VRNN

Author: RuifMaxx

Personal website: ruifmaxx.github.io

RNN background

Shortcoming of RNN

The only source of randomness or variability in the RNN is found in the conditional output probability model.

We suggest that this can be an inappropriate way to model the kind of variability observed in highly structured data, such as natural speech, which is characterized by strong and complex dependencies among the output variables at different timesteps.

Structure of RNN

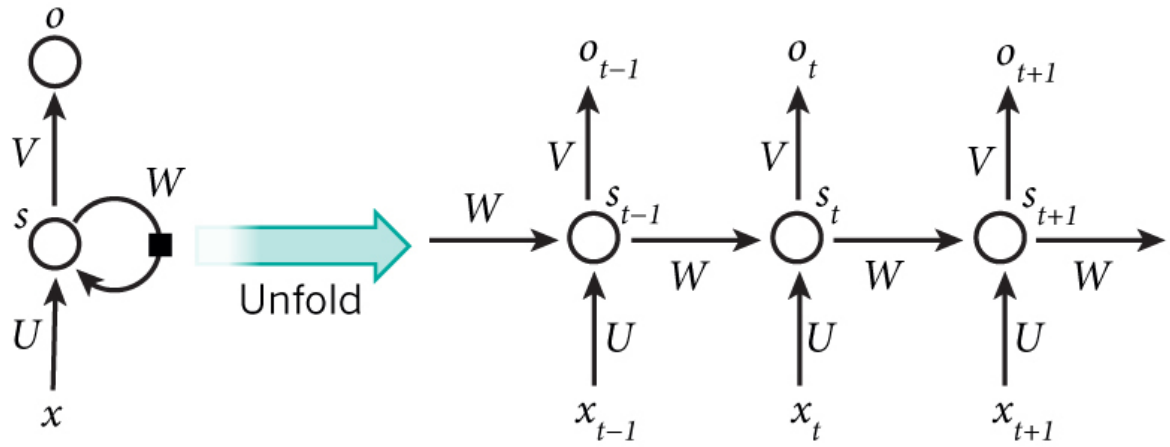
updates its hidden state *ht* by:

$$\mathbf{h}_t = f_\theta\left(\mathbf{x}_t, \mathbf{h}_{t-1}\right)$$

where *f* is a deterministic non-linear transition function, and θ is the parameter set of *f*. The transition function *f* can be implemented with gated activation functions such as long short-term memory [LSTM, 9] or gated recurrent unit [GRU, 5]. RNNs model sequences by parameterizing a factorization of the joint sequence probability distribution as a product of conditional probabilities such that:

$$p\left(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\right) = \prod_{t=1}^{T} p\left(\mathbf{x}_t \mid \mathbf{x}_{<t}\right)$$
$$p\left(\mathbf{x}_t \mid \mathbf{x}_{<t}\right) = g_\tau\left(\mathbf{h}_{t-1}\right)$$

where *g* is a function that maps the RNN hidden state *ht-1* to a probability distribution over possible outputs, and τ is the parameter set of *g*.



With a deterministic transition function f, the choice of g effectively defines the family of joint probability distributions p(x1, . . . , xT) that can be expressed by the RNN

GMM

高斯混合模型是对高斯模型进行简单的扩展，GMM使用多个高斯分布的组合来刻画数据分布。

举例来说：想象下现在咱们不再考察全部用户的身高，而是要在模型中同时考虑男性和女性的身高。假定之前的样本里男女都有，那么之前所画的高斯分布其实是两个高斯分布的叠加的结果。相比只使用一个高斯来建模，现在我们可以用两个（或多个）高斯分布（陈运文）：

$$p(x) = \sum_{i=1}^{K} \phi_i \frac{1}{\sqrt{2\sigma_i^2 \pi}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}$$

该公式和之前的公式非常相似，细节上有几点差异。首先分布概率是K个高斯分布的和，每个高斯分布有属于自己的 μ 和 σ 参数，以及对应的权重参数，权重值必须为正数，所有权重的和必须等于1，以确保公式给出数值是合理的概率密度值。换句话说如果我们把该公式对应的输入空间合并起来，结果将等于1。

回到之前的例子，女性在身高分布上通常要比男性矮，画成图的话如图3

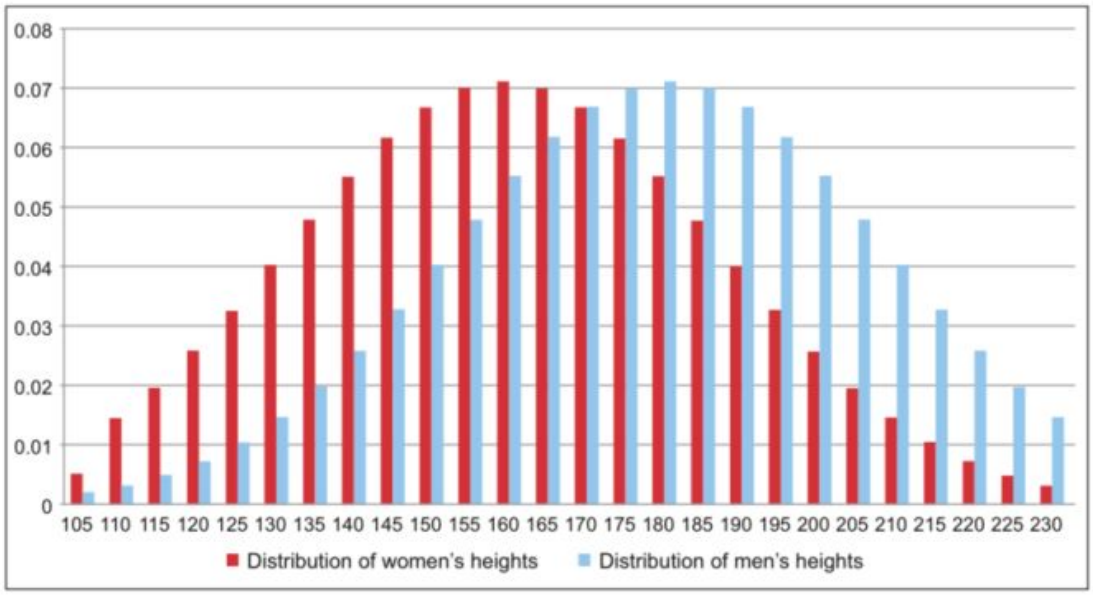


Figure 2.11 Probability distributions of height for men and women. Note that these probabilities are conditioned on the fact that gender is known: so, for example, given that we know a particular person is a woman, her probability of having a height in a particular bucket can be read off the y-axis.

