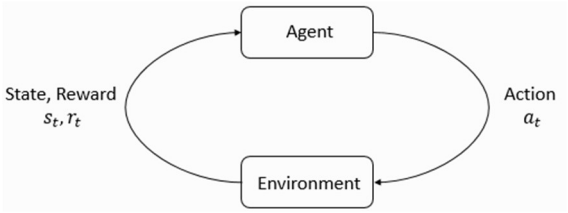
11. Reinforcement Learning (RL)

רוב האלגוריתמים של עולם הלמידה הינם מבוססי דאטה, כלומר, בהינתן מידע מסוים הם מנסים למצוא בו חוקיות מסוימת, ועל בסיסה לבנות מודל שיוכל להתאים למקרים נוספים. אלגוריתמים אלה מחולקים לשניים:

1. אלגוריתמים של למידה מונחית, המבוססים על דאטה $S = \{\boldsymbol{x}, \boldsymbol{y}\}$, כאשר $\boldsymbol{x} \in \mathbb{R}^{n \times d}$ הינו אוסף של אובייקטים (למשל נקודות במרחב, אוסף של תמונות וכדו'), ו-$\boldsymbol{y} \in \mathbb{R}^n$ הינו אוסף של labels. לכל אובייקט $x \in \mathbb{R}^d$ יש label מתאים $y \in \mathbb{R}^1$.
2. אלגוריתמים של למידה לא מונחית עבורם הדאטה $\boldsymbol{x} \in \mathbb{R}^{n \times d}$ הוא אוסף של אובייקטים ללא labels, ומנסים למצוא כללים מסוימים על דאטה זה (למשל – חלוקה לאשכולות, הורדת ממד ועוד).

למידה מבוססת חיזוקים הינה פרדיגמה נוספת תחת התחום של למידת מכונה, כאשר במקרה זה הלמידה לא מסתמכת על דאטה קיים, אלא על חקירה של הסביבה ומציאת המדיניות/האסטרטגיה הטובה ביותר לפעולה. ישנו סוכן שנמצא בסביבה שאינה מוכרת, ועליו לבצע צעדים כך שהתגמול המצטבר אותו הוא יקבל יהיה מקסימלי. בלמידה מבוססת חיזוקים, בניגוד לפרדיגמות האחרות של למידת מכונה, הסביבה לא ידועה מבעוד מועד. הסוכן נמצא באי ודאות ואינו יודע בשום שלב מה הצעד הנכון לעשות, אלא הוא רק מקבל פידבק על הצעדים שלו, וכך הוא לומד מה כדאי לעשות וממה כדאי להימנע. באופן כללי ניתן לומר שמטרת הלמידה היא לייצר אסטרטגיה כך שבכל מיני מצבים לא ידועים הסוכן יבחר בפעולות שבאופן מצטבר יהיו הכי יעילות עבורו. נתאר את תהליך הלמידה באופן גרפי:



איור 11.1 מודל של סוכן וסביבה.

בכל צעד הסוכן נמצא במצב s_t ובוחר פעולה a_t המעבירה אותו למצב s_{t+1}, ובהתאם לכך הוא מקבל מהסביבה תגמול r_t. האופן בה מתבצעת הלמידה היא בעזרת התגמול, כאשר נרצה שהסוכן יבצע פעולות המזכות אותו בתגמול חיובי (-חיזוק) וימנע מפעולות עבורן הוא מקבל תגמול שלילי, ובמצטבר הוא ימקסם את כלל התגמולים עבור כל הצעדים שהוא בחר לעשות. כדי להבין כיצד האלגוריתמים של למידה מבוססת חיזוקים עובדים ראשית יש להגדיר את המושגים השונים, ובנוסף יש לנסח באופן פורמלי את התיאור המתמטי של חלקי הבעיה השונים.

11.1 Introduction to RL

בפרק זה נגדיר באופן פורמלי תהליכי מרקוב, בעזרתם ניתן לתאר בעיות של למידה מבוססת חיזוקים, ונראה כיצד ניתן למצוא אופטימום לבעיות אלו בהינתן מודל וכל הפרמטרים שלו. לאחר מכן נדון בקצרה במספר שיטות המנסות למצוא אסטרטגיה אופטימלית עבור תהליך מרקוב כאשר לא כל הפרמטרים של המודל נתונים, ובפרקים הבאים נדבר על שיטות אלה בהרחבה. שיטות אלה הן למעשה הלב של למידה מבוססת חיזוקים, כיוון שהן מנסות למצוא אסטרטגיה אופטימלית על בסיס תגמולים ללא ידיעת הפרמטרים של המודל המרקובי עבורו רוצים למצוא אופטימום.

11.1.1 Markov Decision Process (MDP) and RL

המודל המתמטי העיקרי עליו בנויים האלגוריתמים השונים של RL הינו תהליך החלטה מרקובי, כלומר תהליך שבה המעברים בין המצבים מקיים את תכונת מרקוב, לפיה ההתפלגות של מצב מסוים תלויה רק במצב הקודם לו:

$$P(s_{t+1} = j | s_1, \dots, s_t) = P(s_{t+1} = j | s_t)$$

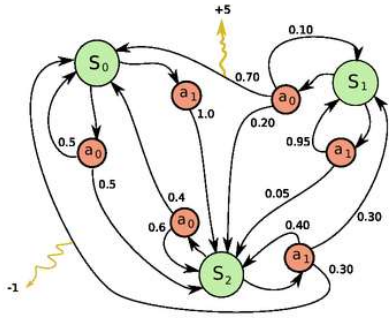
תהליך קבלת החלטות מרקובי מתואר על ידי סט הפרמטרים $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}\}$:

- State space ($\mathcal{S}$) – מרחב המצבים של המערכת. המצב ההתחלתי מסומן ב-S_0.
- Action space ($\mathcal{A}$) – מרחב הפעולות. A_s הוא מרחב הפעולות האפשריות במצב S.
- Transition ($\mathcal{T}$) – הביטוי: $T(s'|s,a) \to [0,1]$ הינו פונקציית מעבר, המחשבת את ההסתברות לעבור בזמן t ממצב s_t למצב s_{t+1} על ידי הפעולה a: $T(s'|s,a) = \mathcal{P}(s_{t+1} = s' | s_t = s, a_t = a)$. ביטוי זה למעשה מייצג את המודל – מה ההסתברות שבחירת הפעולה a במצב s תביא את הסוכן למצב s'.
- Reward ($\mathcal{R}$) – הביטוי: $\mathcal{R}_a(s,s') \to \mathbb{R}$ הינו פונקציה הנותנת תגמול/רווח לכל פעולה a הגורמת למעבר ממצב s למצב s', כאשר בדרך כלל $\mathcal{R}_a \in [0,1]$. לעיתים מסמנים את התגמול של הצעד בזמן t ב-r_t.

המרקוביות של התהליך באה לידי ביטוי בכך שמצב s_t מכיל בתוכו את כל המידע הנחוץ בכדי לקבל החלטה לגבי a_t, או במילים אחרות – כל ההיסטוריה בעצם שמורה בתוך המצב s_t.

ריצה של MDP מאופיינת על ידי הרביעייה הסדורה $\{s_t, a_t, r_t, s_{t+1}\}$ – פעולה a_t המתרחשת בזמן t וגורמת למעבר ממצב s_t למצב s_{t+1}, ובנוסף מקבלת תגמול מיידי r_t, כאשר $r_t \sim \mathcal{R}(s_t, a_t)$ ו-$s_{t+1} \sim p(\cdot\,|s_t, a_t)$.

מסלול (trajectory) הינו סט של שלשות $\tau = \{s_0, a_0, r_0, \dots, s_t, a_t, r_t\}$, כאשר המצב התחלתי מוגרל מהתפלגות כלשהיא $s_0 \sim \rho_0(\cdot)$, והמעבר בין המצבים יכול להיות דטרמיניסטי $s_{t+1} = f(s_t, a_t)$ או סטוכסטי $s_{t+1} \sim p(\cdot\,|s_t, a_t)$.



איור 11.2 תהליך קבלת החלטות מרקובי. ישנם שלושה מצבים – $\{s_0, s_1, s_2\}$, ובכל אחד מהם יש שתי פעולות אפשריות (עם הסתברויות מעבר שונות) – $\{a_0, a_1\}$. עבור חלק מהפעולות יש תגמול שונה מ-0. מסלול יהיה מעבר על אוסף של מצבים דרך אוסף של פעולות, שלכל אחד מהן יש תגמול.

אסטרטגיה של סוכן, המסומנת ב-π, הינה בחירה של אוסף מהלכים. בבעיות של למידה מבוססת חיזוקים, נרצה למצוא **אסטרטגיה אופטימלית** (Optimal Policy) $\pi: S \to A$ הממקסמת את **התגמול המצטבר** $\sum_{t=0}^{\infty} \mathcal{R}\big(s_t, \pi(s_t)\big)$. כיוון שלא תמיד אפשרי לחשב באופן ישיר את האסטרטגיה האופטימלית, ניתן להגדיר ערך החזרה (Return) המבטא סכום של תגמולים, ומנסים למקסם את התוחלת שלו $\mathbb{E}[Return|\mathcal{S}, \mathcal{A}]$. ערך ההחזרה הכי נפוץ נקרא discount return, והוא מוגדר באופן הבא: עבור פרמטר $\gamma \in (0,1)$, ה-Return הינו הסכום הבא:

$$Return = \sum_{t=1}^{T} \gamma^{t-1} r_t$$

אם $\gamma = 0$, אז מתעניינים רק בתגמול המיידי, וככל ש-γ גדל כך נותנים יותר משמעות לתגמולים עתידיים. כיוון ש-$r_t \in [0, 1]$, הסכום חסום על ידי $\frac{1}{1-\gamma}$.

התוחלת של ערך ההחזרה נקראת Value function, והיא נותנת לכל מצב ערך מסוים המשקף את תוחלת התגמול שניתן להשיג דרך מצב זה. באופן פורמלי, כאשר מתחילים ממצב s, ה-Value function מוגדר להיות:

$$\mathcal{V}^{\pi}(s) = \mathbb{E}[R(\tau)|s_0 = s]$$

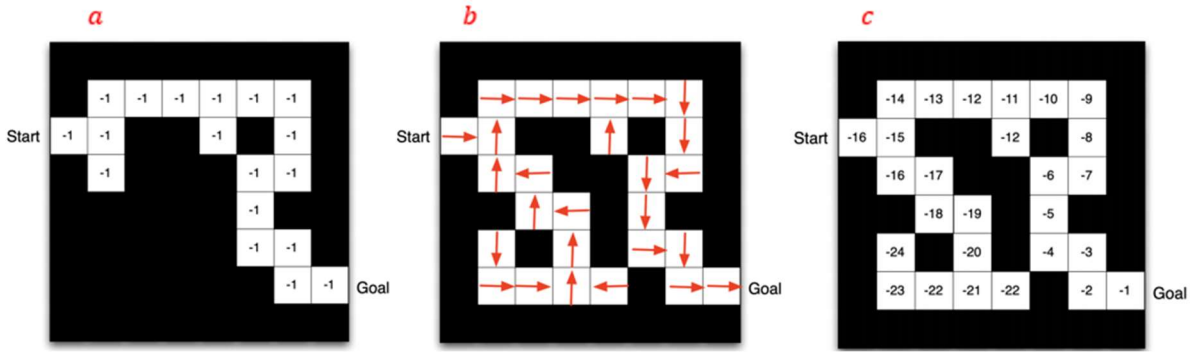
בעזרת ביטוי זה ניתן לחשב את **האסטרטגיה האופטימלית**, כאשר ניתן לנקוט בגישה ישירה ובגישה עקיפה. הגישה הישירה מנסה למצוא בכל מצב מה הפעולה הכי כדאית. בהתאם לכך, חישוב האסטרטגיה האופטימלית יעשה באופן הבא:

$$\pi(s) = \arg\max_{a} \sum_{s'} p_a(s, s')\big(\mathcal{R}_a(s, s') + \gamma \mathcal{V}(s')\big)$$

לעיתים החישוב הישיר מסובך, כיוון שהוא צריך לקחת בחשבון את כל הפעולות האפשריות, ולכן מסתכלים רק על ה-Value function. לאחר שלכל מצב יש ערך מסוים, בכל מצב הסוכן יעבור למצב בעל הערך הכי גדול מבין כל המצבים האפשריים אליהם ניתן לעבור. חישוב הערך של כל מצב נעשה באופן הבא:

$$\mathcal{V}(s) = \sum_{s'} p_{\pi}(s, s')\big(\mathcal{R}_{\pi}(s, s') + \gamma V(s')\big)$$

ניתן לשים לב שבעוד הגישה הראשונה מתמקדת במציאת **אסטרטגיה/מדיניות אופטימלית** על בסיס הפעולות האפשריות בכל מצב, הגישה השנייה לא מסתכלת על הפעולות אלא על הערך של כל מצב, המשקף את **תוחלת התגמול** שניתן להשיג כאשר נמצאים במצב זה.



איור 11.3 a) מודל: המצב של הסוכן הוא המשבצת בו הוא נמצא, הפעולות האפשריות הן ארבעת הכיוונים, כל פעולה גוררת תגמול של -1, והסתברויות המעבר נקבעות לפי הצבעים של המשבצות (אי אפשר ללכת למשבצות שחורות). b) מדיניות – החלטה בכל מצב איזה צעד לבצע. c) Value של כל משבצת.

לסיכום, ניתן לומר שכל התחום של RL מבוסס על שלוש אבני יסוד:

- מודל: האופן בו אנו מתארים את מרחב המצבים והפעולות. המודל יכול להיות נתון או שנצטרך לשערך אותו, והוא מורכב מהסתברויות מעבר בין מצבים ותגמול עבור כל צעד:

$$\mathcal{P}_{ss'}^{a} = p_{\pi}(s,s') = P(s_{t+1} = s' | s_t = s, a_t = a)$$

$$\mathcal{R}_{ss'}^{a} = \mathcal{R}_{\pi}(s,s') = \mathbb{E}\left[r_{t+1} \middle| s_t = s, a_t = a, s_{t+1} = s'\right]$$

- Value function – פונקציה המתארת את התוחלת של התגמולים העתידיים:

$$\mathcal{V}^{\pi}(s) = \mathbb{E}[R(\tau)|s_0 = s]$$

- מדיניות/אסטרטגיה (Policy) – בחירה (דטרמיניסטית או אקראית) של צעד בכל מצב נתון:

$$\pi(s|a)$$

11.1.2 Bellman Equation

לאחר שהגדרנו את המטרה של למידה מבוססת חיזוקים, ניתן לדבר על שיטות לחישוב אסטרטגיה אופטימלית. בפרק זה נתייחס למקרה הספציפי בו נתון מודל מרקובי עם כל הפרמטרים שלו, כלומר אוסף המצבים, הפעולות והסתברויות המעבר ידועים. כאמור, Value function הינה התוחלת של ערך ההחזרה עבור אסטרטגיה נתונה π, כאשר מתחילים ממצב s:

$$\mathcal{V}^{\pi}(s) = \mathbb{E}[R(\tau)|s_0 = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right]$$

ביטוי זה מסתכל על הערך של כל מצב, בלי להתייחס לפעולות המעבירות את הסוכן ממצב אחד למצב אחר. נתינת ערך לכל מצב יכולה לסייע במציאת אסטרטגיה אופטימלית, כיוון שהיא מדרגת את המצבים השונים של המודל. באופן דומה, ניתן להגדיר את ה-Action-Value function – התוחלת של ערך ההחזרה עבור אסטרטגיה נתונה π, כאשר במצב s מבצעים את פעולה a, ולאחר מכן ממשיכים לפי האסטרטגיה π:

$$Q^{\pi}(s,a) = \mathbb{E}[R(\tau)|s_0 = s, a_0 = a] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = t\right]$$

ביטוי זה מסתכל על הזוג (s_t, a_t), כלומר בכל מצב יש התייחסות למצב הנוכחי ולפעולות האפשריות במצב זה. בדומה ל-Value function, גם ביטוי זה יכול לסייע במציאת אסטרטגיה אופטימלית, כיוון שהוא מדרג עבור כל מצב את הפעולות האפשריות.

נוכל לסמן ב-$\mathcal{V}^*(s)$ ו-$Q^*(s,a)$ את הערכים של האסטרטגיה האופטימלית π^* – Optimal Value function ו-Optimal Action-Value function. עבור אסטרטגיה זו מתקיים:

$$\mathcal{V}^*(s) = \max_{\pi} \mathbb{E}[R(\tau)|s_0 = s], \mathcal{Q}^*(s,a) = \max_{\pi} \mathbb{E}[R(\tau)|s_0 = s, a_0 = a]$$

הרבה פעמים מתעניינים ביחס שבין $\mathcal{V}$ ו-$\mathcal{Q}$, וניתן להיעזר במעברים הבאים:

$$\mathcal{V}^{\pi}(s) = \mathbb{E}[\mathcal{Q}^{\pi}(s,a)]$$

$$\mathcal{V}^*(s) = \max_{\pi} \mathcal{Q}^*(s,a)$$

באופן קומפקטי ניתן לרשום את $\mathcal{V}^*(s)$ כך:

$$\forall s \in S \quad \mathcal{V}^*(s) = \max_{\pi} \mathcal{V}^{\pi}(s)$$

כלומר, האסטרטגיה π^* הינה האופטימלית עבור כל מצב s.

כעת נתון מודל מרקובי עם כל הפרמטרים שלו – אוסף המצבים והפעולות, הסתברויות המעבר והתגמול עבור כל פעולה, ומעוניינים למצוא דרך פעולה אופטימלית עבור מודל זה. ניתן לעשות זאת בשתי דרכים עיקריות – מציאת האסטרטגיה $\pi(a|s)$ האופטימלית, או חישוב ה-Value של כל מצב ובחירת מצבים בהתאם לערך זה. משימות אלו יכולות להיות מסובכות מאוד עבור משימות מורכבות וגדולות, ולכן לעיתים קרובות משתמשים בשיטות איטרטיביות ובקירובים על מנת לדעת כיצד לנהוג בכל מצב. הדרך הפשוטה לחישוב $\mathcal{V}^{\pi}(s)$ משתמשת ב-**Bellman equation**, המבוססת על תכנות דינמי. נפתח את הביטוי של $\mathcal{V}^{\pi}(s)$ מתוך ההגדרה שלו:

$$\mathcal{V}^{\pi}(s) = \mathbb{E}[R(\tau)|s_0 = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right]$$

נפצל את הסכום שבתוחלת לשני איברים – האיבר הראשון ויתר האיברים:

$$= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma \cdot \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right]$$

כעת נשתמש בהגדרת התוחלת ונקבל:

$$= \sum_{a,s'} \pi(a|s) p_{\pi}(s,s') \left(\mathcal{R}_{\pi}(s,s') + \gamma \cdot \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right]\right)$$

$$= \sum_{a,s'} \pi(a|s) p_{\pi}(s,s') \left(\mathcal{R}_{\pi}(s,s') + \gamma \cdot \mathcal{V}^{\pi}(s')\right)$$

הביטוי המתקבל הוא מערכת משוואות ליניאריות הניתנות לפתרון באופן אנליטי, אם כי סיבוכיות החישוב יקרה. נסמן:

$$V = [V_1, \dots, V_n]^T, R = [r_1, \dots, r_n]^T$$

$$T = \begin{pmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{pmatrix}$$

ונקבל משוואה מטריציונית:

$$V = R + \gamma T V \rightarrow V = R + \gamma T V$$

$$\rightarrow \mathcal{V}^{\pi}(s) = (\mathbb{I}_n - \gamma T)^{-1} R$$

בגלל שהערכים העצמיים של T חסומים על ידי 1, בהכרח יהיה ניתן להפוך את $\mathbb{I}_n - \gamma T$ מה שמבטיח שיהיה פתרון למשוואה, ופתרון זה הוא אף יחיד עבור $\mathcal{V}^{\pi}$. כשמוצאים את V ניתן למצוא גם את $\mathcal{Q}^{\pi}$ על ידי הקשר:

$$\mathcal{Q}^{\pi}(s,a) = \sum_{s'} p_{\pi}(s,s')\left(\mathcal{R}_{\pi}(s,s') + \gamma \mathcal{V}^{\pi}(s')\right) = \sum_{s'} p_{\pi}(s,s')\left(\mathcal{R}_{\pi}(s,s') + \gamma \sum_{a'} \pi(a'|s') \mathcal{Q}^{\pi}(a'|s')\right)$$

Iterative Policy Evaluation

הסיבוכיות של היפוך מטריצה הינו $\mathcal{O}(n^3)$, ועבור n גדול החישוב נהיה מאוד יקר ולא יעיל. כדי לחשב את הפתרון באופן יעיל, ניתן כאמור להשתמש בשיטות איטרטיביות. שיטות אלו מבוססות על אופרטור בלמן, המוגדר באופן הבא:

$$BO(V) = R^\pi + \gamma T^\pi \cdot V$$

ניתן להוכיח שאופרטור זה הינו העתקה מכווצת (contractive mapping), כלומר הוא מקיים את התנאי:

$$\forall x, y: \|f(x) - f(y)\| < \gamma \|x - y\| \; for \; 0 < \gamma < 1$$

במילים: עבור שני וקטורים במרחב, אופרטור $f(\cdot)$ ומספר γ החסום בין 0 ל-1, אם נפעיל את האופרטור על כל אחד מהווקטורים ונחשב את נורמת ההפרש, נקבל מספר **קטן** יותר מאשר הנורמה בין הווקטורים כפול הפקטור γ. אופרטור המקיים תכונה זו הינו העתקה מכווצת, כיוון שנורמת ההפרש של האופרטור על שני וקטורים קטנה מנורמת ההפרש בין הווקטורים עצמם. הוכחה:

$$\|f(u) - f(v)\|_\infty = \|R^\pi + \gamma T^\pi \cdot v - (R^\pi + \gamma T^\pi \cdot u)\|_\infty = \|\gamma T^\pi (v - u)\|_\infty$$

מטריקת אינסוף מוגדרת לפי: $\|u - v\|_\infty = \max_{s \in \mathcal{S}} |u(s) - v(s)|$. לכן נוכל לרשום:

$$\|\gamma T^\pi (v - u)\|_\infty \leq \|\gamma T^\pi \mathbb{1} \|u - v\|_\infty\|_\infty$$

הביטוי $T^\pi \mathbb{1}$ למעשה סוכם את כל ערכי מטריצת המעברים, לכן הוא מסתכם ל-1, ונקבל:

$$= \gamma \|u - v\|_\infty$$

ובכך הוכחנו את הדרוש.

