Generative Probabilistic Models

- Supervised learning models $P(y|x)$: the probability of one or more target variables $y$ given input variables $x$
- Generative model $P(x)$: just model the distribution of the inputs
Example: synthetic outputs

- See tutorial
- And on-line demo
- And face generator

Figure 7.10: VAE Mnist digits generated from scratch
General Neural Network Approach

- Input random vector $z$ to neural net
  - Typically from a Gaussian bell curve distribution
- Network maps $z$ to output vector $x$
- Intuitively, the output should be like the observations $x$

$p(Z) = \text{Gaussian bell curve}$
Simple Example Mappings

<table>
<thead>
<tr>
<th>Latent distribution</th>
<th>+1 distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform over $[0,1]$</td>
<td>Uniform over $[1,2]$</td>
</tr>
<tr>
<td>Gaussian with mean 0 and standard deviation 1</td>
<td>Gaussian with mean 1 and standard deviation 1</td>
</tr>
</tbody>
</table>

![Diagram showing the mapping process](image-url)
Generate Ring from Gaussian Generative Models

Figure 2: Given a random variable \( z \) with one distribution, we can create another random variable \( X = g(z) \) with a completely different distribution.

Left: samples from a gaussian distribution. Right: those same samples mapped through the function \( g(z) = \frac{z}{10} + \frac{z}{||z||} \) to form a ring. This is the strategy that VAEs use to create arbitrary distributions: the deterministic function \( g \) is learned from data.

To solve Equation 1, there are two problems that VAEs must deal with: how to define the latent variables \( z \) (i.e., decide what information they represent), and how to deal with the integral over \( z \). VAEs give a definite answer to both.

First, how do we choose the latent variables \( z \) such that we capture latent information? Returning to our digits example, the 'latent' decisions that the model needs to make before it begins painting the digit are actually rather complicated. It needs to choose not just the digit, but the angle that the digit is drawn, the stroke width, and also abstract stylistic properties. Worse, these properties may be correlated: a more angled digit may result if one writes faster, which also might tend to result in a thinner stroke. Ideally, we want to avoid deciding by hand what information each dimension of \( z \) encodes (although we may want to specify it by hand for some dimensions). We also want to avoid explicitly describing the dependencies—i.e., the latent structure—between the dimensions of \( z \).

VAEs take an unusual approach to dealing with this problem: they assume that there is no simple interpretation of the dimensions of \( z \), and instead assert that samples of \( z \) can be drawn from a simple distribution, i.e., \( N(0, I) \), where \( I \) is the identity matrix. How...
How to Train Deep Generative Models?

• What training objective?
• 2 main ideas
  • Variational Auto-Encoder
  • Generative Adversarial Network
Variational Auto-Encoder (VAE)
VAE Objective Approximates Log-Likelihood

- Loss function for input $x = -\ln(P(x))$
  - The negative log-likelihood is the standard loss for generative models
- In the decoder architecture
  $P(x) = \int_z 1(f(z) = x) \ p(z) \ dz$
  - $1(f(z) = x)$ returns 1 if the decoder maps random $z$ to $x$, 0 o.w.
- Integral is intractable
- VAE architecture is designed so that *training loss* approximates integral negative log-likelihood

Generative Models
VAE Architecture

- Combines Auto-Encoding with Generative Modeling
- High-level idea
  1. **Training:** Design a non-deterministic version of an auto-encoder
     - Can produce multiple outputs $x$-out for the same input $x$-in
  2. **Testing:** Generate random input $z$, produce (non-deterministic) output

Generative Models
Example: Non-deterministic output

![Original Image](image1.png)
![Reconstruction](image2.png)

original reconstruction

Generative Models
VAE Training Architecture

Regularizer for Mu, Sigma

Variance: noise around reconstruction

reconstruction/encoding
VAE Test Architecture

Generative Models

Random input
Example Application

Embeddings for Hockey Players
Contextualized Player Embedding

• The set-up: we see a hockey game sequence.
• Using a VAE, compute a contextualized player embedding as a function of the game sequence.
• The VAE generates a distribution $P(\text{player}_t = i \mid \text{sequence up to time } t)$
Combining LSTM with VAE

- Sequence is processed using LSTM
- VAE generates probability distribution over players

Sequence Analysis

<table>
<thead>
<tr>
<th>Time 1</th>
<th>Time 2</th>
<th>Time 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM state</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hockey Event</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Decoder Network

player ID

\( p(Z) = \text{Gaussian bell curve} \)
Accuracy in Player Recognition

Given a match sequence, can you predict which player is likely to have the puck now?

