

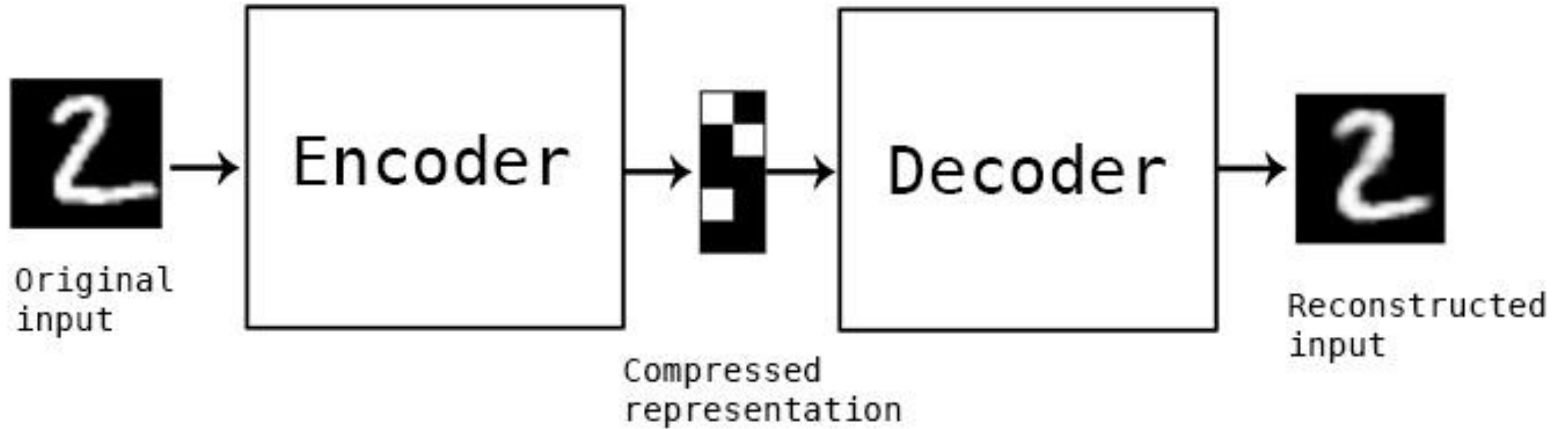
Autoencoders and Variational Autoencoders

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 $z \sim N(\mu(x), \text{diag}(\sigma(x))) \in \mathbf{R}^m$ 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_θ is the decoder network's distribution, q_ϕ 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 $KL(q_\phi(z|x) || p_\theta(z)) = KL(q_\phi(z|x) || N(0, I))$ (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: $\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - KL(q_\phi(z|x) || N(0, I))$
 - Similar to the inequality when doing EM
 - Bigger picture: variational inference

Optimizing: Minimize KL divergence

- $\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - \mathbf{KL}(q_\phi(z|x) || N(\mathbf{0}, I))$
- $\mathbf{KL}(q_\phi(z|x) || N(0, I)) = \mathbf{KL}(N(\mu_\phi(x), \text{diag}(\sigma_\phi^2(x))) || N(0, I))$
- 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

- $\log p_\theta(x) \geq \mathbf{E}_{z \sim q_\phi(z|x)} [\mathbf{log} p_\theta(x|z)] - KL(q_\phi(z|x) || N(0, I))$
- We imagine the density $p_\theta(x|z)$ is that of $N(\mu_\theta(z), I)$ where μ_θ 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

8 6 7 7 8 1 4 8 2 8
9 6 8 9 9 6 8 3 1 9
5 9 7 1 3 6 9 1 7 9
8 9 0 8 6 9 1 9 6 3
8 2 3 3 3 3 1 3 8 6
6 9 9 8 6 1 6 6 6 6
9 5 2 6 6 5 1 8 9 9
9 9 8 7 8 7 2 8 2 3
0 4 6 1 2 3 2 0 8 8
9 9 5 4 9 3 4 8 5 1

(a) 2-D latent space

5 1 6 5 7 0 7 6 7 2
8 5 8 4 6 8 2 1 6 2
6 9 5 3 2 8 8 1 3 8
2 8 6 8 9 1 2 0 4 1
5 1 7 2 0 1 5 3 5 9
6 5 6 2 4 9 1 7 8 8
1 3 4 3 9 2 3 2 7 0
4 5 8 2 9 7 0 1 6 9
6 9 4 4 8 7 2 3 2 3
2 6 4 5 6 0 9 9 9 8

