Eligibility traces are

- Another way of interpolating between MC and TD methods
- A way of implementing compound $\lambda$-return targets
- A basic mechanistic idea — a short-term, fading memory
- A new style of algorithm development/analysis
  - the forward-view $\leftrightarrow$ backward-view transformation
- Forward view:
  - conceptually simple — good for theory, intuition
- Backward view:
  - computationally congenial implementation of the f. view
Unified View

Temporal-difference learning

Dynamic programming

Multi-step bootstrapping

Height (depth) of backup

Monte Carlo

Width of backup

Exhaustive search
Recall \( n \)-step targets

- For example, in the episodic case, with linear function approximation:
  - 2-step target:
    \[
    G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 \theta_{t+1}^\top \phi_{t+2}
    \]
  - \( n \)-step target:
    \[
    G_t^{(n)} = R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \theta_{t+n-1}^\top \phi_{t+n}
    \]
    with \( G_t^{(n)} = G_t \) if \( t + n \geq T \)
Any set of update targets can be averaged to produce new *compound* update targets

- For example, half a 2-step plus half a 4-step

\[ U_t = \frac{1}{2} G_t^{(2)} + \frac{1}{2} G_t^{(4)} \]

- Called a compound backup
- Draw each component
- Label with the weights for that component
The $\lambda$-return is a compound update target

- The $\lambda$-return a target that averages all $n$-step targets
  - each weighted by $\lambda^{n-1}$

\[ G_t^\lambda \doteq (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} \]
\[ G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t \]

Until termination

After termination
Relation to TD(0) and MC

- The $\lambda$-return can be rewritten as:

$$G_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t$$

Until termination

After termination

- If $\lambda = 1$, you get the MC target:

$$G_t^{\lambda} = (1 - 1) \sum_{n=1}^{T-t-1} 1^{n-1} G_t^{(n)} + 1^{T-t-1} G_t = G_t$$

- If $\lambda = 0$, you get the TD(0) target:

$$G_t^{\lambda} = (1 - 0) \sum_{n=1}^{T-t-1} 0^{n-1} G_t^{(n)} + 0^{T-t-1} G_t = G_t^{(1)}$$
The off-line $\lambda$-return “algorithm”

- Wait until the end of the episode (offline)
- Then go back over the time steps, updating

$$
\theta_{t+1} = \theta_t + \alpha \left[ G_t^\lambda - \hat{v}(S_t, \theta_t) \right] \nabla \hat{v}(S_t, \theta_t), \quad t = 0, \ldots, T - 1
$$
The $\lambda$-return alg performs similarly to $n$-step algs on the 19-state random walk (Tabular).

Intermediate $\lambda$ is best (just like intermediate $n$ is best) $\lambda$-return slightly better than $n$-step.
The forward view looks forward from the state being updated to future states and rewards.
The backward view looks back to the recently visited states (marked by eligibility traces)

- Shout the TD error backwards
- The traces fade with temporal distance by $\gamma \lambda$
Here we are marking state-action pairs with a replacing eligibility trace.
Eligibility traces (mechanism)

- The forward view was for theory
- The backward view is for mechanism

New memory vector called eligibility trace \( e_t \in \mathbb{R}^n \geq 0 \)

- On each step, decay each component by \( \gamma \lambda \) and increment the trace for the current state by 1
- Accumulating trace

\[
e_0 \overset{\triangle}{=} 0, \\
e_t \overset{\triangle}{=} \nabla \hat{v}(S_t, \theta_t) + \gamma \lambda e_{t-1}
\]
The Semi-gradient TD($\lambda$) algorithm

$$
\theta_{t+1} = \theta_t + \alpha \delta_t e_t
$$

$$
\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \theta_t) - \hat{v}(S_t, \theta_t)
$$

$$
e_0 = 0,
$$

$$
e_t = \nabla \hat{v}(S_t, \theta_t) + \gamma \lambda e_{t-1}
$$
TD(\(\lambda\)) performs similarly to offline \(\lambda\)-return alg. but slightly worse, particularly at high \(\alpha\).
The online $\lambda$-return algorithm performs best of all.

For comparison, the $\lambda = 0$ line is the same for both methods.

