**למידת חיזוק עמוקה**

מרצה: ד"ר גלעד כץ

סיכם: צבי לדרר

**סמסטר חורף תשפ"ג 2023**

הסיכום על בסיס מצגות הקורס

תוכן עניינים

[יסודות ה-RL 5](#_Toc126324212)

[Markov Decision Processes (MDP) 5](#_Toc126324213)

[פונקציות הערכה 5](#_Toc126324214)

[משוואת Bellman 6](#_Toc126324215)

[Policy Iteration 6](#_Toc126324216)

[Value Iteration 7](#_Toc126324217)

[Model based ו-Model free 7](#_Toc126324218)

[Monte Carlo (MC) 7](#_Toc126324219)

[On Policy ו-Off Policy 8](#_Toc126324220)

[Temporal Difference Learning 9](#_Toc126324221)

[SARSA 10](#_Toc126324222)

[Off-Policy TD Control: Q-Learning 10](#_Toc126324223)

[Deep Q-learning 11](#_Toc126324224)

[Fixed Q-targets 11](#_Toc126324225)

[Experience replay 11](#_Toc126324226)

[Double DQN 12](#_Toc126324227)

[Dueling DQN 12](#_Toc126324228)

[Policy Gradients 14](#_Toc126324229)

[REINFORCE 14](#_Toc126324230)

[REINFORCE with Baseline 15](#_Toc126324231)

[Actor Critic 15](#_Toc126324232)

[Asynchronous Advantage Actor-Critic (A3C) 16](#_Toc126324233)

[Imitation Learning 17](#_Toc126324234)

[מונחים 17](#_Toc126324235)

[Imitation Learning 18](#_Toc126324236)

[Apprenticeship Learning 19](#_Toc126324237)

[Behavioral Cloning 19](#_Toc126324238)

[Forward Training 19](#_Toc126324239)

[DAgger 20](#_Toc126324240)

[Multi-arm bandits 22](#_Toc126324241)

[Upper Confidence Bound (UCB) 22](#_Toc126324242)

[Gradient Bandits Algorithm 23](#_Toc126324243)

[LinUCB 23](#_Toc126324244)

[Thompson Sampling 24](#_Toc126324245)

[Monte Carlo Tree Search 26](#_Toc126324246)

[שלבי האלגוריתם 26](#_Toc126324247)

[Selection 26](#_Toc126324248)

[Expansion 26](#_Toc126324249)

[Simulation 26](#_Toc126324250)

[Backpropagation 26](#_Toc126324251)

[תכונות MCTS 26](#_Toc126324252)

[שיפורים לMCTS 27](#_Toc126324253)

[AlphaGO 27](#_Toc126324254)

[מרחב הכניסה (input) 28](#_Toc126324255)

[Supervised learning policy network 28](#_Toc126324256)

[RL policy network 28](#_Toc126324257)

[RL value network 29](#_Toc126324258)

[Rollout Policy 29](#_Toc126324259)

[תהליך האימון 29](#_Toc126324260)

[AlphaGoZero 29](#_Toc126324261)

[AlphaGO בתחומים אחרים 31](#_Toc126324262)

[Meta-learning 32](#_Toc126324263)

[Meta-Learning with Memory-Augmented Networks 32](#_Toc126324264)

[גישה לזיכרון 33](#_Toc126324265)

[כתיבה לזיכרון 33](#_Toc126324266)

[פרדיקציה 34](#_Toc126324267)

[ניסויים 34](#_Toc126324268)

[Simple Neural Attentive Meta-Learner (SNAIL) 35](#_Toc126324269)

[תיאור השיטה 35](#_Toc126324270)

[ניסויים 36](#_Toc126324271)

[Model Agnostic Meta-Learning (MAML) 36](#_Toc126324272)

[Adaption in Real-Time 37](#_Toc126324273)

[Transfer Learning 39](#_Toc126324274)

[Fine-tuning 39](#_Toc126324275)

[Progressive Networks 40](#_Toc126324276)

[Self-Supervision for RL 42](#_Toc126324277)

[Model-Based RL 43](#_Toc126324278)

[Actor Mimic 43](#_Toc126324279)

[Distillation for Multi-Task Transfer 43](#_Toc126324280)

[Modular Neural Network 43](#_Toc126324281)

[מרחבי מצבים ופעולות גדולים 45](#_Toc126324282)

[מרחב פעולות רציף 45](#_Toc126324283)

[Action Elimination with Deep Reinforcement Learning 46](#_Toc126324284)

[Hierarchical DRL for Sparse Reward Environments 47](#_Toc126324285)

[בעיות מהעולם האמיתי 49](#_Toc126324286)

[Branching Dueling Q-Networks (BQD) 51](#_Toc126324287)

[Jointly-Learned State-Action Embedding 52](#_Toc126324288)

[Learning in Latent Space 53](#_Toc126324289)

[Model-Free Approaches for Learning the Latent Space 53](#_Toc126324290)

[Model-Based Approaches for Learning the Latent Space 53](#_Toc126324291)

[Embed to Control (E2C) 56](#_Toc126324292)

[Action-Conditional Video Prediction 56](#_Toc126324293)

[Inverse Reinforcement Learning 58](#_Toc126324294)

[סיכום model based מול model free 60](#_Toc126324295)

[Exploration with Exemplar Model 60](#_Toc126324296)

[Curiosity-driven Exploration by Self-supervised Prediction 62](#_Toc126324297)

[Transformers in RL 64](#_Toc126324298)

[Longformer ו-Big-Bird 64](#_Toc126324299)

[The Decision Transformer 65](#_Toc126324300)

[Trajectory Transformer 65](#_Toc126324301)

[Model-Based Methods 67](#_Toc126324302)

[גישות ללמידת מודל סביבה 67](#_Toc126324303)

[LQR 69](#_Toc126324304)

[iLQR 72](#_Toc126324305)

[Model Predictive Controls (MPC) 73](#_Toc126324306)

[Gradient Free Methods 73](#_Toc126324307)

# **יסודות ה-RL**

ב-Reinforcement learning אנו מתמודדים עם בעיות בהם נדרש ללמד **סוכן** לבצע **פעולות** ב**סביבה מסוימת** כך שימקסמו **reward**. הבעיות שאותן נפגוש מאופיינות ברצפים (sequences) של פעולות, תצפיות ו-rewards, אשר לרוב אינם מקיימים את הנחת חוסר התלות הסטטיסטית (iid) אשר מאפיינת בעיות supervised learning, ולרוב מאופיינות ב-reward דליל (sparse) אשר מקשה על הלמידה.

אתגרים ב-RL:

* אין dataset סגור, נדרשת גישה לסביבה (environment) על מנת לאסוף data.
* ה-data שאנחנו אוספים מהסביבה תלוי בפוליסה שבה השתמשנו, וכאשר משנים פוליסה ההתפלגות של ה-data גם הוא ישתנה.
* נדרש ביצוע במקביל גם של למידה (חיזוי) וגם של תכנון (planning).

## **Markov Decision Processes (MDP)**

MDP מאופיין במצבים (states) רציפים או בדידים, המסומנים ב-, פעולות (actions) המסומנות ב-, ו-rewards המסומנים ב-. כאשר מבצעים פעולה, עוברים ממצב למצב אחר. מעבר זה מאופיין בפונקציית הסתברות התלויה בפעולה . הנחת המרקוביות היא שהמעבר תלוי אך ורק במאפייני המצב הנוכחי (מצב ופעולה) ולא בעבר הרחוק יותר:

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

כאמור, מה שמייחד בעיות כאלה הוא שנדרשת גישה לסביבה, ה-data שמתקבל תלוי בפעולות, והלמידה והתכנון צריכות 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):

או סכום ממושקל:  
(בד"כ בתהליכים אינסופיים), כאשר בדרך כלל קרוב ל-1 (למשל 0.99).

## **פונקציות הערכה**

אנו מגדירים מספר פונקציות בסיסיות אשר נעשה בהם שימוש לכל אורך הקורס. ה**פוליסה** היא הפונקציה שבאמצעותה בוחרים פעולה:

ה-**value function** היא הפונקציה שמעריכה את תוחלת ה-return שנקבל תחת פוליסה מסוימת:

ה-**Q-function** היא הפונקציה שמעריכה את ה-return שנקבל אם נבחר פעולה בצעד הבא ולאחר מכן נמשיך לפעול לפי פוליסה :

הפונקציות האופטימליות הן הפוליסה שתיתן תוחלת מקסימלית של return על פני כל הפוליסות האפשריות, ופונקציות ה-value וה-q המתאימות לה.

## **משוואת Bellman**

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

את המשוואה ל- ניתן לפתח:

ואת (האופטימלי) ניתן לרשום בצורה הבאה:

בדומה, ניתן לפתח את :

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

## **Policy Iteration**

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

בשלב הevaluation משתמשים במשוואת Bellman כדי לעדכן את ה-value function:  
העדכון נעשה בצורה איטרטיבית עד להתכנסות עבור הפוליסה הנוכחית. בשלב ה-improvement, הדרך הפשוטה ביותר היא לשנות פוליסה עבור מצב אחד בלבד ולהעריך מחדש את הערך שלו. אם הערך שלו גדל, אנו יכולים להיות בטוחים שהפוליסה טובה יותר. אפשר גם לבצע זאת למספר מצבים במקביל:

במקרה של MDP סופי, מספר המצבים הוא סופי, ולכן מובטח לנו שנגיע לפוליסה האופטימלית.

## **Value Iteration**

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

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

## **Model based ו-Model free**

השיטות שראינו עד כה מניחות כי אנו מכירים את כל המצבים, הפעולות, ה-rewards וכן את הדינמיקה של העולם, כלומר את ההסתברויות למעבר ממצב למצב בהינתן פעולה. אלגוריתמים אשר מסתמכים על כל הדברים הללו נקראים **model based**. לעומת זאת, כאשר חלק מהמאפיינים חסרים לנו (למשל, לא מכירים את הסתברויות המעבר ממצב למצב), אנו נכנסים לעולם ה-**model free**. במקרים כאלה בד"כ נמדל ישירות את ללא מידול הסביבה.

## **Monte Carlo (MC)**

*בשיטות שראינו עד כה הנחנו שאנו מכירים את פונקציות המעברים וכן את . במקרים בהם נתונים אלה חסרים ניתן להשתמש בשיטת Monte Carlo, דגימה מתוך הסביבה ושערוך של הנתונים החסרים על בסיסה. ניתן להמיר לשיטה זו את כל האלגוריתמים שתיארנו. למשל, על מנת להעריך את ה-value function ניתן לדגום מספר מסלולים על פי הפוליסה הקיימת, ולמצע על פני ה-rewards שקיבלנו כדי לעדכן אותה. אחת הבעיות ב-MC היא שניתן לעשות זאת רק על מסלולים סופיים (episodic).*

*במהלך episode, יכול להיווצר מצב שנגיע למצב (state) מסוים יותר מפעם אחת. מכיוון שפונקציית ממצעת את ה-reward מאותו המצב והלאה, יש לקבוע מוסכמה איך מתייחסים לסיטואציה בה חוזרים למצב מספר פעמים. שתי האופציות המרכזיות הן First-visit MC ו-Every-visit MC. באופציה הראשונה אנו ממצעים את ה-reward החל מהפעם הראשונה שהגענו למצב ועד סוף ה-episode, ואילו באופציה השנייה אנו ממצעים את ה-return שהתקבל אחרי כל אחד מהביקורים.*

*MC הוא model free. אנחנו לא מנסים למדל באמצעותו את המודל של הסביבה. בנוסף, לא ניתן לבצע אבלואציה ל-state value אלא רק ל- עבור הפוליסה הספציפית. במקום לבצע policy evaluation סטנדרטי על ה-states, נשמור את הערכים שמתקבלים עבור כל state ופעולה ונשערך את . חשוב לציין, שאם הפוליסה דטרמיניסטית, יכול להיווצר מצב שבו ישנם מצבים שלא ביקרנו בהם. פתרון אפשרי הוא להתחיל את ה-episodes כל פעם במצב אחר ועם פעולה אחרת, כך שכל זוגות המצבים-פעולות יידגמו. שיטה זו נקראת exploring starts. ה-policy improvement מתבצע בצורה חמדנית (greedy), כאשר בכל שלב נבחרת הפעולה עם ה- הגבוה ביותר. בצורה הזאת ניתן לבצע policy iteration באמצעות MC.*

*יתרונות בשיטת MC:*

* *ניתן בעזרתו ללמוד בצורה ישירה את מהסביבה*
* *אין צורך למדל את הסביבה (model free)*
* *אין צורך ללמוד את כל המצבים האפשריים*

*חסרונות בשיטת MC:*

* *עובד רק במקרים של מסלולים סופיים (episodic)*
* *לומד רק מ-episodes שלמים, ולא יכול לבצע bootstrapping במהלך ה-episode*
* *חייב לחכות עד סוף ה-episode כדי לדעת מה ה-return ולעדכן בהתאם*

*שיטת TD(0), אותה נראה בהמשך, מתמודדת עם בעיות אלה.*

## **On Policy ו-Off Policy**

קיימות שתי שיטות מרכזיות בנוגע ליחס בין **הפוליסה הנלמדת** לזו **שבה נעשה שימוש ליצירת ה-data**:

* On Policy – בשיטות אלה הפוליסה הנלמדת **זהה** לזו אשר נעשה בה שימוש ביצירת ה-data.
* Off Policy – בשיטות אלה הפוליסה הנלמדת **שונה** מזו אשר נעשה בה שימוש ביצירת ה-data.

היתרונות בשיטות On Policy – הלמידה יעילה יותר (ומהירה יותר), מכיוון שאין הבדל בהתפלגות בין ה-data שממנו לומדים לבין המודל שאותו לומדים. לעומת זאת, בשיטות Off Policy, קיים הבדל בהתפלגויות, ולכן נדרשות שיטות ייעודיות להתגבר על הבדל זה, מה שמוביל ללמידה איטית ומאתגרת יותר.

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

נשים לב לעובדה נוספת, שבשיטות On Policy יותר בולטת ה-exploration-exploitation trade-off. הפוליסה הנלמדת וזו אשר נעשה בה שימוש זהות, ולכן אם נעקוב אך ורק אחרי הפוליסה שנראית לנו אופטימלית ברגע זה – לא נבצע אקספלורציה. מסיבה זו, אנחנו מודעים לכך שהפוליסה הנלמדת תהיה תת-אופטימלית, שכן מוכרח שיהיה בה מרכיב של אקספלורציה.

אתגר נוסף הקיים בשיטות Off Policy הוא שיתכן מאוד שיש **מידע חסר**, למשל מצבים שבהם לא ביקרנו או פעולות שלא ביצענו. במקרה כזה הסוכן לא ידע איך להתנהג כאשר יגיע אליהם תחת הפוליסה הנלמדת.

**פתרונות לאתגרי ה-Off Policy**

נגדיר את הפוליסה הנלמדת כ-target policy ( ואם הפוליסה שממנה לומדים כ-behavior policy (), המטרה היא ללמוד את ה-target באמצעות ה-behavior. נניח ש- היא סטוכסטית, כך שניתן להגיע באמצעותה לכל מצב, ו- היא דטרמיניסטית. על מנת להעריך את התוחלת של ה-reward שנקבל אם נבצע את ניתן להשתמש בשיטה בשם importance sampling. ההסתברות לקבלת מסלול מסוים תחת פוליסה היא:

*נגדיר את היחס בין ההסתברות לבצע את המסלול תחת פוליסה להסתברות לבצע אותו תחת פוליסה :*

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

*כאשר הוא קבוצת כל ה-time stamps שבהם ביקרנו במצב . היחס שבין ההסתברות של הפוליסה ל- מאפשרת לתת משקל מתאים ל-.*

## **Temporal Difference Learning**

ראינו מוקדם יותר כי:

כאשר שיטת MC משערכת את הפונקציה באמצעות (דגימה של ), ושיטות ה-Dynamic programming לסוגיהן משערכות באמצעות (חישוב ישיר של התוחלת באמצעות ידיעת הדינמיקה של העולם). שיטת TD(0) משלבת את שני העולמות, היא מבוססת דגימה (ולכן יכולה לעבוד גם בלי ידיעת הדינמיקה של העולם), ומשתמשת ב- משוערכת, במקום בערך המדוייק (ולכן מאפשרת עדכון תוך כדי episode וכן התמודדות עם מסלולים אינסופיים).

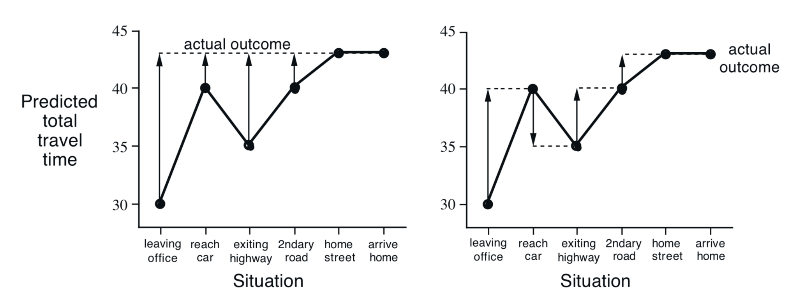
עוד הבדל בין MC ל-TD(0) הוא ש-MC משערך את בעוד ש-TD(0) משערך את . כלל העדכון של TD(0) הוא:

הביטוי נקרא TD error. יש להעיר כי העדכון תלוי בידיעת ולכן נדרש לחכות שלב לפני העדכון. ההנחה היא ש- לא משתנה ממצב למצב.

להלן דוגמה להמחשת TD(0):

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

בדוגמה זאת, בכל נקודה בזמן מחושב הזמן שעבר (שמקביל ל-reward שהתקבל בstate) והזמן המשוערך שנשאר (שמקביל לשערוך של Q של המצב הבא). בכל מצב אנו מתקנים את ההערכה הקודמת שלנו (( לפי התצפית שביצענו () ועוד ההערכה שלנו בשלב זה ((). התרשים שלהלן מדגים את ההבדל בין MC ל-TD(0):



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

## **SARSA**

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

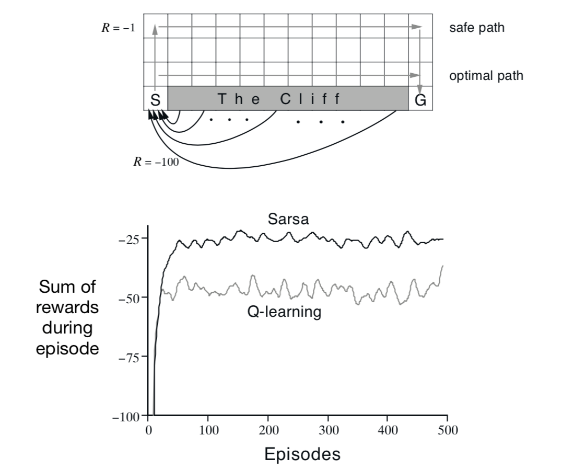
גם כאן מוגדר TD error:

## **Off-Policy TD Control: Q-Learning**

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

פעולת ה-max גורמת לכך שנלמד את הפוליסה האופטימלית, והיא זו שגורמת לאלגוריתם להיות off policy. עוד הבדל בין SARSA ל-Q-learning זה שב-SARSA אנו מעוניינים להעריך פוליסה ספציפית, ולכן נעקוב אחרי הפוליסה הזאת במהלך האלגוריתם, בעוד שב-Q-learning עוקבים אחרי הפוליסה 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 בצורה המהירה ביותר (ה-reward הוא -1 על כלtime step ו—100 על נפילה מהצוק). בשני המקרים אנו מבצעים פוליסה שהיא ϵ-greedy. ההבדל הוא, ש-Q-learning לומד את הפוליסה האופטימלית (off policy), שכן הוא מניח שהחל מהפעולה הבאה נבחר את כל הפעולות האופטימליות. הוא לא מתחשב בכך שהפוליסה היא ϵ-greedy ולכן בצורה מובנית לא נבחר את הפעולה האופטימלית בהסתברות מסוימת. בסופו של דבר, הפוליסה הנלמדת היא אכן האופטימלית, אך מכיוון שלכל האורך אנו לא עוקבים במדויק אחריה אלא מבצעים אקספלורציה, הסוכן בהכרח הוא יפול מהצוק בחלק מהמקרים. לכן בגרף רואים שה-return שלו נמוך בממוצע מ-SARSA. SARSA לעומת זאת מבצעת אופטימיזציה על הפוליסה עצמה, ולכן אם הפוליסה היא ϵ-greedy, האלגוריתם יקח בחשבון שישנה הסתברות שתיבחר פעולה אקראית, ולכן הוא יבחר את הפוליסה ה"בטוחה" יותר שבתוחלת מניבה את הרווח הגבוה.

# **Deep Q-learning**

## **Fixed Q-targets**

ב-deep Q-learning אנו משתמשים ברשת על מנת למדל את ה-Q-function . את הלמידה מבצעים על ידי מינימיזציה של ה-MSE:

ה-**loss function** מוגדר:

וה-backpropagation מתבצע לפי כלל העדכון הבא:

לעיתים, קשה לאלגוריתם להתכנס. חלק מהסיבות הן שהדגימות אינן בלתי תלויות (למשל רצף ארוך של לכל אורך המסלול, כאשר בסופו יש צעד אחד ש-), וכן שמתבצעת אופטימיזציה במקביל על "הדגימה" ועל "התיוג" , או במילים אחרות, ה-target אינו סטציונרי. דרך אחת להתגבר על הבעיה האחרונה היא "להקפיא" את הפרמטרים של רשת הtarget כך שהלמידה תהיה יציבה יותר:  
כאשר אחת למספר צעדים מתבצע עדכון .

## **Experience replay**

בעיה נוספת, אשר קשורה לבעיית התלות שהזכרנו לעיל, היא בעיית השכחה. מכיוון שהדגימות שהרשת רואה הן טוריות, הרשת עלולה "לדרוס" את הניסיון שהיא צברה בעבר, למרות שהוא יכול להיות רלוונטי לעתיד. למשל, במעבר משלב לשלב במשחק מחשב, הרשת עלולה לשכוח את מה שלמדה בשלב הראשון מכיוון שלרוב, התפלגות ה-data בשלב הבא שונה מזו שבשלב הקודם. בעיה נוספת (שהזכרנו לעיל) היא בעיית הקורלציה בין הדגימות. ב-supervised learning בעיה זו לא קיימת מכיוון שניתן להניח שהדגימות בלתי תלויות (למשל מתבצע shuffle של הדגימות בכל epoch). לעומת זאת, בעולם ה-RL לא ניתן להניח דבר כזה, ולמידה של רשת הינה מאתגרת במצב זה (קיים חשש ל-overfitting לעבר הקרוב). Experience replay הוא פתרון לשתי הבעיות הללו. בשיטה זו, אנו שומרים דגימות ישנות על מנת להשתמש בהן שוב במהלך האימון בעתיד.

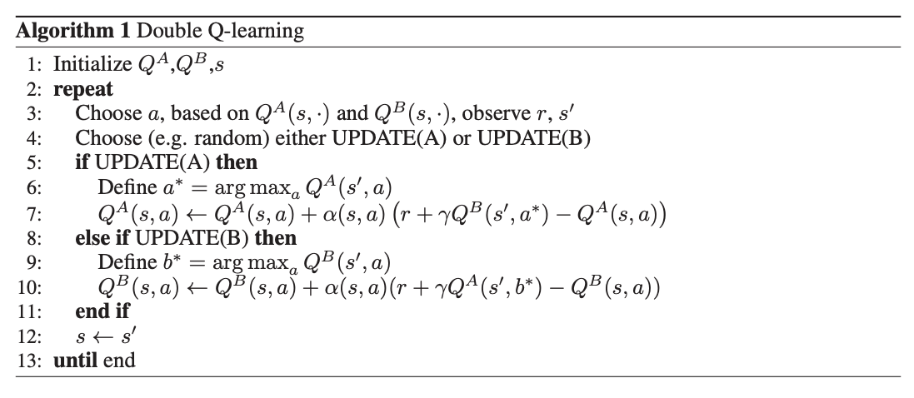
השאלה היא באיזו צורה להציג את הדגימות מחדש לרשת. הדרך הפשוטה ביותר היא האקראית – דגימה יוניפורמית. שיטה זו גם יעילה בהרבה מקרים. אך קיימים מקרים בהן האינפורמציה החשובה לא מפולגת יוניפורמית בין דגימות העבר. קיימות שתי שיטות לדגימה מתוחכמת יותר: דגימה מבוססת TD error ו-stochastic prioritization. ההנחה מאחורי השיטה הראשונה היא שאנו מחפשים דגימות אינפורמטיביות לרשת. דגימות בעלות שגיאה גבוהה מעידות על כך שהרשת "מופתעת" מהתיוג שלהן, ולכן הן תהינה אינפורמטיביות יותר ללמידה. בשיטה זו, ההסתברות להשתמש בדגימה שוב תהיה פרופורציונלית לגודל ה-TD error.

