A Bayesian Approach to solving Partially Observable Dynamic Environments

Sai Praveen B(CS13B025), J S Suhas(CS13B056)

May 2, 2016

Abstract

We take the class of problems where the environment E for every episode is sampled from a set of enviroments $\mathcal{E}$ according to a PDF $P(E)$. In this case, simple TD-learning using history methods tends to learn a sub-optimal approach to solving the problem(sometimes no stable solution common to all $E \in \mathcal{E}$ exists. In this case, the results can fluctutate.) and can take a long time to converge. We provide a framework using generative models to attempt to model the distribution of environments rather than assume that the environment is stochastic and attempt to solve it normally.

1 Problem Definition

1.1 Abstract Problem

We will now define the problem of Partially Observable Stochastic Enviroments. We have a set of environments $\mathcal{E}$ (which can be infinite). Our assumption is that for every episode K_n, we have a new environment E_n sampled according to $Pr(\mathcal{E})$. Note that we make the assumption that E_n is independent of $E_m \forall m < n$. However, the methods provided may be extended to cover markov distributions of the type $Pr(E_n \| E_{n-1})$ as well. We also have an agent A that receives a partial 'kernel' of information about the environment E_n. Let this be the observation at any time step t, $Pr(o|S_t, E_n)$ for the episode n using environment E_n. As usual, we have a transition and reward probability $Pr(R_t, S_t|S_{t-1}, A_{t-1}, E_n)$ but it also depends on the environment currently being observed. Finally we have a set of observations $o \in \Omega$.

1.2 Our Problem Definition

We will now attempt to solve a simple case of a PODE that we defined to illustrate the usefulness of modelling the environment distribution $Pr(\mathcal{E})$ as opposed to marginalizing over it(Like standard TD-methods that ignore the changing environments). We take the case of the problem with an $N \times N$ grid

world with walls/obstacles indicated by 1s and free space by 0s. The agent gets a -1 reward for bumping into walls(including the edge of the grid world and a -1 reward per time step. The agent has a +50 reward for reaching the target T which is always the end of the world $(N-1,\, N-1)$. The agent always starts at $(0, 0)$ although this can be changed quite easily.
The agent can see a box of size $E \times E$ around it's position after every step. The actions and transitions are currently deterministic given the environment E_n but all the methods used to solve it can be directly extended to provide for stochasticity in movement as well.
We also make the assumtion that the agent has the complete model of the actions and rewards in the environment i.e it already knows that moving UP increases it's Y coordinate by 1. It does not, however , know which states are valid or what the configuration of the environment is. This assumption can be removed by introducing a learning unit for models similar to DynaQ. But for the purposes of the solution we propose, we assume the model is known.
For the gridworld distribution $Pr(\mathcal{E})$ we consider 2 different definitions and solve for both:

1.2.1 Single Wall World

We have 4 different possible locations for the walls(as shown). We randomly sample from these 4 configurations.

