CSC 580 Principles of Machine Learning

15 Unsupervised learning with NNs

Chicheng Zhang

Department of Computer Science

THE UNIVERSITY OF ARIZONA

*slides credit: built upon CSC 580 Fall 2021 lecture slides by Kwang-Sung Jun

This lecture

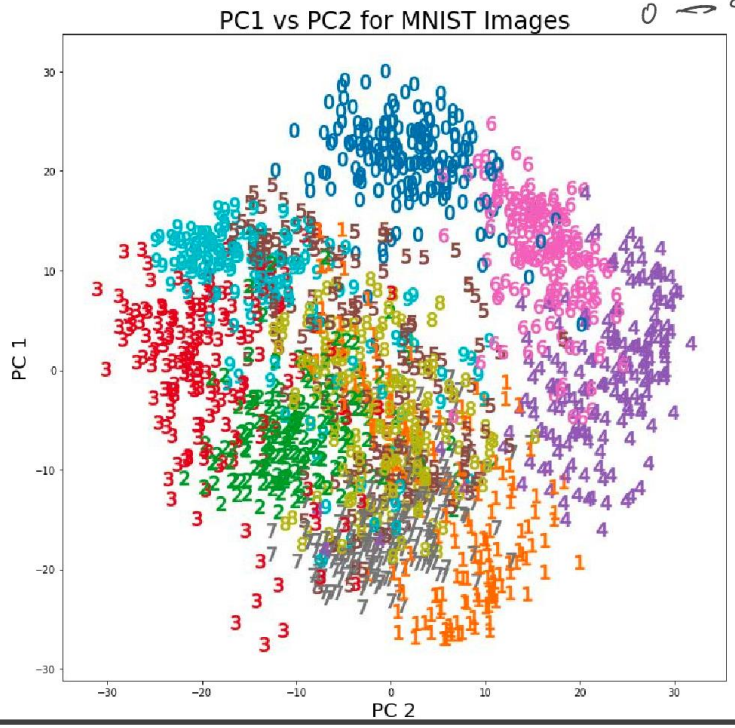
- Autoencoder
- Variational auto-encoder
- Generative Adversarial Networks (GANs)

Unsupervised learning review

- Recall: unlabeled data.
- Q: what is the main goal of unsupervised learning?
  - Represent data using a low degree of freedom
- Examples: clustering, PCA.
- Recall PCA can be used for ‘representation learning’ = learning useful (and compact) features.

  (learned features = projected feature vector)
- NNs can be used to do generalizations of PCA.

Example: MNIST dataset → 16×16 = 256 pixel. 0 → 9



Autoencoder

Introductory example

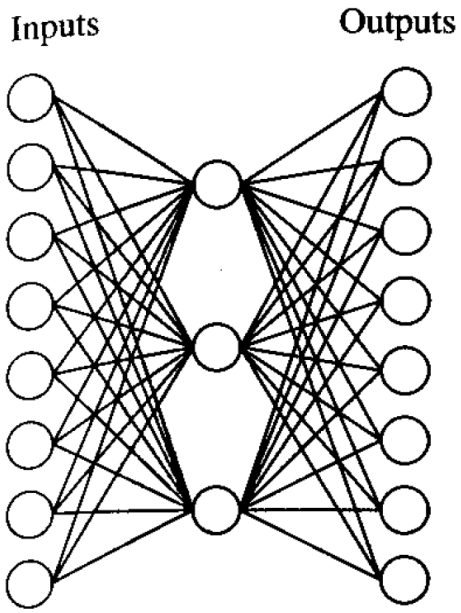
- Suppose you have a number in {0,1,2,3,4,5,6,7}
- What would be a compact representation (say, for computers)?

- Q: how many bits do we need?

Observations from early days

Train a neural net by imposing squared loss on all the output units & backpropping.

Q: What do the hidden values look like?

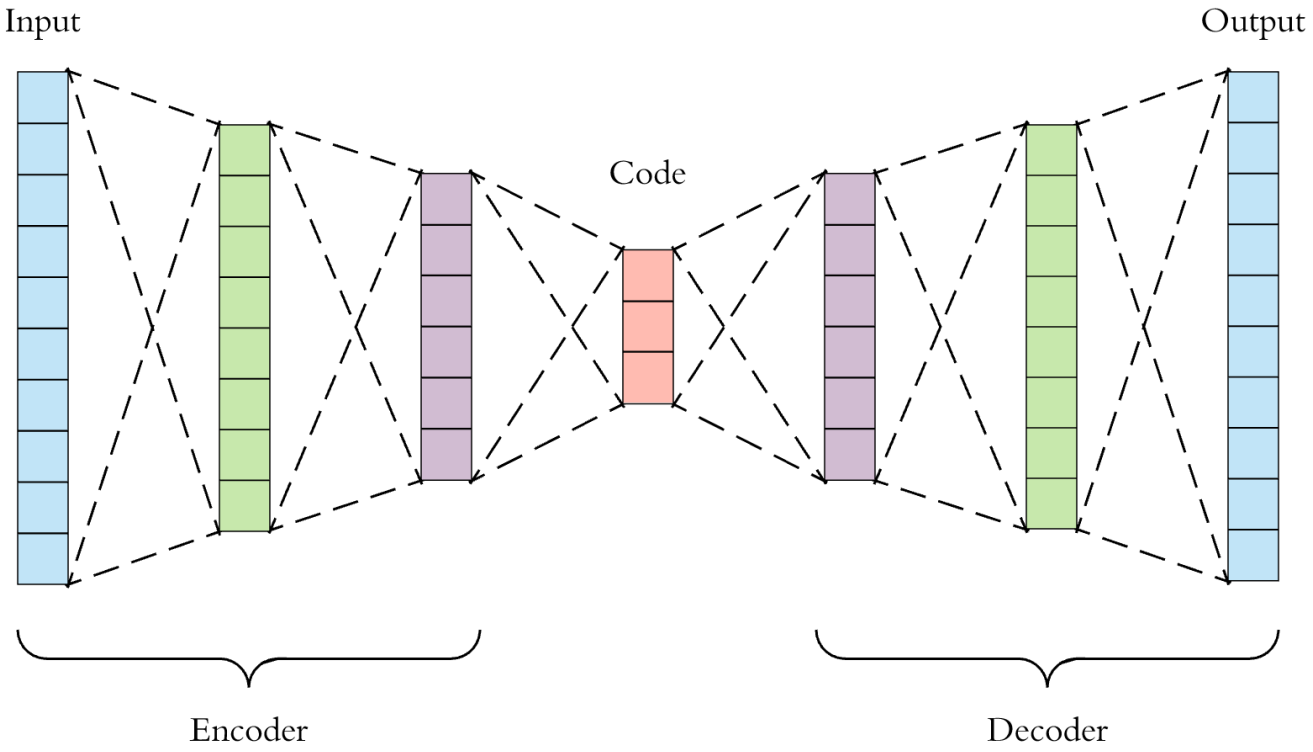


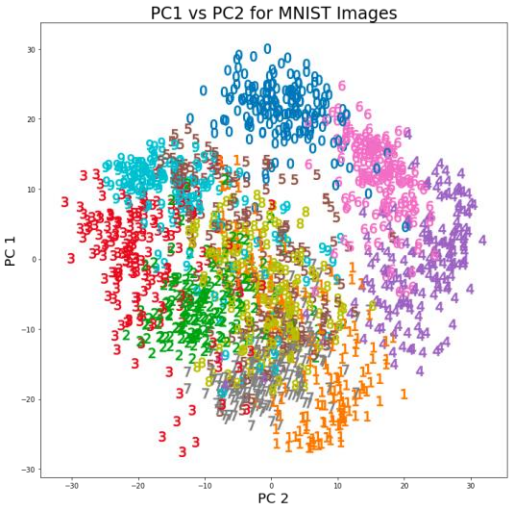
Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

FIGURE 4.7
Learned Hidden Layer Representation. This $8 \times 3 \times 8$ network was trained to learn the identity function, using the eight training examples shown. After 5000 training epochs, the three hidden unit values encode the eight distinct inputs using the encoding shown on the right. Notice if the encoded values are rounded to zero or one, the result is the standard binary encoding for eight distinct values.

p107, Tom Mitchell, “Machine Learning”

Autoencoder using deep networks





We can do this for any data!

How to use it:

- Encoder: for dimensionality reduction
- Decoder: generate new samples from the distribution by varying the input ‘code’

image from https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

PCA as a linear neural network

linear = no activation

- Recall PCA

PCA pseudocode

- Input: data matrix $X \in \mathbb{R}^{n \times d}$
- Preprocess: Let $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$. Compute $x_i' = x_i - \mu, \forall i \in [n]$
- Compute the top k eigenvectors $V = [v_1, \dots, v_k]$ of $\frac{1}{n}\sum_{i=1}^{n} x_i'(x_i')^\top$
- Feature map: $\phi(x) = \left(v_1^\top(x-\mu), \dots, v_k^\top(x-\mu)\right) \in \mathbb{R}^k$ ← coefficient.
- Decorrelating property: ("whitening")
  - $\frac{1}{n}\sum_{i=1}^{n} \phi(x_i) = 0$
  - $\frac{1}{n}\sum_{i=1}^{n} \phi(x_i)\phi(x_i)^\top = Diag(\lambda_1, \dots, \lambda_k)$ → $\begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \ddots \\ & & & \lambda_k \end{bmatrix}$
- Reconstruction (the actual projection): apply $\mu + V\phi(x)$

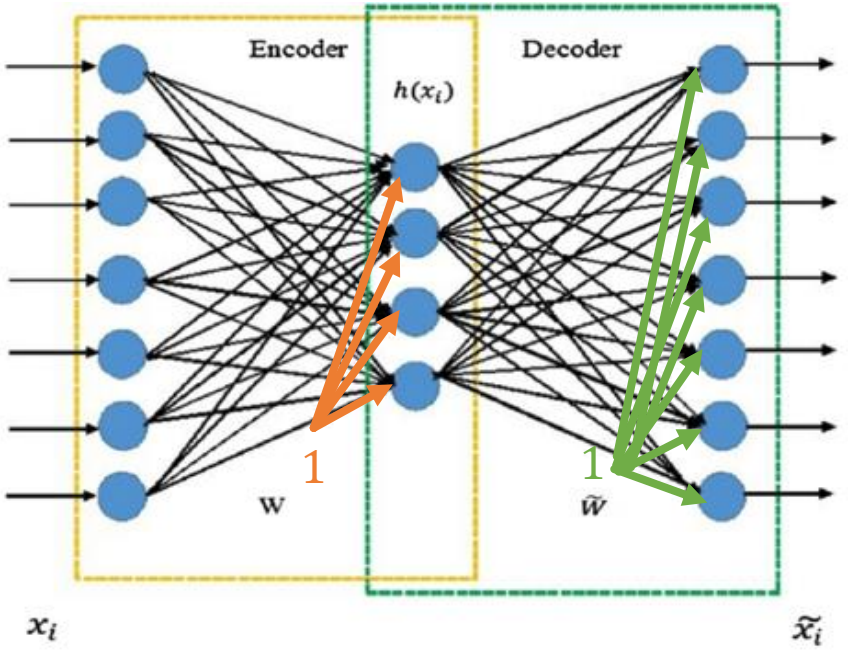
($v_1 \cdots v_k$)

PCA as a linear NN

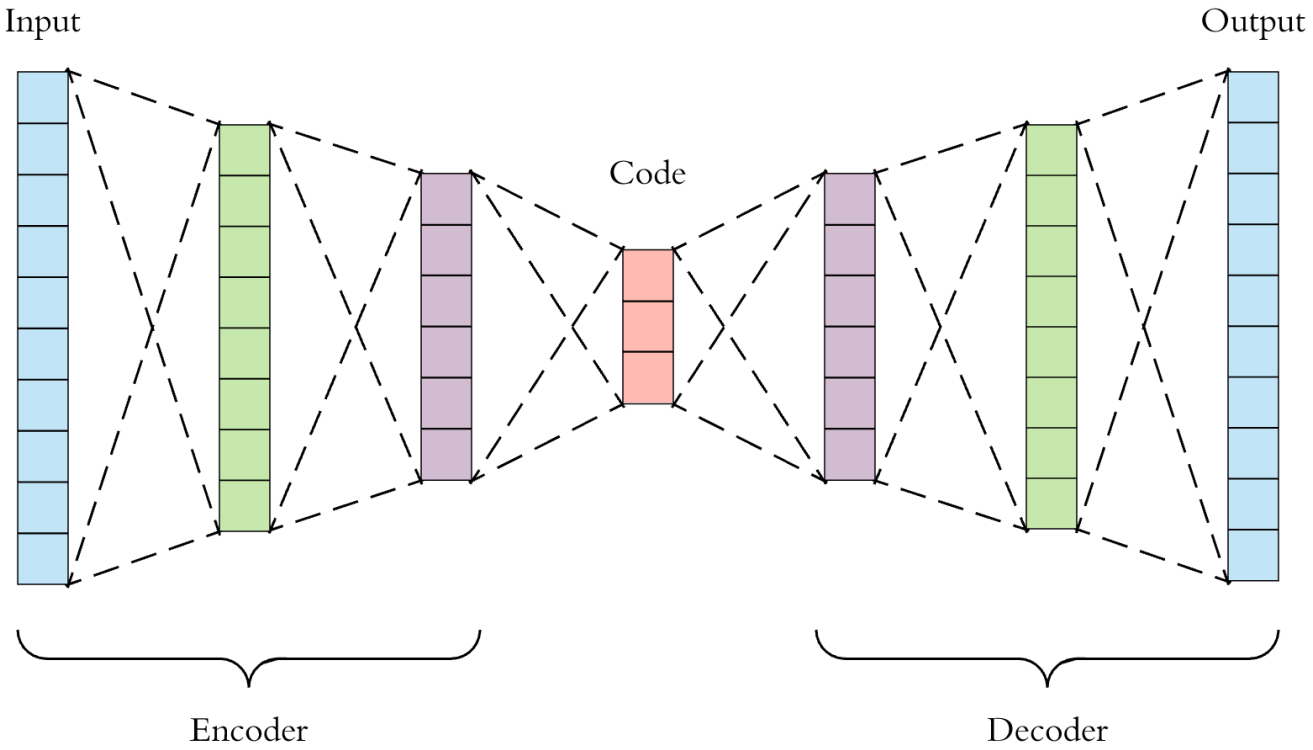
- k units in the hidden layer.
- The PCA can be represented as a neural network (with constant bias added in each layer):

- Encoder: $h = \begin{pmatrix} - & v_1 & - \\ & \dots & \\ - & v_k & - \end{pmatrix} \cdot x + \begin{pmatrix} -v_1^\top \mu \\ \dots \\ -v_k^\top \mu \end{pmatrix}$
- Decoder: $\tilde{x} = \begin{pmatrix} | & & | \\ v_1 & \dots & v_k \\ | & & | \end{pmatrix} \cdot h + \begin{pmatrix} | \\ \mu \\ | \end{pmatrix}$

- Feature map: $\phi(x) = \left(v_1^\top (x-\mu), \dots, v_k^\top (x-\mu)\right) \in \mathbb{R}^k$
- Decorrelating property:
  - $\frac{1}{n}\sum_{i=1}^n \phi(x_i) = 0$
  - $\frac{1}{n}\sum_{i=1}^n \phi(x_i)\phi(x_i)^\top = Diag(\lambda_1, \dots, \lambda_k)$
- Reconstruction (the actual projection): apply $\mu + V\phi(x)$



Autoencoder using deep networks



We can do this for any data!

