Improving Autoencoder Image Interpolation via Dynamic Optimal Transport

Xue Feng[1], Thomas Strohmer

[1]PhD, Department of Mathematics, UC Davis

INFORMS Optimization Society (IOS) conference
March 24th, 2024

Contents

PART 1: Introduction of Dynamic Optimal Transport

- Classical Optimal Transport and Displacement Interpolation
- Dynamic Optimal Transport

Classical Optimal Transport and Displacement Interpolation

Given ρ_0, ρ_T,

$$\min_{\pi \in \Pi(\rho_0, \rho_T)} \mathbb{K}(\pi) = \int_{X \times Y} c(x, y) \mathrm{d}\pi(x, y)$$

Kantorovich's Optimal Transport Problem

- $\rho_0(x) \geq 0$ and $\rho_T(y) \geq 0$ are two density functions which we assume to be nonnegative and have total mass one
- For 2D images $X = Y = [0, 1]^2$, which is the default space domain in this work

Classical Optimal Transport and Displacement interpolation

- Optimal transport map gives a natural way to interpolate between ρ_0 and ρ_T

Step 1: solve the optimal plan: $\pi^\dagger = \arg\min_{\pi \in \Pi(\rho_0, \rho_T)} \mathbb{K}(\pi)$
Step 2: push the source density to the target density: $\rho_t = \left((1-t)\mathrm{Id} + t\pi^\dagger\right)_\# \rho_0$
Displacement interpolation

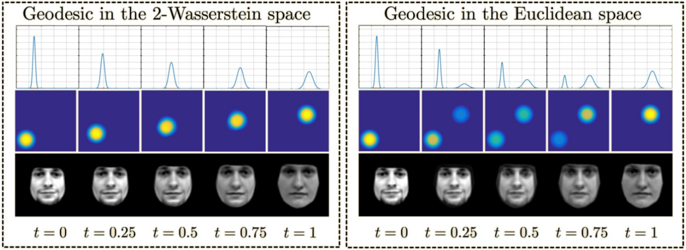


Figure : Interpolation in the optimal transport framework (left) and Euclidean space (right)

Dynamic Optimal Transport

• Dynamic optimal transport DIRECTLY finds the geodesic path $\rho(t,s)$ between $\rho_0(s)$ and $\rho_T(s)$ by minimizing the kinetic energy along the path:

Given ρ_0, ρ_T, solve ρ

$$\begin{aligned} \min_{\rho,\mathbf{v}} \quad & \frac{1}{2}\int_0^T \int_{[0,1]^2} \rho(t,s)|\mathbf{v}(t,s)|^2 dsdt, \\ \text{s.t.} \quad & \partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0, , \\ & \rho(0,\cdot) = \rho_0, \quad \rho(T,\cdot) = \rho_T, \end{aligned}$$

Dynamic Optimal Transport(Benamou2000)

• The square of the L^2-Wasserstein distance is equal to $2T$ times the infimum of dynamic optimal transport defined above

Dynamic Optimal Transport Solver

- Reformulate it to a convex problem:

$$\begin{aligned} \min_{\rho,\mathbf{m}} \quad & J(\rho,\mathbf{m}) = \frac{1}{2}\int_0^T \int_{[0,1]^2} \frac{|\mathbf{m}(t,s)|^2}{\rho(t,s)}\, dsdt, \\ \text{s.t.} \quad & \partial_t \rho + \nabla\cdot(\mathbf{m}) = 0, \\ & \rho(0,\cdot) = \rho_0, \quad \rho(T,\cdot) = \rho_T. \end{aligned}$$

- Discretize ρ on a centered grid, $\mathbf{m}$ on a staggered grid, then

$$\begin{aligned} \min_{\rho,\mathbf{m}} \quad & \sum_{t=0,2,..T-1} \mathbf{m}_t^T \operatorname{Diag}(\mathbf{w}_t)\mathbf{m}_t \\ \text{s.t.} \quad & b_t + \nabla\cdot(\mathbf{m}_t) = 0,\ t = 0,1,2,\dots T-1, \end{aligned}$$

where

$$\mathbf{w}^1_{t,i,j} = \frac{2}{\rho_{t,i,j} + \rho_{t,i+1,j}}, \quad \mathbf{w}^2_{t,i,j} = \frac{2}{\rho_{t,i,j} + \rho_{t,i,j+1}}, \quad b_t = \rho_{t+1} - \rho_t.$$

