Lecture 18: Image Synthesis

CMSC 472 / 672

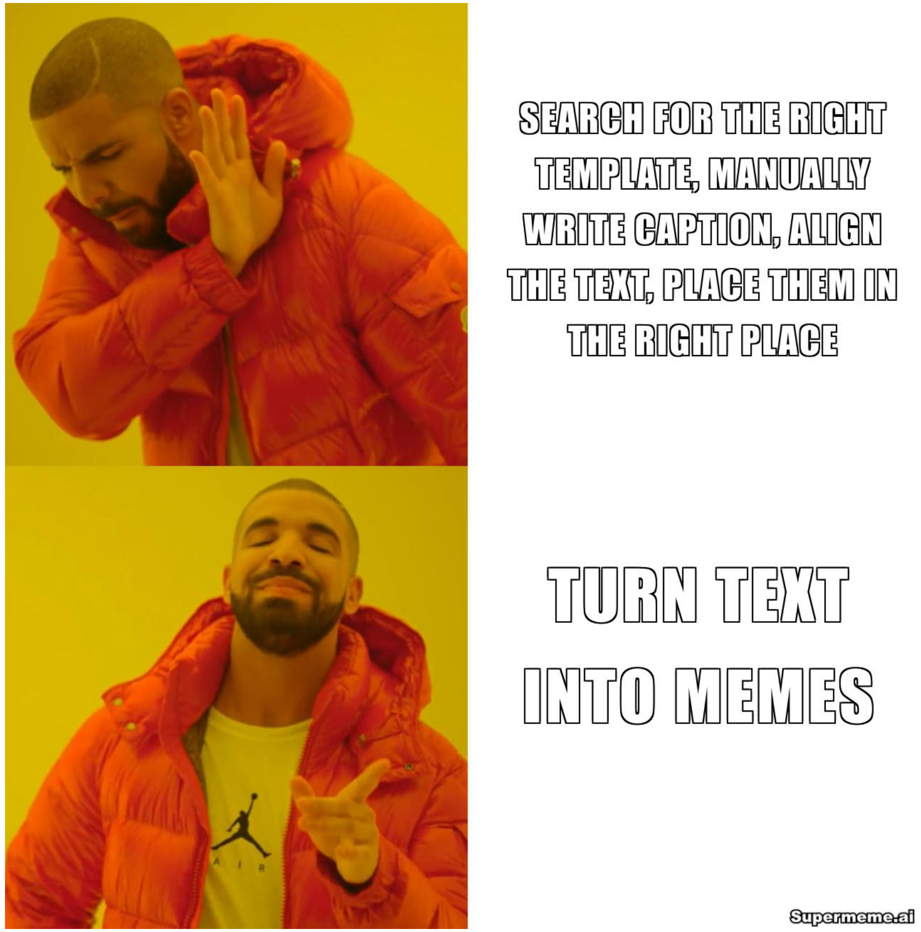
Computer Vision

UMBC

Lecture 18: Image Synthesis

CMSC 472 / 672

Computer Vision



UMBC

Machine Learning Problems

SUPERVISED LEARNING

- Training time
  - data :

$$\{\mathbf{x}^{(t)}, y^{(t)}\}$$

  - setting :

$$\mathbf{x}^{(t)}, y^{(t)} \leftarrow p(\mathbf{x}, y)$$

- Test time
  - data :

$$\{\mathbf{x}^{(t)}, y^{(t)}\}$$

  - setting :

$$\mathbf{x}^{(t)}, y^{(t)} \leftarrow p(\mathbf{x}, y)$$

Example

$Input$: $x^{(t)}$ is an image

$Output$: $y^{(t)}$ is an image category

ONE-SHOT LEARNING

- Training time
  - data :

    $$\{\mathbf{x}^{(t)}, y^{(t)}\}$$

  - setting :

    $$\mathbf{x}^{(t)}, y^{(t)} \leftarrow p(\mathbf{x}, y)$$

    subject to $y^{(t)}\ 2\ \{1, \ldots, C\}$

- Test time
  - data :

    $$\{\mathbf{x}^{(t)}, y^{(t)}\}$$

  - setting :

    $$\mathbf{x}^{(t)}, y^{(t)} \leftarrow p(\mathbf{x}, y)$$

    subject to $y^{(t)}\ 2\ \{C + 1, \ldots, C + M\}$

  - side information :
    - a single labeled example from each of the M new classes

- Example
  - recognizing a person based on a single picture of them

ZERO-SHOT LEARNING

- Training time
  - data :

    $$\{\mathbf{x}^{(t)}, y^{(t)}\}$$

  - setting :

    $$\mathbf{x}^{(t)}, y^{(t)} \dashleftarrow p(\mathbf{x}, y)$$

    subject to $y^{(t)}\ 2\ \{1, \ldots, C\}$
  - side information :
    - description vector z_c of each of the C classes

- Test time
  - data :

    $$\{\mathbf{x}^{(t)}, y^{(t)}\}$$

  - setting :

    $$\mathbf{x}^{(t)}, y^{(t)} \dashleftarrow p(\mathbf{x}, y)$$

    subject to $y^{(t)}\ 2\ \{C + 1, \ldots, C + M\}$
  - side information :
    - description vector z_c of each of the new M classes

- Example
  - recognizing an object based on a sentence description of it

UNSUPERVISED LEARNING

- Training time
  - data :

    $$\{\mathbf{x}^{(t)}\}$$

  - setting :

    $$\mathbf{x}^{(t)} \leftarrow p(\mathbf{x})$$

- Test time
  - data :

    $$\{\mathbf{x}^{(t)}\}$$

  - setting :

    $$\mathbf{x}^{(t)} \leftarrow p(\mathbf{x})$$

How can we train models “unsupervised”?

How can we train models “unsupervised”?

This is the focus of representation learning
and generative models

How can we train models "unsupervised"?

This is the focus of representation learning

There's an entire c

Top publications

Categories

English

	Publication	h5-index	h5-median
1.	Nature	488	745
2.	IEEE/CVF Conference on Computer Vision and Pattern Recognition	440	689
3.	The New England Journal of Medicine	434	897
4.	Science	409	633
5.	Nature Communications	375	492
6.	The Lancet	368	678
7.	Neural Information Processing Systems	337	614
8.	Advanced Materials	327	420
9.	Cell	320	482
10.	International Conference on Learning Representations	304	584

ICLR has become one of the top CS and Engineering (not just AI) publication although it just started in 2013.

In fact top-10 in ALL OF SCIENCE

How

This is th

There's an entire

ICLR

ICLR has become one of the top CS and Engineering (not just AI) publication although it just started in 2013.

In fact top-10 in ALL OF SCIENCE

International Conference on Learning Representations

h5-index:304 h5-median:584

#2 Artificial Intelligence
#4 Engineering & Computer Science

Title / Author	Cited by
An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. A Dosovitskiy, L Beyer, A Kolesnikov, D Weissenborn, X Zhai, ... ICLR	38519
Decoupled Weight Decay Regularization. I Loshchilov, F Hutter ICLR (Poster)	18046
Measuring and Improving the Use of Graph Information in Graph Neural Networks. Y Hou, J Zhang, J Cheng, K Ma, RTB Ma, H Chen, MC Yang ICLR	7849
ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. Z Lan, M Chen, S Goodman, K Gimpel, P Sharma, R Soricut ICLR	7127
Large Scale GAN Training for High Fidelity Natural Image Synthesis. A Brock, J Donahue, K Simonyan ICLR	5675
DARTS: Differentiable Architecture Search. H Liu, K Simonyan, Y Yang ICLR (Poster)	4888
LoRA: Low-Rank Adaptation of Large Language Models. EJ Hu, Y Shen, P Wallis, Z Allen-Zhu, Y Li, S Wang, L Wang, W Chen ICLR	4881
Deformable DETR: Deformable Transformers for End-to-End Object Detection. X Zhu, W Su, L Lu, B Li, X Wang, J Dai ICLR	4444
BERTScore: Evaluating Text Generation with BERT. T Zhang, V Kishore, F Wu, KQ Weinberger, Y Artzi ICLR	4143
ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. K Clark, MT Luong, QV Le, CD Manning ICLR	3903

English

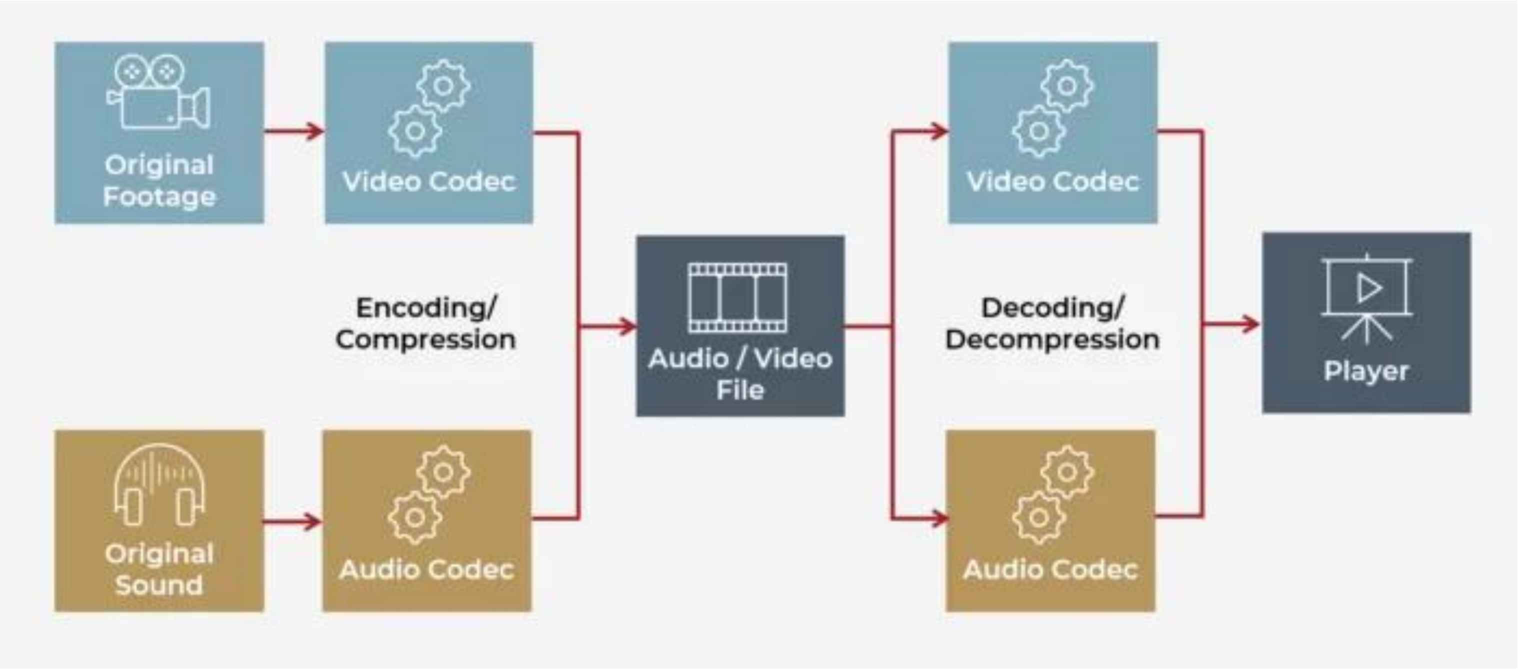
ndex	h5-median
8	745
0	689
4	897
9	633
5	492
8	678
7	614
7	420
0	482
4	584

Warning

I might use the terms “latent”, “embedding”, “representation”, “feature” interchangeably.

Motivation (kind of): Compression

- The idea is similar to compression (signal processing) or hashing (data structures):
  - **encode** an image into a smaller vector s.t. you can **decode** it back to its original form
    - Example: images, audio, video are stored in a compressed form on your computer using compression algorithms like JPEG, MP3, MPEG etc. The computer has software to decode it back so that you can view it (everytime you "open" a JPEG file to view an image, the decoder runs and converts code to RGB)



Motivation (kind of): Compression

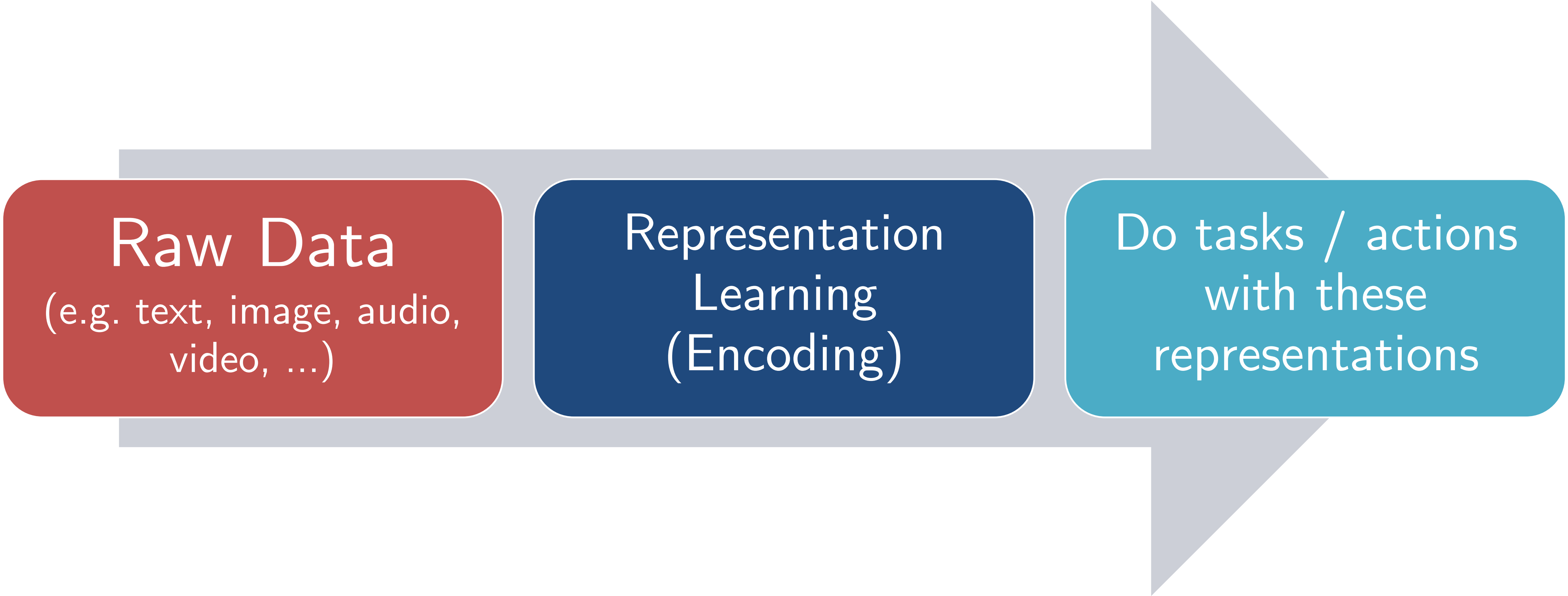
- The idea is similar to compression (signal processing) or hashing (information theory):
  - encode an image into a smaller vector s.t. you can **decode** it back to its original form
    - Example: images, audio, video are stored in a compressed form on your computer using compression algorithms like JPEG, MP3, MPEG etc. The computer has software to decode it back so that you can view it (everytime you "open" a JPEG file to view an image, the decoder runs and converts code to RGB)

- **Representation Learning**
  $\sim$ convert inputs automatically into "codes" (called representations/embeddings / features) s.t. the representations are:
  - Useful for downstream tasks (e.g. classification, regression, ...)
  - "explain the data" and are "meaningful"

- Main difference: "meaningful" representation spaces to do "tasks"

  (The goal for compression is only efficient storage — not data classification/clustering etc.)

Representation Learning Paradigm



Typical Goals for Representations

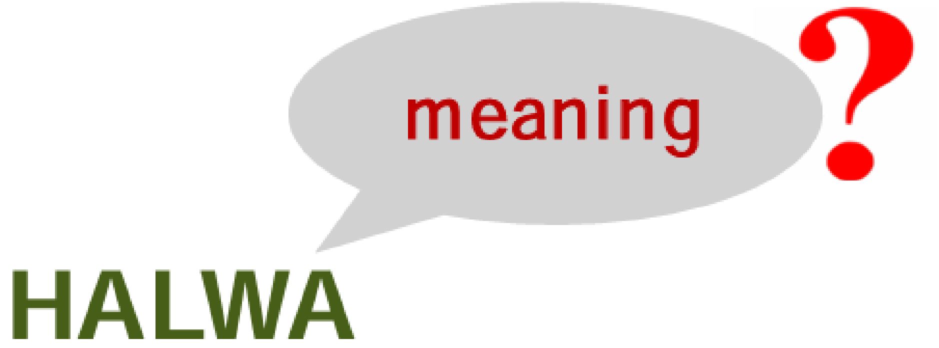
Similar representations for similar concepts

Food

Source: Pankaj Gupta (LMU Munich)

Typical Goals for Representations

A Semblance of "Context" should be encoded ...



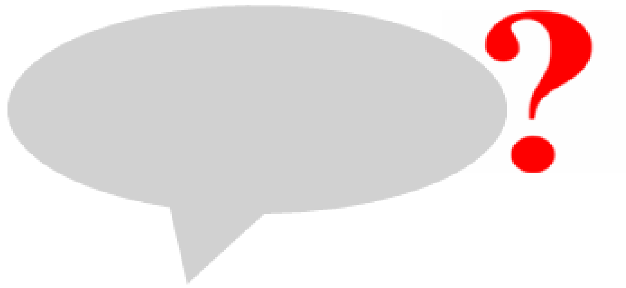
**If you know the answer,
don't share it with the class yet.**

People from lands between Greece and India
might know the answer ...

Source: Pankaj Gupta (LMU Munich)

Typical Goals for Representations

A Semblance of “Context” should be encoded ...



I am very ***hungry***, I will ***eat*** **HALWA**

Source: Pankaj Gupta (LMU Munich)

Typical Goals for Representations

A Semblance of "Context" should be encoded ...

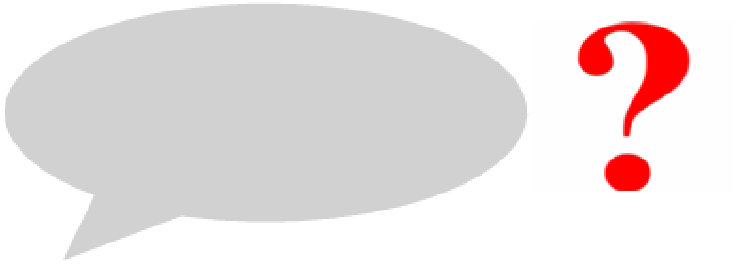
I am very *hungry*, I will *eat* HALWA

If you speak Marathi, this word has two meanings depending on context

Halwa (1): a food item *derived from: Farsi*
Halwa (2): (an instruction to) move (something) *derived from: Sanskrit*

Typical Goals for Representations

A Semblance of “Context” should be encoded ...

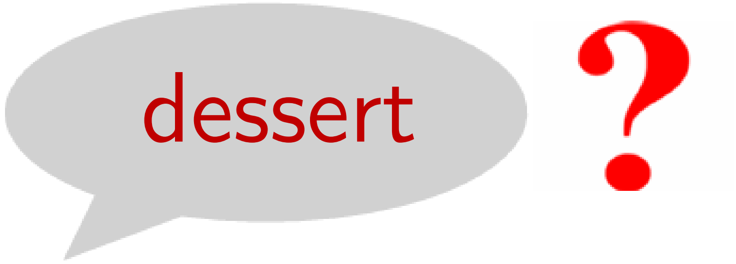


Is it a good idea to *eat* HALWA after a *meal* ?

Source: Pankaj Gupta (LMU Munich)

Typical Goals for Representations

A Semblance of “Context” should be encoded ...



Is it a good idea to *eat* HALWA after a *meal* ?

Typical Goals for Representations

A Semblance of "Context" should be encoded ...

sugary dessert ?

Oh no! I forgot to put sugar in the HALWA

Typical Goals for Representations

Parts, properties, attributes, ontology ?

• "bird"	has	"wing", "beak", "feathers"
• "bird"	can	"fly"
• "bird"	is under category	"animal"
• "bird"	has subcategories	"eagle", "peacock", "sparrow", "seagull", "pigeon"

Source: Pankaj Gupta (LMU Munich)

Representation Learning is a Philosophy for Learning

Key assumptions in this philosophy:

- You can convert a high-dimensional input space into a low-dimensional representation space
    - Example: RGB images → 100 dim vectors
- A good representation space will have a "structure"
    - Example: Similarity, Symmetry, Relations will be easy to understand
    - Why? So that we can do arithmetic in representation space to do tasks
- Representations can be learned from data
- Representations can be leveraged for doing tasks

Parallel Work in Cog.Sci.

Trends in Cognitive Sciences

CellPress

Volume 28, Issue 9, September 2024, Pages 844-856

Review

Why concepts are (probably) vectors

Steven T. Piantadosi [1,2], Dyana C.Y. Muller [2], Joshua S. Rule [1], Karthikeya Kaushik [1], Mark Gorenstein [2], Elena R. Leib [1], Emily Sanford [1]

For decades, cognitive scientists have debated what kind of representation might characterize human concepts. Whatever the format of the representation, it must allow for the computation of varied properties, including similarities, features, categories, definitions, and relations. It must also support the development of theories, *ad hoc* categories, and knowledge of procedures. Here, we discuss why vector-based representations provide a compelling account that can meet all these needs while being plausibly encoded into neural architectures. This view has become especially promising with recent advances in both large language models and vector symbolic architectures. These innovations show how vectors can handle many properties traditionally thought to be out of reach for neural models, including compositionality, definitions, structures, and symbolic computational processes.

Highlights

Modern language models and vector-symbolic architectures show that vector-based models are capable of handling the compositional, structured, and symbolic properties required for human concepts.

Vectors are also able to handle key phenomena from the psychology, including computation of features and similarities, reasoning about relations and analogies, and representation of theories.

Language models show how vector representation of word semantics and sentences can interface between concepts and language, as seen in definitional theories of concepts or *ad hoc* concepts.

The idea of Church encoding, from logic, allows us to understand how meaning can arise in vector-based or symbolic systems.

By combining these recent computational results with classic findings in psychology, vector-based models provide a compelling account of human conceptual representation.

Representation Learning is a Philosophy for Learning

Key assumptions in this philosophy:

- You can convert a high-dimensional input space into a low-dimensional representation space
  - Example: RGB images → 100 dim vectors
- A good representation space will have a "structure"
  - Example: Similarity, Symmetry, Relations will be easy to understand
  - Why? So that we can do arithmetic in representation space to do tasks
- Representations can be learned from data
- Representations can be leveraged for doing tasks

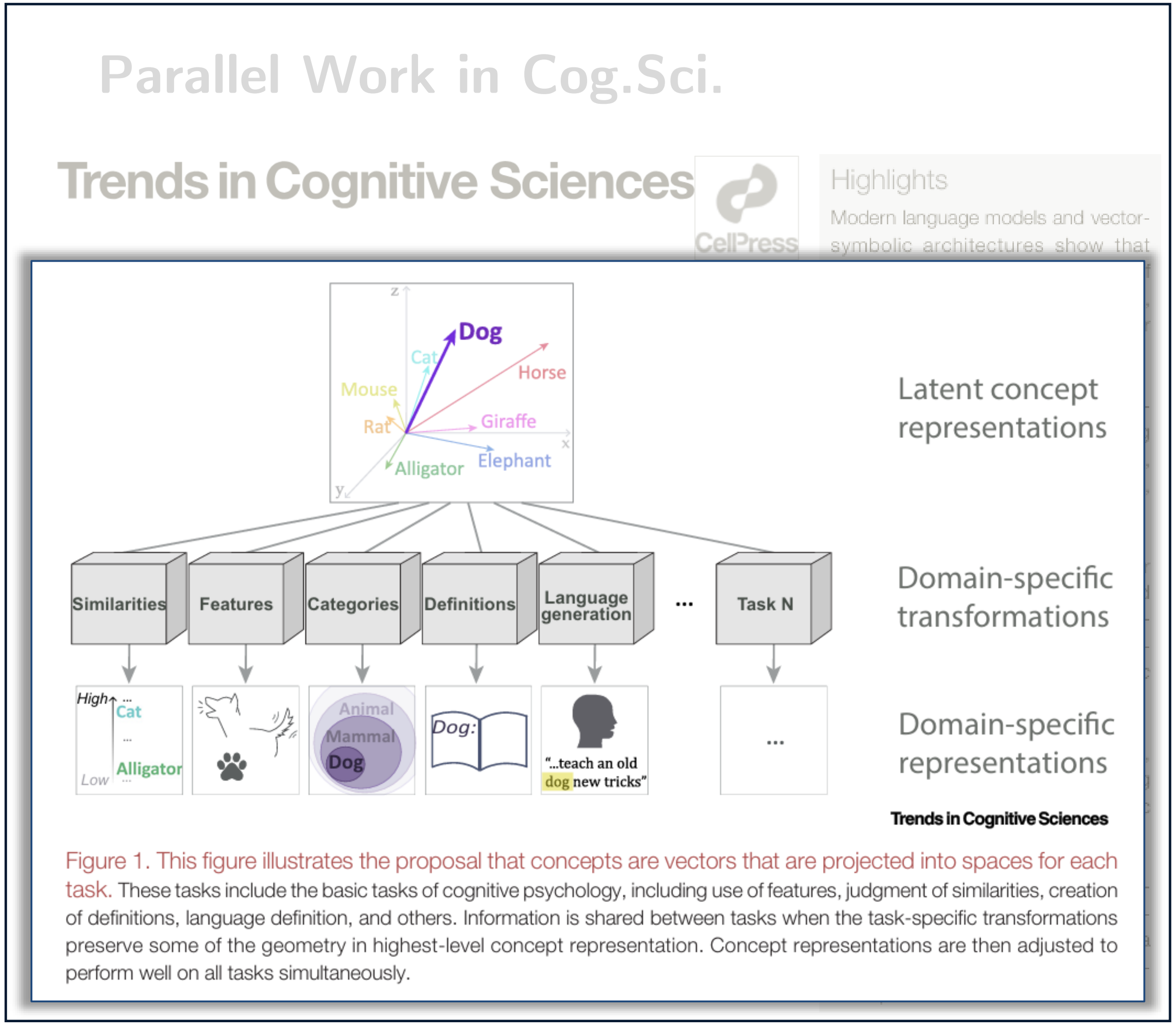


Figure 1. This figure illustrates the proposal that concepts are vectors that are projected into spaces for each task. These tasks include the basic tasks of cognitive psychology, including use of features, judgment of similarities, creation of definitions, language definition, and others. Information is shared between tasks when the task-specific transformations preserve some of the geometry in highest-level concept representation. Concept representations are then adjusted to perform well on all tasks simultaneously.

Ok whatever. Tell us how it works ...

