AIXI Notes

Patrick Robotham

May 1, 2013

This is a collection notes on the paper [1] the emphasis is on description, most of the maths is glossed over.

1 The set up

We must attempt to solve the general reinforcement learning problem. An agent lives in some unknown environment, and interacts with the environment in cycles. In each cycle, the agent must choose some action, and then the environment gives the agent an observation and a reward. The agent's job is to get as much reward as it can from this environment.

1.1 String Notation

A string $x_1 x_2 \dots x_n$ of length n is denoted $x_{1:n}$.
The prefix $x_{1:j}$ of $x_{1:n}$ where $j \leq n$ is denoted by $x_{\leq j}$ or $x_{<j+1}$.
$ax_{1:n}$ denotes $a_1 x_1 a_2 x_2 \dots a_n x_n$.
$ax_{<}j$ denotes $a_1 x_1 a_2 x_2 \dots a_{j-1} x_{j-1}$.
The empty string is denoted by ϵ.
The concatenation of two strings s and r is denoted by sr.

1.2 Formalizing the reinforcement learning problem

$\mathscr{A}$ denotes the set of possible actions.
$\mathscr{O}$ denotes the set of possible observations.
$\mathscr{R}$ denotes the set of possible rewards.
$\mathscr{X}$ is the perception space $\mathscr{O} \times \mathscr{R}$.

Definition 1.1 A history h is an element of $(\mathscr{A} \times \mathscr{X})^* \cup (\mathscr{A} \times \mathscr{X})^* \times \mathscr{A}$.

Definition 1.2 An environment ρ is a sequence of conditional probability functions $\{\rho_0, \rho_1, \rho_2, \ldots\}$, where $\rho_n : \mathscr{A}^n \to \text{Density}(\mathscr{X}^n)$, that satisfies

$$\forall a_{1:n} \forall x_{<n} : \rho_{n-1}(x_{<n}|a_{<n}) = \sum_{x_n \in \mathscr{X}} \rho_n(x_{1:n}|a_{1:n}).$$

This condition corresponds to actions have no effects on earlier perceptions. Given an environment ρ, the predictive probability p is defined as

$$p(x_n|ax_{<n}a_n) = \frac{\rho_n(x_{1:n}|a_{1:n})}{\rho_{n-1}(x_{<n}|a_{<n})} \tag{1}$$

Thus

$$\rho_n(x_{1:n}|a_{1:n}) = p(x_1|a_1)p(x_2|ax_1a_2)\cdots p(x_n|ax_{<n}a_n) \tag{2}$$

2 Planning

MC-AIXI uses a technique called Monte-Carlo Tree Search to plan.

In our search tree, the nodes are histories. There are two types of nodes, *decision nodes* which end in observation-reward pairs, and *chance nodes* which end in actions.

To each node h, we associate an estimate of its future reward $\hat{V}(h)$ and a visit count $T(h)$. The visit count is the number of times h has been sampled.

Monte-Carlo tree search has four phases that are endlessly cycled see

1. Selection Phase: Traverse the search tree from the root to an existing leaf chance node n.
2. Expansion Phase: Add a new decision node as a child to n
3. Simulation Phase: Rollout n with the environment ρ to sample a possible future path
4. Backpropagation Phase: Update the value estimate for each node going from the end of the future path back to the root.

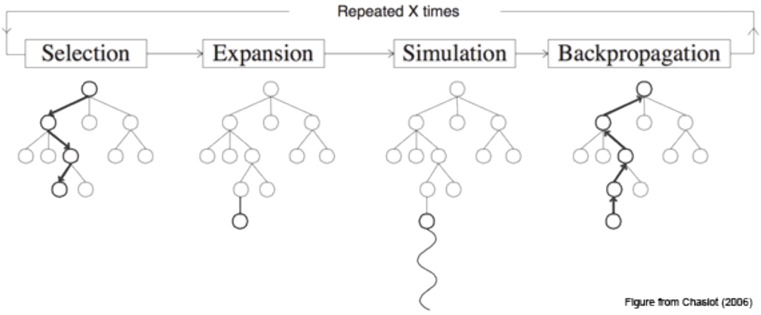

Figure 1: Sourced from http://www.cameronius.com/research/mcts/about/mcts-algorithm-1a.png

2.1 Initialization

We start with a search tree rooted at a single decision node containing $|\mathscr{A}|$ children, one for each possible action.

2.2 Selection / Expansion

Selection and expansion are combined in to a single step, since the selection is such that a non-leaf node may select as its child a node with visit count 0.

MC-AIXI traverses the search tree as follows. Assume rewards are bounded in the interval $[\alpha, \beta]$ and that m is the search horizon.

If the current node h is a decision node, the action picked is:

$$a_{Sel}(h) = \arg\max_{a \in \mathscr{A}} \begin{cases} \frac{1}{m(\beta-\alpha)} \hat{V}(ha) + C\sqrt{\frac{log(T(h))}{T(ha)}} & \text{if } T(ha) > 0 \\ \infty & otherwise \end{cases} \tag{3}$$

If there are multiple maximal actions, one is chosen at random. (I stole this hueristic from the MC-AIXI paper, I don't understand it) If the current node ha is a chance node, the observation-reward pair or is randomly picked according to the probability distribution given by $p(.|ha)$.

2.3 Expansion

Since the selection algorithm is such that a non-leaf node may select as its child node a node which has yet to be generated, the line between selection and expansion is blurry. We say a node is expanded if the node it picks has a visit count of zero.

2.4 Simulation

We now get to the Monte-Carlo bit of Monte Carlo Search Trees: Simulation.

The simulation is done as follows. We rollout until we reach the search horizon m to estimate the sum of future rewards. This rollout is done like so:

We have a rollout policy Π that picks an action at a decision node. Π by default selects actions randomly. Observation-reward pairs from chance nodes during rollout are generated as usual.

