CS4243 : Computer Vision and Pattern Recognition

Lecture 15 : Generative Models
VAE and GAN

Semester 2 2021/22

Xavier Bresson

https://twitter.com/xbresson

Department of Computer Science
National University of Singapore (NUS)

Outline

- Variational autoencoders (VAE)
- Lab on VAE
- Generative Adversarial Networks (GAN)
- Labs on GAN
- Conclusion

Outline

- Variational autoencoders (VAE)
- Lab on VAE
- Generative Adversarial Networks (GAN)
- Labs on GAN
- Conclusion

Autoencoders

- Deterministic systems that learn to represent an image x with a low-dim latent vector z.
- Autoencoder cannot generate new images (a small change of z produces bad images).

$$x \quad \rightarrow \quad z \quad \rightarrow \quad \hat{x} \approx x$$

$$\begin{bmatrix} \\ \end{bmatrix}$$

Variational autoencoders (VAE)

- Upgrade autoencoders with a probabilistic mechanism that produces new images from the distribution p(x) of training images.
- VAE learns probability of image x conditioned by a low-dim latent representation z.

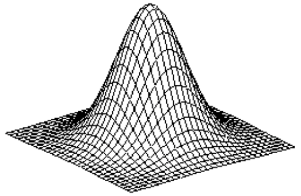
$$x \sim p(x) \;\rightarrow\; z \sim p(z|x) \;\rightarrow\; \hat{x} \sim p(x|z)$$

Sampling operator x is generated by p(x)

Probability of (training) images (what we want to estimate)

Conditional probability of latent representation(s) z given the image x

Conditional probability of image(s) x given the latent representation z



Kingma, Welling, "Auto-encoding variational bayes", 2013

Train VAE and generate new data

- During training :

  (1) Enforce conditional probability p(z|x) to become an unconditional probability p(z) (s.a. unit Normal distribution) independent of images x :

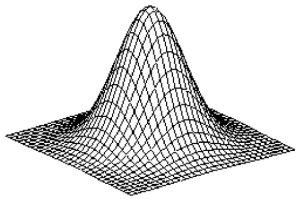
$$p(z|x) \approx p(z)$$

  (2) Impose conditional probability p(x|z) to learn the distribution of training images p(x) :

$$p(x|z) \approx p(x) \quad \text{for} \quad z \sim p(z|x)$$

- After training : Sample first a vector z from the prior distribution p(z), then sample a new x from the conditional probability p(x|z).

$$z \sim p(z) \;\rightarrow\; \hat{x} \sim p(x|z)$$



Training

Learn the probability distribution p(x) where

- x is a training set of images.
- x is represented by low-dimensional latent vector(s) z.

Bayes' theorem :

$$p(x) = \frac{p(x|z)p(z)}{p(z|x)} \quad \text{for a given pair } (x, z)$$

For all z representing image x :

(using log-probability for numerical stability and speed)

$$\log p(x) = \int_z p(z|x) \log \frac{p(x|z)p(z)}{p(z|x)} \, dz$$

$$= \mathbb{E}_{z \sim p(z|x)} \left(\log \frac{p(x|z)p(z)}{p(z|x)} \right)$$

Intractability

- It is not possible to directly estimate (log) p(x) because for each training image x, its latent representation(s) z is unobserved/unknown.
- We need to introduce a surrogate probability q(z|x) that will estimate z given x.
- We will approximate the probability p(x|z).

Approximation

- q(z|x) and p(x|z) are approximated with parametric functions represented by two networks.

$$q(z|x) \rightarrow q_\phi(z|x) \quad \text{Encoder network}$$
$$p(x|z) \rightarrow p_\theta(x|z) \quad \text{Decoder network}$$

$$x \xrightarrow[\text{Encoder network}]{z \sim q_\phi(z|x)} z \xrightarrow[\text{Decoder network}]{\hat{x} \sim p_\theta(x|z)} \hat{x}$$

Computing the likelihood probability $p_{\theta,\Phi}(x)$:

$$\log p_{\theta,\phi}(x) = \int_z q_\phi(z|x) \ \log \frac{p_\theta(x|z)p(z)}{p(z|x)} \cdot \frac{q_\phi(z|x)}{q_\phi(z|x)} \ dz$$

$$= \int_z q_\phi(z|x) \ \log p_\theta(x|z) \ dz - \int_z q_\phi(z|x) \ \log \frac{q_\phi(z|x)}{p(z)} \ dz + \int_z q_\phi(z|x) \ \log \frac{q_\phi(z|x)}{p(z|x)} \ dz$$

$$= \mathbb{E}_{z\sim q_\phi(z|x)}\big(\log p_\theta(x|z)\big) - \mathrm{KL}\big(q_\phi(z|x), p(z)\big) + \mathrm{KL}\big(q_\phi(z|x), p(z|x)\big)$$

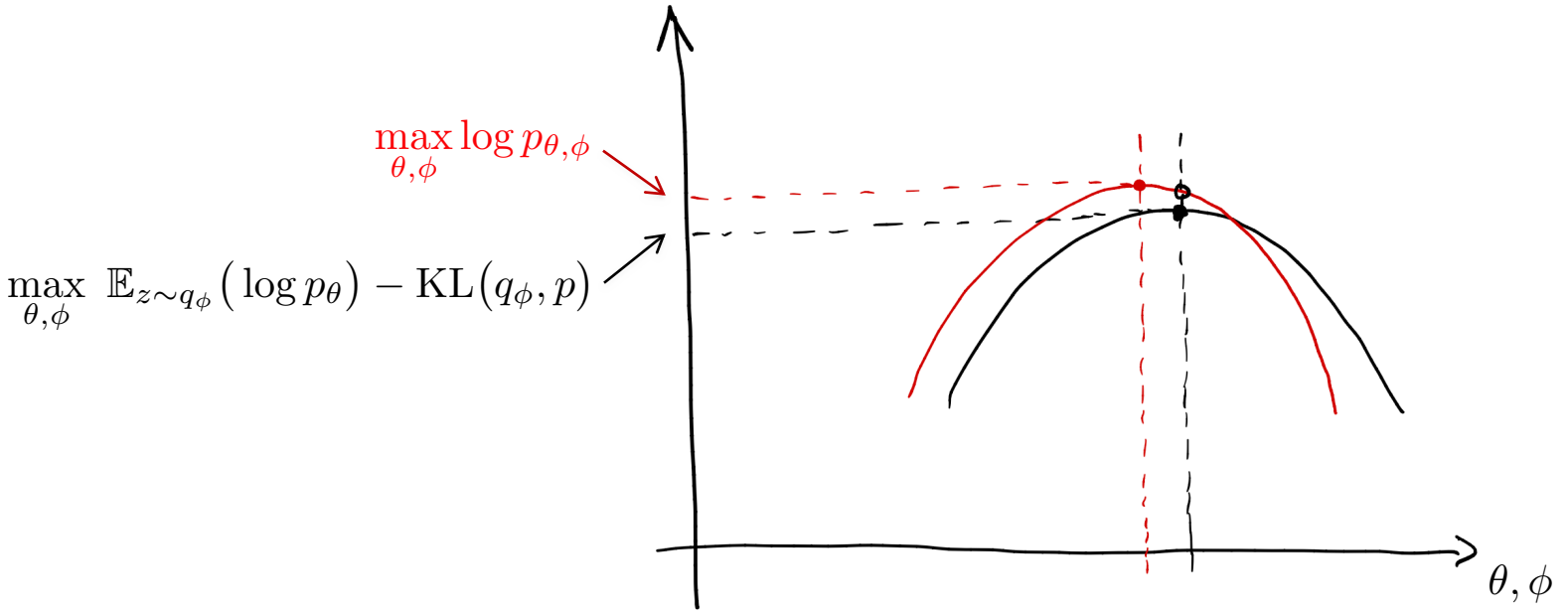
$$\text{with } \mathrm{KL}(q,p) = \mathbb{E}_{z\sim q}\Big(\log \frac{q(z)}{p(z)}\Big) = \int_z q(z) \ \log \frac{q(z)}{p(z)} \ dz$$

Maximizing the log-likelihood probability $p_{\theta,\Phi}(x)$ is not directly possible because $p(z|x)$ is unknown.
Instead, we will maximize a lower bound (which hopefully is close to the original solution).

$$\log p_{\theta,\phi}(x) = \mathbb{E}_{z\sim q_\phi(z|x)}\big(\log p_\theta(x|z)\big) - \mathrm{KL}\big(q_\phi(z|x), p(z)\big) + \underbrace{\mathrm{KL}\big(q_\phi(z|x), p(z|x)\big)}_{\geq 0}$$

$$\geq \mathbb{E}_{z\sim q_\phi(z|x)}\big(\log p_\theta(x|z)\big) - \mathrm{KL}\big(q_\phi(z|x), p(z)\big)$$

Unknown probability



Training VAE

Jointly train encoder $q_\Phi(z|x)$ and decoder $p_\theta(x|z)$ by maximizing the lower bound on the data likelihood :

$$\max_{\theta,\phi} \log p_{\theta,\phi}(x) \geq \max_{\theta,\phi} \mathbb{E}_{z\sim q_\phi(z|x)}\big(\log p_\theta(x|z)\big) - \mathrm{KL}\big(q_\phi(z|x), p(z)\big)$$

$$x \xrightarrow[\text{Encoder}]{z\sim q_\phi(z|x)} z \xrightarrow[\text{Decoder}]{\hat{x}\sim p_\theta(x|z)} \hat{x}$$

Encoder network

Assume that latent representation $z \sim q_\Phi(z|x)$ follows a Normal distribution.