(https://zhuanlan.zhihu.com/p/31103654)

$$p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T) = \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{x}_{<t})$$
$$p(\mathbf{x}_t \mid \mathbf{x}_{<t}) = g_\tau(\mathbf{h}_{t-1})$$

We can express the output function g in Eq. (2) as being composed of two parts. The first part φ_τ is a function that returns the parameter set ϕ_t given the hidden state $\mathbf{h}_{t-1}$, i.e., $\phi_t = \varphi_\tau(\mathbf{h}_{t-1})$, while the second part of g returns the density of $\mathbf{x}_t$, i.e., $p_{\phi_t}(\mathbf{x}_t \mid \mathbf{x}_{<t})$.

When modelling high-dimensional and real-valued sequences, a reasonable choice of an observation model is a Gaussian mixture model (GMM) as used in [7]. For GMM, φ_τ returns a set of mixture coefficients α_t, means $\boldsymbol{\mu}_{\cdot,t}$ and covariances $\Sigma_{\cdot,t}$ of the corresponding mixture components. The probability of $\mathbf{x}_t$ under the mixture distribution is:

$$p_{\boldsymbol{\alpha}_t,\boldsymbol{\mu}_{\cdot,t},\Sigma_{\cdot,t}}(\mathbf{x}_t \mid \mathbf{x}_{<t}) = \sum_j \alpha_{j,t}\mathcal{N}(\mathbf{x}_t; \boldsymbol{\mu}_{j,t}, \Sigma_{j,t})$$

(https://arxiv.org/abs/1506.02216)

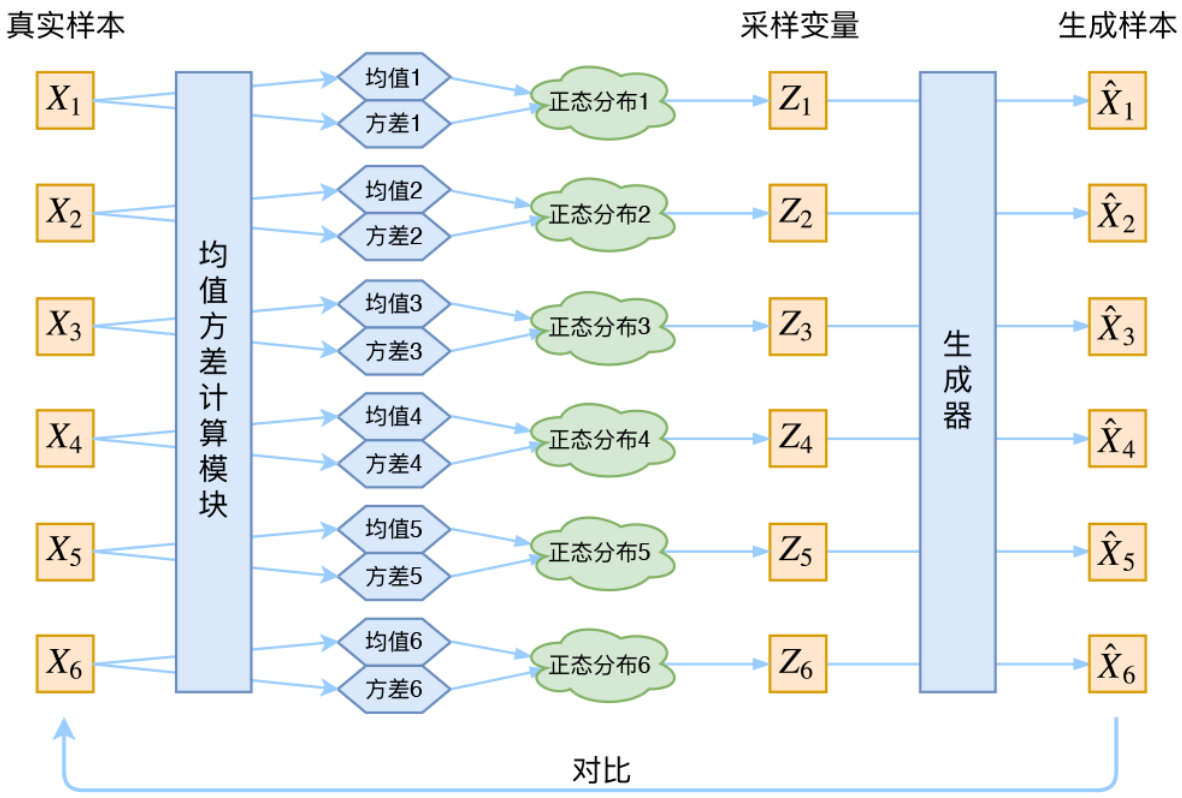
VAE background

首先我们有一批数据样本{X1,...,Xn}，其整体用X来描述，我们本想根据{X1,...,Xn}得到X的分布p(X)，如果能得到的话，那我直接根据p(X)来采样，就可以得到所有可能的X了（包括{X1,...,Xn}以外的），这是一个终极理想的生成模型了。当然，这个理想很难实现，于是我们将分布改一改

p(X)=∑ p(X|Z)p(Z)(1)

这里我们就不区分求和还是求积分了，意思对了就行。此时p(X|Z)就描述了一个由Z来生成X的模型，而我们假设Z服从标准正态分布，也就是p(Z)=N(0,I)

(变分自编码器（一）：原来是这么一回事 - 科学空间|Scientific Spaces)



$$\begin{aligned}\log p(x) &= \log \int p(x \mid z)p(z)dz \\ &= \log \int \frac{p(x|z)p(z)}{q(z|x)} q(z \mid x)dz \\ &= \log \mathbb{E}_{q(z|x)}\left[\frac{p(x|z)p(z)}{q(z|x)}\right]\end{aligned}$$

根据 Jenson 不等式

$$\log p(x) \geq \mathbb{E}_{q(z|x)} \log \left[\frac{p(x \mid z)p(z)}{q(z \mid x)} \right]$$

得到变分下界：

$$E_{q(z|x)} \log p(x \mid z) - KL(q(z \mid x) \| p(z))$$

其中第一项用可用对数正态分布表示为：

$$\log \left(\frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\mu)^2}{2\sigma^2}} \right)$$

写成代码是：

```
def log_normal_pdf(x, mean, logvar):
    const = torch.from_numpy(np.array([2. * np.pi])).float().to(x.device)
    const = torch.log(const)
    return -.5 * ( + logvar + (x - mean) ** 2. / torch.exp(logvar))

logpx = log_normal_pdf(
                samp_trajs, pred_x, noise_logvar).sum(-1).sum(-1)
```