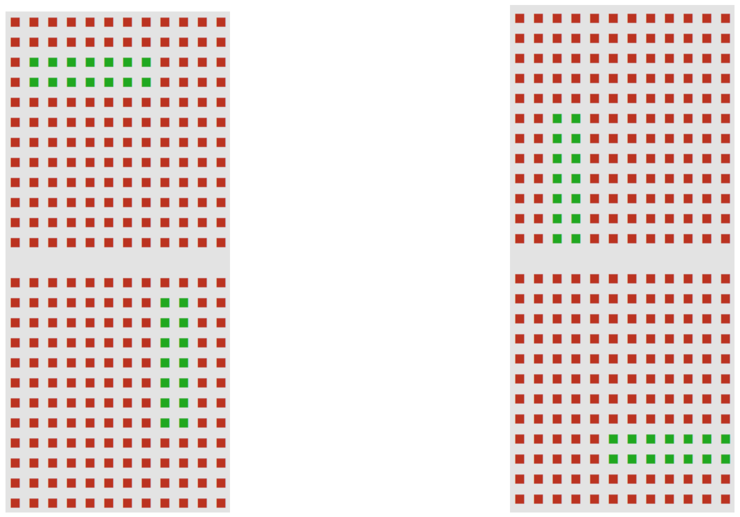

Figure 1: 4 elements of $\mathcal{E}_{SW}$

1.2.2 Multiple Wall World

For a slightly more complicated task and one which will require the system to generalize and not overfit the data is random combinations of the 4 configurations discussed above: This leads to a whole lot more combinations including some configurations where it can get really difficult to solve properly.

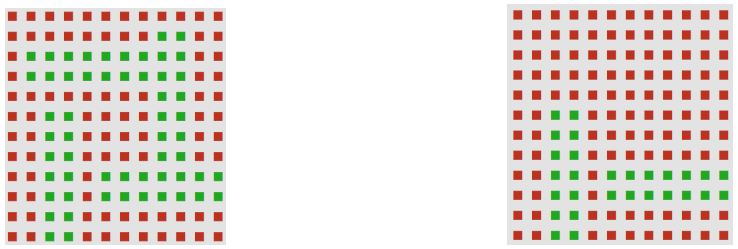

Figure 2: 4 elements of $\mathcal{E}_{SW}$

1.2.3 L-World

The above example environment set is to test the generative model's ability to separate out the components and thus demonstrate that, in the general case, using a model like an RBM is far better than attempting to remeber all the environments(which is also a valid method in the above case).

To clearly demonstrate the power of our framework over a naive implementation(no model) and a Q learning technique that marginalizes over the distribution of $\mathcal{E}$, we introduce another simple yet powwerful world called the L-world. The Q-learning technique would converge to an optimal solution in the above wall world despite the dynamic environment. It would however be slightly unstable.

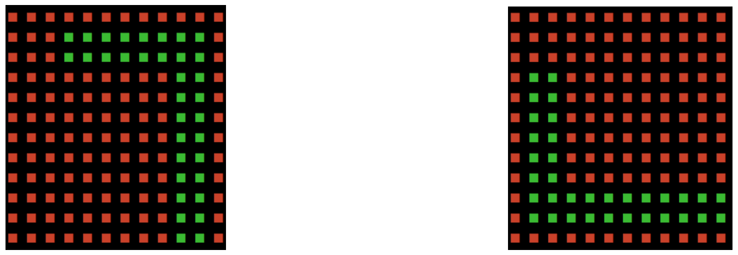

Figure 3: The elements of $\mathcal{E}_{LW}$

2 Approach

The approach uses the following framework as an abstract system that can be adapted to learn from any environment. This model closely follows the system used for DynaQ, but expands upon it to support stochastic episodes.

2.1 Framework

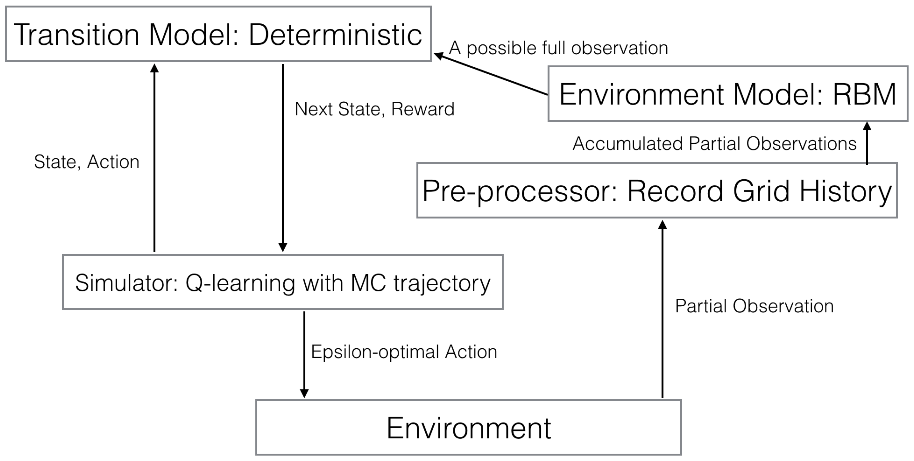
Transition Model: Deterministic
A possible full observation
Environment Model: RBM
Next State, Reward
State, Action
Accumulated Partial Observations
Pre-processor: Record Grid History
Simulator: Q-learning with MC trajectory
Partial Observation
Epsilon-optimal Action
Environment

