# Neural Block Sampling

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Sequential Monte Carlo (SMC) is a simulation based method for performing inference on sequences of probability distributions. SMC is susecptable to sample impoverishment causing deterioration to the estimates of earlier marginals. We propose a block sampling algorithm with a mechanism to find the proposal and conditioning distributions. The algorithm approximates the optimal block proposals with neural density estimators. The proposal is data-driven and trained from the model. We compare neural block SMC to exisiting neural SMC methods on volatility models and non-linear state space models.

## 1 Introduction

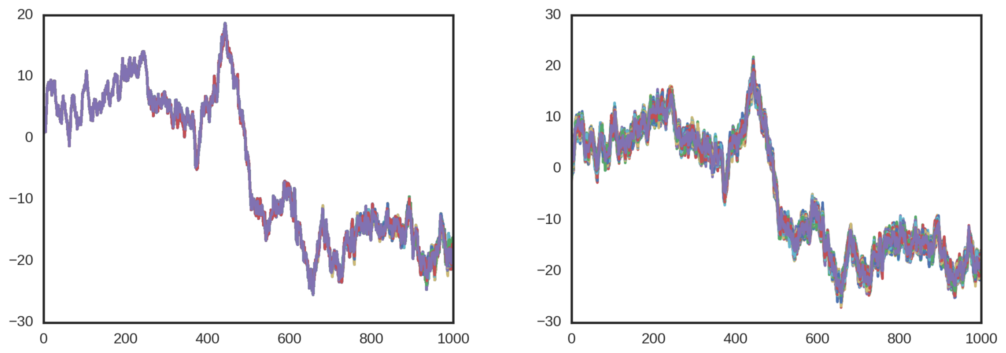
In applications such as probabalistic programming systems or state space models we have an inference problem where we are interested in the properties of the program state or hidden state at given points in execution or timesteps. The usual approach to solving these problems is to use Sequential Monte Carlo (SMC) methods based on simulating a set of weighted particles targeting a sequence of distributions. (See Doucet and Johansen [2011] and Cappé et al. [2007] for more details.) SMC has found applications in fields far ranging from econometrics to engineering.

In order to prevent sample degeneracy, a situation where a few elements of the sample hold the majority of the weight causing poor approximations of the target distributions empirical CDF most SMC algorithms have a resampling step triggered when a degeneracy threshold has been met. The resampling step corrects the particle distribution to the target distribution by deleteing and duplicating particles leaving us with an unweighted sample. But this comes at the cost of reducing the number of unique particles.

Prior work by Douc et al. [2005] and other focus on reducing the impact of resampling however we take the appropach of avoiding the need to resample. In this paper we will demonstrate an method to train proposals that sample efficiently from the target distributions, minimising the number of resampling steps required and consequently maximising the number of unique samples. The proposals are data-driven and are trained on data drawn from the model.

Figure 1 (a) demonstrates the problem we wish to solve. By sampling from the transition dynamics infomation available from the observed data is ignored causing the particle density to quickly drift away from the target density forcing repeated resampling. This results in all trajectories tracing their ancestry back to a handfull of common ancestors after only a few timesteps resulting in poor estimates of marginals not close to the current timestep. Figure 1 (b) shows the benefit of sucessfully applying our method. The block sampler used a lag length of 5 which typically does not require a resampling step for such a small number of timesteps. Each trajectory is unique and the full sample size is realised at all timesteps.

**Related Work** There are many alternative strategies to combating sample degeneracy. Gilks and Berzuini [2001] adds MCMC moves to maintin diversity and Lindsten et al. [2014] uses acestor



(a) SMC sampling using a bootstrap proposal. (b) Block SMC sampling using a Kalman proposal.

Figure 1: Sampled trajectories from an linear Gaussian model. (a) Sampling from the prior leads to frequent resampling causing rapid impoverishment of the sample. (b) Sampling from a Kalman forward-filtering backward-smoothing proposal requires no resampling in the observed timesteps.

sampling. Whilst block sampling could be considered a look backward method, Lin et al. [2013] is lookahead and uses future infomation. Some of these methods and more are covered in Li et al. [2014]'s review of techniques.

