The Neural Moving Average Model for Scalable Variational Inference of State Space Models: Supplementary Material

Thomas Ryder[1,2] **Dennis Prangle**[1] **Andrew Golightly**[1] **Isaac Matthews**[1]

[1]School of Mathematics Statistics and Physics, Newcastle University, Newcastle, United Kingdom
[2]Huawei Noah's Ark Lab

A MCMC ALGORITHM FOR AR(1) VIA FORWARD-FILTER RECURSION

Here we present an MCMC method for the AR(1) model in the main paper,

$$x_{i+1} = \theta_1 + \theta_2 x_i + \theta_3 \epsilon. \tag{1}$$

Recall that we take $\epsilon \sim N(0,1)$ and $x_0 = 10$. We assume observations $y_i \sim N(x_i, 1)$ for $i \in 0{:}T$, and independent $N(0, 10^2)$ priors on $\theta_1, \theta_2, \log\theta_3$.

Assuming T observations, the marginal parameter posterior is given by

$$p(\theta|y_{0:T}) \propto p(\theta)p(y_{0:T}|\theta), \tag{2}$$

where $p(y_{0:T}|\theta)$ is the marginal likelihood obtained from integrating out the latent variables from $p(\theta, x_{0:T}|y_{0:T})$. We sample the marginal parameter posterior $p(\theta|y_{0:T})$ using a random walk Metropolis-Hastings scheme.

This appendix describes the key step of evaluating the marginal likelihood given θ, which is achieved using a *forward filter*. See West and Harrison [2006] for a general introduction to forward-filtering algorithms for linear state space models. We adapt this as follows.

As can be seen from (1), the AR(1) model is linear and Gaussian. Hence, for a Gaussian observation model with variance σ^2, the marginal likelihood is tractable and can be efficiently computed via a forward-filter recursion. This utilises the factorisation

$$p(y_{0:T}|\theta) = p(y_0|\theta)\prod_{i=1}^{T} p(y_i|y_{0:i-1}, \theta), \tag{3}$$

by recursively evaluating each term.

Suppose that $x_i|y_{0:i} \sim N(a_i, c_i)$. Since $x_0 = 10$ we can take $a_0 = 10, c_0 = 0$. It follows that

$$x_{i+1}|y_{0:i} \sim N\left(\theta_1 + \theta_2 a_i, \theta_2^2 c_i + \theta_3^2\right), \tag{4}$$

which, from the observation model, gives us the one-step-ahead forecast

$$y_{i+1}|y_{0:i} \sim N\left(\theta_1 + \theta_2 a_i, \theta_2^2 c_i + \theta_3^2 + \sigma^2\right). \tag{5}$$

Hence the marginal likelihood can be recursively updated using

$$p(y_{0:i+1}|\theta) = p(y_{0:i}|\theta)p(y_{i+1}|y_{0:i}, \theta), \tag{6}$$

where $p(y_{i+1}|y_{0:i}, \theta)$ is the density of (5).

The next filtering distribution is obtained as $x_{i+1}|y_{0:i+1} \sim N(a_{i+1}, c_{i+1})$ where

$$a_{i+1} = \theta_1 + \theta_2 a_i + \frac{\left(\theta_2^2 c_i + \theta_3^2\right)\left(y_{i+1} - \theta_1 - \theta_2 a_i\right)}{\left(\theta_2^2 c_i + \theta_3^2 + \sigma^2\right)} \tag{7}$$

$$c_{i+1} = \theta_2^2 c_i + \theta_3^2 - \frac{\left(\theta_2^2 c_i + \theta_3^2\right)^2}{\left(\theta_2^2 c_i + \theta_3^2 + \sigma^2\right)}. \tag{8}$$

Evaluation of (4)-(8) for $i = 0, 1, \dots, T-1$ gives the marginal likelihood $p(y_{0:T}|\theta)$.

B MINI-BATCH SAMPLING

Algorithm A describes how to sample a subsequence $x_{a:b}$ from $q(x|\theta; \phi_x)$ without needing to sample the entire x sequence.

