Computer Vision & Generative Models

Andrey Filatov[1], Daniil Yakovlev[2,3]

[1]Moscow Institute of Physics and Technology
[2]Higher School of Economics
[3]Yandex School of Data Analysis

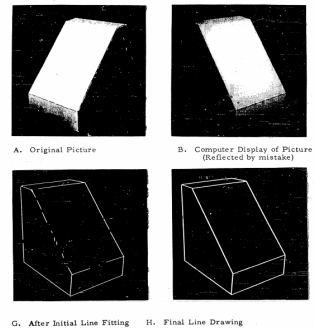
Outline

1. Computer vision
2. Generative models

Computer Vision

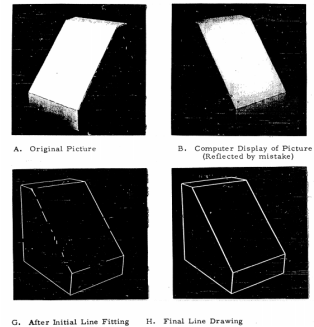
Perception of 3d Solids

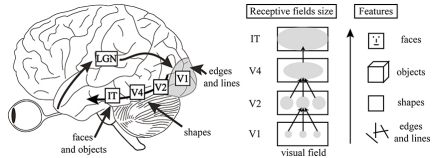
- Ideas of teaching the machine to see started to appear in the early 60-s



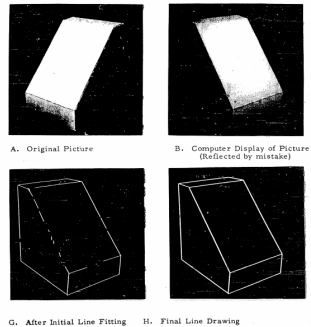
Perception of 3d Solids

- Ideas of teaching the machine to see started to appear in the early 60-s
- And the inspiration came from the neuroscience



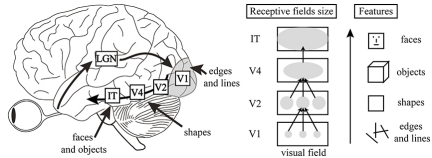


Perception of 3d Solids



- Ideas of teaching the machine to see started to appear in the early 60-s
- And the inspiration came from the neuroscience

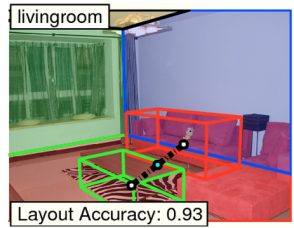
What should we request from such kind of machine?

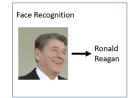


Computer vision

problem types

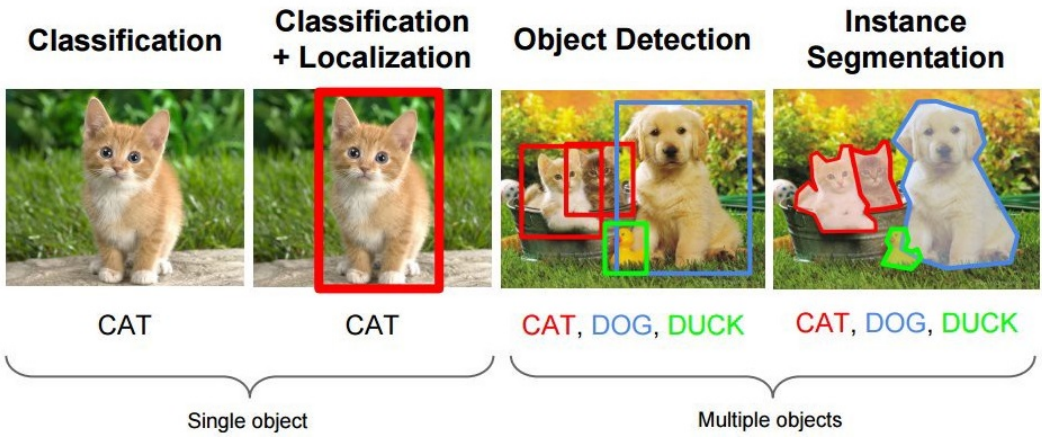
- Object detection
- Object classification
- Scene understanding
- 3D reconstruction
- Object tracking
- . . .





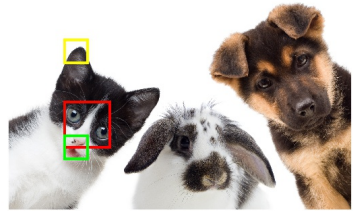
Computer vision

problem types



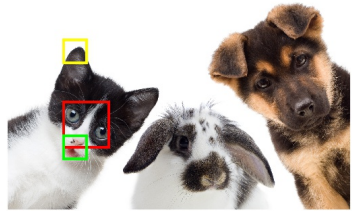
Computer vision

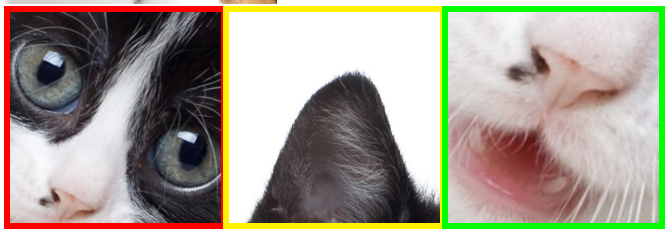
how to deal with features?



Computer vision

how to deal with features?

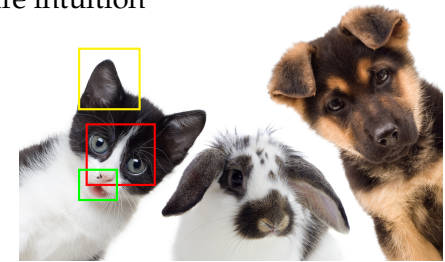


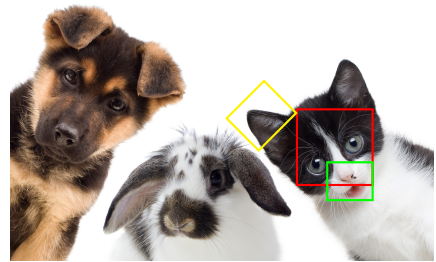


Computer vision

architecture intuition

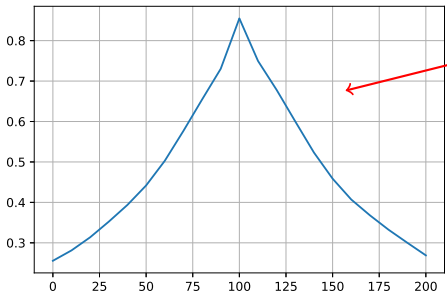
- To perform identification we are seeking for noticeable parts of an object
- However, these parts appear at different locations
- Want to perform a feature search over the whole picture





Computer vision

architecture intuition

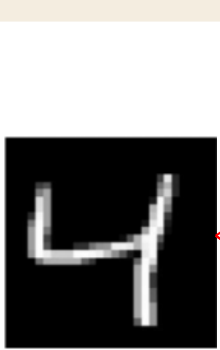


State of similarity

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} * \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{(m-i)(n-j)} y_{(1+i)(1+j)}$$

Computer vision

convolution as filter



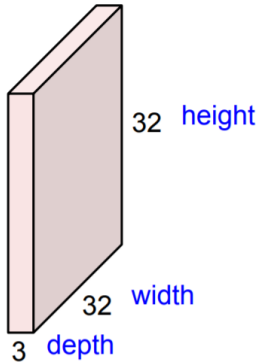
Input picture (X_{ij})

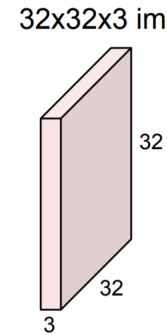
Kernels (Y_{ij})

