Goal of today’s class

- Recall Sammon mapping, T-SNE, AE
- VAE
- Generative adversarial networks (GANs)
Autoencoders

• Autoencoders are designed to reproduce their input, especially for images.
  – Key point is to reproduce the input from a learned encoding.
Variational Autoencoder (VAE)

- Key idea: make both the encoder and the decoder probabilistic.
- I.e., the latent variables, $z$, are drawn from a probability distribution depending on the input, $X$, and the reconstruction is chosen probabilistically from $z$. 

![Diagram of VAE](image)
VAE Encoder

• The encoder takes input and returns parameters for a probability density (e.g., Gaussian): i.e., \( q_\theta(z | x) \) gives the mean and co-variance matrix.

• We can sample from this distribution to get random values of the lower-dimensional representation \( z \).

• Implemented via a neural network: each input \( x \) gives a vector mean and diagonal covariance matrix that determine the Gaussian density \( q_\theta(z | x) \).

• Parameters \( \theta \) for the NN need to be learned – need to set up a loss function.
VAE Decoder

• The decoder takes latent variable $z$ and returns parameters for a distribution. E.g., $p_\phi(x|z)$ gives the mean and variance for each pixel in the output.

• Reconstruction $\tilde{x}$ is produced by sampling.

• Implemented via neural network, the NN parameters $\phi$ are learned.
VAE loss function

• Loss function for autoencoder: $L_2$ distance between output and input (or clean input for denoising case)

• For VAE, we need to learn parameters of two probability distributions. For a single input, $x_i$, we maximize the expected value of returning $x_i$ or minimize the expected negative log likelihood.

\[-\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i | z)]\]

• This takes expected value wrt $z$ over the current distribution $q_\theta(z|x_i)$ of the loss $-\log p_\phi(x_i | z)$
VAE loss function

• **Problem:** the weights may adjust to memorize input images via $z$. I.e., input that we regard as similar may end up very different in $z$ space.

• **We prefer continuous latent representations to give meaningful parameterizations.** E.g., smooth changes from one digit to another.

• **Solution:** Try to force $q_\theta(z|x_i)$ to be close to a standard normal (or some other simple density).
VAE loss function

• For a single data point $x_i$ we get the loss function

$$ l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_0(z|x_i)}[\log p_\phi(x_i | z)] + \text{KL}(q_\theta(z | x_i) || p(z)) $$

• The first term promotes recovery of the input.
• The second term keeps the encoding continuous – the encoding is compared to a fixed $p(z)$ regardless of the input, which inhibits memorization.
• With this loss function the VAE can (almost) be trained using gradient descent on minibatches.
VAE loss function

- For a single data point $x_i$ we get the loss function

\[
l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)] + \text{KL}(q_\theta(z \mid x_i) \mid\mid p(z))
\]

- **Problem:** The expectation would usually be approximated by choosing samples and averaging. This is not differentiable wrt $\theta$ and $\phi$. 
VAE loss function

- **Problem:** The expectation would usually be approximated by choosing samples and averaging. This is not differentiable with respect to $\theta$ and $\phi$. 

![Diagram of encoder and decoder](image.png)
VAE loss function

- **Reparameterization:** If $z$ is $N(\mu(x_i), \Sigma(x_i))$, then we can sample $z$ using $z = \mu(x_i) + \sqrt{\Sigma(x_i)} \epsilon$, where $\epsilon$ is $N(0,1)$. So we can draw samples from $N(0,1)$, which doesn’t depend on the parameters.
VAE generative model

• After training, $q_\theta(z|x_i)$ is close to a standard normal, $N(0,1)$ – easy to sample.

• Using a sample of $z$ from $q_\theta(z|x_i)$ as input to sample from $p_\phi(x|z)$ gives an approximate reconstruction of $x_i$, at least in expectation.

