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

Last lecture: Semi-gradient SARSA

Straightforward analogue of semi-gradient TD:

Interleave using ϵ-greedy policy with weight update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \left(R_t + \hat{q} \left(S_{t+1}, A_{t+1}, \mathbf{w}_t \right) - \hat{q} \left(S_t, A_t, \mathbf{w}_t \right) \right) \nabla \hat{q} \left(S_t, A_t, \mathbf{w}_t \right)$$

Off-policy prediction with approximation

That was quite straightforward 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

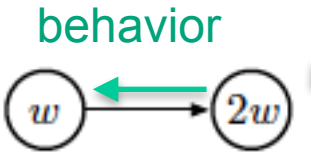
Use importance weight in semi-gradient TD(0)

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

Let's try it!

Off-policy prediction with approximation

Initial w is 10



On-policy, γ=1

From left to right:

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

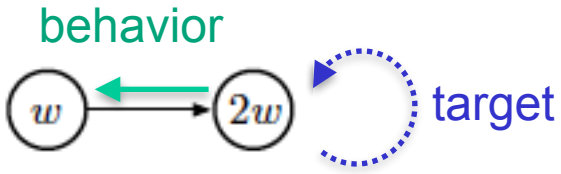
Right to left:

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

Mean 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:

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

Off-policy, γ=1

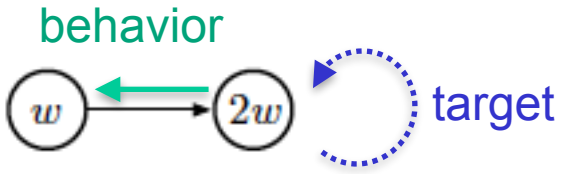
From left to right:

Right to left:

Mean 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:

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

Mean TDE tends to shrink!

Off-policy, γ=1

From left to right: =1

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

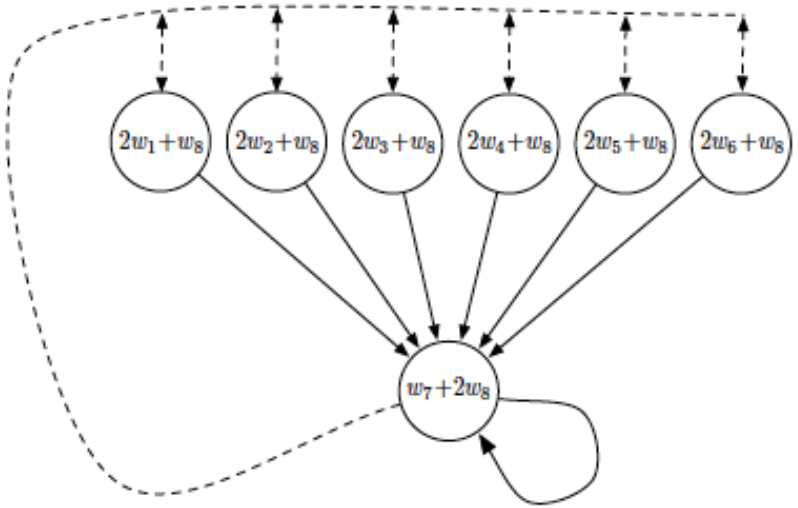
Right to left: =0

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

If ρ<<1, could TDE keep increasing?

Off-policy prediction with approximation

Yes, the TDE can keep increasing

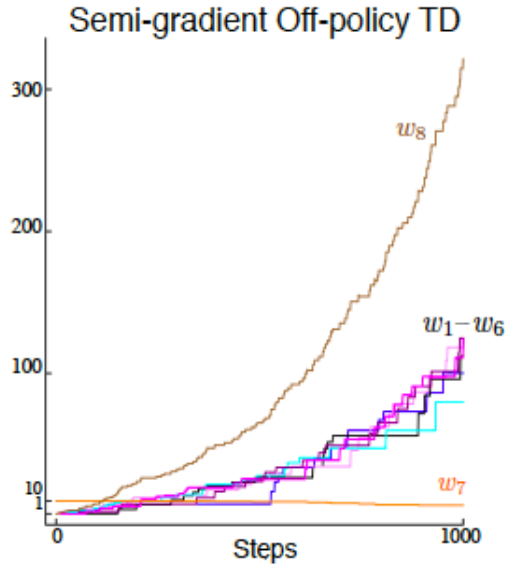


$\pi(\text{solid}|\cdot) = 1$

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

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

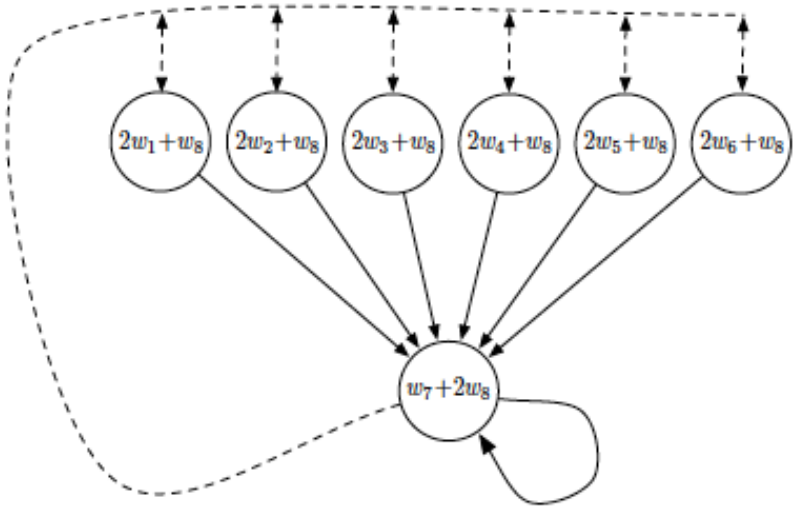
$\gamma = 0.99$



Figures: Sutton & Barto. RL:AI
Reinforcement Learning

Off-policy prediction with approximation

Yes, the TDE can keep increasing

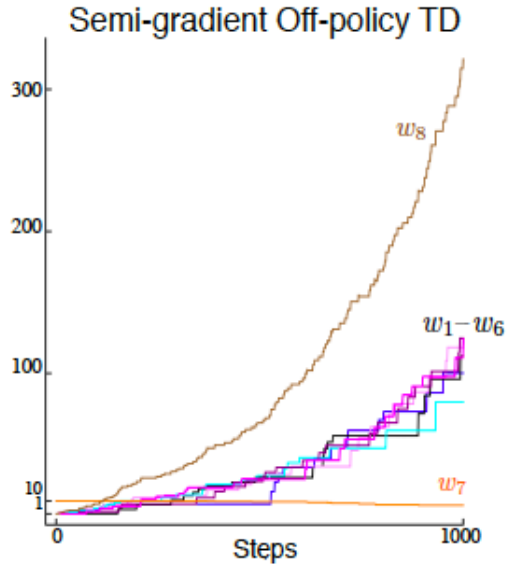


$\pi(\text{solid}|\cdot) = 1$

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

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

$\gamma = 0.99$



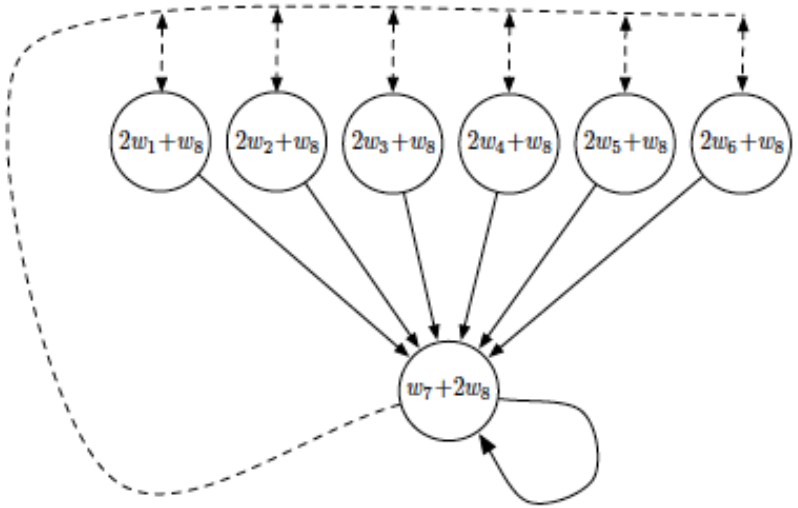
Effect is not because

- V not representable
- Random effects
- Dependent features

Figures: Sutton & Barto. RL:AI

Off-policy prediction with approximation

Yes, the TDE can keep increasing

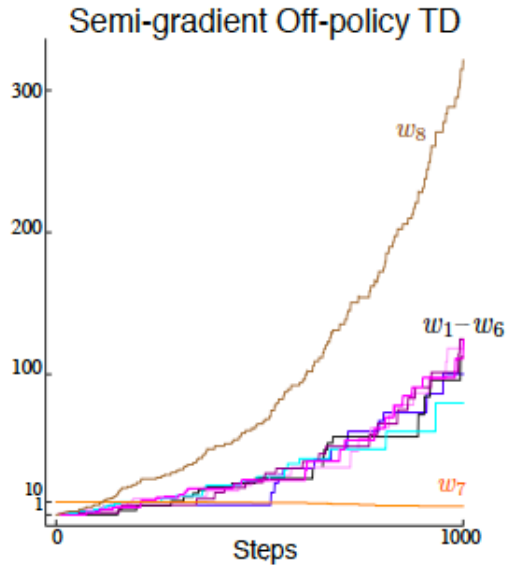


$\pi(\text{solid}|\cdot) = 1$

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

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

$\gamma = 0.99$



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!

