Generative Models, part I

Machine Learning in High Energy Physics

Maxim Borisyak

National Research University Higher School of Economics

June 30, 2019

Generative models

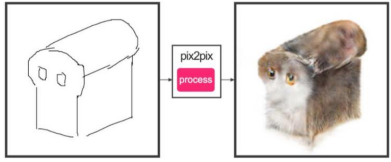
Generative models

Given samples of a random variable X, find X', such that:

$$P(X) \approx P(X')$$

Applications

- auxiliary tasks:
  - pivoted models;
- data manipulation:
  - realistic image-to-image translation;
- approximation of existing generators:
  - fast Monte-Carlo;
- data compression.



Images are from `https://arxiv.org/abs/1611.07004`

Types of generative models

Density estimation:

- usually, density is known up to a constant (e.g. RBM):

$$f(x) = C \cdot p(x)$$

- sampling is via MCMC;
- challenging in high dimensional spaces.

Types of generative models

Sampling procedure:

- learning transformation from a simple random variable to the target one:

$$\begin{aligned} Z &\sim \mathcal{N}^n(0,1); \\ X' &= f(Z). \end{aligned}$$

- density is often intractable:

$$p(x) = \sum_{z|f(z)=x} p(z) \left| \frac{\partial}{\partial z} f(z) \right|^{-1}$$

Kernel Density Estimation

Kernel Density Estimation

Main idea: place a small gaussian-like function around each sample:

- sample $\{x_i\}_{i=1}^N$
- kernel $k(x)$:

$$\int_{\mathcal{X}} k(x)\,dx = 1;$$

- kernel width h;

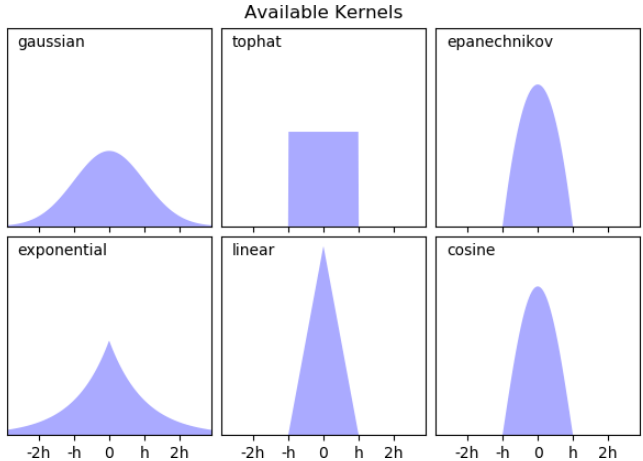
$$f_{KDE}(x) = \frac{1}{hN}\sum_i k\left(\frac{x - x_i}{h}\right);$$

(almost) theorem

When $N \to \infty$ and $h \to 0$:

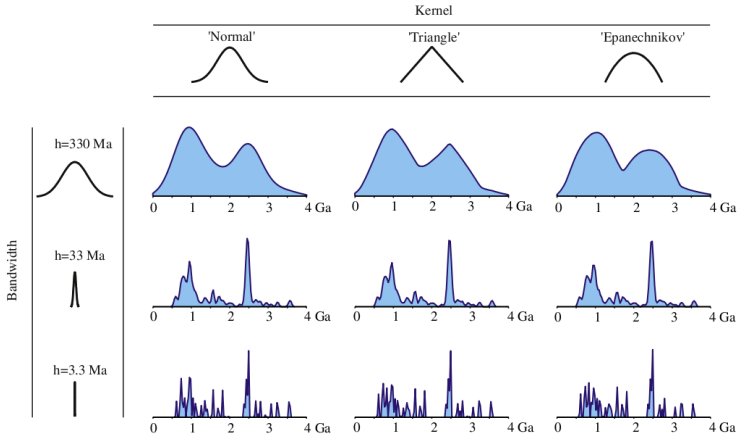
$$f_{KDE}(x) \to p(x).$$

Kernel Density Estimation



https://scikit-learn.org/stable/modules/density.html

Kernel Density Estimation



https://www.ucl.ac.uk/ ucfbpve/papers/VermeeschChemGeol2012/

Kernel Density Estimation

- kernel selection:
  - Gaussian, Epanechnikov kernels produce smooth densities;
  - Epanechnikov, linear kernels are faster to compute:
    - depends on nearest neighbour algorithm;
- kernel width:
  - a hyper-parameter;
  - selected by e.g. cross-validation.