The derivation of true on-line TD(\(\lambda\)) is a little too complex to present here (see the next section and the appendix to the paper by van Seijen et al., in press) but its strategy is simple. The sequence of weight vectors produce by the on-line $\lambda$-return algorithm can be arranged in a triangle:

\[
\begin{array}{cccc}
\lambda_0 & \lambda_1 & \lambda_2 & \lambda_3 \\
\lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 \\
\lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 \\
\lambda_3 & \lambda_4 & \lambda_5 & \lambda_6 \\
\lambda_4 & \lambda_5 & \lambda_6 & \lambda_7 \\
\lambda_5 & \lambda_6 & \lambda_7 & \lambda_8 \\
\lambda_6 & \lambda_7 & \lambda_8 & \lambda_9 \\
\lambda_7 & \lambda_8 & \lambda_9 & \lambda_{10} \\
\end{array}
\]

One row of this triangle is produced on each time step. Really only the weight vectors on the diagonal, the $\lambda_t$, need to be produced by the algorithm. The first, $\lambda_0$, is the input, the last, $\lambda_T$, is the output, and each weight vector along the way, $\lambda_t$, plays a role in bootstrapping in the n-step returns of the updates. In the final algorithm the diagonal weight vectors are renamed without a superscript, $\lambda_t$. The strategy then is to find a compact, efficient way of computing each $\lambda_t$ from the one before. If this is done, for the linear case in which $\hat{v}(s, \lambda_t) = \lambda_t$, then we arrive at the true online TD(\(\lambda\)) algorithm:

\[
\lambda_{t+1} = \lambda_t + \alpha_t e_t \lambda_t \hat{v}(s_t, \lambda_t) - \lambda_t \hat{v}(s_t, \lambda_t - 1),
\]

where we have used the shorthand $\alpha_t = (S_t, t)$, $t$ is defined as in TD(\(\lambda\)) (12.6), and $\lambda_t$ is defined as in TD(\(\lambda\)).
The online $\lambda$-return alg uses a truncated $\lambda$-return as its target

$$G_{t}^{\lambda|h} \doteq (1 - \lambda) \sum_{n=1}^{h-t-1} \lambda^{n-1} G_{t}^{(n)} + \lambda^{h-t-1} G_{t}^{(h-t)}, \quad 0 \leq t < h \leq T$$

There is a separate $\theta$ sequence for each $h$!
The online $\lambda$-return algorithm

$$\theta_{t+1}^h \triangleq \theta_t^h + \alpha \left[ G_t^{\lambda|h} - \hat{v}(S_t, \theta_t^h) \right] \nabla \hat{v}(S_t, \theta_t^h)$$

$h = 1$:

$$\theta_1^1 \triangleq \theta_0^1 + \alpha \left[ G_0^{\lambda|1} - \hat{v}(S_0, \theta_0^1) \right] \nabla \hat{v}(S_0, \theta_0^1),$$

$h = 2$:

$$\theta_1^2 \triangleq \theta_0^2 + \alpha \left[ G_0^{\lambda|2} - \hat{v}(S_0, \theta_0^2) \right] \nabla \hat{v}(S_0, \theta_0^2),$$

$$\theta_2^2 \triangleq \theta_1^2 + \alpha \left[ G_1^{\lambda|2} - \hat{v}(S_1, \theta_1^2) \right] \nabla \hat{v}(S_1, \theta_1^2),$$

$h = 3$:

$$\theta_1^3 \triangleq \theta_0^3 + \alpha \left[ G_0^{\lambda|3} - \hat{v}(S_0, \theta_0^3) \right] \nabla \hat{v}(S_0, \theta_0^3),$$

$$\theta_2^3 \triangleq \theta_1^3 + \alpha \left[ G_1^{\lambda|3} - \hat{v}(S_1, \theta_1^3) \right] \nabla \hat{v}(S_1, \theta_1^3),$$

$$\theta_3^3 \triangleq \theta_2^3 + \alpha \left[ G_2^{\lambda|3} - \hat{v}(S_2, \theta_2^3) \right] \nabla \hat{v}(S_2, \theta_2^3).$$

There is a separate $\theta$ sequence for each $h$!

True online TD($\lambda$) computes just the diagonal, cheaply (for linear FA)
**True online TD(\(\lambda\))**

\[
\theta_{t+1} = \theta_t + \alpha \delta_t \mathbf{e}_t + \alpha \left( \theta_t^\top \phi_t - \theta_{t-1}^\top \phi_t \right) (\mathbf{e}_t - \phi_t)
\]

\[
\mathbf{e}_t = \gamma \lambda \mathbf{e}_{t-1} + \left( 1 - \alpha \gamma \lambda \mathbf{e}_{t-1}^\top \phi_t \right) \phi_t
\]  

*dutch trace*
Accumulating, Dutch, and Replacing Traces

All traces fade the same:

But increment differently!

