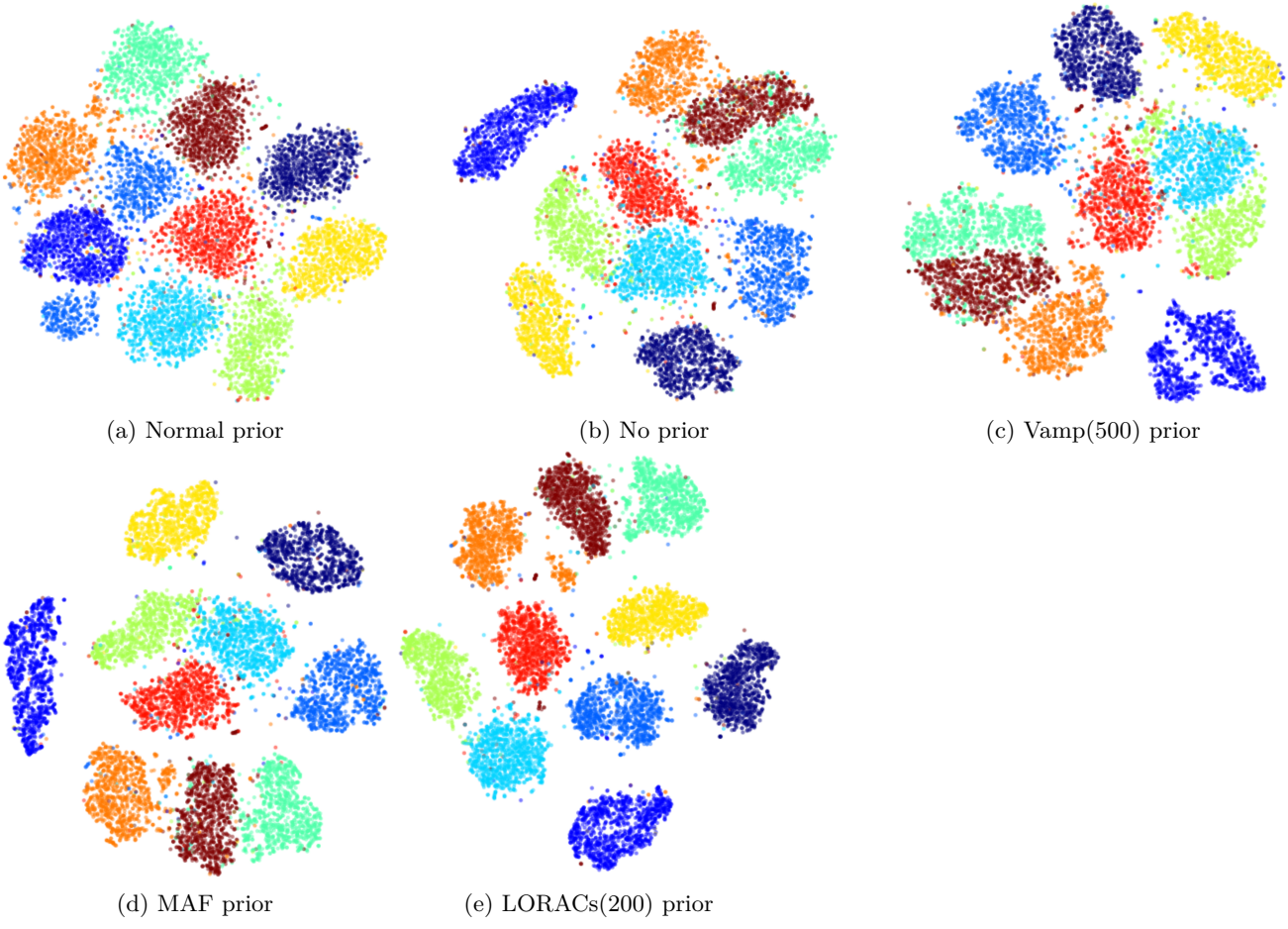
The LORACs Prior for VAEs: Letting the Trees Speak for the Data - Supplement

A Additional visualizations



(a) Normal prior (b) No prior (c) Vamp(500) prior

(d) MAF prior (e) LORACs(200) prior

Figure A.10: TSNE visualizations of the latent space of the MNIST test set with various prior distributions, color-coded according to class.

