12. Eligibility Traces

Notes

- The popular TD(λ) algorithm refers to the use of eligibility trace in temporal difference methods. Almost all TD methods can be combined with eligibility traces to obtain a more general method.
- Eligibility traces unify and generalise MC and TD methods, where one-step TD sits on one end of the spectrum with $\lambda = 0$ and MC methods on the other end with $\lambda = 1$. It is also a way to implement MC methods online and on continuing problems without episodes.
- The eligibility trace mechanism uses an *eligibility trace* $\mathbf{z}_t \in \mathbb{R}^d$ which parallels the weight vector $\mathbf{w}_t \in \mathbb{R}^d$. The rough idea is that when a component of $\mathbf{w}_t$ participates in producing an estimated value, the corresponding component of $\mathbf{z}_t$ is bumped up and begins to fade away. The trace-decay parameter $\lambda \in [0, 1]$ determines the rate at which the trace falls.
- The primary advantage of eligibility traces is that only a single trace vector is required rather than store the last n feature vectors. Learning also occurs continually and uniformly in time, rather than being delayed, and can thus affect behaviour immediately.
- MC methods and n-step TD methods are *forward viewing* methods, where updates are based on the next n rewards and states in the future. With eligibility traces, we can get the exact same updates with an algorithm that uses the current TD error, and this method of implementation is called *backward viewing*.
- The λ-return:
  - A valid update can be done toward any average of n-step returns for different values of n. Such composite returns possess an error reduction property and thus can be used to construct updates with guaranteed convergence. Update that averages simpler components are called *compound updates*.
  - The TD(λ) algorithm averages all n-step updates, each weighted proportionally to $\lambda^{n-1},\ \lambda \in [0, 1]$ and is normalised by $1 - \lambda$ to ensure the weights sum to 1. The λ-*return* is given by

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

    This clearly shows that $\lambda = 1$ represents the MC method and $\lambda = 0$ represents the one-step TD method.

- The *off-line λ-return algorithm* is a learning algorithm which makes no changes to the weight vector during the episode, but only at the end of the episode. The algorithm is given by

  $$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\big[G_t^\lambda - \hat{v}(S_t, \mathbf{w}_t)\big]\nabla\hat{v}(S_t, \mathbf{w}_t), \;\; t = 0, \dots, T-1.$$

  The best performance is often obtained with an intermediate value of λ.

- Note that this approach is called the theoretical, or *forward*, view of a learning algorithm. After looking forward and updating a state, we move on to the next and never have to use the preceding state again, whilst future states are viewed and processed repeatedly.

- TD(λ):
  - The TD(λ) algorithm is able to approximate the off-line λ-return algorithm quite well. It also improves on the off-line method in three ways: 1) it updates the weight vector on every step of an episode rather than only at the end, 2) its computation is equally distributed in time and not at the end of the episode, and 3) it can apply to continuing problems as well.
  - The eligibility trace $\mathbf{z}_t \in \mathbb{R}^d$ is the same size as the weight vector $\mathbf{w}_t$, where the eligibility trace is the short-term memory lasting less than the length of an episode, and the weight vector is the long-term memory accumulating over the lifetime of the system.
  - The eligibility trace is given by

    $$\begin{aligned}\mathbf{z}_{-1} &\doteq 0\\ \mathbf{z}_t &\doteq \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{v}(S_t, \mathbf{w}_t).\end{aligned}$$

    The TD(λ) update algorithm is then given with the TD error:

    $$\begin{aligned}\delta_t &= R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t)\\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t.\end{aligned}$$

  - TD(λ) is a backward viewing algorithm, where we view how much the prior states contributed to the current eligibility trace. We can say that the earlier states are given less *credit* for the TD error. Note that TD(0) refers to the one-step TD method and TD(1) refers to the MC method.
  - Comparing TD(1) to conventional MC methods, TD(1) has a few advantages to it: 1) TD(1) can be applied to discounted continuing tasks, 2) TD(1) can be performed

incrementally and online, where the algorithm can learn immediately and alter its behaviour in the same episode.

