

6 Supplementary Material

6.1 Gradient Derivation

For ease of reference in this section we give a brief outline of the derivation for the derivative of (1). As in the rest of the paper we focus on the case of an infinite planning horizon with discounted rewards, where other frameworks follow similarly. The first point to note is that for any $t \in \mathbb{N}$ we have the following identity, often referred to as the ‘log-trick’,

$$\nabla_{\mathbf{w}} p(\mathbf{z}_{1:t}; \mathbf{w}) = p(\mathbf{z}_{1:t}; \mathbf{w}) \nabla_{\mathbf{w}} \log p(\mathbf{z}_{1:t}; \mathbf{w}),$$

where the usual limit arguments are made in the case $p(\mathbf{z}_{1:t}; \mathbf{w}) = 0$. Upon interchanging the order of integration and differentiation the gradient takes the form

$$\nabla_{\mathbf{w}} U(\mathbf{w}) = \sum_{t=1}^{\infty} \mathbb{E}_{p(\mathbf{z}_{1:t}; \mathbf{w})} \left[\gamma^{t-1} R(\mathbf{z}_t) \nabla_{\mathbf{w}} \log p(\mathbf{z}_{1:t}; \mathbf{w}) \right].$$

Due to the Markovian structure of the trajectory distribution (2) this derivative can be written in the equivalent form

$$\begin{aligned} \nabla_{\mathbf{w}} U(\mathbf{w}) &= \sum_{t=1}^{\infty} \sum_{\tau=1}^t \mathbb{E}_{p(\mathbf{z}_{\tau}, \mathbf{z}_t; \mathbf{w})} \left[\gamma^{t-1} R(\mathbf{z}_t) \nabla_{\mathbf{w}} \log p(\mathbf{a}_{\tau} | \mathbf{s}_{\tau}; \mathbf{w}) \right], \\ &= \sum_{\tau=1}^{\infty} \sum_{t=\tau}^{\infty} \mathbb{E}_{p(\mathbf{z}_{\tau}, \mathbf{z}_t; \mathbf{w})} \left[\gamma^{t-1} R(\mathbf{z}_t) \nabla_{\mathbf{w}} \log p(\mathbf{a}_{\tau} | \mathbf{s}_{\tau}; \mathbf{w}) \right], \end{aligned}$$

where the second line follows from the first through an interchange of the summations. The chain structure of the trajectory distribution allows the expectation over the marginals of the two time-points of the trajectory distribution to be written as follows

$$\nabla_{\mathbf{w}} U(\mathbf{w}) = \sum_{\tau=1}^{\infty} \mathbb{E}_{p_{\tau}(\mathbf{z}; \mathbf{w})} \left[\sum_{t=\tau}^{\infty} \mathbb{E}_{p_t(\mathbf{z}'; \mathbf{w})} \left[\gamma^{t-1} R(\mathbf{z}') | \mathbf{z}_{\tau} = \mathbf{z} \right] \nabla_{\mathbf{w}} \log p(\mathbf{a} | \mathbf{s}; \mathbf{w}) \right], \quad (15)$$

where we have used the notation $p_{\tau}(\mathbf{z}; \mathbf{w}) \equiv p(\mathbf{z}_{\tau} = \mathbf{z}; \mathbf{w})$, for $\tau \in \mathbb{N}$. The summation over the inner expectation in (15) can be seen to be equal to the state-action value function scaled by $\gamma^{\tau-1}$, *i.e.*

$$\gamma^{\tau-1} Q(\mathbf{z}; \mathbf{w}) = \sum_{t=\tau}^{\infty} \mathbb{E}_{p_t(\mathbf{z}'; \mathbf{w})} \left[\gamma^{t-1} R(\mathbf{z}') | \mathbf{z}_{\tau} = \mathbf{z} \right].$$

