] \nabla \hat{q}\left(S_t, a, \mathbf{w}\right)$$

Target depends on w

Dependence of target on w can cause instabilities

- Copy parameters to $\tilde{\mathbf{w}}$ every C steps, and use these in the target

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \max_a \hat{q}(S', a, \tilde{\mathbf{w}}) - \hat{q}\left(S_t, a, \mathbf{w}\right) \right] \nabla \hat{q}\left(S_t, a, \mathbf{w}\right)$$

Deep Q networks

Learning with semi-gradient version of q-learning

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \max_a \hat{q}(S', a, \mathbf{w}) - \hat{q}\left(S_t, a, \mathbf{w}\right) \right] \nabla \hat{q}\left(S_t, a, \mathbf{w}\right)$$

Target depends on w

Dependence of target on w can cause instabilities

- Copy parameters to $\tilde{\mathbf{w}}$ every C steps, and use these in the target

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \max_a \hat{q}(S', a, \tilde{\mathbf{w}}) - \hat{q}\left(S_t, a, \mathbf{w}\right) \right] \nabla \hat{q}\left(S_t, a, \mathbf{w}\right)$$

Error term

Extra trick: clip error to [-1,1] (more stability, easy to set α)

Deep Q networks

Results with the **same** parameters and settings on many games

DQN has good performance on all these games

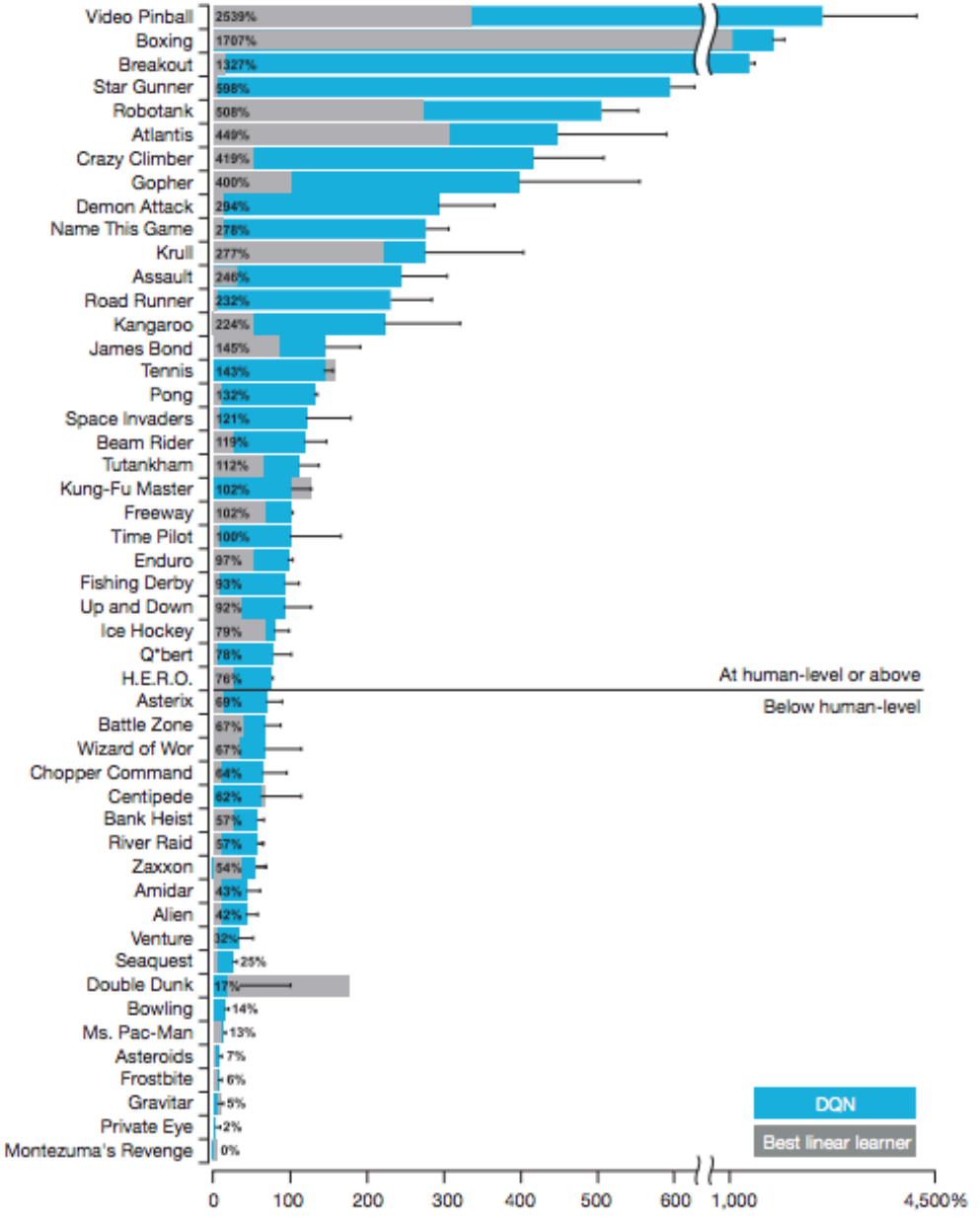


Figure: Mnih et al., 2015

Deep Q networks

Many of the tricks make the problem more ‘like supervised’

- Target network: fixed target value
- Experience replay: closer to iid