  - In general, when comparing TD(λ) to off-line λ-return algorithm, if α is selected optimally, then both algorithms perform identically at each λ value. If α is chosen to be larger than optimal, the off-line λ-return algorithm performs a little worse, but the TD(λ) algorithm performs much worse and may even be unstable.
  - Linear TD(λ) has been proved to converge in the on-policy case if the step-size parameter is reduced over time. The convergence is to a nearby weight vector that depends on λ, and not to the minimum-error weight vector. The bound on the error for the continuing discounted case is

$$\overline{\mathrm{VE}}(\mathbf{w}_\infty) \leq \frac{1-\gamma\lambda}{1-\gamma} \min_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w})$$

- n-step Truncated λ-return Methods:
  - This is a variation of the off-line λ-return algorithm, where the compound return is truncated at a horizon h. The *truncated λ-return* for time t is given by

$$\begin{aligned} G^\lambda_{t:h} &= (1-\lambda)\sum_{n=1}^{h-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{h-t-1} G_{t:h} \\ G^\lambda_{t:t=k} &= \hat{v}(S_t, \mathbf{w}_{t-1}) + \sum_{i=t}^{t+k-1} (\gamma\lambda)^{i-t}\delta'_i, \quad \text{where} \\ \delta'_t &\doteq R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_{t-1}) \end{aligned}$$

    This family of algorithms is known as Truncated TD(λ), or TTD(λ) . The update algorithm is

$$\mathbf{w}_{t+n} \doteq \mathbf{w_{t+n-1}} + \alpha\big[G^\lambda_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1})\big]\nabla\hat{v}(S_t, \mathbf{w}_{t+n-1})$$

    Just like the n-step methods in the tabular case, no updates are made in the first $n-1$ time steps of each episode, and $n-1$ additional updates are made upon termination.

- Redoing Updates: Online λ-return Algorithm:
  - The idea here is that at every new increment of data, we go back and redo all the updates since the beginning of the current episode, where we are shifting the weight vector towards an n-step truncated λ-return target. This is definitely more computationally complex, but it is online.

- Let us use $\mathbf{w}_t^h$ denote the weights used to generate the value of time t in the sequence up to horizon h, where $\mathbf{w}_0^h$ is inherited from the previous episode, and $\mathbf{w}_h^h$ is the ultimate weight vector of the algorithm:

$$h = 1: \quad \mathbf{w}_1^1 \doteq \mathbf{w}_0^1 + \alpha[G_{0:1}^\lambda - \hat{v}(S_0, \mathbf{w}_0^1]\nabla(S_0, \mathbf{w}_0^1)$$

$$\begin{aligned} h = 2: \quad \mathbf{w}_1^2 &\doteq \mathbf{w}_0^2 + \alpha[G_{0:2}^\lambda - \hat{v}(S_0, \mathbf{w}_0^2]\nabla(S_0, \mathbf{w}_0^2) \\ \mathbf{w}_2^2 &\doteq \mathbf{w}_1^2 + \alpha[G_{1:2}^\lambda - \hat{v}(S_1, \mathbf{w}_1^2]\nabla(S_1, \mathbf{w}_1^2) \end{aligned}$$

  The general form for the update is

$$\mathbf{w}_{t+1}^h \doteq \mathbf{w}_t^h + \alpha[G_{t:h}^\lambda - \hat{v}(S_t, \mathbf{w}_t^h]\nabla(S_t, \mathbf{w}_t^h).$$

- This is strictly more complex than the off-line algorithm, but in return, the online algorithm can be expected to perform better than the off-line one, since the weight vector has had a larger number of informative updates to it.

