Flow Matching and Rectified Flow

UW MLJC Summer Workshop 2026

Zihui(Andy) Liu

Incomplete Overview of Generative Models

- Minimax, Game Theory: Generative Adversarial Model (GAN)
- Maximum Likelihood Estimation: Variational Autoencoder (VAE)
- Continuous “Time” Approach:
  - SDE-Based:
    - Denoising Diffusion Model, Score-based Generative Model
  - 1-Step ODE-Based Model:
    - Based on Optimal Transport (OT), but it doesn’t scale well in high-dimensional space.
  - **Flow-Matching: OT-CFM, Rectified Flow, etc.**
- Energy-Based Model: Engression, etc.

Liu et al. 2022, Rectified Flow
Lipman et al. 2023, Flow Matching

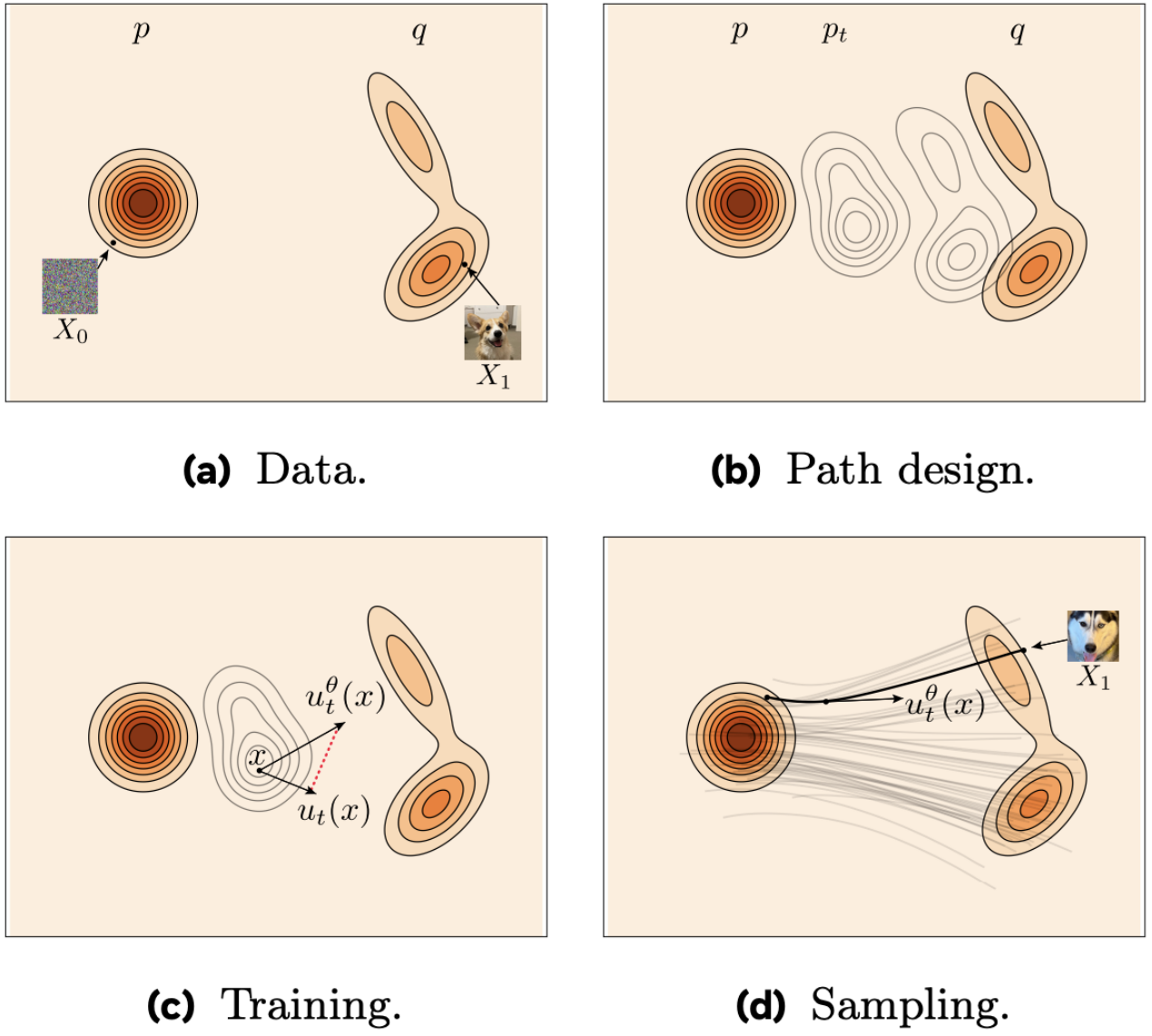
Flow Matching

- Construct an ODE that moves the distribution
  - From **p** at t = 0
  - To **q** at t = 1
- Training: Find u(x, t)

$$\frac{\partial x}{\partial t} = u_\theta(x, t)$$

- Inference: Integrate ODE

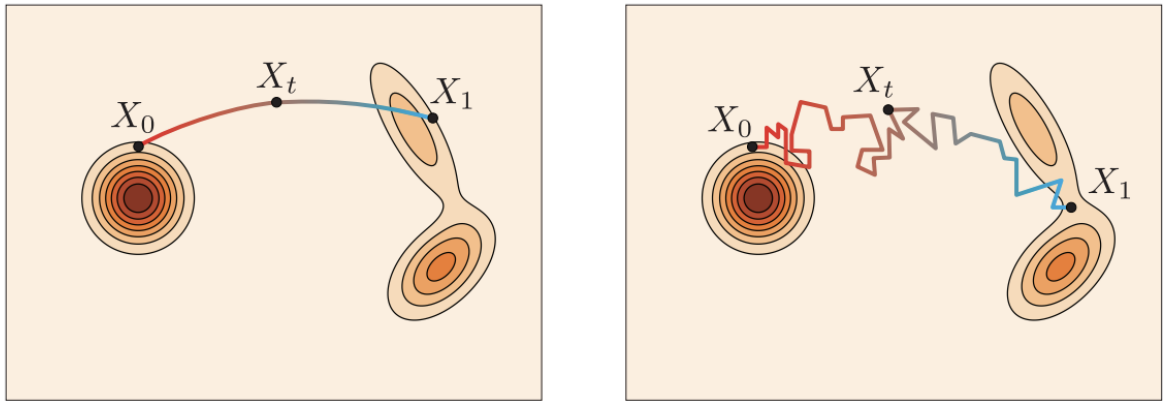
$$X_1 = \int_0^1 u_\theta(x, t)\, dt, \quad x(0) = X_0$$



(a) Data. **(b)** Path design. **(c)** Training. **(d)** Sampling.

Good Resources: https://junwei-lu.github.io/bst236/chapter_generative_model/flow_match/

