On Generative Models

Alexander G. Ororbia II

Neural Networks & Machine Learning

CSCI-736

2/18/2025

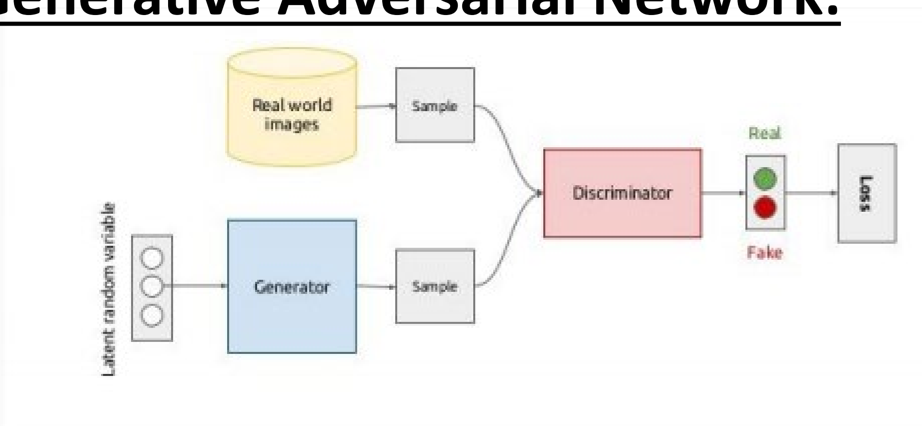
Companion reading:

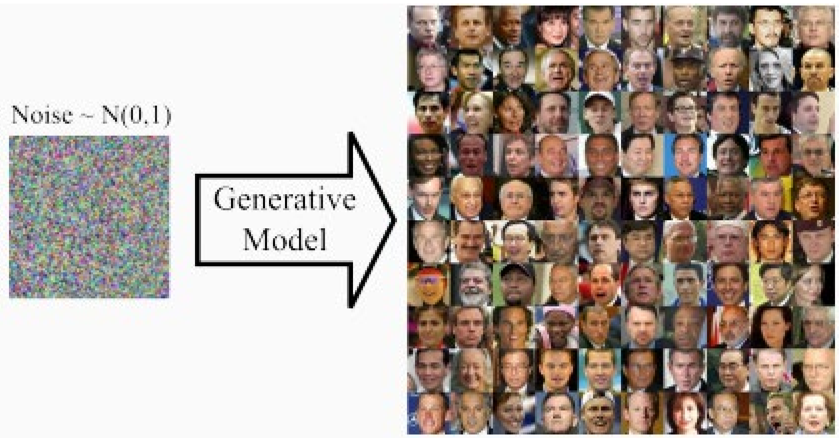
Chapters 14, 16, 19, & 20 of Deep Learning textbook

Generative Modeling & Sampling

- **Task:** Given a dataset of images {X1,X2...} can we learn the distribution of X?
- Typically generative models implies modelling P(X).
  - **Very limited, given an image the model outputs a probability**
- **More Interested** in models which we can **sample** from.
  - Can generate random examples that follow the distribution of P(X).

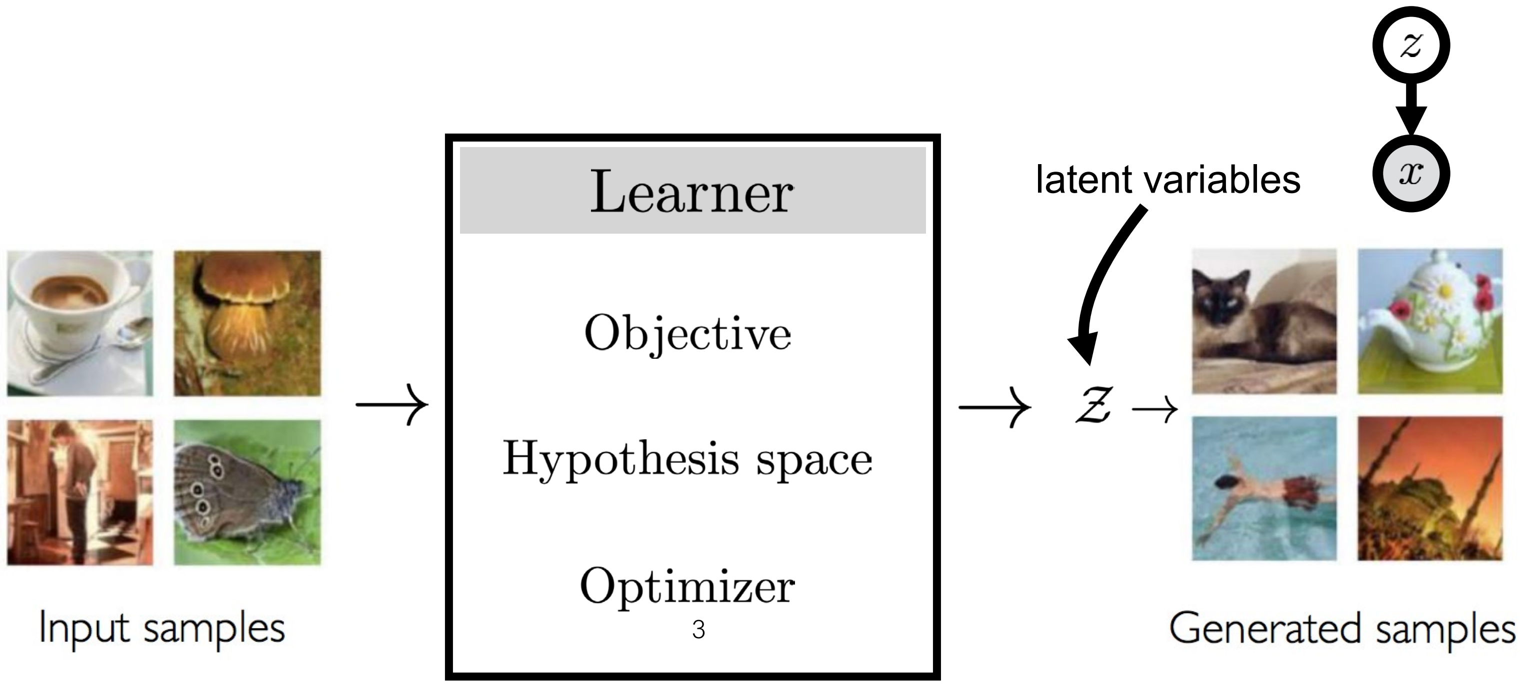
Generative Adversarial Network:





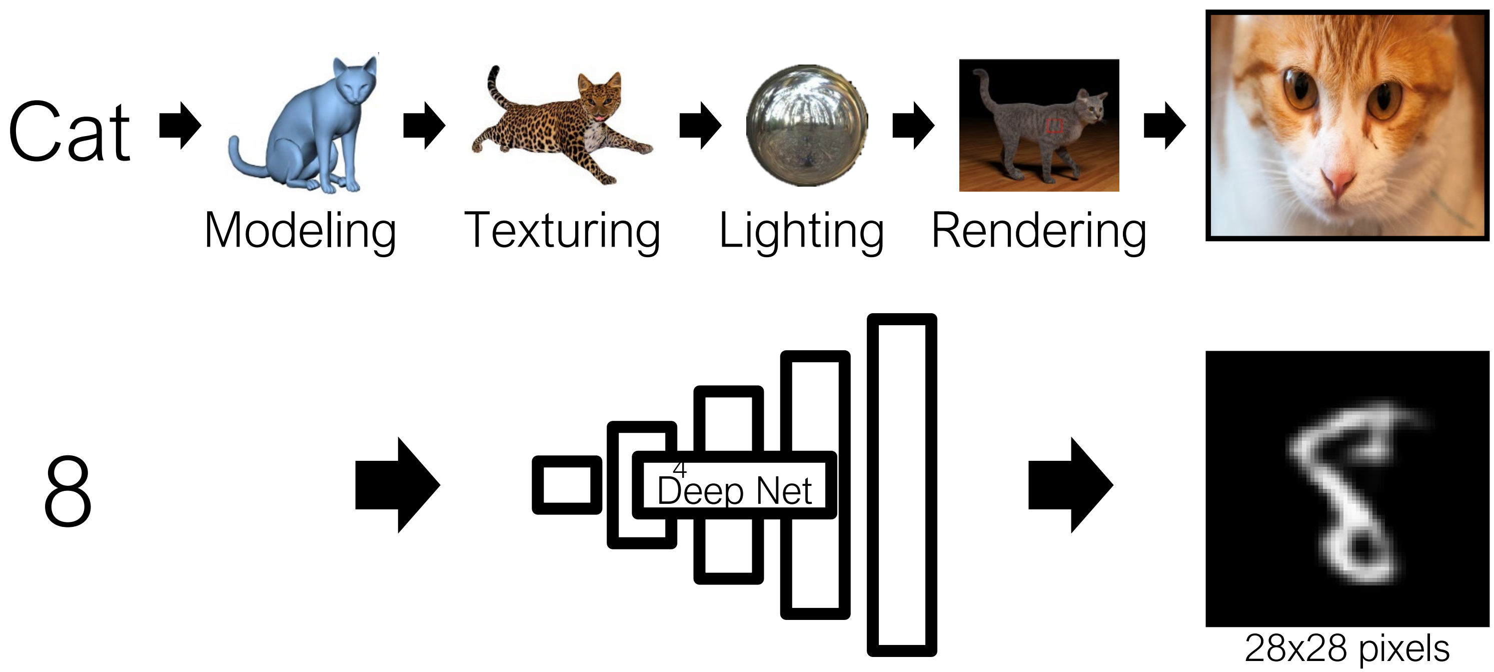
- Pro: Do not have to explicitly specify a form on P(X|z), z is the latent space.
- Con: Given a desired image, difficult to map back to the latent variable.

Thinking about Learning a Generative Model

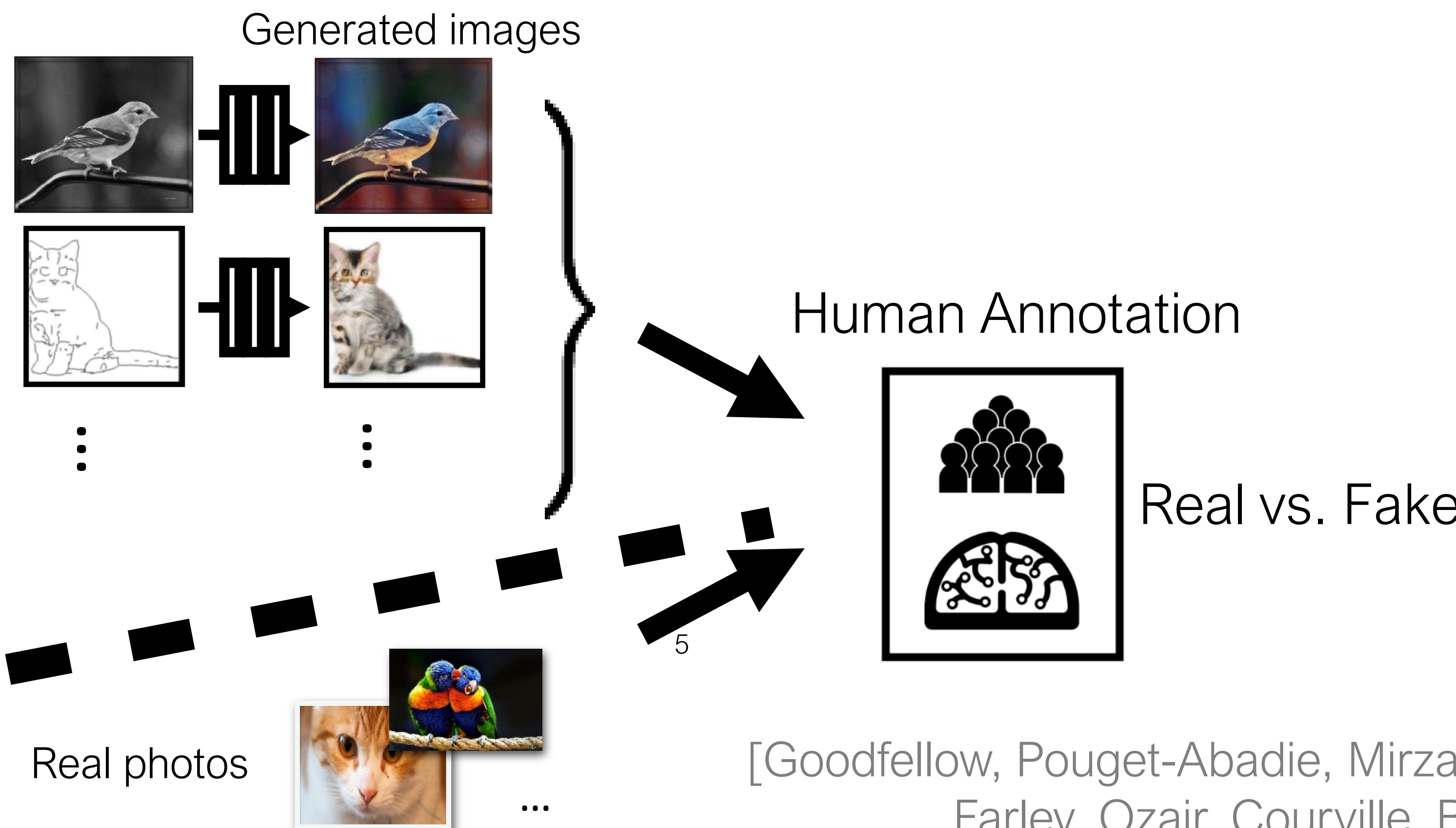


[figs modified from: http://introtodeeplearning.com/materials/2019_6S191_L4.pdf]

Generating Images is Hard!