Figure 4: Framework

In the above figure the environment model(necessarily a generative model) also learns form the accumulated partial observations by treating it as experience.

2.2 Modelling: Modified RBMs

We used the standard RBMs as an appropriate method of modelling the various worlds. We make a minor modification to the RBM Contrastive Divergence($[GH]$) training algorithm to allow the RBM to learn from partially visible data. Implementations of RBMs in C++ for partial data were not readily available. Thus, a full C++ implementation for RBMs was written which implements CD_Nlearning. The standard contrastive divergence equation is:

$$\Delta w = \alpha_w.(< h.v >_P - < h.v >_Q)$$

where P is the sample distribution and Q is the approximating distribution. The<>operator is the outer product operator.h is the binary hidden node vector and v is the binary visible vector.
We make a simple modification to allow for training with partial visibility:

$$\Delta w = \alpha_w. < I.m_v > .(< h.v >_P - < h.v >_Q)$$

where $m_v$is the visibility mask(binary vector with 1s if a visible part is observed and 0 if it's not). I is a vector with all 1s.

We also extended the CD equation to seamlessly support multiple states instead of just 1s and 0s. The update equation becomes slightly more complicated:

$$\Delta w_{ijk} = \alpha\big[m_v(j)\big]\big[I(h_i^{(o)} = k).I(v_j^{(o)} = k) - I(h_i^{(s)} = k).I(v_j^{(s)} = k)\big]$$

-

i and j are the indices of hidden and visible vectors respectively.
For sampling the best assignment to the unobserved visible units given the state of the remaining units, we first find a MAP estimate for the hidden layer and then a MAP estimate for the visible layer.
To find the MAP estimate of the hidden layer from a paritally observed visible layer, we randomly sample the unobserved nodes using their $bias\beta_v$ of the visible layer. Note that we should not assume 0 weight from those nodes as it assumes the nodes are turned OFF rather than unobserved. By random sampling, the effect over all the hidden layer nodes by the unobserved nodes is an expected 0.

2.3 Modelling: Convolutional DBNs

Regular RBMs are good at observing patterns and even manages to break down & recombine patterns to build a general representation. But they are limited in the sense that they do not take advantage of the spatial correlation between nodes in the gridworld. The solution to these shortcomings is the lesser known CDBN[LGA] (Again, good implementations are hard to come by, so a modified version of this was coded in C++) which uses convolutional feature maps and probabilistic max-pooling layers to take advantage of higher level patterns. The CDBNs can also be stacked on top of each other(without having to propogate error. Unlike deep NNs, a belief network can be trained layer by layer) to form a deep learning structure. The major feature of the CDBN is it's probabilistic max-pooling which divides the hidden layer H feature maps(H_k) into blocks of size $A \times A$ and each of the blocks are pooled into a single node in the pooling layer P_k. Each node in P_k takes $A^2 + 1$states each of which correspond to each of the A^2nodes in the block(Only one of which may be active). To enforce this, we clumped the entire block into a single node P_k(pooling node) and eliminated the hidden layer H. Each node inP_k now has $A^2 \times K^2$connections to $(A+1) \times (A+1)$ nodes in the visible layer V. Each of the K^2connections forms the weight for one state of P_k. We finally use softmax sampling to determine the value of P_k.
The code for the CDBN is complete, but the convergence is highly unstable right now. We need to add a regularisation term as well to prevent overfitting which wasn't a big issue with RBMs. As such we do not yet have results for agents using CDBNs.