- True Online TD(λ):
  - There is a computationally less expensive method to the online λ-return method, which is known as the *true online TD(λ)* algorithm. Here, we only concern ourselves with the weight vectors $\mathbf{w}_0^0, \mathbf{w}_1^1, , \mathbf{w}_2^2, \dots, \mathbf{w}_T^T$, where we can just label$\mathbf{w}_t^t \doteq \mathbf{w}_t$ from hereafter.
  - The true online TD(λ) algorithm is then

$$\begin{gathered} \mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t + \alpha(\mathbf{w}_t^\intercal\mathbf{x}_t - \mathbf{w}_{t-1}^\intercal\mathbf{x}_t)(\mathbf{z}_t - \mathbf{x}_t) \\ \mathbf{z}_t \doteq \gamma\lambda\mathbf{z}_{t-1} + (1 - \alpha\gamma\lambda\mathbf{z}_{t-1}^\intercal\mathbf{x}_t)\mathbf{x}_t. \end{gathered}$$

    The eligibility trace here is called the *dutch trace*, which is different from the trace used in conventional TD(λ), which is called the *accumulating trace*.
  - Earlier work often use a third kind of trace called the *replacing trace*, defined only for the tabular case or for binary feature vectors. It is given by

$$z_{i,t} \doteq \begin{cases} 1 & \text{if } x_{i,t} = 1 \\ \gamma\lambda z_{i,t-1} & \text{otherwise} \end{cases}$$

  - This algorithm has been proven to produce the same sequence of weight vectors as the online λ-return algorithm. In addition, the memory algorithm here is identical to those of conventional TD(λ), while the per-step computation is increased by 50%, but still remains at $\mathcal{O}(d)$.

- Pseudocode is given in page 300.

- Dutch Traces in Monte Carlo Learning:
  - Eligibility traces are bit exclusive to TD learning, and it appears in MC methods as well. Using an eligibility trace in MC methods shifts the computation to an incremental algorithm whose time and memory complexity per step is $\mathcal{O}(d)$.
  - The need for eligibility traces arises whenever one tries to learn long-term predictions in an efficient manner.

- Sarsa(λ):
  - Sarsa(λ) is identical to TD(λ), except that the state values are replaced with action values. The relevant equations are given below:

$$\begin{gathered}\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t \\ \delta_t \doteq R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \\ \mathbf{z}_{-1} \doteq 0, \quad \mathbf{z}_t \doteq \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{q}(S_t, A_t, \mathbf{w})\end{gathered}$$

  - Compared to n-step Sarsa, eligibility traces can substantially increase the efficiency of control algorithms. Pseudocode can be found in page 305.
  - The pseudocode for the *true online Sarsa(λ)* is given in page 307.
  - The truncated version of Sarsa(λ), called *forward Sarsa(λ)*, is a particularly effective model-free control method for use in conjunction with multi-layer artificial neural network.

- Variable λ and γ:
  - It is useful to generalise the degree of bootstrapping and discounting beyond constant parameters to functions dependent on state and action. This means that $\lambda_t \doteq \lambda(S_t, A_t)$ and $\gamma_t \doteq \gamma(S_t, A_t)$.
  - The function γ can now be called the *termination function*. The return can be defined generally as

$$\begin{aligned}G_t &\doteq R_{t+1} + \gamma_{t+1}G_{t+1} \\ &= R_{t+1} + \gamma_{t+1}R_{t+2} + \gamma_{t+1}\gamma_{t+2}R_{t+3} + \dots \\ &= \sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k}\gamma_i\right)R_{k+1},\end{aligned}$$

  where we require that $\prod_{k=t}^{\infty}\gamma_k = 0$ to ensure that the sum is finite.

- A convenient aspect of this definition is that it enables the episodic setting to be presented in terms of a single stream of experience, without special terminal states, start distributions, or termination times. A terminal state simply becomes a state at which $\gamma(s) = 0$ and which transitions to the start distribution.
- The new state-based and action-based λ-return can be written as

$$G_t^{\lambda s} \doteq R_{t=1} + \gamma_{t+1}\big((1-\gamma_{t+1})\hat{v}(S_{t+1}, \mathbf{w}_t) + \gamma_{t+1}G_{t+1}^{\lambda s}\big)$$
$$G_t^{\lambda a} \doteq R_{t=1} + \gamma_{t+1}\big((1-\gamma_{t+1})\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) + \gamma_{t+1}G_{t+1}^{\lambda a}\big)$$

  where λ_s and λ_a refers to the returns that bootstraps from the state and action values respectively.