Types of Modeling (Probabilistic Interpretation)

Data: x; Label: y

"cat"

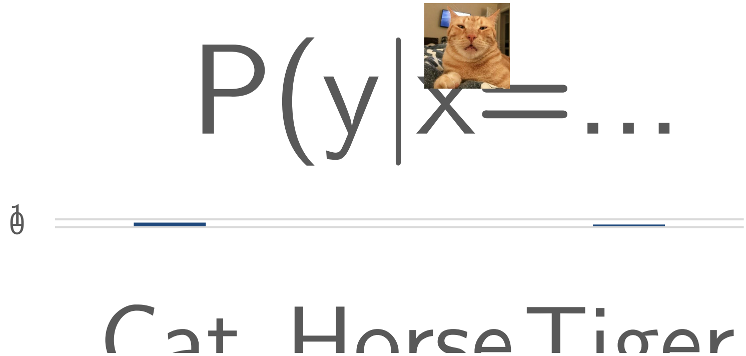
Density Function: $p(x)$

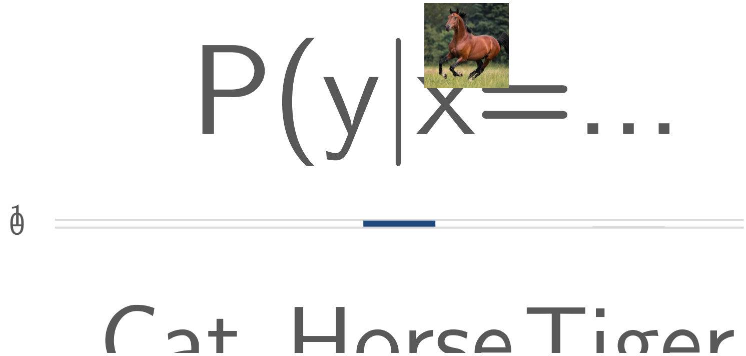
$$\int_X p(x)\, dx = 1$$

(probabilities of all inputs sum to 1)

Discriminative Model

Learn Prob. Dist. $P(y|x)$





$\forall x, \sum_c P(y=c|x) = 1$

Types of Modeling (Probabilistic Interpretation)

Data: x; **Label: y**

"cat"

Density Function: $p(x)$

$$\int_X p(x)\,dx = 1$$

(probabilities of all inputs sum to 1)

Generative Model

Learn Marginal Prob. Dist. $P(x)$

Conditional Generative Model

Learn conditional probability $P(x|y)$

$$\underset{\text{Conditional Generative Model}}{\boxed{P(x \mid y)}} = \frac{\overset{\text{Discriminative Model}}{\boxed{P(y \mid x)}}}{\underset{\text{Prior over labels}}{\boxed{P(y)}}}\overset{\text{(Unconditional) Generative Model}}{\boxed{P(x)}}$$

Types of Modeling (Probabilistic Interpretation)

Data: x; **Label: y**

"cat"

Density Function: $p(x)$

$$\int_X p(x)\,dx = 1$$

(probabilities of all inputs sum to 1)

- **Discriminative Model**

  Learn Prob. Dist. $P(y|x)$

- **Generative Model**

  Learn Marginal Prob. Dist. $P(x)$

- **Conditional Generative Model**

  Learn conditional probability $P(x|y)$

Types of Modeling (Probabilistic Interpretation)

APPLICATIONS

Classification, Regression,
Representation Learning
(with labels)

- **Discriminative Model**

  Learn Prob. Dist. $P(y|x)$

- **Generative Model**

  Learn Marginal Prob. Dist. $P(x)$

- **Conditional Generative Model**

  Learn conditional probability $P(x|y)$

Types of Modeling (Probabilistic Interpretation)

APPLICATIONS

- **Discriminative Model**

  Learn Prob. Dist. $P(y|x)$

- **Generative Model**

  Learn Marginal Prob. Dist. $P(x)$

- **Conditional Generative Model**

  Learn conditional probability $P(x|y)$

Data Generation
Outlier Detection
Representation Learning
(without labels)

Types of Modeling (Probabilistic Interpretation)

APPLICATIONS

- **Discriminative Model**

  Learn Prob. Dist. $P(y|x)$

- **Generative Model**

  Learn Marginal Prob. Dist. $P(x)$

- **Conditional Generative Model**

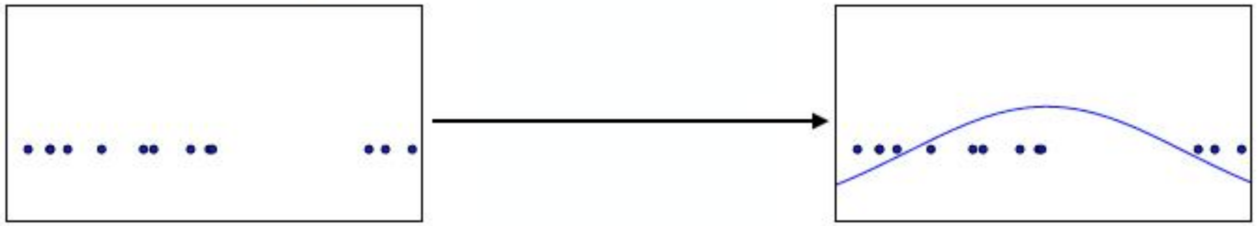
  Learn conditional probability $P(x|y)$

Machine Translation
Text-to-image generation

(pretty much every "GenAI" product you see is a conditional generative model)

Generative Modeling

- Density estimation



- Sample generation

Training examples Model samples

(Goodfellow 2016)

Why Generative Models?

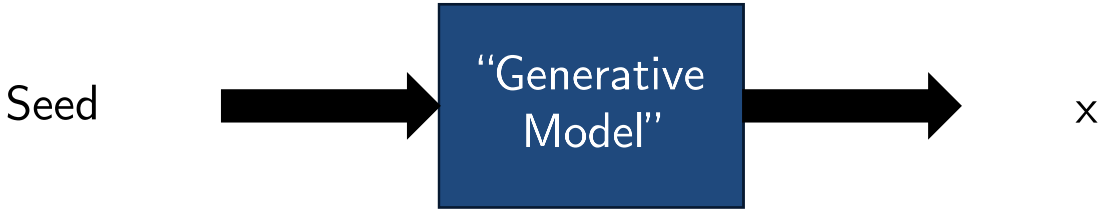
- **We've only seen discriminative models so far**
  - Given an image **X**, predict a label **Y**
  - Estimates **P(Y|X)**

- **Discriminative models have several key limitations**
  - Can't model **P(X)**, i.e. the probability of seeing a certain image
  - Thus, can't sample from **P(X)**, i.e. **can't generate new images**

- **Generative models (in general) cope with all of above**
  - Can model **P(X)**
  - Can generate new images

Generative Models

- What's a Generative Model?
  - A model for the probability distribution of data x P(x)
  - A model that can be used to "generate" data marketing term "genAI"



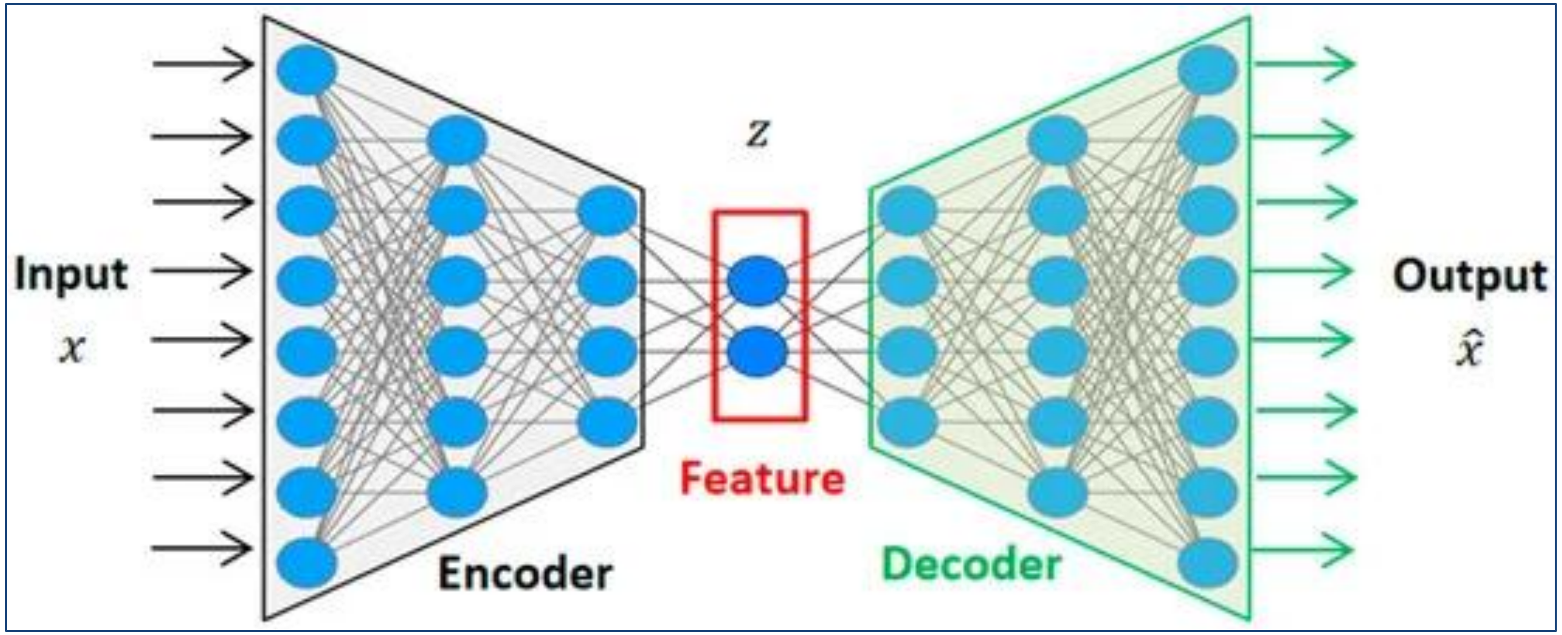
- Generative Models can be *learned*
  - *You are given some observed data X* *(e.g. face images)*
  - *You choose a function (e.g. neural network) to model $P(x;\theta)$ using parameters θ*
  - *You estimate θ s.t. $P(x;\theta)$ best fits the observations X*

Ok whatever. Tell us how it works ...

Let's start simple ...

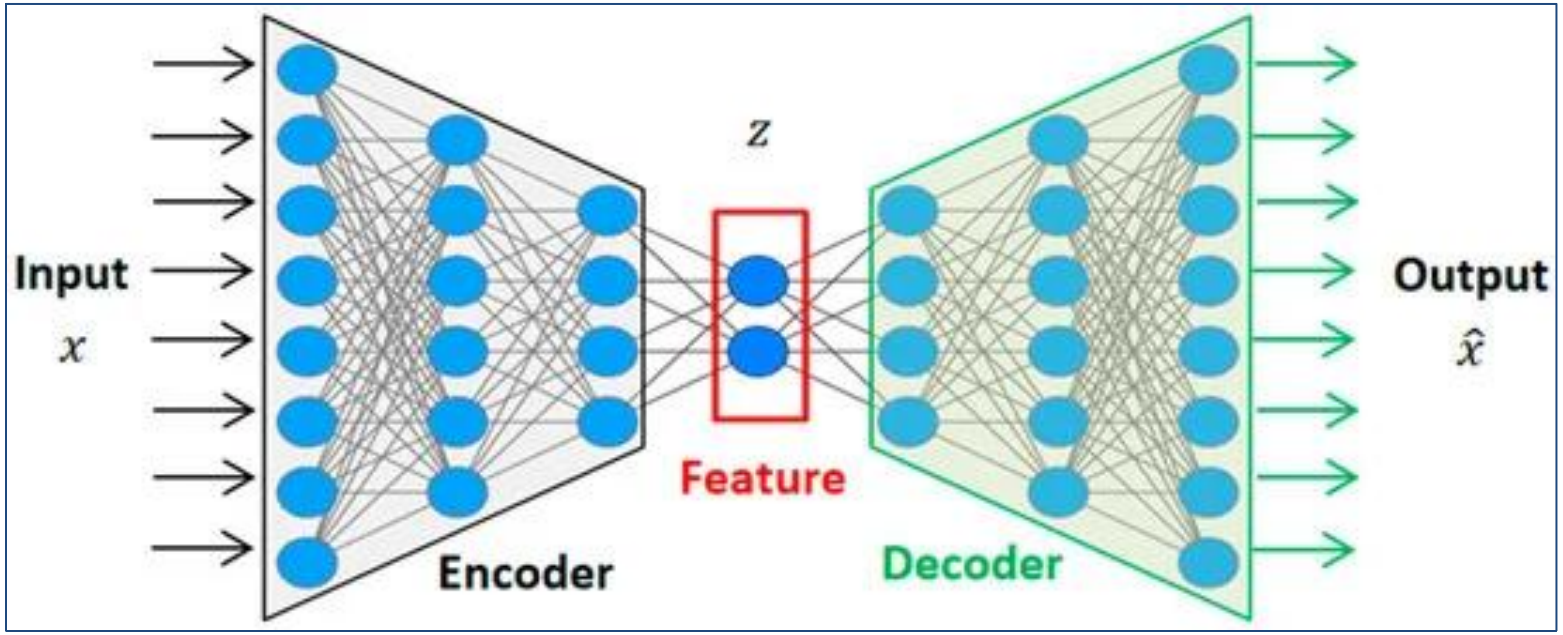
The idea of an "Auto-encoder"

- NN trained to reproduce the input $\hat{x} = F(x)$
- $F()$ is a composition of two functions: encoder $E()$ and decoder $D()$
  - Embedding / Feature / Latent $z = E(x)$
  - Output $\hat{x} = D(z) = D\big(E(x)\big)$



How would you train an autoencoder?

Loss Function?



Autoencoder: Loss Function

- The objective is to minimize the "distance" between x and $\hat{x}$
  - If $d(x, \hat{x}) = 0$ then we get perfect reconstruction

- Mean squared error!

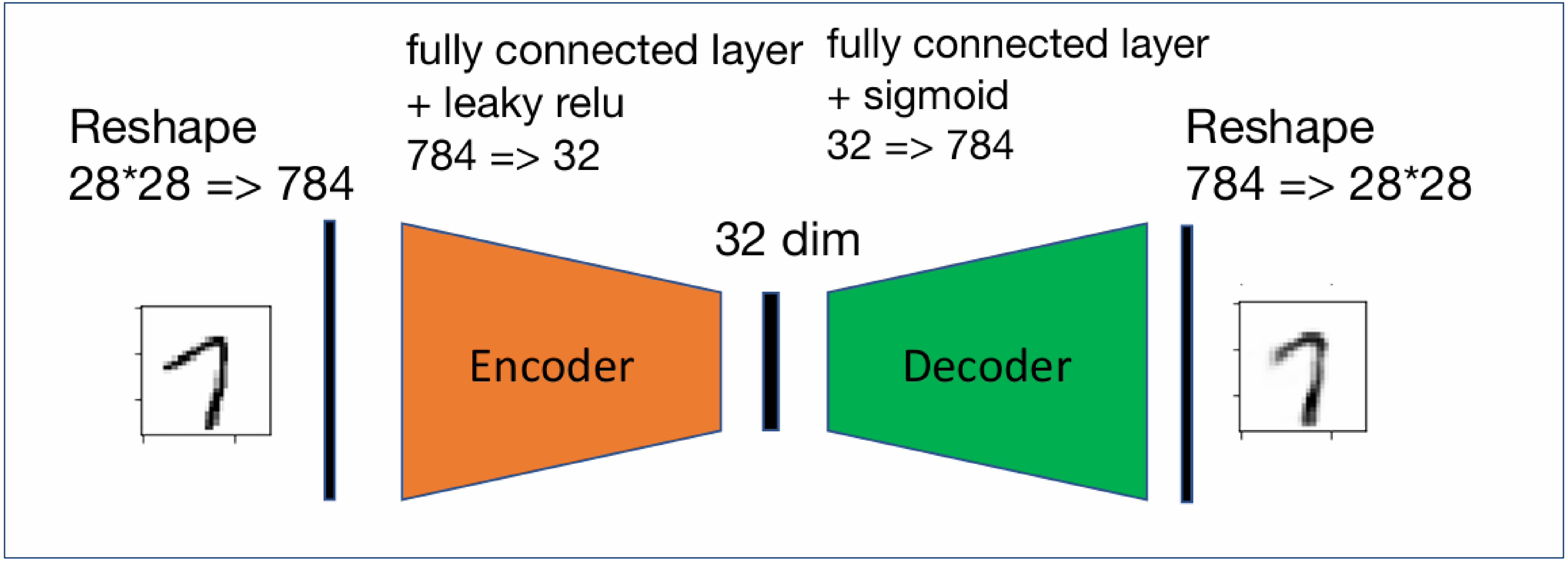
$$l(f(\mathbf{x})) = \frac{1}{2} \sum_k (\widehat{x}_k - x_k)^2$$

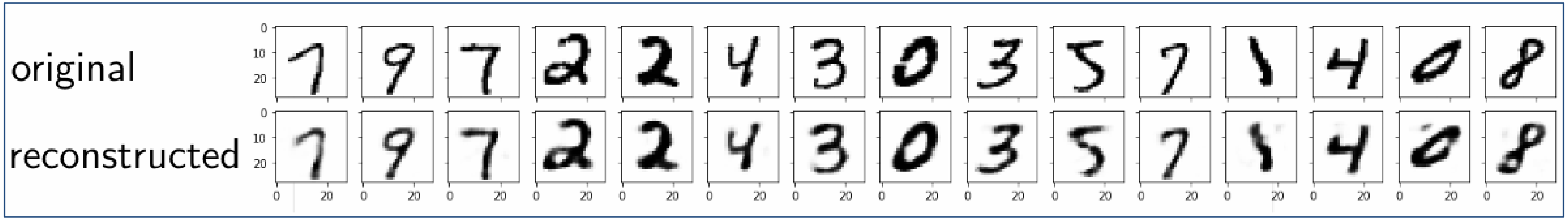
- Cross Entropy (for binary inputs)

$$l(f(\mathbf{x})) = -\sum_k \left(x_k \log(\widehat{x}_k) + (1 - x_k) \log(1 - \widehat{x}_k)\right)$$

- For both cases, gradient is very simple: $\nabla_x L(f(x), x) = \hat{x} - x$

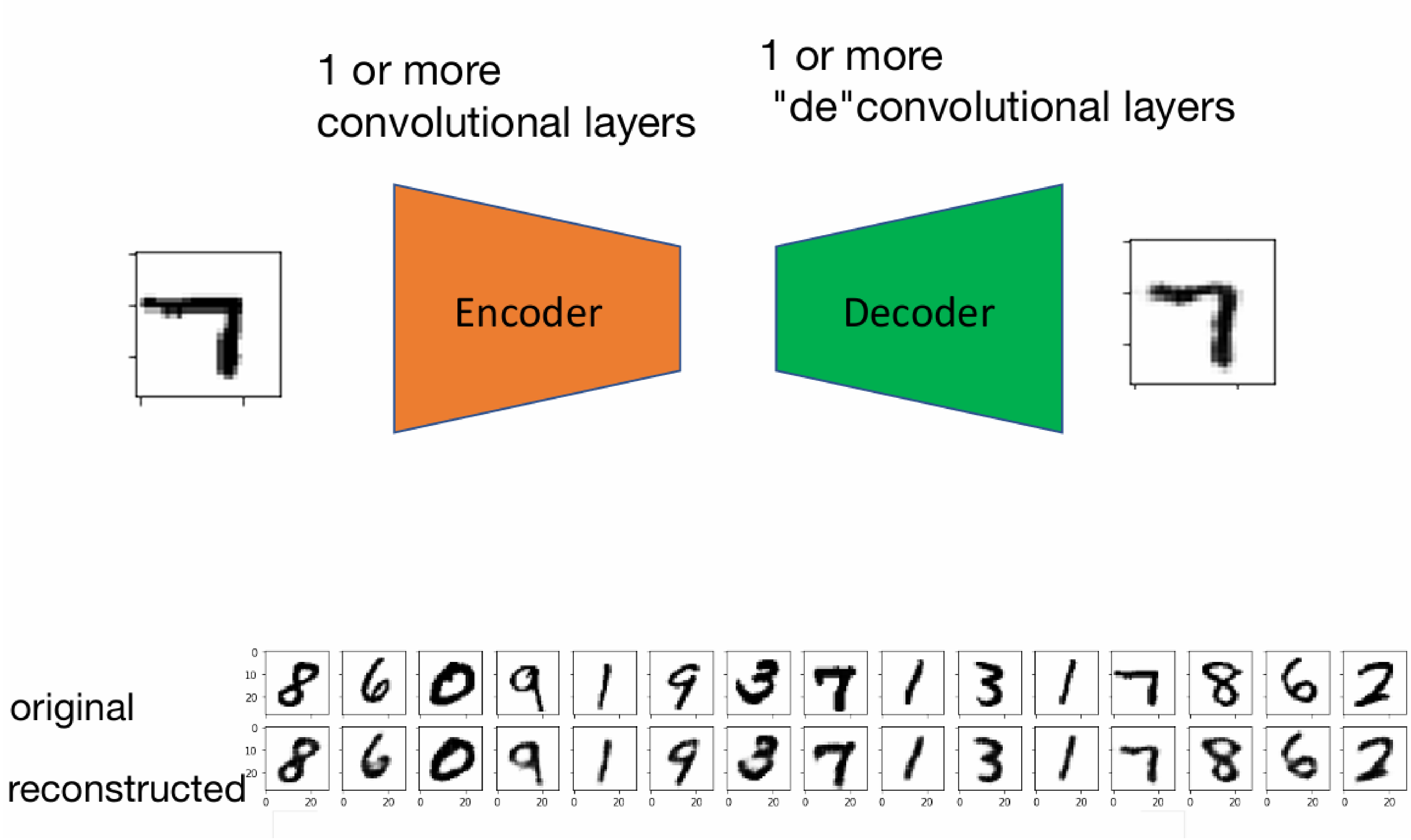
Autoencoder: Simple Example





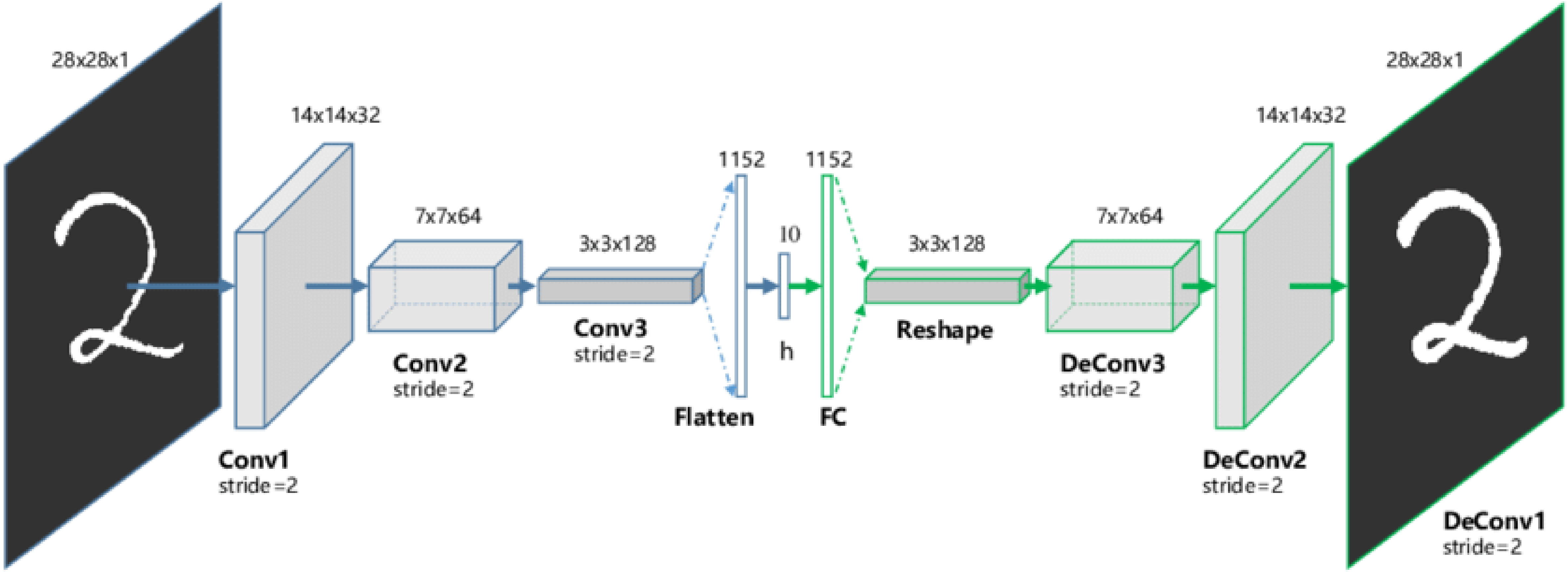
Source: Sebastian Raschka

Convolutional Autoencoder



Source: Sebastian Raschka

Convolutional Autoencoder

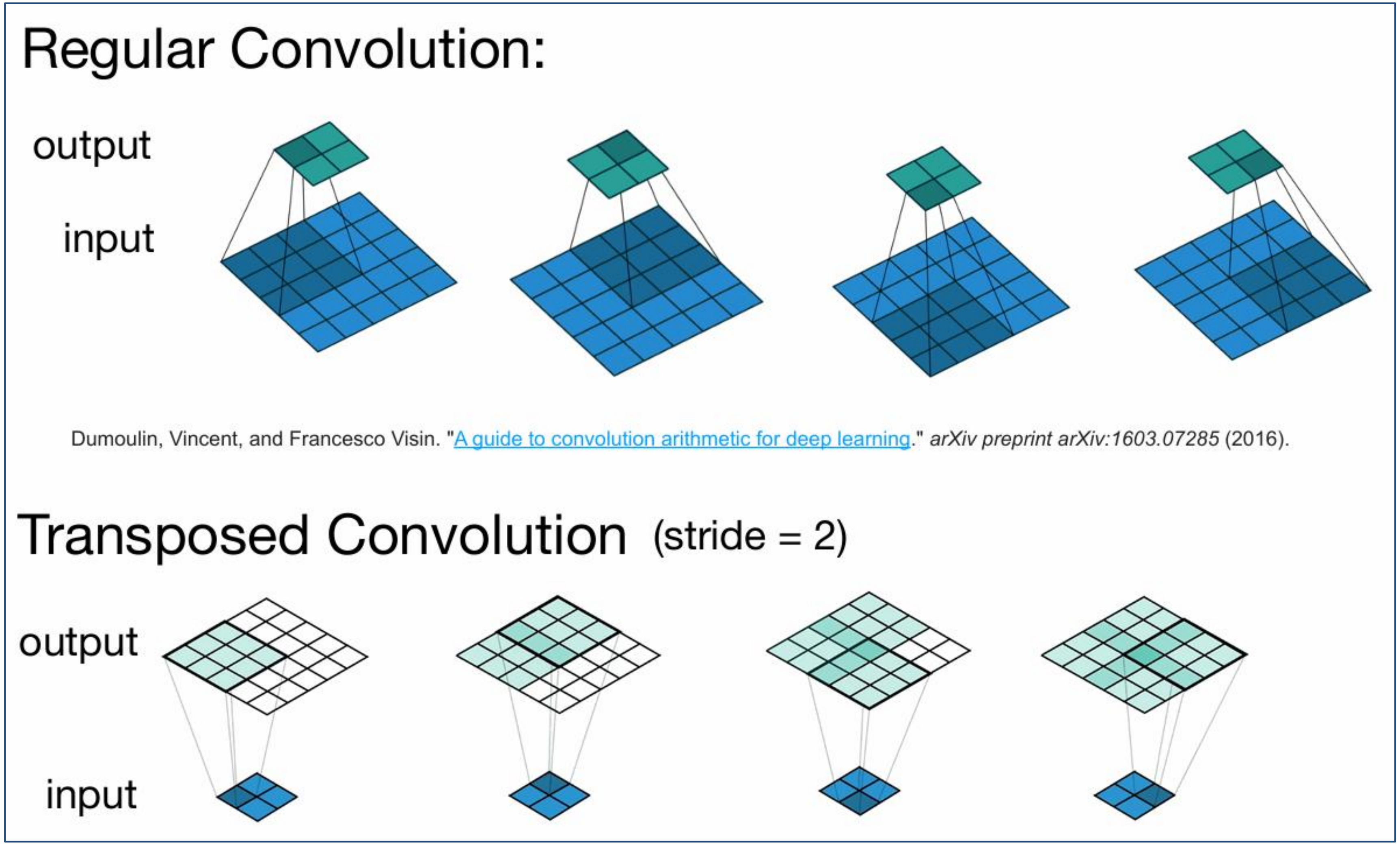


Convolutional Autoencoder: Expand Dimensions? Transposed Convolution!

- The decoder needs to "expand dimensions"
  - Convert a small feature z into a large input x

- Use transposed convolution! A.K.A. fractionally stride convolution
  - Often (incorrectly) called "de"convolution
  - This is an incorrect term because mathematically "deconvolution" is "inverse of convolution"

Convolutional Autoencoder: Expand Dimensions? Transposed Convolution!



