On Entering a New Research Domain: a Case Study

Colin Fyfe

This paper discusses the process by which a seasoned researcher begins an investigation into a new field of study. It is a somewhat personal case study into my initiation into reinforcement learning, something which is ongoing rather than the finished article. I use this to encourage non-research active staff to take the plunge and become research active.

1 INTRODUCTION

In 1995, I defended my PhD thesis and joined this university as a Lecturer. Since then, on fairly regular occasions, I have had discussions with other academic members of staff about how to initiate a research direction if one has no background in formal academic research. Often underlying these discussions is an assumption that I, holding a PhD, am somehow different from the non-PhD holder. I have now supervised to completion 16 PhDs (and been second supervisor to a few more) so I am now in a position to state fairly definitively that there is no one criterion which separates Dr. Smith from Mr. Smith. Indeed, on completion of a PhD, some people never go near research again while others continue as active researchers throughout their lives. In this, PhD graduates are no different from BSc graduates.

So how does someone who has no background in research become an active researcher? The stages are very easily outlined:

1. Select a topic that interests you.
2. Become intimate with the topic.
    a. Get a good book(s) on the topic.
    b. Implement things.
    c. Google for current information
3. Ask questions. Write down questions and answers.

I am of the breed of researchers that does not stick to a single topic. I have core competencies (artificial neural networks with unsupervised learning) but I have continued to find new interests, all of which to date have been in the field of computational intelligence. Recently I have taken on a student who has created a Motocross game (see this edition of the journal) and he has found that evolutionary methods of learning, while not as efficient as neural network methods, have found solutions to driving his bikes that a human would not find. This has suggested to me that we could fruitfully apply reinforcement learning to this topic.

Thus I have identified a topic of interest to me but it is something about which I know little. Therefore (stage 2), I bought a copy of what seems to be the current bible on the topic [1]. Now this is a very readable and easily understandable book. But in order to ensure that I really understand it, I must implement some simple programs. This “getting your hands dirty” stage is time-consuming but absolutely essential. There is code available on the net for this topic but other people’s code will have bells and whistles attached to it, non-essentials which disguise the core ideas. Also, I personally have more confidence in my comprehension of a new topic when I have started off with a clean sheet of paper and come up with some interesting results. The next few sections illustrate my meanderings in this area. Note also that this report is satisfying stage 3: I am writing a report to myself about this topic.

One final point to make is that every individual brings his own history to the research table. This indeed is what makes research by anyone, whatever their background, a viable activity: as I venture into a new topic, I am aware that I am the novice, however I also know that no one has had quite the same experiences as Colin Fyfe and that perhaps I can bring some of my previous expertise to bear on the problems in this new area. Everyone, no matter how little formal research expertise they have, can be assured that they bring a fresh set of experiences to this new field.

2 REINFORCEMENT LEARNING

In computer games, the software agent is typically known as the AI. Just as there are many different types of supervised and unsupervised learning, so there are many different types of reinforcement learning. Reinforcement learning is appropriate for an AI or agent which is actively exploring its environment and also actively exploring what actions are best to take in different situations. Reinforcement learning is so-called because, when an AI finds out it has performed a beneficial action, it receives some reward which reinforces its tendency to perform that beneficial action again. An excellent overview of reinforcement learning (on which much of the following is based) is [0].

There are two main characteristics of reinforcement learning:

1. Trial-and-error search. The AI performs actions appropriate to a given situation without being given instructions as to what actions are best. Only subsequently will the AI learn if the actions taken were beneficial or not.
2. Reward for beneficial actions. This reward may be delayed because the action though leading to a reward (the AI wins the game) may not be (and typically is not) awarded an immediate reward.

Thus the AI is assumed to have some goal which in the context of games is liable to be win the game, drive the car as fast as possible, defeat the aliens and so on. Since the AI has a defined goal, as it plays, it will learn that some actions are more beneficial than others in a specific situation. However this raises the exploitation/exploration dilemma: should the AI continue to use a particular action in a specific situation or should it try out a new action in the hope of doing even better. Clearly the AI would prefer to use the best action it knows about for responding to a specific situation but it does not know whether this action is actually optimal unless it has tried every possible action when it is in that situation. This dilemma is sometimes solved by using ε-greedy policies which stick with the current actions with probability 1-ε but investigate an alternative action with probability ε.

Henceforth we will call the situation presented to the AI as the state of the environment. Note that this state includes not only the passive environment itself but also any changes which may be wrought by other agencies (either other AIs or humans) acting upon the environment. This is sometimes described as the environment starts where the direct action of the AI stops i.e. it is everything which the AI cannot directly control. Every state has a value associated with it. This value is a function of the reward which the AI gets from being in that state but also takes into account any future rewards which it may expect to get from its actions in moving from that state to other states which have their own associated rewards. We also create a value function for each action taken in each state.

In the next section, we will formally define the main elements of a reinforcement learning system.

