

**مینی پروژه سری چهارم**

**نام استاد: آقای دکتر علیاری**

**نام دانشجو: سمانه اعلایی**

**40102094**

**نام درس: یادگیری ماشین**

**لینک کولب:**

[**https://colab.research.google.com/drive/1VbYpjZVFyx1PcspnhQelpqFQ47hEL-25?usp=sharing**](https://colab.research.google.com/drive/1VbYpjZVFyx1PcspnhQelpqFQ47hEL-25?usp=sharing)

**لینک گیت هاب:**

[**https://github.com/samanehalaei**](https://github.com/samanehalaei)

# **آ. برای این مسئله یکبار به روش Q-learning و DQN-learning یک عاملی را طراحی کرده و آموزش دهید.**

**WUMPS:**

در محیط بازی جهان Wumpus، یک شبیه‌سازی از یک شبکه 4\*4 داریم که هر خانه می‌تواند خالی باشد، یک چاله، Wumpus یا طلا داشته باشد. این شبکه با استفاده از ماتریس 4\*4 پیاده‌سازی می‌شود که هر خانه از این ماتریس می‌تواند یکی از وضعیت‌های زیر را داشته باشد:

خالی: خانه‌ای که هیچ موجودیت خاصی در آن قرار ندارد.

چاله: خانه‌ای که ورود به آن موجب اتمام بازی و پاداش منفی 1000 می‌شود.

: Wumpusموجودیتی که اگر وارد محل آن شود، اگر Wumpus زنده باشد، اتمام بازی و پاداش منفی 1000 می‌شود.

طلا: خانه‌ای که هدف اصلی عامل در بازی است و ورود به آن پاداش مثبت 100 را به همراه دارد.

فضای عمل:

عامل می‌تواند اقدامات زیر را در این محیط انجام دهد:

حرکت به بالا، حرکت به پایین، حرکت به چپ، حرکت به راست، شلیک تیر در هر یک از چهار جهت (بالا، پایین، چپ، راست)

در این محیط، Wumpus با هر اقدام یک خانه به سمت چپ، راست، بالا یا پایین حرکت می‌کند.

فضای پاداش:

سیستم پاداش در این محیط به صورت زیر تعریف شده است:

+100برای رسیدن به طلا.

-1000برای افتادن در چاله یا خوردن شدن توسط Wumpus.

+50برای کشتن Wumpus پاداش اضافی.

-1برای هر حرکت انجام شده توسط عامل.

محیط:در این محیط، ابتدا یک شبکه 4\*4 با مکان‌های دلخواه برای Wumpus و طلا ایجاد می‌شود. وضعیت اولیه و وضعیت‌های ممکن پس از هر اقدام توسط عامل به دقت تعریف می‌شوند.