Transposed Conv in PyTorch

```python
import torch
```

```python
torch.manual_seed(123)
a = torch.rand(4).view(1, 1, 2, 2)

conv_t = torch.nn.ConvTranspose2d(in_channels=1,
                                  out_channels=1,
                                  kernel_size=(3, 3),
                                  padding=0,
                                  stride=1)

# output = s(n-1)+k-2p = 1*(2-1)+3-2*0 = 4
conv_t(a)
```

```
tensor([[[[-0.2863, -0.2766, -0.1478, -0.3274],
          [-0.3522, -0.5356, -0.1591, -0.2911],
          [-0.3054, -0.4644, -0.3286, -0.2444],
          [-0.2332, -0.2557, -0.1876, -0.3970]]]],
       grad_fn=<ThnnConvTranspose2DBackward>)
```

$$\text{output} \; = s(n-1)+k-2p$$

```python
torch.manual_seed(123)
a = torch.rand(16).view(1, 1, 4, 4)

conv_t = torch.nn.ConvTranspose2d(in_channels=1,
                                  out_channels=1,
                                  kernel_size=(3, 3),
                                  padding=0,
                                  stride=1)

# output = s(n-1)+k-2p = 1*(4-1)+3-2*0 = 6
conv_t(a).size()
```

```
torch.Size([1, 1, 6, 6])
```

```python
torch.manual_seed(123)
a = torch.rand(64).view(1, 1, 8, 8)

conv_t = torch.nn.ConvTranspose2d(in_channels=1,
                                  out_channels=1,
                                  kernel_size=(3, 3),
                                  padding=0,
                                  stride=1)

# output = s(n-1)+k-2p = 1*(8-1)+3-2*0 = 10
conv_t(a).size()
```

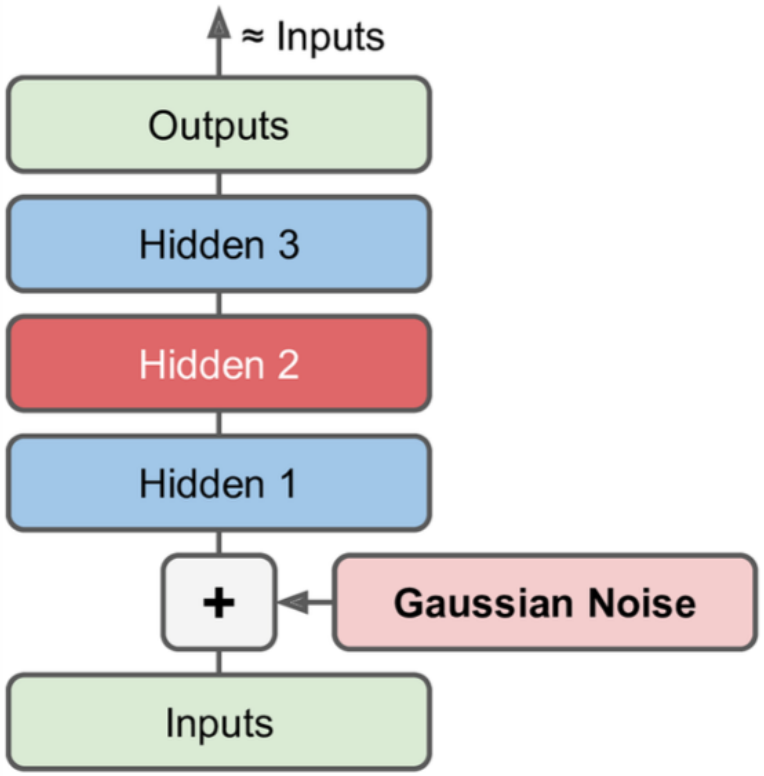
```
torch.Size([1, 1, 10, 10])
```

Denoising Autoencoder

- The input is "noisy" $\tilde{x}$. The expected output is a clean image (denoised image)

- Noise Examples:
  - Gaussian: $\tilde{x} = x + z; \quad z \sim N(0, \sigma^2 I)$
  - Masking: Zero-out some of the components of x (for images, make some pixels 0)
    - Can be random masks
    - Can be square masks

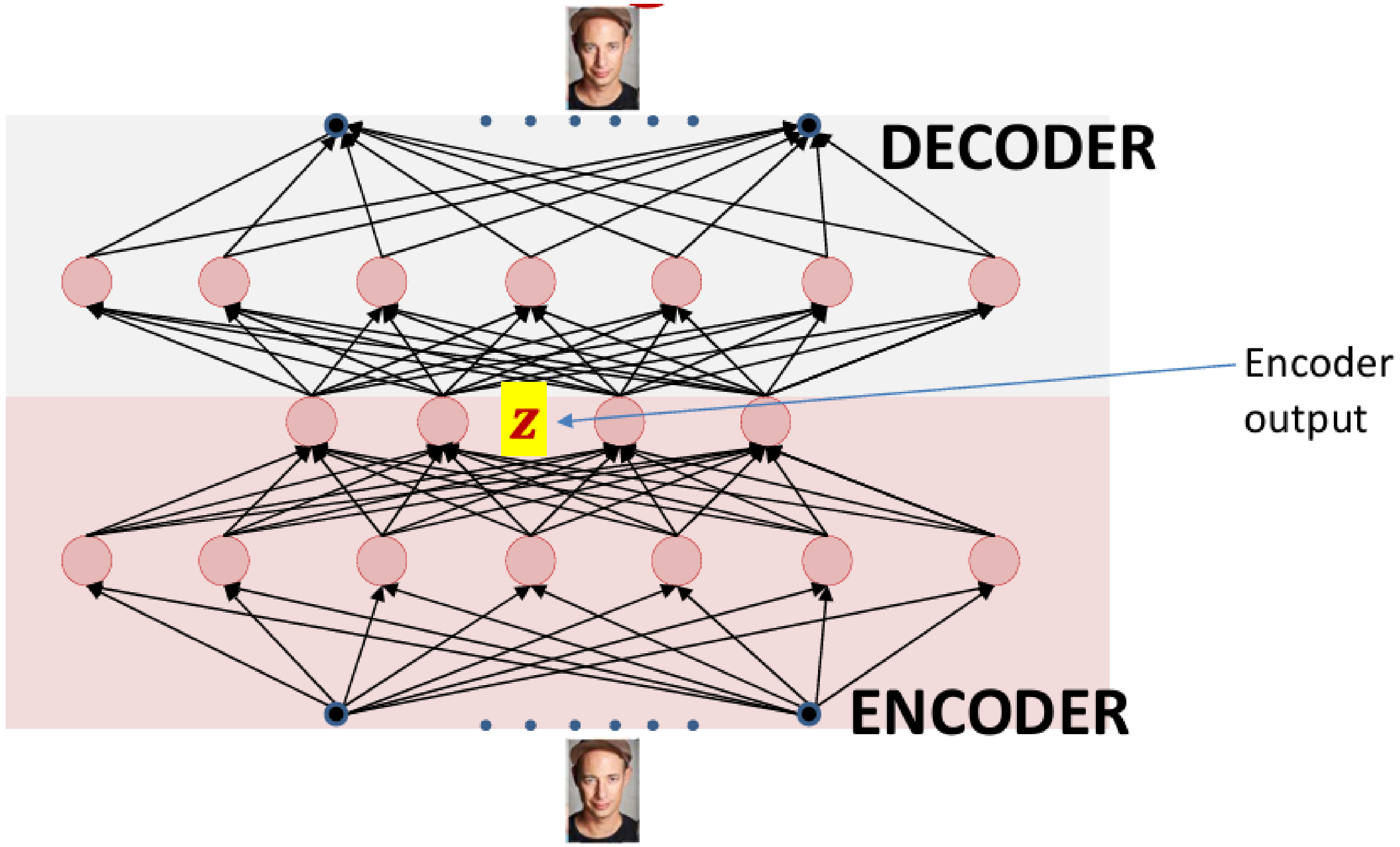
- Adding noise makes representations more robust
  - Expect $D(E((\tilde{x})) = x$ for all z



Vincent, Larochelle, Bengio, Manzagol. ICML 2008

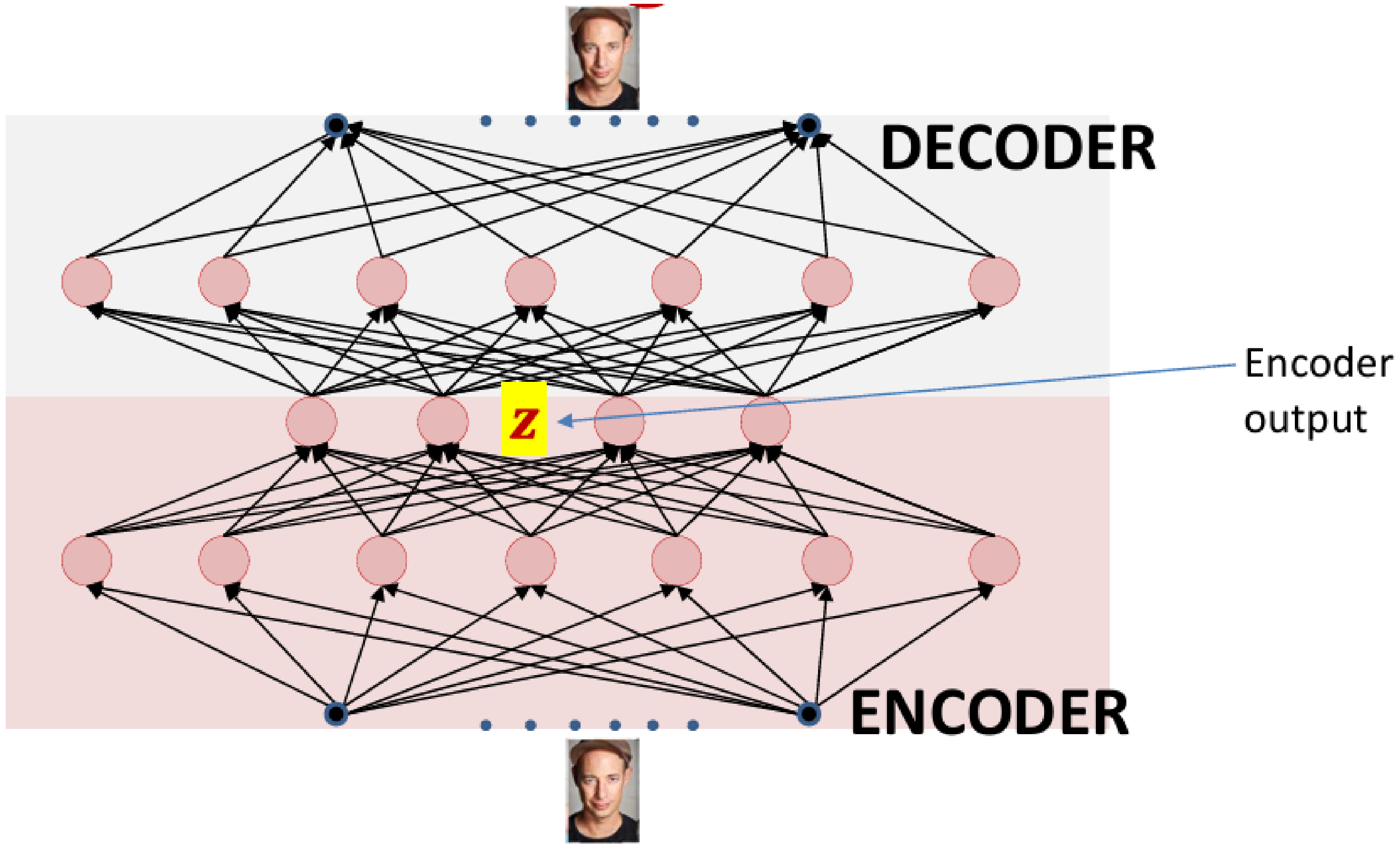
Example: Face Auto-Encoder

Once trained, what can you do with this model?



Example: Face Auto-Encoder

Once trained, what can you do with this model?

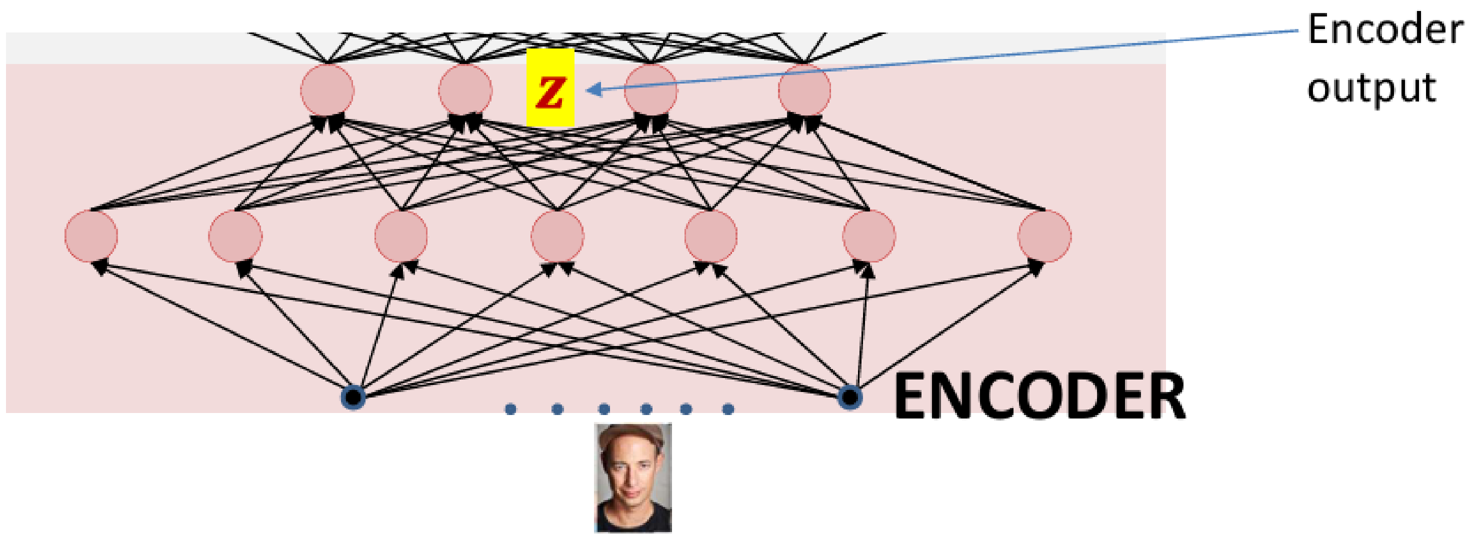


Example: Face Auto-Encoder

Once trained, what can you do with this model?

(1) Encode images into vectors

(*throw away the decoder ...*)



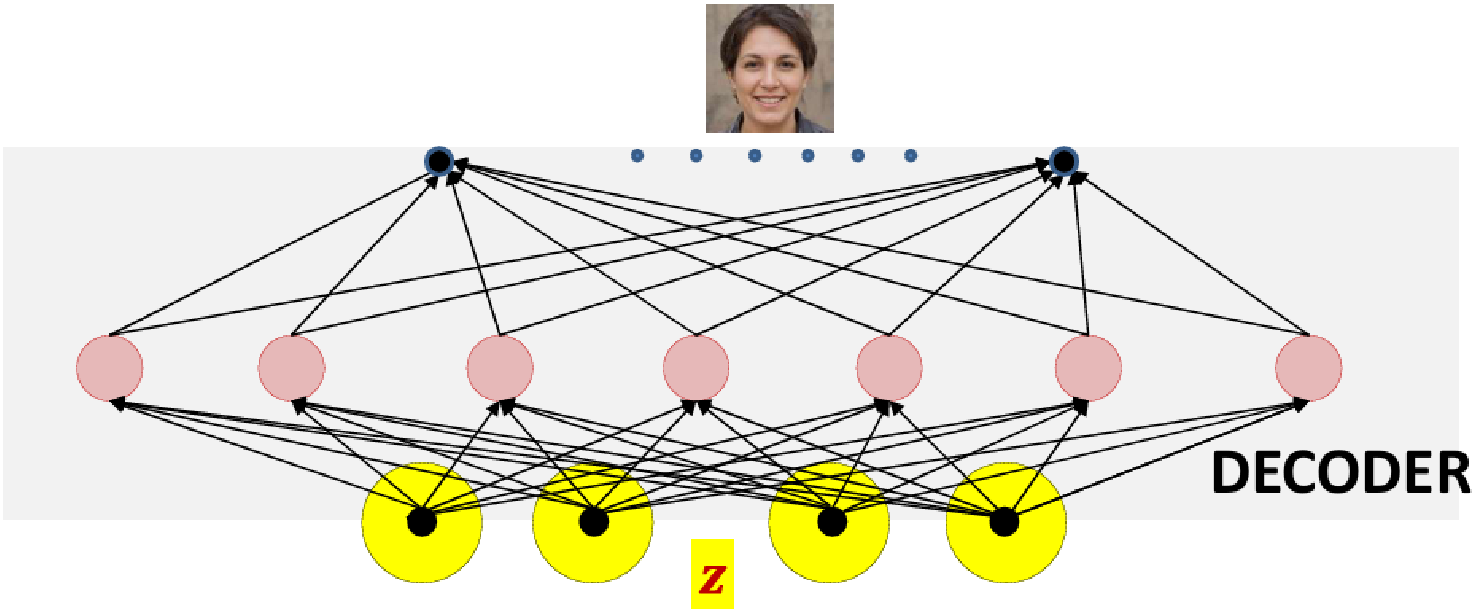
Example: Face Auto-Encoder

Once trained, what can you do with this model?

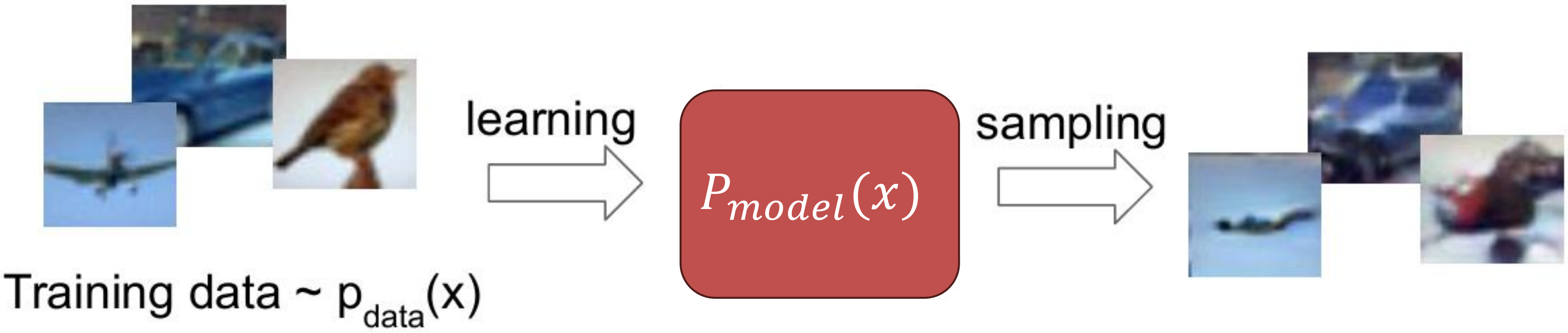
(1) Encode images into vectors

(2) Generate new faces ...

(*throw away the encoder*)



With Generative Models, there are 2 objectives:



Objectives:

1. Learn $p_{model}(x)$ that approximates $p_{data}(x)$
2. Sampling new x from $p_{model}(x)$

figure adapted from Ranjay Krishna

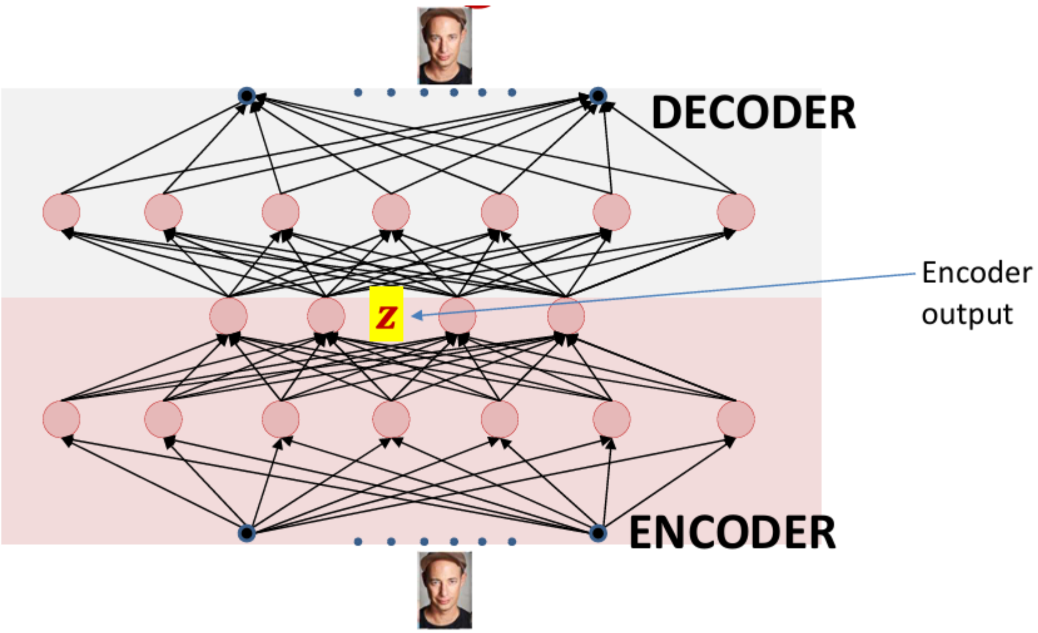
An Auto-Encoder is a Generative Model

Probabilistic Interpretation:

- Encoder $E()$ estimates $P_{\theta_E}(z|x)$
- Decoder $D()$ estimates $P_{\theta_D}(x|z)$
- The marginal $P(x) = \int_z P(x,z)\,dz = \int_z P(z)P(x|z)$

  Bayes/Chain Rule …
- Once the AE is "trained"
  - you get a generative model that generates "x" given a latent code "z"
  - A conditional generative model takes an additional input "y" $P(x|z,y)$
    - E.g. generating images from text y=text, x = image
    - *More on this later …*

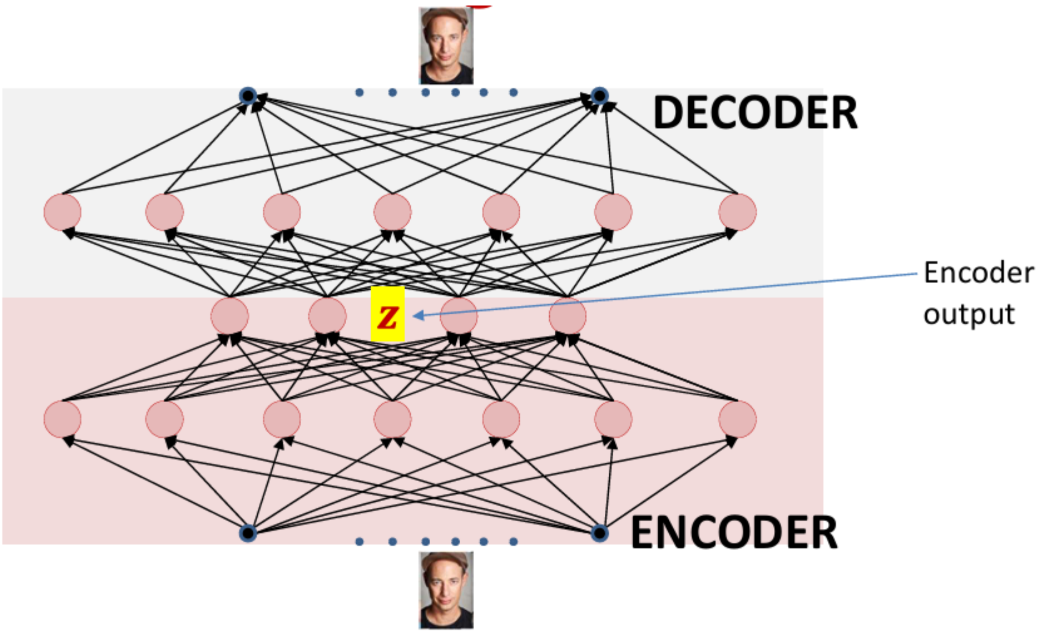
Types of Autoencoders



So far, we have not enforced any “structure” on the latents z

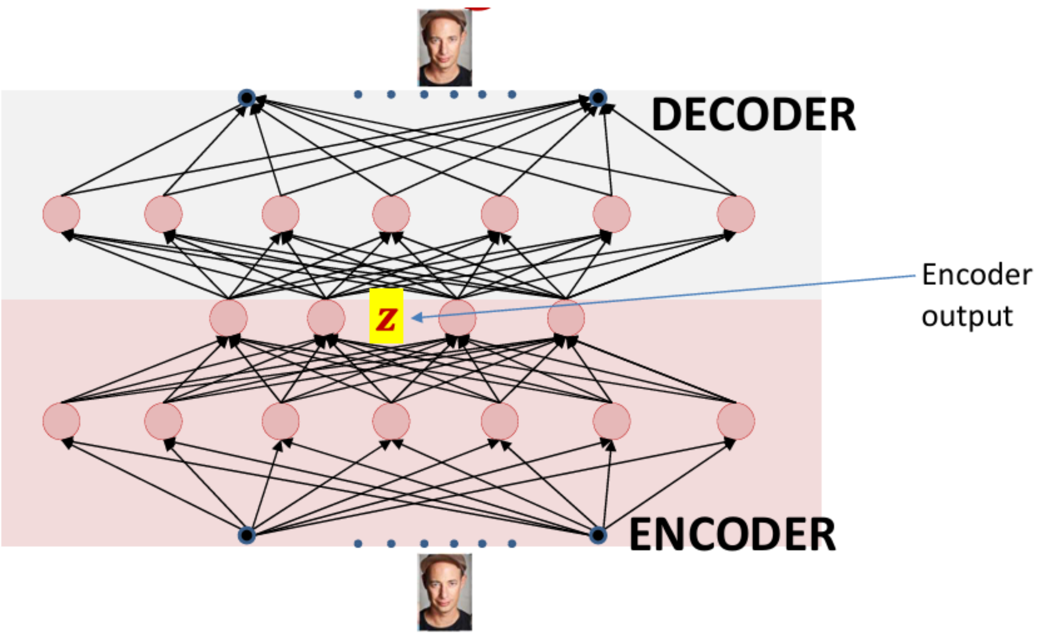
But a structure is desirable

(Remember our motivations / goals for representation learning)



So far, we have not enforced any "structure" on the latents z

We can't generate ***new images*** from D() if we don't understand the z-space

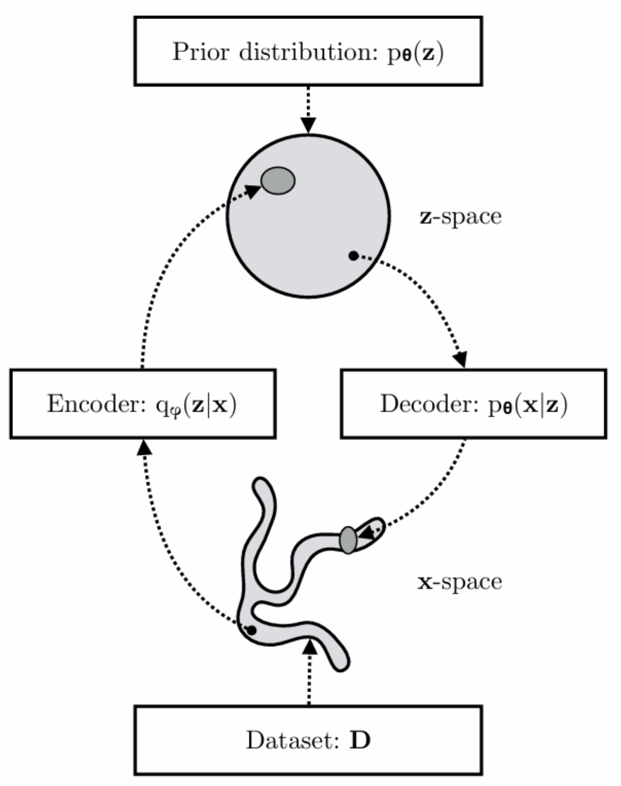


So far, we have not enforced any “structure” on the latents z

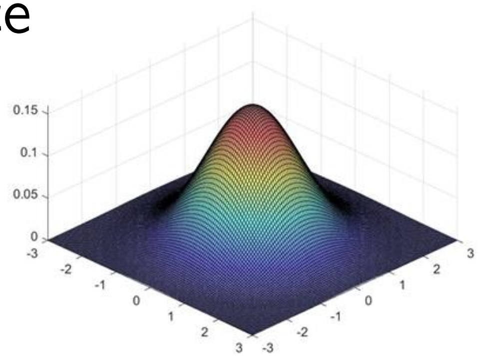
We can’t generate ***new images*** from D() if we don’t understand the z-space

For example, if I ask you to generate a “face with beard, glasses, brown hair”
which z would you choose?

