«Creative» Neural Networks

Machine Learning Techniques (for High Energy Physics)

Adriano Di Florio

Intro to Generative Models

What?

Progress (as of 2018...)

2014

2015

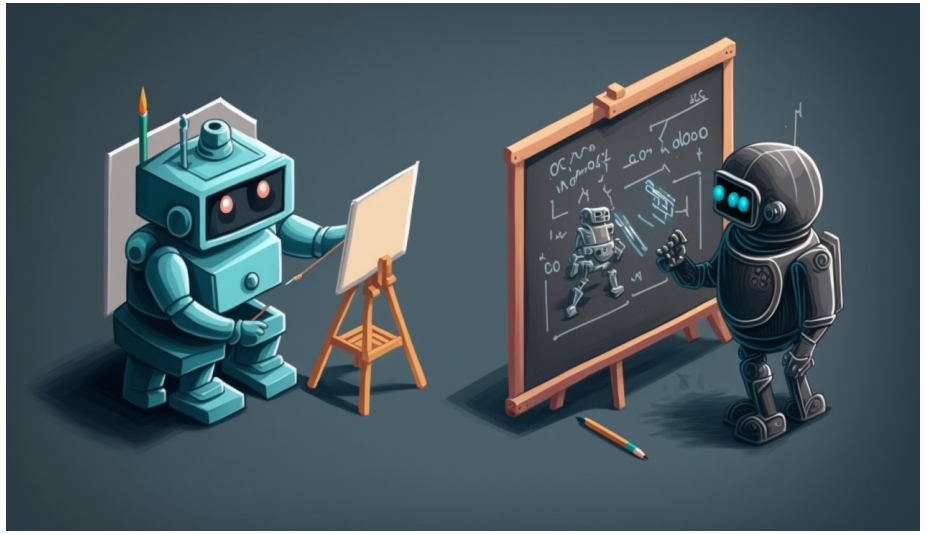
2016

2017

2018

https://twitter.com/goodfellow_ian/status/1084973596236144640

Progress (text-to-image, 2023)



"Robot artist painting a picture and another robot mathematician writing formulas on a blackboard, isometric cartoon" – as interpreted by Midjourney

"Isometric clumsy black-box robot painter at work, minimalistic diagram with white background" – as interpreted by Midjourney

What a generative model does

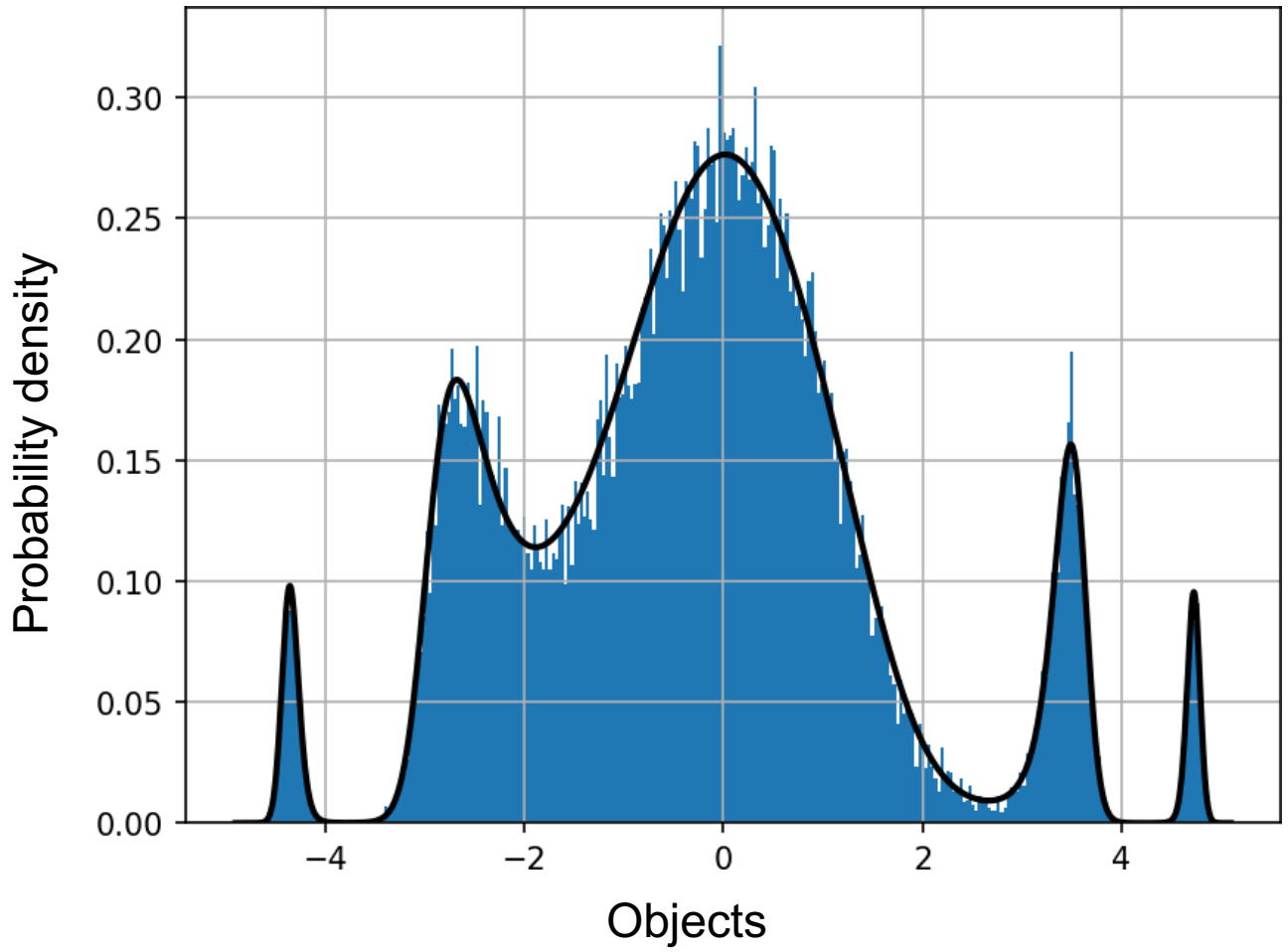
Black-box model

Objects

"Black box with gears and switches, isometric drawing, minimalistic, cartoon, white background" – as interpreted by Midjourney

"A pile of Mona Lisa drawings, minimalistic cartoon, white background, isometric" – as interpreted by Midjourney

What a generative model does



Parameters of the model control the shape of the distribution

May also depend on conditionals (model maps conditionals to distributions)

Learns distribution from data

Generative modelling in HEP

- Screenshot from https://github.com/iml-_wg/HEPML-LivingReview
- Each entry = somebody's work

- Generative models / density estimation
  - GANs:
    - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis [DOI]
    - Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multilayer Calorimeters [DOI]
    - CaloGAN : Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks [DOI]
    - Image-based model parameter optimization using Model-Assisted Generative Adversarial Networks [DOI]
    - How to GAN Event Subtraction [DOI]
    - Particle Generative Adversarial Networks for full-event simulation at the LHC and their application to pileup description [DOI]
    - How to GAN away Detector Effects [DOI]
    - 3D convolutional GAN for fast simulation
    - Fast simulation of muons produced at the SHiP experiment using Generative Adversarial Networks [DOI]
    - Lund jet images from generative and cycle-consistent adversarial networks [DOI]
    - How to GAN LHC Events [DOI]
    - Machine Learning Templates for QCD Factorization in the Search for Physics Beyond the Standard Model [DOI]
    - DijetGAN: A Generative-Adversarial Network Approach for the Simulation of QCD Dijet Events at the LHC [DOI]
    - LHC analysis-specific datasets with Generative Adversarial Networks
    - Generative Models for Fast Calorimeter Simulation.LHCb case [DOI]
    - Deep generative models for fast shower simulation in ATLAS
    - Regressive and generative neural networks for scalar field theory [DOI]
    - Three dimensional Generative Adversarial Networks for fast simulation
    - Generative models for fast simulation
    - Unfolding with Generative Adversarial Networks
    - Fast and Accurate Simulation of Particle Detectors Using Generative Adversarial Networks [DOI]
    - Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks [DOI]
    - Generative models for fast cluster simulations in the TPC for the ALICE experiment
    - RICH 2018 [DOI]
    - GANs for generating EFT models [DOI]
    - Precise simulation of electromagnetic calorimeter showers using a Wasserstein Generative Adversarial Network [DOI]
    - Reducing Autocorrelation Times in Lattice Simulations with Generative Adversarial Networks [DOI]
    - Tips and Tricks for Training GANs with Physics Constraints
    - Controlling Physical Attributes in GAN-Accelerated Simulation of Electromagnetic Calorimeters [DOI]
    - Next Generation Generative Neural Networks for HEP
    - Calorimetry with Deep Learning: Particle Classification, Energy Regression, and Simulation for High-Energy Physics
    - Calorimetry with Deep Learning: Particle Simulation and Reconstruction for Collider Physics [DOI]
    - Getting High: High Fidelity Simulation of High Granularity Calorimeters with High Speed
    - AI-based Monte Carlo event generator for electron-proton scattering
    - DCTRGAN: Improving the Precision of Generative Models with Reweighting [DOI]
    - GANplifying Event Samples
    - Graph Generative Adversarial Networks for Sparse Data Generation in High Energy Physics
    - Simulating the Time Projection Chamber responses at the MPD detector using Generative Adversarial Networks
    - Explainable machine learning of the underlying physics of high-energy particle collisions
    - A Data-driven Event Generator for Hadron Colliders using Wasserstein Generative Adversarial Network [DOI]
    - Reduced Precision Strategies for Deep Learning: A High Energy Physics Generative Adversarial Network Use Case [DOI]
    - Validation of Deep Convolutional Generative Adversarial Networks for High Energy Physics Calorimeter Simulations
    - Compressing PDF sets using generative adversarial networks
    - Physics Validation of Novel Convolutional 2D Architectures for Speeding Up High Energy Physics Simulations
  - Autoencoders
    - Deep Learning as a Parton Shower
    - Deep generative models for fast shower simulation in ATLAS
    - Variational Autoencoders for Anomalous Jet Tagging
    - Variational Autoencoders for Jet Simulation
    - Foundations of a Fast, Data-Driven, Machine-Learned Simulator
    - Decoding Photons: Physics in the Latent Space of a BIB-AE Generative Network
    - Bump Hunting in Latent Space
    - {End-to-end Sinkhorn Autoencoder with Noise Generator
    - Graph Generative Models for Fast Detector Simulations in High Energy Physics
    - DeepRICH: Learning Deeply Cherenkov Detectors [DOI]
  - Normalizing flows
    - Flow-based generative models for Markov chain Monte Carlo in lattice field theory [DOI]
    - Equivariant flow-based sampling for lattice gauge theory [DOI]
    - Flows for simultaneous manifold learning and density estimation
    - Exploring phase space with Neural Importance Sampling [DOI]
    - Event Generation with Normalizing Flows [DOI]
    - i-flow: High-Dimensional Integration and Sampling with Normalizing Flows [DOI]
    - Anomaly Detection with Density Estimation [DOI]
    - Data-driven Estimation of Background Distribution through Neural Autoregressive Flows
    - SARM: Sparse Autoregressive Model for Scalable Generation of Sparse Images in Particle Physics [DOI]
    - Measuring QCD Splittings with Invertible Networks
    - Efficient sampling of constrained high-dimensional theoretical spaces with machine learning
  - Physics-inspired
    - JUNIPR: a Framework for Unsupervised Machine Learning in Particle Physics
    - Binary JUNIPR: an interpretable probabilistic model for discrimination [DOI]
    - Exploring the Possibility of a Recovery of Physics Process Properties from a Neural Network Model [DOI]
    - Explainable machine learning of the underlying physics of high-energy particle collisions
    - Symmetry meets AI
  - Mixture Models
    - Data Augmentation at the LHC through Analysis-specific Fast Simulation with Deep Learning
    - Mixture Density Network Estimation of Continuous Variable Maximum Likelihood Using Discrete Training Samples
  - Phase space generation
    - Efficient Monte Carlo Integration Using Boosted Decision
    - Exploring phase space with Neural Importance Sampling [DOI]
    - Event Generation with Normalizing Flows [DOI]
    - i-flow: High-Dimensional Integration and Sampling with Normalizing Flows [DOI]
    - Neural Network-Based Approach to Phase Space Integration [DOI]
    - VegasFlow: accelerating Monte Carlo simulation across multiple hardware platforms [DOI]
    - A Neural Resampler for Monte Carlo Reweighting with Preserved Uncertainties [DOI]
    - Improved Neural Network Monte Carlo Simulation [DOI]
    - Phase Space Sampling and Inference from Weighted Events with Autoregressive Flows [DOI]
    - How to GAN Event Unweighting
  - Gaussian processes
    - Modeling Smooth Backgrounds and Generic Localized Signals with Gaussian Processes
    - Accelerating the BSM interpretation of LHC data with machine learning [DOI]
    - $\textsf{Xsec}$: the cross-section evaluation code [DOI]
    - AI-optimized detector design for the future Electron-Ion Collider: the dual-radiator RICH case [DOI]

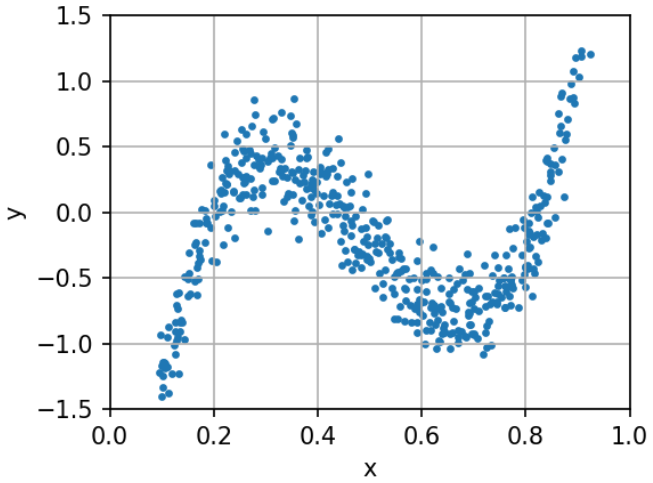
Intro to Generative Modelling

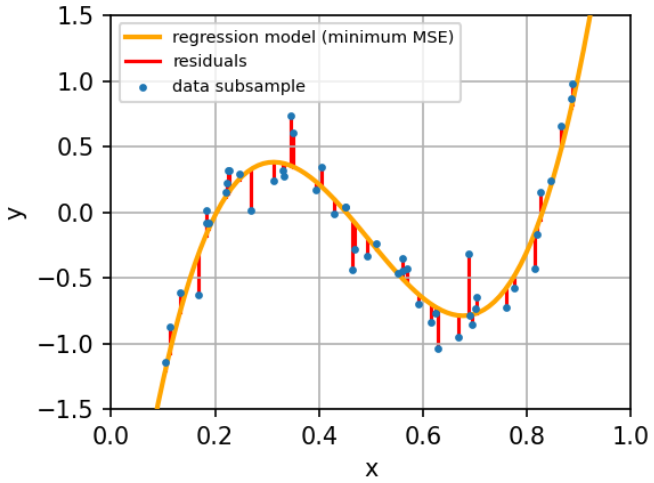
How?

