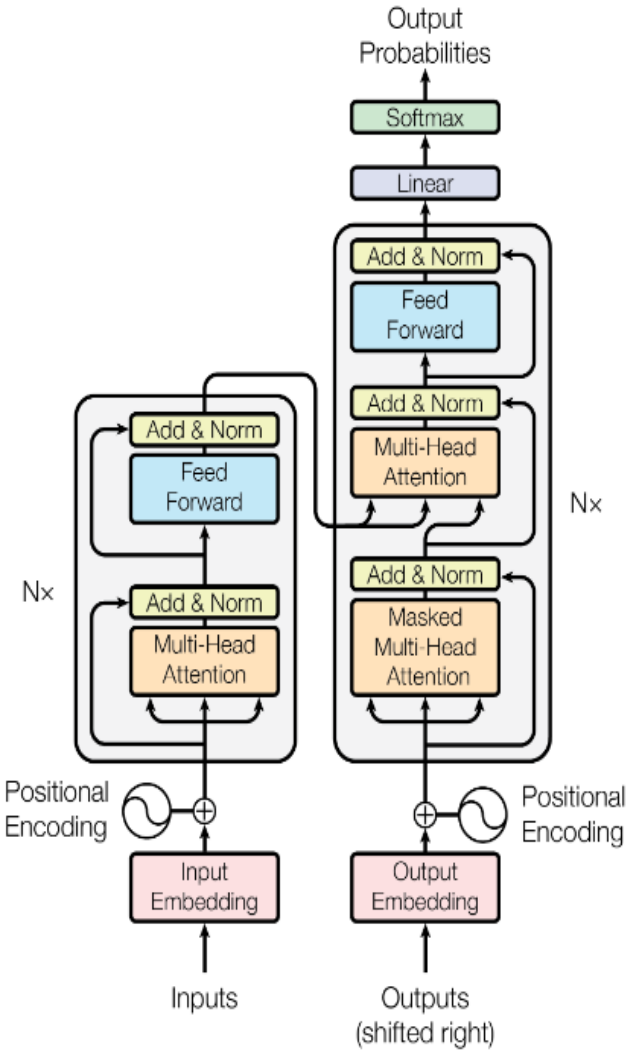
Deep Generative Models

The architectures lectures

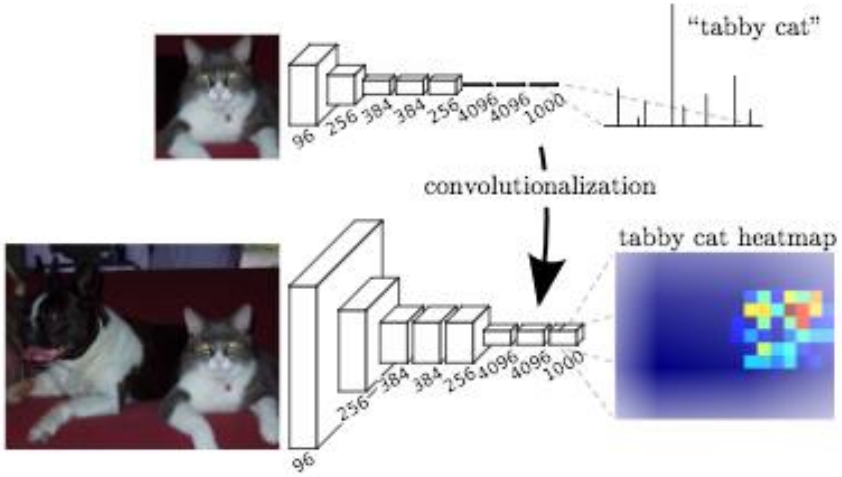
- Basic pieces
  - Dense
  - Conv
  - RNNs
  - Attention
  - Residual connection
  - Upsampling, Deconv
- Blocks
  - Squeeze and Excite
  - Transformer
- Tricks
  - BN, LN
  - Augmentation

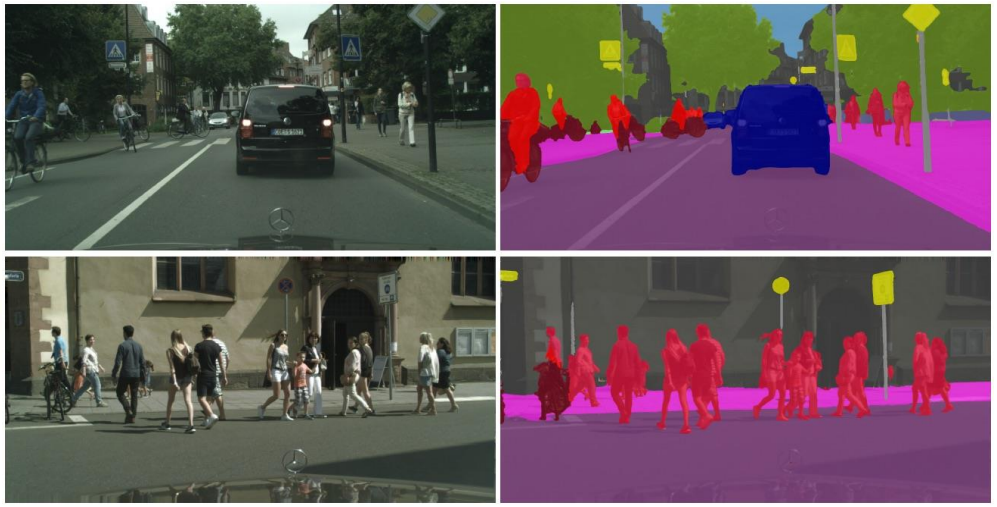


Later lectures will focus on other things beside the architecture

Sometimes you want to increase the size of your feature map

Image segmentation



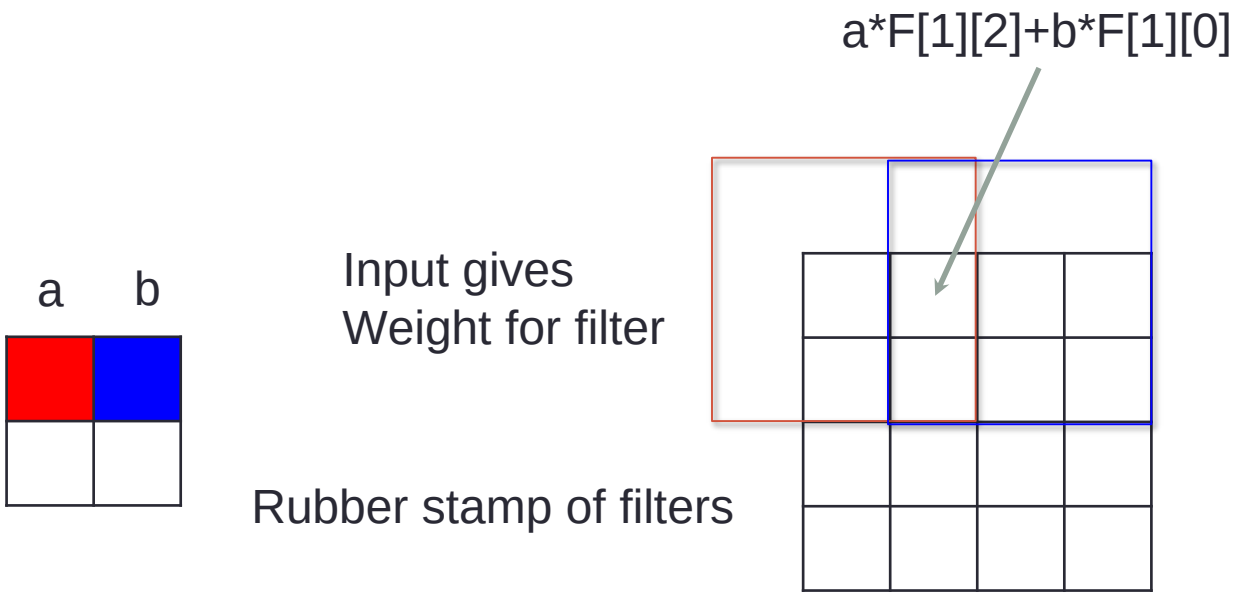


2 main approaches to upsample
De-convolution
resize (unpooling) +convolution

https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf
http://vladlen.info/publications/feature-space-optimization-for-semantic-video-segmentation/

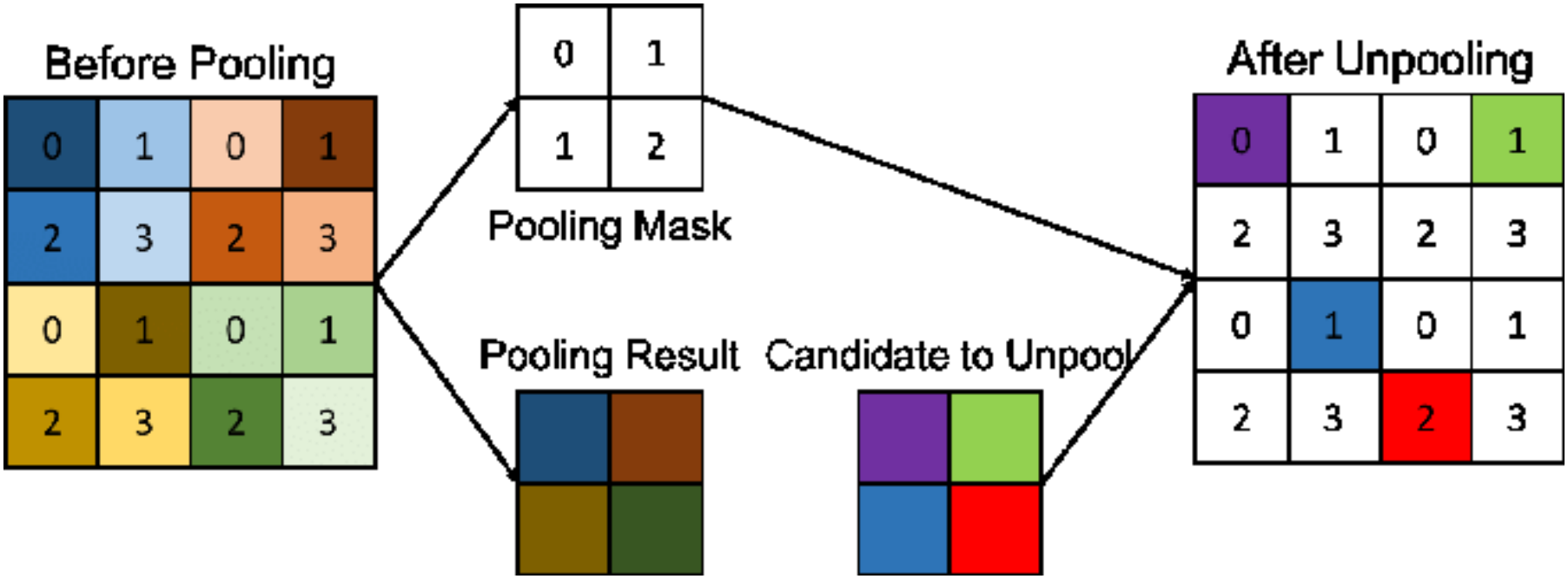
De-convolution (Upsampling)

- 3x3 de-convolution filter, stride 2, pad 1



Other names because this name sucks (for me)
- Convolution transpose, upconvolution, backward strided convolution

Unpooling + conv



Remember the location of the max value

Put the value to unpool at that location

Fill the rest with zeroes

Follow by regular convolution to smooth the image

Fully Automatic Acute Ischemic Lesion Segmentation in DWI Using Convolutional Neural Networks, 2017.

Upsampling notes

Deconvolution filter size should be a multiple of the stride to avoid checkerboard artifacts

Unpooling can be replaced with regular image resizing techniques (interpolation/upsample

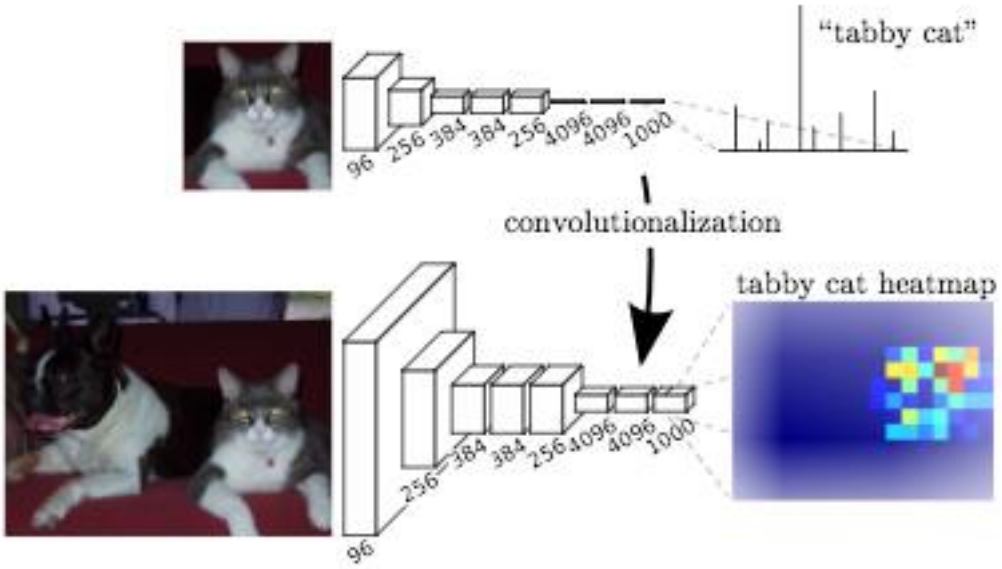
https://pytorch.org/docs/stable/generated/torch.nn.Upsample.html)

Image using deconv

Image using resize upsampling

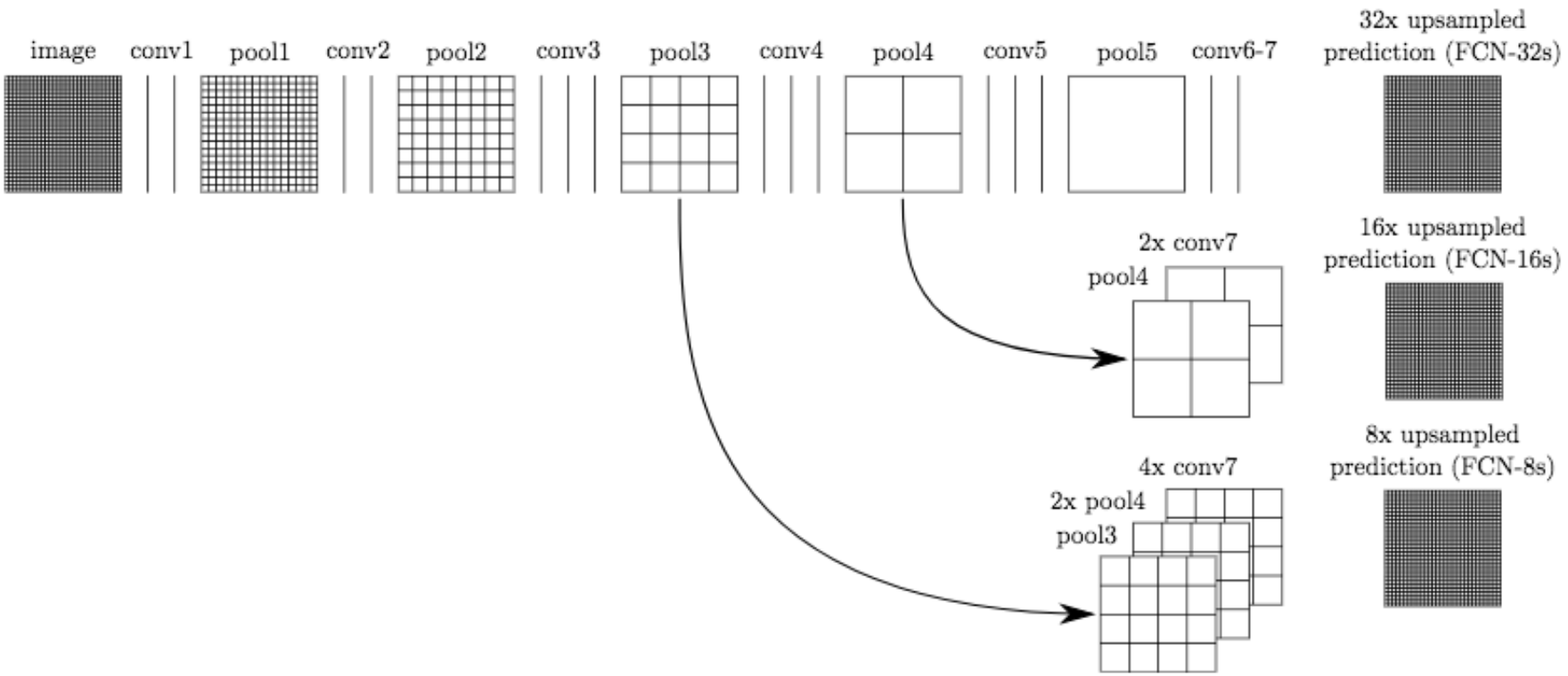
https://distill.pub/2016/deconv-checkerboard/

De-convolution for segmentation



https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

De-convolution for segmentation



https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

De-convolution for segmentation

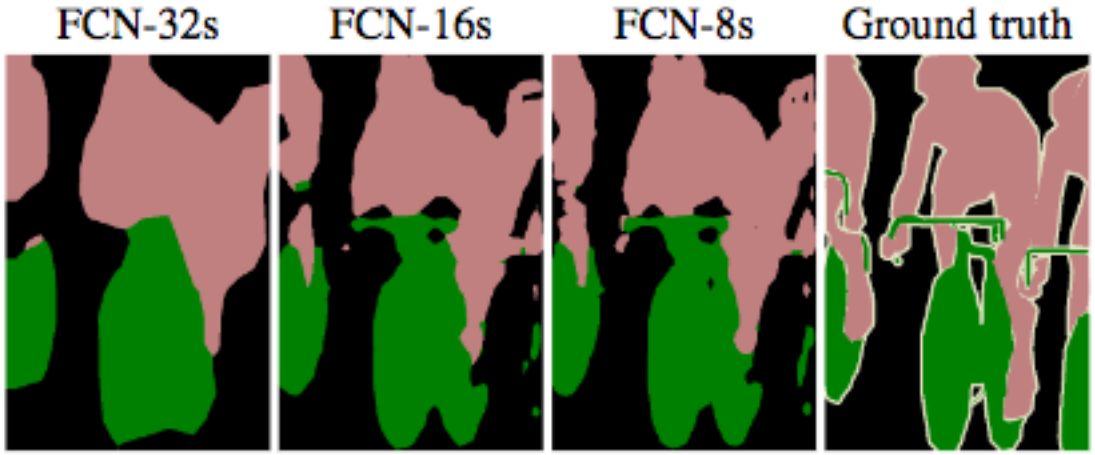


Figure 4. Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3).

https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

Unet

- A typical architecture for segmentation task
- Have a skip connection to preserve resolution

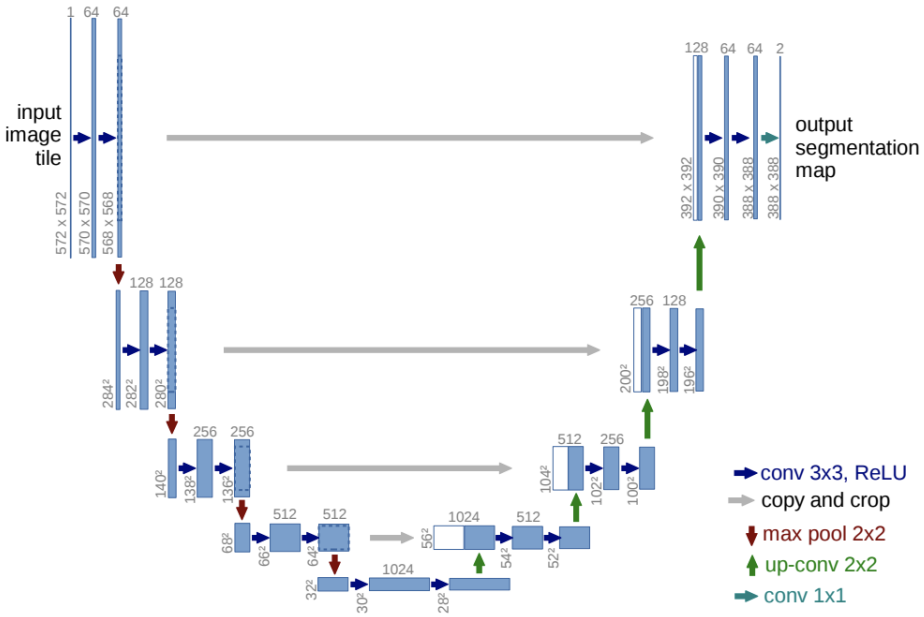


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Implementation

- Codes usually written as blocks

https://github.com/milesial/Pytorch-UNet/

```python
class UNet(nn.Module):
    def __init__(self, n_channels, n_classes, bilinear=False):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bilinear = bilinear

        self.inc = DoubleConv(n_channels, 64)
        self.down1 = Down(64, 128)
        self.down2 = Down(128, 256)
        self.down3 = Down(256, 512)
        factor = 2 if bilinear else 1
        self.down4 = Down(512, 1024 // factor)
        self.up1 = Up(1024, 512 // factor, bilinear)
        self.up2 = Up(512, 256 // factor, bilinear)
        self.up3 = Up(256, 128 // factor, bilinear)
        self.up4 = Up(128, 64, bilinear)
        self.outc = OutConv(64, n_classes)

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
```

```python
class Up(nn.Module):
    """Upscaling then double conv"""

    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()

        # if bilinear, use the normal convolutions to reduce the number of channels
        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
            self.conv = DoubleConv(in_channels, out_channels, in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels, in_channels // 2, kernel_size=2, stride=2)
            self.conv = DoubleConv(in_channels, out_channels)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        # input is CHW
        diffY = x2.size()[2] - x1.size()[2]
        diffX = x2.size()[3] - x1.size()[3]

        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])
        # if you have padding issues, see
        # https://github.com/HaiyongJiang/U-Net-Pytorch-Unstructured-Buggy/commit/0e854509c2cea854
        # https://github.com/xiaopeng-liao/Pytorch-UNet/commit/8ebac70e633bac59fc22bb5195e513d5832
        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)
```

Guide on choosing the architecture

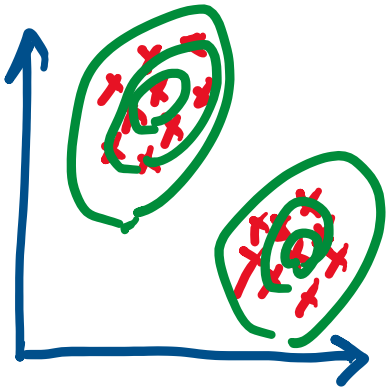
- Is it solved before? -> Copy!
- Is it similar to some standard vision/NLP task? -> copy with modification
- Nothing close?
  - What would a human do?
  - What kind of inductive bias should we put in?
    - What kind of invariance does the data has?
    - What kind of noise/augmentation would work?
- Start simple then add extra
- Overfit then think of ways to regularize

Deep Generative Models

Generative modeling

- Learns P(X,Y) or P(X|Y)
  - Can sample (generates) X from P(X|Y)
  - Y is the controlling parameter

- What is P(X|Y)?
  - Most of the time it's assumed to be a parametric distribution



Generating real world data

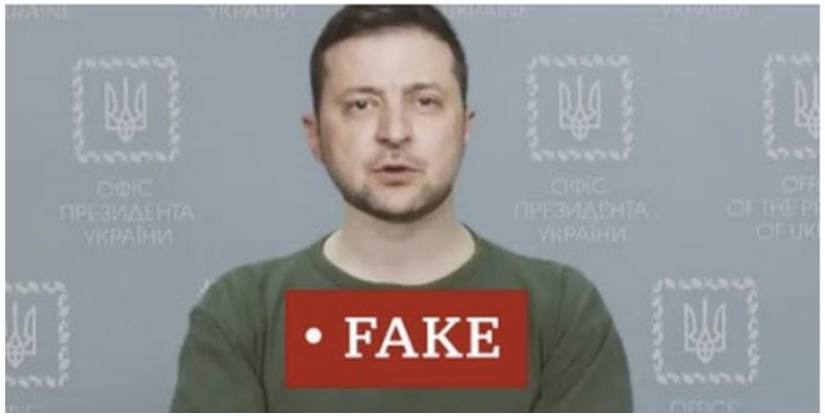
- Let’s say we want to generate faces
  - How do we fit a distribution to it?

