Deep Learning Theory and Applications

Neural ODE

a kind of model useful for modeling dynamic system

Yale

CPSC/AMTH 663

CPSC 452

Ordinary Differential Equations 常微分方程

$$\frac{dx}{dt} + x = 1$$

- Describes the time-evolution or “dynamics” via its derivative
- “Solving a differential equation” refers to starting at some initial condition x(0) and evolving the system up to x(t)
- The solution is called the integral curve because it involves integrating the equation to get the solution 积分线

Initial Value problem

- Initial-value problem: given $\mathbf{z}(t_0)$, find:

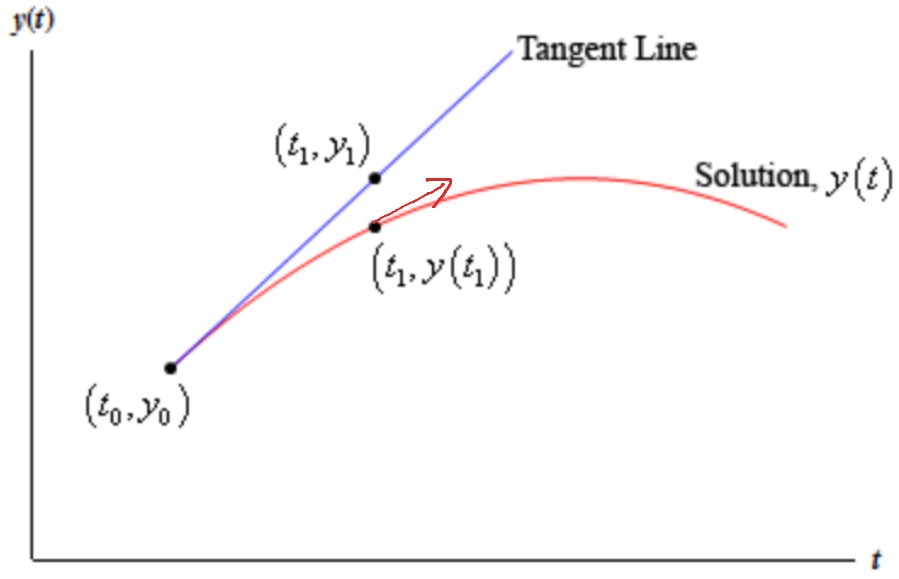
$$\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta) dt$$

value at time t_1 by integrating over derivative of variable z and time t and neural net param θ $f(z(t), t, \theta)$

initial value

Euler's Method

- One of the simplest methods for solving a differential equation
- Approximates the solution step by step using tangent lines



more steps , more accurate the approx of solution

Euler's Method

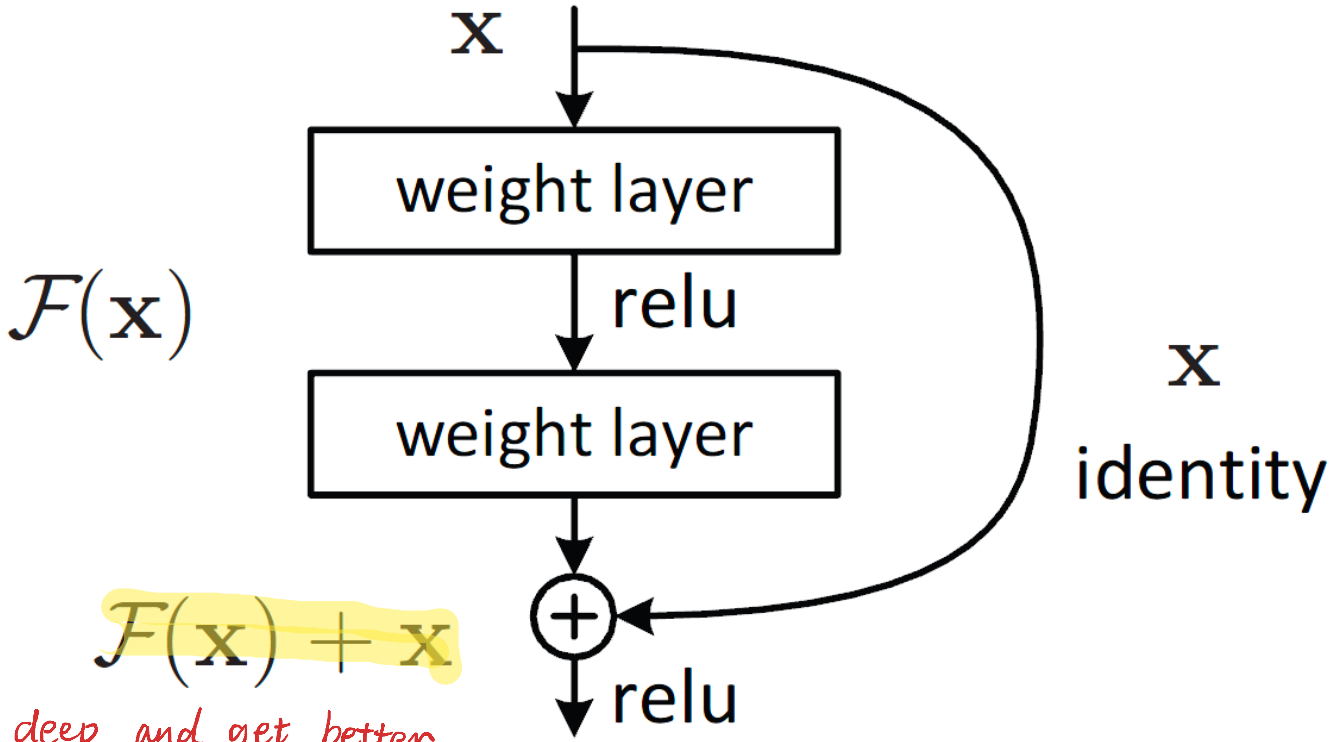
- Euler approximates with small steps:

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

Relate Residual NN (never used to model dynamics) to Euler's method to give a conceptial framework go beyond Resnet do sth else

Recall Resnets

h1 = f1(x) + x
h2 = f2(h1) + h1
h3 = f3(h2) + h2
h4 = f3(h3) + h3
y = f5(h4) + h4



$\mathcal{F}(\mathbf{x})$

$\mathcal{F}(\mathbf{x}) + \mathbf{x}$

ResNet : A neighboring NN go very deep and get better performance

How to compute more steps of dynamic ?
keep adding layers

Resnets as Euler Integrators

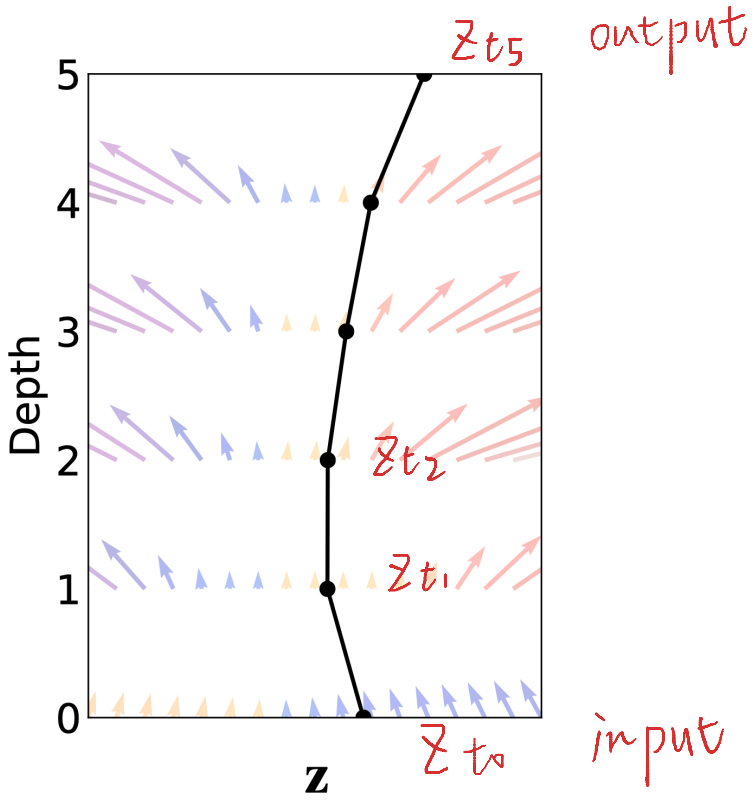
steps of dynamics = # layers = 5

```
def f(z, t, θ):
  return nnet(z, θ[t])

def resnet(z):
  for t in [1:T]:
    z = z + f(z, t, θ)
  return z
```

(layer in Resnet)

↑ residual (new derivative at time t)



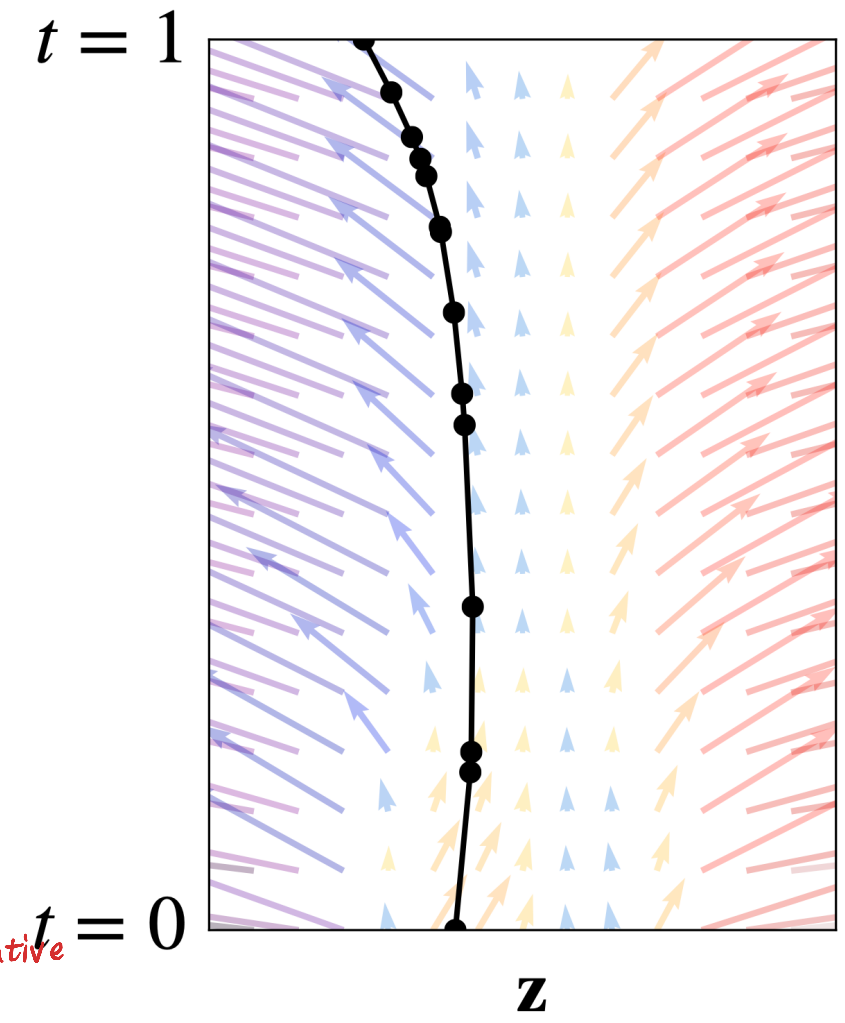
ODEnet

```
def f(z, t, θ):
  return nnet([z, t], θ)

def ODEnet(z, θ):
    return ODESolve(f, z, 0, 1, θ)

def ODESolve(f, z, t0, t1, θ):
    t = t0
    do
        g = f(z, t, θ)
        z = next_z(g, z, t, ...)
    while t < t1
    return z(t1)
```

loop from t0 to t1



1. Replace a resnet step with a call to an ODE solver to compute derivative
2. Internally the ODE solver evaluates dynamics wherever it wants

sample more times if derivative change a lot, Vice versa, No grid-like

How do we train ODE-net?

- ODE-net layers are calls to ODE solvers
- This means that in order to train ODE-net we would have to backpropagate through operations of the ODE-solver!

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

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

ODENet Paper propose

Avoiding Backprop through ODE Solver!

- Backpropagating through operations of the ODE solver are expensive
- 1. We would have to store intermediary operations of the ODE solver and this would be hard in terms of memory
- 2. This would build up numerical errors as well in practice
- Solution: approximate the differentiation through a solver
- Duvenaud: “Don’t differentiate the approximation, approximate the derivative!”

Adjoint Sensitivity to Compute Gradients

相邻的

Standard Backprop:

$$\frac{\partial L}{\partial \mathbf{z}_t} = \frac{\partial L}{\partial \mathbf{z}_{t+1}} \frac{\partial f(\mathbf{z}_t, \theta)}{\partial \mathbf{z}_t}$$

$$\frac{\partial L}{\partial \theta} = \sum_t \frac{\partial L}{\partial \mathbf{z}_t} \frac{\partial f(\mathbf{z}_t, \theta)}{\partial \theta}$$

Reverse mode differentiation through time steps

Adjoint sensitivities:
(Pontryagin et al., 1962):

$$\frac{\partial}{\partial t}\frac{\partial L}{\partial \mathbf{z}(t)} = \frac{\partial L}{\partial \mathbf{z}(t)} \frac{\partial f(\mathbf{z}(t), \theta)}{\partial \mathbf{z}}$$

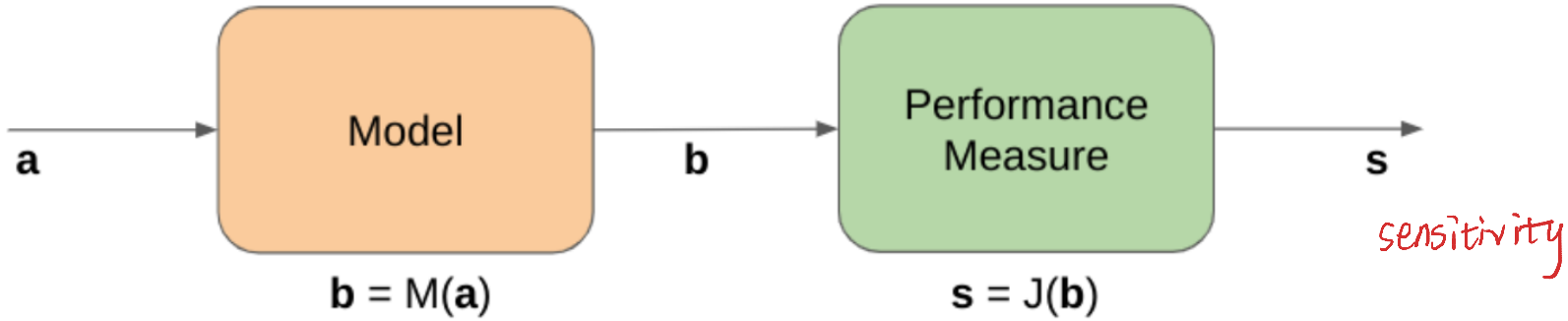
$$\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} \frac{\partial L}{\partial \mathbf{z}(t)} \frac{\partial f(\mathbf{z}(t), \theta)}{\partial \theta} dt$$

Continuous limit of this gives adjoint sensitivities

Meteorology

气象学

use ODE Net to
Predict tomorrow's weather given today weather
(humidity, temp etc)



sensitivity

- a=current weather (humidity, windspeed, temperature…)
- b=tomorrow's weather (same variables)