• If we sample any $z$ from $N(0,1)$ and use it as input to sample from $p_\phi(x|z)$ then we can approximate the entire data distribution $p(x)$. I.e., we can generate new samples that look like the input but aren’t in the input.
Autoencoder

As close as possible

Randomly generate a vector as code

Image ?
Autoencoder with 3 fully connected layers

Training: model.fit(X, X)
Cost function: \( \sum_{k=1..N} (x_k - x'_k)^2 \)
Auto-encoder
Auto-encoder

input \rightarrow \text{NN Encoder} \rightarrow \text{code} \rightarrow \text{NN Decoder} \rightarrow output

VAE

input \rightarrow \text{NN Encoder} \rightarrow \text{code} \rightarrow \text{NN Decoder} \rightarrow output

\begin{align*}
\text{minimize} & \quad \text{reconstruction error} \\
\Sigma_{i=1..3} & \quad [\exp(\sigma_i) - (1+\sigma_i) + (m_i)^2 ]
\end{align*}

This constrains \( \sigma_i \) approaching 0 is good

Problems of VAE

- It does not really try to simulate real images

**code** -> NN Decoder -> Output

As close as possible

One pixel difference to the target

Realistic

Fake

VAE treats these the same

One pixel difference to the target

Also one pixel difference to the target
Gradual and step-wise generation

NN Generator v1 → Discriminator v1 → NN Generator v2 → Discriminator v2 → NN Generator v3 → Discriminator v3

These are Binary classifiers

Real images: 5 0 4 1

Generated images: 2 3 9 9 0 0 0 0
Generative Adversarial Networks

- GAN was first introduced by Ian Goodfellow et al in 2014
- Have been used in generating images, videos, poems, some simple conversation.
- Note, image processing is easy (all animals can do it), NLP is hard (only human can do it).
- This co-evolution approach might have far-reaching implications. Bengio: this may hold the key to making computers a lot more intelligent.

- Tips for training GAN:
  https://github.com/soumith/ganhacks
GAN – Learn a discriminator

Randomly sample a vector

Something like Decoder in VAE

Real images Sampled from DB:

1/0 (real or fake)
GAN – Learn a generator

Updating the parameters of generator

The output be classified as “real” (as close to 1 as possible)

Generator + Discriminator = a network

Using gradient descent to update the parameters in the generator, but fix the discriminator

Randomly sample a vector

Train this

They have Opposite objectives

Do not Train This
Next Video Frame Prediction

Ground Truth  MSE  Adversarial

Traditional mean-squared Error, averaged, blurry

(Lotter et al 2016)
Single Image Super-Resolution

original
bicubic (21.59dB/0.6423)
SRResNet (23.44dB/0.7777)
SRGAN (20.34dB/0.6562)

(Ledig et al 2016)

Last 2 are by deep learning approaches.
Image to Image Translation

(Isola et al 2016)
DCGANs for LSUN Bedrooms

(Radford et al 2015)
Similar to word embedding (DCGAN paper)

Vector Space Arithmetic

- Man with glasses
- Man
- Woman

= Woman with Glasses

(Radford et al, 2015)
256x256 high resolution pictures by Plug and Play generative network

PPGN Samples

(Nguyen et al 2016)
From natural language to pictures

PPGN for caption to image

oranges on a table next to a liquor bottle

(Nguyen et al 2016)
Deriving GAN

• How to derive and train GAN
• I will avoid the continuous case and stick to simple explanations.
Maximum Likelihood Estimation

• Give a data distribution $P_{data}(x)$
• We use a distribution $P_G(x; \theta)$ parameterized by $\theta$ to approximate it
  – E.g. $P_G(x; \theta)$ is a Gaussian Mixture Model, where $\theta$ contains means and variances of the Gaussians.
  – We wish to find $\theta$ s.t. $P_G(x; \theta)$ is close to $P_{data}(x)$
• In order to do this, we can sample
  \{x^1, x^2, \ldots x^m\} from $P_{data}(x)$
• The likelihood of generating these $x^i$'s under $P_G$ is
  \[ L = \prod_{i=1}^{m} P_G(x^i; \theta) \]
• Then we can find $\theta^*$ maximizing the $L$. 
KL (Kullback-Leibler) divergence

