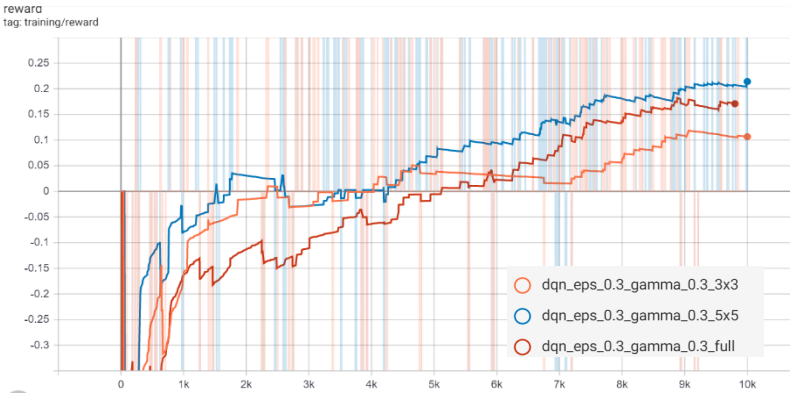
שם: אריאל ברוך

ת"ז: 205925704

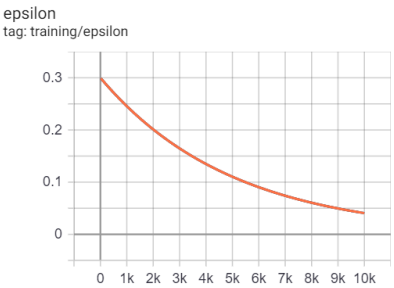
תרגיל 4 – Reinforcement Learning

אלגוריתם Q-Learning:

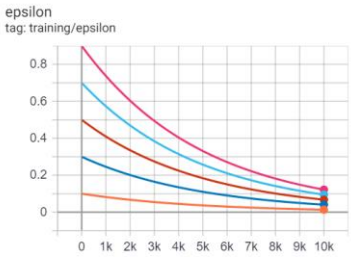
- פסאודו-קוד ויישום האלגוריתם:
לצורך יישום האלגוריתם נעזרתי בtutorial הבא:
https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html.
  - ארכיטקטורת רשת –
לאחר יישום של מספר ארכיטקטורות רשת, נמצא כי ארכיטקטורה של שכבה אחת לינארית מביאה לתוצאות מיטביות.
  - גודל state –
בחנתי את ביצועי האלגוריתם כאשר המידע המתקבל בכל צעד הוא ריבוע בגודל 3X3, ריבוע בגודל 5X5 וכל הלוח. נמצא כי ברוב המקרים כאשר הstate בגודל 5X5 התוצאות שמתקבלות הן מיטביות. לדוגמא, להלן גרף שמציג smoothing מלא של הreward כתלות בצעד (10,000 צעדים לאימון). האימון בוצע עם הערכים: gamma=0.3, epsilon=0.3. ניתן לראות כי הביצועים הטובים ביותר הן עבור state בגודל 5x5, לאחר מכן כל הלוח ולבסוף state בגודל 3x3. התוצאות היו עקביות עם נתוני הרצות שונים. אני משערת כי כאשר המצב בגודל 3x3 חסר מידע כדי שרשת תלמד בצורה טובה יותר מה הaction הרצוי הבא, וכאשר המצב הוא הלוח השלם יש יותר מדי מידע ויתכן שצריך יותר לולאות כדי להגיע להתכנסות לערך סופי.

