Latent Variable Models

Designing, Visualizing and Understanding Deep Neural Networks

CS W182/282A

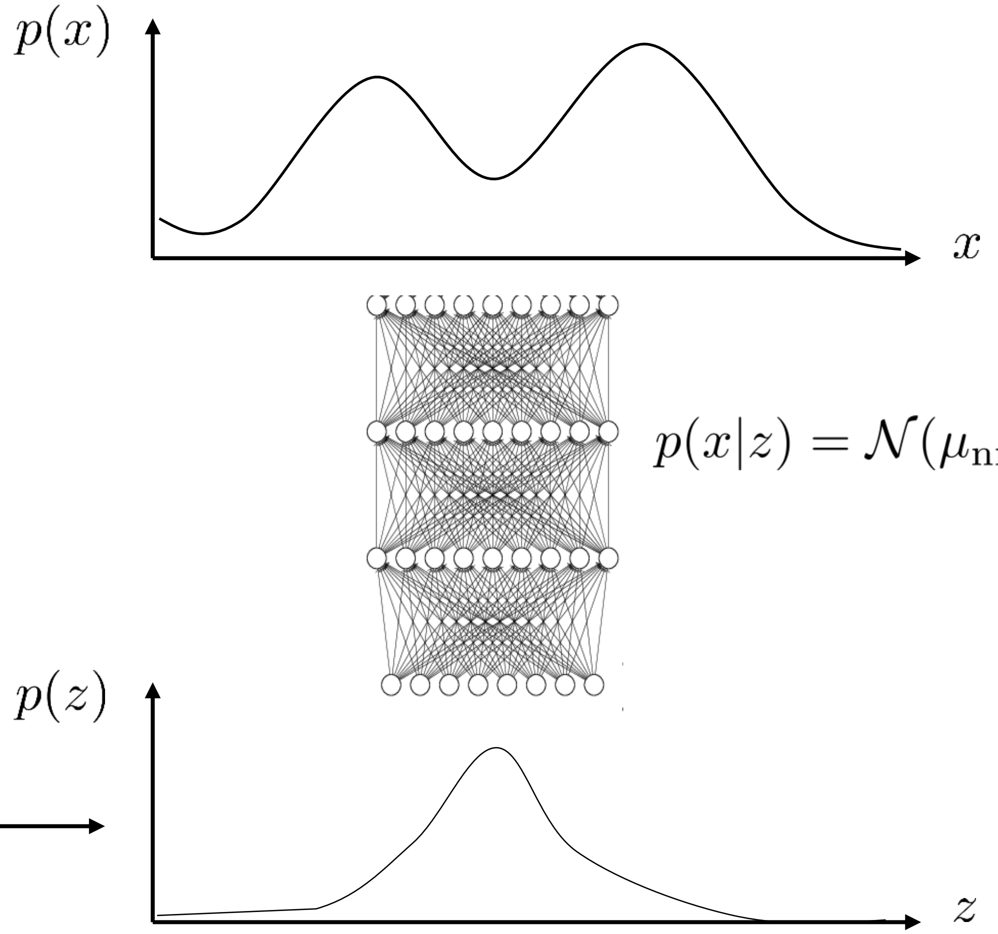
Instructor: Sergey Levine
UC Berkeley

Latent variable models in general

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

"easy" distribution
(e.g., conditional Gaussian)

"easy" distribution
(e.g., Gaussian)



$$p(x|z) = \mathcal{N}(\mu_{\text{nn}}(z), \sigma_{\text{nn}}(z))$$

"easy" distribution
(e.g., Gaussian)

Estimating the log-likelihood

expected log-likelihood:

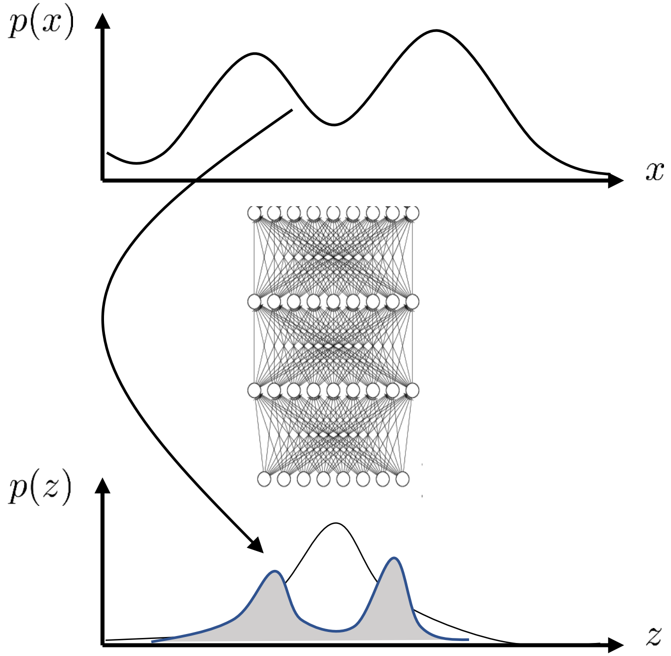
$$\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_i E_{z \sim p(z|x_i)}[\log p_\theta(x_i, z)]$$

but... how do we calculate $p(z|x_i)$?

this is called **probabilistic inference**

intuition: "guess" most likely z given x_i, and pretend it's the right one

...but there are many possible values of z so use the distribution $p(z|x_i)$



The variational approximation

but... how do we calculate $p(z|x_i)$?

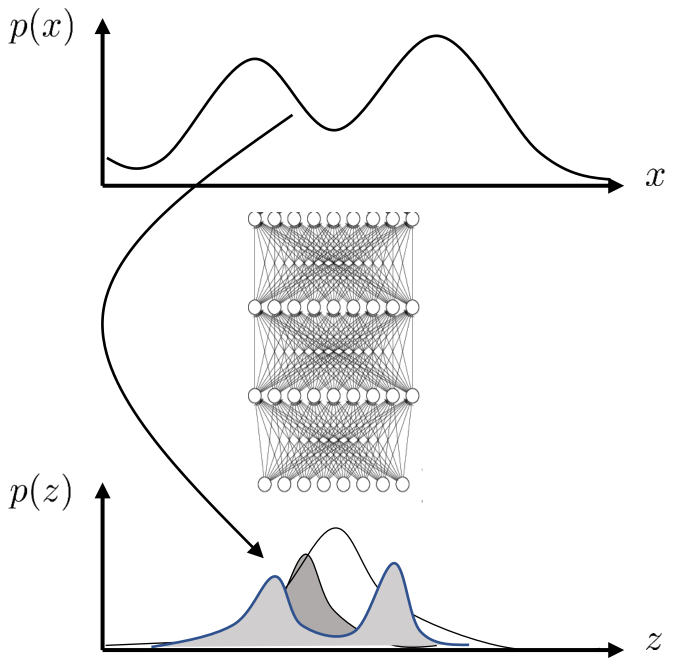
what if we approximate with $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$

can bound $\log p(x_i)$!

$$\log p(x_i) = \log \int_z p(x_i|z)p(z)$$

$$= \log \int_z p(x_i|z)p(z)\frac{q_i(z)}{q_i(z)}$$

$$= \log E_{z\sim q_i(z)}\left[\frac{p(x_i|z)p(z)}{q_i(z)}\right]$$



The variational approximation

Jensen's inequality

$$\log E[y] \geq E[\log y]$$

but… how do we calculate $p(z|x_i)$?

can bound $\log p(x_i)$!

$$\log p(x_i) = \log \int_z p(x_i|z)p(z)$$

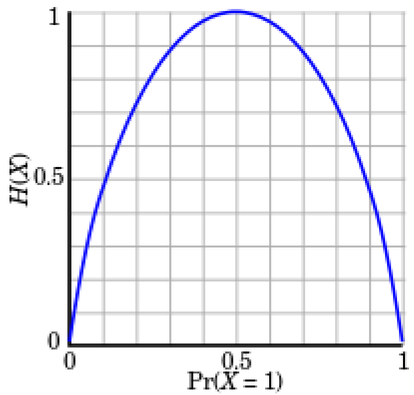
$$= \log \int_z p(x_i|z)p(z)\frac{q_i(z)}{q_i(z)}$$

$$= \log E_{z\sim q_i(z)}\left[\frac{p(x_i|z)p(z)}{q_i(z)}\right]$$

maximizing this maximizes $\log p(x_i)$

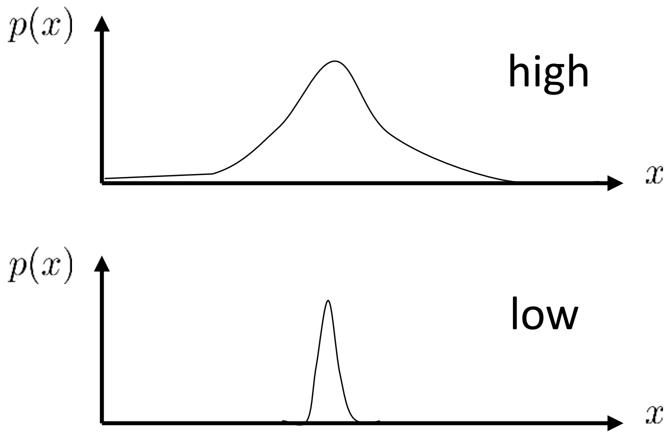
$$\geq E_{z\sim q_i(z)}\left[\log \frac{p(x_i|z)p(z)}{q_i(z)}\right] = E_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] + \mathcal{H}(q_i) \; E_{z\sim q_i(z)}[\log q_i(z)]$$

A brief aside...



