Lecture 4 Deep Generative Modeling

This lecture introduces the foundations of generative modeling, a field focused on building systems that can not only find patterns in data but also generate new data instances based on those learned patterns. This is particularly relevant in the current age of generative AI.

Supervised vs. Unsupervised Learning

The lecture contrasts supervised and unsupervised learning.

- **Supervised Learning:**
  - Data consists of pairs (x,y), where x is the data and y is the label.
  - The goal is to learn a function to map x→y.
  - Examples include classification, regression, object detection, and semantic segmentation.
  - So far in the course, the focus has been on learning these mappings using deep neural networks.
- **Unsupervised Learning:**
  - Data consists only of x (data), with no labels.
  - The goal is to learn some hidden or underlying structure of the data.
  - Examples include clustering, feature or dimensionality reduction.
  - Generative modeling falls under unsupervised learning, aiming to learn the underlying structure and patterns to generate new data instances.

Generative Modeling

The fundamental principle of generative modeling is to take training samples from a distribution ($p_{data}(x)$) and learn a model ($p_{model}(x)$) that represents that distribution. The goal is for $p_{model}(x)$ to be as similar as possible to $p_{data}(x)$.

This can be used in a couple of ways:

1. **Density Estimation:** Given a set of samples along a probability distribution, the goal is to train a model that can learn the underlying probability distribution that describes the space from which the data came.
2. **Sample Generation:** Once the distribution is learned, the model can sample from this probability distribution to generate brand new data instances that reflect the learned distribution.

Why Generative Models?

Generative models have several important applications:

- **Debiasing:** By uncovering underlying features in a dataset, generative models can help identify over- or under-represented features, leading to the creation of fairer and more representative datasets.
- **Outlier Detection:** In scenarios like self-driving cars, generative models can be leveraged to detect rare events or outliers in the data distribution, helping to avoid unpredictable behavior.

- **Sample Generation:** Generative models learn probability distributions, and sampling from these distributions allows for the creation of new data instances. This is the backbone of Generative AI, with applications in generating language, images, videos, and even in biology.

Foundational Classes of Generative Models

The lecture focuses on two foundational classes of generative models:

1. **Latent Variable Models:** These models aim to learn underlying features or latent variables that govern the distribution of a dataset. Autoencoders and Variational Autoencoders (VAEs) fall into this category.
2. **Generative Adversarial Networks (GANs):** This architecture focuses more directly on sample generation through a competitive process.

Latent Variables

A latent variable is an underlying, unobserved feature that influences the data we can observe. The "Myth of the Cave" allegory by Plato is used to illustrate this concept, where the prisoners only see shadows (observed data) but not the actual objects (latent variables) casting them. The goal in generative modeling, particularly with latent variable models, is to capture these underlying features when only given observed data.

Autoencoders

Autoencoders are a simple and intuitive generative model that attempts to self-encode the input data.

- An autoencoder consists of an **encoder** and a **decoder**.
- The **encoder** maps the input data x to a lower-dimensional latent space representation z. The dimensionality of this latent space is important because a lower dimension effectively compresses the data and learns a compact feature representation.
- The **decoder** takes the latent variables z and reconstructs the original input data.
- Autoencoders are trained in an unsupervised way by minimizing the **reconstruction loss** between the original input and the reconstructed output. This loss function does not require any labels or prespecified features.
- The bottleneck created by the lower-dimensional latent space forces the encoder to capture as much information as possible to allow for good reconstruction by the decoder.

However, traditional autoencoders are deterministic, meaning the same input will always produce the same reconstructed output if the weights are fixed. This limits their ability to generate diverse new samples.

Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs) are a variation of autoencoders that introduce randomness or stochasticity into the process to allow for the generation of new data samples.

- Instead of learning deterministic latent variables, VAEs parameterize each latent variable as a probability distribution (typically a normal distribution) defined by a mean (μ) and a standard deviation (σ). The encoder learns these vectors of means and standard deviations.