Figure A.11: A TSNE visualization of the latent space for the TMC(200) model with inducing points and one sample from $q(\tau; s_{1:M})$ plotted. Internal nodes are visualized by computing their expected posterior values, and branches are plotted in 2-d space.

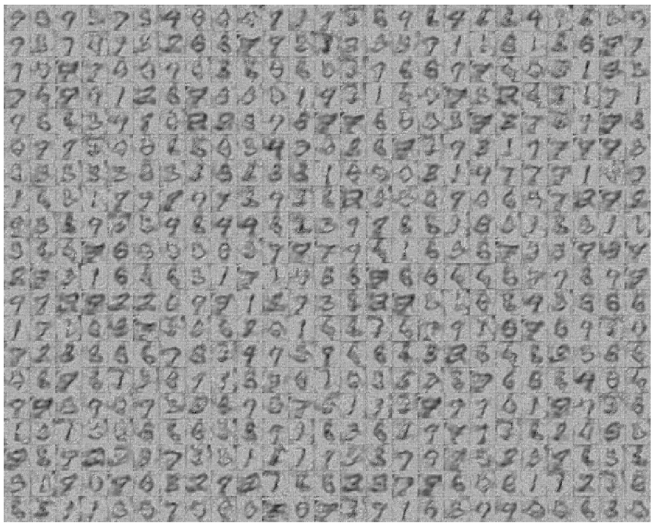


Figure A.12: MNIST VampPrior learned pseudo-inputs.

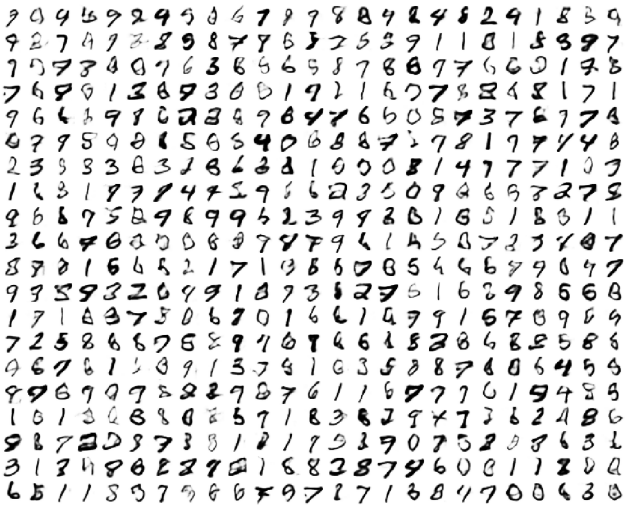


Figure A.13: MNIST VampPrior reconstructed pseudo-inputs obtained by deterministically encoding and decoding each pseudo-input.

Figure A.14: Omniglot learned inducing points.

Figure A.15: CelebA learned inducing points.

