JMLR: Workshop and Conference Proceedings 24:43–57, 2012 10th European Workshop on Reinforcement Learning

Actor-Critic Reinforcement Learning with Energy-Based Policies

Nicolas Heess NHEESS@GATSBY.UCL.AC.UK

David Silver D.SILVER@CS.UCL.AC.UK

Yee Whye Teh Y.W.TEH@STATS.OX.AC.UK

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

We consider reinforcement learning in Markov decision processes with high dimensional state and action spaces. We parametrize policies using energy-based models (particularly restricted Boltzmann machines), and train them using policy gradient learning. Our approach builds upon Sallans and Hinton (2004), who parameterized value functions using energy-based models, trained using a non-linear variant of temporal-difference (TD) learning. Unfortunately, non-linear TD is known to diverge in theory and practice. We introduce the first sound and efficient algorithm for training energy-based policies, based on an actor-critic architecture. Our algorithm is computationally efficient, converges close to a local optimum, and outperforms Sallans and Hinton (2004) in several high dimensional domains.

1. Introduction

A major challenge in reinforcement learning is to find successful policies in problems with high-dimensional state or action spaces. In order to solve these problems effectively, it is advantageous to learn representations of the state and action spaces that enable a policy to achieve high performance. In this paper, we develop a framework for parameterizing richly structured policies, and for finding those with (locally) optimal performance.

We consider a class of policies based on *energy-based models* [LeCun et al., 2006], where the (negative) log probability of selecting an action is proportional to an energy function. Energy-based models have been widely studied in both supervised and unsupervised learning. They can learn deep, distributed representations of high-dimensional data (such as images) and model high-order dependencies, and here we will use them to directly parametrize representations over states and actions. We explore in detail a class of energy-based models called restricted Boltzmann machines (RBMs; Freund and Haussler, 1994; Welling et al., 2004). RBMs and conditional RBMs have been applied to a range of challenging tasks including multi-class classification [Larochelle and Bengio, 2008], collaborative filtering [Salakhutdinov et al., 2007], motion capture modeling [Taylor and Hinton, 2009], modeling of transformations in natural images [Memisevic and Hinton, 2010], and structured output prediction [Mnih et al., 2011].

To successfully use energy-based policies for reinforcement learning, it is necessary to address several challenges. First, the parameterized representations can be highly nonlinear, leading to non-convex objectives with multiple optima. Second, it is often intractable to compute the partition function that normalizes the energy function into a probability distribution. Third, evaluating a policy typically has high variance, since the performance depends on the rewards accumulated over a complete trajectory. Prior work [Sallans, 2002; Sallans and Hinton, 2004; Otsuka et al., 2010; Elfwing et al., 2010] partially addressed these issues using value-based reinforcement learning. The idea was to approximate the action-value function by the free energy of an energy-based model, and to train it by temporal-difference (TD) learning. However, this approach has a serious drawback: TD learning is known to diverge, both in theory and in practice, when using non-linear value functions [Tsitsiklis and Van Roy, 1997]. In addition, the policy is based on an arbitrary temperature which makes the algorithm hard to tune in practice.

We introduce a new approach to reinforcement learning with energy-based policies. The basic idea is to adjust the policy parameters to follow the gradient of the policy performance, using sample trajectories to obtain estimates of the gradient [Williams, 1992; Sutton et al., 1999; Baxter and Bartlett, 2001]. One major advantage of this approach is that it converges to a local optimum, even for non-linear policies. Simple policy gradient methods are sensitive to the parameterization of the policy, and also suffer from high variance in the gradient estimates. We address these shortcomings by following the *natural gradient*, which reduces the dependence of the performance of the policy gradient on the parameterization [Kakade, 2001]; and by using an actor-critic architecture, which uses an approximate value function to reduce the variance in the gradient estimates [Peters and Schaal, 2008; Bhatnagar et al., 2009]. We introduce two novel natural actor-critic algorithms for efficiently following the gradient of energy-based policies without resorting to explicit computation of the partition function. We apply our algorithms to learn RBM policies in a number of tasks, finding that they converge quickly to reasonable solutions in all tasks and consistently outperform Sallans and Hinton [2004], which sometimes even fails to converge.

2. Energy-based policies

We consider stationary Markov decision processes (MDPs) with bounded rewards and high dimensional state and action spaces denoted $\mathcal{S}$ and $\mathcal{A}$ respectively. Denote the state, action and reward at time t by s_t, a_t and r_t respectively, and the state transition probabilities and expected reward function by $P(s_{t+1}|s_t, a_t)$ and $R(s_t, a_t)$. We will consider stochastic and stationary polices described by conditional distributions over actions $\pi^\theta(a; s)$ parameterized by θ. We assume that given policy π^θ the MDP is ergodic with stationary distribution d^θ.

In this paper we consider energy-based policies which can be expressed as conditional joint distributions over actions a and a set of latent variables h:

$$\pi^\theta(a, h; s) = \tfrac{1}{Z(s)} e^{\phi(s,a,h)^\top \theta} \tag{1}$$

where $\phi(s, a, h)$ are a pre-defined set of features and $Z(s) = \sum_{a,h} \exp(\phi(s, a, h)^\top \theta)$ is the normalizing *partition function*. The policy itself is then obtained by marginalizing out h. The latent variables allow energy-based policies to parametrize complex non-linear and non-factorial relationships between actions and states, even though the underlying parameterization (1) is log linear in the features $\phi(s, a, h)$. For example, in a conditional restricted Boltzmann machine (RBM), the states s, actions a and latent variables h are all high-dimensional binary vectors, and (1) is parameterized as:

$$\pi^\theta(a,h;s) = \tfrac{1}{Z(s)} e^{s^\top W_s h + a^\top W_a h + b_s^\top s + b_h^\top h + b_a^\top a} \tag{2}$$

where the parameters are matrices W_s, W_a and vectors b_s, b_a, b_h of appropriate dimensionalities. Marginalizing out h, we get a non-linearly parameterized policy:

$$F^\theta(s,a) = -b_s^\top s - b_a^\top a - \textstyle\sum_i \log(1 + e^{s^\top W_{si} + a^\top W_{ai} + b_{hi}}), \qquad \pi^\theta(a;s) = \tfrac{1}{Z(s)} e^{-F(s,a;\theta)} \tag{3}$$

where i indices the latent variables, and W_{si}, W_{ai}, b_{hi} are parameters associated with latent variable h_i. The quantity $F(s,a;\theta)$ is called the *free energy.*