**بخش پیاده سازی محیط WUMPS**

|  |  |
| --- | --- |
| # Wumpus World Environment  class WumpusWorld:      def \_\_init\_\_(self):          self.size = 4          self.reset()      def reset(self):          self.grid = np.zeros((self.size, self.size))          self.agent\_pos = (0, 0)          self.gold\_pos = (3, 3)          self.wumpus\_pos = (1, 2)          self.pits = [(2, 2), (1, 1)]          self.grid[self.gold\_pos] = 1          self.grid[self.wumpus\_pos] = 2          for pit in self.pits:              self.grid[pit] = -1          self.has\_arrow = True          self.is\_wumpus\_alive = True          return self.\_get\_state()      def \_get\_state(self):          state = np.zeros((self.size, self.size))          state[self.agent\_pos] = 1          state[self.gold\_pos] = 2          if self.is\_wumpus\_alive:              state[self.wumpus\_pos] = 3          for pit in self.pits:              state[pit] = -1          return state.flatten()      def step(self, action):          x, y = self.agent\_pos          if action == 0:  # Move up              x = max(0, x - 1)          elif action == 1:  # Move down              x = min(self.size - 1, x + 1)          elif action == 2:  # Move left              y = max(0, y - 1)          elif action == 3:  # Move right              y = min(self.size - 1, y + 1)          elif action == 4:  # Shoot action              reward = self.shoot\_arrow()              return self.\_get\_state(), reward, False          reward = -1          done = False          if (x, y) == self.gold\_pos:              reward = 100              done = True          elif (x, y) in self.pits:              reward = -1000              done = True          elif (x, y) == self.wumpus\_pos:              if self.is\_wumpus\_alive:                  reward = -1000                  done = True          self.agent\_pos = (x, y)          return self.\_get\_state(), reward, done      def shoot\_arrow(self):          if not self.has\_arrow:              return -1          self.has\_arrow = False          # Determine if Wumpus is killed          ax, ay = self.agent\_pos          wx, wy = self.wumpus\_pos          if (ax == wx and abs(ay - wy) <= 1) or (ay == wy and abs(ax - wx) <= 1):              self.is\_wumpus\_alive = False              return 50          return -1  **CLASS WumpusWorld**  \_\_init\_\_ در این متد، اندازه محیط (بعد شبکه) تعیین می‌شود و با فراخوانی reset، محیط به حالت اولیه بازنشانده می‌شود.  reset این متد محیط را به حالت اولیه برمی‌گرداند. شبکه خالی می‌شود و عامل در موقعیت اولیه قرار می‌گیرد. طلا در مکان خود، Wumpus در مکان خود و چاله‌ها نیز در مکان‌های خود قرار می‌گیرند. وضعیت‌های دیگر محیط مانند داشتن تیر و زنده بودن Wumpus نیز با حالت موجودیت مشخص می‌شوند.  \_get\_state این متد وضعیت فعلی محیط را به صورت یک آرایه numpy با ابعاد مربعی مشخص شده توسط self.size برمی‌گرداند. در اینجا، عامل، طلا، Wumpus در صورت زنده بودن و چاله‌ها با مقادیر مشخص در شبکه نمایش داده می‌شوند.  step این متد نماینده گام‌های تکرار شده در محیط است. با گرفتن عملی (action) از عامل، موقعیت عامل بر اساس عمل انجام شده به‌روزرسانی می‌شود. اگر عامل به طلا برسد، پاداش 100 و انجام گذشته می‌شود. اگر به چاله برخورد کند، پاداش 1000- و انجام وجود دارد. اگر به مکان Wumpus برخورد کند و اگر Wumpus زنده باشد، پاداش 1000- و انجام می‌شود.   |  | | --- | |  |   **Q-LEARNING**  **معادله بلمن (Bellman Equation):**  معادله بلمن مفهومی بنیادی در یادگیری تقویتی است. این معادله، ارتباط بین کیفیت (مقدار Q) برای جفت حالت-عمل جاری، پاداش آنی (R)، و حداکثر پاداش آتی با اعمال تخفیف (γ) را توصیف می‌کند. این معادله به صورت زیر است:  در این معادله، (γ) ضریب تخفیف برای پاداش‌های آتی است. این معادله اساساً بیان می‌کند که مقدار Q بهینه برای یک جفت حالت-عمل خاص، مجموع پاداش آنی و حداکثر پاداش آتی با اعمال تخفیف است.  **فرایند به‌روزرسانی تکرارشونده (Iterative Update Process):**  اگرچه در ابتدا مقادیر Q بهینه را نداریم، می‌توانیم بر روی مقادیر فعلی تکرار (iterate) کنیم و با تعامل با محیط، به مقادیر Q بهینه همگرا شویم. این فرایند یادگیری تکرارشونده با معادله زیر نشان داده می‌شود:  در اینجا، (α) نرخ یادگیری است که نشان می‌دهد اطلاعات جدید چقدر بر اطلاعات قدیمی غالب می‌شود. عبارت داخل براکت مربعی، خطای اختلاف زمانی (TD) است که بیانگر اختلاف بین مقدار تخمین زده شده و پاداش واقعی دریافت شده است.  **نقش عامل تخفیف (γ) در یادگیری Q**  عامل تخفیف (با نماد γ نشان داده می‌شود) در یادگیری Q نقش مهمی در برقراری تعادل بین اهمیت پاداش‌های آنی و آتی ایفا می‌کند. عبارت بخش کلیدی از قانون به‌روزرسانی یادگیری Q است و به شرح زیر قابل تفکیک است:  **قانون به‌روزرسانی یادگیری Q**  قانون به‌روزرسانی یادگیری Q با معادله زیر نشان داده می‌شود:  در این معادله، هر عبارت نشان دهنده موارد زیر است:   * مقدار Q فعلی برای حالت s و عمل a. * α: نرخ یادگیری که کنترل می‌کند اطلاعات جدید چقدر بر اطلاعات قدیمی غلبه می‌کند. * r: پاداشی که پس از انجام عمل a در حالت s دریافت می‌شود. * γ: عامل تخفیف که نشان دهنده اهمیت پاداش‌های آتی است. * حداکثر مقدار Q پیش‌بینی شده برای حالت بعدی بر اساس تمام اعمال ممکن .   **نقش عامل تخفیف (γ)**  عامل تخفیف (γ) که عددی بین 0 و 1 است، تعیین می‌کند که پاداش‌های آتی در مقایسه با پاداش‌های آنی چقدر ارزش‌گذاری شوند:   1. **پاداش‌های آتی ( بلندمدت در مقابل کوتاه مدت )**  * وقتی γ به 1 نزدیک است، پاداش‌های آتی تقریباً به اندازه پاداش‌های آنی مهم تلقی می‌شوند. این عامل، یادگیرنده را تشویق می‌کند تا بلندمدت برنامه‌ریزی کند و تصمیماتی بگیرد که ممکن است فوراً سودمند نباشند، اما در نهایت منجر به نتایج کلی بهتری شوند. * وقتی γ به 0 نزدیک است، بر پاداش‌های آنی اولویت داده می‌شود و یادگیرنده دچار «نزدیک‌بینی» شده و تنها روی منافع کوتاه‌مدت تمرکز می‌کند.   **2. برقراری تعادل بین کاوش (Exploration) و بهره‌برداری (Exploitation)**   * مقدار γ بالاتر، کاوش در محیط را برای یافتن سیاست‌هایی که پاداش‌های بلندمدت بهتری ایجاد می‌کنند، تشویق می‌کند. * مقدار γ پایین‌تر تمایل به بهره‌برداری از پاداش‌های شناخته‌شده دارد و ممکن است استراتژی‌های بلندمدت بهتری را نادیده بگیرد.   **3. پایداری و همگرایی (Convergence)**  تنظیم مناسب γ برای پایداری و همگرایی الگوریتم یادگیری Q بسیار مهم است. اگر γ خیلی زیاد باشد، ممکن است منجر به برآورد بیش از حد پاداش‌های آتی و ایجاد ناپایداری شود. اگر خیلی کم باشد، ممکن است ارزش پاداش‌های آتی را دست‌کم بگیرد و منجر به سیاست‌های زیر بهینه شود. |