Kernel Density Estimation

Advantages:

- (almost) simple;
- (almost) trivial sampling;
- (almost) no training;
- explicit density estimation;
- great for 1-2 dimensional problems.

Disadvantages:

- complexity $O(\log N)$ for evaluation in one point;
- performs poorly on high-dimensional problems;
- need to memorize the whole training set.

Gaussian Mixtures

Gaussian Mixtures

A similar to KDE idea — describe probability density as a mixture of Gaussians:

- unlike KDE, centers and widths of Gaussians are learnt;
- usually, number of components n is much smaller than the number of samples N.

$$f(x) = \frac{1}{n} \sum_j \phi(x \mid \mu_j, \Sigma_j)$$

where:

- $\phi(x \mid \mu, \Sigma)$ — density of the Gaussian distribution with mean μ and covariance matrix Σ.

Gaussian Mixtures

Training is done by maximizing likelihood:

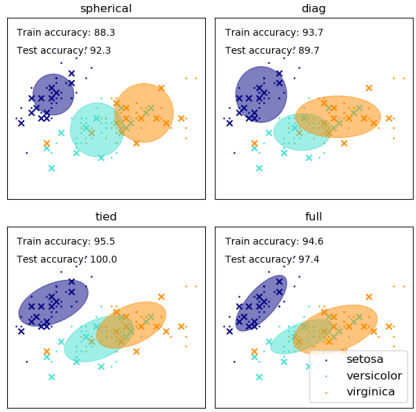
$$L = \sum_i \log \left(\frac{1}{N} \sum_j \phi(x_i \mid \mu_j, \Sigma_j) \right) \to \max$$

- Expectation-Maximization algorithm;
- priors are possible (ML $\to$ MAP).

Gaussian Mixtures

Types of GMM:

- spherical: $\Sigma = \sigma \cdot \mathbb{I}$;
- diagonal: $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \dots)$;
- tied: $\Sigma_1 = \Sigma_2 = \dots$;
- ...
- full;



https://scikit-learn.org/stable/modules/mixture.html

Gaussian Mixtures

Advantages:

- simple;
- trivial sampling;
- explicit density estimation;
- great for 1-2 dimensional problems;
- tends to have fewer parameters than KDE.

Disadvantages:

- complexity $O(n)$ for evaluation in one point;
- performs poorly on high-dimensional problems;

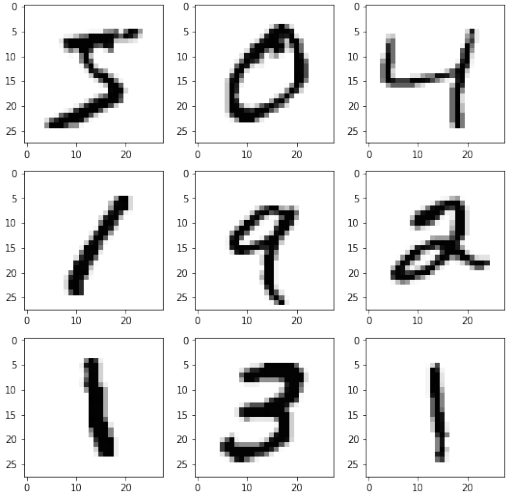
Variational AutoEncoder

Latent variables revisited

Before generating a sample, model should first decide what it should generate:

- which digit to generate: 0, 1, ..., 9
- width of stokes;
- 'speed';
- etc.

Such decision can be represented as **latent variables**.



Variational AutoEncoder

VAE **non-deterministically** transforms latent variables Z into samples X:

1. given latent variables z, VAE computes $f(z)$;
2. $f(z)$ represents parameters of some distribution;
3. examples are sampled from $P(x \mid f(z))$.

This section is largely based on `https://arxiv.org/abs/1606.05908`.

Variational AutoEncoder

Common choices:

- continuous data: $P(x \mid f(z)) = \mathcal{N}(x \mid f(z), \sigma^2 \mathbb{I})$;
  - σ — hyper-parameter;
  - $\mathbb{I}$ — identity matrix.
- discrete data:
  - $P(x \mid f(z)) = \mathrm{Bi}(x \mid f(z))$:
  - $P(x \mid f(z)) = \mathrm{Multi}(x \mid f(z))$.

How to choose latent variables?

How to choose latent variables?

- let $\hat{P}$ be a magical optimal choice of latent variables;
- let $z \sim \mathcal{N}^m(0, 1)$;
- if model $\mathcal{G}$ has enough capacity, then

$$\exists g \in \mathcal{G} : g(z) \sim \hat{P}.$$

Let the network assign the meaning of the latent variables.

