Lecture 15: Unsupervised Deep Learning (III)
Applications of Generative Models; Normalizing Flows
Outline

1. **Some applications of convolutional autoencoders and GAN**
   a. Image-to-Image Translation with Conditional Adversarial Nets
   b. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
   c. PuppetGAN: Cross-Domain Image Manipulation by Demonstration

2. **Normalizing Flows**
   a. Change of variable formula
   b. Planar and radial flows
   c. Real NVP
   d. GLOW
   e. FFJORD
   f. Likelihood vs probability
Recap: GANs

choose \( \theta \) such that the observed dataset could have been generated by \( \text{model}(\theta) \)
Recap: GANs

Choose \( \theta \) such that the observed dataset could have been generated by \( \text{model}(\theta) \)

We learn to compute
\[
\text{model}(x, \theta) = P(x \mid \theta)
\]

We learn to sample
\[
x' \sim \text{model}(\theta)
\]

Maximum Likelihood

Direct GAN

Explicit density
- Tractable density
  - Fully visible belief nets
  - NADE
  - MADE
  - PixelRNN
- Approximate density
  - Variational
  - Markov Chain

Implicit density
- Markov Chain
- GSN

Variational autoencoder
Boltzmann machine

Change of variables
models (nonlinear ICA)
Recap: GANs

choose \( \theta \) such that the observed dataset could have been generated by \( \text{model} (\theta) \)

we learn to compute
\[ \text{model}(x, \theta) = P(x | \theta) \]

we learn to sample
\[ x' \sim \text{model}(\theta) \]

choose \( \theta \) such that the observed dataset could have been generated by \( \text{model}(\theta) \)

train G “to fool” D,
train D “to catch” G
Recap: GANs
Example Application 1: Improving outputs of *supervised* image-to-image models

Example Application 1:
Improving outputs of \textbf{supervised} image-to-image models

Q: This is a regression model: why not just use a simple supervised loss? (L1, L2)
Example Application 1:
Improving outputs of **supervised** image-to-image models

Q: This is a regression model: why not just use a simple supervised loss? (L1, L2)
A: These losses assume that multiple outputs are independent.

\[ Y_i = f(X, \theta) + e_i \quad e_i \sim N(0, I) \quad \Rightarrow \quad \text{L2 loss} \]

Example Application 1:
Improving outputs of **supervised** image-to-image models

Example Application 1:
Improving outputs of **supervised** image-to-image models

Example Application 1: Improving outputs of *supervised* image-to-image models

Figure 2: Training a conditional GAN to map edges→photo. The discriminator, $D$, learns to classify between fake (synthesized by the generator) and real \{edge, photo\} tuples. The generator, $G$, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.
Example Application 1:
Improving outputs of **supervised** image-to-image models

**Takeaway:**

if the output domain has some structure (i.e. an image) adversarial losses force the model to follow that structure
Example Application 1: Improving outputs of *supervised* image-to-image models

**Takeaway:**

if the output domain has some structure (i.e. an image) adversarial losses force the model to follow that structure

plain regression model is encouraged to “interpolate” outputs if uncertain

adversarial losses explicitly penalise outputs that look “too different from outputs in the training set”
Example Application 2: Enabling **unsupervised** image-to-image training

"Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" [Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros]
Example Application 2:
Enabling **unsupervised** image-to-image training

**Figure 2:** Paired training data (left) consists of training examples \( \{x_i, y_i\}_{i=1}^N \), where the correspondence between \( x_i \) and \( y_i \) exists [22]. We instead consider **unpaired** training data (right), consisting of a source set \( \{x_i\}_{i=1}^N (x_i \in X) \) and a target set \( \{y_j\}_{j=1} (y_j \in Y) \), with no information provided as to which \( x_i \) matches which \( y_j \).

Example Application 2: Enabling **unsupervised** image-to-image training

Example Application 2: Enabling **unsupervised** image-to-image training

"Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" [Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros]
Example Application 2: Enabling *unsupervised* image-to-image training

**Takeaway:**

adversarial losses enable discovery of latent correspondances in the structure of two datasets

Example Application 3:
Cross-domain attribute manipulation

“PuppetGAN: Cross-Domain Image Manipulation by Demonstration” [B. Usman, N. Dufour, K. Saenko, C. Bregler]
Example Application 3: Cross-domain attribute manipulation

“PuppetGAN: Cross-Domain Image Manipulation by Demonstration” [B. Usman, N. Dufour, K. Saenko, C. Bregler]
Example Application 3: Cross-domain attribute manipulation

We trained a **disentangled autoencoder**: we **split** the encoded vector into **two parts** and **force** one part to represent the attribute we manipulate (mouth) and other attributes (hair, mic, …).