![Diagram showing different types of traces: accumulating, dutch, and replacing.](image)
The simplest example of deriving a backward view from a forward view

- Monte Carlo learning of a final target
- Will derive dutch traces
- Showing the dutch traces really are not about TD
- They are about efficiently implementing online algs
The Problem:
Predict final target $Z$ with linear function approximation

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>$T-1$</th>
<th>$T$</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>$\phi_0$</td>
<td>$\phi_1$</td>
<td>$\phi_2$</td>
<td>...</td>
<td>$\phi_{T-1}$</td>
<td>$Z$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weights</td>
<td>$\theta_0$</td>
<td>$\theta_0$</td>
<td>$\theta_0$</td>
<td>...</td>
<td>$\theta_0$</td>
<td>$\theta_T$</td>
<td>$\theta_T$</td>
<td>$\theta_T$</td>
<td></td>
</tr>
<tr>
<td>Predictions</td>
<td>$\theta_0^T \phi_0$</td>
<td>$\theta_0^T \phi_1$</td>
<td>$\theta_0^T \phi_2$</td>
<td>...</td>
<td>$\theta_0^T \phi_{T-1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MC: $\theta_{t+1} = \theta_t + \alpha_t (Z - \phi_t^T \theta_t) \phi_t$, \hspace{1cm} $t = 0, \ldots, T - 1$

all done at time $T$
Computational goals

Computation per step (including memory) must be

1. **Constant.** (non-increasing with number of episodes)
2. **Proportionate.** (proportional to number of weights, or $O(n)$)
3. **Independent of span.** (not increasing with episode length) In general, the *predictive span* is the number of steps between making a prediction and observing the outcome

MC:  \[ \theta_{t+1} = \theta_t + \alpha_t \left( Z - \phi_t^\top \theta_t \right) \phi_t, \quad t = 0, \ldots, T - 1 \]

What is the span? $T$

Is MC indep of span? No
Computational goals

Computation per step (including memory) must be

1. **Constant.** (non-increasing with number of episodes)

2. **Proportionate.** (proportional to number of weights, or $O(n)$)

3. **Independent of span.** (not increasing with episode length) In general, the *predictive span* is the number of steps between making a prediction and observing the outcome.

**MC:** \[ \theta_{t+1} = \theta_t + \alpha_t \left( Z - \phi_t^\top \theta_t \right) \phi_t, \quad t = 0, \ldots, T - 1 \]

Computation and memory needed at step $T$ increases with $T \Rightarrow \text{not IoS}$
Final Result

Given:

\[ \theta_0, \phi_0, \phi_1, \phi_2, \ldots, \phi_{T-1}, Z \]

MC algorithm:

\[ \theta_{t+1} = \theta_t + \alpha_t (Z - \phi_t^\top \theta_t) \phi_t, \quad t = 0, \ldots, T - 1 \]

Equivalent independent-of-span algorithm:

\[ \begin{align*}
\theta_T & = a_{T-1} + Ze_{T-1}, \\
a_0 & = \theta_0, \text{ then } a_t = a_{t-1} - \alpha_t \phi_t \phi_t^\top a_{t-1}, \\
e_0 & = \alpha_0 \phi_0, \text{ then } e_t = e_{t-1} - \alpha_t \phi_t \phi_t^\top e_{t-1} + \alpha_t \phi_t, \\
\end{align*} \]

(t = 1, \ldots, T - 1)

Proved:

\[ \theta_T = \theta_T \]
MC: $\theta_{t+1} = \theta_t + \alpha_t (Z - \phi_t^T \theta_t) \phi_t, \quad t = 0, \ldots, T - 1$

