Advantage(λ) learning

Bram Bakker
IDSIA
Galleria 2, 6928 Manno-Lugano, Switzerland
bram@idsia.ch

August 19, 2002

Abstract

Advantage(λ) learning is a novel method combining Advantage learning with eligibility traces. The main result in this report is a proof of the equivalence of the forward and backward views of Advantage(λ) learning with function approximators. That is, offline Advantage learning with the same eligibility traces as Q-learning, and on the basis of the one-step return (backward view), leads to the same weight updates as offline Advantage learning without eligibility traces on the basis of the λ-return (forward view).

1 Overview

This technical report describes Advantage(λ) learning, i.e., it describes how Advantage learning (Harmon & Baird, 1996) can be included into the TD(λ) family of learning algorithms (Sutton, 1988; Sutton & Barto, 1998). Section 2 first describes Advantage learning. The reasons why we would want to extend Advantage learning to Advantage(λ) learning are basically the same as with the other members of TD(λ) family, and they are summarized in section 3. Section 3 also describes the Advantage(λ) learning algorithm, and how to implement it using eligibility traces. Section 4 contains the crucial proof showing that Advantage learning with eligibility traces as described in section 3 accomplishes the desired result, i.e., updates based on the λ-return. Section 5 contains a summary of the results.

2 Advantage learning

2.1 The basic idea behind Advantage learning

Advantage learning (Harmon & Baird, 1996; Baird, 1999) is a reinforcement learning (RL) algorithm derived from Advantage updating (Baird, 1994), which in turn was designed as an improvement on Q-learning for continuous-time RL. In continuous-time RL, values of adjacent states typically differ by only small amounts, relative to the possible overall variance of values. This means that the optimal Q-values of different actions in a given state, from which the policy is directly derived, differ by only small amounts. These differences, then,

can easily get lost in the noise, especially when we use function approximators, such as neural networks.

Advantage learning remedies this problem by artificially decreasing the values of suboptimal actions in each state. The differences between the values of different actions in each state are thus greater than in Q-learning, and less likely to get lost in the noise. Doya (2000) shows that Advantage updating (so presumably Advantage learning as well) can be understood as one approach to deriving an efficient control policy for continuous-time systems using the estimated *gradient* of the Q-value function.

Even though Advantage learning was originally designed for continuous-time RL, it can be and has been used for discrete-time RL as well (Bakker, 2002; Bakker, Linåker, & Schmidhuber, in press). Note that the same problem of small differences between the values of adjacent states applies to any RL problem with long paths to rewards.

2.2 Bellman equation and learning algorithm

In Advantage learning, the value of an environmental state is defined as

$$V^*(s) = \max_a A^*(s,a). \tag{1}$$

The optimal Advantage $A^*(s,a)$ for each action a in state s is defined as

$$A^*(s,a) = V^*(s) + \frac{\langle r + \gamma V^*(s') \rangle - V^*(s)}{\kappa} \tag{2}$$

where $\langle \cdot \rangle$ represents the expected value over all possible results of executing action a in state s, which leads to immediate reward r and a new state s'. γ is a discount factor in the range $[0,1]$. For an optimal action, the second term is zero, meaning that the value of this action is the value of the state. For suboptimal actions, the second term is negative. The size of the second term then depends on κ, a constant scaling the difference between values of optimal and suboptimal actions (κ replaces $K\Delta t$ of the original formulation). Equation 2 is Advantage learning's equivalent of the Bellman equation.

