ECE493 Notes

August 11, 2019

1 Review of Probability Theory

1.1 Probability

1.2 Definition

- **Sample Space** (Ω): The set of all the outcomes of a random experiment.
- **Event Space** (F): A set whose elements $A \in F$ (called events) are subsets of Ω.

1.2.1 Axioms of Probability

1. $P(A) \geq 0, \forall A \in F$.
2. $P(\cup_i A_i) = \sum_i P(A_i)$ if $A_1, A_2, ...$ are disjoint events.
3. $P(\Omega) = 1$.

1.2.2 Properties of Probability

- $A \subseteq B \implies P(A) \leq P(B)$.
- $P(A \cap B) \leq \min(P(A), P(B))$.
- **Union Bound**: $P(A \cup B) \leq P(A) + P(B)$.
- **Complement**: $P(\Omega - A) = 1 - P(A)$.
- **Law of Total Probability**: $\sum_{i=1}^{k} P(A_i) = 1$ if $A_1, ..., A_k$ are disjoint events and $\cup_{i=1}^{k} A_i = \Omega$.

1.3 Conditional Probability

- The **conditional probability** of A, conditioned on B, is the probability that A occurs given that B has definitely occured.

$$\begin{aligned} P(A \mid B) &= \frac{P(A \cap B)}{P(B)} \\ &= \frac{P(B \mid A) \cdot P(A)}{P(B)} \end{aligned}$$

1.4 Chain Rule

- Let $S_1, ..., S_k$ be events, $P(S_i) > 0$.

$$\begin{aligned} &P(S_1 \cap S_2 \cap \cdots \cap S_k) \\ &= P(S_1)P(S_2 \mid S_1)P(S_3 \mid S_2 \cap S_1) \cdots P(S_k \mid S_1 \cap S_2 \cap \cdots \cap S_{k-1}) \end{aligned}$$

- If $k = 2$,
$$P(S_1 \cap S_2) = P(S_1)P(S_2 \mid S_1)$$

1.5 Independence

- Two events A and B are independent if and only if

$$\begin{aligned} P(A \cap B) &= P(A)P(B) \\ P(A|B) &= P(A) \\ P(B|A) &= P(B) \end{aligned}$$

1.5.1 Pairwise Independence

- Events $A_1, ..., A_k$ are pairwise independent if each pair is independent.

1.5.2 Conditional Independence

- Events $A_1, ..., A_k$ are conditionally independent conditioned on event B if

$$P(A_1 \cap ... \cap A_k \mid B) = P(A_1 \mid B)...P(A_k \mid B)$$

1.6 Random Variables

1.7 Definition

- A **random variable** X is a function $X : \Omega \to \mathbb{R}$.

1.8 Cumulative Distribution Functions

- A **cumulative distribution function** (CDF) is a function $F_X : \mathbb{R} \to [0, 1]$ such that $F_X(x) = P(X \leq x)$.

1.8.1 Properties of Cumulative Distribution Functions

- $0 \leq F_X(x) \leq 1$.
- $\lim_{x \to -\infty} F_X(x) = 0$.
- $\lim_{x \to +\infty} F_X(x) = 1$.
- $x \leq y \implies F_X(x) \leq F_X(y)$.

1.9 Probability Mass Functions (Discrete)

- A **probability mass function** (PMF) is a function $p_X : \Omega \to \mathbb{R}$ such that $p_X(x) = P(X = x)$.

1.9.1 Properties of Probability Mass Functions

- $0 \leq p_X(x) \leq 1$.
- $\sum_{x \in \text{Values}(X)} p_X(x) = 1$.
- $\sum_{x \in A} p_X(x) = P(X \in A)$.

1.10 Probability Density Functions (Continuous)

- A **probability density function** (PDF) is a function $f_X(x) : \Omega \to \mathbb{R}$ such that $f_X(x) = \frac{dF_X(x)}{dx}$.

1.10.1 Properties of Probability Density Functions

- $f_X(x) \geq 0$.
- $\int_{-\infty}^{\infty} f_X(x) = 1$.
- $\int_{x \in A} f_X(x)dx = P(X \in A)$.

1.11 Expectation

- Suppose that X is a discrete random variable with PMF $p_X(x)$ and $g : \mathbb{R} \to \mathbb{R}$ is an arbitrary function. The **expected value** of $g(X)$ is the following.

$$\mathbb{E}[g(X)] = \sum_{x \in \text{Values}(X)} g(x)p_X(x)$$

- Suppose that X is a continuous random variable with PDF $f_X(x)$ and $g : \mathbb{R} \to \mathbb{R}$ is an arbitrary function. The **expected value** of $g(X)$ is the following.

$$\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x)f_X(x)dx$$

1.11.1 Properties of Expectation

- $\mathbb{E}[a] = a$ for any constant $a \in \mathbb{R}$.
- $\mathbb{E}[a \cdot f(X)] = a \cdot \mathbb{E}[f(X)]$ for any constant $a \in \mathbb{R}$.
- $\mathbb{E}[f(X) + g(X)] = \mathbb{E}[f(X)] + \mathbb{E}[g(X)]$.

1.12 Variance

- The **variance** of a random variable X is a measure of how concentrated the distribution of a random variable X is around its mean.

$$\begin{aligned}\text{Var}[X] &= \mathbb{E}[(X - \mathbb{E}[X])^2] \\ &= \mathbb{E}[X^2] - \mathbb{E}[X]^2\end{aligned}$$

1.12.1 Properties of Variance

- $\text{Var}[a] = 0$ for any constant $a \in \mathbb{R}$.
- $\text{Var}[a \cdot f(x)] = a^2 \cdot \text{Var}[f(x)]$ for any constant $a \in \mathbb{R}$.

1.13 Two Random Variables

1.14 Joint and Marginal Distributions

- The **joint cumulative distribution function** of X and Y is defined by the following.

$$F_{XY}(x, y) = P(X \leq x, Y \leq y)$$

- The **marginal cumulative distribution functions** of $F_{XY}(x, y)$ is defined by the following.

$$F_X(x) = \lim_{y \to \infty} F_{XY}(x, y)$$
$$F_Y(y) = \lim_{x \to \infty} F_{XY}(x, y)$$

1.14.1 Properties of Joint Distributions

- $0 \leq F_{XY}(x, y) \leq 1$.
- $\lim_{x,y \to \infty} F_{XY}(x, y) = 1$.
- $\lim_{x,y \to -\infty} F_{XY}(x, y) = 0$.

1.15 Joint and Marginal Probability Mass Functions

- If X and Y are discrete random variables, then the **joint probability mass function** p_{XY} : $\text{Values}(X) \times \text{Values}(Y) \to [0, 1]$ is defined by the following.

$$p_{XY}(x, y) = P(X = x, Y = y)$$

- The **marginal probability mass functions** of p_{XY} are defined by the following.

$$p_X(x) = \sum_y p_{XY}(x, y)$$
$$p_Y(y) = \sum_x p_{XY}(x, y)$$

1.16 Joint and Marginal Probability Density Functions

- If X and Y are continuous random variables, then the **joint probability density function** f_{XY} : $\text{Values}(X) \times \text{Values}(Y) \to \mathbb{R}$ is defined by the following.

$$f_{XY}(x, y) = \frac{\partial^2 F_{XY}(x, y)}{\partial x \partial y}$$

- The **marginal probability density functions** of f_{XY} are defined by the following.

$$f_X(x) = \int_{-\infty}^{\infty} f_{XY}(x, y) dy$$
$$f_Y(y) = \int_{-\infty}^{\infty} f_{XY}(x, y) dx$$

1.17 Conditional Distributions

- The **conditional probability mass function** of Y given X is defined by the following.

$$p_{Y|X}(y \mid x) = \frac{p_{XY}(x, y)}{p_X(x)}$$

- The **conditional probability density function** of Y given X is defined by the following.

$$f_{Y|X}(y \mid x) = \frac{f_{XY}(x, y)}{f_X(x)}$$

1.18 Chain Rule

$$\begin{aligned}&p_{X_1,\ldots,X_n}(x_1,\ldots,x_n)\\&= p_{X_1}(x_1)p_{X_2|X_1}(x_2 \mid x_1)\ldots p_{X_n|X_1,\ldots,X_{n-1}}(x_n \mid x_1,\ldots,x_{n-1})\end{aligned}$$

1.19 Bayes' Rule

- For discrete random variables X and Y,

$$P_{Y|X}(y \mid x) = \frac{P_{XY}(x,y)}{P_X(x)} = \frac{P_{X|Y}(x \mid y)P_Y(y)}{\sum_{y' \in \text{Values}(Y)} P_{X|Y}(x|y')P_Y(y')}$$

- For continuous random variables X and Y,

$$f_{Y|X}(y \mid x) = \frac{f_{XY}(x,y)}{f_X(x)} = \frac{f_{X|Y}(x \mid y)f_Y(y)}{\int_{-\infty}^{\infty} f_{X|Y}(x \mid y')f_Y(y')dy'}$$

1.20 Independence

- For discrete random variables, $p_{XY}(x,y) = p_X(x)p_Y(y)$ for all $x \in \text{Values}(X), y \in \text{Values}(Y)$.
- For discrete random variables, $p_{Y|X}(y \mid x) = p_Y(y)$ whenever $p_X(x) \neq 0$ for all $y \in \text{Values}(Y)$.
- For continuous random variables, $f_{XY}(x,y) = f_X(x)f_Y(y)$ for all $x \in \mathbb{R}, y \in \mathbb{R}$.
- For continuous random variables, $f_{Y|X}(y \mid x) = f_Y(y)$ whenever $f_X(x) \neq 0$ for all $y \in \mathbb{R}$.

1.20.1 Independence Lemma

- If X and Y are independent, then for any subsets $A, B \subseteq \mathbb{R}$,

$$P(X \in A, Y \in B) = P(X \in A)P(Y \in B)$$

1.21 Expectation and Covariance

- Suppose that X and Y are random variables, and $g : \mathbb{R}^2 \to \mathbb{R}$ is a function of these two random variables. The **expected value** of $g(X,Y)$ is the following.

$$\mathbb{E}[g(X,Y)] = \sum_{x \in \text{Values}(X)} \sum_{y \in \text{Values}(Y)} g(x,y)p_{XY}(x,y)$$

$$\mathbb{E}[g(X,Y)] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x,y)f_{XY}(x,y)dxdy$$

- The **covariance** of two random variables X and Y is defined by the following.

$$\begin{aligned}\text{Cov}[X,Y] &= \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]\\&= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]\end{aligned}$$

1.21.1 Properties of Expectation and Covariance

- $\mathbb{E}[f(X,Y) + g(X,Y)] = \mathbb{E}[f(X,Y)] + \mathbb{E}[g(X,Y)]$.
- $\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y] + 2\text{Cov}[X,Y]$.
- If X and Y are independent, then $\text{Cov}[X,Y] = 0$.
- If X and Y are independent, then $\mathbb{E}[f(X)g(Y)] = \mathbb{E}[f(X)]\mathbb{E}[g(Y)]$.

2 Representation

- **Problem**: *How do we express a probability distribution $p(x_1, x_2, ..., x_n)$ that models some real-world phenomenon?*
  - *Naive Complexity*: $O(d^n)$
- **Solution**: *Representation with Probabilistic Graphical Models + Verifying Independence Assumptions*

2.1 Bayesian Networks (Directed Probabilistic Graphical Model)

2.2 Definition - What is a Bayesian network?

- A **Bayesian network** is a directed graph G with the following:
  - *Nodes*: A random variable x_i.
  - *Edges*: A conditional probability distribution (CPD) $p(x_i \mid x_{A_i})$ per node, specifying the probability of x_i conditioned on its parent's values.

2.3 Representation - How does a Bayesian network express a probability distribution?

1. Let p be a probability distribution.
2. A naive representation of p can be derived using the chain rule:

$$p(x_1, x_2, ..., x_n) = p(x_1)p(x_2 \mid x_1) \cdots p(x_n \mid x_{n-1}, ..., x_2, x_1)$$

3. A Bayesian network representation of p compacts the naive representation by having each factor in the right hand side depend only on a small number of **ancestor variables** x_{A_i}:

$$p(x_i \mid x_{i-1}, ..., x_2, x_1) = p(x_i \mid x_{A_i})$$

   - e.g., Approximate $p(x_5 \mid x_4, x_3, x_2, x_1)$ with $p(x_5 \mid x_{A_5})$ where $x_{A_5} = \{x_4, x_3\}$.

2.4 Space Complexity - How compact is a Bayesian network?

- Consider each of the factors $p(x_i \mid x_{A_i})$ as a **probability table**:
  - *Rows*: Values of x_i
  - *Columns*: Values of x_{A_i}
  - *Cells*: Values of $p(x_i \mid x_{A_i})$
- If each discrete random variable takes d possible values and has at most k ancestors, then each probability table has at most $O(d^{k+1})$ entries.
- **Naive Representation Space Complexity**: $O(d^n)$
- **Bayesian Networks Representation Space Complexity**: $O(n \cdot d^{k+1})$

$$\approx \text{Bayesian Networks Representation} \leq \text{Naive Representation}$$

2.5 Independence Assumptions - Why are the independence assumptions of a Bayesian network important to identify?

- A Bayesian network expresses a probability distribution p via products of smaller, local conditional probability distributions (one for each variable).
- These smaller, local conditional probability distributions introduces assumptions into the model of p that certain variables are independent.
- **Important Note**: Which independence assumptions are we exactly making by using a Bayesian network?
  - *Correctness: Are these independence assumptions correct?*
  - *Efficiency: Do these independence assumptions efficiently compact the representation?*

2.6 3-Variable Independencies in Directed Graphs - How do you identify independent variables in a 3-variable Bayesian network?

- Let $x \perp y$ indicate that variables x and y are independent.
- Let G be a Bayesian network with three nodes: A, B, and C.

2.6.1 Common Parent

- If G is of the form $A \leftarrow B \rightarrow C$,
  - If B is observed, then $A \perp C \mid B$
  - If B is unobserved, then $A \not\perp C$
- **Intuition**: B contains all the information that determines the outcomes of A and C; once it is observed, there is nothing else that affects A's and Cs' outcomes.

2.6.2 Cascade

- If G equals $A \rightarrow B \rightarrow C$,
  - If B is observed, then $A \perp C \mid B$
  - If B is unobserved, then $A \not\perp C$
- **Intuition**: B contains all the information that determines the outcomes of C; once it is observed, there is nothing else that affects C's outcomes.

2.6.3 V-Structure

- If G is $A \rightarrow C \leftarrow B$, then knowing C couples A and B.
  - If C is unobserved, then $A \perp B$
  - If C is observed, then $A \not\perp B \mid C$

2.7 n-Variable Independencies in Directed Graphs - How do you identify independent variables in a n-variable Bayesian network?

- Let $I(p)$ be the set of all independencies that hold for a probability distribution p.
- Let $I(G) = \{(X \perp Y \mid Z) : X, Y \text{ are } d\text{-sep given } Z\}$ be a set of variables that are d-separated in G.

- If the probability distribution p factorizes over G, then $I(G) \subseteq I(p)$ and G is an I-map (**independence map**) for p.
- **Important Note 1**: Thus, variables that are d-separated in G are independent in p.
- **Important Note 2**: However, a probability distribution q can factorize over G, yet have independencies that are not captured in G.
- **Important Caveat**: A Bayesian network cannot perfectly represent all probability distributions.

2.7.1 d-separation (a.k.a. Directed Separation)

- Q and W are **d-separated** when variables O are observed if they are **NOT CONNECTED** by an active path.

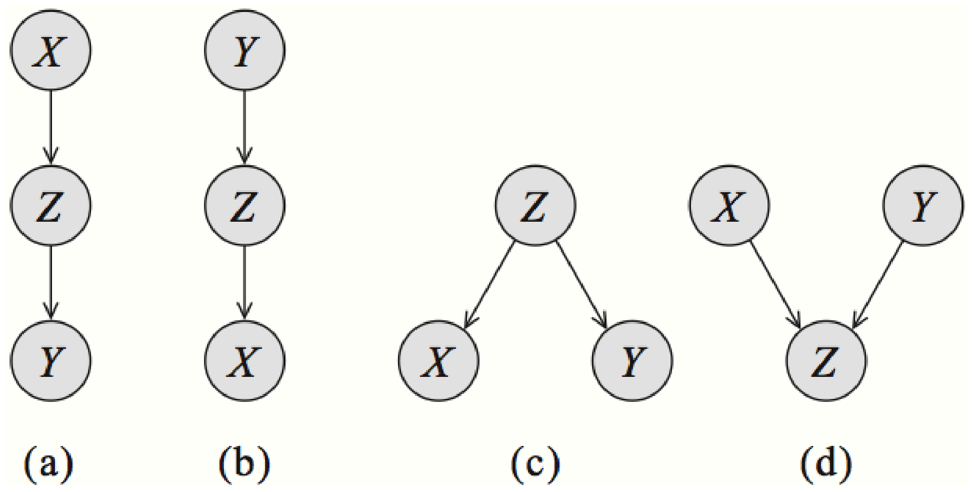
2.7.2 Active Path

- An undirected path in the Bayesian Network structure G is called **active** given observed variables O if for **EVERY CONSECUTIVE TRIPLE** of variables X, Y, Z on the path, one of the following holds:
  - **Evidential Trail**: $X \leftarrow Y \leftarrow Z$, and Y is unobserved $Y \notin O$
  - **Causal Trail**: $X \rightarrow Y \rightarrow Z$, and Y is unobserved $Y \notin O$
  - **Common Cause**: $X \leftarrow Y \rightarrow Z$, and Y is unobserved $Y \notin O$
  - **Common Effect**: $X \rightarrow Y \leftarrow Z$, and Y or any of its descendants are observed

2.8 Equivalence - When are two Bayesian networks I-equivalent?

- G_1 and G_2 are **I-equivalent**...
  - If they encode the same dependencies: $I(G_1) = I(G_2)$.
  - If they have the same skeleton and the same v-structures.
  - If the d-separation between variables is the same.

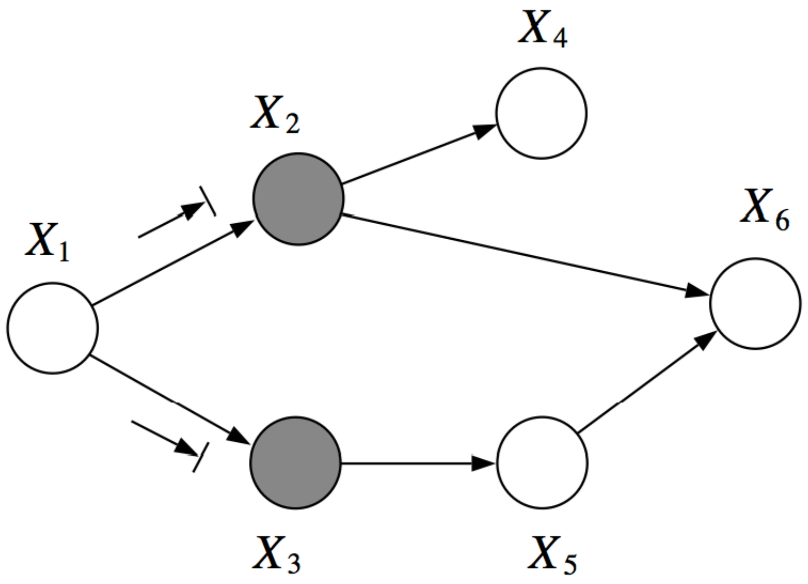
2.8.1 Skeleton



Skeleton

- A **skeleton** is an undirected graph obtained by dropping the directionality of the arrows.
  - (a) is Cascade
  - (b) is Cascade
  - (c) is Common Parent
  - (d) is V-Structure
  - (a), (b), (c), and (d) have the same skeleton.

2.9 Example Problem 1 - *d*-separation



Problem 1 - *d*-separation

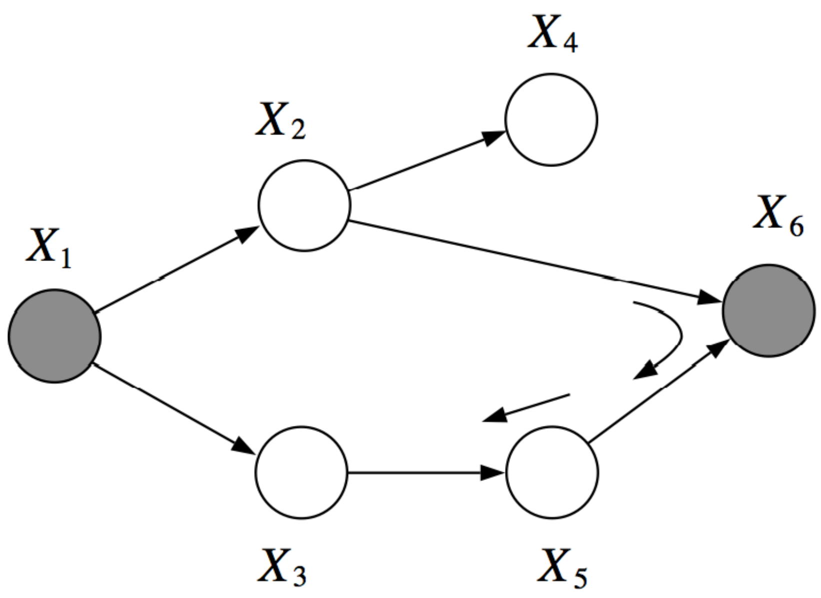
2.9.1 Question

- Are X_1 and X_6 *d*-separated given $\{X_2, X_3\}$?

2.9.2 Solution

1. **Path**: $X_1 \rightarrow X_2 \rightarrow X_6$
    1. *Consecutive Triple*: $X_1 \rightarrow X_2 \rightarrow X_6$
        - Although X_2 is observed, the *common effect* does not hold.
    2. As not all the consecutive triples hold, this path is not *active*.
2. **Path**: $X_1 \rightarrow X_3 \rightarrow X_5 \rightarrow X_6$
    1. *Consecutive Triple*: $X_1 \rightarrow X_3 \rightarrow X_5$
        - Although X_3 is observed, the *common effect* does not hold.
    2. *Consecutive Triple*: $X_3 \rightarrow X_5 \rightarrow X_6$
        - As X_5 is unobserved, the *causal trail* does hold.
    3. As not all the consecutive triples hold, this path is not *active*.
3. As there are no active paths between X_1 and X_6, they are *d*-separated given $\{X_2, X_3\}$.

2.10 Example Problem 2 - *d*-separation



Problem 2 - *d*-separation

2.10.1 Question

- Are X_2 and X_3 *d*-separated given $\{X_1, X_6\}$?

2.10.2 Solution

1. **Path**: $X_2 \leftarrow X_1 \rightarrow X_3$
   1. *Consecutive Triple*: $X_2 \leftarrow X_1 \rightarrow X_3$
      - Although X_1 is observed, the *common effect* does not hold.
   2. As not all the consecutive triples hold, this path is not *active*.
2. **Path**: $X_2 \rightarrow X_6 \leftarrow X_5 \leftarrow X_3$
   1. *Consecutive Triple*: $X_2 \rightarrow X_6 \leftarrow X_5$
      - As X_6 is observed, the *common effect* does hold.
   2. *Consecutive Triple*: $X_6 \leftarrow X_5 \leftarrow X_3$
      - As X_5 is unobserved. the *causal trail* does hold.
   3. As all the consecutive triples hold, this path is *active*.
3. As there exists an active path between X_2 and X_3, they are not *d*-separated given $\{X_1, X_6\}$.

2.11 Markov Random Fields (Undirected Probabilistic Graphical Model)

2.12 Definition - What is a Markov random field?

- A **Markov random field** is an undirected graph G with the following:

  - *Nodes*: A random variable x_i.
  - *Fully Connected Subgraphs*: An optional factor $\phi_c(x_c)$ per clique, specifying the level of coupling (**potentials**) between all the dependent variables within the clique.

- **Important Note**: >...SPECIFYING THE LEVEL OF COUPLING BETWEEN ALL THE DEPENDENT VARIABLES WITHIN THE CLIQUE...

2.13 Representation - How does a Markov random field express a probability distribution?

1. Let p be a probability distribution.
2. A Markov random field representation of p is the following:

$$p(x_1, x_2, ..., x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(x_c)$$

- Where C is the set of cliques of G.
- Where ϕ_c is a **factor** (nonnegative function) over the variables in a clique.
- Where Z is a **normalizing constant** that ensures that p sums to one.

$$Z = \sum_{x_1, x_2, ..., x_n} \prod_{c \in C} \phi_c(x_c)$$

2.14 Space Complexity - How compact is a Markov random field?

2.14.1 Factor Product

- Let A, B, and C be three disjoint sets of variables.
- Let $\phi_1(A, B)$ and $\phi_2(B, C)$ be two factors.
- Let $\phi_3(A, B, C)$ be the **factor product**.

$$\phi_3(A, B, C) = \phi_1(A, B) \cdot \phi_2(B, C)$$

  - Where the two factors are multiplied for common values of B.