3 THE MAIN ELEMENTS

Formally, we can identify [0] 4 main elements of a reinforcement learning system:

1. **A policy**: this determines what action the agent can take in each state. It provides a mapping from the perceived state of the environment to actions which can be taken in that state. It may be deterministic such as a simple look-up table or it may be stochastic and associate probabilities with actions which can be taken in that state.

2. **A reward function**: this defines the goal of the system by providing a reward (usually a floating point number) given to the AI for being in each state. Note that rewards can be positive or negative (i.e. penalties) and so define good and bad events for the AI. The AI's task is to maximise its rewards (over the whole game). The reward function is external to the AI; the AI cannot change the reward function during play but will instead change its policies in order to maximise its long term rewards.

3. **A value function**: which indicates the long term value of a state when operating under the current policy. Thus, for state s, when operating under policy π, and using $E_\pi\{.\}$ to indicate expected value when using policy π, we have a value function $V^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\}$ which is giving the long term value of the state in terms of the discounted rewards which can be expected when using this policy. Since the discount rate, γ<1, the value of the rewards at time t decreases the further into the future they are given. $V^\pi(s)$ is called the state-value function for policy π. We also have an action value function

$Q^\pi(s,a) = E_\pi\{R_t \mid s_t = s, a_t = a\} = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\}$ which measures the expected return from taking action a in state s. A greedy policy would always take the action which had the greatest action value. Note that whereas the rewards are immediate (associated with each state), the values take into account long-term success or failure and are predictions rather than immediate rewards: the AI will know precisely his reward/penalty for being in the current state but will use the value functions to predict or estimate his future rewards. Note that if γ=0, we are simply taking the reward for moving under that policy to the next state, an extremely myopic reward function: $Q^\pi(s,a) = E_\pi\{R_t \mid s_t = s, a_t = a\} = E_\pi\{r_{t+1} \mid s_t = s, a_t = a\}$ At the other extreme, we may have γ=1 and perform no discounting at all which is usually only useful if we have a limited time span so that $Q^\pi(s,a) = E_\pi\{R_t \mid s_t = s, a_t = a\} = E_\pi\{\sum_{k=0}^{K} r_{t+k+1} \mid s_t = s, a_t = a\}$.

4. A model of the environment: Not all reinforcement learning methods require such a model but those that do use these models for planning. The model is required to captures some essence of the environment so that it can be used to predict the next state of the environment given its current state and the AI's action. For a reinforcement method without such a model, every action is a trial-and-error action.

These are the bare essentials of a reinforcement learning system. The central importance of the reward functions must be emphasised. Note also the double uncertainty associated with the reward functions: not only are they stochastic in that they are attempting to predict future rewards but we are often, while learning, having to estimate the functions themselves. This gives central importance to finding ways of accurately and efficiently estimating the correct values associated with specific policies.

4 FINDING THE BEST POLICY

There are a number of different ways of finding the optimal policy. We start with the simplest, the Action-Value methods before discussing more sophisticated methods under the headings Dynamic Programming, Monte Carlo methods and Q-learning.

4.1 Action-Value Methods

Let the true value of an action, a, be Q*(a) while its estimated value at time t is Qt(a). Then, if action a has been chosen k times prior to t resulting in rewards r1, r2, …,rk, then $Q_t(a) = \frac{r_1 + r_2 + \ldots + r_k}{k}$, the average reward each time it has been performed. It is possible to show that $Q_t(a) \to Q^*(a),$ as $k \to \infty$. A totally greedy rule would select the action, a*, for which $Q_t(a^*) = \max_a Q_t(a)$, the action with the greatest perceived pay-off. However we wish all actions to be sampled and so we have to create a non-zero probability that all non-optimal actions will be used at some stage. The three most common approaches are

Use an ε-greedy method. Select a random action with probability ε and the optimal action with probability 1-ε. Thus for an infinite learning process, each action will be sampled infinitely often and its reward can be calculated.

Select each action with probability $\frac{\exp(Q_t(a)/\tau)}{\sum_b \exp(Q_t(b)/\tau)}$ which is known as softmax action selection.

Make an optimistic assessment of the value of each action at the start of the simulation. Any reward received will be less than this and so more exploration will be carried out.

The first method has the disadvantage that the second most optimal action has the same probability of being selected as the worst action, whereas the second method chooses actions proportional to the exponent of their perceived rewards. The third method is only used for stationary problems in which we use an update method which washes out the original estimate of the value. This is not the case for non-stationary problems for which we use a constant update rate (see below).

We note that we may incrementally update the values since

$$Q_{k+1} = \frac{1}{k+1}\sum_{i=1}^{k+1} r_i = \frac{1}{k+1}(r_{k+1} + \sum_{i=1}^{k} r_i) = \frac{1}{k+1}(r_{k+1} + kQ_k) = Q_k + \frac{1}{k+1}(r_{k+1} - Q_k)$$

In this form, we can see that we are adjusting the estimated value incrementally to make it more like the current reward. For a non-stationary reward, we may decide to keep the step size constant to give $Q_{k+1} = Q_k + \alpha(r_{k+1} - Q_k)$ which will continue to track the changing values. One disadvantage of this is that the original value estimated at the start of the simulation remains as a bias in the estimated value after training, since

$$Q_{k+1} = Q_k + \alpha(r_{k+1} - Q_k) = \alpha r_{k+1} + (1-\alpha)Q_k = \alpha r_{k+1} + (1-\alpha)r_k + (1-\alpha)^2 Q_{k-1}$$

$$= \sum_{i=1}^{k+1} \alpha(1-\alpha)^{k+1-i} r_i + (1-\alpha)^{k+1} Q_0$$

which is a weighted sum of the rewards but also the bias caused by the original estimate of the value of the action.