לפי משפט נקודת השבת של בנך, להעתקה מכווצת יש נקודת שבת (fixed point) יחידה המקיימת $x = f(x)$ וסדרה $x_{t+1} = f(x_t)$ המתכנסת לאותה נקודת שבת. לכן נוכל להשתמש באלגוריתם איטרטיבי עבור $\mathcal{V}^\pi$ שיביא אותנו לנקודת שבת, ולפי המשפט זוהי נקודת השבת היחידה וממילא הגענו להתכנסות. בפועל, נשתמש באלגוריתם האיטרטיבי הבא:

$$V_{k+1} = BO(V_k) = R^\pi + \gamma T^\pi \cdot V_k$$

נסתכל על הדוגמא הבאה:

$$T^\pi = \begin{pmatrix} 0.8 & 0.1 & 0.1 & 0 & 0 \\ 0.1 & 0.8 & 0.1 & 0 & 0 \\ 0 & 0.1 & 0.8 & 0.1 & 0 \\ 0 & 0 & 0.1 & 0.8 & 0.1 \\ 0 & 0 & 0.1 & 0.1 & 0.8 \end{pmatrix}, \mathcal{R}^\pi = \begin{pmatrix} 0.1 \\ 1.3 \\ 3.4 \\ 1.9 \\ 0.4 \end{pmatrix}, \gamma = 0.9$$

באמצעות השיטה האיטרטיבית נקבל:

$$V_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, V_1 = \begin{pmatrix} 0.1 \\ 1.3 \\ 3.4 \\ 1.9 \\ 0.4 \end{pmatrix}, V_2 = \begin{pmatrix} 0.6 \\ 2.6 \\ 6.1 \\ 3.7 \\ 1.2 \end{pmatrix}, \dots, V_{10} = \begin{pmatrix} 7.6 \\ 10.8 \\ 18.2 \\ 16.0 \\ 9.8 \end{pmatrix}, \dots, V_{50} = \begin{pmatrix} 14.5 \\ 17.1 \\ 26.4 \\ 26.8 \\ 18.4 \end{pmatrix}, V^\pi = \begin{pmatrix} 14.7 \\ 17.9 \\ 26.6 \\ 27.1 \\ 18.7 \end{pmatrix}$$

ניתן לשים לב שאחרי 50 איטרציות הפתרון המתקבל בצורה האיטרטיבית קרוב מאוד לפתרון המתקבל בצורה האנליטית.

Policy Iteration (PI)

חישוב ה-Value function מאפשר לנו לחשב את ערכו של $\mathcal{V}^\pi(s)$ עבור כל s, אך הוא אינו מבטיח שנגיע לאסטרטגיה האופטימלית. נניח והצלחנו לחשב את $\mathcal{V}^\pi(s)$ וממנו אנו יודעים לגזור אסטרטגיה, עדיין יתכן שקיימת פעולה a שיותר משתלמת מאשר הפעולה המוצעת לפי האסטרטגיה הנגזרת מ-$\mathcal{V}^\pi(s)$. באופן פורמלי ניתן לתאר זאת בצורה פשוטה – נניח שחישבנו את $\mathcal{V}^\pi(s)$ ואת $Q^\pi(s, a)$ יתכן וקיימת פעולה עבורה:

$$\text{for such } s,a: \quad Q^{\pi}(s,a) > \mathcal{V}^{\pi}(s)$$

אם קיימת פעולה כזו, אז ישתלם לבחור בה ולאחר מכן לחזור לפעול בהתאם לאסטרטגיה $\pi(a|s)$ הנגזרת מחישוב ה-Value function. למעשה, ניתן לחפש את **כל** הפעולות עבורן כדאי לבצע פעולה מסיימת עבורה התגמול יהיה גבוה יותר מאשר האסטרטגיה של $\mathcal{V}^{\pi}(s)$. באופן פורמלי יותר, נרצה להגדיר אסטרטגיה דטרמיניסטית, עבורה בהסתברות 1 ננקוט בכל מצב s בפעולה הכי כדאית a:

$$\pi'(a|s) = 1 \text{ for } a = \arg\max_{a'} Q^{\pi}(s,a')$$

נשים לב שרעיון זה הוא בעצם להשתמש באסטרטגיה גרידית – בכל מצב לנקוט בפעולה הכי משתלמת בטווח של צעד יחיד, ואז להמשיך עם האסטרטגיה הנתונה. השאלה העולה היא כמובן – מדוע זה בהכרח נכון? כלומר, האם הרעיון שאומר שלא משנה באיזה מצב אנו נמצאים, הבחירה של הפעולה האופטימלית בהכרח תוביל לקבלת אסטרטגיה יותר טובה מאשר האסטרטגיה הנוכחית? בכדי להוכיח זאת ננסח זאת כמשפט:

בהינתן 2 אסטרטגיות π, π', כאשר π' דטרמיניסטית, אז כאשר $Q^{\pi}\big(s,\pi'(s)\big) > \mathcal{V}^{\pi}(s)$ בהכרח לכל s יתקיים: $\mathcal{V}^{\pi'}(s) > \mathcal{V}^{\pi}(s)$. ראשית נפתח לפי הגדרה:

$$\mathcal{V}^{\pi}(s) < Q^{\pi}\big(s,\pi'(s)\big) = \mathbb{E}_{\pi}\big[r_{t+1} + \gamma \cdot \mathcal{V}^{\pi}(s_{t+1}) | s_t = s, a_t = \pi'(s)\big]$$

כיוון שהאסטרטגיה הינה דטרמיניסטית, הפעולה הנבחרת אינה רנדומלית ביחס ל-π', ולכן נוכל לרשום:

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma \cdot \mathcal{V}^{\pi}(s_{t+1}) | s_t = s]$$

כעת לפי אותו אי שוויון שבהנחה נוכל לבצע את אותו חישוב גם לצעד הבא s_{t+2}:

$$< \mathbb{E}_{\pi'}\left[r_{t+1} + \gamma \cdot Q^{\pi}\left(s_{t+1}, \pi'(s_{t+1})\right) | s_t = s\right]$$

וזה שוב שווה ל:

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot \mathcal{V}^{\pi}(s_{t+2}) | s_t = s]$$

וכך הלאה, ולמעשה הוכחנו את הדרוש – נקיטת הפעולה הכי יעילה בכל מצב תמיד תהיה יותר טובה מהפתרון של $\mathcal{V}^{\pi}(s)$.

כעת יש בידינו שתי טכניקות שאנו יודעים לבצע:

Evaluation (E) – בהינתן אסטרטגיה מסוימת נוכל לפתור את משוואות בלמן ולקבל את $\mathcal{V}^{\pi}(s)$ ו-$Q^{\pi}(s,a)$.

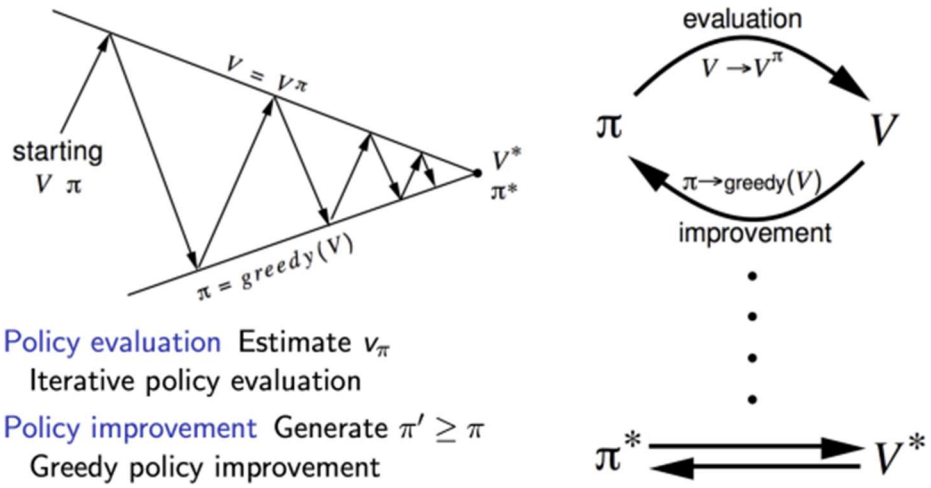
Improve (I) – בהינתן הערך של Value function, נוכל לשפר אותה באמצעות בחירה גרידית של פעולה.

בעזרת טכניקות אלו ניתן להתחיל מאסטרטגיה רנדומלית, ואז לבצע איטרציות המורכבות משתי הטכניקות האלה באופן הבא:

$$\pi_0 \xrightarrow{E} \mathcal{V}^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} \mathcal{V}^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots$$

תהליך זה נקרא Policy iteration – בכל צעד בו יש לנו אסטרטגיה נפתור עבורה משוואת בלמן ובכך נחשב את ה-Value function שלה, ולאחר מכן נשפר את האסטרטגיה באמצעות policy improvement, שכאמור מבצע בחירה גרידית שבטווח הקצר טובה יותר מאשר ה-Value function שחישבנו. ניתן להוכיח שאחרי מספר סופי של איטרציות האסטרטגיה תתכנס לנקודת שבת (fixed point), ואז הפעולה הבאה לפי האסטרטגיה תהיה זהה לבחירה הגרידית:

$$\pi(s) = \arg\max_{a} Q^{\pi}(s,a) = \pi'(s)$$



איור 11.4 Policy iteration – ביצוע איטרציות של Policy evaluation ו-Policy improvement על מנת למצוא בכל שלב את ה- value function ולשפר אותו באמצעות בחירה גרידית.

Bellman optimality equations

השלב הבא בשימוש ב-Policy iteration הוא **להוכיח** שהאסטרטגיה אליה מתכנסים הינה אופטימלית. נסמן את נקודת השבת ב-π^* ונקבל את הקשר הבא:

$$\mathcal{V}^{\pi^*}(s) \equiv \mathcal{V}^*(s) = \max_a \mathcal{Q}^*(s,a) = \max_a \sum_{s'} p_\pi(s,s')\Big(\mathcal{R}_\pi(s,s') + \gamma \cdot \mathcal{V}^*(s')\Big)$$

ובאופן דומה:

$$\mathcal{Q}^*(s,a) = \sum_{s'} p_\pi(s,s')\left(\mathcal{R}_\pi(s,s') + \gamma \cdot \max_{a'} \mathcal{Q}^*(s',a')\right)$$

משוואות אלה נקראות Bellman optimality equation. ניתן לשים לב שהן מאוד דומות למשוואות בלמן מהן יצאנו, אך במקום התוחלת שהייתה לנו בהתחלה, כעת יש max. נרצה להראות שהפתרון של משוואות אלה הוא ה-Value של **האסטרטגיה האופטימלית**. ננסח את הטענה באופן הבא:

אסטרטגיה הינה אופטימלית אם ורק אם היא מקיימת את Bellman optimality equation. כיוון אחד להוכחה הוא טריוויאלי – אם האסטרטגיה הינה אופטימלית אז היא בהכרח מקיימת את משוואות האופטימליות, כיוון שהראינו שהן מתקבלות מנקודת השבת אליה האיטרציות מתכנסות. אם האסטרטגיה לא הייתה אופטימלית אז היה ניתן לשפר עוד את האסטרטגיה ולא היינו מגיעים עדיין לנקודת השבת. בשביל להוכיח את הכיוון השני נשתמש שוב ברעיון של העתקה מכווצת. נגדיר את האופרטור הבא:

$$BV(s) = \max_a \sum_{s'} p_\pi(s,s')\Big(\mathcal{R}_\pi(s,s') + \gamma \cdot \mathcal{V}(s)\Big)$$

ניתן להראות שאופרטור זה הינו העתקה מכווצת, וממילא לפי המשפט של בנך יש לו נקודת שבת יחידה. כיוון שהראינו ששימוש ב-Policy iteration מביא את האסטרטגיה לנקודת שבת מסוימת, נוכל לצרף לכך את העובדה שהאופרטור שהגדרנו הינו העתקה מכווצת וממילא נקבל שאותה נקודת שבת הינה יחידה, וממילא אופטימלית.

Value Iteration

הראנו שבעזרת שיטת Policy iteration ניתן להגיע לאסטרטגיה אופטימלית, אך התהליך יכול להיות איטי. ניתן לנקוט גם בגישה יותר ישירה ולנסות לחשב באופן ישיר את הפתרון של משוואות האופטימליות של בלמן (ופתרונן הינו אופטימלי כיוון שהראינו שהפתרון הוא נקודת שבת יחידה). נתחיל עם פתרון רנדומלי $\mathcal{V}_0$ ולאחר מכן נצבע איטרציות באופן הבא עד שנגיע להתכנסות:

$$\mathcal{V}_{k+1} = \max_a \sum_{s'} p_\pi(s,s')\big(\mathcal{R}_\pi(s,s') + \gamma \cdot \mathcal{V}_k(s')\big)$$

נשים לב שבשיטה זו אין לנו מידע לגבי האסטרטגיה אלא רק חישבנו את ה-Value function, אך ממנה ניתן לגזור את Q ואז לבחור באסטרטגיה גרידית, שהינה במקרה זה גם אופטימלית:

$$\pi(s) = \arg\max_a Q^\pi(s,a)$$

ניתן להראות כי בשיטה זו ההתכנסות מהירה יותר ודרושות פחות איטרציות מהשיטה הקודמת, אך כל איטרציה יותר מורכבת.

Limitations

לשתי השיטות – Policy iteration ו-Value iteration – יש שני חסרונות מרכזיים:

1. הן דורשות לדעת את המודל והסביבה באופן שלם ומדויק.

2. הן דורשות לעדכן בכל שלב את כל המצבים בו זמנית. עבור מערכות עם הרבה מצבים, זה לא מעשי.

11.1.3 Learning Algorithms

בפרק הקודם הוסבר כיצד ניתן לחשב את האסטרטגיה האופטימלית וערך ההחזרה **בהינתן** מודל מרקובי. השתמשנו בשתי הנחות עיקריות על מנת להתמודד עם הבעיה:

1. Tabular MDP – הנחנו שהבעיה סופית ולא גדולה מדי, כך שנוכל לייצג אותה בזיכרון ולפתור אותה.

2. Known environment – הנחנו שהמודל ידוע לנו, כלומר נתונה לנו מטריצת המעברים שקובעת מה הסיכוי לעבור ממצב s למצב s' כשנוקטים בפעולה a (סימנו את זה בתור $\mathcal{P}^a_{ss'} = p_\pi(s,s')$), ובנוסף נתון לנו מה ה-reward המתקבל עבור כל action (סימנו את זה בתור $\mathcal{R}^a_{ss'} = \mathcal{R}_\pi(s,s')$).

בעזרת שתי ההנחות פיתחנו את משוואות בלמן, כאשר היו לנו שני צמדים של משוואות. משוואות בלמן עבור אסטרטגיה נתונה נכתבות באופן הבא:

$$\mathcal{V}^\pi(s) = \sum_{a,s'} \pi(a|s)\mathcal{P}^a_{ss'}\left(\mathcal{R}^a_{ss'} + \gamma \cdot \mathcal{V}^\pi(s')\right)$$

$$Q^\pi(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}\left(\mathcal{R}^a_{ss'} + \gamma \sum_{a'} Q^\pi(s',a')\right)$$

ובנוסף פיתחנו את המשוואות עבור הפתרון האופטימלי:

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'}\big(\mathcal{R}^a_{ss'} + \gamma \cdot \mathcal{V}^*(s')\big)$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss}\left(\mathcal{R}^a_{ss} + \gamma \max_a Q^*(s',a')\right)$$

הראינו שתי דרכים להגיע לפתרון האופטימלי:

1. Policy iteration המורכב מ-Policy evaluation ולאחריו Policy improvement.

2. Value iteration – פתרון משוואות בלמן באופן ישיר בעזרת איטרציות על ה-Value function.

כאמור, דרכי פתרון אלו מניחים שהמודל ידוע, ובנוסף שמרחב המצבים אינו גדול מדי ויכול להיות מיוצג בזיכרון. האתגר האמיתי מתחיל בנקודה בה לפחות אחת מהנחות אלה אינה תקפה, ולמעשה פה מתחיל התפקיד של אלגוריתמי RL. עיקר ההתמקדות של אלגוריתמים אלו יהיה למצוא באופן יעיל את האסטרטגיה האופטימלית כאשר לא נתונים הפרמטרים של המודל, ואז צריך לשערך אותם (Model-based learning) או למצוא דרך אחרת לחישוב האסטרטגיה האופטימלית ללא שימוש במודל (Model free learning). אם למשל יש משחק בין משתמש לבין המחשב, אלגוריתמים השייכים ל-Model based learning ינסו ללמוד את המודל של המשחק או להשתמש במודל

קיים, ובעזרת המודל הם ינסו לבחון כיצד יגיב המשתמש לכל תור שהמחשב יבחר. לעומת זאת אלגוריתמים מסוג Model free learning לא יתעניינו בכך, אלא ינסו ללמוד ישירות את האסטרטגיה הטובה ביותר עבור המחשב.

היתרון המשמעותי של אלגוריתמים המסתכלים על המודל של הבעיה (Model-based) נובע מהיכולת לתכנן מספר צעדים קדימה, כאשר עבור כל בחירה של פעולה המודל בוחן את התגובות האפשריות, את הפעולות המתאימות לכל תגובה, וכך הלאה. דוגמא מפורסמת לכך היא תוכנת המחשב AlphaZero שאומנה לשחק משחקי לוח כגון שחמט או גו. במקרים אלו המודל הוא המשחק והחוקים שלו, והתוכנה משתמשת בידע הזה בכדי לבחון את כל הפעולות והתגובות למשך מספר צעדים רב ובחירה של הצעד הטוב ביותר.