הבעיה בשיטה זו היא שמכיוון שהיא מבוססת על אלגוריתם חמדן (greedy) – מציאת ה-TD error הגבוה ביותר ברגע זה – חלק מהדגימות לא תידגמנה זמן רב ואף עלולות להימחק מה-buffer. זה יכול להיות בעייתי למשל בסביבה בה ה-reward סטוכסטי. אותה דגימה מקבלת חלק מהפעמים reward וחלק מהפעמים לא, ותהליך הדגימה מוטה (יש אפשרות למשל שה-TD error יהיה גדול בפעמים בהם קיים reward ורק הדגימות הללו יוצגו לרשת). בנוסף יש חשש ל-overfitting לדגימות שיוצגו שוב ושוב לרשת. לכן, שיטת ה-stochastic prioritization מנסה למצוא פשרה בין דגימה של דגימות אינפורמטיביות לדגימה אקראית. שיטה זו משתמשת במשוואה הבאה לדגימה מתוך ההיסטוריה:

כאשר היא ההסתברות לדגימה של דוגמה i לפי ה-TD error, ו- הוא פרמטר החלקה. אם נבחר הדגימה תהיה יוניפורמית, ועבור הדגימה תהיה זהה לשיטה הקודמת. p יכול להיות מוגדר כך: כאשר הוא ה-TD error ו- הוא מספר קטן אשר מוודא שישנו סיכוי קטן לכל דגימה. לחילופין, ניתן להגדיר כאשר הוא המקום של דגימה i אם ממיינים את הדגימות לפי ה-TD error.

## **Double DQN**

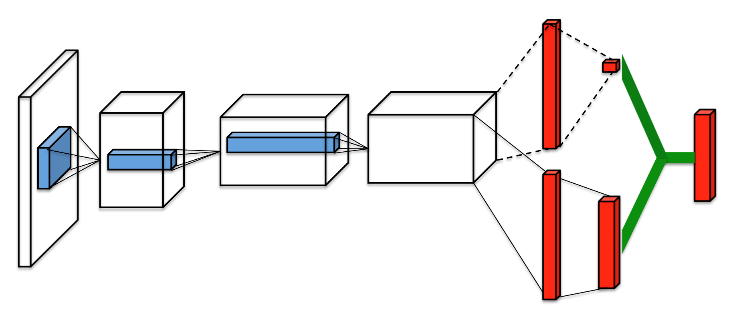
אחד מהקשיים באימון רשת DQN הוא חוסר הדיוק של רשת בחיזוי Q-values. אם פעולה שאינה אופטימלית אותחלה עם Q-value גבוה – יקח זמן לערך להתעדכן. בעיה זו נובעת מכך שהאלגוריתם בוחר בתור target את ערך ה-Q הגבוה ביותר מתוך כל הערכים, לכן תהיה הטיה לכיוון האופטימי. ניתן להתגבר על הבעיה באמצעות שימוש **בשתי רשתות** במהלך האימון, כאשר כל רשת מאומנת על דגימות שונות. כזכור, ה-TD error מחושב על ידי:  
כאשר ה-target הוא . בשיטה זו, הביטוי משתנה. במקום לקחת את ה-Q-value המקסימלי של הרשת שעליה מבצעים את האופטימיזציה, לוקחים את ה-**Q-value של אותה הפעולה מרשת אחרת**. במילים אחרות, מבצעים שני שלבים. בשלב ראשון מחשבים את הפעולה שמניבה ערך מקסימלי בפונקציה , ובשלב שני מחשבים את ה-Q-value של אותה הפעולה ברשת השנייה . שיטה זו מאפשרת לקבל משערך לא מוטה (unbiased) ל-Q-value המקסימלי. להלן האלגוריתם המלא:



## **Dueling DQN**

*המטרה של Dueling DQN היא הפרדת הצימוד שבין ערך המצב (state value) לערך הפעולה (action value). הפרדה זו מאפשרת לרשת להתמקד בהבדלים שבין הפעולות האפשריות 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. הרשת מוציאה ערך Q אשר מורכב משני ערכים:*

*להלן איור של הארכיטקטורה:*



*במהלך האימון והפרדיקציה, מתבצע נירמול של כך שהממוצע יהיה אפס (מחסרים את הממוצע מכל אחד מהערכים). זה מאפשר למידה של* ***הייתרון*** *של פעולה ביחס לשניה, במקום הערך המוחלט שלה:*

# **Policy Gradients**

*השיטות שראינו עד עכשיו מבוססות על value function ועל action value function. כעת נציג שיטות אשר מבוססות על חיזוי ישיר של הפוליסה. בשיטות אלו משתמשים ב-gradient ascent לפרמטרים של המודל על מנת למקסם את ה-reward שמתקבל מהפעולות שהמודל מציע. במטלה עם פעולות בדידות המודל יוציא softmax על אוסף הפעולות, ובמטלה עם פעולות רציפות המודל יוציא את התוחלת והשונות של גאוסיאן שממנו נדגום את הפעולה.*

*יתרון שיש לשיטות אלו על פני שיטות ϵ-greedy, שהפעולות משתנות בצורה רציפה, ללא שינויים חדים. כאשר מתקבלת פרדיקציה ממודל הפוליסה, ניתן לדגום את הפעולות לפי ההתפלגות שהתקבלה והתפלגות זו יכולה להשתנות בצורה רציפה ואיטית במהלך הלמידה.*

*המשוואה הבסיסית ב-policy gradients היא:*

*הוא מספר הפעמים הממוצע שמבקרים במצב .*

*יתרונות ל-policy gradients:*

* *ניתן למדל בצורה טבעית את ההסתברויות לבחירת כל פעולה*
* *ניתן בצורה טבעית לקבל איזון בין exploration ו-exploitation*
* *ניתן למדל בצורה טבעית מרחבי פעולות רציפים*
* *בחלק מהמקרים אפשר למדל בעיה בצורה יותר קלה באמצעות policy gradients*

*חסרונות ל-policy gradients:*

* *השיטה דורשת הרבה דגימה (שכן בכל פעם שהפוליסה מעט משתנה דגימות העבר כבר אינן מייצגות אותה)*
* *במקרים רבים השיטה מתכנסת לאט יותר משיטות אחרות*

## **REINFORCE**

*את הביטוי שלעיל ניתן לפתח לצורה הבאה:  
כאשר הוא משתנה מקרי המכיל את המצב (מתפלג לפי ). נבצע עוד טריק מתמטי:  
כעת ניתן להבחין שהביטוי הוא בעצם התוחלת על כאשר הוא משתנה מקרי של הפעולות, מפולג לפי, במילים אחרות, :*

*כעת, ניתן להשתמש בשיטות דגימה כמו Monte Carlo על מנת לשערך את התוחלת הנ"ל. השיטה שמשתמשת בזה נקראת REINFORCE, וכלל העדכון למשקולות בשיטה זו הוא:  
האינטואיציה היא שכיוון הגרדיאנט הוא הכיוון שגורם לרשת לבצע את בהסתברות הגדולה ביותר. את הגרדיאנט הזה מכפילים ב-, כך שאם ה-reward יהיה חיובי נבצע עדכון על מנת להגדיל את ההסתברות לבצע את פעולה , ואם יהיה שלילי נקטין אותה. בנוסף, במכנה נמצאת ההסתברות לביצוע הפעולה, נרמול זה קורה מכיוון שאנחנו רוצים לעדכן באופן שווה את כל הפעולות שנלקחות, ולא לתת משקל גבוה יותר (כלומר מספר עדכונים גבוה יותר) לפעולה שתיבחר בהסתברות גבוהה.*

*מכיוון שבשיטה זו אנו זקוקים להערכה של , ניתן לבצע אותה רק בתום episode, ולכן היא מתאימה רק לבעיות episodic.*

## **REINFORCE with Baseline**

*מכיוון שאלגוריתם REINFORCE מבוסס דגימה, ה-reward עלול להיות רועש, ויהיה קשה להגיע איתו להתכנסות. אחד הפתרונות לבעיה זו הוא להוסיף baseline, כלומר נירמול ערכי העדכון כך שהתוחלת שלהם על פני כל הפעולות יהיה אפס:*

*בחירה טבעית ל- תהיה state-value function, , שכן היא אינה תלויה בפעולה והיא תנרמל את הביטוי כך שהתוחלת תהיה אפס. כך הלמידה תהיה – איזו פעולה טובה יותר מהממוצע ואיזה גרועה יותר. ניתן לשערך את הפונקציה על ידי דגימה.*

# **Actor Critic**

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

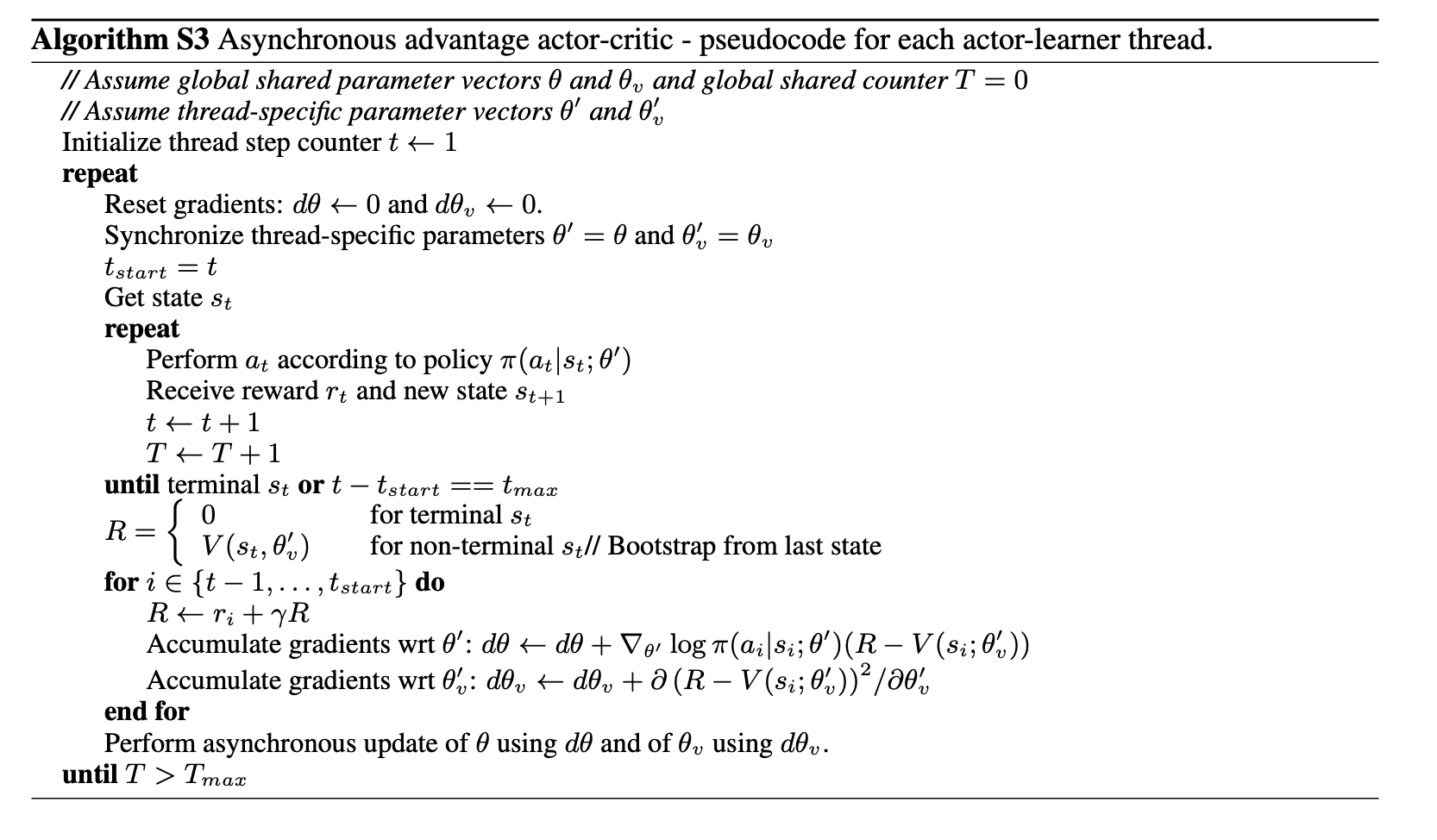
*בשיטות actor critic, מבצעים בצורה איטרטיבית עדכון לפוליסה ול-state value function. ה-actor יכול להיות מבוסס Q-function (במרחבי מצבים קטן), או policy gradients. ה-critic יכול להשתמש ב-data שנוצר באמצעות פוליסה אחרת, וכן ניתן להשתמש ב-bootstrapping.*

*האלגוריתם הבסיסי של actor critic מזכיר שילוב של REINFORCEMENT with baseline ושל TD(0). בכל שלב באיטרציה נעדכן את* ***הפוליסה*** *ואת ה-****value function****. עדכון הפוליסה:  
כאשר .*

*עדכון הvalue function-:*

## **Asynchronous Advantage Actor-Critic (A3C)**

כאשר עוברים לשיטות actor critic מתווספת לנו גמישות לבצע שיטות התייעלות שונות באימון. דוגמה לכך היא ה-A3C. בשיטה זו מנצלים את העובדה שה-critic אינווריאנטי לפוליסה שנבחרת, כלומר תלוי רק במצב ולא בפעולות או בפוליסה. לכן, ניתן למקבל מספר actors שכולם משתמשים באותו critic (כלומר אותה value function) וכולם מעדכנים אותו. ניתן אפילו שכל אחד יעקוב אחרי פוליסה אחרת (ולבצע importance sampling). העדכון של ה-value function מתבצע אחת למספר צעדים, ולכן נקרא asynchronous. בשיטה זו הגרדיאנטים מתווספים (accumulating gradients) ומתבצע איתם עדכון רק לאחר מספר מוגדר מראש של צעדים. להלן פירוט האלגוריתם (באלגוריתם זה נעשה שימוש באותה פוליסה ללא importance sampling):



# **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). נתמקד במספר שיטות – **למידה של מודל סביבה** באמצעות דוגמאות המומחה (apprenticeship learning), **למידה ישירה של פוליסה** מאותם דוגמאות (behavioral cloning), **למידה בסביבה משתנה** (forward training) **ולמידה מתמשכת** בסביבה סטטית (DAgger).

## **מונחים**

**Regret** הוא ההפרש בין ה-reward שיתקבל במקרה של פוליסה אופטימלית לזה המתקבל באמצעות הפוליסה הנוכחית. כמובן שבמהלך הלמידה נרצה למזער את ה-regret ככל הניתן ובכך להתקרב לפוליסה האופטימלית. נגדיר את ה-loss עבור צעד יחיד:

כאשר הוא תוחלת ה-reward בצעד זה כאשר נבצע את הפעולה האופטימלית , ו-

הוא תוחלת ה-reward עבור הצעד הנוכחי (לא להתבלבל עם ה-Q-function, פונקציית מתייחסת לתוחלת ה-reward **בצעד הנוכחי**, ולא ממוצע שלו על פני הזמן). אם כן הוא ההפרש בין ה-reward הממוצע שיתקבל תחת פוליסה אופטימלית, לזה שיתקבל אם נבצע את הפוליסה הנוכחית. ה-regret הכולל מוגדר כ:

*ממוצע ה- על פני הזמן. במהלך הלמידה נרצה למזער את . אפשר לבטא את באמצעות נוטציה קצת שונה:*

*כאשר הוא מספר הפעמים שהתבצעה פעולה (count), ו- מוגדר כ- (gap).*

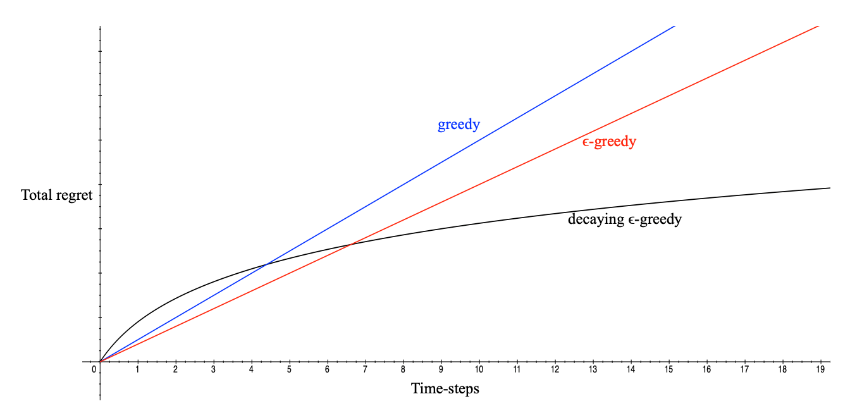
*על מנת לכמת איזה אלגוריתם לומד יותר טוב על פני הזמן, נוהגים להשתמש במונח ה-total regret. מונח זה מתייחס ל-regret המצטבר כפונקציה של הזמן. למשל אלגוריתם חמדן (greedy) הממזער את ה-loss עבור הצעד הקרוב:*

*ה-total regret שלו יהיה לינארי, מכיוון שהוא לא מבצע אקספלורציה הוא יכול להיות "תקוע" בפוליסה שאינה אופטימלית, ובמקרה כזה ה-regret יעלה בצורה לינארית כתלות בזמן. לאלגוריתמים ϵ-greedy ה-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 בגלל האקראיות שלהם):*

*הביטוי מורכב מה-loss הממוצע על פני כל הפעולות וההסתברות לבצע אותן . גם אם הפוליסה שלנו אופטימלית, ו- כאשר מבצעים exploitation, במקרים בהם מתבצעת אקספלורציה (בהסתברות ) ה-regret הממוצע יהיה , ולכן חסום בהכרח. לאלגוריתמים ϵ-greedy גם יהיה total regret לינארי (אם כי, כאמור, נמוך יותר מאלגוריתם חמדן).*

*פתרון אפשרי להורדת ה-total regret הוא שימוש ב*-*Decaying 𝜖-Greedy. אם ההקטנה של ה- מספיק איטית כך שמתאפשר תהליך אקספלורציה, ניתן להוריד את ה-total regret מלינארי. דבר זה מתאפשר מכיוון שככל שעובר הזמן אנחנו פחות ופחות מבצעים פעולות רנדומליות, ומתקרבים לפוליסה האופטימלית. למשל, אם בוחרים את בצורה הבאה:*

*כאשר הוא קבוע, הוא הזמן ו- (אלא אם כן ) הוא ה-gap המינימלי, ה-total regret ירד אסמפטוטית* ***כמו פונקציית*** *. הבעיה היא שבדרך כלל לא נדע את הפוליסה האופטימלית, ולכן לא נדע את . להלן גרף המתאר את ה-total regret כתלות בזמן עבור שלושת האלגוריתמים:*



## **Imitation Learning**

*קיימים מספר אתגרים ייחודיים כאשר מתמודדים עם בעיות בעולם ה-RL. בשונה מבעיות קלאסיות בתחום ה-supervised learning, סוכן RL נדרש לבצע פעולות סדרתיות, כאשר המצבים בהם הוא נתקל והפעולות שאותן הוא מבצע* ***אינן בלתי תלויות****. תכונה זו הינה מאתגרת לאלגוריתמי בינה מלאכותית (רשתות נוירונים). בנוסף, הבעיות במקרים רבים מורכבות מממדים רבים (למשל סוכן המתבסס על מצלמה להחלטה על פעולות), הבעיות הרבה פעמים לא לינאריות (שינוי קטן בפעולה גורם לשינוי גדול בתוצאה), וקיים גם ממד של אקראיות, במיוחד בבעיות עולם אמיתי.*

*פתרון אפשרי לבעיות אלה הוא* ***לנסות לקרב את הבעיה לבעיית supervised learning פשוטה****, על ידי המרת הבעיה לסדרה של בעיות פרדיקציה. ב-imitation learning אנו משתמשים בפעולות שמומחה אנושי ביצע, ומלמדים את הסוכן לבצע פעולה בהינתן מצב.*

*הבעיה היא שגישה זו היא מאוד עדינה, ויכולה שלא להיות יציבה במקרים רבים. למשל, אם מתבצעת למידה ישירה ממומחה, הסוכן ילמד את הפעולות האופטימליות עבור מצבים שהמומחה נתקל בהם, אבל יתכן כי יתקל במצבים אחרים שונים והוא יתקשה להכליל. בעיית ה-"****compounding error****" מתייחסת למקרים בהם סטייה קטנה בפרדיקציה יכולה לגרור סטיות משמעותיות במעלה הדרך, מכיוון שהסטייה הקטנה גרמה לסוכן לפגוש מצבים שהוא לא מכיר, מה שגורם ל"שגיאה נגררת". במצבים כאלה הסוכן יתקשה לחזור למצבים מוכרים. בהמשך נציג מספר גישות להתמודדות עם בעיה זאת.*

*נקודת מבט אחרת על הבעייתיות ב-imitation learning היא שאנו מעוניינים שהסוכן ידע להתמודד עם* ***כל מצב*** *שהוא יפגוש. אך אם אנו מתבססים רק על דגימות שנאספו ממומחה, ה-dataset שלנו סופי ולא מכסה את כל עולם הבעיה. אלגוריתמים שנפגוש בהמשך מנסים לפתור זאת על ידי מציאה של "מצבים מעניינים", כלומר זיהוי איזה מצבים יכולים להיות משמעותיים בלמידה, ו"תיוג" שלהם (על ידי המומחה) בצורה איטרטיבית.*

## **Apprenticeship Learning**

Apprenticeship Learning הוא אחת מהשיטות הבסיסיות ביותר של Imitation learning. בשיטה זו נעשה שימוש ב-data שנאסף על ידי expert (למשל על ידי אדם שביצע את המשימה) על מנת **ללמוד את הדינמיקה של העולם**. לאחר שהדינמיקה נלמדה, לומדים פוליסה על בסיס המודל של העולם באמצעות אלגוריתמי RL בסיסיים. במידה שהפוליסה שנלמדה לא טובה מספיק, מה שמעיד על כך שהמודל של העולם לא נלמד טוב, נאסוף את המצבים החדשים בהם נתקלנו ו"נתייג" אותם מחדש על ידי המומחה. בצורה איטרטיבית האלגוריתם יתכנס לביצועים טובים.

שיטה כזאת מתאימה למקרים בהם קשה לבצע אקספלורציה, למשל כאשר איסוף כזה data הוא יקר. החסרונות המרכזיים של השיטה הם שלא ניתן להתמודד עם מצבים (states) שלא נצפו בעבר, שכן מה שנלמד זהו מודל של הסביבה, ומודל זה נלמד אך ורק באמצעות המצבים שהמומחה נתקל בהם. וכן, מאותה סיבה, קשה בשיטה כזאת להתגבר על טעויות (בעיית ה-compounding error).

מהנאמר לעיל נובע כי שיטה זו מתאימה למצבים בהם מרחב המצבים קטן ופשוט 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 במפעל ונדרש להרים חפצים מעל משטח, או למידה של משחק שחמט.

לעומת זאת, בשיטות Imitation learning מורכבות יותר, אנו מניחים כי המודל של הסביבה מורכב, ולא מתיימרים ללמוד את הדינמיקה של העולם. שיטות אלו מתרכזות בלמידה של הפעולות עצמן אותן נדרש הסוכן לבצע, הן כלליות יותר ומתאימות לבעיות מורכבות יותר ומוגדרות פחות. מכיוון שאנו לא מסתמכים על למידה של מודל הסביבה, הסוכן יכול "להכליל" למצבים דומים לאלה שראה, ויכול להתגבר על טעויות. מצד שני, שיטות אלו הן בדרך כלל קשות יותר ללמידה וההתכנסות לוקחת יותר זמן, ולכן 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 היעילות שלה והפשטות שלה – לומדים ישירות לבצע את אותן הפעולות כמו המומחה. שיטה זו יכולה גם להוות בסיס טוב להמשך למידה בשיטות אחרות. כמובן שהחיסרון העיקרי של שיטה זו הוא הקושי להכליל והסכנה מ-overfitting. חיסרון זה קיים במיוחד כאשר כמות ה-data מוגבלת ולא מספיק מגוונת. בנוסף, קיימת הבעיה של שגיאות מצטברות (compounding errors), שכן שגיאה אחת עלולה להביא את הסוכן לאזורים במרחב המצבים אותם הוא לא מכיר. שיטה זו יכולה להתאים למקרים בהם המשימה מוגדרת היטב, הפעולות של המומחה קונסיסטנטיות ומייצגות את מרחב המצבים אותו הסוכן יפגוש. שיטה זו יכולה להתאים למשל לנהיגה אוטונומית בסימולציה או משחק מחשב, רובוט אשר נדרש לבצע משימה פשוטה או משימות אחרות במרחבי מצבים קטנים יחסית עם משימות יחסית מוגדרות היטב.

HAVOC, ראשי תיבות של Hierarchical behAViOral Cloning, היא שיטה ללמידה היררכית באמצעות Behavioral Cloning. בשיטה זו מלמדים מספר סוכנים לחכות מספר שחקנים. למשל, בהינתן dataset של מספר שחקני 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 יעילה ב-Behavioral cloning למרות המגבלות שלה.