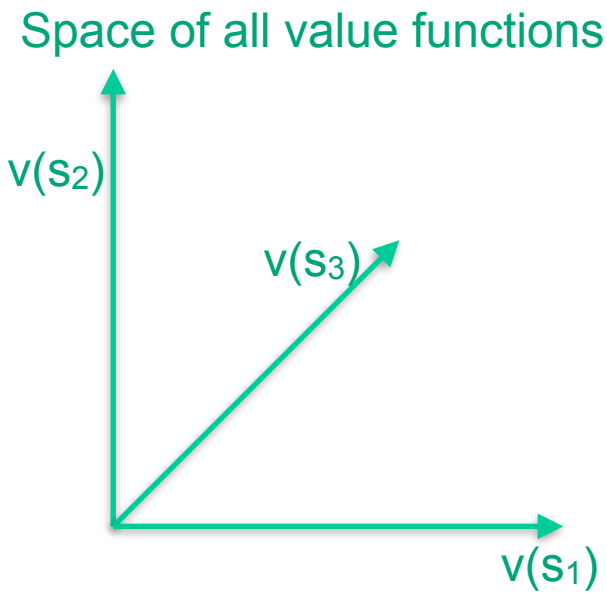
Get rid of off-policy learning?

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

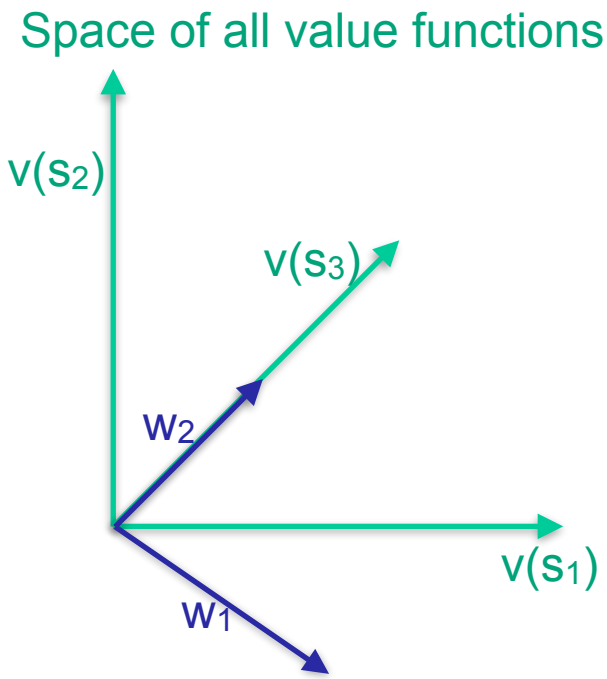
- $x(s_1)=[1,0]$
- $x(s_2)=[-0.5,0]$
- $x(s_3)=[0,1]$

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



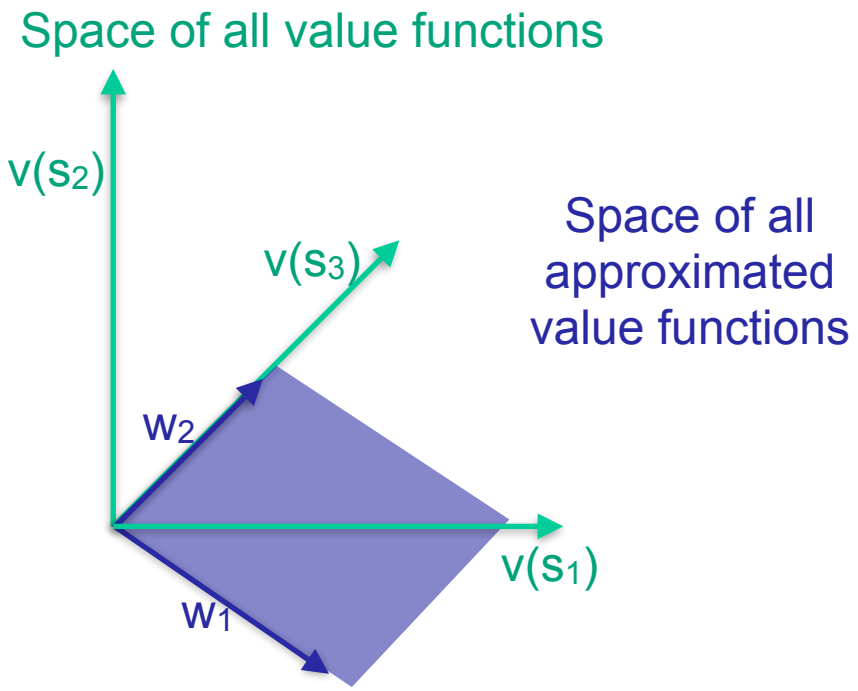
Approximated value
$\hat{v}=w_1x_1(s) + w_2x_2(s)$

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

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, updates approximate 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, updates approximate Bellman operator

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

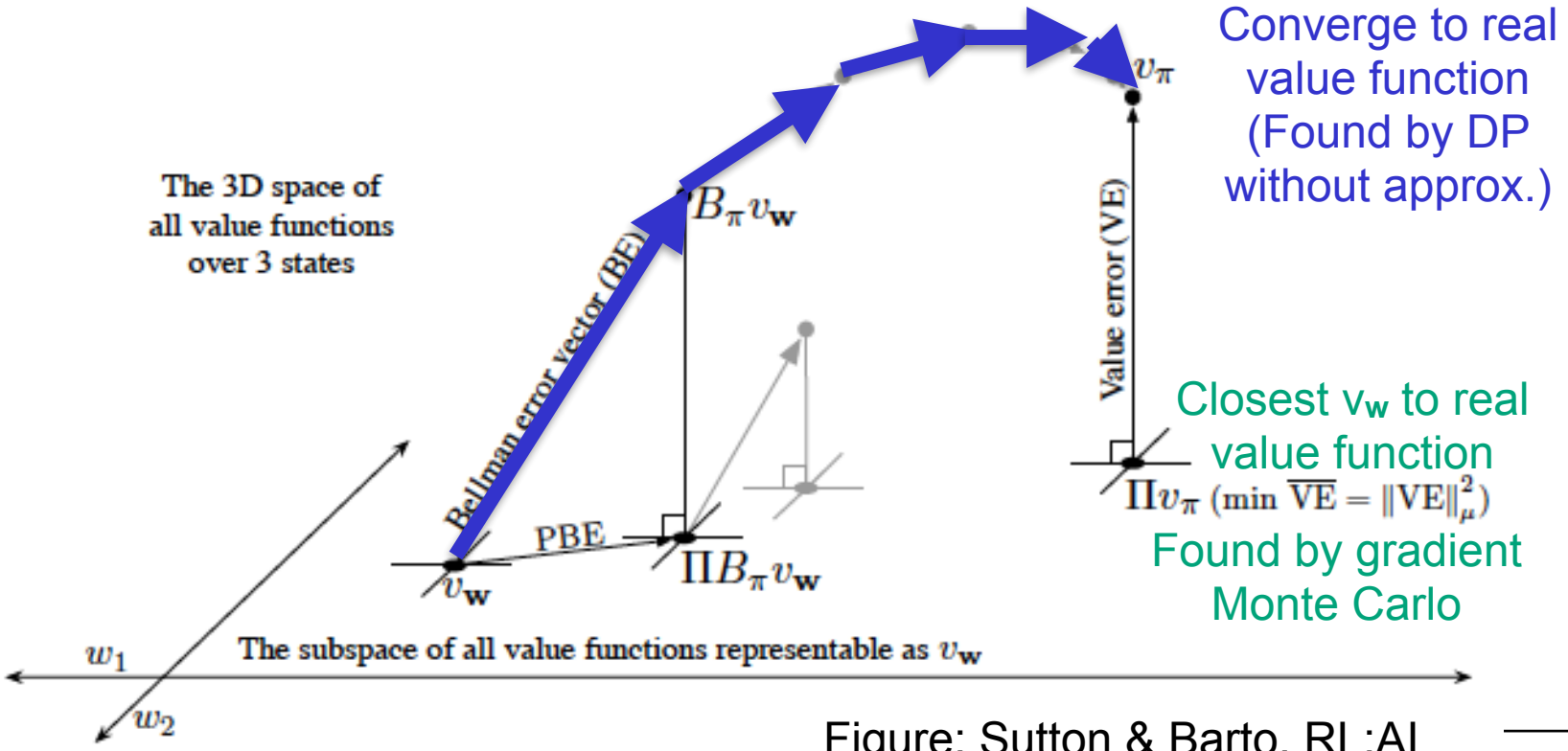


Figure: Sutton & Barto. RL:AI

Alternatives to semi-gradients?