עם זאת, בדרך כלל אף בשלב האימון אין לסוכן מידע חיצוני מהו הצעד הנכון באופן אולטימטיבי, ועליו ללמוד רק מהניסיון. עובדה זו מציבה כמה אתגרים, כאשר העיקרי ביניהם הוא הסכנה שהאסטרטגיה הנלמדת תהיה טובה רק עבור המקרים אותם ראה הסוכן, אך לא תתאים למקרים חדשים שיבואו. אלגוריתמים שמחפשים באופן ישיר את האסטרטגיה האופטימלית אמנם לא משתמשים בידע שיכול להגיע מבחינת צעדים עתידיים, אך הם הרבה יותר פשוטים למימוש ולאימון.

באופן מעט יותר פורמלי ניתן לנסח את ההבדל בין הגישות כך: גישת Model-based learning מנסה למצוא את הפרמטרים המגדירים את המודל $\{\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{R}\}$ ואז בעזרתם לחשב את האסטרטגיה האופטימלית (למשל בעזרת משוואות בלמן). הגישה השנייה לעומת זאת לא מעוניינת לחשב במפורש את הפרמטרים של המודל אלא למצוא באופן ישיר את האסטרטגיה האופטימלית $\pi(a_t|s_t)$ שעבור כל מצב קובעת באיזה פעולה לנקוט. ההבדל בין הגישות נוגע גם לפונקציית המחיר לה נרצה למצוא אופטימום.

בכל אחד משני סוגי הלמידה יש אלגוריתמים שונים, כאשר הם נבדלים אחד מהשני בשאלה מהו האובייקט אותו מעוניינים ללמוד.

Model-free learning

בגישה זו יש שתי קטגוריות מרכזיות של אלגוריתמים:

א. Policy Optimization – ניסוח האסטרטגיה כבעיית אופטימיזציה של מציאת סט הפרמטרים θ הממקסם את $\pi_\theta(a|s)$. פתרון בעיה זו יכול להיעשות באופן ישיר על ידי שיטת Gradient Ascent עבור פונקציית המחיר $\mathcal{J}(\pi_\theta)=\mathbb{E}[R(\tau)]$, או בעזרת קירוב פונקציה זו ומציאת מקסימום עבורה.
ב. Q-learning – שערוך $Q^*(s,a)$ על ידי $Q_\theta(s,a)$. מציאת המשערך האופטימלי יכולה להתבצע על ידי חיפוש θ שיספק את השערוך הטוב ביותר שניתן למצוא, או על ידי מציאת הפעולה שתמקסם את המשערך: $a(s)=\arg\max_a Q_\theta(s,a)$.

השיטות המנסות למצוא אופטימום לאסטרטגיה הן לרוב on-policy, כלומר כל פעולה נקבעת על בסיס האסטרטגיה המעודכנת לפי הפעולה הקודמת. Q-learning לעומת זאת הוא לרוב אלגוריתם off-policy, כלומר בכל פעולה ניתן להשתמש בכל המידע שנצבר עד כה. היתרון של שיטות האופטימיזציה נובע מכך שהן מנסות למצוא באופן ישיר את האסטרטגיה הטובה ביותר, בעוד שאלגוריתם Q-learning רק משערך את $Q^*(s,a)$, ולעיתים השערוך לא מספיק ואז התוצאה המתקבלת אינה מספיק טובה. מצד שני, כאשר השערוך מוצלח, הביצועים של Q-learning טובים יותר, כיוון שהשימוש במידע על העבר מנוצל בצורה יעילה יותר מאשר באלגוריתמים המבצעים אופטימיזציה של האסטרטגיה. שתי הגישות האלה אינן זרות לחלוטין, וישנם אלגוריתמים שמנסים לשלב בין הרעיונות ולנצל את החוזקות והיתרונות שיש לכל גישה.

Model-based learning

גם בגישה זו יש שתי קטגוריות מרכזיות של אלגוריתמים:

א. Model-based RL with a learned model – אלגוריתמים המנסים ללמוד הן את המודל עצמו והן את ה-Value function או את האסטרטגיה π.
ב. Model-based RL with a known model – אלגוריתמים המנסים למצוא את ה-Value function ו/או את האסטרטגיה כאשר המודל עצמו נתון.

ההבדל בין הקטגוריות טמון באתגר איתו מנסים להתמודד. במקרים בהם המודל ידוע, הממד של אי הוודאות לא קיים, ולכן ניתן להתמקד בביצועים אסימפטומטיים. במקרים בהם המודל אינו ידוע, הדגש העיקרי הוא על למידת המודל.

11.2 Model Free Prediction

לאחר שסקרנו בפרק המבוא את הבסיס המתמטי של בעיות RL והצגנו את משוואות בלמן ופתרונן, בפרקים הבאים נציג גישות שונות להתמודדות עם בעיות RL עבורן פתרונות אלה אינן מספיקים – או מפני שהמודל אינו ידוע או מפני שהן ב-Scale גדול יותר מזה שניתן לפתור באמצעות משוואות בלמן. בפרק זה נציג שתי שיטות הבאות להתמודד עם מקרים בהם המודל אינו ידוע (כלומר האלגוריתם הינו **Model-Free**), והדרך שלהן להתמודד עם אתגר זה הינו **לשערך** את האסטרטגיה האופטימלית בדרכים אחרות שאינן מצריכות את ידיעת המודל.

11.2.1 Monte-Carlo (MC) Policy Evaluation

האלגוריתם הראשון אותו נציג הינו Monte Carlo, והוא מציע דרך לשערך את ה-Value function בלי לדעת את המודל. ראשית נסביר בקצרה מהו אלגוריתם Monte Carlo ואז נראה כיצד ניתן ליישם אותו בבעיות RL.

נניח ונרצה לשערך תוחלת של פונקציית התפלגות כלשהיא – $\mathbb{E}_p[f(x)]$. התוחלת יכולה להיות סכום או אינטגרל שקשה מאוד לחשב. ניתן לשערך את התוחלת על ידי דגימות רנדומליות מההתפלגות וחישוב הממוצע של הדגימות:

$$x_1, \dots x_n \sim p(x)$$

$$\mathbb{E}_p[f(x)] \approx \frac{1}{n}\sum_i f(x_i)$$

לפי חוק המספרים הגדולים הממוצע של הדגימות מתכנס לתוחלת. משערך זה הינו חסר הטיה, ובנוסף השונות שלה קטנה ביחס לינארי לכמות הדגימות:

$$\mathbb{E}\left[\frac{1}{n}\sum_i f(x_i)\right] = \mathbb{E}[f(x)]$$

$$Var\left[\frac{1}{n}\sum_i f(x_i)\right] = \frac{Var[f(x)]}{n}$$

כאמור לעיל, ה-Value function הינה תוחלת עבור אסטרטגיה נתונה π, כאשר מתחילים ממצב s:

$$\mathcal{V}^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} | s_t = s\right]$$

אמנם התוחלת היא על סכום אינסופי, אך עבור מקרים רבים נוכל להניח שהריצה היא סופית (Episodic MDP). בפועל נבצע את הפעולה הבאה: עבור אסטרטגיה נתונה π, נייצר ממנה ריצה של T צעדים, ואז ניקח מצב מסוים ונסתכל על כל ה-rewards שמתקבלים בריצה זו החל ממנו. באופן הזה קיבלנו value עבור הערך של אותו מצב. חזרה על אותה פעולה שוב ושוב תייצר ריצות שונות וממילא ערכים שונים למצב מסוים, ומיצוע על פני הערכים ייתן לנו שערוך לערך האמיתי של אותו מצב. נעיר בהערת אגב שכיוון שמצבים יכולים לחזור על עצמם, נוצרת בעיה שבריצה כזו המצבים אינם בלתי תלויים. בכדי להתגבר על כך, אם מצב חוזר על עצמו יותר מפעם אחת מאפשרים לדגום רק את המופע הראשון של אותו מצב ולא את יתר המופעים (ישנן עוד דרכים להתגבר על כך, אך זוהי הדרך פשוטה ביותר). באופן פורמלי, נניח ויש לנו ריצה של T צעדים:

$$S_1, A_1, R_1, \dots, S_{T-1}, A_{T-1}, R_{T-1}, S_T$$

אז דגימה אחת מתוך ההתפלגות $p_{\pi}(\sum_{k=0}^{\infty} r_{t+k+1} | s_t = S_t)$ תראה באופן הבא:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

למשל, הדגימה G_1 תהיה מורכבת מכל ה-rewards שהגיעו לאחר הצעד הראשון: $R_1 + \gamma R_2 + \dots + \gamma^{T-1} R_T$. הסכום הזה ייתן לנו value עבור אותו מצב ממנו התחלנו (S_1), ועל ידי שערוך אותו מצב שוב ושוב ביחס לריצות שונות נוכל לקבל ערכים שונים, ולאחר מכן למצע אותם בכדי לשערך את ה-value האמיתי של אותו מצב S_1.

באופן פורמלי, העדכון של מצב לאחר כל דגימה נראה כך:

$$\#sample\ of\ s_t: N(S_t) = N(S_t) + 1$$

$$update\ the\ value\ of\ s_t: \mathcal{V}(s_t) = \mathcal{V}(s_t) + \frac{1}{N(S_t)}\big(G_t - \mathcal{V}(s_t)\big)$$

ולאחר הרבה דגימות השערוך מתקבל על ידי התוחלת שלהן:

$$\mathcal{V}(s) = \mathbb{E}_\pi[G_t | s_t = s]$$

באופן סכמתי ניתן לתאר את האלגוריתם באופן הבא:

First-visit MC prediction, for estimating $V \approx v_\pi$

```
Initialize:
    π ← policy to be evaluated
    V ← an arbitrary state-value function
    Returns(s) ← an empty list, for all s ∈ S

Repeat forever:
    Generate an episode using π
    For each state s appearing in the episode:
        G ← the return that follows the first occurrence of s
        Append G to Returns(s)
        V(s) ← average(Returns(s))
```

איור 11.5 אלגוריתם MC עבור שערוך ה-Value function בהינתן Policy. עבור כל מצב מאתחלים רשימה ריקה, ולאחר מכן מייצרים המון ריצות שונות. עבור כל ריצה, עוברים על כל המצבים ובודקים מה ה-G שלהם, ומוסיפים אותו לרשימה של אותו מצב. לבסוף מחשבים את ה-value של כל מצב על ידי מיצוע הרשימה (=ערכי G השונים) של אותו מצב.

יש מספר יתרונות לשיטה זו: היא מספקת משערך חסר הטיה עבור ה-Value function, מבטיחה התכנסות אחרי מספיק איטרציות, ובנוסף ניתן לשערך באמצעותה את השגיאה. אפשר גם באותה דרך לשערך גם את $\mathcal{Q}^\pi(s,a)$ אך זה יהיה יותר רועש וידרוש יותר דגימות. מן הצד השני יש גם חסרונות לשיטה זו: ראשית, היא מתאימה רק ל-Episodic MDP ולא לריצות אינסופיות. עם בעיה זו ניתן להתמודד בקלות כיוון שעבור בעיה אינסופית ניתן לקחת ריצה סופית ולחסום את השגיאה באמצעות γ. שנית, ההתכנסות יכולה להיות מאוד איטית, ובנוסף השונות יחסית גבוהה.

11.2.2 Temporal Difference (TD) – Bootstrapping

במקום להסתכל על ריצה שלמה, ניתן אחרי כל צעד לעדכן את ה-Value. ממשוואת בלמן ניתן להראות שהביטוי $R_{t+1} + \gamma\mathcal{V}^\pi(S_{t+1})$ הינו משערך חסר הטיה עבור $\mathcal{V}^\pi(S_t)$. אי אפשר להשתמש במשערך זה כמו שהוא, כיוון שאנחנו לא יודעים את ה-Value function, וממילא הביטוי $\mathcal{V}^\pi(S_{t+1})$ לא ידוע. בשביל בכל זאת לשערך את $\mathcal{V}^\pi(S)$ באמצעות אותו משערך, ניתן להחליף את $\mathcal{V}^\pi(S_{t+1})$ ב-$\mathcal{V}(S_{t+1})$, ולבצע את השערוך באופן הבא:

$$\mathcal{V}^\pi(S_t) = R_{t+1} + \gamma\mathcal{V}(S_{t+1})$$

הרעיון מאחורי השימוש הזה הוא להיעזר במידע שיש לנו מ-R_t. נניח וננחש ערך כלשהוא עבור $\mathcal{V}^\pi(S_t)$ וניחוש זה יהיה גרוע. אפשר מעט לשפר את הניחוש באמצעות ניחוש $\mathcal{V}(S_{t+1})$ ושימוש ב-reward R_{t+1} שהתקבל עבור אותו מצב S_t, שבעצם מספק מידע כלשהוא על מצב זה. שערוך זה עדיף על ניחוש מוחלט כיוון שהאלמנט של הניחוש מקבל משקל נמוך יותר עקב המכפלה ב- γ, ויש יותר משקל ל-R_{t+1} שמספק מידע אמיתי על המצב S_t. צריך לשים לב שאנו מנסים לשערך את ה-Value function מתוך הערכים שלה בעצמה. מאתחלים את כל הערכים במספרים כלשהם (למשל – וקטור של 0), ואז עוברים צעד צעד ומעדכנים את הניחושים בעזרת התגמולים. אינטואיטיבית זה נראה מעט משונה, אך מסתבר שפתרון זה הוא אחד הכלים החזקים בבעיות RL. באופן פורמלי האלגוריתם נראה כך:

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

איור 11.6 אלגוריתם Temporal Difference (TD) עבור שערוך ה-Value function בהינתן Policy. מנחשים ערך עבור כל מצב ואז באופן איטרטיבי משפרים את הניחושים בעזרת שערוך התלוי ב-reward ובערך המצב הבא.

מגדירים את השגיאה של ה-TD כהפרש שבין הניחוש עבור ערך המצב לבין השערוך שלו:

$$\delta_t = R_{t+1} + \gamma \mathcal{V}(S_{t+1}) - \mathcal{V}(S_t)$$

בשונה משערוך MC, השערוך בשיטת TD הוא בעל הטיה, אך השונות קטנה יותר. בנוסף, כיוון שבכל צעד מבצעים שיפור לערך של מצב, תהליך השערוך יותר מהיר מאשר ב-MC. הרבה פעמים מוסיפים פרמטר α לשערוך (כפי שמופיע באיור 11.6):

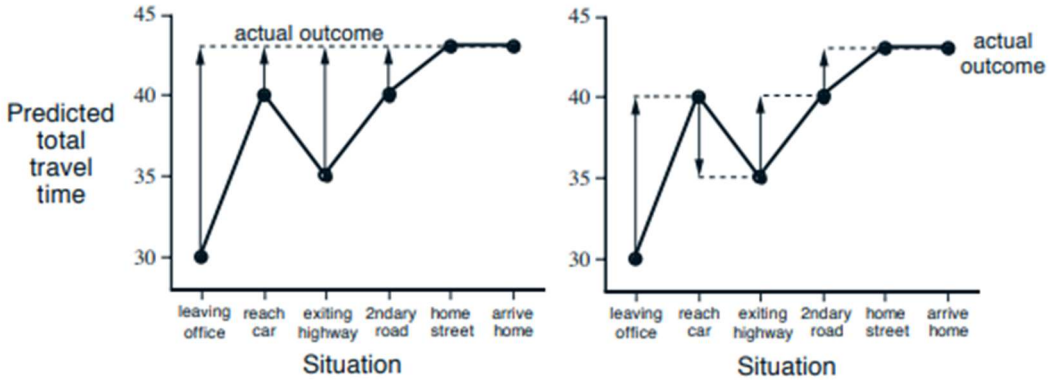
$$\mathcal{V}(S_t) = \mathcal{V}(S_t) + \alpha \cdot [R_{t+1} + \gamma \mathcal{V}(S_{t+1}) - \mathcal{V}(S_t)]$$

עבור ערכי α מתאימים, המשערך מתכנס לתוחלת האמיתית.

ניתן דוגמה שתמחיש את שיטת MC ושיטת TD ואת היחס ביניהם: נהג יצא מהמשרד שלו ונסע לביתו, ובדרך הוא ניסה לשערך את זמן ההגעה שלו וקיבל את הזמנים הבאים:

מצב	כמה זמן עבר	שערוך הזמן שנותר לנסיעה	שערוך זמן הנסיעה הכולל
יציאה מהמשרד	0	30	30
הליכה למכונית תחת גשם	5	35	40
הגעה לכביש מהיר	20	15	35
נסיעה מאחורי משאית	30	10	40
הגעה לרחוב של הבית	40	3	43
הגעה הביתה	43	0	43

נשרטט את שני המשערכים:



איור 11.7 שערוך MC (משמאל) ושערוך TD (מימין) ביחס לתצפית הנתונה.

כיוון ששיטת MC מספקת ערך לאחר ריצה שלמה, אז ניקח את זמן ההגעה בפועל של הנהג וניתן את הערך הזה לכל מצבי הביניים. שיטת TD לעומת זאת מעדכנת את הערך בכל מצב בהתאם למצב הבא. ניתן לראות שהשונות בשערוך TD קטנה מזו שהתקבלה בשערוך MC.

ניתן להסתכל על שיטת TD כבעיית רגרסיה "דינמית":

עבור כל צעד נרצה ש-$\mathcal{V}(S_t)$ יהיה שווה למשוואות בלמן, כלומר נרצה לשערך את $\mathcal{V}(S_t)$ כך שיתקיים:

$$\mathcal{V}(S_t) = \mathbb{E}_\pi[R_{t+1} + \gamma\mathcal{V}(S_{t+1})|s_t = S_t]$$

עבור בעיה זו נוכל להגדיר פונקציית מחיר (Loss) מתאימה:

$$L = \frac{1}{2}\left(\mathbb{E}_\pi[R_{t+1} + \gamma\mathcal{V}(S_{t+1})|s_t = S_t] - \mathcal{V}(S_t)\right)^2$$

את הפונקציה הזו ניתן למזער על ידי דגימות סטוכסטיות של $R_{t+1} + \gamma\mathcal{V}(S_{t+1})$. נשים לב לדבר חשוב – המטרה שלנו היא **לשערך את ההווה באמצעות העתיד ולא להיפך**, כיוון שהעתיד הוא בעל יותר מידע – הוא ראה reward ומצב חדש. הבחנה זו משפיעה על איך שאנחנו רוצים שפונקציית המחיר תתנהג – אנחנו רוצים ש-$\mathcal{V}(S_t)$ יתקרב לערך של $\mathbb{E}_\pi[R_{t+1} + \gamma\mathcal{V}(S_{t+1})|s_t = S_t]$ ולא להיפך. בדוגמה של הנהג שמשערך את זמן הנסיעה – הוא רוצה לעדכן את השערוך של כל מצב בהתאם למצב הבא. אם למשל ההערכה שלו בזמן t הינה 30 דקות ואז הוא מגיע לפקק ומעדכן את ההערכה ל-35 דקות, אז הוא ירצה לתקן את השערוך הקודם $\left(\mathcal{V}(S_t)\right)$ כך שיהיה דומה לשערוך הנוכחי $\left(R_{t+1} + \gamma\mathcal{V}(S_{t+1})\right)$, ולא לעדכן את השערוך הנוכחי כך שיהיה דומה לקודם. בכדי לדאוג לכך, נתייחס לעתיד כקבוע ולא נחשב עבורו גרדיאנט (למרות ש-$\mathcal{V}$ מופיע בו). באופן פורמלי נוכל לנסח זאת כך:

$$T(S_t) = \mathbb{E}_\pi[R_{t+1} + \gamma\mathcal{V}(S_{t+1})|s_t = S_t]$$

$$L = \frac{1}{2}\left(T(S_t) - \mathcal{V}(S_t)\right)^2$$

ואז הגרדינאנט יהיה:

$$\frac{\partial L}{\partial \mathcal{V}} = T(S_t) - \mathcal{V}(S_t)$$

בהתאם לכך, הדגימה $R_{t+1} + \gamma\mathcal{V}(S_{t+1}) - \mathcal{V}(S_t)$ הינה שערוך סטוכסטי לגרדיאנט. כיוון שכל צעד תלוי בצעד הבא, אז המטרה $T(S_t)$ משתנה בכל צעד (אמנם קיבענו אותה בכל צעד יחיד, אך היא עדיין תלויה ב-$\mathcal{V}(S_t)$). במובן הזה בעיית הרגרסיה שהגדרנו הינה "דינמית", כיוון שהמטרה משתנה בכל צעד.