Let z^0 be the base random sequence and z^j be the sequence after j affine layers. We assume no layers permuting the sequence order (but do allow layers permuting the components within each vector in the sequence). Suppose there are m affine layers, so the output is $x = z^m$.

Algorithm A presents the multivariate case where $x_i, z_i^j, \mu_i^j, \sigma_i^j$ are all vectors in $\mathbb{R}^d$. This includes $d = 1$ as a special case. We denote the kth entry of σ_i^j as σ_{ik}^i. Recall that φ is the $N(0, I_d)$ log density function.

Each iteration of the algorithm (except the last) must sample z_i^j vectors over an interval of i which is wider than simply

Algorithm A Sampling a subsequence from a nMA model

```
1: Sample $z_i^0$ for $a - c_0 \le i \le b$. These are sampled from independent $N(0, I_d)$ distributions except when $i \notin 1{:}T$. In the latter case $z_i^0$ is a vector of zeros.
2: for $1 \le j \le m$ do
3:     Apply the CNN with input $z^{j-1}_{a-c_{j-1}:b}$. This outputs $\mu^j_{a-c_j:b}$ and $\sigma^j_{a-c_j:b}$.
4:     Calculate $z^j_{a-c_j:b}$ using affine transformation $z^j_i = \mu^j_i + \sigma^j_i \odot z^{j-1}_i$.
5:     Permute components in $z^j$ if necessary.
6: end for
7: Return sampled subsequence $x_{a:b} = z^m_{a:b}$, and log density contributions $\lambda_{a:b}$, where
       $\lambda_i = \varphi(z_i^0) - \sum_{k=1}^d \sum_{j=1}^m \log \sigma^j_{ik}.$
```

$a{:}b$. The number of extra z_i^js required at the lower end of this interval is

$$c_j = (m-j)\ell. \tag{9}$$

In other words, at each iteration the required interval shrinks by ℓ, the length of the receptive field for z.

C ORDER-REVERSING PERMUTATIONS

The main paper mentions that affine flow layers could be alternated with layers which reverse the order of the $x_{1:T}$ sequence. This can be accomplished by replacing Algorithm A above with Algorithm B. We state this algorithm using only the original sequence ordering. To do so we introduce an operation $\mathfrak{R}$ which reverses the order of a sequence.

As before, each iteration (except the last) samples z_i^j vectors over an interval of i wider than $a{:}b$. However now we need extra entries at both ends of this interval, c_j^- and c_j^+ at the lower and upper ends respectively. These can be defined recursively by $c_m^- = c_m^+ = 0$ and

$$(c_j^-, c_j^+) = (c_{j+1}^-, c_{j+1}^+) + \begin{cases} (\ell, 0) & j \text{ odd} \\ (0, \ell) & j \text{ even} \end{cases} \tag{10}$$

D SIDE INFORMATION

Here we give more details of what side information we inject into our nMA model for x. We inject this information into the first layer of the CNN for each of our affine transformations. Recall that firstly we include the parameters θ as global side information. Also we provide local side information, encoding information in y local to i which is useful for inferring the state x_i.

Algorithm B Sampling a subsequence from a nMA model, with order-reversing permutations

```
1: Sample $z_i^0$ for $a - c_0 \le i \le b$ as in Algorithm A.
2: for $1 \le j \le m$ do
3:     if $j$ odd then
4:         Apply the CNN with input $z^{j-1}_{a-c^-_{j-1}:b+c^+_{j-1}}$. This outputs $\mu^j_{a-c^-_j:b+c^+_j}$ and $\sigma^j_{a-c^-_j:b+c^+_j}$.
5:     else
6:         Apply the CNN with input $\mathfrak{R}(z^{j-1}_{a-c^-_{j-1}:b+c^-_{j-1}})$. This outputs $\mathfrak{R}(\mu^j_{a-c^-_j:b+c^+_j})$ and $\mathfrak{R}(\sigma^j_{a-c^-_j:b+c^+_j})$.
7:     end if
8:     Calculate $z^j_{a-c^-_j:b+c^+_j}$ using affine transformation $z^j_i = \mu^j_i + \sigma^j_i \odot z^{j-1}_i$.
9:     Permute components in $z^j$ if necessary.
10: end for
11: Return sampled subsequence $x_{a:b} = z^m_{a:b}$, and log density contributions $\lambda_{a:b}$, where
       $\lambda_i = \varphi(z_i^0) - \sum_{k=1}^d \sum_{j=1}^m \log \sigma^j_{ik}.$
```

In more detail, first we define s_i to be a vector of data features relevant to x_i. We pick these so that s_i exists for all i even if (1) no y_i observations exist for x_i or (2) i is outside the range $0{:}T$. The data features we use in our examples are listed in the next section.

