CSC311
Lecture Notes

Yuchen Wang

April 2, 2020

Contents

CS GRAD SCHOOL ADMISSION CONSULTING

Yuchen Wang

MSCS@Stanford, SWE@Microsoft

www.yuchenwyc.com

Do you want offers from top CS grad schools?

My team offers

- Personal statements editing
- Resume editing
- Recommendation letter strategies
- Grad school/program list suggestion
- Application timeline planning
- 1:1 mentorship with top school Masters' and PhD graduates

GET STARTED

yuchenw@stanford.edu

1 Lecture 9 - k-Means and EM Algorithm

1.1 Latent variable models

Assumption: data depends on some latent variables that are never observed.
Example: PCA (Even though data is very high dimensional, it can be well represented in low dimensions.)

1.2 Clustering

Grouping data points into clusters, with no observed labels, is called clustering.
Samples within a cluster are similar to each other, and samples in different clusters are dissimilar. It is an unsupervised technique. Such a distribution is multimodal, since it has multiple modes, or regions of high probability mass.

Example Clustering machine learning papers based on topic (deep learning, Bayesian models, etc)

1.3 Clustering Problem

Assumptions

1. The data $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$ lives in a Euclidean space, $\mathbf{x}^{(n)} \in \mathbb{R}^D$
2. Each data point belongs to one of K clusters
3. The data points from the same cluster are similar (i.e. close in Euclidean distance)

1.4 K-means

Objective Find cluster centers $\{\mathbf{m}_k\}_{k=1}^K$ and assignments $\{\mathbf{r}^{(n)}\}_{n=1}^N$ to minimize the sum of squared distances of data points $\{\mathbf{x}^{(n)}\}$ to their assigned cluster centers.

1. Data sample $n = 1, \ldots, N : \mathbf{x}^{(n)} \in \mathbb{R}^D$ (observed)
2. Cluster center $k = 1, \ldots, K : \mathbf{m}_k \in \mathbb{R}^D$ (not observed)
3. Responsibilities: Cluster assignment for sample n: $\mathbf{r}^{(n)} \in \mathbb{R}^K$ 1-of-K encoding (not observed)

Objective Mathematically

$$\begin{array}{l} \min_{\{\mathbf{m}_k\},\{\mathbf{r}^{(n)}\}} J\left(\{\mathbf{m}_k\}, \{\mathbf{r}^{(n)}\}\right) = \min_{\{\mathbf{m}_k\},\{\mathbf{r}^{(n)}\}} \sum_{n=1}^N \sum_{k=1}^K r_k^{(n)} \left\|\mathbf{m}_k - \mathbf{x}^{(n)}\right\|^2 \\ \text{where } r_k^{(n)} = \mathbb{I}\left[\mathbf{x}^{(n)} \text{ is assigned to cluster } k\right], \text{ i.e., } \mathbf{r}^{(n)} = [0, \ldots, 1, \ldots, 0]^\top \end{array}$$

Notes Finding an optimal solution is an NP-hard problem.

1.4.1 K-means Algorithm

Initialization: Set K cluster means $\mathbf{m}_1, \ldots, \mathbf{m}_K$ to random values
Repeat until convergence (until assignments do not change):
Assignment: Optimize J w.r.t. $\{\mathbf{r}\}$: Each data point $\mathbf{x}^{(n)}$ assigned to nearest center

$$\hat{k}^{(n)} = \underset{k}{argmin} ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$$

and **Responsibilities** (1-hot encoding)

$$r_k^{(n)} = \mathbb{I}[\hat{k}^{(n)} = k]$$

for $k = 1, \ldots, K$
Refitting: Optimize J w.r.t. $\{\mathbf{m}\}$: Each center is set to mean of data assigned to it

$$\mathbf{m}_k = \frac{\sum_k r_k^{(n)} \mathbf{x}^{(n)}}{\sum_n r_k^{(n)}}$$

Performance

1. K-means algorithm reduces the cost at each iteration.
2. The objective J is non-convex, so coordinate descent on J is not guaranteed to converge to the global minimum (solution: we could try many random starting points)

1.5 Soft K-means

One cluster may have a responsibility of .7 for a datapoint and another may have a responsibility of .3. This allows a cluster to use more information about the data in the refitting step.