- Off-policy Traces with Control Variates:
  - The bootstrapping generalisation of per-decision sampling with control variates is given by (for the state case):

$$G_t^{\lambda s} \doteq \rho_t\Big(R_{t+1} + \gamma_{t+1}\big((1-\lambda_{t+1})\hat{v}(S_{t+1}, \mathbf{w}_t) + \lambda_{t+1}G_{t+1}^{\lambda s}\big)\Big)$$
$$+ (1-\rho_t)\hat{v}(S_t, \mathbf{w}_t), \quad \text{where } \rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}.$$

    We can approximate this in terms of sums of the state-based TD error:

$$G_t^{\lambda s} \approx \hat{v}(S_t, \mathbf{w}_t) + \rho_t \sum_{k=t}^{\infty} \delta_k^s \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i$$
$$\delta_t^s \doteq R_{t+1} + \gamma_{t+1}\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t)$$

  - The above form is convenient to be used in a forward-view update, and the product looks like an eligibility trace and it is multiplied by TD errors. In fact, following some algebra (page 310), we can show that the accumulating trace can be written as

$$\mathbf{z}_t \doteq \rho_t(\gamma_t\lambda_t\mathbf{z}_{t-1} + \nabla\hat{v}(S_t, \mathbf{w}_t)).$$

    This is in fact the general form of a TD(λ) algorithm which can be applied to on-policy or off-policy data.

  - The equations for the action-value methods can be found in page 310 and 311. The eligibility trace here can be written as

$$\mathbf{z}_t \doteq \rho_t\gamma_t\lambda_t\mathbf{z}_{t-1} + \nabla\hat{q}(S_t, A_t, \mathbf{w}_t).$$

This forms an elegant Expected Sarsa(λ) algorithm that can be used for on-policy or off-policy data. It is probably the best algorithm of this type at the time of writing of the book.

- Watkin's Q(λ) to Tree-Backup(λ):
    - There has been many methods proposed to extend Q-learning to eligibility traces.
    - Watkin's Q(λ) decays its eligibility traces the usual way as long as a greedy action was taken, then cuts the traces to zero after the first non-greedy action.
    - The eligibility trace version of Tree Backup is called *Tree-Backup(λ)*, or *TB(λ)*, where it is the true successor the Q-learning because it retains its appealing absence of importance sampling even though it can be applied to off-policy data.
    - The λ-return of TB(λ) is given by

$$\begin{aligned} G_t^{\lambda a} &\doteq R_{t+1} + \gamma_{t+1}\Bigg((1-\lambda_{t+1})\bar{V}(S_{t+1}) + \lambda_{t+1}\Big[\sum_{a\neq A_{t+1}} \pi(a|S_{t+1})\hat{q}(S_{t+1}, a, \mathbf{w}_t) \\ &\qquad + \pi(A_{t+1}|S_{t+1})G_{t+1}^{\lambda a}\Big]\Bigg) \\ &= R_{t+1} + \gamma_{t+1}\Big(\bar{V}(S_{t+1}) + \lambda_{t+1}\pi(A_{t+1}|S_{t+1})\Big(G_{t+1}^{\lambda a} - \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)\Big)\Big) \end{aligned}$$

$$G_t^{\lambda a} \approx \hat{q}(S_t, A_t, \mathbf{w}_t) + \sum_{k=t}^{\infty} \delta_k^a \prod_{i=t+1}^{k} \gamma_i \lambda_i \pi(A_t|S_t).$$

The eligibility trace here is

$$\mathbf{z}_t \doteq \gamma_t \lambda_t \pi(A_t|S_t)\mathbf{z}_{t+1} + \nabla\hat{q}(S_t, A_t, \mathbf{w}_t)$$

