

למידת חיזוק عمוקה

מרצה : ד"ר גלעד כץ

סיכם : צבי לדר

סמינר חורף תשפ"ג 2023

הסיקום על בסיס מציגות הקורס

תוכן עניינים

5.....	יסודות ה-RL
5.....	Markov Decision Processes (MDP)
5.....	פונקציות הערך
6.....	Bellman משוואת
6.....	Policy Iteration
7.....	Value Iteration
7.....	Model free ו Model based
7.....	Monte Carlo (MC)
8.....	Off Policy ו On Policy
9.....	Temporal Difference Learning
10.....	SARSA
10.....	Off-Policy TD Control: Q-Learning
11.....	Deep Q-learning
11.....	Fixed Q-targets
11.....	Experience replay
12.....	Double DQN
12.....	Dueling DQN
14.....	Policy Gradients
14.....	REINFORCE
15.....	REINFORCE with Baseline
15.....	Actor Critic
16.....	Asynchronous Advantage Actor-Critic (A3C)
17.....	Imitation Learning
17.....	מנחים
18.....	Imitation Learning
19.....	Apprenticeship Learning
19.....	Behavioral Cloning
19.....	Forward Training
20.....	DAgger
22.....	Multi-arm bandits
22.....	Upper Confidence Bound (UCB)
23.....	Gradient Bandits Algorithm
23.....	LinUCB
24.....	Thompson Sampling

26	Monte Carlo Tree Search
26	שלבי האלגוריתם
26	Selection
26	Expansion
26	Simulation
26	Backpropagation
26	תכונות MCTS
27	MCTS לשיפורים
27	AlphaGO
28	מרקח הכניסה (input)
28	Supervised learning policy network
28	RL policy network
29	RL value network
29	Rollout Policy
29	תהליך האימון
29	AlphaGoZero
31	AlphaGO בתחומים אחרים
32	Meta-learning
32	Meta-Learning with Memory-Augmented Networks
33	גישה לזכור.
33	כתבה לזכור
34	פרדיקציה
34	ניסויים
35	Simple Neural Attentive Meta-Learner (SNAIL)
35	תיאור השיטה
36	ניסויים
36	Model Agnostic Meta-Learning (MAML)
37	Adaption in Real-Time
39	Transfer Learning
39	Fine-tuning
40	Progressive Networks
42	Self-Supervision for RL
43	Model-Based RL
43	Actor Mimic

43	Distillation for Multi-Task Transfer
43	Modular Neural Network
45	מרקבי מצלבים ופעולות גדולים
45	מרקחן פעולות רציף
46	Action Elimination with Deep Reinforcement Learning
47	Hierarchical DRL for Sparse Reward Environments
49	בעיות מהעולם האמייתי
51	Branching Dueling Q-Networks (BQD)
52	Jointly-Learned State-Action Embedding
53	Learning in Latent Space
53	Model-Free Approaches for Learning the Latent Space
53	Model-Based Approaches for Learning the Latent Space
56	Embed to Control (E2C)
56	Action-Conditional Video Prediction
58	Inverse Reinforcement Learning
60	סיכון מול model free model based
60	Exploration with Exemplar Model
62	Curiosity-driven Exploration by Self-supervised Prediction
64	Transformers in RL
64	Big-Bird-ו Longformer
65	The Decision Transformer
65	Trajectory Transformer
67	Model-Based Methods
67	גישה ללמידה מודל סביבה
69	LQR
72	iLQR
73	Model Predictive Controls (MPC)
73	Gradient Free Methods

יסודות ה-RL

ב-**Reinforcement learning** אנו מתמודדים עם בעיות בהם נדרש למד **סוכן** לבצע **פעולות** בסביבה מסויימת כך שימקסמו **reward**. הבעיה שאוון נפוץ מאופיינות ברכפים (sequences) של פעולות, תכיפות ו-**rewards**, אשר לרוב אינם מקיימים את הנחת חוסר התלות הסטטיסטי (iid) אשר מאפיינת בעיות sparse, ולווב מאופיינות ב-**reward** דليل (sparse reward), אשר מקשה על הלמידה.

אתגרים ב-RL

- אין סגור, נדרש גישה לסביבה (environment) על מנת לאסוף data.
- שאנו אוסףים מהסביבה תלוי בפוליטה שבה השתמשנו, כאשר משנים פוליטה ההתפלגות של data גם הוא ישנה.
- נדרש ביצוע במקביל גם של למידה (חיזוי) וגם של תכנון (planning).

Markov Decision Processes (MDP)

MDP מאופיין במצבים (states) רציפים או בדידים, המסומנים ב- $\{S = \{s_1, s_2, \dots, s_n\}\}$, פעולות (actions) הממוסנות ב- $\{A = \{a_1, a_2, \dots, a_m\}\}$ ו-rewards הממוסננים ב- $\{R(s, a) \subset \mathbb{R}\}$. כאשר מבצעים פעולה, עבורם מצב אחר. מעבר זה מאופיין בפונקציית הסתברות הבלתייה בפעולה ($a | s \rightarrow P(s' | s, a)$. הנחת המركוביות היא שהמעבר תלוי אך ורק במאפייני המצב הנוכחי (מצב ופעולה) ולא בעבר הרחוק יותר:

$$\Pr(s_{t+1} | s_1, a_1, s_2, a_2, \dots, s_t, a_t) = \Pr(s_{t+1} | s_t, a_t)$$

הנחה זו לא תמיד נכונה, אך ברוב המקרים נכונה אותה. MDP יכול להיות סופי או אינסופי.

כאמור, מה שמייחד בעיות כאלה הוא שנדרשת גישה לסביבה, ה-data שמתקבל תלוי בפעולות, והלמידה והתכנון צרכות להתבצע במקביל.

מטרתנו היא למצוא סט של פעולות שתגרום לתוחלת ה-return להיות מקסימלי. ה-return הוא ממוצע ה-rewards שמתקבלים לאורך הזמן. ממוצע זה יכול להיות ממוצע פשוט (בד"כ במצבים בהם הזמן סופי):

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + \dots + R_T$$

או סכום ממושקל:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

(בד"כ בתהליכים אינסופיים), כאשר $\gamma \in [0, 1]$ בדרך כלל קרובה ל-1 (למשל 0.99).

פונקציות הערך

אנו מגדרים מספר פונקציות בסיסיות אשר נעשו בהם שימוש לכל אורך הקורס. **הפוליטה** היא הפונקציה שבאמצעותה בוחרים פעולה:

$$\pi(s, a) = P(a_t = a | s_t = s)$$

ה-value function היא הפונקציה שמעירכה את תוחלת ה-return שנתקבל תחת פוליטה מסוימת:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right], \text{ for all } s \in S$$

ה-Q-function היא הפונקציה שמעירכה את ה-return שנתקבל אם נבחר פעולה a בצעד הבא ולאחר מכן המשיך לפעול לפי פוליטה π :

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a] = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]$$

הפונקציות האופטימליות π^* , v_{π^*} , q_{π^*} הן הpolloיסה שתיתן תוחלת מקסימלית של return על פני כל הpolloיסות האפשריות, ופונקציות ה-value וה-q המתאימות לה.

Bellman

נשים לב לכך שימושואות ההערכה שראינו הן רקורסיביות. לכן ניתן לרשום אותם בצורה רקורסיבית. את ה- q -function ניתן לרשום :

$$G_t = R_{t+1} + \gamma(R_{t+2} + R_{t+3} + R_{t+4} \dots + R_T) = R_{t+1} + \gamma G_{t+1} \quad : \text{state value function}$$

$$v_{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s] = \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} | S_t = s]$$

את המשוואת ל- $v_{\pi}(s)$ ניתן לפתח :

$$\begin{aligned} v_{\pi}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} | S_t = s] = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) [r + \gamma \mathbb{E}_{\pi}[G_{t+1} | S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\pi}(s')] \end{aligned}$$

ואת $v_*(s)$ (האופטימלי) ניתן לרשום בצורה הבאה :

$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] = \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')]$$

בדומה, ניתן לפתח את $q_*(s)$:

$$\begin{aligned} q_*(s, a) &= \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right] \\ &= \sum_{s', r} p(s', r | s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right] \end{aligned}$$

פתרונות משוואות bellman תניב את הpolloיסה האופטימלית. במקרים בהם כל הנסיבות ידועים (ה-rewards, הפעולות, המצבים ופונקציית המעבר) ניתן להשתמש במשוואת Bellman על מנת לפתור את הבעיה.

Policy Iteration

Policy iteration היא השיטה הבסיסית למציאת הpolloיסה האופטימלית. שיטה זו מבוססת על איטרציות (policy iteration) כאשר כל איטרציה מורכבת מרכיבת policy evaluation ו- policy improvement. בשלב ה-evaluation מתחבצעת הערכה של הpolloיסה (value/q function), ובשלב ה-improvement מתחבצע שיפור של הpolloיסה על סמך פונקציית ההערכה שנלמדה בשלב הראשון.

בשלב evaluation משתמשים במשוואת Bellman כדי לעדכן את ה-value function :

$$v_{k+1}(s) = \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] = \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')]$$

העדכון נעשה בצורה איטרטיבית עד להתקנסות עבור הpolloיסה הנוכחית. בשלב ה-improvement, הדריך הפשוטה ביותר היא לשנות פוליסה עבור מצב אחד בלבד ולהעריך מחדש את הערך שלו. אם הערך שלו גדול, אנו יכולים להיות בטוחים שהpolloיסה טובה יותר. אפשר גם לבצע זאת במספר מצבים במקביל :

$$\pi'(s) = \arg \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] = \arg \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

במקרה של MDP סופי, מספר המ מצבים הוא סופי, ולכן מובטח לנו שגיאע לפוליסת האופטימלית.

Value Iteration

בעיה מרכזית ב-*policy iteration* היא חוסר היעילות שלה. בשינוי פעולה אחד במצב אחד יתכן ונctrax להעניק מחדש את כל המצביעים האחרים. ב-*value iteration*, אנו מבצעים בכל איטרציה שינוי רק **שינוי אחד** בכל מצב כאשר בשינוי אנו מעודכנים את ה-*value function* באמצעות המקסימום שמתתקבל מבין כל הפעולות באותו מצב. למרות זאת, גם במקרה כזה מובטחת התכנסות. כלל העדכון הוא:

$$v_{k+1}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

באופן כללי, policy iteration מתכנס מהר יותר מ-*value iteration*.

Model free ו- Model based

השיטות שראינו עד כה מניחות כי אנו מכירים את כל המצביעים, הפעולות, ה-rewards וכן את הדינמיקה של העולם, כלומר את הסתברויות למעבר במצב בהינתן פעולה. אלגוריתמים אשר

מסתמכים על כל הדברים הללו נקראים **model based**. לעומת זאת, כאשר חלק מהמאפיינים חסרים לנו (למשל, לא מכירם את הסתברויות המעבר ממצב), אנו ננסים לעולם-**model free**. במקרה כאלה בד"כ נemodel ישירות את (a,s) q ללא מידול הסביבה.

Monte Carlo (MC)

בשיטות שראינו עד כה הנחנו שאנו מכירם את פונקציות המעברים $(a,s'|s)$ \hat{G}_t וכן את G_t . במקרה בהם נתונים אלה חסרים ניתן להשתמש בשיטת Monte Carlo, דהיינו מトוך הסביבה ושורץ של הנתונים החסרים על בסיסה. ניתן להמיר לשיטה זו את כל האלגוריתמים שתיארנו. למשל, על מנת להעריך את ה-*value function* ניתן לדוגם מספר מסלולים על פי הפוליסת הקיימת, ולמצע על פני rewards שקיבלו כדי לעדכן אותה. אחת הביעות ב-MC היא שניתן לעשות זאת רק על מסלולים סופיים (episodic).

במהלך *episode*, יכול להיווצר מצב שגיאע למצב (*state*) מסוים יותר מפעם אחת. מכיוון שפונקציית G_t ממצעת את ה-*reward* הנוכחי המצביע והלאה, יש לקבוע מוסכמה איך מתייחסים לSTITואציה בה חוזרים למצב מספר פעמיים. שתי האופציונות המרכזיות הן Every-visit MC ו-First-visit MC. בAOפציה הראשונה אנו ממצאים את ה-*reward* החל מהפעם הראשונה שהגענו למצב ועד סוף ה-*episode*, ואילו באופציה השנייה אנו ממצאים את ה-*return* שהתקבל אחרי כל אחד מה ביקורים.

MC הוא *model free*. אנחנו לא מנסים למודל באמצעותו את המודל של הסביבה. בנוסף, לא ניתן לבצע *policy evaluation* אלא רק ל- (s,a) q עבור הפוליסת הספרטטיבית. במקום לבצע *policy improvement* אבולוציה ל-*state value*, נשמר את הערכיהם שמתקבלים עבור *state* s ופעולה a $q(s,a)$. חשוב לציין, שאם הפוליסת דטרמיניסטית, יכול להיווצר מצב שבו ישנים מצביעים שלא ביקרנו בהם. פתרון אפשרי הוא להתחילה את ה-*episodes* כל פעם במצב אחר ועם פעולה אחרת, כך שככל זוגות המצביעים-פעולות יידגמו. שיטה זו נקראת *exploring starts*. *policy improvement* מתרחש בזרחה חמוצה (*greedy*), כאשר בכל שלב נבחרת הפעולה עם ה- (s,a) q הגדולה ביותר. בזרחה זאת ניתן לבצע *policy iteration* באמצעות *MC*.

יתרונות בשיטת MC :

- ניתן בעורתו למדוד בזרחה ישירה את (s,a) q_π מהסבירה
- אין צורך למודל את הסביבה (*model free*)
- אין צורך למדוד את כל המצביעים האפשריים

חרוניות בשיטת *MC*

- עובד רק במקרים של מסלולים סופיים (*episodic episodes*) שלמים, ולא יכול לבצע *bootstrapping* במהלך ה-*episode*
 - חייב לחכות עד סוף ה-*episode* כדי לדעת מה ה-*return* ולעדכן בהתאם.
- שיטת *TD*, אותה נראה בהמשך, מתמודדת עם בעיות אלה.

Off Policy ו- On Policy

קיימות שתי שיטות מרכזיות בנוגע ליחס בין הפוליסת הנלמדת לבין הפוליסת שמשמשת ליצירת ה-data:

- – בשיטות אלה הפוליסת הנלמדת זהה לו אשר נעשה בה שימוש ביצירת ה-data.
- – בשיטות אלה הפוליסת הנלמדת שונה מזו אשר נעשה בה שימוש ביצירת ה-data.

היתרונות בשיטת On Policy – הלמידה יעילה יותר (ומהירה יותר), מכיוון שאין הבדל בהתפלגות בין ה-data שמננו לומדים לבין המודל שאותו לומדים. לעומת זאת, בשיטות Off Policy, קיימים הבדלים בהתפלגות, ולכן נדרשות שיטות ייעודיות להגבר על הבדל זה, מה שMOVIL למידה איטית ומאתגרת יותר.

היתרונות בשיטת Off Policy – השיטה יותר כללית, ולא נדרשת הרצה מחדש בכל פעם שמעודכנים את הפוליסת. למשל שיטה זו יכולה להתאים למקרה מ-data שאנאף מפעולות אנושיות (למשל נהיגה במכונית). לעומת זאת, בשיטות On Policy, נדרש איסוף data מחדש כל עבור כל עדכון של הפוליסת.

נשים לב לעובדה נוספת, שבשיטות On Policy יותר בולטת ה-*exploration-exploitation trade-off*. הפוליסת הנלמדת זו אשר נעשה בה שימוש זהה, וכך גם נקבע איך ורק אחרי הפוליסת שנראית לנו אופטימלית ברגע זה – לא בוצע אקספלורציה. מנגנון זה, אנחנו מודעים לכך שהפוליסת הנלמדת תהיה תחת אופטימלית, שכן מוכחה לכך בה מרכיב של אקספלורציה.

אתגר נוסף הקדים בשיטות Off Policy הוא שיתכן מאוד שיש **מידע חסר**, למשל מצבים שבהם לא בירנו או פעולות שלא ביצענו. במקרה כזה הסוכן לא ידע איך להתנהג כאשר הגיע תחת הפוליסת הנלמדת.

פתרונות לאתגרי ה-Off Policy

נדיר את הפוליסת הנלמדת π (target policy) ואם הפוליסת שמנה לומדים π (behavior policy) המטריה היא למדוד את ה- π באמצעות ה- b – היא סטוכטית, כך שנitin להגעה באמצעותה לכל מצב, ו- b היא דטרמיניסטית. על מנת להעריך את התוחלת של ה-reward שנתקבל אם נבצע את π ניתן להשתמש בשיטה בשם importance sampling. ההסתברות לקבל מסלול מסוים תחת פוליסת π היא:

$$Pr(s_t, a_t, s_{t+1}, a_{t+1}, \dots, s_T | s_t, a_{T-1} \sim \pi) = \prod_{k=t}^{T-1} \pi(a_k, s_k) p(s_{k+1} | s_k, a_k)$$

נדיר את היחס בין ההסתברות לבצע את המסלול תחת פוליסת π להסתברות לבצע אותו תחת פוליסת b :

$$p_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(a_k, s_k) p(s_{k+1} | s_k, a_k)}{\prod_{k=t}^{T-1} b(a_k, s_k) p(s_{k+1} | s_k, a_k)} = \prod_{k=t}^{T-1} \frac{\pi(a_k, s_k)}{b(a_k, s_k)}$$

כעת, נמתקל את כל המסלולים (על פני כל ה-*episodes*) שבהם עברנו במצב s :

$$V(s) = \frac{\sum_{t \in \mathcal{T}(s)} p_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}$$

כאשר (a) הוא קבוצת כל ה-*time stamps* שבhem בירנו במצב s . היחס שבין ההסתברות של הפוליסה π ל- b מאפשרת לתת משקל מתאים ל- G_t .

Temporal Difference Learning

ראיינו מוקדם יותר כי :

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s]$$

כאשר שיטת MC משערcta את הפונקציה באמצעות $\mathbb{E}_\pi[G_t | S_t = s]$ (דגימה של G_t), ושיטות ה-Dynamic programming לשוגיהן משערכות באמצעות $\mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s]$ (חישוב יישיר של התוחלת באמצעות ידיעת הדינמיקה של העולם). שיטת (0) משלבת את שני העולמות, היא מבוססת דגימה (ולכן יכולה לעבוד גם בלי ידיעת הדינמיקה של העולם), ומשתמשת ב- V_π משוערכת, במקום בערך Q_t המדויק (ולכן מאפשרת עדכון תוך כדי episoide וכן התמודדות עם מסלולים אינסופיים).

עוד הבדל בין MC ל-(0) TD הוא ש-(0) TD משערך את $Q(s, a)$ בעוד MC משערך את (s) . כלל העדכון של TD(0) הוא :

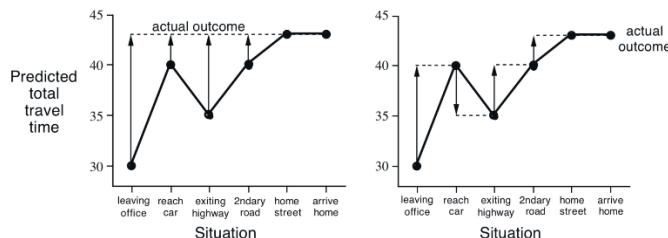
$$V(s) \leftarrow V(s) + \alpha [R + \gamma V(s') - V(s)]$$

הביטוי $R + \gamma V(s_{t+1}) - V(s_t)$ נקרא $\delta_t = R_{t+1} + \gamma V(s_{t+1}) - V(s_t)$. TD error. יש להעיר כי העדכון תלוי במידע s_{t+1} ולכך נדרש לחכום שלב לפני העדכון. ההנחה היא ש- (s) V לא משתנה ממצב למצב.

להלן דוגמה להערכת (0) TD :

	Elapsed Time [minutes]	Predicted Time to Go	Predicted Total Time
Leaving office	0	30	30
Reach car, raining	5	35	40
Exiting highway	20	15	35
Narrow road, stuck behind truck	30	10	40
Entering home street	40	3	43
Arrive home	43	0	43

בדוגמה זאת, בכל נקודה בזמן מחושב הזמן שעבר (state-reward) שהתקבל לשעבר (shocked to reward) והזמן המשוער שנשאר (shocked to reward) של המצב הבא. בכל מצב אנו מתקנים את ההערכתה הקודמת שלנו $V(s_t)$ ו- $V(s_{t+1})$ לפי התכפית שבייצנו (R) ועוד ההערכתה שלנו בשלב זה (V). התרשימים שלහן מדגים את ההבדל בין MC ל-(0) TD :



בעוד שב-MC אנו נאלצים ללחכות לסוף (אחרי שגילינו שלקח 43 דקות) ואנחנו מעדכנים את כל המ מצבים לפי התצפית שקיבלנו בסוף, ב-(0) העדכון מתבצע בכל שלב במהלך ה-episode. בנוסף, בעוד שב-MC כל העדכונים זהים (לפי מה שהתקבל בסוף), ב-(0) העדכונים משתנים במהלך הזמן, מכיוון שהם מבוססים על הרכות שמתבצעות במהלך ה-episode.

SARSA

ניתן להשתמש ב-**TD learning** גם על מנת ללמד את ה-**Q-function**. אלגוריתם זה נקרא SARSA. זהו אלגוריתם On policy, וכל העדכון שלו הוא מהצורה הבאה:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

גם כאן מוגדר TD error :

$$\delta_t = R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

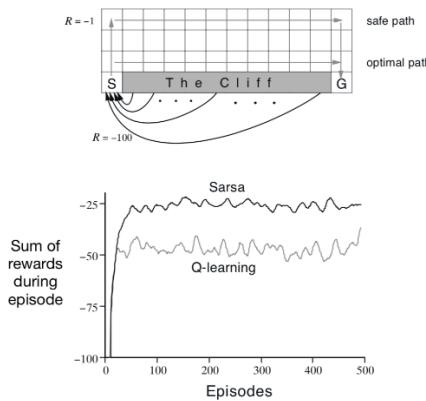
Off-Policy TD Control: Q-Learning

Q-learning הוא עוד וריאנט של TD control. שיטה זו היא off policy (לא מעריכים את הפוליסיה הנוכחית אלא את הפוליסיה האופטימלית). בשיטה זו, כלל העדכון הוא מהצורה הבאה:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

פעולת max גורמת לכך שנלמד את הפוליסיה האופטימלית, והוא זו שגורמת לאלגוריתם להיות off policy. עוד הבדל בין SARSA ל-Q-learning זה שב-SARSA אנו מעריכים להערך פוליסיה ספציפית, ולכן נעקוב אחרי הפוליסיה הזאת במהלך האלגוריתם, בעוד שב-Q-learning עוקבים אחריו הפוליסיה האופטימלית בצורה חמדנית, ולכן אנו נדרשים לבצע אקספלורציה (למשל על ידי ϵ -greedy).

להלן דוגמה הממחישה הבדל בין האלגוריתמים:



בדוגמה זו, המטרה היא להגיע מ-S ל-G בזמן step reward (ה-step הוא 1- על כל 100 נסילה מהצוק). בשני המקרים אנו מבצעים פוליסיה שהיא ϵ -greedy. ההבדל הוא, ש-Q-learning לומד את הפוליסיה האופטימלית (off policy), שכן הוא מנסה שחל מהפעולה הבאה נבחר את כל הפעולות האופטימליות. הוא לא מתחשב בכך שהפוליסיה היא ϵ -greedy ולכן בaczורה מובנית לא נבחר את הפעולה האופטימלית בהסתברות מסוימת. בסופו של דבר, הפוליסיה הנלמדת היא אכן האופטימלית, אך מכיוון שלכל האורך אנו לא עוקבים במדויק אחריה אלא מבצעים אקספלורציה, הסוכן בהכרח הוא יפול מהצוק בחולק מהמרקירים. לכן בגרף רואים שה-return שלו נמוך בממוצע מ-SARSA. SARSA לעומת זאת מבצעת אופטימיזציה על הפוליסיה עצמה, ולכן אם הפוליסיה היא ϵ -greedy, האלגוריתם יקח בחשבון שישנה הסתברות שתיבחר פעולה אקראיית, וכך הוא יבחר את הפוליסיה ה"בטוחה" יותר שבתוחלת מניבה את הרוחות הגבוהה.

Deep Q-learning Fixed Q-targets

בdeep Q-learning אנו משתמשים ברשות על מנת למדל את ה- Q -function $Q(s, a, \theta)$. את הלמידה מבצעים על ידי מינימיזציה של ה- MSE :

$$MSE = \left(r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta) \right)^2$$

ה-loss function מוגדר:

$$L_i(\theta_i) = \mathbb{E}[(y_i - Q(s, a, \theta_i))^2]$$

וה-backpropagation מתבצע לפי כלל העדכון הבא:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}[(R_{t+1} + \gamma \max_{a'} Q(s', a', \theta_{i-1}) - Q(s, a, \theta_i)) \nabla_{\theta_i} Q(s, a, \theta_i)]$$

לעתים, קשה לאלגוריתם להתכנס. חלק מהסיבות הן שהדgelmoths אינן בלתי תלויות (למשל רצף ארוך של $r = 0$ לכל אורך המסלול, כאשר בסופו יש צעד אחד ש- $i = t$), וכן שמתבצע אופטימיזציה במקביל על "הדgelmoths" $Q(s, a, \theta)$ ועל "התיאוג" $(s', a', \theta) + \gamma \max_{a'} Q(s', a', \theta)$, או במילים אחרות, ה-target אינו סטציוני. דרך נוספת להתגבר על הבעיה האחורונה היא "להקפיא" את הפרמטרים של רשות target כך שהלמידה תהיה יציבה יותר:

$$MSE = \left(r + \gamma \max_{a'} Q(s', a', \theta') - Q(s, a, \theta) \right)^2$$

כאשר אחת למספר צעדים מתבצע עדכון $\theta \leftarrow \theta'$.

Experience replay

בעיה נוספת, אשר קשורה לביעית התלות שהזכרנו לעיל, היא בעיית השכחה. מכיוון שהdgelmoths שהרשות רואה הן טוריות, הרשות עלולה "לדרוס" את הניסיון שהוא צברה בעבר, למורות שהוא יכול להיות רלוונטי לעתיד. למשל, במעבר משלב במשחק מחשב, הרשות עלולה לשוכוח את מה שלמדה בשלב הראשוני מכיוון שלרוב, התפלגות data-*the* בא-*stage* שונה מזו שבשלב הקודם. בעיה נוספת (שהזכרנו לעיל) היא בעיית הקורלציה בין dgelmoths. ב-supervised learning בעיה זו לא קיימת מכיוון שנitin להניח שהdgelmoths בלתי תלויות (למשל מתבצע shuffle של dgelmoths בכל epoch). לעומת זאת, בעולם RL לא ניתן להניח דבר כזה, ולמידה של רשות הינה מתוגרת במצב זה (קיים חשש ל-*overfitting* בעבר הקרב). Experience replay הוא פתרון לשתי הבעיה הללו. בשיטה זו, אנו שומרים dgelmoths ישנות על מנת להשתמש בהן שוב במהלך האימון בעתיד.

השאלה היא באיזו צורה להציג את dgelmoths מחדש. הדרך פשוטה ביותר היא akarait – dgelma ionporamit. בשיטה זו גם יעללה בהרבה מקרים. אך קיימים מקרים בהן האינפורמציה החשובה לא מפולגת ionporamit בין dgelmoths בעבר. קיימות שתי שיטות לדגימה מתחכם יותר: dgelma mbosst TD error – prioritization stochastic prioritization. ההנחה מאחרורי השיטה הראשונה היא שאנו מוחפשים dgelmoths אינפורמיטיביות לרשota. dgelmoths בעלות גובהה מעידות על כך שהראשית "מופתעת" מהתיאוג שלו, וכך הנו תרימה אינפורמיטיביות יותר ללמידה. בשיטה זו, ההסתברות להשתמש בdgelma שוב תהיה פרופורציונלית לגודל dgelma – TD error.

הבעיה בשיטה זו היא שמכיוון שהיא מבוססת על אלגוריתם חמדן (greedy) – מציאת TD error הגבוה ביותר ברגע זה – חלק מהdgelmoths לא תידגמנה זמן רב ואף עלולות להימחק מה-buffer. זה יכול להיות בעייתי למשל בסביבה בה reward –*rewards* סטוכסטי. אותה dgelma מקבלת חלק מהפעמים reward וחלק מהפעמים לא, ותהליך dgelma מוטה (יש אפשרות למשל שה-TD error יהיה גדול בפעם הראשונה reward ורך dgelmoths הללו יוצגו לרשות). בנוסף יש חשש ל-*overfitting* לדgelmots שיצאו שוב ושוב לרשות. לכן, שיטת stochastic-

prioritization מנסה למצוא פשרה בין דגימות של דגימות אינפורטטיביות לדגימה אקראית. שיטה זו משתמש במשוואת הבאה לדגימה מתוך ההיסטוריה :

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

כאשר i היא החסתברות לדגימה של דוגמה i פי ה- α , TD error $= 0$ והוא פרמטר החלקה. אם נבחר $0 < \alpha < 1$ הדגימה תהיה יוניפורטטיבית, ועבור $\alpha = 1$ הדגימה תהיה זהה לשיטה הקודמת. α יכול להיות מוגדר כך : $i = \epsilon + |\delta_i|$ כאשר δ_i הוא ה- δ TD error ו- ϵ הוא מספר קטן אשר מודא שישנו סיכון קטן לכל דגימה. לחילופין, ניתן להגדיר $P(i) = \frac{1}{rank(i)}$ כאשר $rank(i)$ הוא המיקום של דגימה i אם ממיינים את הדגימות לפי ה-TD error.

