Adaptive MCMC with Policy Gradient

Goker Erdogan
Department of Brain and Cognitive Sciences
University of Rochester
Rochester, NY 14623
gerdogan@bcs.rochester.edu

Robert A. Jacobs
Department of Brain and Cognitive Sciences
University of Rochester
Rochester, NY 14623
robbie@bcs.rochester.edu

Abstract

1 Introduction[1]

Many problems in AI and cognitive science can be understood as inference in some probabilistic model [7, 23]. One popular technique for probabilistic inference is Markov chain Monte Carlo [19], which allows us to obtain samples from the full posterior (rather than just point estimates). However, MCMC can be rather inefficient, and designing an efficient MCMC procedure remains a manual and time-consuming process. Here we present an adaptive MCMC technique [15] that automates this design procedure and learns efficient samplers with as little manual effort as possible.

Our method is motivated by the following analogy with reinforcement learning. Sampling using MCMC can be understood as a sequential decision making problem, where the transition kernel of the Markov chain determines the decision policy. Given a suitable reward function that measures the quality of a MCMC sampler, we can frame the problem of learning an efficient sampler as finding the policy with maximum expected reward. We use policy gradients to solve this problem [22]. Our proposed method does not require any offline training and improves the sampler online, while ensuring at the same time that we still sample from the correct distribution. Our method is also general in the sense that it can be applied to any Markov chain Monte Carlo technique, as long as we can obtain gradients of the transition kernel of the chain. As an illustration of our method, we apply it to Metropolis-Hastings (MH) algorithm and show how efficient MH proposals can be learned for various problems.

Adaptive MCMC techniques enable learning from past samples. However, they do not address the problem of learning from past inferences. For example, an adaptive MCMC sampler for image segmentation will adjust its parameters to do inference more efficiently for a given image, but there is no mechanism for ensuring the transfer of this knowledge to segmenting a new image. Our method enables this transfer by learning from past inferences, in addition to past samples. We achieve this by allowing the transition kernel of the Markov Chain to depend not only on the current state but also on observed data [20]. As an illustration of this point, we show how our method can learn data-driven MH proposals from past samples and inferences.

2 Related work

Adaptive MCMC [15] techniques aim to adapt the sampler during sampling by looking at the collected samples. These methods are usually concerned with learning better values for parameters such as proposal variance. For example, [8] updates the covariance matrix of the Gaussian proposal distribution for an MH algorithm based on past samples. Similarly, [1] uses stochastic approximation

[1]This is a working paper. Simulations on various problems are currently being run, and these will be added to the results section as they become available. The authors welcome any comments on the present text.

to update the proposal variance to achieve an optimal acceptance rate. These methods are designed to adapt the value of a specific parameter and lack the generality of our approach, which can be applied to any parameter of the MCMC algorithm. Perhaps more similar to our work here is [12], which frames adaptive MCMC as an optimization problem of maximizing some measure of quality of the sampler. However, instead of using gradient-based techniques to solve this problem as we do here, they use Bayesian optimization. In addition, in contrast to our method, their method requires a separate offline adaptation stage.

Our work here is also motivated by work on amortized inference [17], which aims to adapt across many different inferences, rather than focusing on adaptation for a single posterior inference. Since many of the inference problems we face are similar, each new problem should not be solved from scratch, and we can learn from past inferences to build a more efficient inference procedure. Past work in this field usually focused on inference in Bayesian networks. One technique [17] learns inverse conditionals from past inferences and uses these for building efficient proposals for doing inference using MH algorithm. Another line of research focuses on the message passing algorithm [10, 4]. For example, [4] uses past inferences to learn mappings between incoming and outgoing messages for nodes in a Bayesian network, which substantially increase the speed of inference using the message passing algorithm. Our method focuses rather on using MCMC for inference and is more general in the sense that it can be applied to inference in any probabilistic model, not only Bayesian networks.