Learning Inverse Graphics



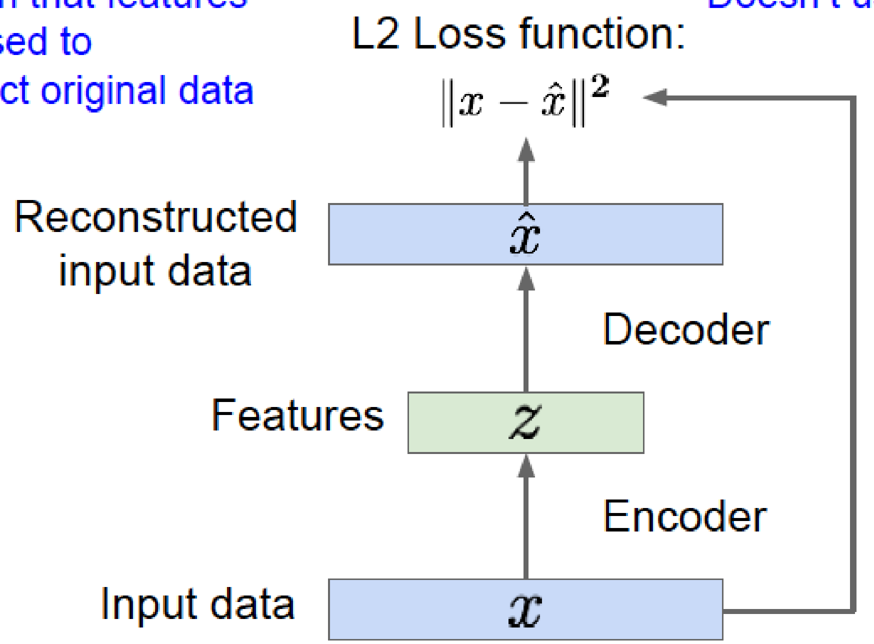
[Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, Bengio 2014]

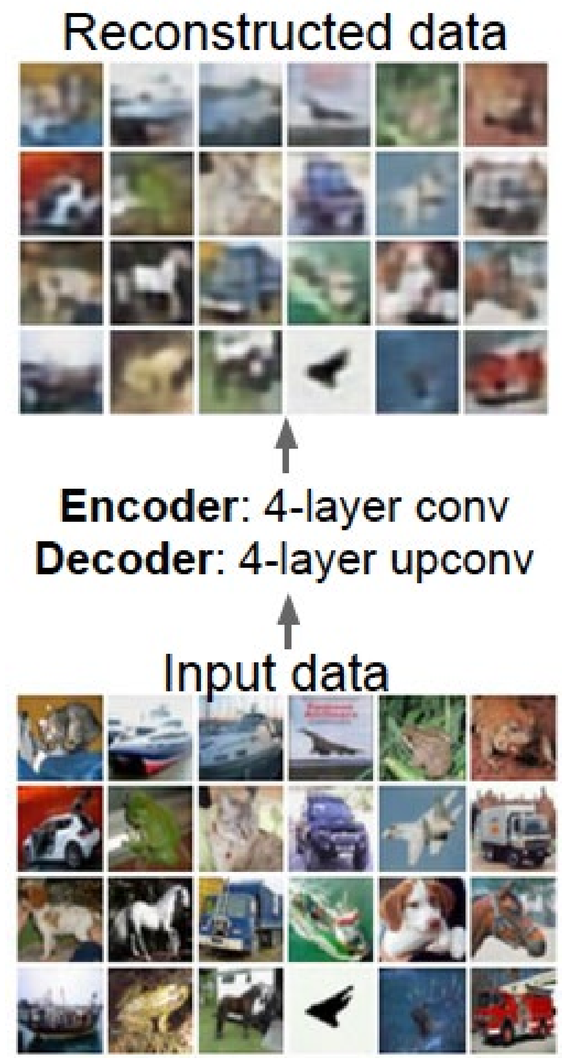
[Zhu et al. 2014]

Concept: Auto-association

Train such that features can be used to reconstruct original data

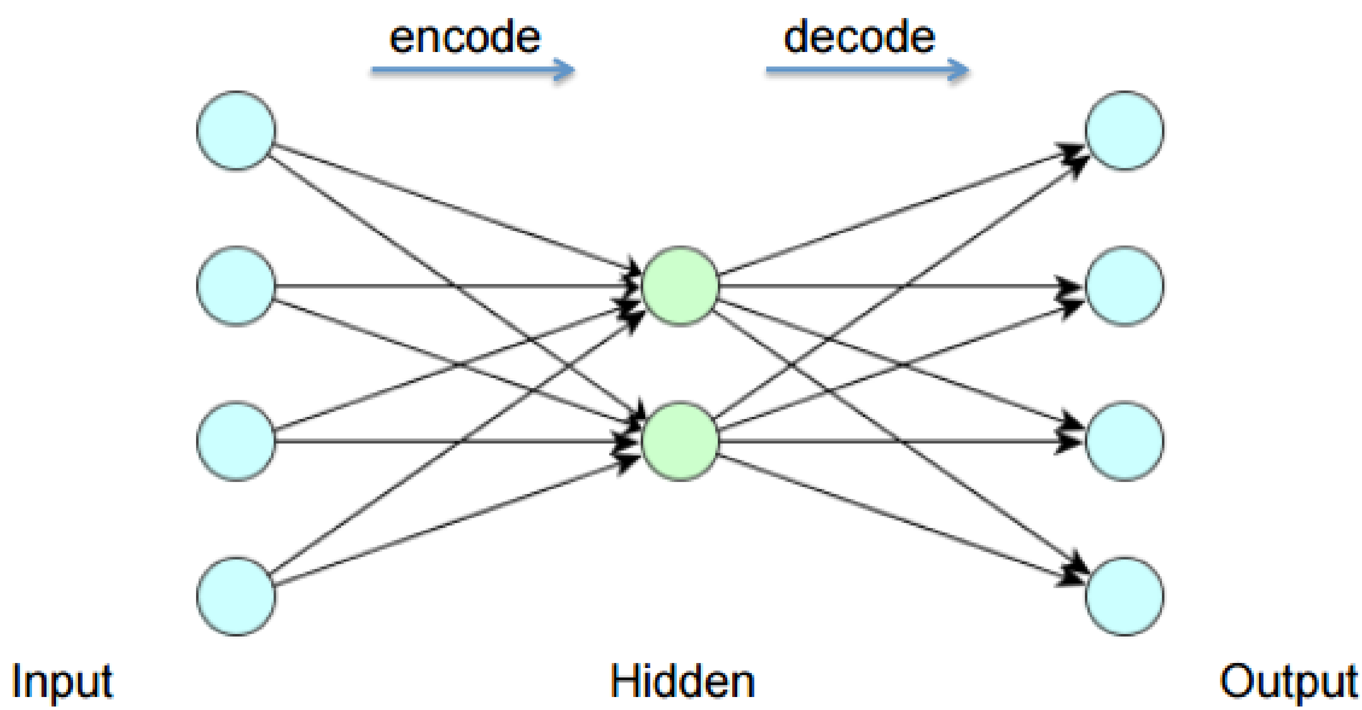
Doesn't use labels!





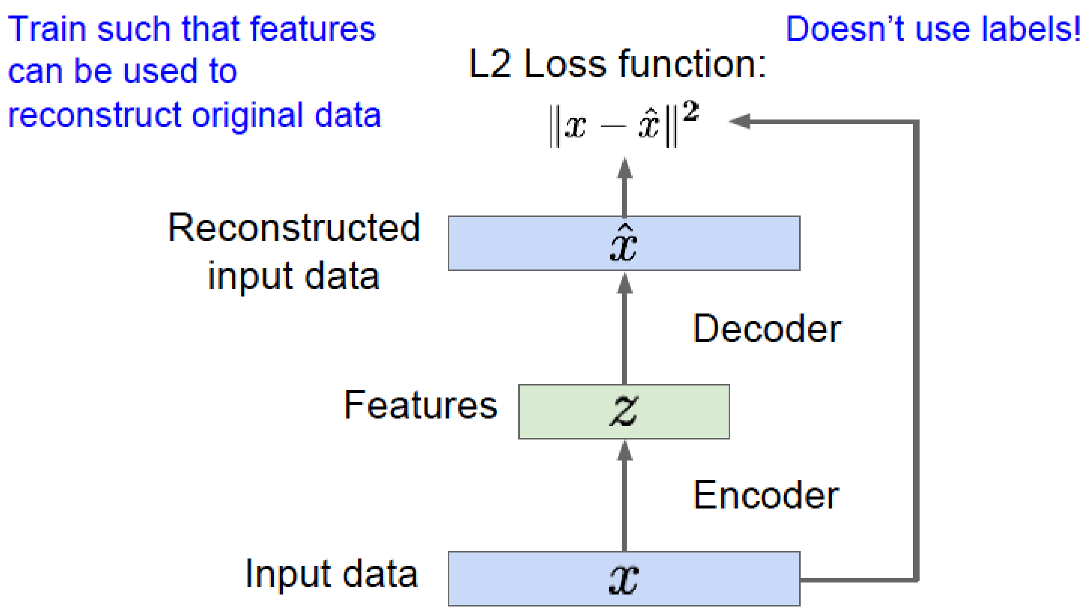
The Encoder-Decoder Framework

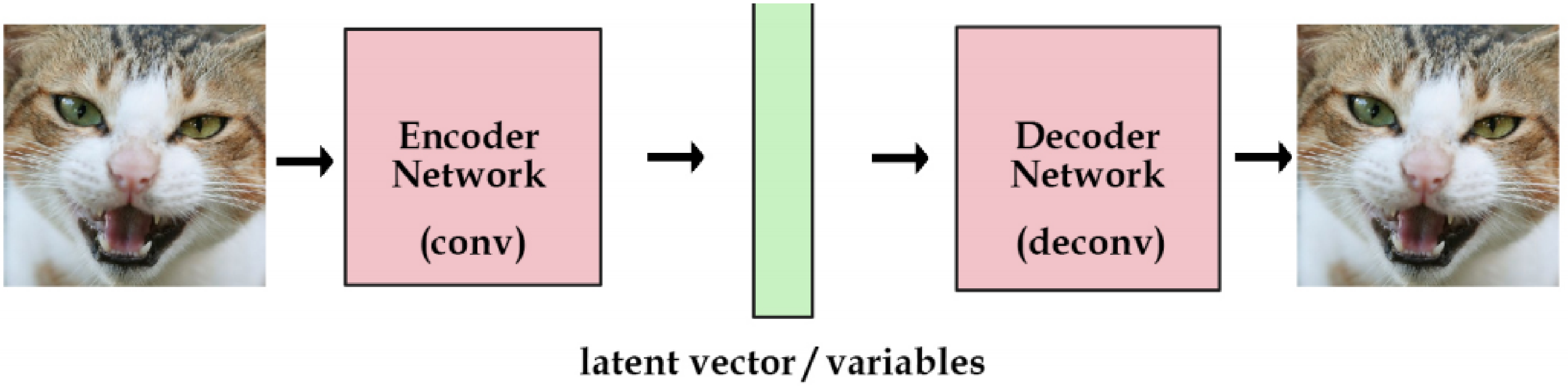
- Auto-association (auto-encoding)
  - Learn a compressed representation of the input, i.e., word2vec
  - Bottleneck layer = meaningful latent space
- Can de-couple encoder & decoder
  - Each can be complex, different functions



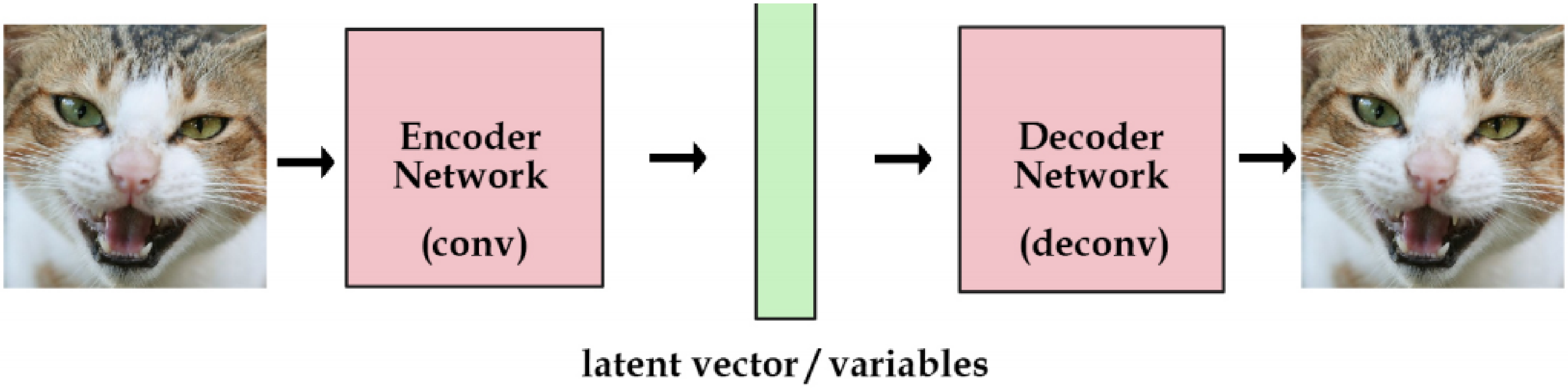
Autoencoding: Encoder-Decoder Framework

- Auto-association (auto-encoding)
  - Learn a compressed representation of the input, i.e., word2vec
  - Bottleneck layer = "meaningful" latent space
- Can de-couple encoder & decoder
  - Each can be complex, different functions