1.5.1 Soft K-means Algorithm

Initialization: Set K cluster means $\mathbf{m}_1, \ldots, \mathbf{m}_K$ to random values
Repeat until convergence (until assignments do not change):
Assignment: Each data point n given soft "degree of assignment" to each cluster mean k, based on responsibilities

$$r_k^{(n)} = \frac{\exp[-\beta||\mathbf{m}_k - \mathbf{x}^{(n)}||^2]}{\sum_j \exp[-\beta||\mathbf{m}_j - \mathbf{x}^{(n)}||^2}]$$

where β is a hyperparameter.

$$\implies \mathbf{r}^{(n)} = softmax(-\beta\{\mathbf{m}_k - \mathbf{x}^{(n)}||^2\}_{k=1}^K)$$

Refitting: Model parameters, means, are adjusted to match sample means of datapoints they are responsible for:

$$\mathbf{m}_k = \frac{\sum_k r_k^{(n)} \mathbf{x}^{(n)}}{\sum_n r_k^{(n)}}$$

Remarks As $\beta \to \infty$, soft k-Means converges to hard k-Means.

1.6 A Generative View of Clustering

Imagine that the data was produced by a generative model, then adjust the model parameters using MLE, i.e., maximize the probability that it would product exactly the data we observed. Assume a datapoint $\mathbf{x}$ is generated as follow:

1. Choose a cluster z from $\{1, \dots, K\}$ such that $p(z = k) = \pi_k$
2. Given z, sample $\mathbf{x}$ from a Gaussian distribution $\mathcal{N}(\mathbf{x}|\mu_z, I)$

Can also be written:

$$p(z = k) = \pi_k$$

$$p(\mathbf{x}|z = k) = \mathcal{N}(\mathbf{x}|\mu_k, I)$$

This defines joint distribution $p(z, \mathbf{x}) = p(z)p(\mathbf{x}|z)$ with parameters $\{\pi_k, \mu_k\}_{k=1}^{K}$, and the marginal of $\mathbf{x}$ is given by $p(\mathbf{x}) = \sum_z p(z, x)$.
$p(z = k|\mathbf{x})$ can be computed using Bayes rule.

$$p(z = k|\mathbf{x}) = \frac{p(z = k)p(\mathbf{x}|z = k)}{p(\mathbf{x})} \tag{1.1}$$

$$= \frac{p(z = k)p(\mathbf{x}|z = k)}{\sum_{j=1}^{K} p(z = j)p(\mathbf{x}|z = j)} \tag{1.2}$$

$$= \frac{\pi_k \mathcal{N}\left(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{I}\right)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\mathbf{x}|\boldsymbol{\mu}_j, \mathbf{I}\right)} \tag{1.3}$$

This tells us the probability $\mathbf{x}$ came from the kth cluster

Choose π_k and μ_k Choose parameters to maximize likelihood of observed data. Given data $\mathcal{D} = \{\mathbf{x}^{(n)}\}_{n=1}^{N}$, choose parameters to maximize:

$$\log p(\mathcal{D}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)})$$

Find $p(\mathbf{x})$ by marginalizing out z:

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(z = k, \mathbf{x}) = \sum_{k=1}^{K} p(z = k)p(\mathbf{x}|z = k) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\mu_k, I)$$

which is an example of Gaussian Mixture Model (GMM), and π_k are known as the mixing coefficients.

Note If we allow arbitrary covariance matrices, GMMs are universal approximators of densities (if you have enough Gaussians). Even diagonal GMMs are universal approximators.

Maximum Likelihood objective

$$\log p(\mathcal{D}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)}) = \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}^{(n)} | \mu_k, I) \right)$$

1.7 Expectation-Maximization algorithm

1. **E-step**: Compute the posterior probabilities $r_k^{(n)} = p(z^{(n)} = k | \mathbf{x}^{(n)})$ given our current model - i.e. How much do we think a cluster is responsible for generating a datapoint.
2. **M-step**: Use the equations on the last slide to update the parameters, assuming $r_k^{(n)}$ are held fixed- change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for.

Mathematically,

1. **E-step**: Evaluate the responsibilities $r_k^{(n)}$ given current parameters

$$r_k^{(n)} = p(z^{(n)} = k | \mathbf{x}^{(n)}) = \frac{\hat{\pi}_k \mathcal{N}(\mathbf{x}^{(n)} | \hat{\mu}_k, I)}{\sum_{j=1}^{K} \hat{\pi}_j \mathcal{N}(\mathbf{x}^{(n)} | \hat{\mu}_j, I)}$$

2. **M-step**: Re-estimate the parameters given current responsibilities

$$\hat{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} r_k^{(n)} \mathbf{x}^{(n)}$$

$$\hat{\pi}_k = \frac{N_k}{N}$$

with

$$N_k = \sum_{n=1}^{N} r_k^{(n)}$$

3. Evaluate log likelihood and check for convergence

$$\log p(\mathcal{D}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)}) = \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}^{(n)} | \mu_k, I) \right)$$