Inserting this form for this inner expectation into (15) gives

$$\begin{aligned} \nabla_{\mathbf{w}} U(\mathbf{w}) &= \sum_{\tau=1}^{\infty} \mathbb{E}_{p_{\tau}(\mathbf{z}; \mathbf{w})} \left[\gamma^{\tau-1} Q(\mathbf{z}; \mathbf{w}) \nabla_{\mathbf{w}} \log p(\mathbf{a} | \mathbf{s}; \mathbf{w}) \right], \\ &= \mathbb{E}_{p_{\gamma}(\mathbf{z}; \mathbf{w}) Q(\mathbf{z}; \mathbf{w})} \left[\nabla_{\mathbf{w}} \log p(\mathbf{a} | \mathbf{s}; \mathbf{w}) \right], \end{aligned}$$

where the second line follows from the definition of $p_{\gamma}(\mathbf{z}; \mathbf{w})$. This completes the derivation of (4). This derivation is different, but equivalent, to the standard derivation in [27].

6.2 Hessian Derivation

Following similar calculations to those used section(6.1) it is simple to see that the Hessian takes the form

$$\mathcal{H}(\mathbf{w}) = \sum_{t=1}^{\infty} \mathbb{E}_{p(\mathbf{z}_{1:t}; \mathbf{w})} \left[\gamma^{t-1} R(\mathbf{z}_t) \left(\nabla_{\mathbf{w}} \log p(\mathbf{z}_{1:t}; \mathbf{w}) \nabla_{\mathbf{w}}^T \log p(\mathbf{z}_{1:t}; \mathbf{w}) + \nabla_{\mathbf{w}} \nabla_{\mathbf{w}}^T \log p(\mathbf{z}_{1:t}; \mathbf{w}) \right) \right],$$

where the first term is obtained by a second application of the ‘log-trick’. This is the form of the Hessian given in (5). The form of $\mathcal{H}_2(\mathbf{w})$ given in (9) follows from analogous calculations to those performed in section(6.1) of the supplementary material.

6.3 Expectation Maximisation Derivation

For ease of reference in this section we give a brief derivation of the application of Expectation Maximisation to Markov Decision Processes. As in the rest of the paper we focus on the case of an infinite planning horizon with discounted rewards, where other frameworks follow similarly. A quantity of central importance in this derivation is the following non-negative function

$$\tilde{p}(\mathbf{z}_{1:t}, t; \mathbf{w}) = \gamma^{t-1} R(\mathbf{z}_t) p(\mathbf{z}_{1:t}; \mathbf{w}).$$

This function is non-negative because the reward function is considered as non-negative. Note that this function is trans-dimensional, *i.e.* it is a function of both $t \in \mathbb{N}$ and the state-action variables up until the t^{th} time-point. It can be seen from (1) and (2) that the ‘normalisation constant’ of $\tilde{p}(\mathbf{z}_{1:t}, t; \mathbf{w})$ is equal to $U(\mathbf{w})$, *i.e.* $U(\mathbf{w}) = \sum_{t=1}^{\infty} \sum_{\mathbf{z}_{1:t}} \tilde{p}(\mathbf{z}_{1:t}, t; \mathbf{w})$. We denote the normalised version of this function by $\hat{p}(\mathbf{z}_{1:t}, t; \mathbf{w})$.

To obtain the lower-bound on the log-objective we note that the Kullback-Leibler divergence between any two distributions is always non-negative and so calculating the Kullback-Leibler divergence between the variational distribution, $q(\mathbf{z}_{1:t}, t)$, and $\hat{p}(\mathbf{z}_{1:t}, t; \mathbf{w})$ gives

$$KL(q||\hat{p}) = \log U(\mathbf{w}) - \mathcal{H}_{\text{entropy}}(q(\mathbf{z}_{1:t}, t)) - \mathbb{E}_{q(\mathbf{z}_{1:t}, t)} \left[\log \tilde{p}(\mathbf{z}_{1:t}, t; \mathbf{w}) \right] \geq 0.$$

The lower-bound (10) is now immediately obtained from the definition of $\tilde{p}(\mathbf{z}_{1:t}, t; \mathbf{w})$.