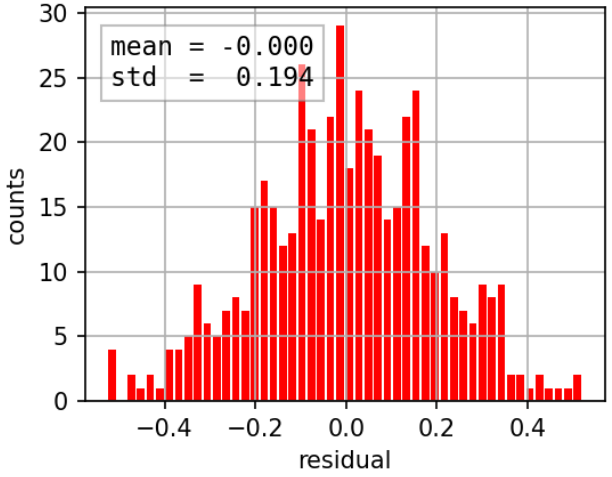
From supervised to generative

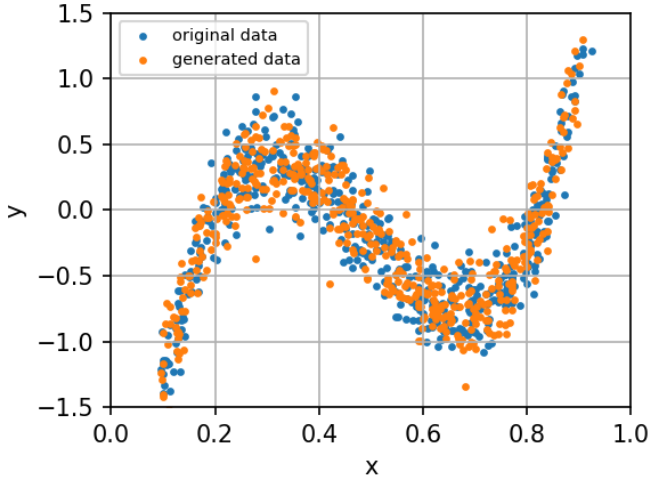
- A regression model may be $\hat{y} = \hat{f}_\theta(x)$ interpreted as a generative one, e.g.:

$$p^{\mathbf{model}}(y|x) = \mathcal{N}\left(y|\mu = \hat{f}_\theta(x), \sigma^2\right)$$









Defining PDF with a NN

- Straightforward way: can we just take a neural net $f_\theta : \mathbb{R}^d \rightarrow \mathbb{R}$ and treat is as some PDF $p_\theta(x) \equiv f_\theta(x)$?
  - Needs to be non-negative
    - can easily achieve this with a proper activation at the last layer
  - Needs to be normalized $\int f_\theta \, x \, d\, x = 1$
    - how on earth do we satisfy this???

Defining PDF with a NN

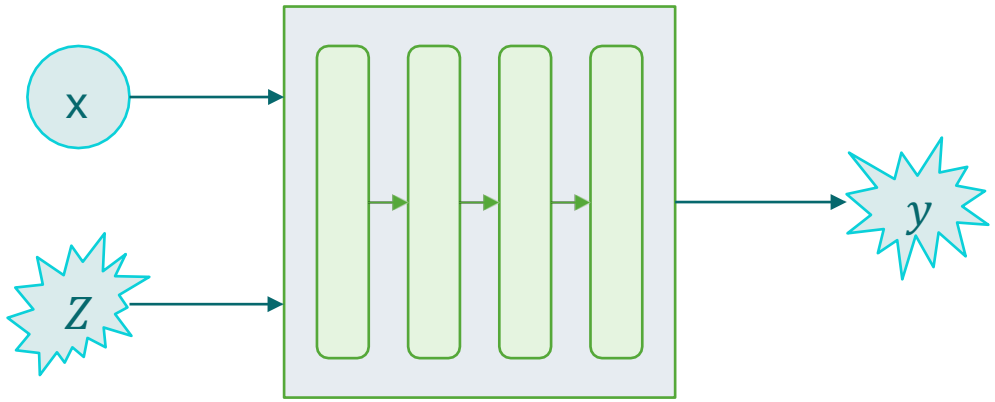
- A bit less straightforward way
  - $\tilde{p}_\theta(x) \equiv f_\theta(x)$ – the NN models the unnormalized PDF value
  - $p_\theta(x) \equiv \frac{\tilde{p}_\theta(x)}{Z_\theta}$ – the true PDF
  - $Z_\theta = \int \tilde{p}_\theta(x')dx'$ – normalization constant (intractable)
- There exist methods to sample from and train such models
- Computationally demanding
  - See, e.g., Metropolis-Hastings algorithm and Contrastive Divergence
  - Check out https://www.deeplearningbook.org/, chapters 16-18 for more info

Defining PDF with a NN

- Alternative:
  - make $y = f_\theta(x)$ a stochastic function
  - treat output as samples from $p_\theta(y)$

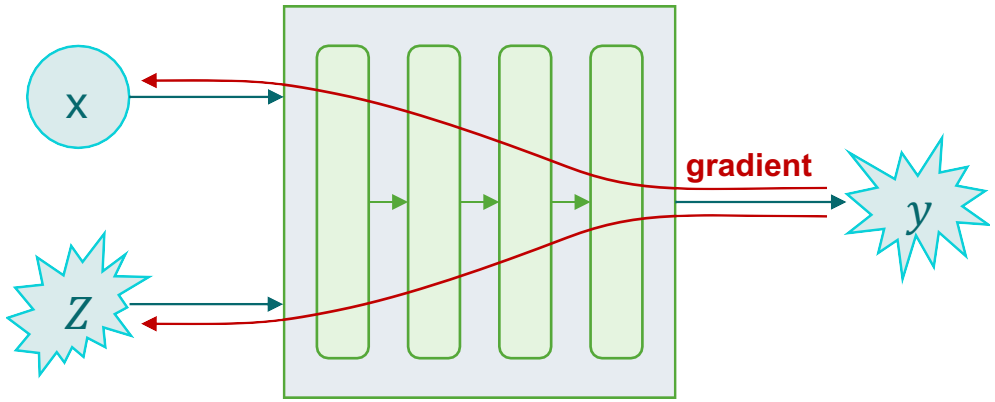
Introducing stochasticity into a NN

- $f_\theta(x) \equiv f_\theta(\mathrm{x}, \mathrm{Z})$
  - x – deterministic input (if needed)
  - Z – random variable of some known fixed distribution



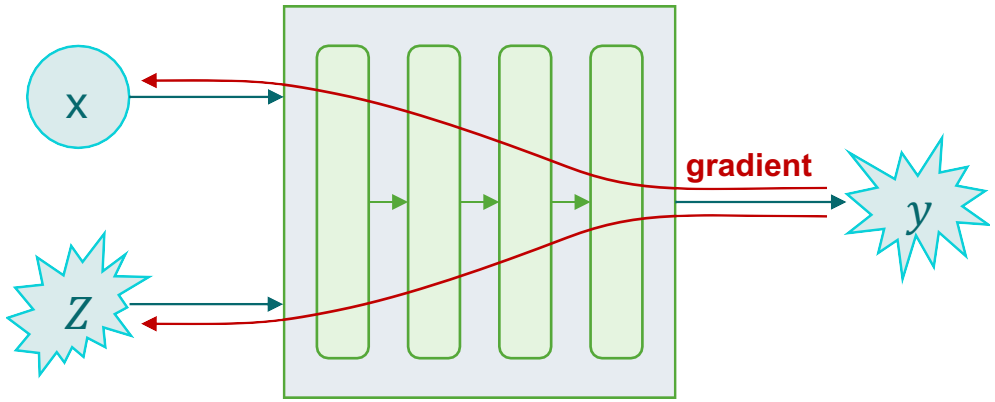
Introducing stochasticity into a NN

- $f_\theta(x) \equiv f_\theta(\mathrm{x}, \mathrm{Z})$
  - x – deterministic input (if needed)
  - Z – random variable of some known fixed distribution



Introducing stochasticity into a NN

- $f_\theta(x) \equiv f_\theta(\mathrm{x}, \mathrm{Z})$
  - x – deterministic input (if needed)
  - Z – random variable of some known fixed distribution



- How does one train this?

Comparing objects?

- Take some loss function $l(y, y')$ – "how similar y and y' are"
- Then, can't we just minimize this:

$$\mathcal{L} = \mathbb{E}_{y \sim p_{\text{data}},\, y' \sim p_\theta}[l(y, y')]$$

(by sampling y and y' and calculating averages)?

Comparing objects?

$$\min_{\theta} \mathcal{L} = \min_{\theta} \int p_{\theta}(y') \int p_{\text{data}}(y) l(y, y') dy \, dy'$$

Comparing objects?

$$\min_\theta \mathcal{L} = \min_\theta \int p_\theta(y') \int p_{\text{data}}(y) l(y, y') dy \, dy'$$

some function $F(y')$

$$= \min_\theta \int p_\theta(y') \, F(y') dy'$$

Comparing objects?

$$\min_\theta \mathcal{L} = \min_\theta \int p_\theta(y') \int p_{\text{data}}(y) l(y, y') dy \, dy'$$

some function $F(y')$

$$= \min_\theta \int p_\theta(y') \, F(y') dy'$$

Comparing objects?

$$\min_\theta \mathcal{L} = \min_\theta \int p_\theta(y') \int p_{\text{data}}(y) l(y, y') dy \, dy'$$

some function $F(y')$

$$= \min_\theta \int p_\theta(y') \, F(y') dy'$$

- Minimized when $p_\theta(y') \equiv \delta(y' - \text{argmin} F)$
- Such model will produce the same sample all the time

Comparing objects?

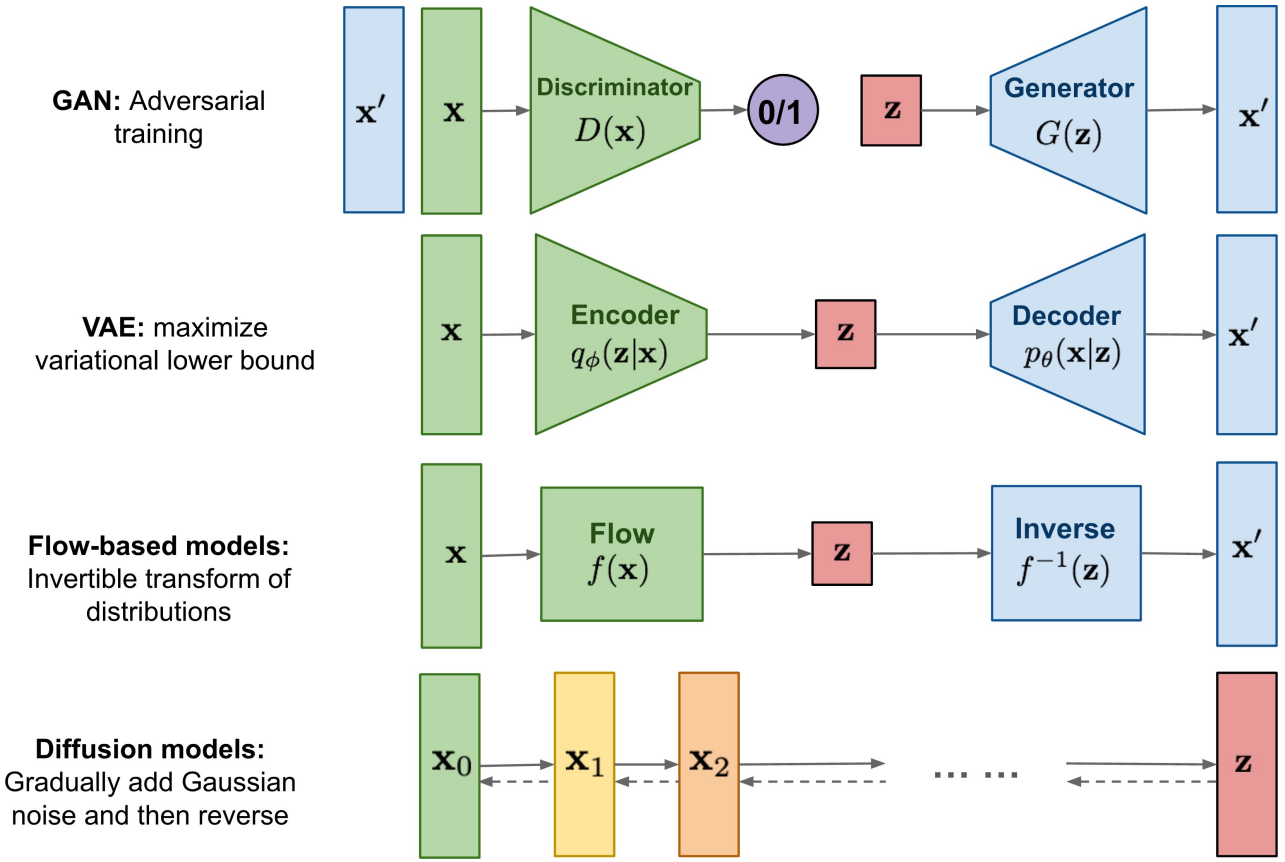
$$\min_\theta \mathcal{L} = \min_\theta \int p_\theta(y') \int p_{\text{data}}(y) l(y, y') dy \, dy'$$

some function $F(y')$

$$= \min_\theta \int p_\theta(y') \, F(y') dy'$$

- Minimized when $p_\theta(y') \equiv \delta(y' - \text{argmin} F)$
- Such model will produce the same sample all the time
- Comparing objects is not enough; we need to compare distributions

