Source Traces

for Temporal Difference Learning

Silviu Pitis
silviu.pitis@gmail.com

Very Roughly:

A “source trace” is a vector that can be used to distribute TD errors (like an eligibility trace)

entries represent eligibilities of *potential* sources rather than *immediate* sources

model-based method

Presentation Outline

1) Motivation & Basic Intuition / Derivation

[TD(0), Eligibility Traces, Successor Representation]

2) Selected Results / Contributions

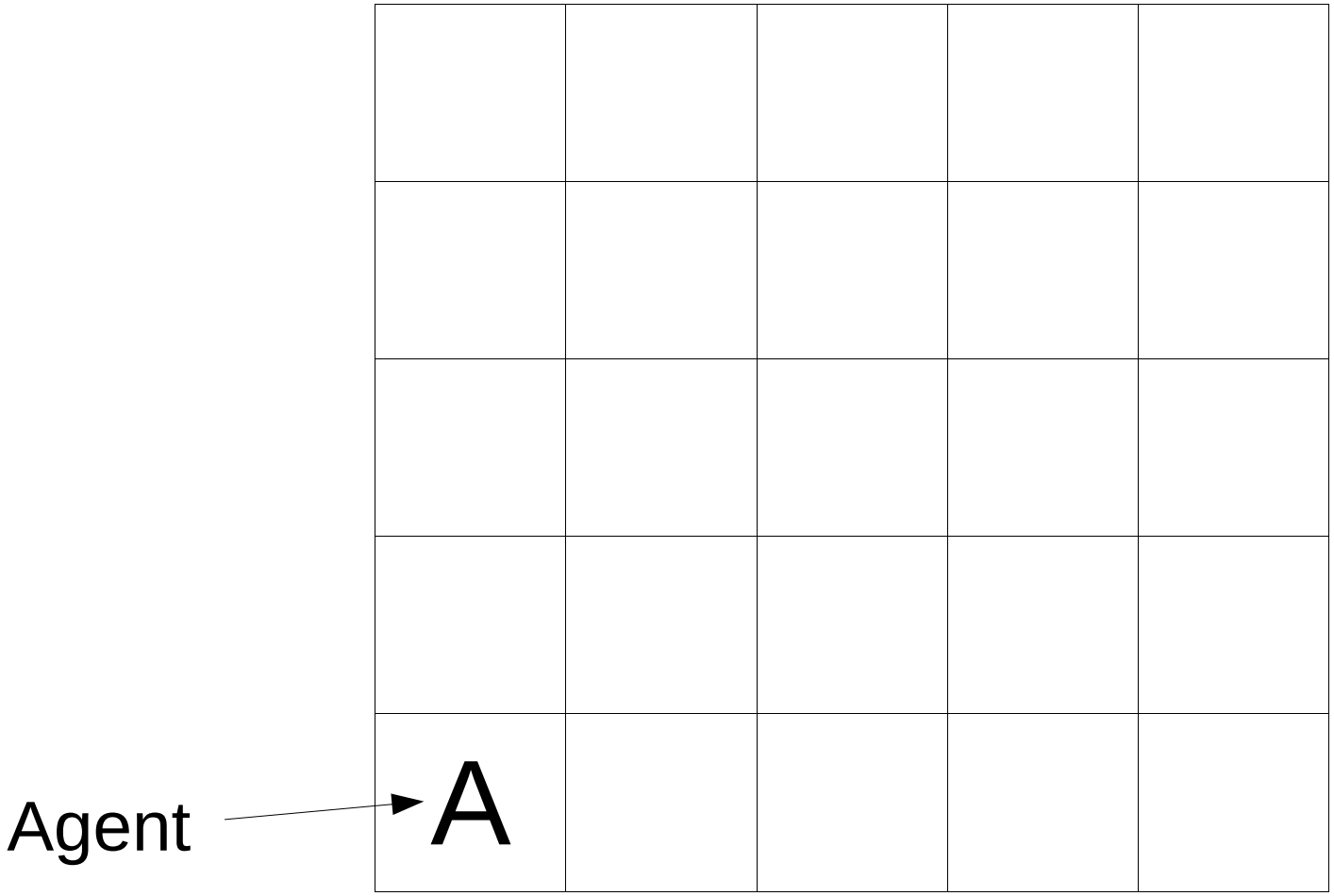
[Source Learning Algorithm, Convergence, TD-Source]

3) Extension

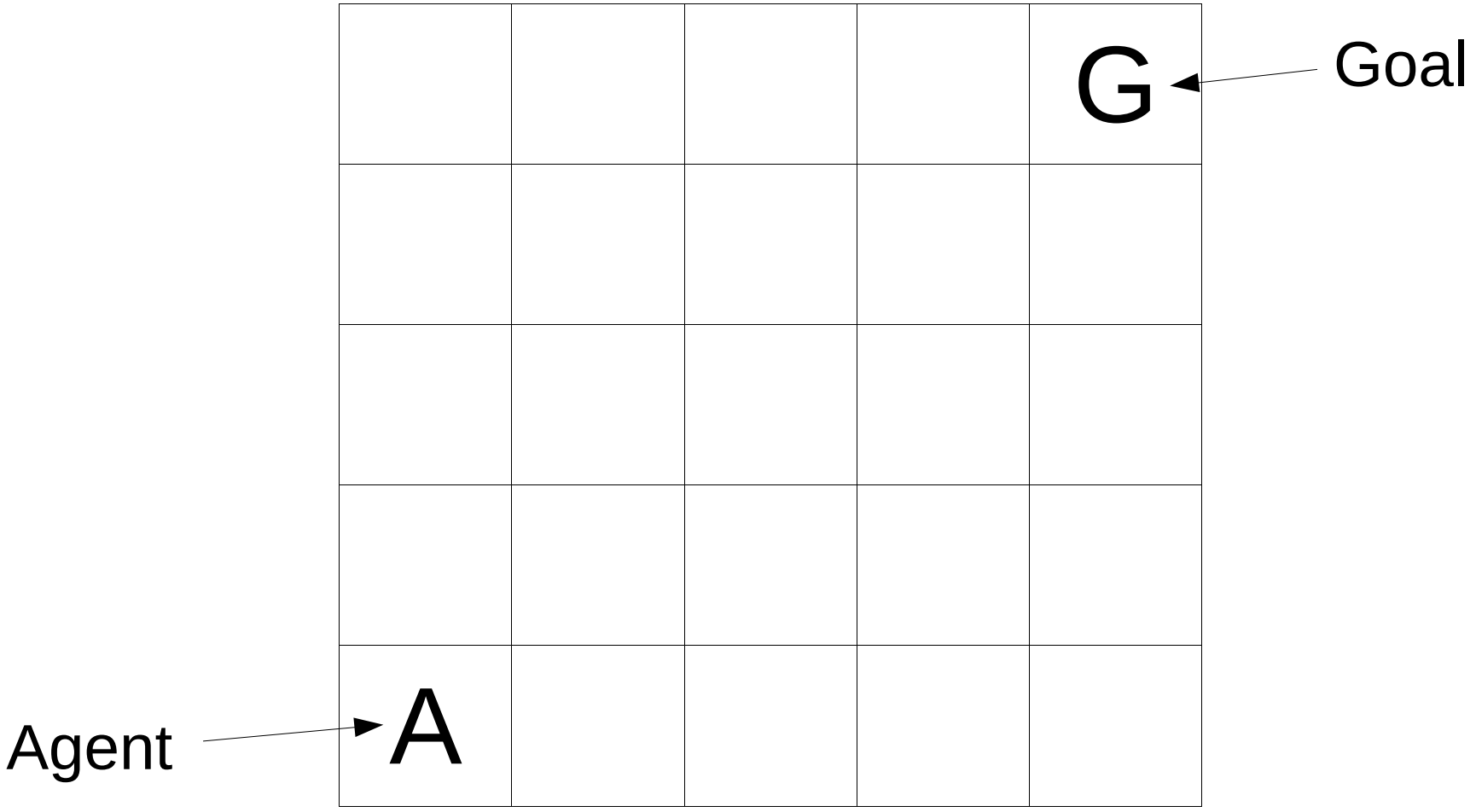
[“S”-function as a GVF on state sets]

Consider a gridworld, where the agent has a fixed random policy (it moves randomly).

Each episode starts like this:

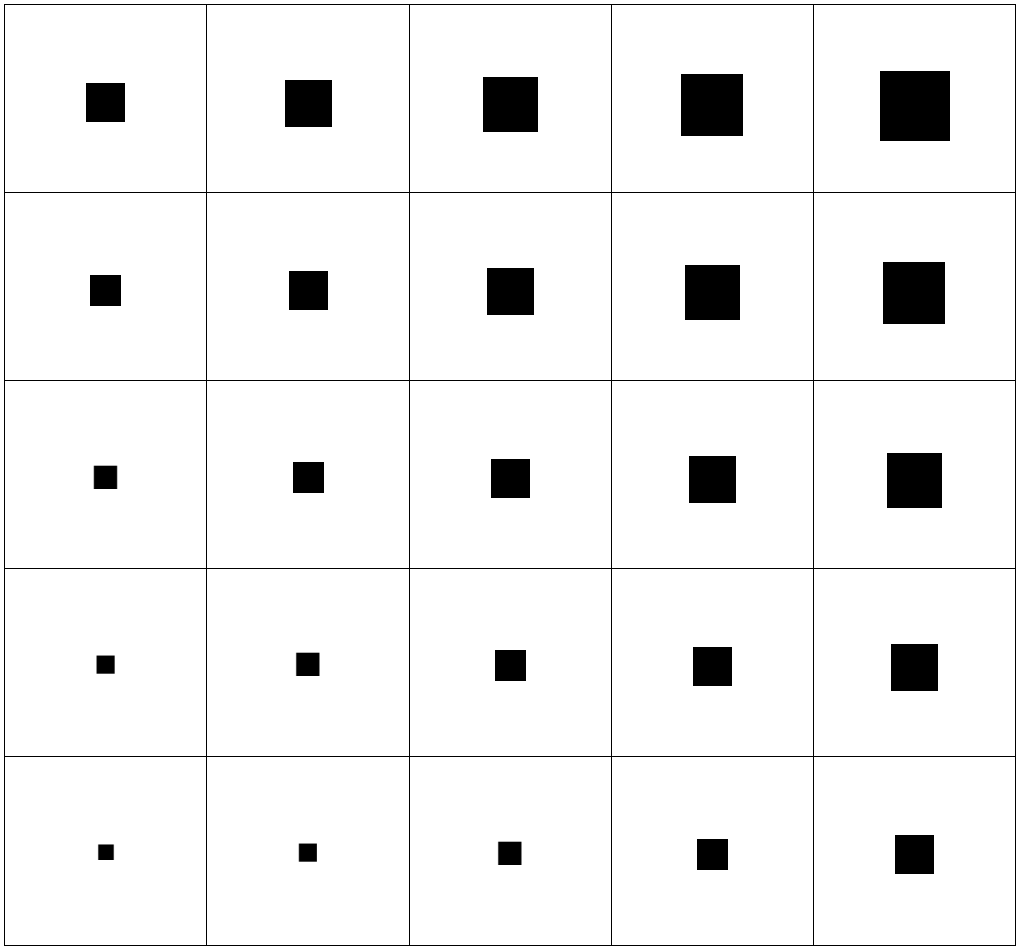


Reward is 0 everywhere, except at G, where it is 1.
Rewards are discounted.
Episode terminates *after* the agent reaches the goal.