2.5 Backpropagation

After rollout, we have a path of nodes $n_1 n_2 \dots n_m$. Let n_k be the the first unvisited node in this path.

Let r_j be the reward of the node n_j. $r_j = 0$ if n_j is a chance node.

Then for each j from 1 to k, the estimated reward of n_j is updated as follows:

$$\hat{V}(n_j) \leftarrow \frac{T(n_j)}{T(n_j)+1}\hat{V}(n_j) + \frac{1}{T(n_j)+1}\sum_{i=j}^{m} r_i \tag{4}$$

Furthermore, the visit count of each node $n_1 n_2 \dots n_k$ is increased by 1.

3 Predicting

Our model class is all d-order Markov models. Markov models are represented using prediction suffix trees.

3.1 The KT Estimator

Let Y be a bitstring. Then

$$Pr_{KT}(Y_{n+1} \mid Y_{1:n}) = \frac{\#\text{ of 1s in } Y_{1:n} + 1/2}{n+1}$$

Pr_{KT} only depends on the number of 0s and 1s of Y. Thus, Pr_{KT} is invariant under permutations.

Theorem 3.1

$$Pr_{KT}(0^a 1^b) = \frac{\prod_{i=0}^{a-1}(1/2+i)\prod_{i=0}^{b-1}(1/2+i)}{(a+b)!} \tag{5}$$

3.2 Prediction Suffix Trees

Definition 3.2 A *prediction suffix tree* (M, θ) is a proper binary tree where each leaf node l is equipped with a probability distribution $\theta(l)$ over $\{0, 1\}$.

Notation: Let M be a binary tree. Let s be a bitstring. M_s is the node in M reached by the following algorithm:

```
set M_s = M.root

for c in s:
    If M_s is a leaf node:
        return M_s
    If c == 0
        M_s = M_s.right
    If c == 1
        M_s = M_s.left
endfor
return M_s
```

3.3 Calculating Probabilities With Prediction Suffix Trees

Let (M, θ) be a prediction suffix tree. Let d be the depth of M. Let s be a bitstring. Assume $|s| > d$. Let n satisfy $d < n < s$.
Then $Pr_M(s_{n+1}|s_{1:n})$ is calculated as follows:

1. Let $s' = \text{reverse}(s_{1:n})$.
2. Return $\theta(M_{s'})$.

3.4 Coding Proper Binary Trees

We assume we have an upper bound D for the proper binary tree M. We then code the tree by calling the function below with M.root. (The idea of this code is that we preform a pre-order traversal, writing down 1 for internal nodses, 0 for leaf nodes of length less than D, and nothing otherwise.)

```
function code(node,depth){
 if(node.leaf() == true){
   if(depth < D){
      message += 0;
   }
 } else {
   message += 1;
   code(node.left, depth+1);
   code(node.right, depth+1);
 }
}
```

```
message = [];
code(M.root, 0);
```

We denote the length of this code by $\Gamma_D(M)$.

3.5 Context Trees

Context Trees are a way of working with a mixture of all Prediction Suffix Trees of depth at most D.

Definition 3.3 A context tree of depth D is a complete binary tree of depth D where every node n (including non-leaf nodes) is equipped with a bitstring, denoted n.bitstring.

3.6 Updating Context Trees

We initially start with a context tree where each node is equipped with just the empty string. We set aside the first D bits gained from interacting with the environment to a variable h.

Then, given a new bit b, we update the bitstrings in the context tree M as follows: (Note that we only update the context trees using bits from observation-reward pairs).

```
node = M.root;
context = reverse(h);
i = 1;
while(node is not a leaf){
  node.bitstring += b;
  if(context[i] = 1){
    node = node.left
    i += 1;
  } else {
    node = node.right
    i += 1;
  }
}
h = h+b;
```

3.7 Calculating Probabilities using Context Trees

The function used to calculate the probability of the next bit of the sequence being 1 is given below. Here `prkt` is the KT estimator Pr_{KT}. Note that since we only use the KT estimator, it is not necessary to keep track of the whole bitstring. We just need to keep track of the number of 1s and 0s.

```
function wprob(node){
  if(node.leaf == true){
    return prkt(node.bitstring);
  } else {
    return 0.5*prkt(node.bitstring) + 0.5*wprob(node.left)*wprob(node.right);
  }
}
```

wprob($M.root$) gives a solomonoff-like probability estimate that the next bit in the sequence will be 1.

The equation is that after seeing $h = ax_{1:t-1}$, and deciding an action a_t,

$$\texttt{wprob}(root) = \sum_{M \in C_D} 2^{-\Gamma_D(M)} Pr_M(x_t \mid ax_{1:t-1}a_t) \tag{6}$$

where C_D is the set of all proper binary trees (which are turned in to prediction suffix trees by taking the KT estimator).

3.8 The Factored Action-Conditional Context Tree Weighting Algorithm

For this algorithm, we create l_x different context trees in order to incorporate type information.

1. Initialize $h = \epsilon, t = 1$. Create l_x context trees.
2. While $|h| < D$, pick action a_t randomly, observe x_t set $h = hax, t = t+1$.
3. Determine action a_t, set h $= ha_t$.
4. Receive x_t. For each bit $x_t[i]$ of x_t, update the ith context tree with bit $x_t[i]$ using history $hx[1, i-1]$ and recompute $wprob(root_i)$.
5. Set $x = hx_t, t = t + 1$. Goto 3.

4 Pseudocode for MC-AIXI

References and Further Reading

References

[1] Joel Veness, Kee Siong Ng, Marcus Hutter, and David Silver. A monte-carlo aixi approximation, 2009.