Project 10 : Learning an Optimal Transport Brenier Map

Professors : Leclaire Arthur & Gallerne Bruno
Students : Janvier Maya [1] & Boyer Margot [2]

1 Introduction

Convex formulations are one of the main pillars of the analytical world. As convex functions are easy to optimize, they are the framework of many signal processing problems. We can incorporate prior domain knowledge using convex formulations when finding solutions.

Unfortunately, convexity is a strong hypothesis when it comes to poorly known or non-convex domains. This is where deep-learning and data-driven methods comes in handy, as they can deal with non-convex or over-parameterized formulations. This approach has anti-symmetric drawbacks, as we lose interpretability and computational efficiency compared to convex optimization.

Convex functions also play a particular role in Optimal Transport (OT) within the Brenier's theorem :

Theorem 1 (Brenier's theorem)**.** *In the case* $X = Y = \mathbb{R}^d$ *and* $c(x; y) = \|x - y\|_2$, *if* α *has a density with respect to the Lebesgue measure, then there exists a unique optimal Monge map* T. *This map is characterized* ***by being the unique gradient of a convex function*** $T = \nabla\phi$ *such that* $(\nabla\phi)_{\#}\alpha = \beta$.

It is then interesting to learn the gradient of the convex push-forward measure when the push-forward is unknown or difficult to compute. This can be used for domain adaptation, which is a critical topic for the development data augmentation strategies (e.g. the classical daytime-to-twilight color OT).

Normalizing flows have recently achieved a significant breakthrough in generative modeling, succeeding in managing high dimensional data. The goal of normalizing flows is to learn an invertible mapping g_θ from the latent space of the study to the observed space (the data), the two spaces having the same dimension. Currently, normalizing flows inspired by 1 have been developed [3], using the gradient of a learnable convex function to solve Monge OT problems.

Such methods leverage famous architectures such as Input Convex Neural Network (ICNN) [1] or its upgraded version Input Convex Gradient Networks (ICGN) [6] to learn the underlying convex function or its Hessian (respectively), which is a complex and time-consuming process as most weights must be positive and nonlinearities must be convex, monotonically increasing functions.

To tackle these issues, Chaudhari et al. [2] suggested to focus on the *gradient* of the convex functions, as many optimization strategies rely on gradient-based

1. maya.janvier@student-cs.fr
2. margot.boyerblr@gmail.com

methods. By making the best out of the convex and data-driven worlds, they introduced deep-learning architectures for monotone gradient functions. Training these networks is directly learning a gradient of a convex function, leading to smaller and easy to train networks that generalise to higher dimension.

In this work, we implemented the monotone gradient networks [2] along with ICNN and ICGN to assess their performances on several tasks of increasing difficulty :

— learning the gradient of a simple convex function in low dimension
— learning a low-dimensional generative neural network for Brenier's maps (Gaussian case)
— learning a high-dimensional generative neural network for color domain adaptation in images

This is, to the best of our knowledge, the first public implementation of this paper.

2 How one can learn the gradient of a convex function ?

Chaudhari et al. [2] proposes two architectures of neural networks that directly learn monotone gradients. Let f be the convex function whose gradient g we want to learn i.e. $\nabla f = g$, we have : f is convex $\iff H_f(x) \succeq 0 \iff J_g(x) \succeq 0$. As a consequence, to compute the gradient of a convex function, we have to ensure that **the Jacobian of the output is semi-definite positive** (SDP).

2.1 Architectures

C-MGN The article introduces a *Cascaded Monotone Gradient Network* (C-MGN) with L layers as such :

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

All layers share the same weight matrix. With common activation functions (tanh, sigmoid, softplus), the Jacobian of the model is SDP (5).

M-MGN The article introduces a *Modular Monotone Gradient Network* (M-MGN) with K layers as such :

$$\begin{aligned} z_k &= W_kx + b_k \\ \textbf{M-MGN}(x) &= a + V^TVx + \sum_{k=1}^{K} s_k(z_k)\, W_k\, \sigma_k(z_k) \end{aligned}$$

With σ_k an activation function. If the scalar-valued function s_k is convex, twice-differentiable, nonnegative and is linked with the activation function by $\sigma_k = \nabla s_k$, then the jacobian of the model is SDP (5).

