ELEC96033: Deep Learning

Carlo Ciliberto and Krystian Mikolajczyk

Department of Electrical and Electronic Engineering
Imperial College London

Previous Class

Unsupervised Learning as a way to learn the uderlying structure of a probability distribution.

Useful to learn a good representation for data that can be useful for

- Clustering,
- Visualization,
- Supervised learning (feature representation)
- ...
- Density estimation **(Today!)**

Today's question: Can we use the structure learned this way to approximate the distribution and possibly even... sample new points from it?

Why Generating Data?

Before going futher let's ask why should we do that (besides because it's cool):

- Data Agumentation for Supervision,
- Simulation for Reinforcement Learning,
- Data Reconstruction (Colorizatrion, Super Resolution, Inpainting, etc.)
- Well... because it **is** cool...

(a) Original face (b) Masked input (c) Our result

Models... for Generative Models

Throughout this class we will consider a parametrization P_θ for the probability distribution that we aim to fit to the underlying unknown data generating $P_* = P_{true}$.

Two main families of parametrizations for P_θ:

- **Explicit:** when the density of P_θ is fully analitically modeled.
- **Implicit:** when the density of P_θ is modeled as a sequence of operations.

Standard Approach: Maximum Likelihood

Intuition: if we observe $x_1, \ldots, x_n$, it means that *they are very likely to be sampled* from P_*.

In other words: $P_*(x_1, \ldots, x_n)$ is large!

Idea: among all possible P_θ find the one that is most likely to produce the dataset...

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \; P_\theta(x_1, \ldots, x_n)$$

We are maximizing the likelihood of observing $x_1, \ldots, x_n$...

Maximum Likelihood Reformulation

Simplification: If we assumne the observed $x_1, \ldots, x_n$ to be sampled independently...

$$
\begin{aligned}
\hat{\theta} &= \operatorname*{argmax}_{\theta} \; P_\theta(x_1, \ldots, x_n) \\
&= \operatorname*{argmax}_{\theta} \; P_\theta(x_1) \cdots P_\theta(x_n) && \text{(independence)} \\
&= \operatorname*{argmax}_{\theta} \; \log\left(P_\theta(x_1) \cdots P_\theta(x_n)\right) && \text{(monotonicity of log)} \\
&= \operatorname*{argmax}_{\theta} \; \sum_{i=1}^{n} \log\left(P_\theta(x_i)\right) && \text{(properties of log)}
\end{aligned}
$$

Typically: much easier to optimize w.r.t. θ

Example: Gaussian Model

Suppose data lies in $\mathcal{X} = \mathbb{R}$ and P_θ parametrize all possible Normal distributions on $\mathcal{X}$ with fixed variance σ^2.

We have $P_\theta(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\theta)^2}{\sigma^2}}$ The maximum likelihood is given by

$$\begin{aligned}\hat{\theta} &= \operatorname*{argmax}_\theta \sum_{i=1}^{n} \log(P_\theta(x_i)) \\ &= \operatorname*{argmin}_\theta \sum_{i=1}^{n} (x_i - \theta)^2\end{aligned}$$

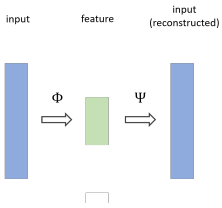
which is minimized by $\theta = \frac{1}{n}\sum_{i=1}^{n} x_i$ the empirical average of the training data.

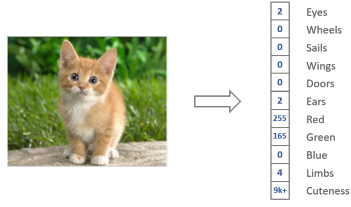
Generative Models: Variational Autoencoder

Recap: Autoencoders

Why again? We will use Autoencoders to find suitable designs for P_θ

Recall: we interpreted Autoencoders as models to learn a representation of the data.



