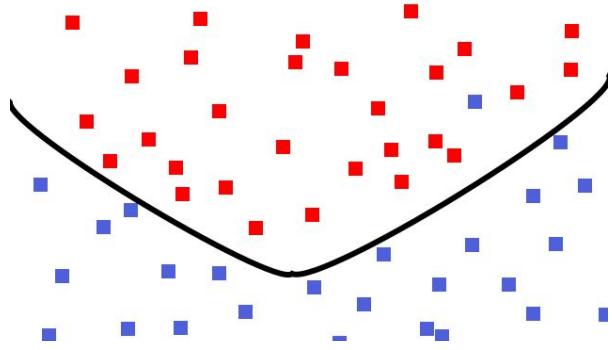


# UNSUPERVISED LEARNING

---

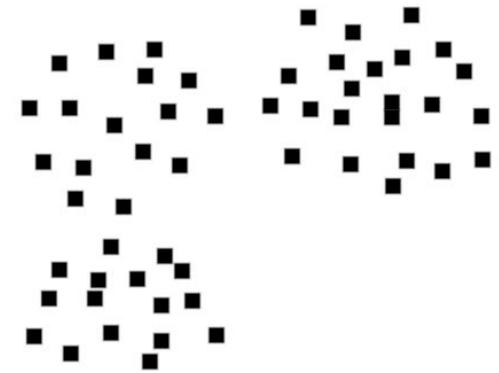
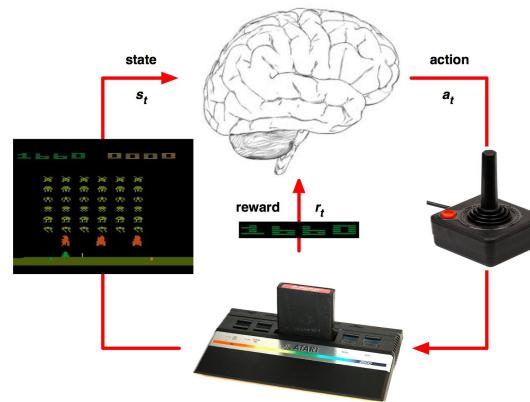
# 3 Modes of Learning



Supervised Learning



Reinforcement Learning

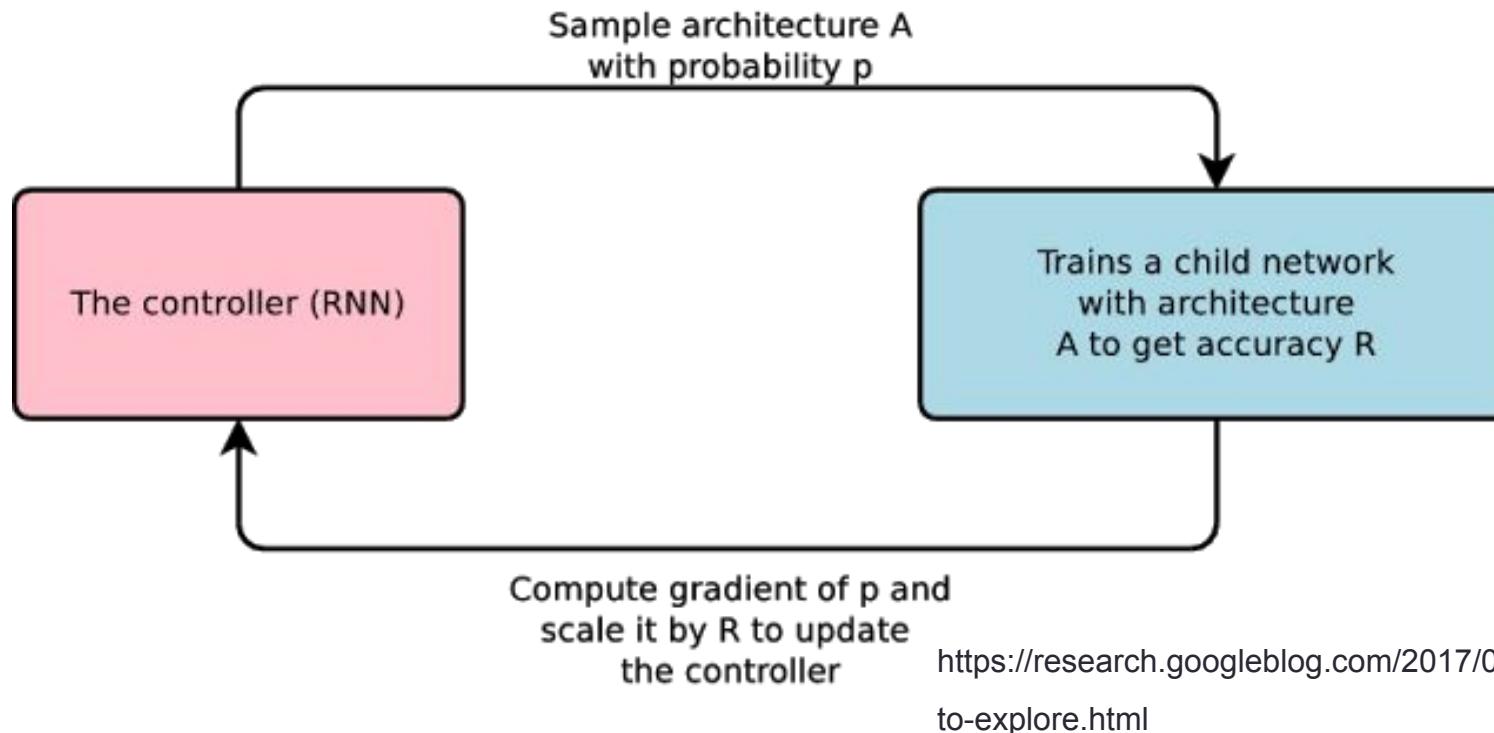


Unsupervised Learning



# Model selection for deep architectures

- Tuning a network takes time
- Let machine learning learns how to tune a network
- Matches or outperforms ML experts performance

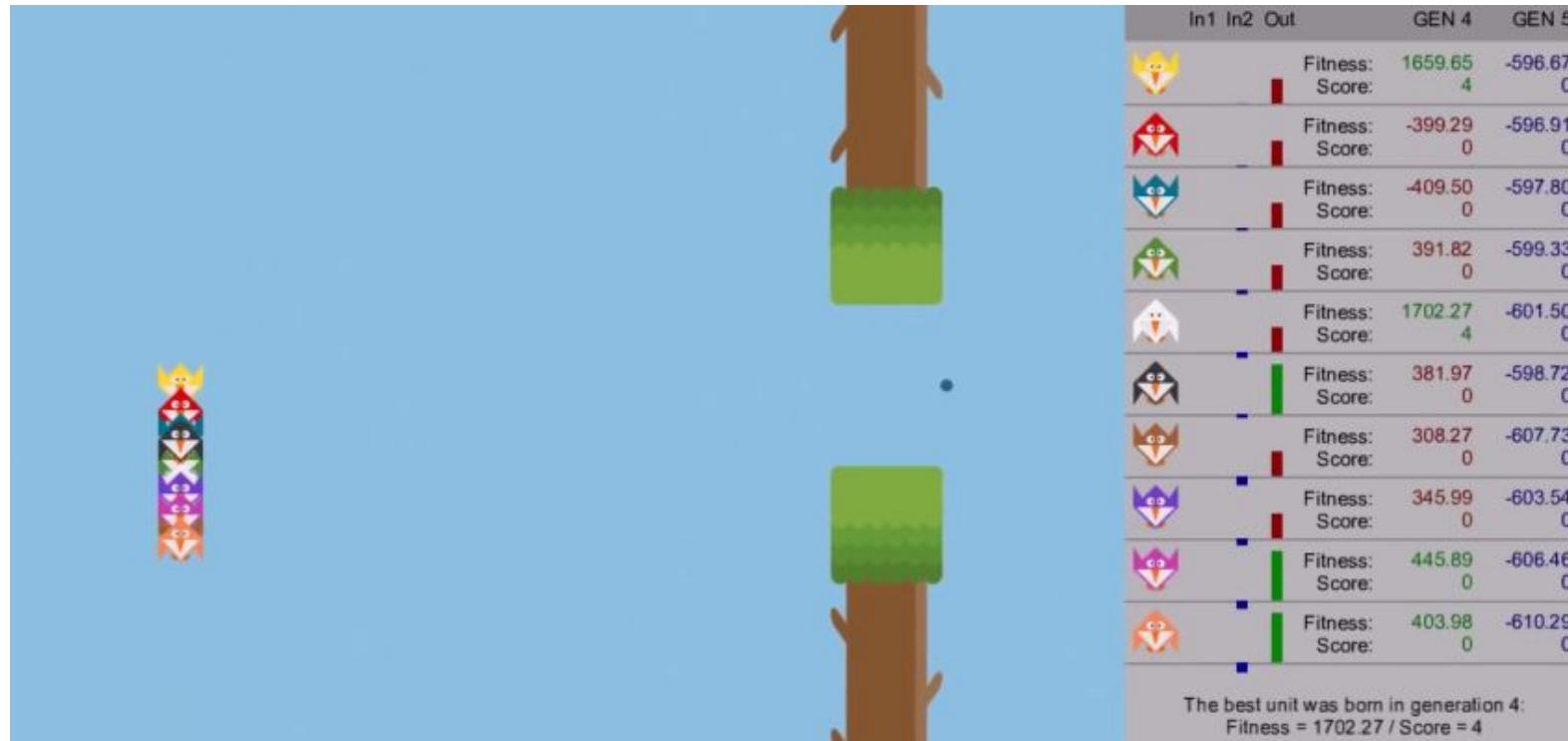


# Genetic Algorithm for Reinforcement learning

- Updating based on the gradients for RL is hard
  - Credit assignment problem
- Estimate the direction to minimize the loss by random perturbations
  - Perturb the weights and let the new agents roam in the environment
- Highly parallelizable

# Flappy bird genetic algorithm

- <https://www.youtube.com/watch?v=aeWmdojEJf0>



# Yann LeCun's comment



**Yann LeCun**

March 14, 2016 ·

Follow



Statement from a Slashdot post about the AlphaGo victory: "We know now that we don't need any big new breakthroughs to get to true AI"

That is completely, utterly, ridiculously wrong.

As I've said in previous statements: most of human and animal learning is unsupervised learning. If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake.

We need to solve the unsupervised learning problem before we can even think of getting to true AI. And that's just an obstacle we know about. What about all the ones we don't know about?

#deeplearning #AI #AlphaGo

# How Much Information Does the Machine Need to Predict?

Y LeCun

## ■ "Pure" Reinforcement Learning (cherry)

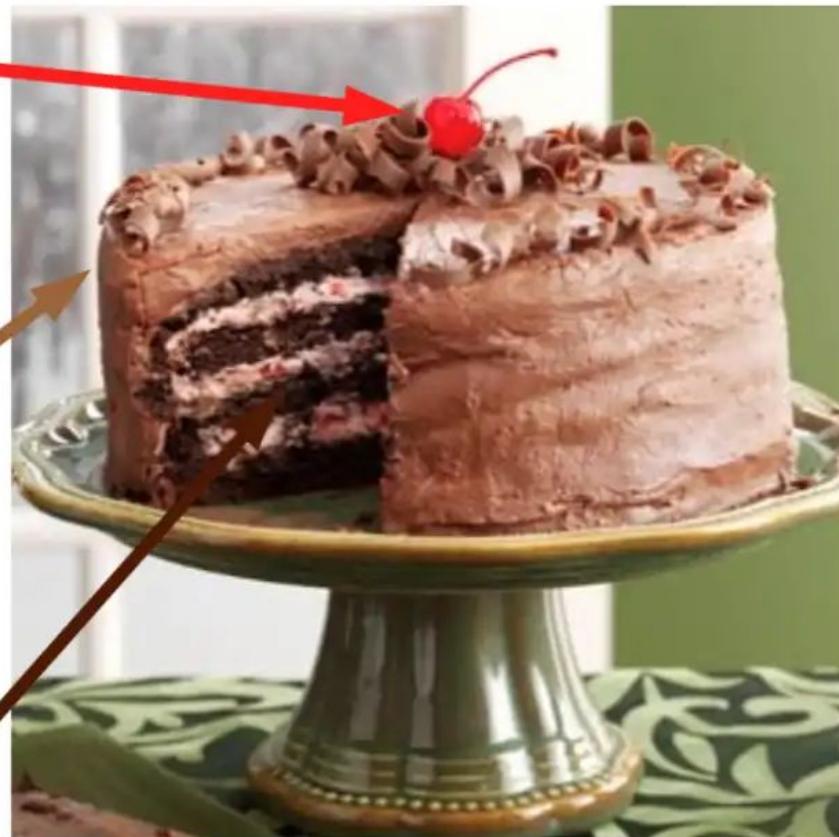
- ▶ The machine predicts a scalar reward given once in a while.
- ▶ **A few bits for some samples**

## ■ Supervised Learning (icing)

- ▶ The machine predicts a category or a few numbers for each input
- ▶ Predicting human-supplied data
- ▶ **10→10,000 bits per sample**

## ■ Unsupervised/Predictive Learning (cake)

- ▶ The machine predicts any part of its input for any observed part.
- ▶ Predicts future frames in videos
- ▶ **Millions of bits per sample**



■ (Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)

# Unsupervised Learning

- Autoencoders
  - Denoising Autoencoder
  - VAE
    - Evidence lower bound
    - Perceptual loss
- Generative Adversarial Networks
  - Random Cool stuff
- Clustering

# “Unsupervised” Learning

- Given an image -> Tell you right away it's a cat without anyone labeling anything?

**NO**

- What does it do exactly?

Can we somehow utilize unlabeled data to help with some task we care about?