An EM-algorithm is obtained through coordinate-wise optimisation of the bound (10) *w.r.t.* the variational distribution (the E-step) and the policy parameters (the M-step). In the E-step the lower-bound (10) is optimised when the Kullback-Leibler divergence is zero, namely when $q(\mathbf{z}_{1:t}, t) \equiv \hat{p}(\mathbf{z}_{1:t}, t; \mathbf{w}_k)$, where \mathbf{w}_k are the current policy parameters. In the E-step the lower-bound is optimised *w.r.t.* \mathbf{w} , which is equivalent to optimising the quantity

$$\mathbb{E}_{\tilde{p}(\mathbf{z}_{1:t}, t; \mathbf{w}_k)} \left[\log \tilde{p}(\mathbf{z}_{1:t}, t; \mathbf{w}) \right] = \mathbb{E}_{\tilde{p}(\mathbf{z}_{1:t}, t; \mathbf{w}_k)} \left[\sum_{\tau=1}^t \log \pi(\mathbf{a}_{\tau} | \mathbf{s}_{\tau}; \mathbf{w}) \right] + \text{terms independent of } \mathbf{w}.$$

Using the definition of $\tilde{p}(\mathbf{z}_{1:t}, t; \mathbf{w}_k)$ we have

$$\mathbb{E}_{\tilde{p}(\mathbf{z}_{1:t}, t; \mathbf{w}_k)} \left[\sum_{\tau=1}^t \log \pi(\mathbf{a}_{\tau} | \mathbf{s}_{\tau}; \mathbf{w}) \right] = \sum_{t=1}^{\infty} \sum_{\tau=1}^t \mathbb{E}_{p(\mathbf{z}_{\tau}, \mathbf{z}_t; \mathbf{w}_k)} \left[\gamma^{t-1} R(\mathbf{z}_t) \log \pi(\mathbf{a}_{\tau} | \mathbf{s}_{\tau}; \mathbf{w}) \right],$$

so that using the same manipulations as those used in section(6.1) of the supplementary material we have that the E-step is equivalent to maximising the function

$$\mathcal{Q}(\mathbf{w}, \mathbf{w}_k) = \mathbb{E}_{p_{\gamma}(\mathbf{z}; \mathbf{w}_k) Q(\mathbf{z}; \mathbf{w}_k)} \left[\log \pi(\mathbf{a} | \mathbf{s}; \mathbf{w}) \right],$$

w.r.t. to the first parameter, \mathbf{w} . This completes the derivation of the EM-algorithm for Markov Decision Processes with an infinite planning horizon and discounted rewards.

6.4 Affine Invariance of Approximate Newton Method

To show that the use of $\mathcal{H}_2(\mathbf{w})$ in place of the true Hessian maintains the linear (affine) invariance of the Newton method we use the following formulae

$$\nabla_{\mathbf{w}} \tilde{f}(\mathbf{w}) = \mathcal{T}^T \nabla_{\mathbf{v}} f(\mathbf{v}), \quad \nabla_{\mathbf{w}} \nabla_{\mathbf{w}}^T \tilde{f}(\mathbf{w}) = \mathcal{T}^T \nabla_{\mathbf{v}} \nabla_{\mathbf{v}}^T f(\mathbf{v}) \mathcal{T},$$

where f is some twice differentiable function of \mathbf{w} , $\tilde{f}(\mathbf{w}) = f(\mathcal{T}\mathbf{w})$ and $\mathbf{v} = \mathcal{T}\mathbf{w}$. Using these formulae we have the following two identities

