Supplementary tables and figures

1 Experiment setup

We test the ODE models for two tasks: interpolation and extrapolation.

Interpolation Consider time series with time points $(t_0...t_N)$. In interpolation task we condition on the subset of points from $(t_0...t_N)$ and reconstruct the full set of points in the same time interval. We subsample the points by setting the value to zero and mark them in the mask as "missing". On MUJoCo dataset we perform experiments with variable percenrages of subsampled points (10%-50%, results shown in table 2). In Physionet and Human Activity datasets, we do not perform subsampling since the data is already sparse.

Extrapolation In extrapolation task we split the time series into two parts: $(t_0...t_{N_2})$ and $(t_{N_2}..t_N)$. We encode the first half of the time series and reconstruct the second half.
Similarly to interpolation task, we randomly sample a percentage time points from the time series and run a recognition network on this subset of points. We evaluate the model by the ability to reconstruct the full time series (without subsampling).
Autoencoder models are straightforward to use for extrapolation task. We run the encoder on the first half of the sequence and decode the second half. For autoregressive models, we train the model to perform interpolation first, and then perform extrapolation at test time by re-feeding previous predictions of the model. During training, we feed in either the previous observed value or predicted value with probability 0.5, a common regularization method (Goodfellow et al., 2016). For all experiments, we report the mean squared error (MSE) on a test set of held-out sequences.

Choice of first time point of ODESolve The generative model has a special time point t_0 where we put a prior on the latent state. ODE can be solved both forward or backward in time, and we are free to choose the time point t_0 depending on the task. As such, for interpolation task, we choose t_0 to be the time point of the first observation, as shown in eq. (5-7), and run ODE-RNN backwards in time to obtain the approximate posterior. For extrapolation task, we choose the initial point to the the last observed time point t_N. In this case, we run ODE-RNN encoder forward in time.

Table 1

Task	Encoder	Time point used for VI	Decoder
Interpolation	Backwards in time $t_N \rightarrow t_0$	t_0	Forward in time in $[t_0, t_N]$ interval
Extrapolation	Forward in time $t_0 \rightarrow t_{N/2}$	$t_{N/2}$	Forward in time in $[t_{N/2}, t_N]$ interval

2 Model details

2.1 GRU update

Algorithm 1 GRU

Input: Observations x, previous hidden state h_{prev}
$z = \sigma(f_z([h_{prev}; x]))$ ▷ Update coefficient
$r = \sigma(f_r([h_{prev}; x]))$ ▷ Reset coefficient
$h' = g([r * h_{prev}; x])$ ▷ Proposed new state
$h = (1 - z) * h' + z * h_{prev}$
Return: new hidden state h

2.2 Latent ODE

Algorithm 2 Latent ODE

Input: Data points $\{x_i\}_{i=1..N}$ and corresponding times $\{t_i\}_{i=1..N}$
$z'_0 = \text{ODE-RNN}(\{x_i\}_{i=1..N})$
$\mu_{z_0}, \sigma_{z_0} = g_\mu(z'_0), g_\sigma(z'_0)$
$z_0 \sim \mathcal{N}(\mu_{z_0}, \sigma_{z_0})$
$\{z_i\} = \text{ODESolve}(f, z_0, (t_0 ... t_N))$
$\tilde{x}_i = OutputNN(z_i)$ for all $i = 1..N$
Return: $\{\tilde{x}_i\}_{i=1..N}$

