Faster Temporal Credit Assignment in Learning Classifier Systems

Paweł Cichosz Jan J. Mulawka
Institute of Electronics Fundamentals
Warsaw University of Technology
Nowowiejska 15/19, 00-665 Warsaw, Poland
{cichosz, jml}@ipe.pw.edu.pl

Abstract

Classifier systems are genetics-based learning systems using the paradigm of reinforcement learning. In the most challenging case of delayed reinforcement, it involves a difficult temporal credit assignment problem. Standard classifier systems solve this problem using the bucket brigade algorithm. In this paper we show how to make the temporal credit assignment process faster by augmenting this algorithm by some refinements borrowed from a related field of reinforcement learning algorithms based on the methods of temporal differences (TD). These algorithms usually converge significantly faster if they are used in combination with TD($\lambda > 0$). As a natural consequence of the easily noticeable similarity between the bucket brigade and TD(0), the BB(λ) algorithm is derived, using the standard technique of eligibility traces. The TTD(λ, m) procedure, which eliminates eligibility traces and implements an approximation of TD(λ) in a computationally efficient way, has also been ported to the context of classifier systems, yielding the TBB(λ, m) algorithm. The two resulting novel algorithms provide promising and, strangely enough, completely unexplored so far possibilities of making learning classifier systems learn faster under the conditions of reinforcement delay.

1 Introduction

Classifier systems (CS, e.g., Booker, Goldberg, & Holland, 1989; Wilson, 1994) constitute the most popular approach to genetics-based machine learning. Their essential idea is to maintain and genetically improve a population of simple decision rules, called *classifiers*. Each classifier is composed of a condition part and an action part. Whenever a classifier's condition matches the input information received by the system, called the *input message*, its action is used to determine the system's response. Learning in such systems consists of two processes: evaluating the quality of each classifier, based on the observed consequences of applying it to generate the system's actions, and processing the population genetically, to hopefully increase the number of successful (i.e., highly evaluated) classifiers in subsequent generations. This paper focuses exclusively on the first of these two processes, traditionally referred to as *credit assignment*.

Classifier systems belong to the family of reinforcement learning (RL) methods. At each time step a reinforcement learning system observes the current input information from its environment (termed input message in the CS community and environment state elsewhere) and performs an action, selected according to its current decision policy. Then it receives a reinforcement value, also called a reward or a payoff, and a state transition takes place in the environment (a new input message arrives). The system's task is to learn a decision policy that leads to the maximization of the rewards it receives *in the long term*. A typical long-term performance measure is the expected discounted sum of reinforcement:

$$\mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{1}$$

where $\mathbf{E}$ is the expectation symbol, r_t is the reinforcement value received at time t, and $\gamma \in [0,1]$ is a *discount factor*, which adjusts the relative significance of long-term rewards versus short-term ones. To maximize this expression for any positive γ, the learning system must take into account the delayed consequences of its actions, by solving the *temporal credit assignment* problem. Learning with such performance measures is called learning from delayed rewards or delayed reinforcement learning.

Classifier systems solve the credit assignment problem using the *bucket brigade* algorithm (e.g., Booker et al., 1989; Wilson, 1994), which will be presented in Section 2.1. However, most of the current work on non-genetic approaches to reinforcement learning uses temporal credit assignment algorithms based on Sutton's *temporal difference* (TD) methods (Sutton, 1988), such as AHC (Sutton, 1984) or Q-learning (Watkins, 1989). TD is a class of prediction learning methods, parameterized by a *recency factor* $\lambda \in [0,1]$, which is written TD(λ). Its reinforcement learning version will be described in Section 2.2.

Some close relationships between TD(0)-based algorithms and the bucket brigade algorithm have been observed by several authors (e.g., Dorigo & Bersini, 1994; Wilson, 1994; Cichosz, 1994), but, strangely enough, these observations have not led to clear practical conclusions. To draw such conclusions, in this paper we port to the context of classifier systems the idea of TD(λ) for general λ. Using positive λ has been found to usually yield a considerable learning speedup for AHC and Q-learning (e.g., Lin, 1993; Cichosz, 1995; Cichosz & Mulawka, 1995), and thus may be expected to give similarly beneficial effects for the bucket brigade.

