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

Abstract

We describe a continuous state/action reinforcement learning method which uses deep belief networks (DBNs) in conjunction with a value function-based reinforcement learning algorithm to learn effective control policies. Our approach is to first learn a model of the state-action space from data in an unsupervised pre-training phase, and then use neural-fitted Q-iteration (NFQ) to learn an accurate value function approximator (analogous to a "fine-tuning" phase when training DBNs for classification). Our experiments suggest that this approach has the potential to significantly increase the efficiency of the learning process in NFQ, provided care is taken to ensure the initial data covers interesting areas of the state-action space, and may be particularly useful in transfer learning settings.

1 Introduction

Real-world tasks often require learning methods to deal with continuous state and continuous action spaces. In these applications, function approximation is often useful in building a compact representation of the value function. One popular framework for implementing such function approximation is the fitted Q-iteration family of algorithms (Ernst, Geurts, and Wehenkel, 2005). These algorithms are a special form of the Experience Replay technique (Lin, 1982), where Q-iteration is performed on all transition experiences gathered so far. An extension of fitted Q-iteration is the neural fitted Q-iteration (NFQ) algorithm (Riedmiller, 2005), which is an effective method for training a Q-function represented by a multilayer perceptron. NFQ has been successful in difficult benchmark tasks such as Keepaway (Kalyanakrishnan and Stone, 2007).

Value-function approximation algorithms such as these are solely based on the value returns, without making use of explicit structural information from the state-action space. In this paper, we have extended the idea of NFQ to incorporate deep belief networks (DBNs, Hinton and Salakhutdinov 2006). In our method, a DBN 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. Hinton, Osindero, and Teh (2006) describe DBNs as probabilistic generative models composed of multiple layers of stochastic latent variables, and can be viewed as a way to train neural networks in a greedy layer-wise fashion. The latent variables in these networks, sometimes called "feature detectors", model the structure of the input data by leveraging structural and distributional patterns in the training data.

In standard neural network training, the weight adaptation is performed using backpropagation of errors in predicting the target (i.e., class label or value function). Hinton, Osindero, and Teh (2006) argue that the error signal from the task (e.g., classification) is not enough information to learn a good model with so many parameters. The pre-training in DBNs adapts the weights to capture the structure of the inputs before the task (targets) is introduced. Thus, performing both the pre-training and the normal backpropagation means that there are now *two* influences to adapting the weights: the inherent structure *and* the error.

Although DBNs have been very successful for pattern recognition problems, for instance on the MNIST database of hand-written digits (Hinton, Osindero, and Teh, 2006), they have not yet been used for learning agent controllers. Since learning from experience is an important part of most AI systems, in this work, we explore potential benefits of using DBNs as Q-function approximators in Reinforcement Learning (RL) problems, and will show that an RL agent can benefit from the DBN structure and training similarly to its supervised counterparts.

As discussed in (Erhan et al., 2009), 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. This will introduce a potential policy bias. On the other hand, gathering data in an RL scenario based on even a good exploration policy will result in *imbalanced* data. This implies that in order to take advantage of the pre-training in RL, we need to modify and adjust the data and provide the DBN with balanced initial data that covers interesting regions of the state space, while avoiding bias in the parameters towards high density regions. Our experiments confirm this hypothesis and show that when the initial data is wisely collected and also under-sampled to have a more uniform distribution

of the state space, our approach can significantly increase the sample efficiency of NFQ.

2 Background

DBNs are probabilistic graphical models built by stacking up restricted Boltzmann machines (RBMs) (Smolensky, 1987). An RBM is an undirected graphical model that consists of one layer of visible and one layer of hidden Bernoulli random variables (or units), with no connections between units of the same layer. Connections between layers are bi-directional and symmetric, which means both directions share the same weights and information flows in both directions. Fig. 1 (left) shows an RBM with 4 visible and 3 hidden units. A sample DBN is illustrated in Fig. 1 (right).