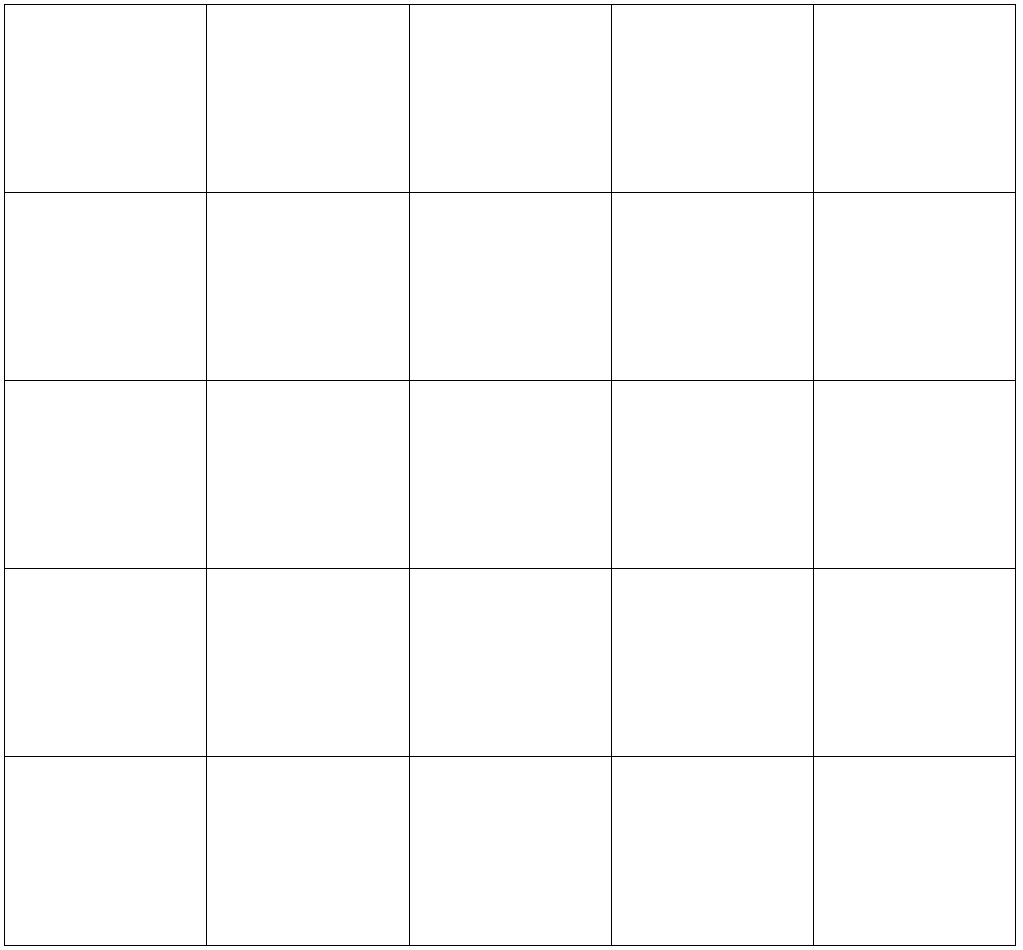
We want to learn the value of each square.

What the value might look like (further squares are valued less because of discount factor):



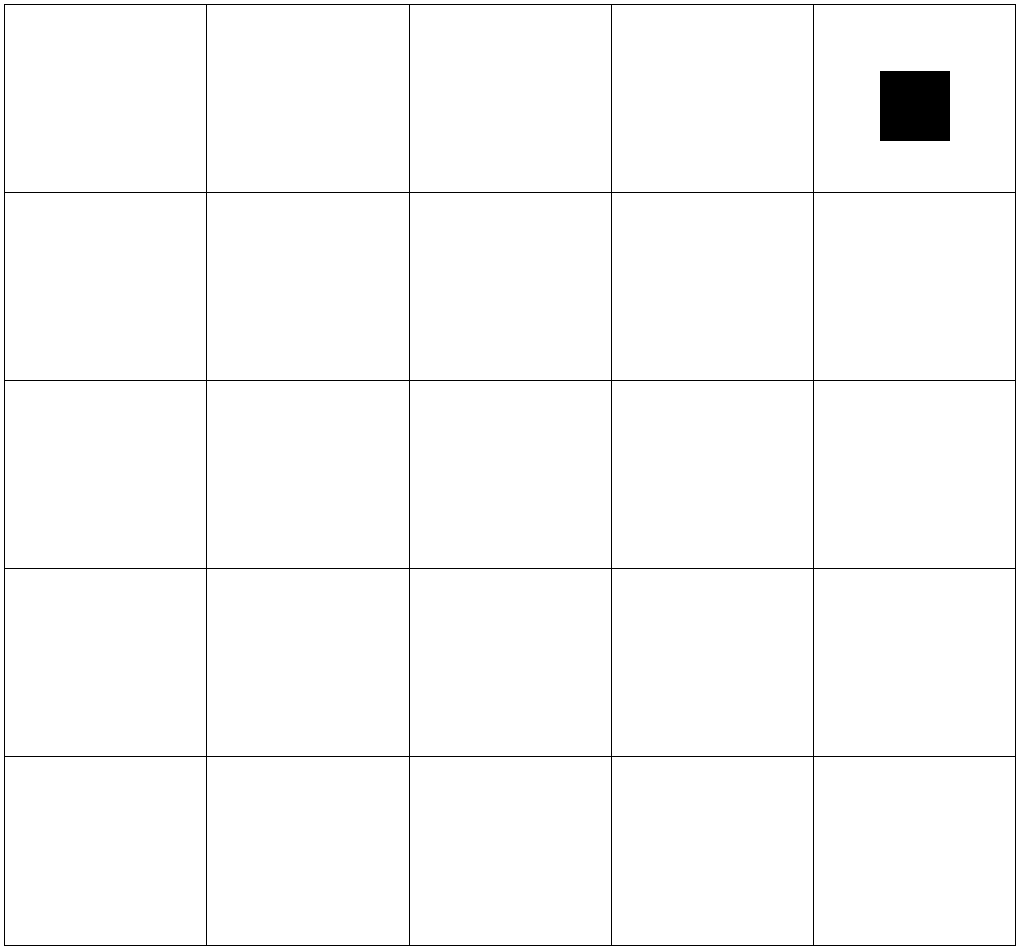
TD(0) – Propagates learning one step at a time.

Initial value of 0 everywhere:



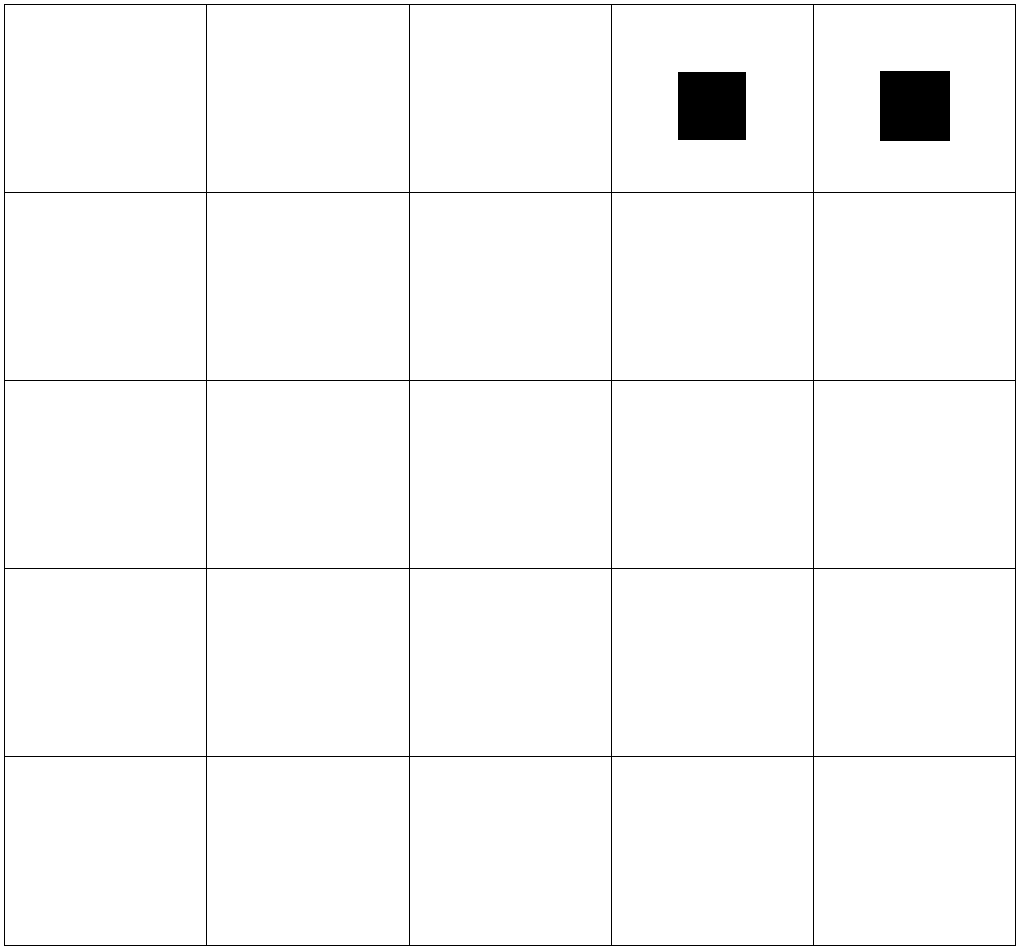
TD(0) – Propagates learning one step at a time.