当X满足伯努利分布时：

由极大似然估计的定义可知：

$$\begin{aligned}
\theta &= \underset{\theta}{\operatorname{argmax}}\, p(X \mid \theta) \\
&= \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{N} p\left(x^{(i)} \mid \theta\right) \\
&= \underset{\theta}{\operatorname{argmax}} \frac{1}{N} \prod_{i=1}^{N} p\left(x^{(i)} \mid \theta\right) \\
&= \underset{\theta}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^{N} \log p\left(x^{(i)} \mid \theta\right)
\end{aligned}$$

由于样本X由

$$p_{\text{data}}(x)$$

采集而来，因此等同于：

$$\begin{aligned}
&= \underset{\theta}{\operatorname{argmax}}\, E_{x \sim p_{\text{data}}(x)} \log p(x \mid \theta) \\
&= \underset{\theta}{\operatorname{argmax}} \int_x p_{\text{data}}(x) \log p(x \mid \theta) dx \\
&= \underset{\theta}{\operatorname{argmin}} \int_x -p_{\text{data}}(x) \log p(x \mid \theta) dx
\end{aligned}$$

上式就是交叉熵损失，所以极大似然与最小化交叉熵损失此时等价。

(极大似然估计与最小化交叉熵损失或者KL散度为什么等价？ - 知乎 (zhihu.com))

下图为keras的代码

(https://keras.io/examples/generative/vae/)

```
def train_step(self, data):
    if isinstance(data, tuple):
        data = data[0]
    with tf.GradientTape() as tape:
        z_mean, z_log_var, z = encoder(data)
        reconstruction = decoder(z)
        reconstruction_loss = tf.reduce_mean(
            keras.losses.binary_crossentropy(data, reconstruction)
        )
        reconstruction_loss *= 28 * 28
        kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
        kl_loss = tf.reduce_mean(kl_loss)
        kl_loss *= -0.5
        total_loss = reconstruction_loss + kl_loss
    grads = tape.gradient(total_loss, self.trainable_weights)
    self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
    return {
        "loss": total_loss,
        "reconstruction_loss": reconstruction_loss,
        "kl_loss": kl_loss,
    }
```

VRNN

Unlike a standard VAE, the prior on the latent random variable is no longer a standard Gaussian distribution, but follows the distribution:

Z的先验分布：

$$p\left(\mathbf{z}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{<t}\right)$$

$$\mathbf{z}_t \sim \mathcal{N}\left(\boldsymbol{\mu}_{0,t}, \operatorname{diag}\left(\boldsymbol{\sigma}_{0,t}^2\right)\right), \text{ where } \left[\boldsymbol{\mu}_{0,t}, \boldsymbol{\sigma}_{0,t}\right] = \varphi_\tau^{\text{prior}}\left(\mathbf{h}_{t-1}\right)$$

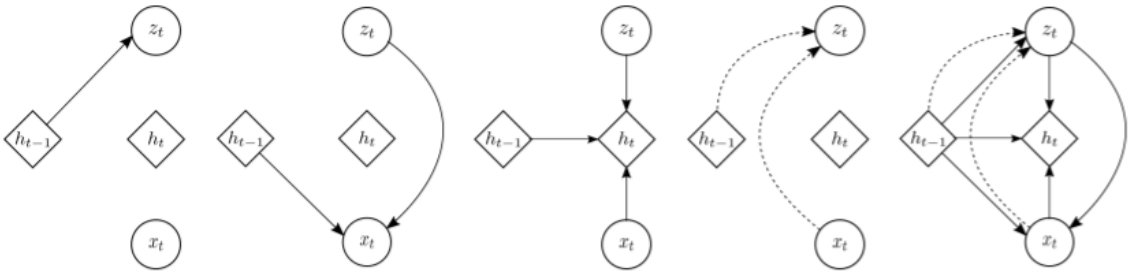
生成量X的分布：

$$p\left(\mathbf{x}_t \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t}\right)$$

$$\mathbf{x}_t \mid \mathbf{z}_t \sim \mathcal{N}\left(\boldsymbol{\mu}_{x,t}, \operatorname{diag}\left(\boldsymbol{\sigma}_{x,t}^2\right)\right), \text{ where } \left[\boldsymbol{\mu}_{x,t}, \boldsymbol{\sigma}_{x,t}\right] = \varphi_\tau^{\text{dec}}\left(\varphi_\tau^{\mathbf{z}}\left(\mathbf{z}_t\right), \mathbf{h}_{t-1}\right)$$

RNN更新：

$$\mathbf{h}_t = f_\theta\left(\varphi_\tau^{\mathbf{x}}\left(\mathbf{x}_t\right), \varphi_\tau^{\mathbf{z}}\left(\mathbf{z}_t\right), \mathbf{h}_{t-1}\right)$$



(a) Prior (b) Generation (c) Recurrence (d) Inference (e) Overall

Z的后验分布：

$$q\left(\mathbf{z}_{\leq T} \mid \mathbf{x}_{\leq T}\right)=\prod_{t=1}^{T} q\left(\mathbf{z}_{t} \mid \mathbf{x}_{\leq t}, \mathbf{z}_{<t}\right)$$

$$\mathbf{z}_{t} \mid \mathbf{x}_{t} \sim \mathcal{N}\left(\boldsymbol{\mu}_{z,t}, \operatorname{diag}\left(\boldsymbol{\sigma}_{z,t}^{2}\right)\right), \text{ where } \left[\boldsymbol{\mu}_{z,t}, \boldsymbol{\sigma}_{z,t}\right]=\varphi_{\tau}^{\mathrm{enc}}\left(\varphi_{\tau}^{\mathbf{x}}\left(\mathbf{x}_{t}\right), \mathbf{h}_{t-1}\right)$$

变分下界：

$$\mathbb{E}_{q(\mathbf{z}\leq T|\mathbf{x}\leq T)}\left[\sum_{t=1}^{T}\left(-\operatorname{KL}(q\left(\mathbf{z}_{t} \mid \mathbf{x}_{\leq t}, \mathbf{z}_{<t}\right) \| p\left(\mathbf{z}_{t} \mid \mathbf{x}_{<t}, \mathbf{z}_{<t}\right)\right)+\log p\left(\mathbf{x}_{t} \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t}\right)\right)\right]$$