    - TB(λ) is not guaranteed to be stable when used with off-policy data and with a powerful function approximator. The following section will detail methods that can provide stability.
- Stable Off-policy Methods with Traces:
    - Here are four of the most important methods that guarantee stability under off-policy training.
    - *GTD(λ)* is based off the Gradient-TD method, where its goal is to learn a parameter $\mathbf{w}_t$ such that $\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}_t^\top \mathbf{x}(s) \approx v_\pi(s)$. Its update is

$$\begin{aligned}
\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha\delta_t^s \mathbf{z}_t - \alpha\gamma_{t+1}(1-\lambda_{t+1})(\mathbf{z}_t^\top \mathbf{v}_t)\mathbf{x}_{t+1} \\
\delta_t^s &= R_{t+1} + \gamma_{t+1}\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \\
\mathbf{z}_t &= \rho_t(\gamma_t\lambda_t\mathbf{z}_{t-1} + \nabla\hat{v}(S_t, \mathbf{w}_t)) \\
\mathbf{v}_{t+1} &= \mathbf{v}_t + \beta\delta_t^s\mathbf{z}_t - \beta(\mathbf{v}_t^\top\mathbf{x}_t)\mathbf{x}_t
\end{aligned}$$

- *GQ(λ)* is the Gradient-TD method for action values with eligibility traces. Its goal is to learn a parameter $\mathbf{w}_t$ such that $\hat{q}(s, a, \mathbf{w}) \doteq \mathbf{w}_t^\top \mathbf{x}(s, a) \approx q_\pi(s, a)$ from off-policy data. Its update is

$$\begin{aligned}
\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha\delta_t^a \mathbf{z}_t - \alpha\gamma_{t+1}(1-\lambda_{t+1})(\mathbf{z}_t^\top \mathbf{v}_t)\bar{\mathbf{x}}_{t+1} \\
\delta_t^a &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top\bar{\mathbf{x}}_{t+1} - \mathbf{w}_t^\top\mathbf{x}_t \\
\mathbf{z}_t &= \rho_t\gamma_t\lambda_t\mathbf{z}_{t-1} + \nabla\hat{q}(S_t, A_t, \mathbf{w}_t) \\
\mathbf{v}_{t+1} &= \mathbf{v}_t + \beta\delta_t^s\mathbf{z}_t - \beta(\mathbf{v}_t^\top\mathbf{x}_t)\mathbf{x}_t
\end{aligned}$$

- *HTD(λ)* is a hybrid state-value algorithm combining aspects of GTD(λ) and TD(λ). It is a strict generalisation of TD(λ) to off-policy learning, which means that if the behaviour policy happens to be the same as the target policy, then HTD(λ) becomes the same as TD(λ). The updates are given in page 315.
- *Emphatic TD(λ)* is given in page 315 and 316. It look similar to conventional TD(λ), however it also includes an emphasis and interest term.

Exercises

12.1 The recursive relation for the λ-return is

$$\begin{aligned}
G_t^\lambda &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n} \\
&= (1-\lambda)\big[G_{t:t+1} + \lambda G_{t:t+2} + \lambda^2 G_{t:t+3} + \dots\big] \\
&= (1-\lambda)\big[(R_{t+1} + \gamma G_{t+1:t+1}) + \lambda(R_{t+1} + \gamma G_{t+1:t+2}) \\
&\quad + \lambda^2(R_{t+1} + \gamma G_{t+1:t+3}) + \dots\big] \\
&= (1-\lambda)\big[R_{t+1}(1 + \lambda + \lambda^2 + \dots) + \gamma(G_{t+1:t+1} + \lambda G_{t+1:t+2} \\
&\quad + \lambda^2 G_{t+1:t+3} + \dots)\big] \\
&= (1-\lambda)\left[R_{t+1}\frac{1}{1-\lambda} + \gamma\Big(G_{t+1:t+1} + \lambda\sum_{n=1}^{\infty}\lambda^{n-1}G_{t+1:t+n}\Big)\right] \\
&= R_{t+1} + \gamma(1-\lambda)G_{t+1:t+1} + \gamma\lambda G_{t+1}^\lambda \\
&= R_{t+1} + \gamma(1-\lambda)\hat{v}(S_{t+1}, \mathbf{w}_t) + \gamma\lambda G_{t+1}^\lambda
\end{aligned}$$