Then $q_\Phi(z|x)$ is parametrized as : $q_\phi(z|x) = \mathcal{N}(\mu_{z|x}, \sigma_{z|x})$

where the mean and the standard deviation can be represented with MLPs :

$$\mu_{z|x} = \text{MLP}_\mu\big(\text{MLP}_x\big(x\big)\big) \in \mathbb{R}^{d_z}$$
$$\log \sigma^2_{z|x} = \text{MLP}_\sigma\big(\text{MLP}_x\big(x\big)\big) \in \mathbb{R}^{d_z}$$

> Note that the network learns $\log\sigma^2$ instead of σ because it is easier to learn a real/sign-free parameter rather than a positive one.
>
> In practice, x is n^2-dim, $\text{MLP}_x(x)$ and $\text{MLP}_{\mu/\sigma}(\text{MLP}_x(x))$ are d_z-dim.

How to sample z from $q_\Phi(z|x)$?

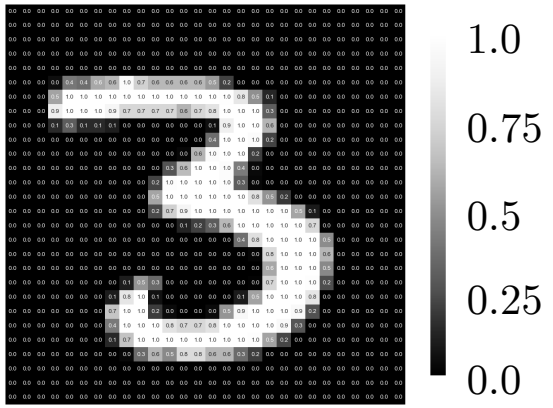
It is done with the re-parametrization trick that expresses the random variable z as a deterministic variable as follows :

$$x \in \mathbb{R}^{n^2} \quad \overset{z\sim q_\phi(z|x)}{\rightarrow} \quad z \in \mathbb{R}^{d_z} \quad \text{with } z = \mu_{z|x} + \varepsilon \odot \sigma_{z|x} \in \mathbb{R}^{d_z} \quad \text{with } \varepsilon \sim \mathcal{N}(0, I)$$
$$= \mu_{z|x} + \varepsilon \odot e^{\frac{\log \sigma^2_{z|x}}{2}} \in \mathbb{R}^{d_z}$$

Decoder network

Assume that images $x \sim p_\theta(x|z)$ follow a Bernoulli distribution (x is binary, i.e. pixel value is 0/1).

MNIST images (approximatively) follow this assumption.



We will approximate the probability $p_\theta(x|z)$ with

$$p_\theta(x|z) = \left(\Pi_{i=1}^{n^2}\ p_\theta(x_i|z)\right)^{1/n^2} \in [0,1] \quad \text{(geometric mean)}$$
$$\text{with } p_\theta(x_i|z) = \text{Sigmoid}\left(\text{MLP}_z(z)\right)_i \in [0,1]$$

where $p_\theta(x_i|z)$ corresponds to the probability that pixel i has value 1.

In practice, z is d_z-dim and $\text{MLP}_z(z)$ is n^2-dim.

Decoder network

How to sample x from $p_\theta(x|z)$?

It is done for each pixel i with Bernoulli sampling with probability value given by $p_\theta(x_i|z)$:

$$z \in \mathbb{R}^{d_z} \quad \overset{x \sim p_\theta(x|z)}{\rightarrow} \quad x = \text{Bernoulli}\big(\text{Sigmoid}\big(\text{MLP}_z\big(z\big)\big)\big) \in \{0,1\}^{n^2}$$

Bernoulli sampling is equivalent to "coin flipping" with probability p_θ (e.g. p_θ=0.5 for a real coin).

In practice, we can remove the sampling as MNIST images are not exact binary images :

$$z \in \mathbb{R}^{d_z} \quad \overset{x \sim p_\theta(x|z)}{\rightarrow} \quad x = \text{Sigmoid}\big(\text{MLP}_z\big(z\big)\big) \in [0,1]^{n^2}$$

Note : For natural images, we can assume that $x \sim p_\theta(x|z)$ follows e.g. a Normal distribution and parametrize it with $p_\theta(x|z)=\mathcal{N}(\mu_{x|z},\sigma_{x|z})$ where $\mu_{x|z},\sigma_{x|z}$ are represented with MLPs applied to z.

Auto-encoder/fidelity term

Let us consider the first term of the variational lower bound : $\max_{\theta,\phi} \mathbb{E}_{z\sim q_\phi(z|x)}\big(\log p_\theta(x|z)\big)$

with the fidelity constraint : $x \overset{z\sim q_\phi(z|x)}{\rightarrow} z \overset{x\sim p_\theta(x|z)}{\rightarrow} \hat{x} \approx x \sim p(x)$

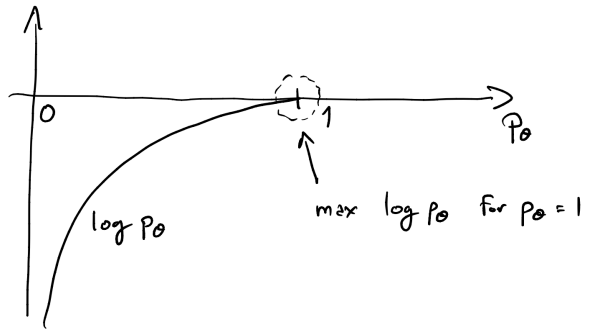
As we assumed that images x/$\hat{x}$ are binary {0,1}, then $p(x_i)/p_\theta(x_i|z)$ corresponds to the probability that pixel i has value 1 (complementary, $1\text{-}p(x_i)/1\text{-}p_\theta(x_i|z)$ is the probability that pixel i has value 0).

Then, the first term of the lower bound is equivalent to maximize the (negative) binary cross-entropy :

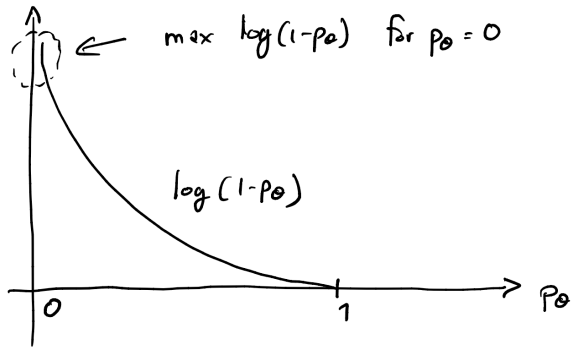
$$\max_{\theta,\phi} \mathbb{E}_{z\sim q_\phi(z|x)}\big(\log p_\theta(x|z)\big) = \frac{1}{n^2}\sum_{i=1}^{n^2}\begin{cases}\log p_\theta(x_i|z) & \text{if } p(x_i)=1 \text{ (pixel value is 1)}\\ \log(1-p_\theta(x_i|z)) & \text{if } p(x_i)=0 \text{ (pixel value is 0)}\end{cases}$$

$$= \frac{1}{n^2}\sum_{i=1}^{n^2} p(x_i)\log p_\theta(x_i|z) + (1-p(x_i))\log(1-p_\theta(x_i|z))$$

for $p(x_i)=1$



for $p(x_i)=0$



Prior distribution/regularization term

Let us consider the second term of the variational lower bound : $\max_{\phi} \; -\mathrm{KL}\big(q_\phi(z|x), p(z)\big)$

Maximizing the KL term enforces $q_\Phi(z|x) \approx p(z)$.

Assuming that the prior distribution is the unit Normal distribution : $p(z) = \mathcal{N}(0, I)$

There exists an analytical expression of the second term :

$$
\begin{aligned}
-\mathrm{KL}\big(q_\phi(z|x), p(z)\big) &= -\int_z q_\phi(z|x) \; \log \frac{q_\phi(z|x)}{p(z)} \; dz \\
&= -\int_z \mathcal{N}(\mu_{z|x}, \sigma_{z|x}) \; \log \frac{\mathcal{N}(\mu_{z|x}, \sigma_{z|x})}{\mathcal{N}(0, I)} \; dz \\
&= \frac{1}{2} \sum_{j=1}^{d_z} \Big(1 + \big(\log \sigma^2_{z|x}\big)_j - \big(\mu_{z|x}\big)^2_j - \big(\sigma^2_{z|x}\big)_j\Big)
\end{aligned}
$$

This term acts as a regularization process that “fills up” the latent space z with the training distribution.

This guarantees that any z sampled from the latent space produces a meaningful image.

