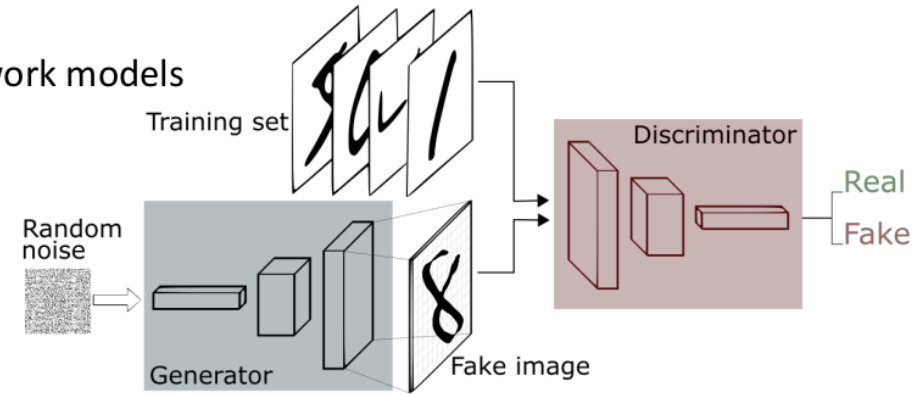
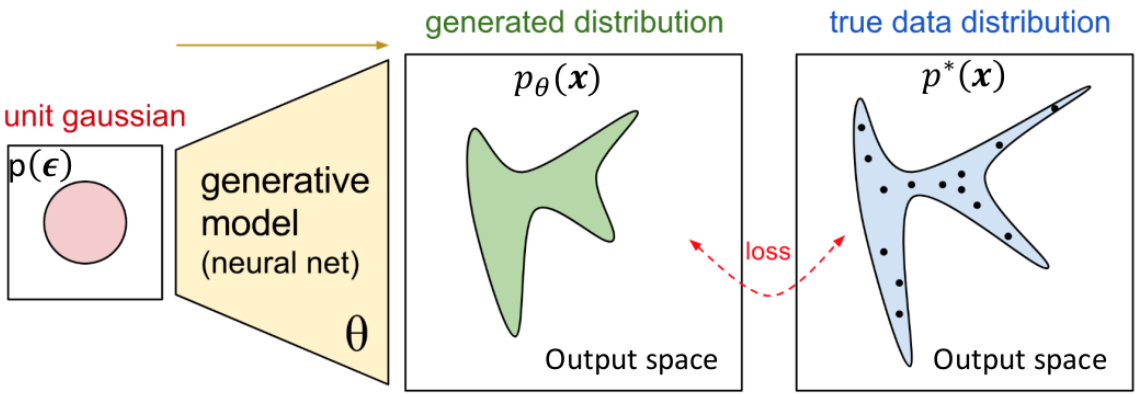
- GANs have become very popular for learning deep generative models
- Informally, the main idea is:
  - Two competing neural network models
  - Generator: takes noise as input and generates ("fake") samples
  - Discriminator: receives samples from both generator and training data and has to distinguish between the two → classify input as "real" or "fake"
  - Goal: Train the generator in such a way that the discriminator can **not** distinguish between real and "fake" samples
    - ➢ In this case, the generator generates realistic examples

