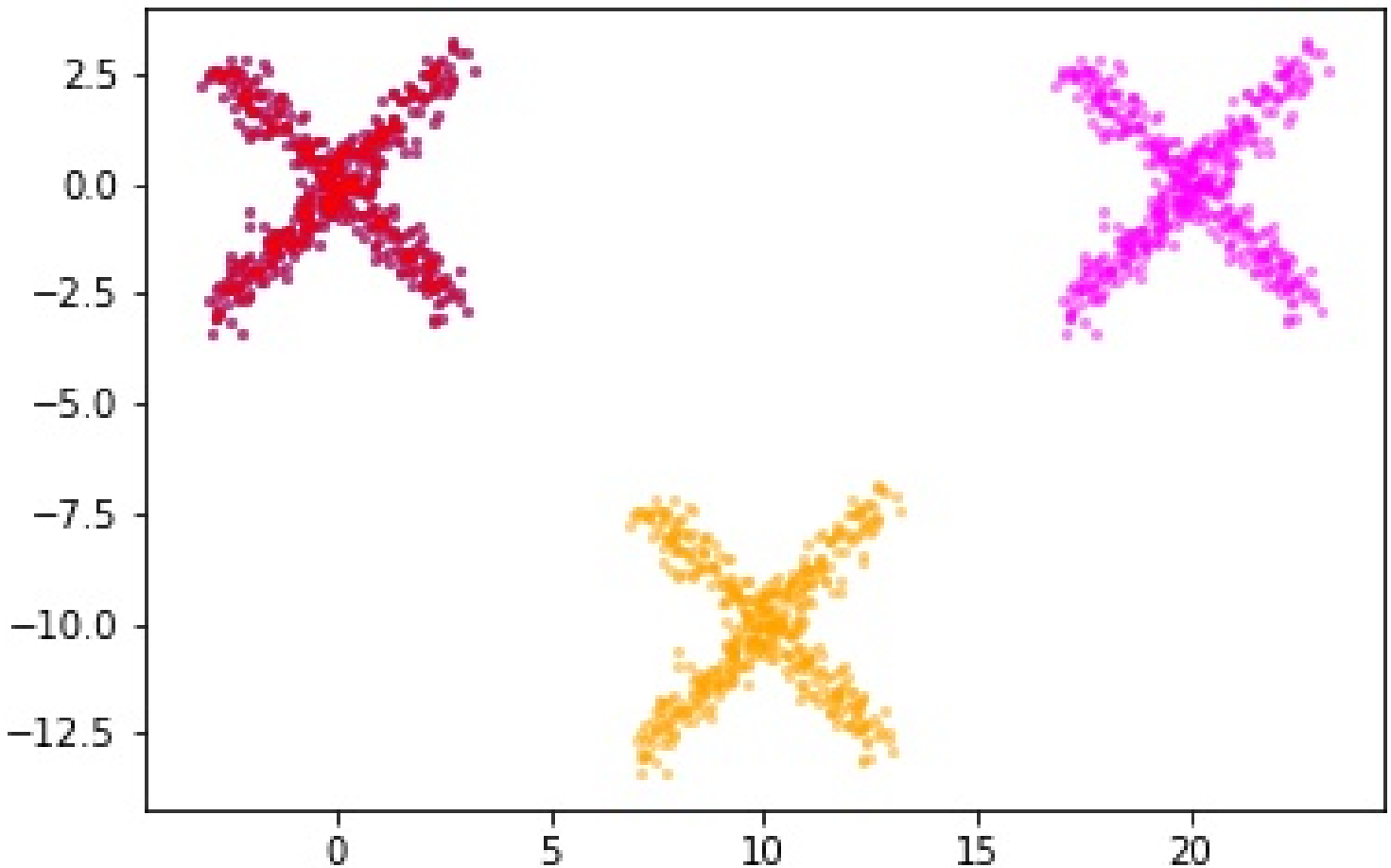
Learning trajectories

joint work with Paul Zhang

GDP group, CSAIL, MIT

August 27th 2021

Trajectory Learning



The model : neural ODE

The model : neural ODE

- Spatio-temporal process $(\mu_t)_{t>0}$

The model : neural ODE

- Spatio-temporal process $(\mu_t)_{t>0}$
- We observe the process at times $t_0, \ldots, t_N$

The model : neural ODE

- Spatio-temporal process $(\mu_t)_{t>0}$
- We observe the process at times $t_0, \ldots, t_N$
- Observation at time t : $\mathrm{x_t} \overset{\text{def.}}{=} (x_t^{(0)}, x_t^{(1)}, \ldots, x_t^{(n_t)}) \in \mathbb{R}^{d \times n_t}$, where $x_t^{(i)} \sim \mu_t$