This is the first algorithm which I programmed in this area. The results are not very interesting (actually entirely predictable) but satisfying in that they are predictable and so I gain some automatic feedback that I am on the right track.

4.2 Dynamic Programming

Dynamic programming is a set of algorithms that can find optimal policies if they are given an accurate model of the environment. They typically iterate between

Policy Evaluation: let π(s,a) be the probability of performing action a when in state s, and let $P_{ss'}^a$ be the probability that we transit from state s to state s' after performing action a. Then the value for state s can be calculated from the values of all other states s' using a set of updates derived from the Bellman equations: $V_{k+1}(s) = E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s\} = \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a \{R_{ss'}^a + \gamma V_k(s')\}$

This is iterated over all states so that we are evaluating the policy as a whole while keeping the policy fixed.

Policy Improvement: now we know how good the current policy is, we can attempt to improve it. Thus we may select one action which is different from the current policy and then subsequently revert to the original policy. If we find that the new policy (the existing policy + the change) is better than the existing policy, we accept the new policy. This is a greedy method.

$$\pi'(s) = \arg\max_a Q^\pi(s,a) = \arg\max_a E\{r_{k+1} + \gamma V^\pi(s_{k+1}) \mid s_k = s, a_k = a\}$$
$$= \arg\max_a \sum P_{ss'}^a \{R_{ss'}^a + \gamma V^\pi(s')\}$$

It can be shown that these two iterations will converge to the optimal policy. [0] make a distinction between the algorithm above and that in which we perform several sweeps through the first (policy evaluation) stage. However it seems to be the case that there is no general rule as to which method is best; it is dependent on the particular problem we are dealing with. Indeed, in the policy evaluation part of the program, we may either have all policies evaluated dependent on the values of the other policies at the start of the sweep or we may use each newly evaluated policy to update all the others as we come to them. The latter method tends to be slightly faster as we might have guessed.

Example

Now I am interested to see if I can get any interesting results with this algorithm and so I implement an example from [0]. A gambler is betting against the house and will stop when he has run out of money or when he reaches \$100. At each moment in time, he holds \$x and so may bet any amount of money between 0 and the minimum of \$x and \$(100-x). Thus the state is the gambler's capital, {1,2,…99} and his actions are to bet {1,2,…min(x,100-x)}. The optimal policy should maximise his probability of getting to \$100. If the probability of success on any one gamble, p=0.4 is known, we may solve this problem with the above algorithm. We set an initial value of the state 100 at 1; all other states have initial value of 0. The results are shown below.

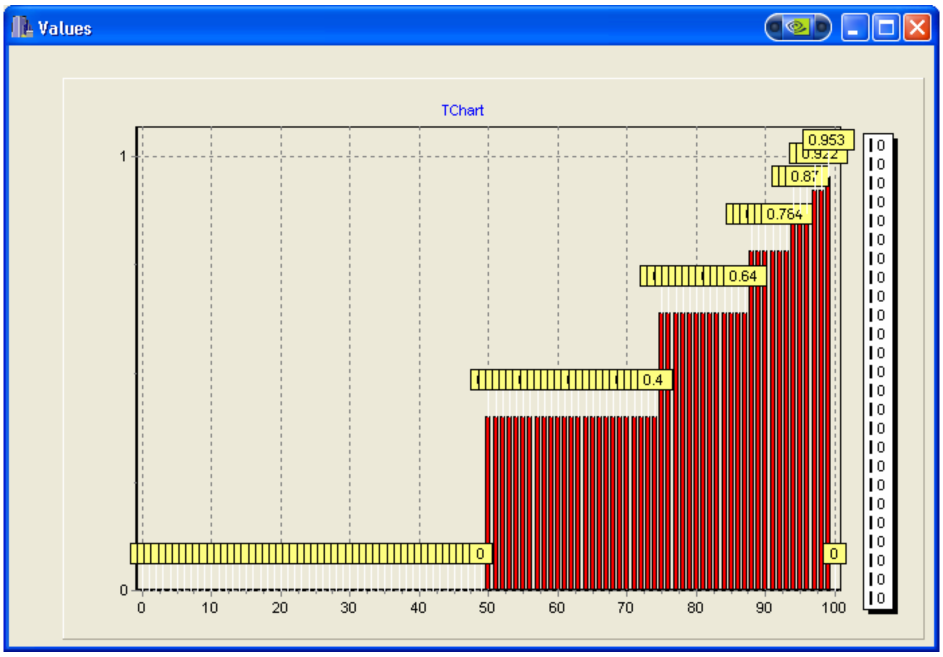

Figure 1 The estimated values of each state after the first iteration.

