קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

Double Deep Q-Learning Network DDQN Space Invaders

גלעד מרקמן

קישור ל GitHub

מבנה השיעור

- בשיעור זה נדגים כיצד לבנות ולאמן סוכן AI באמצעות למידת חיזוק לשחק במשחק Space Invaders.
- נממש את האימון באמצעות שיפור של האלגוריתם DQN הנקרא DDQN.
- בנוסף נלמד לעשות שימוש בספריית WandB למעקב נוח אחר אימון רשתות נוירונים.
- הסבר על מימוש הגרפיקה והמשחק ללא הסוכנים החכמים, נעשה במסגרת הרצאה קודמת בנושא PyGane, הנמצאת באתר תכנות באינטנרט (קישור להרצאה).
- הקוד של הפרויקט ב GitHub.

Deep Q-learning (DQN) - חזרה

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

Initialize $Q(s,a,w)$ with arbitrary w.
Initialize $\hat{Q}(s,a,w^{-})$ with arbitrary w^{-}.
Initialize replay-buffer RB with size N.
For epoch in epochs (for each game):
- Initialize s
- While s is not Terminal:
  - Choose A From S using Q and e-greedy
  - Interact with environment and get s, a, r, s'
  - Store transition in RB
  - $s = s'$
  - sample random minibatch from RB
  - Calculate $Q(s,a,w)$ - forward
  - Finding a' for s' using $\hat{Q}(s,a,w^{-})$
  - Calculate loss with MSELoos :
    - If s' is Terminal: $loss = (r - Q(s,a,w))^2$
    - else: $loss = \left(r + \gamma \cdot \hat{Q}(s',a',w^{-}) - Q(s,a,w)\right)^2$
  - Calculate gradients (backwards): $loss.backward()$
  - Update W : $Optim.SGD.step()$
- Every C epochs $w^{-} \leftarrow w$

Deep Reinforcement Learning with Double Q-learning

Hado van Hasselt and **Arthur Guez** and **David Silver**
Google DeepMind

Abstract

The popular Q-learning algorithm is known to overestimate action values under certain conditions. It was not previously known whether, in practice, such overestimations are common, whether they harm performance, and whether they can generally be prevented. In this paper, we answer all these questions affirmatively. In particular, we first show that the recent DQN algorithm, which combines Q-learning with a deep neural network, suffers from substantial overestimations in some games in the Atari 2600 domain. We then show that the idea behind the Double Q-learning algorithm, which was introduced in a tabular setting, can be generalized to work with large-scale function approximation. We propose a specific adaptation to the DQN algorithm and show that the resulting algorithm not only reduces the observed overestimations, as hypothesized, but that this also leads to much better performance on several games.

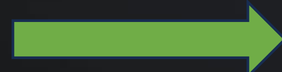
exploration technique (Kaelbling et al., 1996). If, however, the overestimations are not uniform and not concentrated at states about which we wish to learn more, then they might negatively affect the quality of the resulting policy. Thrun and Schwartz (1993) give specific examples in which this leads to suboptimal policies, even asymptotically.

To test whether overestimations occur in practice and at scale, we investigate the performance of the recent DQN algorithm (Mnih et al., 2015). DQN combines Q-learning with a flexible deep neural network and was tested on a varied and large set of deterministic Atari 2600 games, reaching human-level performance on many games. In some ways, this setting is a best-case scenario for Q-learning, because the deep neural network provides flexible function approximation with the potential for a low asymptotic approximation error, and the determinism of the environments prevents the harmful effects of noise. Perhaps surprisingly, we show

van Hasselt, Deep Reinforcement Learning with Double Q-learning

Double Deep Q-learning (DDQN)

Initialize Q(s,a,w) with arbitrary w.
Initialize $\hat{Q}$(s,a,w^-) with arbitrary w^-.
Initialize replay-buffer RB with size N.
For epoch in epochs (for each game):
  Initialize s
  While s is not Terminal:
    Choose A From S using Q and e-greedy
    Interact with environment and get s, a, r, s'
    Store transition in RB
    $s = s'$
    sample random minibatch from RB
    Calculate $Q(s,a,w)$ - forward
    Finding a' for s' using $\hat{Q}(s,a,w^-)$
    Calculate loss with MSELoos :
      If s' is Terminal: $loss = (r - Q(s,a,w))^2$
      else: $loss = \left(r + \gamma \cdot \hat{Q}(s',a',w^-) - Q(s,a,w)\right)^2$
    Calculate gradients (backwards): $loss.backward()$
    Update W : $Optim.SGD.step()$
  Every C epochs $w^- \leftarrow w$