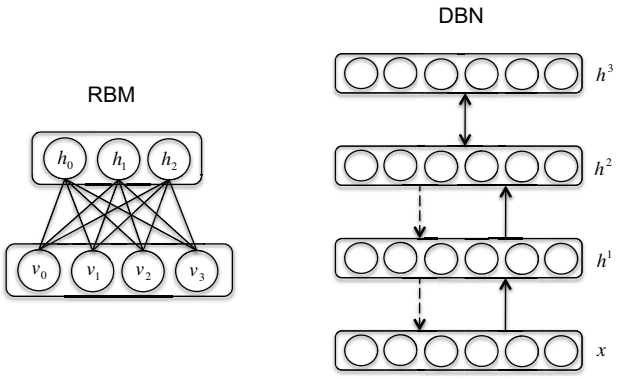

Figure 1: An RBM with 4 visible (input) and 3 hidden units (left) and a DBN with the same number of units in all layers (right).

DBNs can be trained using the contrastive divergence (CD) algorithm (Hinton, 2002) to extract a deep hierarchical representation of the training data. During the learning process, the DBN is first trained one layer at a time, in a greedy unsupervised manner, by treating the values of hidden units in each layer as the training data for the next layer (except for the first layer, which is fed with the raw input data). This learning procedure, called *pre-training*, finds a set of weights that determine how the variables in one layer depend on the variables in the layer above. These parameters capture the structural properties of the training data. If the network is to be used for a classification task, then a supervised discriminative fine-tuning is performed by adding an extra layer of output units and backpropagating the error derivatives (using some form of stochastic gradient descent, or SGD).

Erhan et al. (2009) studies the reasons why pre-trained deep networks work much better than traditional neural networks and proposes several possible explanations. One possible explanation is that pre-training initializes the parameters of the network in an area of parameter space where optimization is easier and a better local optima is found. This is equivalent to penalizing solutions that are outside a particular region of the solution space. Another explanation is that pre-training acts as a kind of *regularizer* that minimizes the variance and introduces a bias towards configurations of the parameters that stochastic gradient descent can explore during the supervised learning phase, by defining a data-dependent prior on the parameters obtained through the unsupervised learning. In other words, pre-training implicitly imposes constraints on the parameters of the network to specify which minimum out of all local minima of the objective function is desired. The effect of pre-training relies on the assumption that the true target conditional distribution $P(Y|X)$, shares structure with the input distribution $P(X)$.

For non-convex optimization problems, SGD methods have the problem that they can often be driven to poor solutions by the order in which training examples are seen, particularly if the data is highly *imbalanced*. As explained by (Provost, 2000), many learning algorithms suffer from data imbalance because of the following assumptions that are built into them:

- The goal is to maximize the accuracy, and
- The learning algorithm will operate on the data drawn from the same distribution as the training data.

As a result, machine learning techniques can produce unacceptable results if trained on imbalanced datasets. For instance, suppose 99% of the samples used to train a classifier come from one class. Then the learning algorithm can try to achieve the best possible accuracy by assigning everything to the majority class. The problems of imbalanced data can be especially pronounced in RL applications in which the value function is represented by a function approximator. In this case, since data collection is performed by executing some inherently biased policy, the value function approximator will ignore regions of the state space where little data is available. Although Erhan et al. (2010) found the strong effect of early examples was diminished in DBNs in an experiment with an online form of MNIST, it seems plausible that this could still be a problem for DBNs in RL if pre-training is viewed as a regularizer which defines a bias towards regions of the state space that are similar to the pre-training data.

Many techniques can be applied to solve the problem of imbalanced data. A possible approach is to manually balance the data by adding samples from minority classes or regions where it is hard to collect data. The hint-to-goal huristic used by Riedmiller (2005) is an example of this approach. In RL domains, another way of balancing the data is to use a better sampling strategy by adding traces of an efficient policy to the training data.

Two other common methods for artificially balancing the data are *under-sampling* (ignoring samples from the majority), and *over-sampling* (replicating cases from the minority). These techniques and many other methods are discussed in (Chawla, Japkowics, and Kotcz, 2004). We explore a form of *informed undersampling* (Liu, Wu, and Zhou, 2009; Drummond and Holte, 2003) at the end of this paper and find that this can indeed dramatically enhance the ability of DBNs to improve NFQ.