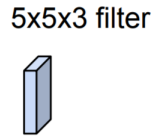
Convolved pictures

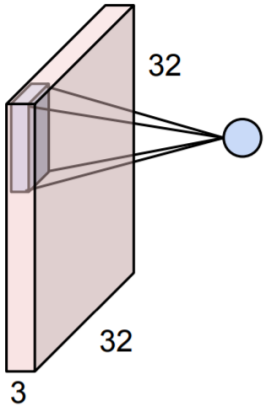
Computer vision

RGB picture convolution

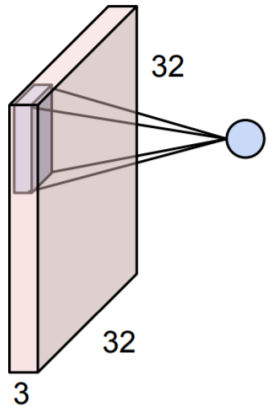




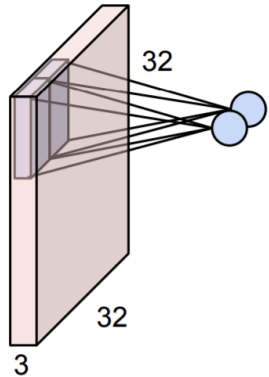




Blue ball is the result of taking a dot product between the filter and a small $5 \times 5 \times 3$ chunk of the image (i.e. $5 \cdot 5 \cdot 3 = 75$-dimensional dot product + bias)

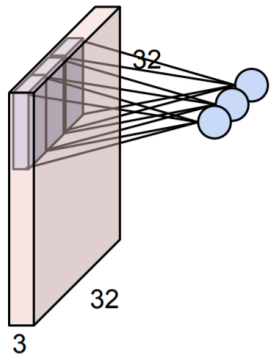

32
32
3

RGB picture convolution



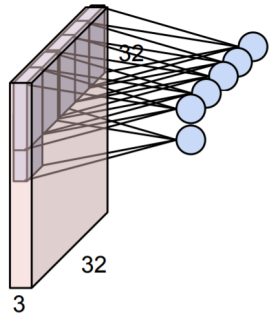
Computer vision

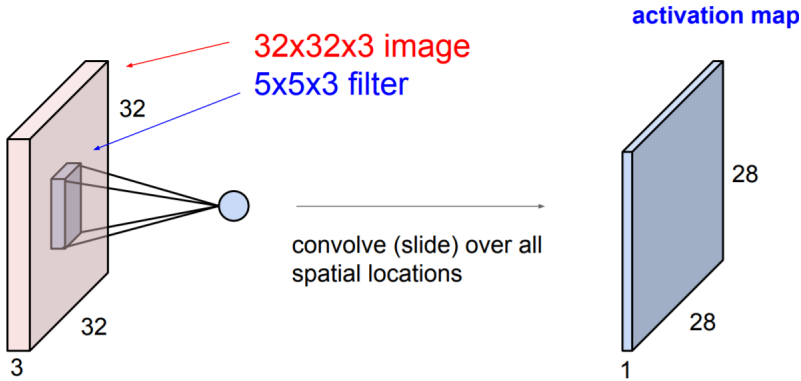
RGB picture convolution



Computer vision

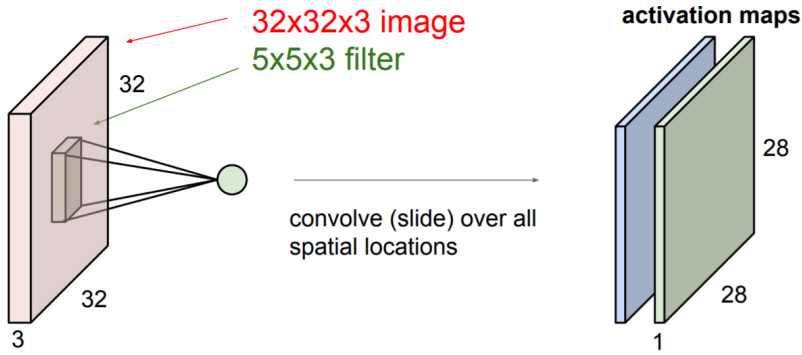
RGB picture convolution

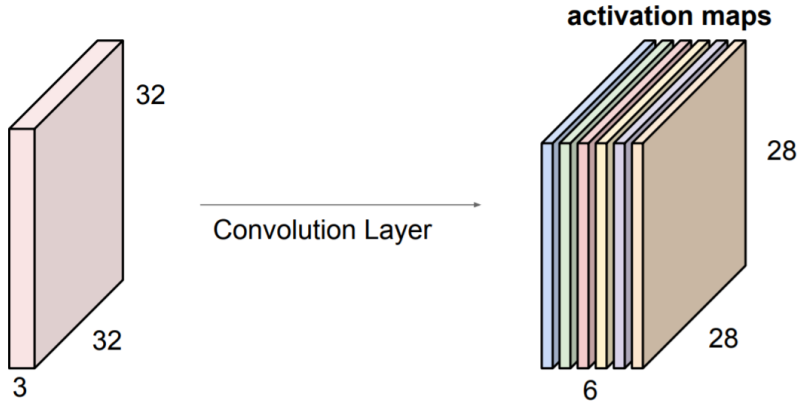


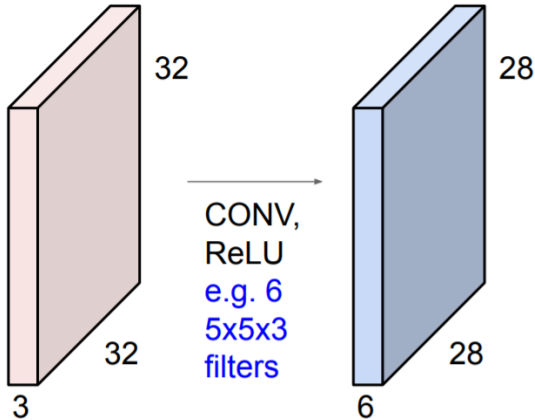

32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all
spatial locations
activation map
28
28
1

Computer vision

adding more layers

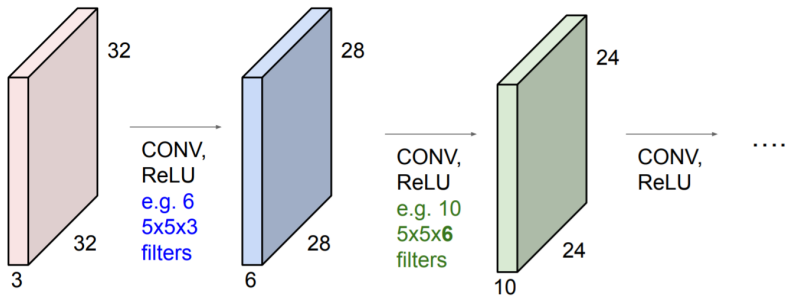



32
32
3
Convolution Layer
activation maps
28
28
6


32
32
3
CONV,
ReLU
e.g. 6
5x5x3
filters
28
28
6

Computer vision

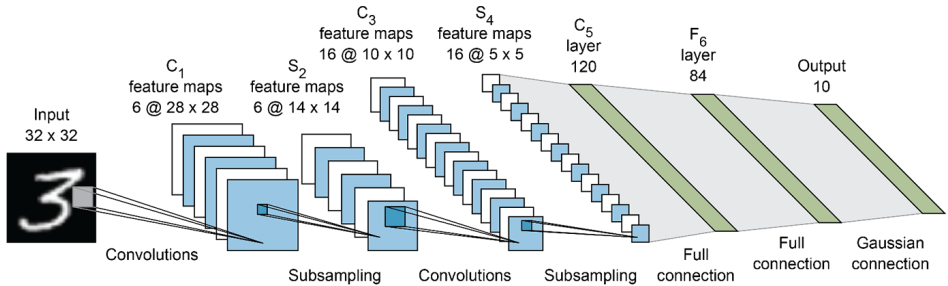
stacking more layers



Computer vision

putting together

paper



Check your understanding

5	3	9	8	0
2	0	8	1	6
9	8	1	1	5
1	5	4	0	3
9	8	6	3	0

*

1	0	0
0	1	0
0	0	1

=

Check your understanding

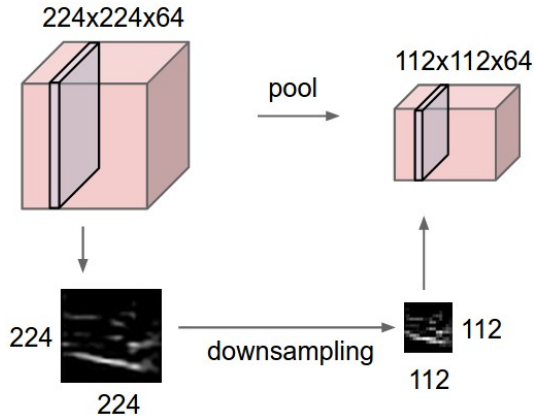
5	3	9	8	0
2	0	8	1	6
9	8	1	1	5
1	5	4	0	3
9	8	6	3	0