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Log-likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM</td>
<td>12.41%</td>
<td>-3.131</td>
</tr>
<tr>
<td>LSTM + VAE</td>
<td>48%</td>
<td>-2.228</td>
</tr>
</tbody>
</table>

• The player embeddings help with applications like predicting
  • Whether a shot will lead to a goal
  • The final game outcome
The Evidence Lower Bound

The Mathematics Behind VAEs
Background

- Monte Carlo Sampling
- The KL Divergence
Sampling Example

- Area of unit circle = \( \pi \)
- Estimate \( \pi \) by randomly sampling points in square, checking if they lie within circle. [Demo]
Sample Complexity

- Basic Problem: Sampling can require many data points for accurate estimate.
Kulback-Leibler Divergence

- **KLD:** \( D(q \mid p) = \sum_j q(x_j) \log(q(x_j)/p(x_j)) \)
  \[ = \mathbb{E}_q[\log(q(x)/p(x))] \]

- **Entropy:** \( H(q) = - \sum_j q(x_j) \log(q(x_j)) \)
  \[ = - \mathbb{E}_q[\log(q(x))] \]

- **Cross-Entropy:** \( H(q,p) = - \sum_j q(x_j) \log(p(x_j)) \)
  \[ = - \mathbb{E}_q[\log(p(x))] \]

- **Exercise:** Prove that \( D(p \mid q) = H(q,p) - H(q) \)

- **Exercise:** Recall the “cross-entropy” of Assignment
  \[ \sum_{j=1}^{N} y_j \ln(p_j^+) + (1 - y_j) \ln(1 - p_j^+) \]
  What is the relationship between this formula and \( H(q,p) \)?

Generative Models
Likelihood: The True Objective

- Recall that the standard objective for generative modelling is the data likelihood

$$\Pi_j P(x_j; f) = \int_z 1(f(z) = x_j) p(z) \, dz$$
for observed data points $x_1, \ldots, x_j$

- We could estimate this integral for each data point using Monte Carlo estimation.
  1. Sample $k$ $z$ points,
  2. apply $f$
  3. Calculate the mean estimate: average $f(z_1), \ldots, f(z_k)$
Sampling With Functions

Given a random variable $z$ with one distribution, we can create another random variable $X = g(z)$ with a completely different distribution.

Left: samples from a Gaussian distribution. Right: those same samples mapped through the function $g(z) = z/10 + z/||z||$ to form a ring. This is the strategy that VAEs use to create arbitrary distributions: the deterministic function $g$ is learned from data.

To solve Equation 1, there are two problems that VAEs must deal with: how to define the latent variables $z$ (i.e., decide what information they represent), and how to deal with the integral over $z$. VAEs give a definite answer to both.

First, how do we choose the latent variables $z$ such that we capture latent information? Returning to our digits example, the 'latent' decisions that the model needs to make before it begins painting the digit are actually rather complicated. It needs to choose not just the digit, but the angle that the digit is drawn, the stroke width, and also abstract stylistic properties. Worse, these properties may be correlated: a more angled digit may result if one writes faster, which also might tend to result in a thinner stroke. Ideally, we want to avoid deciding by hand what information each dimension of $z$ encodes (although we may want to specify it by hand for some dimensions [4]). We also want to avoid explicitly describing the dependencies—i.e., the latent structure—between the dimensions of $z$. VAEs take an unusual approach to dealing with this problem: they assume that there is no simple interpretation of the dimensions of $z$, and instead assert that samples of $z$ can be drawn from a simple distribution, i.e., $N(0, I)$, where $I$ is the identity matrix. How

