

POMDP

- מודל האינטרקציה ב-MDP הוא $MDP = (S, A, T, R)$
- מודל האינטרקציה ב-OMDP הוא $OMDP = (S, A, T, R, O, P)$, כאשר:

$$O = \{o_1, o_2, \dots, o_t\} - \text{תמונות מצב אפשריות}$$

$$P(o|s) - \text{הסתברות לקבל תמונת מצב בהינתן עולם}$$

במודל האינטרקציה, הסוכן לא מקבל מידע מלא על מצב העולם i , אלא רק תמונת מצב (observation) o בהסתברות $P(o|i)$. הוא צריך לבחור פעולה a לפי תמונת המצב, ולקבל $R(i, a)$.

ב-POMDP ניתן להציג את ה-transaction function גם באמצעות מטריצת הסתברויות:

$$T^{A_1} = \begin{bmatrix} 0.6 & 0.4 \\ 0.6 & 0.4 \end{bmatrix}$$

- אם אני נמצא במצב S_1 , בהסתברות 0.6 אני אשאר ב- S_1 ובהסתברות 0.4 אני אעבור ל- S_2
- אם אני נמצא במצב S_2 , בהסתברות 0.6 אני אשאר ב- S_1 ובהסתברות 0.4 אני אעבור ל- S_2

ה-POMDP זה קשה, כי לא רק שהפעולה לא בהכרח מובילה לתוצאה הרצויה, אלא גם לא בהכרח יודעים מה המצב הנוכחי. בדרך כלל policy מתחשב גם בתמונות מצב קודמות, שכן הן משפיעות על חישובי ההסתברויות.

סוגי policy

1. זיכרון סופי - לפי k תמונות המצב האחרונות
2. אוטומט סופי - לפי תמונת המצב הנוכחית
3. believe state policy - תלוי באמונה שלנו על המצב

עדכון האמונה על המצב ב-believe state policy

בנקודת זמן t , אנחנו רוצים לחשב את $P^t(i) = P(S^t = i | o^1, \dots, o^t)$. נעדכן את טבלת ההסתברויות לפי הנוסחה:

$$P^{t+1}(j) = \sum_i P^t(i) T_{ij}^a P(O^{t+1} | S^{t+1} = j)$$

אפיון בעיה

רוצים להסתכל על כמה דברים:

- מידע - שלם? חלקי?
- יריב - יש? אין?
- גודל המשחק - קטן? גדול?

Monte Carlo Tree Search

המודל הזה מאפשר לפתור משחקים שבהם יש או אין יריב, שהמידע שלם או חלקי - אבל לא משחקים שבו העולם מאוד מאוד גדול!

במשחקים שהם גדולים, אי אפשר לפרוש את כל העץ, וצריך להשתמש ביריסטיקות. למשל בשח - מסמלצים מספר מוגבל של מהלכים ומעריכים את טיב הלוח אחרי כל אחד.

הבעיה היא שבדרך כלל במשחקים גדולים אי אפשר לסמלץ יותר מדי מהלכים, וקשה להעריך את הלוח(כי יש יותר מדי גורמים שונים).

$MCTS$ עבור כל פעולה שאפשר לבצע מדמיינים משחק עד הסוף ורואים מי ניצח. במקרה הכי פשוט - מגרילים את הפעולות הבאות. אחרי שמשחקים מספיק פעמים, בודקים אחרי מי מבין המשחקים ניצחנו הכי הרבה פעמים, או קיבלנו הכי הרבה תמורה בתוחלת, או הגענו למצב הכי טוב - ולפי זה בוחרים.

בחירה רנדומית זה לא הדבר הכי חכם, ולכן רוצים לשפר את המשחקים המדומיינים. כאן אפשר להשתמש ב-Reinforcement Learning. לצורך הלמידה, מקובל לנסות קודם כל אחת מהאפשרויות פעם אחת, ואז לכוון להתנהגות שבה ככל שביקרנו בקודקוד יותר פעמים, כך נקטין את ההסתברות שננסה אותו. אבל עדיין רוצים לנסות יותר את היותר מוצלחים. בוחרים מבין הקודקודים בהסתברות פרופורציונאלית ל:

$$v_i + C \cdot \sqrt{\frac{\ln(N)}{n_i}}$$

כאשר:	v_i	הערכת הערך של הקודוד
	C	פרמטר שאפשר לשחק איתו
	N	מספר הפעמים שביקרתי בקודקוד המקורי(איפה שאני נמצא עכשיו - לפני הפעולה)
	n_i	מספר הפעמים שביקרתי בקודקוד הבן(אחרי הפעולה)

אבל בשביל זה צריך לעשות קצת search, ולא רק סימולציה.

פיתוח העץ

1. Selection: בחירת קודקוד ב"חזית"(frontier) - קודקוד שעוד לא פיתחתי לו את כל הבנים
2. Expansion(לא תמיד): מייצרים עלה חדש של אותו קודקוד
3. Simulation: מסמלצים משחק לאחר אותו קודקוד. לא מוסיפים את המצבים של המשחק הזה לעץ!
4. Back Propagation - עולים לאבות של אותו קודקוד, ומעדכנים כל קודקוד כזה שיש לו עוד ניצחון או עוד הפסד.

- ♥ נשים • לא עושים expansion בלי simulation! תמיד נבצע סימולציה אחרי שייצרנו את הקודקוד, כדי שלא יהיו לו 0 משחקים(ואז עלולים לחלק ב0).
- אבל כן אפשר לעשות מדי simulation בלי expansion, כדי לא להגדיל יותר מדי את העץ.
- מצד שני, אין טעם לעשות simulation לקודקוד שכבר פיתחנו לו את כל הבנים - כי כבר עדיף לעשות סימולציה לאחד הבנים.