Deepfake presidents used in Russia-Ukraine war

By Jane Wakefield
BBC Technology

5 days ago

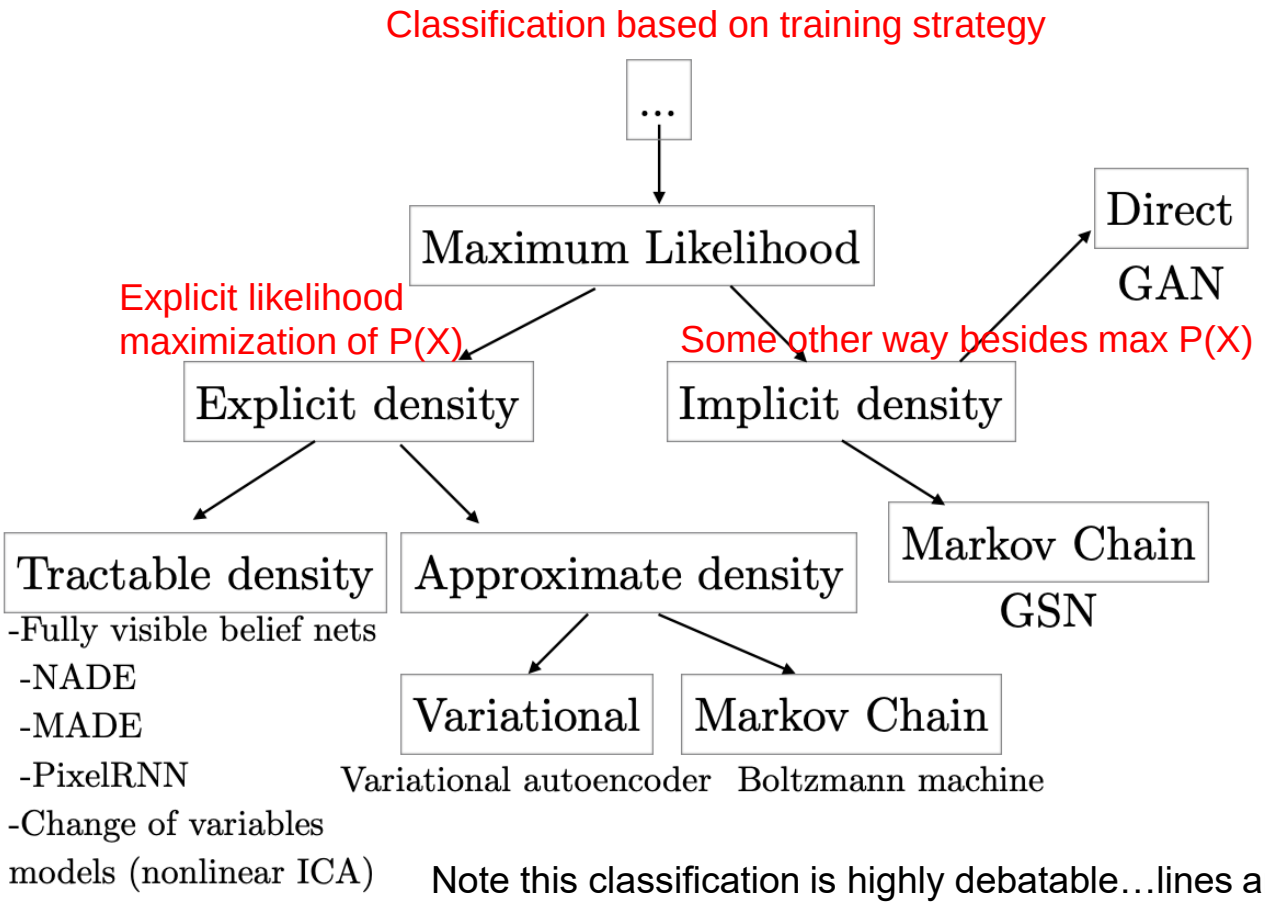
Russia-Ukraine war



https://www.bbc.com/news/technology-60780142

Implicit generative models

- We can generate without explicitly modeling P(X)

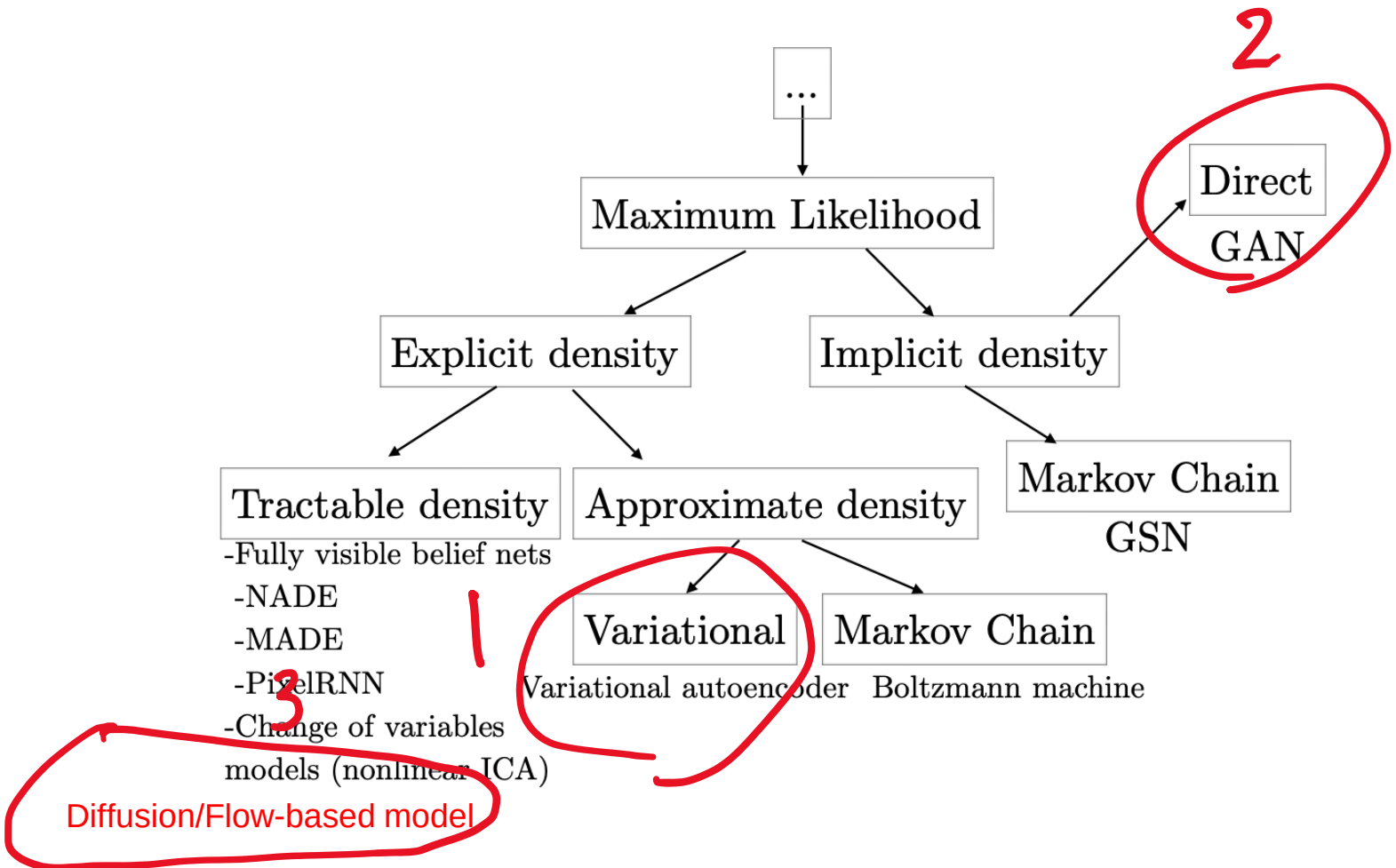


Note this classification is highly debatable…lines are often blurred
Think of sliders, not check boxes.

https://arxiv.org/pdf/1701.00160.pdf

Deep Generative Models

- A deep learning model that can be used to generate X

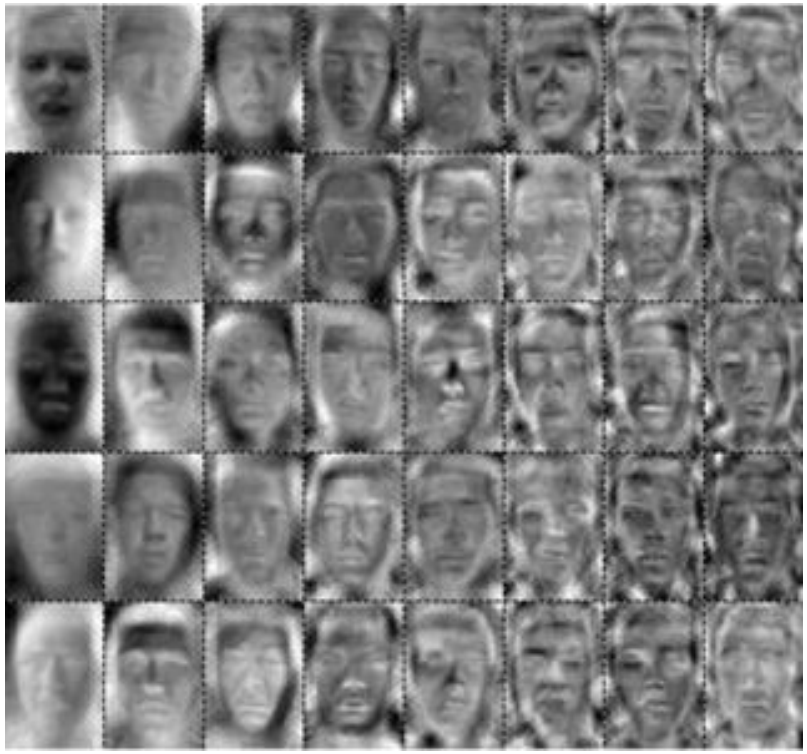


Deep generative models

- Autoencoders
  - Variational autoencoders
- GAN
- Diffusion/Flow

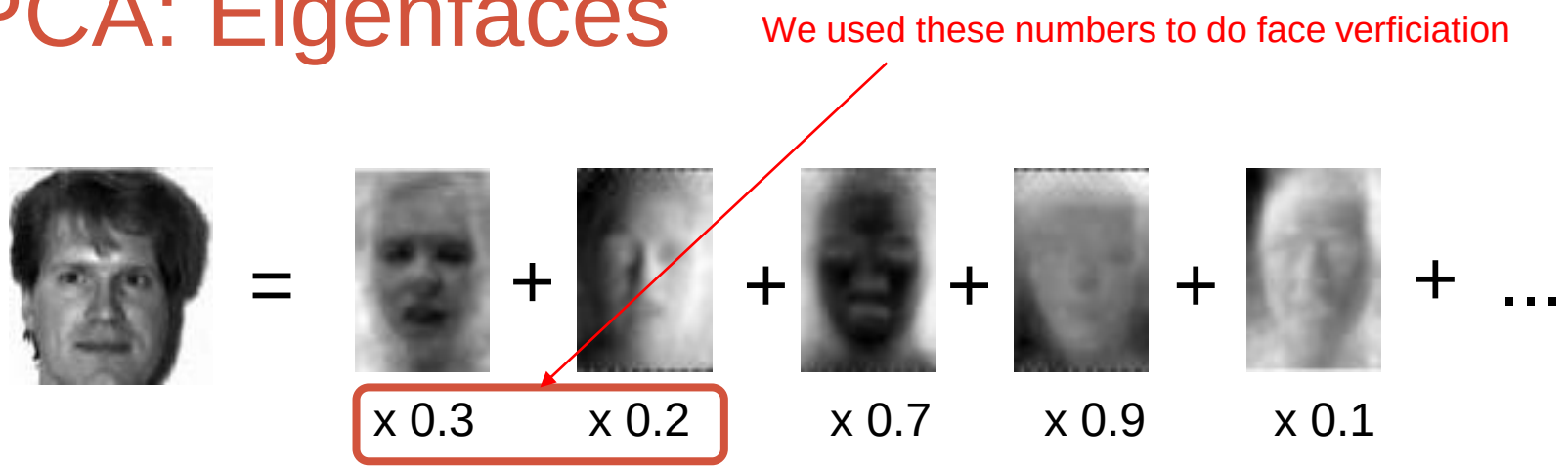
Encoding and reconstructing an image

- We know we can reconstruct a face using PCA to map high dimension data to low dimension data



PCA basis (Eigenfaces)

PCA: Eigenfaces

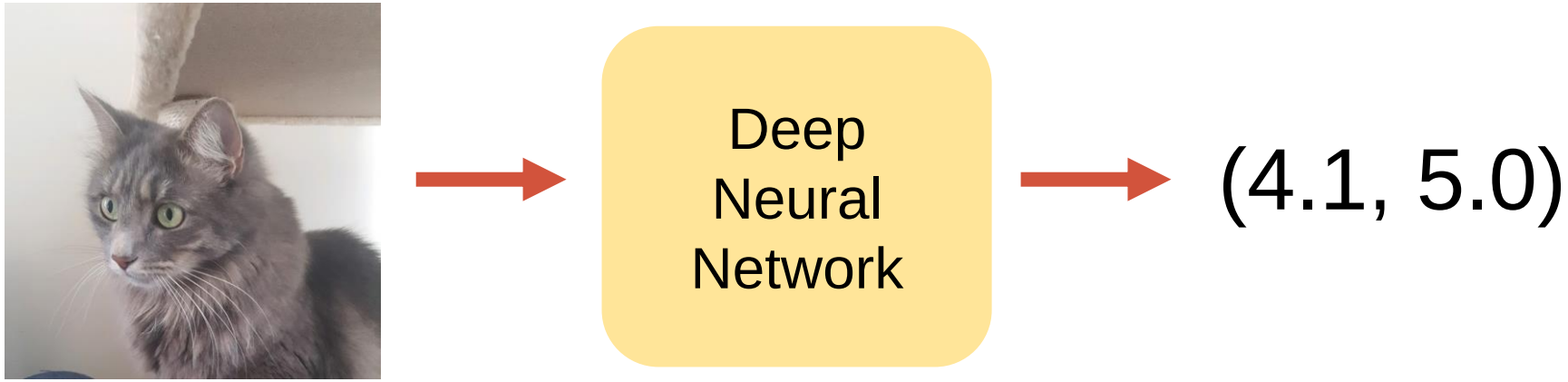


Problems:

1. PCA is a linear operation (as well as LDA or fisherface). Many interesting attributes are highly non-linear, e.g., identity, emotion, and cannot be factored linearly.
2. Faces have to be aligned and centered for this to work
3. Illumination dominates!

How about DNN?

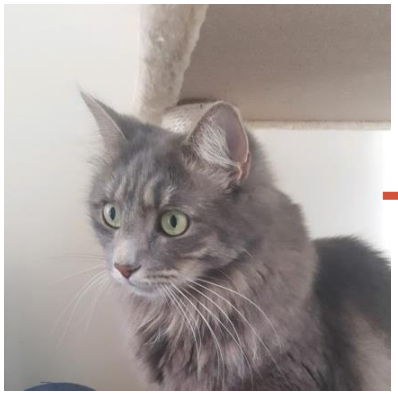
We know that DNN can model highly complex mapping.
How about modeling f with a neural net?



But how to train it? We don't know the target, groundtruth, two numbers.

Autoencoder

Auto means self!



Encoder → (?, ?) → Decoder →

Input

X

Output

X'

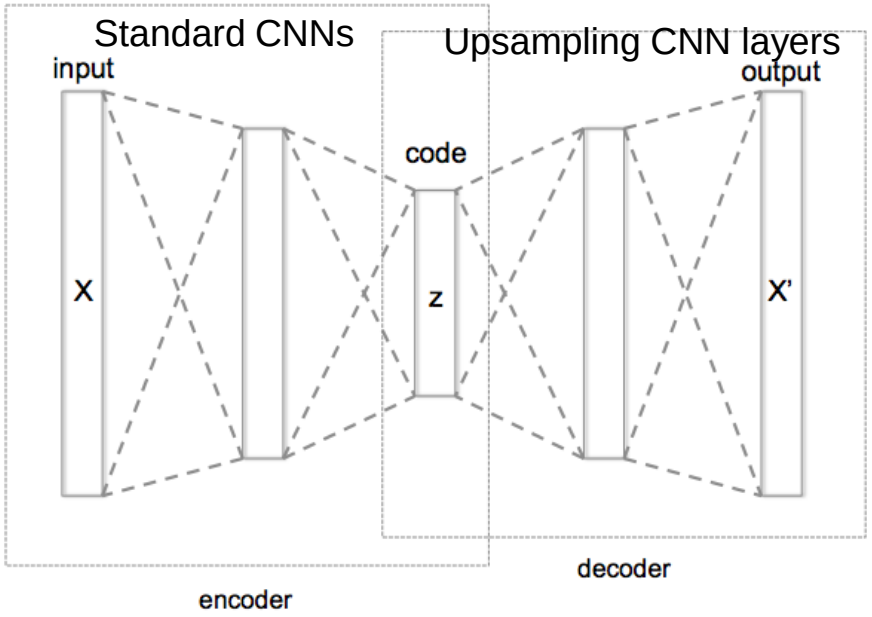
Now we can train the network with L2 Loss: $\|X - X'\|^2$

Autoencoder

Auto means self!

Input

X

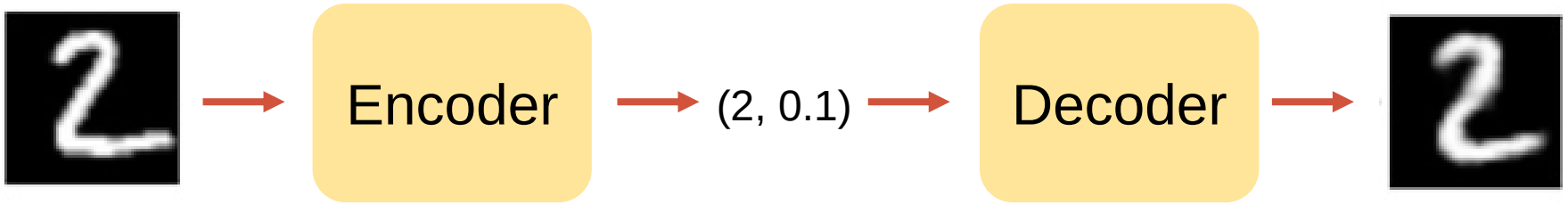


Output

X'

Now we can train the network with L2 Loss: $\|X - X'\|^2$

Autoencoding Game



Both Encoder and Decoder see the entire dataset.

Encoder: What two numbers should I send you so that you can re-draw my input as accurate as possible?

Encoder: Maybe the first number would directly correspond to what digit it is. The second, perhaps, the thickness of the stroke?

Turns out DNN does surprisingly well at encoding important attributes.

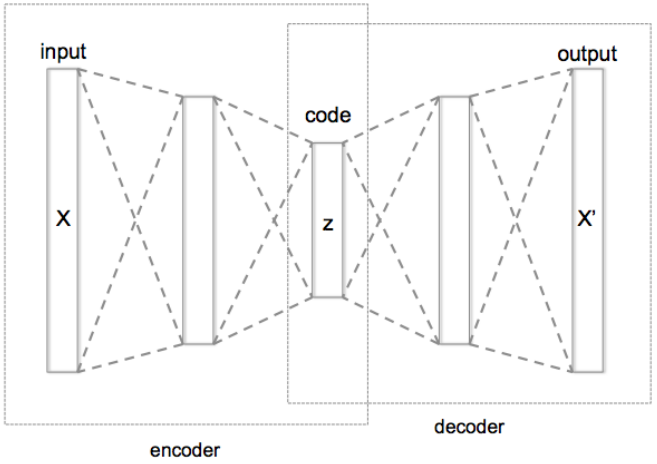
Autoencoder

Can we pick an arbitrary dimension for the code?

- When z has the same dimension as the input and output, NN cheats and outputs an identity mapping -> useless.

Input

X

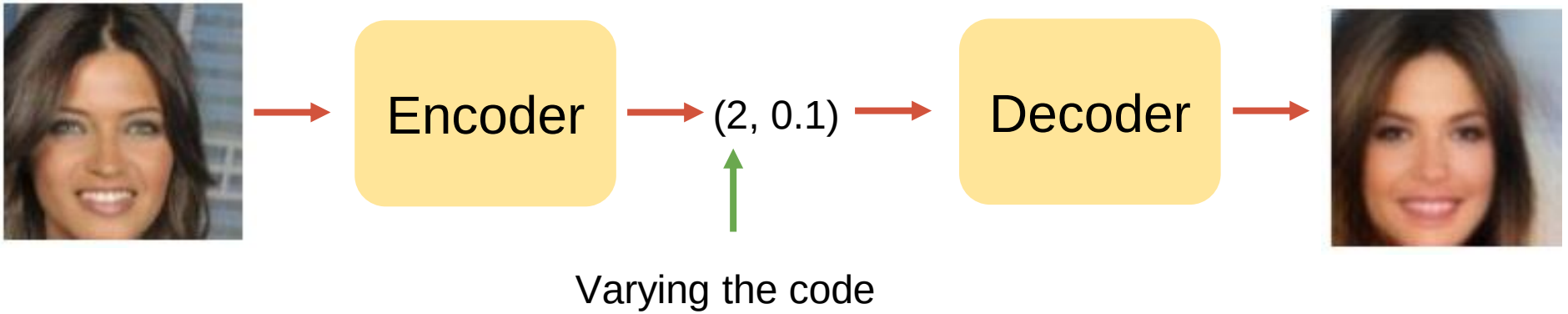


Output

X'

- When z dim << input dim, NN has to work hard!

Inspecting attributes learned from AE



Deep Feature Consistent Variational Autoencoder [Hou et al. 2016]

Attributes from autoencoder

Deep Feature Consistent Variational Autoencoder [Hou et al. 2016]

Utilizing autoencoder for supervised tasks

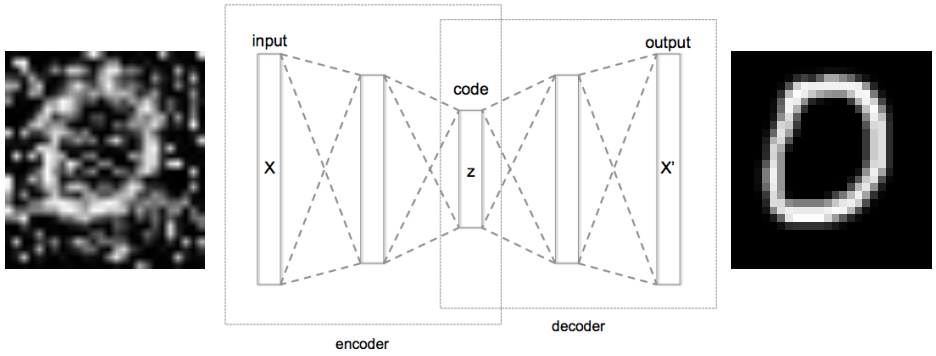
Many ways to help with a supervised task such as recognition.