In step 3, replace $\log p(\mathcal{D})$ with the expected complete data log-likelihood:

$$\sum_{n=1}^{N} \mathbb{E}_{p(z^{(n)} | \mathbf{x}^{(n)})} [\log p(z^{(n)}, \mathbf{x}^{(n)}] = \sum_{n=1}^{N} \sum_{k=1}^{K} r_k^{(n)} \left(\log \mathcal{N}(\mathbf{x}^{(n)} | \mu_k, I) + \log \pi_k \right)$$

which we maximized over parameters $\{\pi_k, \mu_k\}_k$

Notes This method can suffer from bad local optima (non-convex), as well as singularities: Arbitrarily large likelihood when a Gaussian explains a single point with variance shrinking to zero.

Remarks Suppose we sample $\mathbf{x}$ from a Gaussian distribution $\mathcal{N}(\mathbf{x}|\mu_z, \sigma^2 I)$. If $\sigma^2 \to 0$, then EM converges to soft k-Means.

2 Reinforcement Learning

Problem An agent continually interacts with the environment. How should it choose its actions so that its long-term rewards are maximized?

2.1 Formalizing Reinforcement Learning Problems

Markov Decision Process (MDP) A discounted MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$.

1. $\mathcal{S}$: State space, can be either discrete or continuous
2. $\mathcal{A} = \{a_1, \ldots, a_M\}$: Action space (finite)
3. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{M}(\mathcal{S})$: Transition probability kernel
4. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathcal{M}(\mathbb{R})$: Immediate reward distribution
5. γ: Discount factor $(0 \leq \gamma < 1)$

The agent has a **state** $s \in \mathcal{S}$ in the environment.
At every time step $t = 0, 1, \ldots,$, the agent is at state S_t. It takes an **action** A_t, moves into a new state S_{t+1} based on the dynamics of the environment and the selected action:

$$S_{t+1} \sim \mathcal{P}(\cdot|S_t, A_t)$$

It receives some **reward**:

$$R_t \sim \mathcal{R}(\cdot|S_t, A_t)$$

Policy The action selection mechanism is described by a **policy** π: A mapping from states to actions

1. **Deterministic policy**: $A_t = \pi(S_t)$
2. **Stochastic policy**: $A_t \sim \pi(\cdot|S_t)$

Reward The goal is to find a policy π s.t. **long-term rewards** of the agent is maximized. Different notions of the long-term reward:

1. **Cumulative / total reward**: $R_0 + R_1 + R_2 + \ldots$
2. **Discounted (cumulative) reward**: $R_0 + \gamma R_1 + \gamma^2 R_2 + \ldots$

The cumulation reward is problematic for continuing tasks (the agent-environment interactions does not break naturally into identifiable episodes, but goes on continually without limit) because the final time step would be $T = \infty$, and the return could easily be infinite.

Discount Factor The **discount rate** determines the present value of future rewards: a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately. If $\gamma < 1$, the infinite sum has a finite value as long as the reward sequence $\{R_k\}$ is bounded.

1. When γ is closer to 0, the agent prefers to obtain reward as soon as possible. (myopic)
2. When γ is closer to 1, the agent is willing to receive rewards in the farther future. (farsighted)

Transition Probability (or Dynamics) The **transition probability** describes the changes in the state of the agent when it chooses actions

$$\mathcal{P}(S_{t+1} = s' | S_t = s, A_t = a)$$

This model has **Markov property**: the future depends on the past only through the current state.

2.2 Approaches to RL

- **Value-based methods**: Optimize the value function
- **Policy-based methods**: Optimize the policy
- **Model-based methods**: Estimate the true, but unknown, model of environment $\mathcal{P}$ by an estimate $\hat{\mathcal{P}}$, and use $\hat{\mathcal{P}}$ to plan
- Hybrid methods

2.3 Value-Based Methods

Definition 2.1 (Value Function)**. Value function** V^π is the expected future reward, and is used to evaluate the desirability of states.
<u>State value function</u> V^π for policy π is a function defined as

$$V^\pi(s) := \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t R_t | S_0 = s \right]$$

where $R_t \sim \mathcal{R}(\cdot | S_t, A_t, S_{t+1})$.
It describes the expected discounted reward if the agent starts from state s and follows policy π.
Problem: we don't have direct access to the environment's dynamics, i.e. we don't know the transition probabilities, so this function is intractable.
<u>Action-value</u> function Q^π for policy π is

$$Q^\pi(s, a) := \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t R_t | S_0 = s, A_0 = a \right]$$

It describes the expected discounted reward if the agent starts from state s, takes action a, and afterwards follows policy π.
Relationship:

$$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s,a)$$

Goal The goal is to find a policy π that maximizes the value function.
Optimal value function:

$$Q^*(s,a) = \sup_\pi Q^\pi(s,a)$$

Given Q^*, the optimal policy can be obtained as

$$\pi^*(s) = \underset{a}{argmax} Q^*(s,a)$$

The goal of an RL agent is to find a policy π that is close to optimal: $Q^\pi \approx Q^*$.