class Agent:

    def \_\_init\_\_(self, state\_size, action\_size, alpha, gamma, epsilon, epsilon\_decay, epsilon\_min):

        self.state\_size = state\_size

        self.action\_size = action\_size

        self.alpha = alpha

        self.gamma = gamma

        self.epsilon = epsilon

        self.epsilon\_decay = epsilon\_decay

        self.epsilon\_min = epsilon\_min

        self.q\_table = np.zeros((state\_size, action\_size))

    def choose\_action(self, state):

        if np.random.rand() <= self.epsilon:

            return np.random.choice(self.action\_size)

        return np.argmax(self.q\_table[state])

    def learn(self, num\_episodes):

        episode\_rewards = []

        consecutive\_successful\_episodes = 0

        first\_consistent\_episode = None

        for episode in range(1, num\_episodes + 1):

            state = env.reset()

            state\_idx = self.\_state\_to\_index(state)

            total\_reward = 0

            done = False

            steps = 0

            while not done:

                action = self.choose\_action(state\_idx)

                next\_state, reward, done = env.step(action)

                next\_state\_idx = self.\_state\_to\_index(next\_state)

                self.q\_table[state\_idx, action] = (1 - self.alpha) \* self.q\_table[state\_idx, action] + \

                    self.alpha \* (reward + self.gamma \* np.max(self.q\_table[next\_state\_idx]))

                state\_idx = next\_state\_idx

                total\_reward += reward

                steps += 1

            episode\_rewards.append(total\_reward)

            self.epsilon = max(self.epsilon\_min, self.epsilon \* self.epsilon\_decay)

            if total\_reward >= 100 and steps <= 5:

                consecutive\_successful\_episodes += 1

                if consecutive\_successful\_episodes >= 5 and first\_consistent\_episode is None:

                    first\_consistent\_episode = episode

            else:

                consecutive\_successful\_episodes = 0

            print(f"Q-learning Episode {episode}/{num\_episodes}, Total Reward: {total\_reward}")

        return episode\_rewards, first\_consistent\_episode

    def \_state\_to\_index(self, state):

        return hash(tuple(state)) % self.state\_size

مسیر به سمت جایزه را پیدا می‌کند. فضای اقدام (Action Space) شامل 4 گزینه و فضای مشاهده (Observation Space) شامل 16 خانه است که این 16 خانه می‌تواند محتوی جایزه یا موانع باشد. برای اجرای Q-learning ابتدا باید وضعیت فعلی (State) را مشخص کرده و سپس جدول Q را بررسی کنیم.

ما یک عامل (Agent) داریم که باید چند وظیفه انجام دهد. اولین وظیفه مقداردهی اولیه (Initialization) است که در آن محیط را ایجاد کرده و یک دیکشنری می‌سازیم که وضعیت‌ها (States)، اقدامات (Actions) و مقادیر (Values) را در آن ذخیره می‌کند.

تابع دیگری که داریم، تابع نمونه‌گیری (Sample) است. این تابع یک اقدام تصادفی (Random Action) را از میان اقدامات ممکن انتخاب و سپس اعمال می‌کند. ایجاد محیط به معنای قرار دادن عامل در محیط اولیه است.

پس از اینکه عامل در یک وضعیت (State) قرار گرفت، یک اقدام تصادفی انجام می‌دهد، به وضعیت دیگری می‌رود و یک جایزه دریافت می‌کند. خروجی تابع نمونه‌گیری شامل وضعیت قدیمی (s)، اقدام انجام شده (a)، پاداش دریافت شده (r) و وضعیت جدید () است.

عامل (Agent) یک اقدام بهینه (Best Action) دارد. تابع مربوط به عامل در هر مرحله یک حلقه (for) روی اقدامات اجرا می‌کند و بهترین اقدام را به ما می‌دهد. طبیعتاً ما می‌خواهیم که بهترین اقدام را بازگرداند.

