Recurrent Switching Linear Dynamical Systems: Applications on Neuron Data Analysis

Chonghao Cai, Guanyu Huo, Luotong Kang, Jennifer Huang

December 14, 2022

Abstract

This paper discusses the Recurrent switching linear dynamics systems (rSLDS) with its applications to Lorenz dynamical system and brain neuron dynamical system from the data of the Allen Institute. We replicate the result of Lorenz dynamical system from Prof. Linderman's original paper [5]. Then, we implement this model on the data of two mouses' neurons in hippocampus and try to explore and evaluate the neural population activity within CA1 and possible hidden dynamics.

1 Introduction

1.1 Models

Recurrent switching linear dynamics systems (rSLDS) is a mathematical framework for studying the temporal evolution of complex systems. The rSLDS models are characterized by a set of linear differential equations that switch between different regimes over time, allowing for the modeling of systems with multiple modes of operation. The models are based on the switching linear-Gaussian dynamical systems (SLDS) model. Each prediction of the model would lead to a different dynamic, meanwhile, the model regulates the hidden Markov chains of switching states, thus allowing switching between multiple linear dynamical models. For data analysis, switching to new behavior patterns in different situations can all depend on the continuous state of the system or external factors. rSLDS model can decompose data into simple segments, with attribute segments transitioning to changes in the underlying state or environment. By discovering these dynamical units and their switching dependencies, we can gain insight into the complex data production process. It also provides interpretable representations of data dynamics. The rSLDS models have been applied to a wide range of problems, including control and optimization, time series analysis, and biological modeling.

1.2 Experiments and Dataset from Allen Institute

A number of researches [1], [7], [8] have suggested that neural activities can be modeled in a low-dimensional space. In this paper, we use a dataset of Neuropixels recordings from Allen Institute[4] to find the dynamical system of neurons by a rSLDS model. This dataset contains behavioral and neural recordings from 81 mice throughout 153 recording sessions, yielding recordings from over 200,000 units in the visual cortex, thalamus, hippocampus, and midbrain. All data in this dataset were collected with Neuropixels 1.0 probes. Probes were retracted from the brain at a rate of 1 mm/s, after which the probe cartridge was lifted up to its full height.

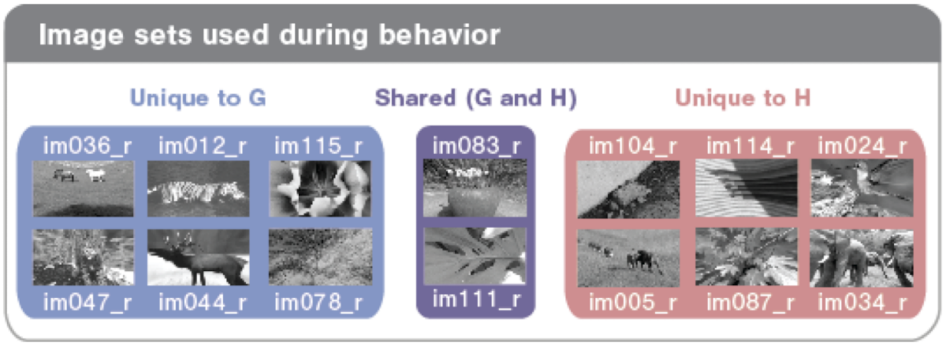


Figure 1: Images used for training and recording sessions [4]

All mice are required to receive the standardized training successively with static grating stimuli, then transition to flashed gratings, and subsequently learn to detect changes in the natural scene images. There are two sets of images used for the task, G and H, both containing eight natural images in both training and recording experiment. After meeting the training criteria and habituating the experimental environment, each mouse undergoes the neural activity recording experiments with Neuropixels once per day. The mice can perform the change detection task with one set of natural images as visual stimuli, and on the second day, the mice are presented with another set of images. This can provide insights into the effects of new stimuli on neural coding. During this period, the task stimuli are presented to wthe mice with the lick spout retracted so that they do not receive the water reward.

Throughout the recording session, mice perform a change detection task, where a series of images are presented to the mice as shown in the Figure 1. In this go/no-go task, mice will earn water rewards, if they correctly report when the identity of the image changes. As for this paper, we are interested in the neural responses generated by watching the same image for a certain period of time.
In the subsequent sessions, we present the data and related experiments used in the paper, the main mathematical models, and their applications to neural data analysis.

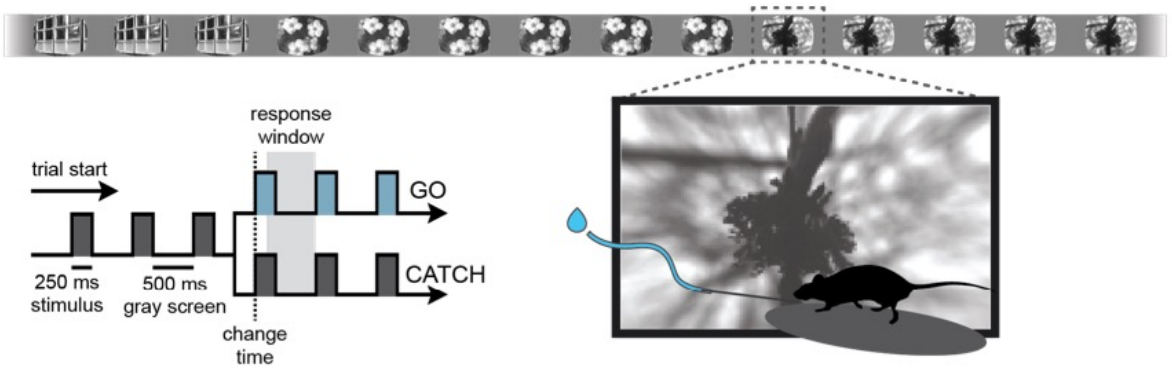


Figure 2: Visual change detection task. The continuous stream of stimulus presentations (250ms per stimulus) and the inter-stimulus gray screen (500ms) that are shown to the mouse during a behavior session are shown in the top row. The task's two trial types are depicted in the lower left: "go" trials, in which the identity of the stimulus is changed and the mouse must lick within a 750ms response window to receive a water reward, and "catch" trials, in which the identity of the image is unchanged and false alarm licking behavior is measured. The bottom right shows the behavioral setup, with stimuli displayed on a monitor facing the right eye of the mouse, a lick spout for response detection via a capacitive sensor and water reward delivery, and a running wheel ([4]).

2 Theoretical Background

2.1 Switching Linear Dynamical Systems

Switching linear dynamical system (SLDS) allows the given system switches between different linear dynamic systems depending on different conditions. In SLDS model, we can break down complex, nonlinear time series data into sequences of coherent discrete units which exhibit more complex behavior than a single linear dynamic system.

For each timestamp of data, we can get a discrete latent state $z_t \in 1, 2, ..., K$ that following Markovian dynamics,

$$z_{t+1}|z_t, \{\pi_k\}_{k=1}^K \sim \pi_{z_t} \tag{1}$$

where $\{\pi_k\}_{k=1}^K$ is the Markov transition matrix and $\pi_k \in [0,1]^K$ is is its kth row. Additionally, we have a continuous latent state $x_t \in \mathbb{R}^m$ which follows conditionally linear dynamics, and the discrete latent state z_t determines the linear dynamical system used at time t:

$$x_{t+1} = A_{z_{t+1}} x_t + b_{z_{t+1}} + v_t, v_t \overset{iid}{\sim} \mathcal{N}(0, Q_{z_{t+1}}) \tag{2}$$

for matrices $A_k, Q_k \in \mathbb{R}^{M \times M}$ and vectors $b_k \in \mathbb{R}^M$ for k = 1,2,...,K. At last, we can generate a linear Gaussian observation $y_t \in \mathbb{R}^N$ at each time t from the corresponding latent continuous state,

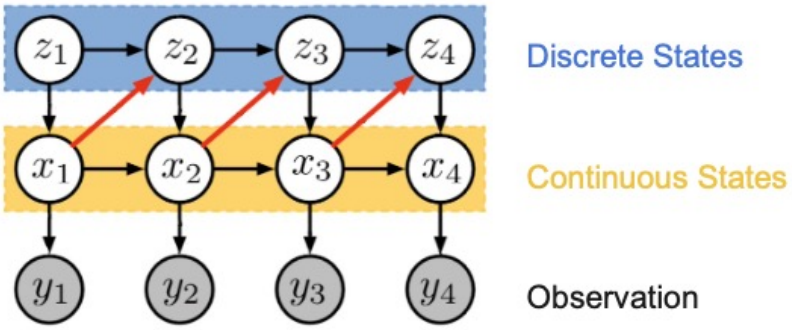


Figure 3: General models for the recurrent SLDS (rSLDS). Red arrows in the graph are recurrent dependencies of discrete states on continuous latent states.

$$y_t = C_{z_t} x_t + d_{z_t} + w_t, w_t \overset{iid}{\sim} \mathcal{N}(0, S_{z_t}) \tag{3}$$

for matrices $C_{z_t} \in \mathbb{R}^{N \times M}, S_{z_t} \in \mathbb{R}^{N \times N}$, and vector $d_k \in \mathbb{R}^N$.

2.2 Recurrent Switching State Space Models

In SLDS, the discrete latent state z_{t+1} only depends on the previous discrete state z_t and $z_{t+1}|z_t$ is independent of the continuous state x_t. In that case, the SLDS cannot capture the switching dynamical systems of the continuous state x_t and update the discrete state z_{t+1} on time.

In order to deal with this problem, we utilize an extended SLDS with recurrent switching state space models, which is called Recurrent Switching Linear Dynamical Systems (rSLDS). Rather than restricting the discrete states to self-update in SLDS, the rSLDS model allows the discrete state transition probabilities to depend on both the previous discrete latent state and the preceding continuous latent state. The discrete states of the rSLDS are updated by the following function:

$$z_{t+1}|z_t, x_t, \{R_k, r_k\} \sim \pi_{SB}(v_{t+1}), v_{t+1} = R_{z_t} x_t + r_{z_t} \tag{4}$$

where $R_k \in \mathbb{R}^{K-1 \times M}$ is a weight matrix that specifies the recurrent dependencies and $r_k \in \mathbb{R}^{K-1}$ is a bias that captures the Markov dependence of z_{t+1} on z_t. In Figure 2, we can see the red arrows represented the recurrent process where the new discrete states depend on the previous discrete states and the preceding continuous states.

3 Reproduction of results in paper

Linderman and his colleagues used several experiments to test on the performance of rSLDS model in approximating the dynamics of systems. Our group chose to reproduce the result for the prediction of simulated Lorenz dynamics.

3.1 Lorenz attractor

Originally expanded from fluid flow equations in a convecting layer, the Lorenz equations describe a canonical nonlinear dynamical system[3]. Describe by three ordinary differential equations:

$$\begin{aligned}\frac{dx}{dt} &= \sigma(y - x)\\ \frac{dy}{dt} &= x(\rho - z) - y\\ \frac{dz}{dt} &= xy - \beta z\end{aligned}$$

As a classic chaotic system, which is characterized by significant sensitivity to perturbations in initial conditions[6], the Lorenz system has many intriguing properties that matches the interest on switching behavior between states. Firstly, the infinite number of periodic orbits inside the attractor can be clearly divided into two states by division of two hyperplanes. Secondly, the result can only be solved numerically. With no explicit governing function, the division of two states becomes crucial.

3.2 Original results

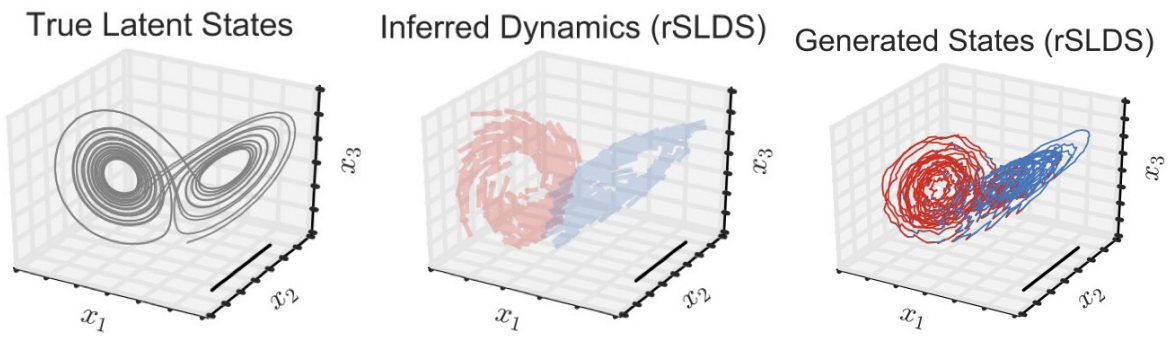


Figure 4: Lorenz attractor dynamic in Linderman's paper. A portion of data is masked, which increases uncertainty. The scale bars are marked in each subplots.[5]

After simulating the true solution numerically, the author used a generalized linear model to derive

$N = 100$ dimensional discrete observations:

$$\rho_{t,n} = \sigma(c_n^T x_t + d_n), \quad y_{t,y} \sim \text{Bern}(\rho_{t,n}), \quad \sigma(x) = \frac{e^x}{(1+e^x)}$$

Due to the inclusion of a masked region of data, the predicted trajectories seem curly. However, the correct state change in dynamics indicates rSLDS's ability of interpolating. The original paper also proposed a comparison with SLDS model, which is not shown here to deviate from our topic.

