

Approaches for the RL Competition 2009

Jose Antonio Martin H.
JAMH Team

I am from Madrid, Spain and I maintain a page about RL at:
<http://www.dia.fi.upm.es/~jamartin/download.htm>

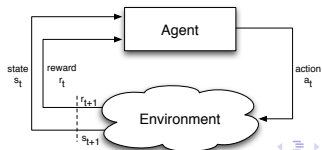


Table of contents

- 1 Introduction
- 2 The PolyAgent
- 3 Particularities for the 2009 PolyAthlon Domain

Introduction

We have run on two domains:

- 1 The PolyAthlon.
- 2 The Helicopter Hovering.

And, we have followed two different approaches, respectively:

- 1 An unpublished and permanently rejected $TD(\lambda)$ method called $kNN-TD(\lambda)$ for PolyAthlon.
- 2 An ad hoc Online-interaction Neuro-Evolution method for Helicopter.

PolyAthlon



The Basic Solution Method

- We used a very simple $TD(\lambda)$ algorithm.
- We used a function approximator based on weighted k -NN.
- We established the operational parameters by HUMAN RL.
- We suffered a lot (by courtesy of Brian and Adam).

PolyAgent Init method

Optimistic initialization, relatively low initial random exploration ($\epsilon = 0.01$), no discount, and relatively large eligibility parameter ($\lambda = 0.95$).

```
def agent_init(self, taskSpec):
    # Construct and init a function approximator
    self.Q = kNNQ(self.nactions, self.input_ranges, nelems=False, npoints=500000, k=2**6, alpha=50.0, lm=0.95)
    self.Q.Q *= 0.0
    self.Q.Q += 1.0
    self.Q.alpha = 50.0
    self.Q.lm = 0.95

    # Set the action selection strategy
    self.SelectAction = e_greedy_selection(epsilon=0.01)
    #self.SelectAction = e_softmax_selection(epsilon=0.5)

    # No discount
    self.gamma = 1.0
```

PolyAgent Start method

```
def agent start(self, observation):  
    self.s = array(observation.doubleArray, float32)  
    self.sp = array(observation.doubleArray, float32)  
    # Select initial action and get its value (Va)  
    self.a, Va = self.SelectAction(self.s)  
    act = self.actionlist[self.a]  
    self.action.intArray = [act]  
    return self.action
```

PolyAgent Step method

I hate rewards with value zero, so I avoid it.

```
def agent step(self, reward, observation):
    self.sp = array(observation.doubleArray, float32)

    # select action prime
    self.ap, Vp = self.SelectAction(self.sp)
    act = self.actionlist[self.ap]
    self.action.intArray = [act]

    # Update the Q values, that is, learn from the experience
    if reward != 0:
        target_value = reward + self.gamma * Vp #off-policy GOOD
    else:
        target_value = 1.0 + self.gamma * Vp #off-policy GOOD

    self.Q.Update(self.s, self.a, target_value)

    #update the current variables
    self.s = self.sp
    self.a = self.ap

    return self.action
```

PolyAgent End method

We tried to detect whether the MPD is a bandit problem in order to use an even more optimistic initialization. Also to set $\lambda = 0$ avoiding updating the entire matrix and saving a lot of unnecessary computational time.

```
def agent_end(self, reward):
    if self.steps <= 2:
        if self.episode==0:
            self.Q.Q*=0
            self.Q.Q+=10
            self.Q.lm = 0.0
            self.Q.alpha = 50.0
            self.Q.Update(self.s, self.a, reward)

        else:
            self.Q.Update(self.s, self.a, reward)
            self.Q.ResetTraces()

    self.SelectAction.epsilon *= 0.99
```

Use the same strategy as last year?

PolyAthlon (Last year RLC2008):

- Regular grid of 21 points in $[0, 1]$ for all the $N = 4$ state variables, i.e. a matrix of $21^4 \times 4$ (states \times actions) with $k = 3^4$, i.e. 81 neighbors for each observation.

PolyAthlon 2009 first attempts:

- Regular grid of 11 points in $[0, 1]$ for all the $N = 6$ state variables, i.e. a matrix of $11^6 \times 6$ (states \times actions) with $k = 2^6$, i.e. 64 neighbors for each observation resulting in

**about 4 days to complete a proving run: AN
UNACCEPTABLE APPROACH.**

Strategy for 2009 PolyAthlon

The $kNN-TD$ algorithm can be also initialized with an arbitrary set of points, so the strategy which is nearer the regular grid is the uniformly distributed points.

Configurations: PolyAthlon 2009:

- $M = 50,000$ uniformly random generated points, i.e. a matrix of $50,000 \times 6$ (states \times actions) with $k = 2^6$, i.e. 64 neighbors for each observation resulting in **about 1 hour to complete a proving run.**
- $M = 75,000$ uniformly random generated points, i.e. a matrix of 75000×6 (states \times actions) with $k = 2^6$, i.e. 64 neighbors for each observation resulting in **about 2 hour to complete a proving run.**
- \vdots
- **$M = 500,000$** five hundred thousands uniformly random generated points, i.e. a matrix of $500,000 \times 6$ (states \times actions) with $k = 2^6$, i.e. 64 neighbors for each observation resulting in **about 10 hour to complete a proving run.**

Questions?

Questions?

Thanks

Thank you very much !!!

