Lecture 14

Generative models

Information Systems
(Machine Learning)

Andrey Filchenkov

27.12.2018

Lecture plan

- Task of geniting new objects
- Variational autoencoders
- PixelCNN and PixelRNN
- Generative adversarial models
- Advances in GANs

- The presentation is prepared with materials of
  - F.F. Li et al.' course "Convolutional Neural Networks for Visual Recognition"
  - A.S. Artamonov's course "Image and video analysis"
- Slides are available online: **goo.gl/BspjhF**

Lecture plan

- Task of geniting new objects
- PixelCNN and PixelRNN
- Variational autoencoders
- Generative adversarial models
- Advances in GANs

Generating new objects

What kind of task is it?

- supervised
- unsupervised
- semi-supervised
- reinforcement

Definition obstacles of generating new objects

What kind of task is it?

It depends on how we define quality and novelty.

We can use empirical rules to generate something new, no machine learning required.

Machine learning vision

- We want new objects to be plausible with respect to domain
- New image of a person should be plausible as an image, and person should be plausible, as well as all details should be plausible

The obstacle of machine learning vision

- Given a sample, of new objects are implausible because we have never seem them in sample.

The obstacle of machine learning vision

- Given a sample, of new objects are implausible because we have never seem them in sample.
- We want to create object plausible with respect to some hidden structure of objects

Generation is unsupervised learning

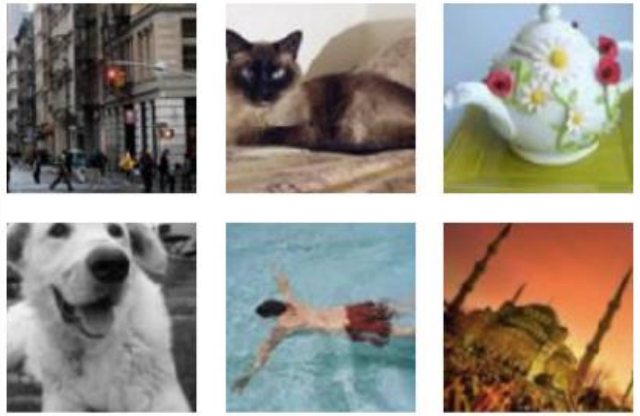
- Given a sample, of new objects are implausible because we have never seem them in sample.
- We want to create object plausible with respect to some hidden structure of objects

- This is unsupervised learning task! We need to learn hidden structure of object, learn distribution over them and then simple sample a new object from this distribution

Generation task

Given training data, generate new samples from same distribution

Training sample $p_{\text{data}}(x)$



Generated sample $p_{\text{model}}(x)$

Learn $p_{\text{model}}(x)$ similar to $p_{\text{data}}(x)$

How to measure distribution similarity?

Kullback-Leibner divergence:

Given P distributed with p and Q distributed with q,

$$D_{\mathrm{KL}}(P||Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)}.$$

Also called the **relative entropy** of P with respect to Q

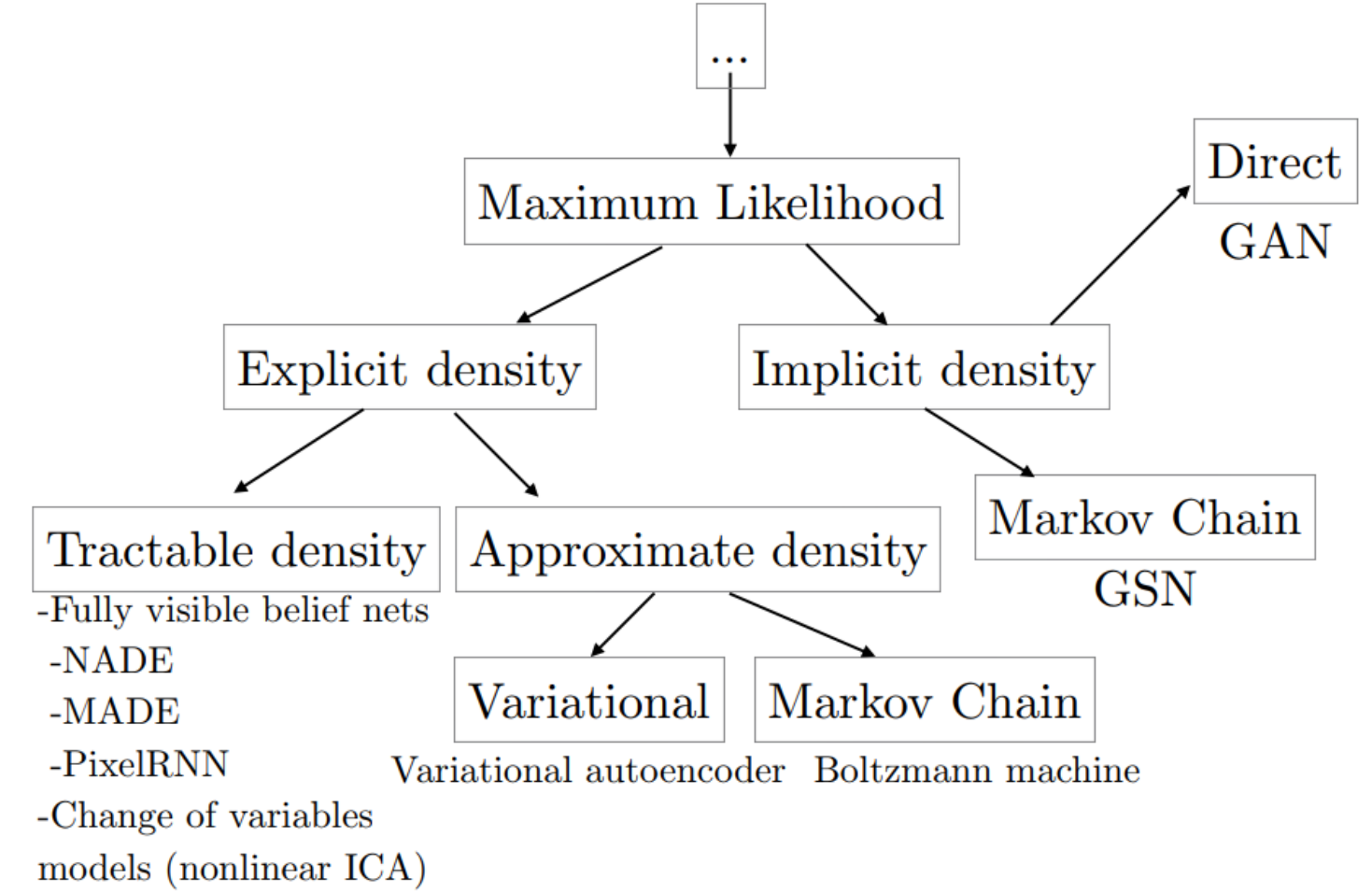
Challenges

Density estimation is a core problem in unsupervised learning

Two general ways:

- Explicit density estimation: explicitly define and solve for $p_{\text{model}}(x)$
- Implicit density estimation: learn model that can sample from $p_{\text{model}}(x)$ without explicitly defining it

Taxonomy of generative models



Lecture plan

- Task of geniting new objects
- PixelCNN and PixelRNN
- Variational autoencoders
- Generative adversarial models
- Advances in GANs

Fully visible belief network

Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-D distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

probability of objects (images)

conditional probability of observing features (pixels)

After this we maximize likelihood of observed data.

Fully visible belief network

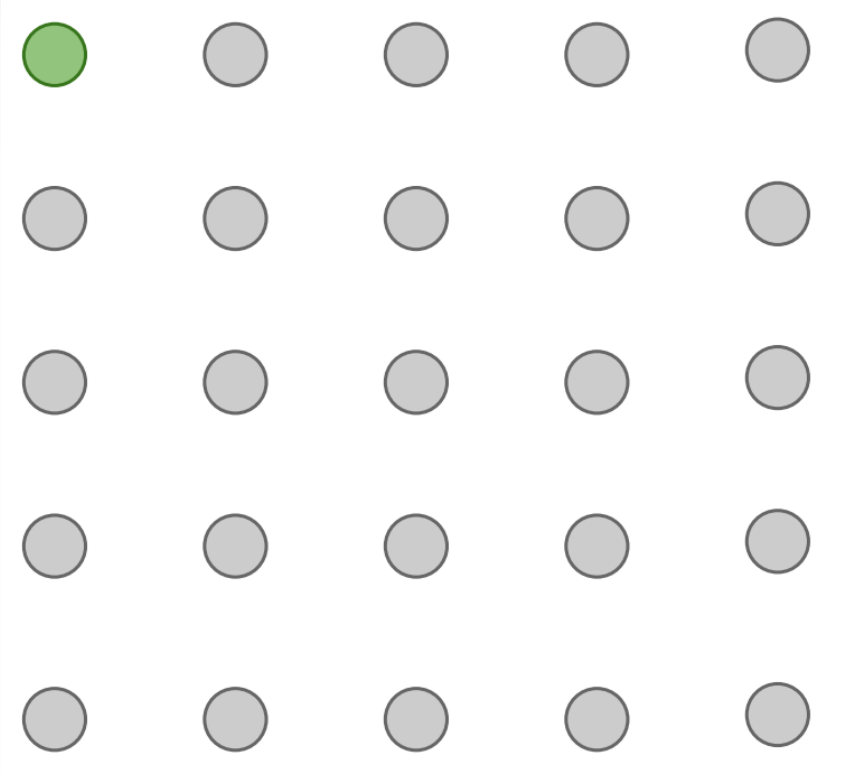
We can express the probabilities using neural networks

