Differentiable Algorithm for Marginalising Changepoints

Hyoungjin Lim, Gwonsoo Che, Wonyeol Lee, Hongseok Yang
School of Computing
KAIST, South Korea
{lmkmkr, gche, wonyeol, hongseok.yang}@kaist.ac.kr

Abstract

We present an algorithm for marginalising changepoints in time-series models that assume a fixed number of unknown changepoints. Our algorithm is differentiable with respect to its inputs, which are the values of latent random variables other than changepoints. Also, it runs in time $\mathcal{O}(mn)$ where n is the number of time steps and m the number of changepoints, an improvement over a naive marginalisation method with $\mathcal{O}(n^m)$ time complexity. We derive the algorithm by identifying quantities related to this marginalisation problem, showing that these quantities satisfy recursive relationships, and transforming the relationships to an algorithm via dynamic programming. Since our algorithm is differentiable, it can be applied to convert a model non-differentiable due to changepoints to a differentiable one, so that the resulting models can be analysed using gradient-based inference or learning techniques. We empirically show the effectiveness of our algorithm in this application by tackling the posterior inference problem on synthetic and real-world data.

1 Introduction

Time-series data from, for instance, econometrics, medical science, and political science (Erdman and Emerson 2008; Lio and Vannucci 2000; Spokoiny and others 2009; Haynes, Eckley, and Fearnhead 2017; Reeves et al. 2007; Lung-Yut-Fong, Lévy-Leduc, and Cappé 2012) often show abrupt regime shifts, so that analysing those data commonly requires reasoning about the moments of these shifts, called changepoints. Two popular reasoning tasks are inferring the number of changepoints and detecting the specific values or distributions of the changepoints. Information found from these tasks enables the use of different statistical models for different segments of the data, identified by changepoints, which leads to accurate analysis of the data. However, due to the discrete nature of changepoints, developing efficient algorithms for the tasks is tricky, and often requires an insight into the structure of a class of models used.

In the paper, we study the problem of marginalising changepoints, which has been under-explored compared with the two tasks mentioned above. We present a differentiable algorithm for marginalising changepoints for a class of time-series models that assume a fixed number of changepoints. Our algorithm runs in $\mathcal{O}(mn)$ time where m is the number of changepoints and n the number of time steps. We do not know of any $\mathcal{O}(mn)$-time algorithm that directly solves this changepoint-marginalisation problem. The class of models handled by our algorithm is broad, including non-Markovian time-series models.

Our marginalisation algorithm is differentiable with respect to its inputs, which enables the use of gradient-based algorithms for posterior inference and parameter learning on changepoint models. Since changepoints are discrete, gradient-based algorithms cannot be applied to these models, unless changepoints are marginalised out. In fact, marginalising discrete variables, such as changepoints, is a trick commonly adopted by the users of the Hamiltonian Monte Carlo algorithm or its variant (Stan Development Team 2018). Our algorithm makes the trick a viable option for changepoint models. Its $\mathcal{O}(mn)$ time complexity ensures low marginalisation overhead. Its differentiability implies that the gradients of marginalised terms can be computed by off-the-shelf automated differentiation tools (Paszke et al. 2017; Abadi et al. 2016). In the paper, we demonstrate these benefits of our algorithm for posterior inference.

The key insight of our algorithm is that the likelihood of latent variables with changepoints marginalised out can be expressed in terms of quantities that satisfy recursive relationships. The algorithm employs dynamic programming to compute these quantities efficiently. Its $\mathcal{O}(mn)$ time complexity comes from this dynamic programming scheme, and its differentiability comes from the fact that dynamic programming uses only differentiable operations. In our experiments with an inference problem, the algorithm outperforms existing alternatives.

The rest of the paper is organised as follows. In §2, we describe our algorithm and its theoretical properties, and in §3, we explain how this algorithm can be used to learn model parameters from given data. In §4, we describe our experiments where we apply the algorithm to a posterior-inference problem. In §5, we put our results in the context of existing work on changepoint models and differentiable algorithms, and conclude the paper.

2 Marginalisation Algorithm

Let n, m be positive integers with $n \geq m$, and $\mathbb{R}_+$ be the set of positive real numbers. We consider a probabilistic model for n-step time-series data with $m+1$ changepoints, which has the following form. Let $\mathcal{X} \subseteq \mathbb{R}^k$ and $\mathcal{Z} \subseteq \mathbb{R}^l$.

$x_{1:n} \in \mathcal{X}^n$ — data points over n time steps.

$w_{1:n} \in \mathbb{R}_+^n$ — w_t expresses a relative chance of the step t becoming a changepoint. $w_n = 1$.

$z_{1:m} \in \mathcal{Z}^m$ — latent parameters deciding the distribution of the data points $x_{1:n}$.

$\tau_{0:m} \in \mathbb{N}^{m+1}$ — changepoints. $0 = \tau_0 < \tau_1 < \cdots < \tau_m = n$.

$$P(\tau_{0:m} \mid w_{1:n}) \triangleq \frac{1}{W} \prod_{i=1}^{m-1} w_{\tau_i} \quad \text{where } W = \sum_{\tau_{0:m}} \prod_{i=1}^{m-1} w_{\tau_i},$$

$$P(x_{1:n}, z_{1:m}, \tau_{0:m} \mid w_{1:n}) \triangleq$$
$$P(z_{1:m}) P(\tau_{0:m} \mid w_{1:n}) \prod_{i=1}^{m} \prod_{(j=\tau_{i-1}+1)}^{\tau_i} P(x_j \mid x_{1:j-1}, z_i).$$

For simplicity, we assume for now that $w_{1:n}$ is fixed and its normalising constant W is known. In §2.1, we will show how the assumption can be removed safely.

Our goal is to find an efficient algorithm for computing the likelihood of the data $x_{1:n}$ for the latent $z_{1:m}$, which involves marginalising the changepoints $\tau_{0:m}$ as shown below:

$$P(x_{1:n} \mid z_{1:m}, w_{1:n}) = \sum_{\tau_{0:m}} P(x_{1:n}, \tau_{0:m} \mid z_{1:m}, w_{1:n}). \quad (1)$$

Note that summing the terms in (1) naively is not a viable option because the number of the terms grows exponentially in the number of changepoints (i.e., $\mathcal{O}(n^{m-1})$).

Our algorithm computes the sum in (1) in $\mathcal{O}(mn)$ time. Two key ideas behind the algorithm are to rewrite the sum in terms of recursively expressible quantities, and to compute these quantities efficiently using dynamic programming.