2.2 Learning a gradient in low dimension

We reproduce the gradient field experiment from [6] in two dimension that was also reproduced in [2]. We want to estimate the gradient field of $f : \mathbb{R}^2 \to \mathbb{R}$ such that :

$f(x) = x_1^4 + \frac{x_2}{2} + \frac{x_1 x_2}{2} + \frac{3x_2^3}{2} - \frac{x_2^3}{3}$ (see Figure 10 in appendix for ∇f).

As described in [2], C-MGN has 14 parameters, M-MGN has 22 parameters and ICGN has 15 parameters. The training procedure is taken from [6] :

— 1 million points randomly sampled from the unit square
— batch size of 100
— mean absolute error loss
— Adam optimizer with learning rate of 0.02

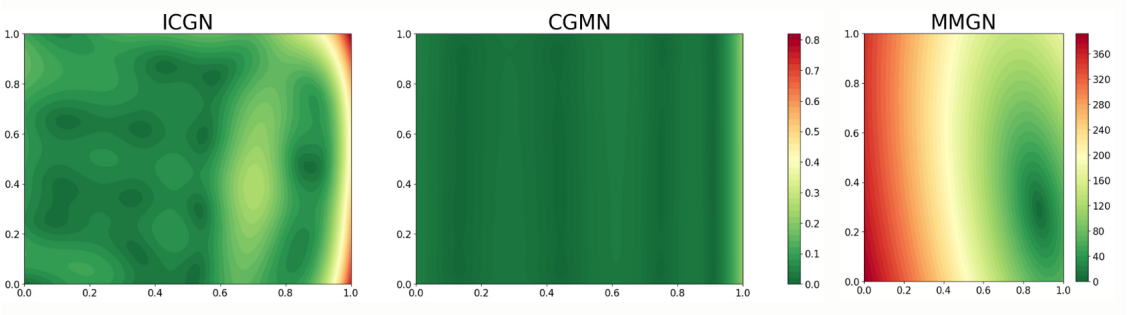


FIGURE 1 – Gradient errors (Fröbenius norm)

We are able to reproduce the results for ICGN and C-MGN : C-MGN computes a better gradient (maximal error of 0.3) field than ICGN with errors up to 0.8. However, our implementation of M-MGN may be incorrect as the errors are too large in general, reaching 360.

3 Computing a low-dimensional generative neural network for Brenier's maps

Following [2], we now consider the Monge OT problem between two gaussian distributions in $\mathbb{R}^2$. In our case, we fitted the gaussian distribution $\alpha = \mathcal{N}\left((0,1), \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}\right)$ to the standard normal distribution $\beta = \mathcal{N}(0_2, I_2)$ (see Figure 2), as the one from the paper was not described.

FIGURE 2 – Source and target gaussian distributions

We tried three training procedures with different losses. Indeed here we can directly derive the Wasserstein 2 cost between gaussians and use it as a loss, or use

the Kullblack-Leibler (KL) divergence between gaussians as it is also explicit in this case. These first losses were computed from scratch. We also tried the implementation from Convex Potential Flow [3] using a normalising flow loss.

3.1 Losses

Wasserstein cost Let $\alpha = \mathcal{N}(m_\alpha, \Sigma_\alpha)$ and $\beta = \mathcal{N}(m_\beta, \Sigma_\beta)$ be two gaussian distributions. The form of the push-forward measure T is known and affine :

$$T : x \to m_\beta + A(x - m_\alpha)$$

T is a gradient of a convex function if and only if A is SDP.
Then their Wasserstein cost $\mathcal{W}^2(\alpha, \beta)$ is :

$$\mathcal{W}^2(\alpha, \beta) = \|m_\alpha - m_\beta\|^2 + \mathcal{B}(\Sigma_\alpha, \Sigma_\beta)^2$$

with $\mathcal{B}\left(\Sigma_\alpha, \Sigma_\beta\right)^2 = \mathrm{Tr}(\Sigma_\alpha + \Sigma_\beta + 2\left(\Sigma_\alpha^{\frac{1}{2}}\ \Sigma_\beta\ \Sigma_\alpha^{\frac{1}{2}}\right))$ the Bure's metric [5].