Our method is built on an analogy between inference and reinforcement learning (RL) that views inference as a sequential decision making problem. Previous work on problems such as structured prediction and combinatorial optimization has also taken a similar perspective. For example, [3] treat combinatorial optimization as a RL problem and use a temporal difference algorithm to learn efficient search procedures for a scheduling problem. Similarly, [2] treats structured prediction as a search problem and frames learning a search policy as a sequential decision making problem. However, instead of using RL techniques to learn this policy, they adopt an imitation learning approach that relies on labeled data. In contrast to our approach here, this line of work focuses on search rather than inference and does not take a probabilistic perspective to inference and learning. In that respect, work by [21] and [16], which address inference in probabilistic models, are more similar to our work here. [16] focus on structured prediction using Gibbs sampling and aim to learn an efficient proposal strategy for picking the next variable to update. They formulate this as a RL problem and use a temporal difference algorithm with a function approximator to learn an action-value function that predicts the value of picking each variable. This action-value function is learned offline in a separate step and later used to speed up inference for new problems. Also, [16]'s approach is designed specifically for Gibbs sampling, and action-value function approach relies on having a finite set of actions and cannot be applied to continuous action spaces (i.e., proposal strategies). These limitations are also shared by [21] which take a similar RL approach to inference in factor graphs. They learn an action-value function using Q-learning with a function approximator to pick the next variable to update in the factor graph. After this function is learned offline in a training phase, it is used as an efficient proposal for doing inference in the factor graph using MH algorithm.

3 Our adaptive MCMC method

Let $P(X|D)$ be the target distribution we want to sample from, where D denotes the observed data. We construct a Markov chain with transition kernel $P(x_{t+1}|x_t)$ such that its stationary distribution is the target distribution $P(X|D)$. The efficiency of such a sampler depends on the transition kernel P, and our aim here is to learn a good transition kernel as we run the chain. Let $x_1, x_2, \ldots, x_T$ denote the states of the chain and θ denote the parameters of the transition kernel. We assume that for each θ, the transition kernel P_θ has the correct stationary distribution, i.e., $\int P(x|D)P_\theta(x'|x)dx = P(x'|D),\ \forall\theta$. If $R : \mathcal{X}^T \to \mathbb{R}$ measures the "goodness" of a transition kernel, we can formulate our problem as finding the transition kernel P_θ with maximum expected "goodness"

$$\max_\theta\ \mathbb{E}_{x_1,x_2,\ldots,x_T}\left[R(x_1, x_2, \ldots, x_T)\right] \tag{1}$$

In order to solve this problem, we make the following analogy with reinforcement learning. We can think of the transition kernel P_θ as our policy and R as our reward function. Then the problem becomes finding the policy that maximizes expected future reward. We can apply techniques from the reinforcement learning literature to solve this problem. Specifically, we can get the gradient of the

above objective using the policy gradient method and use gradient-based methods to optimize it [22].

$$\nabla_\theta \mathbb{E}_{x_1,x_2,\ldots,x_T}\left[R(x_1,x_2,\ldots,x_T)\right] \tag{2}$$

$$=\nabla_\theta \sum_{x_1,x_2,\ldots,x_T} R(x_1,x_2,\ldots,x_T) \prod_{t=1}^{t=T-1} P_\theta(x_{t+1}|x_t) \tag{3}$$

$$= \sum_{x_1,x_2,\ldots,x_T} R(x_1,x_2,\ldots,x_T)\nabla_\theta \prod_{t=1}^{t=T-1} P_\theta(x_{t+1}|x_t) \tag{4}$$

$$= \sum_{x_1,x_2,\ldots,x_T} R(x_1,x_2,\ldots,x_T)\nabla_\theta \prod_{t=1}^{t=T-1} P_\theta(x_{t+1}|x_t)\frac{\prod_{t=1}^{t=T-1} P_\theta(x_{t+1}|x_t)}{\prod_{t=1}^{t=T-1} P_\theta(x_{t+1}|x_t)} \tag{5}$$

$$= \sum_{x_1,x_2,\ldots,x_T} R(x_1,x_2,\ldots,x_T) \prod_{t=1}^{t=T-1} P_\theta(x_{t+1}|x_t)\nabla_\theta \log \prod_{t=1}^{t=T-1} P_\theta(x_{t+1}|x_t) \tag{6}$$

$$=\mathbb{E}_{x_1,x_2,\ldots,x_T}\left[R(x_1,x_2,\ldots,x_T) \sum_{t=1}^{t=T-1} \nabla_\theta \log P_\theta(x_{t+1}|x_t)\right] \tag{7}$$

Hence, given N sample paths, $\{x_t^n\}_{t=1}^T, n = 1,2,\ldots,N$, we can estimate the gradient of the objective in Eqn. 1 as follows.