$$\begin{aligned} \nabla_{\mathbf{w}} \log p(\mathbf{a} | \mathbf{s}; \mathcal{T}\mathbf{w}) &= \mathcal{T}^T \nabla_{\mathbf{v}} \log p(\mathbf{a} | \mathbf{s}; \mathbf{v}), \\ \nabla_{\mathbf{w}} \nabla_{\mathbf{w}}^T \log p(\mathbf{a} | \mathbf{s}; \mathcal{T}\mathbf{w}) &= \mathcal{T}^T \nabla_{\mathbf{v}} \nabla_{\mathbf{v}}^T \log p(\mathbf{a} | \mathbf{s}; \mathbf{v}) \mathcal{T}, \end{aligned}$$

which hold for each $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$. Defining $\tilde{U}(\mathbf{w}) = U(\mathcal{T}\mathbf{w})$, we have $\nabla_{\mathbf{w}} \tilde{U}(\mathbf{w}) = \mathcal{T}^T U(\mathbf{v})$. Following calculations almost identical to those in section(6.2) it can be shown that $\tilde{\mathcal{H}}_2(\mathbf{w})$ takes the form

$$\tilde{\mathcal{H}}_2(\mathbf{w}) = \mathbb{E}_{p_{\gamma}(\mathbf{z}; \mathcal{T}\mathbf{w}) Q(\mathbf{z}; \mathcal{T}\mathbf{w})} \left[\nabla_{\mathbf{w}} \nabla_{\mathbf{w}}^T \log p(\mathbf{a} | \mathbf{s}; \mathcal{T}\mathbf{w}) \right],$$

which gives the following

$$\begin{aligned}
\tilde{\mathcal{H}}_2(\mathbf{w}) &= \mathbb{E}_{p_\gamma(\mathbf{z}; \mathcal{T}\mathbf{w})Q(\mathbf{z}; \mathcal{T}\mathbf{w})} \left[\nabla_{\mathbf{w}} \nabla_{\mathbf{w}}^T \log p(\mathbf{a}|\mathbf{s}; \mathcal{T}\mathbf{w}) \right], \\
&= \mathbb{E}_{p_\gamma(\mathbf{z}; \mathcal{T}\mathbf{w})Q(\mathbf{z}; \mathcal{T}\mathbf{w})} \left[\mathcal{T}^T \nabla_{\mathbf{v}} \nabla_{\mathbf{v}}^T \log p(\mathbf{a}|\mathbf{s}; \mathbf{v}) \mathcal{T} \right], \\
&= \mathcal{T}^T \mathbb{E}_{p_\gamma(\mathbf{z}; \mathcal{T}\mathbf{w})Q(\mathbf{z}; \mathcal{T}\mathbf{w})} \left[\nabla_{\mathbf{v}} \nabla_{\mathbf{v}}^T \log p(\mathbf{a}|\mathbf{s}; \mathbf{v}) \right] \mathcal{T}, \\
&= \mathcal{T}^T \mathcal{H}_2(\mathbf{v}) \mathcal{T}.
\end{aligned}$$

Using these two expressions we have that the parameter updates, under the approximate Newton method, of the objective functions U and \tilde{U} are related as follows

$$\begin{aligned}
\mathbf{v}_{\text{new}} &= \mathbf{v} + \alpha \mathcal{H}_2(\mathbf{v})^{-1} \nabla_{\mathbf{v}} U(\mathbf{v}), \\
&= \mathcal{T}(\mathbf{w} + \alpha \tilde{\mathcal{H}}_2(\mathbf{w})^{-1} \nabla_{\mathbf{w}} \tilde{U}(\mathbf{w})),
\end{aligned}$$

where α is some step-size parameter. This shows that the approximate Newton method is affine invariant. The invariance of the diagonal approximate Newton method to rescaling of the parameters follows similarly.