2 Bucket Brigade and TD-Based Algorithms

There are many possible implementations of classifier systems and several versions of the bucket brigade algorithm. Both simplifications and refinements have been proposed to the "canonical" classifier system model (Booker et al., 1989). For this work a generic classifier system based on ZCS (*zeroth order CS*) described by Wilson (1994) has been adopted, with a version of the bucket brigade algorithm called the *implicit bucket brigade*. This algorithm transfers credit among sequentially activated classifiers along temporal chains, while the "canonical" version uses causal chains of classifiers for this purpose. Discussing the exact nature of the simplifications assumed by ZCS-like systems in comparison to the original CS is beyond the scope of this paper. We strongly believe, however, that the novel algorithms we are going to present can be modified to fit other versions of classifier systems.

As far as TD-based reinforcement learning algorithms are concerned, our description will assume a common generic view previously used by Cichosz (1995). It can be easily instantiated to obtain particular algorithms, including AHC and Q-learning.

2.1 The Bucket Brigade Algorithm

The bucket brigade algorithm assigns credit for the system's outcomes to individual classifiers by modifying their real-valued *strengths*. These strengths may be subsequently used to determine their fitness for genetic processing. There is no need to assume any particular representation for classifier conditions and actions. The only important point is that there is a match predicate defined for input messages and classifier conditions: for any input message we can determine all the classifiers whose condition parts match the message, forming the *match set* for it. The match set is then used to select the system's current action (e.g., the action supported by the majority of its members, but many other selection mechanisms are possible).

The operation of a generic version of the (implicit) bucket brigade algorithm is presented in Figure 1. For a classifier c, s_c denotes its strength, c_c denotes its condition part, and a_c denotes its action part. We also write x_t, a_t, and r_t to designate the input message, the system's action, and reward at time t.

```
At each time step $t$:
    1. observe current input message $x_t$;
    2. $\mathbb{M}_t := match\_set(x_t)$;
    3. $a_t := select\_action(\mathbb{M}_t)$;
    4. $\mathbb{A}_t := action\_set(\mathbb{M}_t, a_t)$;
    5. perform action $a_t$;
    6. observe immediate reward $r_t$;
    7. $s := 0$;
    8. for each $c \in \mathbb{A}_t$ do
        (a) $s := s + s_c$;
        (b) $s_c := s_c + \beta(\frac{r_t}{|\mathbb{A}_t|} - s_c)$;
    9. for each $c \in \mathbb{A}_{t-1}$ do
        $s_c := s_c + \beta\frac{\gamma s}{|\mathbb{A}_{t-1}|}$.
```

Figure 1: The implicit bucket brigade algorithm.

In Step 2 the match set for x_t is formed, $\mathbb{M}_t$, which is then used to select an action to execute. After action selection, another set of classifiers is formed, $\mathbb{A}_t \subseteq \mathbb{M}_t$, called the *action set*. It contains all the classifiers from $\mathbb{M}_t$ which support the selected action a_t. The strengths of the members of this set are modified in Step 8, by subtracting from them a fraction (determined by a learning rate factor β) of their previous value and adding the same fraction of the current reward r_t, divided by the number of elements in $\mathbb{A}_t$. At the same time, the sum of their strengths (before these modifications) is computed and stored in an auxiliary variable s. A fraction (again, determined by β) of this sum, discounted by the discount factor γ, is subsequently divided among the strengths of all the classifiers constituting the previous action set, $\mathbb{A}_{t-1}$. It may be thus said that the total value subtracted from the strengths of the members of $\mathbb{A}_t$ is delivered to the strengths of their predecessors, the members of $\mathbb{A}_{t-1}$ (hence the term 'bucket brigade'). Because r_t and γs both contribute to the increase of classifier strengths, we will call the former a *primary reinforcement* and the latter a *secondary reinforcement*.