B Empirical results

Labels per class	1	10	20	30	40	50	60	70	80	90	100
No prior	0.506 ± 0.095	0.781 ± 0.045	0.820 ± 0.023	0.829 ± 0.020	0.836 ± 0.026	0.839 ± 0.021	0.844 ± 0.017	0.846 ± 0.017	0.847 ± 0.015	0.843 ± 0.015	0.848 ± 0.014
Normal	0.396 ± 0.076	0.775 ± 0.051	0.838 ± 0.020	0.861 ± 0.016	0.874 ± 0.011	0.883 ± 0.011	0.886 ± 0.011	0.892 ± 0.010	0.896 ± 0.010	0.899 ± 0.011	0.901 ± 0.008
Vamp(500)	0.539 ± 0.094	0.849 ± 0.035	0.891 ± 0.019	0.905 ± 0.013	0.911 ± 0.016	0.918 ± 0.012	0.921 ± 0.009	0.925 ± 0.008	0.929 ± 0.007	0.928 ± 0.005	0.932 ± 0.005
DVAE#	0.453 ± 0.101	0.735 ± 0.027	0.784 ± 0.017	0.801 ± 0.012	0.813 ± 0.013	0.824 ± 0.014	0.830 ± 0.012	0.835 ± 0.011	0.841 ± 0.007	0.842 ± 0.007	0.846 ± 0.008
MAF	0.530 ± 0.113	0.869 ± 0.029	0.910 ± 0.012	0.923 ± 0.012	0.930 ± 0.007	0.933 ± 0.010	0.938 ± 0.008	0.940 ± 0.008	0.942 ± 0.006	0.944 ± 0.006	$\mathbf{0.946 \pm 0.005}$
LORACs(200)	$\mathbf{0.670 \pm 0.120}$	$\mathbf{0.903 \pm 0.019}$	$\mathbf{0.923 \pm 0.011}$	$\mathbf{0.929 \pm 0.009}$	$\mathbf{0.934 \pm 0.006}$	$\mathbf{0.938 \pm 0.004}$	$\mathbf{0.939 \pm 0.005}$	$\mathbf{0.941 \pm 0.004}$	$\mathbf{0.943 \pm 0.004}$	$\mathbf{0.944 \pm 0.003}$	0.945 ± 0.003

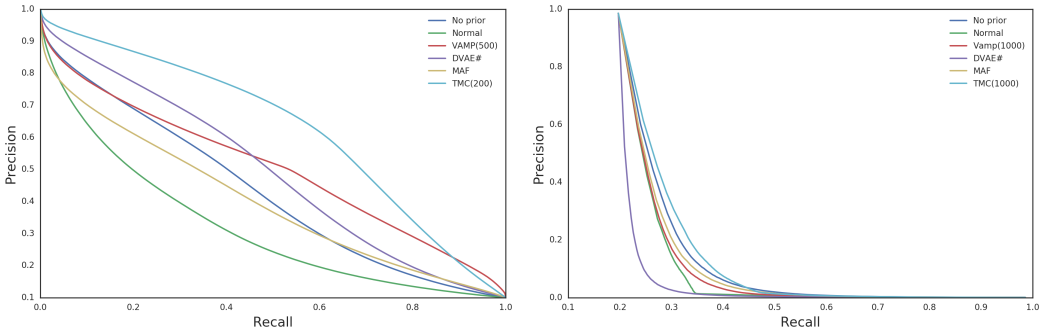
Table B.3: MNIST few-shot classification results.

Labels per class	1	2	5	10	15
No prior	0.140 ± 0.012	0.179 ± 0.008	0.225 ± 0.006	0.252 ± 0.009	0.290 ± 0.001
Normal	0.107 ± 0.007	0.134 ± 0.010	0.187 ± 0.008	0.246 ± 0.006	0.285 ± 0.000
Vamp(1000)	0.116 ± 0.011	0.148 ± 0.009	0.210 ± 0.003	0.270 ± 0.005	0.300 ± 0.000
DVAE#	0.042 ± 0.004	0.060 ± 0.006	0.091 ± 0.003	0.121 ± 0.001	0.141 ± 0.000
MAF	0.096 ± 0.008	0.129 ± 0.006	0.177 ± 0.010	0.222 ± 0.007	0.237 ± 0.002
LORACs(1000)	$\mathbf{0.173 \pm 0.005}$	$\mathbf{0.236 \pm 0.005}$	$\mathbf{0.330 \pm 0.008}$	$\mathbf{0.403 \pm 0.006}$	$\mathbf{0.441 \pm 0.000}$

Table B.4: Omniglot few-shot classification results.

# of inducing points	200	500
	0.9428	0.9474

Table B.5: MNIST few-shot classification with labeled inducing points.



(a) MNIST (b) Omniglot

Figure B.16: Averaged precision-recall curves over test datasets.