Benefit of Flow Matching



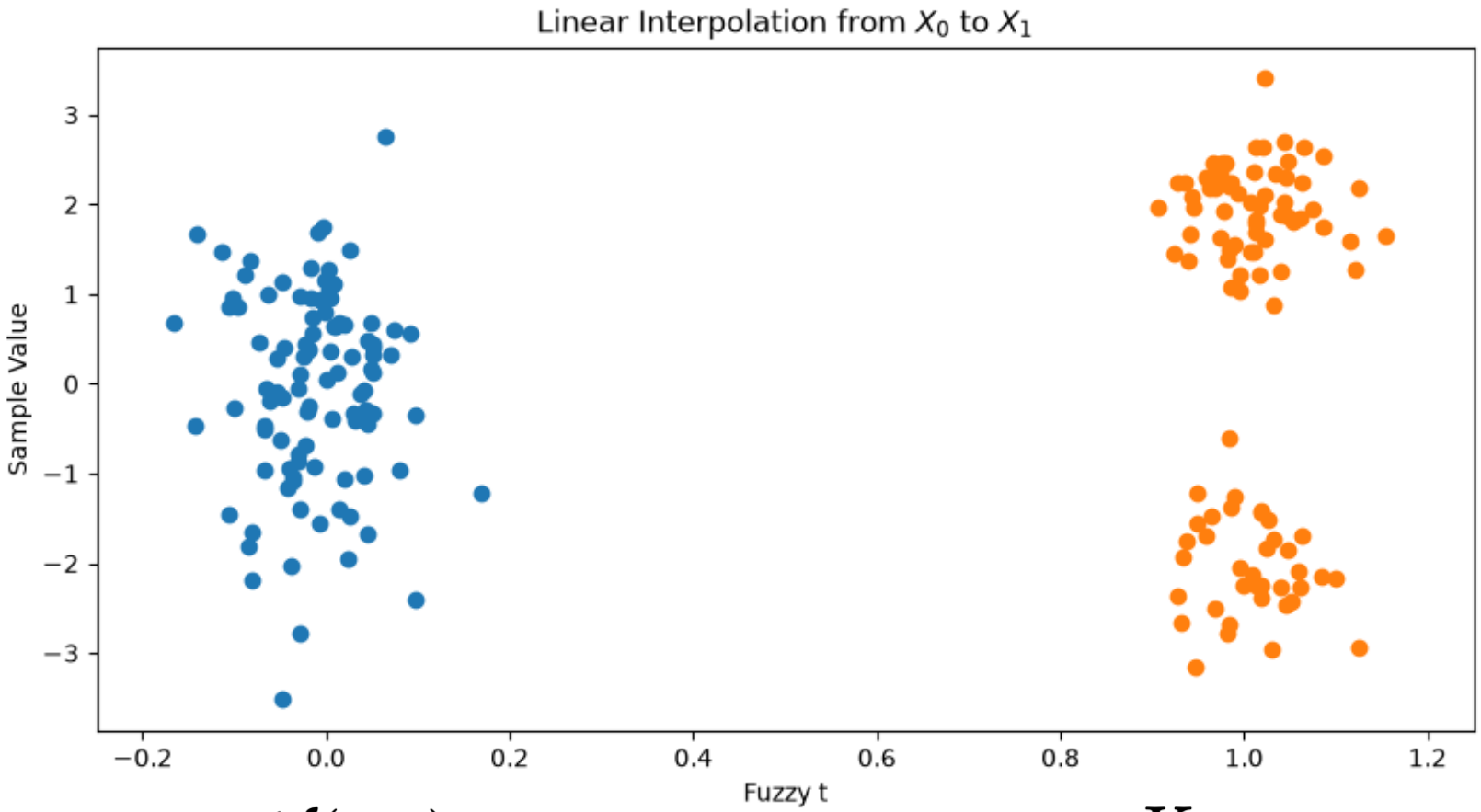
(a) Flow **(b)** Diffusion

- The flow matching has more stable trajectory

despite diffusion model is called stable diffusion

- Only initial X_0 is stochastic, everything else is deterministic**
  - Including both training and inferencing → Easier training and inferencing
  - We can use proven architecture