Kullblack-Leibler divergence The Kullblack-Leibler divergence between gaussians writes (see appendix B for full calculations) :

$$\mathrm{KL}(p_\alpha||q_\beta) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_\beta^{-1}\Sigma_\alpha) + (\mu_\beta - \mu_\alpha)^T\Sigma_\beta^{-1}(\mu_\beta - \mu_\alpha) - 2 + \log\frac{\det(\Sigma_\beta)}{\det(\Sigma_\alpha)}\right)$$

and when β is the standard gaussian, we get :

$$\mathrm{KL}(p_\alpha||\mathcal{N}(0, I)) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_\alpha) + \mu_\alpha^T\mu_\alpha - 2 - \log\det(\Sigma_\alpha)\right)$$

Normalizing flow loss The density of the transported distribution is given by the following change of variable formula 1 :

Let f be a normalizing flow, i.e a differentiable, invertible neural network such that the probability density of the network's output can be computed using :

$$p_\beta(f(x)) = p_\alpha(x)|\frac{\partial f(x)}{\partial x}|^{-1} \iff p_\beta(y) = p_\alpha(f^{-1}(y))|\frac{\partial f^{-1}(y)}{\partial y}| \tag{1}$$

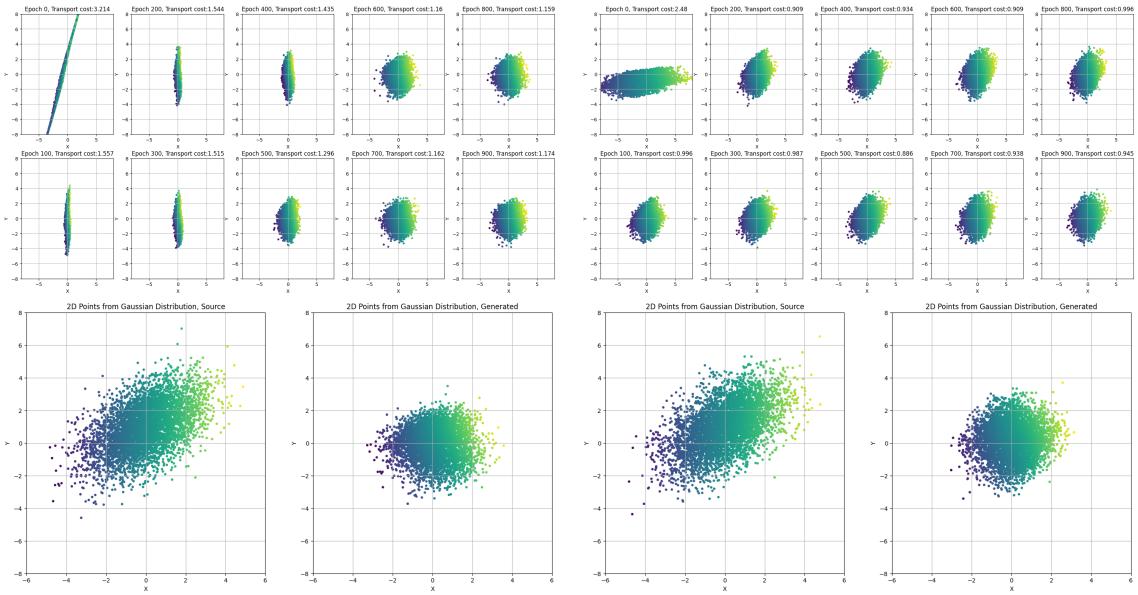
The loss then writes $\log(p_\alpha(f^{-1})) + \log|J(f^{-1})|$ where J denotes the Jacobian.

Transport cost Sometimes, adding a weighted L^2 Euclidian cost acting as a regularizer helped : $c(x) = w_c||f(x) - x||_2$.

3.2 Optimal coupling results

For all experiments, the optimizer was Adam with learning rate 0.02 and the batch size was 100 (except some experiments that will be detailed).

Wasserstein and KL losses We managed to get great results with C-MGN (Figure 3) with both losses. Adding the regularizing parameter with $w_c = 0.01$ also accelerated the training with KL-divergence. We reached a final transport cost of 1.174 for Wasserstein cost and 0.945 for KL-divergence. An example with another source gaussian is done in the notebook, where we discovered that w_c has to be tuned depending on the source (if source too far from target, w_c must be small for the network not to be too conservative).