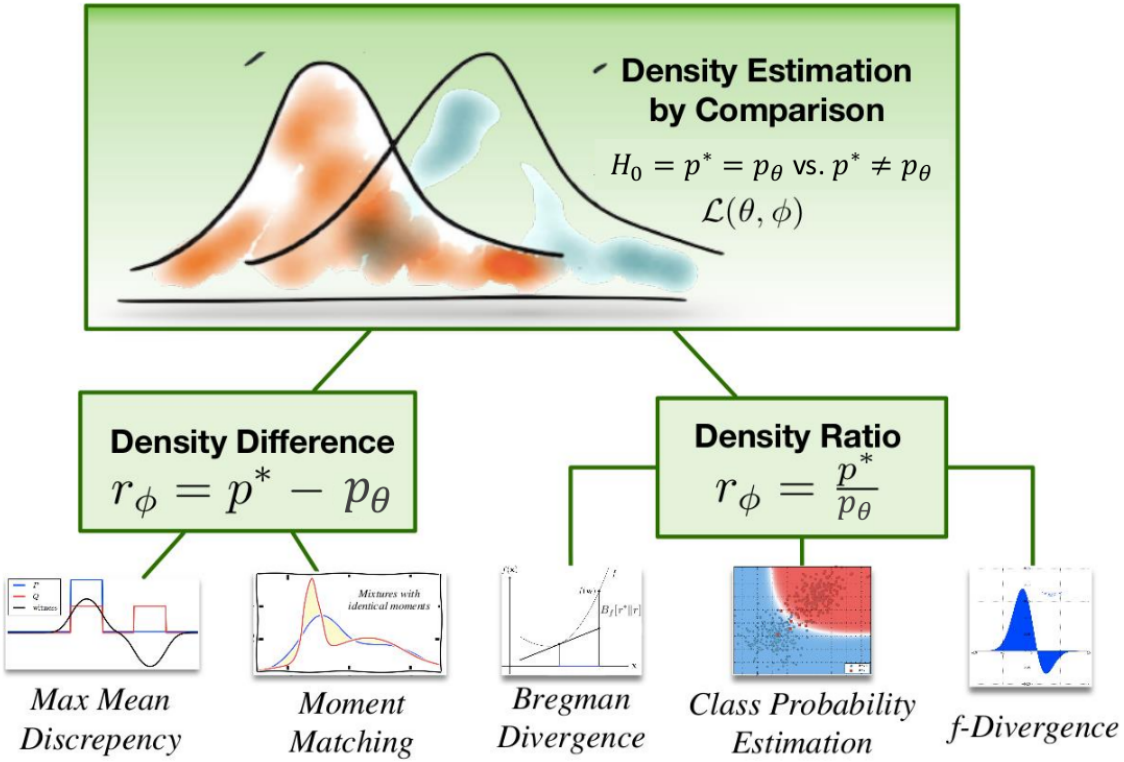


From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

- What can we do if we can **easily draw samples** from the model but we cannot evaluate the density?
- Idea: We can use **any** method that compares two **sets of samples**.
  - This process is called ***density estimation by comparison.***
- We **test** the **hypothesis** that the true data distribution $p^*(\boldsymbol{x})$ and the model distribution $p_\theta(\boldsymbol{x})$ are **equal**



From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.



From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

- Density ratio $r^*(\boldsymbol{x}) = p^*(\boldsymbol{x}) / p_\theta(\boldsymbol{x})$
  - In the best case always 1, i.e. the two distributions are indistinguishable
  - However, we cannot compute ratio in closed form/easily
- Idea: Approximate the true density ratio $r^*(\boldsymbol{x})$ by $r_\phi(\boldsymbol{x})$
  - Finding the approximation $r_\phi(\boldsymbol{x})$ often means solving again a learning problem

- Thus, we get the following general principle for learning
  - **Optimize Ratio loss**:
    approximate the true density ratio $r^*(\boldsymbol{x})$ (i.e. learning ϕ)
  - **Optimize Generative loss**:
    drive the density ratio towards 1 (i.e. learning θ)
  - Essentially a bi-level optimization problem, which is usually just solved alternatingly

From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

- Density ratio $r^*(\boldsymbol{x}) = p^*(\boldsymbol{x}) / p_\theta(\boldsymbol{x})$
  - In the best case always 1, i.e. the two distributions are indistinguishable
  - However, we cannot compute ratio in closed form/easily
- Idea: Approximate the true density ratio $r^*(\boldsymbol{x})$ by $r_\phi(\boldsymbol{x})$
  - Finding the approximation $r_\phi(\boldsymbol{x})$ often means solving again a learning problem

- Thus, we get the following general principle for learning
  - **Optimize Ratio loss**:
    approximate the true density ratio $r^*(\boldsymbol{x})$ (i.e. learning ϕ)
  - **Optimize Generative loss**:
    drive the density ratio towards 1 (i.e. learning θ)

  - Essentially a bi-level optimization problem, which is usually just solved alternatingly

From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019

- Let Y denote a random variable which assigns label $Y = 1$ to samples from the true data distribution; $Y = 0$ to those from the generator distribution
- Then $p^*(\boldsymbol{x}) = p(\boldsymbol{x} \mid Y = 1)$ and $p_\theta(\boldsymbol{x}) = p(\boldsymbol{x} \mid Y = 0)$

- Denote $P(Y = 1) = \pi$. From Bayes we have:

$$r^*(\boldsymbol{x}) = \frac{p^*(\boldsymbol{x})}{p_\theta(\boldsymbol{x})} = \frac{p(Y = 1 \mid \boldsymbol{x})}{p(Y = 0 \mid \boldsymbol{x})} \frac{1 - \pi}{\pi}$$

  - Apparently density ratio estimation is equal to **class-probability estimation**
  - Simply speaking: we can consider a classifier for $\boldsymbol{x}$ (predicting labels Y=0 or 1)

➢ Specify a scoring function or a **discriminator** $D_\phi(\boldsymbol{x}) = p(Y = 1 \mid \boldsymbol{x})$

  - e.g. logistic regression or a neural network

From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

- Specify a scoring function or a **discriminator** $D_\phi(\boldsymbol{x}) = p(Y = 1|\boldsymbol{x})$
  - e.g. logistic regression or a neural network