Computed using numerical integration

- Euler's method
- Runge Kutta

$$M(\mathbf{b}) = M_N\left(M_{N-1}\left(\cdots\left(M_2\left(M_1\left(M_0\left(\mathbf{a}\right)\right)\right)\right)\right)\right)$$

- Sensitivity of the result:
  - How much does the performance measure change if you perturb the input a little bit?

good: output don't change a lot

Sensitivity of a model

- If you give basically the same current information, you should get the same next-state information
- Imagine if you gave it 1mph lower windspeed and it said the temperature tomorrow was 50 degrees higher!
- Computing sensitivity:

bad model

$$\Delta \mathbf{a} = \mathbf{a}_p - \mathbf{a}_c$$

$$\Delta \mathbf{b} = \mathbf{b}_p - \mathbf{b}_c$$

$$\Delta s = s_p - s_c$$

Sensitivity Equation

$$\Delta s = \frac{\partial s}{\partial \mathbf{b}} \frac{\partial \mathbf{b}^{(N)}}{\partial \mathbf{b}^{(N-1)}} \frac{\partial \mathbf{b}^{(N-1)}}{\partial \mathbf{b}^{(N-2)}} \cdots \frac{\partial \mathbf{b}^{(0)}}{\partial \mathbf{a}} \Big|_{\substack{a=a_c \\ b^{(n-1)}=b_c^{(n-1)}}} \Delta \mathbf{a}$$

$$\frac{\partial s}{\partial \mathbf{a}} = \frac{\partial s}{\partial \mathbf{b}} \underbrace{\frac{\partial \mathbf{b}^{(N)}}{\partial \mathbf{b}^{(N-1)}} \frac{\partial \mathbf{b}^{(N-1)}}{\partial \mathbf{b}^{(N-2)}} \cdots \frac{\partial \mathbf{b}^{(0)}}{\partial \mathbf{a}}}_{\partial \mathbf{b}/\partial \mathbf{a}}$$

$$\left(\frac{\partial s}{\partial \mathbf{a}}\right)^T = \left(\frac{\partial s}{\partial \mathbf{b}} \frac{\partial \mathbf{b}}{\partial \mathbf{a}}\right)^T = \left(\frac{\partial \mathbf{b}}{\partial \mathbf{a}}\right)^T \left(\frac{\partial s}{\partial \mathbf{b}}\right)^T$$

As a constrained optimization

constrain is initial state and intermediate state

$$\min_{\mathbf{p}} \; J(\mathbf{b}^{(N)}, \mathbf{p})$$

$$\text{subject to: } \mathbf{b}^{(n)} = M(\mathbf{b}^{(n-1)}, \mathbf{p})$$

$$\mathbf{b}^{(-1)} = \mathbf{a}_c$$

Discrete Time

$$\min_{\mathbf{p}} \; G(\mathbf{x}, \mathbf{p}) = \int_0^T g(\mathbf{x}, t, \mathbf{p}) dt$$

$$\text{subject to: } F(\mathbf{x}, \dot{\mathbf{x}}, t, \mathbf{p}) = 0$$

$$\mathbf{x}(\mathbf{0}) = \mathbf{x}_0(\mathbf{p})$$

$$\text{where } \mathbf{x} \in R^{n_x}, \mathbf{p} \in R^{n_p}$$

Continuous time, this can be solved
By an ODE solver it is a system of
Differential algebraic equations!

Low Memory Costs

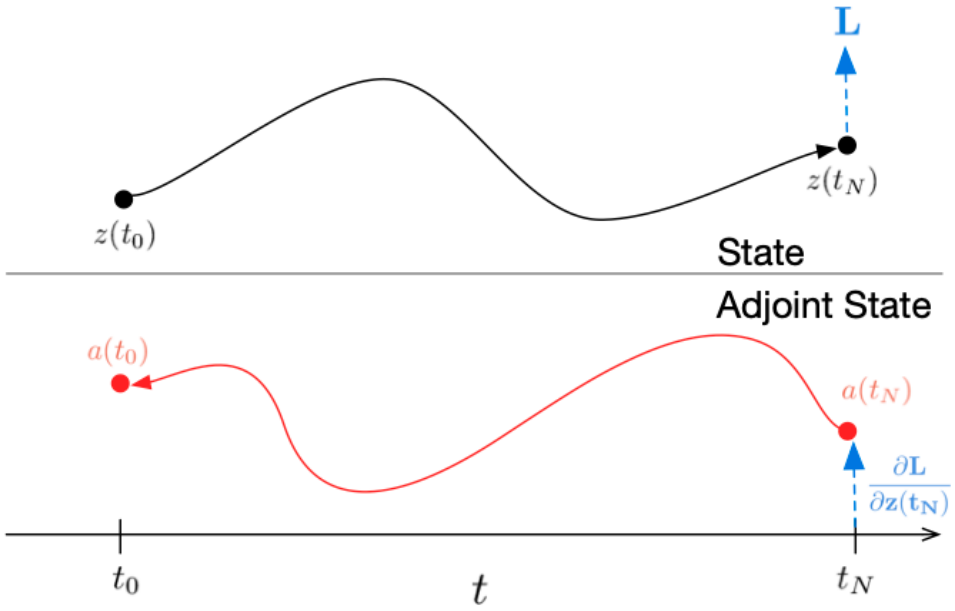
Adj of Neural ODE

can infer dynamics w/ irregular sampling eg. any time point

Recall ResODE : $t_0 \to t_1 \to t_2 \to \cdots \to t_5$

- No longer need to store intermediate values in the continuous case
- You can reconstruct them as you go along by running the system in reverse!

bc states are deterministic



Adjoint Sensitivity in Neural ODE

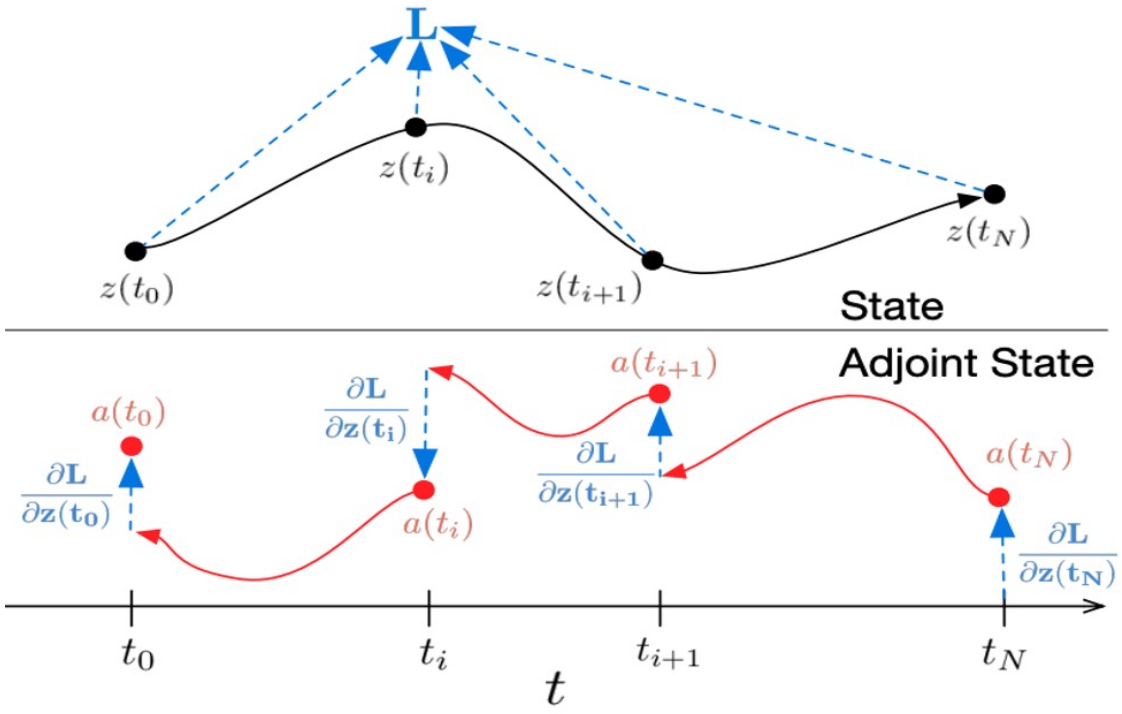
$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{.}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

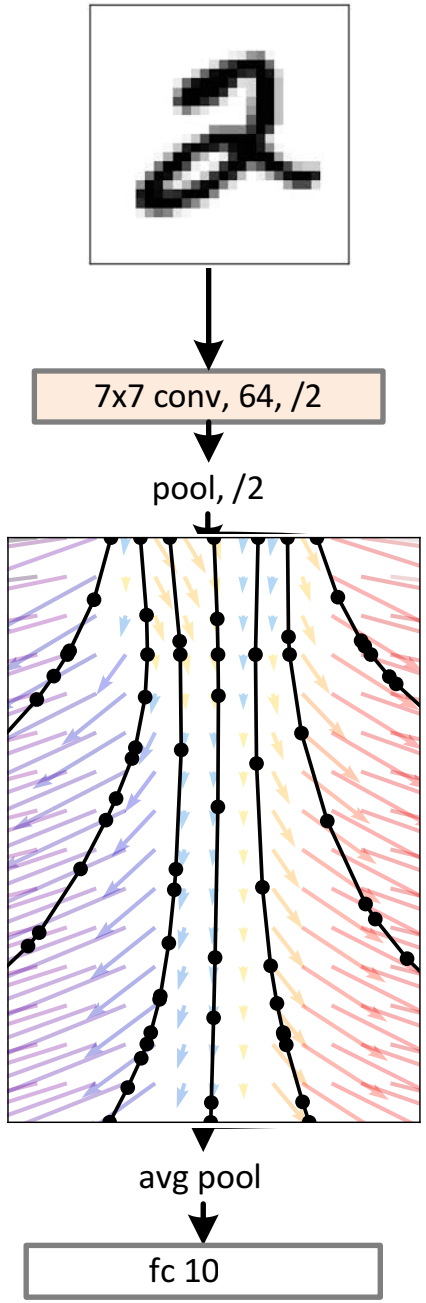
$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$$

$$\mathbf{a}(t) = {}^{\partial L}/_{\partial \mathbf{z}(t)}$$

Adjoint = derivative of Loss, Hidden State or intermediate timepoint

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$





Drop-in replacement for Resnets

- Same optimization algorithm (SGD)
- Same test performance.

baseline →

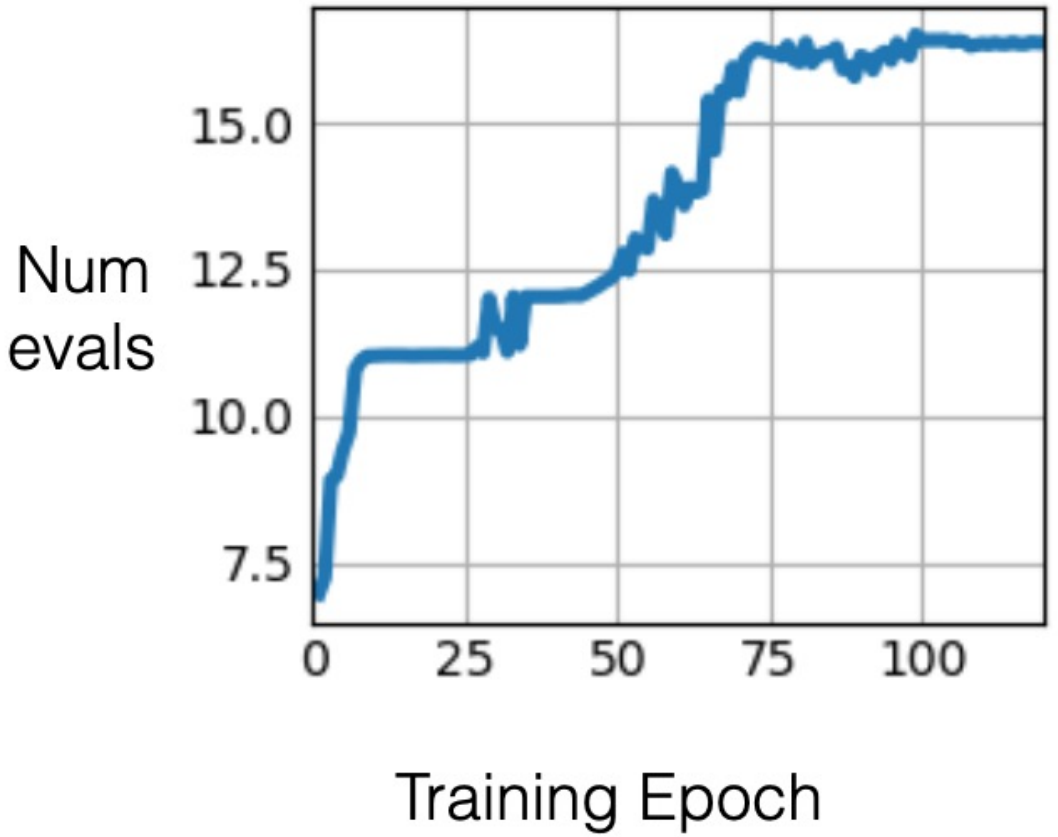
	Test Error
1-Layer MLP	1.60%
ResNet	0.41%
ODE-Net	0.42%

How deep are ODE-nets?

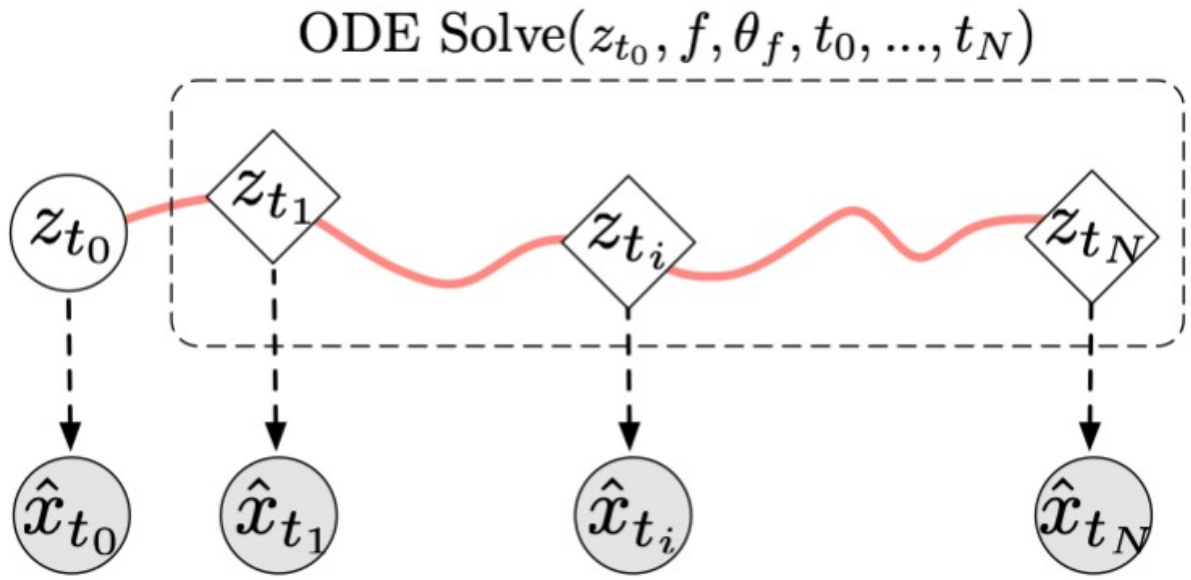
Infinitely deep

- 'Depth' is left to ODE solver.
- Dynamics become more demanding during training
- 2-4x the depth of resnet architectures
- Chang et al. (2018) build such a schedule by hand

easy to train than ResNet



Continuous-time models



$$\mathbf{z}_{t_0} \sim p(\mathbf{z}_{t_0})$$
$$\mathbf{z}_{t_1}, \mathbf{z}_{t_2}, \ldots, \mathbf{z}_{t_N} = \text{ODESolve}(\mathbf{z}_{t_0}, f, \theta_f, t_0, \ldots, t_N)$$
$$\text{each} \quad \mathbf{x}_{t_i} \sim p(\mathbf{x}|\mathbf{z}_{t_i}, \theta_{\mathbf{x}})$$

- Well-defined state at all times
- Dynamics separate from inference
- Irregularly-timed observations.