Initialize Q(s,a,w) with arbitrary w.
Initialize $\hat{Q}$(s,a,w^-) with arbitrary w^-.
Initialize replay-buffer RB with size N.
For epoch in epochs (for each game):
  Initialize s
  While s is not Terminal:
    Choose A From S using Q and e-greedy
    Interact with environment and get s, a, r, s'
    Store transition in RB
    $s = s'$
    sample random minibatch from RB
    Calculate $Q(s,a,w)$ - forward
    Finding a' for s' using $Q(s,a,w)$
    Calculate loss with MSELoos :
      If s' is Terminal: $loss = (r - Q(s,a,w))^2$
      else: $loss = \left(r + \gamma \cdot \hat{Q}(s',a',w^-) - Q(s,a,w)\right)^2$
    Calculate gradients (backwards): $loss.backward()$
    Update W : $Optim.SGD.step()$
  Every C epochs $w^- \leftarrow w$

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

Space Invaders

- נדגים בניית סוכן RL באמצעות משחק Space Invaders.
- בהבדל מהדוגמה של Reversi המשחק אינו מבוסס על לוח משחק, אלא על אובייקטים גרפיים של pyGame.

- ה state מתקבל באמצעות פונקציה שנבנתה בסביבה, env.state() הבונה ומחזירה מערך דו-מימדי עם הקואורדינטות של השחקנים (ונתונים נוספים כגון המהירות).
- הגרפיקה וניהול המשחק נעשית במחלקת הסביבה.

מבנה התוכנית

- Action: מספר בין 0-3:
  - 0 – החללית לא עושה דבר - נשארת במקום.
  - 1 – החללית נעה שמאלה – במהירות SPACESHIP_SPEED = 5.
  - 2 – החללית נעה ימינה;
  - 3 – החללית מבצע ירי צרור מירבי של SPACE_SHIP_BURST=3.
- State: מערך של 88 פרמטרים, הכוללים:
  - 0-53: מיקום 18 חלליות האוייב. כל חללית 3 ערכים: x, y, speed_x. חללית הרוגה 0,0,0.
  - 54: מהירות אנכית של חללית האוייב spped_y.
  - 55-74: מיקום 10 טילי האוייב. כל טיל 2 ערכים: x,y.
  - 75: מהירות טיל אוייב על ציר Y. ENEMY_BULLET_SPEED=5.
  - 76-78: מיקום חללית השחקן, ומהירות החללית. SPACESHIP_SPEED = 5
  - 79-84: מיקום טילי החללית. כל טיל 2 ערכים x,y.
  - 85: מהירות טילי החללית SPACESHIP_BULLET_SPEED = 15.
  - 86: תחמושת שנותרה לחללית (מקסימום בתחילת שלב MAX_AMMUNITION = 80)
  - 87: ה level של המשחק.

מבנה התוכנית - המשך

- הסביבה – Environment כוללת את המאפיינים הבאים:
  - Bullets_Group – קבוצת טילי החללית.
  - Spaceship_group – קבוצה של חללית השחקן (קבוצה יחידה).
  - Enemy_bullets_Group – קבוצה של טילי האויבים.
  - Enemy_Group – קבוצת חלליות האוייב.
  - Ground_Group – קבוצה של הקרקע. הספרייט לא נראה ונעשה בו שימוש כדי לגלות נחיתה של חלליות האוייב.
  - Score – ניקוד של השחקן.
  - Level – רמת המשחק.