To implement Advantage learning, the righthand side of equation 2 can be viewed as the target output for the Advantage of the executed action at the current timestep. As in Q-learning, actual experiences in the environment replace $\langle \ldots \rangle$, and the system's own current approximation to $V^*(s)$ and $V^*(s')$ is used. Thus, the update is based on the immediate reward received after executing action a, and the maximally attainable Advantage value in the next state, estimated by the system itself. This yields the *temporal difference error* E_t^{TD} at time t, which basically computes the difference between the estimated righthand side and lefthand side of equation 2:

$$E_t^{TD} = V(s_t) + \frac{r_{t+1} + \gamma V(s_{t+1}) - V(s_t)}{\kappa} - A(s_t, a_t). \tag{3}$$

Consider the case where Advantage learning is implemented using a function approximator, as in practice is usually the case with Advantage learning. The tabular representation can be viewed as a special case of function approximation. The function approximator has

parameters or weights w_{im}. At each timestep t, the weight update prescribed by direct Advantage learning[1] is

$$\Delta w_{im} = \alpha E_t^{TD} \frac{\partial A(s_t, a_t)}{\partial w_{im}} \tag{4}$$

for all parameters w_{im}. When we use a tabular representation, each parameter w_{im} simply corresponds to one table entry $A(s,a)$ for each state-action pair, and $\frac{\partial A(s_t,a_t)}{\partial w_{im}}$ is 1 for the state and action of time t, and 0 otherwise.

3 Advantage(λ) learning

3.1 The λ-return

In its simplest form, Advantage learning, like other temporal difference learning algorithms, performs back-ups based on the *immediate reward plus the discounted value of the next state.* This is called the 1-step return, or $R_t^{(1)}$. Advantage learning's temporal difference error can be rewritten as

$$\begin{aligned} E_t^{TD} &= V(s_t) + \frac{r_{t+1} + \gamma V(s_{t+1}) - V(s_t)}{\kappa} - A(s_t, a_t) \\ &= V(s_t) + \frac{R_t^{(1)} - V(s_t)}{\kappa} - A(s_t, a_t). \end{aligned} \tag{5}$$

Now consider a task where rewards are sparse. In the beginning of learning, only state-action pairs just preceding the rewards will have significant updates. And the value "contained" in the rewards will only disperse to other state-action pairs very slowly, as the state-action pairs immediately preceding the rewards take on significant value. It is natural to wonder whether we may speed up learning by updating state-action values not just on value information from one timestep later, but also from multiple timesteps later. In this way, state-action pairs a number of actions away from a sparse reward may start doing significant value updates right from the start. In other words, we want to base updates not only on the 1-step return, but also on multiple-step returns. For example, the update may be done toward the average of the 1-step return and the 2-step return $R_t^{(2)}$, where $R_t^{(2)}$ is defined as

$$R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 \max_a Q_t(s_{t+2}, a). \tag{6}$$

Or the update may be done toward the average of the 1-step, 2-step, and 3-step return, etc.

It may seem that the only way to do this updating toward multiple-step returns is to postpone updates until those multiple timesteps later, and in the meantime remember exactly which state-action pairs were visited in which order. However, there is a simple, elegant way to do updates toward multiple-step returns that is still local in time. Algorithms doing this

[1] Certain RL algorithms based on temporal difference learning, such as Q-learning and Advantage learning, can in principle lead to divergence when used in conjunction with function approximators such as neural networks (e.g. see Harmon & Baird, 1996; Baird, 1999; Sutton & Barto, 1998). It is unclear at this point how serious this problem is. If it turns out to be a problem in practical cases, one may use the safer "residual" or "residual gradient" versions of Advantage learning (Harmon & Baird, 1996; Baird, 1999), rather than the direct version this report is concerned with.

are members of the TD(λ) family of algorithms (Sutton, 1988). In the case of off-policy methods (Sutton & Barto, 1998) like Advantage(λ) learning, updates are now done based on the λ-return R_t^λ, defined in the following way (Watkins, 1989):

$$R_t^\lambda = (1 - \Lambda_{t+1})R_t^{(1)} + \Lambda_{t+1}(1 - \Lambda_{t+2})R_t^{(2)} + \Lambda_{t+1}\Lambda_{t+2}(1 - \Lambda_{t+3})R_t^{(3)} + \dots \tag{7}$$