2.14.2 Binary Factor Tables

- Each of the optional factors $\phi_c(x_c)$ can be expressed as a product of **binary factor tables** $\phi(X, Y)$:
  - *Rows*: Values of X
  - *Columns*: Values of Y
  - *Cells*: Values of $\phi(X, Y)$
- If each variable takes d values, each binary factor table has at most $O(d^2)$ entries.
- **Markov Random Fields Representation Space Complexity**: $O(E \cdot d^2)$
  - Where E is the number of edges in a Markov random field.

$$\approx \text{Markov Random Field Representation} \leq \text{Naive Representation}$$

2.15 Markov Random Fields vs. Bayesian Networks - What are the advantages and disadvantages of Markov random fields?

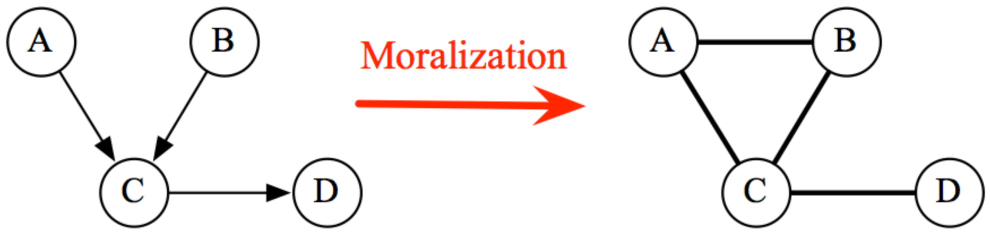
2.15.1 Advantages

- *Applicable for Variable Dependencies Without Natural Directionality*
- *Succinctly Express Dependencies Not Easily Expressible in Bayesian Networks*

2.15.2 Disadvantages

- *Cannot Express Dependencies Easily Expressible in Bayesian Networks*
  - *e.g., V-Structures*
- *Computing Normalization Constant Z Is NP-Hard*
- *Generally Require Approximation Techniques*
- *Difficult to Interpret*
- *Easier to Construct Bayesian Networks*

2.16 Moralization - What is moralization?



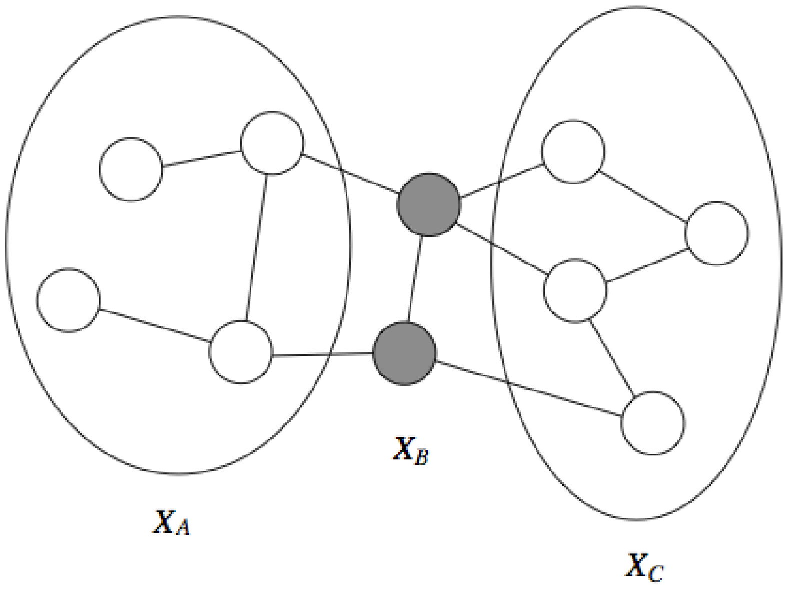
Moralization

- Bayesian networks are a special case of Markov random fields with factors corresponding to conditional probability distributions and a normalizing constant of one.
- **Moralization**: Bayesian Network $\rightarrow$ Markov Random Field
  1. Add side edges to all parents of a given node.
  2. Remove the directionality of all the edges.

2.17 n-Variable Independencies in Undirected Graphs - How do you identify independent variables in a n-variable Markov random field?

1. If variables X and Y are connected by a path of unobserved variables, then X and Y are dependent.
2. If variable X's neighbors are all observed, then X is independent of all the other variables.
3. If a set of observed variables forms a cut-set between two halves of the graph, then variables in one half are independent from ones in the other.

2.17.1 Cut-Set Variable Independencies



Cut-Set Variable Independencies

2.17.2 Markov Blanket

- The **Markov blanket** U of a variable X is the minimal set of nodes such that X is independent from the rest of the graph if U is observed.

$$X \perp (\mathcal{X} - \{X\} - U) \mid U$$

- In an undirected graph, the Markov blanket is a node's neighborhood.

2.18 Conditional Random Fields - What are conditional random fields?

2.18.1 Definition

- A **conditional random field** is a Markov random field over variables $\mathcal{X} \cup \mathcal{Y}$ which specifies a conditional distribution:

$$P(y \mid x) = \frac{1}{Z(x)} \prod_{c \in C} \phi_c(x_c, y_c)$$
$$Z(x) = \sum_{y \in \mathcal{Y}} \prod_{c \in C} \phi_c(x_c, y_c)$$

  - Where $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ are **VECTOR-VALUED** variables.
  - Where $Z(x)$ is the partition function.

- **Important Note 1**: A conditional random field results in an instantiation of a new Markov random field for each input x.
- **Important Note 2**: A conditional random field is useful for structured prediction in which the output labels are predicted considering the neighboring input samples.
  - *See Stanford CS228 - Markov Random Fields: Conditional Random Fields (OCR Example).*

2.18.2 Features

- Assume the factors $\phi_c(x_c, y_c)$ are of the following form:

$$\phi_c(x_c, y_c) = \exp(w_c^T f_c(x_c, y_c))$$

  - Where $f_c(x_c, y_c)$ can be an arbitrary set of features describing the compatibility between x_c and y_c.
  - Where w_c^T is the transposed weight matrix.

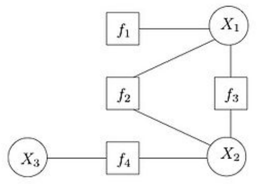
- Accordingly, $f_c(x_c, y_c)$ allows arbitrarily complex features.
  - e.g., $f(x, y_i)$ are features that depend on the entirety of input samples x.
  - e.g., $f(y_i, y_{i+1})$ are features that depend on successive pairs of output labels y.

2.19 Conditonal Random Fields vs. Markov Random Fields - Why is a conditional random field a special case of Markov random fields?

- If we were to model $p(x, y)$ using a Markov random field, then we need to fit two probability distributions to the data: $p(y \mid x)$ and $p(x)$.
  - *Remember Baye's Rule*: $p(x, y) = p(y \mid x) \cdot p(x)$
- However, if all we are interested in is predicting y given x, then modeling $p(x)$ is expensive and unnecessary.

$$\text{Prediction} \implies \text{CRF} > \text{MRF}$$

2.20 Factor Graphs - What is a factor graph? Why does a factor graph exist?



Factor Graph

- A **factor graph** is a bipartite graph where one group is the variables in the distribution being modeled, and the other group is the factors defined on these variables.
  - *Edges Between Factors and Variables*
- **Side Note**: A **bipartite graph** is a graph whose vertices are divided into two disjoint and independent sets.
  - *Set 1: Variables*
  - *Set 2: Factors*
- **Important Note**: Use a factor graph to identify what variables a factor depends on when computing probability distributions.

3 Inference

- **Problem**: *Given a probabilistic model, how do we obtain answers to relevant questions about the world?*
  - **Marginal Inference**: *What is the probability of a given variable in our model after we sum everything else out?*

$$p(y=1) = \sum_{x_1}\sum_{x_2}\cdots\sum_{x_n} p(y=1, x_1, x_2, ..., x_n)$$

    - e.g., What is the overall probability that an email is spam?
    - *Perspective*: We desire to infer the general probability of some real-world phenomenon being observed.
      - i.e., You care more about spam as a whole than specific instances of spam.
  - **Maximum A Posteriori**: *What is the most likely assignment of variables?*

$$\max_{x_1,...,x_n} p(y=1, x_1, x_2, ..., x_n)$$

    - e.g., What is the set of words such that an email has the maximum probability of being spam?
    - *Perspective*: We desire to infer the set of conditions that maximizes the probability of some real-world phenomenon being observed.
      - i.e., You care more about identifying indicators of spam than detecting spam.
  - *Naive Complexity*: NP-Hard **(DIFFICULT PROBLEM)**
- **Solution**: *Exact Inference Algorithms & Approximate Inference Algorithms*

3.1 Vairable Elimination (Exact Inference Algorithm)

3.2 Motivation - Why does the variable elimination algorithm exist?

- Let x_i be a discrete random variable that takes k possible values.
- **Problem**: *Marginal Inference*

$$p(y=1) = \sum_{x_1}\sum_{x_2}\cdots\sum_{x_n} p(y=1, x_1, x_2, ..., x_n)$$

- **Naive Solution's Time Complexity** (*Exponential*): $O(k^n)$
  - *See* *Rule of Product in Combinatorics*
- **Variable Elimination Solution's Time Complexity** (*Non-Exponential*): $O(n \cdot k^{M+1})$
  - *See Below.*

$$\therefore \text{Variable Elimination Solution} \ll \text{Naive Solution}$$

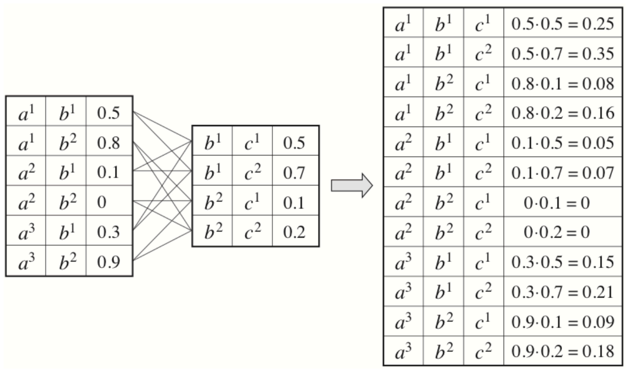
3.3 Factors - How should a probabilistic graphical model express a probability distribution?

- **Assumption**: *Probabilistic Graphical Models = Product of Factors*

$$p(x_1, ..., x_n) = \prod_{c \in C} \phi_c(x_c)$$

- **Representation**: A factor can be represented as a *multi-dimensional table* with a cell for each assignment of x_c.
- *Bayesian Networks*: ϕ is Conditional Probability Distribution
- *Markov Random Fields*: ϕ is Potentials

3.4 Factor Product - What is the product operation?



a^1	b^1	0.5
a^1	b^2	0.8
a^2	b^1	0.1
a^2	b^2	0
a^3	b^1	0.3
a^3	b^2	0.9

b^1	c^1	0.5
b^1	c^2	0.7
b^2	c^1	0.1
b^2	c^2	0.2

a^1	b^1	c^1	$0.5 \cdot 0.5 = 0.25$
a^1	b^1	c^2	$0.5 \cdot 0.7 = 0.35$
a^1	b^2	c^1	$0.8 \cdot 0.1 = 0.08$
a^1	b^2	c^2	$0.8 \cdot 0.2 = 0.16$
a^2	b^1	c^1	$0.1 \cdot 0.5 = 0.05$
a^2	b^1	c^2	$0.1 \cdot 0.7 = 0.07$
a^2	b^2	c^1	$0 \cdot 0.1 = 0$
a^2	b^2	c^2	$0 \cdot 0.2 = 0$
a^3	b^1	c^1	$0.3 \cdot 0.5 = 0.15$
a^3	b^1	c^2	$0.3 \cdot 0.7 = 0.21$
a^3	b^2	c^1	$0.9 \cdot 0.1 = 0.09$
a^3	b^2	c^2	$0.9 \cdot 0.2 = 0.18$

Figure 4.3 An example of factor product

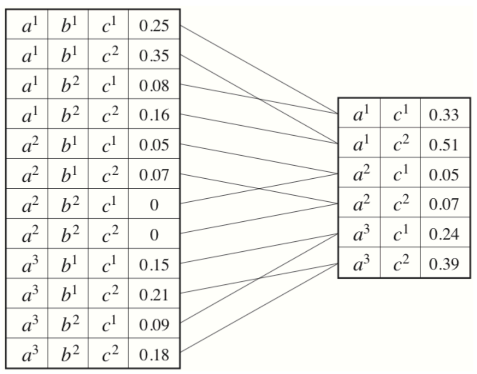
Example of Factor Product

- Let A, B, and C be three disjoint sets of variables.
- Let $\phi_1(A, B)$ and $\phi_2(B, C)$ be two factors.
- Let $\phi_3(A, B, C)$ be the **factor product**.

$$\phi_3(A, B, C) = \phi_1(A, B) \cdot \phi_2(B, C)$$

  - Where the two factors are multiplied for common values of B.

3.5 Factor Marginalization - What is the marginalization operation?



a^1	b^1	c^1	0.25
a^1	b^1	c^2	0.35
a^1	b^2	c^1	0.08
a^1	b^2	c^2	0.16
a^2	b^1	c^1	0.05
a^2	b^1	c^2	0.07
a^2	b^2	c^1	0
a^2	b^2	c^2	0
a^3	b^1	c^1	0.15
a^3	b^1	c^2	0.21
a^3	b^2	c^1	0.09
a^3	b^2	c^2	0.18

a^1	c^1	0.33
a^1	c^2	0.51
a^2	c^1	0.05
a^2	c^2	0.07
a^3	c^1	0.24
a^3	c^2	0.39

Figure 9.7 Example of factor marginalization: summing out B.

Example of Factor Marginalization

- Let A, and B be two disjoint sets of variables.

- Let $\phi(A, B)$ be a factor.
- Let $\tau(A)$ be the **factor marginalization** of B in ϕ.

$$\tau(A) = \sum_B \phi(A, B)$$

- **Important Note**: τ does not need necessarily correspond to a probability distribution.

3.6 Ordering - What is an ordering?

- An **ordering** O is the sequence of variables by which they will be eliminated.
- Although any ordering can be used, different orderings may dramatically alter the running time of the variable elimination algorithm.
- **Important Note**: *Finding Best Ordering = NP-Hard*

3.7 Algorithm - How does the variable elimination algorithm work?

- For each variable X_i (ordered according to O),
  1. Multiply all factors Φ_i containing X_i.
  2. Marginalize out X_i to obtain a new factor τ.
  3. Replace the factors Φ_i with τ.

3.8 Time Complexity - What is the time complexity of variable elimination?

- **Time Complexity**: $O(n \cdot k^{M+1})$
  - Where n is the number of variables.
  - Where M is the maximum number of dimensions of any factor τ formed during the elimination process.

3.9 Ordering Heuristics - How should you choose an ordering for variable elimination?

- **Minimum Neighbors**: Choose a variable with the fewest dependent variables.
- **Minimum Weight**: Choose variables to minimize the product of the cardinalities of its dependent variables.
- **Minimum Fill**: Choose vertices to minimize the size of the factor that will be added to the graph.

3.10 Evidence - How do you perform marginal inference given some evidence using variable elimination?

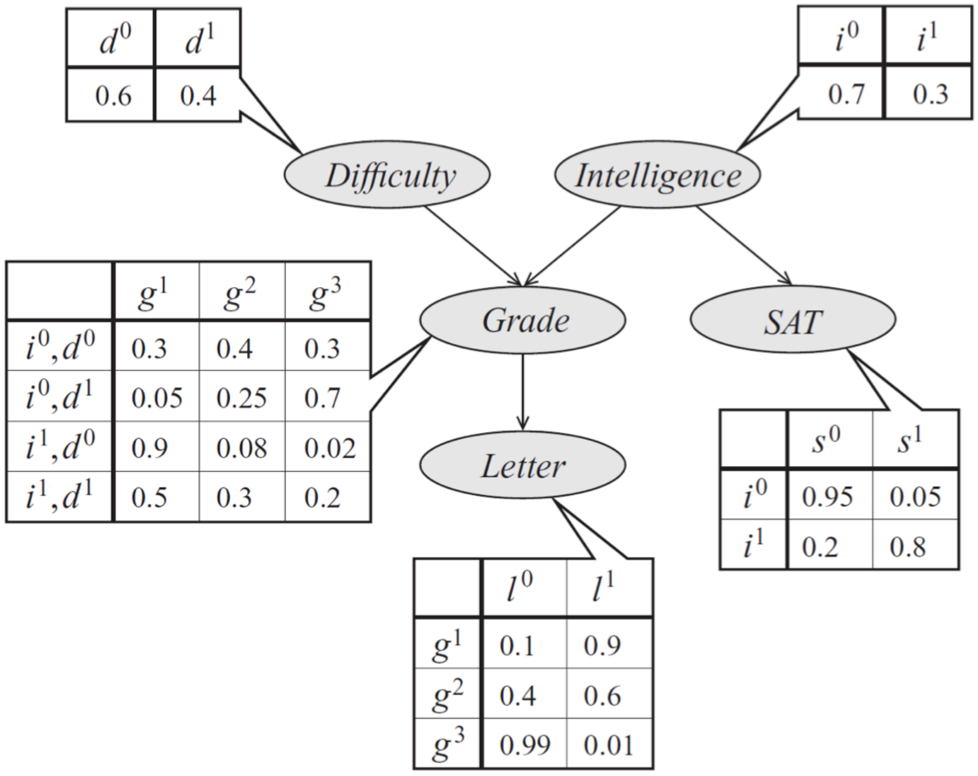
- Given a probability distribution $P(X, Y, E)$ with unobserved variables X, query variables Y, and observed evidence variables E, $P(Y \mid E = e)$ can be calculated using variable elimination.

$$P(Y \mid E = e) = \frac{P(Y, E = e)}{P(E = e)}$$

3.10.1 Variable Elimination with Evidence

1. Set every factor $\phi(X', Y', E')$ with values specified by $E = e$.
2. Compute $P(Y, E = e)$ by performing variable elimination over X.
3. Compute $P(E = e)$ by performing variable elimination over Y.

3.11 Example Problem 1 - Variable Elimination



d^0	d^1
0.6	0.4

i^0	i^1
0.7	0.3

	g^1	g^2	g^3
i^0, d^0	0.3	0.4	0.3
i^0, d^1	0.05	0.25	0.7
i^1, d^0	0.9	0.08	0.02
i^1, d^1	0.5	0.3	0.2

	s^0	s^1
i^0	0.95	0.05
i^1	0.2	0.8

	l^0	l^1
g^1	0.1	0.9
g^2	0.4	0.6
g^3	0.99	0.01

Problem 1 - Variable Elimination

- A Bayesian network that models a student's grade on an exam:
  - g is a ternary variable of the student's grade.
  - d is a binary variable of the exam's difficulty.
  - i is a binary variable of the student's intelligence.
  - l is a binary variable of the quality of a reference letter from the professor who taught the course.
  - s is a binary variable of the student's SAT score.

$$p(l, g, i, d, s) = p(l \mid g) \cdot p(s \mid i) \cdot p(i) \cdot p(g \mid i, d) \cdot p(d)$$

3.11.1 Question (Marginal Inference)

- What is the probability distribution of the quality of a reference letter from the professor who taught the course?

$$p(l) = \sum_g \sum_i \sum_d \sum_s p(l, g, i, d, s)$$

3.11.2 Solution (Variable Elimination)

1. Order the variables according to the topological sort of the Bayesian network.

$$d, i, s, g$$

2. Eliminate d with a new factor τ_1:

$$\tau_1(g, i) = \sum_d p(g \mid i, d) \cdot p(d)$$
$$p(l, g, i, s) = p(l \mid g) \cdot p(s \mid i) \cdot p(i) \cdot \tau_1(g, i)$$

3. Eliminate i with a new factor τ_2:

$$\tau_2(g, s) = \sum_i p(s \mid i) \cdot p(i) \cdot \tau_1(g, i)$$
$$p(l, g, s) = p(l \mid g) \cdot \tau_2(g, s)$$

4. Eliminate s with a new factor τ_3:

$$\tau_3(g) = \sum_s \tau_2(g, s)$$
$$p(l, g) = p(l \mid g) \cdot \tau_3(g)$$

5. Eliminate g with a new factor τ_4:

$$\tau_4(l) = \sum_g p(l \mid g) \cdot \tau_3(g)$$
$$p(l) = \tau_4(l)$$

6. Expanding τ_i:

$$p(l) = \sum_g p(l \mid g) \cdot \sum_s \sum_i p(s \mid i) \cdot p(i) \cdot \sum_d p(g \mid i, d) \cdot p(d)$$

3.11.3 Time Complexity

- **Naive Solution**: $O(k^4)$
- **Variable Elimination Solution**: $O(4 \cdot k^3)$
  - Step 2. takes $O(k^3)$ steps as the factor product $p(g \mid i, d) \cdot p(d)$ has a 3-dimensional table representation, and the factor marginalization of d can execute concurrently with the factor product.
  - Step 3. takes $O(k^3)$ steps as the factor product $p(s \mid i) \cdot p(i) \cdot \tau_1(g, i)$ has a 3-dimensional table representation, and the factor marginalization of i can execute concurrently with the factor product.
  - Step 4. takes $O(k)$ steps for the factor marginalization of s.
  - Step 5. takes $O(k^2)$ steps as the factor product $p(l \mid g) \cdot \tau_3(g)$ has a 2-dimensional table representation, and the factor marginalization of g can execute concurrently with the factor product.
  - As $O(k^3)$ is the largest step, with 4 steps, the time complexity is at most $O(4 \cdot k^3)$.
  - **Thus, with $n = 4$ and $M = 2$, the time complexity is at most $O(n \cdot k^{M+1}) = O(4 \cdot k^3)$.**

3.12 MAP Inference

3.13 Overview - What is MAP inference?

- *See Section* **??**.
- Given a probabilistic graphical model $p(x_1, ..., x_n) = \prod_{c \in C} \phi_c(x_c)$, MAP inference corresponds to the following optimization problem:

$$\max_x \log p(x) = \max_x \sum_{c \in C} \theta_c(x_c) - \log Z$$

  – Where $\theta_c(x_c) = \log \phi_c(x_c)$.

3.13.1 Derivation - Why is the MAP inference optmization problem expressed the way it is?

1. All probabilistic graphical models (as BNs and CRFs are special cases of MRFs) have the following representation:

$$p(x) = \frac{1}{Z} \prod_{c \in C} \phi_c(x_c)$$

   - Where Z is NP-Hard.

2. MAP inference desires to infer the set of conditions that maximizes the probability of some real-world phenomenon being observed.

$$\max_x p(x) = \max_x \frac{1}{Z} \prod_{c \in C} \phi_c(x_c)$$

3. As Z is expensive to calculate, maximize $\log p(x)$ instead of $p(x)$.

$$\max_x \log p(x) = \max_x \log \left[\frac{1}{Z} \prod_{c \in C} \phi_c(x_c) \right]$$

4. Simplify using logarithmic identities.

   - $\log(x \times y) = \log(x) + \log(y)$
   - $\log(x \div y) = \log(x) - \log(y)$

$$\max_x \log p(x) = \max_x \left[\sum_c \log \phi_c(x_c) - \log Z \right]$$

5. Simplify using maximum identities.

   - $\max_x(x \pm 1) = \max_x(x) \pm 1$

$$\max_x \log p(x) = \max_x \sum_c \log \phi_c(x_c) - \log Z$$

6. Let $\theta_c(x_c) = \log \phi_c(x_c)$.

$$\max_x \log p(x) = \max_x \sum_{c \in C} \theta_c(x_c) - \log Z$$

- As $\log Z$ is outside the scope of the maximization, if you desire to infer the set of conditions that maximizes the probability of some real-world phenomenon being observed, then solve the following optimization problem:

$$\arg\max_x \log p(x) = \arg\max_x \sum_{c \in C} \theta_c(x_c)$$

- **Important Note 1**: Without Z, this optimization problem suggests that MAP inference is computationally cheaper than marginal inference questions.
- **Important Note 2**: As maximization and summation both distribute over products, techniques used to solve marginal inference problems can be used to solve MAP inference problems.