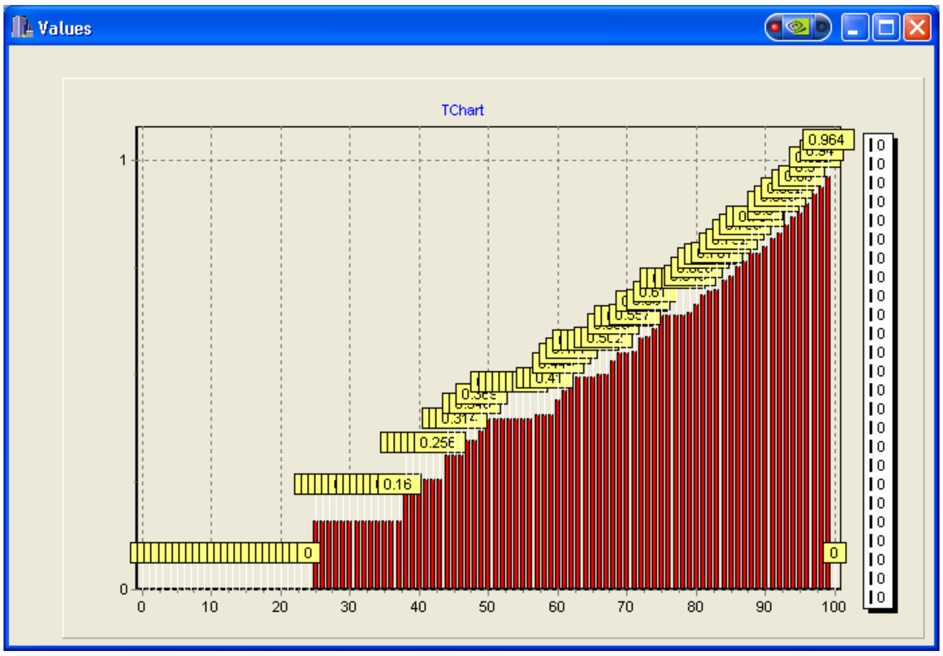

Figure 2 The estimated value of each state after only 2 iterations.

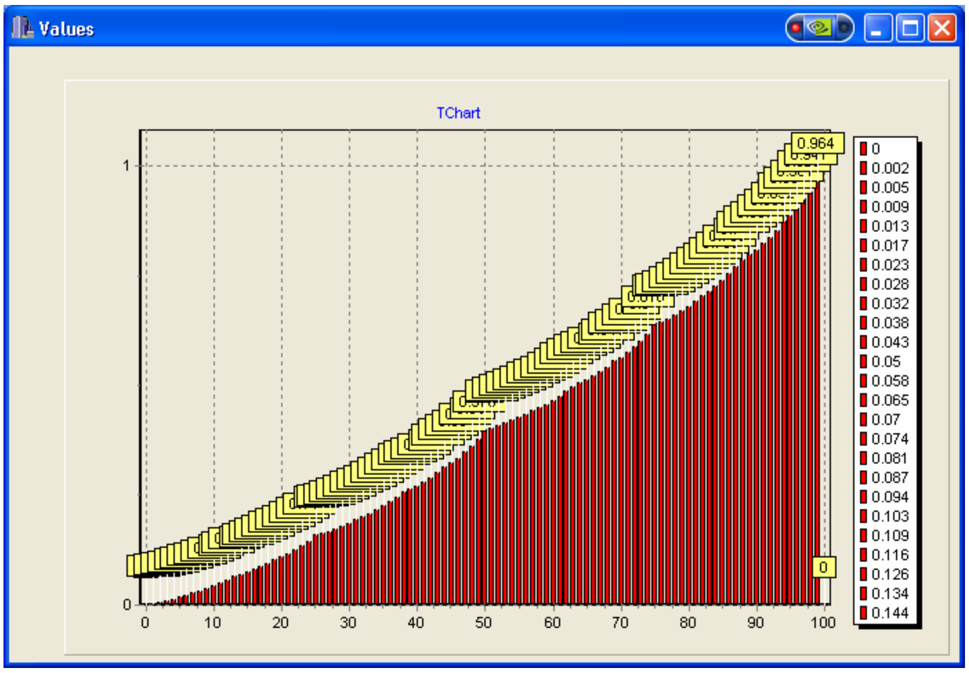

Figure 3 The final estimated values of each state.