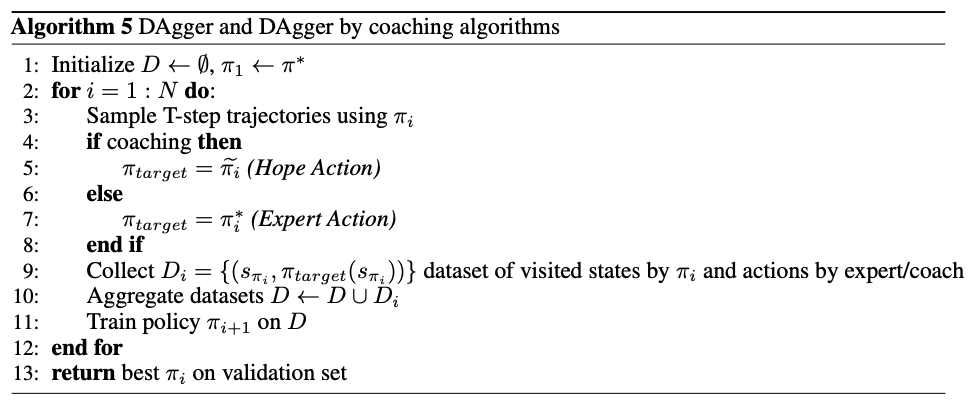
## **Forward Training**

שיטת forward training מתמודדת עם מצב בו נדרשת **פוליסה שאינה סטציונרית**, למשל כשהסביבה משתנה עם הזמן. הרעיון המרכזי בשיטה זו הוא שבשלב נאספות דגימות בעזרת פוליסה , אשר נשלחות לתיוג, ובעזרתן מאומנת פוליסה . שיטה זו מתבצעת בצורה איטרטיבית, כך שבכל שלב מאומן אלגוריתם חדש על דגימות שנאספו מהאלגוריתם שאומן בשלב הקודם. שיטה זו מתאימה למצבים בהם נדרשת פוליסה אשר משתנה עם הזמן. דוגמאות לכך: רובוט שנדרש לבצע משימה בסביבה משתנה (להרים חפץ, כאשר יכולים להימצא חפצים שונים על גבי המשטח), או סוכן שמשחק משחק מחשב בעל מספר שלבים או מצבים. במקרים כאלה, נדרשת התאמה בכל פעם שהסביבה משתנה, וניתן לבצע את ההתאמה על ידי דגימה של מצבים מהסביבה באמצעות הפוליסה שאומנה בסביבה הקודמת. היתרון בשיטה זו הוא שהאלגוריתם מאומן על מצבים ש"נצפו לאחרונה", כלומר שההתפלגות הסטטיסטית שלהם קרובה לזו שהסוכן פוגש כעת במציאות. לכן ה-data שהסוכן מאומן עליו מתאים לזה שהוא יראה ב-test time. חיסרון מרכזי לשיטה הוא שבכל איטרציה של אימון נדרש להעביר את הסוכן את כל הצעדים עד כדי שנוכל לתייג אותם ולאמן מחדש עליהם. במצב בו גדול, זה הופך ללא פרקטי.

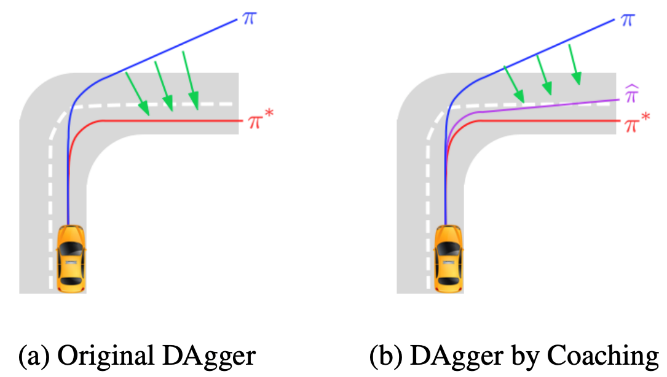
## **DAgger**

*שיטה נוספת ל-imitation learning היא DAgger (Dataset Aggregation). בשונה מ-forward training, שיטה זו מאמנת* ***פוליסה סטציונרית****, כלומר שמניחה שאין שינוי של הסביבה עם הזמן. שיטה זו מאוד דומה לשיטה הקודמת, ההבדל הוא שהאלגוריתם מתאמן על כל ה-data המתויג, גם זה שנאסף בעבר בעזרת פוליסות קודמות. שיטה זו בדרך כלל מביאה להתכנסות מהירה של האלגוריתם. DAgger יכול להתאים במצב שבו הפעולות של המומחה האנושי קונסיסטנטיות, גם כאשר הבעיה יחסית מורכבת. עם זאת, במקרים בהם קיים מגוון גדול מידי של מצבים האלגוריתם יתקשה להכליל. עוד חיסרון באלגוריתם זה הוא, שכמו בforward training, המומחה נדרש לתייג מחדש בכל איטרציה את הפעולות הנדרשות למסלול שהסוכן יצר. דבר זה עלול להקשות בבעיות מהעולם האמיתי.*

*אתגר נוסף בשיטה זו הוא שבמקרים בהם הפוליסה שהסוכן צריך ללמוד מורכבת מידי, הוא יתקשה ללמוד ישירות את הפוליסה המורכבת רק מצפייה בהתבוננות במומחה. באנלוגיה לכך, תלמיד לא יכול ללמוד לנגן כינור ישר מנגן בינלאומי, הוא יצטרך ללמוד בצורה הדרגתית. אלגוריתם Dagger with Coaching בא לפתור בעיה זאת. בשיטת DAgger with Coaching, הסוכן מקבל בתחילת האימון דוגמאות יותר קלות, וככל שעובר הזמן, ככל שהוא משתפר, הוא מקבל דוגמאות מאתגרות יותר. מוגדר hope action:  
אשר קובע מהי הפעולה אותה הסוכן יבצע. הוא ה-loss של המומחה (כמו באלגוריתם הבסיסי), כאשר מתווסף עוד ביטוי אשר מכמת כמה סיכוי יש שהפוליסה הנוכחית תבחר את במצב . ככל שהסיכוי נמוך יותר, כך הפעולה כנראה יותר "מורכבת להבנה" לסוכן בשלב זה ולכן יש פחות סיכוי שנבחר בה. ככל שהלמידה תתקדם, כך יקטן (עד שיגיע ל-0), ויינתן משקל גבוה יותר לפעולות של המומחה. להלן שני אלגוריתמי DAgger:*



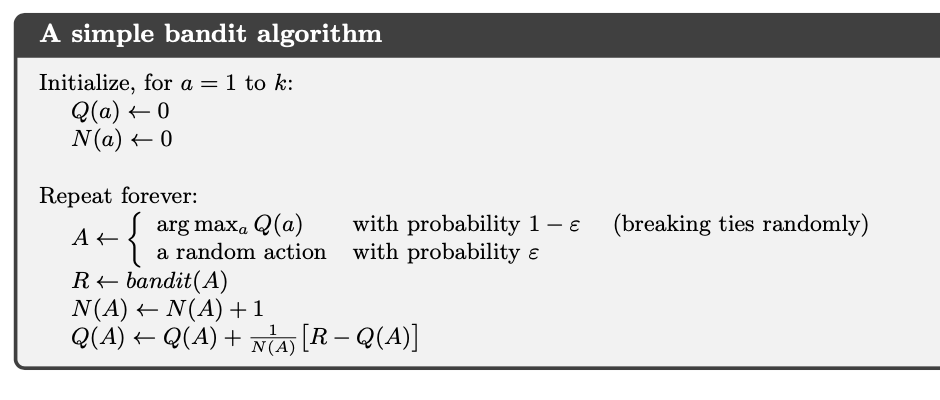
*להלן אילוסטרציה של ההבדל בין האלגוריתמים. כאשר באלגוריתם המקורי יכול להיות הבדל גדול בין הפעולה שהסוכן רצה לבצע (הפרדיקציה) לזו שהמומחה "תייג" , באלגוריתם DAgger with Coaching נבחר פעולה שפחות "מדויקת" אבל קרובה יותר לפרדיקציה של הסוכן בשלב זה.*



# **Multi-arm bandits**

*Multi-arm bandits היא בעיה בסיסית מאוד של RL. בבעיה זו ישנו רק מצב (state) אחד, עם K פעולות אפשריות. ה-reward המתקבל מכל פעולה מתפלג בהתפלגות כלשהי שאינה ידועה. בכל שלב, הסוכן בוחר פעולה אחת מתוך ה-K ומקבל עליה reward. השאלה היא מה הפוליסה שתבטיח את ה-reward המצטבר הגבוה ביותר לאורך זמן. זוהי בעיה מזוקקת של המתח בין exploration ו-exploitation.*

*אלגוריתם בסיסי יכול להיות ϵ-greedy, אשר בהסתברות בוחר את הפעולה שנתנה לנו את הגמול הממוצע הכי גבוה עד כה, ובהסתברות ϵ בוחר פעולה אקראית. במצב כזה, רוב הזמן נבחרת הפעולה שאנחנו מאמינים כעת שהיא אופטימלית, ומצד שני נשמרת האקספלורציה. להלן האלגוריתם:*



*אלגוריתם זה מתאים למקרה הסטציונרי, בו התפלגויות ה-rewards לא משתנות עם הזמן. במקרה בו ההתפלגויות לא סטציונרית נעדיף לתת משקל גדול יותר לדגימות החדשות מאשר לישנות. דרך נפוצה לקבל תכונה זאת היא על ידי constant step-size parameter:*

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

*הבעיות בשיטות ה-* *ϵ-greedy הן:*

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

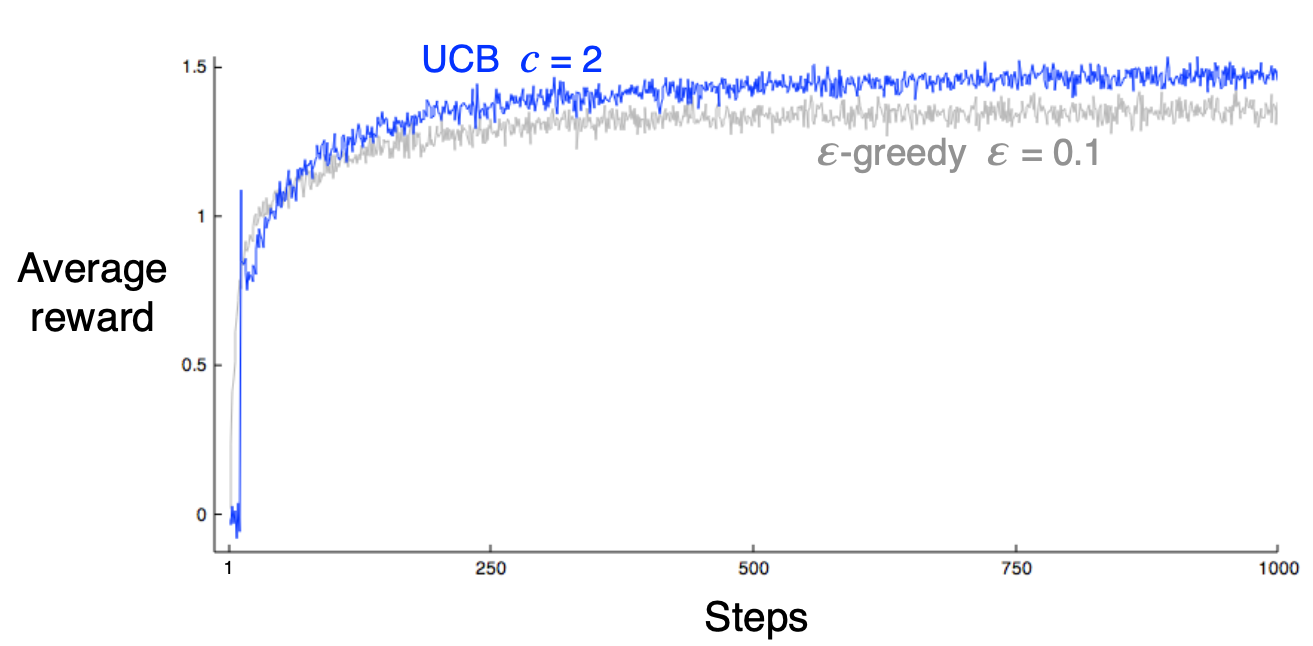
*פתרון אפשרי הוא לעשות אקספלורציה של המצבים בהם אנחנו פחות בטוחים.*

## **Upper Confidence Bound (UCB)**

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

*אינטואיציה: הביטוי השמאלי אחראי על ה-exploitation, והשמאלי על האקספלורציה. הוא אחוז הפעמים שביקרנו במצב . מטרתנו היא שככל שביקרנו יותר במצב מסוים כך נבקר בו פחות בזמן הקרוב, ולכן היה מתאים להשתמש בהפכי: . עם זאת, מכיוון שההנחה היא שעם הזמן אנחנו "מכירים" יותר את הסביבה ואנחנו פחות מעוניינים באקספלורציה, נהפוך את המונה מלינארי ללוגריתמי .*

*חיסרון בשיטה זו הוא ההתמודדות עם התפלגויות לא סטציונריות. מכיוון שהביטוי לא מתחשב בשינוי בהתפלגויות, במצב בו הן משתנות האקספלורציה לא תהיה אופטימלית. חיסרון נוסף בשיטה זו הוא הקושי בהתמודדות מול מרחבי מצבים גדולים. במצבים כאלה קשה מאוד לבצע את האקספלורציה בשיטת UCB. להלן גרף המשווה את UCB ל-ϵ-greedy:*

**

## **Gradient Bandits Algorithm**

*שיטה זו הינה* ***מבוססת גרדיאנטים*** *(מתבצע gradient descent). השיטה מבצעת אופטימיזציה על ההסתברויות לבחירת פעולה מתוך K על בסיס ה-rewards המתקבלים. הסוכן בוחר בכל איטרציה פעולה על בסיס ההסתברויות:*

*ולאחר מכן מתבצע עדכון ל- באמצעות ה-reward שהתקבל. העדכון מתבצע באמצעות:*

## **LinUCB**

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

*אלגוריתם LinUCB בא לפתור את הבעיה של contextual bandit. באלגוריתם זה, תוחלת ה-rewards על כל זרוע ממודלת באמצעות פונקציה לינארית על וקטור המאפיינים של המשתמש:*

*ניתן להגיע לפתרון אנליטי למציאת :*

*כאשר היא מטריצה בגודל ( מספר הדגימות) ו- הוא וקטור התיוגים (ה-reward). להלן האלגוריתם:*

*Text

Description automatically generated*

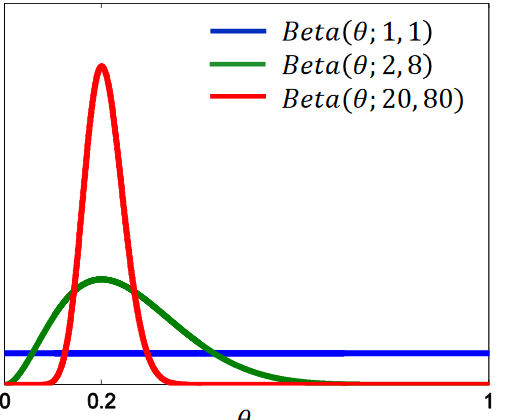
*מבחינת סיבוכיות, האלגוריתם לינארי במספר הזרועות (או המשתמשים) K, וריבועי במספר המאפיינים (features). האלגוריתם מתאים במקרה שה"זרועות" לא קבועות, אלא באות לזמן מסוים ואז נעלמות (למשל במערכת המלצה לכתבות של חדשות).*

## **Thompson Sampling**

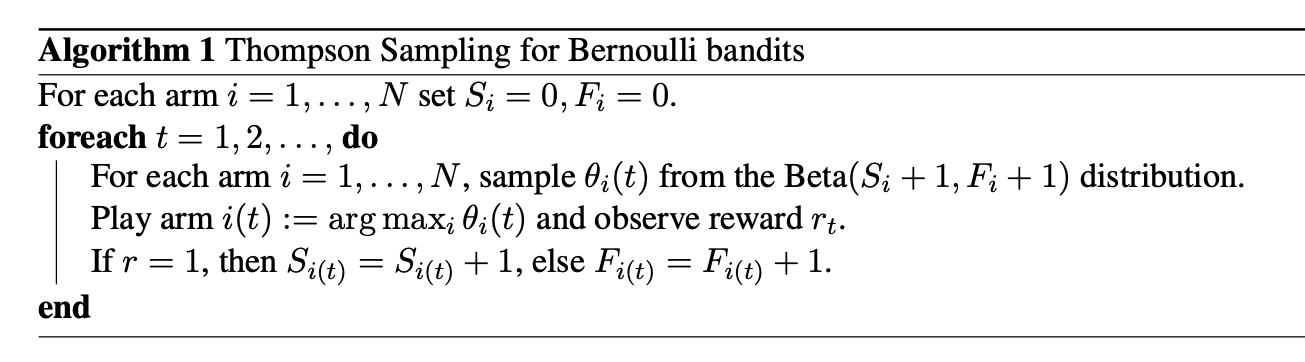
*אלגוריתם זה מניח כי התפלגויות ה-rewards לא משתנות. הוא מתבסס על דגימה בייסיאנית, כלומר בחירה בזרוע בעלת אי-הוודאות הגבוהה ביותר. האלגוריתם משערך פונקציית התפלגות עבור כל זרוע ומתבסס עליה בדגימה. שלבי האלגוריתם הם:*

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

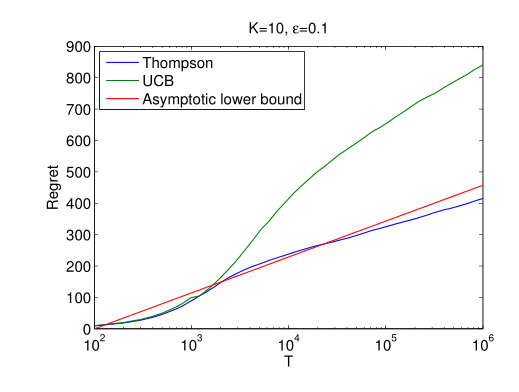
*תהליך זה מתבצע בכל צעד t. עבור ניתן להשתמש בפונקציית . לפונקציה זאת התכונה שהתוחלת שלה היא והשונות שלה קטנה ככל שערכי ה- גדולים. אם נגדיר את כמספר הפעמים שהתקבל ו- כמספר הפעמים שהתקבל , פונקציה זו מתאימה למצב שלנו בו נרצה שבתחילת התהליך תהיה יותר אקספלורציה, וככל שמתקדם התהליך ( גדלים) נתכנס לממוצע.*

**

*להלן תיאור האלגוריתם:*

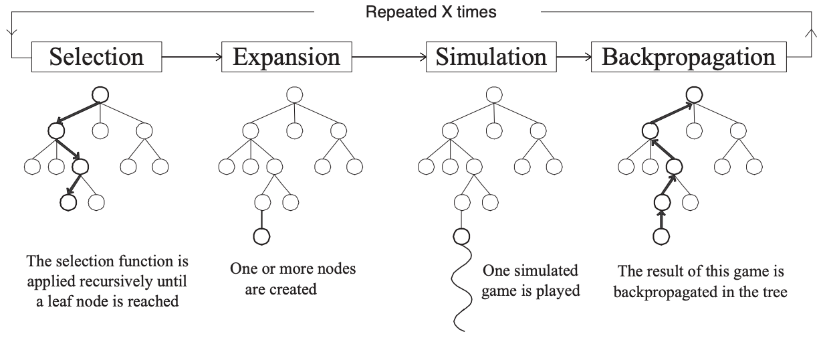
**

*להלן גרף של ה-regret כבפונקציה של הזמן. יש לשים לב שהציר האופקי בסקלה לוגריתמית:*

**

# **Monte Carlo Tree Search**

*אלגוריתם MCTS משלב בתוכו חיפוש בעץ ודגימת Monte Carlo כדי לבצע אקספלורציה של סביבה וללמוד מהי הפעולה האופטימלית לביצוע בכל שלב על מנת למקסם את הרווחים לאורך זמן. האלגוריתם מורכב מארבעה שלבים שמתבצעים בצורה 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 מתבצעת בחירת הצומת שממנה האלגוריתם ימשיך. בתחילת האלגוריתם קיימת רק הצומת הראשונה (root), ולאחר מספר צעדים "נפתחו" צמתים נוספות שאותן אפשר לבחור. הבחירה נעשית על ידי חיפוש בעץ (זהו חלק ה-Tree Search שבאלגוריתם). החיפוש יכול להתבצע למשל על ידי UCB, אשר משלב exploration ו-exploitation:*

*כאשר הוא מספר הפעמים שביקרנו במצב ו- הוא מספר הפעמים שבחרנו בפעולה במצב זה. באמצעות אלגוריתם זה ניתן לבחור את הצומת המתאימה ביותר להמשך התהליך.*

### **Expansion**

*השלב השני באלגוריתם הוא expansion. ישנם שתי אפשרויות. אם המצב שבחרנו הוא עלה (terminal stats), אז ניתן להמשיך לשלב האחרון באיטרציה (backpropagation). אחרת, נבצע הרחבה של העץ למצבים חדשים שלא ראינו. נאתחל את כל המצבים האפשריים הבאים ונבחר מצב אחד על פי פוליסה (rollout policy).*

### **Simulation**

*כעת מגיע שלב ה****סימולציה****. בשלב זה אנחנו מסמלצים משחק (או מספר משחקים אשר ניתן למקבל) החל מהשלב הנבחר ועד לשלב הסופי. במהלך המשחק הפעולות יכולות להיבחר בצורה אקראית או באמצעות rollout policy. כך אנחנו אוספים את ה-rewards מהשלב הנבחר והילך.*

### **Backpropagation**

*כעת נדרש רק לעדכן את באמצעות ה-reward שהתקבל. ניתן לבצע את העדכון בעזרת עדכון ממוצע הזכיות (מול ההפסדים), או לחילופין באמצעות gradient descent לעדכן באמצעות ההפרש בין הערך המצופה לזה שהתקבל.*

## **תכונות MCTS**

* *מכיוון ש-MCTS מתבסס רק על סימולציה, לא נדרש domain-specific knowledge ולא נדרש לתייג data. הדבר היחיד שנדרש הוא ידיעת "חוקי העולם" על מנת לבצע את הסימולציה.*
* *האלגוריתם הינו any-time, במובן שניתן לעצור אותו בכל שלב במהלך האימון ולקבל את הביצועים הטובים ביותר שהתקבלו עד כה.*
* *ניתן לראות אמפירית כי קשה לאלגוריתם לבצע "הקרבות", למשל הקרבה של מלכה בשחמט כדי לנצח את המשחק.*

## **שיפורים לMCTS**

*שני שיפורים אפשריים לאלגוריתם:*

* *שילוב פוליסה (אין הכוונה ל-policy במובנו הרגיל ב-RL)*
* *שילוב value function*

*שילוב פוליסה: במערכת בשם Crazy Stone אשר מטרתה לשחק GO, נעשה שימוש בפוליסה על מנת לבצע גיזום (pruning) לעץ. במערכת זו מחושבים מאפיינים (היוריסטיים) לכל צומת. לאחר שהתבצעו מספר מסוים של סימולציות, הפוליסה קובעת, על בסיס המאפיינים שלעיל, האם צומת זו זוכה להישאר בעץ או שהיא "נגזמת" מהעץ. שיטה זו מאפשרת להישאר עם עץ מצומצם יותר המכיל רק את הצמתים הרלוונטיים ללמידה.*

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

*כאשר הוא וקטור מאפיינים בינרי (hand-crafted) ו- הוא וקטור משקולות נלמד. מנרמלת את הערכים בין 0 ל-1. עם פונקציה כזאת ניתן לבצע דגימה חכמה יותר של מסלולים בשלב הסימולציה. למשל:*

* *ניתן לבצע סימולציות באמצעות פוליסה של ϵ-greedy בהתבסס על .*
* *ניתן להשתמש ב- בתוספת רעש ולבחור בפעולה בצורה חמדנית. בצורה כזאת אחראי על ה-exploitation ו- על ה-exploration.*
* *בחירה של פעולה באמצעות פונקציית ההתפלגות:*(SoftMax על ערכי ). הפרמטר הוא פרמטר טמפרטורה האחראי על החלקת פונקציית ה-SoftMax. ערכים גבוהים שלו יתנו פונקציה הקרובה להתפלגות אחידה, וערכים נמוכים יהפכו את ההתפלגות קרובה יותר ל-one-hot.