优点：为RNN带来随机性，利用生成模型隐变量Z很好地反映X

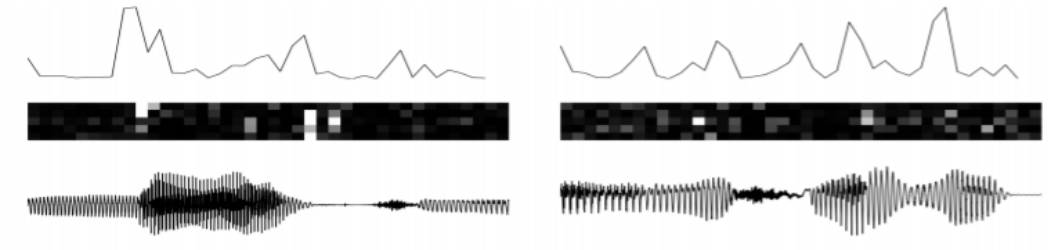


Figure 2: The top row represents the difference δ_t between $\boldsymbol{\mu}_{z,t}$ and $\boldsymbol{\mu}_{z,t-1}$. The middle row shows the dominant KL divergence values in temporal order. The bottom row shows the input waveforms.

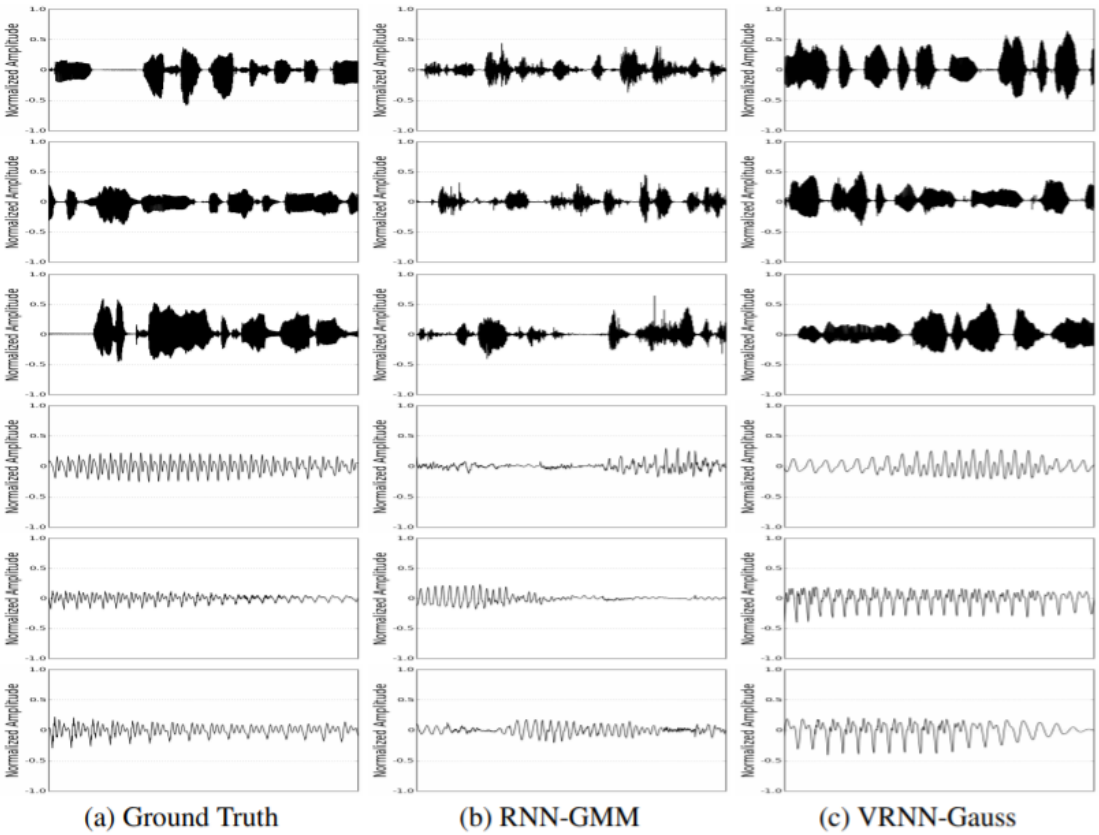


Figure 3: Examples from the training set and generated samples from RNN-GMM and VRNN-Gauss. Top three rows show the global waveforms while the bottom three rows show more zoomed-in waveforms. Samples from (b) RNN-GMM contain high-frequency noise, and samples from (c) VRNN-Gauss have less noise. We exclude RNN-Gauss, because the samples are almost close to pure noise.

Neural ODE

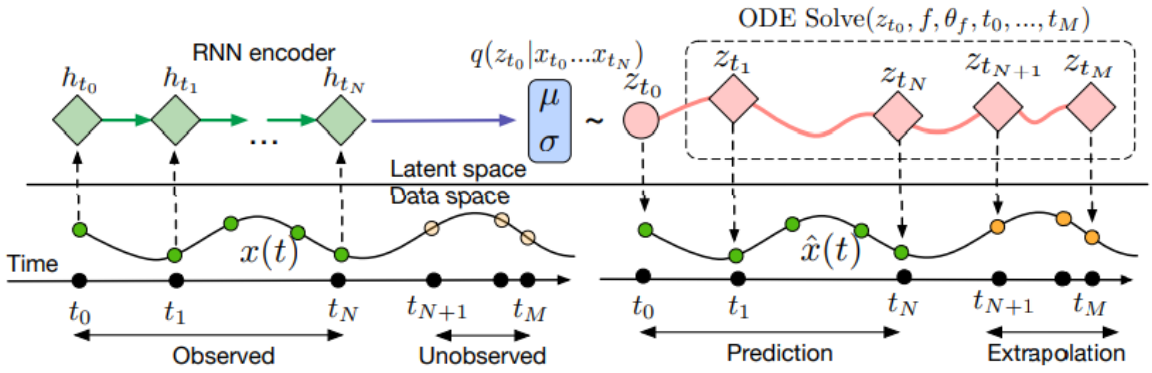


Figure 6: Computation graph of the latent ODE model.