3.14 Graph Cuts - How can MAP inference problems be solved using graph cuts?

- A **graph cut** of an undirected graph $G = (V, E)$ is a partition of V into two disjoint sets V_s and V_t.
- The **min-cut** problem is to find the partition V_s, V_t that minimize the cost of the graph cut.
  - The cost of a graph cut is the sum of the nonnegative costs of the edges that cross between the two partitions:

$$\text{cost}(V_s, V_t) = \sum_{v_1 \in V_s, v_2 \in V_t} \text{cost}(v_1, v_2)$$

  - **Time Complexity 1**: $O(|E||V|\log|V|)$
  - **Time Complexity 2**: $O(|V|^3)$
- A MAP inference problem can be reduced into the min-cut problem in certain restricted cases of MRFs with binary variables.

3.15 Linear Programming - How can MAP inference problems be solved using linear programming?

- An approximate approach to computing the MAP values is to use Integer Linear Programming by introducing:
  - An indicator variable per variable in the PGM.
  - An indicator variable per edge/clique in the PGM.
  - Constraints on consistent values in cliques.

3.16 Local Search - How can MAP inference problems be solved using local search?

- A heuristic solution that starts with an arbitrary assignment and performs modifications on the joint assignment that locally increase the probability.

3.17 Branch and Bound - How can MAP inference problems be solved using branch and bound?

- An exhaustive solution that searches over the space of assignments while pruning branches that can be provably shown not to contain a MAP assignment.

3.18 Simulated Annealing - How can MAP inference problems be solved using simulated annealing?

- A sampling solution that expresses a probability distribution with the following:

$$p_t(x) \propto \exp\left(\frac{1}{t}\sum_{c\in C}\theta_c(x_c)\right)$$

  - Where t is temperature.
    * $t \to \infty \implies p_t$ approaches a continuous uniform distribution.
    * $t \to 0 \implies p_t$ approaches a continuous exponential distribution with a significant peak of $\arg\max_x \sum_{c\in C}\theta_c(x_c)$.
- As the peak is a MAP assignment, a sampling algorithm starting with a high temperature which gradually decreases can eventually find the peak, given a sufficiently slow cooling rate.

3.19 Sampling-Based Inference

3.20 Motivation - Why does sampling-based inference algorithms exist?

- **Exact Inference Algorithms**: Slow/NP-Hard
- **Approximate Inference Algorithms**: Marginal Inference, MAP Inference, Expectations

3.20.1 Expectations $\mathbb{E}[f(X)]$ - Why do we want to estimate expectations of random variables?

- Abstractly, approximate inference algorithms want to estimate the probability of some real-world phenomenon.
- Mathematically, estimating a probability $p(x)$ is a **SPECIALIZATION** of estimating an expectation $\mathbb{E}_{x\sim p}[f(x)] = \sum_x f(x)p(x)$
- If $f(x) = \mathbb{I}_{|x|}$, where $\mathbb{I}_{|x|}$ is an indicator function for event x,

$$\mathbb{E}_{x\sim p}[\mathbb{I}_{|x|}] = p(x)$$

3.21 Multinomial Sampling - How do you sample a discrete CPD?

1. Let p be a multinomial probability distribution with event values $\{x^1, ..., x^k\}$ and event probabilities $\{\theta_1, ..., \theta_k\}$.
2. Generate a sample s uniformly from the interval $[0, 1]$.
3. Partition the interval into k subintervals:

$$[0, \theta_1), [\theta_1, \theta_1 + \theta_2), ..., \left[\sum_{j=1}^{i-1}\theta_j, \sum_{j=1}^{i}\theta_j\right)$$

4. If s is in the ith interval, then the sampled value is x^i.

- **Time Complexity**: $O(\log k)$ - *Using Binary Search*
- *Remember Baye's Rule*: $p(y \mid x) = \frac{p(x,y)}{p(x)}$
  - $p(x, y)$ is a multinomial probability distribution.

3.22 Forward Sampling - How do you sample a discrete Bayesian network?

1. Let G be a Bayesian network representing a probability distribution $p(x_1, ..., x_n)$.
2. Sample the variables in a topological order.
3. Sample the successor variables by conditioning these node's CPDs to the values sampled by their ancestors.
4. Repeat until all n variables have been sampled.

- **Time Complexity**: $O(n)$

3.23 Monte-Carlo Integration/Estimation - How do you take a large number of samples to estimate expectations?

- *Monte-Carlo ≈ Large Number of Samples*

$$\mathbb{E}_{x\sim p}[f(x)] \approx I_T = \frac{1}{T}\sum_{t=1}^{T} f(x^t)$$

  - Where $x^1, ..., x^T$ are i.i.d. samples drawn according to p.

$$\mathbb{E}_{x^1,...,x^T \sim_{\text{i.i.d.}} p}[I_T] = \mathbb{E}_{x\sim p}[f(x)]$$

$$\text{Var}_{x^1,...,x^T \sim_{\text{i.i.d.}} p}[I_T] = \frac{1}{T}\text{Var}_{x\sim p}[f(x)]$$

  - Where the Monte-Carlo estimate I_T

3.23.1 Implications - What is important about Monte-Carlo estimations?

1. I_T is an unbiased estimator for $\mathbb{E}_{x\sim p}[f(x)]$.
2. Referencing the Weak Law of Large Numbers, if $T \to \infty$, then $I_T \to \mathbb{E}_{x\sim p}[f(x)]$.

3.24 Rejection Sampling - How does rejection sampling work?

- Compute a target probability distribution $p(x)$ by sampling a proposal probability distribution $q(x)$, rejecting samples inconsistent with $p(x)$, and applying the Monte-Carlo estimation.
  - **Examples**: *See Rejection Sampling*
  - **Disadvantage**: *Ignores Many Samples*

3.25 Importance Sampling - How does importance sampling work?

- Compute a target probability distribution $p(x)$ by sampling a proposal probability distribution $q(x)$, reweighing samples with $w(x) = \frac{p(x)}{q(x)}$, and applying the Monte-Carlo estimation.

$$\begin{aligned}
\mathbb{E}_{x\sim p}[f(x)] &= \sum_x f(x)p(x) \\
&= \sum_x f(x)\frac{p(x)}{q(x)}q(x) \\
&= \mathbb{E}_{x\sim q}[f(x)w(x)] \\
&\approx \frac{1}{T}\sum_{t=1}^{T} f(x^t)w(x^t)
\end{aligned}$$

- **Examples**: *See Importance Sampling*
- **Advantage**: *Uses All Samples*

3.26 Normalized Importance Sampling - How does normalized importance sampling work?

1. Let $p(x)$ be unknown.
2. Let $\tilde{p}(x) = Z \cdot p(x)$ be known.
3. The weight $w(x) = \frac{\tilde{p}(x)}{q(x)}$ is invalid for unnormalized importance sampling.
4. The **normalizing constant** of the distribution $\tilde{p}(x)$ is the following:

$$\mathbb{E}_{x\sim q}[w(x)] = \sum_x q(x)\frac{\tilde{p}(x)}{q(x)} = \sum_x \tilde{p}(x) = Z$$

5. The **normalized importance sampling estimator** is the following:

$$\begin{aligned}\mathbb{E}_{x\sim p}[f(x)] &= \sum_x f(x)p(x) \\ &= \sum_x f(x)\frac{p(x)}{q(x)}q(x) \\ &= \frac{1}{Z}\sum_x f(x)\frac{\tilde{p}(x)}{q(x)}q(x) \\ &= \frac{1}{Z}\mathbb{E}_{x\sim q}[f(x)w(x)] \\ &= \frac{\mathbb{E}_{x\sim q}[f(x)w(x)]}{\mathbb{E}_{x\sim q}[w(x)]}\end{aligned}$$

3.27 Markov Chain - What is a Markov chain?

- **Markov Chain**: A sequence of random variables $S_0, S_1, S_2, \ldots$ with each random variable $S_i \in \{1, 2, \ldots, d\}$ taking one of d possible values.
  - *Initial State*: $P(S_0)$
  - *Subsequent States*: $P(S_i \mid S_{i-1})$
- **Markov Assumption**: S_i cannot depend directly on S_j where $j < i - 1$.

3.28 Stationary Distribution - Why is it important for a stationary distribution to exist?

- Let $T_{ij} = P(S_{\text{new}} = i \mid S_{\text{prev}} = j)$ be a $d \times d$ transition probability matrix.
- If the initial state S_0 is drawn from a vector probabilities p_0, the probability p_t of ending in **EACH STATE** after t steps is the following:

$$p_t = T^t p_0$$

  - Where T^t is matrix exponentiation.
- **Stationary Distribution**: If it exists, the limit $\pi = \lim_{t\to\infty} p_t$.
- **Important Note 1**: A Markov chain whose states are joint assignments to the variables in a probabilistic graphical model p has a stationary distribution equal to p.

3.28.1 Existence of Stationary Distribution

- **Irreducibility**: It is possible to get from any state x to any other state x' with probability > 0 in a finite number of steps.
- **Aperiodicity**: It is possible to return to any state at any time, i.e. there exists an n such that for all i and all $n' \geq n$, $P(s_{n'} = i \mid s_0 = i) > 0$.
- **Important Note 2**: An irreducible and aperiodic finite-state Markov chain has a stationary distribution.

3.29 Markov Chain Monte Carlo - How do you sample from a MCMC?

1. Let T be a **transition operator** specifying a Markov chain whose stationary distribution is p.
2. Le x_0 be an initial assignment to the variables of p.
3. Run the Markov chain from x_0 for B *burn-in* steps.
   - If B is sufficiently large, $\pi \rightarrow p$.
4. Run the Markov chain for N *sampling* steps and collect all the states that it visits.
   - The collection of states form samples from p.

3.29.1 Applications of Markov Chain Monte Carlo

1. Use samples for Monte Carlo integration to estimate expectations.
2. Use samples to perform marginal inference.
3. Use the sample with the highest probability to perform MAP inference.

3.30 Gibbs Sampling - How do you construct a MCMC?

1. Let $x_1, ..., x_n$ be an ordered set of variables.
2. Let $x^0 = (x_1^0, ..., x_n^0)$ be a starting configuration.
3. Repeat until convergence for $t = 1, 2, ...,$
   1. Set $x \leftarrow x^{t-1}$.
   2. For each variable x_i,
      1. Sample $x_i' \sim p(x_i \mid x_{-i})$.
         - Where x_{-i} is all variables in x except x_i
      2. Update $x \leftarrow (x_1, ..., x_i', ..., x_n)$.
   3. Set $x^t \leftarrow x$

- **Important Note 1**: When x_i is updated, its new value is immediately used for sampling other variables x_j.
- **Important Note 2**: Every iteration of x^t is a new sample from p.

4 Learning

- **Problem**: *Given a dataset D of m i.i.d. samples from some underlying distribution p^*, how do you fit the best model, given a family of models M, to make useful predictions?*
  - **Parameter Learning**: *Where the graph structure is known, and we want to estimate the factors.*

  - **Structure Learning**: *Where we want to estimate the graph, i,e. determine from data how the variables depend on each other.*

- **Solution**: *Best Approximation of p^**
  - **Density Estimation**: *We are interested in the full distribution.*
  - **Specific Prediction Tasks**: *We are using the distribution to make a prediction.*
    * e.g. Is this email spam or not?
  - **Structure or Knowledge Discovery**: *We are interested in the model itself.*
    * e.g. How do some genes interact with each other?

4.1 Maximum Likelihood Estimation

4.2 Motivation - Why does maximum likelihood estimation exist?

- **Goal**: How do we approximate p as close as possible to p^*?
- **Approach**: When the KL divergence between p and p^* is minimal, p is as close as possible to p^*.

4.2.1 KL Divergence - What is KL divergence?

- **KL Divergence**: How different is one probability distribution from another probability distribution?

$$KL(p^* \parallel p) = \sum_x p^*(x) \log \frac{p^*(x)}{p(x)} = -H(p^*) - \mathbb{E}_{x\sim p^*}[\log p(x)]$$

4.2.2 Minimal KL Divergence - When is KL divergence minimal?

- **General Idea**: *Minimizing KL Divergence $\Longleftrightarrow$ Maximizing Likelihood*

$$\min KL(p^* \parallel p) \Longleftrightarrow \max \mathbb{E}_{x\sim p^*}[\log p(x)]$$

- Because p^* is unknown, approximate the log-likelihood with the emperical log-likelihood using a Monte-Carlo estimate.

$$\mathbb{E}_{x\sim p^*}[\log p(x)] \approx \frac{1}{|D|} \sum_{x\in D} \log p(x)$$

4.2.3 Maximum Likelihood Learning - How do you fit the best model using maximum likelihood learning?

- Given a family of models M, to fit the best model p, compute the following.

$$\max_{p\in M} \mathbb{E}_{x\sim p^*}[\log p(x)] \approx \max_{p\in M} \frac{1}{|D|} \sum_{x\in D} \log p(x)$$

4.3 Definition - What is maximum likelihood estimation?

- **Maximum Likelihood Estimation**: Given a data set D, choose parameters $\hat{\theta}$ that satisfy the following.

$$\max_{\theta\in\Theta} L(\theta, D)$$

  - i.e., Maximize the parameters θ to best fit the data set D.

4.4 Loss Function - What is a loss function?

- **Loss Function** ($L(x, p)$): A measure of the loss that a model distribution p makes on a particular instance x.
  - e.g., *MLE Loss Function*: $L(x, p) = -\log p(x)$
- **Important Note**: Assuming instances are sampled from some distribution p^*, to fit the best model, **MINIMIZE** the expected loss.

$$\mathbb{E}_{x\sim p^*}[L(x, p)] \approx \frac{1}{|D|}\sum_{x\in D} L(x, p)$$

4.5 Likelihood Function - What is a likelihood function?

- **Likelihood Function** ($L(\theta, D)$): The probability of observing the i.i.d. samples D for all permissible values of the parameters θ.

4.5.1 Example - Likelihood Function

1. Let $p(x)$ be a probability distribution where $x \in \{h, t\}$ such that $p(x = h) = \theta$ and $p(x = t) = 1 - \theta$.
2. Let $D = \{h, h, t, h, t\}$ be observed i.i.d. samples.
3. Accordingly, $p(x)$ models the outcome of a biased coin where parameter θ represents the probability of flipping heads and $1 - \theta$ represents the probability of flipping tails.
4. Express the likelihood function as the following.

$$L(\theta, D) = \theta \cdot \theta \cdot (1-\theta) \cdot \theta \cdot (1-\theta) = \theta^3 \cdot (1-\theta)^2$$

4.6 Maximum Likelihood Learning - How does maximum likelihood learning estimate the CPDs in Bayesian networks?

1. Let $p(x) = \prod_{i=1}^{n} \theta_{x_i|x_{pa(i)}}$ be a Bayesian network.
   - Where $\theta_{x_i|x_{pa(i)}}$ are parameters (CPDs) with **UNKNOWN VALUES**.
2. Let $D = \{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ be i.i.d. samples.
3. Let $L(\theta, D) = \prod_{i=1}^{n}\prod_{j=1}^{m} \theta_{x_i^j|x_{pa(i)}^j}$ be the likelihood function.
4. Log and collect like terms of the likelihood function.

$$\log L(\theta, D) = \sum_{i=1}^{n}\sum_{x_{pa(i)}}\sum_{x_i} \#(x_i, x_{pa(i)}) \cdot \log \theta_{x_i|x_{pa(i)}}$$

5. Maximize the (log) likelihood function by decomposing it into separate maximizations for the local conditional distributions.

- **Important Note**: The maximum-likelihood estimates of the parameters (CPDs) have closed-form solutions.

$$\theta^*_{x_i|x_{pa(i)}} = \frac{\#(x_i, x_{pa(i)})}{\#(x_{pa(i)})}$$

4.7 Bayesian Learning

4.8 Motivation - What are some problems with maximum likelihood estimation?

- A maximum likelihood estimate does not change as more data is observed because it assumes that the only source of uncertainty is explained by the parameters that are being fitted.
- **Problem 1**: *Cannot Improve Confidence*
- **Problem 2**: *Cannot Incorporate Prior Knowledge*

4.9 Definitions - What are a *prior* and a *posterior*?

- **Bayesian Learning**: Explicitly model uncertainty over both variables X and parameters θ by letting parameters be random variables.
- A **prior** is the earlier probability distribution of parameter θ **BEFORE** observing data D.
- A **posterior** is the later probability distribution of parameter θ **AFTER** observing data D.

$$p(\theta \mid D) = \frac{p(D \mid \theta)p(\theta)}{p(D)} \propto p(D \mid \theta)p(\theta)$$

$$\textit{posterior} \propto \textit{likelihood} \times \textit{prior}$$

- **Important Note 1**: Bayes' rule allows prior knowledge to be incorporated into a model's parameters.
- **Important Note 2**: Using Bayes' rule, the numerator is easy to calculate, but the denominator is difficult to calculate.
- **Important Note 3**: The expected value of the posterior $p(\theta \mid D)$ is the estimate of the parameter θ that causes p to be as close as possible to p^*.

4.10 Conjugate Priors - What is a *conjugate prior*?

- A parametric family ϕ is **conjugate** for the likelihood $P(D \mid \theta)$ if:

$$P(\theta) \in \phi \implies P(\theta \mid D) \in \phi$$

- **Important Note**: If the normalizing constant of ϕ is known, then the denominator in Bayes' rule is easy to calculate.

4.11 Beta Distribution - What is the Beta distribution?

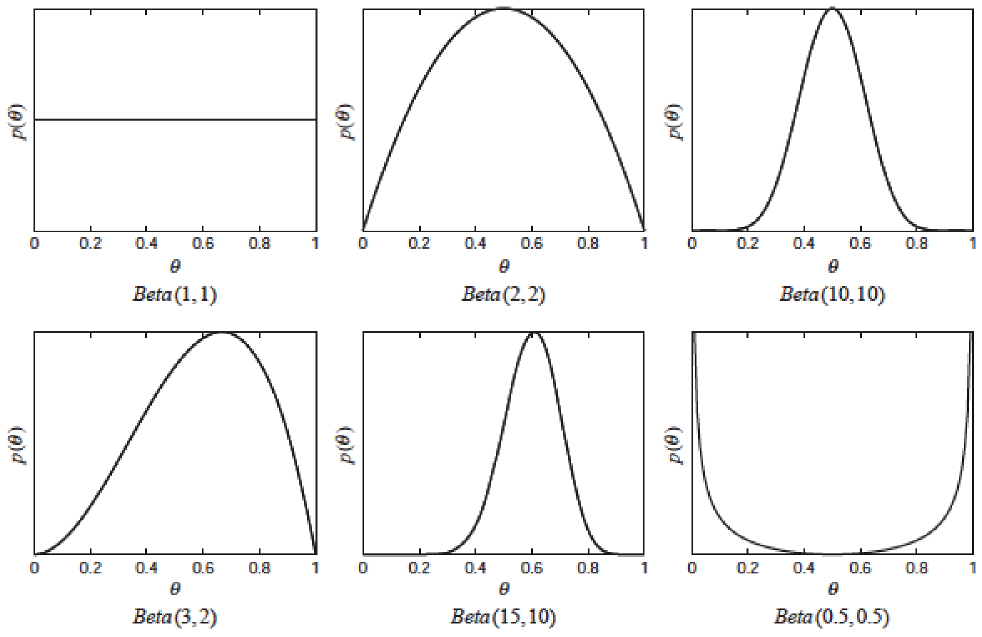


Figure 17.4 Examples of Beta distributions for different choices of hyperparameters

Examples of Beta Distribution

- A **Beta distribution** is parameterized by two hyperparameters $\alpha \in \mathbb{R}$, and $\beta \in \mathbb{R}$ with the following continuous probability distribution.

$$\theta \sim \text{Beta}(\alpha, \beta) \implies p(\theta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha, \beta)}$$

  - Where $\theta \in (0, 1)$.
  - Where the constant α intuitively corresponds to the number of **SUCCESSES** outcomes.
  - Where the constant β intuitively corresponds to the number of **FAILURES** outcomes.
  - Where the constant $B(\alpha, \beta)$ is a normalizing constant defined by the following.

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

  - Where the Gamma function $\Gamma(x)$ is the continuous generalization of the factorial function defined by the following.

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$$

- **Expected Value**:

$$\mathbb{E}[X] = \frac{\alpha}{\alpha+\beta}$$

4.12 Conjugate Priors and Beta Distribution - How do you calculate a posterior with data observed from a binary process?

Think Coin Toss

- The beta distribution is the conjugate prior for the following probability distributions:
  - **Bernoulli**: A discrete Bernoulli random variable, X, is the outcome from a single experiment from which this outcome is classified as either a success, $X = 1$ with probability p, or a failure, $X = 0$ with probability $1 - p$.

$$X \sim \text{Bernoulli}(p) \implies P(\{X = x\}) = \begin{cases} 1 - p & \text{if } x = 0 \\ p & \text{if } x = 1 \end{cases}$$

  - **Binomial**: A discrete binomial random variable, X, is the number of successful outcomes from a sequence of n independent experiments in which each experiment has an outcome classified as either a success with probability p or a failure with probability $1 - p$

$$X \sim \text{Binomial}(n, p) \implies P(\{X = x\}) = \binom{n}{x} p^x (1 - p)^{n-x}$$

  - **Geometric**: A discrete geometric random variable, X, is the number of Bernoulli trials with probability p needed to get one success.

$$X \sim \text{Geometric}(p) \implies P(\{X = x\}) = (1 - p)^{x-1} p$$

  - **Negative Binomial**: A discrete negative binomial random variable, X, is the number of successes in a sequence of independent and identically distributed Bernoulli trials before r failures.

$$X \sim \text{Negative-Binomial}(r, p) \implies P(\{X = x\}) = \binom{x + r - 1}{x} p^x (1 - p)^r$$

- **Important Note**: To best fit a binary model with a probability distribution $p \approx p^*$, the beta distribution can be used by the following.
  1. Assign $\text{Beta}(\alpha, \beta)$ as a prior to p.
  2. Observe n data points generated by a binary process with unknown, underlying p^*.
     - If $X_i \sim \text{Bernoulli}(p^*)$, then the posterior is $\text{Beta}(\alpha + \sum_{i=1}^n x_i, \beta + n - \sum_{i=1}^n x_i)$.
     - If $X_i \sim \text{Binomial}(N, p^*)$, then the posterior is $\text{Beta}(\alpha + \sum_{i=1}^n x_i, \beta + \sum_{i=1}^n N_i - \sum_{i=1}^n x_i)$.
     - If $X_i \sim \text{Geometric}(p^*)$, then the posterior is $\text{Beta}(\alpha + n, \beta + \sum_{i=1}^n x_i - n)$.
     - If $X_i \sim \text{Negative-Binomial}(r, p^*)$, then the posterior is $\text{Beta}(\alpha + \sum_{i=1}^n x_i, \beta + rn)$.

4.13 Dirichlet Distribution (Multivariate Beta Distribution) - What is the Dirichlet distribution?

- A **Dirichlet distribution** is parameterized by the following hyperparameters $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_K) \in \mathbb{R}^K$ with the following continuous probability distribution.

$$\theta \sim \text{Dirichlet}(\boldsymbol{\alpha}) \implies p(\theta) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} \theta_i^{\alpha_i - 1}$$

  - Where $\theta_i \in (0,1)$ and $\sum_{i=1}^K \theta_i = 1$.
  - Where the constant α_i intuitively corresponds to the number of outcomes for category i.
  - Where the constant $B(\boldsymbol{\alpha})$ is a normalizing constant defined by the following.

$$B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^K \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^K \alpha_i)}$$

  - Where the Gamma function $\Gamma(x)$ is the continuous generalization of the factorial function defined by the following.

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$$

- **Expected Value**:

$$\mathbb{E}[X_i] = \frac{\alpha_i}{\sum_k \alpha_k}$$

4.14 Conjugate Priors and Dirichlet Distribution - How do you calculate a posterior with data observed from a categorical process?