In order to do this, we need define probabilities of “previous” pixels

PixelRNN

Generate image pixels starting from left corner

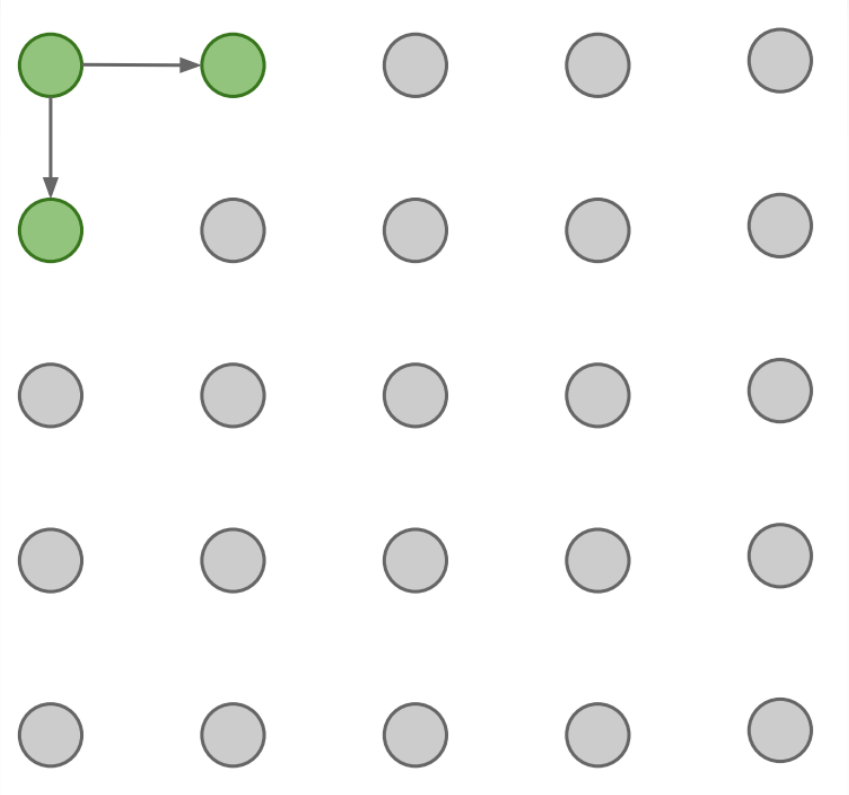
Dependence on previous pixels is modeled with RNNs (LSTMs)



PixelRNN

Generate image pixels starting from left corner

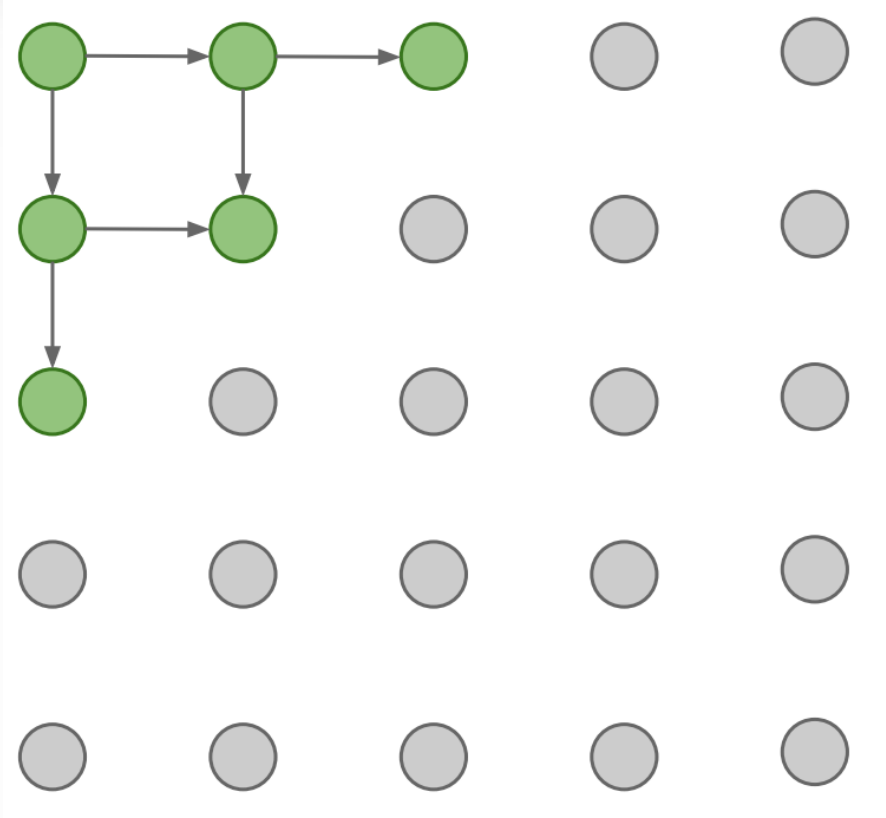
Dependence on previous pixels is modeled with RNNs (LSTMs)



PixelRNN

Generate image pixels starting from left corner

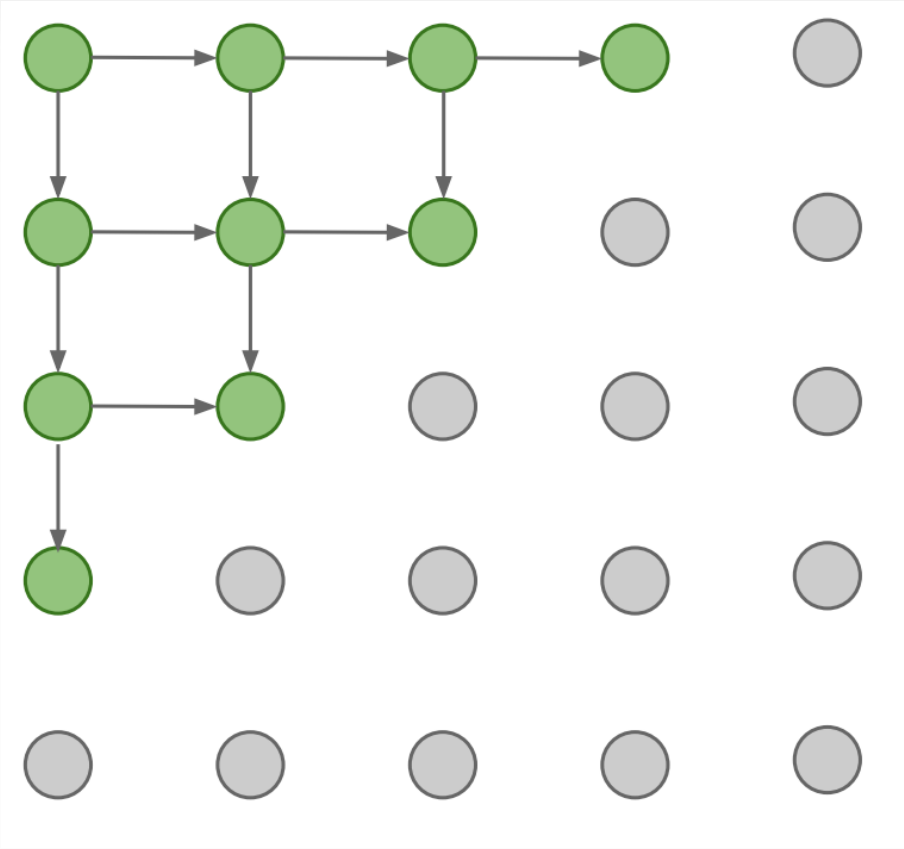
Dependence on previous pixels is modeled with RNNs (LSTMs)



PixelRNN

Generate image pixels starting from left corner

Dependence on previous pixels is modeled with RNNs (LSTMs)



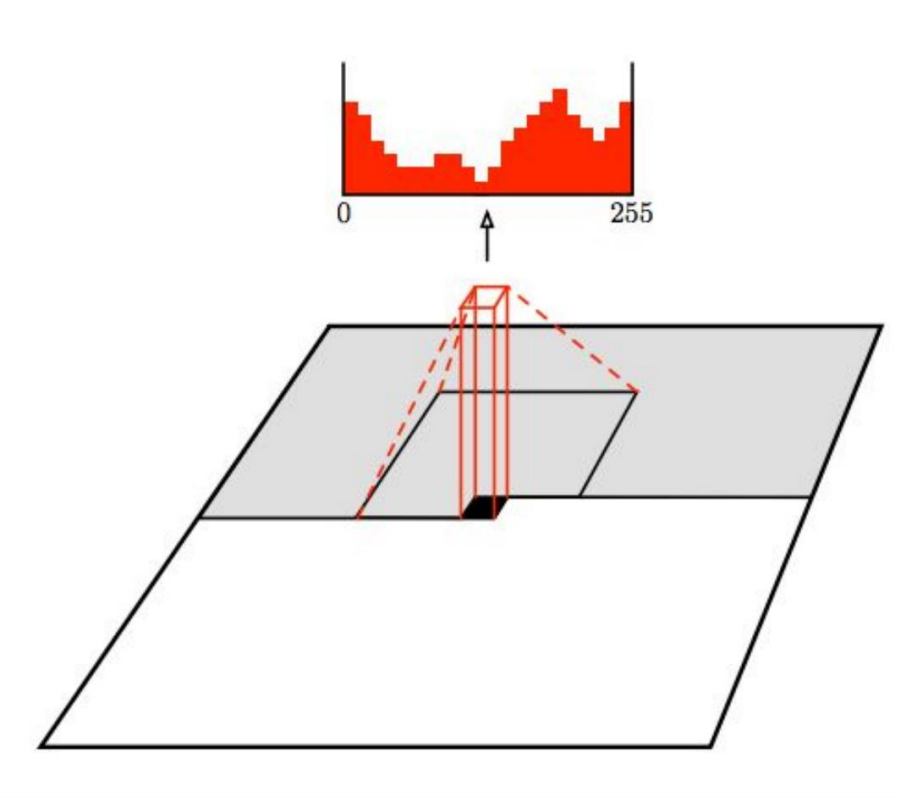
PixelCNN

RNN computation is too slow.

Instead of using RNN, we can use CNN.

Still generate image pixels staring from corner

Dependency on previous pixels is modeled using a CNN over context region



PixelCNN

Training is faster the PixelRNN.

Generation must still proceed sequentially, which makes it still slow.

