Topics:
• Generative Models
• Pixel CNN
• Variational Autoencoders

CS 4644-DL / 7643-A
ZSOLT KIRA
• **Projects!**
  • Due May 1\textsuperscript{rd} (May 3\textsuperscript{th} with grace period)
  • Cannot extend due to grade deadlines!

• **Outline of rest of course:**

**W14: Apr 14** Variational Autoencoders
**W15: Apr 19** Diffusion Models
**W15: Apr 21** Emerging trends, wrap-up.

*•Tutorial on Variational Autoencoders*
Supervised Learning

- Train Input: \( \{X, Y\} \)
- Learning output: \( f : X \to Y, P(y|x) \)
- e.g. classification

Unsupervised Learning

- Input: \( \{X\} \)
- Learning output: \( P(x) \)
- Example: Clustering, density estimation, etc.

Spectrum of Low-Labeled Learning
Traditional unsupervised learning methods:

- **Modeling $P(x)$**
  - Deep Generative Models

- **Comparing/Grouping**
  - Metric learning & clustering

- **Representation Learning**
  - Almost all deep learning!

Similar in deep learning, but from neural network/learning perspective

**What to Learn?**
Discriminative vs. Generative Models

- Discriminative models model $P(y|x)$
  - Example: Model this via neural network, SVM, etc.

- Generative models model $P(x)$
**Discriminative vs. Generative Models**

- **Discriminative models** model $P(y|x)$
  - Example: Model this via neural network, SVM, etc.

- **Generative models** model $P(x)$
  - We can parameterize our model as $P(x, \theta)$ and use maximum likelihood to optimize the parameters given an unlabeled dataset:

  $$
  \theta^* = \arg \max_{\theta} \prod_{i=1}^{m} p_{\text{model}}(x^{(i)}; \theta)
  $$

  $$
  = \arg \max_{\theta} \log \prod_{i=1}^{m} p_{\text{model}}(x^{(i)}; \theta)
  $$

  $$
  = \arg \max_{\theta} \sum_{i=1}^{m} \log p_{\text{model}}(x^{(i)}; \theta)
  $$

- They are called generative because they can often generate *samples*
  - Example: Multivariate Gaussian with estimated parameters $\mu, \sigma$
Generative Models

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks
PixelRNN & PixelCNN
Generative Models

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks
We can use chain rule to decompose the joint distribution

- Factorizes joint distribution into a product of conditional distributions
  - Similar to Bayesian Network (factorizing a joint distribution)
  - Similar to language models!

- Requires some *ordering* of variables (edges in a probabilistic graphical model)
- We can estimate this conditional distribution as a neural network

\[
p(x) = \prod_{i=1}^{n^2} p(x_i|x_1, \ldots, x_{i-1})
\]

*Oord et al., Pixel Recurrent Neural Networks*
\[ p(s) = p(w_1, w_2, \ldots, w_n) \]

\[ = p(w_1) p(w_2 \mid w_1) p(w_3 \mid w_1, w_2) \cdots p(w_n \mid w_{n-1}, \ldots, w_1) \]

\[ = \prod_{i \text{ next word}} p(w_i \mid w_{i-1}, \ldots, w_1) \text{ history} \]
Language modeling involves estimating a probability distribution over sequences of words.

\[ p(s) = p(w_1, w_2, \ldots, w_n) = \prod_{i} p(w_i \mid w_{i-1}, \ldots, w_1) \]

RNNs are a family of neural architectures for modeling sequences.
\[ p(x) = \prod_{i=1}^{n^2} p(x_i|x_1, \ldots, x_{i-1}) \]
\[ p(x) = p(x_1) \prod_{i=2}^{n^2} p(x_i|x_1, \ldots, x_{i-1}) \]

Factorized Models for Images

Oord et al., Pixel Recurrent Neural Networks
Factorized Models for Images

\[ p(x) = p(x_1)p(x_2|x_1)p(x_3|x_1) \prod_{i=1}^{n^2} p(x_i|x_1, \ldots, x_{i-1}) \]

- **Training:**
  - We can train similar to language models: Teacher/student forcing
  - Maximum likelihood approach

- **Downsides:**
  - Slow sequential generation process
  - Only considers few context pixels

---

Oord et al., Pixel Recurrent Neural Networks
Idea: Represent conditional distribution as a convolution layer!