Double DQN

אחד מהקשיים באימון רשת DQN הוא חוסר הדיוק של רשת בבחירה Q-values. אם פעולה שאינה אופטימליתOTAHLAH עם Q-value גבוהה – יκח זמן לערך להתקדם. בעיה זו נובעת מכך שהאלגוריתם בטור target את ערך ה-Q-הגבוה ביוטר מיותר כל הערכים, שכן תחיה הטיה לכיוון האופטימי. ניתן להתגבר על הבעיה באמצעות שימוש בשתי רשותות במהלך האימון, כאשר כל רשות מאומנת על דגימות שונות. כזכור, ה-TD error מחושב על ידי :

$$r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta)$$

כאשר ה-target הוא $(r + \gamma \max_{a'} Q(s', a', \theta))$. בשיטה זו, הביטוי $\max_{a'} Q(s', a', \theta)$ משתנה. במקום לחת את ה-Q-value המksamלי של הרשות שעלייה מבצעים את האופטימיזציה, לוקחים את ה-Q-value של אותה פעולה מרשת אחרת. במקרה אחר, מבצעים שני שלבים. בשלב ראשון מחשבים את הפעולה שמניבת ערך מקסימלי בפונקציה $(Q(s', a') = \arg \max_{a'} Q(s', a', \theta))$, ובשלב שני מחשבים את ה-Q-value של אותה פעולה ברשות השנייה $(Q(s', a^*, \theta))$. שיטה זו מאפשרת לקבל משערץ לא מوطה (unbiased) ל-Q-value המksamלי. להלן האלגוריתם המלא :

Algorithm 1 Double Q-learning

```

1: Initialize  $Q^A, Q^B, s$ 
2: repeat
3:   Choose  $a$ , based on  $Q^A(s, \cdot)$  and  $Q^B(s, \cdot)$ , observe  $r, s'$ 
4:   Choose (e.g. random) either UPDATE(A) or UPDATE(B)
5:   if UPDATE(A) then
6:     Define  $a^* = \arg \max_a Q^A(s', a)$ 
7:      $Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(s, a) (r + \gamma Q^B(s', a^*) - Q^A(s, a))$ 
8:   else if UPDATE(B) then
9:     Define  $b^* = \arg \max_a Q^B(s', a)$ 
10:     $Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(s, a) (r + \gamma Q^A(s', b^*) - Q^B(s, a))$ 
11:   end if
12:    $s \leftarrow s'$ 
13: until end

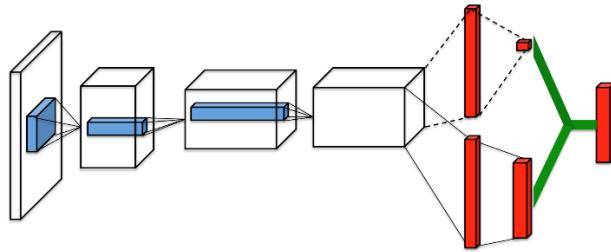
```

Dueling DQN

המטרה של Dueling DQN היא הפרדת הצימוד שבין ערך המצב (*state value*) לערך הפעולה (*action value*). הפרדה זו מאפשרת לרשות להתמקד בהבדלים שבין הפעולות האפשרות בכל מצב, ולא בערך המוחלט של ה-*Q-function* שלהם. הפרדה זו מאפשרת למידה מהירה יותר, שכן היא מפרידה שני ערכים שאינם תלויים אחד בשני, וכן מאפשרת הכללה טובה יותר. הרשות מוציאה ערך Q אשר מורכב משני ערכים :

$$Q^\pi(s, a) = V^\pi(s) + A^\pi(s, a)$$

להלן איור של הארכיטקטורה :



במהלך האימון והפרדיקציה, מתבצע נירמול של $A^\pi(s, a)$ כך שהמוצע יהיה אפס (מחסרים את הממוצע מכל אחד מהערכים). זה מאפשר למידה של **היתרון** של פעולה ביחס לשניה, במקום הערך המוחלט שלה :

$$Q(s, a, \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|A|} \sum_{a'} A(s, a'; \theta, \alpha) \right)$$

Policy Gradients

השיטה שראינו עד עכשו מבוססת על *action value function* ועל *value function*. בשים לב ש**gradient ascent** לפרמטרים של המודל על מנגנון למקסם את ה-*reward* שמתќבל מהפעולות שהמודל מציע. במלטה עם פעולות רציפות המודל יוציא את התוחלת והשונות של גאוסיאן על אוסף הפעולות, ובמלטה עם גאוסיאן את התוחלת והשונות של גאוסיאן שמננו נדגום את הפעולה.

יתרנו שיש לשיטות אלו על פניהם שיטות *-greedy*, שהפעולות משתמשות בצורה רציפה, ללא שינויים חדים. כאשר מתќבלת פרדיקציה מודל הpolloisa, ניתן לדגום את הפעולות לפי ההתפלגות שהתקבלה והתפלגות זו יכולה להשתנות בצורה רציפה ואיתית במהלך הלמידה.

המשוואה הבסיסית ב-*policy gradients* היא :

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla_\theta \pi(a|s, \theta)$$

(s) μ הוא מספר הפעמים הממוצע שמבקרים במצב s .

יתרונות ל-*policy gradients* :

- ניתן למודל בצורה טبيعית את ההסתברויות לבחירת כל פעולה
- ניתן בצורה טبيعית לקבל איזו בין *exploitation* ו-*exploration*
- ניתן למודל בצורה טبيعית מרחבן פעולות רציפים
- בחלק מהמקרים אפשר למודל בעיה בצורה יותר קלה באמצעות *policy gradients*

חסרונות ל-*policy gradients* :

- השיטה דורשת הרבה דוגמה (שכנן בכל פעם שהpolloisa מעט משתנה דוגימות העבר כבר אין מייצגות אותה)
- במקרים רבים השיטה מתכנסת לאט יותר מאשר שיטות אחרות

REINFORCE

את הביטוי שלעיל ניתן לפתח לצורה הבאה :

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla_\theta \pi(a|s, \theta) = \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla_\theta \pi(a|S_t, \theta) \right]$$

כאשר S_t הוא משתנה מקרי המכיל את המצב (מתפלג לפי π). נבצע עוד טריק מתמטי :

$$\mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla_\theta \pi(a|S_t, \theta) \right] = \mathbb{E}_\pi \left[\sum_a \pi(a|S_t, \theta) q_\pi(S_t, a) \frac{\nabla_\theta \pi(a|S_t, \theta)}{\pi(a|S_t, \theta)} \right]$$

בעת ניתן לבדוק שהביטוי $\sum_a \pi(a|S_t, \theta) q_\pi(S_t, a)$ הוא בעצם התוחלת על $q_\pi(S_t, A_t)$ כאשר A_t הוא משתנה מקרי של הפעולות, מפולג לפיה, במילאים אחרים, $\sum_a \pi(a|S_t, \theta) q_\pi(S_t, a) = \mathbb{E}_\pi[q_\pi(S_t, A_t)] = G_t$:

$$\mathbb{E}_\pi \left[\sum_a \pi(a|S_t, \theta) q_\pi(S_t, a) \frac{\nabla_\theta \pi(a|S_t, \theta)}{\pi(a|S_t, \theta)} \right] = \mathbb{E}_\pi \left[G_t \frac{\nabla_\theta \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right]$$

בעת, ניתן להשתמש בשיטות דוגמה כמו Monte Carlo על מנת לשערץ את התוחלת הניל. השיטה שמשתמשת בזיה נקראת *REINFORCE*, וכלל העדכון למשקولات בשיטה זו הוא :

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla_\theta \pi(A_t | S_t, \theta_t)}{\pi(A_t | S_t, \theta_t)}$$

האינטואיציה היא שכיוון הגרדיאנט $\nabla_\theta \pi(A_t | S_t, \theta_t)$ הוא הכוון שגורם לרשת לבצע את A_t בהסתברות הגדולה ביותר. את הגרדיאנט זהה מכפילים ב- G_t , כך שאם $reward$ -*baseline* יהיה חיובי נבצע עדכון על מנת להגדיל את ההסתברות לבצע את פעולה A_t , ואם יהיה שלילי נקטין אותה. בנוסף, מכנה נמצאת ההסתברות לביצוע הפעולה, נרמול זה קורה מכיוון שאנו רוצים לעדכן באופן שווה את כל הפעולות שנלקחות, ולא לתת משקל גובה יותר (כלומר מספר עדכנים גבוה יותר) לפעולה שתיבחר בהסתברות גבוהה.

מכיוון שבשיטה זו אנו זוקקים להערכתה של G_t , ניתן לבצע אותה רק בתום *episode*, וכן היא מתאימה רק לבעיות *episodic*.

REINFORCE with Baseline

מכיוון שאלגוריתם *REINFORCE* מבוסס דגימה, ה-*reward* עלול להיות רועש, ויהיה קשה להגיע אליו להתקנות. אחד הפתרונות לבעה זו הוא להוסיף *baseline*, ככלمر נירמול ערכי העדכון כך שהתחולת שלהם על פני כל הפעולות תהיה אפס:

$$\theta_{t+1} = \theta_t + \alpha((G_t - b(S_t)) \frac{\nabla_\theta \pi(A_t | S_t, \theta_t)}{\pi(A_t | S_t, \theta_t)})$$

בחירה טבעית ל-(*state-value function*) $b(S_t, w)$, שכן היא אינה תלולה בפעולה A_t והיא תNORMAL את הביטוי כך שהתחולת תהיה אפס. כך הלמידה תהיה – איזו פעולה טובה יותר מהמומוצע ואיזה גרוועה יותר. ניתן לשערך את הפונקציה $w(S_t)$ על ידי דגימה.

Actor Critic

ראינו עד כה שתי קטגוריות של שיטות *Actor only* – (*REINFORCE policy gradients* כמו *DQN*), ו- *Critic only* (למשל *Q-function*). החיסרון בראשונות הוא שמכיוון שהן מבוססות דגימה והפוליסת משנה עם הזמן מספר הדגימות הדרושים הוא גדול מאוד. הדגימות שיצרנו בתחילת האימון כבר לא רלוונטיות להמשך האימון מכיוון שהתפלגות הפעולות שונה. הבעיה בשיטות *Critic only* היא שאנו לא מבצעים את האופטימיזציה ישירות על הערך שאנו מעריכים, אלא על ערך אחר (למשל *state value function*), מה שגורם לכך שהשיטה לא תהיה תמיד אופטימלית. שיטות *actor critic* מנסות "להנות משני העולםות".

בשיטות *actor critic*, מבצעים בזורה איטרטיבית עדכון לפוליסת *state value function* ול-*Q-function*. ה-*policy gradients* יכול להשתמש ב-*data bootstrapping* שנוצר באמצעות פוליסת אחרת, וכן ניתן להשתמש ב-*TD*.

האלגוריתם הבסיסי של *actor critic* מזכיר שילוב של *REINFORCEMENT with baseline* ושל *TD*:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \alpha((G_t - \hat{v}(S_t, w)) \frac{\nabla_\theta \pi(A_t | S_t, \theta_t)}{\pi(A_t | S_t, \theta_t)}) \\ &= \theta_t + \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)) \frac{\nabla_\theta \pi(A_t | S_t, \theta_t)}{\pi(A_t | S_t, \theta_t)} \\ &= \theta_t + \alpha \delta_t \nabla_\theta \ln \pi(A | S, \theta) \end{aligned}$$

כאשר $\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)$

עדכון ה-*value function* :

$$w \leftarrow w + \alpha^w \delta \nabla_w \hat{v}(S, w)$$

Asynchronous Advantage Actor-Critic (A3C)

כאשר עוברים לשיטות actor critic מתומות לנו גמישות לבצע שיטות התייעלות שונות באימון. דוגמה לכך היא ה-A3C. בשיטה זו מנצלים את העובדה שה-critic אינוריאנטי לפוליסה שנבחרת, כלומר תלוי רק במצב ולא בפעולות או בפוליסה. לכן, ניתן למקבל מספר actors שכולם משתמשים באותו critic (כלומר אותה importance function) וכולם מעדכנים אותו. ניתן אפילו אחד יעקוב אחרி פוליסה אחרת (ולבצע sampling). העדכון של ה-value function מתקבל כאחת במספר צעדים, וכך נקרא asynchronous (sampling). הגדריאנטים מתווספים (accumulating gradients) ומתרכזים אותם עדכון רק לאחר מספר מוגדר מראש של צעדים. להלן פירוט האלגוריתם (באלגוריתם זה נעשה שימוש באותה פוליסה ללא :

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```

// Assume global shared parameter vectors  $\theta$  and  $\theta_v$  and global shared counter  $T = 0$ 
// Assume thread-specific parameter vectors  $\theta'$  and  $\theta'_v$ 
Initialize thread step counter  $t \leftarrow 1$ 
repeat
    Reset gradients:  $d\theta \leftarrow 0$  and  $d\theta_v \leftarrow 0$ .
    Synchronize thread-specific parameters  $\theta' = \theta$  and  $\theta'_v = \theta_v$ 
     $t_{start} = t$ 
    Get state  $s_t$ 
    repeat
        Perform  $a_t$  according to policy  $\pi(a_t | s_t; \theta')$ 
        Receive reward  $r_t$  and new state  $s_{t+1}$ 
         $t \leftarrow t + 1$ 
         $T \leftarrow T + 1$ 
    until terminal  $s_t$  or  $t - t_{start} == t_{max}$ 
     $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t // \text{Bootstrap from last state} \end{cases}$ 
    for  $i \in \{t - 1, \dots, t_{start}\}$  do
         $R \leftarrow r_i + \gamma R$ 
        Accumulate gradients wrt  $\theta'$ :  $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i | s_i; \theta') (R - V(s_i; \theta'_v))$ 
        Accumulate gradients wrt  $\theta'_v$ :  $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$ 
    end for
    Perform asynchronous update of  $\theta$  using  $d\theta$  and of  $\theta_v$  using  $d\theta_v$ .
until  $T > T_{max}$ 

```

Imitation Learning

Imitation learning הוא תחום ב-RL בו מתבצעת למידה על בסיס **חיקוי של פעולות של מומחה** (בדרך כלל אדם אשר מבצע את הפעולה והאלגוריתם לומד מהפעולות – תיוגים – שלו). נטמקד במספר שיטות – **למידה של מודל סביבה** באמצעות דוגמאות המומחה (apprenticeship learning), **למידה ישירה של פוליסת מומחה** (forward training) ו**למידה מתמשכת בסביבה** (behavioral cloning) (DAgger).

מונחים

Regret הוא ההפרש בין ה-reward שיתקבל במקרה של פוליסת אופטימלית לעומת המתקבל באמצעות הopolיסת הנוכחית. כמובן שבמהלך הלמידה נרצה למזער את ה-regret בכל הינו ובקץ להתקרב לפוליסת האופטימלית. נגידר את ה-loss עבור צעד ייחיד :

$$l_t = \mathbb{E}[V^* - Q(a_t)]$$

כאשר V^* הוא תוחלת ה-reward בצעד זה כאשר נבצע את הפעולה האופטימלית a^* , ו- $Q(a) = \mathbb{E}[r|a]$

הוא תוחלת ה-reward עבור הצעד הנוכחי (לא להתבלבל עם ה-Q-function, פונקציית $Q(a)$ מתייחסת לתוחלת ה-reward **בעוד הנוכחי**, ולא ממוצעו שלו על פני הזמן). אם כן l_t הוא ההפרש בין ה-reward המМОוצע שיתקבל תחת פוליסת אופטימלית, לבין שיתקבל אם נבצע את הopolיסת הנוכחית. ה-regret הכללי מוגדר כ:

$$L_t = \mathbb{E} \left[\sum_{t=1}^T (V^* - Q(a_t)) \right]$$

ממוצע ה- l_t על פני הזמן. במהלך הלמידה נרצה למזער את L_t . אפשר לבטא את L_t באמצעות נוטציה קצרה שונה :

$$L_t = \mathbb{E} \left[\sum_{t=1}^T (V^* - Q(a_t)) \right] = \sum_{a \in A} \mathbb{E}[N_t(a)](V^* - Q(a)) = \sum_{a \in A} \mathbb{E}[N_t(a)] \Delta_a$$

כאשר $N_t(a)$ הוא מספר הפעמים שהתבצעה פעולה a (count), ו- Δ_a מוגדר כ- $\Delta_a = V^* - Q(a)$. מוגדר כת-*loss* על מנת לכמת איזה אלגוריתם לומד יותר טוב על פני הזמן, נהגים להשתמש במונח *total regret*. מונח זה מתייחס ל-*regret* המצטבר כפונקציה של הזמן. למשל אלגוריתם חמדן (*greedy*) המזער את ה-*loss* עבור הצעד הקרוב :

$$a_t^* = \arg \max_a \hat{Q}_t(a)$$

ה-*total regret* שלו יהיה לינארי, מכיוון שהוא לא מבצע אקספלורציה הוא יכול להיות "תקוע" בפוליסת שאינה אופטימלית, ובקרה כזה ה-*regret* יעלה בצורה לינארית כטילות בזמן. לאלגוריתמים ϵ -*greedy* – *total regret* יהיה טוב יותר, מכיוון שהם מבצעים אקספלורציה, אך מצד שני, מכיוון שקיים רכיב אקראי, ה-*loss* שלהם יהיה חסום מלמטה (מכיוון שתמיד הם יבצעו פעולות שאינן אופטימליות בגל האקראיות שליהם) :

$$l_t \geq \frac{\epsilon}{A} \sum_{a \in A} \Delta_a$$

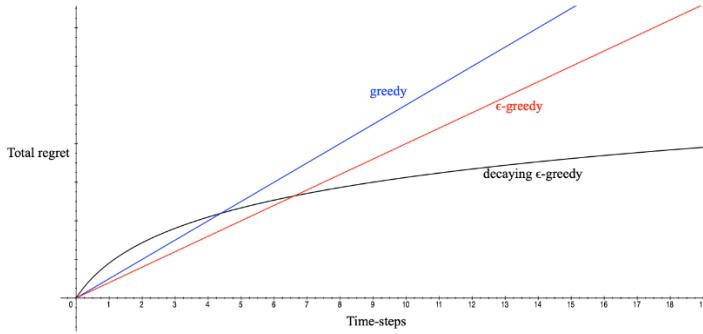
הביטוי מרכיב מה-*loss* הממוצע על פני כל הפעולות $\frac{1}{A} \sum_{a \in A} \Delta_a$ והסתברות לבצע אותן ϵ . גם אם הפוליסת שלנו אופטימלית, ו- $l_t = 0$ כאשר מביצים *exploitation*, במקרים בהם מבצעת אקספלורציה (בהסתברות

ϵ) ה- $regret$ הממוצע יהיה $\sum_{a \in A} \Delta_a$, ולכן $\sum_t \Delta_a$ חסום בהכרח. לאלגוריתמים ϵ -greedy גם יהיה ϵ -greedy (אם כי, כאמור, נזוק יותר מאלגוריתם חמדן).

פתרון אפשרי להורדת $total\ regret$ הוא שימוש ב- ϵ מספייק איטית כך שמתאפשר תהליך אקספלורציה, ניתן להוריד את $total\ regret$ מלינארי. דבר זה מתאפשר מכיוון שככל שעובר הזמן אנחנו פוחות ופחות מבענאים פועלות רנדומליות, ומתקרבים לפוליסיה האופטימלית. למשל, אם בוחרים את ϵ כזרה הבא:

$$\epsilon_t = \min \left\{ 1, \frac{c|A|}{d^2 t} \right\}$$

כאשר $c > 0$ הוא קבוע, t הוא הזמן ו- $i = \min_a \Delta_i$ (אלא אם $\Delta_i = 0$ הוא ה- gap המינימלי, ה- $total\ regret$ ירד אסימפטוטית כמו פונקציית \log). הבעיה היא שבדרך כלל לא נדע את הפוליסיה האופטימלית, ולכן לא נדע את d . להלן גורף המתאר את ה- $total\ regret$ כתלות בזמן עבור שלושת האלגוריתמים:



Imitation Learning

קיים מספר אתגרים ייחודיים כאשר מתמודדים עם בעיות בעולם RL . בשונה מביעות קלאסיות בתחום RL , סוכן RL נדרש לבצע פעולות סדרתיות, כאשר המבצעים בהם הוא נתקל והפעולות אותן הוא מבצע אינן בלתי תלויות. תכוונה זו הינה מהתגרת לאלגוריתמי בינה מלאכותית (רשתות נוירוניים). בנוסף, הבעיות במרקטים רבים מורכבות מממדים רבים (למשל סוכן המתבסס על מצלמה להחלטה על פעולה), הבעיות הרבה פעמים לא ניתנים לניתוח (שינוי בפעולה גורם לשינויי בתוכה), וקיימים גם ממד של אקראיות, במיוחד בעקבות עולם אמיתי.

פתרון אפשרי לביעות אלה הוא לנשות לקרב את הבעיה לבעיית $supervised\ learning$ פשוטה, על ידי המרת הבעיה לסדרה של בעיות פרדיקטיביות. ב- $imitation\ learning$ אנו משתמשים בפעולות שמומחה אנושי ביצוע, ומלהדים את הסוכן לבצע פעולה בהינתן מצב.

הבעיה היא שגיישה זו היא מאוד עדינה, ויכולת שלא להיות יציבה במרקטים רבים. למשל, אם מתבצעת למידה ישירה ממומחה, הסוכן ילמד את הפעולות האופטימליות עבור מצבים שהמומחה נתקל בהם, אבל יתכן כי יתתקל במצבים אחרים שונים והוא יתקשה להקליל. בעיית ה-"**compounding error**" מתייחסת למקרים בהם סטייה קטנה בפרדיקטיביות יכולה לגרום סטיות משמעותיות מעלה הדרך, מכיוון שהסתטיה הקטנה גורמת לסוכן לפגושים מצבים שהוא לא מכיר, מה שגורם לשגייה נגררת". במצבים כאלה הסוכן יתקשה לחזור למצבים מוכרים. בהמשך נציג גישות להתמודדות עם בעיה זאת.

נקודת מבט אחרית על הבעיות ב- $imitation\ learning$ היא שאנו מעוניינים שהסוכן ידע להתמודד עם כל מצב שהוא יפגוש. אך אם אנו מtabססים רק על דוגמאות שנאספו ממומחה, ה- $dataset$ שלנו סופי ולא מכסה את כל עולם הבעיה. אלגוריתמים שנפגש בהמשך מנסים לפתור זאת על ידי מציאה של "מצבים מעוניינים",

כלומר זיהוי איזה מצלבים יכולים להיות משמעותיים בלמידה, ו"תיגר" שלהם (על ידי המומחה) בצורה איטרטיבית.

Apprenticeship Learning

Apprenticeship Learning הוא אחת מהשיטות הבסיסיות ביותר של Imitation learning. בשיטה זו נעשה שימוש-ב-data שනאסר על ידי expert (למשל על ידי אדם שביצע את המשימה) על מנת **ללמוד את הדינמיקה של העולם**. לאחר שהדינמיקה נלמדה, לומדים פוליסיה על בסיס המודל של העולם באמצעות אלגוריתם RL בסיסיים. במידה שהפוליסיה שנלמדה לא טובה מספק, מה שמעיד על כך שהמודל של העולם לא למד טוב, נאוסף את המצלבים החדשניים בהם נתקלנו ו"נтиיגר" אותם מחדש על ידי המומחה. בצורה איטרטיבית האלגוריתם יתכנס לביצועים טובים.

שיטת כזאת מתאימה למרקם בהם קשא לביצוע אקספלורציה, למשל כאשר איסוף כזה data הוא יקר. החסרונות המרכזים של השיטה הם שלא ניתן להתמודד עם מצלבים (states) שלא נצפו בעבר, שכן מה שנלמד זהו מודל של הסביבה, ומודל זה נלמד אך ורק באמצעות המצלבים שהמומחה נתקל בהם. וכן, מאותה סיבה, קשא בשיטה כזו להתגבר על טעויות (בעיית the-compounding error).

מהנארם לעיל נובע כי שיטה זו מתאימה למצלבים בהם מרחב המצלבים קטן ופשטוט יחסית, וכאשר המשימה של הסוכן היא פשוטה ומוגדרת היטב. בנוסף, שיטה זו מתאימה כאשר כמות ה-data אינה גדולה. דוגמאות למשימות אשר מתאימות ל-*apprenticeship learning*: רובוט המוצב בפועל ונדרש להרים חפצים מעלה, או למידה של משחק שחמט.

לעתה זאת, בשיטות Imitation learning מורכבות יותר, אנו מניחים כי המודל של הסביבה מורכב, ולא מתאפיינים ללמידה את הדינמיקה של העולם. שיטות אלו מתרכזות בלמידה של הפעולות עצמן אשר נדרש הסוכן לבצע, הן כלויות יותר ומתחייבות בעיות מורכבות יותר ומוגדרות פחות. מכיוון שאנו לא מסתמכים על מידע של מודל הסביבה, הסוכן יכול "להכליל" למצלבים דומים לאלה שראה, ויכול להתגבר על טעויות. מצד שני, שיטות אלו הן בדרך כלל קשות יותר ללמידה וההתקנסות לוקחת יותר זמן, וכן כאשר ניתן — ננסה להשתמש בשיטת Apprenticeship Learning.

Behavioral Cloning

Behavioral Cloning היא שיטה בה מתבצעת **למידה ישירה של ההתנהגות של הסוכן** מהדגימות שנאספו על ידי המומחה. היתרון בשיטה זו היא העילوت שלה והפשטותהcola — לומדים ישירות לבצע את אותן הפעולות כמו המומחה. שיטה זו יכולה גם להוות בסיס טוב להמשך למידה בשיטות אחרות. כמו כן שהחישרונו העיקרי של שיטה זו הוא הקושי להכליל והסכנה של overfitting. חישרונו זה קיים במיוחד כאשר כמות ה-data מוגבלת ולא מספיק מוגנות. בנוסף, קיימת הבעיה של שגיאות מצלברות (compounding errors), שכן שגיאה אחת עלולה להביא את הסוכן לאזורים למרחב המצלבים אותם הוא לא מכיר. שיטה זו יכולה להתאים למרקם בהם המשימה מוגדרת היטב, הפעולות של המומחה קונסיסטנטיות ומיצגות את מרחב המצלבים אותו הסוכן יפגוש. שיטה זו יכולה להתאים למשל לנήיגה אוטונומית בסימולציה או משחק מחשב, רובוט אשר נדרש לבצע משימה פשוטה או משימות אחרות למרחב מצלבים קטנים וחסית עם משימות יחסית מוגדרות היטב.

HAVOC, ראשי תיבות של **behAViOral Cloning**, היא שיטה ללמידה היררכית באמצעות Behavioral Cloning. בשיטה זו לומדים מספר סוכנים לחכות מספר שחוקנים. למשל, בהינתן dataset של מספר שחוקני פוקר, מאמנים סוכנים כמספר השחקנים כך שכל סוכן לומד משחקן אנושי אחר. למידה זאת נעשית על ידי behavioral cloning. בשלב הבא, מאומן סוכן-על (master), אשר יחליט באיזה סוכן להשתמש בכל שלב. HAVOC היא דוגמה טובה למצב בו ניתן להשתמש בצורה ייעילה ב-*Behavioral cloning* למטרות המגבילות שלה.

Forward Training

שיטת forward training מתמודדת עם מצב בו נדרש **פוליסיה שאינה סטציונית**, למשל כשהסביבה משתנה עם הזמן. הרעיון המרכזי בשיטה זו הוא שבסלב 1 – ? נאספות דוגמאות בעזרת פוליסיה π^{i-1} , אשר נשלחות

لتיאוג, ובעזרתן מאמנת פוליסת π . שיטה זו מتبוצעת בצורה איטרטיבית, כך שבכל שלב מאמנן אלגוריתם חדש על דגימות שנאספו מהאלגוריתם שאומן בשלב הקודם. שיטה זו מתאימה למבנים בהם נדרשת פוליסת אשר משתנה עם הזמן. דוגמאות לכך: רובוט שנדרש לבצע משימה בסביבה משתנה (להרים חפים, כאשר יוכלים להימצא חפצים שונים על גבי המשטח), או סוכן שמשחק משחק בעל מספר שלבים או מבנים. במקרים כאלה, נדרשת התאמת בכל פעם שהסביבה משתנה, ונitinן לבצע את ההתאמת על ידי דגימה של מבנים מהסביבה באמצעות הפוליסת שאומנה בסביבה הקודמת. היתרונות בשיטה זו הוא שהאלגוריתם מאמן על מבנים ש"ניצפו לאחרונה", כלומר שהסוכן יראה ב-test time. חיסרונו מרכז לشيخו פוגש בעת במציאות. לכן data -ההסוכן מאמן עליו מתאים לו שהוא יראה ב- test time. השיסוון מרכז לשיטה הוא שככל איטרציה של אימונו π נדרש להעיבר את הסוכן את כל הצעדים עד T כדי שנוכל לתיאוג אותם ולאמן מחדש עליהם. במצב בו T גדול, זה הופך ללא פרקטטי.

DAgger

שיטה נוספת ל- π -polisfest היא *imitation learning* (*Dataset Aggregation*) *DAgger*. בשונה מ-*forward training*, *DAgger* הוא שיטות פוליסת סטציונרית, כלומר שמנicha שאין שינוי של הסביבה עם הזמן. שיטה זו מאמנת דומה לשיטה הקודמת, ההבדל הוא שהאלגוריתם מותאם על כל ה- data המתויג, גם זה שנאסף בעבר בעורת פוליסות קודמות. שיטה זו בדרכּ כלל מביאה להתקנסות מהירה של האלגוריתם. *DAgger* יכול להתאים במצב שבו הפעולות של המומחה האנושי קוניסטנטיות, גם כאשר הבעייה יחסית מורכבת. עם זאת, במקרים בהם קיימים מגוון גדול מדי של מבנים האלגוריתם יתקשה להצליח. עוד חיסרונו באלגוריתם זה הוא, שכמו *forward training*, המומחה נדרשת לתיאוג מחדש בכל איטרציה את הפעולות הנדרשות למסלול שהסוכן יצר. דבר זה עלול להקשות בעיות מוחולם האמתי.

אתגר נוסף בשיטה זו הוא שבמקרים בהם הפוליסת שהסוכן צריך למדוד מרכיבת מיידי, הוא יתנסה ללמידה ישירות את הפוליסת המורכבת רק מצפיה בהתבוננות במומחה. באנווגיה לכך, תלמיד לא יוכל ללמידה לנגן כינורisher מנגן בינהומי, הוא יצטרך למדוד בצורה הדרגתית. אלגוריתם *Dagger with Coaching* בא לפטור בעיה זאת. בשיטת *DAgger with Coaching*, הסוכן מקבל בתחלת האימון דוגמאות יותר קלות, וככל שעובר הזמן, ככל שהוא מסתפר, הוא מקבל דוגמאות מתוגרות יותר. מוגדר *hope action*:

$$\tilde{\pi}_i(s) = \arg \max_{a \in A} (\lambda_i \cdot score_{\pi_i}(s, a) - L(s, a))$$

אשר קובע מהי הפעולה אותה הסוכן יבצע. $L(s, a)$ הוא *loss* של המומחה (כמו באלגוריתם הבסיסי), אשר מתיוסף עוד ביטוי (a, s) אשר מכמתת כמה סיכויי יש להפוליסת הנוכחית תבחר את a במצב s . ככל שהסיכויי נמוך יותר, כך הפעולה כנראה יותר "מורכבת להבנה" לסוכן בשלב זה ולכן יש פחות סיכוי שנבחר בה. ככל שהלמידה התקדם, כך λ_i יקטן (עד שיגיע ל-0), ווינוון משקל גובה יותר לפעולות של המומחה. להלן שני אלגוריתמיים:

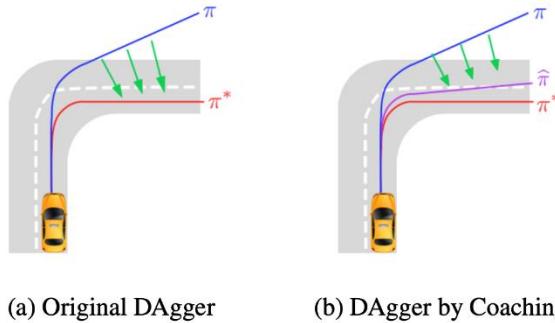
Algorithm 5 DAgger and DAgger by coaching algorithms

```

1: Initialize  $D \leftarrow \emptyset$ ,  $\pi_1 \leftarrow \pi^*$ 
2: for  $i = 1 : N$  do:
3:   Sample T-step trajectories using  $\pi_i$ 
4:   if coaching then
5:      $\pi_{target} = \tilde{\pi}_i$  (Hope Action)
6:   else
7:      $\pi_{target} = \pi_i^*$  (Expert Action)
8:   end if
9:   Collect  $D_i = \{(s_{\pi_i}, \pi_{target}(s_{\pi_i}))\}$  dataset of visited states by  $\pi_i$  and actions by expert/coach
10:  Aggregate datasets  $D \leftarrow D \cup D_i$ 
11:  Train policy  $\pi_{i+1}$  on  $D$ 
12: end for
13: return best  $\pi_i$  on validation set

```

להלן אילוסטרציה של ההבדל בין האלגוריתמים. כאשר באלגוריתם המקורי יכול להיות הבדל גדול בין הפעולה שהסוכן רצה לבצע π (הפרדייקציה) לבין π^* , באלגוריתם *DAgger with Coaching* נבחר פעולה שפחות "מדוייקת" אבל קרובה יותר לפרדייקציה של הסוכן בשלב זה.



(a) Original DAgger

(b) DAgger by Coaching

Multi-arm bandits

Multi-arm bandits היא בעיה בסיסית מאוד של RL . בעיה זו ישנו רק מצב (*state*) אחד, עם K פעולות אפשריות. ה-*reward* המתקבל מכל פעולה מתפלג בהתאם לשאינה ידועה. בכל שלב, הסוכן בוחר פעולה אחת מתוך ה- K ומקבל עליה *reward*. השאלה היא מה הפולישה שתבטיח את ה-*reward* הגבוהה ביותר ליותר זמן. זהה בעיה מזוקקת של המתח בין *exploitation* ו-*exploration*.

אלגוריתם בסיסי יכול להיות ϵ -greedy, אשר בהסתברות ϵ – 1 בוחר את הפעולה שננתנה לנו את הגמול המומוצע הכי גבוה עד כה, ובಹסתברות ϵ בוחר פעולה אקראית. במצב כזה, רוב הזמן נבחרת הפעולה שאנו מאמינים שהיא אופטימלית, ומצד שני נשמרת האקספלורציה. להלן האלגוריתם:

A simple bandit algorithm