2.4 Planning: Q-learning using random trajectories

We have 4 actions that can be taken(Up, Down, Left & Right). Once we have a most probable estimate $\hat{E}$ for the environment, we can now run a planning algorithm with this environment and the deterministic model. Since our model is partially observed, the state varies very frequently, we do not record action or state values once a move has been taken. This is because we don't know the rest of the state and(in general) will have to discard the state values once a new part of the grid is explored. There are ways to split the value function and update the states based on which parts have changed, but we have not currently explored those extensions.

Currently, we have used 2 modes of Monte Carlo trajectory generation. One with uniform policy $\pi(s,a) = 0.25$ and one with policy $\pi(s,a) \neq 0$if $T_M(s,a) = Empty$(Uniform probability over all states that do not lead to a wall) The 2nd method gave much better results but slightly slower times.

From the generated set of trajectories $\mathcal{T}$, we use Q-learning to learn action values for every cell in the grid. The trails are generatied from the current position of the agent in the grid and thus the Q values closest to the agent converge the most(this is not always the case but when the agent and target are close, other states far away are generally badly approximated.)

$\gamma = 0.9$ for all instances in which this planning algorithm is used

The TD update is as follows: for every $< S_t, A_t, R_{t+1}, S_{t+1} >\in \mathcal{T}$,

$\Delta Q(S_t, A_t) = \alpha.\big(R_{t+1} + \gamma.max_a Q(a, S_{t+1}) - Q(S_t, A_t)\big)$

2.5 Pre-trained vs Online training

Depending on the problem situation, we may be able to pre-train the Environment Model $\hat{Pr}(E)$ with samples from $\mathcal{E}$. For this problem we analyse the performance of the system in both cases. In the pre-trained case, note that since no learning occurs, we do not have any change in expected reward with respect to episode number.

In On-line learning, we train the RBM once the episode is completed, with whatever portion of the grid is explored for the episode.

3 Results and Samples

3.1 Pre-trained

We present both intuitive examples and numerical results from when the agent is solving the environment. For an RBM pre-trained with 500 samples, we have the following examples of the RBM in action for environment $\mathcal{E}_{MW}$:

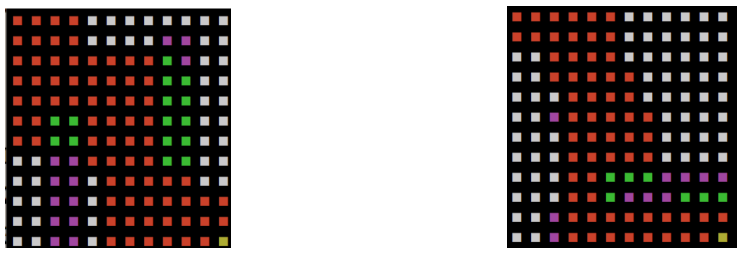

Figure 5: Examples of the agent navigating the environment with a 5×5 visibility kernel.

Note that the RED square is explored area, WHITE square is unobserved area, GREEN is an observed wall and PURPLE is an inferred(sampled from RBM).

We see that when the agent moves from the top-left $(0, 0)$ to bottom-right $(W - 1, H - 1)$, the agent on the left discovers the green squares on the left wall, infers the rest of it and quickly leaves the area for a different route. Thus, modelling the environment can be really helpful.

We also now see the pre trained model navigating the L-world and the clear advantage it has over a model-less system:

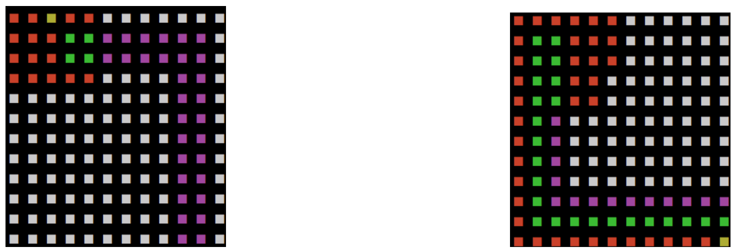

Figure 6: Examples of the agent navigating the environment with a 5×5 visibility kernel.