Prior work on using neural networks to improve SMC include Gu et al. [2015] using long short term memory units to propose mixture density network parameters whilst maintaining state within the neural network between timesteps to approximate the optimal SMC proposal. Perov et al. [2015] applied similar methods to probabalistic programming systems.

Finally the reverse concept, using SMC to assist the training of neural networks, is studied by de Freitas et al. [2000].

## 2 Neural Block Sequential Monte Carlo

Replaceing standard SMC methods with block sampling moves our problem from sample impovrishment to finding good block proposals that reduce resampling. Then the number of neural networks become unscalable and we introduce more efficient methods. Finally we describe how these density estimators can learn an approximation to the optimal block proposal and conditioning distribution.

The solution is to sequentially importance sample and resample when necessary. Instead of attempting to draw a sample of the entire chain immediately the sampling process becomes an iterative algorithm with the opportunity to resample. The resampling turns the weighted sample of particles into an unweighed sample whenever a criteria is met indicating the distribution of the particles has deviated too far from the target. Assume we have a weighted sample $\left\{X_{0:n-1}^{(i)}, W_{n-1}^{(i)}\right\}$ from the time $n-1$ and proceed as follows for time $n$.

Without resampling the particles will eventually be dominated by a few heavily weighted particles and many with negligible weight, this is weight degeneracy. Much computational effort is wasted propagating many insignificant particles, so we resample but every time particles are resampled the number of unique particles in previous time steps decreases eventually leading sample impoverishment in the empirical filtering distribution. This frequent resampling and consequential particle degeneracy is the motivation for Block SISR.

The SMC proposal is user defined and crucial to efficient sampling. Unlike importance sampling the model has observations that could be used by a proposal to match the posterior more closely. A common default value for the proposal distribution is the prior resulting in the bootstrap $f(x_n|x_{n-1})$. This proposal ignores information from the current state in cases when the transition density is relatively uninformative compared to the observation density the bootstrap will generate poor samples. In SMC the optimal proposal for minimising variance in the weights is a conditional from the posterior $\pi_n(x_n|x_{0:n-1})$ and user selected proposals should attempt to approximate this as closely as possible.

### 2.1 Block Sequential Importance Sampling and Resampling

Standard SMC suffers from rapid particle trajectory degeneracy even with good proposals as repeated resampling collapses the ancestry of particles into a single common ancestor. Block sequential Monte Carlo Doucet et al. [2006] aims to prevent this by sampling values over a moving window of fixed length.

The drawback of block SMC is the difficulty in constructing a suitable and well performing proposal distribution. Models that are sufficiently linear can use the Kalman filter but we wish to consider all models. We aim to resolve this challenge by treating the optimal block proposal distribution as a training target for neural density estimator.

**Conditional distribution** also known as the backwards distribution is necessary to ensure both the proposal and target sides of the importance weights are sampling on the same space. The backwards distribution must be independent of the newly sampled states otherwise those variables must be integrated out so as not to modify the importance sampling space.

$$\lambda_n^*(x_{n-L+1:n-1}|x_{1:n-L}, x'_{n-L+1:n}) = \pi_{n-1}(x_{n-L+1:n-1}|x_{1:n-L})$$

**Proposal** distribution must also be independent of the previous values that are discarded similar to the conditional distribution. The optimal proposal is $q_n^*(x'_{n-L+1:n}|x_{1:n-1}) = \pi_n(x'_{n-L+1:n}|x_{1:n-L})$ and the user should construct a proposal that best approximates this distribution.

### 2.2 Neural density Estimation

Combining the necessity for a good proposal in SMC with the flexibility of neural networks to learn functions we review neural density estimation. By using loss functions that penalise of the distance between distributions neural networks are trained to estimate good proposal distributions.

We explain neural density estimators with a look at the simplest non-pathological example of one. The single density estimator predicts the distribution of a single dimensional random variable given any number of predictors. The construction is a neural network which accepts the predictors as input and outputs predictions of the parameters of a fixed probability distribution.

Suppose we have an unknown conditional density $p(\cdot|y_1, \ldots, y_d)$ of interest that we wish to approximate with $\tilde{p}$. Let $\mathfrak{N}_w$ be a neural network with a randomly initialised set of weights $w$ such that:

$$\mathfrak{N}_w : \mathbb{R}^d \to \mathbb{R} \times \mathbb{R}^+$$

Where $d$ is the dimension of the input and the output layer is the identity function for the first dimension and the exponential function for the second dimension to ensure valid parameters for the density.