Without approximation, updates approximate Bellman operator

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

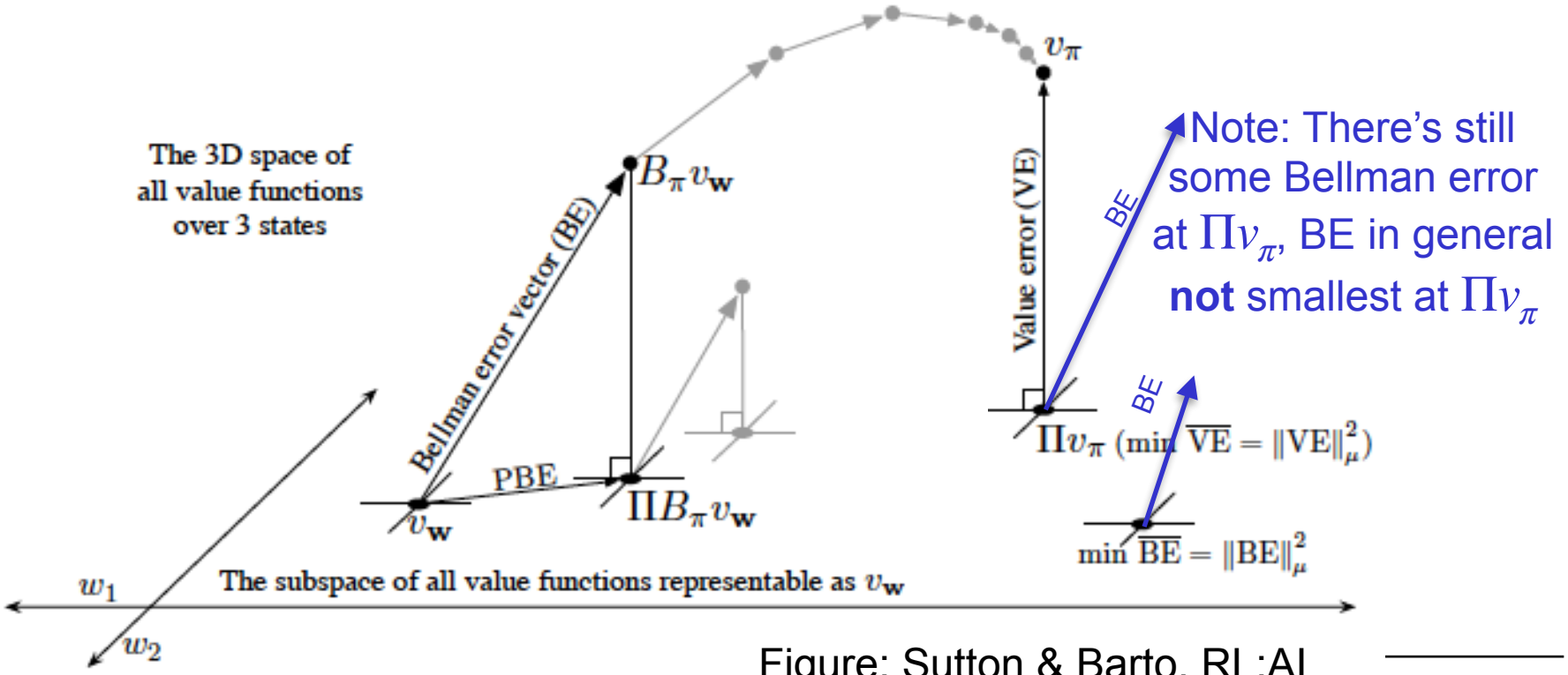
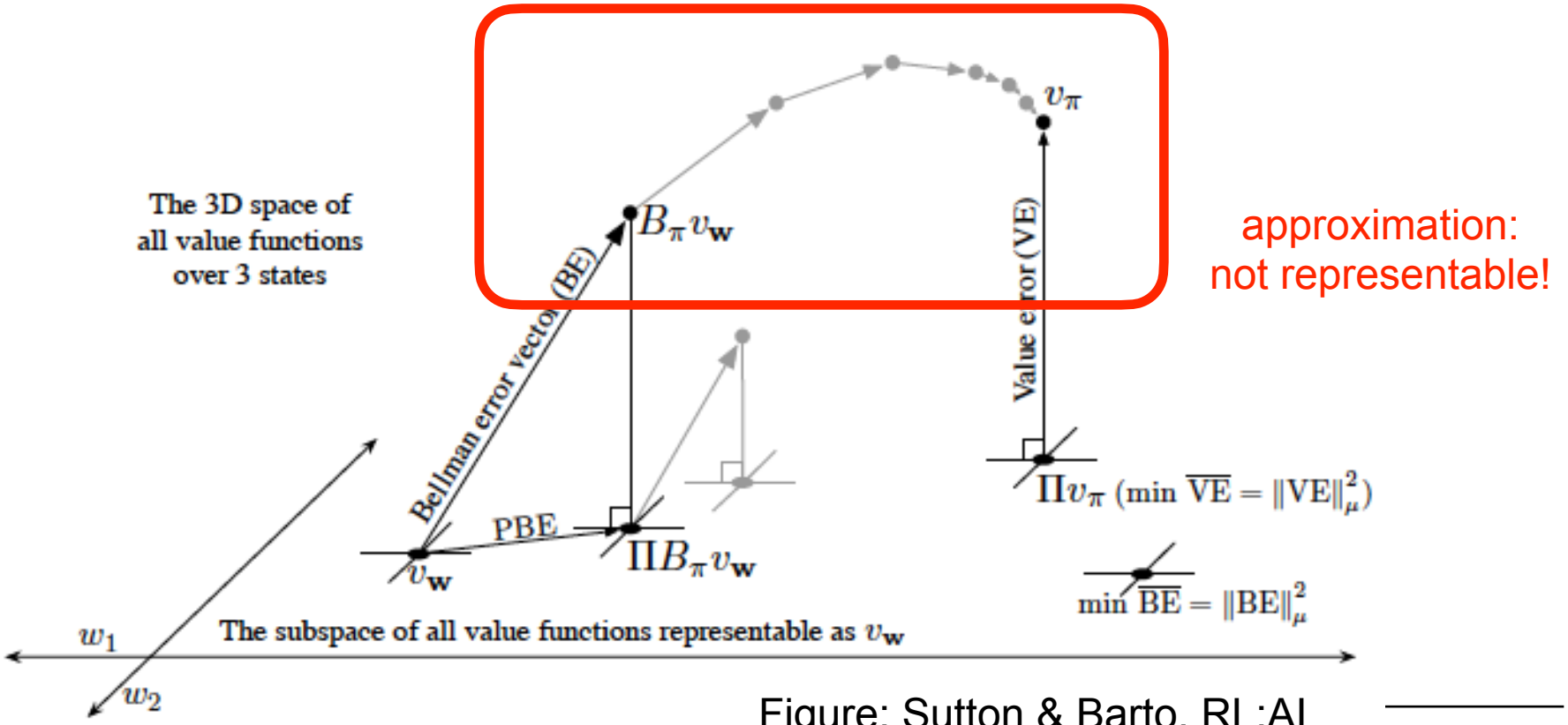


Figure: Sutton & Barto. RL:AI

Alternatives to semi-gradients?

Without approximation, updates approximate 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:

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

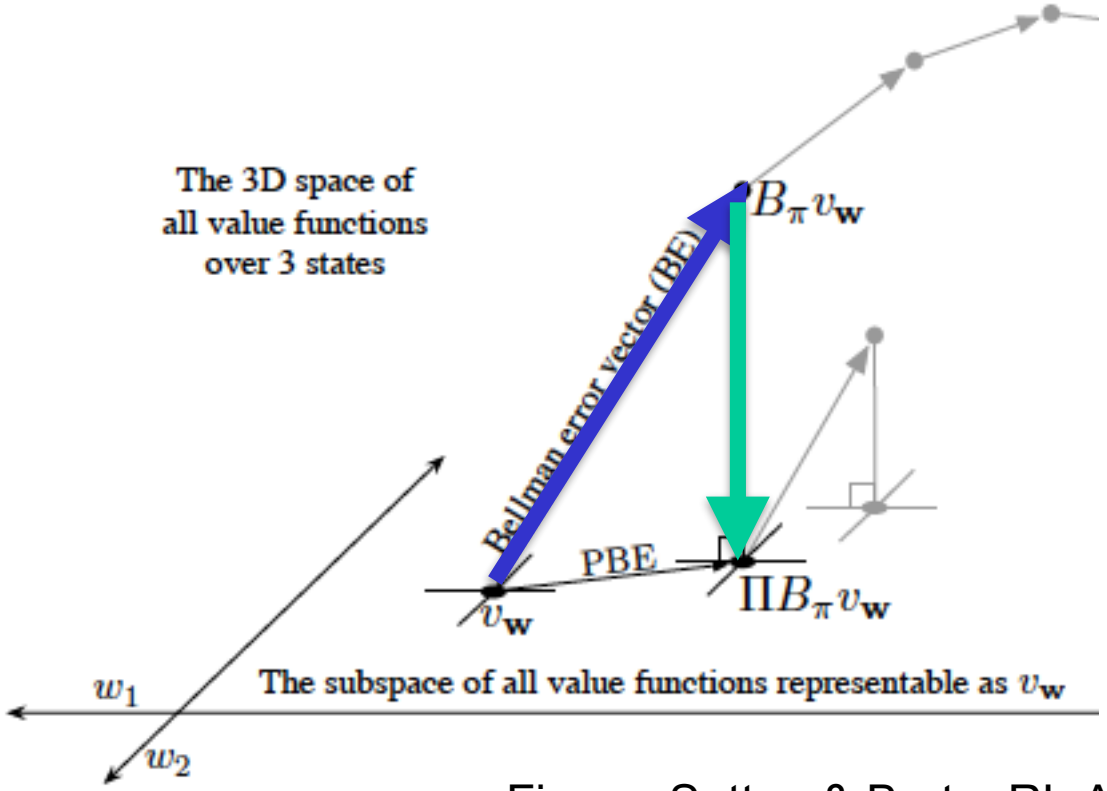
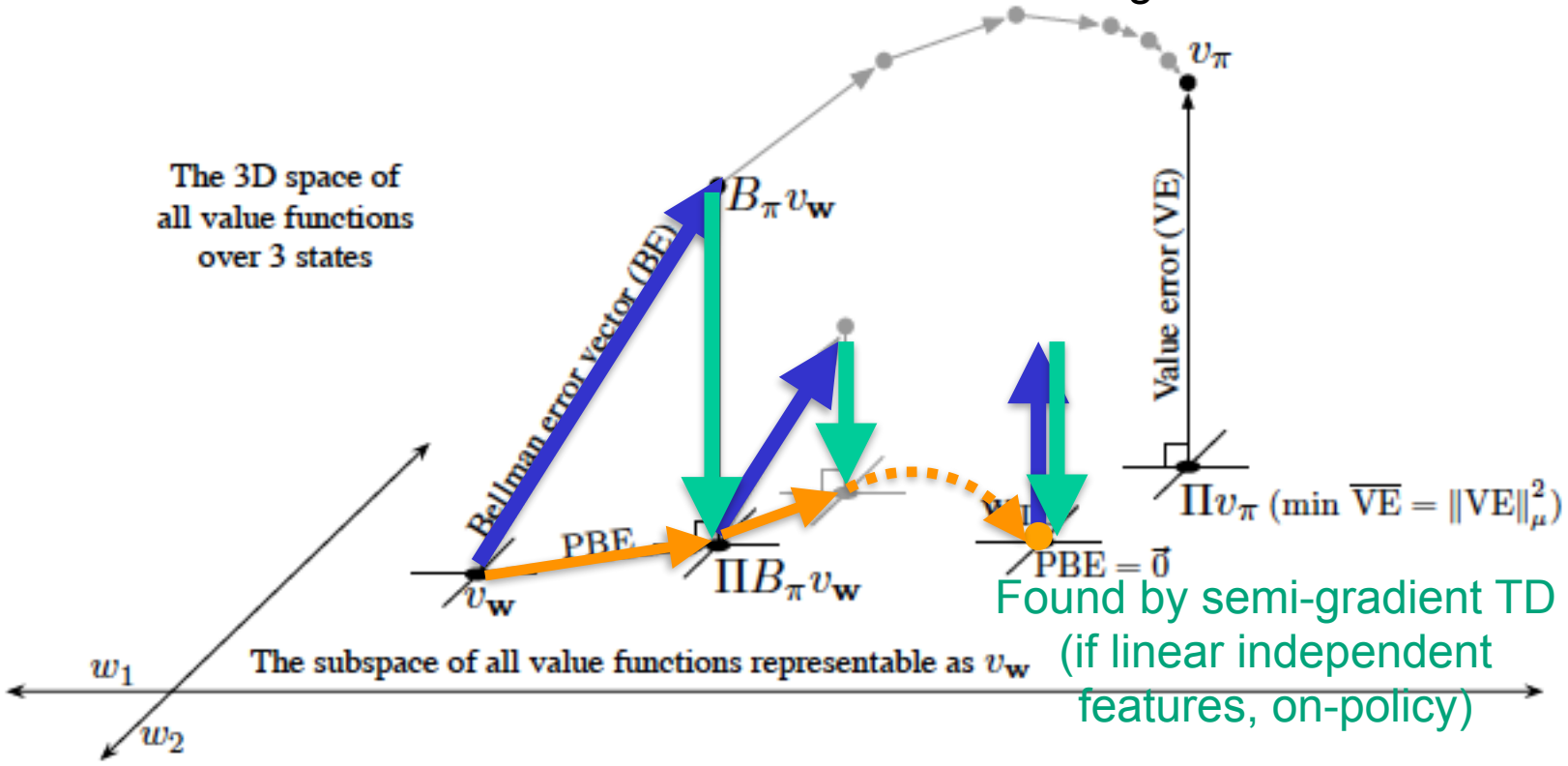


Figure: Sutton & Barto. RL:AI

Alternatives to semi-gradients?

With approximation, need to stay within representable space

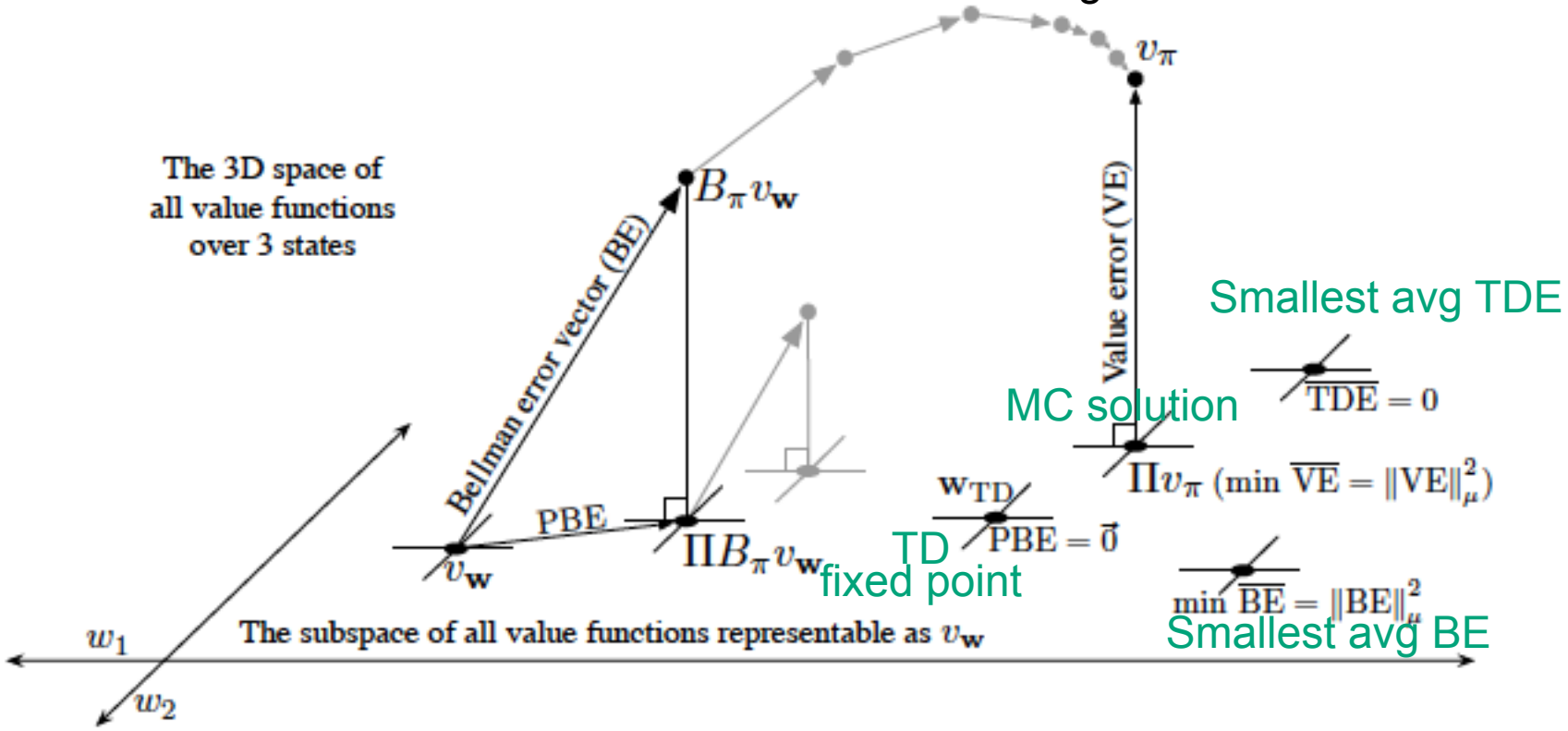
Figure: Sutton & Barto. RL:AI



Alternatives to semi-gradients?