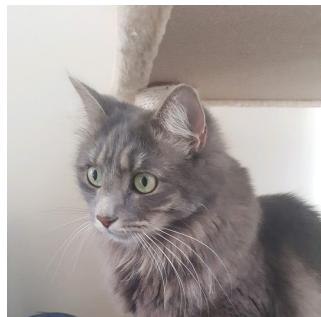
# First Unsupervised Learning Idea

- Suppose we have a magic function  $f$  that maps an image to two numbers

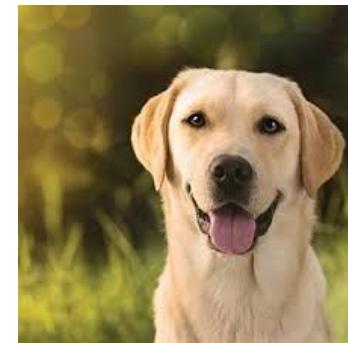


# First Unsupervised Learning Idea

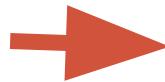
- Suppose we have a magic function  $f$  that maps an image to two numbers



(4.1, 5.0)



(8.1, 8.0)



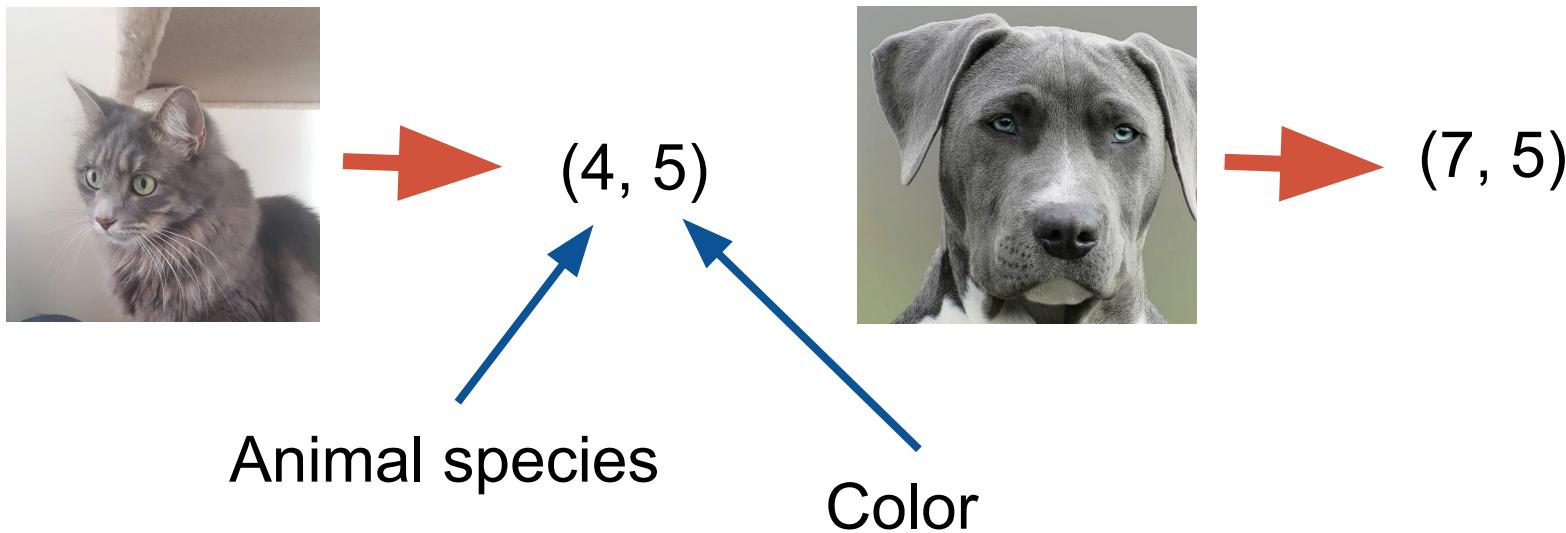
(4.2, 9.1)



(7.9, 4.9)

# First Unsupervised Learning Idea

- Ideal magic function  $f$

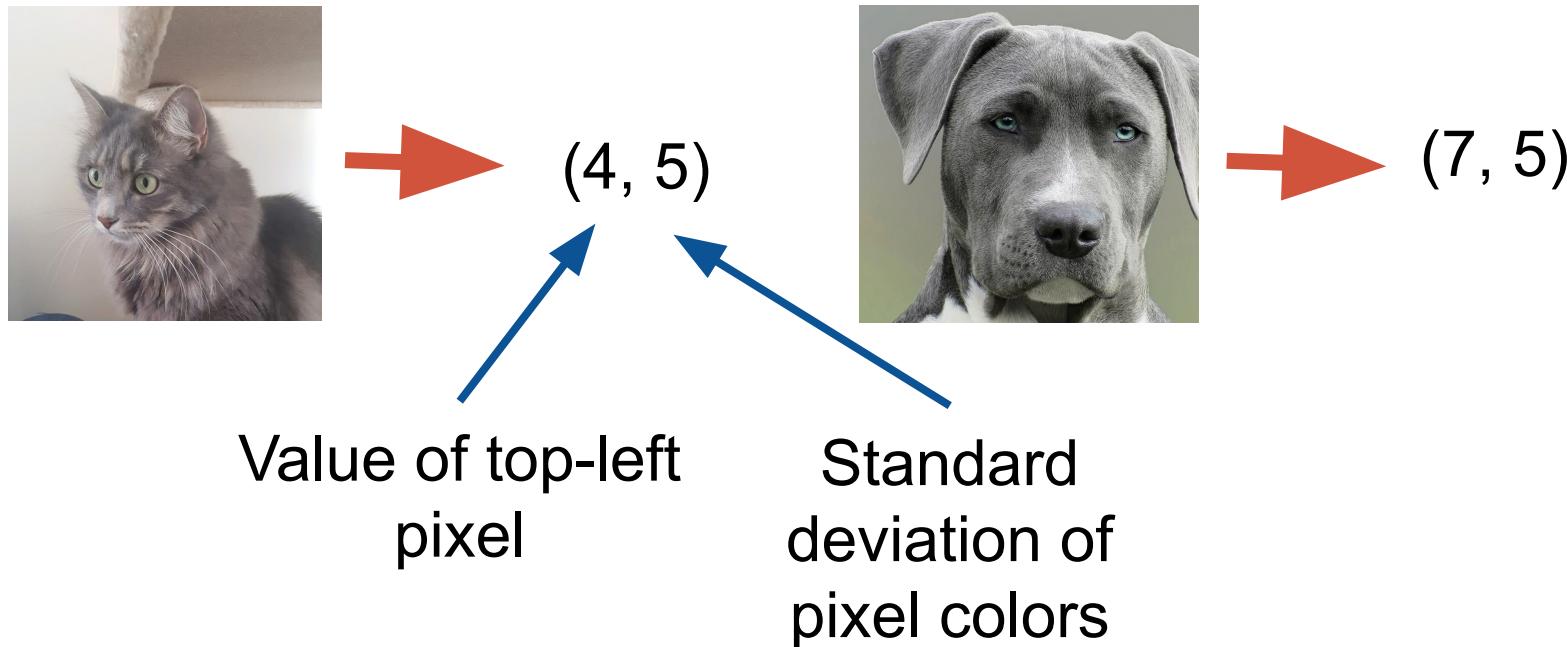


Getting this  $f$  would be so nice!

Given  $f$ , we can recognize animals with a linear classifier!

# First Unsupervised Learning Idea

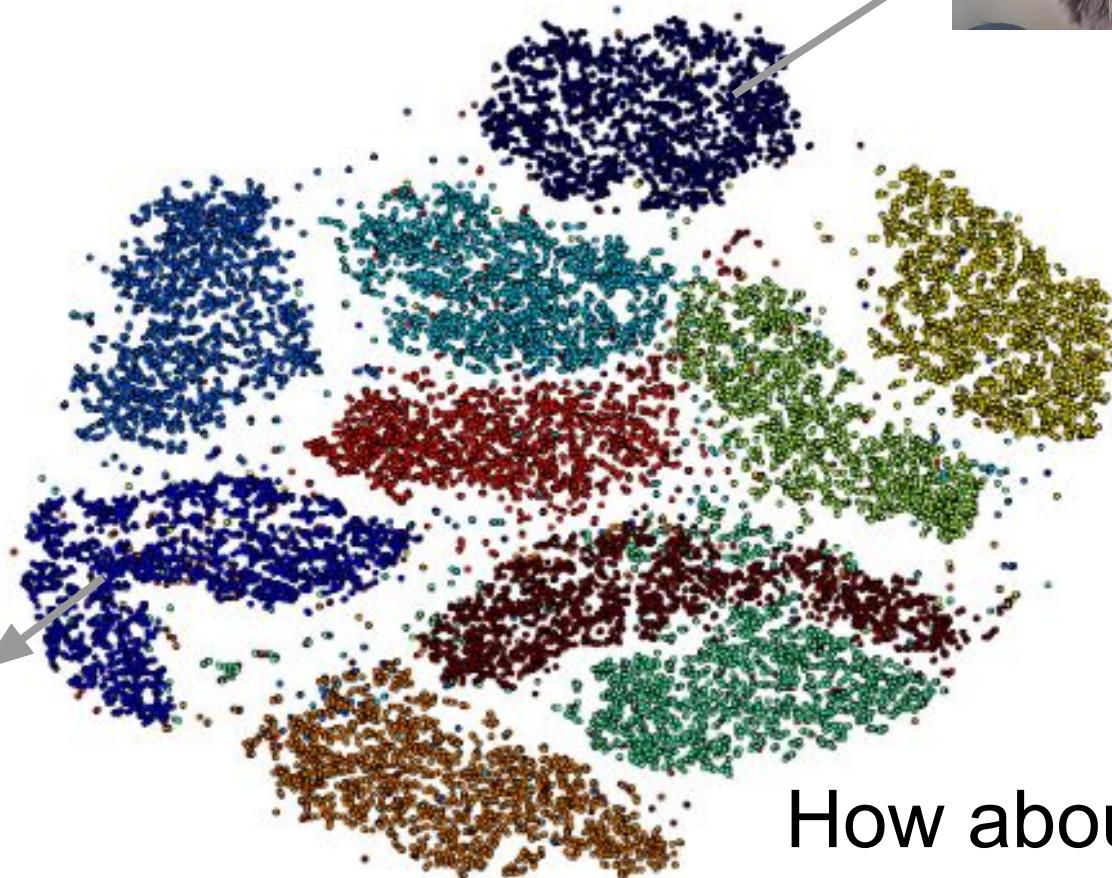
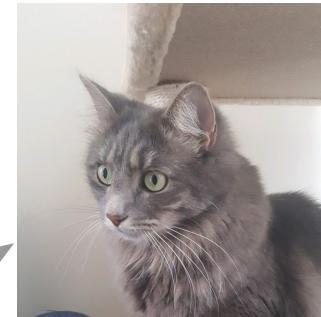
An alternative magic function  $f$



This  $f$  isn't so useful

**Task:** We try to learn the most “useful”  $f()$  from unlabeled data.  
How to define usefulness? [subjective: Natural, human tasks]

# One solution: Clustering



How about K-Mean?  
PCA?

# How about K-mean with L2?

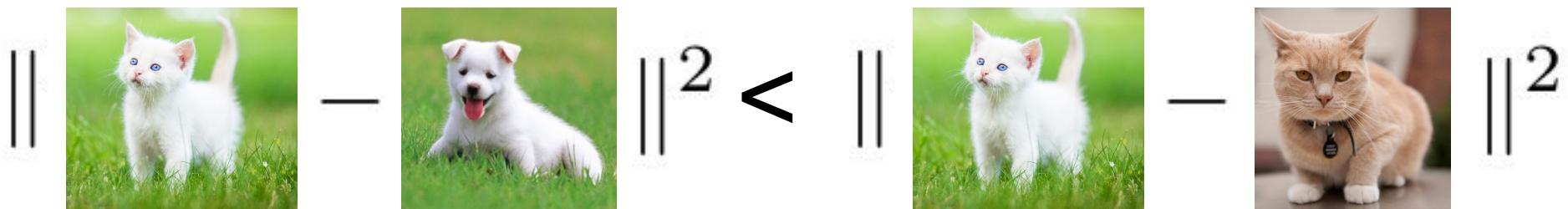
We need a distance metric: How about pixelwise L2 norm?

$$\sum_p |I_p^1 - I_p^2|^2$$

In vector form:



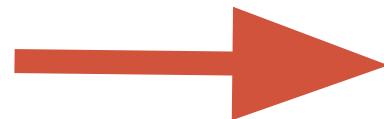
# How about K-mean with L2?



White cat is more similar to white dog than to orange cat!

Pixel-wise L2 doesn't work that well. Too low-level comparison.  
How can we come up with a useful distance metric? [DNN]

# How about PCA?



(4.1, 5.0)

PCA

Dimensionality reduction  
from  $(h \times w)$  dim to 2 dim

Will this work?

Historically, can be used to do simple face recognition.

# PCA: Eigenfaces

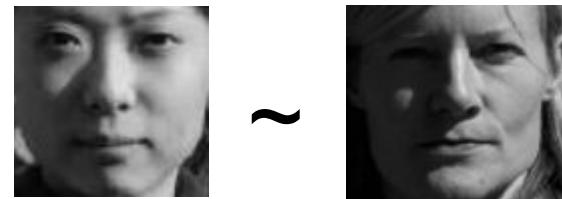


PCA basis

# PCA: Eigenfaces

$$\text{Face} = \text{Eigenface}_1 + \text{Eigenface}_2 + \text{Eigenface}_3 + \text{Eigenface}_4 + \text{Eigenface}_5 + \dots$$

$x 0.3$        $x 0.2$        $x 0.7$        $x 0.9$        $x 0.1$

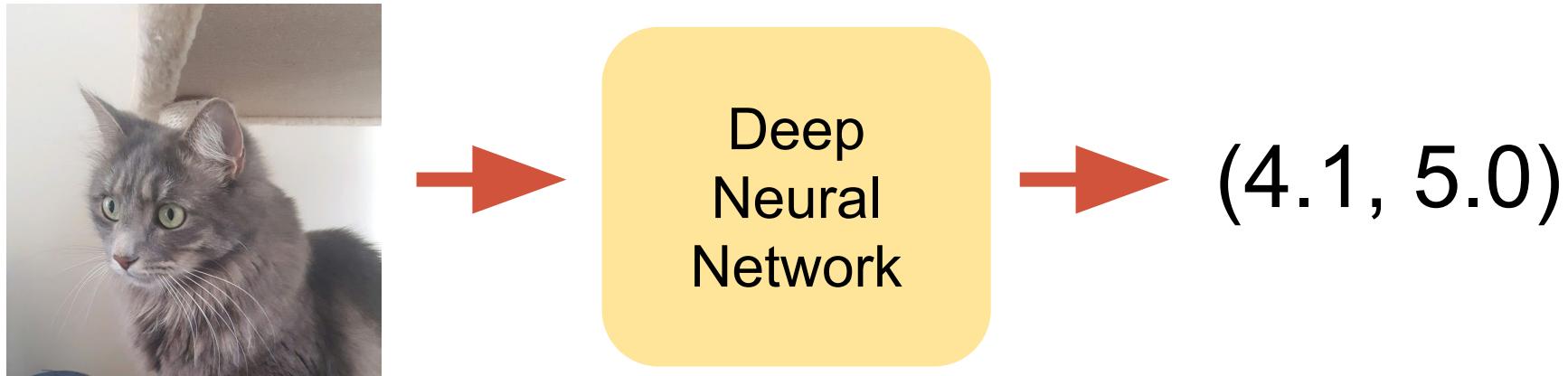


## Problems:

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?