Entropy:

$$\mathcal{H}(p) = -E_{x \sim p(x)}[\log p(x)] = -\int_x p(x) \log p(x) dx$$

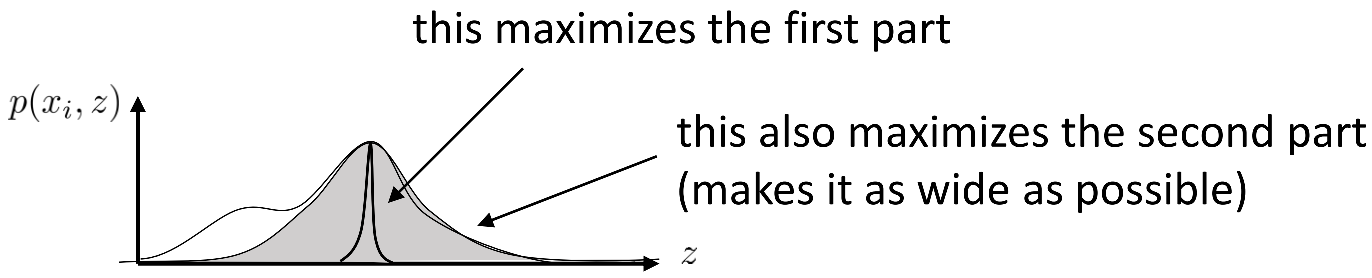


Intuition 1: how *random* is the random variable?

Intuition 2: how large is the log probability in expectation *under itself*

what do we expect this to do?

$$E_{z \sim q_i(z)}[\log p(x_i|z) + \log p(z)] + \mathcal{H}(q_i)$$



A brief aside...

KL-Divergence:

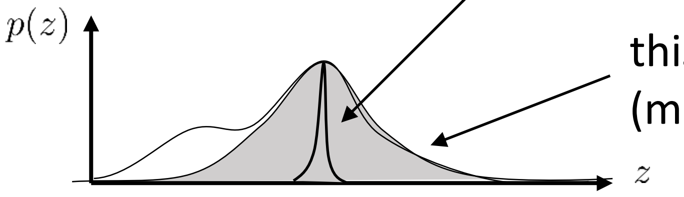
$$D_{\mathrm{KL}}(q\|p) = E_{x\sim q(x)}\left[\log\frac{q(x)}{p(x)}\right] = E_{x\sim q(x)}[\log q(x)] - E_{x\sim q(x)}[\log p(x)] = -E_{x\sim q(x)}[\log p(x)] - \mathcal{H}(q)$$

Intuition 1: how *different* are two distributions?

Intuition 2: how small is the expected log probability of one distribution under another, minus entropy?

why entropy?

this maximizes the first part



this also maximizes the second part
(makes it as wide as possible)

The variational approximation

$$\log p(x_i) \geq \overbrace{E_{z \sim q_i(z)}[\log p(x_i|z) + \log p(z)] + \mathcal{H}(q_i)}^{\mathcal{L}_i(p, q_i)}$$

what makes a good $q_i(z)$?

intuition: $q_i(z)$ should approximate $p(z|x_i)$

approximate in what sense?

compare in terms of KL-divergence: $D_{\mathrm{KL}}(q_i(z)\|p(z|x))$

why?

$$D_{\mathrm{KL}}(q_i(z)\|p(z|x_i)) = E_{z \sim q_i(z)}\left[\log \frac{q_i(z)}{p(z|x_i)}\right] = E_{z \sim q_i(z)}\left[\log \frac{q_i(z)p(x_i)}{p(x_i, z)}\right]$$

$$= -E_{z \sim q_i(z)}[\log p(x_i|z) + \log p(z)] + E_{z \sim q_i(z)}[\log q_i(z)] + E_{z \sim q_i(z)}[\log p(x_i)]$$

$$= -E_{z \sim q_i(z)}[\log p(x_i|z) + \log p(z)] - \mathcal{H}(q_i) + \log p(x_i)$$

$$= -\mathcal{L}_i(p, q_i) + \log p(x_i)$$

$$\log p(x_i) = D_{\mathrm{KL}}(q_i(z)\|p(z|x_i)) + \mathcal{L}_i(p, q_i)$$

$$\log p(x_i) \geq \mathcal{L}_i(p, q_i)$$

The variational approximation

$$\log p(x_i) \geq \overbrace{E_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] + \mathcal{H}(q_i)}^{\mathcal{L}_i(p, q_i)}$$

$$\log p(x_i) = D_{\text{KL}}(q_i(z) \| p(z|x_i)) + \mathcal{L}_i(p, q_i)$$

$$\log p(x_i) \geq \mathcal{L}_i(p, q_i)$$

$$D_{\text{KL}}(q_i(z) \| p(z|x_i)) = E_{z\sim q_i(z)}\left[\log \frac{q_i(z)}{p(z|x_i)}\right] = E_{z\sim q_i(z)}\left[\log \frac{q_i(z)p(x_i)}{p(x_i, z)}\right]$$

$$= \underbrace{-E_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] - \mathcal{H}(q_i)}_{-\mathcal{L}_i(p, q_i)} + \log p(x_i)$$

independent of q_i!

$\Rightarrow$ maximizing $\mathcal{L}_i(p, q_i)$ w.r.t. q_i minimizes KL-divergence!

How do we use this?

$$\log p(x_i) \geq \underbrace{E_{z \sim q_i(z)}[\log p_\theta(x_i|z) + \log p(z)] + \mathcal{H}(q_i)}_{\mathcal{L}_i(p, q_i)}$$

$$\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_i \log p_\theta(x_i)$$

$$\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_i \mathcal{L}_i(p, q_i)$$

for each x_i (or mini-batch):

calculate $\nabla_\theta \mathcal{L}_i(p, q_i)$:

sample $z \sim q_i(z)$

$\nabla_\theta \mathcal{L}_i(p, q_i) \approx \nabla_\theta \log p_\theta(x_i|z)$

$\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}_i(p, q_i)$

update q_i to maximize $\mathcal{L}_i(p, q_i)$ ← how?

let's say $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$

use gradient $\nabla_{\mu_i} \mathcal{L}_i(p, q_i)$ and $\nabla_{\sigma_i} \mathcal{L}_i(p, q_i)$

gradient ascent on μ_i, σ_i

What's the problem?

for each x_i (or mini-batch):

- calculate $\nabla_\theta \mathcal{L}_i(p, q_i)$:
  - sample $z \sim q_i(z)$
  - $\nabla_\theta \mathcal{L}_i(p, q_i) \approx \nabla_\theta \log p_\theta(x_i|z)$
- $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}_i(p, q_i)$
- update q_i to maximize $\mathcal{L}_i(p, q_i)$

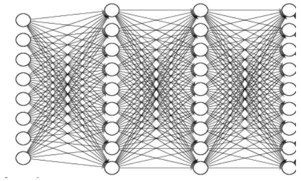
let's say $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$