where

$$\Lambda_t = \begin{cases} \lambda & a_t = \arg\max_a A(s_t, a) \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where $0 \leq \lambda \leq 1$ is a constant that weighs the importance of long-term rewards as opposed to short-term rewards. If $\lambda = 0$ (Advantage(0)-learning or plain Advantage learning), this reduces to standard Advantage learning's 1-step return. If $\lambda = 1$ (Advantage(1)-learning), on the other hand, the λ-return is based only on actual rewards obtained during the episode. Thus, in that case the λ-return is not based on estimated values; in other words, it does not *bootstrap*. In the intermediate cases, $0 < \lambda < 1$, an average of multiple-step returns is taken, weighted in a particular way.

The λ-return is truncated at the point where an exploring action is chosen, i.e. when $a_t \neq \arg\max_a A(s_t, a)$. This corresponds to the notion that rewards obtained after that point no longer reflect the value of the currently estimated best policy. After all, in off-policy methods like Q-learning and Advantage learning we wish to learn about the best policy, and not about a policy with exploring, probably suboptimal actions.

3.2 Eligibility traces

Now, the beautiful thing is that the update of a value based on the λ-return can be accomplished by maintaining a simple scalar measure, called the *eligibility trace*, for each parameter of the reinforcement learning agent (Sutton, 1988, 1989; Watkins, 1989). Thus, there is one eligibility trace e_{im} per parameter w_{im}. The eligibility trace $e_{im,t}$ at time t for Advantage learning is basically the same as that for Q-learning, and it is defined as

$$e_{im,t} = \gamma \Lambda_t e_{im,t-1} + \frac{\partial A(s_t, a_t)}{\partial w_{im}}. \tag{9}$$

The eligibility trace can be viewed as information that says to what extent the corresponding parameter w_{im} can be held "responsible" for rewards obtained later on. If an exploring action is taken, $\Lambda_t = 0$, so the eligibility traces are reset to 0. $e_{im,0}$ is 0 for all parameters.

The update rule for Advantage(λ) learning, implemented using eligibility traces, then becomes

$$\Delta w_{im,t}^{TD} = \alpha E_t^{TD} e_{im,t} \tag{10}$$

for all parameters w_{im}.

A natural question to ask is why we would use bootstrapping methods, i.e. $\lambda < 1$, at all. After all, non-bootstrapping methods, $\lambda = 1$, can be implemented just as easily with eligibility traces, and then the λ-returns are based only on actually obtained rewards rather than on other estimated values, which may be unreliable. The simple answer is that in practice, it turns out that bootstrapping methods often outperform non-bootstrapping methods (Sutton & Barto, 1998). Formal reasons for this are not known. An intuitive reason may be that to

the extent that estimated values go some way toward the correct values, they provide useful "subgoal" information, even when they themselves are still some distance away from actual rewards. Sequences of actions reaching such high-value subgoal states could then be learned more efficiently than when the actual rewards must have been reached. After all, between reaching the high-value subgoal state and the actual rewards one still needs to do a number of actions, each of which can be done incorrectly.

3.3 The forward and backward view

Advantage(λ) learning implemented using eligibility traces can be called the *backward view* (Sutton & Barto, 1998) of Advantage(λ) learning, because at each point in time it looks back to the agent's past, using the eligibility traces, to estimate which parameters used in the past were responsible for the current situation. Advantage(λ) learning can also be implemented by straightforwardly using the λ-return in the target output. This is called the *forward view* because at each point in time it looks forward in time for rewards and values in the future.

The backward or mechanistic view is important because it has an elegant and straightforward implementation. It allows for learning that is local in time, doing value updates as each new experience comes in. This makes it both desirable as an artificial intelligence tool and increases its relevance for understanding biological intelligence. Furthermore, the backward view can give us an intuitive understanding of how a parameter can be held responsible for outcomes later in time.

The forward or theoretical view corresponds to a learning algorithm that is not local in time: value updates at a particular time depend on rewards that are available only later. However, it gives a different, more formal perspective that shows what exactly learning with eligibility traces accomplishes. In other words, it shows what error the value function system actually attempts to minimize.