3.3 Our result

We first simulated the true trajectories of the Lorenz system from the equations we just shown. One of the numerically computed true trajectories is shown in the left graph. Due to the limitation of our engines, we used $N = 10$ simulations instead of 100 in the paper. Because the masking methodology is not documented in details, we choose to add uncertainty in another way.

As described in the paper, the observations y are linear projection of the latent state $x \in \mathbb{R}^3$ with additive Gaussian noise. Rather than directly observing the Lorenz states x, we increased the complexity of dynamic inference by projecting the original 3 dimensional data to 10 dimensional observation using random transformation. Then, we input the 10 dimensional data into rSLDS model to infer the underlying dynamic and the latent states, with 2 discrete states and 3 latent dimensions. As shown in the figure on the right, the model inferred the lower dimensional dynamics from the randomized 10-dimensional space by identifying the two hyperplanes representing two states of the dynamics.

Therefore, it is verified that the model has the ability to extract the hidden dynamics and the corresponding latent states out of chaotic systems.

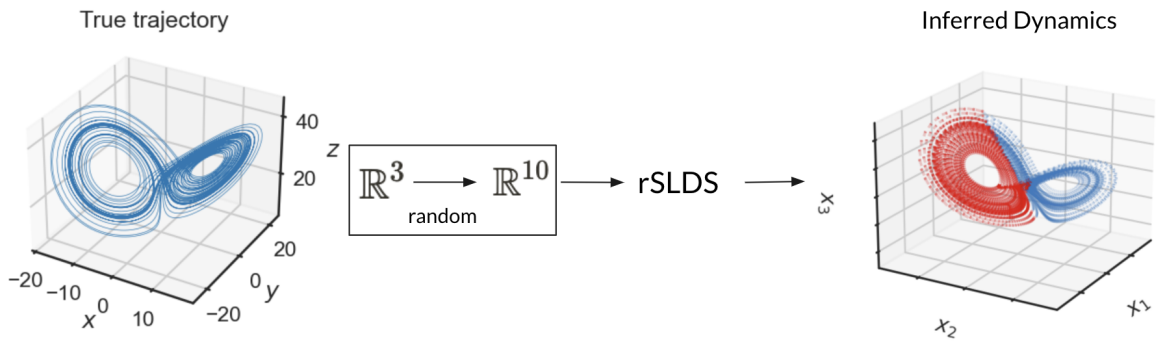


Figure 5: Reproduction of Lorenz attractor dynamic states

4 Novel Question

4.1 Introduction

In the experiment performed by Allen Institute, mice were trained to detect the changes in the images being presented. Within the the process, for most of the time, mice were presented with the same image with their attention still fixed to capture the potential change of the images. Several researches have suggested a potential low-dimensional manifold representation of working memory and time encoding [1] and geometrical information [7]. Following the same path, we are curious on whether there is a low-dimensional representation of the continuous identical input information.

Hippocampus is crucial for the forming of short-delay associative memory between environmental stimuli [8]. However, it is not clear how the neural activity of hippocampal neurons changes dynamically during the process. Hippocampal cornu ammonis (CA1) neurons in rodents play a pivotal role in the processing of hippocampus-dependent memory [7] examined how neurons in the CA1 integrated neural representations and suggested that the hippocampus performs a general computation of task-specific low-dimensional manifolds that contain a geometric representation [7].

4.2 Data acquisition and processing

All experimental data was obtained via the Allen Software Development Kit (AllenSDK) [9]. We chose two recording sessions in total from two mice for our analysis. For both of the recording sessions being analyzed, the image presented are from set H, the recording equipment are both NP.1. Both of the mouses were trained on image set G thus have never seen the the novel images in set H, and both of the mouses are female and of the same genotype (wild-type wt/wt).

Within each recording session, for maintaining quality of the raw data, we manually chose the channels with signal-to-noise-ratio (SNR) greater than 2, interspike-interval-violation less than 1, and firing rate greater than 0.1. Those metrics are embedded in the metadata given by AllenSDK. The raw recording data was given in the form of spiking time. According to the technical whitepaper [4], since we cannot record the exact single neuron, we will refer to the smallest unit we can distinguish as "neuronal unit", which serves as the same function as if we can record single distinguishable neuron. We put the spiking time into non-overlapping bins of 0.01 seconds and calculated the firing rates for all neuron units. By aligning the neuronal units to the corresponding Neuropixel probe where the strongest signal comes, we are able to know the exact location of the neuronal unit, thus knowing the cortical area it records. We selected area CA1 of the hippocampus which is responsible for the memory.

For the image presentation data, we manually selected the time period where the same image was presented for consecutive 10 seconds, and chose the firing rates of all neuronal units corresponding to that time period.

In general, after all the processing steps described above, we are able to get recordings that match our need from 202 neuronal units from mouse 1, and 143 neuronal units from mouse 2. Therefore, we have 202 by 1000 and 143 by 1000 time series to analyze for mouse 1 and mouse 2, respectively.

Below is a figure showing the averaged response of 202 neuronal units for mouse 1 during 10 seconds of being presented the same image, which is the simplest method to track the low-dimension behavior. It can be noticed that even though the image is presented by 0.25 seconds each followed by a 0.5 second of grey screen, we can not only observe the fast dynamics induced by the presentation of each image (the spikes occurring with 0.75 seconds interval), we can also observe a much slower dynamic lasting for around 4 seconds. Similar behavior was also observed at mouse 2. This slow dynamic might be the encoding or the adapting process, which will be analyzed using rSLDS in the next section.

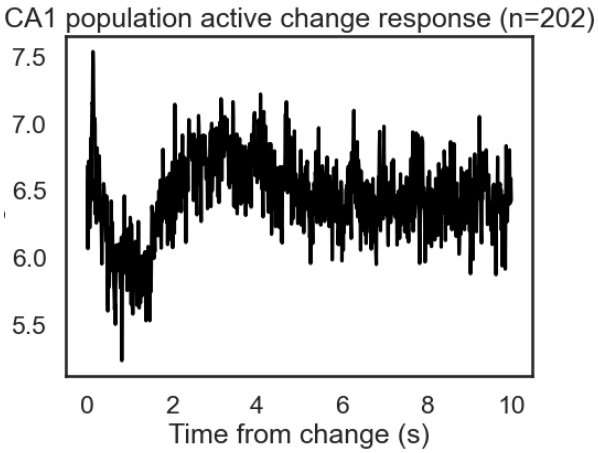


Figure 6: Averaged response in of mouse 1

4.3 Model fitting and result

We then put the two time series data into the rSLDS model to extract. We manually set the number of latent state to be 2 and the latent dimension to be 3. Taking advantage of our observation above on the fast and slow dynamics, we hypothesized that there are two governing transition rules, thus two latent states of the entire 10-second dynamics: adapting and stable. The dimension of latent state was chosen as 3 since it is the highest dimension that we can visualize. The model then inferred the latent dynamics. As from the ELBO (Evidence lower bound) graph suggests, the algorithm had converged.

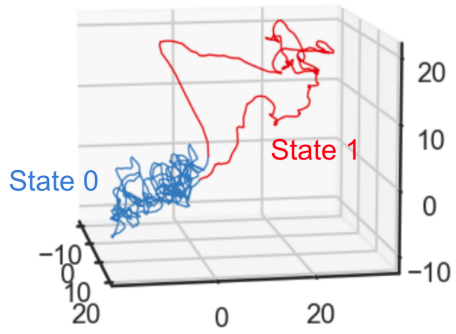


Figure 8: Low-dimensional representation

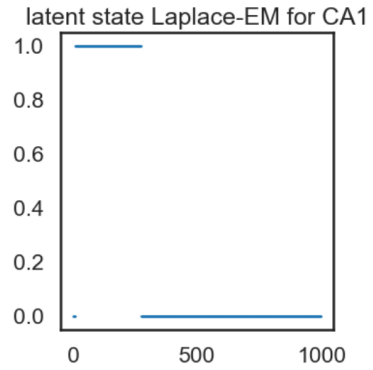


Figure 9: Latent state transition

Figure 10: Inferred dynamics of mouse 1 by rSLDS

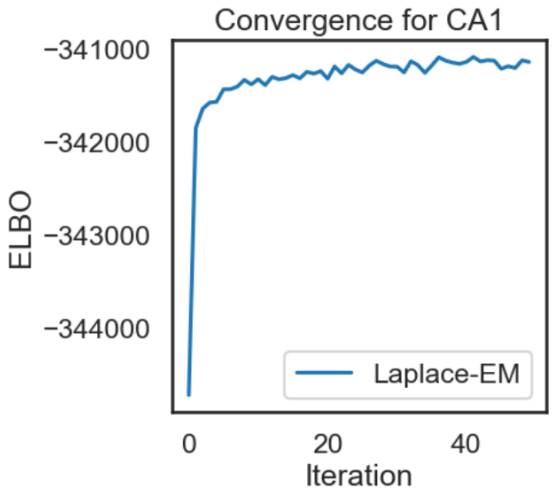


Figure 7: Convergence of rSLDS

The figures below shows the result inferred by rSLDS on mouse 1 and 2.

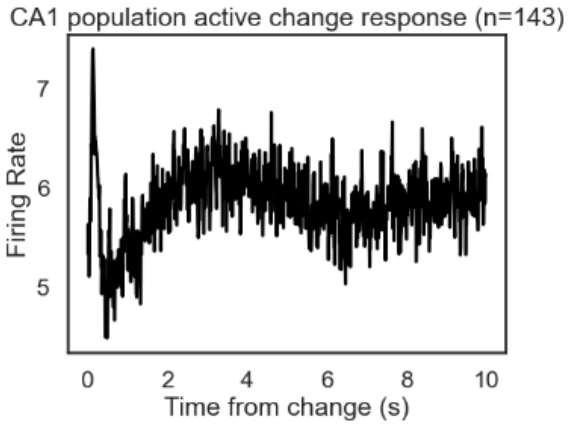


Figure 11: Averaged response in of mouse 2

The rSLDS collapsed the 202-dimensional observation to a 3-dimensional representation in the latent

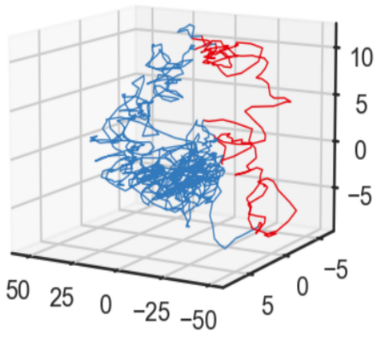


Figure 12: Low-dimensional representation

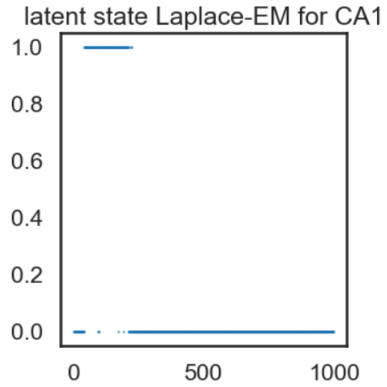


Figure 13: Latent state transition

Figure 14: Inferred dynamics of mouse 2 by rSLDS

space, and two latent states were successfully inferred. From the Figure 9, we can see that the latent states were separated as the two dynamics we hypothesized. The transition between the two states occurs at around 3.5 s, corresponding to the time of the differentiation of the two behaviors we can observe in Figure 6. Within the latent space, we can observe two attractors corresponding to two latent states.

For mouse 2, we can observe the similar pattern where there are two states that corresponds to the two behaviors in the averaged neural activity, and two attractor-liked behaviors. However, the result for mouse 2 is not as clear as mouse 1.

5 Conclusion and Future discussion

By examining the high-dimensional neuronal activities of the mouse being presented same figures, we found found the low-dimensional dynamics underlying the process. In 3-dimensional space, this process is characterized by two distinct attractors where the neurons goes to each in a sequential order. Our findings aligns with previous studies suggesting the potential geometrical structure of task-relevant variables in neural encoding[7]. However, since the inference generated by rSLDS is highly arbitrary to the algorithm itself and the complexity of the figure being presented, it is difficult to infer the meaning of each axis in the latent space. By further investigating the properties of image itself, or separating the images being presented, we might find different or consistent dynamics dependent on the context.

In this paper, we focused on the single cortical area (CA1) of Hippocampus, however, other researches have suggested that memory or information processing is a distributed process [1] involving other brain areas like neocortex. We could elaborate on this topic using a advenced model based on rSLDS for

multiple interacting neural populations [2], which can be used to build a rSLDS model with various cortical datasets. The multi-population model assumes different certain working patterns between neuronal populations at different correspoding states. In the future, we could implement this advanced rSLDS model and try to find the inter-population dynamical systems among different cortical areas to further investigate on this process.

Due to the limitation of computing power, we only trained the dataset of two different mice and compare their results from the trained rSLDS models. However, two mice are not sufficient to find the general biological pattern of mouse's neuron dynamical system. In the future, we could utilize more powerful computing tools like super computer or Azure to train the dataset of more mice. From the larger size of results, we might find some general patterns of those mouse's neural dynamics.

References

[1] Christopher J. Cueva et al. "Low-dimensional dynamics for working memory and time encoding". In: *Proceedings of the National Academy of Sciences* 117.37 (2020), pp. 23021–23032. DOI: 10.1073/pnas.1915984117. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1915984117. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1915984117.