Summary

Training VAE encoder and decoder by maximizing the lower bound on the data likelihood is equivalent to

$$\max_{\theta,\phi} \log p_{\theta,\phi}(x) \geq \max_{\theta,\phi} \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\big(\log p_\theta(x|z)\big)}_{} - \underbrace{\mathrm{KL}\big(q_\phi(z|x), p(z)\big)}_{}$$

Data fidelity term
It encourages the encoder and decoder to produce images from the training set.

$$p_\theta(x|z) \approx p(x) \text{ for } z \sim q_\phi(z|x)$$

Regularization term
It encourages the encoder to generate latent representation of the training images to follow a unit Normal distribution.

$$q_\phi(z|x) \approx p(z) = \mathcal{N}(0, I)$$

The Normal distribution is unconditioned p(z), which allows producing new images independently of any input x.

$$z \sim p(z) \;\rightarrow\; \hat{x} \sim p_\theta(x|z)$$

Training steps

Step 1 : Given training image x, compute a latent representation z :

$$z = \mu_{z|x} + \varepsilon \odot e^{\frac{\log \sigma^2_{z|x}}{2}} \in \mathbb{R}^{d_z} \quad \text{with } \varepsilon \sim \mathcal{N}(0, I)$$
$$\mu_{z|x} = \mathrm{MLP}_\mu\big(\mathrm{MLP}_x\big(x\big)\big) \in \mathbb{R}^{d_z}$$
$$\log \sigma^2_{z|x} = \mathrm{MLP}_\sigma\big(\mathrm{MLP}_x\big(x\big)\big) \in \mathbb{R}^{d_z}$$

Step 2 : Given latent representation z, generate the probability $p_\theta(x_i|z)$:

$$p_\theta(x_i|z) = \mathrm{sigmoid}\big(\mathrm{MLP}_z\big(z\big)\big)_i \in [0, 1]$$

Step 3 : Compute total loss :

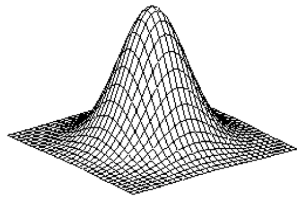
$$L = \frac{1}{n^2} \sum_{i=1}^{n^2} p(x_i) \log p_\theta(x_i|z) + (1 - p(x_i)) \log(1 - p_\theta(x_i|z)) +$$
$$\sum_{j=1}^{d_z} \Big(1 + \big(\log \sigma^2_{z|x}\big)_j - \big(\mu_{z|x}\big)^2_j - \big(\sigma^2_{z|x}\big)_j\Big)$$

Step 4 : Backpropagation

Generating steps

Step 1 : Sample z from prior distribution

$$z \in \mathbb{R}^{d_z} \sim p(z)$$



Step 2 : Generate new x

$$\hat{x} = \begin{cases} \text{Bernoulli}\big(\text{Sigmoid}\big(\text{MLP}_z\big(z\big)\big)\big) & \in \{0,1\}^{n^2} \text{ (sampling)} \\ \text{Sigmoid}\big(\text{MLP}_z\big(z\big)\big) & \in [0,1]^{n^2} \text{ (no sampling)} \end{cases}$$

Variational Autoencoders: Generating Data

32x32 CIFAR-10

Labeled Faces in the Wild

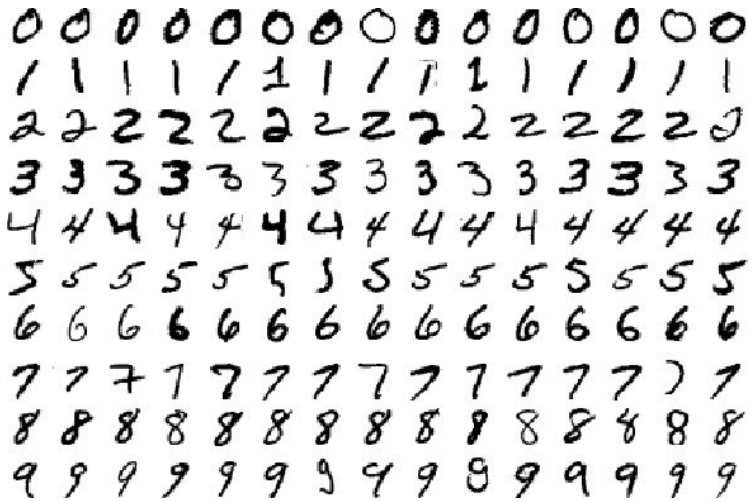
Figures from (L) Dirk Kingma et al. 2016; (R) Anders Larsen et al. 2017.

Outline

- Variational autoencoders (VAE)
- **Lab on VAE**
- Generative Adversarial Networks (GAN)
- Labs on GAN
- Conclusion

Lab 01

- Design a vanilla VAE architecture that generates new MNIST images.



jupyter VAE_exercise Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

Lab 01 : Vanilla VAE - exercise

The goal is to implement a VAE architecture with MLPs to generate new MNIST images.

In [ ]:
```
# For Google Colaboratory
import sys, os
if 'google.colab' in sys.modules:
    # mount google drive
    from google.colab import drive
    drive.mount('/content/gdrive')
    path_to_file = '/content/gdrive/My Drive/CS4243_codes/codes/labs_lecture15/lab01_VAE'
    print(path_to_file)
    # move to Google Drive directory
    os.chdir(path_to_file)
    !pwd
```

