Project 10: Learning an Optimal Transport Brenier Map

Félix Fourreau Mathis Wauquiez
École des Ponts ParisTech (ENPC)
ENS Paris-Saclay
{felix.fourreau, mathis.wauquiez}@eleves.enpc.fr

March 21, 2025

1 Introduction and Methods

Optimal Transport (OT) is a powerful framework for comparing probability distributions. In its simplest and original form, introduced by Monge, the OT problem is written as:

$$\inf_{T:\, T_{\#}p_X = p_Y} \mathbb{E}_{x\sim p_X}\left[c\big(x, T(x)\big)\right],$$

where $T_{\#}p_X = p_Y$ denotes that the mapping T pushes forward the source measure p_X onto the target measure p_Y and c is a cost function (typically, the squared Euclidean distance). When $c(x, y) = \|x - y\|^2$, Brenier's Theorem guarantees that, under mild conditions, the unique optimal transport map is given by the gradient of a convex function.

Brenier's Theorem

Theorem. Let p_X and p_Y be two probability measures on $\mathbb{R}^n$ with finite second moments, and assume p_X is absolutely continuous with respect to the Lebesgue measure. Then, there exists a unique convex function $\varphi : \mathbb{R}^n \to \mathbb{R}$ (up to an additive constant) such that the optimal transport map T between p_X and p_Y is given by:

$$T(x) = \nabla\varphi(x) \quad \text{for } p_X\text{-almost every } x.$$

This result motivates learning the gradient directly from data even when an explicit convex potential is difficult to construct.

Motivation and Prior Approaches

Traditional methods such as Input Convex Neural Networks (ICNN) and Input Convex Gradient Networks (ICGN) enforce convexity through architectural constraints or by learning second-order information. However, these methods can be computationally demanding and complex. The contribution of the MGN (Monotone Gradient Networks) paper is to simplify the process by designing architectures that *directly* learn a monotone gradient mapping while ensuring the necessary convexity properties.

2 Constraining Neural Networks to be Gradients of Convex Functions

A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if its Hessian $H_f(x)$ is positive semidefinite (PSD) for all x. If $g(x) = \nabla f(x)$, then the Jacobian $J_g(x)$ must also be PSD. We now describe the two architectures introduced in the paper, that guarantee this property.

2.1 Cascaded Monotone Gradient Network (C-MGN)

The C-MGN is a multi-layer architecture with shared weight matrices. For an input $x \in \mathbb{R}^d$, the network is defined as:

$$\begin{aligned} z_0 &= Wx + b_0, \\ z_l &= Wx + \sigma_l(z_{l-1}) + b_l, \quad l = 1, \ldots, L-1, \\ \text{C-MGN}(x) &= W^T \sigma_L(z_{L-1}) + V^T V x + b_L. \end{aligned}$$

Here, $\sigma_l(\cdot)$ are element-wise activation functions (e.g., ReLU, where $\sigma'(x) \geq 0$ for all x), ensuring that the Jacobians $J_{\sigma_l}(z_{l-1})$ are non-negative. The network's overall Jacobian is:

$$\mathbf{J}_{\text{C-MGN}}(x) = W^\top \left(\sum_{\ell=1}^{L} \prod_{i=\ell}^{L} J_{\sigma_i}(z_{i-1}) \right) W + V^\top V.$$

Since a sum and product of PSD matrices remain PSD, the Jacobian is PSD and thus the network represents the gradient of a convex function.

2.2 Modular Monotone Gradient Network (M-MGN)

The M-MGN uses K parallel layers:

$$\begin{aligned} z_k &= W_k x + b_k, \quad k = 1, \ldots, K, \\ \text{M-MGN}(x) &= a + V^T V x + \sum_{k=1}^{K} s_k(z_k)\, W_k\, \sigma_k(z_k). \end{aligned}$$

This architecture can be compactly rewritten as:

$$\text{M-MGN}(x) = a + V^T V x + W^T \big(\sigma(Z) \cdot s(Z)\big),$$

where $W = (W_1, \ldots, W_K)^T$, and $s(Z)$ and $\sigma(Z)$ are applied element-wise. For more details, please read the proof of the theorem 1 in the annex. As before, if the activation functions σ_k (and scaling functions s_k) are chosen to be non-decreasing, the network's Jacobian is PSD, ensuring it corresponds to the gradient of a convex function.