*לסיכום, שימוש ב-value function מאפשר דגימה חכמה יותר בשלב הסימולציה, כך ש-MCTS ילמד מדגימות רלוונטיות יותר.*

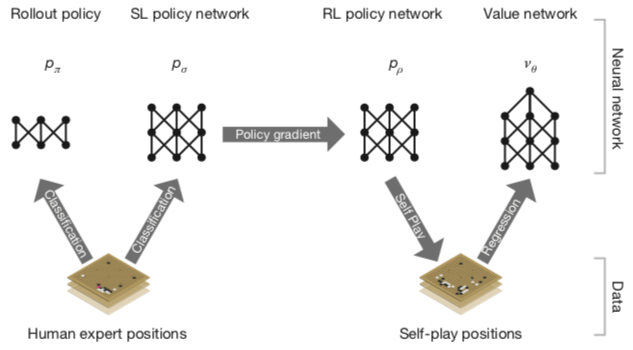
## **AlphaGO**

*משחקים כדוגמת שחמט או GO, מורכבים ממספר מהלכים כאשר בכל מהלך ישנם מספר פעולות אפשריות. אם במשחק ממוצע ישנם מהלכים, ובכל מהלך יש בממוצע פעולות אפשריות, קיימים כ- מצבים, שאם נלמד בצורה רקורסיבית עליהם נוכל ללמוד value function אופטימלית. בשחמט , וב-GO , . לכן, לא ריאלי ללמד על כל המצבים האפשריים, לפחות תחת מגבלות החישוב הקיימות כיום.*

*הארכיטקטורה של AlphaGO מבית DeepMind מורכבת משלוש רשתות:*

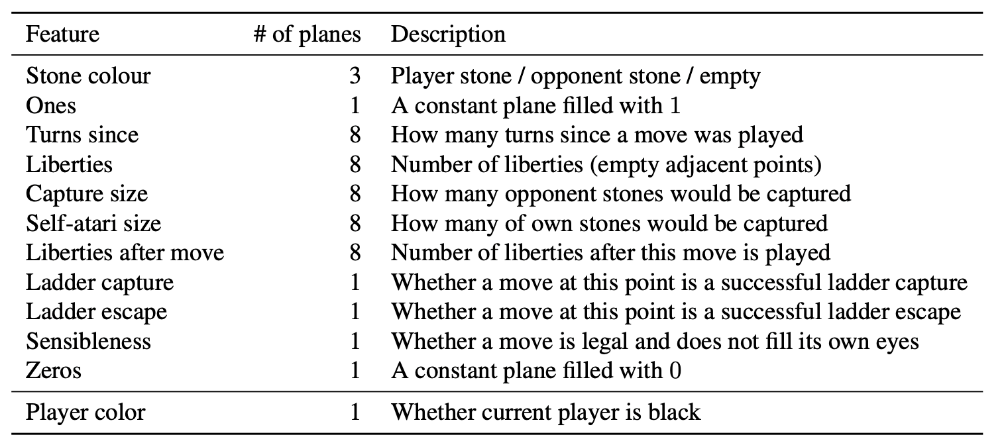
* *Supervised learning policy network (נלמד מה-dataset)*
* *RL policy network (נלמד על ידי self-play)*
* *RL value network (נלמד על ידי self-play)*

*בנוסף, נעשה שימוש במודל לינארי עם SoftMax לטובת ה-rollout בשלב הסימולציה (נלמד מה-dataset).*



### **מרחב הכניסה (input)**

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



### **Supervised learning policy network**

*רשת זו (CNN) אומנה על dataset של משחקי מומחים, אשר מורכב מ160K משחקים ו29.4M מצבים. בנוסף, התבצע data augmentation של סיבובים של הלוח. הרשת חוזה מה המהלך הבא בהינתן מצב הלוח. האימון התבצע במשך שלושה שבועות על 50 GPUs.*

### **RL policy network**

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

*כאשר מאותחל בהתחלה לאפס, ולאחר מכן משתמשים ברשת ה-value כדי להוריד variance. בשלב זה האימון התבצע במשך יום אחד על 50 GPUs עם 10,000 mini-batches של 128 משחקים.*

### **RL value network**

*בשלב זה מאומנת רשת value אשר חוזה עבור הפוליסה הקיימת האם מצב יוביל לניצחון. ארכיטקטורת הרשת זהה לזו של ה-policy network, בשינוי אחד – ישנו ערך אחד בלבד במוצא (במקום ערך לכל פעולה). מתבצע MSE בין מוצא הרשת לבין תוצאת המשחק. ה-dataset מורכב מדגימה אחת בלבד מכל משחק, על מנת ליצור dataset בלתי תלוי.*

### **Rollout Policy**

*ה-rollout הוא הפוליסה שבאמצעותה דוגמים את הסימולציות. המטרה היא ליצור אלגוריתם מהיר, שיוכל לבצע את הסימולציה בצורה מהירה. נעשה שימוש במסווג לינארי, אשר מאומן על מאפיינים שמוגדרים מראש. להלן המאפיינים:*

**

*המאפיינים מוגדרים ממטריצה בינרית מסביב לכל פעולה חוקית (אשר מסמנת האם המקומות תפוסים), סטטיסטיקות של מצב הלוח ומאפיינים שקשורים למשחק GO עצמו. המסווג אומן על מיליוני דגימות מתוך dataset אינטרנטי.*

### **תהליך האימון**

*כזכור, האלגוריתם מתחיל בבחירה של מצב שממנו להתחיל – שלב ה-selection. האלגוריתם מתחיל מה-root ו"מטייל" לעומק העץ עד שהוא בוחר בעלה. החיפוש נעשה* ***באמצעות ה-policy network*** *עם אקספלורציה. ככל שעובר הזמן, נעשית פחות אקספלורציה.*

*לאחר שנבחר עלה מסוים, מתבצעת סימולציה* ***באמצעות ה-rollout policy****. הערך של העלה נקבע על ידי ממוצע משוקלל של ה-value network ותוצאת המשחק:*

*תהליך הסימולציה מתבצע מספר פעמים, וכאשר מספר זה חוצה רף – העלה מתווסף לעץ.*

## **AlphaGoZero**

*ה"דור הבא" של AlphaGO הוא AlphaGoZero. השינוי המרכזי שנעשה הוא* ***שהאימון התבצע באמצעות משחק עצמי בלבד****, ללא supervised learning. אף על פי ששיטה כזאת היא לרוב לא יציבה, אלגוריתם ה-MCTS מצליח להתגבר על כך. עוד הבדל מרכזי הוא שלא התבצע feature engineering, וה-input של האלגוריתם הוא הלוח בלבד ולוחות עבר. ספציפית, האלגוריתם מקבל את המצב הנוכחי ואת שבעת המצבים הקודמים, כולל את פעולות היריב (סה"כ 16 מצבים), עם לוח נוסף שמגדיר תורו של מי כעת (סה"כ 17 לוחות). בנוסף, נעשה שימוש ברשת אחת בלבד, וחיפוש פשוט יותר בעץ, ללא MC rollout.*

*שלושה שלבים באימון שמתבצעים במקביל:*

* *Self-play – על מנת ליצור data. בשלב זה, הרשת הטובה ביותר שאומנה עד כה משחקת 25,000 משחקים נגד עצמה. הנתונים שנשמרים: מצב הלוח, הסתברויות החיפוש שמתקבלות מ-MCTS, ותוצאת המשחק (ניצחון או הפסד).*
* *אופטימיזציה של הרשת. נדגמים 2,048 מצבים מתוך 50,000 המשחקים האחרונים. ה-input לרשת הוא המצבי האחרונים (כפי שתואר לעיל), וה-loss function מורכב מ-policy loss אשר מעדכן את ה-policy ומ-value loss אשר מעדכן את החיזוי של ה-value של המצב הנוכחי.*
* *אבלואציה של הרשת – מתבצעים 400 משחקים של הרשת המעודכנת עם הרשת הקודמת. אם הרשת החדשה ניצחה יותר מ-55% מהפעמים – היא מוכתרת למנצחת החדשה. במהלך המשחק, כל סוכן מתבסס על הרשת שלו ומבצע MCTS כדי לבחור את המהלך שלו.*

*הארכיטקטורה של הרשת מורכבת מ-40 residual blocks עם שני ראשים – אחד ל-policy ואחד ל-value. MCTS משמש גם ל-policy evaluation וגם ל-policy improvement. כל מצב (קודקוד) מכיל את ה-value של המצב, וכל פעולה (קשת) מכילה את ההסתברות להגיע אליה , את מספר הפעמים שביקרו בה ואת ה-action value שלה .*

*בחירת העלה מתבצעת על ידי הכלל הבא:*

*כאשר אחראי על האקספלורציה (הוא ה- upper confidence bound):*

*הוא היפר-פרמטר אשר שולט באיזון בין ה-exploration ל-exploitation, הוא ה-output של ה-policy network (ההסתברות לבצע פעולה במצב ), והוא אחראי לבחור פעולות עם פוטנציאל גבוה מתוך הפעולות שפחות נבחרו. הביטוי האחרון הוא זה שאחראי לאקספלורציה במצבים שבהם ממוצע הביקורים נמוך יותר, כאשר ה-1 במכנה אחראי לכך שהביטוי לא יגדל יותר מידי.*

*לאחר הבחירה בעלה, מאתחלים את כל הפעולות האפשריות (מאתחלים את כל ה-counters ל-0), בוחרים פעולה אחת, ומעריכים אותה באמצעות . ב-AlphaGoZero לא מבצעים אבלואציה באמצעות rollout policy, אלא עוצרים בשלב זה.*

*בשלב הבא – backup – מעדכנים את העץ:*

*לאחר 1,600 סימולציות, מתבצע השלב הבא – בחירת פעולה. בעוד שב-training בחירת הפעולה מתבצעת באמצעות , ב-test time נבחרת הפעולה עם ה-N הגבוה ביותר. לאחר שנבחרה פעולה, המצב החדש הופך ל-root, כל המצבים שאינם בנים 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 . כל המצבים נשמרים ונדגם אחד בצורה אקראית (יוניפורמית) ובו נעשה שימוש לעדכון הרשת. כאמור, ה-loss מורכב מ-MSE על ה-value, ומ-cross entropy על הפעולות:*

*כאשר הרשת מנצחת יותר מ-55% מהמשחקים – היא מוכתרת לטובה ביותר.*

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

* *לא נעשה שימוש ב-rollout*
* *שימוש ברשת יחידה*
* *כאשר מגיעים לעלה בעץ – פותחים את כל הפעולות, ומעריכים את המצבים החדשים באמצעות הרשת (ולא סימולציה)*
* *אין Tree policy*
* *Self-play – ללא data ממשחקים אמיתיים*
* *אין hand crafted features אלא רק מצבי הלוחות האחרונים*

## **AlphaGO בתחומים אחרים**

*לאחר שפורסם AlphaGO נעשו ניסיונות להעביר את הרעיונות שבאלגוריתם לתחומים אחרים. אחד האלגוריתמים הוא AlphaD3M – יצירת pipeline למודלי ML (AutoML). בעבודה זאת, ה-pipeline בכללותו ממודל כמצב, אשר מכיל מידע על ה-dataset, על המטלה ועל ה-pipeline. כל פעולה היא שינוי של מודול ב-pipeline.*

*ההבדל מ-AlphaGO הוא שזה משחק עם שחקן אחד, אך חוץ מהעובדה הזו – התהליך מאוד דומה. הרשת חוזה פעולה ו-value (כמו ב-AlphaGoZero):*

*מבצעים UCT לאקספלורציה:*

*וישנו שינוי קטן בפונקציית ה-loss:*

*השינוי הוא הוספת הרגולריזציה על אורך ה-pipeline באמצעות .*

*שימוש אחר ב-framework הזה נעשה לטובת architecture search. מבנה של רשת יכול להיכתב בצורה הבאה: ["filter width:5", "filter height:3", "num filters:24"…], ולכן controller יכול לבחור באיזו צורה לבנות ארכיטקטורה. בעבודה נעשה שימוש ב-RNN שייצור 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 שה-accuracy לא גזיר, נעשה שימוש באלגוריתם REINFORCE:*

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

*עוד עבודה קרובה היא Deepline. לפירוט נא לעיין בהמשך בפרק על מרחבי מצבים ופעולות גדולים.*

# **Meta-learning**

*Meta-learning הוא תחום ב-ML בו מנסים לבצע למידה של רשת אחת או אלגוריתם אחד על מספר מטלות (tasks). במקרים רבים נדרוש מהאלגוריתם להתאים את עצמו בצורה מהירה למטלות חדשות (few/one/zero shot learning). נראה בהמשך שתחום זה משיק מאוד ל-RL מכיוון שבצורה טיפוסית ב-RL ישנם מקרים של מטלות רבות שסוכן יכול ללמוד מהם במקביל (למשל מספר משחקי מחשב, או רובוט עם משימות מעט שונות), ועם זאת איסוף ה-data יכול להיות מאתגר. בנוסף, מערכות RL הפועלות בעולם האמיתי נפגשים במצבים של שינוי בהתפלגות (distribution shift), אשר ניתן להתמודד איתו באמצעות שיטות מ-meta learning.*

*מונחים:*

* רשימת tasks.
* כל task מאופיין ב-, הסתברויות מעבר בין מצבים ואורך של episode - .
* ממדל את ההסתברויות לפעולה בהינתן המצבים הקודמים.

ב-meta learning, המטרה היא למצוא את האופטימום:

במילים פשוטות, המטרה היא למצוא מודל שמבצע היטב מספר מטלות. קיימים מצבים רבים בהם קשה לאסוף data למשימות רבות, ולכן נהיה מעוניינים ברשת אשר יכולה ללמוד היטב ממספר מועט של דוגמאות. במצב שיש לנו משימות מרובות נרצה למצוא רשת אשר אומנה על data רב יחסית שמורכב ממספר משימות, כך שלאחר שיש מודל ראשוני, הוא יוכל בקלות ללמוד משימה חדשה בעזרת data מועט.

## **Meta-Learning with Memory-Augmented Networks**

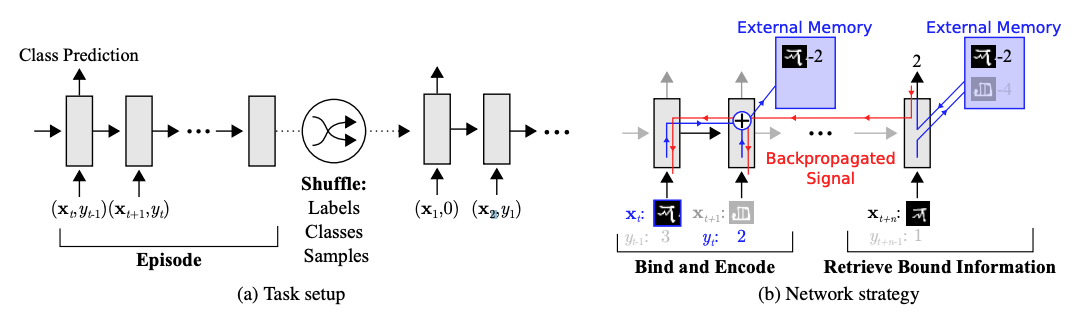
מטרת שיטה זו היא לבצע few shot learning באמצעות זיכרון חיצוני לרשת, אשר הרשת יכולה לגשת אליו בזמן הפרדיקציה. הזיכרון מורכב ממטריצה גדולה, אשר כל שורה בה היא וקטור מאפיינים (feature vector). שורה יכולה להיות מאפיינים של דוגמה ספציפית מתוך סט האימון או קומבינציה של מאפיינים ממספר דוגמאות (כפי שיוסבר בהמשך). מטרת הארכיטקטורה היא לשמור במטריצת הזיכרון מאפיינים של דוגמאות מייצגות ביחד עם אינפורציה על התיוג שלהם. בזמן ה-inference, הרשת תדע לגשת למאפיינים דומים מתוך מטריצת הזיכרון ולדעת לחזות את ה-label שלהם באמצעות האינפורמציה שהרשת למדה מתוך אוסף ה-tasks שנלמדו בזמן האימון.

המערכת מורכבת מהחלקים הבאים:

* Controller (רשת FC או RNN) – מקבל את הדגימה בכניסה, מוציא את הפרדיקציה, ובאינטרקציה 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 מהצורה הבאה. מתוך קבוצת הדגימות אשר נלקחו ממספר tasks, מייצרים רצף של דגימות מהצורה הבאה:

הרשת מקבלת רצף של זוגות המורכבים **גם מהדגימה וגם מהתיוג של הדגימה שלפניה**. בסוף הרצף מקבלים דגימה והרשת צריכה להוציא תיוג שלה. הנקודה היא שהמוצא לא מחובר ישירות לדגימה שהתקבלה בכניסה, אלא דרך הזיכרון. תהליך ה-backpropagation מאלץ את הרשת שראתה בתחילת הרצף דוגמאות רלוונטיות ואת התיוג שלהם – למצוא דרך לקודד את הדגימות והתיוג שלהם בזיכרון, כך שבסוף הרשת תצליח לקשר את הדגימה החדשה לדגימות דומות מהעבר ולתיוג שלהם. להלן תרשים של השיטה:



### **גישה לזיכרון**

הגישה לזיכרון מתבצעת על ידי קבלה של וקטור כניסה והחזרה של וקטור שנקרא מהזיכרון. הגישה מתבצעת בצורה של attention. מתוך ה-controller מחשב וקטור מאפיינים , שממדיו כממדי וקטורי המאפייני שבמטריצת הזיכרון. לאחר מכן, מחושב ה-cosine similarity בין לכל אחת מהשורות במטריצת הזיכרון:

בשלב זה, מחושב softmax על כל איברי :

ערכי הוקטור החדש יהיו גדולים יותר עבור שורות הדומות לוקטור וקטנים עבור שורות השונות ממנו. לאחר מכן, מבצעים ממוצע משוקלל של כל שורות מטריצת הזיכרון באמצעות ערכי :

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

### **כתיבה לזיכרון**

בכתיבה לזיכרון, משנים את השורות במטריצת הזיכרון **שלא נעשה בהם שימוש מזמן**. הרעיון הוא ששורות שלא כתבנו אליהם ושלא קראנו מהם מזמן, כנראה לא נושאות אינפורמציה וניתן להחליף אותן בשורות אינפורמטיביות יותר. לצורך זה, מחשבים את וקטור (הממד שלו כמספר השורות במטריצת הזיכרון) אשר מכמת כמה שימוש נעשה בכל שורה במטריצת הזיכרון. הוקטור מחושב בצורה איטרטיבית בצורה הבאה:

כאשר הוא סוג של discount factor אשר מאפס בצורה הדרגתית את הוקטור. הוא הוקטור שהצגנו לעיל אשר מכיל אינפורמציה כמה אחוז היוותה כל שורה במטריצה מהוקטור שנקרא. בעל תפקיד דומה במשימת הכתיבה.

הוא וקטור בינרי (מכיל ), גם הוא בגודל של מספר השורות במטריצת הזיכרון, אשר מכיל 1 ב-n הערכים שבהם הוקטור הכי קטן. או במילים אחרות, n השורות שהכי פחות בשימוש בזמן האחרון:

הוא האבר ה-n-י הכי קטן ב-. בשלב זה ניתן לחשב את – וקטור הכתיבה (גם בממד של מספר השורות של המטריצה):

*הוא פרמטר נלמד השולט כמה אנחנו נותנים חשיבות לוקטור הכתיבה הקודם, וכמה נותנים לוקטור שחישבנו עכשיו. בסופו של דבר מתקבל וקטור אשר שולט באיזה שורות נעדכן בזיכרון ואילו שורות לא. בשלב האחרון מתבצע העדכון עצמו:*

*אשר מוסיף את וקטור (בצורה ממושקלת) לשורות הרלוונטיות במטריצת הזיכרון.*

### **פרדיקציה**

*בזמן ה-inference, מתבצעים השלבים הבאים:*

1. *ה-controller מקבל דגימה וממפה אותה (באמצעות רשת) ל-*
2. *נשלח ל-read heads ומחושבים המשקלים לקריאה*
3. *בעזרת המשקלים הללו, מחושב הממוצע המשוקלל של שורות הזיכרון*
4. *וקטור זה נשלח על ידי ה-controller לרשת קלסיפיקציה אשר מספקת פרדיקציה*

*האימפלמנטציה המדויקת יכולה להשתנות מעט. למשל, ה-controller יכול להיות רשת FC או RNN, מה שישנה מעט את התהליך.*

### **ניסויים**

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



*ה-dataset מורכב מ-1643 classes שונים, מתוכם 1200 בסט האימון ו-443 ב-test set. באימון יצרו 100,000 רצפים של 50 דוגמאות שנדגמו מתוך חמישה classes – עשרה דוגמאות מכל class. הביצועים של האלגוריתם עלו על ביצועים אנושיים:*

Table

Description automatically generated with medium confidence

## **Simple Neural Attentive Meta-Learner (SNAIL)**

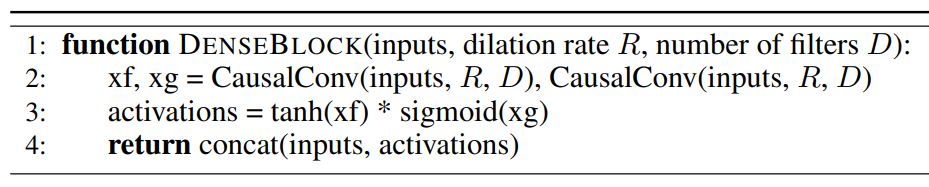
*החיסרון העיקרי בשיטה הקודמת הוא שהארכיטקטורה קבועה מראש. למשל, מטריצת הזיכרון מוגדרת מראש ולא יכולה לגדול. לכן, במקרה של רצף ארוך מאוד, קיים סיכון שהרשת "תשכח" את מה שהיא למדה בעבר הרחוק, מכיוון שמשאבי הזיכרון שלה מוגבלים. השיטה שנציג עכשיו, Simple Neural Attentive Meta-Learner מתגברת על החיסרון הזה באמצעות שני מנגנונים משלימים:*

* *Dilated convolution – קונבולוציה המתבצעת בקפיצות. למשל, עבור kernel של , הקונבולוציה לא חייבת 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 אחד מהכניסות ליציאה הספציפית, ומחושב ממוצע משוקלל של כל הכניסות על פי מידת הרלוונטיות שלהם.*

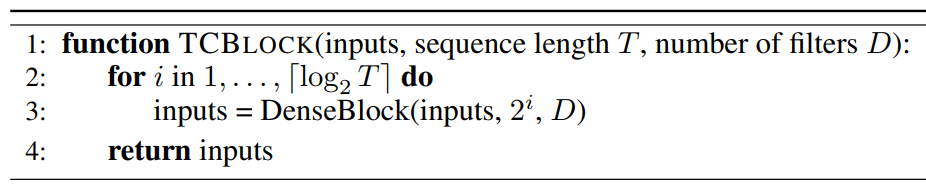
### **תיאור השיטה**

*הארכיטקטורה מורכבת ממספר בלוקים:*

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

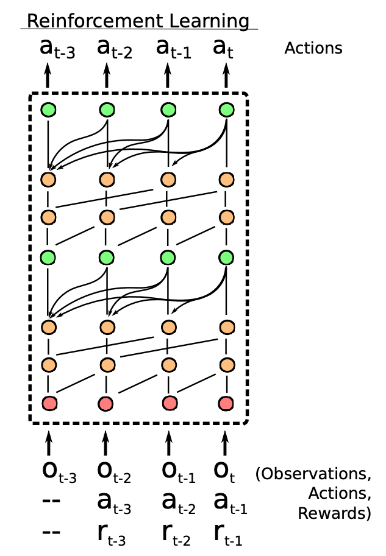
**

*Temporal Convolutions Layer (TCL): מורכב ממספר dense blocks. משרשרים dense blocks אחד אחרי השני, כאשר בכל פעם מגדילים את ה-dilation rate (R) פי 2. התהליך מתבצע עד שגודל ה-dilation rate הוא כאורך ה-input T. זה מאפשר לבלוק האחרון לקבל receptive field של כל אורך הרצף. להלן האלגוריתם:*

**

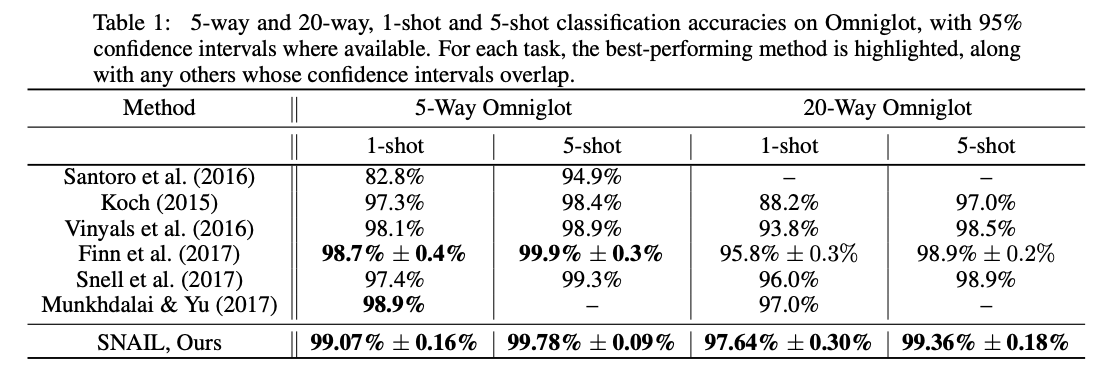
*Attention Mechanism: מבצע מנגנון attention על ה-outputs של שכבת ה-TCL הקודמת. בשיטה זו מקבלים ערך אחד (או וקטור אחד) – לא משנה מה אורך הרצף T. וקטור זה יכול "להתחשב" גם בהיסטוריה רחוקה, מכיוון של "שכחנו" את ההיסטוריה, אלא ממשקלים אותה בכל פעם מחדש. מהמוצא של השכבה אפשר לבצע פרדיקציה לפעולה (action)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