The model : neural ODE

- Spatio-temporal process $(\mu_t)_{t>0}$
- We observe the process at times $t_0, \ldots, t_N$
- Observation at time t : $\mathrm{x_t} \overset{\text{def.}}{=} (x_t^{(0)}, x_t^{(1)}, \ldots, x_t^{(n_t)}) \in \mathbb{R}^{d \times n_t}$, where $x_t^{(i)} \sim \mu_t$
- Particles are governed by an ODE

$$\frac{dx(t)}{dt} = f(x(t), t, \theta) \tag{1}$$

The model : neural ODE

- Spatio-temporal process $(\mu_t)_{t>0}$
- We observe the process at times $t_0, \ldots, t_N$
- Observation at time t : $\mathrm{x_t} \overset{\text{def.}}{=} (x_t^{(0)}, x_t^{(1)}, \ldots, x_t^{(n_t)}) \in \mathbb{R}^{d \times n_t}$, where $x_t^{(i)} \sim \mu_t$
- Particles are governed by an ODE

$$\frac{dx(t)}{dt} = f(x(t), t, \theta) \tag{1}$$

- **Goal** : Learn the vector field f

Learning the vector field

Simple data fidelity loss :

$$L(\theta) = \sum_{i=1}^{N} D(\mathrm{x}_{t_i}, \mathrm{x}(t_i, \theta)),$$

Learning the vector field

Simple data fidelity loss :

$$L(\theta) = \sum_{i=1}^{N} D(\mathrm{x}_{t_i}, \mathrm{x}(t_i, \theta)),$$

- D is a notion of distance between point clouds (MMD, Wasserstein, Sinkhorn)

Learning the vector field

Simple data fidelity loss :

$$L(\theta) = \sum_{i=1}^{N} D(\mathrm{x}_{t_i}, \mathrm{x}(t_i, \theta)),$$

- D is a notion of distance between point clouds (MMD, Wasserstein, Sinkhorn)
- $\mathrm{x}(t_i, \theta)$ point cloud of solutions to the ODE with initial condition $\mathrm{x}(t_0) = \mathrm{x}_{t_0}$.

Learning the vector field

Simple data fidelity loss :

$$L(\theta) = \sum_{i=1}^{N} D(\mathrm{x}_{t_i}, \mathrm{x}(t_i, \theta)),$$

- D is a notion of distance between point clouds (MMD, Wasserstein, Sinkhorn)
- $\mathrm{x}(t_i, \theta)$ point cloud of solutions to the ODE with initial condition $\mathrm{x}(t_0) = \mathrm{x}_{t_0}$.
- Loss ensures that point cloud of observations $x(t_i)$ matches the point cloud of solutions $\mathrm{x}(t_i, \theta)$ at all times t_i.

Learning the vector field

Simple data fidelity loss :

$$L(\theta) = \sum_{i=1}^{N} D(\mathrm{x}_{t_i}, \mathrm{x}(t_i, \theta)),$$

Learning the vector field

Simple data fidelity loss :

$$L(\theta) = \sum_{i=1}^{N} D(\mathrm{x}_{t_i}, \mathrm{x}(t_i, \theta)),$$

- We minimize L using SGD

Learning the vector field

Simple data fidelity loss :

$$L(\theta) = \sum_{i=1}^{N} D(\mathrm{x}_{t_i}, \mathrm{x}(t_i, \theta)),$$

- We minimize L using SGD
- Issue : backpropagating through ODE solver is costly

Learning the vector field

Simple data fidelity loss :

$$L(\theta) = \sum_{i=1}^{N} D(\mathrm{x}_{t_i}, \mathrm{x}(t_i, \theta)),$$

- We minimize L using SGD
- Issue : backpropagating through ODE solver is costly
- Trick from neural ODE : use adjoint method $\rightarrow$ solve adjoint ODE to compute gradient of L with ODE constraint