• If $f$ is invertible, then for each output $x$, there is a unique $f^{-1}(z)$ that generates it.
• Even if $f$ is not invertible, typically for each output $x$, few samples will generate $x$.
Invertible Function Case

- We can replace
  \[ P(x; f) = \int_z 1(f(z) = x) \, p(z) \, dz = \int_z p(f^{-1}(x)) \, dz \]
- I.e., just consider probability of the random input that generates the observed output \( x \)
- How to compute \( f^{-1}(x) \)?
  - Learn another neural network for \( f^{-1} \)!
Probabilistic Version

- Even if the generating function $f$ is not invertible, few samples $z$ have a significant likelihood of generating an observed data point $x$.

- We can measure this with the posterior probability $p(z | x)$: given that $x$ was observed, what is the probability that $z$ generated $x$.

- The posterior probability is the inverse of a conditional probability model $p(x | z)$.

- $P(x) = \int_z p(x | z) \, p(z) \, dz \approx \int_z p(x | z) \, p(z | x) \, d(z)$
Approximate Posterior

- Given the approximation
  \[ P(x) \approx \int_z p(x | z) p(z | x) \, d(z) \]
- How can we compute \( p(z | x) \)?
- Learn a neural network to output (the parameters of) \( p(z | x) \)!
- Hard to get this exactly so we try to learn an \textit{approximate posterior} \( q(z | x) \approx p(z | x) \)

Generative Models
Monte Carlo Estimation of Data Likelihood

- \[ P(x) \approx \int_z p(x | z) p(z | x) \, d(z) \]
  \[ \approx \mathbb{E}_{z \sim q(z | x)} p(x | z) \]

- For each data point \( x \)
  1. Apply the encoder to find an approximate posterior distribution \( q(z | x_j) \)
  2. Sample \( z \) from \( q(z | x_j) \)
  3. Apply the decoder to compute \( p(x | z) \)
  4. Average the estimates from the \( z \) samples to get an estimated \( P(x) \)

- What is the relationship between the encoder-decoder approximations and the true data likelihood \( P(x) \)?
The Evidence Lower Bound

- The following relationship holds for any approximate posterior $q(z | x)$, prior $p(z)$, conditional probability $p(x | z)$, unconditional probability $p(x)$

$$\log(p(x)) \geq E_{z \sim q(z | x)} [\log(p(x | z))] - \text{KLD}(q(z | x) \parallel p(z))$$

• The bigger the RHS, the better our approximation.
• So we try to maximize it:

$$\arg\max p, q \sum_j E_{z \sim q(z | x_j)} [\log(p(x_j | z))] - \text{KLD}(q(z | x_j) \parallel p(z))$$
Implementing the ELBO objective

- The ELBO is a very general result, known for decades.
- The VAE is a neural architecture proposed by Kingma and Welling 2014 that fills in q and p as follows.
  - \( q(z|x) = \text{Normal}(\mu(x), \sigma^2(x)) \)
    - \( \mu \) and \( \sigma \) are vectors of the same dimension as \( z \)
    - They are computed by a neural network (encoder)
  - \( p(x|z) = \text{Normal}(f(z), \sigma^2*I) \)
    - \( f(z) \) is a vector of the same dimension as \( x \)
    - \( \sigma^2 \) is a hyper-parameter
    - \( f(z) \) is computed by a neural network (decoder)

- [Visualization](#)
KLD for Gaussians

- The ELBO requires us to evaluate $\text{KLD}(q(z|\mathbf{x}_j) \mid | p(z))$
- For two multi-variate Gaussians $N_Q(m_Q, \Sigma_Q)$ and $N_p(m_p, \Sigma_p)$, the KLD divergence is given by the formula:
  \[
  D(N_Q \mid \mid N_p) = \frac{1}{2} \left\{ \text{tr} (\Sigma_p^{-1} \Sigma_q) + \ln | \Sigma_p | - \ln | \Sigma_Q | + (m_p - m_Q)^T \Sigma_p^{-1} (m_p - m_Q)^T \right\} - d/2
  \]
  where:
  - $d$ is the latent embedding dimension
  - $\text{tr}$ is the matrix trace (sum of diagonal elements)
  - In a VAE, we have $N_p = (0, I)$ and $\Sigma_q$ is diagonal with entries $\sigma_i^2$ for $i = 1, \ldots, d$
  - Show that $D(N_Q \mid \mid N_p) = \frac{1}{2} \left\{ \sum_i \sigma_i^2 - \ln(\sigma_i^2) + m_i^2 \right\} - d/2$ where $m = m_Q$