For integers k, t with $1 \leq k < m$ and $m - 1 \leq t < n$, let

$$T_{k,t} \triangleq \{\tau_{0:m} \mid \tau_{0:m} \text{ changepoints, } \tau_{m-1} = t, \text{ and } 1 + \tau_i = \tau_{i+1} \text{ for all } i \text{ with } k \leq i < m-1\},$$

$$L_{0,t} \triangleq 0, \qquad L_{k,t} \triangleq \sum_{\tau_{0:m} \in T_{k,t}} P(x_{1:n}, \tau_{0:m} \mid z_{1:m}, w_{1:n}),$$

$$R_{m,t} \triangleq 1, \qquad b(k,t) \triangleq t - ((m-1) - k),$$

$$R_{k,t} \triangleq \prod_{j=k}^{m-1} \frac{P(x_{b(j,t)+1} \mid x_{1:b(j,t)}, z_j) \times w_{b(j,t)+1}}{P(x_{b(j,t)+1} \mid x_{1:b(j,t)}, z_{j+1}) \times w_{b(j,t)}}.$$

The first $T_{k,t}$ consists of changepoints $\tau_{0:m}$ such that τ_{m-1} ends at t and $(\tau_k, \tau_{k+1}, \ldots, \tau_{m-1})$ are consecutive. The next $L_{k,t}$ selects the summands in (1) whose changepoints $\tau_{0:m}$ are in $T_{k,t}$. It then sums the selected terms. The $b(k,t)$ computes the value of the k-th changepoint for $1 \leq k < m$ when the changepoints $\tau_k, \tau_{k+1}, \ldots, \tau_{m-1}$ are consecutive and the value of τ_{m-1} is t. The last $R_{k,t}$ is the ratio of the probabilities of the segment $x_{b(k,t)+1:b(m-1,t)+1}$ ($= x_{b(k,t)+1:t+1}$) and the changepoints $\tau_{k:m-1}$ under two different assumptions. The numerator assumes that $\tau_j = b(j,t) + 1$ for all $k \leq j < m$, whereas the denominator assumes that $\tau_j = b(j,t)$ for all those j. A good heuristic is to view $R_{k,t}$ as the change in the probability of the segment $x_{b(k,t)+1:t+1}$ and the changepoints $\tau_{k:m-1}$ when those changepoints are shifted one step to the right.

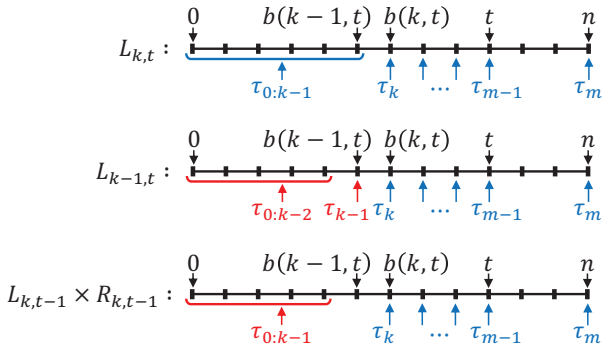


Figure 1: Visualisation of a case split used in Theorem 2.

The next three results formally state what we have promised: the $L_{k,t}$ can be used to express the sum in (1), and the $L_{k,t}$ and the $R_{k,t}$ satisfy recursive relationships.

Proposition 1. *$\sum_{\tau_{0:m}} P(x_{1:n}, \tau_{0:m} \mid z_{1:m}, w_{1:n})$ in (1) is equal to $\sum_{t=m-1}^{n-1} L_{m-1,t}$.*

Proof. Because $\{T_{m-1,t} \mid m-1 \leq t < n\}$ is a partition of the set of all changepoints $\tau_{0:m}$. □

Thus, we can marginalise changepoints by computing $\sum_{t=m-1}^{n-1} L_{m-1,t}$. This begs the question of how to compute the $L_{m-1,t}$'s. The next two results give an answer.

Theorem 2. *For all k, t with $1 \leq k < m$ and $m \leq t < n$,*

$$L_{k,m-1} = P(x_{1:n} \mid z_{1:m}, \tau_{0:m}{=}(0, 1, \ldots, m{-}1, n)) \times P(\tau_{0:m}{=}(0, 1, \ldots, m{-}1, n) \mid w_{1:n}),$$
$$L_{k,t} = L_{k-1,t} + L_{k,t-1} \times R_{k,t-1}.$$

Figure 1 visualises a case split used in the second equation. $L_{k,t}$ is the quantity about changepoints with $\tau_{k:m-1} = (b(k,t), \ldots, t)$. The figure shows that such changepoints can be partitioned into those with $\tau_{k-1} = b(k-1,t)$ and the rest. The first summand $L_{k-1,t}$ computes the contribution of the changepoints in the first partition, and the other summand of the equation that of the changepoints in the second partition.

Proof. By definition, $T_{k,m-1}$ is the singleton set $\{\tau_{0:m} \mid \tau_{0:m} = (0, 1, \ldots, m-1, n)\}$. The first equality in the theorem follows from this and the definition of $L_{k,m-1}$. For the second equality, consider k, t that satisfy the condition in the theorem. We will prove the following two equations:

$$L_{k,t-1} \times R_{k,t-1} = \sum_{\substack{\tau_{0:m} \in T_{k,t} \\ \tau_{k-1} \neq b(k-1,t)}} P(x_{1:n}, \tau_{0:m} \mid z_{1:m}, w_{1:n}), \quad (2)$$

$$L_{k-1,t} = \sum_{\substack{\tau_{0:m} \in T_{k,t} \\ \tau_{k-1} = b(k-1,t)}} P(x_{1:n}, \tau_{0:m} \mid z_{1:m}, w_{1:n}) \,. \quad (3)$$

The desired conclusion follows from these two equations:

$$L_{k-1,t} + L_{k,t-1} \times R_{k,t-1} = \sum_{\tau_{0:m} \in T_{k,t}} P(x_{1:n}, \tau_{0:m} \mid z_{1:m}, w_{1:n}) = L_{k,t} \,.$$

Equation (3) holds since $T_{k-1,t} = \{\tau_{0:m} \in T_{k,t} \mid \tau_{k-1} = b(k-1,t)\}$. Equation (2) is proved as follows. Let

$$\widehat{T}_{k,t} \triangleq \{\tau_{0:m} \in T_{k,t} | \tau_{k-1} \neq b(k-1,t)\},$$

$$\tau'_{0:m} \triangleq (\tau_{0:k-1}, \tau_k - 1, ..., \tau_{m-1} - 1, n) \quad \text{for } \tau_{0:m} \in \widehat{T}_{k,t}.$$