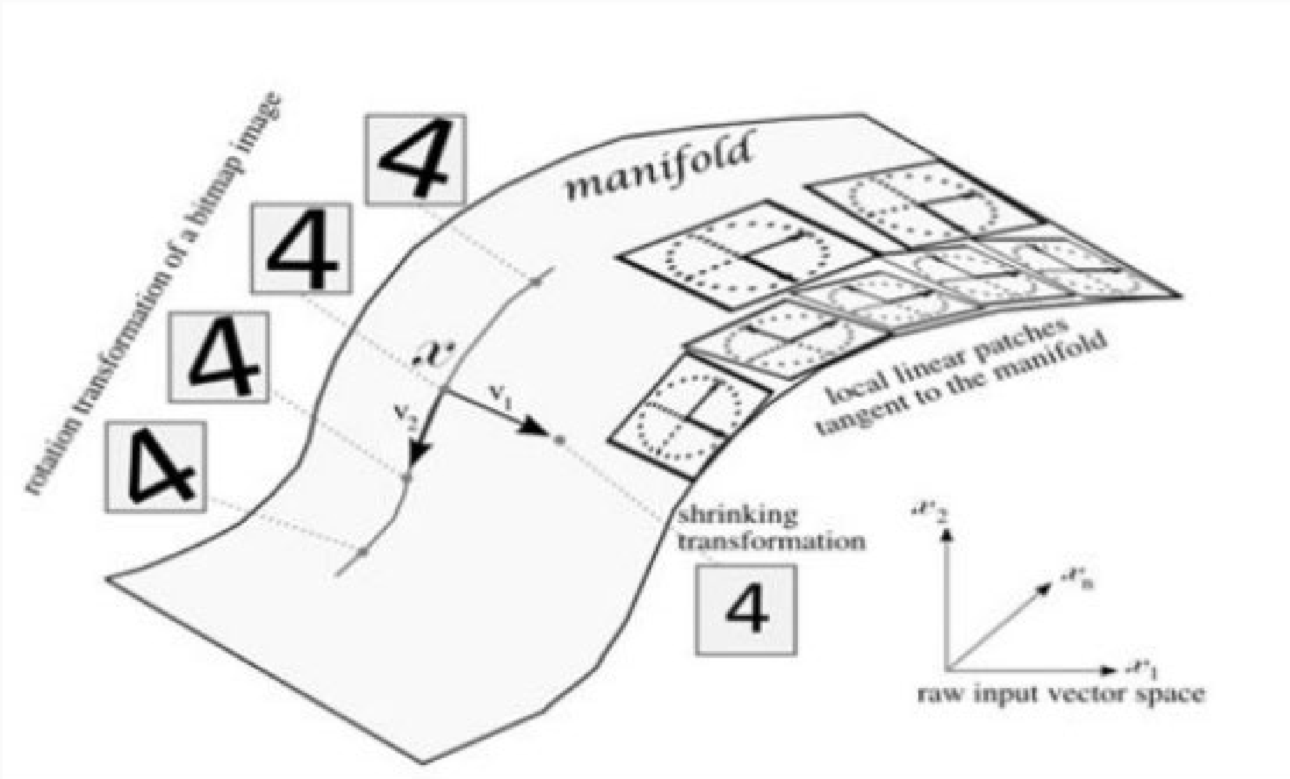
- Attempt to learn identity function
- Constrained in some way (e.g., small latent vector representation)
- Can generate new images by giving different latent vectors to trained network
- Variational: use probabilistic latent encoding



- Attempt to learn identity function
- Constrained in some way (e.g., small latent vector representation)
- Can generate new images by giving different latent vectors to trained network
- Variational: use probabilistic latent encoding

The Manifold Hypothesis

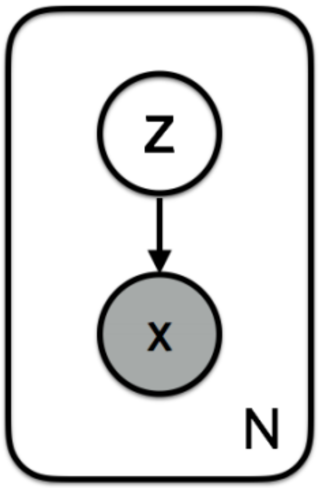
Natural data (high dimensional) actually lies in a low dimensional space.



<u>Nice consequence:</u> many datasets that you think might require many variables/dimensions to describe can actually be explained with only a few of them (a subset that forms a sort of local coordinate system of the underlying manifold)

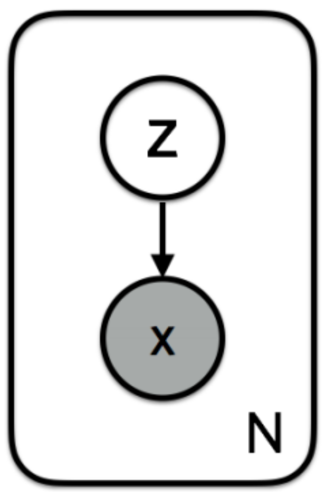
Probabilistic Model Perspective

- Data x and latent variables z
- Joint pdf of the model: $p(x, z) = p(x|z)p(z)$
- Decomposes into likelihood: $p(x|z)$, and prior: $p(z)$
- Generative process:
  Draw latent variables $z_i \sim p(z)$
  Draw datapoint $x_i \sim p(x|z)$
- Graphical model:



Probabilistic Model Perspective

- Data x and latent variables z
- Joint pdf of the model: $p(x, z) = p(x|z)p(z)$
- Decomposes into likelihood: $p(x|z)$, and prior: $p(z)$
- Generative process:
  Draw latent variables $z_i \sim p(z)$
  Draw datapoint $x_i \sim p(x|z)$
- Graphical model:



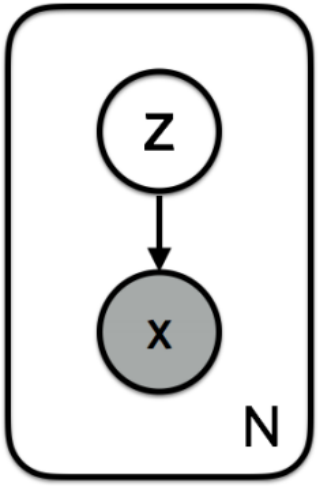
To learn this model, we could appeal to Monte Carlo sampling or to the calculus of variations…

Probabilistic Model Perspective

- Data x and latent variables z
- Joint pdf of the model: $p(x, z) = p(x|z)p(z)$
- Decomposes into likelihood: $p(x|z)$, and prior: $p(z)$
- Generative process:
  Draw latent variables $z_i \sim p(z)$
  Draw datapoint $x_i \sim p(x|z)$
- Graphical model:

Not sure if a learnable generative model…

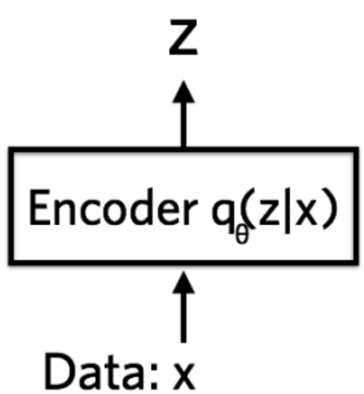
…or an intractable waste of time.



…so…we 're going to develop a ***variational inference*** scheme using your neural building blocks!

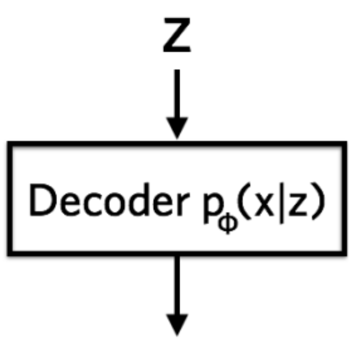
- Goal: Build a neural network that generates digits from random (Gaussian) noise
- Define two sub-networks: Encoder and Decoder

The variational distribution!

- Define a Loss Function
  - A neural network $q_\theta(z|x)$
  - Input: datapoint x (e.g. 28×28-pixel digit)
  - Output: encoding z, drawn from Gaussian density with parameters θ
  - $|z| \ll |x|$
  - 

- Goal: Build a neural network that generates digits from random (Gaussian) noise
- Define two sub-networks: Encoder and $\boxed{\text{Decoder}}$
- Define a Loss Function

- A neural network $p_\phi(x|z)$, parameterized by ϕ
- Input: encoding z, output from encoder
- Output: reconstruction $\tilde{x}$, drawn from distribution of the data
- E.g., output parameters for 28×28 Bernoulli variables



- Reconstruction: $\tilde{x}$

The Loss:

- $\tilde{x}$ is reconstructed from z where $|z| \ll |\tilde{x}|$
- How much information is lost when we go from x to z to $\tilde{x}$?
- Measure this with reconstruction log-likelihood: $\log p_\phi(x|z)$
- Measures how effectively the decoder has learned to reconstruct x given the latent representation z

- Loss function is negative reconstruction log-likelihood + regularizer
- Loss decomposes into term for each datapoint:

$$L(\theta, \phi) = \sum_{i=1}^{N} l_i(\theta, \phi)$$

- Loss for datapoint x_i:

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}\left[\log p_\phi(x_i|z)\right] + KL\left(q_\theta(z|x_i)||p(z)\right)$$

The Cost: $$L(\theta, \phi) = \sum_{i=1}^{N} \left(-\mathbb{E}_{z \sim q_\theta(z|x_i)}\left[\log p_\phi(x_i|z)\right] + KL\left(q_\theta(z|x_i)||p(z)\right) \right)$$

Neural Variational Inference (NVIL)

- *Idea*: Teach neural net to approximate the posterior *p(z|x)*
  - *q(z|x)* with 'variational parameters' φ
  - One-shot approximate inference
  - Also known as a recognition model
    - Construct estimator of the variational (evidence) lower bound (***ELBO***)
    - Can optimize jointly w.r.t. ϕ jointly with θ -> Stochastic gradient ascent

D_{KL} KL-Divergence >= 0 depends on how good q(z|x) can approximate p(z|x)

Recall from the start of the semester:

KL Divergence:

$$D_{\mathrm{KL}}(P\|Q) = \mathbb{E}_{\mathrm{x}\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{\mathrm{x}\sim P}\left[\log P(x) - \log Q(x)\right]. \qquad (3.50)$$

Gaussian KL Divergence:

$$KL(p,q) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$$

$$p \sim N(\mu_1, \sigma_1)$$
$$q \sim N(\mu_2, \sigma_2)$$

Not sure if optimizing log likelihood…

…or a variational evidence lower bound!

__Note__: this form is in terms of log likelihood

$$\mathcal{F}(\mathbf{x}, q) = \underbrace{\mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction}} - \underbrace{KL[q(\mathbf{z})\|p(\mathbf{z})]}_{\text{Penalty}}$$

Approx. Posterior

Interpreting the bound:

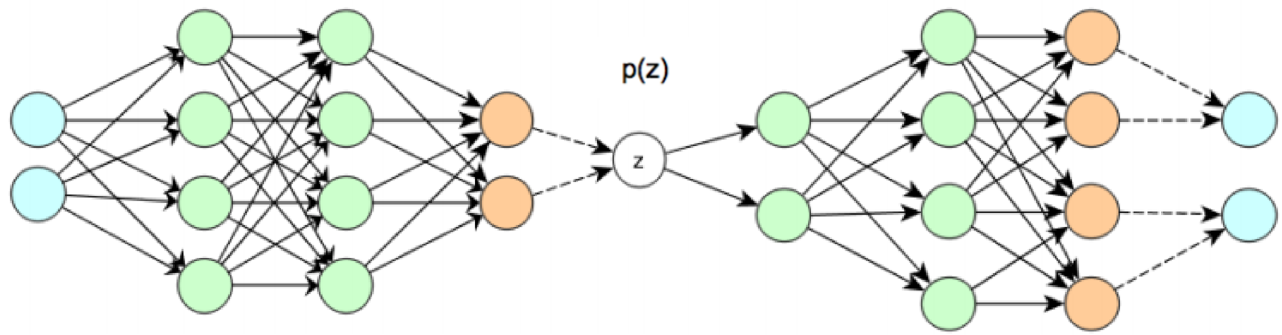
- **Approximate posterior distribution *q(z|x)*:** Best match to true posterior *p(z|x)*, one of the unknown inferential quantities of interest to us.
- **Reconstruction cost**: The expected log-likelihood measures how well samples from *q(z|x)* are able to explain the data *x*.
- **Penalty:** Ensures that the explanation of the data *q(z|x)* doesn't deviate too far from your beliefs *p(z)*. A mechanism for realising Ockham's razor.

The Variational Auto-Encoder

- A feed forward NN + Gaussian $q_{\theta}(z \mid x) = \mathcal{N}(z; \mu_z(x), \sigma_z(x))$

Just a Gaussian, with diagonal covariance.