(a) C-MGN trained with Wasserstein cost, 1000 epochs, $w_c = 0$

(b) C-MGN trained with KL-div, 1000 epochs, $w_c = 0.01$

FIGURE 3 – C-MGN

The M-MGN architecture did not work well using the Wasserstein cost, as it was diverging very far from the target distribution instantly in the beginning, and nor increasing the number of epochs to 10000 and learning rate to 1, neither adding the regularizing term was enough. It is because the Wasserstein cost directly explodes, a behaviour we observed during the gradient field experiment as well.

The KL-divergence however provided good results with no additional regularizing term (Figure 4). We see that we need more epochs to reach a satisfying transport (approximately 4000). We observe once again the "exploding" behaviour of M-MGN that pushed the points too far in the beginning, thus needing thousands of epochs to get to the target. The final transport cost is 1.23, which is worse than C-MGN performances.

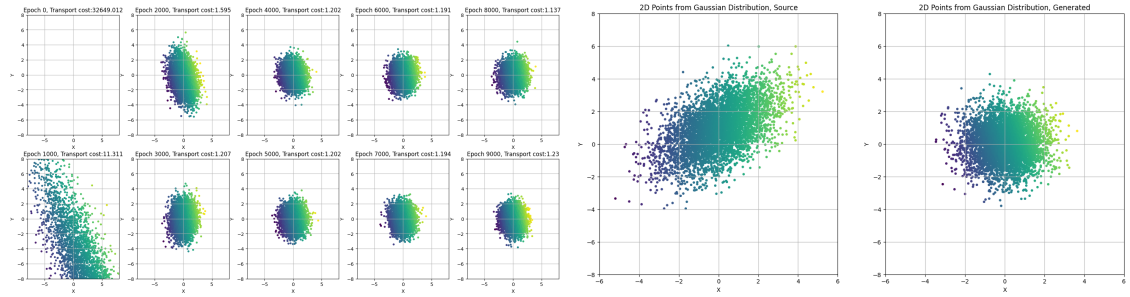


FIGURE 4 – M-MGN trained with KL divergence, 10000 epochs, $w_c = 0$

The ICGN architecture was not performing well under these training procedures, generating pathological "cubic patterns" under the Wasserstein cost or too small variance under KL cost (see appendix C 11). The regularizing parameter did not help, however we can imagine that an additional loss (penalizing the variance for example) would help.

For the experiments in high-dimension (Section 4), we decided to focus on C-MGN with KL-divergence loss rather than optimizing our procedures for M-MGN

and ICGN.

CP-Flow The CP-Flow environment from [3] allowed us to compare C-MGN with ICNN [1] and Linear Inverse Autoregressive Flow (IAF) [4].

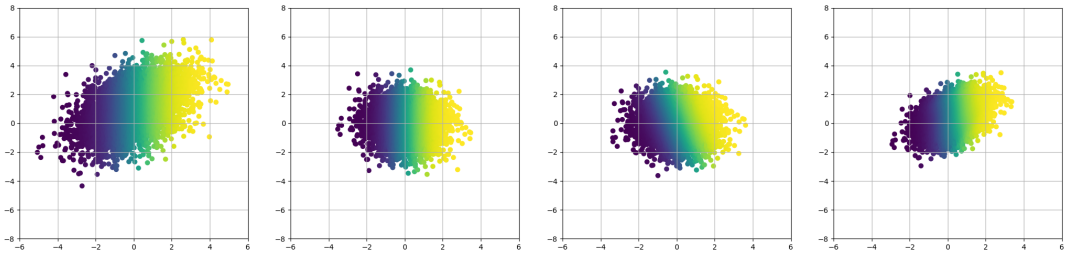


FIGURE 5 – Normalizing flow loss. From left to right, Source distribution, Linear IAF, ICNN, C-MGN

When looking at the points colours from the source (5, left), we observe that our models are increasingly better (from left to right) in reducing the transport cost. Linear IAF sort of projects all the points to the x axis, which is not optimal. The ICNN starts to take into account the transport cost and the C-MGN as well, although it seems it has not entirely converged yet. We did not have time to fine tune its training.