Then, $\{\tau'_{0:m} | \tau_{0:m} \in \widehat{T}_{k,t}\} = T_{k,t-1}$. For every $\tau_{0:m} \in \widehat{T}_{k,t}$,

$$\begin{aligned}
&\frac{P(x_{1:n}, \tau_{0:m} | z_{1:m}, w_{1:n})}{P(x_{1:n}, \tau'_{0:m} | z_{1:m}, w_{1:n})} \\
&= \frac{P(\tau_{0:m} | w_{1:n})}{P(\tau'_{0:m} | w_{1:n})} \times \frac{P(x_{1:n} | \tau_{0:m}, z_{1:m})}{P(x_{1:n} | \tau'_{0:m}, z_{1:m})} \\
&= \prod_{i=k}^{m-1} \frac{w_{\tau_i}}{w_{\tau'_i}} \times \prod_{i=k}^{m-1} \frac{P(x_{\tau_i} | x_{1:\tau_i - 1}, z_i)}{P(x_{\tau'_i + 1} | x_{1:\tau'_i}, z_{i+1})} \\
&= \prod_{i=k}^{m-1} \frac{w_{b(i,t)}}{w_{b(i,t)-1}} \times \prod_{i=k}^{m-1} \frac{P(x_{b(i,t)} | x_{1:b(i,t)-1}, z_i)}{P(x_{b(i,t)} | x_{1:b(i,t)-1}, z_{i+1})} \\
&= R_{k,t-1}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\sum_{\tau_{0:m} \in \widehat{T}_{k,t}} P(x_{1:n}, \tau_{0:m} \mid z_{1:m}, w_{1:n}) \\
&= \sum_{\tau_{0:m} \in \widehat{T}_{k,t}} P(x_{1:n}, \tau'_{0:m} \mid z_{1:m}, w_{1:n}) \times R_{k,t-1} \\
&= \sum_{\tau_{0:m} \in T_{k,t-1}} P(x_{1:n}, \tau_{0:m} \mid z_{1:m}, w_{1:n}) \times R_{k,t-1} \\
&= L_{k,t-1} \times R_{k,t-1}.
\end{aligned}$$

□

Proposition 3. *For all k, t with $1 \leq k < m$ and $m - 1 \leq t < n$,*

$$R_{k,t} = R_{k+1,t} \times \frac{P(x_{b(k,t)+1} \mid x_{1:b(k,t)}, z_k) \times w_{b(k,t)+1}}{P(x_{b(k,t)+1} \mid x_{1:b(k,t)}, z_{k+1}) \times w_{b(k,t)}} \,.$$

Proof. Immediate from the definition of $R_{k,t}$. □

When combined with the idea of dynamic programming, the recursive formulas in the propositions and the theorem give rise to an $\mathcal{O}(mn)$ algorithm for marginalising changepoints. We spell out this algorithm in Algorithm 1.

Theorem 4. *Algorithm 1 computes $P(x_{1:n} \mid z_{1:m}, w_{1:n})$, where m is the number of changepoints and n is the number of steps in given data. Moreover, the algorithm runs in $\mathcal{O}(mn)$ time, if computing $P(x_j \mid x_{1:j-1}, z_i)$ for all $1 \leq i \leq m$ and $2 \leq j \leq n$ takes $\mathcal{O}(mn)$ time (i.e., $P(x_j \mid x_{1:j-1}, z_i)$ can be computed in $\mathcal{O}(1)$ amortised time).*

Algorithm 1 Algorithm for marginalising changepoints.

Input: (i) integer m; (ii) weights $w_{1:n}$ with $w_n = 1$; (iii) normalising constant W for $P(\tau_{0:m} | w_{1:n})$; (iv) latent variables $z_{1:m}$; (v) time-series data $x_{1:n}$ with $1 \leq m \leq n$

Output: likelihood $P(x_{1:n} \mid z_{1:m}, w_{1:n})$ where changepoints $\tau_{0:m}$ are marginalised

1: **for** $t \leftarrow m-1$ to $n-1$ **do**
2: $\quad L_{0,t} \leftarrow 0$; $R_{m,t} \leftarrow 1$
3: $L_{1,m-1} \leftarrow P(x_{1:n} | z_{1:m}, \tau_{0:m} = (0, 1, \ldots, m-1, n)) \times P(\tau_{0:m} = (0, 1, \ldots, m-1, n) | w_{1:n})$
4: **for** $k \leftarrow 2$ to $m-1$ **do**
5: $\quad L_{k,m-1} \leftarrow L_{1,m-1}$
6: $P_{i,j} \leftarrow P(x_j | x_{1:j-1}, z_i)$ for $1 \leq i \leq m$ and $2 \leq j \leq n$
7: **for** $t \leftarrow m-1$ to $n-1$ **do**
8: $\quad$ **for** $k \leftarrow m-1$ downto 1 **do**
9: $\quad\quad R_{k,t} \leftarrow R_{k+1,t} \times \frac{P_{b(k,t)+1,k} \times w_{b(k,t)+1}}{P_{b(k,t)+1,k+1} \times w_{b(k,t)}}$
10: **for** $t \leftarrow m$ to $n-1$ **do**
11: $\quad$ **for** $k \leftarrow 1$ to $m-1$ **do**
12: $\quad\quad L_{k,t} \leftarrow L_{k-1,t} + L_{k,t-1} \times R_{k,t-1}$
13: $L \leftarrow 0$
14: **for** $t \leftarrow m-1$ to $n-1$ **do**
15: $\quad L \leftarrow L + L_{m-1,t}$
return L

Proof. The correctness follows from Propositions 1 and 3 and Theorem 2. We analyse the run time as follows. The line 3 computes the RHS in $\mathcal{O}(n)$. The line 6 runs in $\mathcal{O}(mn)$ by the assumption. In the rest of the algorithm, nested loops and other loops iterate $\mathcal{O}(mn)$ times, and each line inside the loops runs in $\mathcal{O}(1)$. So, the algorithm runs in $\mathcal{O}(mn)$. □

Theorem 5. *When $P(x_j \mid x_{1:j-1}, z_i)$ is differentiable with respect to $x_{1:j}$ and z_i, the result of Algorithm 1 is also differentiable with respect to $x_{1:n}$ and $z_{1:m}$, and can be computed by applying automated differentiation to the algorithm.*

Proof. When $P(x_j \mid x_{1:j-1}, z_i)$ is differentiable with respect to $x_{1:j}$ and z_i, the likelihood $P(x_{1:n} \mid z_{1:m}, w_{1:n})$ is differentiable with respect to $x_{1:n}$ and $z_{1:m}$. So, the correctness of Algorithm 1 in Theorem 4 implies the claimed differentiability. The other claim about the use of automated differentiation holds because Algorithm 1 does not use any non-differentiable operations such as if statements. □

2.1 Computation of normalising constant W

So far we have assumed that weights $w_{1:n}$ are fixed and the normalising constant W for $P(\tau_{0:m} \mid w_{1:n})$ is known. We now discharge the assumption. We present an algorithm for computing W for given $w_{1:n}$. The algorithm uses dynamic programming, runs in $\mathcal{O}(mn)$ time, and is differentiable: the gradient of W with respect to $w_{1:n}$ can be computed by applying automated differentiation to the algorithm.