VAE training

Maximum Likelihood:

$$\sum_i \log P(x_i) \to \max$$

where $\{x_i\}_{i=1}^N$ — observed data.

$$P(x) = \int P(x \mid z) P(z)\, dz = \mathop{\mathbb{E}}_{Z} P(x \mid Z)$$

- for the most of z: $P(x \mid z) \approx 0$.

How to deal with the integral?

Variational bound

$$P(x) = \int P(x \mid z) P(z)\, dz = \mathop{\mathbb{E}}_{Z} P(x \mid Z)$$

In order to make sampling tractable, P(z) can be replaced by some $Q(z \mid x)$:

$$P(x) = \mathop{\mathbb{E}}_{Z} P(x \mid Z) \to \mathop{\mathbb{E}}_{Z \sim Q(z|x)} P(x \mid Z)$$

Let's consider KL divergence:

$$\mathrm{KL}\left(Q(z \mid x) \parallel P(z \mid x)\right) = \mathop{\mathbb{E}}_{Z \sim Q(z|x)} \left[\log Q(Z \mid x) - \log P(Z \mid x)\right]$$

Variational bound

$$\mathrm{KL}\left(Q(z \mid x) \mid P(z \mid x)\right) =$$

$$\mathop{\mathbb{E}}_{Z \sim Q(z|x)} \left[\log Q(Z \mid x) - \log P(Z \mid x)\right] =$$

$$\mathop{\mathbb{E}}_{Z \sim Q(z|x)} \left[\log Q(Z \mid x) - \log P(x \mid Z) - \log P(Z)\right] + \log P(x)$$

$$\boxed{\log P(x) - \mathrm{KL}\left(Q(z \mid x) \parallel P(z \mid x)\right) = \mathop{\mathbb{E}}_{Z \sim Q(z|x)} \log P(x \mid Z) - \mathrm{KL}\left(Q(z \mid x) \parallel P(z)\right)}$$

Variational bound

$$l(x) = \underbrace{\log P(x)}_{\text{MLE objective}} - \underbrace{\mathrm{KL}\left(Q(z \mid x) \parallel P(z \mid x)\right)}_{\text{inference penalty, } \geq 0} =$$

$$\underbrace{\mathop{\mathbb{E}}_{Z \sim Q(z|x)} \log P(x \mid Z)}_{\text{reconstruction error}} - \underbrace{\mathrm{KL}\left(Q(z \mid x) \parallel P(z)\right)}_{\text{regularization}}$$

$$\boxed{\log P(x) \geq l(x) \to \max}$$

VAE objective

$$\mathcal{L} = \sum_i \left[\mathop{\mathbb{E}}_{Z \sim Q(z|x_i)} \log P(x_i \mid Z) - \mathrm{KL}\left(Q(z \mid x_i) \parallel P(z)\right) \right]$$

- reconstruction error can be estimated by sampling z from $Q(z \mid x_i)$:

$$\mathop{\mathbb{E}}_{Z \sim Q(z|x_i)} \log P(x_i \mid Z) \to \mathrm{RE}(x_i, z)$$

- regularization term is, usually, computed analytically.

Reconstruction error

$$\mathrm{RE}(x, z) = \log P(x \mid z)$$

- for Gaussian posterior i.e. $P(x \mid z) = \mathcal{N}(x \mid f(z), \sigma^2 I)$:

$$\mathrm{RE}(x, z) \propto (f(z) - x)^2$$

- for Benulli posterior (e.g. for discrete output) $P(X = 1 \mid z) = f(z)$:

$$\mathrm{RE}(x, z) = x \log f(z) + (1 - x) \log(1 - f(z))$$

Limitations

Image (b) — slightly altered image (a), image (c) — image (a) shifted by several pixels. Under MSE metric, image (b) is much closer to (a), than (c) to (a).

Regularization

Consider:

- $Q(z \mid x) = \mathcal{N}(z \mid \mu(x), \Sigma(x))$;
- $P(z) = \mathcal{N}(0, I)$:

$$\mathrm{KL}\left(\mathcal{N}(x \mid \mu(z), \Sigma(z)) \parallel \mathcal{N}(x \mid \mu(z), \Sigma(z))\right) =$$

$$\frac{1}{2}\left(\mathrm{tr}(\Sigma(x)) + \|\mu(x)\|^2 - k - \log\det\Sigma(x)\right) =$$

$$\frac{1}{2}\left(\|\mu(x)\|^2 + \sum_i \Sigma_{ii}(x) - \log\Sigma_{ii}(x)\right) - \frac{k}{2}$$