Flow Matching Algorithm, in 1d



$X_0 \sim \mathcal{N}(0, 1)$

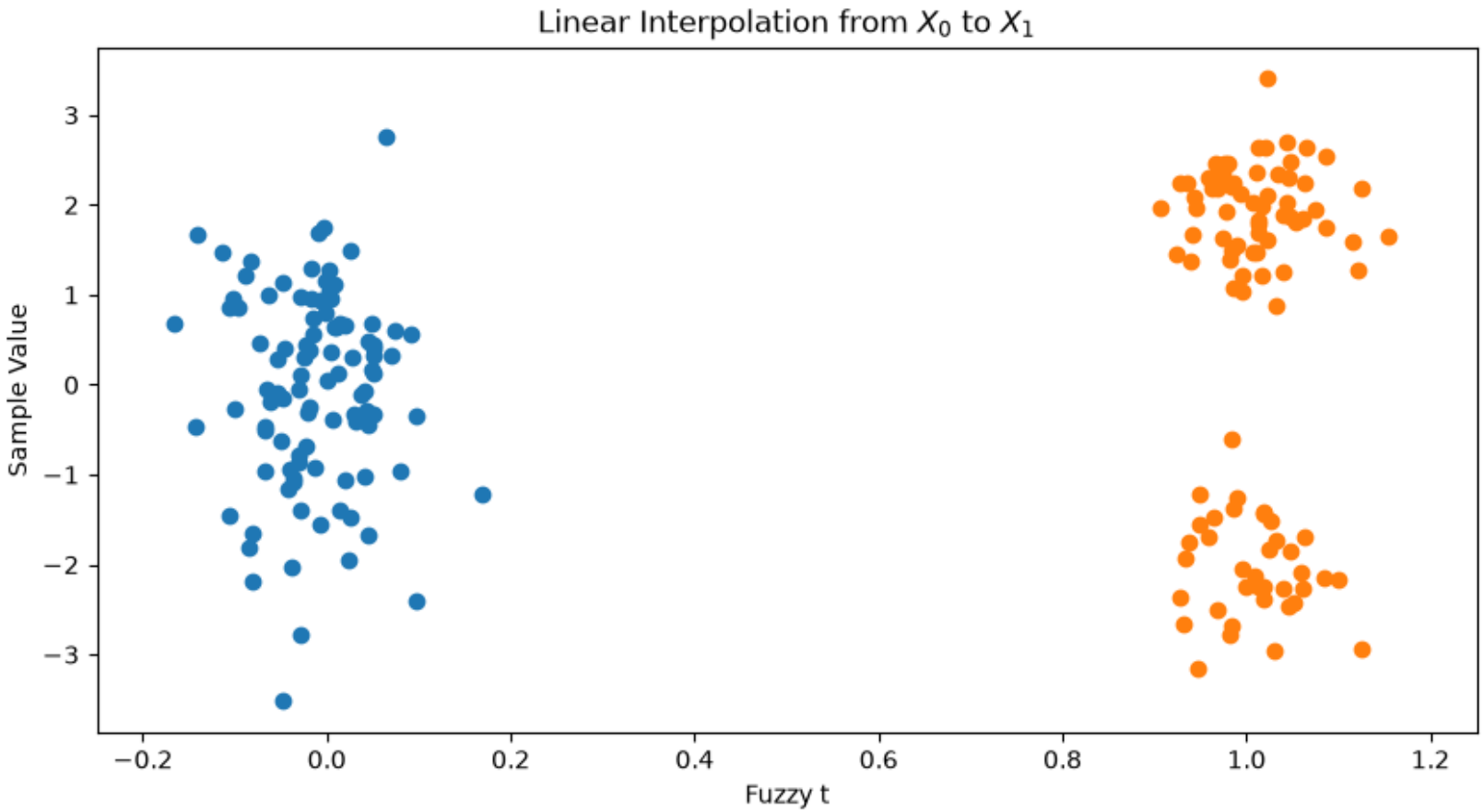
Initial Random Samples at t=0

Flow ~~~~

$X_1 \sim q$

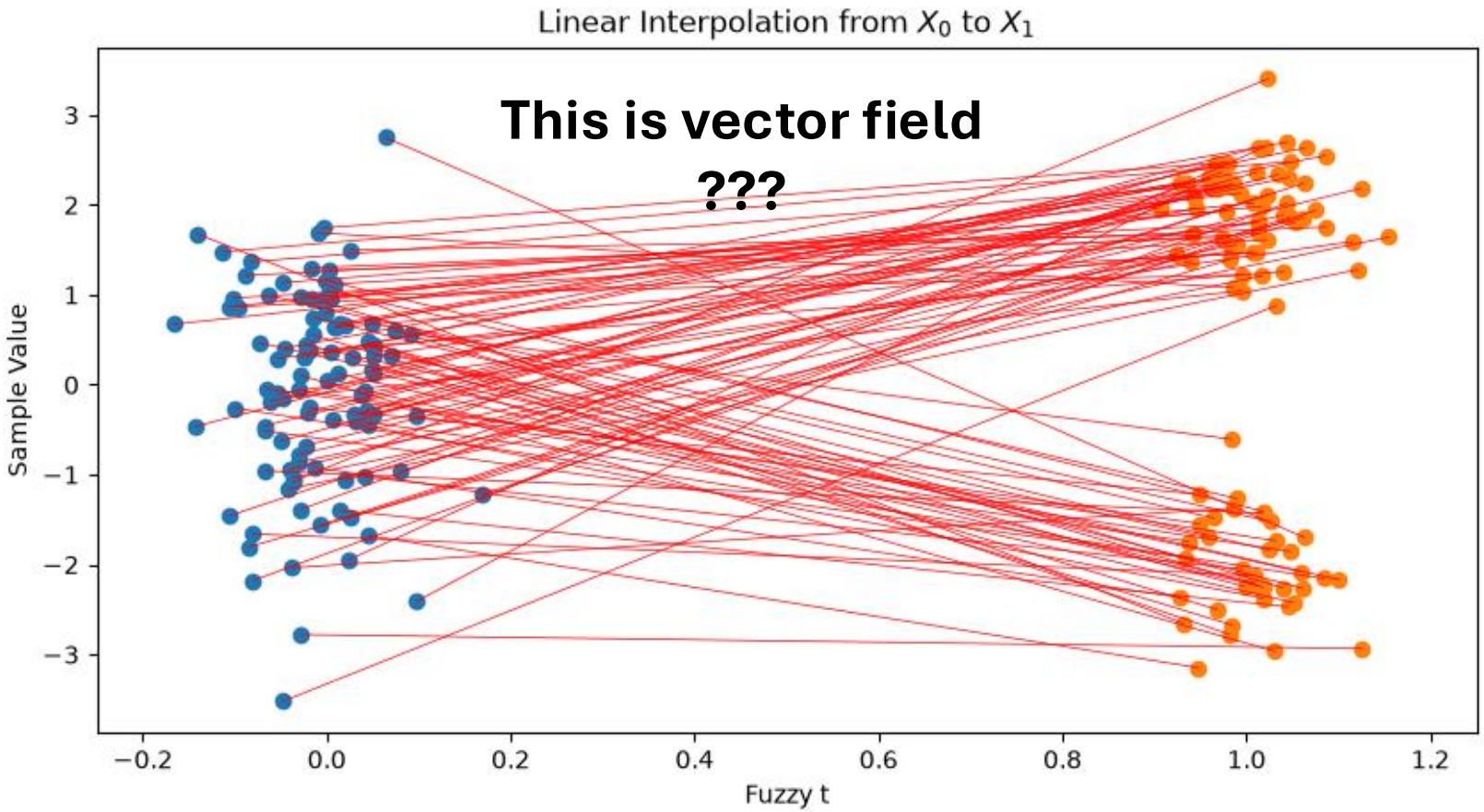
Training Samples at t=1

Flow Matching Algorithm, in 1d



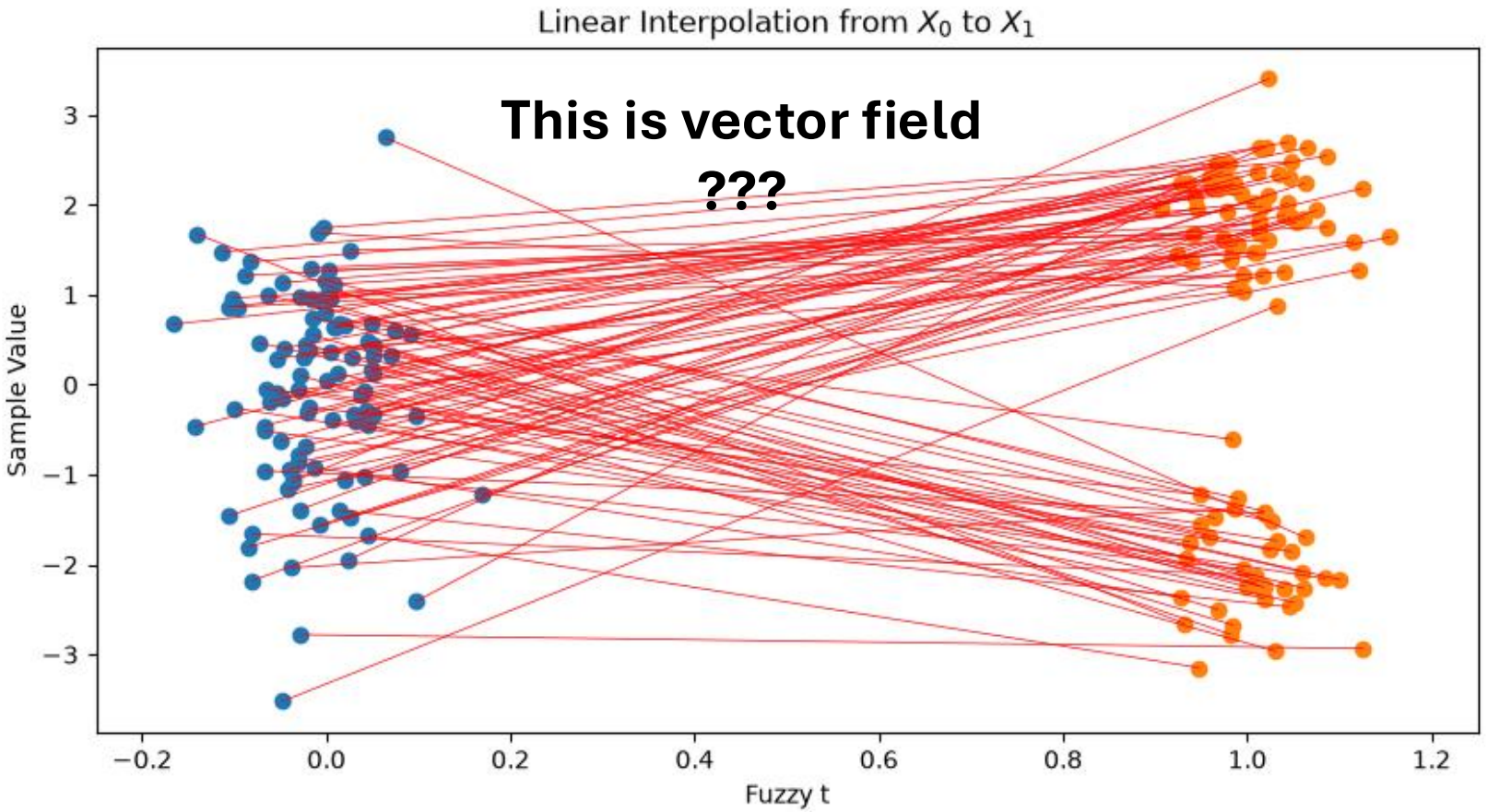
How do we know the corresponding pairs?

Flow Matching Algorithm, in 1d



How do we know the corresponding pairs? **In spirit of ML, we choose randomly**

Flow Matching Algorithm, in 1d



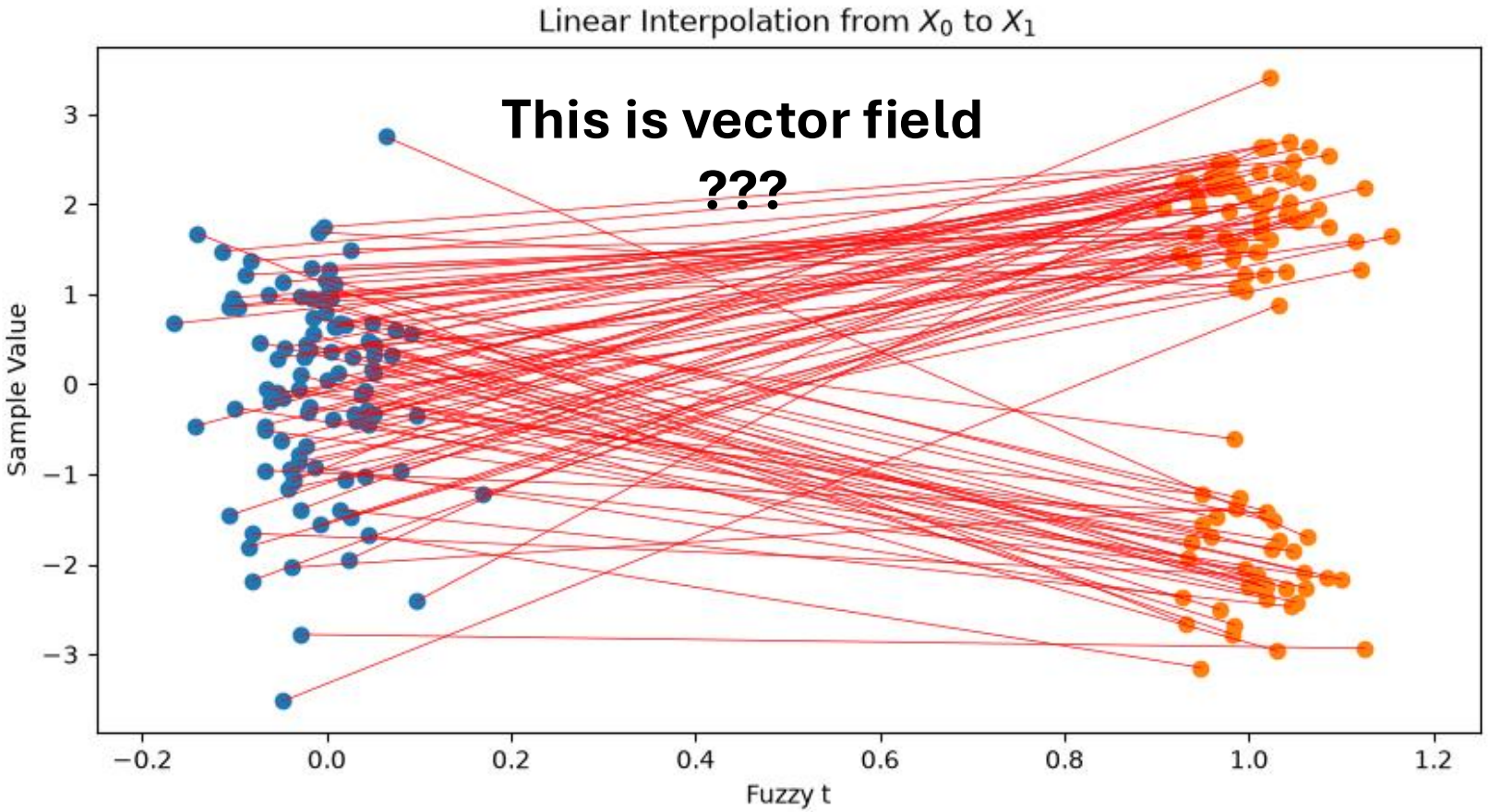
In higher dimension, the probability of intersecting two line is low