Approaches to training generative models



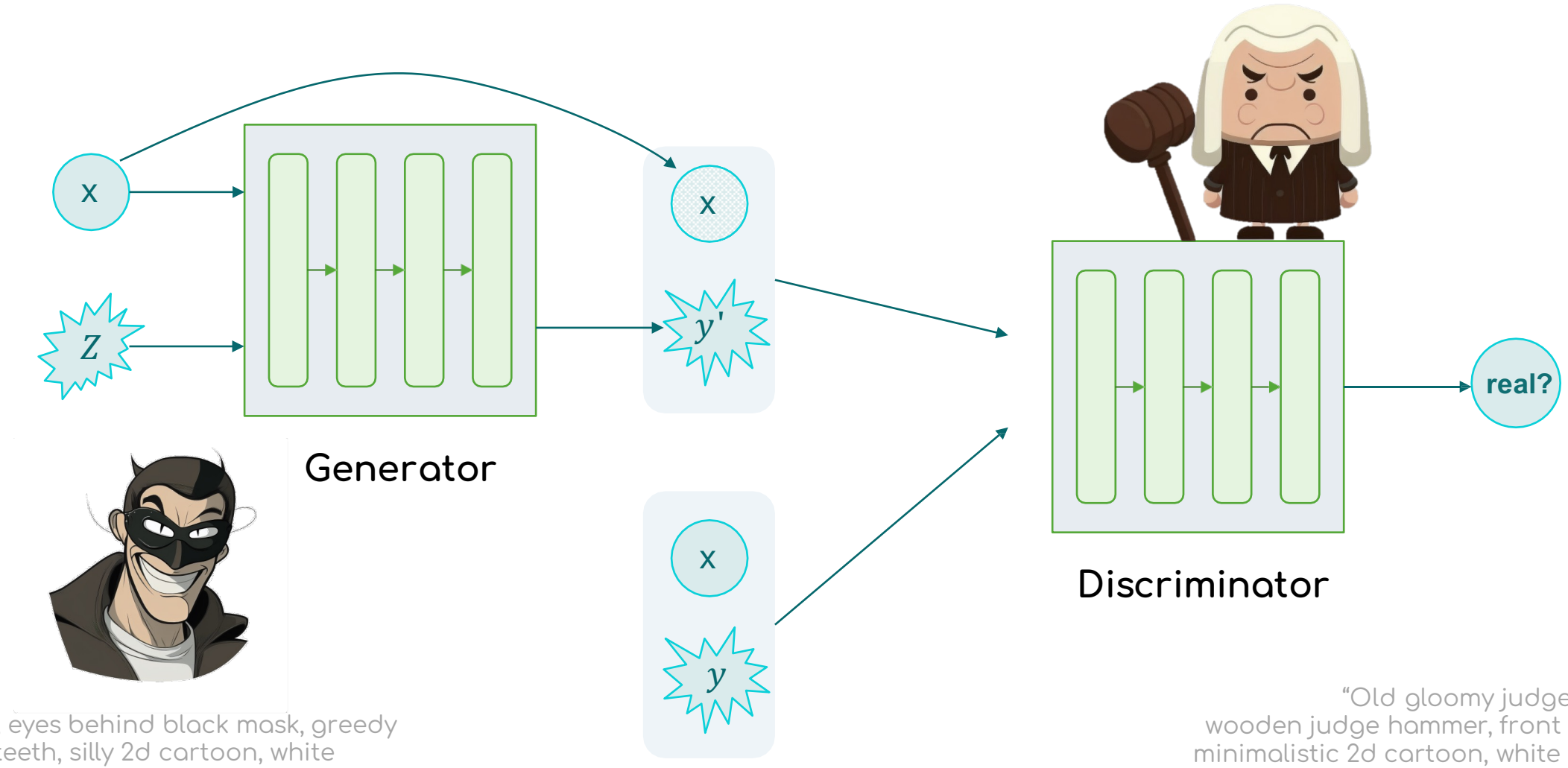
https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Generative Adversarial Networks

Our setting so far

- Training sample $\{y_1, y_2, \dots, y_N\}$ with independent and identically distributed variables (i.i.d.) $y_i \sim p_{\text{data}}$
- A model that can sample arbitrary many $y' \sim p_\theta$
- Ability to calculate gradients $\frac{\partial y'}{\partial \theta}$ for any given generated sample j
- We want to find θ such that $p_\theta \sim p_{\text{data}}$

Adversarial approach



"Sly burglar, eyes behind black mask, greedy grin, white teeth, silly 2d cartoon, white background" – as interpreted by Midjourney

"Old gloomy judge in wig with a wooden judge hammer, front portrait view, minimalistic 2d cartoon, white background" – as interpreted by Midjourney

Adversarial approach

In a nutshel:

- A discriminator **D** estimates the probability of a given sample coming from the real dataset. It works as a critic/judge and is optimized to tell the fake samples from the real ones.

- A generator **G** outputs synthetic samples given a noise variable input z (**z brings in potential output diversity**). It is trained to capture the real data distribution so that its generative samples can be as real as possible, or in other words, can trick the discriminator to offer a high probability.

Mathematical formulation

- On one hand, we want to make sure the discriminator D's decisions over real data are accurate. Meanwhile, given a fake sample , the discriminator is expected to output a probability close to zero.

- Objective function:

$$\mathcal{L} = \mathbb{E}_{y \sim p_{\text{data}}} \left[\log D_\psi(y) \right] + \mathbb{E}_{y' \sim p_\theta} \left[\log \left(1 - D_\psi(y') \right) \right]$$

  - ψ– parameters of the discriminator
  - θ– parameters of the generator

- When combining both aspects together, D and G are playing a minimax game in which we should optimize the following loss function:

  - $f \rightarrow \max_{\psi} \min_{\theta}$

  - Two models are trained simultaneously to find a Nash equilibrium to a two-player non-cooperative game.

Mathematical formulation

Note: the paper uses "x" for the target objects

We used "y" for the objects to allow for features "x" in case of conditional generation

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

https://arxiv.org/abs/1406.2661 (original GAN paper)

Optimal discriminator

- Optimal discriminator:

$$\underset{D_\psi}{\operatorname{argmax}}\,\mathcal{L} = \underset{D_\psi}{\operatorname{argmax}} \int \left[p_{\text{data}}(y) \log D_\psi(y) + p_\theta(y) \log\left(1 - D_\psi(y)\right) \right] dy$$

$$= \underset{D_\psi}{\operatorname{argmax}} \underbrace{\left[p_{\text{data}}(y) \log D_\psi(y) + p_\theta(y) \log\left(1 - D_\psi(y)\right) \right]}_{\ell\left(D_\psi(y)\right)}$$

$$\frac{\partial}{\partial D_\psi(y)} \left[\ell\left(D_\psi(y)\right) \right] = \frac{p_{\text{data}}(y)}{D_\psi(y)} - \frac{p_\theta(y)}{1 - D_\psi(y)} = 0 \quad \Rightarrow \quad \boxed{D_\psi(y) = \frac{p_{\text{data}}(y)}{p_{\text{data}}(y) + p_\theta(y)}}$$

KL and JSD

Two metrics for quantifying the similarity between two probability distributions.

- KL (Kullback–Leibler) divergence measures how one probability distribution p diverges from a second expected probability distribution q. Asymmetric.

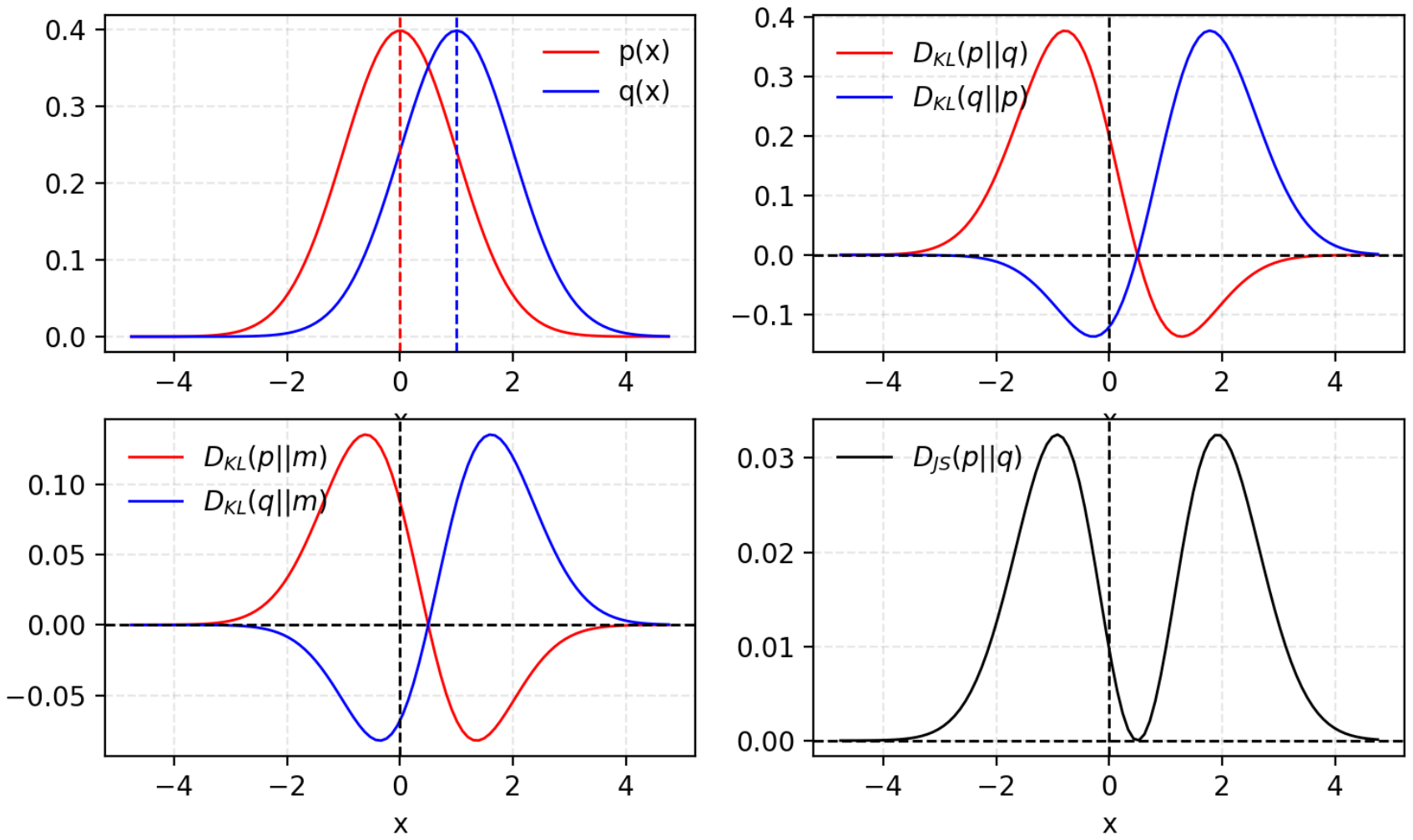
$$D_{KL}(p\|q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

KL achieves the minimum zero when p(x) == q(x) everywhere.

- Jensen–Shannon Divergence(JSD) is another measure of similarity between two probability distributions, bounded by [0,1]. Symmetric!

$$D_{JS}(p\|q) = \frac{1}{2} D_{KL}(p\|\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q\|\frac{p+q}{2})$$

KL and JSD



KL and JSD

- With the "optimal" discriminator, we have our objective as:

$$\mathcal{L} = \mathbb{E}_{y \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(y)}{p_{\text{data}}(y) + p_\theta(y)} \right] + \mathbb{E}_{y' \sim p_\theta} \left[\log \frac{p_\theta(y)}{p_{\text{data}}(y) + p_\theta(y)} \right]$$

KL and JSD

- With the "optimal" discriminator, we have our objective as:

$$\mathcal{L} = \mathbb{E}_{y\sim p_{\text{data}}}\left[\log\frac{p_{\text{data}}(y)}{p_{\text{data}}(y)+p_\theta(y)}\right] + \mathbb{E}_{y'\sim p_\theta}\left[\log\frac{p_\theta(y)}{p_{\text{data}}(y)+p_\theta(y)}\right]$$

$$= -\log 4 + 2\cdot \text{JSD}(p_{\text{data}}\|p_\theta) \rightarrow \min_\theta$$

Jensen–Shannon divergence

$$\text{JSD}(p\|q) \equiv \frac{1}{2}\left[D_{\text{KL}}\left(p\middle\|\frac{p+q}{2}\right) + D_{\text{KL}}\left(q\middle\|\frac{p+q}{2}\right)\right]$$

KL and JSD

- With the "optimal" discriminator, we have our objective as:

$$\mathcal{L} = \mathbb{E}_{y\sim p_{\text{data}}}\left[\log\frac{p_{\text{data}}(y)}{p_{\text{data}}(y)+p_\theta(y)}\right] + \mathbb{E}_{y'\sim p_\theta}\left[\log\frac{p_\theta(y)}{p_{\text{data}}(y)+p_\theta(y)}\right]$$

$$= -\log 4 + 2\cdot \text{JSD}(p_{\text{data}}\|p_\theta) \rightarrow \min_\theta$$

Jensen–Shannon divergence

$$\text{JSD}(p\|q) \equiv \frac{1}{2}\left[D_{\text{KL}}\left(p\middle\|\frac{p+q}{2}\right) + D_{\text{KL}}\left(q\middle\|\frac{p+q}{2}\right)\right]$$

Kullback–Leibler divergence

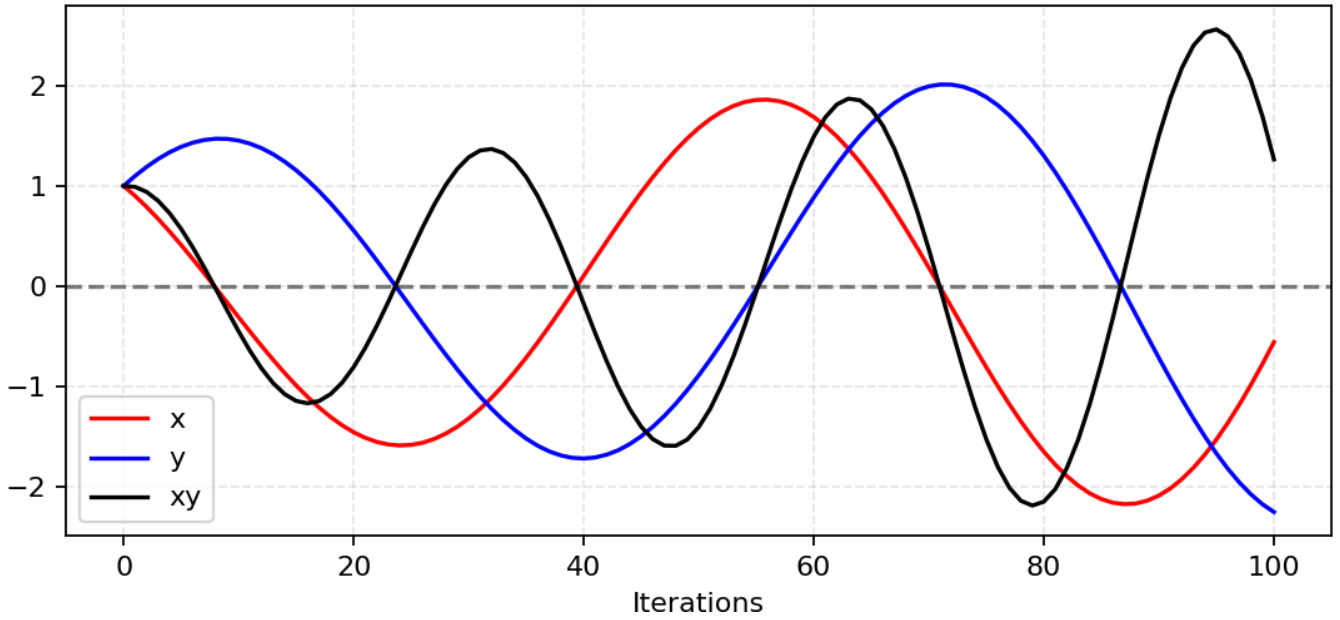
$$D_{\text{KL}}(p\|q) = \mathbb{E}_{X\sim p}\left[\log\frac{p(X)}{q(X)}\right]$$

Nash equilibrium is tough

- Each model updates its cost independently with no respect to another player in the game. Updating the gradient of both models concurrently cannot guarantee a convergence.