- This probabilistic twist allows for sampling from these learned distributions to produce new data instances.
- The VAE trains the encoder and decoder, which effectively learn probability distributions: the encoder learns $P(z|x)$ (the distribution of latent variables given the input data), and the decoder learns $P(x|z)$ (the distribution of data given the latent variables). These networks are trained end-to-end with a single loss function.
- The VAE loss function has two main terms:
    1. **Reconstruction Loss:** Similar to the standard autoencoder, this term measures the difference between the input and the reconstructed output, aiming for a faithful reconstruction.
    2. **Regularization Term:** This new term regularizes the probability distribution of the latent variables learned by the encoder. It encourages the latent variables to follow a predefined **prior distribution**.
- A common choice for the prior distribution is a standard normal distribution (mean of zero, standard deviation of one). This encourages the latent variables to be evenly and smoothly distributed around the center of the latent space.
- The distance between the learned latent variable distribution and the prior is often computed using the **KL divergence**. Minimizing this term encourages the learned distribution to be close to the prior.
- Regularization is crucial for ensuring two desirable properties in the latent space:
    - **Continuity:** Points that are close in the latent space should correspond to data instances that are similar in content. Without regularization, points far apart in the latent space could map to similar data, and vice versa.
    - **Completeness:** Sampling from any point in the latent space should yield a meaningful data instance when decoded. Without completeness, sampling could result in nonsensical outputs.
- Training VAEs presents a challenge because backpropagation requires deterministic nodes, and the sampling step is stochastic. This is solved using the **reparameterization trick**. Instead of directly sampling z from $P(z|x)$, z is expressed as $z=\mu+\sigma\odot\epsilon$, where μ and σ are the mean and standard deviation learned by the encoder, and ϵ is a random constant sampled from the prior distribution (e.g., a standard normal distribution). This reparameterization moves the randomness away from the latent variable itself, allowing gradients to be passed through the deterministic parts of the network.
- After training, the latent variables in a VAE can often be interpretable, with individual dimensions corresponding to meaningful features in the original data space (e.g., the pose of a face). This interpretability can be used to uncover biases in datasets.

In summary, VAEs can compress data into a lower-dimensional latent space, reconstruct the original input in an unsupervised manner, are trainable end-to-end using reparameterization, learn interpretable latent variables, and can generate new data instances by sampling from the latent space and using the decoder. They combine density estimation over latent variables with sample generation through decoding.

Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are another foundational approach to generative modeling, primarily focused on generating new samples that are highly similar to the training data.

- GANs propose transforming a simple distribution, such as completely random noise, into the complex data distribution of the training samples.
- This is achieved through a competitive process between two neural networks:

1. **Generator:** Takes random noise as input and tries to produce data instances that imitate the real data.
2. **Discriminator:** Receives both real data and fake data generated by the generator and attempts to distinguish between the two (real vs. fake).

- The two networks are trained jointly with competing objectives:
  - The **discriminator** is trained to maximize the probability of correctly identifying real data as real and fake data as fake.
  - The **generator** is trained to minimize the probability that its generated data is identified as fake, essentially trying to fool the discriminator.
- This adversarial training process forces the generator to produce increasingly realistic data to deceive the discriminator, while the discriminator becomes better at detecting fakes. Ideally, the training reaches an equilibrium where the generator can perfectly replicate the true data distribution, and the discriminator can no longer distinguish real from fake.
- The core idea of GANs is learning a distribution transformation from a simple input distribution (like Gaussian noise) to the complex target data distribution.
- By smoothly interpolating in the noise space and feeding these points through the trained generator, it's possible to observe smooth transitions in the generated data space, reflecting the learned data manifold.
- Variations of GANs exist, such as **CycleGANs**, which can learn mappings between two different data manifolds without requiring paired examples. This is useful for tasks like style transfer, such as transforming images of horses to zebras. CycleGANs use a cyclic loss and have separate generator and discriminator pairs for each domain, with a dependency linking them. CycleGANs have also been applied to audio transformation by representing audio waveforms as spectrogram images.
- While GANs are powerful for sample generation, training them can be challenging and unstable in practice.

In contrast to VAEs which focus on learning underlying latent variables, GANs prioritize the generation of new samples through an adversarial process. Both VAEs and GANs are foundational to recent advances in generative modeling, including diffusion models and large language models.

Future lectures will delve into more cutting-edge generative models like diffusion models. The accompanying software lab will provide hands-on experience with VAEs for facial detection and debiasing.