For all k and t with $1 \leq k < m$ and $0 \leq t < n$, let $S_{k,t} \triangleq \sum_{\tau_{0:k}, \tau_k \leq t} \prod_{i=1}^{k} w_{\tau_i}$ and $S_{0,t} \triangleq 1$. Note that $W = S_{m-1,n-1}$. So, it suffices to design an algorithm for computing $S_{k,t}$. The next proposition describes how to do it.

Proposition 6. *For all k, t with $1 \leq k < m$ and $k \leq t < n$, we have $S_{k,t} = S_{k,t-1} + S_{k-1,t-1} \times w_t$ and $S_{k,k-1} = 0$.*

The recurrence relation for $S_{k,t}$ in Proposition 6 yields a dynamic-programming algorithm for computing W that runs in $\mathcal{O}(mn)$ time. The standard implementation of the algorithm does not use any non-differentiable operations. So, its gradient can be computed by automated differentiation.

With this result at hand, we remove the assumption that weights $w_{1:n}$ are fixed and the normalising constant W for $P(\tau_{0:m} \mid w_{1:n})$ is known a priori. Algorithm 1 no longer receives W as its input. It instead uses the algorithm described above and computes W from given $w_{1:n}$ before starting line 1. Since the computation of W takes $\mathcal{O}(mn)$ time and can be differentiated by automated differentiation, all the aforementioned results on Algorithm 1 (Theorems 4 and 5) still hold, and can be extended to cover the differentiability of Algorithm 1 with respect to $w_{1:n}$.

3 Learning Model Parameters

Our algorithm can extend the scope of gradient-based methods for posterior inference and model learning such that they apply to changepoint models despite their non-differentiability. In this section, we explain the model-learning application. We consider state-space models with changepoints that use neural networks. The goal is to learn appropriate neural-network parameters from given data.

We consider a special case of the model described in §2 that satisfies the following conditions.

1. The latent parameter z_i at $i \in \{1, \ldots, m\}$ has the fixed value e_i in $\{0,1\}^m$ that has 1 at the i-th position and 0 everywhere else. Formally, this means that the prior $P(z_{1:m})$ is the Dirac distribution at $(e_1, e_2, \ldots, e_m)$.
2. The random variable x_j at $j \in \{1, \ldots, n\}$ consists of two parts, $x_j^S \in \mathcal{X}_S$ for the latent state and $x_j^O \in \mathcal{X}_O$ for the observed value. Thus, $x_j = (x_j^S, x_j^O)$ and $\mathcal{X} = \mathcal{X}_S \times \mathcal{X}_O$.
3. The probability distribution $P_\phi(x_j \mid x_{1:j-1}, z_i)$ is parameterised by $\phi \in \mathbb{R}^p$ for some p, and has the form
$$P_\phi(x_j \mid x_{1:j-1}, z_i) = P_\phi(x_j^O \mid x_j^S, z_i) P_\phi(x_j^S \mid x_{1:j-1}^S, z_i).$$
Typically, P_ϕ is defined using a neural network, and ϕ denotes the weights of the network.

When the model satisfies these conditions, we have

$$\begin{aligned} P_\phi(x_{1:n} \mid w_{1:n}) &= \sum_{z_{1:m}} P_\phi(x_{1:n}, z_{1:m} \mid w_{1:n}) \\ &= \sum_{z_{1:m}} P(z_{1:m}) P_\phi(x_{1:n} \mid z_{1:m}, w_{1:n}) \\ &= P_\phi(x_{1:n} \mid (z_{1:m} = e_{1:m}), w_{1:n}). \end{aligned} \quad (4)$$

By the learning of model parameters, we mean the problem of finding ϕ for given observations $x_{1:n}^O$ that makes the log probability of the observations $\log P_\phi(x_{1:n}^O \mid w_{1:n})$ large. A popular approach (Kingma and Welling 2014) is to maximise a lower bound of this log probability, called ELBO, approximately using a version of gradient ascent:

$$\mathrm{ELBO}_{\phi,\theta} \triangleq \mathbb{E}_{Q_\theta(x_{1:n}^S \mid x_{1:n}^O)} \left[\log \frac{P_\phi(x_{1:n}^S, x_{1:n}^O \mid w_{1:n})}{Q_\theta(x_{1:n}^S \mid x_{1:n}^O)} \right] \quad (5)$$

where Q_θ is an approximating distribution for the posterior $P_\phi(x_{1:n}^S \mid x_{1:n}^O, w_{1:n})$ and $\theta \in \mathbb{R}^q$ denotes the parameters of this distribution, typically the weights of a neural network used to implement Q_θ.

Our marginalisation algorithm makes it possible to optimise $\mathrm{ELBO}_{\theta,\phi}$ in (5) by an efficient stochastic gradient-ascent method based on the so called reparameterisation trick (Kingma and Welling 2014; Rezende, Mohamed, and Wierstra 2014; Kucukelbir et al. 2017). Here we use our algorithm with fixed $m, w_{1:n}$ after setting $z_{1:m}$ to $e_{1:m}$.[1] So, only the $x_{1:n} = (x_{1:n}^S, x_{1:n}^O)$ part of its input may vary. To emphasise this, we write $\textsc{Alg}(\phi, x_{1:n}^S, x_{1:n}^O)$ for the result of the algorithm. Also, we make the usual assumption of the reparameterisation trick: there are a θ-independent distribution $Q(\epsilon)$ and a differentiable function $T_\theta(\epsilon, x_{1:n}^O)$ such that $T_\theta(\epsilon, x_{1:n}^O)$ for $\epsilon \sim Q(\epsilon)$ is distributed according to $Q_\theta(x_{1:n}^S \mid x_{1:n}^O)$. The next theorem shows that the gradient of ELBO can be estimated by computing the gradient through the execution of our algorithm via automated differentiation.