With approximation, need to stay within representable space

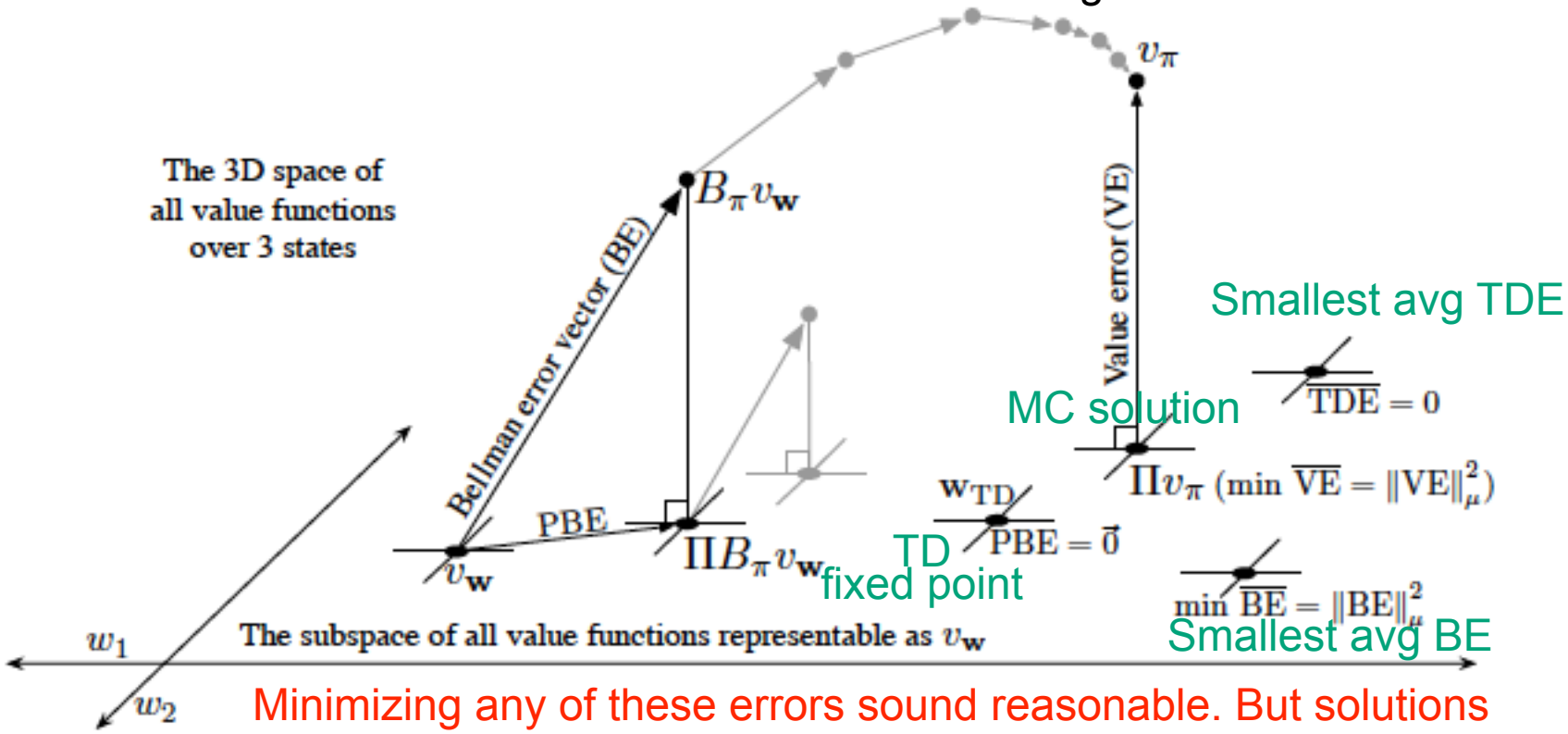
Figure: Sutton & Barto. RL:AI



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?

Alternatives to semi-gradients?

Find a true gradient algorithm based on minimizing an error?

- Mean squared TD error?

$$\delta_t = R_{t+1} + \gamma v_{\mathbf{w}}\left(S_{t+1}\right) - v_{\mathbf{w}}\left(S_t\right)$$

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

Average under visitation distribution μ

- MS Bellman error?

$$\overline{\mathrm{BE}}(\mathbf{w}) = \sum_{s \in S} \mu(s) \mathbb{E}\left[\delta_t | S_t = s, A_t \sim \pi\right]^2$$

- MS Projected Bellman error

$$\overline{\mathrm{PBE}}(\mathbf{w}) = \sum_{s \in S} \mu(s) \left(\Pi \mathbb{E}\left[\delta_t | S_t = s, A_t \sim \pi\right]\right)^2$$

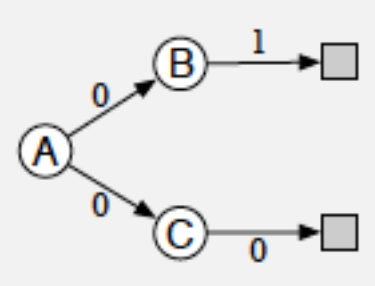
Minimize mean squared TD error?

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

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?

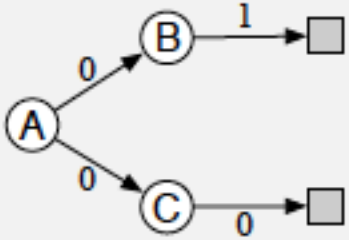
Let's try something else

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

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

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

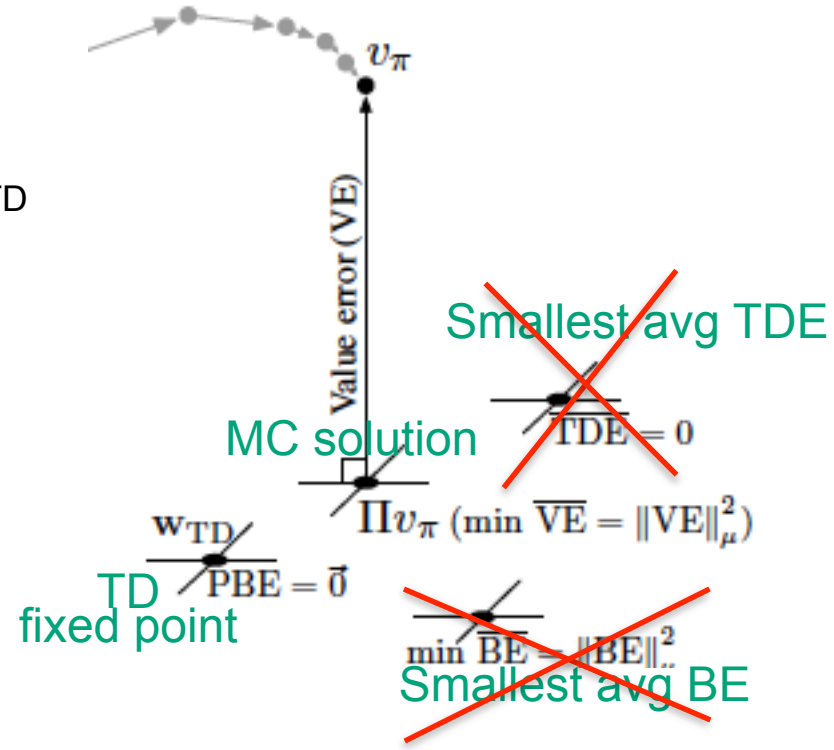


Figure: Sutton & Barto. RL:AI

Mean squared projected Bellman error?

It can be shown that

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

Reproducing the exact steps is outside the scope of the course (the book gives a good explanation if you're curious)

Mean squared projected Bellman error?

It can be shown that

$$\nabla \overline{\mathrm{PBE}}(\mathbf{w}) = \nabla \left\| \Pi \bar{\delta}_{\mathbf{w}} \right\|_{\mu}^{2}$$

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

Reproducing the exact steps is outside the scope of the course (the book gives a good explanation if you're curious)

Can we just plug in a new samples (s,a,s') tuple in each $\mathbb{E}$?

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

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

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 feature approximation, 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
- Extension needed to apply it to non-linear approximators
- *If* semi-gradient TD converges, it seems to converge faster in practice

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.