تابع به‌روزرسانی (Update) مقادیر را می‌گیرد، شامل وضعیت فعلی (s)، اقدام انجام‌شده (a) و وضعیت جدید (s')، و مقدار Q(S,A) را به‌روزرسانی می‌کند. تابع اجرا (Play) یک اپیزود را اجرا می‌کند. ابتدا محیط را بازنشانی (Reset) می‌کند و عامل را به وضعیت شروع می‌برد. سپس بهترین اقدام را در وضعیت فعلی انتخاب می‌کند و آن را در محیط اعمال می‌کند تا زمانی که اپیزود به پایان برسد. یعنی وضعیت فعلی را به وضعیت جدید تغییر می‌دهد و اقدام را اجرا می‌کند.

محیط Wumps را در ابتدا نوشتیم. سپس محیط را Wumps را به‌عنوان نام محیط در اینجا تعریف می‌کنیم. ضریب تخفیف (Gamma) و نرخ یادگیری (Alpha) را تعریف می‌کنیم و تعداد اپیزودهای تست (Test Episode) را برابر 1000 در نظر می‌گیریم که همان تابع اجرا (Play) است.

این کلاس Agent است که شامل تابع init می‌باشد. این تابع محیط را ایجاد (Initialize) می‌کند، آن را بازنشانی (Reset) می‌کند و به وضعیت شروع می‌برد و یک دیکشنری ایجاد می‌کند که وضعیت‌ها، اقدامات و مقادیر Q را در آن ذخیره می‌کند.

یک تابع دیگر به نام sample environment وجود دارد که یک اقدام (action) را به‌صورت نمونه از محیط انتخاب می‌کند. وضعیت فعلی که در آن حضور داریم old\_state نام‌گذاری می‌شود. اقدامی که انتخاب کرده‌ایم در محیط اعمال می‌شود و خروجی‌هایی که دریافت می‌کنیم شامل new\_state، reward و is\_done هستند. new\_state وضعیتی است که بعد از اعمال اقدام به آن می‌رویم، reward پاداشی است که دریافت می‌کنیم و is\_done نشان می‌دهد که اپیزود تمام شده است یا خیر. اگر اپیزود تمام شده باشد مقدار true و در غیر این صورت مقدار false خواهد بود.

اگر مقدار is\_done true باشد، محیط بازنشانی می‌شود و state به وضعیت شروع بازمی‌گردد. اگر false باشد، به وضعیت new\_state می‌رویم. خروجی این تابع old\_state، action، reward و new\_state است.

تابع Best value action از بین اقدامات، بهترین اقدام را انتخاب می‌کند. ابتدا best\_value و best\_action را روی none تنظیم می‌کنیم چون مقدار آن‌ها را نمی‌دانیم.

برای اقدامات در محدوده self.environment.action\_space، اگر مقدار state و action در دیکشنری مقادیر (value) حضور نداشت، مقدار اولیه را صفر قرار می‌دهیم و action\_value را همون self.value(state, action) تنظیم می‌کنیم. اگر در شروع کار باشیم، این مقدار را صفر قرار می‌دهیم.

برای انتخاب بهترین اقدام (action)، اگر best\_value برابر None باشد یا اگر best\_value کمتر از action\_value باشد، مقدار best\_value برابر action\_value و best\_action برابر action قرار داده می‌شود. اگر در طول حلقه for مقدار بهتری از action\_value پیدا کنیم، آن را به‌روزرسانی کرده و best\_action را برابر action قرار می‌دهیم. در این حلقه، بهترین اقدام و بهترین مقدار به دست می‌آید و به عنوان خروجی بازگردانده می‌شود.

تابع بعدی، value\_update، این‌گونه عمل می‌کند: ابتدا در وضعیت فعلی یا همان next\_state، بهترین اقدام را انتخاب می‌کنیم که این کار همان است. بهترین اقدام به ما best\_value را برمی‌گرداند، یعنی بهترین a را انتخاب کرده و مقدار آن را به عنوان best\_value بازمی‌گرداند.

مقدار new\_value برابر با محاسبه می‌شود و مقدار old\_value باید به‌روزرسانی شود. old\_value همان self.value[(s, a)] است که وضعیت اولیه و اقدامی است که در آن وضعیت قرار داریمبرابر با خواهد بود و به این ترتیب مقدار s, a تغییر کرده و value s, a نیز به‌روزرسانی خواهد شد.

تابع دیگری به نام play\_episode وجود دارد که یک اپیزود را اجرا می‌کند. ابتدا total reward برابر صفر قرار داده می‌شود و وضعیت (state) بازنشانی شده و به وضعیت شروع بازمی‌گردد. سپس در آن وضعیت، بهترین اقدام از نظر Q\_VALUE انتخاب و اجرا می‌شود. new\_state و reward دریافت می‌شوند و پاداش (reward) مجدداً اضافه می‌شود. اگر به انتهای اپیزود رسیدیم، حلقه متوقف می‌شود (break) و اگر نرسیدیم، state برابر با new\_state قرار داده می‌شود. این کار ادامه پیدا می‌کند تا اپیزود تمام شود، یعنی is\_done برابر true شود. اپیزود زمانی تمام می‌شود که یا به چاله بیفتیم یا به جایزه برسیم.