2.1. Free-energy Sarsa

Sallans and Hinton [2004] use the RBM free energy as a non-linear function approximator. Specifically, they approximate the action-value function $Q^\theta(s,a)$ of a discounted MDP using $-F^\theta(s,a)$, and select actions according to a Boltzmann exploration policy,

$$\pi^\theta(\mathbf{a};\mathbf{s},T) = \tfrac{1}{Z(\mathbf{s},T)} e^{-F(\mathbf{s},\mathbf{a};\theta)/T} \tag{4}$$

where the temperature T scales units of reward into units of probability: as $T \to \infty$ the policy becomes uniform; as $T \to 0$ it becomes greedy. The parameters of the RBM are updated using a non-linear version of the Sarsa algorithm, with time-varying step size β_t,

$$\begin{aligned} \delta_t &= r_{t+1} - \gamma F^{\theta_t}(s_{t+1}, a_{t+1}) + F^{\theta_t}(s_t, a_t) \\ \theta_{t+1} &= \theta_t - \beta_t \delta_t \nabla_\theta F^{\theta_t}(s_t, a_t), \end{aligned} \tag{5}$$

where δ_t is the TD error and γ is the discount factor of the MDP. We refer to this algorithm as energy-based Sarsa (ESARSA). Since Sarsa is a control algorithm based on TD learning, it may diverge with non-linear function approximations [Tsitsiklis and Van Roy, 1997]. Recently, convergent non-linear prediction algorithms based on gradient TD have been developed, however even these may still diverge when used for control [Maei et al., 2009].

2.2. Policy gradient

In this paper we consider an average reward formulation of MDPs. Our objective is to find parameters θ that (locally) maximize the *average reward* per time-step $J(\theta)$,

$$J(\theta) = \lim_{T\to\infty} \tfrac{1}{T}\, \mathrm{E}^\theta \left[\textstyle\sum_{t=1}^T r_t \right] = \mathrm{E}^\theta_{sa} \left[R(s,a) \right] \tag{6}$$

where E^θ denotes expectation under π^θ and E^θ_{sa} denotes expectation under the stationary distribution (where $s \sim d^\theta$ and $a|s \sim \pi^\theta(\cdot;s)$). The state-value function $V^\theta(s)$ and action-value function $Q^\theta(s,a)$ for policy π^θ are given by the differential reward from s (and a),

$$Q^\theta(s,a) = \sum_{t=1}^\infty \mathrm{E}^\theta \left[r_t - J(\theta) | s_0 = s, a_0 = a \right] \tag{7}$$

$$V^\theta(s) = \sum_{t=1}^\infty \mathrm{E}^\theta \left[r_t - J(\theta) | s_0 = s \right] = \sum_{a\in\mathcal{A}} \pi^\theta(s;a) Q^\theta(s,a) = \mathrm{E}^\theta_a [Q^\theta(s,a)] \tag{8}$$

where E_a^θ denotes expectation under $a|s \sim \pi^\theta(\cdot; s)$. The *advantage function* is the differential value for action a in state s:

$$A^\theta(s,a) = Q^\theta(s,a) - V^\theta(s) \tag{9}$$

Policy gradient methods update the policy parameters θ to follow the gradient of the average reward. The *policy gradient theorem* [Baxter and Bartlett, 2001; Sutton et al., 1999; Bhatnagar et al., 2009] provides the gradient of $J(\theta)$ with respect to θ,

$$\nabla_\theta J(\theta) = \mathrm{E}_{sa}^\theta \left[A^\theta(s,a) \nabla_\theta \log \pi^\theta(a;s) \right] \tag{10}$$

In practice, the advantage function A^θ is unknown and must be approximated. *Actor-critic* policy gradient algorithms estimate the advantage function, $\hat{A}^{\mathbf{w}} \approx A^\theta$, and update this estimate (the *critic*) in parallel with the policy parameters (the *actor*). If the parameterization of the approximation $\hat{A}^{\mathbf{w}}$ is *compatible* with the parameterization of the policy, then this approximation does not introduce any bias into the gradient direction [Sutton et al., 1999]. A parameterization is compatible if it satisfies

$$\nabla_{\mathbf{w}} \hat{A}^{\mathbf{w}}(s,a) = \nabla_\theta \log \pi^\theta(a;s) \tag{11}$$

That is, $\hat{A}^{\mathbf{w}}(s,a) = \psi^\theta(s,a)^\top \mathbf{w}$ where $\psi^\theta(s,a) = \nabla_\theta \log \pi^\theta(a;s)$. Furthermore, the parameters $\mathbf{w}$ should minimize the mean-squared-error (MSE),

$$\mathbf{w} = \operatorname*{argmin}_{\mathbf{w}'} \mathrm{E}_{sa}^\theta \left[\left(A^\theta(s,a) - \psi^\theta(s,a)^\top \mathbf{w}' \right)^2 \right] \tag{12}$$

This can be obtained by stochastic gradient updates:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t(\delta_t - \psi^{\theta_t}(s_t,a_t)^\top \mathbf{w}_t)\psi^{\theta_t}(s_t,a_t) \tag{13}$$

where θ_t is the current policy parameter, α_t is a step size and δ_t is an unbiased estimate of $A^\theta(s_t,a_t)$ (later).