- Let's consider a simple problem. Suppose one player wants to minimize $f_1(x) = xy$ by updating x, while at the same time the other player constantly updates y to minimize $f_2(x) = -xy$.

- Because $\partial f_1/\partial x = y$ and $\partial f_2/\partial x = -y$, we update x with $x - \eta \cdot y$ and y with $y + \eta \cdot x$ simulitanously in one iteration, where η is the learning rate. Once x and y have different signs, every following gradient update causes huge oscillation and the instability gets worse in time

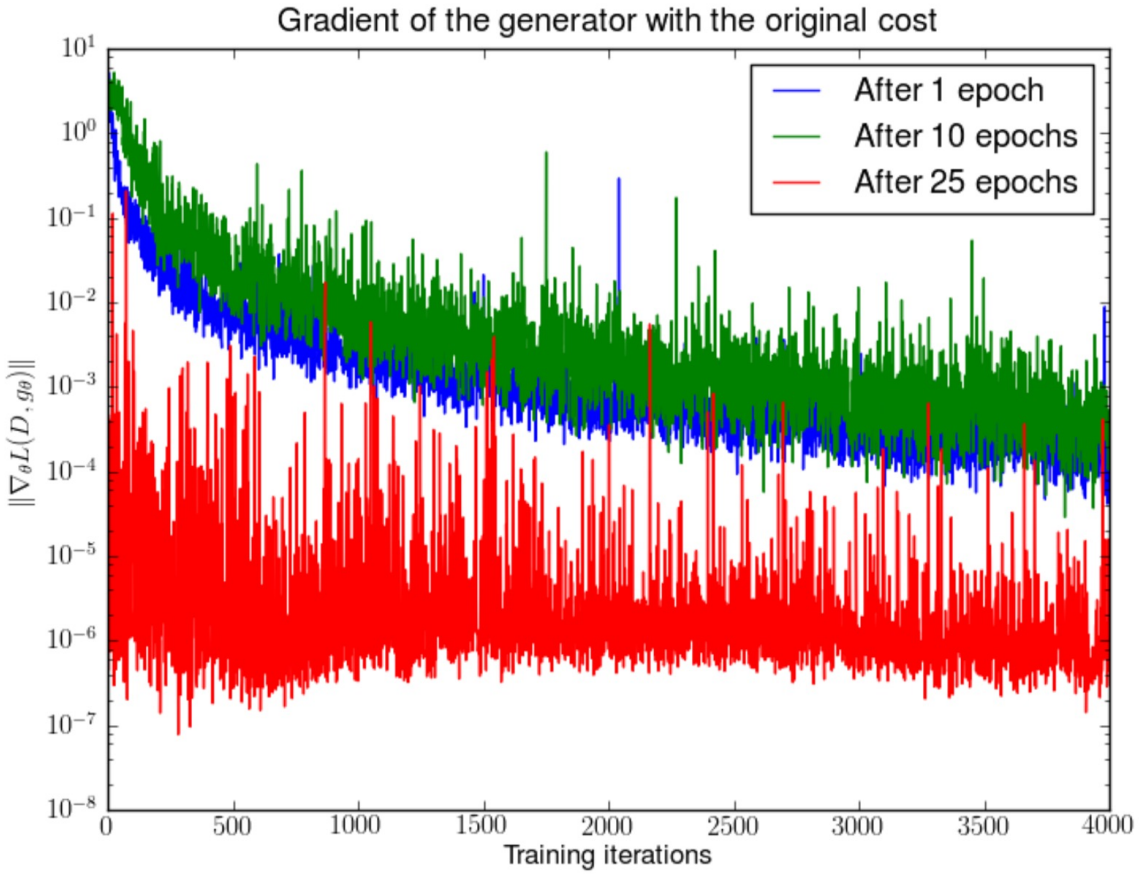


Vanishing Gradients

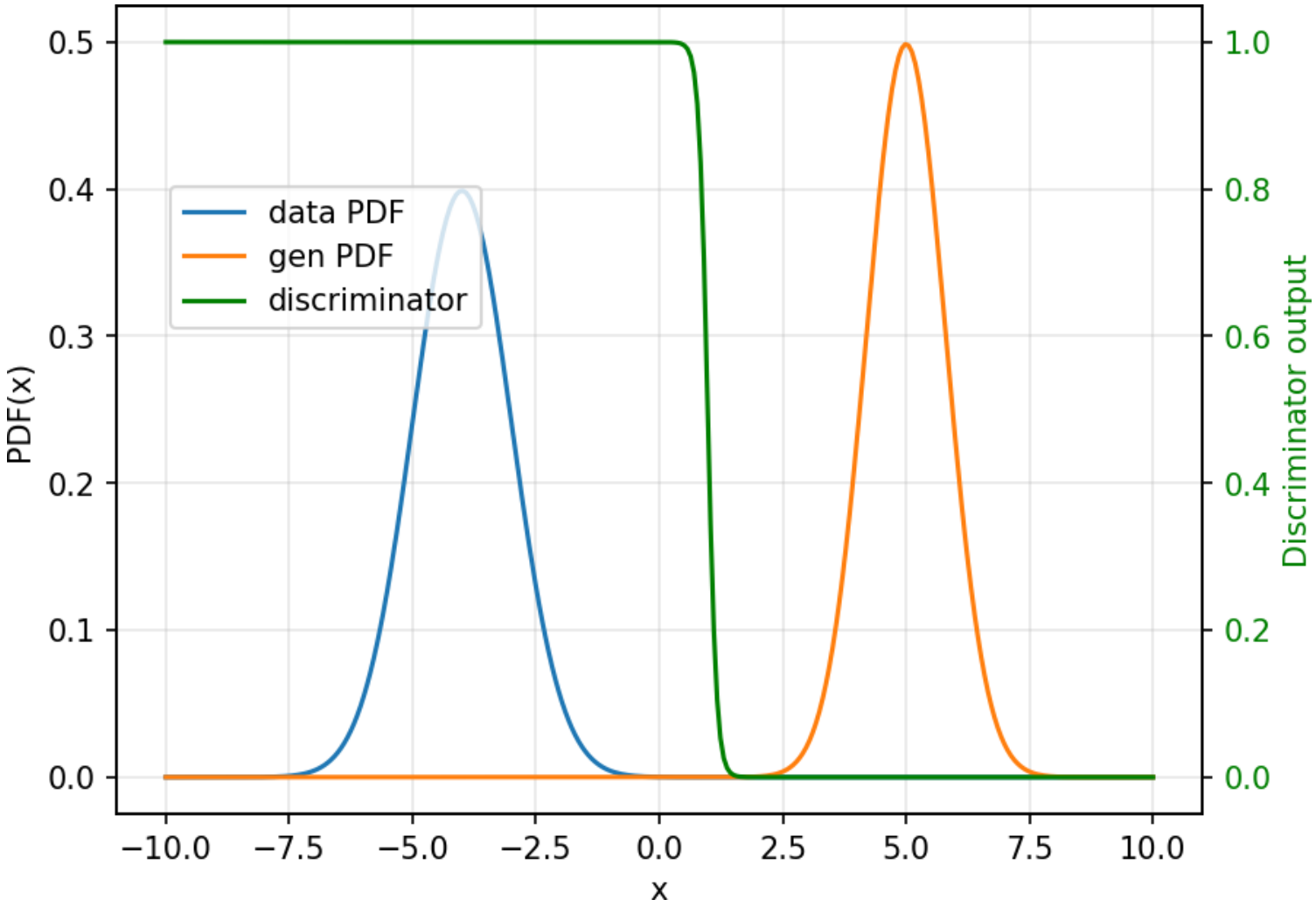
- When the discriminator is perfect, we are guaranteed with $D(y) = 1$, for true inputs, and $D(y) = 0$, for generated ones. Therefore the loss function L falls to zero and we end up with no gradient to update the loss during learning iterations.

As a result, training a GAN faces a dilemma:

- If the discriminator behaves badly, the generator does not have accurate feedback and the loss function cannot represent the reality.
- If the discriminator does a great job, the gradient of the loss function drops down to close to zero and the learning becomes super slow or even jammed.

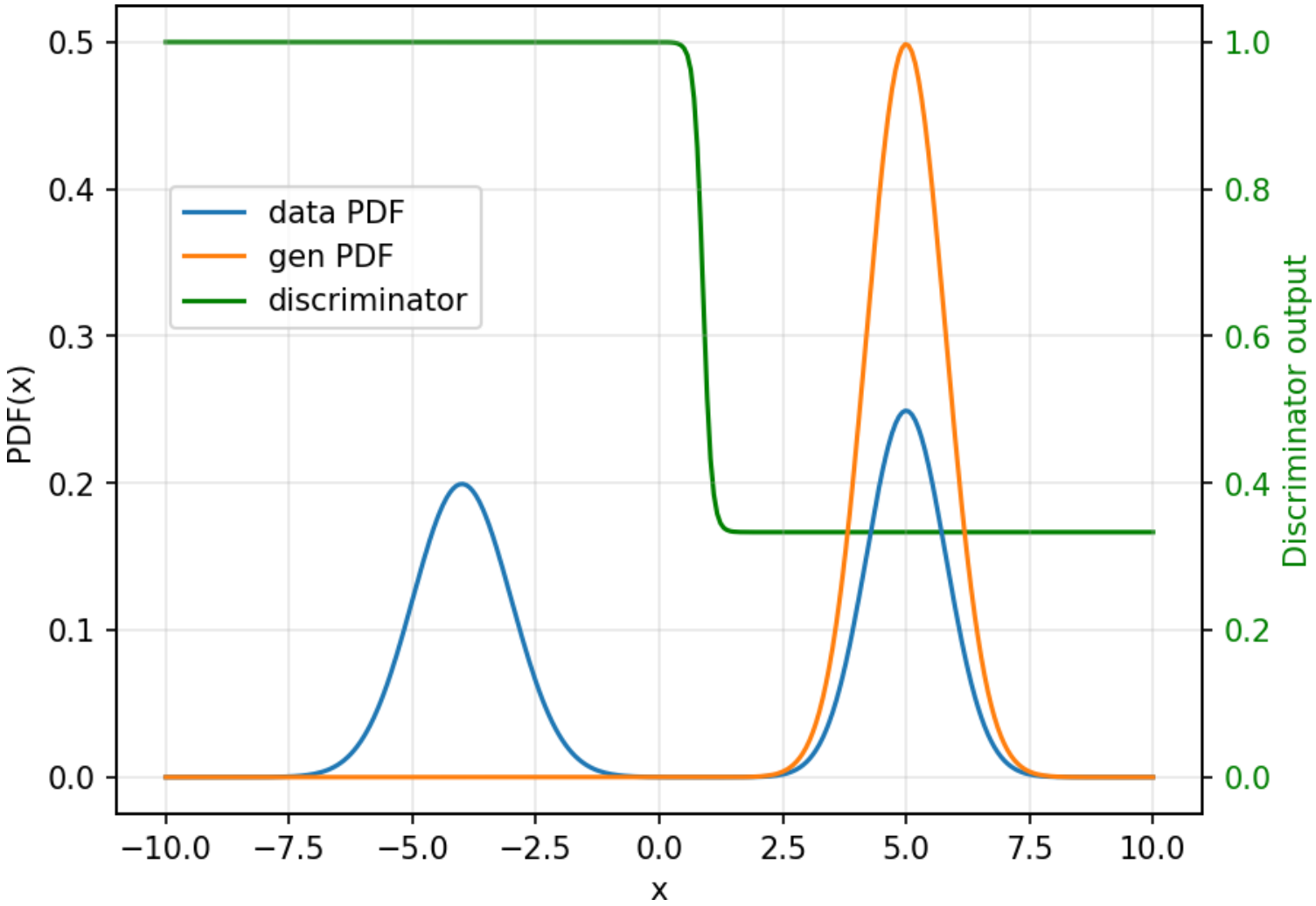


Non-overlapping support



$$D_\psi(y) = \frac{p_{\text{data}}(y)}{p_{\text{data}}(y) + p_\theta(y)}$$

Partially overlapping support



$$D_\psi(y) = \frac{p_{\text{data}}(y)}{p_{\text{data}}(y) + p_\theta(y)}$$

Non overlapping support and KL and JSD

- With the "optimal" discriminator, we h as:

$$\frac{\partial}{\partial\theta}\mathrm{JSD}(p_{\text{data}}\|p_\theta) = 0$$

if p_{data} and p_θ do not overlap

$$\mathcal{L} = \mathbb{E}_{y\sim p_{\text{data}}}\left[\log\frac{p_{\text{data}}(y)}{p_{\text{data}}(y)+p_\theta(y)}\right] + \mathbb{E}_{y'\sim p_\theta}\left[\log\frac{p_\theta(y)}{p_{\text{data}}(y)+p_\theta(y)}\right]$$

$$= -\log 4 + 2\cdot\mathrm{JSD}(p_{\text{data}}\|p_\theta) \rightarrow \min_\theta$$