2.3 Modelling poisson process likelihood

To model poisson process on Physionet dataset, we augment generative ODE $\frac{d}{dt}z = f(z)$ by adding extra latent dimensions z_λ. Intensity function λ is a function of latents z_λ: $\lambda = g_\lambda(z_\lambda)$, where g_λ is two-layer feed-forward neural net. Dimensionality of λ has to be equal to dimensionality of the data (37 in Physionet). We used 30 latent dimensions for z and 30 dimensions for z_λ. We further augment the ODE with the integral over λ. Notice that the derivative $\int_0^t \lambda(\tau)d\tau$ is $\lambda(t)$. Thus, the augmented ODE is defined as follows:

$$\frac{d}{dt}\begin{bmatrix} z \\ z_\lambda \\ \int_0^t \lambda(\tau)d\tau \end{bmatrix} = \begin{bmatrix} f(z) \\ f_{z_\lambda} \\ \lambda \end{bmatrix}; \quad \text{where} \quad \lambda = g(z_\lambda)$$

We set the initial value for $\int_0^t \lambda(\tau)d\tau$ to zero (notice that $\int_0^0 \lambda(\tau)d\tau = 0$. Initial values of z and z_λ are sampled from approximate posterior. The augmented ODE is solved using a call to ODESolve.

Generative model with Poisson Process The joint generative model is specified as $p(z_0)p(t_0, \ldots, t_N|z_0)\prod_{i=0}^N p(x_i|z_0)$, where the distributions are specified below.

$$p(z_0) = \text{Normal}\big(z_0; 0, I\big) \tag{1}$$
$$\{z(t_i)\}_{i=0}^N = \text{ODESolve}\big(f_\theta, z_0, (t_0, \ldots, t_N)\big) \tag{2}$$
$$p(t_0 ... t_N|z_0) = \text{PoissonProcess}\big(t_1 ... t_N; \lambda(z(t))\big) \tag{3}$$
$$p(x_i|z_0) = \text{Normal}\big(x_i; \mu(z(t_i)), \sigma(z(t_i))\big) \tag{4}$$

The Latent ODE framework specifies a generative model for time series.

$$y_0 \sim p(y_0) \tag{5}$$
$$\lambda(t) = \text{ ODESolve}(y_0, f_\lambda, \theta_{f_\lambda}, t) \tag{6}$$
$$t_1 ... t_N \sim \text{ PoissonProcess}(\lambda(t)) \tag{7}$$
$$y_1, ... y_N = \text{ ODESolve}(y_0, f_y, \theta_{f_y}, t_1 ... t_N) \tag{8}$$
$$x_i \sim p(x|y_i, \theta_x), \ \forall i \in \{1..N\} \tag{9}$$

3 Data generation and preprocessing

Toy dataset We generate 1000 one-dimentional trajectories with 100 time points in each on the $[0, 5]$ interval. We use sinusoid with fixed amplitude of 1 and sample frequency from $[0.5, 1]$ interval. We sample the starting point from $\mathcal{N}(\mu = 1, \sigma = 0.1)$.

MuJoCo We generate 10,000 simulations of the "Hopper" model from Deep Mind Control Suite and MuJoCo simulator. We sample the position of the body in 2d space uniformly from $[0, 0.5]$. We sample the relative position of limbs from $[-2, 2]$ and initial velocities from $[-5, 5]$ interval. We generate trajectories with 200 time steps for extrapolation tasks (100 points to condition on and 100 for extrapolation). We use 100 time points to perform interpolation task.

Physionet PhysioNet contains the data from the first 48 hours of patients in ICU. We have excluded four time-invariant features: Age, Gender, Height and ICUType. Hence, each patients has a set of up to 37 features (most patients have only a subset of those features).
The original Physionet challenge has Train set (labeled) and Test set (unlabeled). We combine them into a single dataset (8000 patients, 37 features in each) and randomly split into 80% train and 20% test set, similarly to other datasets. We normalize each feature across all patients in the dataset to be in $[0, 1]$ interval. Each feature in each patients was measured at different time points. In order to perform batching of the features and patients during the training, we take the union of all time points across all features and patients in the dataset. For the sake of reducing the size of the union, we round up the time stamps to 6 minutes. The resulting time series has 480 points. we use a mask in ODE-GRU and in the recognition model of ODE-VAE to annotate the features that are present at the particular time point in order to update the latent state. Note that generative part of ODE-VAE does not require any masking.