In [ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import utils
import time
```

In [ ]:
```
#device= torch.device("cuda")
device= torch.device("cpu")
print(device)
```

In [ ]:
```
# Libraries
import matplotlib.pyplot as plt
import logging
logging.getLogger().setLevel(logging.CRITICAL) # remove warnings
```

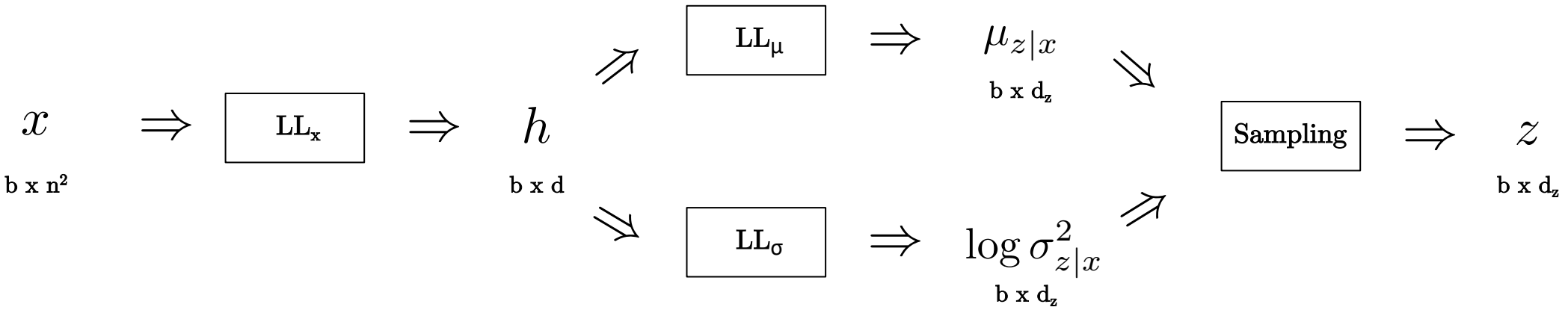
Lab 01

- Objectives
  - Implement the encoder and the decoder with MLPs.
  - Train the network.
  - Verify the network can auto-encode training data.
  - Generate new images.

Lab 01

- Architecture design
  - Encoder

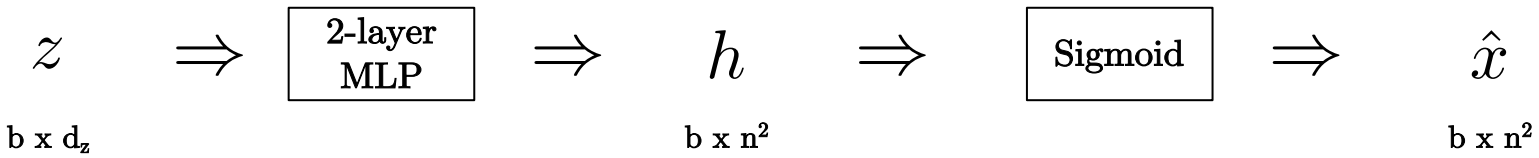
$$z = \mu_{z|x} + \varepsilon \odot e^{\frac{\log \sigma^2_{z|x}}{2}} \in \mathbb{R}^{d_z} \quad \text{with } \varepsilon \sim \mathcal{N}(0, I)$$
$$\mu_{z|x} = \text{MLP}_\mu\big(\text{MLP}_x\big(x\big)\big) \in \mathbb{R}^{d_z}$$
$$\log \sigma^2_{z|x} = \text{MLP}_\sigma\big(\text{MLP}_x\big(x\big)\big) \in \mathbb{R}^{d_z}$$



Lab 01

- Architecture design
  - Decoder

$$\hat{x} = \text{Sigmoid}\big(\text{MLP}_z(z)\big) \in [0,1]^{n^2}$$



Lab 01

- Loss

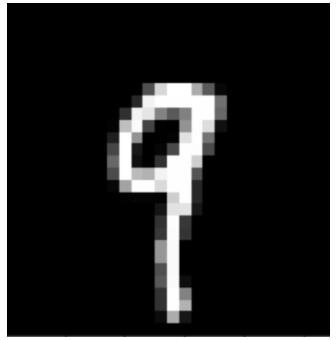
$$L = \beta \cdot \frac{1}{n^2} \sum_{i=1}^{n^2} p(x_i) \log p_\theta(x_i|z) + (1 - p(x_i)) \log(1 - p_\theta(x_i|z))$$

$$+ \frac{1}{2} \sum_{j=1}^{d} \left(1 + \left(\log \sigma^2_{z|x}\right)_j - \left(\mu_{z|x}\right)^2_j - \left(\sigma^2_{z|x}\right)_j\right), \ \beta = 10$$

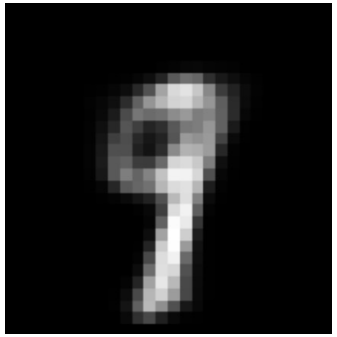
Lab 01

- Hyper-parameters : d_z=20 (latent dim), d=400 (hidden dim)
- Training : n_{epoch}=10, n_{batch}=200 (nb of batches/epoch), b=64 (batch size)
- Testing
  - Verify the network can auto-encode training data :

$$x \overset{z \sim q_\phi(z|x)}{\rightarrow} z \overset{\hat{x} \sim p_\theta(x|z)}{\rightarrow} \hat{x}$$



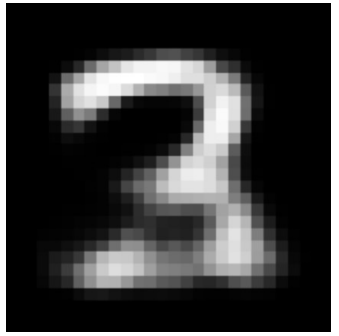
x



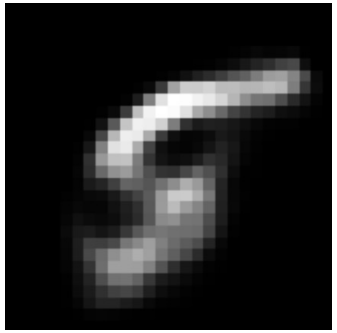
$\hat{x}$

  - Generate new images :

$$z \sim p(z) \rightarrow \hat{x} = \text{Sigmoid}\big(\text{MLP}_z(z)\big)$$



$\hat{x}$



$\hat{x}$

Outline

- Variational autoencoders (VAE)
- Lab on VAE
- Generative Adversarial Networks (GAN)
- Labs on GAN
- Conclusion

Generative Adversarial Networks (GAN)

Goal : Produce a new image x with a generator G parametrized by a latent variable z such that

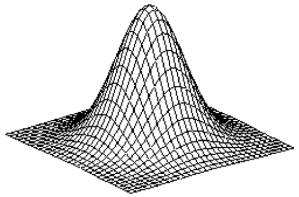
$$z \sim p_z(z) \;\rightarrow\; x = G(z) \sim p_G(x) \quad \text{s.t.} \quad p_G(x) \approx p(x)$$

z is sampled from a prior (unconditional) probability $p_z(z)$.

Generator G produces a new image sample x.

p_G is the generator probability

Probability of training/real images (what we want to estimate)



Note : Neither $p(x)$ nor $p_G(x)$ is explicitly computed with GAN.

Only samples will be observed $x_{train} \sim p(x)$ or computed $x_{new}=G(z)$, $z \sim p_z(z)$.

Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, Bengio, "Generative Adversarial Networks", 2014

Methodology

Formulate the problem as a discriminative problem between real and generated/fake images.

Fake/generated images :

$$z \sim p_z(z) \;\rightarrow\; x = G(z) \sim p_G(x) \;\rightarrow\; p_D(x) = 0$$

x is a fake image produced by the generator G.

x is classified as fake by the discriminator D.

Real images :

$$x \sim p(x) \;\rightarrow\; p_D(x) = 1$$

x is a true/training image, which is sampled from the image probability p(x) (the probability we want to estimate).

x is classified as true by the discriminator D.

Unknown G and D

The generator G and the discriminator D are unobserved.

Approximation : We approximate G and D/p_D with parametric functions represented by two networks.

$$G \rightarrow G_\theta \quad \text{Generator network}$$
$$p_D \rightarrow p_{D_\phi} \quad \text{Discriminator probability network}$$

Estimation of G_θ and D_Φ :

We cast the discriminative problem as an adversarial optimization problem where

(1) The discriminator D_Φ aims at distinguishing real and false/generated images.

(2) The generator G_θ aims at fooling the discriminator D_Φ with fake/generated images classified as real.

Note : "D_Φ discriminator" will be used interchangeably with "discriminator probability $p_{D\Phi}$".

Training the discriminator D_Φ (generator G_θ is fixed)

(i) For real images $x_r \sim p(x)$, we want the probability of the discriminator D_Φ to be 1, i.e. $p_{D\Phi}(x)=1$ for $x = x_r$.

(ii) For fake/generated images $x_f=G_\theta(z)$, $z \sim p_z(z)$, we want the probability of the discriminator D_Φ to be 0, i.e. $p_{D\Phi}(x)=0$ for $x=x_f$.

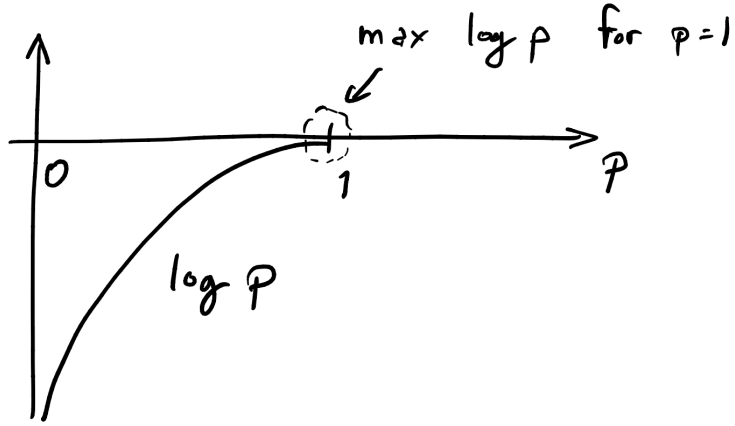
Loss(i) :

$$\max_\phi \sum_{x_r} \log p_{D_\phi}(x_r)$$

$$\max_\phi \int_x p(x) \log p_{D_\phi}(x)\, dx \quad \text{with } p(x) = \begin{cases} 1 \text{ for } x = x_r \text{ (real images)} \\ 0 \text{ otherwise} \end{cases}$$

$$\max_\phi \mathbb{E}_{x\sim p(x)} \Big(\log p_{D_\phi}(x) \Big)$$

This optimization term enforces $p_{D\Phi}(x)=1$ for $x=x_r$ because



Training the discriminator D_Φ (generator G_θ is fixed)

(i) For real images $x_r \sim p(x)$, we want the probability of the discriminator D_Φ to be 1, i.e. $p_{D\Phi}(x)=1$ for $x = x_r$.

(ii) For fake/generated images $x_f=G_\theta(z)$, $z \sim p_z(z)$, we want the probability of the discriminator D_Φ to be 0, i.e. $p_{D\Phi}(x)=0$ for $x=x_f$.

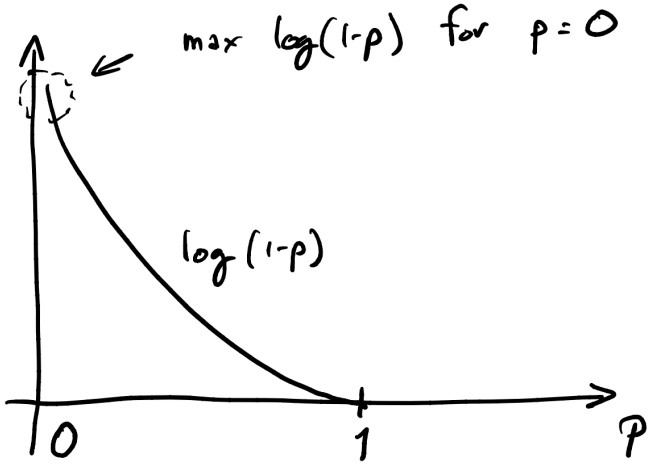
Loss(ii) :

$$\max_\phi \sum_{x_f} \log\left(1 - p_{D_\phi}(x_f)\right), \quad x_f \text{ are unobserved}$$

$$\max_\phi \int_z p_z(z) \log\left(1 - p_{D_\phi}(G_\theta(z))\right) dz, \quad x_f \text{ are approximated with } x_f = G_\theta(z)$$

$$\max_\phi \mathbb{E}_{z\sim p_z(z)}\Big(\log\left(1 - p_{D_\phi}(G_\theta(z))\right)\Big)$$

This optimization term enforces $p_{D\Phi}(x)=0$ for $x=x_f$ because



Summary of the discriminator D_Φ optimization problem (the generator G_θ is fixed) :

$$\max_{\phi} \; \mathbb{E}_{x \sim p(x)} \Big(\log p_{D_\phi}(x) \Big) + \mathbb{E}_{z \sim p_z(z)} \Big(\log \big(1 - p_{D_\phi}(G_\theta(z)) \big) \Big)$$

Solution :

$$p_{D_\phi}(x) = \begin{cases} 1 \text{ for } x = x_r \text{ (real images)} \\ 0 \text{ for } x = x_f \text{ (fake/generated images)} \end{cases}$$

Training the generator G_θ (discriminator D_Φ is fixed)