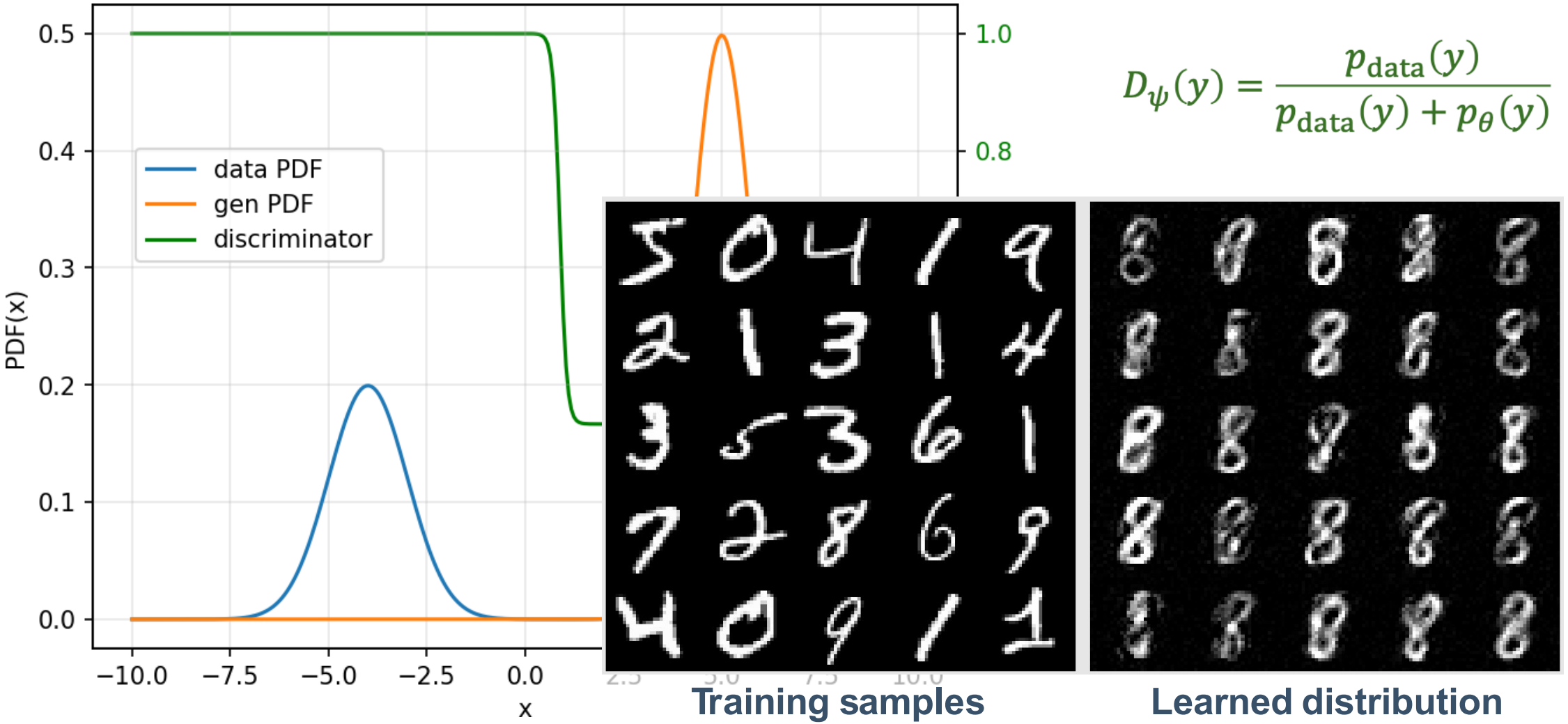
Jensen–Shannon divergence

$$\mathrm{JSD}(p\|q) \equiv \frac{1}{2}\left[D_{\mathrm{KL}}\left(p\middle\|\frac{p+q}{2}\right) + D_{\mathrm{KL}}\left(q\middle\|\frac{p+q}{2}\right)\right]$$

Kullback–Leibler divergence

$$D_{\mathrm{KL}}(p\|q) = \mathbb{E}_{X\sim p}\left[\log\frac{p(X)}{q(X)}\right]$$

Partially overlapping support

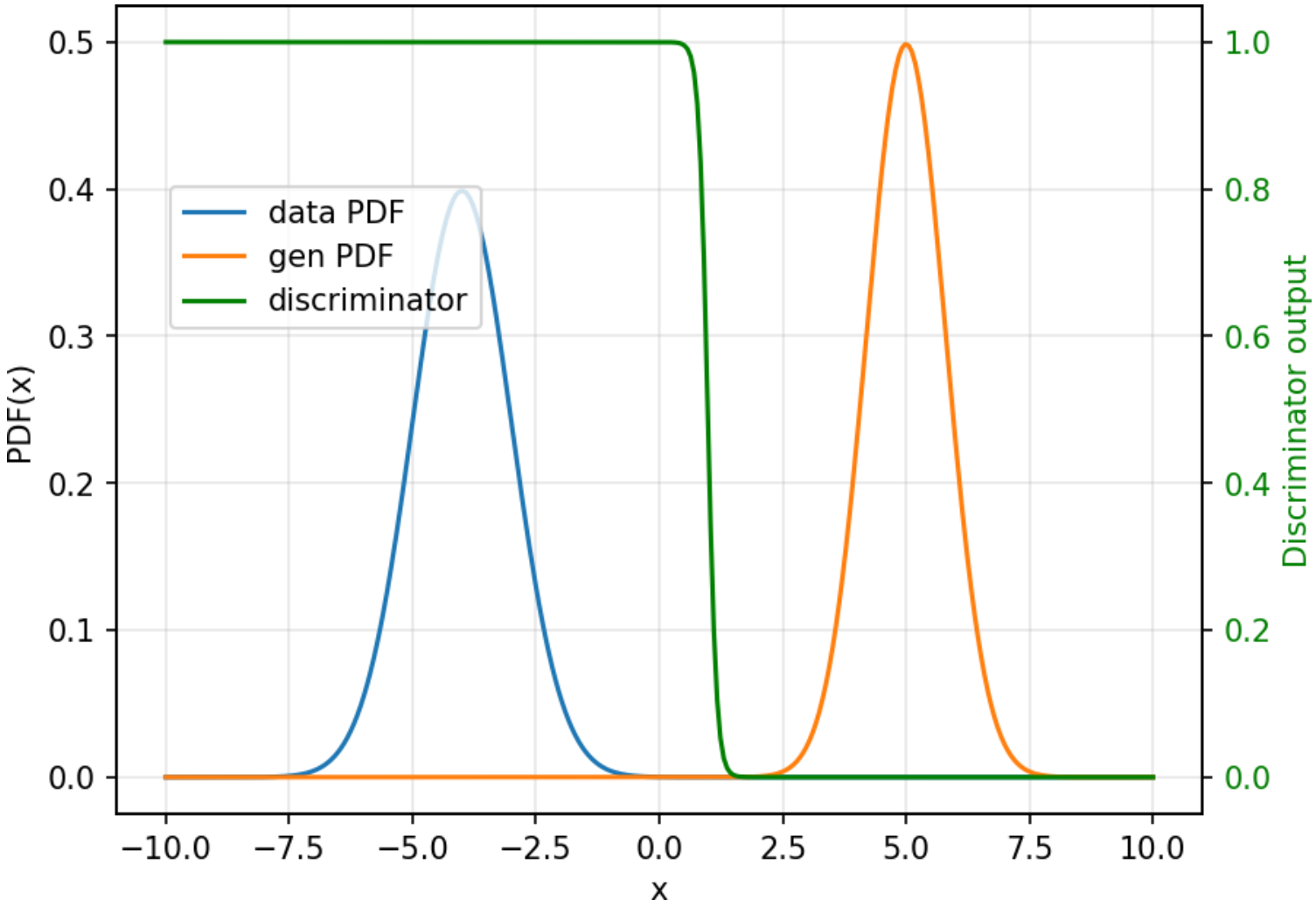


$$D_\psi(y) = \frac{p_{\text{data}}(y)}{p_{\text{data}}(y) + p_\theta(y)}$$

Training samples

Learned distribution

Non-overlapping support



$$D_\psi(y) = \frac{p_{\text{data}}(y)}{p_{\text{data}}(y) + p_\theta(y)}$$

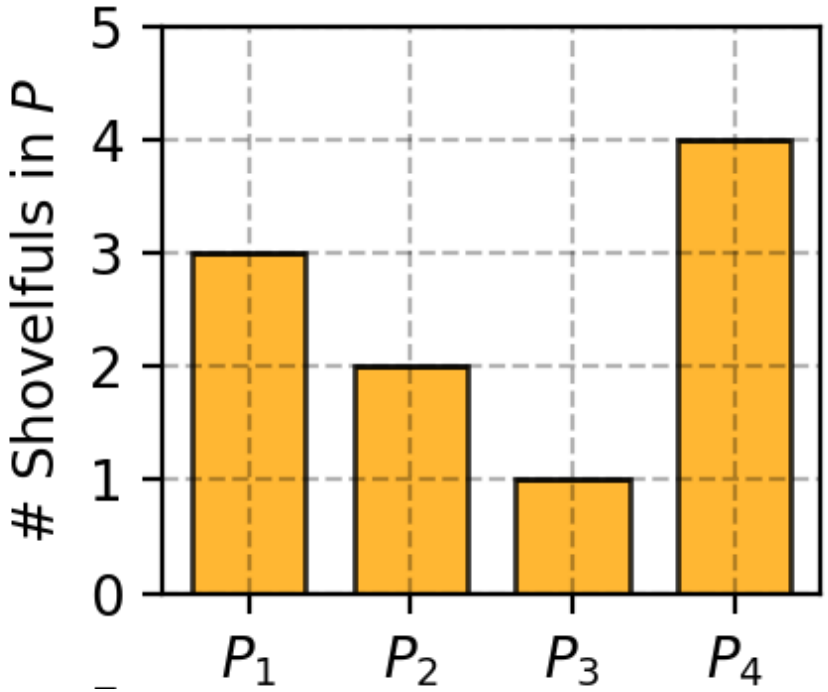
Generative Adversarial Networks

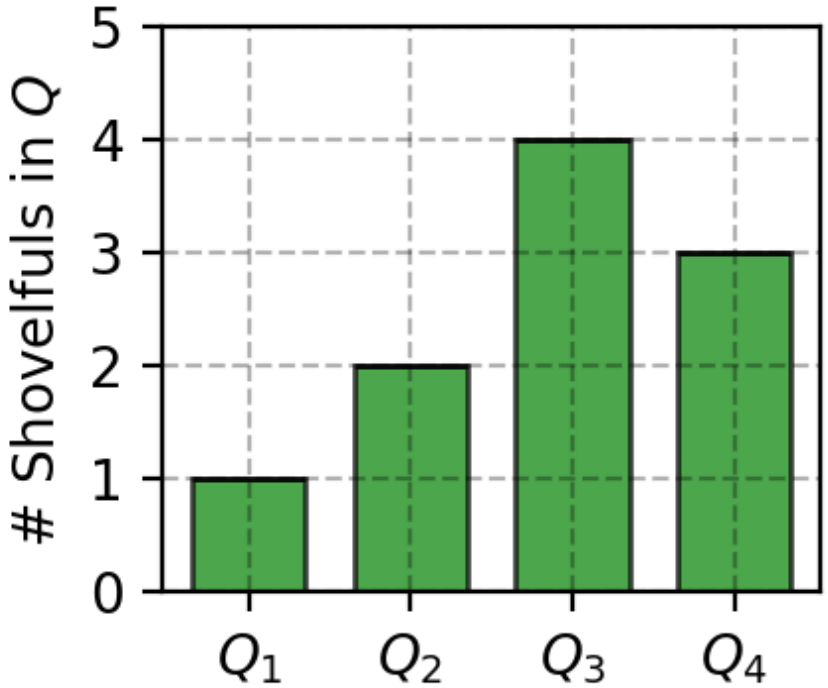
Wassertsein Distance

- Wasserstein Distance is a measure of the distance between two probability distributions. It is also called **Earth Mover's distance**, short for EM distance, because informally it can be interpreted as the minimum energy cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of the other distribution.