1. Append the input with the code from the encoder
2. Stick a classifier on top of the encoder
3. Used for pretraining a network

Denoising autoencoder

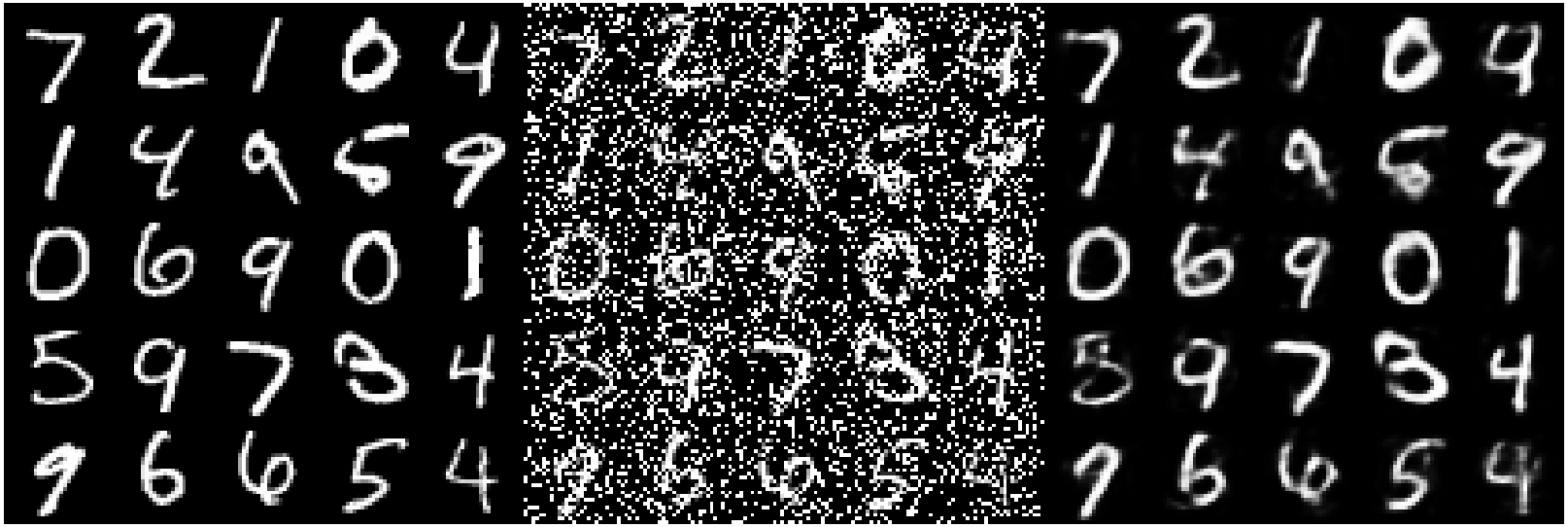
- Motivation: Making AE more robust -- really learn what's important

- An autoencoder that denoise corrupted inputs
  - Blur image, sound corrupted by noise
- How?
  - Corrupt your input x -> x''
  - Use x'' as the input, and minimize the same loss
  - Loss = $||x-x'||^2$



Example

Original Corrupted Generated

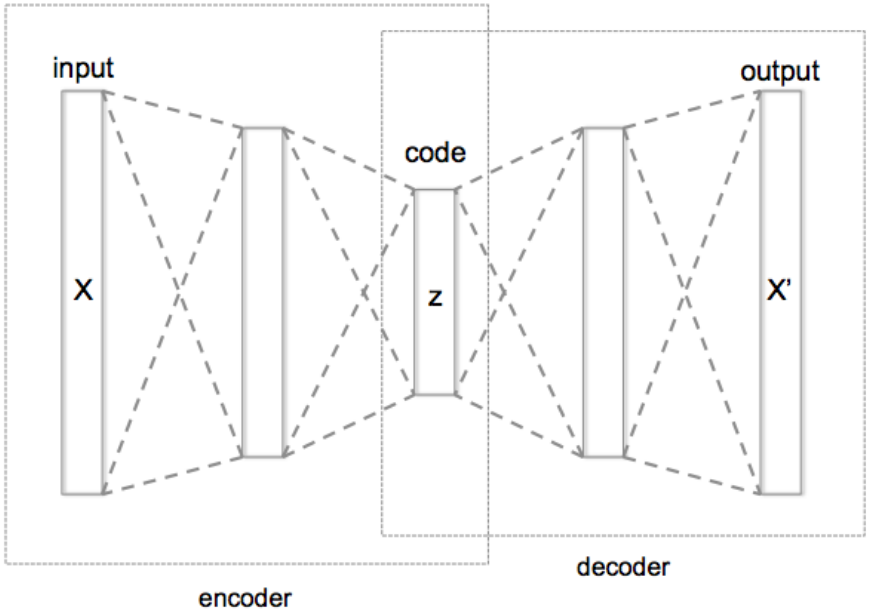


https://www.doc.ic.ac.uk/~js4416/163/website/autoencoders/denoising.html

Other uses

- Besides denoising, can also be used to make your “code” more robust to the type of corruption you introduced.
- Finding similar images. Combining with clustering algos.
- Generative models / generating photos

Autoencoders



- Automatically learn features
- Non-linear dimensionality reduction / data compression

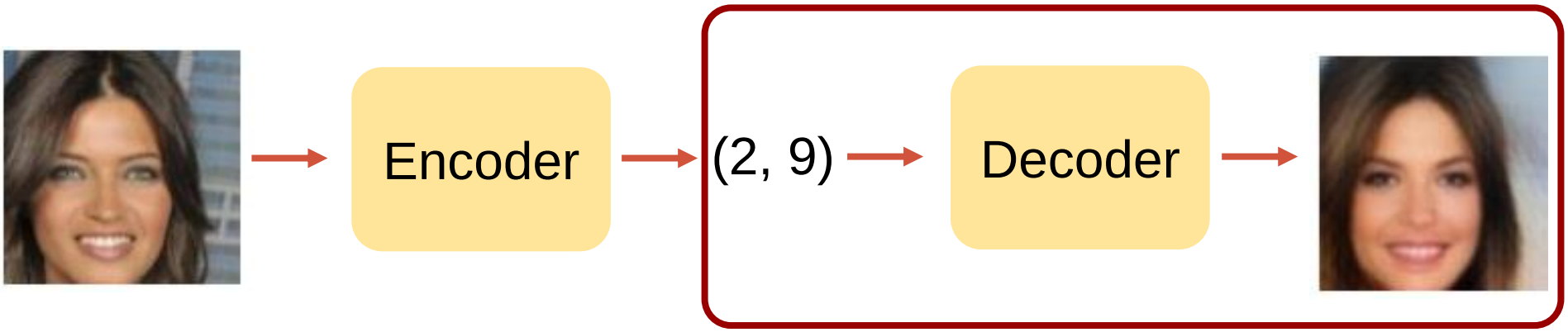
Variational Autoencoder

Original and by-product motivations:

1. Allows approximation of intractable posterior.

2. Generate photos from the decoder

Motivation #2 -- Generating Photos



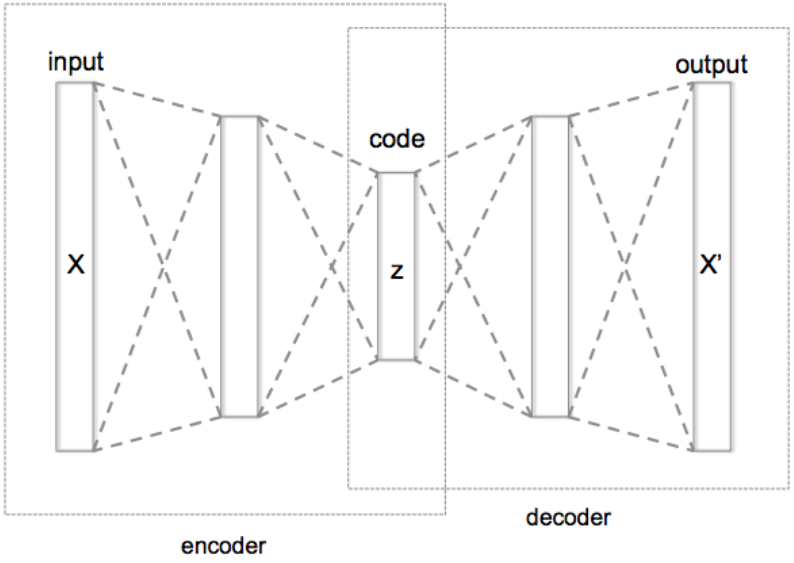
Use this part to generate an image from random code.

But how do you generate the two numbers? Uniform? Gaussian? Encoder could output two numbers with some crazy distribution.

VAE allows us to force the latent code to follow some distribution like a unit normal distribution.

Motivation #1 -- Solving Intractable Inference

Autoencoder can be viewed from a probabilistic perspective.



However, a standard AE isn't quite "probabilistic."

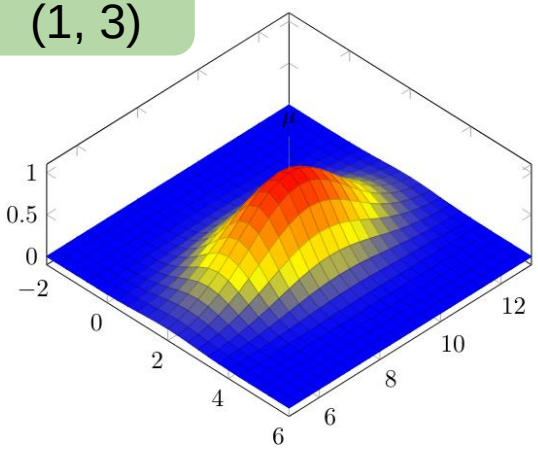
I.e., the encoder deterministically outputs z given x. Rather we want a distribution of the code (posterior p(z | x)).

Standard AE

Encoder → (2, 9) → Decoder →

Probabilistic AE

Encoder → Means (2, 9) / Std. (1, 3) → Sample → (2.1, 9.2) → Decoder →



Autoencoding Variational Bayes: Bayesian Review

One important inference problem : Posterior inference

$$p(\mathbf{z} \mid \mathbf{x})$$

$\mathbf{x}$ Observed data

$\mathbf{z}$ Latent variable (hidden code)

Bayesian Review

One important inference problem : Posterior inference

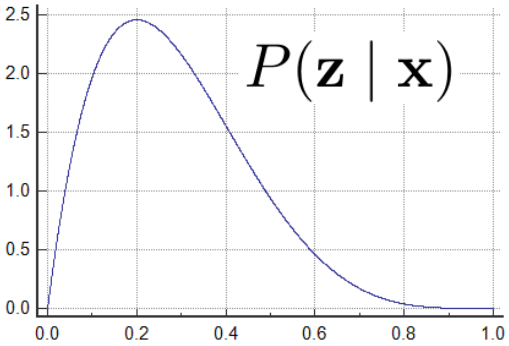
$$p(\mathbf{z} \mid \mathbf{x})$$

$\mathbf{x}$ Observed data

2 heads out of 10 coin flips
(Binomial Dist)

$\mathbf{z}$ Latent representation
(hidden code)

The coin fairness
z = 0.5, fair coin
z = 1, always head
z = 0, always tail

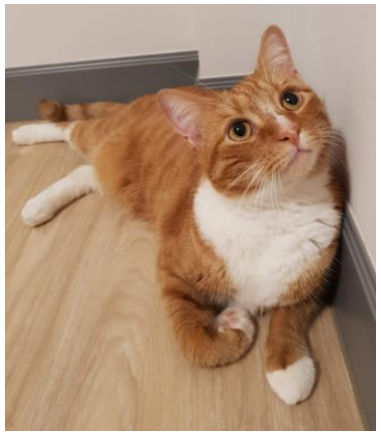


Bayesian Review

One important inference problem : Posterior inference

$$p(\mathbf{z} \mid \mathbf{x})$$

$\mathbf{x}$ Observed data



$\mathbf{z}$ Latent representation (hidden code)

Class label cat, color, can fly?, ...

$p(\mathbf{z} \mid \mathbf{x})$

Tells you **not only** what animal is the most likely, **but also** how “orange cat” it is, how “white dog” it is, etc -- the probability of all hidden code

Variational Bayesian Review

One important inference problem : Posterior inference

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{x})} = \frac{p(\mathbf{z}, \mathbf{x})}{\int p(\mathbf{x}, \mathbf{z}) d\mathbf{z}}$$

Hard to compute because z can be high-dimensional

E.g. $\mathbf{z} = \{z_1, z_2, z_3, \ldots\}$

z_1 : class label {cat, dog, rabbit, rat}

z_2 : color, a real number

z_3 : ability to fly {yes, no}

$$\int d\mathbf{z} = \sum_{z_1} \int_{z_2} \sum_{z_3} \ldots dz_2$$

Bayesian Review

One important inference problem : Posterior inference

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{x})} = \frac{p(\mathbf{z}, \mathbf{x})}{\int p(\mathbf{x}, \mathbf{z}) d\mathbf{z}}$$

A solution:

Let's approximate the true posterior $p(\mathbf{z} \mid \mathbf{x})$

with an approximate, easier-to-work-with $q(\mathbf{z} \mid \mathbf{x})$

and solve this as an optimization problem.

Approximate True Posterior

Let's approximate the true posterior $p(\mathbf{z} \mid \mathbf{x})$ with an approximate, easier-to-work-with $q(\mathbf{z} \mid \mathbf{x})$ and solve this as an optimization problem.

Approximate True Posterior

Let's approximate the true posterior $p(\mathbf{z} \mid \mathbf{x})$ with an approximate, easier-to-work-with $q(\mathbf{z} \mid \mathbf{x})$ and solve this as an optimization problem.

Best $q^*(\mathbf{z} \mid \mathbf{x})$ is the one closest to $p(\mathbf{z} \mid \mathbf{x})$

A solution: $q^*(\mathbf{z} \mid \mathbf{x}) = \underset{q}{\operatorname{argmin}} \operatorname{KL}\left[q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x})\right]$

- But why not $\operatorname{KL}\left[p(\mathbf{z} \mid \mathbf{x}) \| q(\mathbf{z} \mid \mathbf{x})\right]$ or $D\left(p(\mathbf{z} \mid \mathbf{x}), q(\mathbf{z} \mid \mathbf{x})\right)$?
- How is this related to autoencoder?

Approximate True Posterior

Let's approximate the true posterior $p(\mathbf{z} \mid \mathbf{x})$

with an approximate, easier-to-work-with $q(\mathbf{z} \mid \mathbf{x})$

and solve this as an optimization problem.

Because doing this (i.e., using KL (q || p)):

$$q^*(\mathbf{z} \mid \mathbf{x}) = \underset{q}{\operatorname{argmin}} \, \mathrm{KL}\left[q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x})\right]$$

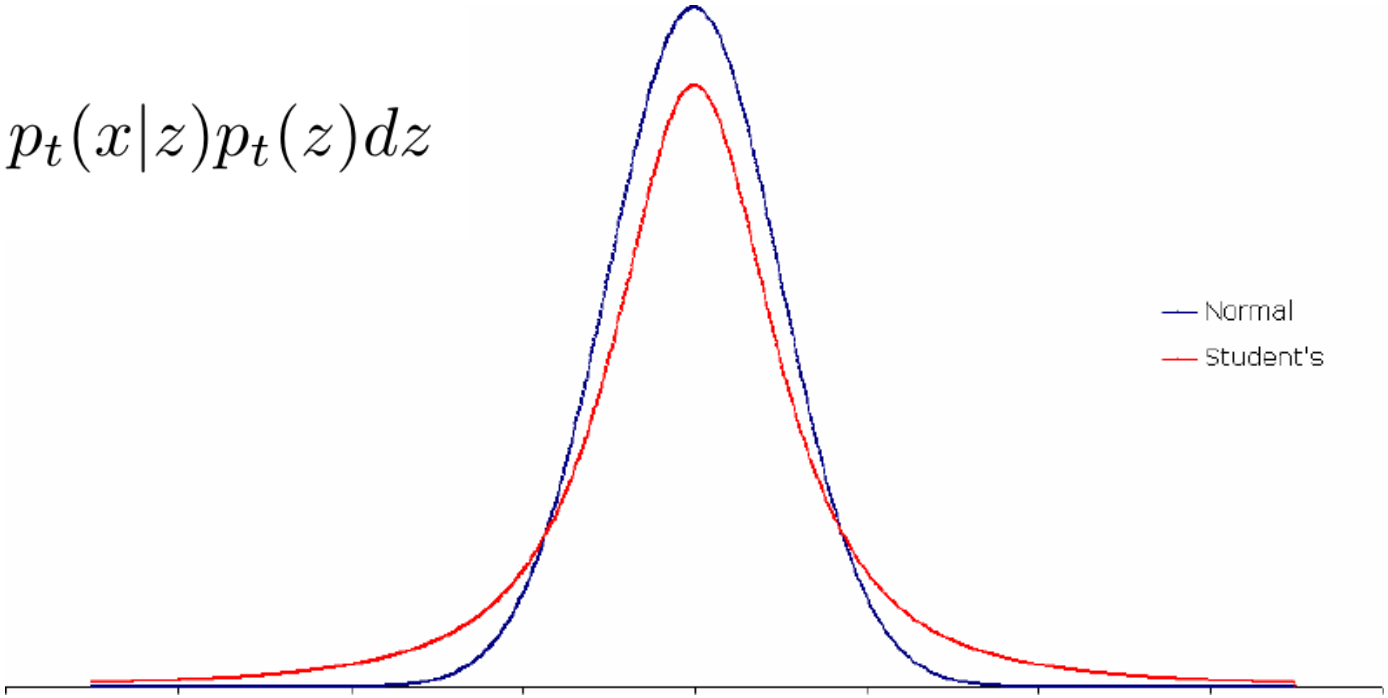
Corresponds to maximizing the evidence lower bound (ELBO)!

Digression: Model evidence

Model evidence (or marginal likelihood) is simply $p(\mathbf{x})$

$$p_g(x) = \int p_g(x|z)p_g(z)dz$$

$$p_t(x) = \int p_t(x|z)p_t(z)dz$$



EM algorithm

- Goal of EM : $\log \sum_k p(x, k;\theta) >= \sum_k Q(k) \log (p(x, k;\theta)/Q(k))$
- Maximize the ELBO instead
- Initialize Θ
- Repeat till convergence
  - Expectation step (E-step) : estimate the conditional expectation $Q(k) = p(k|x;\theta)$ using the current θ.
  - Maximization step (M-step) : Estimate new Θ given by maximizing the ELBO given current Q(k)

P(X) – evidence

ELBO

$E_{k\sim Q}\left[\log \frac{P(x,k)}{Q(k)}\right]$

P(X|Y) – likelihood

P(Y|X) – posterior

Proof

This KL is hard to compute

$$q^*(\mathbf{z} \mid \mathbf{x}) = \underset{q}{\operatorname{argmin}} \operatorname{KL}\left[q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x})\right]$$

$$\operatorname{KL}\left[q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x})\right] = \mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x})] - \mathbb{E}_q[\log p(\mathbf{z} \mid \mathbf{x})]$$

$q(z|x)$

(by definition)

$$= \mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x})] - \mathbb{E}_q[\log p(\mathbf{z}, \mathbf{x})] + \log p(\mathbf{x})$$

(expanding P(z | x), E_q[log p(x)] = log p(x))

$$= \mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{z}, \mathbf{x})] + \log p(\mathbf{x})$$

Constant w.r.t q. So, we can safely kill it

$$\min_q \operatorname{KL}\left[q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x})\right] \iff \min_q \mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{z}, \mathbf{x})]$$

Questions: 1. What's this new term? 2. Is it easier to evaluate?

Proof

$$\min_q \mathrm{KL}\left[q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x})\right] \iff \min_q \mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{z}, \mathbf{x})]$$

Let's interpret this new term:

From previous slide, we have:

$$\log p(\mathbf{x}) = \mathrm{KL}\left[q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x})\right] - \mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{z}, \mathbf{x})]$$

We know KL >= 0, always!

$$\log p(\mathbf{x}) \geq \boxed{-\mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{z}, \mathbf{x})]}$$

(Lower bound of model evidence)

$$\min_q \mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{z}, \mathbf{x})] \iff \max_q -\mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{z}, \mathbf{x})]$$
$$\iff \max_q \mathrm{ELBO}$$

But how to compute this ELBO term then?

Maximizing ELBO

$$q^*(\mathbf{z} \mid \mathbf{x}) = \operatorname*{argmin}_q \mathrm{KL}\left[q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x})\right]$$

$$\max_q \mathrm{ELBO} = \max_q -\mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{z}, \mathbf{x})]$$

$$\iff \min_q \mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{z}, \mathbf{x})]$$

$$\iff \min_q \mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x})] - \mathbb{E}_q[\log p(\mathbf{x} \mid \mathbf{z})] - \mathbb{E}_q[\log p(\mathbf{z})]$$

$$\iff \min_q \mathrm{KL}[q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})] - \mathbb{E}_q[\log p(\mathbf{x} \mid \mathbf{z})]$$

We're free to pick

Standard AE

Encoder → (2, 9) → Decoder →

Variational AE

$q(z|x)$

Encoder → Means, Std.

Sample → (2.1, 9.2)

$q(\mathbf{z} \mid \mathbf{x})$

$p(x|z)$

Decoder →

*Output as a sample from P(x | z).

$$\min_q \mathrm{KL}[q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})] - \mathbb{E}_q[\log p(\mathbf{x} \mid \mathbf{z})]$$

↑ sample from $q(z|x)$

Encoder

Means (2, 9)

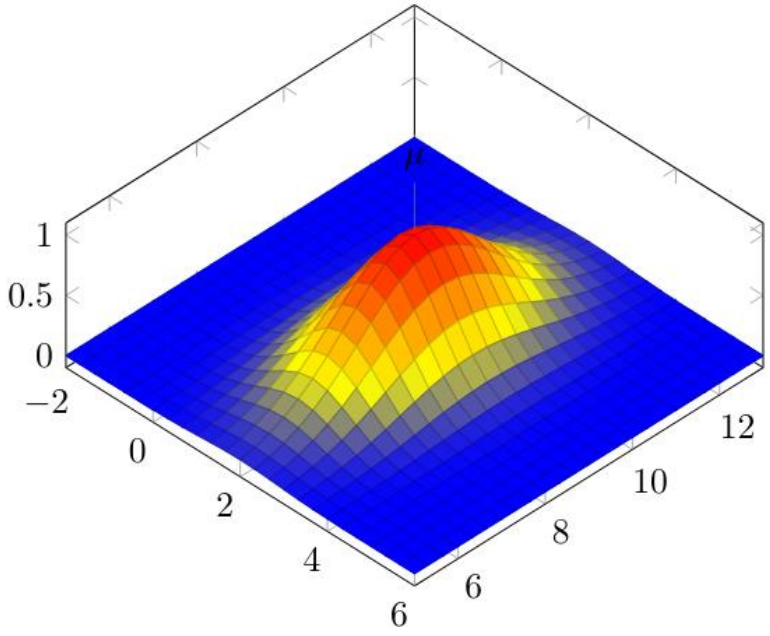
Std.