```

Initialize, for  $a = 1$  to  $k$ :
 $Q(a) \leftarrow 0$ 
 $N(a) \leftarrow 0$ 

Repeat forever:
 $A \leftarrow \begin{cases} \arg \max_a Q(a) & \text{with probability } 1 - \epsilon \\ \text{a random action} & \text{with probability } \epsilon \end{cases}$  (breaking ties randomly)
 $R \leftarrow \text{bandit}(A)$ 
 $N(A) \leftarrow N(A) + 1$ 
 $Q(A) \leftarrow Q(A) + \frac{1}{N(A)} [R - Q(A)]$ 
```

אלגוריתם זה מתאים למקרה הסטטצוני, בו התפלגות ה-*rewards* לא משתנות עם הזמן. במקרה בו ההתפלגות לא סטטצונית נעדיף לתת משקל גדול יותר לדגימות חדשות מאשר לישנות. דרך נפוצה לקבל תכונה זאת היא על ידי *constant step-size parameter* :

$$Q_{n+1} = Q_n + \alpha [R_n - Q_n]$$

Q_n הוא ממוצע של n ה-*rewards* האחרונים. הפרמטר α הוא היפר-פרמטר השולט כמה מהר אנחנו "שוכחים" את ההיסטוריה. ככל שהוא קרוב ל-0 כך אנחנו מתקרבים לממוצע פשוט, וככל שהוא קרוב ל-1 מתחשים אך ורק בהיסטוריה הקרובה.

הבעיות בשיטות ϵ -greedy הן:

- השיטה לא אלגנטית, מתבצעת חלוקה מלאכותית בין זמן ה-*exploitation* ל-*exploration*.
- האקספלורציה לא נעשית בצורה אופטימלית, יתכן ויש מצבים שברור לנו שאין צורך באקספלורציה ובכל זאת מבצעים אותם בהסתברות שווה.

פתרון אפשרי הוא לעשות אקספלורציה של המצביעים בהם אנחנו פחות בטוחים.

Upper Confidence Bound (UCB)

בשיטת זו מתחשבים בזמן בחירת הפעולה גם במספר הפעמים שהפעולה ננקטה. הביטוי לבחירה הפעולה הוא:

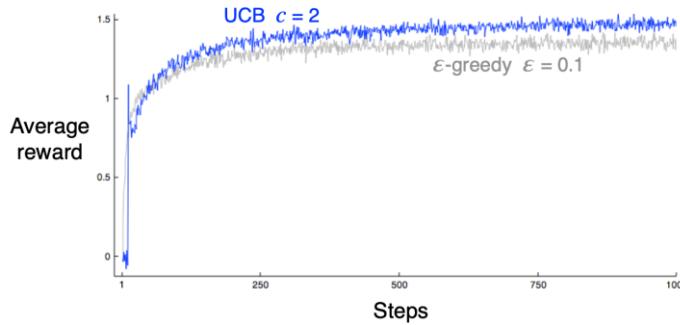
$$A_t = \arg \max \left(Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right)$$

אינטרואיציה: הביטוי השמאלי ($Q_t(a)$) אחראי על ה-*exploitation*, והשמאלי $c \sqrt{\frac{\ln t}{N_t(a)}}$ על האקספלורציה.

הוא אחוז הפעמים שביקרנו במצב a . מטרתנו היא שככל שביקרנו יותר במצב מסוים כך נברך בו פחות

בזמן הקרוב, ולכן יהיה מתאים להשתמש בהפכי: $\frac{t}{N_t(a)}$. עם זאת, מכיוון שההנחה היא שעם הזמן אנחנו "מכירים" יותר את הסביבה ואנחנו פחות מעוניינים באקספלורציה, נהפוך את המונה מלינארית ללוגריתמי $t \rightarrow \ln t$.

חישרונו בשיטה זו הוא ההתמודדות עם התפלגויות לא סטציאונריות. מכיוון שהቢיטוי $c \sqrt{\frac{\ln t}{N_t(a)}}$ לא מתחשב בשינוי בתפלגויות, במצב בו הן משתנות האקספלורציה לא תהיה אופטימלית. חישרונו נוסך בשיטה זו הוא הקושי בהתמודדות מול מוחבי מצבים גדולים. במצבים כאלה קשה מאוד לבצע את האקספלורציה בשיטת UCB . להלן גורף המשווה את UCB ל- ϵ -greedy:



Gradient Bandits Algorithm

שיטה זו הינה **בסיסת גודיאנטים** (*gradient descent* מותבצע *rewards*). השיטה מבצעת אופטימיזציה על הסתברויות לבחירת פעולה מתוך K על בסיס ה-*rewards* המתקיים. הסוכן בוחר בכל איטרציה פועלה על בסיס הסתברויות:

$$Pr(A_t = a) = \frac{e^{H_t(a)}}{\sum_{b=1}^k e^{H_t(b)}} = \pi_t(a)$$

ולאחר מכן ממבצע עדכון ל- $H_t(a)$ באמצעות ה-*reward* שהתקבל. העדכון מ被执行 באמצעות:

$$H_{t+1}(a) = H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a)$$

LinUCB

האלגוריתמים שהזכרנו עד כה מתיחסים רק להיסטורית ה-*rewards* על מנת לבחור פעולה. במקרים בהם קיימים מספר מצבים (*states*) נרצה גם להתייחס למצב, שכן הוא יכול להשפיע על ה-*reward* שיתקבל. מתייחס לSTITואציה בה המצב משפיע על הפרדייקציה. במצב כללי של *RL*, גם הפעולה יכולה להשפיע על המצב, אך לעת-Anchenko מניחים שאין השפעה של הפעולה על הסביבה, אלא רק על ה-*reward* שיתקבל. מבחינת טרמינולוגיה, מקום להשתמש במונח **מצב** (*state*) משתמשים במונח **משתמש** (*user*). כל משתמש מיוצג באמצעות וקטור מאפיינים (*feature vector*).

אלגוריתם *LinUCB* בא לפטור את הבעיה של *contextual bandit*. באlgorigthm זה, תוחלת ה-*rewards* על כל זרווע מודדת באמצעות פונקציה לינארית על וקטור המאפיינים של המשתמש:

$$E[r_{t,a}|x_{t,a}] = [x_{t,a}]^T \theta_a^*$$

ניתן להציג פתרון אנלטי למציאת θ_a :

$$\hat{\theta}_a = (D_a^T D_a + I_d)^{-1} D_a^T b_a$$

כאשר D_a היא מטריצה בגודל $d \times m$ (מספר הדגימות) ו- b_a הוא וקטור התוצאות (the-reward). להלן האלגוריתם :

Algorithm 1 LinUCB with disjoint linear models.

```

0: Inputs:  $c_t \in \mathbb{R}_+$ 
1: for  $t = 1, 2, 3, \dots, T$  do
2:   Observe features of all arms  $a \in \mathcal{A}_t$ :  $\mathbf{x}_{t,a} \in \mathbb{R}^d$ 
3:   for all  $a \in \mathcal{A}_t$  do
4:     if  $a$  is new then
5:        $\mathbf{A}_a \leftarrow \mathbf{I}_d$  ( $d$ -dimensional identity matrix)
6:        $\mathbf{b}_a \leftarrow \mathbf{0}_{d \times 1}$  ( $d$ -dimensional zero vector)
7:     end if
8:      $\hat{\theta}_a \leftarrow \mathbf{A}_a^{-1} \mathbf{b}_a$ 
9:      $p_{t,a} \leftarrow \hat{\theta}_a^\top \mathbf{x}_{t,a} + c_t \sqrt{\mathbf{x}_{t,a}^\top \mathbf{A}_a^{-1} \mathbf{x}_{t,a}}$ 
10:   end for
11:   Choose arm  $a_t = \arg \max_{a \in \mathcal{A}_t} p_{t,a}$  with ties broken arbitrarily, and observe a real-valued payoff  $r_t$ 
12:    $\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_{t,a_t} \mathbf{x}_{t,a_t}^\top$ 
13:    $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t \mathbf{x}_{t,a_t}$ 
14: end for

```

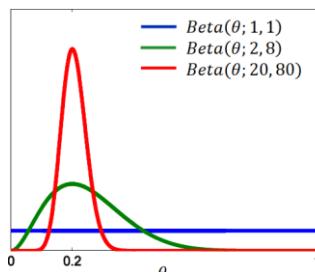
בחינת סיבוכיות, האלגוריתם לינארי במספר הזרועות (או המשתמשים) K , וריבועי במספר המאפיינים (features). האלגוריתם מתאים במקרה שה"זרועות" לא קבועות, אלא באות בזמן מסוים ואז נעלמות (למשל במערכת המלצה לכתבות של חדשות).

Thompson Sampling

אלגוריתם זה מניח כי התפלגויות $rewards$ לא משתנות. הוא מתבסס על דוגמה בייסיאנית, כלומר בחירה בזרוע בעלת אי-חוודאות הגבוהה ביותר. האלגוריתם משערך פונקציית התפלגות עבור כל זרוע ומתבסס עליה בדגם. שלבי האלגוריתם הם :

- דוגמה של משתנה אקראי X עבור כל אחת מ- K הזרועות על בסיס פונקציית ההתפלגות שנלמדה
- בחירה של הזרוע עם X הגבוה ביותר
- שמירה של $reward$ שהתקבל ממנו
- עדכון פונקציית ההתפלגות המשוערת עבור זרוע זאת

תהליך זה מתבצע בכל צעד t . עבור $\theta \in \{0, 1\}$ ניתן להשתמש בפונקציית $Beta(\alpha, \beta)$. לפונקציה זאת התכוונה שהתחולפת שלה היא $\frac{\alpha}{\alpha+\beta}$ והשונות שלה קטנה ככל שערכי α - β , גודלים. אם נגידר את α כמספר הפעם שהתקבל $1 = \theta$ ו- β כמספר הפעם שהתקבל $0 = \bar{\theta}$, פונקציה זו מתאימה למצב שלנו בו נרצה שבתחלת התהליך תהיה יותר אקספלורציה, וככל שמתקרם התהליך (β, α) נתכנס למומוץ.



להלן תיאור האלגוריתם :

Algorithm 1 Thompson Sampling for Bernoulli bandits

For each arm $i = 1, \dots, N$ set $S_i = 0, F_i = 0$.

foreach $t = 1, 2, \dots$, **do**

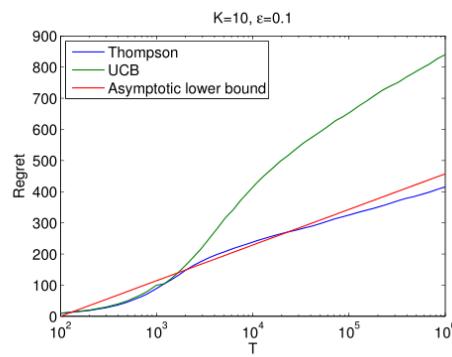
 For each arm $i = 1, \dots, N$, sample $\theta_i(t)$ from the Beta($S_i + 1, F_i + 1$) distribution.

 Play arm $i(t) := \arg \max_i \theta_i(t)$ and observe reward r_t .

 If $r = 1$, then $S_{i(t)} = S_{i(t)} + 1$, else $F_{i(t)} = F_{i(t)} + 1$.

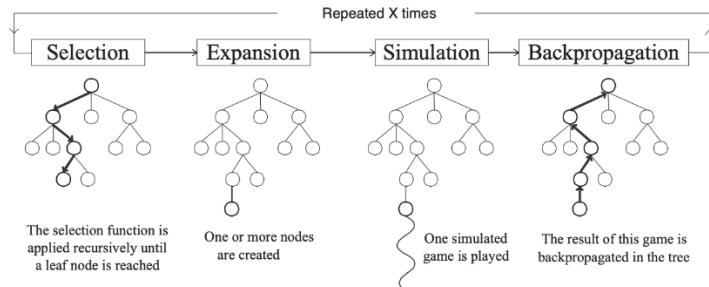
end

להלן גרף של regret כפונקציה של הזמן. יש לשים לב שהציר האופקי בסקלת לוגריתמית :



Monte Carlo Tree Search

אלגוריתם *MCTS* משלב בתוכו חישוב בעז ודגימת *Monte Carlo* כדי לבצע אקספלורציה של סביבה וללמוד מהי הפעולה האופטימלית לbijoux בכל שלב על מנת למקסם את הרוחחים לאורך זמן. האלגוריתם מורכב מארבעה שלבים שמתבצעים בזורה מחזורית. להלן סכמה של השלבים:



העץ מייצג את הסביבה, כאשר כל צומת בעץ מייצגת מצב (*state*), וכל קשת מייצגת פעולה (*action*). להלן נסקור את השלבים.

שלבי האלגוריתם

Selection

בשלב זה מותבצעת בחירות הצומת שמננה האלגוריתם ימשיך. בתחילת האלגוריתם קיימת רק הצומת הראשונה (*root*), ולאחר מספר צעדים "נפתחו" צמתים נוספים שאוון אפשר לבחור. הבחירה נעשית על ידי חישוב בעז (זהו חלק *the UCB* שבסוגיות *Tree Search*) החישוב יכול להתבצע למשל על ידי *UCB*, אשר משלב :

exploitation ו-*exploration*

$$\pi_{UCB}(s) = \arg \max_a Q(s, a) + c \sqrt{\frac{\ln N(s)}{N(s, a)}}$$

כאשר (s) הוא מספר הפעמים שביקרנו במצב s ו- (a) $N(s, a)$ הוא מספר הפעמים שבחרנו ב פעולה a במצב s . באמצעות אלגוריתם זה ניתן לבחור את הצומת המתאימה ביותר להמשך התהליך.

Expansion

השלב השני באלגוריתם הוא *expansion*. אם המצב שבחרנו הוא עליה (*terminal stats*), אז ניתן להמשיך לשלב האחרון באיטרציה (*backpropagation*). אחרת, נבצע הרחבת העץ למצבים חדשים (*rollout policy*). נאתחל את כל הממצבים האפשריים הבאים ונבחר מצב אחד על פי פוליסה (*policy*).

Simulation

cut מגיע שלב *simulation*. בשלב זה אנחנו מסמלצים משחק (או מספר משחקים אשר ניתן למקבל) החל מהשלב הנבחר ועד לשלב הסופי. במהלך המשחק הפעולות יכולות להיות אקראיות או באמצעות *rollout policy*. כך אנחנו אוספים את *rewards* מהשלב הנבחר והילך.

Backpropagation

cut נדרש רק לעדכן את (a) $Q(s, a)$ באמצעות *reward* שהתקבל. ניתן לבצע את העדכן בעזרת ערך ממוצע הנקודות (מול ההפסדים), או לחילוף באמצעות *gradient descent* לעדכן באמצעות ההפרש בין הערך המוצופה ליה שהתקבל.

Tכונות MCTS

- מכיוון ש-*MCTS* מtabsts רק על סימולציה, לא נדרש *domain-specific knowledge* ולא נדרש לתמוך. הדבר היחיד שנדרש הוא ידיעת "חוקי העולם" על מנת לבצע את הסימולציה.

- האלגוריתם הינו *any-time*, במובן שניתן לעצור אותו בכל שלב במהלך האימון ול לקבל את הביצועים הטובים ביותר שהתקבלו עד כה.
 - ניתן לראות אמפירית כי קשה לאלגוריתם לבצע "הקרבות", למשל הקרבה של מלכה בשחמט כדי לנצח את המשחק.

פתרונות ל-MCTS

שני שיפורים אפשריים לאלגוריתם :

- שילוב פוליסיה (אין הכוונה ל-*policy* במובנו הרגילים-*RL*)
 - שילוב *value function*

שילוב פוליסיה : במערכת בשם *Crazy Stone GO*, נעשה שימוש בפוליסיה על מנת לבצע גיזום (*pruning*) לעץ. במערכת זו מחושבים מאפיינים (היוריסטיים) לכל צומת. לאחר שהתבצעו מספר מסויים של סימולציות, הפוליסיה קובעת, על בסיס המאפיינים שלעיל, האם צומת זו זוכה להישאר בעץ או שהיא "ינזמת" מהעץ. שיטה זו מאפשרת להישאר עם עצ מוצמצם יותר המכיל רק את הצלמיים הרלוונטיים לomidah.

שילוב *value function*: ניתן להוסיף *value function* שיעשה בה שימוש במהלך *MCTS*. למשל, מחקר מסוים הראה שימוש בקומבינציה לינארית של מאפיינים של מצב על מנת לקרב את *value function* שלו:

$$Q_{RLGO}(s, a) = \sigma(\phi(s, a)^T \theta)$$

כאשר (a) ϕ הוא וקטור מאפיינים ביניים (*hand-crafted*) ו- θ הוא וקטור משקלות נלמד. σ מרגמת את הערכים בין 0 ל-1. עם פונקציה כזו ניתן לבצע דגימה חכמה יותר של מסלולים בשלב הסימולציה. למשל:

- ניתן לביצוע סימולציות באמצעות-police של ϵ -greedy בהתבסס על $Q_{RLGO}(s, a)$.
 - ניתן להשתמש ב- $Q_{RLGO}(s, a)$ בתוספת רעש ($\eta(s, a) + \epsilon(s, a)$) ולבחר בעולה בצורה חמדנית. בזורה כזאת $Q_{RLGO}(s, a)$ אחראי על ה- exploration ו- $\eta(s, a)$ על ה- exploitation .
 - בחירה של בעולה באמצעות פונקציית ההתפלגות:

$$\frac{e^{Q_{RLGO}(s,a)/\tau}}{\sum_{a'} e^{Q_{RLGO}(s,a')/\tau}}$$

ההטפלוגות קרובות יותר ל-one-hot. ערכיהם גבוהים שלו יתנו פונקציה הקרויה להטפלוגות איחודית, וערך נמוך יהפוך את SoftMax על ערכי (a, s) . הפקטור τ הוא פרמטר טמפרטורה האחראי על הבחירה פונקציית SoftMax.

לסיכום, שימוש ב-*value function* מאפשר דוגימה חכמה יותר בשלב הסימולציה, כך ש-*MCTS* יוכל
מדגים רלוונטיות יותר.

AlphaGO

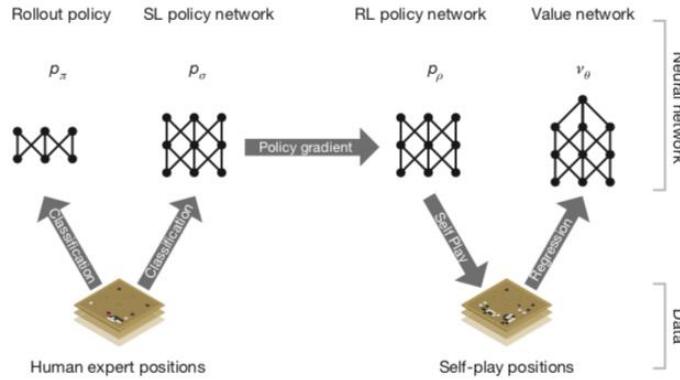
נלמד בקורס רקורסיביות עלייהם נוכל ללווד *value function* אופטימלית. בשחמט $d \approx 35$, $b \approx 80$ ו-*GO* $d \approx 80$ ו- $b \approx 250$. לכן, לא ריאלי ללמד על כל המציבים האפשריים, לפחות תחת מגבלות החישוב הקיימות כיום.

הארQUITטורה של *AlphaGO* מביית *DeepMind* מורכבת משלוש רשותות:

(dataset מה-*Supervised learning policy network*) •

- (self-play *RL policy network*)
- (self-play *RL value network*)

בנוסף, געשה שימוש במודל לינארי עם *SoftMax* בשלב הסימולציה (גולם מה-*dataset*).).



מרחב הכניסה (input)

הבעיה ממודלת כמטריצה בגודל 19×19 (גודל הלוח) עם עומק כמספר המאפיינים. המאפיינים מוגדרים על ידי הטבלה הבאה עבור כל אחד מהמשבצות של הלוח:

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Supervised learning policy network

רשת זו (*CNN*) אומנה על *dataset* של משחקים מומחין, אשר מורכב מ- $160K$ משחקים ל- $29.4M$ מצבים. בנוסף, הzbיצע *data augmentation* של סיבובים של הלוח. הרשת חוזה מה המהלך הבא בהינתן מצב הלוח. האימון הzbיצע במשך שלושה שבועות על 50 *GPs*.

RL policy network

בשלב הבא, רשת-*SL* **המשיכת אימון** באמצעות *RL* – משחק מול עצמה. היריב מולו הרשת משחקת הוא רשת שנבחרת אקראית מתוך קבוצה של רשותות כללה שאומנו בעבר. לאחר כל 500 איטרציות, הרשת החדשה מצטרפת גם היא לקובצת הרשותות הנ"ל. האימון מתבצע על ידי אלגוריתם *REONFORCE*, כאשר בכל איטרציה משחקים משחק אחד עד הסוף, ואז עוברים על כל המהלךים ומעדכנים את ה-*RL policy network*. צעד העדכון מתבצע באמצעות הכלל:

$$\Delta p = \frac{\alpha}{n} \sum_{i=1}^n \sum_{t=1}^T \frac{\partial \log p_p(a_t^i | s_t^i)}{\partial p} (z_t^i - v(s_t^i))$$

כאשר (s_t^i, a_t^i) מאותחל בהתחלה לאפס, ולאחר מכן משתמשים ברשת ה-*value* כדי להוריד *variance*. בשלב זה האימון הzbיצע במשך יום אחד על 50 *GPs* עם $10,000$ *mini-batches* של 128 משחקים.

RL value network

בשלב זה מאומנת רשת *value* אשר חוצה עבור הפוליסה הקיימת האם מצב s יוביל לניצחון. ארכיטקטורת הרשת זהה לזו של ה-*policy network*, בשינוי אחד – ישנו ערך אחד בלבד בmozac (במקום ערך לכל פעולה). מtbody *MSE* בין מוצרת הרשת לבין תוצאה המשחק. ה-*dataset* מרכיב מוגדר אחד בלבד מכל משחק, על מנת ליצור *dataset* בלתי תלוי.

Rollout Policy

ה-*rollout* הוא הפוליסה שבאמצעותה דוגמים את הסימולציות. המטריה היא ליצור אלגוריתם מהיר, שיכל לבצע את הסימולציה בזרחה מהירה. נעשה שימוש במסוג לינארי, אשר מאומן על מאפיינים שמוגדרים מראש. להלן המאפיינים:

Feature	# of patterns	Description
Response	1	Whether move matches one or more response pattern features
Save atari	1	Move saves stone(s) from capture
Neighbour	8	Move is 8-connected to previous move
Nakade	8192	Move matches a <i>nakade</i> pattern at captured stone
Response pattern	32207	Move matches 12-point diamond pattern near previous move
Non-response pattern	69338	Move matches 3×3 pattern around move
Self-atari	1	Move allows stones to be captured
Last move distance	34	Manhattan distance to previous two moves
Non-response pattern	32207	Move matches 12-point diamond pattern centred around move

Features used by the rollout policy (first set) and tree policy (first and second set). Patterns are based on stone colour (black/white/empty) and liberties (1, 2, ≥ 3) at each intersection of the pattern.

המאפיינים מוגדרים ממטריצה ביןרית מסביב לכל פעולה חוקית (אשר מסמנת האם המיקומות תפוסים), סטטיסטיות של מצב הלוח ומאפיינים שקשורים למשחק *GO* עצמו. המסוג אומן על מיליון דגימות מתוך *dataset* אינטרנטני.

תהליכי האימון

זכור, האלגוריתם מתחילה בבחירה של מצב שמננו להתחילה – שלב *selection*. האלגוריתם מתחילה מה-*root* ו"מיטיל" לעומק העז עד שהוא בוחר בעלה. החיפוש נעש**ה**a באמצעות ה-*policy network* עם אקספלורציה. ככל שעובר הזמן, נועשית פחות אקספלורציה.

לאחר שנבחר עלה מסוימת, מtbody *selection* מtbody *value network* ותוצאה המשחק:

$$V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda Z_L$$

תהליכי הסימולציה מtbody *selection* מספר פעמיים, וכך אשר מספר זה חזקה רף – העלה מתווסף לעצ.

AlphaGoZero

ה"דור הבא" של *AlphaGO* הוא *AlphaGoZero*. השינוי המרכזי שנעשה הוא **שהאימון הtbody *selection* באמצעות משחק עצמי בלבד, ולא *supervised learning*.** אף על פי שיטה זאת היא לרוב לא יציבה, אלגוריתם ה-*MCTS* מצליח להתגבר על כך. עוד הבדן מרכזי הוא שלא הtbody *selection* של *input*, *feature engineering* של האלגוריתם הוא הלוח בלבד ולוחות עבר. ספציפית, האלגוריתם מקבל את המצב הנוכחי ואת שבעת המצביעים הקודמים, כולל את פועלות היריב (סה"כ 16 מצבים), עםnoch נספ שmagdir תורו של מי CUT (סה"כ 17 לוחות). בנוסף, נעשה שימוש ברשת אחת בלבד, וחיפוש פשוט יותר בעז, ללא *MC rollout*.

שלושה שלבים באימון שtbody *selection* במקביל:

- על מנת ליצור *data*. בשלב זה, הרשת הטובה ביותר שאותה עד כה משחקת 25,000 משחקים נגד עצמה. הנתונים שנשמרים: מצב הלוח, הסתברויות החיפוש שמתקבלות מ-*MCTS* ותוצאה המשחק (ניצחון או הפסד).

- אופטימיזציה של הרשת. נדגמים 2,048 מצבים מתוך 50,000 המצבים האחרונים. ה-*input* לרשת הוא המצב האחרוניים (כפי שתואר לעיל), וה-*loss function* *policy loss* מורכב מ-*policy loss* אשר מעדכן את *value loss* אשר מעדכן את החיזוי של *value* של המצב הנוכחי.
 - אבולוציה של הרשת – מtbody 400 משחקים של הרשת המעודכנת עם הרשת הקודמת. אם הרשת החדשה ניצחה יותר מ-55% מהפעמים – היא מוכתרת למנצחת החדש. במהלך המשחק, כל סוכן מtabus על הרשת שלו ו מבצע *MCTS* כדי לבחור את המהלך שלו.
 - הארכיטקטורה של הרשת מורכבת מ-40 *residual blocks* עם שני ראשיים – אחד ל-*policy* ואחד ל-*value*. *MCTS* משתמש גם ל-*policy improvement* *policy evaluation*. כל מצב (קודוקוד) מכיל את ה-*value* של המצב, וכל פעולה (קשת) מכילה את ההסתברות להגעה אליה $P(s, a)$, את מספר הפעמים שביקרו בה $N(s, a)$ ואת ה-*action value* שלה $Q(s, a)$
- בחירת העלה מתבצעת על ידי הכלל הבא :
- $$a_t = \arg \max_a (Q(s_t, a) + U(s_t, a))$$
- כאשר $U(s_t, a)$ אחראי על האקספלורציה (הוא ה-*upper confidence bound*) והוא $C_{puct} P(s, a) \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)}$
- הוא היפר-פרמטר אשר שולט באיזון בין ה-*exploitation* *exploration* *output* $P(s, a)$ של *policy network* (ההסתברות לבצע פעולה a במצב s), והוא אחראי לבחור פעולות עם פוטנציאלי גבוה מtbody הפעולות שפחוות נבחרו. הביטוי האחרון הוא זה שהוביל לאקספלורציה במצבים שבהם ממוצע הביקורים נמוך יותר, כאשר ה-1 במכנה אחראי לכך שהbeitioי לא יגדל יותר מיד.
- לאחר הבחירה בעלה, מתחלים את כל הפעולות האפשריות (מאתחלים את כל ה-*counters* ל-0), בוחרים פעולה אחת, ומעריכים אותה באמצעות $Q(s_t, a)$. ב-*AlphaGoZero* לא מבצעים אבולוציה באמצעות *rollout policy*, אלא עוצרים בשלב זה.
- בשלב הבא – *backup* – מעדכנים את העז :
- $$\begin{aligned} N &\leftarrow N + 1 \\ W &\leftarrow W + v \\ Q &\leftarrow \frac{W}{N} \end{aligned}$$
- לאחר 1,600 סימולציות, מתבצע השלב הבא – בחירת פעולה מתבצעת *training* בבחירה הפעולה מתבצעת באמצעות $\pi(a|s_0) = \frac{N(s_0, a)^{\frac{1}{\tau}}}{\sum_b N(s_0, b)^{\frac{1}{\tau}}}$, ב-*test time* נבחרת הפעולה עם ה- N הגבוה ביותר. לאחר שנבחרה פעולה. המצביע החדש הופך ל-*root*, כל המצבים שאינם בנויים שלו מוסרים מהעץ.
- במהלך האימון, הרשת משחקת נגד גרסה ישנה שלה (הטובה ביותר). נשמרים 500,000 פעולות שמהם ה-*MCTS* דוגם. כל משחק מתבצע עד הסוף כאשר מתקבל בסוף $\{z, \pi\} \in \{1, 1\}$. כל המצבים נשמרים ונדגם אחד בצוות אקרטיות (יוניפורמיות) ובו נעשה שימוש לעדכון הרשת. כאמור, ה-*value loss* מורכב מ-*MSE* על ה-*value*, ומ-*cross entropy* על הפעולות :
- $$l = (z - v)^2 - \pi^T \log(p) + c \|\theta\|^2$$
- כאשר הרשת מנצחת יותר מ-55% מהמשחקים – היא מוכתרת לטובה ביותר.

ההידושים ב-*AlphaGO* שלא היו ב-*AlphaGoZero* :

- לא נעשה שימוש ב-*rollout*
- שימוש בראשת יחידה
- כאשר מגיעים לעלה בעז – פותחים את כל הפעולות, ומעריכים את המוצבים החדשניים באמצעות הרשת (ולא סימולציה)
- אין *Tree policy*
- – ללא *data* ממשחקים אמייתיים
- אין רק מוצבי הלוחות האחרונים *hand crafted features*

AlphaGO בתחוםים אחרים

לאחר שפורסם *AlphaGO* נעשו ניסיונות להעביר את הרעיון שбалוגריטם לתחומיים אחרים. אחד האלגוריתמים הוא *AutoML* – *AlphaD3M* *ML pipeline* (בעובדה זאת, ה-*AutoML* מודול *ML* בdataset, על המטלה ועל ה-*pipeline*). כל פעולה היא שינוי של מודול ב-*pipeline*.

ההבדל מ-*AlphaGO* הוא שהוא משחק עם שחקן אחד, אך חוץ מהעובדת הזו – התהליך מאוד דומה. הרשת :

$$f_{\theta}(s) = (P(s, a), v(s))$$

ממצאים *UCT* לאקספלורציה :

$$U(s, a) = Q(s, a) + cP(s, a) \frac{\sqrt{N(s)}}{1 + N(s, a)}$$

וישנו שינוי קטן בפונקציית *loss* :

$$L(\theta) = SlogR + (v - e)^2 + \alpha \|\theta\|_2 + \beta \|S\|_1$$

השינוי הוא הוספת הרגולרייזציה על אורך ה-*pipeline* באמצעות $\|\cdot\|_1$

שימוש אחר ב-*framework* הזה נעשה לטובת *architecture search*. מבנה של רשת יכול להיבדק בצורה הבאה : [5, ..., "num filters: 24", "filter height: 3", "filter width: 3", ...], ולכן יכול לבחור באיזו צורה לבנות ארכיטקטורה. בעובדה נעשה שימוש ב-*RNN* שיצור את ה-*string* שמתאר את הרשת, לאחר מכן הרשת אומנה על *dataset* ונעשה עדכון לפרמטרים של ה-*controller* על פי ביצועי הרשת (ה-*accuracy*). מכיוון :

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^T E_{P(a_{1:T}; \theta_c)} [\nabla_{\theta_c} logP(a_t | a_{(t-1):1}; \theta_c) R]$$

על ידי דוגמה ניתן להשתמש בכלל הבא (כמו ב-*REINFORCE* הסטנדרטי) :

$$\frac{1}{m} \sum_{k=1}^m \sum_{t=1}^T \nabla_{\theta_c} logP(a_t | a_{(t-1):1}; \theta_c) R_k$$

עוד בעובדה קרובה היא *Deepline*. לפירוט נא לעיין בהמשך בפרק על מרחבי מוצבים ופעולות גדולים.

Meta-learning

הו תחום ב-ML בו מנסים לבצע במידה של רשות אחת או אלגוריתם אחד על מספר מטלות tasks. במקרים רבים נדרש מהאלגוריתם להתאים את עצמו בצורה מהירה למטלות חדשות few/one/zero shot learning). נראה בהמשך שתחום זה משיק מאוד ל-RL מכיוון שבקרה טיפוסית ב-RL מקרים של מטלות רבות שסוכן יכול ללמידה ממקביל (למשל מספר משחקי מחשב, או רובוטים עם משימות מעט שונות), ועם זאת איסוף ה-data יכול להיות מוגדר. בנוסף RL הפעולות בעולם האומתני נפגשים במצבים של שינוי בתתפלגות distribution shift), אשר ניתן להתמודד איתו באמצעות meta learning.

מונחים:

- τ_1, \dots, τ_n רישימת tasks.
- כל τ_i task מאופיין ב- $(s_t, a_t, \mathcal{L}(s_t, a_t))$, הסתברויות מעבר בין מצבים $P(s_{t+1}|s_t, a_t)$ ואורך של H_i - episode.
- $\pi(a_t|s_1 \dots s_t; \theta)$ מודל את ההסתברויות לפעה a_t בהינתן המצבים הקודמים.

ב-meta learning, המטרה היא למצוא את האופטימום :

$$\min_{\theta} \mathbb{E}_{\tau_i \sim \tau} \left[\sum_{t=0}^{H_i} \mathcal{L}(s_t, a_t) \right]$$

where $s_t \sim P_i(s_t|s_{t-1}, a_{t-1})$, $a_t \sim \pi(a_t|s_1 \dots s_t; \theta)$

במילים פשוטות, המטרה היא למצוא מודל שמבצע היבט מסוים מטלות. קיימים מצבים בהם קשה לאסוף data למשימות רבות, ולכן נהיה מעוניינים בראשת אשר יכולת ללמידה היבט מסוים של דוגמאות. במצב שיש לנו משימות מרובות נרצה למצוא ראשית אשר אומנה על data רב יחסית שמורכב ממספר משימות, כך שלאחר שיש מודל ראשון, הוא יוכל בקלות ללמידה חדשה בעזרת data נוספת.

Meta-Learning with Memory-Augmented Networks

מטרת שיטה זו היא לבצע few shot learning באמצעות זיכרון חיצוני לרשות, אשר הרשות יכולה לגשת אליו בזמן הפרזיקציה. הזיכרון מורכב ממטריצה גדולה, אשר כל שורה בה היא וקטור מאפיינים (feature vector). שורה יכולה להיות מאפיינים של דוגמה ספציפית מתוך סט האימון או קומבינציה של מאפיינים ממספר דוגמאות (כפי שיאסביר בהמשך). מטרת הארכיטקטורה היא לשמר במטריצת הזיכרון מאפיינים של דוגמאות מייצגות ביחיד עם אינפורמציה על התיאוג שלהם. בזמן ה-inference, הרשות תדע לגשת למאפיינים דומים מתוך מטריצת הזיכרון ולדעת לחזות את ה-label שלהם באמצעות האינפורמציה שהרשות למדה מתוך אוסף ה-tasks שנלמדו בזמן האימון.

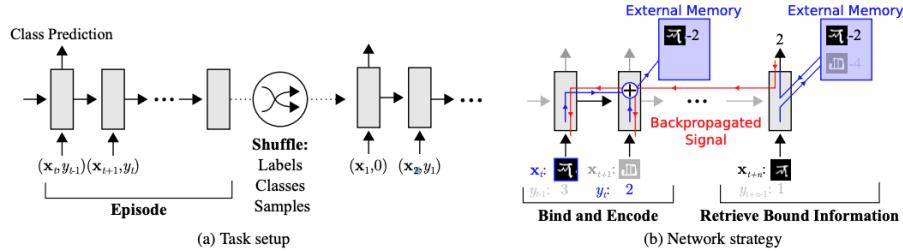
המערכת מורכבת מחלקים הבאים :

- רשות Controller – מקבל את הדגימה בכניסה, מוציאה את הפרזיקציה, ובאינטראקציה עם הרכיבים האחרים (Read head, Write head).
- Read Heads אחראים על קריאה מתוך הזיכרון.
- Write Heads אחראים על כתיבה ועדכון הזיכרון.
- מטריצת זיכרון.

ה-data הוא מהקרה הבאה. מתוך קבוצת הדגימות $\{(x_i, y_i)\}$ אשר נלקחו מספר tasks, מייצרים רצף של דגימות מהקרה הבאה :

$(x_1, \text{NULL}), (x_2, y_1), (x_3, y_2), \dots$

הרשות מקבלת רצף של זוגות המורכבים גם מהדגימה וגם מהתיוג של הדגימה שלפניה. בסוף הרצף מקבלים דגימה והרשות צריכה להוציא תיוג שלה. הנקודה היא שהמוצא לא מוכיח ישירות לדגימה שהתקבלה בכניסה, אלא דרך הזיכרון. תהליך ה-backpropagation מאלץ את הרשות שראתה בתחלת הרצף דוגמאות רלוונטיות ואת התיוג שלהם – למצוא דרך לקובד את הדגימות והתיוג שלהם בזיכרון, כך שבסוף הרשות תצליח לקשר את הדגימה החדשה לדגימות דומות מה עבר ולתиוג שלהם. להלן תרשים של השיטה:



גישה לזכרון

הגישה לזכרון מתבצעת על ידי קבלת שלוקטור כניסה x_t וחווצה שלוקטור כניסה y_t . הגישה מתבצעת בזורה של controller. מתוך x_t חישב וקטורי מאפיינים k_t , שמאדי כטורי המאפייני שבמטריצת הזיכרון. לאחר מכן, מחושב ה-cosine similarity בין k_t לכל אחת מהשורות (i) במטריצת הזיכרון:

$$K(k_t, M_t(i)) = \frac{k_t \cdot M_t(i)}{\|k_t\| \|M_t(i)\|}$$

$$\begin{aligned} & \text{בשלב זה, מחושב softmax על כל איברי } K(k_t, M_t(i)) \\ w_t^r(i) & \leftarrow \frac{\exp(K(k_t, M_t(i)))}{\sum_j \exp(K(k_t, M_t(j)))} \end{aligned}$$

ערכי הוקטור החדש יהיו גדולים יותר עבור שורות הדומות לוקטור k_t וקטנים עבור שורות השונות ממנו. לאחר מכן, מבצעים ממוצע משוקלל של כל שורות מטריצת הזיכרון באמצעות ערכי $w_t^r(i)$:

$$r_t \leftarrow \sum_i w_t^r(i) M_t(i)$$

בסוף דבר קיבלנו את r_t שהוא ממוצע משוקלל של הערכים בזיכרון שמחושב על פי וקטור הכניסה (דומה למנגנון ה-attention הנודע).

כתיבה לזכרון

בכתיבה לזכרון, משנים את השורות במטריצת הזיכרון שלא נועה בהם שימוש מזמן. הרעיון הוא שורות שלא כתבנו אליהם ושלא קראו מהם מזמן, נראה לא נושא אינפורמציה וניתן להחליף אותן בתווות אינפורטטיביות יותר. לצורך זה, מחשבים את וקטור w_t^u (המדד שלו כמספר השורות במטריצת הזיכרון) אשר מכמתות כמה שימוש נעשה בכל שורה במטריצת הזיכרון. הוקטור מחושב בזורה איטרטיבית בזורה הבאה:

$$w_t^u \leftarrow \gamma w_{t-1}^u + w_t^r + w_t^w$$

כאשר γ הוא סוג של discount factor אשר מ�פס בזורה הדרגתית את הוקטור. w_t^w הוא הוקטור שהציגנו לעיל אשר מכיל אינפורמציה כמה אחוז היעותה כל שורה במטריצה מהווקטור שנקרא. w_t^w בעל תפקיד דומה במשימת הכתיבה.

w_t^{lu} הוא וקטור בינרי (מכיל {0,1}), גם הוא בגודל של מספר השורות במטריצה הזיכרון, אשר מכיל 1 ב- t הערכים שבhem הוקטור w_t^u הcy קטן. או במילים אחרות, מ- השורות שהcy פחותה בשימוש בזמן האחרון :

$$w_t^{lu} = \begin{cases} 0 & \text{if } w_t^u(i) > m(w_t^u, n) \\ 1 & \text{if } w_t^u(i) \leq m(w_t^u, n) \end{cases}$$

($n, m(w_t^u)$ הוא האבר ה- m -י הcy קטן ב- w_t^u . בשלב זה ניתן לחשב את w_t^w – וקטור הכתיבה (גם בממך של מספר השורות של המטריצה) :

$$w_t^w \leftarrow \sigma(\alpha)w_{t-1}^r + (1 - \sigma(\alpha))w_{t-1}^{lu}$$

α הוא פרמטר נלמד השולט כמה אנחנו נתונים חשיבות לוקטור הכתיבה w_{t-1}^r הקודם, וכמה נתונים לוקטור w_{t-1}^{lu} שיחסבנו עכשו. בסופו של דבר מתקבל וקטור w_t^w אשר שולט באיזה שורות נעדכן בזיכרון ואילו שורות לא. בשלב האחרון מתבצע העדכון עצמו :

$$M_t(i) \leftarrow M_{t-1}(i) + w_t^w(i)k_t, \quad \forall i$$

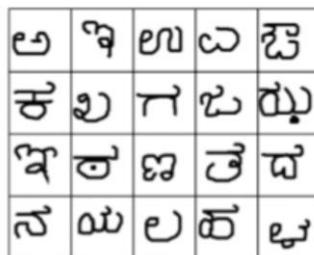
אשר מוסיף את וקטור k_t (בצורה ממושקלת) לשורות הרלוונטיות במטריצה הזיכרון.

פִּרְדִּיקְצִיה
בזמן ה-*inference*, מתבצעים שלבים הבאים :

1. ה-*controller* מקבל דוגימה x ומפעה אותה (באמצעות רשות) ל- k_t
2. k_t נשלח ל-*read heads* ומחושבים המשקלים לקריאה w_t^r
3. בעזרת המשקלים הללו, מחושב הממוצע המשוקל של שורות הזיכרון r_t
4. וקטור זה נשלח על ידי ה-*controller* לרשת קלסיפיקציה אשר מספקת פִּרְדִּיקְצִיה

האימפלמנטציה המדוייקת יכולה להשתנות מעט. למשל, ה-*controller* יכול להיות רשת FC או RNN, מה שি�שנה מעט את התהליך.

ניסויים
הניסויים התקבצו על ה-*dataset* *Omniglot dataset*. להלן דוגמאות מתוכו :



הdataset מורכב מ-1643 *classes* שונים, מתוכם 1200 בסט האימון ו-443 ב-*test set*. באימון יוצרו 100,000 רצפיים של 50 דוגמאות שנדרגו מתוך חמישה *classes* – עשרה דוגמאות מכל *class*. הביצועים של האלגוריתם עלול על ביצועים אנושיים :

Table 1. Test-set classification accuracies for humans compared to machine algorithms trained on the Omniglot dataset, using one-hot encodings of labels and five classes presented per episode.

MODEL	INSTANCE (% CORRECT)					
	1 ST	2 ND	3 RD	4 TH	5 TH	10 TH
HUMAN	34.5	57.3	70.1	71.8	81.4	92.4
FEEDFORWARD	24.4	19.6	21.1	19.9	22.8	19.5
LSTM	24.4	49.5	55.3	61.0	63.6	62.5
MANN	36.4	82.8	91.0	92.6	94.9	98.1

Simple Neural Attentive Meta-Learner (SNAIL)

הчисIRON העיקרי בשיטה הקיימת הוא שהארQUITטורה קבועה מראש. למשל, מטריצת הזיכרון מוגדרת מראש ולא יכולה לגדול. לכן, במקרה של רצף ארוך מאוד, קיים סיכון שהרשת "תשכח" את מה שהיא למדת בעבר הרחוק, מכיוון שמשאבי הזיכרון שלו מוגבלים. השיטה שנציג עבשו, *Simple Neural Attentive Meta-Learner* מוגברת על החיסIRON זהה באמצעות שני מנגנונים משלימים:

- *Dilated convolution* – קונבולוציה המתבצעת בקפיצות. למשל, עבור kernel של 3×3 הקונבולוציה לא חיבת להתבצע על שלושה פיקסלים סמוכים. ניתן לבצע את הקונבולוציה בקפיצות בשיעור R . למשל, הערך המרכזי יכפיל את הפיקסל המרכזי, הערך השמאלי יכפיל פיקסל הרחוק מהפיקסל המרכזי R פיקסלים שמאלה, וכן הלאה. שיטה זו מאפשרת receptive field גדול יותר לקונבולוציה.
- *Attention mechanism* – שיטה למישקול ערכים בצורה "חכמה". בשיטה זו מחושב ערך של "רלוונטיות" של כל מה כניסה ליציאה הספציפית, ומוחושב ממוצע משוקל של כל הכניסותUPIFIOT. פי מידת הרלוונטיות שלהם.

תיאור השיטה

הארQUITטורה מורכבת ממספר בלוקים:

, *Dense Block* : מקבל *input* בעל אורך T ובעל C ערוצים (T הוא אורך הרצף, ו- C יכול להיות למשל ה- *state-action reward*) שהתקבלו עבור כל אחד מ- T -השלבים). מבצעים *dilated convolution* חד ממדית על האינפוטים עם D פילטרים שונים (מת�בלת מטריצה בגודל $D \times T$) ומבצעים שרשור (*concatenation*) על ממד הזמן. הבלוק מוציא *output* בגודל $(C + D) \times T$ המכיל גם את ה- *input* וגם את הקונבולוציות (המאפיינים שנלמדו). להלן אלגוריתם :