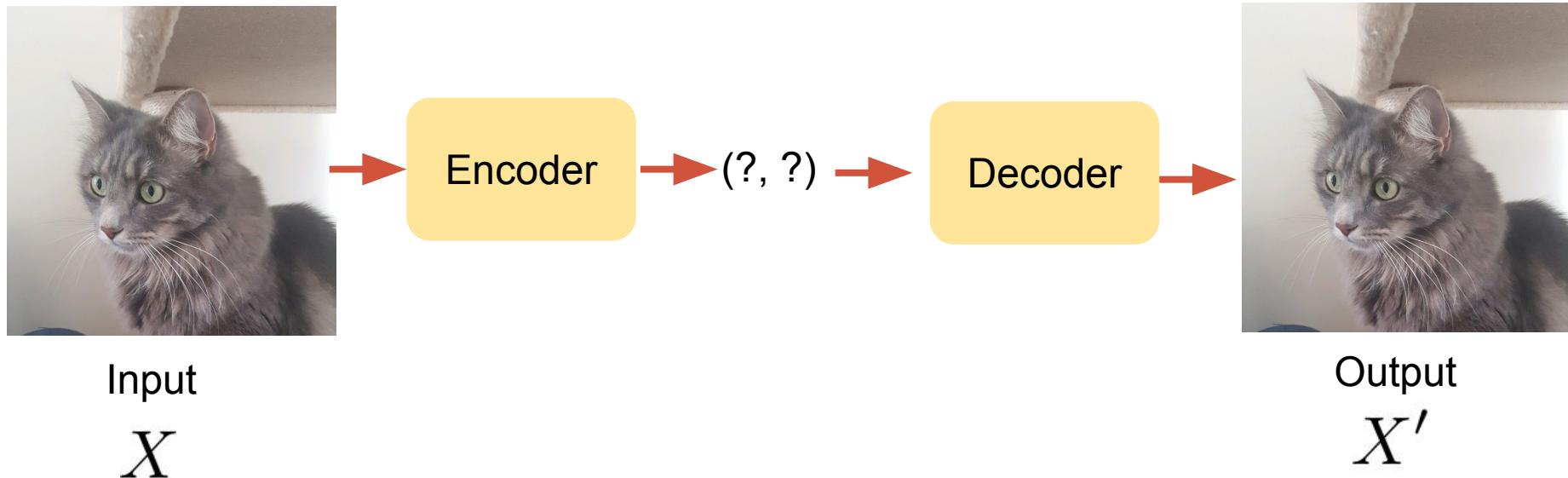
We know that DNN can model highly complex mapping.  
How about modeling  $f$  with a neural net?



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

# Autoencoder

Auto means self!



Input

$X$

Output

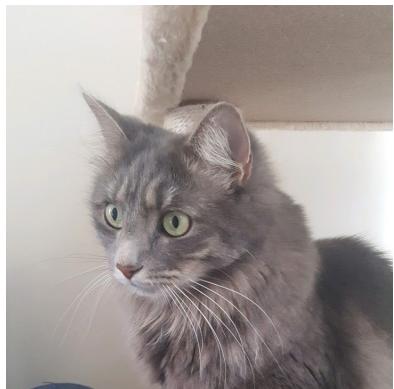
$X'$

Now we can train the network with L2 Loss:

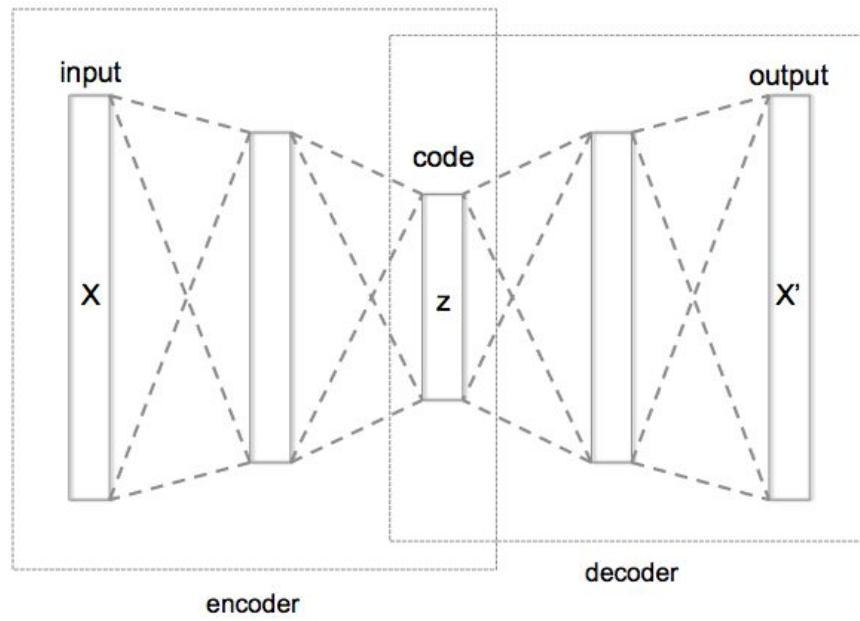
$$\|X - X'\|^2$$

# Autoencoder

Auto means self!



Input  
 $X$

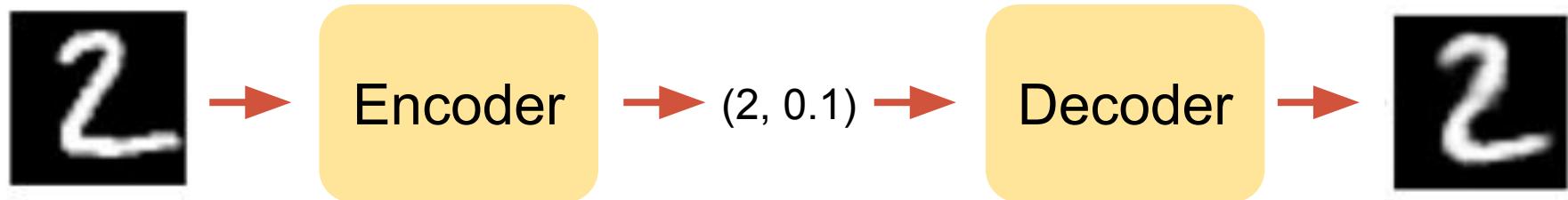


Output  
 $X'$

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.

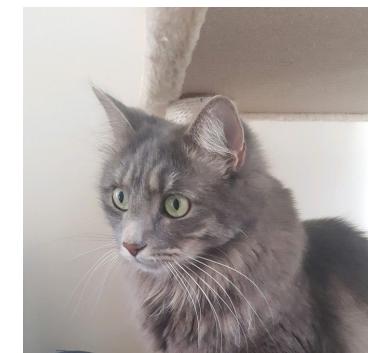
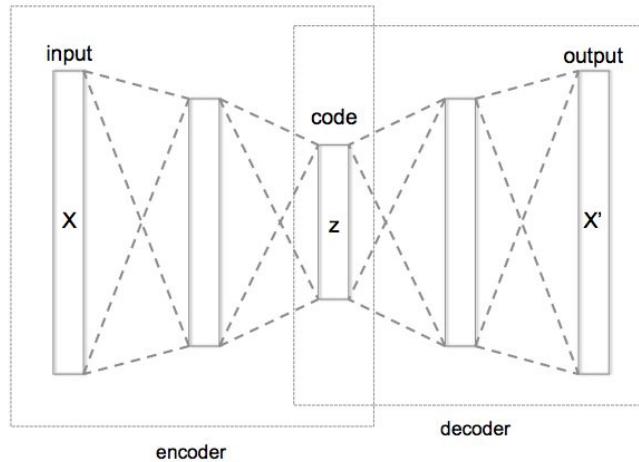
# Autoencoder

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  $\rightarrow$  useless.



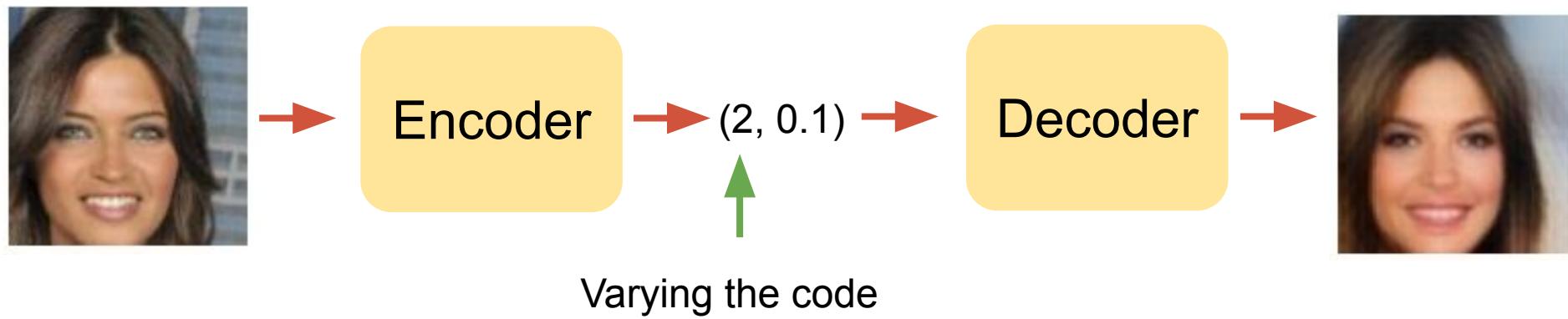
Input  
 $X$



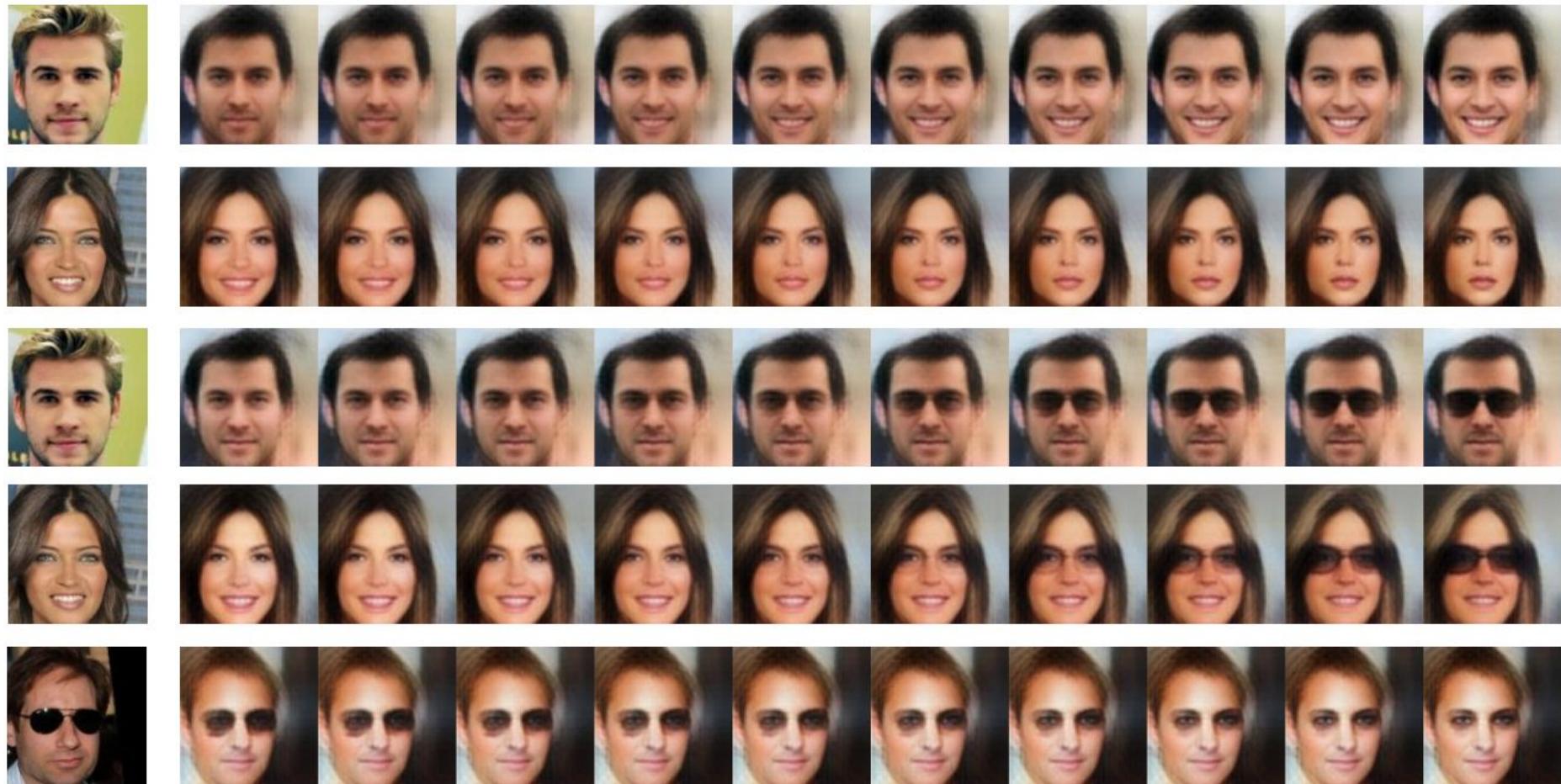
Output  
 $X'$

- When  $z$  dim  $\ll$  input dim, NN has to work hard!