12.2 The equation is given by

$$
\begin{aligned}
\lambda^{\tau_\lambda} &= \frac{1}{2} \\
\tau_\lambda &= \log_\lambda \frac{1}{2} > 0, \text{ since } \lambda < 1
\end{aligned}
$$

12.3 The off-line λ-return error is given by

$$
\begin{aligned}
G_t^\lambda - \hat{v}(S_t, \mathbf{w}_t) &= R_{t+1} + \gamma(1-\lambda)\hat{v}(S_{t+1}, \mathbf{w}_t) + \gamma\lambda G_{t+1}^\lambda - \hat{v}(S_t, \mathbf{w}_t) \\
&= \big(R_{+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t)\big) + \gamma\lambda\big(G_{t+1}^\lambda - \hat{v}(S_{t+1}, \mathbf{w}_t)\big) \\
&= \delta_t + \gamma\lambda\big(G_{t+1}^\lambda - \hat{v}(S_{t+1}, \mathbf{w}_t)\big) \\
&= \delta_t + \gamma\lambda\Big(\delta_{t+1} + \gamma\lambda\big(G_{t+2}^\lambda - \hat{v}(S_{t+2}, \mathbf{w})\big)\Big) \\
&= \sum_{k=t}^{\infty} (\gamma\lambda)^{k-t}\delta_k
\end{aligned}
$$

12.4 The sum of TD(λ) updates is

$$
\begin{aligned}
\sum_{t=0}^{\infty} \alpha\delta_t \mathbf{z}_t &= \sum_{t=0}^{\infty} \alpha\delta_t \big[\gamma\lambda \mathbf{z}_{t-1} + \nabla\hat{v}(S_t, \mathbf{w})\big] \\
&= \sum_{t=0}^{\infty} \alpha\delta_t \big[\gamma\lambda(\gamma\lambda \mathbf{z}_{t-2} + \nabla\hat{v}(S_{t-1}, \mathbf{w})) + \nabla\hat{v}(S_t, \mathbf{w})\big] \\
&= \sum_{t=0}^{\infty} \alpha\delta_t \left[\sum_{k=0}^{t} (\gamma\lambda)^{t-k} \nabla\hat{v}(S_k, \mathbf{w})\right] \\
&= \sum_{k=0}^{\infty}\sum_{t=k}^{\infty} \alpha\delta_t (\gamma\lambda)^{t-k} \nabla\hat{v}(S_k, \mathbf{w}) \\
&= \sum_{k=0}^{\infty}\sum_{t'=0}^{\infty} \alpha\delta_{t'+k} (\gamma\lambda)^{t'} \nabla\hat{v}(S_k, \mathbf{w})
\end{aligned}
$$

The sum of off-line λ-return updates is

$$
\begin{aligned}
\sum_{t=0}^{\infty} \alpha[G_t^\lambda - \hat{v}(S_t, \mathbf{w})]\nabla\hat{v}(S_t, \mathbf{w}) &= \sum_{t=0}^{\infty} \alpha \sum_{k=t}^{\infty} \big[(\gamma\lambda)^{k-t}\delta_k\big] \nabla\hat{v}(S_t, \mathbf{w}) \\
&= \sum_{t=0}^{\infty}\sum_{k'=0}^{\infty} \alpha\delta_{k'+t} (\gamma\lambda)^{k'} \nabla\hat{v}(S_t, \mathbf{w}) \\
&= \sum_{k=0}^{\infty}\sum_{t'=0}^{\infty} \alpha\delta_{t'+k} (\gamma\lambda)^{t'} \nabla\hat{v}(S_k, \mathbf{w})
\end{aligned}
$$

Thus, the two sums are the same.