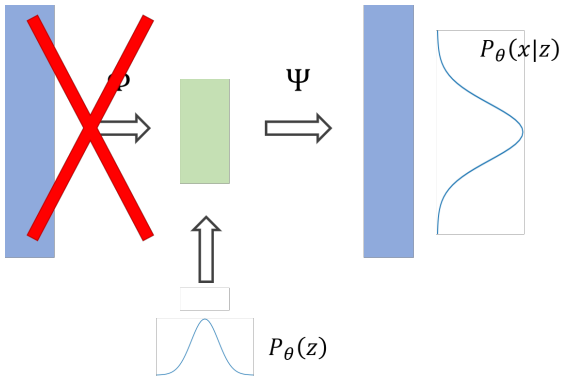


Two key players:

- **Coding**: feature map $\phi : \mathcal{X} \to \mathcal{Z}$ from the input space to a latent space (e.g. $\mathcal{Z} = \mathbb{R}^k$),
- **Decoding:** $\psi : \mathcal{Z} \to \mathcal{X}$ recovers the original input when coupled with ϕ.

Variational Autoencoders

Idea: if the representation can "reconstruct" a point, could it be used to generate new ones?



Variational Autoencoders (Cont.)

Model: we thus consider $P_\theta(x) = \int P_\theta(x|z)P_\theta(z)\ dz$

- $P_\theta(z)$ "prior" distribution on the representation
- $P_\theta(x|z)$ likelihood of observing x given the representation z.

Typically:

- $P_\theta(z)$ "simple", e.g. a Gaussian $\mathcal{N}(\mu, \Sigma)$
- $P_\theta(x|z)$ a sophisticated model, e.g. a $\mathcal{N}(\mu_{\psi(z)}, \Sigma_{\psi(z)})$.
  (here $\psi : \mathcal{X} \to \mathbb{R}^k$ is a decoding network)

Goal: learn the parameters $\theta = (\mu, \Sigma, \psi)$.

Variational Autoencoders (Cont.)

Model: $P_\theta(x) = \int P_\theta(x|z)P_\theta(z)\ dz$

- $P_\theta(z)$ "prior" distribution on the representation
- $P_\theta(x|z)$ likelihood of observing x given the representation z.

Then...

...it is very hard to compute

$$P_\theta(z|x) = \frac{P_\theta(x|z)P_\theta(z)}{P_\theta(x)}$$

(It is much easier to *apprximate it*)

$$Q_\phi(z|x) \approx P_\theta(z|x)$$

Maximum Likelihood for Variational Autoencoders

The Maximum Likelihood estimation problem becomes

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \sum_{i=1}^{n} \log(P_\theta(x_i)) = \underset{\theta}{\text{argmax}} \sum_{i=1}^{n} \log\left(\int P_\theta(x_i|z)P_\theta(z))\ dz\right)$$

Problem: in general we don't know how to deal with that integral...

Connection: there exist other approaches that allow for a tractable parametrization:

- Fully Visible Belief Networks
- Nonlinear ICA

(we are not going to consider them here since they are typically slow...)

Maximum Likelihood for Variational Autoencoders - Take 2

Let's try a different approch to represent $\log P_\theta(x)$ in terms of $P_\theta(x|z)$ and $P_\theta(z)$:

$$
\begin{aligned}
\log(P_\theta(x)) &= \mathbb{E}_{P(z|x)}\ [\log(P_\theta(x))] \\
&= \mathbb{E}_{P(z|x)}\ \left[\log\left(\frac{P_\theta(x|z)P_\theta(z)}{P_\theta(z|x)}\right)\right] \\
&= \mathbb{E}_{P(z|x)}\ [\log(P_\theta(x|z)] - \mathbb{E}_{P(z|x)}\ \left[\log\left(\frac{P_\theta(z|x)}{P_\theta(z)}\right)\right] \\
&= \mathbb{E}_{P(z|x)}\ [\log(P_\theta(z|x)] - KL\big(P_\theta(x|z)\ ||\ P_\theta(z)\big)
\end{aligned}
$$

$KL(P|Q) = \mathbb{E}_P[\log\left(\frac{P}{Q}\right)]$ is the Kullback-Leibler divergence between probabilities P and Q.

Maximum Likelihood for Variational Autoencoders - Take 2 (Cont.)

So the Maximum Likelihood estimation problem can be addressed as...