Theorem 7. *If $P_\phi(x_j \mid x_{1:j-1}, z_i)$ is differentiable with respect to $x_{1:j}$ and ϕ, so is $\textsc{Alg}(\phi, x_{1:n}^S, x_{1:n}^O)$. In that case, the gradient can be computed by automated differentiation.*

Proof. When $P_\phi(x_j \mid x_{1:j-1}, z_i)$ is differentiable with respect to $x_{1:j}$ and ϕ, the likelihood $P_\phi(x_{1:n} \mid z_{1:m}, w_{1:n})$ is differentiable with respect to $x_{1:n}$ and ϕ. Thus, the correctness of Algorithm 1 in Theorem 4 implies the claimed differentiability. The other claim about the use of automated differentiation comes from the fact that Algorithm 1 does not use any non-differentiable operations. □

Theorem 8. *When $P_\phi(x_j \mid x_{1:j-1}, z_i)$ is differentiable with respect to $x_{1:j}$ and ϕ for all j and i,*

$$\widehat{\mathrm{MRep}} \triangleq \nabla_{\phi,\theta} \log \frac{\textsc{Alg}(\phi, T_\theta(\epsilon, x_{1:n}^O), x_{1:n}^O)}{Q_\theta(T_\theta(\epsilon, x_{1:n}^O) \mid x_{1:n}^O)} \quad \textit{for } \epsilon \sim Q(\epsilon)$$

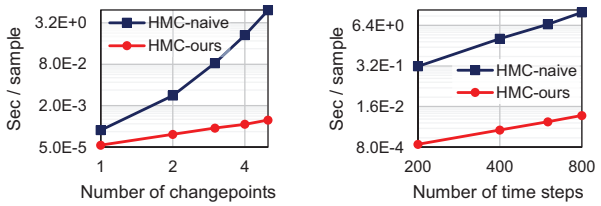
is an unbiased estimate for $\nabla_{\phi,\theta} \mathrm{ELBO}_{\phi,\theta}$, and can be computed via automated differentiation.

Proof. $P_\phi(x_{1:n}^S, x_{1:n}^O | w_{1:n}) = P_\phi(x_{1:n} | (z_{1:m}{=}e_{1:m}), w_{1:n})$ by (4). The RHS of the equation equals $\textsc{Alg}(\phi, x_{1:n}^S, x_{1:n}^O)$ by Theorem 4 and the definition of ALG. So, $\mathrm{ELBO}_{\phi,\theta} = \mathbb{E}_{Q_\theta(x_{1:n}^S \mid x_{1:n}^O)} \left[\log \frac{\textsc{Alg}(\phi, x_{1:n}^S, x_{1:n}^O)}{Q_\theta(x_{1:n}^S \mid x_{1:n}^O)} \right]$. The usual unbiasedness argument of the reparameterisation trick and the differentiability of $\textsc{Alg}(\phi, x_{1:n}^S, x_{1:n}^O)$ with respect to $x_{1:n}$ and ϕ (Theorem 7) give the claimed conclusion. □

4 Experimental Evaluation

As mentioned, another important application of our algorithm is posterior inference. In this section, we report the findings from our experiments with this application, which show the benefits of having an efficient differentiable marginalisation algorithm for posterior inference.

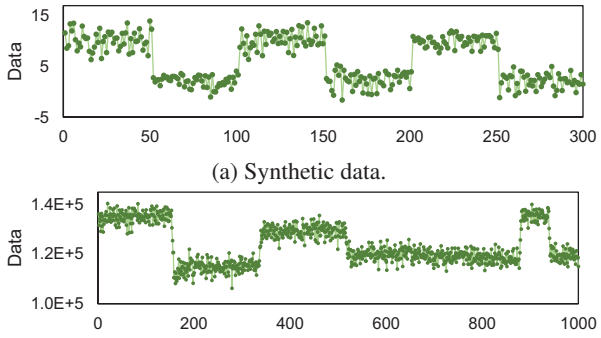
[1] W is computed by the extension of our algorithm in §2.1. In fact, using the same extension, we can even treat $w_{1:n}$ as a part of ϕ, and learn appropriate values for $w_{1:n}$.



(a) Time per sample vs. m when $n = 50$.

(b) Time per sample vs. n when $m = 2$.

Figure 2: Computation time of $\mathsf{HMC}_{\mathsf{naive}}$ and $\mathsf{HMC}_{\mathsf{ours}}$ for different m and n. The x- and y-axes are in log-scale.



(a) Synthetic data.

(b) Part of well-log data.

Figure 3: Synthetic and real-world data for RQ2. The x-axis represents time steps.

Hamiltonian Monte Carlo (HMC) (Duane et al. 1987; Neal 2011) is one of the most effective algorithms for sampling from posterior distributions, especially on high dimensional spaces. However, it cannot be applied to models with changepoints directly. This is because HMC requires that a model have a differentiable density, but changepoint models do not meet this requirement due to discrete changepoints.

One way of addressing this non-differentiability issue is to use our algorithm and marginalise changepoints. Since our algorithm is differentiable, the resulting models have differentiable densities, and we can analyse their posteriors using HMC. We tested this approach experimentally, aiming at answering the following questions:

- RQ1 (Speed): How fast is our marginalisation algorithm when used for HMC?
- RQ2 (Sample quality): Is HMC with our marginalisation algorithm better at generating good posterior samples than other Markov Chain Monte Carlo (MCMC) algorithms that do not use gradients nor marginalisation?

We evaluated different inference algorithms on synthetic and real-world data for $\mathcal{X} = \mathbb{R}$. The synthetic data were generated as follows: we fixed parameters $(n, m^*, \mu^*_{1:m^*}, \sigma^*_{1:m^*}, \tau^*_{0:m^*})$, and then sampled each data point x_j $(1 \leq j \leq n)$ in the i-th segment (i.e., $\tau^*_{i-1} < j \leq \tau^*_i$) independently from a Gaussian distribution with mean μ^*_i and standard deviation σ^*_i. The changepoint model for analysing the synthetic data is: $m = m^*$, $\mathcal{Z} = \mathbb{R} \times \mathbb{R}_+$, $w_{1:n} = (1, \ldots, 1)$, $P(z_i = (\mu_i, \sigma_i)) = \text{Normal}(\mu_i|5, 10) \times \text{LogNormal}(\sigma_i|0, 2)$, and $P(x_j|x_{1:j-1}, z_i = (\mu_i, \sigma_i)) = \text{Normal}(x_j|\mu_i, \sigma_i)$. For the real-world application, we used well-log data (Fearnhead 2006), whose data points represent some physical quantity measured by a probe diving in a wellbore. We took a part of the well-log data by removing outliers and choosing 1000 consecutive data points (Figure 3b). The changepoint model for the well-log data is the same as the above except: $m = 13$ and $P(z_i = (\mu_i, \sigma_i)) = \text{Normal}(\mu_i|120000, 20000) \times \text{LogNormal}(\sigma_i|8.5, 0.5)$.

