Deep Belief Nets as Function Approximators for Reinforcement Learning

Farnaz Abtahi and Ian Fasel
Department of Computer Science
School of Information: Science, Technology, and Arts
The University of Arizona
Tucson, AZ 85721-0077
Email: {farnaza,ianfasel}@cs.arizona.edu

I. Introduction

Real-world tasks often require learning methods to deal with continuous state/action spaces. In these applications, function approximation is useful for building a compact representation of the value function. One popular framework for implementing such function approximation is Neural Fitted Q-Iteration (NFQ) [1]. However NFQ is based solely on the value returns, without making use of explicit structural information from the state space. We have extended the idea of NFQ and proposed a new reinforcement learning approach in which a Deep Belief Network (DBN) [2] is first trained generatively to model the state-action space with a hierarchy of latent binary variables, and the parameters of this model are then used to initialize a neural network value function approximator trained using NFQ.

The unsupervised pre-training phase in DBNs initializes the parameters of the network in a region of the parameter space that is more likely to contain good solutions, given the available data [3]. On the other hand, gathering data in a Reinforcement Learning (RL) scenario will often result in *imbalanced* data. This implies that in order to take advantage of the pre-training in RL, we need to adjust the data to cover interesting regions of the state space, while avoiding bias towards regions that are densely covered by the training set. Experiments confirm that when the initial data is wisely collected and also under-sampled to have a smoother distribution, our approach will significantly increase the learning efficiency.

II. Combining DBNs and RL

Our proposed approach is displayed in Algorithm 1. The two major steps of the algorithm are:

1) Pre-train the DBN on the initial training set.
2) Generate new data using current estimate of the Q-function; append this data to the training set; train the DBN on the training set to get a new estimate of the Q-function; update target values of the training set based on the Q-function; repeat this step until the termination condition is satisfied.

Algorithm 1 DeepRL

```
Input: a set of transition samples $D$, a binary flag $pretrain$; Output: $Q$-value function $Q_N$
$k \leftarrow 0$
if $pretrain = true$ then
  $Q_0 \leftarrow$ pretrain_DBN($D$)
else
  $Q_0 \leftarrow$ rand_init_DBN
end if
repeat
  generate_pattern_set $P$= $\{(input^i, target^i)\}$ where:
      $input^i \leftarrow (s^i, a^i)$,
      $target^i \leftarrow c(s^i, a^i, s'^i) + \gamma min'_a Q_k(s'^i, a')$
      $D \leftarrow$ append($D, P$)
  $Q_{k+1} \leftarrow$ train_DBN($D$)
  $k \leftarrow k + 1$
  for all $(input^j, target^j)$ in $D$ do
    $target^j \leftarrow c(s^j, a^j, s'^j) + \gamma min'_a Q_k(s'^j, a')$
  end for
until $K$ = $N$ or $Q_k \approx Q_{k-1}$
```

III. Experiments

To show the advantages of pre-training in RL problems, three experiments were performed:

1) We applied DeepRL algorithm to Mountain Car and Puddle World problems. The initial data is collected in two ways:

- With hint-to-goal heuristic (Several datapoints inside the goal region were manually added to the training set).
- Without hint-to-goal heuristic.

The learning process consists of 500 episodes for Mountain Car and 150 episodes for Puddle World. Each episode begins with generating a 50-step trajectory, using the current estimate of the Q-function. The performance is tested after every 10 episodes of learning in Mountain Car and after every 5 episodes in Puddle World, on 1000 random starting points. Fig. 1 (left column) shows that with hint-to-goal, pre-training helps since it defines a bias towards the goal area in the state space. But in case of absence of the hint-to-goal, pre-training actually hurts the performance, because the trajectory generated by the random policy has biased the parameters towards some undesirable areas of the state space that were covered by the pre-training set.