C Algorithm details

C.1 Stick breaking process

Consider inserting a node $N+1$ into the tree in between vertices u and v such that $t_v > t_u$, creating branch e_{N+1}. The inserted node has time t_{N+1} with probability according to the stick breaking process, i.e.

$$r(t_{N+1} \mid e_{N+1}, V, E) = \text{Beta}\left(\frac{t_v - t_{N+1}}{1 - t_{N+1}}; a, b\right) \text{Beta}\left(\frac{t_{N+1} - t_u}{1 - t_u}; a, b\right). \tag{C.16}$$

C.2 Belief propagation in TMCs

The TMC is at the core of the LORACs prior. Recall that the TMC is a prior over phylogenies τ, and after attaching a Gaussian random walk (GRW), we obtain a distribution over N vectors in $\mathbb{R}^d$, corresponding to the leaves, $r(z_{1:N} \mid \tau)$. However, the GRW samples latent vectors at internal nodes $z_{V_{\text{int}}}$. Rather than explicitly representing these values, in this work we marginalize them out, i.e.

$$r(z_{1:N} \mid \tau) = \int r(z_{1:N} \mid z_{V_{\text{int}}}, \tau) p(z_{V_{\text{int}}} \mid \tau) dz_{V_{\text{int}}} \tag{C.17}$$

This marginalization process can be done efficiently, because our graphical model is tree-shaped and all nodes have Gaussian likelihoods. Belief propagation is a message-passing framework for marginalization and we utilize message-passing for several TMC inference queries. The main queries we are interested in are:

1. $r(z_{1:N}, \tau)$ - for the purposes of MCMC, we are interested in computing the joint likelihood of a set of observed leaf values and a phylogeny.
2. $r(z_n \mid z_{\setminus n}, \tau)$ - this query computes the posterior density over one leaf given all the others; we use this distribution when computing the posterior predictive density of a TMC.
3. $\nabla_{z_{\setminus n}} r(z_n \mid z_{\setminus n}, \tau)$ - this query is the gradient of the predictive density of a single leaf with respect to the values at all other leaves. This query is used when computing gradients of the ELBO w.r.t $s_{1:M}$ in the LORACs prior.

Message passing Message passing treats the tree as an undirected graph. We first pick start node v_{start} and request messages from each of v_{start}'s neighbors.

Message passing is thereafter defined recursively. When a node v has requested messages from a source node s, it thereafter requests messages from all its neighbors but s. The base case for this recursion is a leaf node v_n, which returns a message with the following contents:

$$\nu_n = \mathbf{0}; \quad \mu_n = z_n; \quad \log Z_n = 0; \quad \nabla_{\nu_n}(\nu) = \mathbf{1}; \quad \nabla_{\nu_n}(\mu) = \mathbf{0}; \quad \nabla_{\mu_n}(\mu) = \mathbf{1} \tag{C.18}$$

where bold numbers $\mathbf{0} \triangleq (0, \ldots, 0)^\top$ and $\mathbf{1} \triangleq (1, \ldots, 1)^\top$ denote vectors obtained by repeating a scalar d times.

In the recursive case, consider being at a node i and receiving a set of messages from its neighbors M.

$$\nu_i = \frac{1}{\sum_{m \in M} \frac{1}{\nu_m + e_{im}}}; \quad \mu_i = v_i \sum_{m \in M} \frac{\mu_m}{\nu_m + e_{im}} \tag{C.19}$$

where e_{im} is the length of the edge between nodes i and m. These messages are identical to those used in Boyles and Welling (2012).