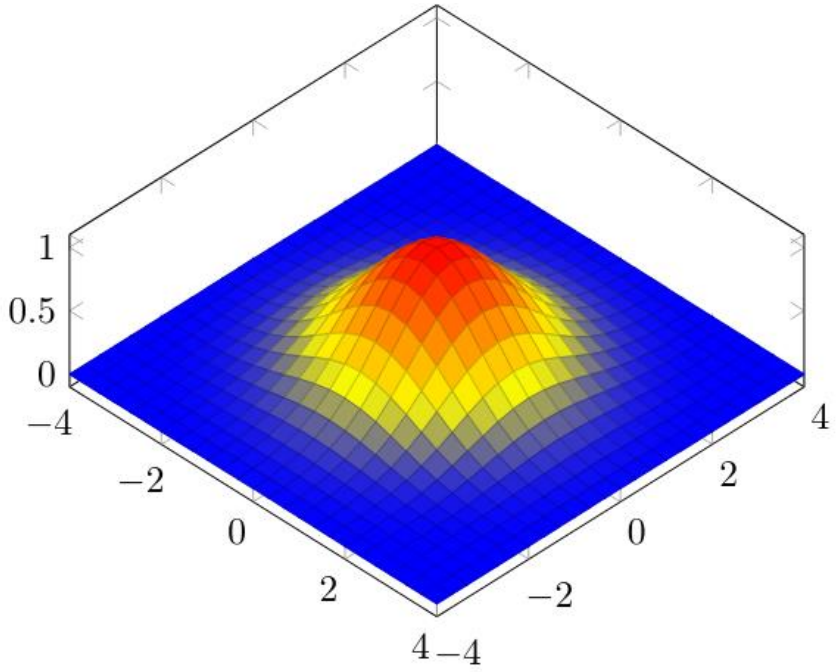
Sample

(2.1, 9.2)

$q(\mathbf{z} \mid \mathbf{x})$

Decoder





$q(\mathbf{z} \mid \mathbf{x})$ is forced to be close to $p(\mathbf{z}) = \mathcal{N}(0, I)$

$$\min_q \mathrm{KL}[q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})] - \mathbb{E}_q[\log p(\mathbf{x} \mid \mathbf{z})]$$

What is this second term?

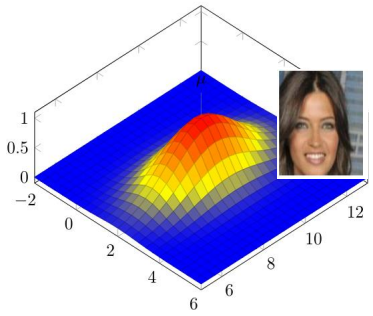
Second Term: Data Log Likelihood

$$\min_q \mathrm{KL}[q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})] - \mathbb{E}_q[\log p(\mathbf{x} \mid \mathbf{z})]$$

If we also model $p(\mathbf{x} \mid \mathbf{z})$ as a Gaussian distribution, where the mean = u is the output of the decoder:

$$p(\mathbf{x} \mid \mathbf{z}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-u)^2}{2\sigma^2}}$$

(Let's consider a single pixel case
u is the output from the decoder
x is the ground-truth pixel)



$$\propto \exp\left(-(x-u)^2\right)$$

$$-\log[\exp\left(-(x-u)^2\right)] = (x-u)^2$$

(with constants hidden)

Becomes the standard L2 error

Yet another reason why assuming Gaussian is nice

Encoder → (2, 9) → Decoder →

X X'

Standard AE Loss $\|X - X'\|^2$

Encoder → Means, Std. → Sample → (2.1, 9.2) → Decoder →

X X'

Regularization of L2 loss

Variational AE Loss $\approx \|X - X'\|^2 + \mathrm{KL}[q(\mathbf{z} \mid \mathbf{x}) \| \mathcal{N}(0, I)]$

Precisely, $\min_{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})] - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})]$

Implementation of KL term

$$\min_{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})] - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})]$$

approximate via minibatch

- You can appromate the KL term by computing in the minibatch
- If you assume both q and p are Gaussians (vanilla VAE) the equation simplifies
  - https://mr-easy.github.io/2020-04-16-kl-divergence-between-2-gaussian-distributions/

When q is $\mathcal{N}(0,\ I)$, we get,

Note the notation difference (p and q)

$$D_{KL}(p||q) = \frac{1}{2}\left[\boldsymbol{\mu_p^T \mu_p} + tr\left\{\Sigma_p\right\} - k - \log|\Sigma_p|\right]$$

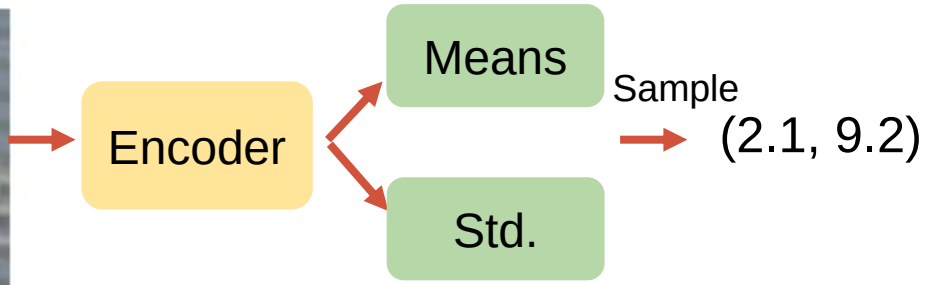
Implementation of KL term

$$\min_{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})] - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})]$$

approximate via minibatch

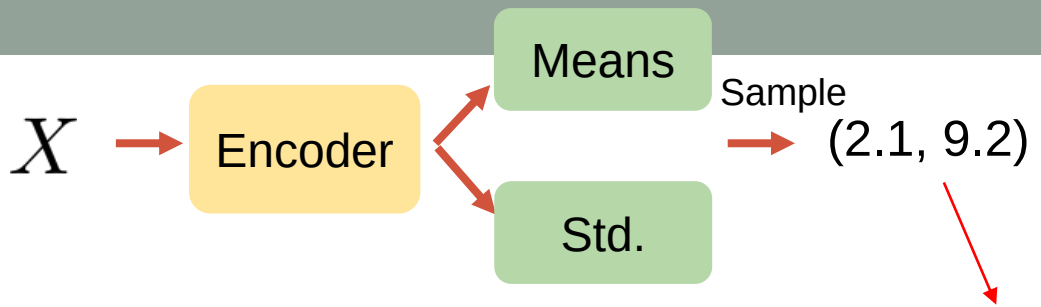
- General form, we have to approximate KL via sampling
- KL[q|p] = $E_{z\sim q(z|x)}$[log q(z|x) – log p(z)]
- For image x_i in minibatch
  - Compute q(z|x) by passing through encoder network
  - Sample z_i ~ q(z|x)
  - Acummulate [log $q(z_i|x)$ - log $p(z_i)$]

X



https://towardsdatascience.com/variational-autoencoder-demystified-with-pytorch-implementation-3a06bee395ed example code

Last Problem



$$\min_{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})] - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})]$$

Estimate in minibatch

When we need to compute the gradient with respect to the network parameters (I.e., ϕ, θ) for SGD, we're struck because we can't backprop through the sampling for computing $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}$

Example:

$$z = y^2 \qquad z \sim \mathcal{N}(y, 1) \qquad z \sim q_\phi(z \mid x)$$

$$\frac{dz}{dy} = 2y \qquad \frac{dz}{dy} = ?? \qquad \frac{dz}{d\phi} = ??$$

Reparameterization Trick

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

$$\frac{\partial z}{\partial \mu} = ??$$

$$\frac{\partial z}{\partial \sigma} = ??$$

$$z = \epsilon_i \times \sigma + \mu$$

$$\epsilon \sim \mathcal{N}(0, 1)$$

$$\frac{\partial z}{\partial \mu} = 1$$

$$\frac{\partial z}{\partial \sigma} = \epsilon_i$$

$$\min_{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})] - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})]$$

$$\iff \min_{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})] - \mathbb{E}_{p(\epsilon)}[\log p_\theta(\mathbf{x} \mid \hat{\mathbf{z}})]$$

$$\hat{\mathbf{z}} = \epsilon \cdot \sigma + \mu, \quad p(\epsilon) \sim \mathcal{N}(0, I)$$

X

Encoder → (2, 9) → Decoder

X'

Standard AE Loss $\|X - X'\|^2$

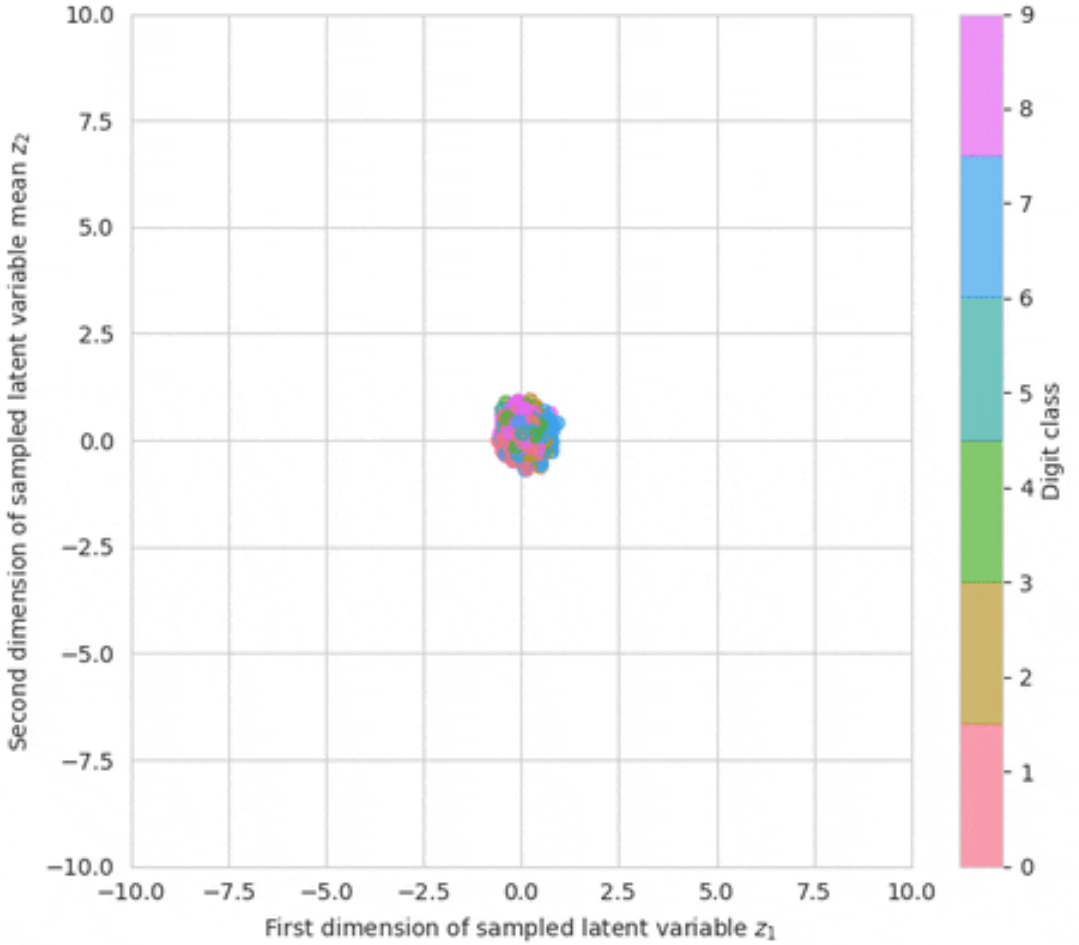
X

Encoder → Means / Std. → Sample → (2.1, 9.2) → Decoder

X'

Variational AE Loss $\approx \|X - X'\|^2 + \mathrm{KL}[q(\mathbf{z} \mid \mathbf{x}) \| \mathcal{N}(0, I)]$

More info: https://arxiv.org/abs/1606.05908



Notes: when we compresses data, meaningful “codes” tend to emerge.
The P(z)~N(0,1) clusters the data and allow for easy interpolation between images

https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

VAE today

- Research focuses on disentangle meaning into different dimension of z (z_1 = hair color, z_2 = glasses, z_3 = age)
  - Controllable generation
- Improve quality of images

Disentangling Representation: β-VAE

We want compact meaningful representation.

But also disentangled representation

$$\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}, \beta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta\, D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

https://openreview.net/forum?id=Sy2fzU9gl

Skin color

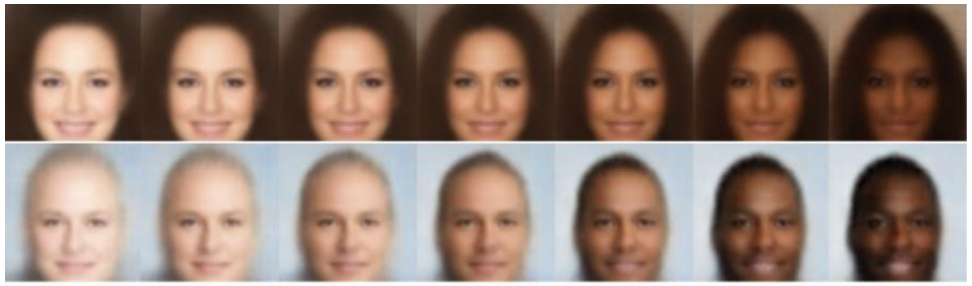


Image Saturation

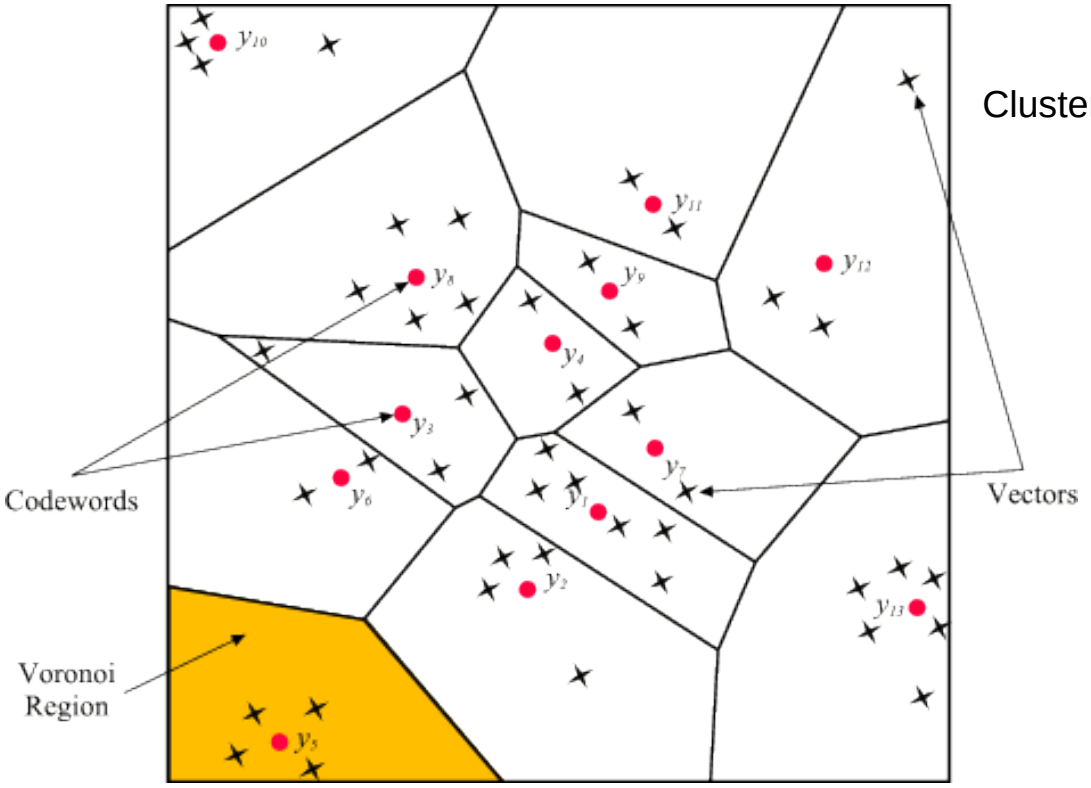
Head Angle

Age / Gender

VQ-VAE

https://arxiv.org/abs/1711.00937

- Learn discrete z by using vector quantization

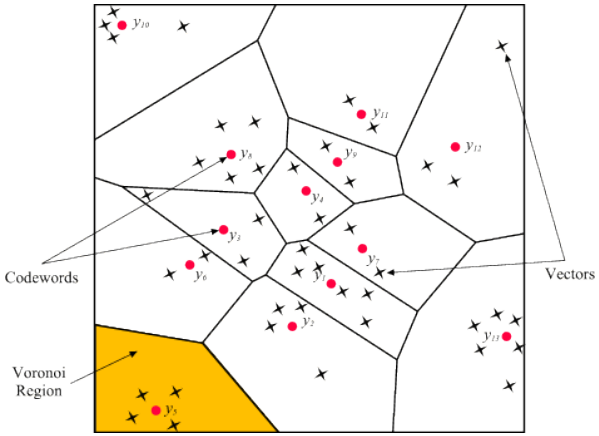


Cluster vectors into discrete indexes

http://www.mqasem.net/vectorquantization/vq.html

VQ-VAE

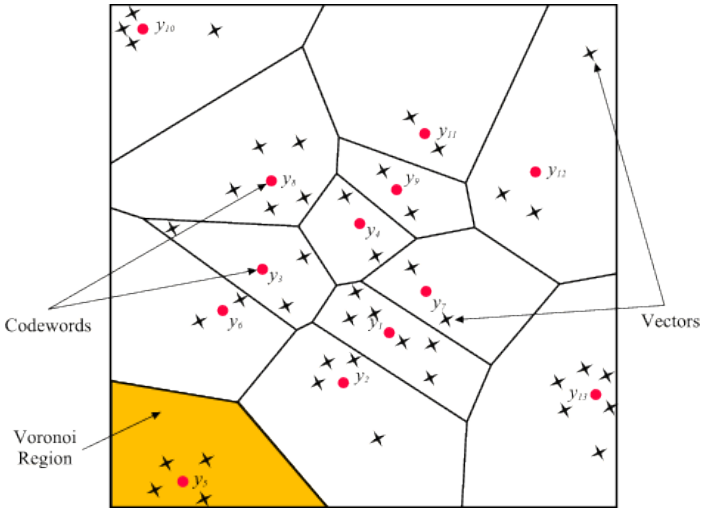
- An image can have multiple codes
- Encoder produces multiple vectors -> multiple codes
  - Ex. Image encodes into grid of NxN vector
- 32x32 grid vectors with 512 codes each
  - = 512^{32x32} possibilities



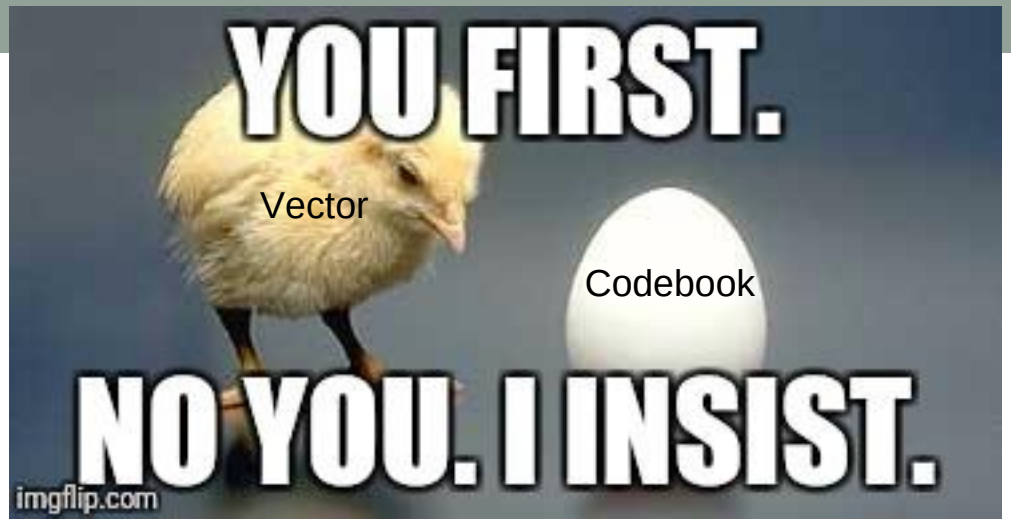
5 (girl), 10 (fire), 3 (road)

Learning the codebook

- Ideally, we want to have the encoded vectors be closed to the codewords
- And we want codebooks are good representation of encoded vectors



Overall loss



1.Codebook can reconstruct

2.Moves codebook to vector

3.Moves vector to codebook

Encoded vector

Code book

Hyperparameter to balance the loss

$$L = \log p(x|z_q(x)) + \|\text{sg}[z_e(x)] - e\|_2^2 + \beta\|z_e(x) - \text{sg}[e]\|_2^2,$$

Normal reconstruction loss

Stop gradient. No gradient flow through this term
Pytorch .detach() see homework ☺

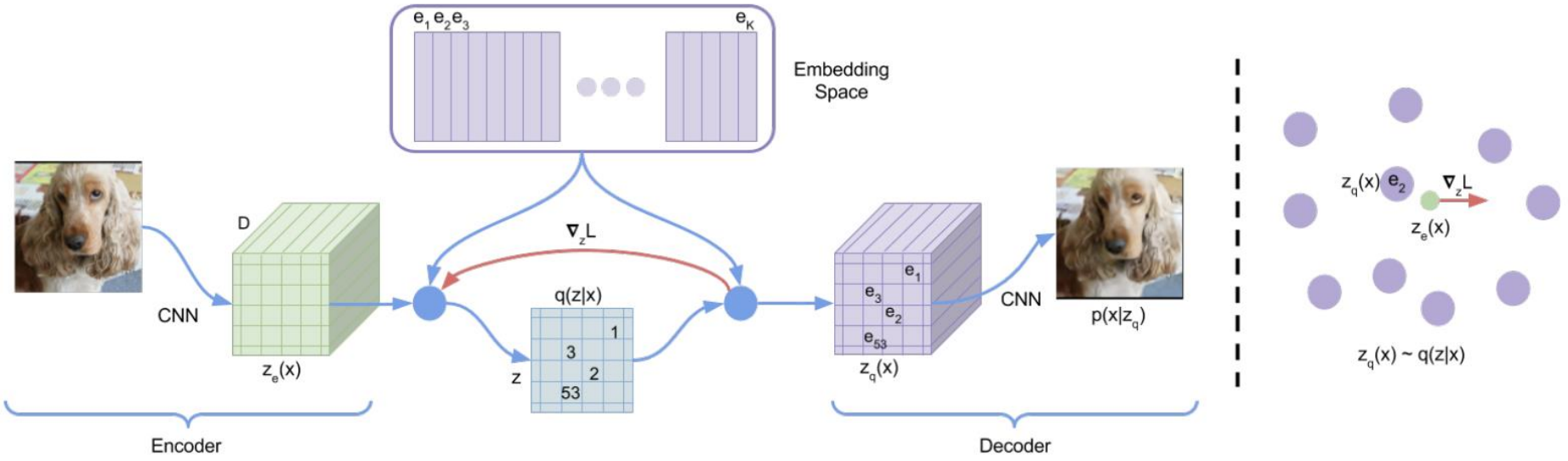


Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(x)$ is mapped to the nearest point e_2. The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

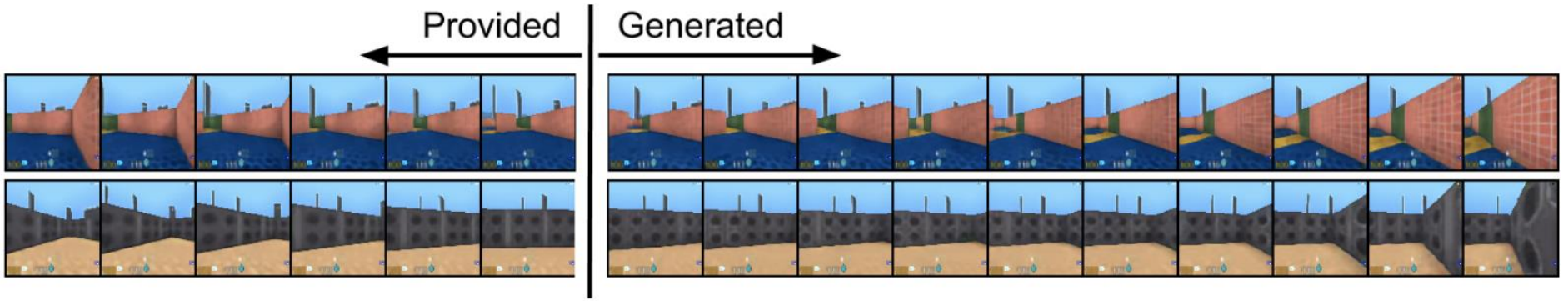
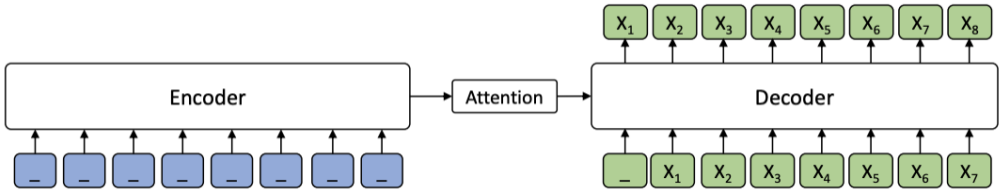


Figure 7: First 6 frames are provided to the model, following frames are generated conditioned on an action. Top: repeated action "move forward", bottom: repeated action "move right".