Pixel*NN analysis

Advantages:

- Can explicitly compute likelihood $p(x)$

- Explicit likelihood of training data gives good evaluation metric

- Good samples

Disadvantages:

- Sequential generation is slow

Improving PixelCNN performance

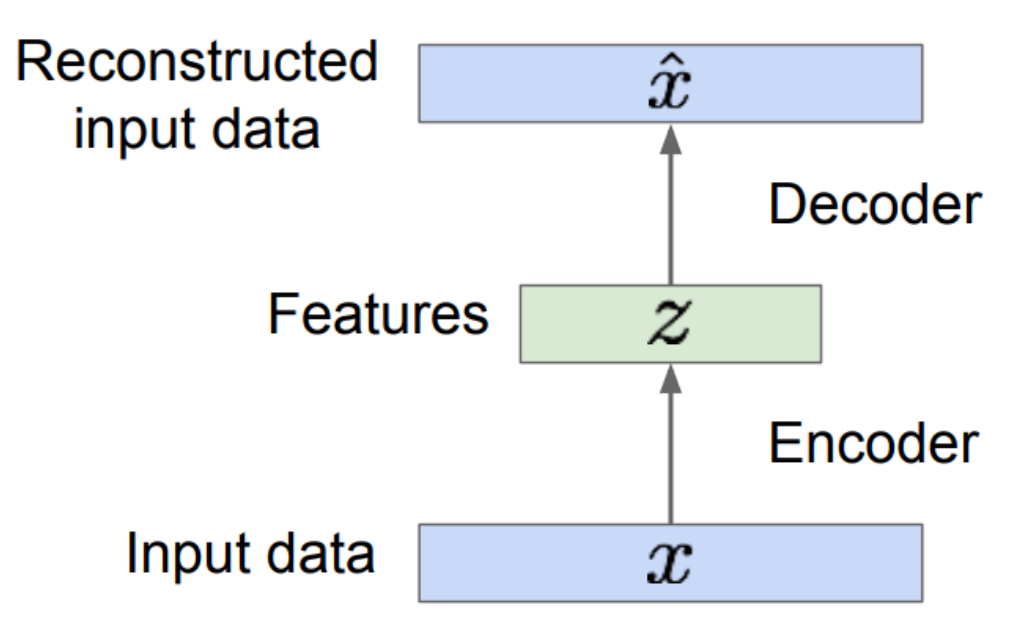
- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks

Lecture plan

- Task of geniting new objects
- PixelCNN and PixelRNN
- Variational autoencoders
- Generative adversarial models
- Advances in GANs

Autoencoders (reminder)

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data



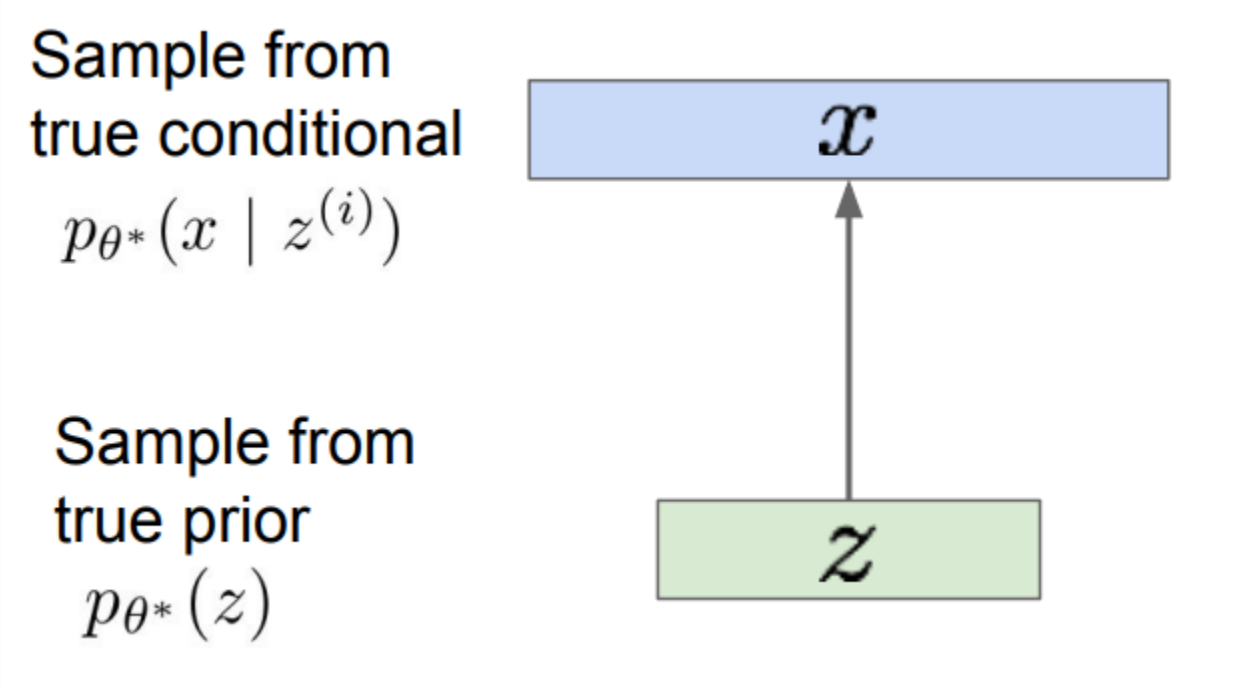
Main idea

Instead of using some assumptions on how a probabilistic model structure should look like, we define intractable density function with some latent variable z:

$$p_\theta(x) = \int p_\theta(x|z)p_\theta(z)dz$$

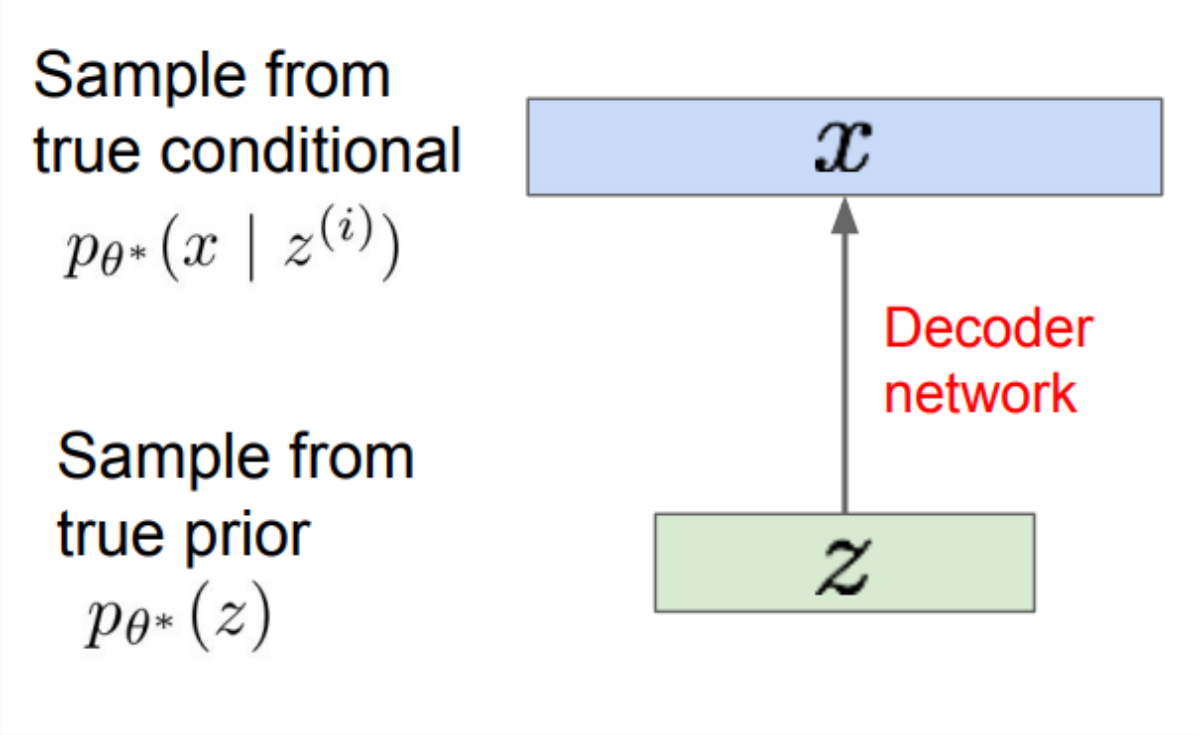
Hidden variables

Assume training data is generated from underlying unobserved representation z space, which is simple enough



Representing $p(x|z)$

Conditional $p(x|z)$ is complex, and we will represent it with a neural network



Learning parameters

$$p_\theta(x) = \int p_\theta(x|z)p_\theta(z)dz$$

$$\theta^* = \arg\min_\theta -\log \int p_\theta(x|z)p_\theta(z)dz$$

What is the problem then?

Learning parameters

$$p_\theta(x) = \int p_\theta(x|z)p_\theta(z)dz$$

$$\theta^* = \arg\min_\theta -\log \int p_\theta(x|z)p_\theta(z)dz$$

$\int p_\theta(x|z)p_\theta(z)dz$ is intractable!

Learning parameters

$$p_\theta(x) = \int p_\theta(x|z)p_\theta(z)dz$$

$$\theta^* = \arg\max_\theta \log \int p_\theta(x|z)p_\theta(z)dz$$

$\int p_\theta(x|z)p_\theta(z)dz$ is intractable!

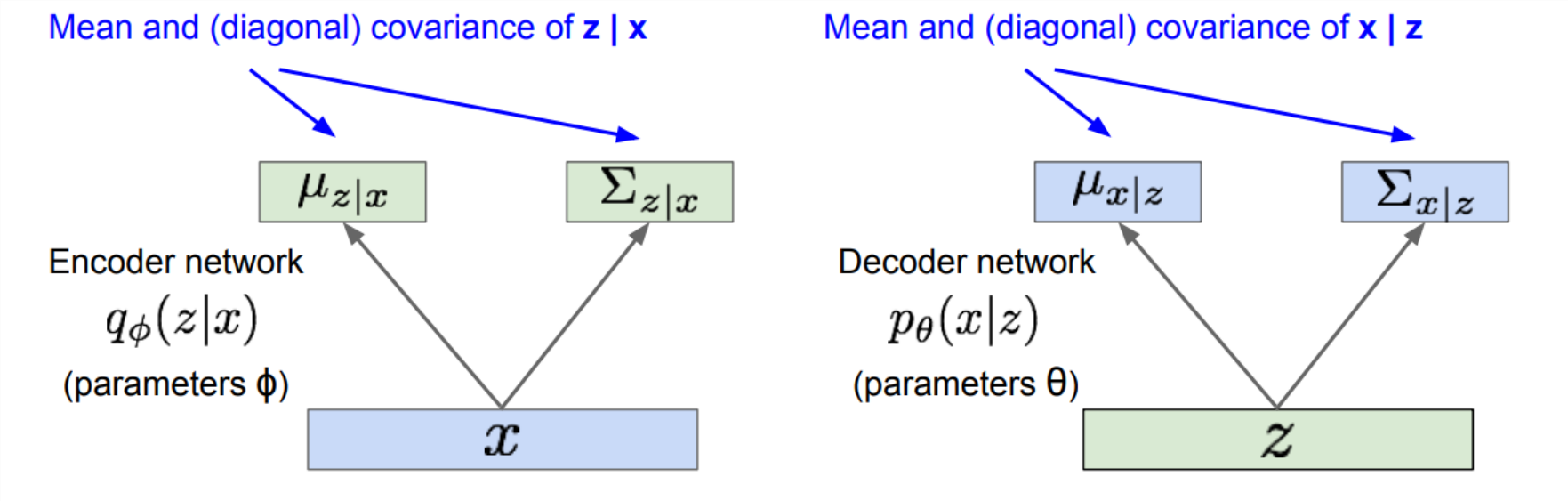
Posterior data likelihood is also intractable!