<u>קישור ל GitHub</u>

מחלקות התוכנית

- State: מערך הנוצר על ידי הסביבה ()environment.state.
- Game – לולאת המשחק הראשית.
- Human_Agent – סוכן אדם הקורא את המקלדת ומחזיר את הפעולה שבחר האדם.
- DQN_Agent – סוכן חכם AI
- DQN – רשת הנוירונים של הסוכן המחשבת את ערכי Q(s,a).
- Environment – הסביבה. המחלקה שמנהלת את המשחק והגרפיקה.
- SpaceShip – מחלקת ספרייט של החללית
- Enemy – מחלקת ספרייט של חללית אוייב.
- Bullet – מחלקת ספרייט של טיל. מתאים לטיל חללית וטיל אויב.
- Ground – מחלקת ספרייט של אדמה. הספרייט לא נראה ונוצר כדי לאתר נחיתה של חלליות האוייב בקצה התחתון של המסך.

הסביבה – מטפלת גם בגרפיקה

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

```python
import pygame ···


class Environment:
    def __init__(self, surface) -> None: ···

    def make_enemy_group (self, row=ENEMY_ROWS, col=ENEMY_COLS, space_row = 80, space_col = 120, speed = ENEMY_START_SPEED): ···

    def update (self): ···

    def draw (self): ···

    def restart (self, add_speed = 0, add_shoot_factor = 0, new_game = True): ···

    def move (self, action): ···

    def is_end_of_stage (self): ···

    def is_end_of_Game (self): ···

    def hits (self): ···

    def state (self): ···
```

מזיז את השחקנים במסך

מצייר את המסך בהתאם למיקום האובייקטים

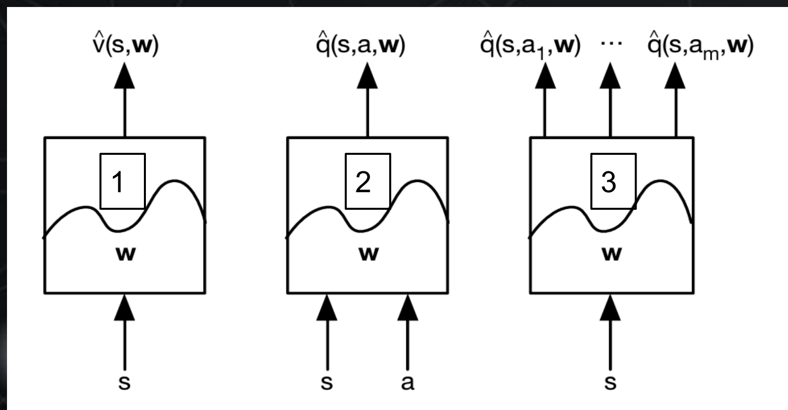
מבצע מהלך של הסוכן, מעדכן את הסביבה, מצייר ומחזיר -> reward, Done

בודק אם היתה פגיעה באויבים, "הורג" את החללית הנפגעת, ומחזיר מספר הפגיעות

מחזיר מערך חד מימדי בגודל 88 עם מיקום כל האובייקטים במשחק ונתונים נוספים

מבנה רשת הנוירונים

- בהרצאות הקודמות דיברנו על מספר אפשרויות לממש את טבלת ה Q באמצעות רשת נוירונים.
- במשחק Reversi יש מספר רב של פעולות (64 פעולות אפשריות), ויש פעולות חוקיות ולא חוקיות. על כן, החלטנו לממש את המשחק באמצעות רשת מס' 2.
- במשחק הנוכחי יש לנו רק 4 פעולות אפשריות וכולן חוקיות. על כן, הבחירה ברשת מס' 3 היא יעילה יותר.



קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

Class DQN

```python
# Parameters
input_size = 88 # Q(state) see environment for state shape
layer1 = 128
layer2 = 64
output_size = 4 # Q(state)-> 4 value of stay, left, right, shoot
gamma = 0.99

class DQN (nn.Module):
    def __init__(self, device = torch.device('cpu')) -> None:
        super().__init__()
        self.device = device
        self.linear1 = nn.Linear(input_size, layer1)
        self.linear2 = nn.Linear(layer1, layer2)
        self.output = nn.Linear(layer2, output_size)
        self.MSELoss = nn.MSELoss()

    def forward (self, x):
        x = self.linear1(x)
        x = F.leaky_relu(x)
        x = self.linear2(x)
        x = F.leaky_relu(x)
        x = self.output(x)
        return x

    def loss (self, Q_values, rewards, Q_next_Values, dones ):...

    def load_params(self, path):...

    def save_params(self, path):...

    def copy (self):...

    def __call__(self, states):...
```

- הקלט הוא בגודל ה state – 88 נוירונים.
- הפלט הוא בגודל ה action – 4 נוירונים.
- מבנה הרשת 88 -> 128 -> 64 -> 4.