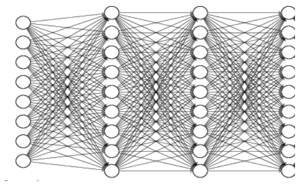
use gradient $\nabla_{\mu_i} \mathcal{L}_i(p, q_i)$ and $\nabla_{\sigma_i} \mathcal{L}_i(p, q_i)$

gradient ascent on μ_i, σ_i

How many parameters are there? $|\theta| + (|\mu_i| + |\sigma_i|) \times N$

intuition: $q_i(z)$ should approximate $p(z|x_i)$ what if we learn a *network* $q_i(z) = q(z|x_i) \approx p(z|x_i)$?

z  $p_\theta(x|z)$

x  $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$

Amortized Variational Inference

What's the problem?

for each x_i (or mini-batch):

- calculate $\nabla_\theta \mathcal{L}_i(p, q_i)$:
  - sample $z \sim q_i(z)$
  - $\nabla_\theta \mathcal{L}_i(p, q_i) \approx \nabla_\theta \log p_\theta(x_i|z)$
- $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}_i(p, q_i)$
- update q_i to maximize $\mathcal{L}_i(p, q_i)$

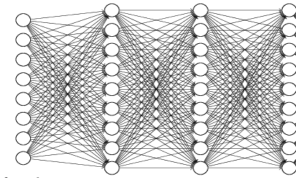
let's say $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$

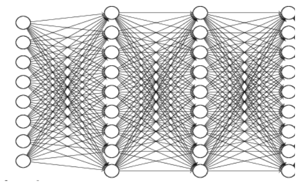
use gradient $\nabla_{\mu_i} \mathcal{L}_i(p, q_i)$ and $\nabla_{\sigma_i} \mathcal{L}_i(p, q_i)$

gradient ascent on μ_i, σ_i

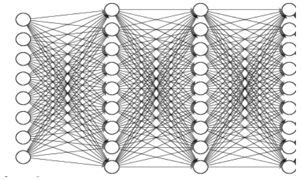
How many parameters are there? $|\theta| + (|\mu_i| + |\sigma_i|) \times N$

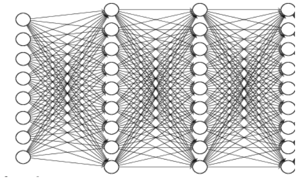
intuition: $q_i(z)$ should approximate $p(z|x_i)$ what if we learn a *network* $q_i(z) = q(z|x_i) \approx p(z|x_i)$?

z  $p_\theta(x|z)$

x  $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$

Amortized variational inference

z  $p_\theta(x|z)$

x  $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$

for each x_i (or mini-batch):

calculate $\nabla_\theta \mathcal{L}(p_\theta(x_i|z), q_\phi(z|x_i))$:

sample $z \sim q_\phi(z|x_i)$

$\nabla_\theta \mathcal{L} \approx \nabla_\theta \log p_\theta(x_i|z)$

$\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}$

$\phi \leftarrow \phi + \alpha \nabla_\phi \mathcal{L}$ ← how do we calculate this?

$$\log p(x_i) \geq \underbrace{E_{z \sim q_\phi(z|x_i)}[\log p_\theta(x_i|z) + \log p(z)] + \mathcal{H}(q_\phi(z|x_i))}_{\mathcal{L}(p_\theta(x_i|z), q_\phi(z|x_i))}$$

Amortized variational inference

for each x_i (or mini-batch):

calculate $\nabla_\theta \mathcal{L}(p_\theta(x_i|z), q_\phi(z|x_i))$:

sample $z \sim q_\phi(z|x_i)$

$\nabla_\theta \mathcal{L} \approx \nabla_\theta \log p_\theta(x_i|z)$

$\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}$

$\phi \leftarrow \phi + \alpha \nabla_\phi \mathcal{L}$

$$q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$$

look up formula for entropy of a Gaussian

$$\mathcal{L}_i = \underbrace{E_{z \sim q_\phi(z|x_i)}[\log p_\theta(x_i|z) + \log p(z)]}_{J(\phi) = E_{z \sim q_\phi(z|x_i)}[r(x_i, z)]} + \mathcal{H}(q_\phi(z|x_i))$$

can just use policy gradient!

What's wrong with this gradient?

$$\nabla J(\phi) \approx \frac{1}{M} \sum_j \nabla_\phi \log q_\phi(z_j|x_i) r(x_i, z_j)$$

The reparameterization trick

Is there a better way?

$$J(\phi) = E_{z \sim q_\phi(z|x_i)}[r(x_i, z)]$$
$$= E_{\epsilon \sim \mathcal{N}(0,1)}[r(x_i, \mu_\phi(x_i) + \epsilon \sigma_\phi(x_i))]$$

$$q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$$

$$z = \mu_\phi(x) + \epsilon \sigma_\phi(x)$$

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

independent of ϕ!

estimating $\nabla_\phi J(\phi)$:

sample $\epsilon_1, \dots, \epsilon_M$ from $\mathcal{N}(0,1)$ (a single sample works well!)

$$\nabla_\phi J(\phi) \approx \frac{1}{M} \sum_j \nabla_\phi r(x_i, \mu_\phi(x_i) + \epsilon_j \sigma_\phi(x_i))$$

most autodiff software (e.g., TensorFlow) will compute this for you!

Another way to look at it...

$$\mathcal{L}_i = E_{z \sim q_\phi(z|x_i)}[\log p_\theta(x_i|z) + \log p(z)] + \mathcal{H}(q_\phi(z|x_i))$$

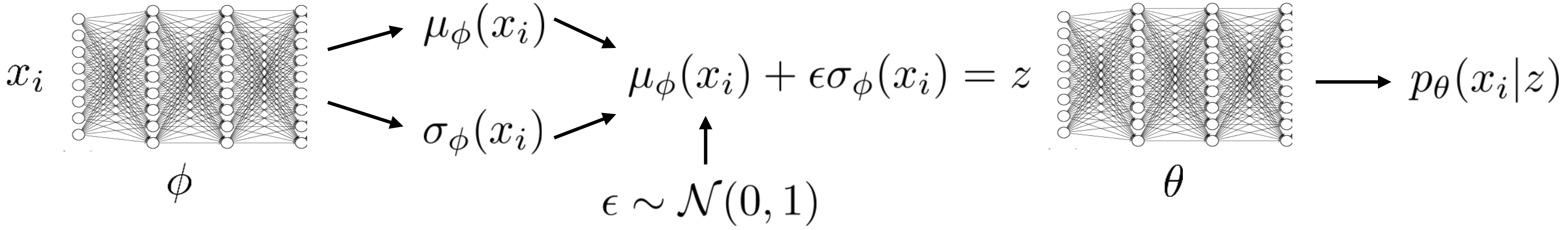
$$= E_{z \sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] + \underbrace{E_{z \sim q_\phi(z|x_i)}[\log p(z)] + \mathcal{H}(q_\phi(z|x_i))}_{-D_{\mathrm{KL}}(q_\phi(z|x_i) \| p(z))}$$

← this often has a convenient analytical form (e.g., KL-divergence for Gaussians)

$$= E_{z \sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] - D_{\mathrm{KL}}(q_\phi(z|x_i) \| p(z))$$

$$= E_{\epsilon \sim \mathcal{N}(0,1)}[\log p_\theta(x_i|\mu_\phi(x_i) + \epsilon\sigma_\phi(x_i))] - D_{\mathrm{KL}}(q_\phi(z|x_i) \| p(z))$$

$$\approx \log p_\theta(x_i|\mu_\phi(x_i) + \epsilon\sigma_\phi(x_i)) - D_{\mathrm{KL}}(q_\phi(z|x_i) \| p(z))$$



Reparameterization trick vs. policy gradient

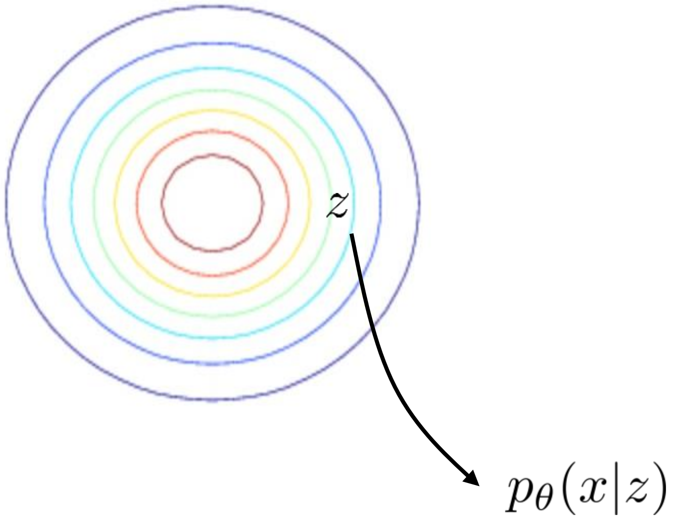
- Policy gradient
  - Can handle both discrete and continuous latent variables
  - High variance, requires multiple samples & small learning rates
- Reparameterization trick
  - Only continuous latent variables
  - Very simple to implement
  - Low variance

$$J(\phi) \approx \frac{1}{M} \sum_j \nabla_\phi \log q_\phi(z_j|x_i) r(x_i, z_j)$$