The direction of the vanilla policy gradient is sensitive to reparameterizations of the policy that don't affect the action probabilities. *Natural* policy gradient algorithms [Kakade, 2001; Peters and Schaal, 2008] remove this dependence, by finding the direction that improves $J(\theta)$ the most for a fixed small amount of change in distribution (e.g. measured by KL divergence). The natural policy gradient $\nabla_\theta^{\mathrm{nat}} J(\theta)$ is defined by

$$\nabla_\theta^{\mathrm{nat}} J(\theta) = G_\theta^{-1} \nabla J(\theta), \qquad G_\theta = \mathrm{E}_{sa}^\theta \left[\nabla \log \pi(s,a) \nabla \log \pi(s,a)^T \right] \tag{14}$$

where G_θ is the Fisher information matrix. When using compatible function approximation, we find that,

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathrm{E}_{sa}^\theta \left[\nabla_\theta \log \pi^\theta(a;s) A^\theta(s,a) \right] \\ &= \mathrm{E}_{sa}^\theta \left[\nabla_\theta \log \pi^\theta(a;s) (\nabla_\theta \log \pi^\theta(a;s))^T \mathbf{w} \right] = G_\theta \mathbf{w} \end{aligned} \tag{15}$$

and the natural gradient simplifies to $\nabla_\theta^{\mathrm{nat}} J(\theta) = \mathbf{w}$. Policy parameters updates are then:

$$\theta_{t+1} = \theta_t + \beta_t \mathbf{w}_{t+1}, \tag{16}$$

The final ingredient is estimating the advantage function $A^{\theta}(s,a)$. One unbiased estimate is the TD error:

$$\delta_t = r_{t+1} - J(\theta) + V^{\theta}(s_{t+1}) - V^{\theta}(s_t) \tag{17}$$

which is used by Bhatnagar et al. [2009] to motivate their natural TD actor-critic (NATDAC) algorithm: They approximate the state-value function, and hence the TD error, using a linear function approximator $\hat{V}_{\theta}(s) \approx f(s)^T \mathbf{v}$, where $f(s)$ is a state-dependent feature vector and $\mathbf{v}$ are updated by TD learning:

$$\begin{aligned} \hat{J}_{t+1} &= (1-\zeta_t)\hat{J}_t + \zeta_t r_{t+1} \\ \delta_t &= r_{t+1} - \hat{J}_{t+1} + f(s_{t+1})^T \mathbf{v}_t - f(s_t)^T \mathbf{v}_t \\ \mathbf{v}_{t+1} &= \mathbf{v}_t + \alpha_t \delta_t f(s_t) \end{aligned} \tag{18}$$

Provided that the average reward, TD error and critic are updated on a slower time scale than for the actor, and step sizes are reduced at appropriate rates, NATDAC converges to a policy achieving near-local maximum average reward [Bhatnagar et al., 2009, Section 5].

3. NATDAC for energy-based policies

Energy-based policies pose a challenge to policy gradient methods. Both vanilla policy gradient updates and natural actor-critic updates require the computation of compatible features $\psi^{\theta}(s,a) = \nabla_{\theta} \log \pi^{\theta}(s,a)$. For an energy-based policy, this requires the computation of two expectations:

$$\begin{aligned} \psi^{\theta}(s,a) &= \nabla_{\theta} \log \pi^{\theta}(a;s) = \nabla_{\theta} \log \textstyle\sum_h e^{\phi(s,a,h)^{\top}\theta} - \nabla_{\theta} \log Z(s) \\ &= \mathrm{E}^{\theta}_{h}\left[\phi(s,a,h)\right] - \mathrm{E}^{\theta}_{a'h}\left[\phi(s,a',h)\right] \overset{def}{=} \phi^{\theta}(s,a) - \mathrm{E}^{\theta}_{a'}\left[\phi^{\theta}(s,a')\right]. \end{aligned} \tag{19}$$

where E^{θ}_{h} is the expectation wrt the conditional of h given a and s while E^{θ}_{ah} is wrt both a and h. The first expectation, which we denote by $\phi^{\theta}(s,a)$, can be computed efficiently in a wide class of energy-based models, including RBMs. The second expectation involves a sum over all actions, and is intractable to compute in high-dimensional action spaces.

We now propose two incremental actor critic algorithms that avoid evaluating either this expectation or the partition function. The first algorithm *energy-based NATDAC* (ENATDAC) uses independent samples from the energy-based policy to approximate the intractable expectations. The second algorithm, *energy-based Q-value natural actor critic* (EQNAC) uses a deterministic approximation and has a lower computational requirement.

3.1. Energy-based NATDAC

For our first algorithm, we note that energy-based policies such as RBMs may be efficiently sampled, for example by blocked Gibbs sampling, without explicitly computing the partition function. Unbiased estimates of the update (13) can then be formed using two independent unbiased estimates of the compatible features $\psi^\theta(s,a)$,

$$\hat{\psi}'_t = \phi^{\theta_t}(s_t,a_t) - \frac{1}{K}\sum_{k=1}^{K}\phi^{\theta_t}(s_t,a'_k), \qquad \hat{\psi}''_t = \phi^{\theta_t}(s_t,a_t) - \frac{1}{L}\sum_{l=1}^{L}\phi^{\theta_t}(s_t,a''_l)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t\left(\delta_t - {\hat{\psi}'_t}{}^\top \mathbf{w}_t\right)\hat{\psi}''_t \tag{20}$$

where $a'_k, a''_l \sim \pi^{\theta_t}(\cdot;s)$ iid. Because (20) is equal in expectation to the critic update (13), the two time scale convergence proof for NATDAC [Bhatnagar et al., 2009, section 5.3] can be applied. Specifically, under our assumption that the critic is updated on a much slower time-scale than the actor, the policy θ_t and therefore ϕ^{θ_t} is effectively stationary during critic updates. As a result, the TD learning updates (20) are based on a function approximator that is linear in ϕ^{θ_t}, and do not suffer from the divergence issues faced by energy-based Sarsa.