درادامه envoroment را میسازیم یک object از کلاس agent میسازیم ، Best\_reward و itteration را صفر میگذاریم

در یک while میایم و اجرا میکنیم و s,a,r,next\_S را به عنوان خروجی میگرداند و با استفاده از اینها agent.value\_update را صدا میزنیم و با استفاده از معادله ای که توضیح دادیم q\_value را اپدیت میکنیم.

بعد ما یکtest\_episode داریم که میخواهیم reward را گزارش کنیم داخلی یک for test میکنم و reward را جمع میکنیم.test\_episode=20 در نظر میگیریم و reward را تقسیم بر test\_episode میکنیم و میگوید اگر reward بزرگتر از best\_reward باشد best\_reward را مساوی reward قرار بده و اگر reward>0.8 شد حل شده است، و به جایزه خواهیم رسید.

**اگر rand() < epsilon**

این به این معنی است که اگر یک عدد تصادفی کمتر از اپسیلون (یک عدد کوچک مثبت) باشد، در این صورت:

یک عمل تصادفی انجام دهید

عامل باید یک عمل تصادفی انجام دهد. این قسمت کاوش است که در آن عامل سعی می‌کند اعمال جدیدی را امتحان کند تا ببیند آیا این اعمال بهتر هستند یا نه.

. else

اگر عدد تصادفی کمتر از اپسیلون نباشد، در این صورت:

.بهترین عمل را بر اساس جدول Q انجام دهید (عمل حریصانه)

عامل باید عملی را انجام دهد که به نظر او بر اساس جدول Q بهترین است. این قسمت بهره‌برداری است که در آن عامل از آنچه آموخته است برای انجام بهترین عمل استفاده می‌کند.

**Epsilon Greedy**

این قانون به‌روزرسانی مقدار Q در روش Epsilon Greedy است. این شبیه به قانون به‌روزرسانی Q-Learning است، اما براساس پارامتر اپسیلون تعادل بین کاوش و بهره‌برداری را برقرار می‌کند.

**سیاست حریصانه**

max\_{a'} Q(s',a')

این نمایانگر بهترین عمل در حالت بعدی s' بر اساس جدول Q فعلی است. این به عنوان سیاست حریصانه شناخته می‌شود زیرا همیشه عملی را انجام می‌دهد که بالاترین پاداش تخمینی را دارد.

سیاست Epsilon Greedy یک روش برای تعادل بین کاوش و بهره‌برداری در یادگیری تقویتی است. با انجام گاه‌گاهی اعمال تصادفی، عامل می‌تواند استراتژی‌های جدیدی را کشف کند، اما بیشتر اوقات از دانش فعلی خود برای انجام بهترین عمل استفاده می‌کند.

**اهمیت اپسیلون (ε):**

اپسیلون یک عدد کوچک مثبت (معمولاً بین 0 و 1) است که تعیین می‌کند چقدر از وقت‌ها عامل باید کاوش کند (عمل تصادفی انجام دهد) و چقدر از وقت‌ها باید بهره‌برداری کند (بهترین عمل را انجام دهد). اگر اپسیلون بالا باشد، عامل بیشتر کاوش می‌کند و اگر پایین باشد، بیشتر بهره‌برداری می‌کند.

**کاوش (Exploration)**

کاوش به عامل کمک می‌کند تا با انجام اعمال جدید، اطلاعات بیشتری در مورد محیط خود به دست آورد. این می‌تواند به کشف استراتژی‌های جدیدی که ممکن است به پاداش‌های بیشتر منجر شوند کمک کند.

**بهره‌برداری (Exploitation)**

بهره‌برداری به عامل کمک می‌کند تا از دانشی که تا کنون کسب کرده است برای به دست آوردن پاداش‌های بیشتر استفاده کند. این معمولاً به معنی انتخاب عملی است که در جدول Q بالاترین مقدار را دارد.

**تعادل بین کاوش و بهره‌برداری:**

یک چالش اساسی در یادگیری تقویتی تعادل بین کاوش و بهره‌برداری است. سیاست Epsilon Greedy این تعادل را با استفاده از پارامتر اپسیلون برقرار می‌کند، به طوری که گاه‌گاهی عامل کاوش می‌کند و بیشتر اوقات بهره‌برداری می‌کند.

با استفاده از سیاست Epsilon Greedy، عامل می‌تواند به تدریج دانش خود را بهبود دهد و در نهایت به یک سیاست بهینه برای به دست آوردن بیشترین پاداش برسد.