11.3.2 TD(λ)

ננסה לבחון את הקשר בין שתי השיטות שראינו. שערוך TD מבצע דגימה של ריצה ואז מעדכן את הערך של כל מצב בהתאם **למצב הבא** בלבד:

$$\mathcal{V}(S_t) = \mathcal{V}(S_t) + \alpha \cdot [\boldsymbol{R_{t+1}} + \gamma\boldsymbol{\mathcal{V}(S_{t+1})} - \mathcal{V}(S_t)]$$

בגלל ההתייחסות למצב אחד בכל פעם השונות של המשערך נמוכה, אך יש הטיה.

שיטת MC לעומת זאת דוגמת ריצה ומעדכנת את הערך של כל מצב בהתאם **לכל המצבים** שבאים לאחר מכן:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1}R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

$$\mathcal{V}(s_t) = \mathcal{V}(s_t) + \frac{1}{N(S_t)}\left(G_t - \mathcal{V}(s_t)\right)$$

משערך זה הינו חסר הטיה, אך עם זאת השונות שלו גבוהה.

נראה כיצד ניתן לחבר בין שני המשערכים ולמצוא שיטה שתהיה אופטימלית מבחינת היחס שבין ההטיה לשונות. ניתן להכליל את שני המשערכים לנוסחה כללית באופן הבא:

$$\mathcal{V}(s_t) = \mathcal{V}(s_t) + \alpha\left(G_t^{(n)} - \mathcal{V}(s_t)\right)$$

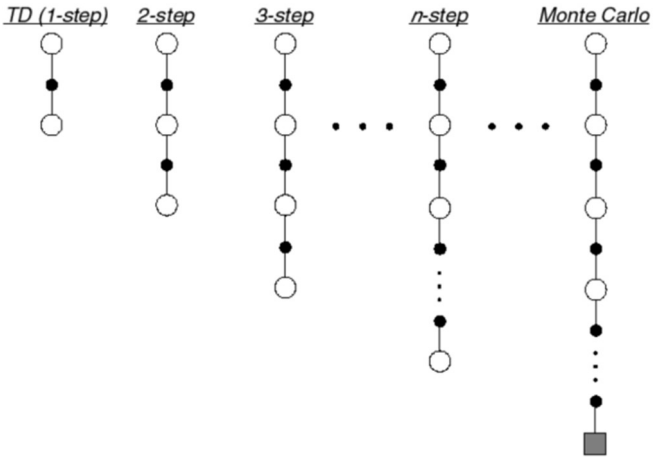
כאשר $G_t^{(n)}$ הוא הסכום של n ה-rewards הבאים:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \mathcal{V}(s_{t+n}) = \sum_{k=0}^{n-t-1} \gamma^k R_{t+k+1}$$

כעת נוכל להגדיר:

$$TD(n) \rightarrow \mathcal{V}(s_t) = \mathcal{V}(s_t) + \alpha \left(G_t^{(n)} - \mathcal{V}(s_t) \right)$$

ולפי סימון זה נוכל לשים לב שמשערך MC הוא למעשה $TD(n \rightarrow \infty)$ ואילו משערך TD שווה ל-$TD(1)$.



איור 11.8 משערך $TD(n)$. משערך MC הינו המקרה הפרטי עבורו $n \rightarrow \infty$ ואילו משערך TD הינו המקרה הפרטי בו $TD(n = 1)$.

אם ניקח $1 < n < \infty$ נקבל משערך "ממוצע" בין MC לבין TD. ניתן להציע גרסה יותר טובה למשערך זה תחת ההנחה שכל שמאורעות סמוכים אחד לשני כך יש להם יותר השפעה. בדומה ל-discount factor שמוריד את ההשפעה של תגמול ככל שהוא יותר רחוק מהמצב הנוכחי, כך גם כאן ניתן לכל מאורע עתידי משקל הולך וקטן. נדאג שכל המשקלים יסתכמו ל-1 ונקבל את השערוך הבא, הידוע גם בכינוי $TD(\lambda)$:

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

$$\mathcal{V}(s_t) = \mathcal{V}(s_t) + \alpha \left(G_t^{\lambda} - \mathcal{V}(s_t) \right)$$

שערוך זה הוא בעל שונות גדולה יותר מאשר TD, כיוון שהוא מתחשב ביותר צעדים, אך ההטיה שלו קטנה יותר. כאמור, הוא מנסה למצע בין שני המשערכים שראינו ולאזן בין השונות להטיה.

נסכם בקצרה את מה שראינו בפרק זה. התייחסנו למקרים בהם המודל אינו ידוע ואנו רוצים לשערך אותו, והצגנו שלוש גישות המנסות לשערך את המודל בעזרת דגימות סטוכסטיות: MC, TD (שזהו מקרה פרטי של $TD(n)$) וגישה המנסה לשלב בין השתיים – $TD(\lambda)$.

לסיום נעיר ששערוך האסטרטגיה כשלעצמה אינו מספיק, כיוון שהשערוך אמנם מספק אסטרטגיה מסוימת עבור הבעיה, אך היא אינה בהכרח אופטימלית. בפרק הקודם ראינו כיצד בהינתן המודל ניתן לשפר אותו ולמצוא את האסטרטגיה האופטימלית. יכולנו לעשות זאת כיוון שידענו את המודל במלואו, מה שאיפשר לבצע בכל צעד מהלך חמדני ובכך להתכנס לבסוף לאסטרטגיה האופטימלית. במקרים בהם אנו רק משערכים את האסטרטגיה ללא ידיעת המודל, אין לנו בהכרח מידע מספיק טוב עבור כל המצבים. מצבים ופעולות שלא נוסו כלל (או נוסו רק בפעמים נדירות), השערוך עבורם יכול להיות די גרוע, ואז השיפור באמצעות בחירה גרידית לא בהכרח יכול להביא את האסטרטגיה להיות אופטימלית.

11.3 Model Free Control

בפרק הקודם ראינו כיצד ניתן לשערך את המודל באמצעות דגימות סטוכסטיות. שיטות השערוך אפשרו לנו לקבל מידע על המודל, אך הן אינן התייחסו לשאלה האם הוא אופטימלי. בפרק הראשון ראינו כיצד ניתן לקחת מודל או

אסטרטגיה ולהביא אותם לאופטימליות, אך אי אפשר להשתמש בשיטה זו עבור מודל משוערך. הסיבה לכך נעוצה שמודל זה לא בהכרח ראה את כל הזוגות האפשריים של המצבים והפעולות, וממילא הוא לא יכול לשפר את הבחירות שלו על סמך אסטרטגיה גרידית. ניקח לדוגמה מקרה בו עבור מצב מסוים האסטרטגיה הינה דטרמיניסטית והפעולה שנבחרת הינה תמיד ללכת למעלה. במקרה כזה אין לנו שום מידע על יתר הפעולות האפשריות במצב זה ולכן המודל שלנו לא שלם, וממילא לא נוכל להשתמש ב-Policy improvement המבוסס על כך שיש לנו מידע על כל המצבים והפעולות.

בכדי להתמודד עם בעיה זו נהיה חייבים לדאוג לכך שנבקר בכל המצבים. כמובן שנוכל לנקוט בגישה פשטנית של אסטרטגיה אקראית, שאחרי מספר מספיק גדול של פעולות קרוב לוודאי שנבקר בכל המצבים. אסטרטגיה זו אמנם טובה לבדיקת מצבים ופעולות חדשים, אך כמובן שהיא רחוקה מלהיות אופטימלית, לכן נרצה להשתמש באסטרטגיה שמצד אחד מנסה להיות אופטימלית ומצד שני יש בה ממד של אקראיות המביא לכך שנבקר גם במצבים שלא היינו מגיעים אליהם לפי האסטרטגיה הנוכחית. בחירת פעולות בהתאם לאסטרטגיה הנוכחית נקראת **Exploitation** ואילו בחירה של מצבים חדשים נקראת **Exploration**, והמטרה שלנו תהיה לאזן בין השניים תחת הדרישות הבאות:

1. ביקור בכל הזוגות של המצבים והפעולות אינסוף פעמים.

2. ככל שמספר הצעדים גדל, כך האסטרטגיה מתכנסת לאסטרטגיה הגרידית:

$$\lim_{k\to\infty} \pi_k(a|s) = 1\left[a = \arg\max_{a'} Q(s,a')\right]$$

מצד אחד נרצה לבקר בכל המצבים ומצד שני נרצה שככל שנעשה יותר צעדים ככה נשפר את האסטרטגיה שלנו. אסטרטגיה המקיימת דרישות אלה נקראת **Greedy in the Limit of Infinite Exploration (GLIE)**, וניתן להוכיח שקיום דרישות אלה מביא את האסטרטגיה להיות אופטימלית. דוגמה לאסטרטגיה פשוטה העונה על הדרישות הינה $\epsilon - greedy$ – בחירה של האסטרטגיה האופטימלית בהסתברות $(1-\epsilon)$ ובחירה מצב אחר בהסתברות ϵ. עבור ϵ שהולך וקטן עד ל-0 בקצב שאינו מהיר מדי, אסטרטגיה זו הינה GLIE.

לעיל הראינו כיצד בהינתן מודל ניתן לשפר את האסטרטגיה (Policy improvement) על ידי כך שבחרנו באופן גרידי פעולות. כעת נרצה להראות שבאופן דומה מתקיים אותו רעיון גם עבור $\epsilon - greedy$, כלומר שאם השתמשנו בשיטה זו והגענו ל-Value function, אז ניתן בצעד הבא לשפר את ה-Value על ידי אסטרטגיה ϵ-גרידית. אסטרטגיה זו בוחרת באופן גרידי את הפעולה האופטימלית לפי האסטרטגיה הנתונה, אך נותנת לכל יתר הפעולות הסתברות הגדולה או שווה להסתברות שהייתה לפעולה זו בשימוש באסטרטגיה ϵ-גרידית. ננסח את המשפט באופן פורמלי:

$$\text{If Policy } \pi_i \text{ has } \forall s,a: \pi_i(a|s) \geq \frac{\epsilon}{|A|} \text{ and } \pi_{i+1} \text{ is } \epsilon - \text{greedy w.r.t } Q^{\pi_i}, \text{then } \mathcal{V}^{\pi_{i+1}} \geq \mathcal{V}^{\pi_i}$$

הוכחה: נסתכל על המקרה בו נוקטים צעד אחד גרידי לפי π_{i+1} ואז ממשיכים לפי האסטרטגיה הקודמת π_i:

$$\sum_a \pi_{i+1}(a|s) Q^{\pi_i}(s,a) = \sum_a \frac{\epsilon}{|A|} Q^{\pi_i}(s,a) + (1-\epsilon)\max_a Q^{\pi_i}(s,a)$$

נרשום את $(1-\epsilon)$ בצורה אחרת: במקום ϵ נרשום $\sum_a \frac{\epsilon}{|A|}$, ובמקום 1 נרשום $\sum_a \pi_i(a|s)$ (הסכום אכן שווה ל-1 כי סוכמים את כל האפשרויות עבור התפלגות נתונה). נקבל:

$$= \sum_a \frac{\epsilon}{|A|} Q^{\pi_i}(s,a) + \max_a Q^{\pi_i}(s,a) \sum_a \left(\pi_i(a|s) + \frac{\epsilon}{|A|}\right)$$

כעת נחליף את הביטוי $\max_a Q^{\pi_i}(s,a)$ באסטרטגיה עצמה $Q^{\pi_i}(s,a)$. כיוון שמתקיים $Q^{\pi_i}(s,a) \leq \max_a Q^{\pi_i}(s,a)$, נקבל:

$$\geq \sum_a \frac{\epsilon}{|A|} Q^{\pi_i}(s,a) + Q^{\pi_i}(s,a) \sum_a \left(\pi_i(a|s) - \frac{\epsilon}{|A|}\right)$$

כעת יש שני איברים זהים שמצטמצמים, ונשאר עם:

$$= \sum_a \pi_i(a|s)\, Q^{\pi_i}(s,a)$$

ובסך הכל קיבלנו:

$$\sum_a \pi_{i+1}(a|s)Q^{\pi_i}(s,a) \geq \sum_a \pi_i(a|s)\, Q^{\pi_i}(s,a)$$

הביטוי שהתקבל מסמל את ה-value function כאשר עוקבים אחר האסטרטגיה π_i. יוצא מכך שאם עושים צעד אחד לפי האסטרטגיה π_{i+1} ואז ממשיכים לפי π_i, זה בהכרח יותר טוב מאשר גם את הצעד הראשון לעשות לפי π_i. כעת בדומה להוכחה שהראינו לעיל, ניתן להוכיח שמתקיים $V^{\pi_i}(s) \leq V^{\pi_{i+1}}(s)$ וביצוע השיפור שוב ושוב יביא את האסטרטגיה להתכנס לזו האופטימלית. הבעיה בשיטה זו היא חוסר היעילות שבה, כיוון שהיא דורשת המון דגימות והמון איטרציות. עקב כך שיטה זו לא פרקטית, ובמקומה נציג כעת שיטות אחרות המאפשרות לשפר את האסטרטגיה עבור מודל משוערך. פורמלית, שיטות אלה נכנסות תחת קטגוריה הנקראת Model Free Control – שליטה (ושיפור) אסטרטגיה שאינה מתבססת על מודל ידוע מראש.

11.3.1 SARSA – On-Policy TD control

בפרק הקודם ראינו כיצד ניתן באמצעות שערוך TD לשפר את ה-Value function, כאשר העדכון בכל צעד מתקיים באופן הבא:

$$\mathcal{V}(S_t) = \mathcal{V}(S_t) + \alpha \cdot [R_{t+1} + \gamma \mathcal{V}(S_{t+1}) - \mathcal{V}(S_t)]$$

כעת אנחנו מתעניינים באסטרטגיה ולא רק ב-Value function, לכן במקום לעדכן את $\mathcal{V}(S_t)$ נעדכן את פונקציית ה-state-action $\mathcal{Q}(S_t, A_t)$. באופן הזה נקבל טבלה בגודל $\mathcal{Q}: |S| \times |A|$ המכילה מידע על כל הזוגות האפשריים של (S, A), ועבור כל זוג יש ערך מסוים (טבלה זו נקראת $\mathcal{Q} - \text{table}$). בהתחלה הערכים בטבלה לא ישקפו את הערכים האמיתיים, אך עם התקדמות הלמידה הטבלה תשתפר ואולי אף תתכנס לאופטימליות. העדכון מתבצע באופן הבא:

$$\mathcal{Q}(S_t, A_t) = \mathcal{Q}(S_t, A_t) + \alpha \cdot [R_{t+1} + \gamma \mathcal{Q}(S_{t+1}, A_{t+1}) - \mathcal{Q}(S_t, A_t)]$$

העדכון בכל צעד מתבצע על סמך המצבים והפעולות של שתי יחידות זמן: $\{S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}\}$ ולכן האלגוריתם נקרא SARSA. אלגוריתם זה הינו **On-Policy Learning**, כלומר, העדכון בכל צעד נעשה על סמך מידע המגיע מהאסטרטגיה הידועה באותו זמן: בוחרים לבצע פעולה A_t במצב S_t ($\mathcal{Q}(S_t, A_t)$) בהתאם לאסטרטגיה, ואז מעדכנים אותה על סמך התגמול R_{t+1} שהתקבל בעקבות הפעולה A_t. באופן סכמתי איטרציה אחת של האלגוריתם מתוארת באופן הבא:

```
Initialize Q(s,a), ∀s ∈ S, a ∈ A(s), arbitrarily, and Q(terminal-state, ·) = 0
Repeat (for each episode):
   Initialize S
   Choose A from S using policy derived from Q (e.g., ε-greedy)
   Repeat (for each step of episode):
      Take action A, observe R, S'
      Choose A' from S' using policy derived from Q (e.g., ε-greedy)
      Q(S,A) ← Q(S,A) + α[R + γQ(S',A') − Q(S,A)]
      S ← S'; A ← A';
   until S is terminal
```

איור 11.7 אלגוריתם SARSA. שערוך on-policy של $\mathcal{Q}(s,a)$ בעזרת דגימות מאסטרטגיה ϵ-גרידית ביחס ל-$\mathcal{Q}$ הנוכחי.

בכל מצב הפעולה שנבחרת הינה ϵ-גרידית ביחס ל-$\mathcal{Q}(s,a)$ הנוכחי. כלומר: אם במצב S_t יש לנקוט לפי האסטרטגיה את המצב $A_t = \bar{A}$, אזי האסטרטגיה ה-ϵ-גרידית ביחס לכך הינה:

$$A_t = \begin{cases} \bar{A}\ w.p & 1-\epsilon \\ A \neq \bar{A} & w.p\ \epsilon \end{cases}$$

ניתן להוכיח שאלגוריתם SARSA מביא את האסטרטגיה לאופטימליות תחת שני תנאים:

א. שהאסטרטגיה תהיה GLIE.

ב. שיתקיים תנאי Robbins-Monroe עבור α (תנאי זה דואג לכך שנגיע בהכרח לכל המצבים ומצד שני $\alpha_i \to 0$):

$$\sum_{i=0}^{\infty} \alpha_i = \infty, \sum_{i=0}^{\infty} \alpha_i^2 < \infty$$

לגישה זו, הפועלת בגישת on-policy learning, יש מספר חסרונות:

1. המטרה היא ללמוד את האסטרטגיה האופטימלית אבל בפועל ה-exploration הוא ביחס לאסטרטגיה הנתונה בכל מצב.

2. לא ניתן להשתמש בצעדים ישנים, כיוון שהם מתייחסים לאסטרטגיה שכבר לא רלוונטית.

3. לא ניתן להשתמש במידע שמגיע מבחוץ.

11.3.2 Q-Learning

ניתן להפוך את אלגוריתם SARSA להיות off-policy, והאלגוריתם המתקבל, שנקרא $Q-\text{Learning}$, הוא אחד האלגוריתמים השימושיים בתחום של RL. נתבונן בפונקציית העדכון של SARSA:

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

האלמנט שתלוי באסטרטגיה הינו $Q(S_{t+1}, A_{t+1})$, כיוון שבו אנחנו נוקטים בפעולה A_{t+1} בהתאם לאסטרטגיה. במקום לבחור בפעולה זו, ניתן להיות גרידי ולקחת את הפעולה בעלת הערך הכי גדול בצעד הקרוב, ועל פיה לעדכן את ה-$Q-value$:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \cdot \left[R_{t+1} + \gamma \max_{\boldsymbol{A}} \boldsymbol{Q(S_{t+1}, A_t)} - Q(S_t, A_t)\right]$$

כעת האסטרטגיה אינה משפיעה על פונקציית העדכון, וממילא היא off-policy.

```
Initialize Q(s,a), ∀s ∈ S, a ∈ A(s), arbitrarily, and Q(terminal-state, ·) = 0
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        Q(S,A) ← Q(S,A) + α[R + γ max_a Q(S',a) − Q(S,A)]
        S ← S';
    until S is terminal
```

איור 11.8 אלגוריתם $Q-\text{Learning}$. שערוך off-policy של $Q(s,a)$ בעזרת דגימות מאסטרטגיה ϵ-גרידית ועדכון ה-$Q-value$ בהתאם לפעולה בעלת הערך הכי גדול בצעד הקרוב.

האסטרטגיה לא משפיעה על עדכון ה-$Q-value$, אך כן יש לה השפעה על **כמות** הפעמים שנבקר בכל מצב. בניגוד לאלגוריתם SARSA בו דרשנו שהאסטרטגיה תהיה GLIE, באלגוריתם $Q-\text{Learning}$ הדרישה הינה רק שנבקר אינסוף פעמים בכל מצב. הבדל זה הוא משמעותי, כיוון ש-GLIE דורש שהאסטרטגיה תתכנס לאסטרטגיה הגרידית (כלומר, הדרישה היא ש-ϵ ילך ל-0). דרישה זו מקשה על הלמידה כיוון שצעדים גרידים מביאים מעט מאוד מידע חדש. הסרת הדרישה על ה-ϵ מאפשרת exploration בצורה יותר חופשית, והלמידה נעשית בצורה הרבה יותר מהירה. עם זאת, יותר מדי exploration זה גם לא טוב, כיוון שמבקרים בהרבה מצבים לא רלוונטיים מספר רב של פעמים.