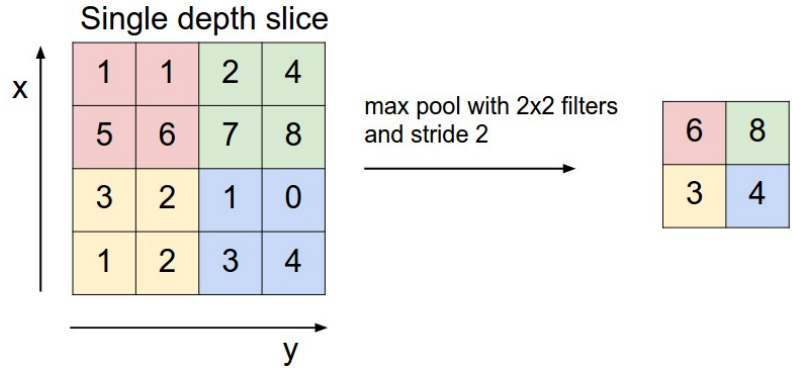
*

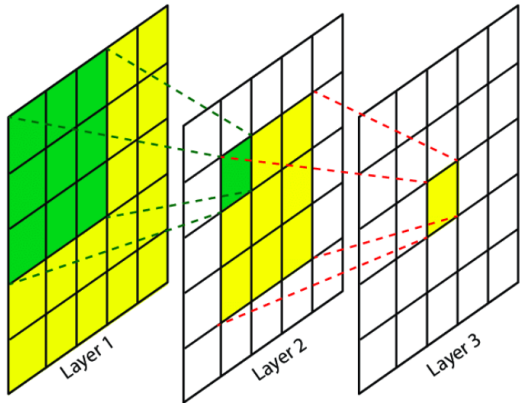
1	0	0
0	1	0
0	0	1

=

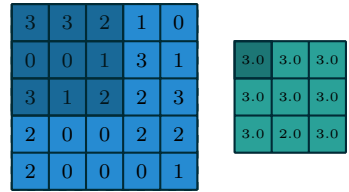
6	12	15
14	1	12
20	15	1


224x224x64
pool
112x112x64
224
224
downsampling
112
112

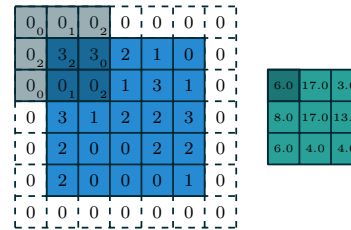

Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4


Layer 1
Layer 2
Layer 3

What can we do to decrease size of image?

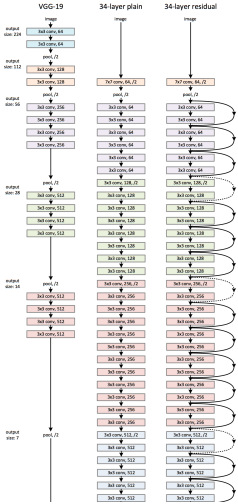


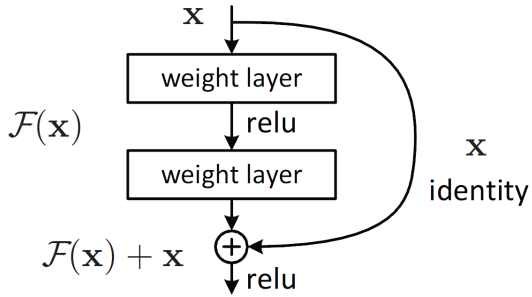
What can we do to decrease size of image?



Computer vision

residual connections

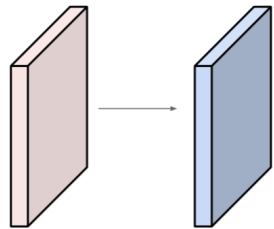




Check your understanding

- Input: $32 \times 32 \times 3$
- Filters: ten 5×5 filters
- Stride: 1
- padding: 2

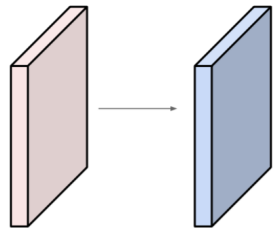
Output volume size?



Check your understanding

- Input: $32 \times 32 \times 3$
- Filters: ten 5×5 filters
- Stride: 2
- padding: 2

Number of parameters in this layer?



Check your understanding

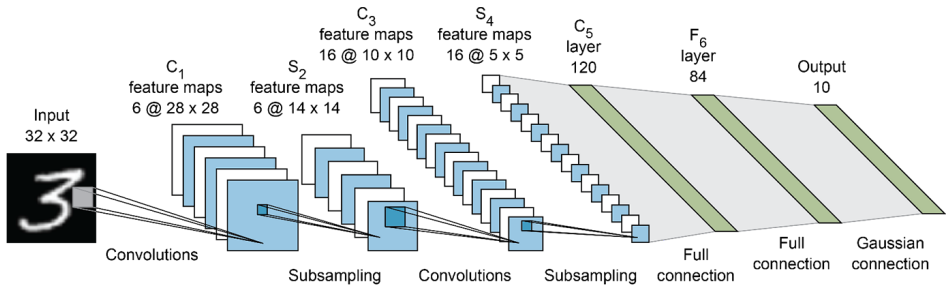
Let's assume input is $W_1 \times H_1 \times C$ Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

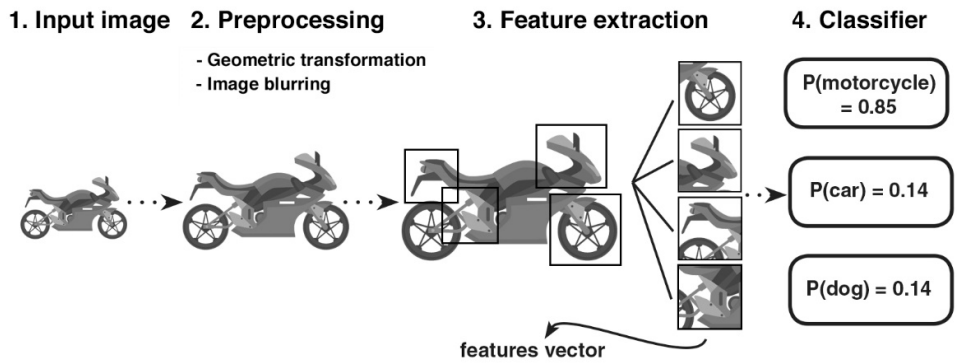
This will produce an output of $W_2 \times H_2 \times K$ where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

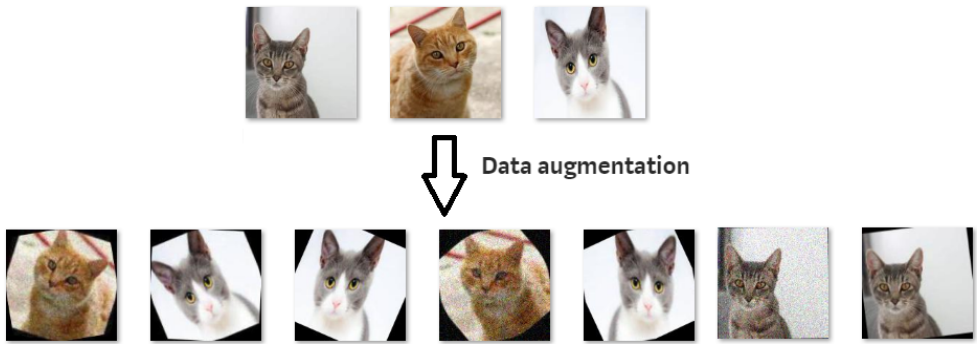

Input
32 x 32
C1
feature maps
6 @ 28 x 28
S2
feature maps
6 @ 14 x 14
C3
feature maps
16 @ 10 x 10
S4
feature maps
16 @ 5 x 5
C5
layer
120
F6
layer
84
Output
10
Convolutions
Subsampling
Convolutions
Subsampling
Full
connection
Full
connection
Gaussian
connection