**Reforcement learning**

یادگیری تقویتی یک شاخه از یادگیری ماشین است که بر پایه تعامل عامل با محیط عمل می‌کند تا بهترین رفتار یا تصمیمات را برای دستیابی به یک هدف مشخص پیدا کند. در این روش، یک عامل (agent) با محیط (environment) تعامل می‌کند و اقدامات (actions) مختلفی را انجام می‌دهد تا به اهداف خود برسد و پاداش (reward) مناسب را دریافت کند. این روند به طور مداوم و در هر لحظه، بر اساس عملکرد و پاداش‌های دریافتی، تصمیم‌گیری‌های بهینه را برای رسیدن به هدف نهایی تسهیل می‌دهد.

یکی از مفاهیم اصلی در یادگیری تقویتی، مسیر بهینه است که بهترین مسیر یا دنباله اقدامات را برای دستیابی به هدف نشان می‌دهد. در این روند، ارزیابی حرکت‌ها (actions) به عنوان خوب یا بد، بر اساس پاداش‌های دریافتی، انجام می‌شود. پاداش نشان‌دهنده عملکرد خوب یا بد عامل در محیط است و به عنوان انگیزه برای ادامه یا بهبود رفتار عامل عمل می‌کند.

در یادگیری تقویتی، هدف عملکرد (target) مشخصی وجود دارد که عامل باید به آن دست یابد. این هدف ممکن است برای عامل در ابتدا مشخص نباشد ولی با انجام تجربه و تعامل با محیط، عامل سعی می‌کند راه‌حل بهینه را برای رسیدن به این هدف پیدا کند.

به طور کلی، یادگیری تقویتی به وسیله انتخاب و اجرای اقدامات به طوری که بهترین نتیجه و پاداش را فراهم کند، به عامل امکان می‌دهد تا به طور خودکار و با تجربه، به دستیابی به اهداف خود برسد.

در یادگیری تقویتی، حرکت عامل (agent) بر اساس تجربه و تعامل با محیط (environment) صورت می‌گیرد. عامل با انجام اقدامات مختلف، سعی در بهینه‌سازی پاداش (reward) خود دارد. پاداش یک ورودی از محیط است که به عامل نشان می‌دهد که آیا اقدامات انجام شده منجر به افزایش یا کاهش میزان پاداش و بهره‌وری او شده است.

هر لحظه در یادگیری تقویتی بستگی به لحظه قبل دارد. این به این معنی است که تصمیم‌گیری‌های عامل بر اساس تجربه‌های گذشته و وضعیت فعلی محیط انجام می‌شود. عامل سعی می‌کند از تجربه‌های گذشته خود یاد بگیرد و رفتار بهینه را در آینده انتخاب کند.

(action) بعدی عامل بر اساس راهبردهایی که یادگرفته است و استراتژی‌هایی که می‌تواند امتحان کند، انتخاب می‌شود تا به پاداش مطلوب برسد. اکتشاف این ویژگی بر اساس برنامه‌ریزی عامل در محیط، تلاش می‌کند که پیش بینی کند که گام بعدی چه پاداشی خواهد داشت.

در یادگیری تقویتی، اصطلاح و به دو مفهوم اساسی اشاره دارد:

این مفهوم به مجموع اقداماتی که عامل در طول زمان انجام می‌دهد اشاره دارد. به عبارت دیگر، به مجموعه تمامی انتخاب‌ها یا اقداماتی که عامل انجام می‌دهد، اشاره دارد. این شامل تمام اقداماتی است که عامل برای بررسی محیط انجام می‌دهد، از جمله اقداماتی که می‌تواند پاداشی دریافت کند یا حتی اقداماتی که برای بررسی احتمالات یا آزمون راهبردهای مختلف انجام می‌دهد.

این مفهوم به مجموع وزن‌دار اقدامات مختلف اشاره دارد که عامل انجام می‌دهد. در برخی الگوریتم‌های یادگیری تقویتی، وزن‌های مختلف به اقدامات نسبت داده می‌شود تا عامل بهترین استراتژی را برای کسب حداکثر پاداش یاد بگیرد. به عبارت دیگر، نشان‌دهنده مجموع وزن‌هایی است که به اقدامات مختلف اختصاص یافته است و در انتخاب بهینه‌ترین اقدامات تاثیرگذار است.

در مدل‌های پویا برنامه‌ریزی مارکوف (MDP)، احتمال حرکت از یک حالت به حالت دیگر یک مفهوم اساسی است که رفتار عامل را تعیین می‌کند. در ابتدای مسئله، احتمالات برای حرکت از یک حالت به حالت دیگر معمولاً برابر است. این به این معنی است که عامل در ابتدا هیچ اطلاعات خاصی درباره محیط ندارد و به صورت تصادفی حرکت می‌کند.

با گذشت زمان و با تجربه بیشتر، عامل یاد می‌گیرد که برخی حرکات بهتر از بقیه هستند و منجر به پاداش بیشتری می‌شوند. بنابراین، احتمالات حرکت در برخی مکان‌ها افزایش می‌یابد. این افزایش احتمالات نتیجه یادگیری عامل از تجربه‌های گذشته و بهبود استراتژی‌های آن برای رسیدن به هدف است.

