Lecture Summary for Deep Learning Advanced I - Wednesday March 26th

Objectives of the Lecture

By the end of this lecture, students should be able to:

1. Understand the fundamental concepts of probabilistic generative models
2. Explain the architecture and training process of Variational Autoencoders
3. Describe the principles behind Generative Adversarial Networks
4. Compare different types of generative models and their applications
5. Recognize how disentangled representations can be learned and utilized

Key Concepts and Definitions

- **Generative Models**: Probabilistic models of high-dimensional data (text, image, video, audio, biological sequences) used for density estimation, anomaly detection, representation learning, dimensionality reduction, and data generation
- **Prior data distribution**: The *intractable* distribution $\boldsymbol{p(x)}$ that generates the finite samples $\boldsymbol{x}$ in a dataset (images, text, video, sequences, etc). Using generative models, we aim to learn an approximation of this data distribution such that we can sample new instances from it.
    - **Divergence**: A measure of how different or *divergent* two distributions $\boldsymbol{P}$ and $\boldsymbol{Q}$ are. Popular divergences include KL Divergence, JS Divergence. If divergence is 0, the two distributions are the same. There is no upper bound on divergence
    - **Distance**: A measure of how similar or close by two objects (eg: points in $\boldsymbol{R^n}$) are. To qualify as a distance, the triangle inequality must be met. Popular choices are Euclidean distance, Manhattan distance, etc. If distance is 0, the objects are the same, otherwise they are different
- **Latent Variable Models**: Models that use unobserved variables to explain observed data patterns
- **Variational Autoencoder (VAE)**: A generative model that combines an encoder network producing a distribution over latent variables and a decoder network that reconstructs the input from the latent space
    - **Reparameterization Trick**: A technique used in VAEs to enable backpropagation through random sampling by separating deterministic and stochastic components
- **Generative Adversarial Network (GAN)**: A framework consisting of a generator and discriminator network trained in adversarial fashion
    - **Wasserstein GAN (WGAN)**: A GAN variant that uses Wasserstein distance instead of Jensen-Shannon divergence or standard KL Divergence to improve training stability
    - **Conditional GAN**: A GAN that incorporates conditional information to guide the generation process
    - **CycleGAN**: A model for unpaired image-to-image translation using cycle consistency loss
- **Disentangled Representation**: A representation where individual latent dimensions correspond to distinct, interpretable factors of variation

Main Content/Topics

Variational Autoencoders

Variational Autoencoders are directed probabilistic generative models with latent variables that serve as a compressed representation of input data. The architecture consists of two main components: an encoder network that maps input data to a distribution over latent variables, and a decoder network that reconstructs the input from sampled latent variables. Unlike traditional autoencoders, VAEs encode inputs as probability distributions (typically multivariate Gaussians) rather than deterministic points in latent space.

The training objective of VAEs is to maximize the log-likelihood of observed data, which is achieved by optimizing the Evidence Lower Bound (ELBO). This consists of two terms: a reconstruction loss measuring how well the decoder reconstructs the input, and a KL divergence term that keeps the encoder's distribution close to a prior distribution (typically a standard Gaussian). This regularization ensures a smooth, structured latent space suitable for generation.

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(N)}) = \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)})$$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[-\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) + \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\right]$$

ELBO:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right]$$

To explain how the intuition behind using the ELBO loss, we seek to compute the posterior distribution p(z | x). However this is intractable due to the complexity of the marginal likelihood which requires integrating all values of z. We approximate this true posterior p(z | x) with a more tractable distribution q(z|x) that is chosen from a family of distributions.