Human Activity The original dataset contains 25 sequences from five people (6600 points on average in each sequence). For the sake of reducing the size of union of time points, we round up the time stamps of the measurement to 100 ms – such discretization does not change the overall number of points. We join the data from four tags (belt, chest, ankles) into a single time series and then split the each sequence into partially overlapping intervals of 50 time points (with overlap of 25 points). We combine sequence from all individuals into a single dataset. After taking the union of all time points in the dataset, we get the dataset of 6554 sequences with 211 time points in each. We do not perform normalization on this dataset.
The labels are provided for each observation and denote the type of activity that the person is performing, such as walking, sitting, lying, etc. Original dataset has 11 classes. Some classes correspond to very similar activities, which are hard to distinguish. We decided to combine the classes within the following groups: ("lying", "lying down"), ("sitting", "sitting down"), ("standing up from lying", "standing up from sitting", "standing up from sitting on the ground"). The resulting set of classes describes seven activity types: "walking", "falling", "lying", "sitting", "standing up", "on all fours", "sitting on the ground". We run all experiments and report results using these seven classes.

General notes For all datasets, we take the union of all time points in the dataset and run all models on the union. We randomly split the dataset into 80% train and 20% test. In Physionet dataset we rescale

each feature to be between 0 and 1. We do not normalize other datasets. We also rescale the timeline to be in $[0, 1]$.

4 Architecture

ODE function We use a feed-forward neural net for ODE function f. For most experiments, we used a two-layer network for ODE function in generative model and four layers in recognition model in Latent ODE. See section 5 for sizes of the network in each experiment.
We used Tanh activations in ODE function. Tanh activation constrains the ODE gradients and prevent them from taking high values, which is important in the beginning of the training. We do not recommend using ReLU – if the ODE gradients are too big, the ODE solver might fail to solve an ODE with the specified tolerance.

Loss We use negative gaussian log-likelihood with fixed variance as a reconstruction loss. We used fixed variance of 0.001 for MuJoCo and 0.01 for other datasets. We report MSE on the time series in the test set. For classification task, we use cross-entropy loss.
On Physionet dataset, we got best performance by training reconstruction loss and cross-entropy loss together, with multiplier 100 on cross-entropy loss. We train on the whole dataset of 8000 patients and compute the CE loss only on labeled samples (4000 patients).
On Human Activity dataset, we trained solely with cross-entropy loss. Note that on Physionet labels are provided per time series, and reconstruction loss prevents overfitting on classification task. On Human Activity dataset, the labels are provided per time point and training with CE loss only does not lead to overfitting.

ODE Solver We used ODE Solvers from torchdiffeq python package. We used fifth-order "dopri5" solver with adaptive step for generative model of Latent ODE. In ODE-RNN we solve ODE within small time intervals, and there is no benefit of using sophisticated solver. For the sake of speed, we use a simple Euler solver for ODE-RNN. We used relative tolerance of 1e-3 and absolute tolerance of 1e-4.

RNN Baselines We follow the available implementation of RNN GRU-D: `https://github.com/zhiyongc/GRU-D/blob/master/GRUD.py`. We use exponential decay between hidden states and imputation technique from GRU-D as separate baselines.

5 Hyperparameters

Choosing hyperparameters First, we choose the hyperparameters such that delivered the best performance for the *RNN baselines* in our experiments. Then we run corresponding ODE models with the same hyperparameters. In autoregressive models we use the same size of hidden state, number of layers and units in encoder and decoder networks, etc. in both ODE-RNN and RNN baselines.
Matching the hyperparameters between encoder-decoder and autoregressive models is more tricky, since encoder-decoder perform a harder task of reconstructing the whole time series (rather than doing one-step-ahead prediction) and require more parameters. We decided that it is fair to match the size of the generative model in encoder-decoder models to the size of hidden state of autoregressive models.