(b) 5-D latent space

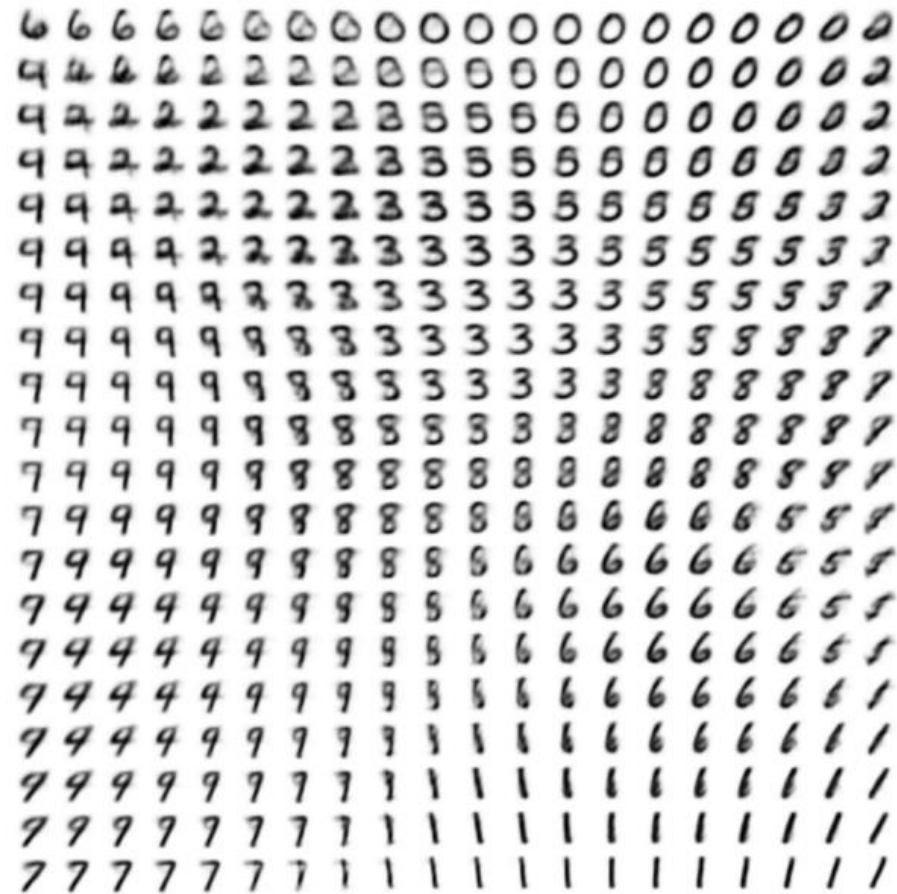
2 8 7 1 3 8 5 7 3 8
8 3 8 2 7 9 2 3 3 8
3 5 9 9 2 2 9 5 1 6
1 9 2 8 8 3 2 1 9 7
2 7 3 6 4 2 0 2 6 3
5 9 7 0 5 8 2 3 4 5
6 9 4 3 6 2 8 5 5 7
8 4 9 0 5 0 7 0 6 6
7 4 1 6 2 0 3 6 0 1
2 7 2 0 4 7 7 9 6 0

(c) 10-D latent space

8 2 0 8 7 2 3 7 0 0
7 5 9 9 1 1 7 1 4 4
8 9 6 2 0 8 2 8 2 9
2 9 8 6 3 1 7 0 6 7
5 7 7 1 8 9 7 9 1 0
6 8 2 4 3 4 8 2 8 7
7 5 8 2 1 6 1 3 8 8
7 9 3 9 2 7 9 3 9 6
4 5 2 4 3 9 0 1 6 4
8 8 7 2 5 1 6 2 3 2

(d) 20-D latent space

Interpolation using VAEs (Explain how)



The image displays a 20x20 grid of handwritten digits, illustrating a smooth interpolation from the digit '6' to the digit '1'. The digits are arranged in a grid where each row represents a different point in the interpolation space. The top row consists of 20 '6's. As you move down the rows, the digits gradually transform, showing intermediate forms that blend the features of '6' and '1'. The bottom row consists of 20 '1's. The overall effect is a continuous and smooth transition between the two digits.