According to the presented algorithm, the strength of a classifier may be modified if it matches the current input message and supports the action which has actually been selected and executed in response to this message. It is increased (the classifier is positively reinforced) if, informally, a large reward has been received after executing the action, or the next input message activated good classifiers (with large strengths). Let $s_{t_1}(\mathbb{A}_{t_2})$ designate the total strength of the classifiers in set $\mathbb{A}_{t_2}$ at time t_1. The effects of the bucket brigade algorithm can be then more precisely written as follows:

$$s_{t+1}(\mathbb{A}_t) := s_t(\mathbb{A}_t) + \beta[r_t + \gamma s_{t+1}(\mathbb{A}_{t+1}) - s_t(\mathbb{A}_t)]. \tag{2}$$

This is of course a simplification which holds true only if $\mathbb{A}_t \cap \mathbb{A}_{t+1} = \emptyset$, but we find it very insightful and sufficiently exact.

2.2 TD-Based Algorithms

The generic TD-based reinforcement learning rule may be written as

$$update^{\beta}(U,\ x_t,\ r_t + \gamma U_t(x_{t+1}) - U_t(x_t)), \tag{3}$$

where U is the state utility function, assigning to each state x an estimate of the total discounted reinforcement received starting in state x and following the current policy. In particular, $U(x_t)$ is intended to predict the discounted sum of future rewards

$$z_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \tag{4}$$

called the *TD return* for time t. The update notation used in Equation 3 is supposed to mean that the value of function U for argument x_t should be adjusted using error value $\Delta = r_t + \gamma U_t(x_{t+1}) - U_t(x_t)$, i.e., drawn towards $U_t(x) + \Delta$, to a degree controlled by a learning rate parameter β. Cichosz (1995) shows how particular RL algorithms can be presented as instantiations of this generic formulation.

2.2.1 General λ: Eligibility Traces

Actually, Equation 3 corresponds to the simplest form of temporal difference methods, TD(0). For general values of λ, the TD(λ) update rule, applied to each state x at time step t, is given by

$$update^{\beta}(U,\ x,\ (r_t + \gamma U_t(x_{t+1}) - U_t(x_t))e_x(t)), \tag{5}$$

where

$$e_x(t) = \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \chi_x(k) \tag{6}$$

is the value of the *eligibility trace* (Sutton, 1984) for state x at time t, and $\chi_x(t)$ is equal 1 if $x = x_t$ and 0 otherwise. Eligibility traces for all states are also incrementally updated at each time step, according to the following trace update rule:

$$e_x(t) := \gamma\lambda e_x(t-1) + \chi_x(t), \tag{7}$$

where $e_x(0) = \chi_x(0)$ by definition.

2.2.2 General λ: Truncated Temporal Differences

For positive λ, one has to update at each time step predictions and eligibility traces for all states, while for $\lambda = 0$ Equation 5 reduces to Equation 3 and a prediction update is required for one state only. This is why with this implementation using $\lambda > 0$ is much more computationally expensive than using $\lambda = 0$. There is a simple technique proposed by Cichosz (1995), called the TTD procedure (*truncated temporal differences*), which makes it possible to use general λ at low computational costs.

The technique of truncated temporal differences is based on the following definition of truncated TD(λ) returns:

$$z_t^{\lambda,m} = \sum_{k=0}^{m-2} (\gamma\lambda)^k \Big[r_{t+k} + \gamma(1-\lambda)U_{t+k}(x_{t+k+1}) \Big] + (\gamma\lambda)^{m-1} \Big[r_{t+m-1} + \gamma U_{t+m-1}(x_{t+m}) \Big]. \quad (8)$$

We call $z_t^{\lambda,m}$ the m-step truncated TD(λ) return, or the *TTD(λ, m) return* for time t. It is an approximation of the TD(λ) return, defined as

$$z_t^{\lambda} = \sum_{k=0}^{\infty} (\gamma\lambda)^k \Big[r_{t+k} + \gamma(1-\lambda)U_{t+k}(x_{t+k+1}) \Big], \quad (9)$$

or recursively as

$$z_t^{\lambda} = r_t + \gamma(\lambda z_{t+1}^{\lambda} + (1-\lambda)U_t(x_{t+1})). \quad (10)$$

