A. Appendix

A.1. Derivation of the gradient of the ELBO

The evidence lower bound objective (ELBO) of the model is defined as:

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{q_{\theta,\phi}(\mathbf{z},\mathbf{s}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z},\mathbf{s}) - \log q_{\theta,\phi}(\mathbf{z},\mathbf{s}|\mathbf{x})\right] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})p_\theta(\mathbf{s}|\mathbf{x},\mathbf{z})}[\log p_\theta(\mathbf{x},\mathbf{z})p_\theta(\mathbf{s}|\mathbf{x},\mathbf{z}) \\ &\qquad - \log q_\phi(\mathbf{z}|\mathbf{x})p_\theta(\mathbf{s}|\mathbf{x},\mathbf{z})] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z})\right] + H(q_\phi(\mathbf{z}|\mathbf{x})) \qquad (12)\end{aligned}$$

where the first term is the model likelihood, and the second is the conditional entropy for variational posterior of continuous hidden states. We can approximate the entropy of $q_\phi(\mathbf{z}|\mathbf{x})$ as:

$$H(q_\phi(\mathbf{z}|\mathbf{x})) = H(q_\phi(\mathbf{z}_1)) + \sum_{t=2}^{T} H(q_\phi(\mathbf{z}_t|\tilde{\mathbf{z}}_{1:t-1}))$$

where $\tilde{\mathbf{z}}_t \sim q(\mathbf{z}_t)$ is a sample from the variational posterior. In other words, we compute the marginal entropy for the output of the RNN inference network at each time step, and then sample a single latent vector to update the RNN state for the next step.

In order to apply stochastic gradient descent for end-to-end training, the minibatch gradient for the first term in the ELBO (Eq. 12) with respect to $\boldsymbol{\theta}$ is estimated as

$$\nabla_\theta \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z})\right] = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\nabla_\theta \log p_\theta(\mathbf{x},\mathbf{z})\right]$$

For the gradient with respect to $\boldsymbol{\phi}$, we can use the reparameterization trick to write

$$\begin{aligned}&\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z})\right] \\ &= \mathbb{E}_{\boldsymbol{\epsilon}\sim N}\left[\nabla_\phi \log p_\theta(\mathbf{x},\mathbf{z}_\phi(\boldsymbol{\epsilon},\mathbf{x}))\right]\end{aligned}$$

Therefore, the gradient is expressed as:

$$\begin{aligned}&\nabla_\theta \mathcal{L} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\nabla_\theta \log p_\theta(\mathbf{x},\mathbf{z})\right], \\ &\nabla_\phi \mathcal{L} = \mathbb{E}_{\boldsymbol{\epsilon}\sim N}\left[\nabla_\phi \log p_\theta(\mathbf{x},\mathbf{z}_\phi(\boldsymbol{\epsilon},\mathbf{x}))\right] + \nabla_\phi H(q_\phi(\mathbf{z}|\mathbf{x})).\end{aligned}$$

To compute the derivative of the log-joint likelihood $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \log p_\theta(\mathbf{v})$, where we define $\mathbf{v} = (\mathbf{x}_{1:T}, \mathbf{z}_{1:T})$ as the visible variables for brevity. Therefore

$$\begin{aligned}\nabla \log p(\mathbf{v}) =& \mathbb{E}_{p(\mathbf{s}|\mathbf{v})}\left[\nabla \log p(\mathbf{v})\right] \\ =& \mathbb{E}_{p(\mathbf{s}|\mathbf{v})}\left[\nabla \log p(\mathbf{v},\mathbf{s})\right] \\ &- \mathbb{E}_{p(\mathbf{s}|\mathbf{v})}\left[\nabla \log p(\mathbf{s}|\mathbf{v})\right] \\ =& \mathbb{E}_{p(\mathbf{s}|\mathbf{v})}\left[\nabla \log p(\mathbf{v},\mathbf{s})\right] - 0\end{aligned}$$

where we used the fact that $\log p(\mathbf{v}) = \log p(\mathbf{v},\mathbf{s}) - \log p(\mathbf{s}|\mathbf{v})$ and

$$\begin{aligned}\mathbb{E}_{p(\mathbf{s}|\mathbf{v})}\left[\nabla \log p(\mathbf{s}|\mathbf{v})\right] &= \int p(\mathbf{s}|\mathbf{v}) \frac{\nabla p(\mathbf{s}|\mathbf{v})}{p(\mathbf{s}|\mathbf{v})} \\ &= \nabla \int p(\mathbf{s}|\mathbf{v}) = \nabla 1 = 0.\end{aligned}$$