3 Combining DBNs and RL

The basic idea underlying combining DBNs with RL is to take advantage of the unsupervised pre-training phase in DBNs, and then use the DBN as the starting point for a neural network function approximator for representing the Q-function. This points us towards extending the popular neural fitted Q-iteration framework to a version in which the Q-function is approximated with a DBN. Our proposed algorithm is displayed in Algorithm 1.

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
  get_new_experiences 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 $Q_k \approx Q_{k-1}$

The algorithm consists of two main steps. First, the training set and the DBN are initialized. Depending on the setting that we would like to use in a particular experiment, we can use different initializations. The initial transition samples are a set of $< state, action, target >$ tuples. If we decide to start with unsupervised pre-training, the DBN is pre-trained on the set of transition samples, without taking the target values (i.e., return estimates) into account.

The second step of the algorithm is the reinforcement learning loop. From this point on, the algorithm works similar to the NFQ approach and the DBN weights are used as the initial configuration of a regular neural network value function approximator. This part of the algorithm begins with using the current Q-function for a greedy policy which is run in the environment to gather an additional set of experiences, which are then attached to the initial training set. Afterwards, the combined set is used to update the network and get a new estimate of the Q-function. This is done by using the current Q-function to recalculate the target values for every experience tuple in the updated training set, and then SGD is used to update the value function outputs. These steps are repeated N times, or until the Q-function converges and the updated targets are successfully learned. To show progress, we periodically test the current policy in the environment without using those experiences for learning.

4 Experiments

4.1 The effect of pre-training on the efficiency of learning

In our first set of experiments, we would like to test if pre-training improves the performance in some standard benchmark RL problems. The first domain we have selected for this purpose is the Mountain Car problem in which the system has to reach a certain area in state space, and the task will immediately terminate as soon as it gets there. We will use the same settings as Riedmiller (2005) for the cost function and system specifications. For the network, we used two hidden layers with 5 units in all cases, as in (Riedmiller, 2005).

The learning process consists of 500 episodes. Each episode begins with generating a 50-step trajectory, using the current estimate of the Q-function. Then this trajectory is added to the training set and the entire set is fed to the neural network to learn a new approximation of the Q-function. After the network is trained, target values of the training set are updated given the new estimate of the Q-function.

The system performance is tested after every 10 episodes of learning. The test set comprises 1000 random initial points in the state space and the system has to reach the goal region starting from each of these initial points within 250 steps. The performance is defined as the percentage of the tests that are completed successfully.

We initialize our training set in two different ways. In first case, the initial data is a 50-step random walk in the state space. We collect this data by running a completely random policy. In second case, in addition to the 50-step random walk, we also use the "hint-to-goal huristic" from Riedmiller (2005) and add 100 artificially generated random points from near the goal region to the training set. This makes our experiments a more fair comparison with the work of Riedmiller (2005).

The above settings and also whether we start the learning process with or without pre-training on the initial dataset will form the four cases of our experiment:

- Without pre-training, without hint-to-goal
- Without pre-training, with hint-to-goal
- With pre-traing, without hint-to-goal
- With pre-training, with hint-to-goal

Note that the second case is in fact equivalent to the NFQ approach presented in (Riedmiller, 2005).

Figure 2 (top) shows the results of the above four cases, averaged over 40 replications of the experiment with different random seed. Comparing with vs. without pre-training curves indicates that pre-training helps, especially at the beginning of learning, and this is true both for with and without hint-to-goal heuristic.