נתבונן על האלגוריתמים שראינו עד כה ונשווה בין הפתרונות שהיו מבוססים על ידיעת המודל לבין פתרונות משוערכים:

Full Backup (Dynamic Programming)	Sample Backup (TD)
Iterative Policy Evaluation: $\mathcal{V}(S) = \mathbb{E}[\mathcal{R} + \gamma \cdot \mathcal{V}(S') \mid S]$	TD Learning: $\mathcal{V}(S) = \mathcal{V}(S) + \alpha \cdot [R_{t+1} + \gamma \mathcal{V}(S') - \mathcal{V}(S)]$
Q-Policy Iteration: $\mathcal{Q}(S,A) = \mathbb{E}[\mathcal{R} + \gamma \cdot \mathcal{Q}(S',A') \mid S, A]$	SARSA: $\mathcal{Q}(S,A) = \mathcal{Q}(S,A) + \alpha \cdot [\mathcal{R} + \gamma \mathcal{Q}(S',A') - \mathcal{Q}(S,A)]$
Q-Value Iteration:	Q-Learning:

$Q(S,A) = \mathbb{E}\left[\mathcal{R} + \gamma \cdot \max_A Q(S,A) \mid S,A\right]$	$Q(S,A) = Q(S,A) + \alpha \cdot \left[\mathcal{R} + \gamma \max_A Q(S',A) - Q(S,A)\right]$

הפתרונות המשוערכים מצליחים להתמודד עם בעיות בינוניות, אך הצורך בדגימות יכול להיות בעייתי משתי סיבות: א. האלגוריתמים נדרשים להמון דגימות בשביל להצליח. ב. חסרון נוסף בגישות שאינן model-based קשור ל-exploration עבור בעיות של העולם האמיתי, לא תמיד אפשרי לראות דוגמאות שליליות כדי ללמוד שהן לא טובות. רכב אוטונומי למשל, אם רוצים שהוא ילמד לא לנסוע ברמזור אדום, אי אפשר לאמן אותו על ידי זה שניתן לו יד חופשית לבצע exploration וכך הוא יגיע למצבים בהם הוא ייסע באור אדום ויקבל על כך תגמול שלילי. לעומת זאת, בעיות הכרוכות בסימולציה הן יותר מתאימות לאלגוריתמים שהצגנו המבוססים על דגימות ושערוך, כיוון שניתן בצורה יחסית זולה לבצע המון דגימות, ובנוסף אין בהן בעיות בטיחות בביצוע ה-exploration.

מלבד בעיית הדגימות והשערוך, האתגר העיקרי של גישות אלה נעוץ ביכולת ההכללה של המודלים הנלמדים. התוצאה של SARSA ו-$Q-$Learning הינה כאמור טבלה של ה-$Q-$value , ובטבלה זו אין שום קשר בין הערכים השונים בטבלה. כל איבר בטבלה עומד בפני עצמו, ואי אפשר ללמוד ממנו על שאר האיברים. אם למשל הגענו למצב חדש שעוד לא ראינו אך ראינו הרבה מצבים דומים לו, לא נוכל ללמוד מהם שום דבר לגבי המצב החדש. אתגר זה משליך גם על גודל הבעיות אותן ניתן לפתור – אם אי אפשר להכליל ממצב אחד למצב אחר, ממילא זה מגביל מאוד את גודל הבעיה איתה ניתן להתמודד בעזרת אלגוריתמים אלו. במקרים בהם מרחב המצבים הוא רציף, אז מרחב המצבים הוא אינסופי ואז בכלל לא ניתן להשתמש בגישות אלו.

11.3.3 Function Approximation

כאמור, אלגוריתמים שמנסים לשערך את ה-$Q-$table אינם ישימים בבעיות גדולות בעיקר בגלל חוסר היכולת שלהם להכליל ממצב אחד למצב אחר. גם אם יש בידינו אפשרות לשמור טבלאות ענקיות של $Q-$value, לא נוכל לשפר את הטבלה ולעדכן בה ערכים אופטימליים, כיוון שלא ניתן להשליך ממצבים בהם ביקרנו על מצבים חדשים. בכדי להתמודד עם בעיה זו, נרצה להחליף את ה-$Q-$table בפונקציה שמחזירה ערך עבור כל זוג של state-action. במקום לחשב טבלה ענקית $Q: |S| \times |A|$, נרצה למצוא פונקציה עם סט פרמטרים θ כך שיתקיים:

$$Q(S,A) \approx Q_\theta(S,A), \mathcal{V}(S) \approx \mathcal{V}_\theta(S)$$

היתרון של שימוש בפונקציה שמנסה לשערך את הערכים הינו כפול: א. לא צריך לשמור טבלה בגודל של מרחב המצבים. ב. כן ניתן ללמוד ממצבים שיש לנו עליהם ידע על מצבים חדשים. בכדי למצוא פרמטרים שישערכו את המודל בצורה איכותית יש לפתור בעיית אופטימיזציה, כפי שנגדיר בהמשך, אך הלמידה יכולה להיות מאוד לא יציבה. ראשית, כפי שראינו ב-TD, המטרה אותה רוצים לשערך ($\mathcal{V}(S_t)$ או $Q(S_t,A_t)$) משתנה בכל צעד. בנוסף, בניגוד לאלגוריתמים הקודמים, כעת יש קשר בין המצבים, ולכן אם אנחנו משנים ערך מסוים, זה גם ישפיע על ערכים אחרים, מה שמקשה מאוד על יציבות הלמידה.

הדוגמה הפשוטה ביותר לפונקציה כזו הינה מודל לינארי מהצורה:

$$Q(S,A) = w^T \phi(S,A)$$

מודל זה מניח שיש סט של פרמטרים המקיים קשר לינארי בין ה-state-action ומפת פיצ'רים כלשהיא $\phi(S,A)$ לבין הערך שלהם $Q(S,A)$. באופן הפשטני ביותר, בשביל למצוא את הפונקציה הזו, נרצה למזער כמה שיותר את המרחק שבין $\mathcal{V}^\pi(S)$ לבין $\mathcal{V}_\theta(S)$, ולשם כך נבנה את פונקציית המטרה הבאה:

$$L(\theta) = \frac{1}{2}\mathbb{E}_\theta\left[\left(\mathcal{V}^\pi(S) - \mathcal{V}_\theta(S)\right)^2\right]$$

כמובן שחישוב זה אינו אפשרי משום שאנחנו לא יודעים מה הוא $\mathcal{V}^\pi(S)$ – זה בדיוק הביטוי אותו אנו רוצים לשערך! בנוסף, חישוב התוחלת יכול להיות מאוד מסובך, בטח במקרה בו אנו לא יודעים את כל הערכים האמיתיים של $\mathcal{V}^\pi(S)$. בשביל להתגבר על בעיות אלו נשים לב כיצד כן ניתן לשפר את $\mathcal{V}_\theta(S)$ – אם נגזור את פונקציית המטרה ונבצע צעד בכיוון הגרדיאנט, נקטין את ערכה, ולכן נרצה להיות מסוגלים לחשב את $\Delta\theta$. נבצע זאת בדומה בעזרת דגימות סטוכסטיות, כפי שכבר ראינו בפרקים קודמים, כלומר נרצה לדגום מצבים מהאסטרטגיה בשביל לחשב את הביטוי הבא:

$$\Delta\theta = \left(\mathcal{V}^\pi(S_t) - \mathcal{V}_\theta(S_t)\right)\nabla_\theta \mathcal{V}_\theta(S_t)$$

עם ביטוי זה עדיין לא ניתן להתקדם כיוון ש-$\mathcal{V}^{\pi}(S)$ לא ידוע, אך גם אותו ניתן לשערך, למשל בעזרת MC או TD. בעזרת שיטת MC ניתן להריץ episode שלם, לחשב את התגמול המצטבר ולשים אותו במקום $\mathcal{V}^{\pi}(S)$, ובעזרת TD ניתן להריץ צעד אחד ולהחליף את $\mathcal{V}^{\pi}(S)$ בתגמול המיידי והשערוך של המצב הבא:

$$\text{MC: } \Delta\theta = \left(G_t - \mathcal{V}_\theta(S_t)\right)\nabla_\theta \mathcal{V}_\theta(S_t)$$

$$\text{TD: } \left(R_{t+1} + \gamma\mathcal{V}_\theta(S_{t+1}) - \mathcal{V}_\theta(S_t)\right)\nabla_\theta \mathcal{V}_\theta(S_t)$$

יש לשים לב שבביטוי האחרון יש שני איברים שתלויים ב-θ – $\mathcal{V}_\theta(S_t)$, $\gamma\mathcal{V}_\theta(S_{t+1})$, אך נגזור רק את הביטוי הראשון כיוון שנרצה לשנות אותו כך שיתקרב לביטוי השני ולא להיפך. עם זאת, בניגוד למה שראינו לעיל בנוגע ל-TD, שם שינוי של $\mathcal{V}(S_t)$ **לא** השפיע על $\mathcal{V}(S_{t+1})$ כיוון שהערכים בטבלה היו בלתי תלויים אחד בשני, כאן כן יש השפעה בין שני ערכים אלו, מה שמקשה על היציבות של הלמידה.

בסופו של דבר, אנו יכולים לחשב את הביטוי $\Delta\theta$ ובעזרתו לנסות לאפטם את הפרמטרים באיזה שיטות אופטימיזציה סטנדרטיות (למשל – stochastic gradient descent).

בדומה לשערוך שהצענו עבור MC ו-TD, ניתן גם לבצע control ולשערך את ה-Q – table:

$$\text{SARSA: } \Delta\theta = \left(R_{t+1} + \gamma Q_\theta(S_{t+1}, A_{t+1}) - Q_\theta(S_t, A_t)\right)\nabla_\theta Q_\theta(S_t, A_t)$$

$$Q - \text{Learning: } \Delta\theta = \left(R_{t+1} + \gamma \max_A Q_\theta(S_{t+1}, A_t) - Q_\theta(S_t, A_t)\right)\nabla_\theta Q_\theta(S_t, A_t)$$

וגם כאן, בגזירה נרצה להתייחס ל-$Q_\theta(S_{t+1}, A_t)$ ו-$Q_\theta(S_{t+1}, A_{t+1})$ כקבועים ולגזור רק את $Q_\theta(S_t, A_t)$, כיוון שנרצה לשנות את $Q_\theta(S_t, A_t)$ כך שיתקרב בערכו לשאר הביטויים המבוססים על מידע נוסף.

יש כמה אתגרים שיכולים להקשות מאוד על שיטות אלה להביא את המודל להתכנסות:

א. ראשית, הדרישה שאומרת שאנו רוצים לשנות את $Q_\theta(S_t, A_t)$ ביחס למידע העתידי הינה בעייתית, כיוון שכעת כל הביטויים תלויים באותו פרמטר θ.

ב. באלגוריתם שהוא off-policy אנו דוגמים מאסטרטגיה ואף פועלים לפי הדגימות האלו, אך אנו לא מנסים להביא לאופטימום דווקא את האסטרטגיה הזו (וממילא היא לא בהכרח האופטימלית עבורנו). כאשר ניסינו ללמוד tabular Q-table, זה היה יכול לגרום לכך ששכיחות המצבים שנדגום אינה תואמת את השכיחות שלהם לפי האסטרטגיה האמיתית, מה שיכול להשפיע על קצב הלמידה, אך זה לא יגרום לשגיאות באסטרטגיה הנלמדת. כעת כאשר יש קשר בין המצבים על ידי הפרמטר θ, אז דגימה לא לפי השכיחות האמיתית יכולה להטות את הלמידה לכיוונים שגויים.

אתגרים אלו משפיעים על יכולת המודל להתכנס לאסטרטגיה האופטימלית. בעוד שעבור יכולנו להוכיח התכנסות, עבור functional-approximation זה כלל לא מובטח, כפי שמוצג בטבלה הבאה:

Algorithm	Tabular	Linear	Non-Linear
MC	Y	Y	Y
SARSA	Y	Y	N
Q-Learning	Y	N	N

אחת השיטות פורצות הדרך התחום הייתה שימוש ברשתות נוירונים בתור ה-functional-approximation, ובשם הנפוץ שלה – Deep Q – Learning. השימוש ברשת נוירונים בפני עצמה לא הייתה מספיקה כיוון שכאמור הלמידה היא מאוד לא יציבה ויש להכניס מנגנונים נוספים שיעזרו למודל להשתפר. נציין שתי טכניקות שהוטמעו בתהליך הלמידה:

1. כאשר דוגמים כל מיני מצבים, יש ביניהם קורלציה יחסית גבוהה, מה שמונע מהשונות של הגרדיאנט לקטון. כדי להתגבר על כך, במקום לעדכן את Q בכל שלב, שומרים את הרביעיות $(S_t, A_t, R_{t+1}, S_{t+1})$ של N המצבים האחרונים שראינו (למשל: $N = 1{,}000$), ואז מעדכנים לפי k מצבים (למשל: $k = 10$) אקראיים מתוך N המצבים הקודמים שיש לנו בזיכרון (לאחר שהזיכרון מתמלא, אז כל מצב חדש שמגיע דורס את המצב הכי ישן בזיכרון). בצורה הזו

השונות תקטן כי הדוגמאות שנעדכן לפיהן יהיו בעלות קורלציה נמוכה הרבה יותר. המחיר של טכניקה זו הוא השימוש בהרבה זיכרון, אבל יש לה השפעה משמעותית על הביצועים.

2. הזכרנו לעיל שאנו רוצים לשנות את $Q_\theta(S_t, A_t)$ ביחס למידע העתידי, אך זה יכול להיות בעייתי, כיוון שכעת כל הביטויים תלויים באותו פרמטר θ. במילים אחרות – אנו רוצים לקרב את האסטרטגיה למטרה מסוימת, אך שתיהן תלויות באותו פרמטר. בכדי לייצב את הרגרסיה, ניתן לשנות את המטרה באופן הרבה פחות תדיר. באופן פורמלי, נשמור סט פרמטרים $\hat{\theta}$ ונשנה אותו כל מספר קבוע של צעדים לסט הפרמטרים החדש, כלומר – כל k צעדים נעדכן את $\hat{\theta}$ להיות שווה ל-θ_t. יחד עם הפרמטרים החדשים, נשנה מעט את הביטוי אותו נרצה להביא לאופטימום:

$$\mathbb{E}_{S_t,R_{t+1},A_t,S_{t+1}}\left[R_{t+1} + \gamma \max_A Q_{\hat{\theta}}(S_{t+1}, A_t) - Q_\theta(S_t, A_t)\right]$$

11.3.4 Policy-Based RL

יש כמה מגבלות וחסרונות בשיטות שראינו עד כה. שיטות אלו מתמקדות בלמידת ה-$Q - table$, בין אם על ידי עדכון ישיר של ערכי הטבלה ובין אם על ידי למידת הפונקציה $Q_\theta(S, A)$ התלויה בפרמטר θ. ראינו שבבעיות גדולות הייצוג והלמידה יכולים להיות קשים. אתגר זה מתעצם ואף נהיה בלתי אפשרי עבור בעיות רציפות, בהן מרחב המצבים הוא כביכול אינסופי (למשל – תנועה של רובוט). חסרון נוסף שיש בגישות שראינו נעוץ במטרה של הלמידה – אנו מנסים ללמוד את $Q_\theta(S, A)$ אך בפועל מה שמעניין אותנו זה $\max_A Q_\theta(S, A)$, מה שיכול להביא לבזבוז משאבים עבור למידה של דברים לא הכרחיים. למשל – אם אנחנו צריכים להחליט בין להשקיע כסף בשוק ההון לבין השקעה בתרמית פירמידה, אז אנחנו רק צריכים להחליט איזו אופציה עדיפה לנו, אך אנו לא נדרשים לדעת מה בדיוק כוללת כל אופציה. ההחלטה בין האופציות היא פשוטה, ואילו ללמוד מה ואיך להשקיע בשוק ההון זה דבר מאוד מורכב. לכן במקום ללמוד את כל המערכת, נוכל פשוט ללמוד מה הפעולות הנדרשות עבורנו, וגישה ישירה זו יכולה לחסוך במשאבים.

כאמור, במקום ללמוד את Q, ננסה ללמוד באופן ישיר את האסטרטגיה האופטימלית – $\pi_\theta(a|s)$. באופן מעט יותר פורמלי נגדיר את פונקציית המטרה:

$$\mathcal{J}(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{t+1}\right]$$

מציאת האסטרטגיה האופטימלית שקולה למציאת המקסימום של פונקציית המטרה $\mathcal{J}(\theta)$.

נרצה להשתמש באסטרטגיה סטוכסטית מכמה סיבות. ראשית, הרבה פעמים יותר קל למצוא אופטימום עבור בעיה רציפה מאשר בעיה דיסקרטית, ואסטרטגיה סטוכסטית יכולה להפוך בעיה דיסקרטית לרציפה. שנית, יהיו מקרים בהם נתעניין דווקא באסטרטגיה הסטוכסטית ולא בזו הדטרמיניסטית. בנוסף, הסטוכסטיות מאפשרת exploration, וכפי שראינו זה מאפיין הכרחי בשביל להגיע לכל המצבים. במבט יותר רחב, אסטרטגיה סטוכסטית היא הכללה של אסטרטגיה דטרמיניסטית – אם כל ההסתברויות של המודל הינן 0 או 1. במובן הזה, השימוש באסטרטגיה סטוכסטית לא ממעיט מהיכולת של אסטרטגיה דטרמיניסטית אלא הוא רק מכליל אותה.

באופן הכי פשוט, עבור בעיה בה מרחב בפעולות הינו דיסקרטי, ניתן להפוך אסטרטגיה שמספקת ערך לכל פעולה $\phi(a; x, \theta)$ להתפלגות על ידי SoftMax:

$$\pi_\theta(a|s) = \frac{\exp\big(\phi(a; x, \theta)\big)}{\sum_{a'} \exp\big(\phi(a'; x, \theta)\big)}$$

אם מרחב הפעולות הוא רציף, ניתן להשתמש בגאוסיאן:

$$\pi_\theta(a|s) \sim \mathcal{N}\big(\mu(x; \theta), \Sigma(x; \theta)\big)$$

ואם רוצים לאפשר יותר חופש פעולה, ניתן להשתמש ב-Gaussian mixture model. הפונקציה שבאמצעותה רוצים לייצג את האסטרטגיה יכולה להיות כרצוננו – פונקציה לינארית, רשת נוירונים וכו'.