One drawback of the NATDAC algorithm is that the quality of the state-value function approximator depends on the feature vector $f(s)$. Although the algorithm converges, the quality of the final policy may be significantly affected by the choice of features. In an energy-based framework, it is possible to marginalize over $\phi^\theta(s,a)$ to define an energy-based feature vector $f(s) = \mathrm{E}^\theta_a[\phi^\theta(s,a)]$. Again, this expectation is intractable to compute, but can be approximated by a third set of samples from the policy, $a'''_m \sim \pi^\theta(s,\cdot)$

$$\hat{f}_t = \frac{1}{M}\sum_{m=1}^{M}\phi^{\theta_t}(s_t,a'''_m) \tag{21}$$

Algorithm 4.1: ENATDAC

```
Input: $\hat{J}_0, \mathbf{v}_0, \mathbf{w}_0, \theta_0,\ s_0$
for $t = 0, 1, \ldots$ do
    $a_t, a'_{1\ldots K}, a''_{1\ldots L} \sim \pi^{\theta_t}(\cdot;s_t)$
    $s_{t+1} \sim P(\cdot|s_t,a_t)$
    $r_{t+1} = R(s_t,a_t)$
    $\hat{\psi}'_t = \phi^{\theta_t}(s_t,a_t) - \frac{1}{K}\sum_{k=1}^{K}\phi^{\theta_t}(s_t,a'_k)$
    $\hat{\psi}''_t = \phi^{\theta_t}(s_t,a_t) - \frac{1}{L}\sum_{l=1}^{L}\phi^{\theta_t}(s_t,a''_l)$
    $\hat{J}_{t+1} = (1-\zeta_t)\hat{J}_t + \zeta_t r_{t+1}$
    $\delta_t = \mathbf{r}_{t+1} - \hat{J}_{t+1} + \mathbf{v}^\top \hat{f}_{t+1} - \mathbf{v}^\top \hat{f}_t$
    $\mathbf{v}_{t+1} = \mathbf{v}_t + \alpha_t\delta_t f(s_t)$
    $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t(\delta_t - \hat{\psi}'^T_t \mathbf{w}_t)\hat{\psi}''_t$
    $\theta_{t+1} = \theta_t + \beta_t \mathbf{w}_{t+1}$
end for
```

Pseudocode for energy-based NATDAC is given in Algorithm 4.1. In summary, it uses an advantage actor-critic, based on TD learning with an energy-based feature vector, and using the sample approximation to the intractable expectations, and updates the policy by following the natural gradient.

3.2. Energy-based Q-value natural actor-critic

We now introduce our second algorithm, EQNAC, which is also based on a natural actor-critic approach. The key new idea is to note that, when using the compatible function approximator $\psi^\theta(s,a)$, the approximate advantage function $\hat{A}^{\mathbf{w}}(s,a)$ can be represented in terms of an approximate action-value $\hat{Q}^{\mathbf{w}}(s,a)$,

$$\begin{aligned}\hat{A}^{\mathbf{w}}(s,a) = \psi^\theta(s,a)^\top\mathbf{w} = \left(\phi^\theta(s,a) - \mathrm{E}^\theta_{a'}\left[\phi^\theta(s,a')\right]\right)^\top \mathbf{w} = \phi^\theta(s,a)^\top\mathbf{w} - \mathrm{E}^\theta_{a'}\left[\phi^\theta(s,a')\right]^\top \mathbf{w} \\ \overset{def}{=} \hat{Q}^{\mathbf{w}}(s,a) - \mathrm{E}^\theta_{a'}[\hat{Q}^{\mathbf{w}}(s,a')].\end{aligned} \tag{22}$$

This suggests that it is sufficient to estimate the action-value function $Q^\theta(s,a)$ by a linear approximator $\hat{Q}^{\mathbf{w}} = \phi^\theta(s,a)^\top \mathbf{w}$. This representation does not involve any intractable expectations. The EQNAC algorithm updates the critic parameters $\mathbf{w}$ by TD learning, and updates the policy parameters θ by following the natural gradient in the direction of $\mathbf{w}$. Pseudocode for EQNAC is shown in Algorithm 4.2. Again, for an effectively stationary policy the TD learning procedure is linear in ϕ^θ and therefore avoids the divergence issues encountered by energy-based Sarsa. Unlike ENATDAC, this algorithm does not require repeated sampling of actions to estimate the expectations. Furthermore, it only maintains two sets of parameters θ and $\mathbf{w}$, and does not require a state feature vector to be defined.

Algorithm 4.2: EQNAC

```
Input: $\hat{J}_0, \mathbf{w}_0, \theta_0, s_0$
$a_0 \sim \pi^{\theta_0}(\cdot; s_0)$
for $t = 0, 1, 2, \ldots$ do
    $r_{t+1} = R(s_t, a_t)$
    $s_{t+1} \sim P(\cdot | s_t, a_t)$
    $a_{t+1} \sim \pi^{\theta_t}(\cdot; s_t)$
    $\hat{J}_{t+1} = (1 - \zeta_t)\hat{J}_t + \zeta_t r_{t+1}$
    $\delta_t = r_{t+1} - \hat{J}_{t+1} +$
        $\phi^{\theta_t}(s_{t+1}, a_{t+1})^\top \mathbf{w}_t - \phi^{\theta_t}(s_t, a_t)^\top \mathbf{w}_t$
    $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \delta_t \phi^{\theta_t}(s_t, a_t)$
    $\theta_{t+1} = \theta_t + \beta_t \mathbf{w}_{t+1}$
end for
```

4. Experiments

We evaluate our two algorithms, comparing them against ESARSA on three tasks below. As an additional comparison, we also considered a neural network (NN) with stochastic output units and the same structure as the RBM except that the hidden units were deterministic instead of stochastic. The NN was trained with NATDAC (details in supplemental material). The learning algorithms are sensitive to the choice of learning parameters. For all tasks and algorithms we therefore perform a grid search over different settings of the relevant parameters, and report results for the best setting.[1] In a cross-validation style setup, we first choose the parameters based on a preliminary parameter sweep, then fix the parameters and perform a final run for which we report the results. In all cases, we learn several policies with the same parameter settings but different random initializations. We obtain learning curves by evaluating each policy at regular intervals during learning, performing several hundred repeats of each task.[2] We use exact sampling for all algorithms in the stochastic policy and blocker tasks, and block Gibbs sampling for the octopus arm task. Note that our algorithms use an average reward formulation; whereas ESARSA uses a discounted reward formulation. For ESARSA on the blocker and octopus arm tasks γ is set to 1; for the stochastic policy task $\gamma = 0$.