Computer vision — pipeline



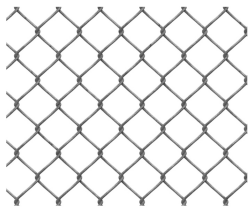
Computer vision

augmentation



Computer vision ——o representation

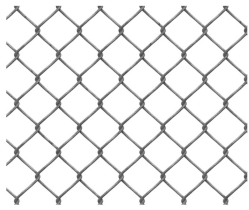
distill blog post

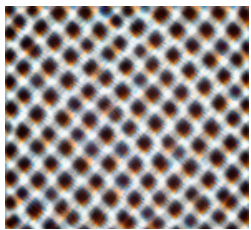


Computer vision

representation

distill blog post



Generative Models

Introduction

more from TF

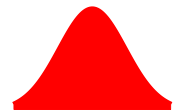
- Previously in the course we were mostly working in a **supervised** mode, trying to predict some y about data $\boldsymbol{x} \Rightarrow p(y|\boldsymbol{x})$
- Suppose now we have dataset $X = \{\boldsymbol{x}_i,\ i = 1 \dots N\}$ and instead we want to understand its *nature*:
  - Find underlying **probability density function** (pdf): $\boldsymbol{x} \sim p(\boldsymbol{x})$
  - **Sample** from $p(\boldsymbol{x})$: generate new objects with the same "properties" as $\boldsymbol{x}$
- And we want to do it in **unsupervised** way:
  no labels/supervision $\Rightarrow$ learn about data *directly* from data itself

Introduction

more from TF

- Previously in the course we were mostly working in a **supervised** mode, trying to predict some y about data $\boldsymbol{x} \Rightarrow p(y|\boldsymbol{x})$
- Suppose now we have dataset $X = \{\boldsymbol{x}_i,\ i = 1 \dots N\}$ and instead we want to understand its *nature*:
  - Find underlying **probability density function** (pdf): $\boldsymbol{x} \sim p(\boldsymbol{x})$
  - **Sample** from $p(\boldsymbol{x})$: generate new objects with the same "properties" as $\boldsymbol{x}$
- And we want to do it in **unsupervised** way:
  no labels/supervision $\Rightarrow$ learn about data *directly* from data itself

  NB: in this section we will focus on **images**, since GMs are widely applied in this domain. However, one can always apply the concept to any kind of data (tabular/sound/text/etc.)

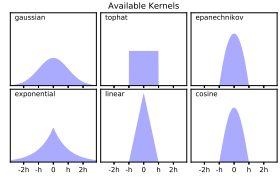


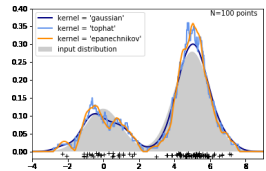
KDE

- We could use **Kernel Density Estimation** (KDE):

$$p(\boldsymbol{x}) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where h – bandwidth (smoothing parameter), K - kernel function





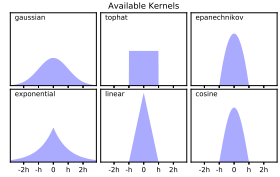
KDE

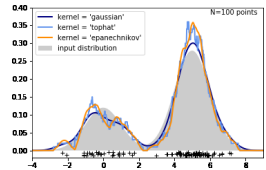
- We could use **Kernel Density Estimation** (KDE):

$$p(\boldsymbol{x}) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where h – bandwidth (smoothing parameter), K - kernel function

- However, KDE fails to capture densities in high-dimensional spaces because of the **curse of dimensionality**





KDE

thiscatdoesnotexist.com

- We could use **Kernel Density Estimation** (KDE):

$$p(\boldsymbol{x}) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where h – bandwidth (smoothing parameter), K - kernel function

- However, KDE fails to capture densities in high-dimensional spaces because of the **curse of dimensionality**
- How can we model complex high-dimensional data?

Autoencoder

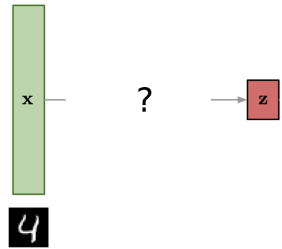
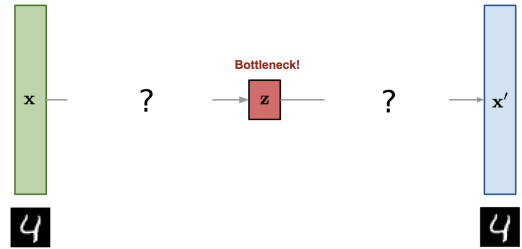


illustration credit

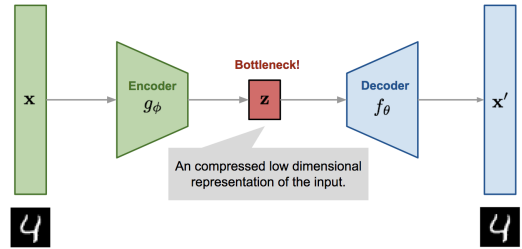
- Suppose now our "true" density has a lower number of dimensions than we observe in data – that is, there are *redundant dimensions*
- Let's use this fact and try to reconstruct this "true" (aka **embedded** or **latent**) representation $z \sim q(z)$ of data $x \sim p(x)$ – but how?

Autoencoder



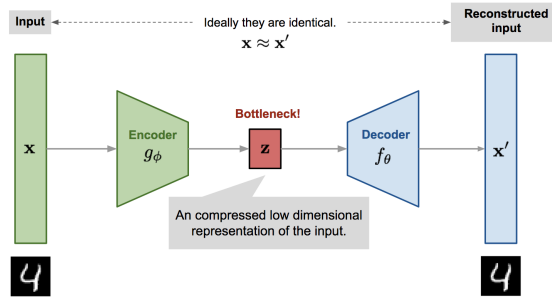
- Suppose now our "true" density has a lower number of dimensions than we observe in data – that is, there are *redundant dimensions*
- Let's use this fact and try to reconstruct this "true" (aka **embedded** or **latent**) representation $z \sim q(z)$ of data $x \sim p(x)$ – but how?
- Well, in the end we still need to have data, since we want to sample it: $x' \sim p(x)$

Autoencoder



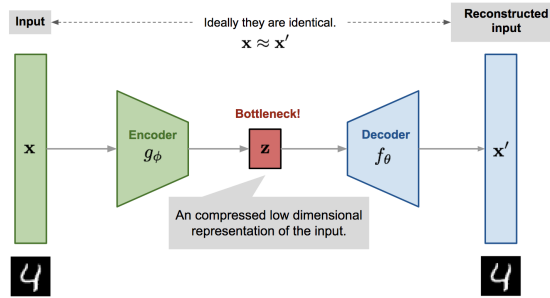
- Suppose now our "true" density has a lower number of dimensions than we observe in data – that is, there are *redundant dimensions*
- Let's use this fact and try to reconstruct this "true" (aka **embedded** or **latent**) representation $z \sim q(z)$ of data $x \sim p(x)$ – but how?
- Well, in the end we still need to have data, since we want to sample it: $x' \sim p(x)$
- Filling space in between with *parametrized* encoder $g_\phi(x)$ and decoder $f_\theta(z) \leftarrow$ NNs

Autoencoder



- Suppose now our "true" density has a lower number of dimensions than we observe in data – that is, there are *redundant dimensions*
- Let's use this fact and try to reconstruct this "true" (aka **embedded** or **latent**) representation $\boldsymbol{z} \sim q(\boldsymbol{z})$ of data $\boldsymbol{x} \sim p(\boldsymbol{x})$ – but how?
- Well, in the end we still need to have data, since we want to sample it: $\boldsymbol{x}' \sim p(\boldsymbol{x})$
- Filling space in between with *parametrized* encoder $g_{\boldsymbol{\phi}}(\boldsymbol{x})$ and decoder $f_{\boldsymbol{\theta}}(\boldsymbol{z}) \leftarrow$ NNs
- Adding loss function which compares input and output (e.g MSE)