Hyperparameters for Latent ODE model We empirically found that ODE of the same or slightly smaller dimensionality as the data works best for generative model in Latent ODE. For instance, we used 15-dimensional latent state for 14-dimensional Mujoco dataset. For 37-dimensional physionet dataset we used 30 latent dimensions in the generative model. As a rule-of-thumb for Latent ODE model, we used a recognition model with twice as many dimensions in the latent states in Latent ODE model. To compute ELBO, we use three samples from distribution $\mathcal{N}(\mu_{z_0}, \sigma_{z_0})$.

Toy dataset To model 1-dimensional toy dataset, we used 10 latent dimensions in the generative model and 20 in recognition model, batch size of 50. ODE function for both generative and recognition models consist of 1 hidden layer with 100 units.

MuJoCo For 14-dimensional MuJoCo dataset we used 15 latent dimensions in generative model, 30 dimensions in recognition model, batch size of 50. ODE functions had 3 layers and 500 units.

Physionet For Physionet (37 dims), we used 30 latent dimensions in the generative model, 40 dimensions in recognition model and batch size of 100. ODE function have 3 layers with 500 units.

Fitting Poisson process on Physionet dataset For fitting Poisson process along with the likelihood together with reconstruction likelihood, our generative model had the following dimensions: 30 for modelling $z(t)$ (for reconstruction), 30 for modelling z_λ (for poisson process) and 37 for $\int \lambda_0^t(\tau)d\tau$ (equals data dimensionality). Hence, 97 dimensions in total. ODE function in recognition model (for computing derivative of $z(t)$ and z_λ jointly) has 2 layers and 1000 units in each. Recognition model has 100 dimensions, and ODE function has 4 layers, 1000 units in each.
To create plots in fig. 6 we used five first dimensions of Physionet. For the sake of visualization, we trained the model with 2 latent dimensions in generative model and 10 dimensions in recognition model. We reduced the size of ODE network to 100 units.

Classification on Physionet In classification task, the goal is to classify each patient with a binary label. We used "in-hospital mortality" label from the original dataset. Since every patient (e.g. every training example) has hundreds of observations across all feature types, it is very easy to overfit. Previous works, e.g. Cao et al, (2018), also mention issues with overfitting. To prevent overfitting, we train the model together with reconstruction loss with coefficient 100 on cross-entropy loss. We compute reconstruction loss on all 8000 patients, and cross-entropy loss on 4000 labeled patients. We find that all models, including RNN GRU-D require careful hyper-parameter tuning on this task. We used a 2-layer classifier with 300 units and ReLU activations.
For each model, we achieved the best results with the following parameters:
ODE-RNN We used 15-dimensional hidden states, batch size 100 and learning rate 0.01. For ODE function, we used a 3-layer neural net with 100 units and Tanh activation.
GRU-D We used 60-dimensional hidden states, batch size 100 and learning rate 0.001.

Classification on Human Activity On this dataset, we chose the hyperparameteres of the models separately to achieve best performance for each of them. We used a linear classifier on hidden states h_i.
ODE-RNN We used 30-dimensional hidden state, batch size 30 and learning rate 0.01. For ODE function, we used a neural net with four hidden layers and 1000 units.
GRU-D We used 50-dimensional latent states, batch size 30, learning rate 0.01.
Latent ODE (ODE enc.) In the generative model, we used 10-dimensional hidden states, 2-layer neural nets with 500 units. In recognition model, we used 100-dimensional hidden state, 4-layer neural net with 500 units. We used batch size of 100 and learning rate 0.01.

6 Training details

We used Adamax optimizer with the learning rate of 0.01 for toy dataset and MuJoCo experiments with learning rate decay to 0.01. We used 0.0005 for Physionet and 0.005 for PersonActivity. We use a small learning rate decay rate of 0.999. We use KL annealing for VAE models with coefficient 0.993. We train all models for 300 epochs.