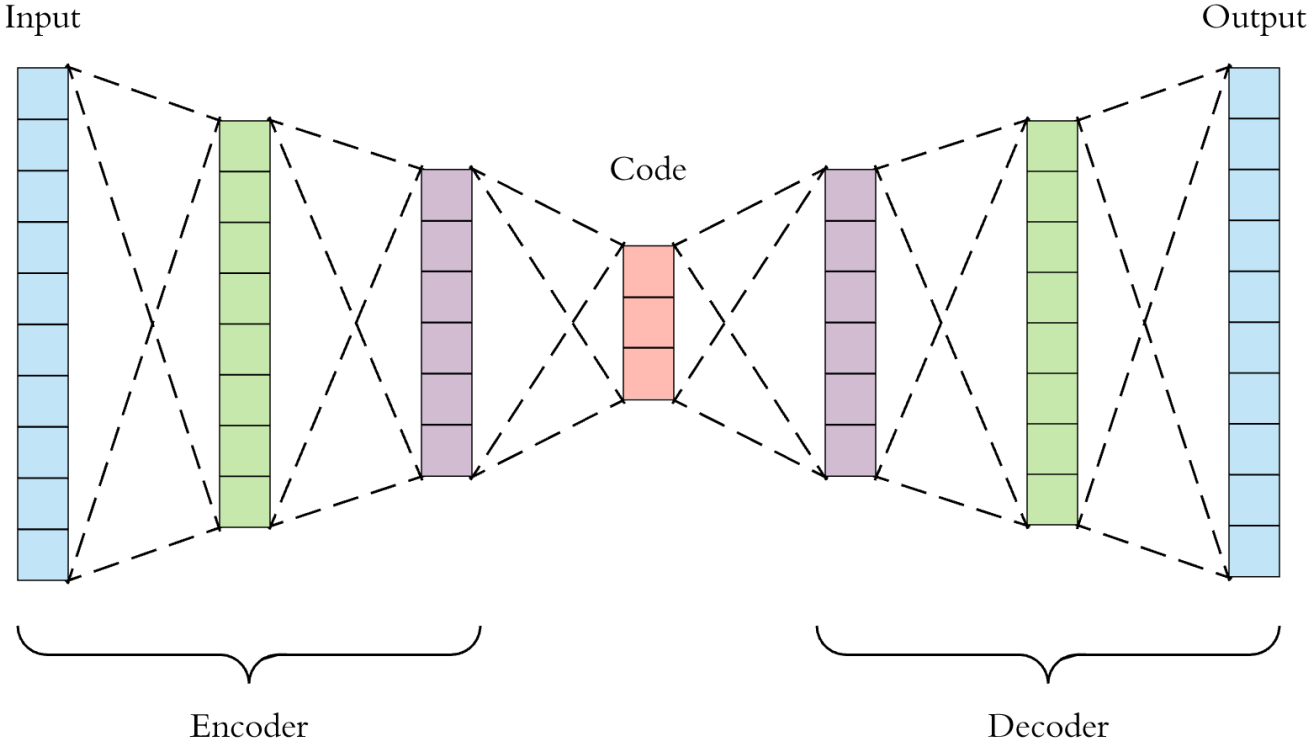
What about images?

image from https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Training autoencoders

- Given:
  - data $x_1, \ldots, x_n \in \mathbb{R}^d$,
  - Embedding dimension k ($k \ll d$)

- Goal: obtain
  - Encoder network $f_\theta: \mathbb{R}^d \to \mathbb{R}^k$
  - Decoder network $g_\phi: \mathbb{R}^k \to \mathbb{R}^d$
  - Such that for every i, $x_i \approx g_\phi(f_\theta(x_i))$

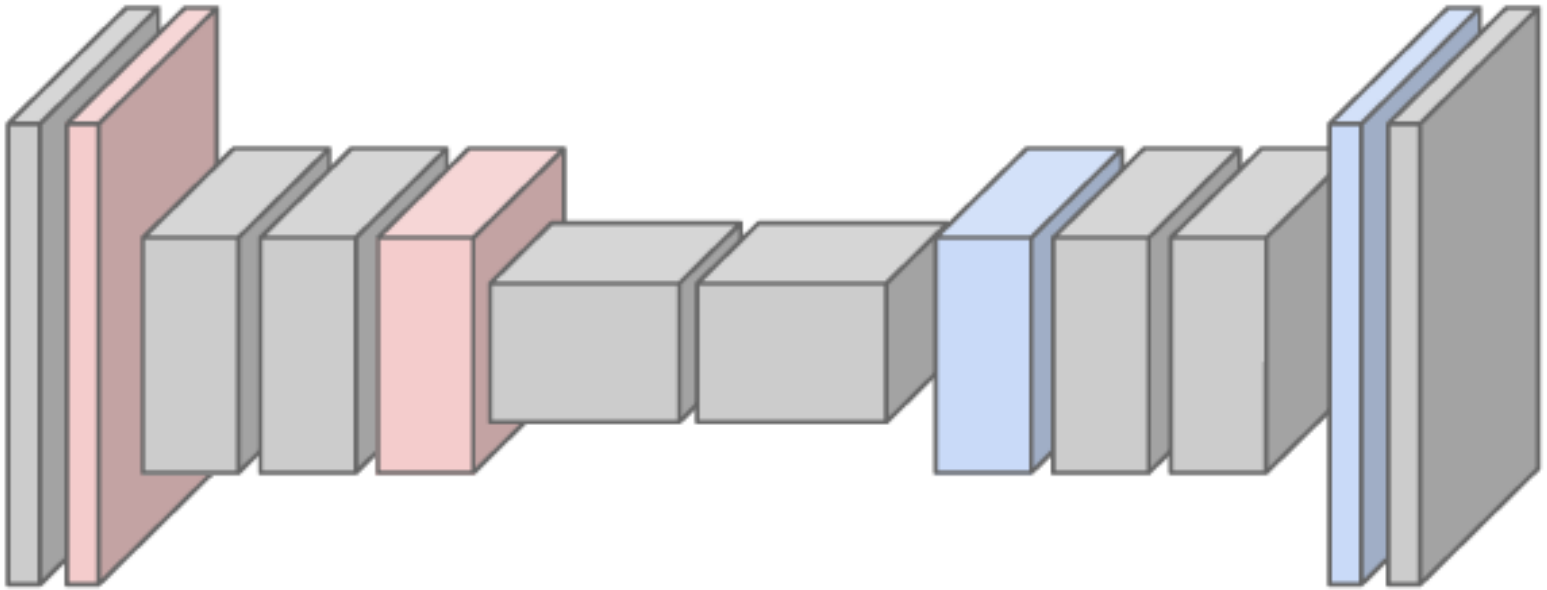


- Most commonly used formulation (can be straightforwardly trained by gradient-based methods):

$$\text{minimize}_{\theta,\phi} \sum_{i=1}^{n} \underbrace{\left\| x_i - g_\phi\left(f_\theta(x_i)\right)\right\|^2}_{\text{Reconstruction error}}$$

Autoencoder for images

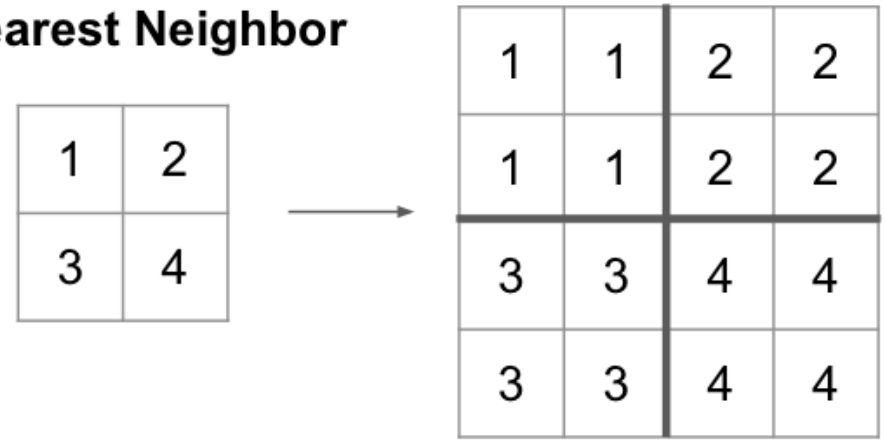
- Encoder: conv-conv-pool-conv-conv-pool-…,
- Decoder: conv-conv-pool-…?? It will reduce the spatial dimension rather than increasing it.
- How to do the opposite of pooling (or conv with stride length >= 2)?



Following slides largely based on Stanford cs231n https://youtu.be/nDPWywWRIRo?t=1109
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

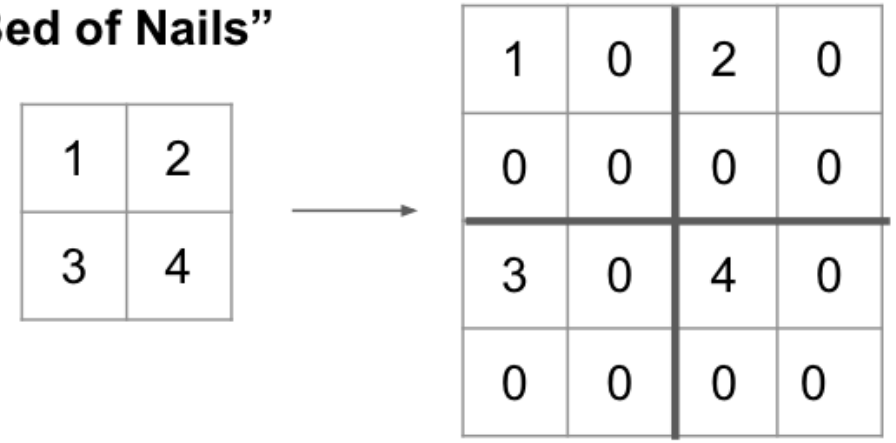
“Un”pooling

Nearest Neighbor



Input: 2 x 2 Output: 4 x 4

“Bed of Nails”



Input: 2 x 2 Output: 4 x 4

(fig. from Stanford cs231n)

Max unpooling

Max Pooling
Remember which element was max!

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

Input: 4 x 4

5	6
7	8

Output: 2 x 2

Rest of the network

Max Unpooling
Use positions from pooling layer

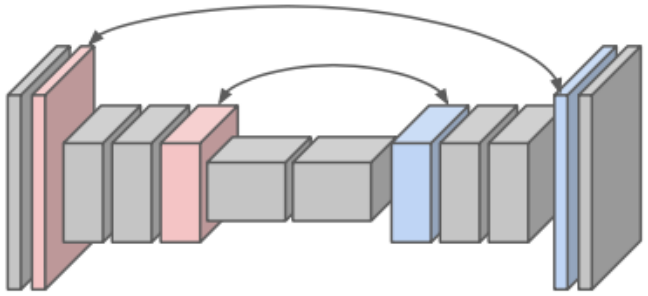
1	2
3	4

Input: 2 x 2

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers

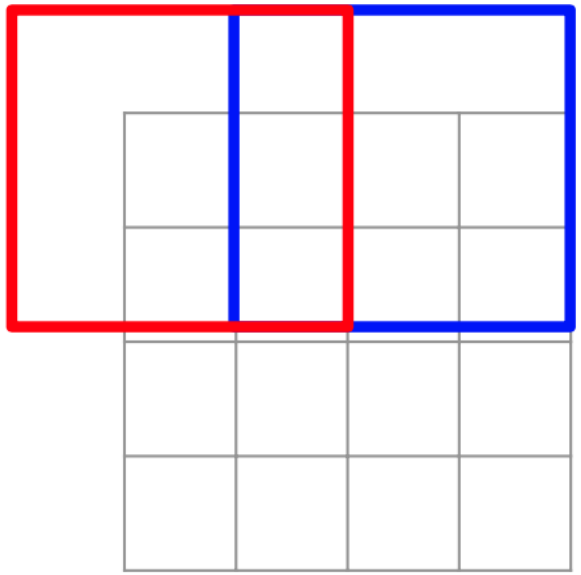


The network must be symmetric!