$$\nabla_\theta \mathbb{E}_{x_1,x_2,\ldots,x_T}\left[R(x_1,x_2,\ldots,x_T)\right] \approx \frac{1}{N}\sum_{n=1}^{N} R(x_1^n,x_2^n,\ldots,x_T^n)\left(\sum_{t=1}^{t=T-1} \nabla_\theta \log P_\theta(x_{t+1}^n|x_t^n)\right) \tag{8}$$

This motivates the following algorithm for an adaptive MCMC method. We run N parallel chains, each for T timesteps and collect samples $\{x_t^n\}_{t=1}^T, n = 1,2,\ldots,N$. We use Eqn. 8 to estimate the gradient and update θ using a gradient-based optimization procedure. However, even though the transition kernel for each value of θ has the desired stationary distribution, the adaptive chain may not, since the transition kernel of the chain changes after each θ update. We can ensure that the adaptive chain still has the correct stationary distribution if we can show that our adaptive scheme satisfies the following diminishing adaptation condition [14]

$$\lim_{t\to\infty} \sup_{x\in\mathcal{X}} ||P_{\theta_{t+1}}(.|x) - P_{\theta_t}(.|x)|| \tag{9}$$

where $||.||$ denotes the total variation distance. This condition is satisfied when the amount of adaptation in the kernel goes to 0 in the limit. We ensure that by requiring that the learning rate γ (step size of θ updates) go to 0 in the limit. Our proposed algorithm can be seen in Fig. 1.

Learning data-driven proposals for Metropolis-Hastings algorithm Our method can be applied to any Markov chain Monte Carlo technique that provides a way for constructing a Markov chain with a desired stationary distribution. Here we focus on one such technique, and show how our method can be used to learn data-driven proposals for Metropolis-Hastings algorithm (MH) [9]. In MH, we construct the Markov chain by splitting the transition from x_t to x_{t+1} into a proposal step followed by an accept/reject step. We sample the proposed state x' from a data-driven proposal distribution $q_\theta(x'|x_t, D)$ and accept this new state with some probability $a_\theta(x_t \to x')$. Note that acceptance ratio $a(. \to .)$ also depends on θ since the proposal probabilities are needed to calculate the acceptance ratio. The transition kernel can be written as

$$P_\theta(x_{t+1}|x_t) = \begin{cases} q_\theta(x_{t+1}|x_t, D)a_\theta(x_t \to x_{t+1}) & x_t \neq x_{t+1} \text{ (accept)} \\ 1 - \int q_\theta(x'|x_t)a_\theta(x_t \to x')dx' & x_t = x_{t+1} \text{ (reject)} \end{cases} \tag{10}$$

The acceptance ratio for the Metropolis-Hastings algorithm is given by

$$a_\theta(x \to x') = \min\left(1, \frac{P(x'|D)q_\theta(x|x', D)}{P(x|D)q_\theta(x'|x, D)}\right) \tag{11}$$

We need the derivative of the MH transition kernel with respect to θ, which can be estimated as follows by looking at three cases. Let x' denote the proposed state at time t. If the proposed move is accepted $x_{t+1} = x'$; if not, $x_{t+1} = x_t$. Let us first look at the case when

$$\frac{P(x'|D)q_\theta(x_t|x', D)}{P(x_t|D)q_\theta(x'|x_t, D)} \geq 1$$

function ADAPTIVEMCMCWITPOLICYGRADIENT(N: number of parallel chains, T: episode length, E: number of episodes, $\gamma_{e=1}^{\infty}$: learning rate sequence with $\lim_{e\to\infty}\gamma_e = 0$)
  Initialize θ
  Initialize the initial states x_1^n for each chain
  for $e = 1, 2, \ldots, E$ **do**
    for $n = 1, 2, \ldots, N$ **do**
      $dT^n \leftarrow 0$
      for $t = 2, 3, \ldots, T$ **do**
        $x_t^n \leftarrow P_\theta(.|x_{t-1}^n)$ ▷ Sample next state from the transition kernel
        $dT^n \leftarrow dT^n + \nabla_\theta \log P_\theta(x_t^n|x_{t-1}^n)$ ▷ Accumulate the gradient of transition kernel
      end for
      $x_1^n \leftarrow x_T^n$ ▷ Set the initial state for the next episode
      $R^n \leftarrow R(x_1^n, x_2^n, \ldots, x_T^n)$ ▷ Get the reward for this chain
    end for
    $\delta\theta \leftarrow \frac{1}{N}\sum_{n=1}^{N} R^n dT^n$ ▷ Calculate the estimate of the gradient
    $\theta \leftarrow \theta + \gamma_e \delta\theta$ ▷ Update θ
  end for
end function