רוב האלגוריתמים שעושים אופטימיזציה ל-$\mathcal{J}(\theta)$ מבוססי גרדיאנט, אך ישנם גם אלגוריתמים אחרים, כמו למשל אלגוריתמים גנטיים, אך עליהם לא נדבר. הדרך הסטנדרטית לבצע אופטימיזציה היא להשתמש ב-gradient descent/ascent ובפיתוחים שלו (כמו למשל stochastic gradient descent/ascent), והמוקד שלנו יהיה בשאלה

כיצד לשערך את הגרדיאנט ופחות נתעסק בשיטות האופטימיזציה השונות. האלגוריתם הבסיסי מבוסס על עדכון מהצורה הבאה:

$$\theta_{t+1} = \theta_t + \alpha \nabla f(\theta)$$

נניח ויש לנו episodic MDP, ומהאסטרטגיה הנתונה נוכל לדגום N מסלולים $\tau_1, \dots, \tau_N \sim \pi_\theta$, כאשר כל מסלול הינו $\tau_i = \left(S_0^i, A_o^i, R_1^i, S_1^i \dots, R_T^i, S_T^i\right)$, או בקצרה: $G_i = \sum_{t=0}^{T(i)} \gamma^t R_{t+1}^{(i)}$. שערוך מונטה-קרלו הרגיל ישערך את המטרה באופן הבא:

$$\mathcal{J}(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{t+1}\right] \approx \frac{1}{N}\sum_i G_i$$

הבעיה היא שהאפקט של הפרמטר θ על המסלול G_i היא לא ישירה – אנחנו דוגמים מסלול מאסטרטגיה π_θ התלויה בפרמטר θ, ואז מחשבים את התגמול של המסלול וממנו מרכיבים את G. לכן אי אפשר פשוט לגזור את G_i לפי θ, שהרי הקשר ביניהם הוא עקיף ותלוי ב-$f(\theta)$. בכדי להתמודד עם בעיה זו נשתמש ב-log-derivative trick (לעיתים התהליך הזה נקרא Reinforce). נגדיר:

$$g(\theta) = \mathbb{E}_{\pi_\theta}[f(x)]$$

ואז הנגזרת הינה:

$$\nabla_\theta g = \mathbb{E}_{\pi_\theta}\left[f(x)\nabla \log\left(\pi_\theta(x)\right)\right]$$

הוכחה:

$$\nabla_\theta g = \nabla_\theta \mathbb{E}_{\pi_\theta}[f(x)] = \nabla_\theta \int \pi_\theta(x) f(x) dx$$

נחליף את הסדר של הגרדיאנט והאינטגרל:

$$= \int \nabla_\theta \pi_\theta(x) f(x) dx$$

כעת נשתמש בקשר הבא (המתקיים עבור כל פונקציה): $\nabla_\theta \log\left(h(\theta)\right) = \frac{\nabla_\theta h(x)}{h(\theta)}$, ונקבל:

$$= \int \nabla_\theta \log\left(\pi_\theta(x)\right) \pi_\theta(x) f(x) dx$$

וזהו בדיוק שווה לתוחלת הבאה:

$$= \mathbb{E}_{\pi_\theta}\left[f(x)\nabla \log\left(\pi_\theta(x)\right)\right]$$

כעת, במקום לחשב את הגרדיאנט של G_i, נוכל לחשב את התוחלת $\mathbb{E}_{\pi_\theta}\left[f(x)\nabla \log\left(\pi_\theta(x)\right)\right]$ על ידי דגימות ושערוך מונטה קרלו, רק כעת הנגזרת אותה אנו צריכים לחשב **אינה תלויה** בפרמטר θ ולכן ניתן לחשב אותה בצורה ישירה. בצורה מפורשת, פונקציית המטרה שלנו הינה:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[G(\tau)], G(\tau) = \sum_{t=0}^{T(\tau)} \gamma^t R_{t+1}^{(\tau)}$$

ושימוש ב-log-derivative trick יביא אותנו לביטוי:

$$\nabla \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[G(\tau)\nabla \log\left(\pi_\theta(\tau)\right)\right]$$

נשים לב שהאסטרטגיה $\pi_\theta(\tau)$ מורכבת משני אלמנטים – הפעולה שאני בוחר לבצע, והמצב אליו אני מגיע יחד עם התגמול המתקבל מצעד זה. החלק הראשון הוא לגמרי בשליטה שלנו ואנו יודעים אותו, אך החלק השני הוא דינמי ותלוי בסטוכסטיות של המערכת, ולכן לכאורה הוא לא ידוע ולא יאפשר לבצע את הגזירה. אך אם נפתח את הביטוי שקיבלנו נראה שכל החלקים התלויים בדינמיות של המערכת אינם תלויים ב-θ ויתאפסו בגזירה. ניתן לרשום את האסטרטגיה כמכפלת ההסתברויות של כל הצעדים באופן הבא:

$$\pi_\theta(\tau_i) = P\left(S_0^i\right) \cdot \pi_\theta\left(A_0^i \middle| S_0^i\right) \cdot P\left(S_1^i, R_1^i \middle| S_0^i, A_0^i\right) \cdots \pi_\theta\left(A_{T^i-1}^i \middle| S_{T^i-1}^i\right) \cdot P\left(S_{T^i}^i, R_{T^i}^i \middle| S_{T^i-1}^i, A_{T^i-1}^i\right)$$

כעת אם נוציא לוג, המכפלה תהפוך לסכום:

$$\log\left(\pi_\theta(\tau_i)\right) = \log\left(P\left(S_0^i\right)\right) + \sum_{t=0}^{T^i-1} \log\left(\pi_\theta\left(A_t^i|S_t^i\right)\right) + \sum_{t=0}^{T^i-1} \log\left(P\left(R_{t+1}^i, S_{t+1}^i|S_t^i, A_t^i\right)\right)$$

הביטוי האמצעי מבטא את בחירות הצעדים של המשתמש, **והוא היחיד** שתלוי בפרמטר θ. שני הביטויים הנוספים מבטאים את הדינמיות של המערכת שאין לנו מידע כלפיהם, והם אינם תלויים ב-θ, לכן בנגזרת לפי θ הם מתאפסים. עובדה זה בעצם מספקת יתרון נוסף לשימוש בטריק של הלוג – מלבד האפשרות לחשב את הנגזרת באופן ישיר, הלוג גם מפריד בין הצעדים של המשתמש לבין הדינמיות הלא ידועה של המערכת. לכן בסך הכול נקבל:

$$\nabla_\theta \log\left(\pi_\theta(\tau_i)\right) = \sum_{t=0}^{T^i-1} \nabla_\theta \log\left(\pi_\theta\left(A_t^i|S_t^i\right)\right)$$

כעת נציב את זה בגרדיאנט של פונקציית המטרה ונשתמש בדגימות מונטה קרלו:

$$\nabla\mathcal{J}(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[G(\tau)\nabla\log\left(\pi_\theta(\tau)\right)\right] \approx \frac{1}{N}\sum_i G_i \nabla_\theta \log\left(\pi_\theta(\tau_i)\right)$$

$$= \frac{1}{N}\sum_i \left(\sum_{t=0}^{T^i-1} \gamma^t \mathcal{R}_{t+1}\right)\left(\sum_{t=0}^{T^i-1} \nabla_\theta \log\left(\pi_\theta\left(A_t^i|S_t^i\right)\right)\right)$$

חישוב הגרדיאנט בצורה זו נקרא בשם הפופולרי **Policy Gradient**. כעת משחישבנו את הגרדיאנט, נוכל להשתמש בשיטות אופטימיזציה סטנדרטיות על מנת למצוא את θ האופטימלי. בקצרה נוכל לסכם את כל התהליך ה-**Reinforce** בשלושה שלבים:

1. Run the policy and sample $\{\tau^i\}$ from $\pi_\theta(A_t|S_t)$.

2. Estimate gradients: $\nabla_\theta\mathcal{J}(\theta) \approx \frac{1}{N}\sum_i \left(\sum_{t=0}^{T^i-1}\gamma^t\mathcal{R}_{t+1}^i\right)\left(\sum_{t=0}^{T^i-1}\nabla_\theta\log\left(\pi_\theta\left(A_t^i|S_t^i\right)\right)\right)$.

3. Improve the policy using gradient descent/ascent: $\theta \leftarrow \theta_\alpha \nabla_\theta\mathcal{J}(\theta)$.

נתבונן על הביטוי של הנגזרת שקיבלנו וננסה לנתח אותו. נניח ויש מומחה שיודע לייצר trajectories טובים (-imitation learning) – במקרה כזה נרצה לקחת את מה שהוא מייצר ולגרום לכך שהאסטרטגיה שלנו תייצר בהסתברות יותר גבוהה דגימות כמו שקיבלנו מומחה. במילים אחרות, כאשר נתונים לנו מסלולים מוצלחים, נרצה לגרום לאסטרטגיה שלנו "לחקות" את אותם מסלולים. הדרך הסטנדרטית לבצע כזו משימה היא בעזרת maximum likelihood – לקחת את המסלולים האלה ולסכום אותם, ואז לחפש את המקסימום באמצעות גזירה של הביטוי המתקבל. באופן שקול ניתן גם לחפש מקסימום עבור הסכום של **לוג** המסלולים, ובסך הכל נרצה לחשב את הביטוי:

$$\sum_{t=0}^{T^i-1} \nabla_\theta \log\left(\pi_\theta(\tau_i)\right)$$

וזה בדיוק הביטוי השני בגרדיאנט שראינו קודם. בנוסף אליו יש גם את הגורם הראשון שממשקל כל מסלול בהתאם ל-reward, כך שמסלולים בעלי reward גבוה יקבלו משקל גבוה ואילו מסלולים בעלי reward נמוך יקבלו משקל נמוך. יוצא מכך, שהגרדיאנט מנסה להביא לכך שמסלולים "מוצלחים" יופיעו בהסתברות יותר גבוהה.

השערוך של הגרדיאנט באמצעות הביטוי שראינו הוא אמנם חסר הטיה, אך יש לו שונות גבוהה. בנוסף, אם כל הדגימות הם בעלי reward חיובי (או כולן בעלי reward שלילי), אז יהיה קשה להבחין איזה צעדים יותר טובים ומה לא כדאי לעשות. נראה שתי דרכים לשיפור היציבות של השימוש בגרדיאנט:

בסיבתיות (casualty) – כיוון שהצעד a_t שהתרחש בצעד t לא יכול להשפיע על כל הצעדים שקרו לפני כן $\{a_0, \dots, a_{t-1}\}$, ניתן להחליף את הביטוי $\sum_{t=0}^{T^i-1}\gamma^t\mathcal{R}_{t+1}$ בביטוי שמתחשב רק בצעדים שקרו החל מצעד t, כלומר: $\sum_{k=t}^{T^i-1}\gamma^k\mathcal{R}_{k+1}^i$ – ביטוי זה נקרא גם **Reward to go**. החלפה זו מורידה את השונות כיוון שהיא גורמת לכך שכל

צעד יהיה תלוי בפחות איבריים אקראיים. הביטוי שהתעלמנו ממנו הוא $\sum_{k=0}^{t-1}\gamma^k\mathcal{R}_{k+1}^i$, וניתן לראות שאכן אין לו השפעה על העתיד.

Baseline – ניתן להראות שהוספה של קבוע ל-reward משפיעה על התוצאה של הגרדיאנט. קבוע מסוים יכול לגרום לכך שנרצה לשפר צעדים מסוימים ולהימנע מצעדים אחרים, ואילו קבוע אחר יכול לגרום לכך שנרצה לשפר דווקא צעדים אחרים. קבוע זה הוא למעשה פרמטר נוסף שניתן לשלוט בו, ועל ידי בחירה מושכלת שלו ניתן להפחית את השונות. באופן פורמלי, את הביטוי $\sum_{t=0}^{T^i-1}\gamma^t\mathcal{R}_{t+1}^i$ נחליף בביטוי זהה עם תוספת קבוע:

$$\sum_{t=0}^{T^i-1}\gamma^t\mathcal{R}_{t+1}^i - b$$

השאלה היא כיצד לבחור את הקבוע b בצורה טובה? באופן פשוט ניתן לבחור את הקבוע בתור הממוצע של ה-rewards. בחירה כזו תביא לכך שמה שמתחת לממוצע נרצה להוריד את ההסתברות שלו ומה שמעל הממוצע נרצה לחזק אותו. בחירה כזו אכן מורידה את השונות, אך היא אינה אופטימלית. ניתן לחשב את הקבוע האופטימלי בצורה מדויקת. הנגזרת של פונקציית המטרה יחד עם האיבר הנוסף הינה:

$$\nabla\mathcal{J}(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[(G(\tau)-\mathrm{b})\nabla\log\left(\pi_\theta(\tau)\right)\right]$$

הממוצע אינו תלוי ב-b, לכן נוכל לרשום את השונות כך:

$$Var = \mathbb{E}_{\tau\sim\pi_\theta}\left[(G(\tau)-\mathrm{b})^2\nabla\log\left(\pi_\theta(\tau)\right)^2\right] + C$$

$$= \mathbb{E}_{\tau\sim\pi_\theta}\left[G(\tau)^2\nabla\log\left(\pi_\theta(\tau)\right)^2\right] + \mathbb{E}_{\tau\sim\pi_\theta}\left[b^2\nabla\log\left(\pi_\theta(\tau)\right)^2\right] - 2\mathbb{E}_{\tau\sim\pi_\theta}\left[G(\tau)\cdot\mathrm{b}\nabla\log\left(\pi_\theta(\tau)\right)^2\right] + C$$

כעת נגזור את השונות לפי b:

$$\frac{dVar}{db} = 2\mathbb{E}_{\tau\sim\pi_\theta}\left[\mathrm{b}\nabla\log\left(\pi_\theta(\tau)\right)^2\right] - 2\mathbb{E}_{\tau\sim\pi_\theta}\left[G(\tau)\cdot\mathrm{b}\nabla\log\left(\pi_\theta(\tau)\right)^2\right] = 0$$

$$\rightarrow b_{opt} = \frac{\mathbb{E}_{\tau\sim\pi_\theta}\left[G(\tau)\nabla\log\left(\pi_\theta(\tau)\right)^2\right]}{\mathbb{E}_{\tau\sim\pi_\theta}\left[\mathrm{b}\nabla\log\left(\pi_\theta(\tau)\right)^2\right]}$$

הביטוי המתקבל הוא אכן הממוצע של ה-reward, אך מוכפל במשקל התלוי בנגזרת של לוג האסטרטגיה. השערוך של הביטוי המדויק יכול להיות מסובך ורועש, ולכן לרוב כן משתמשים פשוט ב-reward הממוצע.

אלגוריתם policy gradient הינו on-policy, כלומר האסטרטגיה שבעזרתה אנו דוגמים מסלולים הינה אותה אסטרטגיה שאנו מנסים להביא לאופטימום. כאמור לעיל, באלגוריתמים שהם on-policy, אי אפשר ל"מחזר" דגימות, כלומר בכל נקודת זמן ניתן להשתמש רק בדגימות שנלקחו מהאסטרטגיה הנתונה באותה נקודה. עבור מצבים בהם הדגימות יקרות, נרצה להשתמש באותן דגימות עבור נקודות זמן שונות, כלומר להפוך את האלגוריתם להיות off-policy. בכדי לעשות זאת יש לתקן את השגיאה שנוצרת מכך שבנקודת זמן מסוימת אנו משתמשים בדגימות שנלקחו מאסטרטגיה שכבר השתנתה וכעת ההתפלגות של האסטרטגיה שונה. במילים אחרות – יש לנו דגימות שנלקחו מתוחלת מסוימת, ובעזרתן אנו רוצים לשערך תוחלת אחרת. זו בעיה מתחום הסטטיסטיקה, והפתרון שלה נקרא **Importance sampling**:

נניח ואנו רוצים לשערך את $\mathbb{E}_p[f(x)]$, אך אנו לא יכולים לדגום מהתפלגות p אלא רק מהתפלגות q, כאשר ההנחה היחידה שלנו היא שאם $p>0$ אז גם $q>0$. באמצעות מעבר מתמטי די פשוט ניתן לייצג את התוחלת של p באמצעות התוחלת של q:

$$\mathbb{E}_p[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = \mathbb{E}_q\left[f(x)\frac{p(x)}{q(x)}\right]$$

כעת נדגום $x_1,\dots,x_N\sim q$ ונשערך את התוחלת באמצעות מוטנה קרלו:

$$\mathbb{E}_p[f(x)] = \mathbb{E}_q\left[f(x)\frac{p(x)}{q(x)}\right] \approx \frac{1}{N}\sum_i f(x_i)\frac{p(x_i)}{q(x_i)}$$

אם ההתפלגויות $p(x), q(x)$ הן יחסית דומות, אז התוצאה המתקבלת משערכת בצורה יחסית טובה את התוחלת הרצויה. אם זה לא המצב, השונות תהיה גדולה והתוצאה תהיה לא יציבה. מכל מקום, נוכל להשתמש ברעיון זה עבור שערוך הגרדיאנט:

- For trajectory probability the dynamics does not depend on θ so
$$\frac{\pi_{\theta'}}{\pi_\theta} = \frac{\prod \pi_{\theta'}(a_t \mid s_t)}{\prod \pi_\theta(a_t \mid s_t)}$$

- Can now Compute $\nabla_{\theta'} J(\theta')$ with samples from π_θ

- $$\sum_{i=1}^{N} \left(\sum_{t=0}^{T^{(i)}-1} \gamma^t r_{t+1}^{(i)} \right) \left(\sum_{t=0}^{T^{(i)}-1} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) \right) \frac{\prod \pi_{\theta'}(a_t \mid s_t)}{\prod \pi_\theta(a_t \mid s_t)}$$

- Can improve with causality

- $$\sum_{i=1}^{N} \sum_{t=0}^{T^{(i)}-1} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) \frac{\prod_{k=0}^{t} \pi_{\theta'}(a_k \mid s_k)}{\prod_{k=0}^{t} \pi_\theta(a_k \mid s_k)} \left(\sum_{k=t}^{T^{(i)}-1} \gamma^t r_{k+1}^{(i)} \right)$$

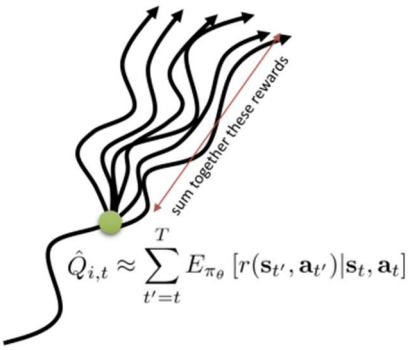
כאמור, שימוש ברעיון זה יכול להיות טוב אם ההתפלגויות θ, θ' יחסית דומות. אם הן רחוקות, התוצאה המתקבלת לא בהכרח מהווה שערוך טוב, ולכן הלמידה לא תהיה מספיק טובה. לסיכום – אפשר להפוך את האלגוריתם של policy gradient להיות off-policy באמצעות importance sampling, אך גם שיטה יכולה לסבול משונות גבוהה.

11.3.5 Actor Critic

בפרקים הקודמים דיברנו על שתי גישות בכדי לשערך את האסטרטגיה בלי לדעת המודל. גישה אחת התמקדה בשערוך ה-Action-Value function (SARSA/Q-Learning-), בעוד הגישה השנייה ניסתה לשערך את האסטרטגיה באופן ישיר (Policy gradient). בפרק זה נציג אלגוריתם המנסה לשלב את שתי הגישות יחד. נתבונן שוב בביטוי של הגרדיאנט, כאשר בהתייחסות ל-rewards נתייחס רק ל-reward to go, כפי שהוסבר לעיל:

$$\frac{1}{N} \sum_i \left(\sum_{k=t}^{T^i-1} \gamma^k \mathcal{R}_{k+1}^i \right) \left(\sum_{t=0}^{T^i-1} \nabla_\theta \log \left(\pi_\theta \left(A_t^i \middle| S_t^i \right) \right) \right)$$

ניתן להבחין כי הביטוי $\sum_{k=t}^{T^i-1} \gamma^k \mathcal{R}_{k+1}^i$ הוא למעשה דגימה של $Q^{\pi_0}(s_t, a_t)$, כלומר עבור נקודת זמן, ניתן לדגום מסלול באמצעות ה-Action-Value function ולקבל סכום של rewards. החיסרון בדגימה זו נעוץ בכך שיש לה שונות גבוהה והיא יכולה להיות מאוד רועשת. כפי שניתן לראות באיור, יתכנו הרבה מסלולים שונים היוצאים מאותה נקודה, ולכן הביטוי של ה-reward to go הוא מאוד רועש ביחד ל-$Q^{\pi_0}(s_t, a_t)$:



איור 11.9 מסלולים שונים אפשריים היוצאים מאותה נקודה. המיצוע/התוחלת של הביטוי אמנם מוריד את השונות ל-0, אך כל דגימה בפני עצמה היא בעלת שונות גבוהה.