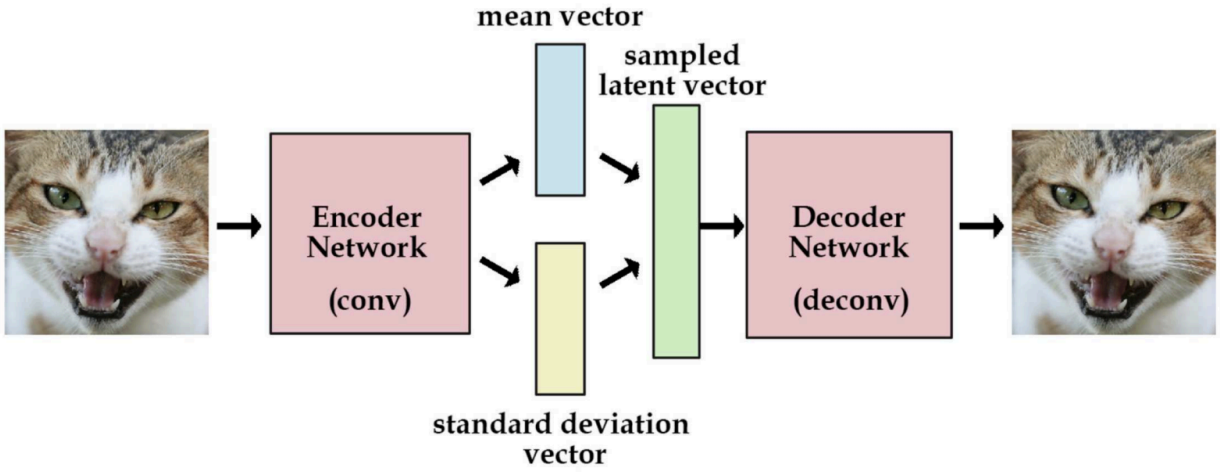
Additionally, the reparameterization trick is a key innovation allowing backpropagation through the sampling process by expressing a random sample as a deterministic function of the parameters plus a random noise term.

$$\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I})$$

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$$

Concretely, we use an auxiliary variable epsilon sampled from a normal distribution. The rest of the calculation becomes a deterministic function of the parameters which allows gradients to flow to allow for standard backpropagation.

For data generation, VAEs sample points from the prior distribution and pass them through the decoder. Conditional VAEs incorporate additional information (such as class labels or text descriptions) into both the encoder and decoder, enabling controlled generation. The regularized latent space of VAEs ensures that even when sampling from regions not well-covered by training data, the generated outputs remain reasonable rather than producing nonsensical results that can occur with standard autoencoders. We will cover these two topics more in-depthly in later sections. Below shows an example of how convolutional neural networks (CNNs) can be used as an encoder and decoder.

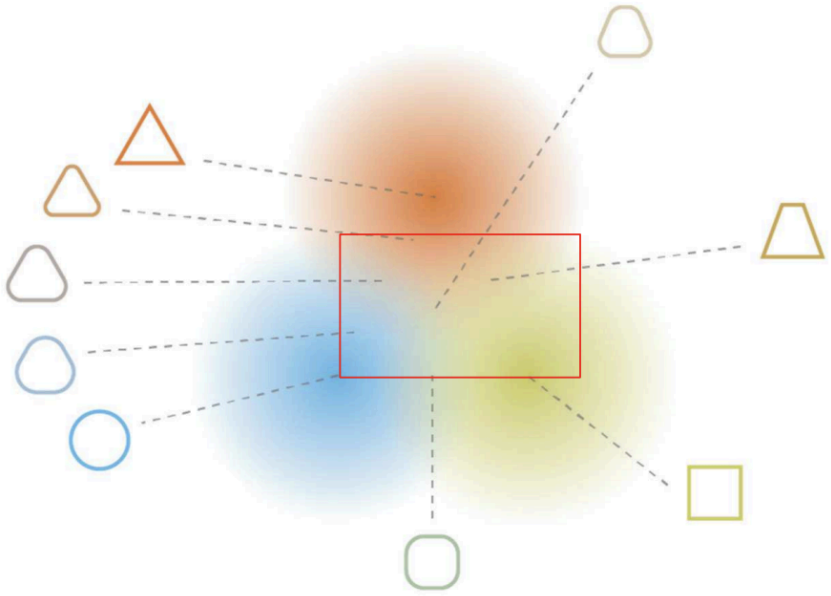


Applications of VAEs include generating MNIST digits, learning 2D manifolds, and conditional generation of images with specific attributes. Below shows the use of a VAE to generate MNIST digits, at different epochs of training.

left: 1st epoch, middle: 9th epoch, right: original

Recalling the ELBO loss, VAEs are trained with a KL divergence term that encourages their latent representations to stay close to a prior, such as a standard Gaussian prior. This means that samples

from differing input types are still pushed into the same overlapping regions during training, which is illustrated below.