- Solver: Proximal Splitting Methods such as the Douglas–Rachford (DR) algorithm, the alternating direction method of multipliers (ADMM), and a primal-dual (PD) algorithm

Dynamic Optimal Transport Variants

$$
\begin{aligned}
\min_{\rho, \mathbf{m}} \quad & \frac{1}{2}\int_0^T \int_{[0,1]^2} \frac{|\mathbf{m}(t,s)|^2}{\rho(t,s)} dsdt, \\
\text{s.t.} \quad & \partial_t \rho + \nabla \cdot (\mathbf{m}) = 0, \\
& \mathbf{m}_C = 0, \\
& \rho(0,\cdot) = \rho_0, \quad \rho(T,\cdot) = \rho_T.
\end{aligned}
$$

when there are obstacles in the environment

$$
\begin{aligned}
\min_{\rho, \mathbf{m}} \quad & \frac{1}{2}\int_0^T \int_{[0,1]^2} \frac{|\mathbf{m}(t,s)|^2}{\rho(t,s)} + \tau \frac{|\mathfrak{s}(t,s)|^2}{\rho(t,s)} dsdt \\
\text{s.t.} \quad & \partial_t \rho + \nabla \cdot (\mathbf{m}) = \mathfrak{s}, \\
& \rho(0,\cdot) = \rho_0, \quad \rho(T,\cdot) = \rho_T
\end{aligned}
$$

unbalanced OT where $\int_X \rho_0(s)ds \neq \int_X \rho_T(s)ds$

PART 2: Innovating with Autoencoders: A New Take on Dynamic OT

- Our algorithm: Reformulating Dynamic OT with Autoencoder
  - key 1: eliminate the momentum m to make the minimization problem unconstrained
  - key 2: parameterize the path ρ using autoencoder

Step 1: eliminate m to make the problem unconstrained

- Fix ρ, we define the path energy over ρ as below:

$$
\begin{aligned}
J(\rho) = \min_{\mathbf{m}} \quad & \sum_{t=0,1,2,..T-1} \mathbf{m}_t^T \, \mathrm{Diag}(\mathbf{w}_t)\mathbf{m}_t \\
\text{s.t.} \quad & \nabla \cdot (\mathbf{m}_t) = b_t, \, t = 0, 1, 2, \dots T-1
\end{aligned}
$$

Note that **w** and b are both defined using ρ

- This is a quadratic problem with linear constraint, and its KKT condition is given by

$$
\begin{bmatrix} \mathrm{Diag}(\mathbf{w}_t) & \nabla \cdot^\top \\ \nabla \cdot & 0 \end{bmatrix} \begin{bmatrix} \mathbf{m}_t \\ \lambda_t \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b_t} \end{bmatrix}, t = 0, 1, 2, \dots, T-1, \tag{1}
$$

where λ_t is the Lagrange multiplier.

- After solving the KKT condition, we have

$$
J(\rho) = \sum_{t=0}^{T-1} b_t^T \left(\nabla \cdot \mathrm{Diag}(\mathbf{w_t})^{-1} \nabla \cdot^\top \right)^{-1} b_t,
$$

Path energy function

Derivative of the path energy function $J(\rho)$

- The first-order gradient of the path energy is given by

$$(\frac{\partial J}{\partial \rho_t})_{i,j} = -\frac{1}{4} \sum_{(k,l)\in\mathcal{O}_{i,j}} (y_{t,k,l} - y_{t,i,j})^2 + 2y_{t,i,j},$$

where $\mathcal{O}_{i,j}$ is the connected neighbor of (i,j), and

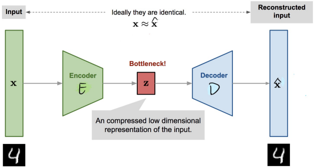
$$y_t = \left(\nabla \cdot \mathrm{Diag}(\mathbf{w_t})^{-1} \nabla \cdot^{\top}\right)^{-1} b_t$$

- The computational bottleneck is solving the sparse linear system, and we used numpy.spspsolve in python to solve it.