Additionally, our messages include gradients w.r.t. *every* leaf node downstream of the message. We update each of these gradients when computing the new message and pass them along to the source node. Gradients with respect to one of these nodes j are calculated as

$$\begin{aligned} \nabla_{\nu_j}(\nu) &= \nabla_{\nu_j} \nu_i \\ \nabla_{\nu_j}(\mu) &= \nabla_{\nu_j} \mu_i \\ \nabla_{\mu_j}(\mu) &= \nabla_{\mu_j} \mu_i \end{aligned} \tag{C.20}$$

The most complicated message is the $\log Z_i$ message, which depends on the number of incoming messages. v_{start} gets three incoming messages, all other nodes get only two. Consider two messages from nodes v_k and v_l:

$$\begin{aligned} \Sigma_i &\triangleq (\nu_k + e_{ik} + \nu_l + e_{il}) I \\ \log Z_i &= -\frac{1}{2} \|\mu_k - \mu_l\|^2_{\Sigma_i} - \frac{1}{2} \left(\log |\Sigma_i|_d \log 2\pi \right) \end{aligned} \tag{C.21}$$

For three messages from nodes v_k, v_l, and v_m:

$$\begin{aligned} \Sigma_i &\triangleq \left((\nu_k + e_{ik})(\nu_l + e_{il}) + (\nu_l + e_{il})(\nu_m + e_{im}) + (\nu_m + e_{im})(\nu_k + e_{ik}) \right) I \\ \log Z_i &= -\frac{1}{2} \left((\nu_m + e_{im}) \|\mu_k - \mu_l\|^2_{\Sigma_i} + (\nu_k + e_{ik}) \|\mu_l - \mu_m\|^2_{\Sigma_i} + (\nu_l + e_{il}) \|\mu_m - \mu_k\|^2_{\Sigma_i} \right) - \frac{1}{2} \log |\Sigma_i| - \log 2\pi \end{aligned} \tag{C.22}$$

With these messages, we can answer all the aforementioned inference queries.

1. We can begin message passing at any internal node and compute: $\log r(z_{1:N}, \tau) = \sum_{v \in V} \log Z_v$
2. We start message passing at v_n. $r(z_n \mid z_{\setminus n}, \tau)$ is a Gaussian with mean μ_n and variance ν_n.
3. $\nabla_{z_{\setminus n}} r(z_n \mid z_{\setminus n}, \tau)$ is $\nabla_{z_{\setminus n}} \mathcal{N}(z_n \mid \mu_n, \nu_n I)$, which in turn utilizes gradients sent via message passing.

Implementation We chose to implement the TMC and message passing in Cython because we found raw Python to be too slow due to function call and type-checking overhead. Furthermore, we used diagonal rather than scalar variances in the message passing implementation to later support diagonal variances handed from the variational posterior over z_n.

C.3 Variational inference for the LORACs prior

The LORACs prior involves first sampling a tree from the posterior distribution over TMCs with $s_{1:M}$ as leaves. We then sample a branch and time for each data z_n according to the posterior predictive distribution described in subsection 2.1. We then sample a z_n from the distribution induced by the GRW likelihood model. Finally, we pass the sampled z_n through the decoder.

$$\begin{aligned} \tau &\sim p(\tau; s_{1:M}) \\ e_n, t_n &\sim p(e_n, t_n | \tau) \\ z_n | e_n, t_n, \tau &\sim p(z_n | e_n, t_n, \tau; s_{1:M}) \triangleq r(s_{M+1} = z_n | e_n, t_n, \tau) \\ x_n | z_n &\sim p_\theta(x_n | z_n) \end{aligned} \tag{C.23}$$

Consider sampling the optimal $q^*(\tau; s_{1:M})$.

$$\begin{aligned} q^*(\tau; s_{1:M}) &\propto \exp\{\mathbb{E}_q\left[\log p(\tau, z_{1:N}, x_{1:N})\right]\} \\ &\propto \exp\{\log p(\tau; s_{1:M}) + \sum_n \mathbb{E}_q\left[p(z_n|e_n, t_n, \tau)\right]\} \\ &\propto \exp\{\log \mathrm{TMC}_N(\tau; a, b) + \sum_{m=1}^{M} \log r(s_m|s_{1:m-1}, \tau) \\ &+ \sum_n \mathbb{E}_q\left[\log p(z_n|e_n, t_n, \tau)\right]\} \end{aligned} \tag{C.24}$$

