
Ratbert: Nearest Sequence Memory Based Prediction Model Applied to Robot Navigation

Sergey Alexandrov

SERGE1@EDEN.RUTGERS.EDU

Department of Computer Science, Rutgers University, New Brunswick, NJ 08903 USA

Abstract

In this paper I examine the effectiveness of NSMP, a predictive algorithm based on Nearest Sequence Memory, in a robot navigational task. Learning an unknown environment typically involves successfully modeling the POMDP describing it. NSMP uses the Nearest Sequence Memory method to predict the consequence of a given action based on the set of instances that comprise its previous experience. I evaluate the performance of NSMP and examine the results and issues raised by this experiment.

1. Introduction

A fundamental problem in robot navigation in an unknown environment is that not all state can be perceived at any given time. In a simplified case, the sensors can only be relied on to give the state of the immediate surroundings – the rest of the state (coordinates in the environment, the actual map) is hidden. And since many different locations can be perceptually aliased (the immediate sensory data is identical), learning the environment requires learning a Partially Observable Markov Decision Process (POMDP).

McCallum (1994) had suggested a set of instance-based learning algorithms for learning POMDPs – algorithms that use previous experience data to extrapolate the current state to the desired goal. One of these was Nearest Sequence Memory (NMS); as the name suggests, NSM uses matches in previous (already experienced) state sequences to learn optimal policies. This can be readily applied to a goal-finding robot navigational task.

In this paper, we look at the approach NSM takes as a way to learn a model in terms of correctly predicting the immediate consequences of any given action. Specifically, we want to know how well such an approach can predict the next set of percept given existing experience and an arbitrary (but feasible) action. Because this is not a goal-finding task, reward propagation is not involved, thus here we will not examine the Q-value calculation in NSM; instead, we concentrate on its ability to successfully use learned data.

2. NSMP – Nearest Sequence Memory Predictor

To measure how well the model has been learned, we test the algorithm's ability to predict the next observation given previous experience and a selected action. To do this, we use NSMP – a basic algorithm based on Nearest Sequence Memory. First, the environment is explored via a random walk, and the experience is saved as training data. Then, another random walk is taken, and the consequence of each action is predicted using NSMP.

2.1 Algorithm Details

We will denote both the training data and the existing experience as sequences of states s_i : T (training sequence) and S (testing sequence). We define the state s_i as the pair (a_i, o_i) where a_i is the action taken to arrive at s_i , and o_i is the observation (percept) at s_i . Thus, each sequence looks like this: $\{(a_1, o_1), (a_2, o_2), (a_3, o_3), \dots, (a_n, o_n)\}$. Generally, instances of both actions and observations belong to finite sets: $a_i \in \{a^1, a^2, \dots, a^n\}$ and $o_i \in \{o^1, o^2, \dots, o^m\}$.

First, the training sequence T is generated. This simply involves saving the information (actions and subsequent observations) as a sequence of states. Then, for each step in the test phase, the following steps occur:

Let s_i denote the current state, and a_{i+1} be the action chosen at this state. As in NSM, we evaluate the neighborhood metric for each s_j in T :

$$metric(s_j) = \begin{cases} neighbor(s_{j-1}, s_i) & \text{if } a_j = a_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

Where the neighbor function is defined as:

$$neighbor(s_j, s_k) = \begin{cases} 1 + neighbor(s_{j-1}, s_{k-1}) & \text{if } (a_j = a_k) \wedge (o_j = o_k) \\ 0 & \text{otherwise} \end{cases}$$

Again, as in NSM, we then select the k nearest neighbors – the k states with the highest neighborhood metric – we'll call this *NNS* – Nearest Neighbor Set. We can now

assign a prediction value to each observation occurring in the neighboring states. I chose to do this by weighing the average of the metrics of all neighbors in *NNS* containing the observation by the percentage of such neighbors among all neighbors in *NNS*:

$$weight(o^j) = \frac{count(s_k | s_k \in NNS, o_k = o^j)}{k} \times avg(metric(s_k))_{s_k \in NNS, o_k = o^j}$$

This establishes the tradeoff between match lengths and the frequency of matches. For example, let $k = 4$, $NNS = \{s_1=(a^1, o^1), s_2=(a^1, o^2), s_3=(a^1, o^2), s_4=(a^1, o^2)\}$, where $metric(s_1)=4$, $metric(s_2)=1$, $metric(s_3)=1$, $metric(s_4)=1$. Thus $weight(o^1) = 1$, and $weight(o^2) = .75$. The predicted observation is simply the observation with the greatest weight:

$$prediction = \arg \max_o (weight(o))$$

In the example above, o^1 is the predicted observation. If a probabilistic prediction is desired rather than a deterministic one, the next section derives the probability calculation.

