Lecture 11: Unsupervised and reinforcement learning
Announcements

- Project milestone due Friday 2/28
Supervised learning

Data: \((x, y)\)

- \(x\) is data, \(y\) is label

Goal: Learn a function to map \(x \rightarrow y\)

Examples: Classification, regression, semantic segmentation, object detection, instance segmentation

Classification

Right effusion
Now: Unsupervised learning

Data: $x$

Just data, no labels!

Goal: Learn some underlying hidden structure of the data

Examples: Clustering, representation / feature learning, density estimation, etc.
Now: Unsupervised learning

**Data**: $x$

Just data, no labels!

**Goal**: Learn some underlying hidden *structure* of the data

**Examples**: Clustering, representation / feature learning, density estimation, etc.

K-means clustering
Now: Unsupervised learning

**Data:** $x$

Just data, no labels!

**Goal:** Learn some underlying hidden *structure* of the data

**Examples:** Clustering, representation / feature learning, density estimation, etc.

---

Diagram:

- **Input data** $x$
- **Features** $z$
- **Encoder**
- **Unsupervised training objective** $\hat{x}$
- **Representation learning**
Every unique token is numerically represented by an “embedding vector” that will represent the token in the model. The embedding vector values are learned; similar tokens will probably have similar embedding vectors.

Remember: Token embeddings from EHR

\[ \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & \ldots & 0 \end{bmatrix} \times \begin{bmatrix} 0.5 & 0.2 & 0.1 \\ 0.6 & 0.1 & 0.6 \\ 0.5 & 0.8 & 0.2 \\ 0.7 & 0.9 & 0.3 \\ 0.3 & 0.5 & 0.1 \\ \ldots \\ 0.7 & 0.8 & 0.1 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.8 & 0.2 \end{bmatrix} \]

1xN token input (one-hot selection of token)

N x D embedding matrix

D-dim token embedding
Can learn token (or word) embeddings in unsupervised manner from large amounts of unlabeled data

From natural language processing: Word2Vec

Training data: a large corpus of unlabeled text

Train a neural network to predict words in the immediate context of any given word (classification problem)


Figure credit: https://medium.com/datadriveninvestor/word2vec-skip-gram-model-explained-383fa6ddc4ae
Can learn token (or word) embeddings in an unsupervised manner from large amounts of unlabeled data.

From natural language processing:
Word2Vec

Training data: a large corpus of unlabeled text

Train a neural network to predict words in the immediate context of any given word (classification problem)


Figure credit: https://medium.com/datadriveninvestor/word2vec-skip-gram-model-explained-383fa6ddc4ae
Can learn token (or word) embeddings in unsupervised manner from large amounts of unlabeled data

From natural language processing: Word2Vec

Training data: a large corpus of unlabeled text

Train a neural network to predict words in the immediate context of any given word (classification problem)

Can learn token (or word) embeddings in unsupervised manner from large amounts of unlabeled data

From natural language processing:
Word2Vec

Training data: a large corpus of unlabeled text

Train a neural network to predict words in the immediate context of any given word (classification problem)

Can learn token (or word) embeddings in unsupervised manner from large amounts of unlabeled data

From natural language processing:
Word2Vec

Training data: a large corpus of unlabeled text

Train a neural network to predict words in the immediate context of any given word (classification problem)

Can learn token (or word) embeddings in unsupervised manner from large amounts of unlabeled data

From natural language processing:
Word2Vec

Training data: a large corpus of unlabeled text

Train a neural network to predict words in the immediate context of any given word (classification problem)

Once trained, the first fully connected layer maps from input words to word embeddings (aka features, word representations)

Can learn token (or word) embeddings in unsupervised manner from large amounts of unlabeled data

From natural language processing:
Word2Vec

Training data: a large corpus of unlabeled text

Train a neural network to predict words in the immediate context of any given word (classification problem)

Many unsupervised objectives are possible: predict center word from average of context vectors (also in Mikolov 2013, predict next word of a sequence, etc. See ELMo (Peters 2018), BERT (Devlin 2018), for more recent examples.