We set $q(\tau; s_{1:M}) = r(\tau \mid s_{1:M})$. We use additional variational factors $q(e_n)$, $q_\xi(t_n|e_n, z_n; s_{1:M})$, and $q_\phi(z_n|x_n)$. $q_\xi(t_n|e_n, z_n; s_{1:M})$ is a recognition network that outputs the attach time for a particular branch. Since the $q(\tau; s_{1:M})$ and $p(\tau; s_{1:M})$ terms cancel out, we obtain the following ELBO.

$$\mathcal{L}[q] \triangleq \mathbb{E}_q\left[\log \frac{\prod_n p(e_n, t_n|\tau)p(z_n|e_n, t_n, \tau; s_{1:M})p_\theta(x_n|z_n)}{\prod_n q(e_n)q_\xi(t_n|e_n, z_n; s_{1:M})q_\phi(z_n|x_n)}\right] \tag{C.25}$$

Inference procedure In general, $q(\tau; s_{1:M})$ can be sampled using vanilla SPR Metropolis-Hastings, so samples from this distribution are readily available.

For each data in the minibatch x_n, we pass it through the encoder to obtain $q(z_n|x_n)$. We then compute

$$q^*(e_n) = \exp\left\{\mathbb{E}_q\left[\log p(e_n|t_n, z_n, \tau; s_{1:M})\right]\right\} \tag{C.26}$$

This quantity is computed by looping over every branch b of a sample from $q(\tau)$, storing incoming messages at each node, passing the μ and ν and a sample from $q(z_n|x_n)$ into $q_\xi(t_n|e_n, s_{1:M}, z_n)$, outputting a logistic-normal distribution over times for that branch. We sample that logistic normal to obtain a time t to go with branch b. We can then compute the log-likelihood of z_n if it were to attach to b and t, using TMC inference query #2. This log-likelihood is added to the TMC prior log-probability of the branch being selected to obtain a joint probability $\mathbb{E}_q\left[\log p(e_n)p(t_n)p(z_n|e_n, t_n, \tau; s_{1:M})\right]$ over the branch. After doing this for every branch, we normalize the joint likelihoods to obtain the optimal categorical distribution over every branch for z_n, $q^*(e_n)$. We then sample this distribution to obtain an attach location and time e_n, t_n for each data in the minibatch.

The next stage is to compute gradients w.r.t. to the learnable parameters of the model (θ, $s_{1:M}$, ϕ, and ξ). In the process of calculating $q^*(e_n)$, we have obtained samples from its corresponding $q_\xi(t_n|e_n, z_n, \tau; s_{1:M})$ and $q(z_n|x_n)$. We plug these into the ELBO and can compute gradients via automatic differentiation w.r.t. ϕ, θ, and ξ. Computing gradients w.r.t. $s_{1:M}$ is more tricky. We first examine the ELBO.

$$\mathcal{L}[q] = \mathbb{E}_q\left[\log \frac{\prod_n p(e_n|\tau)p(t_n)p(z_n|e_n, t_n, \tau; s_{1:M})p_\theta(x_n|z_n)}{\prod_n q(e_n)q_\xi(t_n|e_n, z_n; s_{1:M})q_\phi(z_n|x_n)}\right] \tag{C.27}$$

Consider the gradient of the ELBO with respect to $s_{1:M}$.