```

1: function DENSEBLOCK(inputs, dilation rate  $R$ , number of filters  $D$ ):
2:   xf, xg = CausalConv(inputs,  $R$ ,  $D$ ), CausalConv(inputs,  $R$ ,  $D$ )
3:   activations = tanh(xf) * sigmoid(xg)
4:   return concat(inputs, activations)

```

אחרי השני, כאשר בכל פעם מגדילים את ה- *rate* (R) פי 2. התהיליך מתבצע עד שגודל ה- *receptive field* של כל אורך ה- *input*. זה מאפשר לבlok האחרון לקבל *input* של כל אורך הרצף. להלן האלגוריתם :

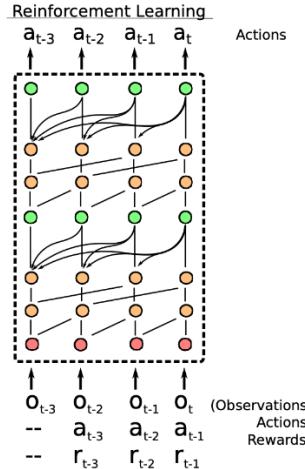
```

1: function TCBLOCK(inputs, sequence length  $T$ , number of filters  $D$ ):
2:   for  $i$  in  $1, \dots, \lceil \log_2 T \rceil$  do
3:     inputs = DenseBlock(inputs,  $2^i$ ,  $D$ )
4:   return inputs

```

מבצע מנגן על ה-*outputs* של שכבה ה-*TCL* הקודמת. בשיטה זו מקבלים ערך אחד (או וקטור אחד) – לא משנה מה אורך הרצף T . וקטור זה יכול "להתחשב" גם בהיסטוריה רחוקה, מכיוון של "שכחנו" את ההיסטוריה, אלא משקלים אותה בכל פעם מחדש. מהווצא של השכבה אפשר לבצע פרדיקציה לפעולה (*action*) הטובה ביותר.

להלן תרשים של השיטה:



ניסויים

גם בעבודה הזאת התבכע ניסוי על ה-*Omniglot dataset*. להלן הביצועים:

Table 1: 5-way and 20-way, 1-shot and 5-shot classification accuracies on Omniglot, with 95% confidence intervals where available. For each task, the best-performing method is highlighted, along with any others whose confidence intervals overlap.

Method	5-Way Omniglot		20-Way Omniglot	
	1-shot	5-shot	1-shot	5-shot
Santoro et al. (2016)	82.8%	94.9%	–	–
Koch (2015)	97.3%	98.4%	88.2%	97.0%
Vinyals et al. (2016)	98.1%	98.9%	93.8%	98.5%
Finn et al. (2017)	98.7% ± 0.4%	99.9% ± 0.3%	95.8% ± 0.3%	98.9% ± 0.2%
Snell et al. (2017)	97.4%	99.3%	96.0%	98.9%
Munkhdalai & Yu (2017)	98.9%	–	97.0%	–
SNAIL, Ours	99.07% ± 0.16%	99.78% ± 0.09%	97.64% ± 0.30%	99.36% ± 0.18%

הניסוי השני התבכע במשחק הליכה במבוך. האימון התבכע על מובנים קטנים, וה-*test* על קטנים וגדולים. בזמן-*test* מראים לסוכן פעם אחת את המבוך על מנת שילמד אותו (*one shot*), ואז מתחילה מחדש באותו המבוך ומבצעים את האבלואציה. להלן הביצועים:

Method	Small Maze		Large Maze	
	Episode 1	Episode 2	Episode 1	Episode 2
Random	188.6 ± 3.5	187.7 ± 3.5	420.2 ± 1.2	420.8 ± 1.2
LSTM	52.4 ± 1.3	39.1 ± 0.9	180.1 ± 6.0	150.6 ± 5.9
SNAIL (ours)	50.3 ± 0.3	34.8 ± 0.2	140.5 ± 4.2	105.9 ± 2.4

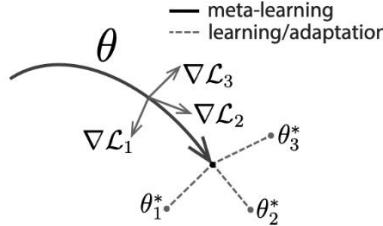
Model Agnostic Meta-Learning (MAML)

שיטת זו נגישה לבניית ה-*Meta Learning* מכיוון שהוא מושנה מעט. הרעיון הוא שביצוע *fine tuning* על מספר קטן של דוגמאות (*few shot*) עלול ליצור *overfitting*. לכן, מטרת השיטה היא למצוא סט של משקלים לרשות,

אשר יהיה קל לבצע עליו *fine tuning* עבור מספר גדול של משימות. בambilים אחרות, נמצא θ ספציפי שהיה קרוב כמו שיתר ל- θ_i^* - סט המשקלות האופטימלי למטלות i , כך תהיה קל להגיע מ- θ ל- θ_i^* :

$$\min_{\theta} \sum_{task i} \sum_{test}^i (\theta - \alpha \nabla_{\theta} \mathcal{L}_{train}^i(\theta))$$

במהלך האימון, המודל פוגש במספר *tasks* שעליין הוא מבצע אופטימיזציה כך שהמשקלות יגיעו לאופטימום עבור מספר קטן של דוגמאות ושל צעדי אימון. בזמן $test$, אנחנו מקווים שסט הפרמטרים האלה הצליח להכפיל עבור מספר גדול של מטלות, וגם אם המודל יראה מטלה חדשה הוא יוכל למצוא לה אופטימום בעזרת מספר קטן של דוגמאות וצעדי אימון. להלן אילוסטרציה לשיטה:



במהלך *training* אנו מנסים להגיע למרחב המשקלות למקום שיהיה "קרוב" ל- $\theta_1^*, \theta_2^*, \theta_3^*$. בכל איטרציה במהלך האימון נדגמים מספר מטלות ($\mathcal{T} \sim p(\mathcal{T})$, מכל מטלה N דוגמאות ומבצע עדכון למשקלות באמצעות *gradient descent* כך שמתקובל θ . בשלב האחרון, מוצבעת גזירה של *loss* על פי הפרדיקציות של הרשות עם המשקלות המעודכנות θ' לפי המשקלות המקוריים θ . זהה נקודה מעט עדינה. מכיוון

שאנו רוצים לחשב כיצד יש לשנות את המשקלות המקוריים, כך שלאחר גזירה על *data* מטלה \mathcal{T} על אותה המטלה יהיה נמוך, אךנו אנו גוזרים את *loss* שמתקובל מהרשת על המשקלות המעודכנים לפי המשקלות המקוריים. יש לנו לב כי המשקלות המעודכנים מכילים גנזרת של המשקלות המקוריים. לכן, על מנת לבצע את שלב העדכון ישנה **涅槃**. פעולה זו יכולה לאתגר את היציבות של תהליך האימון. להלן תאור האלגוריתם:

Algorithm 1 Model-Agnostic Meta-Learning

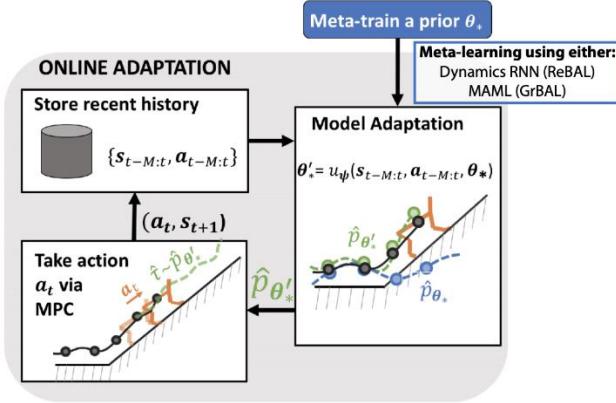
```