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \sum_{i=1}^{n} \underbrace{\mathbb{E}_{P_\theta(z|x_i)}[\log(P_\theta(x_i|z)]}_{\substack{\text{encourages good} \\ \text{reconstruction}}} - \underbrace{KL\big(P_\theta(z|x_i) \;||\; P_\theta(z)\big)}_{\substack{\text{encourages similar} \\ \text{resentation for each input}}}$$

Optimization: we can optimize w.r.t. θ by stochastic approaches by iterating:

- sample i uniformly between 1 and n,
- sample $\tilde{z}$ from $P_\theta(z|x_i)$,
- perform an ascent step with gradient $\nabla_\theta \left[\log(P_\theta(x_i|\tilde{z})) - \log\left(\frac{P_\theta(\tilde{z}|x_i)}{P_\theta(\tilde{z})}\right) \right]$,

Problem: we don't know how to get $P_\theta(z|x)$!

Maximum Likelihood for Variational Autoencoders - Take 3

$P_\theta(z|x)$ captures the likelihood of a representation (encoding) z of an input x...

Idea: approximate $P_\theta(z|x)$ via a probability $Q_\phi(z|x)$ such as $\mathcal{N}(\mu_{\phi(x)}, \Sigma_{\phi(x)})$,
(namely a distribution parametrized by the coding network ϕ).

Maximum Likelihood: we can substitute $P_\theta(z|x_i)$ with $Q_\phi(z|x_i)$

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \sum_{i=1}^{n} \mathbb{E}_{Q_\phi(z|x_i)}[\log(P_\theta(x_i|z)] - KL\big(Q_\phi(z|x_i) \;||\; P_\theta(z)\big)$$

Maximum Likelihood for Variational Autoencoders - Take 3 (Cont.)

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \sum_{i=1}^{n} \mathbb{E}_{Q_\phi(z|x_i)}[\log(P_\theta(x_i|z)] - KL\big(Q_\phi(z|x_i) \;||\; P_\theta(z)\big)$$

We have a similar structure to standard Autoencoders:

- $Q_\phi(z|x)$ parametrized by the encoding network ϕ,
- $P_\theta(x|z) = P_\psi(x|z)$ parametrized by the decoding network ψ,

Intuitively: we are adding a probabilistic perspective on top of the Autoencoder model.

Maximum Likelihood for Variational Autoencoders - Take 3 (Cont.)

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \sum_{i=1}^{n} \mathbb{E}_{Q_\phi(z|x_i)}[\log(P_\theta(x_i|z)] - KL\big(Q_\phi(z|x_i) \;||\; P_\theta(z)\big)$$

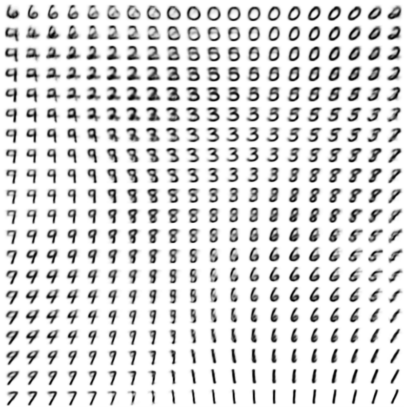
Optimization: we can optimize w.r.t. θ and ϕ by stochastic approaches by iterating:

- sample i uniformly between 1 and n,
- sample $\tilde{z}$ from $Q_\phi(z|x_i)$,
- update θ by performing an ascent step with gradient $\nabla_\theta \left[\log(P_\theta(x_i|\tilde{z})) - \log\left(\frac{Q_\phi(\tilde{z}|x_i)}{P_\theta(\tilde{z})}\right) \right]$,
- update ϕ by performing an ascent step with gradient $\nabla_\phi \; - \log\left(\frac{Q_\phi(\tilde{z}|x_i)}{P_\theta(\tilde{z})}\right)$,

Generating Data with Variational Autoencoders

Once $\hat{\theta}$ has been learned, we can generate new points by:

- Sampling $z \sim P_{\hat{\theta}}(z)$,
- Sampling $x \sim P_{\hat{\theta}}(x|z)$.