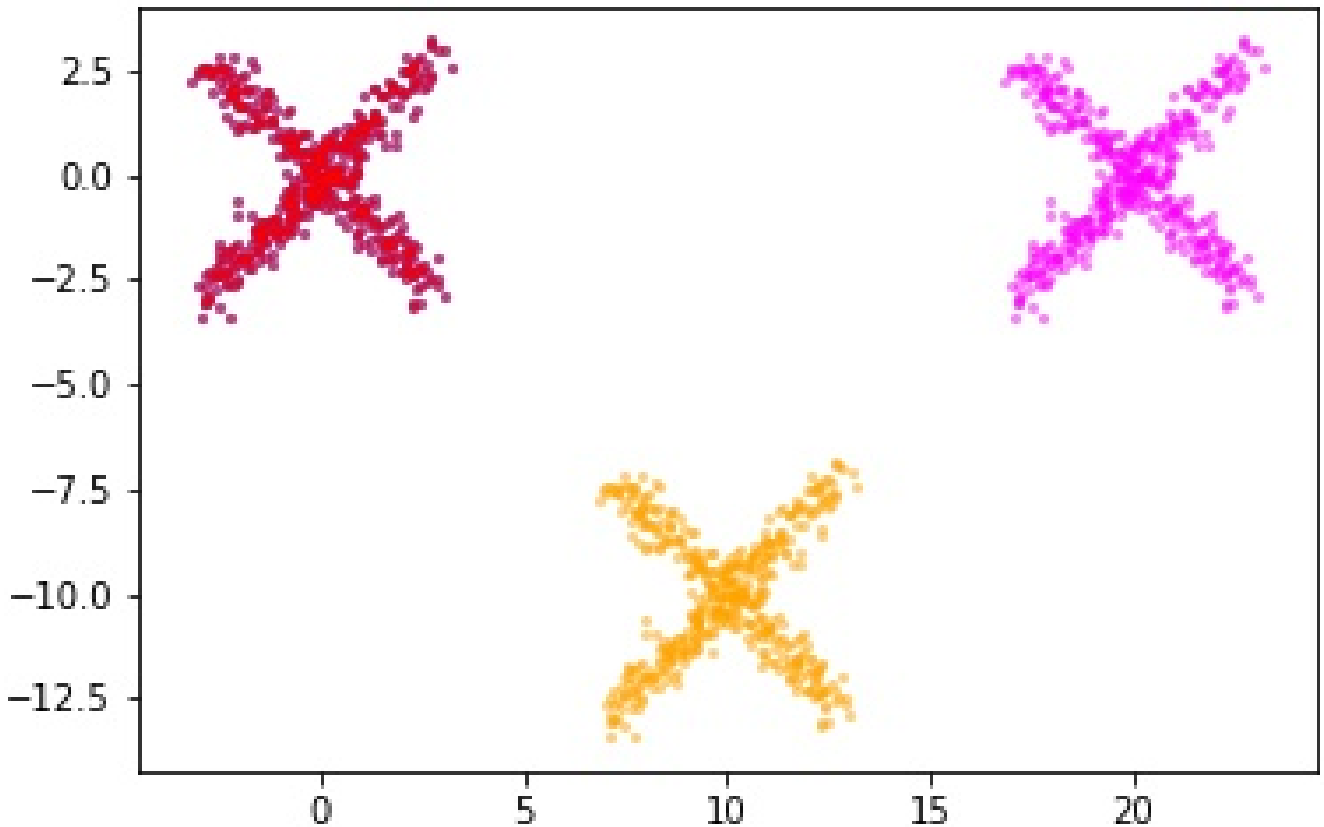
Learning the vector field

Simple data fidelity loss :

$$L(\theta) = \sum_{i=1}^{N} D(\mathrm{x}_{t_i}, \mathrm{x}(t_i, \theta)),$$

- We minimize L using SGD
- Issue : backpropagating through ODE solver is costly
- Trick from neural ODE : use adjoint method $\rightarrow$ solve adjoint ODE to compute gradient of L with ODE constraint
- Adjoint method is implemented in Torchdiffeq library