As mentioned in previous sections, pre-training seems to introduce a bias towards configurations of the parameters that are more desirable for later gradient learning steps. This explains why the case with the hint-to-goal heuristic shows better performance when learning begins with a pre-training phase. In this case the DBN learns that the goal is more

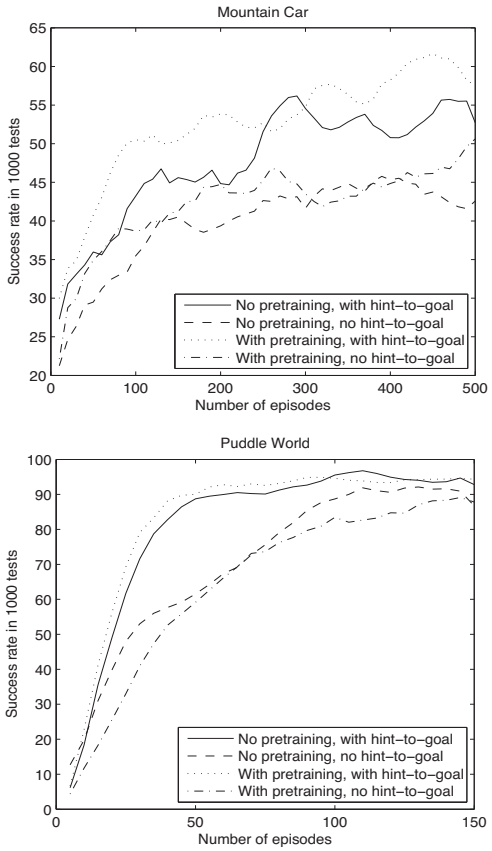

Figure 2: Comparison of the performance during learning with/without pre-training and with/without the hint-to-goal heuristic, averaged over 40 trials in Mountain Car (top) and Puddle World (bottom).

likely to be around the points which are more often seen in the training data.

However without hint-to-goal, it seems that pre-training only make a little difference, if any. This may be because a random walk is not a good enough representative of the desirable regions of the state space, since the system often gets stuck at the bottom of the hill and the data collected during random walk will be highly imbalanced and mostly includes points inside the valley.

This is even clearer in Fig. 2 (bottom), where we repeat the same experiment on our second domain, Puddle World. In this case, the environment is a 2-dimensional grid world with a puddle located in a random position on the grid. The agent tries to reach the goal area, which is at a corner of the grid, while avoiding the puddle. All the experiment settings are as before, except that learning continues for 150 episodes and the performance is tested after each 5 episodes over a set of 1000 random starting points. As we can see, in case of absence of hint-to-goal, pre-training seems to do nothing, or perhaps even slightly *hurt* performance. We believe this may be because pre-training with trajectories generated by a random policy may be biasing the parameters towards an area of the state space too dissimilar from the states experienced when the agent is performing well, thereby reducing the ability of later data from improved policies to influence the weight configuration.

4.2 Providing the DBN with good initial data

Although these initial results are promising and it seems credible to use DBNs to improve function approximation in NFQ, the results of Erhan et al. (2009) makes us think that there might be an additional advantage in pre-training. If we give the deep nets some good data up-front, it puts the parameters of the DBN in a better region of the parameter space to learn from. This means that it might be very good for a transfer learning setting, where we already know something about the task, but need to do some more learning.

In order to test this idea, in our second experiment we repeat the case where pre-training is done on a random walk, and then compare it with another case where the pre-training set is a 50-step trace of a good policy instead of a random walk. This "good" policy is chosen from those policies obtained during the first experiment that attained 100% success. The result of this experiment is quite surprising.

In Mountain Car domain, a random policy mostly ends up in the valley because of the gravity effect and remains stuck in that area. So a trace of a random policy by no means covers all the interesting areas of the state space. On the other hand, a good policy can escape the valley more easily and get to other regions of the space. This means that an agent with this policy will explore the entire state space more thoroughly, thus the training set is more balanced. Fig. 3 (top) confirms that when pre-training a DBN with initial data from a "good" policy, the resulting policy trained with NFQ can significantly outperform the one trained on a random walk.

Contrarily, as we can see in Fig. 3 (bottom), the trajectory generated by the *random* policy seems to be much more helpful than the trajectory obtained from a good policy in Puddle World. The intuitive explanation for this is that unlike Mountain Car, a random policy in Puddle World can cover almost every possible area of the space, while a good policy might always avoid the puddle and the area around it. So with a good policy, the agent will not have any information about how good or bad it is to be inside the puddle. The result is that the DBN pre-training initializes the weights in such a way that it is apparently very difficult to learn a representation of the value function for important (perhaps importantly *bad*) parts of the state space, greatly hurting the performance of the agent.