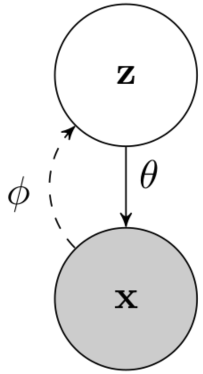
$$\nabla_\phi J(\phi) \approx \frac{1}{M} \sum_j \nabla_\phi r(x_i, \mu_\phi(x_i) + \epsilon_j \sigma_\phi(x_i))$$

Variational Autoencoders

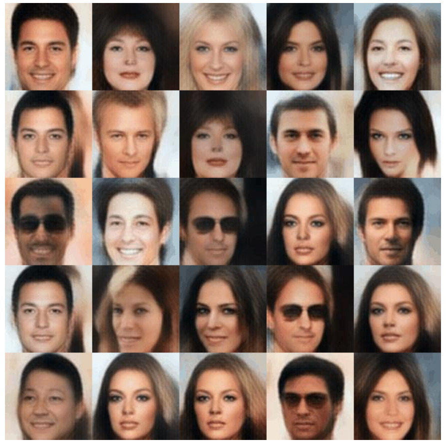
The *variational* autoencoder



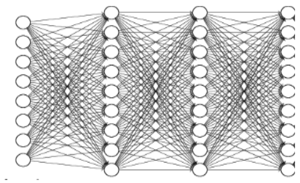
$q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$



$p_\theta(x|z) = \mathcal{N}(\mu_\theta(z), \sigma_\theta(z))$



x_i



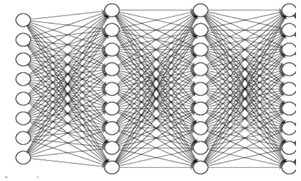
ϕ

$\mu_\phi(x_i)$

$\sigma_\phi(x_i)$

$\mu_\phi(x_i) + \epsilon\sigma_\phi(x_i) = z$

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

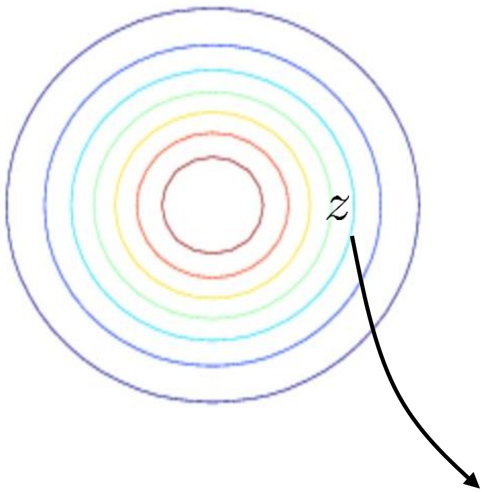


θ

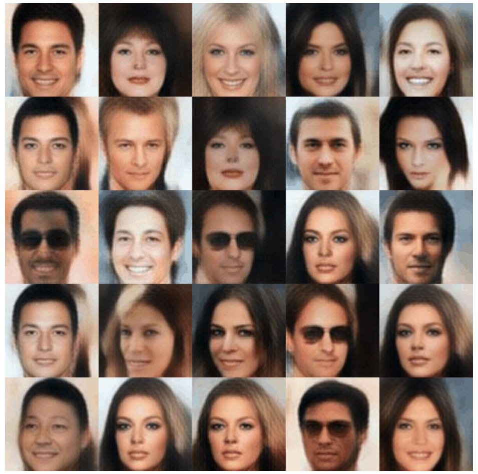
$p_\theta(x|z)$

$$\max_{\theta,\phi} \frac{1}{N} \sum_i \log p_\theta(x_i|\mu_\phi(x_i) + \epsilon\sigma_\phi(x_i)) - D_{\mathrm{KL}}(q_\phi(z|x_i) \| p(z))$$

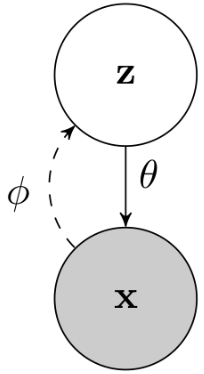
Using the variational autoencoder



$p_\theta(x|z)$



$q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$



$p_\theta(x|z) = \mathcal{N}(\mu_\theta(z), \sigma_\theta(z))$

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

why does this work?

sampling:

$z \sim p(z)$

$x \sim p(x|z)$

$$\mathcal{L}_i = E_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] - D_{\mathrm{KL}}(q_\phi(z|x_i)\|p(z))$$

Conditional models

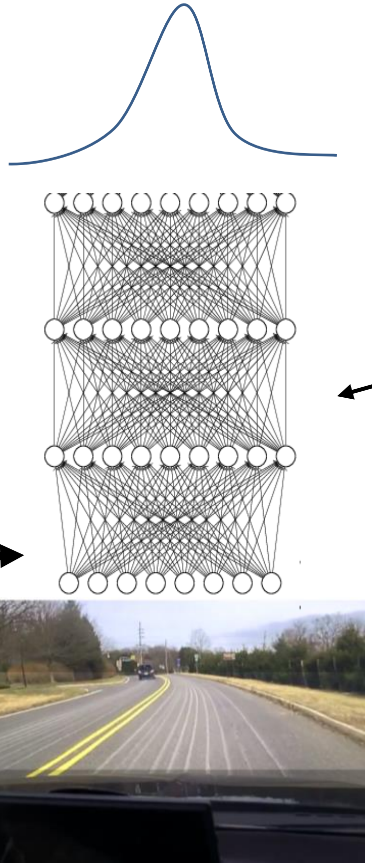
$$\mathcal{L}_i = E_{z \sim q_\phi(z|x_i, y_i)}[\log p_\theta(y_i|x_i, z) + \log p(z|x_i)] + \mathcal{H}(q_\phi(z|x_i, y_i))$$

just like before, only now generating y_i
and *everything* is conditioned on x_i

at test time:

$z \sim p(z|x_i)$

$y \sim p(y|x_i, z)$



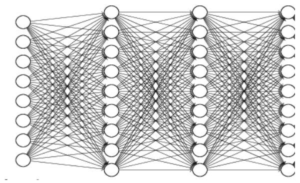
$p(y|x, z)$

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

$p(z)$

can *optionally* depend on x

x_i
y_i



ϕ

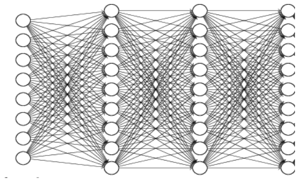
$\mu_\phi(x_i, y_i)$

$\sigma_\phi(x_i, y_i)$

$\mu_\phi + \epsilon\sigma_\phi = z$

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

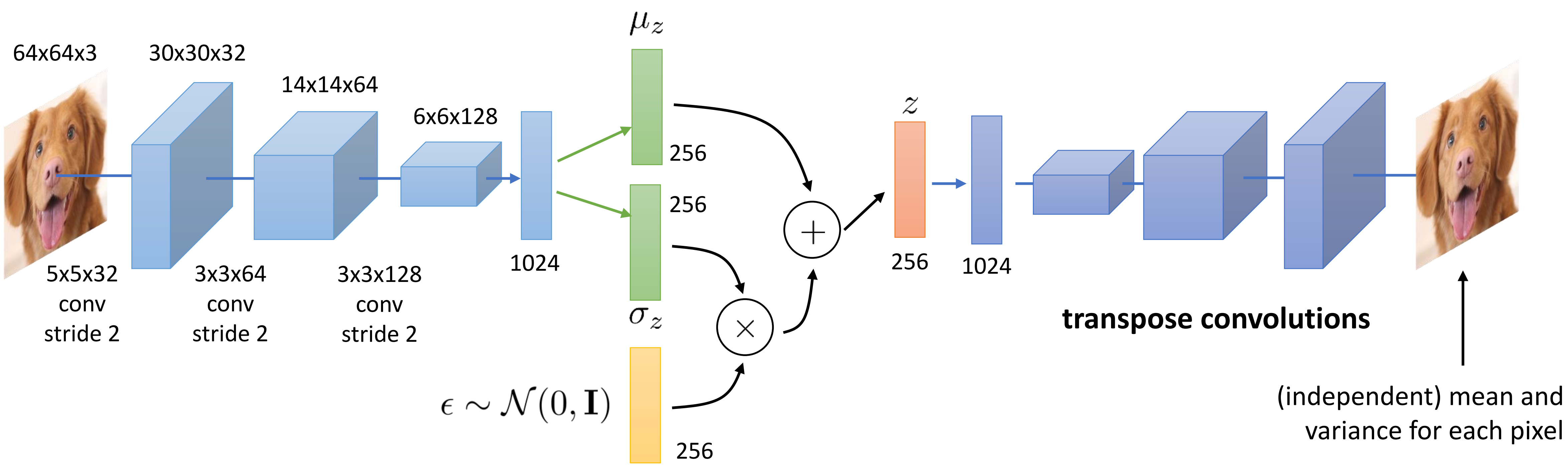
x_i



θ

$p_\theta(y_i|x_i, z)$

VAEs with convolutions



Question: can we design a **fully convolutional** VAE?

Yes, but be careful with the latent codes!

$p(z) = \mathcal{N}(0, \mathbf{I})$ ⟵ implies all z dimensions are independent

VAEs in practice

Common issue: very tempting for VAEs (especially **conditional** VAEs) to ignore the latent codes, or generate poor samples

↑

why?

Problem 1: latent code is ignored

$$p_\theta(x|z) \rightarrow p(x)$$

what does this look like? blurry "average" image when *reconstructing*

$$z \sim q_\phi(z|x) \quad x \sim p_\theta(x|z)$$

Problem 2: latent code is not *compressed*

$$q_\phi(z|x) \text{ very far from } p(z)$$

what does this look like? garbage images when *sampling*

$$z \sim p(z) \quad x \sim p_\theta(x|z)$$

too low $\quad$ no info in z $\qquad D_{\mathrm{KL}}(q_\phi(z|x)\|p(z))$ $\qquad$ **too high** $\quad$ too much info in z

↑

need to control this quantity carefully to get good results!

VAEs in practice

Problem 1: latent code is ignored

too low — no info in z

$$D_{\mathrm{KL}}(q_\phi(z|x)\|p(z))$$

Problem 2: latent code is not *compressed*

too high — too much info in z

↑ need to control this quantity carefully to get good results!

$$\max_{\theta,\phi} \frac{1}{N} \sum_i \log p_\theta(x_i|\mu_\phi(x_i) + \epsilon\sigma_\phi(x_i)) - \beta D_{\mathrm{KL}}(q_\phi(z|x_i)\|p(z))$$

↑ multiplier to adjust regularizer strength

adjust β manually to get good reconstructions **and** good samples

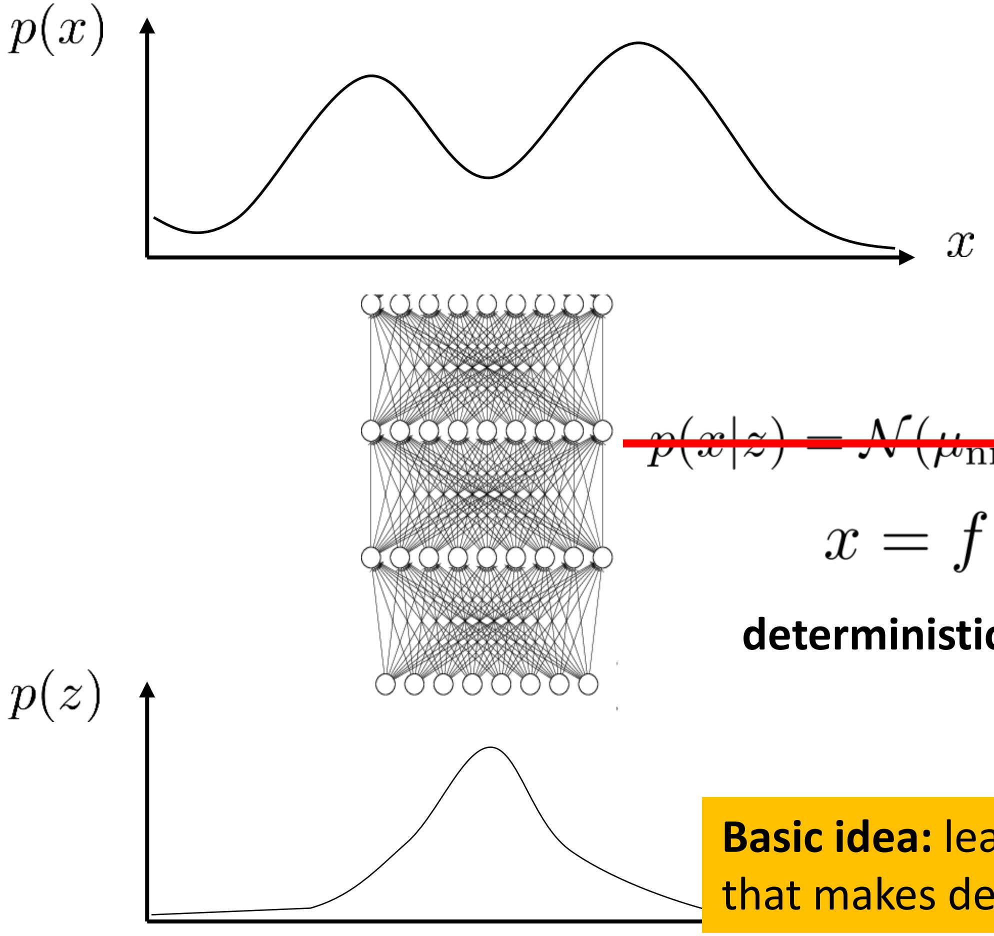
could **schedule** β

start low (to get VAE to use z to reconstruct)

end high (to get samples to be good)

Invertible Models and Normalizing Flows

A simpler kind of model



~~$p(x|z) = \mathcal{N}(\mu_{\text{nn}}(z), \sigma_{\text{nn}}(z))$~~

$$x = f(z)$$

deterministic function

Why is this such a big deal?

change of variables formula:

$$p(x) = p(z) \left| \det \left(\frac{df(z)}{dz} \right) \right|^{-1}$$

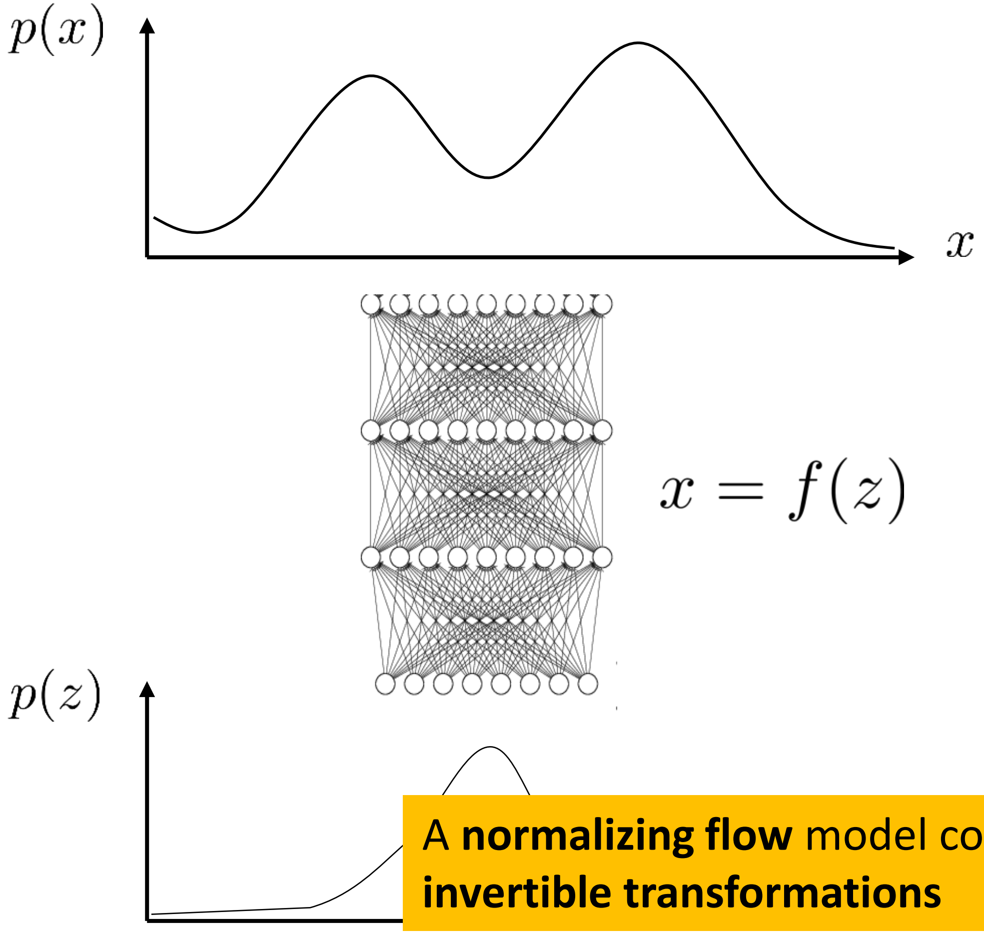
where $z = f^{-1}(x)$

correction for change in local density due to f

Basic idea: learn *invertible* mapping from **z** to **x** that makes determinant easy to compute

No more need for lower bounds! Can get exact probabilities/likelihoods!

Normalizing flow models



Training objective:

$$\max_{\theta} \frac{1}{N} \sum_{i=1}^{N} \log p(x_i)$$

$$p(x) = p(z) \left| \det \left(\frac{df(z)}{dz} \right) \right|^{-1}$$

where $z = f^{-1}(x)$

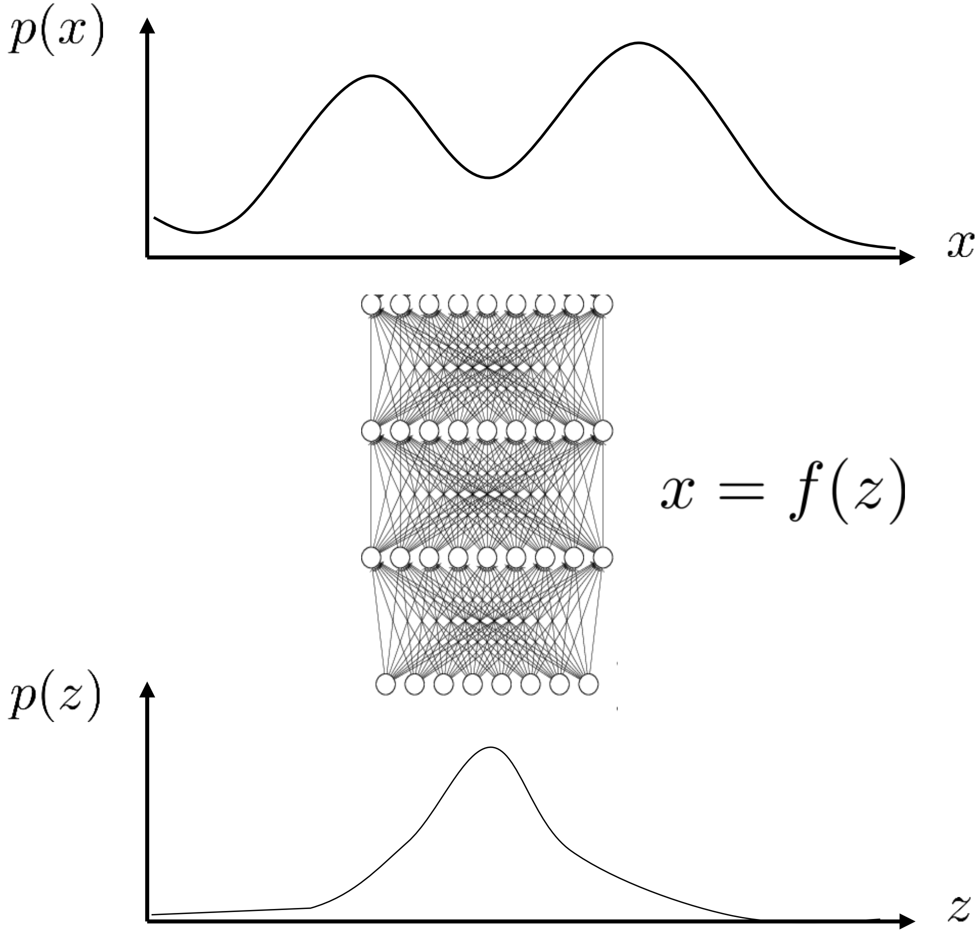
$$\max_{\theta} \frac{1}{N} \sum_{i=1}^{N} \log p(f^{-1}(x_i)) - \log \left| \det \left(\frac{df(z)}{dz} \right) \right|$$

choose a **special** architecture that makes these easy to compute

A **normalizing flow** model consists of multiple layers of **invertible transformations**

We need to figure out how to make an invertible layer, and then compose many of them to make a deep network

Normalizing flow models



$$\max_\theta \frac{1}{N} \sum_{i=1}^{N} \log p(f^{-1}(x_i)) - \log \left| \det \left(\frac{df(z)}{dz} \right) \right|$$

$$f(z) = f_4(f_3(f_2(f_1(z))))$$

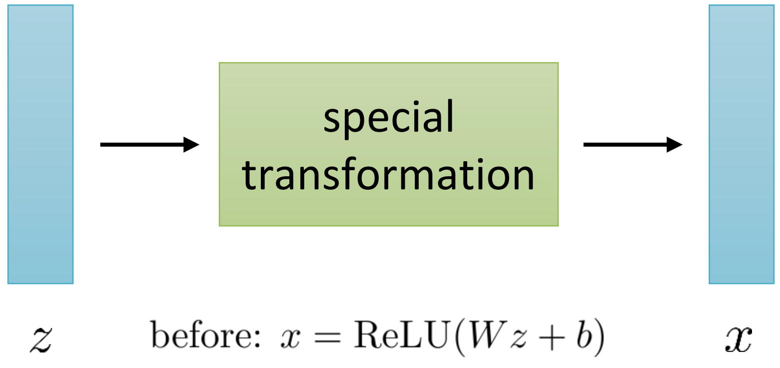
If each **layer** is invertible, the whole thing is invertible

Oftentimes, invertible layers also have very convenient determinants

Log-determinant of whole model is just the sum of log-determinants of the layers

Goal: design an invertible layer, and then compose many of them to create a fully invertible neural net

NICE: Nonlinear Independent Components Estimation

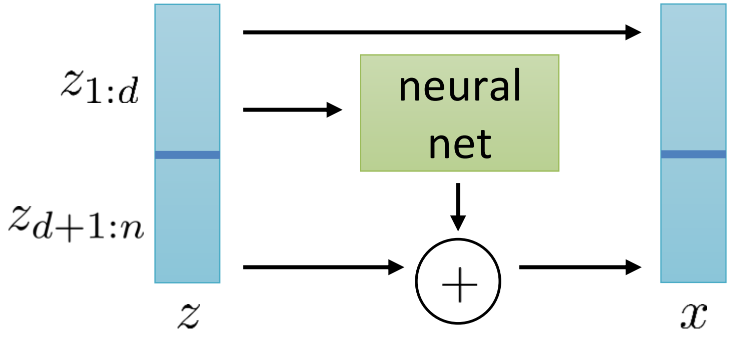


before: $x = \mathrm{ReLU}(Wz + b)$

but this is **not** invertible

Idea: what if we force **part** of the layer to keep all the information so that we can then recover anything that was changed by the nonlinear transformation?