For $\nabla \log p(\mathbf{v},\mathbf{s})$, we use the Markov property to rewrite it as:

$$\begin{aligned}&\nabla \log p(\mathbf{v},\mathbf{s}) = \\ &\sum_{t=2}^{T} \nabla \log p(\mathbf{x}_t|\mathbf{z}_t) p(\mathbf{z}_t|\mathbf{z}_{t-1}, s_t) p(s_t|s_{t-1}, \mathbf{x}_{t-1}) \\ &\qquad + \nabla \log p(\mathbf{x}_1|\mathbf{z}_1) p(\mathbf{z}_1|s_1) p(s_1),\end{aligned}$$

with the expectation being:

$$\begin{aligned}&\nabla \log p(\mathbf{v}) = \mathbb{E}_{p(\mathbf{s}|\mathbf{v})}\left[\nabla \log p(\mathbf{v},\mathbf{s})\right] \\ &= \sum_k p(s_1 = k|\mathbf{v}) \nabla \log p(\mathbf{x}_1|z_1) p(\mathbf{z}_1|s_1 = k) p(s_1 = k) \\ &+ \sum_{t=2}^{T} \sum_{j,k} p(s_{t-1} = j, s_t = k|\mathbf{v}) \nabla \left[\log p(\mathbf{x}_t|\mathbf{z}_t) \cdot \right. \\ &\qquad \left. p(\mathbf{z}_t|\mathbf{z}_{t-1}, s_t = k) p(s_t = k|s_{t-1} = j, \mathbf{x}_{t-1})\right] \\ &= \sum_{t=2}^{T} \sum_{j,k} \gamma_t^2(j,k) \nabla \log B_t(k) A_t(j,k) \\ &\qquad + \sum_k \gamma_1^1(k) \nabla \log B_1(k) \pi(k).\end{aligned}$$

Therefore we reach the Eq. 9.

An alternative derivation is by Jensen's inequality,

$$\begin{aligned}\nabla \log p(\mathbf{v}) &= \nabla \left(\log \mathbb{E}_{p(\mathbf{s}|\mathbf{v})}\left[p(\mathbf{v},\mathbf{s})\right]\right) \\ &\geq \nabla \left(\mathbb{E}_{p(\mathbf{s}|\mathbf{v})}\left[\log p(\mathbf{v},\mathbf{s})\right]\right). \qquad (13)\end{aligned}$$

In order to estimate the Eq. 9, we could maximize the lower bound, Eq. 13, by applying the auto-differentiation over

$$\begin{aligned}&\mathbb{E}_{p(\mathbf{s}|\mathbf{v})}\left[\log p(\mathbf{v},\mathbf{s})\right] = \\ &\qquad \sum_{t=2}^{T} \sum_{j,k} \gamma_t^2(j,k) \left[\log B_t(k) A_t(j,k)\right] \\ &\qquad + \sum_k \gamma_1^1(k) \left[\log B_1(k) \pi(k)\right]\end{aligned}$$

In summary, one step of stochastic gradient ascent for the ELBO can be implemented as Algorithm 1.

Algorithm 1 SVI for Training SNLDS

1: Compute $\mathbf{h}_t^x$ from $\mathbf{x}_{1:T}$ using a Bi-RNN;
2: Recursively sample $\mathbf{z}_t \sim q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{1:T})$ using RNN over $\mathbf{z}_{t-1}$ and $\mathbf{h}_t^x$;
3: Run forward-backward messages to compute $\mathbf{A}$, $\mathbf{B}$, $\boldsymbol{\pi}$, $\boldsymbol{\gamma}_{1:T}^1$, $\boldsymbol{\gamma}_{1:T-1}^2$ from $(\mathbf{x},\mathbf{z})$;
4: Compute $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \log p(\mathbf{x},\mathbf{z})$ from Eqn. 9;
5: Take gradient step.

A.2. Gumbel-Softmax SNLDS

Instead of marginalizing out the discrete states with the forward-backward algorithm, one could use a continuous relaxation via reparameterization, e.g. the Gumbel-Softmax trick (Jang et al., 2017), to infer the most likely discrete states. We call this `Gumbel-Softmax SNLDS`.