Dall E



Transformer (GPT3) + VQ VAE

More explanation here
https://ml.berkeley.edu/blog/posts/dalle2/

(a) a tapir made of accordion. a tapir with the texture of an accordion.

(b) an illustration of a baby hedgehog in a christmas sweater walking a dog

(c) a neon sign that reads "backprop". a neon sign that reads "backprop". backprop neon sign

(d) the exact same cat on the top as a sketch on the bottom

Figure 2. With varying degrees of reliability, our model appears to be able to combine distinct concepts in plausible ways, create anthropomorphized versions of animals, render text, and perform some types of image-to-image translation.

https://openai.com/blog/dall-e/
https://arxiv.org/pdf/2102.12092.pdf

Perceptual loss

Typical Autoencoder uses per pixel loss (for example L2)

Might not capture the important content

https://arxiv.org/abs/1603.08155

Perceptual loss

Measure semantic differences

Use the feature extracted from a pre-trained object classifier as a representation for semantic

Use the L2 between features as an additional loss

$$\ell_{feat}^{\phi,j}(\hat{y}, y) = \frac{1}{C_j H_j W_j} \|\phi_j(\hat{y}) - \phi_j(y)\|_2^2$$

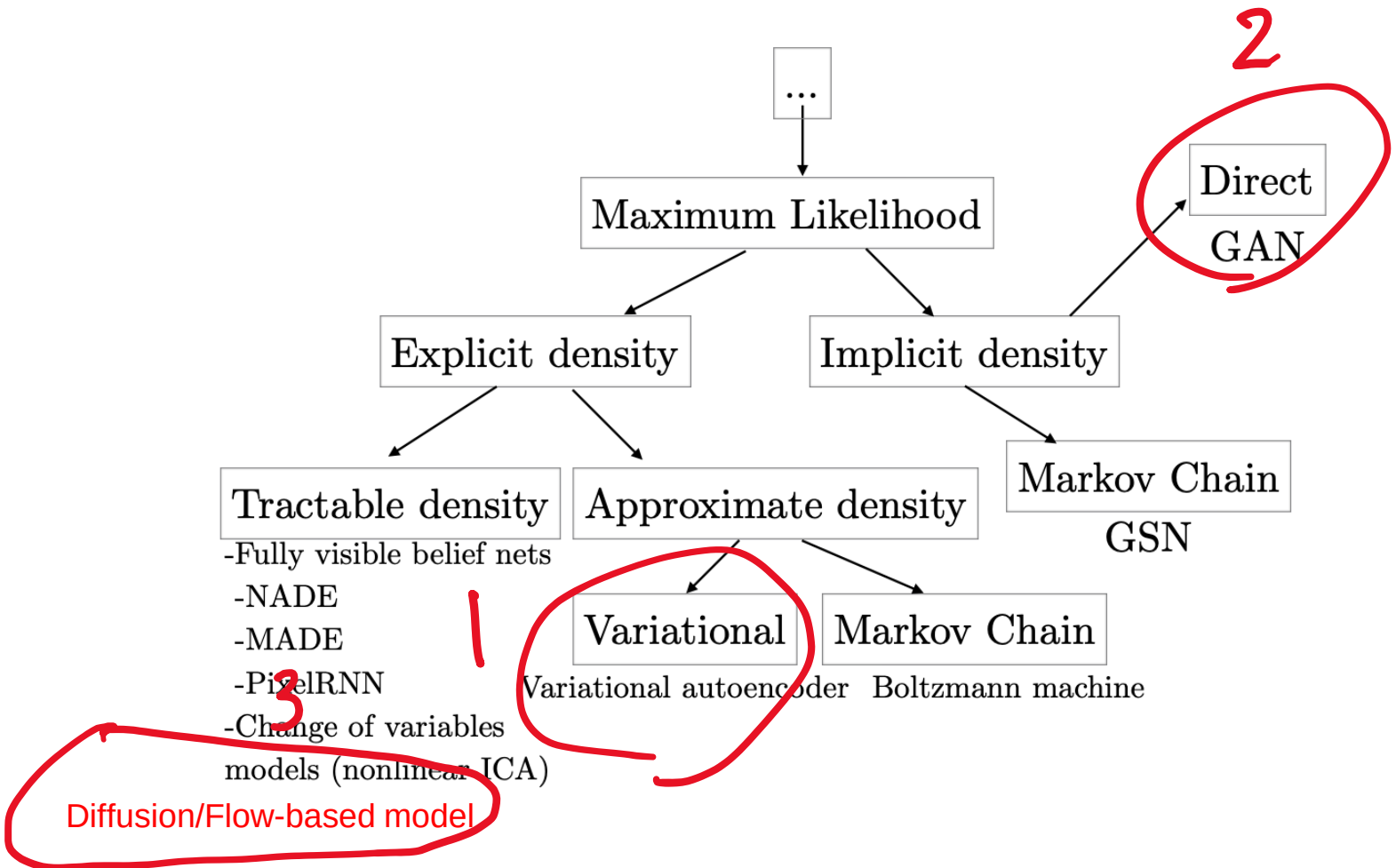
https://arxiv.org/abs/1603.08155

Deep generative models

- Autoencoders
  - Variational autoencoders
- GAN
- Diffusion/Flow

Deep Generative Models

- A deep learning model that can be used to generate X



Generating Images

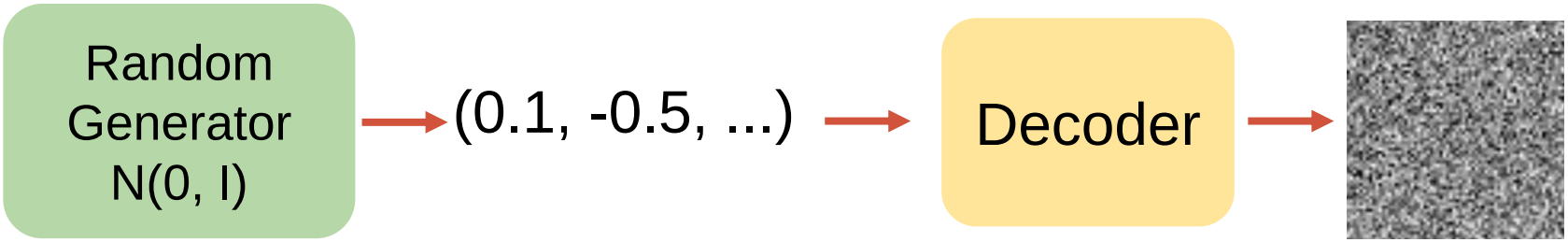
VAE can be used to generate images by feeding numbers into the decoder.

Are there other ways?

Given a collection of, say, faces, how can we generate new faces?

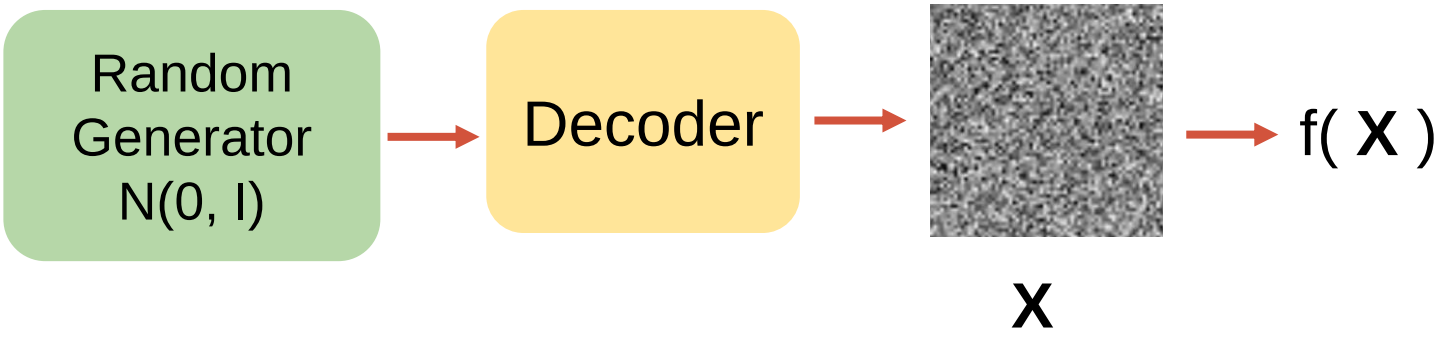
Maybe let’s start with a decoder, because we need something that outputs an image anyway.

Generating Realistic Faces



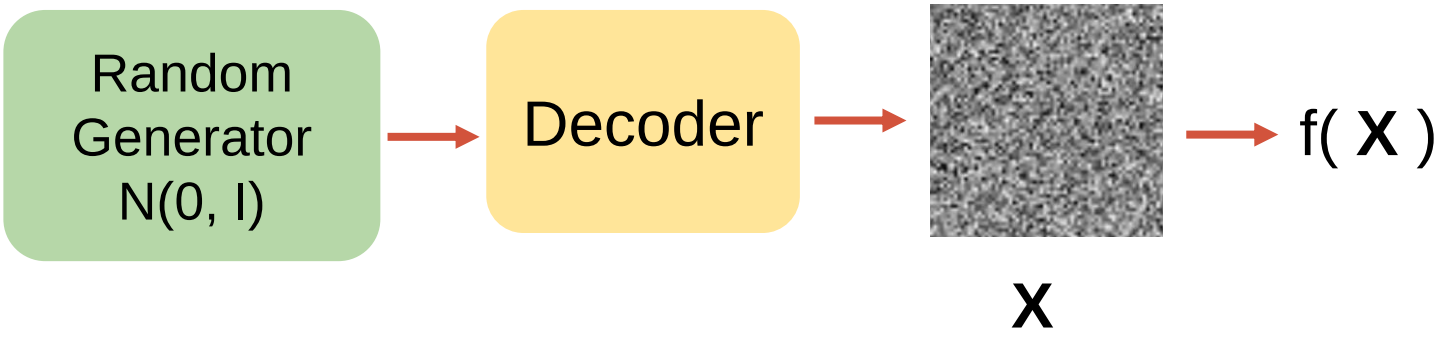
What kind of loss should we use?

Generating Realistic Faces



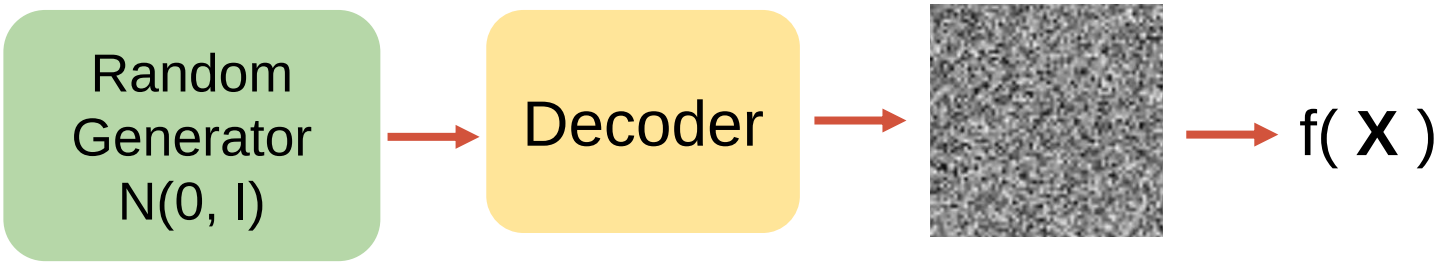
We'll be done if we can come up with a smooth function f, that takes in the output image x and outputs [0, 1] where:

Generating Realistic Faces



Given this f, we can compute its gradient with respect to the decoder parameters and use SGD to train. Done!

Generating Realistic Faces

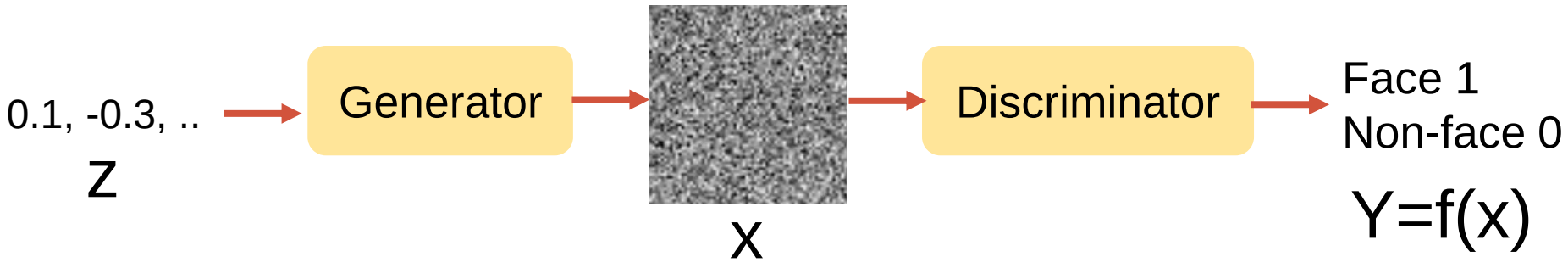


Solution: We will use another network as f. This network will predict the probability of x being a face.

Training this f is a simple supervised learning!

All face photos -> positive examples
All non-face photos -> negative examples

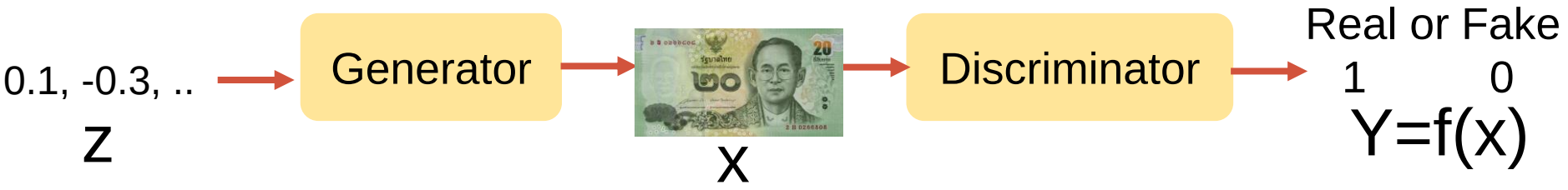
Generating Realistic Faces



Generative Adversarial Networks (GAN)

- Consider a money counterfeiter
  - He wants to make fake money that looks real
  - There's a police that tries to differentiate fake and real money.
- The counterfeiter is the adversary and is generating fake inputs. – Generator network
- The police tries to discriminate between fake and real inputs. – Discriminator network

Generative Adversarial Networks (GAN) – minimax loss



- Generator (Money Faker):
  - Maximize Y

$$\min_G \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z}))]$$

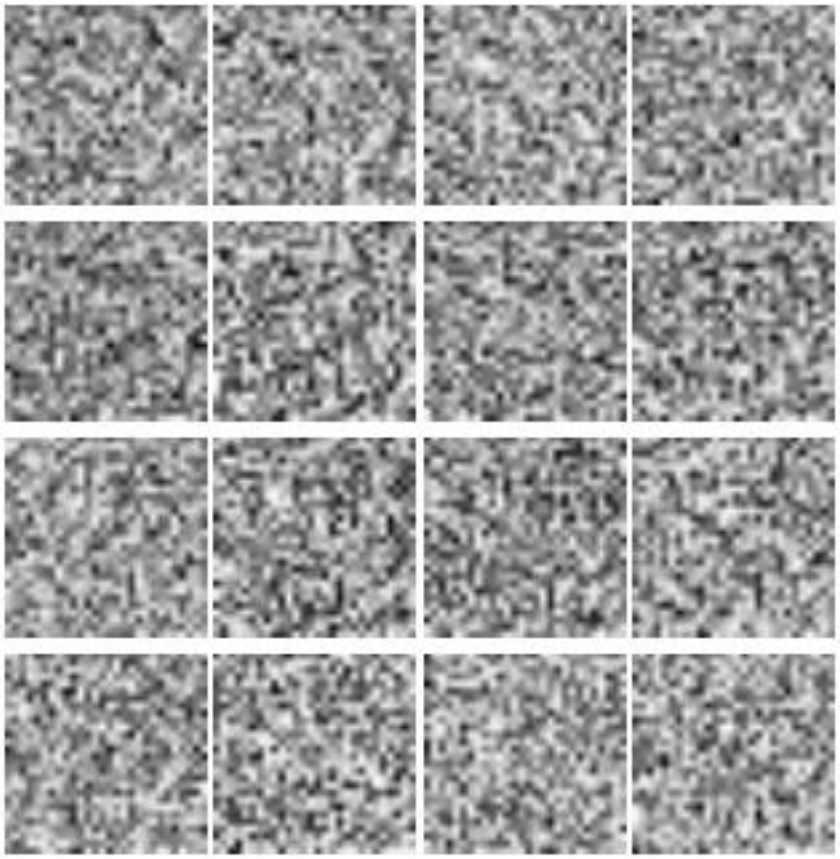
- Discriminator (Police):
  - For real images => Maximize Y
  - For generated images from the faker => Minimize Y

$$\max_D \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z}))] + \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log(D(\mathbf{x}))]$$

This comes from binary cross entropy loss

GAN example

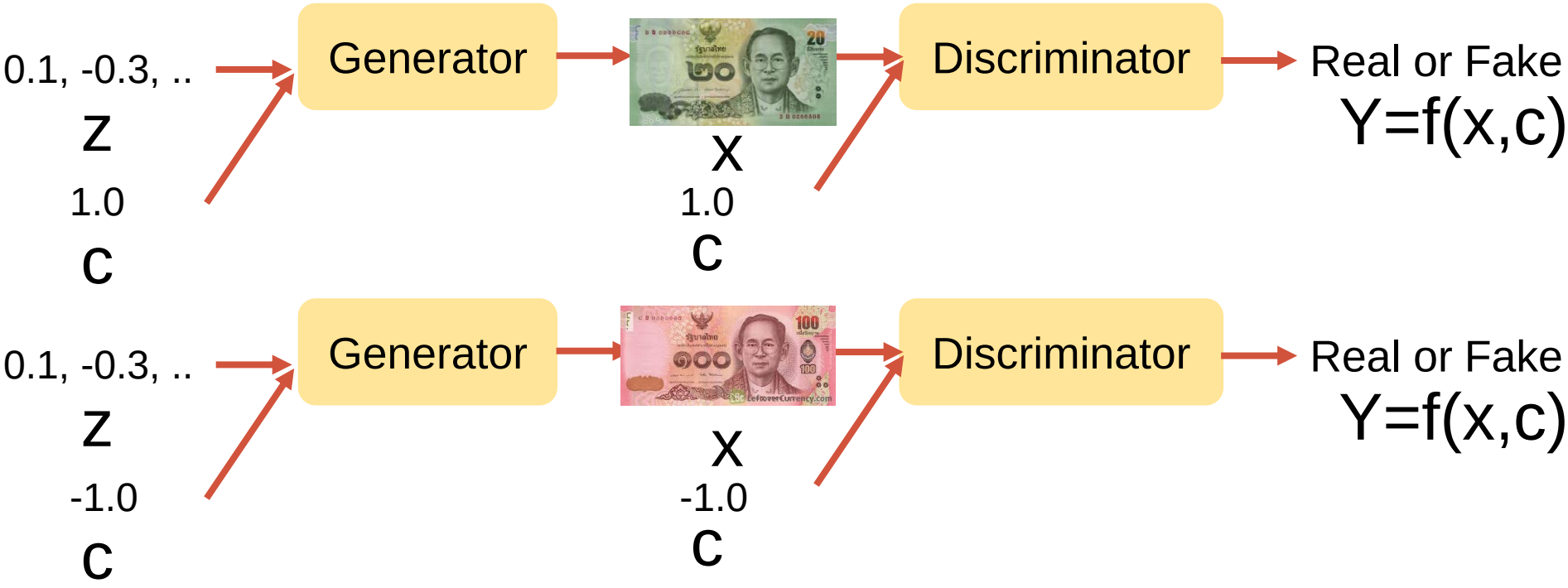
Generator output starts from random noise and gets better as we train.



https://wiseodd.github.io/techblog/2016/09/17/gan-tensorflow/

Conditional GAN

- Can conditioned GAN to generate a certain trait/class



Just concatenate C to the noise and the discriminator input
Discriminator also checks if it's the correct class

Mode collapse

Model only learn a couple types (modes) of inputs

https://arxiv.org/pdf/1701.07875.pdf

Ian Goodfellow
@goodfellow_ian

4.5 years of GAN progress on face generation.
arxiv.org/abs/1406.2661 arxiv.org/abs/1511.06434
arxiv.org/abs/1606.07536 arxiv.org/abs/1710.10196
arxiv.org/abs/1812.04948