(fig from Stanford cs231n)

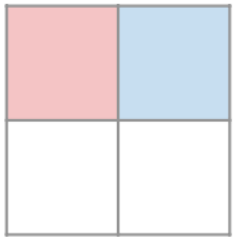
A smarter way (recommended): Transposed convolution

- Other names: upconvolution, fractionally strided convolution, backward strided convolution, deconvolution (don't use this name)
- Recall: 3 x 3 convolution with stride 2 pad 1.



Input: 4 x 4

Dot product between filter and input



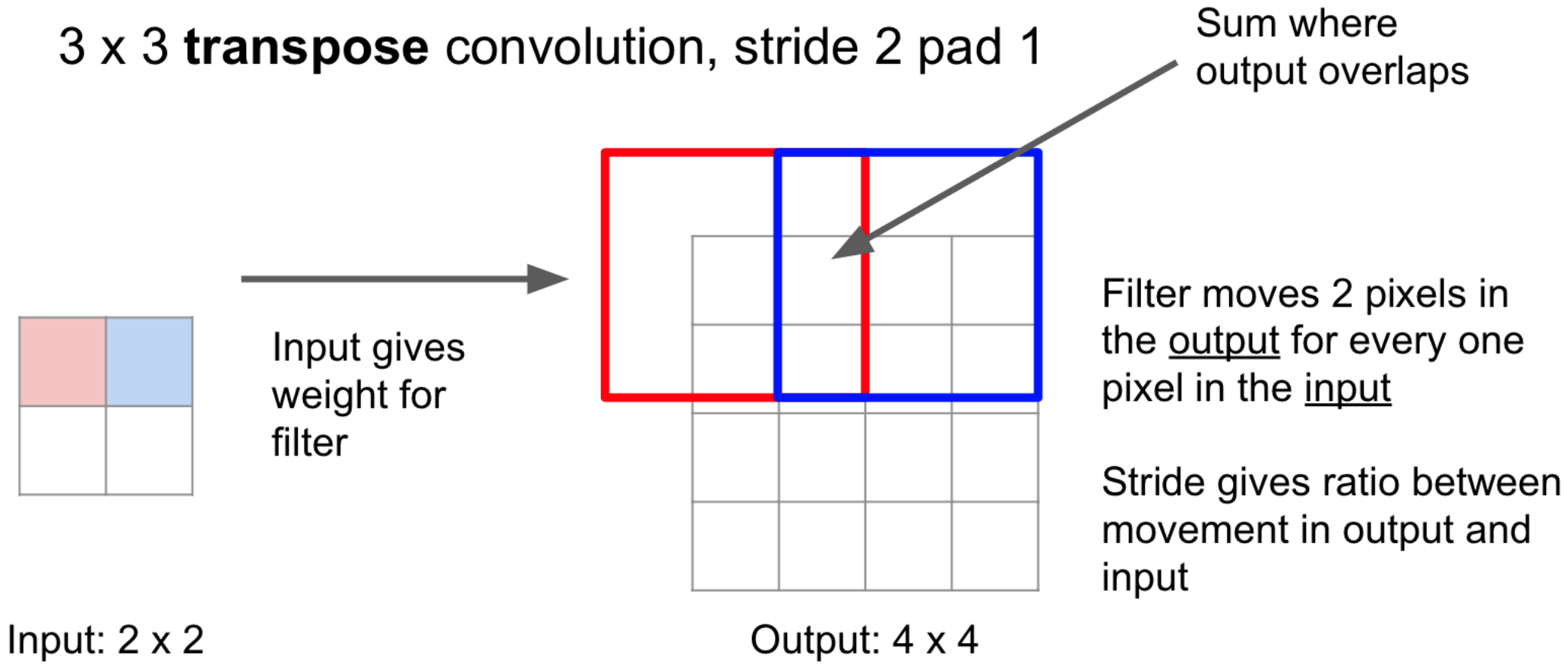
Output: 2 x 2

Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

(fig from Stanford cs231n)

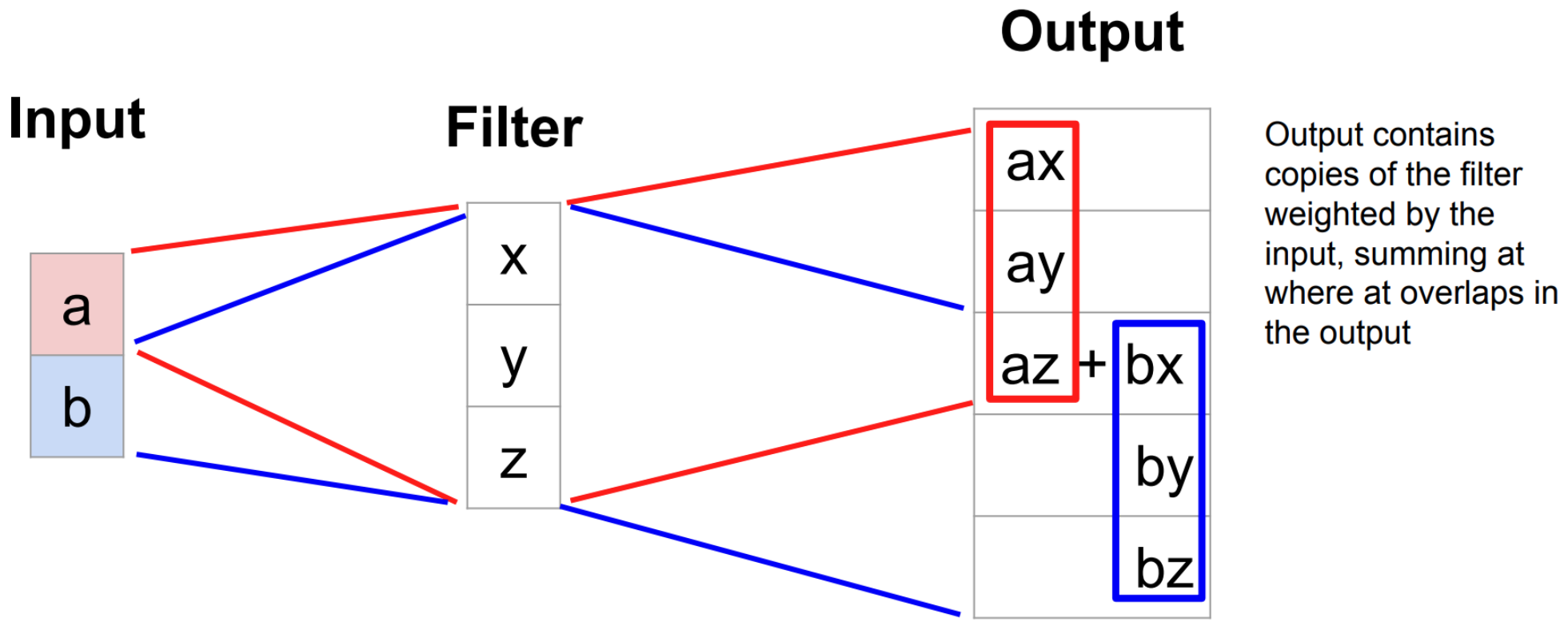
Transposed convolution



Disclaimer: this is not the inverse of convolution!
Rather, it's just a variation of the convolution.

(fig from Stanford cs231n)

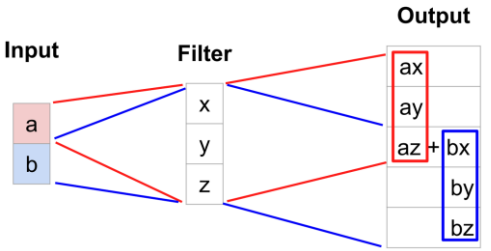
1D transposed convolution



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

(fig from Stanford cs231n)

1D transposed convolution: matrix form



We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

Transposed convolution multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

Example: 1D transposed conv, kernel size=3, stride=2, padding=0

(fig from Stanford cs231n)

Next lecture (11/21)

- Variational autoencoders; Generative Adversarial Networks (GANs)

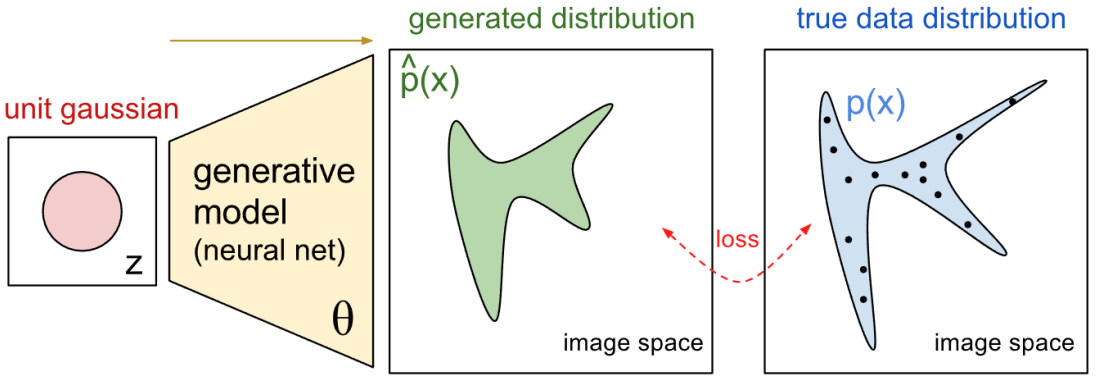
- Assigned reading this week:
  - Stanford CS231n (currently offered by Fei-Fei Li, Jiajun Wu, and Ruohan Gao): notes on Convolutional Networks

Generative networks

- Popular method for learning distributions
- Input: $x_1, \dots, x_n \sim D$ on $\mathbb{R}^d$, some base distribution μ (e.g. uniform, standard Gaussian)
- Goal: learn network $f_\theta: \mathbb{R}^k \to \mathbb{R}^d$, such that $f_\theta \# \mu \approx D$ (usually $k \ll d$)

- $f_\theta \# \mu$: the distribution of $f_\theta(z)$ when $z \sim \mu$

  "Push-forward measure"



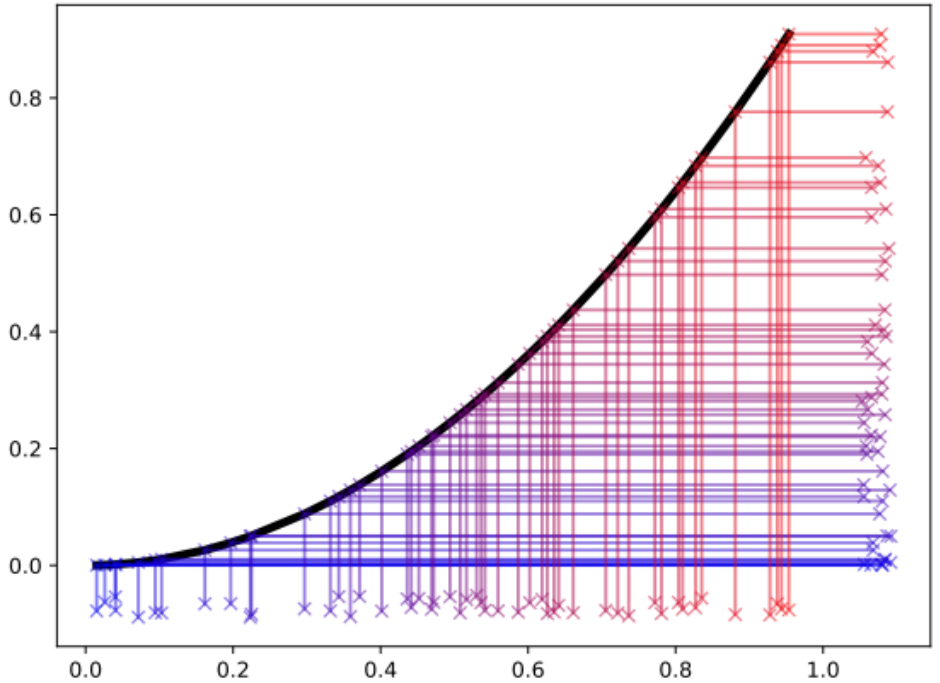
- f_θ gives a low-degree-of-freedom representation of D
- Many applications, e.g. Digital art, Image denoising, Privacy-Preserving Synthetic Data, ..
- We will see two methods to train such networks: VAE and GAN

https://www.cylab.cmu.edu/_files/documents/2020-partners-conference-files/4-wu.pdf
https://openai.com/blog/generative-models/

Generative network: a simple example

- E.g. $d = k = 1$
- $p_D(x) \propto \frac{1}{\sqrt{x}}; \mu = \text{uniform}([0,1])$

=> set $f_\theta(z) = z^2$ ensures that $f_\theta \# \mu = D$



https://mjt.cs.illinois.edu/ml/slides-gn-1.pdf

Variational autoencoder

Variational autoencoder (VAE)

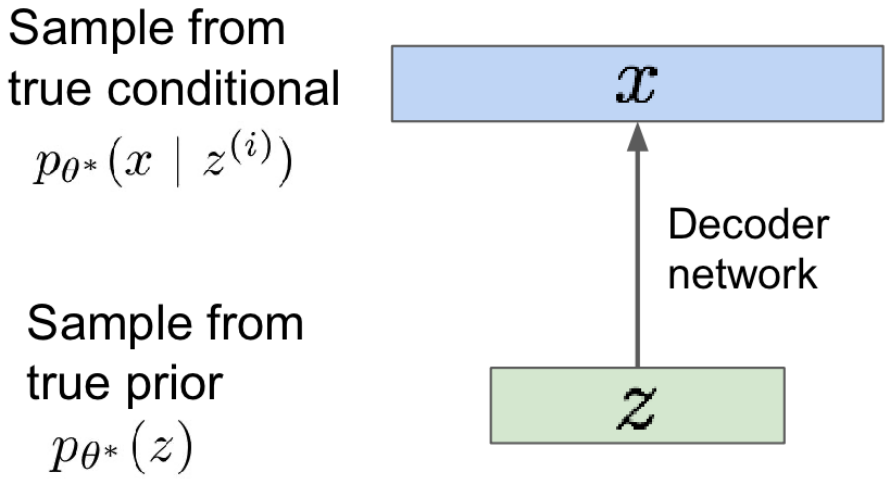
- Generative model counterpart of autoencoder. Also, a good example of using deep learning for probabilistic models.