\[
\begin{align*}
\theta_T &= \theta_{T-1} + \alpha_{T-1} (Z - \phi_{T-1}^T \theta_{T-1}) \phi_{T-1} \\
&= \theta_{T-1} + \alpha_{T-1} \phi_{T-1} (-\phi_{T-1}^T \theta_{T-1}) + \alpha_{T-1} Z \phi_{T-1} \\
&= (I - \alpha_{T-1} \phi_{T-1}^T \phi_{T-1}) \theta_{T-1} + Z \alpha_{T-1} \phi_{T-1} \\
&= F_{T-1} \theta_{T-1} + Z \alpha_{T-1} \phi_{T-1} \\
&= F_{T-1} (F_{T-2} \theta_{T-2} + Z \alpha_{T-2} \phi_{T-2}) + Z \alpha_{T-1} \phi_{T-1} \\
&= F_{T-1} F_{T-2} \theta_{T-2} + Z (F_{T-1} \alpha_{T-2} \phi_{T-2} + \alpha_{T-1} \phi_{T-1}) \\
&= F_{T-1} F_{T-2} (F_{T-3} \theta_{T-3} + Z \alpha_{T-3} \phi_{T-3}) + Z (F_{T-1} \alpha_{T-2} \phi_{T-2} + \alpha_{T-1} \phi_{T-1}) \\
&= F_{T-1} F_{T-2} F_{T-3} \theta_{T-3} + Z (F_{T-1} F_{T-2} \alpha_{T-3} \phi_{T-3} + F_{T-1} \alpha_{T-2} \phi_{T-2} + \alpha_{T-1} \phi_{T-1}) \\
&\vdots \\
&= F_{T-1} F_{T-2} \cdots F_0 \theta_0 + Z \sum_{k=0}^{T-1} F_{T-1} F_{T-2} \cdots F_{k+1} \alpha_k \phi_k \\
&= a_{T-1} + Z e_{T-1}
\end{align*}
\]

auxiliary short-term-memory vectors $a_t \in \mathbb{R}^n$, $e_t \in \mathbb{R}^n$
\[
\begin{align*}
F_{T-1}F_{T-2} \cdots F_0 \theta_0 & + Z \sum_{k=0}^{T-1} F_{T-1}F_{T-2} \cdots F_{k+1} \alpha_k \phi_k \\
& = a_{T-1} + Ze_{T-1}
\end{align*}
\]

\[
\begin{align*}
e_t & = \sum_{k=0}^{t-1} F_t F_{t-1} \cdots F_{k+1} \alpha_k \phi_k, \quad t = 0, \ldots, T - 1 \\
& = \sum_{k=0}^{t-1} F_t F_{t-1} \cdots F_{k+1} \alpha_k \phi_k + \alpha_t \phi_t \\
& = F_t \sum_{k=0}^{t-1} F_{t-1} F_{t-2} \cdots F_{k+1} \alpha_k \phi_k + \alpha_t \phi_t \\
& = F_t e_{t-1} + \alpha_t \phi_t, \quad t = 1, \ldots, T - 1 \\
& = e_{t-1} - \alpha_t \phi_t \phi_t^T e_{t-1} + \alpha_t \phi_t, \quad t = 1, \ldots, T - 1
\end{align*}
\]

\[
a_t = F_t F_{t-1} \cdots F_0 \theta_0 = F_t a_{t-1} = a_{t-1} - \alpha_t \phi_t \phi_t^T a_{t-1}, \quad t = 1, \ldots, T - 1
\]
Final Result

Given:

\[ \theta_0 \quad \phi_0, \phi_1, \phi_2, \ldots, \phi_{T-1} \quad Z \]

MC:

\[ \theta_{t+1} = \theta_t + \alpha_t (Z - \phi_t^T \theta_t) \phi_t, \quad t = 0, \ldots, T - 1 \]

Equivalent independent-of-span algorithm:

\[ \theta_T = a_{T-1} + Z e_{T-1}, \]
\[ a_0 = \theta_0, \text{ then } a_t = a_{t-1} - \alpha_t \phi_t \phi_t^T a_{t-1}, \]
\[ e_0 = \alpha_0 \phi_0, \text{ then } e_t = e_{t-1} - \alpha_t \phi_t \phi_t^T e_{t-1} + \alpha_t \phi_t, \]

Proved:

\[ \theta_T = \theta_T \]
Conclusions from the forward-backward derivation

- We have derived dutch eligibility traces from an MC update, without any TD learning.

- Dutch traces, and in fact all eligibility traces, are not about TD; they are about efficient multi-step learning.

- We can derive new non-obvious algorithms that are equivalent to obvious algorithms but have better computational properties.

- This is a different type of machine-learning result, an algorithm equivalence.
Conclusions regarding Eligibility Traces

- Provide an efficient, incremental way to combine MC and TD
  - Includes advantages of MC (better when non-Markov)
  - Includes advantages of TD (faster, comp. congenial)
- True online TD(\(\lambda\)) is new and best
  - Is exactly equivalent to online \(\lambda\)-return algorithm
- Three varieties of traces: accumulating, dutch, (replacing)
- Traces to control in on-policy and off-policy forms
- Traces do have a small cost in computation (\(\approx x2\))