**Mixtures of Densities** As the target distribution is not necessarily unimodal, nor the number of modes fixed or even belonging to any common parametric distribution the mixture density estimator Bishop [1994] aims to fit a mixture of distributions.

Again we use Gaussians for our mixture density however any distribution, including any combination of distributions could be substituted as necessary and the network output layer adjusted accordingly. This is particularly important to train the correct tails. In the SMC context it may be advantageous to include the transition density as one of the mixture components to kick start training.

Suppose we have another unknown density $p(\cdot|\mathbf{y})$ which we think can be approximated with at most $K$ Gaussian components. Let $\mathfrak{N}_w$ be a neural network with a randomly initialised set of weights $w$ such that:

$$\mathfrak{N}_w : \mathbb{R}^d \to \mathbb{R}^K_{[0,1]} \times \mathbb{R}^K \times \mathbb{R}^K_+$$

The output layer for the first $K$ dimensions is a softmax function which are the mixing parameters, the second $K$ dimensions is an identity function which are the Gaussian means and the remaining $K$ dimensions is an exponential function which are the Gaussian variances. Then the neural network is denoted $\mathfrak{N}_w$ and the mixture of Gaussians is $\tilde{p}_w$.

**Real-valued Neural Autoregressive Density Estimator** Finally we examine the Real-valued Neural Autoregressive Density Estimator by Uria et al. [2014] which will eventually be used to construct proposals for block sampling.

Suppose we have an unknown density $p(\mathbf{x}|\mathbf{y})$ where $\mathbf{x}$ is a $D$ dimensional random variable and we will approximate each conditional dimension by a mixture of $K$ Gaussians. Let $\{\mathfrak{N}_{w_{<d}} | d \in [1, D]\}$ be a sequence of neural networks where $w$ is the weight matrix and $w_{<d}$ denotes the first $d-1$ columns. Then each neural network accepts an additional dimension of input from the previously sampled dimension:

$$\mathfrak{N}_{w_{<d}} : \mathbb{R}^{i+d-1} \to \mathbb{R}^K_{[0,1]} \times \mathbb{R}^K \times \mathbb{R}^K_+$$

And outputs the same mixture of Gaussian parameters as a mixture density network.

$$\mathfrak{N}_{w_{<d}}(\mathbf{x}_{<d}, \mathbf{y}) = (\pi_{1:K}, \mu_{1:K}, \sigma^2_{1:K})$$

$$\tilde{p}(x_d|\mathbf{x}_{<d}, \mathbf{y}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\cdot|\mu_k, \sigma_k^2)$$

Resulting in an untrained neural density estimator:

$$\tilde{p}(\mathbf{x}|\mathbf{y}) = \prod_{d=1}^{D} \tilde{p}_{w_{<d}}(x_d|\mathbf{x}_{<d}, \mathbf{y})$$

Each of the neural network requires training to find the optimal matrix of shared weights. Using gradient descent for each $d \in [1, D]$ and by scaling the objective function by $\frac{-D}{D-d+1}$ we find the weights $w^*$.

## 2.3 Neural Block Proposal

With the framework for block SMC set by Doucet et al. [2006] the challenge of finding a good proposal has been made more difficult with the increased space. Just like standard SMC a poorly chosen proposal will perform worse than the bootstrap filter, taken in this section to be a lower bound on performance. In this section we generalise a Kalman filter based forward filter backward smoother to work with arbitrary models, irrespective of linearity.

**Forward filtering backward smoothing** To perform localised smoothing for a block proposal, we will sample from an approximation of the joint distribution $p(x_{n-L+1:n}|y_{n-L+1:n}, x_{n-L})$. The forward filtering-backward smoothing formula uses the following decomposition:

$$\tilde{p}(x_{n-L+1:n}|y_{n-L+1:n}, x_{n-L}) = \tilde{p}(x_n|y_{n-L+1:n}, x_{n-L}) \prod_{l=n-L+1}^{n-1} \tilde{p}(x_l|y_{n-L+1:l}, x_{n-L}, x_{l+1}) \tag{1}$$

Forward filtering, using Kalman filter for example, we obtain the marginals $\{\tilde{p}(x_l|y_{n-L+1:l}, x_{n-L})\}_{l=n-L+1,...,L}$ and then sample the final state $X_n \sim \tilde{p}(x_n|y_{n-L+1:n}, x_{n-L})$.