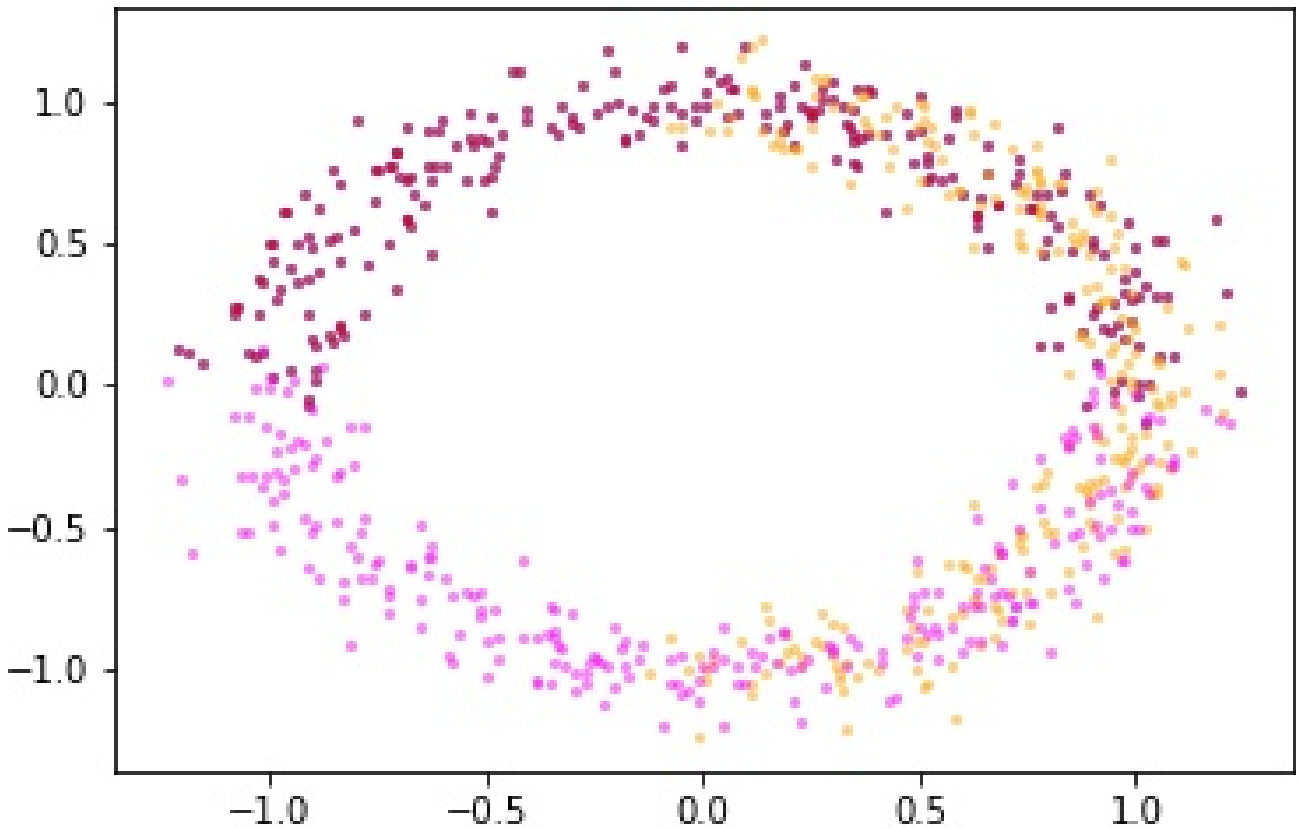
First experiments



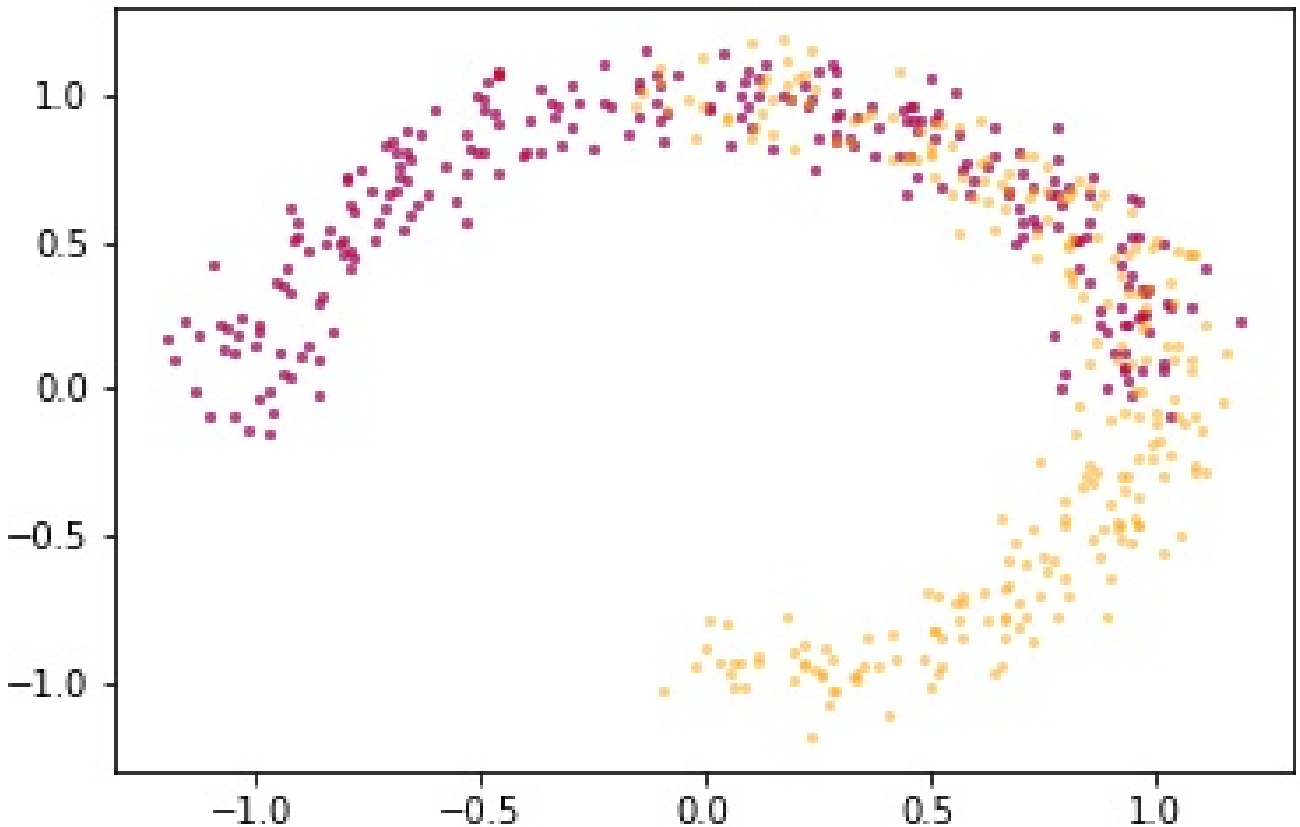
Some remarks about the trajectory :