Encoder network

Idea: add encoder network $q_\phi(z|x)$ approximating $p_\phi(z|x)$

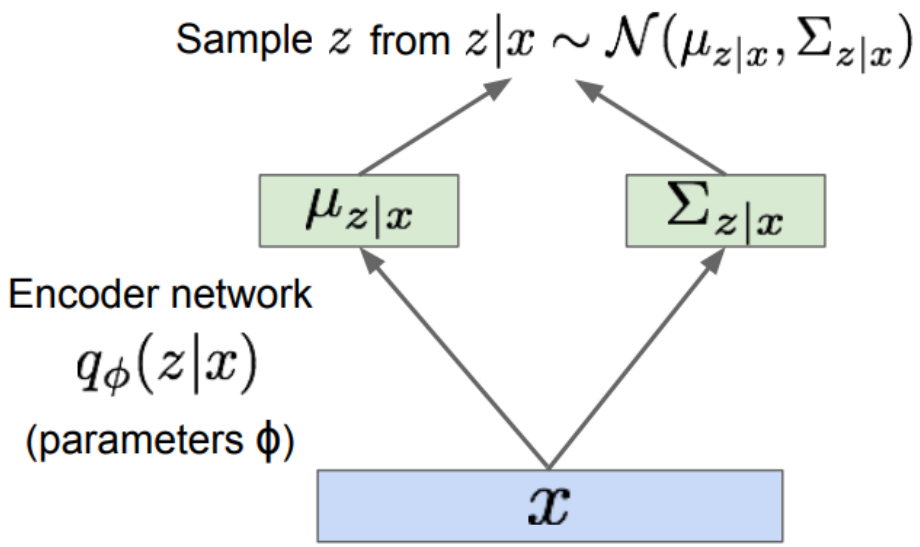
Encoder and decoder

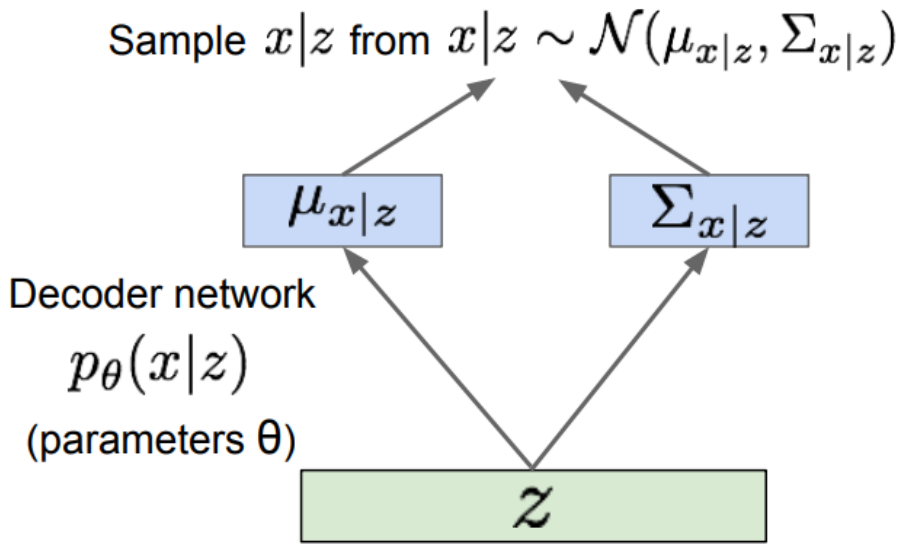
Encoder and decoder are probabilistic, both inferring hyperparameters of distribution (say, Gaussian)



Sampling with encoder and decoder

We can sample z using encoder and $x|z$ using decoder.





Coming back to likelihood

$$\theta^* = \arg\max_\theta \log p_\theta(x)$$

$$\log p_\theta(x^{(i)}) = \mathrm{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta\left(x^{(i)}\right)\right] =$$

$$= \mathrm{E}_z\left[\log \frac{p_\theta\left(x^{(i)}|z\right)p_\theta(z)}{p_\theta(z|x^{(i)})}\right] =$$

$$= \mathrm{E}_z\left[\log \frac{p_\theta\left(x^{(i)}|z\right)p_\theta(z)}{p_\theta(z|x^{(i)})}\frac{q_\phi\left(z|x^{(i)}\right)}{q_\phi(z|x^{(i)})}\right] =$$

$$= \mathrm{E}_z\left[\log p_\theta\left(x^{(i)}|z\right)\right] - \mathrm{E}_z\left[\log \frac{q_\phi\left(z|x^{(i)}\right)}{p_\theta(z)}\right] + \mathrm{E}_z\left[\log \frac{q_\phi\left(z|x^{(i)}\right)}{p_\theta(z|x^{(i)})}\right] =$$

$$= \mathrm{E}_z\left[\log p_\theta\left(x^{(i)}|z\right)\right] - D_{KL}\left(q_\phi\left(z|x^{(i)}\right)||p_\theta(z)\right)$$

$$+ D_{KL}\left(q_\phi\left(z|x^{(i)}\right)||p_\theta\left(z|x^{(i)}\right)\right)$$

Lower bounds

$$\mathrm{E}_z\left[\log p_\theta\left(x^{(i)}|z\right)\right] - D_{KL}\left(q_\phi\left(z|x^{(i)}\right)||p_\theta(z)\right) + D_{KL}\left(q_\phi\left(z|x^{(i)}\right)||p_\theta\left(z|x^{(i)}\right)\right)$$

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling.

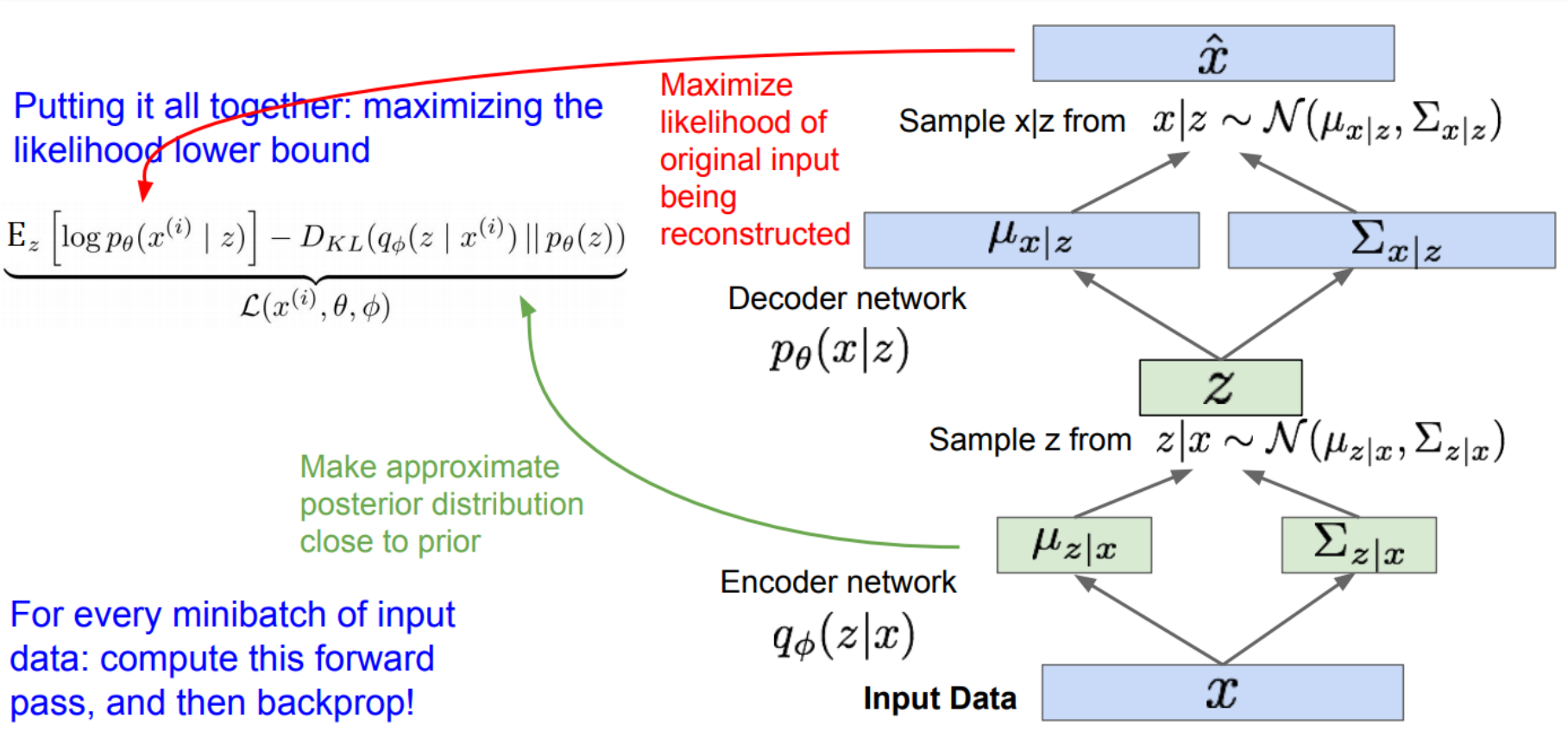
This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term :(But we know KL divergence always >= 0.

