

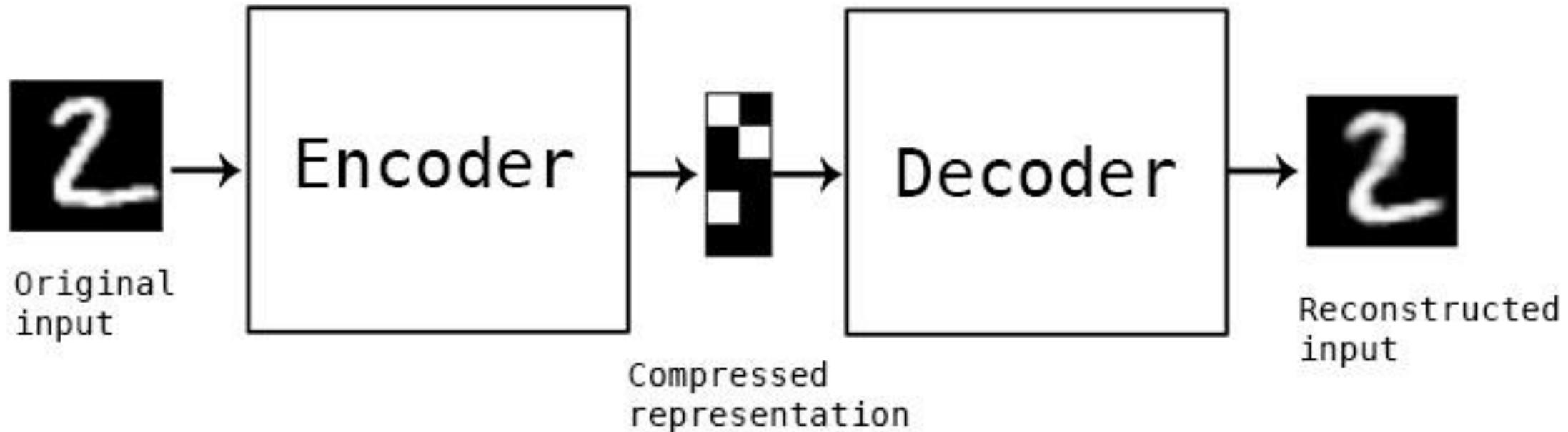
# Autoencoders and Variational Autoencoders

Gautam Kamath

# Autoencoder

- A type of “compression”
- Input:  $x \in \mathbf{R}^d$
- Encoder:  $f: \mathbf{R}^d \rightarrow \mathbf{R}^m$ , Decoder:  $g: \mathbf{R}^m \rightarrow \mathbf{R}^d$
- Goal:  $g(f(x)) = x$
- Trivial if  $m = d$ : just let  $f(x) = x$  and  $g(x) = x$
- Interesting when  $m \ll d$  (e.g.,  $d = 1000, m = 10$ )

# Autoencoder



# Linear Autoencoder

- (Draw simple autoencoder, label weights  $W_f$  and  $W_g$  and bottleneck)
- Output:  $W_g W_f x$
- How to optimize? Use objective

$$\min_{W_f, W_g} \sum_i \frac{1}{2} \|W_g W_f x_i - x_i\|_2^2$$

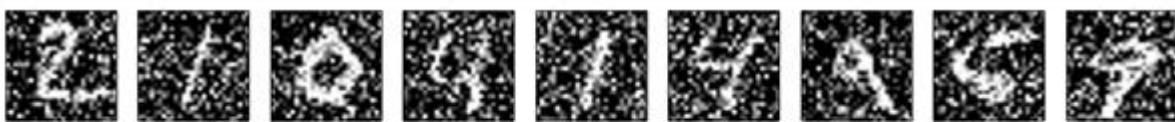
- $W_f x$  is a compression of  $x$
- With linear autoencoder, similar to principal component analysis (PCA) (draw)

# Nonlinear Autoencoder

- $f$  and  $g$  are non-linear (draw non-linear auto-encoder, label  $W_f, W_g$ )
- $\min_{W_f, W_g} \sum_i \frac{1}{2} \|g(f(x_i)) - x_i\|_2^2$
- Deep autoencoder (draw)