It can be shown (Cichosz, 1995) that the objectives of TD(λ) learning could be achieved inexpensively by updating at each time step the predicted utility of only one, current state, according to:

$$update^{\beta}(U,\ x_t,\ z_t^{\lambda} - U_t(x_t)). \quad (11)$$

The TTD procedure approximates Equation 11, which cannot be implemented directly, by

$$update^{\beta}(U,\ x_{t-m+1},\ z_{t-m+1}^{\lambda,m} - U_t(x_{t-m+1})). \quad (12)$$

This can be implemented using an m-element *experience buffer*, containing the records of x_{t-k}, a_{t-k}, r_{t-k}, and $U_{t-k}(x_{t-k+1})$ for all $k = 0, 1, \ldots, m-1$, where t is the current time step. The resulting algorithm is parameterized by λ and m values, which is written TTD(λ, m).

The crucial operation of the TTD procedure performed at each time step t is computing the TTD return for the experience from time step $t-m+1$, $z_{t-m+1}^{\lambda,m}$, on the basis of the r_{t-k} and u_{t-k} values for $k = 0, 1, \ldots, m-1$ stored in the experience buffer. The methods of performing this computation will not be presented here due to lack of space, but the interested reader can compare them to their bucket brigade counterparts given in Section 4 by referring to our work on TTD (Cichosz, 1995; Cichosz & Mulawka, 1995).

2.3 Bucket Brigade and TD(0): A Comparison

Note the apparent similarity of Equation 3 to Equation 2. If we assume a tabular representation of U and rewrite the former as

$$U_{t+1}(x_t) := U_t(x_t) + \beta[r_t + \gamma U_t(x_{t+1}) - U_t(x_t)], \quad (13)$$

the only difference is that $U_t(x_t)$ replaces $s_t(\mathbb{A}_t)$ and $U_t(x_{t+1})$ replaces $s_{t+1}(\mathbb{A}_{t+1})$. This basic observation is nothing original — it has been made before several times, e.g., by Dorigo and Bersini (1994), Wilson (1994) or Cichosz (1994).

The analogy between the bucket brigade and TD(0) would be perfect if the former used $s_t(\mathbb{A}_{t+1})$ instead of $s_{t+1}(\mathbb{A}_{t+1})$ in Equation 2. However, the difference should not be very significant in practice, especially for relatively small β, and we feel free to ignore it. Consequently, we may say that the bucket brigade algorithm corresponds to TD(0) and refer to it as BB(0). The reminder of this paper is devoted to deriving generalizations of this algorithm that could be referred to as BB(λ) and TBB(λ, m).

3 The BB(λ) Algorithm

Having discovered the close similarity between the bucket brigade strength update rule and the TD(0) prediction update rule, we are ready to look for a bucket brigade counterpart of the TD(λ) rule given by Equation 5, using eligibility traces updated according to Equation 7. What seems most natural is to associate an eligibility trace with each classifier c, designated by $e_c(t)$ at time t, discounted by a factor of $\gamma\lambda$ at each time step and incremented whenever c belongs to the current action set, $\mathbb{A}_t$. The strength of each classifier in the population, which is designated by $\mathbb{C}$, would be then modified (reinforced) to a degree determined by its eligibility trace. This idea is concretized by the algorithm presented in Figure 2.

In Step 5 the eligibility traces of all classifiers are discounted by the $\gamma\lambda$ factor and in the next step the traces of the members of $\mathbb{A}_t$ are incremented. Step 7 computes, also by applying an incremental update operation, $E(t) = \sum_{c\in\mathbb{C}} e_c(t)$. This value is subsequently used in Step 12 to normalize the primary and secondary reinforcements, r_t and γs. The original bucket brigade algorithm used $\mathbb{A}_t$ and $\mathbb{A}_{t-1}$, respectively, for this purpose.

Note that for $\lambda = 0$ we have $e_c(t) = 1$ if $c \in \mathbb{A}_t$ and $e_c(t) = 0$ otherwise. Accordingly, $E(t) = |\mathbb{A}_t|$ and the BB(λ) algorithm reduces to the algorithm from Figure 1.