2014

2015

2016

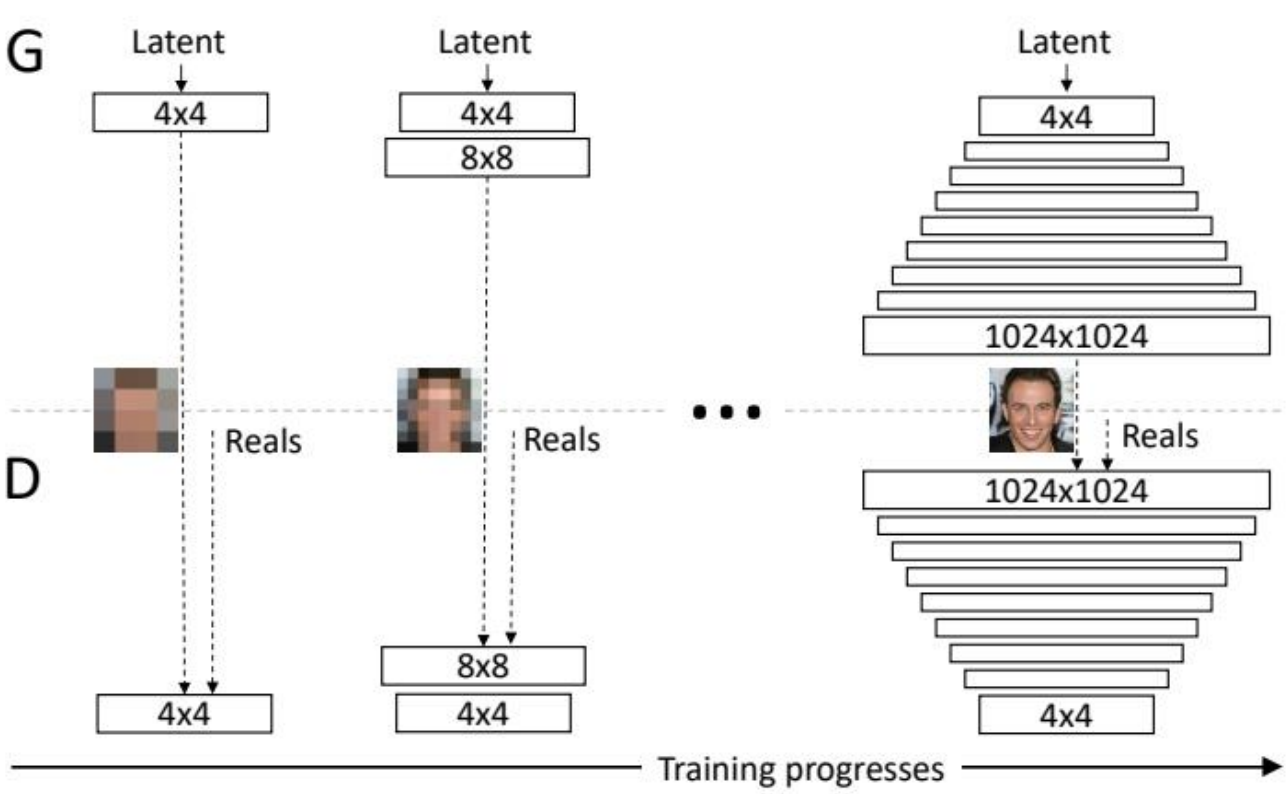
2017
Progressive GAN

2018
Style GAN

More realistic, higher res, more controllable

Progressive GANS

Newer GANs are usually trained from low res to high res

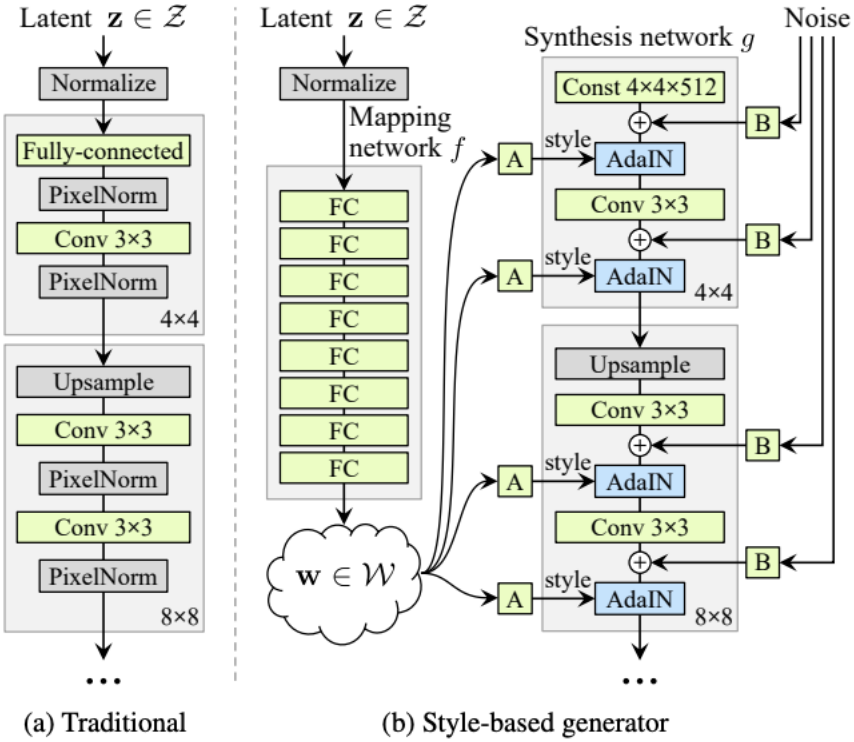


Progressive Growing of GANs for Improved Quality, Stability, and Variation [Karras et al. 2017]

https://drive.google.com/drive/folders/1I9wsMzDkxF85pZ39RTVjiP83BLIEXUBQ

Style-GAN (V1)

- Based on progressive GAN
- Inspired from style transfer network (no actual style transfer here)
- Embedded “Styles” from z
- Use styles to control the generator



$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}, \qquad (1)$$

Instance norm

Style value

V1 https://arxiv.org/abs/1812.04948 V2 https://arxiv.org/abs/1912.04958 V3 https://arxiv.org/abs/2106.12423

GAN output in paper

Your GAN output

https://medium.com/nybles/understanding-machine-learning-through-memes-4580b67527bf

Extra reading
https://machinelearningmastery.com/generative-adversarial-network-loss-functions/

GANs Loss Formulations

$$\max_D \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z}))] + \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log(D(\mathbf{x}))]$$

$$\min_G \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z}))]$$

Discriminator

Generator

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\text{D}}^{\text{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\text{G}}^{\text{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\text{D}}^{\text{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\text{G}}^{\text{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
* WGAN	$\mathcal{L}_{\text{D}}^{\text{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\text{G}}^{\text{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\text{D}}^{\text{WGANGP}} = \mathcal{L}_{\text{D}}^{\text{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\|\nabla D(\alpha x + (1 - \alpha\hat{x})\|\|_2 - 1)^2]$	$\mathcal{L}_{\text{G}}^{\text{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
*LS GAN	$\mathcal{L}_{\text{D}}^{\text{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x) - 1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\text{G}}^{\text{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x} - 1))^2]$
*DRAGAN	$\mathcal{L}_{\text{D}}^{\text{DRAGAN}} = \mathcal{L}_{\text{D}}^{\text{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\|\nabla D(\hat{x})\|\|_2 - 1)^2]$	$\mathcal{L}_{\text{G}}^{\text{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\text{D}}^{\text{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|\|x - \text{AE}(x)\|\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\|\hat{x} - \text{AE}(\hat{x})\|\|_1]$	$\mathcal{L}_{\text{G}}^{\text{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\|\hat{x} - \text{AE}(\hat{x})\|\|_1]$

Are GANs Created Equal? A Large-Scale Study [Lucic et al. 2018]

Wasserstein GAN (WGAN)

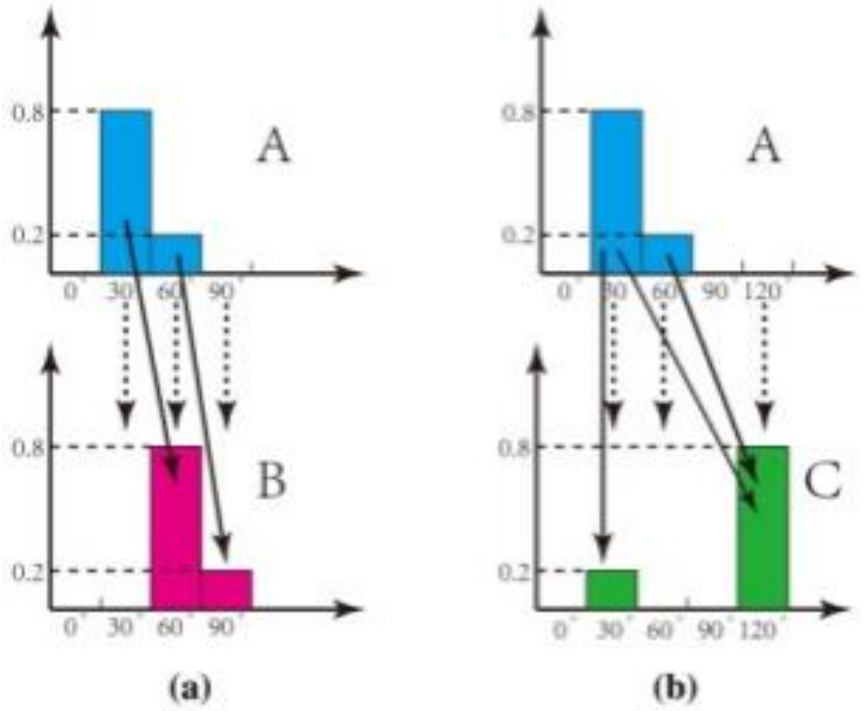
Wasserstein distance? (Earth mover distance)

Energy required to move mass to make two distributions look the same

WGAN

approximates and minimizes this distance

(Note vanilla GAN minimizes Jensen-Shannon divergence)



https://medium.com/@adriensieg/text-similarities-da019229c894

https://arxiv.org/abs/1701.07875
https://arxiv.org/abs/1704.00028

WGAN

Discriminator as a critic (no fake/real sigmoid) but output a score (linear activation at output layer)

WD has better gradient and convergence

	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$	$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$
WGAN	$\nabla_w \frac{1}{m} \sum_{i=1}^{m} \left[f\left(x^{(i)}\right) - f\left(G\left(z^{(i)}\right)\right) \right]$	$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f\left(G\left(z^{(i)}\right)\right)$

WGAN

Approximating EMD requires the critic model to be a k-Lipschitz function

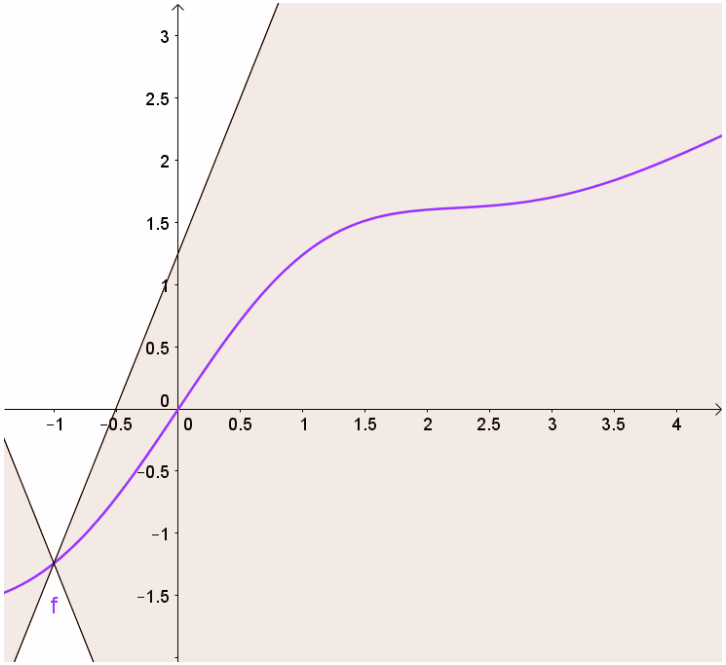
k-Lipschitz?

Bounded in slope of k

$|f(a) - f(b)| < k|a - b|$

Example

$f(x) = 5x$ is 5-Lipschitz



https://en.wikipedia.org/wiki/Lipschitz_continuity

WGAN to WGAN-GP

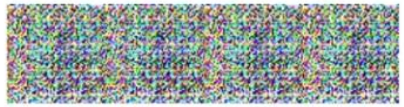
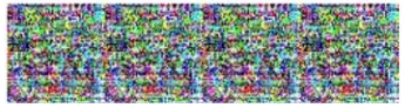
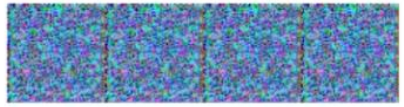
To make k-Lipschitz

WGAN caps the weights of all layers to 1

WGAN-GP improves and add Gradient Penalty to reduce the weights instead

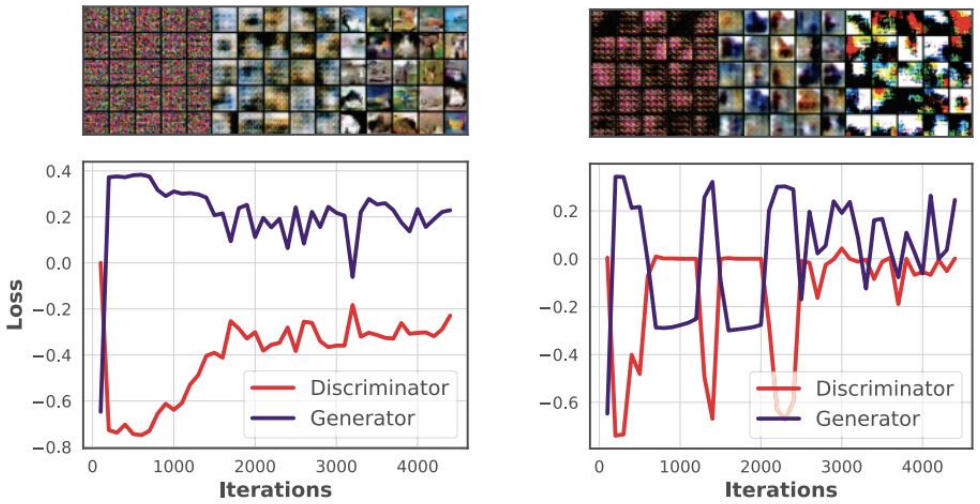
A differentiable function f is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere.

$$L = \underbrace{\underset{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}{\mathbb{E}}\left[D(\tilde{\boldsymbol{x}})\right] - \underset{\boldsymbol{x}\sim\mathbb{P}_r}{\mathbb{E}}\left[D(\boldsymbol{x})\right]}_{\text{Original critic loss}} + \underbrace{\lambda \underset{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}{\mathbb{E}}\left[\left(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1\right)^2\right]}_{\text{Our gradient penalty}}.$$

DCGAN	LSGAN	WGAN (clipping)	WGAN-GP (ours)
Baseline (G: DCGAN, D: DCGAN)			
G: No BN and a constant number of filters, D: DCGAN			
G: 4-layer 512-dim ReLU MLP, D: DCGAN			
No normalization in either G or D			
Gated multiplicative nonlinearities everywhere in G and D			
tanh nonlinearities everywhere in G and D			
101-layer ResNet G and D			

GAN without competition

- Minimax nature of GAN makes it hard to train
  - Optimize the duality gap instead



(a) Convergence (b) Divergence

Figure 1. Loss curves throughout the training progress of WGAN on the CIFAR-10 dataset.

$$
\begin{aligned}
DG^{\lambda}(\theta_d, \theta_g) &= V_{D_w}(\theta_g) - V^{\lambda}_{G_w}(\theta_d), \ \textit{where} \\
V_{D_w}(\theta_g) &= \max_{\theta'_d \in \Theta_D} V(D_{\theta'_d}, G_{\theta_g}) \\
V^{\lambda}_{G_w}(\theta_d) &= \min_{\theta'_g \in \Theta_G} V^{\lambda}(D_{\theta_d}, G_{\theta'_g})
\end{aligned}
$$

Estiamte V_D by optimizing discriminator to estimate worst case discriminator
Estiamte V_G by optimizing generator to estimiate worst case generator
Optimize the gap

Generative Minimization Networks: Training GANs Without Competition
https://arxiv.org/pdf/2103.12685.pdf
On Characterizing GAN Convergence Through Proximal Duality Gap
http://proceedings.mlr.press/v139/sidheekh21a/sidheekh21a.pdf

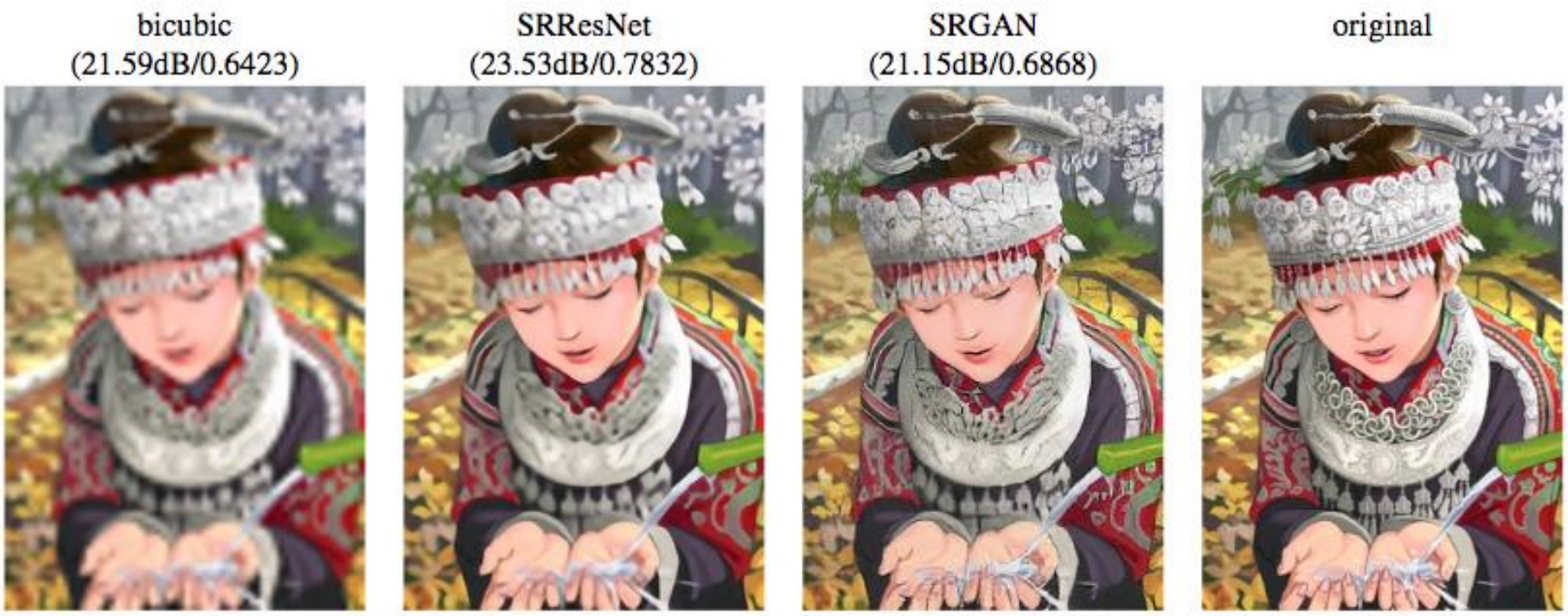
Other uses

- StackGAN: text to photo

StackGAN: Text to Photo-realistic Image Synthesis with Stacked GANs [Zhang et al. 2017]