3 Loss Functions

We experimented with two losses:

3.1 Kullback–Leibler Divergence:

One possible loss is the Kullback-Leibler Divergence:

$$D_{KL}(p \parallel q) = \int p(x) \log \frac{p(x)}{q(x)}\, dx.$$

This loss function has a closed form formula in the case where both p and q are multivariate normal distributions:

$$D_{\mathrm{KL}}(\mathcal{N}(\mu_0, \Sigma_0) \parallel \mathcal{N}(\mu_1, \Sigma_1)) = \frac{1}{2}\left[\mathrm{tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1}(\mu_1 - \mu_0) - d + \log \frac{\det \Sigma_1}{\det \Sigma_0}\right]$$

We hypothesize that the authors might have used this loss to train models. One can then compute the mean and covariance of the push forward distribution $T_{\#}\mu$, and use this loss function to align it with the target. The issue here, which is not mentioned in the paper, is that the push forward distribution $T_{\#}\mu$ is probably not multivariate normal. Therefore, this loss function will only control the mean and covariance of our distribution, but lacks more precise information about the divergence between the push forward and target distributions.

3.2 Dual Wasserstein Distance:

$$W(\mu, \nu) = \sup_{f \in \mathrm{Lip}_1} \left\{ \mathbb{E}_{x \sim \mu}[f(x)] - \mathbb{E}_{y \sim \nu}[f(y)] \right\}.$$

Inspired by the rise of Wassertein Generative Networks, we choose to implement the adversarial Wassertein distance as a metric to quantify the quality of the push-forward distribution.
This adversarial loss trains a critic to differentiate between the mapped samples and the target distribution. We choose the critic to be a 3 layers MLP with hidden dimension of 64 and a leakyReLU activation, trained using gradient penalty and an Adam optimizer.

4 Experiments on Low-Dimensional Examples

We validate the models proposed by the authors on three tasks.

4.1 Gaussian Mapping

Example We construct a source distribution from a Gaussian distribution with a known mean μ_1 and covariance Σ_1, and a target Gaussian distribution with mean μ_2 and covariance Σ_2. This example is supposed to replicate the section 5.2 of the ICNN paper, where the authors solves the same problem by "minimizing the KL-divergence with the standard normal prior distribution (their target distribution) with the standard change-of-variable formula for probability densities, as

discussed in [**?**]". From their definition, it is not exactly clear how they optimize the NLL, as the change-of-variable formula they quote needs an explicit inverse mapping:

$$p_Y(f(x)) = p_X(x) \left| \frac{\partial f(x)}{\partial x} \right|^{-1} \iff p_Y(y) = p_X(f^{-1}(y)) \left| \frac{\partial f^{-1}(y)}{\partial y} \right|$$

We are not sure that such a mapping exists, as we have no guarantee that the networks we train are **strictly** monotonic. Furthermore, they validate their approach against several methods by comparing the log-likelihood of the transported samples. This approach is questionable, as this quantity does not really reflect the divergence between the target distribution and the transported distribution, but rather how all points are realistic. Indeed, a model could learn to transport all points to the mean of the Gaussian target distribution, which would not be solving the Optimal Transport problem but would get the lowest NLL.

Results The results, in the appendix, 1 show that, as expected, the negative log-likelihood metric is too coarse, only aligning the covariance and mean of the distributions. On the other hand, the adversarial loss shows promising results, with a push forward distribution matching closely the target distribution. It is to be noted that, since the functions we use are monotonic gradients and thanks to Brenier's Theorem, we do not need to incorporate the transport cost $c(x, T(x))$ into our loss function. In this experiment and all the following experiments, our model is trained using an Adam optimizer, with a learning rate of 10^{-4}.

4.2 Gaussian Mixture Model (GMM) Mapping

Example: We construct a source distribution from a GMM with two Gaussian components and a target GMM with three components, each having distinct means and covariances. For instance, if the source components are centered at $(-2, 0)$ and $(2, 0)$ while the target components are centered at $(-3, 1)$, $(0, -1)$, and $(3, 1)$, our network learns to push forward samples from p_X to p_Y. This extends the work done by the authors, which could only transfer a source distribution toward a multivariate normal distribution.

Results As before, only the results obtained with the Wasserstein distance are satisfying, although the covariance and mean are perfectly aligned when using the Kullback Leibler divergence. Moreover, it seems that the CMGN model outperforms MMGN, with a better distribution match. This trend was confirmed experimentally on other Gaussian mixture models. 3

4.3 Pixel Distribution Mapping

Example: Our third experiment involves transforming the pixel distribution of a daytime image to that of a sunset image (with warmer, orange hues). In this experiment, it is essential that the transport we define is near-optimal, so as to preserve the structure of the source image.