הבחנה זו של הקשר בין ה-reward to go לבין ה-$Q^{\pi_0}(s_t, a_t)$ מעלה את השאלה האם אפשר להחליף את בין הביטוי הרועש $\sum_{k=t}^{T^i-1} \gamma^k \mathcal{R}_{k+1}^i$ לבין הפונקציה עצמה $Q^{\pi_0}(s_t, a_t)$. בשביל לבצע החלפה זו, יש להוכיח שהיא אכן חוקית ושומרת על תוצאה נכונה, וכדי לעשות זאת נפתח את פונקציית המטרה כך שתופיע בצורה יותר נוחה. כזכור, פונקציית המטרה אותה אנו מעוניינים להביא לאופטימום הינה התוחלת של ה-rewards (מוכפלים ב- discount factor):

$$\mathcal{J}(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{t+1}\right]$$

המשמעות של התוחלת היא לבצע אינטגרל על כל צמד של action-state – (s, a), כפול המשקל של אותו צמד. כיוון שצמד יכול לחזור על עצמו בצעדים שונים (כלומר בשתי נקודות זמן t_1, t_2 המערכת יכולה להיות באותו מצב ולבצע את אותו פעולה), אז למעשה באינטגרל ישנם ביטויים שחוזרים על עצמם. בכדי להימנע מכך, ניתן לחשב כל ביטוי באינטגרל פעם אחת, ולהכפיל אותו במשקל המתאים למספר הביקורים באותו צמד action-state. עם זאת, עדיין צריך לזכור שהמופעים של אותו צמד נבדלים זה מזה ב-discount factor, כיוון שהמשקל של reward בזמן t_1 שונה מהמשקל שלו בזמן t_2, ויש לקחת את ההבדל הזה בחשבון. כעת נתבונן על הביטוי המפורש של האינטגרל המתקבל, כאשר ראשית נחליף את סדר התוחלת והסכימה:

$$\mathcal{J}(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{t+1}\right] = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{\pi_\theta}[\mathcal{R}_{t+1}]$$

התוחלת של כל ה-rewards מורכבת ממיצוע שלהם על פני: **1)** ההסתברות להתחיל ממצב s_0 <u>כפול</u> **2)** ההסתברות להגיע למצב s בזמן t בהנתן שהתחלנו מ-s_0 <u>כפול</u> **3)** הסיכוי לבצע פעולה a במצב s שהתקבל בזמן t. באופן פורמלי הביטוי המתקבל הינו

$$= \sum_{t=0}^{\infty} \gamma^t \int_{\mathcal{S}} \int_{\mathcal{S}} \int_{\mathcal{A}} p(s_0)\, p_t^{\pi}(s|s_0)\, \pi(a|s)\, \mathcal{R}(s,a)\, ds_0\, ds\, da$$

$$= \int_{\mathcal{S}} \left(\sum_{t=0}^{\infty} \gamma^t \int_{\mathcal{S}} p(s_0) p_t^{\pi}(s|s_0) ds_0 \right) \int_{\mathcal{A}} \pi(a|s)\ \mathcal{R}(s,a)\, ds\, da$$

נסמן את הביטוי שבסוגריים כ-$\rho^{\pi}(s)$, והמשמעות שלו היא מה הסיכוי להגיע ב-t צעדים ממצב s_0 למצב s, כאשר עבור כל t יש להכפיל ב-discount factor המתאים. ביטוי זה נקרא Discount state visitation measure, ולמעשה הוא כולל בתוכו את כל המופעים של צמד (s, a), ובכך במקום לעשות אינטגרל על אותו צמד (s, a) מספר פעמים, עושים זאת פעם אחת עבור כל נקודות הזמן השונות ומכפילים ב-discount factor המתאים. באופן מפורש:

$$= \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \pi(a|s)\ \mathcal{R}(s,a)\, ds\, da,$$

$$\rho^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t \int_S p(s_0) p_t^{\pi}(s|s_0) ds_0$$

יש לשים לב שהביטוי הכולל הוא לא בצורה הפשוטה של תוחלת – אינטגרל של הסתברות כפול פונקציית צפיפות, אך הוא עדיין משקף תוחלת ולכן ניתן לשערך אותו באמצעות מונטה-קרלו. כעת נתבונן על הגרדיאנט של פונקציית המטרה. אנחנו רוצים לגזור את הביטוי הבא לפי θ:

$$\mathcal{J}(\theta) = \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \pi(a|s)\ \mathcal{R}(s,a)\ ds\ da$$

כל האיברים תלויים ב-θ אך ניתן להראות (שקופיות 10-11 במצגת 7) שהנגזרת תלויה רק ב-$\pi(a|s)$, כלומר:

$$\nabla_{\theta}\mathcal{J} = \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \nabla_{\theta}\big(\pi(a|s)\big)\, Q^{\pi_\theta}(s,a)\ ds\ da$$

כעת ניתן להיעזר ב-log-derivative trick ולקבל:

$$\nabla_{\theta}\mathcal{J} = \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \pi(a|s) \nabla_{\pi} \log\big(\pi(a|s)\big)\, Q^{\pi_\theta}(s,a)\ ds\ da$$

ואז לשערך את הביטוי באמצעות דגימות מונטה-קרלו:

$$\nabla_{\theta}\mathcal{J} \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T^{(i)}-1} \nabla_{\theta} \log \pi_{\theta}\left(a_t^{(i)} \middle| s_t^{(i)}\right) Q^{\pi_\theta}\left(s_t^{(i)}, a_t^{(i)}\right)$$

הביטוי המתקבל טוב מאוד מבחינת זה שהוא מצליח להפחית את השונות, אך הבעיה בו נעוצה בכך שהאיבר $Q^{\pi_\theta}\left(s_t^{(i)}, a_t^{(i)}\right)$ אינו ידוע! אלגוריתם **Actor Critic** בא להתמודד עם בעיה זו, והרעיון הוא לנסות לשערך את Q^{π_θ} באמצעות Q^{w} במקביל לשערוך של הגרדיאנט. כלומר, בתהליך הלמידה אנו מנסים לשערך שני דברים במקביל:

π_θ (Actor) – הגורם המחליט איזה צעד לנקוט בכל מצב.

Q^w (Critic) – הגורם המבצע אבלואציה של Q על מנת לשפר את הבחירה של ה-Actor.

הערת אגב: לעיל ראינו שהשימוש ב-$Q-$Learning יכול להיות בעייתי בבעיות רציפות בגלל שמרחב המצבים הוא אינסופי ומאוד קשה לפתור את בעיית האופטימזציה של בחירת action בעזרת הביטוי $\arg\max_{A} Q_{\theta}(s,a)$. כאן בעיה זאת איננה, כיוון שאנחנו לא נדרשים לבחור את ה-action באמצעות Q אלא רק לשפר באמצעותו את האסטרטגיה (או במילים אחרות – אנחנו לא צריכים לפתור בעיית אופטימזציה על Q עבור מרחב מצבים רציף).

האלגוריתם פועל בהתאם לשלבים הבאים. ראשית נאתחל את θ, w. כעת נבצע T איטרציות באופן הבא:

- נדגום מצב התחלתי s_0 מתוך מרחב המצבים $\mathcal{S}$ ופעולה מתוך האסטרטגיה $a_0|s_0 \sim \pi_\theta(\cdot\ |s_0)$.
- כעת כל עוד לא הגענו למצב הסופי:
  - נחשב את המצב החדש $new\ s$ ואת התגמול המתקבל r.
  - בנוסף, נדגום פעולה חדשה $new\ a \sim \pi_\theta(\cdot\ |new\ s)$.
  - נחשב את ה-TD error:

$$\delta = r + \gamma Q^w(new\ s, new\ a) - Q^w(s,a)$$

  - נעדכן את הפרמטרים:

$$\theta \leftarrow \theta + \alpha Q^w(s,a) \nabla_{\theta} \log \pi_{\theta}(a|s)$$
$$w \leftarrow w + \beta \delta \nabla_w Q^w(s,a)$$

  - נעדכן את המצב והפעולה:

$$a = new\ a, s = new\ s$$

פסאודו קוד עבור האלגוריתם:

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \dots T$: **do**
  Sample reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s',a') - Q_w(s,a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s,a)$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

איור 11.10 אלגוריתם Actor-Critic. שערוך הגרדיאנט באמצעות Q^w במקום הביטוי של ה-reward-to-go Policy gradient.

חשוב להדגיש שהעדכון של הגרדיאנט מתבצע בדומה ל-SARSA ולא כמו $Q-$Learning. כאמור לעיל, ההבדל ביניהם הוא באופי האלגוריתם – SARSA הוא אלגוריתם on-policy, כלומר הוא מנסה לאפטם את האסטרטגיה ממנו הוא דוגם את המסלולים, בעוד ש-$Q-Learning$ מנסה ללמוד את האסטרטגיה האופטימלית בלי תלות בדוגמאות אותן הוא רואה. אלגוריתם Actor Critic רוצה לשפר את ה-Policy gradient, כלומר המטרה היא לשפר את האסטרטגיה הנתונה, ולכן יש לבצע למידה שהיא On policy.

בדומה ל-Policy gradient, גם ל-Actor Critic ניתן להוסיף Baseline על מנת לשפר את תהליך הלמידה. בניגוד ל-Policy gradient הוא התוספת היתה של קבוע, כאן נהוג להוסיף את $\mathcal{V}^\pi(s)$, שהוא תלוי ב-s. אם ה-baseline תלוי רק במצב s, אז זה בסדר כפי שנוכיח מיד (שקף 16), אבל אם הוא תלוי גם ב-action a, אז זה מוסיף bias והלמידה תפגע. הביטוי: $A^\pi(s,a) = Q^\pi(s,a) - \mathcal{V}^\pi(s)$ נקרא ה-advantage function, והסיבה לבחירה שלו כ-baseline היא פשוטה – אם הערך של $A^\pi(s,a)$ חיובי, אז זה סימן לכך שהבחירה בצעד a היא טובה, ולכן נרצה לחזק אותה. אם לעומת זאת הערך שלילי, זה אומר שהפעולה לא טובה וממילא נרצה להימנע ממנה. במילים אחרות: $\mathcal{V}^\pi(s)$ הינו הערך הממוצע של המצב s, והביטוי $Q^\pi(s,a)$ הינו הערך של פעולה ספציפית עבור מצב s, ולכן $A^\pi(s,a)$ אומר לנו עד כמה הפעולה a טובה או גרועה ביחס לממוצע של המצב בו אנו נמצאים, וממילא נוכל לדעת האם נרצה לחזק את הפעולה הזו או לא.

לאחר שראינו שניתן להשתמש ב-Baseline עבור Actor Critic ויש היגיון להשתמש ב-$\mathcal{V}^\pi(s)$ בתור ה-baseline, נראה כיצד ניתן ליישם זאת בפועל. אם מרחב המצבים הוא דיסקרטי, ניתן ללמוד את $Q^w(s)$ בשיטות הסטנדרטיות שראינו, ואז לחשב באופן מדויק את $\mathcal{V}^\pi(s)$ בעזרת הקשר $\mathcal{V}^\pi(s) = \sum_a \pi(a|s) Q^w(s,a)$. במצבים מסוימים ניתן להשתמש באותו רעיון גם עבור מרחב מצבים רציף – נניח ו-$Q^w(s)$ הוא לינארי ביחס למרחב הפעולות, כלומר מתקיים: $Q^w = \phi_w(s)^T \cdot a$, והאסטרטגיה היא גאוסיאנית: $\pi_\theta(a|s) \sim \mathcal{N}\big(\mu(s), \sigma^2(s)\big)$, אז $\mathcal{V}^\pi(s)$ הוא התוחלת של $Q^w(s)$ על פני כל הפעולות: $\mathcal{V}^w(s) = \mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[Q^w(s)]$. בעזרת הנוסחה הסגורה הזו ניתן להשתמש בשערוך $Q^w(s)$ ולחשב את $\mathcal{V}^w(s)$ ואז להשתמש בו כ-Baseline.

כאשר לא ניתן להשתמש ב-$Q^w(s)$ על מנת לחשב את $\mathcal{V}^w(s)$, ניתן לנסות לשערך את שני הביטויים בנפרד, אך זה יכול להיות בזבזני. לחילופין, ניתן לבנות מודל (למשל רשת נוירונים) שתנסה לשערך במקביל את שני הביטויים, כלומר הפלט שלה יהיה גם $Q^w(s)$ וגם $\mathcal{V}^w(s)$. גישה אחרת מציעה להשתמש במשוואות בלמן. לפי משוואות אלו מתקיים בקירוב $Q(s_t,a_t) = r_{t+1} + \gamma \mathcal{V}(s_t)$, ולפי זה ה-advantage function יהיה:

$$A(s_t,a_t) = r_{t+1} + \gamma \mathcal{V}(s_{t+1}) - \mathcal{V}(s_t)$$

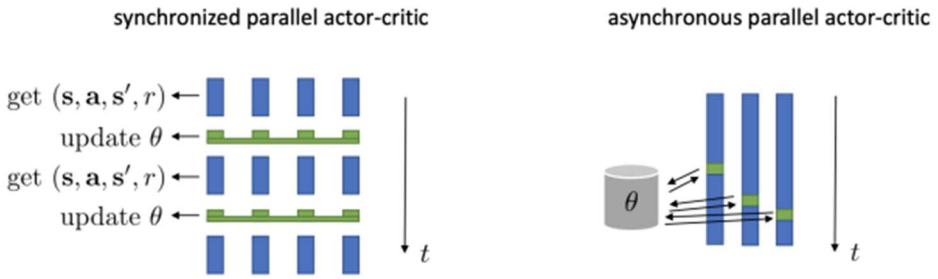
את הביטוי הזה נכניס לגרדיאנט ובכך נשערך את האסטרטגיה. נשים לב שביטוי זה תלוי במצב בלבד ולא בפעולה, מה שמקל על הלמידה, כיוון שיותר קל ללמוד את $\mathcal{V}(s)$ השייך למרחב בגודל $\mathcal{S}$ מאשר את $Q(s,a)$ השייך למרחב בגודל $\mathcal{S} \times \mathcal{A}$. המשמעות הפרקטית של הביטוי היא שמסתכלים על שני מצבים, ובוחנים את הפרש הערכים שלהם ועוד ה-reward. אם הערך חיובי – אז נרצה לחזק את המעבר מהמצב הראשון למצב השני, ואם הערך שלילי, אז נרצה למנוע כזה מעבר.

חסר – שקופית 19.

אחת הבעיות שיש גם ב-$Q-$Learning וגם ב-Actor Critic היא הקורלציה שיש בין הדגימות – עבור מסלול שנדגם, יש קשר בין המצבים השונים, ולכן השונות עדיין גבוהה. לעיל ראינו שבאלגוריתם $Q-$Learning ניתן להתגבר על

כך בעזרת שמירה של N מצבים ועדכון לפי k מצבים אקראיים מתוך אלה שנשמרו (למשל: $N = 1000, k = 10$). כאן לא ניתן להשתמש בפתרון זה, שהרי הלמידה היא on-policy ואי אפשר להשתמש במצבים מדגימות קודמות, כיוון שהן נדגמו מאסטרטגיה שכרגע היא לא רלוונטית. ניתן בכל אופן להשתמש במצבים שנדגמו מה-epochs האחרונים, תחת הנחה שהאסטרטגיה לא השתנה הרבה (ובנוסף ניתן לתקן באמצעות Importance sampling).

פתרון יותר מוצלח הוא לאמן מספר סוכנים במקביל ולהשתמש במידע שמגיע מכולם, כאשר היתרון הגדול של שיטה זו הוא שהדגימות של הסוכנים השונים הן חסרות קורלציה. שיטה זו הציגה תוצאות מעולות ובזמן הרבה יותר מהיר מהשיטות הקודמות שראינו. יש שתי אפשרויות להריץ במקביל – ריצה סינכרונית (הסוכנים רצים במקביל, ולאחר שכולם מסיימים epoch מחשבים את הגרדיאנט ומעדכנים את האסטרטגיה) וריצה אסינכרונית (כל פעם שסוכן מסיים epoch הוא מחשב גרדיאנט ומעדכן האסטרטגיה). הריצה הסינכרונית יכולה להיות קצת יותר איטית כיוון שבכל עדכון יש לחכות שכל הסוכנים יסיימו. מצד שני הריצה האסינכרונית יכולה להיות פחות מדויקת כיוון שיתכנו מצבים שלסוכנים שונים תהיה אסטרטגיה מעט שונה (אם הם עדכנו את האסטרטגיה שלהם בזמנים שונים ובאמצע היה עדכון של הגרדיאנט). עם זאת ניתן להניח שההבדלים יחסית לא גדולים, כיוון שהסוכנים מעדכנים את האסטרטגיה שלהם בערך באותן נקודות זמן.



איור 11.11 אימון מספר סוכנים במקביל בשיטת Actor-Critic. ניתן לבצע את העדכון בצורה סינכרונית – כאשר כל הסוכנים מסיימים epoch מחשבים את הגרדיאנט ומעדכנים את האסטרטגיה, או בצורה אסינכרונית – כל סוכן שמסיים epoch מעדכן את האסטרטגיה בהתאם.

לסיכום, שיטת Actor-Critic מנסה לשלב עקרונות מתוך $Q-$ Learning יחד עם Policy gradient. הרעיון הבסיסי הוא להחליף את הביטוי של ה-reward-to-go Policy gradient בשערוך של Q, שהוא הרבה פחות רועש. את Q ניתן לשערך במקביל לגרדיאנט עצמו, אך יש לעשות זאת on-policy. אמנם השונות יורדת אך עדיין יש הטיה, כיוון שיש צורך לשערך גם את Q. בדומה ל-policy gradient, גם כאן ניתן להוסיף Baseline, וראינו שמקובל להוסיף את $\mathcal{V}$.

11.4 Model Based Control

בפרק הקודם דיברנו על אלגוריתמים שלא מתעסקים במודל של הבעיה, אלא מנסים ללמוד ישירות מתוך הניסיון את הערך של כל מצב/מצב-פעולה ($Q(s,a)/\mathcal{V}(s)$) ו/או את האסטרטגיה האופטימלית עבור הסוכן. בפרק זה נדבר על אלגוריתמים שמנסים ללמוד את האסטרטגיה מתוך המודל או הדינמיקה של הבעיה (בין אם הם ידועים ובין אם הם נלמדת).

יש מספר יתרונות ללמידה מתוך המודל ולא מתוך ניסיון בלבד. ראשית, הרבה פעמים הלמידה של מודל יכולה להיות משמעותית פשוטה יותר מאשר למידה של $\mathcal{V}(s)/Q(s,a)$, מה שהופך את הלמידה להרבה יותר חסכונית. ניקח לדוגמה מבוך פשוט – הבנה של הדינמיקה שלו היא הרבה יותר פשוטה מאשר למידה של הערך של כל משבצת. בנוסף, למידה של המודל מאפשרת Self-exploration – נניח ואנו רוצים לאמן סוכן שמבצע נהיגה אוטונומית, אם הוא לא ידע את הדינמיקה של הסביבה אלא ינסה ללמוד רק על בסיס הניסיון, הוא יהיה חייב לבצע פעולות שליליות (-למשל לדרוס אנשים ולעשות תאונות) על מנת לקבל משוב שלילי וכך ללמוד שיש להימנע מפעולות אלה. אם לעומת זאת אנו יודעים את המודל והסביבה, ניתן לייצר סימולציה ובה לבה לבחון את המצבים השונים וכך לאמן את הסוכן.

עם זאת, ישנם גם חסרונות בלמידה של המודל. ראשית, הרבה פעמים יש פערים בין המודל לבין העולם האמיתי. נניח ואימנו סוכן של רכב אוטונומי בסביבה וירטואלית, המעבר לעולם האמיתי אינו טריוויאלי כי הסימולציה פשוט לא מספיק דומה לעולם האמיתי (למשל – רזולוציית תמונת הקלט של הסוכן בסימולציה הרבה יותר גבוהה מזו של מצלמה אמיתית המותקנת על רכב). בנוסף, למידה של מודל מוסיפה מקור נוסף של שגיאה – נניח ומנסים לשערך מודל ועל בסיס המודל רוצים ללמוד אסטרטגיה, אז יש פה שני שלבים של למידה וכל אחד מהם יכול לגרום לשגיאה.

11.4.1 Known Model – Dyna algorithm

נתחיל מהמקרה היותר פשוט, בו המודל ידוע ואנו מנסים באמצעותו ללמוד אסטרטגיה אופטימלית. בפרק המבוא ראינו כיצד ניתן להשתמש ב-Policy iteration עבור Tabular MDP בעל מרחב מצבים לא גדול על מנת ללמוד את האסטרטגיה האופטימלית, ובפרק זה נרצה להתמקד במקרים בהם פתרון זה לא מספיק טוב.