- **Discrete:**
  \[ D_{KL}(P \| Q) = \sum_i P(i) \log[P(i)/Q(i)] \]

- **Continuous:**
  \[ D_{KL}(P \| Q) = \int_{-\infty}^{\infty} p(x) \log \left[\frac{p(x)}{q(x)}\right] \]

- **Explanations:**
  - **Entropy:** \( \sum_i P(i) \log P(i) \) - expected code length (also optimal)
  - **Cross Entropy:** \( \sum_i P(i) \log Q(i) \) – expected coding length using optimal code for Q
  
  \[ D_{KL} = \sum_i P(i) \log[P(i)/Q(i)] = \sum_i P(i)[\log P(i) - \log Q(i)], \text{ extra bits} \]
  
  \[ JSD(P \| Q) = \frac{1}{2} D_{KL}(P \| M) + \frac{1}{2} D_{KL}(Q \| M), \text{ M= } \frac{1}{2} (P+Q), \text{ symmetric KL} \]

* JSD = Jensen-Shannon Divergency
**Maximum Likelihood Estimation**

\[ \theta^* = \arg \max_{\theta} \prod_{i=1..m} P_G(x^i; \theta) \]

\[ \arg \max_{\theta} \log \prod_{i=1..m} P_G(x^i; \theta) \]

\[ = \arg \max_{\theta} \sum_{i=1..m} \log P_G(x^i; \theta), \ {x^1, ..., x^m} \text{ sampled from } P_{\text{data}}(x) \]

\[ = \arg \max_{\theta} \sum_{i=1..m} P_{\text{data}}(x^i) \log P_G(x^i; \theta) \quad \text{--- this is cross entropy} \]

\[ \approx \arg \max_{\theta} \sum_{i=1..m} P_{\text{data}}(x^i) \log P_G(x^i; \theta) - \sum_{i=1..m} P_{\text{data}}(x^i) \log P_{\text{data}}(x^i) \]

\[ = \arg \min_{\theta} \text{KL} \left( P_{\text{data}}(x) \mid \mid P_G(x; \theta) \right) \quad \text{--- this is KL divergence} \]

**Note:** \( P_G \) is Gaussian mixture model, finding best \( \theta \) will still be Gaussians, this only can generate a few blubs. Thus this above maximum likelihood approach does not work well.

Next we will introduce GAN that will change \( P_G \), not just estimating \( P_G \) is parameters. We will find best \( P_G \), which is more complicated and structured, to approximate \( P_{\text{data}} \).
Thus let’s use an NN as $P_G(x; \theta)$

How to compute the likelihood?

$$P_G(x) = \int P_{\text{prior}}(z) I_{[G(z) = x]} \, dz$$
Basic Idea of GAN

• Generator G
  – G is a function, input z, output x
  – Given a prior distribution $P_{\text{prior}}(z)$, a probability distribution $P_G(x)$ is defined by function G

• Discriminator D
  – D is a function, input x, output scalar
  – Evaluate the “difference” between $P_G(x)$ and $P_{\text{data}}(x)$

• In order for D to find difference between $P_{\text{data}}$ from $P_G$, we need a cost function $V(G,D)$:
  $$G^* = \text{arg min}_G \text{max}_D V(G,D)$$

Note, we are changing distribution G, not just update its parameters (as in the max likelihood case).
Basic Idea

\[ G^* = \arg \min_G \max_D V(G, D) \]

Pick JSD function: \( V = E_{x \sim P_{\text{data}}} [\log D(x)] + E_{x \sim P_G}[\log(1-D(x))] \)

Given a generator \( G \), \( \max_D V(G, D) \) evaluates the “difference” between \( P_G \) and \( P_{\text{data}} \)

Pick the \( G \) s.t. \( P_G \) is most similar to \( P_{\text{data}} \)
Max_D V(G,D), G* = arg min_G max_D V(G,D)