3. Evaluation of Accuracy

There are several ways of determining the accuracy of the prediction model. The obvious way is to calculate the successful prediction rate:

$$\frac{\#of\ correct\ predictions + \frac{\#of\ guesses}{\|o\|}}{\#of\ prediction\ requests}$$

We include the guess factor for instances for which the algorithm is unable to generate a prediction (insufficient data). Other prediction accuracy measures have been developed. One such example is the cross-entropy measure (Wong 1999):

$$\sum_{i=1}^n -\log(P(s_i | s_1, s_2, \dots, s_{i-1}, M))$$

where M is the model being evaluated, and s_i are the states in the test sequence. This measure was originally developed to evaluate bigram models. A bigram model predicts the next state by selecting the two-state sequence in the training data with the highest probability such that the first state matches the current state when the prediction is made. A bigram is part of a more general set of probabilistic predictive models called n -grams, where n is the length of the sequence match considered. In fact NSMP can be looked at as a variable- n -gram model, since the basic mechanism also involves sequence matching, but the sequence match length in NSMP is variable.

To use the cross-entropy measure, we need to define how to measure $P(s_i | s_1, s_2, \dots, s_{i-1}, M)$. We do so in the following manner:

$$P(s_i | s_1, s_2, \dots, s_{i-1}, NSMP) = \frac{weight(o_i)}{\sum_{NNS} weight(o^k)}$$

which is basically a normalization of the observation weights in *NNS*.

To test NSMP's performance, we'll generate both the prediction rate, and the cross-entropy measure. We'll compare it to generic bigram performance (which works surprisingly well given the restricted space).

Also note that to generate cross-entropy, smoothing must be performed, since the model may give probabilities of 0 for some state pairs, resulting in an increment of infinity ($-\log(0) = 8$). Here we use the most basic approach – additive smoothing. This modifies the calculation by adding $1/n$, where n is the training sequence length, to instances of probabilities of 0 – in other words, we replace the zero probability with the least possible probability.

4. Experiments

4.1 Ratbert

Because we are specifically interested in NSMP's performance with realistic data, the first test bed was a configurable maze navigated by Ratbert (Figure 1). Ratbert is a basic robot built using the Lego Mindstorms platform. It's equipped with a basic sensor array, allowing it to sense whether there is a passage in front of it, on the left, or the right, as well as a (noisy) estimate of distance traveled. It's capable of 90° turns and 180° u-turns (this is necessary when a dead end is reached).

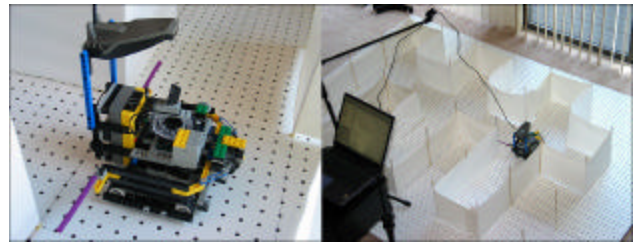


Figure 1. Ratbert and its maze

4.1.1 DEALING WITH REAL DATA

While the environment is in a sense a grid world, we are interested in getting away from the notion of pure grid worlds, as the concept is unrealistic when interacting with a physical robot. For example, Ratbert's odometric information is too noisy to properly tell when a "gridline" is crossed while traversing a straight hallway. For this reason, observations are taken at intersections – since there are no decisions to be taken in the middle of a hallway (the robot must continue to go straight). However, to distinguish traveling one "square" from two

“squares” (or in more realistic terms, since things are rarely separated by “squares”, simply distinguishing different distances traveled), the percept at each state consists of not just the three binary wall indicators (left, right, front), but the distance traveled since the last state. This data is not exact; the robot does not always travel in a straight line (it may bump into a wall and straighten itself out¹). We could ignore the distance, but it can help to distinguish otherwise aliased states (the use of noisy distance data was inspired by Shatkay’s (1997) utilization of odometric data to improve the Baum-Welch performance in POMDP model-learning).