Moving dirt

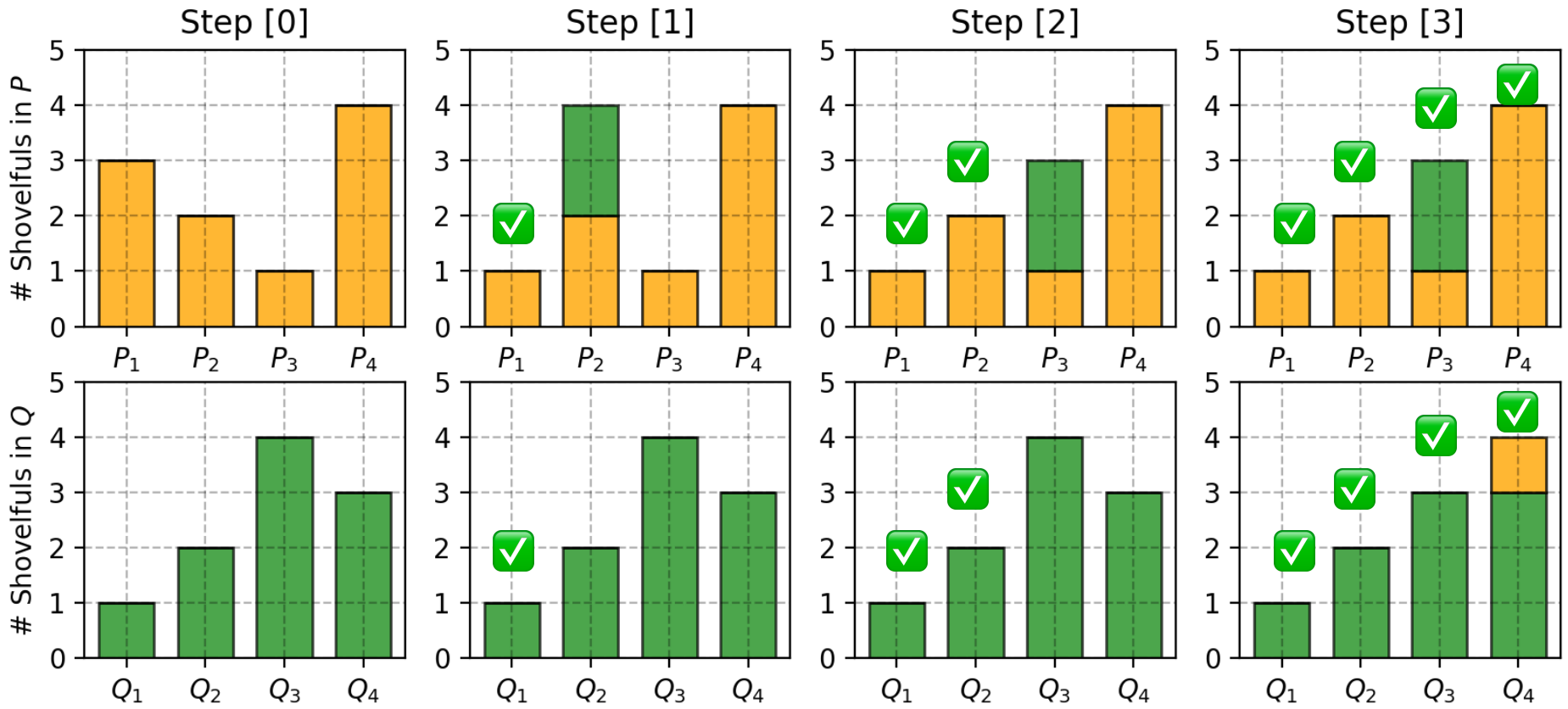
- A simple case where the probability domain is *discrete*. Suppose we have two distributions P and Q. Both:
  - four piles of dirt;
  - ten shovelfuls of dirt in total.





Moving dirt

- Let's move dirt:



Moving dirt

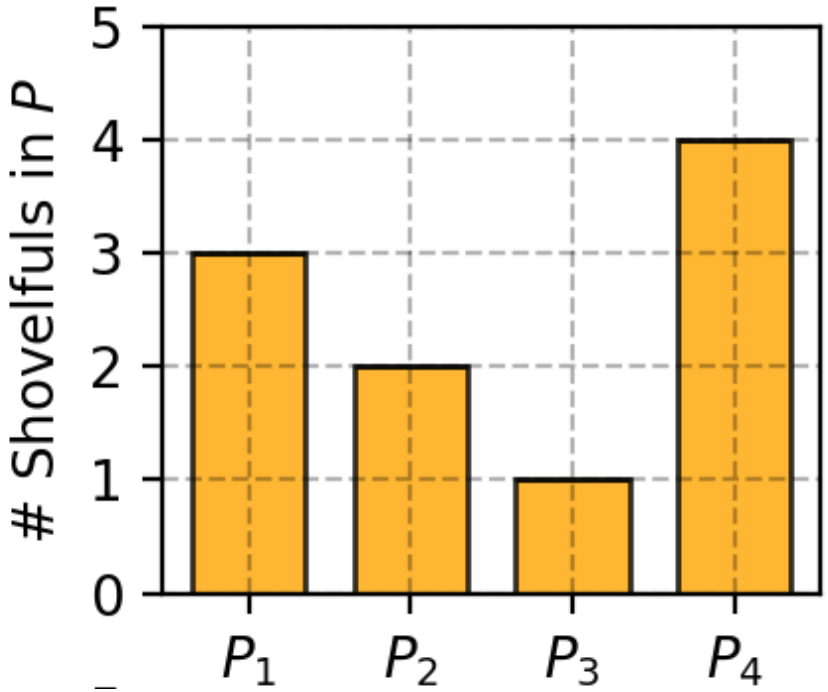
- If we label the cost to pay to make P_i and Q_i match as δ_i, we would have

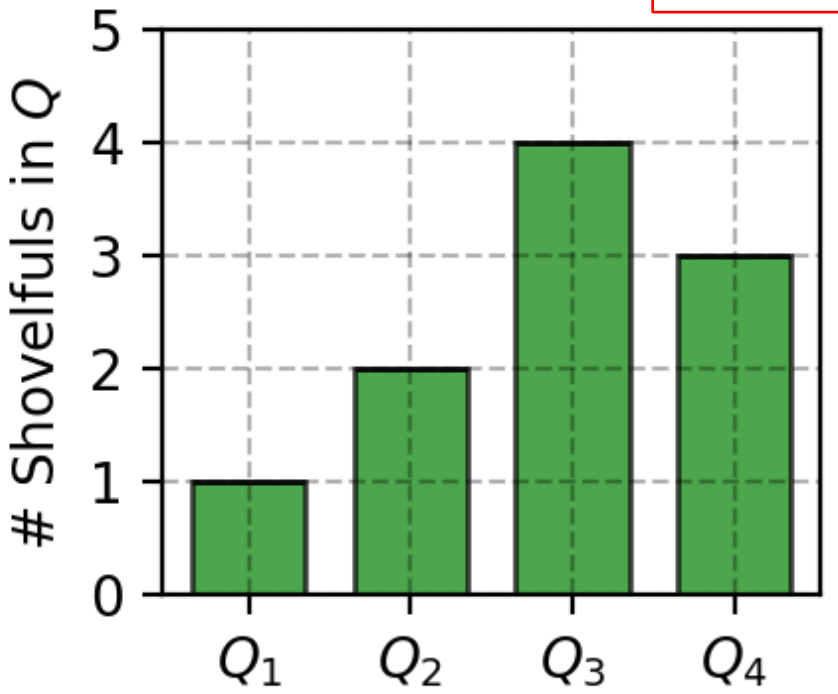
$$\delta_{i+1} = \delta_i + P_i - Q_i$$

- and in the example:

$$\delta_1 = P_1 - Q_1 = 3 - 1 = 2 \quad ; \quad \delta_2 = P_2 - Q_2 = 2 + 2 - 2 = 2$$
$$\delta_3 = P_3 - Q_3 = 2 + 1 - 4 = -1 \quad ; \quad \delta_4 = P_4 - Q_4 = -1 + 4 = 3 = 0$$

0 because it's «free», fully depend on the other movements





Moving dirt

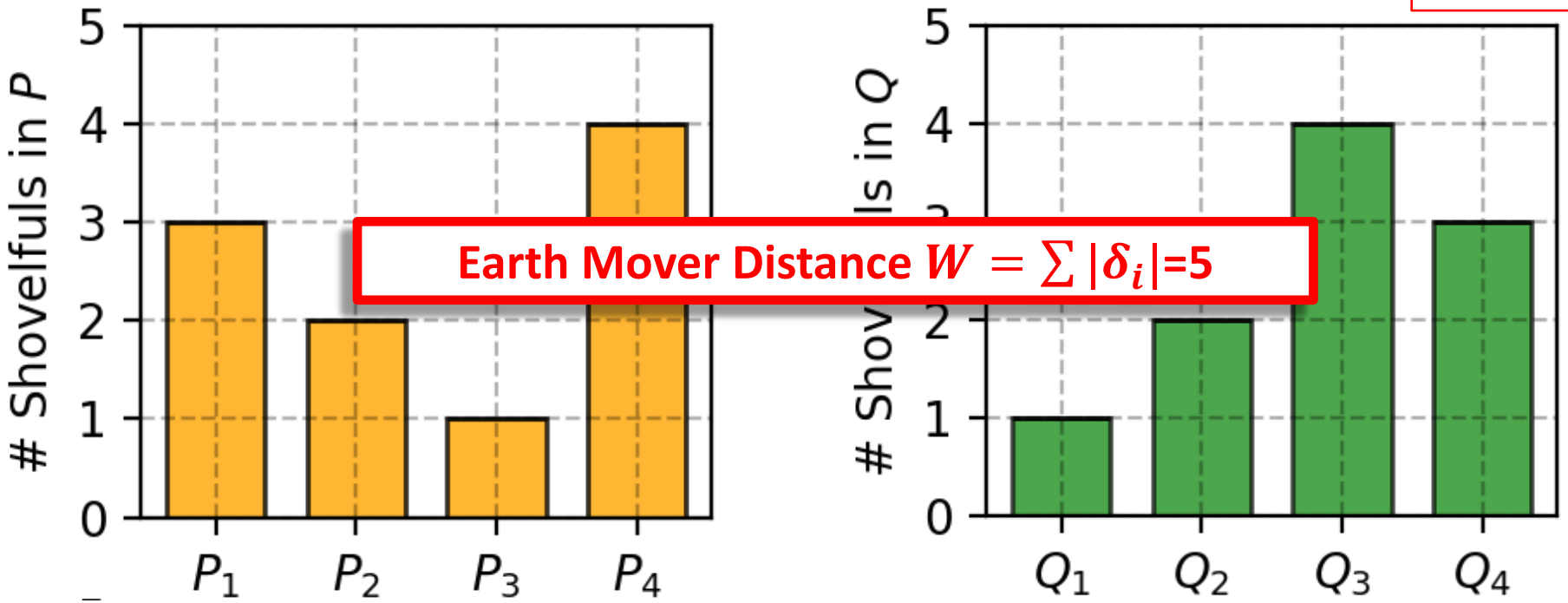
- If we label the cost to pay to make P_i and Q_i match as δ_i, we would have

$$\delta_{i+1} = \delta_i + P_i - Q_i$$

- and in the example:

$$\delta_1 = P_1 - Q_1 = 3 - 1 = 2 \quad ; \quad \delta_2 = P_2 - Q_2 = 2 + 2 - 2 = 2$$
$$\delta_3 = P_3 - Q_3 = 2 + 1 - 4 = -1 \quad ; \quad \delta_4 = P_4 - Q_4 = -1 + 4 = 3 = 0$$

0 because it's «free», fully depend on the other movements



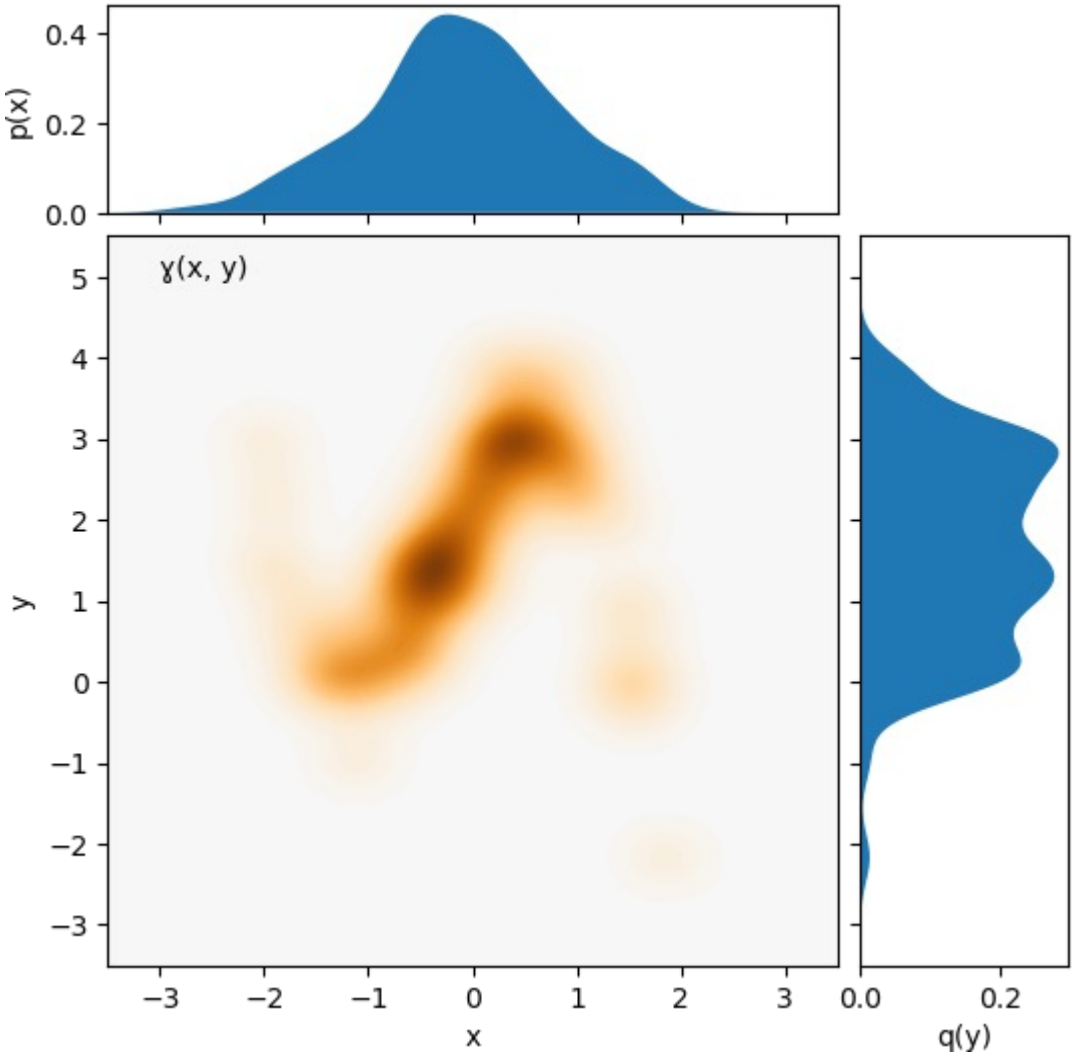
Earth Mover Distance $W = \sum |\delta_i|$=5

Continous p.d.f.s

- Earth Mover's distance or Wasserstein-1 distance:

$$W(p,q) := \\ := \inf_{\gamma \in \prod(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]$$

  - γ is the transportation «plan»