Curse/Bless(?) of dimensionality

Also, don't worry,
we can **rectify** it later

How do we know the corresponding pairs? **In spirit of ML, we choose randomly**

Flow Matching Algorithm, in 1d



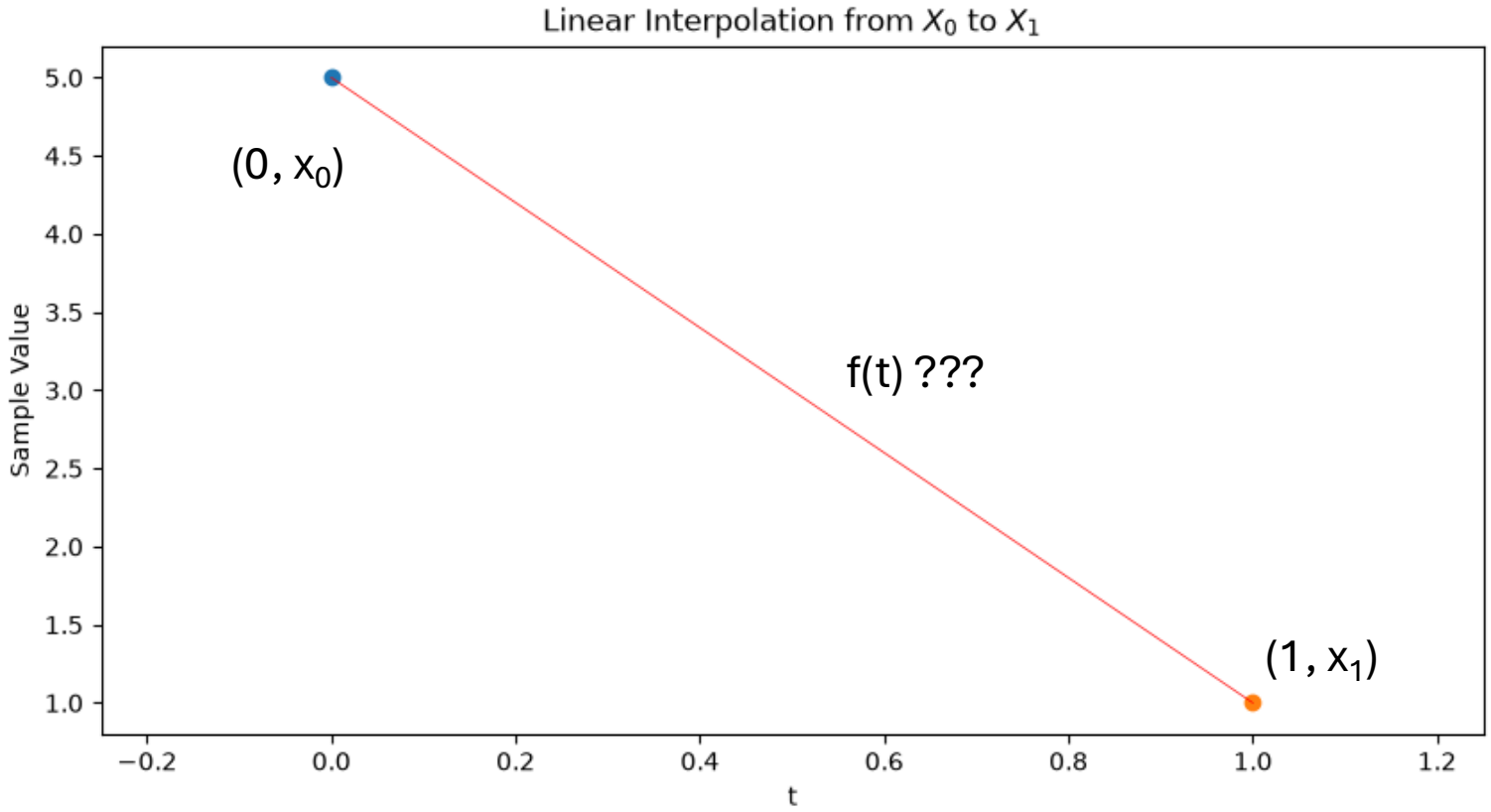
In higher dimension, the probability of intersecting two line is low

Curse/Bless(?) of dimensionality

How do we know the corresponding pairs? **In spirit of choose randoml**

Constructing Target Flow Trajectory

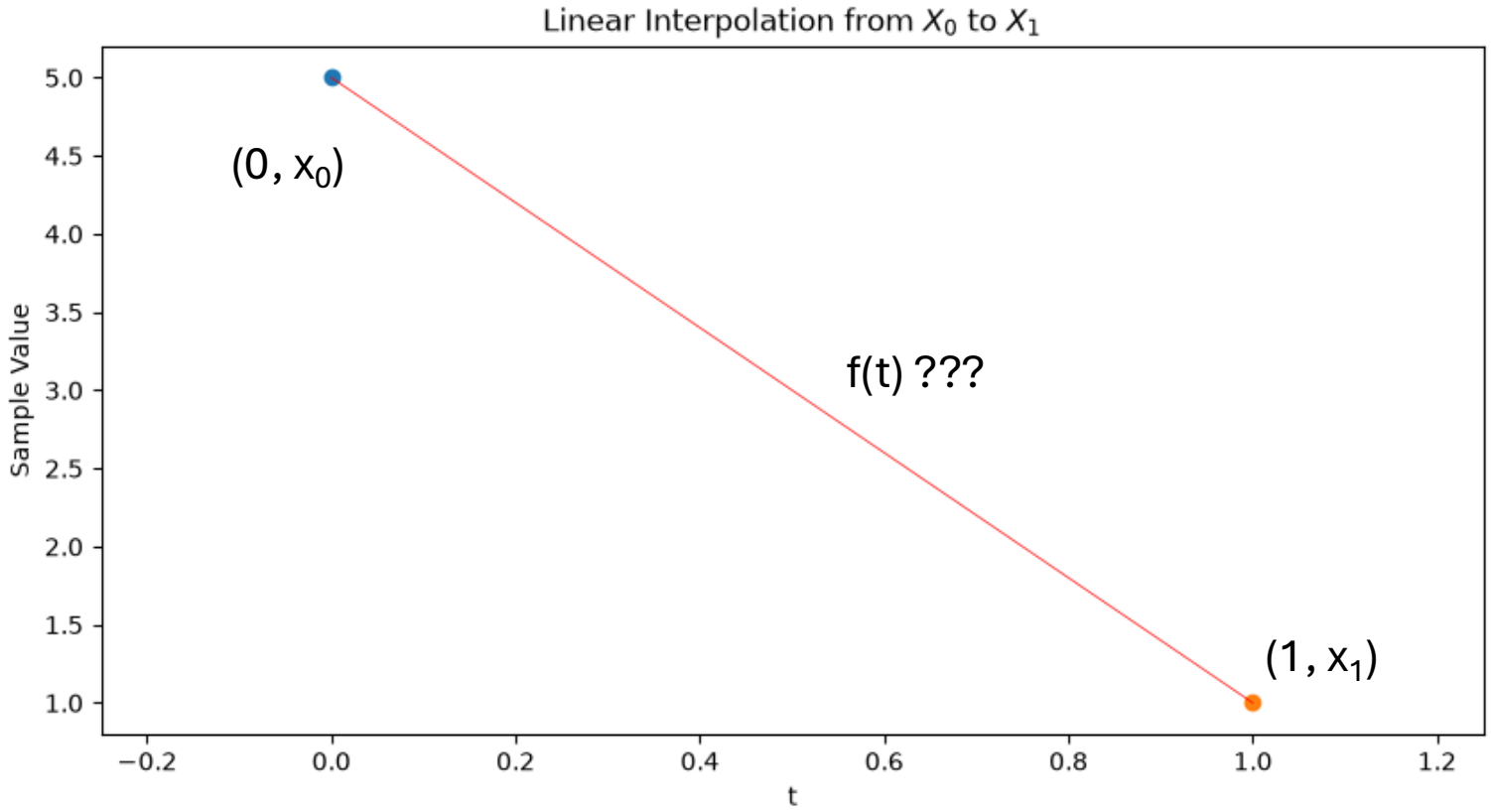
Warm-up exercise: Find a function that interpolate $(0, x_0)$ and $(1, x_1)$