Generative Models
Generative Adversarial Models

GANs
GAN Architecture

- Recall the basic generative NN architecture
- How to train?
- VAE: approximate log-likelihood of the observations
- GAN: train the generator so that synthetic generated examples cannot be distinguished from actual observations

\[ p(Z) = \text{Gaussian bell curve} \]
Intuition: Taste Test

- Coke is the real thing
- Pepsi wants to imitate coke
- How does Pepsi know they have succeeded?
- When a blind taste test cannot tell the difference!
Chatbots and the Turing Test

Alan Turing (1950) “Computing Machines and Intelligence”

• How can we know if a machine is intelligent?

➢ Test if it can fool a human!

• Loebner Prize

• Chatbots
Distinguishing Uniform and Gaussians

- Fake data: uniform [-8,8]
- Real data: Gaussian with mean 5, standard deviation 1

![Graph showing the probability density functions of a uniform distribution and a Gaussian distribution. The uniform distribution is represented by a constant value across the range [-8,8], while the Gaussian distribution is represented by a bell-shaped curve centered around the mean of 5 with a standard deviation of 1. The graph illustrates the difference between the two distributions, with the uniform distribution being flat and the Gaussian distribution being more concentrated around the mean.](image-url)
The Tester

- How can we test whether a network generates realistic outputs?
- Train another classifier network to distinguish synthetic from real examples!
- Discriminator $D$ outputs $o_D(x) = P(x \text{ is real example})$
- Why can we not just backpropagate to train the generator?

**Figure 7.11: The structure of a generative adversarial network**

Generative Models
The Loss Functions

- Discriminator objective function given real examples $R$ and fake examples $F$
  \[ V(D, G) = \text{mean}_r \ln(o_D(r)) + \text{mean}_f \ln(1-o_D(f)) \]
- The discriminator wants to maximize $V$, the generator to minimize $V$
- Minmax problem: GANs are tricky to train

<table>
<thead>
<tr>
<th>Discriminator</th>
<th>Generator</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>x</strong></td>
<td>Output</td>
</tr>
<tr>
<td>Real $r$</td>
<td>$o_D(r)$</td>
</tr>
<tr>
<td>Fake $f$</td>
<td>$o_D(f)$</td>
</tr>
</tbody>
</table>
Other Loss Functions

- Other loss functions can be considered to help with training, e.g. in the book:
  - The generator wants $o_D(f)$ to be big
    - So maximize $\ln o_D(f)$ or minimize $-\ln o_D(f)$
  - However it is more usual to ensure the zero-sum condition:
    
    \[
    \text{loss(discriminator)} = -\text{loss(generator)}
    \]

<table>
<thead>
<tr>
<th>Book</th>
<th>Discriminator</th>
<th>Generator</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>Output</td>
<td>Objective</td>
</tr>
<tr>
<td>Real $r$</td>
<td>$o_D(r)$</td>
<td>$\ln(o_D(r))$</td>
</tr>
<tr>
<td>Fake $f$</td>
<td>$o_D(f)$</td>
<td>$\ln(1-o_D(f))$</td>
</tr>
</tbody>
</table>
Examples and Demos

- Style GAN
- Deep Fakes
- GANlab
- nvidia-ai-playground
Conclusion

- Generative models generate outputs without an input
- Basic idea: map a random input to an output
- Without a target output, what is the training objective?
- Variational AE: approximate the log-likelihood of the observed data $\mathbf{x}$
  - Also produces an embedding of the input $\mathbf{x}$
- Generative Adversarial Network: generate synthetic outputs that cannot be distinguished from actual observations by an adversarial classifier network