Figure 1: Our proposed adaptive MCMC algorithm

i.e., $a_\theta(x_t \to x') = 1$. In this case, we can ignore the acceptance ratio[2] and write

$$P_\theta(x_{t+1}|x_t) = q_\theta(x'|x_t)$$

In the second case, we accept the move to x' but $a_\theta(x_t \to x') < 1$. Then

$$P_\theta(x_{t+1}|x_t) = q_\theta(x'|x_t, D)\frac{P(x'|D)q_\theta(x_t|x', D)}{P(x_t|D)q_\theta(x'|x_t, D)}$$

which simplifies to

$$P_\theta(x_{t+1}|x_t) = \frac{P(x'|D)}{P(x_t|D)}q_\theta(x_t|x', D)$$

The final case is rejection for which $P(x_{t+1}|x_t)$ is given by

$$\int q_\theta(x'|x_t, D)(1 - a_\theta(x_t \to x')dx'$$

However, this integral requires summing over all possible x'. We will approximate it with the current rejected sample as follows

$$\int q_\theta(x'|x_t, D)(1 - a_\theta(x_t \to x')dx' \approx q_\theta(x'|x_t, D)(1 - a_\theta(x_t \to x'))$$

Collecting all three cases, we get the following expressions for the derivative of log transition probabilities for MH algorithm

$$\nabla_\theta \log P_\theta(x_{t+1}|x_t) = \begin{cases} \nabla_\theta \log q_\theta(x_{t+1}|x_t) & \text{accept} \wedge a_\theta(x_t \to x_{t+1}) = 1 \\ \nabla_\theta \log q_\theta(x_t|x_{t+1}) & \text{accept} \wedge a_\theta(x_t \to x_{t+1}) < 1 \\ \nabla_\theta \log[q_\theta(x'|x_t)(1 - a_\theta(x_t \to x'))] & \text{reject} \end{cases} \tag{12}$$

Measuring "goodness" of a transition kernel A natural way to measure the efficiency of a sampler is to look at its autocorrelation time [6, 12]. Given a set of samples $x_1, x_2, \ldots, x_T$ from a Markov chain with $E[x_t] = \mu$ and var$(x_t) = \sigma^2$, the autocorrelation time is the value τ such that [18]

$$\sqrt{\frac{T}{\tau}}\frac{\bar{x} - \mu}{\sigma} \to N(0, 1) \tag{13}$$

[2]We assume that small changes in θ does not affect the acceptance ratio. This is a plausible assumption that is only violated at the boundary where $\frac{P(x'|D)q_\theta(x_t|x', D)}{P(x_t|D)q_\theta(x'|x_t, D)} = 1$.

where $\bar{x}$ is the sample mean. Intuitively, autocorrelation time measures how many samples from the chain are needed to get the equivalent of one independent sample from the target distribution. Autocorrelation time is difficult to calculate theoretically, but there are various methods for estimating it from a given set of samples. Here we use the method of batch means where we split the set of samples into batches of size M and calculate the mean for each batch. Let s_b^2 and s^2 denote the sample variance of the batch means and the whole sample respectively. We can obtain an estimate of the autocorrelation time as follows.

$$\hat{\tau} = M \frac{s_b^2}{s^2} \tag{14}$$

Reducing the variance In practice, the policy gradient estimate given in Eqn. 8 has too high variance to be useful. Therefore, one needs to resort to variance reduction techniques. We have experimented with various techniques but opted for a simple method inspired by [13]. For each chain n, we define the constant baseline b_{-n} that is the average of the rewards from the other chains, i.e.,

$$b_{-n} = \frac{1}{N-1} \sum_{j \neq n} R(x_1^j, x_2^j, \ldots, x_T^j) \tag{15}$$

Then our gradient estimate is

$$\frac{1}{N} \sum_{n=1}^{N} [R(x_1^n, x_2^n, \ldots, x_T^n) - b_{-n}] \left(\sum_{t=1}^{t=T-1} \nabla_\theta \log P_\theta(x_{t+1}^n | x_t^n) \right) \tag{16}$$

We can see that such a constant baseline does not introduce any bias into the gradient estimate since the constant baseline has expectation 0.

$$\begin{aligned}
\mathbb{E}_x[b \nabla_\theta \log P_\theta(x)] &= b \mathbb{E}_x[\nabla_\theta \log P_\theta(x)] && (17)\\
&= b \sum_x [P_\theta(x) \nabla_\theta \log P_\theta(x)] && (18)\\
&= b \sum_x [\nabla_\theta P_\theta(x)] && (19)\\
&= b \nabla_\theta \sum_x [P_\theta(x)] && (20)\\
&= b \nabla_\theta 1 && (21)\\
&= 0 && (22)
\end{aligned}$$

This simple baseline subtraction reduces the variance substantially with no extra cost.