The main result of this technical report, presented in the next section, is a proof that shows the equivalence of the forward and backward views of Advantage(λ) learning with function approximators. More precisely, it shows that offline Advantage learning with the same eligibility traces as Q-learning, and on the basis of the one-step return (the backward view), leads to the same weight updates as offline Advantage learning without eligibility traces on the basis of the λ-return (the forward view). This is important because it is not obvious that Advantage learning can be combined with eligibility traces in the same way as Q-learning and lead to the same desired result, namely back-ups based on the λ-return. It is not obvious because the corresponding Bellman equations differ for the two algorithms, and the known results for Q(λ) and TD(λ) depend on the corresponding Bellman equations.

4 Equivalence of the forward and backward view of Advantage(λ) learning

4.1 Preliminaries

The following proof makes use of work by Watkins (1989), Sutton (1988), Sutton (1989), and Sutton and Barto (1998). It is based on the proof presented in Sutton (1988) and Sutton and Barto (1998) regarding the equivalence of the forward and backward view of TD(λ)

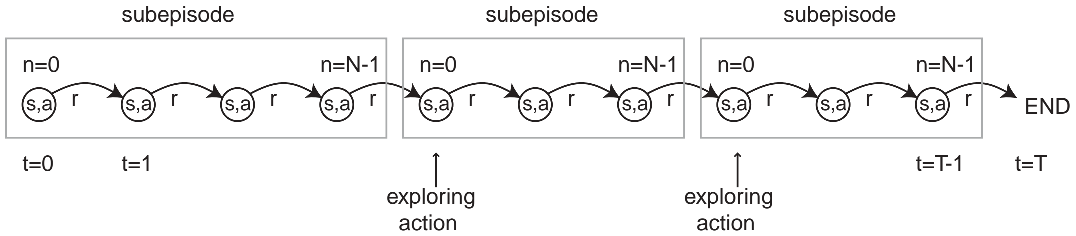

Figure 1: Schematic representation of how an episode is made up of subepisodes. The moments at which exploring actions are taken mark the beginning of a new subepisode.

prediction using a tabular representation. Compared to their proof, the proof presented here is not only adapted for Advantage learning's different Bellman equation, it is also slightly more general, because it allows for λ to be set to 0 occasionally during the episode (as is necessary during exploration in off-policy control methods like Q-learning and Advantage learning), and it allows for function approximators, of which the tabular representation is a special case.

4.2 Subepisodes

Let $\Delta w^{\lambda}_{im,t}$ denote the weight update at time t according to the λ-return algorithm (forward view), and let $\Delta w^{TD}_{im,t}$ denote the weight update according to the algorithm with eligibility traces (backward view). Let the episode start at time $t = 0$ and end at $t = T$. What we need to establish, then, is that the sum of all weight updates over an episode is always the same for both algorithms:

$$\sum_{t=0}^{T-1} \Delta w^{\lambda}_{im,t} = \sum_{t=0}^{T-1} \Delta w^{TD}_{im,t}. \tag{11}$$

The first step is to define a *subepisode* as a substring of the entire string of state-action pairs making up the episode, such that the first element in the substring is either the first state-action pair of the episode ($t = 0$) or a state-action pair in which the action is exploratory, and the last element is either the last state-action pair before the end ($t = T - 1$) or the last state-action pair before the next exploratory action. Let $n = 0$ denote the first element of the subepisode and $n = N - 1$ the last element. Figure 1 shows how an episode is composed of subepisodes. The entire episode is simply the concatenation of all subepisodes. Therefore, if we want to show equivalence of the weight updates over an entire episode, it is sufficient to show equivalence over each subepisode. Thus, we must establish that

$$\sum_{n=0}^{N-1} \Delta w^{\lambda}_{im,n} = \sum_{n=0}^{N-1} \Delta w^{TD}_{im,n}. \tag{12}$$