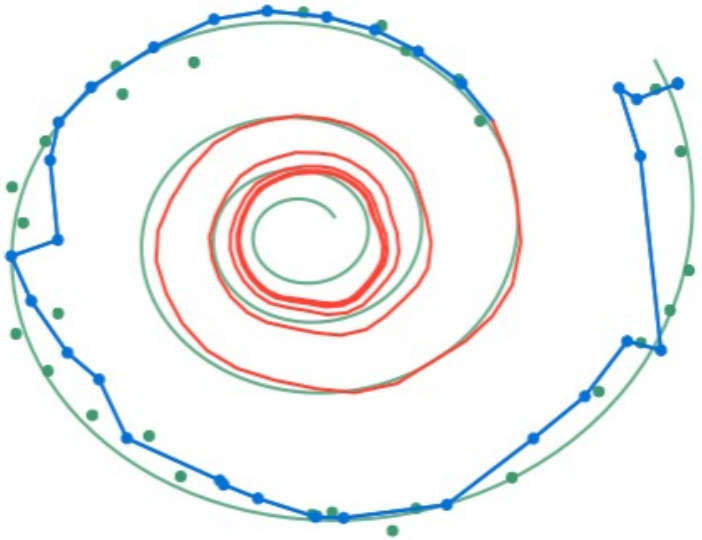
can be patient visits

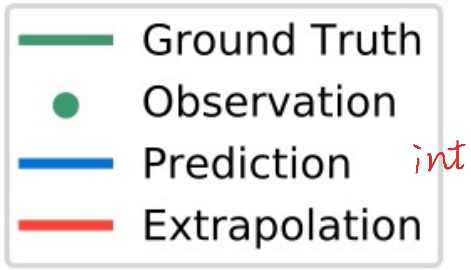
Continuous-time models

Recurrent Neural Net

discrete time point

use truncate cut-off 4 hours (fixed bin size)

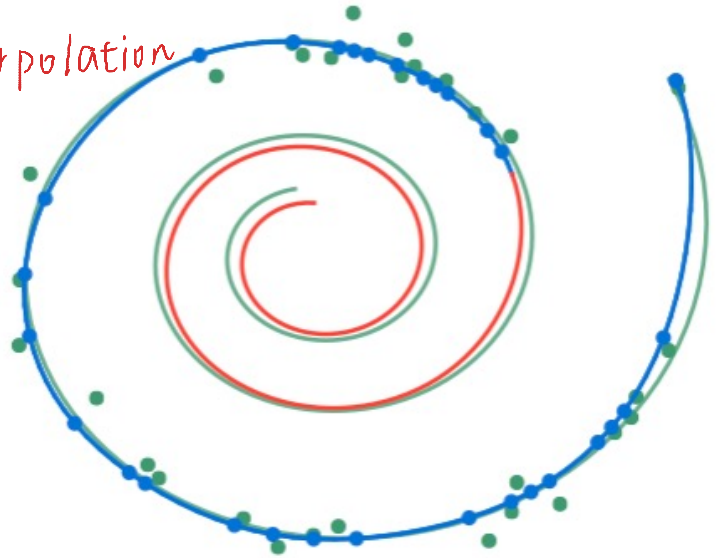




Latent ODE

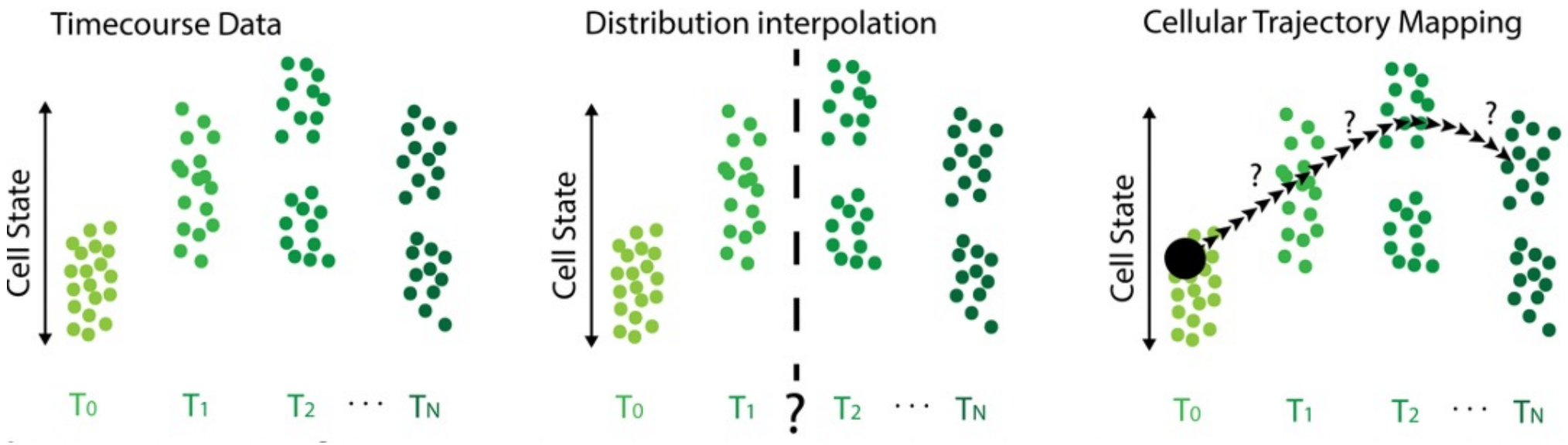
no bin , can sample any time point

interpolation

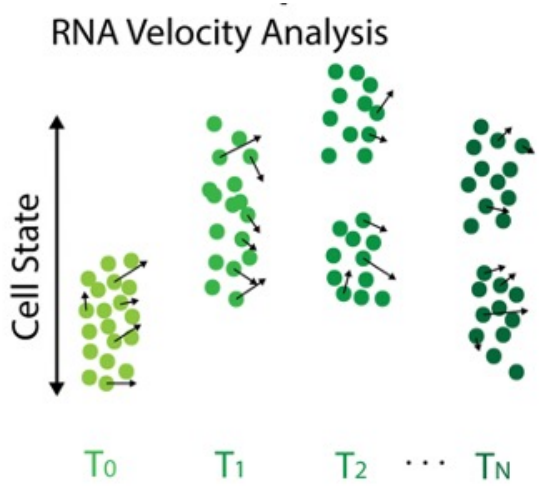


Population Dynamics

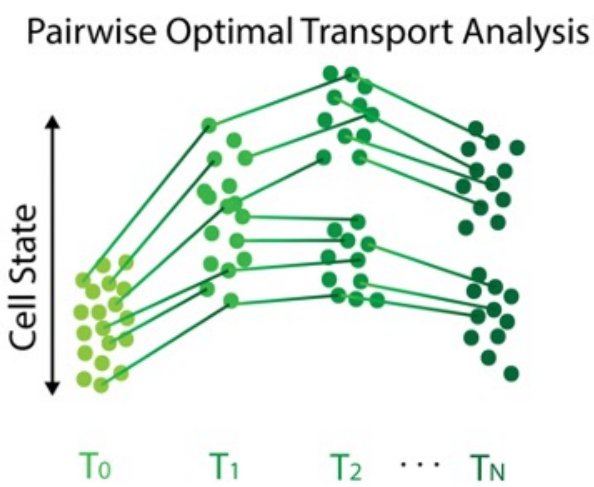
Dynamics from Static Snapshot Datasets



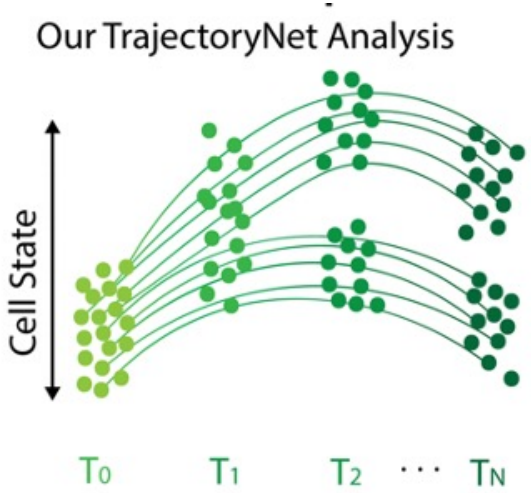
Comparison to Previous Methods



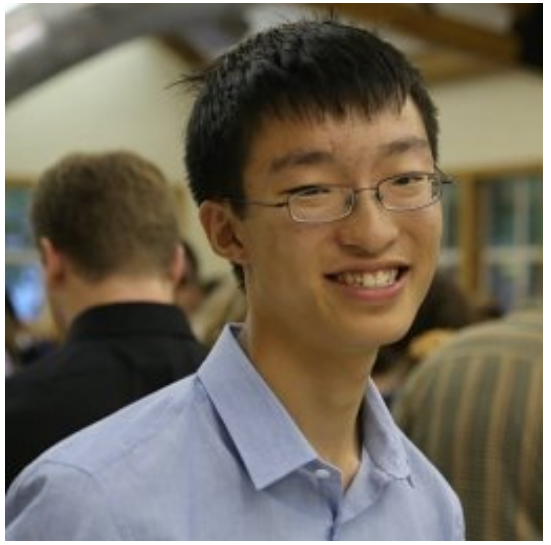
La Manno et al.
Nature



Schiebinger et al.
Cell



Tong et al.
ICML



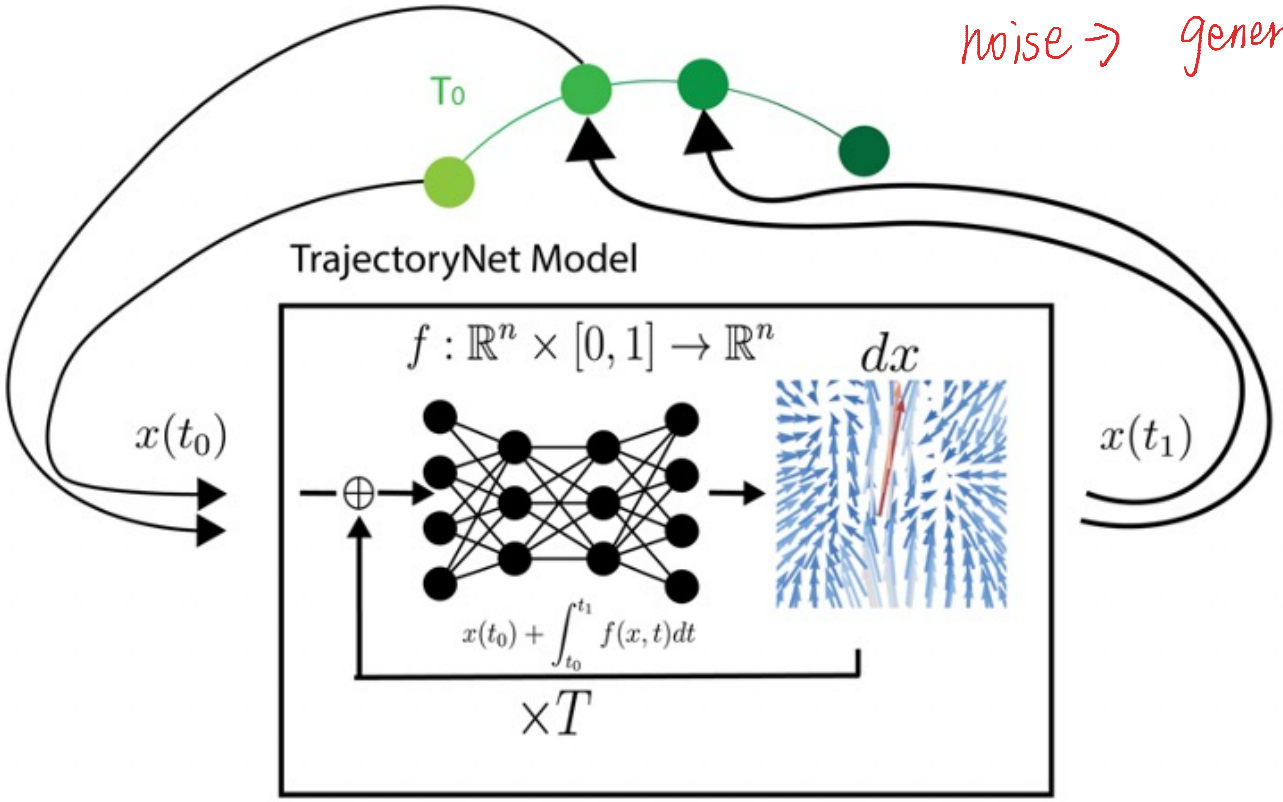
TrajectoryNet: A Dynamic Optimal Transport Network for Modeling Cellular Dynamics

Alexander Tong, Jessie Huang, Guy Wolf, David Van Dijk, Smita Krishnaswamy *Proceedings of the 37th International Conference on Machine Learning*, PMLR 119:9526-9536, 2020.