6.5 Recurrent State Search Direction Evaluation for Approximate Newton Method

In [31] a sampling algorithm was provided for estimating the gradient of an infinite horizon MDP with average rewards. This algorithm makes use of a recurrent state, which we denote by \mathbf{s}^* . In algorithm(1) we detail a straightforward extension of this algorithm to the estimation of the approximate Hessian, $\mathcal{H}_2(\mathbf{w})$, in this MDP framework. The analogous algorithm for the estimation of the diagonal matrix, $\mathcal{D}_2(\mathbf{w})$, follows similarly. In algorithm(1) we make use of an *eligibility trace* for both the gradient and the approximate Hessian, which we denote by Φ^1 and Φ^2 respectively. The estimates (up to a positive scalar) of the gradient and the approximate Hessian are denoted by Δ^1 and Δ^2 respectively.

6.6 Tetris Experiment Specification

In this section we give a detailed specification of the procedure used for each of the algorithms in the tetris experiment. The same general procedure was used for all the algorithms considered in the experiments. We modelled the environment through with an infinite planning horizon with average rewards. The reward at each time-point is equal to the number of lines deleted. We used a recurrent state formulation [31] of the gradient of the average reward framework to perform the gradient evaluation. We used analogous versions of this recurrent state formulation for natural gradient ascent, the diagonal approximate Newton method and the full approximate Newton method. See section(6.5) of the supplementary material for more details. As in [16] we used the sample trajectories obtained during the gradient evaluation to estimate the Fisher information matrix. Irrespective of the policy a game of tetris is guaranteed to terminate after a finite number of turns, see *e.g.* [7]. As we are considering the average reward framework a new game starts when the previous one terminates. We used the empty board as a recurrent state where, because a new game starts with an empty board, this state is recurrent.¹ During each training iteration the (approximation of the) search direction was obtained by sampling 1000 games, where these games were sampled using the current policy parameters. Given the current approximate search direction the following basic line search method was used to obtain a step-size: For every step-size in a given finite set of step-sizes sample a set number of games and then return the step-size with the maximal score over these games. In practice, in order to reduce the susceptibility to random noise, we used the same simulator seed for each possible step-size in the set. To avoid over-fitting a different simulator seed was used during each training iteration. In this line search procedure we sampled 1000 games for each of the possible

¹This is actually an approximation because it doesn't take into account that the state is given by the configuration of the board and the current piece, so this particular 'recurrent state' ignores the current piece. Empirically we found that this approximation gave better results, presumably due to reduced variance in the estimands, and there is no reason to believe that it is unfairly biasing the comparison between the various parametric policy search methods.

Algorithm 1 Recurrent state sampling algorithm to estimate the search direction of the approximate Newton method when applied to a MDP with an infinite planning horizon with average rewards.

Sample a state from the initial state distribution:

$$\mathbf{s}_1 \sim p_1(\cdot).$$

for $t = 1, \dots, N$, for some $N \in \mathbb{N}$, **do**

Given the current state, sample an action from the policy:

$$\mathbf{a}_t \sim \pi(\cdot | \mathbf{s}_t; \mathbf{w}).$$

if $\mathbf{s}_t \neq \mathbf{s}^*$ **then**

Update the eligibility traces:

$$\Phi^1 \leftarrow \Phi^1 + \nabla_{\mathbf{w}} \log \pi(\mathbf{a}_t | \mathbf{s}_t; \mathbf{w}) \quad \Phi^2 \leftarrow \Phi^2 + \nabla_{\mathbf{w}} \nabla_{\mathbf{w}}^T \log \pi(\mathbf{a}_t | \mathbf{s}_t; \mathbf{w})$$

else

reset the eligibility traces:

$$\Phi^1 = \mathbf{0}, \quad \Phi^2 = \mathbf{0}.$$

end if

Update the estimates of the gradient and the approximate Hessian:

$$\Delta^1 \leftarrow \Delta^1 + R(\mathbf{a}_t, \mathbf{s}_t) \Phi^1, \quad \Delta^2 \leftarrow \Delta^2 + R(\mathbf{a}_t, \mathbf{s}_t) \Phi^2.$$

Sample state from the transition dynamics:

$$\mathbf{s}_{t+1} \sim p(\cdot | \mathbf{a}_t, \mathbf{s}_t).$$

end for

Return the estimated gradient and approximate Hessian, which up to a positive scaling are given by Δ^1 and Δ^2 respectively.