Med2Vec (Choi et al. 2016)

Built on ideas from Word2Vec, on EHR data (medical codes) from Children's Hospital of Atlanta and CMS claims data

Used learned feature representation as input for downstream supervised prediction tasks (e.g. prediction diagnosis codes on next visit)

Also showed interpretability of different axes (coordinates) learned feature representation

<table>
<thead>
<tr>
<th>Coordinate 112</th>
<th>Coordinate 152</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kidney replaced by transplant (V42.0)</td>
<td>X-ray, knee (P)</td>
</tr>
<tr>
<td>Hb-SS disease without crisis (282.61)</td>
<td>X-ray, thoracolumbar (P)</td>
</tr>
<tr>
<td>Heart replaced by transplant (V42.1)</td>
<td>Accidents in public building (E849.6)</td>
</tr>
<tr>
<td>RBC antibody screening (P)</td>
<td>Activities involving gymnastics (E005.2)</td>
</tr>
<tr>
<td>Complications of transplanted bone marrow (996.85)</td>
<td>Struck by objects/persons in sports (E917.0)</td>
</tr>
<tr>
<td>Sickle-cell disease (282.60)</td>
<td>Encounter for removal of sutures (V38.32)</td>
</tr>
<tr>
<td>Liver replaced by transplant (V42.7)</td>
<td>Struck by object in sports (E917.5)</td>
</tr>
<tr>
<td>Hb-SS disease with crisis (282.62)</td>
<td>Unspecified fracture of ankle (824.8)</td>
</tr>
<tr>
<td>Prograf PO (R)</td>
<td>Accidents occurring in place for recreation and sport (E849.4)</td>
</tr>
<tr>
<td>Complications of transplanted heart (996.83)</td>
<td>Activities involving basketball (E007.6)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Coordinate 184</th>
<th>Coordinate 190</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pain in joint, shoulder region (719.41)</td>
<td>Down's syndrome (758.0)</td>
</tr>
<tr>
<td>Pain in joint, lower leg (719.46)</td>
<td>Congenital anomalies (759.89)</td>
</tr>
<tr>
<td>Pain in joint, ankle and foot (719.47)</td>
<td>Tuberous sclerosis (759.5)</td>
</tr>
<tr>
<td>Pain in joint, multiple sites (719.49)</td>
<td>Anomalies of larynx, trachea, and bronchus (748.3)</td>
</tr>
<tr>
<td>Generalized convulsive epilepsy (345.10)</td>
<td>Autosomal deletions (758.39)</td>
</tr>
<tr>
<td>Pain in joint, upper arm (719.42)</td>
<td>Conditions due to anomaly of unspecified chromosome (758.9)</td>
</tr>
<tr>
<td>Cerebral artery occlusion (434.91)</td>
<td>Acquired hypothyroidism (244.9)</td>
</tr>
<tr>
<td>MRI, brain (780.59)</td>
<td>Conditions due to chromosome anomalies (758.89)</td>
</tr>
<tr>
<td>Other joint derangement (718.81)</td>
<td>Anomalies of spleen (759.0)</td>
</tr>
<tr>
<td>Pelvic occult blood (790.6)</td>
<td>Conditions due to autosomal anomalies (758.5)</td>
</tr>
</tbody>
</table>

Codes with strongest values along different coordinates of learned feature representation

Another way of representation learning: autoencoders

L2 Loss function:
\[ ||x - \hat{x}||^2 \]

Input data \( x \) → Encoder → Features \( z \) → Decoder → Reconstructed data \( \hat{x} \)

Encoder: 4-layer conv
Decoder: 4-layer upconv

Autoencoders

Slide credit: CS231n
Another way of representation learning: autoencoders

L2 Loss function: $\| x - \hat{x} \|^2$

Autoencoders

Encoder: 4-layer conv
Decoder: 4-layer upconv

Input data

Features

Reconstructed input data

Reconstructed data

Slide credit: CS231n
Representation learning: autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data
Representation learning: autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

**Originally:** Linear + nonlinearity (sigmoid)

**Later:** Deep, fully-connected

**Later:** ReLU CNN

Slide credit: CS231n
Representation learning: autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

- \( z \) usually smaller than \( x \)  
  (dimensionality reduction)

Q: Why dimensionality reduction?