Our goal is to infer the posterior distribution of latent parameters $z_{1:m}$ and changepoints $\tau_{0:m}$. For this task, we compared four posterior-inference algorithms: $\mathsf{HMC}_{\mathsf{naive}}$, $\mathsf{HMC}_{\mathsf{ours}}$, IPMCMC, and LMH. $\mathsf{HMC}_{\mathsf{naive}}$ (resp. $\mathsf{HMC}_{\mathsf{ours}}$) generates samples as follows: it forms a probabilistic model $P(x_{1:n}, z_{1:m} \,|\, w_{1:n})$ where $\tau_{0:m}$ are marginalised out by a naive marginalisation scheme (resp. by our marginalisation algorithm); samples $z_{1:m}$ from $P(z_{1:m} \,|\, w_{1:n}, x_{1,n})$ by running HMC on $P(x_{1:n}, z_{1:m} \,|\, w_{1:n})$; finally samples $\tau_{0:m}$ from $P(\tau_{0:m} \,|\, w_{1:n}, x_{1:n}, z_{1:m})$ using dynamic programming. IPMCMC and LMH jointly sample $z_{1:m}$ and $\tau_{0:m}$ from $P(z_{1:m}, \tau_{0:m} \,|\, w_{1:n}, x_{1:n})$ by running the variants of the Metropolis-Hastings algorithm called interacting particle MCMC (IPMCMC) (Rainforth et al. 2016) and lightweight Metropolis-Hastings (LMH) (Wingate, Stuhlmüller, and Goodman 2011), respectively. IPMCMC and LMH are applicable to models with discrete or non-differentiable random variables. They neither exploit gradients nor marginalise out any random variables.

For $\mathsf{HMC}_{\mathsf{naive}}$ and $\mathsf{HMC}_{\mathsf{ours}}$, we used the No-U-Turn Sampler (NUTS) (Hoffman and Gelman 2014) in PyStan (Carpenter et al. 2017) with default hyper-parameters, except for adapt_delta $= 0.95$. For IPMCMC and LMH, we used the implementations in Anglican (Wood, van de Meent, and Mansinghka 2014; Tolpin et al. 2016) with default hyper-parameters, except for the following IPMCMC setup: number-of-nodes $= 8$ for both the synthetic and well-log data, and pool $= 8$ for the well-log data.

For RQ1, we compared the time taken to generate a single posterior sample by $\mathsf{HMC}_{\mathsf{naive}}$ and $\mathsf{HMC}_{\mathsf{ours}}$. For RQ2, we compared the quality of posterior samples from $\mathsf{HMC}_{\mathsf{ours}}$, IPMCMC, and LMH, by means of the following quantities: estimates of the first and second moments, the Gelman-Rubin convergence statistic ($\widehat{R}$) (Gelman, Rubin, and others 1992; Brooks and Gelman 1998), and effective sample size (ESS). The experiments were performed on a Ubuntu 16.04 machine with Intel i7-7700 CPU with 16GB of memory.

Results for RQ1. We measured the average time per sample taken by $\mathsf{HMC}_{\mathsf{naive}}$ and $\mathsf{HMC}_{\mathsf{ours}}$ for different numbers of changepoints and time steps: for fixed $n = 50$, we varied $m^* = m$ from 1 to 5, and for fixed $m^* = m = 2$, we varied $n \in \{200, 400, 600, 800\}$. The details of the parameter values we used appear in the extended version of the paper on arXiv. We ran five independent runs of the NUTS algorithm, and averaged the time spent without burn-in samples.

Figures 2a and 2b show how the time depends on m and n, respectively, in the two approaches. In the log-log plots, $\log(\text{time})$ of $\mathsf{HMC}_{\mathsf{ours}}$ is linear in both $\log m$ and $\log n$, due

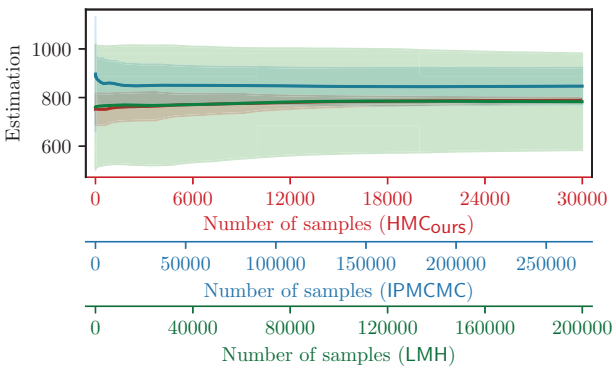


Figure 4: Convergence plots for estimating the sum of the first moments for synthetic data, with HMC$_{\text{ours}}$ (red), IPMCMC (blue), and LMH (green). Each x-axis represents the number of samples generated by the corresponding procedure, and the y-axis denotes estimated values. Each darker line shows the mean value at each point, and the corresponding error band around it shows the standard deviation.

to its time complexity $\mathcal{O}(mn)$. On the other hand, $\log(\text{time})$ of HMC$_{\text{naive}}$ is exponential in $\log m$, and linear in $\log n$ yet with a slope nearly two times larger than that for HMC$_{\text{ours}}$, because of its time complexity $\mathcal{O}(n^m)$. Overall, the results show that HMC$_{\text{naive}}$ quickly becomes infeasible as the number of changepoints or time steps grows, but HMC$_{\text{ours}}$ avoids such an upsurge by virtue of having the linear relationship between the two varying factors and the time per sample.

Results for RQ2. Figure 3 shows the synthetic and real-world data used in answering RQ2. The synthetic data was generated with parameters $n = 300$, $m^* = 6$, $\mu^*_{1:m^*} = (10, 2, 10, 2, 10, 2)$, $\sigma^*_{1:m^*} = (1.8, 1.1, 1.7, 1.5, 1.2, 1.3)$, and $\tau^*_{0:m^*} = (0, 50, 100, 150, 200, 250, 300)$.

For each chain of HMC$_{\text{ours}}$, we generated 30K samples with random initialisation (when possible) after burning in 1K samples. We computed $\widehat{R}$ and ESS for each latent parameter and changepoint using three chains, and repeated this five times as the $\widehat{R}$ and ESS results varied across different runs. We also estimated the sum of the first moments of $(z_{1:m}, \tau_{1:m-1})$ and that of the second moments of them using the same 15 chains.[2] The same setup was applied to IPMCMC and LMH except the following: since they sample faster than HMC$_{\text{ours}}$, we let IPMCMC and LMH generate 270K (resp. 1855K) and 200K (resp. 1750K) samples, respectively, for synthetic data (resp. well-log data) so that every run of them spends more time than the corresponding slowest HMC$_{\text{ours}}$ run.

We first discuss results on synthetic data. Figure 4 shows the estimates for the sum of the first moments by HMC$_{\text{ours}}$, IPMCMC, and LMH. HMC$_{\text{ours}}$ shows a gradual trend towards convergence, while IPMCMC and LMH exhibit substantial variation across runs without convergence. We obtained similar results for the sum of the second moments (see the extended paper on arXiv). Table 1 shows the ranges of the time taken by the Markov chains in Figure 6, and the ranges of the estimates from the chains.

[2]Concretely, we estimated $\mathbb{E}_{P(z_{1:m}, \tau_{1:m} \mid x_{1:n})}\left[\left(\sum_{i=1}^m \mu_i + \sigma_i + \tau_i\right) - n\right]$ and $\mathbb{E}_{P(z_{1:m}, \tau_{1:m} \mid x_{1:n})}\left[\left(\sum_{i=1}^m \mu_i^2 + \sigma_i^2 + \tau_i^2\right) - n^2\right]$.