Training time

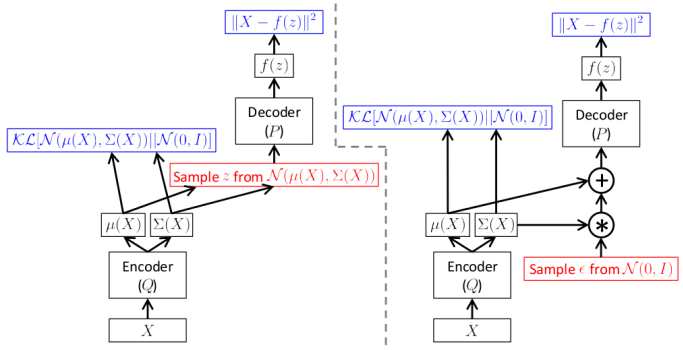


Figure 4: A training-time variational autoencoder implemented as a feed-forward neural network, where $P(X|z)$ is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but backpropagation can be applied only to the right network.

Testing time

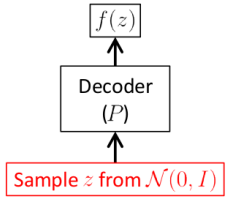


Figure 5: The testing-time variational "autoencoder," which allows us to generate new samples. The "encoder" pathway is simply discarded.

Conditional VAE

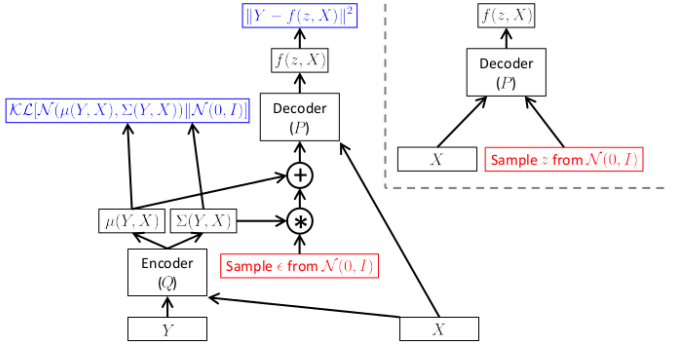


Figure 6: Left: a training-time conditional variational autoencoder implemented as a feedforward neural network, following the same notation as Figure 4. Right: the same model at test time, when we want to sample from $P(Y|X)$.

Generative Adversarial Networks

Fitting Distributions

Notation: Q - ground truth distribution, P - model distribution.

Maximum Likelihood:

$$\mathcal{L} = \sum_i \log P(x_i) \approx \mathop{\mathbb{E}}_{X\sim Q} \log P(X) \to_P \min;$$

$$\mathrm{KL}\left(Q \parallel P\right) = \mathop{\mathbb{E}}_{X\sim Q} \log Q(X) - \mathop{\mathbb{E}}_{X\sim Q} \log P(X) \to_P \min.$$

Jensen-Shannon distance:

$$\mathrm{JS}(P, Q) = \frac{1}{2}\left[\mathrm{KL}\left(P \parallel M\right) + \mathrm{KL}\left(Q \parallel M\right)\right] \to_P \min;$$

$$M = \frac{1}{2}(P + Q).$$

Approximating JS distance

$$\mathrm{JS}(P, Q) = \frac{1}{2}\left[\mathop{\mathbb{E}}_{X\sim P} \log \frac{P(X)}{M(X)} + \mathop{\mathbb{E}}_{X\sim Q} \log \frac{Q(X)}{M(X)}\right] =$$

$$\frac{1}{2}\left[\mathop{\mathbb{E}}_{X\sim P} \log \frac{P(X)}{P(X)+Q(X)} + \mathop{\mathbb{E}}_{X\sim Q} \log \frac{Q(X)}{P(X)+Q(X)}\right] + \log 2 =$$

$$\mathop{\mathbb{E}}_{X\sim M} \frac{P(X)}{P(X)+Q(X)} \log \frac{P(X)}{P(X)+Q(X)} + \mathop{\mathbb{E}}_{X\sim M} \frac{Q(X)}{P(X)+Q(X)} \log \frac{Q(X)}{P(X)+Q(X)} + \log 2$$