Require:  $p(\mathcal{T})$ : distribution over tasks
Require:  $\alpha, \beta$ : step size hyperparameters
1: randomly initialize  $\theta$ 
2: while not done do
3:   Sample batch of tasks  $\mathcal{T}_i \sim p(\mathcal{T})$ 
4:   for all  $\mathcal{T}_i$  do
5:     Evaluate  $\nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$  with respect to  $K$  examples
6:     Compute adapted parameters with gradient descent:  $\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$ 
7:   end for
8:   Update  $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$ 
9: end while

```

Adaption in Real-Time

שימוש באlgorigthמי *DRL* בעולם האמיתי הוא מתגרר במספר סיבות. איסוף *data* בסביבת עולם אמיתי הוא יקר, קיים הרבה רעש בסביבות "אמיות", וקיים מקרים רבים של *distribution shift* (פתאים ירד גשם והמנועים של הרובוט מגיבים לאט יותר). הרעיון בעבודה זו שנצג הוא שימוש ב-*meta-learning* על מנת להסתגל מהר לשינויים בסביבה. ההנחה היא שבכל *time step* יכול להיות סביבה חדשה. לכל סביבה יש את אותו מבנה אבל עם פרמטרים שונים (*dynamics*). לכן, אנחנו לומדים מהמסלול (*trajectory*) האחרון שהסוכן חווה. השיטה שמשתמשת ב-*MAML* ללמידה *RL* בעולם האמיתי נקראת *GrBAL*. להלן סכמה של השיטה:



להלן סיכום של היתרונות והחסרונות של השיטות שנסקרו :

הערות	<i>ReBAL</i>	<i>MAML</i>	<i>SNAIL</i>	הסבר	קריטריון
	V	V	X	אם נסיף <i>data</i> נתכנס לאופטימום	<i>Consistent</i>
<i>RL reward</i> מוגע רק בסוף למשל כאשר ה-	\sim	X	V	האם יש מספיק כוח ייצוג לרשת (אלגוריתמים שונים)	<i>Expressive</i>
	X	\sim	\sim	חיפוש יעיל במרחב הבעה	<i>Structured Exploration</i>
	V	X	X	האם מתאים לביעות עולם אמיתי	<i>Efficient & off-policy</i>

Transfer Learning

הו תחום ב-ML אשר מנסה להשתמש באלגוריתמים (רשתות) שאומנו על משימה אחת ולהתאים אותם למשימה אחרת. למשל אפשר להשתמש בשיטה שאומנה על *image classification dataset* על *object detection*, ולחתאים אותה ל-*imageNet*.

נרחיב על מספר סוגים של אלגוריתמים:

- – התאמה של רשת שאומנה על מטלה אחת לביצוע מטלה אחרת
 - – שימוש ברשת ללא כל התאמה *"Fingers crosses"*
 - – *Fine-tuning*
 - – ארכיטקטורות המותאמות ל-*transfer learning* – למשל *progressive networks*
- – אימון רשת על מספר מטלות והתאמתה למטלה חדשה
 - – *Model based RL*
 - – *Model distillation*
 - – *Modular policy networks*

לא נרחיב ב-*style transfer*, שכן לא נעשו בכך שימוש. יש לציין שאפשר לראות את ה-*style* כסוג של שיטה זאת. להלן נרחיב בשאר השיטות.

Fine-tuning

ה策ורה הפושאה ביוטר של *fine-tuning* היא שימוש ברשת שאומנה על מטלה אחת למטלה אחרת. ניתן לבצע אימון על כל משקלות הרשת או להקפיא את השכבות הראשונות ולאמן רק את האחרונה (או האחרונות).策ורה יותר מתקדמת היא לאמן את הרשת את האימון הראשוני כך שתיהיה כללית ומוגנת יותר, כולל תנשה לפטור את הבעה במספר רב ומגוון של דרכים. אימון זהה גורם לרשות להיות כללית יותר ולהתאים למגוון רב יותר של בעיות. להלן נskoor דרכיים לבצע אימון זהה (אימון מגוון).

מאמר בשם *Reinforcement Learning with Deep Energy-Based Policies* מתמודד עם בעיה זו. הגישה היא למידה של פוליסת סטוכסיתית, כך שתתבצע אקספלורציה ויכולה למידה של מספר גישות לפתרון. במקום ללמידה שפותרת את הבעיה策ורה הטובה ביותר, בשיטה זו מנסים למצוא כמה שיותר פתרונות לבעה. רשת שלמדו策ורה כזו יכולה להיות בסיס טוב ל-*transfer learning*.

הגישה בה משתמשים במאמר לטובת למידה של פוליסת סטוכסיתית היא *energy-based models*. פונקציית האנרגיה היא *Q-function* אשר עברה התאמת כך שתתאים למיקסום האנתרופופיה, בנוסף למיקסום ה-*reward*. כאשר פונקציית ה-*Q* הסטנדרטית באה למקסם את :

$$\pi_{std}^* = \arg \max_a \sum_t \mathbb{E}_{(s_t, a_t) \sim p_\pi} [r(s_t, a_t)]$$

פונקציית האנרגיה שפותחה במאמר באה למקסם בנוסח את האנתרופופיה :

$$\pi_{MaxEnt}^* = \arg \max_a \sum_t \mathbb{E}_{(s_t, a_t) \sim p_\pi} [r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot | s_t))]$$

בשיטה זו ניתן למקסם את האנתרופופיה על כל המסלול ולא רק לכל *step* כמו שנעשה בשיטות קודומות. להלן ההגדרה של ה-*Q-function* מהמאמר :

$$Q_{soft}^*(s_t, a_t) = r_t + \mathbb{E}_{(s_{t+1}, \dots) \sim p_\pi} \left[\sum_{l=1}^{\infty} \gamma^l \left(r_{t+l} + \alpha \mathcal{H}(\pi_{MaxEnt}^*(\cdot | s_{t+l})) \right) \right]$$

ו π ה- q -function :

$$V_{soft}^{\theta}(s_t) = \alpha \log \mathbb{E}_{a'} \left[\frac{\exp\left(\frac{1}{\alpha} Q_{soft}^{\theta}(s_t, a')\right)}{q_{a'}(a')} \right]$$

להלן האלגוריתם המלא :

Algorithm 1 Soft Q-learning

$\theta, \phi \sim$ some initialization distributions.

Assign target parameters: $\bar{\theta} \leftarrow \theta, \bar{\phi} \leftarrow \phi$.

$\mathcal{D} \leftarrow$ empty replay memory.

for each epoch **do**

for each t **do**

Collect experience

Sample an action for s_t using f^{ϕ} :

$a_t \leftarrow f^{\phi}(\xi; s_t)$ where $\xi \sim \mathcal{N}(\mathbf{0}, I)$.

Sample next state from the environment:

$s_{t+1} \sim p_s(s_{t+1}|s_t, a_t)$.

Save the new experience in the replay memory:

$\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$.

Sample a minibatch from the replay memory

$\{(s_i^{(i)}, a_i^{(i)}, r_i^{(i)}, s_{i+1}^{(i)})\}_{i=0}^N \sim \mathcal{D}$.

Update the soft Q-function parameters

Sample $\{a^{(i,j)}\}_{j=0}^M \sim q_{a'}$ for each $s_{t+1}^{(i)}$.

Compute empirical soft values $\hat{V}_{soft}^{\bar{\theta}}(s_{t+1}^{(i)})$ in (10).

Compute empirical gradient $\hat{\nabla}_{\theta} J_Q$ of (11).

Update θ according to $\hat{\nabla}_{\theta} J_Q$ using ADAM.

Update policy

Sample $\{\xi^{(i,j)}\}_{j=0}^M \sim \mathcal{N}(\mathbf{0}, I)$ for each $s_t^{(i)}$.

Compute actions $a_t^{(i,j)} = f^{\phi}(\xi^{(i,j)}, s_t^{(i)})$.

Compute Δf^{ϕ} using empirical estimate of (13).

Compute empirical estimate of (14): $\hat{\nabla}_{\phi} J_{\pi}$.

Update ϕ according to $\hat{\nabla}_{\phi} J_{\pi}$ using ADAM.

end for

if epoch mod update_interval = 0 **then**

 Update target parameters: $\bar{\theta} \leftarrow \theta, \bar{\phi} \leftarrow \phi$.

end if

end for

Progressive Networks

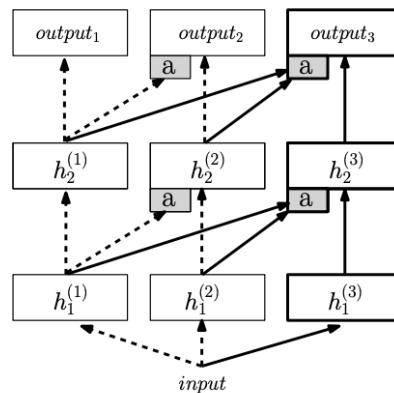
חסרונות של שיטת ה-*fine tuning* הם שאימון מחדש של כל הרשות על-*dataset* קטן יכול לגרום ל-*overfitting*. בנוסף, אימון מחדש גורם לרשות לשכוח את ה-*task* המקורי, דבר שיכול להיות בעייתי ב-*use cases* מסוימים. הרעיון ב-*soft-parallel networks* הוא לתוכנן ארכיטקטורתה שתתאים במיוחד ל-*learning cases*. דבר זה מתרחש בצורה הבאה. בשלב הראשון מאמנים רשות סטנדרטית על משימה ראשונה. כאשר מעוניינים בהוספה של משימה חדשה, מקפיאים את הרשות הראשונה, ומוסיפים רשות שנייה (יכולת להיות זהה לראשונה) אשר מקבלת בכל שכבה את המאפיינים של השכבה המתאימה ברשות הקודמת, בנוסף למאפיינים

החדשים שנלמדים. בדומה כזו ניתן להרוויח בכך מהמאפיינים שנלמדו במתלה הקודמת אשר יכולים להיות רלוונטיים למשימה הבאה, והן מיומו של פרמטרים חדשים ייעודים למשימה.

ספציפית, כל שכבה מבצעת את החישוב הבא :

$$h_i^{(k)} = f \left(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j < k} U_i^{(k:j)} h_{i-1}^{(j)} \right)$$

כאשר הביטוי הראשון הוא שכבה לנארית סטנדרטית ($h_{i-1}^{(k)}$ הוא היציאה מהשכבה הקודמת ו- $W_i^{(k)}$ היא מטריצת המשקלים המתאימה), והביטוי השני ($\sum_{j < k} U_i^{(k:j)} h_{i-1}^{(j)}$) הוא קומבינציה של היציאות מהשכבות המקבילות של רשותות שלמדו מטלות אחרות. ספציפית בשיטה זו, לכל וקטור יציאה של שכבה i מטלות קודמות קיימת מטריצת משקלות בעלת ממדים זהים, וכל התוצאות של המכפלות של המטריצות הללו עם וקטורי המאפיינים נסכמו ביחד עם אלה של השכבה הנוכחית ומוסברות בפונקציית האקטיבציה לשכבה הבאה. להלן סכמה של הארכיטקטורה :



במאמר המקורי נעשה שימוש ב-*adapter layer*, מימוש מעט מורכב יותר שנועד לתת משקל נכוון לכל אחד מוקטוריו המאפיינים של המטלות הקודמות. במימוש זה, וקטורי המאפיינים מוכפלים ב- α (סקלר – פרמטר נלמד) ומוסברים למרחיב קטן (באמצעות מטריצה נלמדת) וועברים אקטיבציה (סיגמוואיד). תוצאה האקטיבציה מועברת לשכבה לנארית ומוגוסףת לשכבה הילינארית של המאפיינים של המטלה זו :

$$h_i^{(k)} = \sigma \left(W_i^{(k)} h_{i-1}^{(k)} + U_i^{(k:j)} \sigma(V_i^{(k:j)} \alpha_{i-1}^{(<k)} h_{i-1}^{(<k)}) \right)$$

יתרונות השימוש :

- יכולת ללמידה K מטלות שונות
- במידה מהירה יותר בעזרת מאפיינים שנלמדו במטלות קודמות (*transfer learning*)
- אין שכחה של משקלות (*catastrophic forgetting*)

חסרונות השימוש :

- גודל הרשות ומספר המשקלות עולה משמעותית עם כל הוספה של מטלה
- לא ניתן לדעת בצורה ישירה איזה מתוך המטלות הקודמות אכן רלוונטיות למטלה הבאה

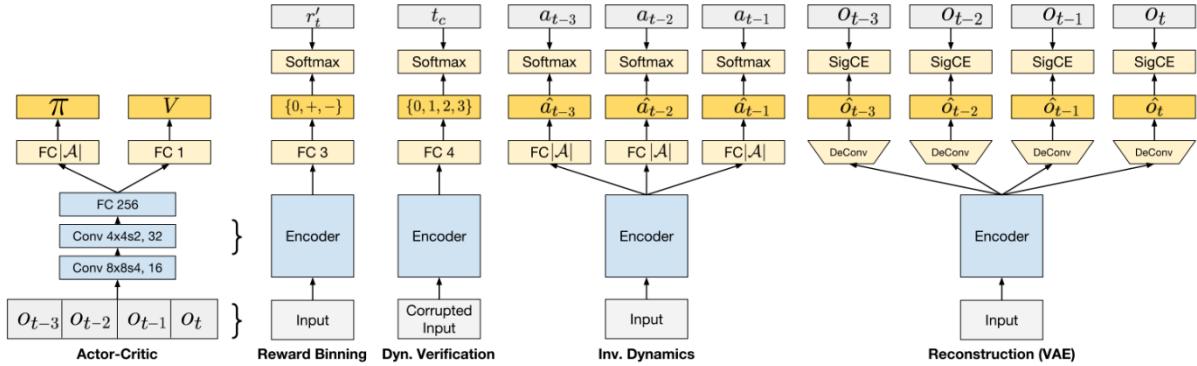
Self-Supervision for RL

תחום ה-*representation learning* *SSL* ו-*RL* יכול להיות מאד רלוונטי ללמידה ייעילה ב-*RL*. למשל שימוש בשכבה לפני אחורונה של רשות *policy* שאומנה בסביבה מסוימת יכולה לייצג בצורה טובה את הסביבה שאורה הרשות למדעה. נציג את המאמר *Loss is its own reward: Self-supervision for reinforcement learning*

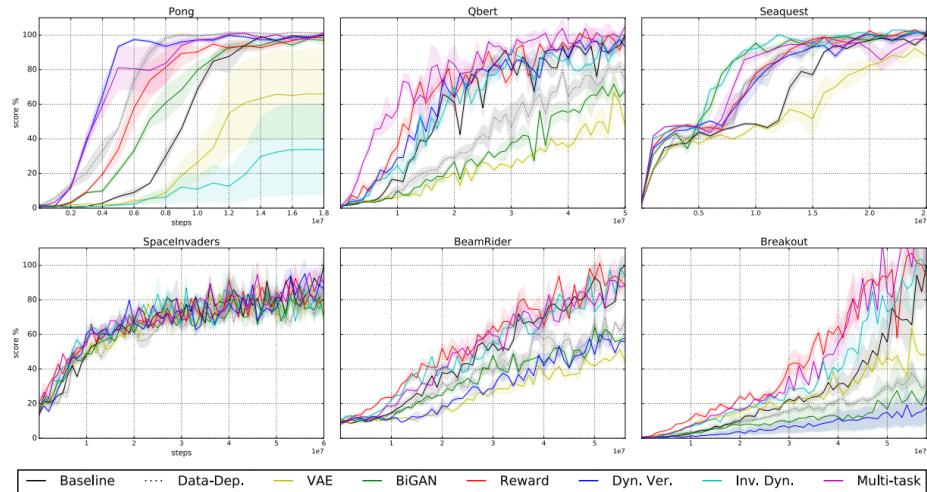
נטען במאמר של מידה של *rewards* היא קשה מכיוון שהם יכולים להיות מאוד דלילים (*sparse*) בזמן. לעומת זאת, ניתן למצוא **auxiliary losses** אשר **ילמדו את הסביבה** וייהו רלוונטיים לחיזויי-*rewards*. להלן רשימה של *auxiliary tasks* שנעשה בהם שימוש במאמר:

- בנוסף לחיזויי-*reward*, הרשות הייתה צריכה לחזות האם הוא חיובי או שלילי – *Reward binning* •
- חיזויי s_{t+1} בהינתן (s_t, a_t) – *Dynamics* •
- חיזויי a_{t+1} בהינתן (s_t, a_t) – *Inverse Dynamics* •
- החלפת מצב במצב קרוב – מה עבר או מה עתיד (הרשות צריכה לחזות את אותו מצב) – *Corrupt Dynamics* •
- שימוש בשינויים קטנים במצב – *Reconstruction* •

להלן סכמה של השיטה:



הרשות אומנה על מטלה אחת, ואז, בעזרת ה-encoder בלבד, אומנה על מטלה חדשה. הראו במאמר שהביצועים במטלה השנייה טובים יותר כאשר מאומנים את כל הרשות מאשר כאשר מקפאים את ה-encoder. להלן ביצועים על מטלות שונות:



כעת ניבור לשיטות transfer learning המבוססות .multi-task

Model-Based RL

המטרה בשיטות אלה היא שעל ידי אימון על מספר שיטות, הרשת תלמד את **המודל** שמאחורי הסביבה – למשל חוקי הפיזיקה.

Actor Mimic

Actor-Mimic Approach היא שיטה לאימון סוכן לשחק במשחקי Atari רבים, כך שיגיע לרמה קרובת לשוכנים שאומנו על משחקים ספציפיים. המודל מאומן באמצעות expert DQN networks שאומנו על משחק בודד כל אחת. הסוכן לא לומד על בסיס ה-reward אלא על בסיס המוצא של הרשות :

$$\pi_{E_i}(a|s) = \frac{e^{Q_{E_i}(s,a)/\tau}}{\sum_{a \in A_{E_i}} e^{Q_{E_i}(s,a)/\tau}}$$

כאשר Q היא ה-Q-function של הרשות הספציפית למשחק i . לאחר מכן, מחושב ה-loss של הרשות הכללית :

$$\mathcal{L}_{policy}^i(\theta) = \sum_{a \in A_{E_i}} \pi_{E_i}(a|s) \log \pi_{AMN}(a|s; \theta)$$

פונקציה זו מקרבת את ההתפלגות של הרשות הכללית לזו של הרשות הספציפית. בנוסף, על מנת לשפר את הביצועים, ניתן להוסיף MSE loss על הבדלים באקטיבציות בין הרשותות (בדומה ל-distillation :

$$\mathcal{L}_{FeatureRegression}^i(\theta, \theta_{f_i}) = |f_i(h_{AMN}(s; \theta); \theta_{f_i}) - h_{E_i}(s)|_2^2$$

ה-loss הכלול מחושב כסכום של שני ה-losses :

$$\mathcal{L}_{ActorMimic}^i(\theta, \theta_{f_i}) = \mathcal{L}_{policy}^i(\theta) + \beta \cdot \mathcal{L}_{FeatureRegression}^i(\theta, \theta_{f_i})$$

לאחר האימון, אם רוצים לאמן על מטלה חדשה, ניתן להוריד את השכבה الأخيرة ולבצע transfer learning .

Distillation for Multi-Task Transfer

Distillation היא שיטה להעברת מידע מודול אחד (בצורה טיפוסית מודל גדול) למודול שני (קטן יותר). ניתן להשתמש בשיטה זו גם כאשר המודול הוא ensemble של מודלים, ואין אפשרות להריץ הכל-b-production. בשיטה זו לוקחים מודל קטן ומאמנים אותו לחכotta את וקטור ההסתברויות (softmax) שהמודול הגדל מוציאה.

בשיטה זו מאמנים את המודלים ב-softmax עם טמפרטורה :

$$q_i = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)}$$

במהלך האימון נבחר T גבוה, אשר גורם ל-logits להיות גבוהים יותר. אינטואיטיבית, זה גורם "לחזק את הסיגנל" שעובר ברשות. לאחר האימון, T של הרשות אותה למדו מוגדר ל-1. בפרדיוקציה מתבצע ממוצע משקלל בין שני הערכים, עם הטמפרטורה ובלדייה ($T=1$).

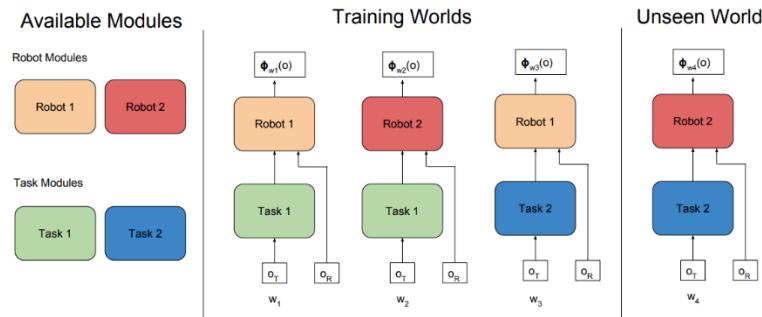
Modular Neural Network

נניח שישנה סביבה בה פועלים מספר רובוטים. בסביבה יכולות להיות מספר מטלות (tasks). רובוט יכול לבצע מספר מטלות שונות. אנו מעוניינים להפריד בין הייצוג של הרובוט לייצוג של מטלה. אם נצליח לבצע זאת,

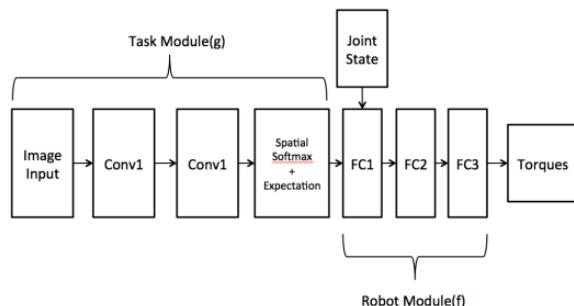
ונכל ביכולת לבצע התאמה בין רובוט למטרלה, גם אם ספציפיות אותו הרובוט לא אומן על המטרלה הזאת. את הפלישה ניתן למדל בצורה הבא :

$$\phi_{w_{rk}}(o_w) = \phi_{w_{rk}}(o_{w,T}, o_{w,R}) = f_r(g_k(o_{w,T}), o_{w,R})$$

כאשר g_k היא פונקציה שאחרית על המטרלה, ו- f_r אחראית על הרובוט (היא זו שמחזירה את הפלישה של הרובוט). הבעה חייבת להיות מודולת בצורה הזאת, מכיוון שלכל רובוט צריך להגדיר פולישה שונה (למשל מספר המנוועים יכול להיות שונה) ולכל הפונקציה החיצונית חייבת להיות זו שמודלת את הרובוט. אם הלמידה התרחשה כמו שצרכיך, ניתן יהיה להחליף רק את g_k למטרלה אחרת והרובוט ידע להתמודד אליה, או להחליף את הרובוט f_r לרובוט אחר, אשר ידע לפתור את המטרלה. להלן סכמה של השיטה :



הארQUITטורה שבה השתמשו במאמר :



מרחבי מצבים ופעולות גדולים

ביעות עולם אמיתי מותאייניות לעיטים במרחבי פעולה גדולים – הרבה פעולות בדידות שהסוכן צריך לבחור מתוכן, ומרחבי מצבים גדולים. מקרים כאלה יכולים להיות מוגבלים, למשל קשה לבצע אקספלורציה עיליה במרחב מצבים גדול. פרק זה נסקרו שיטות להתמודדות עם הבעיה.

מרחב פעולה רציף

פיתרון אפשרי לעיליה הוא להמיר את מרחב הפעולות הבדייד למרחב embedding רציף. בדומה לכך ניתן לאפשר ליחסות בדורה י呜ה יותר, ובנוסף לקבל מטריקה של קרבה בין פעולות (מרחב אוקלידי cosine similarity וכו'). ניתן למשתמש במודל גנטטי ליצירה של וקטור למרחב הפעולות ובחור פעולות קרובות לאבלואציה, למשל על ידי KNN. עוד ניתן שיש בשיטה זו, שבונה מהשיטות המסורתיות שראינו בעבר, אין צימוד הכרחי בין הpolloisa לאבלואציה של הפעולות. למשל ב-policy based-Q, Q-function מה-Q-function. עם זאת, בשיטה שהציגנו כתע, ה-policy תלויה ב-Q, שכן היא אחראית רק על הצעה של וקטור למרחב הרציף, והיא אינה תלואה בפונקציית האבלואציה. דבר זה יכול לאפשר גמישות בתכנון האלגוריתם.

האלגוריתם מורכב משלבים הבאים:

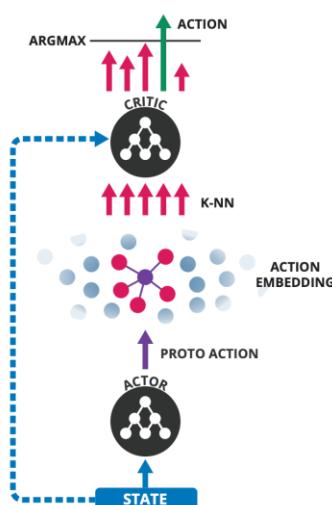
1. Generation של וקטור על ידי $\text{actor}.\text{actor}$. הוקטור נקרא proto-action מכיוון שהוא אינו פעולה עצמו, אבל הוא במרחב הפעולות, והוא מייצג פעולה רצiosa
2. בחירה של K פעולות קרובות ביותר באמצעות KNN (l_2)
3. critic נתן ציון לכל אחת מפעולות אלה, ובחרת הפעולה עם ה-value הגבוה ביותר

להלן תיאור האלגוריתם:

Algorithm 1 Wolpertinger Policy

State s previously received from environment.
 $\hat{a} = f_{\theta^{\pi}}(s)$ {Receive proto-action from actor.}
 $\mathcal{A}_k = g_k(\hat{a})$ {Retrieve k approximately closest actions.}
 $a = \arg \max_{a_j \in \mathcal{A}_k} Q_{\theta^Q}(s, a_j)$
 Apply a to environment; receive r, s' .

ולහלא סכמה של השיטה:



Action Elimination with Deep Reinforcement Learning

שיטות המשמשות ב-action elimination, מנסות להפחית את מרחב הפעולות על ידי שלילה של פעולות שהסתברות גבוהה לא יניבו ביצועים טובים. המאמר "Learn what not to learn: Action elimination" מציין שתי גישות: "with deep reinforcement learning"

- – שינוי ה-reward המתקבל מפעולות מסוימות (פעולות שאנו יודעים שלא יעילו). נונה את ה-reward עבור פעולות שאנו יודעים שהן "לא נכוןות" כך שיהיה נמוך יותר. בשיטה זו בדרך כלל נעשה שימוש במומחה שיגדר אל מול הפעולות פחות מבטיחות. שיטה זו בעיינית מבחינת הקושי לביצוע *tuning* ל-shaping והיא לא עילה מבחינת דגימות, מכיוון שנדרש "אקספלורציה" למציאת הפעולות הביעיניות.

- – שימוש בשני ראשים ב-policy network – אחד שוחזה את הפעולה שתמקסם את ה-reward (כפי שמתבצע בדרך כלל), ובוסף, ראש שוחזה פעולות שambilאים למינימום את ה-error, action elimination error, ככלומר שיזכר מצב שבו לא נבחרות הפעולות שאנו יודעים שלא יעילות לנו. את הפעולות האלה אפשר להגדיר על ידי מומחה, או על ידי אקספלורציה של הסוכן עצמו. חיסרונו בשיטה זו שתי המטרות מאוד מוצמדות (*coupled*) אחת לשניה, מכיוון שתיהן משפיעות על הפעולות שיבחרו ועל התוצאות שלהן. למשל פעולה אחד יכולה להביא ל-reward גבוה (שהראש הראשון יעודד) אבל היא יכולה להיות תת-אופטימלית (והראש השני לא יעודד), ולכן הלמידה יכולה להיות מأتגרת ולא יציבה.

במאמר, הכותבים מציעים שיטה להפרדת הצימוד בין ה-policy network לרשות ה-action elimination. כל אחד מהרכיבים מקבל מטרה (*objective*) אחרת, מה שմספר את ההתקנסות. ה-policy network רשות סטנדרטית (DQN) אשר חוזה את הפעולה האופטימלית. רכיב ה-action elimination מمدל את הבעיה *c-bandit* – באמצעות וקטור מאפיינים המייצג את המצב (contextual linear bandit) בפועלות אחת מתחום ורבות (bandits). האטגרים ביצוג כזה הם שמרח הפעולות גדול, ולכן מדובר ב-*contextual bandits* עם מספר גדול של bandits (מה שמקשה את הבעיה). בנוסף, על מנת ללמידה פתרון יציב, וקטור המאפיינים (context) נדרש לא להשנות עם הזמן, ולכן ניתן למשל להשתמש במאפיינים שהpolicy network מחלצת. לכן, חלק זה מרכיב מרשת אשר נקראת AEN (action elimination network) ביחס לастטוקסט (contextual bandit). בפתרון שלהם, רשות ה-AEN מחלצת וקטור מאפיינים, אשר משמש כ-input ל-UCB אשר נסקר בפרק על *multi arm bandit*.

מתמטית, מניחים שקיים $\theta_a^T \phi(s_t)$ כך ש- $e_t(s_t, a) = \theta_a^T \phi(s_t)$ היא רשות המחלצת מאפיינים מהמצב s_t , ו- $\theta_a^T \phi(s_t)$ הוא וקטור עבר כל מצב, כך שהמכפלה הסקלרית שלו עם המאפיינים שחילצנו תיתן סקלר – elimination signal. האתגר בגישה זו הוא שבמלה'ן אימון ϕ וקטור המאפיינים משתנה, ולכן –/contextual bandit לא יכול להימדד בצורה טובה. הפיתרון הוא שהרשות תעדכן פעם אחת כל מספר פעדים, ולאחר כל עדכון contextual bandit יתאמן מחדש.

האלגוריתם מורכב משתי רשותות – Q – רשות ה-policy ורשת ה-AEN. האלגוריתם משתמש ב- E על מנת ליצור את וקטור המאפיינים ל-*contextual bandit* לינארי. הרשות מתעדכנת בכל L איטרציות (*AENUpdate*), כאשר בכל איטרציה המודל הלינארי מתעדכן מחדש. וקטור המאפיינים ($\phi(s)$ – אלגוריתם), אשר באלגוריתם מועבר ל-*LastLayerActivations*:

$$V_a = \lambda I + \sum_j \phi(s_j) \phi(s_j)^T$$

$$b_a = \sum_j \phi(s_j)^T e_j$$

לאחר מכן פותרים את המודל **AENUpdate** (*AEN*) ומעדכנים את h -*AEN*: $V_a^{-1}b_a$. בשלב זהה משתמשים ב- *contextual linear bandit model* כדי לשלול פעולות שקיבלו הסתברות גבוהה (*Targets, ACT*). הפעולות שנשארו, שאוטם ניתן לבצע הן:

$$A' = \left\{ a : E(s)_a - \sqrt{\beta \phi(s)^T V_a^{-1} \phi(s)} < l \right\}$$

מתוך אותן פועלות נבחר באמצעות ϵ -greedy. במקרה של *exploitation* נבחר את הפעולה עם ה-*Q-value* הגבוהה ביותר (A'), ובמקרה של *exploration* נבחר פעולה אקראית מתוך A . להלן האלגוריתם:

Algorithm 1 deep Q-learning with action elimination

Input: $\epsilon, \beta, \ell, \lambda, C, L, N$

Initialize AEN and DQN with random weights ω, θ respectively, and set target networks Q^-, E^- with a copy of θ, ω
 Define $\phi(s) \leftarrow \text{LastLayerActivations}(E(s))$
 Initialize Replay Memory D to capacity N

for $t = 1, 2, \dots$, **do**

$$a_t = \text{ACT}(s_t,$$

Execute action a_t and observe $\{r_t\}$

Store transition $\{s_t, a_t, r_t, e_t, s_{t+1}\}$ in D

Store transition $\{s_t, a_t, r_t, c_t, s_{t+1}\}$ in D

Sample transitions

$$\{s_j, a_j, r_j, e_j, s_{j+1}\}_{j=1}^m \in L$$

$$y_j = \text{Targets}(s_{j+1}, r_j, \gamma, Q^-, E^-, V^{-1}, \beta, \ell)$$

$$\theta = \theta - \nabla_{\theta} \sum_j (y_j - Q(s_j, a_j; \theta))^2$$

$$\omega \equiv \omega = \nabla_{\alpha} \cdot \sum_i (e_i - E(s_i; g_i; \omega))^2$$

$$\text{If } (t \bmod C) = 0 : Q^- \leftarrow Q^+$$

If $(t \bmod C) = 0$:

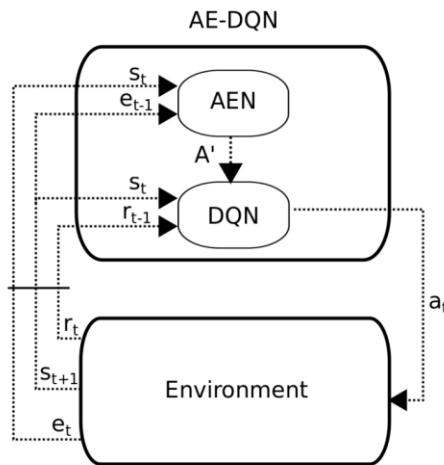
If $(t \bmod L) \equiv 0$:

```

function ACT( $s, Q, E, V^{-1}, \epsilon, \beta, \ell$ )
   $A' \leftarrow \{a : E(s)_a - \sqrt{\beta\phi(s)^T V_a^{-1} \phi(s)} < \ell\}$ 
  With probability  $\epsilon$ , return Uniform( $A'$ )
  Otherwise, return  $\arg \max_{a \in A'} Q(s, a)$ 
end function
function TARGETS( $s, r, \gamma, Q, E, V^{-1}, \beta, \ell$ )
  if  $s$  is a terminal state then return  $r$  end if
   $A' \leftarrow \{a : E(s)_a - \sqrt{\beta\phi(s)^T V_a^{-1} \phi(s)} < \ell\}$ 
  return  $(r + \gamma \max_{a \in A'} Q(s, a))$ 
end function
function AENUPDATE( $E^-, \lambda, D$ )
  for  $a \in A$  do
     $V_a^{-1} = \left( \sum_{j:a_j=a} \phi(s_j)\phi(s_j)^T + \lambda I \right)^{-1}$ 
     $b_a = \sum_{j:a_j=a} \phi(s_j)^T e_j$ 
    Set LastLayer( $E_a^-$ )  $\leftarrow V_a^{-1} b_a$ 
  end for
  return  $E^-, V^{-1}$ 
end function

```

להלן סכמה של התהילה:



Hierarchical DRL for Sparse Reward Environments

במאמר "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation" מתייחסים לבעה של RL במרחב מצבים גדול-ו-rewards-sparse. הגישה של האלגוריתם היא שימוש ב-**"intrinsic goals"** – תת-מטרות אשר עוזרות לסקון לצבע אקספלורציה. תת מטרה כזו יכולה להיות למשל למלכט במספר מקומות רוחק ממחשב, או להרים כמה שיותר חפצים ולנסות

להשתמש בהם. למשל במשחק "Montezuma's Revenge" ניתן להגדיר מטרות כגון : לאסוסף כמה שיותר מפתחות ולפתוח כמה שיטור דלותות, לחקור כמה שיטור חרדים או להגיע לנקודות מסוימות במהלך המשחק.

בנוספ', נעשה שימוש בשני רכיבים שונים : **the meta controller** וה-**controller**. ה-**meta controller** שולט בסופו ובפעולותיו שלו, ומקבל intrinsic goals מה-controller. מטרתו היא לגרום לסוכן לבצע את ה-**intrinsic goals**. הוא מבצע כל מספר מוגדר של צעדים ולבסוף מוגדר האם הוא הצליח או לא. הגודל שהוא מנסה למקסם הוא :

$$R_t(g) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}(g)$$

ה-**meta controller** אחראי על task הראשי, וכן אחראי להגדיר תת-משימות (intrinsic tasks) ל-**controller**. הגודל שהוא מנסה למаксם הוא :

$$F_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} f_{t'}$$

כאשר $f_{t'}$ הוא ה-**reward** המתקבל מהסביבה. בנוספ', קיים רכיב הנקרא **internal critic**, אשר אחראי להגדיר כמה טוב ה-**controller** מבצע את ה-**intrinsic tasks**. הוא מקבל את המצב, את המטרה ואת הפעולה ומוחזיר internal reward תחת המשימה הספרטנית. למשל, אם תת המשימה היא להגיע למפתחה, הוא יהיה אחראי להחזיר reward ברגע שהסוכן הגיע למפתחה, כך שה-**controller** יוכל ללמוד את תת המשימה :

הנקודה המרכזית היא שה-**controller** וה-**meta controller** עובדים בקבועי זמן שונים. בעוד ה-**meta controller** מקבל משימות בעלות זמן קצר יחסית, ומעדכן את המשקלות שלו בכל צעד, ה-**controller** אחראי על המשימות ארוכות הטווח (קרי – המשימות האקסטרינזיות), ומעדכן את המשקלות שלו אחת למספר צעדים. העבודה ההיררכית בקבועי זמן שונים מאפשרת למדוד ביעילות תת-משימות אשר ייעשה בהן שימוש עבור המשימה המרכזית. להלן תאור האלגוריתם :

Algorithm 1 Learning algorithm for h-DQN

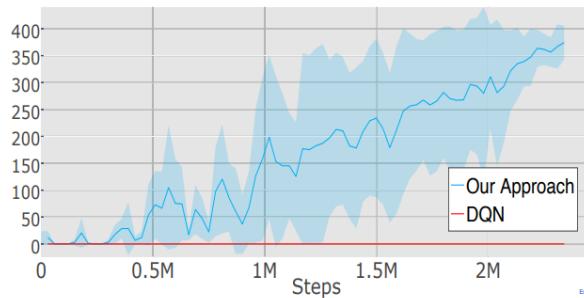
```