*להלן תרשים של השיטה:*

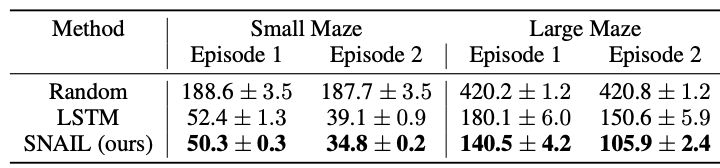
**

### **ניסויים**

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

**

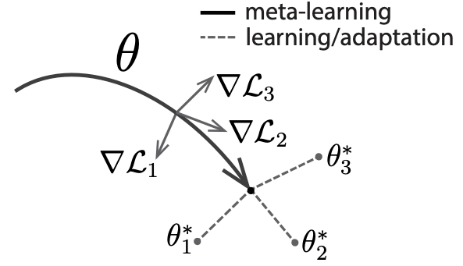
*הניסוי השני התבצע במשחק מחשב המדמה הליכה במבוך. האימון התבצע על מבוכים קטנים, וה-test על קטנים וגדולים. בזמן ה-test מראים לסוכן פעם אחת את המבוך על מנת שילמד אותו (one shot), ואז מתחילים מחדש באותו המבוך ומבצעים את האבלואציה. להלן הביצועים:*

**

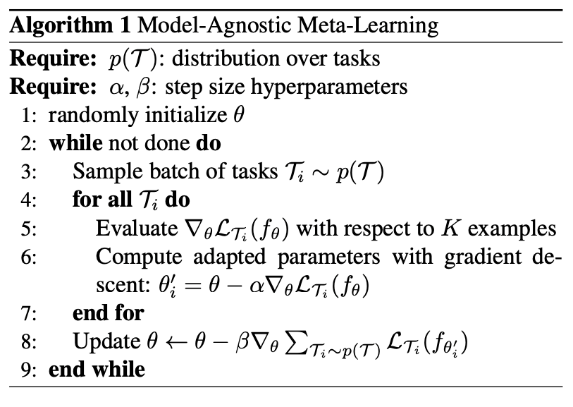
## **Model Agnostic Meta-Learning (MAML)**

*שיטה זו ניגשת לבעיית ה-Meta Learning מכיוון שונה מעט. הרעיון הוא שביצוע fine tuning על מספר קטן של דוגמאות (few shot) עלול ליצור overfitting. לכן, מטרת השיטה היא למצוא סט של משקלים לרשת, אשר יהיה קל לבצע עליו fine tuning עבור מספר גדול של משימות. במילים אחרות, למצוא ספציפי שיהיה קרוב כמה שיותר ל- - סט המשקולות האופטימלי למטלות , כך שיהיה קל להגיע מ- ל-:*

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

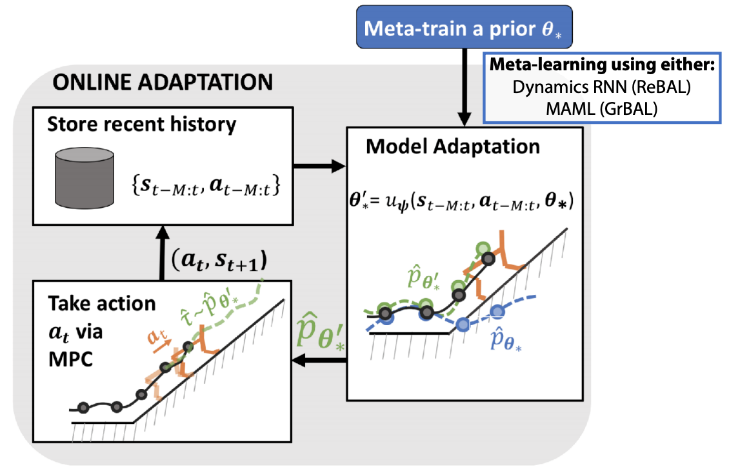
**

*במהלך ה-training אנו מנסים להגיע במרחב המשקולות למקום שיהיה "קרוב" ל-. בכל איטרציה במהלך האימון נדגמים מספר מטלות , מכל מטלה נדגמות K דגימות ומתבצע עדכון למשקולות באמצעות gradient descent כך שמתקבל . בשלב האחרון, מתבצעת גזירה של ה-loss על פי הפרדיקציות של הרשתות* ***עם המשקולות המעודכנים******לפי המשקולות המקוריים*** *. זוהי נקודה מעט עדינה. מכיוון שאנו רוצים לחשב כיצד יש לשנות את המשקולות המקוריים, כך שלאחר גזירה על data ממטלה i ה-loss על אותה המטלה יהיה נמוך, לכן אנו גוזרים את ה-loss שמתקבל מהרשת על המשקולות המעודכנים לפי המשקולות המקוריים. יש לשים לב כי המשקולות המעודכנים מכילים נגזרת של המשקולות המקוריים. לכן, על מנת לבצע את שלב העדכון ישנה* ***נגזרת שניה****. פעולה זו יכולה לאתגר את היציבות של תהליך האימון. להלן תאור האלגוריתם:*

**

## **Adaption in Real-Time**

*שימוש באלגוריתמי DRL בעולם האמיתי הוא מאתגר ממספר סיבות. איסוף data בסביבת עולם אמיתי הוא יקר, קיים הרבה רעש בסביבות "אמיתיות", וקיימים מקרים רבים של distribution shift (פתאום ירד גשם והמנועים של הרובוט מגיבים לאט יותר). הרעיון בעבודה זו שנציג 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 ללמידת RL בעולם האמיתי נקראת GrBAL. להלן סכמה של השיטה:*

**

*להלן סיכום של היתרונות והחסרונות של השיטות שנסקרו:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***קריטריון*** | ***הסבר*** | ***SNAIL*** | ***MAML*** | ***ReBAL*** | ***הערות*** |
| *Consistent* | *אם נוסיף data נתכנס לאופטימום* | *X* | *V* | *V* |  |
| *Expressive* | *האם יש מספיק כוח ייצוג לרשת (לאלגוריתמים שונים)* | *V* | *X* | *~* | *MAML יתקשה בRL למשל כאשר ה-reward מגיע רק בסוף* |
| *Structured Exploration* | *חיפוש יעיל במרחב הבעיה* | *~* | *~* | *X* |  |
| *Efficient & off-policy* | *האם מתאים לבעיות עולם אמיתי* | *X* | *X* | *V* |  |

# **Transfer Learning**

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

*נרחיב על מספר סוגי אלגוריתמים:*

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

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

## **Fine-tuning**

*הצורה הפשוטה ביותר של fine-tuning היא שימוש ברשת שאומנה על מטלה אחת למטלה אחרת. ניתן לבצע אימון על כל משקולות הרשת או להקפיא את השכבות הראשונות ולאמן רק את האחרונה (או האחרונות). צורה יותר מתקדמת היא לאמן את הרשת את האימון הראשוני כך שתהיה כללית ומגוונת יותר, כלומר תנסה לפתור את הבעיה במספר רב ומגוון של דרכים. אימון כזה גורם לרשת להיות כללית יותר ולהתאים למגוון רב יותר של בעיות. להלן נסקור דרכים לבצע אימון כזה (אימון מגוון).*

*מאמר בשם Reinforcement Learning with Deep Energy-Based Policies מתמודד עם בעיה זו. הגישה היא למידה של פוליסה סטוכסטית, כך שתתבצע אקספלורציה ויכולת למידה של מספר גישות לפתרון. במקום ללמוד שיטה אחת שפותרת את הבעיה בצורה הטובה ביותר, בשיטה זו מנסים למצוא כמה שיותר פתרונות לבעיה. רשת שלמדה בצורה כזאת יכולה להיות בסיס טוב ל-transfer learning.*

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

*פונקציית האנרגיה שפותחה במאמר באה למקסם בנוסף את האנטרופיה:*

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

*וכן ה-value function:*

*להלן האלגוריתם המלא:*

*Text

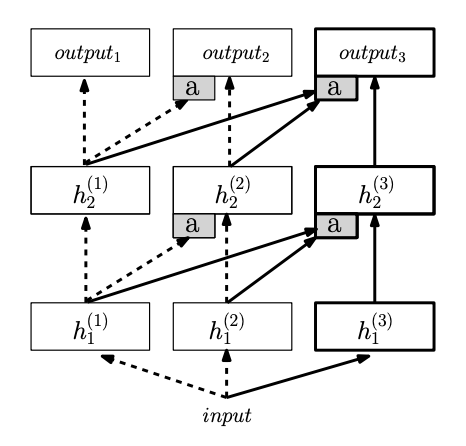
Description automatically generated*

## **Progressive Networks**

*חסרונות של שיטת ה-fine tuning הם שאימון מחדש של כל הרשת על dataset קטן יכול לגרום ל-overfitting. בנוסף, אימון מחדש גורם לרשת לשכוח את ה-task המקורי, דבר שיכול להיות בעייתי ב-use-cases מסוימים. הרעיון ב-progressive networks הוא לתכנן ארכיטקטורה שתתאים במיוחד ל-transfer learning. דבר זה מתבצע בצורה הבאה. בשלב הראשון מאמנים רשת סטנדרטית על משימה ראשונה. כאשר מעוניינים בהוספה של משימה חדשה, מקפיאים את הרשת הראשונה, ומוסיפים רשת שניה (יכולה להיות זהה לראשונה) אשר מקבלת בכל שכבה את המאפיינים של השכבה המתאימה ברשת הקודמת, בנוסף למאפיינים החדשים שנלמדים. בצורה כזו ניתן להרוויח הן מהמאפיינים שנלמדו במטלה הקודמת אשר יכולים להיות רלוונטיים למשימה הבאה, והן מאימון של פרמטרים חדשים ייעודיים למשימה.*

*ספציפית, כל שכבה מבצעת את החישוב הבא:*

*כאשר הביטוי הראשון הוא שכבה לינארית סטנדרטית ( הוא היציאה מהשכבה הקודמת ו- היא מטריצת המשקלים המתאימה), והביטוי השני הוא קומבינציה של היציאות מהשכבות המקבילות של רשתות שלמדו מטלות אחרות. ספציפית בשיטה זו, לכל וקטור יציאה של שכבה i ממטלות קודמות קיימת מטריצת משקולות בעלת ממדים זהים, וכל התוצאות של המכפלות של המטריצות הללו עם וקטורי המאפיינים נסכמות ביחד עם אלה של השכבה הנוכחית ומועברות בפונקציית האקטיבציה לשכבה הבאה. להלן סכמה של הארכיטקטורה:*

**

*במאמר המקורי נעשה שימוש ב-adapter layer, מימוש מעט מורכב יותר שנועד לתת משקל נכון לכל אחד מווקטורי המאפיינים של המטלות הקודמות. במימוש זה, ווקטורי המאפיינים מוכפלים ב- (סקלר – פרמטר נלמד) ומועברים למרחב קטן (באמצעות מטריצה נלמדת) ועוברים אקטיבציה (סיגמואיד). תוצאת האקטיבציה מועברת בשכבה לינארית ומתווספת לשכבה הלינארית של המאפיינים של המטלה הזו:*

*יתרונות השיטה:*

* *יכולת ללמוד K מטלות שונות*
* *למידה מהירה יותר בעזרת מאפיינים שנלמדו במטלות קודמות (transfer learning)*
* *אין שכחה של המשקולות (catastrophic forgetting)*

*חסרונות השיטה:*

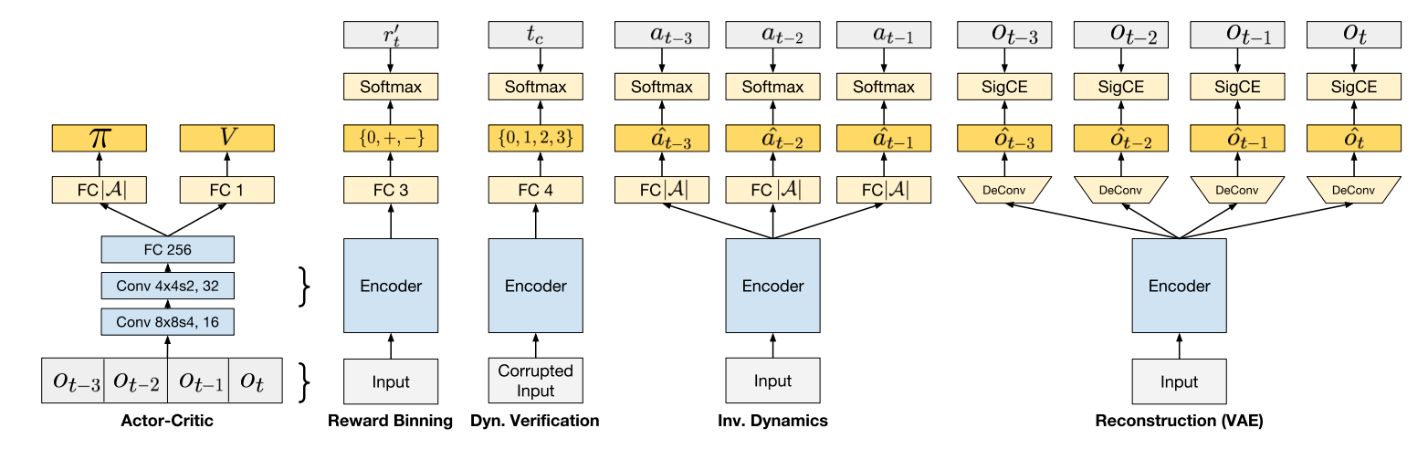
* *גודל הרשת ומספר המשקולות עולה משמעותית עם כל הוספה של מטלה*
* *לא ניתן לדעת בצורה ישירה איזה מתוך המטלות הקודמות אכן רלוונטית למטלה הבאה*

## **Self-Supervision for RL**

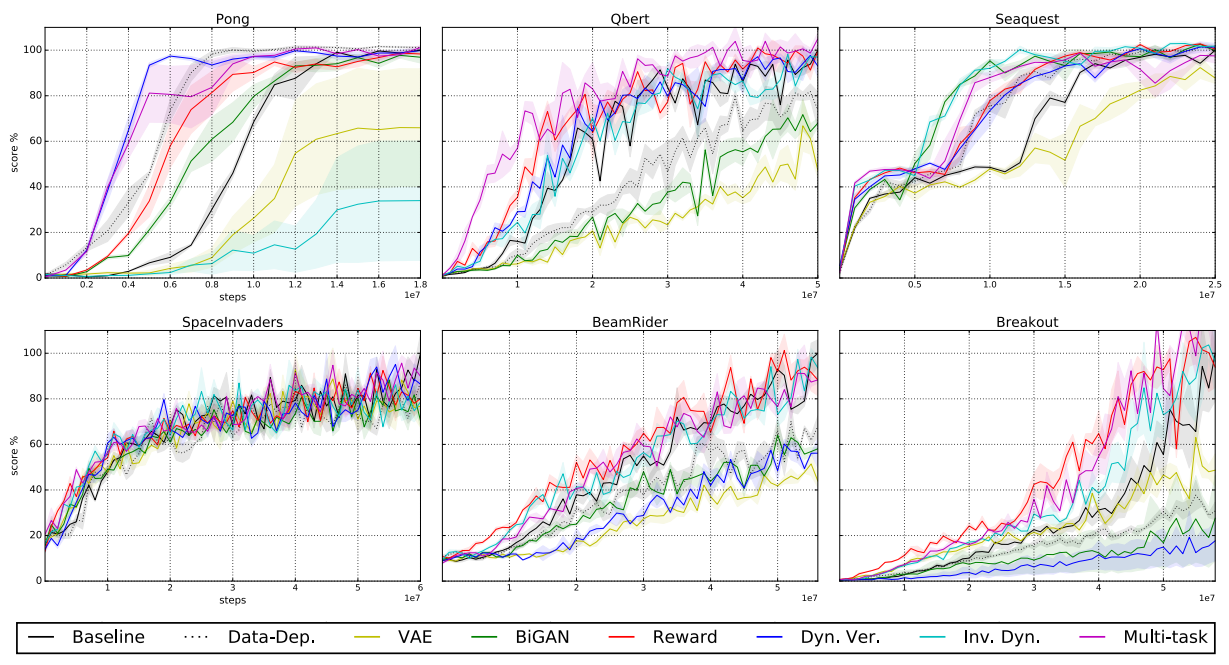
*תחום ה-SSL וה-representation learning יכול להיות מאוד רלוונטי ללמידה יעילה ב-RL. למשל שימוש בשכבה לפני אחרונה של רשת policy שאומנה בסביבה מסוימת יכולה לייצג בצורה טובה את הסביבה שאותה הרשת למדה. נציג את המאמר Loss is its own reward: Self-supervision for reinforcement learning. נטען במאמר שלמידה של rewards היא קשה מכיוון שהם יכולים להיות מאוד דלילים (sparse) בזמן. לעומת זאת, ניתן למצוא auxiliary losses אשר* ***ילמדו את הסביבה*** *ויהיו רלוונטיים לחיזוי ה-rewards. להלן רשימה של auxiliary tasks שנעשה בהם שימוש במאמר:*

* *Reward binning – בנוסף לחיזוי ה-reward, הרשת הייתה צריכה לחזות האם הוא חיובי או שלילי*
* *Dynamics* – חיזוי בהינתן
* *Inverse Dynamics – חיזוי בהינתן*
* *Corrupt Dynamics – החלפת מצב במצב קרוב – מהעבר או מהעתיד (הרשת צריכה לחזות את אותו חיזוי למרות שינויים קטנים במצב)*
* Reconstruction – שימוש ב-variational auto encoder לשחזור אות הכניסה

להלן סכמה של השיטה:



הרשת אומנה על מטלה אחת, ואז, בעזרת ה-encoder בלבד, אומנה על מטלה חדשה. הראו במאמר שהביצועים במטלה השניה טובים יותר כאשר מאמנים את כל הרשת מאשר כאשר מקפיאים את ה-encoder. להלן ביצועים על מטלות שונות:



כעת נעבור 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 קרובה לסוכנים 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 הרשתות:

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

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

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

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

## **Distillation for Multi-Task Transfer**

Distillation היא שיטה להעברת מידע ממודל אחד (בצורה טיפוסית מודל גדול) למודל שני (קטן יותר). ניתן להשתמש בשיטה זו גם כאשר המודל הוא ensemble של מודלים, ואין אפשרות להריץ הכל ב-production. בשיטה זו לוקחים מודל קטן ומאמנים אותו לחכות את וקטור ההסתברויות (softmax) שהמודל הגדול מוציא.

בשיטה זו מאמנים את המודלים ב-softmax עם טמפרטורה:

במהלך האימון נבחר T גבוה, אשר גורם ל-logits להיות גבוהים יותר. אינטואיטיבית, זה גורם "לחזק את הסיגנל" שעובר ברשת. לאחר האימון, T של הרשת שאותה למדו מוגדר ל-1. בפרדיקציה מתבצע ממוצע משוקלל בין שני הערכים, עם הטמפרטורה ובלעדיה (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, נוכל בקלות לבצע התאמה בין רובוט למטלה, גם אם ספציפיות אותו הרובוט לא אומן על המטלה הזאת. את הפוליסה ניתן למדל בצורה הבאה:

כאשר היא פונקציה שאחראית על המטלה, ו- אחראית על הרובוט (היא זו שמחזירה את הפוליסה של הרובוט). הבעיה חייבת להיות ממודלת בצורה הזאת, מכיוון שלכל רובוט צריך להגדיר פוליסה שונה (למשל מספר המנועים יכול להיות שונה) ולכן הפונקציה החיצוני חייבת להיות זו שממדלת את הרובוט. אם הלמידה התרחשה כמו שצריך, ניתן יהיה להחליף רק את למטלה אחרת והרובוט ידע להתמודד איתה, או להחליף את הרובוט לרובוט אחר, אשר ידע לפתור את המטלה. להלן סכמה של השיטה:

Diagram

Description automatically generated

הארכיטקטורה שבה השתמשו במאמר:

Diagram

Description automatically generated

# **מרחבי מצבים ופעולות גדולים**

בעיות עולם אמיתי מתאפיינות לעיתים במרחבי פעולה גדולים – הרבה פעולות בדידות שהסוכן צריך לבחור מתוכן, ומרחבי מצבים גדולים. מקרים כאלה יכולים להיות מאתגרים, למשל קשה לבצע אקספלורציה יעילה במרחב מצבים גדול. בפרק זה נסקור שיטות להתמודדות עם הבעיה.

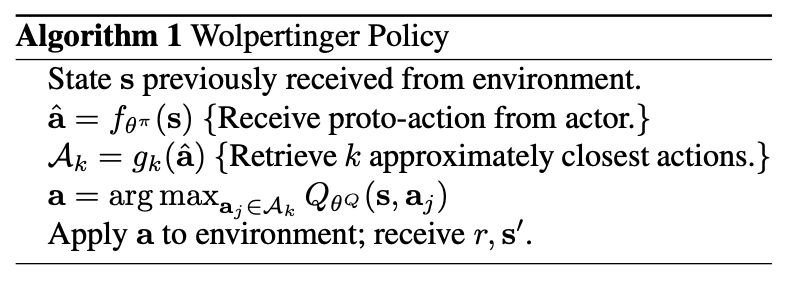
## **מרחב פעולות רציף**

פיתרון אפשרי לבעיה הוא להמיר את מרחב הפעולות הבדיד למרחב embedding רציף. בצורה כזאת אפשר לדחוס את הפעולות בצורה יעילה יותר, ובנוסף לקבל מטריקה של קרבה בין פעולות (מרחק אוקלידי/cosine similarity וכו'). ניתן למשל להשתמש במודל גנרטיבי ליצירה של וקטור במרחב הפעולות ולבחור פעולות קרובות לאבלואציה, למשל על ידי KNN. עוד יתרון שיש בשיטה הזו, שבשונה מהשיטות המסורתיות שראינו בעבר, אין צימוד הכרחי בין הפוליסה לאבלואציה של הפעולות. למשל ב-Q-learning הפוליסה נקבעת ישירות מה-Q-function. עם זאת, בשיטה שהצגנו כעת, ה-policy בלתי תלויה ב-Q-function, שכן היא אחראית רק על הצעה של וקטור במרחב הרציף, והיא אינה תלויה בפונקציית האבלואציה. דבר זה יכול לאפשר גמישות בתכנון האלגוריתם.

האלגוריתם מורכב מהשלבים הבאים:

1. Generation של וקטור על ידי ה-actor. הוקטור נקרא proto-action מכיוון שהוא אינו פעולה בעצמו, אבל הוא במרחב הפעולות, והוא מייצג פעולה רצויה
2. בחירה של K פעולות קרובות ביותר באמצעות KNN ()
3. ה-critic נותן ציון לכל אחת מפעולות אלה, ונבחרת הפעולה עם ה-value הגבוה ביותר

להלן תיאור האלגוריתם:



ולהלן סכמה של השיטה:

Diagram

Description automatically generated with medium confidence

## **Action Elimination with Deep Reinforcement Learning**

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

* Reward shaping – שינוי ה-reward המתקבל מפעולות מסוימות (פעולות שאנחנו יודעים שלא יעילות). נשנה את ה-reward עבור פעולות שאנחנו יודעים שהן "לא נכונות" כך שיהיה נמוך יותר. בשיטה זו בדרך כלל נעשה שימוש במומחה שיגדיר אלו פעולות פחות מבטיחות. שיטה זו בעייתית מבחינת הקושי לבצע tuning ל-shaping והיא לא יעילה מבחינת דגימות, מכיוון שנדרשת "אקספלורציה" למציאת הפעולות הבעייתיות.
* Two headed (interleaved) policy – שימוש בשני ראשים ב-policy network – אחד שחוזה את הפעולה שתמקסם את ה-reward (כפי שמתבצע בדרך כלל), ובנוסף, ראש שחוזה פעולות שמביאים למינימום את ה-action elimination error, כלומר שיוצר מצב שבו לא נבחרות הפעולות שאנו ידעים שלא יעילות לסוכן. את הפעולות האלה אפשר להגדיר על ידי מומחה, או על ידי אקספלורציה של הסוכן עצמו. חיסרון בשיטה זו ששתי המטרות מאוד מצומדות (coupled) אחת לשניה, מכיוון ששתיהן 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 מאתגרת ולא יציבה.