4 Color Domain Transfer with C-MGN

Color Domain Transfer is a critical topic for data augmentation. For example in autonomous driving we need a large variety of images, and also images at all times of the day for better generalization of the models. The idea here is to generate new images by transporting daytime images into a twilight setting. For that, we fit a gaussian distribution to the pixel colors of the target image, and we transport the pixels of the training image to this distribution.

We train the C-MGN using a single target image and a single training image of a road from Dark Zurich [7] dataset's validation set. We use the Kullblack-Leibler divergence loss as it was the best performing loss on our low-dimensional generative network.

4.1 Maximum Likelihood Estimation for target

In our first and best-performing approach, we computed explicitly the fitted gaussian distribution of colors of the training and testing images using Maximum Likelihood Estimation, that is to say we computed the empirical mean and covariance matrices. Our C-MGN has only 27 parameters.

Sunset target With a training of 1000 epochs on the full training image (no batching), we obtain the results displayed in Figure 6 and 7 :

The images kept all of the elements of the training image and applied the colors of the target image, generating a new realistic image of the road. Once the C-MGN is trained, it can be applied to any other image with same results, which is quite remarkable : we implemented a network with only 27 parameters that can perform data augmentation from a single training image.

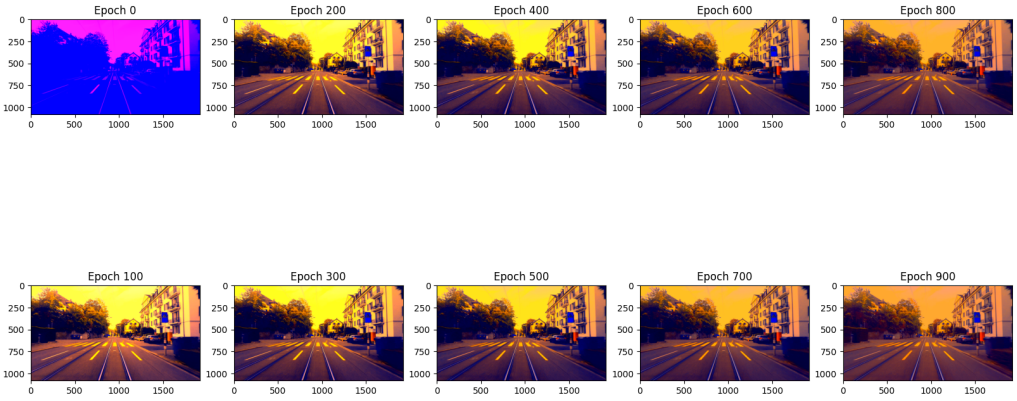


FIGURE 6 – C-MGN sunset - Training evolution)

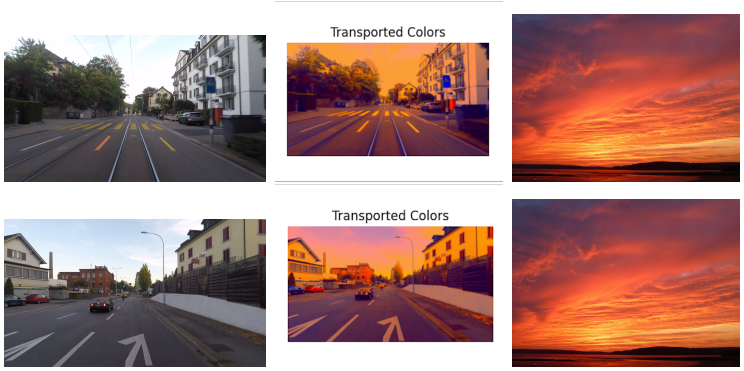


FIGURE 7 – From left to right - **Top row : TRAIN**, training image, transported training image, target image. **Bottom row : TEST**, test image, transported test image, target image

Night target We tried a night target image to see if the model is robust, which was not presented in the original paper. We trained during 2000 epochs to reach better results (Figure 8).

FIGURE 8 – C-MGN night - Transported training image (left), transported test image (middle) and target (right)

We obtained good results with realistic images, although they look more like dawn images. It reveals that the semantical meaning of the target image is not actually important, and what matters most are the colors displayed.

4.2 Multivariate Gaussian Fit for target

In order to explore possibilities of the density induced by the setting sun, we implemented the fit to a multivariate gaussian as in [2]. The number of components of the gaussian mixture used here is 500. In this approach, we also used a batch

approach to accelerate the training, by randomly selecting some pixels in the image to transport at each epoch.