Approximating JS distance

Let's introduce binary indicator y: $y = 1$ if x is sampled from P and $y = 0$ for Q:

$$\mathrm{JS}(P, Q) - \log 2 =$$

$$\mathop{\mathbb{E}}_{X \sim M} \frac{P(X)}{P(X) + Q(X)} \log \frac{P(X)}{P(X) + Q(X)} + \mathop{\mathbb{E}}_{X \sim M} \frac{Q(X)}{P(X) + Q(X)} \log \frac{Q(X)}{P(X) + Q(X)} =$$

$$\mathop{\mathbb{E}}_{X \sim M,\, Y} P(\,Y = 1 \mid X) \log P(\,Y = 1 \mid X) + P(\,Y = 0 \mid X) \log P(\,Y = 0 \mid X) =$$

$$\max_{f} \mathop{\mathbb{E}}_{X,\, Y} Y \log f(X) + (1 - Y) \log(1 - f(X))$$

Approximating JS distance

$$\mathrm{JS}(P, Q) =$$

$$\log 2 + \max_f \mathop{\mathbb{E}}_{X,Y} Y \log f(X) + (1 - Y) \log(1 - f(X)) =$$

$$\log 2 - \min_f \mathcal{L}(f \mid P, Q)$$

where $\mathcal{L}$ — cross-entropy loss.

Approximating JS distance

$$\arg\min_{P} \mathrm{JS}(P, Q) = \arg\max_{P} \left[\min_{f} \mathcal{L}(f \mid P, Q) \right]$$

Generative Adversarial Networks

GAN makes no assumptions about nature of P:

- the most popular choice is via a **generator** g:

$$\begin{aligned} Z &\sim \mathcal{N}^m(0,1); \\ X &= g(Z). \end{aligned}$$

Discriminator

Minimization of $\mathcal{L}(f \mid P, Q)$ is a classical classification problem:

- f is often defined by a neural network — **discriminator**;
- Q is defined by given dataset;
- P is defined by the generator.

Discriminator

Algorithm 1 Disriminator Training

while not enough **do**

sample x from the dataset;

sample latent variables z from $\mathcal{N}^m(0, 1)$;

$$\theta \leftarrow \theta + \lambda_\theta \nabla_\theta \left[\log f_\theta(x) + \log\left(1 - f_\theta(g_\psi(z))\right)\right]$$

end while

- θ — parameters of the discriminator f_θ;
- ψ — parameters of the generator g_ψ;
- λ_θ — SGD learning rate.

Generator training

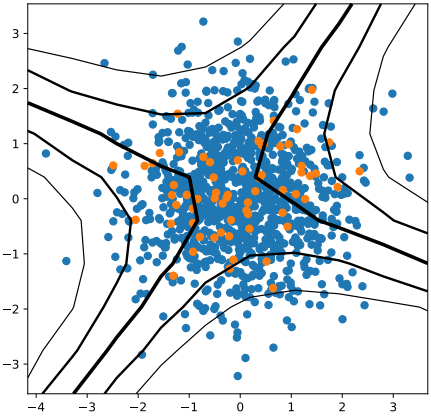
Generator is often trained by gradient methods, using:

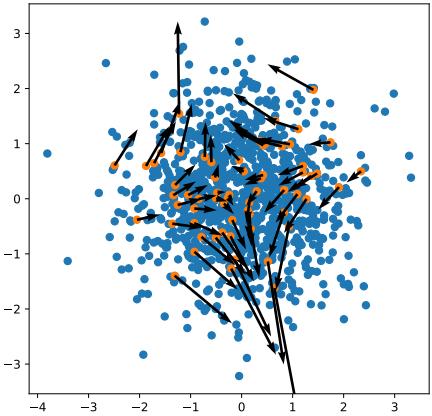
$$\Delta\psi \propto \nabla_\psi \mathop{\mathbb{E}}_{Z} \log(1 - f(g_\psi(Z)))\Big|_{f=f^*}$$

as subderivative, where:

- $f^* = \arg\min_f \mathcal{L}(f^* \mid P_\psi, Q)$.