$$\underbrace{\mathrm{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

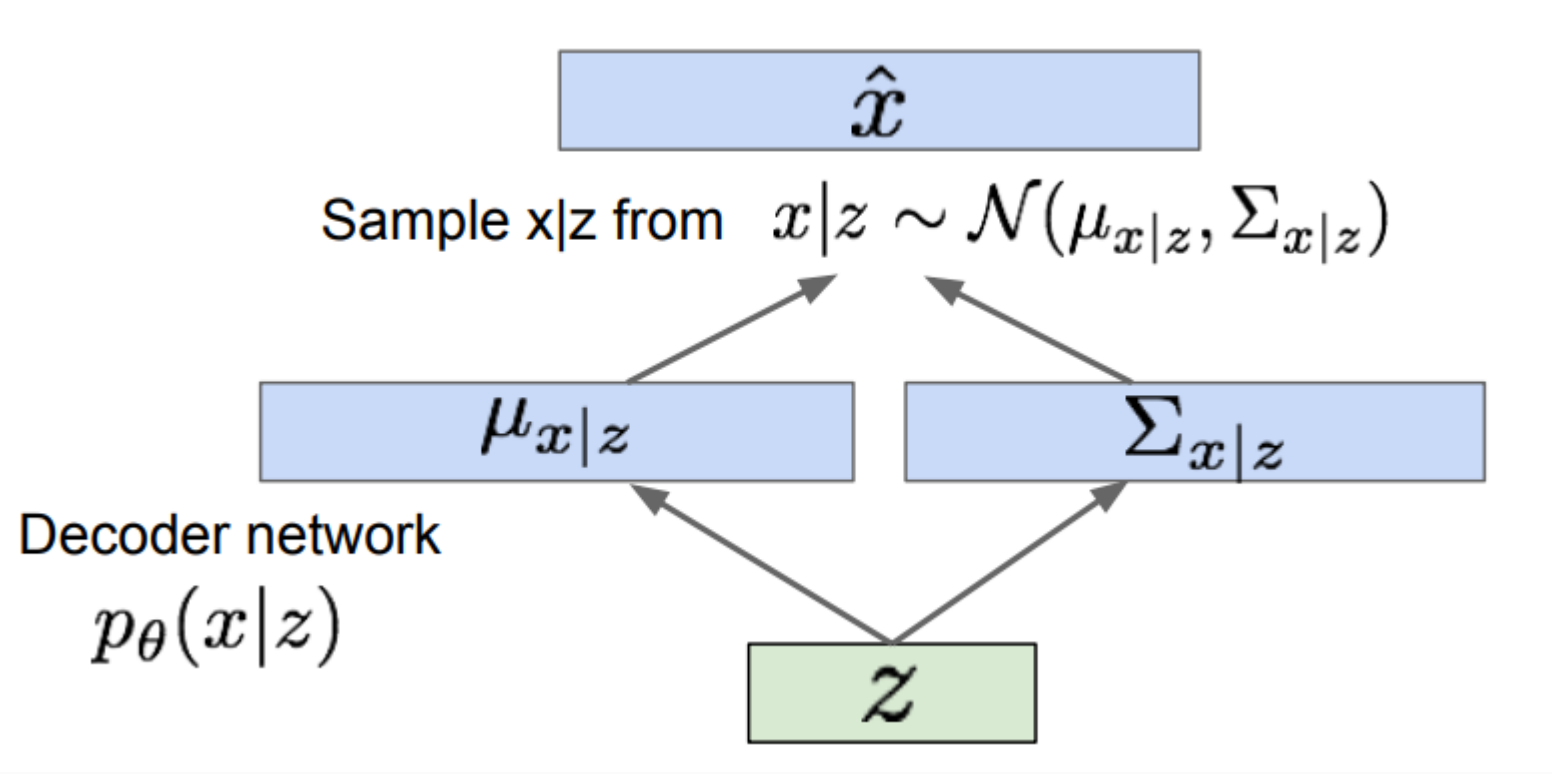
Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

Learning VAE



Generating data with VAE

Just sample with decoder



VAE analysis

Advantages:

- Principled approach to generative models
- Allows inference of $q(z|x)$, can be useful feature representation for other tasks

Disadvantages:

- Maximizing lower bound of likelihood is not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

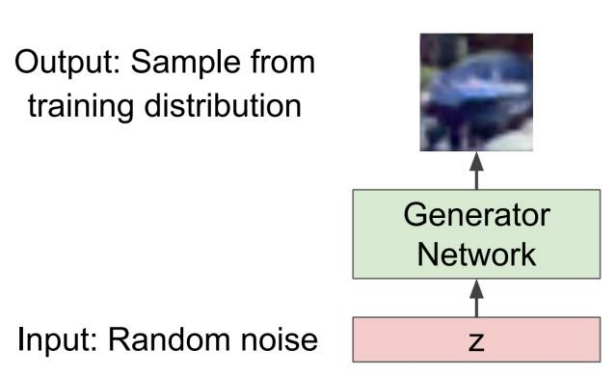
Lecture plan

- Task of geniting new objects
- PixelCNN and PixelRNN
- Variational autoencoders
- Generative adversarial models
- Advances in GANs

Main ideas

First, skip idea of working with distributions explicitly

Second, learn transformation from random noise to the training sample distribution



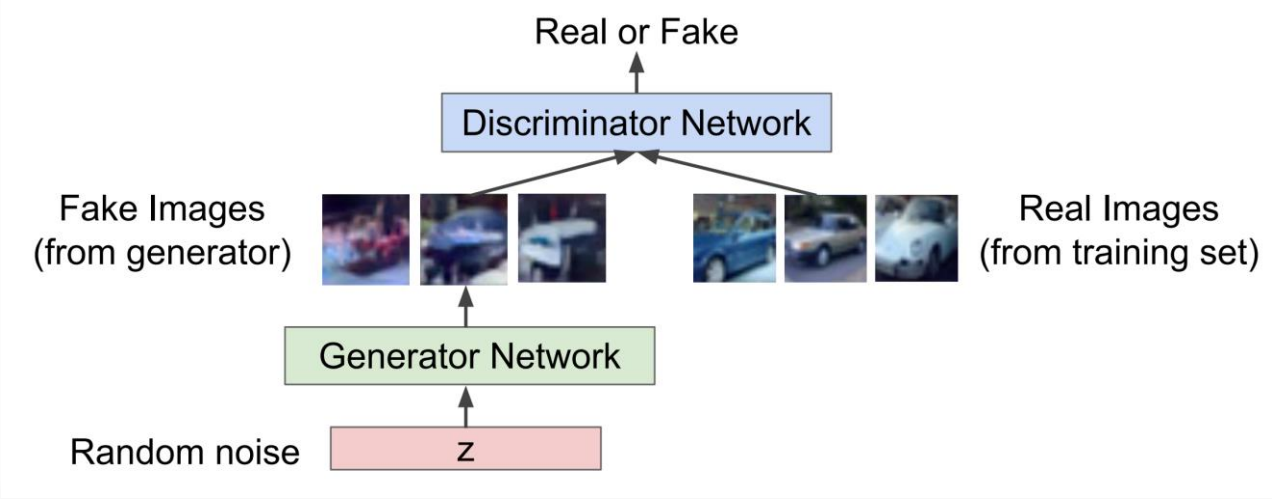
Third, use game-theoretic approach

Generator vs Discriminator

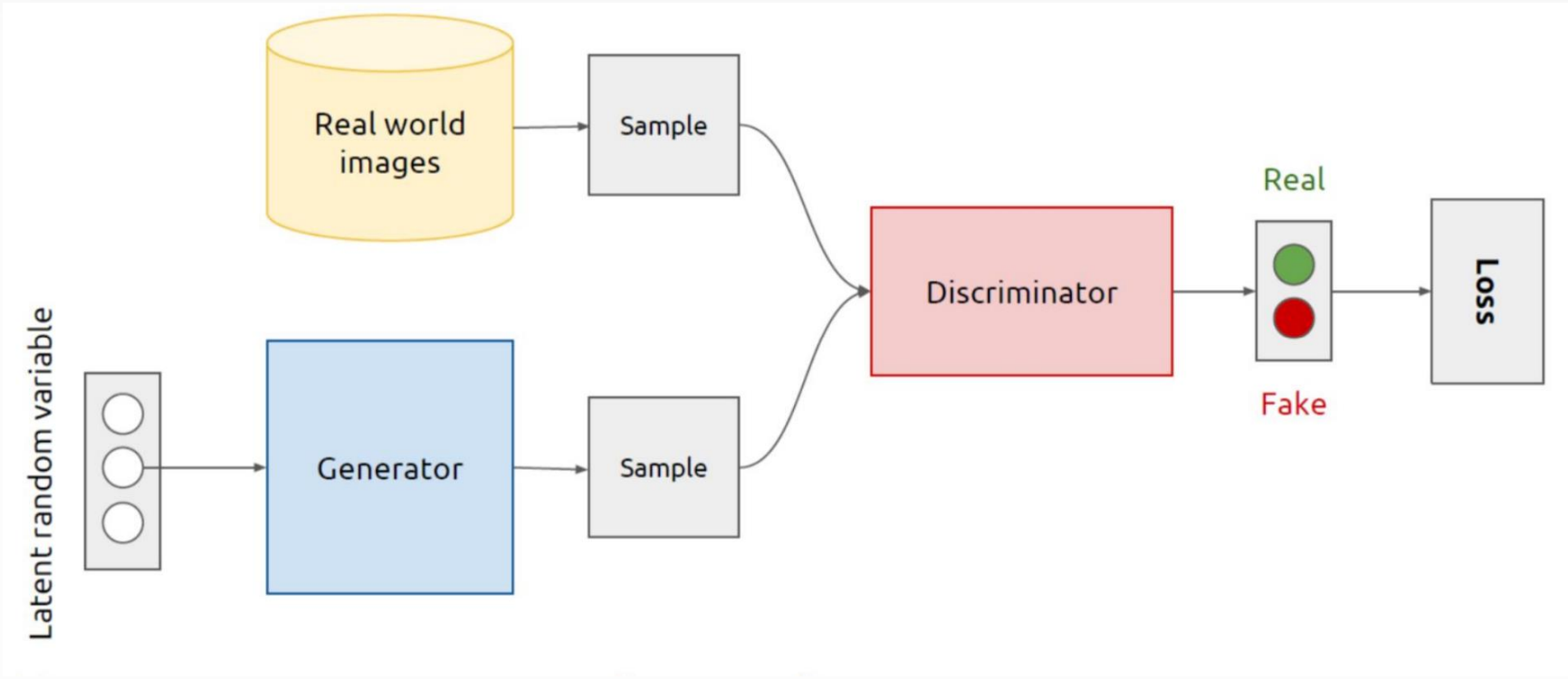
Training GANs is a two-player game

Generator network tries to fool the discriminator by generating real-looking images