We consider the same state space model as SNLDS:

$$p_\theta(\mathbf{x}, \mathbf{z}, \mathbf{s}) = p(\mathbf{x}_1|\mathbf{z}_1)p(\mathbf{z}_1|s_1) \left[\prod_{t=2}^{T} p(\mathbf{x}_t|\mathbf{z}_t)p(\mathbf{z}_t|\mathbf{z}_{t-1}, s_t)p(s_t|s_{t-1}, \mathbf{x}_{t-1})\right],$$

where $s_t \in \{1, \dots, K\}$ is the discrete hidden state, $\mathbf{z}_t \in \mathbb{R}^L$ is the continuous hidden state, and $\mathbf{x}_t \in \mathbb{R}^D$ is the observed output, as in Figure 2(a). The inference network for the variational posterior now predicts both $\mathbf{s}$ and $\mathbf{z}$ and is defined as

$$q_{\boldsymbol{\phi}_\mathbf{z}, \boldsymbol{\phi}_\mathbf{s}}(\mathbf{z}, \mathbf{s}|\mathbf{x}) = q_{\boldsymbol{\phi}_\mathbf{z}}(\mathbf{z}|\mathbf{x}) q_{\boldsymbol{\phi}_\mathbf{s}}(\mathbf{s}|\mathbf{x}) \tag{14}$$

where

$$\begin{aligned} q_{\boldsymbol{\phi}_\mathbf{z}}(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}) &= \prod_t q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{1:T}) \\ &= \prod_t q(\mathbf{z}_t|\mathbf{h}_t)\delta(\mathbf{h}_t|f_{RNN}(\mathbf{h}_{t-1}, \mathbf{z}_{t-1}, \mathbf{h}_t^b)) \\ q_{\boldsymbol{\phi}_\mathbf{s}}(\mathbf{s}_{1:T}|\mathbf{x}_{1:T}) &= \prod_t q(s_t|s_{t-1}, \mathbf{x}_{1:T}) \\ &= \prod_t q_{\text{Gumbel}-\text{Softmax}}(s_t|g(\mathbf{h}_t^b, s_{t-1}), \tau) \end{aligned}$$

where $\mathbf{h}_t$ is the hidden state of a deterministic recurrent neural network, $f_{RNN}(\cdot)$, which works from left ($t = 0$) to right ($t = T$), summarizing past stochastic $\mathbf{z}_{1:t-1}$. We also feed in $\mathbf{h}_t^b$, which is a bidirectional RNN, which summarizes $\mathbf{x}_{1:T}$. The Gumbel-Softmax distribution $q_{\text{Gumbel}-\text{Softmax}}$ takes the output of a feed-forward network $g(\cdot)$ and a softmax temperature τ, which is annealed according to a fixed schedule.

The evidence lower bound (ELBO) could be written as

$$\begin{aligned} \mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}_\mathbf{z}}(\mathbf{z}|\mathbf{x})q_{\boldsymbol{\phi}_\mathbf{s}}(\mathbf{s}|\mathbf{x})} \big[&\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}, \mathbf{s}) \\ &- \log q_{\boldsymbol{\phi}_\mathbf{z}}(\mathbf{z}|\mathbf{x})q_{\boldsymbol{\phi}_\mathbf{s}}(\mathbf{s}|\mathbf{x})\big] \end{aligned} \tag{15}$$

One step of stochastic gradient ascent for the ELBO can be implemented as Algorithm 2.

A.3. Details on the bouncing ball experiment

The input data for bouncing ball experiment is a set of 100000 sample trajectories, each of which is of 100 timesteps with its initial position randomly placed between two walls separated by a distance of 10. The velocity of the ball for each sample trajectory is sampled from $\mathcal{U}([-0.5, 0.5])$. The exact position of ball is obscured with Gaussian noise $\mathcal{N}(0, 0.1)$. The training is performed with batch size 32. The evaluation is carried on a fixed, held-out subset of the data with 200 samples. For the inference network, the bi-directional and forward RNNs are both 16 dimensional GRU. The dimensions of discrete and continuous hidden state are set to be 3 and 4. For SLDS, we use linear transition for continuous states. For SNLDS, we use GRU with 4 hidden units followed by linear transformation for continuous state transition. The model is trained with fixed learning rate of 10^{-3}, with the Adam optimizer (Kingma & Ba, 2015), and gradient clipping by norm of 5 for 10000 steps.

Algorithm 2 SVI for Training Gumbel-Softmax SNLDS

1: Use Bi-RNN to compute $\mathbf{h}_t^x$ from $\mathbf{x}_{1:T}$;
2: Recursively sample $\mathbf{z}_t \sim q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{1:T})$ using RNN over $\mathbf{z}_{t-1}$ and $\mathbf{h}_t^x$;
3: Recursively sample s_t with distribution $q_{\text{Gumbel}-\text{Softmax}}(s_t|g(\mathbf{h}_t^b, s_{t-1}), \tau)$, where g is a feedforward network;
4: Compute the likelihood for eq. (15);
5: Take gradient step.