You can imagine this would lead to outputs that seem to be blurry between different classes when sampling from the latent space within this overlap (red box in the image). There is a trade off of not having more class separation for a more regular latent representation with VAEs.

Conditional VAEs

Conditional VAEs try to mitigate this overlapping latent space problem by making their latent variable generation conditional on the input X. This means that each class will have its own part of the latent space while having a global prior still.

$$\log p_\theta(\mathbf{y}|\mathbf{x}) \geq -KL\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\|p_\theta(\mathbf{z}|\mathbf{x})\right) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log p_\theta(\mathbf{y}|\mathbf{x},\mathbf{z})\right]$$

and the empirical lower bound is written as:

$$\widetilde{\mathcal{L}}_{\text{CVAE}}(\mathbf{x},\mathbf{y};\theta,\phi) = -KL\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\|p_\theta(\mathbf{z}|\mathbf{x})\right) + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\mathbf{y}|\mathbf{x},\mathbf{z}^{(l)}),$$

$$\mathbf{z}^{(l)} = g_\phi(\mathbf{x},\mathbf{y},\epsilon^{(l)}),\ \epsilon^{(l)} \sim \mathcal{N}(\mathbf{0},\mathbf{I}) \text{ and } L \text{ is the number of samples.}$$

The encoder q, takes in the conditioning variable x and y the target output. It outputs parameters that define a multivariate Gaussian distribution over the latent variable z. We use a similar reparameterization trick as seen with the definition of z here which uses an epsilon sampled from normal. This lets us compute gradients w.r.t the encoder's parameters theta. We use a similar ELBO loss, again which utilizes KL divergence to penalize divergence from the prior and reward a higher expected log-likelihood of reconstructing y from a given x and z.

Generative Adversarial Network

Generative Adversarial Network is a network architecture made up of two neural networks—a generator and a discriminator. The generator transforms a simple distribution. (Gaussian or uniform noise) into a real data distribution with the goal of generating samples that will fool the discriminator. The discriminator distinguishes between data generated by the generator or from the actual data distribution. Together, the objective function is a min-max two-player game, where the discriminator tries to maximize the probability of identifying real and generated data. At the same time, the generator minimizes the probability that the discriminator will determine its output.

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

if $y = a\log(y) + b\log(1-y)$, the optimal y is

$$\implies y^* = \frac{a}{a+b}$$

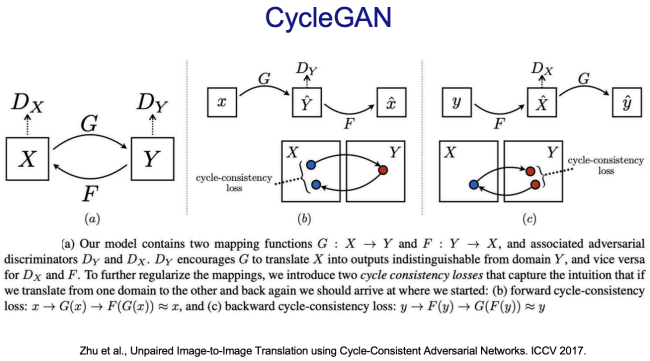
$$y = a\log(y) + b\log(1-y)$$
$$y' = \frac{a}{y} - \frac{b}{1-y}$$
$$\frac{a}{y^*} = \frac{b}{1-y^*}$$
Find optimal y^* by setting $y' = 0$.
$$\frac{1-y^*}{y^*} = \frac{b}{a}$$
$$\frac{1}{y^*} = \frac{a+b}{a}$$
$$y^* = \frac{a}{a+b}$$