12.5

$$\begin{aligned}
G^{\lambda}_{t:t+k} &= (1-\lambda)\sum_{n=1}^{k-1}\lambda^{n-1}G_{t:t+n} + \lambda^{k-1}G_{t:t+k} \\
&= \sum_{n=1}^{k-1}\lambda^{n-1}G_{t:t+n} - \sum_{n=1}^{k-1}\lambda^{n}G_{t:t+n} + \lambda^{k-1}G_{t:t+k} \\
&= \sum_{n=0}^{k-1}\lambda^{n}G_{t:t+n+1} - \sum_{n=1}^{k-1}\lambda^{n}G_{t:t+n} \\
&= G_{t:t+1} + \sum_{n=1}^{k-1}\lambda^{n}G_{t:t+n+1} - \sum_{n=1}^{k-1}\lambda^{n}G_{t:t+n} \\
&= R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}_t) + \sum_{n=1}^{k-1}\lambda^{n}[G_{t:t+n+1} - G_{t:t+n}] \\
&= R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}_t) + \sum_{n=1}^{k-1}(\gamma\lambda)^{n}[R_{t+n+1} + \gamma\hat{v}(S_{t+n+1},\mathbf{w}_{t+n}) - \hat{v}(S_{t+n},\mathbf{w}_{t+n-1}] \\
&= \delta'_t + \hat{v}(S_t,\mathbf{w}_{t-1}) + \sum_{n=t+1}^{t+k-1}(\gamma\lambda)^{n-t}\delta'_n \\
&= \hat{v}(S_t,\mathbf{w}_{t-1}) + \sum_{n=t}^{t+k-1}(\gamma\lambda)^{n-t}\delta'_n
\end{aligned}$$

12.6 Dutch trace is given by $\mathbf{z}_t \doteq \gamma\lambda\mathbf{z}_{t-1} + (1-\alpha\gamma\lambda\mathbf{z}_{t-1}^{\intercal}\mathbf{x}_t)\mathbf{x}_t$. To modify the pseudocode, we need to replace the $\text{Loop for } i \text{ in } \mathcal{F}(S,A)$ with

```
Loop for $i$ in $\mathcal{F}(S,A)$ :
    $d \leftarrow d + z_i$
Loop for $i$ in $\mathcal{F}(S,A)$ :
    $\delta \leftarrow \delta - w_i$
    $z_i \leftarrow \gamma\lambda z_i + 1 - \alpha\gamma\lambda d$
```

12.7

$$
\begin{aligned}
G^{\lambda_s}_{t:h} &\doteq R_{t=1} + \gamma_{t+1}\big((1-\gamma_{t+1})\hat{v}(S_{t+1},\mathbf{w}_t) + \gamma_{t+1} G^{\lambda_s}_{t+1:h}\big) \\
G^{\lambda_a}_{t:h} &\doteq R_{t=1} + \gamma_{t+1}\big((1-\gamma_{t+1})\hat{q}(S_{t+1},A_{t+1},\mathbf{w}_t) + \gamma_{t+1} G^{\lambda_a}_{t+1:h}\big) \\
G^{\lambda_a}_{t:h} &\doteq R_{t=1} + \gamma_{t+1}\big((1-\gamma_{t+1})\bar{V}(S_{t+1}) + \gamma_{t+1} G^{\lambda_a}_{t+1:h}\big)
\end{aligned}
$$

12.8

$$
\begin{aligned}
G^{\lambda s}_t &\doteq \rho_t\Big(R_{t+1} + \gamma_{t+1}\big((1-\lambda_{t+1})V_{t+1} + \lambda_{t+1}G^{\lambda s}_{t+1}\big)\Big) + (1-\rho_t)V_t \\
&= \rho_t\Big(R_{t+1} + \gamma^{t+1}V_{t+1} - V_t + \gamma_{t+1}\lambda_{t+1}\big(G^{\lambda s}_{t+1} - V_{t+1}\big)\Big) + V_t \\
&= \rho_t\Big(\delta^s_t + \gamma_{t+1}\lambda_{t+1}\big(G^{\lambda s}_{t+1} - V_{t+1}\big)\Big) + V_t \\
&= \rho_t\Bigg(\delta^s_t + \gamma_{t+1}\lambda_{t+1}\Big(\rho_{t+1}\big(\delta^s_{t+1} + \gamma_{t+2}\lambda_{t+2}(G^{\lambda s}_{t+2} - V_{t+2})\big) + \cancel{V_{t+1}} - \cancel{V_{t+1}}\Big)\Bigg) + V_t \\
&\dots \\
&= \rho_t\big(\delta^s_t + \rho_{t+1}\gamma_{t+1}\lambda_{t+1}\delta^s_{t+1} + \rho_{t+1}\rho_{t+2}\gamma_{t+1}\gamma_{t+2}\lambda_{t+1}\lambda_{t+2}\delta^s_{t+2} + \dots\big) \\
&= V_t + \sum_{k=t}^{\infty}\delta^s_k \prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i
\end{aligned}
$$