After 1 episode:



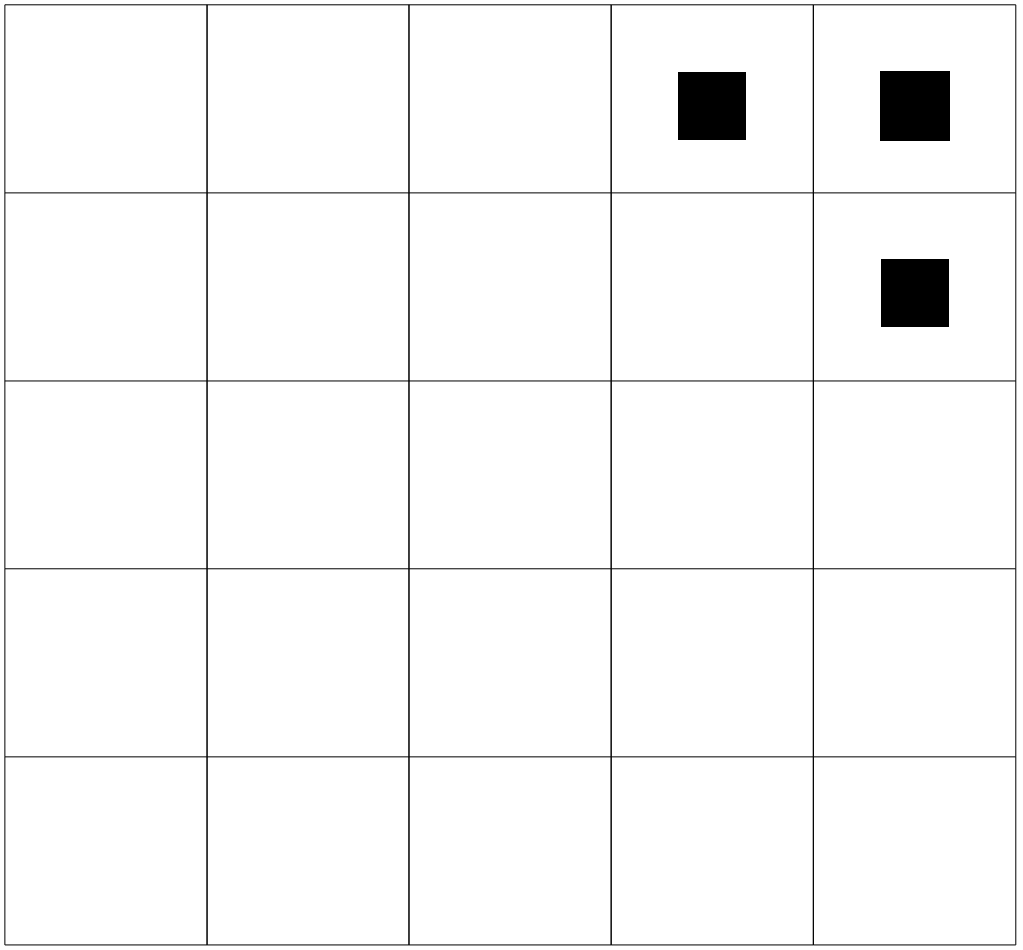
TD(0) – Propagates learning one step at a time.

After 2 episodes:



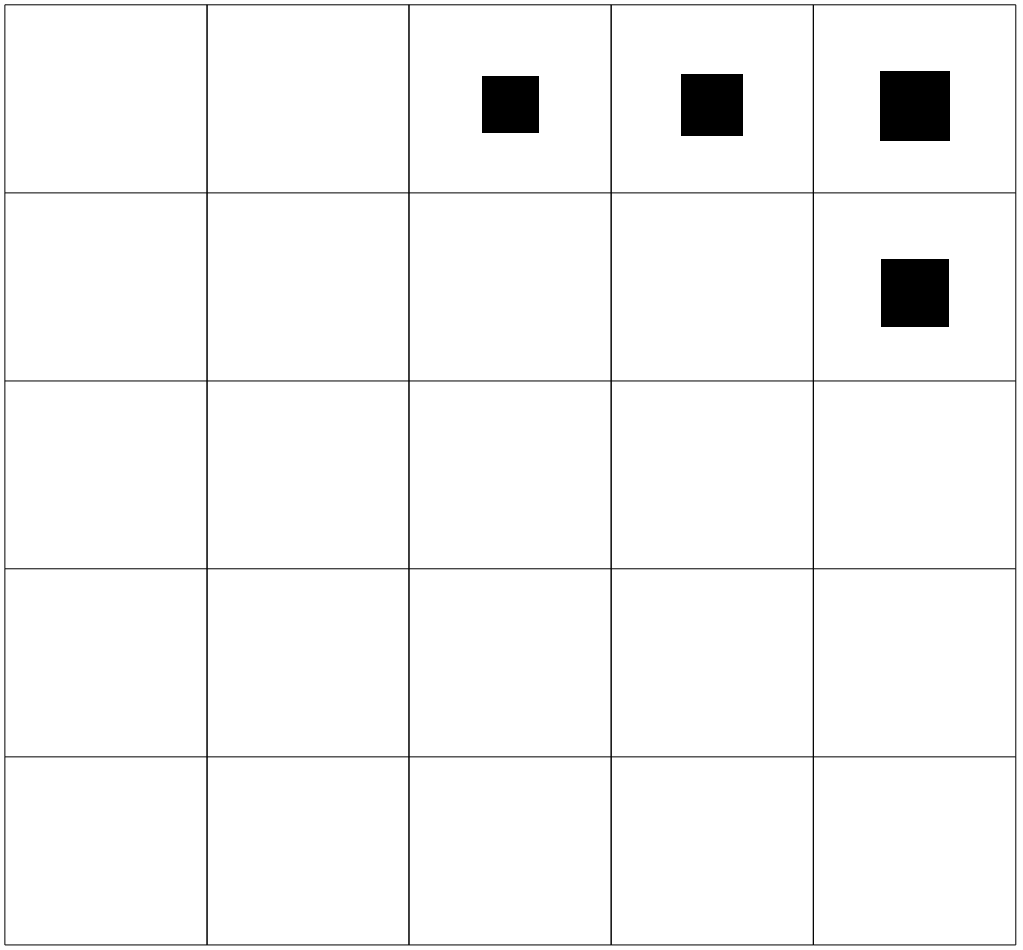
TD(0) – Propagates learning one step at a time.

After 3 episodes:



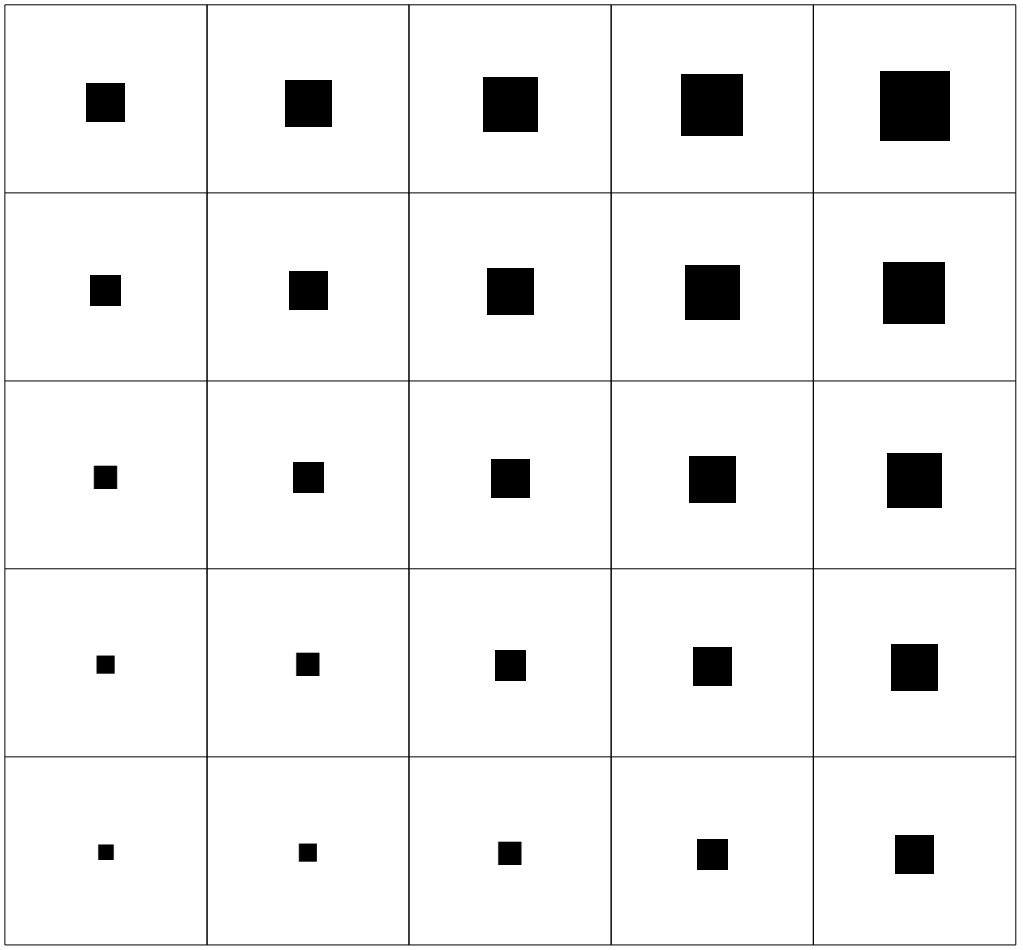
TD(0) – Propagates learning one step at a time.

After 4 episodes:



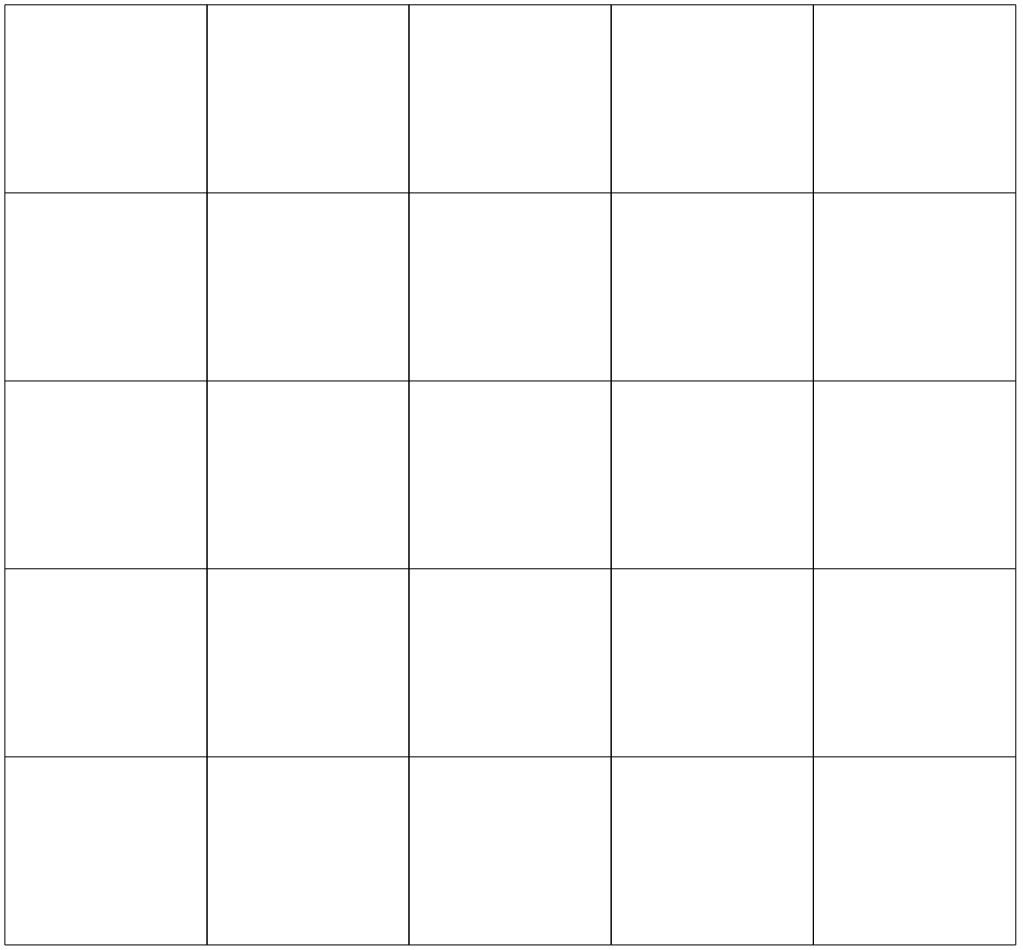
TD(0) – Propagates learning one step at a time.