در نهایت، ممکن است مسیر عامل تغییر کند و یا دچار لغزش شود. این می‌تواند به دلیل تغییر در محیط، تغییر در هدف یا به‌روزرسانی استراتژی‌های عامل باشد. این لغزش در مسیر نشان‌دهنده نیاز عامل به تطبیق با شرایط جدید و به‌روز رسانی استراتژی‌ها برای رسیدن به بهترین نتیجه ممکن است.

**Learning rate**

نرخ یادگیری (learning rate) یک پارامتر مهم در الگوریتم‌های یادگیری ماشین است که مشخص می‌کند مدل با چه سرعتی باید پارامترهای خود را بر اساس خطا یا تفاوت بین پیش‌بینی و نتیجه واقعی به‌روزرسانی کند.

نرخ یادگیری به میزان تغییراتی که در وزن‌ها و پارامترهای مدل انجام می‌شود، اشاره دارد. اگر نرخ یادگیری بسیار بالا باشد، مدل ممکن است به سرعت تغییر کند و نتایج ناپایداری داشته باشد یا نتواند به یک راه‌حل بهینه برسد. از سوی دیگر، اگر نرخ یادگیری بسیار پایین باشد، فرآیند آموزش ممکن است بسیار کند باشد و مدل نتواند در زمان معقولی به نتایج مطلوب دست یابد.

تنظیم صحیح نرخ یادگیری بسیار مهم است. در بسیاری از موارد، از روش‌هایی مانند کاهش تدریجی نرخ یادگیری در طول زمان استفاده می‌شود تا در ابتدای آموزش تغییرات بزرگتری در پارامترها انجام شود و به مرور زمان این تغییرات کاهش یابد تا مدل به یک نقطه بهینه نزدیک شود.

در یادگیری تقویتی، نرخ یادگیری تعیین می‌کند که چقدر از اطلاعات جدید برای به‌روزرسانی ارزش‌های Q یا سیاست‌ها استفاده شود. فرمول به‌روزرسانی Q در یادگیری تقویتی به شکل زیر است:

در این فرمول، α نشان‌دهنده نرخ یادگیری است. این پارامتر تعیین می‌کند که چقدر از تفاوت بین ارزش پیش‌بینی شده و ارزش واقعی برای به‌روزرسانی Q استفاده شود.

نرخ یادگیری یک پارامتر کلیدی در الگوریتم‌های یادگیری ماشین است که تأثیر زیادی بر سرعت و دقت آموزش دارد. تنظیم مناسب این پارامتر می‌تواند به مدل کمک کند تا به سرعت به نتایج بهینه برسد و از نوسانات شدید جلوگیری کند.

**gamma**

در یادگیری تقویتی، گاما (gamma) یک پارامتر بسیار مهم است که به عنوان عامل تخفیف (discount factor) شناخته می‌شود. این پارامتر تعیین می‌کند که چقدر از ارزش پاداش‌های آینده در مقایسه با پاداش‌های فعلی مهم هستند.

مقدار گاما بین ۰ و ۱ قرار دارد. اگر مقدار گاما به ۱ نزدیک باشد، این به معنی اهمیت زیاد پاداش‌های آینده است و عامل سعی می‌کند اقدامات خود را طوری برنامه‌ریزی کند که بیشترین پاداش ممکن را در بلندمدت دریافت کند. از سوی دیگر، اگر مقدار گاما به ۰ نزدیک باشد، این نشان‌دهنده اهمیت بیشتر پاداش‌های فعلی است و عامل بیشتر به پاداش‌های کوتاه‌مدت توجه می‌کند.

به عنوان مثال، در فرمول به‌روزرسانی Q در یادگیری Q-learning، گاما نقش کلیدی دارد:

در این فرمول، گاما تعیین می‌کند که ارزش پاداش‌های آینده (max Q(s',a')) با چه ضریبی در به‌روزرسانی ارزش Q فعلی دخیل باشد.

به طور کلی، تنظیم مناسب مقدار گاما بستگی به مسئله خاص دارد. در مسائلی که پاداش‌های بلندمدت اهمیت بیشتری دارند، معمولاً از مقدار گامای بزرگتر استفاده می‌شود و در مسائلی که پاداش‌های کوتاه‌مدت مهم‌تر هستند، از مقدار گامای کوچکتر استفاده می‌شود.

Exploration and explotation

در الگوریتم Q-learning، اکتشاف (Exploration) و بهره‌برداری (Exploitation) دو جنبه کلیدی هستند که باید بین آن‌ها تعادل برقرار شود.

اکتشاف به معنای امتحان کردن اقدامات جدید و ناشناخته برای جمع‌آوری اطلاعات بیشتر درباره محیط است. این کار به عامل کمک می‌کند که:

از افتادن در دام انتخاب‌های محلی بهینه جلوگیری کند.

اطلاعات کاملی درباره تمامی اقدامات ممکن و نتایج آن‌ها کسب کند.

