Lecture 12: Unsupervised and Reinforcement Learning
Announcements

- Project milestone due Friday 10/30
- Project milestone presentations next Monday 11/2 in-class
  - See upcoming Piazza post for details
  - Please show up at the beginning of the class time, we will share presentation order at that time
- We want to hear how things are going for you in the class, and your feedback! A survey was released on Piazza, please fill this out.
Supervised learning

Data: \((x, y)\)

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

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

Examples: Classification, regression, semantic segmentation, object detection, instance segmentation
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.
Unsupervised representation learning: autoencoders

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

Autoencoders
Unsupervised representation learning: autoencoders

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

Autoencoders

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

(Feature representation)
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

![Diagram showing input data x, encoder, and features z.](image)
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

Input data \( x \) → Encoder → Features \( z \)
Representation learning: autoencoders

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

\[ x \]
Input data

\[ z \]
Features

\[ \text{Encoder} \]

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

Representation learning: autoencoders
Representation learning: autoencoders

How to learn this feature representation?

Input data \( \mathbf{x} \)

Features \( \mathbf{z} \)

Encoder
Representation learning: autoencoders

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

Diagram:
- Input data \( x \)
- Features \( z \)
- Encoded \( \hat{x} \)
- Reconstructed input data

Serena Yeung
BIODS 220: AI in Healthcare
Representation learning: autoencoders

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)

![Diagram of autoencoder process]

Input data $\mathbf{x}$

Encoder

Features $\mathbf{z}$

Decoder

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

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

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

Input data
Features
Reconstructed input data
Reconstructed data
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

Decoded input data

Reconstructed data

Serena Yeung  BIODS 220: AI in Healthcare

Lecture 12 - 17
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

Serena Yeung
BIODS 220: AI in Healthcare

18 Lecture 12 - 18
Representation learning: autoencoders

Encoder

Input data $x$

Features $z$

Decoder

Reconstructed input data $\hat{x}$

After training, throw away decoder
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 \textbf{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


Autoencoder for each code-based modality (e.g. medication, treatment, diagnosis), and signal time-series (e.g. heart rate)
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

Aside: self-supervised learning

- Also learns representations without external (e.g., manually provided) labels, but instead using labels generated from inherent structure in the data
- Remember BERT training

Aside: self-supervised learning

- Also learns representations without external (e.g., manually provided) labels, but instead using labels generated from inherent structure in the data
- Remember BERT training

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

Reconstructed data

Encoder
Decoder
Features
Input data
Reconstructed input data
Probabilistic version: variational autoencoder

Input Data

Encoder network

\( \mathcal{Z} \)

Decoder network

\( \hat{x} \)
Probabilistic version: variational autoencoder

- Encoder network: $q_\phi(z|x)$
- Decoder network: $p_\theta(x|z)$
Probabilistic version: variational autoencoder

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

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

Input Data
$x$

$z$

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

Decoder network

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

Sample \( z \) from

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

Encoder network

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

Input Data

\[ x \]

[Diagram of variational autoencoder with nodes labeled for encoder and decoder networks, and probabilistic relationships.]
Probabilistic version: variational autoencoder

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

Encoder network:
\[ q_\phi(z|x) \]

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

Input Data:
\[ x \]
Probabilistic version: variational autoencoder

Encoder network

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

\[ \mu_{z|x} \]

\[ \Sigma_{z|x} \]

Sample z from

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

Decoder network

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

\[ \mu_{x|z} \]

\[ \Sigma_{x|z} \]

Sample x from

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

Input Data

\[ x \]

Sample z from

\[ z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x}) \]
Probabilistic version: variational autoencoder

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

Input Data

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

\[ \mu_{z \| x} \]
\[ \Sigma_{z \| x} \]

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

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

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

\[ \mu_{x \| z} \]
\[ \Sigma_{x \| z} \]

\[ \hat{x} \]
Probabilistic version: variational autoencoder

**Loss function**

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

**Encoder network**

$q_\phi(z | x)$

**Decoder network**

$p_\theta(x | z)$

Maximize likelihood of original input being reconstructed

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

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

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

\[
\mu_{x|z} \quad \Sigma_{x|z}
\]

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

Encoder network

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

\[
\mu_{z|x} \quad \Sigma_{z|x}
\]

Make output distribution of encoder close to a prior

Input Data

\( x \)
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!

\[
\hat{x} \\
\text{Sample } x|z \text{ from } x|z \sim \mathcal{N}(\mu_x|z, \Sigma_x|z) \\
\mu_x|z \quad \Sigma_x|z \\
\text{Decoder network } p_\theta(x|z) \\
\text{Sample } z \text{ from } z \sim \mathcal{N}(0, I)
\]

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

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!

\[
\hat{x} \\
\begin{align*}
\text{Sample } x|z \text{ from } & \quad x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z}) \\
\mu_{x|z} & \\
\Sigma_{x|z} &
\end{align*}
\]

Decoder network

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

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

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Data manifold for 2-d \( z \)

Vary \( z_1 \)

Vary \( z_2 \)
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.

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?
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
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

---

Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.
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 output** for real data $x$
- **Discriminator output for generated fake data** $G(z)$

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:

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

- 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

   $$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z)))$$

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

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

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

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

$$\max_{\theta_g} E_{z \sim p(z)} \log (D_{\theta_d}(G_{\theta_g}(z)))$$

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

Minimax objective function:

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

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.

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. **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.
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

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

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

---

**Fake Images** (from generator) → **Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.**

**Real Images** (from training set) →

**Random noise** → **z** → **Generator Network**

**Discriminator Network**

Real or Fake

After training, use generator network to generate new images
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
Reinforcement learning

State $s_t$

Agent

Environment
Reinforcement learning

Agent

State $s_t$

Environment

Action $a_t$
Reinforcement learning

Agent

Environment

State $s_t$

Reward $r_t$

Action $a_t$
Reinforcement learning

- Agent
- Environment
- State $s_t$
- Reward $r_t$
- Next state $s_{t+1}$
- 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:

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

function parameters (weights)

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

Figures copyright Volodymyr Mnih et al., 2013. Reproduced with permission.
Q-network architecture

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

- FC-256
- 32 4x4 conv, stride 2
- 16 8x8 conv, stride 4
- FC-4 (Q-values)

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

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

Output expected future reward from taking each of the 4 possible actions

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

[Serena Yeung; BIODS 220: AI in Healthcare; Lecture 12 - 69]

[Mnih et al. NIPS Workshop 2013; Nature 2015]
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:

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

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

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

How can we do this?
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

- Your milestone presentations!