It can take a while for reward to propagate to the start:



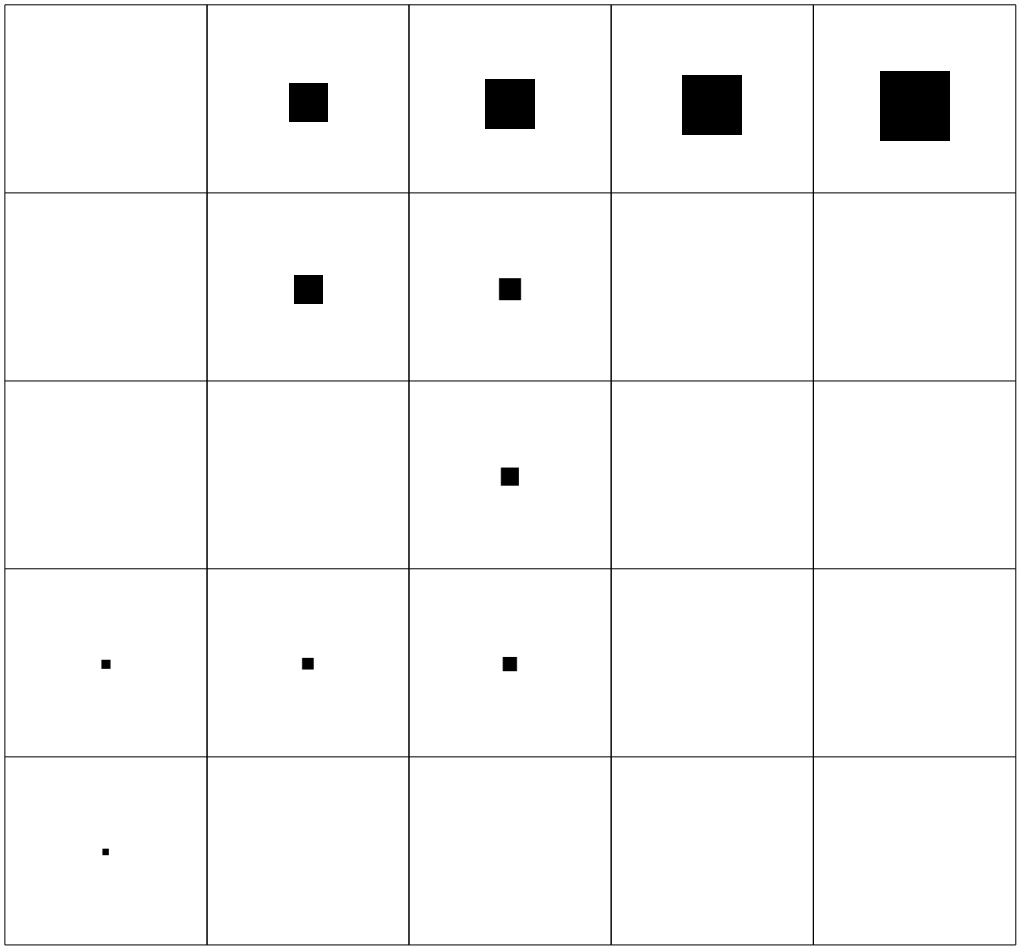
TD(λ) – Propagates learning to all immediate sources, discounted by both λ and γ.

Initial value of 0 everywhere:



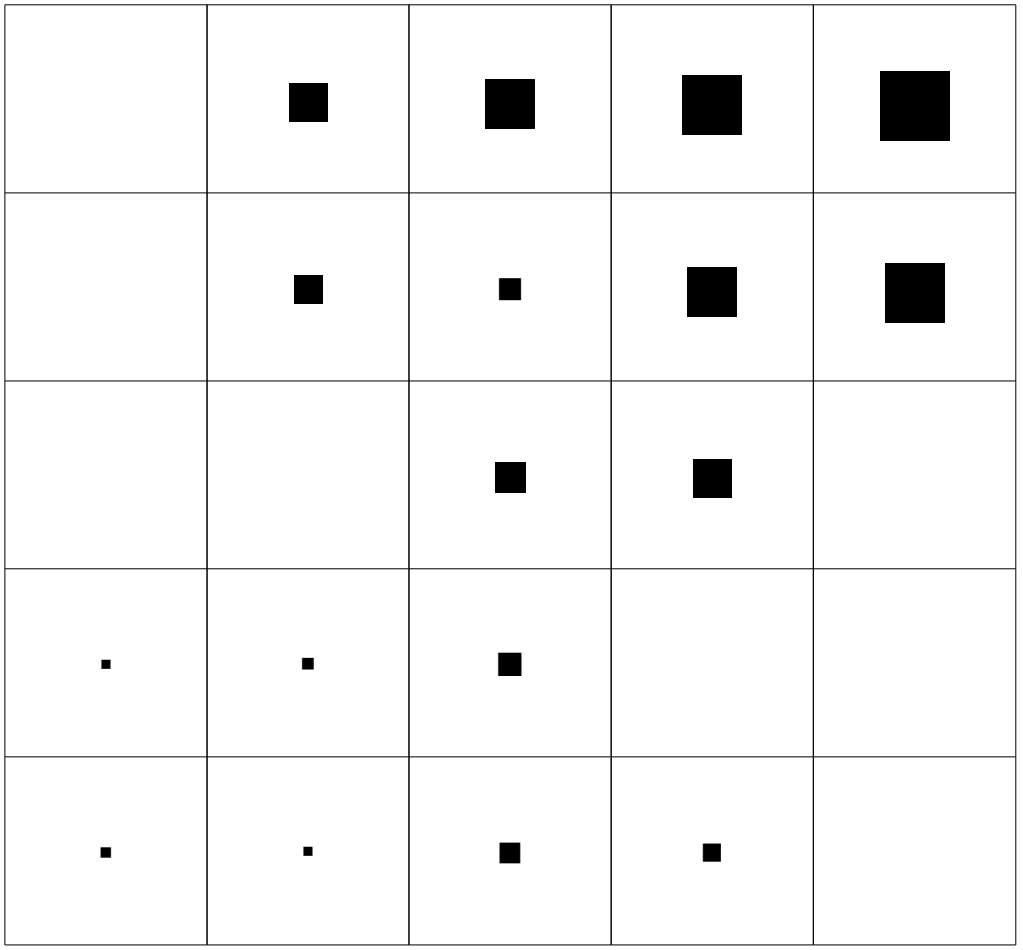
TD(λ) – Propagates learning to all immediate sources, discounted by both λ and γ.

After 1 episode:



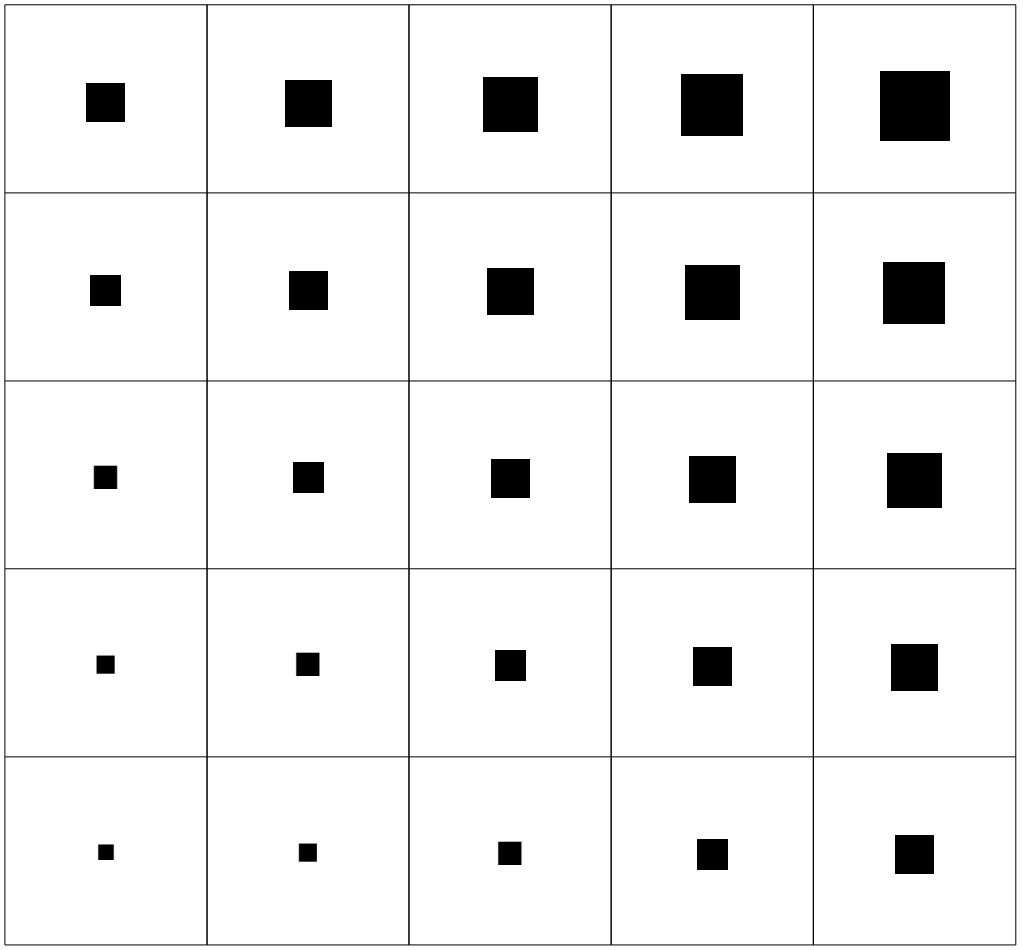
TD(λ) – Propagates learning to all immediate sources, discounted by both λ and γ.