For fake/generated images $x_f = G_\theta(z)$, $z \sim p_z(z)$, we want the probability of the discriminator D_Φ to be 1, i.e. $p_{D\Phi}(x)=1$ for $x=x_f$.

Equivalently, we want the generator G_θ to be able to "fool" the discriminator D_Φ to predict that the fake images are true! If the generator suceeds, then the problem is solved as the distribution of generated images cannot be distinguished from the distribution of the training images.

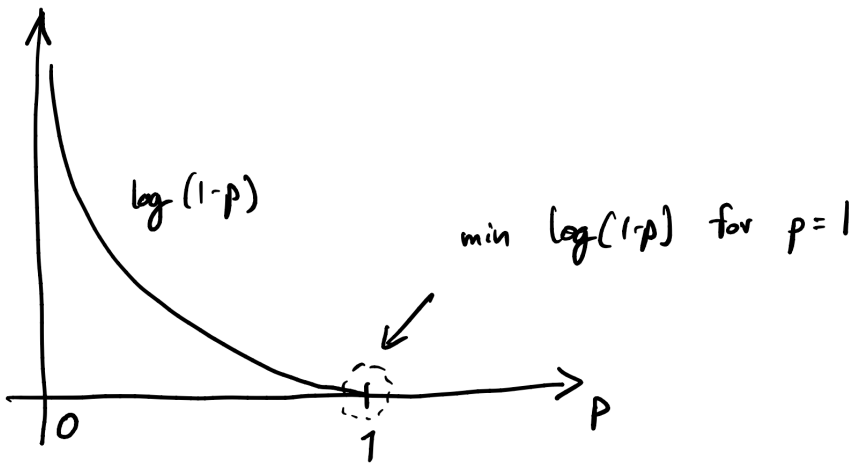
Loss :

$$\min_\theta \sum_{x_f} \log\left(1 - p_{D_\phi}(x_f)\right), \quad x_f \text{ are unobserved but approximated with } x_f = G_\theta(z)$$

$$\min_\theta \int_z p_z(z) \log\left(1 - p_{D_\phi}(G_\theta(z))\right) dz$$

$$\min_\theta \mathbb{E}_{z \sim p_z(z)} \Big(\log\left(1 - p_{D_\phi}(G_\theta(z))\right) \Big)$$

This optimization term enforces $p_{D\Phi}(x)=1$ for $x=x_f$ because



Final optimization problem

$$\min_{\theta} \max_{\phi} \mathbb{E}_{x\sim p(x)}\Big(\log p_{D_\phi}(x)\Big) + \mathbb{E}_{z\sim p_z(z)}\Big(\log\big(1 - p_{D_\phi}(G_\theta(z))\big)\Big)$$

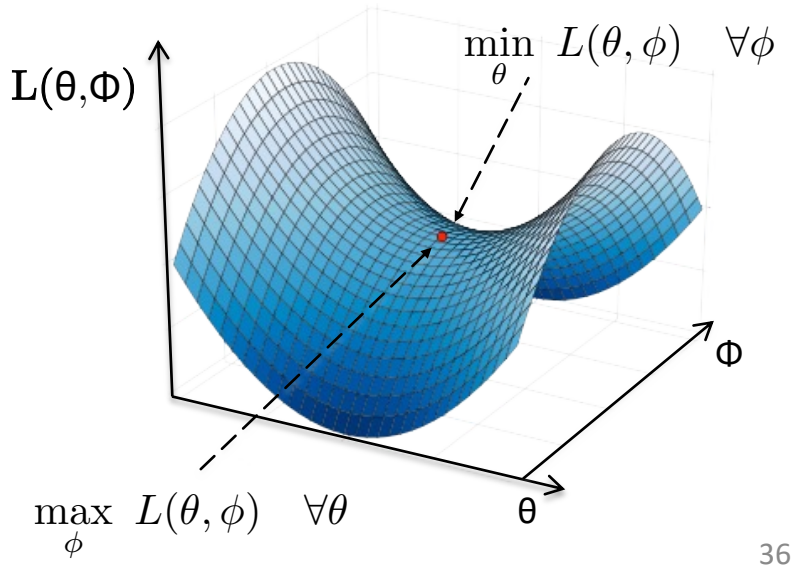
Solution of the max problem w.r.t. the discriminator D_Φ (G_θ fixed) :

$$p_{D_\phi}(x) = \begin{cases} 1 \text{ for } x = x_r \text{ (real images)} \\ 0 \text{ for } x = x_f \text{ (fake/generated images)} \end{cases}$$

Solution of the min problem w.r.t. the generator G_θ (D_Φ fixed) :

$$p_{D_\phi}(x) = 1 \text{ for } x = x_f \text{ (fake/generated images)}$$

Simultaneously enforcing "$p_D(x)=1$ for $x=x_f$" to estimate the generator G_θ and "$p_D(x)=0$ for $x=x_f$" to estimate the discriminator D_Φ lead to an adversarial problem or minimax game where the optimal solution is called Nash equilibrium in Game Theory.



Optimality

What is the solution of the GAN optimization problem?

$$\min_{\theta} \max_{\phi} \mathbb{E}_{x \sim p(x)} \Big(\log p_{D_\phi}(x) \Big) + \mathbb{E}_{z \sim p_z(z)} \Big(\log \big(1 - p_{D_\phi}(G_\theta(z)) \big) \Big)$$

The solution (Nash equilibrium) is given for :

$$p_{G_\theta}(x) = p(x) \qquad \text{Optimal generator for any discriminator}$$

$$p_{D_\phi}(x) = \frac{p(x)}{p(x) + p_{G_\theta}(x)} = \frac{1}{2} \qquad \text{Optimal discriminator for any generator}$$

Goodfellow et al, "Generative Adversarial Networks", NeurIPS 2014

Properties

- The GAN minimax problem may not have a solution.
- If a solution exists, then the optimal generator G_θ captures the statistics of real images :

$$p_{G_\theta}(x) = p(x)$$

and new images can be generated as :

$$z \sim p_z(z) \;\; \rightarrow \;\; x = G_\theta(z) \sim p_{G_\theta}(x) \approx p(x)$$

- SGD is not guaranteed to find a solution even if it exists.
- SGD may not converge to a steady state for G_θ and D_Φ.
- Networks G_θ and D_Φ are not guaranteed to represent the solutions G and D if they exist (mode collapsing problem).
- Training loss $L(\theta,\Phi)$ does not simply decrease (network G_θ minimizes L and network D_Φ maximizes L).

Training steps

Step 1 : Architecture design

Generator G_θ : MLP architecture

$$G_\theta(z) = \text{MLP}_G\big(z\big) \in \mathbb{R}^{n \times n},\ z \in \mathbb{R}^{d_z}$$

Discriminator D_Φ / $p_{D\Phi}$: MLP architecture

$$p_{D_\phi}(x) = \text{Sigmoid}\Big(\text{MLP}_D\big(x\big)\Big) \in [0,1],\ x \in \mathbb{R}^{n \times n}$$

$\text{MLP}_G(z)$ is a n^2-dim vector (reshaped to n x n image) and $\text{MLP}_D(x)$ is a scalar (score).

CNN architectures are used in practice.

Training steps

Step 2 : Update generator G_θ (D_Φ fixed)

Batch : b fake samples

Sample : $z \sim p_z(z) = \mathcal{N}/\text{Unif} \in \mathbb{R}^{b \times d_z}$

Generate : $x_{\text{fake}} = G_\theta(z) \in \mathbb{R}^{b \times n \times n}$

Classify : $p_{D_\phi}(x_{\text{fake}}) \in [0,1]^b$

Loss : $L_G = \frac{1}{B} \sum_{i=1}^{b} \log\left(1 - p_{D_\phi}(x_{\text{fake}})\right)$

Backpropagation : $\theta \leftarrow \theta - \text{lr} \frac{\partial L_G}{\partial \theta}$

Training steps

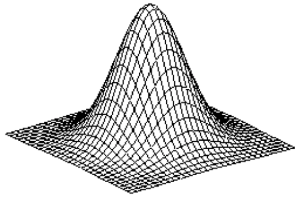
Step 3 : Update discriminator D_Φ (G_θ fixed)

Batch :	b real and fake samples
Sample :	$z \sim p_z(z) = \mathcal{N}/\text{Unif} \in \mathbb{R}^{b \times d_z}$
Generate :	$x_{\text{fake}} = G_\theta(z) \in \mathbb{R}^{b \times n \times n}$
Sample :	$x_{\text{real}} \sim \text{training set} \in \mathbb{R}^{b \times n \times n}$
Classify :	$p_{D_\phi}(x_{\text{fake}}) \in [0,1]^b,\ p_{D_\phi}(x_{\text{real}}) \in [0,1]^b$
Loss :	$L_D = \frac{1}{b}\sum_{i=1}^{b} \log p_{D_\phi}(x_{\text{real}}) + \frac{1}{b}\sum_{i=1}^{b} \log\left(1 - p_{D_\phi}(x_{\text{fake}})\right)$
Backpropagation :	$\phi \leftarrow \phi + \text{lr}\frac{\partial L_D}{\partial \phi}$

Generating steps

Step 1 : Sample z from prior distribution

$$z \sim p_z(z) = \mathcal{N}/\text{Unif} \in \mathbb{R}^{d_z}$$

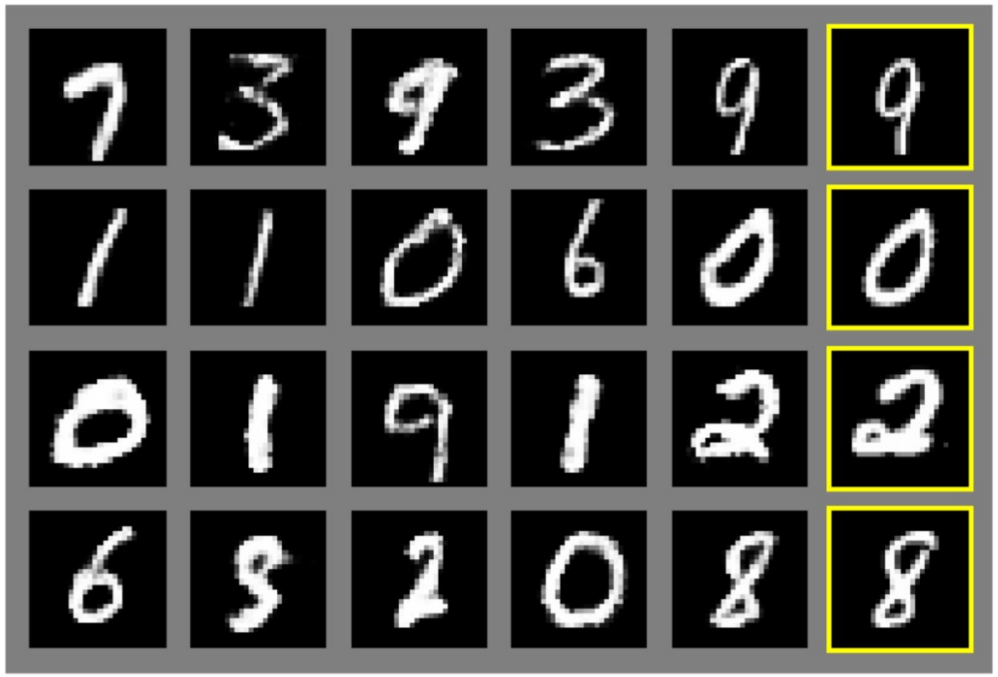


Step 2 : Generate new x

$$x_{\text{new}} = G_\theta(z) \in \mathbb{R}^{n \times n}$$

Generative Adversarial Networks: Results

Generated samples



Nearest neighbor from training set

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

Generative Adversarial Networks: DC-GAN

Samples from the model look much better!

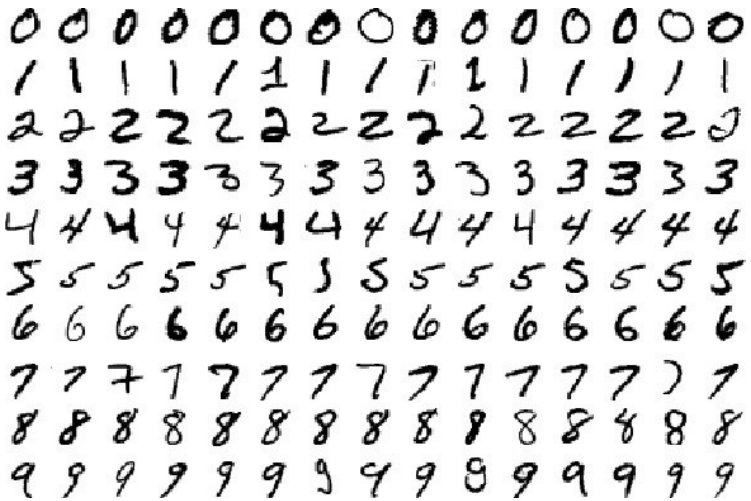
Radford et al, ICLR 2016

Outline

- Variational autoencoders (VAE)
- Lab on VAE
- Generative Adversarial Networks (GAN)
- **Labs on GAN**
- Conclusion

Lab 02

- Design a vanilla GAN architecture that generates new MNIST images.



jupyter GAN_MLP_exercise Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

Lab 02 : Vanilla GAN with MLP - exercise

The goal is to implement a GAN architecture with MLPs to generate new MNIST images.

In [ ]:
```
# For Google Colaboratory
import sys, os
if 'google.colab' in sys.modules:
    # mount google drive
    from google.colab import drive
    drive.mount('/content/gdrive')
    path_to_file = '/content/gdrive/My Drive/CS4243_codes/codes/labs_lecture15/lab02_GAN_MLP'
    print(path_to_file)
    # move to Google Drive directory
    os.chdir(path_to_file)
    !pwd
```

In [ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import utils
import time
```