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

	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

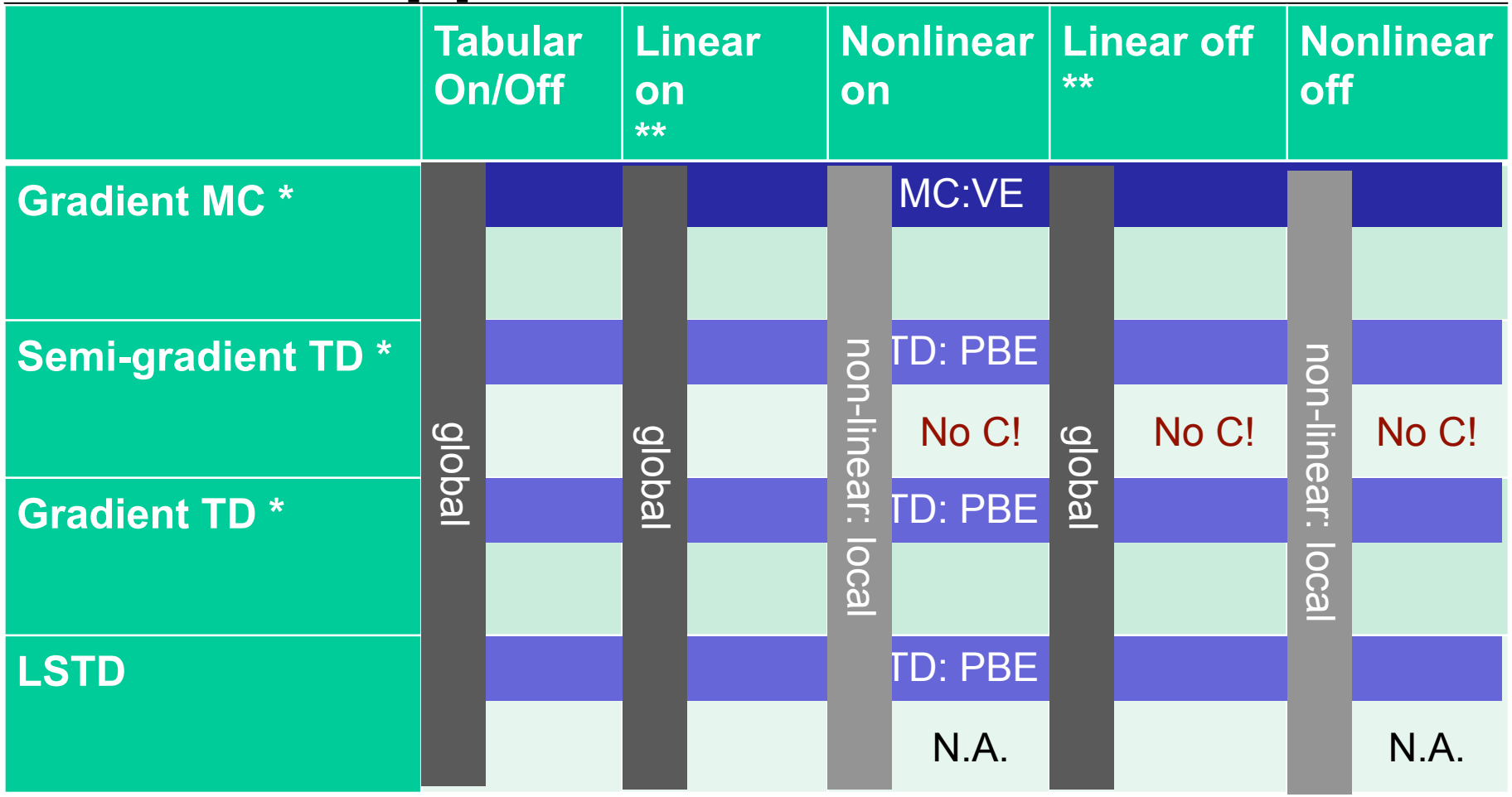
Convergence with function approximation

	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



* with appropriate step-size schedule

** if features independent, single solution

Deep Q networks

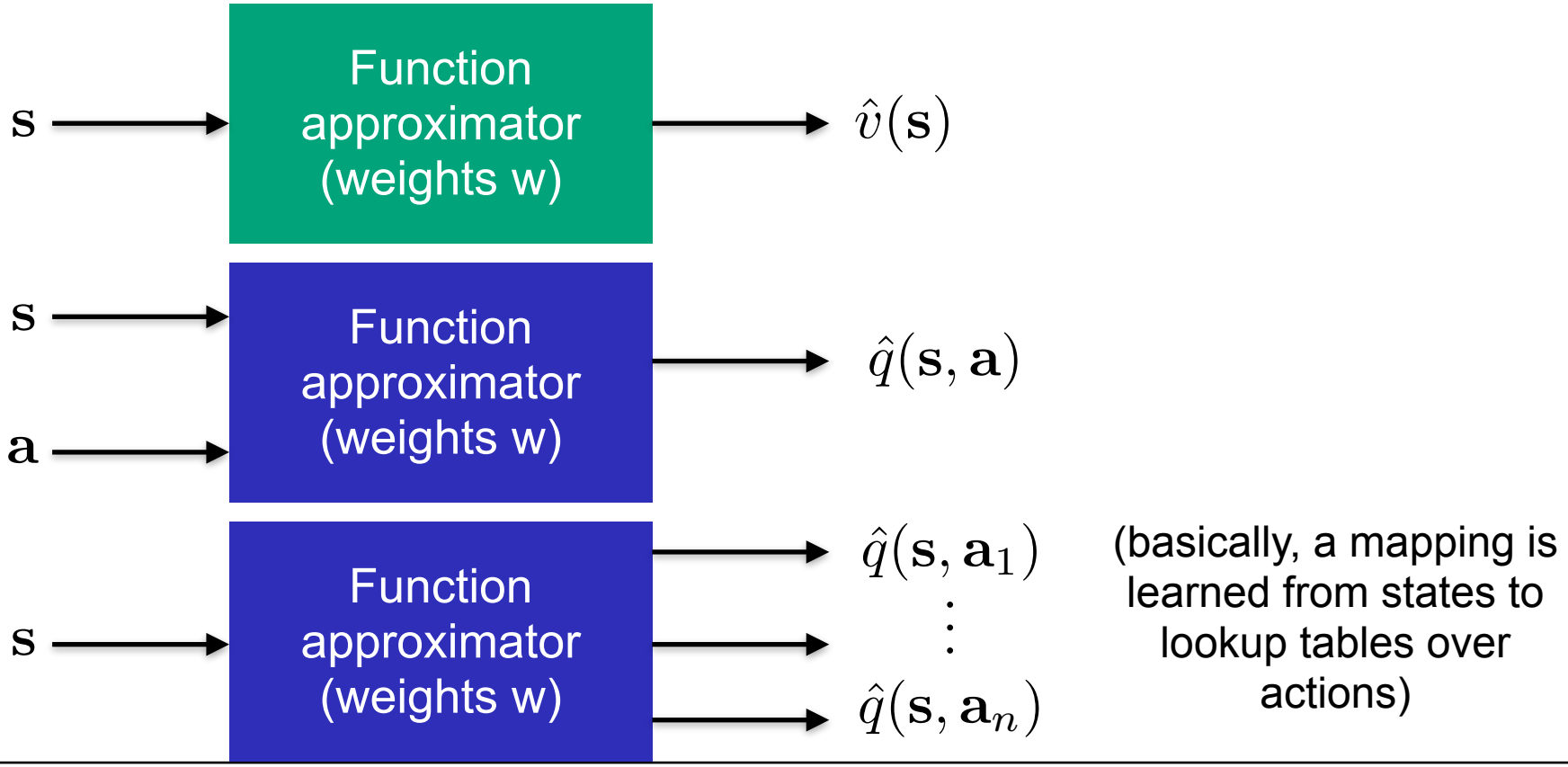
The famous algorithm for learning to play Atari games: Deep Q networks (DQN)

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

Figure: Deepmind.com

Deep Q networks

DQN approximates the Q function. Two possibilities compared to approximating V:



(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

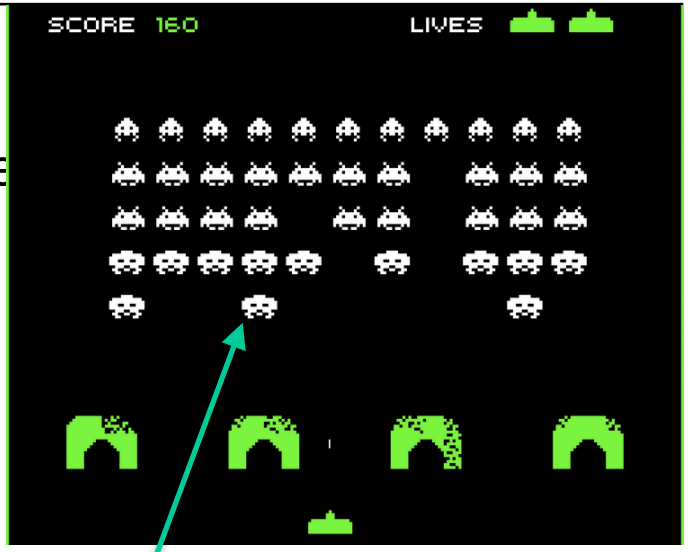
DQN: Instead, use a multi-layer neural network. Idea is that early layers can learn features

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

Deep Q networks

Pre-processing (Task-agnostic)

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



Going left or right?