SRGAN

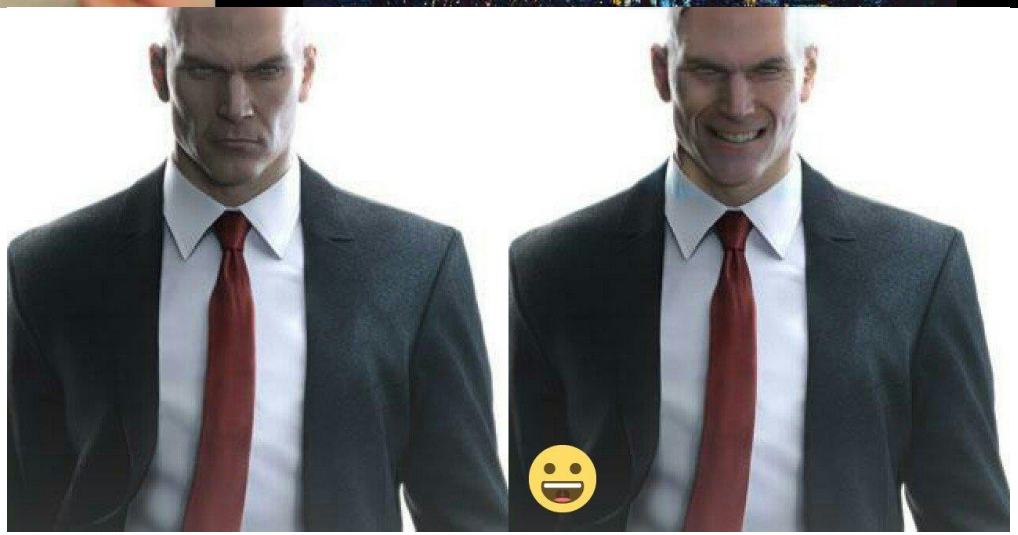
- https://arxiv.org/pdf/1609.04802.pdf



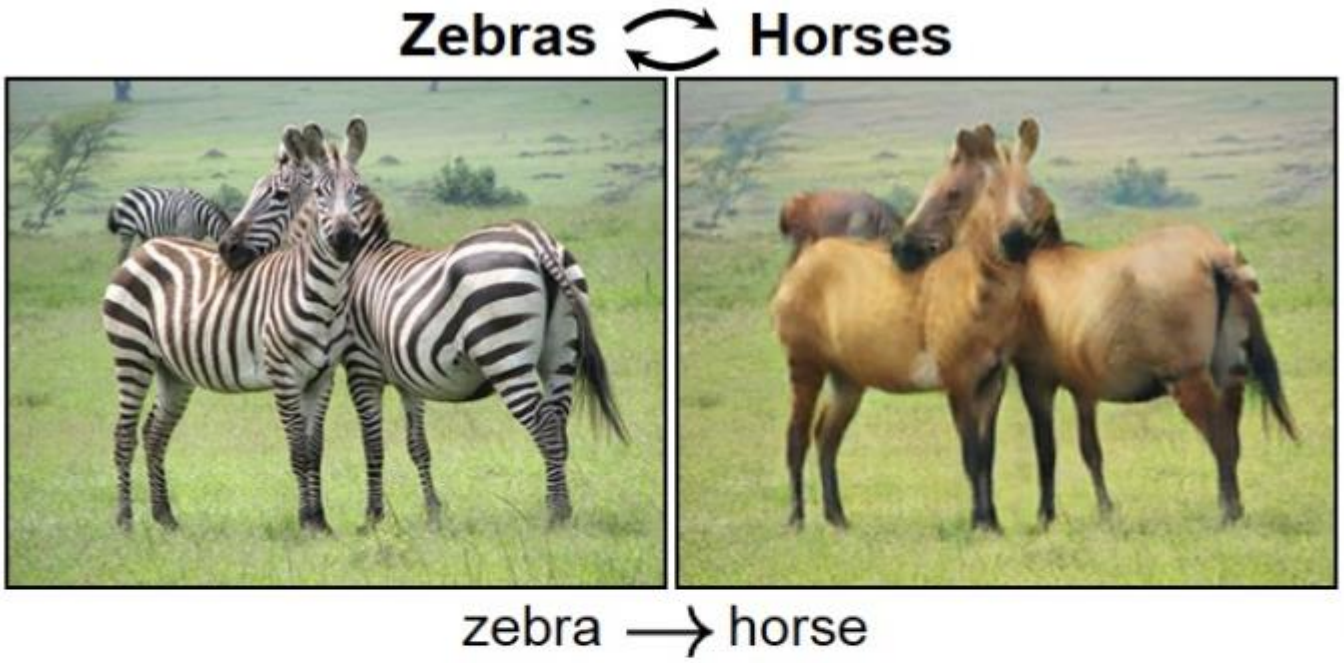
Content-aware Filling

Globally and Locally Consistent Image Completion [Iizuka et al.,

Style transfer with cycleGAN

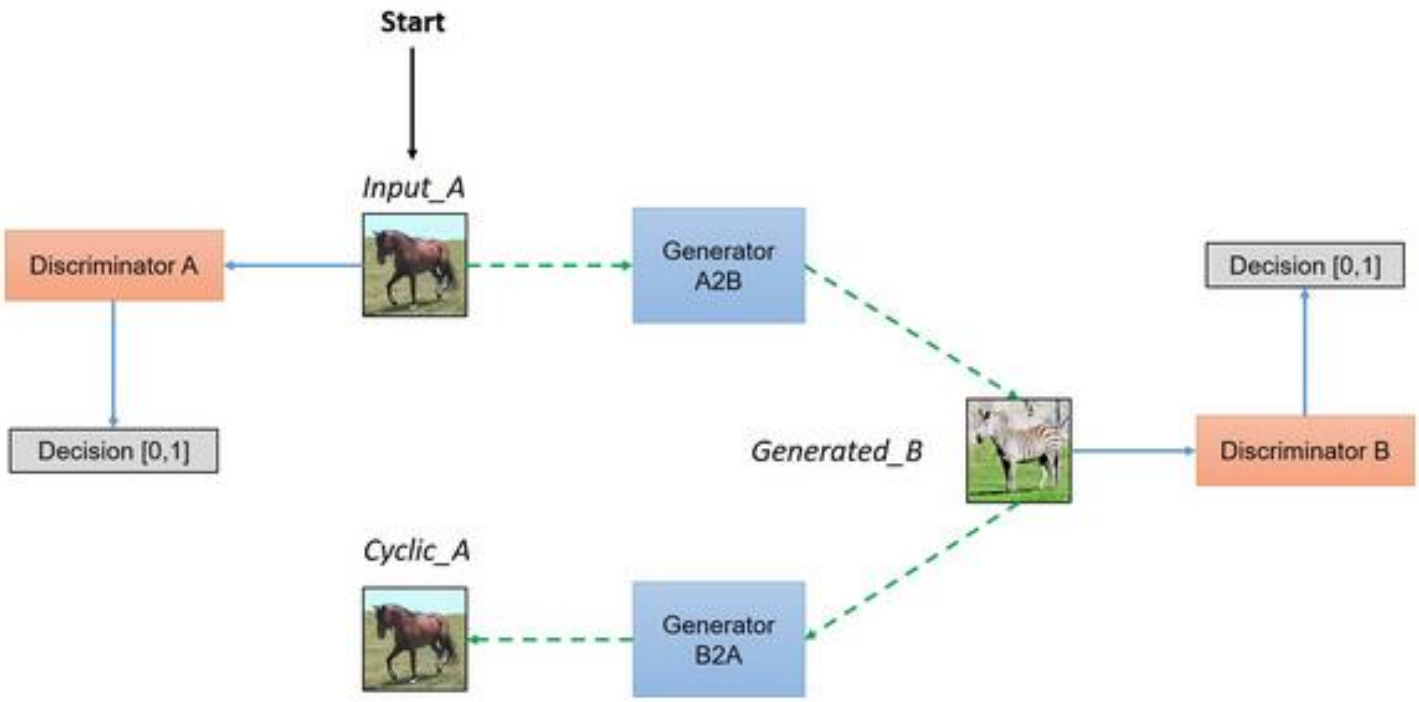


https://hardikbansal.github.io/CycleGANBlog/



Cycle consistency

Note, you don’t need a paired dataset



Cycle consistency

Machine translation with no parallel text

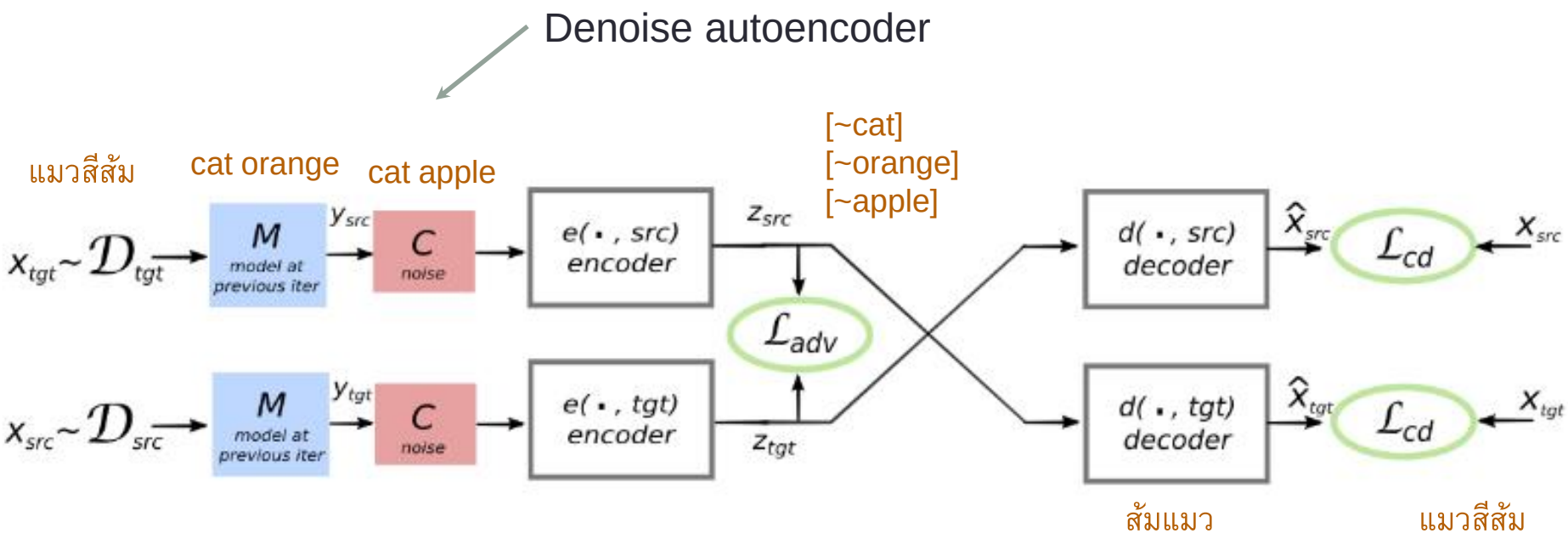
- MT usually requires parallel text

  This is a cat

  นี่คือแมว

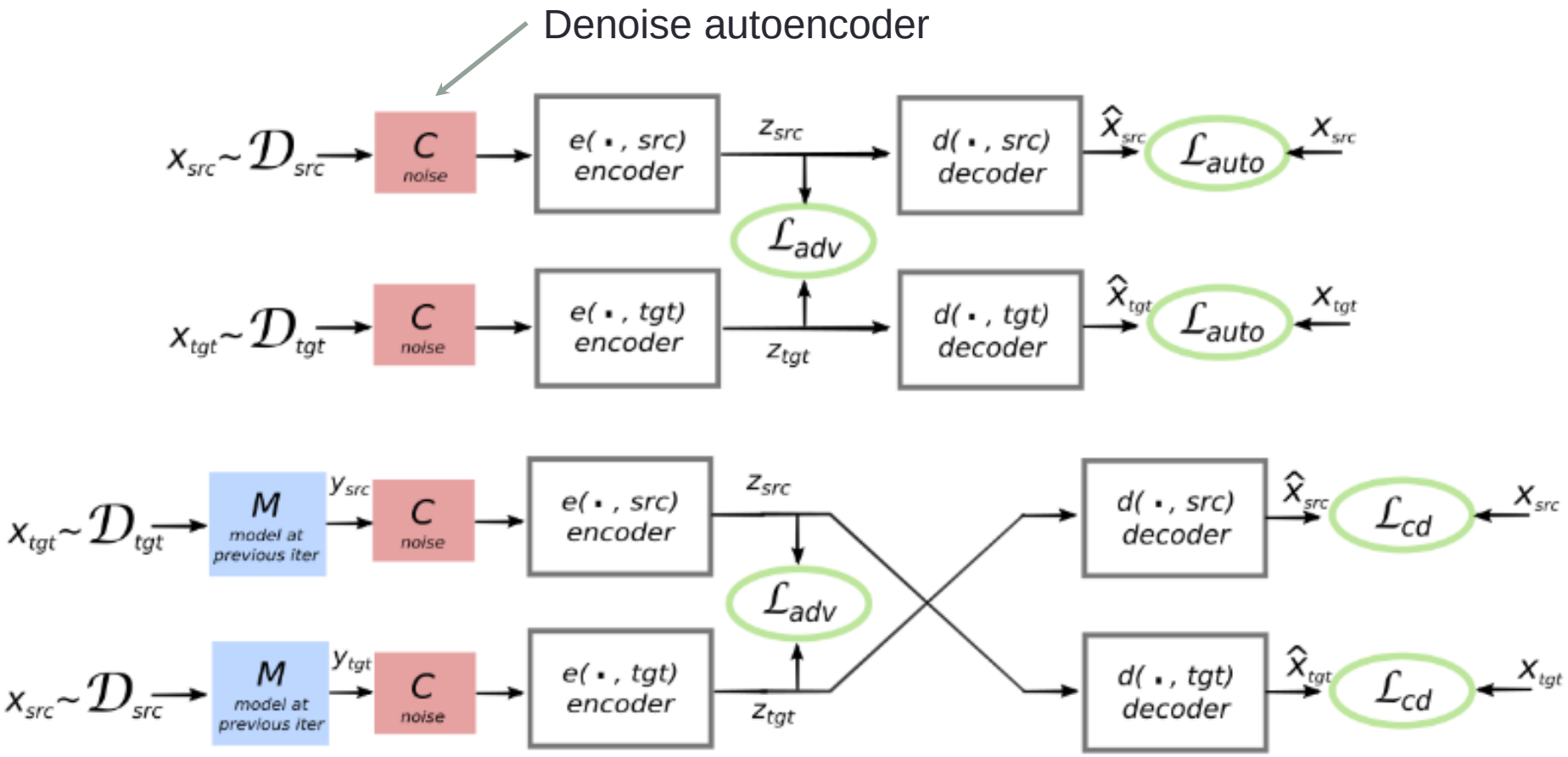
- Most of the time we don’t have parallel text. Just text.
- Can we still do MT?
  - Use GANs + Autoencoders

https://arxiv.org/abs/1711.00043



https://arxiv.org/abs/1711.00043

Use GAN Loss (L_{adv}) to enforce that source and target language pairs share the same distributions.
Then enforce the translation distribution matches.



https://arxiv.org/abs/1711.00043

Results

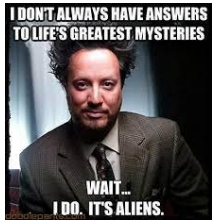
	Multi30k-Task1				WMT			
	en-fr	fr-en	de-en	en-de	en-fr	fr-en	de-en	en-de
Supervised	56.83	50.77	38.38	35.16	27.97	26.13	25.61	21.33
word-by-word	8.54	16.77	15.72	5.39	6.28	10.09	10.77	7.06
word reordering	-	-	-	-	6.68	11.69	10.84	6.70
oracle word reordering	11.62	24.88	18.27	6.79	10.12	20.64	19.42	11.57
Our model: 1st iteration	27.48	28.07	23.69	19.32	12.10	11.79	11.10	8.86
Our model: 2nd iteration	31.72	30.49	24.73	21.16	14.42	13.49	13.25	9.75
Our model: 3rd iteration	32.76	32.07	26.26	22.74	15.05	14.31	13.33	9.64

https://arxiv.org/abs/1711.00043

Adversarial debiasing

- Use the discriminator to force the encoder to drop bias information

Training data



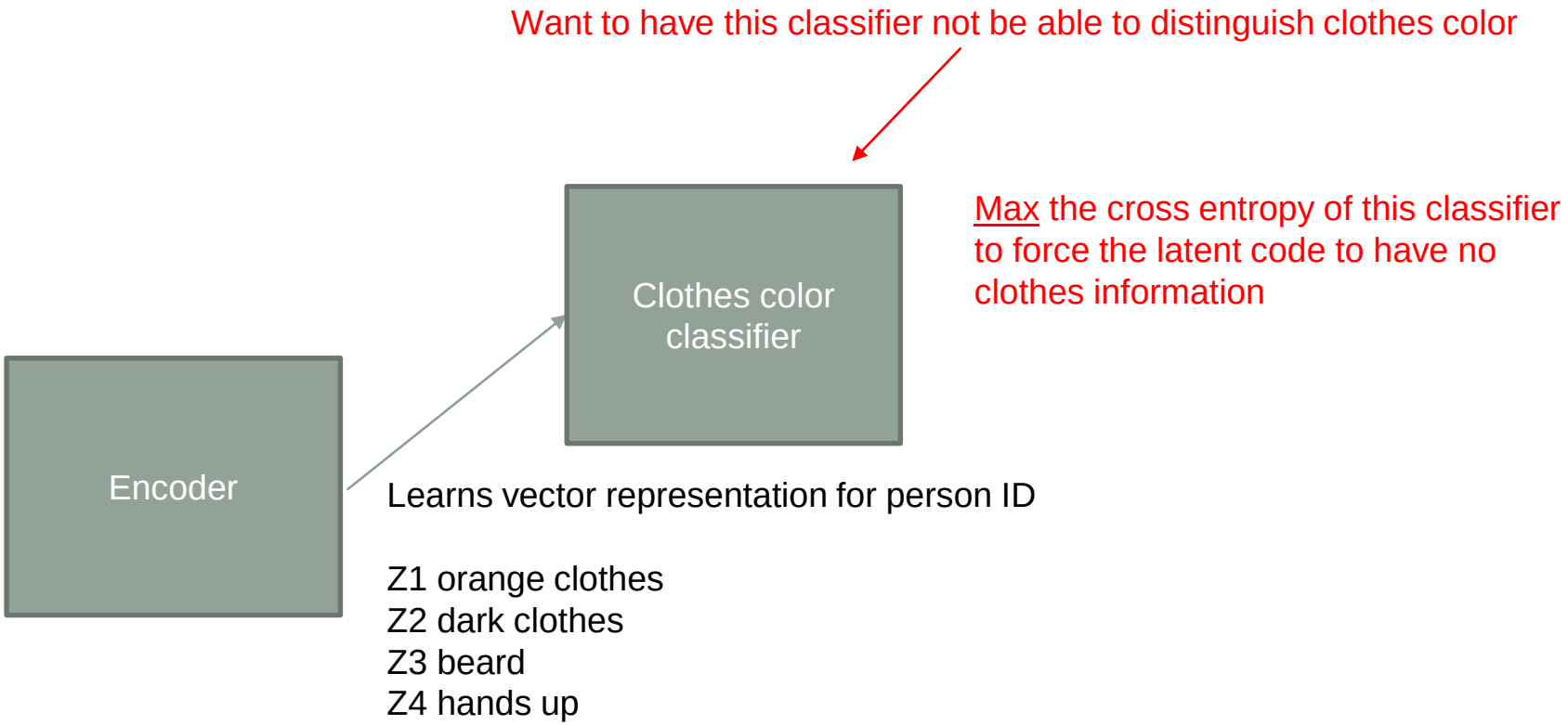
Encoder

Learns vector representation for person ID

Z1 orange clothes
Z2 dark clothes
Z3 beard
Z4 hands up

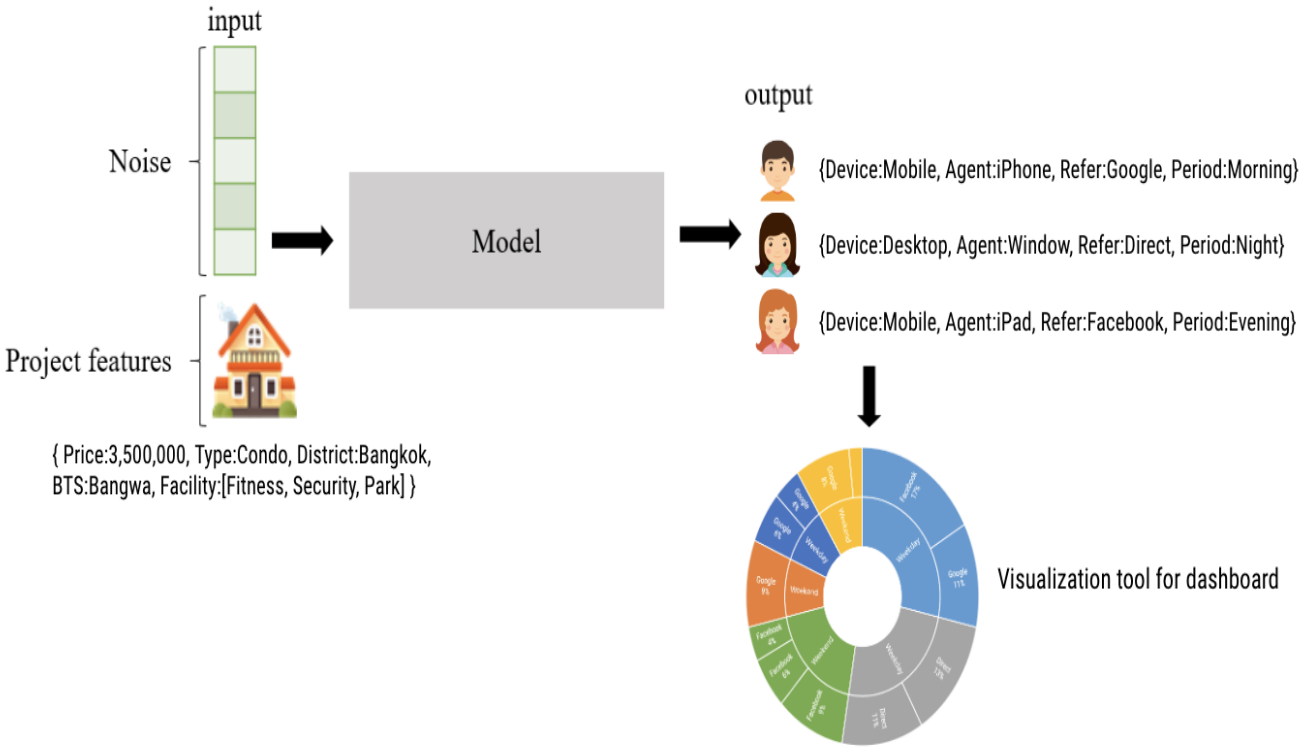
How to remove unwanted information in latent code?

Adversarial debiasing



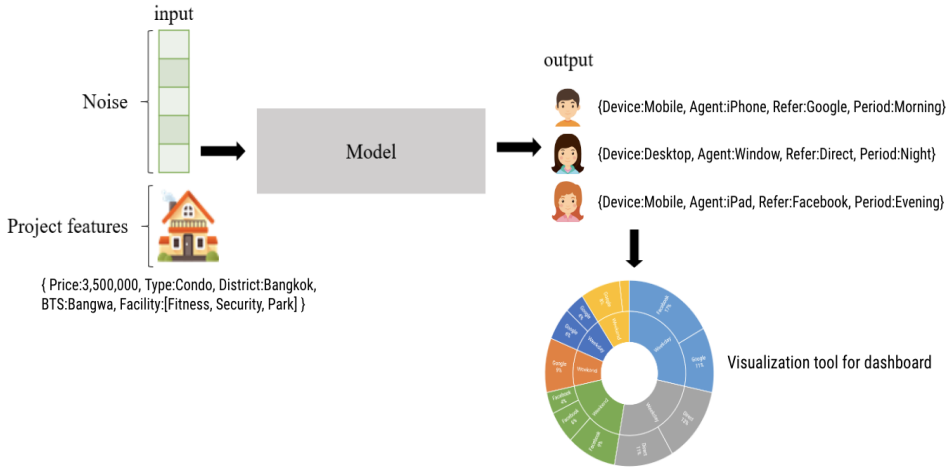
Reading https://arxiv.org/pdf/1801.07593.pdf
Example application usages
Bioinformatics https://www.biorxiv.org/content/10.1101/2021.04.14.439903v1.full.pdf
Face recognition https://arxiv.org/pdf/1911.08080.pdf