The side information corresponding to the ith position in the sequence processed by the CNN is θ and the vector $s_{i-\ell':i+\ell'}$. The tuning parameter ℓ' is a receptive field length (like ℓ earlier). This receptive field extends in both directions from the sequence position i, so it can take account of both recent and upcoming observations. The side information is encoded using a feed-forward network, and this vector is then used as part of the input to the first layer of the CNN.

E IMPLEMENTATION DETAILS FOR ALGORITHM 1

Optimisation We use the AdaMax optimiser [Kingma and Ba, 2015], due to its robustness to occasional large gradient estimates. These sometimes occurred in our training procedure when different batches of the time series had significantly different properties. See Section F for its tuning choices. To stabilise optimisation, we also follow Pascanu et al. [2013] and clip gradients using the global L_1 norm.

Variational Approximation for θ For $q(\theta; \phi_\theta)$ we use a masked IAF as described in Section 3.1 of the main paper. In all our examples, this alternates between 5 affine layers and random permutations. Each affine transformation is based on a masked feed-forward network of 3 layers with 10 hidden units.

Unequal Batch Sizes The main paper assumes the training batch is split into batches $B_1, B_2, \dots, B_b$ of equal length. Recall that in this case a batch B_κ is sampled at random to use in a training iteration where κ is drawn uniformly from $1{:}b$.

Often the length of the data will require batches of unequal lengths to be used. To do so, simply take $\Pr(\kappa) = |B_\kappa|/T$, and replace T/M in (15) (in the main paper) with $T/|B_\kappa|$.

Pre-Training We found that pre-training our variational approximation to sensible initial values reduced the training time. A general framework for this is to train $q(\theta; \phi_\theta)$ to be close to the prior, and $q(x|\theta; \phi_x)$ to be close to the observations, or some other reasonable initial value. See Section F of for details of how we implemented this in our examples.

One of our examples required more complex pre-training, described below in Section G.2. Although our method does sometimes require such non-trivial tuning choices, so do most other competing methods for Bayesian inference of SSMs (see e.g. Sherlock et al., 2015).

Local Side Information Our local side information vector s_i is made up of:

- Time i.
- Binary variable indicating whether or not $i \in \mathcal{S}$ (i.e. whether there is an observation of x_i).
- Vector of observations y_i if $i \in \mathcal{S}$. Replaced by the next recorded observation vector if $i \notin \mathcal{S}$, or by a vector of zeros if there is no next observation.
- Time until next observation (omitted in settings where every i has an observation).
- Binary variable indicating whether $i \in 0{:}T$ (as the s_i receptive field can stretch beyond this).

Choice of ℓ' Throughout we use $\ell' = 10$. We found that this relatively short receptive field length for local side information was sufficient to give good results for our examples.

F EXPERIMENTAL DETAILS

This section lists tuning choices for our examples. In all of our examples we set both n (number of samples used in ELBO gradient estimate) and M (batch length) equal to 50, and use $m = 3$ affine layers in our flow for x.

Each affine layer has a CNN with 4 layers of one-dimensional convolutional networks. Each intermediate layer has 50 filters, uses ELU activation and batch normalisation (except the output layer). Before being injected to the first CNN layer, side information vectors (see Section D) are processed through a feed-forward network to produce an encoded vector of length 50. We use a vanilla feed-forward network of 50 hidden units by 3 layers, with ELU activation.

We use the AdaMax optimiser with tuning parameters $\beta_1 = 0.95$ (non-default choice) and $\beta_2 = 0.999$ (default choice). See the tables below for learning rates used.