As soon as the top 4 blocks are observed, the rest of the wall is immediately inferred and the system can now navigate around it as shown by Fig 6(b).

3.2 Online Training

During online training, we have more interesting samples, these are for RBM pre-trained with 0 samples and trained for 500 episodes. The agent does not learn what's not necessary. The following are examples of the agent in action after 500 episodes of training.

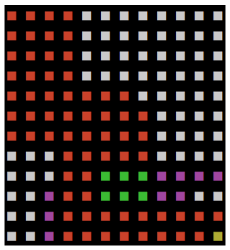
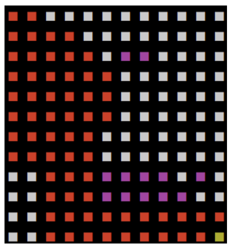

Figure 7: Examples of the agent navigating the environment with a 3×3 visibility kernel.

The learning is somewhat incomplete because of lack of fully visible training sets. The agent on the right only approximately figures out the lower wall, but it is enough for the simulation algorithm to decide to not take that path.

3.3 Variation with critical parameters

We analyse the variation in convergence times for vibibility span size K. For $K = 2$, the kernel is $5 \times 5$and for $K = 3$ the kernel is 7×7. We observe that for both K, they converge to the optimal policy but for higher K, naturally, the convergence is faster. The plots are averaged over very few trials and are thus very noisy, but near around $\#e = 100$, we can find $K = 3$ performing much better than $K = 2$.

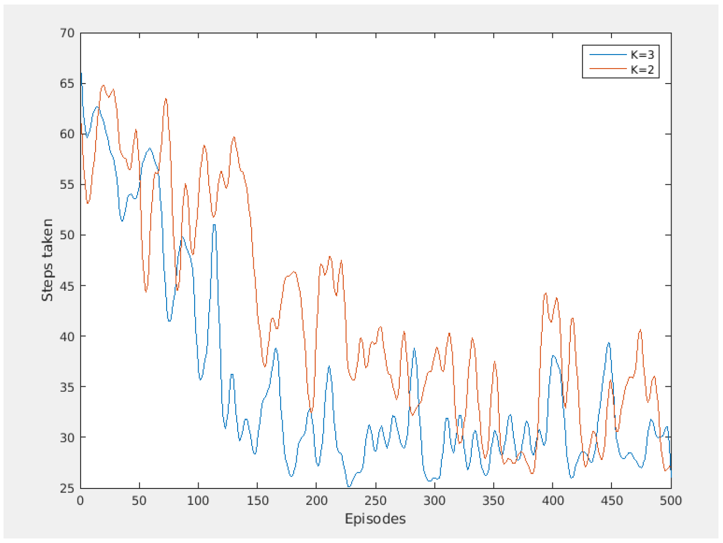

Figure 8: Variation with Visibility Span size

One of the most critical paramers in deciding the next action to take during simulation is the one that handles the behaviour of the inferred wall segments.

We must make an allowance for the fact that the wall may not be real. If we assume it's real then we may end up with an unsolvable problem even though in fact, it is solvable. For this, we allow movement through inferred wall segments in the planning algorithm, but we penalize it heavily. The system is quite sensitive to this parameter and for small penalties, the system may act sub-optimally. We see that for $N = -10$ and $N = -100$ it does not converge, but for $N = -250$, it finally reduces the expected steps taken.