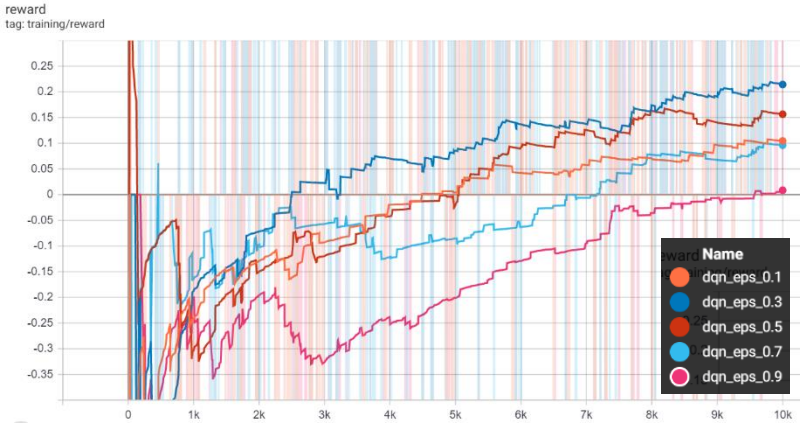


- השפעת Epsilon על ביצועי האלגוריתם:
Epsilon הוא המשתנה שמשפיע על כמה אחוז מהפעמים האלגוריתם יעשה exploitation (כלומר, יבחר ברגע הנתון את הצעד שהכי רווחי לו) לעומת exploration (כלומר יגריל צעד ובכך יכיר יותר מלוח המשחק ובצעדים עתידיים יוכל להרוויח יותר). במהלך הריצה, עושים decay לepsilon כך שבהתחלה הערך הוא המקסימלי ובסיום הריצה מופחת. למשל, אם נתחיל עם epsilon=0.3 אחרי 10,000 צעדים נסיים עם epsilon=0.05:



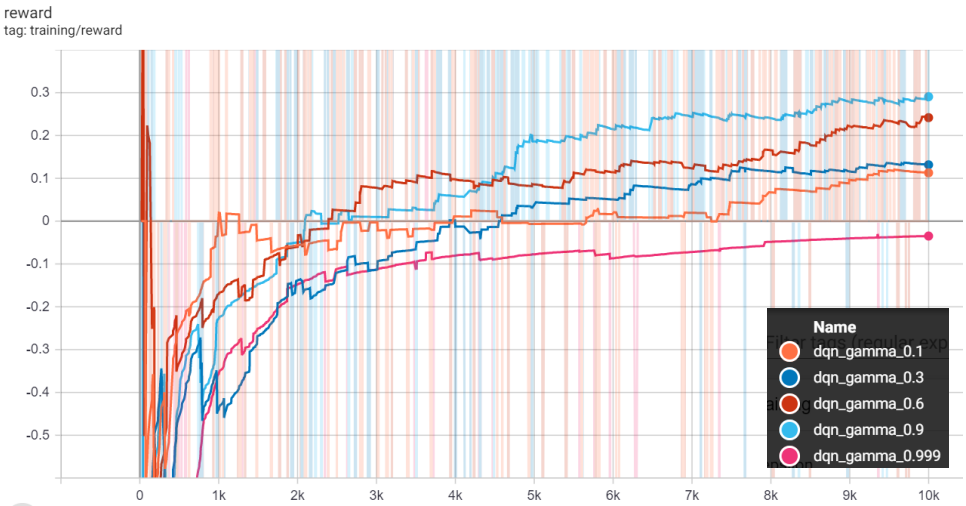
הסיבה לכך היא שבהתחלה האלגוריתם לא מכיר את לוח המשחק, אך בהמשך עדיף לעשות צעדים מחושבים ולא אקראיים.
כדי לבחון את השינוי של ביצועי האלגוריתם כתלות בערך epsilon ההתחלתי, הרצתי עבור ערכים התחלתיים של eps=0.1, 0.3, 0.5, 0.7, 0.9 ובחנתי את ההתכנסות של הreward לאחר 10,000 צעדים. כצפוי, ערכים epsilon גבוהים מדי הביאו לביצועים גרועים, שכן האלגוריתם מנחש הרבה אפילו לאחר שלמד טוב את המערכת. כמו כן, ערך epsilon של 0.1 הביא לביצועים בינוניים, כיוון שהאלגוריתם לא מנחש מספיק בשלבים ההתחלתיים, ובכך לא מכיר את הלוח היטב. הביצועים הכי טובים הם עבור epsilon=0.3 כפי שניתן לראות בגרף הבא:





- השפעת gamma על ביצועי האלגוריתם:

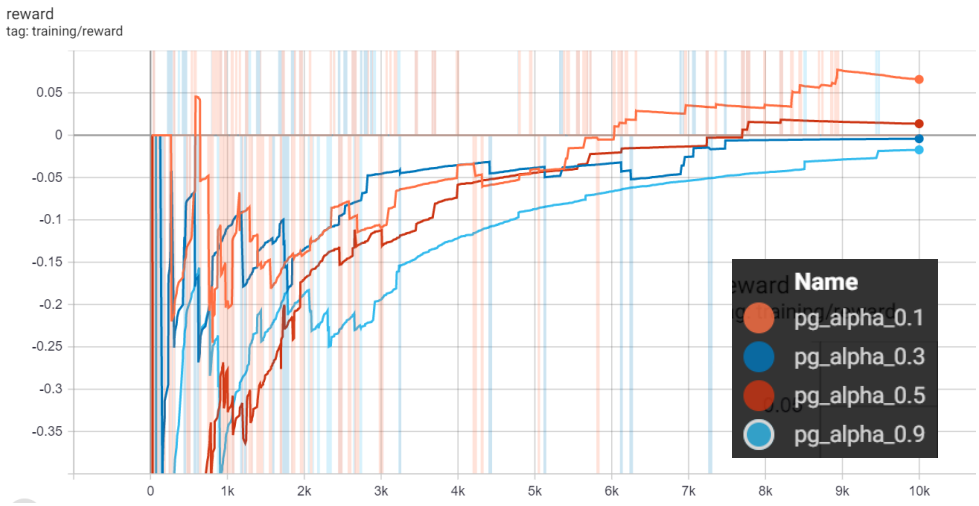
Gamma הוא משתנה שמשפיע על חשיבות הreward העתידי לאלגוריתם. כאשר gamma=1 , reward עתידי שווה ערך לreward נוכחי, ו - gamma=0 גורם לכך שהאלגוריתם יתחשב רק בreward של הצעד הנוכחי מבלי להתחשב בהשפעה העתידית שלו. כדי לבחון את השפעת המשתנה על הביצועים, הרצתי עבור הערכים: gamma=0.1, 0.3, 0.6, 0.9, 0.999 ועם ערך epsilon=0.3 התחלתי קבוע לכל ההרצות. ניתן לראות כי ככל שgamma גדול יותר, כך התוצאות שהתקבלו מיטביות יותר למעט עבור gamma=0.999. אפשר להסיק כי במשחק כמו snake יש חשיבות לתכנון לטווח ארוך (למשל כדי להגיע לפרי שנמצא בחצי השני של הלוח) ולכן כאשר יש פחות התחשבות בreward עתידי, יתקבלו תוצאות פחות טובות. אך עבור gamma=0.999 ניתן משקל עבור צעדים עתידיים מדי, ולכן יש פגיעה משמעותית בתוצאות האלגוריתם וה-reward המתקבל הוא שלילי.