- הפלט כולל מערך של 4 ערכים:
  - Output[0] – Q(s, a=0) נשאר במקום
  - Output[1] – Q(s, a=1) הסוכן נע שמאלה
  - Output[2] – Q(s, a=2) הסוכן נע ימינה
  - Output[3] – Q(s, a=3) הסוכן יורה

- על הפונקציה loss נפרט במסגרת האימון.

DQN_Agent

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

- הסוכן DQN_Agent נבנה בדומה לסוכן שבנינו למשחק Reversi.
  נפרט רק את ההבדלים והשדרוגים שעשינו בסוכן הנוכחי.

```
class DQN_Agent:
    def __init__(self, parametes_path = None, train = True, env= None, devive = torch.device('cpu')):...

    def setTrainMode (self):...

    def get_Action (self, state, epoch = 0, events= None, train = True) -> tuple:...

    def get_Actions_Values (self, states):...

    def Q (self, states, actions):...

    def epsilon_greedy(self,epoch, start = epsilon_start, final=epsilon_final, decay=epsiln_decay):...

    def loadModel (self, file):...

    def save_param (self, path):...

    def load_params (self, path):...

    def fix_update (self, dqn, tau=0.001):...

    def soft_update (self, dqn, tau=0.001):...

    def __call__(self, events= None, state=None):...
```

get_Action()

- הפונקציה מקבלת מצב ומחזירה את הפעולה של הסוכן.
- אנחנו מזינים לרשת את המצב (state) והפלט כולל את הערכים של 4 הפעולות האפשריות. מהם אנו בוחרים את הערך המקסימלי.
- אם הסוכן במצב אימון אזי אנחנו משתמשים במדיניות אפסילון-גרידי.

```
def get_Action (self, state, epoch = 0, events= None, train = True)
    actions = [0,1,2,3]
    if self.train and train:
        epsilon = self.epsilon_greedy(epoch)
        rnd = random.random()
        if rnd < epsilon:
            return random.choice(actions)

    with torch.no_grad():
        Q_values = self.DQN(state)
    max_index = torch.argmax(Q_values)
    return actions[max_index]
```

חישוב Q(state, action)

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

- הפונקציה מקבלת מספר זהה של מצבים (טנסור דו מימדי) ופעולות (טנסור חד מימדי) – ומחזירה את הערך של כל state, action.
- המצבים מוזנים לרשת שמחזירה לכל מצב 4 תשובות - הערך של כל אחת מהפעולות.
- הפונקציה מחזירה לכל מצב את הערך של הפעולה שמשוייכת לו לפי ה actions, לאו דווקא הפעולה הטובה ביותר.

```python
def Q (self, states, actions):
    Q_values = self.DQN(states)
    rows = torch.arange(Q_values.shape[0]).reshape(-1,1)
    cols = actions.reshape(-1,1)
    return Q_values[rows, cols]
```

Epsilon_greedy

- המדיניות אפסילון גרידי נועדה לטפל בדילמה של Exploration vs. Explotation.
- בתחילת המשחק נבחר מספר רב של מהלכים אקראיים, וזאת במשך מספר משחקים המוגדר ב decay. ככל שנתקדם במשחק ההסתברות לבחור פעולה אקראית קטנה.
- Start = 1; final = 0.01; decay = 5000
- בחרנו הפעם לממש מודל לינארי של אפסילון גרידי.

```
def epsilon_greedy(self,epoch, start = epsilon_start, final=epsilon_final, decay=epsiln_decay):
    # res = final + (start - final) * math.exp(-1 * epoch/decay)
    if epoch < decay:
        return start - (start - final) * epoch/decay
    return final
```