# Inspecting attributes learned from AE



# Attributes from autoencoder



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

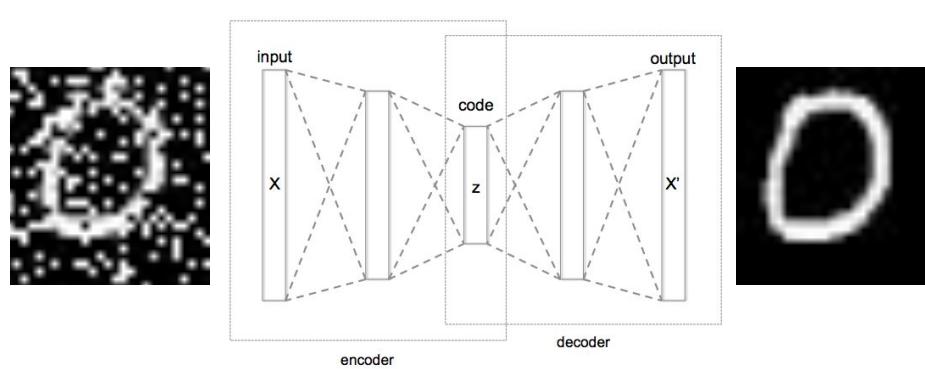
# Utilizing autoencoder for supervised tasks

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  $x \rightarrow x''$
  - Use  $x''$  as the input, and minimize the same loss
  - Loss =  $\|x-x'\|^2$



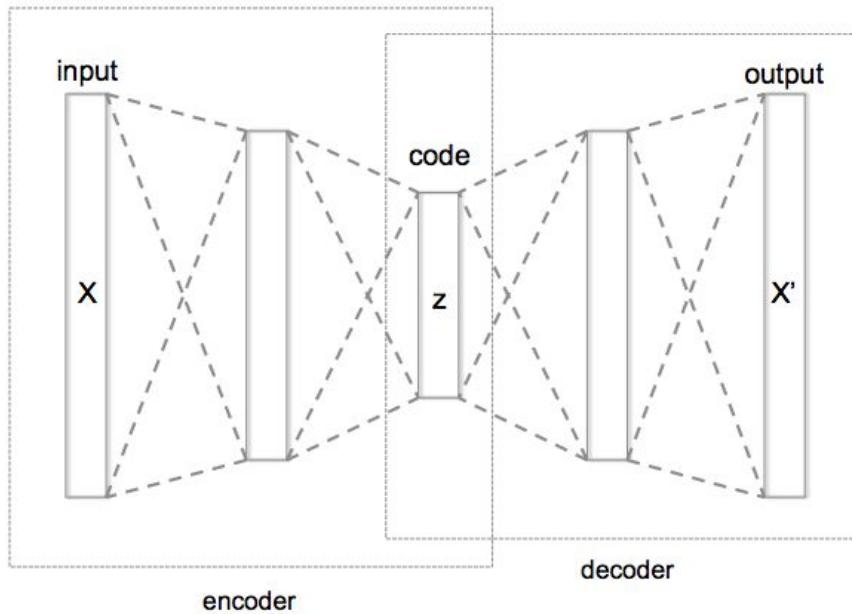
# Example

| Original  | Corrupted | Generated |
|-----------|-----------|-----------|
| 7 2 1 0 4 | 7 2 1 0 4 | 7 2 1 0 4 |
| 1 4 9 5 9 | 1 4 9 5 9 | 1 4 9 5 9 |
| 0 6 9 0 1 | 0 6 9 0 1 | 0 6 9 0 1 |
| 5 9 7 3 4 | 5 9 7 3 4 | 5 9 7 3 4 |
| 9 6 4 5 4 | 9 6 4 5 4 | 9 6 4 5 4 |

# 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



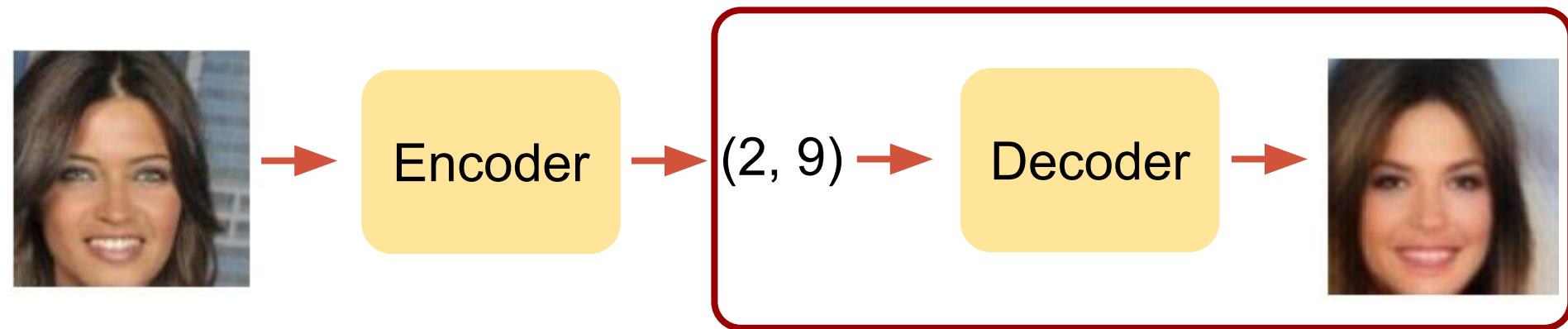
- Automatically learn features
- Non-linear dimensionality reduction / data compression

# Variational Autoencoder

Original and by-product motivations:

1. Allows approximation of intractable posterior.
2. Generate photos from the decoder

# Motivation #2 -- Generating Photos



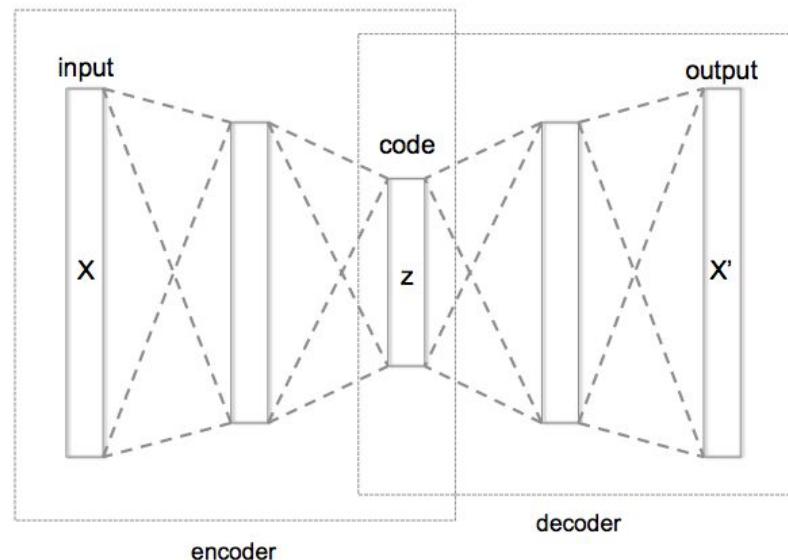
Use this part to generate an image from random code.

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.

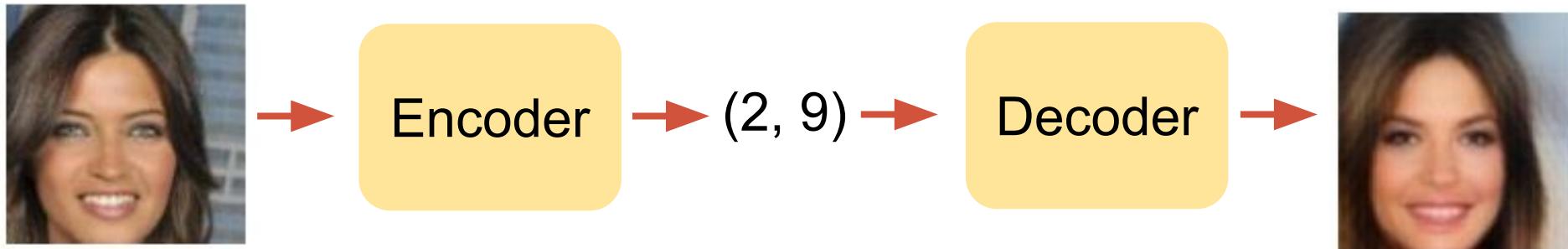
# Motivation #1 -- Solving Intractable Inference

Autoencoder can be viewed from a probabilistic perspective.

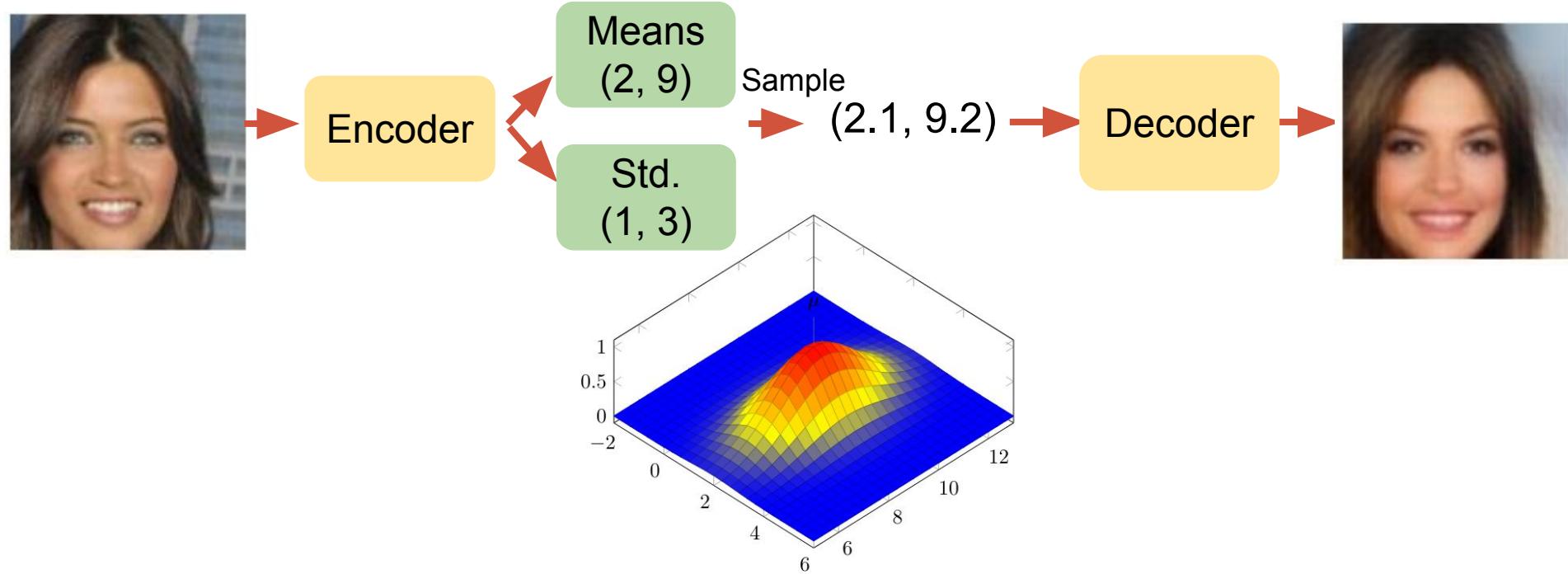


However, a standard AE isn't quite "probabilistic."  
I.e., the encoder deterministically outputs  $z$  given  $x$ . Rather we want a distribution of the code ( posterior  $p(z|x)$  ).

Standard AE



Probabilistic AE



# Autoencoding Variational Bayes: Bayesian Review

One important inference problem : Posterior inference

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

**x** Observed data

**z** Latent variable  
(hidden code)

# Bayesian Review

One important inference problem : Posterior inference

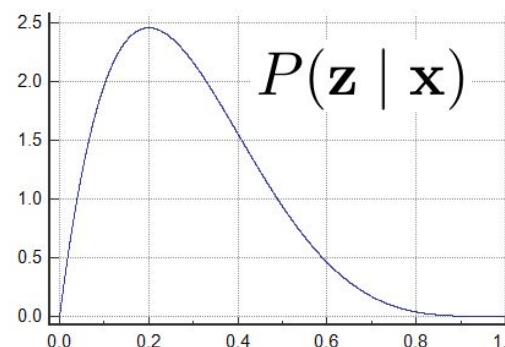
$$p(\mathbf{z} \mid \mathbf{x})$$

**x** Observed data

2 heads out of 10 coin flips  
(Binomial Dist)

**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

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

**x** Observed data



**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

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

Hard to compute because  $\mathbf{z}$  can be high-dimensional

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

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

$z_2$  : color, a real number

$z_3$  : ability to fly {yes, no}

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

# Bayesian Review

One important inference problem : Posterior inference

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

**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:**  $q^*(\mathbf{z} \mid \mathbf{x}) = \operatorname{argmin}_q \text{KL}[q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z} \mid \mathbf{x})]$

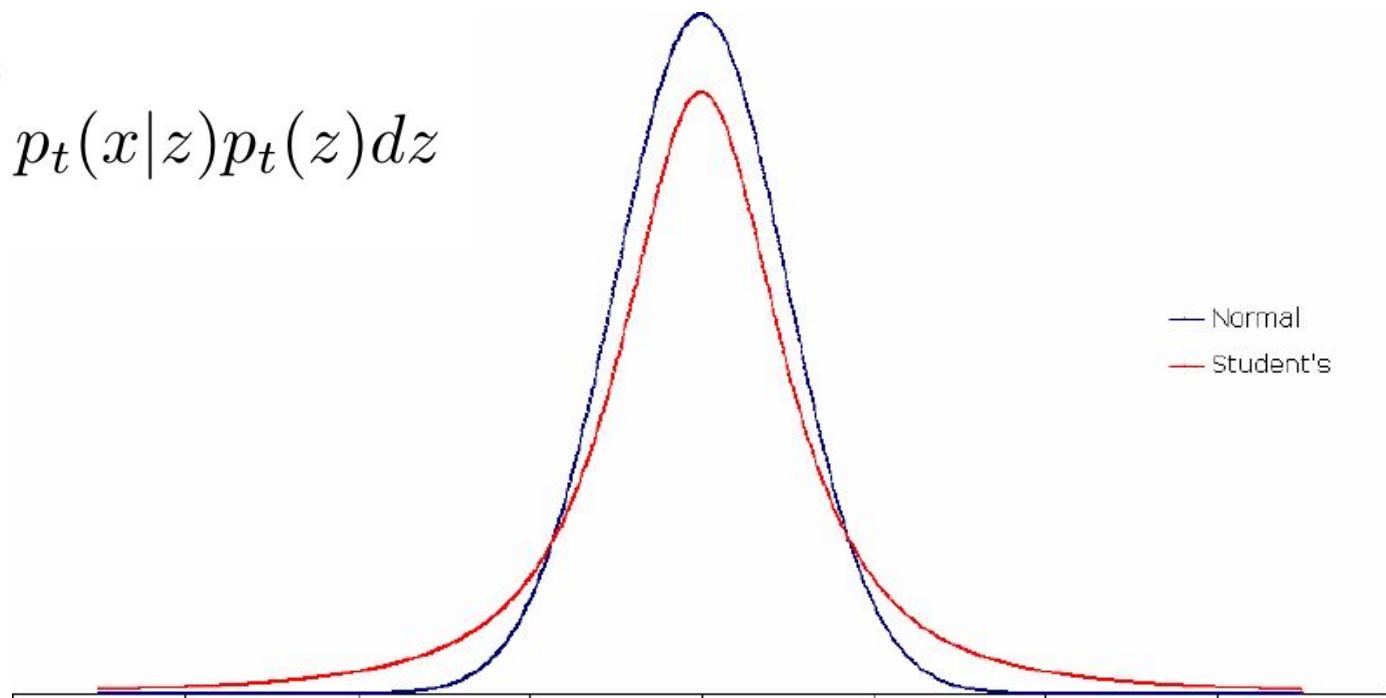
- But why not  $\text{KL}[p(\mathbf{z} \mid \mathbf{x}) \parallel q(\mathbf{z} \mid \mathbf{x})]$  or  $D(p(\mathbf{z} \mid \mathbf{x}), q(\mathbf{z} \mid \mathbf{x}))$ ?
- How is this related to autoencoder?