Discriminator network tries to distinguish between real and fake images



Learning GANs



Minimax training

We can train them jointly in minimax game

Minimax objective function:

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

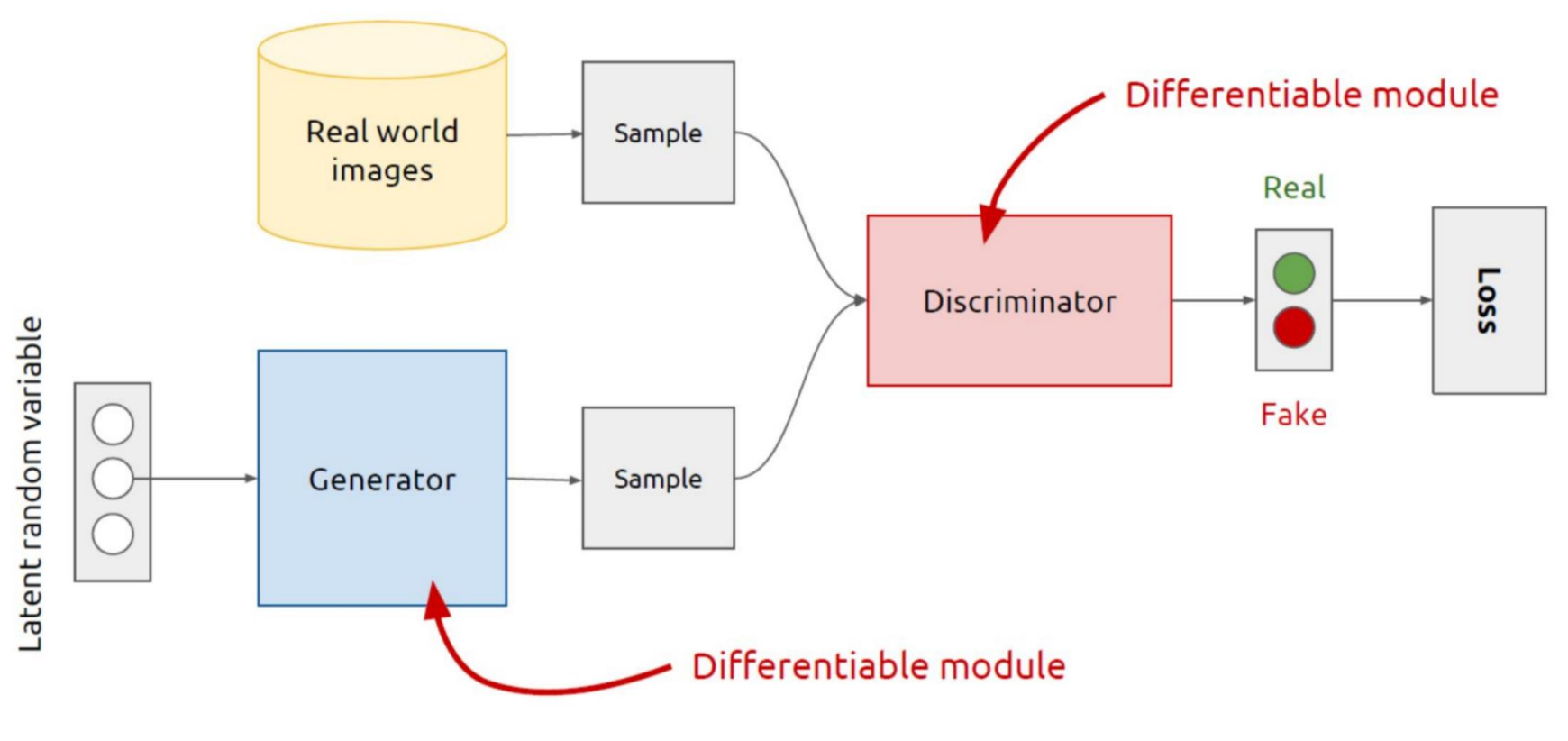
where D_{θ_d} is discriminator with parameters θ_d

(wants to maximize objective such that $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake))

and G_{θ_g} is generator with parameters θ_g

(wants to minimize objective such that D(G(z)) is close to 1)

Learning GANs



Training GANs

Alternate between:

1. **Gradient ascent** on discriminator

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

2. **Gradient descent** on generator

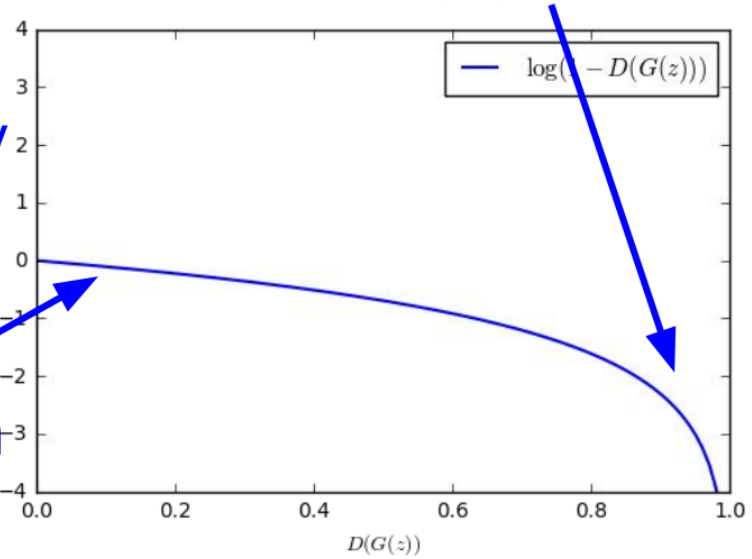
$$\min_{\theta_g} \mathrm{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Training GANs

In practice, it doesn't work well, because they look in the same direction

Gradient signal dominated by region where sample is already good

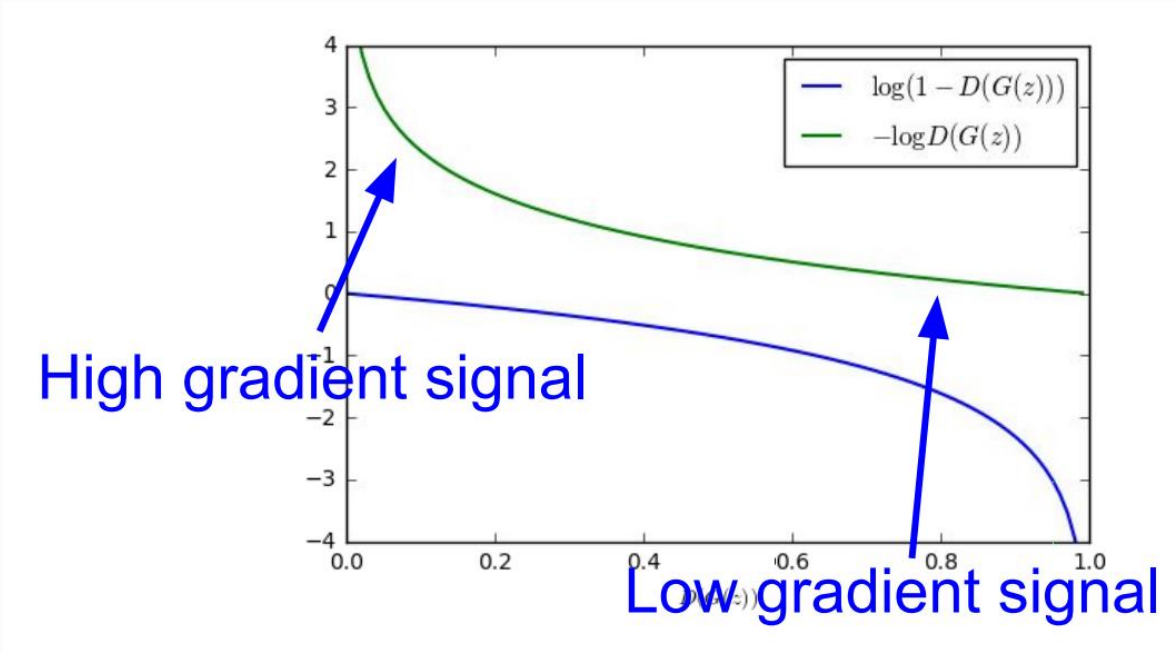
When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