$q_{\theta}(x|z)$ $\qquad$ $p_{\varphi}(x|z) \quad x|z \sim N(\mu_x, \sigma_x^{\,2})$



- For illustration z one dimensional x 2D
- Want a complex model of distribution of x given z

Learning the parameters φ and θ via backpropagation

Determining the loss function

$$\mathcal{F}(\mathbf{x}, q) = \boxed{\mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x}|\mathbf{z})]} - \boxed{KL[q(\mathbf{z})\|p(\mathbf{z})]}$$

Approx. Posterior $\qquad$ Reconstruction $\qquad$ Penalty

Interpreting the bound:

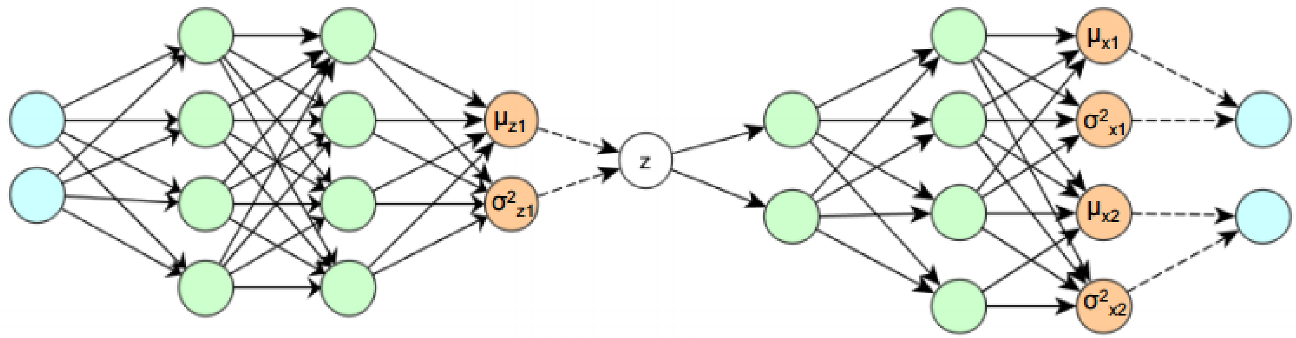
- **Approximate posterior distribution** ***q(z|x)*:** Best match to true posterior *p(z|x)*, one of the unknown inferential quantities of interest to us.

- **Reconstruction cost**: The expected log-likelihood measures how well samples from *q(z|x)* are able to explain the data *x*.

- **Penalty:** Ensures that the explanation of the data *q(z|x)* doesn't deviate too far from your beliefs *p(z)*. A mechanism for realising Ockham's razor.

Putting It All Together!

Prior p(z) ~ N(0,1) and p, q Gaussian, extension to dim(z) > 1 trivial



Cost: Regularisation

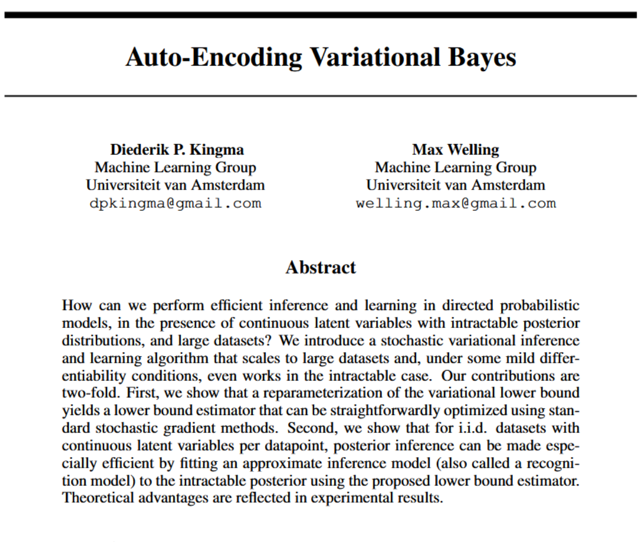
$$-D_{KL}\left(q(z|x^{(i)})||p(z)\right) = \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log(\sigma_{z_j}^{(i)^2}) - \mu_{z_j}^{(i)^2} - \sigma_{z_j}^{(i)^2}\right)$$

Cost: Reproduction

$$-\log\left(p(x^{(i)}|z^{(i)})\right) = \sum_{j=1}^{D}\frac{1}{2}\log(\sigma_{x_j}^2) + \frac{(x_j^{(i)} - \mu_{x_j})^2}{2\sigma_{x_j}^2}$$

We use mini batch gradient decent to optimize the cost function over all $x^{(i)}$ in the mini batch

Least Square for constant variance



Auto-Encoding Variational Bayes

Diederik P. Kingma
Machine Learning Group
Universiteit van Amsterdam
dpkingma@gmail.com

Max Welling
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

Abstract

How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case. Our contributions are two-fold. First, we show that a reparameterization of the variational lower bound yields a lower bound estimator that can be straightforwardly optimized using standard stochastic gradient methods. Second, we show that for i.i.d. datasets with continuous latent variables per datapoint, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator. Theoretical advantages are reflected in experimental results.

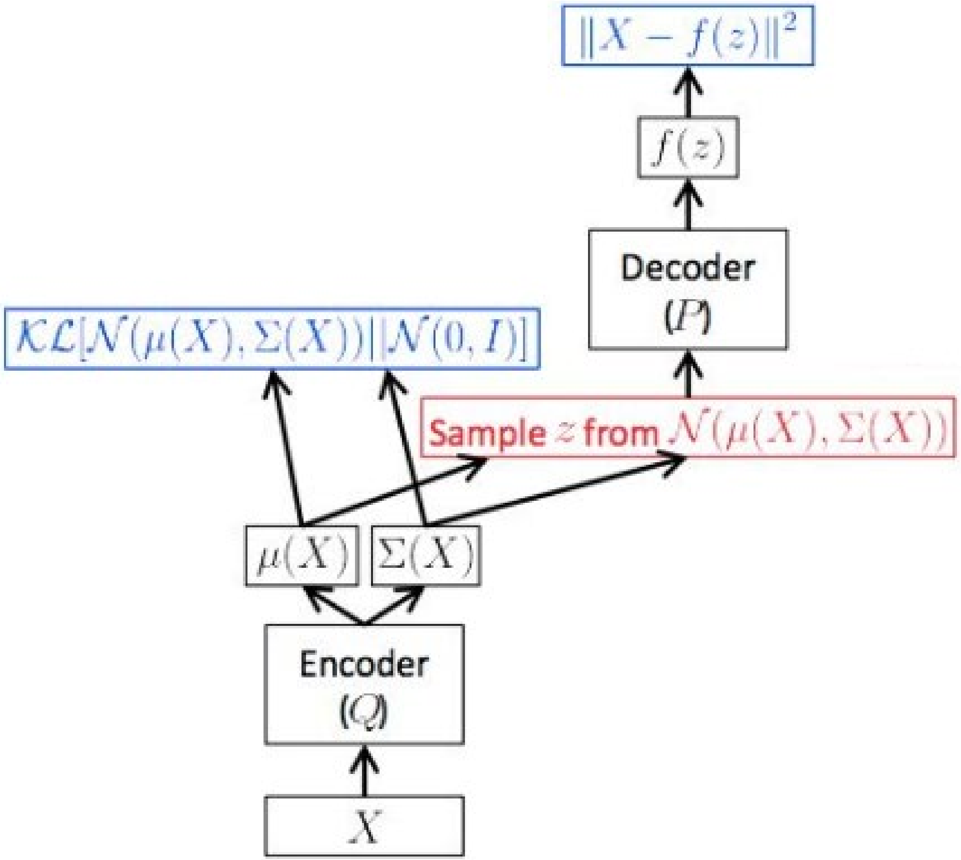
1 Introduction

<u>Issue</u>: Backprop and Sampling

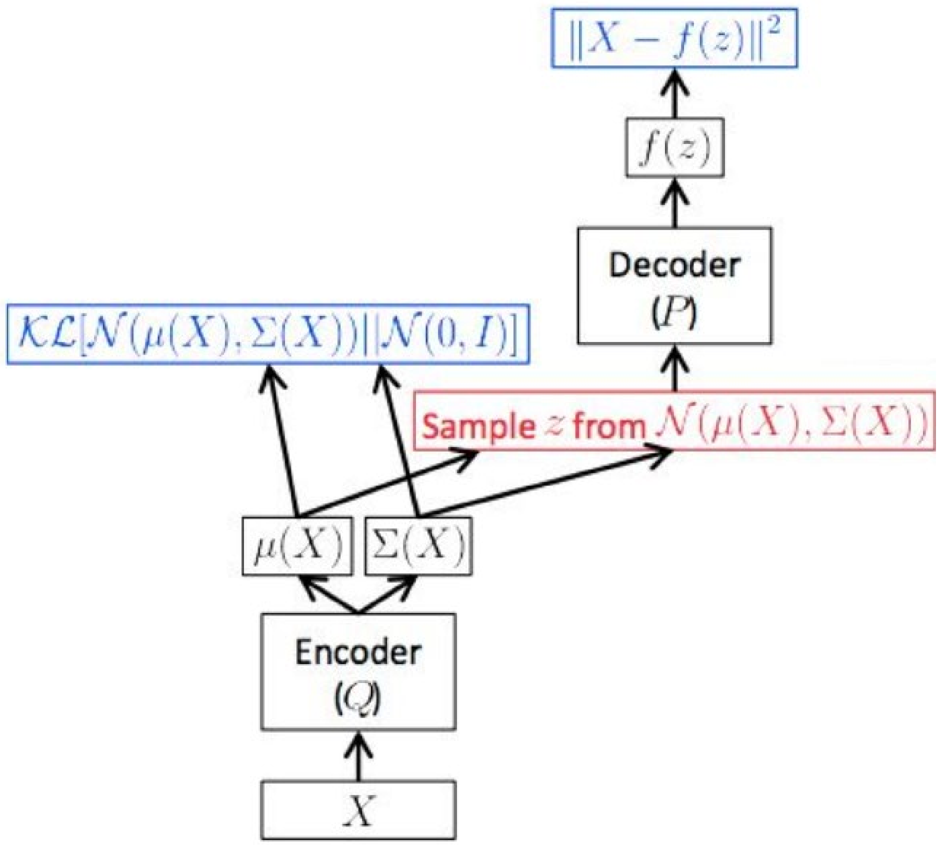
Training the Decoder is easy, just standard backpropagation.

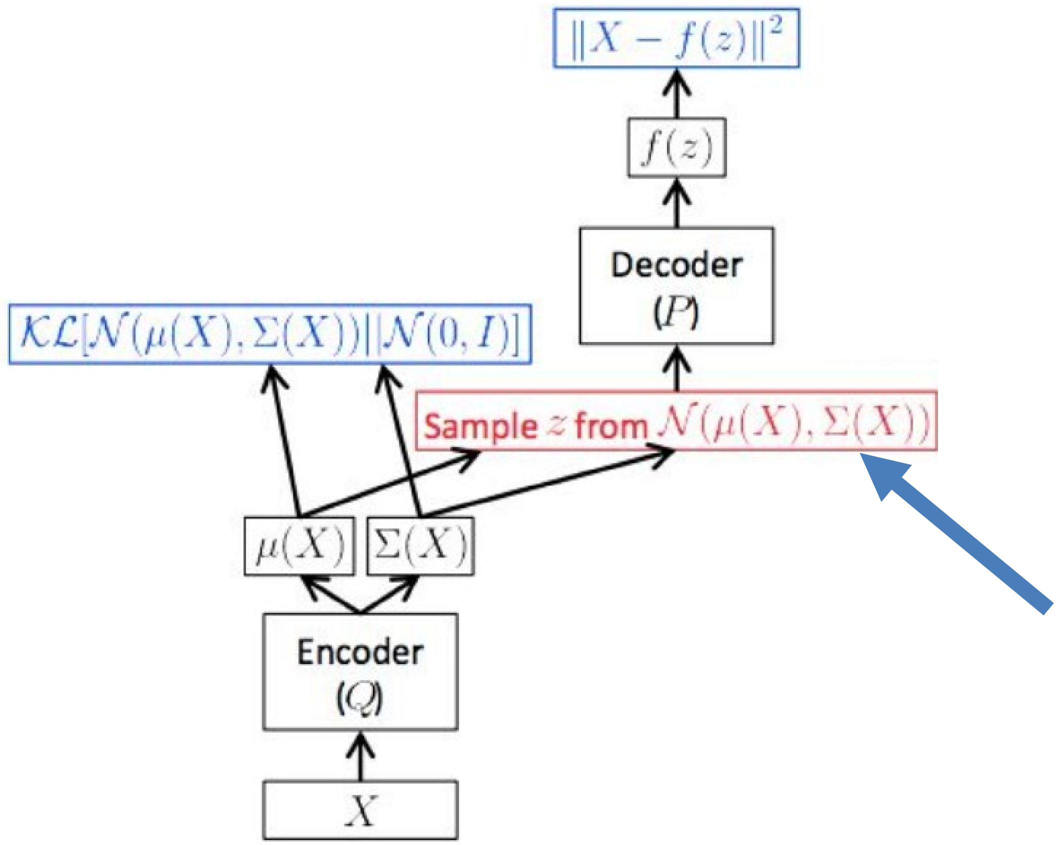
How to train the Encoder?

- Not obvious how to apply gradient descent through samples.



But…we have a **problem**!