Given G, what is the optimal D* maximizing

V = \( E_{x \sim P_{\text{data}}} [\log D(x)] + E_{x \sim P_G} [\log(1-D(x))] \)
\[= \Sigma \left[ P_{\text{data}}(x) \log D(x) + P_G(x) \log(1-D(x)) \right] \]

Thus: \( D^*(x) = \frac{P_{\text{data}}(x)}{P_{\text{data}}(x) + P_G(x)} \)

Assuming D(x) can have any value here

Given x, the optimal D* maximizing is:
\( f(D) = a \log D + b \log(1-D) \Rightarrow D^* = \frac{a}{a+b} \)
\[
\max_D V(G,D), \quad G^* = \arg \min_G \max_D V(G,D)
\]

\[
D_1^*(x) = \frac{P_{\text{data}}(x)}{(P_{\text{data}}(x)+P_{G_1}(x))}
\]

\[
D_2^*(x) = \frac{P_{\text{data}}(x)}{(P_{\text{data}}(x)+P_{G_2}(x))}
\]

“difference” between \(P_{G_1}\) and \(P_{\text{data}}\)
\[ \max_D V(G,D) \]

\[ V = E_{x \sim P_{data}} [\log D(x)] + E_{x \sim P_G} [\log(1-D(x))] \]

\[ \max_D V(G,D) = V(G,D^*), \text{ where } D^*(x) = \frac{P_{data}}{(P_{data} + P_G)}, \text{ and} \]

\[ 1-D^*(x) = \frac{P_G}{(P_{data} + P_G)} \]

\[ = E_{x \sim P_{data}} \log D^*(x) + E_{x \sim P_G} \log (1-D^*(x)) \]

\[ \approx \sum [P_{data}(x) \log D^*(x) + P_G(x) \log (1-D^*(x))] \]

\[ = -2\log2 + 2 \text{ JSD}(P_{data} \parallel P_G), \]

JSD(P||Q) = Jensen-Shannon divergence

\[ = \frac{1}{2} D_{KL}(P||M) + \frac{1}{2} D_{KL}(Q||M) \]

where \( M = \frac{1}{2} (P+Q) \).

\[ D_{KL}(P||Q) = \sum P(x) \log P(x) / Q(x) \]
Summary:

- Generator $G$, Discriminator $D$
- Looking for $G^*$ such that

$$G^* = \arg \min_G \max_D V(G, D)$$

- Given $G$, $\max_D V(G, D)$

$$= -2\log 2 + 2\text{JSD}(P_{\text{data}}(x) \parallel P_G(x))$$

- What is the optimal $G$? It is $G$ that makes JSD smallest $= 0$:

$$P_G(x) = P_{\text{data}}(x)$$
Algorithm

\[ G^* = \arg \min_G \max_D V(G,D) \]

L(G), this is the loss function

- To find the best G minimizing the loss function L(G):
  \[ \theta_G \leftarrow \theta_G = -\eta \frac{\partial L(G)}{\partial \theta_G} \text{, } \theta_G \text{ defines } G \]

- Solved by gradient descent. Having max ok. Consider simple case:

\[ f(x) = \max \{ D_1(x), D_2(x), D_3(x) \} \]

If \( D_i(x) \) is the Max in that region, then do \( \frac{dD_i(x)}{dx} \)
Algorithm

\[ G^* = \arg \min_G \max_D V(G, D) \]

- Given \( G_0 \)
- Find \( D_0^* \) maximizing \( V(G_0, D) \)
  
  \( V(G_0, D_0^*) \) is the JS divergence between \( P_{\text{data}}(x) \) and \( P_{G_0}(x) \)
  
  \[ \theta_G \leftarrow \theta_G - \eta \frac{\Delta V(G, D_0^*)}{\theta_G} \rightarrow \text{Obtaining } G_1 \text{ (decrease JSD)} \]
- Find \( D_1^* \) maximizing \( V(G_1, D) \)
  
  \( V(G_1, D_1^*) \) is the JS divergence between \( P_{\text{data}}(x) \) and \( P_{G_1}(x) \)
  
  \[ \theta_G \leftarrow \theta_G - \eta \frac{\Delta V(G, D_1^*)}{\theta_G} \rightarrow \text{Obtaining } G_2 \text{ (decrease JSD)} \]
- And so on ...
In practice ...