This implies that we need to consider the observation comparison from the previous section: what does $o_j = o_k$ mean if data is inexact? Assuming that observation o_i is a vector of individual sensor readings $(r_1(t), r_2(t), \dots, r_m(t))$, then for any sensor r_j that is expected to have inexact data, we can generate a distribution of values over the training set T . To obtain a finite $\|o\|$, this data must be split into discrete bins based on this distribution. Then, we can assume for the purposes of observation comparison that $r_j(i) = r_j(k)$ when both values fall into the same bin. Splitting the data into uniform width bins would rarely simulate realistic experience, so histogram approaches don’t apply. If some prior information is known, k -means clustering could be used. (Shatkay 1997). However, having no prior about the environment, we must resort to some prior about the robot itself. Performance tests with Ratbert yielded an error rate of approximately ± 8 cm per 33 cm traveled. Thus for the purposes of this paper we let the distance data of two observations be equal for the purposes of NSMP if they fall within each other’s error range (this can be looked at as sensor calibration). This however can only be a temporary solution, as a better approach needs to be found. For example, perhaps it is to apply hierarchical clustering until cluster distance is within the sensor’s error.

4.2 Software Simulation

Given the limitations of the physical model (size, time and effort needed to perform a trial), we also used a software simulation based on Ratbert’s performance (so for example the odometric noise was simulated according to the error rate in the previous section). This trial was run using a different maze, *CubicleLand2*, (Figure 2) to evaluate the performance on a larger environment.

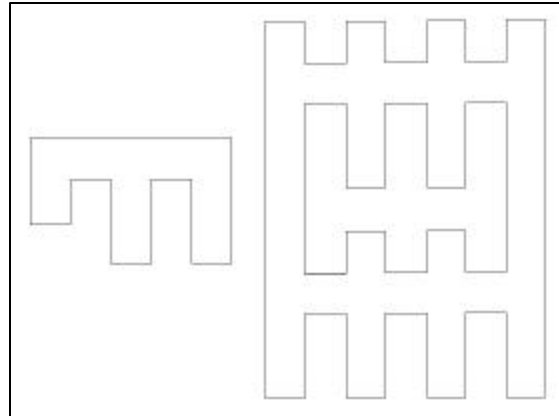


Figure 2. *CubicleLand1* (left) and *CubicleLand2* (right)

5. Results

The first test run was performed by Ratbert on the *CubicleLand1* environment. Four walks of 50 steps were performed, allowing 6 trials (6 ways of choosing a pair among the 4 sequences). Both NSMP, with $k=4$, and the bigram prediction algorithm were executed for training sequences of lengths 0 to 50. Figure 3 shows the prediction rates and cross-entropies as functions of testing sequence length.

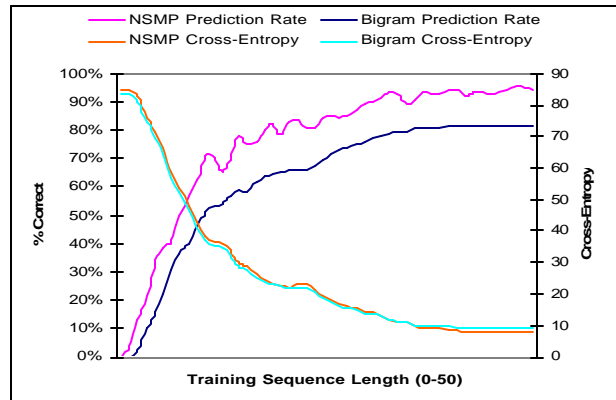


Figure 3. Ratbert’s performance averages on *CubicleLand1*

This test shows NSMP performing somewhat better than bigrams. The limitation of bigrams is shown in the convergence of the bigram prediction rate before an optimal value (the only reason NSMP does not converge to 100% is that typically the first or second prediction may be false due to the insignificant testing sequence length). Thus NSMP can accurately learn the *CubicleLand1* model, while bigrams can do so only fairly accurately – here we see evidence of a need for a better memory than that encoded by bigrams. Interestingly, the cross-entropies of the two approaches follow each other closely.

¹ Part of Ratbert’s control code takes care of straightening it out while traveling inside hallways (since actuation of 90° turns is not exact), leaving the learning only to intersections. Realistically, this too should be considered a learning task. Research has been done in the area of such multi-task learning, such as hierarchical approaches - one such approach is the NSM-based HSM (Hernandez-Gardiol, Mahadevan 2000).

The next test run was performed on the software simulation based on Ratbert, using the *CubicleLand2* environment. 10 trials were executed using different data sequences each time, for training sequence lengths of 5 to 200, and a testing sequence length of 100 states. Again, NSMP with $k=4$ and the bigram prediction algorithm were used. The results of this test are shown in Figure 4.