Neural ODE for Population Flow

based on normalizing flow

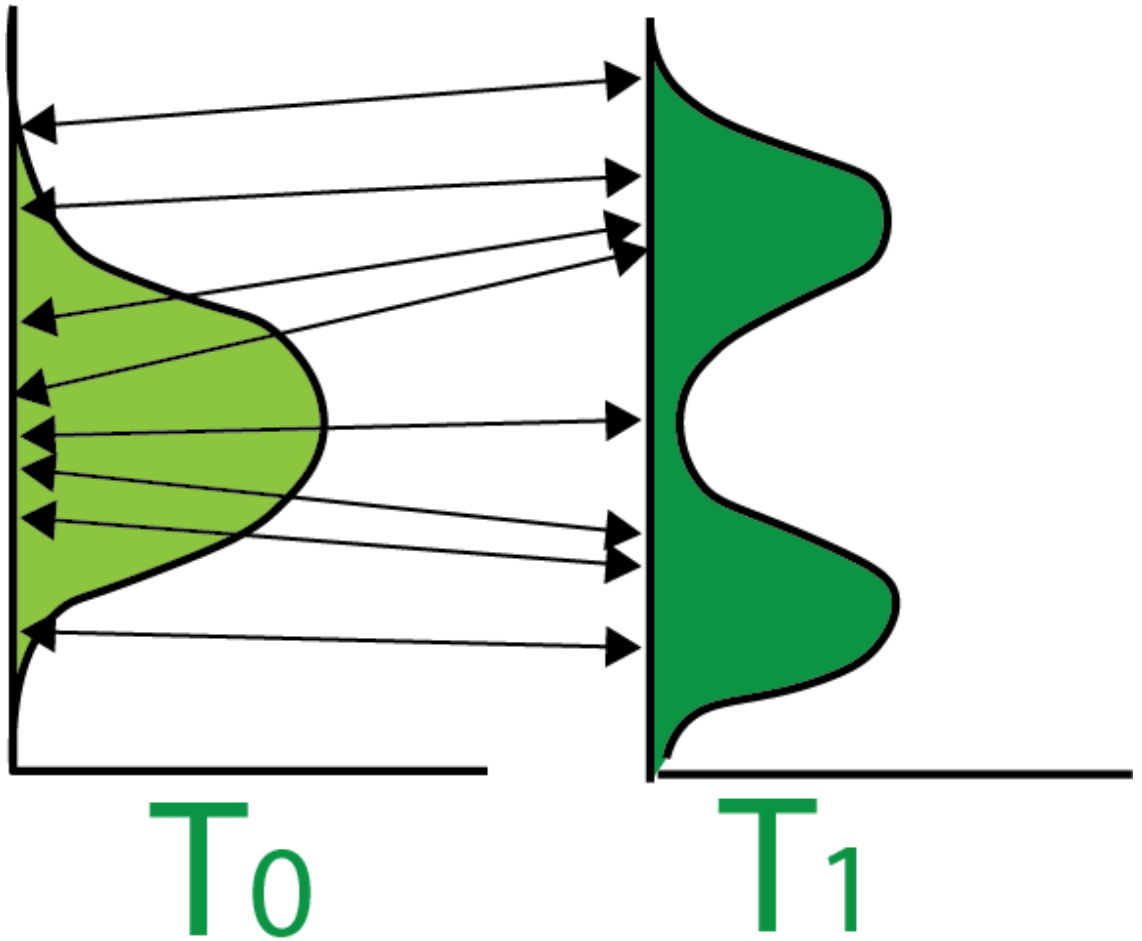
noise → generative model → signal



Normalizing Flows (NFs)

Gaussian dist → harder dist

- Begin with a simple distribution
  $p_{t_0}(x) \sim \mathcal{N}(0, 1)$
  Gaussian
- Apply an invertible transformation(s)
  $x_{t_1} = f(x_{t_0})$
- Use change of variables to calculate probability

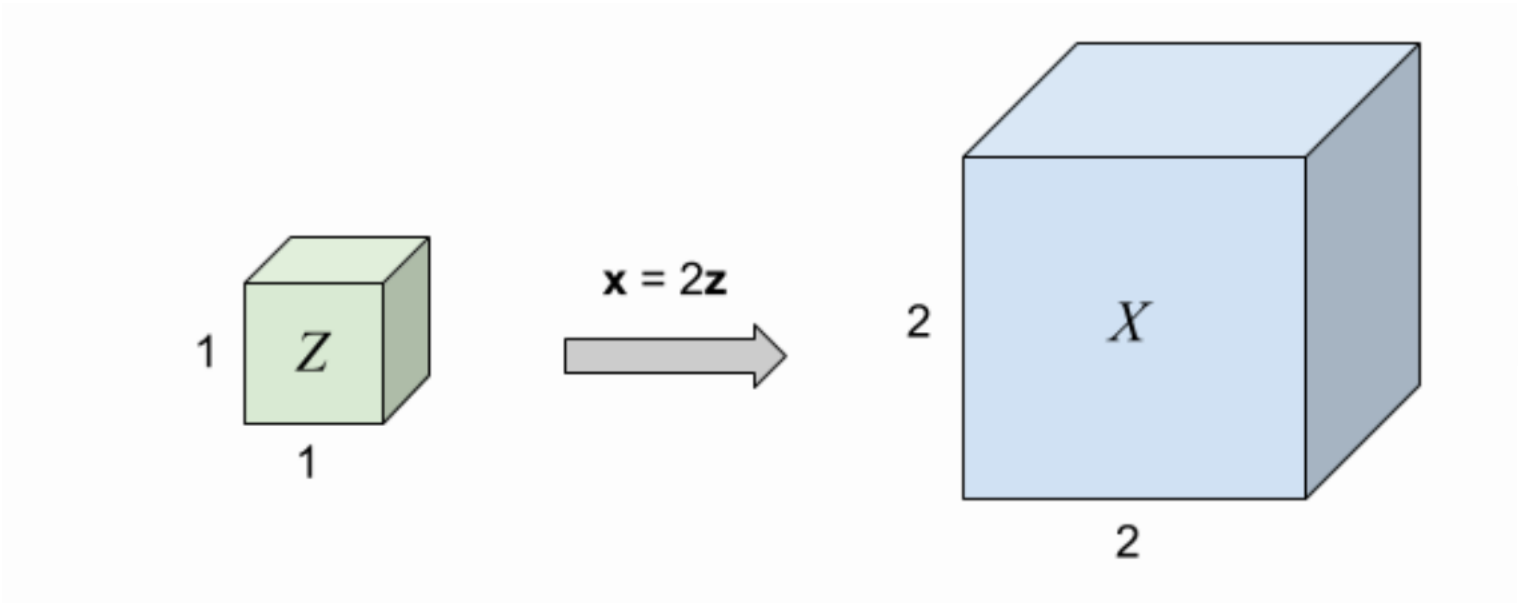


maximize

$$\log p_{t_1}(x_{t_1}) = \log p_{t_0}(x_{t_0}) - \log \det \left| \frac{\partial f}{\partial x_{t_0}} \right|$$

log determinant of derivative of transformation f

Change of variables formula: volume normalization



$$p_X(\mathbf{x}) = \frac{p_Z(\mathbf{z})V_Z}{V_X} = \frac{1}{8}$$

determinant of gradient

$$p_X(\mathbf{x}) = p_Z(\mathbf{z}) \left| \det \left(\frac{\partial f^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right| = p_Z(\mathbf{z}) \left| \det \left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right) \right|^{-1}$$

Deep Normalizing Flows (NFs)

- Apply a series of transformations

$$x_{t_1} = f(x_{t_0}) \quad \Rightarrow \quad x_{t_N} = f_N \circ f_{N-1} \circ \cdots \circ f_1(x_{t_0})$$

- Use change of variables to calculate probability

$$\log p_{t_1}(x_{t_1}) = \log p_{t_0}(x_{t_0}) - \log \det \left| \frac{\partial f}{\partial x_{t_0}} \right| \quad \Rightarrow \quad \log p_{t_N}(x_{t_N}) = \log p_{t_0}(x_{t_0}) - \sum_{n=1}^{N} \log \det \left| \frac{\partial f_n}{\partial x_{t_{n-1}}} \right|$$

Continuous Normalizing Flows

$$x_{t_N} = f_N \circ f_{N-1} \circ \cdots \circ f_1(x_{t_0}) \quad \Rightarrow \quad x_{t_1} = F(x_{t_0}) = \int_{t_0}^{t_1} f(x(t), t)\mathrm{d}t$$

change setting from discrete to continuous

determinant → trace = Σ diagonal entry

$$\log p_{t_N}(x_{t_N}) = \log p_{t_0}(x_{t_0}) - \sum_{n=1}^{N} \log \det \left| \frac{\partial f_n}{\partial x_{t_{n-1}}} \right| \quad \Rightarrow \quad \log p_{t_1}(x_{t_1}) = \log p_{t_0}(x_{t_0}) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f}{\partial x(t)} \right) \mathrm{d}t$$

save computation

Instantaneous change of variables

$$\det(e^A) = e^{\mathrm{tr}(A)}$$

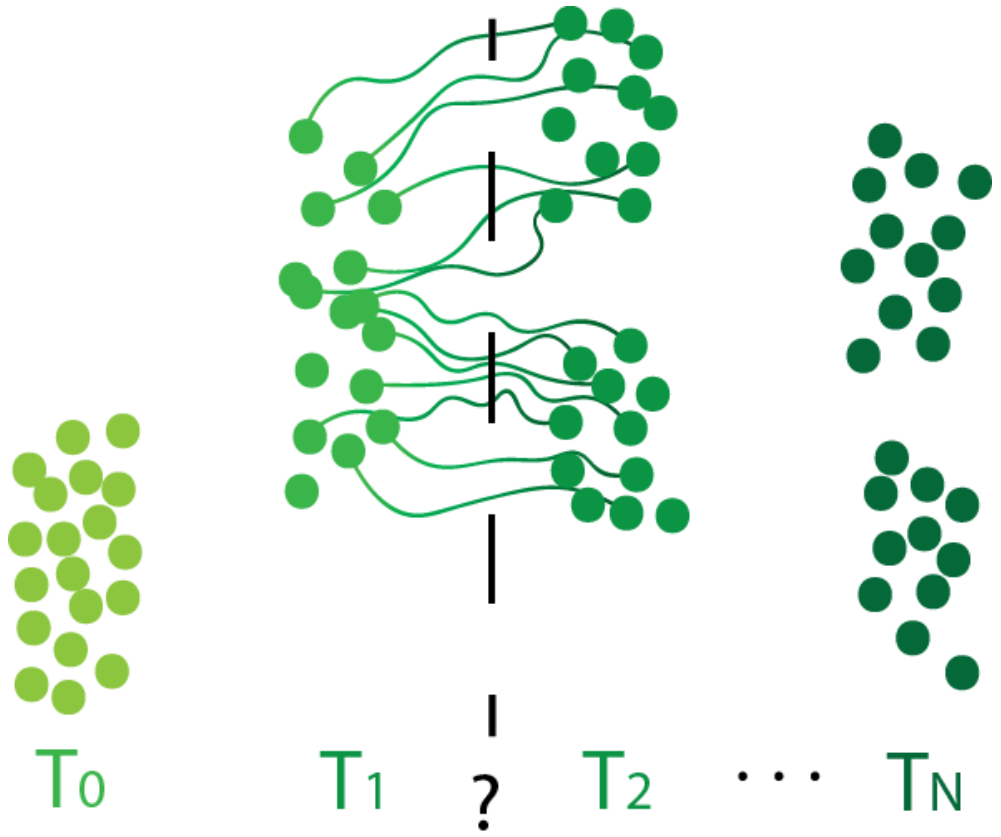
Cannot model:

$$F(x) = -x$$

add: ODENet layer is naturally reversible

[Chen et al. 2018]

CNFs create continuous paths



Creates continuous paths, but they may not be biologically plausible — no restriction on circuitous paths!

迂回的

[Chen et al. 2018]

l2-norm

Obtain More Realistic Paths via Regularization

we want derivative change less

rather than

Dynamic Optimal Transport

Dynamic OT:

$$W(\mu,\nu)_2^2 = \underbrace{\inf_{p,f} \int_{\mathbb{R}^d} \int_0^1 p(x,t)\,\|f(x,t)\|\,\mathrm{d}t\mathrm{d}x}_{\text{Eulerian frame}} = \underbrace{\inf_{p,f} \mathop{\mathbb{E}}_{x_0\sim\mu} \int_0^1 \|f(x(t),t)\|\,\mathrm{d}t}_{\text{Lagrangian frame}}$$

Subject to: μ ν

$$\underbrace{\partial_t p + \nabla\cdot(pf) = 0}_{\text{Mass is preserved}}$$

$$p(\cdot,0) = \mu \qquad p(\cdot,1) = \nu$$

Benamou and Brenier 2000

Regularized CNF approximates dynamic optimal transport

~~Dynamic OT:~~ regularized CNF (continuous normalized flow)

$$W(\mu,\nu)_2^2 = \underbrace{\inf_{p,f} \int_{\mathbb{R}^d} \int_0^1 p(x,t)\,\|f(x,t)\|\,\mathrm{d}t\mathrm{d}x}_{\text{Eulerian frame}} = \underbrace{\inf_{p,f} \mathbb{E}_{x_0\sim\mu} \int_0^1 \|f(x(t),t)\|\,\mathrm{d}t}_{\text{Lagrangian frame}}$$

$$+ \mathrm{KL}(p(\cdot,1)||\nu)$$

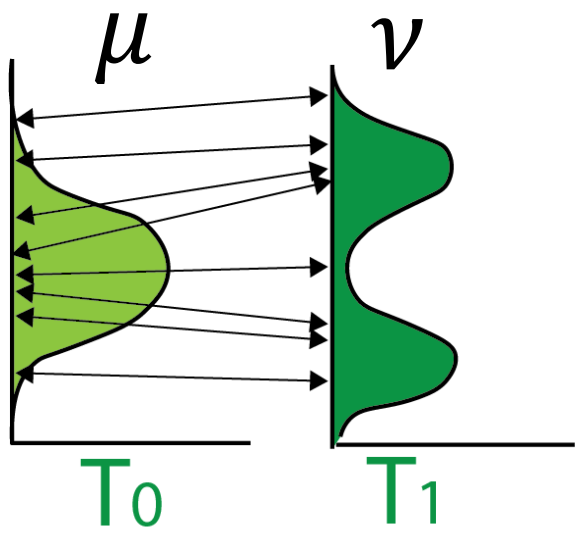
KL divergence

penalize path for magnitude of derivative that give path

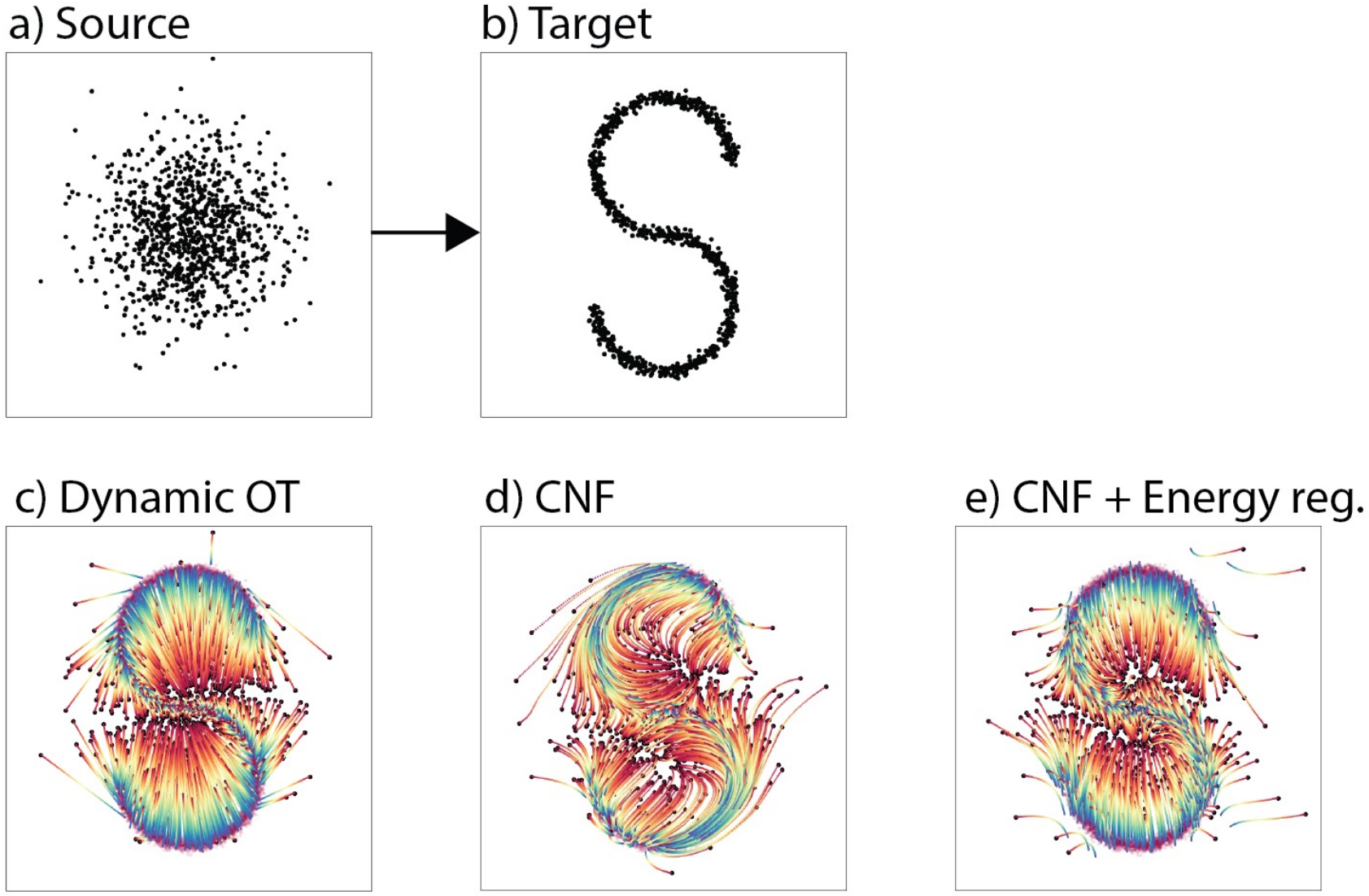
Subject to:

$$\underbrace{\partial_t p + \nabla\cdot(pf) = 0}_{\text{Mass is preserved}}$$

$$p(\cdot,0) = \mu \qquad \sout{p(\cdot,1) = \nu}$$



CNFs model Dynamic Optimal Transport



Dynamic OT via TrajectoryNet

- Dynamic OT via TrajectoryNet can be utilized to infer continuous trajectories of any populations adhering to energy or transport constraints
  - Population migration
  - Disease spread