- Originally: Linear + nonlinearity (sigmoid)
- Later: Deep, fully-connected
- Later: ReLU CNN

Slide credit: CS231n
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

\[ z \text{ usually smaller than } x \] (dimensionality reduction)

Q: Why dimensionality reduction?

A: Want features to capture meaningful factors of variation in data

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN

Representation learning: autoencoders

Slide credit: CS231n
Representation learning: autoencoders

How to learn this feature representation?

Slide credit: CS231n
Representation learning: autoencoders

How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself

Encoder
Input data \( x \)
Features \( z \)
Decoder
Reconstructed input data \( \hat{x} \)

Slide credit: CS231n
How to learn this feature representation? 
Train such that features can be used to reconstruct original data 
“Autoencoding” - encoding itself

Originally: Linear + nonlinearity (sigmoid) 
Later: Deep, fully-connected 
Later: ReLU CNN (upconv)

Representation learning: autoencoders

Slide credit: CS231n
Representation learning: autoencoders

How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself

Input data $\mathbf{x}$

Encoder

Features $\mathbf{z}$

Decoder

Reconstructed input data $\hat{\mathbf{x}}$
Representation learning: autoencoders

Train such that features can be used to reconstruct original data

L2 Loss function:

\[ \| x - \hat{x} \|^2 \]

Encoder: 4-layer conv
Decoder: 4-layer upconv

Input data
Features
\[ Z \]
Encoder
\[ X \]
Decoder
\[ \hat{X} \]
Reconstructed input data
Reconstructed data

Slide credit: CS231n
Representation learning: autoencoders

Train such that features can be used to reconstruct original data

L2 Loss function: $\| x - \hat{x} \|^2$

Doesn’t use labels! -> unsupervised

Encoder: 4-layer conv
Decoder: 4-layer upconv

Input data
Features
Reconstructed input data

Reconstructed data
Input data
Encoder
Decoder

$\hat{x}$
Representation learning: autoencoders

After training, throw away decoder

Slide credit: CS231n
Representation learning: autoencoders

Encoder network can now be used as a feature extractor! Should be semantically meaningful features due to autoencoder loss from training.
Representation learning: autoencoders

Encoder network can now be used as a feature extractor! Should be semantically meaningful features due to autoencoder loss from training.

Features can be used for clustering, retrieval (e.g. find the closest patient to this one), etc.
In supervised learning tasks, an encoder trained in an unsupervised way (potentially on larger amounts of data) can also be used as a feature extractor for the task, or to initialize a supervised model.
Miotto 2016

- Used stack of denoising autoencoders (add noise to inputs to avoid overfitting) to learn feature representation from EHR data of 700,000 patients from Mount Sinai

- Used learned feature representation for downstream disease classification tasks

Darabi 2019

- Autoencoder-based unsupervised representation learning for **multimodal data** of 200,000 records from 250 hospital sites (eICU collaborative Research Database)

- Used feature representation to train models for downstream mortality, readmission prediction tasks
Darabi 2019

- Autoencoder-based unsupervised representation learning for **multimodal data** of 200,000 records from 250 hospital sites (eICU collaborative Research Database)

- Used feature representation to train models for downstream mortality, readmission prediction tasks

---

Darabi 2019

- Autoencoder-based unsupervised representation learning for **multimodal data** of 200,000 records from 250 hospital sites (eICU collaborative Research Database)

- Used feature representation to train models for downstream mortality, readmission prediction tasks


Concatenate feature representations from each autoencoder, and further fine-tune on predicting future elements in data.
Probabilistic version: variational autoencoder

Encoder network

Decoder network

Input Data

\( \mathcal{X} \)

\( \mathcal{Z} \)

\( \hat{\mathcal{X}} \)

Slide credit: CS231n
Probabilistic version: variational autoencoder

Input Data $\mathcal{X}$

Encoder network $q_\phi(z|x)$

Decoder network $p_\theta(x|z)$

$\hat{x}$

$z$

Slide credit: CS231n
Probabilistic version: variational autoencoder

Decoder network

\[ p_\theta(x|z) \]