Notes If the state space is finite, then we can use a table to represent the value function.

2.3.1 Bellman Equation

Write $R_t = R(S_t, A_t)$ and $r(s,a) := \mathbb{E}_\pi[R(s,a)]$.
The value function satisfies the following recursive relationship:

$$\begin{aligned}
Q^\pi(s,a) &= \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s, A_0 = a\right] && (2.1)\\
&= \mathbb{E}_\pi\left[R(S_0, A_0) + \sum_{t=1}^{\infty} \gamma^t R_t | S_0 = s, A_0 = a\right] && (2.2)\\
&= \mathbb{E}_\pi\left[R(s, a) + \sum_{t=1}^{\infty} \gamma^t R_t | S_0 = s, A_0 = a\right] && (2.3)\\
&= \mathbb{E}_\pi\left[R(s,a) + \gamma R(S_1, A_1) + \gamma^2 R(S_2, A_2) + \ldots | S_0 = s, A_0 = a\right] && (2.4)\\
&= \mathbb{E}_\pi\left[R(s,a) + \gamma\{R(S_1, \pi(S_1)) + \gamma R(S_2, \pi(S_2)) + \ldots\} | S_0 = s, A_0 = a\right] && (2.5)\\
&= \mathbb{E}_\pi\left[R(s,a) + \gamma \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = S_1, A_0 = \pi(S_1)]\right] && (2.6)\\
&= r(s,a) + \gamma \mathbb{E}[Q^\pi(S_1, \pi(S_1))] && (2.7)\\
&= r(s,a) + \gamma \int_S Q^\pi(s', \pi(s')) \mathcal{P}(s'|s,a)\, ds' && (2.8)
\end{aligned}$$

Or for countable state spaces:

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_S Q^\pi(s', \pi(s')) \mathcal{P}(s'|s,a)\, ds'$$

We define the **Bellman operator** $T^\theta : B(\mathcal{S} \times \mathcal{A}) \to B(\mathcal{S} \times \mathcal{A})$

$$(T^\theta Q)(s,a) := r(s,a) + \gamma \int_S Q(s', \theta(s')) \mathcal{P}(s'|s,a)\, ds'$$

Similarly, define the **Bellman optimality operator** T^*

$$(T^*Q)(s,a) := r(s,a) + \gamma \int_S \max_{a'} Q(s',a')\mathcal{P}(s'|s,a)\,ds'$$

Key observation

1. $Q^\pi \equiv T^\pi Q^\pi$
2. $Q^* \equiv T^* Q^*$

The solutions of these fixed-point equations are unique.

Proof. (for (2))

$$\begin{aligned}
(T^*Q^*)(s,a) &= r(s,a) + \gamma \int_S \max_{a'} Q^*(s',a')\mathcal{P}(s'|s,a)\,ds' && (2.9)\\
&= r(s,a) + \gamma \int_S Q^*(s',\pi^*(s))\mathcal{P}(s'|s,a)\,ds' && (2.10)\\
&= \sup_\pi r(s,a) + \gamma \int_S Q^\pi(s',\pi(s'))\mathcal{P}(s'|s,a)\,ds' && (2.11)\\
&= \sup_\pi Q^\pi(s,a) && (2.12)\\
&= Q^*(s,a) && (2.13)
\end{aligned}$$

■ ■

Notes Value-based approaches try to find a $\hat{Q}$ s.t.

$$\hat{Q} \approx T^*\hat{Q}$$

Greedy policy The **greedy policy** of $\hat{Q}$ is close to the optimal policy:

$$Q^{\pi(s;\hat{Q})} \approx Q^{\pi^*} \equiv Q^*$$

where the greedy policy of $\hat{Q}$ is defined as

$$\pi(s;\hat{Q}) = \underset{a\in\mathcal{A}}{argmax}\,\hat{Q}(s,a)$$

2.3.2 Finding the Value Function (Policy Evaluation Problem)

Given policy π, find Q^π (or V^π).
The uniqueness of the fixed-point of the Bellman operator implies that if we find a Q s.t. $T^\pi Q = Q$, then $Q = Q^\pi$. If we find a Q such that $T^*Q = Q$, then $Q = Q^*$
Assume that $\mathcal{P}$ and $r(s,a) = \mathbb{E}[\mathcal{R}(\cdot|s,a)]$.
Case 1:
If the state-action space $\mathcal{S}\times\mathcal{A}$ is finite and small, we can solve the following Linear System of Equations over Q:

$$Q(s,a) = r(s,a) + \gamma \sum_S Q(s',a)\mathcal{P}(s'|s,a)\,ds' \quad \forall (s,a) \in \mathcal{S}\times\mathcal{A}$$