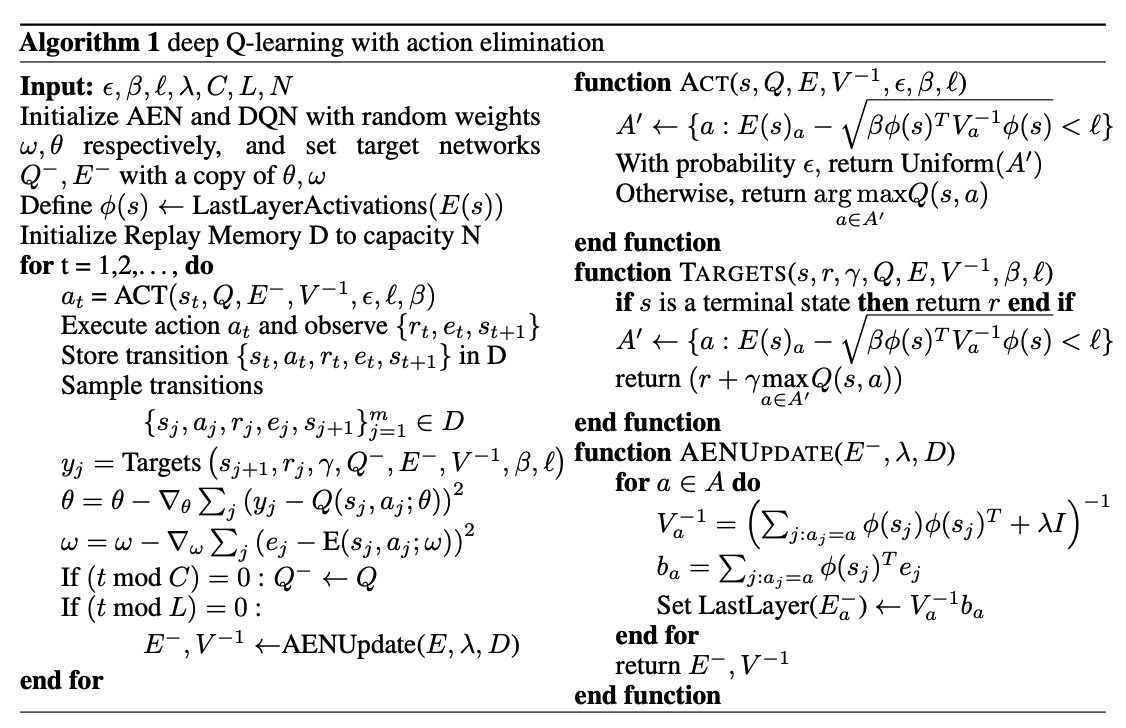
במאמר, הכותבים מציעים שיטה להפרדת הצימוד בין ה-policy network לבין רשת ה-action elimination. כל אחד מהרכיבים מקבל מטרה (objective) אחרת, מה שמשפר את ההתכנסות. ה-policy network היא רשת סטנדרטית (DQN) אשר חוזה את הפעולה האופטימלית. רכיב ה-action elimination ממדל את הבעיה כ-contextual linear bandit – באמצעות וקטור מאפיינים המייצג את המצב (contextual) נדרש לבחור בפעולה אחת מתוך רבות (bandits). האתגרים בייצוג כזה הם שמרחב הפעולות גדול, ולכן מדובר ב-contextual bandits עם מספר גדול של bandits (מה שמקשה את הבעיה). בנוסף, על מנת ללמוד פתרון יציב, וקטור המאפיינים (context) צריך לא להשתנות עם הזמן, ולכן לא ניתן למשל להשתמש במאפיינים שה-policy network מחלצת. לכן, חלק זה מורכב מרשת אשר נקראת AEN (action elimination network) האחראית על חילוץ המאפיינים (הקונטקסט), ומאלגוריתם UCB לבחירה של הפעולה על בסיס אותם המאפיינים. בפתרון שלהם, רשת ה-AEN מחלצת וקטור מאפיינים, אשר משמש כ-input ל-contextual bandit. לפתרון ה-contextual bandit נעשה שימוש בUCB אשר נסקר בפרק על multi arm bandit.

מתמטית, מניחים שקיים כך ש- כאשר היא רשת המחלצת מאפיינים מהמצב , ו- הוא וקטור עבור כל מצב, כך שהמכפלה הסקלרית שלו עם המאפיינים שחילצנו תיתן סקלר – elimination signal. האתגר בגישה זו הוא שבמהלך אימון וקטור המאפיינים משתנה, ולכן ה-contextual bandit לא יכול להילמד בצורה טובה. הפיתרון הוא שהרשת תעודכן פעם אחת כל מספר צעדים, ולאחר כל עדכון ה-contextual bandit יתאמן מחדש.

האלגוריתם מורכב משתי רשתות – – רשת ה-policy ורשת ה-AEN. האלגוריתם משתמש ב- על מנת ליצור את וקטור המאפיינים ל-contextual bandit לינארי. הרשת מתעדכנת בכל L איטרציות (**AENUpdate** באלגוריתם), כאשר בכל איטרציה המודל הלינארי מתעדכן מחדש. וקטור המאפיינים (**LastLayerActivations** באלגוריתם) מועבר ל-contextual linear bandit model:

*לאחר מכן פותרים את המודל (****AENUpdate****) ומעדכנים את ה-AEN: . בשלב הזה משתמשים ב-contextual linear bandit model כדי לשלול פעולות שקיבלו הסתברות גבוהה (****ACT, Targets****). הפעולות שנשארו, שאותם ניתן לבצע הן:*

*מתוך אותן פעולות נבחר באמצעות ϵ-greedy. במקרה של exploitation נבחר את הפעולה עם ה-Q-value הגבוה ביותר (מתוך ), ובמקרה של exploration נבחר בפעולה אקראית מתוך . להלן האלגוריתם:*

**

להלן סכמה של התהליך:

Diagram

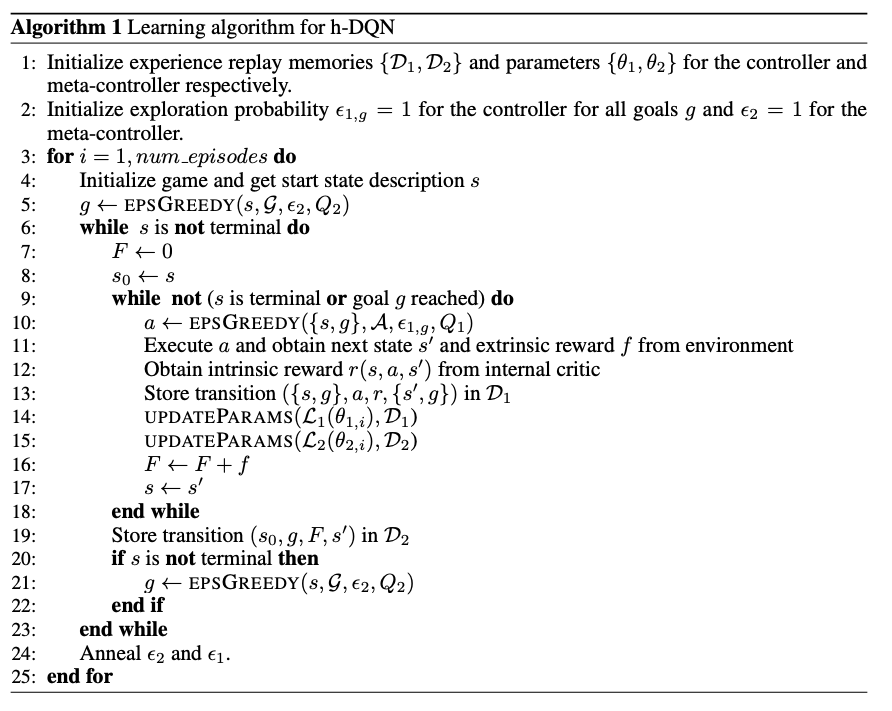
Description automatically generated

## **Hierarchical DRL for Sparse Reward Environments**

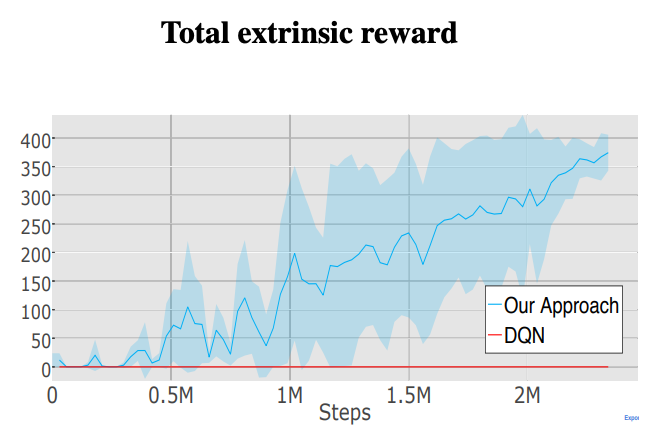
במאמר "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation" מתייחסים לבעיה של RL במרחב מצבים גדול ו-rewards דלילים (sparse). הגישה של האלגוריתם היא שימוש ב-"intrinsic goals" – תת-מטרות אשר עוזרות לסוכן לבצע אקספלורציה. תת מטרה כזאת יכולה להיות למשל ללכת למספר מקומות רחב במשחק מחשב, או להרים כמה שיותר חפצים ולנסות להשתמש בהם. למשל במשחק "Montezuma's Revenge" ניתן להגדיר מטרות כגון: לאסוף כמה שיותר מפתחות ולפתוח כמה שיותר דלתות, לחקור כמה שיותר חדרים או להגיע לנקודות מסוימות במהלך המשחק.

בנוסף, נעשה שימוש בשני רכיבים שונים: ה-**controller** וה-**meta controller**. ה-controller שולט בסוכן ובפעולות שלו, ומקבל intrinsic goals מה-meta controller. מטרתו היא לגרום לסוכן לבצע את ה-intrinsic goals. הוא מבצע כל intrinsic goal במשך מספר מוגדר של צעדים ולבסוף מוגדר האם הוא הצליח או לא. הגודל שהוא מנסה למקסם הוא:  
ה-meta controller אחראי על ה-task הראשי, וכן אחראי להגדיר תתי משימות (intrinsic tasks) ל-controller. הגודל שהוא מנסה למקסם הוא:  
כאשר הוא ה-reward המתקבל מהסביבה. בנוסף, 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 ומחזיר internal reward לתת המשימה הספציפית. למשל, אם תת המשימה היא להגיע למפתח, הוא יהיה אחראי להחזיר reward ברגע שהסוכן הגיע למפתח, כך שה-controller יוכל ללמוד את תת המשימה.

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



במאמר הוצג שימוש באלגוריתם על מנת ללמוד את המשחק "Montezuma’s Revenge". להלן השוואה לאלגוריתם DQN:



ניתן לראות שגם אחרי 2.5M צעדים, אלגוריתם ה-DQN לא מצליח לפתור את השלב. תוצאה זו אינה מפתיעה, מכיוון שעל מנת לפתור את השלב נדרש לבצע מספר משימות בצורה טורית. אלגוריתם DQN מתבסס על ה-reward המתקבל מהסביבה, ובשביל להצליח הוא חייב להצליח לפחות באחוז קטן מהפעמים לקבל סיגנל חזרה. ולכן הוא לא מצליח ללמוד.

## **בעיות מהעולם האמיתי**

יצירה אוטומטית של pipeline עבור אלגוריתמי ML באמצעות RL, הוא דוגמה לבעיה עם מספר מצבים ופעולות גדולים. ניתן לסווג את אלגוריתמי ה-pipeline האוטומטי לשתים:

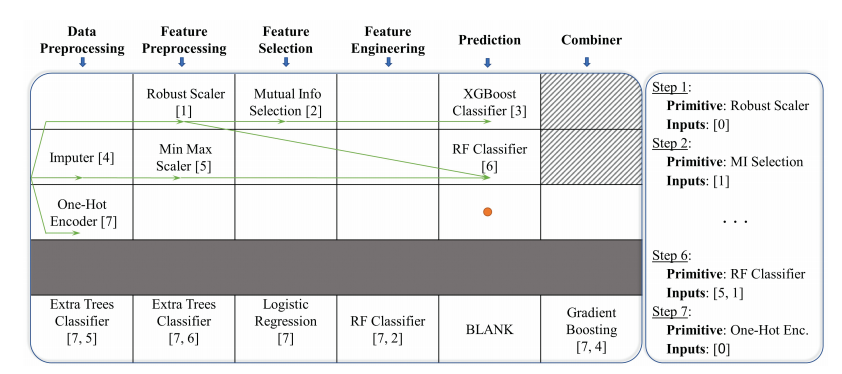
* מרחב מובנה (constrained space) – בהם קיים שלד של רכיבים שניתן להשתמש בהם בכל שלב
* מרחב לא מובנה (unconstrained space) – בהם אין אילוצים (או שיש מספר קטן של אילוצים) על הרכיבים בהם ניתן להשתמש בכל שלב, ואין "שלד" של מבנה ה-pipeline.

עוד דוגמה לבעיה דומה הוא neural architecture search – חיפוש ארכיטקטורה (מספר שכבות, מספר פילטרים, גודל פילטר, stride וכדו') המתאימה לבעיה 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 Reinforcement Learning and Hierarchical Actions Filtering" (מאמר מבית המחלקה להנדסת מערכות מידע בבן גוריון). במאמר מגדירים את משפחות הרכיבים הבאות:

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

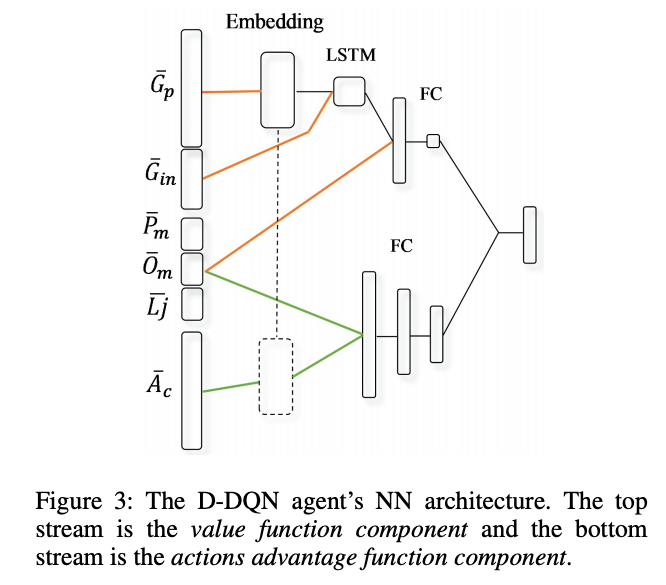
להלן תרשים של התהליך:



הסוכן משתמש במאפיינים (features) הבאים:

* – מייצג את אוסף אבני הבניין שנעשה בהן שימוש עד כה (למשל data preprocessing, feature selection וכדו')
* – הקשתות של ה-flow שהוגדר עד כה, או במילים אחרות המסלול של ה-data ב-pipeline
* – מטא-דאטא על הטופולוגיה של המסלול שהוגדר עד כה, למשל מספר הקודקודים והקשתות, העומק של הגרף וכדו'
* – מידע על ה-dataset, למשל מספר הדגימות, מספר המאפיינים וכדו'
* – נתונים על המטלה (למשל קלסיפיקציה או רגרסיה) ועל המטריקות שאותן מודדים (למשל accuracy או F1)
* – סט הפעולות שהסוכן יכול לבצע, למשל הוספה של בלוק, הורדה של בלוק או שינוי פרמטרים בבלוק ספציפי

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



בעיות נוספות בעלות מרחבי מצבים ופעולות גדולים שניתן לגשת אליהן באמצעות כלי RL:

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

## **Branching Dueling Q-Networks (BQD)**

המאמר "Action branching architectures for deep reinforcement learning" מציג שיטה להתמודד עם מרחב פעולות גדול, על ידי שימוש במספר רשתות אשר כל אחת תתמחה על חלק ממרחב הפעולות. בשיטה זו, המצב (state) נשלח לרשת משותפת אשר לומדת ייצוג שלו, והייצוג נשלח לכל אחת מתתי הרשתות אשר מתמקדות בפעולות ספציפיות. שיטה זו מתאימה לבעיות בהן ניתן לחלק את מרחב המצבים לתתי בעיות מוגדרות – למשל תנועה של גוף האדם יכולה להתחלק לתנועה של כל אחד מהמפרקים.

במאמר נעשה שימוש ב-dueling double DQN. כזכור, double DQN היא שיטת אופטימיזציה בה הרשת שאותה מאפטמים בוחרת את ה- אך ה-Q-value של אותה הפעולה a משוערכת על ידי רשת שניה. דבר זה נעשה על מנת להתגבר על אופטימיות היתר שיש באימון DQN סטנדרטית. Dueling 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 שניתן לראות באיור, ה-state עובר לרשת אשר מוציאה ייצוג משותף לתתי הרשתות. בענף אחד מחושב ה-state value הכללי, וכל תת רשת מחשבת את ה-action advantage של כל אחד מהפעולות שמוכלות בה. לאחר מכן ה-state value הכללית מתווספת ל-action advantage של כל ענף ויוצרת Q-function עבור כל ענף. לבסוף נבחרות הפעולות בעלות ה-Q-value הגבוה ביותר. מימוש זה דומה ל-dueling DQN במובן הזה שהוא מפריד בין ה-state value ל-action. החידוש הוא שכאן ה-state value משותף לכל תתי הרשתות. כמו ב-dueling DQN, ה-state value וה-action advantage מחוברים דרך שכבת אגרגציה, אשר מחסרת מה-action advantage את הממוצע (כך שהממוצע יהיה שווה לאפס), ומחברת את ה-state value.

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

כאשר היא ה-target network (מתוך double DQN). וה-loss חושב בצורה הבאה:  
כאשר הוא ה-buffer reply.

Diagram

Description automatically generated

במאמר מראים ביצועים טובים מאוד על tasks שונים.

## **Jointly-Learned State-Action Embedding**

ישנם מאמרים שמראים איך ניתן ללמוד ייצוג לפעולות ולמצבים במרחב embedding. עם זאת, ברוב המקרים 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 במרחבי מצבים ופעולות גדולים. היתרון בשיטה שהם מציגים זה שניתן ליישם אותה בכל אלגוריתם פוליסה מבוססת גרדיאנטים.

השיטה מורכבת מהפונקציות הגזירות (רשתות) אשר ממפה **ממרחב המצבים למרחב ה-embedding**, אשר ממפה **ממרחב ה-embedding למרחב הפעולות** ופוליסה אשר ממפה מצב לפעולה **בתוך מרחב ה-embedding**. להלן תרשים:

A picture containing text, clock

Description automatically generated

שיטה זו מאפשרת בצורה implicit ליצור מרחב משותף למצבים ולפעולות, תוך כדי למידת פוליסה . עם זאת, לא ניתן ללמוד את הפונקציות בצורה ישירה. לצורך כך מגדירים את אשר ממפה ממרחב הפעולות למרחב ה-embedding (הפוך מ-), ואת אשר ממפה מצב + פעולה (שניהם במרחב ה-embedding) למצב הבא (במרחב המצבים). בהינתן דגימות ניתן לחשב את המודלים כפי שמוצג באיור:

*Diagram

Description automatically generated*

*לאחר מכן, ניתן לחשב את בעזרת זוגות של שחושבו בפונקציה ההפכית – .*

# **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 – שהן כמובן פונקציה של המצב, אך לא תמיד פשוט להסיק מהן את המצב. אפשר להשתמש גם בשיטות model based וגם בשיטות model free ללמידה על בסיס ה-observations.*

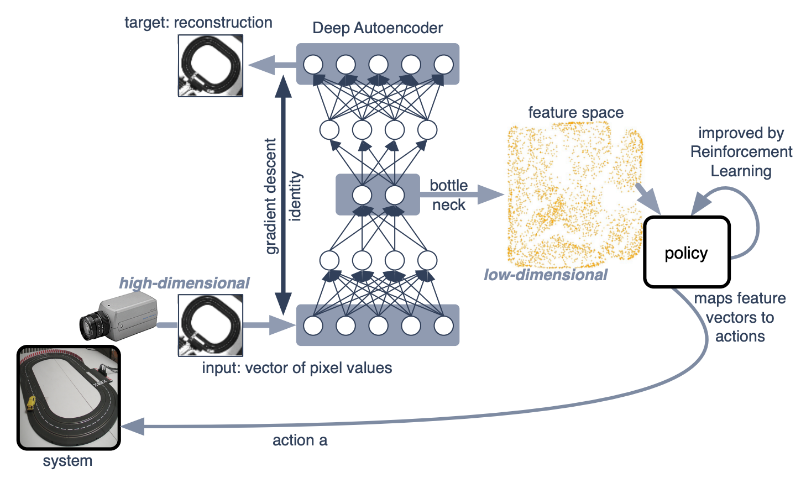
## **Model-Free Approaches for Learning the Latent Space**

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

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

1. *שימוש בפוליסה שמבצעת exploration על מנת לאסוף data על הדינמיקה של העולם.*
2. *שימוש ב-auto-encoder להעברת התמונה לוקטור מאפיינים מממד נמוך יותר.*
3. *שימוש ב-Q-learning ללמידת הפוליסה (נעשה שימוש בפריים אחד בכל צעד, ללא אינפורמציה מהעבר).*

*להלן תרשים של השיטה:*

**

*היתרון בשיטה הוא שהסוכן מצליח ללמוד בצורה יעילה על בסיס תמונה בלבד. החסרונות הן שלא מובטח שה-autoencoder יחלץ את המאפיינים הרלוונטיים (הוא מחלץ את המאפיינים שיסייעו לו לעשות שחזור של תמונת ה-RGB, ואינו מוכוון דווקא לחילוץ מאפיינים שיעזרו בבעיה שלנו). בנוסף, פתרון זה מתאים יותר ל-model free methods שבהם אנו עושים חיזוי ישיר של הפעולה מהדגימה. פתרונות model free בדרך כלל יותר קשים ללמידה, וההתכנסות איטית יותר.*

## **Model-Based Approaches for Learning the Latent Space**

*דוגמה אחרת היא המאמר "Deep Spatial Autoencoders for Visuomotor Learning", שבו מבצעים בקרה לזרוע רובוטית באמצעות מצלמה. השיטה במאמר היא model based, ונלמדת הדינמיקה של העולם. האלגוריתם עצמו בו משתמשים בלמידה נקרא Linear Gaussian Controllers והוא מתואר להלן:*

**

*נלמד מודל לינארי אשר חוזה את הפעולה בהינתן המצב . בכל איטרציה, נאסף מסלול באמצעות הפוליסה* *שנלמדה עד כה. באמצעות המסלול נלמד מודל של העולם על ידי מודל לינארי (רגרסיה לינארית). באמצעות המודל שנלמד, מעודכנת הפוליסה* *שתביא loss מינימלי, תחת האילוץ שהפוליסה לא תשתנה יותר מידי (נעשה שימוש ב-KL divergence). האילוץ קיים משיקולי בטיחות, על מנת שלא ליצור פוליסה לא צפויה ששונה מאוד מהפוליסה הקיימת. בצורה איטרטיבית מעודכן לחילופין המודל של העולם והפוליסה.*

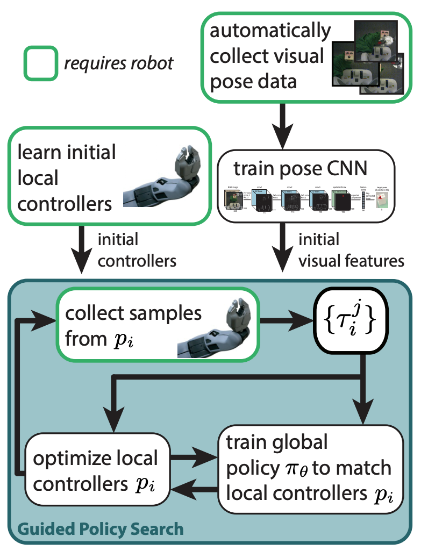
*השיטה השלמה מורכבת מהצעדים הבאים:*

1. *בשלב הראשון מתבצע אימון ראשוני של מודל, על ידי השיטה שתוארה לעיל. במודל זה, מרחב המצבים* ***אינו כולל מאפיינים מבוססי מצלמה****, אלא רק מיקומים וזויות של חלקי הרובוט ונגזרות זמניות שלהם. מודל זה אינו מוצלח במיוחד, אך נעשה בו שימוש לאיסוף data להמשך האלגוריתם.*
2. *אימון של spatial autoencoder על בסיס ה-data (הכולל תמונות ממצלמה) שנאסף מהמודל מהסעיף הקודם. ה-encoder מחלץ מאפיינים אשר מייצגים את מיקום האובייקטים בתמונה וה-state vector החדש מורכב ממוצא ה-AE אשר משורשר למאפייני הבקר (זויות, מהירויות וכדו'). בנוסף, מחושב ה-cost function כמרחק של מוצא ה-AE של התמונה בשלב זה ממוצא ה-AE של התמונה במצב הסופי שאליו מעוניינים להגיע.*
3. *אימון של סוכן בהתבסס באמצעות אותו האלגוריתם (model based) על וקטור המצב החדש (אשר מכיל מאפיינים שחולצו מהתמונה בנוסף למאפייני המנועים).*

*הרציונל מאחורי השיטה בה אימנו את ה-autoencoder בסעיף 2 הוא שעליו לקודד את* ***נתוני המיקום של האובייקטים בתמונה****, בעוד ש-autoencoder בדרך כלל מתרכז בקידוד* ***מה קיים בתמונה****. על מנת "להכריח" את הרשת להתרכז בלמידה של מאפיינים מרחביים, נעשה שימוש ברעיון הבא. לאחר שלושת שכבות הקונבולוציה הראשונות של ה-encoder, מתבצעת פעולת softmax על כל ערוץ בנפרד:  
כאשר הוא ערך האקטיבציה של פיקסל בערוץ (feature) , ו- הוא פרמטר טמפרטורה (היפר פרמטר). לאחר מכן, מחושב ממוצע משוקלל של* ***המיקומים*** *של האקטיבציות:  
כאשר הם האינדקסים של הפיקסלים. בצורה כזאת, מתקבל וקטור מאפיינים* *(בממד השווה למספר הפילטרים של שכבת הקונבולוציה השלישית), אשר כל מאפיין מכיל בתוכו את המיקומים (הממוצעים) של האקטיבציות של כל ערוץ. וקטור זה הוא הוקטור שבו נעשה שימוש ללמידת הבקרה.*