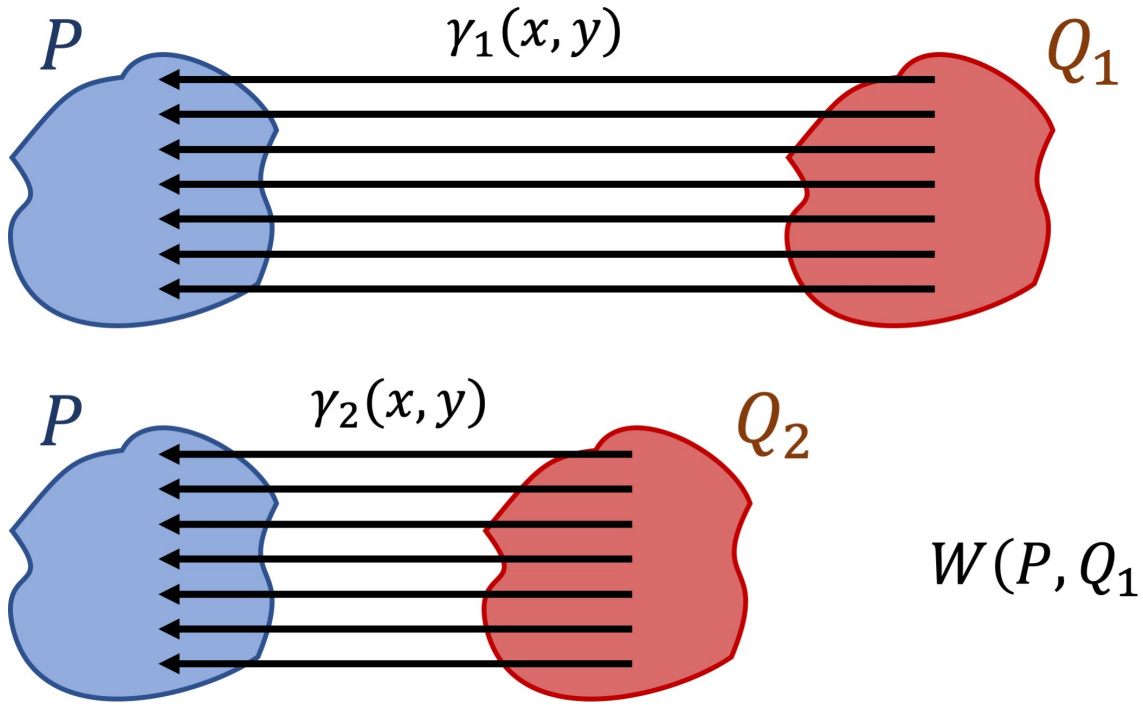


Optimal transport

- Set of all valid plans:

$$\Pi(p,q) \equiv \left\{ \gamma(x,y) \quad : \quad \begin{array}{c} \int_x \gamma(x,y)dx = q(y) \\ \int_x \gamma(x,y)dy = p(x) \\ \int_{x\times x} \gamma(x,y)dxdy = 1 \\ \gamma(x,y) \geq 0 \end{array} \right\}$$

Non-overlapping supports

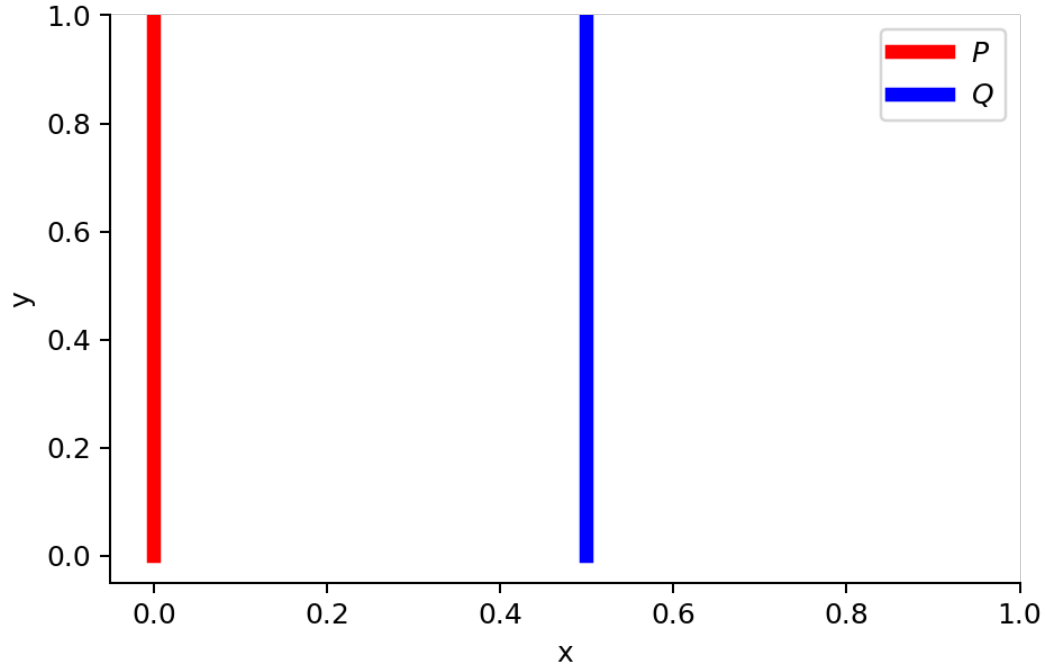


$$W(P,Q_1) > W(P,Q_2)$$

Why EM/W is better than JDS and KL

- Two dummy p.d.f.s:

  - $P: x = 0 \rightarrow y \sim U([0,1])$
  - Q: $x = \theta \rightarrow y \sim U([0,1])$ with $\theta \in [0,1]$
  - If $\theta \neq 0$ they have disjoint supports



$\theta \neq 0$

$$D_{KL}(P\|Q) = \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{KL}(Q\|P) = \sum_{x=\theta, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

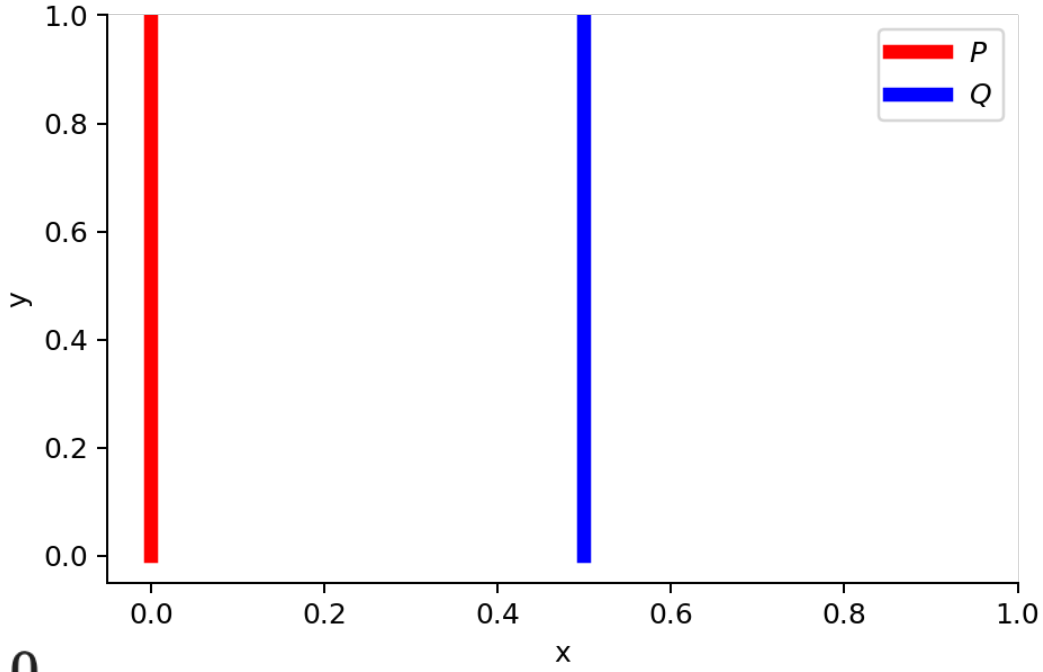
$$D_{JS}(P,Q) = \frac{1}{2}\left(\sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2}\right) = \log 2$$

$$W(P,Q) = |\theta|$$

Cost of moving x from θ to 0

Why EM/W is better than JDS and KL

- Two dummy p.d.f.s:
  - $P: x = 0 \rightarrow y \sim U([0,1])$
  - Q: $x = \theta \rightarrow y \sim U([0,1])$ with $\theta \in [0,1]$
  - If $\theta \neq 0$ they have disjoint supports



$\theta = 0$

Jumps!

$$D_{KL}(P\|Q) = D_{KL}(Q\|P) = D_{JS}(P,Q) = 0$$

$$W(P,Q) = 0 = |\theta|$$

Smooth

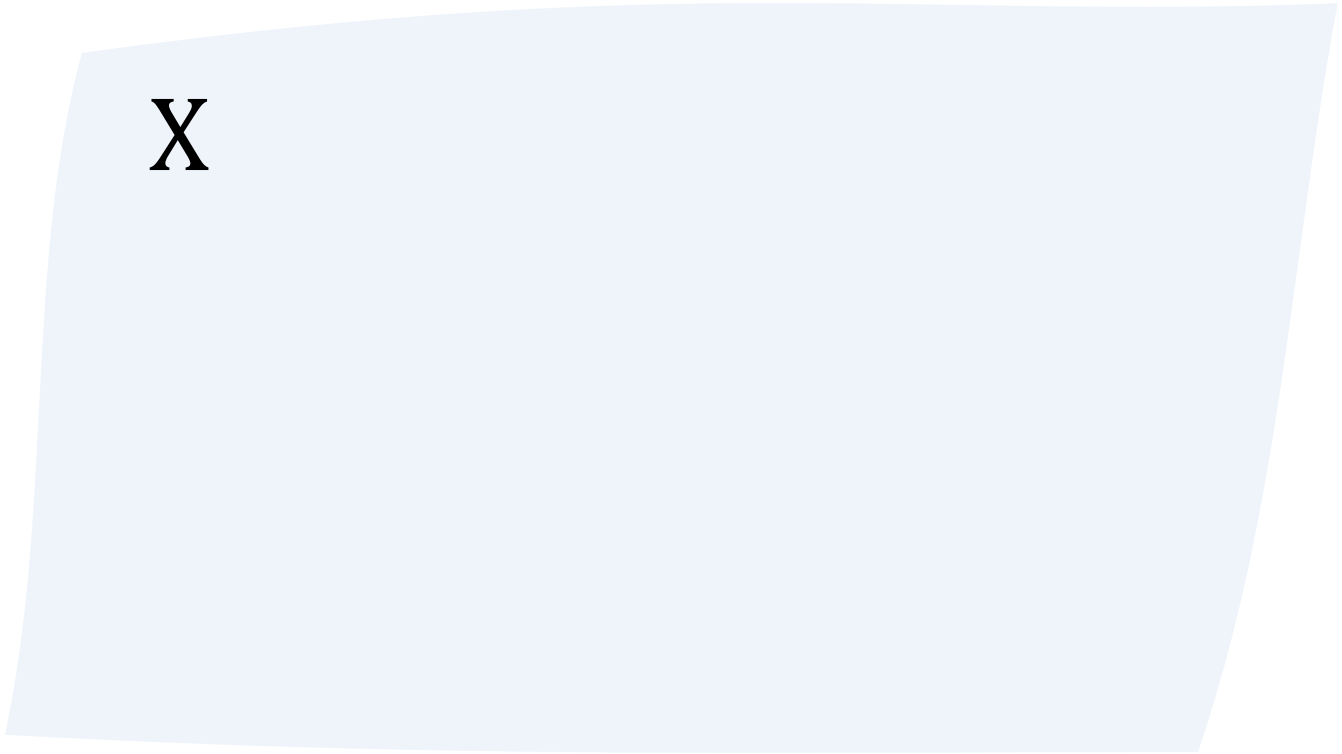
Problem

- This is intractable

$$W(p,q) := \inf_{\gamma \in \prod(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]$$

Impossible to calculate all the possible paths.

Optimal transport



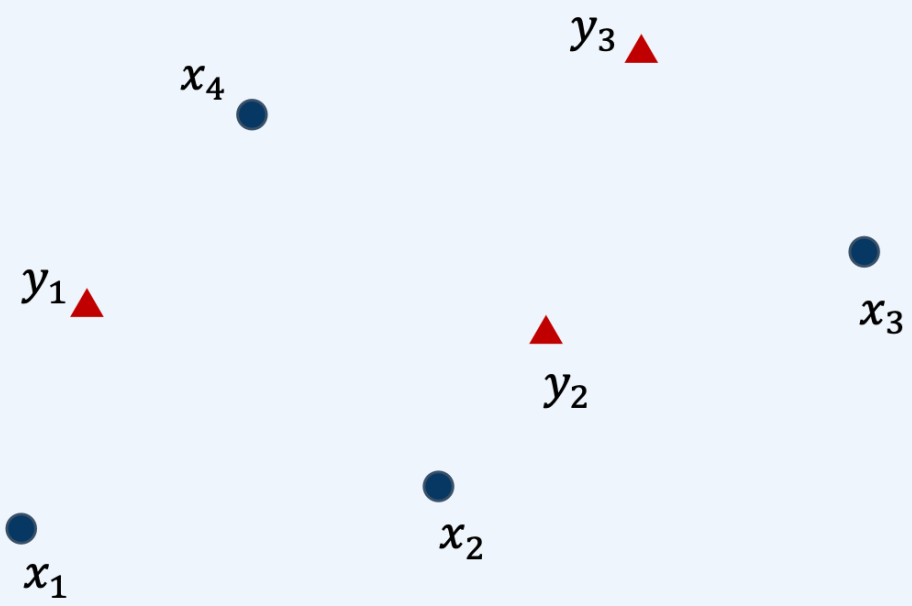
Optimal transport

$p(x)$ – amount of pastry produced at x

$q(y)$ – amount of pastry consumed at y

● bakery

▲ café



"Cozy patisserie cafe, happy customers, silly 2d cartoon, white background" – as interpreted by Midjourney