Step 2: parameterize the path variable ρ using autoencoder

$$J(\rho) = \sum_{t=0}^{T-1} b_t^T \left(\nabla \cdot \mathrm{Diag}(\mathbf{w_t})^{-1} \nabla^\top\right)^{-1} b_t,$$

Path energy function



Autoencoder

- Motivation: use generator to produce ρ to achieve smooth transitions along $\rho(t)$

- We adopted the decoder of the autoencoder, denoted as $\mathcal{D}$, as the generator:

$$\rho(t) = \mathcal{D}(tz_0 + (1-t)z_1), \quad 0 \le t \le 1,$$

where z_0, z_1 are the latent code of input ρ_0, ρ_1.

- Denote the autoencoder parameter as θ, then

$$\min_\rho J(\rho) \longrightarrow \min_\theta J(\rho_\theta)$$

optimization problem $\longrightarrow$ NN training

Our algorithm for image interpolation

- When we have two data in the training dataset,

Step 1: train an autoencoder whose loss function is

$$||\hat{\rho}_0 - \rho_0||^2 + ||\hat{\rho}_1 - \rho_1||^2 + \alpha J(\mathcal{D}(t\mathcal{E}(\rho_1) + (1-t)\mathcal{E}(\rho_2))$$

Step 2: generate the interpolation between ρ_0 and ρ_1 using $\mathcal{D}(t\mathcal{E}(\rho_1) + (1-t)\mathcal{E}(\rho_2)$

- When we have multiple data in the training dataset,

Step 1: train an autoencoder whose loss function is

$$\sum_i ||x_i - \hat{x}_i||^2 + \alpha \sum_{i,j} J(\mathcal{D}_{i \to j})$$

Step 2: generate the interpolation between ρ_i and ρ_j using $\mathcal{D}(t\mathcal{E}(\rho_i) + (1-t)\mathcal{E}(\rho_j)$

- Comparison with normalizing flow(NF):
  - Recall $\partial_t \rho = -\nabla \cdot (\rho(t)\mathbf{v}(t))$
  - normalizing flow use NN to generate $\mathbf{v}(t)$, and then push ρ_0 to generate $\rho(t)$ using integration; but the target density may not be matched.
  - we generate a path between ρ_0 and ρ_1 directly; $\mathbf{v}$ is baked into the loss function.

Experiment results

- (a) The result of our proposed method. (b) The result of the proximal splitting method

(a)
(b)

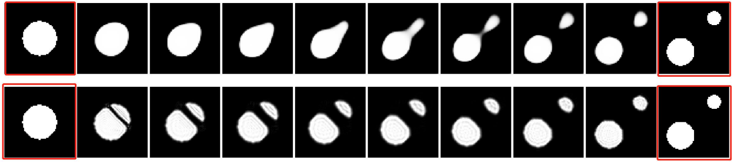


Figure: example (1)

Experiment results

- (a) The result of our proposed method. (b) The result of the proximal splitting method

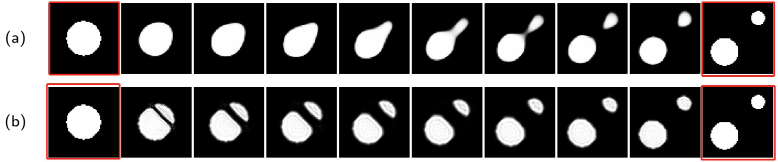


Figure: example (1)

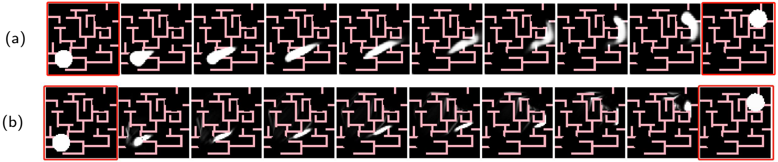
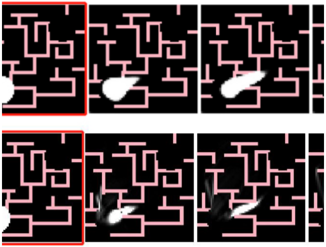


Figure: example when obstacles in the environment (marked pink) are present

Experiment results



(a)

(b)