*לבסוף, ה-decoder מורכב משכבת FC פשוטה, אשר משחזרת תמונה שעברה downscaling.*

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

**

*השיטה מורכבת משני חלקים מרכזיים. הראשון הוא* ***רשת שמאומנת כ-supervised learning****, שחוזה פוליסה מהצורה , כאשר הפונקציות הנלמדות הן ( היא רשת CNN ו- נלמדת בצורה בלתי תלויה באובסרבציה). החלק השני הוא* ***סוכן RL*** *אשר יוצר התפלגויות (אשר נקראות guiding distributions), שבעזרתן מאמנים את הפוליסה באמצעות supervised learning. במהלך האימון, יוצרים מסלולים באמצעות אשר מדמים את הסביבה הפיזית. בצורה כזאת ניתן לאמן מודל גם בלי לאסוף data מהסביבה הפיזיקלית.*

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

*הרשת מורכבת מ-7 שכבות (גדול יחסית לאלגוריתמי RL), עם 92,000 פרמטרים. כמו בעבודה הקודמת, גם כאן נעשה שימוש ב-spatial softmax על מנת לחלץ מאפייני מיקום. על מנת לשמר את אינפורמציית המיקום בצורה מיטבית, הרשת אינה מבצעת pooling.*

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

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

*חסרונות:*

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

## **Embed to Control (E2C)**

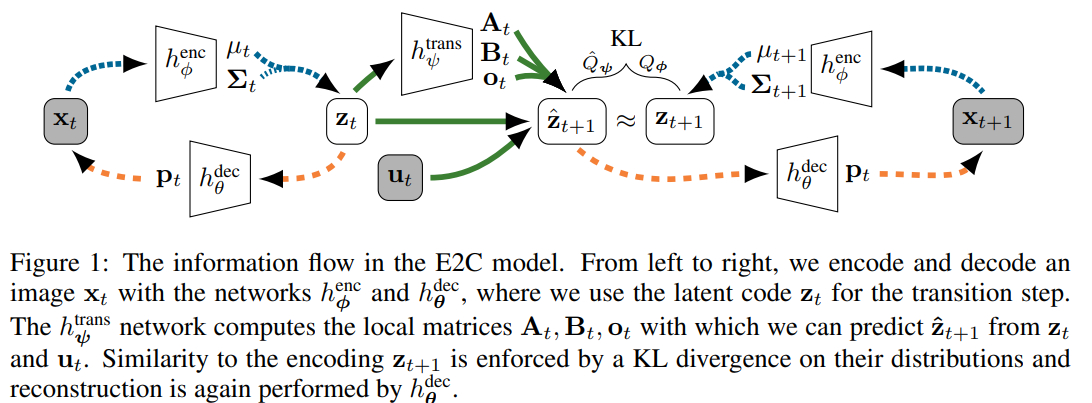
*E2C הוא מאמר נוסף אשר משתמש ב-Variational Auto Encoder ללמידה של latent space ל-RL. שיטה זו היא Model based, והיא מייצגת את המצב (state) ב-latent space מממד נמוך. מטרת המאמר הוא בקרה של מערכות מורכבות באמצעות מידע גולמי מממד גבוה (תמונות).*

*במאמר מניחים מודל מהצורה:*

*כאשר הוא הרעש של המערכת, ו-**היא הדינמיקה של הסביבה. בנוסף, ישנה גישה אך ורק למצלמה . המטרה היא למצוא פונקציה הממפה מהמרחב של למרחב בעל ממד נמוך יותר( הוא הוא הרעש של המערכת), כך שהבקרה תתבצע בממד הנמוך. מגדירים את פונקציית המעבר ממצב למצב בממד הנמוך:*

*את המודל של הסביבה ניתן ללמוד בממד נמוך על ידי לינאריזציה של הבעיה (הופכים את הבעיה ללינארית למקוטעין):*

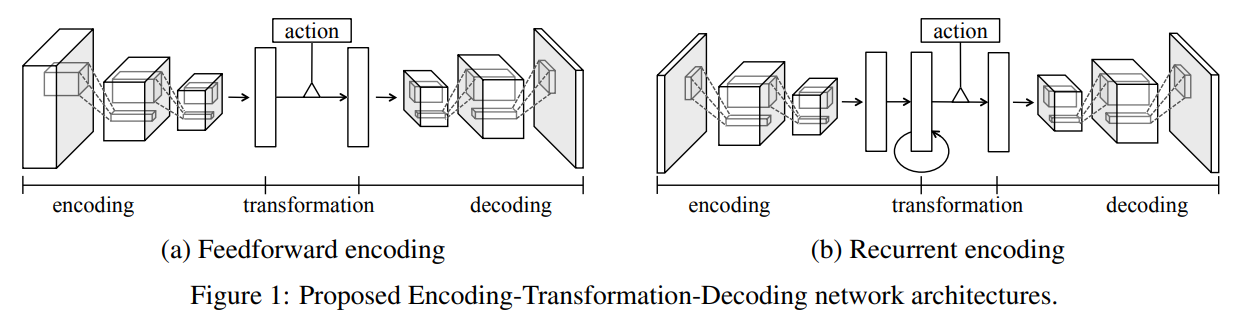
*כאשר ו ו- הוא ה-offset. לא נתעמק בפתרון, אך נאמר שלאחר הלינאריזציה ניתן להגיע לפתרון מקורב של מציאת הפעולות והמצבים אשר מביאים למינימום את פונקציית המחיר. להלן סכמה של השיטה:*

**

*כפי שניתן לראות, בשיטה זו נלמד גם הייצוג במרחב הנמוך על ידי ה-VAE, וגם הדינמיקה של המודל על ידי הרשת .*

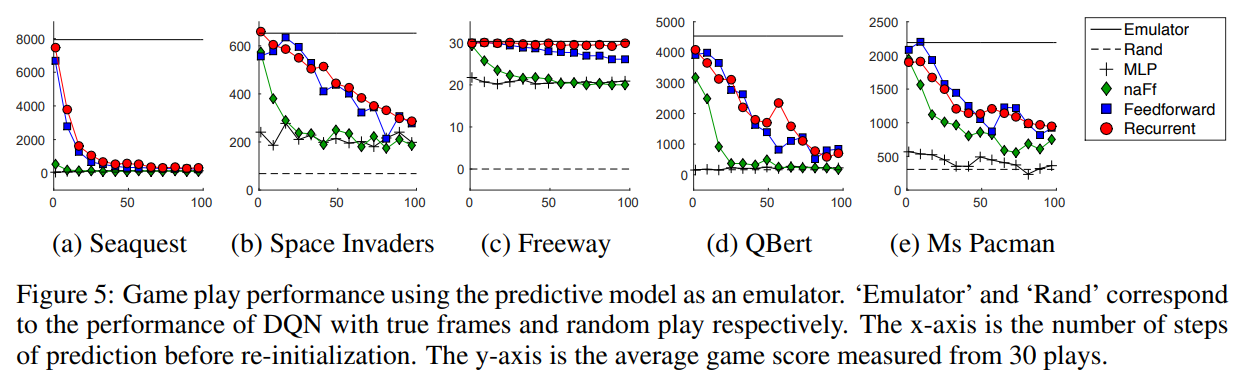
## **Action-Conditional Video Prediction**

*המאמר "Action-conditional video prediction using deep networks in Atari games" מציג שיטה מעניינת להתמודד עם מרחב מצבים גדול – תמונות. הרעיון שלו הוא ליצור רשת שחוזה את הפריים הבא, על בסיס הפריימים הקודמים והפעולה.*

**

*באיור ניתן לראות שתי ארכיטקטורות שונות עם אותו רעיון. בארכיטקטורה הראשונה מחברים (concatenate) m פריימים קודמים לאחר ה-encoder, ובעזרתם חוזים את הפריים הבא. בארכיטקטורה השנייה משתמשים ב-LSTM אשר מקבל את מוצא ה-encoder פריים אחרי פריים, וחוזה את הפריים הבא. את הפעולה הם מכניסים באמצעות מכפלה אבר אבר של מוצא ה-decoder במטריצה נלמדת עבור כל פעולה. בנוסף, מתבצע curriculum learning על פני הזמן, כאשר בהתחלה הרשת לומדת לחזות רק את הפריים הקרוב וככל שהאימון מתקדם הוא חוזה יותר קדימה.*

*בהקשרי RL, הציגו במאמר שתי מטרות: לנסות לבצע control באמצעות התחזית של הפריימים (במקום פריימים אמיתיים) ולבצע אקספלורציה באמצעות סימלוץ של פעולות שונות באמצעות הרשת המאומנת. להלן תוצאות של אימון DQN על הפריימים שיצרה הרשת, במקום לקבל פריימים מהמשחק:*

**

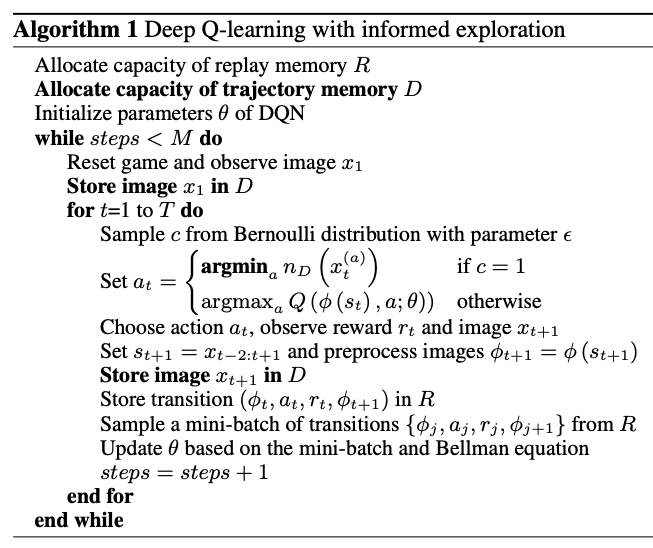
*ה-DQN מקבל רק פריימים מסומלצים. ציר x מציין אחרי כמה פריימים מאתחלים את הרשת שיוצרת את הפריימים עם פריים אמיתי שממנו תמשיך לחזות פריימים. ניתן לראות כי למרות שהביצועים יורדים, בחלק מהמקרים עבור אופק אירועים לא רחוק מידי, ניתן ללמוד גם באמצעות הסימולציה.*

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

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

*[](https://youtu.be/DLIWo16r5LA)*

*להלן האלגוריתם:*

**

*לסיכום, יתרונות השיטה:*

* *נראה שמצליחים לחזות לא רע את הפריימים הבאים*
* *האקספלורציה אכן משתפרת על ידי שיטת ה-informed exploration*

*חסרונות:*

* *לא ברור כמה שיטה זו תעבוד בבעיות עולם אמיתי, למשל תמונות ממצלמה אמיתית ולא משחק*
* *לא ברור כיצד ניתן להשתמש בשיטה זו ל-planning, בין השאר כי חסר מדד טוב לבחינת ביצועים (MSE פחות מתאים בעולם התמונות)*

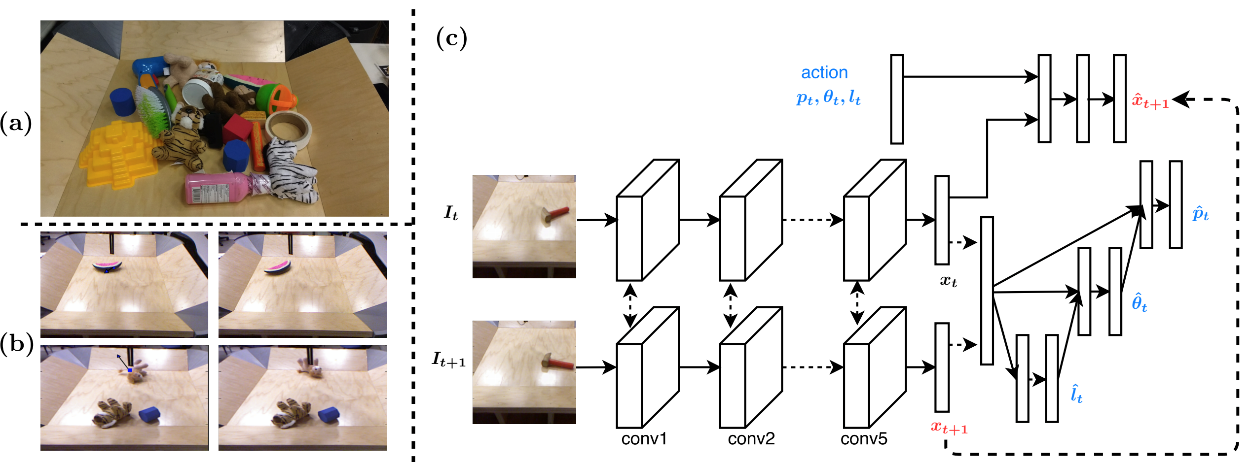
## **Inverse Reinforcement Learning**

*המאמר "Learning to poke by poking: Experiential learning of intuitive physics" מציע שיטה מעניינת ללמידת ה-latent space מתוך תמונה. הרעיון הוא שניתן להיעזר* ***במיפוי ההפוך*** *כלומר לחזות את הפעולה שהתבצעה מזוג תמונות (אחת לפני הפעולה ואחת אחריה). הסיבה לכך היא שתחזית ישירה, כפי שנעשה במאמרים הקודמים שהצגנו, היא מאוד קשה. סיבה אחת לכך היא שקשה לרשת לבצע בנייה של תמונה, במיוחד כשמדובר בעולם האמיתי. וסיבה שני, שהרבה מהמאפיינים שהרשת יוצרת בתמונה הם לא רלוונטיים, והרשת מתרכזת בחלקים הפחות חשובים לבעיות ה-RL.*

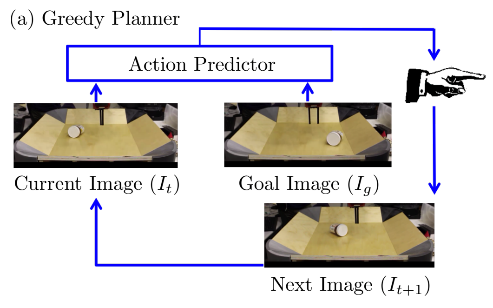
*במאמר משתמשים ברובוט שמזיז חפצים על גבי שולחן. מציגים שני מודלים: ה-forward model אשר חוזה את המצב הבא בהינתן המצב הקודם והפעולה , וה-inverse model אשר חוזה את הפעולה שהתבצעה בין שני מצבים. המודל שמעניין אותנו לחזות הוא ה-forward model, שכן נרצה לדעת את הפעולה שעלינו לבצע כדי לעבור ממצב למצב (כך נוכל להשתמש בכלים מתחום ה-model based RL). עם זאת, למידה זו קשה, מכיוון שקשה לרשת לחזות במדויק את כל הפיקסלים של המצב החדש, ובנוסף, ה-reconstruction המלא לא מעניין אותנו, שכן מה שחשוב הוא המאפיינים שקשורים למעבר ממצב למצב. מצד שני, את ה-inverse model קל יותר ללמוד, שכן יותר טבעי לרשת לקבל שתי תמונות ולהחזיר סקלר (או וקטור של פרדיקציות). הבעיה במודל זה היא שניתן להגיע ממצב למצב באמצעות מספר פעולות ולכן הלמידה עלולה להיות לא יציבה.*

*ה-loss במהלך האימון הוא:*

*כאשר הוא סכום של ה-cross entropy של שלושת מאפייני הפעולה (זווית, מיקום, אורך), ו- הוא בין התמונה הבאה לתמונה החזויה* ***במרחב המאפיינים****. נדגיש שנעשה שימוש לשתי המשימות באותה רשת עם אותם פרמטרים . להלן תרשים של השיטה:*

**

*כאשר הרובוט מתבקש 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 בתהליך הלמידה*
* *אין צורך לבצע reconstruction של התמונה הסופית*

*חסרונות*

* *לא ניתן לבצע planning עם ה-inverse model*
* *ה-inverse model מתרכז בלמידה של הפעולות, ולכן יכול להיות שהרשת לא תקודד מאפיינים חשובים שאינם קשורים ישירות לפעולה*

## **סיכום model based מול model free**

*לסיכום הפרק, להלן יתרונות וחסרונות של שיטות model based. יתרונות:*

* *יחסית קל לאסוף -data רלוונטי ב-scale, מכיוון שלומדים רק את המודל של העולם*
* *ניתן ללמוד ממטלה למטלה, שכן נלמד המודל של העולם, ולא מטרה ספציפית*
* *בדרך כלל, נדרשת כמות קטנה יותר של data מתוייג*

*חסרונות:*

* *הלמידה לא מאופטמת למטלה ספציפית, אלא מנסים "ללמוד הכל", לכן יתכן והביצועים על מטלה ספציפית יהיו נמוכים יותר*
* *לפעמים קל יותר ללמוד פוליסה בצורה ישירה מאשר ללמוד את המודל של המערכת*
* *לפעמים הלמידה של הדינמיקה קשה מידי, ונדרש להניח הנחות מקלות שעלולות לפגוע בביצועים*

*לעומת זאת, היתרונות של model free:*

* *לא נדרשות הנחות מלבד ה-reward function*
* *במקרים בהם הפוליסה מורכבת יותר קל ללמוד אותה ישירות*

*חסרונות:*

* *בדרך כלל נדרש הרבה data*
* *לא ניתן להעביר בצורה קלה את הידע ממטלה למטלה*

## **Exploration with Exemplar Model**

*נושא המאמר "Ex2: Exploration with exemplar models for deep reinforcement learning" הוא כיצד לבצע אקספלורציה יעילה במרחב מצבים גדול. בבעיות עם מרחב מצבים קטן, קיימות שיטות רבות לאקספלורציה באמצעות counters, כלומר שמירת מספר הביקורים בכל מצב והוספת "בונוס" ל-reward function כך שיעודד את הסוכן לבקר במצבים בהם הוא לא ביקר. למשל במאמר בשם MBIE-EB מוסיפים reward של כדי לעודד אקספלורציה. עם זאת, במקרים בהם מרחב המצבים גדול מאוד שיטות אלה אינן רלוונטיות. עם זאת, ניתן לבצע הרחבה שלהן, כלומר להעניק reward למצבים* ***השונים ביותר*** *מהמצבים בהם ביקרנו.*

*בשיטה המוצגת, מציגים שיטה בשם exemplar models, אשר בהינתן dataset מאומנים מודלים (discriminators) אחד לכל דוגמה, אשר חוזה 1 אם מציגים את הדוגמה המתאימה ו-0 אם מציגים דוגמה אחרת. במאמר עצמו, נעשה שימוש ברשת אחת מאוחדת לכל הדוגמאות, אבל הרעיון המקורי מבוסס על מודלים. ה-discriminators מאומנים באמצעות ה-loss הבא:  
כאשר הוא ההתפלגות של הדוגמה (התפלגות של דוגמה אחת), ו- היא ההתפלגות של שאר הדגימות. כעת מציגים כיצד exemplar model יכול לשמש ל-density estimator.*

*קודם כל, מוכיחים במאמר שבמקרה האופטימלי ה-discriminator (בבעיה הדיסקרטית) יהיה:  
כאשר היא פונקציית הדלתא של דיראק, ו- היא פונקציית ההתפלגות של ה-data מלבד . אינטואיטיבית, כאשר ההסתברות לקבל את בשאר ה-data שואף לאפס, ה-discriminator יחזיר ערך 1 (100% הצלחה), וכאשר ההסתברות שואפת לאחד (למשל dataset שמורכב רק מ-) הערך יהיה 0.5 (50% הצלחה) – הטלת מטבע. נשים לב, שמקרה בו מתקבל ולכן ניתן לשערך את ההתפלגות על ידי:  
כל זה נכון במקרה הבדיד, אבל במקרה הרציף פונקציית הדלתא של דיראק שואפת לאינסוף ולכן לא ניתן לשערך את ההתפלגות. עם זאת, ניתן לפתור את העניין על ידי טריק מתמטי. אם מוסיפים רעש לדגימה במהלך האימון, ניתן להוכיח (לא נוכיח פה) שההתפלגות מקיימת:  
בסופו של דבר, ל-reward הסטנדרטי מוסיפים חלק של "novelty", אשר תלוי ב-, ה-discriminator הנלמד:  
כאשר הוא היפר פרמטר לכוונון היחס בין ה-exploration ל-exploitation, ו- היא פונקציה התלויה ב-, אשר נרצה שתהיה בעל ערך גבוה כאשר המצב נדיר יותר, ולהיפך. במאמר משתמשים בפונקציה (כאשר תלוי ב-). להלן האלגוריתם המלא:*

*Text

Description automatically generated*

*לצורך השלמת התמונה, האלגוריתם הבסיסי של exemplar models משתמש במודל שונה לכל דגימה. עם זאת, במאמר נעשה שימוש ברשת יחידה אשר משערכת את ההתפלגות של בהינתן . להלן סכמה:*

*A picture containing graphical user interface

Description automatically generated*

## **Curiosity-driven Exploration by Self-supervised Prediction**

*המאמר האחרון שנסקור בנושא זה הוא "Curiosity-driven Exploration by Self-supervised 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 האינפורמטיביים. סקרנות במובן 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 rewards הקשורים לסקרנות, ופוליסה שתפקידה למקסם את ה-reward הזה. בנוסף, הסוכן יכול לקבל גם reward חיצוני המגיע מהסביבה. רכיב הפוליסה מנסה למקסם את כאשר הוא ה-reward האינטרינזי (סקרנות) ו- הוא ה-reward האקסטרינזי (מהסביבה). הוא אפס רוב הזמן (sparse). נעשה שימוש ב-A3C (שנסקר בפרקים קודמים) לצורך אימון הפוליסה .*

*הכותבים מסבירים מדוע שימוש ב-raw data של המצב, כלומר הפיקסלים במקרה של תמונה, לא מתאימים לצורך מידול הסקרנות של הסוכן. הם מביאים דוגמה – עצים בעלי עלים שזזים ברוח. במקרה זה, הסוכן לא יצליח לחזות לעולם את הפיקסלים הללו (אם הם זזים בצורה אקראית), ולכן ציון הסקרנות יהיה שם לעולם גבוה. סוכן זה יזדקן וימות בעודו צופה בסקרנות בעלי העץ הזזים ברוח. במאמר מסווגים את העולם לשלושה סוגים: observations שהסוכן יכול להשפיע עליהם, observations שהסוכן לא יכול להשפיע עליהן אך יכול להיות מושפע מהם (למשל רכבים שאחרים נוהגים), וכאלה שאין לו השפעה עליהם והם גם לא יכולים להשפיע עליו. סקרנות בשני הסוגים הראשונים יכולה להיות פרודוקטיבית, אך בסוג השלישי לא.*

*במאמר משתמשים בשיטה שהזכרנו לעיל (למשל ב-inverse RL) – חיזוי הפעולה באמצעות ה-feature vectors של שני פריימים עוקבים. ממירים בשלב הראשון את הפריימים למרחב מאפיינים באמצעות רשת, ובשלב שני חוזים את הפעולה שהובילה מ- ל-. אימון רשת כזאת מאפשרת ל- לקודד רק את המאפיינים שרלוונטיים לפעולה. מודל זה נקרא inverse dynamics model – חיזוי הפעולה מהדינמיקה במקום לחזות את הדינמיקה מהפעולה. מודל זה הינו אינווריאנטי לפוליסה, שכן היא ממדלת את הסביבה.*

*לאחר שלמדנו את ה-inverse dynamics model, אנו יכולים להשתמש בקידוד שנלמד על מנת ללמוד את הדינמיקה במרחב המאפיינים (forward dynamics model):  
לאחר שנלמד המודל ניתן ליצור את ה-intrinsic reward signal, כלומר את ה-reward על הסקרנות, באמצעות:*

*האימון של ה-forward וה-backward models נלמדים ביחד. להלן תרשים של השיטה:*

**

*לסיום, להלן התוצאות:*

*Chart

Description automatically generated*

# **Transformers in RL**

*אף על פי ש-RL מבוסס על sequential data רוב המודלים שמשתמשים בהם לעיבוד data כזה בתחום הם RNNs. הסיבות לכך הן שטרנספורמרים דורשים כמות גדולה מאוד של data כדי להגיע לביצועים טובים ושהם קשים לאופטימיזציה. אך למרות 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 לבעיות RL. במאמר מוצגת הארכיטקטורה Gated Transformer-XL אשר מראה ביצועים טובים יותר מ-LSTM ומרשתות מבוססות זיכרון במספר מטלות. בנוסף, ארכיטקטורה זו לומדת מהר יותר ובצורה יציבה יותר מהארכיטקטורה הבסיסית של הטרנספורמר.*