Autoencoder



- Suppose now our "true" density has a lower number of dimensions than we observe in data – that is, there are *redundant dimensions*
- Let's use this fact and try to reconstruct this "true" (aka **embedded** or **latent**) representation $\boldsymbol{z} \sim q(\boldsymbol{z})$ of data $\boldsymbol{x} \sim p(\boldsymbol{x})$ – but how?
- Well, in the end we still need to have data, since we want to sample it: $\boldsymbol{x}' \sim p(\boldsymbol{x})$
- Filling space in between with *parametrized* encoder $g_{\boldsymbol{\phi}}(\boldsymbol{x})$ and decoder $f_{\boldsymbol{\theta}}(\boldsymbol{z}) \leftarrow$ NNs
- Adding loss function which compares input and output (e.g MSE)
- Use **backprop + gradient descend** to find optimal $\boldsymbol{\phi}, \boldsymbol{\theta} \rightarrow$ architecture forced by loss function to reproduce $\boldsymbol{x}$

Autoencoder

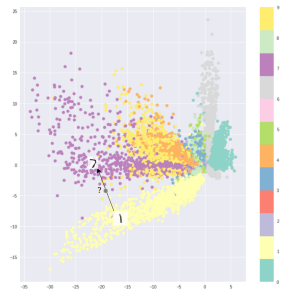
But can we really use AE as a generative model?

Autoencoder

But can we really use AE as a generative model? No

Autoencoder

illustration credit

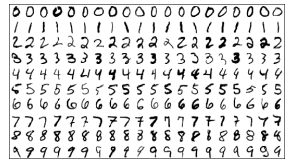


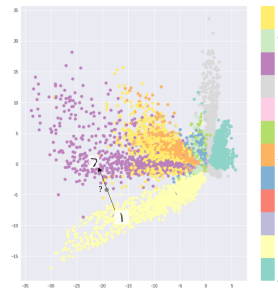
But can we really use AE as a generative model? No

- **Latent space isn't continuous**: essentially a bunch of delta functions ⇒ hardly meet requirements of pdf
- **Latent space can't be interpolated**: because of many "gaps" w/o training samples can replicate the *same* images but don't generalise ⇒ no reasonable sampling

Autoencoder

illustration credit





But can we really use AE as a generative model? No

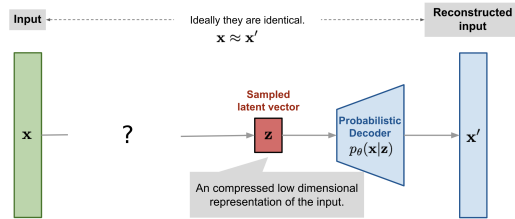
- **Latent space isn't continuous**: essentially a bunch of delta functions ⇒ hardly meet requirements of pdf
- **Latent space can't be interpolated**: because of many "gaps" w/o training samples can replicate the *same* images but don't generalise ⇒ no reasonable sampling

Can we construct a better latent representation?

Varionational Autoencoder

architecture

- Can we construct a better latent representation?



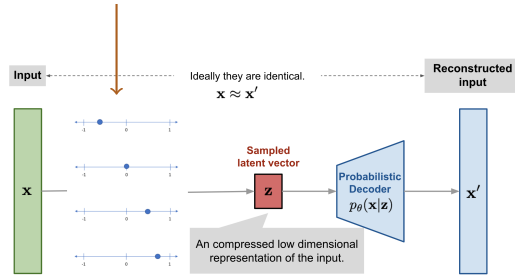
Varionational Autoencoder

architecture

- Can we construct a better latent representation?

1. Previously we populated latent space with (discrete) points ⇒ **continuity problem**

layered structure of encoder is implied (layers aren't shown)



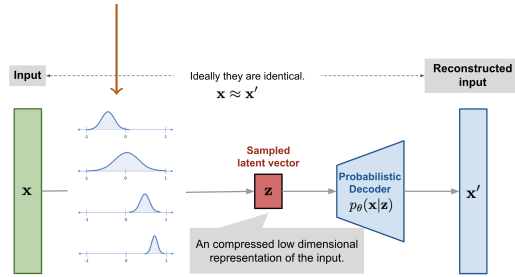
Varionational Autoencoder

architecture

- Can we construct a better latent representation?

1. Previously we populated latent space with (discrete) points ⇒ **continuity problem**
2. Then let's map each object to a distribution!

layered structure of encoder is implied (layers aren't shown)



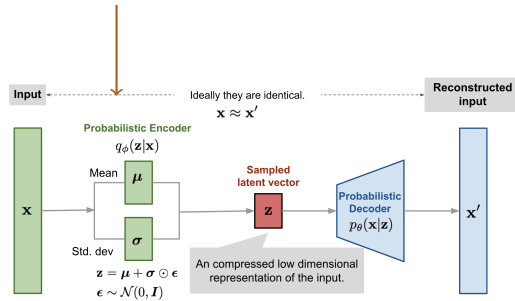
Varionational Autoencoder

architecture

- Can we construct a better latent representation?

1. Previously we populated latent space with (discrete) points ⇒ **continuity problem**
2. Then let's map each object to a distribution!
3. Which we take as a usual Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$

layered structure of encoder is implied (layers aren't shown)



Varionational Autoencoder

architecture

- Can we construct a better latent representation?

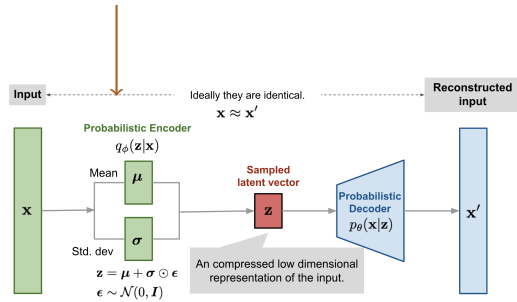
1. Previously we populated latent space with (discrete) points ⇒ **continuity problem**
2. Then let's map each object to a distribution!
3. Which we take as a usual Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$
4. Then encoder will predict its parameters

$$\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\sigma}(\boldsymbol{x}) = q_{\boldsymbol{\phi}}(\boldsymbol{x})$$

and decoder return $\boldsymbol{x}' = p_{\boldsymbol{\theta}}(\boldsymbol{z})$ where $\boldsymbol{z}$ is sampled from a corresponding Gaussian:

$$\boldsymbol{z} \sim \mathcal{N}\big(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\sigma}(\boldsymbol{x})\big)$$

layered structure of encoder is implied (layers aren't shown)



Varionational Autoencoder

architecture

- Can we construct a better latent representation?

1. Previously we populated latent space with (discrete) points ⇒ **continuity problem**
2. Then let's map each object to a distribution!
3. Which we take as a usual Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$
4. Then encoder will predict its parameters

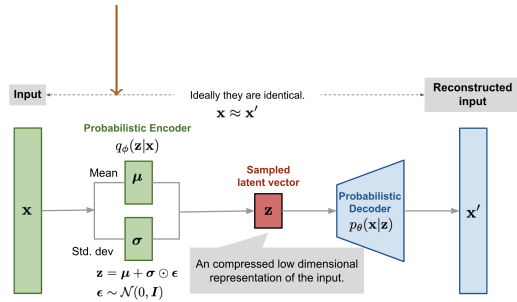
$$\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\sigma}(\boldsymbol{x}) = q_{\boldsymbol{\phi}}(\boldsymbol{x})$$

and decoder return $\boldsymbol{x}' = p_{\boldsymbol{\theta}}(\boldsymbol{z})$ where $\boldsymbol{z}$ is sampled from a corresponding Gaussian:

$$\boldsymbol{z} \sim \mathcal{N}\big(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\sigma}(\boldsymbol{x})\big)$$

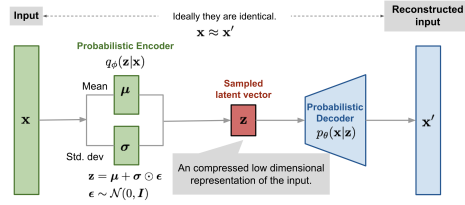
→ this approach is called **variational autoencoder**

layered structure of encoder is implied (layers aren't shown)