Figure: Smithsonianmag.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

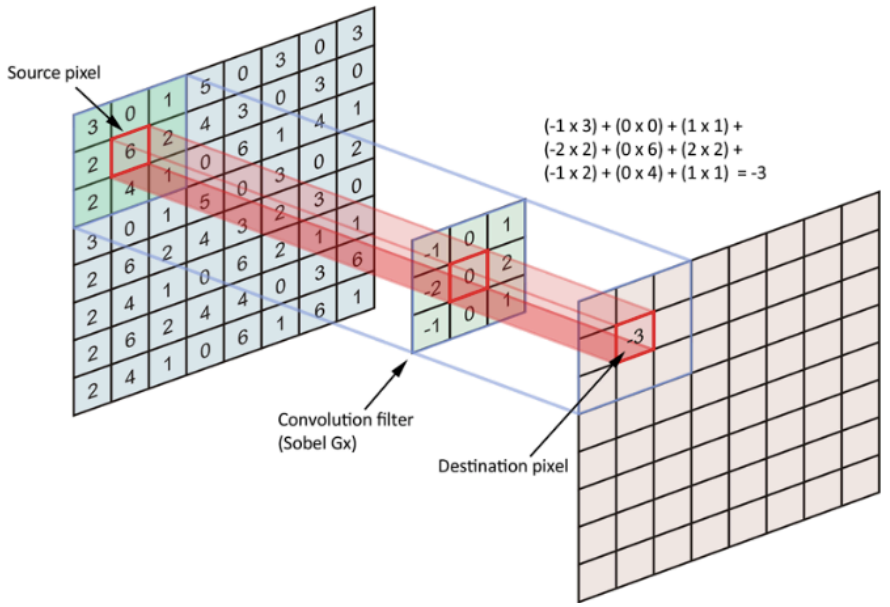


Figure: towardsdatascience.com

Deep Q networks

Convolutional layers and fully connected layers of ReLU units

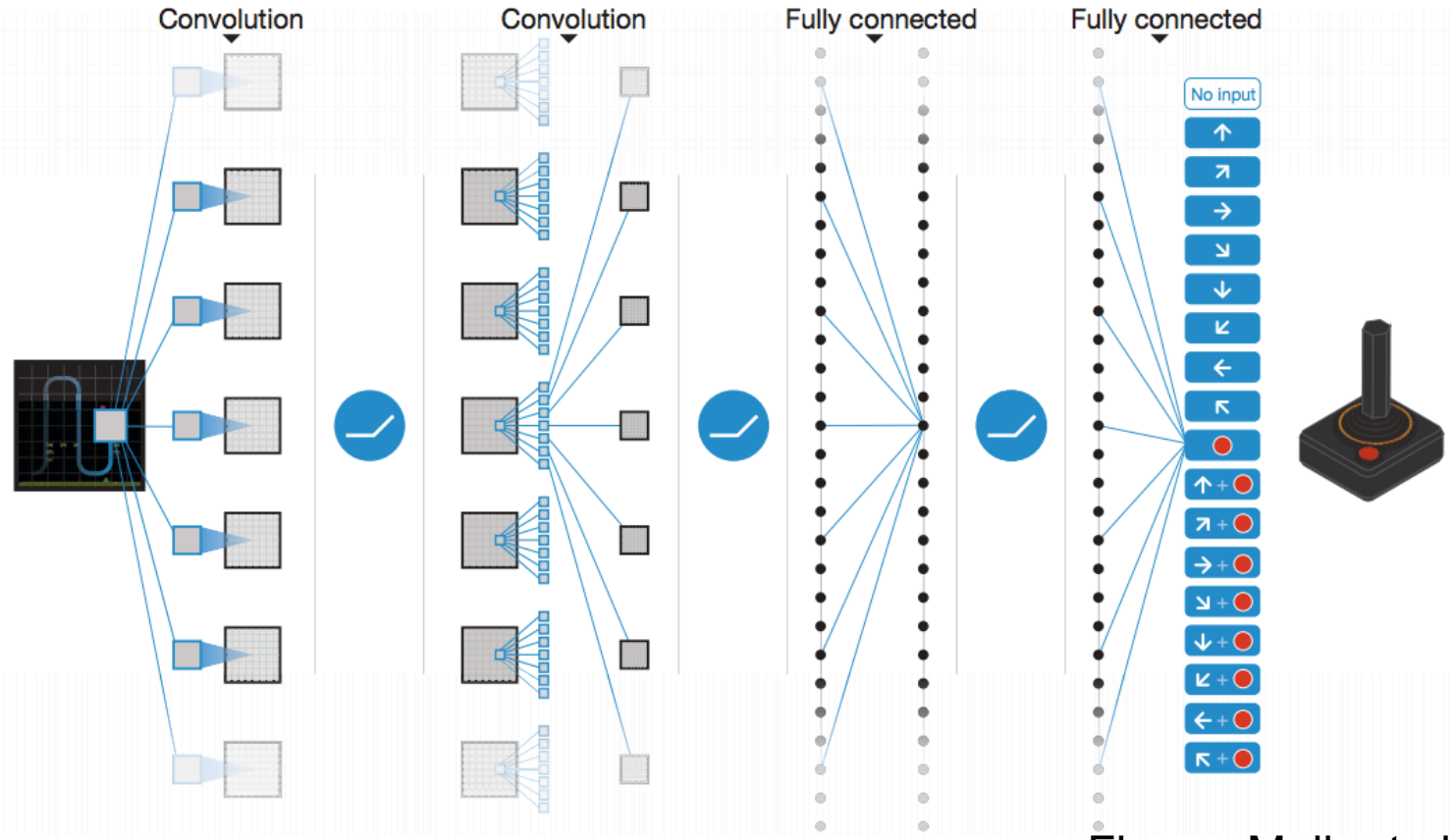


Figure: Mnih et al., 2015

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?