The **stochastic policy** task [Sallans, 2002] is an example where a stochastic policy is required to deal with state aliasing, and serves to demonstrate that RBMs can model stochastic policies with high-dimensional action spaces. The action space consists of N binary variables and there are K *unobserved* states. Each of the unobserved states is associated with one particular configuration of the action variables. If the environment is in state k and the associated action is performed a reward of 10 is received and the environment

[1] For ESARSA we consider initial learning rate, speed of decay of the learning rate, decay of the exploration temperature T (starting at 1), and variance of the Gaussian distribution used to randomly initialize the policy parameters at the beginning of learning. For the policy gradient algorithms we consider α_0, β_0, rate of decay for α, β, and scale of random initialization. We use $\zeta = \alpha$ in all experiments.

[2] 500 actions for stochastic policy; 1000 epochs for blocker task; 100 epochs for octopus arm.

	N	K	prototypical actions
task 1	5	2	(0 0 1 0 1) (0 1 0 1 1)
task 2	6	3	(1 1 0 0 0 0) (0 0 1 1 0 0) (0 0 0 0 1 1)
task 3	8	3	(0 0 1 0 1 0 1 1) (0 1 0 1 1 1 0 0) (1 1 1 0 1 0 0 0)

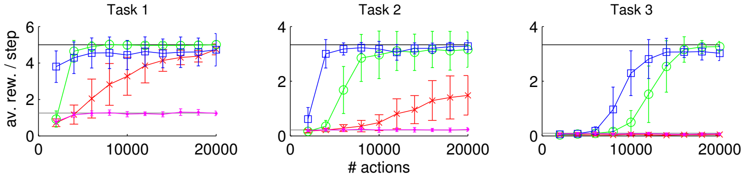
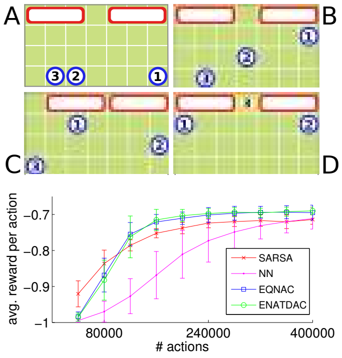


Figure 4.1: Stochastic policy task: The table shows the task-parameters for the three versions of the task. Results are shown for ENATDAC (green circles), EQNAC (blue squares), SARSA (red crosses), and NN (magenta). Black lines show average reward for optimal policy; light gray lines show rewards with independent actions.

Figure 4.2: Blocker task: The *top* plot illustrates several steps in a blocker game (adapted from Sallans and Hinton 2004). To win, the agents need to cooperate. Here, agents 1,2 force the blockers to split so that agent 3 can enter the end zone in the middle. The *bottom* plot shows, for each algorithm, mean and standard deviation of the average reward per step of the learned policies for 20 restarts with different random initialziations (see Fig. B.3 in the supplemental material for individual traces).



transitions into state $k+1$ (or into state 1 from K). Otherwise the state is unchanged and no reward is provided. Since the agent does not observe the environment state the optimal policy is to choose the K "good" action configurations with equal probability, achieving an average reward of $\frac{10}{K}$. A deterministic policy will get zero reward since it will get stuck in one of the states. Further, it is necessary to model *dependencies* between action variables: for independently drawn action variables the probability of generating a "good" configuration is very low. We consider three versions of the task with $N = 5, 6, 8$, and $K = 2, 3, 3$ (cf. Fig. 4.1). The first version was considered in Sallans [2002]. We used an RBM with 1 hidden unit for task 1 and 2 hidden units for tasks 2 and 3. Training was performed for 20000 actions. Results are shown in Fig. 4.1. The NN models each action dimension independently and therefore performs very poorly (see supplemental material for further discussion). In contrast, the RBM learns to represent these correlations via stochastic latent variables and performs very effectively. When training the RBM, ESARSA is only able to find the optimal policy for task 1, whereas EQNAC and ENATDAC find optimal policies in all cases.[3]

The **blocker task** (Sallans 2002; Sallans and Hinton 2004) is a multi-agent task in which agents have to reach the end zone at the top of a playing field while blockers try to stop

[3] Although only the first 20000 actions are shown we allowed 50000 actions for ESARSA training. For task 2 additional training leads to a small improvement relative to the performance after 20000 actions but not for task 3 (see also Fig. B.1 in supplemental material).

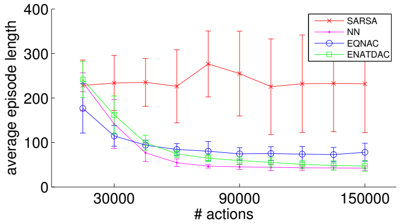


Figure 4.3: Octopus arm: Average number of steps required to hit the target. For each algorithm mean and std.-dev. for 10 restarts with different random initializations are shown. Figs. B.5,B.6 in the suppl. material show individual traces.

them. In each iteration of the game, each agent moves by one step in one of four directions, while the blockers behave in a deterministic manner, moving left or right to block the agents. We used a 7 × 4 board with 3 agents and 2 blockers as illustrated in Fig. 4.2. Blockers have a limited range of responsibility, with blocker 1 responsible for columns 1-4 and blocker 2 for columns 4-7. The state space consists of locations for each agent and for blocker (using 1-of-N encoding for each). A game is started by randomly placing agents at the bottom and the blockers at the top. It is played for 40 steps or until one of the agents reaches the end zone. A reward of -1 is given for each step prior to reaching the end zone, and a reward of 1 when one of the agents succeeds and the game is ended. We used RBMs with 16 hidden units, and learning was performed for 400000 actions. Results are shown in Fig. 4.2. EQNAC and ENATDAC consistently achieved good results. ESARSA and NN generally converged more slowly and found lower quality policies.

In this task we also experimented with several different state feature vectors $f(s)$, as required by ENATDAC (but not the other algorithms). We tested: randomly generated binary features (as e.g. Bhatnagar et al., 2009), choosing the dimensionality D of $f(s)$ to be lower than the number of states (which is 87808; we experimented with $D = 400, 1000$); raw feature encoding based on the state vector s directly; and the "energy based" feature vector described in section 3.1. We found that a poor choice of $f(s)$ can significantly reduce the asymptotic performance. The energy-based features performed very well, only equalled by large sets of carefully scaled random features.