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

Get_Actions_Values

- פונקציה זו מקבלת batch של מצבים (טנסור דו מימדי) ומחשבת לכל מצב את הפעולה הכי טובה לפי רשת ה Q.
- הפונקציה מחזירה לכל מצב את הפעולה הכי טובה, ואת הערך מטבלת ה Q. הערכים מוחזרים בשני טנסורים:
  - טנסור של הפעולות לכל מצב.
  - טנסור של הערכים לכל מצב, פעולה.

```
def get_Actions_Values (self, states):
    with torch.no_grad():
        Q_values = self.DQN(states)
        max_values, max_indices = torch.max(Q_values,dim=1) # best_values, best_actions

    return max_indices.reshape(-1,1), max_values.reshape(-1,1)
```

עדכון של רשת המטרה

- באלגוריתם DQN או DDQN משתמשים בשתי רשתות:
  - הרשת הראשית – main Q. רשת המטרה – target Q.
- הרשת הראשית מאומנת ומתעדכנת בכל איטרציה. לעומת זאת, רשת המטרה נותרת קבועה, ומתעדכנת מידי מספר קבוע של משחקים C.
- קיימות שתי שיטות לעדכון הרשת: עדכון קבוע ועדכון "רך".
- בעדכון רך מעודכן כל פרמטר באופן חלקי (tau) לכיוון הפרמטר של הרשת הראשית.

```
def fix_update (self, dqn, tau=0.001):
    self.DQN.load_state_dict(dqn.state_dict())

def soft_update (self, dqn, tau=0.001):
    with torch.no_grad():
        for dqn_hat_param, dqn_param in zip(self.DQN.parameters(), dqn.parameters()):
            dqn_hat_param.data.copy_(tau * dqn_param.data + (1.0 - tau) * dqn_hat_param.data)
```

אימון הסוכן

קריית החינוך פארק המדע
בית לערכים
למצוינות ולחדשנות

- האימון דומה מאוד לאימון של רשת DQN עם שינוי אחד בלבד:
  - מציאת הפעולה למצב הבא a' נעשית באמצעות הרשת הראשית ולא רשת המטרה - DDQN.
- הבדלים נוספים נעוצים במבנה התוכנית שהיא מבוססת על סביבת pyGame.
- נעבור על התוכנית.
-

```
def main ():
    #region ################ init Surface of pyGame ##################...

    #region ################ params and models ######################...

    #region ################ checkpoint Load ########################...

    #region ################ Wandb.init #############################...

    #region ################ Main Loop ##############################
    for epoch in range(start_epoch, ephocs):
        #region ########### Episode loop - one game loop ############...

        #region ########### Update target network ###################...

        #region ########### Printing and saving #####################...

    #endregion
```

Init Surface of Pygame

- אנו נציג את מהלך האימון על המסך ונוכל לראות את הסוכן מתאמן.
- נאתחל את pygame ונצייר את המסך ההתחלתי.

```
def main ():
    #region ################# init Surface of pyGame ##################
    pygame.init()
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption('Space')
    # clock = pygame.time.Clock()
    header_surf = pygame.Surface((WIDTH, 100))
    main_surf = pygame.Surface((WIDTH, HEIGHT - 100))
    header_surf.fill(BLUE)
    main_surf.fill(LIGHTGRAY)
    env = Environment(surface=main_surf)
    screen.blit(header_surf, (0,0))
    screen.blit(main_surf, (0,100))
    write (header_surf, "Score: " + str(env.score) + " Ammunition: " + str(env.spaceship.ammunition))
    #endregion
```


קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

Params and models

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

```
#region ################ params and models ######################
best_score = 0
if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')

player = DQN_Agent(devive=device)
player_hat = DQN_Agent(device=device)
player_hat.DQN = player.DQN.copy()
batch_size = 128
buffer = ReplayBuffer(path=None)
learning_rate = 0.0001
ephocs = 200000
start_epoch = 0
C, tau = 3, 0.001
loss = torch.tensor(0)
avg = 0
scores, losses, avg_score = [], [], []
optim = torch.optim.Adam(player.DQN.parameters(), lr=learning_rate)
# scheduler = torch.optim.lr_scheduler.StepLR(optim,100000, gamma=0.50)
scheduler = torch.optim.lr_scheduler.MultiStepLR(optim,[5000*1000, 10000*1000,
                    15000*1000, 20000*1000, 25000*1000, 30000*1000], gamma=0.5)
step = 0
```