After 2 episodes:



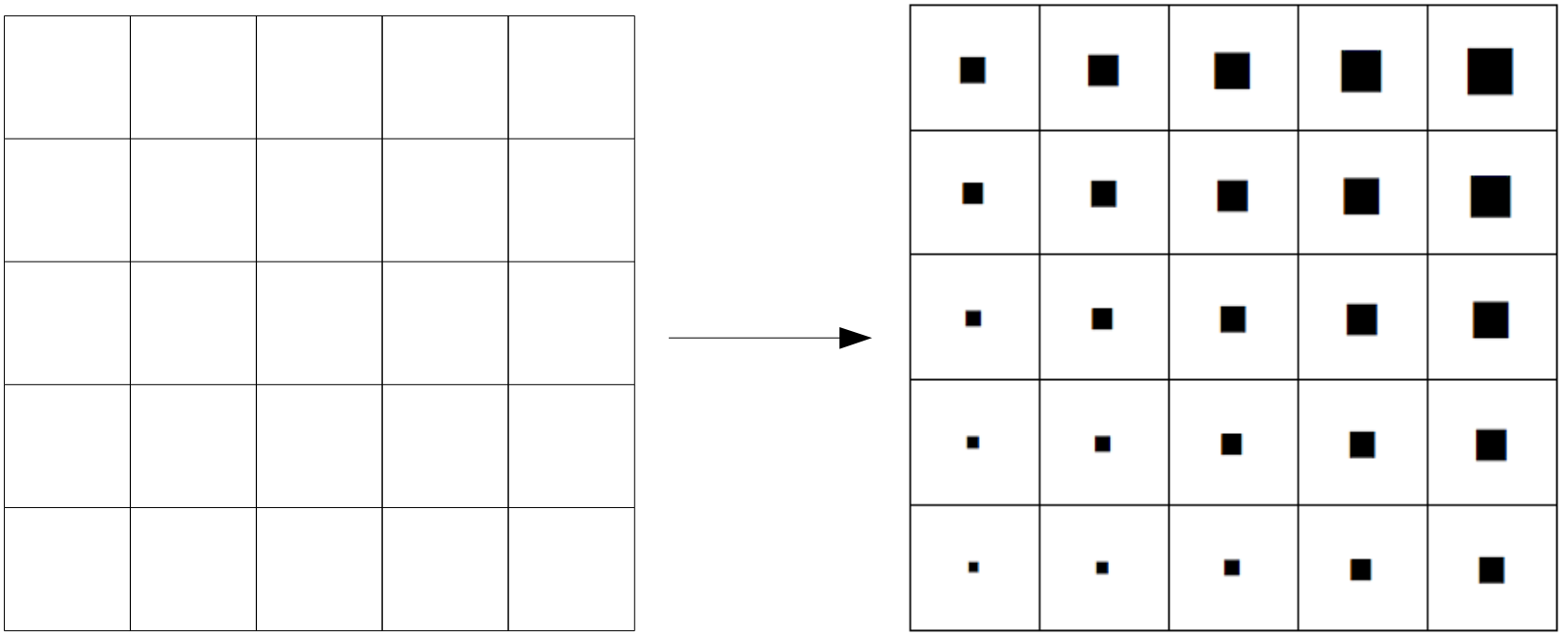
TD(λ) – Propagates learning to all immediate sources, discounted by both λ and γ.

After some time (faster than TD(0)):



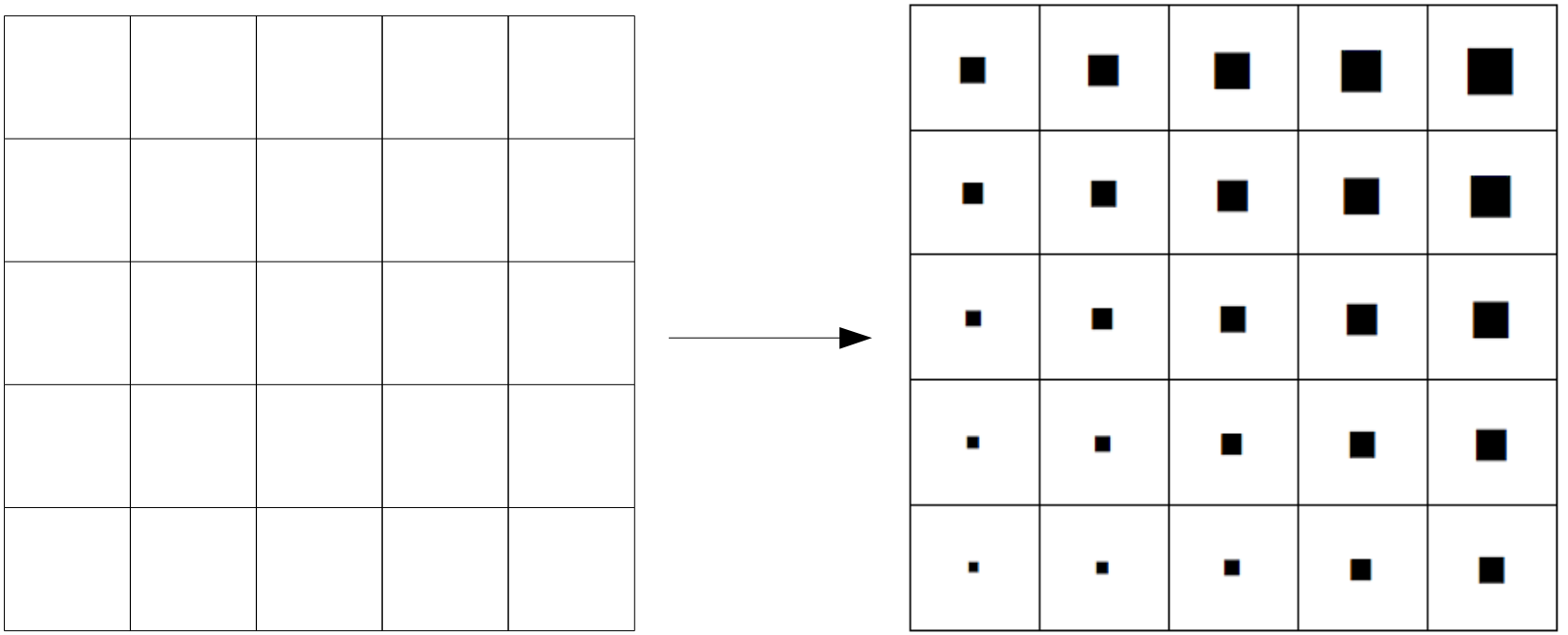
How to distribute surprise back to *potential* sources?

Can we do this in just *one* update?



How to distribute surprise back to *potential* sources?

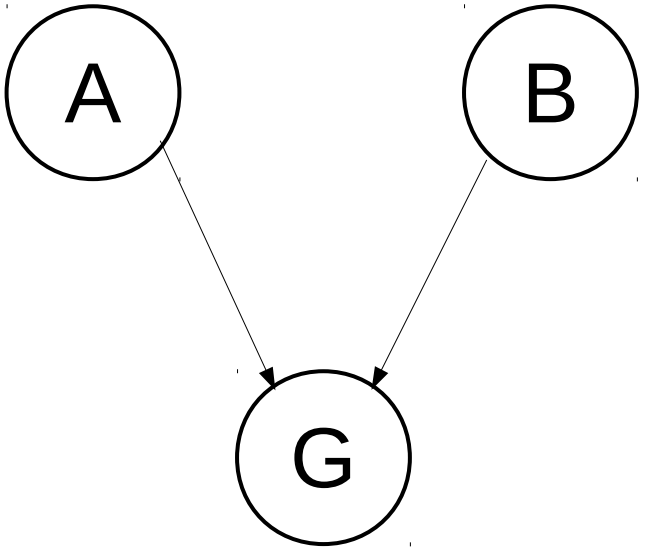
Can we do this in just *one* update?



E.g.,
perhaps one could create a model of the *average* eligibility trace at G, and use it distribute credit?

How to distribute surprise back to *potential* sources?

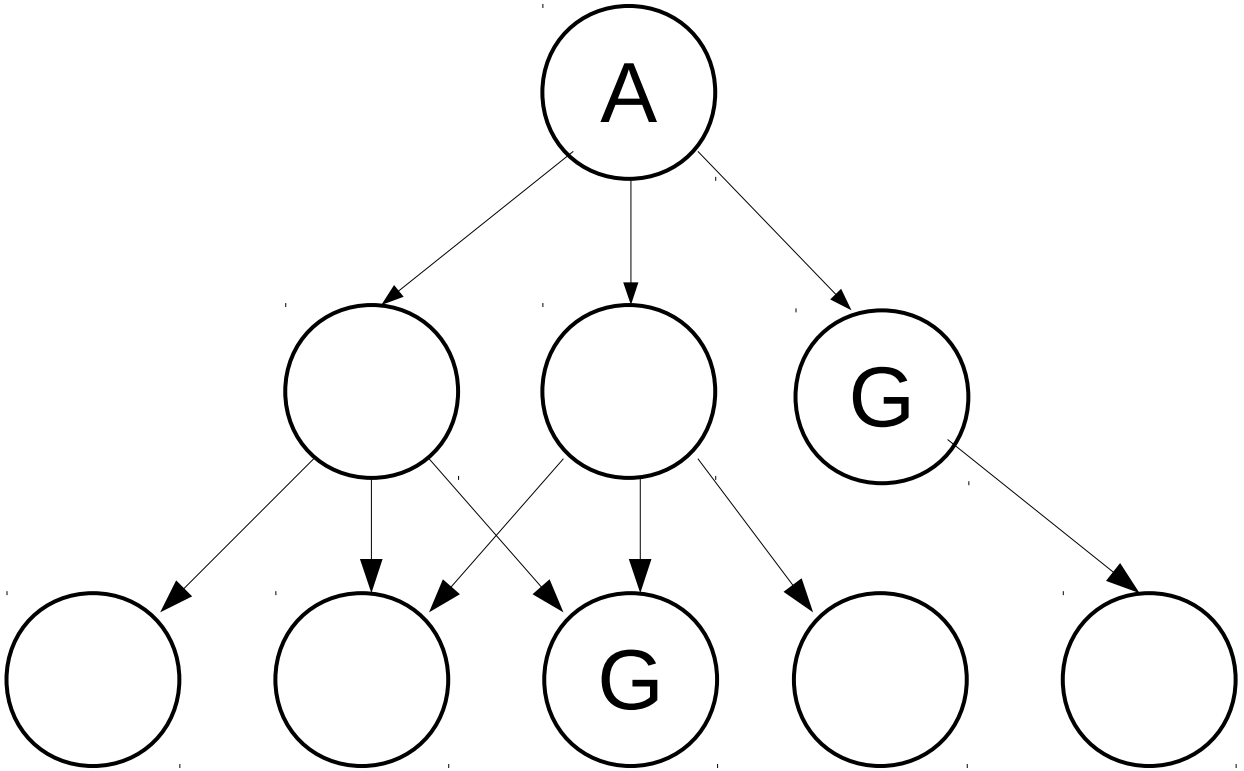
“Average” eligibility trace won’t work. Consider:



If almost all trajectories to G go through A instead of B, distributing TD errors according to the average eligibility trace will “under-update” B (and may “over-update” A).