1: Initialize experience replay memories  $\{\mathcal{D}_1, \mathcal{D}_2\}$  and parameters  $\{\theta_1, \theta_2\}$  for the controller and
   meta-controller respectively.
2: Initialize exploration probability  $\epsilon_{1,g} = 1$  for the controller for all goals  $g$  and  $\epsilon_2 = 1$  for the
   meta-controller.
3: for  $i = 1, num\_episodes$  do
4:   Initialize game and get start state description  $s$ 
5:    $g \leftarrow \text{EPSGREEDY}(s, \mathcal{G}, \epsilon_2, Q_2)$ 
6:   while  $s$  is not terminal do
7:      $F \leftarrow 0$ 
8:      $s_0 \leftarrow s$ 
9:     while not ( $s$  is terminal or goal  $g$  reached) do
10:       $a \leftarrow \text{EPSGREEDY}(\{s, g\}, \mathcal{A}, \epsilon_{1,g}, Q_1)$ 
11:      Execute  $a$  and obtain next state  $s'$  and extrinsic reward  $f$  from environment
12:      Obtain intrinsic reward  $r(s, a, s')$  from internal critic
13:      Store transition  $(\{s, g\}, a, r, \{s', g\})$  in  $\mathcal{D}_1$ 
14:      UPDATEPARAMS( $\mathcal{L}_1(\theta_{1,i}), \mathcal{D}_1$ )
15:      UPDATEPARAMS( $\mathcal{L}_2(\theta_{2,i}), \mathcal{D}_2$ )
16:       $F \leftarrow F + f$ 
17:       $s \leftarrow s'$ 
18:    end while
19:    Store transition  $(s_0, g, F, s')$  in  $\mathcal{D}_2$ 
20:    if  $s$  is not terminal then
21:       $g \leftarrow \text{EPSGREEDY}(s, \mathcal{G}, \epsilon_2, Q_2)$ 
22:    end if
23:  end while
24:  Anneal  $\epsilon_2$  and  $\epsilon_1$ .
25: end for

```

במאמר הוצג שימוש באלגוריתם על מנת ללמד את המשחק "Montezuma's Revenge". להלן השוואת אלגוריתם DQN:

Total extrinsic reward



ניתן לראות שגם אחרי 2.5M צעדים, אלגוריתם-h-DQN לא מצליח לפתור את השלב. תוצאה זו אינה מפתיעה, מכיוון שעלה מנת לפתרור את השלב נדרש לבצע מספר משימות בצורה טורית. אלגוריתם DQN מtabstס על ה-reward המתקבל מההסביבה, ובשביל להצלחה הוא חייב להצליח לפחות באחוז קטן מהפעמים לקבל סיגナル חוזה. ולכן הוא לא מצליח ללמידה.

בעיות מהעולם האמיתי

יצירה אוטומטית של pipeline עבור אלגוריתמי ML באמצעות RL, הוא דוגמה לבעה עם מספר מרכיבים ופעולות גדולים. ניתן לסוג את אלגוריתמי ה-pipeline האוטומטי לשתיים:

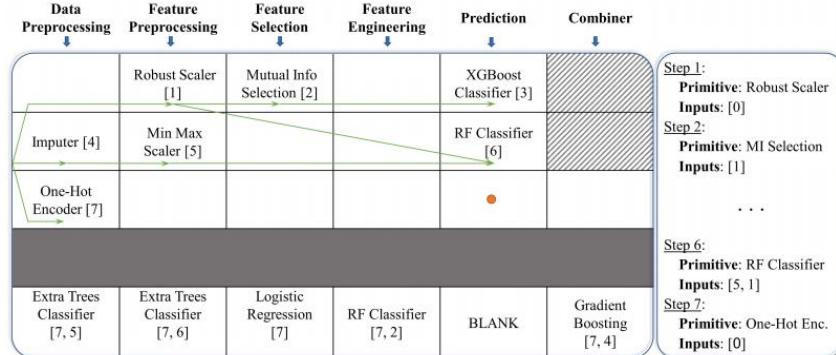
- מרחב מוגנה (constrained space) – בהם קיים שלד של רכיבים שנitin להשתמש בהם בכל שלב
- מרחב לא מוגנה (unconstrained space) – בהם אין אילוצים (או שיש מספר קטן של אילוצים) על הרכיבים בהם ניתן להשתמש בכל שלב, ואין "שלד" של מבנה ה-pipeline.

עוד דוגמה לבעה דומה הוא neural architecture search – חיפוש ארכיטקטורת (מספר שכבות, מספר פילטרים, גודל פילטר, stride וכדו) המתאימה לבעה מסוימת. אלגוריתם AlphaD3M משתמש באלגוריתם דומה לזה של AlphaGO ל-automatic pipeline generation.

מאמר נוסף בתחום זה הוא : DeepLine: AutoML Tool for Pipelines Generation using Deep" (מאמר מבית המחלקה להנדסת מערכות מידע בבן גוריון). במאמר מגדרים את משפחות הרכיבים הבאות :

- Data pre-processing
- Feature pre-processing
- Feature selection
- Feature engineering
- Classification & regression
- Combiners

להלן תרשימים של התהילה:



הסוכן משתמש במאפיינים (features) הבאים :

- \bar{G}_p – מייצג את אוסף אבני הבניין שנעשה בהן שימוש עד כה (למשל data preprocessing, feature selection וכדו')
- \bar{G}_{in} – הקשנותות של ה-flow שהוגדר עד כה, או במילים אחרות המסלול של ה-data ב-pipeline
- \bar{P}_m – מטיא-דאטא על הטופולוגיה של המסלול שהוגדר עד כה, למשל מספר הקודקודים והקשנותות העומק של הגראף וכדו'
- \bar{O}_m – מידע על ה-dataset, למשל מספר הדגימות, מספר המאפיינים וכדו'
- L_j – נתוניים על המטללה (למשל קלסיפיקציה או רגרסיה) ועל המטריקות שאוונן מודדים (למשל F1 accuracy)
- \bar{A}_c – סט הפעולות שהסוכן יכול לבצע, למשל הוספה של בלוק, הורדה של בלוק או שינוי פרמטרים בבלוק ספציפי

שם הסוכן הוא N-D-DQN, והוא משתמש במאפיינים אלה לשערוך ה-Q-function. להלן סכמה של הארכיטקטורה שלו :

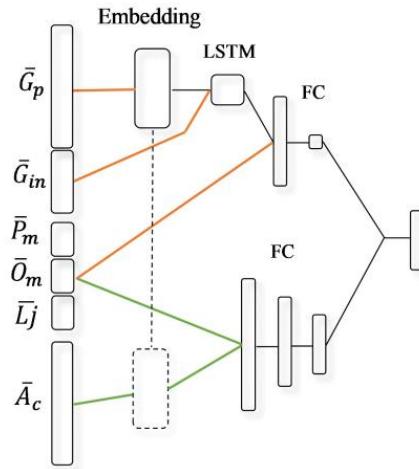


Figure 3: The D-DQN agent's NN architecture. The top stream is the *value function component* and the bottom stream is the *actions advantage function component*.

בעיות נוספות בעלות מרחבי מצבים ופעולות גדוילים שניתן לגשת אליהן באמצעות כלים RL :

- בקרת תנועה בכבישים
- Resource-efficient Malware Detection
- ניהול תורים

Branching Dueling Q-Networks (BQN)

המאמר "Action branching architectures for deep reinforcement learning" מציג שיטה להתמודד עם מרחב פעולה גודל, על ידי שימוש במספר רשות אשר כל אחת תתמחה על חלק ממרחבי הפעולות. בשיטה זו, המצב (state) נשלח לרשות משותפת אשר לומדת ייצוג שלו, והייצוג נשלח לכל אחת מתרתי הרשות אשר מתמקדות בפעולות ספציפיות. שיטה זו מתאימה לביעות בהן ניתן לחלק את מרחב המצבים לתחומי בעיות מוגדרות – למשל תנועה של גוף האדם יכולה להתחלק לתנועה של כל אחד מהמפרקים.

במאמר נעשہ שימוש ב-BQN-dueling double DQN. כזכור, double DQN היא שיטת אופטימיזציה בה הרשת שאותה מאפטמים בוחרת את ה- $a = \arg \max_a Q(s, a, \theta)$ של אותה פעולה a משוערת על Dueling DQN דו-פעמייה. דבר זה נעשה על מנת להתגבר על אופטימיות היתר שיש באימון DQN סטנדרטית. DQN היא שיטה בה מפרידים את הפרדיקציה של ה-state value ושל ה-action advantage לשני ענפים נפרדים כך ניתן לבצע הפרדה בין ערך ה-state לערך של כל אחת מהפעולות ביחס לשאר.

כפי שנitinן לראות באירור, ה-state עובר לרשות אשר מוציאה ייצוג משותף לתחומי הרשות. בענף אחד מחושב ה-state value הכללי, וכל תחת רשות מחשבת את ה-action advantage של כל אחד מהפעולות שמוכנות בה. לאחר מכן מכך ה-state value הכללית מטופסת(action advantage) של כל ענף ויוצרת Q-function עבור כל ענף. לבסוף נבחרות הפעולות בעלות ה-Q-value הגבוהה ביותר. שימוש זה דומה ל-dueling DQN בסיסי במבנה הזה שהוא מפריד בין ה-state value ל-action. החידוש הוא שכן ה-state value משותף לכל תתי הרשות. כמו ב-BQN-dueling DQN, ה-action advantage state value ו-ה-state value-dueling DQN מוחברים דרך שכבת אינרגזיה, אשר מחסרת מה-state value את הממוצע (כך שהממוצע יהיה שווה לאפס), ומחברת את ה-action advantage.

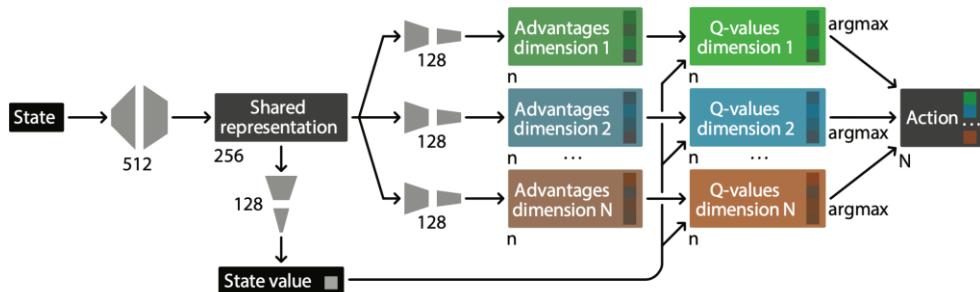
חישוב ה-TD error התבצע באמצעות ממוצע על הענפים השונים :

$$y = r + \gamma \frac{1}{N} \sum_d Q_d^-(s', \arg \max_{a'_d \in A_d} Q_d(s', a'_d))$$

כאשר Q_d^- היא ה-target loss (מtówך target network). (double DQN) ו- y חושב בצורה הבאה :

$$L = \mathbb{E}_{(s, a, r, s') \sim D} \left[\frac{1}{N} \sum_d (y_d - Q_d(s, a_d))^2 \right]$$

כאשר D הוא ה-buffer reply.

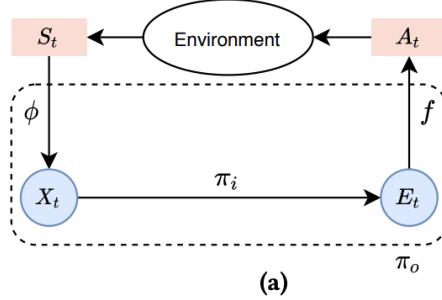


במאמר מראים ביצועים טובים מאוד על tasks שונים.

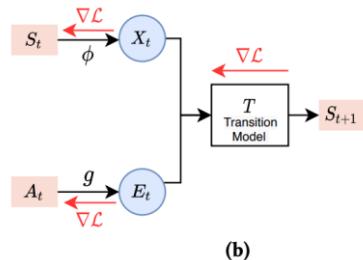
Jointly-Learned State-Action Embedding

ישנים מאמריהם שמארים איך ניתן ללמידה ייצוג לפועלות ולמצבים במרחב embedding. עם זאת, ברוב המקדים המרחב של הפעולות והמרחב של המצבים אינם נלמדים ביחד, ואינם לוקחים בחשבון את הקשר בין הפעולות למצבים. במאמר "Jointly-Learned State-Action Embedding for Efficient Reinforcement Learning" מציעים שיטה ללמידה של שני המרחבים הללו ביחד. דבר זה מאפשר הכללה טובה יותר במרחב מצבים ופעולות גדולים. היתרון בשיטה שהם מציגים זה שהוא שnitן לישם אותה בכל אלגוריתם פוליסת מושסשת גרדיאנטים.

השיטה מורכבת מהפונקציות הגזירות (רשנות) ϕ אשר ממפה מרחב המצבים למרחב ה-embedding f , אשר ממפה מרחב ה-embedding למרחב הפעולות ופוליסת π אשר ממפה מצב פעולה בתוך מרחב ה-embedding. להלן תרשים:



שיטה זו מאפשרת בצורה implicit ליצור מרחב משותף למצבים ולפעולות, תוך כדי למידת פוליסת π . עם זאת, לא ניתן ללמידה את הפונקציות בצורה ישירה. לצורך כך מגדרים את g אשר ממפה מרחב הפעולות למרחב ה-embedding (f -embedding), ואת T אשר ממפה מצב + פעולה (שניהם במרחב ה-embedding) למצב הבא (במרחב המצבים). בהינתן דוגמאות $(s_t, a_t) \rightarrow s_{t+1}$ ניתן לחשב את המודלים T , g , ϕ , כפי שown בairo:



לאחר מכן, ניתן לחשב את f בעזרת זוגות של A_t, E_t שחושו בפונקציה ההיפוכית – g .

Learning in Latent Space

עד כה דיברנו על מקרים בהם המצב (*state*) ידוע. עם זאת, בהרבה מקרים אין לנו מידע מדויק על המצב, ויש בידנו רק *observations* – שහן כМОבון פונקציה של המצב, אך לא תמיד פשוט להסיק מהן את המצב. אפשר להשתמש גם בשיטות *model based* וגם בשיטות *model free* למידה על בסיס *observations*.

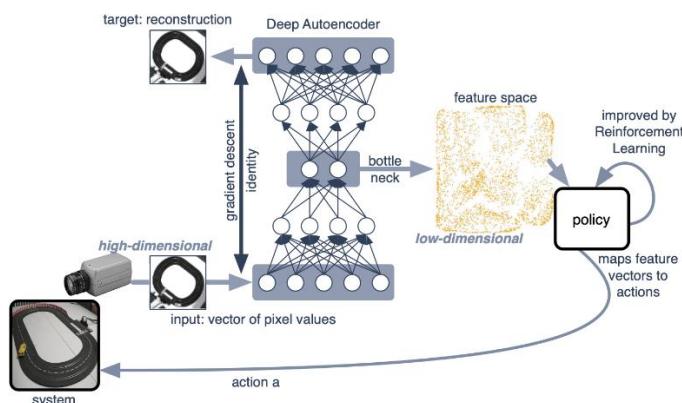
Model-Free Approaches for Learning the Latent Space

ב-*model free* אנו לומדים פוליסיה ישירות מה-*observation*, ומתעלמים מהמצב. אפשר לחילופין ללמידה ב-*embedding* של *observations* ולבצע את למידת-*RL* במרחב ה-*embedding*.

דוגמה אחת לבעה כזו היא מוצגת במאמר "Autonomous reinforcement learning on raw visual input", בו נלמדת בקרה על מכונית מושך קטנה כאשר ה-*input* הוא *data in a real world application* תמונה מצלמה את המסלול, ללא מידע על הדינמיקה של העולם. במאמר פותרים את הבעיה באמצעות *Q-learning*. האלגוריתם מורכב משלושת השלבים:

1. שימוש בפוליסיה SMBCTU *exploration* על מנת לאסוף *data* על הדינמיקה של העולם.
2. שימוש ב-*auto-encoder* להעברת התמונה לוקטור מאפיינים ממוקד יותר.
3. שימוש ב-*Q-learning* ללמידה הפוליסיה (נעשה שימוש בפריים אחד בכל צעדי, ללא אינפורמציה מה עבר).

להלן תרשימים של השיטה:



היתרון בשיטה הוא שהסוכן מצליח ללמידה בצורה יעילה על בסיס תמונה בלבד. החסרונות הם שלא מובטח שה-*autoencoder* יחלץ את המאפיינים הרלוונטיים (הוא מחלץ את המאפיינים שישוו לו לעשות שחזור של תמונה ה-*RGB*, ואני מוכו דווקא לחייב מאפיינים שייעזרו בבעיה שלנו). בנוסף, פתרון זה מתאים יותר ל-*model free methods* שבهم אנו עושים חיזוי ישיר של הפעולה מהדגם. פתרונות אלו בדרך כלל יותר קשים ללמידה, וההתקנסות איטית יותר.

Model-Based Approaches for Learning the Latent Space

דוגמה אחרת היא המאמר "Deep Spatial Autoencoders for Visuomotor Learning", שבו מביצעים בקרה לירוע רוביוטית באמצעות מצלמה. השיטה במאמר היא *model based*, ולמדת הדינמיקה של העולם. האלגוריתם עצמו בו משתמשים במידה נקראה *Linear Gaussian Controllers* והוא מתואר להלן:

Algorithm 1 RL with linear-Gaussian controllers

```

1: initialize  $p(\mathbf{u}_t | \mathbf{x}_t)$ 
2: for iteration  $k = 1$  to  $K$  do
3:   run  $p(\mathbf{u}_t | \mathbf{x}_t)$  to collect trajectory samples  $\{\tau_i\}$ 
4:   fit dynamics  $p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t)$  to  $\{\tau_j\}$  using linear regression
       with GMM prior
5:   fit  $p = \arg \min_p E_{p(\tau)}[\ell(\tau)]$  s.t.  $D_{KL}(p(\tau) || \bar{p}(\tau)) \leq \epsilon$ 
6: end for

```

נלמד מודל לינארי ($\mathbf{u}_t | \mathbf{x}_t$) p אשר חוזה את הפעולה \mathbf{u}_t בהינתן המצב \mathbf{x}_t . בכל איטרציה, נאסף מסלול $\{\tau_i\}$ באמצעות הPOLISHE ($\mathbf{u}_t | \mathbf{x}_t$) p שנלמד עד כה. באמצעות המסלול נלמד מודל של העולם ($\mathbf{u}_t, \mathbf{x}_{t+1}$) p עד כדי מודל לינארי (רגרסיה לינארית). באמצעות המודל שנלמד, מעודכנת הPOLISHE ($\mathbf{u}_t | \mathbf{x}_t$) p שתביא *loss* מינימלי, תחת האילוץ שהPOLISHE לא תשנה יותר מדי (נעשה שימוש ב-*KL divergence*). האילוץ קיים משיקולי בטיחות, על מנת שלא ייצור POLISHE לא צפופה שונה מאוד מהPOLISHE הקיימת. בדומה איטרטיבית מעודכן לחילופין המודל של העולם והPOLISHE.

השיטה השלמה מורכבת ממחצדים הבאים :

1. בשלב הראשון מתבצע אימון ראשוני של מודל, על ידי השיטה שתוארה לעיל. במודל זה, מרחב הממצבים **אינו** **כולל מאפיינים מבוססי מצלה**, אלא רק מיקומים וזווית של חלקי הרובוט ונגורות זמניות שלהם. מודל זה אינו מומצא במיוחד, אך געשה בו שימוש לaiソフ להמשך האלגוריתם.
2. אימון של *spatial autoencoder* על בסיס ה-*data* (הכולל תמונות מצלה) שנאסף ממהודל מההסיע *state vector encoder*. הקודם מחליף מאפיינים אשר מייצגים את מיקום האובייקטים בתמונה וה-*AE*-החדש מרכיב מומצא ה-*AE* אשר מושרשר למאפייני הבקר (זיווית, מהירות וכדו'). בנוסח, מחושב ה-*cost function* כ碼חיק של מומצא ה-*AE* של התמונה בשלב זה מומצא ה-*AE* של התמונה במצב הסופי שאליו מעוניינים להגעים.
3. אימון של סוכן בהתבסס באמצעות אותו האלגוריתם (*model based*) על וקטור המצב החדש (אשר מכיל מאפיינים שחולצו מהתמונה בנוסף למאפייני המונעים).

הרצינול מהחורי השיטה בה אימנו את ה-*autoencoder* בסעיף 2 הוא שעליו לקודד את **נתוני המיקום של האובייקטים בתמונה**, בעוד ש-*autoencoder* בדרך כלל מתרכו בקידוד מה **קיים בתמונה**. על מנת "להכריח" את הרשות להתרכו במלידה של מאפיינים מרחביים, נעשה שימוש ברעיון הבא. לאחר שלושת שכבות הקונבולוציה הראשונות של ה-*encoder*, מתבצעת פעולה *softmax* על כל ערוֹץ בנפרד :

$$s_{cij} = \frac{\exp \frac{a_{cij}}{\alpha}}{\sum_{i'j'} \exp \frac{a_{ci'j'}}{\alpha}}$$

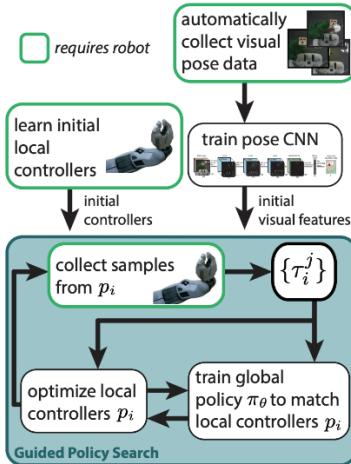
כאשר a_{cij} הוא ערך האקטיבציה של פיקסל j , i , c (*feature*) בערוֹץ (i, j) והוא פרמטר טמפרטורה (היפר פרמטר). לאחר מכן, מחושב ממוצע משוקלל של **המיקומים של האקטיבציות**:

$$\mathbf{f}_c = \left(\sum_i i \cdot s_{cij}, \sum_j j \cdot s_{cij} \right)$$

כאשר j, i הם האינדקסים של הפיקסלים. בדומה לכך, מתקבל וקטור מאפיינים \mathbf{f}_c (בממד השווה במספר הפילטרים של שכבת הקונבולוציה השלישי), אשר כל מאפיין מכיל בתוכו את המיקומים (הממווצעים) של האקטיבציות של כל ערוֹץ. וקטור זה הוא הוקטור שבו נעשה שימוש למידת הבקעה.

לבסוף, ה-*decoder* מורכב משכבה *FC* פשוטה, אשר משוחררת תמונה שערבה *downscaling*.

עוד שיטה במאמר "End-to-end training of deep visuomotor policies" מוצגת במאמר *"model based"* (אותה קבוצת מחקר). במחקר זה, מציגים שיטה שבה מתבצע *pre-training* על כל חלק של האלגוריתם בנפרד – ה-*CNN* והbakr. את ה-*CNN* אימנו לחזות מיקום של אובייקטים שונים בסצנה, ואת הבקר אימנו לבצע את פעולות הבקרה בצורה בסיסית, ולאחר מכן חיברו את הכל ואמנו *end-to-end*. בנוסח, ל-*CNN* השתמשו במודל שאומן על *ImageNet*. סכמה של השיטה מוצגת בתרשימים הבא:



השיטה מורכבת משני חלקים מרכזיים. הראשון הוא **רשות שמאומנת כ-*supervised learning***, שחוצה פולישה מהצורה $(\mathbf{o}_t^\pi, \mathbf{x}_t^\pi | \mathbf{u}_t^\pi)$, כאשר הפונקציות הנלמדות הן $(\mathbf{o}_t^\pi, \Sigma^\pi(\mathbf{o}_t^\pi | \mathbf{u}_t^\pi))$, כאשר $\Sigma^\pi(\mathbf{o}_t^\pi | \mathbf{u}_t^\pi)$ היא רשת *CNN* ו- Σ^π נלמדת בצורה בלתי תלולה באמצעות אובייקטיבית. החלק השני הוא **סוכן RL** אשר יוצר התפלגיות $(\mathbf{x}_t, \mathbf{u}_t)$ אשר נקראות *guiding distributions*, שבעזרתן מאמנים את הפולישה באמצעות *supervised learning*. במהלך האימון, יוצרים מסלולים באמצעות $(\mathbf{x}_t, \mathbf{x}_{t+1}, \dots, \mathbf{x}_T, \mathbf{u}_t, \dots, \mathbf{u}_T)$ אשר מדמים את הסביבה הפיזית. בצורה זאת ניתן לאמן מודל גס בלי לאסוף *data* מהסביבה הפיזיקלית.

הבעיה בשימוש בשיטות *supervised learning* ב-*RL* הוא בעיית ה-*compounding error* (שהוזכרנו בפרק על *imitation learning*), שכן ברגע שהתחזויות יוצאות מההתקבולה של *data* שהאלגוריתם אומן עליו – ברוב המקרים הוא יתבדר. כדי להתגבר על כך, מבצעים בצורה איטרטיבית אימון של הרשות ועדכו של המודל הפיזיקלי (*trajectory centric RL*).

הרשות מורכבת מ-7 שכבות (גדול יחסית לאלגוריתמי *RL*), עם 92,000 פרמטרים. כמו בעובדה הקודמת, גם כאן נעשו שימוש ב-*spatial softmax* על מנת לחוץ מאפייני מיקום. על מנת לשמור את אינפורמציית המיקום בצורה מיטבית, הרשות אינה מבצעת *pooling*.

לסיכום, היתרונות בשימוש ב-*latent space model based* ללמידה:

- יעיל בלמידת יכולות מורכבות, מכיוון שהוא יכול להתבסס על מודל של העולם שהוא יותר מותמצט
- הוא יכול ללמידה קשרים מורכבים יותר בין מאפיינים וכך ללמידה ביעילות *tasks*
- ייצוג מובנה יכול לעזור ללמידה להיות יעילה יותר, שכן המידע יותר מזוקק

חרוגונות:

- כאשר משתמשים ברשות (למשל *autoencoder*) לחילוץ מאפיינים, לא מובטח שהמאפיינים שיחולצו יהיו רלוונטיים ללמידה

Embed to Control (E2C)

הוא מאמר לנו אשר משתמש ב-*Variational Auto Encoder*-latent space ל-*RL*. שיטה זו היא *Model based*, והיא מייצגת את המצב (state) ב-*latent space* (state) מממד נמוך. מטרת המאמר הוא בקרה של מערכות מורכבות באמצעות מידע גולמי מממד גבוה (תמונות).

במאמר מנים מודל מהצורה :

$$s_{t+1} = f(s_t, u_t) + \xi, \quad \xi \sim \mathcal{N}(0, \Sigma_\xi)$$

כאשר ξ הוא הרעש של המערכת, ו- $f(s_t, u_t)$ הינה הדינמיקה של הסביבה. בנוסף, ישנה גישה אך ורק לצלמה x_t . המטרה היא למצוא פונקציה ω + $z_t = m(x_t) +$ המפה מהמרחב של x_t למרחב בעל מממד נמוך יותר(ω הוא הרעש של המערכת), כך שהבקרה תתבצע בממד הנמוך. מגדרים את פונקציית המעבר מ מצב למצב בממד הנמוך :

$$z_{t+1} = f^{lat}(z_t, u_t)$$

את המודל של הסביבה ניתן ללמידה בממד נמוך על ידי לינאריזציה של הבעה (הופכים את הבעה לlienearita למקוטען) :

$$z_{t+1} = A(\bar{z}_t)z_t + B(\bar{z}_t)u_t + o(\bar{z}_t) + \omega$$

כאשר ω הינה ה_offset של $A(\bar{z}_t)$ ו- $B(\bar{z}_t)$ והוא ה- $o(\bar{z}_t)$. לא נתעמק בפתרון, אך נאמר שלאחר הלינאריזציה ניתן להגיע לפתרון מקרוב ממציאת הפעולות והמצבים אשר מבאים למינימום את פונקציית המchiaר. להלן סכמה של השיטה :

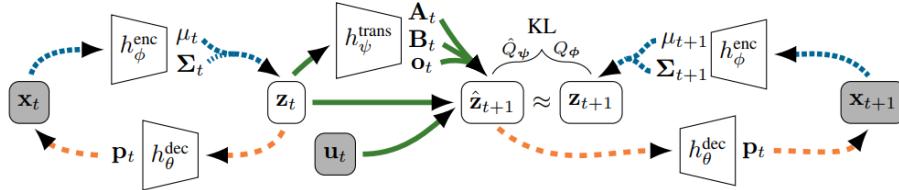


Figure 1: The information flow in the E2C model. From left to right, we encode and decode an image x_t with the networks h_ϕ^{enc} and h_θ^{dec} , where we use the latent code z_t for the transition step. The h_ψ^{trans} network computes the local matrices A_t, B_t, o_t with which we can predict \hat{z}_{t+1} from z_t and u_t . Similarity to the encoding z_{t+1} is enforced by a KL divergence on their distributions and reconstruction is again performed by h_θ^{dec} .

כפי שניתן לראות, בשיטה זו נלמד גם הייצוג במרחב הנמוך על ידי ה-VAE, וגם הדינמיקה של המודל על ידי הרשת h_ψ^{trans} .

Action-Conditional Video Prediction

המאמר 'ההמודד עם מרחב מצבים גדול – תמונות. הרעיון שלו הוא ליצור רשות שחוצה את הפריים הבא, על בסיס הפריים הקודמים והפעולה.

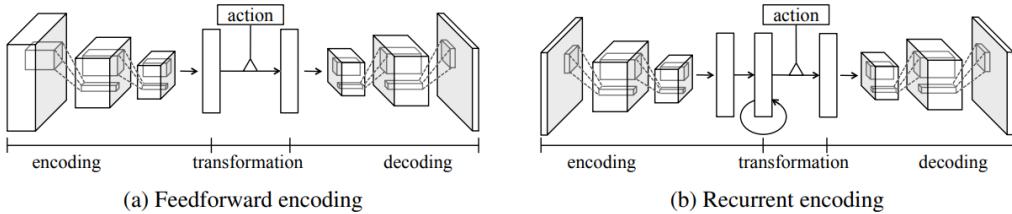


Figure 1: Proposed Encoding-Transformation-Decoding network architectures.

באյור ניתן לראות שתי ארכיטקטורות שונות עם אותו רעיון. בארכיטקטורה הראשונה מחברים m פרויימים קודמים לאחר ה-*encoder*, *encoder*-*concatenate* (*concatenate*) השניה משתמשים ב-*LSTM* אשר מקבל את מוצא ה-*encoder* *decoder* פרויימ אחר פרויימ, ווחזק את הפרויימ הבא. את הפעולה הם מכנים באמצעות מכפלה אבר אבר של מוצא ה-*decoder* במטריצה נלמדת עבור כל פוליה. בנוסף, מתבצע *curriculum learning* על פני הזמן, כאשר בהתחלה הרשת לומדת לחזות רק את הפרויימ הקרוב וככל שהאימון מתקדם הוא חוזה יותר קדימה.

בקשרי *RL*, הציגו במאמר שתי מטריות: לנסות לבצע *control* באמצעות התחזית של הפרויימים (במקום פרויימים אמייטיים) ולבצע אקספלורציה באמצעות סימולוץ של פעולות שונות באמצעות הרשת המאומנת. להלן תוצאות של אימון *DQN* על הפרויימים שיצרה הרשת, במקום לקבל פרויימים מהמשחק:

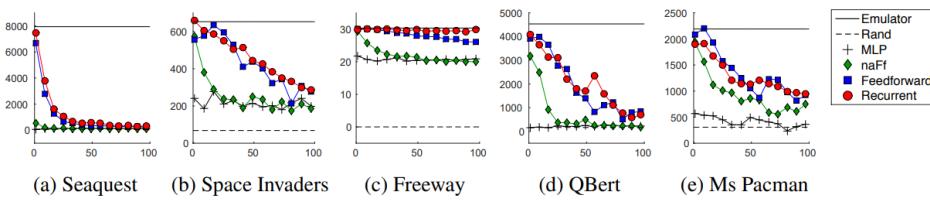


Figure 5: Game play performance using the predictive model as an emulator. ‘Emulator’ and ‘Rand’ correspond to the performance of DQN with true frames and random play respectively. The x-axis is the number of steps of prediction before re-initialization. The y-axis is the average game score measured from 30 plays.

ה-*DQN* מקבל רק פרויימים מסומלצים. ציר x מצין אחרי כמה פרויימים מתחילה את הרשת שייצרת את הפרויימים עם פרויימ אמייטי שמננו המשיך לחזות פרויימים. ניתן לראות כי למגוון שהביזועים יורדים, חלק מהמקרים עבר אופקים אירוניים לא רחוק מידי, ניתן ללמוד גם באמצעות הסימולציה.

הניסוי השני שהתבצע נקרא *informed exploration*. נטען במאמר שבמוקם לבצע אקספלורציה באמצעות ϵ -greedy, ניתן לשפר אותה באמצעות סימולציה. ספציפית, בכל שלב שבו הגיע תור האקספלורציה (בהתבסירות ϵ), בדקו את התמונה שתתקבל עבור כל אחת מהפעולות, והשוו אותן מול מחסנית של d פרויימים אחרים שנשמרו לטובה זה. את ההשוואה עשו באמצעות *kernel* גaussi:

$$n_D(s^{(a)}) = \sum_{i=1}^d k(s^{(a)}, s^{(i)}); k(x, y) = \exp\left(-\frac{1}{\sigma} \sum_j \min\left(\max\left((x_j - y_i)^2 - \delta, 0\right), 1\right)\right)$$

הראו במאמר ששיטה זו עזרה לאקספלורציה ושיפרה את הביצועים ברוב המקרים. להלן תמונה של האקספלורציה עם ו בלי השיטה:

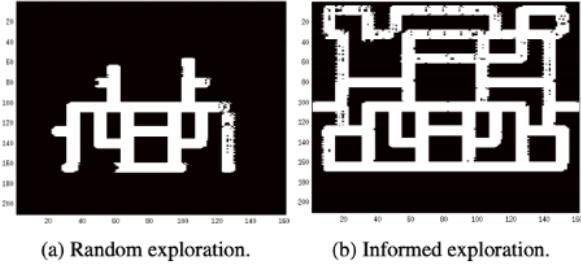


Figure 6: Comparison between two exploration methods on Ms Pacman. Each heat map shows the trajectories of the controlled object measured over 2500 steps for the corresponding method.

להלן האלגוריתם:

Algorithm 1 Deep Q-learning with informed exploration