- Given: $p(z)$ (e.g. standard Gaussian), data $x_1, \ldots, x_n \sim D$
- Goal: build a network θ representing $p(x|z)$, so that

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

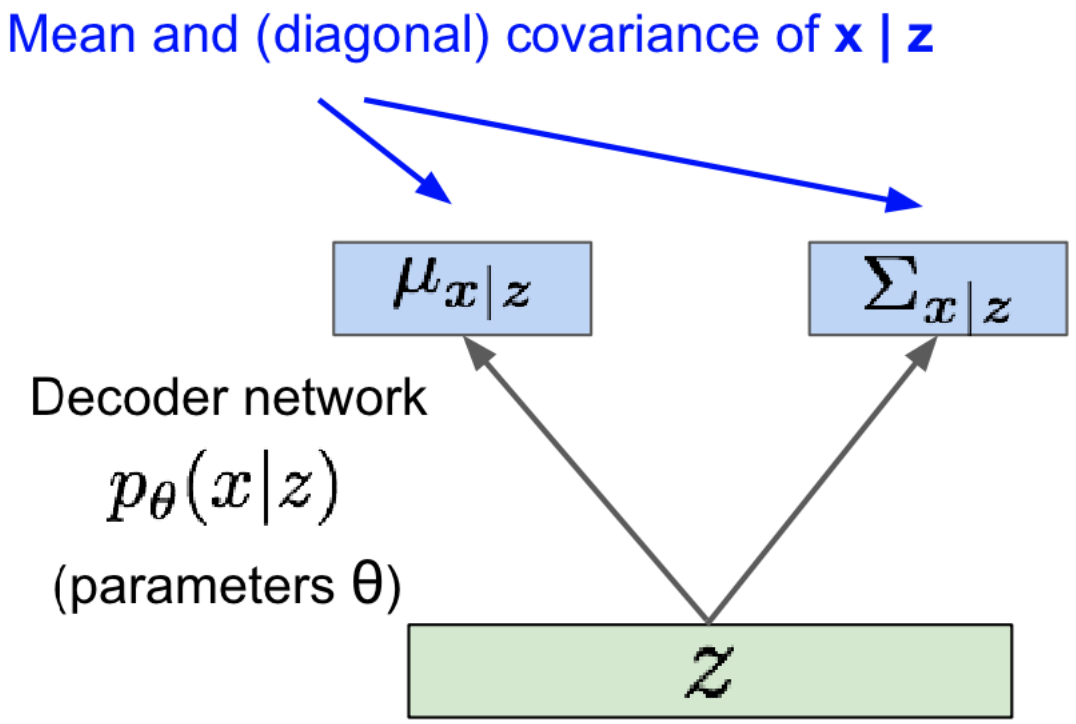
matches D as well as possible

- How do we model $p_\theta(x|z)$ using a neural network?



Modeling conditional distribution using neural networks

- Assume $p(x|z)$ has a specific form $x \mid z \sim N(\mu_{x|z}, \Sigma_{x|z})$, where $\Sigma_{x|z}$ is diagonal
- Use neural network (with weight θ) to model the *parameters* of the distribution

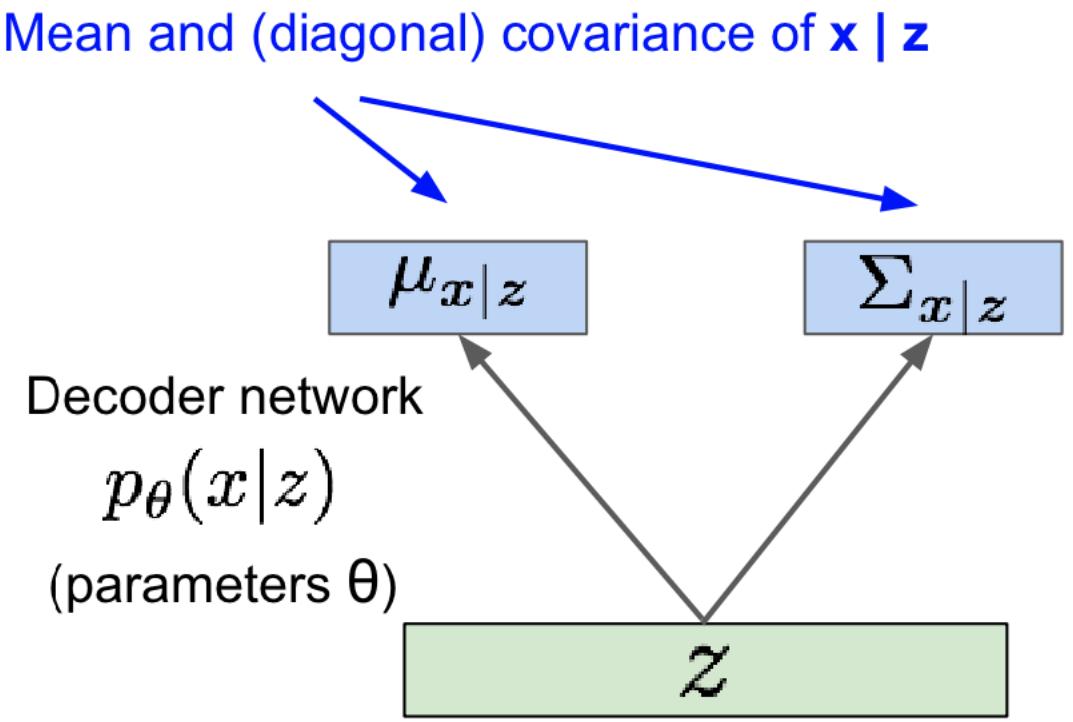


- It's now easy to sample from $p_\theta(x)$!
  - Draw $z \sim p(z)$
  - Draw $x \sim p_\theta(x \mid z)$:
    - Use network θ to compute $\mu_{x|z}, \Sigma_{x|z}$
    - Draw $\epsilon \sim \mathrm{N}(0, I_d)$
    - Compute $x_i = (\mu_{x|z})_i + \sqrt{(\Sigma_{x|z})_{ii}}\epsilon_i$

(fig from Stanford cs231n)

Variational autoencoder (VAE)

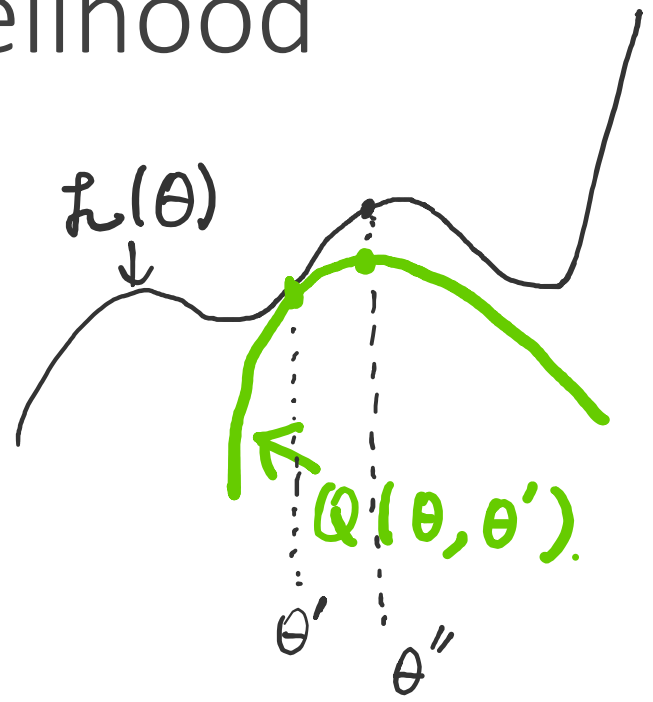
- Recall: $p_\theta(x) = \int p_\theta(x,z)\,dz = \int p_\theta(x|z)p(z)\,dz$
- Q: How can we train the network?
- MLE: $\text{maximize}_\theta \sum_{i=1}^{n} \ln p_\theta(x_i)$
- The issue:
  - Naively calculating $p_\theta(x_i)$ is intractable
- It takes 3 tricks!
  - evidence lower bound (ELBO)
  - auxiliary network
  - reparameterization trick



(fig from Stanford cs231n)

Recall: EM as maximizing surrogate likelihood

$$
\begin{aligned}
\mathcal{L}(\theta) &= \ln p(x;\theta) \\
&= \sum_z q(z) \ln p(x;\theta) \qquad \text{// for any choice of } q(\cdot) \\
&= \sum_z q(z) \ln \left(\frac{q(z)}{q(z)} \cdot \frac{p(x;\theta)p(x,z;\theta)}{p(x,z;\theta)} \right) \\
&= \underbrace{\sum_z q(z) \ln \left(\frac{p(x,z;\theta)}{q(z)} \right)}_{=: Q(\theta,\theta')} + \underbrace{\sum_z q(z) \ln \left(\frac{p(x;\theta)q(z)}{p(x,z;\theta)} \right)}_{=: \mathrm{KL}(q(\cdot) \,\|\, p(\cdot \mid x;\theta))}
\end{aligned}
$$



KL-divergence properties

$KL(p||q) \geq 0$, for all p,q

$KL(q||q) = 0$, for all q

- Previously in EM: choose $q(z) = p(z \mid x, \theta')$
- Due to computational intractability of $p(z \mid x, \theta')$, VAE does *not* choose q this way

KL divergence is an “asymmetric” distance measure

The first trick: maximizing ELBO instead

- $\log p(x) = \underbrace{\int q(z|x) \log \frac{p(x,z)}{q(z|x)} dz}_{=: \mathcal{L}(p,q)} + \mathrm{KL}(q(z|x) \,\|\, p(z|x))$

  (Just changed q(z) to q(z|x) from our previous derivation)

- $\mathcal{L}(p,q)$ is called the evidence lower bound (**ELBO**)
  - It is a lower bound of log-likelihood. Why?

- Key idea of VAE: Instead of $\underset{\theta}{\text{maximize}} \sum_{i=1}^{n} \log p_\theta(x_i)$, we perform

$$\underset{\theta,q}{\text{maximize}} \sum_{i=1}^{n} \mathcal{L}(x_i, p_\theta, q)$$

  where $q = q(z \mid x)$ is chosen from a family of conditional distributions

- Why this works: when q approximates $p_\theta(z \mid x)$ well, then ELBO also approximates likelihood well

Side note: alternative form of ELBO

- $\log p(x) = \int q(z|x) \log \frac{p(x,z)}{q(z|x)} dz + KL(q(z|x) \| p(z|x))$

  $=: \mathcal{L}(p, q)$

  Just changed q(z) to q(z|x) from our previous derivation.

- $\mathcal{L}(p,q)$ is called the evidence lower bound (ELBO)

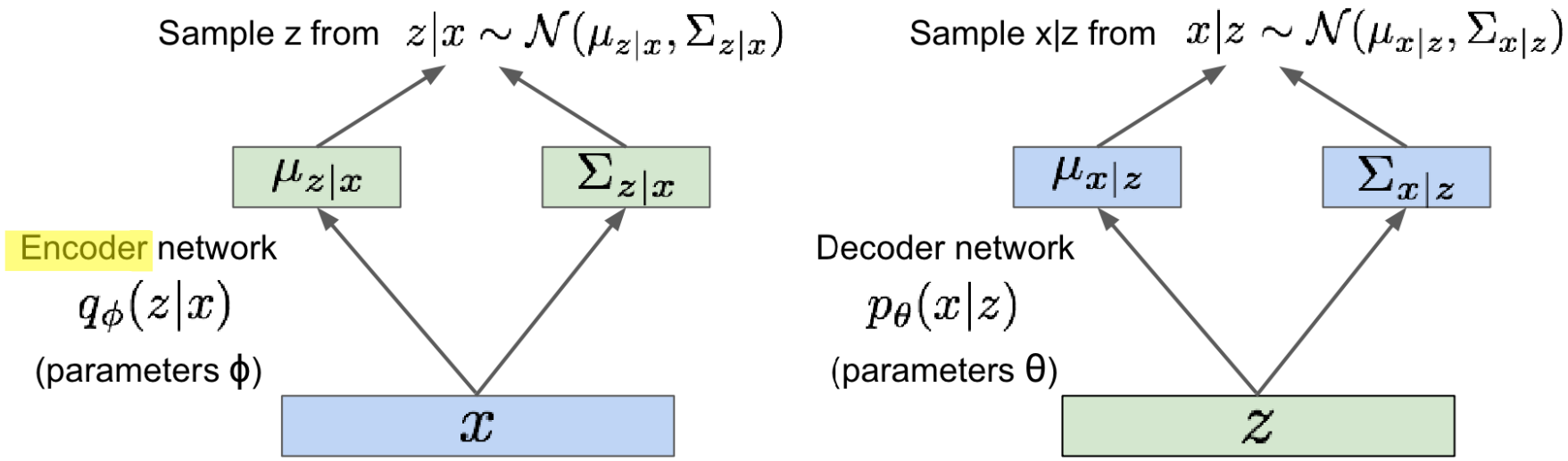
- $\mathcal{L}(p,q) = \int q(z|x) \log \frac{p(x,z)}{q(z|x)} dz = \int q(z|x) \log p(x|z)\, dz + \int q(z|x) \log \frac{p(z)}{q(z|x)} dz$

  $= \int q(z|x) \log p(x|z)\, dz - \mathrm{KL}(q(z|x) \| p(z))$

- Interpretation of maximizing ELBO over p, q:
  - Maximize "consistency" between p and q: p is MLE over data drawn from q
  - keep $q(z|x)$ not too far from the prior $p(z)$

The second trick: auxiliary network

- Note: the inequality works for any $q(z|x)$ (Q: which one is a good one?)
- Idea: build a neural network (with weight ϕ) to model $q(z|x)$ – denote it by $q_\phi(z|x)$
- We didn't intend it, but it now looks like the autoencoder!