Sampling backwards from the smoothing distributions.

$$\tilde{p}(x_l|y_{n-L+1:l}, x_{n-L}, x_{l+1}) = \frac{f(x_{l+1}|x_l)\tilde{p}(x_l|y_{n-L+1:l}, x_{n-L})}{\int f(x_{l+1}|x_l)\tilde{p}(x_l|y_{n-L+1:l}, x_{n-L})dx_l}$$

Which is the renormalised product of the filtering density and the transition density with the future state. Difficulties occur when the integral in the backward smoothing steps do not have a closed form or if a particular filter cannot effectively approximate the marginals. We resolve this with the neural block proposal.

**Construction** Instead of obtaining marginal densities from the Kalman filter and then simulating a chain backwards and conditional on the next predicted state we use neural density estimators to learn each of the densities sampled from in the forward filtering-backward smoother trained with data sampled from the model. That is, we fit neural density estimators to $p(x_n|x_{n-L}, y_{n-L+1:n})$ forward density, and then each of the backward densities from $p(x_{n-1}|y_{n-L+1:n-1}, x_{n-L}, x_n)$ to $p(x_{n-L+1}|y_{n-L+1:n-L+2}, x_{n-L}, x_{n-L+2})$.

As only the final marginal density from the filter is directly sampled it is the only density that needs to be learned. Without knowing whether the product of the forward density estimate and state transition density can be integrated the density estimators are trained on draws from the model learning backward smoothing densities circumventing this step.

Careful ordering of input and output variables make it possible to replace all of the density estimators with a single RNADE taking advantage of its autoregressive property to replicate the decomposition used in backward smoothing. The draw from the filter is dependent only on initial state and observations then subsequent backward smoothing is additionally dependent on the future state which is fed back when RNADE samples additional dimensions. The drawback is that each marginal of RNADE is given many exogenous variables as input which may impede learning. This could be solved by using MADE Germain et al. [2015], an alternative to RNADE which uses masking instead of subsetting of the weights, and dictating the inputs independent of the current variable.

$$(X_n, \ldots, X_{n-L+1}) \sim \mathfrak{R}(\cdot|Y_{n-L+1}, \ldots, Y_n, X_{n-L})$$

Neural Block SMC will also require backwards densities constructed in an identical manner except with lag $L-1$. There could be a possibility to adopt a complex weight sharing scheme to take advantage of the similar backward smoothing densities but as the forward densities will be different causing the most difficulty should RNADE be used. Unless extraordinarily long lag lengths or deep and wide neural density estimators are used the benefits of a combined proposal and backwards density are outweighed by complexity of joint weights.

**Example** We demonstrate the neural block proposal on a univariate linear Gaussian model, chosen for the availability of an exact posterior.

Where $V_n, W_n \sim \mathcal{N}(0, 1)$. Setting the lag $L = 10$ we draw 10000 chains of length 10 from the model and train an RNADE over up to 100 epochs or until the validation error converges.

The proposal is benchmarked against two known proposals. First the bootstrap proposal is extended to create a 'block bootstrap' by repeated application of the transition density to set a lower performance benchmark. Then the optimum proposal computed from the Kalman forward filter backward smoother is the upper performance benchmark. A single chain is sampled and using the samples observations 1000 samples are drawn from the neural block proposal and both benchmark samplers. The mean trajectory is plotted and standard deviation shaded in figure 2.

