The Tolman Mechanism

Development Notes

Stefano Ghirlanda

December 7, 2020

1 Rationale

- The Tolman mechanism implements a learning mechanism that formalizes common theorizing in animal cognition.
- The animal is assumed to build a mental model of its environment.
- The model is then used to plan how to reach states with high value.
- There are major computational challenges in such a mechanism and it is unlikely that animals (or even people) can apply this strategy in cases beyond a certain complexity.
- We restrict the mechanism to work in worlds with a single goal, and that "restart" after the goal is obtained.

2 Variables

- The Tolman mechanism learns $S \to B \to S'$ transition probabilities.
- These estimates are called $z(S, B, S')$.
- Everything else is decided when responding.

3 Learning

- There is a single learning rate, α_z.

- Imagine first that all stimuli are made of a single element. When $S \rightarrow\ B \rightarrow S'$ is observed, $z(S,B,X)$ is updated for all X as follows:

$$\Delta z(S,B,X) = \alpha_z \left(\lambda_X - z(S,B,X)\right) \tag{1}$$

  where $\lambda_X = 1$ when $X = S'$ (the state that actually occurred), and 0 for all other states.

- With sufficient experience and sufficiently small α_z, eq. (3) will converge to the actual transition probabilities.

- How to extend to stimuli with more elements? Let's first consider that all intensities are $= 1$. We can use

$$\forall S'_j: \quad z(S,B,S'_j) = \sum_{i=1}^{n} z(S_i,B,S'_j) \tag{2}$$

  where the S_i are the elements of S and the S'_j the elements of S'. According to eq. (2), different elements of S compete for predicting each element of S' in the same way as they would compete for accruing v or w values in A-learning (and other mechanisms).

- Using eq. (2) to calculate $z(S_i \ldots S_n, B, S'_i)$ we can update each S_i as follows:

$$\Delta z(S_i,B,X) = \alpha_z \left(\lambda_X - z(S,B,S'_j)\right) \tag{3}$$

  where $\lambda = 1$ if $X \in S'$ and $\lambda = 0$ otherwise.

- With intensities written as x we can use:

$$\forall S'_j: \quad z(S,B,S'_j) = \sum_{i=1}^{n} z(S_i,B,S'_j)x_i \tag{4}$$

$$\Delta z(S_i,B,S'_j) = \alpha_z \left(\lambda_X - z(S,B,S'_j)\right) x_i x'_j \tag{5}$$

3.1 Trial implementation *<2020-12-07 Mon>*

- The equations above have been implemented in the `github` branch `tolman-mechanism`

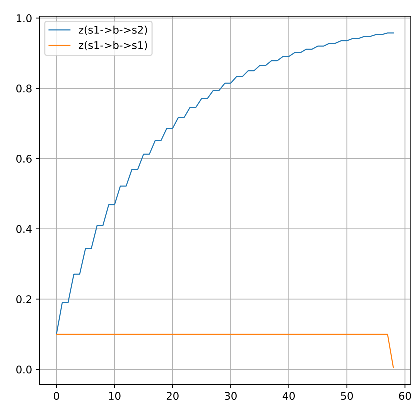
- I have made a super simple test where there is only behavior B and two stimuli S_1 and S_2), and the only thing that happens is the sequence:

$$\cdots \rightarrow S_1 \rightarrow B \rightarrow S_2 \rightarrow B \rightarrow S_1 \rightarrow \cdots \tag{6}$$

- In this case, the Tolman mechanism should learn:

$$z(S_1, B, S_2) = 1 \tag{7}$$
$$z(S_1, B, S_1) = 0 \tag{8}$$
$$z(S_2, B, S_1) = 1 \tag{9}$$
$$z(S_2, B, S_2) = 0 \tag{10}$$

- Printing this values throughout the simulation shows that they are in fact learned, but the graphs do not come out well. I think this is because the current plotting code only updates the plot for stimulus elements that are actually presented, but the Tolman mechanism also updates $z(S, B, S')$ for S' elements that are not present. For example, this is a graph I get:



- The initial values of all z is 0.1 for testing purposes. The blue line goes to 1 as it should. The orange is plotted at the initial 0.1 throughout the simulation, but for the last point, which is the correct value of 0. As I mentioned above, however, if I print the values during learning I do see that they are decreasing toward 0 all the time.

- (Element intensities and multiple elements are implemented but not tested yet.)

4 Responding

- Responding assumes that the world model in z is true.

- Responding should then make the "best plan" given this knowledge.
- Evaluating all possible plans is tricky in general. Let's work with a particular kind of world to begin with:
  1. There is only one valued stimulus, called the goal.
  2. When the goal is reached, the next state is determined by the world using a fixed rule (deterministic or stochastic).
- The second condition means that we need to consider just how to reach the goal once, since after that the world "resets" to a statistically equivalent state.
- The first step is to find all paths to the goal. We start from the goal and we go backwards along possible transitions, that is, all transitions with $z > 0$ for some B. If we eventually reach the current state, we store the sequence of transitions and call it a "path."
- We calculate the expected *rate* of return for all paths to the goal, that is the expected value divided the path length (number of actions):

$$v(\text{path}) = \frac{u(S_{\text{goal}}) - \sum_{B \in \text{path}} c(B)}{l_{\text{path}}} \prod_{S' \in \text{path}} \Pr(S \to B \to S') \qquad (11)$$

- In eq. (11), the sum is over all behaviors in the path, l_{path} is path length, and the product is over all transition probabilities in the path. (This is the probability that the plan will succeed.)
- We then give a value to each behavior that is feasible in response to the current stimulus. If the behavior is the first step on a path to the goal, its value is the v value of the goal. If it is not, its value is `start_v`.
- If a behavior is the first on more than one path, we can average the v's using the success probability of each path, so we should store these somewhere.
- We then use softmax to choose a behavior.