4.3 Under-sampling the training data

The idea of providing the DBN with a good trajectory (which can be a trace of a good policy, or even a random walk, depending on the domain), to some extent deals with the issue of imbalanced data in the sense that it covers the areas of the state space that are normally hard to reach. This makes DBNs an appropriate tool for transfer learning. The problem with this method is that it can generate several redundant samples in the state space which can have the effect of increasing the data imbalance problem. In order

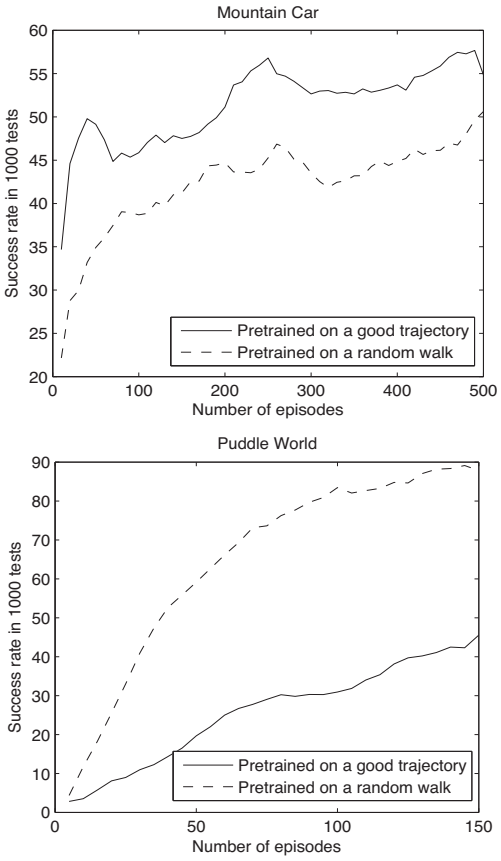

Figure 3: Comparison of the performance during learning when the pre-training data is 1) a random walk, and 2) a good trajectory generated by a successful policy, in Mountain Car (top) and Puddle World (bottom). Both curves are averaged over 40 trials.

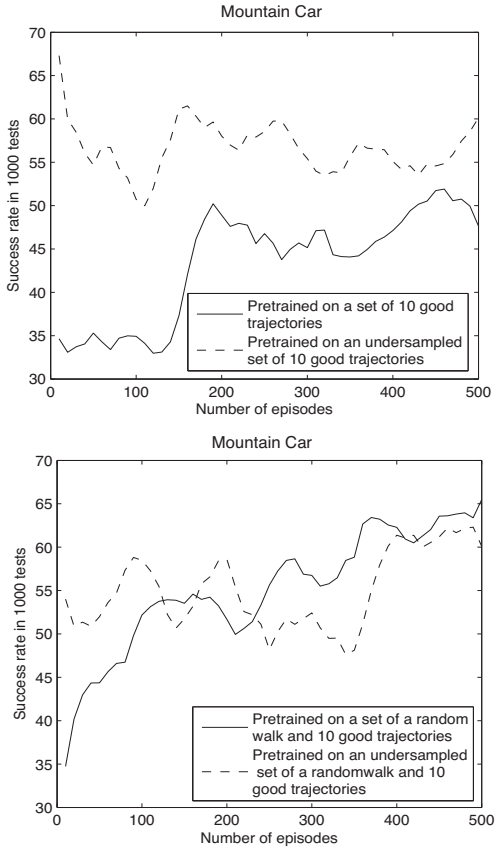

Figure 4: The effect of under-sampling the pre-training data in Mountain Car problem.

to overcome this problem, we under-sample the initial pre-training data by discretizing the state space and removing samples which appear more than once in each bin, resulting in a smoother (higher entropy) distribution of training data. Fig. 4 shows that this under-sampling method makes a significant improvement in the learning performance in Mountain Car problem, producing the best overall performance we have seen so far.