- For learning $D_\phi(\boldsymbol{x})$, we need a loss function, e.g., the cross-entropy loss:

$$\mathcal{L}_{\theta,\phi} = \mathbb{E}_{p(\boldsymbol{x}|y)p(y)}\left[-y\log[D_\phi(\boldsymbol{x})] - (1-y)\log[1-D_\phi(\boldsymbol{x})]\right]$$

$$= \pi\mathbb{E}_{p^*(\boldsymbol{x})}\left[-\log D_\phi(\boldsymbol{x})\right] + (1-\pi)\mathbb{E}_{p(\boldsymbol{z})}\left[-\log\left[1-D_\phi(f_\theta(\boldsymbol{z}))\right]\right]$$

From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

1. Solving $\phi^*(\theta) = \underset{\phi}{\operatorname{argmin}}\, \mathcal{L}_{\theta,\phi}$

   leads to the "best" discriminator for a given generative model (θ)

- That is, we well approximate $r^*(\boldsymbol{x}) = \frac{p^*(\boldsymbol{x})}{p_\theta(\boldsymbol{x})} = \frac{p(Y=1 \mid \boldsymbol{x})}{p(Y=0 \mid \boldsymbol{x})} \approx \frac{D_{\phi^*(\theta)}(\boldsymbol{x})}{1-D_{\phi^*(\theta)}(\boldsymbol{x})}$

  – here w.l.o.g. we set $\pi = 0.5$

2. We aim to drive the density ratio $r^*(\boldsymbol{x})$ towards 1

   – aim: $p(Y = 1 \mid \boldsymbol{x}) = p(Y = 0 \mid \boldsymbol{x})$

- That is, find generative model (θ) such that (even) the "best" discriminator cannot distinguish the classes

➢ Solving $\theta^* = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}_{\theta,\phi^*(\theta)}$

From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

- Generator and discriminator play a **minimax game**:

$$\min_{\theta}\max_{\phi}\pi\mathbb{E}_{p^*(\boldsymbol{x})}\left[\log D_{\phi}(\boldsymbol{x})\right]+(1-\pi)\mathbb{E}_{p(\boldsymbol{z})}\left[\log\left[1-D_{\phi}(f_{\theta}(\boldsymbol{z}))\right]\right]$$

  - Discriminator: aims to distinguish between (samples from) $p^*(\boldsymbol{x})$ and $p_{\theta}(\boldsymbol{x})$
    - Maximization // minimization of cross-entropy
  - Generator: aims to generate samples that are indistinguishable
    - Minimization // maximization of (lowest) cross-entropy

- This bilevel problem is typically tackled via alternating optimization.
- **Ratio loss** (discriminator loss) optimization:

$$\min_{\phi}\pi\mathbb{E}_{p^*(\boldsymbol{x})}\left[-\log D_{\phi}(\boldsymbol{x})\right]+(1-\pi)\mathbb{E}_{p(\boldsymbol{z})}\left[-\log\left[1-D_{\phi}(f_{\theta}(\boldsymbol{z}))\right]\right]$$

- **Generative loss** optimization**:**

$$\min_{\theta}\mathbb{E}_{p(\boldsymbol{z})}\left[\log\left[1-D_{\phi}(f_{\theta}(\boldsymbol{z}))\right]\right]$$

From "Mining Massive Datasets - Graphs Deep Generative Models.", Prof. Dr. Stephan Günnemann et.al. TUM, 2019.

Wasserstein Generative Adversarial Networks

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.

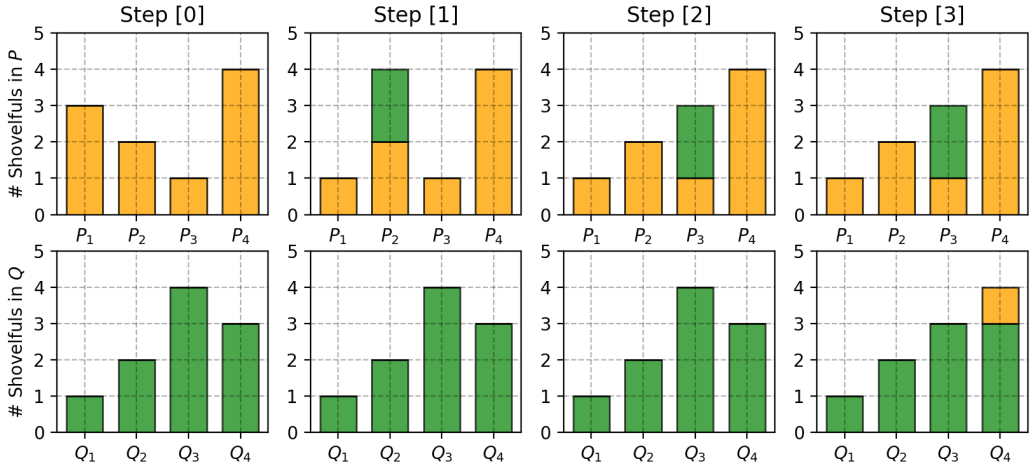
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

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.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  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

Image source: Arjovsky, Chintala, Bottou, 2017.