Table 1: The ranges of the time (sec) taken by the three approaches and the ranges of the estimates computed by them, for synthetic data. For the estimated sum of the first/second moments (i.e., the third/fourth row), we computed the values at 510.8 sec (the minimum time taken among all the runs) from each Markov chain, assuming that generating each sample (in a chain) took an equal amount of time.

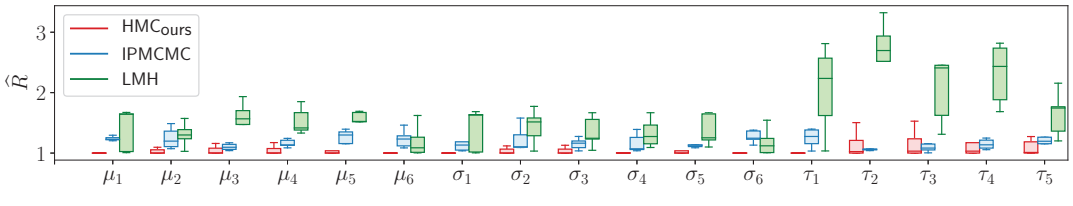
	HMC$_{\text{ours}}$	IPMCMC	LMH
Time	[510.8, 1043.9]	[1053.2, 1170.6]	[1092.2, 1105.2]
1st	[746.9, 794.2]	[651.3, 993.8]	[415.4, 1164.0]
2nd	[1.28E+05, 1.39E+05]	[1.12E+05, 1.89E+05]	[3.56E+04, 2.62E+05]

Figure 5a shows the $\widehat{R}$ values from HMC$_{\text{ours}}$, IPMCMC, and LMH. Three out of five experiments with HMC$_{\text{ours}}$ were satisfactory in the sense that the $\widehat{R}$ statistics for *all* the latent $(z_{1:m}, \tau_{1:m-1})$ were between 0.9 and 1.1. Though the $\widehat{R}$ statistics for some of the latent were over 1.1 in the other two experiments, most of the $\widehat{R}$ values were less than or close to 1.1. On the other hand, none of the IPMCMC and LMH experiments placed $\widehat{R}$ values for all the latent, within the interval. Also the values were farther from the interval.

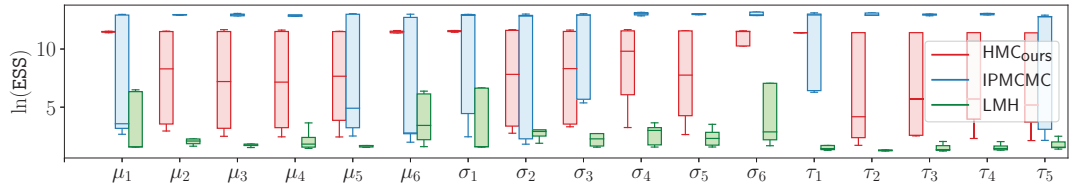
Figure 5b shows $\ln(\text{ESS})$ from HMC$_{\text{ours}}$, IPMCMC, and LMH in a similar manner. HMC$_{\text{ours}}$ produced significantly higher ESS values than LMH, demonstrating that HMC$_{\text{ours}}$ draws samples more effectively than LMH within a fixed amount of time. However, HMC$_{\text{ours}}$ was not superior in ESS to IPMCMC despite the excellence in $\widehat{R}$. We conjecture that this is due to IPMCMC running eight parallel nodes independently, each with two particles to propose samples.

For well-log data, HMC$_{\text{ours}}$ similarly outperformed the other two in terms of convergence, $\widehat{R}$, and ESS; $\widehat{R}$ for *all* the latent $(z_{1:m}, \tau_{1:m-1})$ were between 0.9 and 1.1 in *all* five experiments. One exception is that IPMCMC showed much higher ESS than HMC$_{\text{ours}}$, although it failed to converge (Figure 6). We think that this is again due to IPMCMC's tendency of generating independent samples. The full results are in the extended paper on arXiv.

We remark that HMC$_{\text{ours}}$ performed poorly on well-log data with the same model but smaller m (e.g., 6 instead of 13). According to our manual inspection, this was because HMC$_{\text{ours}}$ in this case got stuck at some modes, failing to generate samples from other modes. We think that increasing m (up to some point) lessens the barriers between modes in the marginal space; for large enough m, only a small amount of $P(z_{1:m} \mid x_{1:n}, w_{1:n})$ should be reduced to change some of $z_{1:m}$. One practical lesson is that having enough changepoints may help analyse real-world data using Bayesian inference and changepoint models.



(a) $\widehat{R}$.



(b) ln(ESS).

Figure 5: $\widehat{R}$ and ESS from HMC$_{\text{ours}}$ (red), IPMCMC (blue), and LMH (green) for synthetic data. The x-axis denotes the latent parameters and changepoints, and the y-axis $\widehat{R}$ or ln(ESS) values.

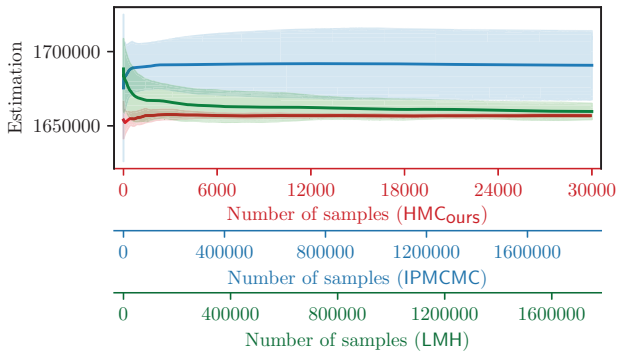


Figure 6: Convergence plots for estimating the sum of the first moments for well-log data. See the caption of Figure 4 for details.

5 Related Work and Conclusion

Related work. Modelling and reasoning about time-series data with changepoints is a well-established topic in statistics and machine learning (Eckley, Fearnhead, and Killick 2011; Truong, Oudre, and Vayatis 2018). We discuss two lines of research most relevant to ours. The first is the work by Fearnhead and his colleagues (Fearnhead 2006; 2005; Fearnhead and Liu 2007), which is further extended to multi-dimensional time-series data (Xuan 2007). Fearnhead (2006) proposed an $\mathcal{O}(n^2)$-time algorithm for generating changepoint positions from the posterior of a given changepoint model in a particular form, where n is the number of time steps. Their algorithm also uses a form of dynamic programming on certain recursive formulas, but it does not target at marginalisation. Its conversion for marginalisation is possible, but inherits this $\mathcal{O}(n^2)$ time complexity. The other work is Chib (1995)'s technique for estimating the model evidence of changepoint models (Chib 1998), whose properties, such as sensitivity on chosen parameters, is analysed by Bauwens and Rombouts (2012). The technique is based on Gibbs sampling, and it is unclear whether the technique leads to a differentiable algorithmic component that can be used in the context of gradient-based algorithms.