6.1 Run time

In the proposed Latent ODE the encoder matches ODE-RNN architecture, while the generative model requires solving an ODE in the time interval of interest. Depending on the smoothness of the learned ODE,

the time for solving this ODE can vary, however we find that Latent ODE has comparable runtime to RNN-VAE.
Empirically, ODE-RNN and standard RNN took the approximately the same time to train.
VAE models took more epochs to train than autoregressive models. This is expected, since autoregressive models only predicts one-step-ahead, while VAE models need to reconstruct the whole trajectories given the vector summary z_0.

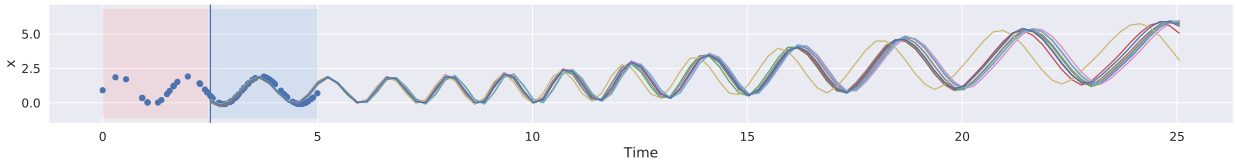
6.2 Computing infrastructure

All experiments were run on one Nvidia P100 with 2 physical Intel Xeon(R) Silver 4110 CPU.

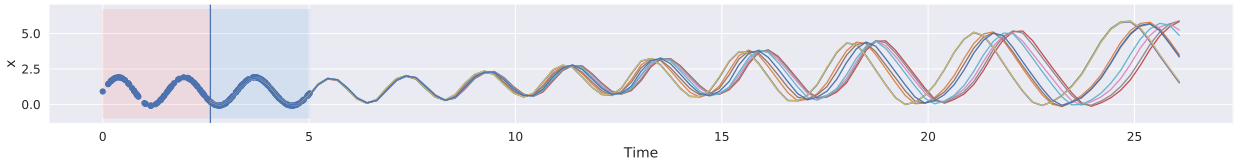
6.3 Source code and datasets

The code for generating and pre-processing of the datasets is included with the submission. The model is implemented in PyTorch 1.0.
We used the ODE solvers from torchdiffeq package (`https://github.com/rtqichen/torchdiffeq`)
Datasets:
Human Activity: `https://archive.ics.uci.edu/ml/datasets/Localization+Data+for+Person+Activity`
Physionet: `https://physionet.org/physiobank/database/challenge/2012/`
MuJoCo dataset was created using DeepMind Control Suite: `https://github.com/deepmind/dm_control`.
The link to the downloadable to the dataset will be provided in camera-ready version.