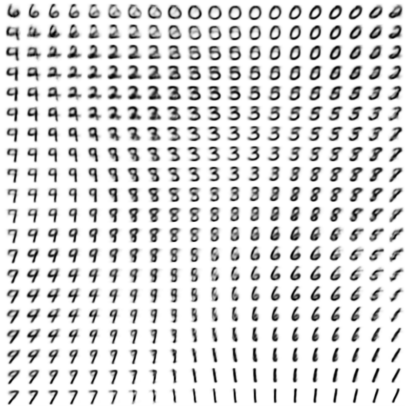
Variational Autoencoders: Pros & Cons

Pros:

- We can re-use everything we have learned for autoencoders.
- VAEs are typically easy to train.

Cons:

- Generated samples are not "sharp".



Next: Take a different perspective to enforce more explicitly our desire to generate samples that are similar to training data.

Generative Models: Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs)

Setting: we move away from Maximum Likelihood settings into the domain of "games"...

Idea: a successful P_θ should produce same samples of P_*.

Therefore it should be able to "fool" an anomaly detector D for P_* (ideally $D =$ "us").

Approach: suppose to be provided with a detector (or discriminator) $D : \mathcal{X} \to [0, 1]$.

We could find a good $P_\theta(x)$ by solving

$$\hat{\theta} = \underset{\theta}{\text{argmin}} \ \mathbb{E}_{P_\theta(x)}[\log(1 - D(x))]$$

Problem: this could simply encourage P_θ to concentrate on a single (correct) point from P_*.

Generative Adversarial Networks (GANs): Discriminator

We would like a discriminator that cannot be "fooled" by just choosing one (or a few) correct examples...

Actually, we would like D to be the best discriminator between P_* and P_θ:

$$\hat{D} = \underset{D}{\text{argmax}} \; \mathbb{E}_{P_*(x)}[\log(D(x))] + \mathbb{E}_{P_\theta(x)}[\log(1 - D(x))]$$

- $\mathbb{E}_{P_*(x)}[\log(D(x))]$: encourages D to classify samples from P_* with "label" 1,
- $\mathbb{E}_{P_\theta(x)}[\log(1 - D(x))]$: encourages D to classify samples from P_θ with "label" 0.

Generative Adversarial Networks (GANs) as a Min/Max Game

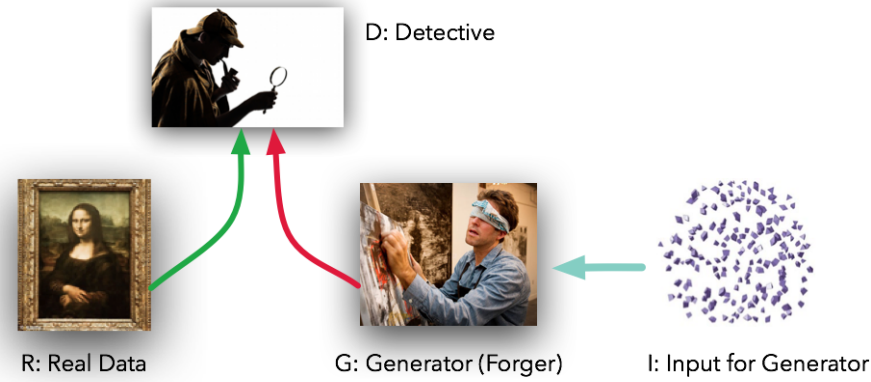
Therefore we can formulate the optimization problem as

$$\min_{\theta} \max_{D} \quad \mathbb{E}_{P_*(x)}[\log(D(x))] + \mathbb{E}_{P_\theta(x)}[\log(1 - D(x))]$$

This corresponds to a "game" between two adversaries:

- A **Generator** P_θ, trying to create samples that can confuse...
- ... a **Discriminator** D, whose goal is to be able to identify samples from P_θ as wrong.

Generative Adversarial Networks (GANs) as a Min/Max Game



Generative Adversarial Networks (GANs) as a Min/Max Game

This suggest also a strategy to find the best generator P_θ and discriminator D:

Suppose that we are provided with the usual set of observations $x_1, \ldots, x_n$ from P_*.

Initialize: the generator with parameter θ_0.