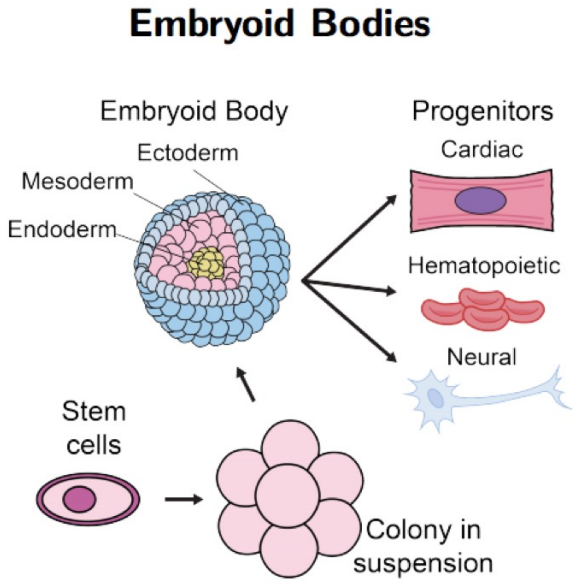
- However, cellular systems are more constrained, and other domain specific priors apply

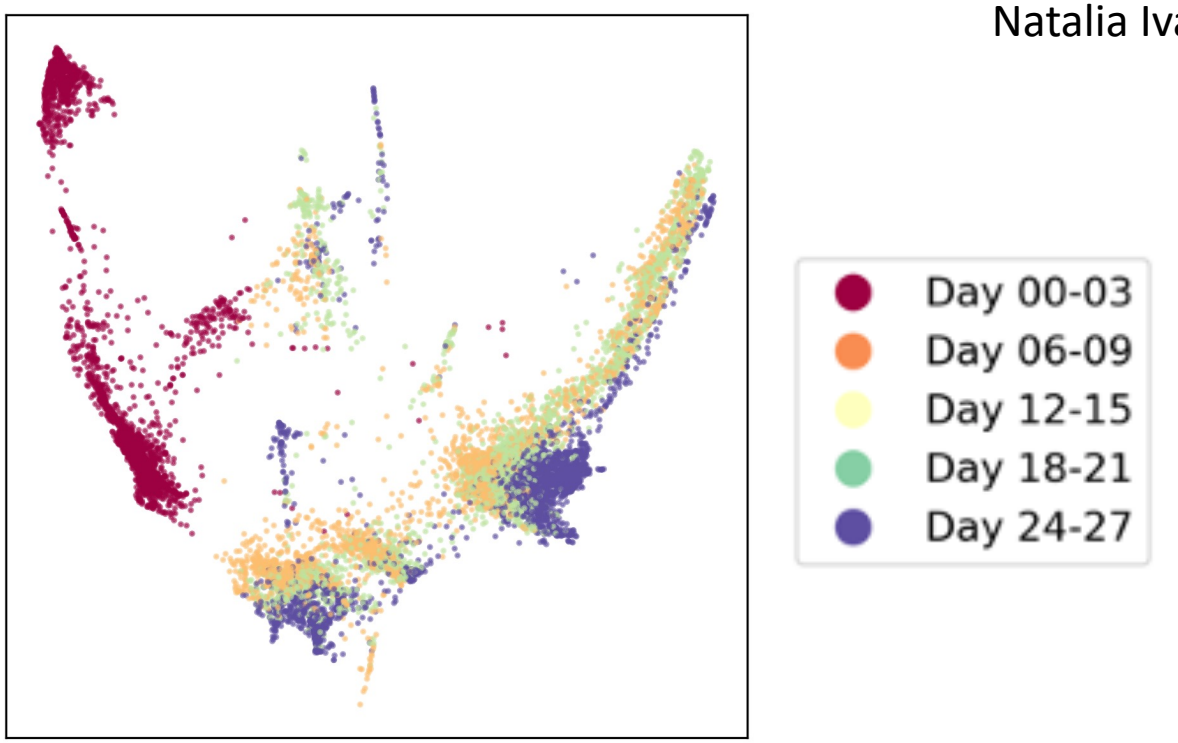
add constraint in your system to Net as regularization

to help customize your flow to fit your problem

PHATE: Embryonic Stem Cell Data

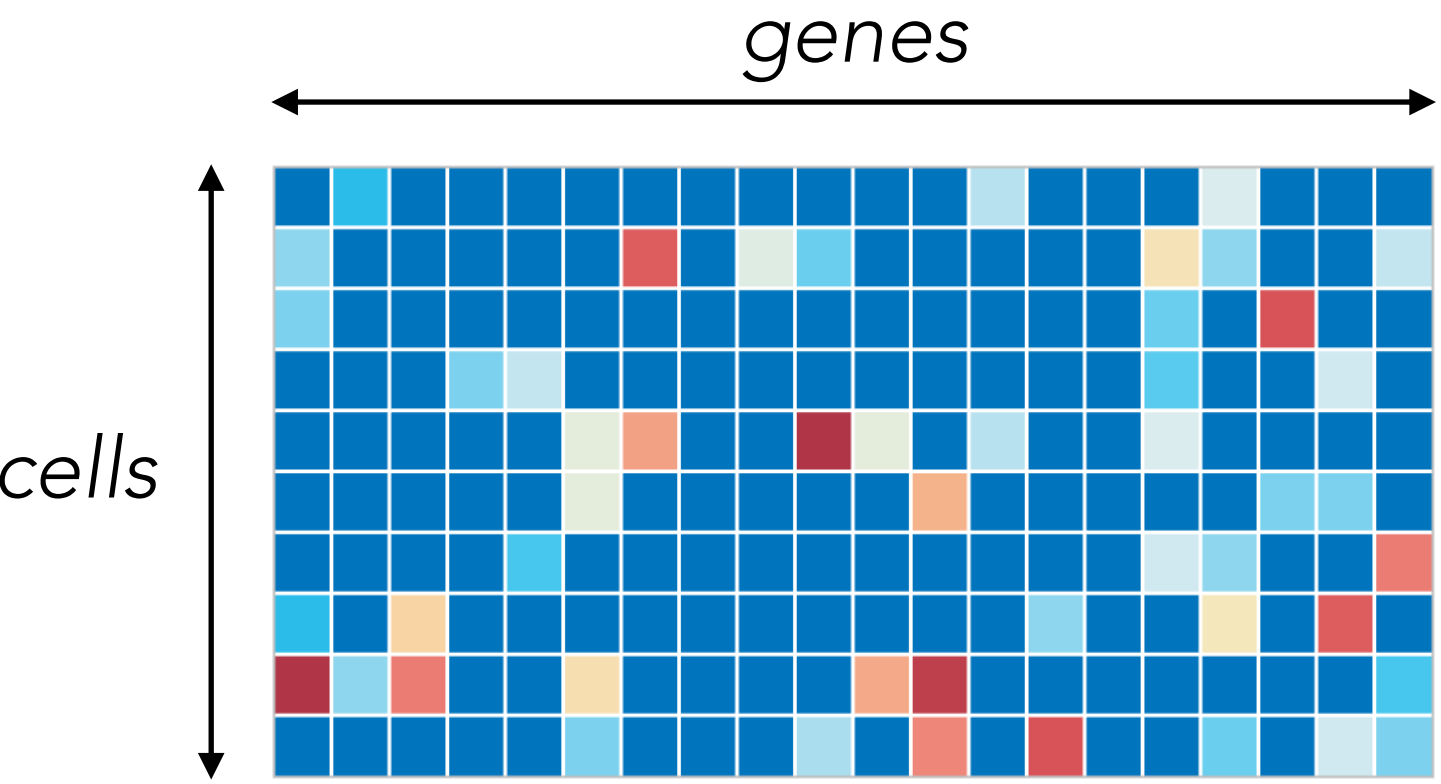
Natalia Ivanova

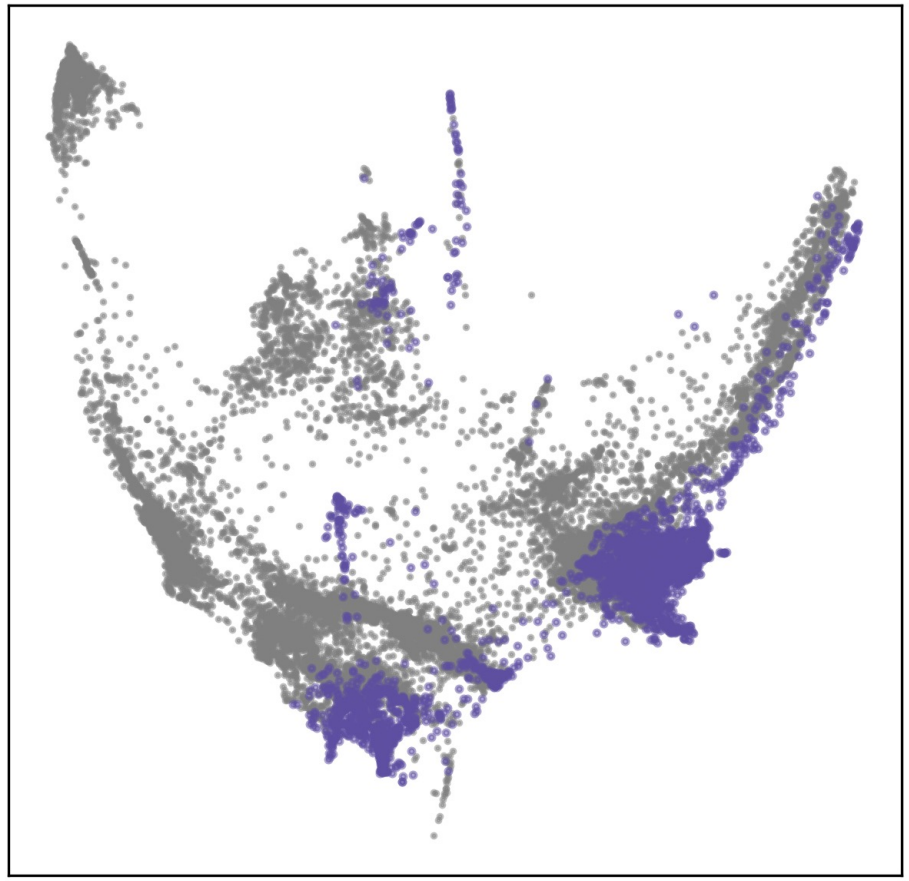




27 day timecourse collected at 5 timepoints
[Moon et al. Nature Biotechology 2019]

Inferring Continuous Flow in Static Snapshots





Additional Properties of cells

1. Cells are not simply transported from one timepoint to another, ***they cells divide and die.*** growth / death rate
2. Cells cannot travel in straight paths through Euclidean space in terms of measured dimensions, ***cells only travel along a cellular manifold.***
3. Though cells are destroyed when measured, we can estimate their direction of transition-based ***RNA velocity***

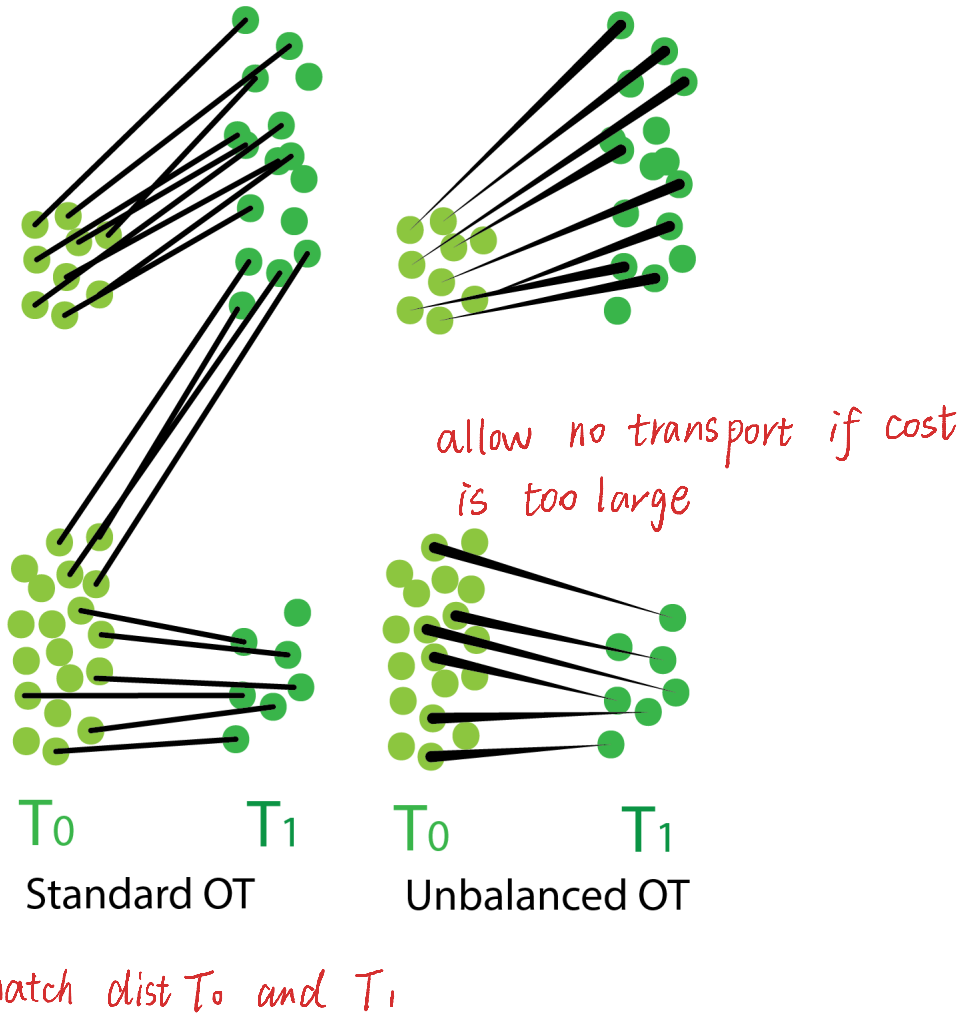
old RNA transcript will degrade and new RNA transcript will take over