We can also solve the following Nonlinear System of Equations over Q:

$$Q(s,a) = r(s,a) + \gamma \sum_{S} \max_{a' \in \mathcal{A}} Q(s',a')\mathcal{P}(s'|s,a)\, ds' \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A}$$

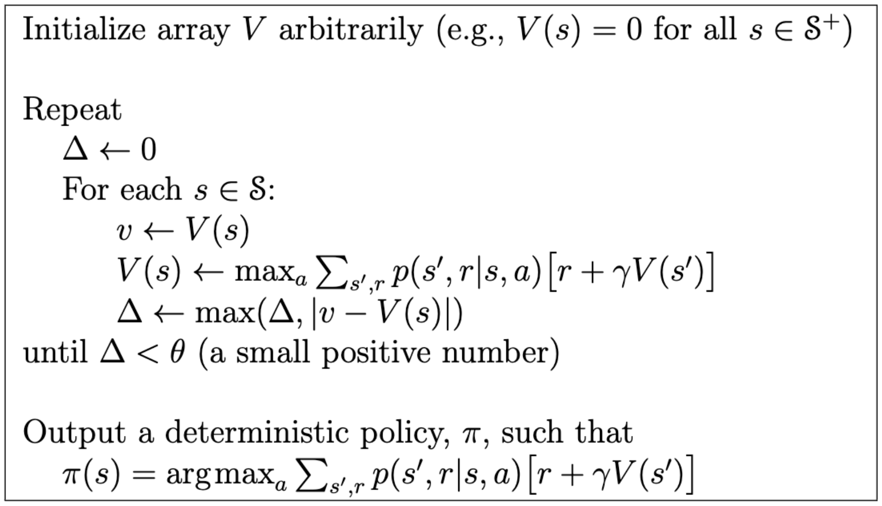
Case 2: Removing the assumption that the state-action space $\mathcal{S} \times \mathcal{A}$ is finite and small.

the Value Iteration method Assume that we know the model $\mathcal{P}$ and $\mathcal{R}$. Finding the optimal policy/value function when the model is known is sometimes called the **planning** problem.
We use a method called **Value Iteration**: Start from an initial function Q_1. For each $k = 1, 2, \ldots$, apply

$$Q_{k+1} \leftarrow T^* Q_k$$

$$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \int_{\mathcal{S}} \max_{a' \in \mathcal{A}} Q_k(s',a')\mathcal{P}(s'|s,a)\, ds'$$

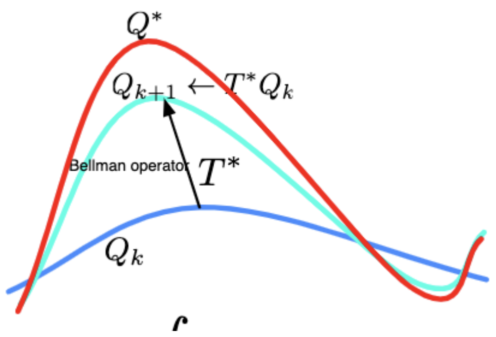
$$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \max_{a' \in \mathcal{A}} Q_k(s',a')\mathcal{P}(s'|s,a)$$



```
Initialize array $V$ arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that
    $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
```

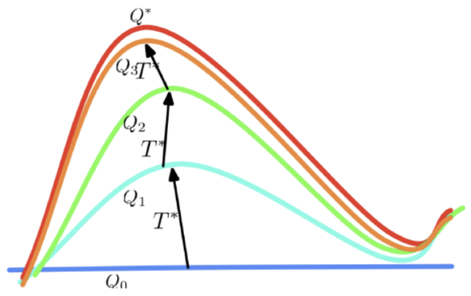


Contraction property of the Bellman operator

$$||T^*Q_1 - T^*Q_2||_\infty = ||Q_2 - Q_3||_\infty \tag{2.14}$$
$$\leq \gamma ||Q_1 - Q_2||_\infty \tag{2.15}$$