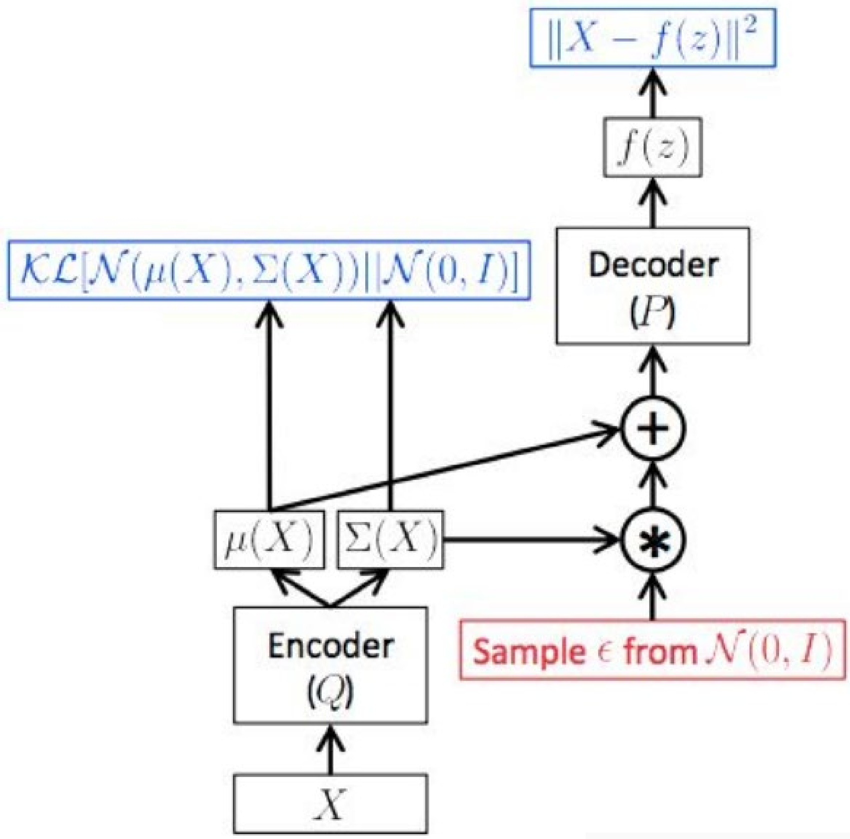


?? Backprop through numpy's Gaussian RNG**??**

The Reparameterization ‘Trick’

- We want to use gradient descent to learn the model’s parameters
- Given z drawn from $q_\theta(z|x)$, how do we take derivatives of (a function of) z w.r.t. θ?
- We can reparameterize: $z = \mu + \sigma \odot \epsilon$
- $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\odot$ is element-wise product
- Can take derivatives of (functions of) z w.r.t. μ and σ
- Output of $q_\theta(z|x)$ is vector of μ’s and vector of σ’s

The Reparameterization Trick

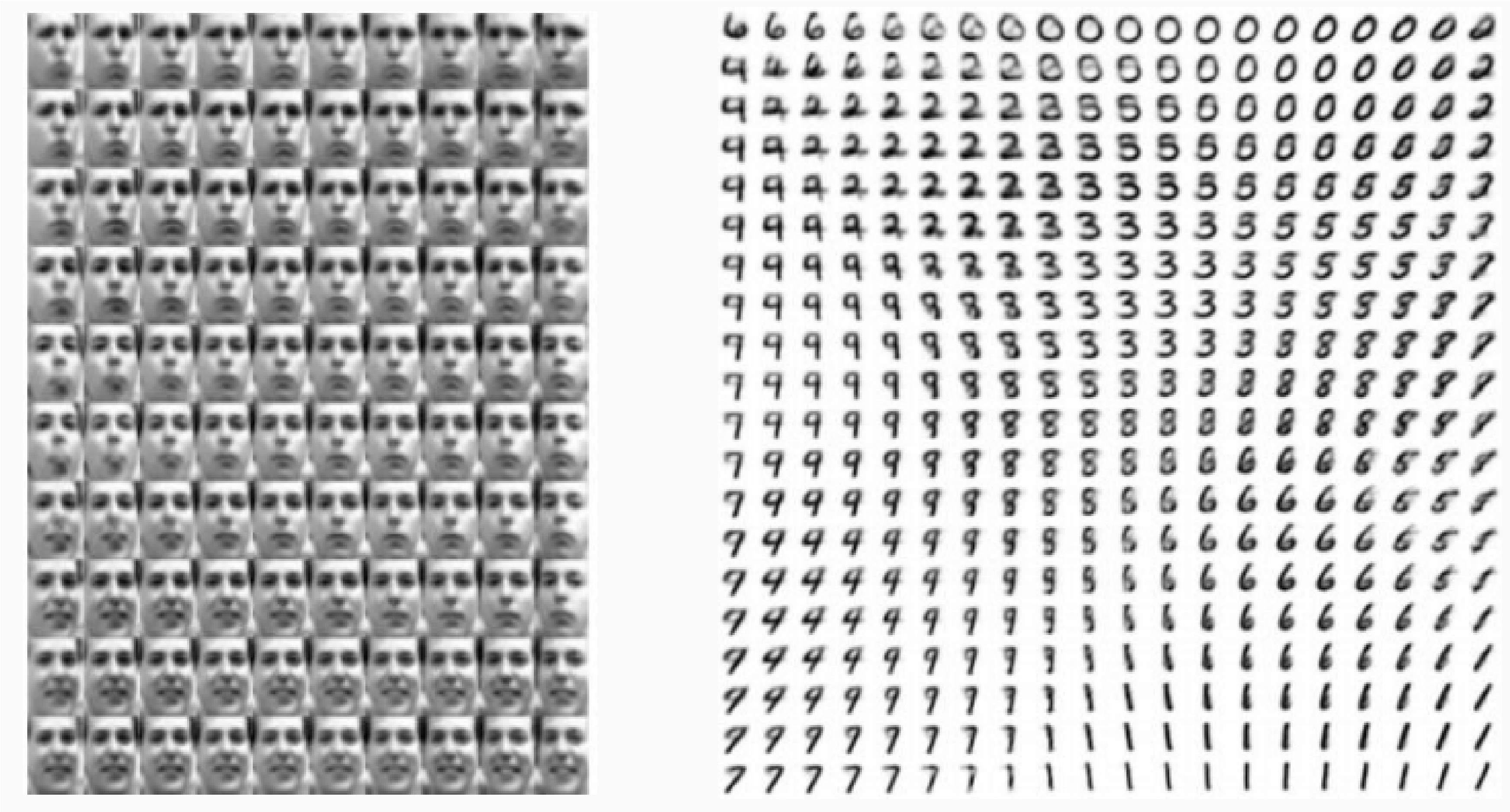


z ~ N(μ, σ) is equivalent to

μ + σ · ε, where ε ~ N(0, 1)

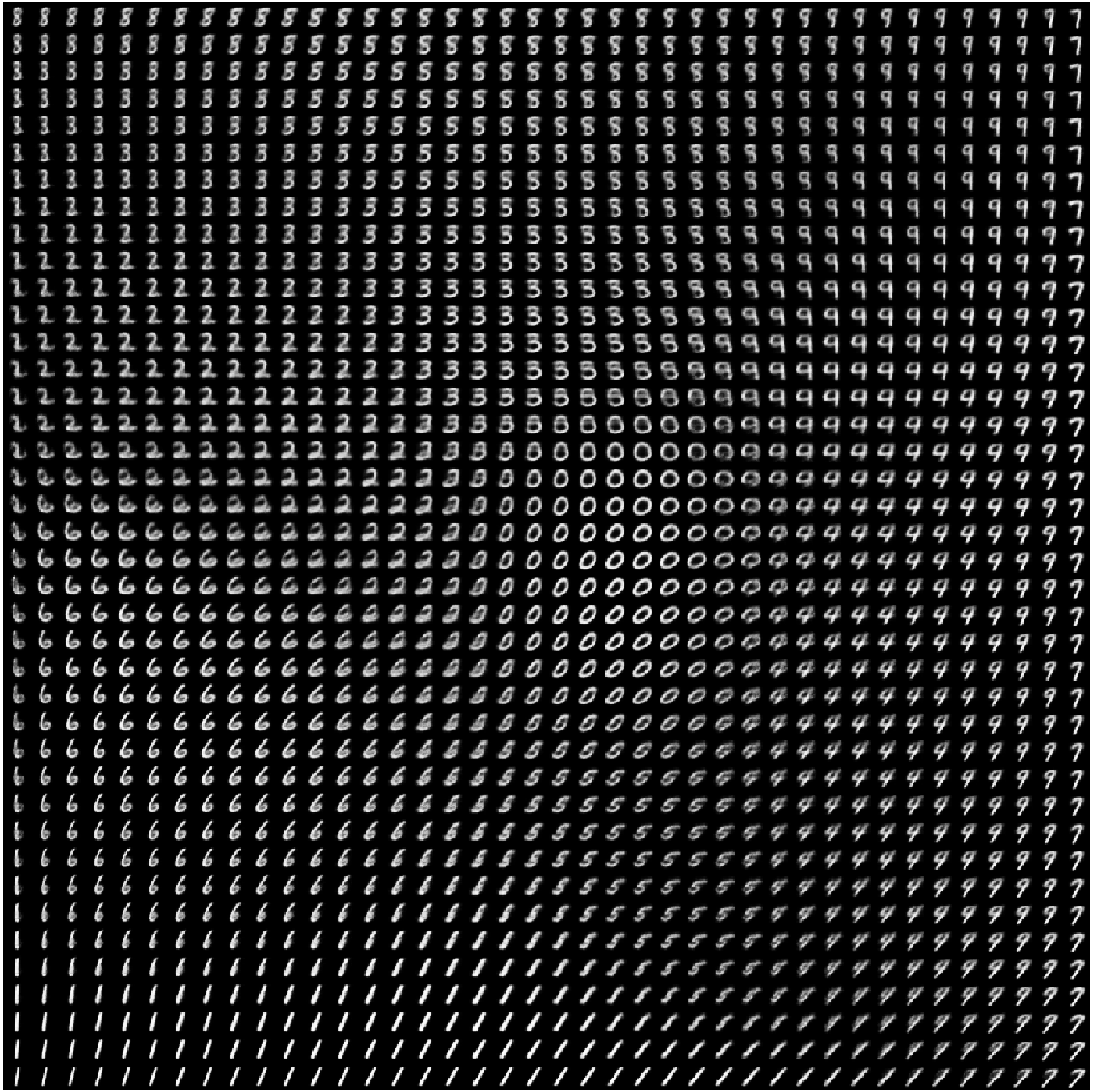
Fantasies/Dreams of the VAE

- VAEs can disentangle potential factors of variation



http://www.dpkingma.com/sgvb_mnist_demo/demo.html

Deep digit dreams…



Issues/Limitations with VAEs

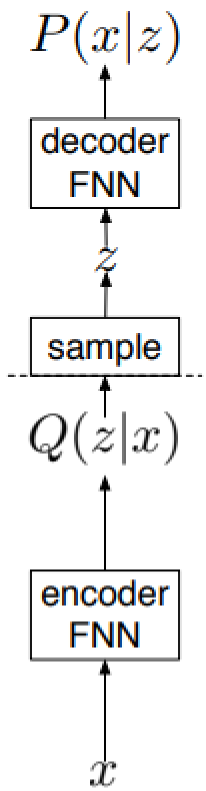
- Images are blurry (compared to models based on GANs, for example)
  - Result of likelihood objective? (places probability mass on training images and nearby points, which include blurry images)
    - Has tendency to ignore input features that occupy few pixels or that cause only small change in brightness of pixels (that they occupy)
  - Uses only small subset of latent variables? (struggles to find enough transformation directions to match factorized prior over z)