A.4. Details on the reacher experiment

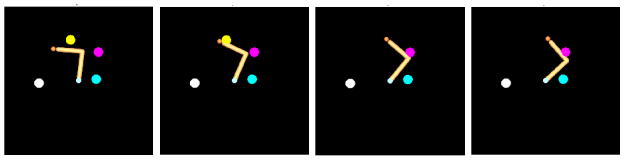

Figure 7. Illustration of the observations in reacher experiment. This is 2-D rendering of the observational vector, but the inputs to the model are sequences of vectors, as in Kipf et al. (2019), not images.

The observations in the reacher experiment are sequences of 36 dimensional vectors, as described in Kipf et al. (2019). First 30 elements are the target indicator, α, and location, x, y, for 10 randomly generated objects. 3 out of 10 objects start as targets, $\alpha = 1$. The (x, y) location for 5 of the non-target objects are set to $(0, 0)$. A deterministic controller moves the arm to the indicated target objects. Once a target is reached, the indicator is set to $\alpha = 0$. (Depicted as the yellow dot disappearing in Figure 7.) The remaining 6 elements of the observations are the two angles of reacher arm and the positions of two arm segment tips. The training dataset consists of 10000 observation samples, each 50 timesteps in length.

This more complex task requires more careful training. The learning rate schedule is a linear warm-up, 10^{-5} to 10^{-3} over 5000 steps, from followed by a cosine decay, with

decay rate of 2000 and minimum of 10^{-5}. Both entropy regularization coefficient starts to exponentially decay after 50000 steps, from initial value 1000 with a decay rate 0.975 and decay steps 500. The temperature annealing follows the same exponential but only starts to decay after 100000 steps. The training is performed in minibatches of size 32 for 300000 iterations using the Adam optimizer (Kingma & Ba, 2015).

The model architecture is relatively generic. The continuous hidden state z is 8 dimensional. The number of discrete hidden states is set to 5 for training, which is larger than the ground truth 4 (including states targeting 3 objects and a finished state). The observations pass through an encoding network with two 256-unit ReLU activated fully-connected nets, before feeding into RNN inference networks to estimate the posterior distributions $q(z_t|x_{1:T})$. The RNN inference networks consist of a 32-unit bidirection LSTM and a 64-unit forward LSTM. The emission network is a three-layer MLP with $[256, 256, 36]$ hidden units and ReLU activation for first two layers and a linear output layer. Discrete hidden state transition network takes two inputs: the previous discrete state and the processed observations. The observations are processed by the encoding network and a 1-D convolution with 2 kernels of size 3. The transition network outputs a 5×5 matrix for transition probability $p(s_t|s_{t-1})$ at each timestep. For SNLDS, we use a single-layer MLP as the continuous hidden state transition functions $p(z_t|z_{t-1}, s_t)$, with 64 hidden units and ReLU activation. For SLDS, we use linear transitions for the continuous state.

A.5. Details on the Dubins path experiment

The Dubins path model[4] is a simplified flight, or vehicle, trajectory that is the shortest path to reach a target position, given the initial position (x_0, y_0), the direction of motion θ_0, the speed constant V, and the maximum curvature constraint $\dot{\theta} \leq u$. The possible motion along the path is defined by

$$\dot{x}_t = V\cos(\theta_t),\ \dot{y}_t = V\sin(\theta_t),\ \dot{\theta}_t = u.$$

The path type can be described by three different modes/regimes: 'right turn (R)' , 'left turn (L)' or 'straight (S).'

To generate a sample trajectory used in training or testing, we randomly sample the velocity from a uniform distribution $V \sim \mathcal{U}([0.1, 0.5])$ (pixel/step), angular frequency from a uniform distribution $u/2\pi \sim \mathcal{U}([0.1, 0.15])$ (/step), and initial direction $\theta_0 \sim \mathcal{U}([0, 2\pi))$. The generated trajectories always start from the center of image $(0, 0)$. The duration of each regime is sampled from a Poisson distribution with mean 25 steps, with full sequence length 100 steps. The floating-point positional information is rendered onto a 28×28 image with Gaussian blurring with 0.3 standard deviation to minimize aliasing.

The same schedules as in the reacher experiment are used for the learning rate, temperature annealing and regularization coefficient decay.