(fig from Stanford cs231n)

Training

- Goal: $\underset{\theta,\phi}{\text{maximize}} \sum_x \int q_\phi(z|x) \log p_\theta(x|z)\, dz - KL\big(q_\phi(z|x) \,\|\, p(z)\big)$
- How to train? Gradient ascent on both θ and ϕ!
- The gradient for θ:

Rewrite the ELBO objective:

$$\mathcal{L}(p, q) = \mathbb{E}_{q_\phi(z|x)}\left[\ln p_\theta(x \mid z) - \ln \frac{q_\phi(z|x)}{p(z)}\right]$$

Exchange gradient with expectation (integration):

$$\nabla_\theta \mathbb{E}_{q_\phi(z|x)}\left[\ln p_\theta(x \mid z) - \ln \frac{q_\phi(z \mid x)}{p(z)}\right] = \mathbb{E}_{q_\phi(z|x)}\left[\nabla_\theta \ln p_\theta(x \mid z)\right]$$

Now, the consequence is that we can draw a sample $z' \sim q_\phi(z \mid x)$ and then compute the gradient $\nabla_\theta \ln p_\theta(x \mid z)$, which is an unbiased estimator for $\nabla_\theta \mathbb{E}_{q_\phi(z|x)}\left[\ln p_\theta(x \mid z) - \ln \frac{q_\phi(z|x)}{p(z)}\right]$. Thus, we can take the computed gradient as a *stochastic* gradient, and perform the standard SGD!

Training

$$\mathcal{L}(p,q) = \mathbb{E}_{q_\phi(z|x)}\left[\ln p_\theta(x \mid z) - \ln \frac{q_\phi(z|x)}{p(z)}\right]$$

The gradient for ϕ: $$\nabla_\phi \mathbb{E}_{q_\phi(z|x)}\left[\ln p_\theta(x \mid z) - \ln \frac{q_\phi(z \mid x)}{p(z)}\right]$$

Q: Can we exchange the gradient operator and the expectation operator?

No: change in ϕ affects both the integrand *and the underlying distribution*!

The third trick: reparameterization for gradient calculations

- key observation: $q_\phi(z \mid x)$ is the pdf of $\mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$

  Suppose this is a diagonal matrix

- then, a sample $z \sim q_\phi(\cdot \mid x)$ can be written as

$$x_i = \left(\mu_{x|z}\right)_i + \sqrt{\left(\Sigma_{x|z}\right)_{ii}}\epsilon_i, \text{ where } \epsilon_i \sim \mathrm{N}(0,1),\ i = 1, \ldots, d$$

- let's use the notation $z(\phi, x, \{\epsilon_i\})$ to make the dependence explicit. then,

$$\nabla_\phi \mathbb{E}_{q_\phi(z|x)}\left[\ln p_\theta(x \mid z) - \ln \frac{q_\phi(z \mid x)}{p(z)}\right] = \nabla_\phi \mathbb{E}_{\{\varepsilon_i\}}\left[\ln p_\theta(x \mid z(\phi, x, \{\varepsilon_i\})) - \ln \frac{q_\phi(z(\phi, x, \{\varepsilon_i\}) \mid x)}{p(z(\phi, x, \{\varepsilon_i\}))}\right]$$

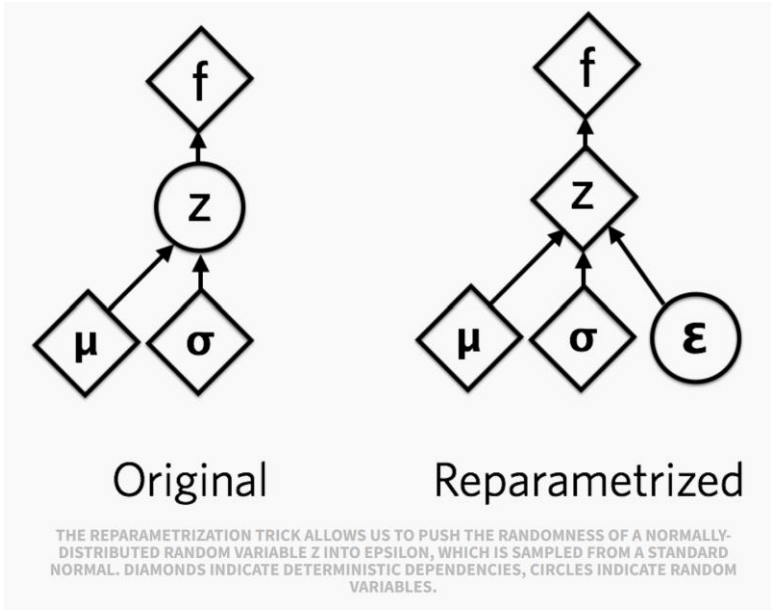
$$= \mathbb{E}_{\{\varepsilon_i\}}\left[\nabla_\phi\left(\ln p_\theta(x \mid z(\phi, x, \{\varepsilon_i\})) - \ln \frac{q_\phi(z(\phi, x, \{\varepsilon_i\}) \mid x)}{p(z(\phi, x, \{\varepsilon_i\}))}\right)\right]$$

- sample $\{\epsilon_i\}$, compute the gradient, and descend!

The reparameterization trick

- A general way to calculate unbiased estimates of $\nabla_\phi \mathbb{E}_{z\sim q_\phi} f(z)$

- "Pushes the randomness of z into ϵ"
  - Tutorial - What is a variational autoencoder? – Jaan Altosaar



THE REPARAMETRIZATION TRICK ALLOWS US TO PUSH THE RANDOMNESS OF A NORMALLY-DISTRIBUTED RANDOM VARIABLE Z INTO EPSILON, WHICH IS SAMPLED FROM A STANDARD NORMAL. DIAMONDS INDICATE DETERMINISTIC DEPENDENCIES, CIRCLES INDICATE RANDOM VARIABLES.

- It's about the way we define the distribution $q_\phi(z \mid x)$ through the normal distribution.
  - Let the network output the parameters of the normal distribution, so we can easily reparameterize it – disentangle the network output and the source of randomness!
  - More generally, the same technique can be applied to any exponential family distribution.

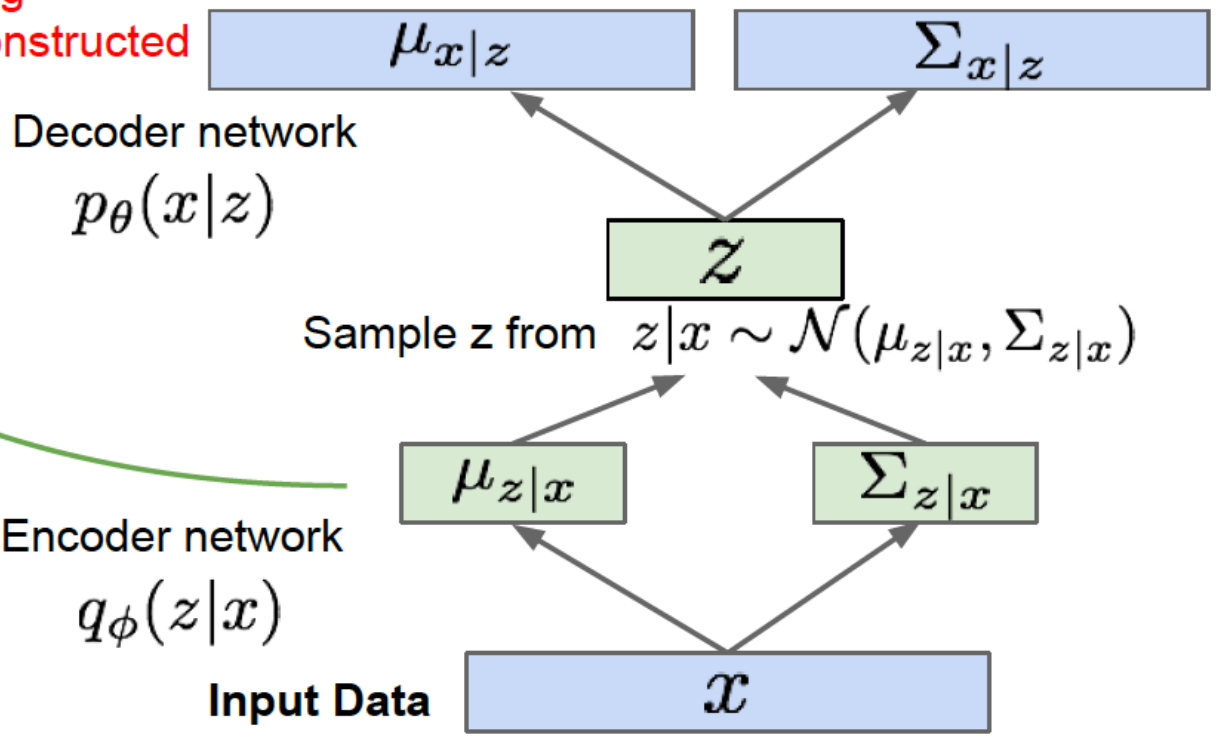
- Otherwise, computing the derivative is a pain (and researchers tried it and then found that it has a very high variance in the stochastic gradient..)

Training VAE Summary

- For each x, encoder forward pass, get $(\mu_{z|x}, \Sigma_{z|x})$, sample $\{\epsilon_i\}$, compute z, decoder forward pass, get $(\mu_{x|z}, \Sigma_{x|z})$, compute the ELBO.
- Then, compute the gradients for both encoder ϕ and decoder θ, perform gradient ascent.

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

Maximize likelihood of original input being reconstructed



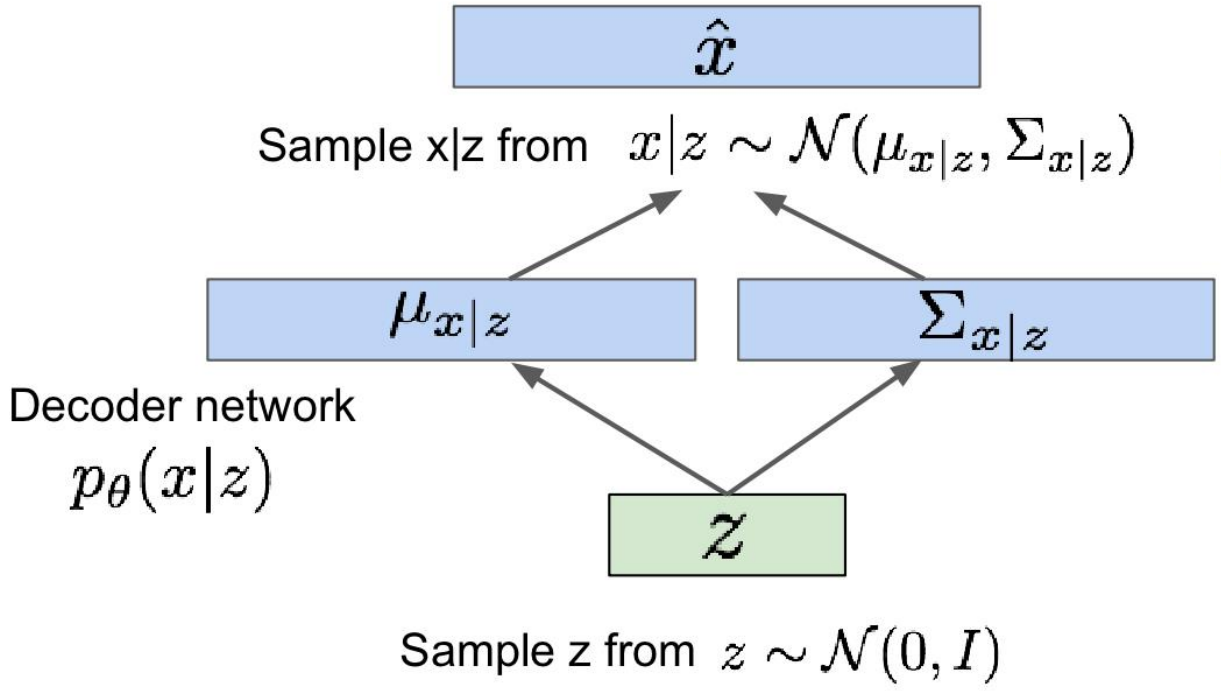
Make approximate posterior distribution close to prior

Implementation detail: the KL term has a closed form – exact gradient can be obtained.
=> use sample-based gradients for the first term only.