DRAW

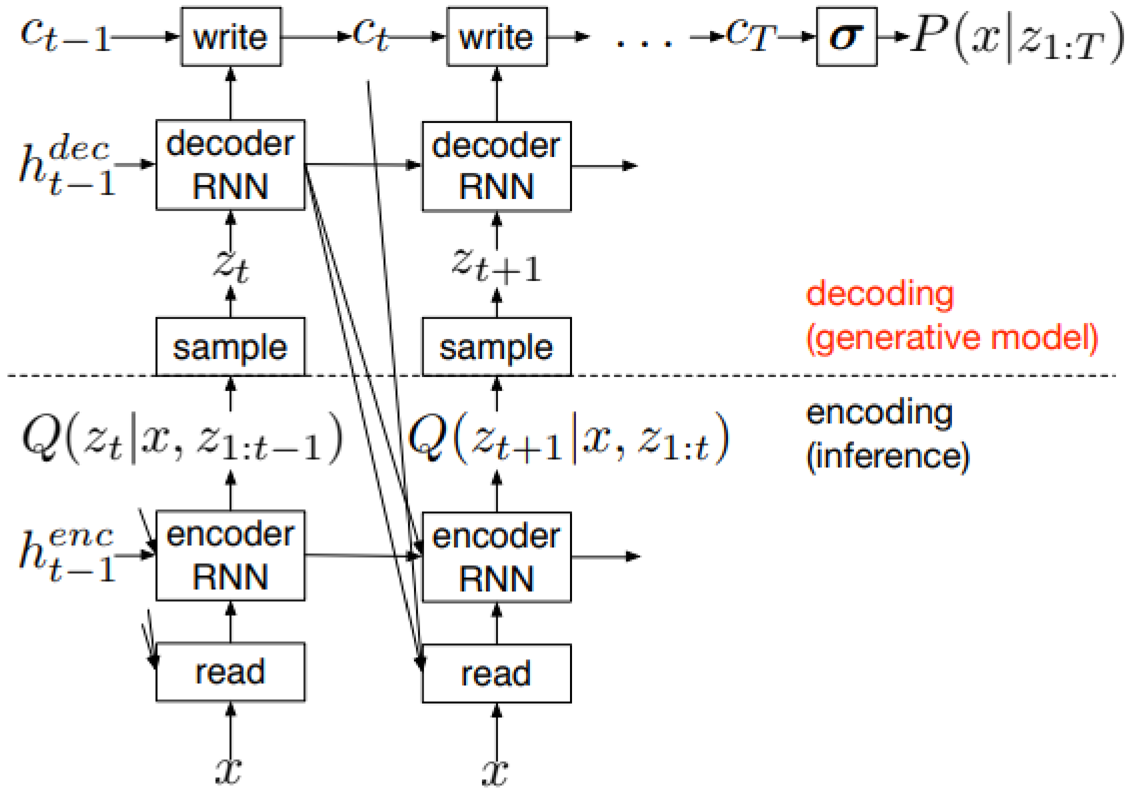
The Deep Recurrent Attentive Writer

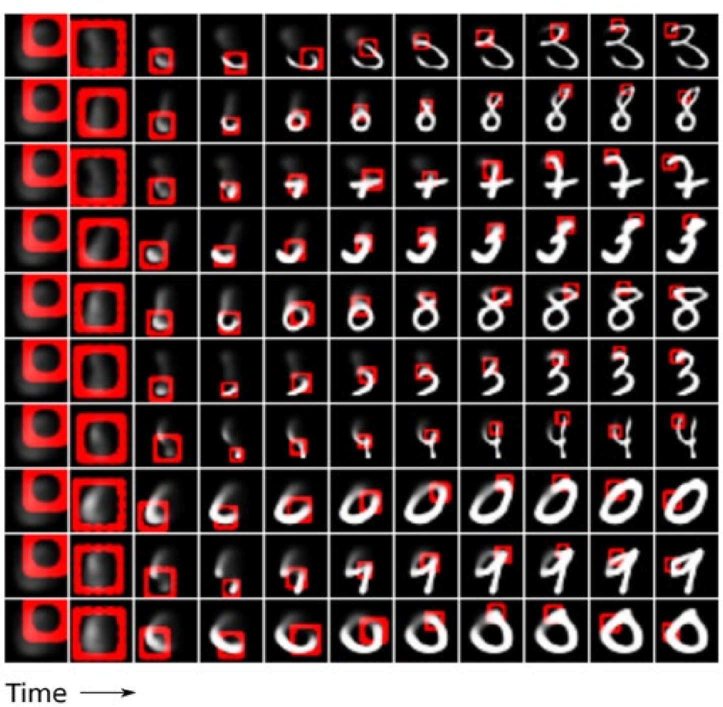
Combining latents, variational inference, and RNNs

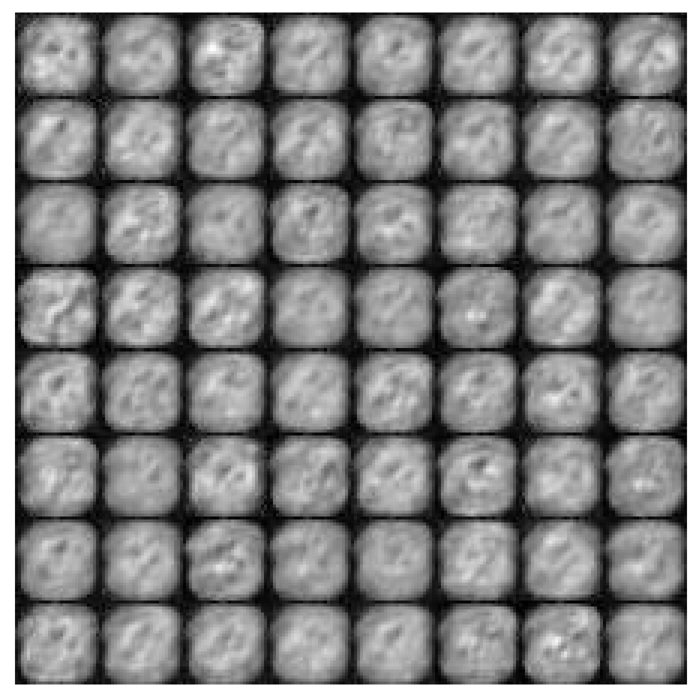
VAE



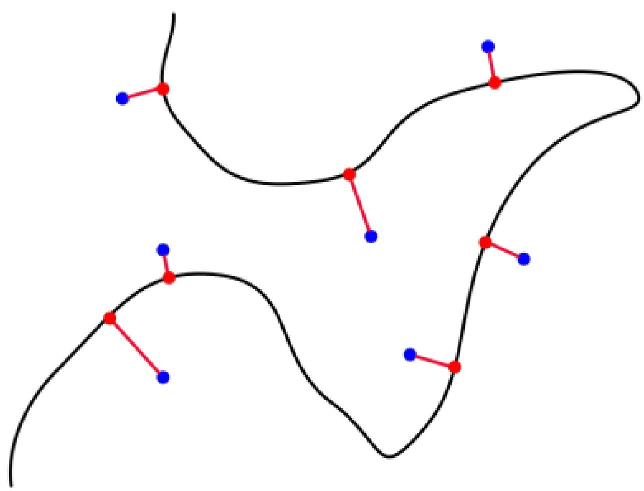
DRAW



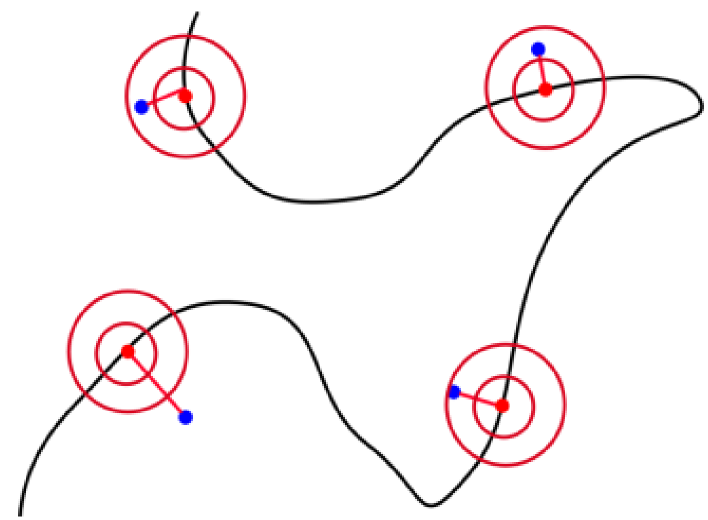




Simple Autoencoder

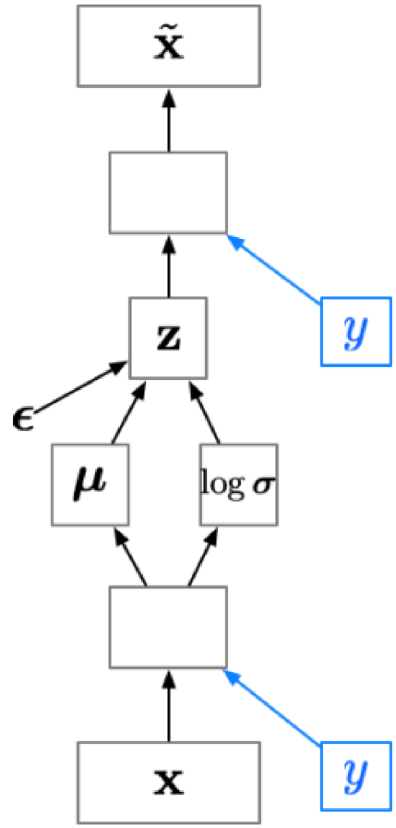


Variational Autoencoder



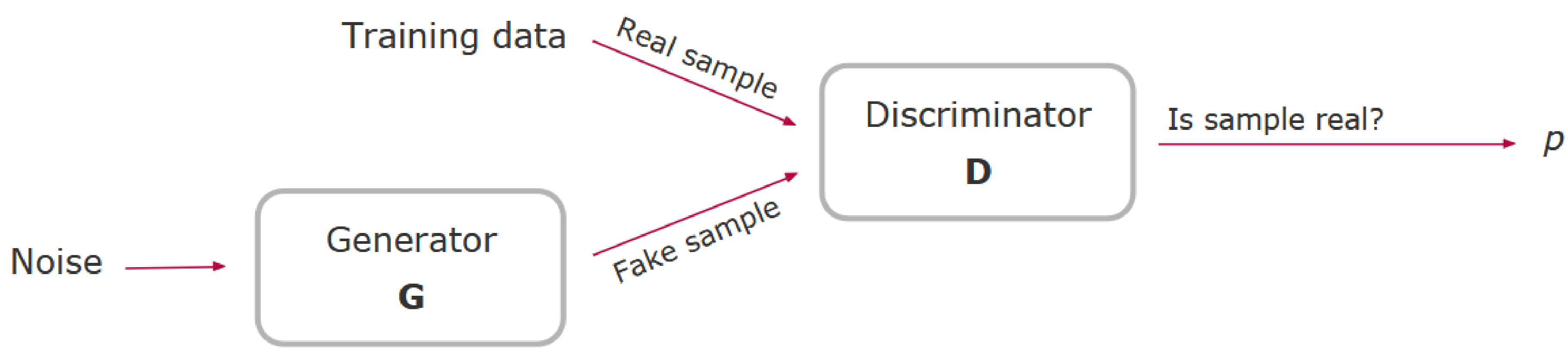
Class-Conditional VAE

- A class-conditional VAE provides labels to both encoder and decoder
  - Since latent code **z** no longer has to model data category, can focus on modeling stylistic features



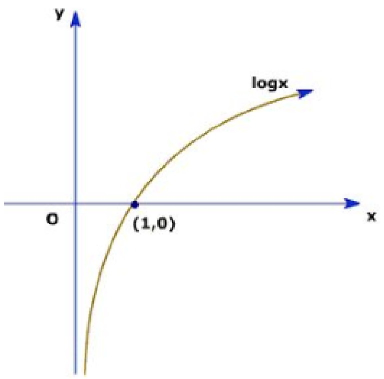
Enter the Generative Adversarial Network (the GAN)

- **Generator (G)** that learns the real data distribution to generate fake samples
- **Discriminator (D)** that attributes a probability *p* of confidence of a sample being real (*i.e.* coming from the training data)



GAN Objective (Evaluation)

- Both models are trained together **(minimax game):**
  - G: Increase the probability of D making mistakes
  - D: Classify real samples with greater confidence



- G *slightly changes* the generated data based on D's feedback

$$\min_{G}\max_{D} V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

- **Ideal scenario (equilibrium)**: G will eventually produce such realistic samples that D attributes p = 0.5 (*i.e.* cannot distinguish real and fake samples)

GAN Optimization

for number of training iterations **do**

for k steps **do**

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

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

end for

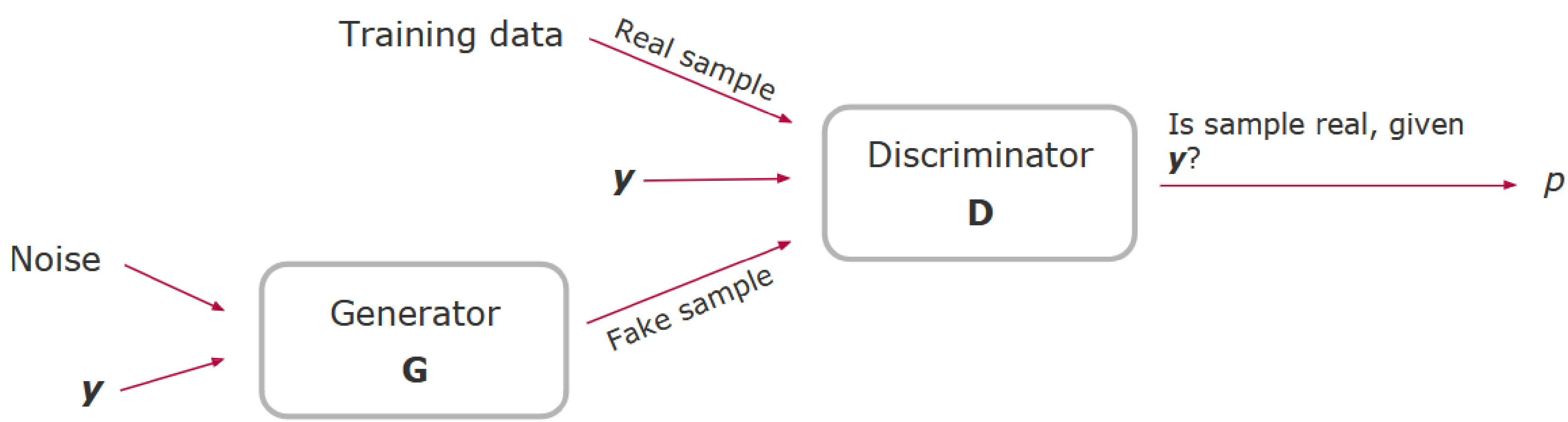
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