Varionational Autoencoder — KL divergence

- Problem: without constraints $\boldsymbol{\mu}$ will roam around latent space and $\boldsymbol{\sigma}$ might be set to 0 during the training $\Rightarrow$ VAE overfits and **collapses to AE**



Varionational Autoencoder

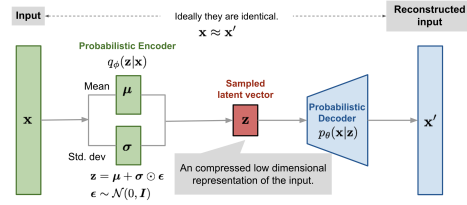
KL divergence

- Problem: without constraints $\boldsymbol{\mu}$ will roam around latent space and $\boldsymbol{\sigma}$ might be set to 0 during the training $\Rightarrow$ VAE overfits and **collapses to AE**

: Need some **regularisation** in latent space $\Rightarrow$ introducing **Kullback-Leibler divergence**:

$$\mathrm{KL}[P||Q] = \sum_x P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

it measures "distance" between two pdfs $P(x)$ and $Q(x)$



Varionational Autoencoder — KL divergence

- Problem: without constraints $\boldsymbol{\mu}$ will roam around latent space and $\boldsymbol{\sigma}$ might be set to 0 during the training $\Rightarrow$ VAE overfits and **collapses to AE**

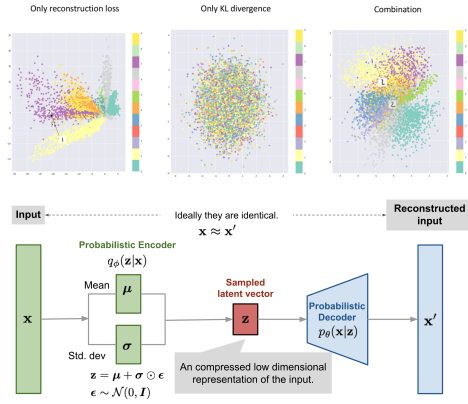
: Need some **regularisation** in latent space $\Rightarrow$ introducing **Kullback-Leibler divergence**:

$$\mathrm{KL}[P||Q] = \sum_x P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

it measures "distance" between two pdfs $P(x)$ and $Q(x)$

- Let's regularise VAE by **penalizing** encoded Gaussian **for being "far"** from a standard normal in terms of KL:

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}(\boldsymbol{x}, \boldsymbol{x}') + \mathrm{KL}[\mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\sigma}(\boldsymbol{x}))||\mathcal{N}(\mathbf{0}, \mathbf{1})]$$

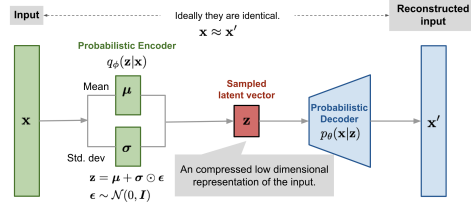


Varionational Autoencoder

training

more intuition

- For training VAE we can also use backprop to find encoder/decoder parameters $\phi, \boldsymbol{\theta}$



Varionational Autoencoder

training

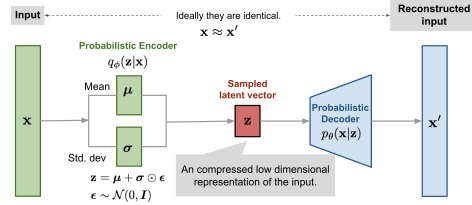
more intuition

- For training VAE we can also use backprop to find encoder/decoder parameters ϕ, θ

  Problem: we need to backpropagate gradients through **non-differentiable sampling** operation in the computational graph:

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}(\boldsymbol{x}, \boldsymbol{x}') + \text{KL}[\mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\sigma}(\boldsymbol{x}))||\mathcal{N}(\mathbf{0}, \mathbf{1})]$$

$$\boldsymbol{x}' = p_{\boldsymbol{\theta}}(\boldsymbol{z}), \boldsymbol{z} \sim \mathcal{N}\big(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\sigma}(\boldsymbol{x})\big)$$



Varionational Autoencoder

training

more intuition

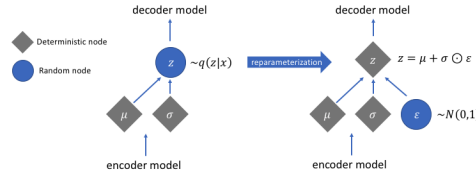
- For training VAE we can also use backprop to find encoder/decoder parameters ϕ, θ

  Problem: we need to backpropagate gradients through **non-differentiable sampling** operation in the computational graph:

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}(x, x') + \text{KL}[\mathcal{N}(\mu(x), \sigma(x))||\mathcal{N}(0, 1)]$$

$$x' = p_\theta(z), z \sim \mathcal{N}\big(\mu(x), \sigma(x)\big)$$

→ Apply **reparameterization trick**



Varionational Autoencoder

training

more intuition

- For training VAE we can also use backprop to find encoder/decoder parameters ϕ, θ

  Problem: we need to backpropagate gradients through **non-differentiable sampling** operation in the computational graph:

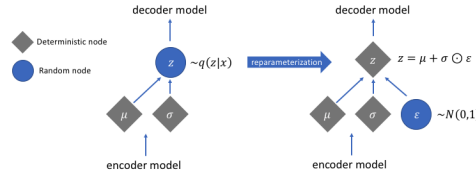
$$\mathcal{L}_{\text{VAE}} = \mathcal{L}(\boldsymbol{x}, \boldsymbol{x}') + \text{KL}[\mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\sigma}(\boldsymbol{x}))||\mathcal{N}(\mathbf{0}, \mathbf{1})]$$

$$\boldsymbol{x}' = p_{\boldsymbol{\theta}}(\boldsymbol{z}), \boldsymbol{z} \sim \mathcal{N}\big(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\sigma}(\boldsymbol{x})\big)$$

- → Apply **reparameterization trick**

  Note: you can impose any other *prior* $p(\boldsymbol{z})$ on the latent space, not necessarily $\mathcal{N}(\mathbf{0}, \mathbf{1})$:

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}(\boldsymbol{x}, \boldsymbol{x}') + \text{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})]$$



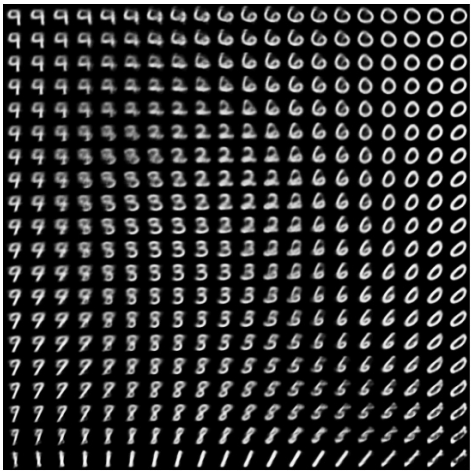
Note: this was a somewhat informal explanation of VAE – if one wants to take things seriously, they should go bayesian

Varionational Autoencoder

latent space

from TF tutorial

wow



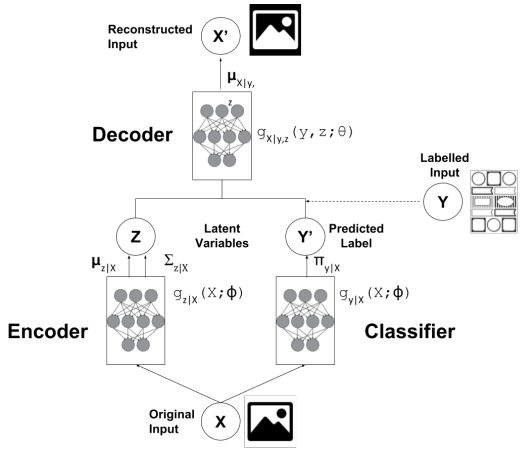
so continuity

that interpretable

much interpolation

Varionational Autoencoder

semi-supervised learning



source

NVIDIA paper

- VAE represents **explicit density approach**: for each object $\boldsymbol{x}$ it outputs $p(\boldsymbol{x})$ (we skipped this but it's true)
- What if we don't have to know density? Just want to generate data and nothing else

NVIDIA paper

- VAE represents **explicit density approach**: for each object $\boldsymbol{x}$ it outputs $p(\boldsymbol{x})$ (we skipped this but it's true)
- What if we don't have to know density? Just want to generate data and nothing else
- What about implicit approach?