- Good
  - Smooth trajectory
  - Good data fidelity
- Bad
  - Not symmetric : rotates $0/rightarrow1$ but translates $1/rightarrow2$

Half-moon with 3 steps



Half-moon with 2 steps



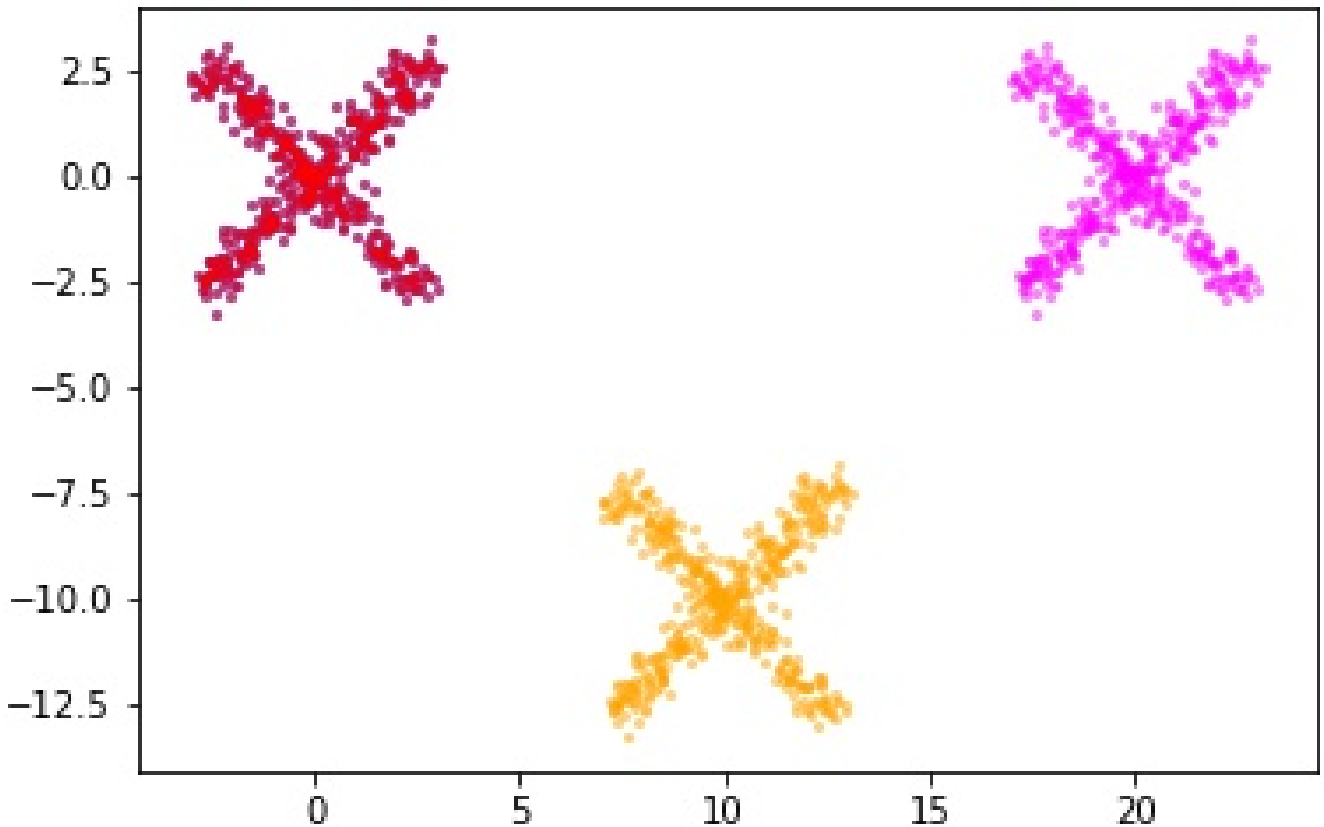
Adding prior knowledge

We can favor certain behaviors by penalizing the loss with the following energy :