- נגדיר פרמטרים.
- נבנה סוכן DQN_Agent
- נבנה סוכן לשימוש ברשת המטרה player_hat.
- נבנה replay_buffer.
- נגדיר פרמטרים ואובייקטים נוספים כפי שעשינו באימון reversi.

checkpoint load

```python
#region ############### checkpoint Load ########################
num = 400
checkpoint_path = f"Data/checkpoint{num}.pth"
buffer_path = f"Data/buffer{num}.pth"
resume_wandb = False
if os.path.exists(checkpoint_path):
    resume_wandb = True
    checkpoint = torch.load(checkpoint_path)
    start_epoch = checkpoint['epoch']+1
    player.DQN.load_state_dict(checkpoint['model_state_dict'])
    player_hat.DQN.load_state_dict(checkpoint['model_state_dict'])
    optim.load_state_dict(checkpoint['optimizer_state_dict'])
    scheduler.load_state_dict(checkpoint['scheduler_state_dict'])
    buffer = torch.load(buffer_path)
    losses = checkpoint['loss']
    scores = checkpoint['scores']
    avg_score = checkpoint['avg_score']
player.DQN.train()
player_hat.DQN.eval()
#endregion
```

- נשמור נתונים על האימון בשני קבצים:
  - קובץ checkpoint.pth
  - קובץ buffer.pth
- השמירה נעשית במהלך לולאת האימון בהמשך.
- בקטע קוד זה אנחנו משחזרים נקודת עצירה להמשך אימון, אם הקבצים קיימים בדיסק.
- אנו טוענים את כל הנתונים הנדרשים להמשך אימון מהנקודה האחרונה בה שמרנו את הנתונים.

לולאת האימון

```
#region ################ Main Loop ##########################
for epoch in range(start_epoch, ephocs):

    #region ########### Episode loop - one game loop ########
    env.restart()
    end_of_game = False
    state = env.state()
    while not end_of_game:
        print (step, end='\r')
        step += 1
        #region ############# Play and Sample Environement ##

        #region ############# Train ################ ···

    #endregion

    #region ########### Update target network ###############

    #region ########### Printing and saving ################

#endregion
```

- לולאת האימון מתבצעת כמספר המשחקים epochs שאנו קובעים מראש.
- עבור כל משחק:
  - שלב א – דוגמים את הסביבה.
  - שלב ב- מאמנים את הרשת.
- בכל מספר קבוע של משחקים אנחנו מעדכנים את רשת המטרה.
- בכל מספר קבוע של משחקים אנחנו מדפיסים תוצאות האימון ושומרים checkpoint.

קריית החינוך פארק המדע
בית לערכים
למצוינות ולחדשנות

משחק ודגימת הסביבה

```python
#region ########### Episode loop - one game loop ############
env.restart()
end_of_game = False
state = env.state()
while not end_of_game:
    print (step, end='\r')
    step += 1
    #region ############# Play and Sample Environement ###
    main_surf.fill(LIGHTGRAY)
    header_surf.fill(BLUE)
    events = pygame.event.get()
    for event in events:
        if event.type == pygame.QUIT:
            return
    action = player.get_Action(state=state, epoch=epoch)
    reward, done = env.move(action=action)
    next_state = env.state()
    buffer.push(state, torch.tensor(action, dtype=torch.int64), torch.tensor(reward, dtype=torch.float32),
                next_state, torch.tensor(done, dtype=torch.float32))
    if done:
        best_score = max(best_score, env.score)
        break
    state = next_state

    write(header_surf,"Level: " + str(env.level), (200, 20))
    write(header_surf, "epoch: " + str (epoch), (400, 20))
    write(header_surf, "Score: " + str(env.score), (200, 60))
    write(header_surf, "Ammunition: " + str(env.spaceship.ammunition),(400, 60))
    screen.blit(header_surf, (0,0))
    screen.blit(main_surf, (0,100))
    pygame.display.update()
    # clock.tick(FPS)
```

- שלבי המשחק והדגימה:
  - אתחול לתחילת משחק.
  - טיפול בציור המסך וטיפול באירועים של pyGame.
  - בחירת צעד באמצעות הרשת הראשית.
  - ביצוע הצעד בסביבה וקבלת התגמול.