2) We repeated the case where pre-training is done on a random walk, and compare it with another case where the pre-training set is a 50-step trace from a good policy (learned in a previous experiment). In Mountain Car, a random walk mostly ends up in the valley, however a good policy escapes the valley more easily and gets to other regions of the space, giving good coverage of the state space. Conversely, a random policy in Puddle World can cover almost the entire state space, while a good policy typically avoids and misses the puddle area, so that much of the state space remains unsampled. This explains the opposite effects in Fig 1 (middle column), in which pretraining with a good policy helps in Mountain Car but hurts in Puddle World.
3) Finally, in order to solve the problem of imbalanced data, we under-sampled the training set to remove redundant datapoints before using it in pre-training. Fig 1 (right column) shows that this method significantly improves the learning performance in Mountain Car.

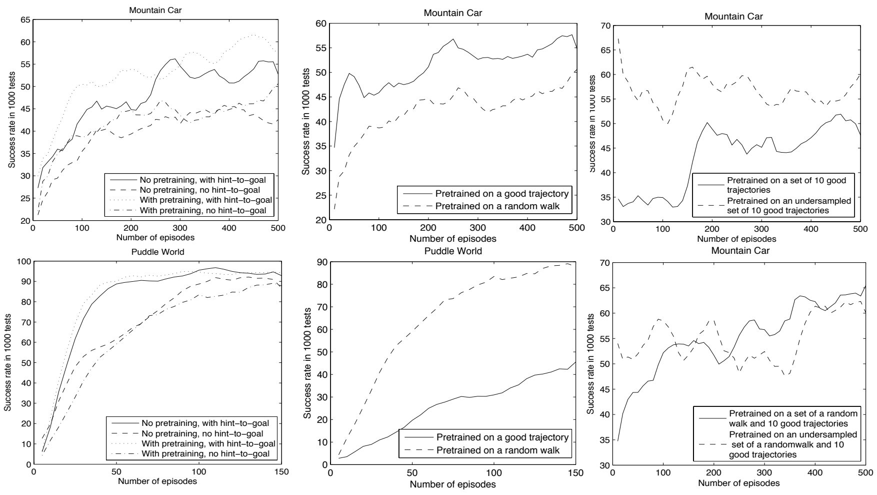

Fig. 1. Left column: Performance during learning with/without pre-training and with/without the hint-to-goal heuristic in Mountain Car (top) and Puddle World (bottom). Middle Column: Performance during learning when the pre-training data is 1) a random walk, and 2) a good trajectory generated by a successful policy, in Mountain Car (top) and Puddle World (bottom). Right column: The effect of under-sampling the pre-training data in Mountain Car. All curves are averaged over 40 trials

IV. Conclusion

Our experiments indicate that the unsupervised pre-training in DBNs can be very helpful in pulling the parameters toward interesting solutions in continuous state/action reinforcement learning problems, if the pre-training data covers the desirable areas of the state space. To overcome the problem of imbalanced data in reinforcement learning problems, we added datapoints from important areas of the state space to the training set and under-sampled this set to make the data distribution smoother. These adjustments considerably improved the performance.

Acknowledgments

This research was supported by ONR "Science of Autonomy" contract N00014-09-1-065 and DARPA contract N10AP20008. The authors would also like to thank Tom Walsh for his contributions to this work.

References

[1] M. Riedmiller, *Neural fitted Q-iteration - first experiences with a data effcient neural reinforcement learning method*, In Proceedings of ECML 2005, 317–328. Porto, Portugal, 2005
[2] G. E. Hinton, S. Osindero, and Y. Teh, *A fast learning algorithm for deep belief nets*, Neural Computation, 18: 1527–1554, 2006.
[3] D. Erhan, Y. Bengio, A. Courville, P. Manzagol, P. Vincent, and S. Bengio, *Why does unsupervised pre-training help deep learning?*, In Proceedings of AISTATS 2010, 201–208. Chia Laguna, Sardinia, Italy, 2010.