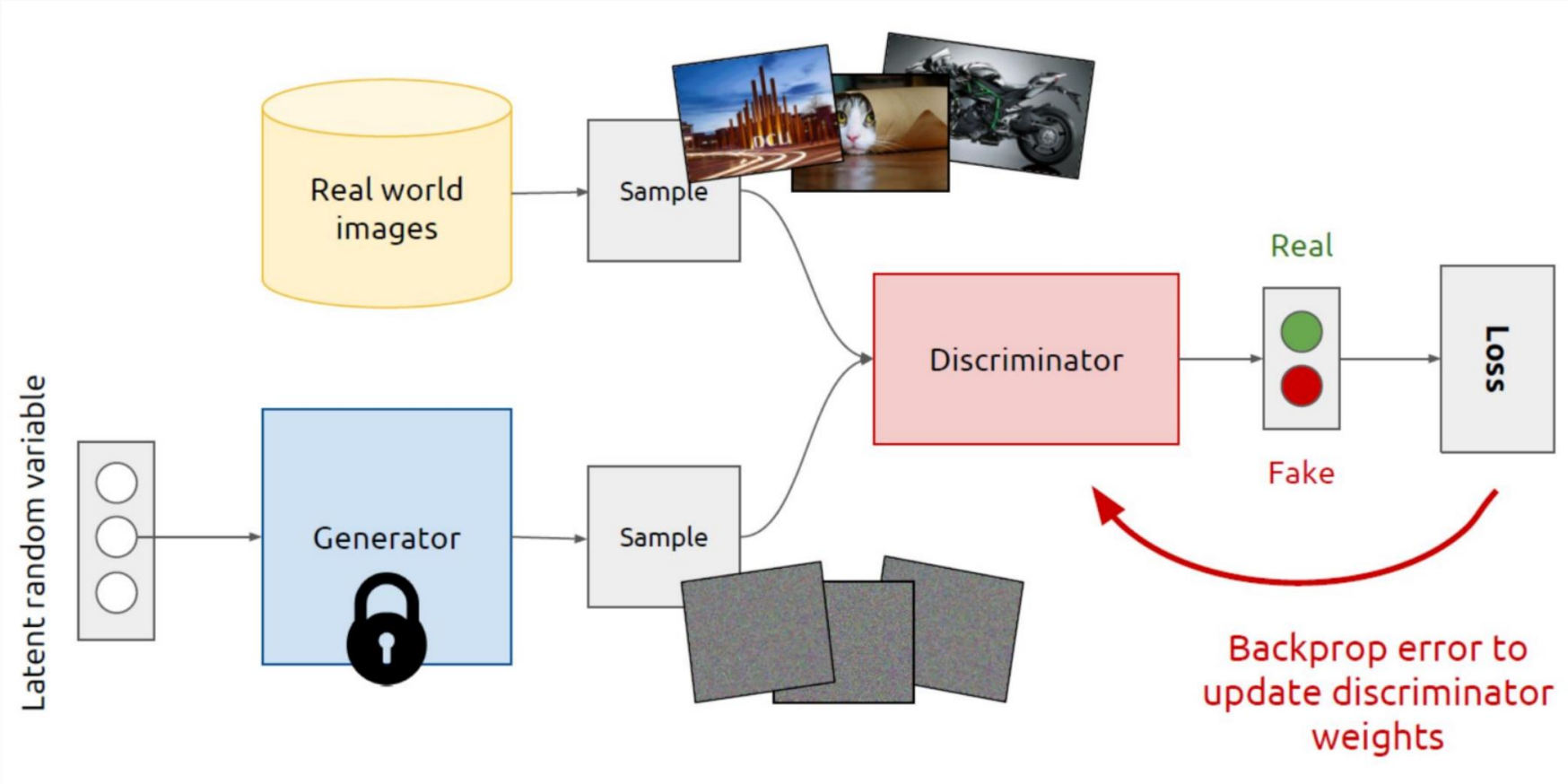
Changing generator learning

Use **gradient ascent** on generator instead

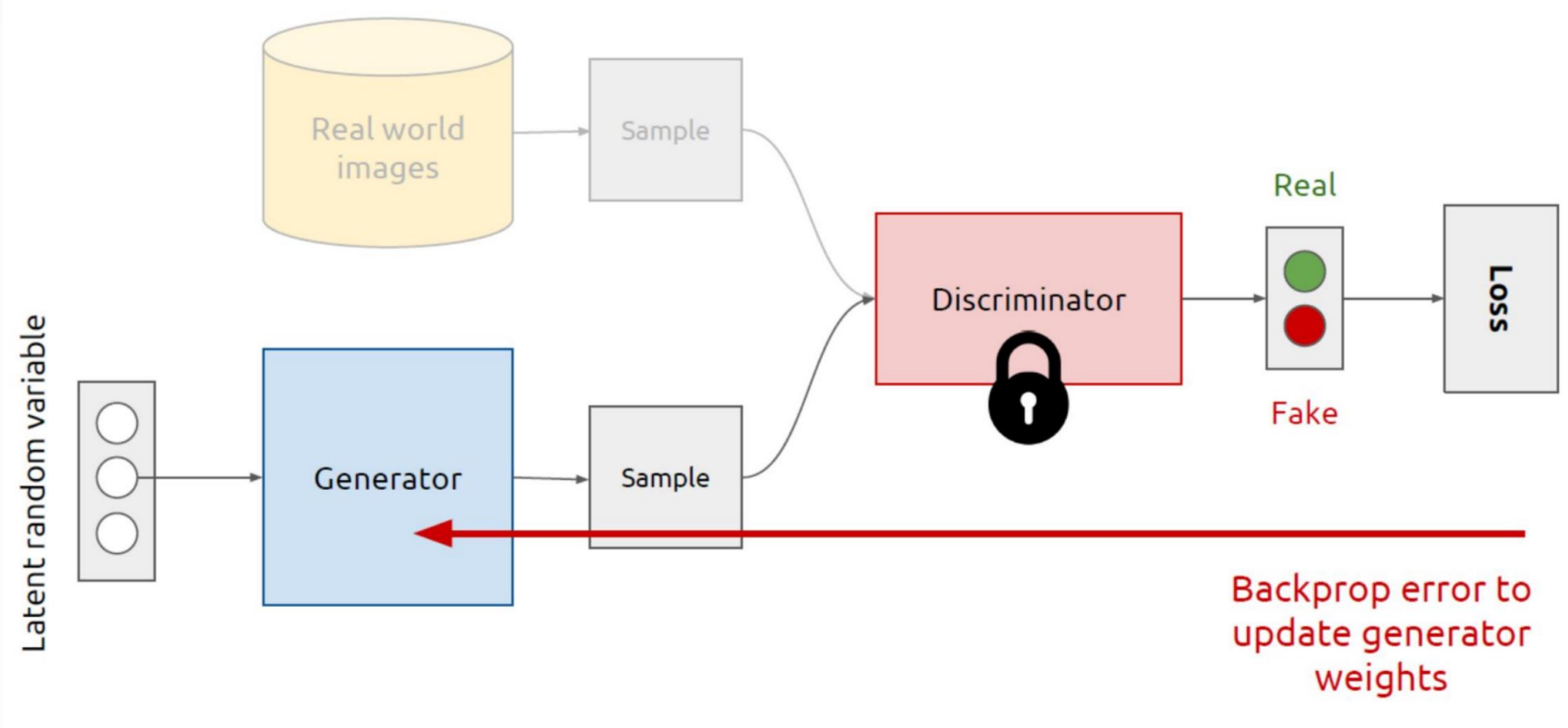
$$\max_{\theta_g} \mathrm{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$