# Digression: Model evidence

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

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

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



# 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  $\text{KL}(q \parallel p)$ ):

$$q^*(\mathbf{z} \mid \mathbf{x}) = \operatorname{argmin}_q \text{KL}[q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z} \mid \mathbf{x})]$$

Corresponds to maximizing the evidence lower bound (ELBO)!

## Proof

This KL is hard to compute



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

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

(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  $P(z \mid x)$ ,  $\mathbb{E}_q[\log p(x)] = \log p(x)$ )

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

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

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

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

# Proof

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

Let's interpret this new term:

From previous slide, we have:

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

We know  $\text{KL} \geq 0$ , 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)

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

But how to compute this ELBO term then?

# Maximizing ELBO

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


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

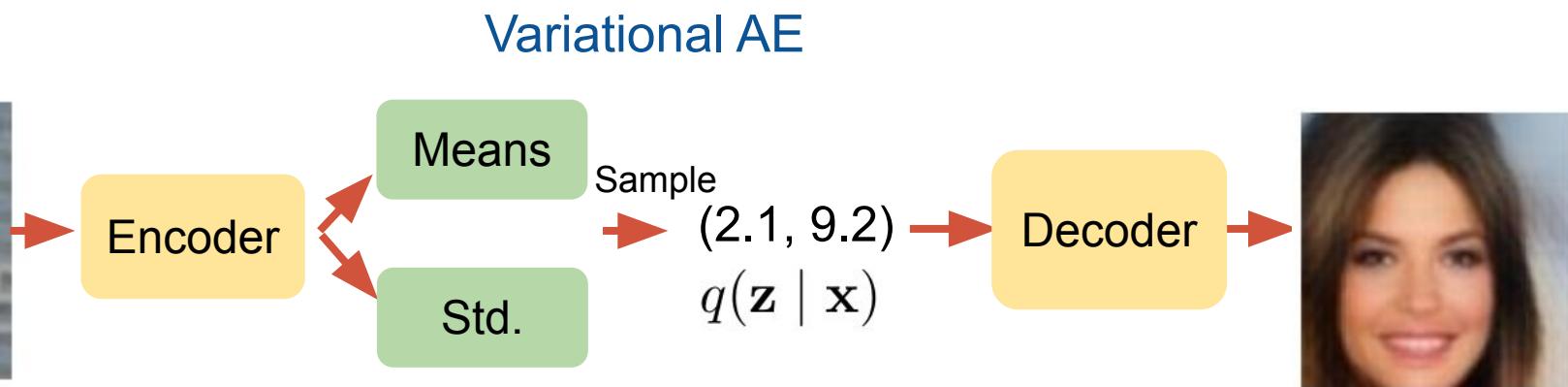
$$\iff \min_q \mathbb{E}_q[\log q(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{z}, \mathbf{x})]$$

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

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

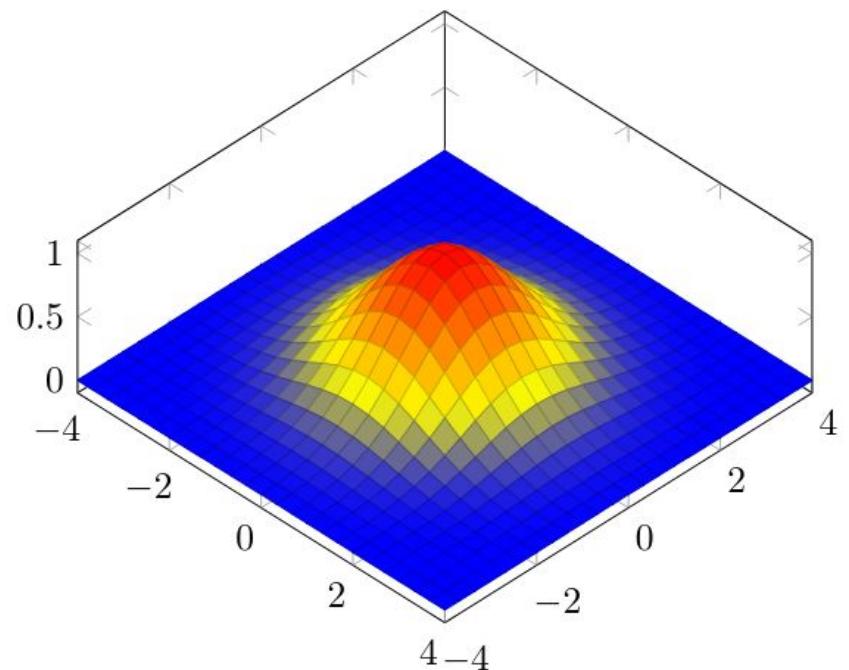
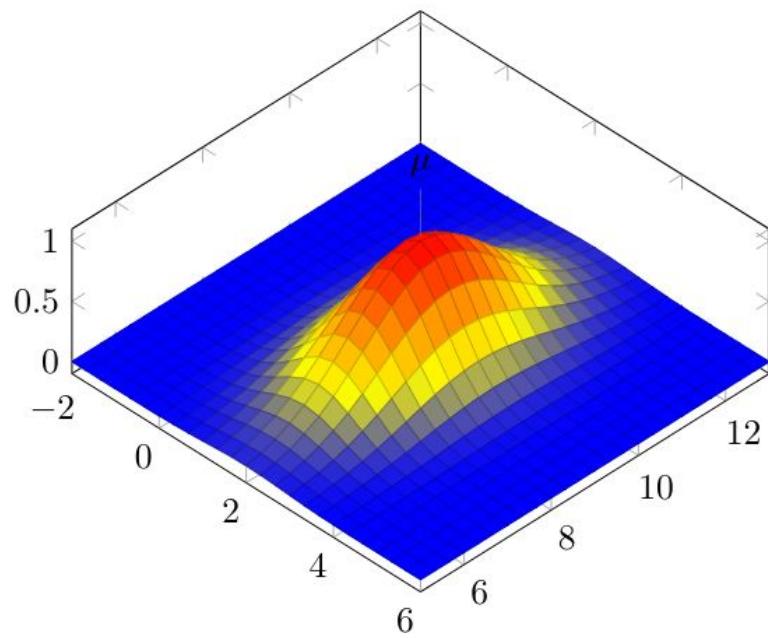
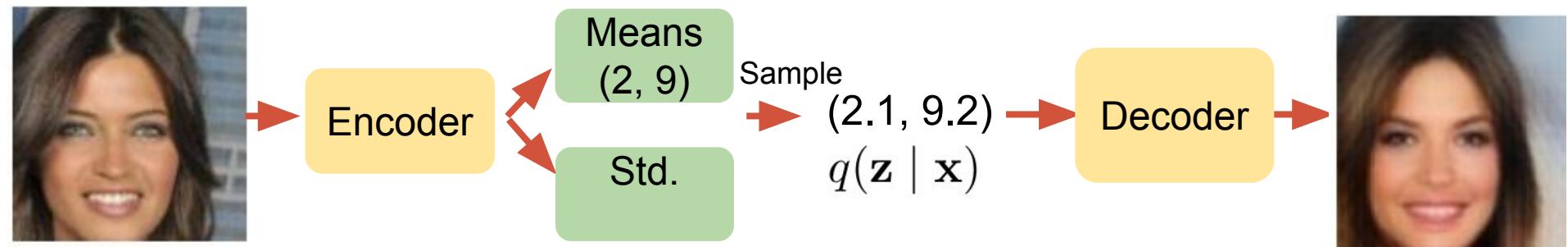


We're free to pick



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

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



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

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

What is this second term?

# Second Term: Data Log Likelihood

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

If we also model  $p(\mathbf{x} \mid \mathbf{z})$  as a Gaussian distribution:

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

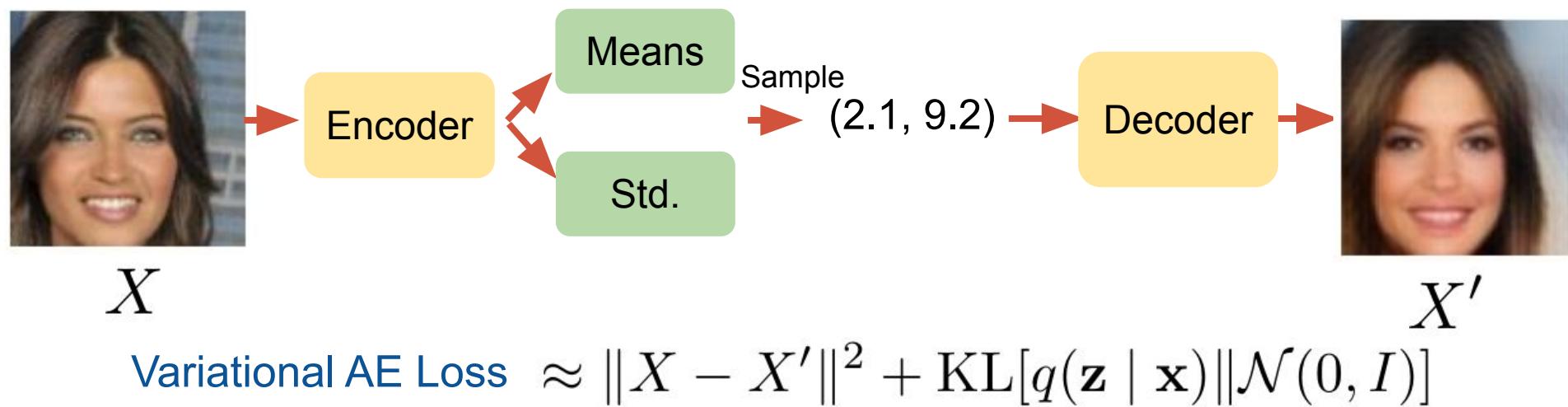
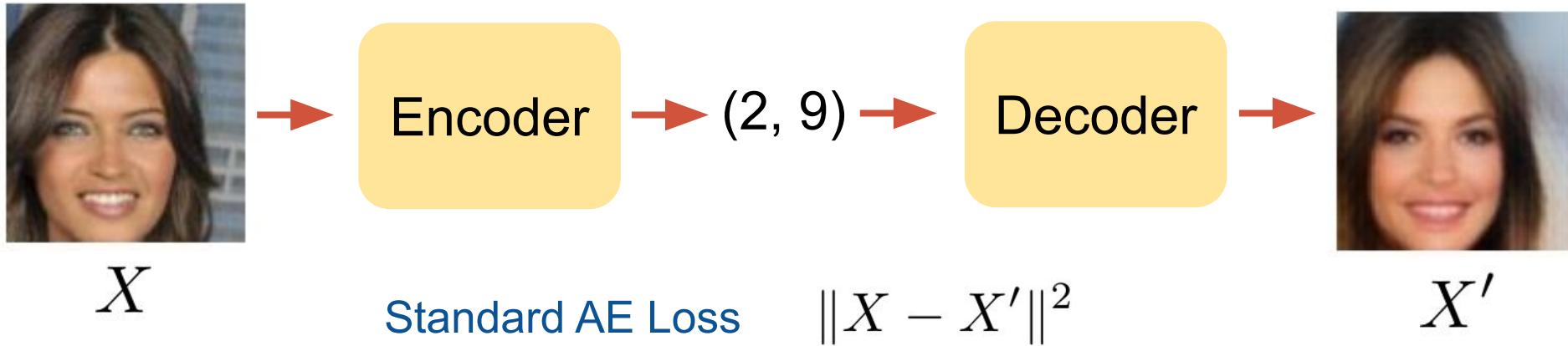
(Let's consider a single pixel case  
u is the output from the decoder  
x is the ground-truth pixel)

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

$$-\log[\exp(-(x-u)^2)] = (x-u)^2 \quad (\text{with constants hidden})$$

Becomes the standard L2 error

Yet another reason why assuming Gaussian is nice



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

# Last Problem

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

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

**Example:**

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

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

# Reparameterization Trick

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

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

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

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

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

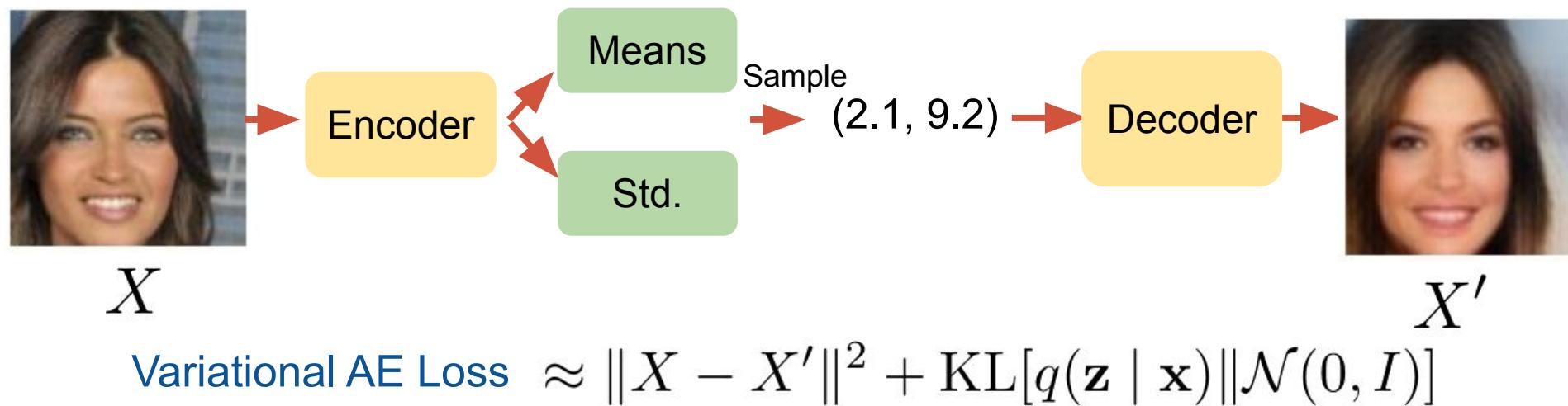
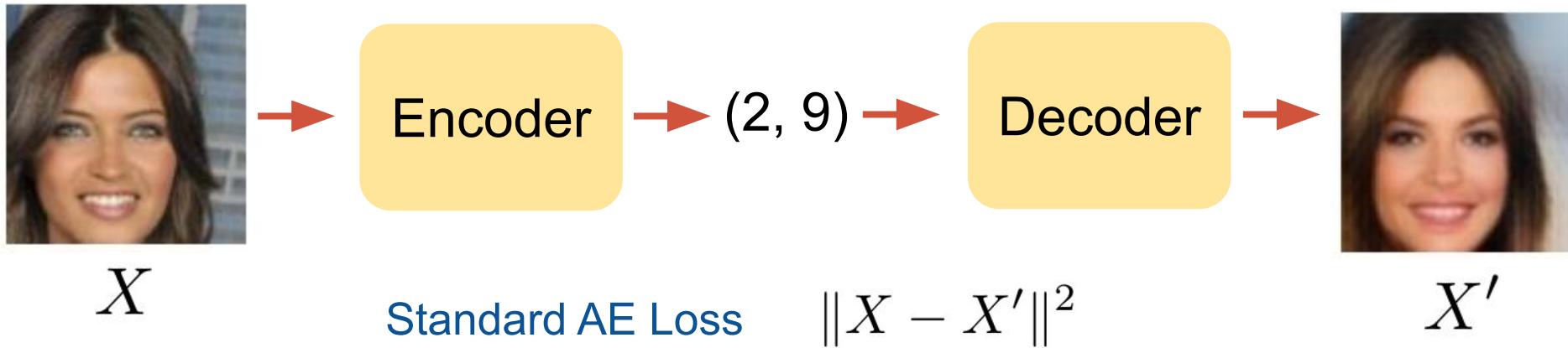
$$\frac{\partial z}{\partial \mu} = 1$$

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

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

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

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



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

# Disentangling Representation: $\beta$ -VAE

We want compact meaningful representation.