How?

“PuppetGAN: Cross-Domain Image Manipulation by Demonstration” [B. Usman, N. Dufour, K. Saenko, C. Bregler]
Example Application 3:
Cross-domain attribute manipulation

We combined autoencoder and cycle losses on both domains ...
Example Application 3: Cross-domain attribute manipulation

.. with **supervised losses** on synthetic data ...
Example Application 3: Cross-domain attribute manipulation

... with GAN losses ...

“PuppetGAN: Cross-Domain Image Manipulation by Demonstration” [B. Usman, N. Dufour, K. Saenko, C. Bregler]
Example Application 3: Cross-domain attribute manipulation

... and **compositional constraint losses** to ensure that all components are used.

“PuppetGAN: Cross-Domain Image Manipulation by Demonstration” [B. Usman, N. Dufour, K. Saenko, C. Bregler]
Example Application 3:
Cross-domain attribute manipulation

Takeaway:

adversarial losses enable “forcing” the model to store information necessary for reconstructing specific “aspects” of the input image at specific dimensions of the latent code

“PuppetGAN: Cross-Domain Image Manipulation by Demonstration” [B. Usman, N. Dufour, K. Saenko, C. Bregler]
Outline

1. Some applications of convolutional autoencoders and GAN
   a. Image-to-Image Translation with Conditional Adversarial Nets
   b. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
   c. PuppetGAN: Cross-Domain Image Manipulation by Demonstration

2. Normalizing Flows
   a. Change of variable formula
   b. Planar and radial flows
   c. Real NVP
   d. GLOW
   e. FFJORD
   f. Likelihood vs probability
we learn to compute \( \text{model}(x, \theta) = P(x \mid \theta) \)

choose \( \theta \) such that the observed dataset could have been generated by \( \text{model}(\theta) \)

we learn to sample \( x' \sim \text{model}(\theta) \)
How to train a data model from positive samples only?

If we trained a neural network $f(x; \theta)$ to have high values at our training points $x_i$, it could just shift everything upwards.

We could train a GAN to generate “negative” samples, but the whole procedure becomes fragile.

We could use a model with a “fixed budget”, i.e. an autoencoder (# points it can “remember”) or a density models (integrates to one).
Why train a model from positive samples only?

1. **Adversarial Robustness**: If the input $X$ is not from the training distribution $P(X)$, refuse classification.

2. **Detecting Data Shift**: if $P(X)$ shifted over time, retrain the model.

3. **Outlier Detection**: detect abnormalities in observed data.

4. **“Learned” data priors**: improved image synthesis or structure in segmentation maps.
Background: change of variable formula

\[ \text{U}[0, 1] \quad \text{P}(0.5) = 1 \]

\[ \text{U}[1, 3] \quad \text{P}(T(0.5)) = \text{P}(2) = 0.5 \]

\[ T(x) = 2x + 1 \]
Background: change of variable formula

\[ \int_{\varphi(a)}^{\varphi(b)} f(u) \, du = \int_{a}^{b} f(\varphi(x)) \varphi'(x) \, dx. \]

\[ p(x) \, dx = p(y) \, dy \]

\[ p(y) = p(x) \, \det \left| \frac{dT^{-1}(y)}{dy} \right| \]

\[ \log p(y) = \log p(x) + \log \det \left| \frac{dT^{-1}(y)}{dy} \right| \]
Background: change of variable formula

\[ T(x) = 2x + 1 \]
\[ T^{-1}(y) = \frac{y}{2} - \frac{1}{2} \]
\[ \frac{dT^{-1}(y)}{dy} = \frac{1}{2} \]

\[ p_Y(y) = p_X(T^{-1}(y)) \cdot \text{det} \left| \left( \frac{dT^{-1}(y)}{dy} \right) \right| \]
\[ p_Y(y) = I[0 < (y/2 - \frac{1}{2}) < 1] \cdot \frac{1}{2} \]
\[ = I[1 < y < 3] \cdot \frac{1}{2} \]

\[ U[0, 1] \]
\[ P(0.5) = 1 \]

\[ U[1, 3] \]
\[ P(T(0.5)) = P(2) = 0.5 \]
Normalizing flows for density estimation

like shuffling a sand castle - we move sand around to increase the amount of sand near data points, but the total amount of the sand stays constant
Normalizing flows for density estimation

\[ P(T(x)) \]

\[ P_A(x; T) = P(T(x)) * \det | \nabla T(x) | * \text{Vol}(B(x)) \]

let's solve the problem “backwards”
Normalizing flows for density estimation

\[ P_A(x; F) = P(T(x)) \cdot \det |\nabla T(x)| \cdot \text{Vol}(T(x)) \]