The network architecture is similar to the reacher task except for the encoder and decoder networks. Each observation is encoded with a CoordConv (Liu et al., 2018b) network before passing into RNN inference networks, the archicture is defined in Table 3. The emission network $p(\mathbf{x}_t|\mathbf{z}_t)$ also uses a CoordConv network as described in Table 4. The continuous hidden state z in this experiment is 4 dimensional. The number of discrete hidden states s is set to be 5, which is larger than ground truth 3. The inference networks are a 32-unit bidirection LSTM and a 64-unit forward LSTM. The discrete hidden state transition network takes the output of observation encoding network in the same manner as the reacher task. For SNLDS, we use a two-layer MLP as continuous hidden state transition function $p(z_t|z_{t-1}, s_t)$, with $[32, 32]$ hidden units and ReLU activation. For SLDS, we use linear transition for continuous states.

See Figure 8 for an illustration of the reconstruction abilities (of the observed images) for the SLDS and SNLDS models. They are visually very similar; however, the SNLDS has a more interpretable latent state as described in Section 5.3.

Figure 8. Image sequence reconstruction for Dubins path. The sequence is averaged with early timepoints scaled to low intensity, late timepoints unchanged to indicate direction.

A.6. Regularization and Multi-steps Training

Training our SNLDS model with a powerful transition network but without regularization will fit the dynamics $p(\mathbf{z}_t|\mathbf{z}_{t-1}, s_t)$ with a single state. With randomly initialized networks, one state fits the dynamics better at the beginning and the forward-backward algorithm will cause more gradients to flow through that state than others. The best state is the only one that gets better.

To prevent this, we use regularization to cause the model to select each mode with uniform likelihood until the inference and emission network are well trained. Thus all discrete modes are able to learn the dynamics well initially. When

[4] https://en.wikipedia.org/wiki/Dubins_path

Table 3. CoordConv encoder Architecture. Before passing into the following network, the image is padded from $[28, 28, 1]$ to $[28, 28, 3]$ with the pixel coordinates.

Layer	Filters	Shape	Activation	Stride	Padding
1	2	[5, 5]	relu	1	same
2	4	[5, 5]	relu	2	same
3	4	[5, 5]	relu	1	same
4	8	[5, 5]	relu	2	same
5	8	[7, 7]	relu	1	valid
6	8	2 (Kernel Size)	None	1	causal

Table 4. CoordConv decoder Architecture. Before passing into the following network, the input z_t is tiled from $[8]$ to $[28, 28, 8]$, where 8 is the hidden dimension, and is then padded to $[28, 28, 10]$ with the pixel coordinates.

Layer	Filters	Shape	Activation	Stride	Padding
1	14	[1, 1]	relu	1	valid
2	14	[1, 1]	relu	1	valid
3	28	[1, 1]	relu	1	valid
4	28	[1, 1]	relu	1	valid
5	1	[1, 1]	relu	1	same

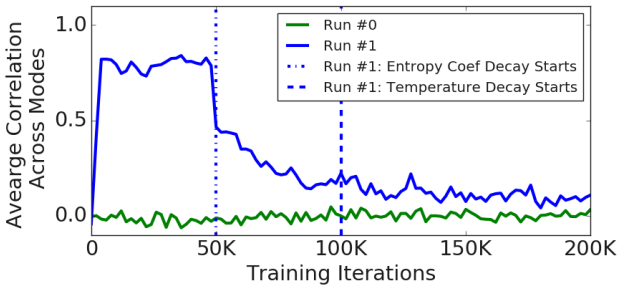

Figure 9. Comparing the average Pearson correlations among the weights from individual dynamical transition modes, $p(\mathbf{z}_t|\mathbf{z}_{t-1}, s_t = k)$, trained on Dubins Paths. Run 0 (green) is trained without regularization. Run 1 (blue) has its entropy coefficient starting to exponentially decay at step $50,000$, and the temperature starting to anneal at step $100,000$.

the regularization decays, the transition dynamics of each mode can then specialize. One effect of this regularization strategy is that the weights for each dynamics module are correlated early during training and decorrelate when the regularization decays. The regularization helps the model to better utilize its capacity, and the model can achieve better likelihood, as demonstrated in Section 5.5 and Figure 6.

Multi-steps training has been used by previous models, and it serves the same purpose as our regularization. SVAE first trains a single transition model, then uses that one set of parameters to initialize all the transition dynamics for multiple states in next stage of training. rSLDS training begins by fitting a single AR-HMM for initialization, then fits a standard SLDS, before finally fitting the rSLDS model. We follow these implementations of both SVAE and rSLDS in our paper. Both multi-step training and our regularization ensure the hidden dynamics are well learned before learning the segmentation. What makes our regularization approach interesting is that it allows the model to be trained with a smooth transition between early and late training.