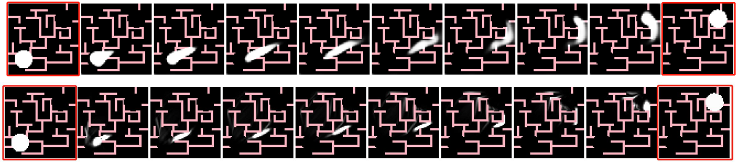


Figure: example when obstacles in the environment (marked pink) are present

Summary

• Our algorithm: Reformulating Dynamic OT with Autoencoder

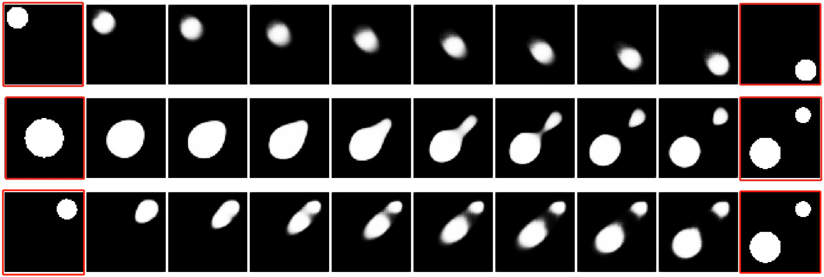


Figure: more examples of our algorithm.

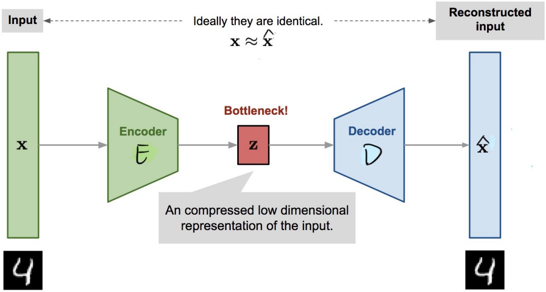
• Feature of our interpolation results:

- follow the least energy principle
- shows a smooth effect visually
- works from limited training data to large training data(will show in the following)

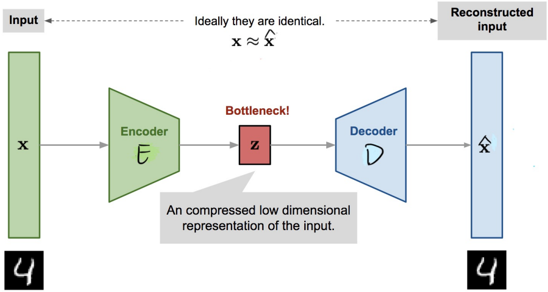
PART 3: Improving Autoencoder Image Interpolation

- A second view of our algorithm

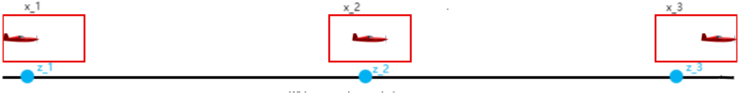
Autoencoder



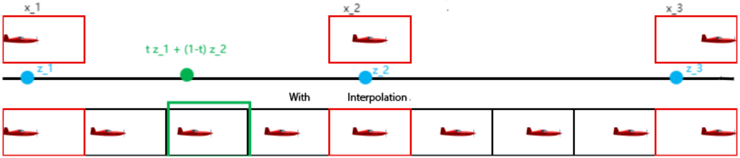
Autoencoder



Train data: $x_1, x_2, x_3, ...$
latent code: $z_1, z_2, z_3, ...$



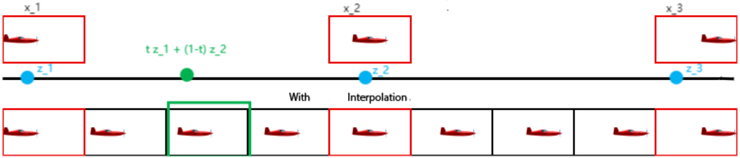
Autoencoder interpolation



Interpolation using a baseline autoencoder

- Step 1: train an autoencoder with MSE loss
- Step 2: decode $(tz_i + (1-t)z_j), 0 < t < 1$ to interpolate between image x_i and x_j

Autoencoder interpolation



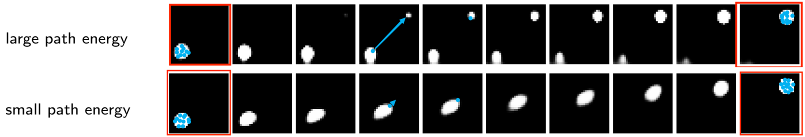
Interpolation using a baseline autoencoder