Generator training



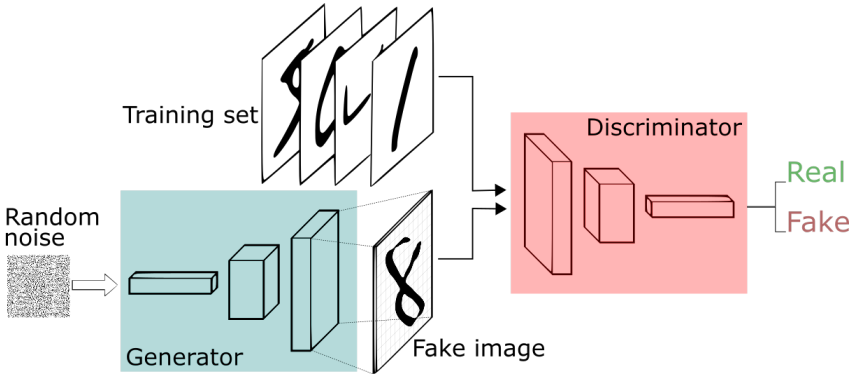


Algorithm 2 Generative Adversarial Training

while not enough **do**
  for $i := 1, \ldots, n$ **do**
    sample x from the dataset;
    sample latent variables z from $\mathcal{N}^m(0, 1)$;
    $\theta \leftarrow \theta + \lambda_\theta \nabla_\theta \left[\log f_\theta(x) + \log\left(1 - f_\theta(g_\psi(z))\right)\right]$
  end for

  sample latent variables z from $\mathcal{N}^m(0, 1)$;
  $\psi \leftarrow \psi - \lambda_\psi \nabla_\psi \left[\log\left(1 - f_\theta(g_\psi(z))\right)\right]$
end while

Generative Adversarial Networks



Source: https://sthalles.github.io/assets/dcgan/GANs.png

Game interpretation

$$\mathcal{L}(\theta, \psi) = -\frac{1}{2}\left[\mathop{\mathbb{E}}_{X\sim Q} \log f_\theta(X) + \mathop{\mathbb{E}}_{Z\sim Z} \log\left(1 - f_\theta(g_\psi(Z))\right)\right]$$

Min-max game:

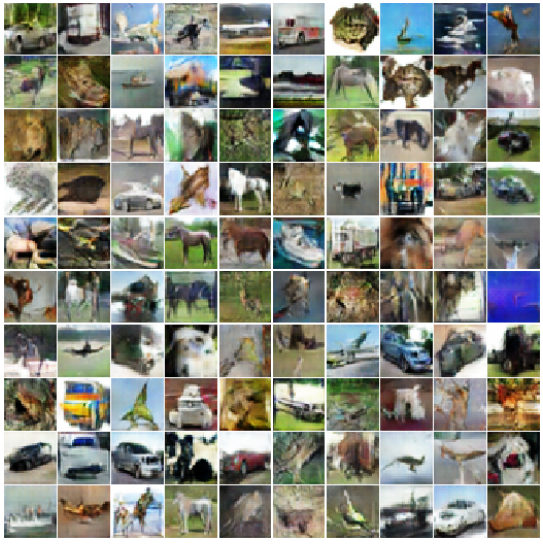
- goal of discriminator: distinguish between real and generated samples:

$$\mathcal{L}(\theta, \psi) \to_\theta \min$$

- goal of generator: 'fool' discriminator:

$$\mathcal{L}(\theta, \psi) \to_\psi \max$$

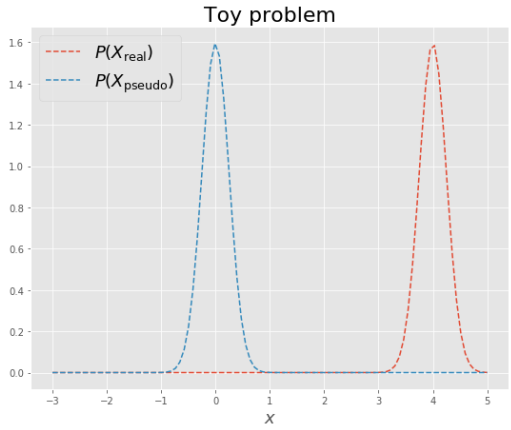
CIFAR examples



Vanishing gradients

Consider toy problem:

- powerfull discriminator;
- (almost) disjoint supports:
  - unlucky initial guess;
  - target data is on low-dimensional manifold;



