Latent Stochastic Differential Equations

[Hegde et al., 2018]
Irregularly-timed datasets arise all the time

- Most patient data, gene assays irregularly sampled through time.
- All sorts of observation models (likelihoods), not just Gaussian
- Most large parametric models in ML are discrete time: RNNs, HMM, DMM
Project to discrete time?

- **Binning**: End up averaging many entries per bin, or leaving bins empty

- **Imputation**: Need to solve original problem, messes with uncertainty
Latent variable models

- Hidden Markov Models, Deep Markov Models

- specify $p(z), p(x | z)$

$$p(x) = \int p(x | z)p(z)dz$$

- Can integrate out $z$ however you want!

- Variational inference, MCMC

- A neural net or whatever can specify proposal or approx. posterior

- [Krishnan, Shalit, Sontag]

https://pyro.ai/examples/dmm.html
Latent variable models

- Can use a neural net to guess optimal variational params from data
- Structure of recognition net an implementation detail
- Only there to speed things up.
- Just needs to output a normalized distribution over $z$

[Image: https://pyro.ai/examples/dmm.html]
What about continuous time?
Ordinary Differential Equations

- Vector-valued $\mathbf{z}$ changes in time
- Time-derivative: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$
Ordinary Differential Equations

- Vector-valued $\mathbf{z}$ changes in time
- Time-derivative: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}(t_0)$, find:

$$\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta) \, dt$$
Ordinary Differential Equations

- Vector-valued $z$ changes in time.
- Time-derivative: $\frac{dz}{dt} = f(z(t), t)$
- Initial-value problem: given $z(t_0)$, find:

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)\,dt$$
Ordinary Differential Equations

- Vector-valued $\mathbf{z}$ changes in time
- Time-derivative: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}(t_0)$, find:

$$\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta) dt$$
Ordinary Differential Equations

- Vector-valued $\mathbf{z}$ changes in time
- Time-derivative: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}(t_0)$, find:

$$\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta) dt$$
Ordinary Differential Equations

- Vector-valued $z$ changes in time
- Time-derivative: $\frac{dz}{dt} = f(z(t), t)$
- Initial-value problem: given $z(t_0)$, find:
  \[ z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta) \, dt \]
- Euler approximates with small steps:
  \[ z(t + h) = z(t) + hf(z, t) \]
Ordinary Differential Equations

- Vector-valued $\mathbf{z}$ changes in time
- Time-derivative: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}(t_0)$, find:
  $$\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt$$
- Euler approximates with small steps:
  $$\mathbf{z}(t + h) = \mathbf{z}(t) + hf(\mathbf{z}, t)$$
Ordinary Differential Equations

- Vector-valued $z$ changes in time
- Time-derivative: $\frac{dz}{dt} = f(z(t), t)$
- Initial-value problem: given $z(t_0)$, find:
  \[ z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta) dt \]
- Euler approximates with small steps:
  \[ z(t + h) = z(t) + hf(z, t) \]
Ordinary Differential Equations

- Vector-valued \( \mathbf{z} \) changes in time
- Time-derivative: \( \frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t) \)
- Initial-value problem: given \( \mathbf{z}(t_0) \), find:

\[
\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta) \, dt
\]

- Euler approximates with small steps:

\[
\mathbf{z}(t + h) = \mathbf{z}(t) + hf(\mathbf{z}, t)
\]
Ordinary Differential Equations

- Vector-valued $\mathbf{z}$ changes in time
- Time-derivative: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}(t_0)$, find:
  \[ \mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta) dt \]
- Euler approximates with small steps:
  \[ \mathbf{z}(t + h) = \mathbf{z}(t) + hf(\mathbf{z}, t) \]
Ordinary Differential Equations

- Vector-valued $\mathbf{z}$ changes in time
- Time-derivative: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}(t_0)$, find:
  $$\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)\,dt$$
- Euler approximates with small steps:
  $$\mathbf{z}(t + h) = \mathbf{z}(t) + hf(\mathbf{z}, t)$$
Ordinary Differential Equations

- Vector-valued $\mathbf{z}$ changes in time
- Time-derivative: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}(t_0)$, find:

$$\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta) dt$$

- Euler approximates with small steps:

$$\mathbf{z}(t + h) = \mathbf{z}(t) + hf(\mathbf{z}, t)$$
Ordinary Differential Equations

- Vector-valued $\mathbf{z}$ changes in time
- Time-derivative: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}(t_0)$, find:
  \[ \mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)\,dt \]
- Euler approximates with small steps:
  \[ \mathbf{z}(t + h) = \mathbf{z}(t) + hf(\mathbf{z}, t) \]
Autoregressive continuous-time?

- ODE-RNN:
  - Between datapoints, $\frac{dh}{dt} = f(h(t))$
  - At observations, $h(t)' = g(x_t, h(t))$
  - $h$ represents belief state (like in an RNN)
  - Separates belief update due to time passing vs seeing data. Good!
• $z(t)$ represents true state of system at time $t$

• Need to approximate posterior $p(z_{t0} | x_{t1}...)$

• Well-defined state at all times, dynamics separate from inference

\[
\begin{align*}
 z_{t0} & \sim p(z_{t0}) \\
 z_{t1}, z_{t2}, \ldots, z_{tN} & = \text{ODESolve}(z_{t0}, f, \theta_f, t_0, \ldots, t_N) \\
 \text{each } x_{ti} & \sim p(x | z_{ti}, \theta_x)
\end{align*}
\]
An ODE latent-variable model

- Can do VAE-style inference with an RNN encoder
Mujoco: State versus Belief states

- States are more interpretable than belief states
- True dynamics are deterministic
Table 4: Test MSE (mean ± std) on PhysioNet. Autoregressive models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Interp ($\times 10^{-3}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN $\Delta_t$</td>
<td>3.520 ± 0.276</td>
</tr>
<tr>
<td>RNN-Impute</td>
<td>3.243 ± 0.275</td>
</tr>
<tr>
<td>RNN-Decay</td>
<td>3.215 ± 0.275</td>
</tr>
<tr>
<td>RNN GRU-D</td>
<td>3.384 ± 0.274</td>
</tr>
<tr>
<td>ODE-RNN (Ours)</td>
<td>2.361 ± 0.086</td>
</tr>
</tbody>
</table>

Table 5: Test MSE (mean ± std) on PhysioNet. Encoder-decoder models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Interp ($\times 10^{-3}$)</th>
<th>Extrap ($\times 10^{-3}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN-VAE</td>
<td>5.930 ± 0.249</td>
<td>3.055 ± 0.145</td>
</tr>
<tr>
<td>Latent ODE (RNN enc.)</td>
<td>3.907 ± 0.252</td>
<td>3.162 ± 0.052</td>
</tr>
<tr>
<td>Latent ODE (ODE enc)</td>
<td><strong>2.118 ± 0.271</strong></td>
<td><strong>2.231 ± 0.029</strong></td>
</tr>
<tr>
<td>Latent ODE + Poisson</td>
<td>2.789 ± 0.771</td>
<td>2.208 ± 0.050</td>
</tr>
</tbody>
</table>
Poisson Process Likelihoods

\[
\log p(t_1, \ldots, t_N | t_{\text{start}}, t_{\text{end}}) = \sum_{i=1}^{N} \log \lambda(z(t_i)) - \int_{t_{\text{start}}}^{t_{\text{end}}} \lambda(z(t)) dt
\]

- Model joint \( p(\text{obs, time}) \) instead of \( p(\text{obs} | \text{time}) \)
- Non-intervention model
Code available

• Latent ODEs for Irregularly-Sampled Time Series

• Yulia Rubanova, Ricky T. Q. Chen, David Duvenaud

• https://github.com/YuliaRubanova/latent_ode
Limitations of Latent ODEs

- Deterministic dynamics!
- State size grows with sequence length
- Special time $t_0$, only reason about $z_{t0}$
Let’s be like a Deep Markov Model

- Nonlinear latent variable with noise at each step:
  \[ z_{t+1} = z_t + f_\theta(z_t) + \epsilon \]

- Could add more steps between observations.

- Infinitesimal limit some sort of stochastic ODE…?

https://pyro.ai/examples/dmm.html
Stochastic Differential Equations

\[
\frac{dz}{dt} = f(z(t)) + \epsilon
\]

\[
dz = f(z(t)) dt + \sigma(z(t)) dB(t)
\]

- Implicit distribution over functions
Life is an SDE

• natural fit for many small, unobserved interactions:
  • motion of molecules in a liquid
  • allele frequencies in a gene pool
  • prices in a market
  • Interactions don’t need to be Gaussian; as long as CLT kicks in, you get Brownian motion

• Let’s put neural networks into SDE dynamics and fit giant SDE models to everything!

\[ dz = f_{\theta}(z(t))dt + \sigma_{\theta}(z(t))dB(t) \]
Related work 1

- Neural Ordinary Differential Equations
  - Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud

- Neural Stochastic Differential Equations: Deep Latent Gaussian Models in the Diffusion Limit
  - Belinda Tzen, Maxim Rabinshy

- Neural Stochastic Differential Equations
  - Stefano Peluchetti, Stefano Favaro

- Neural Jump Stochastic Differential Equations
  - Junteng Jia, Austin R. Benson
Related work 1

Related work 1


- Peluchetti + Favaro: Worked out SDE corresponding to infinitely-deep convnets with uncertain weights
Related work 1

• Tzen + Raginski: Deep LVMs become SDEs in the limit. Variational inf framework. Forward-mode autodiff.

• Peluchetti + Favaro: Worked out SDE corresponding to infinitely-deep convnets with uncertain weights

• Jia + Benson: Added countably many discrete jumps to latent ODEs
Related work 2
Related work 2

Related work 2


Related work 2


• Markus Heinonen, Cagatay Yildiz, Henrik Mannerström, Jukka Intosalmi, and Harri Lähdesmäki. *Learning unknown ODE models with gaussian processes.*
Related work 2


Related work 2


- All use Euler discretizations. Not clear what limiting algorithm is (e.g. enforces invariants?), and not memory-efficient.
Related work 2


- All use Euler discretizations. Not clear what limiting algorithm is (e.g. enforces invariants?), and not memory-efficient.

- Not even going to discuss methods that require solving a PDE - not scalable.
Related work 2


- Markus Heinonen, Cagatay Yildiz, Henrik Mannerström, Jukka Intosalmi, and Harri Lähdesmäki. Learning unknown ODE models with gaussian processes.


- All use Euler discretizations. Not clear what limiting algorithm is (e.g. enforces invariants?), and not memory-efficient.

- Not even going to discuss methods that require solving a PDE - not scalable.

- We want to use adaptive, (high-order?) SDE solvers.
How to fit ODE params?

\[ L(\theta) = L \left( \int_{t_0}^{t_1} f(z(t), t, \theta) dt \right) \]

\[ \frac{\partial L}{\partial \theta} = ? \]
How to fit ODE params?

\[ L(\theta) = L \left( \int_{t_0}^{t_1} f(z(t), t, \theta) dt \right) \]

\[ \frac{\partial L}{\partial \theta} = ? \]

- Don’t backprop through solver: High memory cost, extra numerical error
- Alexey Radul: Approximate the derivative, don’t differentiate the approximation!
Continuous-time Backpropagation

Standard Backprop:

\[
\frac{\partial L}{\partial z_t} = \frac{\partial L}{\partial z_{t+1}} \frac{\partial f(z_t, \theta)}{\partial z_t}
\]

\[
\frac{\partial L}{\partial \theta} = \sum_t \frac{\partial L}{\partial z_t} \frac{\partial f(z_t, \theta)}{\partial \theta}
\]
Continuous-time Backpropagation

Standard Backprop:

\[
\frac{\partial L}{\partial z_t} = \frac{\partial L}{\partial z_{t+1}} \frac{\partial f(z_t, \theta)}{\partial z_t}
\]

\[
\frac{\partial L}{\partial \theta} = \sum_t \frac{\partial L}{\partial z_t} \frac{\partial f(z_t, \theta)}{\partial \theta}
\]

Adjoint sensitivities: (Pontryagin et al., 1962):

\[
\frac{\partial}{\partial t} \frac{\partial L}{\partial z(t)} = \frac{\partial L}{\partial z(t)} \frac{\partial f(z(t), \theta)}{\partial z} \frac{\partial}{\partial \theta}
\]

\[
\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} \frac{\partial L}{\partial z(t)} \frac{\partial f(z(t), \theta)}{\partial \theta} dt
\]
Continuous-time Backpropagation

Adjoint sensitivities:
(Pontryagin et al., 1962):

$$\frac{\partial}{\partial t} \frac{\partial L}{\partial z(t)} = \frac{\partial L}{\partial z(t)} \frac{\partial f(z(t), \theta)}{\partial z}$$

$$\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} \frac{\partial L}{\partial z(t)} \frac{\partial f(z(t), \theta)}{\partial \theta} dt$$
Continuous-time Backpropagation

- Can build adjoint dynamics with autodiff, compute gradients with second ODE solve:

\[
\frac{\partial}{\partial t} \frac{\partial L}{\partial z(t)} = \frac{\partial L}{\partial z(t)} \frac{\partial f(z(t), \theta)}{\partial z} \\
\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} \frac{\partial L}{\partial z(t)} \frac{\partial f(z(t), \theta)}{\partial \theta} \, dt
\]

Adjoint sensitivities:
(Pontryagin et al., 1962):
Continuous-time Backpropagation

• Can build adjoint dynamics with autodiff, compute gradients with second ODE solve:

```python
def f_aug([z, a, d], t):
    return [f, -a*df/dz, -a*df/dθ]
```

Adjoint sensitivities: (Pontryagin et al., 1962):

\[
\frac{\partial}{\partial t} \frac{\partial L}{\partial z(t)} = \frac{\partial L}{\partial z(t)} \frac{\partial f(z(t), \theta)}{\partial z} \\
\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} \frac{\partial L}{\partial z(t)} \frac{\partial f(z(t), \theta)}{\partial \theta} dt
\]
Continuous-time Backpropagation

- Can build adjoint dynamics with autodiff, compute gradients with second ODE solve:

```python
def f_aug([z, a, d], t):
    return [f, -a*df/dz, -a*df/dθ]

[z0, dL/dz(t0), dL/dθ] =
ODESolve(f_aug,
[z(t1), dL/dz(t1), 0], t1, t0)
```

Adjoint sensitivities: (Pontryagin et al., 1962):

\[
\frac{\partial}{\partial t} \frac{\partial L}{\partial z(t)} = \frac{\partial L}{\partial z(t)} \frac{\partial f(z(t), \theta)}{\partial z}
\]

\[
\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} \frac{\partial L}{\partial z(t)} \frac{\partial f(z(t), \theta)}{\partial \theta} dt
\]
O(1) Memory Gradients

- No need to store activations, just run dynamics backwards from output.
- Easy to run ODE backwards, just run negate dynamics and time:
  - $\text{back}_f(z, t) = -f(z, -t)$
Why not repeat same trick?

• If an SDE is just “an ODE with noise”, why not apply same adjoint method?
Need to store noise

- Reparameterization trick: Use same noise from forward pass on reverse pass
- Infinite reparameterization trick: Use same Brownian motion sample on forward and reverse passes.
Brownian Tree

- Can ‘zoom in’ arbitrarily close at any point
- splittable random seed ensures all entire sample is consistent
type UnitInterval = Real

bmIter :: (Key, Real, Real, UnitInterval) -> (Key, Real, Real, UnitInterval)
bmIter (key, y, sigma, t) =
  (kDraw, kL, kR) = splitKey3 key
  t' = abs (t - 0.5)
  y' = sigma * randn kDraw * (0.5 - t')
  key' = select (t > 0.5) kL kR
  (key', y + y', sigma / sqrt 2.0, t' * 2.0)

sampleBM :: Key -> UnitInterval -> Real
sampleBM key t =
  (_, y, _, _) = fold (key, 0.0, 1.0, t) for i::10. bmIter y
What is “running an SDE backwards”?

\[ dz = -f(z(-t))dt + \sigma(z(-t))dB(-t) \]
What is “running an SDE backwards”? 

• Me: Let’s just slap negative signs on everything and hope for the best

\[ dz = -f(z(-t))dt + \sigma(z(-t))dB(-t) \]
What is “running an SDE backwards”?

- Me: Let’s just slap negative signs on everything and hope for the best
- Xuechen and Leonard: What does that even mean? Much later: Actually we proved that’s correct.

\[ dz = -f(z(-t))dt + \sigma(z(-t))dB(-t) \]
What is “running an SDE backwards”?

- Me: Let’s just slap negative signs on everything and hope for the best.

- Xuechen and Leonard: What does that even mean? Much later: Actually we proved that’s correct.

- Builds on results from Kunita 2019.

\[ dz = -f(z(-t))dt + \sigma(z(-t))dB(-t) \]
What is “running an SDE backwards”?

- Me: Let’s just slap negative signs on everything and hope for the best

- Xuechen and Leonard: What does that even mean? Much later: Actually we proved that’s correct.

- Builds on results from Kunita 2019.

- Adjoint formula is analogous to ODE.

\[ dz = -f(z(-t))\, dt + \sigma(z(-t))\, dB(-t) \]
Generalize adjoint to diffusion func

\[ \tilde{A}_{s,t}(z) = \nabla \mathcal{L}(z) + \int_s^t \nabla b(\tilde{\Psi}_{r,t}(z), r) ^T \tilde{A}_{r,t}(z) \, dr + \sum_{i=1}^m \int_s^t \nabla \sigma_i(\tilde{\Psi}_{r,t}(r), u) ^T \tilde{A}_{r,t}(z) \circ dW_r^i, \]

- Dynamics already known
- Diffusion adjoint almost the same as drift adjoint.
- Just more vector-Jacobian products!

```python
def drift_adjoint(augmented_state, t, flat_args):
    y, y_adj, args_adj = unpack(augmented_state)
    fval, vjp_all = vjp(flat_f, y, t, flat_args)
    f_vjp_a, f_vjp_t, f_vjp_args = vjp_all(-y_adj)
    return np.concatenate([[fval, f_vjp_a, f_vjp_args]])

def diffusion_adjoint_prod(augmented_state, t, args, v):
    y, y_adj, arg_adj = unpack(augmented_state)
    gval, vjp_g = vjp(flat_g, y, t, args)
    vjp_a, vjp_t, vjp_args = vjp_g(-y_adj * v)
    return np.concatenate([[gval * v, vjp_a, vjp_args]])
```
Time complexity (fixed-step)

<table>
<thead>
<tr>
<th>Method</th>
<th>Memory</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward pathwise [14, 60]</td>
<td>(O(1))</td>
<td>(O(LD))</td>
</tr>
<tr>
<td>Backprop through solver [11]</td>
<td>(O(L))</td>
<td>(O(L))</td>
</tr>
<tr>
<td>Stochastic adjoint (ours)</td>
<td>(O(1))</td>
<td>(O(L \log L))</td>
</tr>
</tbody>
</table>

- Just solving forwards costs \(O(L)\) time.
- Time more like \(O(L)\) when dynamics are expensive.
- Okay! Now we can fit SDE paths to data…
Need Latent (Bayesian) SDE

• Can’t just fit SDE to maximize likelihood - optimal solution has no randomness and just hugs data

• Define prior and approx. posterior implicitly:

\[ dz_p = f_\theta(z(t))dt + \sigma_\theta(z(t))dB(t) \]
Variational inference

- Define prior and approx. posterior implicitly:

\[ dz_p = f_\theta(z(t))dt + \sigma_\theta(z(t))dB(t) \]
\[ dz_q = f_\phi(z(t))dt + \sigma_\theta(z(t))dB'(t) \]

\[ u(t) = \left\| \frac{f_\theta(z(t)) - f_\phi(z(t))}{\sigma_\theta(z(t))} \right\|^2 \]

\[ \mathcal{L}_{VI} = \mathbb{E} \left[ \frac{1}{2} \int_0^T |u(Z_t, t)|^2 \, dt - \sum_{i=1}^N \log p(y_{t_i} | z_{t_i}) \right] \]
Latent SDEs: An unexplored model class

- Define implicit prior over functions with neural nets for dynamics (like GP hyperparams)
- Define implicit approximate posterior through neural nets for dynamics (variational params)
- Define observation likelihoods. Anything differentiable wrt latent state will work (e.g. text models!)
- Train everything jointly with ADVI. Should scale to millions of params. Can use Milstein solver for diagonal noise.
Early latent SDE results

- OU prior, Laplace likelihood
Early latent SDE results

- OU prior,
  Laplace likelihood
Early latent SDE results

- OU prior, Laplace likelihood
- Inference problem is more local in time than for ODE (recognition net can steer posterior sample towards data)
Early latent SDE results

- OU prior, Laplace likelihood
- Inference problem is more local in time than for ODE (recognition net can steer posterior sample towards data)
Early latent SDE results: Mocap

- 50D data, 6D latent space, sharing dynamics and recognition params across time series (11000 params)

<table>
<thead>
<tr>
<th>Method</th>
<th>Test MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>npODE [18]</td>
<td>22.96†</td>
</tr>
<tr>
<td>NeuralODE [4]</td>
<td>22.49 ± 0.88†</td>
</tr>
<tr>
<td>ODE$^2$VAE [61]</td>
<td>10.06 ± 1.4†</td>
</tr>
<tr>
<td>ODE$^2$VAE-KL [61]</td>
<td>8.09 ± 1.95†</td>
</tr>
<tr>
<td>Latent ODE [4, 50]</td>
<td>5.98 ± 0.28</td>
</tr>
<tr>
<td>Latent SDE (this work)</td>
<td><strong>4.03 ± 0.20</strong></td>
</tr>
</tbody>
</table>
Early latent SDE results: Mocap

- 50D data, 6D latent space, sharing dynamics and recognition params across time series (11000 params)

<table>
<thead>
<tr>
<th>Method</th>
<th>Test MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>npODE [18]</td>
<td>22.96†</td>
</tr>
<tr>
<td>NeuralODE [4]</td>
<td>22.49 ± 0.88†</td>
</tr>
<tr>
<td>ODE$^2$VAE [61]</td>
<td>10.06 ± 1.4†</td>
</tr>
<tr>
<td>ODE$^2$VAE-KL [61]</td>
<td>8.09 ± 1.95†</td>
</tr>
<tr>
<td>Latent ODE [4, 50]</td>
<td>5.98 ± 0.28</td>
</tr>
<tr>
<td>Latent SDE (this work)</td>
<td><strong>4.03 ± 0.20</strong></td>
</tr>
</tbody>
</table>
Limitations

- SDE solvers much lower-order convergence than ODE solvers
  - (e.g. Milstein order 1 vs RK4)
- Non-diagonal noise requires Levy areas
- Diagonal noise requires funny parameterization
- Need jump-style noise? (e.g. hit by a car)
- Not scalable in input dimension (diffusions)?
SDEs vs GPs

- Distinct sets of priors over functions
- Easy to construct non-Gaussian SDE

From “Handbook of Statistics”, Mubayi et al, 2019. Line means "can be used to construct", but not "contains"
Future work:

• Skipping short-scale dynamics between observations (mixes back to prior)

• Infinitely deep Bayesian neural networks

• Code in a week or two (prototype)
Thanks!

Xuechen Li, Leonard Wong, Ricky Chen, Yulia Rubanova, David Duvenaud