4 Results

We first apply our method to a simple problem where we sample from a 1D Gaussian with a Gaussian proposal distribution that is centered at the current state. What is the optimal variance for such a proposal distribution? Previous research has shown that the optimal proposal variance (i.e., variance that minimizes autocorrelation time) in such a case is around $2.4\sigma^2$ where σ^2 is the variance of target distribution [5]. We use our method to learn a Gaussian proposal with state-dependent variance, $q_\theta(x'|x) = N(x, f_\theta(x))$ where the variance depends on the state linearly $f_\theta = \exp(wx + b)$. We run 10 chains in parallel for 2000 episodes with 100 timesteps per episode. We use a linearly decreasing learning rate from 0.01 to 0.0 and perform parameter updates using the Adam optimizer [11]. In Fig. 2, we plot how parameters w and b evolve as learning progresses. We see that our method converges to the proposal distribution with variance around 2.4, ($b = \log(2.4) = 0.875$). Note that we also see that w converges to 0, i.e., the variance of the optimal proposal distribution with this particular form is independent of state. The proposal learned by our model is significantly better than the naive proposal with variance 1 (autocorrelation with learned proposal: 3.08 ± 0.11, autocorrelation with naive proposal: 4.97 ± 0.29).

As another illustrative example, we next apply our method to a Gaussian proposal with state-dependent mean, $q_\theta(x'|x) = N(x + f_\theta(x), 1)$ where $f_\theta(x) = wx + b$. This case has not been investigated in previous research, but we can at least intuitively expect the mean of the proposal to be pulled towards

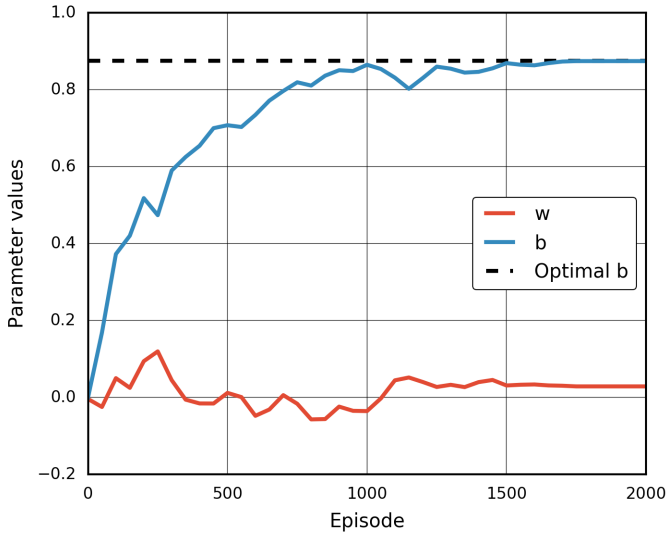
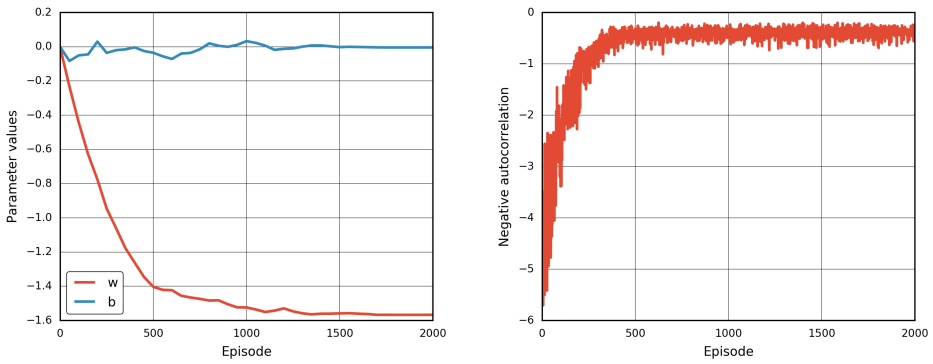


Figure 2: Weight and bias parameters' values with respect to training episode for a proposal with state-dependent variance.

the mean of the target distribution. In fact, when $w = -1$ and $b = 0$, the proposal distribution becomes the target distribution and one might expect that is the optimal proposal since each sample will be independent ($\tau = 1$). However, one can do better if the next state is negatively correlated with the current state. Hence, we would expect $w > -1$. We run our method with the same parameters for the previous example and look at how w and b evolve as learning progresses (Fig. 3a). w seems to converge to a value around -1.6, in line with our prediction. The autocorrelation time τ for the final proposal is 0.37 ± 0.02 (Fig. 3b). One sample with the learned proposal is worth around 2.7 independent samples.