Optimize $D(x) = p_r(x)\log D(x) + p_g(x)\log(1 - D(x))$, we get

$$\implies D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

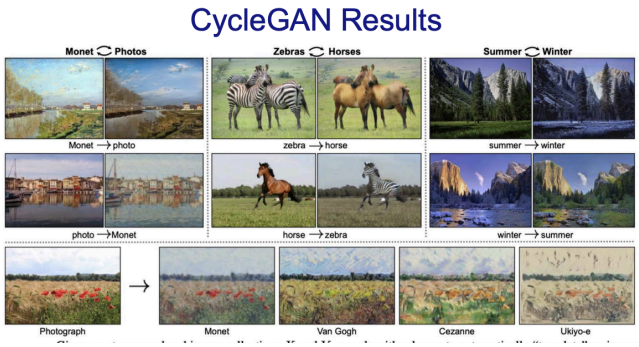
The global optimum, D will output ½ everywhere $p_g(x) = p_r(x)$, meaning that the discriminator will be unable to differentiate between the two distributions.

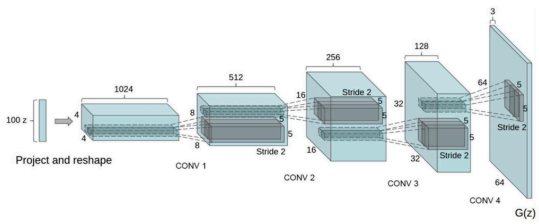
Examples of GAN networks into DCGAN (Deep Convolutional GAN), which uses convolutional layers for both the generator and discriminators to upsample noise vectors and learn spatial representations better, respectively. Conditional GAN also includes label information for the generator and discriminator, allowing it to generate data for a specific category. CycleGan leverages image-to-image translation in both directions and introduces cycle consistency losses to capture the intuition of translating from one domain to the other and back should arrive at the original image again.



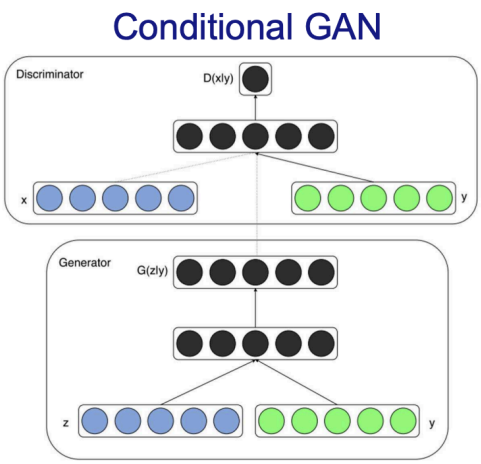
(a) Our model contains two mapping functions $G : X \to Y$ and $F : Y \to X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \to G(x) \to F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \to F(y) \to G(F(y)) \approx y$

Zhu et al., Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. ICCV 2017.





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.



https://arxiv.org/pdf/1411.1784.pdf

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 its stochastic gradient:

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

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

Goodfellow *et al.*, Generative Adversarial Nets. NIPS 2014.

Training GANs

The training process involves updating both the discriminator and the generator. For each training iteration, you will update the discriminator, D, *k* times. To do that you sample real examples, and noisy vectors, and then compute the discriminator's gradient to maximize the log-likelihood of correctly identifying real samples from faked generated samples. Then, you update the generator, G, one time per training iteration, by sampling noise vectors, and then compute the gradient to minimize the probability of the discriminator classifying the sample as fake.

GAN Limitations

In practice, the minimax training of GAN doesn't necessarily converge. For example, there can be slow learning because if the discriminator starts off perfect, the generator's gradients become very small. On the other hand, if we have a very bad discriminator, it can't provide meaningful feedback to the generator to generate better fake samples. Other problems include training instability, where training can oscillate or fail to converge, and mode collapse, where the generator only learns a subset of training data to fool the discriminator. Therefore, it is hard for it to generalize and explore other modes.