Magical Word: Convex Combination

Constructing Target Flow Trajectory

Warm-up exercise: Find a function that interpolate $(0, x_0)$ and $(1, x_1)$



$$f(t) = (1-t)x_0 + tx_1$$

Define NN Model

See Notebook

```
# input 2d (x, t)
# output 1d dx/dt
self.net = nn.Sequential(
    ...
)
```

Define NN Model

Example MLP, feel free to replace with your own favorite: CNN, RNN, transformer, Unet, …

```
self.net = nn.Sequential(
    nn.Linear(2, 10),
    nn.ReLU(),

    nn.Linear(10, 64),
    nn.ReLU(),

    nn.Linear(64, 10),
    nn.ReLU(),

    nn.Linear(10, 1),
)
```

Training Loop

See Notebook

```
# interpolate the state at random time t
# convex combination between tensor_X0 and tensor_X1
x_at_t = ...
```

```
# forward path, invoke the NN model
dx_dt_pred = ...
```

```
# calculate target velocity, Hint
#  dx     x1 - x0
# ---- = ---------
#  dt     t1 - t0
dx_dt_target = ...
```

```
# loss function: mean squared error MSE
loss = ...
```

Training Loop

See Notebook

```
# interpolate the state at random time t
# convex combination between tensor_X0 and tensor_X1
x_at_t = (1-t) * tensor_X0 + t * tensor_X1
```

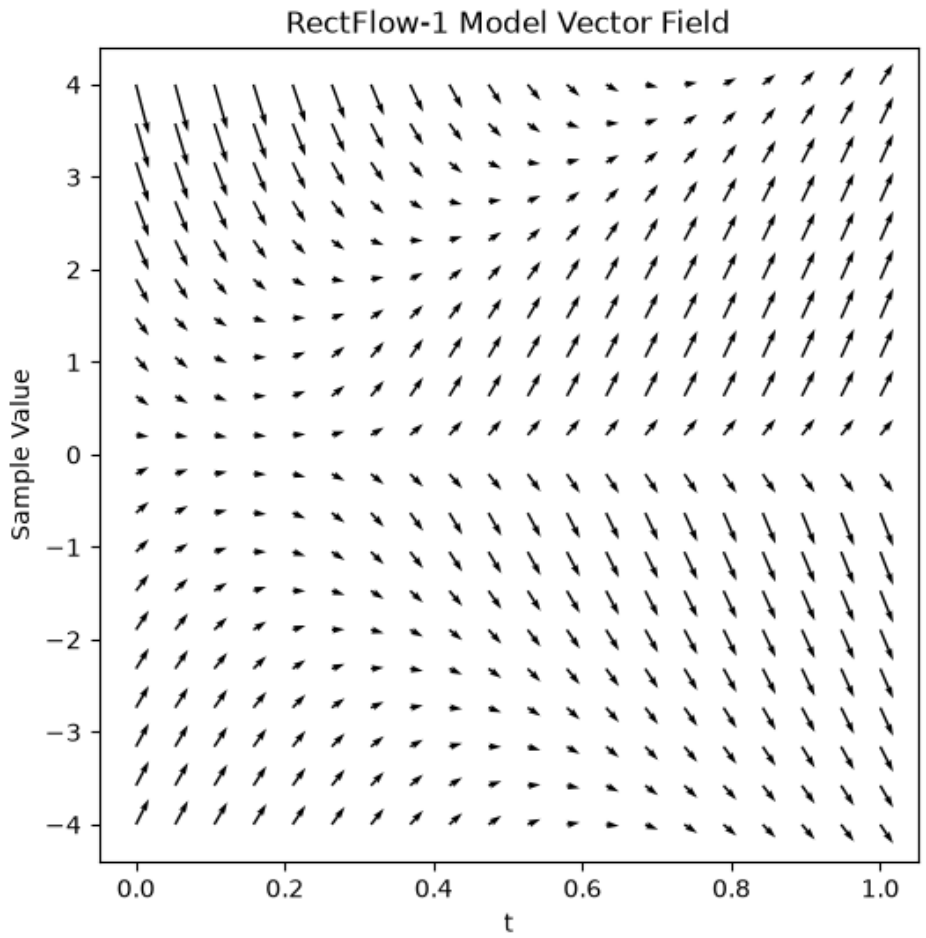
```
# forward path, invoke the NN model
dx_dt_pred = model(x_at_t, t)
```

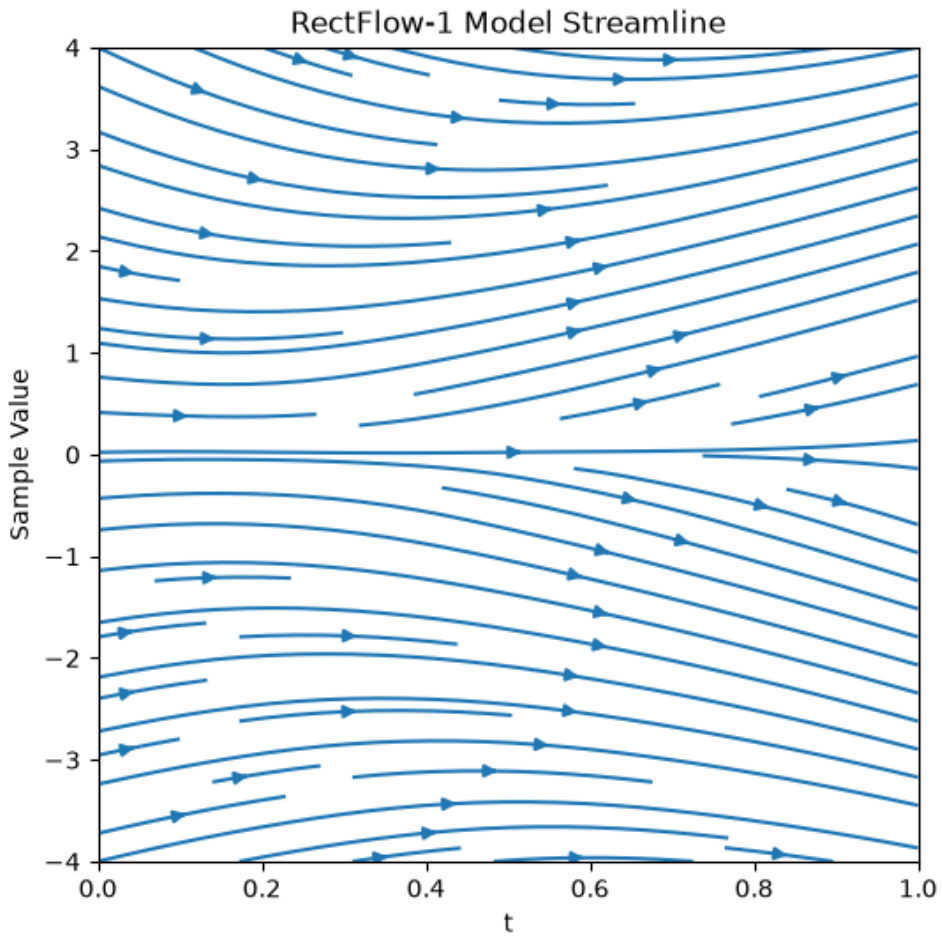
```
# calculate target velocity, Hint
#  dx     x1 - x0
# ---- = ---------
#  dt     t1 - t0
dx_dt_target = tensor_X1 - tensor_X0
```

```
# loss function: mean squared error MSE
loss = torch.mean((dx_dt_pred - dx_dt_target) ** 2)
```

Vector Field of $\frac{\partial x}{\partial t} = u_\theta(x, t)$

x





Integrating the ODE

$$X_1 = \int_0^1 u_\theta(x, t)\, dt, \quad x(0) = X_0$$

```
# construct input for current "time"
t = np.ones_like(x) * (i / steps)
```

```
# get the vector field
dx_dt = ...
```

```
# simple explicit Euler method
x = ...
```