Encoder network

\[ q_\phi(z|x) \]

Input Data

\[ \mathcal{X} \]

\[ \hat{x} \]

\[ z \]

\[ \mu_z|x \]

\[ \Sigma_z|x \]

Slide credit: CS231n
Probabilistic version: variational autoencoder

Sample \( z \) from
\[ z \mid x \sim \mathcal{N}(\mu_z \mid x, \Sigma_z \mid x) \]

Encoder network
\[ q_\phi(z \mid x) \]

Decoder network
\[ p_\theta(x \mid z) \]
Probabilistic version: variational autoencoder

Encoder network
$q_{\phi}(z|x)$

Decoder network
$p_{\theta}(x|z)$

Sample $z$ from
$z|x \sim \mathcal{N}(\mu_{z|x}, \sum_{z|x})$

Input Data
$\mathcal{X}$

$\hat{x}$
Probabilistic version: variational autoencoder

Input Data

Encoder network
$q_\phi(z|x)$

Sample $z$ from
$z|x \sim \mathcal{N}(\mu_z|x, \Sigma_z|x)$

Decoder network
$p_\theta(x|z)$

Sample $x|z$ from
$x|z \sim \mathcal{N}(\mu_x|z, \Sigma_x|z)$
Probabilistic version: variational autoencoder

Loss function

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

\[
\mathcal{L}(x^{(i)}, \theta, \phi)
\]

Sample \( x \mid z \sim \mathcal{N}(\mu_x \mid z, \Sigma_x \mid z) \)

Decoder network

\( p_{\theta}(x \mid z) \)

\( \mu_x \mid z \)

\( \Sigma_x \mid z \)

Sample \( z \mid x \sim \mathcal{N}(\mu_z \mid x, \Sigma_z \mid x) \)

Encoder network

\( q_{\phi}(z \mid x) \)

\( \mu_z \mid x \)

\( \Sigma_z \mid x \)

Input Data

\( x \)
Probabilistic version: variational autoencoder

Loss function

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

$$\mathcal{L}(x^{(i)}, \theta, \phi)$$

Maximize likelihood of original input being reconstructed

Sample $x|z$ from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$

Decoder network

$p_\theta(x|z)$

$\mu_{x|z}$

$\Sigma_{x|z}$

Sample $z$ from $z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$

Encoder network

$q_\phi(z|x)$

$\mu_{z|x}$

$\Sigma_{z|x}$

Input Data

$x$

$\hat{x}$

$z$

Slide credit: CS231n
Probabilistic version: variational autoencoder

Loss function

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

$$
\mathcal{L}(x^{(i)}, \theta, \phi)
$$

Maximize likelihood of original input being reconstructed

Sample x|z from

$$
x | z \sim \mathcal{N}(\mu_x|z, \Sigma_x|z)
$$

Decoder network

$$
p_\theta(x | z)
$$

$$
\mu_x | z
$$

$$
\Sigma_x | z
$$

Sample z from

$$
z | x \sim \mathcal{N}(\mu_z|x, \Sigma_z|x)
$$

Encoder network

$$
q_\phi(z | x)
$$

$$
\mu_z | x
$$

$$
\Sigma_z | x
$$

Make output distribution of encoder close to a prior

$$
\hat{x}
$$

Input Data

$$
\mathcal{X}
$$

Slide credit: CS231n
Since variational autoencoders learn distribution of the data, can also be used to generate new (synthetic) data

Use decoder network. Now sample $z$ from prior!

$z \sim \mathcal{N}(0, I)$

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014
Since variational autoencoders learn distribution of the data, can also be used to generate new (synthetic) data.

Use decoder network. Now sample z from prior!

\[ \hat{x} \]

Sample \( x|z \) from
\[ x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z}) \]

Decoder network
\[ p_\theta(x|z) \]

Sample z from
\[ z \sim \mathcal{N}(0, I) \]

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014
Since variational autoencoders learn distribution of the data, can also be used to generate new (synthetic) data.

Use decoder network. Now sample $z$ from prior!