# Other Autoencoders

- Sparse autoencoders
  - Encourage a sparse encoding of input
  - May have wider bottleneck layer (draw)
  - $\min_{W_f, W_g} \sum_i \frac{1}{2} \|g(f(x_i)) - x_i\|_2^2 + \lambda \|f(x_i)\|_1$
- Denoising autoencoders
  - Given noised input  $\tilde{x}$ , produce denoised  $x$  as output (draw)
  - $\min_{W_f, W_g} \sum_i \frac{1}{2} \|g(f(\tilde{x}_i)) - x_i\|_2^2$



# Uses of Autoencoders

- Can “detach” input and output, use separately
- Can compress data to a smaller dimension
- Can find interesting representations of data
- Generally, finds some underlying structure of the dataset
- However, is not useful to understand *distribution* of dataset
  - In particular, can’t necessarily generate new images

# Generative Modelling

- Given  $X_1, \dots, X_n \sim D$ , can we generate  $X_{n+1}, X_{n+2}, \dots$ ?
  - Ideally from  $D$ , but actually from something *close* to  $D$
- $D$  may be more complex than a GMM
  - E.g., the distribution of all handwritten numbers, or ImageNet (draw)
- Solution: use a neural network to do the work
- Draw a sample from  $N(0, I)$ , use an NN to map it to a sample from  $D$
- (Draw NN version, where low d Gaussian mapped to high d output)
- Actually: use variational autoencoder (VAE)

# Variational Autoencoder

- (Draw encoder, from  $x \in \mathbf{R}^d$  to  $\mu(x), \sigma(x) \in \mathbf{R}^m$ , decoder from  $z \sim N(\mu(x), \text{diag}(\sigma(x))) \in R^m$  to  $\tilde{x} \in \mathbf{R}^d$ )

# Variational Autoencoder (VAE)

- Some notation:  $x$ 's live in the *data* space (in  $\mathbf{R}^d$ ), while  $z$ 's live in the *latent* space (in  $\mathbf{R}^m$ ).  $p_\theta$  is the decoder network's distribution,  $q_\phi$  is the encoder network's distribution
- E.g.,  $p_\theta(x)$  is density of decoder network's outputs.  $p_\theta(x|z)$  is density of decoder network's outputs, *conditioned on* some latent vector input  $z$ .  $p_\theta(z)$  is density of decoder network's latent vector input.  $q_\phi(z|x)$  is distribution of encoder network's outputs, *conditioned on* some data input  $x$ 
  - $p_\theta(z)$  generally chosen to be  $N(0, I)$
  - Why does  $p_\theta(x|z)$  have a distribution? Isn't it deterministic? For loss calculation, we assume the output of the network is fed into a Gaussian sampler. Will revisit shortly.
  - (Draw mapping from data space to latent space and back)

# VAE Goals

- Ensure that input image distribution maps to latent distribution  $N(0, I)$  (draw)
  - Minimize  $KL\left(q_{\phi}(z|x) \parallel p_{\theta}(z)\right) = KL\left(q_{\phi}(z|x) \parallel N(0, I)\right)$  (draw lines)
- Similar to autoencoder, ensure that an input gets encoded and mapped back to itself
  - Maximize  $E_{z \sim q_{\phi}(\cdot|x)}[\log p_{\theta}(x|z)]$
- Claim:  $\log p_{\theta}(x) \geq E_{z \sim q_{\phi}(z|x)}[\log p_{\theta}(x|z)] - KL\left(q_{\phi}(z|x) \parallel N(0, I)\right)$ 
  - Similar to the inequality when doing EM
  - Bigger picture: variational inference

# Optimizing: Minimize KL divergence

- $\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - KL(q_\phi(z|x)||N(\mathbf{0}, I))$
- $KL(q_\phi(z|x)||N(0, I)) = KL(N(\mu_\phi(x), \text{diag}(\sigma_\phi^2(x)))||N(0, I))$
- For two Gaussians, this KL divergence has a simple expression  
$$= \frac{1}{2} \left( \|\mu_\phi(x)\|_2^2 - m + \sum_{j=1}^m (\sigma_\phi^2(x)_j - \log(\sigma_\phi^2(x)_j)) \right)$$
- Sanity check: what if  $\mu_\phi(x) = 0$  and  $\sigma_\phi^2(x)_j = 1$  for all  $j$ ?