Think K-Sided Dice Roll

- The Dirichlet distribution is the conjugate prior for the following probability distributions:
  - **Categorical** (*Generalized Bernoulli*): A discrete categorical random variable, X, is the outcome from a single experiment from which this outcome is classified as one of K categories with probability $p_i > 0$ and $\sum_{i=1}^K p_i = 1$.

$$X \sim \text{Categorical}(\boldsymbol{p}) \implies P(\{X = i\}) = p_i$$

  - **Multinomial** (*Generalized Binomial*): A discrete multinomial random variable, X, is the number of outcomes from a sequence of n independent experiments in which each experiment has an outcome classified as one of K categories with probability $p_i > 0$ and $\sum_{i=1}^K p_i = 1$.

$$X \sim \text{Multinomial}(n, \boldsymbol{p}) \implies P(\{\mathbf{X} = \boldsymbol{x}\}) = \frac{n!}{x_1!...x_k!} p_1^{x_1}...p_k^{x_k}$$

- **Important Note**: To best fit a categorical model with a probability distribution $\boldsymbol{p} \approx \boldsymbol{p}^*$, the Dirichlet distribution can be used by the following.
  1. Assign Dirichlet($\boldsymbol{\alpha}$) as a prior to $\boldsymbol{p}$.
  2. Observe n data points generated by a categorical process with unknown, underlying $\boldsymbol{p}^*$.
     - If $X_i \sim \text{Categorical}(\boldsymbol{p}^*)$, then the posterior is Dirichlet($\boldsymbol{\alpha} + (c_1, ..., c_K)$).
       * Where c_i is the number of observations in category i.
     - If $X_i \sim \text{Multinomial}(N, \boldsymbol{p}^*)$, then the posterior is Dirichlet($\boldsymbol{\alpha} + \sum_{i=1}^n \boldsymbol{x_i}$).

4.15 Gamma Distribution - What is the Gamma distribution?

- A **Gamma distribution** is parameterized by two hyperparameters $\alpha \in \mathbb{R}$, and $\beta \in \mathbb{R}$ with the following continuous probability distribution.

$$\theta \sim \text{Gamma}(\alpha, \beta) \implies p(\theta) = \frac{\beta^\alpha}{\Gamma(\alpha)} \theta^{\alpha-1} e^{-\beta\theta}$$

  - Where $\theta \in (0, \infty)$.
  - Where the constants α and β intuitively corresponds to α total occurrences in β intervals.
  - Where the Gamma function $\Gamma(x)$ is the continuous generalization of the factorial function defined by the following.

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

- **Expected Value**:

$$\mathbb{E}[X] = \frac{\alpha}{\beta}$$

4.16 Conjugate Priors and Gamma Distribution - How do you calculate a posterior with data observed from a Poisson point process?

Think Rate (Events/Second)

- The Gamma distribution is the conjugate prior for the following probability distributions:
  - **Poisson**: A discrete Poisson random variable, X, is the number of events occuring in a fixed interval of time or at a fixed rate.

$$X \sim \text{Poisson}(\lambda) \implies P(\{X = x\}) = e^{-\lambda} \frac{\lambda^x}{x!}$$

  - **Exponential**: A continuous exponential random variable, X, is the time between events in a Poisson point process.

$$X \sim \text{Exponential}(\lambda) \implies P(\{X = x\}) = \lambda e^{-\lambda x}$$

- **Important Note**: To best fit a Poisson point model with a probability distribution $p \approx p^*$, the Gamma distribution can be used by the following.
  1. Assign $\text{Gamma}(\alpha, \beta)$ as a prior to p.
  2. Observe n data points generated by a Poisson point process with unknown, underlying p^*.
     - If $X_i \sim \text{Poisson}(\lambda^*)$, then the posterior is $\text{Gamma}(\alpha + \sum_{i=1}^n x_i, \beta + n)$.
     - If $X_i \sim \text{Exponential}(\lambda^*)$, then the posterior is $\text{Gamma}(\alpha + n, \beta + \sum_{i=1}^n x_i)$.

5 Decision Making Under Uncertainty

5.1 A/B Testing

- *See Bayesian Methods for Hackers: Probabilistic Programming and Bayesian Inference*
  - *Chapter 1: Introduction to Bayesian Methods*
  - *Chapter 2: A little more on PyMC*
  - *Chapter 6: Getting our prior-ities straight*

5.2 Definition - What is A/B Testing?

- A randomized experiment with two variants, A and B, that determines which of the two variants is more effective with their respective cohorts at achieving successes in n trials.

5.3 Bayesian Learning and A/B Testing - How do you determine which of A and B are better after observing n_A and n_B trials?

1. Let $0 < p_A < 1$ be the unknown probability that A is truly effective.
2. Let $0 < p_B < 1$ be the unknown probability that B is truly effective.
3. Let $\delta = A - B$ be the measure whether A is more effective than B.
4. Let $\text{Beta}(1, 1)$ be the initial prior for p_A.
    - Use different values for α and β to inject a subjective prior belief about p_A.
5. Let $\text{Beta}(1, 1)$ be the initial prior for p_B.
    - Use different values for α and β to inject a subjective prior belief about p_B.
6. Observe n_A data points generated by a Bernoulli process with unknown, underlying p_A.
7. Observe n_B data points generated by a Bernoulli process with unknown, underlying p_B.
8. Update the posterior p_A given n_A using Bayesian learning.
9. Update the posterior p_B given n_B using Bayesian learning.
10. Approximate the joint distribution of δ using a MCMC constructed using MCMCs of posteriors p_A and p_B.
    - Count the number of samples greater than 0 for the probability that A is better than B; i.e. the area under the curve after 0.
    - Count the number of samples less than 0 for the probability that A is worse than B; i.e. the area under the curve before 0.

5.4 Multi-Armed Bandits

- *See Reinforcement Learning: An Introduction*
    - *Chapter 2: Multi-armed Bandits*

5.5 Definition - What is the Multi-armed bandit problem?

- A fixed limited set of resources must be allocated between competing (alternative) actions in a way that maximizes their expected reward, when each action's properties are only partially known at the time of allocation, and may become better understood as time passes or by allocating resources to the action.

5.6 Exploration-Exploitation Tradeoff Dilemma - What is the exploration-exploitation tradeoff dilemma?

- **Exploitation**: If we have chosen an action that yields a pretty good reward, do we keep choosing it to maintain a pretty good reward?
- **Exploration**: Otherwise, do we choose other actions in hopes of finding an even-better action?

5.7 Learning Rule as Averaging - What is the standard form of learning rules?

$$\text{New Estimate} \leftarrow \text{Old Estimate} + \text{Step Size} \cdot [\text{Target} - \text{Old Estimate}]$$

5.8 ϵ-Greedy Bandits Algorithm - How do you choose actions using the ϵ-greedy bandits algorithm?

1. Let A_t be the action chosen from k possibilities at time step t.
2. Let R_t be the independent and identically distributed reward yielded by action A_t.
3. Let $Q_t(a)$ be the expected reward from choosing action a until time step t.
4. Let $N_t(a)$ be the number of times action a was chosen until time step t.
5. Initialize $Q_0(a) \leftarrow 0, \forall a \in \{1, ..., k\}$.
6. Initialize $N_0(a) \leftarrow 0, \forall a \in \{1, ..., k\}$.
7. Repeat for $t = 0, 1, 2, ...,$

    1. **Set** $A_t \leftarrow \max_a Q_t(a)$ **with probability** $1 - \epsilon$**.**
    2. **Set** $A_t \leftarrow$ **random**(a) **with probability** ϵ**.**
    3. Sample $R_t \leftarrow \text{bandit}(A_t)$.
    4. Update $N_{t+1}(A_t) \leftarrow N_t(A_t) + 1$.
    5. Update $Q_{t+1}(A_t) \leftarrow Q_t(A_t) + \frac{1}{N_t(A_t)}[R_t - Q_t(A_t)]$.

5.9 ϵ-Greedy Action Selection - How does ϵ-greedy action selection work?

$$A_t \leftarrow \begin{cases} \max_a Q_t(a) & \text{with probability } 1-\epsilon \\ \text{random}(a) & \text{with probability } \epsilon \end{cases}$$

- Balances exploration and exploitation by choosing randomly a small fraction of the time.
- **Problem**: Indiscriminately, with no preference for those that are nearly greedy or particularly uncertain, selects non-greedy actions.

5.10 Upper-Confidence-Bound Action Selection - How does upper-confidence-bound action selection work?

$$A_t \leftarrow \max_a \left[Q_t(a) + c \cdot \sqrt{\frac{\log t}{N_t(a)}} \right]$$

- Balances exploration and exploitation by choosing deterministically but achieving exploration by subtly favoring at each step the actions that have so far received fewer samples.
    - $\sqrt{\frac{\log t}{N_t(a)}}$ is a measure of the uncertainty or variance in the estimate of a's value.
    - $c > 0$ determines the confidence level by controlling the degree of exploration.
    - $\log t$ increases the uncertainty in the estimate of a's value as a is exploited less.
    - $N_t(a)$ decreases the uncertainty in the estimate of a's value as a is exploited more.
- **Solution for ϵ-Greedy Action Selection's Problem**: As $\log t$'s increases get smaller over time, but are unbounded; all actions will eventually be selected, but actions with lower value estimates or that have already been selected frequently, will be selected with decreasing frequency over time.

5.11 Thompson Sampling Bandits Algorithm - How do you choose actions using the Thompson sampling bandits algorithm?

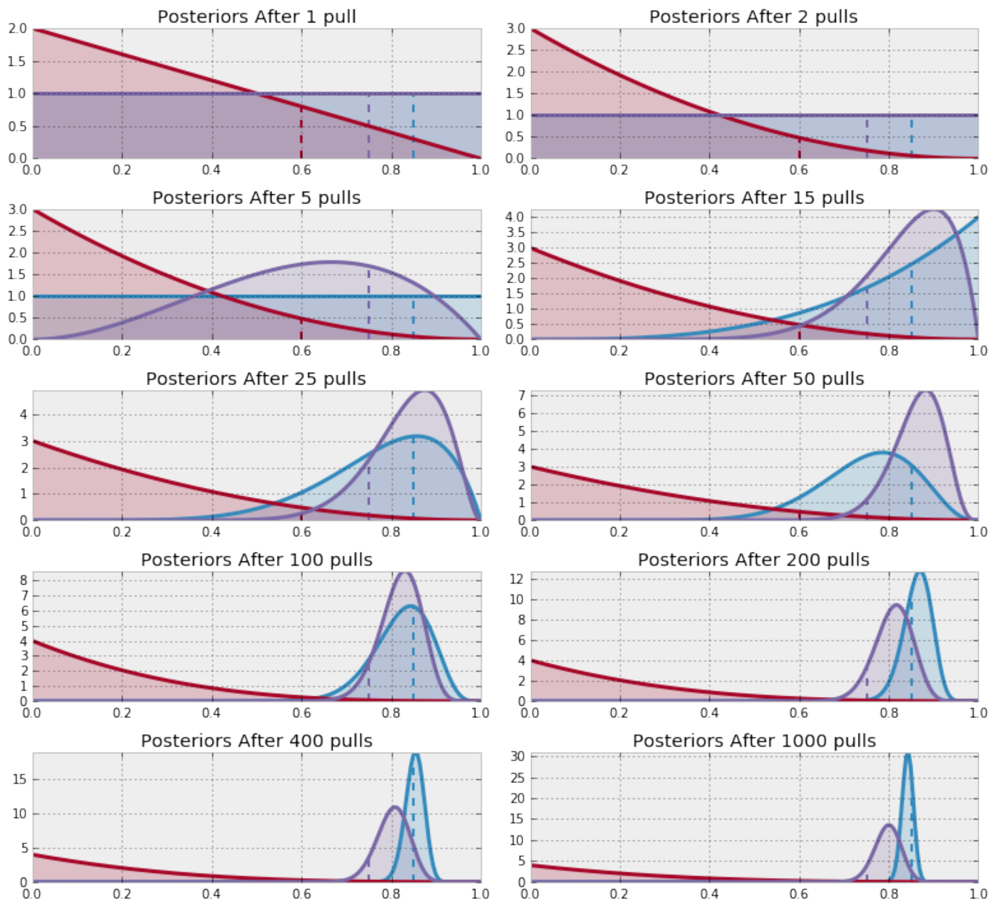
1. Let A_t be the action chosen from k possibilities at time step t.
2. Let R_t be the independent and identically distributed reward yielded by action A_t.
3. Let $p_t(a)$ be the initially unknown probabilitites of the rewards yielded by choosing action a until time step t.
4. Repeat for $t = 0, 1, 2, ...,$
    1. Sample a random variable X_a using $p_t(a)$ for each $a \in \{1, ..., k\}$.
    2. Set $A_t \leftarrow \max_a X_a$.
    3. Sample $R_t \leftarrow \text{bandit}(A_t)$.
    4. Update $p_{t+1}(A_t \mid R_t) \propto p_t(R_t \mid A_t) p_t(A_t)$.
        - *i.e., According to Bayesian learning.*

5.12 Thompson Sampling Action Selection - How does Thompson sampling action selection work?

$$X_a \sim p_t(a)$$
$$A_t \leftarrow \max_a X_a$$

- Balances exploration and exploitation by sampling actions in proportion to the belief they are optimal.
- **Solution for ϵ-Greedy Action Selection's Problem**: As TS selects actions according to **SAMPLES** from posterior distributions of the unknown probabilities that each respective action is optimal, TS explores to resolve uncertainty where there is a chance that resolution will help the agent identify the optimal action, but avoids probing where feedback would not be helpful.

5.12.1 Example: Thompson Sampling Intuition



Thompson Sampling Action Selection Example

- The wider a posterior distribution, there is more uncertainty in probability that an action is optimal, so it still has the chance of generating the best sample at time t which leads to further exploration to reduce uncertainty.
- The narrower a posterior distribution, there is less uncertainty in probability that an action is optimal, so either its action will be exploited more (if more certainty in its *optimality*) or explored less (if more certainty in its *sub-optimality*).

5.13 Upper-Confidence-Bounds and Thompson Sampling

- Both heuristics explore the actions that are currently predicted to be optimal, and they are optimistic about action if there is too little information.
- **Bayesian Optimization**: Both optimize Bayesian regret and their bounds can be converted between each other.

5.14 Limitations of Thompson Sampling

1. **Problems Not Requiring Exploration**: Some problems are better solved by a greedy algorithm.

2. **Problems Not Requiring Exploitation**: Some problems do not need a balance between exploration and exploitation.
3. **Time Sensitivity**: Some time-sensitive problems favor intensification of exploitation than a balance between exploration and exploitation.
4. **Problems Requiring Careful Assessment of Information Gain**: Some problems require a more careful assessment of the information actions provide instead of favoring the most promising actions.

5.15 Markov Decision Processes

5.16 Agent-Environment Interface - How does an agent interact with its environment?

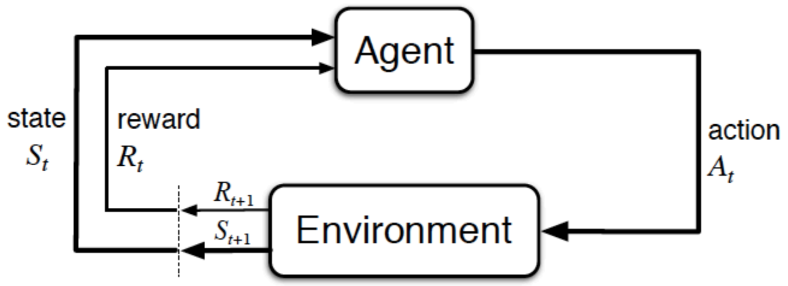


Figure 3.1: The agent–environment interaction in a Markov decision process.

Agent-Environment Interface

- At discrete time steps $t = 0, 1, 2, 3, ...,$
  1. The agent observes the state from the environment, at step t: $S_t \in \mathcal{S}$.
  2. The agent produces an action upon the environment, at step t: $A_t \in \mathcal{A}(S_t)$.
  3. The agent gets a resulting reward from the environment: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$.
  4. The environment transitions into the next state as a result of the agent's action: $S_{t+1} \in \mathcal{S}^+$.

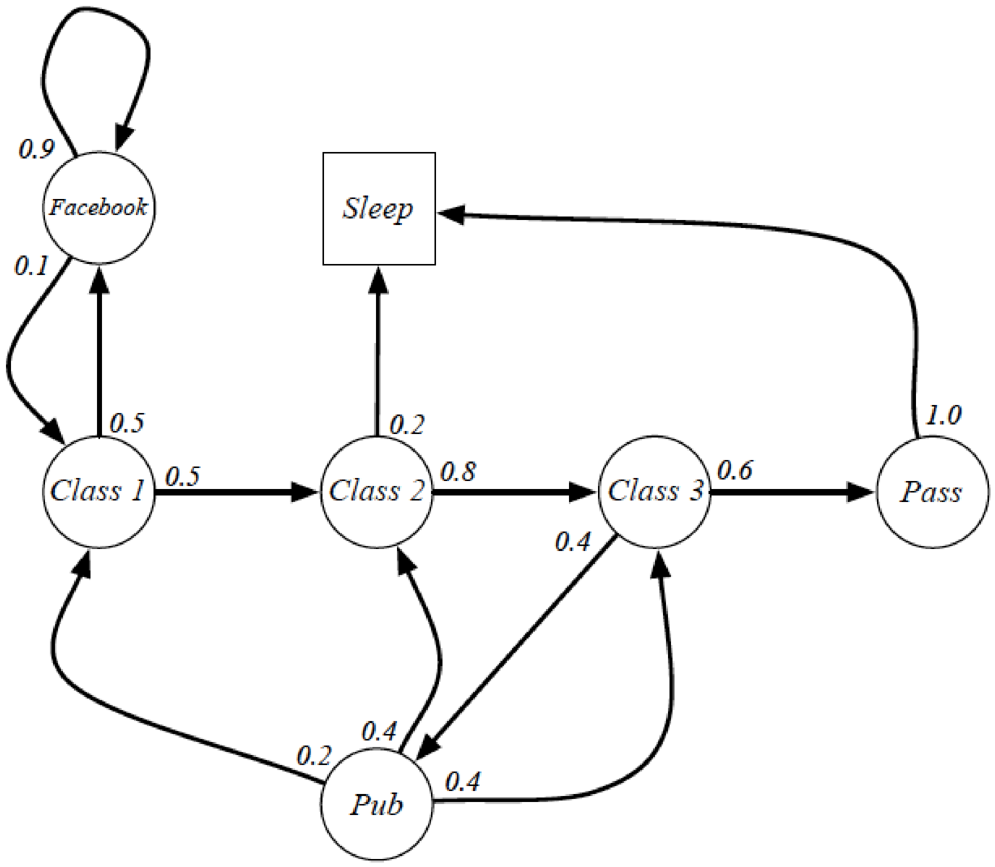
5.17 Markov Property - What is the Markov property?

- A state S_t is **Markov** if and only if,

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

- **Important Note**: i.e., the future is independent of the past given the present.

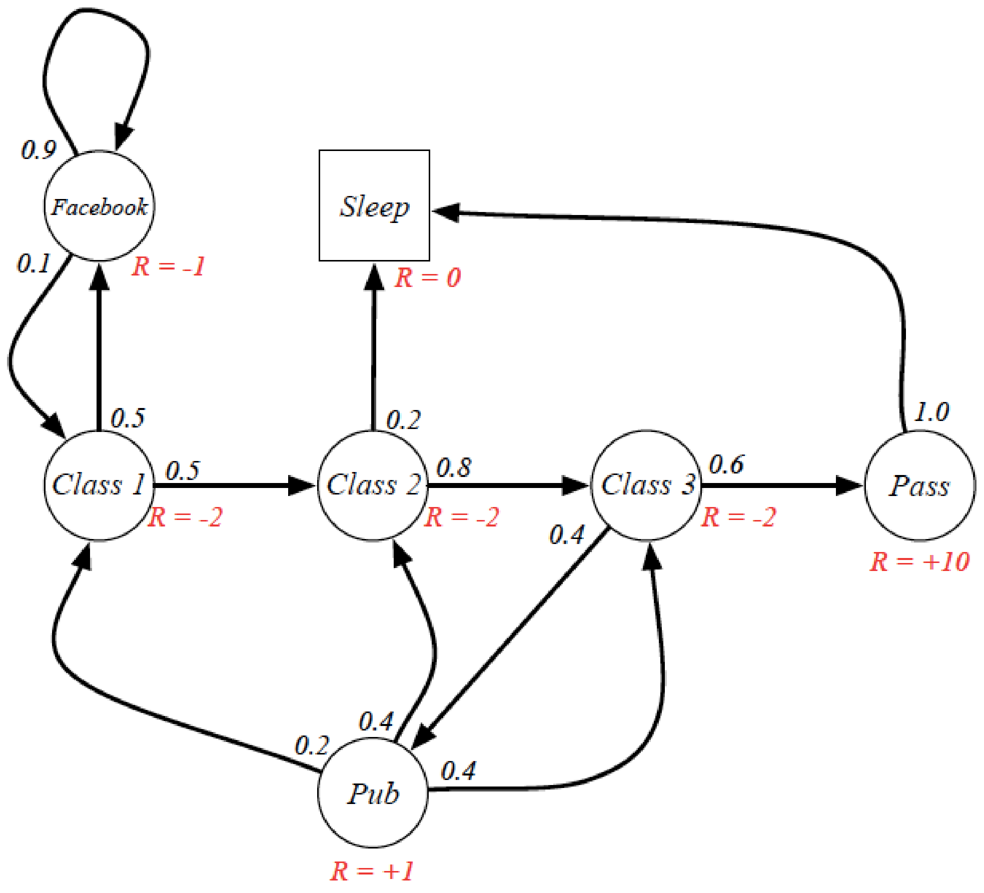
5.18 Markov Process/Chain - What is a Markov process/chain?



Markov Process Example

- A **Markov process/chain** is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$,
  - $\mathcal{S}$ is a finite set of states.
  - $\mathcal{P}$ is a *state transition probability matrix*, $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$.
- **Important Note**: i.e., a memoryless random process.

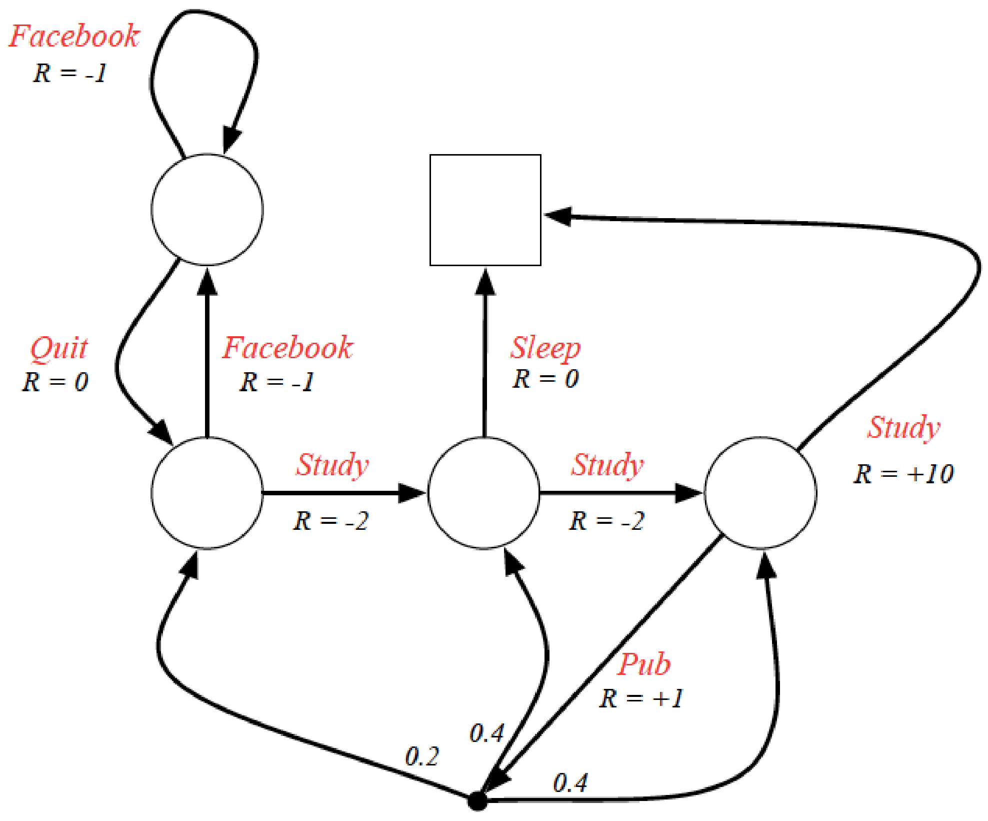
5.19 Markov Reward Process - What is a Markov reward process?



Markov Reward Process Example

- A **Markov reward process** is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{S}$ is a finite set of states.
  - $\mathcal{P}$ is a *state transition probability matrix*, $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$.
  - $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$.
  - $\gamma \in [0, 1]$ is a discount factor.

5.20 Markov Decision Process - What is a Markov decision process?



Markov Decision Process Example

- A **Markov decision process** is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{S}$ is a finite set of states.
  - $\mathcal{A}$ is a finite set of actions.
  - $\mathcal{P}$ is a *state transition probability matrix*, $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$.
  - $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$.
  - $\gamma \in [0, 1]$ is a discount factor.
- **Important Note**: A majority of reinforcement learning problems can be formalized as MDPs.