4.3 The forward view

Let us start with the lefthand side of eq. 12:

$$\sum_{n=0}^{N-1} \Delta w_{im,n}^{\lambda} = \sum_{n=0}^{N-1} \alpha E_n^{\lambda} \frac{\partial A(s_n, a_n)}{\partial w_{im}} \tag{13}$$

where

$$E_n^{\lambda} = V(s_n) + \frac{R_n^{\lambda} - V(s_n)}{\kappa} - A(s_n, a_n). \tag{14}$$

Let us look again at equation 7, which defines R_n^{λ}:

$$R_n^{\lambda} = (1 - \Lambda_{n+1}) R_n^{(1)} + \Lambda_{n+1}(1 - \Lambda_{n+2}) R_n^{(2)} + \Lambda_{n+1}\Lambda_{n+2}(1 - \Lambda_{n+3}) R_n^{(3)} + \dots \tag{15}$$

where $R_n^{(p)}$ is the p-step return, and equation 8, which defines

$$\Lambda_n = \begin{cases} \lambda & a_n = \arg\max_a A(s_n, a) \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

The λ-return is truncated at the point where an exploring action is chosen. From the definition of a subepisode it follows that $\Lambda_n = \lambda$ for all $0 < n < N$ and $\Lambda_N = 0$, so (15) can be rewritten as

$$R_n^{\lambda} = (1 - \lambda) R_n^{(1)} + (1 - \lambda)\lambda R_n^{(2)} + (1 - \lambda)\lambda^2 R_n^{(3)} + \dots + \lambda^{N-1-n} R_n^{(N)}. \tag{17}$$

This means that $R_n^{\lambda} - V(s_n)$ from eq. 14 can be written as

$$\begin{aligned} R_n^{\lambda} - V(s_n) = & - V(s_n) \\ & + (1-\lambda)\lambda^0 (r_{n+1} + \gamma V(s_{n+1})) \\ & + (1-\lambda)\lambda^1 (r_{n+1} + \gamma r_{n+2} + \gamma^2 V(s_{n+2})) \\ & + (1-\lambda)\lambda^2 (r_{n+1} + \gamma r_{n+2} + \gamma^2 r_{n+3} + \gamma^3 V(s_{n+3})) \\ & + \dots \\ & + \lambda^{N-1-n} (r_{n+1} + \gamma r_{n+2} + \gamma^2 r_{n+3} + \dots + \gamma^{N-1-n} r_N + \gamma^{N-n} V(s_N)). \end{aligned} \tag{18}$$

Now, R_n^{λ} was designed as a weighted sum of p-step returns. Thus, $(1-\lambda)\lambda^0 + (1-\lambda)\lambda^1 + \dots + \lambda^{N-1-n}$ sums to 1. This means that we can pull out the first column inside the brackets and get an unweighted term for that sum. We can do a similar thing for the second column,

etc. In all, we can rewrite eq. 18 as

$$
\begin{aligned}
R_n^\lambda - V(s_n) = & - V(s_n) \\
& + (\gamma\lambda)^0(r_{n+1} + \gamma V(s_{n+1}) - \gamma\lambda V(s_{n+1})) \\
& + (\gamma\lambda)^1(r_{n+2} + \gamma V(s_{n+2}) - \gamma\lambda V(s_{n+2})) \\
& + (\gamma\lambda)^2(r_{n+3} + \gamma V(s_{n+3}) - \gamma\lambda V(s_{n+3})) \\
& + \ldots \\
& + (\gamma\lambda)^{N-1-n}(r_N + \gamma V(s_N)) \\
= & \quad (\gamma\lambda)^0(r_{n+1} + \gamma V(s_{n+1}) - V(s_n)) \\
& + (\gamma\lambda)^1(r_{n+2} + \gamma V(s_{n+2}) - V(s_{n+1})) \\
& + (\gamma\lambda)^2(r_{n+3} + \gamma V(s_{n+3}) - V(s_{n+2})) \\
& + \ldots \\
& + (\gamma\lambda)^{N-1-n}(r_N + \gamma V(s_N) - V(s_{N-1})) \\
= & \sum_{k=n}^{N-1} (\gamma\lambda)^{k-n}(r_{k+1} + \gamma V(s_{k+1}) - V(s_k)).
\end{aligned} \tag{19}
$$

