Generative Models

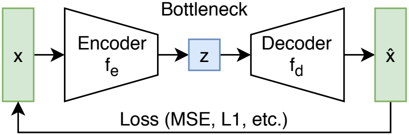
ETH Zürich

7–8 May 2020

Overview

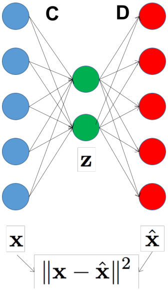
(Variational) Autoencoders

Autoencoder (paradigm)



- A form of unsupervised learning
- **Applications**: dimensionality reduction, compression, representation learning, pretraining/semi-supervised learning
- Encoder-decoder architecture with reconstruction loss
  - Encoder (latent code): $\mathrm{z} = f_e(\mathrm{x})$
  - Decoder (reconstruction): $\hat{\mathrm{x}} = f_d(\mathrm{z})$
  - The loss is usually MSE, L1, or cross-entropy
  - Example (MSE objective): $\min \|f_d(f_e(\mathrm{x})) - \mathrm{x}\|^2$
- Can be applied to any kind of data (not just images)

Linear autoencoder (refresher)



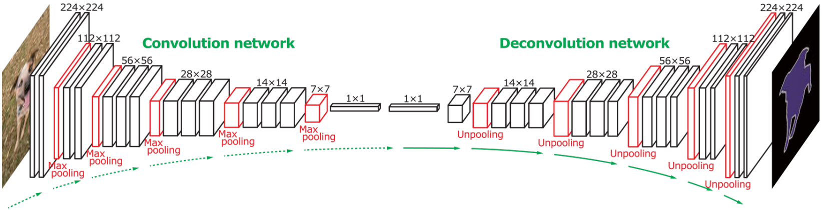
- Simplest case: f_e and f_d are linear maps
  - $z = Cx$
  - $\hat{x} = Dz$
- MSE objective: $\min \|DCx - x\|^2$
  - Can be solved efficiently using SVD
- Same thing as principal component analysis (PCA)

Non-linear autoencoder (aka *the* autoencoder)

- f_e and f_d are neural networks (learnable non-linear functions)

- Also referred to as non-linear PCA

- Typical modern architecture for images: (de)convolutional
  - Encoder: convolutions + pooling/strides
  - Decoder: transposed convolutions



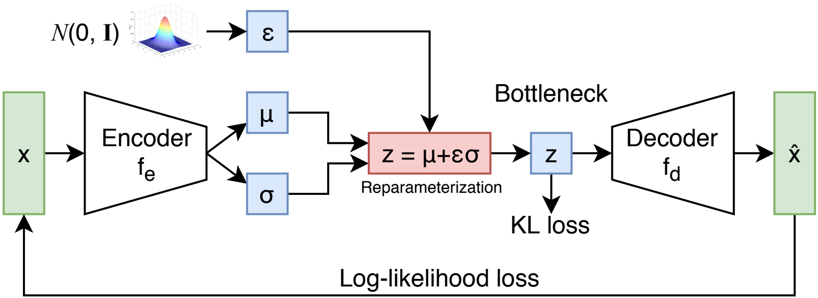
Non-linear autoencoder (continued)

- Powerful data-driven compression
- Can also be used for denoising (*denoising autoencoder*)
- **However:** no clear interpretation/structure of latent space
- Unclear how to sample or interpolate
- Visualization of the latent space is tricky
  - Many dimensions are used in practice (128+)

Variational autoencoder (motivation)

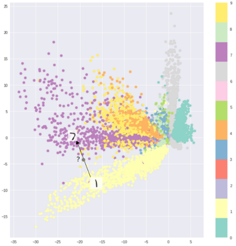
- We want to enforce a structure on the latent space, at the expense of the reconstruction quality
- One possible choice: force a prior on the latent space (e.g. Gaussian distribution)
- We can then generate by decoding a sample from the distribution
- The compactness of the latent space enables smooth interpolation

Variational autoencoder (idea)

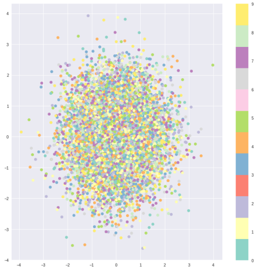


- Model latent codes as soft regions instead of points
- Sampling with reparameterization trick
- KL divergence to enforce Gaussian prior
  - Without it, the model would learn $\sigma \to 0$, reverting to a normal autoencoder

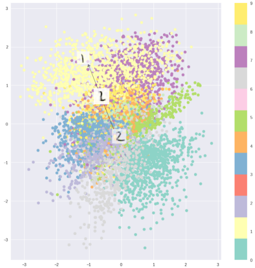
AE vs VAE (on MNIST digits)