5.21 Policy - What is a policy?

- A **policy** π is a distribution over actions given states,

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$$

- **Important Note 1**: A policy fully defines the behavior of an agent.
- **Important Note 2**: Reinforcement learning methods specify how an agent changes its policy according to history.

5.22 Return - What is a return?

- The **return** G_t is some measure of reward from time-step t.

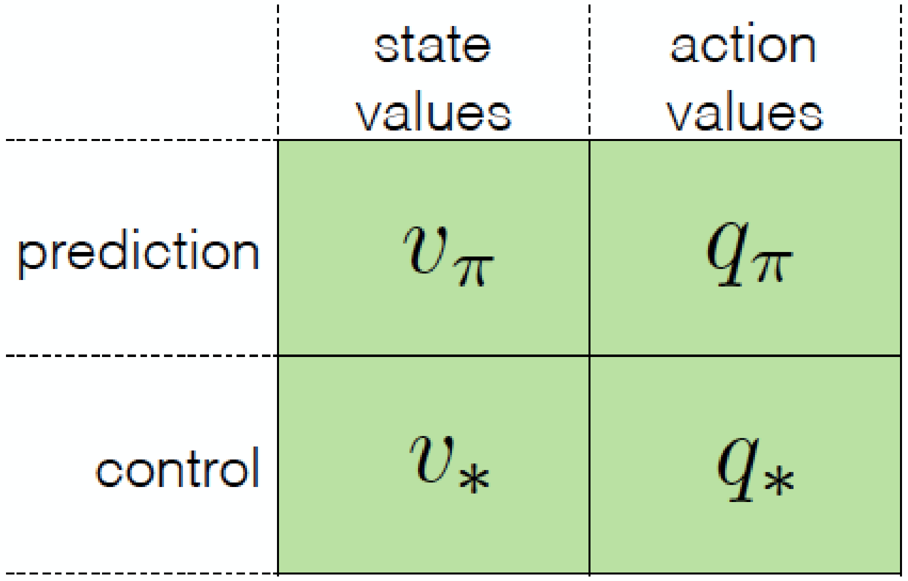
$$G_t = R_{t+1} + \gamma \cdot R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- **Important Note**: Reinforcement learning methods seek to maximize the *expected return* $\mathbb{E}[G_t]$, on each time-step t.

5.22.1 Variations of Reward

- **Total Reward**: The sum of all future rewards for episodic tasks.
  - $\gamma = 1$.
  - $R_t = 0, \forall t > T$, where T is the terminal time-step.
- **Discounted Reward**: The sum of all future discounted rewards for continuous tasks.
  - $\gamma \rightarrow 0$ implies *shortsighted* rewards.
  - $\gamma \rightarrow 1$ implies *farsighted* rewards.
  - Typically, $\gamma = 0.9$.
- **Important Note**: A discount factor avoids infinite returns in cyclic Markov processes.

5.23 Value Functions - What are the 4 value functions?



	state values	action values
prediction	v_π	q_π
control	v_*	q_*

4 Ideal Value Functions

- The **state-value function** $v_\pi(s)$ is the expected return starting from state s, and then following policy π.

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

- The **action-value function** $q_\pi(s,a)$ is the expected return starting from state s, taking action a, and then following policy π.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

- The **optimal state-value function** $v_*(s)$ is the maximum value function over all policies.

$$v_*(s) = \max_\pi v_\pi(s)$$

- The **optimal action-value function** $q_*(s,a)$ is the maximum action-value function over all policies.

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

- **Important Note 1**: The optimal value function specifies the best possible performance in the MDP.
- **Important Note 2**: A MDP is solved when the optimal value function is known.

5.24 Optimal Policy - What is an optimal policy?

- **FOR ANY MDP**,
  1. There exists an **optimal policy** π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$.
  $$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$
  2. All optimal policies achieve the same optimal value function, $v_{\pi_*}(s) = v_*(s)$.
  3. All optimal policies achieve the same optimal action-value function, $q_{\pi_*}(s,a) = q_*(s,a)$.

5.24.1 Finding a Deterministic Optimal Policy

$$\pi_*(a \mid s) = \begin{cases} 1 & \text{if } a = \max_{a \in \mathcal{A}} q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

- **Important Note**: The optimal policy π_* is greedy with respect to q_* or v_*. > *i.e., Deterministic Decision Making Under Uncertainty*

5.25 Bellman Equation - What are the Bellman equations for the 4 value functions?

5.25.1 State-Value Function

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid s) q_\pi(s,a) \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right) \\ v_\pi(s) &= \sum_a \pi(a \mid s) \left[\sum_{s',r} p(s',r \mid s,a) \left[r + \gamma \cdot v_\pi(s') \right] \right] \end{aligned}$$

5.25.2 Action-Value Function

$$\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma \cdot q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \\
&= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \\
&= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \left(\sum_{a' \in \mathcal{A}} \pi(a' \mid s') q_\pi(s', a') \right) \\
q_\pi(s,a) &= \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \cdot v_\pi(s') \right]
\end{aligned}$$

5.25.3 Optimal State-Value Function

$$\begin{aligned}
v_*(s) &= \max_a q_*(s,a) \\
&= \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right) \\
v_*(s) &= \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \cdot v_*(s') \right]
\end{aligned}$$

5.25.4 Optimal Action-Value Function

$$\begin{aligned}
q_*(s,a) &= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \\
&= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \left(\max_{a'} q_*(s', a') \right) \\
q_*(s,a) &= \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \cdot \max_{a'} q_*(s', a') \right]
\end{aligned}$$

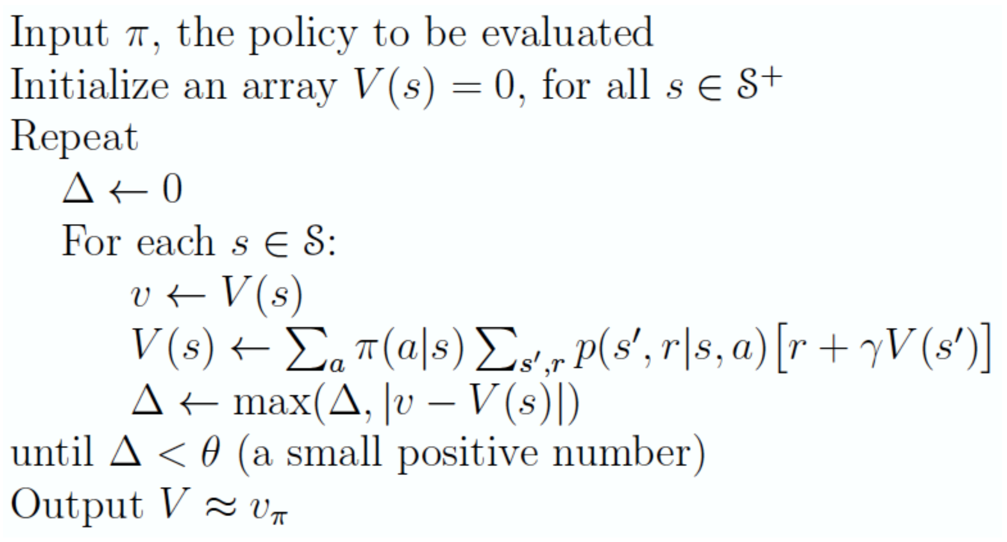
6 Reinforcement Learning

6.1 Dynamic Programming

6.2 Motivation - Why do you use dynamic programming to solve MDPs?

- MDPs have optimal substructure and overlapping subproblems in the recursive decomposition of Bellman equations.
- Appropriately, dynamic programming is used for *planning* in a MDP to predict and to control.
  - **Prediction**: Given a MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π, output a value function v_π.
  - **Control**: Given a MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, output the optimal value function v_* and the optimal policy π_*.
- *Caveats 1: Requires Full Knowledge of MDP.*
- *Caveats 2: Polynomial Time, Yet Astronomically Large Number of States.*

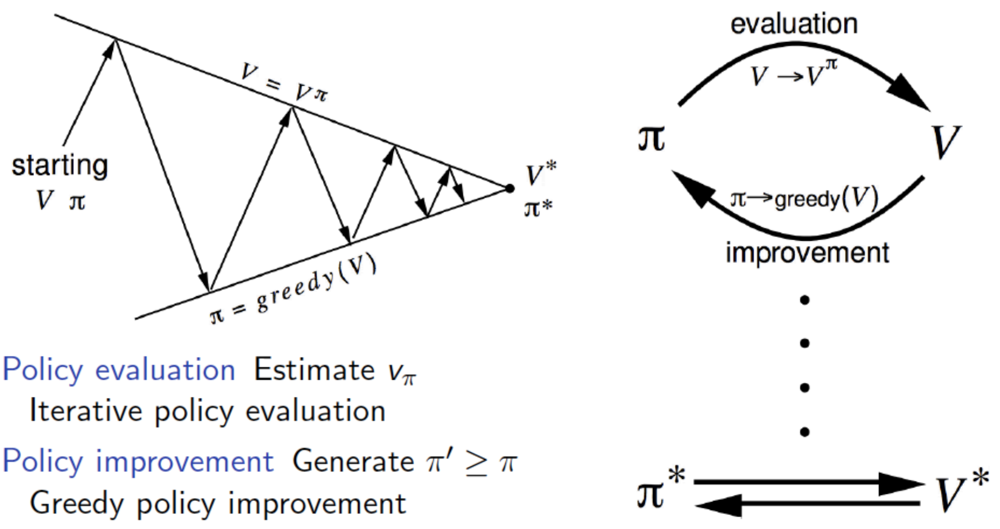
6.3 Policy Evaluation - How do you use dynamic programming to perform prediction?



```
Input $\pi$, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$
```

Policy Evaluation Algorithm

6.4 Policy Improvement Theorem - How does policy iteration converge to the optimal policy?

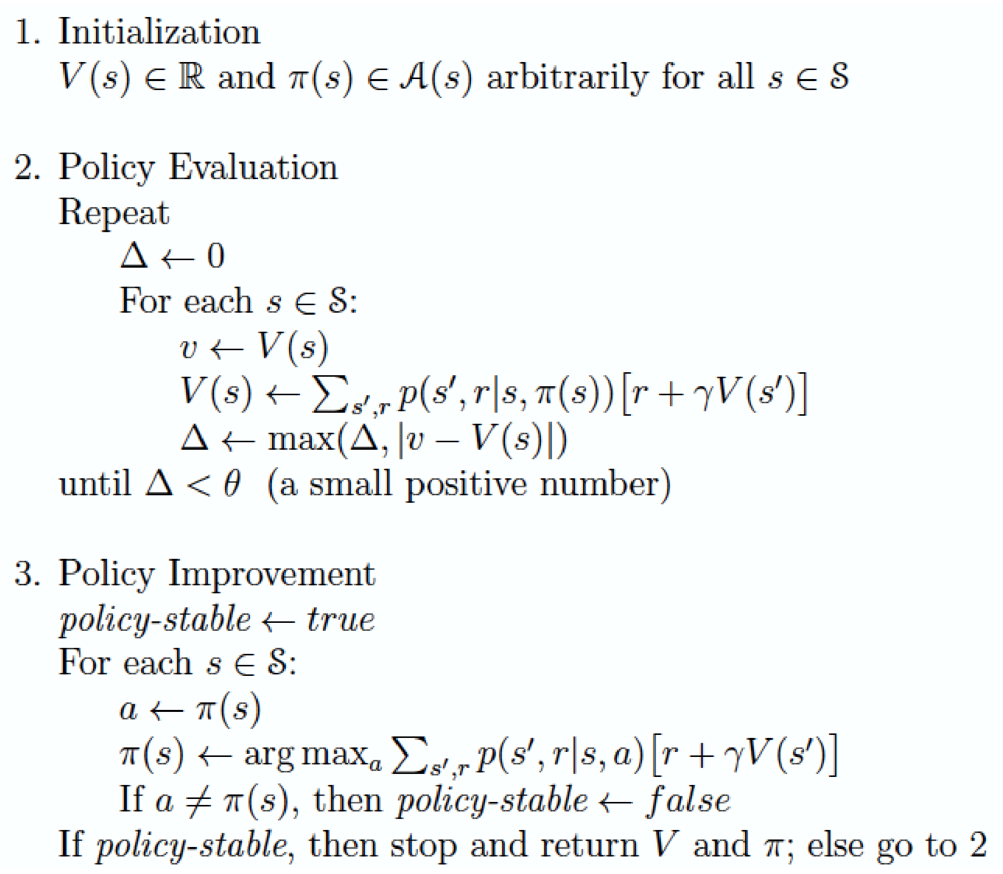


starting
V π
$V = v_\pi$
$\pi = \text{greedy}(V)$
V^*
π^*
Policy evaluation Estimate v_π
Iterative policy evaluation
Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement
evaluation
$V \to V^\pi$
π
V
$\pi \to \text{greedy}(V)$
improvement
π^*
V^*

Policy Improvement Theorem

- Given a policy π,
  1. **Evaluate** the policy π,
  $$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$
  2. **Improve** the policy by acting greedily with respect to v_π.
  $$\pi' = \text{Greedy}(v_\pi)$$
- The process of **policy iteration** has π' converge to π_* in a finite number of steps.

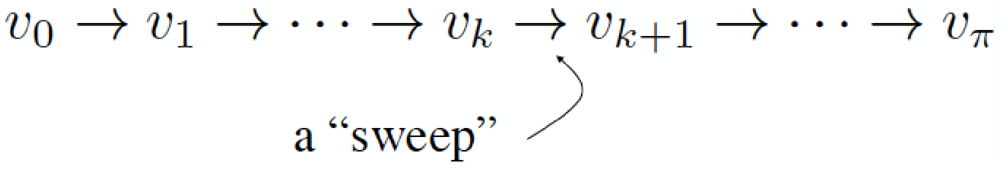
6.5 Policy Iteration - How do you use dynamic programming to perform control?



```
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

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

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
       $a \leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
       If $a \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V$ and $\pi$; else go to 2
```

Policy Iteration Algorithm

6.6 Backup Operations - What are sweeps and backups?


$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k \rightarrow v_{k+1} \rightarrow \cdots \rightarrow v_\pi$
a "sweep"

Backup and Sweeps

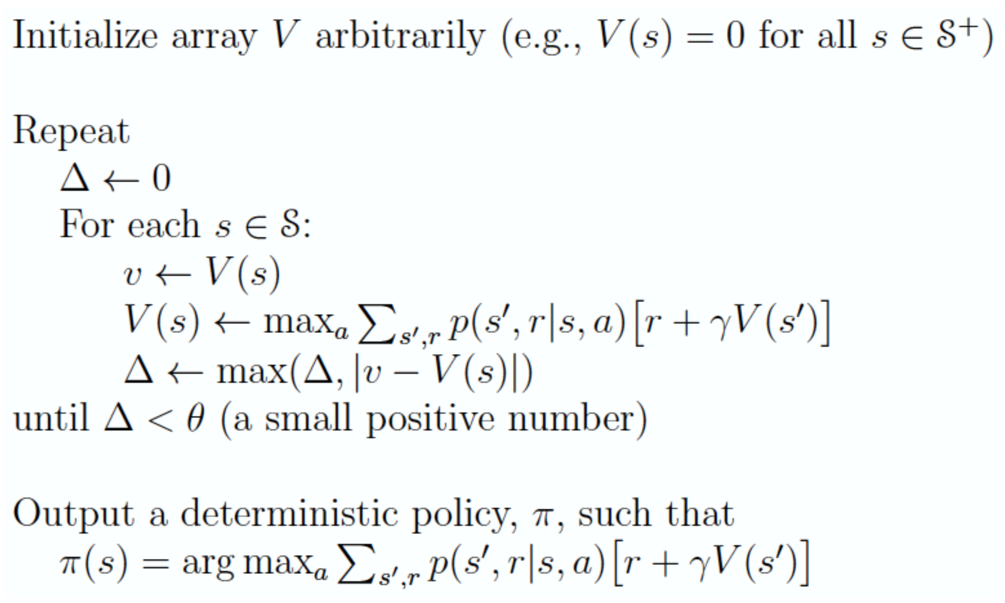
- A **backup** is a one-step lookahead, iterative application of the Bellman equation.
- A **sweep** is the iterative application of the backup operation to each state.
- **Full Policy-Evaluation Backup**:

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \cdot v_k(s')\right], \forall s \in S$$

- **Full Value-Iteration Backup**:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \cdot v_k(s')\right], \forall s \in S$$

6.7 Value Iteration - How do you use dynamic programming to output a deterministic policy?

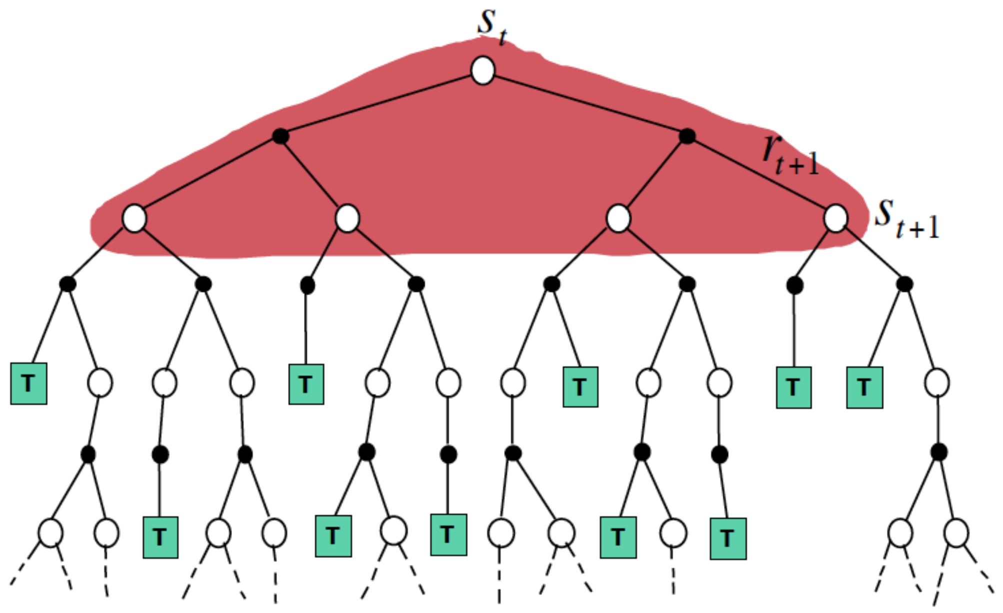


```
Initialize array $V$ arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

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

Value Iteration Algorithm

6.8 Backup Diagram - N/A

$$V(S_t) \leftarrow \mathbb{E}_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right]$$



Dynamic Programming Backup Diagram

- **Important Note**: DP considers all the choices at each state, so it suffers a performance dilemma.

6.9 Monte-Carlo Reinforcement Learning

6.10 Motivation - Why does Monte-Carlo reinforcement learning exist?

- **Goal**: How do you learn v_π from episodes of experience under policy π?
  - *i.e. Model-Free Prediction*
- **Approach**: Apply Monte-Carlo methods to learn v_π as the *empirical mean return* instead of *expected return*.
  - **Advantages**: *No Bootstrapping, Model-Free, Time Complexity Independent from Number of States*
  - **Disadvantages**: *All Episodes Must Terminate*

6.11 First-Visit Monte-Carlo Policy Evaluation - How does first-visit Monte-Carlo policy evaluation perform prediction?

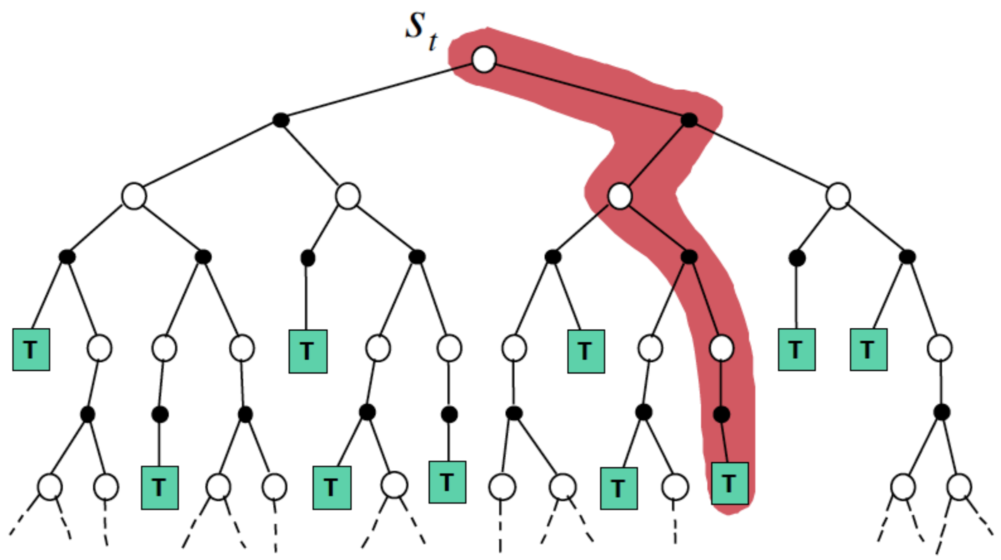
- To evaluate state s, the **first** time-step t that state s is visted in an episode,
  1. Increment counter $N(s) \leftarrow N(s) + 1$.
  2. Increment total return $S(s) \leftarrow S(s) + G_t$.
  3. Value is estimated by mean return $V(s) = \frac{S(s)}{N(s)}$.
- By the weak law of large numbers, $V(s) \to v_\pi(s)$ as $N(s) \to \infty$.

6.12 Every-Visit Monte-Carlo Policy Evaluation - How does every-visit Monte-Carlo policy evaluation perform prediction?

- To evaluate state s, the **every** time-step t that state s is visted in an episode,
  1. Increment counter $N(s) \leftarrow N(s) + 1$.
  2. Increment total return $S(s) \leftarrow S(s) + G_t$.
  3. Value is estimated by mean return $V(s) = \frac{S(s)}{N(s)}$.
- By the weak law of large numbers, $V(s) \to v_\pi(s)$ as $N(s) \to \infty$.

6.13 Backup Diagram - N/A

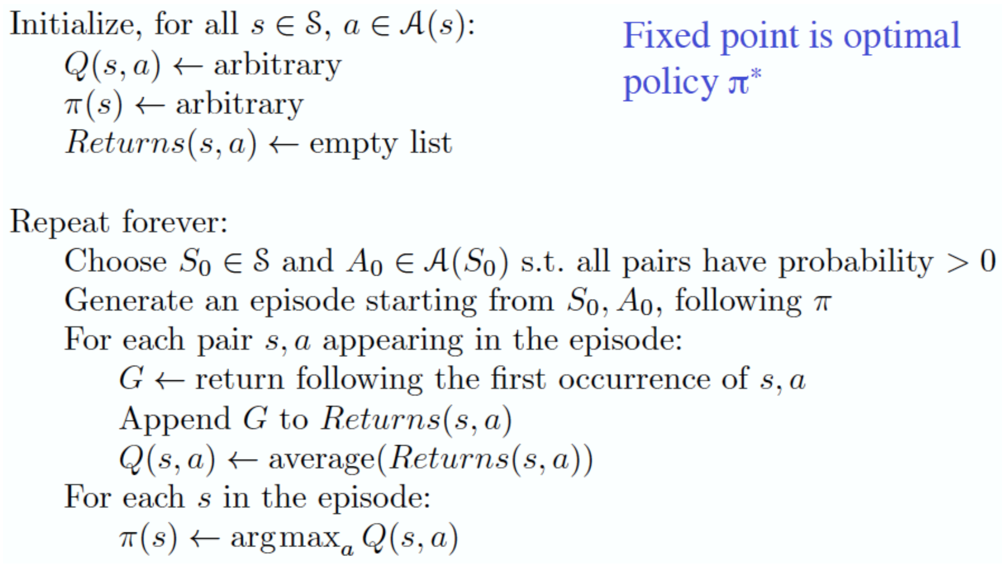
$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$



Monte-Carlo Backup Diagram

- **Important Note**: MC considers only one choice at each state, so it suffers from an explore/exploit dilemma.

6.14 Monte-Carlo Exploring Starts - How do you use Monte-Carlo Exploring Starts method to perform control?



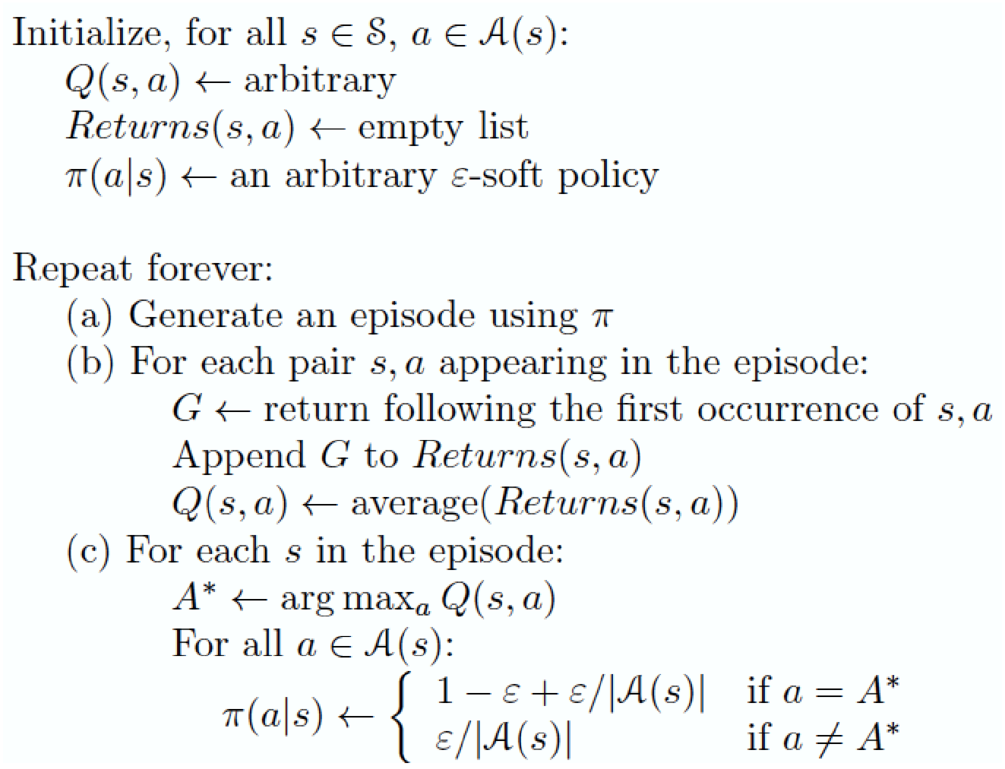
```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s, a) \leftarrow$ arbitrary
    $\pi(s) \leftarrow$ arbitrary
    $Returns(s, a) \leftarrow$ empty list