Cell Death and Growth

Allow Unbalanced Transport

optimal transport is balanced

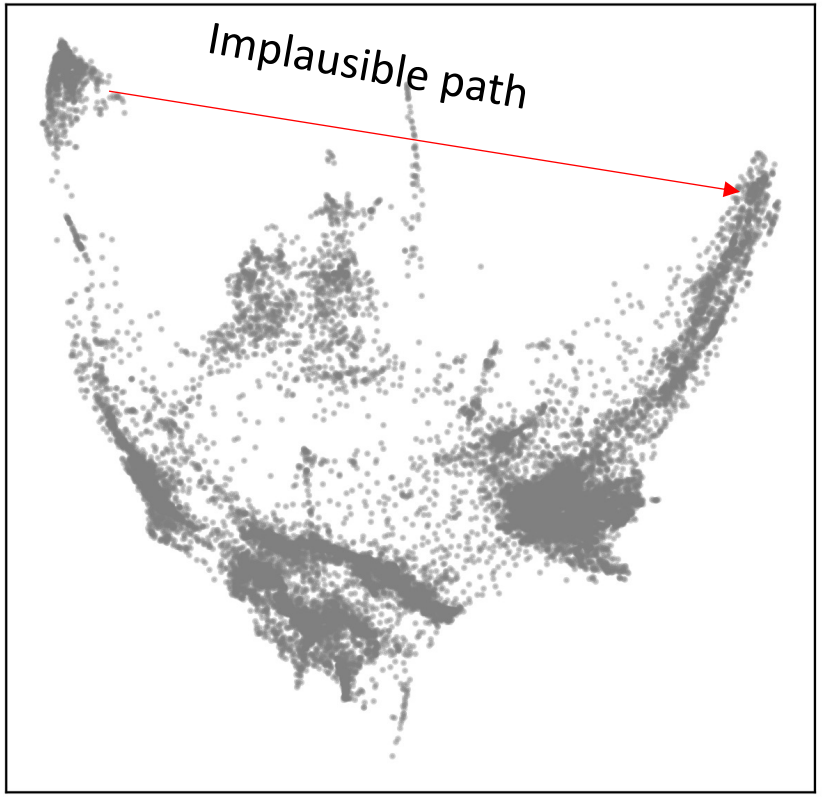
- Allowing unbalanced transport can let cells “die” instead of moving them to implausible locations
- Unbalanced transport hard to achieve dynamically
- We use discrete optimal transport to assign growth and death rates



hard to match dist T_0 and T_1

Liero et al. 2018

Cellular Manifolds



Cells have to transition through allowable parts of the state space

Enforce this with a density penalty. Based on a knn density estimate

$$L_{density}(x,t) = \sum_k \max(0, \text{min-k}(\{\|x(t) - z\| : z \in \mathcal{X}\}) - h)$$

encourage cells to transition through neighboring dense region

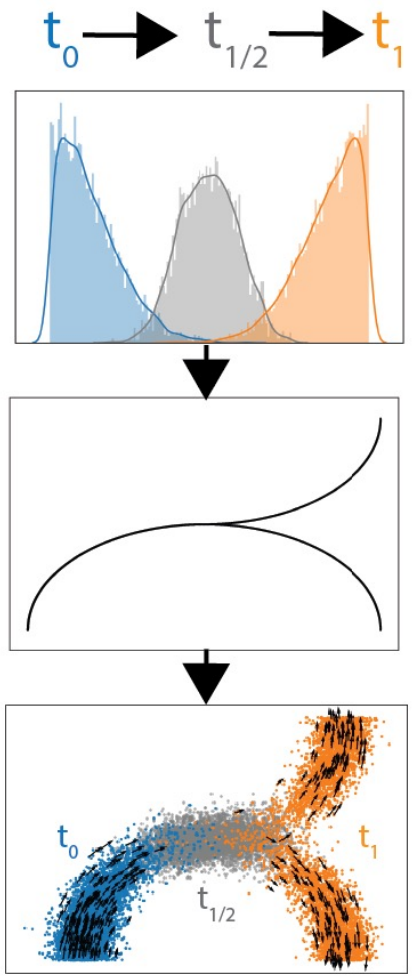
Velocity Regularization

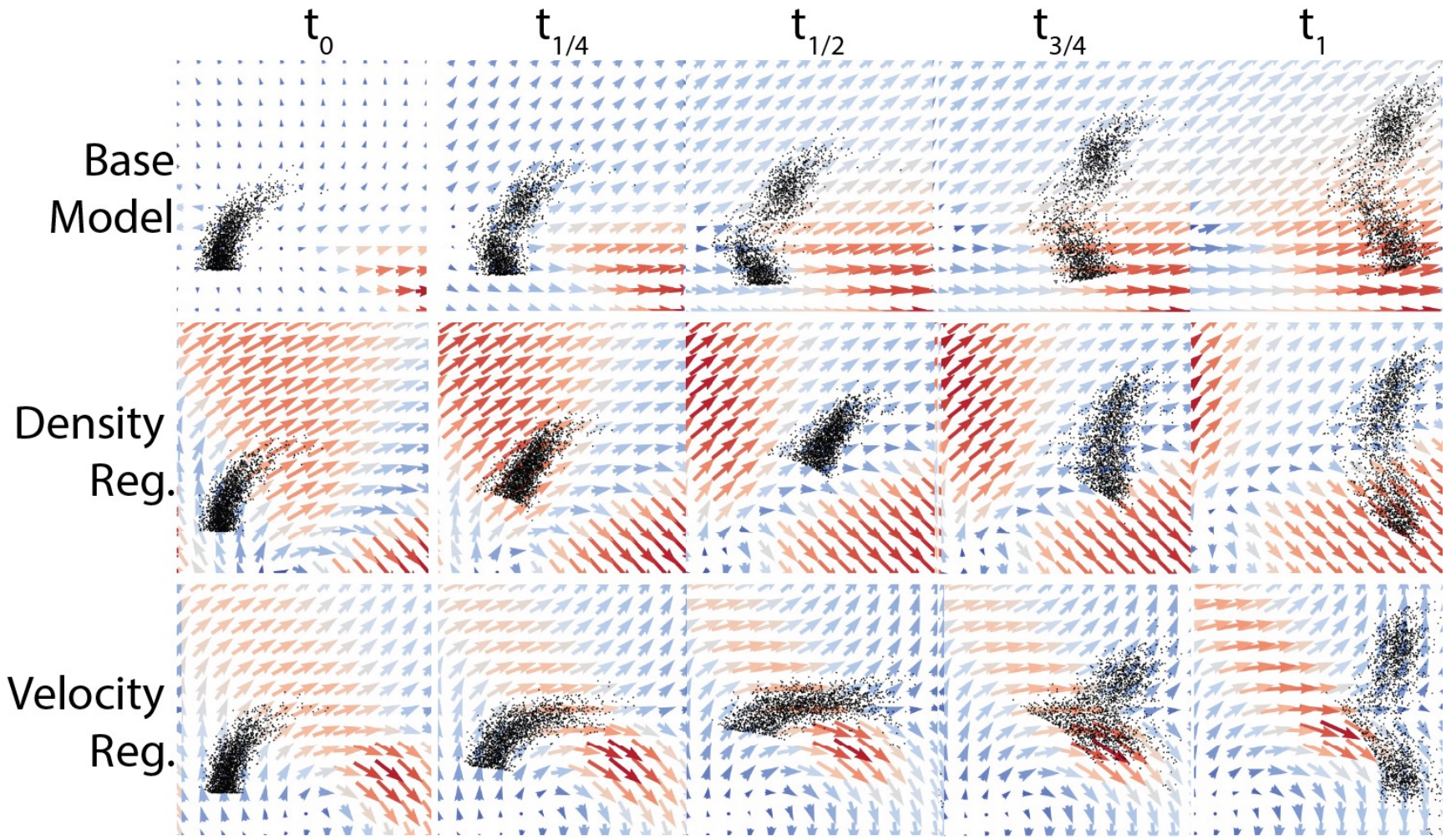
RNA Velocity, estimate of direction of change

[La Manno et al. 2018 Velocyto; Volker et al. ScVelo]

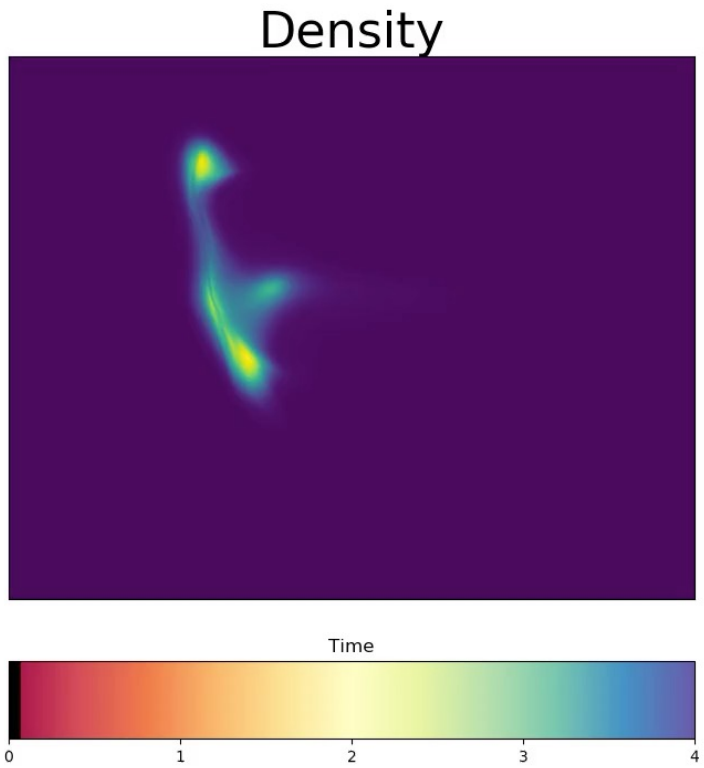
$$(x, t, \widehat{dx/dt}) = \frac{f(x,t) \cdot \widehat{dx/dt}}{\|f(x,t)\| \left\|\widehat{dx/dt}\right\|}$$

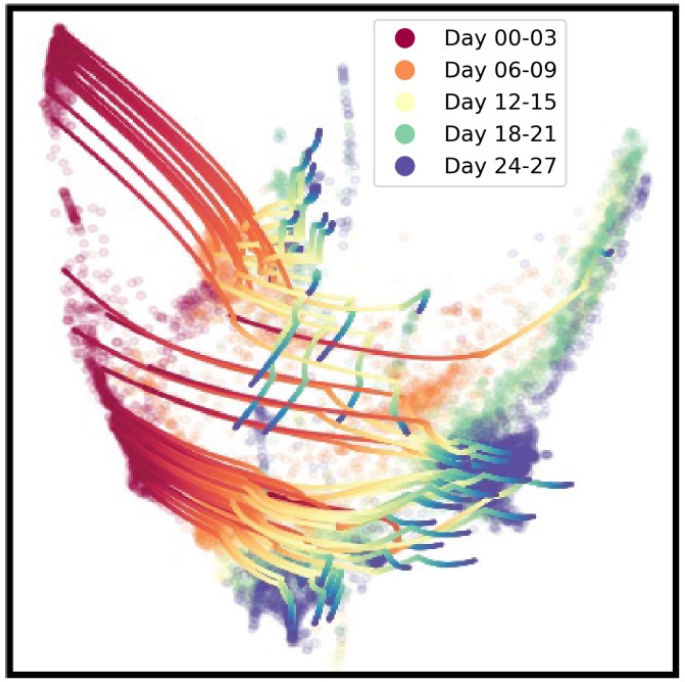
Toy Example


t_0 → $t_{1/2}$ → t_1
t_0
$t_{1/2}$
t_1


t_0
$t_{1/4}$
$t_{1/2}$
$t_{3/4}$
t_1
Base Model
Density Reg.
Velocity Reg.

Continuous Trajectories in Single Cell Data



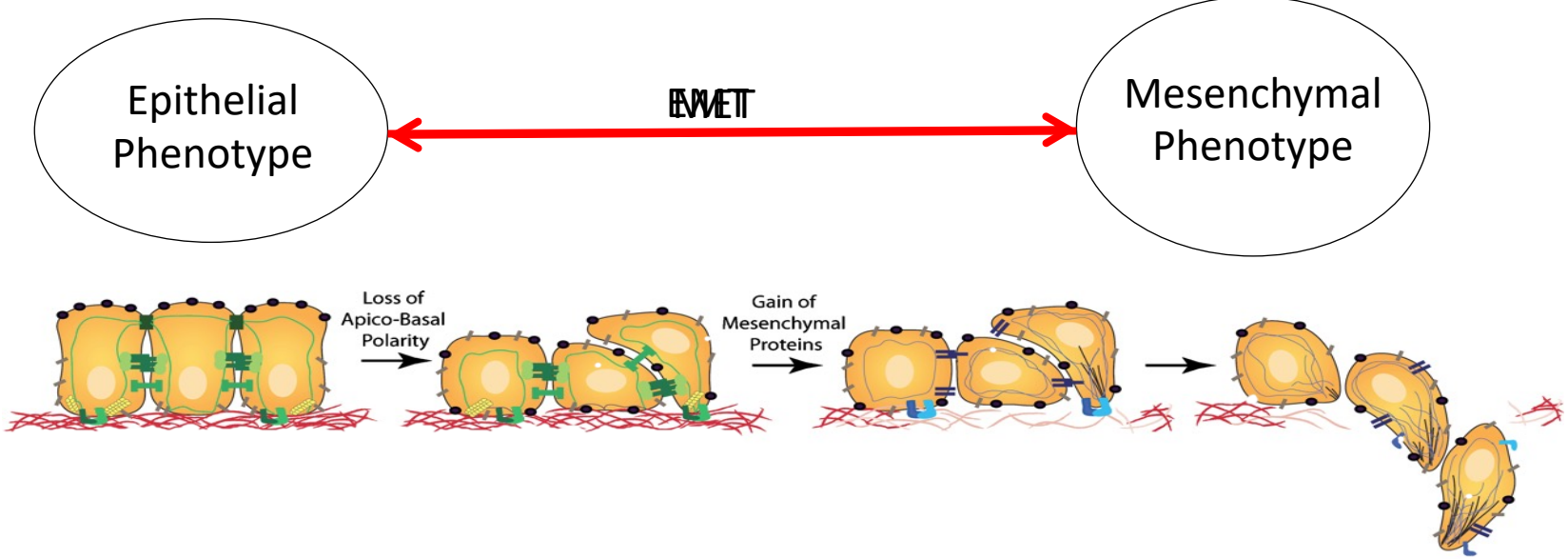


Single Cell Trajectories

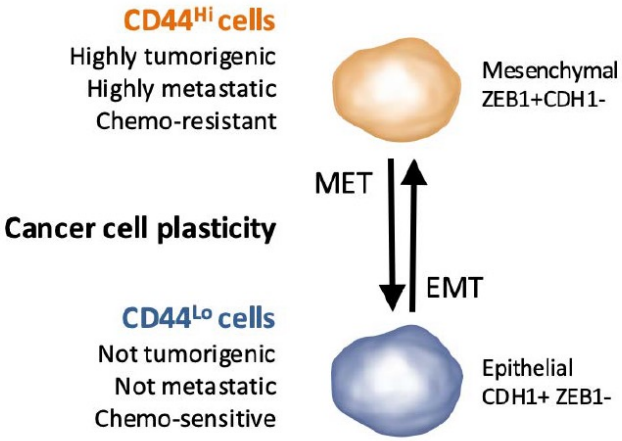
Results — Embryoid body dataset

	t=1	t=2	t=3	mean
Base	0.764	0.811	0.863	0.813
Base + D	0.759	**0.783**	0.811	**0.784**
Base + V	0.816	0.839	0.865	0.840
Base + D + V	0.930	0.806	**0.810**	0.848
Base + E	0.737	0.896	0.842	0.825
Base + G	**0.700**	0.913	0.829	0.814
OT	0.791	0.831	0.841	0.821
prev	1.715	1.400	0.814	1.309
next	1.400	0.814	1.694	1.302
rand	0.872	1.036	0.998	0.969

- Wasserstein distance between predicted and true distributions for different left out timepoints
- Different regularizations have different assumptions and tradeoffs



Beatriz Perez



EMT is relatively well-studied but MET which is equally important is Not studied well because of the need for 3D culture conditions

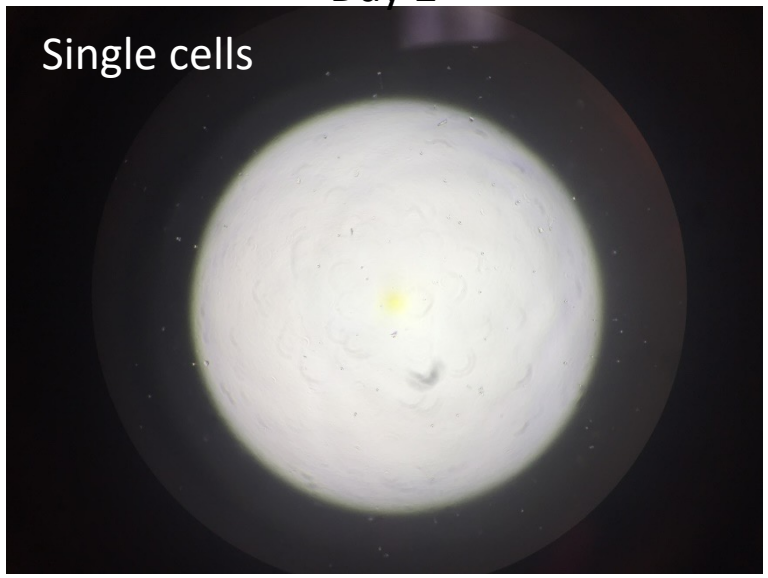
Here we are studying Triple Negative Breast Cancer, which has no targeted therapies.