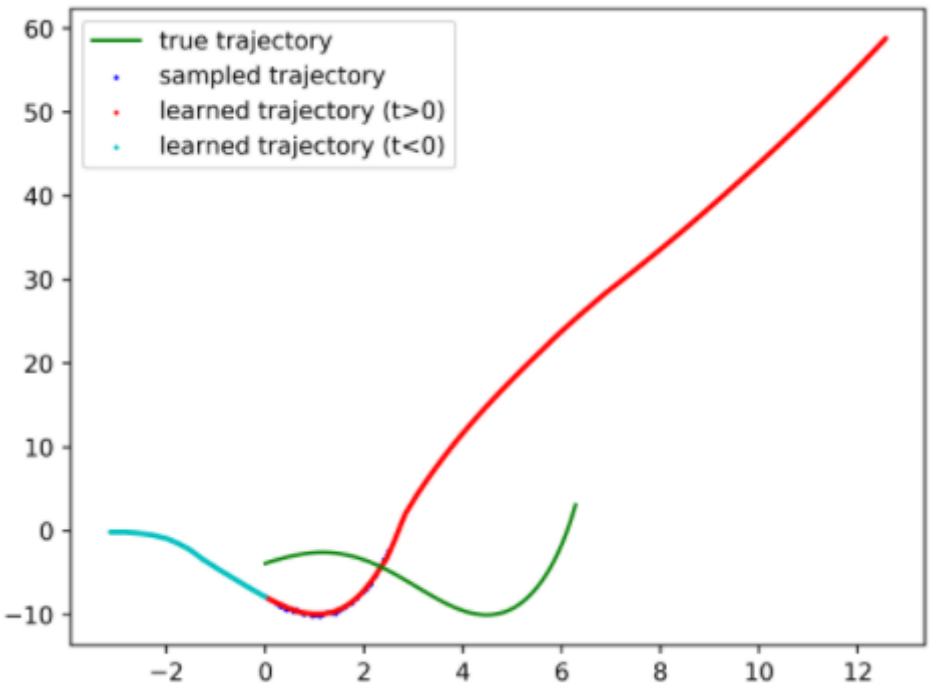
To obtain the latent representation $\mathbf{z}_{t_0}$, we traverse the sequence using RNN and obtain parameters of distribution $q(\mathbf{z}_{t_0}|\{\mathbf{x}_{t_i}, t_i\}_i, \theta_{enc})$. The algorithm follows a standard VAE algorithm with an RNN variational posterior and an ODESolve model:

1. Run an RNN encoder through the time series and infer the parameters for a posterior over $\mathbf{z}_{t_0}$:
$$q(\mathbf{z}_{t_0}|\{\mathbf{x}_{t_i}, t_i\}_i, \phi) = \mathcal{N}(\mathbf{z}_{t_0}|\mu_{\mathbf{z}_{t_0}}, \sigma_{\mathbf{z}_0}), \tag{53}$$
where $\mu_{\mathbf{z}_0}, \sigma_{\mathbf{z}_0}$ comes from hidden state of RNN$(\{\mathbf{x}_{t_i}, t_i\}_i, \phi)$
2. Sample $\mathbf{z}_{t_0} \sim q(\mathbf{z}_{t_0}|\{\mathbf{x}_{t_i}, t_i\}_i)$
3. Obtain $\mathbf{z}_{t_1}, \mathbf{z}_{t_2}, \ldots, \mathbf{z}_{t_M}$ by solving ODE ODESolve$(\mathbf{z}_{t_0}, f, \theta_f, t_0, \ldots, t_M)$, where f is the function defining the gradient $d\mathbf{z}/dt$ as a function of $\mathbf{z}$
4. Maximize ELBO $= \sum_{i=1}^{M} \log p(\mathbf{x}_{t_i}|\mathbf{z}_{t_i}, \theta_{\mathbf{x}}) + \log p(\mathbf{z}_{t_0}) - \log q(\mathbf{z}_{t_0}|\{\mathbf{x}_{t_i}, t_i\}_i, \phi)$, where $p(\mathbf{z}_{t_0}) = \mathcal{N}(0, 1)$

原本库里面是二维的，issues里面有一维的版本

(https://github.com/rtqichen/torchdiffeq/issues/91)

[^]: 所以我花了一些时间来研究代码并找到了主要问题。在修改代码时，您可能已经注意到了这一点。我最初能够对所有轨迹使用相同的样本的主要原因是目标是在（x，y）坐标的2D空间中生成数据。真正重要的是观察之间的时间间隔，时间戳的精确值并不重要。 当目标是准确地将Latent ODE的动态与基本真相的动态一致时，问题就变得稍微复杂了。当然，在这种情况下，您不应该假设批处理中的所有数据项都是从同一开始时间生成的，因为实际上这并不是它的工作方式。因此，更合适的做法是正确地提取所有时间戳，并根据查询到的时间戳对每个条目进行集成。您可能还希望将Latent ODE参数化为时间不均匀。 但是我认为库目前不支持集成批处理时间，因此如果您的数据包含具有不同时间戳的不同条目，最好的办法是将这些条目进行并集，将批中的所有条目集成到联合中的所有时间，然后根据各自的时间戳选择正确的值。 尤利娅的项目在这方面取得了一些进展，她的代码是开源的。



Latent ODE

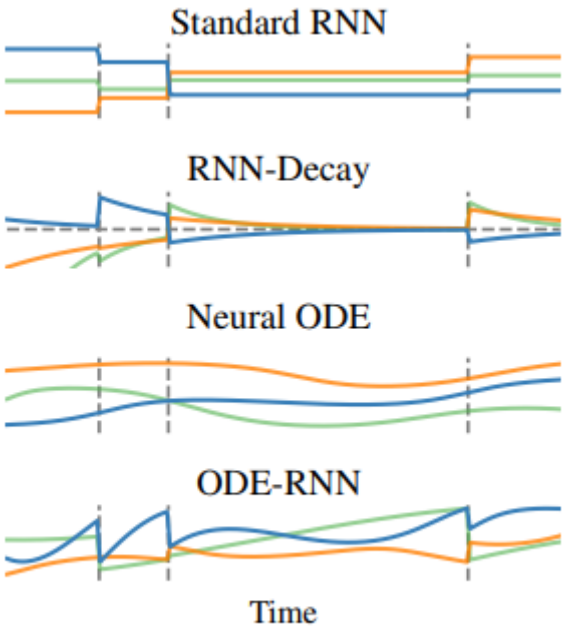
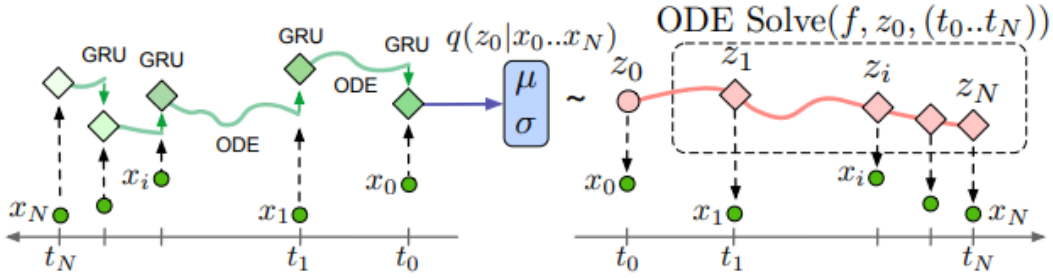


Figure 1: Hidden state trajectories. Ver-



Algorithm 1 The ODE-RNN. The only difference, highlighted in bluc, from standard RNNs is that the pre-activations h' evolve according to an ODE between observations, instead of being fixed.