הרעיון הפשוט ביותר בו ניתן להשתמש הוא לייצר סימולציה, כלומר לקחת את המודל הידוע, ולאמן בעזרתה סוכן כאילו שהוא נמצא בעולם האמיתי. למידה של האסטרטגיה או של $\mathcal{V}(s)/\mathcal{Q}(s,a)$ באופן הזה לרוב תהיה זולה משמעותית מאשר למידה שלהם ללא עזרת המודל, כיוון שנצטרך הרבה פחות איטרציות והשימוש במודל הידוע מסייע להכוון את הלמידה. עם זאת, הבעיה היא שיש פער בין הסימולציה לבין העולם האמיתי, מה שיוצר אתגר כפול: א. הסימולציה לא מגיעה לרמת דיוק מספיק טובה בתיאור העולם האמיתי ולכן הלמידה לא מספיק איכותית. ב. נניח ויש לנו אפשרות ללמוד על העולם האמיתי, אז גם אם יש שגיאה – היא לרוב תהיה קטנה מאוד. בסימולציה לעומת זאת, שגיאה כזו יכולה להצטבר ולגרום לסוכן לשגיאות גדולות.

בכדי להתגבר על אתגרים אלו ניתן **לשלב** בין Model free ו-Model Based, וכעת נציג אלגוריתם בשם **Dyna** שעושה בדיוק את השילוב הזה. אלגוריתם זה משלב ידע משני מקורות – דאטה אמיתי ודאטה המגיע מסימולציה, כאשר הוא פועל באופן איטרטיבי לפי השלבים הבאים:

- ראשית מקבלים דאטה אמיתי, ומשפרים באמצעותו **שני** דברים – מודל שעליו מושתת הסימולציה וסוכן שמאומן בכלים של model free.
- לאחר מכן מייצרים דאטה מהסימולציה, ומשתמשים בו בשביל לשפר את הסוכן.

מבצעים את השלבים האלה שוב ושוב וככה משפרים במקביל גם את הסימולציה וגם את הסוכן (שלומד על בסיס דאטה מהסימולציה). באופן הזה אנחנו משפרים את הסימולציה כל הזמן, ואז גם אם יש בעיה בסימולציה והיא לא זהה בדיוק למודל של העולם האמיתי, אנחנו לא מתעלמים ממנה אלא מנסים לתקן אותה. בניגוד לסימולציה פשוטה שלא משתמשת בדאטה מהעולם האמיתי אלא רק מתבססת על המודל הידוע, כאן אנו לא זונחים את הדאטה האמיתי אלא עושים לו אוגמנטציות בעזרת הסימולציה (שבעצמה גם הולכת ומשתפרת). נניח מקרה פשוט של מודל דטרמיניסטי ובחירה של אלגוריתם $Q-$ Learning לאימון הסוכן, האלגוריתם יפעל לפי לשלבים הבאים (כמובן שניתן להרחיב בצורה טריוויאלית לכל מודל ולכל אלגוריתם Model-Free):

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon$-greedy(S,Q)
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

איור 11.12 אלגוריתם Dyna. שילוב של למידה מסימולציה המבוססת על מודל ידוע יחד עם סוכן שלומד בעזרת אלגוריתם Model-Free.

מלבד העובדה שתוך כדי הלמידה אנו משפרים את המודל, גם הדאטה שמיוצר הוא יותר רלוונטי ובעל ערך ללמידה – לא סתם מוגרלים מצבים אקראיים, אלא המצבים שהסוכן רואה קשורים לאסטרטגיה שנלמדה על בסיס הדאטה מהעולם האמיתי. באופן הזה יש שימוש כפול ו"ממצה" יותר של הדאטה מהעולם האמיתי – הוא גם מסייע לבנות את המודל וגם מסייע לסוכן להשתפר.

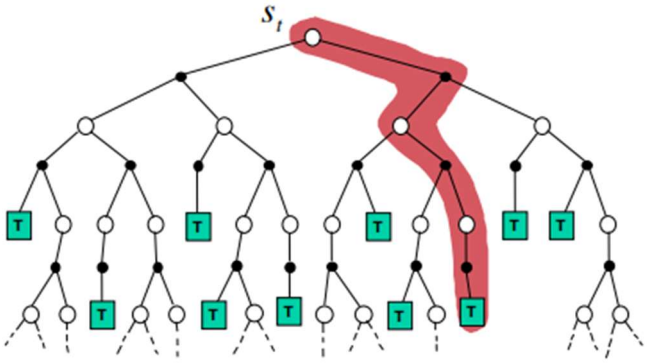
גרף ודוגמה – שקופיות 11,12. שינוי מודל למודל לא נכון – שקופית 13 (מודל קבוע לא היה מצליח להשתפר לאחר השינוי).

11.4.2 Known Model – Tree Search

כאמור, אלגוריתם Dyna נעזר בידיעת המודל על מנת לאמן סוכן בעזרת אלגוריתם של Model-Free. יוצא מכך שאמנם הסוכן אינו תלוי במודל, אך עדיין האסטרטגיה אותה אנו מנסים ללמוד היא **גלובלית**, כלומר מנסה לדעת מה לעשות בכל סיטואציה אפשרית. עבור משימות מאוד מורכבות, כמו למשל משחק שחמט, זה עדיין לא מספיק, כיוון שאסטרטגיה כללית עבור כל המצבים אינה אפשרית ללמידה עקב ריבוי המצבים. בכדי להתמודד עם בעיות מורכבות

ניתן להפוך את הפתרון להיות **לוקאלי**, כלומר להסתכל על המצב הנוכחי בלבד ולנסות ללמוד בעזרת הסימולציות מה הפעולה שתביא את התוצאה הכי טובה.

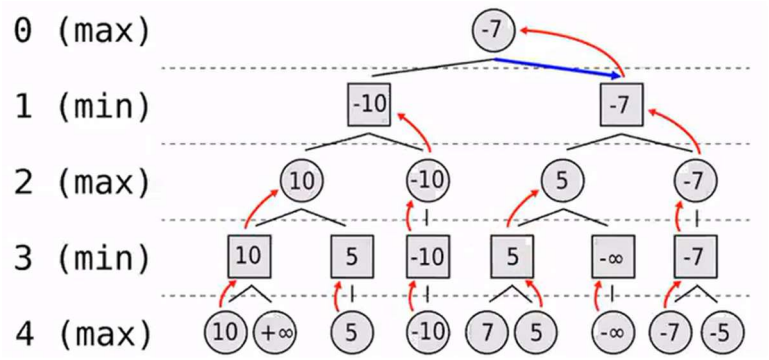
הרעיון של הלמידה הלוקאלית – חיפוש הפעולה הכי טובה עבור מצב נתון – קשור לתחום רחב שנקרא Forward Search (נעיר שבעיה זו קשורה להרבה תחומים ולא רק ל-RL, ופה נדון בעיקר במה שקשור לחיפוש באמצעות אלגוריתם של RL). תחום זה בא להתמודד עם מצבים בהם הבעיה הכללית היא מאוד מורכבת, אך אנו מנסים לפתור תת-בעיה המסתכלת רק על המצב הנוכחי ומה שמסתעף ממנו. ניתן לדמות את הסביבה לעץ בו כל צומת הוא מצב והוא מתפצל למצבים נוספים לפי הפעולות השונות האפשריות באותו מצב. בעזרת הסימולציה נבדוק כל מיני הסתעפויות שונות מאותו מצב, ונוכל לבחון מי מבניהן הטובה ביותר.



איור 11.13 Forward Search. חיפוש הפעולה הכי טובה ביחס למצב נתון. כל צומת בעץ מייצג מצב, וההסתעפויות מתפתחות בהתאם לפעולות האפשריות בכל מצב, כאשר terminal state מיוצג על ידי T. סימולטור המבוסס על מודל ידוע יכול לעזור לבחון את ההסתעפויות השונות.

כאשר כחלק מהבעיה יש גם יריב שהאסטרטגיה שלו אינה ידועה, יש כמה דרכים איך להתייחס להתנהגות שלו, כאשר הנפוצות שבהן הן לייחס ליריב את אותה אסטרטגיה כמו של הסוכן (Self-play) או לייחס אליו את האסטרטגיה האופטימלית (Minimax), כאשר יש להגדיר במדויק מהי אותה אסטרטגיה אופטימלית. אם נניח שמדובר במצבים של משחק סכום-אפס (בהם התגמול של שני המשתתפים הוא קבוע), אסטרטגיה אופטימלית מניחה שהיריב ינקוט בפעולות שימקסמו את התגמול שלו וימזערו עד כמה שניתן את התגמול של הסוכן. במקרה זה, הסוכן ירצה לבחור בפעולה שתמקסם את התגמול שלו ב-worst case, כלומר במקרה בו היריב בחר את הפעולות הטובות ביותר עבורו.

הבעיה שעולה מיד היא שבפועל אי אפשר להתחיל ממצב מסוים ולבדוק את כל ההסתעפויות עבור כל עומק אפשרי של העץ. לכן יש להגדיר עומק חיפוש מקסימלי, ואז כל מצב שהגענו אליו שנמצא באותו עומק יקבל ערך לפי נוסחה מוסכמת מראש (למשל – בשחמט ניתן לתת ערך למצב לפי שווים של כלי הסוכן הנמצאים באותו רגע על הלוח פחות שווים של כלי היריב). נתבונן על דוגמה של עץ Minimax בעומק 4 בכדי להבהיר את המצב אותו אנו מנסים ללמוד:



איור 11.14 עץ Minimax בעומק 4. העיגולים מסמנים את המצבים בהם תור הסוכן, והריבועים מסמנים את המצבים בהם תור היריב. החיצים האדומים מסמנים את הפעולות האופטימליות עבור היריב, ואנו מניחים שהיריב יבחר בהן.

הסוכן מתחיל מהמצב בשורה 0, ובעזרת סימולציה בודק את כל ההסתעפויות עד עומק 4. בשלב ראשון הסוכן משערך באמצעות הסימולציה רק את הערכים של הצמתים **בעומק המקסימלי**. אם זה מצב טרמינלי (כלומר המשחק הסתיים), הערך בעומק זה יהיה הערך שהתקבל בסימולציה, ואם המצב אינו טרמינלי אז הערך נקבע לפי היוריסטיקה קבועה מראש. לאחר מכן הסוכן הולך אחורה וממשקל את כל הצמתים בעומק הקודם תחת הנחת ה-worst case – כלומר מניחים שהיריב יבחר את ההתפצלות הגרועה עבור הסוכן (מסומן בחיצים האדומים). ככה למשל הצומת

השמאלי בעומק 3 יקבל את הערך 10, כי אנו מניחים שבצומת זה הסוכן יבחר ללכת שמאלה. בדומה, הצומת הימני בעומק 3 יקבל את הערך 7−, כי אנו מניחים שהוא יבחר ללכת שמאלה באותו צומת. כך נותנים ערכים לכל השורה הזאת, כאשר הכלל הוא שכל צומת מקבל את הערך **המינימלי** מבין כל אפשרויות הפיצול שלו. כעת עוברים לתת ערכים לשורה בעומק 2, כאשר כאן הכלל הוא הפוך – כיוון שהתור הוא של הסוכן, נניח שהוא יבחר בפעולה הכי טובה, ולכן כל צומת יקבל את הערך המקסימלי מבין הפיצולים האפשריים שלו. למשל הצומת השמאלי בעומק 2 יקבל את הערך 10, כיוון שזהו הערך של ההתפצלות הכי טובה שהסוכן יכול לבחור. באופן הזה עולים למעלה וממלאים את כל ערכי העץ עד השורש, עד שמקבלים ערך עבור המצב הנוכחי בו אנו נמצאים.

כמה הערות חשובות:

1. יתכן והיריב יעשה טעויות ולא יבחר את הצעדים הכי טובים עבורו, ואז בפועל נקבל ערך אחר למצב בו אנו נמצאים, אבל השיטה הזהירה ביותר היא להניח שהיריב יהיה אופטימלי ועל בסיס זה לתת ערכים לצמתים ולמצב הנוכחי. לאחר שביצענו את הפעולה הכי טובה בהתאם להערכה הנוכחית, היריב בתורו בוחר צעד ומתקבל מצב חדש, ואז הסוכן שוב יבצע סימולציות על מנת לחשב את כל ההסתעפויות מאותה נקודה ולתת ערך למצב החדש שהתקבל.

2. כיוון שהעץ גדל אקספוננציאלית, העומקים אותם ניתן לחשב הם לא גדולים, ולכן יש חשיבות להיוריסטיקה שקובעת את הערכים עבור הצמתים שבעומק המקסימלי שלא קיבלו ערך בסימולציה עצמה (כלומר מצבים שאינם טרמינליים).

3. במשחקים בהם יש גם גורם של מזל (כמו למשל הטלת קוביות), ניתן להוסיף שכבת של chance nodes בין כל שתי שורות בעץ, כאשר החישוב ביחס לשכבות החדשות הינו בעזרת התוחלת של התוצאה שלהן. עץ כזה נקרא **Expectiminimax Tree**.

אלגוריתם חיפוש זה עובד טוב עבור מגוון בעיות, אך הוא עדיין נתקל בבעיה עבור אתגרים בעלי מרחב מצבים גדול מאוד, כמו למשל המשחק Go. במקרים כאלה מספר הפיצולים גדול מדי בשביל שיהיה בר חישוב, וממילא לא ניתן לבחון את כל ההסתעפויות עבור עומק סטנדרטי, מה שלא מאפשר לחשב בצורה איכותית את ערכי הצמתים. לכן, במקום לבדוק את **כל** ההסתעפויות, ניתן להיעזר בסימולציה ולבחון רק חלק מהן, ועבור כל סימולציה נלך עד לעומק המקסימלי כדי שהתוצאה תהיה אמיתית ולא תלוית היוריסטיקה. כמובן שבחירת ההסתעפויות לא נעשית באופן אקראי, אלא היא מבוססת על מדיניות נלמדת של שני היריבים. באופן הזה בהתחלה הבעיה היא קלה כיוון שליריב אין אסטרטגיה טובה, אך ככל שאני משתפר כך גם היריב משתפר והבעיה הולכת ונעשית קשה, וממילא ניתן להשתפר עוד ועוד. אלגוריתם חיפוש זה נקרא **Monte-Carlo Search**, וניתן לנסח אותו באופן פורמלי כך:

נניח ונתונה אסטרטגיה התחלתית π שאינה מספיק טובה. נתון מצב s_t, ורוצים לדעת מה הערך של כל פעולה a אפשרית. נדגום K מסלולים עבור כל פעולה a אפשרית, ניעזר בסימולציה כדי לבצע את המסלול עד הסוף, ונחשב את תוחלת הרווח של כלל הסימולציות. באופן הזה, עבור המצב s_t הערך של פעולה a יהיה:

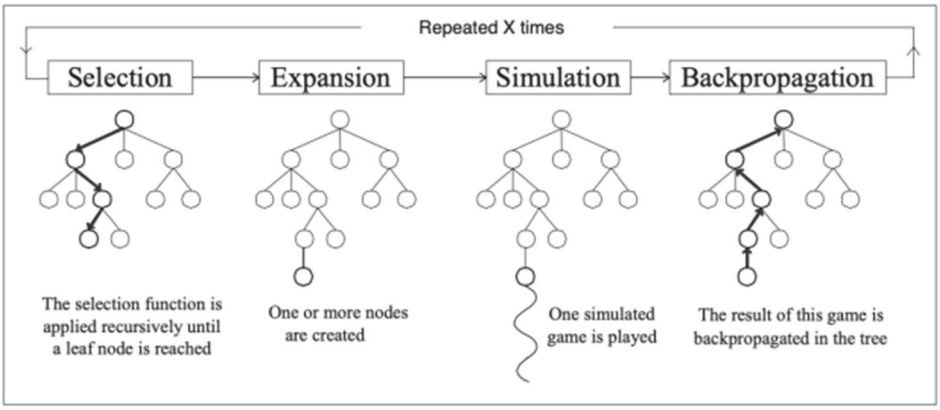
$$Q^{\pi}(s_t, a) = \frac{1}{K}\sum_{k}\sum_{i}\gamma^{i}\mathcal{R}_{t+i+1}^{k}$$

לאחר שחישבנו את הערך של כל הפעולות, נבחר את הפעולה בעלת הערך הכי גבוה:

$$\max_{a} Q^{\pi}(s_t, a)$$

יש להעיר שכדי שהאלגוריתם יעבוד, צריך שהאסטרטגיה ההתחלתית π תהיה ברמה סבירה, כיוון שאם היא גרועה, השיפור יהיה מאוד איטי ולא מספיק.

ניתן לקחת את הרעיון הזה ולהרחיב אותו לחיפוש על עץ כפי שתיארנו קודם. כאמור לעיל, האתגר המרכזי בחיפוש על עץ נעוץ בכך שהוא גדל אקספוננציאלית ולא ניתן לחשב באופן יעיל בעיות עם מספר מצבים גדול מאוד. בכדי להתגבר על כך ניתן להשתמש באלגוריתם **Monte-Carlo Tree Search** (MCTS), שהוכיח את עצמו כמאוד חזק ואף יעיל לבעיות גדולות מאוד. אלגוריתם זה למעשה משלב בין שני חלקים שכל אחד מהם בפני עצמו לא מספיק טוב – מצד אחד יש לנו אסטרטגיה התחלתית, אך היא בינונית ולא מספיק איכותית, ומצד שני יש לנו סימולציה המבוססת על ידיעת המודל, אך היא אינה יכולה לבדוק את כל האפשרויות. הרעיון של MCTS הוא לקחת אסטרטגיה בינונית ולשפר אותה באמצעות בדיקה מושכלת של העתיד – ההרחבה של העץ לא נעשית באמצעות בחינת כל האפשרויות, אלא באמצעות בחירה של כמה אפשרויות שנראות סבירות ביחס לאסטרטגיה הנוכחית. האלגוריתם ניתן לתיאור באמצעות ארבעה שלבים, שחוזרים על עצמם באופן איטרטיבי x פעמים עבור **כל** פעולה a:



איור 11.14 אלגוריתם Monte-Carlo Tree Search: 1. בחירת הצומת הבא אותו רוצים לבחון. 2. הרחבת העץ מאותו צומת. 3. הרצת סימולציה מאותו צומת עד הסוף (=הגעה למצב טרמינלי). 4. נתינת ערך לכל צומת מהצומת הנבחר ואחורה עד השורש.

כאמור, האלגוריתם מורכב מארבעה שלבים:

1. בחירה (Selection) – ראשית יש לבחור את המצב s עבורו אנו רוצים לבדוק מה הפעולה הכי טובה במצב זה. לשם כך נבצע סדרה של צעדים עד שנגיע למצב s_t, כאשר את הצעדים ניתן לבחור על בסיס ה-score שלהם ואפשר גם לשלב רנדומליות עבור exploration כפי שנראה בהמשך.

2. הרחבה (Expansion) – משהגענו לאותו מצב s_t נרחיב אותו בצומת נוסף על ידי בחירה של אחת מהפעולות האפשריות. את הפעולה ניתן לבחור באופן רנדומלי או על בסיס האסטרטגיה הנתונה (שכאמור היא רחוקה מאופטימליות).

3. הרצת סימולציה (Simulation) – כעת נריץ סימולציה מהמצב החדש עד הסוף ונבדוק כיצד הסתיים המשחק. בהתאם לתוצאת המשחק ניתן לאותו מצב חדש ערך.

4. נתינת ערכים לצמתים (Backpropagation) – נחלחל אחורה את הערך שהתקבל ונעדכן את כל הצמתים שנבחרו. אם למשל המשחק הסתיים בניצחון, אז נעדכן את ערך הצומת החדש (שהרחיב את העץ) ל-1/1 (=ניצחון אחד מתוך משחק אחד). בהתאם נעדכן את כל יתר המצבים שעברנו בהם – אם למשל השורש ממנו יצאנו היה בעל יחס של 3/5, אז נעדכן אותו ל-4/6.