4 The TBB(λ, m) Algorithm

Similarly as for TD(λ), eligibility traces are responsible for the increased computational costs of BB(λ) in comparison to BB(0). Although this inefficiency may be often negligible, because the bucket brigade algorithm is relatively costly anyway (mainly due to the expense of creating the match set, which requires matching the current input message against every classifier), we can get rid of eligibility traces using essentially the same idea upon which the TTD procedure is based.

4.1 Truncated Bucket Brigade Returns

We begin with the definition of the TBB(λ, m) return, the m-step truncated BB(λ) return:

$$z_t^{\lambda,m} = \sum_{k=0}^{m-2} (\gamma\lambda)^k \Big[r_{t+k} + \gamma(1-\lambda)s_{t+k+1}(\mathbb{A}_{t+k+1})\Big] + (\gamma\lambda)^{m-1}\Big[r_{t+m-1} + \gamma s_{t+m}(\mathbb{A}_{t+m})\Big]. \quad (14)$$

Now we only need an algorithm that would implement the following modification of Equation 2:

$$s(\mathbb{A}_{t-m}) := s(\mathbb{A}_{t-m}) + \beta[z_{t-m}^{\lambda,m} - s(\mathbb{A}_{t-m})], \quad (15)$$

```
At each time step $t$:
  1. observe current input message $x_t$;
  2. $\mathbb{M}_t := match\_set(x_t)$;
  3. $a_t := select\_action(\mathbb{M}_t)$;
  4. $\mathbb{A}_t := action\_set(\mathbb{M}_t, a_t)$;
  5. for each $c \in \mathbb{C}$ do
       $e_c(t) := \gamma\lambda e_c(t-1)$;
  6. for each $c \in \mathbb{A}_t$ do
       $e_c(t) := e_c(t) + 1$;
  7. $E(t) := \gamma\lambda E(t-1) + |\mathbb{A}_t|$;
  8. perform action $a_t$;
  9. observe immediate reward $r_t$;
 10. $s := 0$;
 11. for each $c \in \mathbb{A}_t$ do
       $s := s + s_c$;
 12. for each $c \in \mathbb{C}$ do
       (a) $s_c := s_c + \beta(\frac{r_t}{E(t)} - s_c)e_c(t)$;
       (b) $s_c := s_c + \beta\frac{\gamma s}{E(t-1)}e_c(t-1)$.
```

Figure 2: The BB(λ) algorithm.

intended to be a bucket brigade counterpart of Equation 12. Time step subscripts are omitted from s values in this equation, because the update which it describes will not be performed directly. Indeed, we want the TBB algorithm to be as similar to the original bucket brigade algorithm as possible. In particular, TBB$(0, 1)$ should be exactly equivalent to BB(0). That is why the above update operation will be implemented in two stages, described by the following two update rules, written using two different time variables for the sake of clarity:

$$s_{t_1+1}(\mathbb{A}_{t_1}) := s_{t_1}(\mathbb{A}_{t_1}) + \beta[r_{t_1} - s_{t_1}(\mathbb{A}_{t_1})] \tag{16}$$

at time t_1, and

$$s_{t_2+1}(\mathbb{A}_{t_2-m}) := s_{t_2}(\mathbb{A}_{t_2-m}) + \beta[z^{\lambda,m}_{t_2-m} - r_{t_2-m}] \tag{17}$$

at time $t_2 = t_1 + m$. Thus, at time t the members of both $\mathbb{A}_t$ and $\mathbb{A}_{t-m}$ have their strengths updated. On the other hand, the strengths of the classifiers in $\mathbb{A}_{t-m}$ are updated twice, at time $t - m$ and at time t. Under an unrealistic simplifying assumption that all action sets between time $t - m$ and t do not contain any common classifier, we could write the joint effects of these two updates as

$$s_{t+1}(\mathbb{A}_{t-m}) := s_{t-m}(\mathbb{A}_{t-m}) + \beta[z^{\lambda,m}_{t-m} - s_{t-m}(\mathbb{A}_{t-m})]. \tag{18}$$