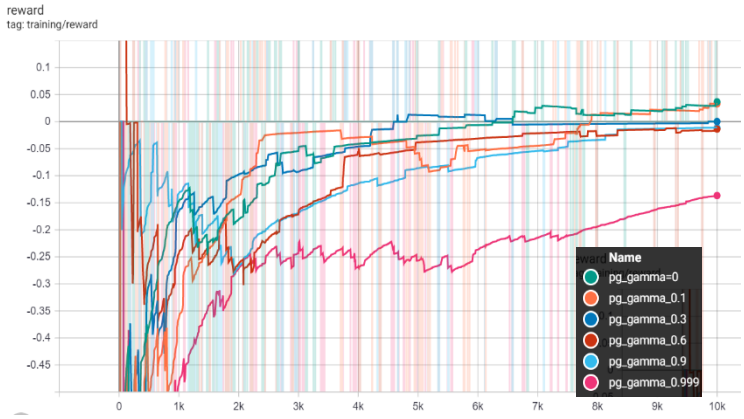
אלגוריתם Monte Carlo Policy Gradient

- פסאודו קוד ויישום האלגוריתם:
נעזרתי בtutorial הבא באופן חלקי לצורך מימוש האלגוריתם:
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63
האלגוריתם מומש בצורה זהה למה שהוצג בתרגול, לכן לא אפרט על הפסאודו קוד (כפי שנכתב בפורום התרגיל).
  - ארכיטקטורת רשת:
    הארכיטקטורה שנבחרה עבור הרשת היא שתי שכבות ReLU עם 250 ו-64 נוירונים בהתאמה, ושכבת softmax שמחזירה 3 פלטים (בהתאם לכמות הactions).
  - גודל state:
    בדקתי עבור אלגוריתם זה עבור איזה גודל state מתקבלות תוצאות מיטביות, ובשונה מdqn כאן נמצא כי גודל 3x3 הוא האידיאלי.

- השפעת מקדם האנתרופיה על ביצועי האלגוריתם:
את המקדם alpha של הentropy מימשתי כך שהוא מתקבל כערך התחלתי שדועך עם כל איטרציה כדי לאפשר למודל להתכנס, בדומה לepsilon. עבור הערכים הבאים:alpha=0.1, 0.3, 0.5, 0.7, 0.9 השוויתי בין הביצועים כאשר שאר הפרמטרים נשארים קבועים. הערך שהביא לביצועים המיטביים הוא alpha=0.1, והביצועים הכי פחות טובים עבור alpha=0.9, בשונה מDQN בו ערך epsilon=0.3 שמאפשר יותר exploration הביא לתוצאות טובות. בנוסף, בדקתי בנפרד את הביצועים ללא האנטרופיה (כלומר alpha=0) והם פחות טובים מalpha=0.1 אבל עדיפים על פני שאר ערכי alpha.



- השפעת gamma על ביצועי האלגוריתם:
בדומה לאלגוריתם הקודם, כדי לבדוק את השפעת הפרמטר gamma ביצעתי הרצה עבור כל אחד מהערכים: gamma=0, 0.1, 0.3, 0.6, 0.9, 0.999 ועם ערך alpha=0.1 התחלתי קבוע לכל ההרצות. ציפיתי שבדומה לאלגוריתם DQN, PG יגיב טוב לערכי gamma גבוהים, אבל בפועל הערך הכי גבוה התקבל עבור gamma=0 וgamma=0.1 (עם gamma=0 שהתכנס מהר יותר), ואילו שאר הערכים התכנסו לאזור הreward=0. הביצועים הגרועים ביותר, בדומה לDQN, היו עבור gamma=0.999.



השוואה בין האלגוריתמים:

DQN הורץ באופן הבא: epsilon=0.3, gamma=0.9 וה-state ניתן כריבוע 5x5 סביב ראש הנחש.
PG הורץ באופן הבא: alpha=0.1, gamma=0.1 וה-state ניתן כריבוע 3x3 סביב ראש הנחש.
הביצועים של DQN היו משמעותית יותר טובים, והוא הצליח להתכנס לreward של 0.3 בעוד שPG התכנס ל0.1. בנוסף, זמן ההתכנסות של אלגוריתם PG היה משמעותית יותר איטי מאשר DQN.
הסיבות לכך יכולות להיות קשורות בשיטת מונטה-קרלו שאיתה מעדכנים את הגרדיאנט בPG.
בשיטה זאת משערכים את הגרדיאנט ע"י דגימת נקודות מה שיכול להביא לרעש ולעדכון לא טוב של הגרדיאנט. ואכן, הייתה וריאנטיות רבה בתוצאות ההרצות של שיטת PG עם אותם הפרמטרים.