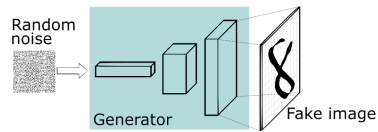
GAN — motivation

NVIDIA paper

- VAE represents **explicit density approach**: for each object $\boldsymbol{x}$ it outputs $p(\boldsymbol{x})$ (we skipped this but it's true)
- What if we don't have to know density? Just want to generate data and nothing else
- What about implicit approach? $\Rightarrow$ **Generative Adversarial Network** (GAN)

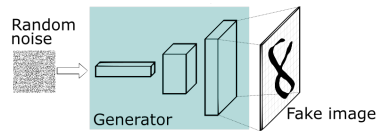
GAN — architecture

- Suppose we know how to generate images we want from random noise



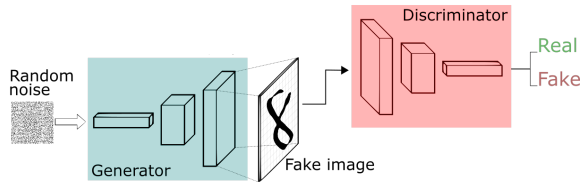
GAN — architecture

- Suppose we know how to generate images we want from random noise
- Then, it makes sense to evaluate the quality of these images



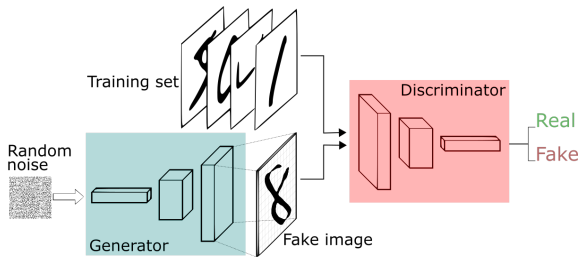
GAN — architecture

- Suppose we know how to generate images we want from random noise
- Then, it makes sense to evaluate the quality of these images
- Let's add a model which makes it for us by telling whether input image is "fake" or not

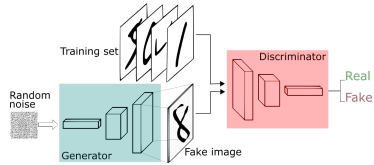


GAN — architecture

- Suppose we know how to generate images we want from random noise
- Then, it makes sense to evaluate the quality of these images
- Let's add a model which makes it for us by telling whether input image is "fake" or not
- Then, the model needs "real" examples to be able to learn the difference btw. them



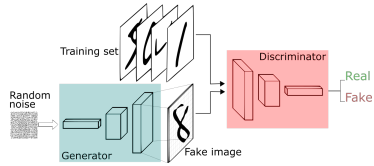
GAN — training



- Let's sample some noise as input (usually, from normal/uniform prior $p_z(z)$):

$$z_i \sim p_z(z)$$

GAN — training



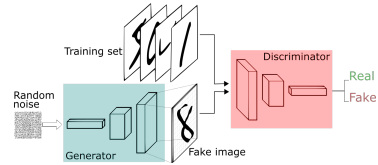
- Let's sample some noise as input (usually, from normal/uniform prior $p_z(z)$):

$$z_i \sim p_z(z)$$

- Then propagate the noise through the **generator** G_θ to get "fake" images x'_i:

$$x'_i = G_\theta(z_i)$$

GAN — training



- Let's sample some noise as input (usually, from normal/uniform prior $p_z(z)$):

$$z_i \sim p_z(z)$$

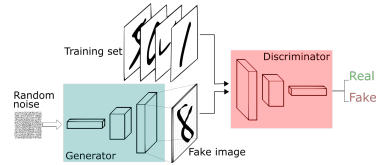
- Then propagate the noise through the **generator** G_θ to get "fake" images x'_i:

$$x'_i = G_\theta(z_i)$$

- **Discriminator** D_ϕ, given a bunch of "real" and "fake" images, will return for them probability to be "real"

$$p(\text{real}|x) = D_\phi(x)$$

GAN — training



- Let's sample some noise as input (usually, from normal/uniform prior $p_z(\boldsymbol{z})$):

$$\boldsymbol{z}_i \sim p_z(\boldsymbol{z})$$

- Then propagate the noise through the **generator** G_θ to get "fake" images $\boldsymbol{x}'_i$:

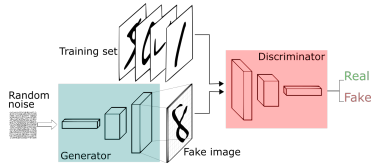
$$\boldsymbol{x}'_i = G_\theta(\boldsymbol{z}_i)$$

- **Discriminator** D_ϕ, given a bunch of "real" and "fake" images, will return for them probability to be "real"

$$p(\text{real}|\boldsymbol{x}) = D_\phi(\boldsymbol{x})$$

- On the one hand we want D_ϕ to be strong in distinguishing "fakes" and "reals". On the other, we want also a strong G_θ so that it creates good images

GAN — training



- Let's sample some noise as input (usually, from normal/uniform prior $p_z(z)$):

$$z_i \sim p_z(z)$$

- Then propagate the noise through the **generator** G_θ to get "fake" images x'_i:

$$x'_i = G_\theta(z_i)$$

- **Discriminator** D_ϕ, given a bunch of "real" and "fake" images, will return for them probability to be "real"

$$p(\text{real}|x) = D_\phi(x)$$

- On the one hand we want D_ϕ to be strong in distinguishing "fakes" and "reals". On the other, we want also a strong G_θ so that it creates good images
- Let's make them **compete** with each other (hence "adversarial"):

$$\max_\phi \Big[\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D_\phi(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D_\phi(G_\theta(z)))]\Big] \leftarrow \text{make } \textbf{discriminator} \text{ good in distinguishing}$$

$$\max_\theta \Big[\mathbb{E}_{z\sim p_z(z)}[\log(D_\phi(G_\theta(z)))]\Big] \leftarrow \text{make } \textbf{generator} \text{ good in "fooling" discriminator}$$

GAN — algorithm

Ian Goodfellow et al.

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

for number of training iterations **do**

  for k steps **do**

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

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

  end for

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

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

end for

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

GAN — algorithm

Ian Goodfellow et al.

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

for number of training iterations **do**

  for k steps **do**

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

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

  end for

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

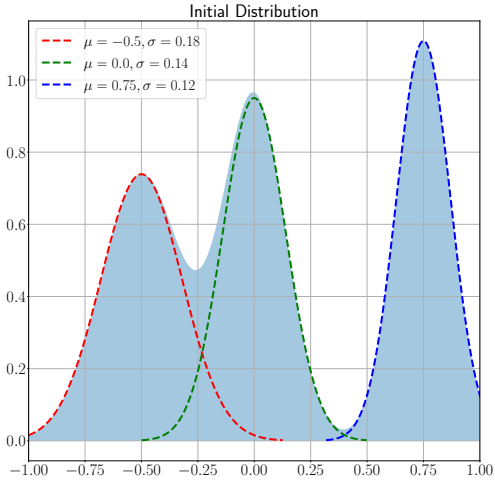
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

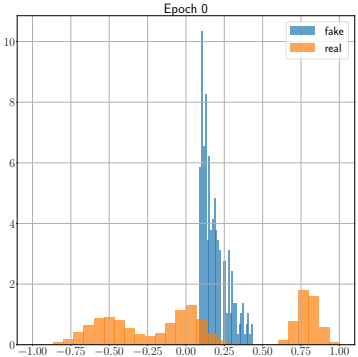
Note: there is no need to choose loss function here: **GAN automatically learns it** for us by optimizing the discriminator → just need to tell him what is good and what is not

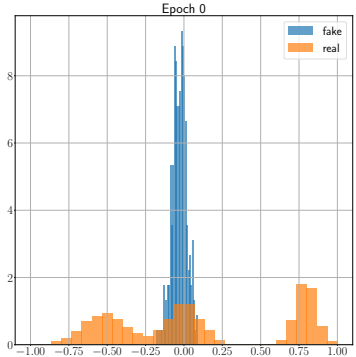
GAN — example


Initial Distribution
$\mu = -0.5, \sigma = 0.18$
$\mu = 0.0, \sigma = 0.14$
$\mu = 0.75, \sigma = 0.12$
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

GAN — example

this slide is animated – things should start moving by now!