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

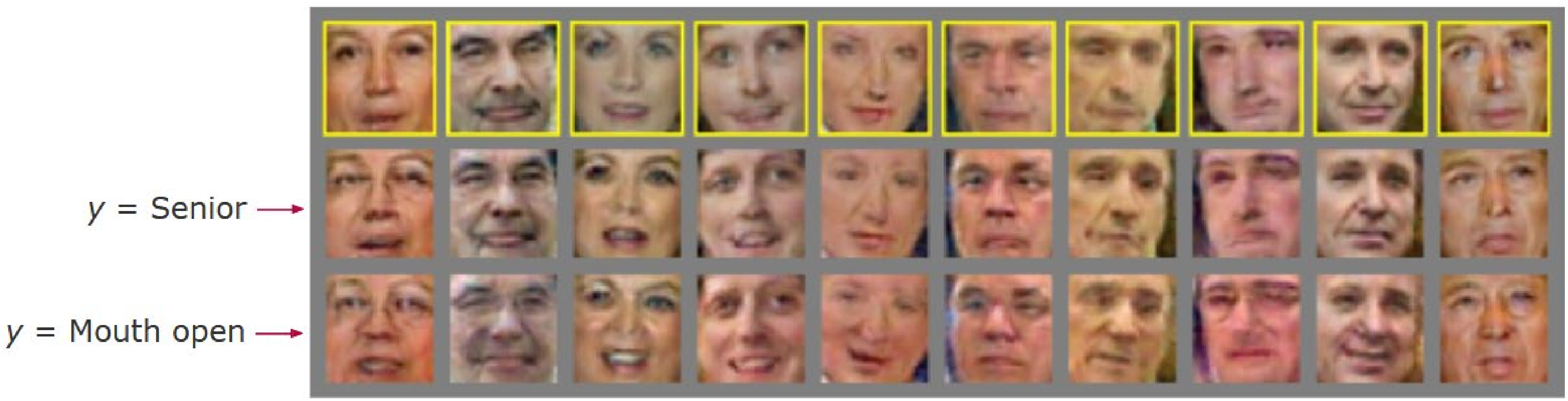
end for

Conditional GANs

- G and D can be **conditioned by additional information *y***
- Adding y as an input of both networks will condition their outputs
- y can be external information or data from the training set

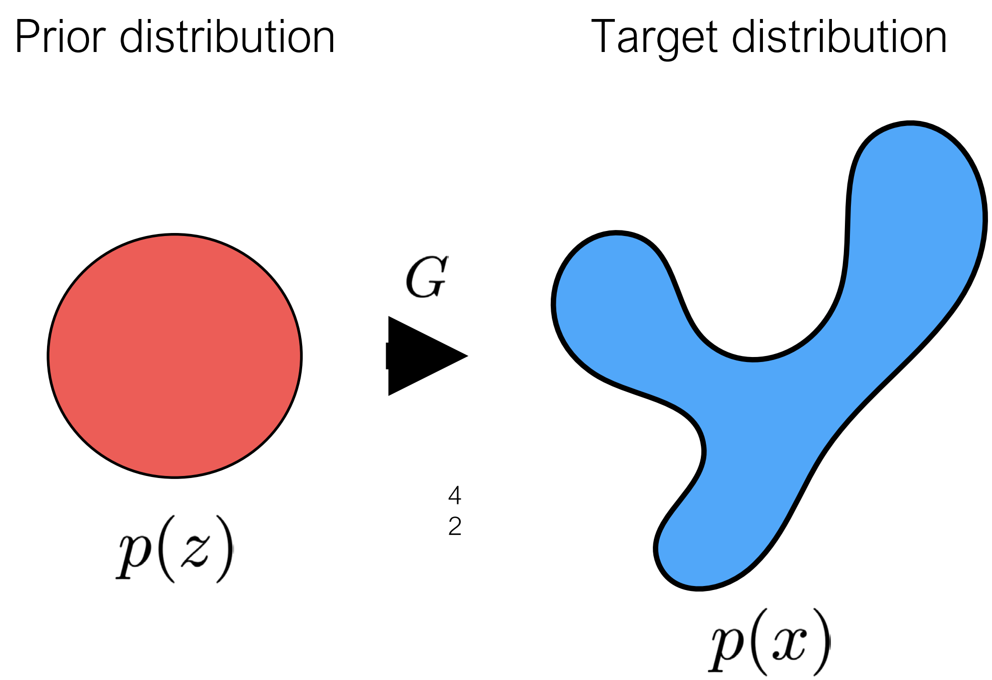


Conditional GANs

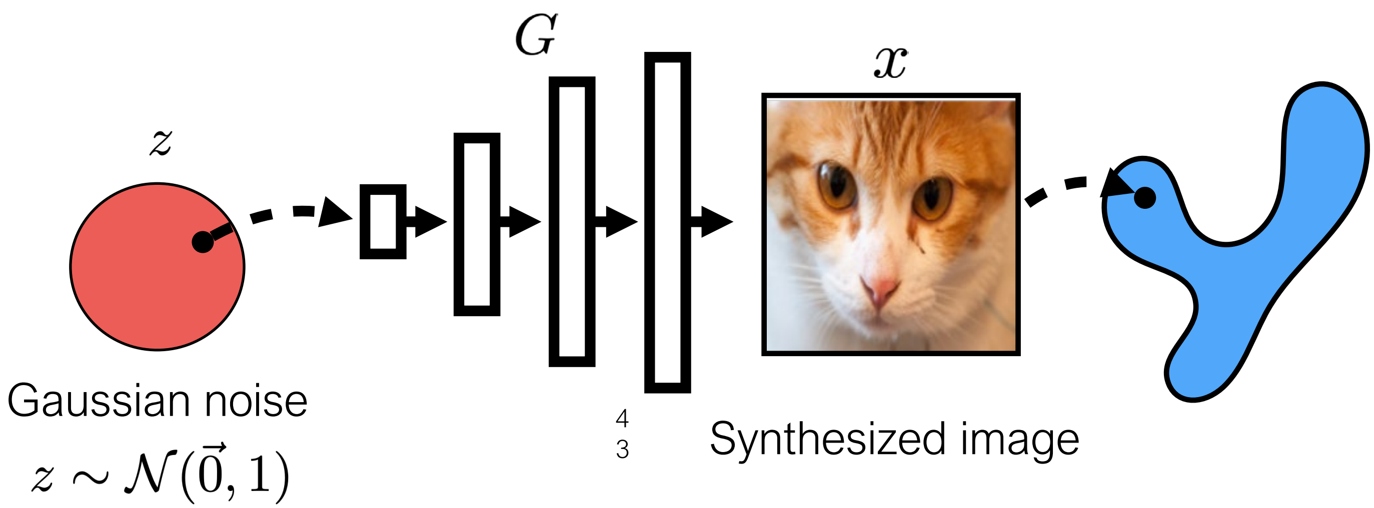


Gauthier, 2015

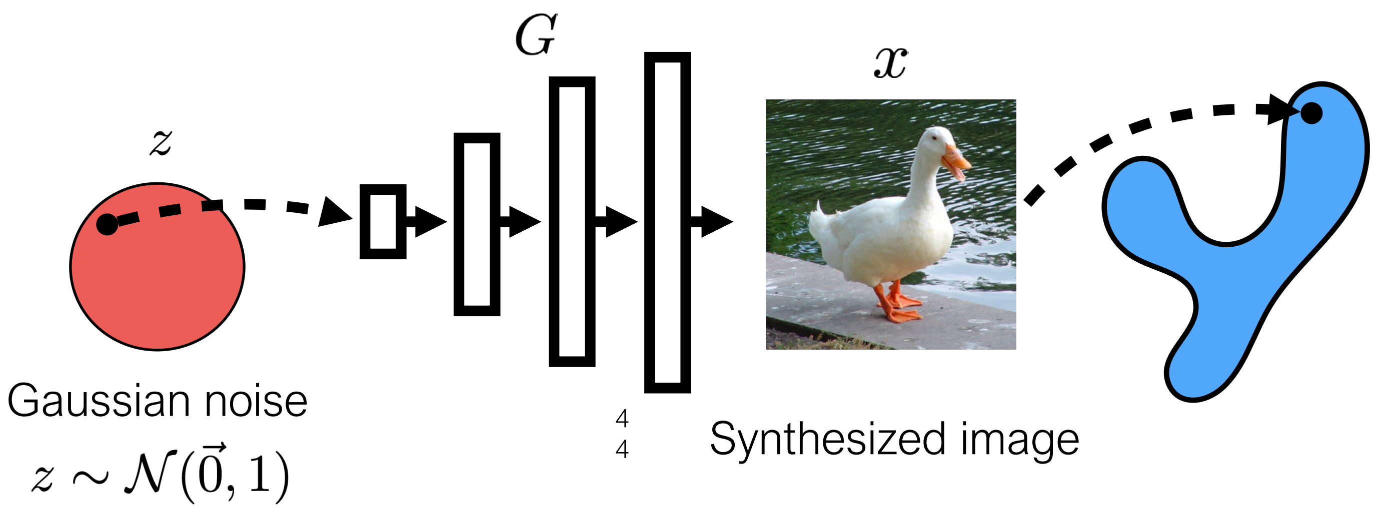
Deep generative models are distribution transformers



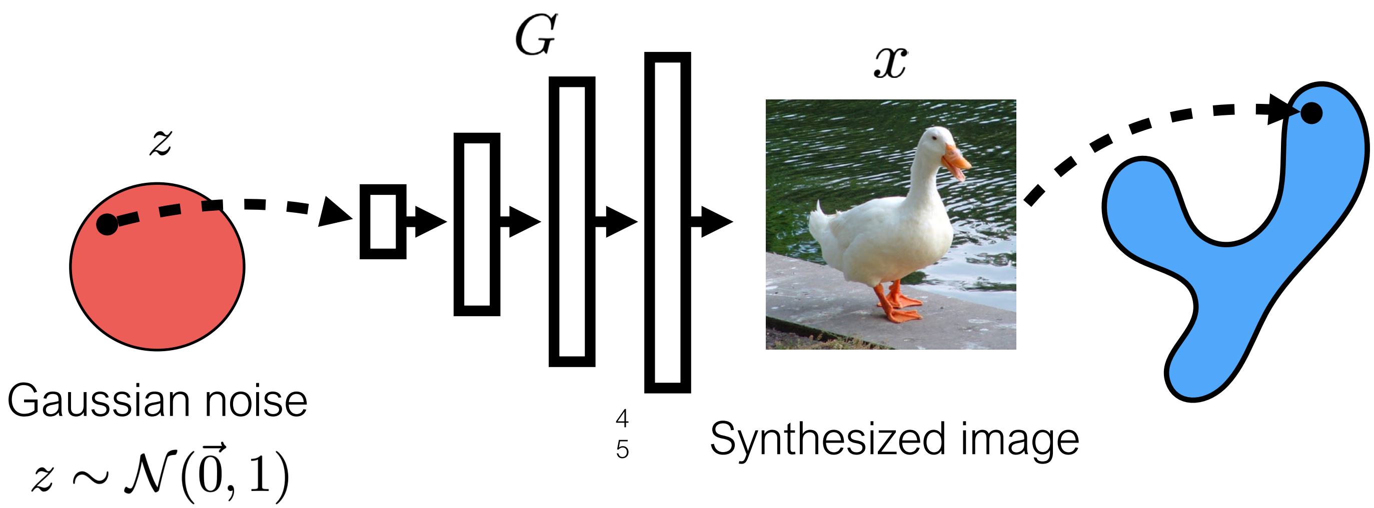
Deep generative models are distribution transformers

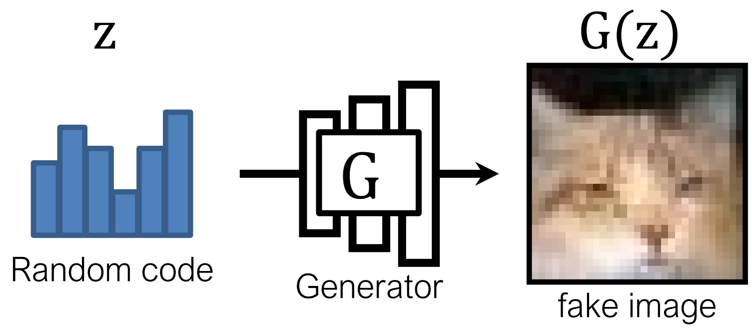


Deep generative models are distribution transformers

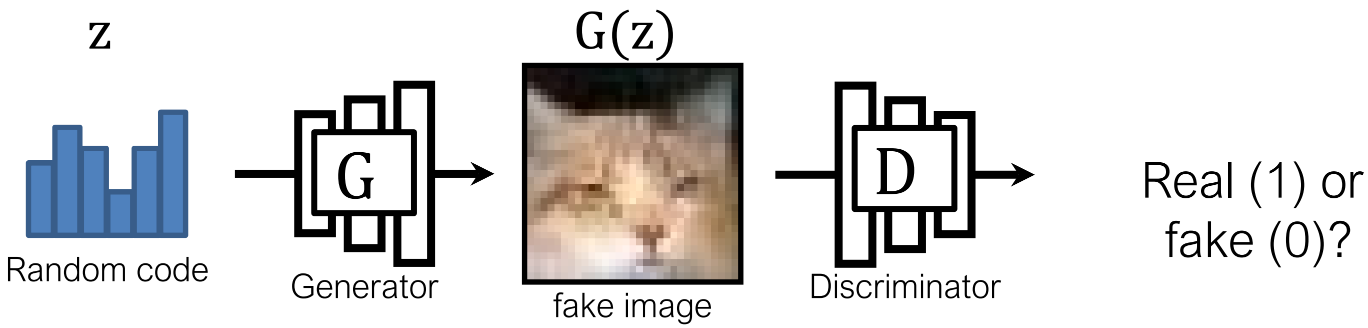


Generative Adversarial Networks Are Distributional Transformers!