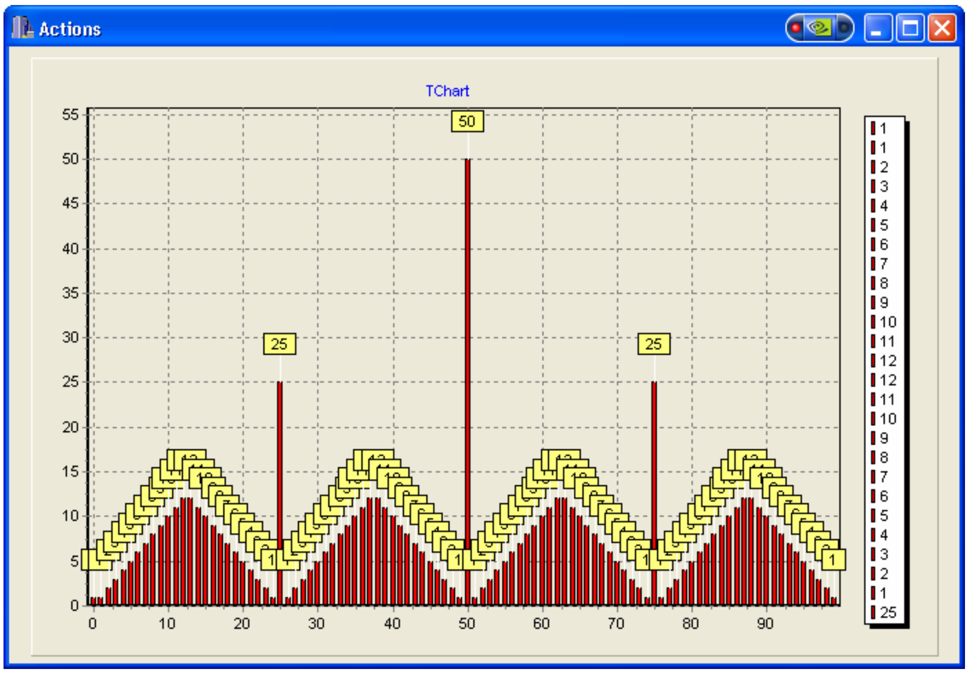

Figure 4 The final actions taken in each state. The height of the column represents the amount of the bet undertaken in the state.

Note that the policy is very strict: it says bet as little as you need to do to reach the next jump state when you should bet all your money on one gamble.

At this stage, I am fairly happy: I have implemented an algorithm, found the same results as those discussed in my bible [1] and found that I require to make a small arrangement for floating point precision which my bible neglected to mention. Note that this last point would not have occurred to me had I not got into the details of the simulation.

4.3 Monte Carlo Methods

Dynamic programming methods assume we have an accurate model of the environment. Monte Carlo methods make no such assumption but rather they learn from experiencing the environment and average the sample returns which they get from their experiences. Since we cannot investigate infinitely long sets of experiences, we assume that experiences occur in episodes which means we can always calculate returns and update policies and values when the episode completes. Having said that, we will see that there is a great deal of similarity between Monte Carlo methods and Dynamic Programming.

The simplest Monte Carlo method can be used for policy evaluation. It initialises the policy, π, for evaluation, and begins with a random state-value function and an empty list of returns for all states in S. Of course some states may not be known at the start of play so when a new state is encountered, it initially has an empty list of returns. Then we iterate (forever):

1. Generate an episode using π.
2. For each state s appearing in the episode
    a. R← return following first occurrence of s
    b. Append R to the set of returns for s.
    c. Value of s, V(s) = average of the list of returns

An alternative is to find R for every occasion s is visited.

Note first that we did not require $P^a_{ss'}$ or $R^a_{ss'}$ which is necessary for the Dynamic Programming methods. We are letting the episode of data speak for itself. Note also that the estimate for one state does not build upon the estimate of any other state and so, unlike Dynamic Programming, the computational expense of the method is independent of the number of states.

However it is actions on which we really wish to be focussed and which we really wish to optimise. One assumption which the Monte Carlo method employs is that every state-action pair has a non-zero probability of being the starting point of an episode. However the method has the same elements as before: the policy is evaluated and then, using the current evaluation, the policy is optimised. Thus the method is

1. Initialise for all s, a
    a. Q(s,a) randomly
    b. Π(s) randomly.
    c. Returns(s,a)← empty list.
2. Repeat (forever)
    a. Generate an episode using π.
    b. For each state-action pair, s-a, appearing in the episode
        i. R← return following first occurrence of s,a
        ii. Append R to the set of Returns(s,a).
        iii. Value of s, V(s) = average of the list of returns
3. For each s in the episode, π(s)= arg maxa Q(s,a)

However, it is particularly important in Monte Carlo methods, that we continue to explore the environment. Thus one version of Monte Carlo control uses an exploratory policy to generate episodes (the behaviour policy) while optimising the estimation policy. This is known as off-policy Monte Carlo control whereas the above algorithm uses on-policy control (we are generating episodes from the policy we are optimising). Off-policy control requires that every state-action pair which appears in the estimation policy has a non-zero probability of appearing in the behaviour policy. Also the updates to the estimation policy require to take into account the relative rate of appearances in each of the policies. See [0] for details.

Again I implement this but do not report on the results for space reasons.