But also disentangled representation

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

Skin color

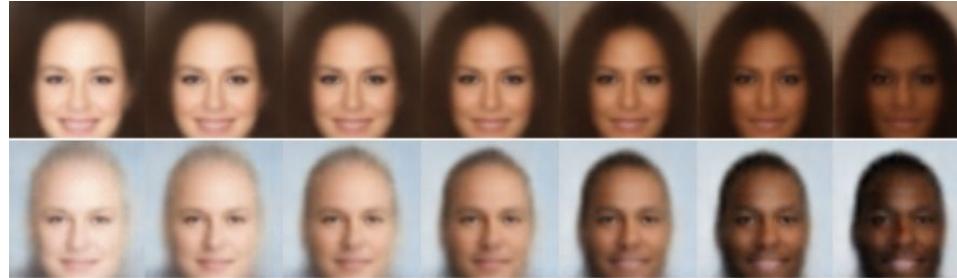
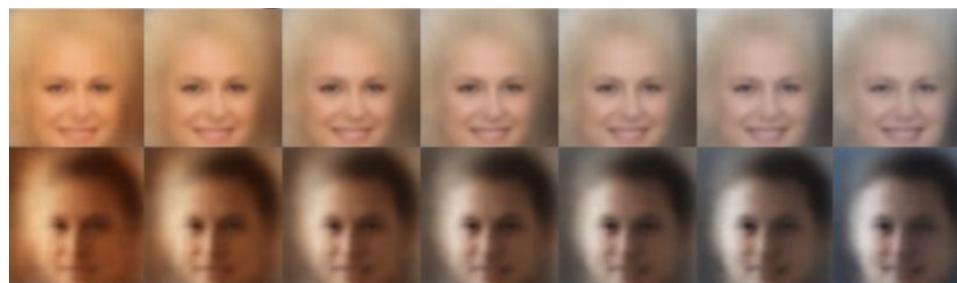
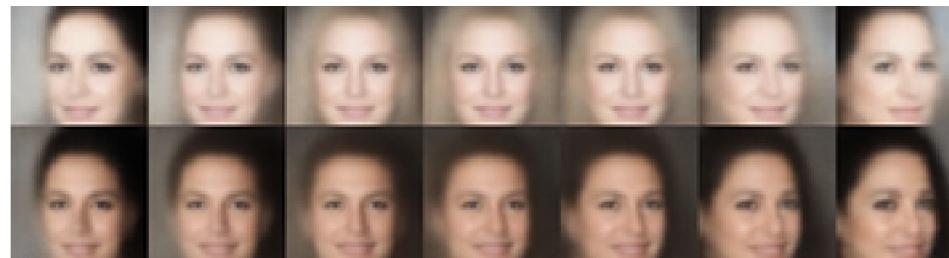


Image  
Saturation



Head  
Angle

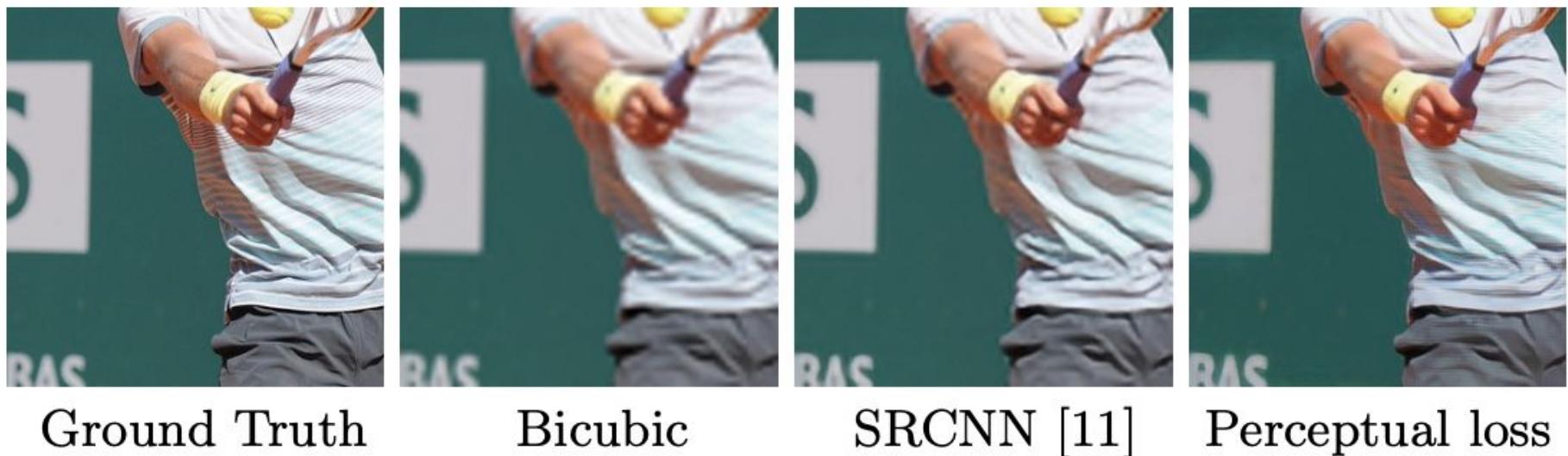


Age / Gender



# Perceptual loss

Typical Autoencoder uses per pixel loss (for example L2)  
Might not capture the important content



# 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

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

# Unsupervised Learning

- Autoencoders
  - Denoising Autoencoder
  - VAE
    - Evidence lower bound
    - Perceptual loss
- Generative Adversarial Networks
  - Random Cool stuff
- Clustering

# Generating Images

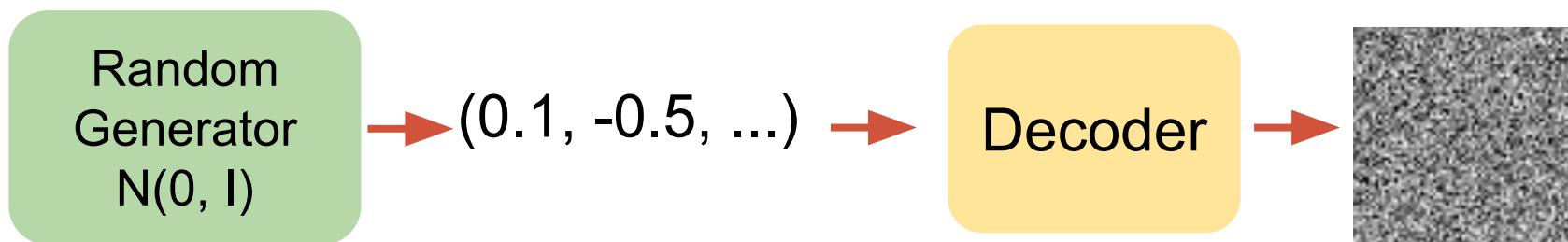
VAE can be used to generate images by feeding numbers into the decoder.

Are there other ways?

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

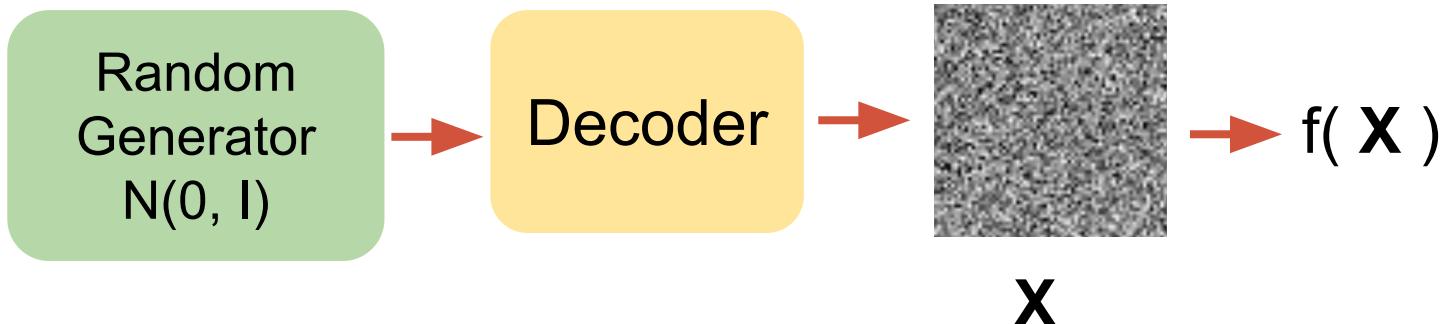
Maybe let's start with a decoder, because we need something that outputs an image anyway.

# Generating Realistic Faces



**What kind of loss should we use?**

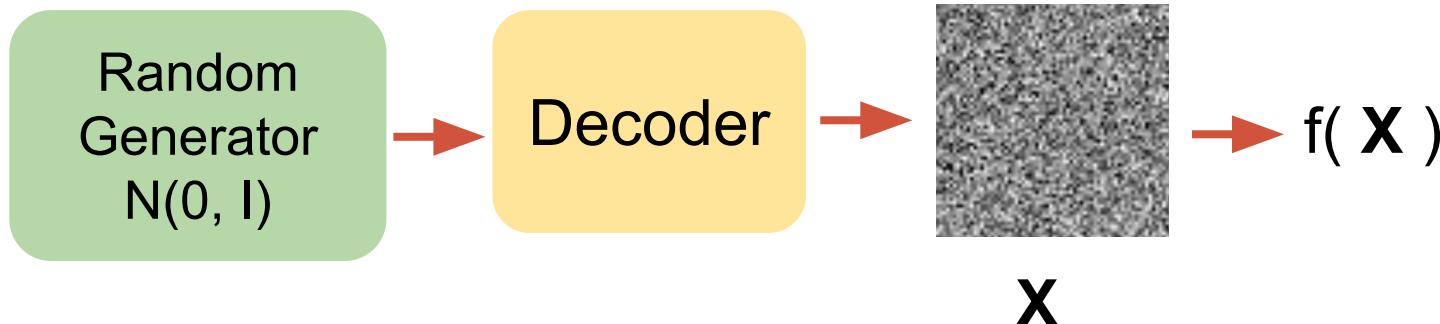
# Generating Realistic Faces



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:

$$f(\text{Image of Face}) = 1 \quad f(\text{Image of Non-Face}) = 0.2 \quad f(\text{Noise Image}) = 0$$

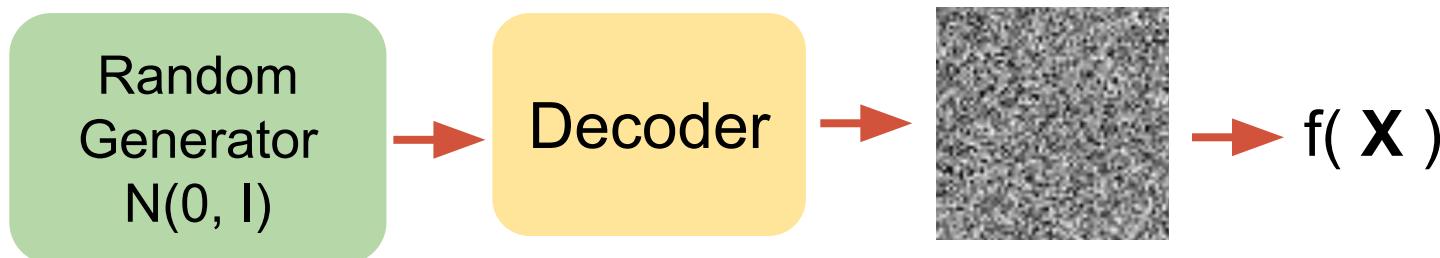
# Generating Realistic Faces



$$f(\text{Image of Face}) = 1 \quad f(\text{Image of Tap}) = 0.2 \quad f(\text{Image of Noise}) = 0$$

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

# Generating Realistic Faces



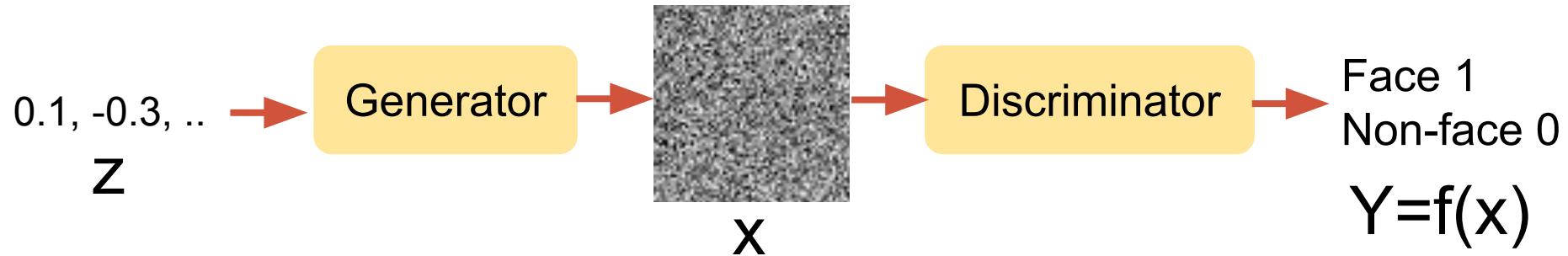
**Solution:** We will use another network as  $f$ . This network will predict the probability of  $x$  being a face.