In [ ]:
```
device= torch.device("cuda")
#device= torch.device("cpu")
print(device)
```

In [ ]:
```
# Libraries
import matplotlib.pyplot as plt
import logging
logging.getLogger().setLevel(logging.CRITICAL) # remove warnings
```

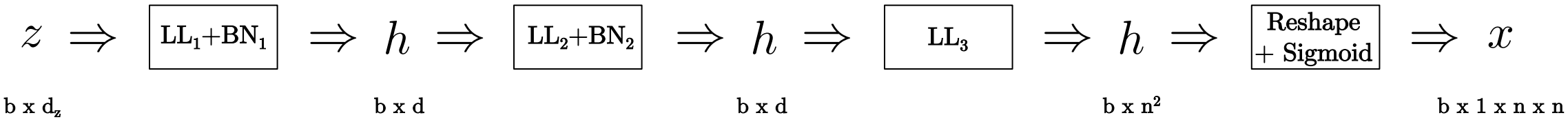
Lab 02

- Objectives
  - Implement the generator and the discriminator with MLPs.
  - Train the network.
  - Generate new images.

Lab 02

- Architecture design
  - Generator :

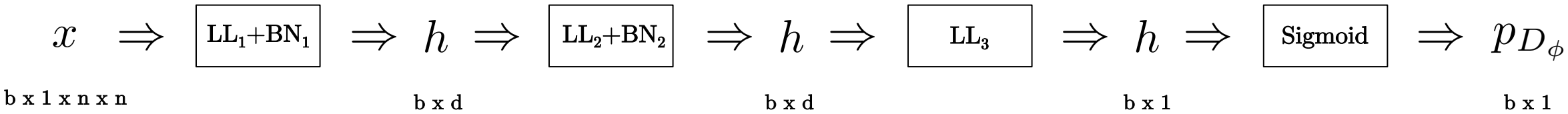
$$x = G_\theta(z) \in \mathbb{R}^{b \times n \times n}$$



Lab 02

- Architecture design
  - Discriminator :

$$p_{D_\phi}(x) \in [0,1]^b$$



Lab 02

- Loss :

$$\max_{\phi} \min_{\theta} \mathbb{E}_{x\sim p(x)}\Big(\log p_{D_\phi}(x)\Big) + \mathbb{E}_{z\sim p_z(z)}\Big(\log\big(1 - p_{D_\phi}(G_\theta(z))\big)\Big)$$

- Alternate optimization :

$$\max_{\phi} \mathbb{E}_{x\sim p(x)}\Big(\log p_{D_\phi}(x)\Big) + \mathbb{E}_{z\sim p_z(z)}\Big(\log\big(1 - p_{D_\phi}(G_\theta(z))\big)\Big)$$

$$\min_{\theta} \mathbb{E}_{z\sim p_z(z)}\Big(\log\big(1 - p_{D_\phi}(G_\theta(z))\big)\Big) \Leftrightarrow \min_{\theta} \mathbb{E}_{z\sim p_z(z)}\Big(-\log p_{D_\phi}(G_\theta(z)\Big)$$

(Next slide)

Lab 02

- Optimization trick for early steps

  When optimization starts, the generator is bad at producing fake images to fool the discriminator, that is $p_{D\Phi}(x_f)=0$.

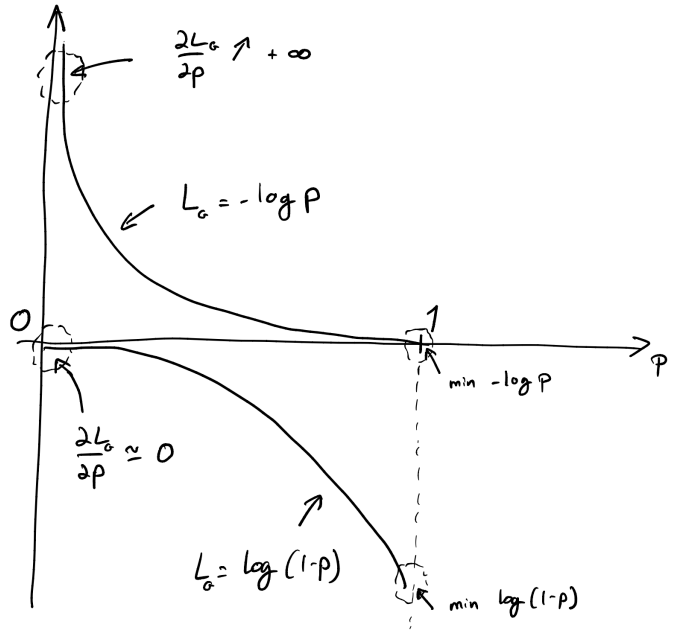
  For $p_{D\Phi}$ close to zero, the gradient of the generator loss L_G is close to zero.