(fig from Stanford cs231n)

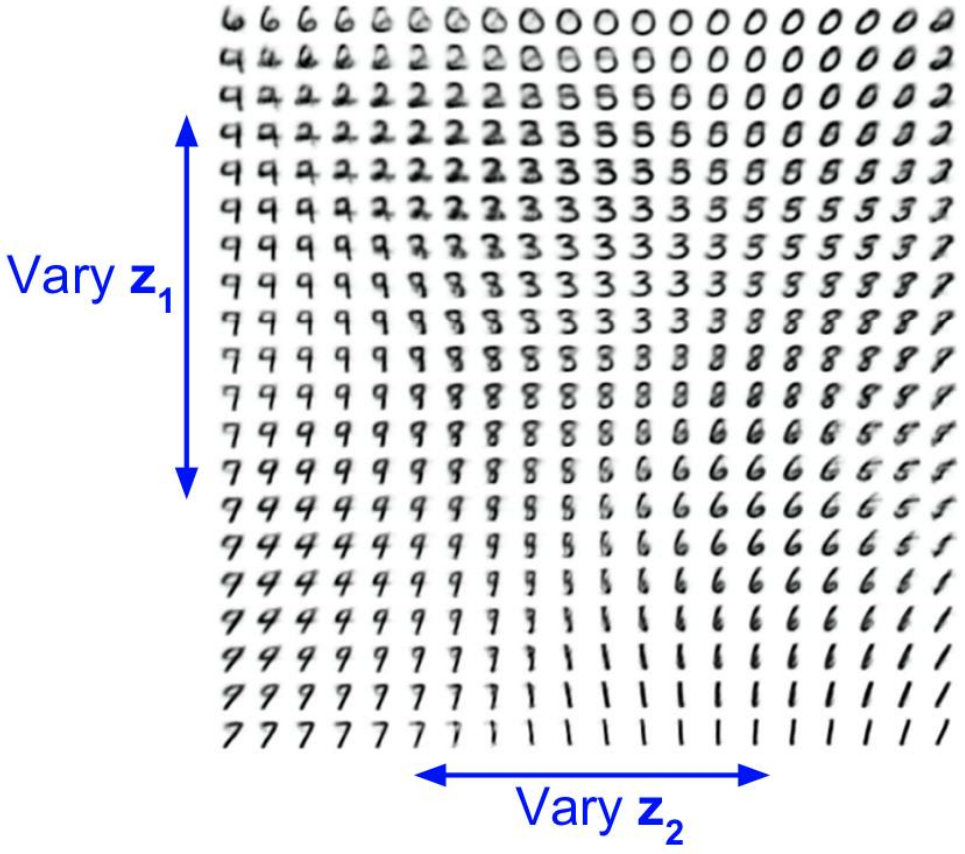
VAE: interpreting the decoder network

Use decoder network. Now sample z from prior!



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Data manifold for 2-d **z**



(fig from Stanford cs231n)

VAE: interpreting the decoder network

Diagonal prior on **z**
=> independent
latent variables

Different
dimensions of **z**
encode
interpretable factors
of variation

Also good feature representation that can be computed using $q_\phi(z|x)$!

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

(fig from Stanford cs231n)

Takeaways

- ELBO is a very useful technique.
- Tools change, but the principles live long!
- How one combines NNs with probabilistic models – place NN for parameters of well-known distributions (e.g. Gaussian)

- Additional resources for VAEs:
  - Blog 1: https://ermongroup.github.io/cs228-notes/extras/vae/
  - Blog 2 (with code): https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
  - Tutorial paper: https://arxiv.org/pdf/1606.05908.pdf
  - Kingma and Welling, “An Introduction to Variational Autoencoders”, Foundations and Trends in Machine Learning, 2019.

Next lecture (11/23)

- More application of GANs
- Reinforcement learning

- Assigned reading: https://jaan.io/what-is-variational-autoencoder-vae-tutorial/ by Jaan Altosaar

Generative adversarial network (GAN)

VAEs tend to generate blurry images

- Especially towards the corner.
- Why? (next slide)
- Generative adversarial networks (GANs): sharper images!
  - Cons: does not explicitly learn a distribution; specialized for sampling only.

GAN paper: Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

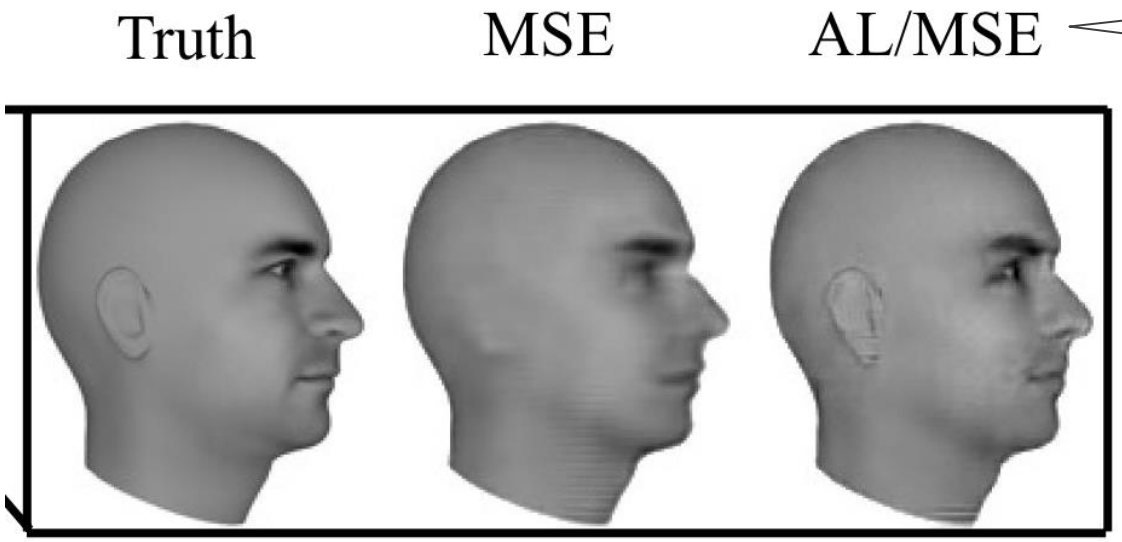
VAE

GAN

fig. from Larsen et al., Autoencoding beyond pixels using a learned similarity metric, 2016

Why are VAEs generating blurry images?

- One hypothesis: the loss imposed is inherently **mean squared error** – minimizing this tends to learn the 'average'.
  - Some says that it could be that we are using diagonal covariance.
  - At least, it seems to be true for the vanilla VAE.



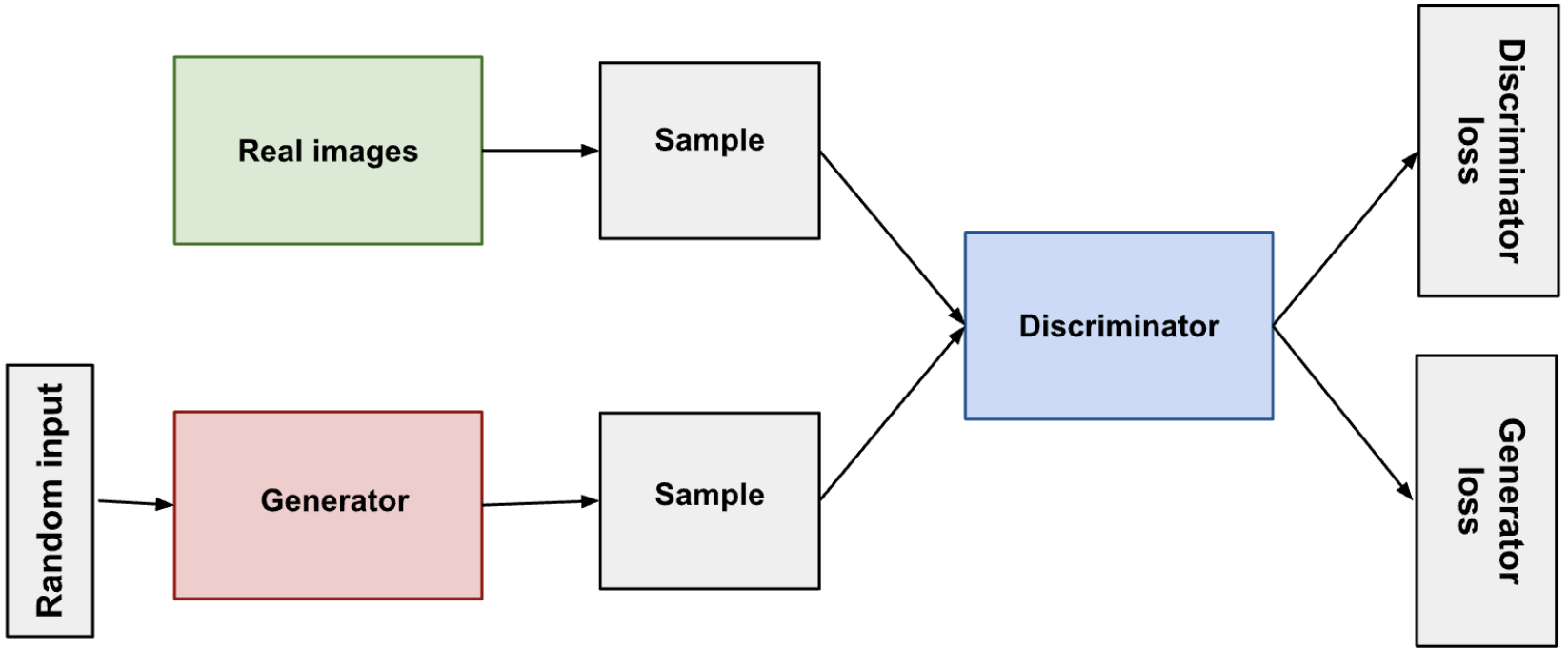
This paper combined MSE loss with the loss of GAN to make it converge faster.

Lotter et al., unsupervised learning of visual structure using predictive generative networks, 2016

Reddit discussion: https://www.reddit.com/r/MachineLearning/comments/9t712f/dwhy_are_images_created_by_gan_sharper_than/

GAN: high-level structure

- **Discriminator** network: try to distinguish between real and fake images
- **Generator** network: try to fool the discriminator by generating real-looking images



- Note: the discriminator network is distinguishing real and fake *distributions*, not individual examples

fig. from https://developers.google.com/machine-learning/gan/gan_structure

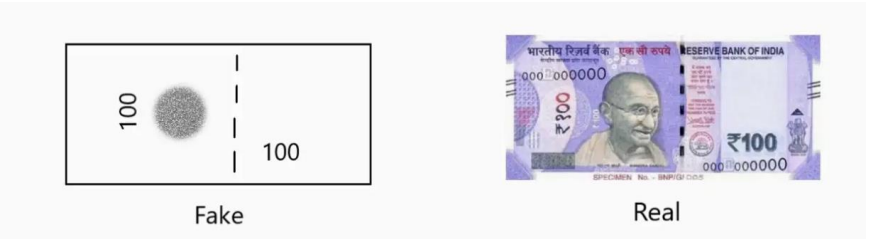
GANs: optimization formulation

- Solving a minimax game:
- $$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim \mu} \log \left(1 - D_{\theta_d}\left(G_{\theta_g}(z) \right) \right) \right]$$

discriminator output $\in (0,1)$ for real data x

discriminator output for fake data

- θ_g: generator network parameters – counterfeiters
  - would like $D(G(z))$ is close to 1 (real) for fake data
- θ_d: discriminator network parameters – police officers
  - would like $D(x)$ close to 1 (real) and $D(G(z))$ close to 0



https://towardsdatascience.com/generative-adversarial-networks-gans-8fc303ad5fa1

GANs: optimization formulation

- What is the inner maximization problem?
- Simplifications
  - Let $p = p_{data}$, $q = G_{\theta_g}\#\mu$, the distribution of $G_{\theta_g}(z)$ where $z \sim \mu$
  - Let D_{θ_d} be able to represent any function
- The inner maximization problem becomes

$$\max_{D:\mathcal{X}\to[0,1]} \left[\mathbb{E}_{x\sim p} \ln D(x) + \mathbb{E}_{x\sim q} \ln\left(1 - D(x)\right)\right]$$

- Optimal discriminator D: $D(x)$ minimizes $p(x) \ln D(x) + q(x) \ln\left(1 - D(x)\right)$

$$\Rightarrow D(x) = \frac{p(x)}{p(x)+q(x)} = P(y = \text{real image}|x)$$

- Optimal objective value:

$$\mathbb{E}_{x\sim p} \ln \frac{p(x)}{p(x)+q(x)} + \mathbb{E}_{x\sim q} \ln \frac{q(x)}{p(x)+q(x)} = KL\left(p, \frac{p+q}{2}\right) + KL\left(q, \frac{p+q}{2}\right) - 2\ln 2 = JS(p,q) - 2\ln 2$$

- $JS(p,q)$: Jensen-Shannon divergence

GANs: optimization formulation

- With the simplification above:

$$\min_{\theta_g} JS\left(p_{data}, G_{\theta_g}\#\mu\right)$$

- Generator θ_g wants to **minimize** the distance between $G_{\theta_g}\#\mu$ and p_{data}

Training GANs

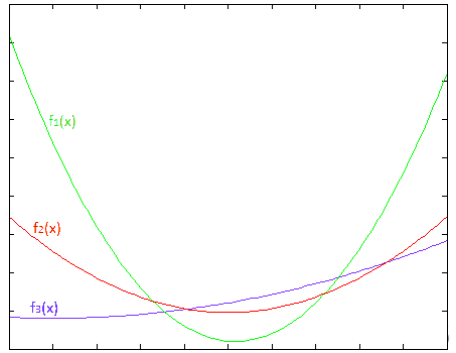
$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim \mu} \log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