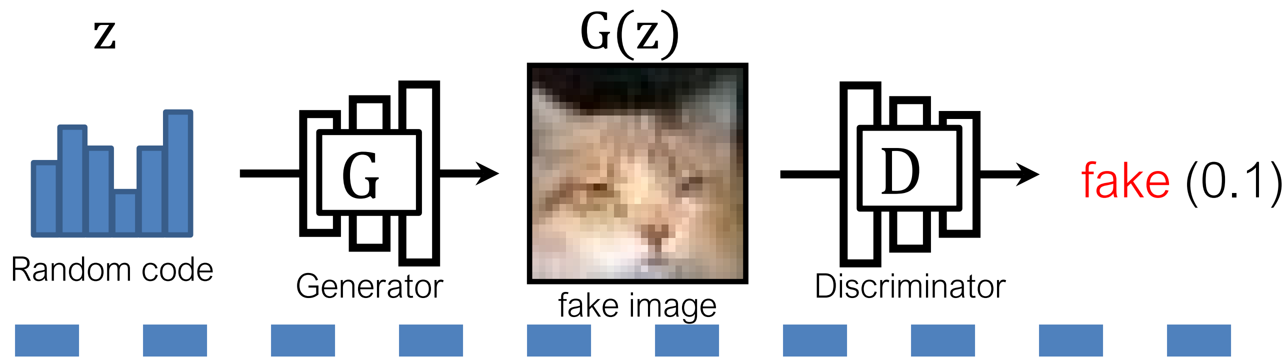
[Goodfellow et al. 2014]



A two-player game:

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

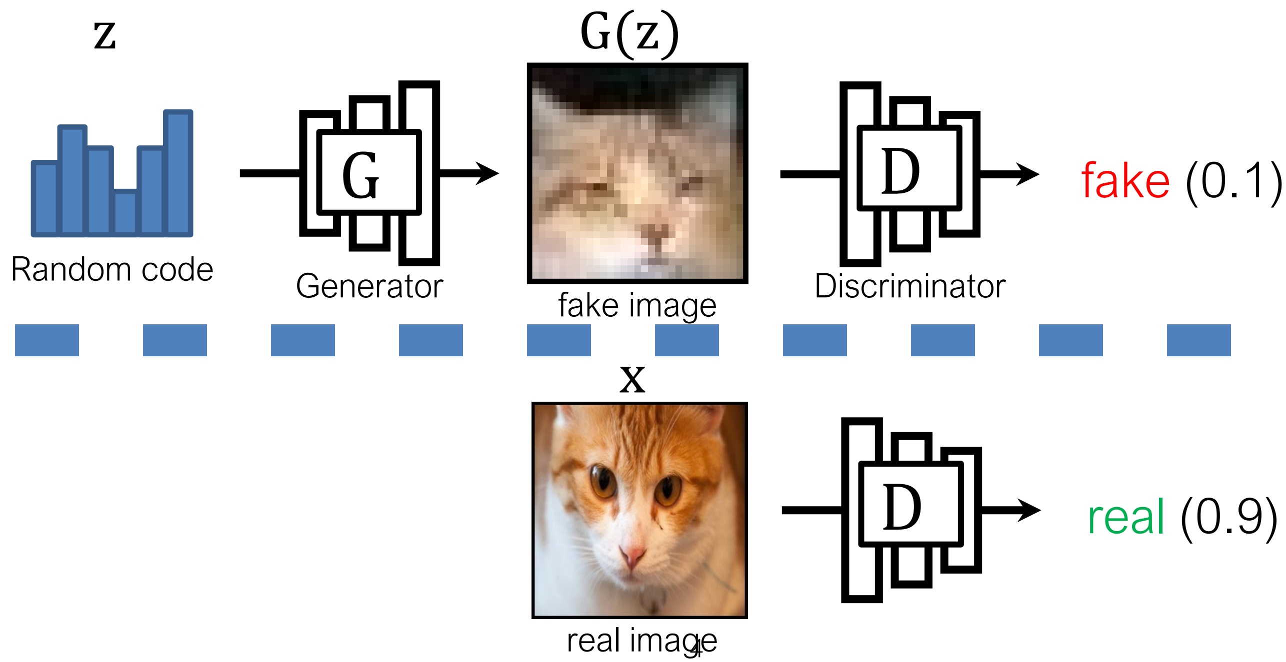
[Goodfellow et al. 2014]



Learning objective (GANs)

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

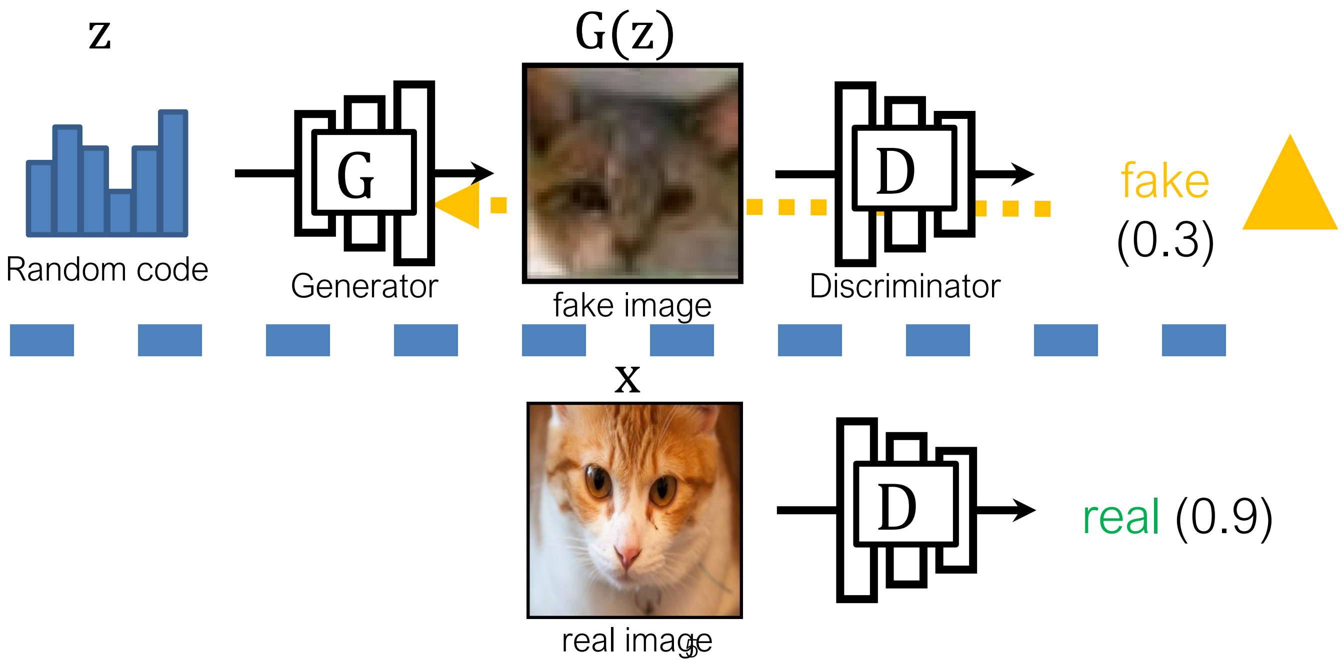
[Goodfellow et al. 2014]



Learning objective (GANs)

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

[Goodfellow et al. 2014]

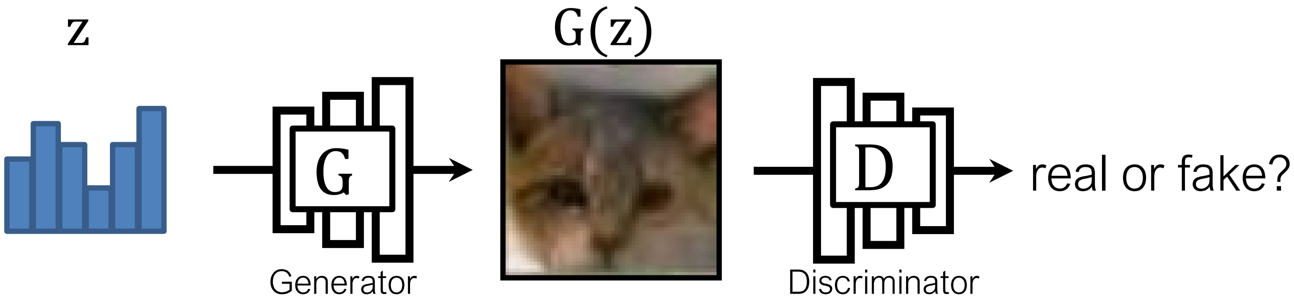


Learning objective (GANs)

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

[Goodfellow et al. 2014]

GANs Training



G tries to synthesize fake images that fool **D**

D tries to identify the fakes

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

[Goodfellow et al., 2014]

Generative Adversarial Network

Data $\rightarrow$

Learner

Objective

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

Hypothesis space

Deep nets G and D

Optimizer

Alternating SGD on G and D

$\rightarrow$

Critic

$D : \mathcal{X} \rightarrow [0,1]$

Sampler

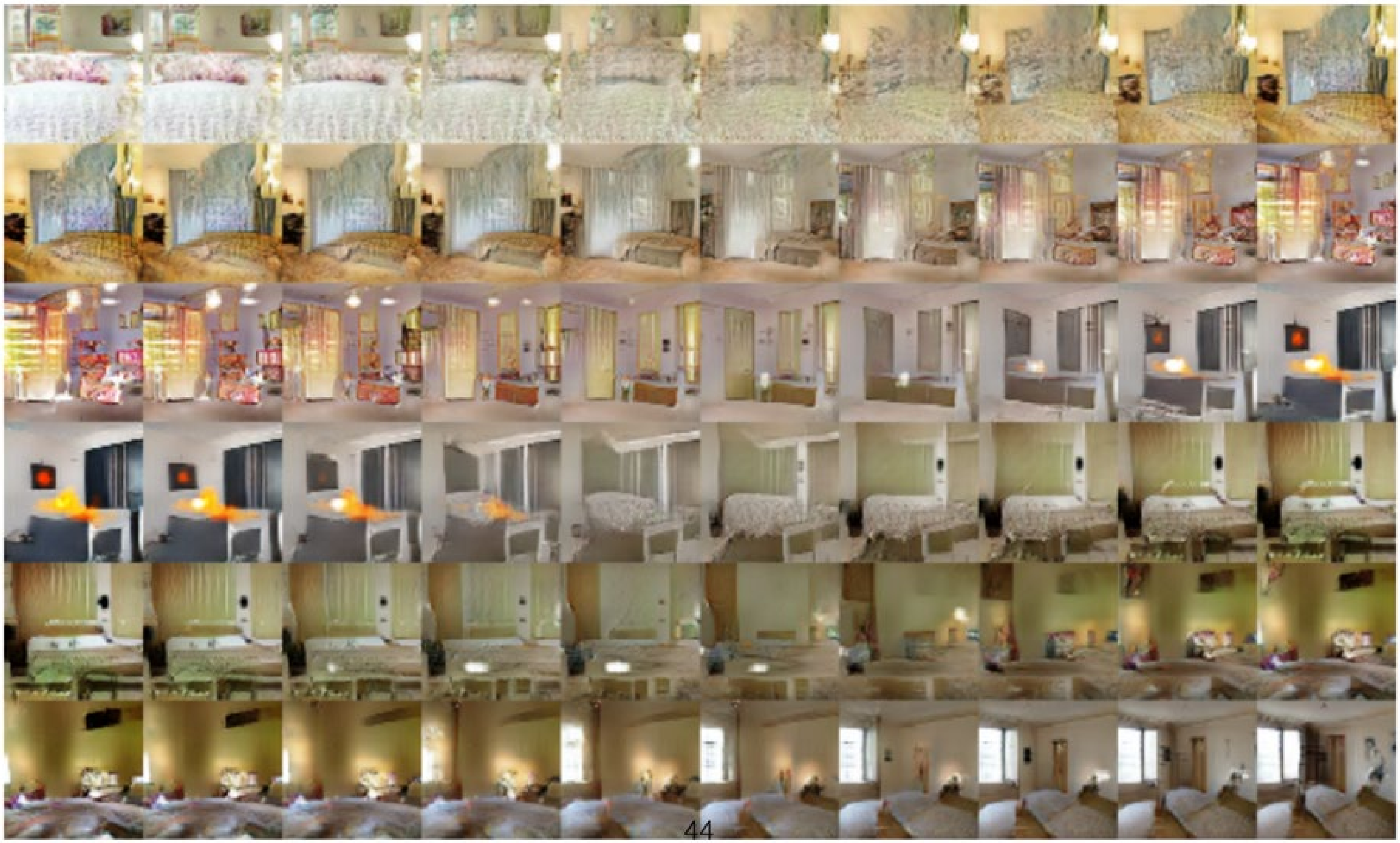
$G : \mathcal{Z} \rightarrow \mathcal{X}$

DCGAN

[Radford, Metz, Chintala 2015]

DCGAN

[Radford, Metz, Chintala 2015]



Problems with GANs

1. **Training instability**
   - Good sample generation requires reaching Nash Equilibrium in the game, which might not always happen
2. **Mode collapse**
   - When G is able to fool D by generating similarly looking samples from the same data mode
3. GANs were original made to work only with **real-valued, continuous data** (*e.g.* images)
   - *Slight changes* in **discrete data** (*e.g.* text) are impractical

Evaluation is Really Hard!

- What makes a good generative model?
  - Each generated sample is indistinguishable from a real sample

  - Generated samples should have variety

QUESTIONS?