[2] Joshua I. Glaser et al. "Recurrent Switching Dynamical Systems Models for Multiple Interacting Neural Populations". In: *bioRxiv* (2020). DOI: 10.1101/2020.10.21.349282. eprint: https://www.biorxiv.org/content/early/2020/10/22/2020.10.21.349282.full.pdf. URL: https://www.biorxiv.org/content/early/2020/10/22/2020.10.21.349282.

[3] J Guckenheimer. "Review of The Lorenz Equations: Bifurcations, Chaos, and Strange Attractors., by C. Sparrow". In: *The American Mathematical Monthly* 91.5 (1984), pp. 325–326. DOI: 10.2307/2322694. eprint: https://doi.org/10.1137/1027034. URL: https://doi.org/10.2307/2322694.

[4] Allen Institute. *Visual Behavior Neuropixels Technical Whiteppeer*. Accessed: 12-14-2022. URL: https://brainmapportal-live-4cc80a57cd6e400d854-f7fdcae.divio-media.net/filer_public/f7/06/f706855a-a3a1-4a3a-a6b0-3502ad64680f/visualbehaviorneuropixels_technicalwhitepaper.pdf.

[5] Scott W. Linderman et al. "Recurrent switching linear dynamical systems". In: (2016). DOI: 10.48550/ARXIV.1610.08466. eprint: https://arxiv.org/pdf/1610.08466.pdf. URL: https://doi.org/10.48550/arxiv.1610.08466.

[6] Edward Lorenz. *The Essence of Chaos*. University of Washington Press, 1993, pp. 181–206.

[7] Edward H. Nieh et al. "Geometry of abstract learned knowledge in the hippocampus". en. In: *Nature* 595.7865 (July 2021), pp. 80–84. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-021-03652-7. URL: http://www.nature.com/articles/s41586-021-03652-7 (visited on 12/14/2022).

[8] Shogo Takamiya et al. "Hippocampal CA1 Neurons Represent Positive Feedback During the Learning Process of an Associative Memory Task". In: *Frontiers in Systems Neuroscience* 15 (2021). ISSN: 1662-5137. DOI: 10.3389/fnsys.2021.718619. URL: https://www.frontiersin.org/articles/10.3389/fnsys.2021.718619.

[9] *Welcome to the Allen SDK — Allen SDK dev documentation*. URL: https://allensdk.readthedocs.io/en/latest/ (visited on 12/14/2022).

6 Appendix - Code

Before testing out our result, please set up ssm module (https://github.com/lindermanlab/ssm).

6.1 Codes to reproduce graph of Lorenz attractor:

```
import autograd.numpy as np
import autograd.numpy.random as npr
import copy
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
color_names = ["windows blue", "red", "amber", "faded green"]
colors = sns.xkcd_palette(color_names)
sns.set_style("white")
sns.set_context("talk")

import ssm
from ssm.util import random_rotation, find_permutation
from scipy.integrate import solve_ivp

# Global parameters
T = 5000
K = 2
D_obs = 10
D_latent = 3

# Initialize integrator keywords for solve_ivp to replicate the odeint defaults
integrator_keywords = {}
integrator_keywords['rtol'] = 1e-12
integrator_keywords['method'] = 'LSODA'
integrator_keywords['atol'] = 1e-12

# lorenz
def lorenz(t, y, sigma, beta, rho):
    """
    lorenz chaotic differential equation: dy/dt = f(t, y)
```

```
    _t: time tk to evaluate system
    _y: 3D state vector [x, y, z]
    sigma: constant related to Prandtl number
    beta: geomatric physical property of fluid layer
    rho: constant related to the Rayleigh number
    return: [x_dot, y_dot, z_dot]
    """
    return np.array([
        sigma * (y[1] - y[0]),
        y[0] * (rho - y[2]) - y[1],
        (y[0] * y[1]) - (beta * y[2]),
    ])

np.random.seed(100)

# True trajectory
dt = .01
num_trajectories=10
scalesigma=1
scalerho=1
scalebeta=1
sigmas=10 + scalesigma*(np.random.random(num_trajectories)-0.5)
rhos=28*np.ones(num_trajectories) + scalerho*(np.random.random(num_trajectories)-0.5)
betas=2.66667*np.ones(num_trajectories) + scalebeta*(np.random.random(num_trajectories)-0.5)
x_trains=[]
t_trains=[]

for i in range(len(sigmas)):
    t_train = np.arange(0, 50, dt)
    x0_train = [-8, 8, 27]
    t_train_span = (t_train[0], t_train[-1])
    x_train = solve_ivp(lorenz, t_train_span, x0_train, args=(sigmas[i],betas[i],rhos[i]),
                t_eval=t_train, **integrator_keywords).y.T
    x_trains=x_trains+[x_train]
    t_trains=t_trains+[t_train]

x_trains = np.array(x_trains)
```

```
def explore(seed):
  np.random.seed(seed)
  W = np.random.randn(D_obs, D_latent)
  x_trainsd = np.empty([10, 5000, 10])
  for i in range(D_obs):
      for j in range(5000):
          x_trainsd[i, j, :] = W@x_trains[i, j, :]
  try:
    # Fit an rSLDS with its default initialization, using Laplace-EM with a structured
        variational posterior
    rslds = ssm.SLDS(D_obs, K, D_latent,
                transitions="recurrent_only",
                dynamics="diagonal_gaussian",
                emissions="gaussian_orthog",
                single_subspace=True)

    rslds.initialize(x_trainsd[0, :, :])
    q_elbos_lem, q_lem = rslds.fit(x_trainsd[0, :, :], method="laplace_em",
                                variational_posterior="structured_meanfield",
                                initialize=False, num_iters=50, alpha=0.0)
    xhat_lem = q_lem.mean_continuous_states[0]
    # rslds.permute(find_permutation(z_att, rslds.most_likely_states(xhat_lem, y_att)))
    zhat_lem = rslds.most_likely_states(np.array(xhat_lem), np.array(x_trainsd[0, :, :]))
  except:
    print("error")
    return

  plt.figure(figsize=(4, 4))

  return rslds, xhat_lem, zhat_lem, x_trainsd[0, :, :]

plot_model, lemx19, lemz19, y0 = explore(19)

plt.figure(figsize=(4, 4))
ax = plt.axes(projection='3d')
ax.plot(x_trains[0][:,0],x_trains[0][:,1],x_trains[0][:,2])
```

```
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_zlabel('$z$')
ax.set_title('true trajectory')
plt.show()

K = plot_model.K
assert plot_model.D == 3
xyz = lemx19

# Get the probability of each state at each xy location
z = np.argmax(xyz.dot(plot_model.transitions.Rs.T) + plot_model.transitions.r, axis=1)

ax = plt.axes(projection='3d')

for k, (A, b) in enumerate(zip(plot_model.dynamics.As, plot_model.dynamics.bs)):
    dxyzdt_m = xyz.dot(A.T) + b - xyz
    zk = z == k
    if zk.sum(0) > 0:
        ax.quiver(xyz[zk, 0], xyz[zk, 1], xyz[zk, 2],
                  dxyzdt_m[zk, 0], dxyzdt_m[zk, 1], dxyzdt_m[zk, 2],
                  color=colors[k % len(colors)], alpha=0.2)
plt.title("Inferred Dynamics, Laplace-EM")
ax.view_init(20, -150)
ax.set_xlim([-90,90])
ax.set_ylim([-90,90])
ax.set_zlim([-50,50])
ax.set_xticklabels([])
ax.set_yticklabels([])
ax.set_zticklabels([])
ax.grid(True)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')
plt.show()
```

6.2 Codes to analyze Allen Institute data:

```
import numpy as np
import allensdk
import os
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import ssm



from allensdk.brain_observatory.behavior.behavior_project_cache.\
    behavior_neuropixels_project_cache \
    import VisualBehaviorNeuropixelsProjectCache

# Update this to a valid directory in your filesystem. This is where the data will be stored.
cache_dir = Path("/Users/ccai/opt/anaconda3/envs/allenenv/allendata")


cache = VisualBehaviorNeuropixelsProjectCache.from_s3_cache(
           cache_dir=cache_dir)
# get the metadata tables
units_table = cache.get_unit_table()
channels_table = cache.get_channel_table()
probes_table = cache.get_probe_table()
behavior_sessions_table = cache.get_behavior_session_table()
ecephys_sessions_table = cache.get_ecephys_session_table()
# select the sessions that are characterized as 'novel'.
slt_ses = ecephys_sessions_table[ecephys_sessions_table['experience_level']=='Novel']
print('There are %.0f selected sessions' % len(slt_ses) ) # 52 columns in total,
     corresponding to 39 + 10 + 3
slt_ses.head()
plt.hist(unit_channels['firing_rate'].values)
sp = session.stimulus_presentations
sp = sp[(sp['active'] == True) & (sp['image_name'] != 'omitted')][['start_time',
    'stop_time', 'is_change', 'image_name']]
image_list = sp['image_name'].drop_duplicates().tolist()
```

```
image_dict = dict()
for image_data in sp.groupby('image_name'):
    # print(image_data)
    image_dict[str(image_data[0])] = image_data[1]
    #Convenience function to compute the PSTH
def makePSTH(spikes, startTimes, windowDur, binSize=0.01):
    bins = np.arange(0,windowDur+binSize,binSize)
    counts = np.zeros(bins.size-1)
    for i,start in enumerate(startTimes):
        startInd = np.searchsorted(spikes, start)
        endInd = np.searchsorted(spikes, start+windowDur)
        counts = counts + np.histogram(spikes[startInd:endInd]-start, bins)[0]

    counts = counts/startTimes.size
    return counts/binSize, bins



area_change_responses = []
time_before_change = 0
duration = 0.25
for iu, unit in area_units.iterrows():
    unit_spike_times = spike_times[iu]
    unit_change_response, bins = makePSTH(unit_spike_times,
                                          image_dict['im083_r']['start_time'].values-time_before_change,
                                          duration, binSize=0.001)
    area_change_responses.append(unit_change_response)
area_change_responses = np.array(area_change_responses)

#Plot the results
fig, ax = plt.subplots(1,2)
fig.set_size_inches([12,4])

clims = [np.percentile(area_change_responses, p) for p in (0.1,99.9)]
im = ax[0].imshow(area_change_responses, clim=clims)
ax[0].set_title('Active Change Responses for {}'.format(area_of_interest))
ax[0].set_ylabel('Unit number, sorted by depth')
ax[0].set_xlabel('Time from change (s)')
```

```
ax[0].set_xticks(np.arange(0, bins.size-1, 20))
_ = ax[0].set_xticklabels(np.round(bins[:-1:20]-time_before_change, 2))

ax[1].plot(bins[:-1]-time_before_change, np.mean(area_change_responses, axis=0), 'k')
ax[1].set_title('{} population active change response (n={})'\
              .format(area_of_interest, area_change_responses.shape[0]))
ax[1].set_xlabel('Time from change (s)')
ax[1].set_ylabel('Firing Rate')

area_fr = area_change_responses
# extract and make all spike times into an array
area_units_list = np.array(area_units.index.tolist())
area_spike_times = [spike_times.get(iu) for iu in area_units_list]
trial_table = session.trials
valid_trials = trial_table[(trial_table['aborted'] == False) &
    (trial_table['stimulus_change'] == True)]
change_times = trial_table['change_time_no_display_delay'].tolist()
change_times = [x for x in change_times if x == x] # remove all nan
cdt_endtime = max(trial_table['stop_time'].tolist()) # change detection task endtime
print('The last change detection task ends at', cdt_endtime, 's')
for i, iu in enumerate(area_spike_times):
   area_spike_times[i] = [j for j in iu if j <= cdt_endtime]
area_spike_times = np.asarray(area_spike_times)
print(np.shape(area_spike_times))
plt.hist(valid_trials['trial_length'], bins=np.arange(7, 14-0.5, 0.5))
plt.title('non-aborted stimulus-changing trial length')
plt.show()

plt.figure(figsize=(10, 4))
plt.subplot(121)
plt.hist(valid_trials['change_time_no_display_delay'] - valid_trials['start_time'])
plt.title('change time after trial start')
plt.subplot(122)
plt.hist(valid_trials['stop_time'] - valid_trials['change_time_no_display_delay'])
plt.title('trial stop time after change')
plt.show()
```

```
Tmax = len(sp)
dwell_times = np.zeros(Tmax)
start_times = np.zeros(Tmax)
image_names = np.zeros(Tmax)
index = 0
dwell_times[0] = sp['start_time'].iloc[1]-sp['start_time'].iloc[0]
start_times[0] = sp['start_time'].iloc[0]
image_names[0] = sp['image_name'].iloc[0]

for i in range(1,Tmax):
    if sp['image_name'].iloc[i] != sp['image_name'].iloc[i-1]:
        index += 1
        dwell_times[index] += (sp['start_time'].iloc[i]-sp['start_time'].iloc[i-1])
        start_times[index] = sp['start_time'].iloc[i]
        image_names[index] = sp['image_name'].iloc[i]
    else:
        dwell_times[index] += (sp['start_time'].iloc[i]-sp['start_time'].iloc[i-1])

dwell_data = pd.DataFrame({'start_times': start_times, 'dwell_times': dwell_times,
    'image_name': image_names})
dwell_data = dwell_data[(dwell_data['dwell_times'] > 10)]
dwell_data

#Convenience function to compute the PSTH
def makePSTH(spikes, startTimes, windowDur, binSize=0.01):
    bins = np.arange(0,windowDur+binSize,binSize)
    counts = np.zeros(bins.size-1)
    for i,start in enumerate(startTimes):
        startInd = np.searchsorted(spikes, start)
        endInd = np.searchsorted(spikes, start+windowDur)
        counts = counts + np.histogram(spikes[startInd:endInd]-start, bins)[0]

    counts = counts/startTimes.size
    return counts/binSize, bins

def dwell_fun(area_units, dwell_data):
    area_change_responses = []
```

```
    time_before_change = 0
    duration = 10
    for iu, unit in area_units.iterrows():
        unit_spike_times = spike_times[iu]
        unit_change_response, bins = makePSTH(unit_spike_times,
                                        dwell_data['start_times'].values-time_before_change,
                                        duration, binSize=0.01)
        area_change_responses.append(unit_change_response)
    area_change_responses = np.array(area_change_responses)

    #Plot the results
    fig, ax = plt.subplots(1,2)
    fig.set_size_inches([12,4])

    clims = [np.percentile(area_change_responses, p) for p in (0.1,99.9)]
    im = ax[0].imshow(area_change_responses, clim=clims)
    ax[0].set_title('Active Change Responses for {}'.format(area_of_interest))
    ax[0].set_ylabel('Unit number, sorted by depth')
    ax[0].set_xlabel('Time from change (s)')
    ax[0].set_xticks(np.arange(0, bins.size-1, 20))
    _ = ax[0].set_xticklabels(np.round(bins[:-1:20]-time_before_change, 2))

    ax[1].plot(bins[:-1]-time_before_change, np.mean(area_change_responses, axis=0), 'k')
    ax[1].set_title('{} population active change response (n={})'\
                    .format(area_of_interest, area_change_responses.shape[0]))
    ax[1].set_xlabel('Time from change (s)')
    ax[1].set_ylabel('Firing Rate')

    area_fr = area_change_responses

    return area_fr

    # Plot some results
def elbo_graph(n):
    plt.figure(figsize=(4, 4))
    plt.plot(q_elbos_bbvi, label="BBVI")
    plt.plot(q_elbos_lem[1:], label="Laplace-EM")
```

```
    plt.legend()
    plt.xlabel("Iteration")
    plt.ylabel("ELBO")
    plt.title('ELBO vs iteration for '+str(n))
    plt.show()

def state_graph(n):
    plt.figure(figsize=(8, 4))
    plt.subplot(121)
    plt.plot(zhat_lem, '.', markersize = 1)
    plt.title('latent state Laplace-EM for '+str(n))

    plt.subplot(122)
    plt.plot(zhat_bbvi, '.', markersize = 1)
    plt.title('latent state bbvi for '+str(n))
    plt.show()

def plot_trajectory_3d(z, x, ls="-"):
    color_names = ["windows blue", "red", "amber", "faded green"]
    colors = sns.xkcd_palette(color_names)
    sns.set_style("white")
    sns.set_context("talk")
    zcps = np.concatenate(([0], np.where(np.diff(z))[0] + 1, [z.size]))
    ax = plt.axes(projection='3d')
    for start, stop in zip(zcps[:-1], zcps[1:]):
        ax.plot(x[start:stop + 1, 0],
                x[start:stop + 1, 1],
                x[start:stop + 1, 2],
                lw=1, ls=ls, color=colors[z[start] % len(colors)],
                alpha=1.0)

    return ax

    def model(area_of_interest, dwell_data):
    area_units = area_choose(area_of_interest)
    area_fr = dwell_fun(area_units, dwell_data).T
```

```
# Global parameters
T = 250
K = 2
D_obs = area_fr.shape[1]
D_latent = 3

# Fit an rSLDS with its default initialization, using Laplace-EM with a structured
    variational posterior
rslds_lem = ssm.SLDS(D_obs, K, D_latent,
                  transitions="recurrent_only",
                  dynamics="diagonal_gaussian",
                  emissions="gaussian_orthog",
                  single_subspace=True)

rslds_lem.initialize(area_fr)

q_elbos_lem, q_lem = rslds_lem.fit(area_fr, method="laplace_em",
                               variational_posterior="structured_meanfield",
                               initialize=False, num_iters=50, alpha=0.0)
xhat_lem = q_lem.mean_continuous_states[0]
# rslds.permute(find_permutation(z_att, rslds.most_likely_states(xhat_lem, y_att)))
zhat_lem = rslds_lem.most_likely_states(np.array(xhat_lem), np.array(area_fr))

# Fit an rSLDS with its default initialization, using BBVI with a structured variational
    posterior
rslds_bbvi = ssm.SLDS(D_obs, K, D_latent,
                  transitions="recurrent_only",
                  dynamics="diagonal_gaussian",
                  emissions="gaussian_orthog",
                  single_subspace=True)
rslds_bbvi.initialize(area_fr)

q_elbos_bbvi, q_bbvi = rslds_bbvi.fit(area_fr, method="bbvi",
                                 variational_posterior="meanfield",
                                 initialize=False, num_iters=5000)
# Get the posterior mean of the continuous states
xhat_bbvi = q_bbvi.mean[0]
```

```
# Find the permutation that matches the true and inferred states
# rslds.permute(find_permutation(z, rslds.most_likely_states(xhat_bbvi, y_att)))
zhat_bbvi = rslds_bbvi.most_likely_states(xhat_bbvi, area_fr)


q_elbos_bbvi=np.array(q_elbos_bbvi)
q_elbos_lem=np.array(q_elbos_lem)

elbo_graph(area_of_interest)
plot_trajectory_3d(zhat_lem, xhat_lem)
state_graph(area_of_interest)
plot_trajectory_3d(zhat_bbvi, xhat_bbvi)
```