Christine Chaffer

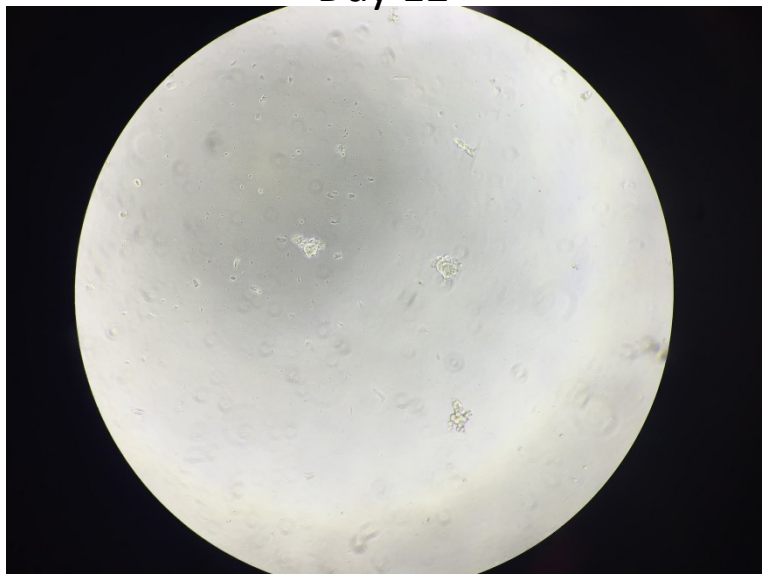
Mamospheres

乳腺球

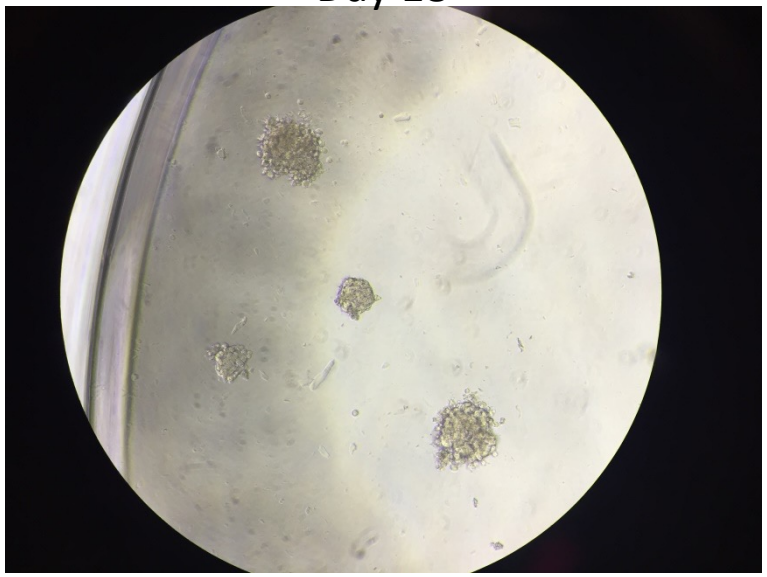
Day 2



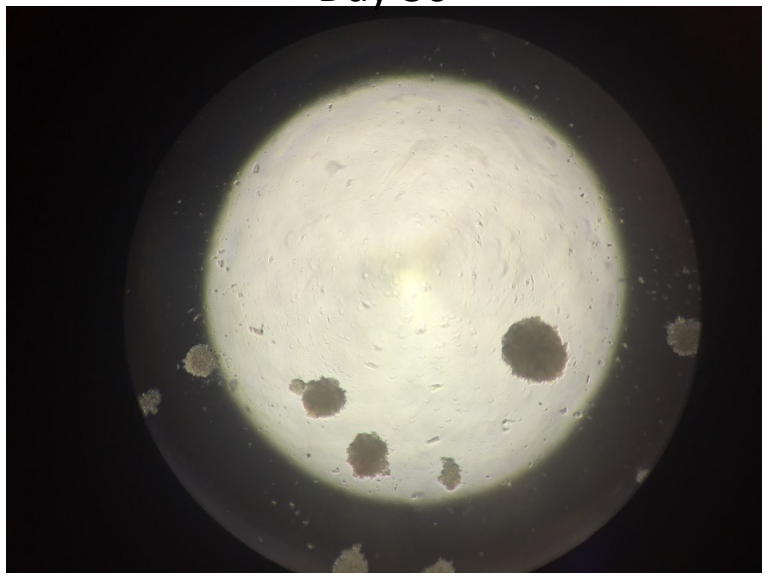
Day 12



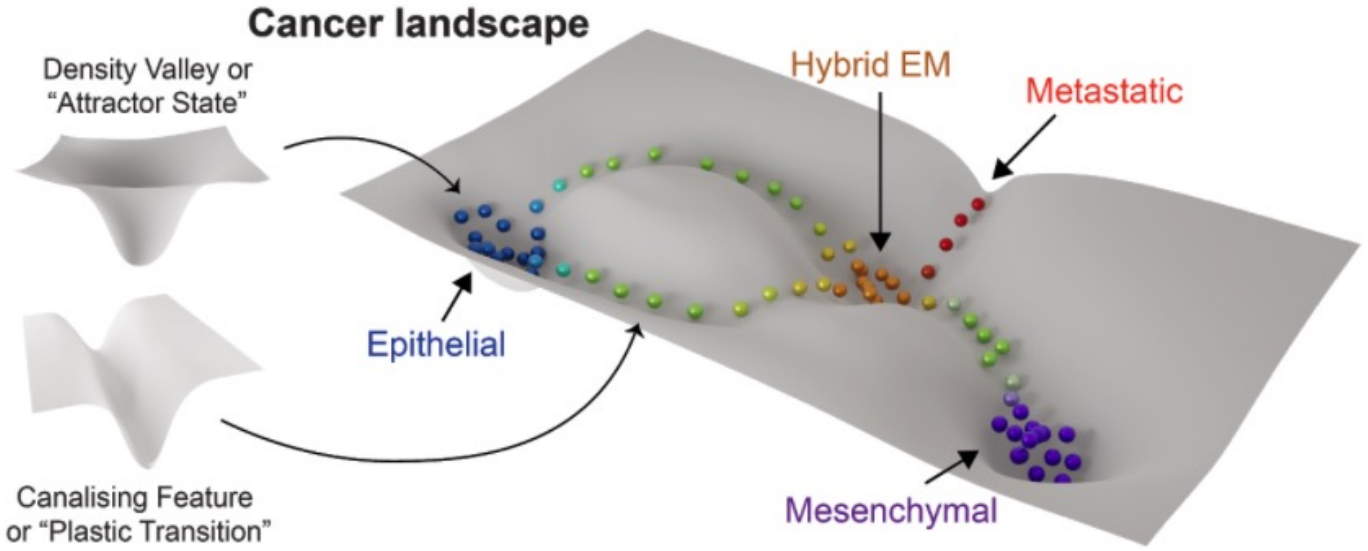
Day 18



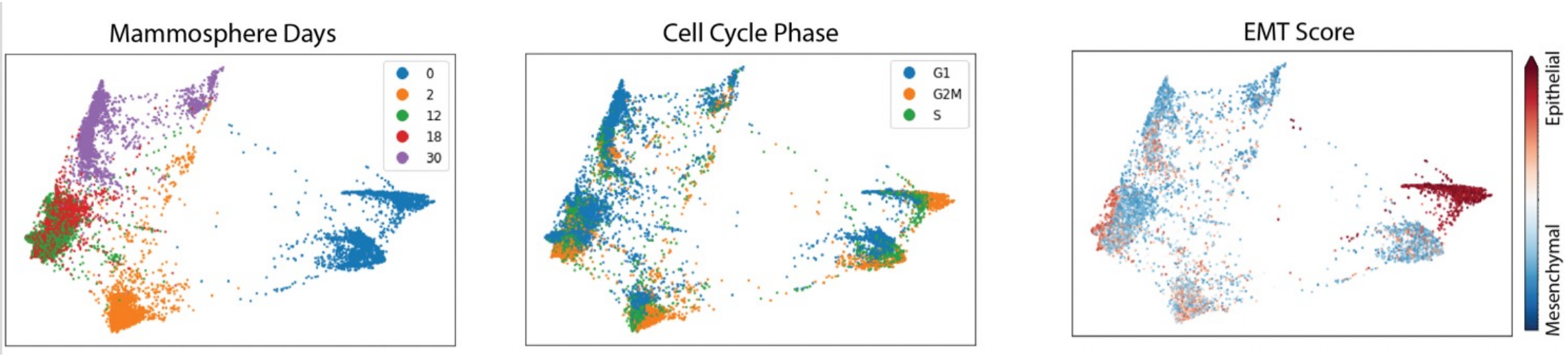
Day 30



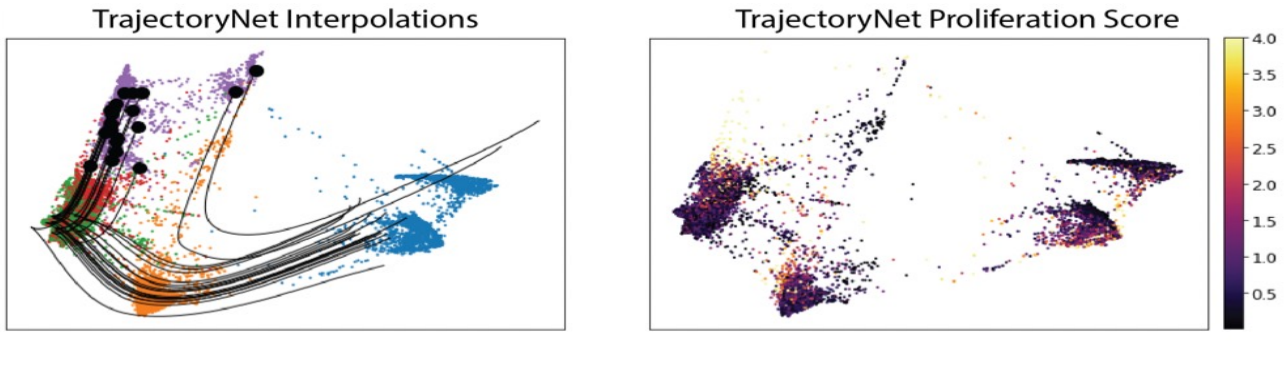
Trying to Understand the Landscape of the Cancer State Space