- Step 1: train an autoencoder with MSE loss
- Step 2: decode $(tz_i + (1-t)z_j), 0 < t < 1$ to interpolate between image x_i and x_j

However, poor interpolation when the training data is limited:

Our method

- Illustration:



- Our loss function penalizes bad interpolation

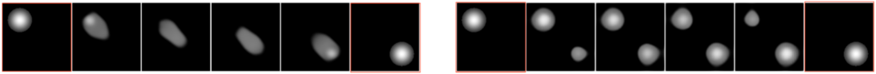
$$\sum_i ||x_i - \hat{x}_i||^2 + \alpha \sum_{i,j} J(\mathcal{D}_{i\to j})$$

where $\mathcal{D}_{i\to j}$ is the generated image path between image x_i and image x_j, α is a parameter to tune

PART 3.1: Comparison results when the training data contain only two images

Experiments

(a) Interpolation results on binary images. Left: Our method (SSIM score: 0.87). Right: Baseline autoencoder (SSIM score: 0.91).



(b) Interpolation results on gray-scale images. Left: Our method (SSIM score: 0.89). Right: Baseline autoencoder (SSIM score: 0.93)

(c) Interpolation results on RGB images. Left: Our method (SSIM score: 0.91). Right: Baseline autoencoder (SSIM score: 0.88).

Figure □ Comparison of our proposed method and the baseline autoencoder method across different image types.

- while baseline autoencoder captures mainly local changes, our result is more realistic and approximating the geodesic path

PART 1.2: when training dataset set is large

MNIST dataset

Figure: The interpolation results on MNIST dataset using our proposed method.
The training dataset is the whole MNIST dataset

- We randomly choose some (i, j) pairs to reduce computation cost in each training epoch: $\sum_i ||x_i - \hat{x}_i||^2 + \alpha \sum_{(i,j)\in\mathcal{S}} J(\mathcal{D}_{i\to j})$

Comparison with Existing Autoencoder Methods

(a) Ours (SSIM score mean: 0.96, std: 0.01).

(b) ACAI (SSIM score mean: 0.87, std: 0.10). Note that the transition is abrupt in the middle.

(c) VAE (SSIM score mean: 0.9, std 0.10).

(d) Baseline Autoencoder(SSIM score mean: 0.9, std: 0.11). Note that the interpolation results are blurry.

Figure □ Comparison of interpolation results on the MNIST dataset using four different autoencoder methods: Ours, ACAI, VAE, and a baseline autoencoder.

- we achieve one of the best visual interpolation results
- other methods do not work well when training data are limited.

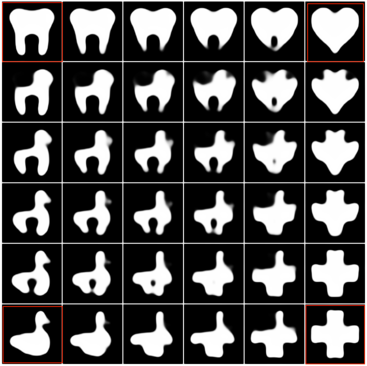
PART 3.3: Exploration of the output space of our Trained Autoencoder

Example

Our method:

- Step 1: train an autoencoder with loss function $\sum_i ||x_i - \hat{x}_i||^2 + \alpha \sum_{i,j} J(\mathcal{D}_{i\to j})$
- Step 2: generate the barycenter by decoding a convex combination of their corresponding latent codes

$$\mathcal{D}\left(c_1 z_1 + c_2 z_2 + c_3 z_3 + c_4 z_4\right), \quad \sum_i c_i = 1.$$



there are only four images (at the corner) in the training dataset.

- The output space is a smooth manifold even with limited training data; $\mathcal{W}(\mathcal{D}(\sum_i \hat{c}_i z_i), \mathcal{D}(\sum_i \tilde{c}_i z_i))$ is small when $\hat{\mathrm{c}}, \tilde{\mathrm{c}} < \epsilon$
- ongoing work: application on signal recovery

Summary

What we have done

- We reformulate the dynamic optimal tranport problem using autoencoder
- We evaluate our approach in a variety of scenarios, from limited training data to large training data. Our method produces robust and smooth interpolation results in all cases.

Ongoing Work

- Explore the output space of our trained autocoder beyond interpolation

Thanks!

For any further questions, feel free to contact: xffeng@ucdavis.edu