12.9 Let's guess the answer:

$$
G^{\lambda s}_{t:h} = \hat{v}(S_t,\mathbf{w}_t) + \rho_t\sum_{k=t}^{h}\delta^s_k\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i
$$

12.10

$$
\begin{aligned}
G^{\lambda a}_t &\doteq R_{t+1} + \gamma_{t+1}\Big(\bar{V}_t(S_{t+1}) + \lambda_{t+1}\rho_{t+1}\big[G^{\lambda a}_{t+1} - Q_{t+1}\big]\Big) \\
&= R_{t+1} + \gamma_{t+1}\bar{V}_t(S_{t+1}) - Q_t + Q_t + \gamma_{t+1}\lambda_{t+1}\rho_{t+1}\Big(G^{\lambda a}_{t+1} - Q_{t+1}\Big) \\
&= Q_t + \delta^a_t + \gamma_{t+1}\lambda_{t+1}\rho_{t+1}\Big(G^{\lambda a}_{t+1} - Q_{t+1}\Big) \\
&= Q_t + \delta^a_t + \gamma_{t+1}\lambda_{t+1}\rho_{t+1}\Big(\cancel{Q_{t+1}} + \delta^a_{t+1} + \gamma_{t+2}\lambda_{t+2}\rho_{t+2}\big(G^{\lambda a}_{t+2} - Q_{t+2}\big) - \cancel{Q_{t+1}}\Big) \\
&\dots \\
&= Q_t + \sum_{k=t}^{\infty}\delta^a_k\prod_{i=t+1}\gamma_i\lambda_i\rho_i
\end{aligned}
$$

12.11

$$G_{t:h}^{\lambda a} = \hat{q}(S_t, A_t, \mathbf{w}_t) + \sum_{k=t}^{h} \delta_k^a \prod_{i=t+1} \gamma_i \lambda_i \rho_i$$

12.12

Update Equation:

$$\begin{aligned}
\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha\big[G_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}_t)\big]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t) \\
&\approx \mathbf{w}_t + \alpha\left[\sum_{k=t}^{\infty} \delta_k^a \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i\right]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)
\end{aligned}$$

Update Sum:

$$\begin{aligned}
\sum_{t=1}^{\infty}(\mathbf{w}_{t+1} - \mathbf{w}_t) &\approx \sum_{t=1}^{\infty}\sum_{k=t}^{\infty} \alpha\delta_k^a \nabla\hat{q}(S_t, A_t, \mathbf{w}_t) \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i\Big] \\
&= \sum_{k=1}^{\infty} \alpha\delta_k^a \sum_{t=1}^{k} \nabla\hat{q}(S_t, A_t, \mathbf{w}_t) \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i\Big]
\end{aligned}$$

Trace at k:

$$\begin{aligned}
\mathbf{z}_k &= \sum_{t=1}^{k} \nabla\hat{q}(S_t, A_t, \mathbf{w}_t) \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i \\
&= \gamma_k \lambda_k \rho_k \sum_{t=1}^{k-1} \nabla\hat{q}(S_t, A_t, \mathbf{w}_t) \prod_{i=t+1}^{k-1} \gamma_i \lambda_i \rho_i \;+\; \nabla\hat{q}(S_k, A_k, \mathbf{w}_k) \\
&= \gamma_k \lambda_k \rho_k \mathbf{z}_{k-1} \;+\; \nabla\hat{q}(S_k, A_k, \mathbf{w}_k)
\end{aligned}$$

12.13 ?