Repeat forever:
    Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability $> 0$
    Generate an episode starting from $S_0, A_0$, following $\pi$
    For each pair $s, a$ appearing in the episode:
        $G \leftarrow$ return following the first occurrence of $s, a$
        Append $G$ to $Returns(s, a)$
        $Q(s, a) \leftarrow$ average$(Returns(s, a))$
    For each $s$ in the episode:
        $\pi(s) \leftarrow \arg\max_a Q(s, a)$
```

Fixed point is optimal policy π^*

Monte-Carlo Exploring Starts Methods

- **Exploring Starts**: Requires every state-action pair to have a non-zero probability of being the starting pair.
  - **Advantages**: *Asymptotically Converges*
  - **Disadvantages**: *Restricts Modelling*

6.15 On-Policy Monte-Carlo - How do you use On-Policy Monte-Carlo method to perform control?

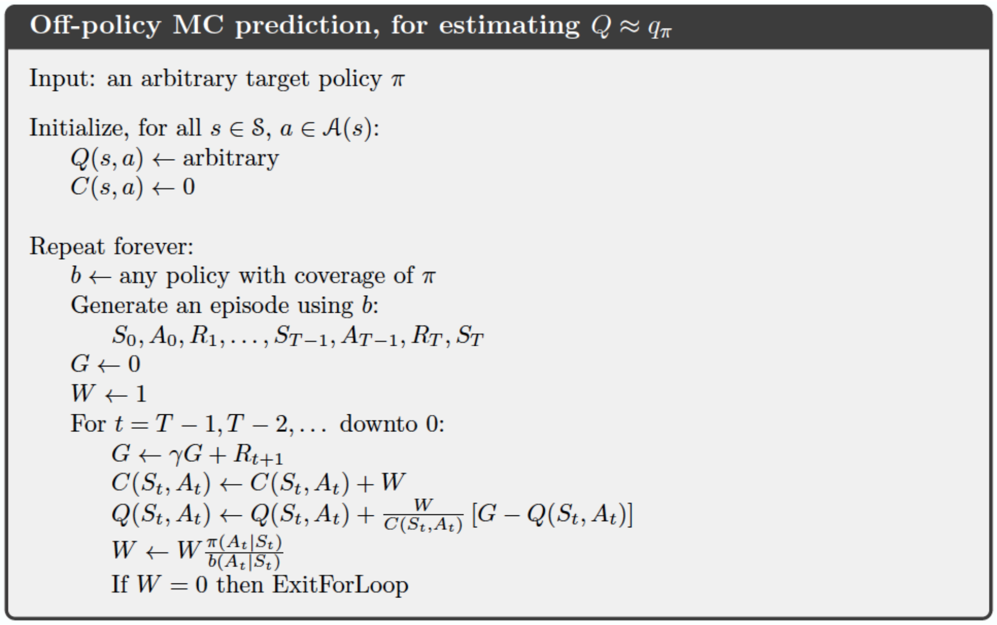


```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $Returns(s,a) \leftarrow$ empty list
    $\pi(a|s) \leftarrow$ an arbitrary $\varepsilon$-soft policy

Repeat forever:
    (a) Generate an episode using $\pi$
    (b) For each pair $s, a$ appearing in the episode:
            $G \leftarrow$ return following the first occurrence of $s, a$
            Append $G$ to $Returns(s,a)$
            $Q(s,a) \leftarrow$ average($Returns(s,a)$)
    (c) For each $s$ in the episode:
            $A^* \leftarrow \arg\max_a Q(s,a)$
            For all $a \in \mathcal{A}(s)$:
                $\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$
```

On-Policy Monte-Carlo Method

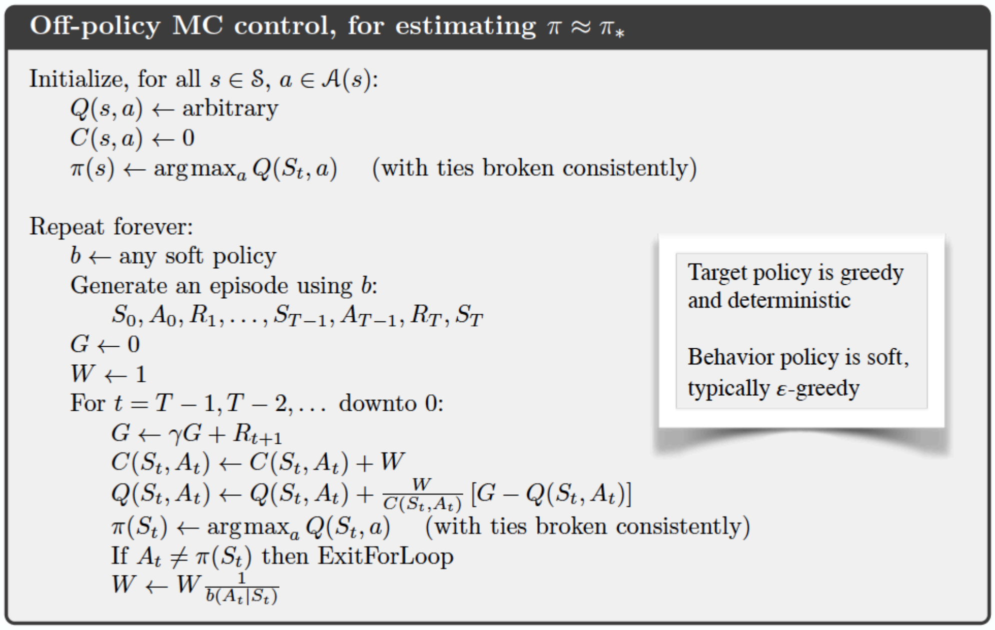
- **On-Policy Learning**: Learn about policy π from experience sampled from π.
  - *i.e. Learn on the job.*
- **Soft Policy**: Requires every state-action pair to have a non-zero probability.
  - **Advantages**: *Removes Exploring Starts*

6.16 Off-Policy Monte Carlo - How do you use Off-Policy Monte-Carlo method to perform control?



```
Off-policy MC prediction, for estimating $Q \approx q_\pi$

Input: an arbitrary target policy $\pi$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $C(s,a) \leftarrow 0$

Repeat forever:
    $b \leftarrow$ any policy with coverage of $\pi$
    Generate an episode using $b$:
        $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    For $t = T-1, T-2, \ldots$ downto 0:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t,A_t)} [G - Q(S_t, A_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
        If $W = 0$ then ExitForLoop
```

Off-Policy Monte-Carlo Method 1



```
Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $C(s,a) \leftarrow 0$
    $\pi(s) \leftarrow \arg\max_a Q(S_t, a)$    (with ties broken consistently)

Repeat forever:
    $b \leftarrow$ any soft policy
    Generate an episode using $b$:
        $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    For $t = T-1, T-2, \ldots$ downto 0:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t,A_t)} [G - Q(S_t, A_t)]$
        $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$    (with ties broken consistently)
        If $A_t \neq \pi(S_t)$ then ExitForLoop
        $W \leftarrow W \frac{1}{b(A_t|S_t)}$
```

Target policy is greedy and deterministic

Behavior policy is soft, typically ε-greedy

Off-Policy Monte-Carlo Method 2

- **Off-Policy Learning**: Learn about policy π from experience sampled from behavior policy b.
    - *i.e. Look over someone's shoulder.*

- **Behavior Policy**: Requires b generates behavior that covers, or includes, π.

$$b(a \mid s) > 0 \implies \pi(a \mid s) > 0, \forall a \in \mathcal{A}, s \in \mathcal{S}$$

  - **Question**: How do you extract/emphasize π from b?
  - **Answer**: Use importance sampling.

- **Importance Sampling**: Weigh each return by the ratio of the probabilities of the trajectory under the two policies.

6.17 Importance Sampling - How do you calculate state-value function using importance sampling?

- **Importance Sampling Ratio**: A relative probability of the trajectory under the two policies.

$$\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)}$$

$$\mathbb{E}\left[\frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)}\right] = 1$$

6.18 Monte-Carlo Methods vs. Dynamic Programming - What are the advantages and disadvantages of Monte-Carlo methods?

6.18.1 Advantages

- *Learn Directly from Interactions with Environment*
- *Does Not Require Full Models*
- *Does Not Require Learning About All States (No Bootstrapping)*
- *Less Harmed by Violating Markov Property*

6.18.2 Disadvantages

- *Requires Sufficient Exploration*
- *Requires Exploring Starts or Soft Policies*

6.19 Temporal Difference Reinforcement Learning

6.20 Motivation - Why does temporal difference reinforcement learning exist?

- **Goal**: How do you learn v_π from incomplete episodes of experience under policy π?
  - *i.e. Model-Free Prediction*

- **Approach**: Update an estimate of v_π with an estimated return $G_t^{(n)}$ such as $G_t^{(1)} = R_{t+1} + \gamma \cdot V(S_{t+1})$.
  - **Advantages**: *Model-Free, Time Complexity Independent from Number of States, Incomplete Episodes*
  - **Disadvantages**: *Bootstrapping*

6.21 TD(0) Policy Evaluation - How does TD(0) policy evaluation perform prediction?

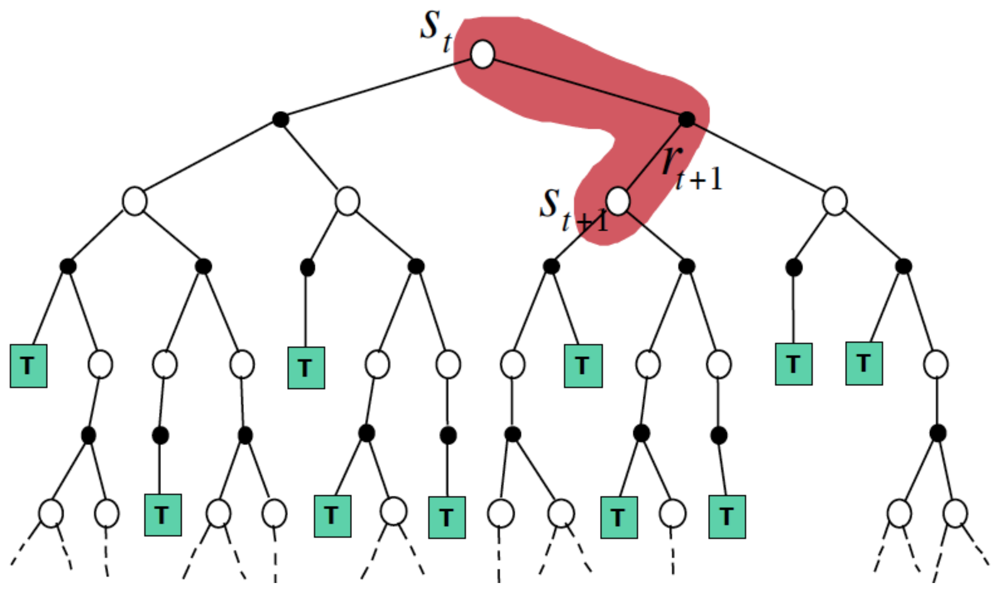
- Update value $V(S_t)$ toward the estimated return $R_{t+1} + \gamma \cdot V(S_{t+1})$.

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma \cdot V(S_{t+1}) - V(S_t)\right)$$

  - Where α is a constant step-size.
    * e.g., $\alpha = \frac{1}{N(s=S_t)}$
  - Where $R_{t+1} + \gamma \cdot V(S_{t+1})$ is the **TD target**.
  - Where $\delta_t = R_{t+1} + \gamma \cdot V(S_{t+1}) - V(S_t)$ is the **TD error**.

6.22 Backup Diagram - N/A

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



Temporal Difference Backup Diagram

6.23 Temporal Difference vs. Monte-Carlo - What are the advantages and disadvantages of temporal difference methods?

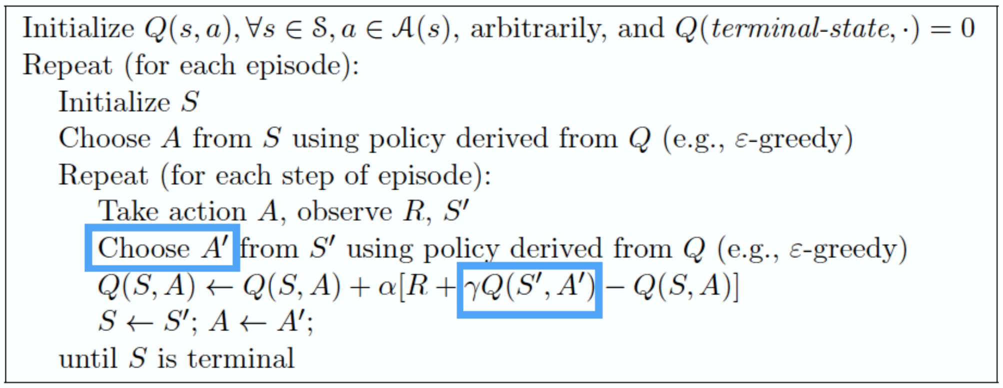
6.23.1 Advantages

- *Learn Without/Before Final Outcome*
- *Learn From Incomplete/Non-Terminating Sequences*
- *Low Variance*
- *Memory Efficienct*
- *Exploits Markov Property* $\implies$ *Less Error & Faster Convergence*

6.23.2 Disadvantages

- *Some Bias*
- *More Sensitive to Initial Parameters*

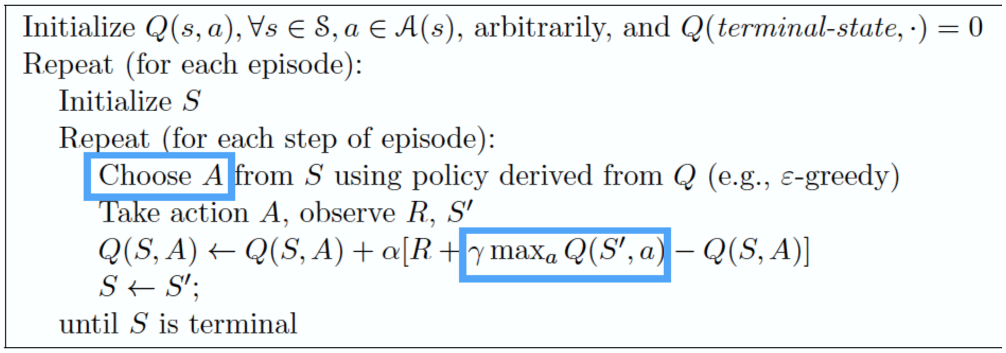
6.24 SARSA - How is on-policy TD control achieved?



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        $\boxed{\text{Choose } A'}$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \boxed{\gamma Q(S',A')} - Q(S,A)]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

SARSA Algorithm

6.25 Q-Learning - How is off-policy TD control achieved?



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        $\boxed{\text{Choose } A}$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \boxed{\gamma \max_a Q(S',a)} - Q(S,A)]$
        $S \leftarrow S'$;
    until $S$ is terminal
```

Q-Learning Algorithm

6.26 Expected SARSA - N/A

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \cdot \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\right]$$

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \cdot \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

6.27 SARSA vs. Q-Learning - What are the similarities and differences between SARSA and Q-Learning?

6.27.1 SARSA

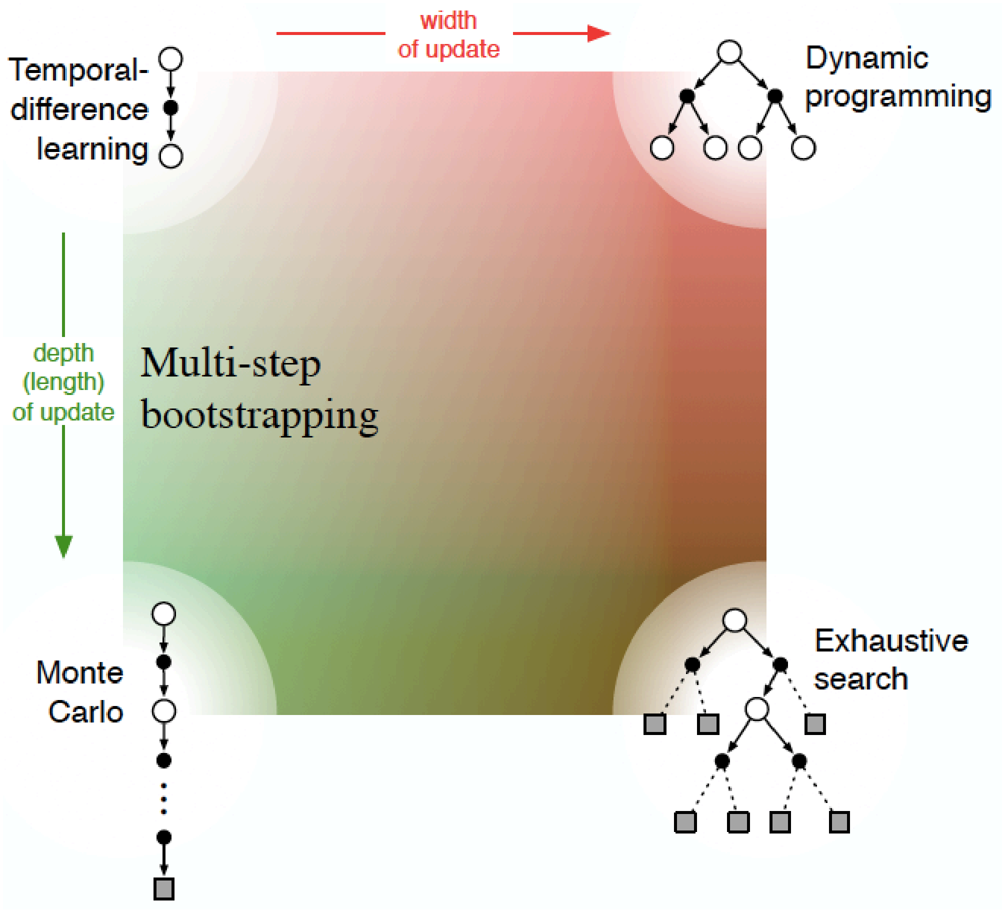
- *On-Policy*
- *Chooses Action, Sees Result*
- *Update It's Value Function With Result*
- *Yes Guarantee Convergence*

6.27.2 Q-Learning

- *Off-Policy*
- *Chooses Action, Sees Result*
- *Update It's Value Function With Different Action*
- *No Guarantee Convergence*

6.28 Multi-Step Reinforcement Learning

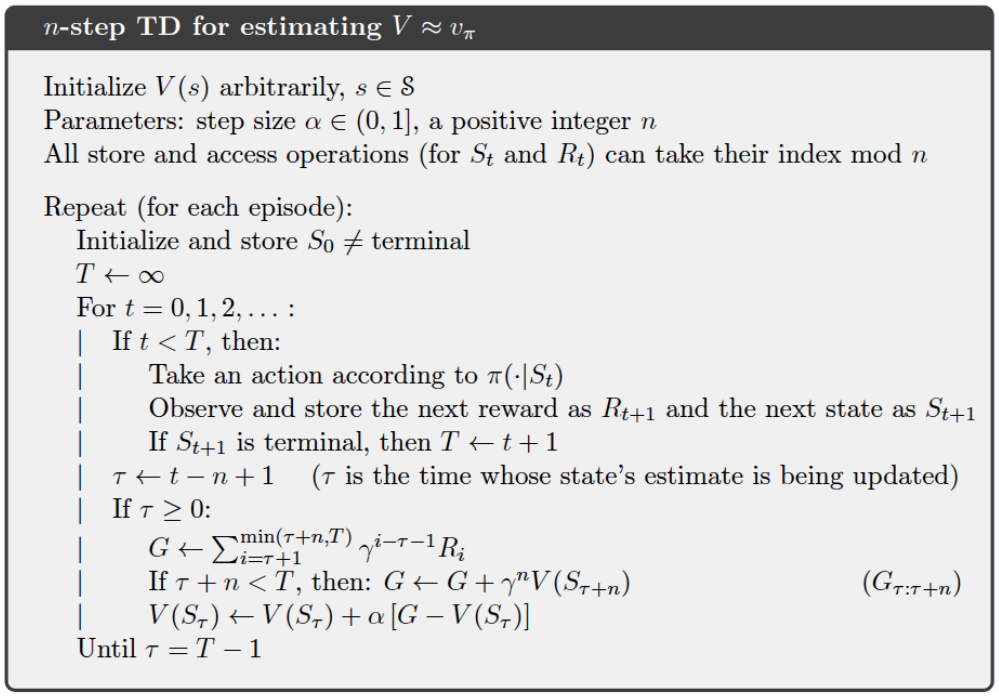
6.29 Unified View of Reinforcement Learning - How are MC and TD methods of reinforcement learning related?



Unified View of Reinforcement Learning

- **MC**: $n = \infty$-Steps of Predictions.
  - $G_t = R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + ... + \gamma^{T-t-1} \cdot R_T$
- **TD**: $n = 1$-Steps of Predictions.
  - $G_{t:t+1} = R_{t+1} + \gamma \cdot V_t(S_{t+1})$

6.30 *n*-Step TD - How do you use *n*-step TD to perform prediction?



```
n-step TD for estimating $V \approx v_\pi$

Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0, 1]$, a positive integer $n$
All store and access operations (for $S_t$ and $R_t$) can take their index mod $n$

Repeat (for each episode):
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    For $t = 0, 1, 2, \ldots$ :
    |   If $t < T$, then:
    |       Take an action according to $\pi(\cdot|S_t)$
    |       Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
    |       If $S_{t+1}$ is terminal, then $T \leftarrow t + 1$
    |   $\tau \leftarrow t - n + 1$    ($\tau$ is the time whose state's estimate is being updated)
    |   If $\tau \geq 0$:
    |       $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    |       If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$          $(G_{\tau:\tau+n})$
    |       $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$
    Until $\tau = T - 1$
```

n-Step TD Algorithm

- ***n*-Step Return** (*Forwards*): $G_{t:t+n} = R_{t+1} + \gamma \cdot R_{t+2} + ... + \gamma^{n-1} \cdot R_{t+n} + \gamma^n \cdot V_{t+n-1}(S_{t+n})$
- ***n*-Step TD** (*Backwards*): $V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha\,[G_{t:t+n} - V_{t+n-1}(S_t)]$
- **Important Note 1**: The last n states must be kept in memory.
- **Important Note 2**: Actual learning is delayed by n steps.