Wasserstein distance

Wasserstein Distance is a measure of the distance between two probability distributions. It is also called Earth Mover's distance. 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.



From "https://arxiv.org/pdf/1904.08994.pdf"

$$P_1 = 3, P_2 = 2, P_3 = 1, P_4 = 4$$
$$Q_1 = 1, Q_2 = 2, Q_3 = 4, Q_4 = 3$$

If we label the cost to pay to make P_i and Q_i match δ_i, we would have $\delta_{i+1} = \delta_i + P_i - Q_i$ and in example:

$$\begin{aligned}
\delta_0 &= 0 \\
\delta_1 &= 0 + 3 - 1 = 2 \\
\delta_2 &= 2 + 2 - 2 = 2 \\
\delta_3 &= 2 + 1 - 4 = -1 \\
\delta_4 &= -1 + 4 - 3 = 0
\end{aligned} \tag{1}$$

When dealing with the continuous probability domain, the distance formula becomes:

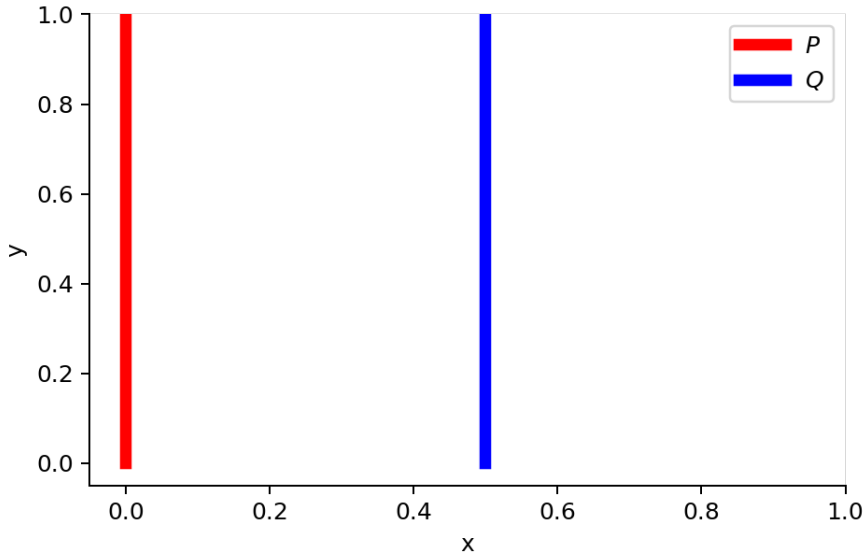
$$W(p^*(x), p(z)) = \inf_{\gamma \sim \Pi(p^*(x), p(z))} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \tag{2}$$

Image source: https://arxiv.org/pdf/1904.08994.pdf".

Why Wasserstein is better than JS or KL divergence?

Suppose we have two probability distributions, P and Q:

$$\forall(x, y) \in P, x = 0 \text{ and } y \sim U(0, 1)$$
$$\forall(x, y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim U(0, 1) \tag{3}$$



$$
\begin{aligned}
D_{KL}(P\|Q) &= \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty \\
D_{KL}(Q\|P) &= \sum_{x=\theta, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty \\
D_{JS}(P, Q) &= \frac{1}{2}\left(\sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2}\right) = \log 2 \\
W(P, Q) &= |\theta|
\end{aligned}
\tag{4}
$$

But when $\theta = 0$, two distributions are fully overlapped:

$$
\begin{aligned}
D_{KL}(P\|Q) = D_{KL}(Q\|P) = D_{JS}(P, Q) = 0 \\
W(P, Q) = 0 = |\theta|
\end{aligned}
\tag{5}
$$

From "https://arxiv.org/pdf/1904.08994.pdf".

The differences in implementation for the WGAN

1. Use a linear activation function in the output layer of the critic model (instead of sigmoid).
2. Use -1 labels for real images and 1 labels for fake images (instead of 1 and 0).
3. Use Wasserstein loss to train the critic and generator models.
4. Constrain critic model weights to a limited range after each mini batch update (e.g. [-0.01,0.01]).
5. Update the critic model more times than the generator each iteration (e.g. 5).
6. Use the RMSProp version of gradient descent with a small learning rate and no momentum (e.g. 0.00005).

From "https://arxiv.org/pdf/1904.08994.pdf".