Training this  $f$  is a simple supervised learning!

All face photos -> positive examples

All non-face photos -> negative examples

# Generating Realistic Faces

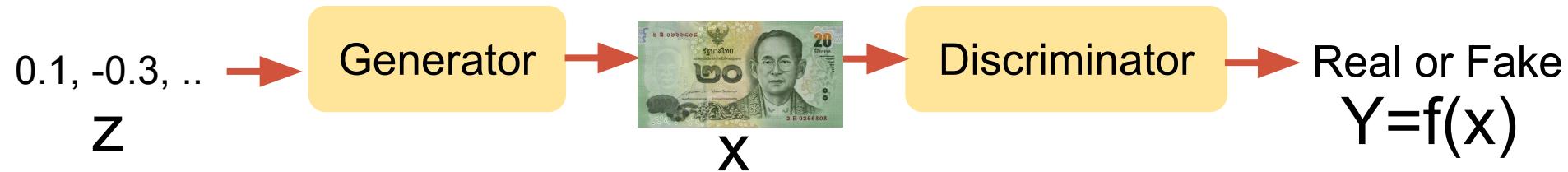


# Generative Adversarial Networks (GAN)



- 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 is try to discriminate between fake and real inputs. – Discriminator network

# Generative Adversarial Networks (GAN)



- Generator (Money Faker):

- Maximize  $Y$

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

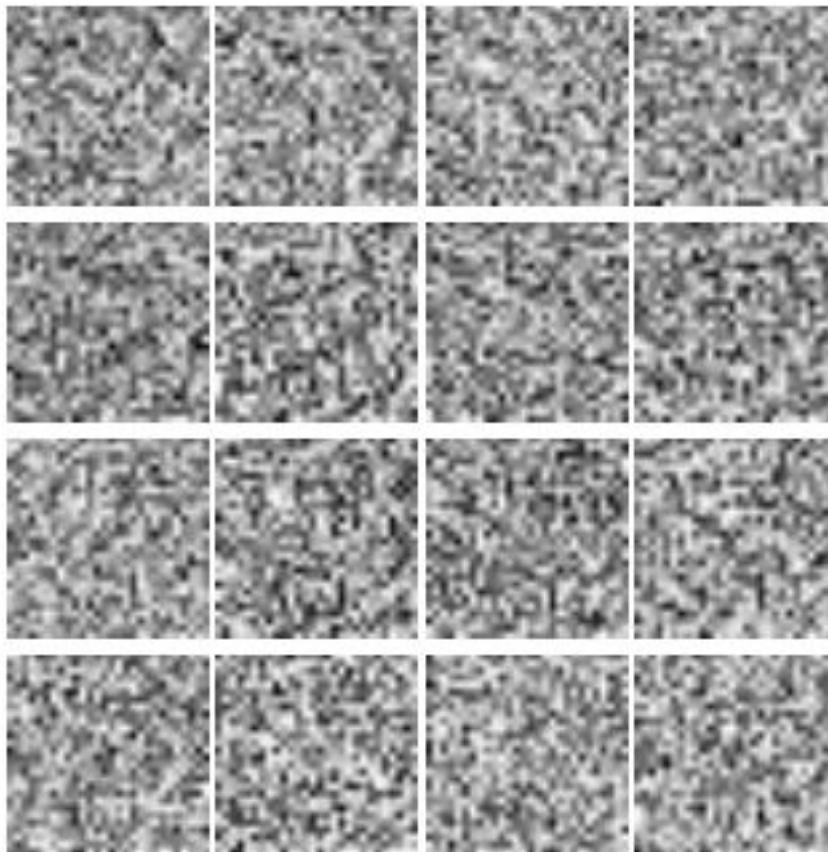
- Discriminator (Police):

- For real images => Maximize  $Y$
  - For generated images from the faker => Minimize  $Y$

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

# GAN example

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



# Progressive GANS



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

# GANs Loss Formulations

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

Discriminator

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

Generator

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

Another problem: Mode collapsing

# Mode collapse

Model only learn a couple types (modes) of inputs

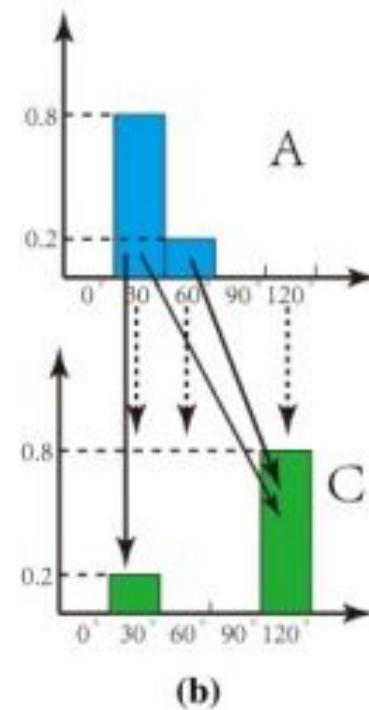
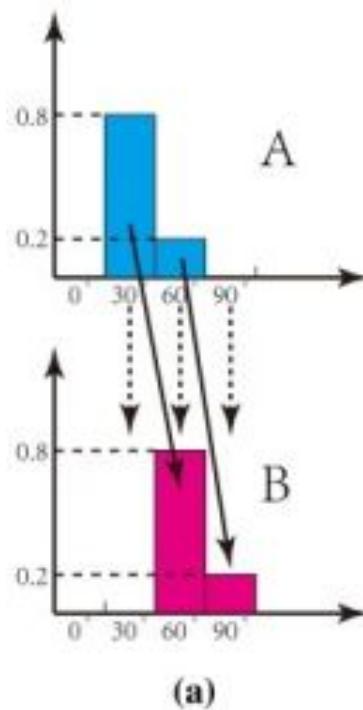


<https://arxiv.org/pdf/1701.07875.pdf>

# Wasserstein GAN (WGAN)

Wasserstein distance? (Earth mover distance)

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



# WGAN

Discriminator to a critic (no fake/real sigmoid) but output a score

WD has better gradient and convergence

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

# WGAN

WGAN 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

# 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-Lipschitz if and only if it has gradients with norm at most 1 everywhere.

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

**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,  $D$ : DCGANNo 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  $D$ 

# Other uses

- StackGAN: text to photo

(a) StackGAN  
Stage-I  
64x64  
images

This bird is white with some black on its head and wings, and has a long orange beak

This bird has a yellow belly and tarsus, grey back, wings, and brown throat, nape with a black face

This flower has overlapping pink pointed petals surrounding a ring of short yellow filaments

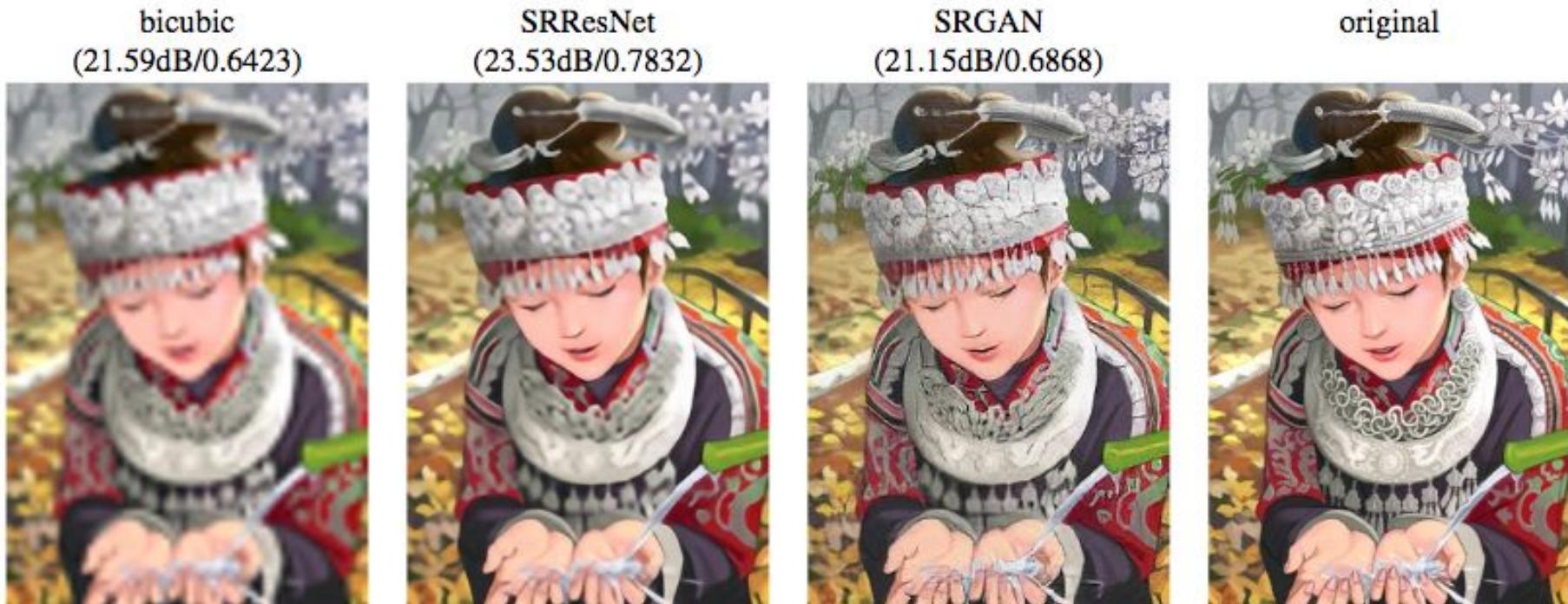


(b) StackGAN  
Stage-II  
256x256  
images

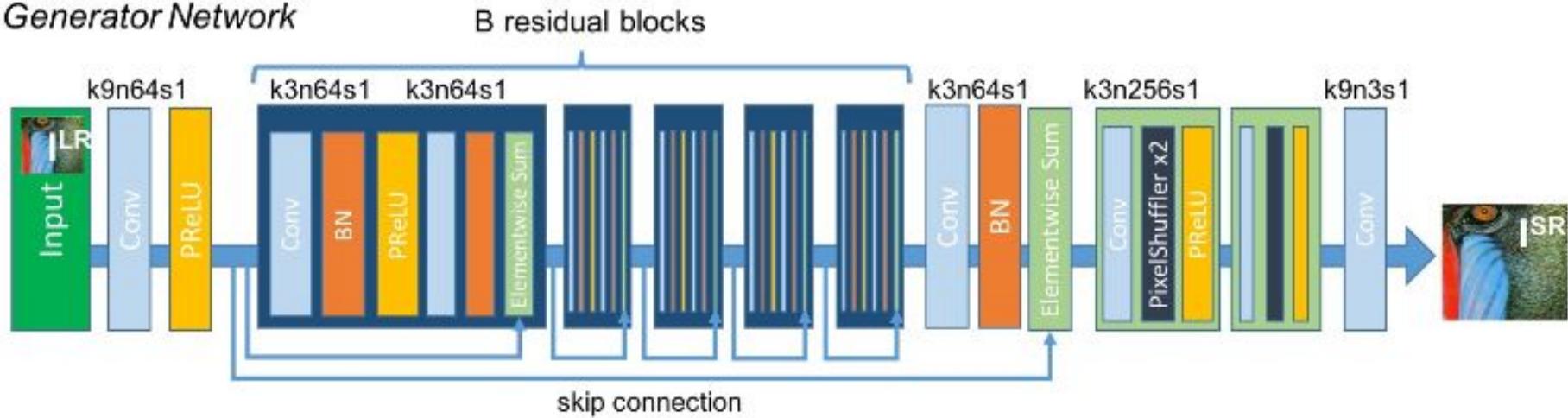
(c) Vanilla GAN  
256x256  
images

# SRGAN

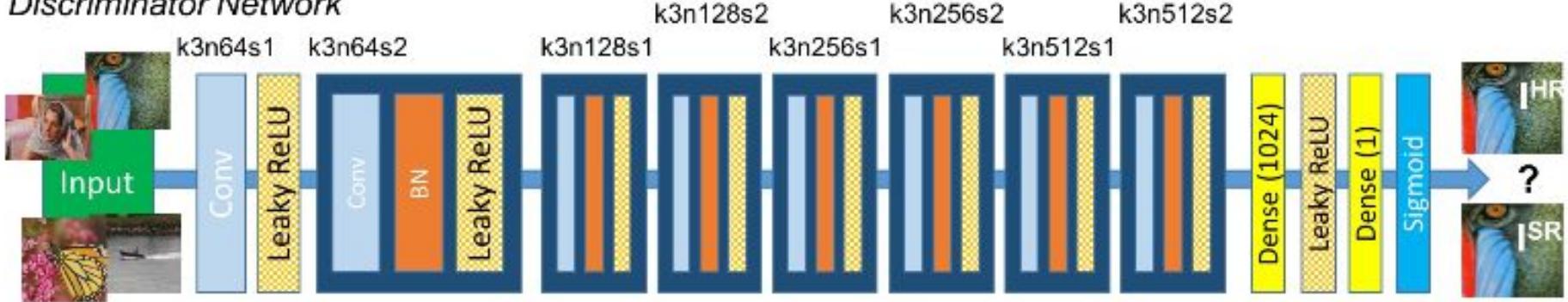
- <https://arxiv.org/pdf/1609.04802.pdf>