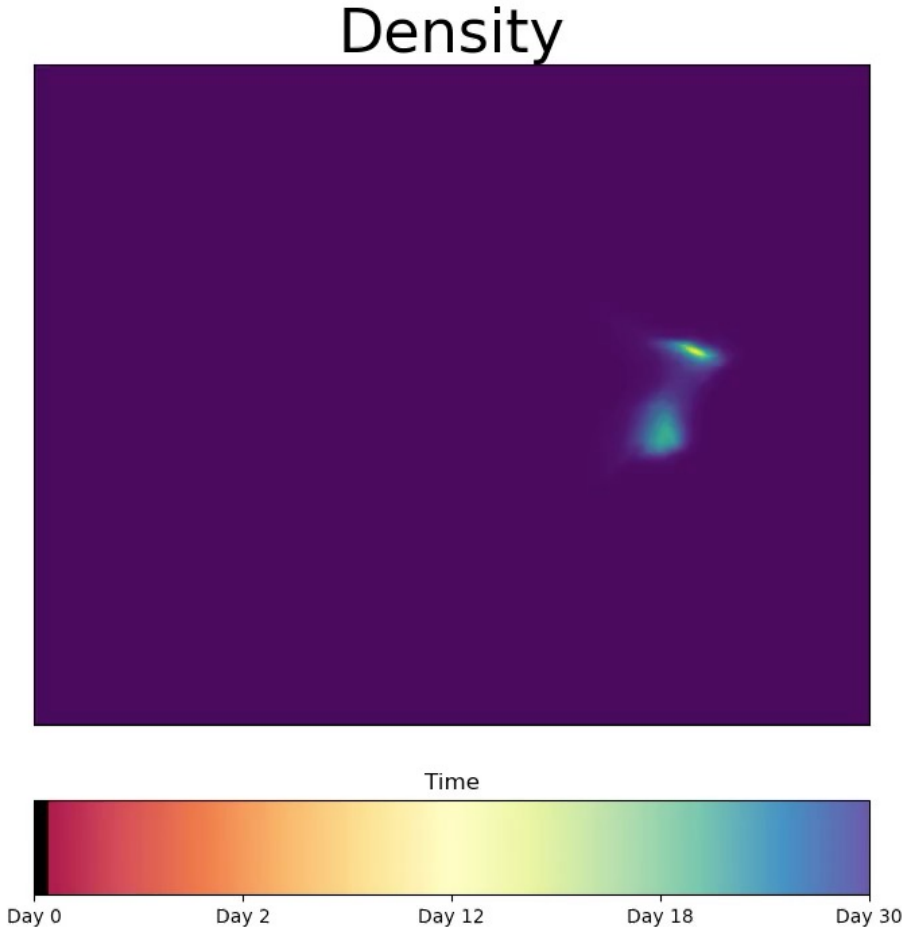


5-point timecourse of mamosphere culture

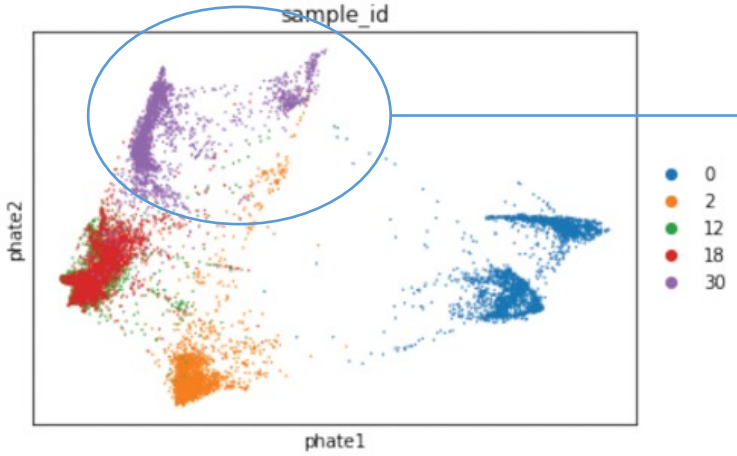


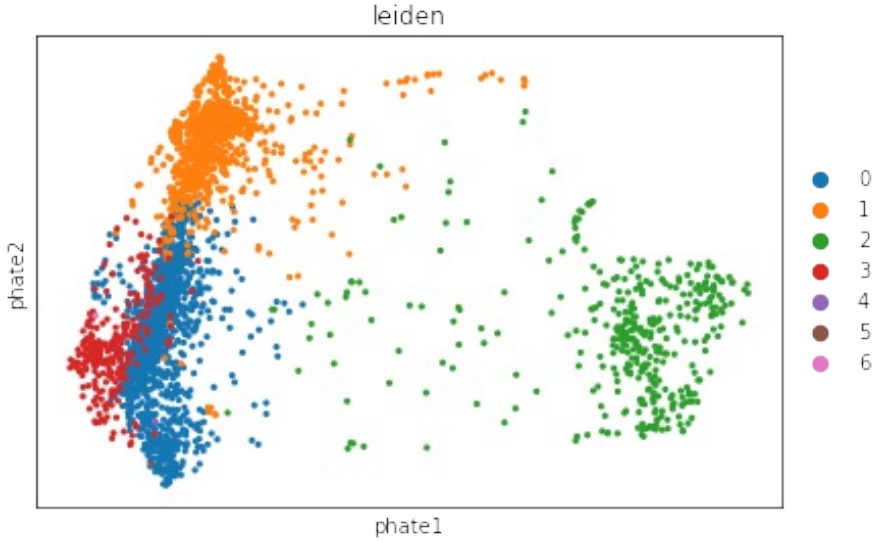
4-Timepoints of MET

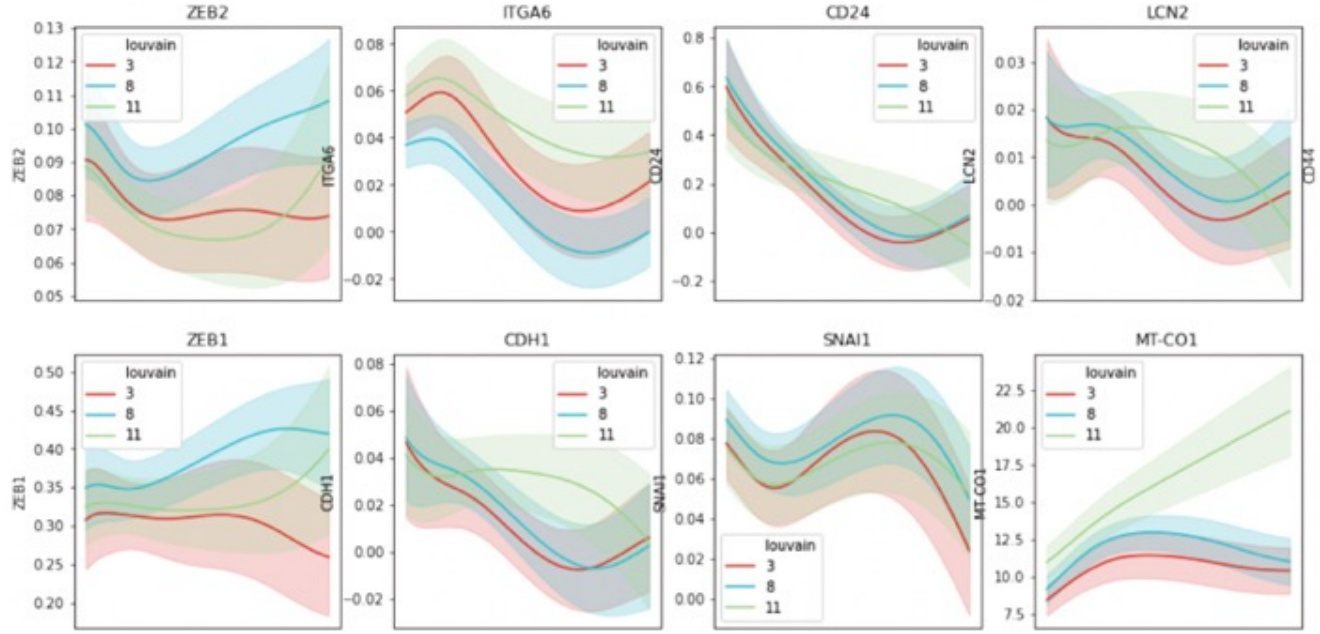


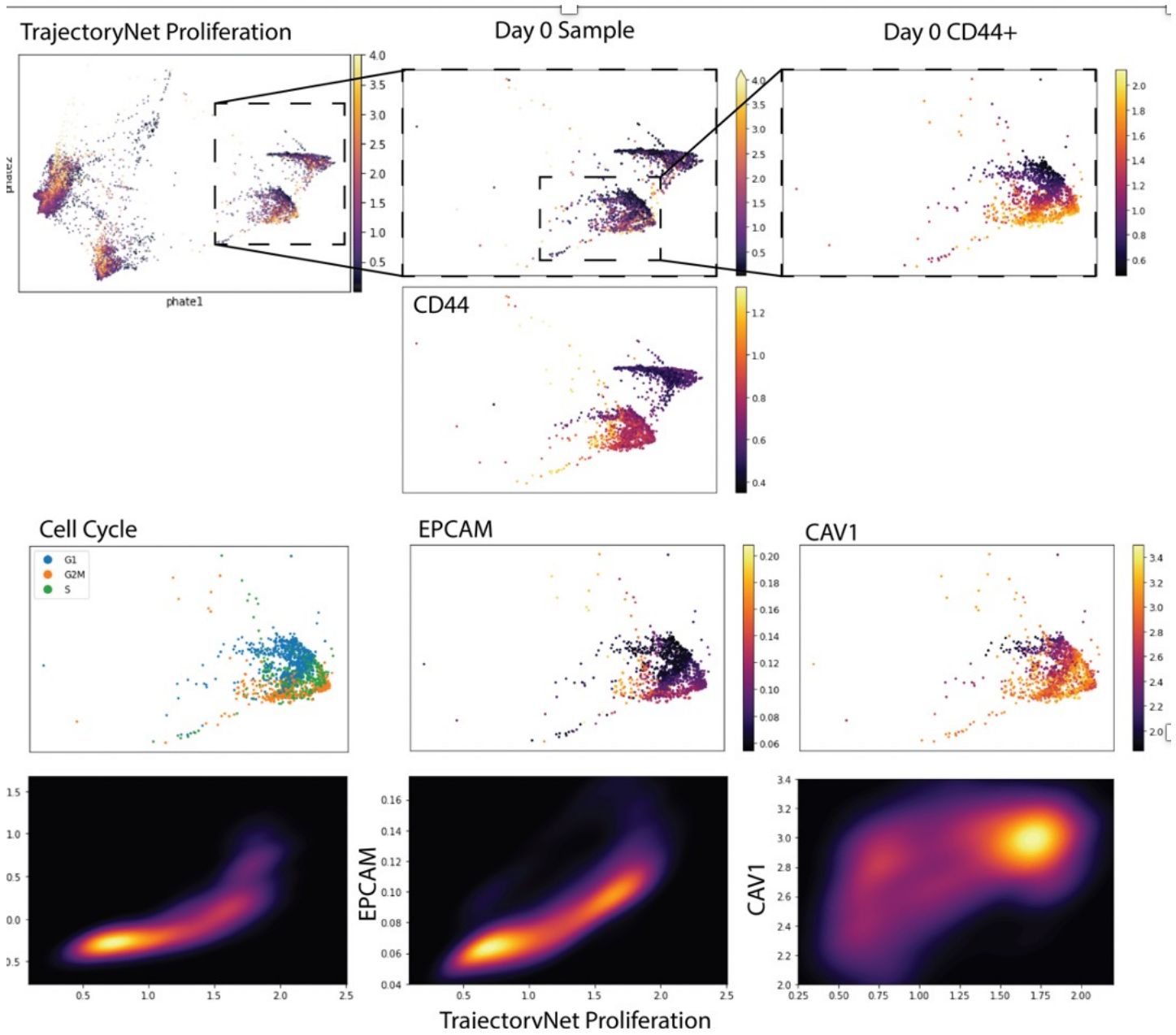


Tracing Cells Backwards



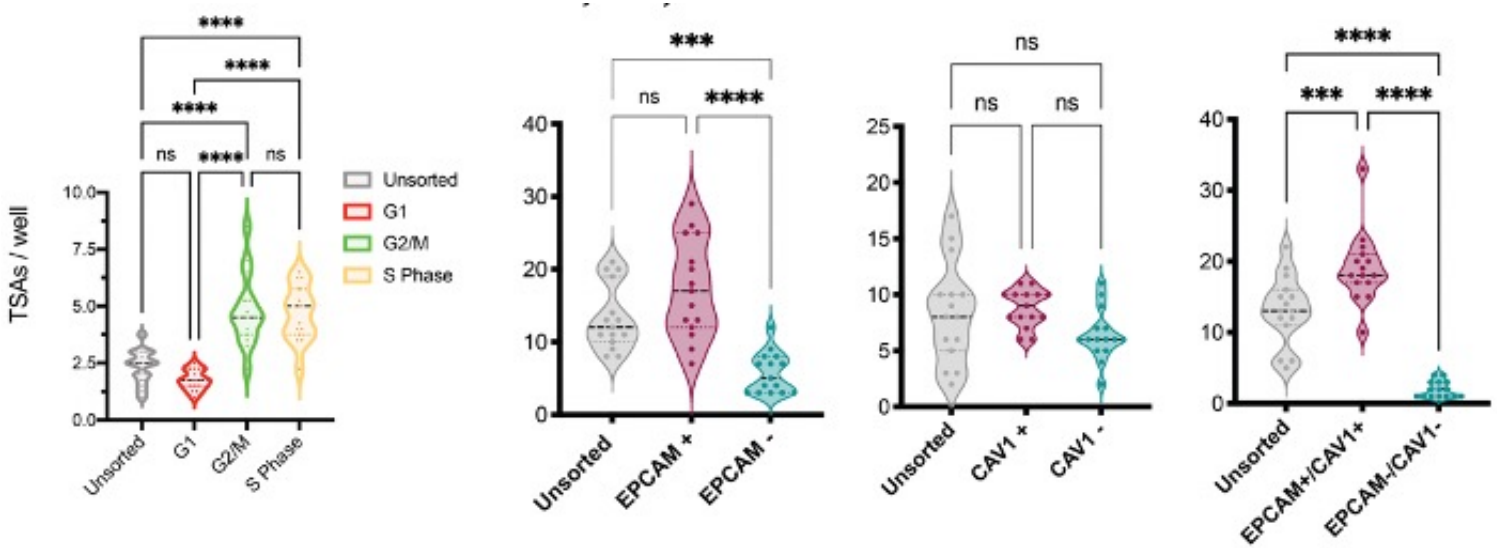




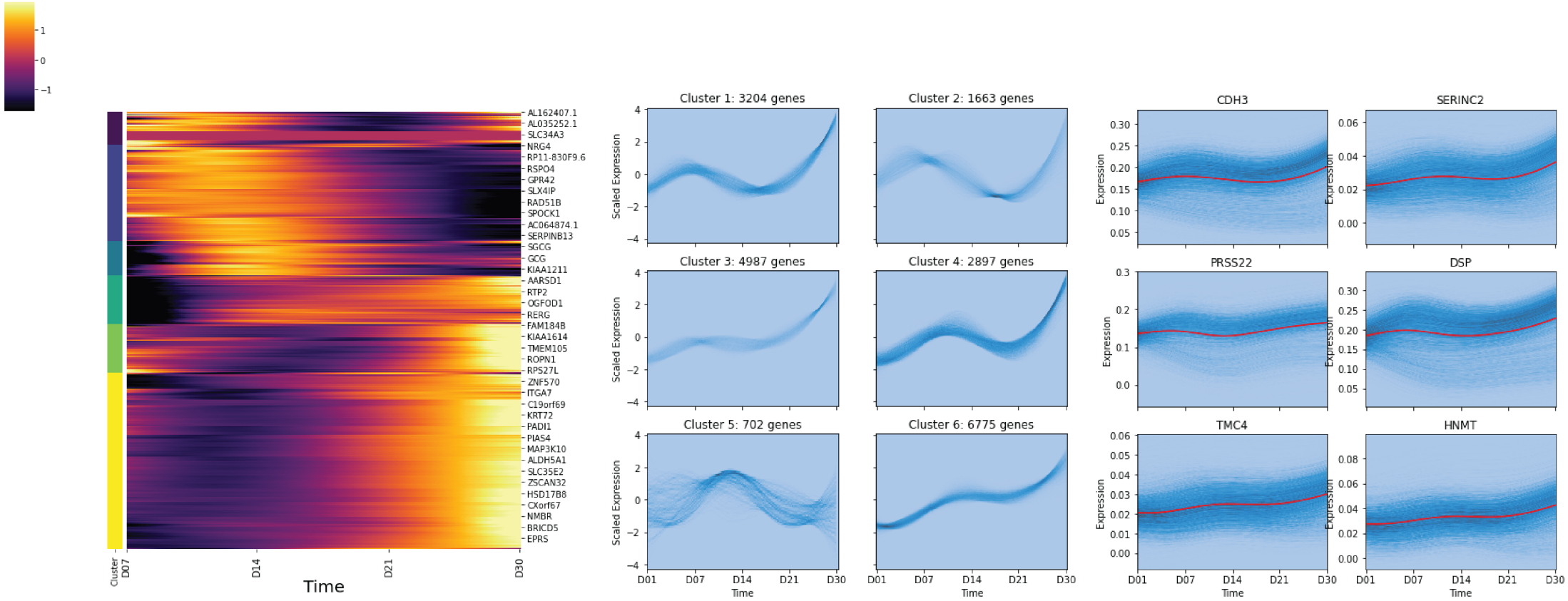


Cells of Origin

Validation


TSAs / well
****
****
****
ns
****
ns
10.0
7.5
5.0
2.5
0.0
Unsorted
G1
G2/M
S Phase
Unsorted
G1
G2/M
S Phase
***
ns
****
40
30
20
10
0
Unsorted
EPCAM +
EPCAM -
ns
ns
ns
25
20
15
10
5
0
Unsorted
CAV1 +
CAV1 -
****
***
****
40
30
20
10
0
Unsorted
EPCAM+/CAV1+
EPCAM-/CAV1-

Gene Regulatory Programs



Reading list

Chen et al. NeurIPS 2018 Neural Ordinary Differential Equations easy

Tong et al. ICML 2020 TrajectoryNET

Errico 1997 Adjoint Model

https://towardsdatascience.com/the-story-of-adjoint-sensitivity-method-from-meteorology-906ab2796c73