VAE: Variational Autoencoder

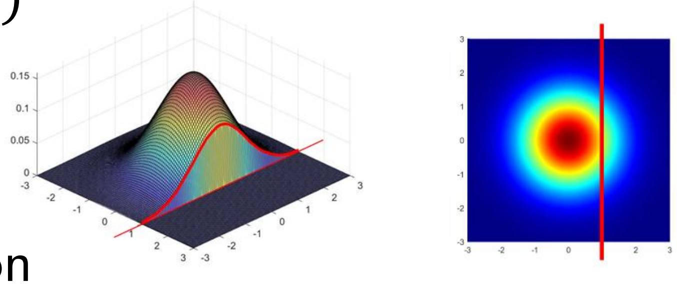



- Force a "prior" distribution on the latent space
  - Example: Gaussian $N(0, I)$

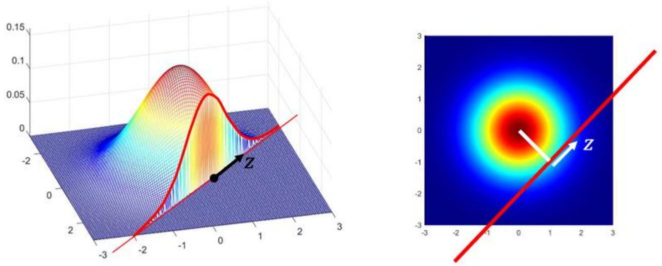


- Gaussians are nice because they are perfectly symmetrical in every dimension
  - Isotropic (covariance matrix is identity I)
  - Dimensions are independent,

    i.e. $\mathrm{P}(z_1|z_2) = P(z_1) = N(0, I)\ \forall z_1, z_2$



  - Property holds for any linear combination of z elements
    - i.e. $P(z_1|az_2 + bz_3) = N(0, I)$



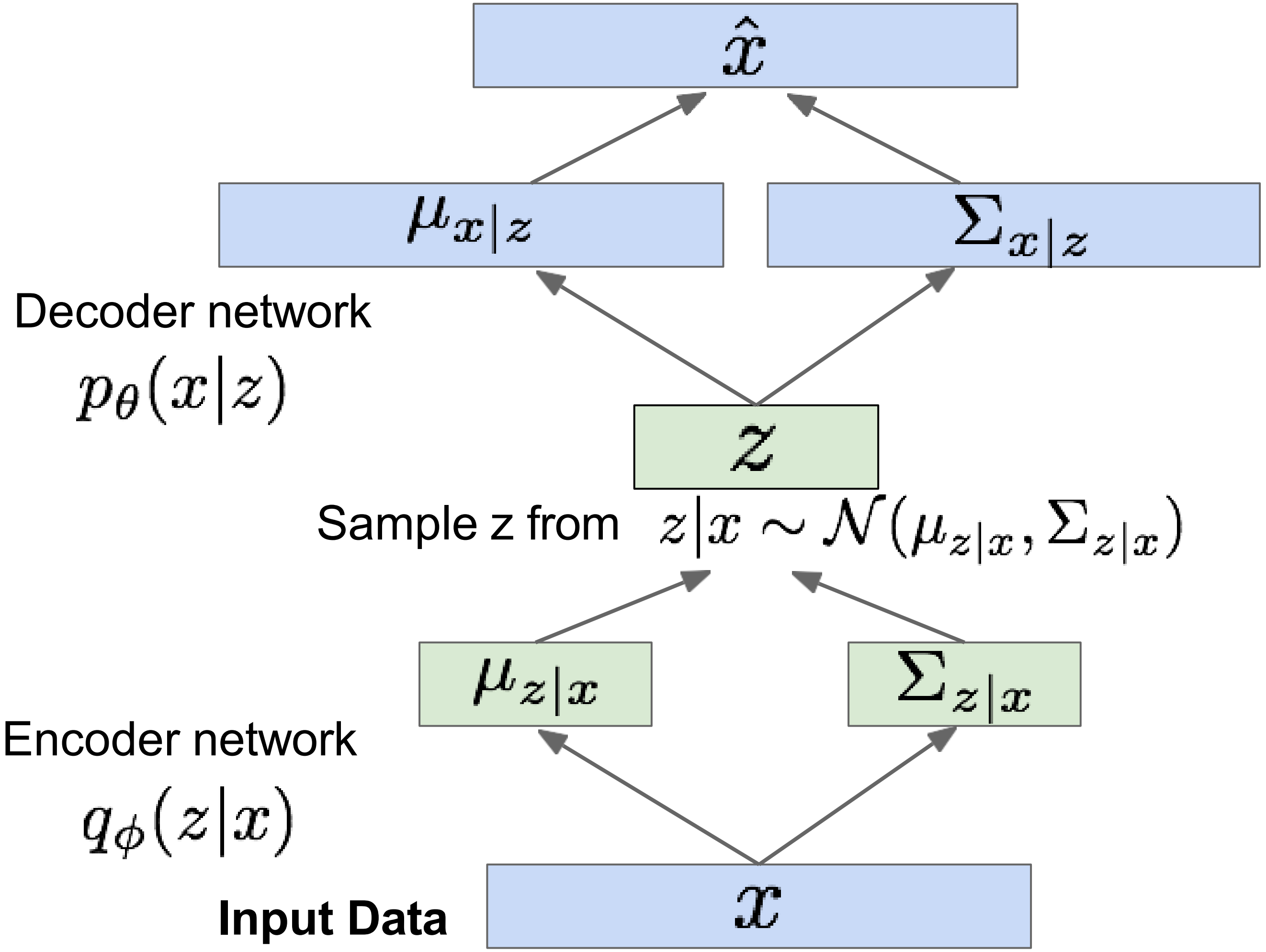
Kingma and Welling. Auto-Encoding Variational Bayes. ICLR 2014

Variational Autoencoders

Key idea:

Force a Gaussian distribution on the latent space z

Details: take my NN class!



Variational Autoencoders: Generating Data!

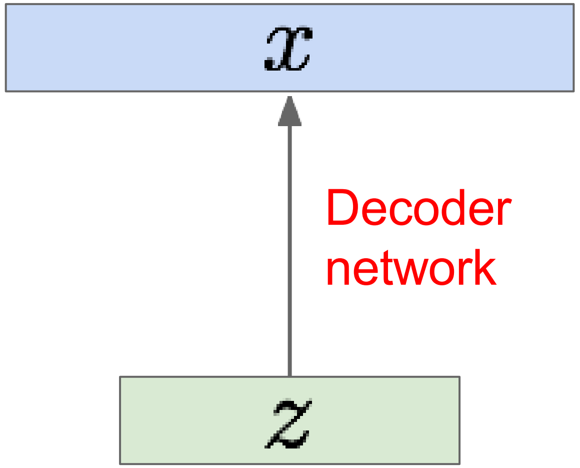
Our assumption about data generation process

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$z^{(i)} \sim p_{\theta^*}(z)$



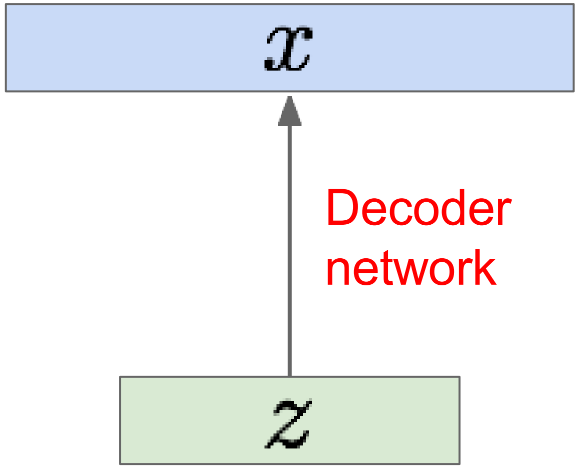
Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Generating Data!

Our assumption about data generation process

Sample from true conditional

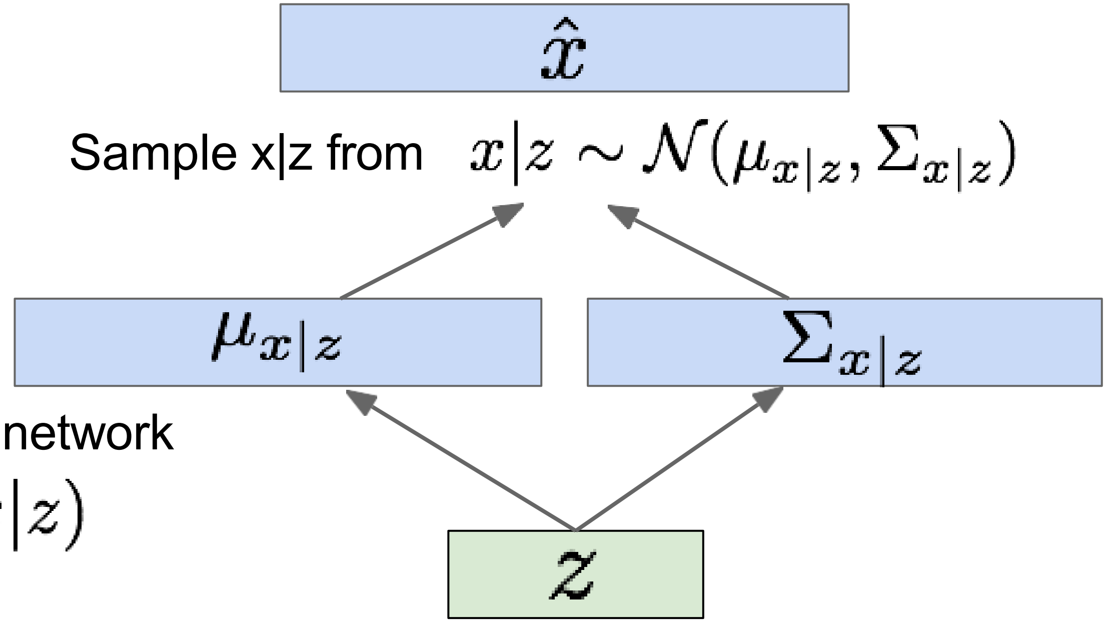
$p_{\theta^*}(x \mid z^{(i)})$



Sample from true prior

$z^{(i)} \sim p_{\theta^*}(z)$

Now given a trained VAE:
use decoder network & sample z from prior!



Sample x|z from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$

Decoder network

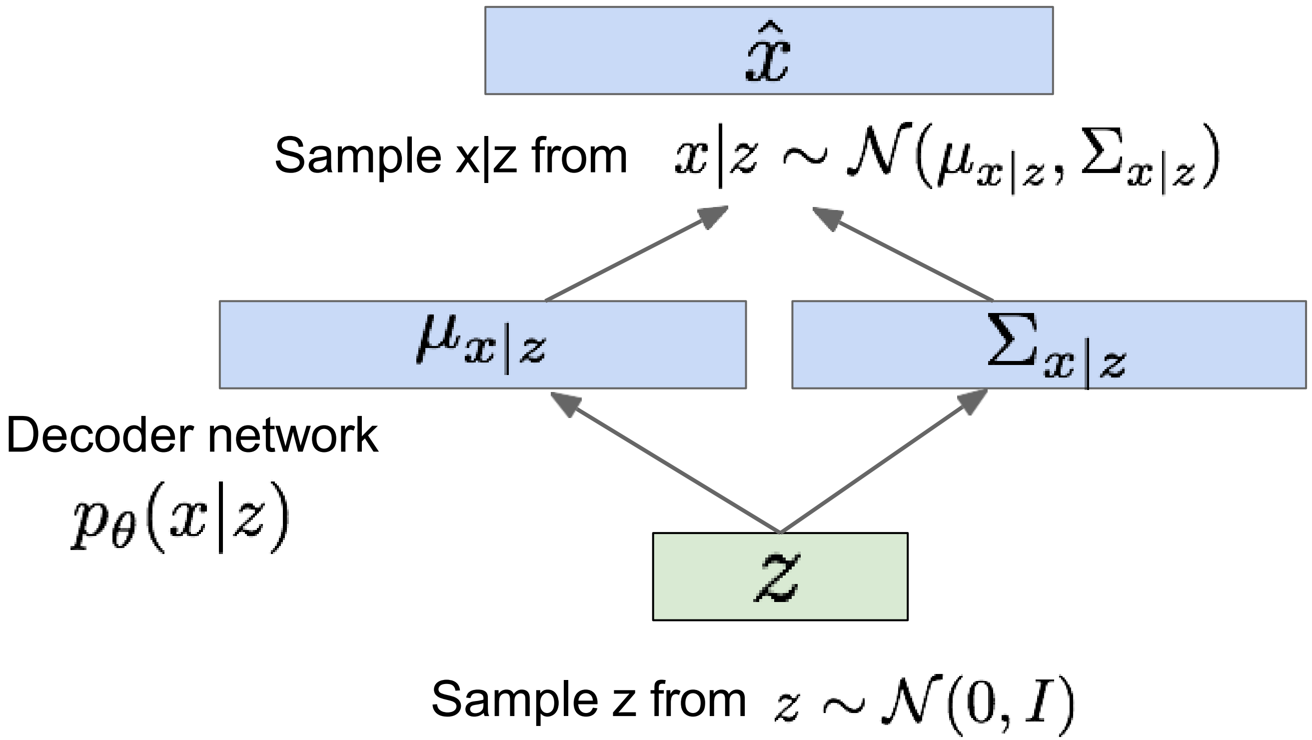
$p_\theta(x|z)$

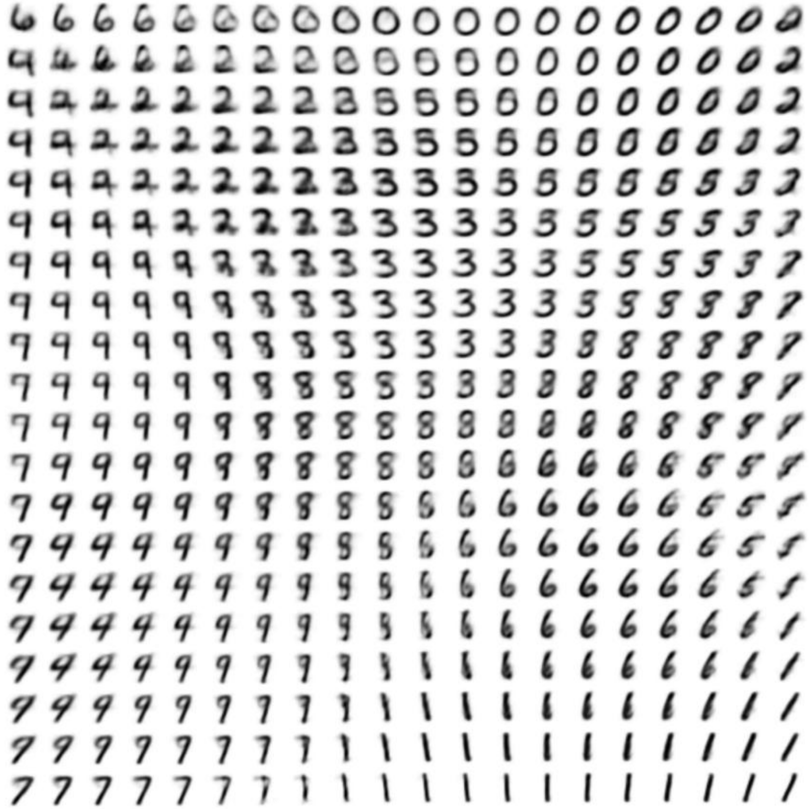
Sample z from $z \sim \mathcal{N}(0, I)$

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!

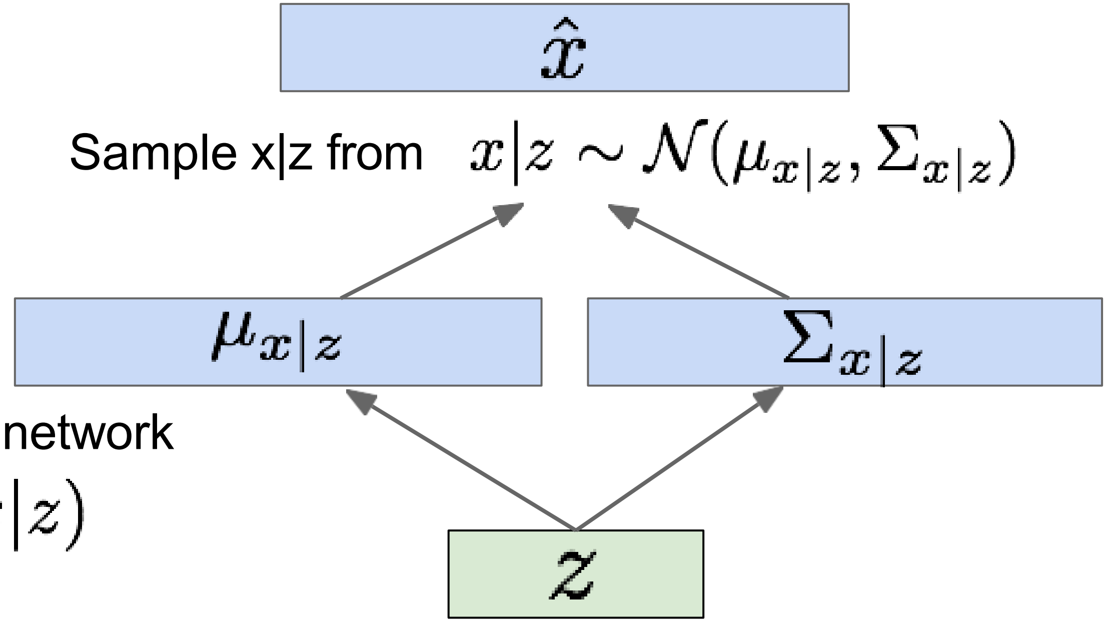




Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!



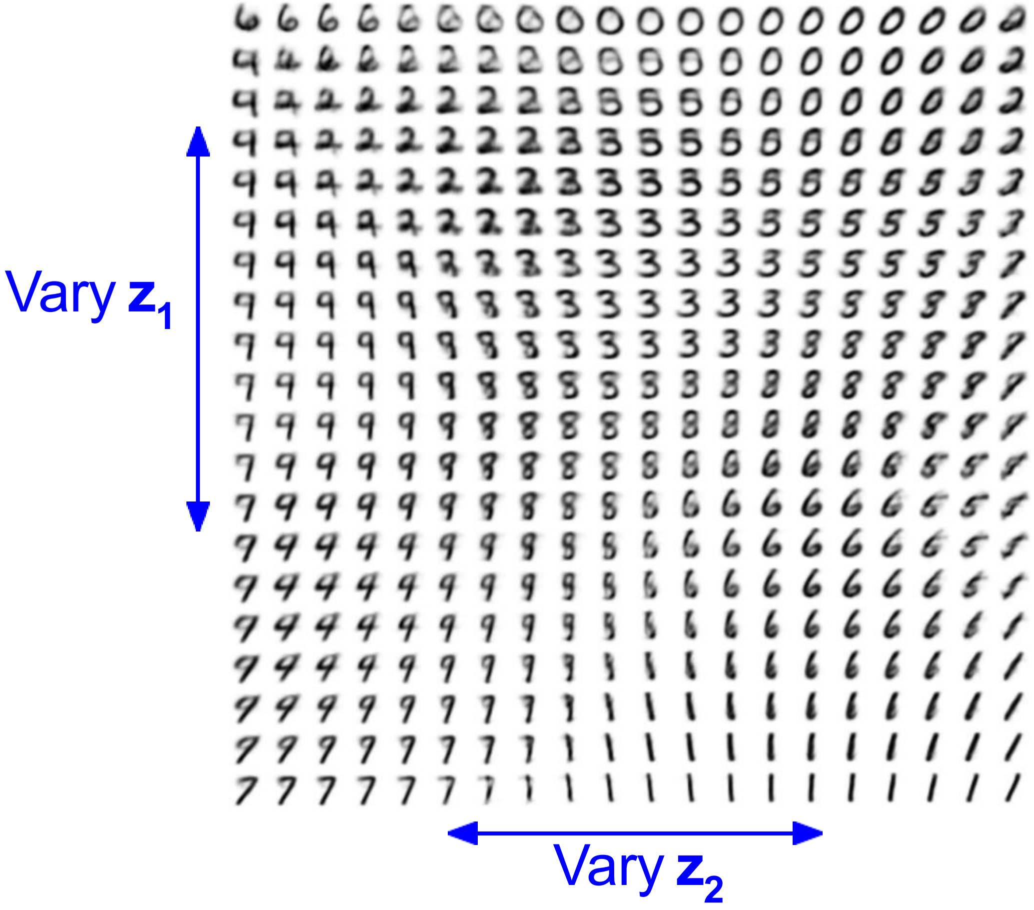
Decoder network

$p_\theta(x|z)$

Sample z from $z \sim \mathcal{N}(0, I)$

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

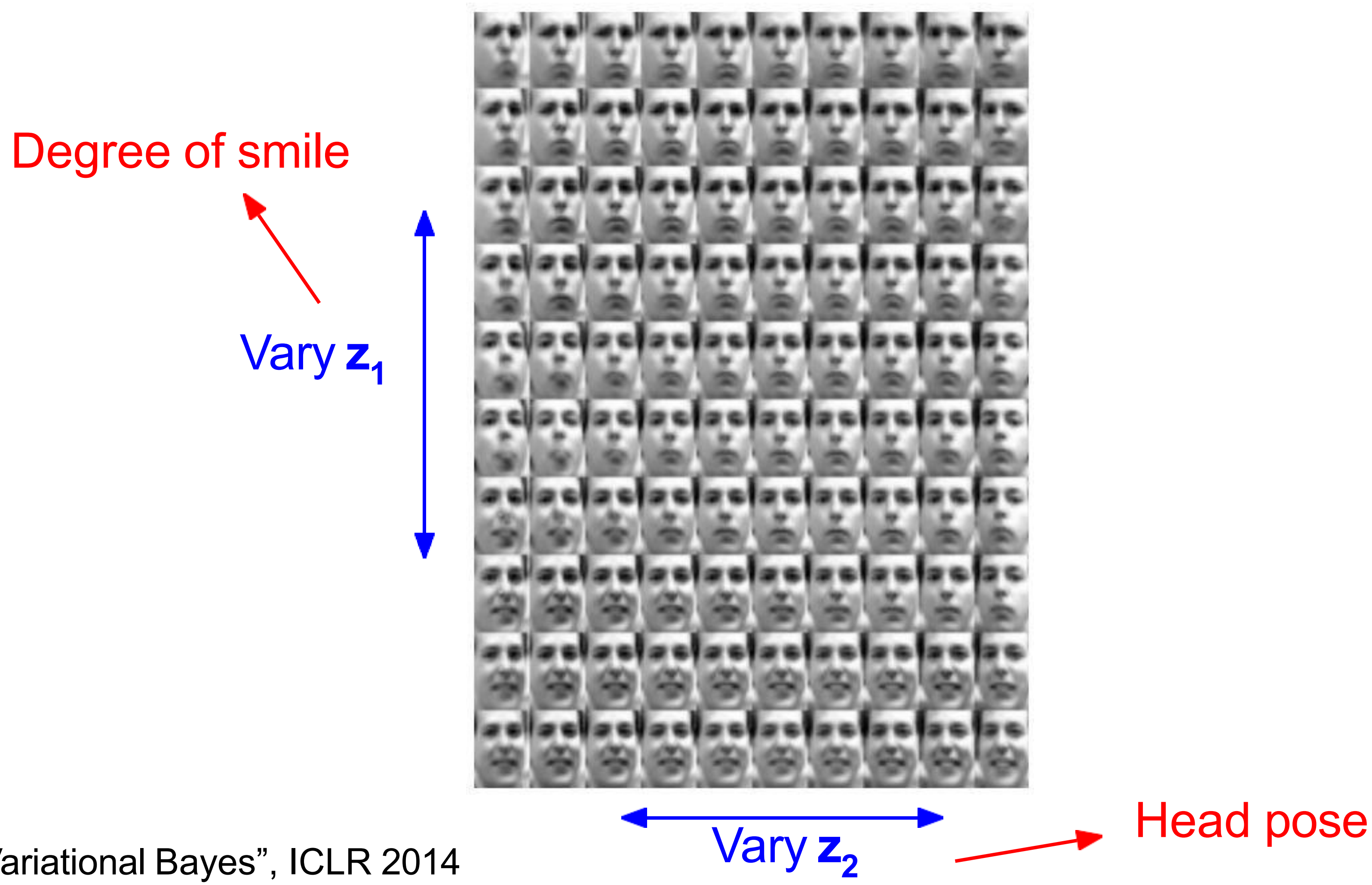
Data manifold for 2-d **z**



Variational Autoencoders: Generating Data!

Diagonal prior on **z**
=> independent
latent variables

Different
dimensions of **z**
encode
interpretable factors
of variation



Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Generating Data!

Diagonal prior on **z** => independent latent variables

Different dimensions of **z** encode interpretable factors of variation

Degree of smile

Vary $\mathbf{z}_1$

Vary $\mathbf{z}_2$

Head pose

Also good feature representation that can be computed using $q_\phi(z|x)$!

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

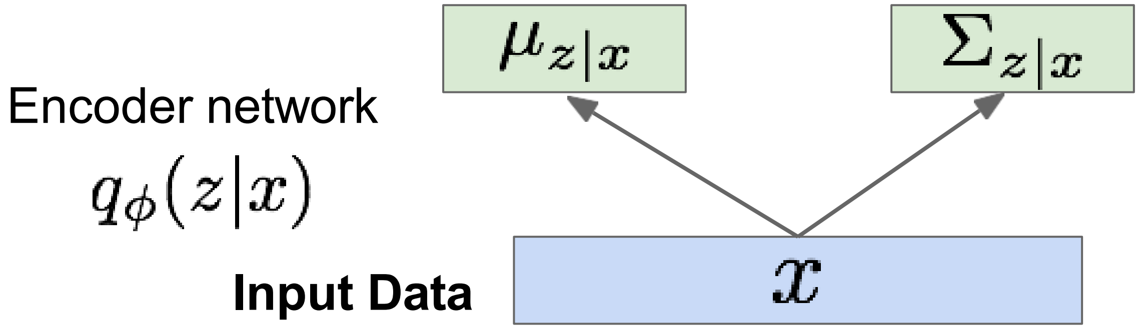
Variational Autoencoders: Generating Data!