Input: Data points and their timestamps $\{(x_i, t_i)\}_{i=1..N}$
$h_0 = \mathbf{0}$
for i in 1, 2, . . . , N **do**
  $h'_i = \text{ODESolve}(f_\theta, h_{i-1}, (t_{i-1}, t_i))$ ▷ Solve ODE to get state at t_i
  $h_i = \text{RNNCell}(h'_i, x_i)$ ▷ Update hidden state given current observation x_i
end for
$o_i = \text{OutputNN}(h_i)$ for all $i = 1..N$
Return: $\{o_i\}_{i=1..N}; h_N$

Poisson Process likelihoods The fact that an observation occurred often tells us something about the latent state. For example, a patient may be more likely to take a medical test if they are sick. The rate of events can be parameterized by a function of the latent state: $p(\text{event at time } t | \mathbf{z}(t)) = \lambda(\mathbf{z}(t))$. Given this rate function, the likelihood of a set of independent observation times in the interval $[t_{\text{start}}, t_{\text{end}}]$ is given by an inhomogeneous Poisson process (Palm, 1943):

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

We can parameterize $\lambda(\cdot)$ using another neural network. Conveniently, we can evaluate both the latent trajectory and the Poisson process likelihood together in a single call to an ODE s learned by such a model on a toy dataset.

the joint distribution is defined as:

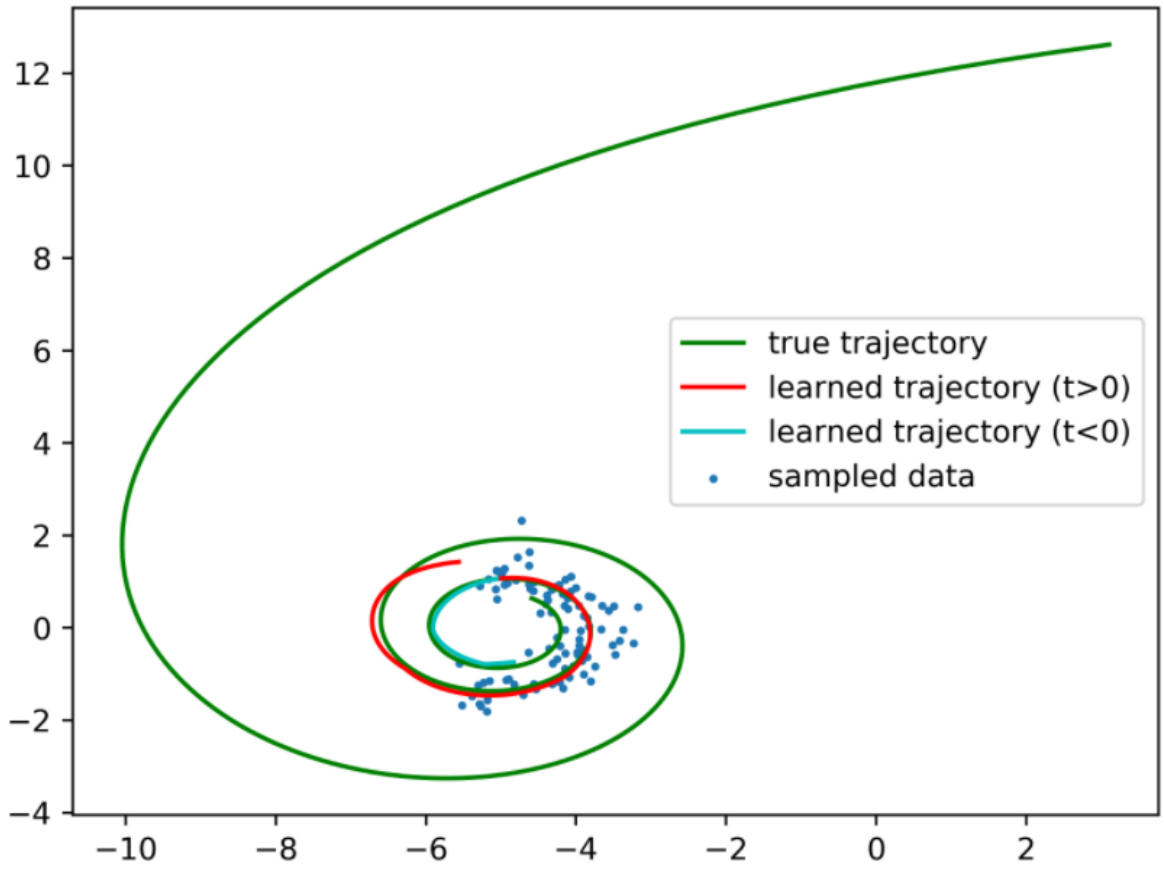
$$p(\mathbf{x}, \mathbf{t}) = p(\mathbf{x} \mid \mathbf{t}) p(\mathbf{t})$$