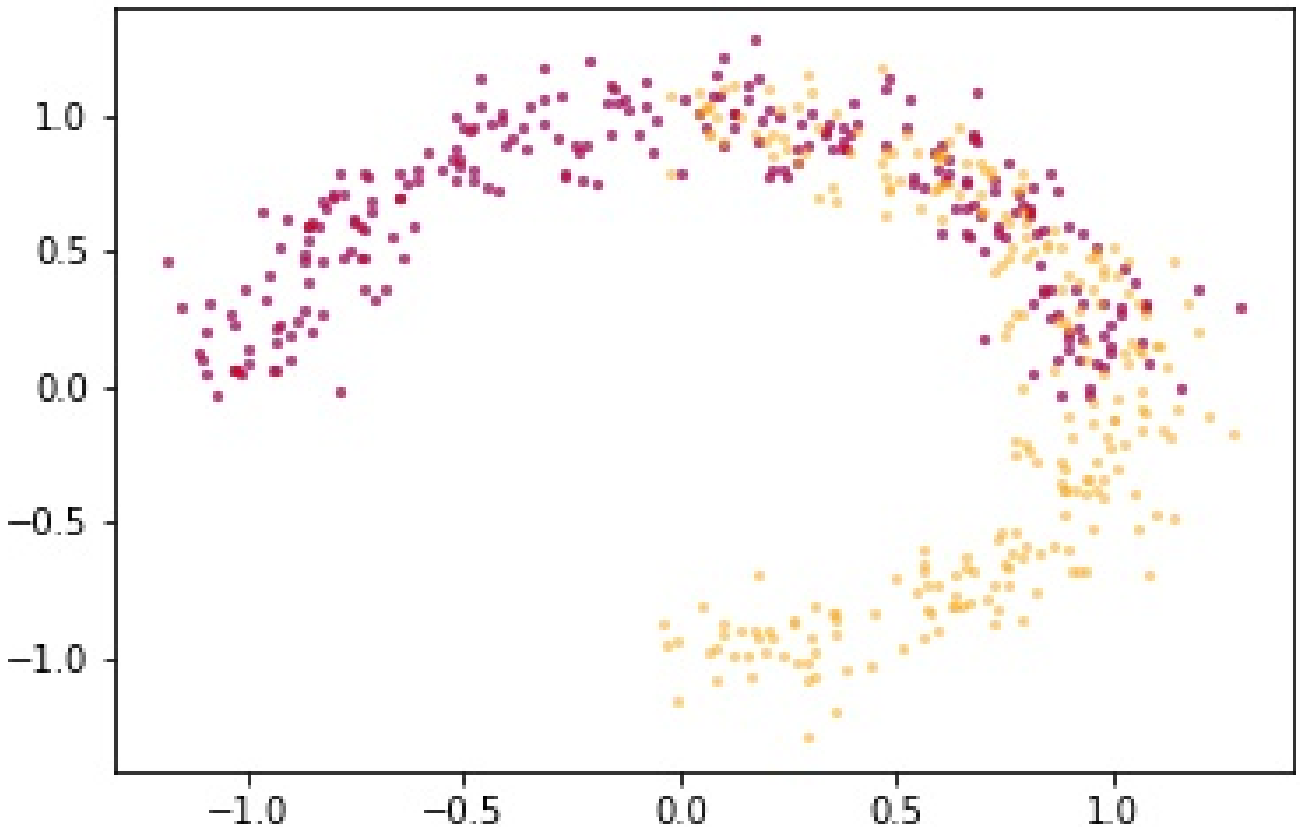
$$E(\theta) = \int_{\Omega \times [0,T]} \left\| L \left[\frac{\partial f(\mathbf{x}, t, \theta)}{\partial x} \right] \right\|_F^2 d\mathbf{x} dt \tag{2}$$

- $L_d[A] = \mathrm{Tr}[A]$ computes a divergence based energy,
- $L_r[A] = A + A^T$ computes a rigidity based energy,
- $L_\times[A] = A - A^T$ computes a curl based energy.