32x32 CIFAR-10

Labeled Faces in the Wild

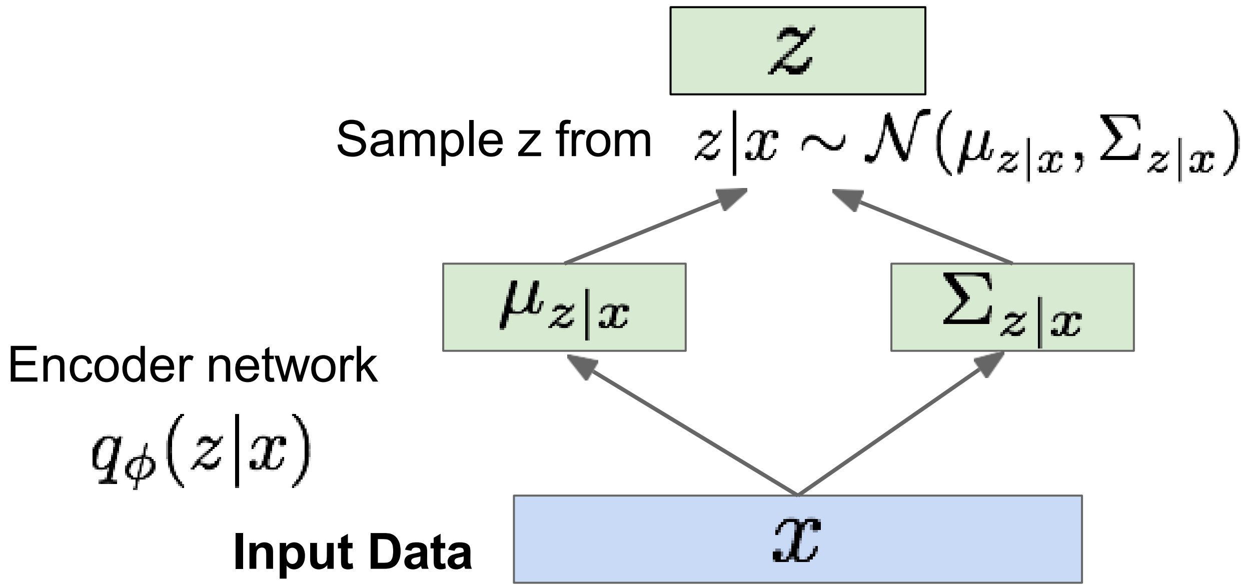
Editing images with VAEs

1. Run input data through encoder to get a distribution over latent codes



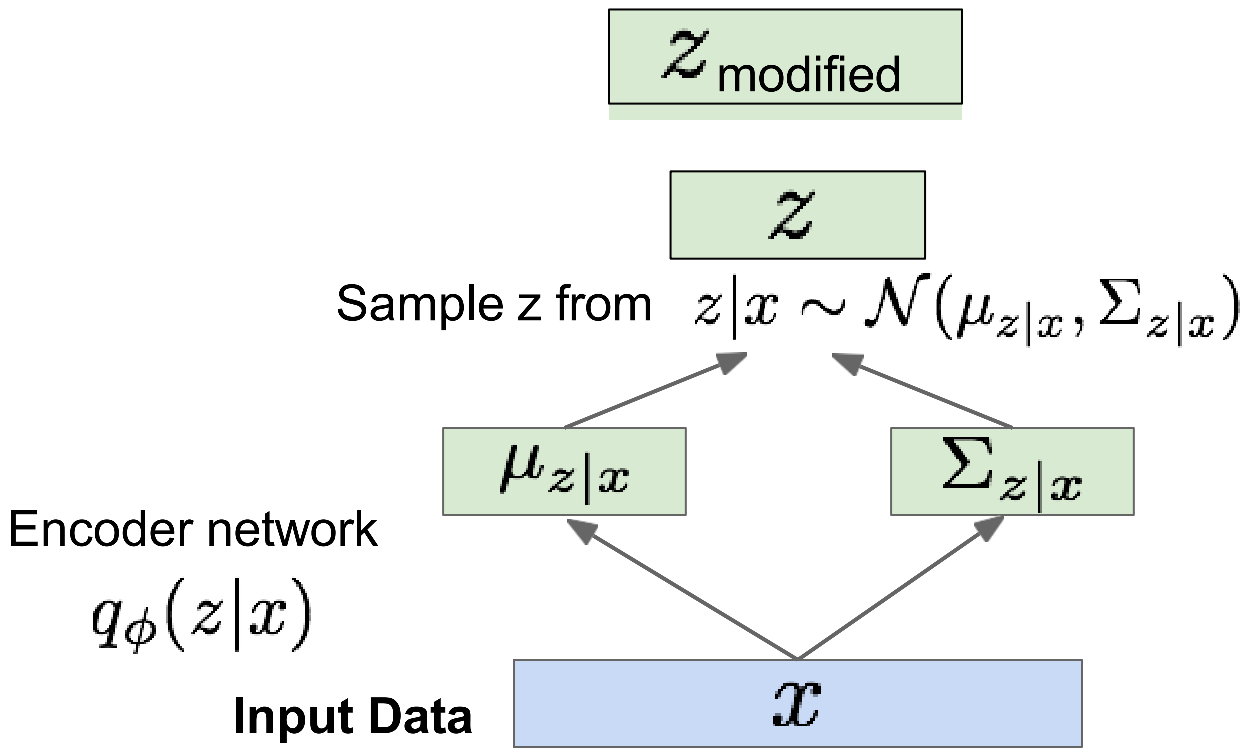
Editing images with VAEs

1. Run input data through encoder to get a distribution over latent codes
2. Sample code z from encoder output



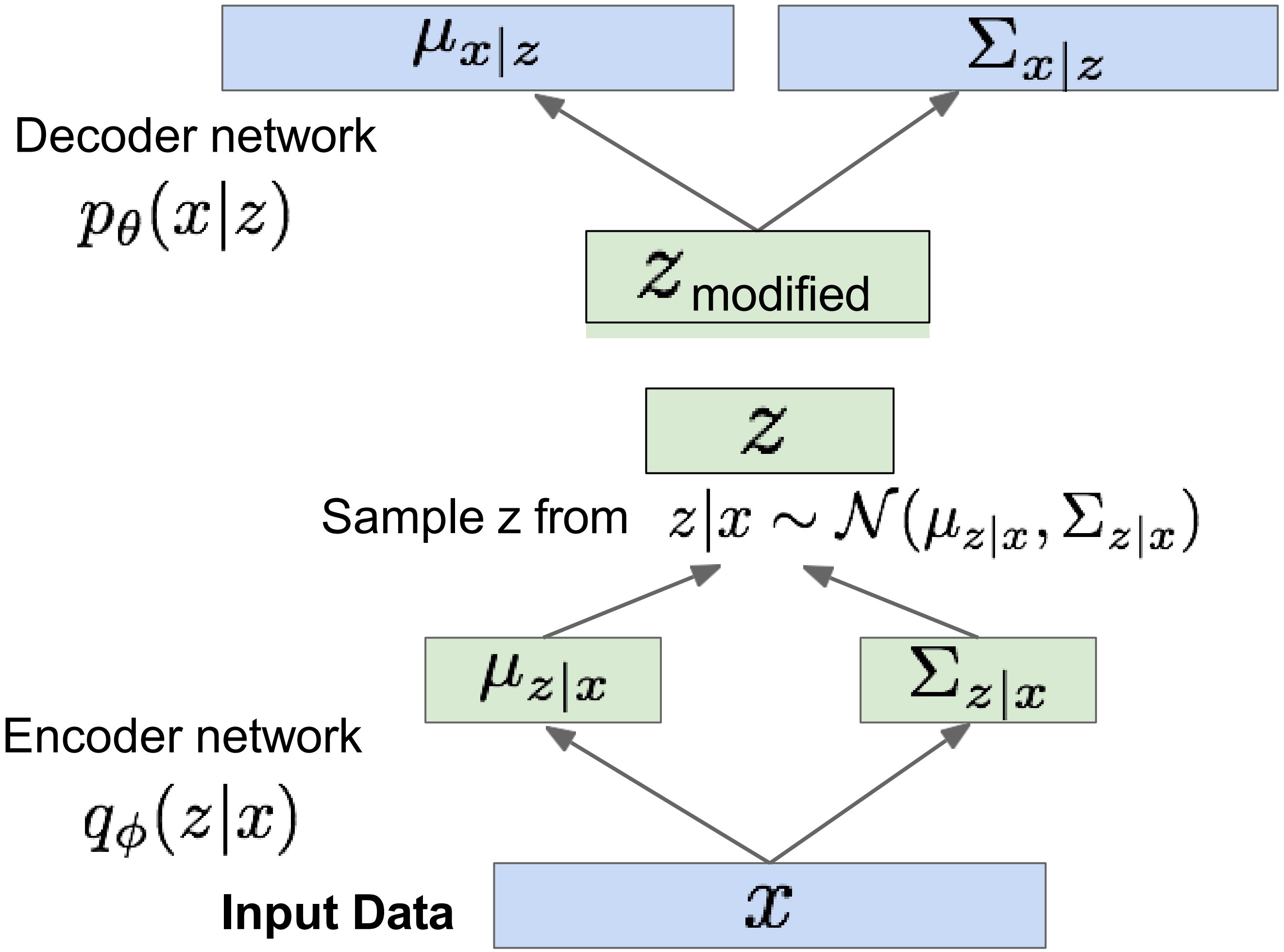
Editing images with VAEs

1. Run input data through encoder to get a distribution over latent codes
2. Sample code z from encoder output
3. Modify some dimensions of sampled code



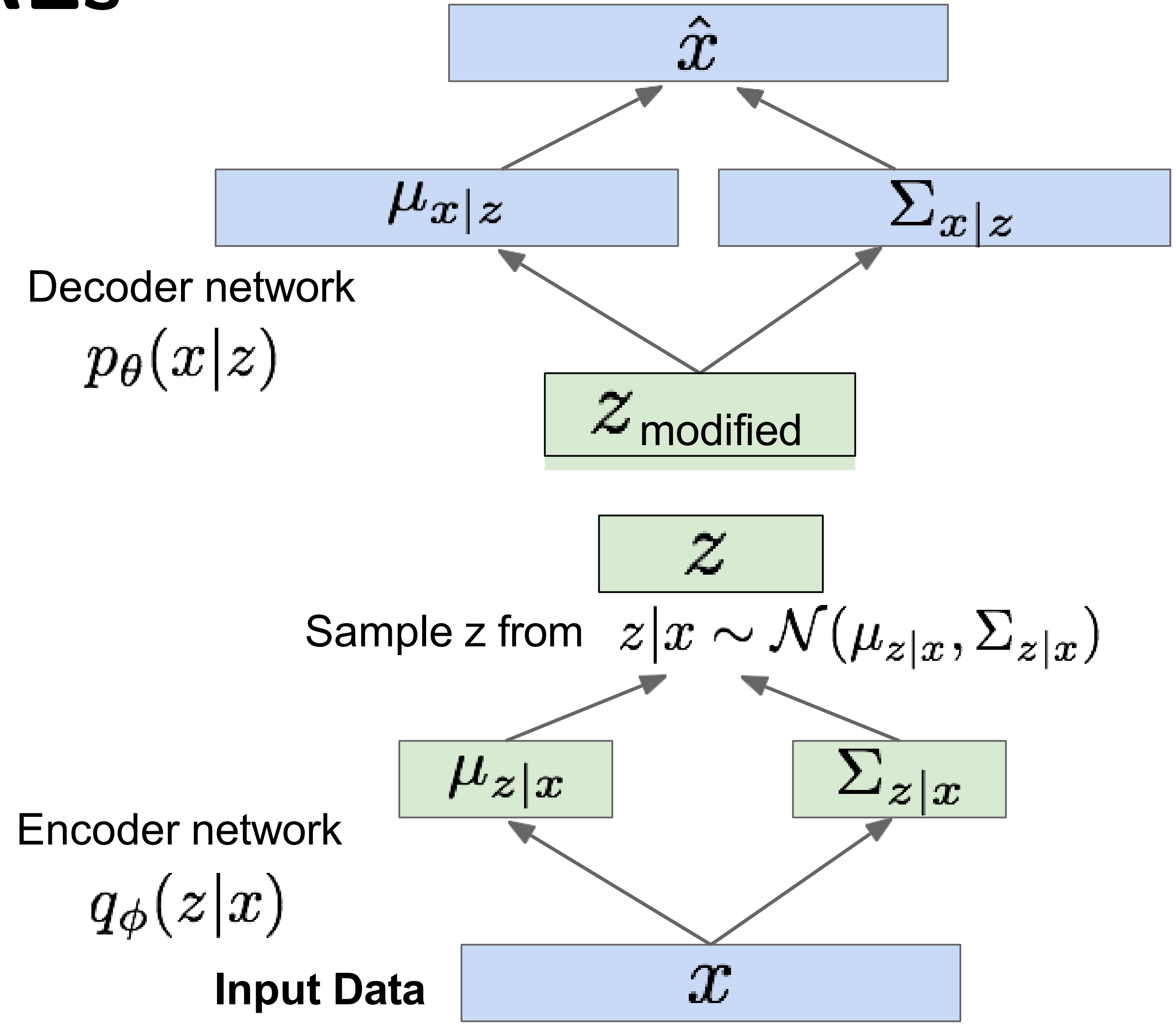
Editing images with VAEs

1. Run input data through encoder to get a distribution over latent codes
2. Sample code z from encoder output
3. Modify some dimensions of sampled code
4. Run modified z through decoder to get a distribution over data sample

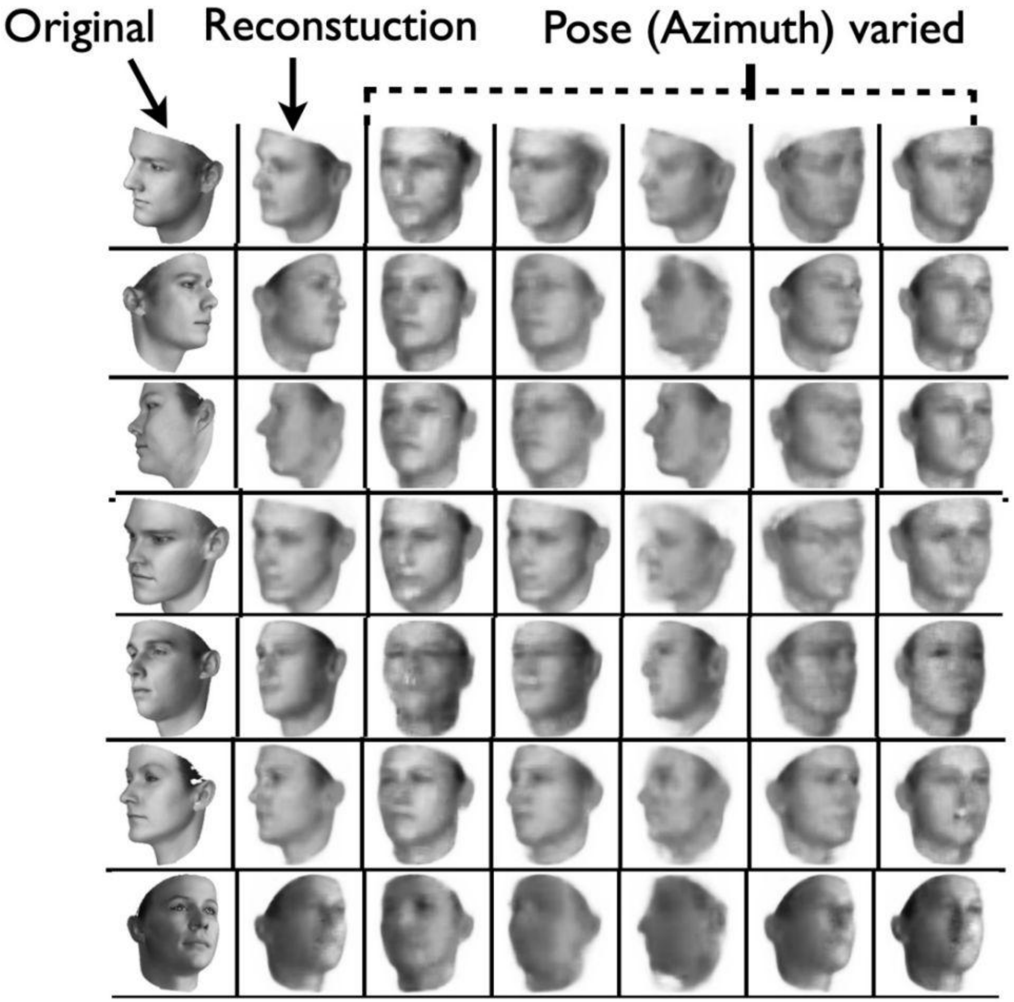


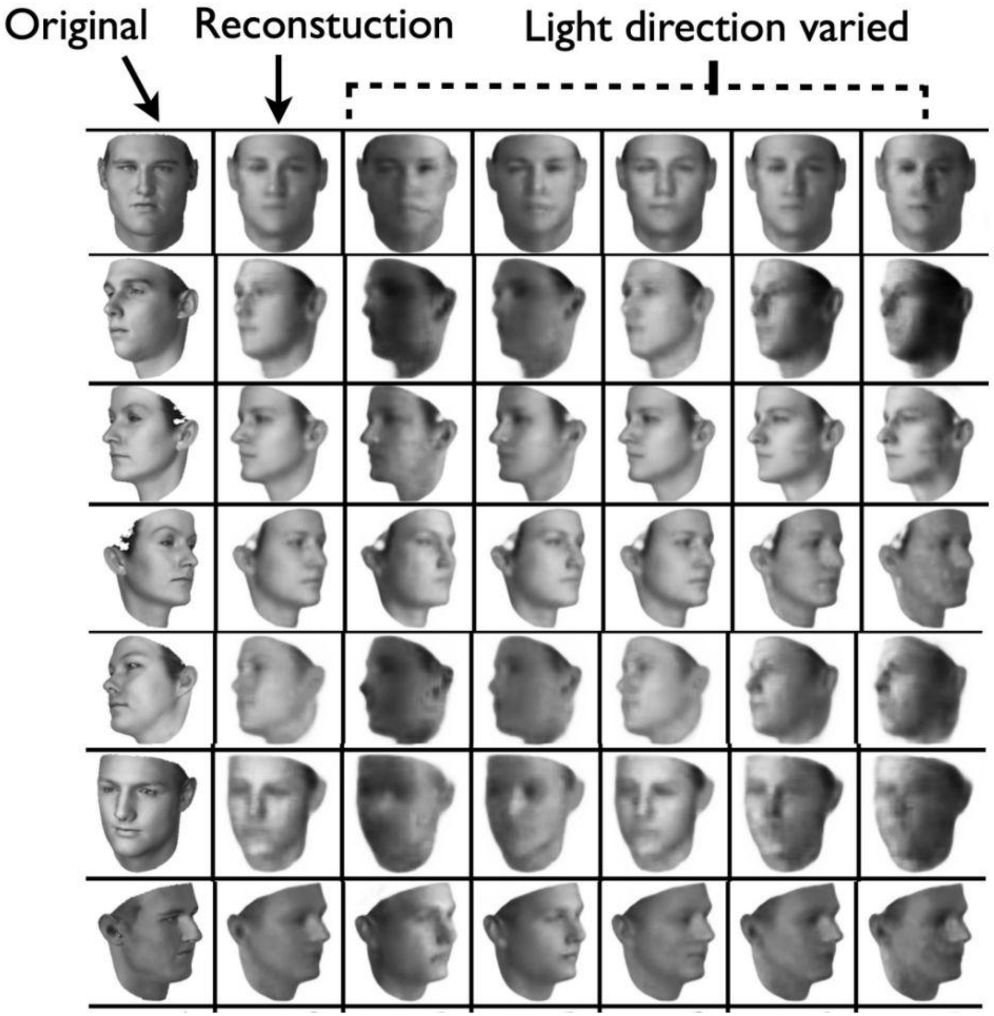
Editing images with VAEs

1. Run input data through encoder to get a distribution over latent codes
2. Sample code z from encoder output
3. Modify some dimensions of sampled code
4. Run modified z through decoder to get a distribution over data sample
5. Sample new data from (4)



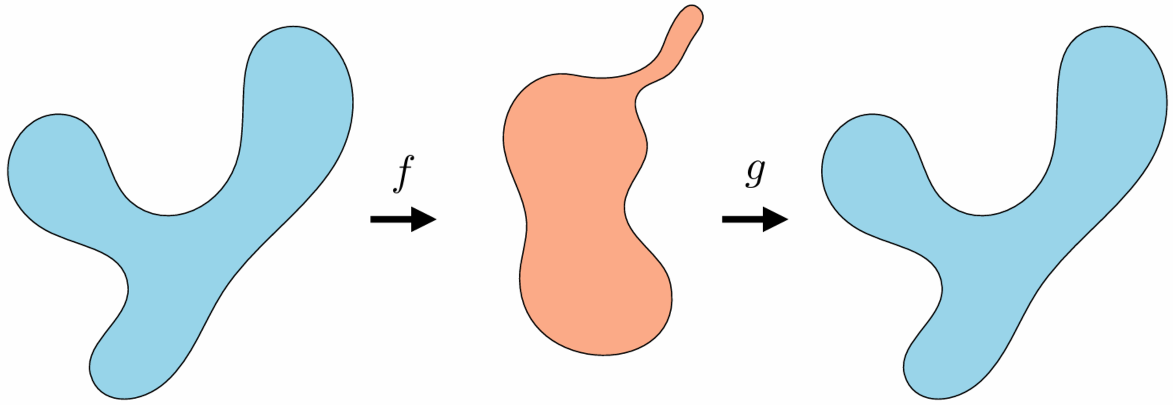
Editing images with VAEs





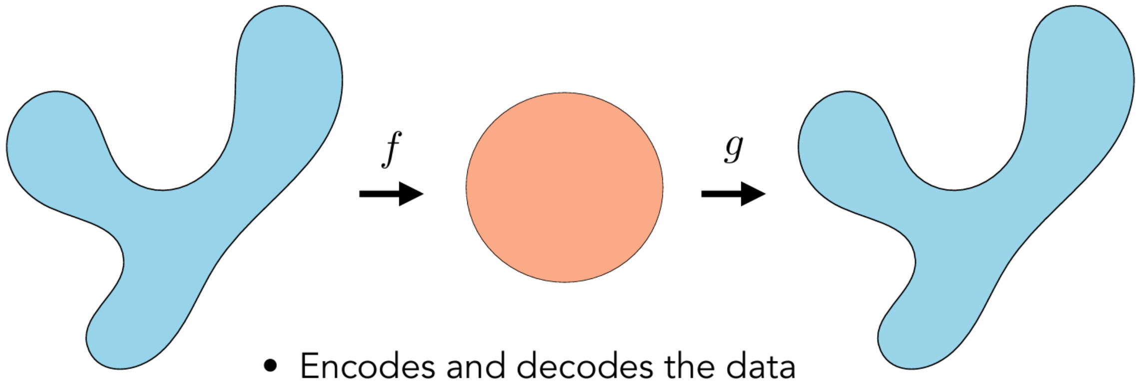
To Summarize: AE, VAE

Autoencoder



- Encodes and decodes the data
- Low-dimensional bottleneck

Variational Autoencoder



- Encodes and decodes the data
- Low-dimensional bottleneck
- Gaussian bottleneck (**can sample**; **disentangled**)

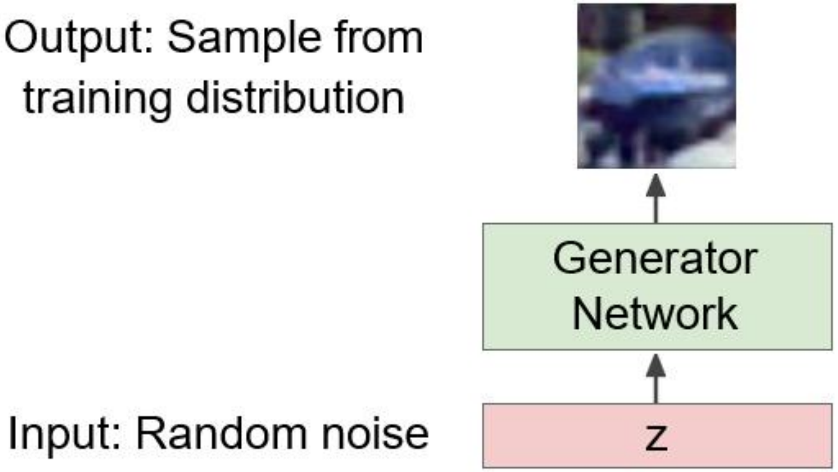
Generative Adversarial Networks

Ian J. Goodfellow, Jean Pouget-Abadie[*], Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair[†], Aaron Courville, Yoshua Bengio[‡]

Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

NIPS 2014

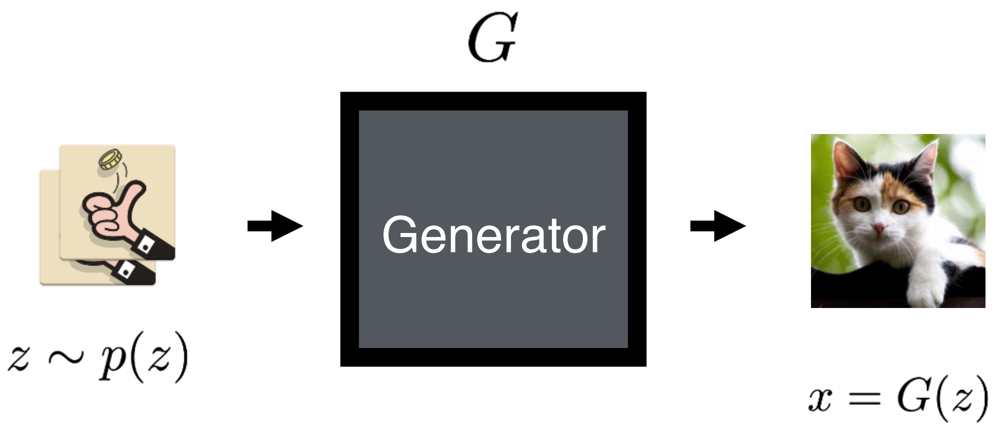
- **Problem:** We want to sample from a high-dimensional training distribution $p(x)$
  - **But there is no direct way to do this …**
  - **We don't know which z maps to which image (so we can't use autoencoders)**
- We know how to sample from a random distribution (e.g. Gaussian)
  - Can we map a random distribution ***directly*** to $p(x)$?