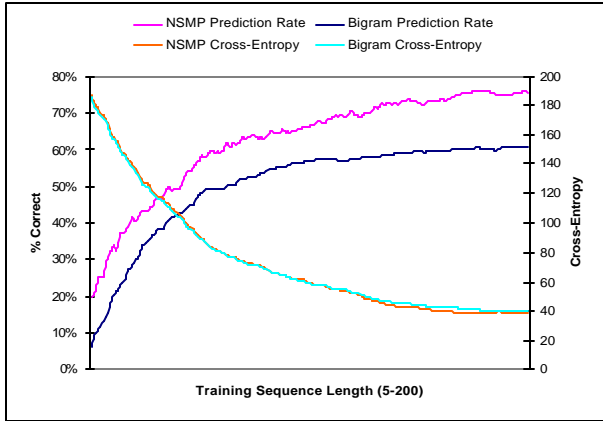


Figure 4. Software simulation results for *CubicleLand2*

Once again, NSMP shows consistently better performance, though even with a 200 step training sequence it only achieves an average of about 75% accuracy. Both approaches show significant leveling off, indicating that additional work must be done to allow a more precise predictor.

6. Analysis and Further Work

The tests show that NSMP can be a reasonable method to anticipate consequences of actions in a POMDP, given enough training data, and its accuracy is consistently better than that of bigrams. (In simulation runs using different values of k and modifications of *CubicleLand2*, to decrease the number of aliased states, NSMP produced a consistently better prediction rate, although not overwhelmingly so). The application in robotic navigation is natural – once reasonably accurate predictions can be made, we can generate a probably optimal sequence of actions to reach any desired state that has been experienced. Adding rewards to the states, we can instead look for the optimal path in terms of reward – which in effect is what is accomplished in the NSM algorithm through its dynamic programming component.

How can NSMP be improved? Throughout this paper, the training sequence was generated by a random walk. Such undirected exploration is unlikely to be the optimal training method – instead techniques such as counter or recency-based exploration (Thrun, 1992) should be used. Since the utility of any exploration technique is dependent on the learning method, picking the optimal one for NSMP may be a matter of experimentation. Furthermore, other instance-based methods, like McCallum’s USM and

U-Tree (1994), can be used to derive similar probabilistic prediction models. The performance can also be compared to that of a POMDP/HMM learner (McCallum showed that Baum-Welch, for example, required longer test sequences for comparable goal-finding performance in comparison to NSM).

Finally, a key need in the utilization of instance-based learning methods, which require instance comparisons, in robotics is the ability to successfully differentiate noisy sensor data, as mentioned in Section 4.1.1. This is a necessary step in moving away from grid worlds and towards the ability to explore real-world POMDPs.

7. Conclusion

I demonstrated the benefit of NSMP as a predictive algorithm in a sequential hidden state environment, such as robotic navigation, when compared to the bigram method. NSMP demonstrates that variable length matches in the memory can improve the probability of correct predictions. The experiment also shows there are limitations to NSMP’s performance, and more work needs to be done to approach near-perfect prediction rates.

Acknowledgements

This paper would have been impossible without the help of Professor Michael Littman (Rutgers University). Ratbert was programmed using the open-source leJOS platform (lejos.sourceforge.net), a task made much easier by virtue of Brian Bagnall’s *Core Lego Mindstorms Programming*. Finally, I’d like to thank Scott Adams for having no idea that a character created, wholly-owned, and trademarked by him has been shamelessly stolen to name my robot.

References

- Hernandez-Gardiol, H. and Mahadevan, S. *Hierarchical Memory-Based Reinforcement Learning*. Proceedings of Neural Information Processing Systems, 2001.
- McCallum, R. A. (1994). *Reinforcement Learning with Selective Perception and Hidden State*. Doctoral Thesis, University of Rochester, Rochester, New York.
- Shatkay, H. (1998). *Learning Models for Robot Navigation*. Doctoral Thesis, University, Providence, Rhode Island.
- Thrun, S. B. (1992). *Efficient Exploration in Enforcement Learning*. Technical Report CMU-CS-92-102, Carnegie Mellon University, Pittsburgh, Pennsylvania.
- Wong, B. (1999). *Bigram Model Generalization Using Singular Value Decomposition*. Masters Thesis, Department of Computer Science, Victoria University of Wellington.