"Cozy bakery, silly 2d cartoon, white background" – as interpreted by Midjourney

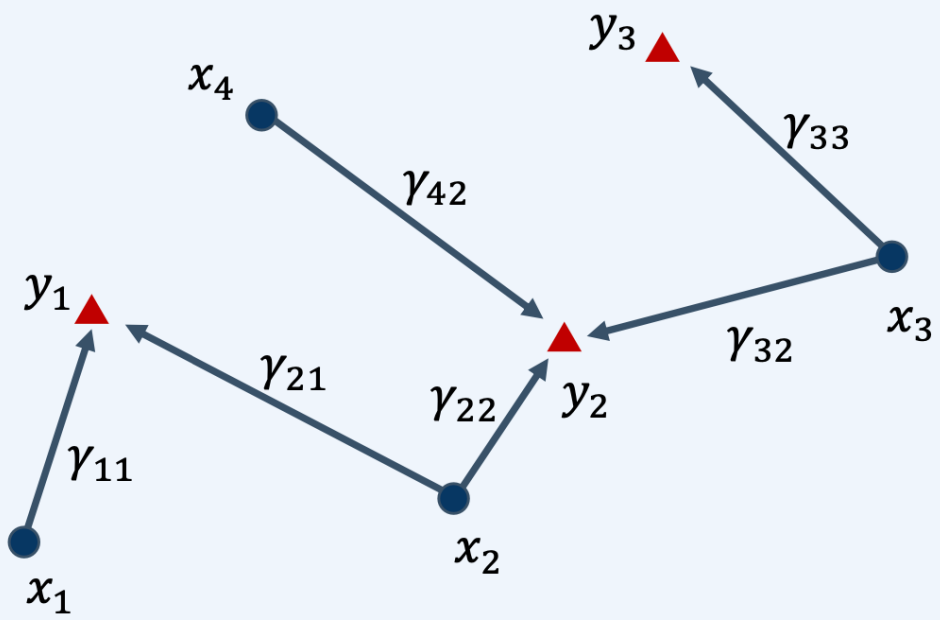
Optimal transport

$p(x)$ – amount of pastry produced at x

$q(y)$ – amount of pastry consumed at y

▲ café

X



● bakery

The cost of the transport is the amount of pastry transported from x to y

"Cozy patisserie cafe, happy customers, silly 2d cartoon, white background" – as interpreted by Midjourney

"Cozy bakery, silly 2d cartoon, white background" – as interpreted by Midjourney

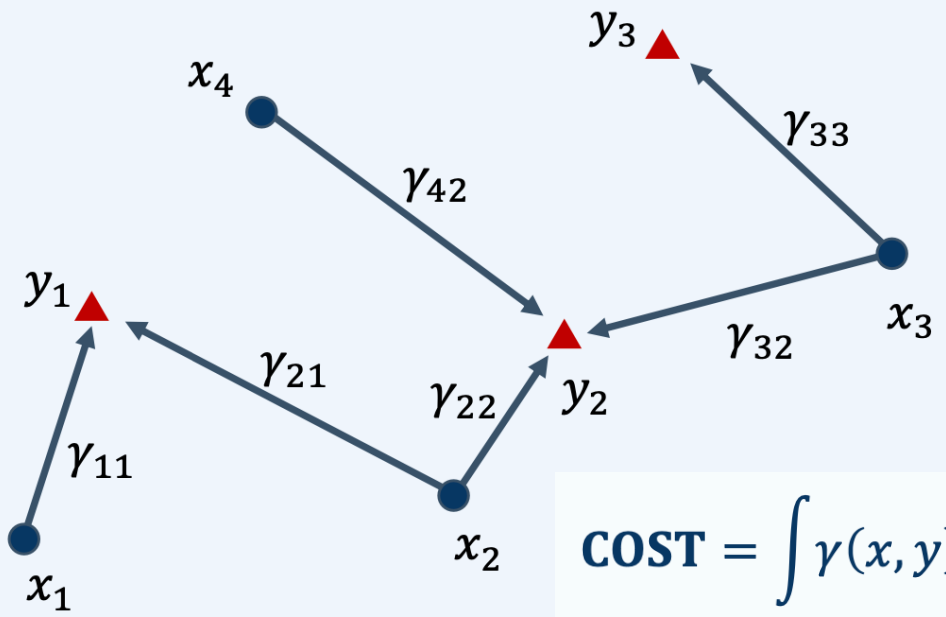
Optimal transport

$p(x)$ – amount of pastry produced at x

$q(y)$ – amount of pastry consumed at y

▲ café

X



● bakery

$\gamma(x,,y)$ – amount of pastry transported from x to y

$$\text{COST} = \int \gamma(x,y)\|x-y\|dxdy = \mathbb{E}_\gamma\|x-y\| \to \min_\gamma$$

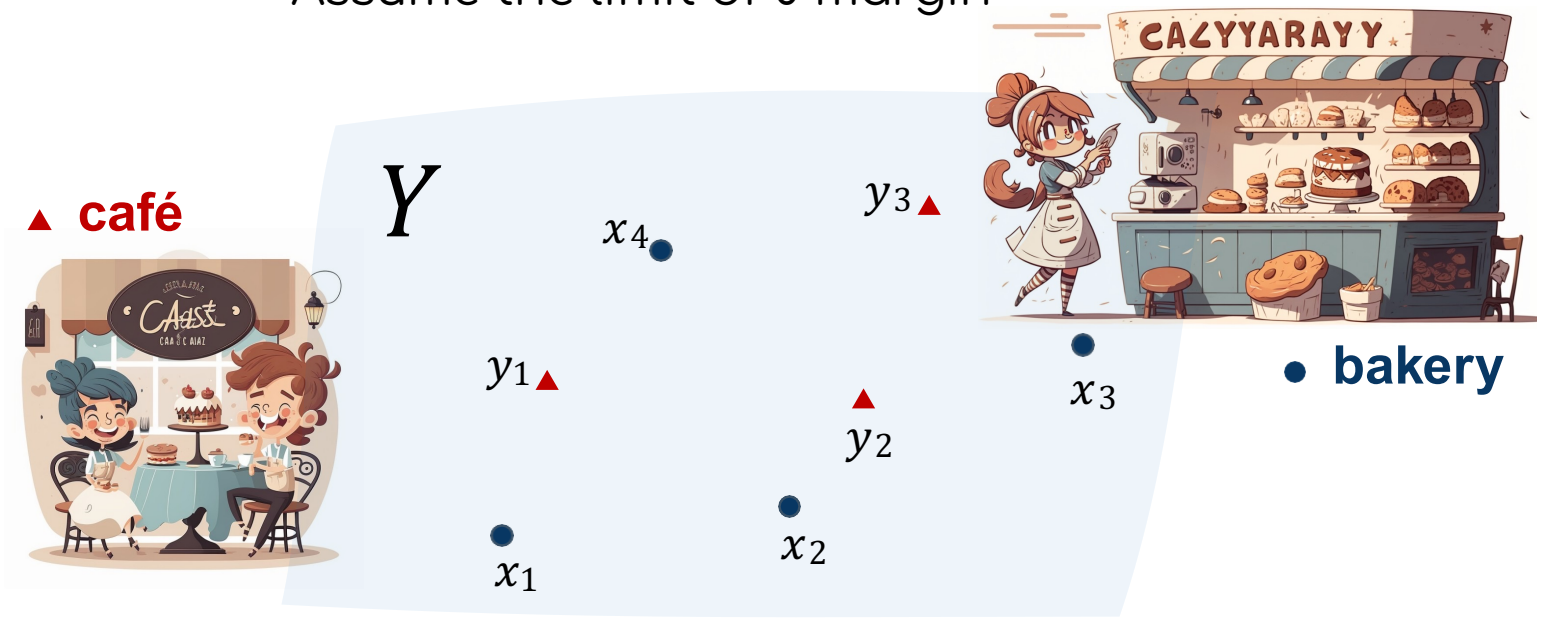
"Cozy patisserie cafe, happy customers, silly 2d cartoon, white background" – as interpreted by Midjourney

"Cozy bakery, silly 2d cartoon, white background" – as interpreted by Midjourney

Dual formulation

- Let's outsource the delivery
  - 3rd party company will buy the bread at bakeries for price $p(x)$ and sell to cafés for price $g(y)$
  - Assume the limit of 0 margin

"Shady trader in a van, silly 2d cartoon, white background" – as interpreted by Midjourney



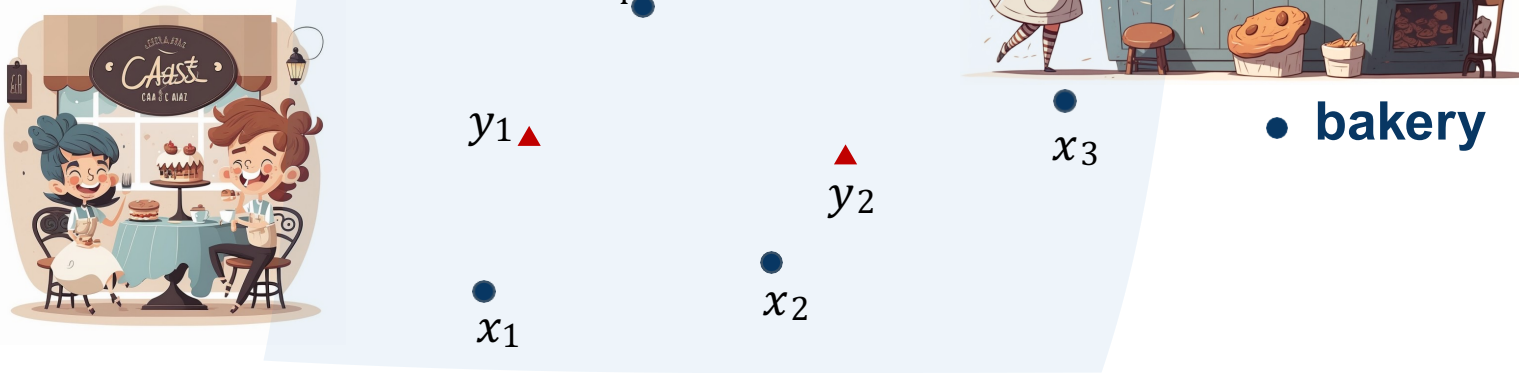
"Cozy patisserie cafe, happy customers, silly 2d cartoon, white background" – as interpreted by Midjourney

"Cozy bakery, silly 2d cartoon, white background" – as interpreted by Midjourney

Dual formulation

- Let's outsource the delivery
  - 3rd party company will buy the bread at bakeries for price $p(x)$ and sell to cafés for price $g(y)$
  - Assume the limit of 0 margin

"Shady trader in a van, silly 2d cartoon, white background" – as interpreted by Midjourney



"Cozy patisserie cafe, happy customers, silly 2d cartoon, white background" – as interpreted by Midjourney

"Cozy bakery, silly 2d cartoon, white background" – as interpreted by Midjourney

- The company's revenue is:

$$\mathop{\mathbb{E}}_{y\sim q} g(y) - \mathop{\mathbb{E}}_{x\sim p} f(x)$$

- To be competitive, the company's prices should satisfy:

$$g(y) - f(x) \le \|x - y\|$$

$$\Rightarrow g(x) \equiv f(x)$$

Dual formulation

- Functions satisfying $f(x) - f(y) \leq C\,\|x - y\|$ are called Lipschitz-continuous with Lipschitz constant $C = 1$. Let's denote them as $\|f_L\| \leq 1$.

- The company seeks to maximize the revenue:

$$\max_{\|f_L\| \leq 1} \mathbb{E}_{y \sim q}[f(y)] - \mathbb{E}_{x \sim q}[f(x)]$$

https://en.wikipedia.org/wiki/Lipschitz_continuity

Dual formulation

$$W(p,q) := \inf_{\gamma \in \prod(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]$$

- Let's reformulate

$$\min_{\gamma \sim \prod(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||] \geq \mathbb{E}_{(x,y)\sim\gamma}[f(y)-f(x)]$$
$$= \mathbb{E}_{(x,y)\sim\gamma}[f(y)] - \mathbb{E}_{(x,y)\sim\gamma}[f(x)]$$
$$= \mathbb{E}_{y\sim q}[f(y)] - \mathbb{E}_{x\sim p}[f(x)] \qquad \max_{||f_L||\leq 1}$$

$$W(p,q) = \inf_{\gamma\in\Pi(p,q)} \left[\mathbb{E}_{\gamma}\|x-y\|\right] = \sup_{\|f\|_L\leq 1} \left[\mathop{\mathbb{E}}_{y\sim q} f(y) - \mathop{\mathbb{E}}_{x\sim p} f(x)\right]$$

What is changing for our (W)GAN?

$$\mathcal{L} = \mathbb{E}_{y \sim p_{\text{data}}}[\log D_\psi(y)] + \mathbb{E}_{y' \sim p_\theta}\left[\log\left(1 - D_\psi(y')\right)\right]$$

$$\mathcal{L} = W = \sup_{\|f_L\| \leq 1} [\mathbb{E}_{y \sim p_{data}}[f(y)] - \mathbb{E}_{y' \sim \boldsymbol{p_\theta}}[f(y')]]$$

- Thus the discriminator is not a direct critic of telling the fake samples apart from the real ones anymore. Instead, it is trained to learn a Lipschitz continuous function to help compute W.
- As the loss function decreases in the training, the Wasserstein distance gets smaller and the generator model's output grows closer to the real data distribution.

Algorithm (WGAN)

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$ Useful to keep the Lipschitz continuity
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

https://arxiv.org/abs/1701.07875

Algorithm (WGAN)

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$ Useful to keep the Lipschitz continuity
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

https://arxiv.org/abs/1701.07875

Algorithm (WGAN-GP)

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, \ldots, n_{\text{critic}}$ **do**
    for $i = 1, \ldots, m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

https://arxiv.org/abs/1704.00028

WGAN-GP

- Allowed for training of very deep GANs

Figure 2: Different GAN architectures trained with different methods. We only succeeded in training every architecture with a shared set of hyperparameters using WGAN-GP.

References

1. Goodfellow, Ian, et al. "Generative adversarial nets." NIPS, 2014.
2. Tim Salimans, et al. "Improved techniques for training gans." NIPS 2016.
3. Martin Arjovsky and Léon Bottou. "Towards principled methods for training generative adversarial networks." arXiv preprint arXiv:1701.04862 (2017).
4. Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein GAN." arXiv preprint arXiv:1701.07875 (2017).
5. Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville. Improved training of wasserstein gans. arXiv preprint arXiv:1704.00028 (2017).
6. Computing the Earth Mover's Distance under Transformations
7. Wasserstein GAN and the Kantorovich-Rubinstein Duality
8. zhuanlan.zhihu.com/p/25071913
9. Ferenc Huszár. "How (not) to Train your Generative Model: Scheduled Sampling, Likelihood, Adversary?." arXiv preprint arXiv:1511.05101 (2015).
10. Lilian Weng From GAN to WGAN, https://lilianweng.github.io/posts/2017-08-20-gan/