Generative Adversarial Networks

Goal: Map **all** $\boldsymbol{z}$ to ***some realistic-looking*** $\boldsymbol{x}$

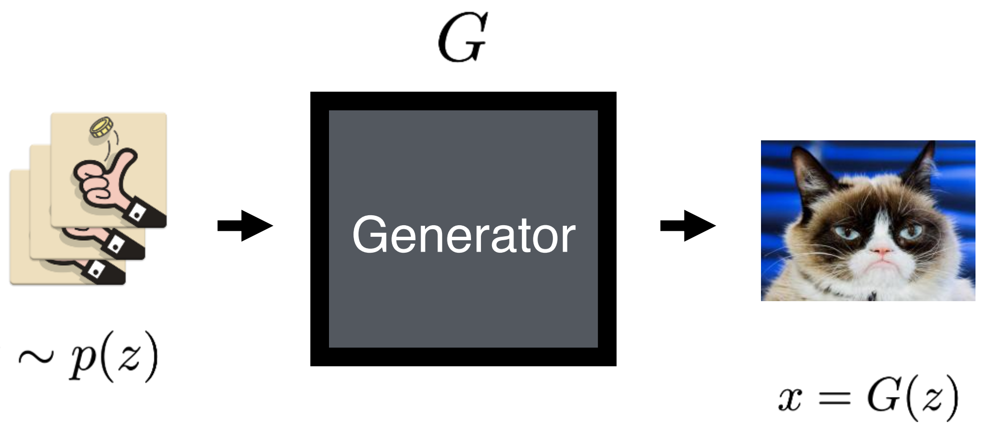
Image synthesis from “noise”



Sampler

$$G : \mathcal{Z} \rightarrow \mathcal{X}$$
$$z \sim p(z)$$
$$x = G(z)$$

Image synthesis from “noise”

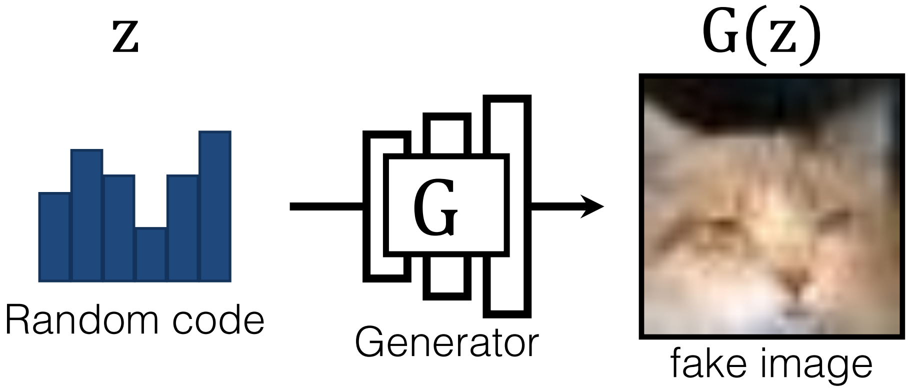


Sampler

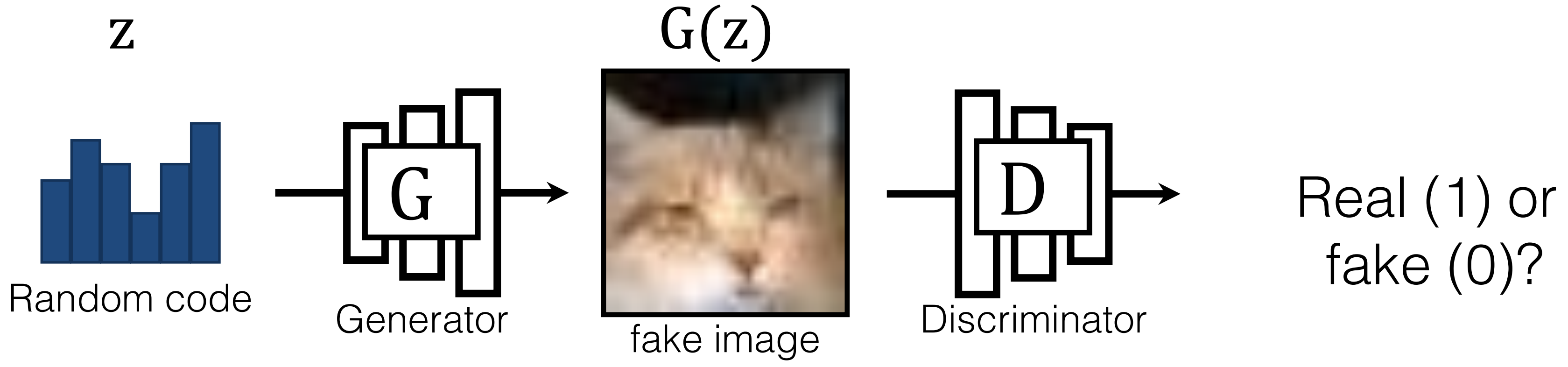
$$G : \mathcal{Z} \to \mathcal{X}$$

$$z \sim p(z)$$

$$x = G(z)$$



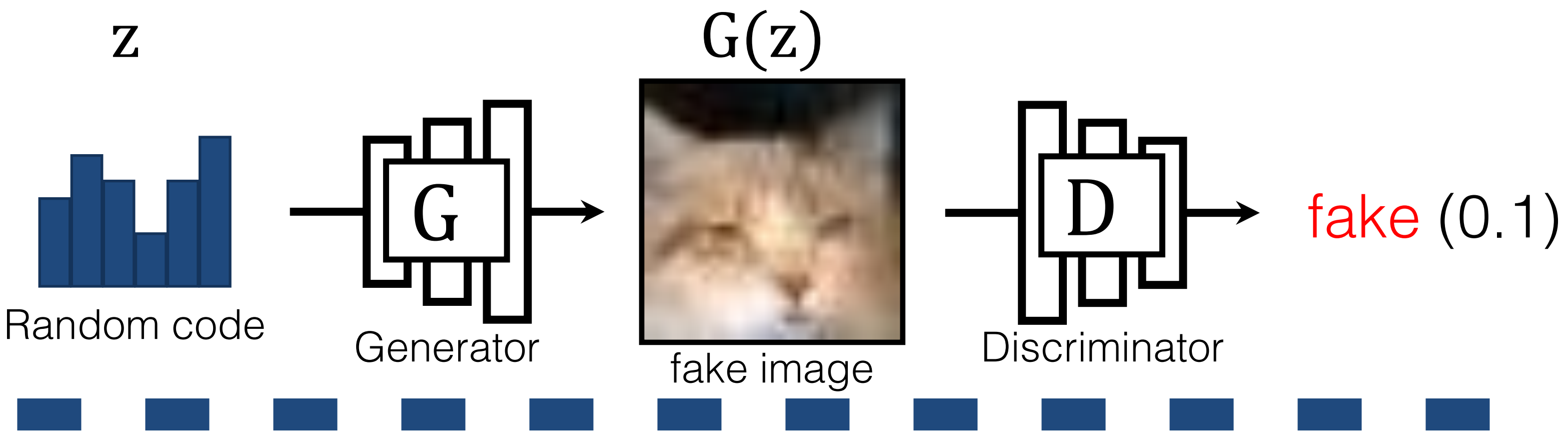
[Goodfellow et al. 2014]



A two-player game:

- G tries to generate fake images that can fool D.
- D tries to detect fake images.

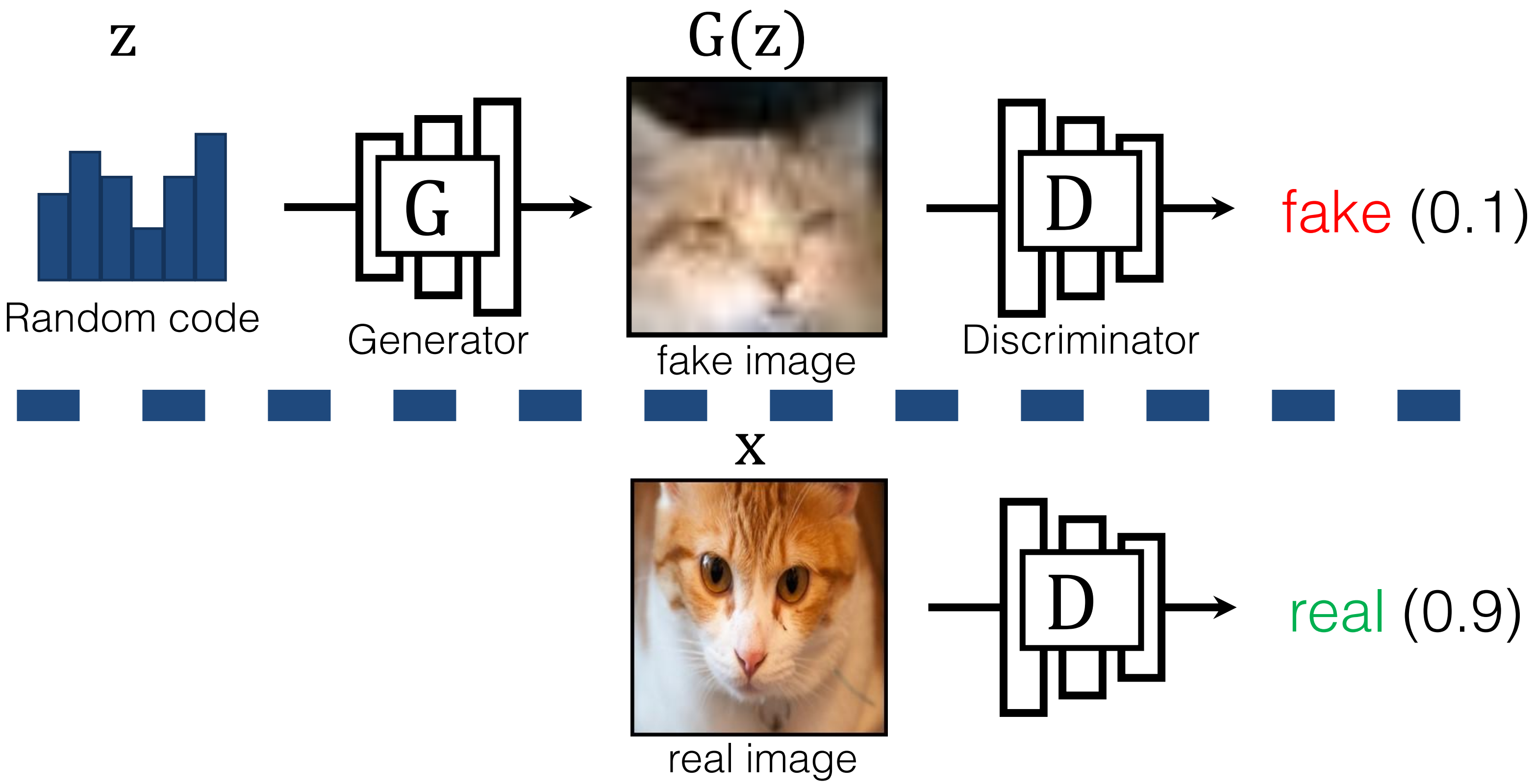
[Goodfellow et al. 2014]



Learning objective (GANs)

$$\min_{G} \max_{\boxed{D}} \boxed{\mathbb{E}_z[\log(1 - D(G(z)))]}$$

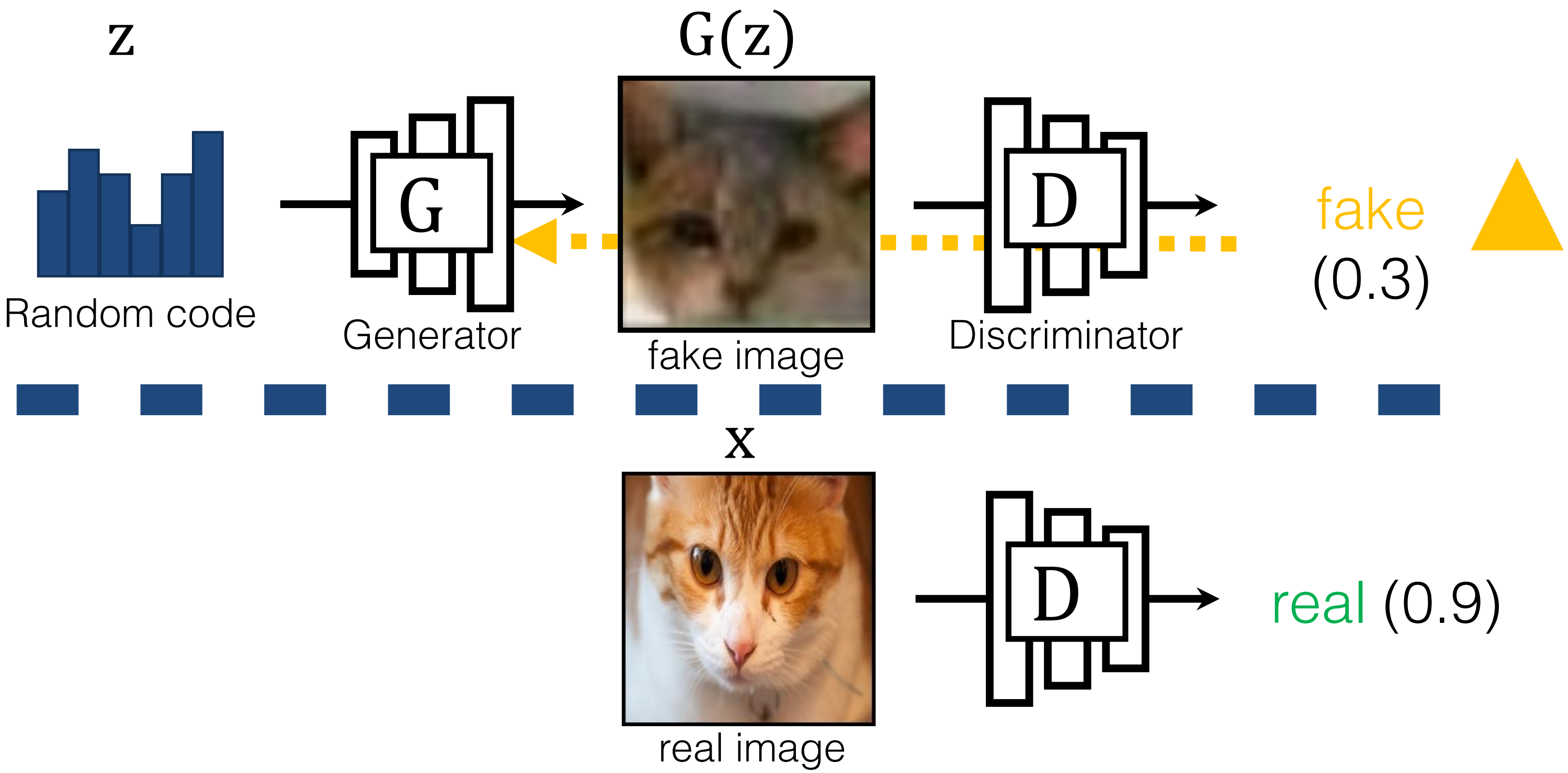
[Goodfellow et al. 2014]



Learning objective (GANs)

$$\min_{G} \max_{\boxed{D}} \boxed{\mathbb{E}_z[\log(1 - D(G(z)))]} + \boxed{\mathbb{E}_x[\log D(x)]}$$

[Goodfellow et al. 2014]



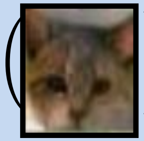
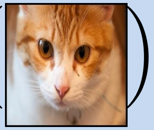
Learning objective (GANs)

$$\min_{\boxed{G}} \max_{D} \boxed{\mathbb{E}_z[\log(1 - D(G(z)))]} + \mathbb{E}_x[\log D(x)]$$

[Goodfellow et al. 2014]

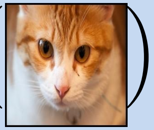
GAN Training Breakdown

- From the discriminator D's perspective:
  - binary classification: real vs. fake.
  - Nothing special: similar to 1 vs. 7 or cat vs. dog

$\max_D \mathbb{E}[\log(1-D($  $))]+\mathbb{E}[\log D($  $)]$

GAN Training Breakdown

- From the discriminator D's perspective:
  - binary classification: real vs. fake.
  - Nothing special: similar to 1 vs. 7 or cat vs. dog

$\max_D \mathbb{E}[\log(1-D($  $))] + \mathbb{E}[\log D($  $)]$

- From the generator G's perspective:
  - Optimizing a loss that depends on a classifier D

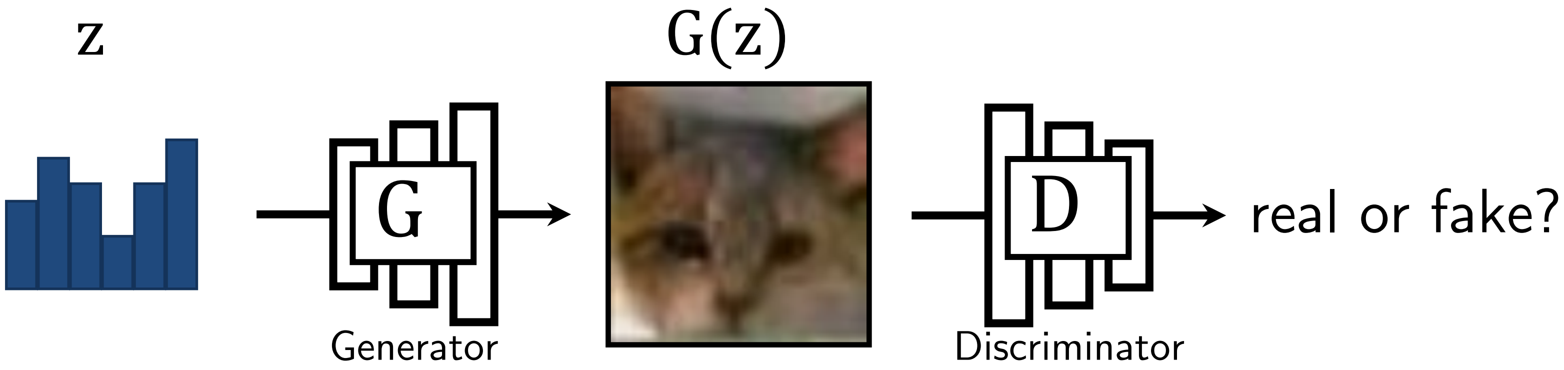
$$\min_G \mathbb{E}_z[\mathcal{L}_D(G(z))]$$

GAN loss for G

$$\min_G \mathbb{E}_{(x,y)}||F(G(x)) - F(y)||$$

Perceptual Loss for G

GAN Training Breakdown



G tries to synthesize fake images that fool D

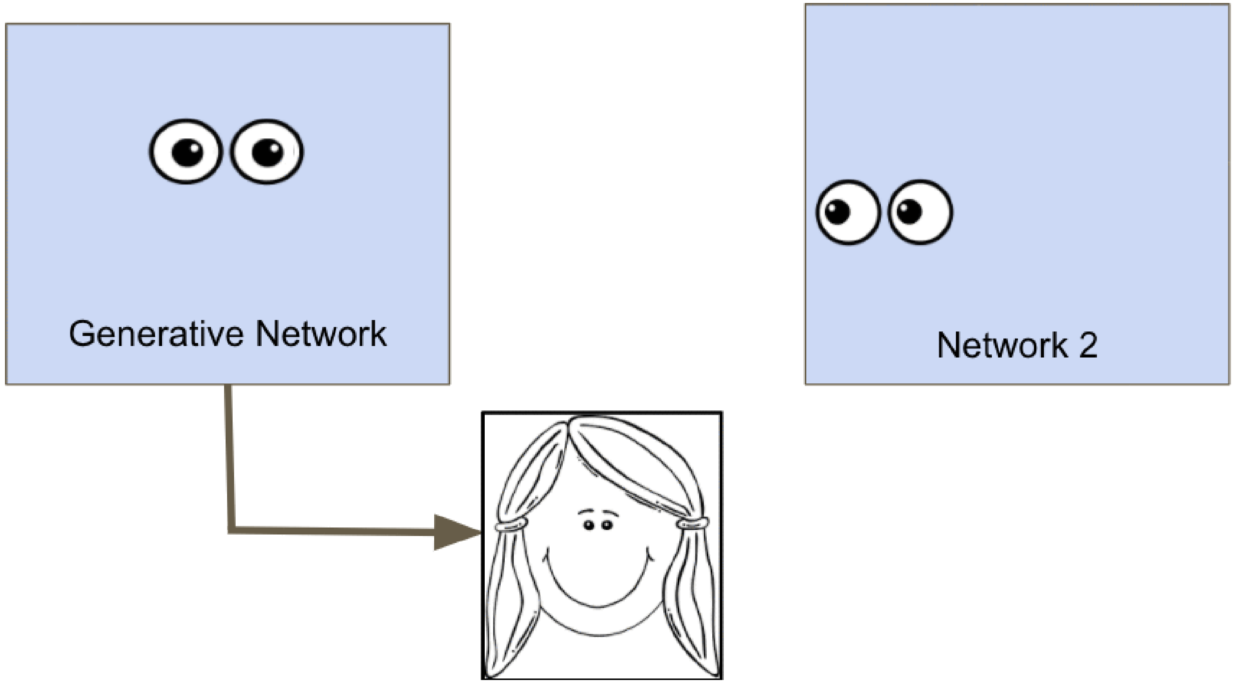
D tries to identify the fakes

- Training: iterate between training D and G with backprop.
- Global optimum when G reproduces data distribution.

[Goodfellow et al., 2014]

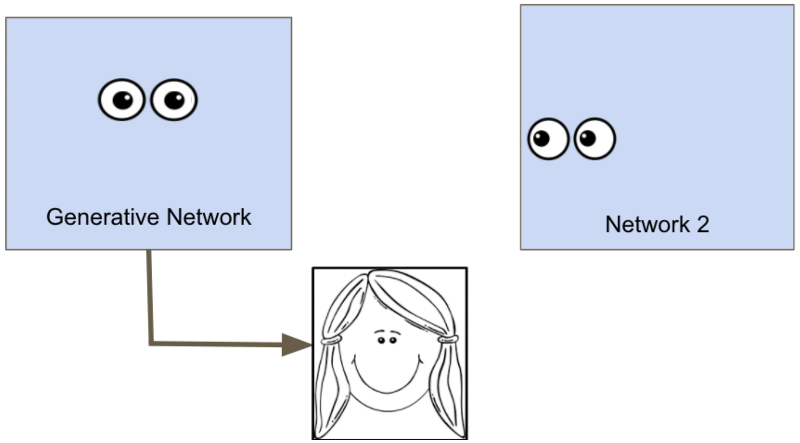
Training GANs: Two-player game

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images



Training GANs: Two-player game

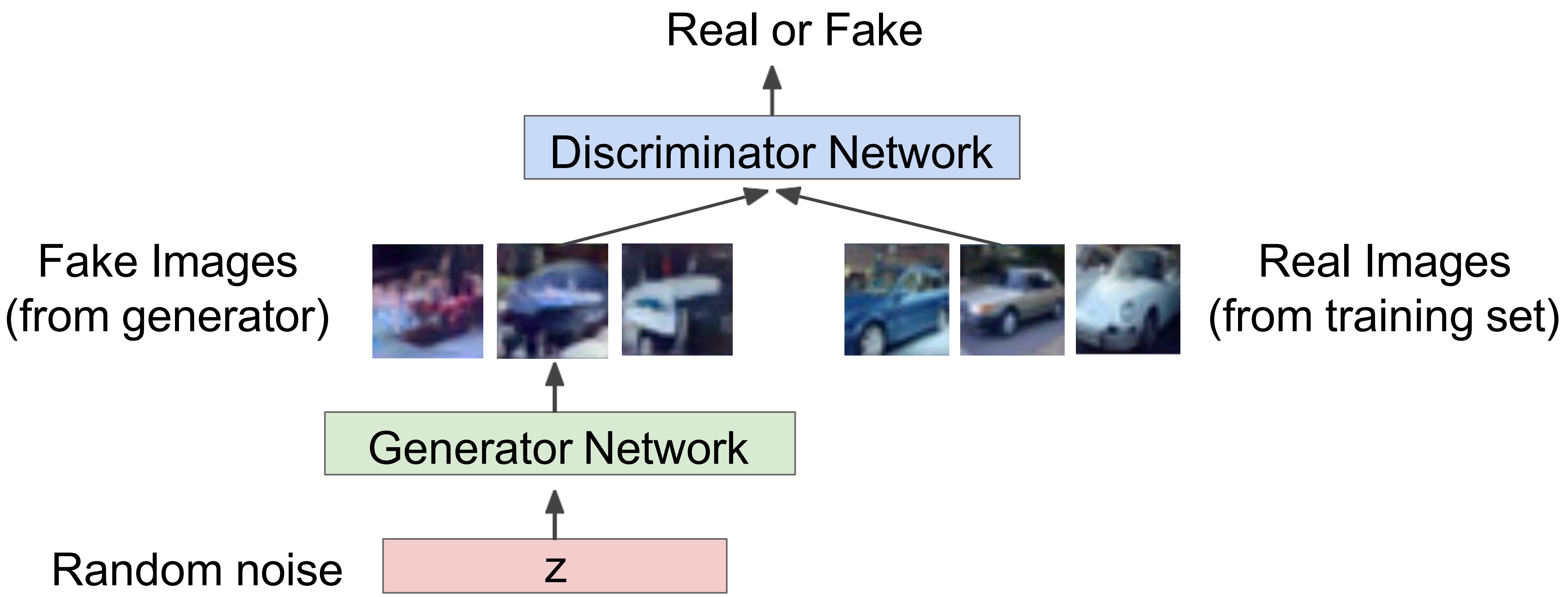
Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images



Connection to Game Theory: Zero-Sum "Minimax" Game
- Each player trying to minimize the opponent's profits
- Each player trying to maximize their own profits

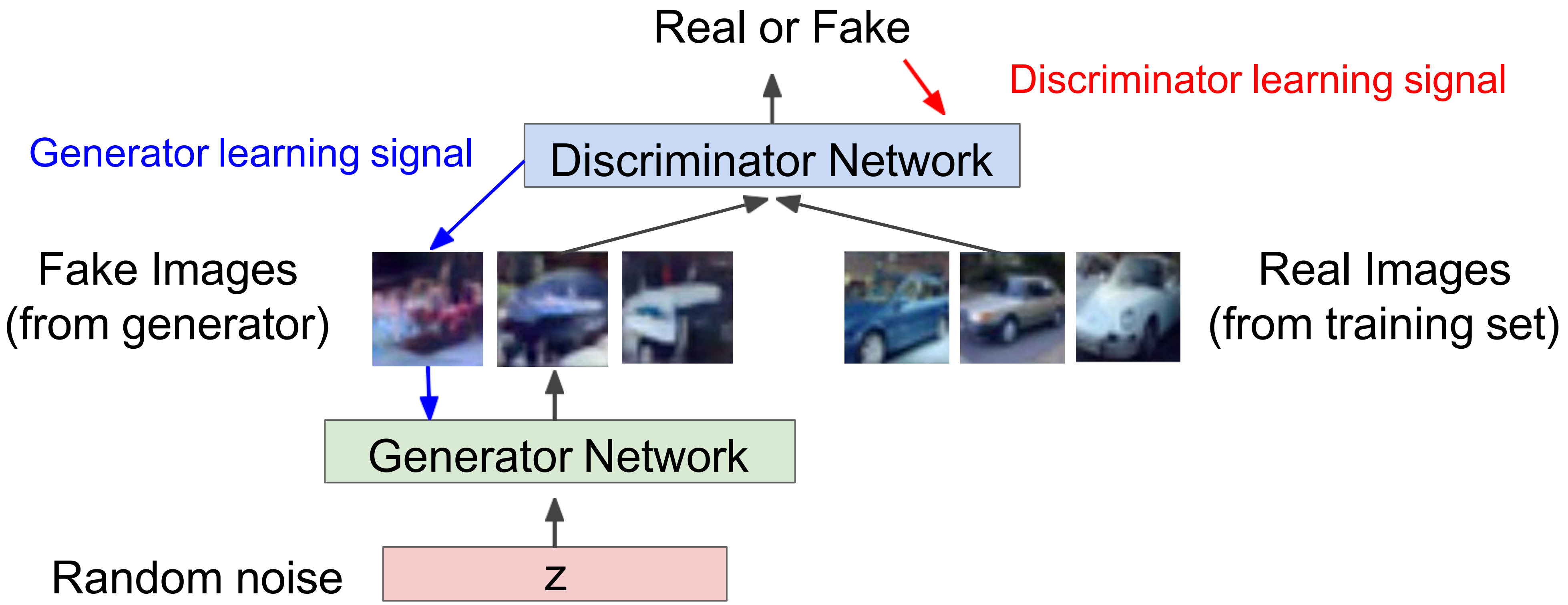
Training GANs: Two-player game

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images



Training GANs: Two-player game

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images



Training GANs: Two-player game

Discriminator network:	try to distinguish between real and fake images
Generator network:	try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

$$\underbrace{\min_{\theta_g}}_{\text{Generator objective}} \underbrace{\max_{\theta_d}}_{\text{Discriminator objective}} \left[\mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))\right]$$

Training GANs: Two-player game

Discriminator network:	try to distinguish between real and fake images
Generator network:	try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- Discriminator (θ_d) wants to **maximize objective** s.t.
  D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)

- Generator (θ_g) wants to **minimize objective** s.t.
  D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Training GANs: Two-player game

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

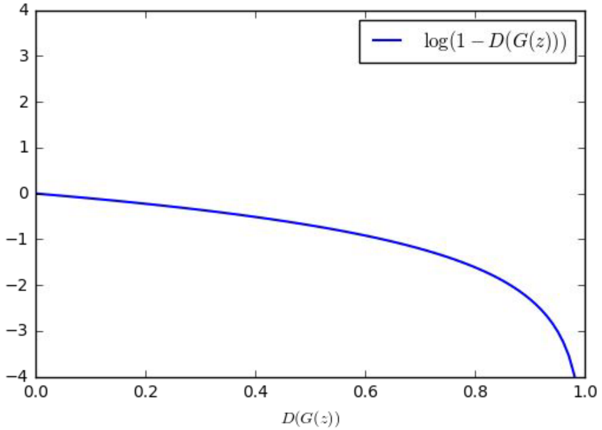
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).