FIGURE 9 – C-MGN sunset - Transported colors (right) of the multivariate gaussian on the test image(left)

We can observe that image we obtained (Figure 9) is still dark, has a strong contrast and is saturated, compared to the first approach. It may be because our batching is not exhaustive in the end.

The model does not necessarily converge towards interesting images and may shift towards monochromatic images tending towards white. This depends on the initialization of the model, and some restart can be required to have good results. The randomness of the parameters could then be adjusted to ensure stability of the model. We did not have time to fine tune the number of components of the gaussian mixture, and tried 10, 50, 500 components. The randomness of the initialization process complicated a possible observation of an interesting direction to search the optimal number of components.

5 Conclusion

We implemented the two main models discussed in the article : C-MGN and M-MGN, and applied them to perform low-dimensional transport of two Gaussian distributions, as well as color transfer on road images to adjust the time setting. The C-MGN performs well and could be improved with a better initialization. One can also imagine to train on more than one target image or training image in order to improve the generalization of the model.

The advantages of these two models are numerous : they entail a reduced number of hyperparameters and exhibit better performance. They can be applied without requiring knowledge of the explicit convex function underlying them. Their potential applications span across vast and promising domains, such as data augmentation.

Github The experiments are reproducible with this Github https://github.com/mayajanvier/Monotone_Gradient_Networks.

Références

[1] Brandon Amos, Lei Xu, and J. Zico Kolter. Input convex neural networks, 2017.

[2] Shreyas Chaudhari, Srinivasa Pranav, and José M. F. Moura. Learning gradients of convex functions with monotone gradient networks, 2023.

[3] Chin-Wei Huang, Ricky T. Q. Chen, Christos Tsirigotis, and Aaron Courville. Convex potential flows : Universal probability distributions with optimal transport and convex optimization, 2021.

[4] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving variational inference with inverse autoregressive flow, 2017.

[5] Gabriel Peyré. Course notes on computational optimal transport, 2021.

[6] Jack Richter-Powell, Jonathan Lorraine, and Brandon Amos. Input convex gradient networks, 2021.

[7] Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Guided curriculum model adaptation and uncertainty-aware evaluation for semantic nighttime image segmentation, 2019.

Appendix

Monotony of C-MGN and M-MGN models

C-MGN

Activation functions such as tanh, sigmoid, softplus are differentiable, monotically increasing and element-wise.

The Jacobian of the model is :

$$J_{\text{C-MGN}} = W^T \left(\sum_{l=1}^{L} \prod_{i=1}^{L} J_{\sigma_l}(z_{i-1}) \right) W + V^T V$$

With σ_l increasing and element-wise, J_{σ_l} is diagonal and with non-negative values, therefore it is PSD. Then $W^T J_{\sigma_l} W \succeq 0$ and by sum of PSD matrices, $W^T \left(\sum_{l=1}^{L} \prod_{i=1}^{L} J_{\sigma_l}(z_{i-1}) \right) W \succeq 0$.

The second term $V^T V$ is PSD as they are Gram matrices. We conclude that $\forall x, J_{\text{C-MGN}} \succeq 0$.

M-MGN

The Jacobian of the M-MGN is :

$$J_{\text{M-MGN}} = \sum_{k=1}^{K} \left(s_k(z_k) W_k^T J_{\sigma_k}(z_k) W_k + (W_k^T \sigma_k(z_k))(W_k^T \sigma_k(z_k))^T \right) + V^T V$$

If s_k is convex, by definition $J_{\sigma_k} \succeq 0$. We can now apply the same reasoning as in the paragraph above, as s_k is a scalar, nonnegative function. $V^T V$ is PSD.

We conclude that $\forall x, J_{\text{M-MGN}} \succeq 0$.