Iterate: for $t = 0, 1, \ldots, T, \ldots$

- Generate $\tilde{x}_1, \ldots \tilde{x}_n$ from P_{θ_t}.
- Train a classifier D_t to discriminate between $(x_i)_{i=1}^n$ (class 1) and $(\tilde{x}_i)_{i=1}^n$ (class 0).
- Update the generator from P_{θ_t} to $P_{\theta_{t+1}}$ in order to minimize

$$\mathbb{E}_{P_\theta(x)}[\log(1 - D_t(x))]$$

Question: how do we perform the last step?

Generative Adversarial Networks (GANs): Generator

We need to choose a parametrization of P_θ that is suitable for the Min Max game.

Idea: similarly to what done for the Variational Autoencoders $P_\theta(x) = \int P_\theta(x|z)P_\theta(z)\ dz$

With the following two differencences:

- $P_\theta(z) = P(z)$ is fixed a prior, e.g. $\mathcal{N}(0, I)$ (it is just "noise"!),
- $P_\theta(x|z) = \delta_{\psi(z)}(x)$ is a Dirac's delta: namely x is a function of z.

Why? Because now, the expectation w.r.t. $P_\theta(x)$ becomes

$$\mathbb{E}_{P_\theta(x)}[\log(1 - D(x))] = \mathbb{E}_{P(z)}[\log(1 - D(\psi(z))]$$

We can easily compute the gradient with respect to θ (actually ψ)!

Generative Adversarial Networks (GANs): Training

Initialize: Generator and Discriminator networks ψ and ϕ.

Iterate: for number of epochs $t = 1, \ldots, T, \ldots$

- Generate "noise" samples $z_1, \ldots, z_m$ from $P(z)$. Evaluate $\tilde{x}_i = \psi(z_i)$,
- Sample a minibatch of "true" samples $x_1, \ldots, x_m$ from the training set.
- Update ϕ by one step of Sochastic Gradient Ascent

$$\nabla_\phi \sum_{i=1}^{m} \log(\phi(x_i)) + \log(1 - \phi(\tilde{x}_i))$$

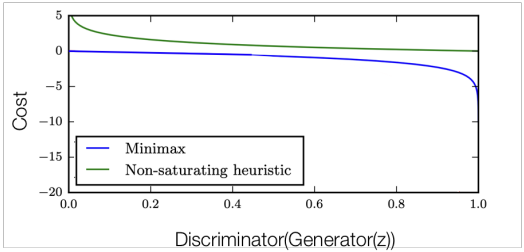
- Update ψ by one step of Stochastic Gradient Descent

$$\nabla_\psi \sum_{i=1}^{m} \log(1 - \phi(\psi(z_i)))$$

Issues with Training

Consider two cost functions for the Generator:

- **Min Max cost:** $\log(1 - \phi(\psi(z_i)))$
- **Non-saturating Euristic:** $-\log(\phi(\psi(z_i)))$



They have same argmax but the gradients are very different:

- Min Max has a large gradient when samples from ψ are already good...
- Non-saturating has a large gradient when ψ is producing bad samples!

In practice: this suggests to optimize ψ with respect to the non-saturating heuristic...

Generative Adversarial Networks (GANs): Training (Improved)

Initialize: Generator and Discriminator networks ψ and ϕ.

Iterate: for number of epochs $t = 1, \ldots, T, \ldots$

- Generate "noise" samples $z_1, \ldots, z_m$ from $P(z)$. Evaluate $\tilde{x}_i = \psi(z_i)$,
- Sample a minibatch of "true" samples $x_1, \ldots, x_m$ from the training set.
- Update ϕ by one step of Sochastic Gradient Ascent

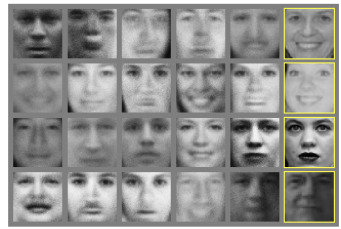
$$\nabla_\phi \sum_{i=1}^{m} \log(\phi(x_i)) + \log(1 - \phi(\tilde{x}_i))$$

- Update ψ by one step of Stochastic Gradient Ascent

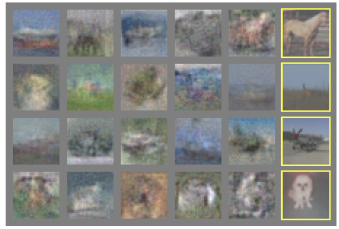
$$\nabla_\psi \sum_{i=1}^{m} \log(1 - \phi(\psi(z_i))) \quad \longrightarrow \quad \nabla_\psi \sum_{i=1}^{m} \log(\phi(\psi(z_i)))$$