For convenience, let us define

$$\delta_k = r_{k+1} + \gamma V(s_{k+1}) - V(s_k). \tag{20}$$

Now we can rewrite eq. 14 as

$$E_n^\lambda = V(s_n) + \frac{\sum_{k=n}^{N-1} (\gamma\lambda)^{k-n}\delta_k}{\kappa} - A(s_n, a_n) \tag{21}$$

and eq. 13 as

$$\sum_{n=0}^{N-1} \Delta w_{im,n}^\lambda = \sum_{n=0}^{N-1} \alpha \frac{\partial A(s_n, a_n)}{\partial w_{im}} (V(s_n) + \frac{\sum_{k=n}^{N-1} (\gamma\lambda)^{k-n}\delta_k}{\kappa} - A(s_n, a_n)). \tag{22}$$

This is what the backward, mechanistic view of Advantage(λ) learning must realize using eligibility traces.

4.4 Backward view

Let us turn now to the righthand side of eq. 12, which corresponds to weight updates according to the backward view:

$$\sum_{n=0}^{N-1} \Delta w_{im,n}^{TD} = \sum_{n=0}^{N-1} \alpha E_n^{TD} e_{im,n} \tag{23}$$

where, as defined before in equation 3,

$$
\begin{aligned}
E_n^{TD} &= V(s_n) + \frac{r_{n+1} + \gamma V(s_{n+1}) - V(s_n)}{\kappa} - A(s_n, a_n) \\
&= V(s_n) + \frac{\delta_n}{\kappa} - A(s_n, a_n)
\end{aligned} \tag{24}
$$

and the eligibility traces for function approximators $e_{im,n}$ (eq. 9) are

$$e_{im,n} = \gamma\Lambda_n e_{im,n-1} + \frac{\partial A(s_n,a_n)}{\partial w_{im}}. \tag{25}$$

For reasons that will become apparent shortly, we decompose the sum in eq. 23 into two parts:

$$\sum_{n=0}^{N-1} \Delta w_{im,n}^{TD} = \sum_{n=0}^{N-1} \alpha\frac{\delta_n}{\kappa} e_{im,n} + \sum_{n=0}^{N-1} \alpha(V(s_n) - A(s_n,a_n))e_{im,n}. \tag{26}$$

Let's consider the second part of this sum. Note that $e_{im,0} = \frac{\partial A(s_0,a_0)}{\partial w_{im}}$. Furthermore, note that for $0 < n < N$, $A(s_n,a_n) = V(s_n)$, because these are all exploiting actions. Thus, eq. 26 can be rewritten as

$$\sum_{n=0}^{N-1} \Delta w_{im,n}^{TD} = \sum_{n=0}^{N-1} \alpha\frac{\delta_n}{\kappa} e_{im,n} + \sum_{n=0}^{N-1} \alpha(V(s_n) - A(s_n,a_n))\frac{\partial A(s_n,a_n)}{\partial w_{im}}. \tag{27}$$

We know that within a subepisode, $\Lambda_n = \lambda$ for all $0 < n < N$. The recursive definition of eq. 25 can then be rewritten in the following non-recursive form (Watkins, 1989; Sutton & Barto, 1998):

$$e_{im,n} = \sum_{k=0}^{n} (\gamma\lambda)^{n-k} \frac{\partial A(s_k,a_k)}{\partial w_{im}}. \tag{28}$$