Instead, let us take a forward view.

How does the reward at G contribute to value of A?



In proportion to expected (discounted) future occupancy of G, when starting in state A.

Algebraically:

$$\mathbf{v} = \mathbf{r} + \gamma\mathbf{P}\mathbf{r} + \gamma^2\mathbf{P}^2\mathbf{r} + \ldots$$

$$= \left(\mathbf{I} + \gamma\mathbf{P} + \gamma^2\mathbf{P}^2 + \ldots\right)\mathbf{r}$$

$$= \boxed{\left(\mathbf{I} - \gamma\mathbf{P}\right)^{-1}}\mathbf{r}$$

$$= \boxed{\mathbf{S}}\,\mathbf{r}$$

Source map
Successor representation [Dayan 1993]

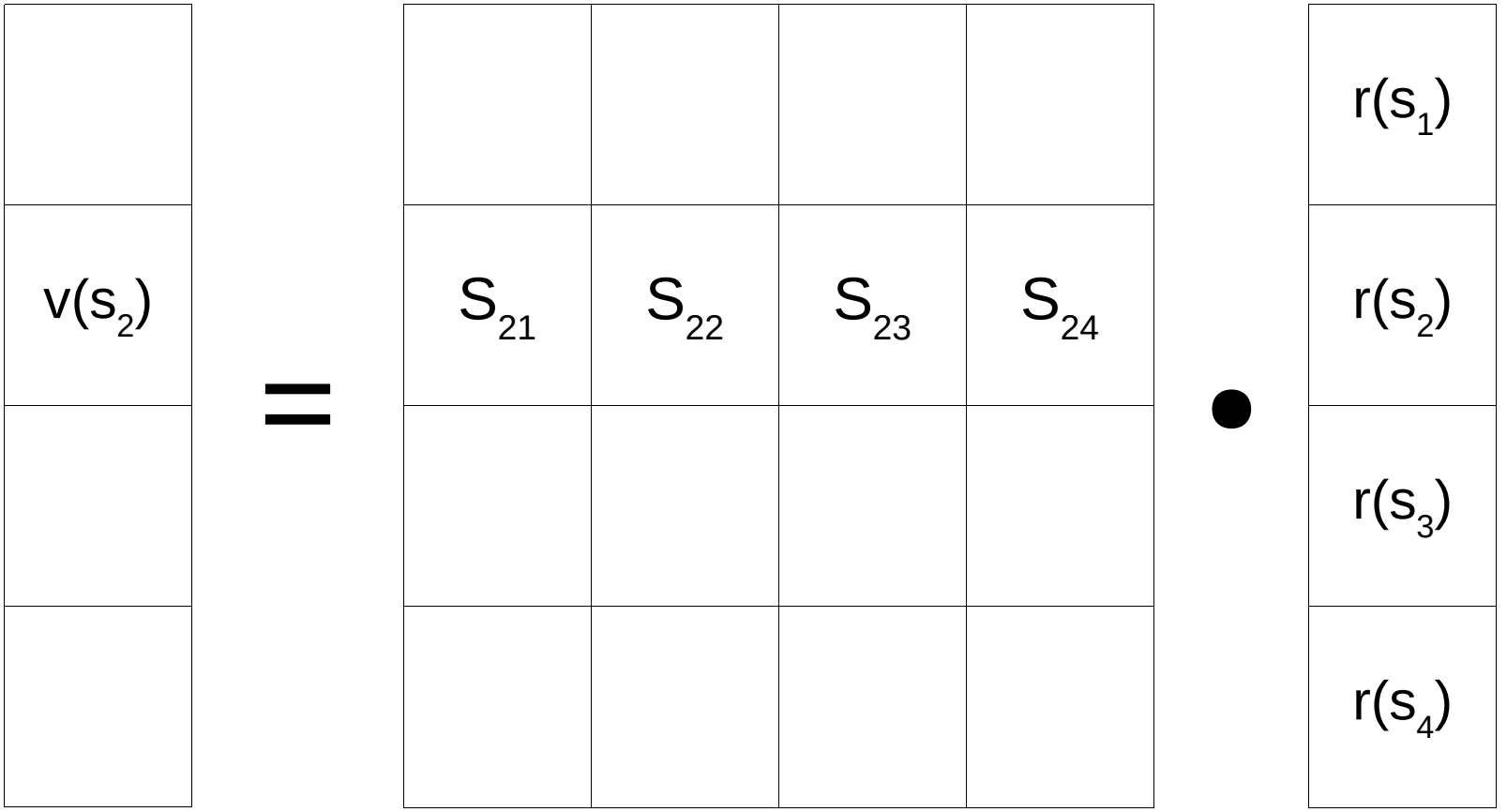
Or: Feature expectations [Abbeel, Ng 2004],
Universal option models [Yao et al. 2014],
Successor features [Barreto et al. 2017]

$$\mathbf{v} = \mathbf{S} \cdot \mathbf{r}$$

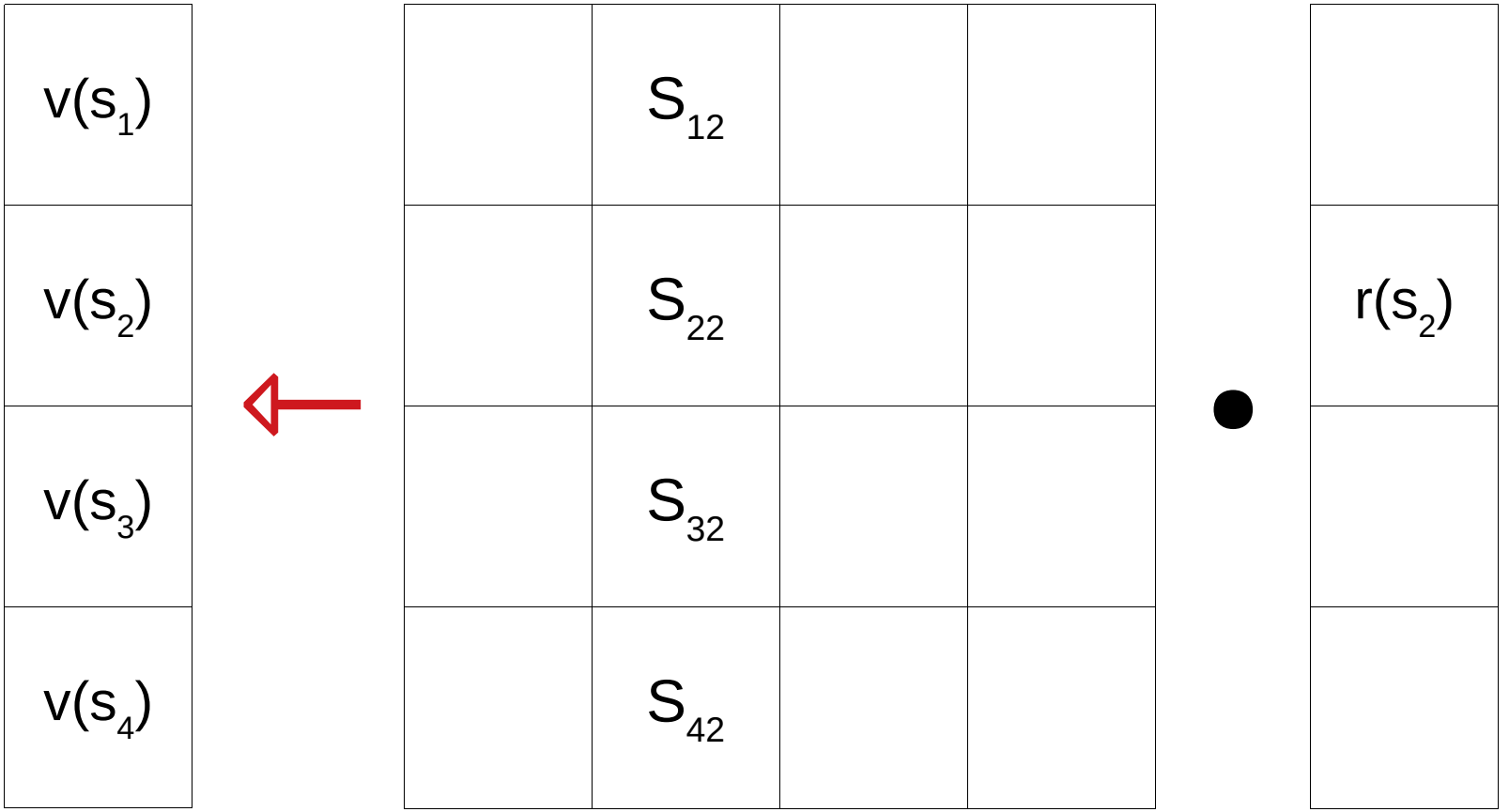
$$\begin{bmatrix} v(s_1) \\ v(s_2) \\ v(s_3) \\ v(s_4) \end{bmatrix} = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix} \cdot \begin{bmatrix} r(s_1) \\ r(s_2) \\ r(s_3) \\ r(s_4) \end{bmatrix}$$

Matrix of Expected Discounted Future State Occupancy

Successor Representation



Source Traces



Presentation Outline

1) Motivation & Basic Intuition / Derivation

[TD(0), Eligibility Traces, Successor Representation]

2) Selected Results / Contributions

[Source Learning Algorithm, Convergence, TD-Source]

3) Extension

[“S”-function as a GVF on state sets]

Presentation Outline

1) Motivation & Basic Intuition / Derivation

[TD(0), Eligibility Traces, Successor Representation]

2) Selected Results / Contributions

[Source Learning Algorithm, Convergence, TD-Source]

3) Extension

[“S”-function as a GVF on state sets]

Source Learning Algorithm

Instead of the regular TD update, do a “source update”:

Model of Source Trace (Vector)

TD Error (Real Number)

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \alpha_n [\mathbf{S}_0]_{\cdot\langle n\rangle} \left(r(s_{\langle n\rangle}) + \gamma v_n(s_{\langle n+1\rangle}) - v_n(s_{\langle n\rangle}) \right)$$

Source Learning Algorithm

Instead of the regular TD update, do a “source update”:

Model of Source Trace (Vector)