Examples

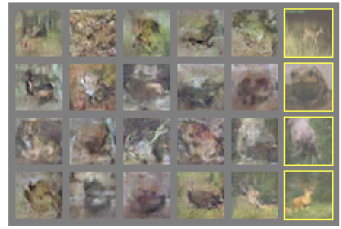
a)



b)

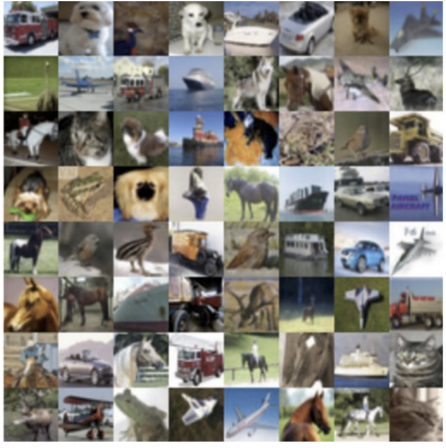


c)

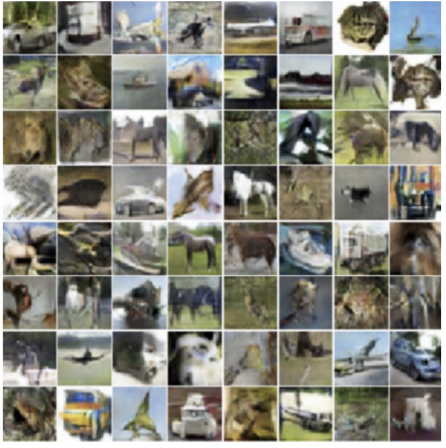


d)

More Examples



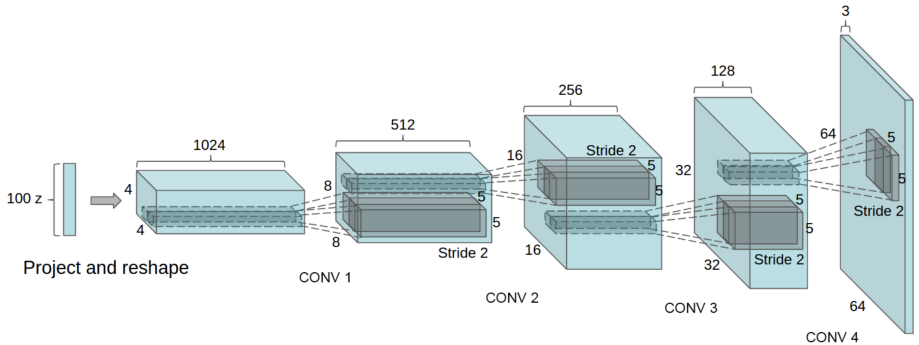
Training Data



Samples

Better Models for ψ: Deep Convolutional Generative Adversarial Networks (DCGANs)

When dealing with images, a well-established model for ψ is a "reverse" convolutional network

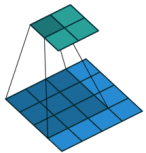


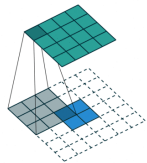
For the discriminator D we can simply use a CNN ϕ

Deep Convolutional Generative Adversarial Networks (DCGANs): Tips & Tricks

Things to keep in mind:

- Do not use pooling but only convolutions (discriminator) and deconvolutions (generator) (aka respectively as strided and transposed convolutions)





- Use ReLUs for all activation functions of the generator except tanh for the final output,
- Use Leaky ReLUs for the discriminator.
- Use batch normalization for all layers except for the image layer (both Gen. and Disc.)

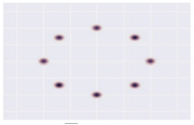
Examples: DCGANs

Problems with GANs: Mode Collapse

It is in general quite challenging to properly train GANs.