Crosses with 3 steps, no curl



Adding negative curl loss



Dealing with negative curl loss

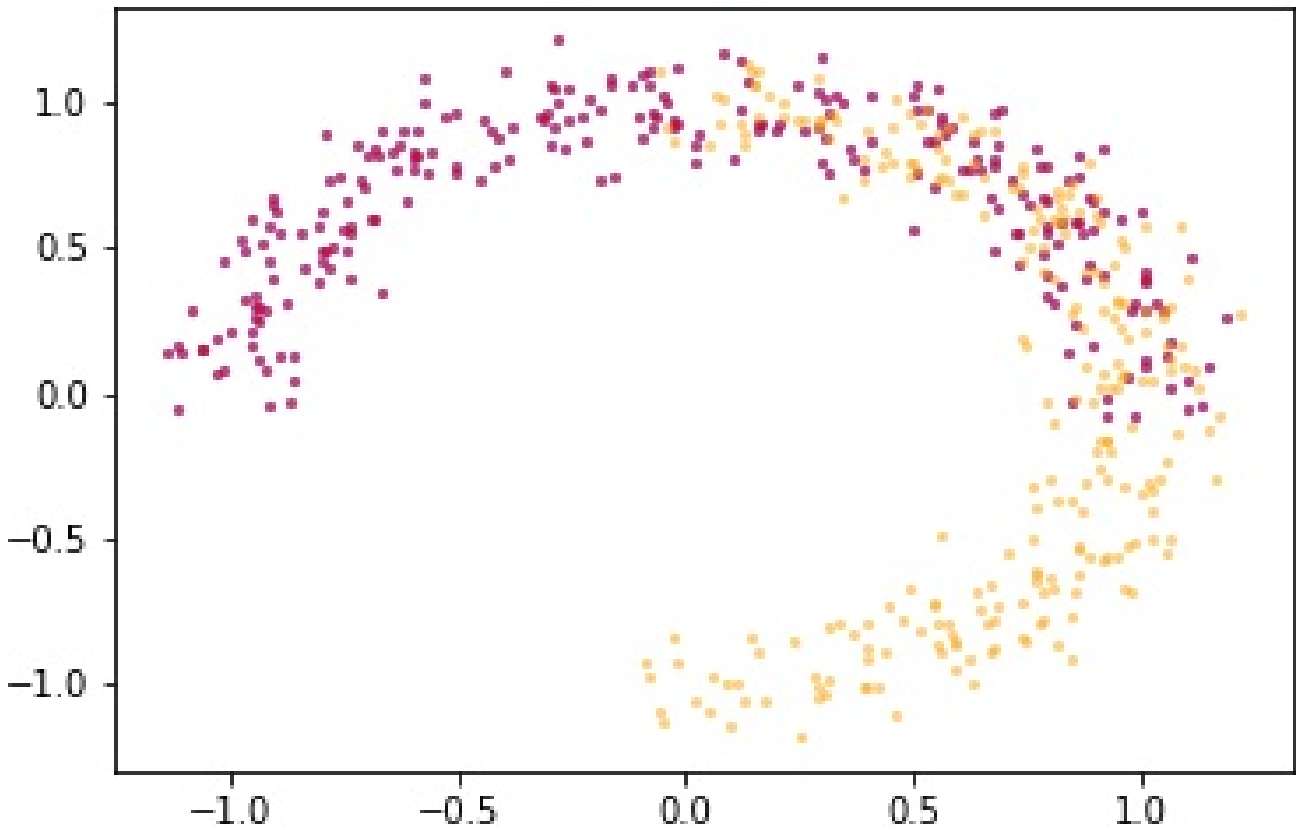
Dealing with negative curl loss

- Issue : Curl is unbounded, so negative curl loss will go to $-\infty$ when minimized...