## Generator Network



## Discriminator Network

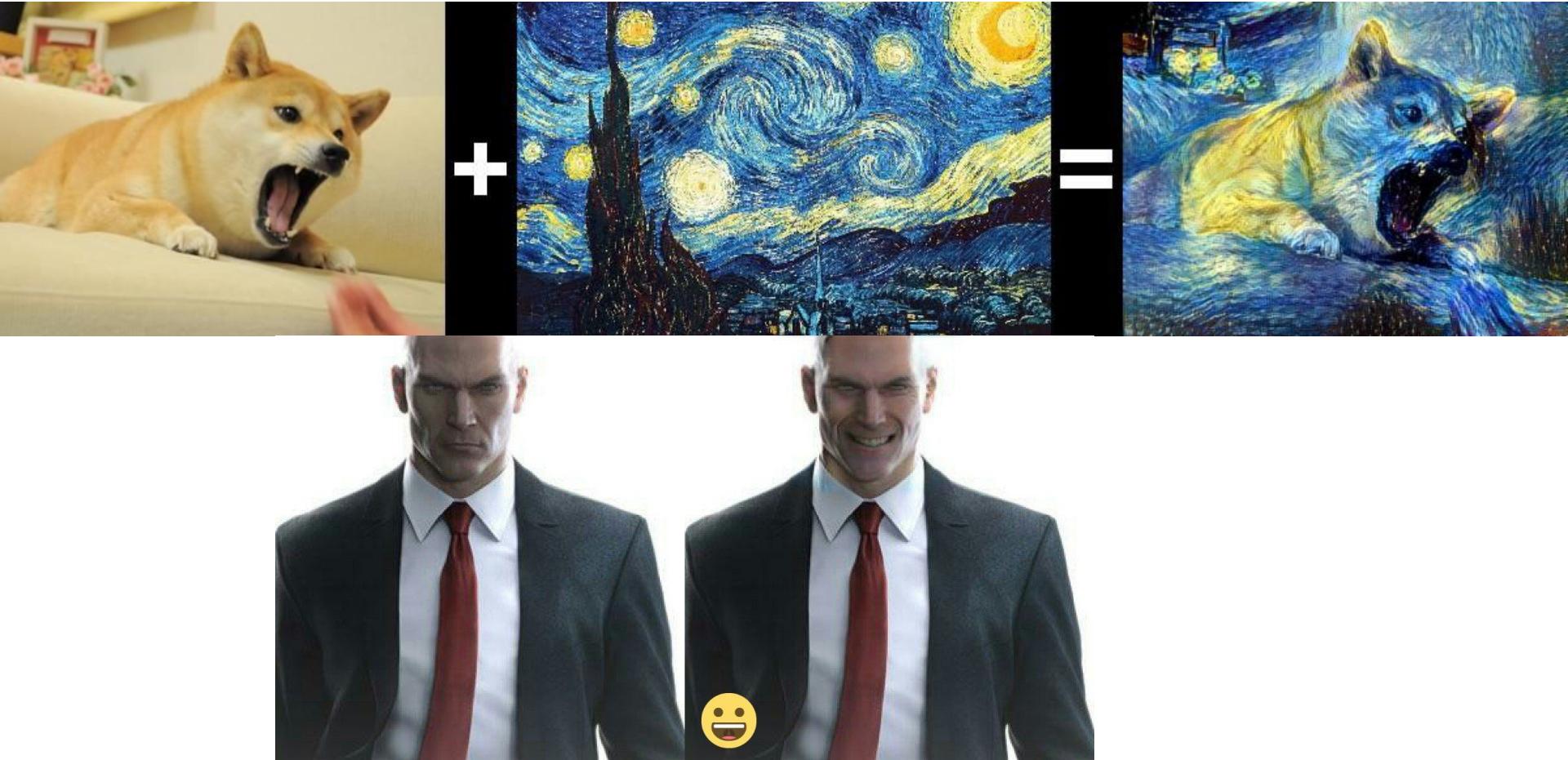


# Content-aware Filling



Globally and Locally Consistent Image Completion [Iizuka et al., 2017]

# Style transfer with cycleGAN



Zebras  Horses



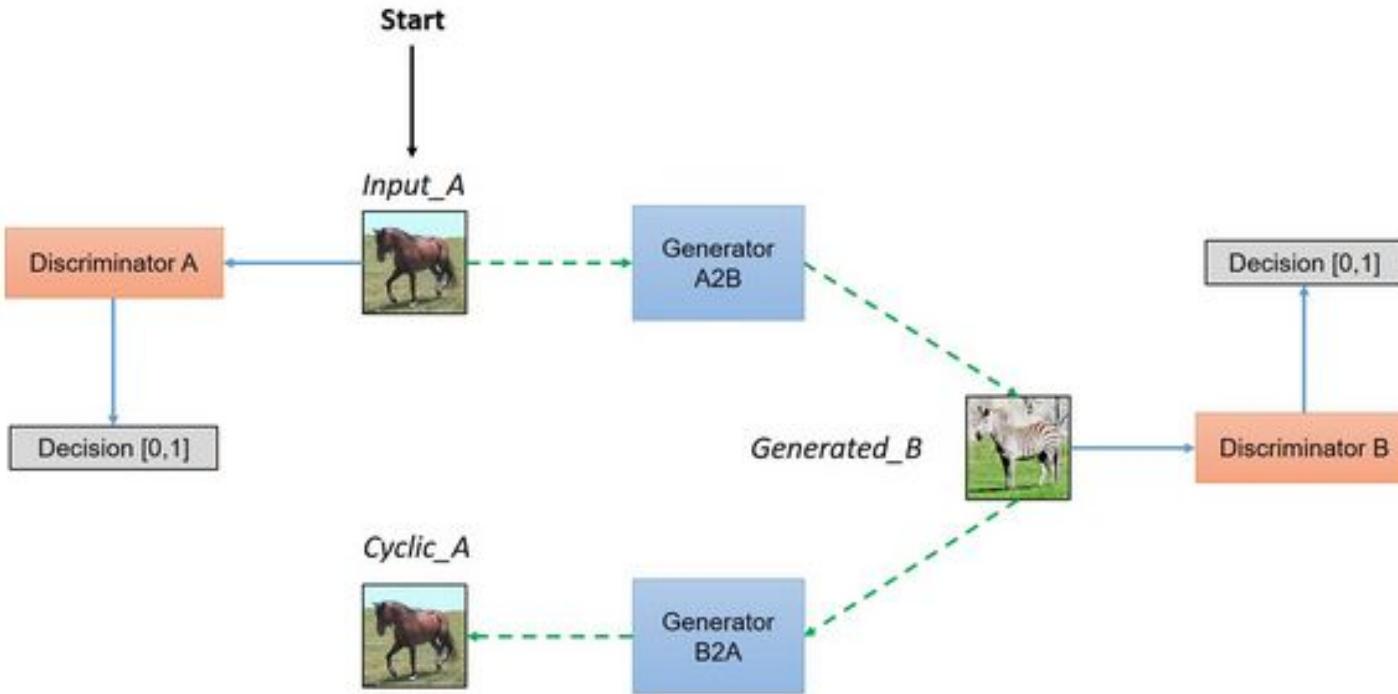
zebra → horse



horse → zebra

Cycle consistency

Note, you don't  
need a paired  
dataset



Cycle consistency

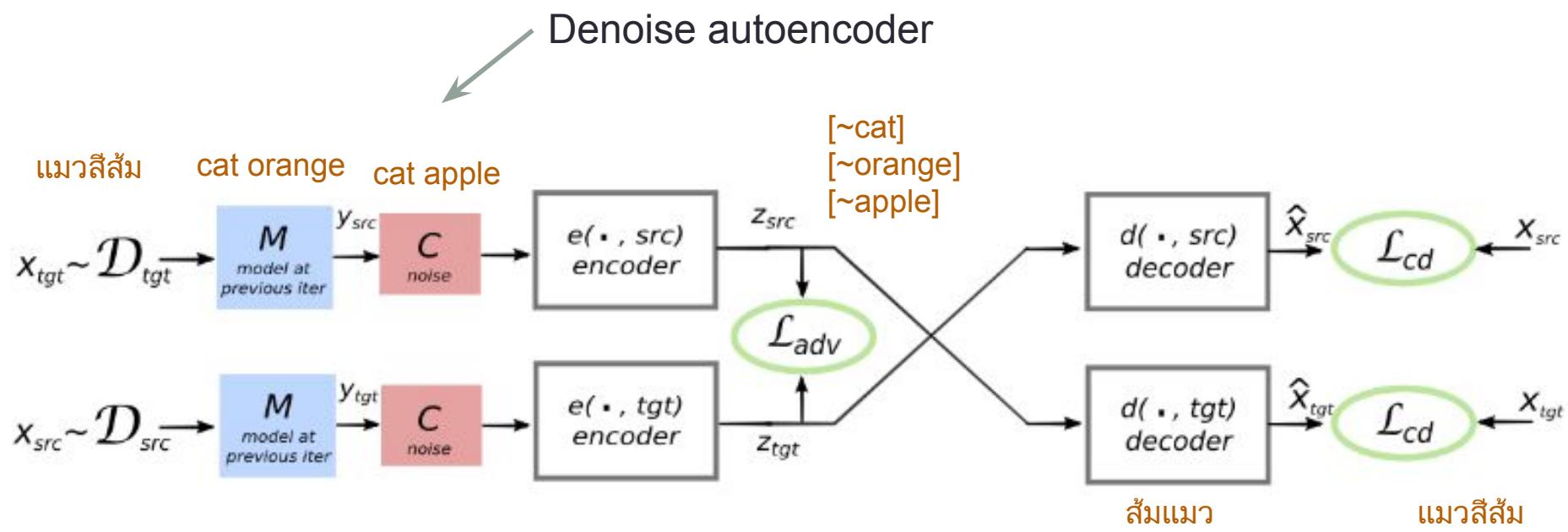
# Machine translation with no parallel text

- MT usually requires parallel text

This is a cat

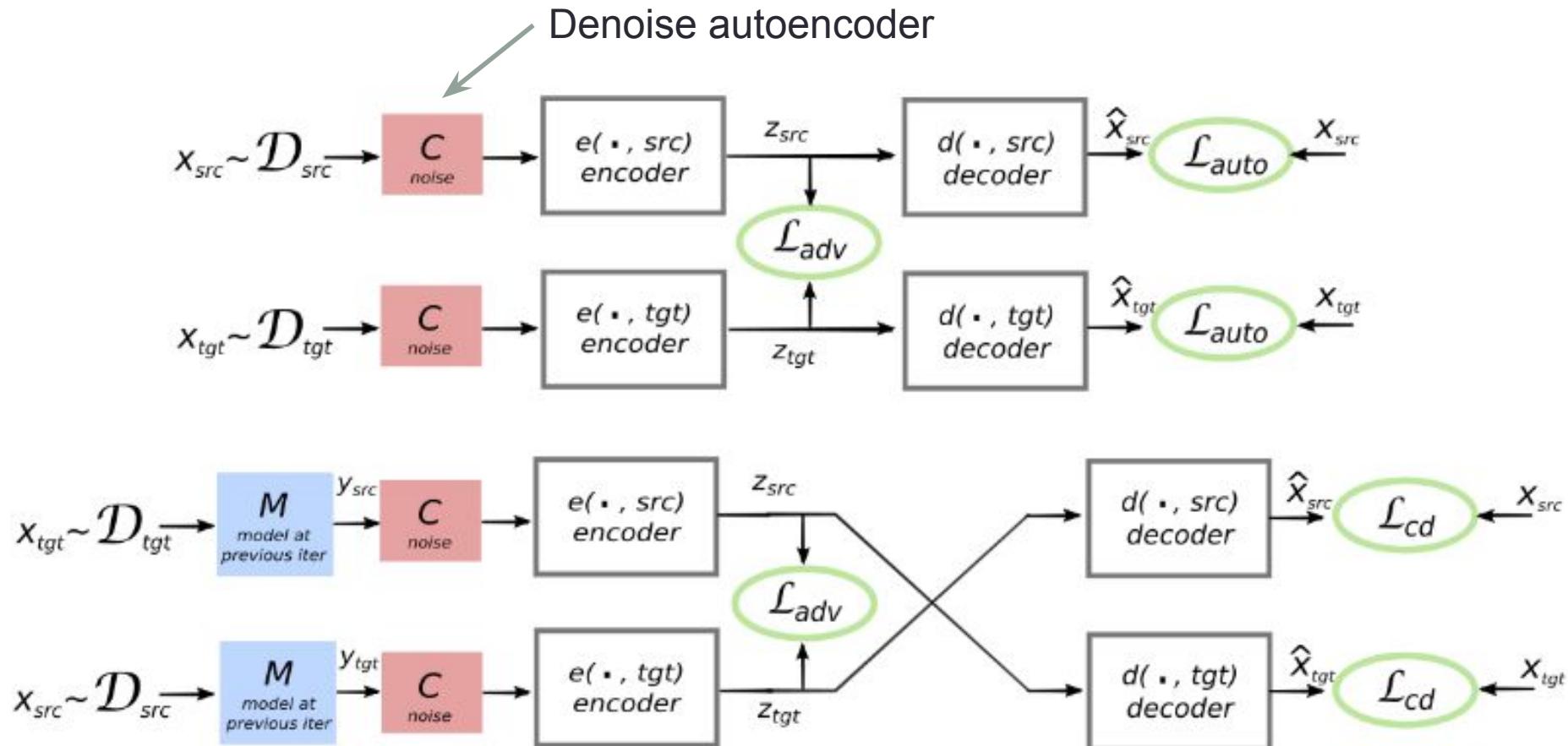
นี่คือแมว

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



Use GAN Loss ( $\mathcal{L}_{adv}$ ) to enforce that source and target language pairs share the same distributions.

Then enforce the translation distribution matches.

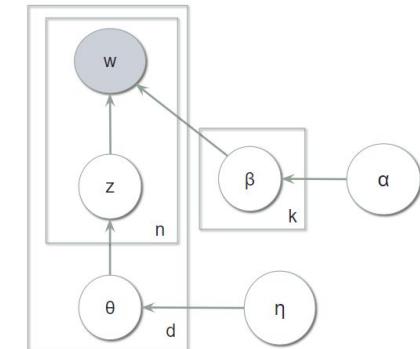
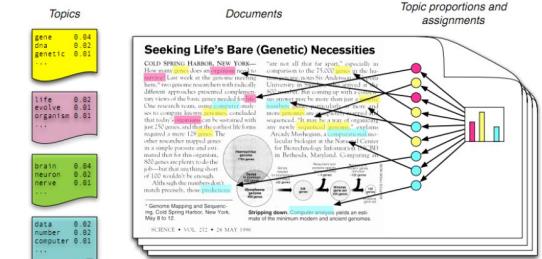
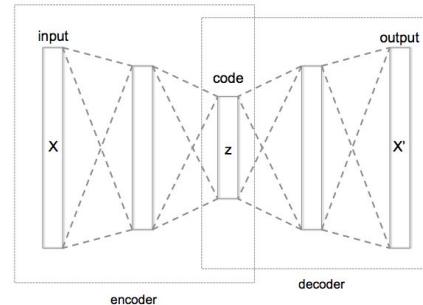


# Results

|                          | 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  |

# Unsupervised Learning Summary

- Clustering
  - Introduction to probabilistic graphical models
  - LDA
- Autoencoders
  - Denoising Autoencoder
  - VAE
    - Evidence lower bound
- Generative Adversarial Networks
- Random Cool stuff



# Reminder

Next class: 3-5 minutes project update presentation

Scope: Topic, methodology (data, train/test, etc.) and metrics, features, initial model, initial results

Goal: At least 1 number