  Solution : Change the minimization problem w.r.t. the generator G_θ :

$$\min_\theta \ \mathbb{E}_{z\sim p_z(z)}\Big(\log\big(1-p_{D_\phi}(G_\theta(z))\big)\Big)$$

  with this one :

$$\min_\theta \ \mathbb{E}_{z\sim p_z(z)}\Big(-\log\big(p_{D_\phi}(G_\theta(z))\big)\Big)$$



Lab 02

- Optimization w.r.t. D_Φ :

$$\max_{\phi} \; \mathbb{E}_{x\sim p(x)}\Big(\log p_{D_\phi}(x)\Big) + \mathbb{E}_{z\sim p_z(z)}\Big(\log\big(1-p_{D_\phi}(G_\theta(z))\big)\Big)$$

- PyTorch BCE (binary cross-entropy) loss :

$$\begin{aligned}\text{BCELoss}(p,\hat{p}) &= -\sum_i \hat{p}_i \log p_i + (1-\hat{p}_i)\log(1-p_i)\\ &= -\sum_i \log p_i \quad \text{if } \hat{p}_i = 1\\ &= -\sum_i \log(1-p_i) \quad \text{if } \hat{p}_i = 0\end{aligned}$$

- Implementation :

$$\max_{\phi} \; \mathbb{E}_{x\sim p(x)}\Big(\log p_{D_\phi}(x)\Big) \;\Leftrightarrow\; \min_{\phi} \; \text{BCELoss}(p_{D_\phi},\hat{p}=1)$$

$$\max_{\phi} \; \mathbb{E}_{z\sim p_z(z)}\Big(\log\big(1-p_{D_\phi}(G_\theta(z))\big)\Big) \;\Leftrightarrow\; \min_{\phi} \; \text{BCELoss}(p_{D_\phi},\hat{p}=0)$$

Lab 02

- Optimization w.r.t. G_θ :

$$\min_\theta \ \mathbb{E}_{z\sim p_z(z)}\Big(\log\big(1-p_{D_\phi}(G_\theta(z))\big)\Big) \ \Leftrightarrow \ \min_\theta \ \mathbb{E}_{z\sim p_z(z)}\Big(-\log p_{D_\phi}(G_\theta(z)\Big)$$

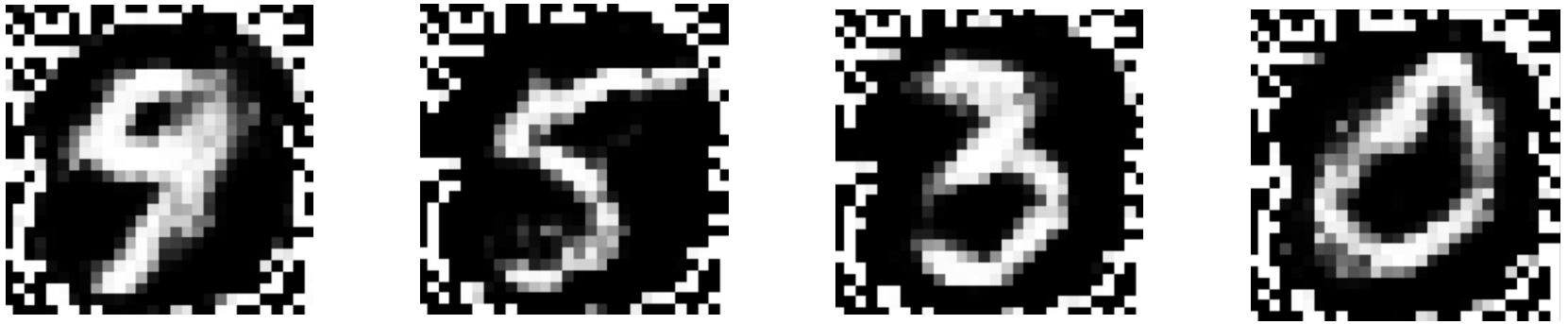
- Implementation :

$$\min_\theta \ \mathbb{E}_{z\sim p_z(z)}\Big(-\log p_{D_\phi}(G_\theta(z)\Big) \ \Leftrightarrow \ \min_\theta \ \text{BCELoss}(p_{D_\phi}, \hat{p}=1)$$

Lab 02

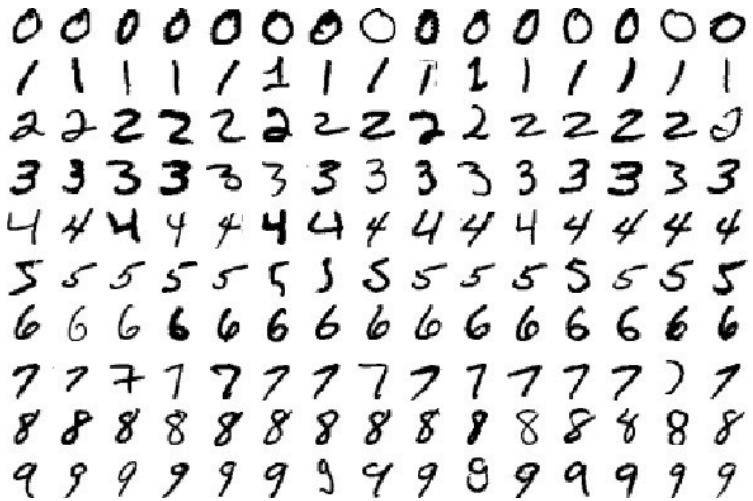
- Hyper-parameters : d_z=28 (latent dim), d=32.(n/4)² (hidden dim)
- Training : n_{epoch}=50, n_{batch}=200 (nb of batches/epoch), b=64 (batch size)
- Testing
  - Generate new images :

$$z \sim p_z(z) = \text{Unif}[0,1] \in \mathbb{R}^{d_z} \rightarrow x = G_\theta(z) \in \mathbb{R}^{n \times n}$$



Lab 03

- Design a GAN architecture with CNN that generates new MNIST images.



jupyter GAN_CNN_exercise Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

Lab 03 : GAN with CNN - exercise

The goal is to implement a GAN architecture with CNNs to generate new MNIST images.

In [ ]:
```
# For Google Colaboratory
import sys, os
if 'google.colab' in sys.modules:
    # mount google drive
    from google.colab import drive
    drive.mount('/content/gdrive')
    path_to_file = '/content/gdrive/My Drive/CS4243_codes/codes/labs_lecture15/lab03_GAN_CNN'
    print(path_to_file)
    # move to Google Drive directory
    os.chdir(path_to_file)
    !pwd
```