Thus, eq. 27 becomes

$$\begin{aligned}
\sum_{n=0}^{N-1} \Delta w_{im,n}^{TD} &= \sum_{n=0}^{N-1} \alpha\frac{\delta_n}{\kappa} \sum_{k=0}^{n} (\gamma\lambda)^{n-k} \frac{\partial A(s_k,a_k)}{\partial w_{im}} + \sum_{n=0}^{N-1} \alpha(V(s_n) - A(s_n,a_n))\frac{\partial A(s_n,a_n)}{\partial w_{im}} \\
&= \sum_{k=0}^{N-1} \alpha \sum_{n=0}^{k} (\gamma\lambda)^{k-n} \frac{\partial A(s_n,a_n)}{\partial w_{im}} \frac{\delta_k}{\kappa} + \sum_{n=0}^{N-1} \alpha(V(s_n) - A(s_n,a_n))\frac{\partial A(s_n,a_n)}{\partial w_{im}} \\
&= \sum_{n=0}^{N-1} \alpha \sum_{k=n}^{N-1} (\gamma\lambda)^{k-n} \frac{\partial A(s_n,a_n)}{\partial w_{im}} \frac{\delta_k}{\kappa} + \sum_{n=0}^{N-1} \alpha(V(s_n) - A(s_n,a_n))\frac{\partial A(s_n,a_n)}{\partial w_{im}} \\
&= \sum_{n=0}^{N-1} \alpha \frac{\partial A(s_n,a_n)}{\partial w_{im}} \sum_{k=n}^{N-1} (\gamma\lambda)^{k-n} \frac{\delta_k}{\kappa} + \sum_{n=0}^{N-1} \alpha(V(s_n) - A(s_n,a_n))\frac{\partial A(s_n,a_n)}{\partial w_{im}} \\
&= \sum_{n=0}^{N-1} \alpha \frac{\partial A(s_n,a_n)}{\partial w_{im}} (V(s_n) + \frac{\sum_{k=n}^{N-1}(\gamma\lambda)^{k-n}\delta_k}{\kappa} - A(s_n,a_n))
\end{aligned} \tag{29}$$

which is equal to eq. 22. This proves the equivalence of the forward and backward view.

This result only holds for offline updating, i.e. updating after each episode (or, in our case, after each subepisode), otherwise $A(s_n,a_n)$, $V(s_n)$, and $\frac{\partial A(s_n,a_n)}{\partial w_{im}}$ are not necessarily exactly equal in both cases. However, since weight changes during an episode using online updating will in general be small, we can expect a close approximation in the online case.

5 Conclusion

This report shows that if we simply use the same type of eligibility traces with Advantage learning as we do with Q-learning, we obtain a similar end result as with Q-learning, namely gradient descent in parameter space based on the λ-return. In fact, just as Q-learning without eligibility traces is a special case of Advantage learning without eligibility traces, Q(λ)-learning is a special case of Advantage(λ) learning, when $\kappa = 1$.

References

Baird, L. C. (1994). Reinforcement learning in continuous time: Advantage updating. In *Proceedings of the international conference on neural networks.*

Baird, L. C. (1999). *Reinforcement learning through gradient descent.* PhD thesis, Carnegie Mellon University, Pittsburgh.

Bakker, B. (2002). Reinforcement learning with Long Short-Term Memory. In T. G. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in Neural Information Processing Systems 14.* Cambridge, MA: MIT Press.

Bakker, B., Linåker, F., & Schmidhuber, J. (in press). Reinforcement learning in partially observable mobile robot domains using unsupervised event extraction. In *Proceedings of the 2002 IEEE/RSJ international conference on intelligent robots and systems.*

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural Computation, 12 (1)*, 219–245.

Harmon, M. E., & Baird, L. C. (1996). *Multi-player residual advantage learning with general function approximation* (Technical report No. WL-TR-1065). Wright-Patterson Air Force Base Ohio: Wright Laboratory.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning, 3*, 9–44.

Sutton, R. S. (1989). *Implementation details of the TD(λ) procedure for the case of vector predictions and backpropagation* (Technical report No. TN87-509.1). GTE Laboratories.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* PhD thesis, Cambridge University.