Gradient field

The target field is :

$$\nabla f(x) = g(x_1, x_2) = \begin{pmatrix} 4x_1^3 + \frac{1}{2}x_2 + x_1 \\ 3x_2 - x_2^2 + \frac{1}{2}x_1 \end{pmatrix}$$

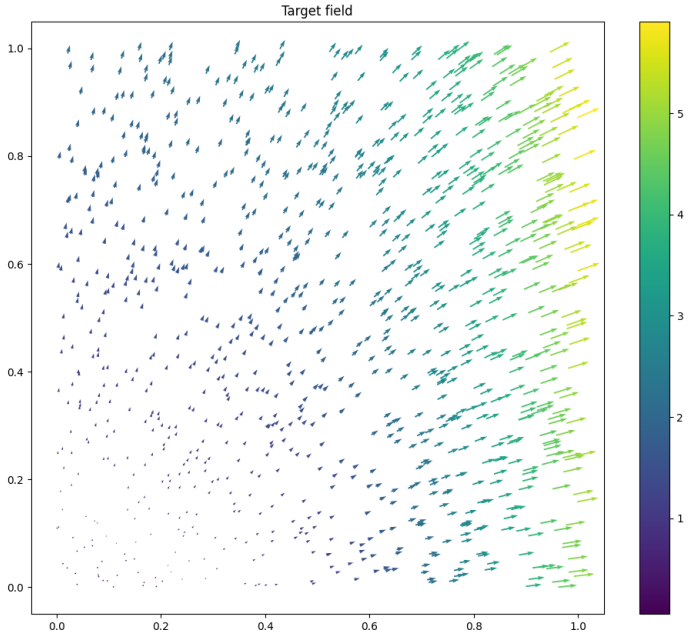


FIGURE 10 – Target field ∇f

KL-divergence

The KL divergence between two multivariate Gaussian distributions p and q of a continuous random variable is given by :

$$D_{KL}(p\|q) = \int_x p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

And the probability density function of a normal distribution writes

$$p(x) = \frac{1}{(2\pi)^{\frac{k}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Then for two k-dimensional Normal distributions $N(\mu_p, \Sigma_p)$ and $N(\mu_q, \Sigma_q)$, the KL divergence can be expressed as :

$$\begin{aligned} D_{\mathrm{KL}}(P||Q) &= \mathbb{E}_p[\log(p) - \log(q)] \\ &= \mathbb{E}_p[\tfrac{1}{2}\log(\tfrac{\Sigma_q}{\Sigma_p}) - \tfrac{1}{2}(x-\mu_p)^T\Sigma_p^{-1}(x-\mu_p) + \tfrac{1}{2}(x-\mu_q)^T\Sigma_q^{-1}(x-\mu_q)] \\ &= \mathbb{E}_p[\tfrac{1}{2}\log(\tfrac{\Sigma_q}{\Sigma_p})] - \mathbb{E}_p[\tfrac{1}{2}(x-\mu_p)^T\Sigma_p^{-1}(x-\mu_p)] + \mathbb{E}_p[\tfrac{1}{2}(x-\mu_q)^T\Sigma_q^{-1}(x-\mu_q)] \\ &= \tfrac{1}{2}\log(\tfrac{\Sigma_q}{\Sigma_p}) - \mathbb{E}_p[\tfrac{1}{2}(x-\mu_p)^T\Sigma_p^{-1}(x-\mu_p)] + \mathbb{E}_p[\tfrac{1}{2}(x-\mu_q)^T\Sigma_q^{-1}(x-\mu_q)] \end{aligned}$$

— Noting that $(x-\mu_p)^T\Sigma_p^{-1}(x-\mu_p) \in \mathbb{R}$, we can reformulate it with the trace : $\mathrm{tr}\left((x-\mu_p)^T\Sigma_p^{-1}(x-\mu_p)\right)$. Then, by commutativity of the trace : $\mathrm{tr}\left((x-\mu_p)(x-\mu_p)^T\Sigma_p^{-1}\right)$.

We can now interchange the trace and the expectation :

$$\begin{aligned}(x-\mu_p)^T\Sigma_p^{-1}(x-\mu_p) &= \mathrm{tr}\left((x-\mu_p)(x-\mu_p)^T\Sigma_p^{-1}\right)\\ &= \mathbb{E}_p[\mathrm{tr}\left((x-\mu_p)(x-\mu_p)^T\Sigma_p^{-1}\right)]\\ &= \mathrm{tr}\left(\mathbb{E}_p[(x-\mu_p)(x-\mu_p)^T\Sigma_p^{-1}]\right)\\ &= \mathrm{tr}\left(\mathbb{E}_p[(x-\mu_p)(x-\mu_p)^T]\Sigma_p^{-1}\right)\end{aligned}$$