Considers larger context (receptive field)

Practically can be implemented by applying a mask, zeroing out “future” pixels

Faster training but still slow generation
  - Limited to smaller images

*Oord et al., Conditional Image Generation with PixelCNN Decoders*
Example Results: Image Completion (PixelRNN)

Oord et al., Conditional Image Generation with PixelCNN Decoders
Variational Autoencoders (VAEs)
Generative Models

Goodfellow, NeurIPS 2016 Tutorial: Generative Adversarial Networks
Autoencoders

Encoder

Minimize the difference (with MSE)

Decoder

Low dimensional embedding

Linear layers with reduced dimension or Conv-2d layers with stride

Linear layers with increasing dimension or Conv-2d layers with bilinear upsampling

Autoencoders
What is this?
Hidden/Latent variables
Factors of variation that produce an image:
(digit, orientation, scale, etc.)

\[ P(X) = \int P(X|Z; \theta)P(Z)dz \]

- We cannot maximize this likelihood due to the integral
- Instead we maximize a variational lower bound (VLB) that we can compute

Kingma & Welling, Auto-Encoding Variational Bayes
We can combine the probabilistic view, sampling, autoencoders, and approximate optimization.

Sample $Z$ from simpler distribution.
We would like to sample from $p(x)$ using a neural network.

**Idea:**
- Sample from a simple distribution (Gaussian)
- Transform the sample to $p(x)$
Input can be a vector with (independent) Gaussian random numbers

We can use a CNN to generate images!

\[ N(\mu, \sigma) \quad \text{Neural Network} \quad p(x) \]
- We can combine the probabilistic view, sampling, autoencoders, and approximate optimization.

- Assume $Z$ comes from a simpler distribution (Normal).

- We can also output parameters of a probability distribution!
  
  - **Example**: $\mu, \sigma$ of Gaussian distribution
  - For multi-dimensional version output diagonal covariance

- How can we maximize

$$P(X) = \int P(X|Z; \theta)P(Z)dZ$$
We can combine the probabilistic view, sampling, autoencoders, and approximate optimization.

Given an image, estimate $Z$

Again, output parameters of a distribution.

Variational Autoencoder: Encoder
We can tie the encoder and decoder together into a probabilistic autoencoder:

- Given data \((X)\), estimate \(\mu_z, \sigma_z\) and sample from \(N(\mu_z, \sigma_z)\)
- Given \(Z\), estimate \(\mu_x, \sigma_x\) and sample from \(N(\mu_x, \sigma_x)\)
How can we optimize the parameters of the two networks?

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = E_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$
\[
\log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad \text{(Bayes' Rule)} \\
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad \text{(Multiply by constant)} \\
= \mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad \text{(Logarithms)}
\]
Aside: KL Divergence (distance measure for distributions), always $\geq 0$

$$KL(p || q) = H_c(p, q) - H(p) = \sum p(x) \log q(x) - \sum p(x) \log p(x)$$

Definition of Expectation

$$E[f] = E_{x \sim q}[f(x)] = \sum_{x \in \Omega} q(x) f(x)$$

$$KL(a || b) = E[\log a(x)] - E[\log b(x)] = E[\log \frac{a(x)}{b(x)}]$$
\[
\log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)
\]

\[
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)}) q_\phi(z | x^{(i)})} \right] \quad (\text{Bayes’ Rule})
\]

\[
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] q_\phi(z | x^{(i)}) \quad (\text{Multiply by constant})
\]

\[
= \mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad (\text{Logarithms})
\]

\[
= \mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z)) + D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z | x^{(i)}))
\]

The expectation wrt. z (using encoder network) let us write nice KL terms
\[
\log p_{\theta}(x^{(i)}) = \mathbf{E}_z \sim q_{\phi}(z | x^{(i)}) \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)
\]

\[
= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes’ Rule})
\]

\[
= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \cdot \frac{q_{\phi}(z | x^{(i)})}{q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant})
\]

\[
= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms})
\]

\[
= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) \parallel p_{\theta}(z)) + D_{KL}(q_{\phi}(z | x^{(i)}) \parallel p_{\theta}(z | x^{(i)}))
\]

---

Decoder network gives \( p_{\theta}(x|z) \), can compute estimate of this term through sampling. (Sampling differentiable through reparam. trick, see paper.)

This KL term (between Gaussians for encoder and \( z \) prior) has nice closed-form solution!