### 2.4 Neural Block Sequential Monte Carlo

**Model** Let us consider a generic time homogeneous state space model

$$\begin{aligned} X_1 &\sim \pi(X_1) \\ X_n &= f(X_n|X_{n-1}) \\ Y_n &= g(Y_n|X_n) \end{aligned} \tag{2}$$

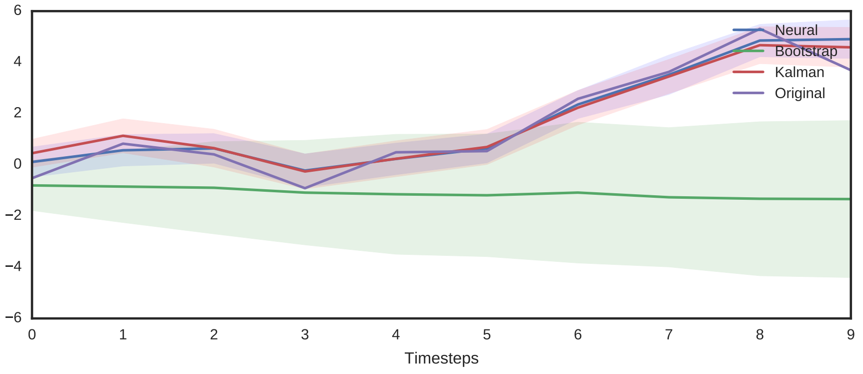


Figure 2: Median trajectories and standard deviations for the length 10 neural block, optimal block and bootstrap proposals.

where $f(\cdot|\cdot)$ and $g(\cdot|\cdot)$ are known densities.

We have a set of observations $Y_{1:N}$ and wish to infer the posterior distribution of $X_{1:N}$

**Training** For a fixed lag length $L$ we train a neural density estimator proposal which predicts $L$ states and another backward density which computes weights over $L-1$ states.

1. $\{x^{(i)}_{1:L}, y^{(i)}_{1:L}\}_{i=1:B} \leftarrow \text{SimulateMiniBatch}(L, B)$
2. $\Delta w^f \leftarrow \sum_i \frac{\partial}{\partial w^f} \log \mathfrak{R}_{w^f}(x^{(i)}_{L:1}|y^{(i)}_{1:L})$
3. $\Delta w^b \leftarrow \sum_i \frac{\partial}{\partial w^b} \log \mathfrak{R}_{w^b}(x^{(i)}_{L-1:1}|y^{(i)}_{1:L-1})$
4. $w^f \leftarrow \text{Optimise}(w^f, \Delta w^f)$
5. $w^b \leftarrow \text{Optimise}(w^b, \Delta w^b)$

Where we draw minibatches of training data from the model and compute the gradients of the weights from both proposal and backward distributions. Optimise is a user selected variant of stochastic gradient descent to determing the weight updates. An adaptive gradient descent algorithm such as Adam by Kingma and Ba [2014] should be used to prevent learning a local minima.

**Sampling** The key difference between the block SISR algorithm and the neural block algorithm is the assumed unavailability to propose blocks of variable length as it was necessary for the first $L-1$ time steps of the algorithm. During these timesteps we sample using a non-block proposal until time $L$ to maintain the online property of the algorithm. This causes a trade off in lag size for sufficiently large $L$ as the inefficiency of the first $L-1$ timesteps overcomes the gains of block sampling.

At time $n < L$.

1. Sample $X^{(i)}_n \sim f(\cdot|X^{(i)}_{1:n-1})$
2. Weigh $W^{(i)}_n \propto W^{(i)}_{n-1} g(Y_n|X^{(i)}_n)$
3. Set $\left\{X^{(i)}_{1:n}\right\} \leftarrow \left\{X^{(i)}_{1:n-1}, X^{(i)}_n\right\}$
4. If $\text{ESS}(W_n)$ low, resample.

At time $n \geq L$.

1. Sample $X'^{(i)}_{n-L+1:n} \sim q_n(\cdot|X^{(i)}_{1:n-1})$
2. Weigh

$$W^{(i)}_n \propto W^{(i)}_{n-1} \frac{\pi_n\left(X^{(i)}_{1:n-L}, X'^{(i)}_{n-L+1:n}\right) \lambda_n\left(X^{(i)}_{n-L+1:n-1}|X^{(i)}_{1:n-L}, X'^{(i)}_{n-L+1:n}\right)}{\pi_{n-1}\left(X^{(i)}_{1:n-1}\right) q_n\left(X'^{(i)}_{n-L+1:n}|X^{(i)}_{1:n-1}\right)} \tag{3}$$

3. Set $\left\{X_{1:n}^{(i)}\right\} \leftarrow \left\{X_{1:n-L}^{(i)}, X_{n-L+1:n}^{\prime(i)}\right\}$

4. If $\mathrm{ESS}(W_n)$ low, resample.

## 2.5 Algorithm Settings

Although for small models, especially the early examples considered in this paper it is feasible to train density estimators for lag lengths ranging from 1 to $L$ this is undesirable. As we wish to preserve the number of unique particles and we cannot block sample with a neural proposal until $n \geq L$ we must devise an alternative sampling scheme for the first $L - 1$ time steps that requires resampling minimally.