$$
\begin{aligned}
\nabla_{s_{1:M}}\mathcal{L}[q] &= \nabla_{s_{1:M}}\mathbb{E}_q\left[\log\frac{\prod_n p(e_n|\tau)p(t_n)p(z_n|e_n,t_n,\tau;s_{1:M})p_\theta(x_n|z_n)}{\prod_n q(e_n)q_\xi(t_n|e_n,z_n;s_{1:M})q_\phi(z_n|x_n)}\right] \\
&= \nabla_{s_{1:M}}\sum_\tau q(\tau;s_{1:M})\mathbb{E}_q\left[\log\frac{\prod_n p(e_n|\tau)p(t_n)p(z_n|e_n,t_n,\tau;s_{1:M})p_\theta(x_n|z_n)}{\prod_n q(e_n)q_\xi(t_n|e_n,z_n;s_{1:M})q_\phi(z_n|x_n)}\right] \\
&= \sum_\tau q(\tau;s_{1:M})\nabla_{s_{1:M}}\mathbb{E}_q\left[\log\frac{\prod_n p(e_n|\tau)p(t_n)p(z_n|e_n,t_n,\tau;s_{1:M})p_\theta(x_n|z_n)}{\prod_n q(e_n)q_\xi(t_n|e_n,z_n;s_{1:M})q_\phi(z_n|x_n)}\right] \\
&+ \sum_\tau \left(\nabla_{s_{1:M}}q(\tau;s_{1:M})\right)\mathbb{E}_q\left[\log\frac{\prod_n p(e_n|\tau)p(t_n)p(z_n|e_n,t_n,\tau;s_{1:M})p_\theta(x_n|z_n)}{\prod_n q(e_n)q_\xi(t_n|e_n,z_n;s_{1:M})q_\phi(z_n|x_n)}\right] \\
&= \sum_\tau q(\tau;s_{1:M})\nabla_{s_{1:M}}\mathbb{E}_q\left[\log\frac{\prod_n p(e_n|\tau)p(t_n)p(z_n|e_n,t_n,\tau;s_{1:M})p_\theta(x_n|z_n)}{\prod_n q(e_n)q_\xi(t_n|e_n,z_n;s_{1:M})q_\phi(z_n|x_n)}\right] \\
&+ \sum_\tau \left(q(\tau;s_{1:M})\nabla_{s_{1:M}}\log q(\tau;s_{1:M})\right)\mathbb{E}_q\left[\log\frac{\prod_n p(e_n|\tau)p(t_n)p(z_n|e_n,t_n,\tau;s_{1:M})p_\theta(x_n|z_n)}{\prod_n q(e_n)q_\xi(t_n|e_n,z_n;s_{1:M})q_\phi(z_n|x_n)}\right] \\
&= \mathbb{E}_{q(\tau)}\left[\nabla_{s_{1:M}}\mathbb{E}_q\left[\log\frac{\prod_n p(e_n|\tau)p(t_n)p(z_n|e_n,t_n,\tau;s_{1:M})p_\theta(x_n|z_n)}{\prod_n q(e_n)q_\xi(t_n|e_n,z_n;s_{1:M})q_\phi(z_n|x_n)}\right]\right. \\
&\left.+\ \nabla_{s_{1:M}}\log q(\tau;s_{1:M})\mathbb{E}_q\left[\log\frac{\prod_n p(e_n|\tau)p(t_n)p(z_n|e_n,t_n,\tau;s_{1:M})p_\theta(x_n|z_n)}{\prod_n q(e_n)q_\xi(t_n|e_n,z_n;s_{1:M})q_\phi(z_n|x_n)}\right]\right] \\
&= \mathbb{E}_q\left[\nabla_{s_{1:M}}\left(-\log q(e_n)-\log q(t_n\,|\,z_n,e_n,\tau;s_{1:M})+\log p(z_n\,|\,e_n,t_n,\tau;s_{1:M})\right)\right] \\
&+ \mathbb{E}_q\left[\nabla_{s_{1:M}}\left(\log q(\tau)+\log q(e_n)\right)\log\frac{p(e_n|\tau)}{q(e_n)}\frac{p(z_n\,|\,z_n,e_n,t_n,\tau;s_{1:M})}{q(t_n\,|\,z_n,e_n,\tau;s_{1:M})}\right]
\end{aligned}
\tag{C.28}
$$

In the last step, we expand out expectation over e_n and then pass the derivative through like we did for τ. The gradients w.r.t. $q(e_n)$ are zero, since $q^*(e_n)$ is a partial optimum of the ELBO and we are left with:.