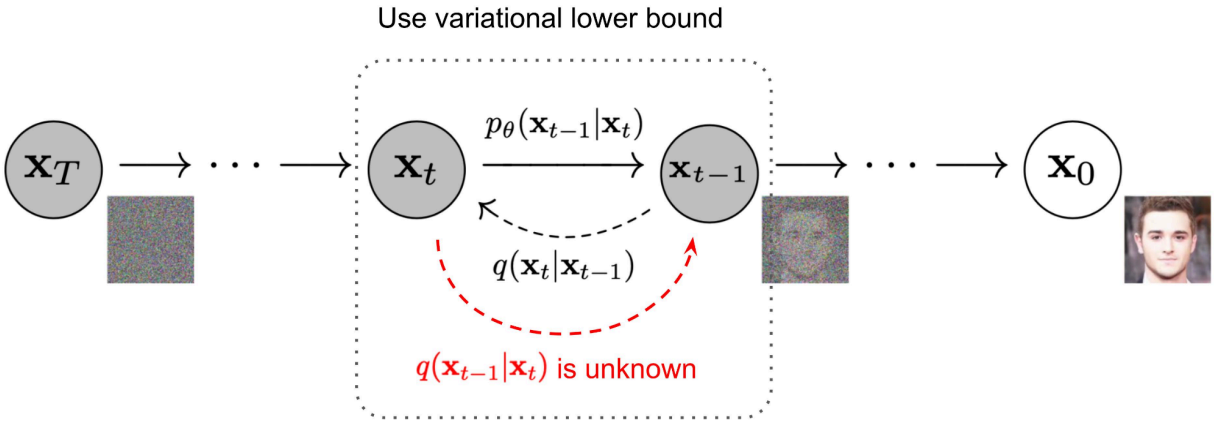
Bonus: Diffusion Models

Denoising Diffusion Probabilistic Models (DDPM) are a new generative framework that starts with random noise (assumed to be Gaussian) at time ***t=1*** and iteratively *denoises* it to generate a sample from the target data distribution at ***t=0***. Denoising means removing some noise from a sample to make it slightly more meaningful. By repeatedly denoising a sample, the sample starts to resemble other samples from the target distribution. The denoising is done by a neural network that takes in an intermediate sample $\boldsymbol{x}_t$ and the timestep $\boldsymbol{t}$.

DDPMs have two steps: a *forward process* and a *reverse process*. During the forward process, which is used during training the neural network denoiser, a valid sample is taken from the dataset and is *noised* according to a *noise schedule* that controls how much noise to add to it. The model is given such noisy samples with different amounts of noise added to them and is asked to reconstruct the

samples without the noise. Formally, the noise schedule is given a randomly sampled timestep ***t*** between 0 and 1 and returns the appropriate amount of noise to add to the sample. Since we're going from ***t=0*** to ***t=1*** during the forward process, the denoiser takes in a sample x_{t-1} and outputs x_t.

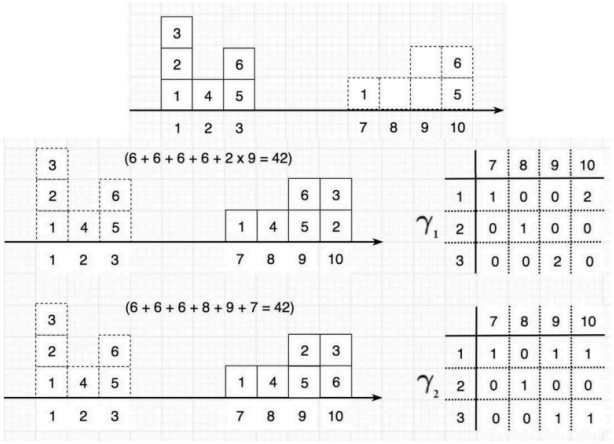
To sample from this learned distribution, we rely on the reverse process. We start from noise x_1 at ***t=1*** and pass it into the neural network denoiser for some ***T*** steps till ***t=0*** to go from x_t to x_{t-1}. With adequate training, the final sample x_0 will look like it belongs to the target distribution.