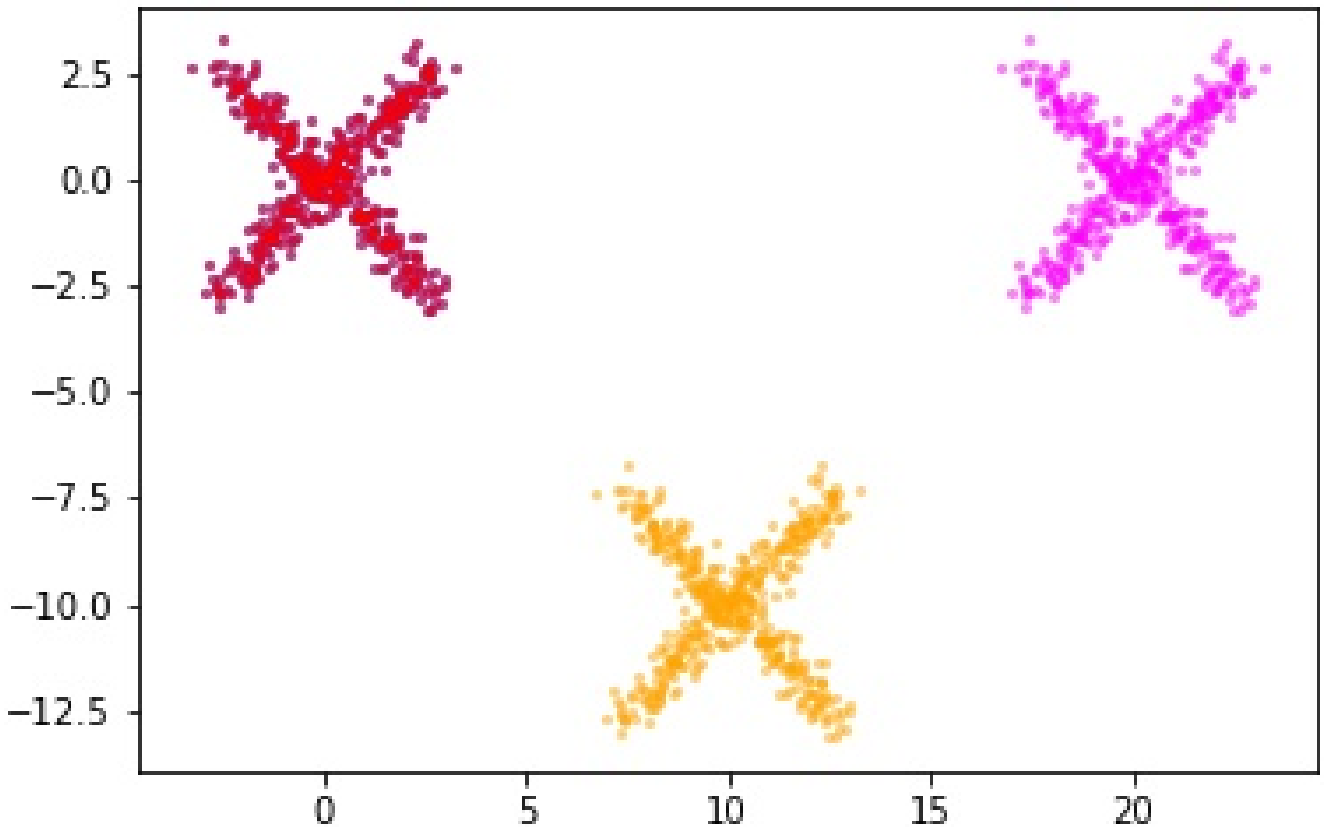
Dealing with negative curl loss

- Issue : Curl is unbounded, so negative curl loss will go to $-\infty$ when minimized...
- Solution : Clamp the curl loss (using torch.clamp) to ignore large values

Adding clamped negative curl loss



Crosses with 3 steps, clamped curl



Applications

Applications

- Graphics : interactive tool to generate trajectories

Applications

- Graphics : interactive tool to generate trajectories
  - need to handle more complex shapes

Applications

- Graphics : interactive tool to generate trajectories
  - need to handle more complex shapes
  - move to 3D

Applications

- Graphics : interactive tool to generate trajectories
  - need to handle more complex shapes
  - move to 3D
- Biology : cell differentiation

Applications

- Graphics : interactive tool to generate trajectories
  - need to handle more complex shapes
  - move to 3D
- Biology : cell differentiation
  - move to SDE framework to account for cell division : as simple as adding small Gaussian noise along ODE trajectory ?

Applications

- Graphics : interactive tool to generate trajectories
  - need to handle more complex shapes
  - move to 3D
- Biology : cell differentiation
  - move to SDE framework to account for cell division : as simple as adding small Gaussian noise along ODE trajectory ?
  - how many observation time steps can we handle ?

Applications

- Graphics : interactive tool to generate trajectories
  - need to handle more complex shapes
  - move to 3D
- Biology : cell differentiation
  - move to SDE framework to account for cell division : as simple as adding small Gaussian noise along ODE trajectory ?
  - how many observation time steps can we handle ?
  - what are meaningful regularizers in high dimensional gene expression space ?