Reconstruction loss only
(normal autoencoder)



KL loss only
(meaningless Gaussian samples)



Both
(variational autoencoder)

Figures from
`https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf`

- The latent space of a VAE approximates a Gaussian distribution, which makes sampling easy
- The lack of “holes” allows for smooth interpolation

Practical considerations

- Diagonal covariance
  - For D dimensions, $\mathcal{O}(D)$ parameters instead of $\mathcal{O}(D^2)$ for full covariance matrix

- Enforcing $\sigma > 0$ in the model architecture
  - Solution: predict $\log(\sigma^2)$ (defined across $\mathbb{R}$) and update formulas accordingly

- **Posterior collapse**
  - Model gets stuck in a bad local minimum, no learning occurs
  - Can be easily detected (KL term goes to 0)
  - Workaround: decrease strength of KL term (β-VAE)

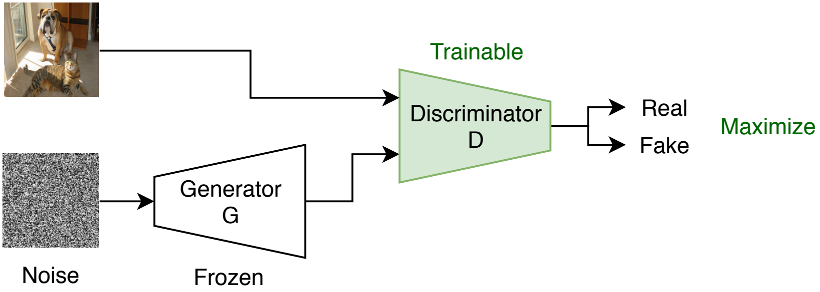
Generative Adversarial Networks

Idea

- Two networks, **generator** and **discriminator** learn to fool each other
  - They play a minimax game
- Generator: generates a sample given input noise
- Discriminator: classifies the sample as real (coming from the data distribution) or fake (coming from the generator)
- Generator and discriminator are trained in alternation by optimizing opposite objectives
  - The generator becomes increasingly better at fooling the discriminator

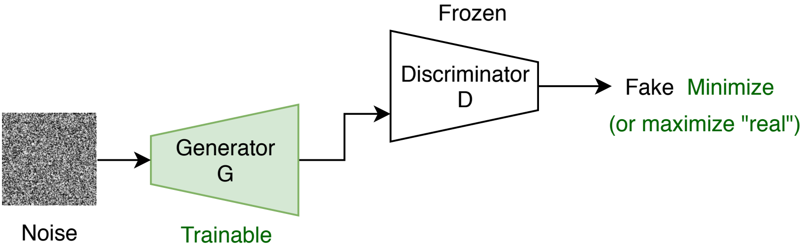
$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Training (discriminator)



$$\min_{G} \max_{D} V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

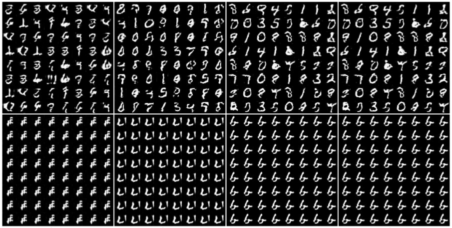
Training (generator)



$$\min_{G}\max_{D} V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Practical considerations

- Very hard to train (may not converge)!
- Mode collapse (limited diversity)
  - The generator may just learn to generate a few samples
  - Input noise is (partially or totally) ignored
  - Data distribution not entirely captured



10k steps 20k steps 50K steps 100k steps

- Need to balance generator and discriminator
  - They may learn at different speeds

Evaluation

- How do you evaluate something qualitatively without humans?
- **Inception score**
  - Quality: classify images with pretrained Inception network and compute entropy of classes (must be low)
  - Diversity: look at entropy of generated images (must be high)
- **Fréchet Inception Distance (FID)**
  1. Use pretrained Inception network to extract features from generated images
  2. Compare their distributions with those of a real dataset
- Not entirely convincing, but this is what we have

GANs vs VAEs

Quality

- (V)AEs tend to generate blurry images
  - Caused by pixel-wise factorization and local loss
  - High-frequency details are poorly correlated and hard to predict

- GANs generate sharper images
  - Discriminator learns a “perceptual” loss

Training

- GANs are very hard to train
  - Architecture and hyperparameters play a crucial role
  - Many variants have been proposed
- VAEs are somewhat easier to train
  - But not easy (especially for other domains like text)!

Applications

- GANs learn an implicit density
  - Can only generate (sample)
- VAEs learn an explicit density
  - Can sample and encode
- Some approaches combine VAEs and GANs to take the best of both of worlds
  - VAE-GAN
  - VAE to guide the style of a GAN (e.g. SPADE in figure below)