By definition $\Sigma_p = \mathbb{E}_p[(x-\mu_p)(x-\mu_p)^T]$. We can now conclude :

$$\begin{aligned}(x-\mu_p)^T\Sigma_p^{-1}(x-\mu_p) &= \mathrm{tr}\left(\Sigma_p\Sigma_p^{-1}\right)\\ &= \mathrm{tr}\left(I\right)\\ &= k\end{aligned}$$

— To modify the last term of the equation, we use this formula : if x is a gaussian $\mathcal{N}(m,\Sigma)$,

$$\mathbb{E}[(x-\bar{m})A(x-\bar{m})] = (m-\bar{m})^TA(m-\bar{m}) + \mathrm{tr}(A\Sigma)$$

,

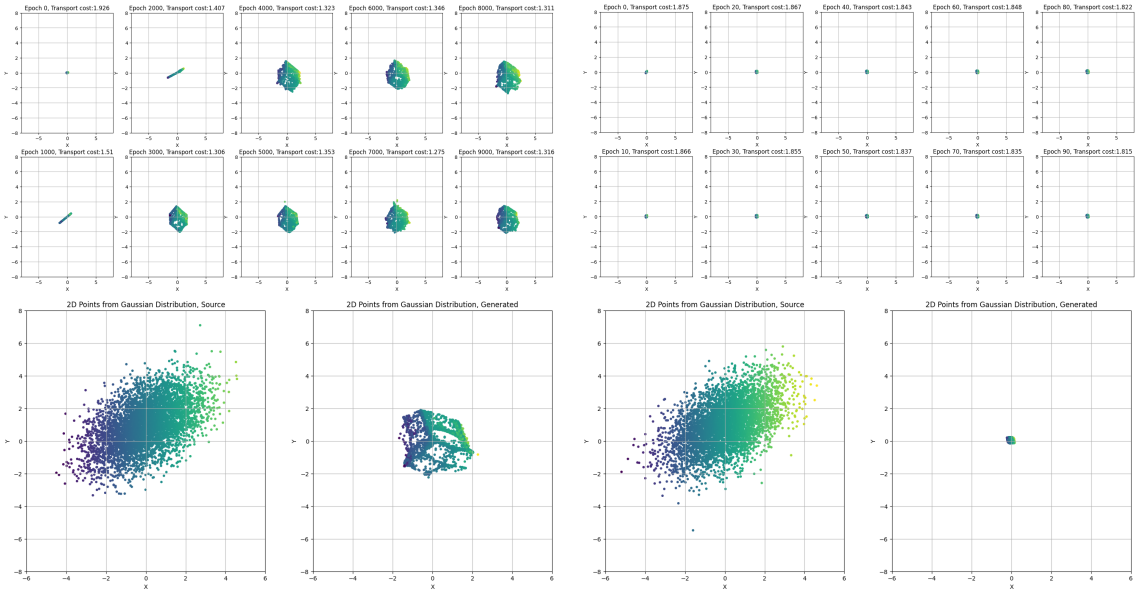
$$\mathbb{E}_p[\frac{1}{2}(x-\mu_q)^T\Sigma_q^{-1}(x-\mu_q)] = (\mu_p-\mu_q)^T\Sigma_q^{-1}(\mu_p-\mu_q) + \mathrm{tr}(\Sigma_q^{-1}\Sigma_p)$$

We can conclude :

$$D_{\mathrm{KL}}(P||Q) = \frac{1}{2}\left[\log\frac{|\Sigma_q|}{|\Sigma_p|} - k + (\mu_p-\mu_q)^T\Sigma_q^{-1}(\mu_p-\mu_q) + \mathrm{tr}\{\Sigma_q^{-1}\Sigma_p\}\right]$$

ICGN for optimal coupling betwen 2D gaussians

The behaviour of ICGN with KL-div is apparently not due to an overfitting as I reduced the batch size to 10 but the output still had a too small variance.



(a) ICGN trained with Wasserstein cost, 10000 epochs, batch size 100, $w_c = 0$

(b) ICGN trained with KL-div, 100 epochs, batch size 10, $w_c = 0$

FIGURE 11 – ICGN