Finally, we tested our algorithms on an **octopus arm** [Engel et al., 2005] task. The aim is to learn to control a simulated octopus arm to hit a target. The arm consists of C compartments and is attached to a rotating base. There are $8C + 2$ continuous state variables (x,y position/velocity of the nodes along the upper/lower side of the arm; angular position/velocity of the base) and $3C + 2$ action variables that control three muscles (dorsal, transversal, central) in each compartment as well as the clockwise and counter-clockwise rotation of the base. Previous work [Engel et al., 2005] simplified the high-dimensional action space using 6 "macro-actions" corresponding to particular patterns of muscle activations. Here, we do not make any such simplification; patterns of muscle activations are learnt by latent variables. Each muscle output was restricted to a binary activation. Below we present results for an arm with $C = 4$ compartments. The goal is to strike the target with any part of the arm. To direct exploration we provide a shaping reward at each time step: if the minimum distance between arm and target is decreased then a shaping reward of $+1$ is provided; if the distance is increased the reward is -1. The final reward for hitting the target is $+50$. An episode ends when the target is hit or after 300 steps.

We learned RBM-based policies with 32 stochastic hidden units, and an NN-based policy with 32 deterministic hidden units. We used energy-based features for (E)NATDAC as these are directly applicable in continuous-valued state spaces. Results for the octopus arm are shown in Fig. 4.3. The policy-gradient approaches performed significantly better in this experiment. In particular, the RBM trained by ENATDAC, and the NN trained by NATDAC, both learnt good policies that were consistently able to hit the target. Some ENATDAC runs were trapped in suboptimal maxima where the arm moves only in one direction, thus taking a longer time to reach the target on some episodes (see Figs. B.5, B.6 in the supplemental material). The policies learned by EQNAC are slightly worse but the arm still learns to find and hit the target in the majority of cases. ESARSA performed relatively poorly. A video demonstration of the learnt octopus arm policies is available at: http://www.gatsby.ucl.ac.uk/~nheess/papers/ebrl/

5. Discussion

In our most challenging task, an energy-based policy was able to represent and learn, using ENATDAC, an effective control policy for a high-dimensional octopus arm. Unlike prior work, the latent variables *automatically* learnt to represent useful patterns of activations. In this task, a stochastic NN with deterministic hidden nodes was also able to solve the problem. This is not entirely unexpected as here a deterministic final policy is likely to be sufficient, while learning is simpler in the NN as it allows direct sampling of actions and the exact calculation of ψ^θ and $f(s)$. However, as demonstrated by the stochastic policy task, there are many domains in which partial observability, or an adversary, entails the need for highly structured stochastic policies that cannot be encoded by deterministic hidden nodes.

In our experiments the ESARSA algorithm did not achieve satisfactory results in the larger stochastic policy tasks, and also performed relatively poorly on the blocker and octopus arm tasks. The poor performance in some runs, and sensitivity to initial values, suggests that ESARSA is indeed suffering from divergence problems, presumably caused by the use of non-linear temporal-difference learning. In contrast, ENATDAC and EQNAC converged more robustly to a satisfactory solution in all tasks, suggesting that policy gradient algorithms may be more appropriate than value-based ones in this setting.

Our first algorithm, ENATDAC, is guaranteed to converge to a policy that achieves an average-reward that is close to local maximum. The algorithm requires the policy to be sampled many times: to select an action, to estimate the compatible features, and to construct the energy-based feature vector $f(s)$. However, we found empirically that a very small number of samples was sufficient to achieve good results: We used $K, L = 1$ in our experiments (cf. line 3 of Algorithm 4.1) except for the octopus arm where $K = L = 5$. Although the samples should be independent to ensure convergence, in practice the performance was unaltered by reusing the same samples.

Our second algorithm, EQNAC, uses a simple approximation to the advantage function. This algorithm does not require additional samples of the policy, beyond selecting the next action, making it more computationally efficient. It is also simpler than ENATDAC, using two parameter vectors rather than three. In practice, both algorithms performed well, achieving very similar performance on the blocker task, but with ENATDAC performing slightly better on the stochastic policy and more consistently on the octopus arm tasks.

In conclusion, reinforcement learning is particularly challenging in high-dimensional state and action spaces. Energy-based policies provide a promising architecture for addressing this challenge. We have demonstrated that prior work on value-based reinforcement learning, using the ESARSA algorithm, is often unsatisfactory when using highly non-linear policies such as an RBM. We have introduced two novel algorithms, based on energy-based policy gradient methods, that are more robust and effective than ESARSA. ENATDAC is guaranteed to converge to a near-local maximum in average reward; EQNAC is also robust in practice, and requires less computation. Both algorithms outperformed ESARSA on several high dimensional tasks.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 270327, and from the Gatsby Charitable foundation.

References

J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *JAIR*, 15, 2001.

S. Bhatnagar, R. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor-critic algorithms. *Automatica*, 45: 2471–2482, 2009.

S. Elfwing, M. Otsuka, E. Uchibe, and K. Doya. Free-energy based reinforcement learning for vision-based navigation with high-dimensional sensory inputs. In *ICONIP*, 2010.

Y. Engel, P. Szabó, and D. Volkinshtein. Learning to control an octopus arm with gaussian process temporal difference methods. In *NIPS*, 2005.

Y. Freund and D. Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. Technical Report UCSC-CRL-94-25, University of California, Santa Cruz, 1994.

S. Kakade. A natural policy gradient. In *NIPS*, 2001.

H. Larochelle and Y. Bengio. Classification using discriminative restricted Boltzmann machines. In *ICML*, 2008.

Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F.-J. Huang. A tutorial on energy-based learning. *Predicting Structured Data*. MIT Press, 2006.

H. Maei, C. Szepesvari, S. Bhatnagar, D. Precup, D. Silver, and R. Sutton. Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation. In *NIPS*, 2009.

R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order Boltzmann machines. *Neural Computation*, 22, 2010.

V. Mnih, H. Larochelle, and G. E. Hinton. Conditional restricted Boltzmann machines for structured output prediction. In *UAI*, 2011.

M. Otsuka, J. Yoshimoto, and K. Doya. Free-energy-based reinforcement learning in a partially observable environment. In *ESANN*, 2010.

J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71, 2008.