- Given G, how to compute $\max_D V(G, D)$?
  - Sample $\{x^1, ..., x^m\}$ from $P_{\text{data}}$
  - Sample $\{x^{*1}, ..., x^{*m}\}$ from generator $P_G$

Maximize:

$$V' = \frac{1}{m} \sum_{i=1}^{m} \log D(x^i) + \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(x^{*i}))$$

This is what a Binary Classifier do

Output is $D(x)$

Minimize Cross-entropy

If $x$ is a positive example

Minimize $-\log D(x)$

If $x$ is a negative example

Minimize $-\log(1 - D(x))$
Binary Classifier

Output is $f(x)$  

\[ \text{Minimize} \quad \text{Cross-entropy} \]

If $x$ is a positive example  
Minimize $-\log f(x)$

If $x$ is a negative example  
Minimize $-\log(1-f(x))$

$D$ is a binary classifier (can be deep) with parameters $\theta_d$

\[ \{x^1, x^2, \ldots, x^m\} \text{ from } P_{\text{data}}(x) \quad \Rightarrow \quad \text{Positive examples} \]

\[ \{x^{*1}, x^{*2}, \ldots, x^{*m}\} \text{ from } P_G(x) \quad \Rightarrow \quad \text{Negative examples} \]

Minimize \[ L = -V' \]

or

Maximize \[ V' = \sum_{i=1..m} \log D(x^i) + \frac{1}{m} \sum_{i=1..m} \log(1-D(x^{*i})) \]
Algorithm

In each training iteration:

- Sample $m$ examples $\{x^1, x^2, \ldots, x^m\}$ from data distribution $P_{\text{data}}(x)$
- Sample $m$ noise samples $\{z^1, \ldots, z^m\}$ from a simple prior $P_{\text{prior}}(z)$
- Obtain generated data $\{x^{*1}, \ldots, x^{*m}\}$, $x^{*i} = G(z^i)$
- Update discriminator parameters $\theta_d$ to maximize
  
  - $V' \approx \frac{1}{m} \sum_{i=1}^{m} \log D(x^i) + \frac{1}{m} \sum_{i=1}^{m} \log(1-D(x^{*i}))$
  
  - $\theta_d \leftarrow \theta_d + \eta \Delta V'(\theta_d)$ (gradient ascent)

- Simple another $m$ noise samples $\{z^1, z^2, \ldots, z^m\}$ from the prior $P_{\text{prior}}(z)$, $G(z^i) = x^{*i}$
- Update generator parameters $\theta_g$ to minimize
  
  - $V' = \frac{1}{m} \sum_{i=1}^{m} \log D(x^i) + \frac{1}{m} \sum_{i=1}^{m} \log(1-D(x^{*i}))$
  
  - $\theta_g \leftarrow \theta_g - \eta \Delta V'(\theta_g)$ (gradient descent)

Initialize $\theta_d$ for D and $\theta_g$ for G

Can only find lower bound of JSD or $\max_d V(G,D)$
Objective Function for Generator in Real Implementation

\[ V = E_{x \sim P_{\text{data}}} \left[ \log D(x) \right] + E_{x \sim P_G}[\log(1-D(x))] \]

Training slow at the beginning

\[ V = E_{x \sim P_G} \left[ - \log(D(x)) \right] \]

Real implementation: label \( x \) from \( P_G \) as positive

\[ \log(1-D(x)) \]

\[ -\log(D(x)) \]
Evaluating JS divergence

Discriminator is too strong: for all three Generators, JSD = 0

Evaluating JS divergence

- JS divergence estimated by discriminator telling little information
Discriminator

Reason 1. Approximate by sampling

Weaken your discriminator?