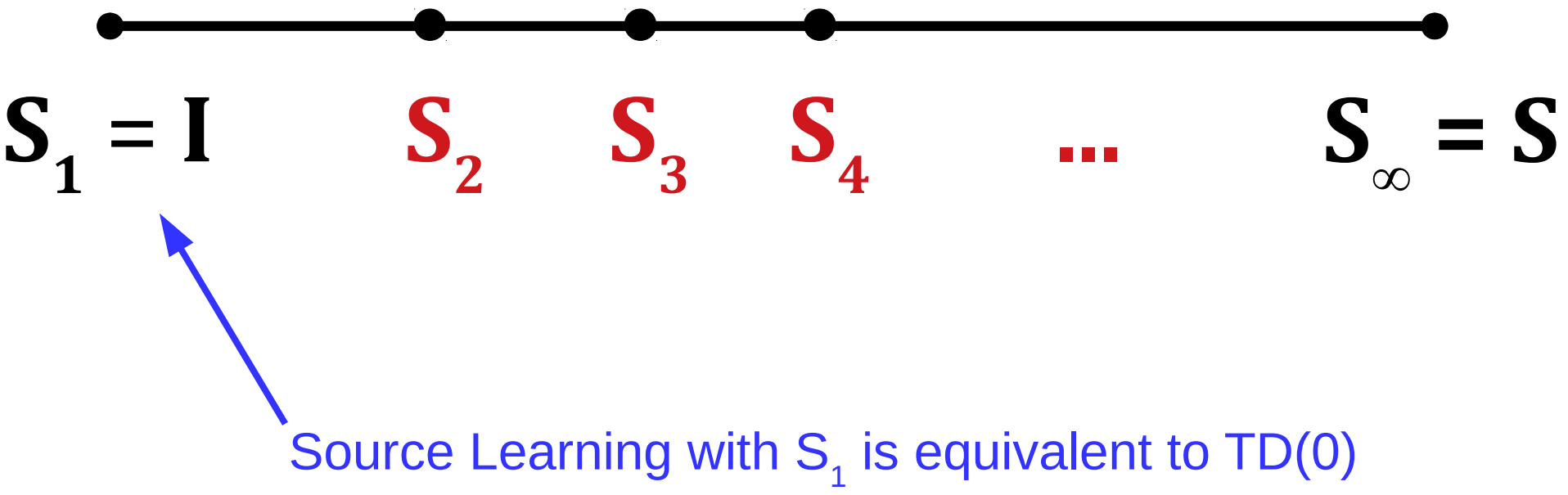
TD Error (Real Number)

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \alpha_n [\mathbf{S}_0]_{\cdot\langle n\rangle} \left(r(s_{\langle n\rangle}) + \gamma v_n(s_{\langle n+1\rangle}) - v_n(s_{\langle n\rangle}) \right)$$

Converges, so long as $\left\| \left(\mathbf{I} - \mathbf{S}^{-1} \underbrace{\mathbf{S}_0}_{\text{Model Source Matrix}} \right) \right\| \leq \gamma.$

Speed of Convergence

We will interpolate between TD(0) and Source Learning, by defining intermediate source models:



Speed of Convergence

Concretely, these are “partial” source traces:

TD(0)

$$\mathbf{S}_1 = \mathbf{I}$$

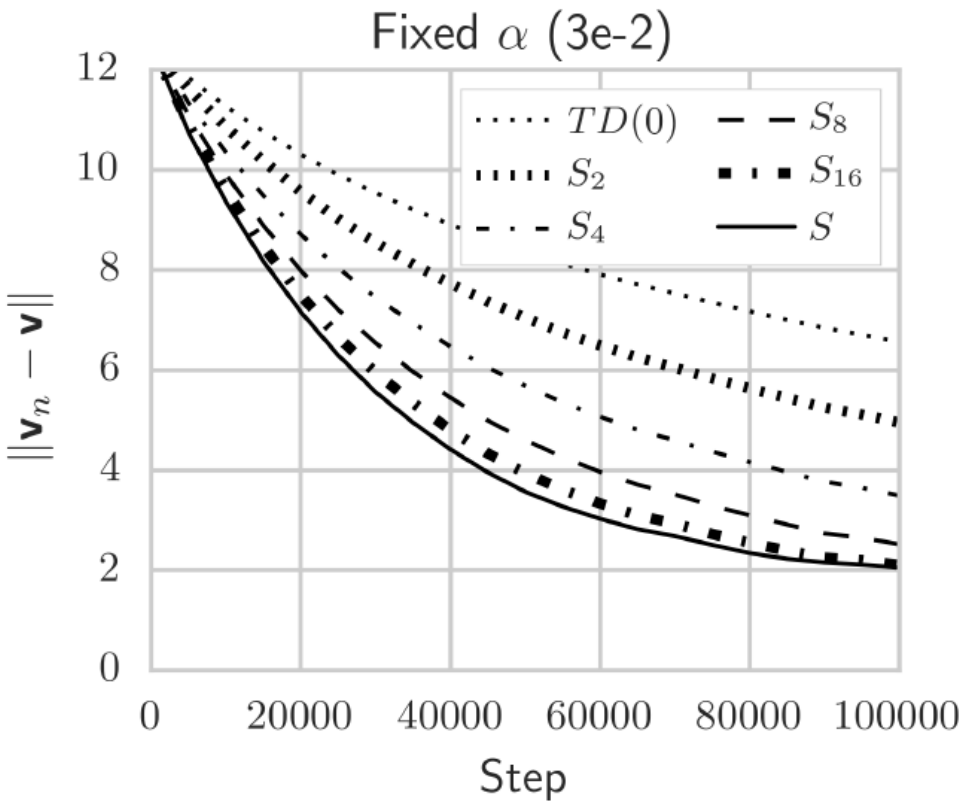
$$\mathbf{S}_2 = \mathbf{I} + \gamma\mathbf{P}$$

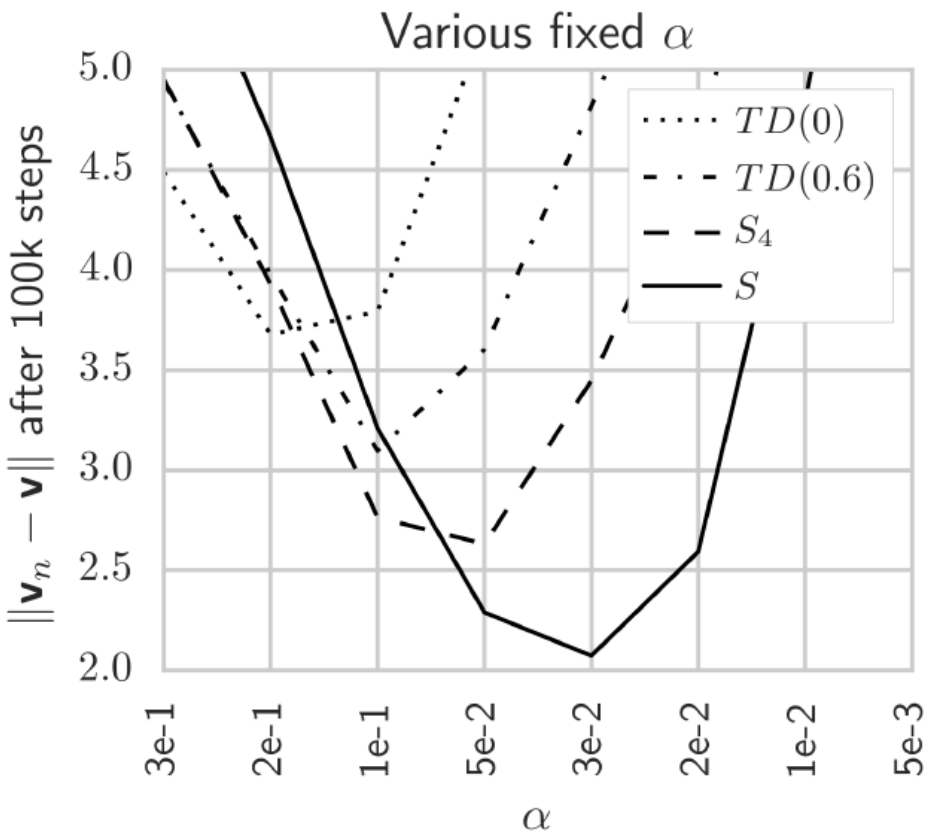
$$\mathbf{S}_3 = \mathbf{I} + \gamma\mathbf{P} + \gamma^2\mathbf{P}^2$$

...

$$\mathbf{S} = \mathbf{I} + \gamma\mathbf{P} + \gamma^2\mathbf{P}^2 + \ldots$$

Speed of Convergence





Learning the Source Model

Learning the Source Model

Existing algorithm (Dayan 1993):

Treat each column of **S** (i.e., each source trace) as the value function in an MDP where reward is 0 everywhere, except at the state corresponding to that column, where reward is 1

==> corresponds to discounted future occupancy

==> row-wise recurrence

Learning the Source Model

Existing algorithm (Dayan 1993):

Treat each column of **S** (i.e., each source trace) as the value function in an MDP where reward is 0 everywhere, except at the state corresponding to that column, where reward is 1

==> corresponds to discounted future occupancy

==> row-wise recurrence

1	**0**	**0**	**0**
0	**1**	**0**	**0**
0	**0**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Existing algorithm (Dayan 1993):

Treat each column of **S** (i.e., each source trace) as the value function in an MDP where reward is 0 everywhere, except at the state corresponding to that column, where reward is 1

==> corresponds to discounted future occupancy

==> row-wise recurrence

1	**0**	**0**	**0**
0	**1**	**0**	**0**
0	**0**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Existing algorithm (Dayan 1993):

Treat each column of **S** (i.e., each source trace) as the value function in an MDP where reward is 0 everywhere, except at the state corresponding to that column, where reward is 1

==> corresponds to discounted future occupancy

==> row-wise recurrence

1	**0**	**0**	**0**
0	**1**	**0**	**0**
0.1	**0**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Existing algorithm (Dayan 1993):

Treat each column of **S** (i.e., each source trace) as the value function in an MDP where reward is 0 everywhere, except at the state corresponding to that column, where reward is 1

==> corresponds to discounted future occupancy

==> row-wise recurrence

1	**0**	**0**	**0**
0	**1**	**0**	**0**
0.1	**0**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Existing algorithm (Dayan 1993):

Treat each column of **S** (i.e., each source trace) as the value function in an MDP where reward is 0 everywhere, except at the state corresponding to that column, where reward is 1

==> corresponds to discounted future occupancy

==> row-wise recurrence

1	**0**	**0**	**0**
0	**1**	**0**	**0**
0.1	**0**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Existing algorithm (Dayan 1993):

Treat each column of **S** (i.e., each source trace) as the value function in an MDP where reward is 0 everywhere, except at the state corresponding to that column, where reward is 1

==> corresponds to discounted future occupancy

==> row-wise recurrence

1	**0.1**	**0**	**0**
0	**1**	**0**	**0**
0.1	**0**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Novel algorithm:

Accumulate traces in the same way as eligibility traces are accumulated in TD(λ), but include an importance sampling ratio.

==> importance sampling “fixes” the average eligibility trace

==> column-wise recurrence

Learning the Source Model

Novel algorithm:

Accumulate traces in the same way as eligibility traces are accumulated in TD(λ), but include an importance sampling ratio.

==> importance sampling “fixes” the average eligibility trace

==> column-wise recurrence

1	**0**	**0**	**0**
0	**1**	**0**	**0**
0	**0**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Novel algorithm:

Accumulate traces in the same way as eligibility traces are accumulated in TD(λ), but include an importance sampling ratio.