Still semi-gradient used off-policy

- Hence the need to stabilise learning
- Lack theoretical guarantees, but good empirical performance

Non-linear function approximation

- Risk of local optima seems not so bad in practice for deep nets

Extension: double DQN (similar to double Q learning)

Powerful demonstration of RL without task-specific features!

Conclusion

On-policy control straightforward

Off-policy prediction and control very tricky!

- Gradient TD, that uses gradient of PBE, is a solution
- Overhead: keep track of more values

Alternative: use additional mechanisms to keep learning stable

- Good empirical performance with DQN

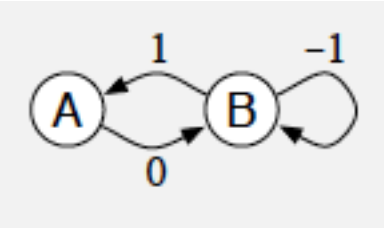
Thanks for your attention?

Feedback?

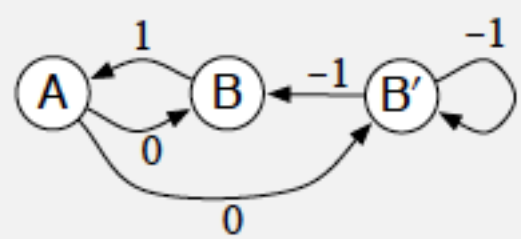
h.c.vanhoof@uva.nl

Minimize mean squared Bellman error?

Is it really impossible to learn from data?



$x(A)=[1,0]^T$
$x(B)=[0,1]^T$
$x(B')=[0,1]^T$



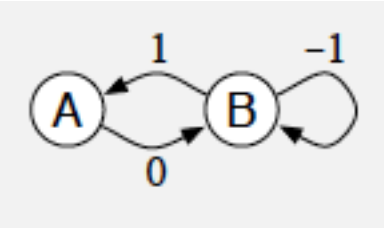
Observed data?

Best **w**?

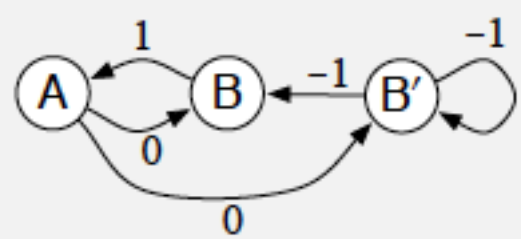
Figure: Sutton & Barto. RL:AI

Minimize mean squared Bellman error?

Is it really impossible to learn from data?



$x(A)=[1,0]^T$
$x(B)=[0,1]^T$
$x(B')=[0,1]^T$



Observed data?

- Same for both models

Best **w**?

- $[0,0]^T$ for model 1, but this doesn't have 0 error for model 2
- In fact, BE minimised at $[-0.5,0]^T$ for model 2 ($\gamma=1$)

Figure: Sutton & Barto. RL:AI