Results The results we obtained are really satisfying: by integrating the adversarial Wasserstein distance into our training process, we are able to map, almost perfectly in the case of CMGN, one pixel distribution to another. Being able to solve optimal transport problems for arbitrary source and target distributions is usually a complicated problem, and here we are able to solve it with much effort, since the models we use are small and can be trained in approximately 10 seconds on a Gtx 1050 Ti GPU. On this problem, we also see MMGN perform worse than CMGN. 5

5 Learning a Generative Network on MNIST

We further evaluated the approach on the MNIST dataset. Starting from a Gaussian latent distribution, the network is trained to generate digits that match the empirical distribution of MNIST. This experiment demonstrates the scalability of the method from low-dimensional examples to large-scale image data.

Number Generation

The generative network is initialized with a Gaussian prior $\mathcal{N}(0, I)$ and optimized so that the transformed samples reproduce the distribution of handwritten digits, using the adversarial Wasserstein distance. Qualitative inspection shows that the CMGN network produces good samples, compared to the number of parameters it has. We still see a lot of 'salt and pepper noise', and sometimes one sample can be in between multiple classes. MMGN, however, performs a lot worse. This is expected, as its architecture, as proven (1), can be broken down to a perceptron with only one hidden layer, and a weight between the hidden layer and the output that is the transpose of the weights between the input and the hidden layer. Such a simple model is not fit for approximating complex distributions. Overall, both models perform poorly when compared to the ICNN, which consistently generates sharper, more class-consistent, and realistic samples thanks to its deeper architecture.

The results shown in the appendix 7

6 Conclusion

In this project, we implemented the two neural network architectures—C-MGN and M-MGN—that directly learn a monotone gradient mapping corresponding to the Brenier map in Optimal Transport. By ensuring that the network Jacobian remains PSD (through non-decreasing activation functions), the approach guarantees that the learned mapping is the gradient of a convex function. Experimental results on low dimensionnal exemple both Gaussian mixtures and real image data validate the method's effectiveness. However learning a generative network for a large-scale image dataset like MNIST show the limitation of the small architecture. Future work may focus on building more complex model that ensure learning a monotone gradient.

A M-MGN output reformulation proof

Theorem 1. *The function M-MGN(x) defined by*

$$\textit{M-MGN}(x) = a + \sum_{k=1}^{K} s_k(z_k) W_k^\top \sigma_k(z_k),$$

where $z_k = W_k x + b_k$, with $a \in \mathbb{R}^{d_{out}}$, $W_k \in \mathbb{R}^{d \times d_{in}}$, $b_k \in \mathbb{R}^d$, $W_k^\top \in \mathbb{R}^{d_{out} \times d}$, $s_k : \mathbb{R}^d \to \mathbb{R}$, and $\sigma_k : \mathbb{R}^d \to \mathbb{R}^d$, can be rewritten as:

$$\textit{M-MGN}(x) = a + W^\top \sigma(Wx + b),$$

for some matrices $W \in \mathbb{R}^{D \times d_{in}}$, $b \in \mathbb{R}^D$, and function $\sigma : \mathbb{R}^D \to \mathbb{R}^D$, where $D = K \cdot d$.

Note: The theorem addresses a simplified version of the M-MGN function compared to the original paper, which includes an additional linear term $V^T V x$:

$$\text{M-MGN}_{\text{original}}(x) = a + V^T V x + \sum_{k=1}^{K} s_k(z_k) W_k^\top \sigma_k(z_k)$$

This simplification does not limit generality, as the linear term can be incorporated into our framework by:

1. Augmenting the stacked weight matrix: $W_{\text{aug}} = \begin{bmatrix} W \\ V \end{bmatrix} \in \mathbb{R}^{(D+d_{\text{in}}) \times d_{\text{in}}}$
2. Augmenting the bias vector: $b_{\text{aug}} = \begin{bmatrix} b \\ 0 \end{bmatrix} \in \mathbb{R}^{D+d_{\text{in}}}$
3. Defining an augmented activation function $\sigma_{\text{aug}} : \mathbb{R}^{D+d_{\text{in}}} \to \mathbb{R}^{D+d_{\text{in}}}$ as:

   $$\sigma_{\text{aug}}(z) = \begin{bmatrix} \sigma(z_{1:D}) \\ z_{D+1:D+d_{\text{in}}} \end{bmatrix}$$

   where the last d_{in} components act as the identity function
4. Defining the augmented output projection: $W_{\text{aug}}^T = \begin{bmatrix} W^T & V^T \end{bmatrix} \in \mathbb{R}^{d_{\text{out}} \times (D+d_{\text{in}})}$