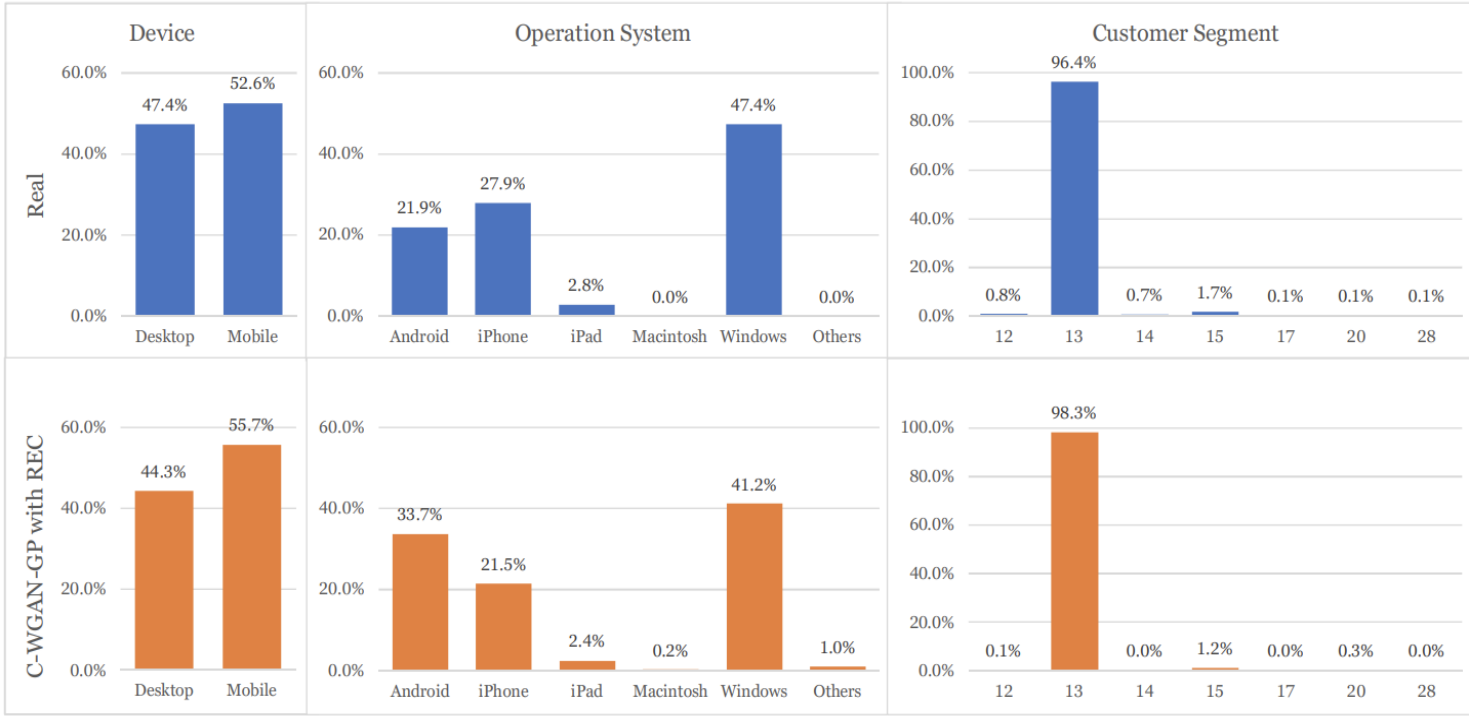
New product development



https://ieeexplore.ieee.org/document/9016238

Predicting users





https://ieeexplore.ieee.org/document/9016238

GAN vs VAE

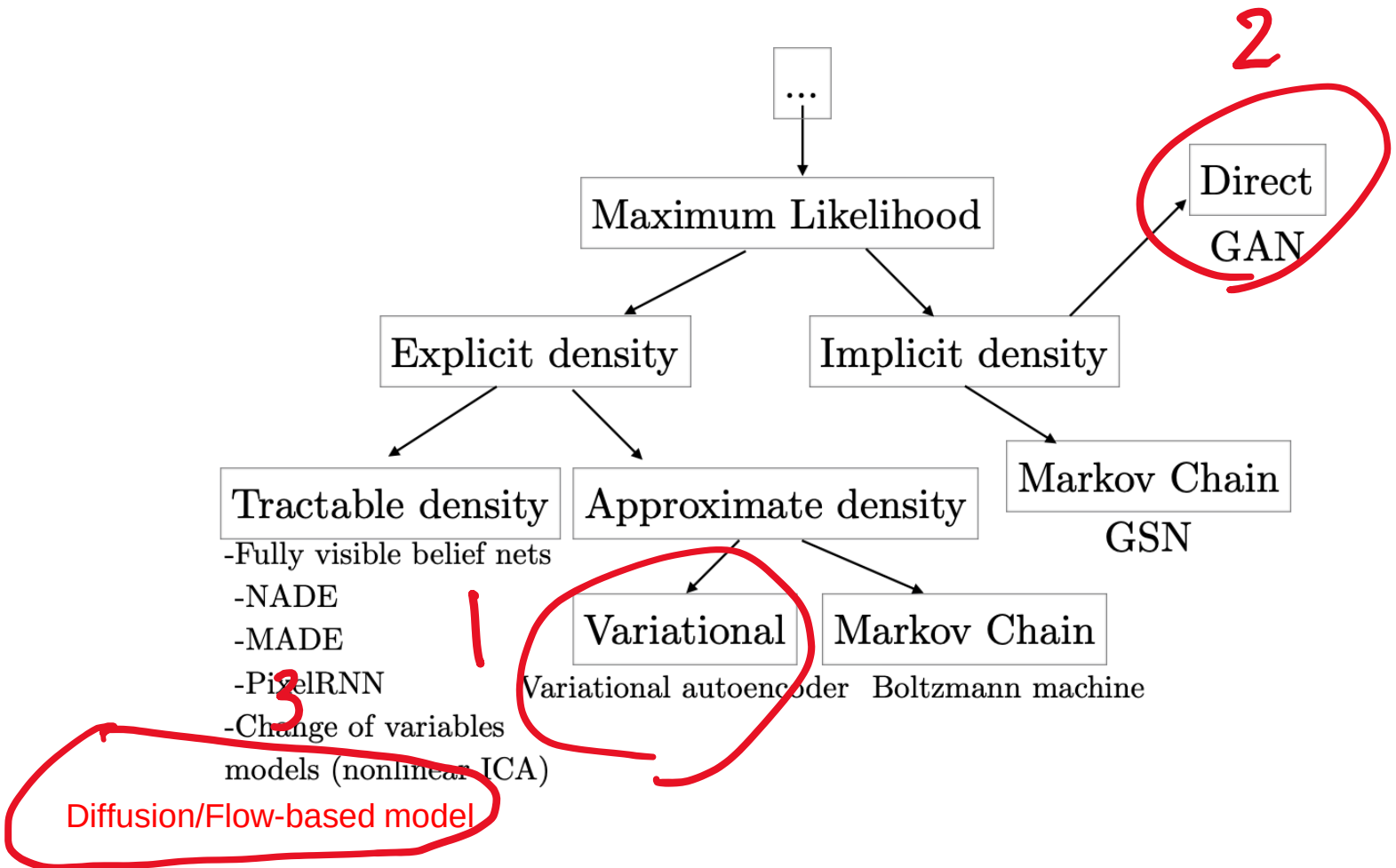
- Typically GAN provides better quality
- Typically VAE provides more controllability
  - Representation more disentangled
  - Can easily interpolate two inputs
- VAE can encode nice embeddings/features for other task
- Research exist to fill both gaps

Deep generative models

- Autoencoders
  - Variational autoencoders
- GAN
- Diffusion/Flow

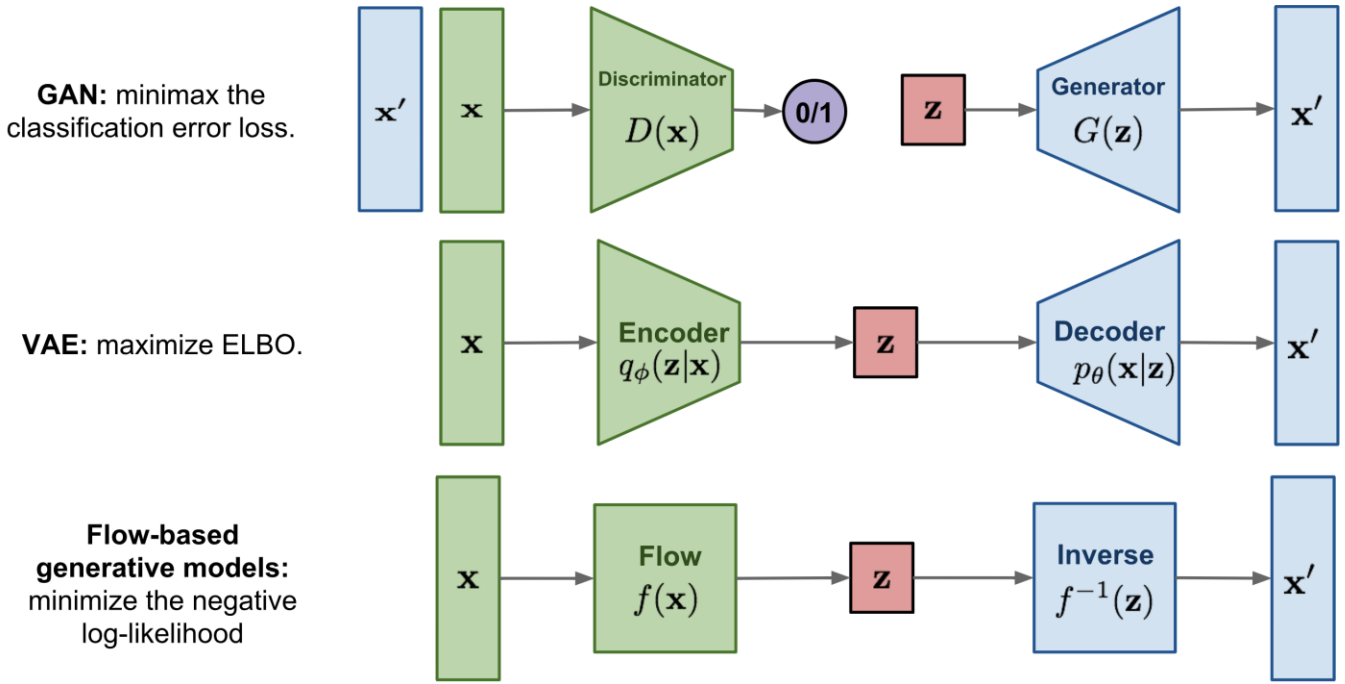
Deep Generative Models

- A deep learning model that can be used to generate X



Flow-based model

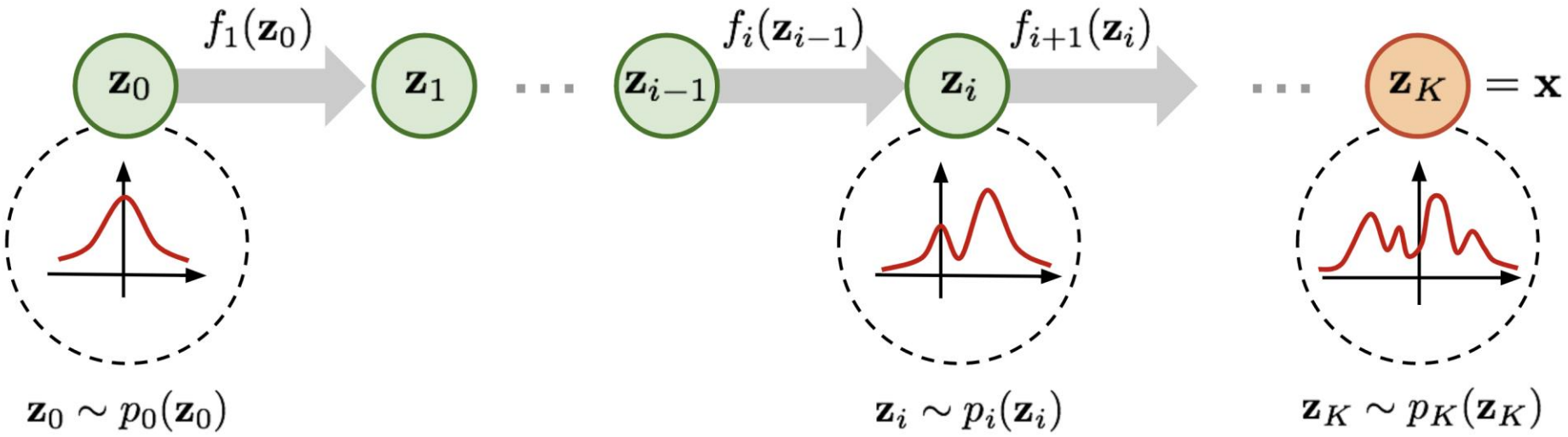
- Tweak distribution until you get the target distribution
- Deterministic reversible/invertable tweaks



https://lilianweng.github.io/posts/2018-10-13-flow-models/

https://lilianweng.github.io/posts/2018-10-13-flow-models/

Flow-based model

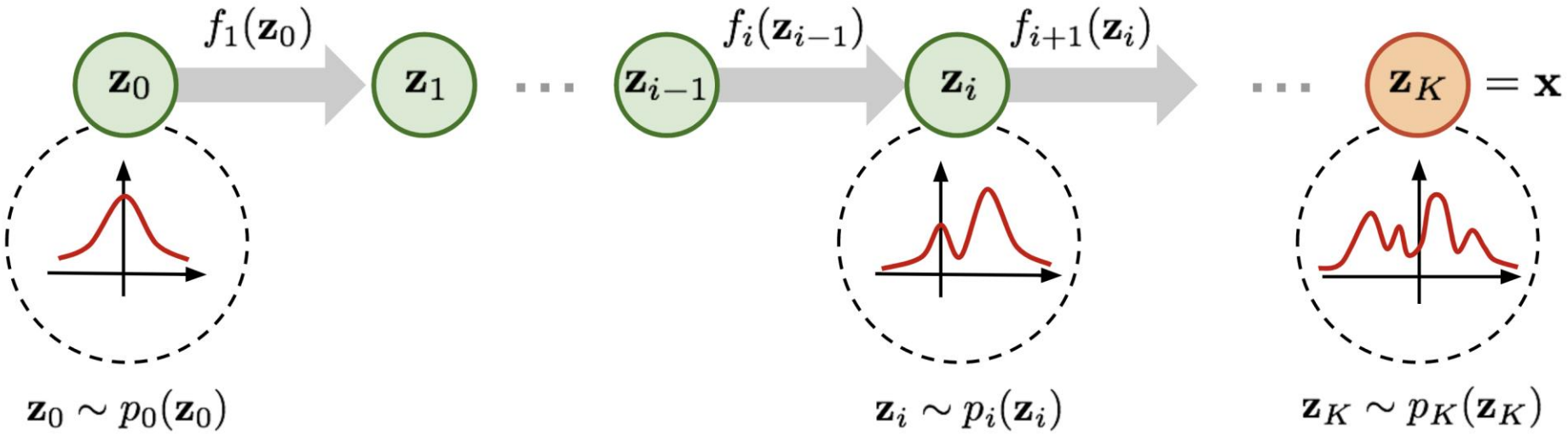


Noise

Target data

$$
\begin{aligned}
\mathbf{x} = \mathbf{z}_K &= f_K \circ f_{K-1} \circ \cdots \circ f_1(\mathbf{z}_0) \\
\log p(\mathbf{x}) = \log \pi_K(\mathbf{z}_K) &= \log \pi_{K-1}(\mathbf{z}_{K-1}) - \log \left| \det \frac{df_K}{d\mathbf{z}_{K-1}} \right| \\
&= \log \pi_{K-2}(\mathbf{z}_{K-2}) - \log \left| \det \frac{df_{K-1}}{d\mathbf{z}_{K-2}} \right| - \log \left| \det \frac{df_K}{d\mathbf{z}_{K-1}} \right| \\
&= \dots \\
&= \log \pi_0(\mathbf{z}_0) - \sum_{i=1}^{K} \log \left| \det \frac{df_i}{d\mathbf{z}_{i-1}} \right|
\end{aligned}
$$

Flow-based model



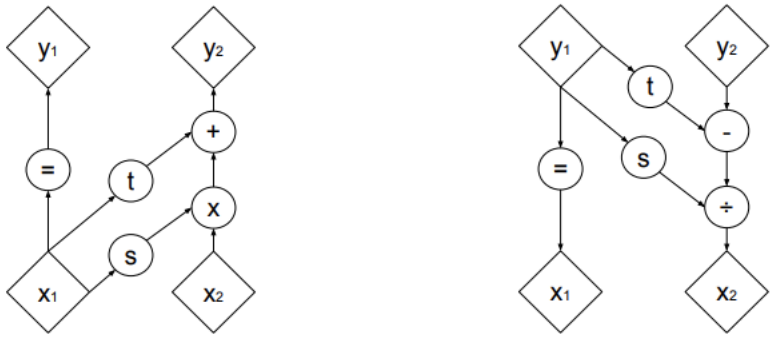
Noise

Target data

Find f that maximizes the likelihood of the data
Constrain f so that it is easily invertable and det easy to compute
f is a neural network (needs networks that's easily invertable by design)

Example of invertible layer

- RealNVP



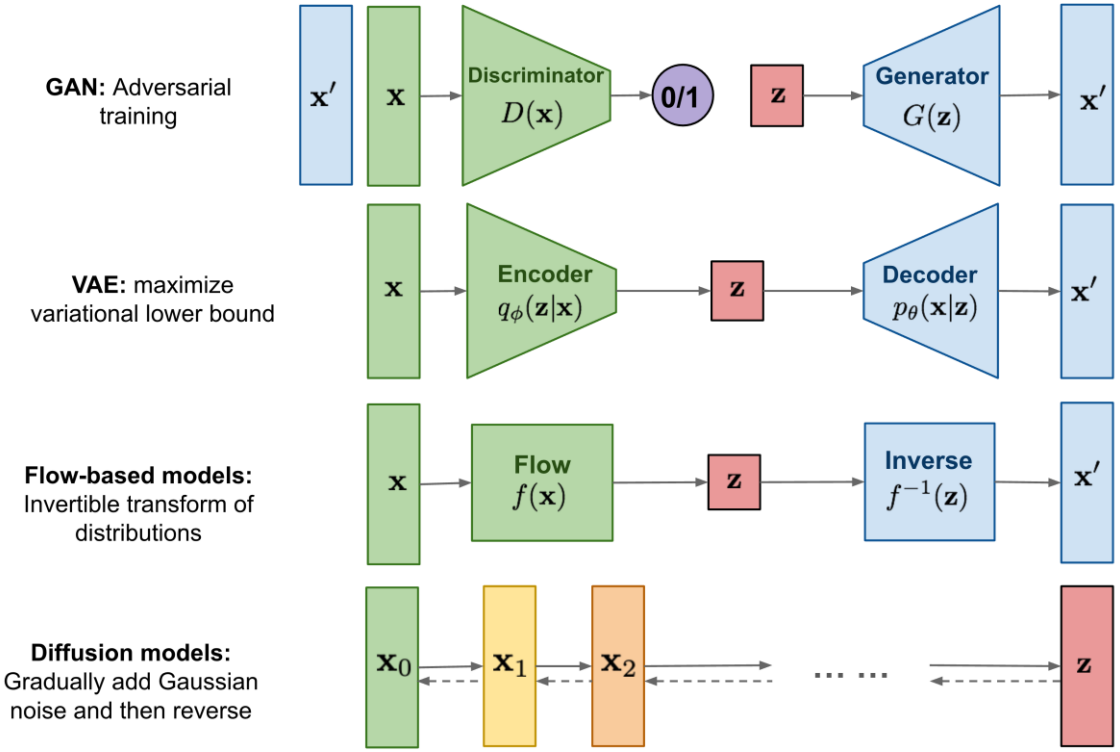
(a) Forward propagation (b) Inverse propagation

Figure 2: Computational graphs for forward and inverse propagation. A coupling layer applies a simple invertible transformation consisting of scaling followed by addition of a constant offset to one part $\mathbf{x}_2$ of the input vector conditioned on the remaining part of the input vector $\mathbf{x}_1$. Because of its simple nature, this transformation is both easily invertible and possesses a tractable determinant. However, the conditional nature of this transformation, captured by the functions s and t, significantly increase the flexibility of this otherwise weak function. The forward and inverse propagation operations have identical computational cost.

https://arxiv.org/abs/1605.08803

Diffusion model

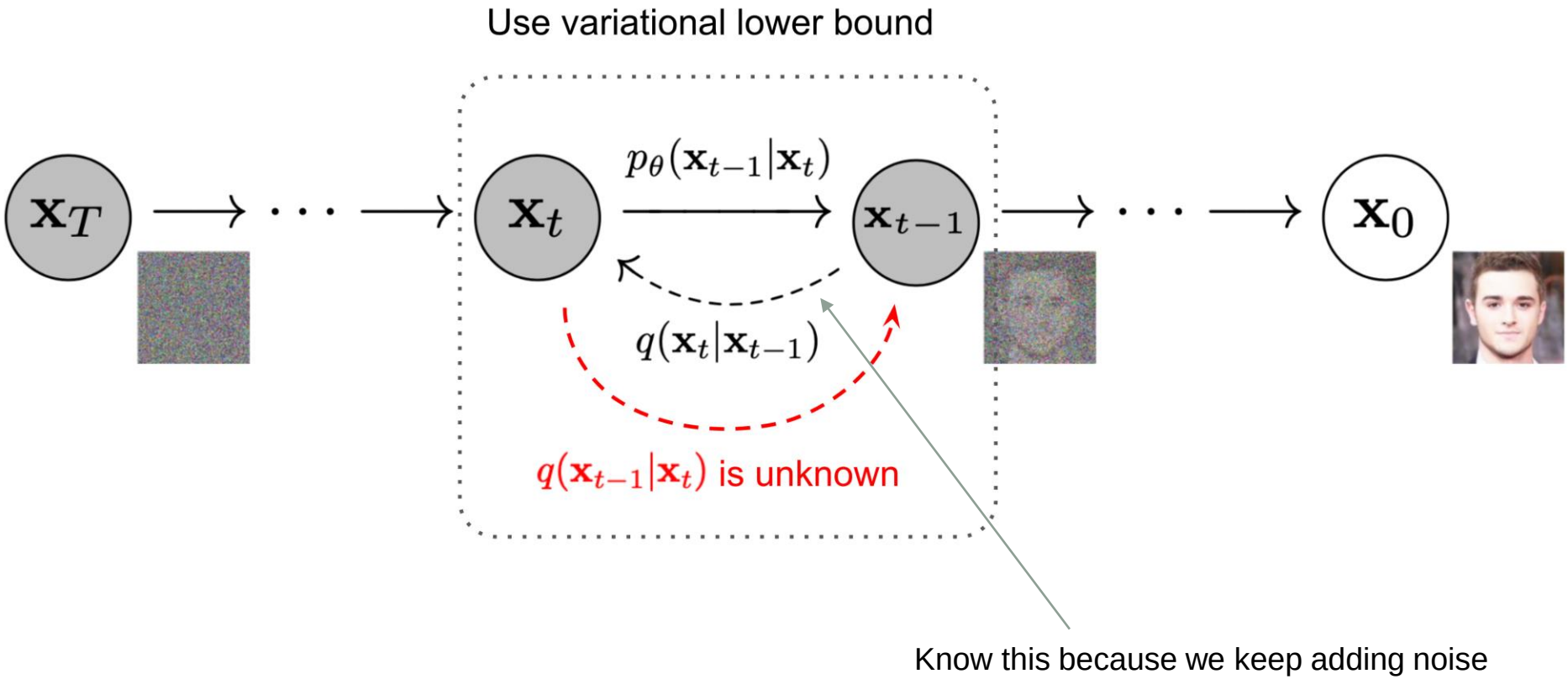
- Similar to flow
  - No longer constrained to revertible operations
  - Add noise to image until you get just noise



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion model

Diffusion model tries to learn how to remove the noise in each step



Notes

- Diffusion model is now state-of-the-art in many generation tasks
- Very slow (need hundreds iterative steps of adding noise)
- The latent of diffusion and flow models are of the same size of the original data (does not compress).
- Diffusion + flow = Diffusion normalization flow

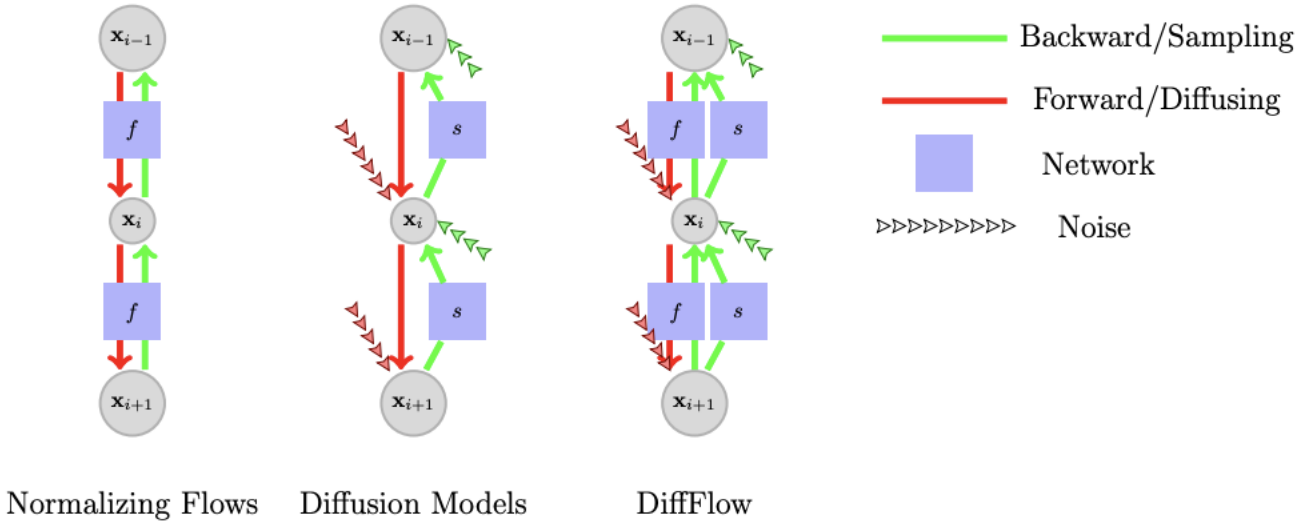


Figure 1: The schematic diagram for normalizing flows, diffusion models, and DiffFlow. In normalizing flow, both the forward and the backward processes are deterministic. They are the inverse of each other and thus collapse into a single process. The diffusion model has a fixed forward process and trainable backward process, both are stochastic. In DiffFlow, both the forward and the backward processes are trainable and stochastic.

https://arxiv.org/abs

Recommend reading
https://ai-scholar.tech/en/articles/diffusion-model/diffusion_normalizing_flow

Deep Generative Models

- A deep learning model that can be used to generate X