Equivalently,

$$\sup_{(s,a)\in\mathcal{S}\times\mathcal{A}} |(T^*Q_1)(s,a) - (T^*Q_2)(s,a)| \leq \sup_{(s,a)\in\mathcal{S}\times\mathcal{A}} |Q_1(s,a) - Q_2(s,a)|$$



Theorem By the Contraction property of the Bellman operator, the Value Iteration converges to the optimal value function.
Case 3: If $\mathcal{S} \subset \mathbb{R}^d$ or $|\mathcal{S}\times\mathcal{A}|$ is very large: Exact representation of the value function is infeasible for all $(s,a) \in \mathcal{S}\times\mathcal{A}$. The exact integration in the Bellman operator is challenging:

$$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \int_{\mathcal{S}} \max_{a'\in\mathcal{A}} Q_k(s',a')\mathcal{P}(s'|s,a)\, ds'$$

We often do not know the dynamics $\mathcal{P}$ and the reward function $\mathcal{R}$, so we cannot calculate the Bellman operators.

Solution: Approximate/Fitted Value Iteration (AVI) Suppose that we are given the following dataset

$$D_N = \{(S_i, A_i, R_i, S_i')\}_{i=1}^N \tag{2.16}$$
$$(S_i, A_i) \sim v \quad v \text{ is a distribution over } \mathcal{S}\times\mathcal{A} \tag{2.17}$$
$$S_i' \sim \mathcal{P}(\cdot|S_i, A_i) \tag{2.18}$$
$$R_i \sim \mathcal{R}(\cdot|S_i, A_i) \tag{2.19}$$

This is called **Batch RL** problem.
Recall that each iteration of VI computes

$$Q_{k+1} \leftarrow T^*Q_k$$

We cannot directly compute T^*Q_k, but we can use data to approximately perform one step of Value Iteration.

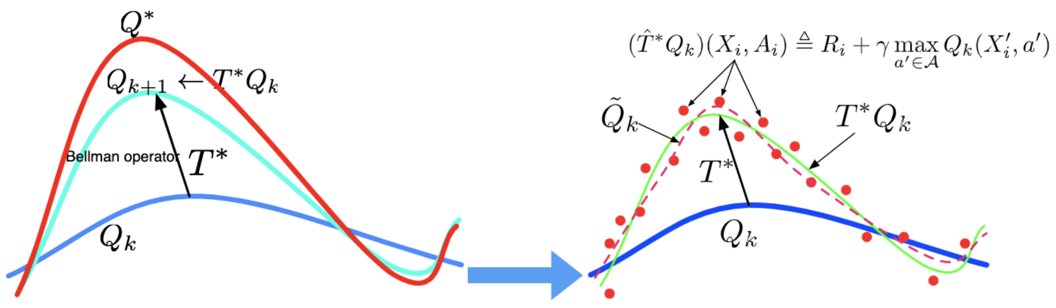
Consider a function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
We can define a random variable $t_i = R_i + \gamma \max_{a' \in \mathcal{A}} Q(S'_i, a')$
Notice that

$$\mathbb{E}[t_i|S_i, A_i] = \mathbb{E}\left[R_i + \gamma \max_{a' \in \mathcal{A}} Q(S'_i, a')|S_i, A_i\right] \tag{2.20}$$
$$= r(S_i, A_i) + \gamma \int \max_{a' \in \mathcal{A}} Q(s', a')\mathcal{P}(s'|S_i, A_i)\, ds' \tag{2.21}$$
$$= (T^*Q)(S_i, A_i) \tag{2.22}$$

So t_i has the mean of $(T^*Q)(S_i, A_i)$, and we can use it as our target:
We treat the problem of estimating Q_{k+1} as a regression problem with noisy data. We minimize the squared error:

$$Q_{k+1} \leftarrow \underset{Q \in \mathcal{F}}{argmin} \frac{1}{N} \sum_{i=1}^{N} \left| \underbrace{Q(S_i, A_i)}_{\text{current Q}} - \underbrace{(R_i + \gamma \max_{a' \in \mathcal{A}} Q_k(S'_i, a))}_{\text{target } t_i} \right|^2$$

We run this procedure K-times.


Q^*
$Q_{k+1} \leftarrow T^*Q_k$
Bellman operator
T^*
Q_k
$(\hat{T}^*Q_k)(X_i, A_i) \triangleq R_i + \gamma \max_{a' \in \mathcal{A}} Q_k(X'_i, a')$
$\tilde{Q}_k$
T^*Q_k
T^*
Q_k

The policy of the agent is selected to be the greedy policy w.r.t. the final estimate of the value function: At state $s \in \mathcal{S}$, the agent chooses

$$\pi(s; Q_k) \leftarrow \arg\max_{a \in \mathcal{A}} Q_k(s, a)$$

Choice of Estimator We have many choices for the regression method (and the function space $\mathcal{F}$).