R. Salakhutdinov, A. Mnih, and G. E. Hinton. Restricted Boltzmann machines for collaborative filtering. In *ICML*, 2007.

B. Sallans. *Reinforcement Learning for Factored Markov Decision Processes*. PhD thesis, Department of Computer Science, University of Toronto, 2002.

B. Sallans and G. E. Hinton. Reinforcement learning with factored states and actions. *JMLR*, 5, 2004.

R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

G. Taylor and G. E. Hinton. Factored conditional restricted Boltzmann machines for modeling motion style. In *ICML*, 2009.

J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42, 1997.

M. Welling, M. Rosen-Zvi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In *NIPS*, 2004.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, 1992.

Appendix A. Neural Network

In order to provide a comparison with an alternative structured parameterization of a stochastic policy we considered a 2-layer neural network (NN). The network's structure is very similar to the RBM but it has deterministic hidden units and a feedforward archictecture.

For binary or continuous valued states s and binary action units a the state-conditional action distribution $\pi^\theta(a; s)$ defined by the NN is given as follows:

$$\pi^\theta(a; s) = \prod_i p(a_i|s), \tag{23}$$

$$p(a_i = 1|s) = \sigma(W_a h(s) + b), \tag{24}$$

$$h(s) = \sigma(W_s s + c), \tag{25}$$

where $h(s)$ is a vector of (deterministic) hidden units with activation between 0 and 1, and $\sigma(y) = 1/(1 + \exp(-y))$ is the sigmoid function that is applied element-wise. Weight matrices W_s, W_a, and bias vectors c, and b are the parameters of the network. The above architecture is easily generalized to discrete stochastic output units with more than two states.

This NN has the same number of parameters as the RBM and is able to model nonlinear dependence of the actions on the states. Unlike in a RBM, however, the stochastic output units are conditionally independent given the state. The network therefore fails to model correlations between the action variables. In particular, it is unable to represent multi-modal state-conditional action distributions as required for the the stochastic policy task.

In our experiments we trained the NN using NATDAC with the "energy-based" state features $f(s) = \mathrm{E}_a^\theta[\phi^\theta(s, a)]$ described in section 3.1 of the main text. Due to the independence of the action units (cf. eq. 23 above) for the NN the gradient of the log-policy $\nabla_\theta \log \pi^\theta(a; s) = \psi^\theta(s, a)$ and the energy-based state features $f(s)$ can be computed analytically and the approximations described in section 3.1 of the main text are not required.

It is worth noting that it is, in principle, possible to make the hidden units stochastic while maintaining a feed-forward architecture[4]. This would introduce correlations between the action variables. At the same time, however, it would also render the exact computation of e.g. ψ intractable (except in simple cases) due to the need to integrate – or sum – out the hidden variables.

Appendix B. Additional Experiments and Results

B.1. Stochastic policy task

Note that in the stochastic policy task there is no observed state. In this case the NN reduces to a set of biases that model the activation of each action unit independently. The RBM in contrast can model correlations between the action units and hence represent the multi-modal distribution over binary action vectors required for this task.

For instance, when the NN is applied to stochastic policy task 1 with $K = 2$ unobserved states the first and last action units a_1 and a_5 are effectively set deterministically to 1 and 0 respectively, but the remaining action units a_2, a_3, a_4 will have $p(a_i = 1) = 0.5$ for $i = 2, 3, 4$ after learning. Hence, the probability of drawing one of the two "good" actions is only $\frac{1}{8}$ instead of $\frac{1}{2}$ for the RBM (which is why the NN achieves a reward of only 10/8 instead of $10/K$.

As explained in footnote 3 in the main text we allowed 50000 actions for learning for SARSA but only the performance after 20000 actions is shown in Fig. 1 in the main text (in line with the number of actions available to the other algorithms). Figure B.1 shows the full learning curves for SARSA for the stochastic policy task.

B.2. Grid world

The grid world task requires an agent to learn how to navigate to a final state. At the beginning of each trial the agent is placed at a random position in the environment. At each step he chooses one of the four actions

[4] While the RBM corresponds to an *undirected* graphical model, a feedforward NN with stochastic hidden units would correspond to a *directed* graphical model.

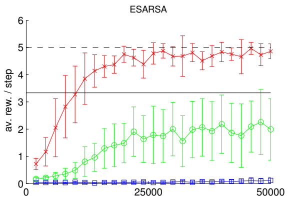


Figure B.1: Stochastic policy task: full learning curves for SARSA for all three versions of the task (red: task 1; green: task 2; blue: task 3). Same data as in Fig. 1 in the main text but all 50000 actions are shown. The dashed black line indicates the reward achieved by the optimal policy for task 1, the solid black line the optimal reward for tasks 2 and 3.

(N,S,E,W). The reward is -1 for all steps except if the agent reaches the end state, in which case it is 2. An episode ends when the agent reaches the end state or after a maximum of 20 actions. If an action cannot be performed, the agent remains at its current position. States and actions both use a 1-of-N encoding, and we also use a 1-of-N encoding for the state-value function approximation in ENATDAC, allowing it to represent the state-value function exactly. We used RBMs with 3 hidden units, and learning was performed for 40000 actions. The particular environment used in the experiments and the results obtained are shown in Fig. B.2. ENATDAC and EQNAC both find an optimal policy quickly and robustly, while ESARSA takes longer to converge to a policy, the policy found is usually not optimal, and the policy found is strongly dependent on the initialization of the policy parameters. Also, we found ESARSA to be much more sensitive to the settings of the learning parameters than our algorithms.