7 Appendix - Original paper

Recurrent Switching Linear Dynamical Systems

Scott W. Linderman
Columbia University

Andrew C. Miller
Harvard University

Ryan P. Adams
Harvard University and Twitter

David M. Blei
Columbia University

Liam Paninski
Columbia University

Matthew J. Johnson
Harvard University

Abstract

Many natural systems, such as neurons firing in the brain or basketball teams traversing a court, give rise to time series data with complex, nonlinear dynamics. We can gain insight into these systems by decomposing the data into segments that are each explained by simpler dynamic units. Building on switching linear dynamical systems (SLDS), we present a new model class that not only discovers these dynamical units, but also explains how their switching behavior depends on observations or continuous latent states. These "recurrent" switching linear dynamical systems provide further insight by discovering the conditions under which each unit is deployed, something that traditional SLDS models fail to do. We leverage recent algorithmic advances in approximate inference to make Bayesian inference in these models easy, fast, and scalable.

1 Introduction

Complex dynamical behaviors can often be broken down into simpler units. A basketball player finds the right court position and starts a pick and roll play. A mouse senses a predator and decides to dart away and hide. A neuron's voltage first fluctuates around a baseline until a threshold is exceeded; it spikes to peak depolarization, and then returns to baseline. In each of these cases, the switch to a new mode of behavior can depend on the continuous state of the system or on external factors. By discovering these behavioral units and their switching dependencies, we can gain insight into complex data-generating processes.

This paper proposes a class of recurrent state space models that captures these dependencies and a Bayesian inference and learning algorithm that is computationally tractable and scalable to large datasets. We extend switching linear-Gaussian dynamical systems (SLDS) [Ackerson and Fu, 1970, Chang and Athans, 1978, Hamilton, 1990, Ghahramani and Hinton, 1996, Murphy, 1998, Fox et al., 2009] by introducing a class of models in which the discrete switches can depend on the continuous latent state and exogenous inputs through a logistic regression. Previous models including this dependence, like the piecewise affine framework for hybrid dynamical systems [Sontag, 1981], abandon the conditional linear-Gaussian structure in the continuous states and thus greatly complicate inference [Paoletti et al., 2007, Juloski et al., 2005]. Our main technical contribution is a new inference algorithm that leverages auxiliary variable methods [Polson, Scott, and Windle, 2013, Linderman, Johnson, and Adams, 2015] to make inference both fast and easy.

The class of models and the corresponding learning and inference algorithms we develop have several advantages for understanding rich time series data. First, these models decompose data into simple segments and attribute segment transitions to changes in latent state or environment; this provides interpretable representations of data dynamics. Second, we fit these models using fast, modular Bayesian inference algorithms; this makes it easy to handle missing data, multiple observation modalities, and hierarchical extensions. Finally, these models are interpretable, readily able to incorporate prior information, and generative; this lets us take advantage of a variety of tools for model validation and checking.

In the following section we provide background on the key models and inference techniques on which our method builds. Next, we introduce the class of recurrent switching state space models, and then explain the main algorithmic contribution that enables fast learning and inference. Finally, we illustrate the method on synthetic data experiments and an application to recordings of professional basketball players.

2 Background

Our model has two main components: switching linear dynamical systems and stick-breaking logistic regression. Here we review these components and fix the notation we will use throughout the paper.

2.1 Switching linear dynamical systems

Switching linear dynamical system models (SLDS) break down complex, nonlinear time series data into sequences of simpler, reused dynamical modes. By fitting an SLDS to data, we not only learn a flexible nonlinear generative model, but also learn to parse data sequences into coherent discrete units.

The generative model is as follows. At each time $t = 1, 2, \ldots, T$ there is a discrete latent state $z_t \in \{1, 2, \ldots, K\}$ that following Markovian dynamics,

$$z_{t+1} \,|\, z_t, \{\pi_k\}_{k=1}^K \sim \pi_{z_t} \tag{1}$$

where $\{\pi_k\}_{k=1}^K$ is the Markov transition matrix and $\pi_k \in [0,1]^K$ is its kth row. In addition, a continuous latent state $x_t \in \mathbb{R}^M$ follows conditionally linear (or affine) dynamics, where the discrete state z_t determines the linear dynamical system used at time t:

$$x_{t+1} = A_{z_{t+1}} x_t + b_{z_{t+1}} + v_t, \quad v_t \overset{\text{iid}}{\sim} \mathcal{N}(0, Q_{z_{t+1}}), \tag{2}$$

for matrices $A_k, Q_k \in \mathbb{R}^{M \times M}$ and vectors $b_k \in \mathbb{R}^M$ for $k = 1, 2, \ldots, K$. Finally, at each time t a linear Gaussian observation $y_t \in \mathbb{R}^N$ is generated from the corresponding latent continuous state,

$$y_t = C_{z_t} x_t + d_{z_t} + w_t, \qquad w_t \overset{\text{iid}}{\sim} \mathcal{N}(0, S_{z_t}), \tag{3}$$

for $C_k \in \mathbb{R}^{N \times M}$, $S_k \in \mathbb{R}^{N \times N}$, and $d_k \in \mathbb{R}^N$. The system parameters comprise the discrete Markov transition matrix and the library of linear dynamical system matrices, which we write as

$$\theta = \{(\pi_k, A_k, Q_k, b_k, C_k, S_k, d_k)\}_{k=1}^K.$$

For simplicity, we will require C, S, and d to be shared among all discrete states in our experiments. In general, equations (2) and (3) can be extended to include linear dependence on exogenous inputs, $u_t \in \mathbb{R}^P$, as well.

To learn an SLDS using Bayesian inference, we place conjugate Dirichlet priors on each row of the transition matrix and conjugate matrix normal inverse Wishart (MNIW) priors on the linear dynamical system parameters, writing

$$\pi_k \,|\, \alpha \overset{\text{iid}}{\sim} \text{Dir}(\alpha), \quad (A_k, b_k), Q_k \,|\, \lambda \overset{\text{iid}}{\sim} \text{MNIW}(\lambda),$$
$$(C_k, d_k), S_k \,|\, \eta \overset{\text{iid}}{\sim} \text{MNIW}(\eta),$$

where α, λ, and η denote hyperparameters.

2.2 Stick-breaking logistic regression and Pólya-gamma augmentation

Another component of the recurrent SLDS is a stick-breaking logistic regression, and for efficient block inference updates we leverage a recent Pólya-gamma augmentation strategy [Linderman, Johnson, and Adams, 2015]. This augmentation allows certain logistic regression evidence potentials to appear as conditionally Gaussian potentials in an augmented distribution, which enables our fast inference algorithms.

Consider a logistic regression model from regressors $x \in \mathbb{R}^M$ to a categorical distribution on the discrete variable $z \in \{1, 2, \ldots, K\}$, written as

$$z \,|\, x \sim \pi_{\mathsf{SB}}(\nu), \quad \nu = Rx + r,$$

where $R \in \mathbb{R}^{K-1 \times M}$ is a weight matrix and $r \in \mathbb{R}^{K-1}$ is a bias vector. Unlike the standard multiclass logistic regression, which uses a softmax link function, we instead use a stick-breaking link function $\pi_{\mathsf{SB}} : \mathbb{R}^{K-1} \to [0,1]^K$, which maps a real vector to a normalized probability vector via the stick-breaking process

$$\pi_{\mathsf{SB}}(\nu) = \begin{pmatrix} \pi_{\mathsf{SB}}^{(1)}(\nu) & \cdots & \pi_{\mathsf{SB}}^{(K)}(\nu) \end{pmatrix},$$
$$\pi_{\mathsf{SB}}^{(k)}(\nu) = \sigma(\nu_k) \prod_{j<k} (1 - \sigma(\nu_j)) = \sigma(\nu_k) \prod_{j<k} \sigma(-\nu_j),$$

for $k = 1, 2, \ldots, K-1$ and $\pi_{\mathsf{SB}}^{(K)}(\nu) = \prod_{k=1}^K \sigma(-\nu_k)$, where $\sigma(x) = e^x/(1+e^x)$ denotes the logistic function. The probability mass function $p(z \,|\, x)$ is

$$p(z \,|\, x) = \prod_{k=1}^K \sigma(\nu_k)^{\mathbb{I}[z=k]} \sigma(-\nu_k)^{\mathbb{I}[z>k]}$$

where $\mathbb{I}[\cdot]$ denotes an indicator function that takes value 1 when its argument is true and 0 otherwise.