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

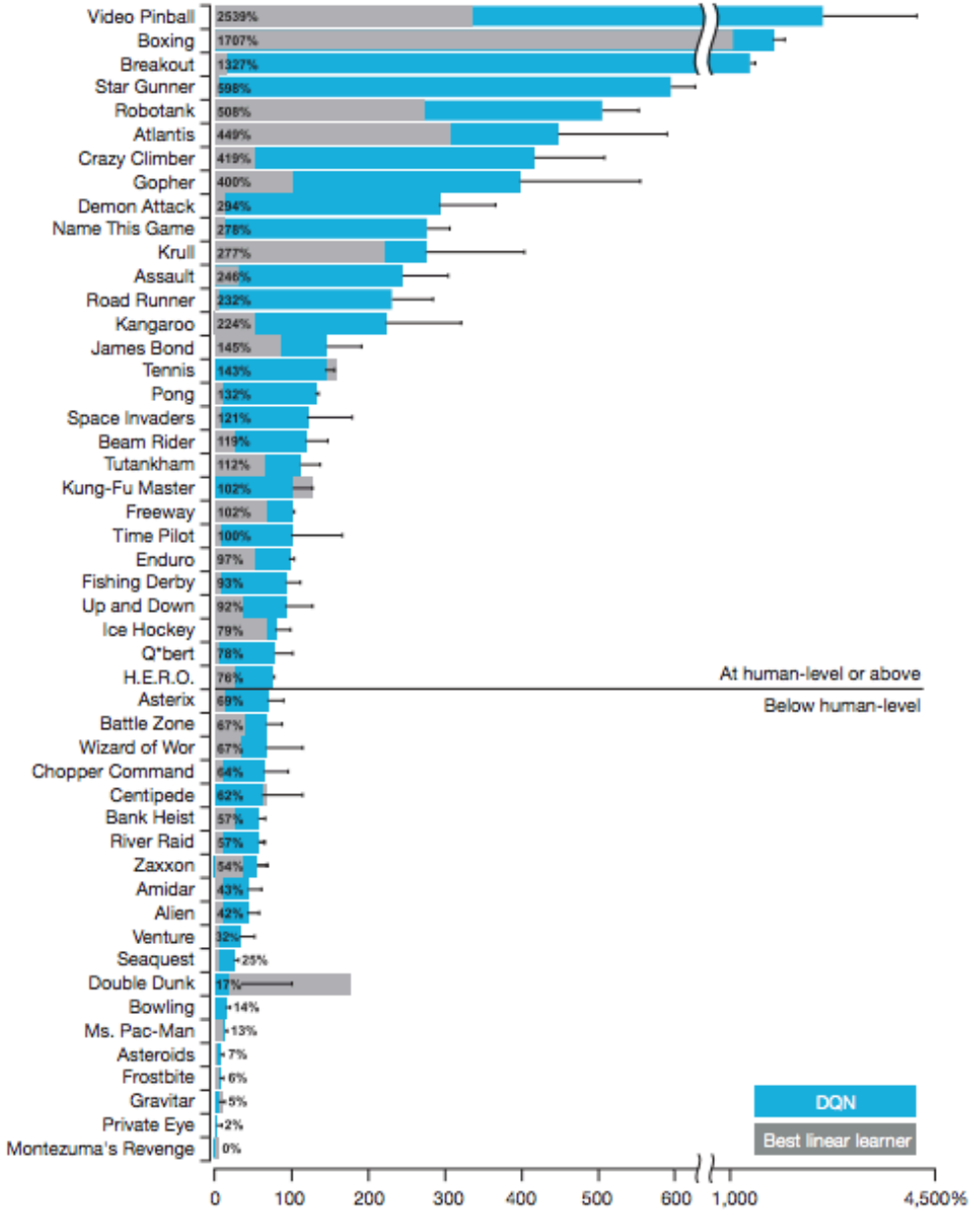


Figure: Mnih et al., 2015

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

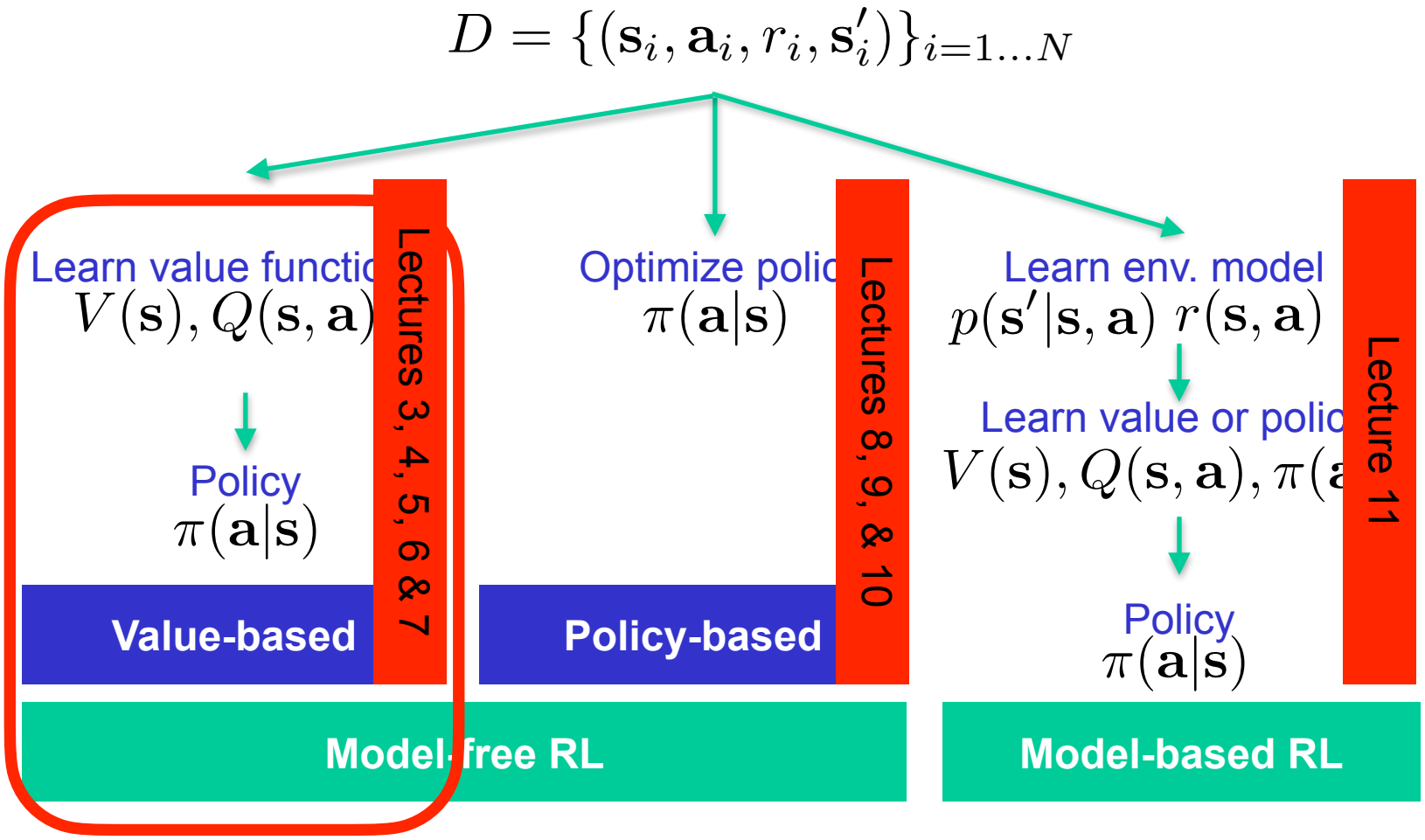
Non-linear function approximation

- 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!

Big picture: How to learn policies



Thanks to Jan Peters

Back to the big picture

Last week: *prediction* methods

Approximate method

semi-gradient TD(0)
& LSTD

gradient
Monte-Carlo

Exact methods

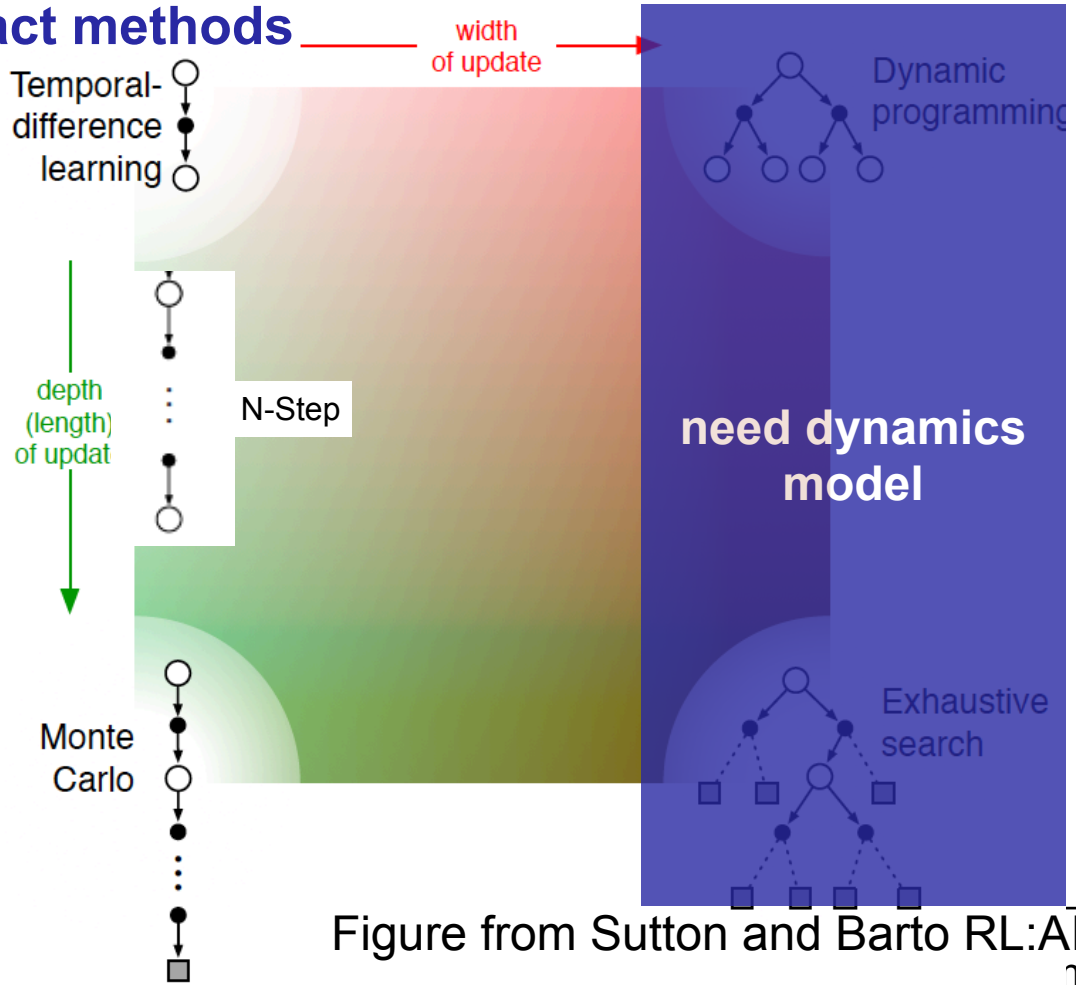
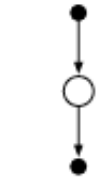


Figure from Sutton and Barto RL:AI

Control methods

On-policy (narrow)



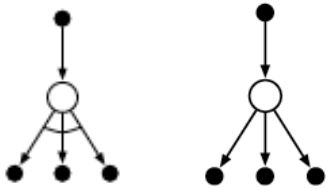
Sarsa

N-step SARSA

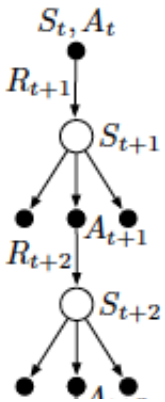
Monte-Carlo control

Depth

Off-policy (wide)



Q-learning & Expected SARSA



(Tree-back-up)

Control methods

On-policy (narrow)

Sarsa

Semi-gradient SARSA

N-step SARSA

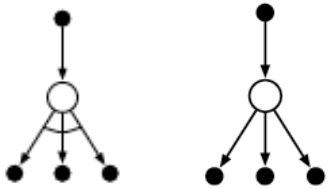
(Semi-gradient n-step SARSA)

Monte-Carlo control

(Gradient MC)

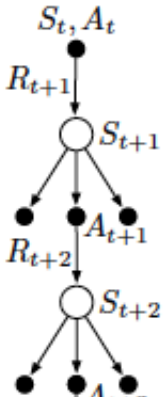
Depth

Off-policy (wide)



Q-learning & Expected SARSA

DQN
GTD2 (off-policy but 'narrow')



(Tree-back-up)

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
- GTD2 as discussed, only works for linear f.a.:
  not really applicable to ‘raw data’

Alternative: use additional mechanisms to keep learning stable

- Good empirical performance with DQN directly from ‘raw’ data!

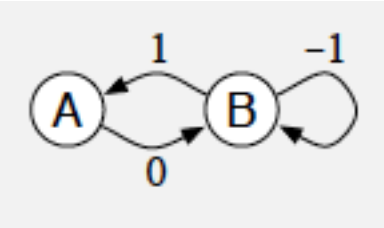
Thanks for your attention?

Feedback?

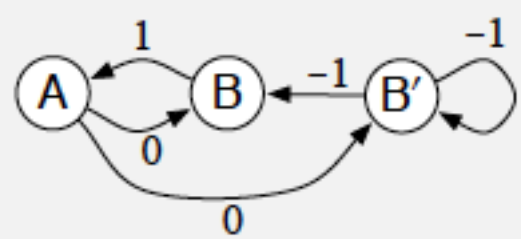
h.c.vanhoof@uva.nl

Minimize mean squared Bellman error?

Is it really impossible to learn from data?



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



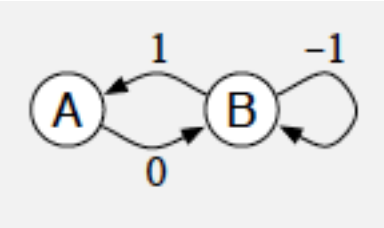
Observed data?

Best **w**?

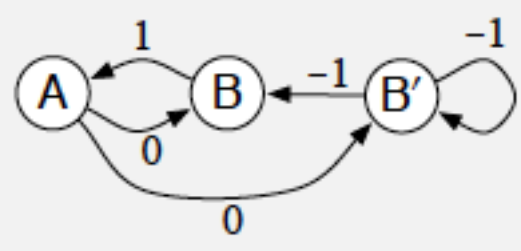
Figure: Sutton & Barto. RL:AI

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