Integrating the ODE

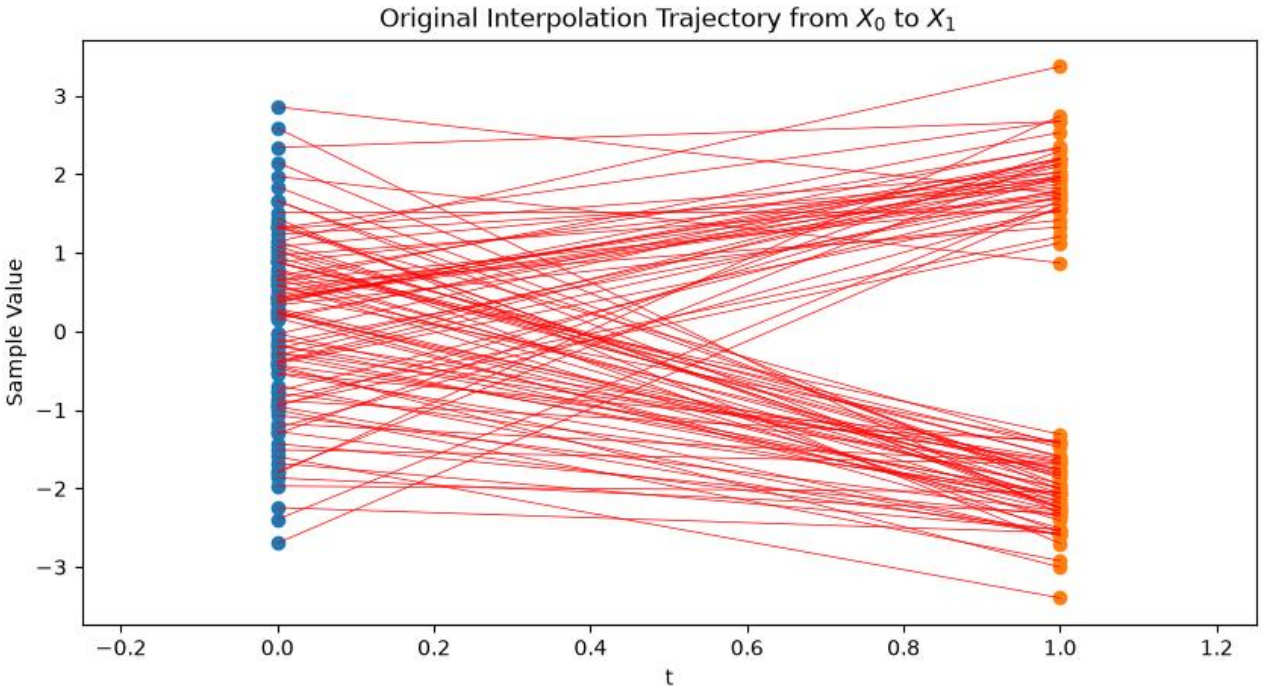
Forward Integration

```
# construct input for current "time"
t = np.ones_like(x) * (i / steps)

# get the vector field
dx_dt = model(x, t)

# simple explicit euler method
x = x + dx_dt * dt
```

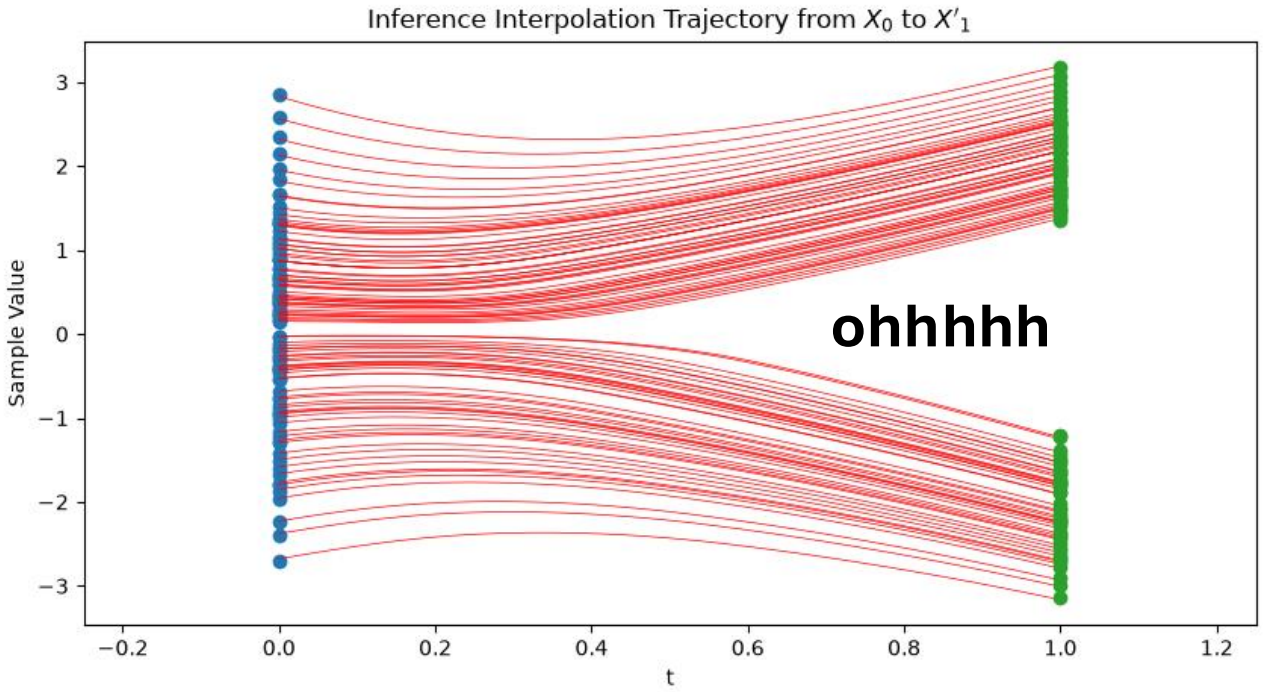
Backward Integration

```
# construct input for current "time"
t = np.ones_like(x) * (i / steps)

# get the vector field
dx_dt = model(x, t)

# simple explicit euler method
x = x - dx_dt * dt
```

The Flow ~~~~

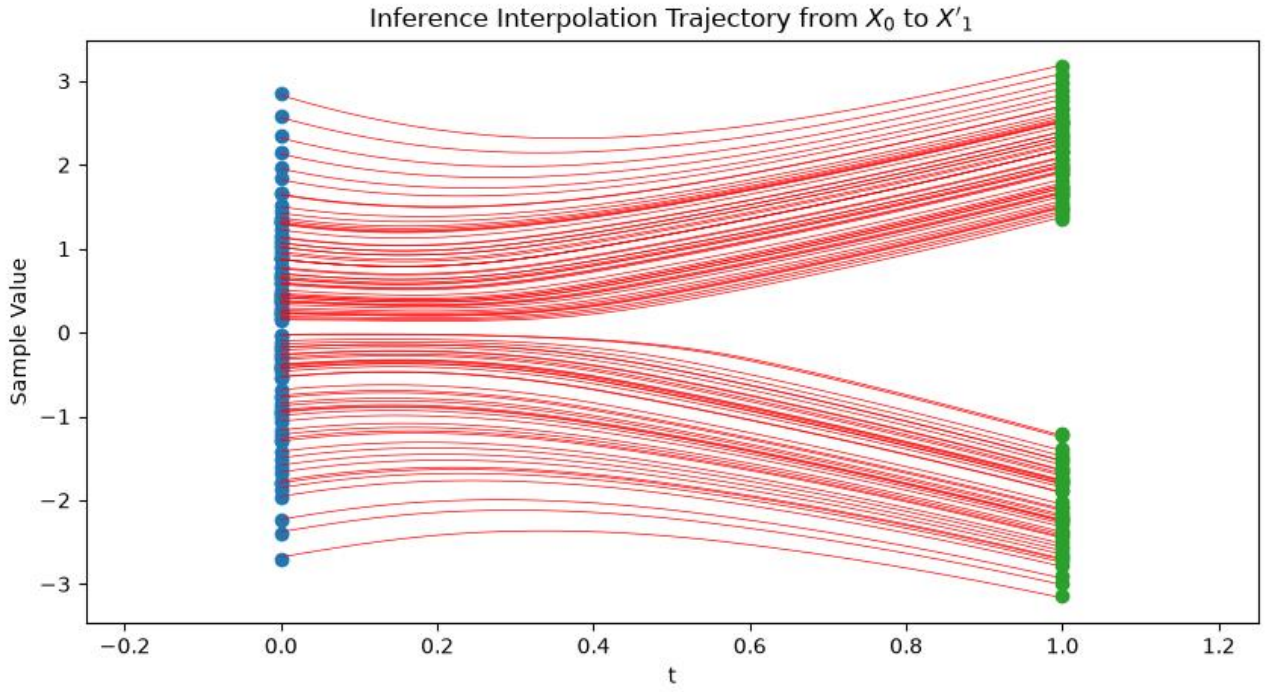


Original Training Sample



Generated Training Sample

The Flow ~~~~



ODEs always have a unique solution for a given initial condition → Trajectory cannot intersect

Can we use this to improve the initial random sampling and path interpolation?