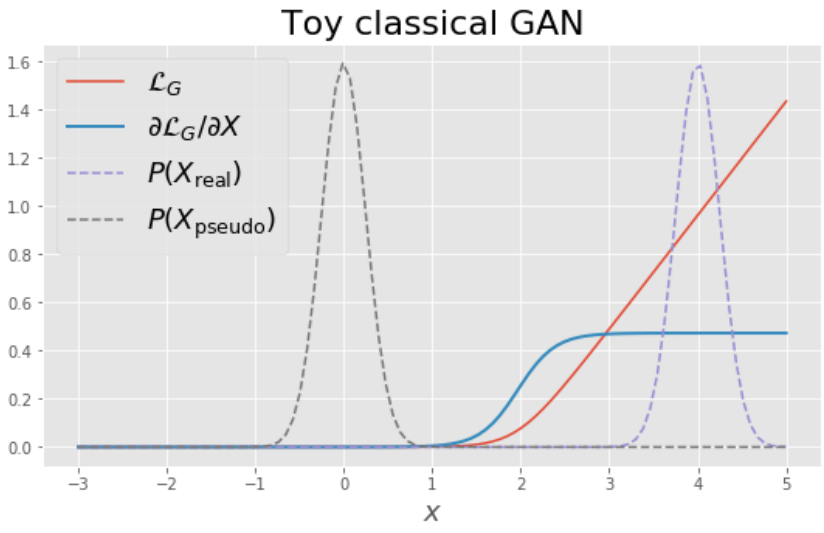
Vanishing gradients

After training discriminator:

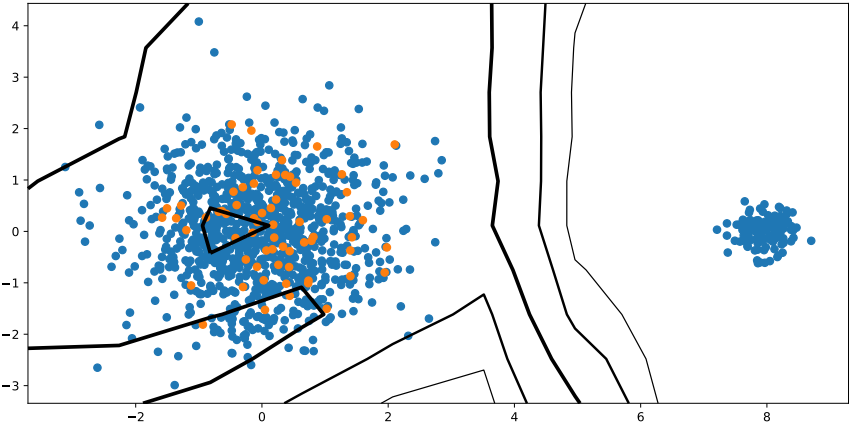
$$\begin{aligned}
\frac{\partial \mathcal{L}(\theta, \psi)}{\partial \psi} &= -\frac{1}{1 - f(g(z))} \cdot \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial \psi}; \\
f(g(z)) &\approx 0; \\
\frac{f}{\partial g} &\approx 0.
\end{aligned}$$

$\Rightarrow$ gradients tend to vanish on early stages.