Supplementary figures and tables

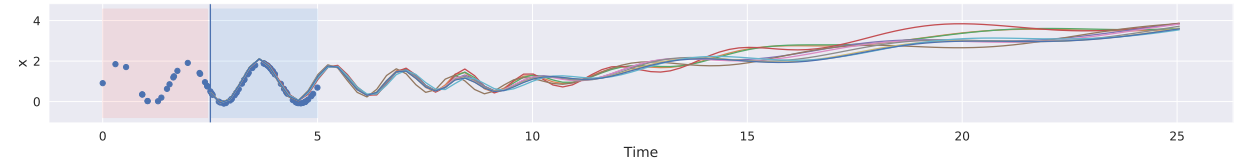


(a) Latent ODE with ODE encoder (ours): conditioned on 20 points

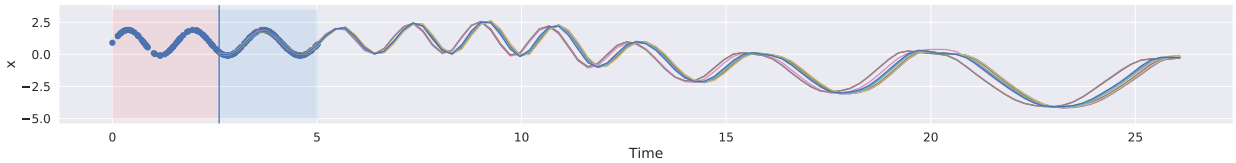


(b) Latent ODE with ODE encoder (ours): conditioned on 80 points

Figure 1

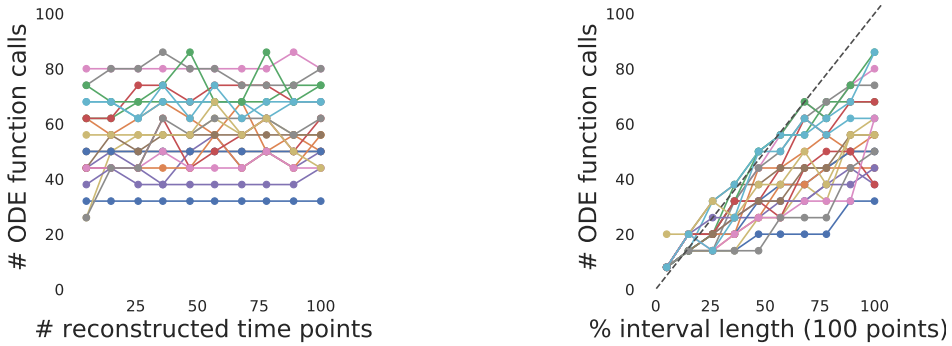


(a) Latent ODE with RNN encoder (Chen et al. 2018): conditioned on 20 points



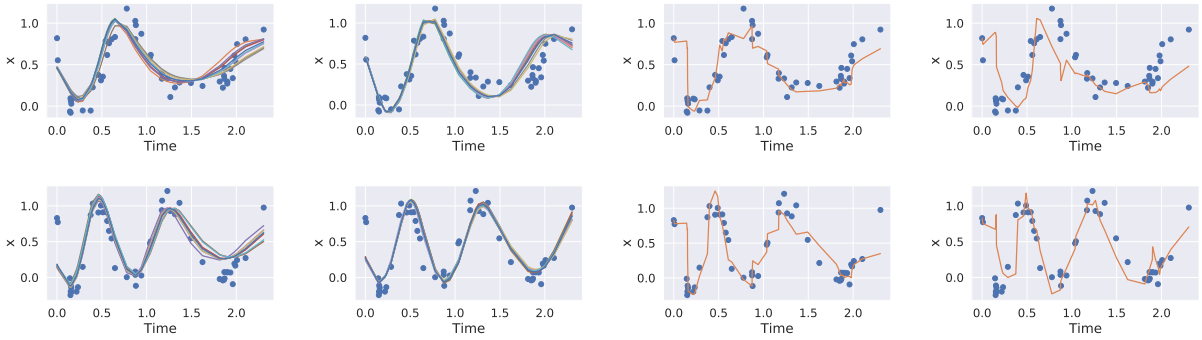
(b) Latent ODE with RNN encoder (Chen et al. 2018): conditioned on 80 points

Figure 2



(a) (b)

Figure 3: (a) Number of evaluations of ODE function f does not depend on the number of time points $(t_0..t_N)$ where we evaluate ODE solution (b) Instead, number of ODE function evaluations depends on the length of the time interval $[t_0..t_N]$ where ODE is solved.
For plot (a) we randomly subsampled time series in MuJoCo dataset by variable number of points and computed number of function evaluations required by ODESolve in each case. For plot (b) we truncated the time series to $(t_0..t_i)$ and computed number of ODE function evaluations for different i. Each line shows a number of ODE func evals for a single time series from MuJoCo dataset.