4.2 The TBB Procedure

To implement this idea, an $(m+1)$-element experience buffer will be used, storing the records of $\mathbb{A}_{t-k}$, r_{t-k}, and $s_{t-k}(\mathbb{A}_{t-k})$ for all $k = 0, 1, \ldots, m$, where t is the current time step. The elements of such records will be referred to as $\mathbb{A}_{[k]}$, $r_{[k]}$, and $s_{[k]}$, respectively, which can be achieved by shifting appropriately the buffer's indices on each time tick. The reason why the buffer is 1 element longer than for TTD is that, while we can compute $U_t(x_{t+1})$ as soon as x_{t+1} can be observed, $s_{t+1}(\mathbb{A}_{t+1})$ is unknown until an action a_{t+1} is selected. Consequently, $s_{[k]}$ stores $s_{t-k}(\mathbb{A}_{t-k})$ instead of $s_{t-k}(\mathbb{A}_{t-k+1})$ and we need $m+1$ buffer elements if we want to use m-step truncated returns.

The resulting TBB(λ, m) algorithm is presented in Figure 3. For simplicity reasons the initial m time steps (when the experience buffer is not completely filled) are not covered by this algorithm. They are also ignored below in the discussion of the TBB return computation.

```
At each time step $t$:
  1. observe current input message $x_t$;
  2. $\mathbb{M}_t := match\_set(x_t)$;
  3. $a_t := select\_action(\mathbb{M}_t)$;
  4. $\mathbb{A}_t := action\_set(\mathbb{M}_t, a_t)$; $\mathbb{A}_{[0]} := \mathbb{A}_t$;
  5. perform action $a_t$;
  6. observe immediate reward $r_t$; $r_{[0]} := r_t$;
  7. $s_{[0]} := 0$;
  8. for each $c \in \mathbb{A}_{[0]}$ do
       (a) $s_{[0]} := s_{[0]} + s_c$;
       (b) $s_c := s_c + \beta(\frac{r_{[0]}}{|\mathbb{A}_{[0]}|} - s_c)$;
  9. $z := tbb\_return(0, m)$;
 10. for each $c \in \mathbb{A}_{[m]}$ do
       $s_c := s_c + \beta\frac{z - r_{[m]}}{|\mathbb{A}_{[m]}|}$;
 11. shift the indices of the experience buffer.
```

Figure 3: The TBB(λ, m) algorithm.

The operation of Step 9, written as $tbb\_return(0, m)$, computes the TBB(λ, m) return for time $t - m$, $z_{t-m}^{\lambda,m}$, using the values of $r_{[k]}$ for $k = 1, 2, \ldots, m$ and $s_{[k]}$ for $k = 0, 1, \ldots, m-1$. This TBB return is then used to reinforce the classifiers in $\mathbb{A}_{[m]}$. It is straightforward to verify that for $\lambda = 0$ and $m = 1$ the algorithm reduces to the very same BB(0) algorithm as presented in Figure 1.

4.3 TBB Return Computation

The most important thing to know about the TBB algorithm is how TBB returns are computed. Two methods existing for TTD returns can be easily adopted for this purpose.

The simpler, but less efficient iterative method is based on the repeated application of Equation 10. The bucket brigade version of this method may be written as follows:

```
1. $z := s_{[0]}$;
2. for $k = 1, 2, \ldots, m$ do
       $z := r_{[k]} + \gamma(\lambda z + (1-\lambda)s_{[k-1]})$.
```

To see how the same computation can be performed incrementally, we rewrite the definition of the TBB return for time t in the following form:

$$z_t^{\lambda,m} = S_t^{\lambda,m} + T_t^{\lambda,m}, \tag{19}$$

where

$$S_t^{\lambda,m} = \sum_{k=0}^{m-1} (\gamma\lambda)^k \Big[r_{t+k} + \gamma(1-\lambda)s_{t+k+1}(\mathbb{A}_{t+k+1})\Big], \tag{20}$$

$$T_t^{\lambda,m} = (\gamma\lambda)^m s_{t+m}(\mathbb{A}_{t+m}). \tag{21}$$

The $T^{\lambda,m}$ term can be computed directly in constant time. For the $S^{\lambda,m}$ it is easy to verify that

$$\begin{aligned} S_{t+1}^{\lambda,m} = \frac{1}{\gamma\lambda}\Big\{ & S_t^{\lambda,m} - \Big[r_t + \gamma(1-\lambda)s_{t+1}(\mathbb{A}_{t+1})\Big] \\ & + (\gamma\lambda)^m \Big[r_{t+m} + \gamma(1-\lambda)s_{t+m+1}(\mathbb{A}_{t+m+1})\Big]\Big\}. \end{aligned} \tag{22}$$