1. Linear models: $\mathcal{F} = \{Q(s, a) = w^T \psi(s, a)\}$, where ψ is the feature mapping.
2. Decision Trees, Random Forest, etc
3. Kernel-based methods, and regularized variants
4. Deep Neural Networks. Deep Q Network (DQN) is an example of performing AVI with DNN.

Compute Q with DNN Update Q using backpropagation:

$$t \leftarrow R_i + \gamma \max_{a' \in \mathcal{A}} Q_k(S_i', a))$$
$$\theta \leftarrow \theta + \alpha(t - Q(s,a)) \frac{\partial Q}{\partial \theta}$$

Some Remarks on AVI AVI converts a value function estimation problem to a sequence of regression problems. As opposed to the conventional regression problem, the target of AVI, which is $T^* Q_k$, changes at each iteration. Usually we cannot guarantee that the solution of the regression problem Q_{k+1} is exactly equal to $T^* Q_k$. We may only have $Q_{k+1} \approx T^* Q_k$. These errors might accumulate and may even cause divergence.

2.3.3 Q-Learning

Online RL The agent continually interacts with the environment, updates its knowledge of the world and its policy, with the goal of achieving as much rewards as possible.

Q-Learning with ϵ-Greedy Policy Update the estimate of the action-value function Q online and only based on (S_t, A_t, R_t, S_{t+1}), using the policy π_t to explore the environment, such that Q converges to the optimal value function Q^*.

- learning rate α, exploration parameter ϵ
- Initialize $Q(s,a)$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$

The agent starts at state S_0. For time step $t = 0, 1, \ldots$, choose A_t according to the ϵ-greedy policy

$$\pi_\epsilon(S_t; Q) = A_t \leftarrow \begin{cases} \underset{a \in \mathcal{A}}{argmax}\, Q(S_t, a) & \text{with probability } 1 - \epsilon \\ \text{Uniformly random action in } \mathcal{A} & \text{with probability } \epsilon \end{cases}$$

Take action A_t in the environment.
The state of the agent changes from S_t to $S_{t+1} \sim \mathcal{P}(\cdot | S_t, A_t)$.
Observe S_{t+1} and R_t.
Update the action-value function at state-action (S_t, A_t):

$$Q(S_t, A_t) \leftarrow \; Q(S_t, A_t) + \alpha \left[R_t + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') - Q(S_t, A_t) \right]$$

Remarks The ϵ-greedy policy ensures that most of the time the agent exploits its incomplete knowledge of the world by choose the best action, but occasionally it explores other actions. One can show that under certain conditions, Q-Learning indeed converges to the optimal action-value function Q^* for finite state-action spaces.

2.4 Policy-Based Methods

Assume a fully observable environment, i.e. S_t can be observed directly for each t.

Definition 2.2 (rollout)**.** A rollout (or trajectory) τ is defined as

$$\tau := (S_0, A_0, S_1, A_1, \dots, S_T, A_T)$$

where T is a stopping time.
The probability of τ can be expanded using the chain rule of probabilities:

$$p(\tau) = p(S_0)\pi_\theta(A_0|S_0)p(S_1|S_0, A_0)\dots p(S_T|S_{T-1}, A_{T-1})\pi_\theta(A_T|S_T) \tag{2.23}$$

$$= p(S_0)\prod_{t=0}^{T}\pi_\theta(A_t|S_t)\prod_{t=1}^{T}p(S_t|S_{t-1}, A_{t-1}) \tag{2.24}$$

Definition 2.3. The return for a rollout is

$$r(\tau) := \sum_{t=0}^{T} R(S_t, A_t)$$

Goal The goal is to maximize the expected return, $R := \mathbb{E}_{p(\tau)}[r(\tau)]$.

2.4.1 REINFORCE

REINFORCE is an element algorithm for the goal of maximizing the expected return R.
Intuition: trial and error. Sample a rollout τ. If you get a high reward, try to make it more likely. If you get a low reward, try to make it less likely. Interestingly, this can be seen as stochastic gradient ascent on the expected return R.