Low probability areas are “stretches”
In order to define a normalizing flow model we need

1. A **one-to-one** mapping $F(x, \theta): \mathbb{R}^n \rightarrow \mathbb{R}^n$
2. $F^{-1}(x, \theta)$
3. $\det[\text{Jac } F(x, \theta)]$

$$
J = \begin{bmatrix}
\frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_n}
\end{bmatrix} = 
\begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n}
\end{bmatrix}.
$$
Planar Flow

\[ f(z) = z + u h(w^T z + b), \tag{4} \]

with \( u, w \in \mathbb{R}^d \) and \( b \in \mathbb{R} \) and \( h \) an element-wise non-linearity. Let \( \psi(z) = h'(w^T z + b) w \). The determinant can be easily computed as

\[ \left| \det \frac{\partial f}{\partial z} \right| = \left| 1 + u^T \psi(z) \right|. \tag{5} \]

We can think of it as slicing the \( z \)-space with straight lines (or hyperplanes), where each line contracts or expands the space around it, see figure 1.

Radial Flow

\[ f(z) = z + \beta h(\alpha, r)(z - z_0), \tag{6} \]

with \( r = \|z - z_0\|_2 \), \( h(\alpha, r) = \frac{1}{\alpha + r} \) and parameters \( z_0 \in \mathbb{R}^d \), \( \alpha \in \mathbb{R}_+ \) and \( \beta \in \mathbb{R} \).
to learn more complex distributions, apply multiple flows in a row
Real Non-Volume Preserving Flows (R-NVP)

\[ y_{1:k} = z_{1:k}, \]
\[ y_{k+1:d} = z_{k+1:d} \circ \sigma(z_{1:k}) + \mu(z_{1:k}). \]

\[ \det \frac{\partial y}{\partial z} = \prod_{i=1}^{d-k} \sigma_i(z_{1:k}). \]

“Density estimation using Real NVP” by Laurent Dinh, Jascha Sohl-Dickstein, Samy Bengio
Real Non-Volume Preserving Flows (R-NVP)

\[ x \sim \hat{p}_X \]
\[ z = f(x) \]
Real Non-Volume Preserving Flows (R-NVP)

Inference
\[ x \sim \hat{p}_X \]
\[ z = f(x) \]

Generation
\[ z \sim p_Z \]
\[ x = f^{-1}(z) \]
Real Non-Volume Preserving Flows (R-NVP)

checkerboard swap

“squeeze”
Real Non-Volume Preserving Flows (R-NVP)
Glow: Generative Flow with Invertible 1x1 Convolutions

= very deep RealNVP
+ invertible 1x1 conv instead of swap
+ multiscale features
Glow: Generative Flow with Invertible 1x1 Convolutions

= very deep R-NVP
+ invertible 1x1 conv instead of swap
+ multiscale features
Neural ODE and FFJORD

\[ x(t_0) = x_0 \]
\[ x'(t) = f(x(t)) \]
\[ x(t_1) = ? \]

What if we use an **explicit method** with **fixed** step size?
Neural ODE and FFJORD

\[ x(t_0) = x_0 \]
\[ x'(t) = f(x(t)) \]
\[ x(t_1) = ? \]

If we use a “proper solver”, we get adaptive step size.
Neural ODE and FFJORD

\[ x(t_0) = x_0 \]
\[ x'(t) = f(x(t)) \]
\[ x(t_1) = ? \]

If we use a “proper solver”, we get adaptive step size.
Neural ODE and FFJORD

FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models
If model $P_A(x)$ is trained on $A$ then for many datasets $B$

$$P_A(B) > P_A(A)$$

which is quite counter-intuitive …

and we never see anything like $B$ if we sample from $P(A) ..$
High likelihood of $X$ does not mean that $X$ is likely!

$$A = \{ y \in \mathbb{R}^{100} \mid \|y\| \leq \varepsilon \}$$

$$P(A) = \int_A N_{100}(x; 0, I) \, dx$$

- the **probability** of point being in set $A$ is **low**

$$B = \{ x_i \in \mathbb{R}^{100} \mid \|x\| \leq \varepsilon \}_{i=0}^N$$

$$L(B) = \frac{1}{N} \sum_{x_i \in B} \log N_{100}(x; 0, I)$$

- but the mean **likelihood** of points from this set is **high**
Final takeaways

1. GANs can help to learn and use the structure of the output domain
2. Normalizing Flows enable density estimation in higher dimensions