Important: here I describe the case for **one** layer, but in reality we'll have many layers!



$$x_{1:d} = z_{1:d}$$

$$x_{d+1:n} = z_{d+1:n} + g_\theta(z_{1:d})$$

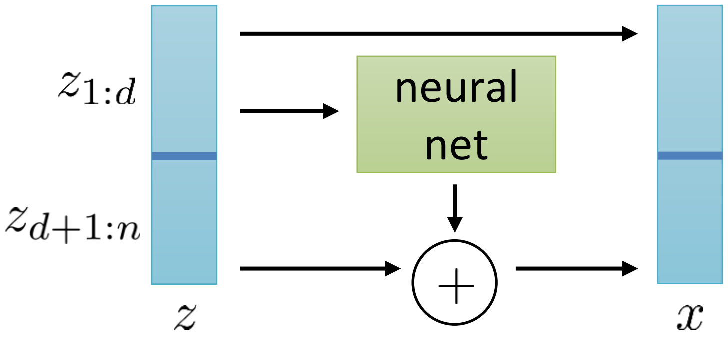
e.g., $g_\theta(a) = \mathrm{ReLU}(Wa + b)$

Question: if we have x, can we recover z?

if so, then this layer is **invertible**

Dinh et al. **NICE: Non-linear Independent Components Estimation**. 2014.

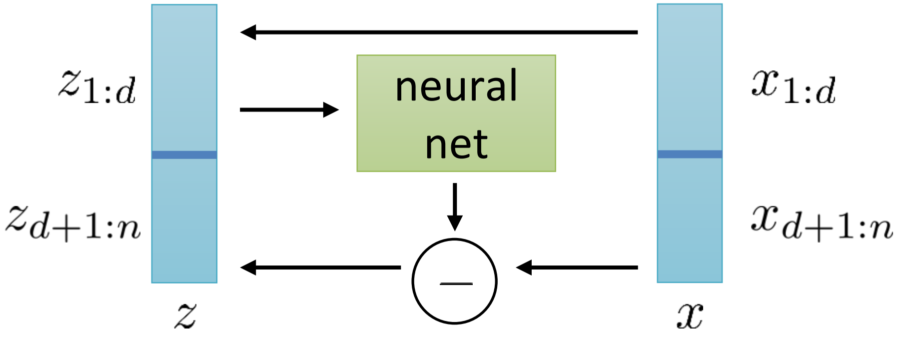
NICE: Nonlinear Independent Components Estimation



$$x_{1:d} = z_{1:d}$$

$$x_{d+1:n} = z_{d+1:n} + g_\theta(z_{1:d})$$

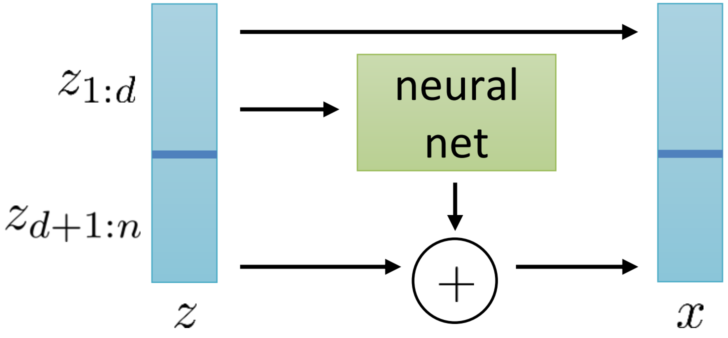
Question: if we have x, can we recover z?
if so, then this layer is **invertible**



1. Recover $z_{1:d} = x_{1:d}$
2. Recover $g_\theta(z_{1:d})$
3. Recover $z_{d+1:n} = x_{d+1:n} - g_\theta(z_{1:d})$

Dinh et al. **NICE: Non-linear Independent Components Estimation**. 2014.

What about the Jacobian?



$$x_{1:d} = z_{1:d}$$

$$x_{d+1:n} = z_{d+1:n} + g_\theta(z_{1:d})$$

$$\left|\det\left(\frac{df(z)}{dz}\right)\right| = 1$$

This is very simple and convenient

But it's representationally a bit limiting

$$\frac{df(z)}{dz} = \begin{bmatrix} \frac{dx_{1:d}}{dz_{1:d}} & \frac{dx_{1:d}}{dz_{d+1:n}} \\ \frac{dx_{d+1:n}}{dz_{1:d}} & \frac{dx_{d+1:n}}{dz_{d+1:n}} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ \frac{dg_\theta}{dz_{1:d}} & \mathbf{I} \end{bmatrix}$$

$\mathbf{I}$ → $\frac{dx_{1:d}}{dz_{1:d}}$; 0 → $\frac{dx_{1:d}}{dz_{d+1:n}}$; $\frac{dg_\theta}{dz_{1:d}}$ → $\frac{dx_{d+1:n}}{dz_{1:d}}$; $\mathbf{I}$ → $\frac{dx_{d+1:n}}{dz_{d+1:n}}$

Dinh et al. **NICE: Non-linear Independent Components Estimation**. 2014.

NICE: Nonlinear Independent Components Estimation

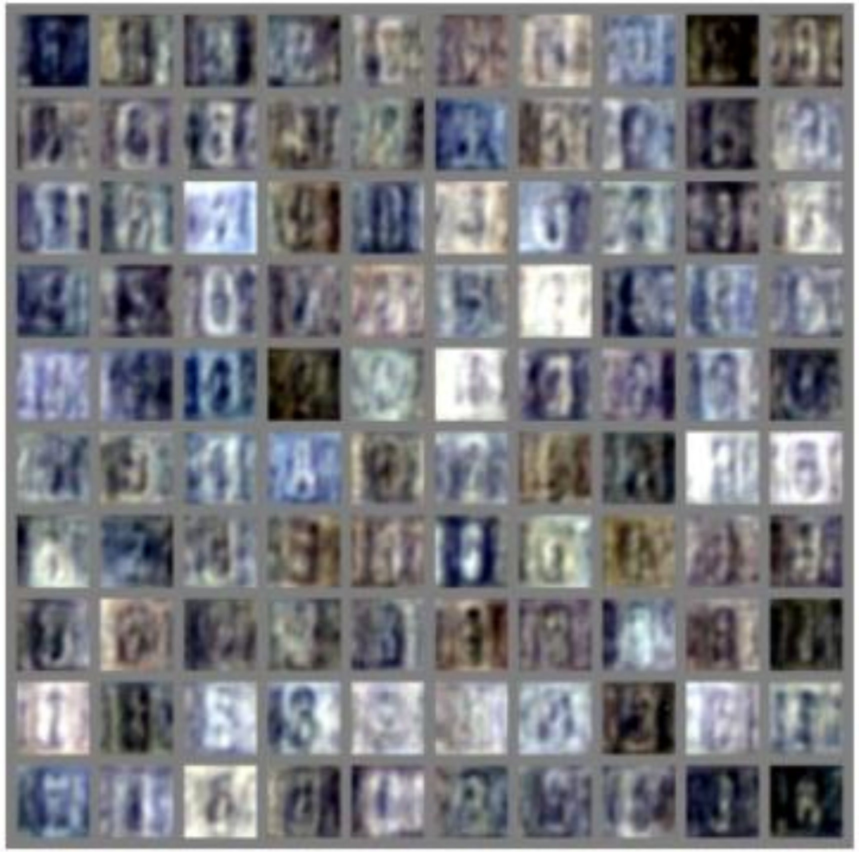

(a) Model trained on MNIST


(b) Model trained on TFD

Material based on Grover & Ermon CS236

Dinh et al. **NICE: Non-linear Independent Components Estimation**. 2014.

NICE: Nonlinear Independent Components Estimation



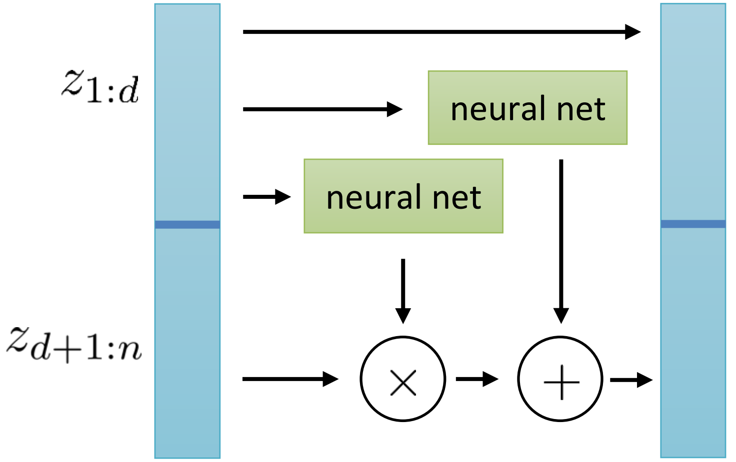
(c) Model trained on SVHN

(d) Model trained on CIFAR-10

Material based on Grover & Ermon CS236

Dinh et al. **NICE: Non-linear Independent Components Estimation**. 2014.