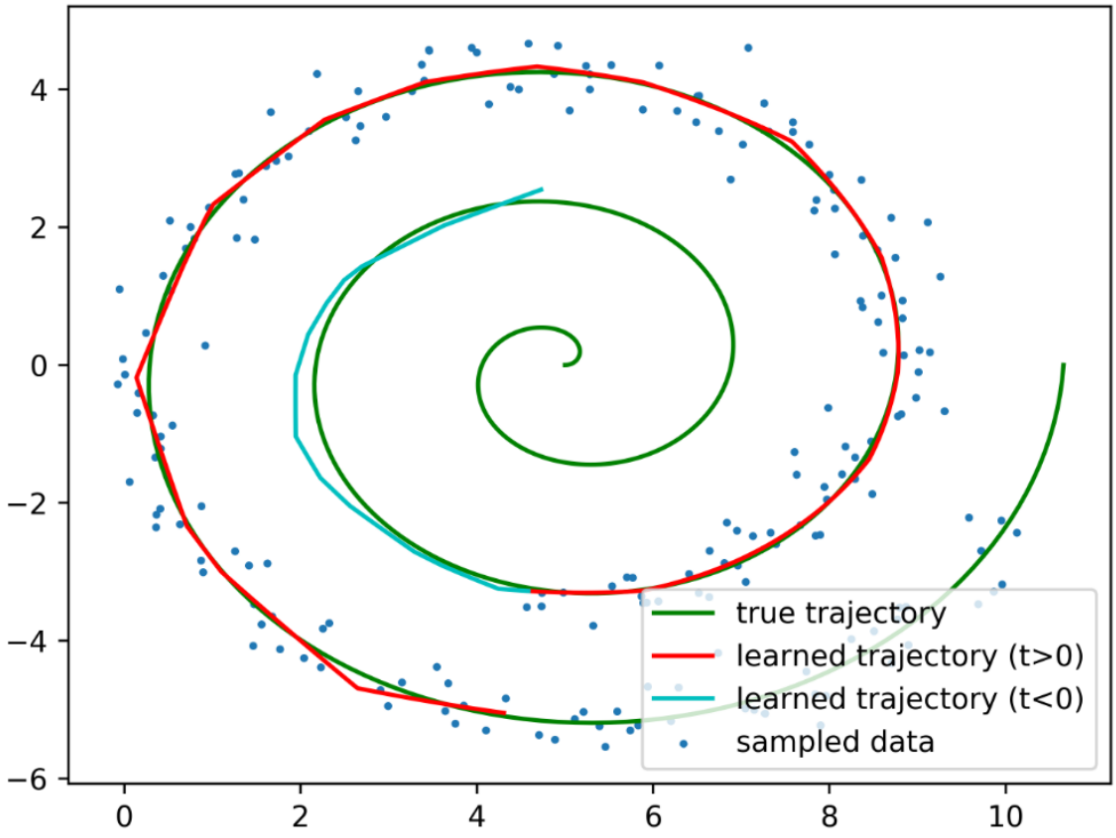
We augment the Latent ODE framework with a Poisson process over the observation times, where we parameterize $\lambda(t)$ as a function of $z(t)$. This means that instead of specifying and maximizing the conditional marginal likelihood $p(x_1, \ldots, x_N | t_1, \ldots, t_N, \theta)$, we can instead specify and maximizing the joint marginal likelihood $p(x_1, \ldots, x_N, t_1, \ldots, t_N, |\theta)$. To compute the joint likelihood, we can evaluate the Poisson intensity $\lambda(t)$, precisely estimate its integral, and the compute latent states at all required time points, using a single call to an ODE solver.