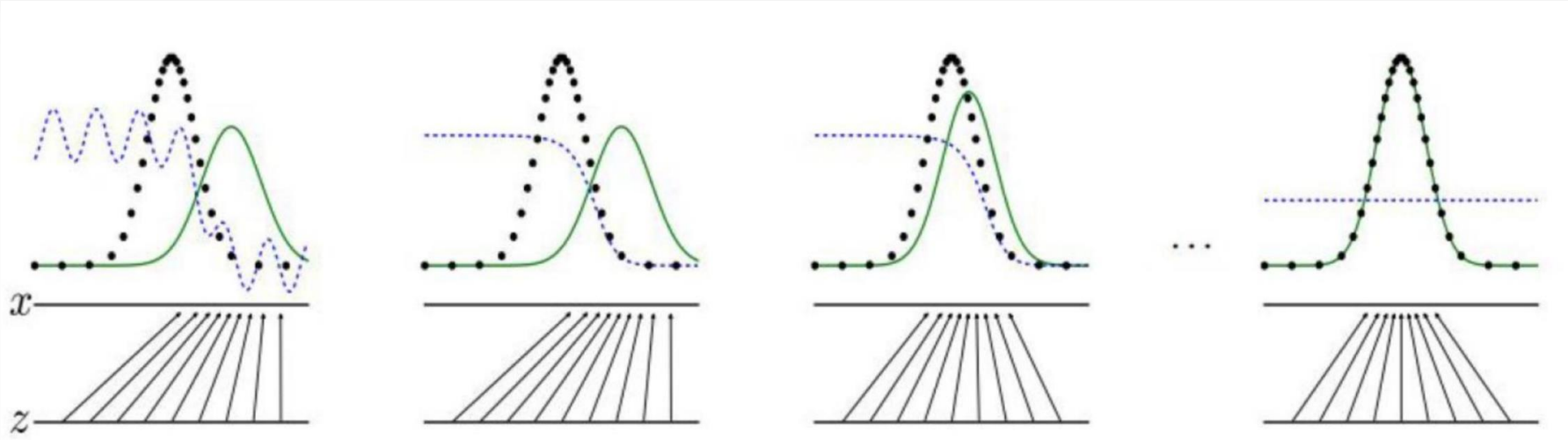
Discriminator learning step



Generator learning step



Changing generator learning



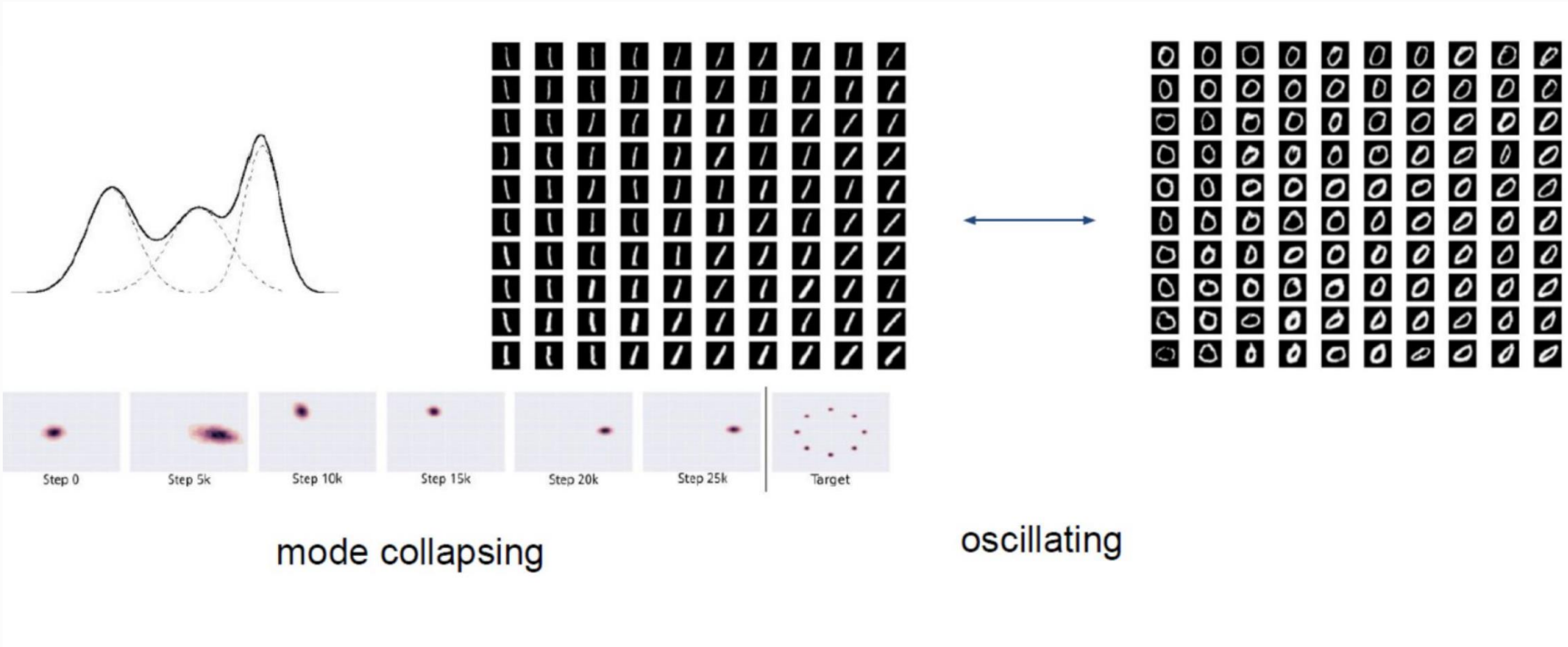
We continue ascent steps until convergence

However, we may not see convergence

GANs drawbacks

- No guarantee to equilibrium
  - Mode collapsing
  - Oscillation
  - No indicator when to finish

Collapsing or oscillating



Lecture plan

- Task of geniting new objects
- PixelCNN and PixelRNN
- Generative adversarial models
- Variational autoencoders
- Advances in GANs

Conditional GANs

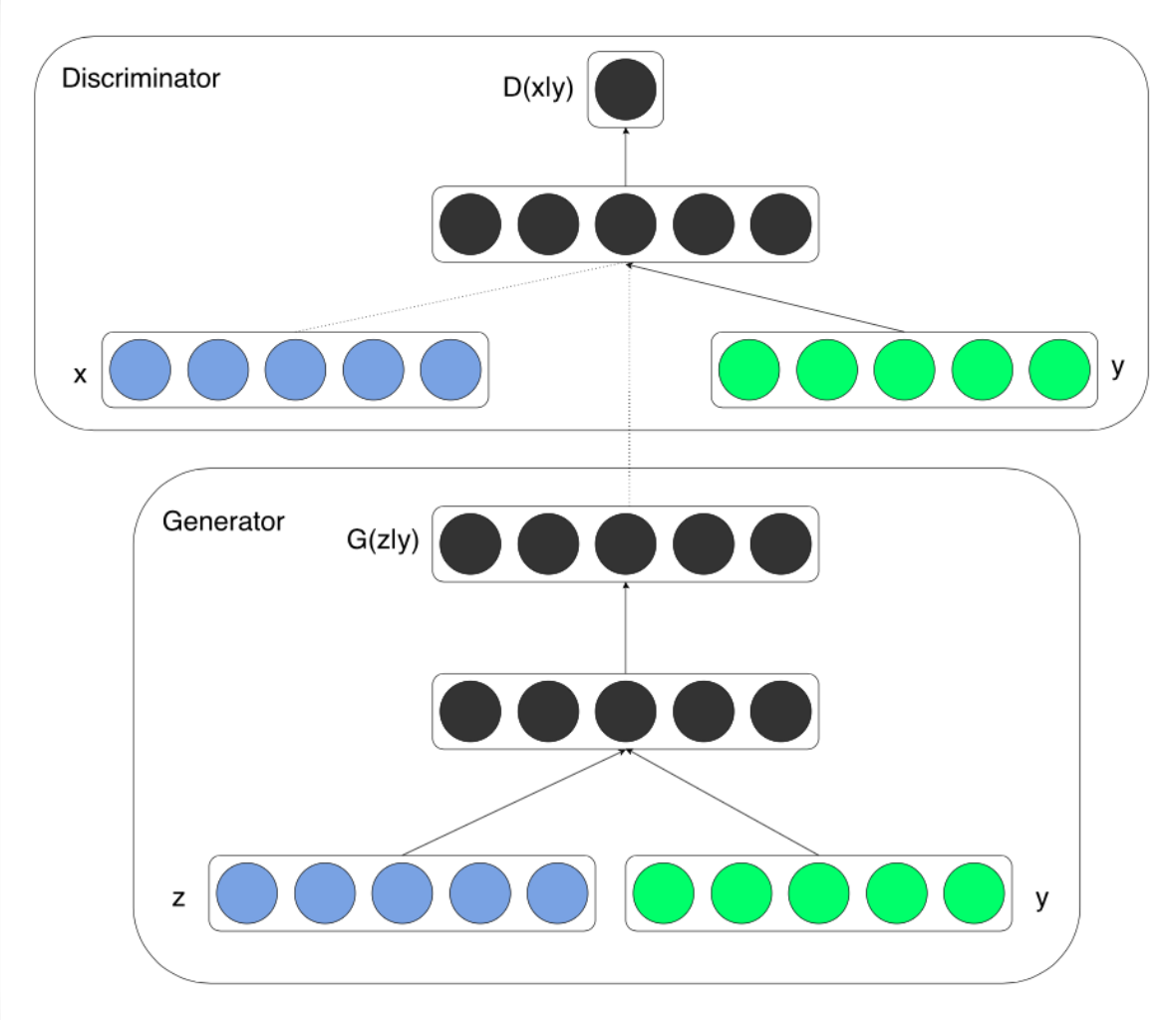
- Idea is to add some labels, so discriminator can work as a classifier with respect to some labeling
- In this case, we have different distribution for each class

Conditional GANs

- Objective function is then looks like this:

$$\min_{\theta_g} \max_{\theta_d} \Big[\mathrm{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x, y) + \mathrm{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z, y), y)) \Big]$$

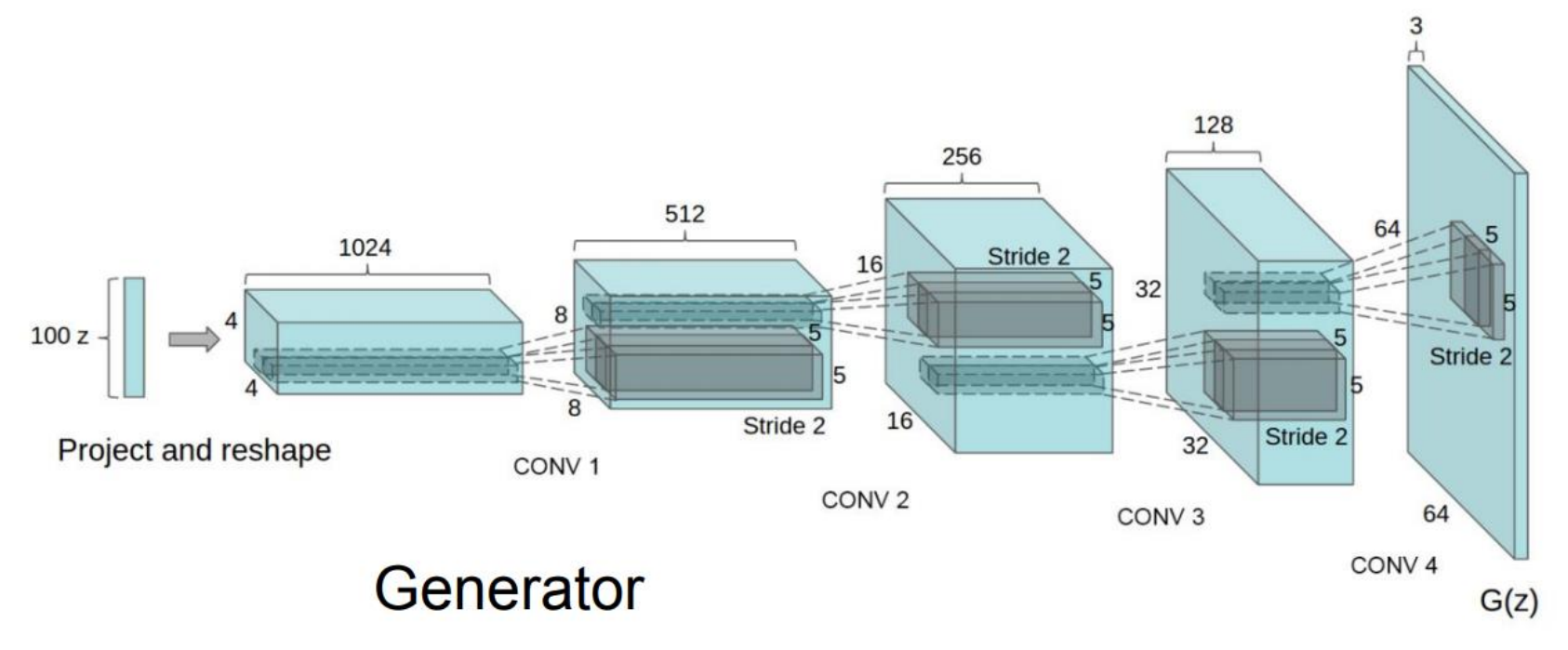
CGANs



DCGAN improvement

- Replace all pooling layers with strided convolutions (discriminators) and fractional-strided convolutions (generator)
- Use batchnorm in both the generator and the discriminator
- Remove fully connected hidden layers
- Use ReLU in generator for all the layers and tanh for the output
- Use LeakyReLU activation in the discriminator for all layers

DCGAN


100 z
Project and reshape
4
4
1024
8
8
512
Stride 2
CONV 1
16
16
256
Stride 2
CONV 2
32
32
128
Stride 2
CONV 3
64
64
3
Stride 2
CONV 4
G(z)
5
Generator

State-of-the-art on 12th Dec 2018