- יצירת המצב הבא.
- שמירה ב replay_buffer
- Push(s, a, r, s’, done)
- רענון מסך המשחק.

אימון הרשת DDQN

קריית החינוך פארק המדע
בית לערכים למצוינות ולחדשנות

- שליפת מספר דוגמאות צעדים מהחוצץ buffer.
- חישוב ערכי Q באמצעות הרשת הראשית.
- מציאת הפעולות הטובות ביותר של next_state לפי הרשת הראשית.
- חישוב ערכי $\hat{Q}\ (next_state, next_action)$ לפי רשת המטרה.
- חישוב הטעות loss לפי נוסחת בלמן (ראה בהמשך).
- עדכון הפרמטרים ברשת הראשית.

```
#region ############# Train ################
if len(buffer) < MIN_BUFFER:
    continue
states, actions, rewards, next_states, dones = buffer.sample(batch_size)
Q_values = player.Q(states, actions)
next_actions, _ = player.get_Actions_Values(next_states)
Q_hat_Values = player_hat.Q(next_states, next_actions)
loss = player.DQN.loss(Q_values, rewards, Q_hat_Values, dones)
loss.backward()
optim.step()
optim.zero_grad()
scheduler.step()
#endregion
```

חישוב הטעות loss

- חישוב הטעות נעשה בהתאם לנוסחת בלמן:

$$Q(s,a) = R + \gamma \cdot Q(s',a')$$

$$MSEloss = (Q(s,a) - [R + \gamma \cdot Q(s',a')])^2$$

- כאשר את a’ עבור s’ אנחנו מוצאים לפי המקסימום של טבלת Q:
  - DQN - טבלת Target Q
  - DDQN – טבלת Q
- ואת הערך Q(s’,a’) אנחנו מוצאים לפי טבלת Q_target.

```
def loss (self, Q_values, rewards, Q_next_Values, dones ):
    Q_new = rewards.to(self.device) + gamma * Q_next_Values * (1- dones.to(self.device))
    return self.MSELoss(Q_values, Q_new)
```

עדכון טבלת Q_target

- מידי תקופה אנחנו מעדכנים את טבלת Q_hat (Q-Target).
- עדכון אחיד - מידי C משחקים אנחנו משווים בין הרשתות fix_update.
- עדכון גמיש - אנחנו מעדכנים בתדירות גבוהה יותר, אך כל פעם בקצת לכיוון הרשת הראשית. העדכון נעשה בלולאה על כל משקל ומשקל של הרשת.

```
#region ########### Update target network ####################
if epoch % C == 0:
    player_hat.fix_update(dqn=player.DQN)
    # player_hat.soft_update(dqn=player.DQN, tau=tau)
    # player_hat.DQN.load_state_dict(player.DQN.state_dict())
#endregion
```

הדפסה ושמירה

- מידי מספר משחקים אנחנו מבצעים הדפסה ושמירה של הנתונים.
- את השמירה אנחנו מבצעים באמצעות מילון לקובץ הנקרא checkpoint.

```
if epoch % 1000 == 0 and epoch > 0:
    checkpoint = {
        'epoch': epoch,
        'model_state_dict': player.DQN.state_dict(),
        'optimizer_state_dict': optim.state_dict(),
        'scheduler_state_dict': scheduler.state_dict(),
        'loss': losses,
        'scores':scores,
        'avg_score': avg_score
    }
    torch.save(checkpoint, checkpoint_path)
    torch.save(buffer, buffer_path)
```

הספריה WandB

קריית החינוך פארק המדע
בית לערכים למצוינות ולחדשנות

- הספריה weights and Biases (wandb) מקנה קלים להקל באימון רשתות נוירונים.
- הספריה מאפשרת לנו, בצורה קלה, לשמור את הנתונים שלנו תוך כדי אימון בענן. להציג נתונים אילו באמצעות גרפים, ולצפות בהתקדמות האימון בכל עת.
- הספריה והשימוש באתר הוא ללא עלות.

- אתר WandB

שימוש בספריה wandb

- השימוש בספריה ובאתר הוא מאוד קל.
  - ראה הדרכה לתחילת עבודה באתר.