Training GANs: Two-player game

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

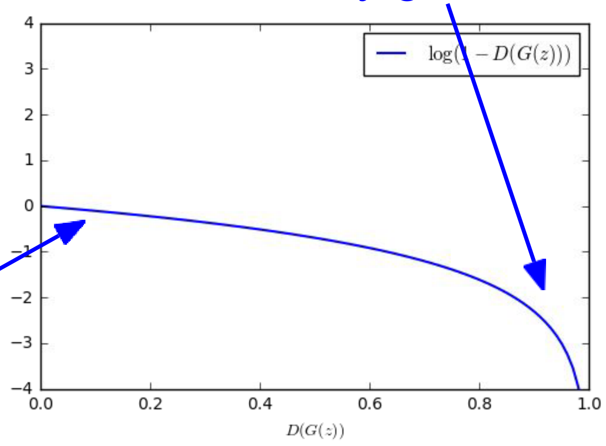
$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).

But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good



Training GANs: Two-player game

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

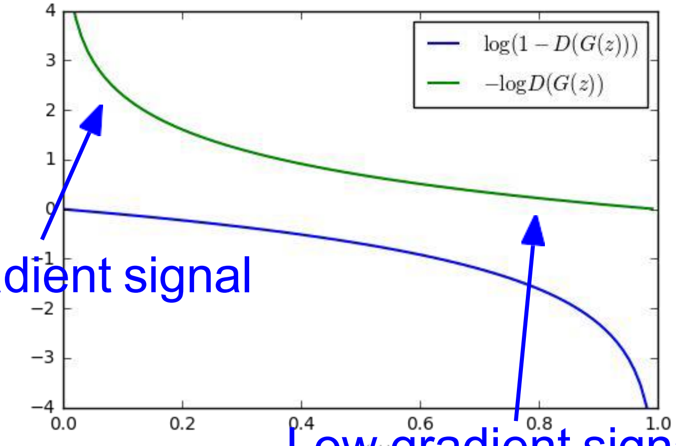
Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$



High gradient signal

Low gradient signal

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.

Training GANs: Two-player game

Putting it together: GAN training algorithm

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:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

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 ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Training GANs: Two-player game

Putting it together: GAN training algorithm

Some find k=1 more stable, others use k > 1, no best rule.

Followup work (e.g. Wasserstein GAN, BEGAN) alleviates this problem, better stability!

for number of training iterations **do**

for $\boxed{k \text{ steps}}$ **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:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

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 ascending its stochastic gradient (improved objective):

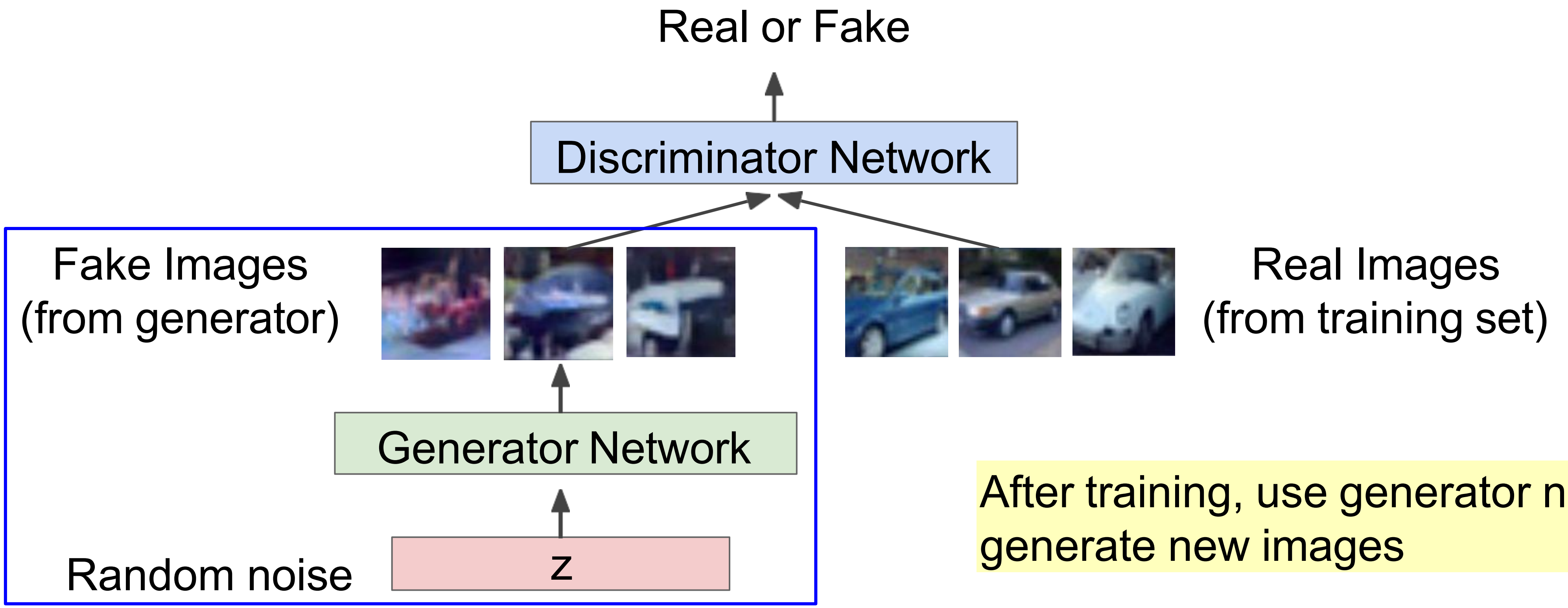
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Arjovsky et al. "Wasserstein gan." arXiv preprint arXiv:1701.07875 (2017)
Berthelot, et al. "Began: Boundary equilibrium generative adversarial networks." arXiv preprint arXiv:1703.10717 (2017)

Training GANs: Two-player game

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images



After training, use generator network to generate new images

Generative Adversarial Nets: Convolutional Architectures

Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

Samples from the model look much better!

Radford et al, ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

Interpolating between random points in latent space

Radford et al, ICLR 2016

Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

Smiling woman   Neutral woman   Neutral man

Samples from the model

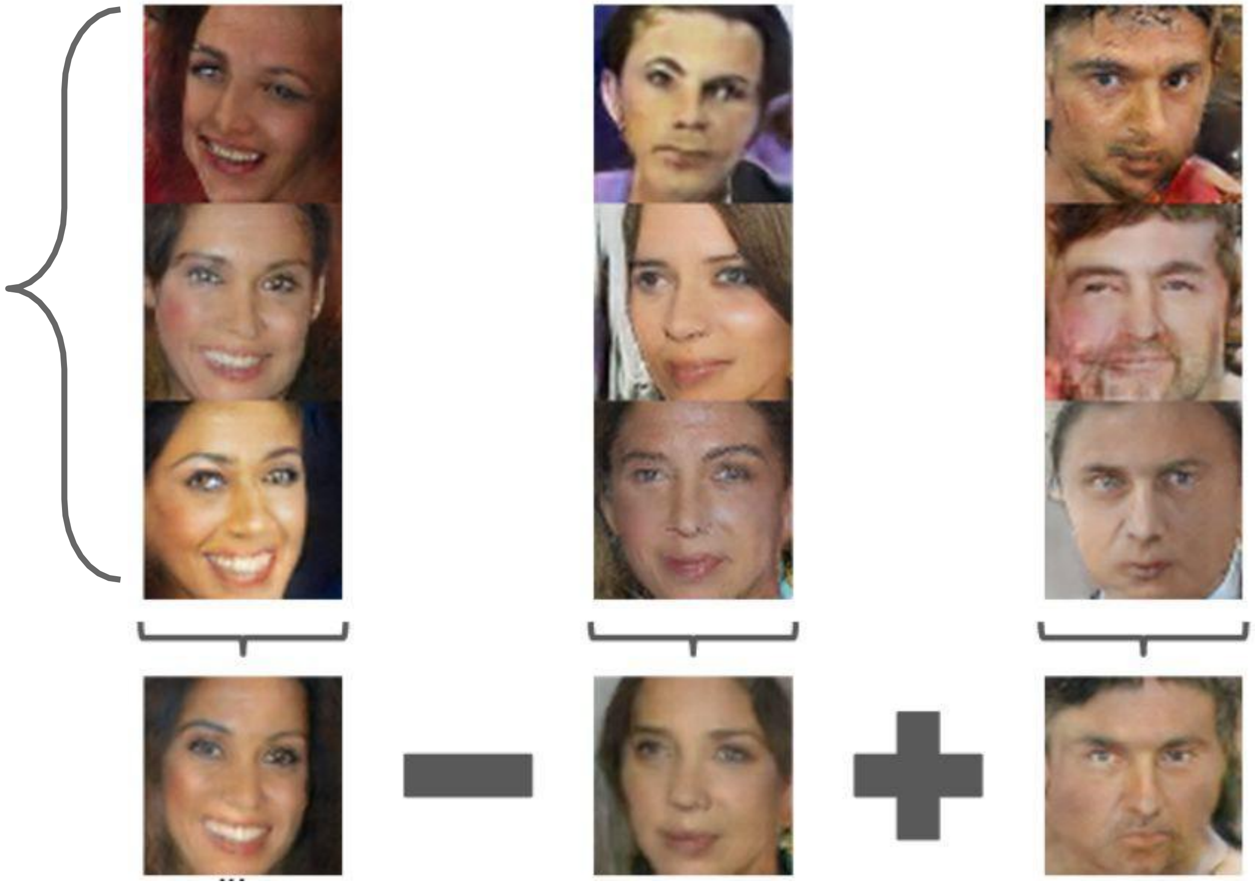
Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

Smiling woman   Neutral woman   Neutral man

Samples from the model

Average Z vectors, do arithmetic



Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

Smiling woman Neutral woman Neutral man

Samples from the model

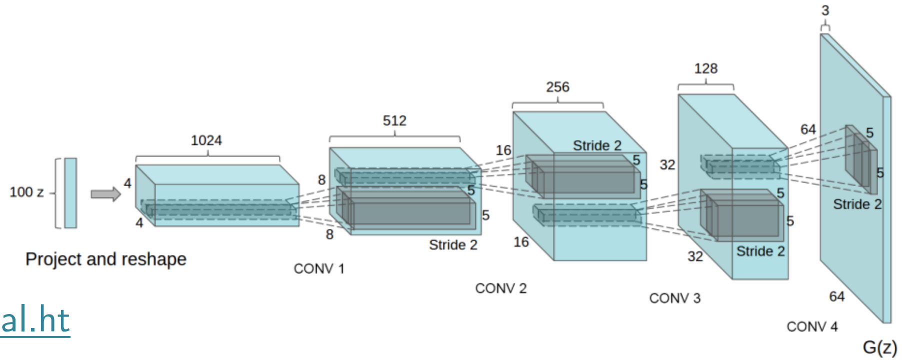
Smiling Man

Average Z vectors, do arithmetic

Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

GAN in PyTorch



https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

```
class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is ``(nc) x 64 x 64``
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf) x 32 x 32``
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf*2) x 16 x 16``
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf*4) x 8 x 8``
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. ``(ndf*8) x 4 x 4``
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

```
class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            # state size. ``(ngf*8) x 4 x 4``
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            # state size. ``(ngf*4) x 8 x 8``
            nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            # state size. ``(ngf*2) x 16 x 16``
            nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            # state size. ``(ngf) x 32 x 32``
            nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. ``(nc) x 64 x 64``
        )

    def forward(self, input):
        return self.main(input)
```

https://github.com/soumith/ganhacks for tips and tricks for training GANs

Since then: Explosion of GANs

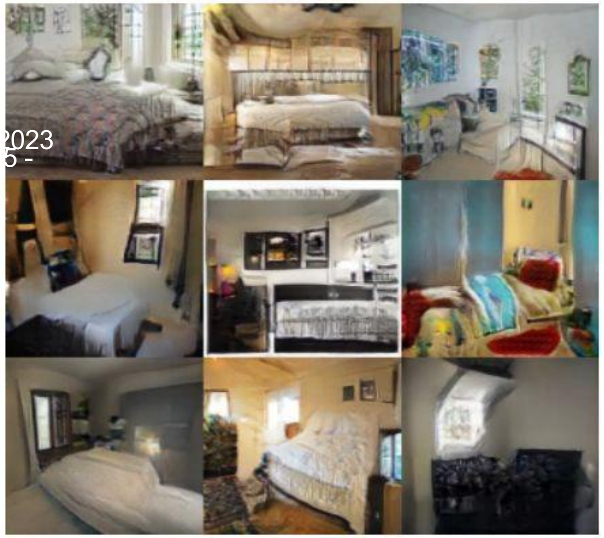
"The GAN Zoo"

- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorial GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

https://github.com/hindupuravinash/the-gan-zoo

2017: Explosion of GANs

Better training and generation



LSGAN, Zhu 2017.

Wasserstein GAN, Arjovsky 2017. Improved Wasserstein GAN, Gulrajani 2017.

Progressive GAN, Karras 2018.

Some challenges with GANs ...

Challenges with GANs

- **Vanishing gradients:**
  - the discriminator becomes too good and the generator gradient vanishes.
- **Non-Convergence:**
  - the generator and discriminator oscillate without reaching an equilibrium.
- **Mode Collapse:**
  - the generator distribution collapses to a small set of examples.
- **Mode Dropping:**
  - the generator distribution doesnt fully cover the data distribution.

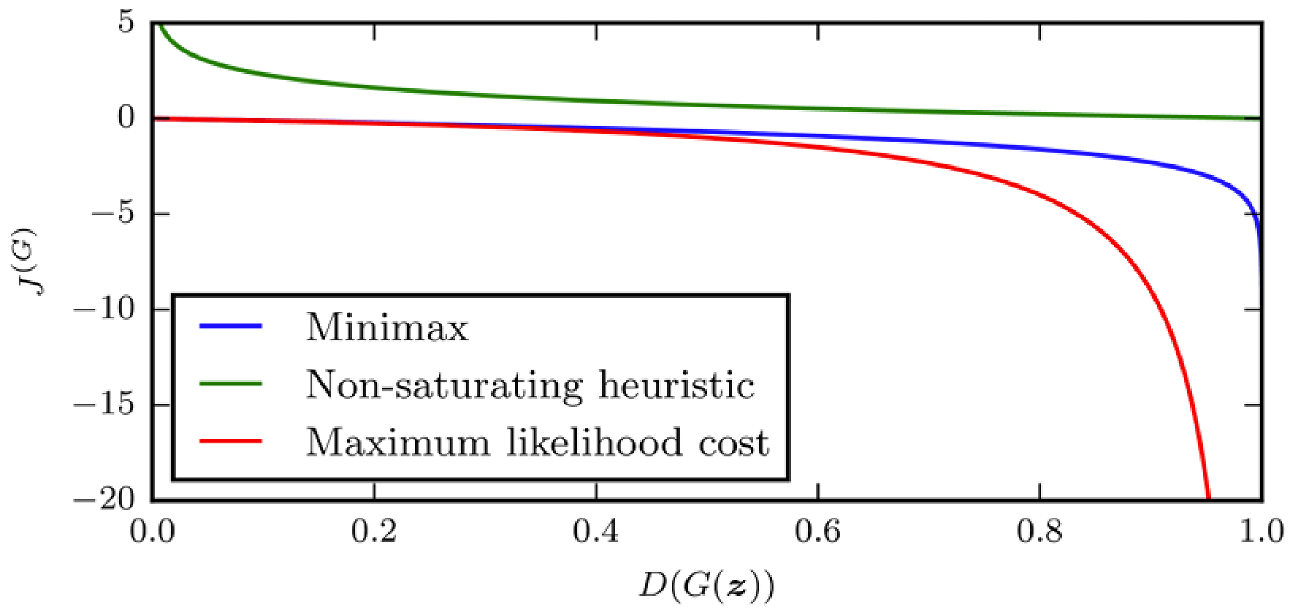
Challenges with GANs: Vanishing Gradients

- The **minimax** objective saturates when D_{θ_d} is close to perfect:

$$V(\theta_d, \theta_g) = \mathbb{E}_{p_{\text{data}}} [\log D_{\theta_d}(\mathbf{x})] + \mathbb{E}_{p_{\mathbf{z}}(\mathbf{z})} \left[\log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)\right].$$

- A **non-saturating heuristic** objective for the generator is

$$J(G_{\theta_g}) = -\mathbb{E}_{p_{\mathbf{z}}(\mathbf{z})} \left[\log \left(D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)\right].$$



https://arxiv.org/abs/1701.00160

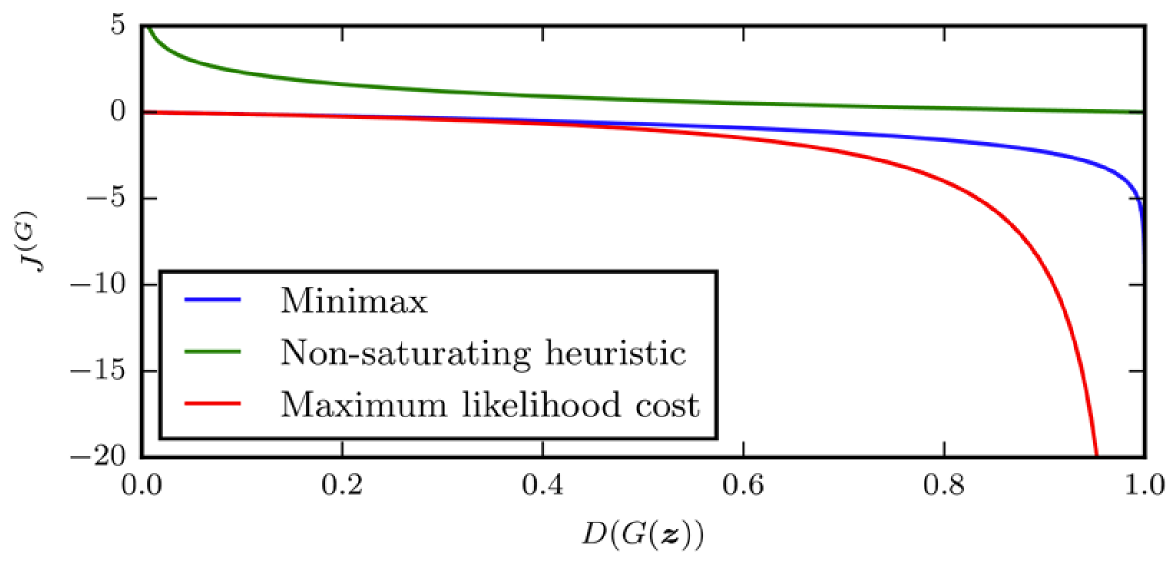
Challenges with GANs: Vanishing Gradients

- The **minimax** objective saturates when D_{θ_d} is close to perfect:

$$V(\theta_d, \theta_g) = \mathbb{E}_{p_{\text{data}}} [\log D_{\theta_d}(\mathbf{x})] + \mathbb{E}_{p_{\mathbf{z}}(\mathbf{z})} \left[\log\left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)\right].$$

- A **non-saturating heuristic** objective for the generator is

$$J(G_{\theta_g}) = -\mathbb{E}_{p_{\mathbf{z}}(\mathbf{z})} \left[\log\left(D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)\right].$$



https://arxiv.org/abs/1701.00160

Potential Solutions:

1. **Explore other training objectives?**
2. **Discriminator Capacity:**
   - make it small ?
   - train it less ?
   - slow learning rate?
3. **Learning Schedule:**
   - try to balance training G and D

Problems: Nonconvergence

- **Deep Learning models (in general) involve a single player**
  - The player tries to maximize its reward (minimize its loss).
  - Use SGD (with Backpropagation) to find the optimal parameters.
  - SGD has convergence guarantees (under certain conditions).
  - **Problem:** With non-convexity, we might converge to local optima.

$$\min_G L(G)$$

- **GANs instead involve two (or more) players**
  - Discriminator is trying to maximize its reward.
  - Generator is trying to minimize Discriminator's reward.

$$\min_G \max_D V(D, G)$$

  - SGD was not designed to find the Nash equilibrium of a game.
  - **Problem:** We might not converge to the Nash equilibrium at all.

Salimans, Tim, et al. "Improved techniques for training gans." *Advances in Neural Information Processing Systems*. 2016.

Challenges with GANs: Non-Convergence

- Simultaneous gradient descent is not guaranteed to converge for minimax objectives.
- Goodfellow et al. only showed convergence when updates are made in the function space.
- The parameterization of D and G results in highly non-convex objective.
- In practice, training tends to oscillate – **updates undo each other!**

Challenges with GANs: Non-Convergence

- Simultaneous gradient descent is not guaranteed to converge for minimax objectives.
- Goodfellow et al. only showed convergence when updates are made in the function space.
- The parameterization of D and G results in highly non-convex objective.
- In practice, training tends to oscillate – **updates undo each other!**

Potential Solutions (HACKS) https://github.com/soumith/ganhacks

How to Train a GAN? Tips and tricks to make GANs work

While research in Generative Adversarial Networks (GANs) continues to improve the fundamental stability of these models, we use a bunch of tricks to train them and make them stable day to day.

Here are a summary of some of the tricks.

Here's a link to the authors of this document

If you find a trick that is particularly useful in practice, please open a Pull Request to add it to the document. If we find it to be reasonable and verified, we will merge it in.

1. Normalize the inputs

- normalize the images between -1 and 1
- Tanh as the last layer of the generator output

2: A modified loss function

In GAN papers, the loss function to optimize G is `min (log 1-D)`, but in practice folks practically use `max log D`

- because the first formulation has vanishing gradients early on
- Goodfellow et. al (2014)

In practice, works well:

- Flip labels when training generator: real = fake, fake = real

3: Use a spherical Z

- Dont sample from a Uniform distribution

4: BatchNorm

- Construct different mini-batches for real and fake, i.e. each mini-batch needs to contain only all real images or all generated images.
- when batchnorm is not an option use instance normalization (for each sample, subtract mean and divide by standard deviation).

5: Avoid Sparse Gradients: ReLU, MaxPool

- the stability of the GAN game suffers if you have sparse gradients
- LeakyReLU = good (in both G and D)
- For Downsampling, use: Average Pooling, Conv2d + stride
- For Upsampling, use: PixelShuffle, ConvTranspose2d + stride
  - PixelShuffle: https://arxiv.org/abs/1609.05158

6: Use Soft and Noisy Labels

- Label Smoothing, i.e. if you have two target labels: Real=1 and Fake=0, then for each incoming sample, if it is real, then replace the label with a random number between 0.7 and 1.2, and if it is a fake sample, replace it with 0.0 and 0.3 (for example).
  - Salimans et. al. 2016
- make the labels the noisy for the discriminator: occasionally flip the labels when training the discriminator

7: DCGAN / Hybrid Models

- Use DCGAN when you can. It works!

8: Use stability tricks from RL

- Experience Replay
  - Keep a replay buffer of past generations and occassionally show them
  - Keep checkpoints from the past of G and D and occassionaly swap them out for a few iterations
- All stability tricks that work for deep deterministic policy gradients
- See Pfau & Vinyals (2016)

9: Use the ADAM Optimizer

- optim.Adam rules!
  - See Radford et. al. 2015
- Use SGD for discriminator and ADAM for generator

10: Track failures early

- D loss goes to 0: failure mode
- check norms of gradients: if they are over 100 things are screwing up
- when things are working, D loss has low variance and goes down over time vs having huge variance and spiking
- if loss of generator steadily decreases, then it's fooling D with garbage (says martin)

11: Dont balance loss via statistics (unless you have a good reason to)

- Dont try to find a (number of G / number of D) schedule to uncollapse training
- It's hard and we've all tried it.
- If you do try it, have a principled approach to it, rather than intuition

For example