```

Allocate capacity of replay memory  $R$ 
Allocate capacity of trajectory memory  $D$ 
Initialize parameters  $\theta$  of DQN
while  $steps < M$  do
    Reset game and observe image  $x_1$ 
    Store image  $x_1$  in  $D$ 
    for  $t=1$  to  $T$  do
        Sample  $c$  from Bernoulli distribution with parameter  $\epsilon$ 
        Set  $a_t = \begin{cases} \operatorname{argmin}_a n_D(x_t^{(a)}) & \text{if } c = 1 \\ \operatorname{argmax}_a Q(\phi(s_t), a; \theta) & \text{otherwise} \end{cases}$ 
        Choose action  $a_t$ , observe reward  $r_t$  and image  $x_{t+1}$ 
        Set  $s_{t+1} = x_{t-2:t+1}$  and preprocess images  $\phi_{t+1} = \phi(s_{t+1})$ 
        Store image  $x_{t+1}$  in  $D$ 
        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $R$ 
        Sample a mini-batch of transitions  $\{\phi_j, a_j, r_j, \phi_{j+1}\}$  from  $R$ 
        Update  $\theta$  based on the mini-batch and Bellman equation
         $steps = steps + 1$ 
    end for
end while

```

לסיכום, יתרונות השיטה:

- נראת שמצליחים לחזות לא רע את הפריים הבאים
- האקספלורציה אכן משתפרת על ידי שיטת *informed exploration*

חסרונות:

- לא ברור כמה שיטה זו תעבור במערכות עולם אמיתי, למשל תМОונות ממצלמה אמיתית ולא משחק
- לא ברור כיצד ניתן להשתמש בשיטה זו ל-*planning*, בין השאר כי חסר מודד טוב לבחינת ביצועים (*MSE*)

Inverse Reinforcement Learning

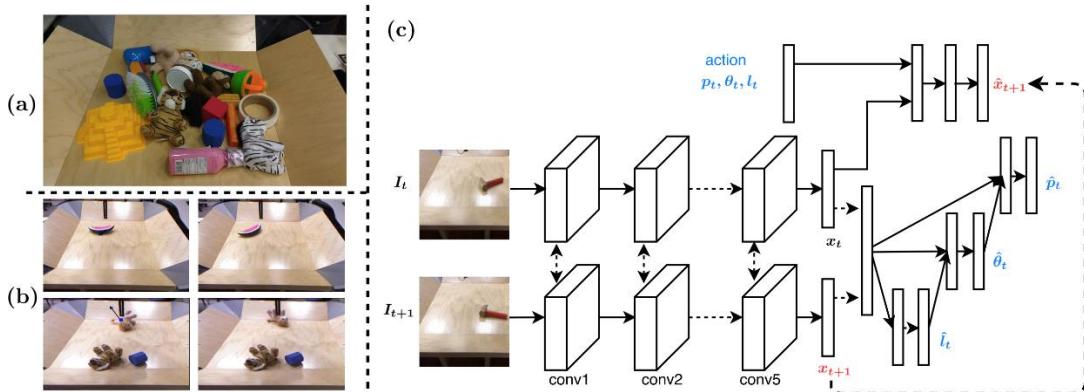
המאמר "Learning to poke by poking: Experiential learning of intuitive physics" מציע שיטה מעניינת ללמידה *latent space*-מתקן תМОונה. הרעיון הוא שניתן להיעזר **במייפוי ההפוך** כЛОMER לחזות את הפעולה שהתבצעה מזוג תМОונות (אחת לפניו הפעולה ואחת אחרת). הסיבה לכך היא שתוחזית ישירה, כפי שנעשה במאמרים הקודמים שהציגו, היא מאד קשה. סיבה נוספת לכך היא שקשה לרשף לבצע בניהה של תМОונה, במיוחד כשמדבר בעולם האמיתי. וסיבה שנייה, שהרבה מהמאפיינים שהרשף יוצרת בתМОונה הם לא רלוונטיים, והרשף מתרכז בחלקים הפחות חשובים בעולם ה-*RL*.

במאמר משתמשים ב Robbins-Shimony חפצים על גבי שולחן. מציגים שני מודלים : $\hat{x}_{t+1} = F(x_t, u_t; W_{fwd})$ וה- $inverse model$ אשר חוזה את המצב הבא x_{t+1} בהינתן המצב הנוכחי x_t והפעולה u_t , והוא $\hat{u}_t = G(x_t, x_{t+1}; W_{inv})$ אשר חוזה את הפעולה שהתבצעה בין שני מצבים. המודל שמעוניין אותנו לחזות הוא ה- fwd , שכן נרצה לדעת את הפעולה שעשינו לבצע כדי לעבור במצב (כך יוכל להשתמש בזמנים מתחום $model based RL$). עם זאת, למידה זו קשה, שכן מושג החזות במדוקיק את כל הפיקסלים של המצב החדש, ובנוסף, ה- $reconstruction$ הבלתי מלא לא מעוניין אותנו, שכן מה חשוב הוא המאפיינים שקשורים למעבר במצב. מצד שני, את ה- $inverse model$ קל יותר ללמידה, שכן יותר טבעי לקבל שתי תמונות ולהחזיר סקלר (או וקטור של פרדיקציות). הבעה במודול זה היא שניתן להגיע במצב באמצעות מספר פעולות ולכן הלמידה עלולה להיות לא יציבה.

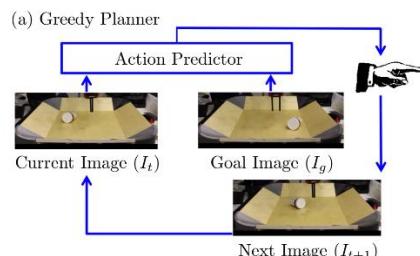
ה- $loss$ במהלך האימון הוא :

$$L_{joint} = L_{inv}(u_t, \hat{u}_t, W) + \lambda L_{fwd}(x_{t+1}, \hat{x}_{t+1}, W)$$

כאשר L_{inv} הוא סכום של $cross entropy$ של שלוות מאפייני הפעולה (זווית, מיקום, אורך), ו- L_{fwd} הוא L_1 בין התמונה הבאה לתמונה החזויה **במרחב המאפיינים**. נציגו שונעיה שימוש לשתי המשימות באוטה רשות עם אותם פרמטרים W . להלן תרשים של השיטה :



כאשר הרובוט מתקבש לעשות מעבר שלא אפשרי בצעד אחד, מתבצע תכנון חמדני (*greedy*) של מספר פעולות עד למטרה. להלן תרשים :



לסיכום, היתרונות בשיטה זו הם :

- מעט מאוד התערבות אנושית בתהליכי הלמידה
- אין צורך לבצע *reconstruction* של התמונה הסופית

חסרונות

- לא ניתן לבצע *planning* עם ה-*inverse model*

- מתרכז בלמידה של הפעולות, ולכן יכול להיות שהרשות לא תקודד מאפיינים חשובים שאינם קשורים לשיטה

סיכום הפרק, להלן יתרונות וחסרונות של שיטות *model based*.

- יחסית קל לאוסף *data* רלוונטי ב-*scale*, מכיוון שלומדים רק את המודל של העולם
- ניתן למודד ממילה למילה, שכן נלמד המודל של העולם, ולא מטרה ספציפית
- בדרך כלל, נדרש כמות קטנה יותר של *data* מתיויג

חסרונות:

- הלמידה לא מאופטמת ממילה ספציפית, אלא מנסים "למוד הכל", שכן יתכן והביצועים על ממילה ספציפית יהיו נמנוכים יותר
- לעיתים קל יותר למודד פוליסיה בצורה ישירה מאשר למודד את המודל של המערכת
- לעיתים הלמידה של הדינמיקה קשה מדי, ונדרש להניח הנחות מוקלות שעולות לפגוע בביצועים

לעומת זאת, היתרונות של *model free*:

- לא נדרש הנחות מלבד ה-*reward function*
- במקרים בהם הpolloסה מורכבת יותר קל למודד אותה ישירות

חסרונות:

- בדרך כלל נדרש הרבה *data*
- לא ניתן להעיביר בצורה קלה את הידע ממילה ממילה

Exploration with Exemplar Model

ושא המאמר "עליה במרחב מצבים גדול. בעיות עם מרחב מצבים קטן, קיימות שיטות רבות לביצוע אקספלורציה באמצעות *counters*, כלומר שמרת מספר הביקורים בכל מצב (*s*) *N* ותוספת "בונוס" ל-*MBIE-EB* לאקספלורציה באמצעות *reward function* כך שיעודד את הסוכן לבצע במצבים בהם הוא לא ביקר. למשל במקרה בשם מושפעים *reward* של $\frac{\beta}{\sqrt{N(s,a)}}$ כדי לעודד אקספלורציה. עם זאת, במקרים בהם מרחב המצבים גדול מאוד שיטות אלה אינן רלוונטיות. עם זאת, ניתן לבצע הרחבת שלוחן, כלומר *reward* למצבים **השוניים** **bijouter** מה המצבים בהם ביקרנו.

בשיטת המוצגת, מציגים שיטה בשם *exemplar models*, אשר בהינתן *dataset* $X = \{x_1, \dots, x_n\}$ מאמנים *n* מודלים (*discriminators*) $\{D_{x_1}, \dots, D_{x_n}\}$ אחד לכל דוגמה, אשר חוזה 1 אם מציגים את הדוגמה המתאימה ו-0 אם מציגים דוגמה אחרת. במאמר עצמו, נעשה שימוש ברשף אחת לכל הדוגמאות, אבל הרעיון המקורי מבוסס על *n* מודלים. ה-*discriminators* מאמנים באמצעות *loss* הבא :

$$D_{x^*} = \arg \max(E_{\delta_{x^*}}[\log D(x)] + E_{P_X}[\log 1 - D(x)])$$

כאשר δ_x הוא ההתפלגות של הדוגמה x (התפלגות של דוגמה אחת), ו- P_X היא ההתפלגות של שאר הדוגמאות. **cut-off** מציגים כיצד *exemplar model* יכול לשמש *density estimator*.

קודם כל, מוכחים במאמר שבמקרה האופטימלי ה-*discriminator* (בבעה הדיסקרטית) יהיה :

$$D_{x^*}(x) = \frac{\delta_{x^*}(x)}{\delta_{x^*}(x) + P_x(x)}$$

כאשר δ_{x^*} היא פונקציית הדلتא של דיראק, ו- $P_x(x)$ היא פונקציית ההסתפלגות של ה-*data* בלבד x^* .
1 אינטואיטיבית, כאשר ההסתברות לקבל את x^* בשאר ה-*data* שואף לאפס, ה-*discriminator* יחזיר ערך 1 (100% הצלחה), ואשר ההסתברות שואפת לאחד (למשל *dataset* שמורכב רק מ- x^*) הערך יהיה 0.5 (50% הצלחה) – הטלה מטבعة. נשים לב, שמדובר בו $x^* = x$ מתקבל $D_{x^*}(x^*) = \frac{1}{1+P_x(x)}$ ולכן ניתן לשער את ההסתפלגות (x^*) על ידי $P_x(x^*)$:

$$P_x(x^*) = \frac{1 - D_{x^*}(x^*)}{D_{x^*}(x^*)}$$

כל זה נכון במקרה ההפוך, אבל במקרה הרציף פונקציית הדلتא של דיראק שואפת לאינסוף ולכן לא ניתן לשער את ההסתפלגות. עם זאת, ניתן לפתור את העניין על ידי טריק מתמטי. אם מוסיפים רעש לדגימה x^* במהלך האימון, ניתן להוכיח (לא נוכחה פה) שההסתפלגות מקיימת :

$$P_x(x^*) \propto \frac{1 - D_{x^*}(x^*)}{D_{x^*}(x^*)}$$

בסוף דבר, ל-*reward*, ל-*discriminator* הסטנדרטי מוסיפים חלק של "novelty", אשר תלוי ב- $(s, D_s(s))$ הנקרא :

$$R'(s, a) = R(s, a) + \beta f(D_s(s))$$

כאשר β הוא היפר פרמטר לכובונו היחס בין ה-*exploration* ל-*exploitation*, ו- f היא פונקציה התלויה ב- $D_s(s)$, אשר נרצה שתהייה בעל ערך גבוה כאשר המצב s נדיר יותר, ולהיפך. במאמר משתמשים בפונקציה $f = -\log p(s)$ (כאשר p תלוי ב- $D_s(s)$). להלן האלגוריתם המלא :

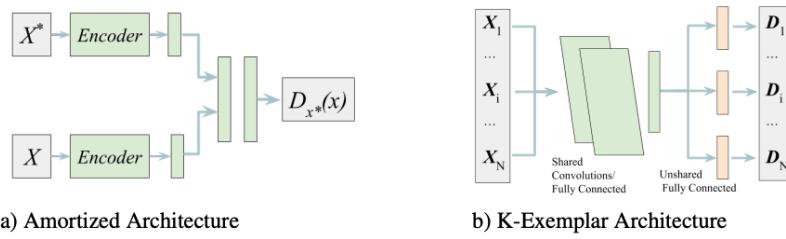
Algorithm 1 EX² for batch policy optimization

```

1: Initialize replay buffer  $B$ 
2: for iteration  $i$  in  $\{1, \dots, N\}$  do
3:   Sample trajectories  $\{\tau_j\}$  from policy  $\pi_i$ 
4:   for state  $s$  in  $\{\tau\}$  do
5:     Sample a batch of negatives  $\{s'_k\}$  from  $B$ .
6:     Train discriminator  $D_s$  to minimize Eq. (1) with positive  $s$ , and negatives  $\{s'_k\}$ .
7:     Compute reward  $R'(s, a) = R(s, a) + \beta f(D_s(s))$ 
8:   end for
9:   Improve  $\pi_i$  with respect to  $R'(s, a)$  using any policy optimization method.
10:   $B \leftarrow B \cup \{\tau_i\}$ 
11: end for

```

לצורך השלמת התמונה, האלגוריתם הבסיסי של *exemplar models* משתמש במודל שונה לכל דוגמה. עם זאת, במאמר נעשה שימוש ברשת ייחודית אשר משערת את ההסתפלגות של (x) בהינתן x^* . להלן סכמה:



Curiosity-driven Exploration by Self-supervised Prediction

Curiosity-driven Exploration by Self-supervised Prediction

המאמר האחרון שנסקרו בנושא זה הוא "Prediction". גם בעבודה זאת, הנושא המרכזי הוא ביצוע אקספלורציה חכמה בסביבה עם ממד מצבים גדול. הרעיון הוא שבעזרת הוספת רכיב **סקרנות** לסוכן, האקספלורציה תתמקד במצבים האינפורטטיביים. סקרנות מבוסנת זה מתבטאת במקרים בהם הסוכן טועה כאשר הוא מנסה לחזות את תוצאות הפעולות שלו על העולם.

הסוכן מורכב משני רכיבים – *reward generator* אשר תפקידו ליצור הקשורים לסקרנות, ופוליסיה שתפקידה למקסם את *the-h*-*reward* זהה. בנוסף, הסוכן יכול לקבל גם *reward* חיצוני המגיע מהסביבה. רכיב הפוליסיה מנסה למקסם את $r_t^i + r_t^e$ כאשר r_t^i הוא *the-h*-*reward* האקטרוני (סקרנות) ו- r_t^e הוא *the-h*-*reward* האקטרוני (מחשבה). r_t^e הוא אפס רוב הזמן (*sparse*). נעשה שימוש ב-*A3C* (נסקר בפרקים קודמים) לצורך אימון הפוליסיה (θ_P ; s ; π).

הכוונים מסבירים מדוע שימוש *raw data* של המצב, כלומר הפיקסלים במקורה של תמונה, לא מתאים לצורך מידול הסקרנות של הסוכן. הם מביאים דוגמה – עצים בעלי עלים שזרים ברוח. במקרה זה, הסוכן לא יכול לחזות לעולם את הפיקסלים הללו (אם הם זרים בצורה אקראית), ולכן ציון הסקרנות יהיה שם לעולם גבוה. סוכן זה יזקן וימوت בעודו צופה בסקרנות בעלי העלים ברוח. במאמר מסווגים את העולם לשלווה *observations*: שהסוכן יוכל להשפיע עליהם, *observations* שהסוכן לא יוכל להשפיע עליהם אף יוכל להשפיע עליהם. סקרנות בשני הסוגים הראשונים יכולה להיות פרודוקטיבית, אך בסוג השלישי לא.

במאמר משתמשים בשיטה שהזכרנו לעיל (למשל *RL-inverse*) – חיזוי הפעולה באמצעות *the-h*-*feature vectors* של שני פרמטרים עוקבים. מmirirs בשלב הראשון את הפרמטרים למרחב מאפיינים ($\phi(s_t)$, $\phi(s_{t+1})$) באמצעות רשות, ובשלב שני חוזים את הפעולה \hat{a} שהובילה מ- s_t ל- s_{t+1} . אימון רשות זאת מאפשרת $-\cdot(\phi)$ לקודד רק את המאפיינים שרלוונטיים לפעולה. מודל זה נקרא *inverse dynamics model* – חיזוי הפעולה מהдинמיקה במקומות לחזות את הדינמיקה מהפעולה. מודל זה הינו אינוריאנטי לפוליסיה, שכן היא מمدלת את הסביבה.

לאחר שלמדנו את *the-h*-*inverse dynamics model*, אנו יכולים להשתמש בקידוד שנלמד $(\cdot)\phi$ על מנת ללמידה את הדינמיקה במרחב המאפיינים *forward dynamics model*:

$$\hat{\phi}(s_{t+1}) = f(\phi(s_t), a_t; \theta_F)$$

לאחר שנלמד המודל f ניתן ליצור את *the-h*-*reward signal*, כלומר את *the-h*-*reward* על הסקרנות, באמצעות:

$$r_t^i = \frac{\eta}{2} \left\| \hat{\phi}(s_{t+1}) + \phi(s_{t+1}) \right\|_2^2$$

האימון של *the-h*-*backward models* ו-*the-h*-*forward models* נלמדים ביחד. להלן תרשימים של השיטה:

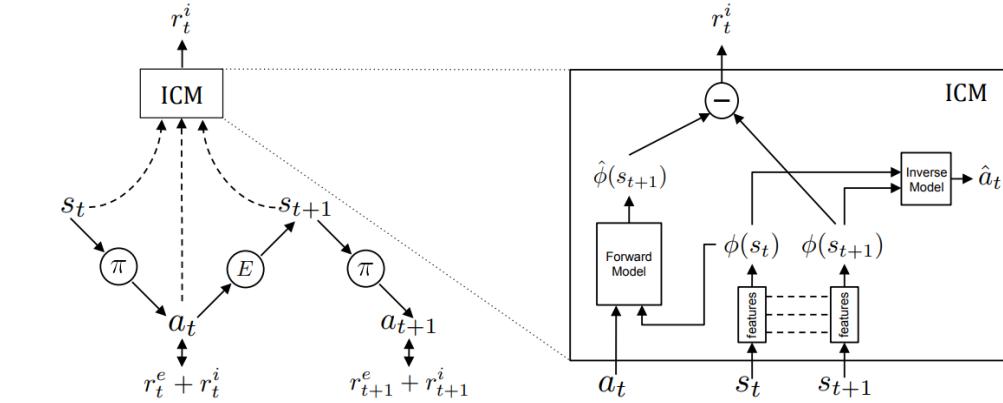


Figure 2. The agent in state s_t interacts with the environment by executing an action a_t sampled from its current policy π and ends up in the state s_{t+1} . The policy π is trained to optimize the sum of the extrinsic reward (r_t^e) provided by the environment E and the curiosity based intrinsic reward signal (r_t^i) generated by our proposed Intrinsic Curiosity Module (ICM). ICM encodes the states s_t , s_{t+1} into the features $\phi(s_t)$, $\phi(s_{t+1})$ that are trained to predict a_t (i.e. inverse dynamics model). The forward model takes as inputs $\phi(s_t)$ and a_t and predicts the feature representation $\hat{\phi}(s_{t+1})$ of s_{t+1} . The prediction error in the feature space is used as the curiosity based intrinsic reward signal. As there is no incentive for $\phi(s_t)$ to encode any environmental features that can not influence or are not influenced by the agent’s actions, the learned exploration strategy of our agent is robust to uncontrollable aspects of the environment.

לסויום, להלן התוצאות :

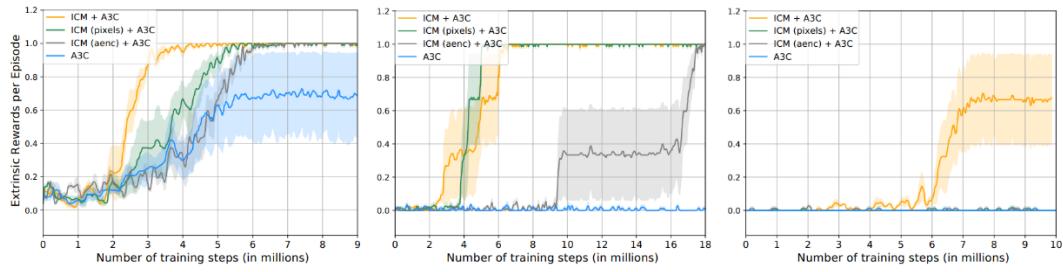


Figure 5. Comparing the performance of the A3C agent with no curiosity (blue), ICM-pixels + A3C (green) and the proposed ICM + A3C agent (orange) in the “dense”, “sparse” and “very sparse” reward scenarios of VizDoom. The curious A3C agents significantly outperforms baseline A3C agent as the sparsity of reward increases. Pixel based curiosity works in dense and sparse but fails in very sparse reward setting. The dark line and shaded area show mean and mean \pm standard error averaged over three independent runs.

Transformers in RL

אף על פי ש-RL מבוסס על *sequential data* רוב המודלים משתמשים בהם לעיבוד *data* כזה בתחום ה-*RNNs*. הסיבות לכך הן טרנספורמרים דורשים כמות גדולה מאוד של *data* כדי להגיע לביצועים טובים ושהם קשים לאופטימיזציה. אך למגוון החסרונות, ישנו פוטנציאל גדול בשימוש בטרנספורמרים ב-RL.

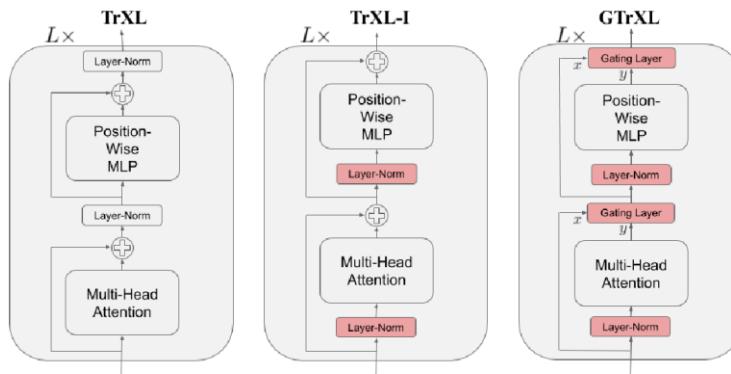
עבודה בסיסית בתחום פורסמה ב-2019 במאמר "Stabilizing transformers for reinforcement learning" בו הם מראים שיפורים והתאמות של טרנספורמרים לביעות RL. במאמר מוצגת הארכיטקטורה *Gated Transformer-XL* אשר מראה ביצועים טובים יותר מ-LSTM ומרשתות מובססות זיכרון במספר מוגבל. בנוסף, ארכיטקטורה זו לומדת מהר יותר ובצורה יציבה יותר מהארQUITקטורה הבסיסית של הטרנספורמר.

השינויים המרכזיים בארכיטקטורה הם :

- שינוי מיקום ה-*normalization*. בארכיטקטורה הקלאסית, הנוורמליזציה מגיעה מיד אחרי ה-*skip connection* בבלוק *skip connection*. בדומה לכך ב-*attention* מועבר את המידע לאחר נורמליזציה. במאמר שינוי מיקום כך שיופיע לאחר *skip connection* כל *attention*. אין שינוי מבחן ה-*MLP* שאחורי *skip connection* העובר ב-*data* ומוגרם (ראה איור למטה).
- שינוי ה-*gating mechanism* כך שתתמכו ברצפים ארוכים יותר.
- הוספת *layer normalization* ושינוי של ה-*gating mechanism*. ניסו במאמר מספר שיטות, בין השאר :

$$g^{(l)}(x, y) = \sigma(W_g^{(l)}y - b) \odot \tanh(U_g^{(l)}y)$$

שיטת זו, בדומה למה שנעשה ב-LSTM ו-GRU, נועדה לווסת את זרם האינפורמציה, ולאפשר לחסום חלק ממנו.



הכותבים הציגו שני מודדים לארכיטקטורה החדשה, זיכרו (אם המודל מצליח לפחות מידע גם *tokens* ו-*rules*) (פעולות מורכבות כמו משחקים *Atari*). המודל הראה שיפור בביצועים בשני המודדים.

Big-Bird ו-Longformer

אחד מהאתגרים בשימוש בטרנספורמרים ב-RL הוא אורך הרצפים. מודלים של שפה עובדים עם כמה מאות מילימ (למשל *BERT* מקבל רצף של 512 tokens), ואילו ב-RL לעיתים הרצפים באורכים ארוכים הרבה יותר. מכיוון שהיחסיות של שכבות *attention* היא ריבועית באורך הקלט, רצפים ארוכים מאוד יכולים להיות אטגר. נסקור שלושה עבדות שמתמודדות עם בעיה זו : *Extended-Big-Bird*, *Longformer* ו-*Transformer Construction (ETC)*.

: *attention patterns* *Longformer*

- – בשיטה זו, משתמשים ב"חלון רץ" לצורך העיבוד, כאשר עבור כל *token* מtabצע עיבוד רק של מספר קבוע של *tokens* לפניו ואחריו.
 - – שילוב של חלון מסביב ל-*token*, *dilated convolution* ו-*sliding window*. בשיטה זו, ה- *receptive field* של העיבוד, החלון מtabצע על חלון מסביב ל-*token*, אך כדי להגדיל את ה-*attention* מtabצע ב"קפיות" כמו הקונבולוציה ב-*dilated convolution*. באמצעות שיטה זו ניתן לעבד אפלו עשרות אלפי *tokens*. ה-*dilation rate* נקבע בצורה שונה בכל שכבה, כאשר בשכבות הראשונות הוא עמוק יותר, על מנת למדוד מאפיינים מקומיים, ובהמשך הוא גדול.
 - – במידה וודעים ש-*tokens* מסוימים חשובים במיוחד (למשל רצף של הودעת *header* יהיה חשוב) ניתן לקבוע *tokens* מסוימים אשר יהיו מחוברים לכל הכניםות שבסכנת ה-*attention*.
- להלן תרשים של ה-*patterns* של *tokens*:

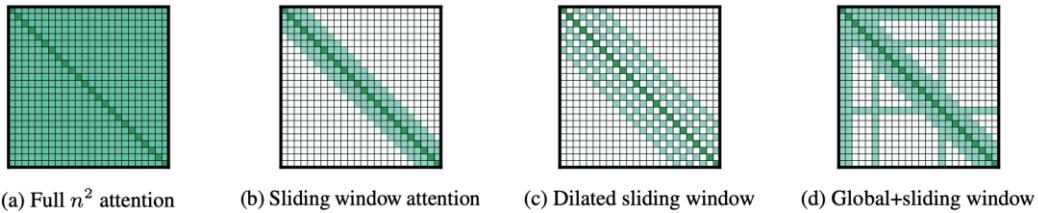


Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

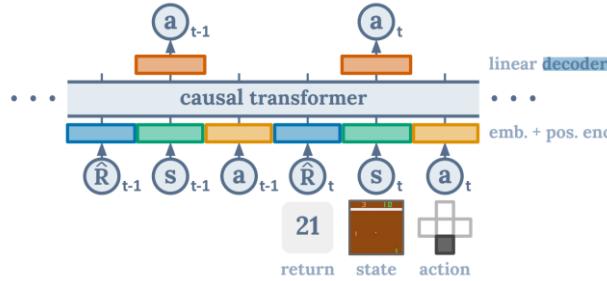
היא עבודה נוספת, שנעשתה באותה הזמן (עם *longformer*), ומtabסת על רעיון דומה. בנוסף לכך – זוגות של *tokens* – *global attention* ו-*local attention*, נעשה שימוש ב-*tokens* שמוגרים, כאשר כל זוג ש"זוכה" בהגלה מתחבר בסכנת ה-*attention*.

The Decision Transformer

שיטה זו משתמשת ב-*episode-based RL* לביעות *transformer* *model free offline RL* ממודול כ-:

$$\tau = (\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, \dots, \hat{R}_T, s_T, a_T)$$

כאשר \hat{R}_t הוא \hat{R} -*return to go* (סכום ה-*rewards* עד סוף ה-*episode*), והמודול לומד לחזות את הפעולות, באמצעות ארכיטקטורה הדומה ל-*GPT* (*generative*). להלן תרשים של השיטה:



הלמידה נעשית בדיקום כמו מודל שפה, באמצעות *masking*, אם כי המודול נדרש לחזות רק את הפעולות. ב-*test time*, המשתמש נותן מודול את ה-*return* המבוקש – קלומר הגובה ביותר, והמודול יכוין אותו לפעולות שיתנו לו את זה. אם המשתמש ייתן מודול *return* לא רalistiy, המודול יתבדר. במאמר מראים ביצועים טובים של המודול במספר מטלות (משחקי *Atari*).

Trajectory Transformer

הבודה זו גם עוסקת ב-*offline learning* ובמידול בעיית *RL* כ-*sequence modeling*. הכותבים מודלים את הבעה בצורה הבאה. *Trajectory* אחד מסומן כ-:

$$\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_T, a_T, r_T)$$

כאשר המצב או הפעולה עם מספר מדדים (למשל N ו- M בהתאם), הם מפרקים גם את הממדים לתוך הרצף בצורה הבאה:

$$\tau = (s_1^1, s_1^2, \dots, s_1^N, a_1^1, \dots, a_1^M, r_1, \dots, s_T^N, \dots, a_T^M, r_T)$$

בנוסף, המבצע נרמול לכל אחד מהמדדים (המ השוו נרמול *Uniform quantile*). הארכיטקטורה בה השתמשו היא *GPT*. *loss*-ו הוא:

$$\mathcal{L}(\tau) = \sum_{t=1}^T \left(\sum_{i=1}^N \log P_\theta(s_t^i | s_t^{<i}, \tau_{<t}) + \sum_{j=1}^M \log P_\theta(a_t^j | a_t^{<i}, s_t, \tau_{<t}) + \log P_\theta(r_t | a_t, s_t, \tau_{<t}) \right)$$

כאשר P_θ היא הרשת (הטרנספורמר). אפשר לשים לב כי כל ממד בפעולות ובמצביים מקבל רק את הממדים שקדמו לו. ב-*test time*, משתמשים באלגוריתם *beam search* על מנת למצוא את הפעולה שתביא לתוצאות הטובות ביותר.

Model-Based Methods

ניתן לחלק את שיטות ה-RL לקטגוריות הבאות:

- Gradient free methods – שיטות שאינן מבוססות גרדיאנטים. השיטה הפחותה ביותר ביוטר היא random search, אבל קיימות שיטות נוספות (בסיסות סטטיסטיקה). שיטות אלו בדרך כלל דורשות יותר דגימות ויתר איטיות עד להתכנסות. נסקור מעט שיטות כאלה בפרק הבא.
- Fully online methods – למשל actor critic, מבוססים על למידה של הסביבה ועל גבי זה של הpolloסה. שיטות אלו יותר יעילות מ-gradient free methods.
- Policy gradients methods – שיטות אלה לומדות את הpolloסה בצורה ישירה, ולכון בדרך כלל דורשות פחות דגימות על מנת להתכנס (מאשר fully online methods).
- Off policy methods – למשל Q-learning, יותר מהירות (ודורות פחות דגימות) משיטות ה-policy gradients, מכיוון שהpolloסה שנלמדת היא האופטימלית, והיא נלמדת במהלך כל הלמידה, גם כאשר הpolloסה שהסוכן משתמש בה עדין לא אופטימלית.
- Model-based DRL – למידה של הדינמיקה של העולם. שיטה זו יותר יעילה משיטות ה-off policy שבהרבה מקרים הסביבה יותר קלה להבנה מאשר המטלה בלי ידיעת הסביבה, ואם הסביבה ידועה קל ללמידה המטלה. נסקור שיטות כאלה בפרק זה.
- Model-based “shallow” DRL – שימוש ברשות רזרווה או באלגוריתם פשוט יחסית ללמידה הסביבה. בצורה כזו את הלמידה מהירה באופן משמעותי.

בפרק זה נרחיב על *model-based DRL*. נציין שלמרות שיטות אלה הן מהירות ויעילות, הן דורשות הנחות מסוימות לגבי המידע שלנו על הסביבה והдинמיקה שלה. אף על פי שהן יותר יעילות מבחינת מספר דגימות, ניתן שלهن יכול להיות מורכב. כאשר הצלחנו ללמידה את הדינמיקה של הסביבה ניתן להשתמש בשיטות פשוטות (אפילו לינהיות) לפתרון בעיית ה-RL. על מנת ללמידה את הדינמיקה ניתן לבצע דוגמה של המרחב וללמוד ממנו, אם כי יכול להיות מושג *model bias*, כלומר שבאזורים מסוימים לאצליח ללמידה את הסביבה, או לפחות לא בצורה מספקת. במקרים בהם המרחב גדול, קשה לדוגום אותו בצורה “צפופה”, ולכן המודל יעריך את הפונקציה במקומות הדילילים. קיימים מודלים אשר יודעים גם להעריך אי-ודאות (במקומות בהם אין מספק *data*). נשתמש במודל (רשת נוירונית) על מנת למדוד את הסביבה. רשות זו צריכה להיות גזירה. להלן נציג מספר שיטות ללמידה של מודל כזה.

גישות ללמידה מודל סביבה

גישה ראשונה:

1. שימוש במודל ראשוני ($a_t | s_t$) לaisוף *data* (מודול זה יכול להיות אקראי).
2. שימוש ב-*data* זה לאימון מודל $\sum_i \|f(s_{i-1}, a_{i-1}) - f(s_i)\|^2$.
3. לאחר מכן ניתן להשתמש במודל על מנת לבחור פעולה. ניתן לבצע זאת על ידי *backpropagation* מモץ המודל, בו נחוצה המצב הבא, עד לכינוסה למודל, בו התקבלה הפעולה, וכך לבחור פעולה אשר תמקסם את הסיכון להגעה למצב הרצוי.

שיטה זו מתאימה לקרים בהם *data* מיוצג בצורה טובה, כך שיוביל ללמידה קלה של הדינמיקה. הבעיה בשיטה זו היא ה-*compounding error drifting*, שכן סטייה קטנה בפעולה עלולה להוביל את הסוכן לאזור שהמודל “לא ראה”, ולגרום לשגיאות מצטברות עד להתדרגות מוחלטת. הבעיה נובעת מכך שהמודל אומן על סמן *data* שנאסף מpolloסה ספציפית (π_0).

גישה שנייה: *DAgger*

כפי שהסביר לעיל, מבצע איסוף של נתונים שנאפו לאורך הזמן, כאשר הpolloסה מתעדכנת בכל פעם שנאסף מספק *data*. שלבי השיטה:

1. שימוש במודל ראשוני ($a_t|s_t$) π_0 לאיסוף $data$.
2. שימוש ב- $data$ זה לאימון מודל $s_i^2 - \sum_i \|f(s_{i-1}, a_{i-1}) - s_i\|^2$.
3. מוצאת המודל, לכניסה למודל, ובחירה פעולה a .
4. ביצוע פעולה a ואיסוף דגימות חדשות על בסיסה $\{(s_j, a_j, s'_j)\}$, דגימות אלה מותוספות ל- $dataset$.
5. חוזרת לשלב 2

בעה בשיטה זו היא שהאימון מחדש מתבצע רק לאחר שנאספו מסלולים (trajectories) חדשים לאימון.

גישה 3 - MPC

לשיטה הבאה קוראים *model predictive control (MPC)*, בשיטה זו מעדכנים בתדריות את המודל:

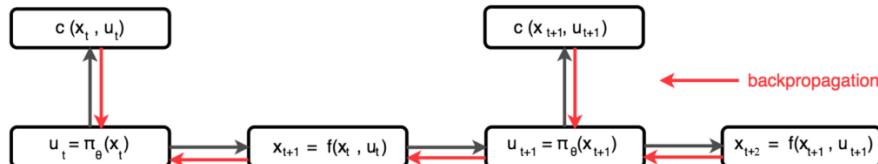
1. שימוש במודל ראשוני ($a_t|s_t$) π_0 לאיסוף $data$.
2. שימוש ב- $data$ זה לאימון מודל $s_i^2 - \sum_i \|f(s_{i-1}, a_{i-1}) - s_i\|^2$.
3. מוצאת המודל, לכניסה למודל, ובחירה פעולה a .
4. ביצוע פעולה a ואיסוף דגימות חדשות על בסיסה $\{(s_j, a_j, s'_j)\}$, דגימות אלה מותוספות ל- $dataset$.
5. **אחת ל-N צעדים – חוזרת ל-2 (ולמידה מחדש של מודל), אחרות חוזרת לשלב 3.**

גישה 4 – למידת הפוליסיה

בגישה זו, לומדים במהלך האימון גם את הדינמיקה וגם את הפוליסיה (למשל באמצעות רשת). הרעיון הוא שאמם הפוליסיה והמודל של הסביבה שניים גזירים, ניתן לבצע *backpropagation* גם לפוליסיה. שלבי האלגוריתם:

1. איסוף $data$ על ידי פוליסיה ההתחלתית ($\pi_0(a_t|s_t)$).
2. שימוש ב- $data$ זה לאימון מודל $s_i^2 - \sum_i \|f(s_{i-1}, a_{i-1}) - s_i\|^2$.
3. **ביצוע reward f backpropagation** ועדכון הפוליסיה על פי ה- $reward$ - $dataset$ שהתקבל.
4. הרצת ($a_t|s_t$) π לאיסוף $data$ והוספתו ל- $dataset$.

בשיטה זו "פותחים" את צעדי הסוכן ומעבירים גרדיאנטים לכל אורך ה-*episode* (בדומה ל-RNN). להלן תרשימים של השיטה:



PILCO היא שיטה המבוססת על גישה זו. בשיטה זו משתמשים ב-*Gaussian Processes* על מנת למ笪 את הסביבה. *GP* היא שיטה ללמידה שאינה מבוססת רשותות, אשר ייעלה בלמידה כאשר אין כמות גדולה של *data*. החישרונו בשיטה זו הוא שקשה לה להתמודד עם *data* מממד גבוה (כמו תמונות), וקשה לה ללמידה דינמיקה שאינה חלקה. בכל שלב באלגוריתם, נלמדת מחדש הדינמיקה באמצעות *GP*, ואז מושך *episode* אחד שבמהלכו מבצעים אופטימיזציה לסוכן. להלן תיאור השיטה:

Algorithm 1 PILCO