If we use this regression model as a likelihood $p(z \,|\, x)$ with a Gaussian prior density $p(x)$, the posterior density $p(x \,|\, z)$ is non-Gaussian and does not admit easy Bayesian updating. However, Linderman, Johnson, and Adams [2015] show how to introduce Pólya-gamma auxiliary variables $\omega = \{\omega_k\}_{k=1}^K$ so that the conditional density $p(x \,|\, z, \omega)$ becomes Gaussian. In particular, by choosing $\omega_k \,|\, x, z \sim \text{PG}(\mathbb{I}[z \geq k], \nu_k)$, we have,

$$x \,|\, z, \omega \sim \mathcal{N}(\Omega^{-1}\kappa,\ \Omega^{-1}),$$

where the mean vector $\Omega^{-1}\kappa$ and covariance matrix Ω^{-1} are determined by

$$\Omega = \text{diag}(\omega), \qquad \kappa_k = \mathbb{I}[z = k] - \frac{1}{2}\mathbb{I}[z \geq k].$$

Thus instantiating these auxiliary variables in a Gibbs sampler or variational mean field inference algorithm enables efficient block updates while preserving the same marginal posterior distribution $p(x \,|\, z)$.

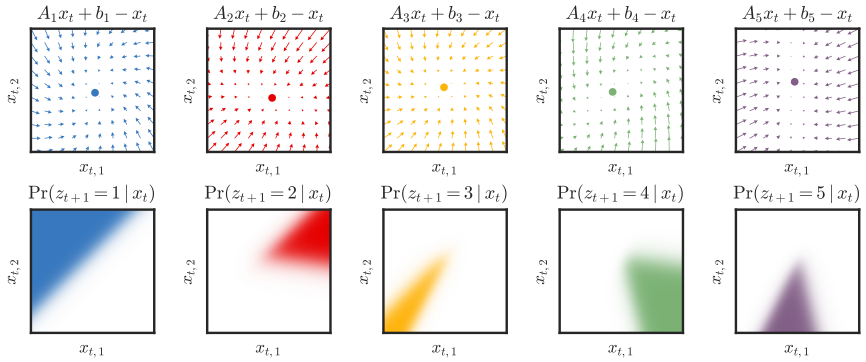


Figure 1: A draw from the prior over recurrent switching linear dynamical systems with $K = 5$ discrete states shown in different colors. **(Top)** The linear dynamics of each latent state. Dots show the fixed point $(I - A_k)^{-1}b_k$. **(Bottom)** The conditional $p(z_{t+1}|x_t)$ plotted as a function of x_t (white=0; color=1). Note that the stick breaking construction iteratively partitions the continuous space with linear hyperplanes. For simpler plotting, in this example we restrict $p(z_{t+1} \mid x_t, z_t) = p(z_{t+1}|x_t)$.

3 Recurrent Switching State Space Models

The discrete states in the "classical" SLDS of Section 2.1 are generated via an *open loop*: the discrete state z_{t+1} is a function only of the preceding discrete state z_t, and $z_{t+1} \mid z_t$ is independent of the continuous state x_t. That is, if a discrete switch should occur whenever the continuous state enters a particular region of state space, the SLDS will be unable to learn this dependence.

We introduce the *recurrent switching linear dynamical system* (rSLDS), an extension of the SLDS to model these dependencies directly. Rather than restricting the discrete states to open-loop Markovian dynamics as in Eq. (1), the rSLDS allows the discrete state transition probabilities to depend on additional covariates, and in particular on the preceding continuous latent state. That is, the discrete states of the rSLDS are generated as

$$\begin{aligned} z_{t+1} \mid z_t, x_t, \{R_k, r_k\} &\sim \pi_{\mathsf{SB}}(\nu_{t+1}), \\ \nu_{t+1} &= R_{z_t} x_t + r_{z_t}, \end{aligned} \tag{4}$$

where $R_k \in \mathbb{R}^{K-1 \times M}$ is a weight matrix that specifies the recurrent dependencies and $r_k \in \mathbb{R}^{K-1}$ is a bias that captures the Markov dependence of z_{t+1} on z_t. The remainder of the rSLDS generative process follows that of the SLDS from Eqs. (2)-(3). See Figure 2a for a graphical model, where the edges representing the new dependencies of the discrete states on the continuous latent states are highlighted in red. As with the standard SLDS, both the discrete and continuous rSLDS dynamics could be extended with linear dependence on exogenous inputs, u_t, as well.

Figure 1 illustrates an rSLDS with $K = 5$ discrete states and $M = 2$ dimensional continuous states. Each discrete state corresponds to a set of linear dynamics defined by A_k and b_k, shown in the top row. The transition probability, π_t, is a function of the previous states z_{t-1} and x_{t-1}. We show only the dependence on x_{t-1} in the bottom row. Each panel shows the conditional probability, $\Pr(z_{t+1} = k \mid x_t)$, as a colormap ranging from zero (white) to one (color). Due to the logistic stick breaking, the latent space is iteratively partitioned with linear hyperplanes.

There are several useful special cases of the rSLDS.

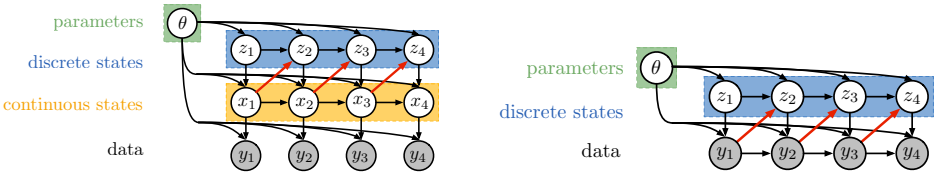
Recurrent ARHMM (rAR-HMM) Just as the autoregressive HMM (AR-HMM) is a special case of the SLDS in which we observe the states $x_{1:T}$ directly, we can define an analogous rAR-HMM. See Figure 2b for a graphical model, where the edges representing the dependence of the discrete states on the continuous observations are highlighted in red.

Shared rSLDS (rSLDS(s)) Rather than having separate recurrence weights (and hence a separate partition) for each value of z_t, we can share the recurrence weights as,

$$\nu_{t+1} = Rx_t + r_{z_t}.$$

Recurrence-Only (rSLDS(ro)) There is no dependence on z_t in this model. Instead,

$$\nu_{t+1} = Rx_t + r.$$



(a) rSLDS (b) rAR-HMM

Figure 2: Graphical models for the recurrent SLDS (rSLDS) and recurrent AR-HMM (rAR-HMM). Edges that represent recurrent dependencies of discrete states on continuous observations or continuous latent states are highlighted in red.

While less flexible, this model is eminently interpretable, easy to visualize, and, as we show in experiments, works well for many dynamical systems. The example in Figure 1 corresponds to this special case.

Standard SLDS We can recover the standard SLDS with $\nu_{t+1} = r_{z_t}$.

Recurrent Sticky SLDS Rather than allowing the continuous states to govern the entire distribution over next discrete states, x_t may only affect whether or not to stay in the current state. That is,

$$s_{t+1} \sim \text{Bern}(\sigma(r_{z_t}^\mathsf{T} x_t)), \quad z_{t+1} \sim \begin{cases} \delta_{z_t} & \text{if } s_{t+1} = 1, \\ \widetilde{\pi}_{z_t} & \text{o.w.} \end{cases}$$

where $\widetilde{\pi}_k \in [0,1]^{K-1}$ is a distribution on states *other than* the current state. The "stay-or-leave" variable, s_{t+1}, depends on the current location.

4 Bayesian Inference

Adding the recurrent dependencies from the latent continuous states to the latent discrete states introduces new inference challenges. While block Gibbs sampling in the standard SLDS can be accomplished with message passing because $x_{1:T}$ is conditionally Gaussian given $z_{1:T}$ and $y_{1:T}$, the dependence of z_{t+1} on x_t renders the recurrent SLDS non-conjugate. To develop a message-passing procedure for the rSLDS, we first review standard SLDS message passing, then show how to leverage a Pólya-gamma augmentation along with message passing to perform efficient Gibbs sampling in the rSLDS. We discuss stochastic variational inference [Hoffman et al., 2013] in the supplementary material.

4.1 Message Passing

First, consider the conditional density of the latent continuous state sequence $x_{1:T}$ given all other variables, which is proportional to

$$\prod_{t=1}^{T-1} \psi(x_t, x_{t+1}, z_{t+1})\, \psi(x_t, z_{t+1}) \prod_{t=1}^{T} \psi(x_t, y_t),$$

where $\psi(x_t, x_{t+1}, z_{t+1})$ denotes the potential from the conditionally linear-Gaussian dynamics and $\psi(x_t, y_t)$ denotes the evidence potentials. The potentials $\psi(x_t, z_{t+1})$ arise from the new dependencies in the rSLDS and do not appear in the standard SLDS. This factorization corresponds to a chain-structured undirected graphical model with nodes for each time index.

We can sample from this conditional distribution using message passing. The message from time t to time $t' = t+1$, denoted $m_{t \to t'}(x_{t'})$, is computed via

$$\int \psi(x_t, y_t)\psi(x_t, z_{t'})\psi(x_t, x_{t'}, z_{t'}) m_{t'' \to t}(x_t)\, \mathrm{d}x_t, \quad (5)$$

where t'' denotes $t-1$. If the potentials were all Gaussian, as is the case without the rSLDS potential $\psi(x_t, z_{t+1})$, this integral could be computed analytically. We pass messages forward once, as in a Kalman filter, and then sample backward. This constructs a joint sample $\hat{x}_{1:T} \sim p(x_{1:T})$ in $O(T)$ time. A similar procedure can be used to jointly sample the discrete state sequence, $z_{1:T}$, given the continuous states and parameters. However, this computational strategy for sampling the latent continuous states breaks down when including the non-Gaussian rSLDS potential $\psi(x_t, z_{t+1})$.

Note that it is straightforward to handle missing data in this formulation; if the observation y_t is omitted, we simply have one fewer potential in our graph.

4.2 Augmentation for non-Gaussian Factors

The challenge presented by the recurrent SLDS is that $\psi(x_t, z_{t+1})$ is not a linear Gaussian factor; rather, it is a categorical distribution whose parameter depends nonlinearly on x_t. Thus, the integral in the message computation (5) is not available in closed form. There are a number of methods of approximating such integrals, like particle filtering [Doucet et al.,

2000] or Laplace approximations [Tierney and Kadane, 1986], but here we take an alternative approach using the recently developed Pólya-gamma augmentation scheme [Polson et al., 2013], which renders the model conjugate by introducing an auxiliary variable in such a way that the resulting marginal leaves the original model intact.

According to the stick breaking transformation described in Section 2.2, the non-Gaussian factor is

$$\psi(x_t, z_{t+1}) = \prod_{k=1}^{K} \sigma(\nu_{t+1,k})^{\mathbb{I}[z_{t+1}=k]}\, \sigma(-\nu_{t+1,k})^{\mathbb{I}[z_{t+1}>k]},$$

where $\nu_{t+1,k}$ is the k-th dimension of ν_{t+1}, as defined in (4). Recall that ν_{t+1} is linear in x_t. Expanding the definition of the logistic function, we have,

$$\psi(x_t, z_{t+1}) = \prod_{k=1}^{K-1} \frac{(e^{\nu_{t+1,k}})^{\mathbb{I}[z_{t+1}=k]}}{(1+e^{\nu_{t+1,k}})^{\mathbb{I}[z_{t+1}\geq k]}}. \quad (6)$$

The Pólya-gamma augmentation targets exactly such densities, leveraging the following integral identity:

$$\frac{(e^{\nu})^a}{(1+e^{\nu})^b} = 2^{-b} e^{\kappa\nu} \int_0^\infty e^{-\omega\nu^2/2} p_{\mathrm{PG}}(\omega \mid b, 0)\, \mathrm{d}\omega, \quad (7)$$

where $\kappa = a - b/2$ and $p_{\mathrm{PG}}(\omega \mid b, 0)$ is the density of the Pólya-gamma distribution, $\mathrm{PG}(b, 0)$, which does not depend on ν.

Combining (6) and (7), we see that $\psi(x_t, z_{t+1})$ can be written as a marginal of a factor on the augmented space, $\psi(x_t, z_{t+1}, \omega_t)$, where $\omega_t \in \mathbb{R}_+^{K-1}$ is a vector of auxiliary variables. As a function of ν_{t+1}, we have

$$\psi(x_t, z_{t+1}, \omega_t) \propto \prod_{k=1}^{K-1} \exp\left\{\kappa_{t+1,k}\, \nu_{t+1,k} - \tfrac{1}{2}\omega_{t,k}\, \nu_{t+1,k}^2\right\},$$

where $\kappa_{t+1,k} = \mathbb{I}[z_{t+1} = k] - \frac{1}{2}\mathbb{I}[z_{t+1} \geq k]$. Hence,

$$\psi(x_t, z_{t+1}, \omega_t) \propto \mathcal{N}(\nu_{t+1} \mid \Omega_t^{-1}\kappa_{t+1},\, \Omega_t^{-1}),$$

with $\Omega_t = \mathrm{diag}(\omega_t)$ and $\kappa_{t+1} = [\kappa_{t+1,1} \ldots, \kappa_{t+1,K-1}]$. Again, recall that ν_{t+1} is a linear function of x_t. Thus, after augmentation, the potential on x_t is effectively Gaussian and the integrals required for message passing can be written analytically. Finally, the auxiliary variables are easily updated as well, since $\omega_{t,k} \mid x_t, z_{t+1} \sim \mathrm{PG}(\mathbb{I}[z_{t+1} \geq k], \nu_{t+1,k})$.