Can weak discriminator compute JS divergence?

\[ V = E_{x \sim P_{data}} [\log D(x)] + E_{x \sim P_{G}}[\log(1-D(x))] \]
\[ = \frac{1}{m} \sum_{i=1..m} \log D(x^i) + \frac{1}{m} \sum_{i=1..m} \log(1-D(x^*)i) \]

\[ \max_D V(G,D) = -2 \log 2 + 2 \text{ JSD}(P_{data} \parallel P_{G}) \]

\[ = 0 \]

log 2 when \( P_{data} \) and \( P_{G} \) differ completely
One simple solution: add noise

- Add some artificial noise to the inputs of discriminator
- Make the labels noisy for the discriminator

Discriminator cannot perfectly separate real and generated data

$P_{\text{data}}(x)$ and $P_G(x)$ have some overlap

Noises need to decay over time
Mode Collapse

Data Distribution

Generated Distribution

Converge to same faces

Sometimes, this is hard to tell since one sees only what’s generated, but not what’s missed.
Mode Collapse Example

8 Gaussian distributions:

What we want …

In reality …
Experimental Results

• Approximate a mixture of Gaussians by single mixture
## Text to Image, by conditional GAN

<table>
<thead>
<tr>
<th>Caption</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>a pitcher is about to throw the ball to the batter</td>
<td><img src="image1.jpg" alt="Images from Text to Image" /></td>
</tr>
<tr>
<td>a group of people on skis stand in the snow</td>
<td><img src="image2.jpg" alt="Images from Text to Image" /></td>
</tr>
<tr>
<td>a man in a wet suit riding a surfboard on a wave</td>
<td><img src="image3.jpg" alt="Images from Text to Image" /></td>
</tr>
<tr>
<td>Caption</td>
<td>Image</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>this flower has white petals and a yellow stamen</td>
<td>![Image of white flowers with yellow center]</td>
</tr>
<tr>
<td>the center is yellow surrounded by wavy dark purple petals</td>
<td>![Image of purple flowers]</td>
</tr>
<tr>
<td>this flower has lots of small round pink petals</td>
<td>![Image of pink flowers]</td>
</tr>
</tbody>
</table>
WGAN Background

• We have seen that JSD does not give GAN a smooth and continuous improvement curve.
• We would like to find another distance which gives that.
• This is the Wasserstein Distance or earth mover’s distance.
Earth Mover’s Distance

• Considering one distribution $P$ as a pile of earth (total amount of earth is 1), and another distribution $Q$ (another pile of earth) as the target

• The “earth mover’s distance” or “Wasserstein Distance” is the average distance the earth mover has to move the earth in an optimal plan.

$$W(P,Q) = d$$
Earth Mover’s Distance: best plan to move
JS vs Earth Mover’s Distance

\[ W(P_{G_0}, P_{data}) = d_0 \]
\[ W(P_{G_{50}}, P_{data}) = d_{50} \]
\[ W(P_{G_{100}}, P_{data}) = 0 \]

\[ JS(P_{G_0}, P_{data}) = \log 2 \]
\[ JS(P_{G_{50}}, P_{data}) = \log 2 \]
\[ JS(P_{G_{100}}, P_{data}) = 0 \]
Explaining WGAN

• Let W be the Wasserstein distance.

\[ W(P_{\text{data}}, P_G) = \max_{D \text{ is } 1-\text{Lipschitz}} [E_{x \sim P_{\text{data}}} D(x) - E_{x \sim P_G} D(x)] \]

Where a function \( f \) is a k-Lipschitz function if

\[ ||f(x_1) - f(x_2)|| \leq k ||x_1 - x_2|| \]

How to guarantee this?

Weight clipping: for all parameter updates, if \( w > c \)
Then \( w = c \), if \( w < -c \), then \( w = -c \).

Blue: \( D(x) \) for original GAN
Green: \( D(x) \) for WGAN

WGAN will provide gradient to push \( P_G \) towards \( P_{\text{data}} \).
Earth Mover Distance Examples:

Multi-layer perceptron
Lab this week

• T-SNE plots on MNIST, ex.19.2
• Next week: train a simple AE on MNIST (will upload a pdf doc for the AE architecture)