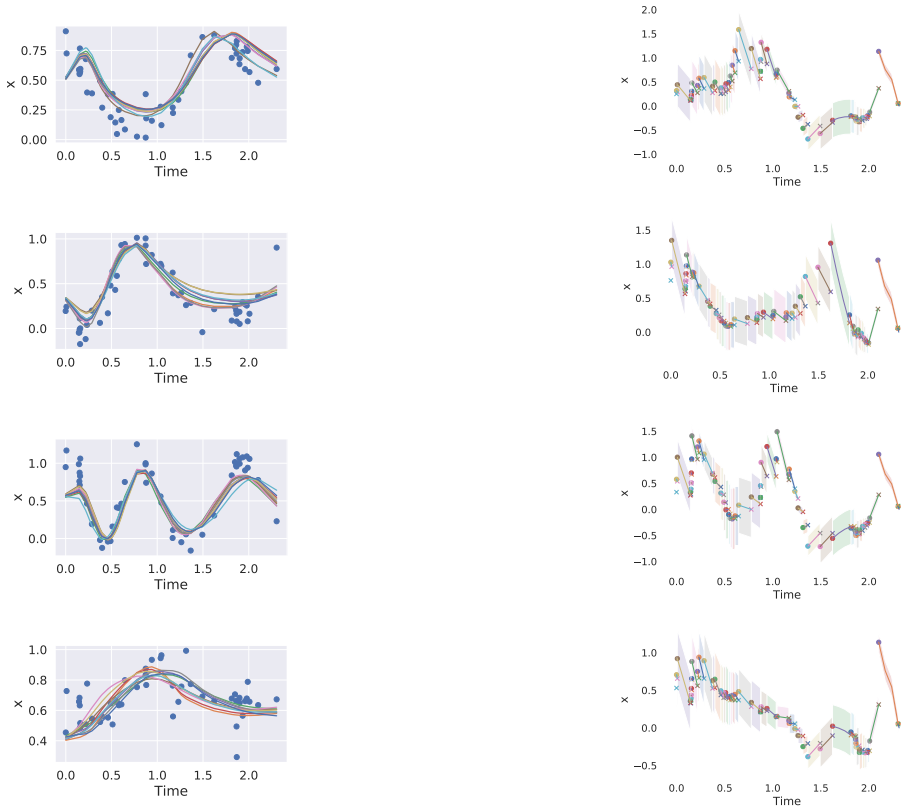


(a) Latent ODE (ODE enc) (b) Latent ODE (RNN enc) (c) ODE-RNN (d) Standard RNN

Figure 4: Reconstructions on toy dataset

Table 2: Mean squared error on the toy dataset

		Interpolation				Extrapolation			
	% observed points	10	20	30	50	10	20	30	50
Autoreg	RNN Δt	0.06081	0.04680	0.05822	0.04116	0.06172	0.06115	0.06891	0.05617
	RNN-imputed	0.08558	0.06043	0.03922	0.04116	**0.06095**	0.07212	**0.06541**	**0.05049**
	RNN-exp	1.65891	0.05344	0.04974	0.03275	0.06172	0.06115	0.06891	0.05617
	RNN GRU-D	2.35628	0.05997	0.04832	0.04116	**0.06095**	0.07212	**0.06541**	**0.05049**
	ODE-RNN	**0.05150**	**0.03211**	**0.02643**	**0.01666**	0.06592	**0.04774**	0.10940	0.08000
VAE	RNN-VAE	0.07352	0.07346	0.07323	0.07304	0.20107	0.03710	0.07281	0.02871
	Latent ODE (RNN enc)	**0.06860**	0.06764	**0.02754**	0.05721	**0.04920**	0.04807	**0.01788**	0.02703
	Latent ODE (ODE enc)	0.07133	**0.03144**	0.05354	**0.01717**	0.05313	**0.04427**	0.03572	**0.01388**



(a) (b)

Figure 5: **(a)** Reconstructions on toy dataset from Latent ODE. Observations are shown as points. Lines are reconstructions for different samples of z_0 in Latent ODE model. **(b)** Corresponding latent state in the recognition model (first dimension). The recognition model encodes the data backwards in time (from right to left). The lines show latent ODE path in-between the encoded data points. The discontinuities between the paths show the update of the latent state using the observation at that time point. The end of the end of the ODE path from the previous observation is shown as a small circle. The updated state (and the start a new ODE path) is shown as cross. The shaded area shows the predicted standard deviation for the initial latent state z_0. Notice that the right-most point is similar for all four trajectories – only one data point was encoded, which does not contain much information about the trajectory. As the encoding progresses (from right to left), the latent updated generally become smaller.