*השינויים המרכזיים בארכיטקטורה הם:*

* *שינוי מיקום ה-layer normalization. בארכיטקטורה הקלאסית, הנורמליזציה מגיע מיד אחרי ה-skip connection בבלוק ה-attention. בצורה כזאת ה-skip connection מעביר את המידע* ***לאחר נורמליזציה****. במאמר שינו את המיקום כך שימוקם* ***אחרי*** *כל skip connection. אין שינוי מבחינת ה-multi-head attention וה-MLP שאחריו, אבל ה-data העובר ב-skip connection מנורמל (ראה איור למטה).*
* *שינוי ה-positional encoding כך שיתמכו ברצפים ארוכים יותר.*
* *הוספת gating mechanism ושינוי של ה-layer normalization. ניסו במאמר מספר שיטות gating, בין השאר:  
  שיטה זו, בדומה למה שנעשה ב-LSTM ו-GRU, נועדה לווסת את זרם האינפורמציה, ולאפשר לחסום חלק ממנה.*

Diagram

Description automatically generated

*הכותבים הציגו שני מדדים לארכיטקטורה החדשה, זיכרון (האם המודל מצליח לחלץ מידע גם מ-tokens רחוקים) ו-reactive tasks (פעולות מורכבות כמו משחקי Atari). המודל הראה שיפור בביצועים בשני המדדים.*

## **Longformer ו-Big-Bird**

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

*Longformer מציג מספר attention patterns:*

* *Sliding window – בשיטה זו, משתמשים ב"חלון רץ" לצורך העיבוד, כאשר עבור כל token מתבצע עיבוד רק של מספר קבוע של tokens לפניו ואחריו.*
* *Dilated sliding window – שילוב של sliding window ו-dilated convolution. בשיטה זו, ה-attention מתבצע על חלון מסביב ל-token, אך כדי להגדיל את ה-receptive field של העיבוד, החלון מתבצע ב"קפיצות" כמו הקונבולוציה ב-dilated convolution. באמצעות שיטה זו ניתן לעבד אפילו עשרות אלפי tokens. ה-dilation rate נקבע בצורה שונה בכל שכבה, כאשר בשכבות הראשונות הוא נמוך יותר, על מנת ללמוד מאפיינים מקומיים, ובהמשך הוא גדל.*
* *Global attention – במידה ויודעים ש-tokens מסוימים חשובים במיוחד (למשל רצף של הודעת HTTP שבה ה-header יהיה חשוב) ניתן לקבוע tokens מסוימים אשר יהיו מחוברים לכל הכניסות בשכבת ה-attention.*

*להלן תרשים של ה-patterns:*

Chart, table

Description automatically generated with medium confidence

*Bid-Bird היא עבודה נוספת, שנעשתה באותה הזמן (עם longformer), ומתבססת על רעיון דומה. בנוסף ל-window attention ול-global attention, נעשה שימוש ב-Random attention – זוגות של tokens שמוגרלים, כאשר כל זוג ש"זוכה" בהגרלה מתחבר בשכבת ה-attention.*

## **The Decision Transformer**

*שיטה זו משתמשת ב-transformer לבעיות model free offline RL, על ידי מידול הרצפים. כל רצף (episode) ממודל כ:  
כאשר הוא ה-return to go (סכום ה-rewards עד סוף ה-episode), והמודל לומד לחזות את* ***הפעולות****, באמצעות ארכיטקטורה הדומה ל-GPT (generative). להלן תרשים של השיטה:*

Timeline

Description automatically generated

*הלמידה נעשית בדיוק כמו מודל שפה, באמצעות masking, אם כי המודל נדרש לחזות רק את הפעולות. ב-test time, המשתמש נותן למודל את ה-return המבוקש – כלומר הגבוה ביותר, והמודל יכווין אותו לפעולות שיתנו לו את זה. אם המשתמש ייתן למודל return לא ראליסטי, המודל יתבדר. במאמר מראים ביצועים טובים של המודל במספר מטלות (משחקי Atari).*

## **Trajectory Transformer**

*עבודה זו גם עוסקת ב-offline learning ובמידול בעיית RL כ-sequence modeling. הכותבים ממדלים את הבעיה בצורה הבאה. Trajectory אחד מסומן כ:*

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

*בנוסף, התבצע נרמול לכל אחד מהממדים (הם השוו נרמול Uniform ו-quantile). הארכיטקטורה בה השתמשו היא GPT. ה-loss בו השתמשו הוא:  
כאשר היא הרשת (הטרנספורמר). אפשר לשים לב כי כל ממד בפעולות ובמצבים 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 מבוססי גרדיאנטים. השיטה הפשוטה ביותר היא random search, אבל קיימות שיטות נוספות (מבוססות סטטיסטיקה). שיטות אלו בדרך כלל דורשות יותר דגימות ויותר איטיות עד להתכנסות. נסקור מעט שיטות כאלה בפרק הבא.
* Fully online methods – למשל actor critic, מבוססים על למידה של הסביבה ועל גבי זה של הפוליסה. שיטות אלו יותר יעילות מ-gradient free methods.
* Policy gradients methods – שיטות אלה לומדות את הפוליסה בצורה ישירה, ולכן בדרך כלל דורשות פחות דגימות על מנת להתכנס (מאשר fully online methods).
* Off policy methods – למשל Q-learning, יותר מהירות (ודורשות פחות דגימות) משיטות ה-policy gradients, מכיוון שהפוליסה שנלמדת היא הפוליסה האופטימלית, והיא נלמדת במהלך כל הלמידה, גם כאשר הפוליסה שהסוכן משתמש בה עדיין לא אופטימלית.
* Model-based DRL – למידה של הדינמיקה של העולם. שיטה זו יותר יעילה משיטות ה-off policy מכיוון שבהרבה מקרים הסביבה יותר קלה להבנה מאשר המטלה בלי ידיעת הסביבה, ואם הסביבה ידועה קל ללמוד את המטלה. נסקור שיטות כאלה בפרק זה.
* Model-based “shallow” DRL – שימוש ברשת רדודה או באלגוריתם פשוט יחסית ללמידת הסביבה. בצורה כזאת הלמידה מהירה באופן משמעותי.

*בפרק זה נרחיב על model-based DRL. נציין שלמרות ששיטות אלה הן מהירות ויעילות, הן דורשות הנחות מסוימות לגבי המידע שלנו על הסביבה והדינמיקה שלה. אף על פי שהן יותר יעילות מבחינת מספר דגימות, אימון שלהן יכול להיות מורכב. כאשר הצלחנו ללמוד את הדינמיקה של הסביבה ניתן להשתמש בשיטות פשוטות (אפילו לינאריות) לפתרון בעיית ה-RL. על מנת ללמוד את הדינמיקה ניתן לבצע דגימה של המרחב וללמוד ממנה, אם כי יכול להיווצר model bias, כלומר שבאזורים מסויימים לא נצליח ללמוד את הסביבה, או לפחות לא בצורה מספיק מדויקת. במקרים בהם המרחב גדול, קשה לדגום אותו בצורה "צפופה", ולכן המודל יעריך את הפונקציה במקומות הדלילים. קיימים מודלים אשר יודעים גם להעריך אי וודאות (במקומות בהם אין לנו מספיק data). נשתמש במודל (רשת נוירונים) על מנת למדל את הסביבה. רשת זו צריכה להיות גזירה. להלן נציג מספר שיטות ללמידה של מודל כזה.*

## **גישות ללמידת מודל סביבה**

*גישה ראשונה:*

1. *שימוש במודל ראשוני לאיסוף data (מודל זה יכול להיות אקראי).*
2. *שימוש ב-data זה לאימון מודל .*
3. *לאחר מכן ניתן להשתמש במודל על מנת לבחור פעולה. ניתן לבצע זאת על ידי backpropagation ממוצא המודל, בו נחזה המצב הבא, עד לכניסה למודל, בו התקבלה הפעולה, וכך לבחור פעולה אשר תמקסם את הסיכוי להגעה למצב הרצוי.*

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

*גישה שניה – DAgger:*

*כפי שהוסבר לעיל, DAgger מבצע איסוף של נתונים שנצפו לאורך הזמן, כאשר הפוליסה מתעדכנת בכל פעם שנאסף מספיק data. שלבי השיטה:*

1. *שימוש במודל ראשוני לאיסוף data.*
2. *שימוש ב-data זה לאימון מודל .*
3. *Backpropagation ממוצא המודל, לכניסה למודל, ובחירת פעולה .*
4. ***ביצוע פעולה ואיסוף דגימות חדשות על בסיסה , דגימות אלה מתווספות ל-dataset***
5. ***חזרה לשלב 2***

*בעיה 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 בתדירות את המודל:*

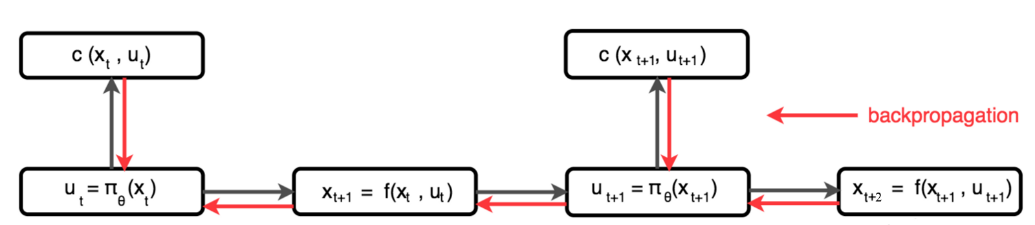
1. *שימוש במודל ראשוני לאיסוף data.*
2. *שימוש ב-data זה לאימון מודל .*
3. *Backpropagation ממוצא המודל, לכניסה למודל, ובחירת פעולה .*
4. *ביצוע פעולה ואיסוף דגימות חדשות על בסיסה , דגימות אלה מתווספות ל-dataset.*
5. ***אחת ל-N צעדים – חזרה ל-2 (ולמידה מחדש של מודל), אחרת חזרה לשלב 3.***

*גישה 4 – למידת הפוליסה:*

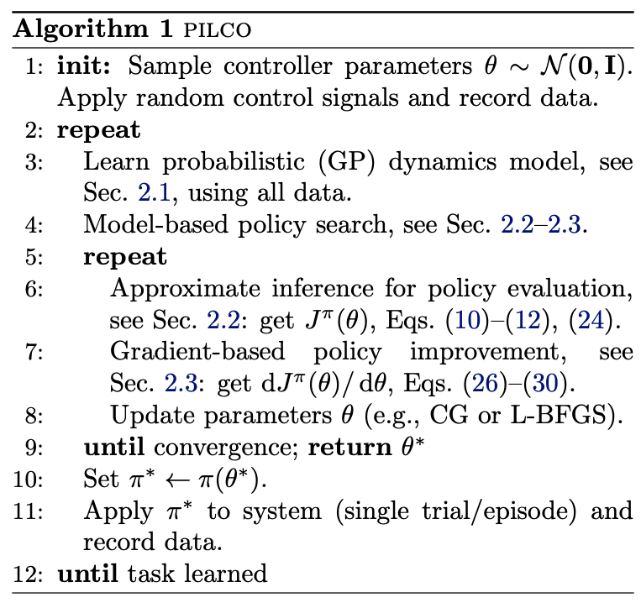
*בגישה זו, לומדים במהלך האימון* ***גם את הדינמיקה וגם את הפוליסה*** *(למשל באמצעות רשת). הרעיון הוא שאם הפוליסה והמודל של הסביבה שניהם גזירים, ניתן לבצע backpropagation גם לפוליסה. שלבי האלגוריתם:*

1. *איסוף data על ידי פוליסה התחלתית*
2. *שימוש ב-data זה לאימון מודל .*
3. ***ביצוע backpropagation דרך לפוליסה ועדכון הפוליסה על פי ה-reward שהתקבל***
4. ***הרצת לאיסוף data והוספתו ל-dataset***

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

**

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

**

*לסיכום:*

|  |  |  |  |
| --- | --- | --- | --- |
| ***גישה*** | ***סיכום*** | ***יתרונות*** | ***חסרונות*** |
| *1* | *איסוף data, למידת דינמיקה, planning* | *פשוטה, אין איטרציות* | *בעיית השינוי בתהפלגויות* |
| *2* | *איסוף data, למידה ו-planning בצורה איטרטיבית* | *פשוטה, פותרת את בעיית השינוי בתהפלגויות* | *ה-planning הוא open-loop, כלומר לא מתקבל פידבק על הפוליסה, ביצועים ירודים במיוחד בסביבות סטוכסטיות* |
| *3* | *MPC – planning בכל צעד, עדכון המודל כל N צעדים* | *רובסטית לטעויות קטנות* | *יקר חישובית* |
| *4* | *Backpropagation לפוליסה* | *מהירה בזמן ה-inference* | *יכולה להיווצר בעיה של חוסר יציבות, במיוחד עם מסלולים ארוכים (vanishing/exploding gradients)* |

## **LQR**

*עד כה דיברנו על מודלים* ***גלובליים****, מודל אחד שמידל את כל הסביבה. גישה אחרת ללמידה של מודל סביבה היא באמצעות מספר מודלים לוקליים, אשר כל אחד ממדל חלק אחר בסביבה. בגישה זו נחפש מודל פשוט יותר אשר ממדל טוב את הסביבה שבה אנו נמצאים ברגע זה.*

*במודל לינארי, ניתן למדל את הדינמיקה על ידי:*

*בנוסף, נניח שקיים cost ריבועי מהצורה:*

*הסיבה שאנו משתמשים ב-cost ריבועי ולא לינארי היא שאנו מעוניינים בגודל חיובי. מטריצות מוגדרות חיובית, והן ממשקלות את ה-cost של ושל . למשל, state מסוים יכול להיות עם ערך גבוה (רכב שנופל מצוק), ופעולה מסוימת יכולה להיות עם ערך גבוה (למשל בזבוז משאב במשחק אסטרטגיה). המטרה ב-LQR היא למצוא מטריצה K שמקיימת:  
מטריצה זו נקראת gain matrix, וניתן להוכיח אנליטית שהיא מביאה למינימום את .*

*במקרה שלנו, ניתן לבטא את ה-cost function בצורה הבאה:  
כאשר כל מצב מבוטא באמצעות המצב הקודם בצורה רקורסיבית. בביטוי זה ישנו רק המצב ההתחלתי ורצף הפעולות . את מטריצת המעבר ניתן לבטא באמצעות הביטוי הבא:*

*כאשר מטריצה היא מטריצת המעבר (שילוב של מטריצות שהזכרנו למעלה, ו- הוא קבוע. את פונקציית המחיר ניתן לבטא בצורה הבאה (בשונה מהצורה הקודמת):*

*כאשר היא מטריצה הממשקלת את הביטויים הריבועיים , ו- הוא וקטור הממשקל את הביטויים הלינאריים . נשים לב ש- תלויים ב-t, מכיוון שניתן להציב בהם משתנים שתלויים בזמן. כעת, ניתן להתחיל ללמוד את המקבילה ל-Q-function – cost to go. כדי לבלבל, נסמן גם אותה ב-Q. נתחיל מהסוף (צעד T) ונתקדם אחורה, כמו ב-value iteration. נתחיל בצעד האחרון:  
ה-const הוא קבוע (שניתן להציב בו אפס), שנותן לנו אפשרות להוסיף cost קבוע למסלול. נשים לב כי את הוקטור בנינו כך שהאיברים הראשונים שייכים למצב, והאחרונים לפעולה. לכן ניתן לפרק את בצורה נוחה לרכיבים שמכפילים איברים של , של או של איברים מעורבים. ניתן אם כן לכתוב אותה בצורה הבאה:*

*כאשר ה-subscripts מסמנים את אילו ערכים כופלת כל תת מטריצה. בדומה לכך, ניתן לפרק את הוקטור :  
מכיוון שאנו מעוניינים באופטימיזציה על הפעולות, נגזור את לפי . ניתן אלגברית לייצג את הנגזרת בצורה הבאה:  
ומכיוון שהבעיה קונבקסית (כי היא ריבועית) ניתן למצוא מינימום גלובלי על ידי השוואת הנגזרת לאפס:  
כאשר ו-. כלומר מצאנו את שיביא למינימום את ה-cost function תחת הנחות הבעיה (פונקציית מעבר מצב לינארית ופונקציית מחיר ריבועית). מכיוון שאנו בשלב T, . לכן:*

*ניתן להגדיר:*

*ולקבל את הביטוי המפושט:*

*כעת ניתן להתקדם צעד אחד אחורה. ה-cost-to-go של המצב T-1 הוא:  
ניתן לבטא את באמצעות הפונקציה הלינארית של הדינמיקה של המודל: , ולכן ניתן לבטא את בצורה הבאה:*

*כאשר במהלך האחרון כל הקבועים (כל הביטויים שאינם תלויים ב- עברו תחת המשתנה  . כעת ניתן להציב את במשוואה של . ניתן לראות שגם ב- יש חלק לינארי וחלק ריבועי, ולכן ניתן לקבץ אותם ביחד כאשר מציבים אותם ב-:*

*כאשר:*

*ושוב ניתן לגזור על מנת למצוא פעולה אופטימלית עבור כפי שעשינו עבור T:*

*ולמצוא את האופטימלי. בצורה הזו ניתן למצוא את הפעולות האופטימליות עד .*

*שיטת LQR מתאימה בעיקר למקרים פשוטים, בהם נרצה להישאר בקרבת מצב נתון (למשל cart-pole). ניתן להשתמש בה גם כאשר הדינמיקה לא דטרמיניסטית, אלא עם פילוג גאוסי עם תוחלת שניתן למדל לינארית:  
ושונות שהיא פונקציה של ה-cost. במקרה כזה, עדיין ניתן להשתמש בפתרון . עם זאת, במקרים בהם הדינמיקה אינה לינארית, לא ניתן להשתמש בשיטה זו, ונדרש להתקדם לשיטות מתקדמות יותר.*

## **iLQR**

*iLQR היא ההרחבה הלא-לינארית של LQR (iterative LQR). בשיטה זו, מקרבים דינמיקה לא לינארית על ידי לינאריזציה באמצעות טור טיילור. הדינמיקה של LQR היא לינארית, ולכן הקירוב לדינמיקה ב-LQR הוא מסדר ראשון:  
ואילו ה-cost function ב-LQR היא ריבועית, ולכן נקרב אותה על ידי סדר שני:*

*לצורך נוחות, נגדיר קירובים ל-:*

*כאשר , , ו-.*

*האלגוריתם עובד בצורה הבאה:*

*בשלב ראשון, בהינתן מצב התחלה מבצעים סדרה של פעולות ומקבלים את המצבים .*

*בשלב הבא מבצעים את הפעולות הבאות בצורה איטרטיבית עד להתכנסות:*

1. *חישוב מהדינמיקה הלא-לינארית על כל אחד מהמצבים והפעולות*
2. *חישוב*
3. *חישוב*
4. *הרצת LQR על המצבים והפעולות*
5. *הרצה של המסלול האופטימלי שנצפה* ***עם הדינמיקה הלא-לינארית***
6. *עדכון המצבים והפעולות*

*iLQR רחוק מלהיות מודל מושלם. לאחר האופטימיזציה מבצעים "שגר ושכח", 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 שכדאי להכיר, והיא משמשת כאבן בניין באלגוריתמים מורכבים יותר.*

## **Model Predictive Controls (MPC)**

*הרעיון המרכזי ב-MPC הוא עדכון של iLQR לאורך המסלול. מבצעים חישוב iLQR בכל נקודת זמן t עד T, במקום לבצע את האופטימיזציה פעם אחת בתחילת המסלול. המטרה היא להתגבר על אירועים לא צפויים במהלך המסלול, ולבצע תיקונים ככל שמתקרבים לסיומו. בכל שלב t מחשבים את ה-iLQR ומקבלים , אך משתמשים רק ב-.*

*לפעמים, במקום לחשב את ה-iLQR עד לסוף המסלול, משתמשים בשיטה שנקראת "receding horizon control". בשיטה זו קובעים חלון T קבוע, ומבצעים את האופטימיזציה עד זמן . שיטה זו פותרת את הבעיה שיתכן ובעתיד הרחוק אירוע בלתי צפוי ישנה את המסלול ובכך ייתר את החישובים שהתבצעו. בנוסף, גם כאשר מתקרבים לסוף המסלול, ו- נהיה אחרי תום המסלול, עדיין האלגוריתם "אופטימי" מכיוון שהוא חושב שיש לו עוד זמן. עובדה זו גורמת לו לא "להתייאש" כאשר הוא רואה בהרצה של ה-iLQR שה-cost גבוה מידי.*

*לסיכום נושא ה-local models נעיר שניתן להשתמש בשיטות לוקליות שנלמדו על מצבים שונים על מנת ללמד מודל גלובלי יחיד (רשת נוירונים). למשל, ניתן להשתמש במודל LQR על מנת למדל את ההתנהגות בסביבה הקרובה, ואז להשתמש בפרדיקציות על מנת לאמן רשת policy gradients. בנוסף, ניתן להשתמש במספר רשתות קטנות למידול הסביבה הקרובה (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 משיטות 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:*

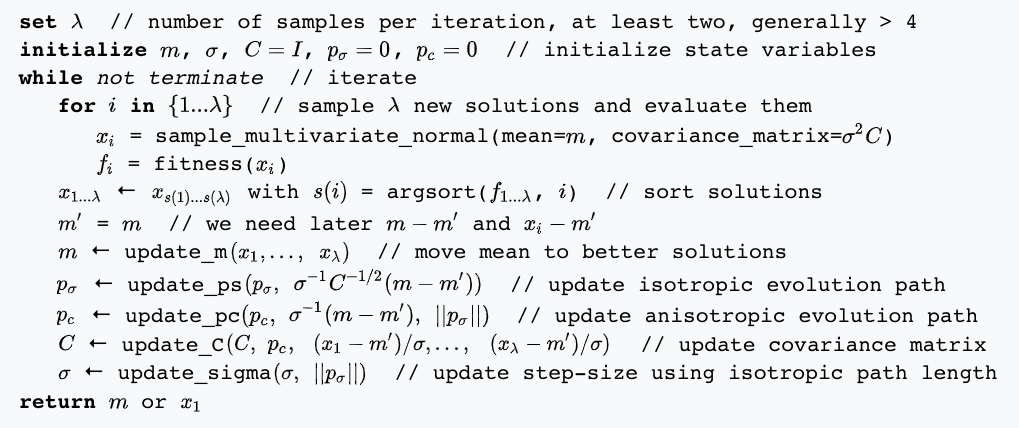
* *Selection – תהליך אקראי אשר בוחר את האלגוריתמים הטובים על פי ה-fitness function.*
* *Mutation* – יצירת "ילדים" לאלגוריתמים אלו, באמצעות שינויים קטנים באלגוריתמים המקוריים.
* *Recombination* – שילוב של מספר אלגוריתמים לאלגוריתם אחד

גם באלגוריתמים אלו ניתן להתייחס ל-exploration exploitation tradeoff, כאשר יצירת האלגוריתמים החדשים היא האקספלורציה, ובחירת האלגוריתמים הטובים היא ה-exploitation.

תהליך האופטימיזציה הוא black-box במובן שניתן לעבוד עם פונקציית fitness שרק מחזירה הערכה של טיב האלגוריתם. אלגוריתמים אלו מתאימים גם לבעיות שאינן קונבקסיות ושאינן "מתנהגות יפה" (שינויים חדים בפונקציה, רעש, נקודות מינימום לוקלי). חלק מהאלגוריתם משתמש בשיטות דומות לשיטת ניוטון המבוססת על נגזרת שניה (אך בלי backpropagation ובלי *gradient descent.*

*אלגוריתם החיפוש מתבצע בצורה הבאה. בהתחלה מתבצע אתחול לפרמטרים של מודל הבחירה (יכול להיות רשת), ולגודל האוכלוסייה . בכל איטרציה, נדגמות דגימות בעזרת ההתפלגות של המודל , ומתקבלים הפתרונות (האלגוריתמים) . מחושב הערך של ה-objective function על הדגימות . לבסוף מתעדכנים הפרמטרים על ידי פונקציה ייעודית . עדכון זה מתבצע על מנת שהדגימה תהיה יעילה יותר.*

*עוד שיטת Gradient Free היא "Covariance Matrix Adaptation Evolution Strategy (CMA-ES)". מטרת השיטה היא לאפשר לדגום ממרחב הפתרונות פתרונות עם ביצועים טובים. השיטה מבוססת על Maximum Likelihood – חיפוש הסתברות שתסביר את הדגימות בצורה "הטובה ביותר" (כלומר שבהינתן פונקציית ההסתברות הזאת, ההסתברות שהדגימות שהתקבלו להתקבל היא הגבוהה ביותר מבין פונקציות ההתפלגות). הדגימות במקרה זה הן פתרונות. בנוסף, השיטה מתבססת על covariance matrix adaptation, שינוי הדרגתי של מטריצת ה-covariance, כך שתתמקסם ההסתברות לקבל את הפתרונות הטובים שהתקבלו בעבר. האופטימיזציה מתבצעת על ידי Natural Gradient Descent, שיטה לאופטימיזציה של פונקציית התפלגות. בנוסף להתחשבות בגרדיאנטים, NGD מבצעת עדכון גם על פי* ***הגיאומטריה של ההתפלגות*** *כך שהצורה של ההתפלגות תהיה דומה לזו הנצפית בדגימות. נעשה שימוש במטריצת האינפורמציה של פישר (Fisher information matrix) אשר מכמתת תכונות גיאומטריות של מרחב הפרמטרים. לצורך כך ניתן להשתמש בדגימת MC, על מנת להעריך את המטריצה. להלן תיאור האלגוריתם:*

**