6.31 On-Policy *n*-Step Control - How does the *n*-step method change on-policy control methods?

6.31.1 *n*-Step SARSA

$$G_{t:t+n} = R_{t+1} + \gamma \cdot R_{t+2} + ... + \gamma^{n-1} \cdot R_{t+n} + \gamma^n \cdot Q_{t+n-1}(S_{t+n}, A_{t+n})$$
$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha\,[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$$

6.31.2 *n*-Step Expected SARSA

$$G_{t:t+n} = R_{t+1} + \gamma \cdot R_{t+2} + ... + \gamma^{n-1} \cdot R_{t+n} + \gamma^n \cdot \mathbb{E}[Q_{t+n-1}(S_{t+n}, A_{t+n}) \mid S_{t+n}]$$
$$G_{t:t+n} = R_{t+1} + \gamma \cdot R_{t+2} + ... + \gamma^{n-1} \cdot R_{t+n} + \gamma^n \cdot \sum_a \pi(a \mid S_{t+n}) Q_{t+n-1}(S_{t+n}, a)$$
$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha\,[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$$

6.32 Off-Policy n-Step Control - How does the n-step method change off-policy control methods?

6.32.1 Importance-Sampling Ratio

$$\rho_{t:h} = \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)}$$

6.32.2 Off-Policy n-Step TD

$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha\rho_{t:t+n-1}\left[G_{t:t+n} - V_{t+n-1}(S_t)\right]$ - Where $G_{t:t+n}$ is previously defined.

6.32.3 Off-Policy n-Step SARSA

$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha\rho_{t+1:t+n-1}\left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right]$ - Where $G_{t:t+n}$ is previously defined.

6.32.4 Off-Policy n-Step Expected SARSA

$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha\rho_{t+1:t+n-2}\left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right]$ - Where $G_{t:t+n}$ is previously defined.

6.33 λ-Return - What is λ-return?

- **λ-Return**: The return from averaging all n-step backups with λ^{n-1} weights.

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$$

$$\Delta V_t(s_t) = \alpha\left[R_t^\lambda - V_t(s_t)\right]$$

- If $\lambda = 1$, TD(1) is MC.
- If $\lambda = 0$, TD(0) is TD(0).
- If episodic, λ-return can be expressed by the following.

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t$$

  - Where the first term represents the return until termination.
  - Where the second term represents the return after termination.

6.34 Eligibility Trace - How do you use eligibility traces to compute TD(λ)?

- **Eligibility Trace**: A backward view mechanism for implementing TD(λ) by decaying all traces by $\gamma\lambda$ and increment the trace for the current state by 1, and accumulating the trace.

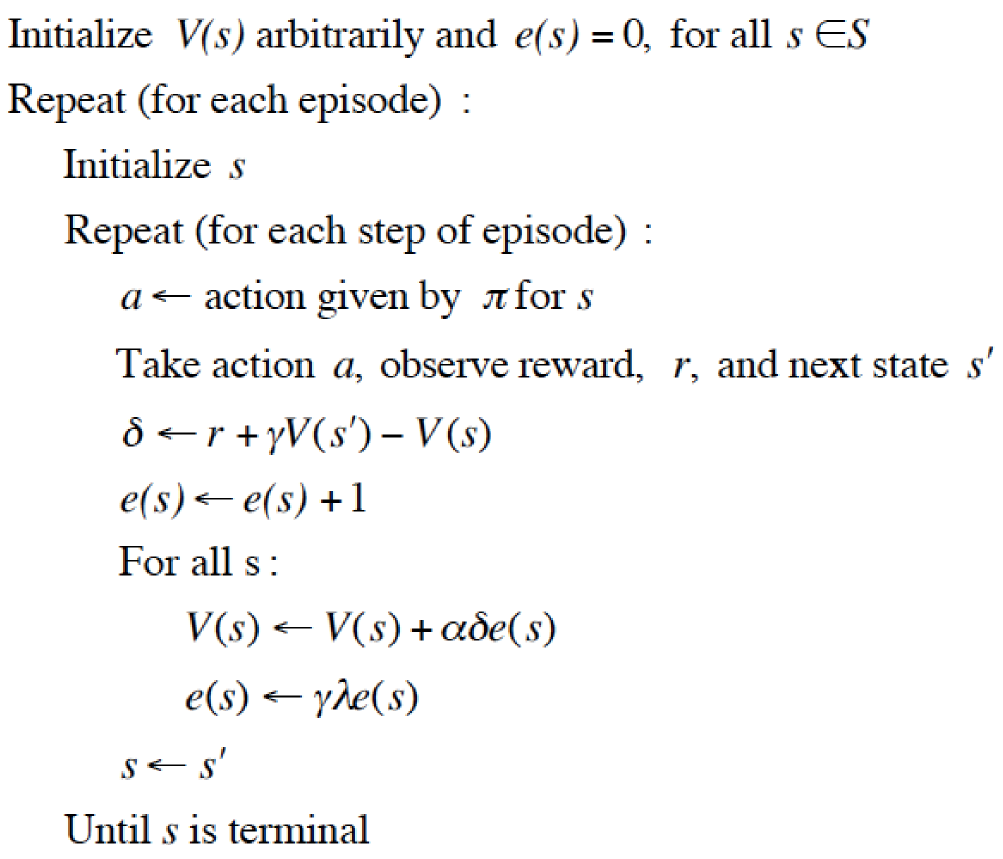
$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

$$V_{t+1}(s) = V_t(s) + \alpha\delta_t e_t(s)$$

- **Important Note**: Although TD(1) is MC, TD(1) executes incrementally and online.

6.35 Online Tabular TD(λ) Algorithm - How does λ extend TD for prediction?



```
Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in S$
Repeat (for each episode) :
    Initialize $s$
    Repeat (for each step of episode) :
        $a \leftarrow$ action given by $\pi$ for $s$
        Take action $a$, observe reward, $r$, and next state $s'$
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $e(s) \leftarrow e(s) + 1$
        For all s :
            $V(s) \leftarrow V(s) + \alpha\delta e(s)$
            $e(s) \leftarrow \gamma\lambda e(s)$
        $s \leftarrow s'$
    Until $s$ is terminal
```

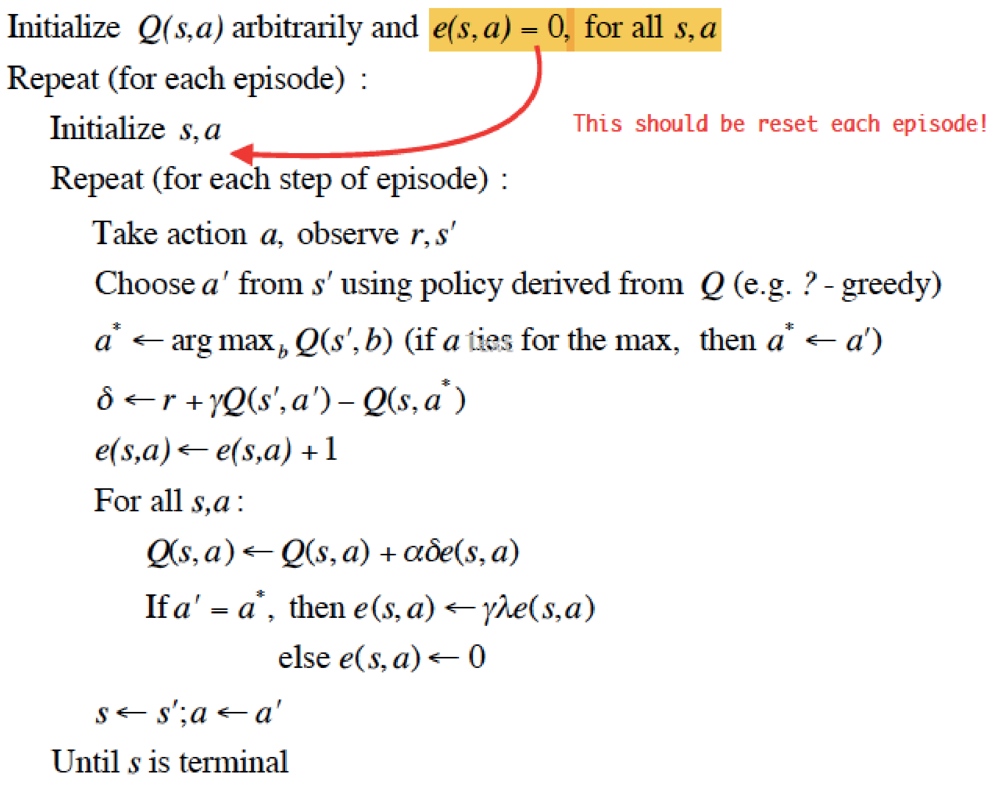
Online Tabular TD(λ) Algorithm

6.36 SARSA(λ) Algorithm - How does λ extend SARSA for on-policy control?

```
Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all $s, a$
Repeat (for each episode) :
    Initialize $s, a$
    Repeat (for each step of episode) :
        Take action $a$, observe $r, s'$
        Choose $a'$ from $s'$ using policy derived from $Q$ (e.g. ? - greedy)
        $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
        $e(s,a) \leftarrow e(s,a) + 1$
        For all $s,a$ :
            $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
            $e(s, a) \leftarrow \gamma\lambda e(s, a)$
        $s \leftarrow s'; a \leftarrow a'$
    Until $s$ is terminal
```

SARSA(λ) Algorithm

6.37 Watkin's Q(λ) Algorithm - How does Watkin's algorithm achieve off-policy control?



```
Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all $s,a$
Repeat (for each episode) :
  Initialize $s, a$
  Repeat (for each step of episode) :
    Take action $a$, observe $r, s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g. ? - greedy)
    $a^* \leftarrow \arg\max_b Q(s', b)$ (if $a$ ties for the max, then $a^* \leftarrow a'$)
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a^*)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all $s,a$ :
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      If $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
                 else $e(s,a) \leftarrow 0$
    $s \leftarrow s'; a \leftarrow a'$
  Until $s$ is terminal
```

This should be reset each episode!

Watkin's Q(λ) Algorithm

1. Zero out eligibility trace after a non-greedy action.
2. Take the maximum when backing up at the first non-greedy choice.

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t, a = a_t, Q_{t-1}(s_t,a_t) = \max_a Q_{t-1}(s_t,a) \\ 0 & \text{if } Q_{t-1}(s_t,a_t) \neq \max_a Q_{t-1}(s_t,a) \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} \end{cases}$$

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)$$

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a)$$

6.37.1 Peng's Q(λ) Variant

1. Never zero out eligibility trace after a non-greedy action.
2. Take the maximum when backing up at the last non-greedy choice.

6.37.2 Naive Q(λ) Variant

1. Never zero out eligibility trace after a non-greedy action.
2. Take the maximum when backing up at current action.

6.38 Value Function Approximation

6.39 Motivation - Why does value function approximation exist?

- **Goal**: How do you represent a value function for a very large MDP whose states and/or actions cannot be all stored in memory inside a lookup table?
- **Approach**: Estimate the value function with function approximation, generalizing from known states to unknown states and updating parameter $\boldsymbol{w}$ using MC or TD learning.

$$\hat{v}(s,\boldsymbol{w}) \approx v_\pi(s)$$
$$\hat{q}(s,a,\boldsymbol{w}) \approx q_\pi(s,a)$$

6.39.1 Differentiable Function Approximators

- Linear Combination of Features
- Neural Network

6.40 Gradient Descent - What is gradient descent?

- Let $J(\boldsymbol{w})$ be a differentiable function of parameter vector $\boldsymbol{w}$.
- The **gradient** of $J(\boldsymbol{w})$ is the following.

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \begin{pmatrix} \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}_1} \\ \vdots \\ \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}_n} \end{pmatrix}$$

- **Gradient Descent**: Adjust $\boldsymbol{w}$ in the direction of the negative gradient to find a local minimum of $J(\boldsymbol{w})$.

$$\Delta \boldsymbol{w} = -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

  – Where α is a step-size parameter.

6.41 Stochastic Gradient Descent - How do you use stochastic gradient descent to approximate the value function?

1. Find a parameter vector $\boldsymbol{w}$ minimising mean-squared error between the *approximate value function* $\hat{v}(s,\boldsymbol{w})$ and the *true value function* $v_\pi(s)$.

$$J(\boldsymbol{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S,\boldsymbol{w}))^2 \right]$$

2. A gradient descent finds a local minimum of $J(\boldsymbol{w})$ such that the parameter vector $\boldsymbol{w}$ has the approximate value function close to the true value function.

$$\Delta \boldsymbol{w} = \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S,\boldsymbol{w})) \nabla_{\boldsymbol{w}} \hat{v}(S,\boldsymbol{w}) \right]$$

3. Stochastic gradient descent samples the gradient.

$$\Delta \boldsymbol{w} = \alpha \left(v_\pi(S) - \hat{v}(S,\boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{v}(S,\boldsymbol{w})$$

6.42 Linear Value Function Approximation - What is one approximate value function representation?

6.42.1 Linear Combination of Features

$$\hat{v}(S, \boldsymbol{w}) = \boldsymbol{x}(S)^T \boldsymbol{w} = \sum_{j=1}^{n} \boldsymbol{x}_j(S) \boldsymbol{w}_j$$

- Where $\boldsymbol{x}$ is a feature vector constructed from the original perceptron, the Least-Mean-Square (LMS) algorithm, Support Vector Machines (SVMs), or etc. . . .

6.42.2 Properties

1. The objective function is quadratic in parameters $\boldsymbol{w}$:

$$J(\boldsymbol{w}) = \mathbb{E}_\pi \left[\left(v_\pi(S) - \boldsymbol{x}(S)^T \boldsymbol{w} \right)^2 \right]$$

2. The stochastic gradient descent converges on the **GLOBAL OPTIMUM**.
3. The update rule is simple:

$$\begin{aligned} \nabla_{\boldsymbol{w}} \hat{v}(S, \boldsymbol{w}) &= \boldsymbol{x}(S) \\ \Delta \boldsymbol{w} &= \alpha \left(v_\pi(S) - \hat{v}(S, \boldsymbol{w}) \right) \boldsymbol{x}(S) \\ \text{Update} &= \text{Step-Size} \times \text{Prediction Error} \times \text{Feature Value} \end{aligned}$$

6.43 Incremental Prediction Algorithms - How do you substitute the true value function $v_\pi(s)$ in linear value function approximation?

- For MC, the substitute is the return G_t.

$$\Delta \boldsymbol{w} = \alpha \left(G_t - \hat{v}(S_t, w) \right) \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w})$$

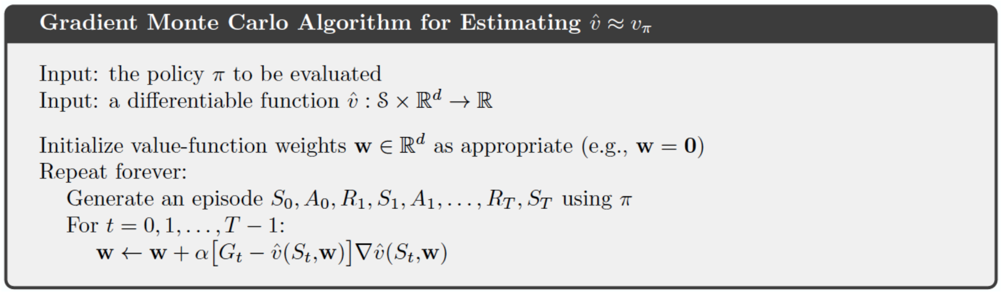
- For TD(0), the substitute is the TD target $R_{t+1} + \gamma \cdot \hat{v}(S_{t+1}, \boldsymbol{w})$.

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \cdot \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, w) \right) \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w})$$

- For TD(λ), the substitute is the λ-return G_t^λ.

$$\Delta \boldsymbol{w} = \alpha \left(G_t^\lambda - \hat{v}(S_t, w) \right) \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w})$$

6.44 Gradient MC - N/A

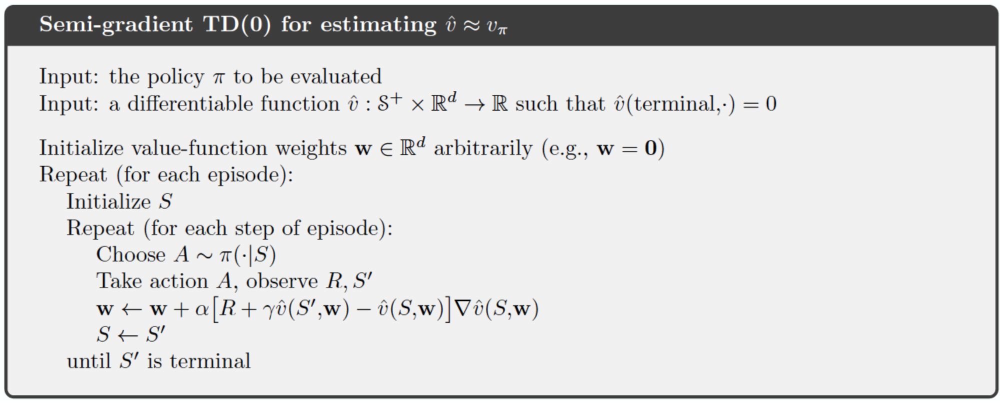


Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ as appropriate (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat forever:
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using $\pi$
    For $t = 0, 1, \ldots, T-1$:
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[ G_t - \hat{v}(S_t, \mathbf{w}) \big] \nabla \hat{v}(S_t, \mathbf{w})$
```

Gradient MC

6.45 Gradient TD - N/A



Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
   Initialize $S$
   Repeat (for each step of episode):
      Choose $A \sim \pi(\cdot|S)$
      Take action $A$, observe $R, S'$
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
      $S \leftarrow S'$
   until $S'$ is terminal
```

Gradient TD

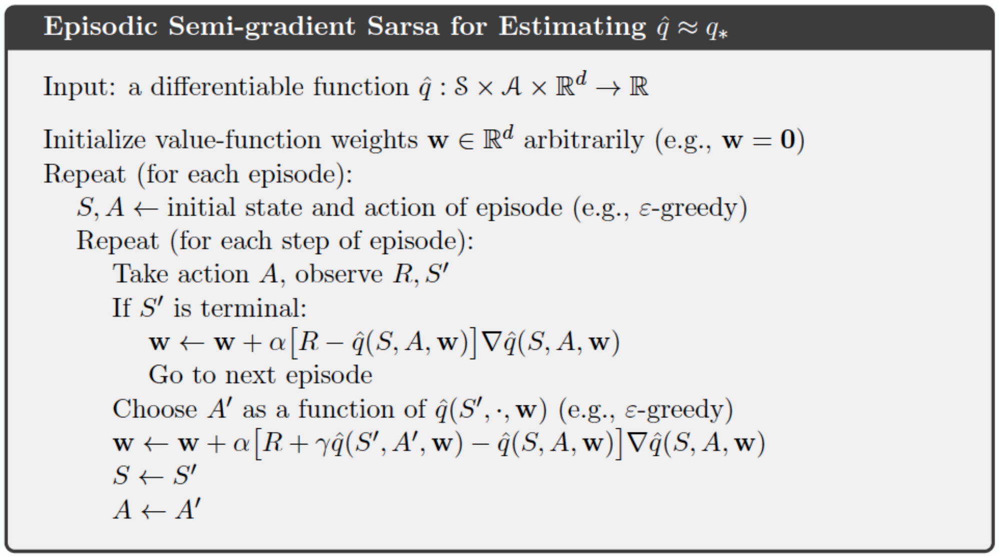
6.46 Gradient On-Policy Control - N/A

6.46.1 Learning Rule

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\left[U_t - \hat{q}(S_t, A_t, \boldsymbol{w}_t)\right]\nabla\hat{q}(S_t, A_t, \boldsymbol{w}_t)$$

1. **MC**: $U_t = G_t$
2. **SARSA**: $U_t = R_{t+1} + \gamma \cdot \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}_t)$
3. **Expected SARSA**: $U_t = R_{t+1} + \gamma \cdot \sum_\alpha \pi(\alpha \mid S_{t+1})\hat{q}(S_{t+1}, \alpha, \boldsymbol{w}_t)$

6.46.2 Example



Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

```
Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
   $S, A \leftarrow$ initial state and action of episode (e.g., $\varepsilon$-greedy)
   Repeat (for each step of episode):
      Take action $A$, observe $R, S'$
      If $S'$ is terminal:
         $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
         Go to next episode
      Choose $A'$ as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., $\varepsilon$-greedy)
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      $S \leftarrow S'$
      $A \leftarrow A'$
```

Gradient On-Policy Control

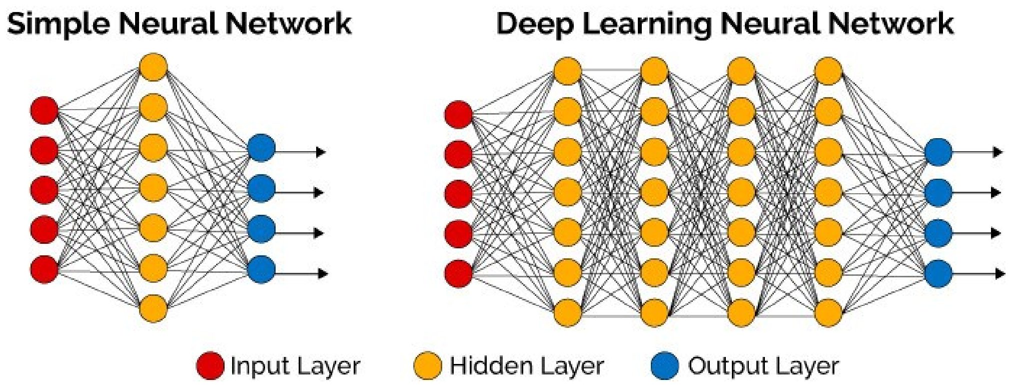
7 Deep Reinforcement Learning

7.1 Introduction to Neural Networks

7.2 Neural Networks - What is a neural network?

- A **neural network** connects many nonlinear (classically logistic/sigmoid) units together into a network to learn $f : \boldsymbol{X} \to \boldsymbol{Y}$.
  - Where f is a non-linear function.
  - Where $\boldsymbol{X}$ is a vector of continuous/discrete variables.
  - Where $\boldsymbol{Y}$ is a vector of continuous/discrete variables.

7.3 Three Layer Neural Network - What are the three layers in a basic neural network?



Neural Network

1. **Input Layer**: Each input unit collects one feature/dimension of the vector data and passes it to the (first) hidden layer.
2. **Hidden Layer**: Each hidden unit computes a weighted sum of all the units from the input later (or any previous layer) and passes it through a *nonlinear activation function*.

$$y_j = f(net_j)$$

3. **Output Layer**: Each output unit computes a weighted sum of all the hidden units and passes it through a (possibly nonlinear) *threshold function*.

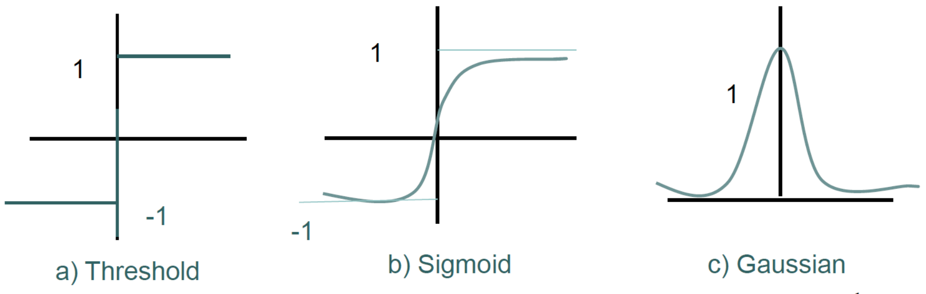
$$z_k = f(net_k)$$

7.4 Properties of Neural Networks - What are the properties of neural networks?

- Given a large enough layer of hidden units (or multiple layers), a NN can represent any function.
  - Too few units, the network will be under-parameterized and will not be able to learn complex functions.
  - Too many units, the network will be over-parameterized and will not be forced to learn a generlizable model.

- Connections between the units of all layers can be forward, backward, or both.
- Each unit can be fully or partially connected.
- Each unit has an activation function and a set of weighted connections.