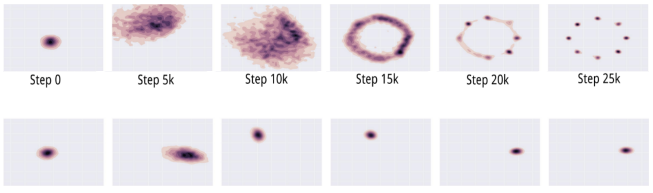
Among the many problems one major concern is the so-called mode collapse:

Suppose to have a target distribution on the plane $\mathcal{X} = \mathbb{R}^2$



Expected

Output



Mode Collapse: Why and How to Avoid It

Intuition: The GAN functional corresponds to minimizing a "distance" $d(P_*, P_\theta)$

$$\min_\theta \underbrace{\max_D \quad \mathbb{E}_{P_*(x)}[\log(D(x))] + \mathbb{E}_{P_\theta(x)}[\log(1 - D(x))]}_{d(P_*, P_\theta)}$$

This measure $d(P_*, P_\theta)$ is not able to capture all moments of the distribution.

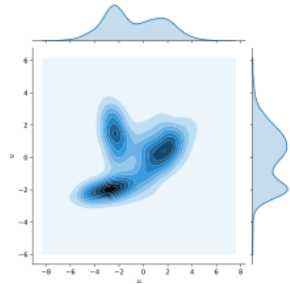
A Better Distance. We can consider the Wasserstein Distance (aka Earth Mover Distance)

$$W(P_*, P_\theta) \;=\; \min \left\{ \mathbb{E}_{T(x,y)}\Big[\|x - y\|\Big] \;\middle|\; P_*(x) = \int T(x,y)\, dy, \quad P_\theta(y) = \int T(x,y)\, dx \right\}$$

Wasserstein Distance

Very natural metric between probabilities:

- Intuitively measures the "effort" required to move the mass of P_* to recover P_θ



Dual Formulation: the Wasserstein distance can be also formulated as

$$W(P_*, P_\theta) = \sup_{\|f\|_{\text{Lip}} \leqslant 1} \mathbb{E}_{P_*(x)}[f(x)] - \mathbb{E}_{P_\theta(x)}[f(x)]$$

Wasserstein GANs (WGANs)

Therefore, the problem

$$\min_{\theta} W(P_*, P_\theta) = \min_{\theta} \sup_{\|f\|_{\text{Lip}} \leqslant 1} \mathbb{E}_{P_*(x)}[f(x)] - \mathbb{E}_{P_\theta(x)}[f(x)]$$

can again be interpreted as a game:

- P_θ aims to generate samples such that in expectation...
- ...the "discriminator" f cannot separate from those of P_*.

In Practice: we don't know how to optimize over the set $\{\|f\|_{\text{Lip}} \leqslant 1\}$

However, a neural network f_θ can be enforced to be L-Lipschitz by clipping its weights θ.

WGAN Training

Initialize: Generator and Discriminator networks ψ and f_θ.

Iterate: for number of epochs $t = 1, \ldots, T, \ldots$

- Generate "noise" samples $z_1, \ldots, z_m$ from $P(z)$. Evaluate $\tilde{x}_i = \psi(z_i)$,
- Sample a minibatch of "true" samples $x_1, \ldots, x_m$ from the training set.
- Update ϕ by one step of Sochastic Gradient Ascent

$$\nabla_\theta \sum_{i=1}^{m} f_\theta(x_i) - f_\theta(\tilde{x}_i)$$

- For any network parameter $|\theta_j| > \tau$ greater than a thresold τ: clip it.
- Update ψ by one step of Stochastic Gradient Descent

$$\nabla_\psi - \sum_{i=1}^{m} f_\theta(\phi(\psi(z_i)))$$

Examples with WGANs

Classic DCGANs

Wasserstein DCGANs

Summary

We have investigated two approaches to Density Estimation:

- **Variational Autoencoders** (via Maximum Likelihood)
  - Pros: reasonably stable to train
  - Cons: typically less sharp generated samples
- **Generative Adversarial Networks** (via Min Max Games)
  - Pros: very sharp samples
  - Cons: often hard to train, Mode Collaps (improvable via WGANs!)

Next Class

We will move from Unsupervised Learning to a different topic: Deep Reinforcement Learning