Sample $x|z$ from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$

Decoder network $p_\theta(x|z)$

Sample $z$ from $z \sim \mathcal{N}(0, I)$

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014
Since variational autoencoders learn distribution of the data, can also be used to generate new (synthetic) data.

Different dimensions of $z$ encode interpretable factors of variation.

Degree of smile

Vary $z_1$

Vary $z_2$

Head pose

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014
Another approach for learning to generate data: generative adversarial networks (GANs)

Motivation: Want to sample (generate data) from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

Slide credit: CS231n
Another approach for learning to generate data: generative adversarial networks (GANs)

Motivation: Want to sample (generate data) from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?
A: A neural network!

Input: Random noise
Output: Sample from training distribution

Slide credit: CS231n
Another approach for learning to generate data: generative adversarial networks (GANs)

Motivation: Want to sample (generate data) from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

A: A neural network!

If goal is generating high quality samples, most current state-of-the-art approaches based on this...
Training GANs: Two-player game

**Generator network**: try to fool the discriminator by generating real-looking images

**Discriminator network**: try to distinguish between real and fake images

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014
Training GANs: Two-player game

**Generator network**: try to fool the discriminator by generating real-looking images

**Discriminator network**: try to distinguish between real and fake images

Train jointly in **minimax game**

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$
Training GANs: Two-player game

**Generator network**: try to fool the discriminator by generating real-looking images

**Discriminator network**: try to distinguish between real and fake images

Train jointly in **minimax game**

Minimax objective function:

\[
\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]
\]

- Discriminator outputs likelihood in (0,1) that image is real
- Discriminator output for real data \(x\)
- Discriminator output for generated fake data \(G(z)\)

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Slide credit: CS231n

Serena Yeung  BIODS 220: AI in Healthcare
Training GANs: Two-player game

**Generator network**: try to fool the discriminator by generating real-looking images

**Discriminator network**: try to distinguish between real and fake images

Train jointly in **minimax game**

Minimax objective function:

\[
\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]
\]

- **Discriminator** ($\theta_d$) wants to **maximize objective** such that $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake)
- **Generator** ($\theta_g$) wants to **minimize objective** such that $D(G(z))$ is close to 1 (discriminator is fooled into thinking generated $G(z)$ is real)

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014
Training GANs: Two-player game

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

   $$\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

   $$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z)))$$
Training GANs: Two-player game

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

   $$\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **In practice: Gradient ascent** on generator, different objective

   $$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$
Training GANs: Two-player game

Minimax objective function:
\[
\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]
\]

Alternate between:
1. **Gradient ascent** on discriminator
\[
\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]
\]

2. **In practice**: Gradient ascent on generator, different objective
\[
\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))
\]

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.

Slide credit: CS231n
Training GANs: Two-player game

Minimax objective function:

\[
\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]
\]

Alternate between:

1. **Gradient ascent** on discriminator

\[
\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]
\]

2. **In practice: Gradient ascent** on generator, different objective

\[
\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log (D_{\theta_d}(G_{\theta_g}(z)))
\]

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong. 

Same objective of fooling discriminator, but this objective has some nice properties that make optimization work better in practice.

Slide credit: CS231n
Training GANs: Two-player game

Minimax objective function:

\[
\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]
\]

Alternate between:
1. **Gradient ascent** on discriminator

\[
\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]
\]

2. **In practice**: Gradient ascent on generator, different objective

\[
\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))
\]

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong. 

Same objective of fooling discriminator, but this objective has some nice properties that make optimization work better in practice.

Aside: Jointly training two networks is challenging, can be unstable. Lots of active research to improve GAN training.
Training GANs: Two-player game

Putting it together: GAN training algorithm

for number of training iterations do
    for k steps do
        • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
        • Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
        • Update the discriminator by ascending its stochastic gradient:
          $$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$
    end for
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Update the generator by ascending its stochastic gradient (improved objective):
      $$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$
end for
Training GANs: Two-player game

Putting it together: GAN training algorithm

\[
\text{for number of training iterations do} \\
\quad \text{for } k \text{ steps do} \\
\quad \quad \bullet \text{Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z). \\
\quad \quad \bullet \text{Sample minibatch of } m \text{ examples } \{x^{(1)}, \ldots, x^{(m)}\} \text{ from data generating distribution } p_{data}(x). \\
\quad \quad \bullet \text{Update the discriminator by ascending its stochastic gradient:} \\
\quad \quad \quad \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right] \\
\quad \quad \text{end for} \\
\quad \bullet \text{Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z). \\
\quad \bullet \text{Update the generator by ascending its stochastic gradient (improved objective):} \\
\quad \quad \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \\
\quad \text{end for}
\]

Some find \(k=1\) more stable, others use \(k > 1\), no best rule.