With these definitions, the original M-MGN function can still be expressed in our unified form as:

$$\text{M-MGN}_{\text{original}}(x) = a + W_{\text{aug}}^T \sigma_{\text{aug}}(W_{\text{aug}} x + b_{\text{aug}})$$

Proof: We will construct the appropriate matrices and function σ to demonstrate this equivalence.

Step 1: Constructing the parameter matrices

Let us define the block matrices by stacking the individual components:

$$W_1 = \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_K \end{bmatrix} \in \mathbb{R}^{D \times d_{\text{in}}}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_K \end{bmatrix} \in \mathbb{R}^D$$

With these definitions, we can express the concatenated pre-activation values as:

$$Z = W_1 x + b = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_K \end{bmatrix} \in \mathbb{R}^D$$

Step 2: Defining the activation function

We now define a single activation function $\sigma : \mathbb{R}^D \to \mathbb{R}^D$ that combines the scalar modulation and vector activation from the original formulation:

$$\sigma(Z) = \begin{bmatrix} s_1(z_1)\sigma_1(z_1) \\ s_2(z_2)\sigma_2(z_2) \\ \vdots \\ s_K(z_K)\sigma_K(z_K) \end{bmatrix}$$

Step 3: Constructing the output projection

The original summation can be rewritten using a block matrix multiplication:

$$\sum_{k=1}^{K} s_k(z_k) W_k^\top \sigma_k(z_k) = \begin{bmatrix} W_1^\top & W_2^\top & \cdots & W_K^\top \end{bmatrix} \sigma(Z)$$

Let us define $W_2^\top = \begin{bmatrix} W_1^\top & W_2^\top & \cdots & W_K^\top \end{bmatrix} \in \mathbb{R}^{d_{\text{out}} \times D}$. Then:

$$\text{M-MGN}(x) = a + W_2^\top \sigma(Z) = a + W_2^\top \sigma(W_1 x + b)$$

Step 4: Final representation

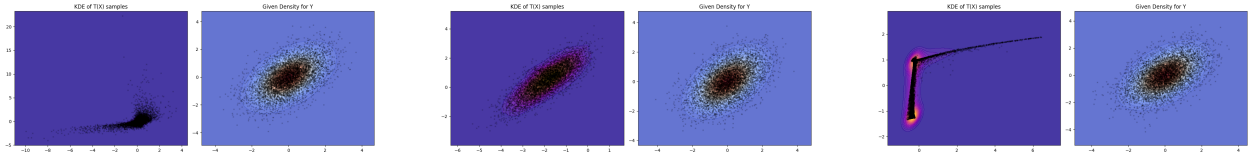
For the sake of notational consistency with the theorem statement, we can rename W_1 to W and note that $W_2^\top = W^\top$ (with appropriate dimensions), giving us:

$$\text{M-MGN}(x) = a + W^\top \sigma(Wx + b)$$

This matches the desired form stated in the theorem, where $W \in \mathbb{R}^{D \times d_{\text{in}}}$, $b \in \mathbb{R}^D$, and $\sigma : \mathbb{R}^D \to \mathbb{R}^D$.

B Results for Gaussian Mapping

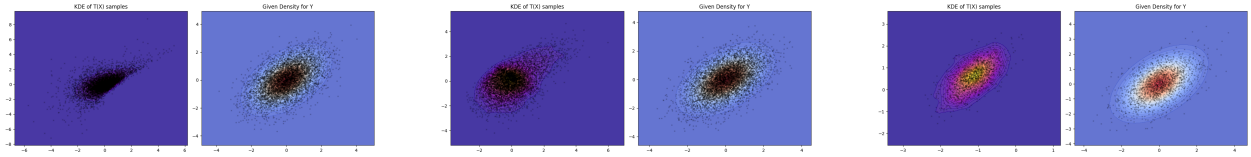
Results using the Kullback-Leibler divergence



(a) MMGN (b) CMGN (c) ICNN

Figure 1: Results using the Kullback-Leibler divergence for Gaussian Mapping. We note that, even though the mean and covariance are reproduced, the target distribution is not well modeled.