```

1: init: Sample controller parameters  $\theta \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ .  

   Apply random control signals and record data.  

2: repeat  

3:   Learn probabilistic (GP) dynamics model, see Sec. 2.1, using all data.  

4:   Model-based policy search, see Sec. 2.2–2.3.  

5:   repeat  

6:     Approximate inference for policy evaluation,  

      see Sec. 2.2: get  $J^\pi(\theta)$ , Eqs. (10)–(12), (24).  

7:     Gradient-based policy improvement, see  

      Sec. 2.3: get  $dJ^\pi(\theta)/d\theta$ , Eqs. (26)–(30).  

8:     Update parameters  $\theta$  (e.g., CG or L-BFGS).  

9:   until convergence; return  $\theta^*$   

10:  Set  $\pi^* \leftarrow \pi(\theta^*)$ .  

11:  Apply  $\pi^*$  to system (single trial/episode) and  

      record data.  

12: until task learned

```

: לsicoms :

גישה	סיכון	יתרונות	חרוגות
1	איסוף, <i>data</i> , למידת דינמיקה, <i>planning</i>	פשטה, אין איטרציות	בעיית השינוי בתהפלגיות
2	איסוף, <i>data</i> ו- בצורה איטרטיבית <i>planning</i>	פשטה, פורתת את בעיית השינוי בתהפלגיות	ה- <i>open-loop planning</i> הוא <i>open-loop</i> , כלומר לא מתќבל פידבק על הpolloisa, ביצועים ירודים במיוחד בסביבות סטוכסטיות
3	<i>MPC</i> בכל צעדים עדכן המודל כל N צעדים	רוביSTITית לטיעוית קטנות	יקר חישובית
4	<i>Backpropagation</i> <i>inference</i>	מירה בזמון ה-	יכולת להיווצר בעיה של חוסר יציבות, במיוחד עם מסלולים ארוכים (<i>vanishing/exploding gradients</i>)

LQR

עד כה דיברנו על מודלים **גLOBליים**, מודל אחד שמיידל את כל הסביבה. גישה אחרת ללמידה של מודל סביבה היא באמצעות מספר מודלים מקומיים, אשר כל אחד מمدל חלק אחר בסביבה. בגישה זו נחפש מודל פשוט יותר אשר מmdl טוב את הסביבה בהנו נמצאים ברגע זה.

במודל לינארי, ניתן למدل את הדינמיקה על ידי :

$$x_{t+1} = Ax_t + Bu_t$$

בנוסף, נניח שקיים *cost* ריבועי מהצורה :

$$g(x_t, u_t) = x_t^T Q x_t + u_t^T R u_t$$

הסיבה שאנו משתמשים ב-*cost* ריבועי ולא לינארי היא שאנו מעוניינים בגודל חיובי. Q, R מטריצות מוגדרות חיובית, והן מושקלות את *cost* של x_t ושל u_t . למשל, *state* x_t מסויים יכול להיות עם ערך Q גבוה (רכיב שנופל מזמן), ופעולה מסוימת יכולה להיות עם ערך R גבוה (למשל בזבוז משאבות משחק אסטרטגייה). המטרה ב-*LQR* היא למצוא מטריצה K שמקיימת :

$$u = -Kx$$

מטריצה זו נקראת *gain matrix*, *gain matrix*, וניתן להוכחה אנליטית שהיא מביאה למינימום את $g(x_t, u_t)$.

במקרה שלנו, ניתן לבטא את ה-*cost function* בצורה הבאה :

$$\min_{u_1, \dots, u_T} c(x_1, u_1) + c(f(x_1, u_1), u_2) + c(f(f(x_1, u_1), u_2), u_3) + \dots + c(f(f(\dots, u_{T-1}), u_T)$$

כאשר כל מצב מבוטא באמצעות המצב הקודם בצורה רקורסיבית. בביוטי זה ישנו רק המצב ההתחלתי x_1 ורצף הפעולות $\{u_1, \dots, u_T\}$. את מטריצת המעבר ניתן לבטא באמצעות הביטוי הבא :

$$f(x_t, u_t) = F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t$$

כאשר מטריצת המעבר F_t היא מטריצת המעבר (שילוב של מטריצות A, B שהזכרנו לעיל), ו- f_t הוא קבוע. את פונקציית המחיר ניתן לבטא בצורה הבאה (בשונה מהצורה הקודמת) :

$$c(x_t, u_t) = \frac{1}{2} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T C_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T c_t$$

כאשר C_t היא מטריצה המשקלת את הביטויים הריבועיים $c_t, x_t^2, u_t^2, x_t u_t$ והוא וקטור המשקל את הביטויים הליינריים u_t, x_t . נשים לב ש- C_t, c_t תלויים ב- t , מכיוון שניתן להציב בהם משתנים שתלויים בזמן. בעת, ניתן להתחיל ללמידה את המקבילה *cost to go - Q-function*. כדי בלבבל, נסמן גם אותה ב- Q . נתחילה מהסוף (T) ונתקדם אחורה, כמו ב-*value iteration*. מתחילה בצעד האחרון :

$$Q(x_T, u_T) = \text{const} + \frac{1}{2} \begin{bmatrix} x_T \\ u_T \end{bmatrix}^T C_T \begin{bmatrix} x_T \\ u_T \end{bmatrix} + \begin{bmatrix} x_T \\ u_T \end{bmatrix}^T c_T$$

ה-*const* הוא קבוע (ניתן להציב בו אפס), שנוון לנו אפשרות להוסיף להוסיף *cost* קבוע למסלול. נשים לב כי את

הוקטור $\begin{bmatrix} x_T \\ u_T \end{bmatrix}$ בנינו כך שהאיברים הראשונים שייכים במצב, והאחרונים לפעולה. לכן ניתן לפרק את C_T

בצורה נוחה לרכיבים שככפילים איברים של x_T , של u_T או של איברים מעורבים. ניתן אם כן לכתוב אותה בצורה הבאה :

$$C_T = \begin{bmatrix} C_{x_T x_T} & C_{x_T u_T} \\ C_{u_T x_T} & C_{u_T u_T} \end{bmatrix}$$

כאשר ה-*subscripts* מסמנים את אילו ערכים כופלת כל תת מטריצה. בדומה לכך, ניתן לפרק את הוקטור c_T :

$$c_T = \begin{bmatrix} c_{x_T} \\ c_{u_T} \end{bmatrix}$$

מכיוון שאנו מעוניינים באופטימיזציה על הפעולות, נגזר את Q לפי u_T . ניתן אלגברית לייצג את הנגזרת בצורה הבאה :

$$\nabla_{u_T} Q(x_T, u_T) = C_{u_T x_T} x_T + C_{u_T u_T} u_T + c_{u_T}^T$$

ומכיוון שהבעיה קוונטקטית (כי היא ריבועית) ניתן למצוא מינימום גלובלי על ידי השוואת הנגזרת לאפס :

$$C_{u_T x_T} x_T + C_{u_T u_T} u_T + c_{u_T}^T = 0 \Rightarrow u_T = -C_{u_T u_T}^{-1} (C_{u_T x_T} x_T + c_{u_T}^T) = -C_{u_T u_T}^{-1} C_{u_T x_T} x_T - C_{u_T u_T}^{-1} c_{u_T}^T$$

$$= K_T x_T + k_T$$

כאשר $K_T = -C_{u_T u_T}^{-1} C_{u_T x_T}$ ו- $k_T = -C_{u_T u_T}^{-1} c_{u_T}^T$. ככלומר מצאנו את u_T שיביא למינימום את ה-*cost function* תחת הנחות הבעיה (פונקציית מעבר מצב לינארית ופונקציית מחיר ריבועית). מכיוון שאנו בשלב T , $Q(x_T, u_T) = V(x_T)$. לכן :

$$\begin{aligned}
V(x_T) &= Q(x_T, u_T) = \text{const} + \frac{1}{2} \begin{bmatrix} x_T \\ u_T \end{bmatrix}^T C_T \begin{bmatrix} x_T \\ u_T \end{bmatrix} + \begin{bmatrix} x_T \\ u_T \end{bmatrix}^T c_T \\
&= \text{const} + \frac{1}{2} \begin{bmatrix} x_T \\ K_T x_T + k_T \end{bmatrix}^T C_T \begin{bmatrix} x_T \\ K_T x_T + k_T \end{bmatrix} + \begin{bmatrix} x_T \\ K_T x_T + k_T \end{bmatrix}^T c_T \\
&= \frac{1}{2} x_T^T C_{x_T, x_T} x_T + \frac{1}{2} x_T^T C_{x_T, u_T} K_T x_T + \frac{1}{2} x_T^T K_T^T C_{u_T, x_T} x_T + \frac{1}{2} x_T^T K_T^T C_{u_T, u_T} K_T x_T \\
&\quad + x_T^T K_T^T C_{u_T, u_T} k_T + \frac{1}{2} x_T^T C_{x_T, u_T} k_T + x_T^T c_{x_T} + x_T^T K_T^T c_{u_T} + \text{const}
\end{aligned}$$

ניתן להגדיר :

$$\begin{aligned}
V_T &= C_{x_T, x_T} + C_{x_T, u_T} K_T + K_T^T C_{u_T, x_T} + K_T^T C_{u_T, u_T} K_T \\
v_T &= c_{x_T} + C_{x_T, u_T} k_T + K_T^T C_{u_T} + K_T^T C_{u_T, u_T} k_T
\end{aligned}$$

ולקבל את הביטוי המפוסט :

$$V(x_T) = \text{const} + \frac{1}{2} x_T^T V_T x_T + x_T^T v_T$$

כעת ניתן להתקדם צעד אחד אחורה. ה-*T-1* של המצב הוא :

$$\begin{aligned}
Q(x_{T-1}, u_{T-1}) &= \text{const} + \frac{1}{2} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T C_{T-1} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} + \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T c_{T-1} + Q(x_T, u_T) \\
&= \text{const} + \frac{1}{2} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T C_{T-1} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} + \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T c_{T-1} + V(x_T)
\end{aligned}$$

ניתן לבטא את x_T באמצעות הפונקציה הליינרית של הדינמיקה של המודל :

$$x_T = f(x_{T-1}, u_{T-1}) = F_{t-1} \begin{bmatrix} x_{t-1} \\ u_{t-1} \end{bmatrix} + f_{t-1}$$

$$\begin{aligned}
V(x_T) &= \text{const} + \frac{1}{2} x_T^T V_T x_T + x_T^T v_T \\
&= \text{const} + \frac{1}{2} (F_{t-1} \begin{bmatrix} x_{t-1} \\ u_{t-1} \end{bmatrix} + f_{t-1})^T V_T (F_{t-1} \begin{bmatrix} x_{t-1} \\ u_{t-1} \end{bmatrix} + f_{t-1}) \\
&\quad + (F_{t-1} \begin{bmatrix} x_{t-1} \\ u_{t-1} \end{bmatrix} + f_{t-1})^T v_T \\
&= \text{const} + \frac{1}{2} \begin{bmatrix} x_{t-1} \\ u_{t-1} \end{bmatrix}^T F_{t-1}^T V_T F_{t-1} \begin{bmatrix} x_{t-1} \\ u_{t-1} \end{bmatrix} + \begin{bmatrix} x_{t-1} \\ u_{t-1} \end{bmatrix}^T F_{t-1}^T V_T f_{t-1} + \begin{bmatrix} x_{t-1} \\ u_{t-1} \end{bmatrix}^T F_{t-1}^T v_T \\
&\quad + \left(\frac{1}{2} f_{t-1}^T V_T f_{t-1} + f_{t-1}^T v_T \right) \\
&= \widehat{\text{const}} + \frac{1}{2} \begin{bmatrix} x_{t-1} \\ u_{t-1} \end{bmatrix}^T F_{t-1}^T V_T F_{t-1} \begin{bmatrix} x_{t-1} \\ u_{t-1} \end{bmatrix} + \begin{bmatrix} x_{t-1} \\ u_{t-1} \end{bmatrix}^T F_{t-1}^T V_T f_{t-1} + \begin{bmatrix} x_{t-1} \\ u_{t-1} \end{bmatrix}^T F_{t-1}^T v_T
\end{aligned}$$

כאשר במהלך האחרון כל הקבועים (כל הביטויים שאינם תלויים ב- u , x עברו תחת המשנה $\widehat{\text{const}}$. כעת

ניתן להציב את $V(x_T)$ במשוואת $Q(x_{T-1}, u_{T-1})$. ניתן לראות שגם $V(x_T)$ יש חלק ליינארי וחלק

ריבועי, ולכן ניתן לקובץ אותםividually כאשר מציבים אותם ב- Q :

$$\begin{aligned}
Q(x_{T-1}, u_{T-1}) &= \widehat{\text{const}}' + \frac{1}{2} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T (C_{T-1} + F_{t-1}^T V_T F_{t-1}) \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} \\
&\quad + \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T (c_{T-1} + F_{t-1}^T V_T f_{t-1} + F_{t-1}^T v_T) \\
&= \widehat{\text{const}}' + \frac{1}{2} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T Q_{T-1} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} + \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T q_{T-1}
\end{aligned}$$

כאשר :

$$\begin{aligned} Q_{T-1} &= C_{T-1} + F_{t-1}^T V_T F_{t-1} \\ q_{T-1} &= c_{T-1} + F_{t-1}^T V_T f_{t-1} + F_{t-1}^T v_T \end{aligned}$$

ושוב ניתן לגוזר על מנת למצוא פוליה אופטימלית עבור $1 - T$ כפי שעשינו בעבר :
 $\nabla_{u_{T-1}} Q(x_{T-1}, u_{T-1}) = Q_{u_{T-1}, x_{T-1}} x_{T-1} + Q_{u_{T-1}, u_{T-1}} + q_{u_{T-1}}^T = 0$
ולמצוא את u_{T-1} האופטימי. בדומה זו ניתן למצוא את הפעולות האופטימליות עד u_1 .

שיטת LQR מתאימה בעיקר למקרים פשוטים, בהם נרצה להישאר בקרבת מצב נתון (למשל *cart-pole*). ניתן להשתמש בה גם כאשר הדינמיקה לא דטרמיניסטיבית, אלא עם פילוג גאוסי עם תוחלת שנייתן למצל לינארית :

$$f(x_t, u_t) = F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t$$

ושונות שהיא פונקציה של *cost*. במקרה זה, עדיין ניתן להשתמש בפתרון $K_t x_t + k_t = K_t x_t + k_t(u_t)$. עם זאת, במקרים בהם הדינמיקה אינה לינארית, לא ניתן להשתמש בשיטה זו, ונדרש להתקדם לשיטות מתקדמות יותר.

iLQR

iLQR היא הרחבה הלא-LINEAR של *LQR* (*iterative LQR*). בשיטה זו, מקרים דינמיקה לא לינארית על ידי לינאריזציה באמצעות טור טילוור. הדינמיקה של *LQR* היא לינארית, ולכן הקירוב לדינמיקה ב-*LQR* הוא מסדר ראשון :

$$f(x_t, u_t) \approx f(\hat{x}_t, \hat{u}_t) + \nabla_{x_t, u_t} f(\hat{x}_t, \hat{u}_t) \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix}$$

ואילו ה-*cost function* ב-*LQR* היא ריבועית, ולכן נקרב אותה על ידי סדר שני :

$$c(x_t, u_t) \approx c(\hat{x}_t, \hat{u}_t) + \nabla_{x_t, u_t} c(\hat{x}_t, \hat{u}_t) \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix}^T \nabla_{x_t, u_t}^2 c(\hat{x}_t, \hat{u}_t) \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix}$$

לצורך נוחות, נגדיר קירובים ל- f, c :

$$\bar{f}(\delta x_t, \delta u_t) = F_t \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix}$$

$$\bar{c}(\delta x_t, \delta u_t) = \frac{1}{2} \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix}^T C_t \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix} + \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix}^T c_t$$

כאשר $c_t = \nabla_{x_t, u_t} c(\hat{x}_t, \hat{u}_t)$, $C_t = \nabla_{x_t, u_t}^2 c(\hat{x}_t, \hat{u}_t)$, $F_t = \nabla_{x_t, u_t} f(\hat{x}_t, \hat{u}_t)$

האלגוריתם עובד بصورة הבא :

בשלב ראשון, בהינתן מצב התחלה \hat{x}_0 מבצעים סדרה של פעולות $\hat{u}_0 \dots \hat{u}_T$ ומקבלים את המצביעים $\hat{x}_0 \dots \hat{x}_T$.

בשלב הבא מבצעים את הפעולות הבאות بصورة איטרטיבית עד להתכנסות :

1. חישוב ($F_t = \nabla_{x_t, u_t} f(\hat{x}_t, \hat{u}_t)$) מהdinמיקה הלא-LINEAR על כל אחד מהמצביעים והפעולות

2. חישוב ($c_t = \nabla_{x_t, u_t} c(\hat{x}_t, \hat{u}_t)$)

3. חישוב ($C_t = \nabla_{x_t, u_t}^2 c(\hat{x}_t, \hat{u}_t)$)

4. הרצת *LQR* על המצביעים \hat{x}_t והפעולות \hat{u}_t ($\delta x_t = x_t - \hat{x}_t$ ו- $\delta u_t = u_t - \hat{u}_t$)

5. הרצה של המסלול האופטימי שנמצא עם הדינמיקה הלא-LINEAR $u_t = \hat{u}_t + K_t(x_t - \hat{x}_t) + k_t$

6. עדכון המצביעים \hat{x}_t והפעולות \hat{u}_t

$iLQR$ רחוק מלהיות מודל מושלם. לאחר האופטימיזציה מבצעים "שגר וصاحب", והוא אינו מתעדכן בצורה אוטומטית במהלך המסלול. היא אינה מתמודדת טוב עם רעש, עם שגיאה במצב ההתחלתי ועם שגיאה במידול הסביבה. עם זאת זו שיטה בסיסית שכדי להכיר, והיא משמשת כאבן בניין באlgorigthms מורכבים יותר.

Model Predictive Controls (MPC)

הweeney המרכז ב-MPC הוא עדכון של $iLQR$ לאורך המסלול. מבצעים חישוב $iLQR$ בכל נקודת זמן t עד T , במקומות לבצע את האופטימיזציה פעם אחת בתחלת המסלול. המטרה היא להתגבר על אי-יעדים לא צפויים במהלך המסלול, ולבצע תיקונים ככל שמתקרבים לסיומו. בכל שלב t מחשבים את ה- $iLQR$ ומתקבלים $u_T, \dots, u_{t+1}, u_t, u$, אך משתמשים רק ב- u_t .

לפעמים, במקרים לחשב את ה- $iLQR$ עד לשוף המסלול, משתמשים בשיטה שנראית "*receding horizon control*". בשיטה זו קובעים חלון T קבוע, ובמבצעים את האופטימיזציה עד זמן $t + T$. שיטה זו פותרת את הבעיה שיתכן ובעתיד הרחוק ארירע בלתי צפוי ישנה את המסלול ובכך יותר את החישובים שהתבצעו. בנוסף, גם כאשר מתקרבים לשוף המסלול, ו- $t + T$ נהייה אחורי תום המסלול, עדין האלגוריתם "אופטימי" מכיוון שהוא חשוב שיש לו עוד זמן. עובדה זו גורמת לו לא "להתיאש" כאשר הוא רואה בהערכתו של ה- $iLQR$ שהוא מידי.

לסיפורים נושא *local models* נעיר שנית להשתמש בשיטות לוקליות שנלמדו על מצבים שונים על מנת ללמוד מודל גלובלי יחיד (רשת ניורונים). למשל, ניתן להשתמש במודל LQR על מנת למדד את ההתנהגות בסביבה הקדומה, ואז להשתמש בפרדייקציות על מנת לאמן רשת *policy gradients*. בנוסף, ניתן להשתמש במספר רשותות קטנות למידול הסביבה הקדומה (מספר סביבות), ובעזרתן לעדכן רשת *policy* גדולה.

Gradient Free Methods

כפי שהזכרנו לעיל, *Gradient Free Methods* הן שיטות שלא משתמשות באופטימיזציה מבוססת גרדיאנטים אלא בשיטות סטטיסטיות ואחרות ל-*RL*. שיטות אלה בדרך כלל פחות יעילות מאשרות אחרות ודורשות כמות גדולה מאוד של *data*.

אלגוריתמים אבולוציוניים הם שיטה שאינה מבוססת גרדיאנטים. בשיטה זו, יוצרים **אוכלוסייה** של אלגוריתמים, אשר מקבלים ציון על פי *fitness function*. האוכלוסייה מעודכנת בכל שלב באמצעות הפעולות הבאות:

- – תהליכי אקראי אשר בוחר את האלגוריתמים הטוביים על פי ה-*fitness function* *Selection*
- – ייצור "ילדים" לאלגוריתמים אלו, באמצעות שינויים קטנים באלגוריתמים המקוריים *Mutation*
- – שילוב של מספר אלגוריתמים לאלגוריתם אחד *Recombination*

גם באלגוריתמים אלו ניתן להתייחס ל-*exploration-exploitation tradeoff*, כאשר ייצור האלגוריתמים החדשניים היא האקספלורציה, ובבחירה האלגוריתמים הטוביים היא ה-*exploitation*.

תהליכי האופטימיזציה הוא *black-box* במובן שניתן לעבוד עם פונקציית *fitness* שרק מחזירה הערכה של טיב האלגוריתם. אלגוריתמים אלו מתאימים גם לביעות שאין קונבנציות ושאין "מתחנוגותיפה" (שינויים חדים בפונקציה, רעש, נקודות מינימום מקומי). חלק מהאלגוריתם משתמש בשיטות דומות לשיטת ניוטון המבוססת על גזרת שנייה (אך בלי *backpropagation* ובלי *gradient descent*).

אלגוריתם החיפוש מתבצע בצורה הבאה. בהתחלה מתבצע אתחול לפרמטרים של מודל הבחירה θ (יכול להיות רשת), ולגדיל האוכלוסייה \mathcal{L} . בכל איטרציה, נציגות \mathcal{L} דגימות בעוזרת התפלגות של המודל $P(\theta|\alpha)$ ומתקבלים הפתרונות (האלגוריתמים) $\alpha_1, \dots, \alpha_n$. מחושב הערך של ה-*objective function* $\mathcal{L}(\alpha)$ על הדגימות

לבסוף מתעדכנים הפרמטרים θ על ידי פונקציה ייועודית $f(x_1), \dots, f(x_\lambda)$ ($F_\theta(\theta, x_1, \dots, x_\lambda, f(x_1), \dots, f(x_\lambda))$). עדכון זה מתרחש על מנת שהדגימה תהיה יותר.

עוד שיטה היא "Covariance Matrix Adaptation Evolution Strategy (CMA-ES)" או "Gradient Free". השיטה מבוססת על מטרת השיטה היא לאפשר לדוגם ממוחב הפתורונות פתרונות עם ביצועים טובים. השיטה מבוססת על *Maximum Likelihood* – חישוב הסתברות שבסביר את הדגימות בצורה "הטובה ביותר" (כלומר שבاهינתן פונקציית ההסתברות הזאת, ההסתברות שהדגימות שהתקבלו להתקבל היא הגבוהה ביותר מבין פונקציות התפלגות). הדגימות במרקחה זה הן פתרונות. בנוסף, השיטה מتبוססת על *covariance matrix adaptation*, שינוי הדרוגתי של מטריצת ה-*covariance*, כך שתתמקם ההסתברות לקבל את הפתרונות הטובים שהתקבלו בעבר. האופטימיזציה מתרחשת על ידי אופטימיזציה של פונקציית *Natural Gradient Descent*, שיטה לאופטימיזציה של פונקציית התפלגות. בנוסף להתחשבות בגרדיינטים *NGD* מוצעת ערך גם על פי *האיומטריה* של התפלגות כך שהצורה של התפלגות תהיה דומה לזה הנכיפת בדגימות. נעשה שימוש במטריצת האינפורמציה של פישר (*Fisher information matrix*) אשר מכממתת תכונות גיאומטריות של מוחב הפרמטרים. לצורך כך ניתן להשתמש בדגםת *MC*, על מנת להעריך את המטריצה. להלן תיאור האלגוריתם:

```

set λ // number of samples per iteration, at least two, generally > 4
initialize m, σ, C = I, pσ = 0, pc = 0 // initialize state variables
while not terminate // iterate
    for i in {1...λ} // sample λ new solutions and evaluate them
        xi = sample_multivariate_normal(mean=m, covariance_matrix=σ²C)
        fi = fitness(xi)
        x1...λ ← xs(1)...xs(λ) with s(i) = argsort(f1...λ, i) // sort solutions
        m' = m // we need later m - m' and xi - m'
        m ← update_m(x1, ..., xλ) // move mean to better solutions
        pσ ← update_ps(pσ, σ⁻¹C⁻¹/²(m - m')) // update isotropic evolution path
        pc ← update_pc(pc, σ⁻¹(m - m'), ||pσ||) // update anisotropic evolution path
        C ← update_C(C, pc, (x1 - m')/σ, ..., (xλ - m')/σ) // update covariance matrix
        σ ← update_sigma(σ, ||pσ||) // update step-size using isotropic path length
    return m or x1

```