The observation that the summation version of dynamic programming is differentiable is a folklore. For instance, Eisner (2016) points out the differentiability of the inside algorithm, which is a classic dynamic-programming-based algorithm in natural language processing (NLP). He then explains how to derive several well-known NLP algorithms by differentiating the inside algorithm or its variants. However, we do not know of existing work that uses such dynamic programming algorithms for the type of application we consider in the paper: converting non-differentiable models to differentiable models via marginalisation in the context of posterior inference and model learning. The optimisation version of dynamic programming is not differentiable, and its differentiable relaxation has been studied recently (Corro and Titov 2019; Mensch and Blondel 2018).

Conclusion. We presented a differentiable $\mathcal{O}(mn)$-time algorithm for marginalising changepoints in time-series models, where m is the number of changepoints and n the number of time steps. The algorithm can be used to convert a class of non-differentiable time-series models to differentiable ones, so that the resulting models can be analysed by gradient-based techniques. We described two applications of this conversion, posterior inference with HMC and model-parameter learning with reparameterisation gradient estimator, and experimentally showed the benefits of using the algorithm in the former posterior-inference application.

Acknowledgements. The authors were supported by the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (NRF-2018R1A5A1059921), and also by Next-Generation Information Computing Develop-

ment Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (2017M3C4A7068177).

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; Kudlur, M.; Levenberg, J.; Monga, R.; Moore, S.; Murray, D. G.; Steiner, B.; Tucker, P. A.; Vasudevan, V.; Warden, P.; Wicke, M.; Yu, Y.; and Zheng, X. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*, 265–283.

Bauwens, L., and Rombouts, J. V. 2012. On marginal likelihood computation in change-point models. *Computational Statistics & Data Analysis* 56(11):3415–3429.

Brooks, S. P., and Gelman, A. 1998. General methods for monitoring convergence of iterative simulations. *J. Comput. Graph. Stat.* 7(4):434–455.

Carpenter, B.; Gelman, A.; Hoffman, M. D.; Lee, D.; Goodrich, B.; Betancourt, M.; Brubaker, M.; Guo, J.; Li, P.; and Riddell, A. 2017. Stan: A probabilistic programming language. *Journal of Statistical Software* 76(1).

Chib, S. 1995. Marginal likelihood from the gibbs output. *J. Am. Stat. Assoc.* 90(432):1313–1321.

Chib, S. 1998. Estimation and comparison of multiple change-point models. *J. Econometrics.* 86(2):221–241.

Corro, C., and Titov, I. 2019. Differentiable perturb-and-parse: Semi-supervised parsing with a structured variational autoencoder. In *ICLR*.

Duane, S.; Kennedy, A. D.; Pendleton, B. J.; and Roweth, D. 1987. Hybrid Monte Carlo. *Physics Letters B* 195:216–222.

Eckley, I. A.; Fearnhead, P.; and Killick, R. 2011. Analysis of changepoint models. In Barber, D.; Cemgil, A. T.; and Chiappa, S., eds., *Bayesian Time Series Models*. Cambridge University Press. chapter 10, 205—224.

Eisner, J. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Workshop on Structured Prediction for NLP@EMNLP*, 1–17.

Erdman, C., and Emerson, J. W. 2008. A fast bayesian change point analysis for the segmentation of microarray data. *Bioinformatics* 24(19):2143–2148.

Fearnhead, P., and Liu, Z. 2007. On-line inference for multiple changepoint problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 69(4):589–605.

Fearnhead, P. 2005. Exact bayesian curve fitting and signal segmentation. *IEEE Transactions on Signal Processing* 53(6):2160–2166.

Fearnhead, P. 2006. Exact and efficient bayesian inference for multiple changepoint problems. *Statistics and Computing* 16(2):203–213.

Gelman, A.; Rubin, D. B.; et al. 1992. Inference from iterative simulation using multiple sequences. *Statistical Science* 7(4):457–472.

Haynes, K.; Eckley, I. A.; and Fearnhead, P. 2017. Computationally efficient changepoint detection for a range of penalties. *J. Comput. Graph. Stat.* 26(1):134–143.

Hoffman, M. D., and Gelman, A. 2014. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *JMLR* 15(1):1593–1623.

Kingma, D. P., and Welling, M. 2014. Auto-Encoding Variational Bayes. In *ICLR*.

Kucukelbir, A.; Tran, D.; Ranganath, R.; Gelman, A.; and Blei, D. M. 2017. Automatic differentiation variational inference. *JMLR* 18(1):430–474.

Lio, P., and Vannucci, M. 2000. Wavelet change-point prediction of transmembrane proteins. *Bioinformatics* 16(4):376–382.

Lung-Yut-Fong, A.; Lévy-Leduc, C.; and Cappé, O. 2012. Distributed detection/localization of change-points in high-dimensional network traffic data. *Statistics and Computing* 22(2):485–496.

Mensch, A., and Blondel, M. 2018. Differentiable dynamic programming for structured prediction and attention. In *ICML*, 3459–3468.

Neal, R. M. 2011. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo* 2(11):2.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

Rainforth, T.; Naesseth, C.; Lindsten, F.; Paige, B.; Vandemeent, J.-W.; Doucet, A.; and Wood, F. 2016. Interacting particle markov chain monte carlo. In *ICML*, 2616–2625.

Reeves, J.; Chen, J.; Wang, X. L.; Lund, R.; and Lu, Q. Q. 2007. A review and comparison of changepoint detection techniques for climate data. *Journal of Applied Meteorology and Climatology* 46(6):900–915.

Rezende, D. J.; Mohamed, S.; and Wierstra, D. 2014. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *ICML*, 1278–1286.

Spokoiny, V., et al. 2009. Multiscale local change point detection with applications to value-at-risk. *The Annals of Statistics* 37(3):1405–1436.

Stan Development Team. 2018. *Stan Modeling Language Users Guide and Reference Manual*. Version 2.18.0.

Tolpin, D.; van de Meent, J.-W.; Yang, H.; and Wood, F. 2016. Design and implementation of probabilistic programming language anglican. In *IFL*, 6:1–6:12.

Truong, C.; Oudre, L.; and Vayatis, N. 2018. Selective review of offline change point detection methods. *ArXiv* abs/1801.00718.

Wingate, D.; Stuhlmüller, A.; and Goodman, N. 2011. Lightweight implementations of probabilistic programming languages via transformational compilation. In *AISTATS*.

Wood, F.; van de Meent, J. W.; and Mansinghka, V. 2014. A new approach to probabilistic programming inference. In *AISTATS*, 1024–1032.

Xuan, X. 2007. Bayesian inference on change point problems. Master's thesis, The University of British Columbia, Vancouver, Canada.