The Flow ~~~~

ODEs always have a unique solution for a given initial condition → Trajectory cannot intersect

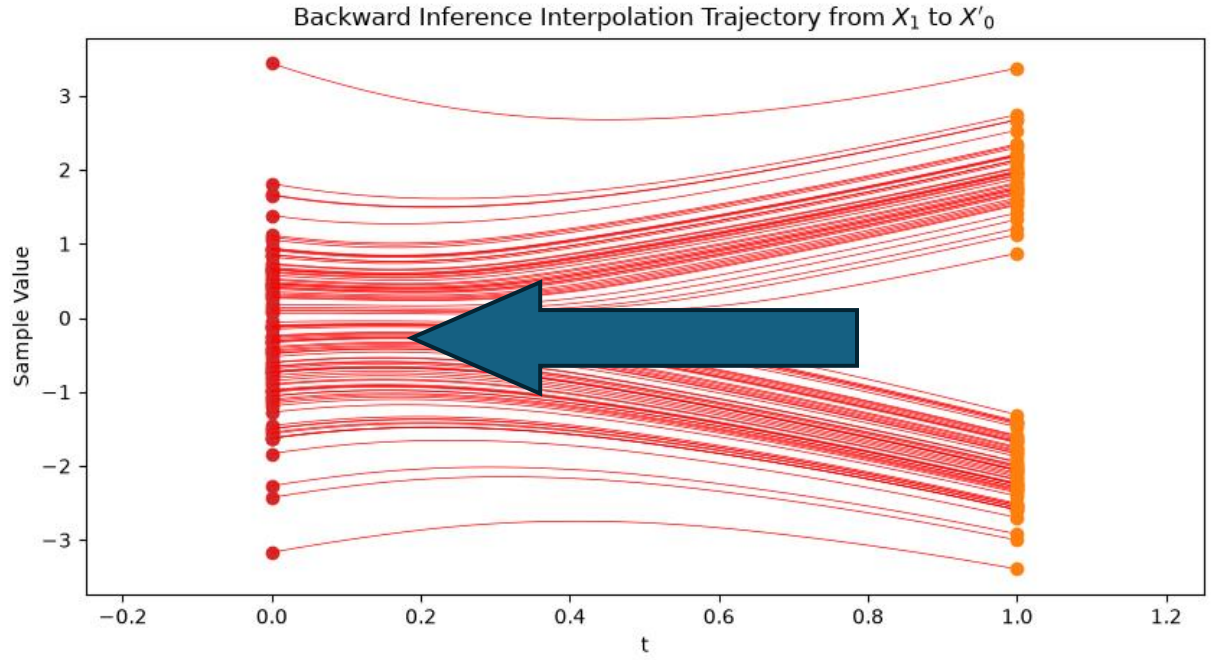
Can we use this to improve the initial random sampling and path interpolation?

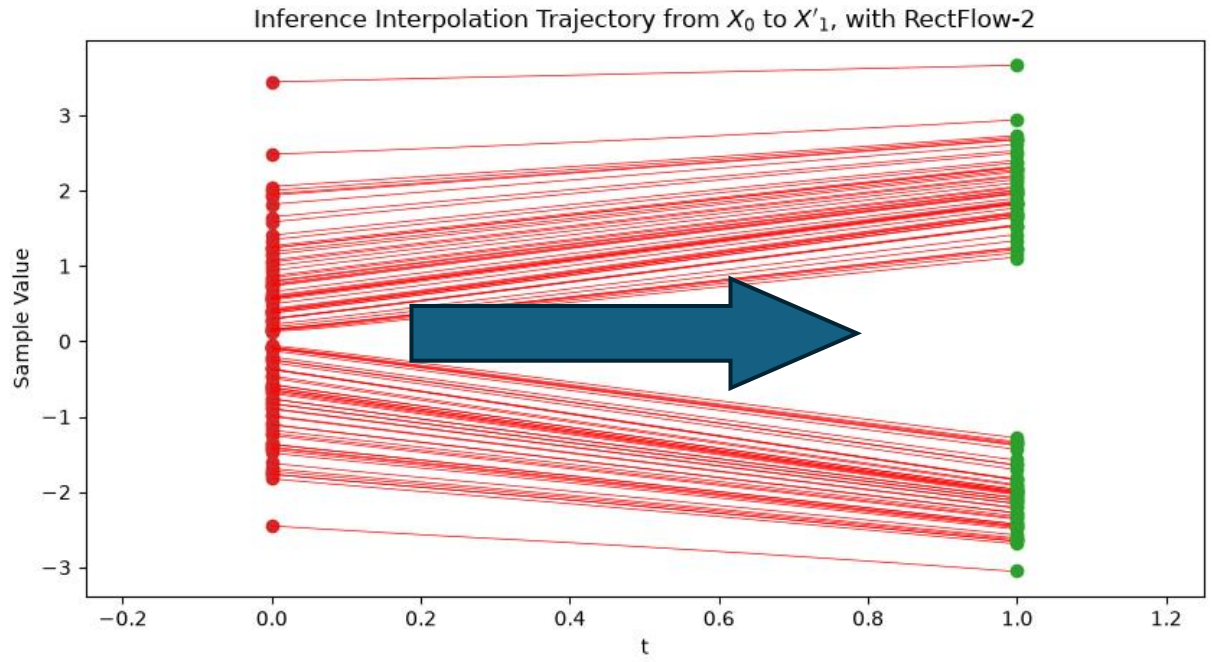
Let's go back by running back in time!

The Reflow ~~~~

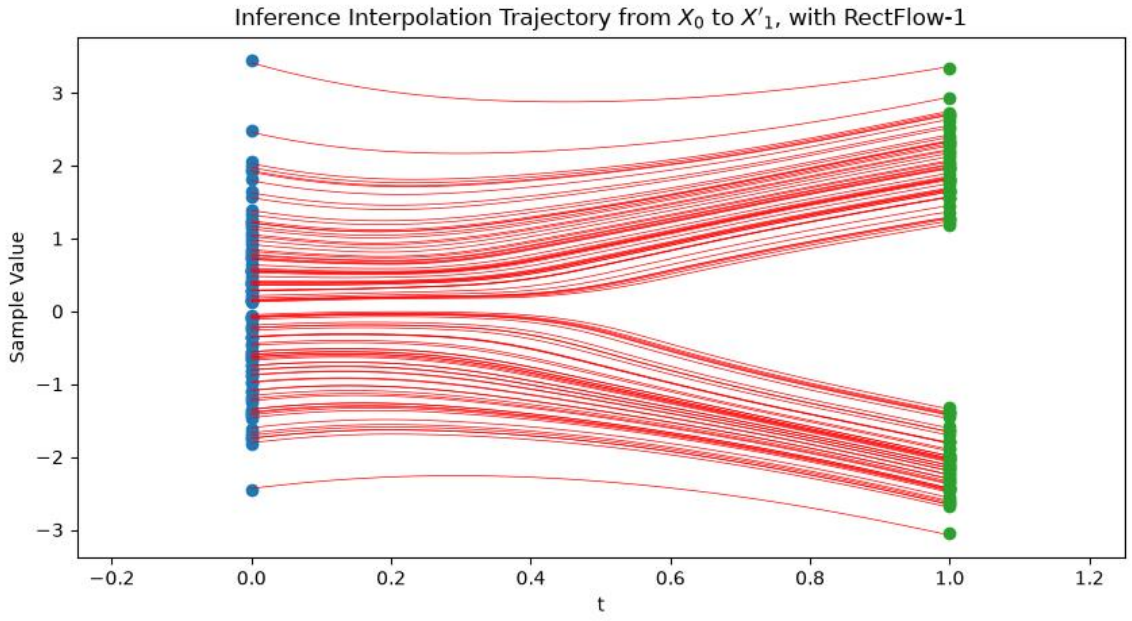
Use backward inference to refine the initial random sample