Here, the denoiser model is $p_\theta(x_{t-1} \mid x_t)$.

The theory for DDPMs builds on top of VAE: by treating the intermediate sample x_t as a latent variable, we can chain together many of them and use the reparameterization trick and the ELBO to train the neural network for these noising (forward) and denoising (reverse) tasks.

<u>Wasserstein Distance</u>



Wasserstein distance is a measure of how dissimilar two probability distributions are. It can be visualized with the metaphor of how much Earth/mass needs to be moved from one pile (one distribution) to the other. In the diagram above, the top row shows the two distributions. The next two rows show different transport plans of how to map how much mass to move from each source bin to target bin. These are different ways of moving them but the cost, which includes considering mass and distance, are the same at 42.

Wasserstein GAN

	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$	$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$
WGAN	$\nabla_w \frac{1}{m} \sum_{i=1}^{m} \left[f(x^{(i)}) - f(G(z^{(i)})) \right]$	$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f(G(z^{(i)}))$

In terms of the discriminator/critic, a Wasserstein GAN (WGAN) is replaced by a critic f that does not output probabilities but instead gives a score for how real something is using Wasserstein distance between real and generated distributions.

For a WGAN, the generator objective tries to maximize the critic's score on the fake samples so the generated samples have a high "realness score". This attempts to address challenges of vanishing gradients and mode collapse, where the output is limited in variety. These are addressed by how Wasserstein distance is continuous and smooth even for non-overlapping distributions. Classification is also replaced by scores which give continuous feedback even when the generated distribution differs wildly from the real one.

Example Use of These Model Architectures - Text2Video

Text2Video generation involves a conditional generative model that can generate videos from text descriptions. The strengths of VAEs and GANs can be combined to achieve this task. An intermediate step called gist generation is introduced. A gist is a low-resolution abstract representation that captures global information about the video. This is done via a VAE. A GAN is used for video generation with inputs of the gist, noise, and the prompt text to make the high-resolution video.

Other Uses

Other models are using a Disentangled Wasserstein Autoencoder for engineering T-cell receptors and 3D molecule generation with disentangled equivariant autoencoders.

Discussion/Comments

- VAEs provide structured latent spaces useful for controlled generation and exploration
- GAN has a lot of applications in generative tasks such as image generation and synthesis and image-to-image translation, upscaling low quality images and domain adaptation.
- The challenge of training stability remains significant for GANs, while VAEs often produce blurrier results. Recent research increasingly combines ideas from both approaches.
- Text-to-image synthesis is difficult because not only do you need to capture the static content, but you also need to capture the dynamic motion features of videos. In other words, the frames between the generated videos need to be consistent. This makes it hard to build a powerful video generator, and no publicly available dataset makes it even harder.

- Diffusion Models are expensive to train as they rely on repeated calls to a neural network denoiser. But they significantly outperform frameworks like VAEs and GANs.
- Generative models are widely being used in fields like drug discovery to generate new drug molecules, proteins, promoter/enhancer DNA sequences, and biomedicine.
- The current trend of Large Language Models (LLM) also follows this generative modelling hype wave where the AI community is interested in building models that can model *text distributions* to generate new text (eg: ChatGPT). While they rely on technology like *Transformers* and not the ones in this document, they are still categorized as generative models.

List all suggested reading here:

No suggested readings. See suggested references below.

Other Suggest references for many of the key concepts

- Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. Advances in Neural Information Processing Systems, 27.
    - Original paper introducing Generative Adversarial Networks (GANs)
- Kingma, D. P., & Welling, M. (2013, December). Auto-encoding variational bayes.
- Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan: Machine learning. stat. ML.
- Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE international conference on computer vision (pp. 2223-2232).
- Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. Advances in neural information processing systems, 33, 6840-6851.
- Dhariwal, P., & Nichol, A. (2021). Diffusion models beat gans on image synthesis. Advances in neural information processing systems, 34, 8780-8794.