(a) (b)

Figure 3: (a) Weight and bias parameters' values with respect to training episode for a proposal with state-dependent mean. (b) Reward (negative autocorrelation) with respect to training episode.

...

5 Conclusion

...

Acknowledgments

References

[1] YF Atchade and JS Rosenthal. On Adaptive Markov Chain Monte Carlo Algorithms. *Bernoulli*, 11(5):815–828, 2005.

[2] Hal Daume, John Langford, and Daniel Marcu. Search-based Structured Prediction. *Machine Learning*, 75(3):297–325, 2009.

[3] Thomas G Dietterich and Wei Zhang. A Reinforcement Learning Approach to Job-shop Scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1114–1120, San Francisco, CA, 1995. Morgan Kauffman.

[4] S. M. Ali Eslami, Daniel Tarlow, Pushmeet Kohli, and John Winn. Just-In-Time Learning for Fast and Flexible Inference. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 154–162. Curran Associates, Inc., 2014.

[5] Andrew Gelman, G Roberts, and W Gilks. Efficient metropolis jumping rules. In J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics 5*, pages 599–607. Oxford University Press, 1996.

[6] Charles J. Geyer. Practical Markov Chain Monte Carlo. *Statistical Science*, 7(4):473–483, 1992.

[7] Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.

[8] Heikki Haario, Eero Saksman, and Johanna Tamminen. An Adaptive Metropolis Algorithm. *Bernoulli*, 7(2):223–242, 2001.

[9] WK Hastings. Monte Carlo Sampling Methods Using Markov Chains and their Applications. *Biometrika*, 57(1):97–109, 1970.

[10] Nicolas Heess, Daniel Tarlow, and John Winn. Learning to Pass Expectation Propagation Messages. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3219–3227. Curran Associates, Inc., 2013.

[11] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, 2014.

[12] Nimalan Mahendran, Z Wang, F Hamze, and N Freitas. Adaptive MCMC with Bayesian Optimization. In *International Conference on Artificial Intelligence and Statistics*, volume 9, 2010.

[13] Andriy Mnih and Danilo J. Rezende. Variational inference for Monte Carlo objectives. *arXiv:1602.06725 [cs, stat]*, 2016.

[14] Gareth O Roberts and Jeffrey S. Rosenthal. Coupling and Ergodicity of Adaptive MCMC. *Journal of Applied Probability*, 44(2):458–475, 2007.

[15] Gareth O. Roberts and Jeffrey S. Rosenthal. Examples of Adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349–367, 2009.

[16] Tianlin Shi, Jacob Steinhardt, and Percy Liang. Learning Where to Sample in Structured Prediction. In *AISTATS 2015*, volume 38, 2015.

[17] Andreas Stuhlmueller, Jessica Taylor, and Noah D Goodman. Learning Stochastic Inverses. *Advances in Neural Information Processing Systems 27 (NIPS 2013)*, pages 1–9, 2013.

[18] Madeleine B. Thompson. A Comparison of Methods for Computing Autocorrelation Time. *arXiv:1011.0175 [stat]*, 2010.

[19] L Tierney. Markov Chains for Exploring Posterior Distributions. *The Annals of Statistics*, 22(4):1701–1728, 1994.

[20] Z Tu and SC Zhu. Image Segmentation by Data-Driven Markov Chain Monte Carlo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):657–673, 2002.

[21] Michael Wick, Khashayar Rohanimanesh, Sameer Singh, and Andrew McCallum. Training Factor Graphs with Reinforcement Learning for Efficient MAP Inference. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 2044–2052. Curran Associates, Inc., 2009.

[22] Ronald J. Williams. Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3-4):229–256, 1992.

[23] Alan Yuille and Daniel Kersten. Vision as Bayesian Inference: Analysis by Synthesis? *Trends in Cognitive Sciences*, 10(7):301–8, 2006.