Each experiment uses a small number of pre-training SGD iterations for ϕ_θ optimising $E_{\theta\sim q}[p(\theta)]$, the expected prior density. We separately pre-train ϕ_x to optimise an objective detailed in the tables below. As discussed above (Section E), where possible we aim to initialise x to be close to the observations, or some other reasonable initial value.

Choices specific to each experiment are listed below.

AR(1)

Learning rate	10^{-3}
Pre-training for x	500 iterations minimising $E_{\theta,x\sim q}[\|\|x - \hat{y}\|\|_2]$, where $\hat{y}$ is the observed data.
ℓ	10

LOTKA-VOLTERRA: DATA SETTING (A)

Learning rate	10^{-3}
Pre-training for x	500 iterations minimising $E_{\theta,x\sim q}[\|\|x - \hat{y}\|\|_2]$, where $\hat{y}$ is linear interpolation of the data.
ℓ	20

LOTKA-VOLTERRA: DATA SETTING (B)

Learning rate	5×10^{-4}
Pre-training for x	500 iterations maximising $E_{x\sim q}[p(x\|\theta^*)]$ where $\theta^* = (0.5, 0.0025, 0.3)$. See Section G.2 for more details.
ℓ	20

FITZHUGH-NAGUMO

Learning rate	5×10^{-4}
Pre-training for x	500 iterations minimising $E_{\theta,x\sim q}[\|\|x\|\|_2]$. Here the model has some unobserved components, so we cannot initialise x close to the observations. Instead we simply encourage x to take small initial values.
ℓ	20

G LOTKA-VOLTERRA DETAILS

Here we discuss some methodology specific to the Lotka-Volterra example in more detail.

G.1 RESTRICTING x TO POSITIVE VALUES

For our Lotka-Volterra model, $x_i = (u_i, v_i)$ represents two population sizes. Negative values don't have a natural interpretation, and also cause numerical errors in the model i.e. the matrix β in (17) may no longer be positive definite so that a Cholesky factor, required in (2), is not available[1].

Therefore we wish to restrict the support of $q(x|\theta; \phi_x)$ to positive values. We so by the following method, which can be applied more generally, beyond this specific model. We add a final elementwise softplus bijection to our nMA model. Let $\tilde{x}$ be the output before this final bijection. The log density (7) gains an extra term to become

$$\log q(x) = \sum_{i=1}^{T} \varphi(z_i) - \sum_{k=1}^{d} \sum_{j=1}^{m} \sum_{i=1}^{T} \log \sigma_{ik}^{j} - \sum_{k=1}^{d} \sum_{i=1}^{T} \gamma(\tilde{x}_{ik}), \tag{11}$$

where γ is the derivative of the softplus function (i.e. the logistic function). The ELBO calculations remain unchanged except for taking

$$\lambda_i = \varphi(z_i) - \sum_{k=1}^{d} \sum_{j=1}^{m} \log \sigma_{ik}^{j} - \sum_{k=1}^{d} \gamma(\tilde{x}_{ik}). \tag{12}$$

We implement our method as before with this modification to λ_i.

G.2 MULTIPLE MODES AND PRE-TRAINING

Observation setting (b) of our Lotka-Volterra example has multiple posterior modes. Without careful initialisation of $q(x|\theta)$, the variational approach typically finds a mode with high frequency oscillations in x. An example is displayed in Figure 1. The corresponding estimated maximum a-posteriori parameter values are $\hat{\theta} = (4.428, 0.029, 2.957)$.

Ideally we would aim to find the most likely modes and evaluate their posterior probabilities, but this is infeasible for our method. (It could be feasible to design a reversible jump MCMC algorithm, following Green, 1995, to do this, but we are unaware of such a method for this application.) Instead we attempt to constrain our analysis to find the mode we expect to be most plausible – that giving a single oscillation between each pair of data points. It is difficult to encode this belief in our prior distribution, so instead we use pretraining so that VI concentrates on this mode. This is comparable to the common MCMC tuning strategy of choosing a plausible initial value.