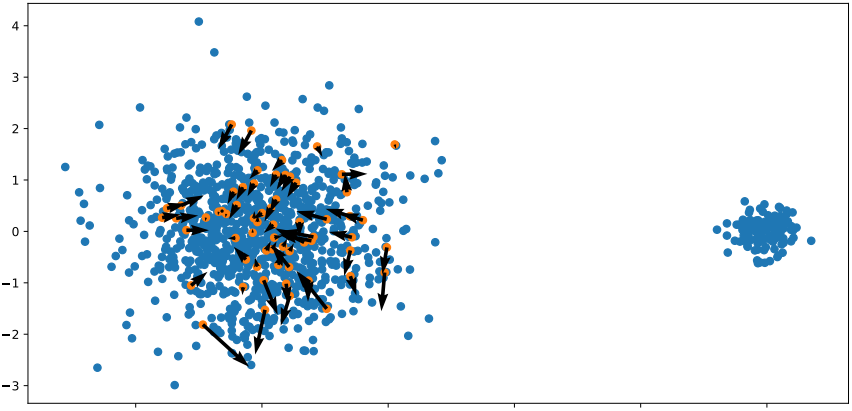
Vanishing gradients



Mode collapse



Mode collapse



GAN training tricks

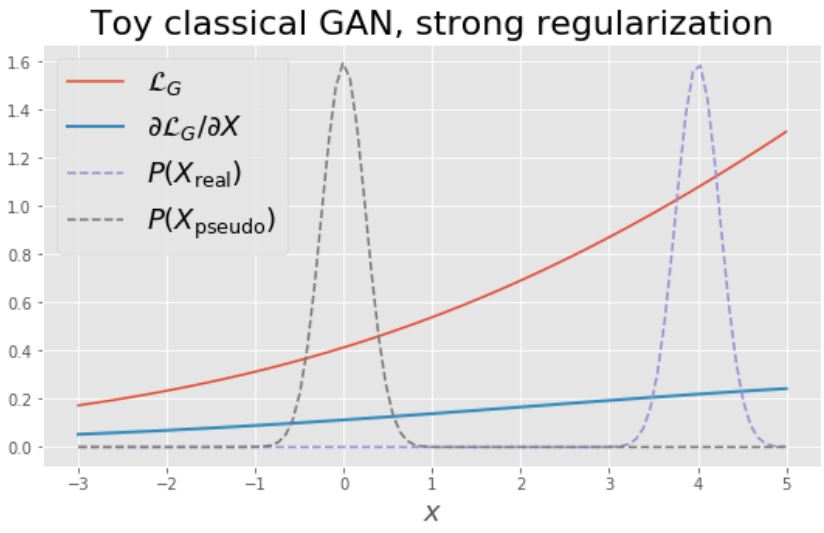
Fight for the gradients

Start with heavily restricted discriminator:

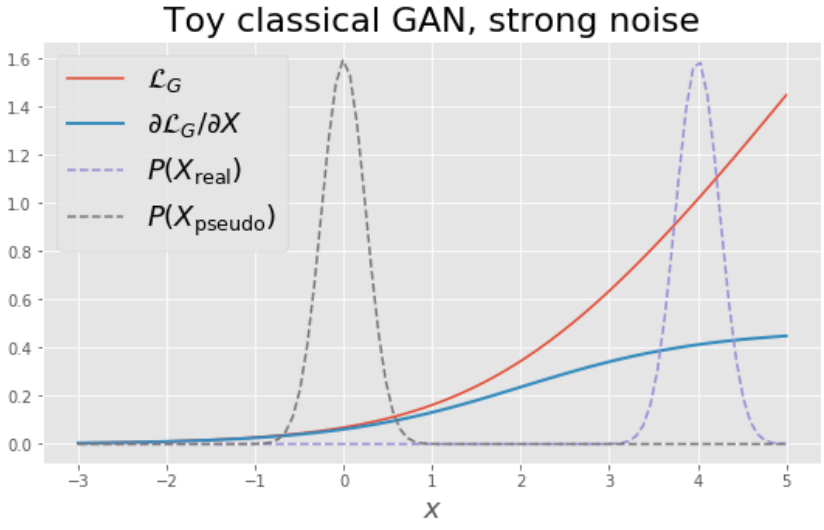
- don't train discriminator fully:
  - poor-man solution;
- add noise to the samples:
  - nicely works for target on low-dimensional manifolds;
  - easy to control.
- heavy regularization:
  - might interfere with the convergence.

As learning progresses gradually relax restrictions.

Fight for the gradients



Fight for the gradients



Generator collapse

Often generator learns to output constant or just few values. This is a syndrome of poorly trained discriminator:

- generator aims to maximize discriminator loss;
- discriminator does not adapt quickly enough;
- generator collapses into a current maxima of discriminator.

Feature matching

Let h be some **feature**, then feature matching is an auxiliary objective:

$$\mathcal{L}_{\text{fm}} = \| \mathop{\mathbb{E}}_{X \sim \text{data}} h(X) - \mathop{\mathbb{E}}_{Z} h(g(Z)) \|^2$$

Alternatively, adversarial objective might be used as well:

- just add feature h to the discriminator input;
- use a separate (simple) discriminator.

Summary

Summary

Kernel Density Estimation:

- good for 1-2 dimensional problems;
- might be computationally expensive;
- explicit probability density;

Gaussian Mixture Models:

- similar to KDE;
- explicit probability density;

Summary

Variational Auto-Encoder:

- a powerful generative model;
- easy to train;

Generative Adversarial:

- a powerful generative model;
- hard to train;
- a huge number of modifications:
  - Wasserstein-GAN solves problem of vanishing gradients;
  - BiGAN, ALI add inference;
  - CycleGAN allows to learn transformation between two unpaired sets;
  - and many more.

References I

- Bengio Y. Learning deep architectures for AI. Foundations and trends® in Machine Learning. 2009 Nov 15;2(1):1-27.
- Hinton GE, Osindero S, Teh YW. A fast learning algorithm for deep belief nets. Neural computation. 2006 Jul;18(7):1527-54.
- Nair V, Hinton GE. Rectified linear units improve restricted boltzmann machines. InProceedings of the 27th international conference on machine learning (ICML-10) 2010 (pp. 807-814).

References II

- Hinton G. A practical guide to training restricted Boltzmann machines. Momentum. 2010 Aug 2;9(1):926.
- Tieleman T. Training restricted Boltzmann machines using approximations to the likelihood gradient. InProceedings of the 25th international conference on Machine learning 2008 Jul 5 (pp. 1064-1071). ACM.