This observation leads to the following incremental algorithm for computing $z_{t-m}^{\lambda,m}$ in constant time for arbitrary m:

```
1. $S := \frac{1}{\gamma\lambda}\Big[S - (r_{[m+1]} + \gamma(1-\lambda)s_{[m]}) + (\gamma\lambda)^m(r_{[1]} + \gamma(1-\lambda)s_{[0]})\Big]$;
2. $T := (\gamma\lambda)^m s_{[0]}$;
3. $z := S + T$;
```

where $r_{[m+1]}$ is an auxiliary variable storing the value of $r_{[m]}$ from the previous time step. The existing TTD literature (Cichosz, 1995; Cichosz & Mulawka, 1995) contains a more elaborated discussion of the original TTD return computation methods, which applies to TBB returns as well.

5 Conclusion

This paper has shown how the ideas studied in the area of TD-based reinforcement learning algorithms can be ported to the related, but independently developed area of classifier systems. Two simple techniques, eligibility traces and truncated returns, that allow one to use TD(λ) for general λ, have been incorporated into the bucket brigade algorithm, yielding the BB(λ) and TBB(λ, m) algorithms. Our deep surprise that the apparent and widely known similarity between the bucket brigade algorithm and TD(0) has never been practically exploited before was the major motivation for this research.

Using positive λ with TD-based RL algorithms has been found to usually yield a significant learning speedup. We expect the same to occur for classifier systems. It seems unquestionable that using either of the two algorithms proposed in this paper instead of the standard BB(0) algorithm will result in faster learning for delayed reinforcement tasks. However, it might be questionable whether the scale of this improvement would be equally impressive as for AHC or Q-learning (e.g., Lin, 1993; Cichosz, 1995; Cichosz & Mulawka, 1995). It certainly requires a more thorough theoretical analysis and an empirical verification on a variety of tasks, which appears to the most important field for future work on BB(λ) and TBB(λ, m).

Another interesting question that should be addressed by future research is whether TD(λ)-like extensions are also possible with other versions of the bucket brigade algorithm. While we can probably say 'yes' in advance for any versions of the implicit bucket brigade algorithm, where credit is transferred along temporal chains, it is not so evident with the more complex "canonical" bucket brigade (Booker et al., 1989), using causal rather than temporal chains. We believe that answers to all the remaining questions would be valuable contributions to both the CS and TD research.

References

Booker, L. B., Goldberg, D. E., & Holland, J. H. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, *40*, 235–383.

Cichosz, P. (1994). Reinforcement learning algorithms based on the methods of temporal differences. Master's thesis, Institute of Computer Science, Warsaw University of Technology.

Cichosz, P. (1995). Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning. *Journal of Artificial Intelligence Research*, *2*, 287–318.

Cichosz, P., & Mulawka, J. J. (1995). Fast and efficient reinforcement learning with truncated temporal differences. In *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*. Morgan Kaufmann.

Dorigo, M., & Bersini, H. (1994). A comparison of Q-learning and classifier systems. In *Proceedings of From Animals to Animats, the Third International Conference on Simulation of Adaptive Behavior (SAB-94)*.

Lin, L.-J. (1993). *Reinforcement Learning for Robots Using Neural Networks.* Ph.D. thesis, School of Computer Science, Carnegie-Mellon University.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning.* Ph.D. thesis, Department of Computer and Information Science, University of Massachusetts.

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

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* Ph.D. thesis, King's College, Cambridge.

Wilson, S. W. (1994). ZCS: A zeroth order classifier system. *Evolutionary Computation*, *2*, 1–18.