step-sizes. The same set of step-sizes was used in all of the different training algorithms considered in section(4), where we used the set

$$\{0.1, 0.5, 1.0, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0, 128.0\}.$$

To reduce the amount of noise in the results we used the same set of simulator seeds in the search direction evaluation for each of the algorithms considered in section(4). In particular, we generated a $n_{\text{experiments}} \times n_{\text{iterations}}$ matrix of simulator seeds, where $n_{\text{experiments}}$ was the number of repetitions of the experiment and $n_{\text{iterations}}$ was the number of training iterations in each experiment. We then used this one matrix in all of the different training algorithms, where the element in the j^{th} column and i^{th} row corresponds to the simulator seed used in the j^{th} training iteration of the i^{th} experiment. In a similar manner the set of simulator seeds used for the line search procedure was the same for all of the different training algorithms. Finally, to make the line search consistent among all of the different training algorithms the search direction was normalised and the resulting unit vector was the vector used in the line search procedure.

6.7 Linear System Specification

The N -link rigid robot arm manipulator is a standard continuous MDP model, consisting of an end effector connected to an N -linked rigid body [17]. A typical continuous control problem for such systems is to apply appropriate torque forces to the joints of the manipulator so as to move the end effector into a desired position. The state of the system is given by $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \in \mathbb{R}^N$, where \mathbf{q} , $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ denote the angles, velocities and accelerations of the joints respectively, while the control variables

are the torques applied to the joints $\tau \in \mathbb{R}^N$. The nonlinear state equations of the system are given by, see *e.g.* [17],

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\dot{\mathbf{q}}, \mathbf{q})\dot{\mathbf{q}} + g(\mathbf{q}) = \tau \quad (16)$$

where $M(\mathbf{q})$ is the inertia matrix, $C(\dot{\mathbf{q}}, \mathbf{q})$ denotes the Coriolis and centripetal forces and $g(\mathbf{q})$ is the gravitational force. While this system is highly nonlinear it is possible to define an appropriate control function $\hat{\tau}(\mathbf{q}, \dot{\mathbf{q}})$ that results in linear dynamics in a different state-action space. This process is called *feedback linearisation*, see *e.g.* [17], and in the case of an N -link rigid manipulator recasts the torque action space into the acceleration action space. This means that the state of the system is now given by \mathbf{q} and $\dot{\mathbf{q}}$, while the control is $\mathbf{a} = \ddot{\mathbf{q}}$. Ordinarily in such problems the reward would be a function of the generalised co-ordinates of the end effector, which results in a non-trivial reward function in terms of \mathbf{q} , $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$. While this reward function can be modelled as a mixture of Gaussians for simplicity we consider the simpler problem where the reward is a function of \mathbf{q} , $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ directly.

In the experiments we considered a 3-link rigid manipulator, which results in a 9-dimensional state-action space and a 22-dimensional policy. In the experiment we discretised the continuous time dynamics into time-steps of $\Delta_t = 0.1$ and considered a finite planning horizon of $H = 100$. The mean of the initial state distribution was set to zero. The elements of the policy parameters m and π_σ were initialised randomly from the interval $[-2, 2]$ and $[1, 2]$ respectively. The matrix K was initialised to be zero on inter-link entries, while intra-link entries were initialised using rejection sampling. We sampled the parameters for each link independently from the set $[-400, 40] \times [-50, 10]$ and rejected the sample if the corresponding link was unstable. In the reward function the desired angle of each joint was randomly sampled from the interval $[\pi/4, 3\pi/4]$. The covariance matrices of the initial state distribution and state transition dynamics were set to diagonals, where the diagonal elements were initialised randomly from the interval $[0, 0.05]$. The covariance matrix of the reward function was set to be a diagonal with all entries equal to 0.1.