Then repeat using the new X_0 and the training dataset X_1

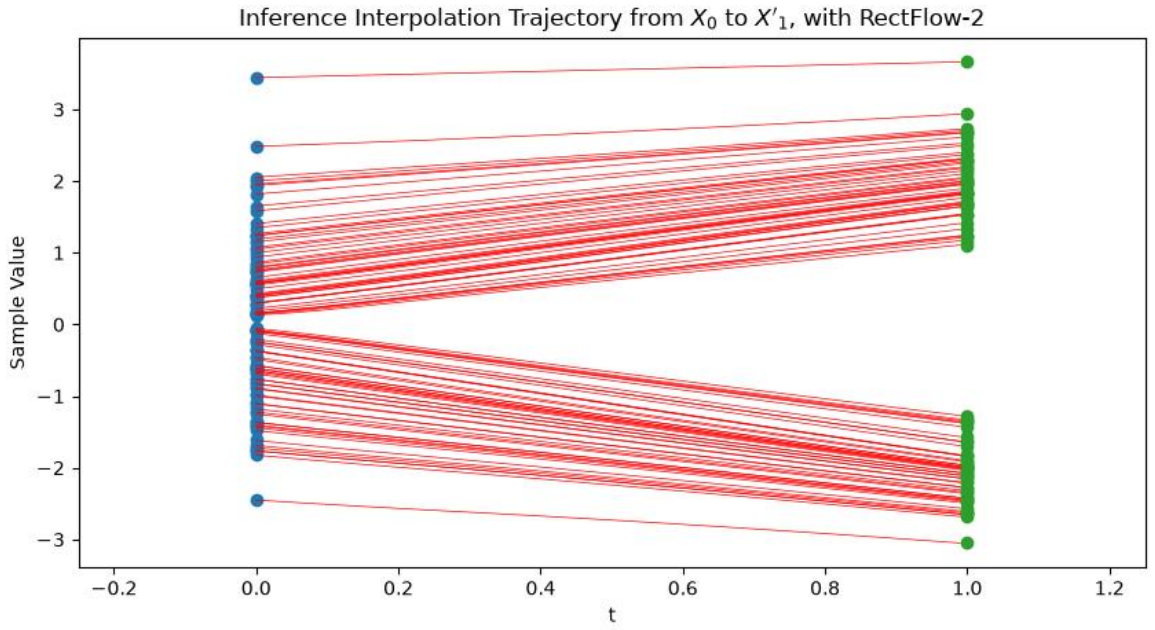




The Reflow ~~~~

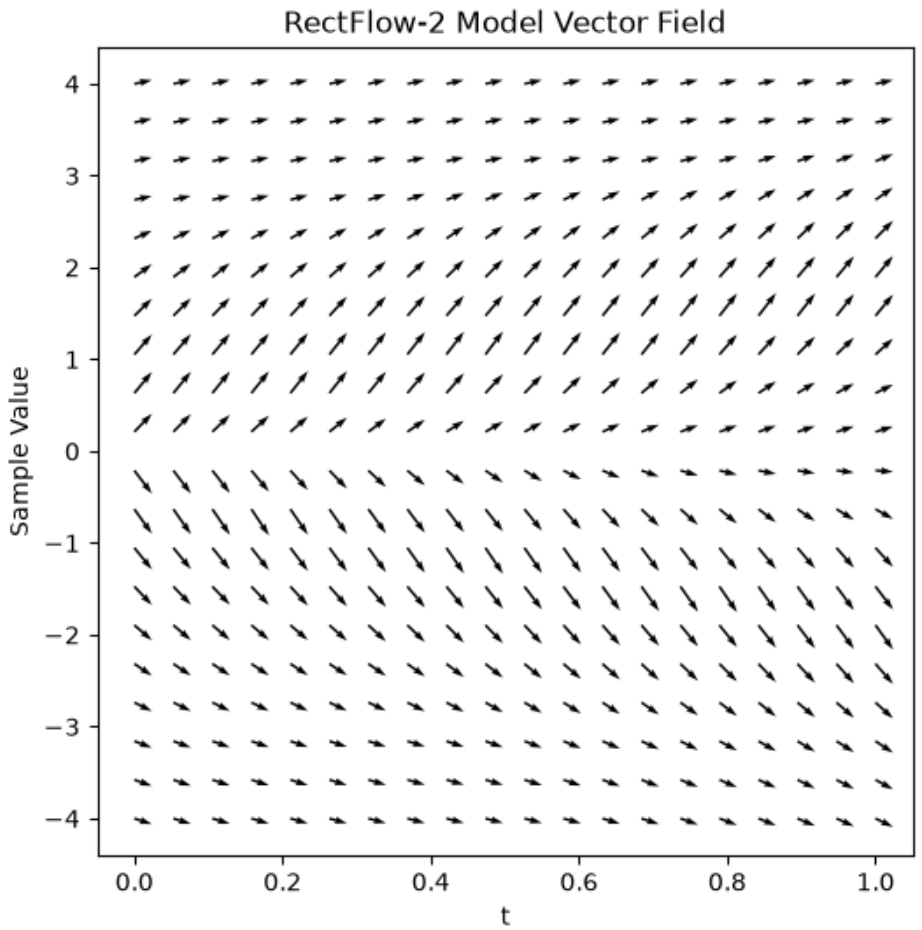


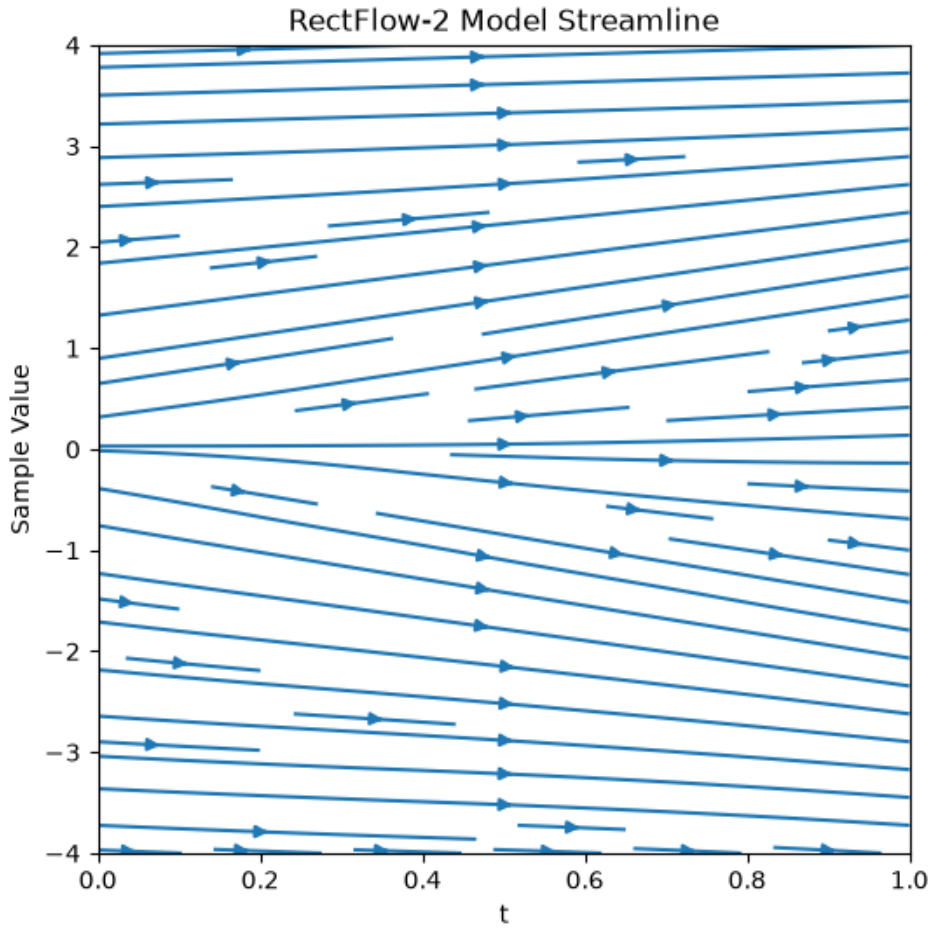
First Pass



Second Pass

The Reflow ~~~~


RectFlow-2 Model Vector Field
Sample Value
t
4
3
2
1
0
−1
−2
−3
−4
0.0
0.2
0.4
0.6
0.8
1.0


RectFlow-2 Model Streamline
Sample Value
t
4
3
2
1
0
−1
−2
−3
−4
0.0
0.2
0.4
0.6
0.8
1.0

MNIST Example

- Now you have the basics.
- Move on to the MNIST Example.

- Have Fun