5 TEMPORAL DIFFERENCE LEARNING

Temporal Difference (TD) learning is central to most modern reinforcement learning. It attempts to marry the best bits of dynamic programming – the updates to the values of policies being immediate – with the best of Monte Carlo methods – the updates being model-free and totally dependent on the experiences created by the current policies. Thus the TD-learning is a bootstrapping method (like dynamic programming) in that it updates value estimates based on existing estimates of values

$$V_{k+1}(s_t) \leftarrow V_k(s_t) + \alpha\{r_{t+1} + \gamma V_k(s_{t+1}) - V_k(s_t)\}$$

Note the double estimation present in this: instead of the expected value, it uses the value of a sample of the next state and also this value is itself used to bootstrap the estimation. Despite this double estimation, the TD method has been found to converge faster than Monte Carlo methods on stochastic tasks (which most interesting games are). To see why this is, [0, p143] give a nice argument which we repeat here. Let us imagine you observe the following 8 episodes.

A,0,B,0 B,1 B,1 B,1 B,1 B,1 B,1 B,0

The first episode starts in state A transits to B with a reward of 0 and terminates there with a reward of 0. The next 6 episodes all start at B and then terminate with a reward of 1. The final episode starts at B and immediately terminates with a reward of 0. We might agree that the value of B is 0.75 since that is the average return from B. TD learning would then say that on the one instance A was observed, it immediately moved to state B (with no reward) and so A’s value should be the same as that of B, 0.75. However a Monte Carlo simulation would say that, in the single episode in which A was observed, the total reward was 0 and so A’s value is 0. Thus Monte Carlo methods tend to fit the observed data very well but may be lacking in generalisation.

As with Monte Carlo methods, we can identify two types of TD control: on-policy and off-policy. On-policy TD control considers the transitions from one state-value pair to the next using $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$ after every transition. If st+1 is terminal, Q(st+1,at+1) =0. This method is also known as Sarsa since it concentrates on the pentuple (st+1,at+1,rt+1, st+1,at+1).

The off-policy TD control is also known as Q-learning and uses

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)]$$

i.e. regardless of the actual action taken during the episode, it uses the maximum possible value from the next state –action pair in its target.

Simulations

We illustrate the method with a route-finding task. The challenge is to find routes from anywhere on the square [0,9]X[0,9] to the point (9,9), the goal. In Figures 5 and 6, we show examples of learned routes from three random starting locations. In training, each episode had a maximum of 1000 states, each move gained a reward of -1 with a move off the board at the bottom or left being awarded -100 and a move off the other two sides being awarded -10. In both of the latter cases, the move was disallowed. The negative reward/penalty was to encourage a fast route to the goal. We see that both have found routes but of a different nature: Q-learning tends to go to one edge of the map and then follow it to the goal while Sarsa goes straight up the middle. These types of interesting divergences mean that both types of learning will form part of the arsenal of learning methods used by modern computational intelligence.