- שלבי העבודה:
  - פתיחת חשבון ללא עלות באתר.
  - התקנת הספריה בסביבת העבודה באמצעות pip install wandb.
  - כניסה לאתר וקישור המחשב לענן באמצעות פקודה בטרמינל wandb.login(API key).
  - את המפתח ניתן לקבל באתר לאחר הרישום.

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

איתחול פרויקט wandb

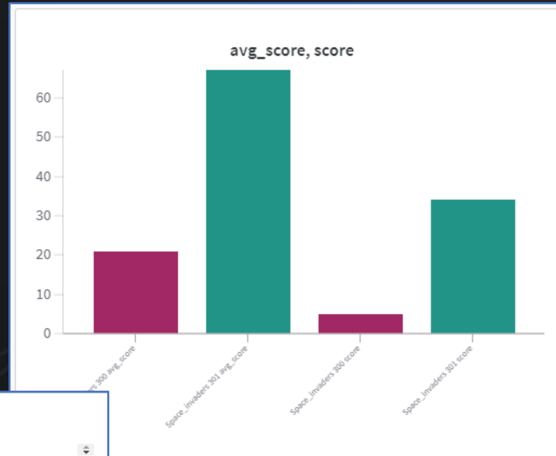

```
#region ################ Wandb.init #############################
wandb.init(
    # set the wandb project where this run will be logged
    project="Space_Invaders",
    resume=resume_wandb,
    id=f'Space_invaders {num}',
    # track hyperparameters and run metadata
    config={
    "name": f"Space_invaders DDQN {num}",
    "checkpoint": checkpoint_path,
    "learning_rate": learning_rate,
    "Schedule": f'{str(scheduler.milestones)} gamma={str(scheduler.gamma)}',
    "epochs": ephocs,
    "start_epoch": start_epoch,
    "decay": epsiln_decay,
    "gamma": 0.99,
    "batch_size": batch_size,
    "C": C,
    "tau":tau,
    "Model":str(player.DQN),
    "device": str(device)
    }
)
# wandb.config.update({"Model":str(player.DQN)}, allow_val_change=True)
```

- דוגמה לאיתחול הפרויקט.
- ניתן לשמור באתר מידע על כל ניסוי.
- אם resume=true הנתונים מעודכנים ומתווספים לניסוי קיים.

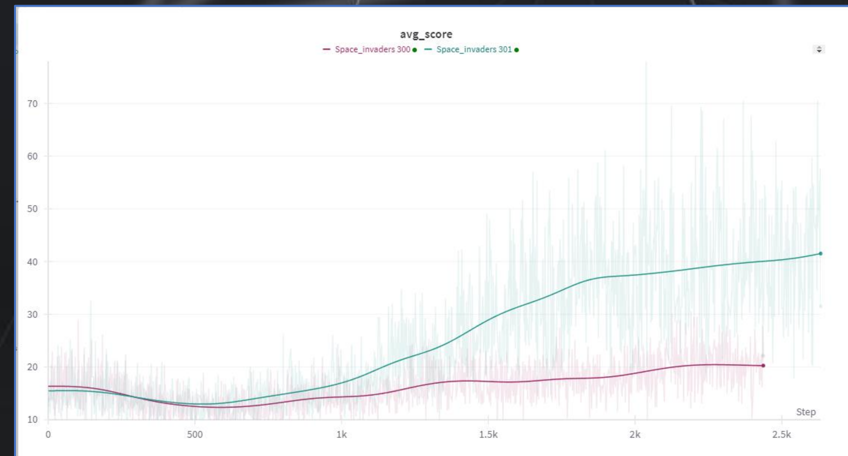
קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

שמירת נתונים ב wandb

- שמירת הנתונים נעשית באמצעות פקודת log פשוטה בתוכנה, ומשם הנתונים מועלים לאתר באופן אוטומטי וניתן להציג אותם כגרפים.




```python
if (epoch + 1) % 10 == 0:
    avg_score.append(avg)
    wandb.log ({
        "score": env.score,
        "loss": loss.item(),
        "avg_score": avg
    })
```



קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

אימון של הסוכן

Space

Level: 1
epoch: 24624
Score: 0
Ammunition: 78

- תוצאות אימון של הסוכן.
- האימון מבוצע תוך כדי משחק הגלוי לנו.