We used the `minFunc`² optimisation library in all of the gradient-based algorithms. We found that both the line search algorithm and the step-size initialisation had an effect on the performance of all the algorithms. We therefore tried various combinations of these settings for each algorithm and selected the one that gave the best performance. We tried bracketing line search algorithms with: *step-size halving*; *quadratic/cubic interpolation from new function values*; *cubic interpolation from new function and gradient values*; *step-size doubling and bisection*; *cubic interpolation/extrapolation with function and gradient values*. We tried the following step-size initialisations: *quadratic initialization using previous function value and new function value/gradient*; *twice the previous step-size*. To handle situations where the initial policy parameterisation was in a ‘flat’ area of the parameter space far from any optima we set the function and point toleration of `minFunc` to zero for all algorithms. To handle the large number of training iterations required by steepest gradient ascent we increased the maximum number of function evaluations and training iterations to 10,000.

6.8 Non-Linear System Specification

In section we detail the implementation of the experiment on the synthetic two-dimensional non-linear MDP considered in [29]. The state-space of the problem is two-dimensional, $\mathbf{s} = (s^1, s^2)$, where s^1 is the agent’s position and s^2 is the agent’s velocity. The control is one-dimensional and the dynamics of the system is given as follows

$$\begin{aligned} s_{t+1}^1 &= s_t^1 + \frac{1}{1 + e^{-u_t}} - 0.5 + \kappa, \\ s_{t+1}^2 &= s_t^2 - 0.1s_{t+1}^1 + \kappa, \end{aligned}$$

where κ is a zero-mean Gaussian random variable with standard deviation $\sigma_\kappa = 0.02$. The agent starts in the state $\mathbf{s} = (0, 1)$, with the addition of Gaussian noise with standard deviation 0.001, and the objective is to transport the agent to the state $(0, 0)$. We use the same policy as in [29], which is given by $\mathbf{a}_t = (\mathbf{w} + \epsilon_t)^T \mathbf{s}_t$, where \mathbf{w} are the control parameters and $\epsilon_t \sim \mathcal{N}(\epsilon_t; \mathbf{0}, \sigma_\epsilon^2 I)$. The objective function is non-trivial for $\mathbf{w} \in [0, 60] \times [-8, 0]$. In the experiment the initial control parameters were sampled from the region $\mathbf{w}_0 \in [0, 60] \times [-8, 0]$. We considered a finite planning

²This software library is freely available at <http://www.di.ens.fr/~mschmidt/Software/minFunc.html>.

horizon with a planning horizon of $H = 80$. We used forward sampling to perform the inference, where in all algorithms 50 trajectories were sampled during each training iteration.

We also detail the procedure used for the step-size tuning for the approximate Newton method and natural gradient ascent. The tuning of the step-size sequence in steepest gradient ascent was similar in nature to natural gradient ascent and so is omitted from the discussion. Using the intuition that the approximate Newton method has a natural step-size of one, which corresponds to an EM-step in this problem because the log-policy is quadratic in the control parameters, we considered step-size sequences of the form $\alpha_k = (1 - \frac{k}{N})\alpha + \frac{k}{N}$, where N is the total number of training iterations considered and $\alpha \in \mathbb{R}^+$. In the experiment we considered the values $\alpha = 1, 6, 12, 18$ and 24 . The intuition used in this selection was that, provided that the steps were not so large as to cause overshooting in the parameter space, larger steps will increase performance. In steepest and natural gradient ascent we used step-size sequences that satisfied the Robbins-Munro conditions. In both of these methods it was necessary to obtain a gauge of a reasonable scale of a good step-size sequence. For this reason in the experiment we considered step-size sequences of the form $\alpha_k = \frac{\alpha}{\sqrt{k}}$ with $\alpha = 0.0001, 0.001, 0.01, 0.1, 1, 2$ and 4 . It was found that the sequence $\alpha = 18$ gave the best results for the approximate Newton method, while the sequence $\alpha = 0.001$ gave the best results for natural gradient ascent.