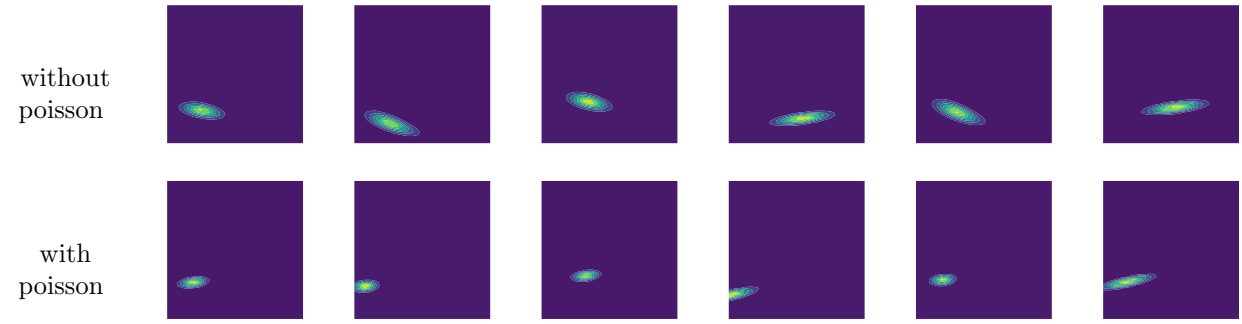


Figure 6: True posterior $p(z_0|x_1..x_N)$ of Latent ODE trained on Physionet dataset with five features and two-dimensional latent space with and without poisson process likelihood. We train the model with 2-dimensional latent space on a subset of first five attributes. We compute the unnormalized density of the true posterior using the Bayes rule: $p(z_0|x_1..x_N) \propto p(z_0)p(x_1..x_N|z_0)$. Similarly, we train the model with poisson process likelihood in the same manner. The posterior distribution is clearly more narrow if trained without poisson

Figure 7: Reconstructed trajectories on Mujoco dataset

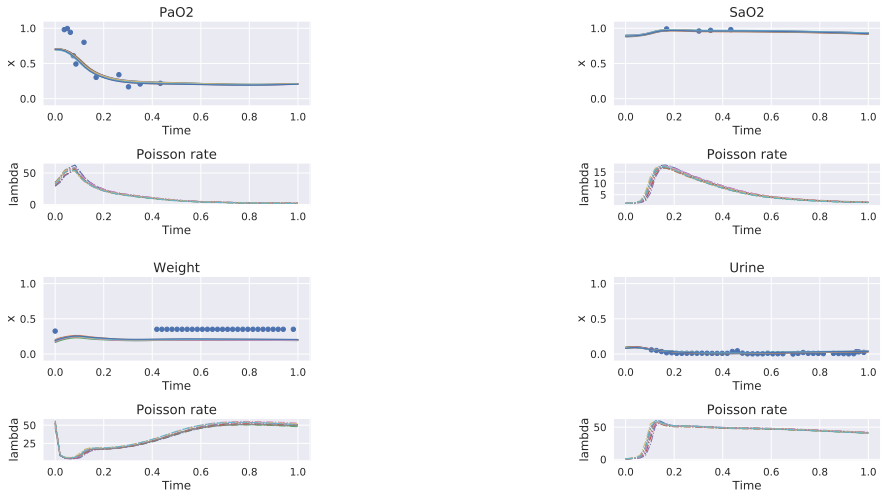


Figure 8: Examples of learned poisson rate for different features in Physionet.