More recent GAN variants alleviate this problem, better stability!
Training GANs: Two-player game

**Generator network**: try to fool the discriminator by generating real-looking images

** Discriminator network**: try to distinguish between real and fake images

---

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

---

Slide credit: CS231n
Example: GAN-based medical image synthesis

Liver lesions of different types (Frid-Adar 2018)

Dermatology lesions (Ghorbani 2019)

Brain MRIs with lesions (Han 2018)

Can be used for data augmentation!
A third paradigm of learning: reinforcement learning

Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals

**Goal**: Learn how to take actions in order to maximize reward

Atari games figure copyright Volodymyr Mnih et al., 2013. Reproduced with permission.
Reinforcement learning

Agent

Environment

Slide credit: CS231n
Reinforcement learning

State $s_t$ → Agent → Environment

Slide credit: CS231n
Reinforcement learning

State $s_t$  \rightarrow Agent  \rightarrow Environment  \rightarrow Action $a_t$
Reinforcement learning

Agent

Environment

State $s_t$

Reward $r_t$

Action $a_t$
Reinforcement learning

Agent

State $s_t$

Reward $r_t$

Next state $s_{t+1}$

Environment

Action $a_t$
Q-learning (one class of RL methods)

Learn a function (called Q-function) to estimate the expected future reward from taking a particular action from any given state:

\[ Q(s, a; \theta) \]

function parameters (weights)
Q-learning (one class of RL methods)

Learn a function (called Q-function) to estimate the expected future reward from taking a particular action from any given state:

\[ Q(s, a; \theta) \]

If the function is a deep neural network => **deep q-learning**!
Famous example: playing Atari games

**Objective:** Complete the game with the highest score

**State:** Raw pixel inputs of the game state

**Action:** Game controls e.g. Left, Right, Up, Down

**Reward:** Score increase/decrease at each time step
Q-network architecture

$$Q(s, a; \theta)$$: neural network with weights $\theta$

- Current state $s_t$: 84x84x4 stack of last 4 frames (after RGB->grayscale conversion, downsampling, and cropping)

[Slide credit: CS231n]
Policy gradients (another class of RL methods)

What is a problem with Q-learning?
The Q-function can be very complicated!

Example: a robot grasping an object has a very high-dimensional state => hard to learn exact value of every (state, action) pair
Policy gradients

What is a problem with Q-learning?
The Q-function can be very complicated!

Example: a robot grasping an object has a very high-dimensional state => hard to learn exact value of every (state, action) pair

But the policy can be much simpler: just close your hand
Can we learn a policy directly, e.g. finding the best policy from a collection of policies?
Policy gradients

Formally, let’s define a class of parameterized policies: \( \Pi = \{ \pi_\theta, \theta \in \mathbb{R}^m \} \)

For each policy, define its value:

\[
J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid \pi_\theta \right]
\]
Formally, let’s define a class of parameterized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]$$

We want to find the optimal policy $\theta^* = \arg \max_\theta J(\theta)$

How can we do this?
Policy gradients

Formally, let’s define a class of parameterized policies: \( \Pi = \{ \pi_\theta, \theta \in \mathbb{R}^m \} \)

For each policy, define its value:

\[
J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]
\]

We want to find the optimal policy \( \theta^* = \arg \max_\theta J(\theta) \)

How can we do this?

Gradient ascent on policy parameters!
Example: Raghu et al. 2017

Learned a Q-learning based policy to take treatment actions for sepsis patients, using the MIMIC dataset

5x5 possible policy actions at any timestep

Next time

- Guest lecture on research in deep learning for genomics, by Anshul Kundaje