Results using the adversarial Wasserstein distance

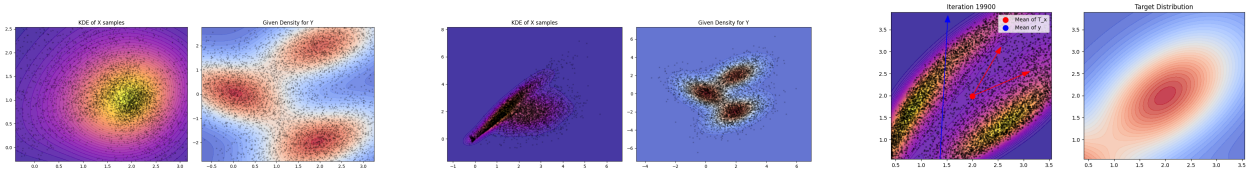


(a) MMGN (b) CMGN (c) ICNN

Figure 2: Results using the adversarial Wasserstein distance for Gaussian Mapping

C Results for GMM Mapping

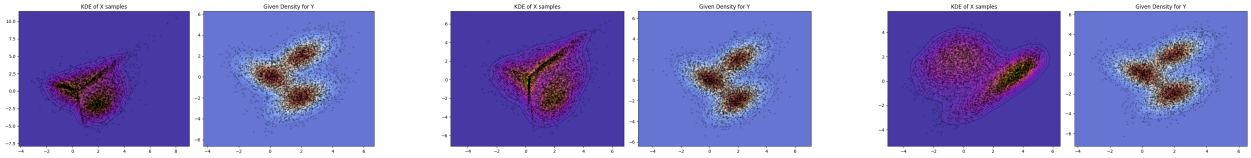
Results using the Kullback-Leibler divergence



(a) MMGN (b) CMGN (c) ICNN

Figure 3: Results using the Kullback-Leibler divergence for GMM mapping

Results using the adversarial Wasserstein distance

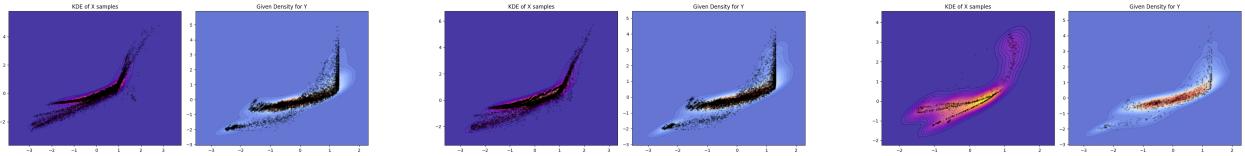


(a) MMGN (b) CMGN (c) ICNN

Figure 4: Results using the adversarial Wasserstein distance for GMM mapping

D Results for Pixel Distribution Mapping

Results using the adversarial Wasserstein distance



(a) MMGN (b) CMGN (c) ICNN

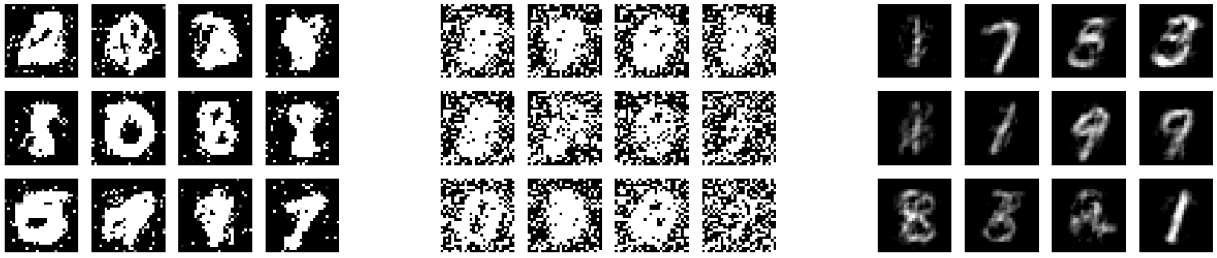
Figure 5: Mapped and target R and G channels

(a) MMGN (b) CMGN (c) ICNN

Figure 6: Results using the adversarial Wasserstein distance

E Results for MNIST generation

Results using the adversarial Wasserstein distance



(a) CMGM (b) MMGN (c) ICNN

Figure 7: Results using the adversarial Wasserstein distance for Gaussian Mapping