Tricks

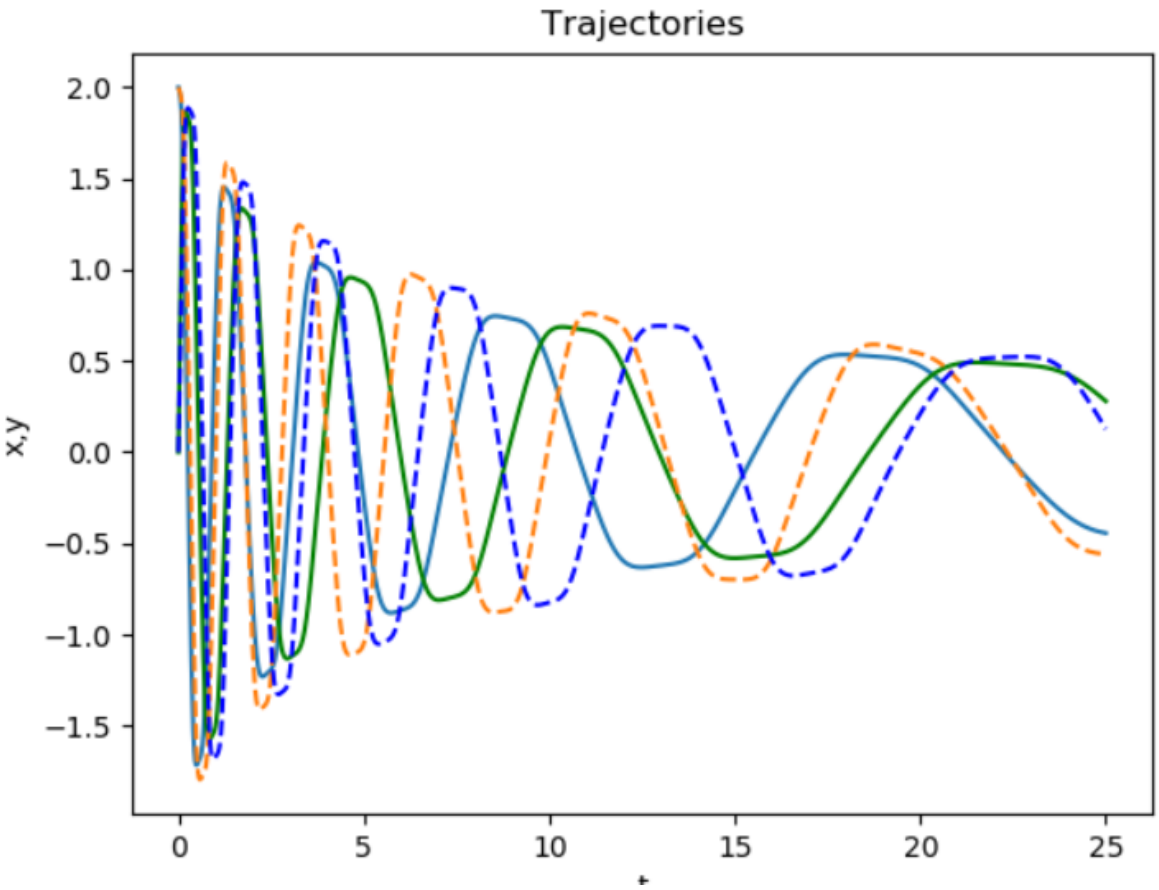
latent_ode

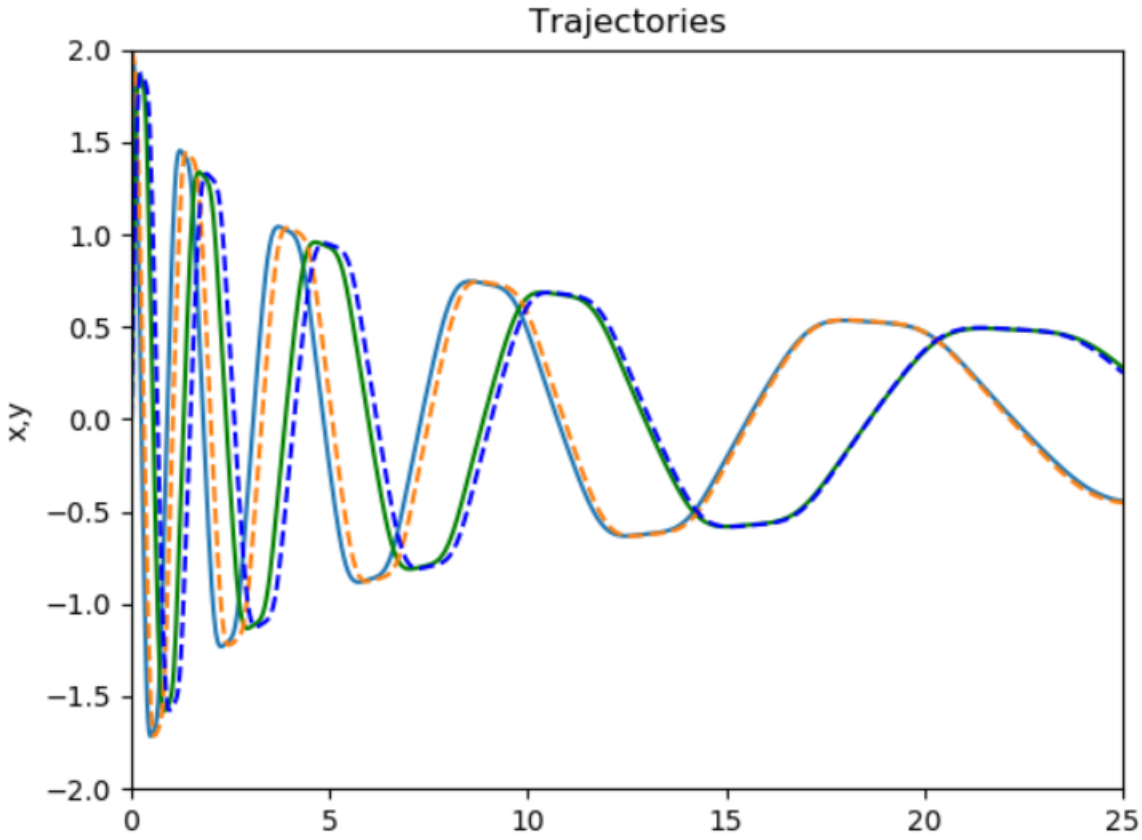
发现原程序的`samp_trajs`和Pull Request里面的这张图片很不一样，就做了修改。





ode-demo参数修改





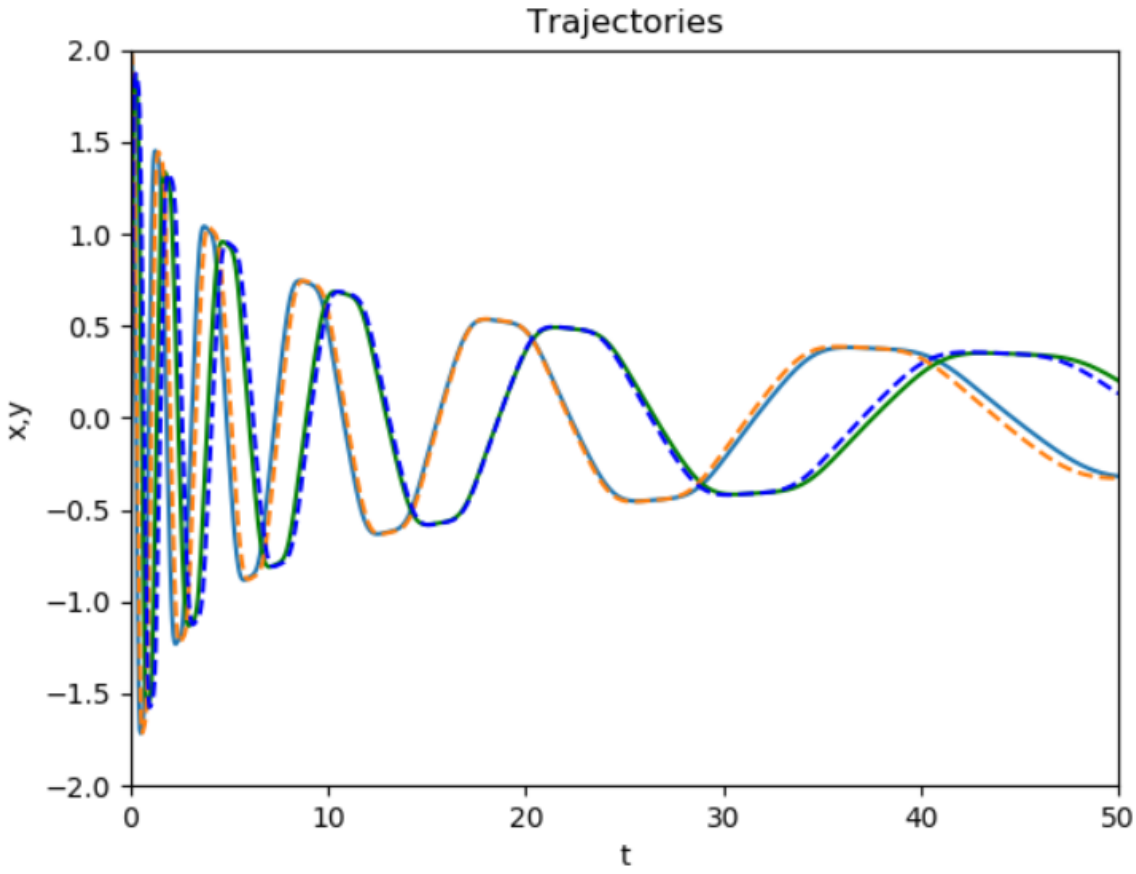
1.扩大batch_size： 发现训练集上面的loss已经降得很小而测试集的loss乱跳，这是由于训练集过小导致的。 尝试扩大batch_time和batch_size，发现单独扩大batch_size效果较好。

2.扩大神经网络的宽度： 发现随着积分次数的增加模型拟合能力变弱，这是神经网络复杂度不够导致的。 尝试扩大神经网络的层数或宽度，发现扩大宽度效果较好。

3.更换优化器，添加正则项： 把优化器从rmsprop换成Adam后精度有明显提高，同时运行速度变快。进一步换成带正则项的AdamW，精度进一步提高。

4.动态调整学习率： 开始时快速拟合曲线形状，之后微调幅度。

5.早停法： 当模型性能不再提高时，停止训练。


Trajectories
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
x,y
0
10
20
30
40
50
t