4.3 Updating Model Parameters

Given the latent states and observations, the model parameters benefit from simple conjugate updates. The dynamics parameters have conjugate MNIW priors, as do the emission parameters. The recurrence weights are also conjugate under a MNIW prior, given the auxiliary variables $\omega_{1:T}$. We set the hyperparameters of these priors such that random draws of the dynamics are typically stable and have nearly unit spectral radius in expectation, and we set the mean of the recurrence bias such that states are equiprobable in expectation. We will discuss initialization in the following section.

4.4 Initialization

Given the complexity of these models, it is important to initialize the parameters and latent states with reasonable values. We used the following initialization procedure: (i) use probabilistic PCA or factor analysis to initialize the continuous latent states, $x_{1:T}$, and the observation, C, S, and d; (ii) fit an AR-HMM to $x_{1:T}$ in order to initialize the discrete latent states, $z_{1:T}$, and the dynamics models, $\{A_k, Q_k, b_k\}$; and then (iii) greedily fit a decision list with logistic regressions at each node in order to determine a permutation of the latent states most amenable to stick breaking. In practice, the last step alleviates the undesirable dependence on ordering that arises from the stick breaking formulation. We discuss this and alternative approaches in more depth in the supplementary material.

With this initialization, the Gibbs sampler refines the parameter and state estimates and explores (at least a mode of) the posterior.

5 Experiments

We demonstrate the potential of recurrent dynamics in a variety of settings. First, we consider a case in which the underlying dynamics truly follow an rSLDS, which illustrates some of the nuances involved in fitting these rich systems. With this experience, we then apply these models to simulated data from a canonical nonlinear dynamical system – the Lorenz attractor – and find that its dynamics are well-approximated by an rSLDS. Moreover, by leveraging the Pólya-gamma augmentation, these nonlinear dynamics can even be recovered from discrete time series with large swaths of missing data, as we show with a Bernoulli-Lorenz model. Finally, we apply these recurrent models to real trajectories on basketball players and discover interpretable, location-dependent behavioral states.

5.1 Synthetic NASCAR®

We begin with a toy example in which the true dynamics trace out ovals, like a stock car on a NASCAR®

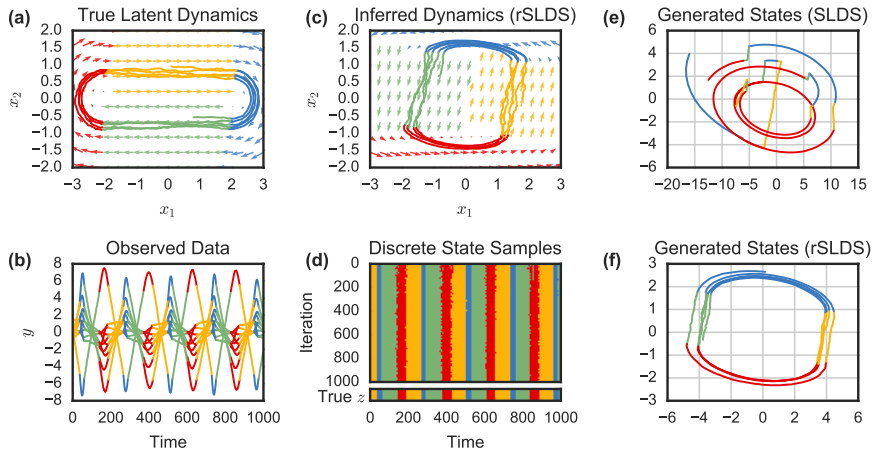


Figure 3: Synthetic NASCAR®, an example of Bayesian inference in a recurrent switching linear dynamical system (rSLDS). **(a)** In this case, the true dynamics switch between four states, causing the continuous latent state, $x_t \in \mathbb{R}^2$, to trace ovals like a car on a NASCAR® track. The dynamics of the most likely discrete state at a particular location are shown with arrows. **(b)** The observations, $y_t \in \mathbb{R}^{10}$, are a linear projection with additive Gaussian noise (colors not given; for visualization only). **(c)** The rSLDS correctly infers the continuous state trajectory, up to affine transformation. It also learns to partition the continuous space into discrete regions with different dynamics. **(d)** Posterior samples of the discrete state sequence match the true discrete states, and show uncertainty near the change points. **(e)** Generative samples from a standard SLDS differ dramatically from the true latent states in (a), since the run lengths in the SLDS are simple geometric random variables that are independent of the continuous state. **(f)** In contrast, the rSLDS learns to generate states that shares the same periodic nature of the true model.

track.[1] There are four discrete states, $z_t \in \{1, \ldots, 4\}$, that govern the dynamics of a two dimensional continuous latent state, $x_t \in \mathbb{R}^2$. Fig. 3a shows the dynamics of the most likely state for each point in latent space, along with a sampled trajectory from this system. The observations, $y_t \in \mathbb{R}^{10}$ are a linear projection of the latent state with additive Gaussian noise. The 10 dimensions of y_t are superimposed in Fig. 3b. We simulated $T = 10^4$ time-steps of data and fit an rSLDS to these data with 10^3 iterations of Gibbs sampling.

Fig. 3c shows a sample of the inferred latent state and its dynamics. It recovers the four states and a rotated oval track, which is expected since the latent states are non-identifiable up to invertible transformation. Fig. 3d plots the samples of $z_{1:1000}$ as a function of Gibbs iteration, illustrating the uncertainty near the change-points.

From a decoding perspective, both the SLDS and the rSLDS are capable of discovering the discrete latent states; however, the rSLDS is a much more effective generative model. Whereas the standard SLDS has only a Markov model for the discrete states, and hence generates the geometrically distributed state durations in Fig 3e, the rSLDS leverages the location of the latent state to govern the discrete dynamics and generates the much more realistic, periodic data in Fig. 3f.

[1] Unlike real NASCAR drivers, these states turn right.

5.2 Lorenz Attractor

Switching linear dynamical systems offer a tractable approximation to complicated nonlinear dynamical systems. Indeed, one of the principal motivations for these models is that once they have been fit, we can leverage decades of research on optimal filtering, smoothing, and control for linear systems. However, as we show in the case of the Lorenz attractor, the standard SLDS is often a poor generative model, and hence has difficulty interpolating over missing data. The recurrent SLDS remedies this by connecting discrete and continuous states.

Fig. 4a shows the states of a Lorenz attractor, whose nonlinear dynamics are given by,

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \begin{bmatrix} \alpha(x_2 - x_1) \\ x_1(\beta - x_3) - x_2 \\ x_1 x_2 - \gamma x_3 \end{bmatrix}.$$

Though nonlinear and chaotic, we see that the Lorenz attractor roughly traces out ellipses in two opposing

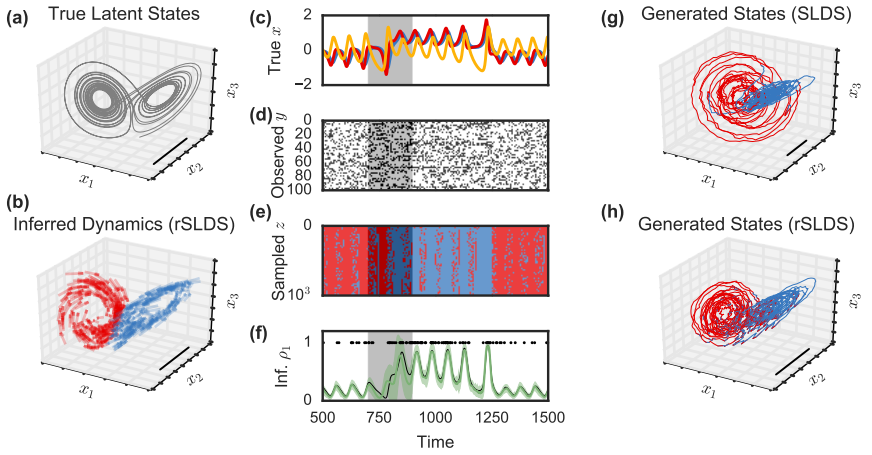


Figure 4: A recurrent switching linear dynamical system (rSLDS) applied to simulated data from a Lorenz attractor — a canonical nonlinear dynamical system. **(a)** The Lorenz attractor chaotically oscillates between two planes. Scale bar shared between (a), (b), (g) and (h). **(b)** The rSLDS, with $x_t \in \mathbb{R}^3$, identifies these two modes and their approximately linear dynamics, up to an invertible transformation. It divides the space in half with a linear hyperplane. **(c)** Unrolled over time, we see the points at which the Lorenz system switches from one plane to the other. Gray window denotes masked region of the data. **(d)** The observations come from a generalized linear model with Bernoulli observations and a logistic link function. **(e)** Samples of the discrete state show that the rSLDS correctly identifies the switching time even in the missing data. **(f)** The inferred probabilities (green) for the first output dimension along with the true event times (black dots) and the true probabilities (black line). Error bars denote ± 3 standard deviations under posterior. **(g)** Generative samples from a standard SLDS differ substantially from the true states in (a) and are quite unstable. **(h)** In contrast, the rSLDS learns to generate state sequences that closely resemble those of the Lorenz attractor.

planes. Fig. 4c unrolls these dynamics over time, where the periodic nature and the discrete jumps become clear.

Rather than directly observing the Lorenz states, $x_{1:T}$, we simulate $N = 100$ dimensional discrete observations from a generalized linear model,

$$\rho_{t,n} = \sigma(c_n^\mathsf{T} x_t + d_n), \qquad y_{t,n} \sim \text{Bern}(\rho_{t,n}),$$

where $\sigma(\cdot)$ is the logistic function. A window of observations is shown in Fig. 4d. Just as we leveraged the Pólya-gamma augmentation to render the continuous latent states conjugate with the multinomial discrete state samples, we again leverage the augmentation scheme to render them conjugate with Bernoulli observations. As a further challenge, we also hold out a slice of data for $t \in [700, 900)$, identified by a gray mask in the center panels. We provide more details in the supplementary material.

Fitting an rSLDS via the same procedure described above, we find that the model separates these two planes into two distinct states, each with linear, rotational dynamics shown in Fig. 4b. Note that the latent states are only identifiable up to invertible transformation. Comparing Fig. 4e to panel c, we see that the rSLDS samples changes in discrete state at the points of large jumps in the data, but when the observations are masked, there is more uncertainty. This uncertainty in discrete state is propagated to uncertainty in the event probability, ρ, which is shown for the first output dimension in Fig. 4f. The times $\{t : y_{t,1} = 1\}$ are shown as dots, and the mean posterior probability $\mathbb{E}[\rho_{t,1}]$ is shown with ± 3 standard deviations. Again, even in the absence of observations, the model is capable of intelligently interpolating, inferring when there should be a discrete change in dynamics.

Finally, as expected, data generated from a standard SLDS fit in the same manner is quite different from the real data. It switches from one state to another independent of the x location, resulting in large divergences from the origin. In contrast, the rSLDS states are noisy yet qualitatively similar to those of the true Lorenz attractor.

5.3 Basketball Player Trajectories

We further illustrate our recurrent models with an application to the trajectories run by five National

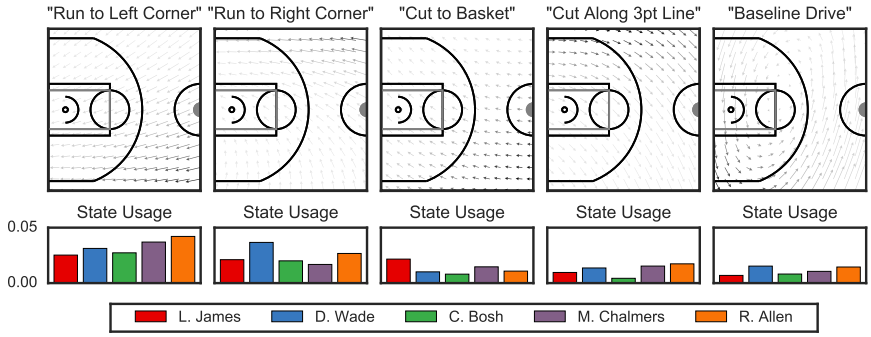


Figure 5: Exploratory analysis of NBA player trajectories from the Nov. 1, 2013 game between the Miami Heat and the Brooklyn Nets. **(Top)** When applied to trajectories of five Heat players, the recurrent AR-HMM (ro) discovers $K = 30$ discrete states with linear dynamics; five hand-picked states are shown here along with our names. Speed of motion is proportional to length of arrow. Location-dependent state probability is proportional to opacity of the arrow. **(Bottom)** The probability with which each player uses the corresponding state under the posterior.

Basketball Association (NBA) players from the Miami Heat in a game against the Brooklyn Nets on Nov. 1st, 2013. We are given trajectories, $y^{(p)}_{1:T_p} \in \mathbb{R}^{T_p \times 2}$, for each player p. We treat these trajectories as independent realizations of a "recurrence-only" AR-HMM with a shared set of $K = 30$ states. Positions are recorded every 40ms; combining the five players yields 256,103 time steps in total. We use our rAR-HMM to discover discrete dynamical states as well as the court locations in which those states are most likely to be deployed. We fit the model with 200 iteration of Gibbs sampling, initialized with a draw from the prior.