The $kNN-TD(\lambda)$ Perception

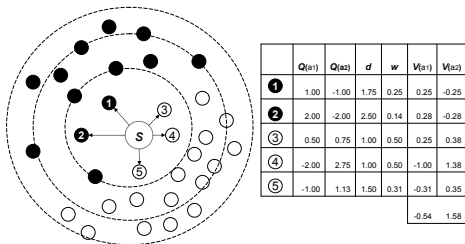


Figure: The k -nearest neighbors rule at some time step for a value of $k = 5$.

- ① A set (knn) which contains the k -nearest neighbors of the observation s .
- ② A vector (w) which contains the weights (or activations) of each element in the knn set.

An example applied to the Mountain Car problem

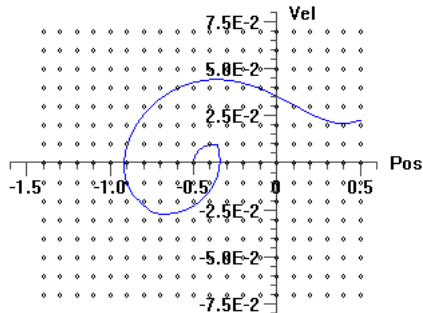


Figure: State space for the MountainCar problem

The $kNN-TD(\lambda)$ Perception and Prediction

$$w_i = \frac{1}{1 + d_i^2} \quad \forall i \in [1, \dots, k] \quad (1)$$

$$p(i) = \frac{w_i}{\sum w_i} \quad \forall i \in knn, \quad (2)$$

In this way, the expected value of the future reward for a given action a is estimated by (3).

$$\langle V(a) \rangle = \sum_{i=1}^{knn} Q(i, a)p(i), \quad (3)$$

hence $p(i)$ is the probability $P(V(a) = Q(i, a)|s)$ that $V(a)$ takes the value of neighbor i for action a , i.e. $Q(i, a)$, given the state of the environment s . Then the action selection mechanism can directly derive a greedy policy from the expectations for each perceptual representation as shown in (4).

$$\operatorname{argmax} V(a) \quad (4)$$

The k -NNQ(λ) Learning at state s'

For whichever (off/on policy) method be followed, an expected (predicted) value is calculated:

$$\langle V'(a) \rangle = \sum_{i=1}^{knn'} Q(i, a) p'(i) \text{ for } s', \quad (5)$$

where knn' is the set of the k -nearest neighbors of s' , $p'(i)$ are the probabilities of each neighbor in the set knn' and the TD error (δ) can be calculated as shown in (6) or (7).

$$\delta = r + \gamma V'(a') - V(a) \quad (6)$$

$$\delta = r + \gamma \max_a V'(a) - V(a) \quad (7)$$

Thus we can simply update the prediction of each point in the knn set by applying the update rule (8).

$$Q(i, a)_{t+1} = Q(i, a)_t + \alpha \delta p(i) \quad \forall i \in knn \quad (8)$$

The $kNN-TD(\lambda)$ Algorithm Probability Traces

$$e(knn, j) = \begin{cases} p(knn) & j = a \\ 0 & j \neq a \end{cases} \quad (9)$$

where knn is the set of k -nearest neighbors of the observation s , $p(knn)$ are its corresponding probabilities and a is the last performed action. The traces always decay following the expression (13).

$$e_{t+1} = \gamma \lambda e_t \quad (10)$$

As we can see, the eligibility traces scheme is replaced by a probability traces scheme given the fact that the stored values are just probabilities and not the value 1 as is used in classical eligibility traces theory.

$$Q_{t+1} = Q_t + \alpha \delta e \quad (11)$$

The $kNN-TD(\lambda)$ Algorithm Pseudo Code

```

Initialize the space of classifiers  $cl$ 
Initialize  $Q(cl, a)$  arbitrarily and  $e_i(cl, a) \leftarrow 0$ 
repeat {for each episode}
  Initialize  $s$ 
   $knn \leftarrow k$ -nearest neighbors of  $s$ 
   $p(knn) \leftarrow$  probabilities of each  $cl \in knn$ 
   $V(a) \leftarrow Q(knn, a) \cdot p(knn)$  for all  $a$ 
  Chose  $a$  from  $s$  according to  $V(a)$ 
  repeat {for each step of episode}
    Take action  $a$ , observe  $r, s'$ 
     $knn' \leftarrow k$ -nearest neighbors of  $s'$ 
     $p(knn') \leftarrow$  probabilities of each  $cl \in knn'$ 
     $V'(a) \leftarrow Q(knn', a) \cdot p(knn')$  for all  $a$ 
    Chose  $a'$  from  $s'$  according to  $V'(a)$ 
    Update  $Q$  and  $e$ :
       $e(knn, \dots) \leftarrow 0$  {optional}
       $e(knn, a) \leftarrow p(knn)$ 
       $\delta \leftarrow r + \gamma V'(a') - V(a)$ 
       $Q \leftarrow Q + \alpha \delta e$ 
       $e \leftarrow \gamma \lambda e$ 
     $a \leftarrow a', s \leftarrow s', knn \leftarrow knn', p \leftarrow p'$ 
  until  $s$  is terminal
until learning ends

```

As we can see, the eligibility traces scheme is replaced by a probability traces scheme given the fact that the stored values are just probabilities and not the value 1 as is used in classical eligibility traces theory.

$$e(knn, j) = \begin{cases} p(knn) & j = a \\ 0 & j \neq a \end{cases} \quad (12)$$

where knn is the set of k -nearest neighbors of the observation s , $p(knn)$ are its corresponding probabilities and a is the last performed action. The traces always decay following the expression (13).

$$e_{t+1} = \gamma \lambda e_t \quad (13)$$