The naive method to sample the first $L - 1$ states is to run the bootstrap filter. However this will trigger frequent resampling defeating the goals of this algorithm. Starting a sample from the initialisation distribution we instead draw a sample from the neural block proposal and perform non-sequential importance sampling to weigh the sample. Alternatively if the block proposal is sufficiently good and the samples have high weight we can perform rejection sampling to start the block sampler with an unweighed sample.

## 3 Experiments

It is possible to exactly recover the optimal block distribution for certain models. We use this as an opportunity to compare the performance for the neural block proposal in figure 3. In terms of maintining the most unique particles we see that even at lag length $L = 2$ the optimum SMC proposal is easily surpassed. However the neural proposal requiring a longer lag length to replicate the performance of the optimal block proposal.

Next we test the performance on two non-linear models, the stochastic volatility model and benchmark non-linear model.

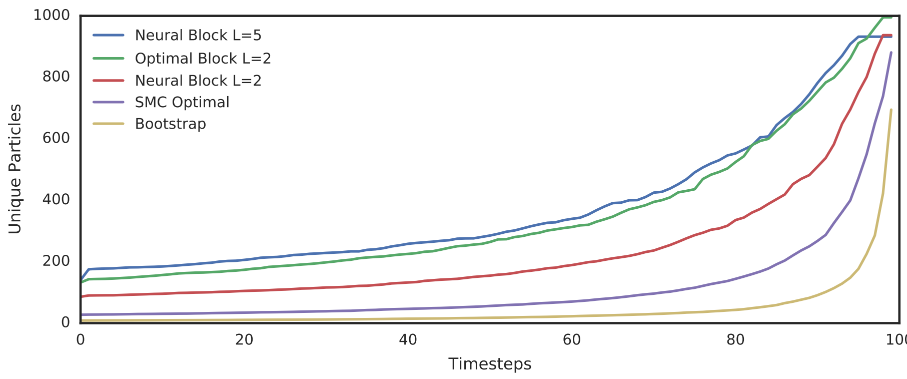


Figure 3: Unique particles from a linear Gaussian model.

## 3.1 Stochastic Volatility

We reproduce a stochastic volatility model of foreign exchange rate returns between the British Pound and US Dollar from a period between 1/10/1981 and 28/06/1985 in Durbin and Koopman [2012] and used as an example of block sampling in Doucet et al. [2006].

$$
\begin{aligned}
X_1 &\sim \mathcal{N}(0, \frac{\sigma^2}{1-\phi^2}) \\
X_n &= \phi X_{n-1} + \sigma V_n \\
Y_n &= \beta e^{X_n/2} W_n
\end{aligned} \tag{4}
$$

Where $V_n, W_n \sim \mathcal{N}(0,1)$.

As the underlying model is linear we don't expect it to be possible to outperform the Kalman filter forward filter backward smoother proposal. So for this model we aim to learn the Kalman filter with density estimators and match its performance.

The non-linear models are much more challenging for neural block sampling, but given long enough lag lengths it still acheives less resampling.

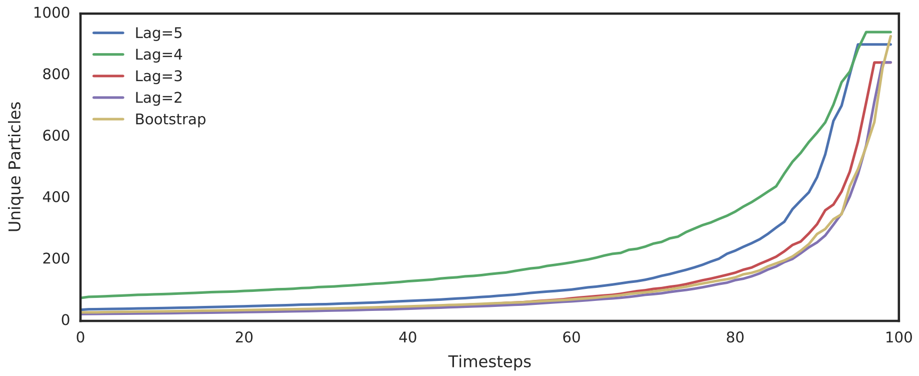


Figure 4: Unique particles in a stochastic volatility model.