We use 500 pretraining iterations maximising the likelihood of $p(x|\theta^*)$, where $\theta^* = 0.1\hat{\theta}$. The basis for this choice is that periodic Lotka-Volterra dynamics roughly correspond to cycles in (u, v) space. Multiplying θ, the rate constants of the dynamics, by ξ should give similar dynamics but increase the frequency by a factor of ξ. Based on Figure 1 we wish to reduce the frequency by a factor of 10, so we choose $\xi = 0.1$. Using this pre-training approach, we obtain the results shown in the main paper (Figure 3), corresponding to a more plausible mode.

[1]Note all equation references in Section G.1 are to the main paper.

References

Peter J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732, 1995.

Durk P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, 2013.

Chris Sherlock, Alexandre H. Thiery, Gareth O. Roberts, and Jeffrey S. Rosenthal. On the efficiency of pseudo-marginal random walk Metropolis algorithms. *The Annals of Statistics*, 43(1):238–275, 2015.

Mike West and Jeff Harrison. *Bayesian forecasting and dynamic models*. Springer, 2006.

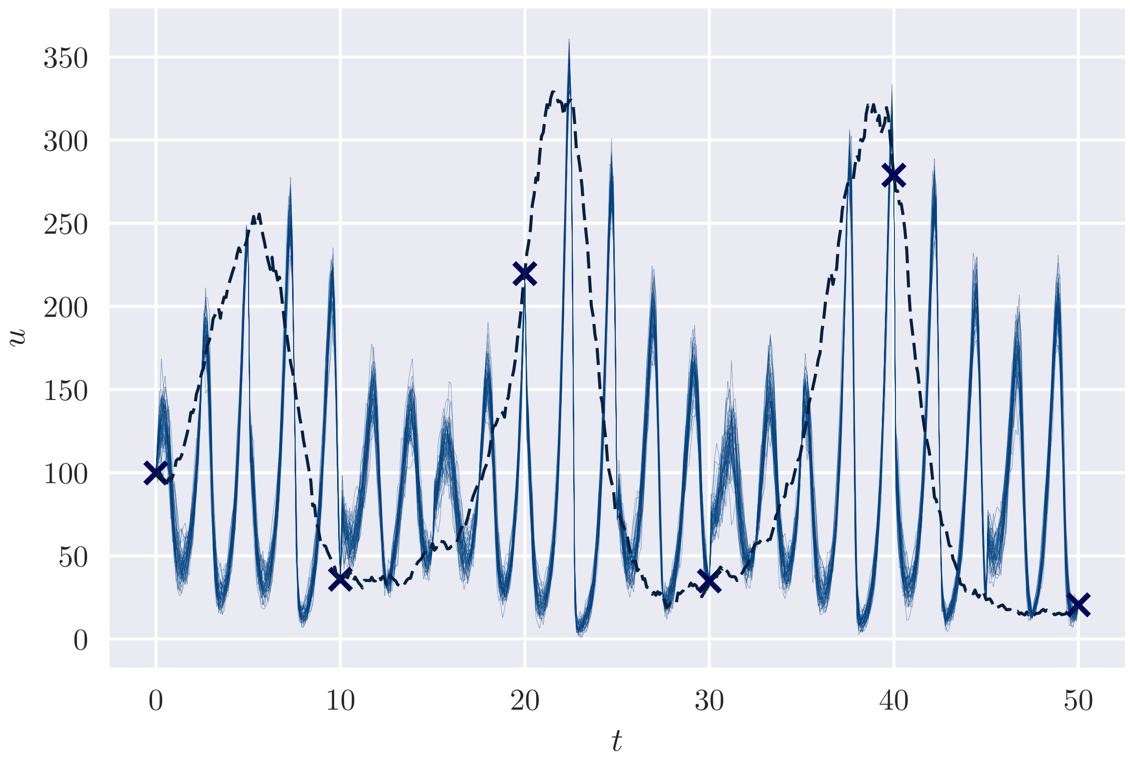

Figure 1: Lotka-Volterra results finding a high-frequency mode. This shows the latent path (dashed line), available observations (crosses) and 50 samples of the variational posterior for x. Here, for ease of presentation, we present results for u only. The horizontal axis shows $t = 0.1i$.