\( p_{\theta}(z|x) \) intractable (saw earlier), can’t compute this KL term :( But we know KL divergence always \( \geq 0 \).
\[
\log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\phi(z | x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right] (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} q_\phi(z | x^{(i)}) \right] \quad \text{(Bayes’ Rule)} \\
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)}{p_\theta(z | x^{(i)})} \right] - \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] + \mathbb{E}_z \left[ \log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad \text{(Multiply by constant)} \\
= \mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - \mathbb{KL}(q_\phi(z | x^{(i)}) \| p_\theta(z)) + \mathbb{KL}(q_\phi(z | x^{(i)}) \| p_\theta(z)) \quad \text{(Logarithms)} \\
\mathcal{L}(x^{(i)}, \theta, \phi) \\
\geq 0 \\
\theta^*, \phi^* = \arg \max_{\theta, \phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi) \\
\text{Training: Maximize lower bound}
\]

Variational lower bound ("ELBO")
Putting it all together: maximizing the likelihood lower bound

$$\mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) \| p_\theta(z))$$

Make approximate posterior distribution close to prior

From CS231n, Fei-Fei Li, Justin Johnson, Serena Yeung
Putting it all together: maximizing the likelihood lower bound

$$E_z \left[ \log p_\theta(x^{(i)} \mid z) \right] - D_{KL}(q_\phi(z \mid x^{(i)}) \mid \mid p_\theta(z))$$

Sample from $Q(Z \mid X) \sim N(\mu_z, \sigma_z)$

From CS231n, Fei-Fei Li, Justin Johnson, Serena Yeung
Putting it all together: maximizing the likelihood lower bound

$$
E_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) \| p_\theta(z))
$$

Maximize likelihood of original input being reconstructed

Sample from $P(X|Z; \theta) \sim N(\mu_x, \sigma_x)$

From CS231n, Fei-Fei Li, Justin Johnson, Serena Yeung
Problem with respect to the VLB: updating $\phi$

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

$Z \sim Q(Z|X; \phi):$ need to differentiate through the sampling process w.r.t $\phi$ (encoder is probabilistic)

From: Tutorial on Variational Autoencoders
https://arxiv.org/abs/1606.05908

From: http://gokererdogan.github.io/2016/07/01/reparameterization-trick/
Solution: make the randomness independent of encoder output, making the encoder deterministic

Gaussian distribution example:
- Previously: encoder output = random variable $z \sim N(\mu, \sigma)$
- Now encoder output = distribution parameter $[\mu, \sigma]$
- $z = \mu + \epsilon \cdot \sigma, \epsilon \sim N(0,1)$

From: Tutorial on Variational Autoencoders
https://arxiv.org/abs/1606.05908

From: http://gokererdogan.github.io/2016/07/01/reparameterization-trick/
- Variational Autoencoders (VAEs) provide a principled way to perform approximate maximum likelihood optimization
  - Requires some assumptions (e.g. Gaussian distributions)

- Samples are often not as competitive as other methods (GANs, diffusion)

- Latent features (learned in an unsupervised way!) often good for downstream tasks:
  - Example: World models for reinforcement learning (Ha et al., 2018)

*Ha & Schmidhuber, World Models, 2018*
Generative Adversarial Networks (GANs)
- Input can be a vector with (independent) Gaussian random numbers
- We can use a CNN to generate images!
Goal: We would like to generate realistic images. How can we drive the network to learn how to do this?

Idea: Have another network try to distinguish a real image from a generated (fake) image

Why? Signal can be used to determine how well it’s doing at generation
Generative Adversarial Networks (GANs)

Vector of Random Numbers

Generator

Discriminator

Question: What loss functions can we use (for each network)?

- **Generator**: Update weights to improve realism of generated images
- **Discriminator**: Update weights to better discriminate

Mini-batch of real & fake data

Cross-entropy (Real or Fake?) We know the answer (self-supervised)
Generative Adversarial Networks (GANs)

Vector of Random Numbers

Generator Loss

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

Discriminator Loss

\[ \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(x^{(i)}\right) + \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right]. \]

Mini-batch of real & fake data

Cross-entropy (Real or Fake?)
We know the answer (self-supervised)
Example Generated Images - BigGAN

Brock et al., Large Scale GAN Training for High Fidelity Natural Image Synthesis
Video Generation

https://www.youtube.com/watch?v=PCBTZh41Ris
Several ways to learn *generative* models via deep learning

- **PixelRNN/CNN:**
  - Simple tractable densities we can model via a NN and optimize
  - Slow generation – limited scaling to large complex images

- **Generative Adversarial Networks (GANs):**
  - Pro: Amazing results across many image modalities
  - Con: Unstable/difficult training process, computationally heavy for good results
  - Con: Limited success for discrete distributions (language)
  - Con: Hard to evaluate (implicit model)

- **Variational Autoencoders:**
  - Pro: Principled mathematical formulation
  - Pro: Results in disentangled latent representations
  - Con: Approximation inference, results in somewhat lower quality reconstructions

---

*Ha & Schmidhuber, World Models, 2018*