- How to solve the outer minimization problem?
- It is of the form

$$\underset{x}{\text{minimize}}\ g(x), \text{ where } g(x) = \max_{y} f(x, y)$$

- Gradient descent! But how to calculate $\nabla_x g(x)$?
- Fact: if f is "nice enough", then $\nabla_x g(x) = \nabla_x f(x, y^*(x))$, where $y^*(x) = \text{argmax}_y f(x, y)$

- Implication: to calculate descent direction for θ_g, find corresponding optimal θ_d



https://math.stackexchange.com/questions/373229/show-that-the-maximum-of-a-set-of-convex-functions-is-again-convex

Training GANs (cont'd)

Alternate between

- Maximization on discriminator

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

  - Often approximated by a few gradient descent iterations

- Gradient descent on generator

$$\min_{\theta_g} \left[\mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

Training GANs

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim \mu} \log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

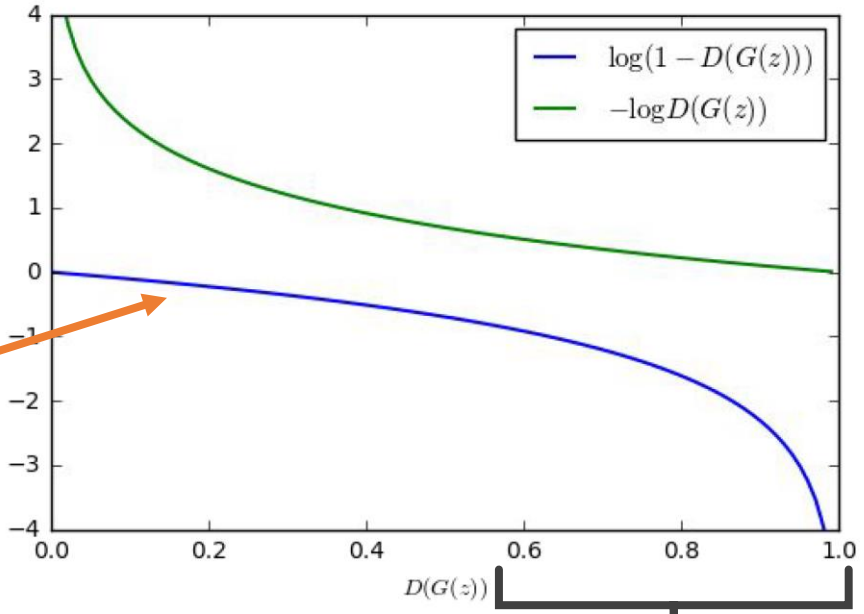
Alternate between

- Maximization on discriminator

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

- ~~Gradient descent on generator~~ => small gradients issue
- Gradient ascent on an alternative objective

$$\max_{\theta_g} \mathbb{E}_{z \sim \mu} \left[\log D_{\theta_d} \left(G_{\theta_g}(z) \right) \right]$$



at the beginning where the generator sucks..

where the generator wants to be

Training GANs

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

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

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

$\log(D(G(z^{(i)})))$

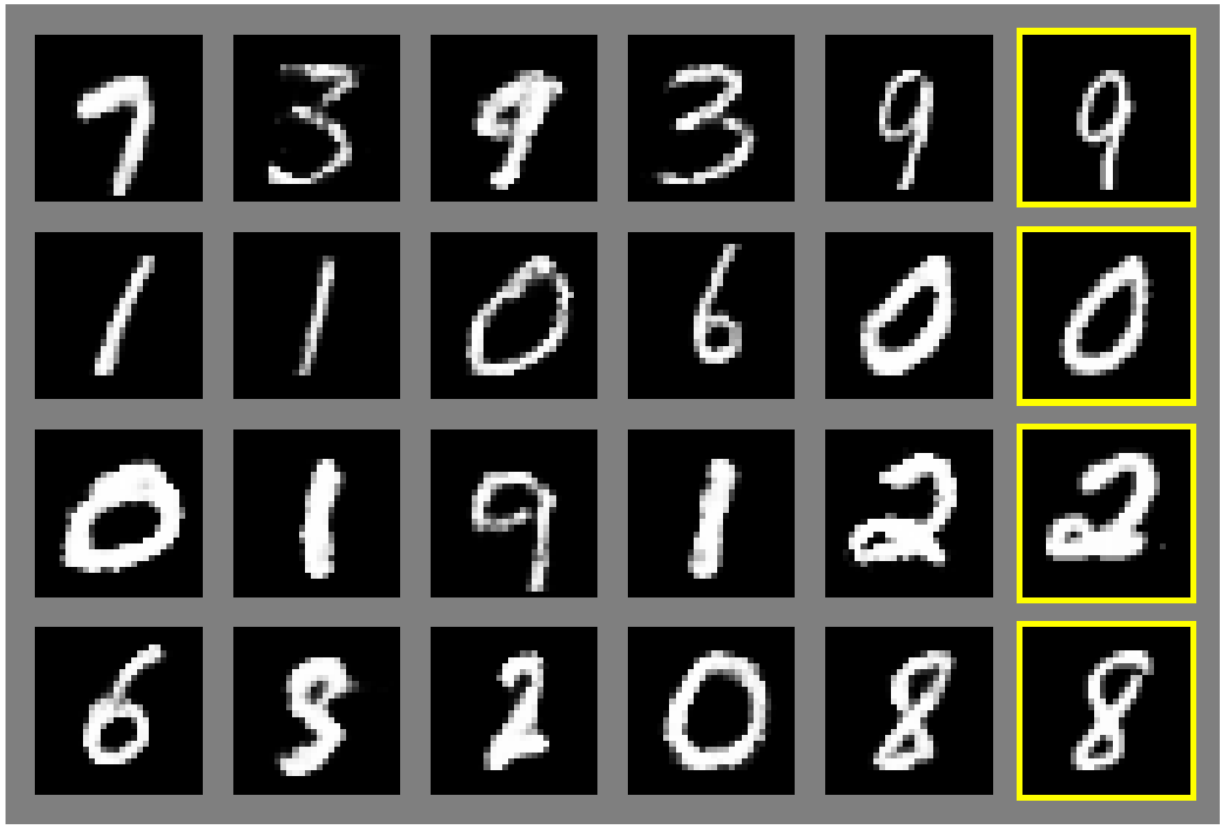
end for

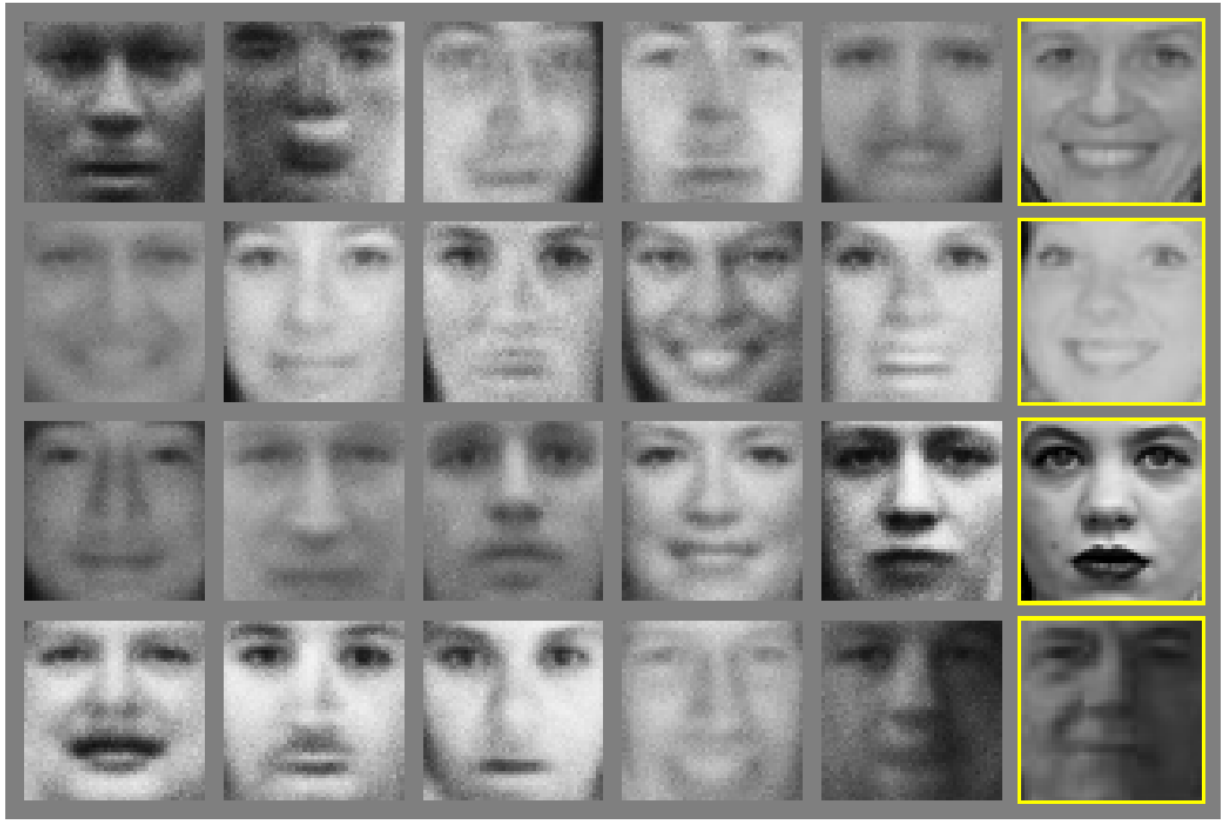
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Ian Goodfellow et al., "Generative Adversarial Nets", NeurIPS 2014

GAN samples

Ian Goodfellow et al., “Generative Adversarial Nets”, NeurIPS 2014

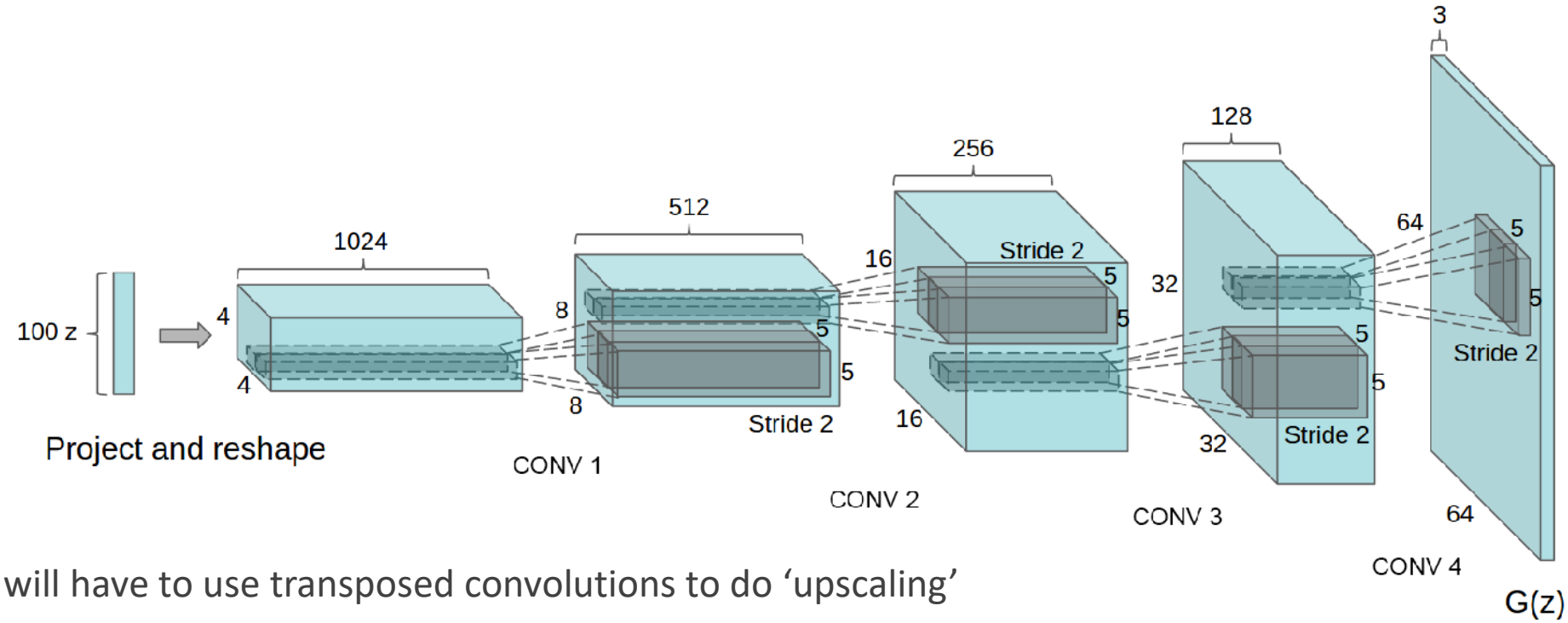




nearest data point from the training set (this shows that the network is **not memorizing**)

Architecture example

- Radford et al., UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS, 2016.