# Optimizing: Autoencoding points

- $\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - KL(q_\phi(z|x) || N(0, I))$
- We imagine the density  $p_\theta(x|z)$  is that of  $N(\mu_\theta(z), I)$  where  $\mu_\theta$  is the decoder network
  - When sampling, can instead just output  $\mu_\theta(z)$  rather than additional sampling
    - Analogy: when we run softmax on outputs of an NN, we output the max index, we don't sample from it
- $E_{z \sim q_\phi(z|x)} [\|x - \mu_\theta(z)\|_2^2] - d \log \sqrt{2\pi}$  (essentially same as AE)
- Given sampling capability, can draw  $z \sim q_\phi(z|x)$  to optimize
- Reparameterization trick (Draw how to sample  $Z \sim N(\mu, \sigma^2)$  as  $\mu + \sigma G$  where  $G \sim N(0, 1)$ )

# Summary

- Solve generative modelling
- Use neural network to map Gaussian samples to data distribution
- Do it by using variational autoencoder: tries to map original distribution to a Gaussian, and also maps back to original distribution. Each is encoded in the loss function.

# Samples from a VAE

8817814828  
9683960319  
5391369179  
8908691463  
9233331386  
6998616666  
9526651899  
9989312823  
0461232089  
9754934851

3165767672  
8594682168  
6103288433  
2868910041  
5193015359  
6561491788  
1343983270  
4582970958  
6944872393  
2645609798

2838385738  
8382793538  
3599239516  
1918333197  
2736430203  
5970593845  
6943628552  
8490807066  
7436303601  
2180471850

7208923900  
751991171944  
8962082829  
2984387461  
5479898910  
6884948281  
7582361383  
7939279390  
4524390184  
8872316236

(a) 2-D latent space

(b) 5-D latent space

(c) 10-D latent space

(d) 20-D latent space

# Interpolation using VAEs (Explain how)

A 10x10 grid of handwritten digits, likely generated by a Variational Autoencoder (VAE) during an interpolation process. The digits transition smoothly from '6' at the top-left to '7' at the bottom-right, illustrating the latent space manifold. The grid includes:

- Row 1: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
- Row 2: 5, 4, 4, 2, 2, 2, 2, 2, 2, 2
- Row 3: 5, 2, 2, 2, 2, 2, 2, 2, 2, 2
- Row 4: 5, 9, 2, 2, 2, 2, 2, 2, 2, 2
- Row 5: 5, 9, 4, 2, 2, 2, 2, 2, 2, 2
- Row 6: 5, 9, 9, 4, 2, 2, 2, 2, 2, 2
- Row 7: 5, 9, 9, 9, 2, 2, 2, 2, 2, 2
- Row 8: 5, 9, 9, 9, 9, 3, 3, 3, 3, 3
- Row 9: 5, 9, 9, 9, 9, 9, 3, 3, 3, 3
- Row 10: 5, 9, 9, 9, 9, 9, 9, 3, 3, 3
- Row 11: 5, 9, 9, 9, 9, 9, 9, 9, 3, 3
- Row 12: 5, 9, 9, 9, 9, 9, 9, 9, 9, 3
- Row 13: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 14: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 15: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 16: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 17: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 18: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 19: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 20: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 21: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 22: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 23: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 24: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 25: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 26: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 27: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 28: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 29: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 30: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 31: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 32: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 33: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 34: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 35: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 36: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 37: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 38: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 39: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 40: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 41: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 42: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 43: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 44: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 45: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 46: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 47: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 48: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 49: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 50: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 51: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 52: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 53: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 54: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 55: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 56: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 57: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 58: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 59: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 60: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 61: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 62: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 63: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 64: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 65: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 66: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 67: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 68: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 69: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 70: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 71: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 72: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 73: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 74: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 75: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 76: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 77: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 78: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 79: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 80: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 81: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 82: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 83: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 84: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 85: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 86: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 87: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 88: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 89: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 90: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 91: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 92: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 93: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 94: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 95: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 96: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 97: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 98: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 99: 5, 9, 9, 9, 9, 9, 9, 9, 9, 7
- Row 100: 7, 7, 7, 7, 7, 7, 7, 7, 7, 7