The dynamics of five of the discovered states are shown in Fig. 5 (top), along with the names we have assigned them. Below, we show the frequency with which each player uses the states under the posterior distribution. First, we notice lateral symmetry; some players drive to the left corner whereas others drive to the right. Anecdotally, Ray Allen is known to shoot more from the left corner, which agrees with the state usage here. The third state corresponds to a drive toward the hoop, which is most frequently used by LeBron James. Other states correspond to unique plays made by the players, like cuts along the three-point line and drives along the baseline. The complete set of states is shown in the supplementary material.

6 Discussion

Through a variety of experiments, we have demonstrated how the addition of recurrent connections from continuous states to future discrete states lends increased flexibility and interpretability to the classical switching linear dynamical system. This work is similar in spirit to the *piecewise affine* (PWA) framework in control systems [Sontag, 1981]. While Bayesian inference algorithms have been derived for the special case of the rAR-HMM [Juloski et al., 2005], the class of recurrent state space models has eluded a general Bayesian treatment. However, heuristic methods for learning PWA models mimic our initialization procedure [Paoletti et al., 2007].

The recurrent SLDS models strike a balance between flexibility and tractability. Composing linear systems and linear partitions achieves globally nonlinear dynamics while admitting efficient Bayesian inference algorithms. Directly modeling nonlinearities in the dynamics and transition models, e.g. via Gaussian processes or recurrent neural networks (RNNs), either necessitates more complex non-conjugate inference and learning algorithms [Frigola et al., 2013] or non-Bayesian, data-intensive gradient methods [Hochreiter and Schmidhuber, 1997, Graves, 2013] (though see Gal and Ghahramani [2015] for recent steps toward Bayesian learning for RNNs). Moreover, specifying a reasonable class of nonlinear dynamics models may be more challenging than for conditionally linear systems. Finally, while these augmentation schemes do not apply to nonlinear models, recent developments in structured variational autoencoders [Johnson et al., 2016] provide new tools to extend the rSLDS with nonlinear transition or emission models.

Recurrent state space models provide a flexible and interpretable class of models for many nonlinear time series. Our Bernoulli-Lorenz example is promising for other discrete domains, like multi-neuronal spike trains with globally nonlinear dynamics [e.g. Sussillo et al., 2016]. Likewise, beyond the realm of basketball, these

models naturally apply to modeling of consumer shopping behavior — e.g. how do environmental features influence shopper paths? — and could be extended to model social behavior in multiagent systems. These are exciting avenues for future work.

Acknowledgments

SWL is supported by the Simons Foundation SCGB-418011. ACM is supported by the Applied Mathematics Program within the Office of Science Advanced Scientific Computing Research of the U.S. Department of Energy under contract No. DE-AC02-05CH11231. RPA is supported by NSF IIS-1421780 and the Alfred P. Sloan Foundation. DMB is supported by NSF IIS-1247664, ONR N00014-11-1-0651, DARPA FA8750-14-2-0009, DARPA N66001-15-C-4032, Adobe, and the Sloan Foundation. LP is supported by the Simons Foundation SCGB-325171; DARPA N66001-15-C-4032; ONR N00014-16-1-2176; IARPA MICRONS D16PC00003.

References

Guy A Ackerson and King-Sun Fu. On state estimation in switching environments. *IEEE Transactions on Automatic Control*, 15(1):10–17, 1970.

Chaw-Bing Chang and Michael Athans. State estimation for discrete systems with switching parameters. *IEEE Transactions on Aerospace and Electronic Systems*, (3):418–425, 1978.

Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and computing*, 10(3):197–208, 2000.

Nicholas Foti, Jason Xu, Dillon Laird, and Emily Fox. Stochastic variational inference for hidden Markov models. In *Advances in Neural Information Processing Systems*, pages 3599–3607, 2014.

Emily Fox, Erik B Sudderth, Michael I Jordan, and Alan S Willsky. Nonparametric Bayesian learning of switching linear dynamical systems. *Advances in Neural Information Processing Systems*, pages 457–464, 2009.

Roger Frigola, Fredrik Lindsten, Thomas B Schön, and Carl Rasmussen. Bayesian inference and learning in Gaussian process state-space models with particle MCMC. In *Advances in Neural Information Processing Systems*, pages 3156–3164, 2013.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. *arXiv preprint arXiv:1512.05287*, 2015.

Zoubin Ghahramani and Geoffrey E Hinton. Switching state-space models. Technical report, University of Toronto, 1996.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

James D Hamilton. Analysis of time series subject to changes in regime. *Journal of econometrics*, 45(1): 39–70, 1990.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Matthew D Hoffman, David M Blei, Chong Wang, and John William Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1): 1303–1347, 2013.

Matthew J. Johnson and Alan S. Willsky. Stochastic variational inference for Bayesian time series models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1854–1862, 2014.

Matthew J Johnson, David Duvenaud, Alexander B Wiltschko, Sandeep R Datta, and Ryan P Adams. Composing graphical models with neural networks for structured representations and fast inference. *Advances in Neural Information Processing Systems*, 2016.

Aleksandar Lj Juloski, Siep Weiland, and WPMH Heemels. A Bayesian approach to identification of hybrid systems. *IEEE Transactions on Automatic Control*, 50(10):1520–1533, 2005.

Scott W Linderman, Matthew J Johnson, and Ryan P Adams. Dependent multinomial models made easy: Stick-breaking with the Pólya-gamma augmentation. In *Advances in Neural Information Processing Systems*, pages 3438–3446, 2015.

Kevin P Murphy. Switching Kalman filters. Technical report, Compaq Cambridge Research, 1998.

Simone Paoletti, Aleksandar Lj Juloski, Giancarlo Ferrari-Trecate, and René Vidal. Identification of hybrid systems a tutorial. *European Journal of Control*, 13(2):242–260, 2007.

Nicholas G Polson, James G Scott, and Jesse Windle. Bayesian inference for logistic models using Pólya-gamma latent variables. *Journal of the American Statistical Association*, 108(504):1339–1349, 2013.

Eduardo Sontag. Nonlinear regulation: The piecewise linear approach. *IEEE Transactions on Automatic Control*, 26(2):346–358, 1981.

David Sussillo, Rafal Jozefowicz, L. F. Abbott, and Chethan Pandarinath. LFADS: Latent factor analysis via dynamical systems. *arXiv preprint arXiv:1608.06315*, 2016.

Luke Tierney and Joseph B Kadane. Accurate approximations for posterior moments and marginal densities. *Journal of the American Statistical Association*, 81(393):82–86, 1986.

A Stochastic Variational Inference

The main paper introduces a Gibbs sampling algorithm for the recurrent SLDS and its siblings, but it is straightforward to derive a mean field variational inference algorithm as well. From this, we can immediately derive a stochastic variational inference (SVI) [Hoffman et al., 2013] algorithm for conditionally independent time series.

We use a structured mean field approximation on the augmented model,

$$p(z_{1:T}, x_{1:T}, \omega_{1:T}, \theta \,|\, y_{1:T}) \approx q(z_{1:T})\, q(x_{1:T})\, q(\omega_{1:T})\, q(\theta; \eta).$$

The first three factors will not be explicitly parameterized; rather, as with Gibbs sampling, we leverage standard message passing algorithms to compute the necessary expectations with respect to these factors. Moreover, $q(\omega_{1:T})$ further factorizes as,

$$q(\omega_{1:T}) = \prod_{t=1}^{T} \prod_{k=1}^{K-1} q(\omega_{t,k}).$$

To be concrete, we also expand the parameter factor,

$$q(\theta; \eta) = \prod_{k=1}^{K} q(R_k, r_k \,|\, \eta_{\mathsf{rec},k})\, q(A_k, b_k, B_k; \eta_{\mathsf{dyn},k}) \times q(C_k, d_k, D_k; \eta_{\mathsf{obs},k}).$$

The algorithm proceeds by alternating between optimizing $q(x_{1:T})$, $q(z_{1:T})$, $q(\omega_{1:T})$, and $q(\theta)$.

Updating $q(x_{1:T})$. Fixing the factor on the discrete states $q(z_{1:T})$, the optimal variational factor on the continuous states $q(x_{1:T})$ is determined by,

$$\ln q(x_{1:T}) = \ln \psi(x_1) + \sum_{t=1}^{T-1} \ln \psi(x_t, x_{t+1}) + \sum_{t=1}^{T-1} \ln \psi(x_t, z_{t+1}, \omega_t) + \sum_{t=1}^{T} \ln \psi(x_t; y_t) + c.$$

where

$$\psi(x_1) = \mathbb{E}_{q(\theta)q(z)} \ln p(x_1 \,|\, z_1, \theta) \tag{8}$$

$$\psi(x_t, x_{t+1}) = \mathbb{E}_{q(\theta)q(z)} \ln p(x_{t+1} \,|\, x_t, z_t, \theta), \tag{9}$$

$$\psi(x_t, z_{t+1}) = \mathbb{E}_{q(\theta)q(z)q(\omega)} \ln p(z_{t+1} \,|\, x_t, z_t, \omega_t, \theta),$$

Because the densities $p(x_1 \,|\, z_1, \theta)$ and $p(x_{t+1} \,|\, x_t, z_t, \theta)$ are Gaussian exponential families, the expectations in Eqs. (8)-(9) can be computed efficiently, yielding Gaussian potentials with natural parameters that depend on both $q(\theta)$ and $q(z_{1:T})$. Furthermore, each $\psi(x_t; y_t)$ is itself a Gaussian potential. As in the Gibbs sampler, the only non-Gaussian potential comes from the logistic stick breaking model, but once again, the Pólya-gamma augmentation scheme comes to the rescue. After augmentation, the potential as a function of x_t is,

$$\mathbb{E}_{q(\theta)q(z)q(\omega)} \ln p(z_{t+1} \,|\, x_t, z_t, \omega_t, \theta) = -\frac{1}{2} \nu_{t+1}^{\mathsf{T}} \Omega_t \nu_{t+1} + \nu_{t+1}^{\mathsf{T}} \kappa(z_{t+1}) + c.$$

Since $\nu_{t+1} = R_{z_t} x_t + r_{z_t}$ is linear in x_t, this is another Gaussian potential. As with the dynamics and observation potentials, the recurrence weights, (R_k, r_k), also have matrix normal factors, which are conjugate after augmentation. We also need access to $\mathbb{E}_q[\omega_{t,k}]$; we discuss this computation below.

After augmentation, the overall factor $q(x_{1:T})$ is a Gaussian linear dynamical system with natural parameters computed from the variational factor on the dynamical parameters $q(\theta)$, the variational parameter on the discrete states $q(z_{1:T})$, the recurrence potentials $\{\psi(x_t, z_t, z_{t+1})\}_{t=1}^{T-1}$, and the observation model potentials $\{\psi(x_t; y_t)\}_{t=1}^{T}$.

Because the optimal factor $q(x_{1:T})$ is a Gaussian linear dynamical system, we can use message passing to perform efficient inference. In particular, the expected sufficient statistics of $q(x_{1:T})$ needed for updating $q(z_{1:T})$ can be computed efficiently.

Updating $q(\omega_{1:T})$. We have,

$$\begin{aligned} \ln q(\omega_{t,k}) &= \mathbb{E}_q \ln p(z_{t+1} \,|\, \omega_t, x_t) + c \\ &= -\frac{1}{2} \mathbb{E}_q[\nu_{t+1}^2] \omega_{t,k} \\ &\quad + \mathbb{E}_{q(z_{1:T})} \ln p_{\mathrm{PG}}(\omega_{t,k} \,|\, \mathbb{I}[z_{t+1} \geq k], 0) + c \end{aligned}$$

While the expectation with respect to $q(z_{1:T})$ makes this challenging, we can approximate it with a sample, $\hat{z}_{1:T} \sim q(z_{1:T})$. Given a fixed value $\hat{z}_{1:T}$ we have,

$$q(\omega_{t,k}) = p_{\mathrm{PG}}(\omega_{t,k} \,|\, \mathbb{I}[\hat{z}_{t+1} \geq k], \mathbb{E}_q[\nu_{t+1}^2]).$$

The expected value of the distribution is available in closed form:

$$\mathbb{E}_q[\omega_{t,k}] = \frac{\mathbb{I}[\hat{z}_{t+1} \geq k]}{2\mathbb{E}_q[\nu_{t+1}^2]} \tanh\left(\tfrac{1}{2}\mathbb{E}_q[\nu_{t+1}^2]\right).$$

Updating $q(z_{1:T})$. Similarly, fixing $q(x_{1:T})$ the optimal factor $q(z_{1:T})$ is proportional to

$$\exp\left\{ \ln \psi(z_1) + \sum_{t=1}^{T-1} \ln \psi(z_t, x_t, z_{t+1}) + \sum_{t=1}^{T} \ln \psi(z_t) \right\},$$

where

$$\psi(z_1) = \mathbb{E}_{q(\theta)} \ln p(z_1 \mid \theta) + \mathbb{E}_{q(\theta)q(x)} \ln p(x_1 \mid z_1, \theta)$$
$$\psi(z_t, x_t, z_{t+1}) = \mathbb{E}_{q(\theta)q(x_{1:T})} \ln p(z_{t+1} \mid z_t, x_t)$$
$$\psi(z_t) = \mathbb{E}_{q(\theta)q(x)} \ln p(x_{t+1} \mid x_t, z_t, \theta)$$

The first and third densities are exponential families; these expectations can be computed efficiently. The challenge is the recurrence potential,

$$\psi(z_t, x_t, z_{t+1}) = \mathbb{E}_{q(\theta),q(x)} \ln \pi_{\mathsf{SB}}(\nu_{t+1}).$$

Since this is not available in closed form, we approximate this expectation with Monte Carlo over x_t, R_k, and r_k. The resulting factor $q(z_{1:T})$ is an HMM with natural parameters that are functions of $q(\theta)$ and $q(x_{1:T})$, and the expected sufficient statistics required for updating $q(x_{1:T})$ can be computed efficiently by message passing in the same manner.

Updating $q(\theta)$. To compute the expected sufficient statistics for the mean field update on η, we can also use message passing, this time in both factors $q(x_{1:T})$ and $q(z_{1:T})$ separately. The required expected sufficient statistics are of the form

$$\mathbb{E}_{q(z)}\mathbb{I}[z_t = i, z_{t+1} = j], \quad \mathbb{E}_{q(z)}\mathbb{I}[z_t = i],$$
$$\mathbb{E}_{q(z)}\mathbb{I}[z_t = k]\mathbb{E}_{q(x)}\left[x_t x_{t+1}^\mathsf{T}\right],$$
$$\mathbb{E}_{q(z)}\mathbb{I}[z_t = k]\mathbb{E}_{q(x)}\left[x_t x_t^\mathsf{T}\right], \quad \mathbb{E}_{q(z)}\mathbb{I}[z_1 = k]\mathbb{E}_{q(x)}[x_1],$$

where $\mathbb{I}[\,\cdot\,]$ denotes an indicator function. Each of these can be computed easily from the marginals $q(x_t, x_{t+1})$ and $q(z_t, z_{t+1})$ for $t = 1, 2, \ldots, T-1$, and these marginals can be computed in terms of the respective graphical model messages.

Given the conjugacy of the augmented model, the dynamics and observation factors will be MNIW distributions as well. These allow closed form expressions for the required expectations,

$$\mathbb{E}_q[A_k], \quad \mathbb{E}_q[b_k], \quad \mathbb{E}_q[A_k B_k^{-1}], \quad \mathbb{E}_q[b_k B_k^{-1}], \quad \mathbb{E}_q[B_k^{-1}],$$
$$\mathbb{E}_q[C_k], \quad \mathbb{E}_q[d_k], \quad \mathbb{E}_q[C_k D_k^{-1}], \quad \mathbb{E}_q[d_k D_k^{-1}], \quad \mathbb{E}_q[D_k^{-1}].$$

Likewise, the conjugate matrix normal prior on (R_k, r_k) provides access to

$$\mathbb{E}_q[R_k], \qquad \mathbb{E}_q[R_k R_k^\mathsf{T}], \qquad \mathbb{E}_q[r_k].$$

Stochastic Variational Inference. Given multiple, conditionally independent observations of time series, $\{y_{1:T_p}^{(p)}\}_{p=1}^P$ (using the same notation as in the basketball experiment), it is straightforward to derive a stochastic variational inference (SVI) algorithm [Hoffman et al., 2013]. In each iteration, we sample a random time series; run message passing to compute the optimal local factors, $q(z_{1:T_p}^{(p)})$, $q(x_{1:T_p}^{(p)})$, and $q(\omega_{1:T_p}^{(p)})$; and then use expectations with respect to these local factors as unbiased estimates of expectations with respect to the complete dataset when updating the global parameter factor, $q(\theta)$. Given a single, long time series, we can still derive efficient SVI algorithms that use subsets of the data, as long as we are willing to accept minor, controllable bias [Johnson and Willsky, 2014, Foti et al., 2014].

B Stick Breaking and Decision Lists

As mentioned in Section 4, one of the less desirable features of the logistic stick breaking regression model is its dependence on the ordering of the output dimensions; in our case, on the permutation of the discrete states $\{1, 2, \ldots, K\}$. To alleviate this issue, we first do a greedy search over permutations by fitting a decision list to $(x_t, z_t), z_{t+1}$ pairs. A decision list is an iterative classifier of the form,

$$z_{t+1} = \begin{cases} o_1 & \text{if } \mathbb{I}[p_1] \\ o_2 & \text{if } \mathbb{I}[\neg p_1 \wedge p_2] \\ o_3 & \text{if } \mathbb{I}[\neg p_1 \wedge \neg p_2 \wedge p_3] \\ \vdots & \\ o_K & \text{o.w.,} \end{cases}$$

where $(o_1, \ldots, o_K)$ is a permutation of $(1, \ldots, K)$, and $p_1, \ldots, p_k$ are predicates that depend on (x_t, z_t) and evaluate to true or false. In our case, these predicates are given by logistic functions,

$$p_j = \sigma(r_j^\mathsf{T} x_t) > 0.$$

We fit the decision list using a greedy approach: to determine o_1 and r_1, we use maximum a posterior estimation to fit logistic regressions for each of the K possible output values. For the k-th logistic regression, the inputs are $x_{1:T}$ and the outputs are $y_t = \mathbb{I}[z_{t+1} = k]$. We choose the best logistic regression (measured by log likelihood) as the first output. Then we remove those time points for which $z_{t+1} = o_1$ from the dataset and repeat, fitting $K - 1$ logistic regressions in order to determine the second output, o_2, and so on.

After iterating through all K outputs, we have a permutation of the discrete states. Moreover, the predicates $\{r_k\}_{k=1}^{K-1}$ serve as an initialization for the recurrence weights, R, in our model.

C Bernoulli-Lorenz Details

The Pólya-gamma augmentation makes it easy to handle discrete observations in the rSLDS, as illustrated in

the Bernoulli-Lorenz experiment. Since the Bernoulli likelihood is given by,

$$\begin{aligned} p(y_t \mid z_t, \theta) &= \prod_{n=1}^{N} \mathrm{Bern}(\sigma(c_n^\mathsf{T} x_t + d_n) \\ &= \prod_{n=1}^{N} \frac{(e^{c_n^\mathsf{T} x_t + d_n})^{y_{t,n}}}{1 + e^{c_n^\mathsf{T} x_t + d_n}}, \end{aligned}$$

we see that it matches the form of (7) with,

$$\nu_{t,n} = c_n^\mathsf{T} x_t + d_n, \quad b(y_{t,n}) = 1, \quad \kappa(y_{t,n}) = y_{t,n} - \frac{1}{2}.$$

Thus, we introduce an additional set of Pólya-gamma auxiliary variables,

$$\xi_{t,n} \sim \mathrm{PG}(1, 0),$$

to render the model conjugate. Given these auxiliary variables, the observation potential is proportional to a Gaussian distribution on x_t,

$$\psi(x_t, y_t) \propto \mathcal{N}(C x_t + d \mid \Xi_t^{-1} \kappa(y_t), \Xi_t^{-1}),$$

with

$$\begin{aligned} \Xi_t &= \mathrm{diag}\left([\xi_{t,1}, \ldots, \xi_{t,N}]\right), \\ \kappa(y_t) &= [\kappa(y_{t,1}), \ldots, \kappa(y_{t,N})]. \end{aligned}$$

Again, this admits efficient message passing inference for $x_{1:T}$. In order to update the auxiliary variables, we sample from their conditional distribution, $\xi_{t,n} \sim \mathrm{PG}(1, \nu_{t,n})$.

This augmentation scheme also works for binomial, negative binomial, and multinomial obsevations as well [Polson et al., 2013].

D Basketball Details

For completeness, Figures 6 and 7 show all $K = 30$ inferred states of the rAR-HMM (ro) for the basketball data.

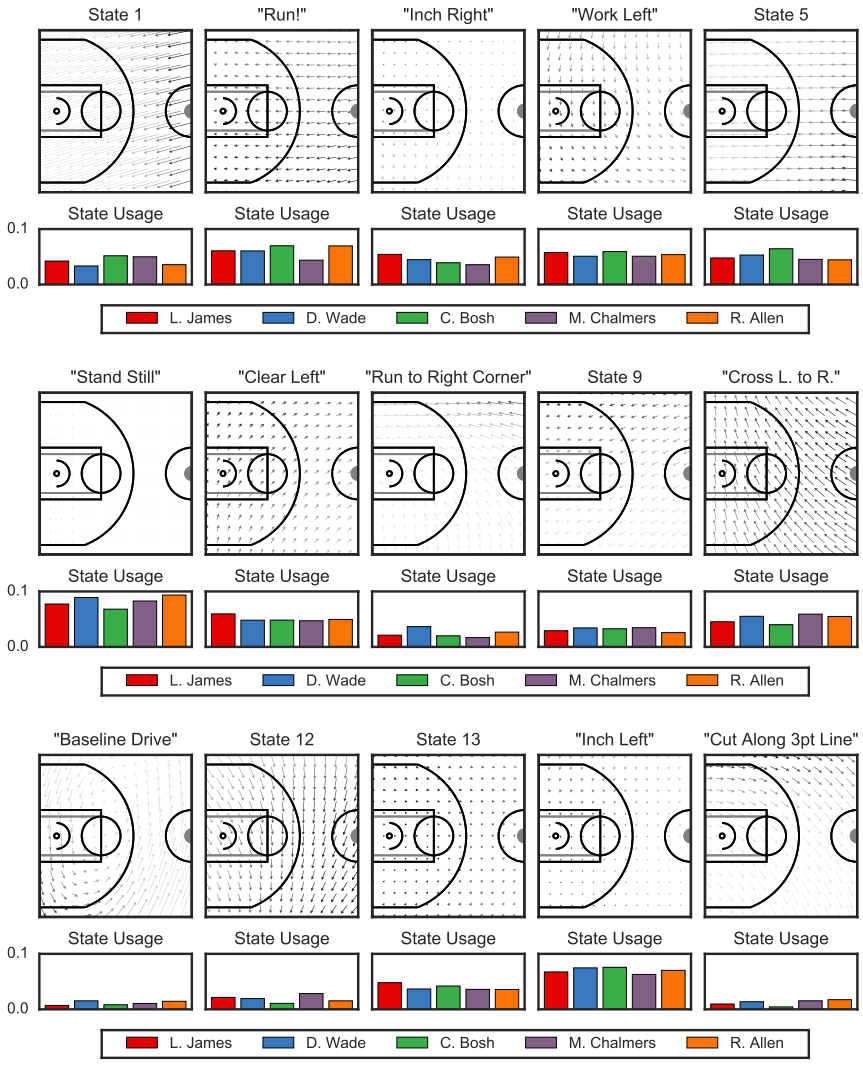


Figure 6: All of the inferred basketball states

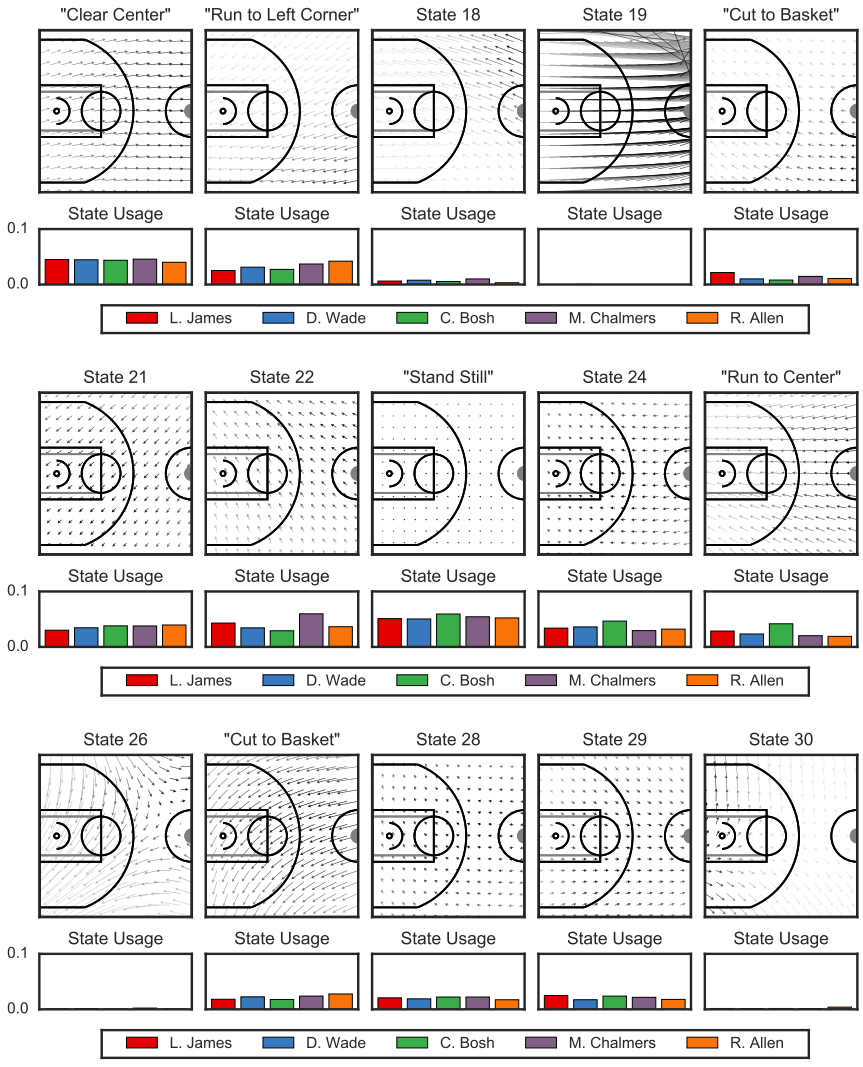


Figure 7: All of the inferred basketball states