In [ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import utils
import time
```

In [ ]:
```
device= torch.device("cuda")
#device= torch.device("cpu")
print(device)
```

In [ ]:
```
# Libraries
import matplotlib.pyplot as plt
import logging
logging.getLogger().setLevel(logging.CRITICAL) # remove warnings
```

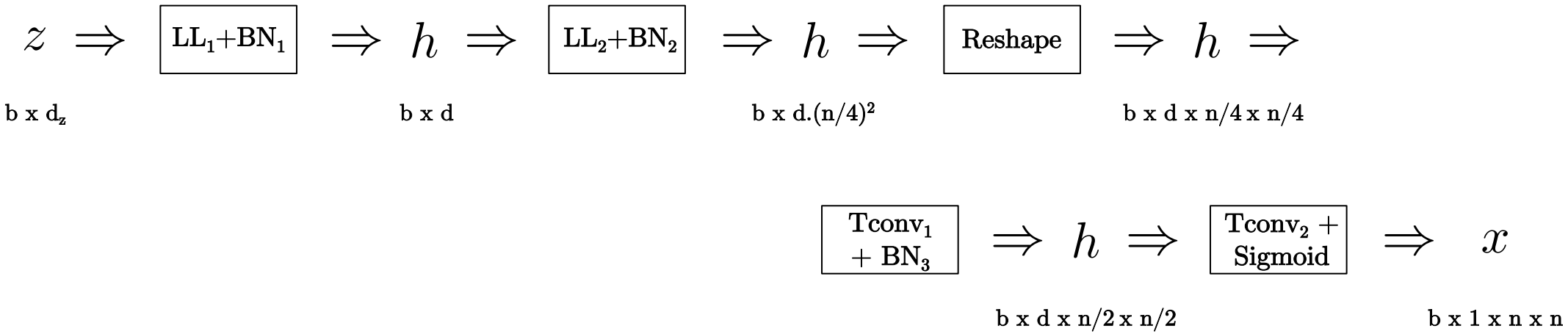
Lab 03

- Objectives
  - Implement the generator and the discriminator with convolutions and transposed convolutions.
  - Train the network.
  - Generate new images.

Lab 03

- Architecture design
  - Generator :

$$x = G_\theta(z) \in \mathbb{R}^{b \times n \times n}$$



Lab 03

- Architecture design
  - Discriminator :

$$p_{D_\phi}(x) \in [0,1]^b$$

$$x \;\Rightarrow\; \boxed{\text{Conv}_1 + \text{BN}_1} \;\Rightarrow\; h \;\Rightarrow\; \boxed{\text{Conv}_2 + \text{BN}_2} \;\Rightarrow\; h \;\Rightarrow\; \boxed{\text{Reshape}} \;\Rightarrow\; h \;\Rightarrow$$

b x 1 x n x n — b x d x n/2 x n/2 — b x d x n/4 x n/4 — b x d.(n/4)²

$$\boxed{\text{LL}_1 + \text{BN}_3} \;\Rightarrow\; h \;\Rightarrow\; \boxed{\text{LL}_2 + \text{Sigmoid}} \;\Rightarrow\; p_{D_\phi}$$

b x d — b x 1

Lab 03

- Loss :

$$\max_{\phi} \min_{\theta} \mathbb{E}_{x\sim p(x)}\Big(\log p_{D_\phi}(x)\Big) + \mathbb{E}_{z\sim p_z(z)}\Big(\log\big(1 - p_{D_\phi}(G_\theta(z))\big)\Big)$$

- Alternate optimization :

$$\min_{\theta} \mathbb{E}_{z\sim p_z(z)}\Big(\log\big(1 - p_{D_\phi}(G_\theta(z))\big)\Big) \Leftrightarrow \min_{\theta} \mathbb{E}_{z\sim p_z(z)}\Big(-\log p_{D_\phi}(G_\theta(z)\Big)$$

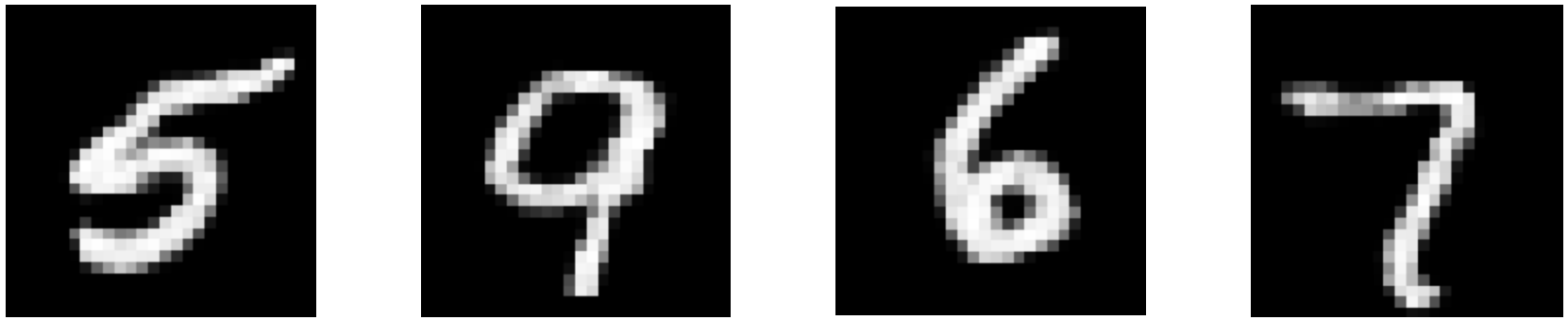
$$\max_{\phi} \mathbb{E}_{x\sim p(x)}\Big(\log p_{D_\phi}(x)\Big) + \mathbb{E}_{z\sim p_z(z)}\Big(\log\big(1 - p_{D_\phi}(G_\theta(z))\big)\Big)$$

- Implementation : Same as Lab 02

Lab 03

- Hyper-parameters : d_z=28 (latent dim), d=128 (hidden dim)
- Training : n_{epoch}=50, n_{batch}=200 (nb of batches/epoch), b=64 (batch size)
- Testing
  - Generate new images :

$$z \sim p_z(z) = \text{Unif}[0,1] \in \mathbb{R}^{d_z} \rightarrow x = G_\theta(z) \in \mathbb{R}^{n \times n}$$



Tricks for Training GANs

- GANs are hard to train because of the competition between the generator and the discriminator that optimize opposite objectives.
  - The discriminator wants to distinguish real and generated data, and
  - The generator wants to produce new data indistinguishable from real data.
- This makes the min-max optimization problem unstable and hard to converge to a solution.
- Several critical tricks are required to train successfully GAN.

- Tricks will be ordered from most important to less critical.

Radford, Metz, Chintala, DCGAN : Unsupervised representation learning with deep convolutional generative adversarial networks, 2015

Tricks for Training GANs

- Most important tricks
  - Generator and discriminator architectures must be “symmetric”.
    - Observe the sequences of G and D layers (tensor sizes) :
      - Gen : $d_z \Rightarrow d \Rightarrow d.(n/2^p)^2 \Rightarrow d \times n/2^p \times n/2^p \Rightarrow d \times n/2^{p-1} \times n/2^{p-1} \Rightarrow \ldots \Rightarrow c \times n \times n$
      - Dis : $c \times n \times n \Rightarrow d \times n/2 \times n/2 \Rightarrow d \times n/2^2 \times n/2^2 \Rightarrow \ldots \Rightarrow d \times n/2^p \times n/2^p \Rightarrow d.(n/2^p) \Rightarrow d \Rightarrow 1$

      with n=nb of pixels, c=nb of colors, d=hidden dim, d_z=latent dim, p=nb of hierarchical levels
    - Use convolution for reducing domain size by 2 : $d \times n/2^p \times n/2^p \Rightarrow d \times n/2^{p+1} \times n/2^{p+1}$
    - Use transposed convolution for increasing domain size by 2 : $d \times n/2^p \times n/2^p \Rightarrow d \times n/2^{p-1} \times n/2^{p-1}$
    - Generator and discriminator have the same number of parameters (same learning capacity).

Tricks for Training GANs

- Most important tricks
  - Adam optimizer default parameter must be changed.
    - Parameter β_1 in the computation of the running average of the gradient has default value 0.9, meaning that the new gradient counts for 90% of the running average. Smaller value in the range of [0.3,0.5] must be used to stabilize the training.
  - Batch normalization must be used to regulate the training and prevent mode collapse.
    - The most critical BN is after the first layer of the generator that transforms the low-dim z latent vector into a higher-dim vector (that will be used to generate a new image with the subsequent transposed convolution layers).
  - Architecture design (with convolution, transposed convolution and G/D symmetry) is more important than the number of learnable parameter (you can learn MNIST wit 25k parameters and not 25M parameters).

Tricks for Training GANs

- Monitoring the losses :
  - At the start : D loss is > 1 and G loss is < 1.
  - During training : D will decrease and G will increase.
  - If success : D loss is [0.3-0.6] and G loss > 2.
  - If failure : D loss is close to zero and G loss > 2.
- The failure mode can be clearly identified :
  - D loss decreases continuously to 0 and stay close to zero.
  - After 1-2 epochs, generated images are block pixels.

Tricks for Training GANs

- Less critical tricks
  - Re-scale your training data to be in the range [-1,1] or [0,1].
    - For [-1,1], the last layer of the generator must be the tanh function.
    - For [0,1], the last layer of the generator must be the sigmoid function.
  - It is better to represent $p_z(z)$ with the Uniform distribution than the Normal distribution (less mode collapse).
  - Update the generator 2x more than the discriminator.

Outline

- Variational autoencoders (VAE)
- Lab on VAE
- Generative Adversarial Networks (GAN)
- Labs on GAN
- Conclusion

Conclusion

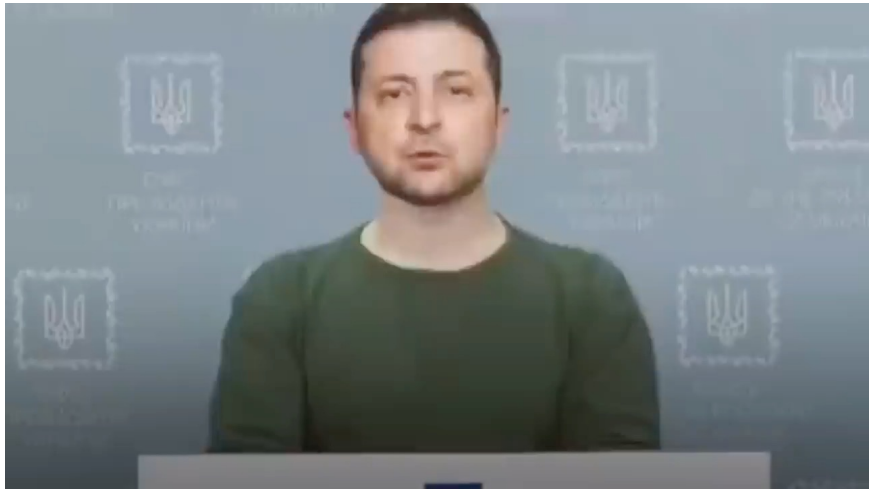
- Generative models are learning techniques that produce new data from a training dataset.
- New data are produced either by
  - Sampling explicitly from the learned (possibly approximated) data distribution (PixelRNN, PixelCNN, VAE), or
  - Generating samples from an implicit data distribution (GAN).

Conclusion

- Exciting applications :
  - Entertainment industry with generated new movies and games. Do you want to be the star of the movie? :)
  - Arts with new creative tools.

Deepfake video of Jim Carrey in the role of Jack Nicholson in Stanley Kubrick's The Shining (Jul 9, 2019)

- Alarming applications :
  - Deepfakes for blackmail, revenge porn, politics.



Deepfake video of Volodymyr Zelensky telling Ukrainians to surrender (16 March 2022)

Questions?