$$
\begin{aligned}
\nabla_{s_{1:M}}\mathcal{L}[q] &= \mathbb{E}_q\left[\nabla_{s_{1:M}}\log p(z_N|e_n,t_n,\tau;s_{1:M})\right]-\mathbb{E}_q\left[\log q(t_n\,|\,z_n,e_n,\tau;s_{1:M})\right] \\
&+\mathbb{E}_q\left[\nabla_{s_{1:M}}\log q(\tau;s_{1:M})\log\frac{p(e_n|\tau)}{q(e_n)}\frac{p(z_n\,|\,z_n,e_n,t_n,\tau;s_{1:M})}{q(t_n\,|\,z_n,e_n,\tau;s_{1:M})}\right]
\end{aligned}
\tag{C.29}
$$

The first term of the gradient is the expected gradient of the posterior predictive density w.r.t $s_{1:M}$. This can be calculated by using TMC inference query #3 using samples from $q(e_n)$ and $q(t_n\,|\,z_n,e_n,\tau;s_{1:M})$. The second term also uses the same gradients, by means of the chain rule to differentiate through the time-amortization network. The third term of this gradient is a score function gradient, which we decide to not use due to the high-variance nature of score function gradients. We found that we were able to obtain strong results even with biased gradients.

D Details of experiments

We implemented the LORACs prior in Tensorflow and Cython. For MNIST and Omniglot, our architectures are in Table D.6 and CelebA is in Table D.7.

Layer type	Shape
Conv + ReLU	[3, 3, 64], stride 2
Conv + ReLU	[3, 3, 32], stride 1
Conv + ReLU	[3, 3, 16], stride 2
FC + ReLU	512
Gaussian	40

(a) Encoder

Layer type	Shape
FC + ReLU	3136
Deconv + ReLU	[3, 3, 32], stride 2
Deconv + ReLU	[3, 3, 32], stride 1
Deconv + ReLU	[3, 3, 1], stride 2
Bernoulli	

(b) Decoder

Table D.6: Network architectures for MNIST and Omniglot

Layer type	Shape
Conv + ReLU	[3, 3, 64], stride 2
Conv + ReLU	[3, 3, 32], stride 1
Conv + ReLU	[3, 3, 16], stride 2
FC + ReLU	512
Gaussian	40

(a) Encoder

Layer type	Shape
FC + ReLU	4096
Deconv + ReLU	[3, 3, 32], stride 2
Deconv + ReLU	[3, 3, 32], stride 1
Deconv + ReLU	[3, 3, 3], stride 2
Bernoulli	

(b) Decoder

Table D.7: Network architectures for CelebA

In general, we trained the model interleaving one gradient step with 100 sampling steps for $q(\tau; s_{1:M})$. We also found that experimenting with values of a and b in the TMC prior did not impact results significantly. We initialized the networks with weights from a VAE trained for 100 epochs and inducing points were initialized using k-means. All parameters were trained using Adam (Kingma and Ba, 2015) with a 10^{-3} learning rate for an 100 epochs with learning rate decay to 10^{-5} for the last 20 epochs. Finally, we initialized trees with all node times close to 0, to emulate a VAE prior.

D.1 Baseline details

All baselines were trained with the default architecture. They were trained for 400 epochs, with KL warmup (β started at 10^{-2}, and ramped up to $\beta = 1$ linearly over 50 epochs). They were trained using Adam with a learning rate of 10^{-3}, with a learning rate of 10^{-5} for the last 80 epochs.

For VampPrior, we sued 500 pseudo-inputs for MNIST and 1000 for Omniglot. For MAF, we used a two layer, 512 wide MADE. DVAE# was trained using the default implementation from `https://github.com/QuadrantAI/dvae`, which is hierarchical VAE consisting of two Bernoulli latent variables, 200-dimensional each. Each is learned via a feed-forward neural network 4-layers deep. The default DVAE# implementation also uses statically binarized MNIST where we use dynamically binarized.