```
while lossD > A:
  train D
while lossG > B:
  train G
```

12: If you have labels, use them

- if you have labels available, training the discriminator to also classify the samples: auxillary GANs

13: Add noise to inputs, decay over time

- Add some artificial noise to inputs to D (Arjovsky et. al., Huszar, 2016)
  - http://www.inference.vc/instance-noise-a-trick-for-stabilising-gan-training/
  - https://openreview.net/forum?id=Hk4_qw5xe
- adding gaussian noise to every layer of generator (Zhao et. al. EBGAN)
  - Improved GANs: OpenAI code also has it (commented out)

14: [notsure] Train discriminator more (sometimes)

- especially when you have noise
- hard to find a schedule of number of D iterations vs G iterations

15: [notsure] Batch Discrimination

- Mixed results

README

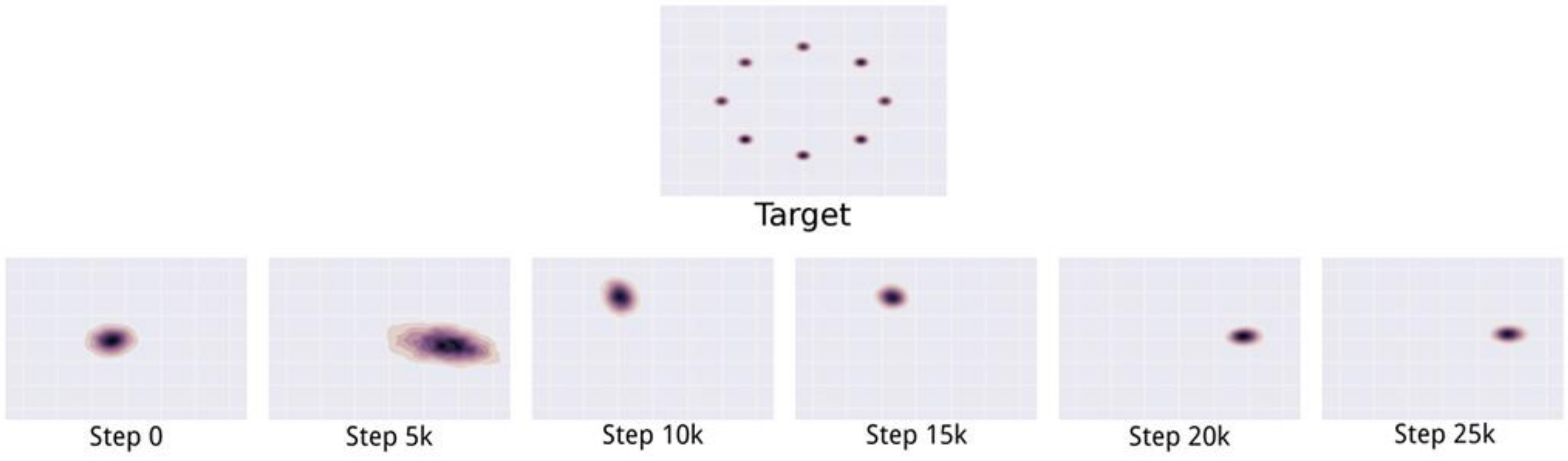
- Use an Embedding layer
- Add as additional channels to images
- Keep embedding dimensionality low and upsample to match image channel size

17: Use Dropouts in G in both train and test phase

- Provide noise in the form of dropout (50%).
- Apply on several layers of our generator at both training and test time
- https://arxiv.org/pdf/1611.07004v1.pdf

Challenges with GANs: Mode Collapse

The generator maps all z values to the x that is mostly likely to fool the discriminator.



Some real examples

Reed, S., et al. ***Generating interpretable images with controllable structure.*** Technical report, 2016. 2, 2016.

Challenges with GANs: Mode Collapse

Possible Solutions:

Wasserstein GAN

There are a large variety of divergence measures for distributions:

- **f-Divergences:** (e.g. Jensen-Shannon, Kullback-Leibler)

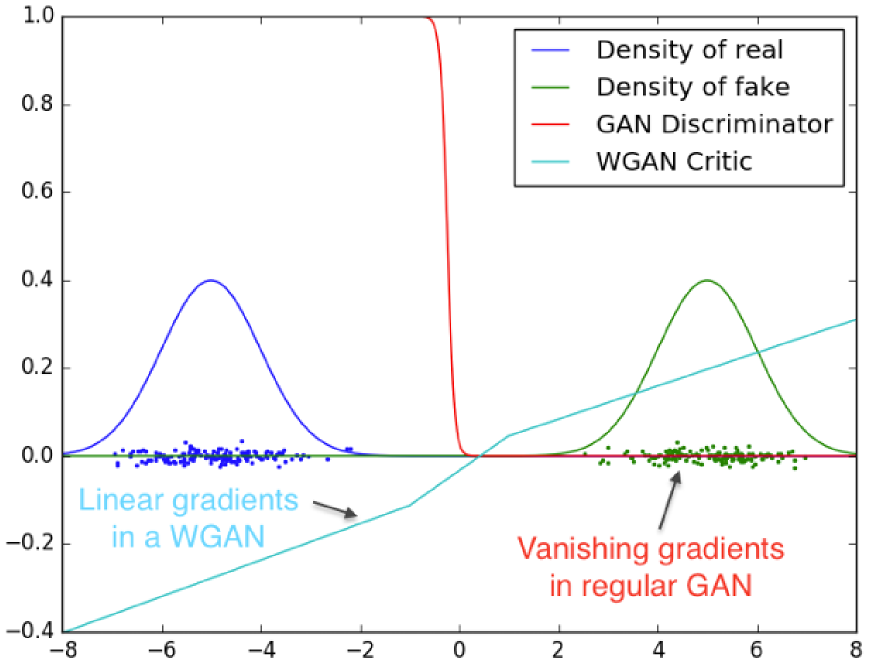
$$D_f\ (P\,||Q) = \int_{\chi} q(\mathbf{x}) f\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right) d\mathbf{x}$$

  - GANs [2], f-GANs [7], and more.

- **Integral Probability Metrics:** (e.g. Earth Movers Distance, Maximum Mean Discrepancy)

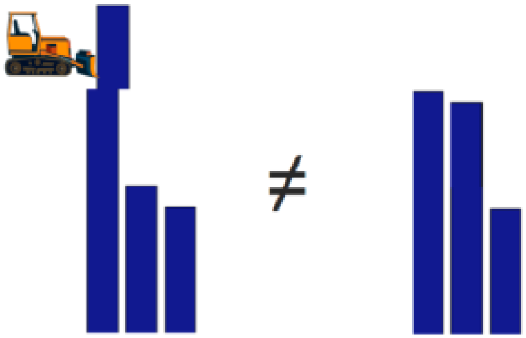
$$\gamma_F\ (P\,||Q) = \sup_{f \in F} \left| \int f dP - \int f dQ \right|$$

  - Wasserstein GANs [1], Fisher GANs [6], Sobolev GANs [5] and more.



WGAN

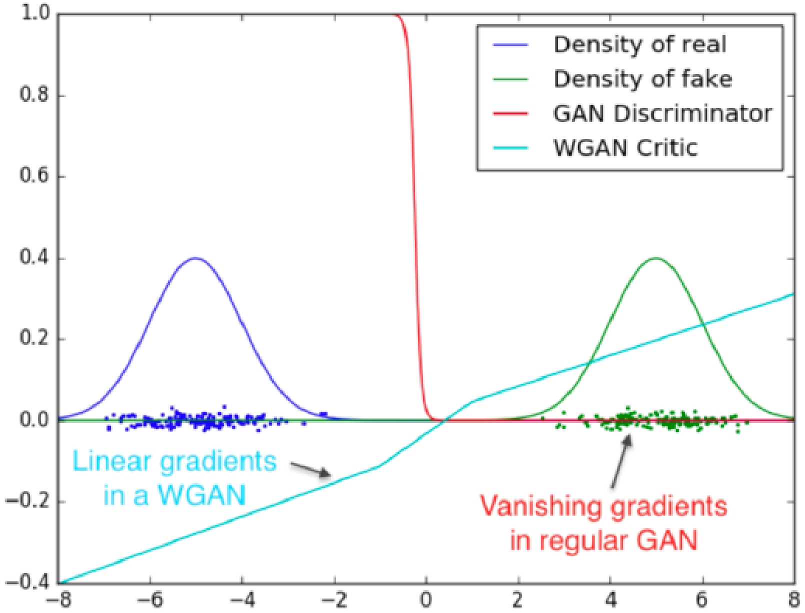
- If our data are on a low-dimensional manifold of a high dimensional space the model's manifold and the true data manifold can have a negligible intersection in practice
- KL divergence is undefined or infinite
- The loss function and gradients may not be continuous and well behaved
- The Earth Mover's Distance is well defined:
    - Minimum transportation cost for making one pile of dirt (pdf/pmf) look like the other



WGAN

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -[\mathbb{E}_{x\sim p_{data}} D(x) - \mathbb{E}_z D(G(z))]$$
$$J^{(G)}(\theta^{(D)}, \theta^{(G)}) = -\mathbb{E}_z D(G(z))$$

- Importantly, the discriminator is trained for many steps before the generator is updated
- Gradient-clipping is used in the discriminator to ensure $D(x)$ has the Lipschitz continuity required by the theory
- The authors argue that this solves many training issues, including mode collapse



Conditional GANs

MNIST digits generated conditioned on their class label.

$[1, 0, 0, 0, 0, 0, 0, 0, 0, 0] \rightarrow$
$[0, 1, 0, 0, 0, 0, 0, 0, 0, 0] \rightarrow$
$[0, 0, 1, 0, 0, 0, 0, 0, 0, 0] \rightarrow$
$[0, 0, 0, 1, 0, 0, 0, 0, 0, 0] \rightarrow$
$[0, 0, 0, 0, 1, 0, 0, 0, 0, 0] \rightarrow$
$[0, 0, 0, 0, 0, 1, 0, 0, 0, 0] \rightarrow$
$[0, 0, 0, 0, 0, 0, 1, 0, 0, 0] \rightarrow$
$[0, 0, 0, 0, 0, 0, 0, 1, 0, 0] \rightarrow$
$[0, 0, 0, 0, 0, 0, 0, 0, 1, 0] \rightarrow$
$[0, 0, 0, 0, 0, 0, 0, 0, 0, 1] \rightarrow$

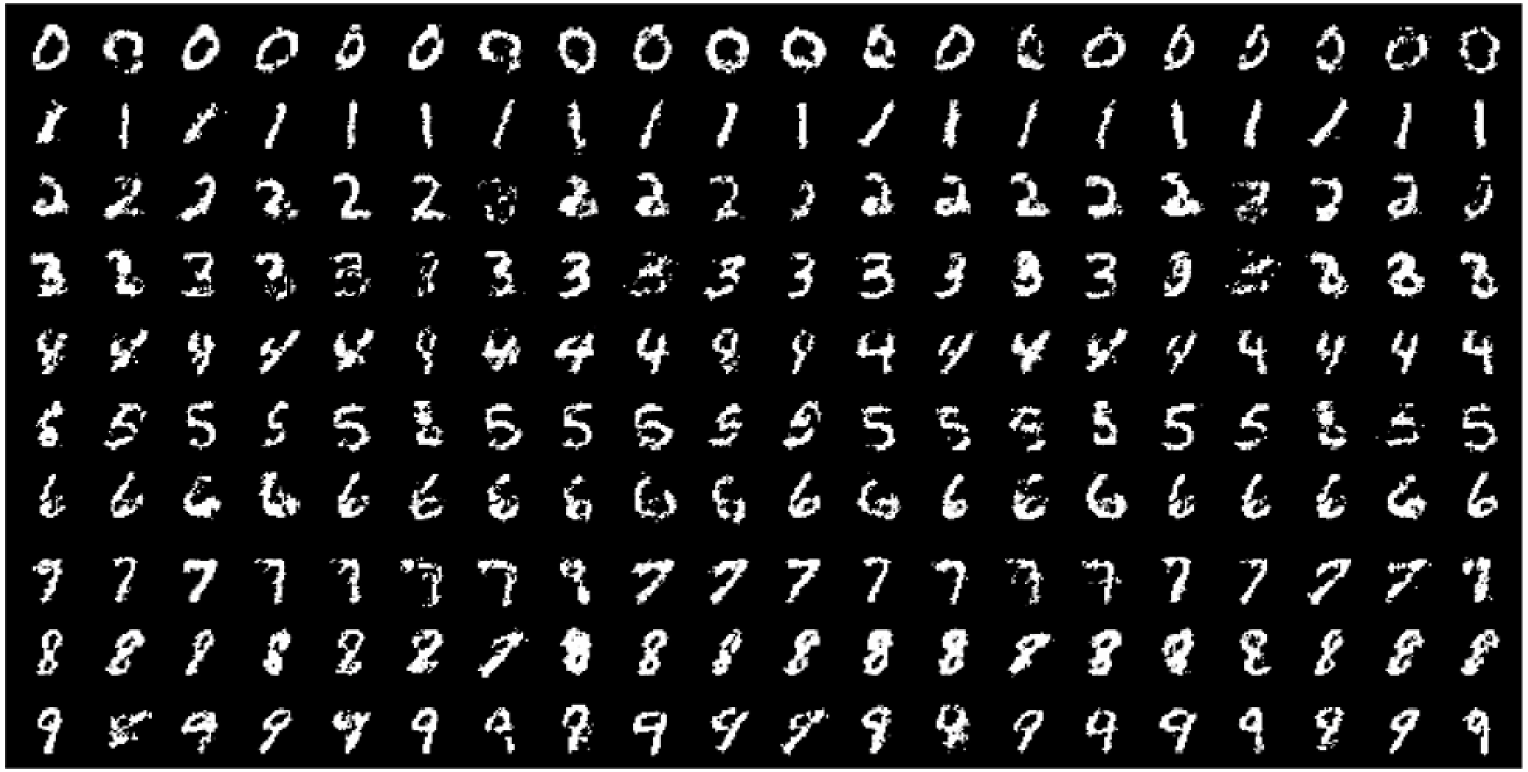
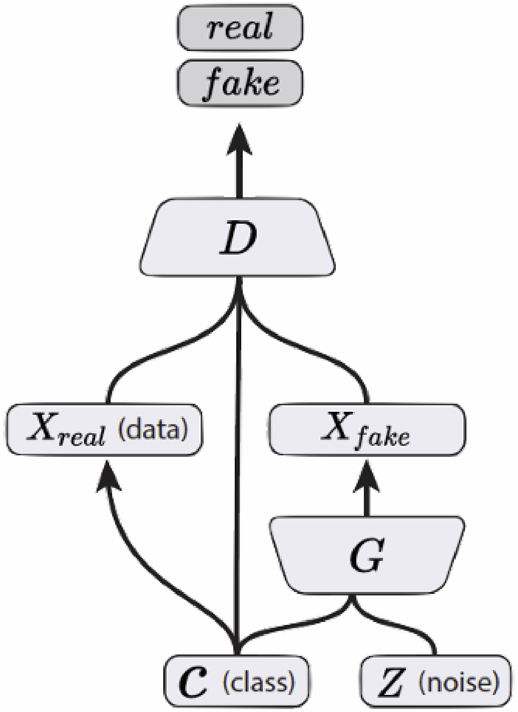


Figure 2 in the original paper.

Mirza, Mehdi, and Simon Osindero. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784 (2014).

Conditional GANs

- Simple modification to the original GAN framework that conditions the model on *additional information* for better multi-modal learning.
- Lends to many practical applications of GANs when we have explicit *supervision* available.



Conditional GAN
(Mirza & Osindero, 2014)

Image Credit: Figure 2 in Odena, A., Olah, C. and Shlens, J., 2016. Conditional image synthesis with auxiliary classifier GANs. *arXiv preprint arXiv:1610.09585*.

Mirza, Mehdi, and Simon Osindero. "**Conditional generative adversarial nets**". arXiv preprint arXiv:1411.1784 (2014).

Image-to-Image Translation

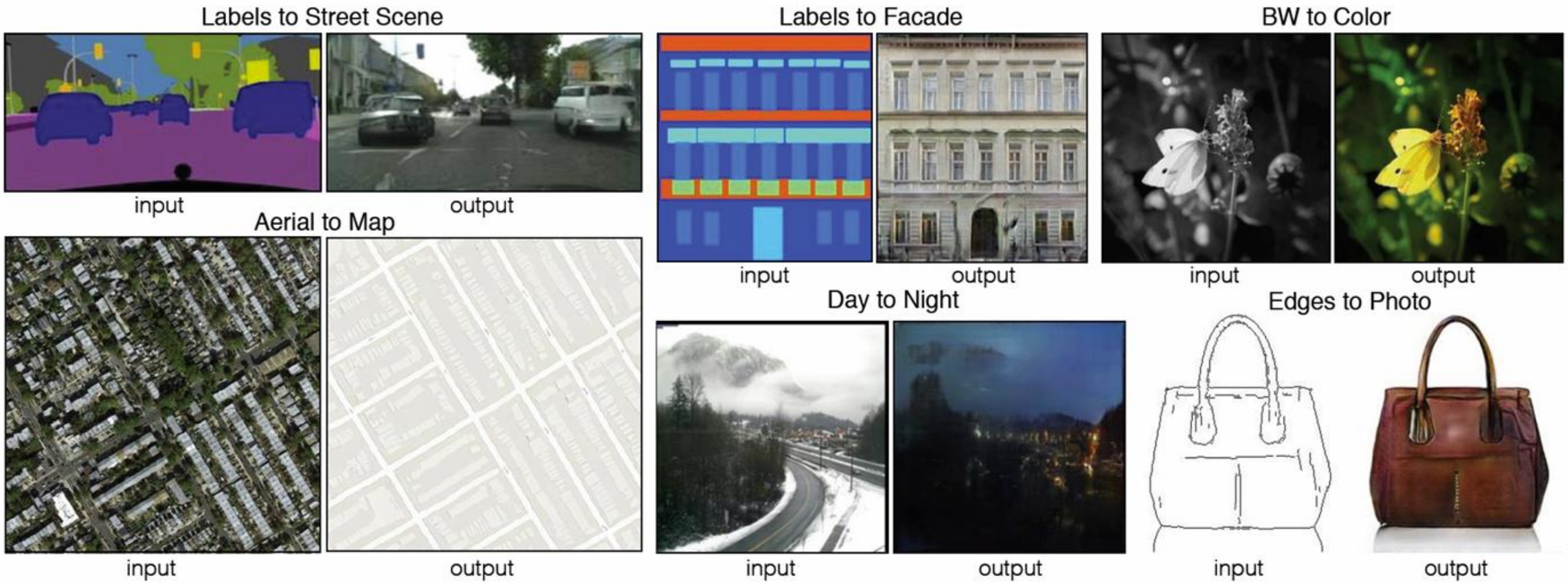


Figure 1 in the original paper.

[Link to an interactive demo of this paper]

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. "**Image-to-image translation with conditional adversarial networks**". arXiv preprint arXiv:1611.07004. (2016).

Image-to-Image Translation

- Architecture: *DCGAN*-based architecture
- Training is conditioned on the images from the source domain.
- Conditional GANs provide an effective way to handle many complex domains without worrying about designing *structured loss* functions explicitly.

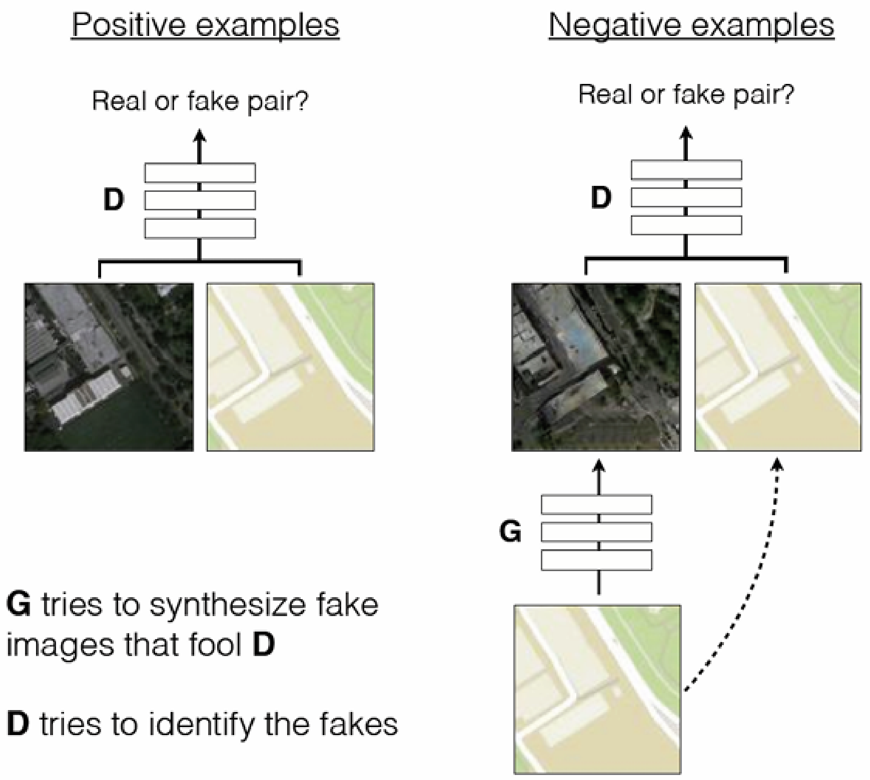


Figure 2 in the original paper.

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. "**Image-to-image translation with conditional adversarial networks**". arXiv preprint arXiv:1611.07004. (2016).

Text-to-Image Synthesis

Motivation

Given a text description, generate images closely associated.

Uses a conditional GAN with the generator and discriminator being condition on "dense" text embedding.

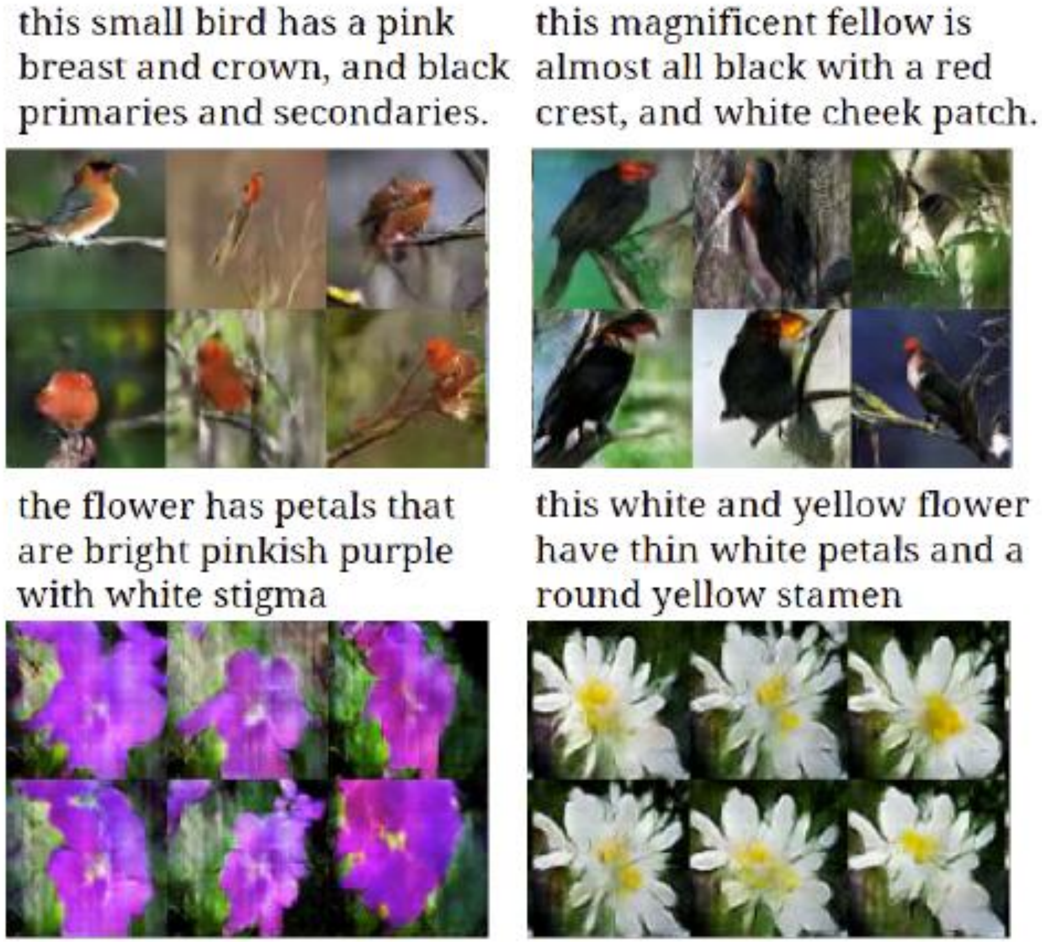


Figure 1 in the original paper.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. "**Generative adversarial text to image synthesis**". ICML (2016).

Text-to-Image Synthesis

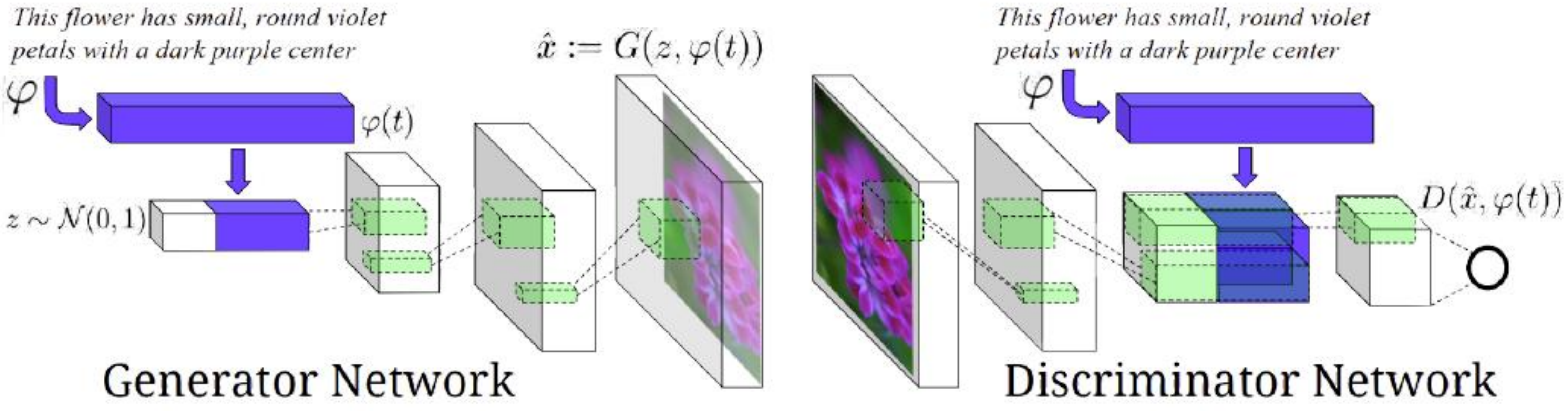


Figure 2 in the original paper.

Positive Example:

Real Image, Right Text

Negative Examples:

Real Image, Wrong Text

Fake Image, Right Text

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. "**Generative adversarial text to image synthesis**". ICML (2016).

Face Aging with Conditional GANs

- Differentiating Feature: Uses an *Identity Preservation Optimization* using an auxiliary network to get a better approximation of the latent code (z*) for an input image.
- Latent code is then conditioned on a discrete (one-hot) embedding of age categories.

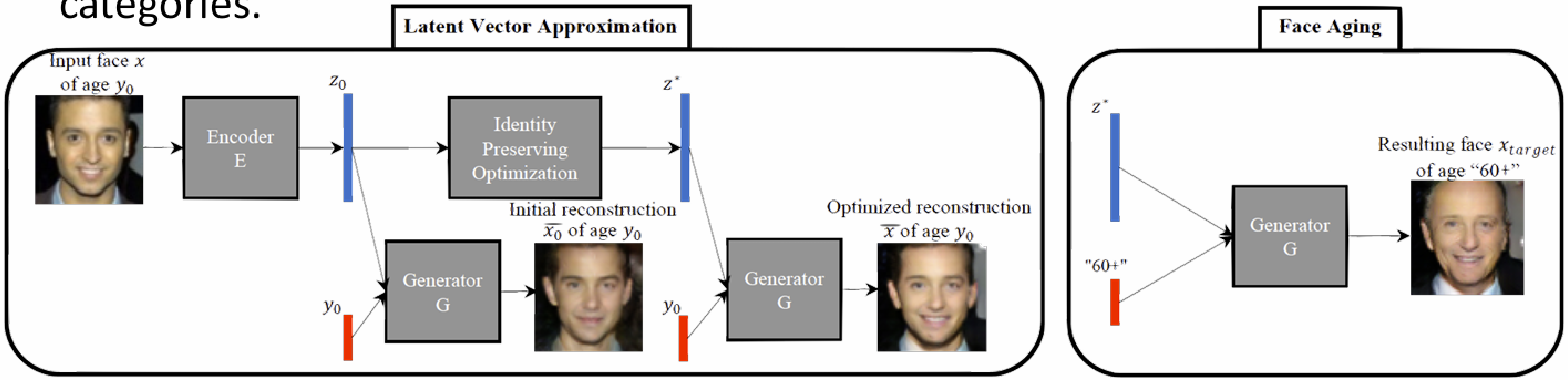


Figure 1 in the original paper.

Antipov, G., Baccouche, M., & Dugelay, J. L. (2017). "**Face Aging With Conditional Generative Adversarial Networks**". arXiv preprint arXiv:1702.01983.

Face Aging with Conditional GANs

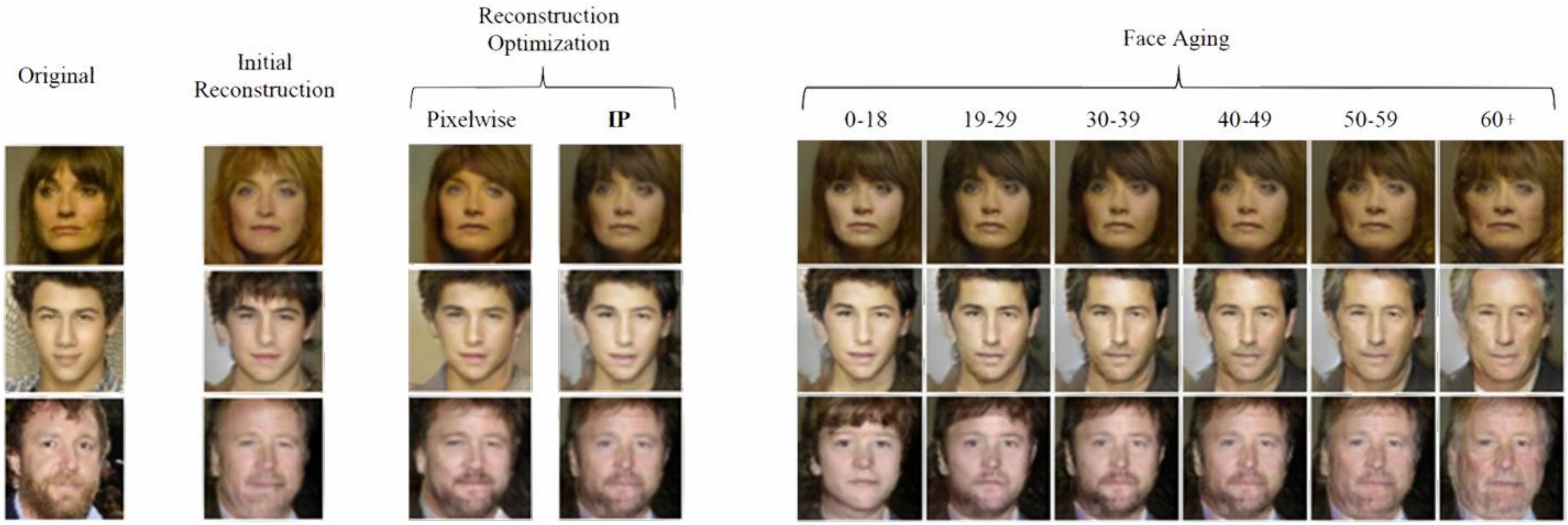


Figure 3 in the original paper.

Antipov, G., Baccouche, M., & Dugelay, J. L. (2017). "**Face Aging With Conditional Generative Adversarial Networks**". arXiv preprint arXiv:1702.01983.

Conditional GANs

Conditional Model Collapse

- Scenario observed when the Conditional GAN starts *ignoring* either the code (c) or the noise variables (z).
- This limits the diversity of images generated.

Credit?

Mirza, Mehdi, and Simon Osindero. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784 (2014).

Additional Sources:

There are lots of excellent references on GANs:

- Sebastian Nowozins presentation at MLSS 2018:
  https://github.com/nowozin/mlss2018-madrid-gan
- NIPS 2016 tutorial on GANs by Ian Goodfellow:
  https://arxiv.org/abs/1701.00160
- A nice explanation of Wasserstein GANs by Alex Irpan:
  https://www.alexirpan.com/2017/02/22/wasserstein-gan.html