7.5 Nonlinear Activation Function - What is a nonlinear activation function?



$$F_T(x) = \begin{cases} 1 & \mathit{if}\ x > \tau \\ -1 & \mathit{otherwise} \end{cases} \quad F_S(x) = \frac{1}{(1+e^{-cx})} \quad F_G(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

Nonlinear Activation Functions

- A hidden unit emits an output that is a **nonlinear activation function** of its net activation.

$$y_j = f(net_j) = f\left(\sum_{i=0}^{d} x_i w_{ji}\right)$$

  - Where i indexes the input units.
  - Where j indexes the hidden units.
  - Where w_{ji} denotes the synaptic weights.
  - Outputs are thresholded through a nonlinear activation function.

- **Rectified Linear Units** (*ReLU*): The standard nonlinear activation function: $y_j = \max(0, net_j)$.
  - Easily distinguishes strong signals.
  - Values are mostly zero.
  - Derivative is mostly zero.
  - Does not easily saturate compared to the sigmoid function.

7.6 Backpropagation Algorithm - How does the backpropagation algorithm iteratively compute the gradient to update network weights?

- **Important Idea**: Minimize the loss function (mean-squared errors) on training examples.

1. **Forward Propagation**: Input the training data to the network and compute outputs.
2. Compute output unit errors.
$$\delta_k' = o_k'(1 - o_k')(y_k' - o_k')$$
3. Compute hidden unit errors.
$$\delta_h' = o_h'(1 - o_h') \sum_k w_{h,k} \delta_k'$$

4. Update network weights.
$$w_{i,j} = w_{i,j} + \Delta w'_{i,j} = w_{i,j} + \eta \delta'_j o'_i$$

- Where y is the target output of a unit.
- Where o is the actual output of a unit.
- Where η is the training rate.

7.7 Convolutional Neural Network - What is a CNN?

- **Convolutional Neural Network** (*CNN*): A neural network that uses convolution in place of general matrix multiplication in at least one of its layers.

7.7.1 Additional Properties

1. Sparse Connectivity.
2. Parameter Sharing.
3. Equivariant Representations.

7.8 Deep Q-Networks (DQN)

7.9 Motivation - What are the stability issues with naive deep Q-learning?

- A naive deep Q-learning approximates the action-value function by using a deep Q-network. However, it suffers from divergent and oscillating behavior.
    1. As data is sequential, the successive samples are correlated and not i.i.d., yet NNs require i.i.d. samples.
    2. Accordingly, the learned policy may oscillate with slight changes to Q-values.
    3. Finally, the scales of rewards and Q-values are unknown such that naive deep Q-learning gradients can be unstable when backpropagated.

7.10 Overview - What is a DQN?

1. *Use Experience Replay*:
    - Provides i.i.d. samples.
2. *Freeze Target Q-Network*:
    - Avoids oscillations.
3. *Clip Rewards / Normalize Network to Sensible Range*:
    - Stabilizes gradients.

7.11 Experience Replay - How does experience replay remove correlations to provide i.i.d. samples?

- **Important Idea**: To remove correlations, sample from the agent's own experience.

1. Take action a_t according to ϵ-greedy policy.
2. Store the transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in the agent's replay memory $\mathcal{D}$.

3. Sample a random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$.
4. Optimise the mean squared error between the Q-network and the Q-learning targets.

$$\mathcal{L}(\boldsymbol{w}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \cdot \max_{a'} Q(s', a', \boldsymbol{w}) - Q(s, a, \boldsymbol{w}) \right)^2 \right]$$

7.12 Frozen/Fixed Target Q-Network - How does freezing/fixing the target Q-network avoid oscillations?

- **Important Idea**: To avoid oscillations, freeze/fix parameters used in the Q-learning target.

1. Compute the Q-learning targets with respect to the old, frozen/fixed parameters $\boldsymbol{w}^-$.

$$r + \gamma \cdot \max_{a'} Q(s', a', \boldsymbol{w}^-)$$

2. Optimise the mean squared error between the Q-network and the Q-learning targets.

$$\mathcal{L}(\boldsymbol{w}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \cdot \max_{a'} Q(s', a', \boldsymbol{w}^-) - Q(s, a, \boldsymbol{w}) \right)^2 \right]$$

3. Periodically, update the frozen/fixed parameters: $\boldsymbol{w}^- \leftarrow \boldsymbol{w}$.

7.13 Clip Rewards - How does clipping rewards stabilize gradients?

- **Important Idea**: To stabilize gradients, DQN clips rewards to a sensible range of $[-1, +1]$ such that Q-values can never become too large, ensuring that gradients are well-conditioned.

7.14 Policy Gradient

7.15 Motivation - Why should you approximate policies versus action-values?

7.15.1 Advantages

- *Better Convergence Properties*
- *High-Dimensional Action Spaces*
- *Continuous Action Spaces*
- *Stochastic Policies; e.g., Rock-Paper-Scissors*

7.15.2 Disadvantages

- *Typically, Converge Local Optimum vs. Global Optimum*
- *Typically, Inefficient Policy Evaluation*
- *Typically, High Variance Policy Evaluation*

7.16 Policy Objective Functions - What policy objective functions can be used for a given type of environment?

- **Goal**: Given a policy $\pi(a \mid s, \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$, find best $\boldsymbol{\theta}$.
- **Episodic Environments**: *Use Start Value.*

$$J_1(\boldsymbol{\theta}) = V_\pi(s_1) = \mathbb{E}_\pi[v_1]$$

- **Continuing Environments**:
  - *Use Average Value.*
$$J_{avV}(\boldsymbol{\theta}) = \sum_s d^\pi(s) V^\pi(s)$$
  - *Use Average Reward Per Time-Step.*
$$J_{avR}(\boldsymbol{\theta}) = \sum_s d^\pi(s) \sum_a \pi(a \mid s, \boldsymbol{\theta}) \mathcal{R}_s^a$$
  - Where $d^\pi(s)$ is the stationary distribution of Markov chain for $\pi_{\boldsymbol{\theta}}$.

7.17 Policy Gradient - How does policy gradient algorithms achieve policy based reinforcement learning?

- **Important Idea**: *Policy Based Reinforcement Learning = Finding $\boldsymbol{\theta}$ That Maximises $J(\boldsymbol{\theta})$.
- **Policy gradient algorithms** search for a local maximum in $J(\boldsymbol{\theta})$ by ascending the gradient of the policy, with respect to parameters $\boldsymbol{\theta}$.
$$\Delta\boldsymbol{\theta} = \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$
  - Where α is a step-size parameter.

7.18 Policy Gradient Theorem - How do you compute the policy gradient analytically?

7.18.1 Likelihood Ratios

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \pi(a \mid s, \boldsymbol{\theta}) &= \pi(a \mid s, \boldsymbol{\theta}) \frac{\nabla_{\boldsymbol{\theta}} \pi(a \mid s, \boldsymbol{\theta})}{\pi(a \mid s, \boldsymbol{\theta})} \\ &= \pi(a \mid s, \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log \pi(a \mid s, \boldsymbol{\theta})\end{aligned}$$

7.18.2 Softmax Policy (*Popular*)

1. Weight actions using a linear combination of features $\phi(s,a)^T\boldsymbol{\theta}$.
2. Accordingly, the probability of an action is proportional to its exponentiated weight.
$$\pi(a \mid s, \boldsymbol{\theta}) \propto \exp(\phi(s,a)^T\boldsymbol{\theta})$$
3. Thus, the *score function* is the following.
$$\nabla_{\boldsymbol{\theta}} \log \pi(a \mid s, \boldsymbol{\theta}) = \phi(s,a) - \mathbb{E}_\pi[\phi(s,*)]$$

7.18.3 Gaussian Policy (*Continuous*)

1. Let the mean be a linear combination of state features $\mu(s) = \phi(s)^T\boldsymbol{\theta}$.
2. Let the variance be fixed/parameterised σ^2.
3. Accordingly, an action is a Gaussian random variable.
$$a \sim \mathcal{N}(\mu(s), \sigma^2)$$
4. Thus, the *score function* is the following.
$$\nabla_{\boldsymbol{\theta}} \log \pi(a \mid s, \boldsymbol{\theta}) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$

7.18.4 Theorem

- For any differentiable policy $\pi(a \mid s, \boldsymbol{\theta})$ and for any of the policy objective functions $J = J_1$, J_{avR}, or $\frac{1}{1-\gamma} J_{avV}$, the policy gradient is the following.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[\nabla_{\boldsymbol{\theta}} \log \pi(a \mid s, \boldsymbol{\theta}) \cdot Q^{\pi_{\boldsymbol{\theta}}}(s, a) \right]$$

7.19 Monte-Carlo Policy Gradient (REINFORCE) - How does REINFORCE apply the policy gradient theorem?

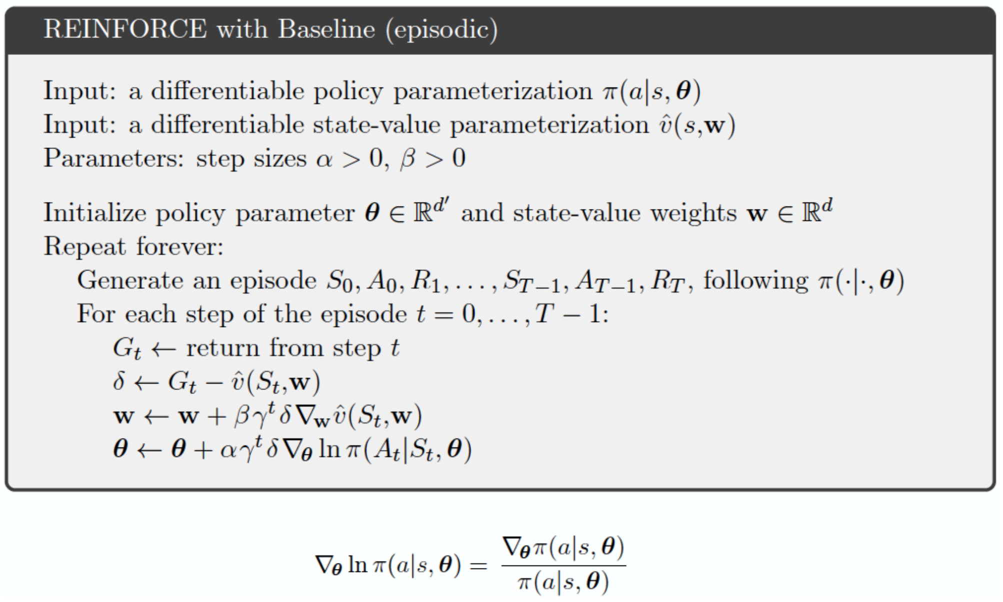
$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

```
function REINFORCE
    Initialise $\theta$ arbitrarily
    for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ do
        for $t = 1$ to $T - 1$ do
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        end for
    end for
    return $\theta$
end function
```

REINFORCE Algorithm

- It updates parameters using stochastic gradient ascent.
- It uses the return v_t as an unbiased sample of $Q^{\pi_{\boldsymbol{\theta}}}(s_t, a_t)$.
- **Problem**: The Monte-Carlo policy gradient has high variance.

7.20 Monte-Carlo Policy Gradient (REINFORCE with Baseline) - N/A



REINFORCE with Baseline (episodic)

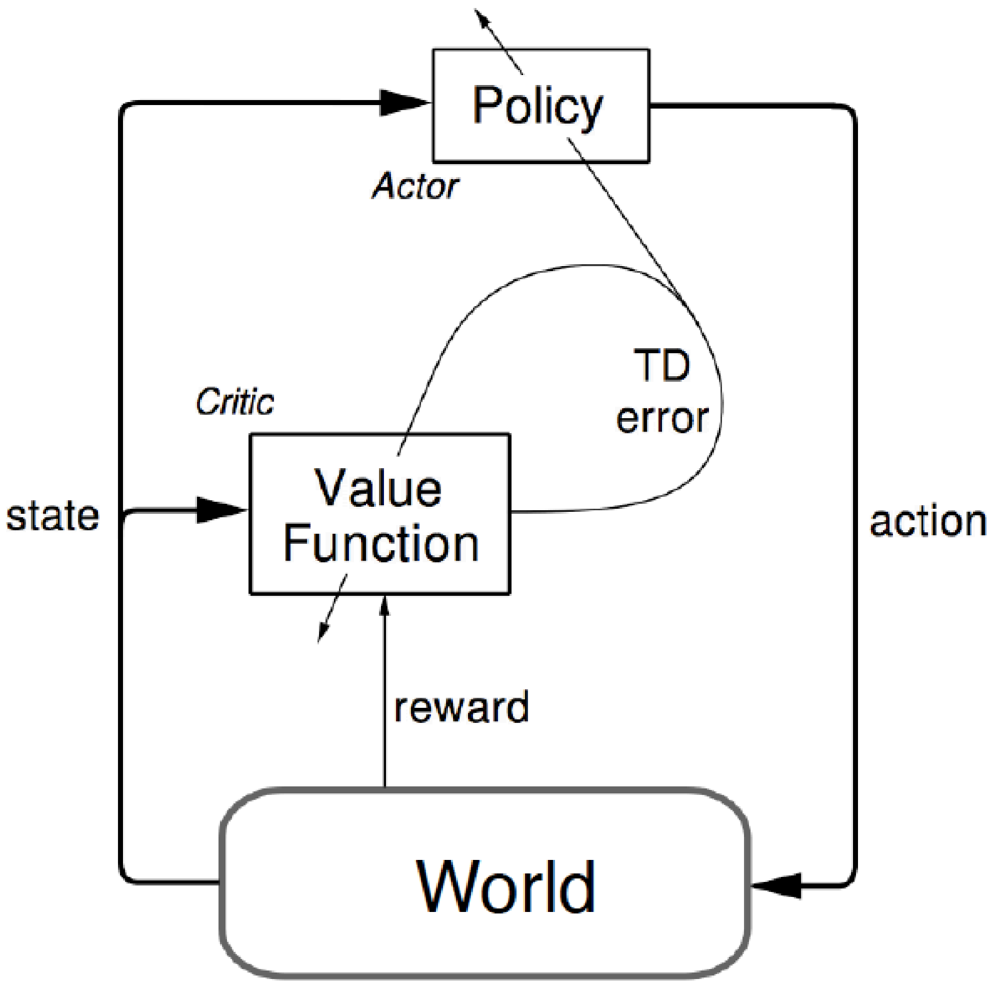
```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha > 0$, $\beta > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$
Repeat forever:
   Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
   For each step of the episode $t = 0, \ldots, T-1$:
      $G_t \leftarrow$ return from step $t$
      $\delta \leftarrow G_t - \hat{v}(S_t,\mathbf{w})$
      $\mathbf{w} \leftarrow \mathbf{w} + \beta\gamma^t\delta\nabla_{\mathbf{w}}\hat{v}(S_t,\mathbf{w})$
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t\delta\nabla_{\boldsymbol{\theta}}\ln\pi(A_t|S_t,\boldsymbol{\theta})$
```

$$\nabla_{\boldsymbol{\theta}} \ln \pi(a|s,\boldsymbol{\theta}) = \frac{\nabla_{\boldsymbol{\theta}}\pi(a|s,\boldsymbol{\theta})}{\pi(a|s,\boldsymbol{\theta})}$$

REINFORCE with Baseline Algorithm

- **Important Note**: *Faster.*

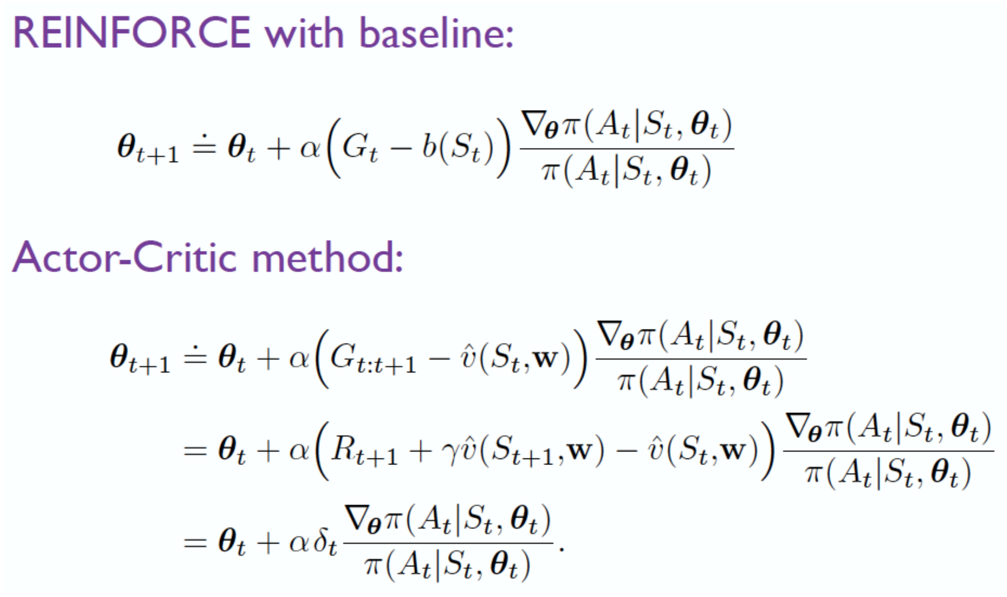
7.21 Actor-Critic Architecture - How does the actor-critic archiecture reduce variance in Monte-Carlo policy gradient algorithms?



Actor-Critic Architecture

- Actor-critic methods consist of two models, which may optionally share parameters.
  1. **Critic**: Updates the action-value function parameters $\boldsymbol{w}$.
  2. **Actor**: Updates policy parameters $\boldsymbol{\theta}$, in the direction suggested by the critic.

7.21.1 Actor-Critic Formulas



REINFORCE with baseline:

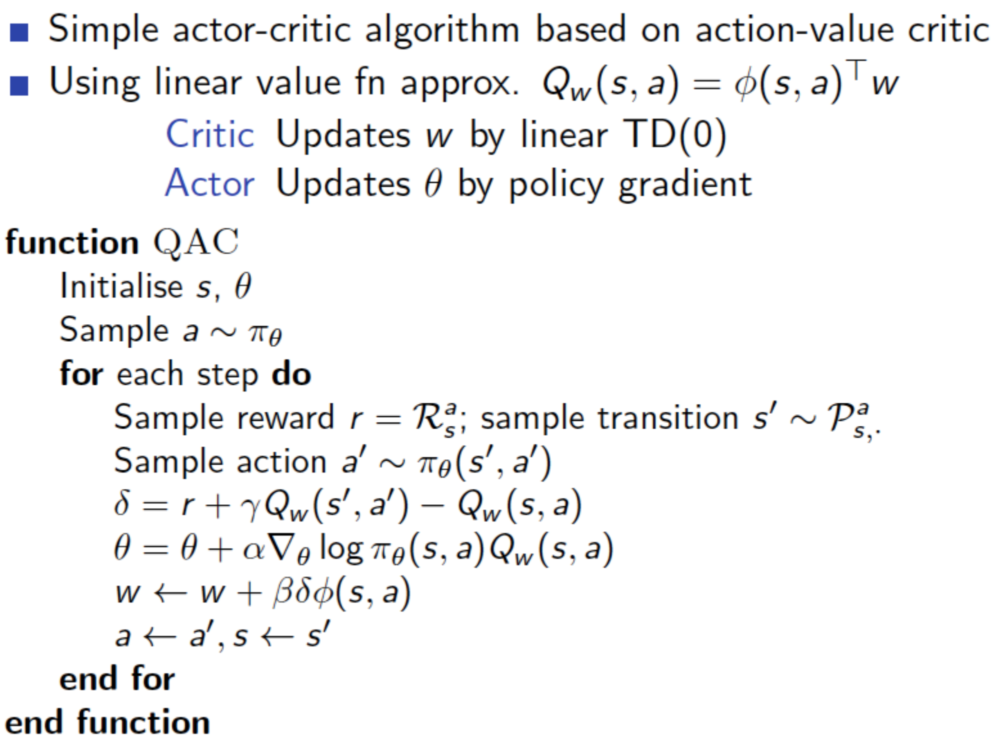
$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big(G_t - b(S_t)\Big)\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}$$

Actor-Critic method:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha\Big(G_{t:t+1} - \hat{v}(S_t,\mathbf{w})\Big)\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}\\ &= \boldsymbol{\theta}_t + \alpha\Big(R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})\Big)\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}\\ &= \boldsymbol{\theta}_t + \alpha\delta_t\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}.\end{aligned}$$

Actor-Critic Formulas

7.22 Actor-Critic Policy Gradient Algorithm - N/A



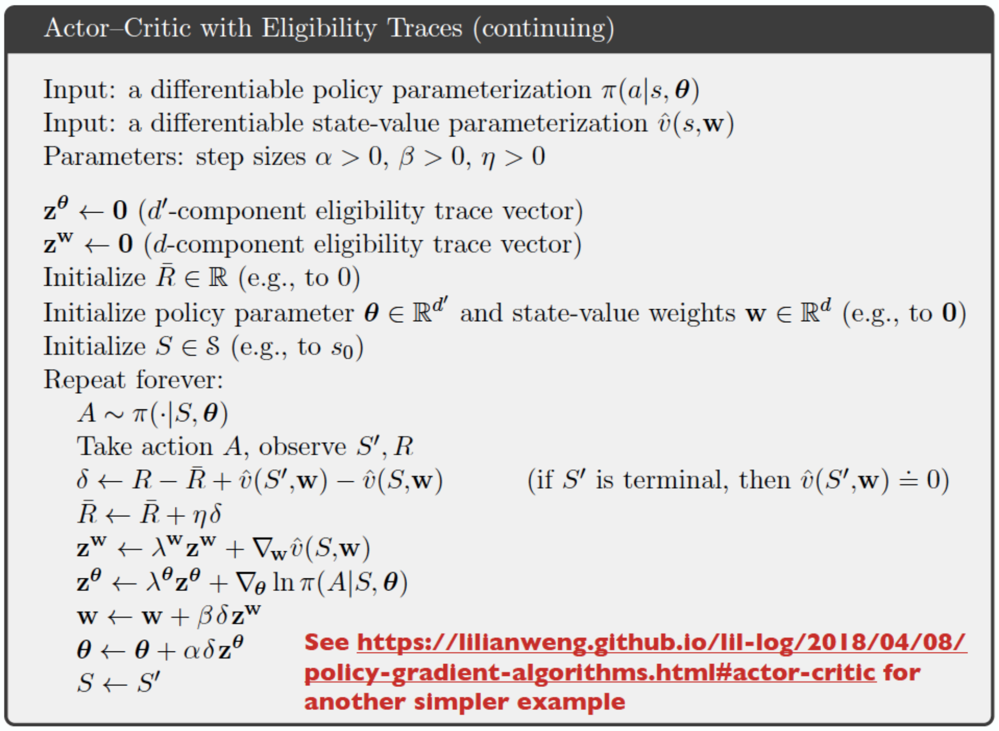
- Simple actor-critic algorithm based on action-value critic
- Using linear value fn approx. $Q_w(s,a) = \phi(s,a)^\top w$

Critic	Updates w by linear TD(0)
Actor	Updates θ by policy gradient

```
function QAC
    Initialise $s$, $\theta$
    Sample $a \sim \pi_\theta$
    for each step do
        Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
        Sample action $a' \sim \pi_\theta(s', a')$
        $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
        $\theta = \theta + \alpha\nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$
        $w \leftarrow w + \beta\delta\phi(s, a)$
        $a \leftarrow a', s \leftarrow s'$
    end for
end function
```

Actor-Critic Policy Gradient Algorithm

7.23 Actor-Critic Policy Gradient with Eligibility Traces Algorithm - N/A



```
Actor–Critic with Eligibility Traces (continuing)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha > 0$, $\beta > 0$, $\eta > 0$

$\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ ($d'$-component eligibility trace vector)
$\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$ ($d$-component eligibility trace vector)
Initialize $\bar{R} \in \mathbb{R}$ (e.g., to 0)
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Initialize $S \in \mathcal{S}$ (e.g., to $s_0$)
Repeat forever:
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action $A$, observe $S', R$
    $\delta \leftarrow R - \bar{R} + \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$        (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\bar{R} \leftarrow \bar{R} + \eta\delta$
    $\mathbf{z}^{\mathbf{w}} \leftarrow \lambda^{\mathbf{w}}\mathbf{z}^{\mathbf{w}} + \nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w})$
    $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \lambda^{\boldsymbol{\theta}}\mathbf{z}^{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} \ln \pi(A|S,\boldsymbol{\theta})$
    $\mathbf{w} \leftarrow \mathbf{w} + \beta\delta\mathbf{z}^{\mathbf{w}}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\mathbf{z}^{\boldsymbol{\theta}}$
    $S \leftarrow S'$
```

See https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#actor-critic for another simpler example

Actor-Critic Policy Gradient with Eligibility Traces Algorithm