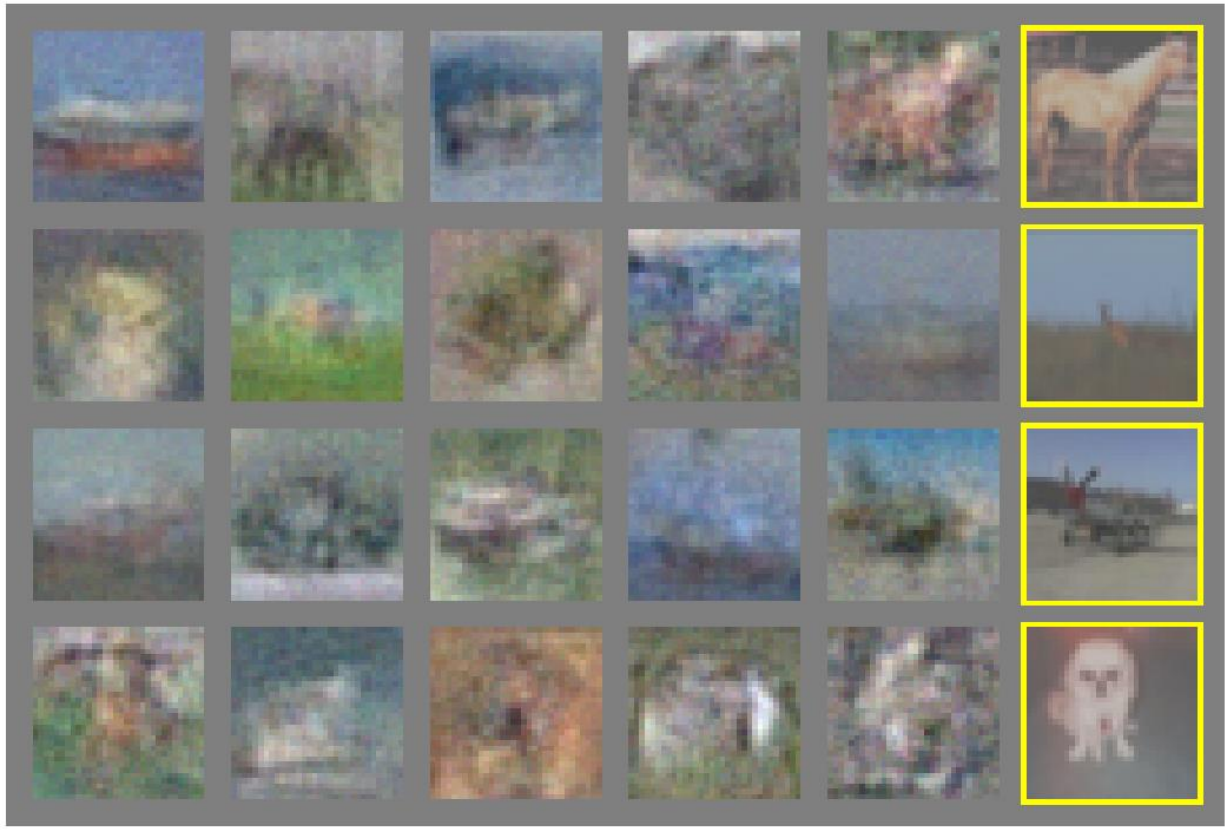


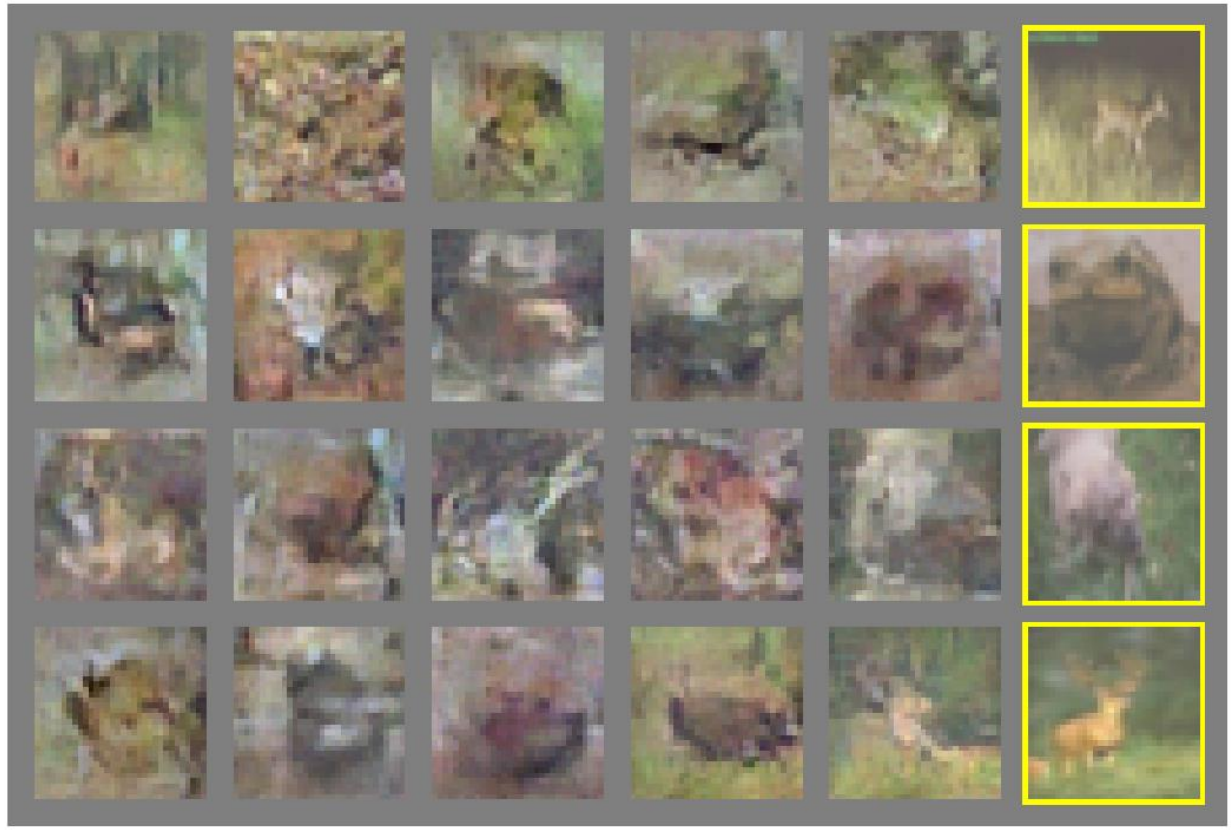
Note: you will have to use transposed convolutions to do 'upscaling'

Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

GAN samples

Ian Goodfellow et al., “Generative Adversarial Nets”, NeurIPS 2014





(not using conv layers)

GAN with conv layers

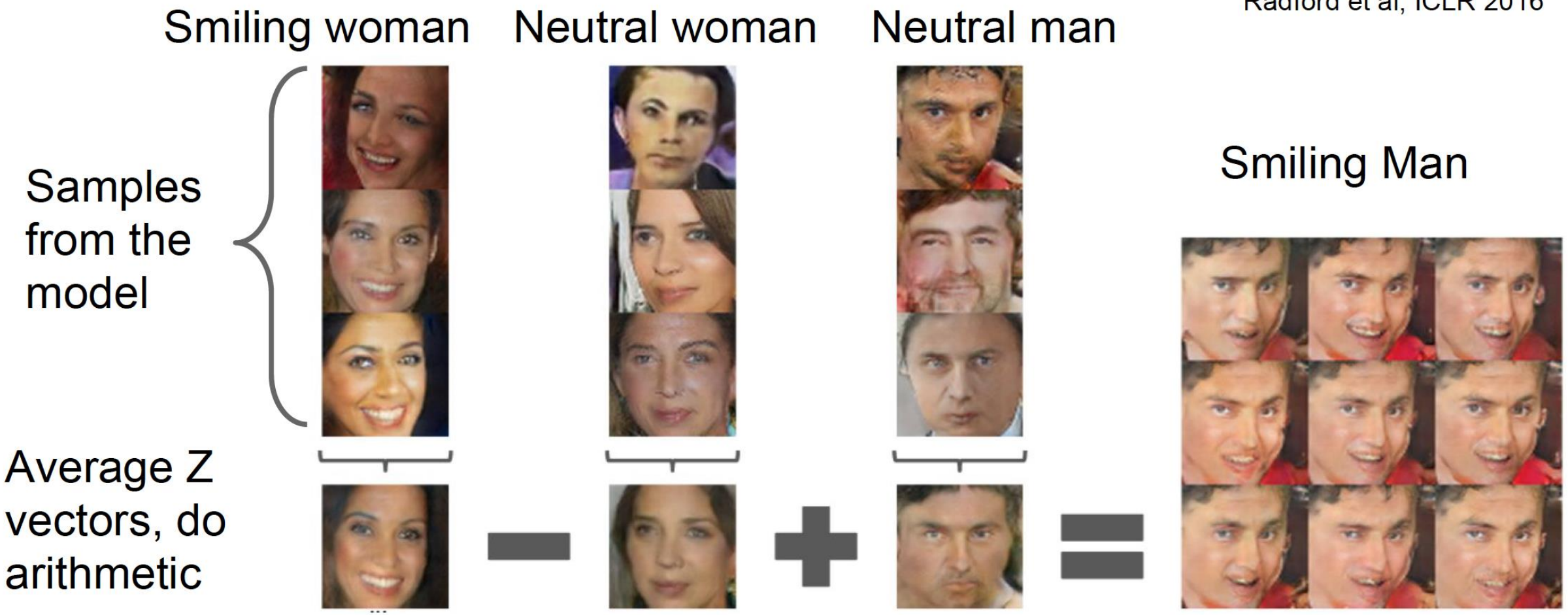
Radford et al, ICLR 2016

GAN with conv layers

- Possible to interpolate between two samples. The learned representation is really useful!

Learned representation is meaningful

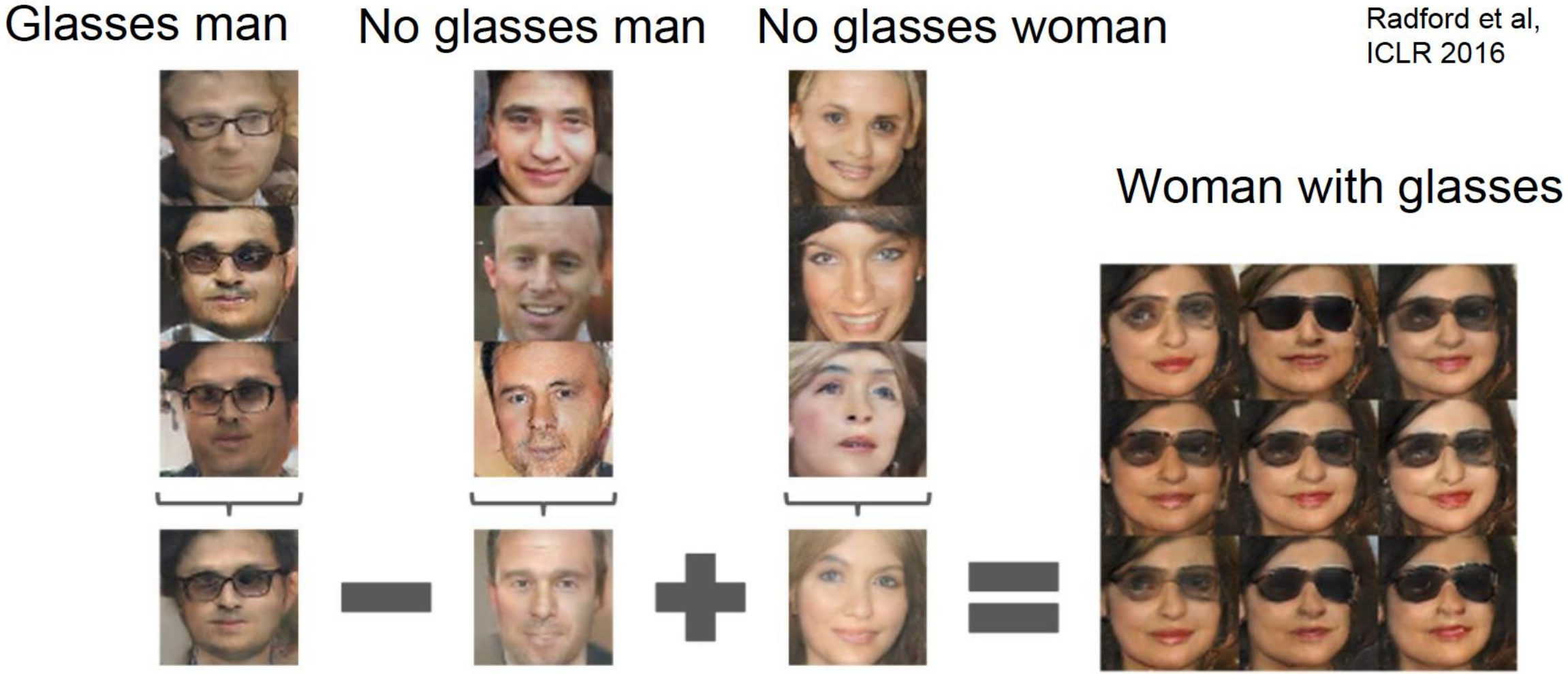
Radford et al, ICLR 2016



- This means that there are 'directions' in the representation z that have particular meanings!

(slide from Stanford CS231n)

Learned representation is meaningful



Miscellanea

- Training GANs are notoriously hard; it's an active area of research.
  - Generic tips: https://github.com/soumith/ganhacks
- The fact that we require significantly small dimensions forces the network to learn **compact representations**.
  - E.g., binary codes
  - E.g., represent objects as 'combinations' of commonly observed patterns like smile face + female face + glasses.

Summary

- Generative models for unsupervised learning

	Trains an encoder	explicit probabilistic model	able to sample easily from D
Autoencoder	Yes	No	Unclear
Variational autoencoder	Yes	Yes	Yes
Generative adversarial networks	No	Yes	Yes

Next lecture (11/28)

- Reinforcement learning: Markov Decision Processes (MDPs), Planning in MDPs

- Assigned reading: “Generative Adversarial Nets” (NeurIPS 2014) by Goodfellow et al