==> importance sampling “fixes” the average eligibility trace

==> column-wise recurrence

1	**0**	**0**	**0**
0	**1**	**0**	**0**
0	**0**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Novel algorithm:

Accumulate traces in the same way as eligibility traces are accumulated in TD(λ), but include an importance sampling ratio.

==> importance sampling “fixes” the average eligibility trace

==> column-wise recurrence

1	**0**	**0**	**0**
0	**1**	**0**	**0**
0.09	**0**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Novel algorithm:

Accumulate traces in the same way as eligibility traces are accumulated in TD(λ), but include an importance sampling ratio.

==> importance sampling “fixes” the average eligibility trace

==> column-wise recurrence

1	**0**	**0**	**0**
0	**1**	**0**	**0**
0.09	**0**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Novel algorithm:

Accumulate traces in the same way as eligibility traces are accumulated in TD(λ), but include an importance sampling ratio.

==> importance sampling “fixes” the average eligibility trace

==> column-wise recurrence

1	**0**	**0**	**0**
0	**1**	**0**	**0**
0.09	**0**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Novel algorithm:

Accumulate traces in the same way as eligibility traces are accumulated in TD(λ), but include an importance sampling ratio.

==> importance sampling “fixes” the average eligibility trace

==> column-wise recurrence

1	**0.11**	**0**	**0**
0	**1**	**0**	**0**
0.09	**0.01**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Dayan's algorithm (row-wise):

+ doesn't require importance sampling

1	0.1	0	0
0	1	0	0
0.1	0	1	0
0	0	0	1

Novel algorithm (column-wise):

+ uses most recent information

1	0.11	0	0
0	1	0	0
0.09	0.01	1	0
0	0	0	1

Learning the Source Model

Why not combine the two?

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Learning the Source Model

Why not combine the two?

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Learning the Source Model

Why not combine the two?

1	0	0	0
0	1	0	0
0.19	0	1	0
0	0	0	1

Learning the Source Model

Why not combine the two?

1	**0**	**0**	**0**
0	**1**	**0**	**0**
0.19	**0**	**1**	**0**
0	**0**	**0**	**1**

Learning the Source Model

Why not combine the two?

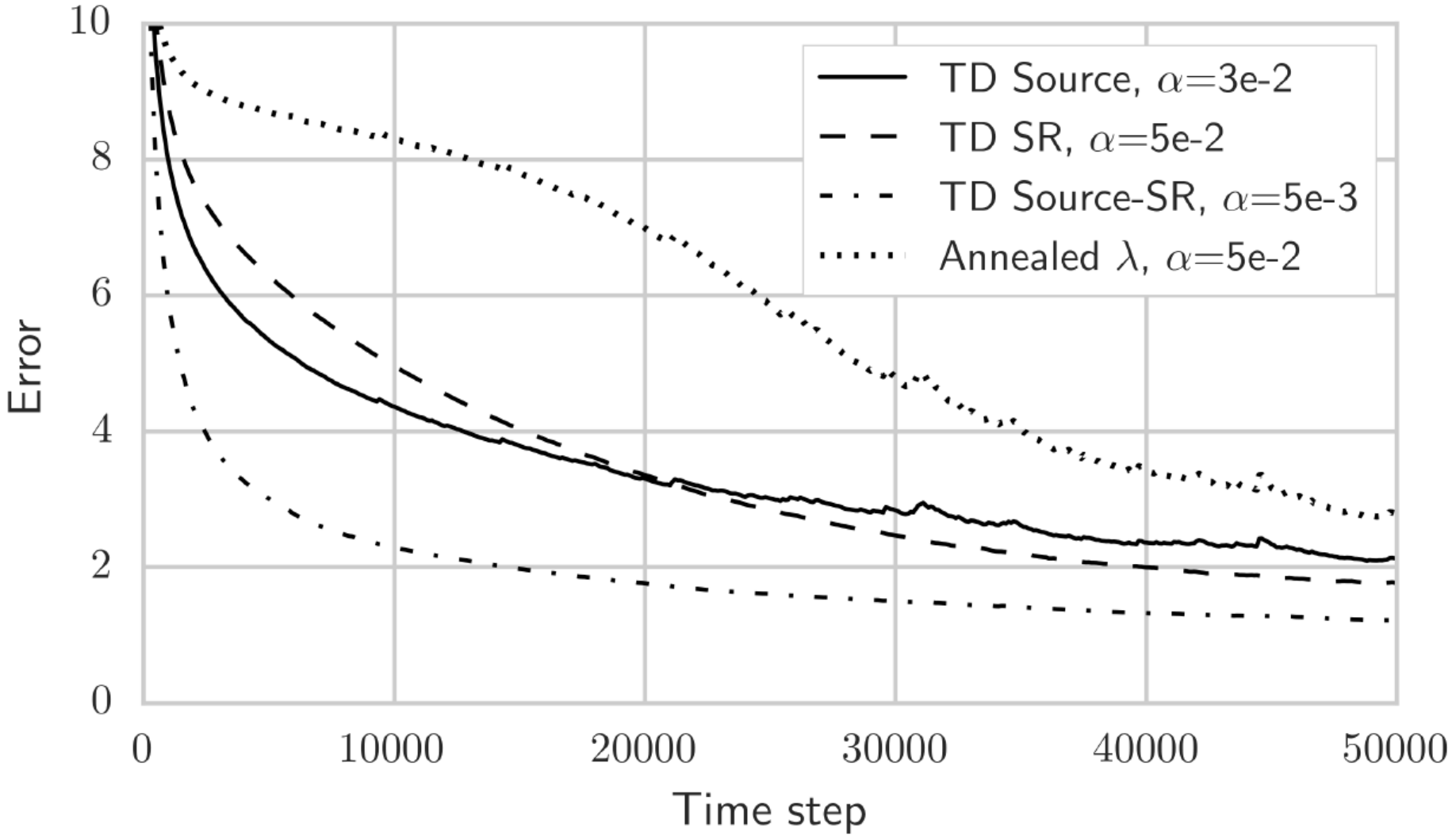
1	0	0	0
0	1	0	0
0.19	0	1	0
0	0	0	1

Learning the Source Model

Why not combine the two?

1	0.21	0	0
0	1	0	0
0.19	0.02	1	0
0	0	0	1

Learning the Source Model



Presentation Outline

1) Motivation & Basic Intuition / Derivation

[TD(0), Eligibility Traces, Successor Representation]

2) Selected Results / Contributions

[Source Learning Algorithm, Convergence, TD-Source]

3) Extension

[“S”-function as a GVF on state sets]

Presentation Outline

1) Motivation & Basic Intuition / Derivation

[TD(0), Eligibility Traces, Successor Representation]

2) Selected Results / Contributions

[Source Learning Algorithm, Convergence, TD-Source]

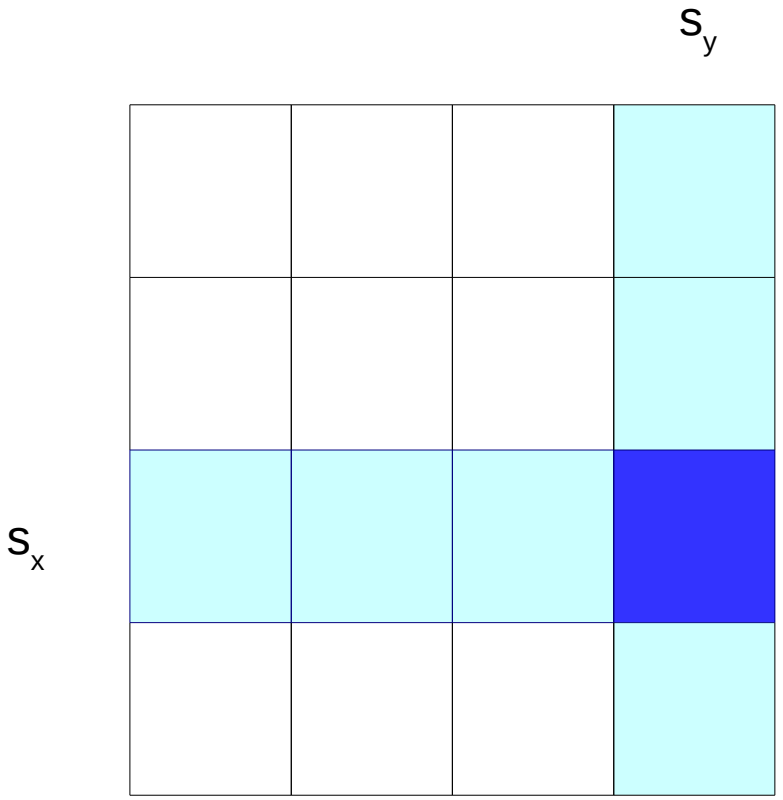
3) Extension

[“S”-function as a GVF on state sets]

The "*S*" Function

- *S* stands for "source" or "successor"
- For tabular policy valuation, this is one cell of **S**:

$S(s_x, s_y) = \mathbf{S}_{xy}$



The "*S*" Function

Let us relax restrictions:

- Policy valuation ---> Control
- *S*(p, q) is now S^{π}(p, q) --- *future occupancy depends on policy!*
- Can learn S(p, q) off-policy as a GVF
  (i.e., w/ Horde, Sutton et al. 2011, or w/ a UVFA, Schaul et al. 2015)

The "*S*" Function

But policy space is huge!

The "*S*" Function

But policy space is huge!

1. Focus on "interesting" policies, like the shortest path between p and q.

   - Can be useful w/r/t subgoal or bottleneck states (cf. UOMs)

The "*S*" Function

But policy space is huge!

1. Focus on "interesting" policies, like the shortest path between p and q.

   - Can be useful w/r/t subgoal or bottleneck states (cf. UOMs)

2. Focus on policy independent cases---where S(p, q) is static across policies.

In both cases, *S* is not only a measure of causality, but also control!

The "*S*" Function

Let us relax restrictions:

- Policy valuation ---> Control
- Simple state space ---> Huge state space requiring function approximation

The "*S*" Function

- But *S*(p, q), where p and q are state representations will typically be 0!

The "*S*" Function

- But *S*(p, q), where p and q are state representations will typically be 0!
- Need the ability to abstract states --- i.e., work with *sets* of states. See *Li, Walsh, Littman 2006*.
- "Balls" of states in R^N are not enough.
- We want meaningful, descriptive sets, like "it is raining", "I am outside", and "I am wet", and the ability to take unions, intersections, differences.

The "*S*" Function

- Then *S*(p, q) would allow us to make complex statements about causality and control!

[It is an "interesting" GVF]

The "*S*" Function

- Then *S*(p, q) would allow us to make complex statements about causality and control!

[It is an "interesting" GVF]

- E.g., if added a γ parameter:

S^{STATIC}("Raining", "Sunny", [large γ]) = small

"It is unusual for it to be Sunny shortly after it is Rainy."