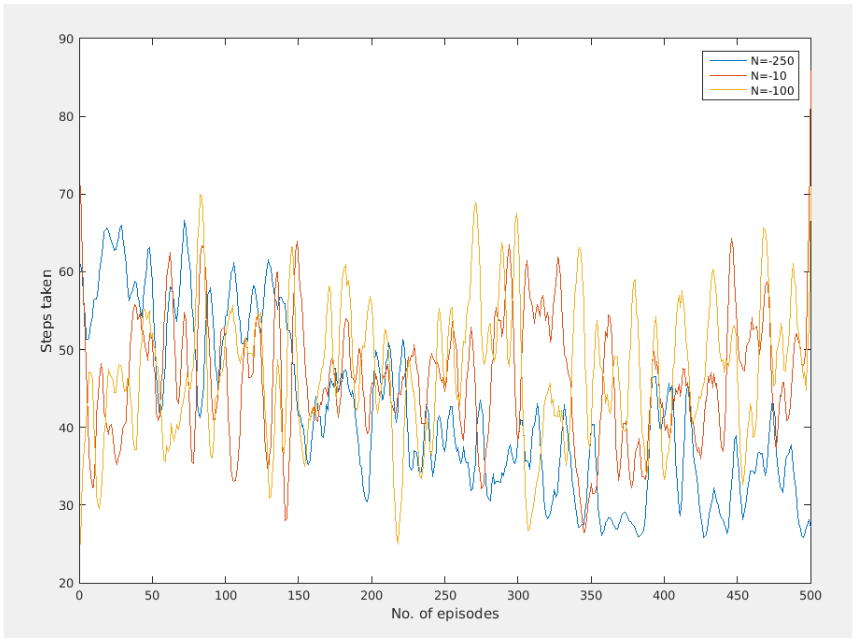

Figure 9: Variation with Inferred Wall penalty

3.4 Comparision with Q-learning using random trajectories & storing history.

The plot of Model-less learning vs RBM-based learning on the L-world is as follows:

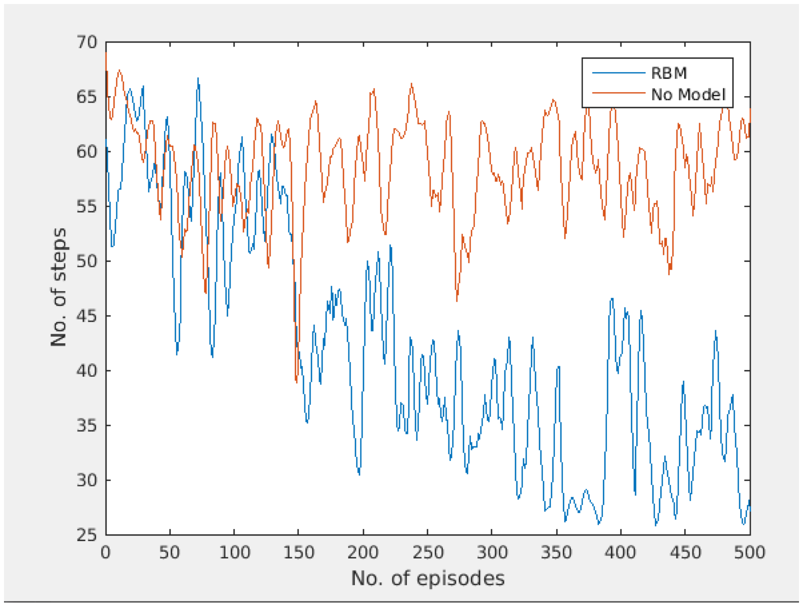

Figure 10: Comparision of the 2 methods on$\mathcal{E}_{LW}$

4 References

1. [$GH10$] Geoffrey Hinton; 2010 "Practical Guide to RBM Training" https://www.cs.toronto.edu/~hinton/a

2. [GH] Geoffrey Hinton; "On Contrastive Divergence Learning" http://www.cs.toronto.edu/~fritz/absps/cd

3. [LGA] Lee, Honglak; Grosse, Ranganath; Andrew Ng. 2009 "Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations"