Infinitely Deep Bayesian Neural Networks

Winnie Xu

Xuechen Li

Ricky T.Q. Chen

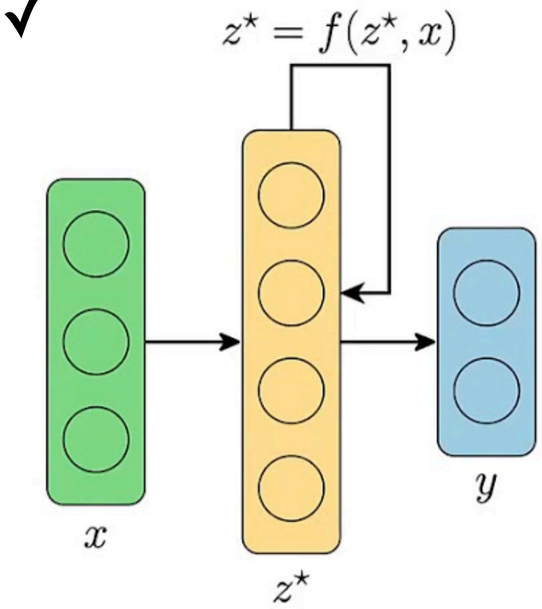
VECTOR INSTITUTE

What is the ultimate architecture?

- Specify hyperparameters indirectly through utility funcs
  - E.g. speed vs accuracy tradeoffs
  - Decouple compute from number of params
  - O(1) memory training
- Predictions that include model uncertainty
  - Arbitrarily expressive approx. posterior
- Low-variance gradients

Do DEQs Dominate ODE-nets?

- Specify hyperparameters indirectly through utility funcs ✓
  - E.g. speed vs accuracy tradeoffs ✓
  - Decouple compute from # params ✓
  - O(1) memory training ✓
- Predictions that include model uncertainty (Bayesian Deqs) ?
  - Arbitrarily expressive approx. posterior ?
  - Low-variance gradients ?



Outline

- Can now fit large latent SDEs
- Infinitely deep Bayesian neural nets
  - O(1) memory training
  - Arbitrarily expressive approx. posterior
  - Tunable speed vs precision
  - Asymptotically zero-variance gradients

Building an infinitely-deep BNN

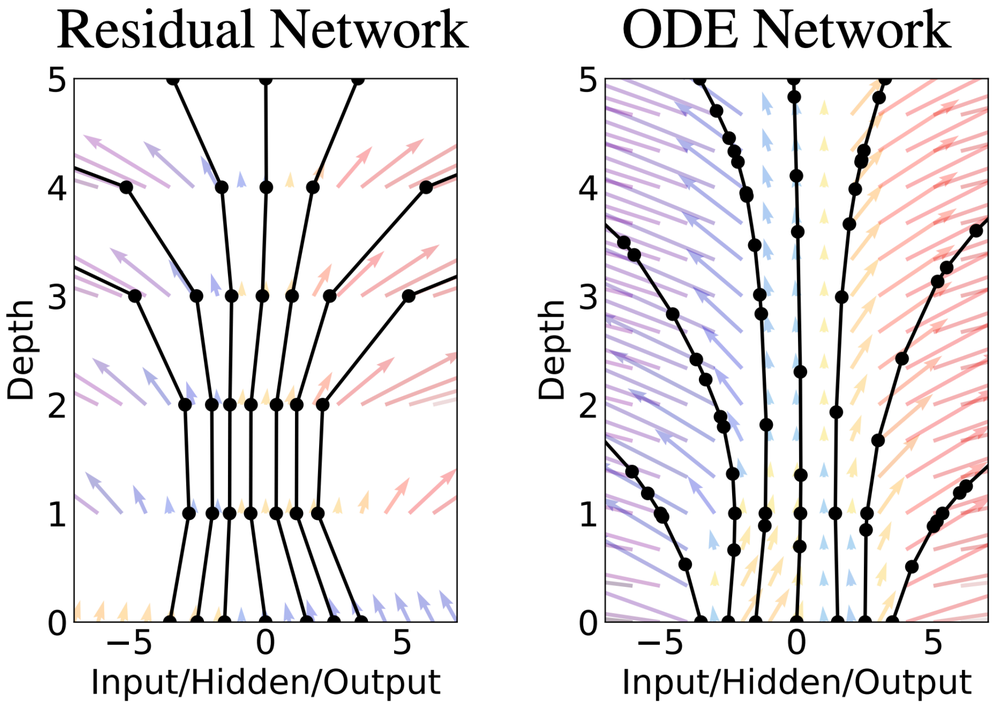
- Start with a ResNet:

$$h_{t+\epsilon} = h_t + \epsilon f_h(h_t, w_t)$$

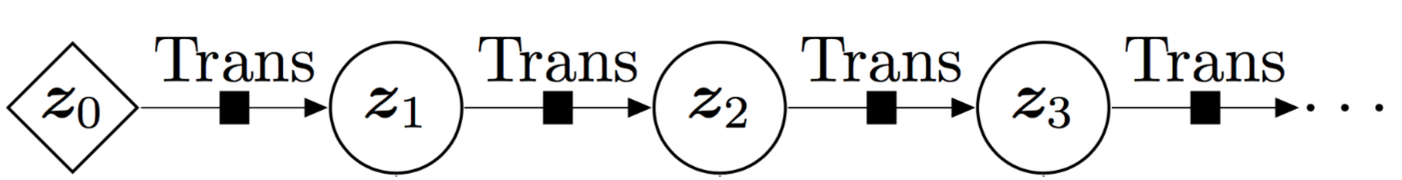
- Take limit as eps -> 0, number of layers grows.

- Given a process over weights w_t, activations h follow a random ODE:

$$dh_t = f_h(h_t, w_t)$$



Why SDEs?



$$z_{t+1} = z_t + f_\theta(z_t) + \epsilon$$

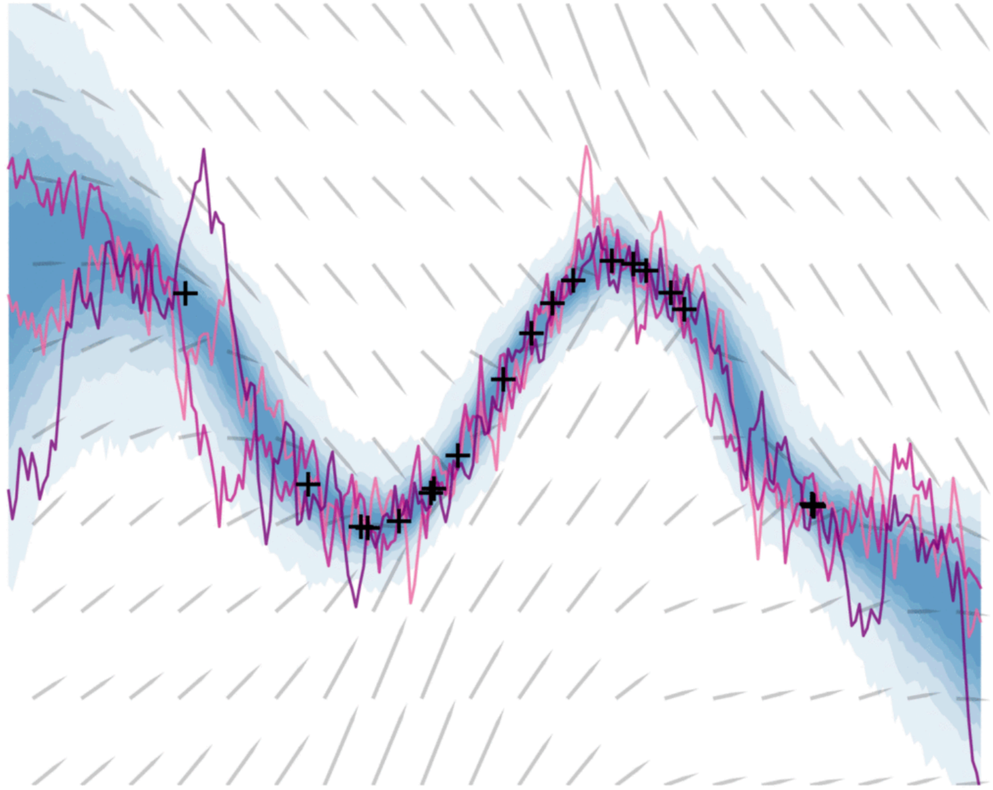
- Distribution due to uncertainty about weights
- Infinitesimal limit some sort of stochastic ODE…?

Stochastic Differential Equations

$$\frac{dz}{dt} = f(z(t)) + \text{“}\epsilon\text{”}$$

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

Drift Diffusion



- Implicit distribution over functions.

How to fit ODE params?

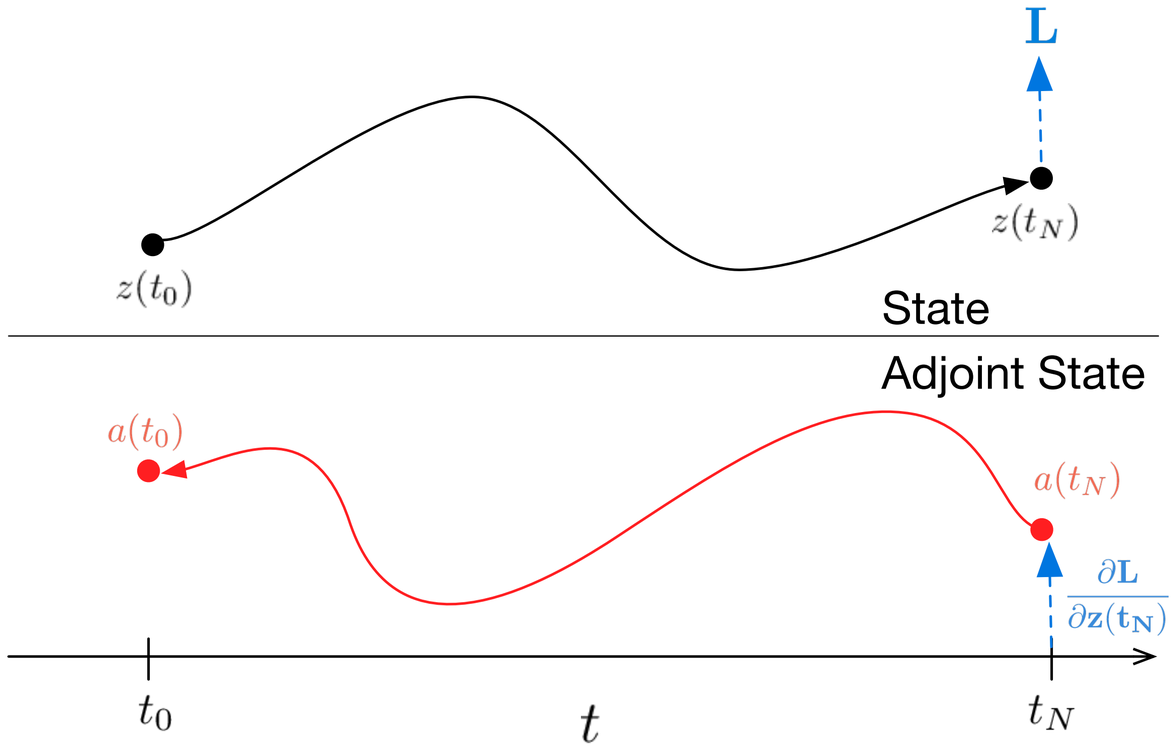
$$L(\theta) = L\left(\int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta) dt\right)$$

$$\frac{\partial L}{\partial \theta} = ?$$

- Backprop through solver has high memory cost

O(1) Memory Gradients

- To reconstruct trajectory, just run dynamics backwards from output:
  - back_f(z, t) = -f(z, -t)

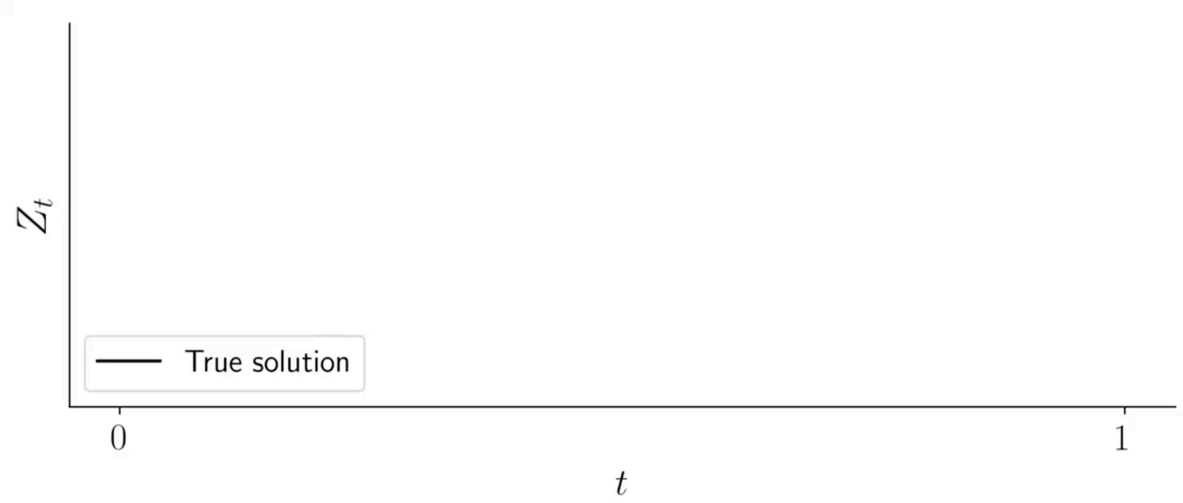


Why not repeat same trick?

- "Unfortunately, there is no straightforward way to port this construction to SDEs." - Tzen & Raginsky (2019)

What is “running an SDE backwards”?

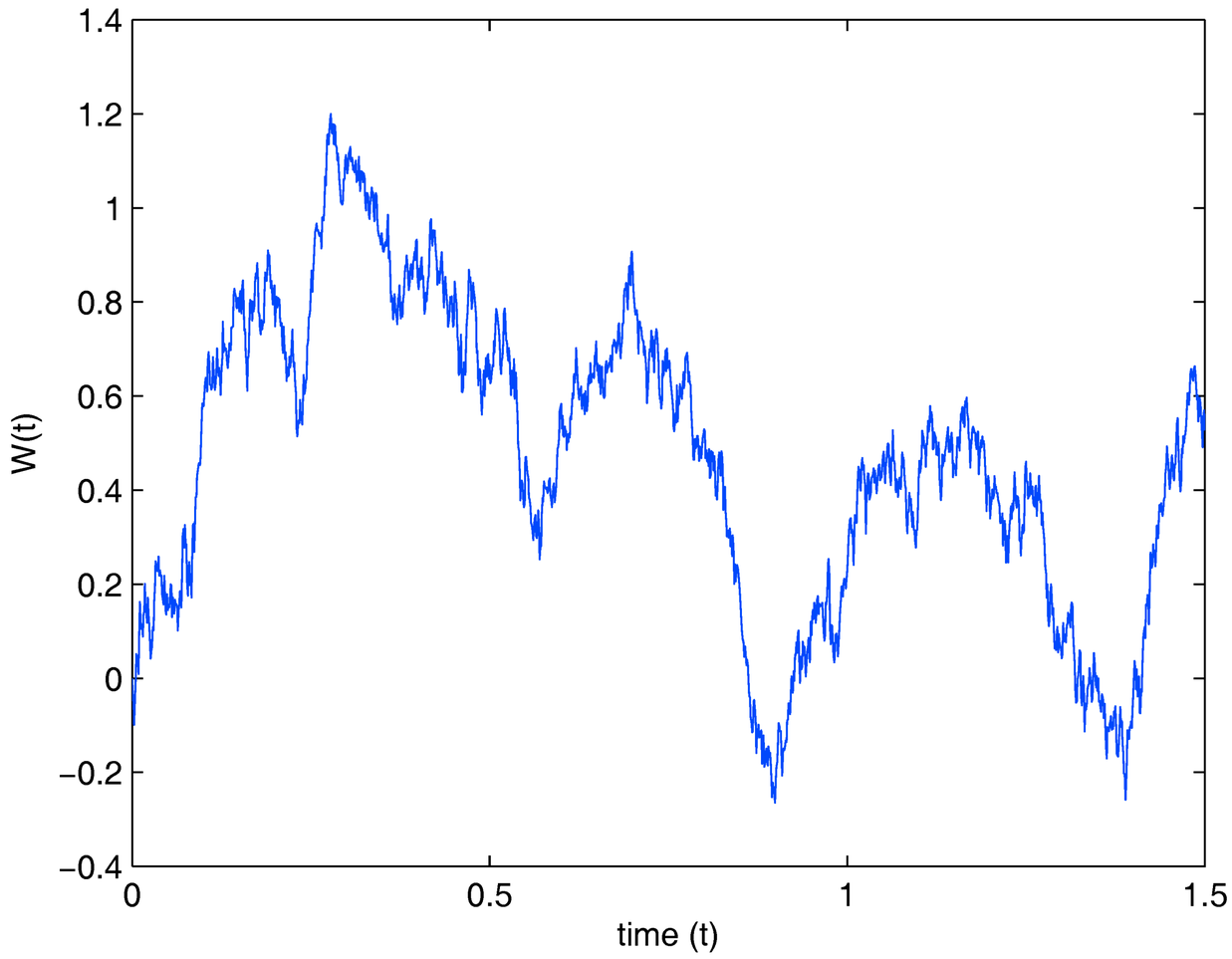
- Me: Let’s just slap negative signs on everything and hope for the best
- Xuechen and Leonard: What does that even mean? Later: that’s correct.
- Builds on Kunita (2019)



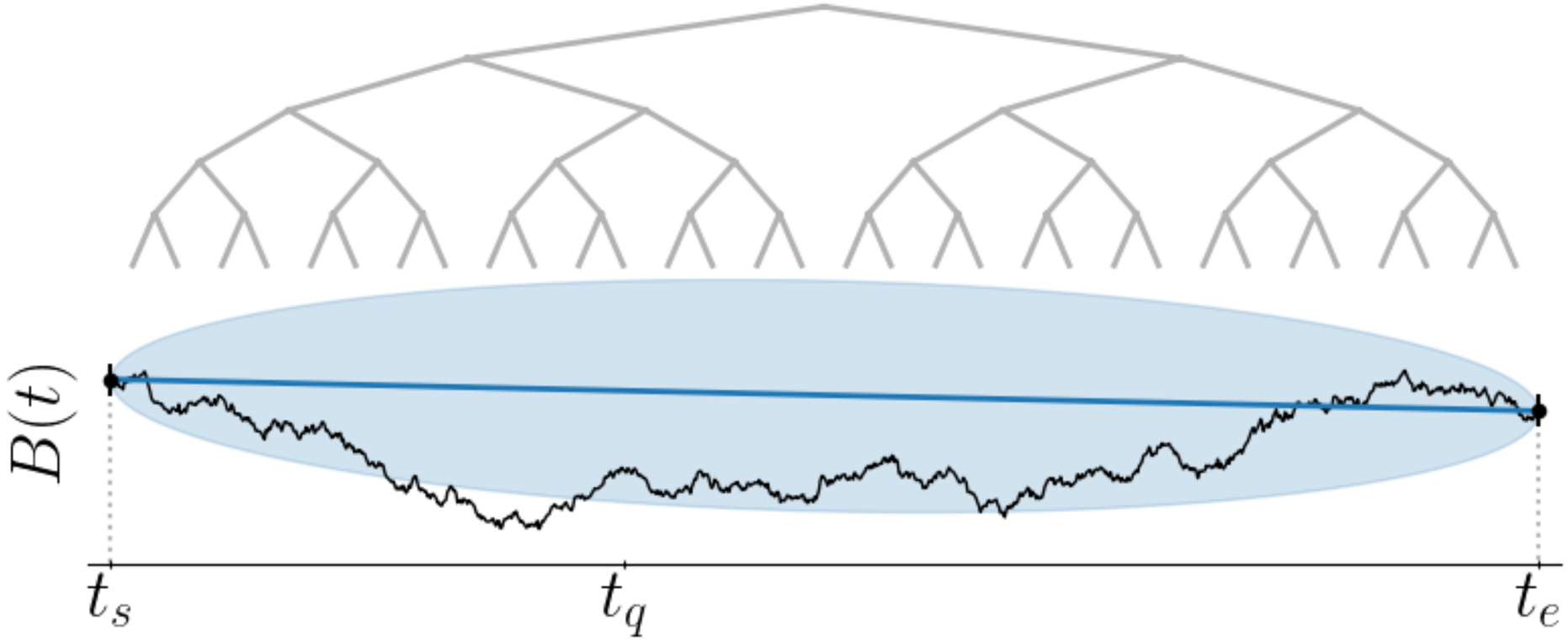
$$dz = -f(z(-t))dt + \sigma(z(-t))dB(-t)$$

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.
- Need to sample entire function



Virtual Brownian Tree



- Can ‘zoom in’ arbitrarily close at any point.
  - O(1) memory, O(log(1/eps)) time
- splittable random seed ensures all entire sample is consistent

Algorithm 1 ODE Adjoint Sensitivity

Input: Parameters θ, start time t_0, stop time t_1, final state z_{t_1}, loss gradient $\partial \mathcal{L}/z_{t_1}$, dynamics $f(z, t, \theta)$.

def $\overline{f}([z_t, a_t, \cdot],\ t,\ \theta)$: ▷ Augmented dynamics
  $v = f(z_t, -t, \theta)$
  return $[-v,\ a_t \partial v/\partial z,\ a_t \partial v/\partial \theta\,]$

$$\begin{bmatrix} z_{t_0} \\ \partial \mathcal{L}/\partial z_{t_0} \\ \partial \mathcal{L}/\partial \theta \end{bmatrix} = \texttt{odeint}\left(\begin{bmatrix} z_{t_1} \\ \partial \mathcal{L}/\partial z_{t_1} \\ \mathbf{0}_p \end{bmatrix}, \overline{f}, -t_1, -t_0 \right)$$

return $\partial \mathcal{L}/\partial z_{t_0}, \partial \mathcal{L}/\partial \theta$

Algorithm 1 ODE Adjoint Sensitivity

Input: Parameters θ, start time t_0, stop time t_1, final state z_{t_1}, loss gradient $\partial\mathcal{L}/z_{t_1}$, dynamics $f(z, t, \theta)$.

def $\overline{f}([z_t, a_t, \cdot],\ t,\ \theta)$: ▷ Augmented dynamics
  $v = f(z_t, -t, \theta)$
  return $[-v,\ a_t \partial v/\partial z,\ a_t \partial v/\partial \theta\,]$

$$\begin{bmatrix} z_{t_0} \\ \partial\mathcal{L}/\partial z_{t_0} \\ \partial\mathcal{L}/\partial\theta \end{bmatrix} = \texttt{odeint}\left(\begin{bmatrix} z_{t_1} \\ \partial\mathcal{L}/\partial z_{t_1} \\ \mathbf{0}_p \end{bmatrix}, \overline{f}, -t_1, -t_0 \right)$$

return $\partial\mathcal{L}/\partial z_{t_0}, \partial\mathcal{L}/\partial\theta$

Algorithm 2 SDE Adjoint Sensitivity (Ours)

Input: Parameters θ, start time t_0, stop time t_1, final state z_{t_1}, loss gradient $\partial\mathcal{L}/z_{t_1}$, drift $f(z, t, \theta)$, diffusion $\sigma(z, t, \theta)$, Wiener process sample $w(t)$.

def $\overline{f}([z_t, a_t, \cdot],\ t,\ \theta)$: ▷ Augmented drift
  $v = f(z_t, -t, \theta)$
  return $[-v,\ a_t \partial v/\partial z,\ a_t \partial v/\partial \theta]$

def $\overline{\sigma}([z_t, a_t, \cdot],\ t,\ \theta)$: ▷ Augmented diffusion
  $v = \sigma(z_t, -t, \theta)$
  return $[-v, a_t \partial v/\partial z, a_t \partial v/\partial \theta]$

def $\overline{w}(t)$: ▷ Replicated noise
  return $[-w(-t), -w(-t), -w(-t)]$

$$\begin{bmatrix} z_{t_0} \\ \partial\mathcal{L}/\partial z_{t_0} \\ \partial\mathcal{L}/\partial\theta \end{bmatrix} = \texttt{sdeint}\left(\begin{bmatrix} z_{t_1} \\ \partial\mathcal{L}/\partial z_{t_1} \\ \mathbf{0}_p \end{bmatrix}, \overline{f}, \overline{\sigma}, \overline{w}, -t_1, -t_0 \right)$$

return $\partial\mathcal{L}/\partial z_{t_0}, \partial\mathcal{L}/\partial\theta$

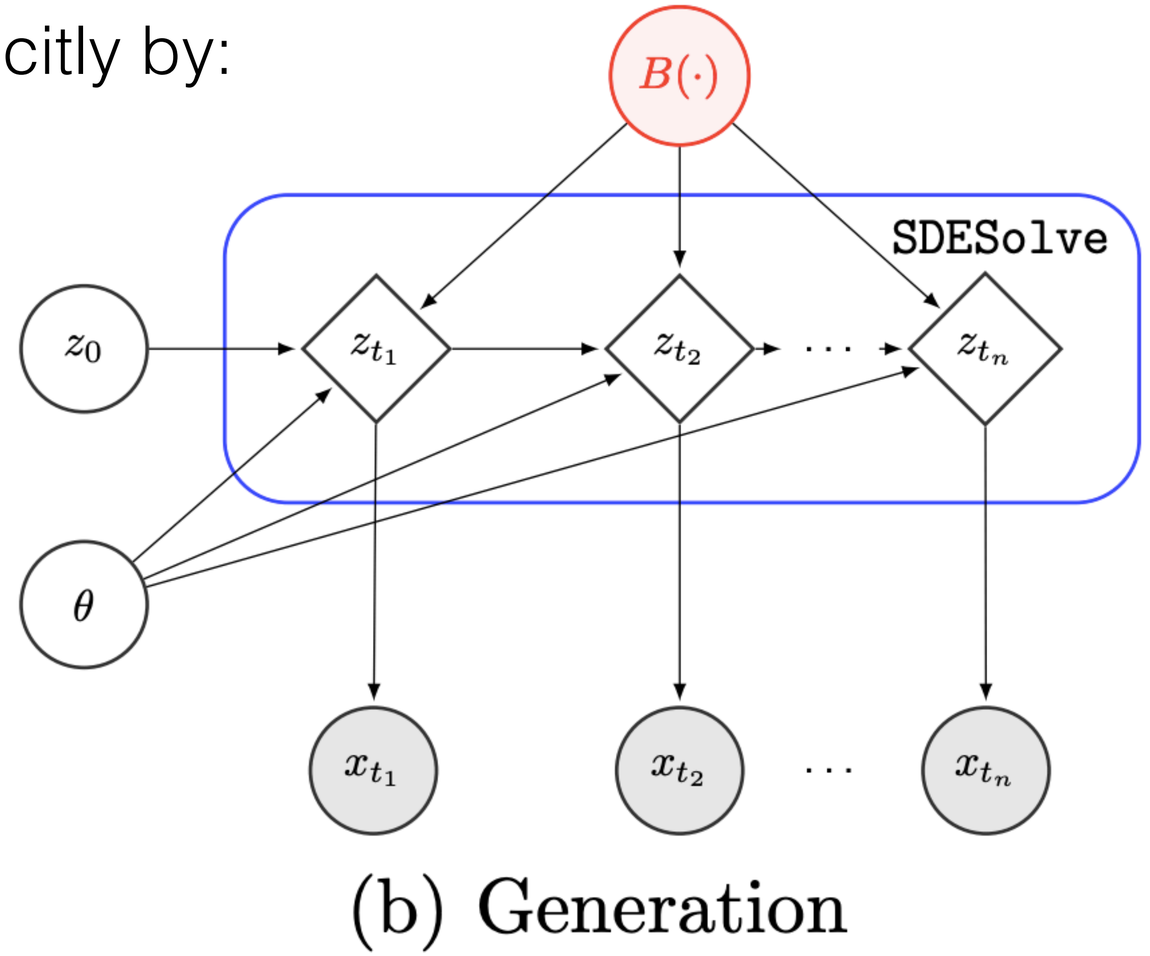
Time and memory cost

Method	Memory	Time
Forward pathwise [14, 60]	$\mathcal{O}(1)$	$\mathcal{O}(LD)$
Backprop through solver [11]	$\mathcal{O}(L)$	$\mathcal{O}(L)$
Stochastic adjoint (ours)	$\mathcal{O}(1)$	$\mathcal{O}(L \log L)$

- Can now fit large SDE models by gradient descent!

Latent SDE Model

- Generative model (decoder) defined implicitly by:
  - an SDE $dz_p = f_\theta(z(t))dt + \sigma_\theta(z(t))dB(t)$
  - A likelihood (noise model) p(x_t | z_t)



(b) Generation

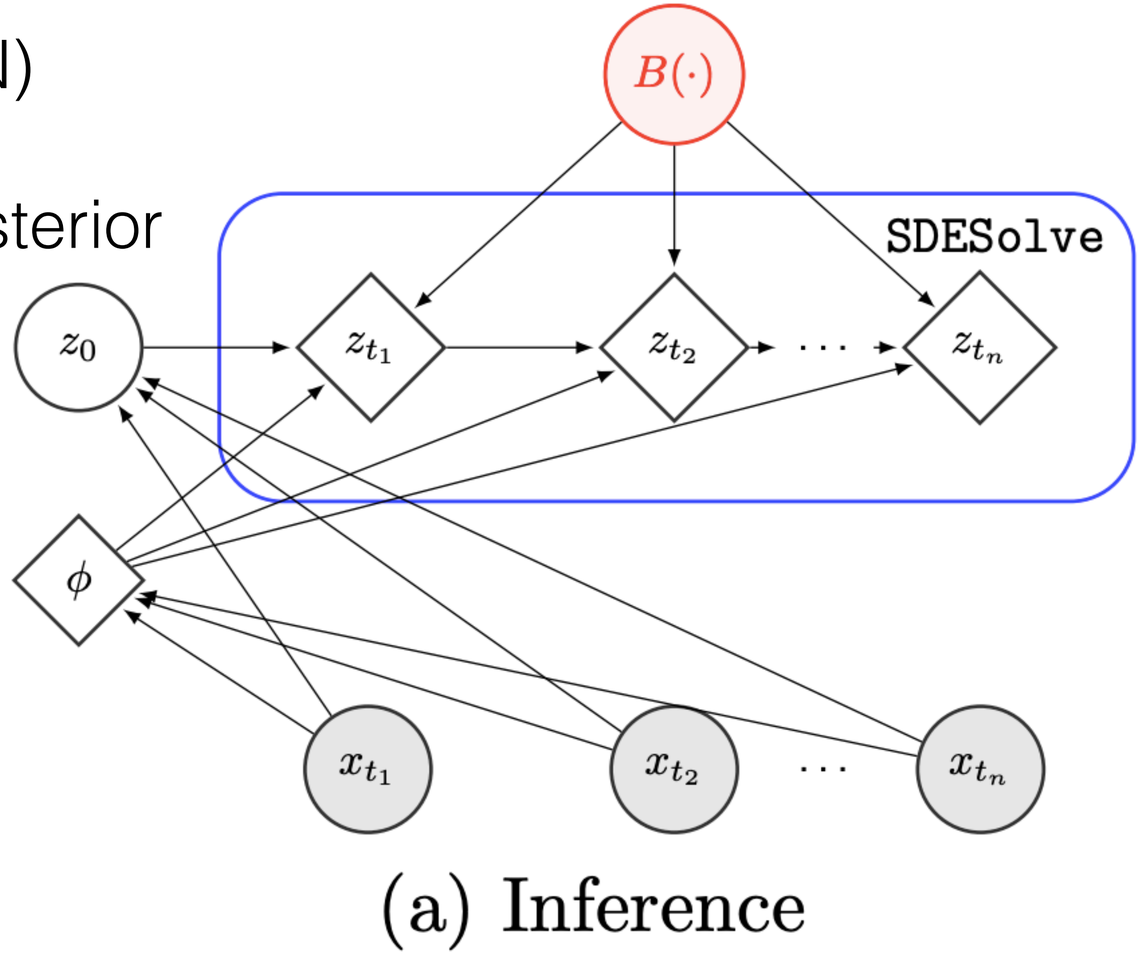
Variational inference

- Recognition model (encoder) takes in data, outputs:
  - Distribution over initial state q(z0 | x1.. xN)
  - Params of SDE defining approximate posterior

$$dz_q = f_\phi(z(t))dt + \sigma_\theta(z(t))dB(t)$$

  - True posterior also has this form (Tzen & Raginsky, 2018 + 2019)

- Can handle arbitrary likelihoods.



(a) Inference

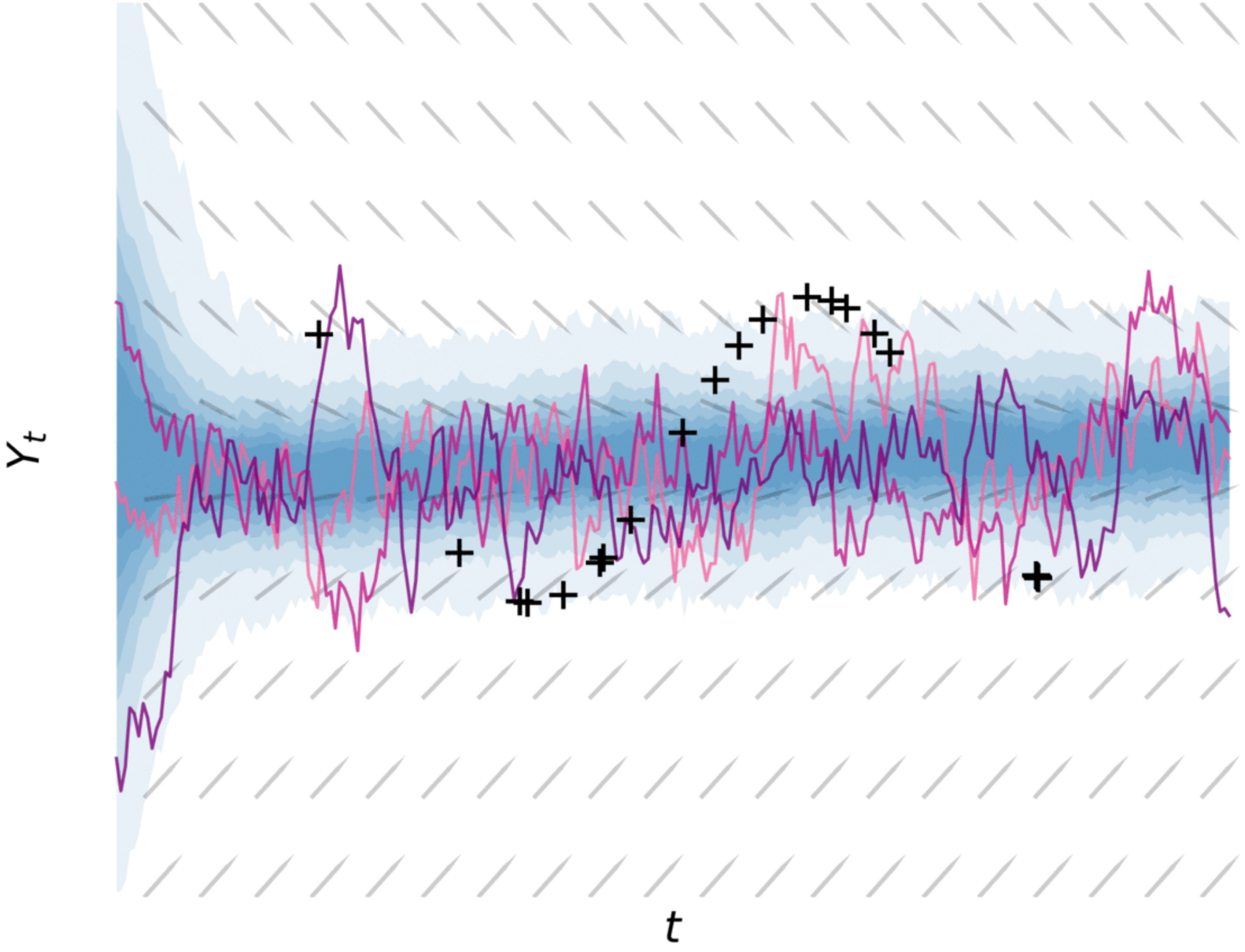
Variational inference

- prior: $dz_p = f_\theta(z_t)dt + \sigma_\theta(z_t)dB(t)$
- approximate posterior: $dz_q = f_\phi(z_t)dt + \sigma_\theta(z_t)dB(t)$

$$ELBO(q) = \log p(y_i \mid x_i) - \mathbb{E}_{q(w|\phi)}\left[\frac{1}{2}\int \left|\frac{f_\theta(w_t) - f_\phi(w_t)}{\sigma_\theta(w_t)}\right|_2^2 dw\right]$$

1D Latent SDE

- Ornstein-Uhlenbeck prior, Laplace likelihood
- Posterior SDE steers sample paths to data



Building an infinitely-deep BNN

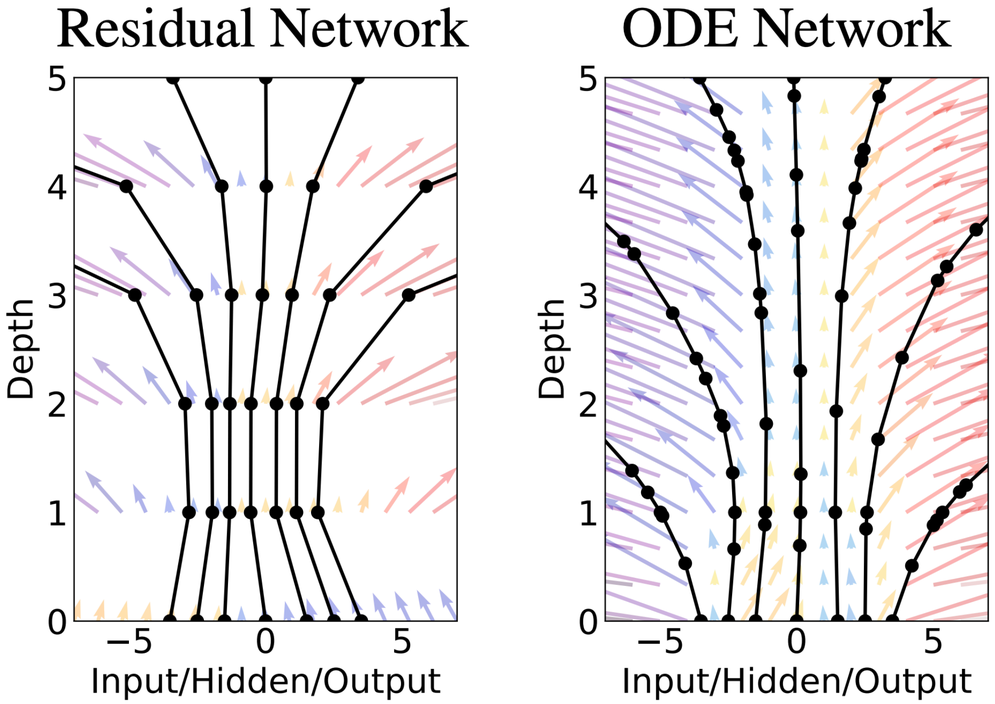
- Start with a ResNet:

$$h_{t+\epsilon} = h_t + \epsilon f_h(h_t, w_t)$$

- Take limit as eps -> 0, number of layers grows.

- Given a process over weights, activations h follow a random ODE:

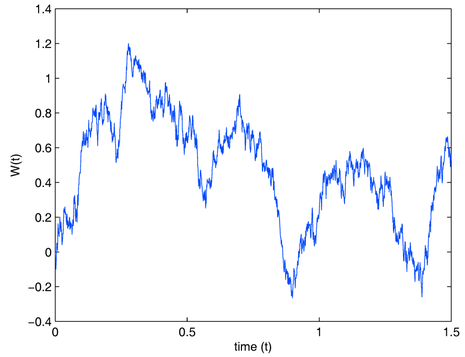
$$dh_t = f_h(h_t, w_t)$$



Building an infinitely-deep BNN

- Prior on weights is a OU process

$$dw_t = -w_t dt + dB_t$$

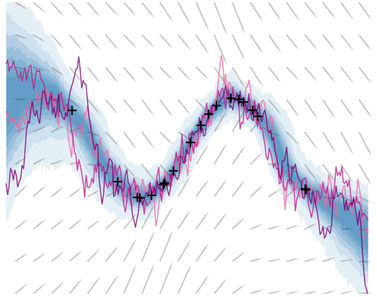


- Likelihood depends on activation at time 1:

$$p(y \mid x, w) = \mathcal{N}(y \mid h_1, w)$$

- Define approximate posterior on weights:

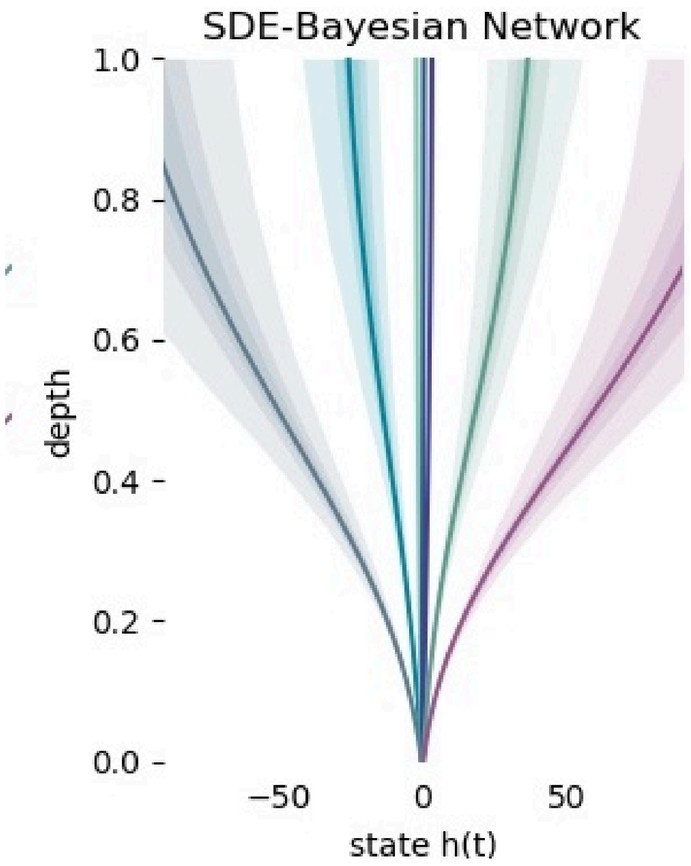
$$dw_t = f_w(w_t, \phi)dt + dB_t$$



Building an infinitely-deep BNN

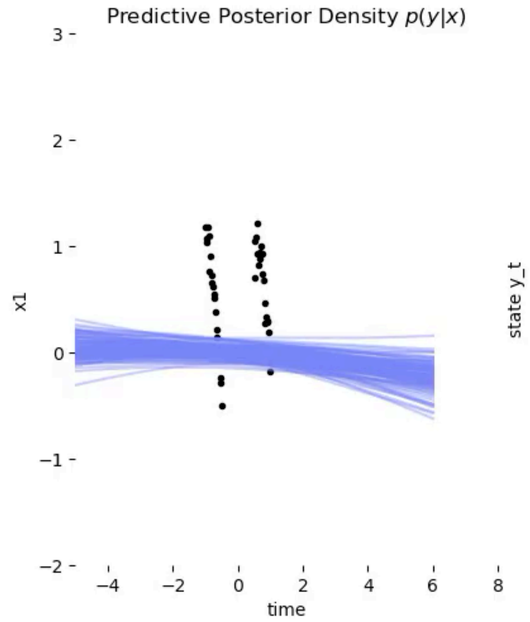
- Can sample weights from approx posterior and evaluate network output in one SDE solve:

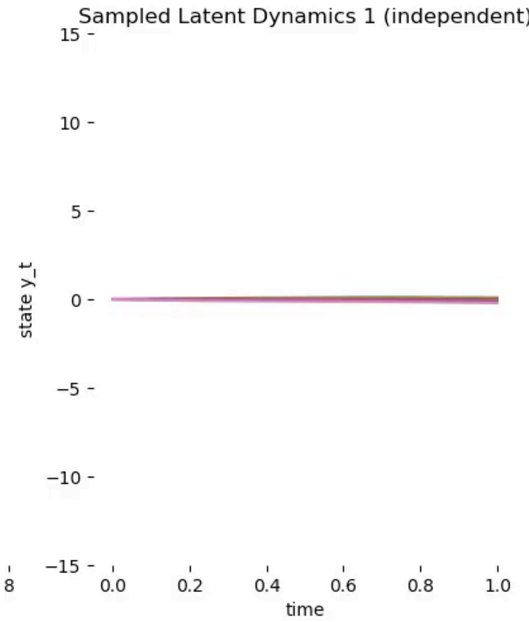
$$d\begin{bmatrix} w_t \\ h_t \end{bmatrix} = \begin{bmatrix} f_w(w_t, \phi) \\ f_h(h_t, w_t) \end{bmatrix} dt + \begin{bmatrix} I \\ \mathbf{0} \end{bmatrix} dB_t$$

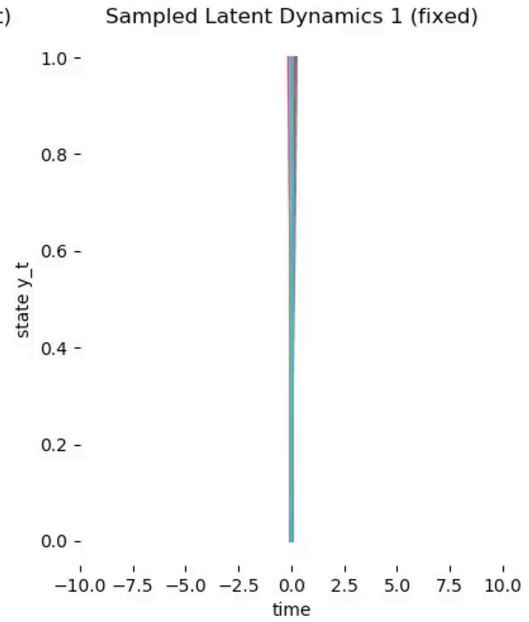


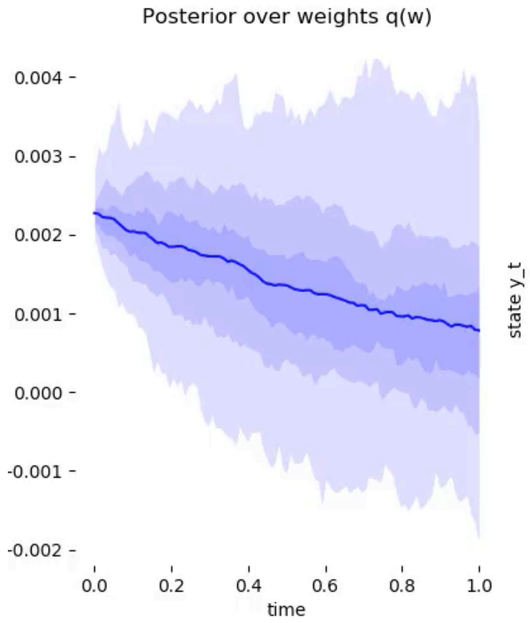
- Start h_0 at input to neural network x.
- h_1 is output of neural network

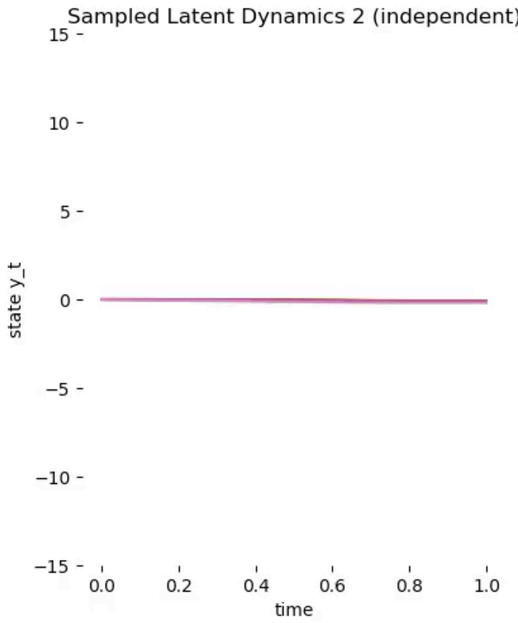
Training an infinitely-deep BNN

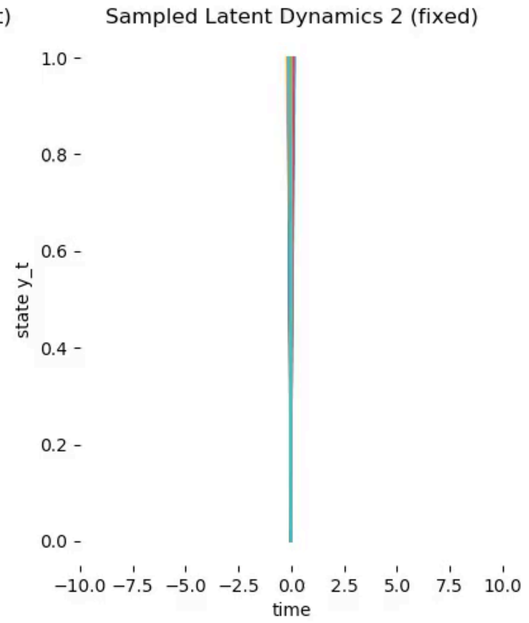












Speed vs precision

- Continuous-time formulation allows use of adaptive SDE solvers.
- Can adjust adaptive solver tolerance at test time.
- Trades off speed vs precision

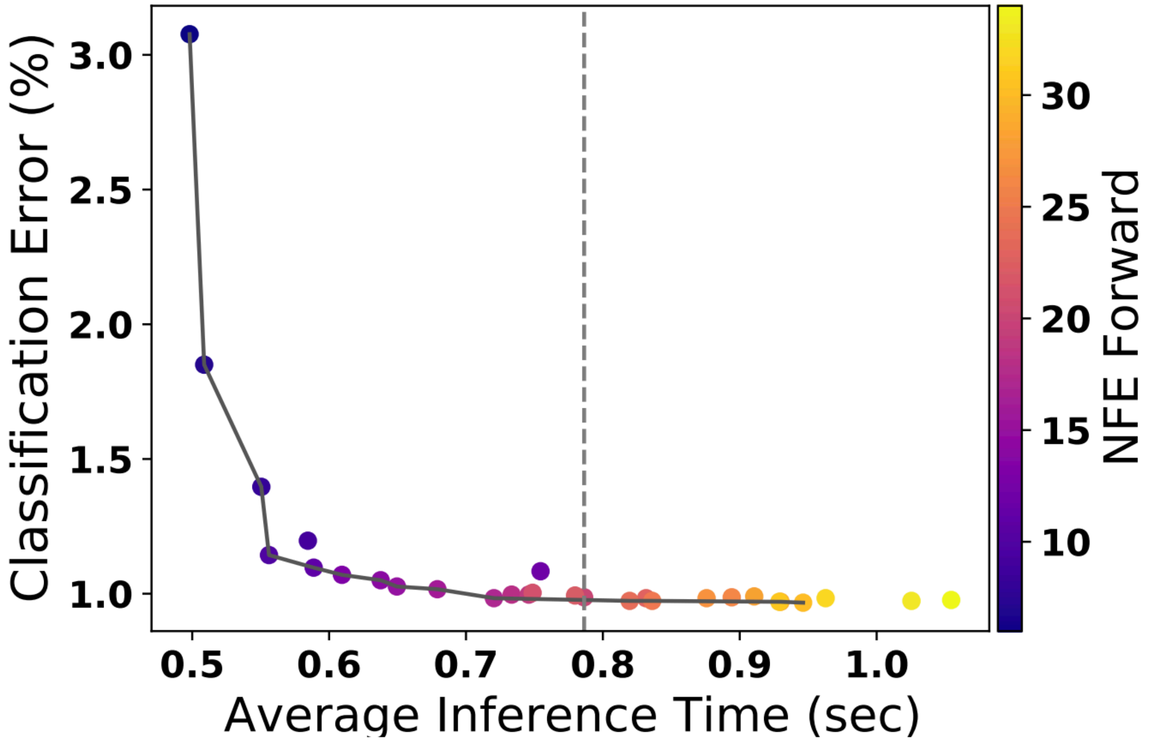
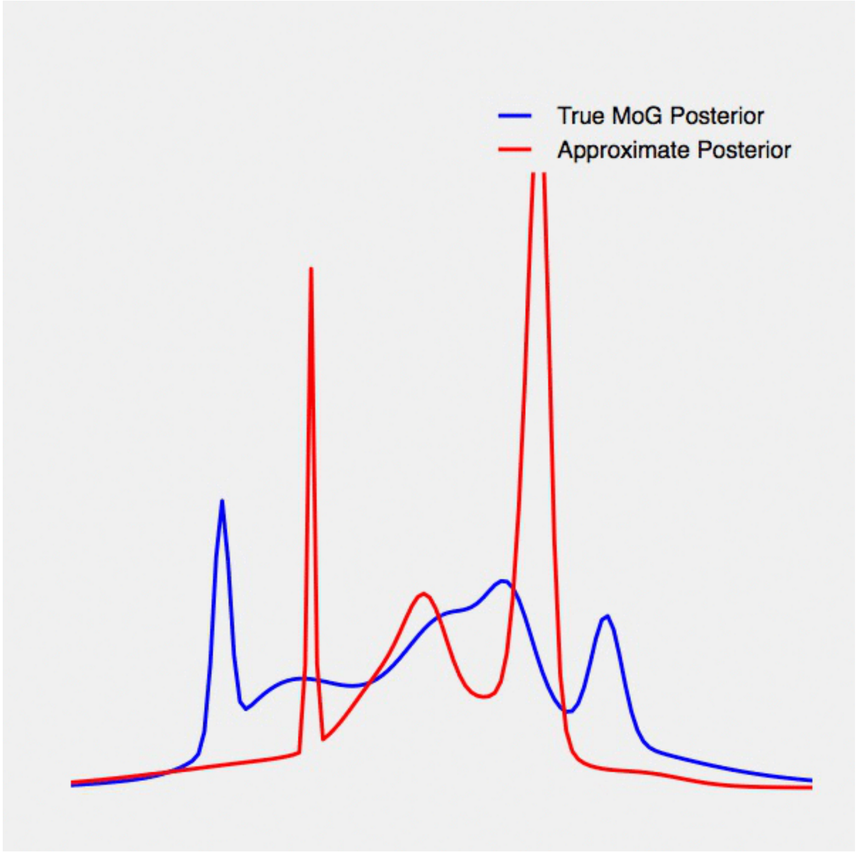


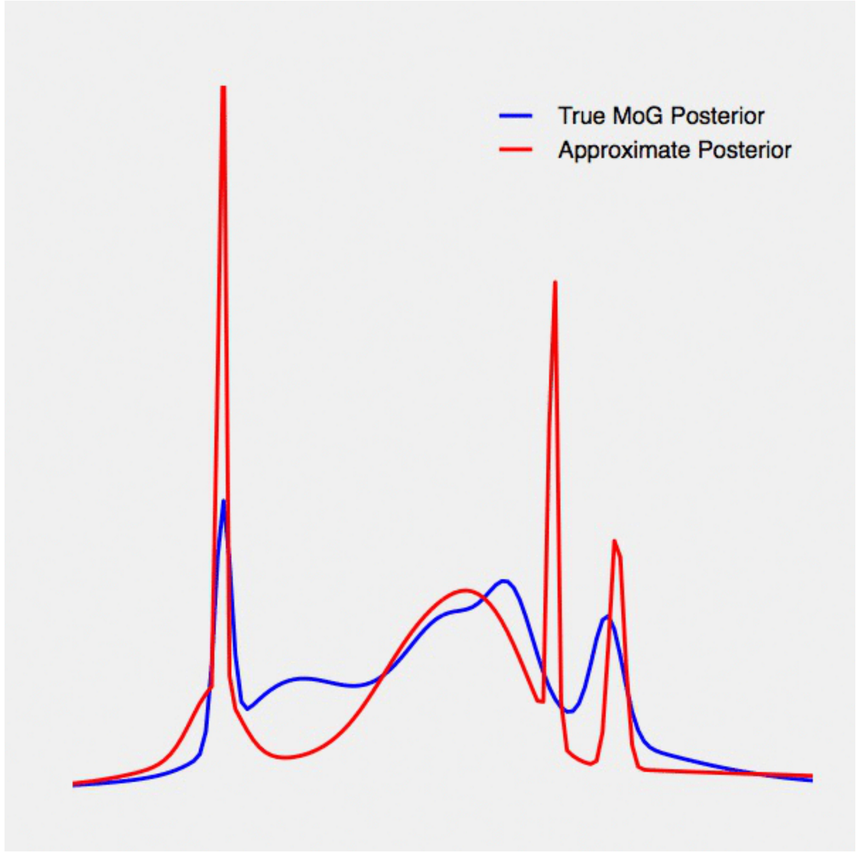
Image Classification Results

	MNIST		CIFAR-10	
Model	Accuracy (%)	ECE ($\times 10^{-2}$)	Accuracy (%)	ECE ($\times 10^{-2}$)
ResNet32	99.46 ± 0.00	2.88 ± 0.94	87.35 ± 0.00	8.47 ± 0.39
ODEnet	98.90 ± 0.04	1.11 ± 0.10	88.30 ± 0.29	8.71 ± 0.21
HyperODEnet	99.04 ± 0.00	1.04 ± 0.09	87.92 ± 0.46	15.86 ± 1.25
MFVI ResNet32	99.44 ± 0.00	2.76 ± 1.28	86.97 ± 0.00	3.04 ± 0.94
MFVI†	—	—	86.48	1.95
Deep Ensemble†	—	—	89.22	2.79
HMC ("gold standard")†	98.31	1.79	90.70	5.94
MFVI ODEnet	98.81 ± 0.00	2.63 ± 0.31	81.59 ± 0.01	3.62 ± 0.40
MFVI HyperODEnet	98.77 ± 0.01	2.82 ± 1.34	80.62 ± 0.00	4.29 ± 1.10
SDE BNN	99.30 ± 0.09	0.63 ± 0.10	89.84 ± 0.94	7.19 ± 0.37
SDE BNN (+ STL)	99.10 ± 0.09	0.78 ± 0.12	89.10 ± 0.45	7.97 ± 0.51

SVI Gradient variance

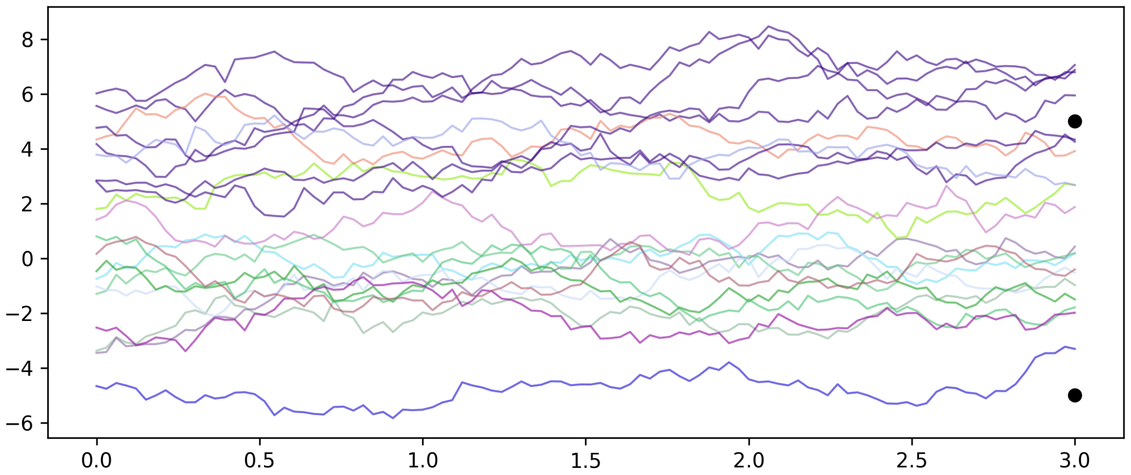
- Sticking the landing [Roeder, Wu, Duvenaud, NIPS 2017]
  SVI gradient estimator whose variance goes to zero as q(z) -> p(z|x)

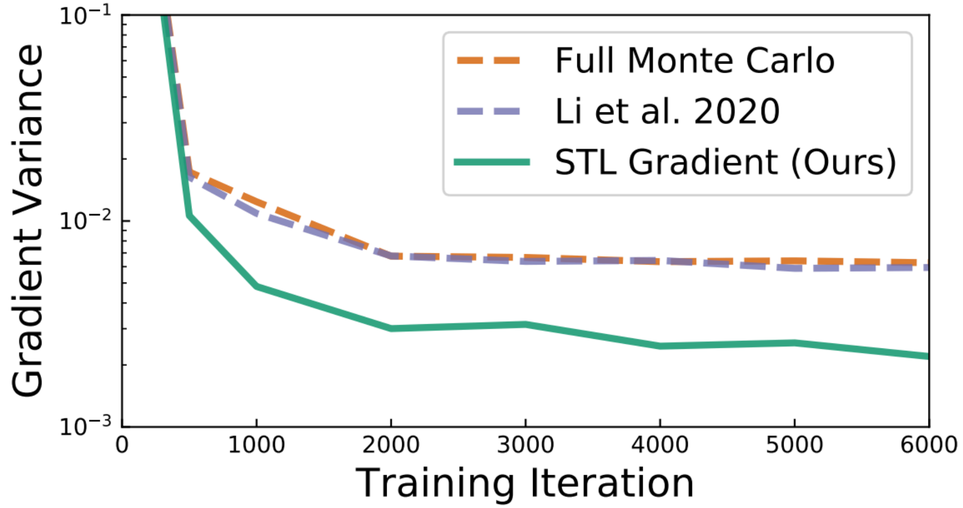


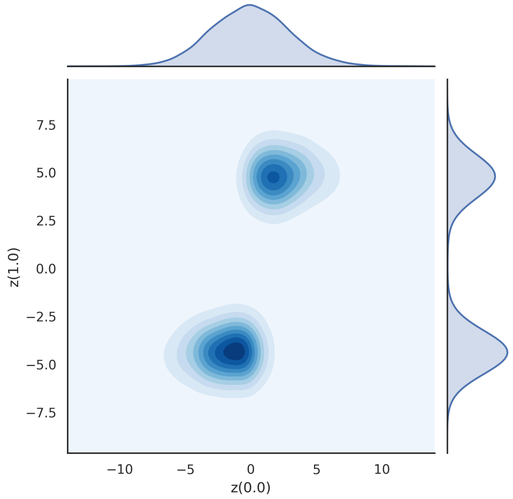


SVI Gradient variance

- Approx posterior can be arbitrarily close to true posterior!



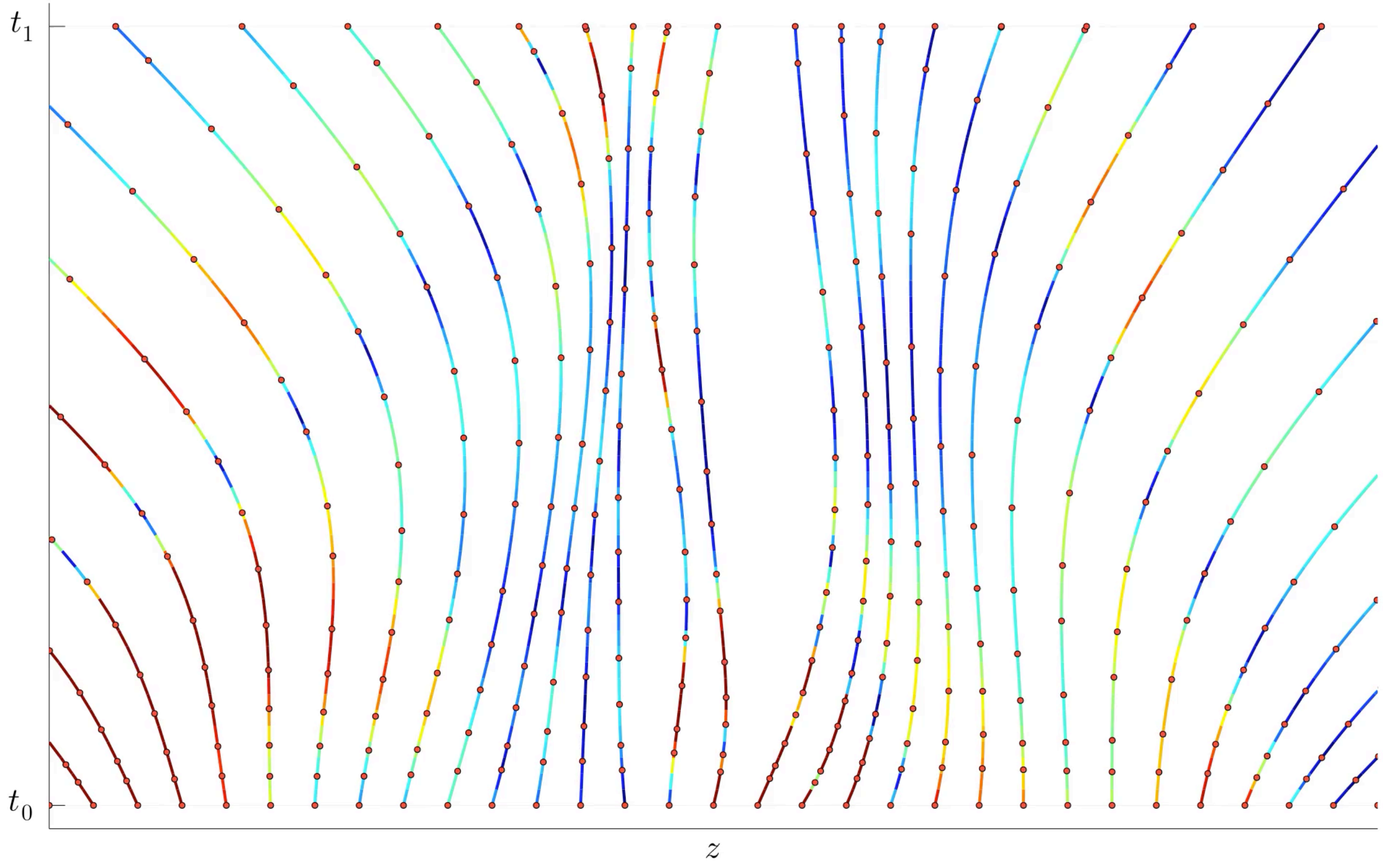


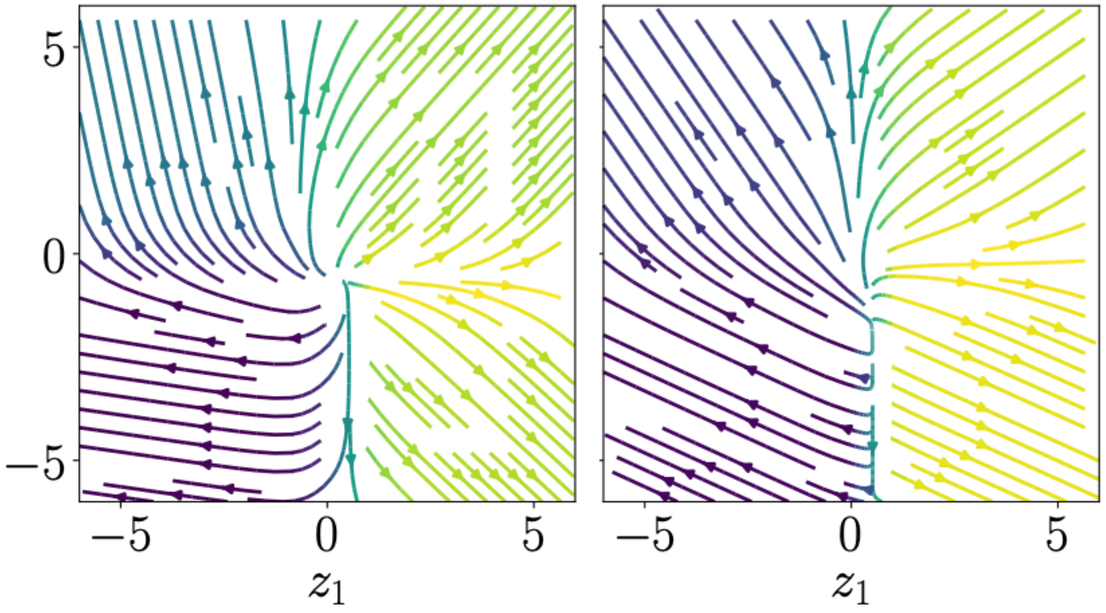


- ICML 2021: extended "Sticking the Landing" to SDEs

Regularizing Dynamics to be Easy To Solve

$$\mathcal{R}_K(\theta) = \int_{t_0}^{t_1} \left\| \frac{\mathrm{d}^K \mathbf{z}(t)}{\mathrm{d}t^K} \right\|_2^2 \mathrm{d}t$$





(a) Unregularized (b) Regularized

Learning Differential Equations that are Easy to Solve
Jacob Kelly*, Jesse Bettencourt*, Matthew Johnson, David Duvenaud

Limitations of continuous-time architectures

- Need all layers Lipschitz
- Sometimes need extra dimensions
- 2-4x slowdown up front due to solvers
  - But can regularize to be fast to solve
- Error tolerance not directly convertible to precision / second

The Lipschitz Constant of Self-Attention

Hyunjik Kim [1] **George Papamakarios** [1] **Andriy Mnih** [1]

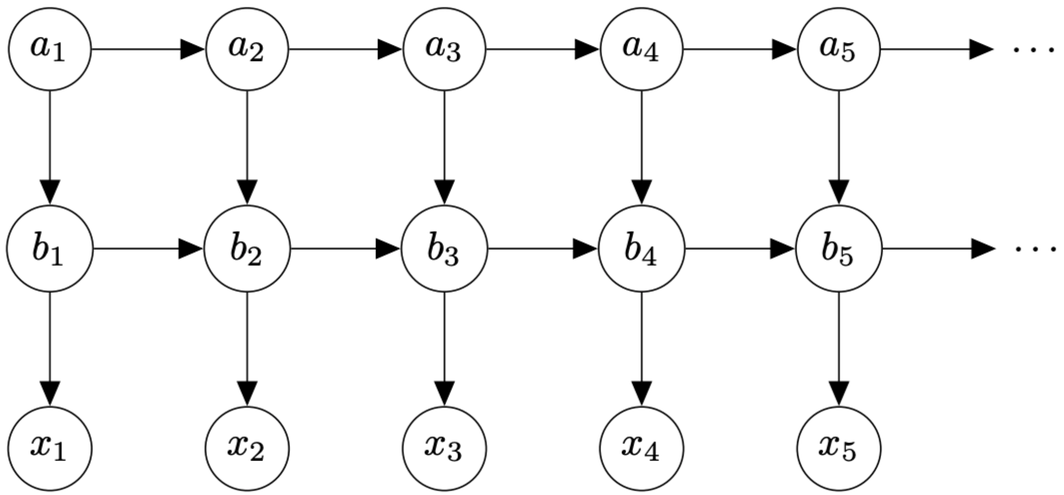
in sequence modelling. We prove that the standard dot-product self-attention is *not* Lipschitz for unbounded input domain, and propose an alternative L2 self-attention that *is* Lipschitz. We

What is the ultimate architecture?

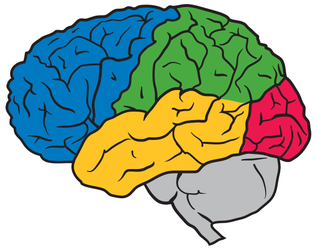
- Specify hyperparameters indirectly through utility funcs
  - E.g. speed vs accuracy tradeoffs
  - Decouple compute from number of params
  - O(1) memory training
- Predictions that include model uncertainty
  - Arbitrarily expressive approx. posterior
- Low-variance gradients

Future directions

- Bayesian DEQs
- Regularize to be faster to solve
- Back-port low-variance trick to finite-depth networks
- Explore latent SDE model class!
- Multi-scale latent SDEs



Winnie Xu, Xuechen Li, Ricky T.Q. Chen

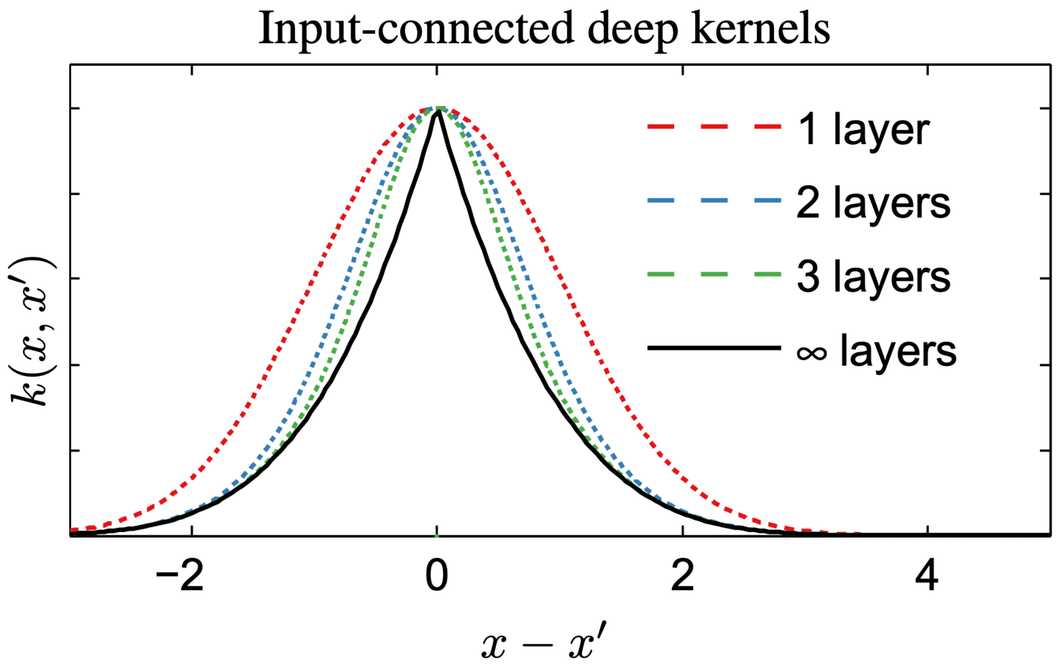


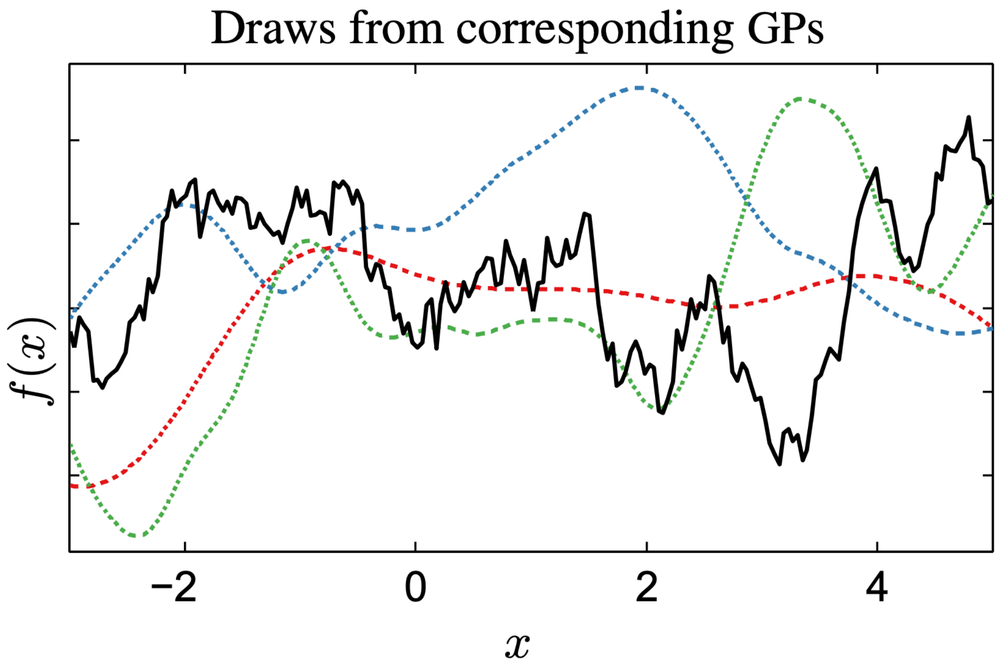
github.com/google-research/torchsde

Thanks!

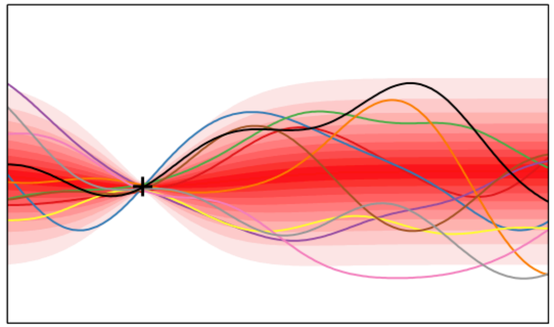
Some Previous Work

- Radford Neal's thesis (1994):
  - Infinite width -> GPs
  - Infinite depth -> Forget input. (Remedy: Connect input to each layer)
- Avoiding Pathologies in Very Deep Networks. (2014)



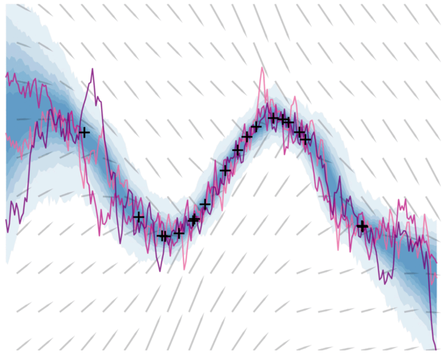


GPs



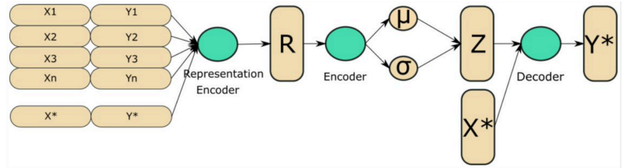
- mean and kernel funcs
- Not closed over marginal transforms.
  exp(f(x) ~ GP)
  not a GP
- Multi-dim input fine

SDEs



- Drift and diffusion funcs
- Closed over marginal transforms.
  exp(f(x) ~ SDE)
  still an SDE
- Only single-dim index (Can discretize space)

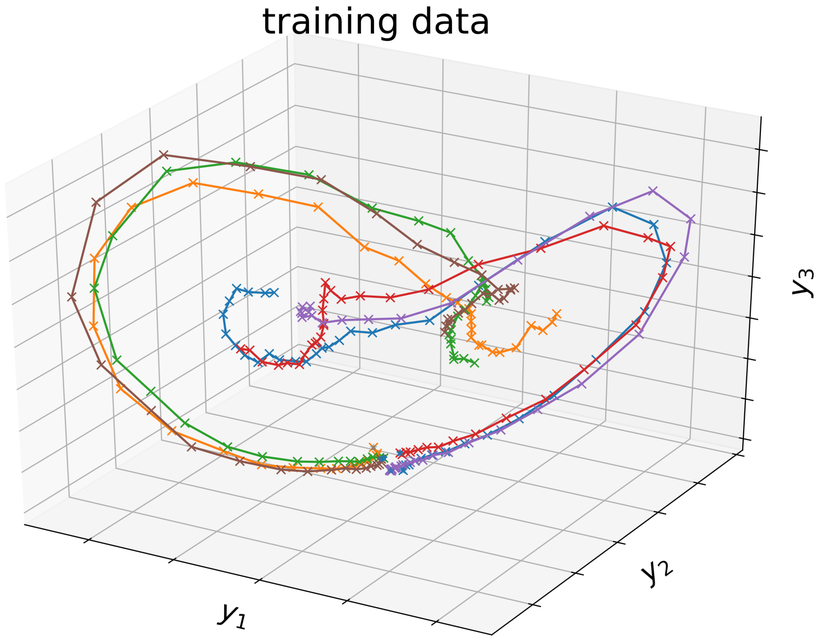
NPs

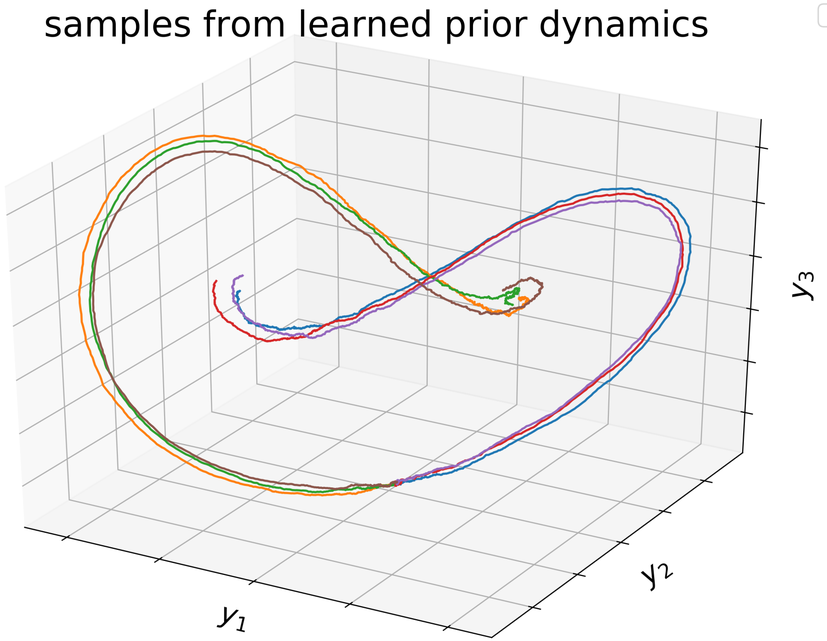


- Everything from meta-learning
- Hard to get joint predictives
- Multi-dim input

Latent SDEs:
An unexplored model class

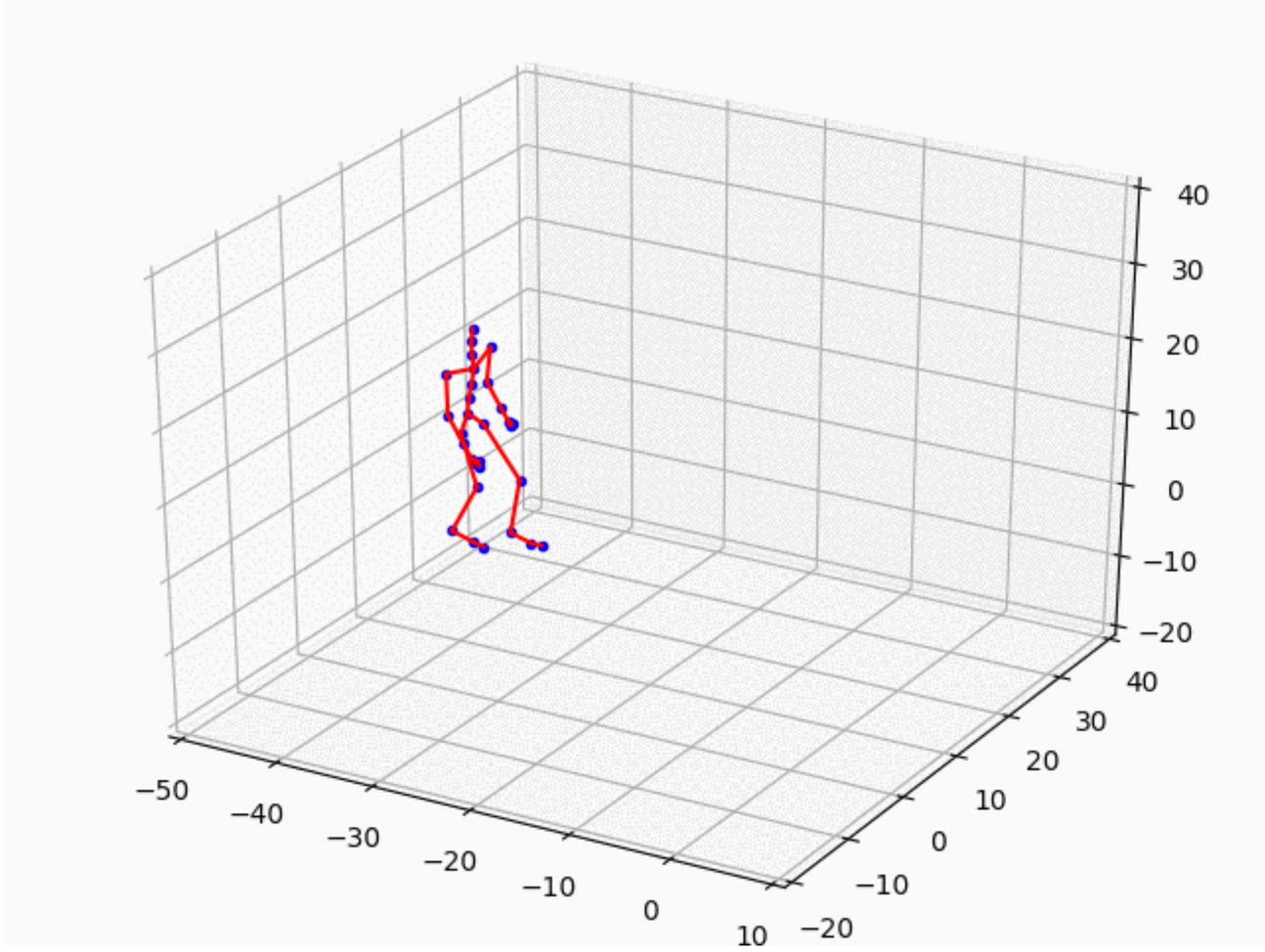
- Define implicit prior + posterior over functions
- Define observation likelihoods. Anything differentiable wrt latent state (e.g. text models!)
- Train everything with stochastic variational inference.
- Can use adaptive-step SDE solvers.
- Should scale to millions of params, huge states. Can use adaptive Milstein solver (only diagonal noise).