5 Conclusions and Future Work

In this paper we explored the potential benefits of using DBNs to approximate the Q-function in continuous state/action RL problems where it is impractical (or impossible) to use a tabular representation for the Q-function. We saw that the unsupervised pre-training of DBNs can be very helpful in initializing the parameters of the network to areas of the solution space that allow faster learning of good policies. However, this can be highly dependent on how the initial training data is collected by the agent. That is, since pre-training defines a bias towards representing aspects of the initial data, the final solution can be very efficient if the data covers the desirable areas of the state space, but highly inefficient if the data does not cover much of the state space.

We also tried to address the problem of imbalanced data that often arises in RL problems, and which still (apparently) can cause problems in DBNs. For this purpose, we first added some datapoints around the goal region in the state space that were manually generated using the hint-to-goal heuristic to the initial data and used this set to pre-train the DBN. Then, we also added traces of an efficient policy to this training set. Finally, we applied an under-sampling method to remove duplicate datapoints and make the data distribution smoother. Our results show that each of these steps can significantly improve the efficiency of the solutions.

We would like to extend this work in several different directions. First, The fact that DBNs are in fact generative statistical models means that we should really take advantage of the ability to sample datapoints from the state, action and/or reward/cost spaces in RL problems. This might be particularly useful in various forms of model-based RL, or even actor-critic RL in which the actor must be trained by samples drawn from state-action pairs modeled as "good" by a DBN-based value function approximator. Second, we intend to study the problem of imbalanced data more deeply in

RL applications. This includes introducing a variation on the deep net algorithm that can deal with the potential bias problem by either resolving the bias issue, or efficiently learn to recognize if early data appears to have been biased based on later data. Finally, we would like to apply our proposed algorithm to more practical and complex environments to see how well it can scale to real-world problems.

6 Acknowledgments

This research was supported by ONR contract N00014-09-1-065 and DARPA contract N10AP20008. The authors would also like to thank Tom Walsh for his contributions to this work, as well as Shivaram Kalyanakrishnan and Matthew Taylor for many helpful discussions.

References

Chawla, N. V.; Japkowics, N.; and Kotcz, A. 2004. Editorial: special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter - Special issue on learning from imbalanced datasets* 6(1).

Drummond, C., and Holte, R. 2003. C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *ICML'03 Workshop on Learning from Imbalanced Datasets*.

Erhan, D.; Manzagol, P. A.; Bengio, Y.; Bengio, S.; and Vincent, P. 2009. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *AISTATS'2009*, 153–160.

Erhan, D.; Bengio, Y.; Courville, A.; Manzagol, P.; Vincent, P.; and Bengio, S. 2010. Why does unsupervised pre-training help deep learning? In *AISTATS 2010*, 201–208.

Ernst, D.; Geurts, P.; and Wehenkel, L. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 6:503–556.

Hinton, G. E., and Salakhutdinov, R. R. 2006. Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507.

Hinton, G. E.; Osindero, S.; and Teh, Y. 2006. A fast learning algorithm for deep belief nets. *Neural Computation* 18:1527–1554.

Hinton, G. 2002. Training products of experts by minimizing contrastive divergence. *Neural Computation* 14(8):1771–1800.

Kalyanakrishnan, S., and Stone, P. 2007. Batch reinforcement learning in a complex domain. In *AAMAS07*.

Lin, L. J. 1982. Self-improving reactive agents based on reinforcement learning, planning and teaching. In *Machine Learning*, volume 8, 293–321.

Liu, X.; Wu, J.; and Zhou, Z. 2009. Exploratory undersampling for class-imbalance learning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 39(2):539–550.

Provost, F. 2000. Machine learning from imbalanced data sets 101 (extended abstract). AAAI'2000 Workshop on Imbalanced Data Sets.
Riedmiller, M. 2005. Neural fitted q-iteration - first experiences with a data efficient neural reinforcement learning method. In *ECML 2005*, 317–328.

Smolensky, P. 1987. Information processing in dynamical information processing in dynamical systems: foundations of harmony theory. In Rumelhart, D. E.; McClelland, J. L.; et al., eds., *Parallel Distributed Processing*, volume 1. Cambridge: MIT Press. 194–281.