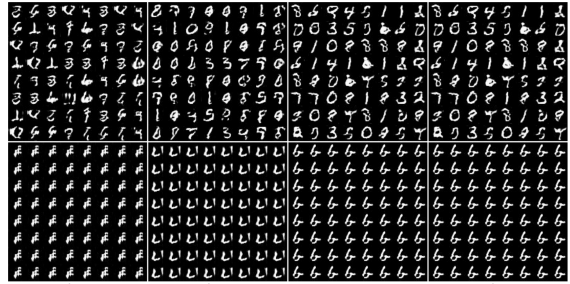




GAN — mode collapse

source

- Since our data is complex and diverse (formally, highly **multimodal**), we want generator to be able to capture this variety and be able to reproduce it
- In practise though, if generator learns to reproduce *some* part of data very well and thus fools the discrimator, they can both got stuck in this local optimum
- This is called **mode collapse**



- It was shown that when discriminator is too good, then generator training can fail due to **vanishing gradients**
- Essentially, an optimal discriminator doesn't provide enough information for the generator to make progress

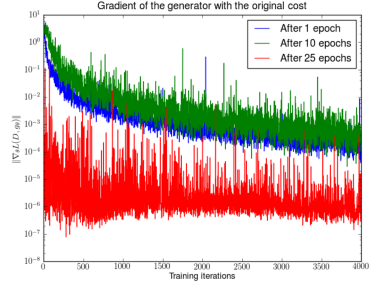
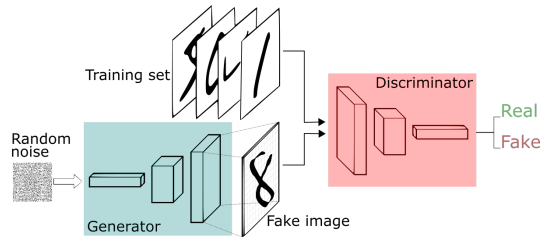


Figure 2: First, we trained a DCGAN for 1, 10 and 25 epochs. Then, with the generator fixed we train a discriminator from scratch and measure the gradients with the original cost function. We see the gradient norms decay quickly, in the best case 5 orders of magnitude after 4000 discriminator iterations. Note the logarithmic scale.

GAN — summary

- GANs dont need any Monte Carlo approximations to train
- GANs can generate data that looks VERY similar to original data
- There is a lot of awesome applications aside of simple image generation, for example:
  - style transfer
  - image inpainting
  - super-resolution
- Its hard to learn to generate discrete data, like text
- Training GAN is VERY unstable: mode collapse and vanishing gradients
- There are some tricks to stabilize it



WGAN — motivation

WGAN paper

- However, stability problem is a critical one
  $\Rightarrow$ can we improve it?
- There are several ways to fix it, but the main bottleneck is the *training paradigm* itself
- Or in fact, the underlying loss function, which turned out to be yet another divergence, **Jensen-Shannon**:

$$\text{JS}[P||Q] = \frac{1}{2}\text{KL}\left[P\middle\|\frac{1}{2}(P+Q)\right] + \frac{1}{2}\text{KL}\left[Q\middle\|\frac{1}{2}(P+Q)\right]$$

$$\mathcal{L}_{\text{GAN}}(G_\theta, D^*_\phi) = 2 \cdot \text{JS}(p_{\text{data}} \| p_{\text{gen}}) - 2\log 2$$

- Here we set D_ϕ to its optimal value:

$$D^*_\phi(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{gen}}(\boldsymbol{x})}$$

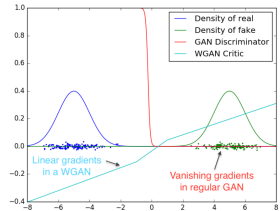


Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

WGAN — motivation

WGAN paper

- However, stability problem is a critical one
  $\Rightarrow$ can we improve it?
- There are several ways to fix it, but the main bottleneck is the *training paradigm* itself
- Or in fact, the underlying loss function, which turned out to be yet another divergence, **Jensen-Shannon**:

$$\text{JS}[P||Q] = \frac{1}{2}\text{KL}\left[P\middle|\middle|\frac{1}{2}(P+Q)\right] + \frac{1}{2}\text{KL}\left[Q\middle|\middle|\frac{1}{2}(P+Q)\right]$$

$$\mathcal{L}_{\text{GAN}}(G_\theta, D^*_\phi) = 2 \cdot \text{JS}(p_{\text{data}} \| p_{\text{gen}}) - 2\log 2$$

- Here we set D_ϕ to its optimal value:

$$D^*_\phi(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{gen}}(\boldsymbol{x})}$$

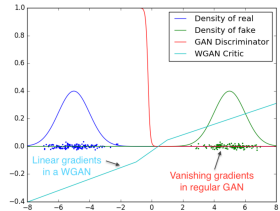


Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

maybe change of distance metric will help?

WGAN — distance example

source

Kulback-Leibler:

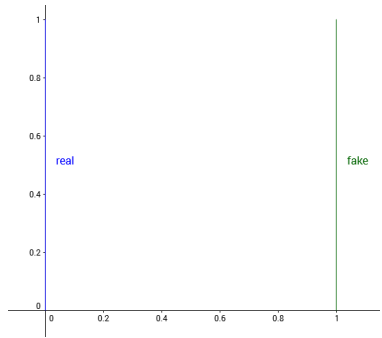
$$KL\left(P_0; P_\theta\right) = \begin{cases} +\infty & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}$$

Jenson-Shannon:

$$JS\left(P_0; P_\theta\right) = \begin{cases} \log 2 & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}$$

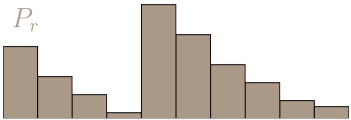
Wasserstein:

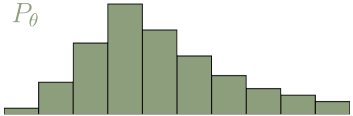
$$W\left(P_0; P_\theta\right) = |\theta|$$



WGAN — Earth-Mover distance

more on theory

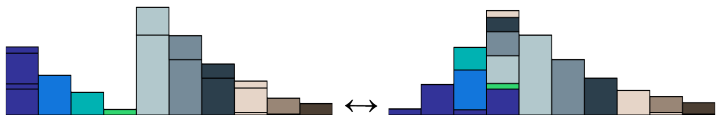




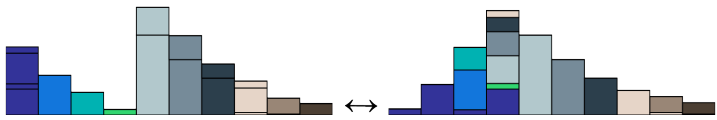
$$\sum_x \gamma(x, y) = P_r(y)$$

$$\sum_y \gamma(x, y) = P_\theta(x)$$

$$\text{EMD}(P_r, P_\theta) = \inf_{\gamma \in \Pi} \sum_{x,y} \|x - y\| \gamma(x, y) = \inf_{\gamma \in \Pi} \mathbb{E}_{(x,y) \sim \gamma} \|x - y\|$$



$$\text{EMD}(P_r, P_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r} f(x) - \mathbb{E}_{x \sim P_\theta} f(x).$$



$$\text{EMD}(P_r, P_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r} f(x) - \mathbb{E}_{x \sim P_\theta} f(x).$$

Several ways to ensure **Lipschitz-1 continuity**: gradient clipping (original WGAN), gradient penalty (WGAN-GP), spectral normalisation (SN-GAN)

Summary

- **Computer vision**
  - problem types
  - architecture intuition
  - convolutions, pooling, strides, receptive field
  - residual connections
  - augmentation
- **Generative models**
  - KDE
  - Autoencoder
  - Variation Autoencoder
  - Generative Adversarial Network (GAN)
  - Wasserstein GAN