Real-**NVP**: **N**on-**V**olume **P**reserving Transformation



$$x_{1:d} = z_{1:d}$$

$$x_{d+1:n} = z_{d+1:n} \times \exp(h_\theta(z_{1:d})) + g_\theta(z_{1:d})$$

elementwise product

Inverse can be derived in the same way as before:

1. Recover $z_{1:d} = x_{1:d}$
2. Recover $g_\theta(z_{1:d})$ and $h_\theta(z_{1:d})$
3. Recover $z_{d+1:n} = (x_{d+1:n} - g_\theta(z_{1:d}))/\exp(h_\theta(z_{1:d}))$

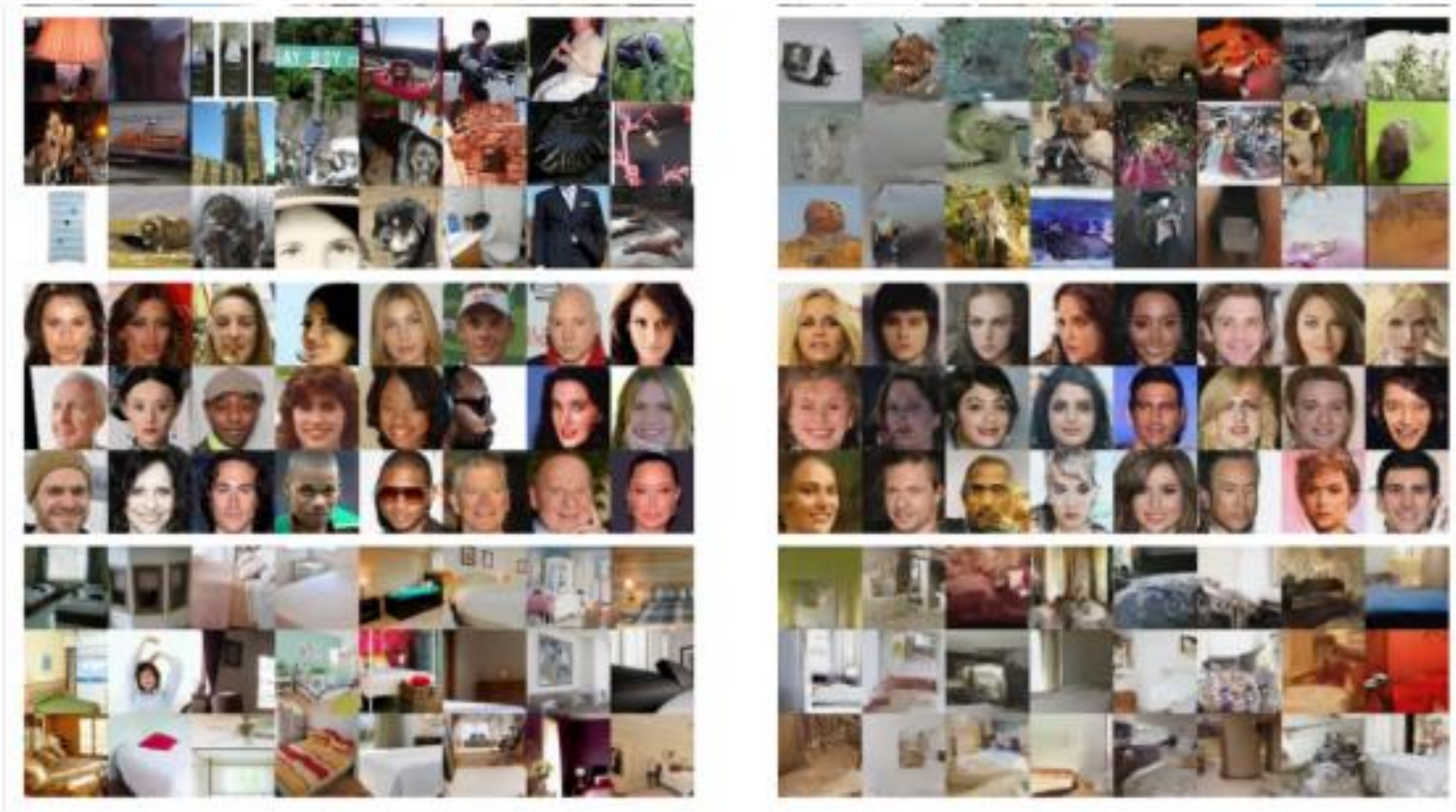
$$\left|\det\left(\frac{df(z)}{dz}\right)\right| = \prod_{i=d+1}^{n} \exp(h_\theta(z_{1:d})_i)$$

$$\frac{df(z)}{dz} = \begin{bmatrix} \mathbf{I} & 0 \\ \frac{dx_{d+1:n}}{dz_{1:d}} & \operatorname{diag}\left(\exp\left(h_\theta(z_{1:d})\right)\right) \end{bmatrix}$$

This is significantly more expressive

Dinh et al. **Density estimation using Real-NVP**. 2016.

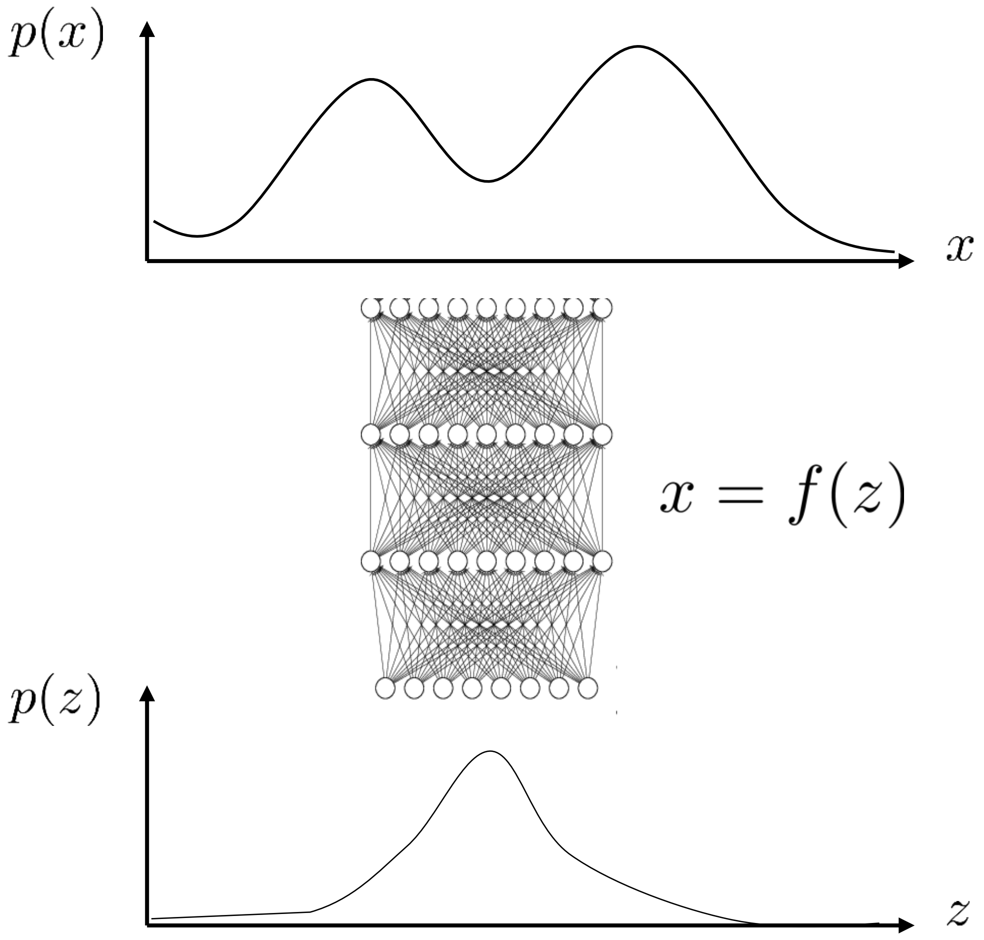
Real-NVP Samples



Material based on Grover & Ermon CS236

Dinh et al. **Density estimation using Real-NVP**. 2016.

Concluding Remarks



+ can get exact probabilities/likelihoods

+ no need for lower bounds

+ conceptually simpler (perhaps)

- requires special architecture

- Z must have same dimensionality as X