Gradient of R Recall the derivative formula for log:

$$\frac{\partial}{\partial\theta}\log p(\tau) = \frac{\frac{\partial}{\partial\theta}p(\tau)}{p(\tau)} \implies \frac{\partial}{\partial\theta}p(\tau) = p(\tau)\frac{\partial}{\partial\theta}\log p(\tau) \tag{2.25}$$

$$\frac{\partial}{\partial\theta}\mathbb{E}_{p(\tau)}[r(\tau)] = \frac{\partial}{\partial\theta}\sum_{\tau} r(\tau)p(\tau) \tag{2.26}$$

$$= \sum_{r} r(\tau)\frac{\partial}{\partial\theta}p(\tau) \tag{2.27}$$

$$= \sum_{r} r(\tau)p(\tau)\frac{\partial}{\partial\theta}\log p(\tau) \qquad \text{(by 2.25)} \tag{2.28}$$

$$= \mathbb{E}_{p(\tau)}\left[r(\tau)\frac{\partial}{\partial\theta}\log p(\tau)\right] \tag{2.29}$$

$$\tag{2.30}$$

which can be estimated by a stochastic sample of rollouts.

The intuition is that we label each rollout by how much reward ($r(\tau)$) we get. If we get a large reward, make the rollout more likely. If you get a small reward, make it less likely.

$$\frac{\partial}{\partial\theta}\log p(\tau) = \frac{\partial}{\partial\theta}\log\left[p(S_0)\prod_{t=0}^{T}\pi_\theta(A_t|S_t)\prod_{t=1}^{T}p(S_t|S_{t-1},A_{t-1})\right] \quad \text{(by 2.24)} \tag{2.31}$$

$$= \frac{\partial}{\partial\theta}\log\prod_{t=0}^{T}\pi_\theta(A_t|S_t) \quad (\theta \text{ doesn't depend on } p) \tag{2.32}$$

$$= \sum_{t=0}^{T}\frac{\partial}{\partial\theta}\log\pi_\theta(A_t|S_t) \tag{2.33}$$

Notice that it tries to make **all** the actions more likely or less likely, depending on the reward. If a rollout happens to be good, all the actions get reinforced, even if some of them were bad.

the Algorithm Note that actions should only be reinforced based on future rewards, since they cannot possibly influence past rewards.

```
Algorithm REINFORCE:
while True do
    Sample a rollout $\tau = (S_0, A_0, S_1, A_1, \dots, S_T, A_T)$
    for $t = 0, \dots, T$ do
        $r_t(\tau) \leftarrow \sum_{k=t}^{T} r(S_k, A_k)$
        $\theta \leftarrow \theta + \alpha r_t(\tau)\frac{\partial}{\partial\theta}\log\pi_\theta(A_t|S_t)$
    end for
end while
```

Optimizing Discontinuous Objectives

To be implemented...

2.4.2 Evolution Strategies (ES)

REINFORCE can handle discontinuous dynamics and reward functions, but it requires a differentiable network since it computes $\frac{\partial}{\partial\theta}\log\pi_\theta(A_t|S_t)$.
Evolution strategies take the policy gradient idea a step further, and avoid backprop entirely. It randomizes over the choice of policy rather than over the choice of actions:
Sample a random policy from a distribution $p_\eta(\theta)$ parameterized by η and apply the policy gradient trick

$$\frac{\partial}{\partial\eta}\mathbb{E}_{\theta\sim p_\eta}[r(\tau(\theta))] = \mathbb{E}_{\theta\sim p_\eta}\left[r(\tau(\theta))\frac{\partial}{\partial\eta}\log p_\eta(\theta)\right]$$

In this way, the neural net architecture itself can be discontinuous.

```
**Algorithm Evolution Strategies**:
**Input:** Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\theta_0$
**for** $t = 0, 1, 2, \dots$ **do**
    Sample $\epsilon_1, \dots, \epsilon_n \sim \mathcal{N}(0, I)$
    Compute returns $R_i = R(\theta_t + \sigma\epsilon_i)$ for $i = 1, \dots, n$
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha\frac{1}{n\sigma}\sum_{i=1}^{n} F_i\epsilon_i$
**end for**
```

the Algorithm

2.5 Value-based v.s. Policy-based

Value-based methods	Policy-based methods
biased updates[1]	unbiased estimate of gradients
low variance updates	high variance updates
does credit assignment	doesn't do credit assignment
hard to handle many actions [2]	can handle a large space of actions [3]

Advantage of both methods: don't need to model the environment.

2.6 Model-Based Methods (in brief)

Intuition: If we know the function space $\mathcal{F}$, and can evaluate $R(s)$ at any state, then we can just simulate the interaction with the environment and pick the action with the best outcome.

[1]Since Q function is approximated

[2]Since you need to take the max. Therefore we usually use value-based methods for discrete actions

[3]Since you only need to sample one. Therefore we usually use policy-based methods for continuous actions