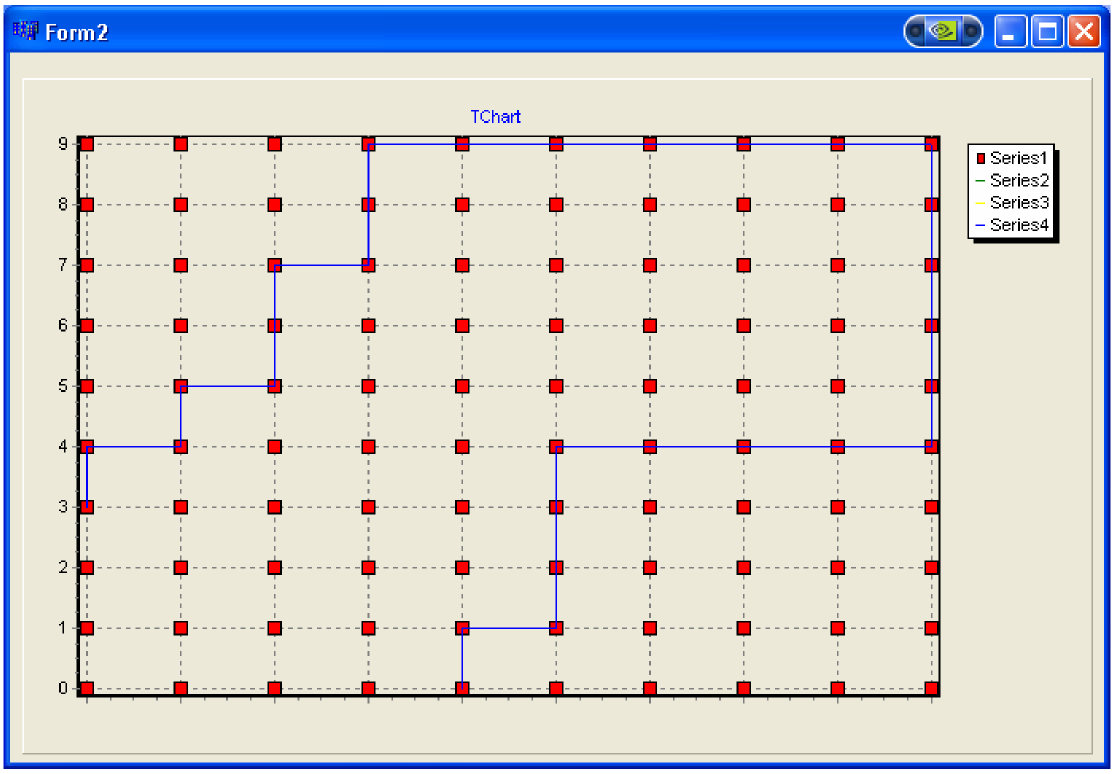

Figure 5 The routes found by Q learning

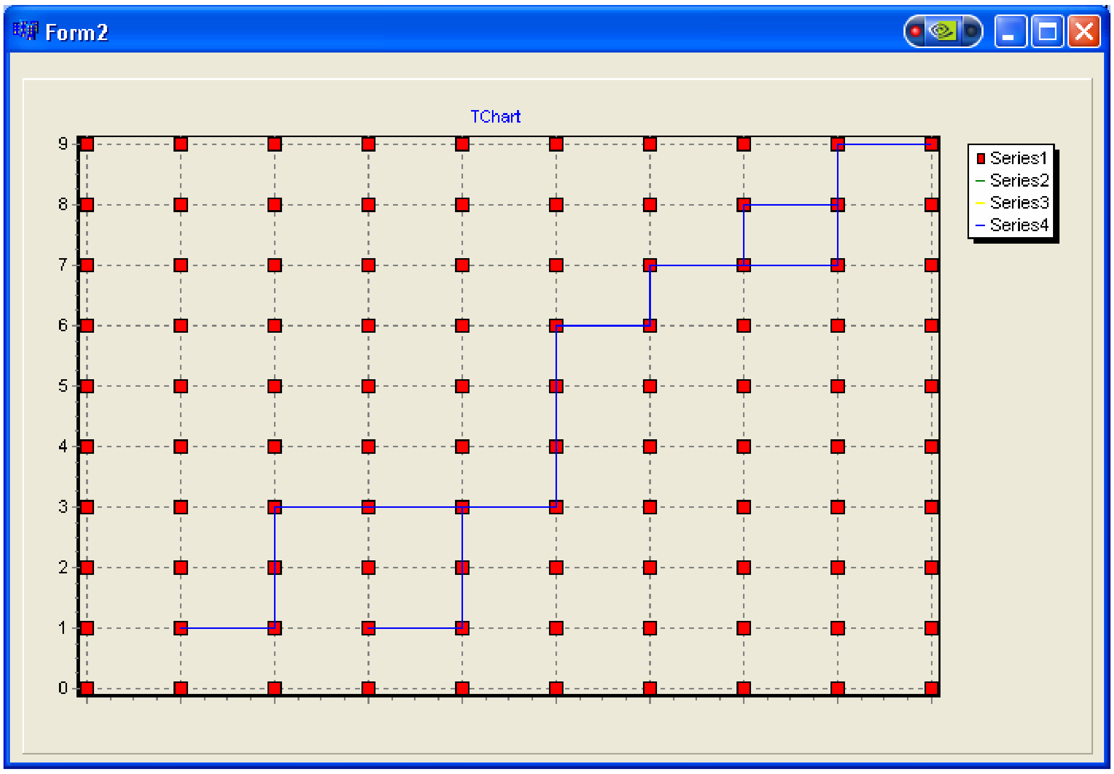

Figure 6 The solutions found by Sarsa learning.

6 TD(Λ) METHODS

The above methods are actually known as TD(0) methods. They change the valuation by looking one step ahead in the episode whereas Monte Carlo methods look ahead to the end of the episode before changing the valuation. This suggests that we might consider re-evaluations on the fly taking into account the states most recently met. One way to do this is keep a note of the trace of the states met and use it in the learning rule. Of course the trace itself needs to decay in time which is where the λ is used. An algorithm for TD(λ) is to generate an action and change state from st to st+1 to get reward rt+1 and then calculate

$$\delta \leftarrow r_{t+1} + \gamma V_k(s_{t+1}) - V_k(s_t)$$
$$e(s_t) \leftarrow e(s_t) + 1$$
$$\forall s$$
$$V(s) \leftarrow V(s) + \alpha\delta e(s)$$
$$e(s) \leftarrow \gamma\lambda e(s)$$

Note that the change to values and traces occurs for all states.