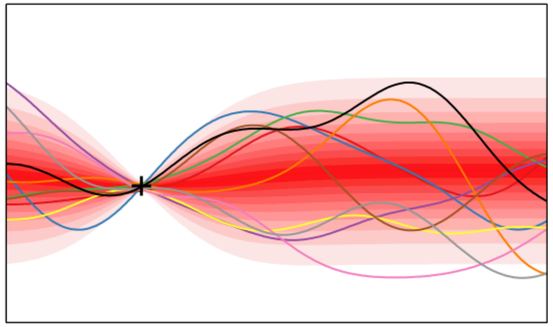
Early latent SDE results: Mocap

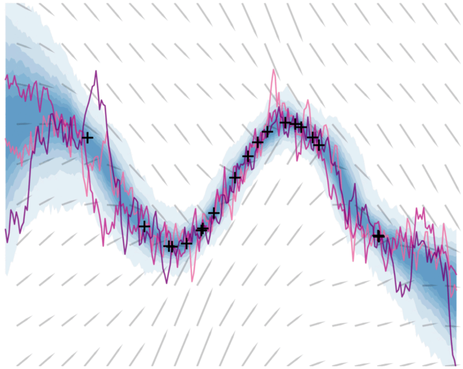
- 50D data, 6D latent space, sharing dynamics and recognition params across time series (11000 params)



Method	Test MSE
npODE [18]	22.96†
NeuralODE [4]	$22.49 \pm 0.88^{\dagger}$
ODE^{2}VAE [61]	$10.06 \pm 1.4^{\dagger}$
ODE^{2}VAE-KL [61]	$8.09 \pm 1.95^{\dagger}$
Latent ODE [4, 50]	5.98 ± 0.28
Latent SDE (this work)	$\mathbf{4.03 \pm 0.20}$

GPs vs Markov SDEs





- Drift and diffusion funcs ~ mean and kernel funcs (both are hyperparams that define a dist over funcs)

- SDEs: Closed over marginal transforms.
  g(x) = exp(f(x))
  not a GP if f ~ GP,
  is still an SDE if f ~ SDE

- SDEs: tractable with only one continuous index (time) as far as I know. Multivariate state is fine.

Dex: a typed array language built for speed

```
def map (f : a->b) (xs : n=>a) : n=>b =
  for i. f x.i
```

Flexibility

- Ragged and sparse arrays
- Algebraic data types (e.g. `Value|NaN|Missing`)

Correctness

- Dependent types for compile-time debugging (e.g. shape checking)
- Composable, zero-cost abstractions (e.g. run on any vector space)

Performance

- Fast nested loops + gradients (e.g. CTC loss)
- CPU, GPU, TPU backends, JAX interop

Ray tracer written in Dex
google-research.github.io/dex-lang/raytrace.html

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

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 Raginsky

Neural Stochastic Differential Equations

Stefano Peluchetti, Stefano Favaro

Neural Jump Stochastic Differential Equations

Junteng Jia, Austin R. Benson

Related work 2

- Thomas Ryder, Andrew Golightly, A Stephen Mc-Gough, and Dennis Prangle. *Black-box variational inference for stochastic differential equations.*

- Pashupati Hegde, Markus Heinonen, Harri Lähdesmäki, and Samuel Kaski. *Deep learning with differential gaussian process flows.*

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

- *C. Garcıa, A. Otero, P. Felix, J. Presedo, and D. Marquez. Nonparametric estimation of stochastic differential equations with sparse 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.

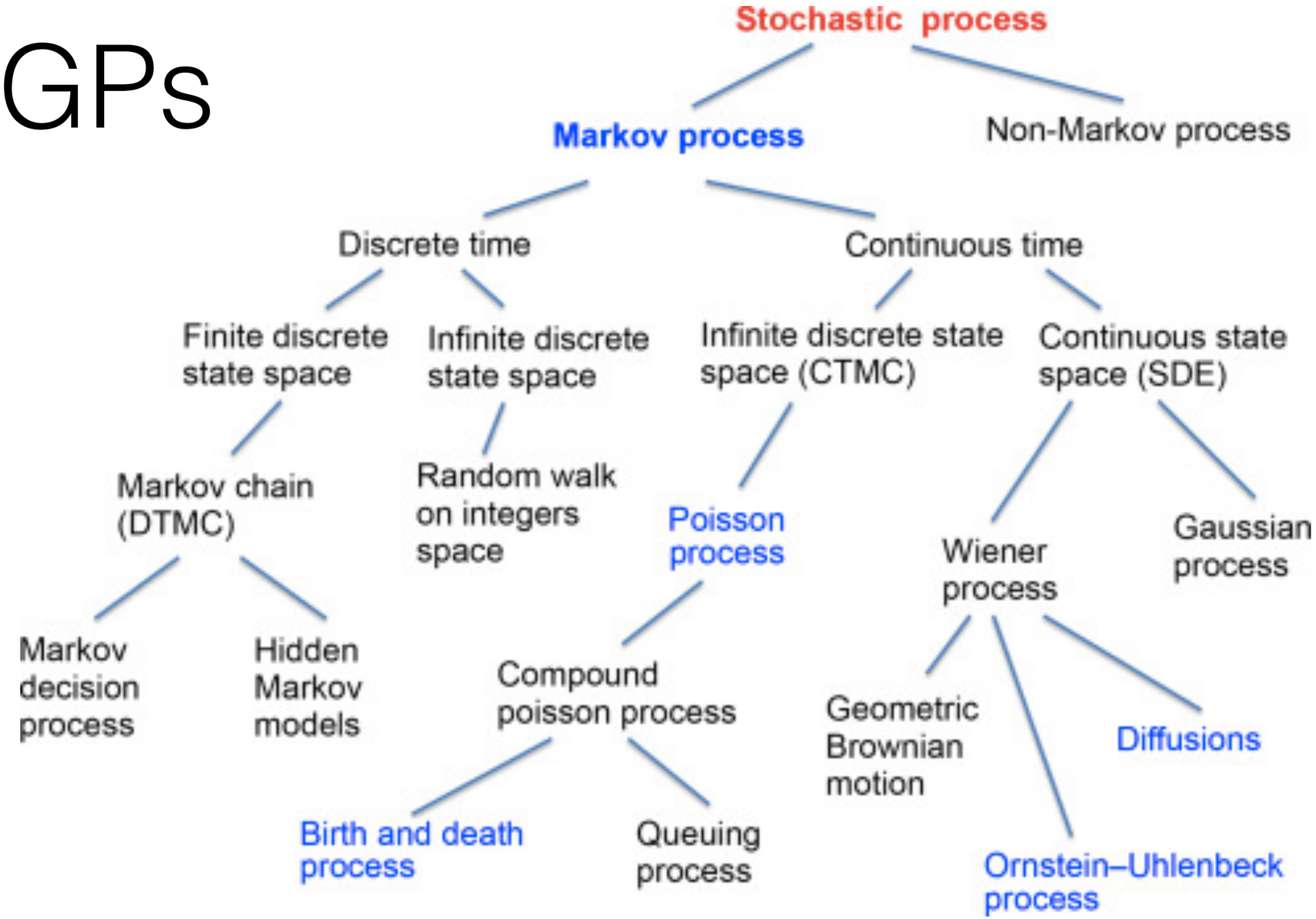
- We want to use adaptive, (high-order?) SDE solvers.

Limitations

- SDE solvers generally 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)
- Only one input dimension (unlike GPs)

SDEs vs GPs

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

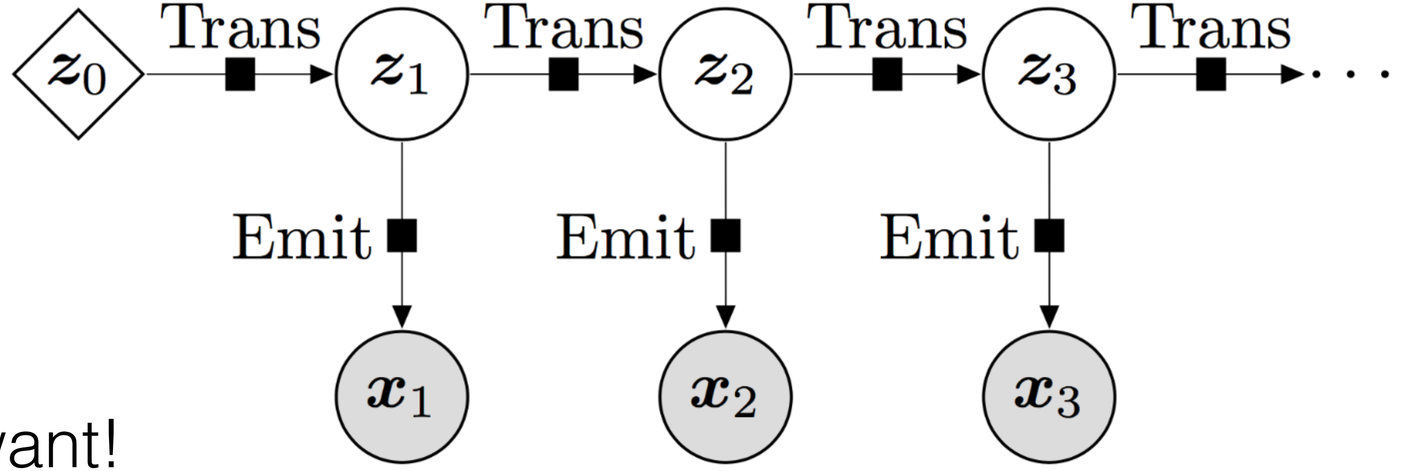


From “Handbook of Statistics”, Mubayi et al, 2019.
Line means "can be used to construct", but not "contains"

Latent variable models

- Kalman Filters, Hidden Markov Models, Deep Markov Models
  - specify p(z), p(x|z)

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

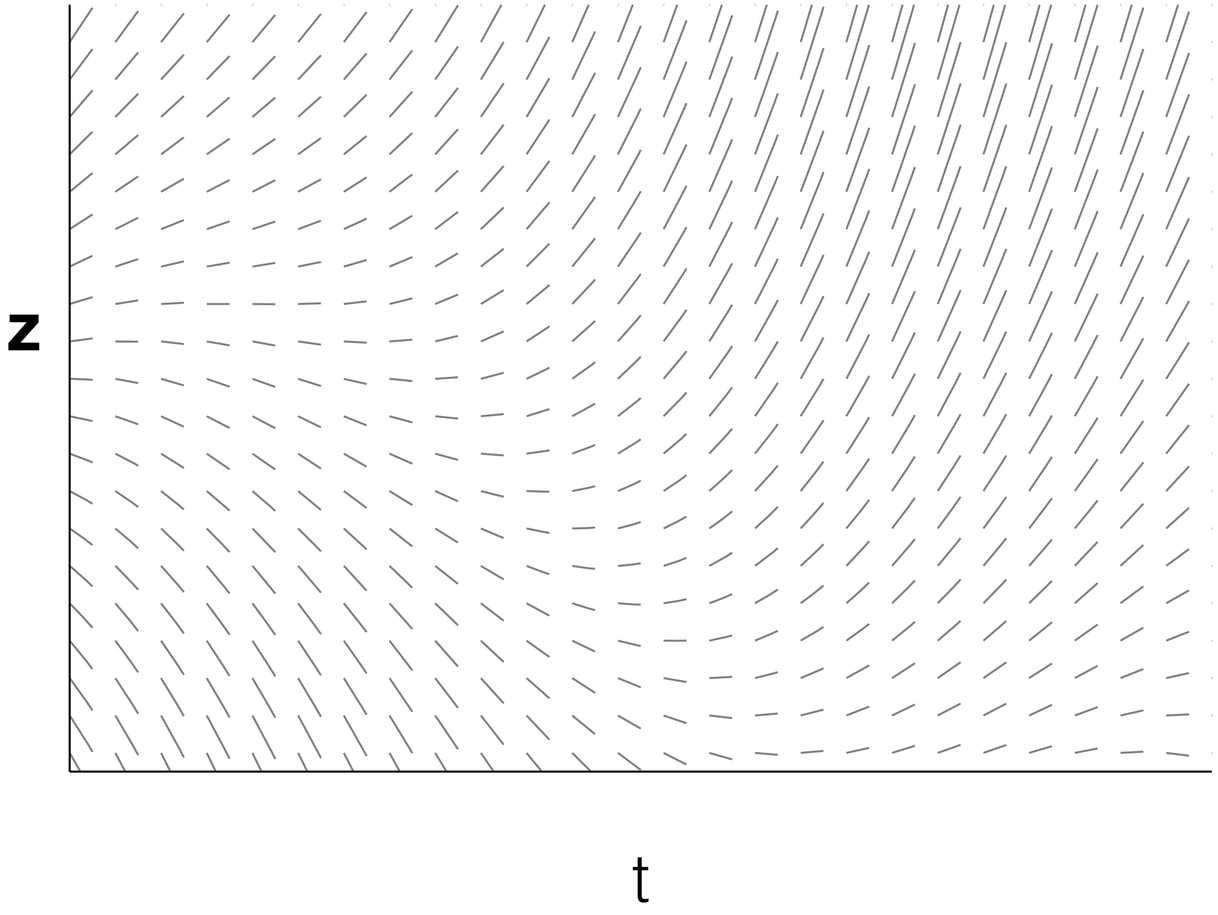


https://pyro.ai/examples/dmm.html

- Can integrate out z however you want!
  - Recognition net can give approx. posterior

[Krishnan, Shalit & Sontag '15]

Ordinary Differential Equations



- Vector-valued **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:

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