### 3.2 Benchmark Non-Linear State Space Model

Next we consider the benchmark non-linear state space model from Gu et al. [2015] to benchmark Neural Adaptive SMC. Due to it's non-linear behaviour we expect the neural block proposal to outperform alternative block proposals and standard SMC.

$$\begin{aligned} X_1 &\sim \mathcal{N}(0,5) \\ X_n &= \frac{X_{n-1}}{2} + \frac{25X_{n-1}}{1+X_{n-1}^2} + 8\cos(1.2n) + \sigma_v V_n \\ Y_n &= \frac{X_n^2}{20} + \sigma_w W_n \end{aligned} \tag{5}$$

where $V_n, W_n \sim \mathcal{N}(0,1)$.

## 4 Conclusion

We have developed a method to perform block sampling without requiring the user specify a performant proposal and backward conditioning distribution. Given any state space model we are now able to pre-train these distributions with scalable neural networks that approximate the optimal block disributions.

## References

Christopher M Bishop. Mixture density networks. 1994.

Olivier Cappé, Simon J Godsill, and Eric Moulines. An overview of existing methods and recent advances in sequential monte carlo. *Proceedings of the IEEE*, 95(5):899–924, 2007.

João FG de Freitas, Mahesan Niranjan, Andrew H. Gee, and Arnaud Doucet. Sequential monte carlo methods to train neural network models. *Neural computation*, 12(4):955–993, 2000.

Randal Douc, Olivier Cappé, and Eric Moulines. Comparison of resampling schemes for particle filtering. In *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, pages 64–69. IEEE, 2005.

Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: fifteen years later. In *Oxford Handbook of Nonlinear Filtering*, 2011.

Arnaud Doucet, Mark Briers, and Stéphane Sénécal. Efficient block sampling strategies for sequential monte carlo methods. *Journal of Computational and Graphical Statistics*, 15(3):693–711, 2006. ISSN 10618600. URL `http://www.jstor.org/stable/27594204`.

James Durbin and Siem Jan Koopman. *Time series analysis by state space methods*, chapter 14, page 319. Number 38. Oxford University Press, 2012.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: masked autoencoder for distribution estimation. In *Proceedings of the 32nd International Conference on Machine Learning, JMLR W&CP*, volume 37, pages 881–889, 2015.

Walter R Gilks and Carlo Berzuini. Following a moving targetmonte carlo inference for dynamic bayesian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63 (1):127–146, 2001.

Shixiang Gu, Zoubin Ghahramani, and Richard E Turner. Neural adaptive sequential monte carlo. In *Advances in Neural Information Processing Systems*, pages 2611–2619, 2015.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Tiancheng Li, Shudong Sun, Tariq Pervez Sattar, and Juan Manuel Corchado. Fight sample degeneracy and impoverishment in particle filters: A review of intelligent approaches. *Expert Systems with Applications*, 41(8):3944 – 3954, 2014. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2013.12.031. URL `http://www.sciencedirect.com/science/article/pii/S0957417413010063`.

Ming Lin, Rong Chen, Jun S Liu, et al. Lookahead strategies for sequential monte carlo. *Statistical Science*, 28(1):69–94, 2013.

Fredrik Lindsten, Michael I Jordan, and Thomas B Schön. Particle gibbs with ancestor sampling. *The Journal of Machine Learning Research*, 15(1):2145–2184, 2014.

Yura N Perov, Tuan Anh Le, and Frank Wood. Data-driven sequential monte carlo in probabilistic programming. *arXiv preprint arXiv:1512.04387*, 2015.

Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. *JMLR: W&CP*, 32(1):467–475, 2014.