Simulations

We repeat the goal finding experiment with Sarsa(λ) with λ=0.8, γ=0.9 and α=0.01. Resulting routes from three initial starting points are shown in Figure 7. The additional computational overhead of this method was alleviated by training for only 10000 iterations

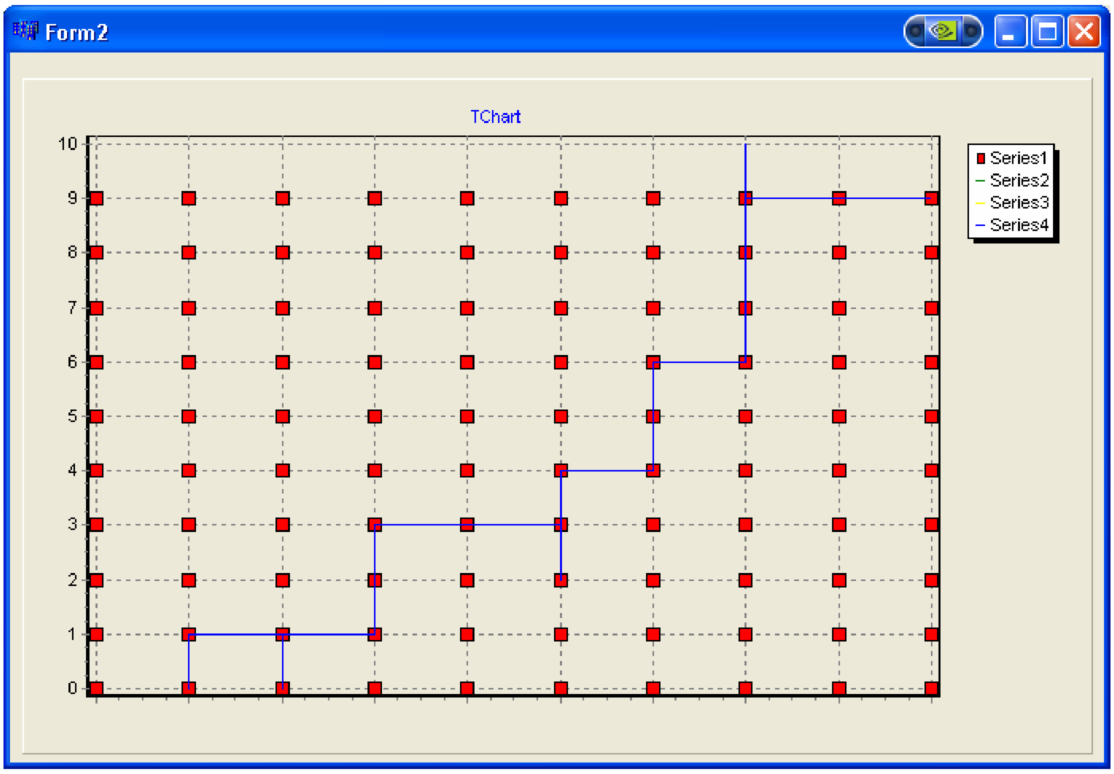

Figure 7 The route found by Sarsa(lambda) with lambda=0.8, gamma=0.9 and alpha=0.01.

Note that all three starting points quickly converged to the same route but one has experimented (unsuccessfully) with a different route at the point (7,9).

7 QUESTIONS AND ANSWERS

Before going on, note how detailed the above is: I can, at any time, re-create any of the above because it is written down in enough details that performing any simulation again is very easy. But now I can stand back a little and see how close we are to applying these techniques to the Motocross game. The first major difference I see is that the motocross game does not have a finite number of states. The states (current velocity, terrain) vary continuously as indeed do the actions (steer left/right, accelerate/decelerate, lean forward/backward). It is possible to quantise these states to get a finite set but a more attractive proposition is to learn the mapping from state-action pairs to rewards, something that has often been done with supervised artificial neural networks. But this suggests a possible research direction: I have spent a great deal of time recently becoming familiar with the topic of Gaussian Processes [2] which create supervised associations of just the type we are looking for. A possible research direction might be to use Gaussian Processes for this mapping. Naturally I tried to Google for this to see if it had been done already and found that unfortunately it had, but only in 2005. Therefore, now, after only 3 weeks work, I am only about 2 years behind the leading edge. This gives me great confidence that I can very quickly master these techniques and produce something noteworthy research-wise within a very short spell of time.

8 DISCUSSION

There are no real conclusions to be drawn from this case study. As you see, I have a very simple approach to new research. When engaging a new area, you are very much the new boy but this can and does change. As you gain expertise, you become recognised as an active researcher in the field and the next generation of new boys regard you as one of the old hands. My methodology is very much one of jumping in to try things out. In the case study given above, we have yet to be successful at innovating but, given that this is based on only three weeks work (which has been interrupted by all manner of other things), I am very encouraged to be thinking along the same lines as experienced practitioners in the field.

There is one aspect which I must concede: I am aware that I have been involved successfully in research in the past which gives me confidence that I can be successful in the future. However, there is no doubt in my mind that I was involved in innovation when I was a secondary school teacher trying to find the best ways to teach school pupils. We are all, whether we call it research or not, involved in this type of innovation and so I am confident that every academic can become an established researcher.

The above is clearly on-going work but the message I hope that every academic takes on board is that the activity of doing research is a very simple one indeed and there is no one who is incapable of it. All it requires is to jump out of the comfort zone where every thing is known and settled into a world where there is an infinite amount of unknown riches waiting to be discovered. I will be happy to talk to anyone about this anytime.

References

R. S. Sutton and A. G. Barto. Reinforcement Learning: an Introduction, MIT Press, 1998.

C. K. I. Williams and C. Rasmussen, Gaussian Processes, Springer, 2006.