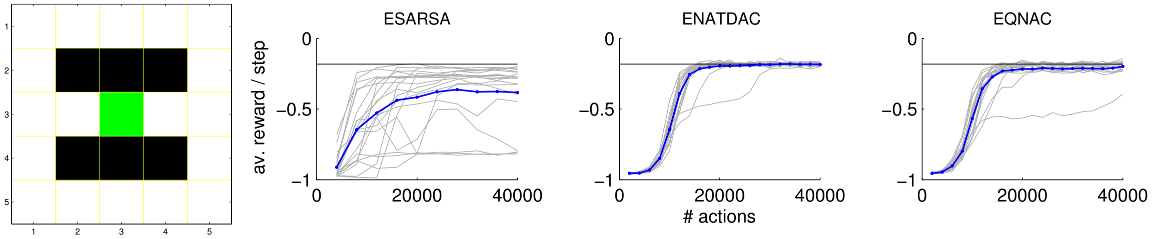
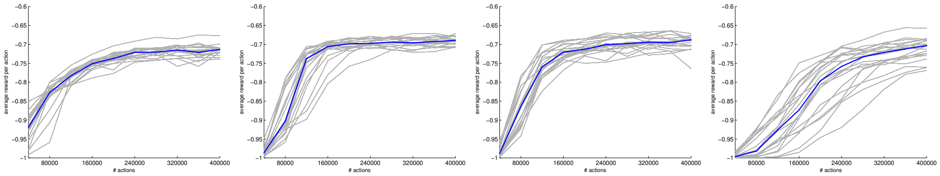


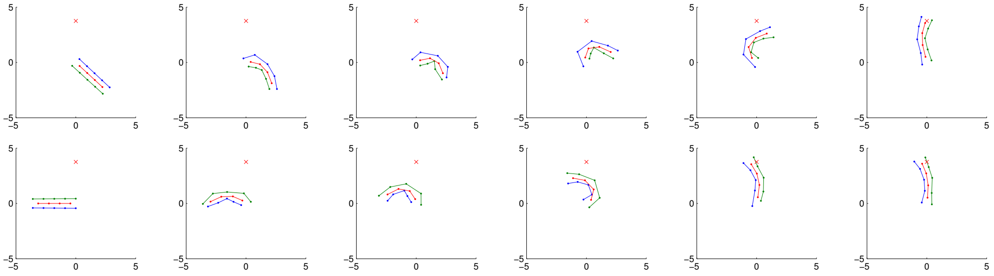
Figure B.2: 5 × 5 grid world: white squares are fields the agent can move to; black squares are walls; the end state is marked green. Results shown are for 20 runs. Gray curves indicate results for individual runs, blue curves show average.

B.3. Blocker Task



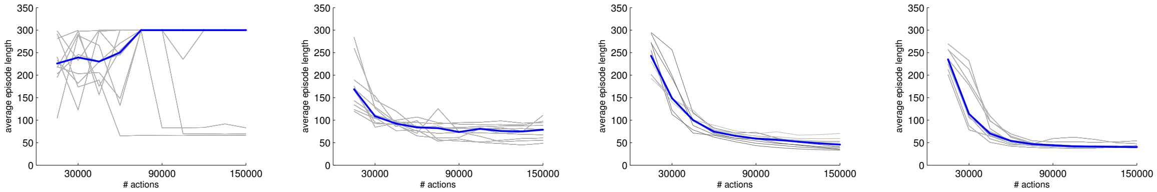
ESARSA ENATDAC EQNAC Neural Network

Figure B.3: Blocker task: Same data as in Fig. 4.2 in the main text but showing results for the 20 restarts with different random initializations individually for each algorithm: Gray lines show individual runs; thick blue line shows the median.



initial 5 steps 10 steps 20 steps 30 steps final

Figure B.4: Octopus arm hitting the target: Each row corresponds to a different run, starting from two different initial positions. The arm is shown in its initial position, after 5, 10, 20, and 30 steps, and after having hit the target. The target is shown by the red cross.



ESARSA EQNAC ENATDAC Neural Network

Figure B.5: Octopus arm: Average number of steps required to hit the target. Same data as in Fig. 4.3 in the main text but results are shown separately for the 10 restarts with different random initializations. The policies are evaluated every 15000 steps with 100 episodes. Gray lines correspond to individual runs, the median is shown in blue. For ENATDAC runs that got trapped in a local optimum (as explained in the text) are shown in light gray, others in dark gray.

B.4. Octopus Arm

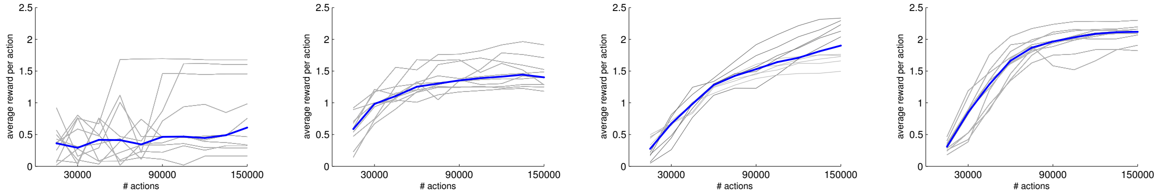
For the simulations we used the octopus arm simulator that is part of the RL Glue package[5]. We considered an arm with $C = 4$ compartments. Thus, the continuous state space was 34 dimensional, the binary action space 14 dimensional. (On the accompanying website[6] we also show results for an arm with $C = 6$ compartments, i.e. with a 50 dimensional state space, and a 20 dimensional action space.) Note that continuous valued states do note pose a problem for binary *conditional* RBMs.

At the beginning of each episode the arm is initialized at one of 4 random positions and the goal is then to hit the target as quickly as possible with any part of the arm. Fig. B.4 shows frames of two movies of the arm controlled by a learned policy hitting the target from two different initial positions. See the accompanying website for more results.

In Fig. B.6 we show as an alternative visualization of the results the average reward per step achieved with the learned policies. As pointed out in the main text ENATDAC occasionally got trapped in local optima, learning policies in which the arm rotated always in one direction independently of its initial position. With these policies the arm still hits the target reliably but requires a larger number of steps from some initial positions. This leads to a lower average reward per step at the end of learning. ENATDAC runs in which the arm learned suboptimal policies are shown in light gray in Figs. B.5, B.6.

[5] `http://glue.rl-community.org/wiki/Main\_Page`

[6] `http://www.gatsby.ucl.ac.uk/~nheess/papers/ebrl/`



ESARSA EQNAC ENATDAC Neural Network

Figure B.6: Octopus arm: Alternative visualization of the results in Fig. B.5 (Fig. 4.3 in the main text). Average reward per step of the learned policies as a function of actions performed during learning. Results are shown for 10 restarts with different random initializations. Same format as in Fig. B.5