راه‌حل‌های جدید و احتمالا بهتری را کشف کند.

**بهره‌برداری (Exploitation)**

بهره‌برداری به معنای استفاده از اطلاعات فعلی برای انتخاب بهترین اقدام ممکن است. در این روش، عامل بر اساس دانش فعلی خود اقداماتی را انجام می‌دهد که بیشترین پاداش را به همراه دارند. بهره‌برداری به عامل کمک می‌کند که:

در کوتاه‌مدت پاداش‌های بیشتری کسب کند.

از دانش موجود به بهترین شکل ممکن استفاده کند.

**تعادل بین اکتشاف و بهره‌برداری**

در الگوریتم‌های یادگیری تقویتی، لازم است بین اکتشاف و بهره‌برداری تعادل برقرار شود. یکی از روش‌های رایج برای این کار، استفاده از سیاست epsilon-greedy است که به این صورت عمل می‌کند:

با احتمال ε (epsilon) عامل اکتشاف می‌کند و یک اقدام تصادفی انتخاب می‌کند.

- با احتمال عامل بهره‌برداری می‌کند و بهترین اقدام ممکن بر اساس جدول Q را انتخاب می‌کند.

در ابتدای فرآیند یادگیری، مقدار ε بالا تنظیم می‌شود تا عامل بیشتر اکتشاف کند و اطلاعات بیشتری کسب کند. به مرور زمان و با افزایش دانش عامل، مقدار ε کاهش می‌یابد تا عامل بیشتر بهره‌برداری کند و از دانش خود برای کسب پاداش‌های بیشتر استفاده کند.

|  |
| --- |
|  |

اکتشاف به معنای انجام اقدامات تصادفی یا کم‌تر شناخته شده است که می‌تواند به کشف اطلاعات جدید و یادگیری بهتر منجر شود. از آنجایی که در این حالت عامل ریسک می‌پذیرد، می‌توان از مقادیر کمتری برای k استفاده کرد. این کار باعث می‌شود احتمال انتخاب اقدامات با ارزش Q پایین‌تر افزایش یابد.

بهره‌برداری به معنای استفاده از اطلاعات فعلی و دانش به دست آمده است تا اقداماتی که به احتمال زیاد بیشترین پاداش را دارند را انتخاب کنیم. در این حالت، می‌توان از مقادیر بالاتری برای k استفاده کرد. این باعث می‌شود اقدامات با ارزش Q بالاتر انتخاب شوند و احتمال انتخاب اقدامات با ارزش Q پایین‌تر کاهش یابد.

**فرمول احتمال انتخاب اقدام**

فرمول معمول برای محاسبه احتمال انتخاب یک اقدام ai در حالت s به صورت زیر است:

p(ai|s) احتمال انتخاب اقدام ai در حالت s است.

Q(s,ai) ارزش Q برای اقدام ai در حالت s است.

kیک ثابت است که بسته به میزان اکتشاف یا بهره‌برداری می‌تواند مقادیر مختلفی داشته باشد. اگر بخواهیم اکتشاف کنیم، می‌توانیم k را کم و اگر بخواهیم بهره‌برداری کنیم، می‌توانیم k را بیشتر در نظر بگیریم.

DQN(DEEP Q-LEARNING)

در DQN زمانی به این روش روی می‌آوریم که نتوانیم یک جدول تشکیل دهیم و Q\_value را حساب کنیم، زیرا تعداد stateها بینهایت است و باید از تقریب استفاده کنیم. با استفاده از stateهایی که قبلاً دیده‌ایم، می‌توانیم به سایر stateها تعمیم دهیم و تقریب بزنیم. در بازی‌هایی مثل بعضی بازی ها که تعداد stateها زیاد است و ممکن است به stateهایی برخورد کنیم که قبلاً ندیده‌ایم، نیاز به یک تقریب‌زننده داریم که تابع q را تقریب بزند. شبکه‌های عصبی تقریب‌زننده‌های بسیار خوبی هستند که می‌توانند با استفاده از stateهایی که دیده‌اند، stateهایی را که ندیده‌اند تقریب بزنند.

هدف DQN تقریب زدن تابع Q است، بنابراین باید به دنبال یک تابع تقریب‌زننده باشیم که تابع x(t) که مشاهدات ما هستند را بگیرد و وزن‌ها و اکشن‌ها را در نظر بگیرد. تابع F(xt, w, a) باید تقریب Qhat(st, a) را بازگرداند، یعنی در یک state و با یک action بتواند با استفاده از یک تابع شبکه عصبی که xt, w, a را می‌گیرد، تقریب بزند. هدف ما این است که شبکه عصبی‌ای بسازیم که وقتی st را به آن می‌دهیم، Qhat(st, a) را به ما بدهد و یک تابع ارزش برای هر اکشن خاص ارائه دهد.

برای هر اکشن، ما می‌خواهیم Qhat را بازگردانیم. به این صورت که یک st می‌دهیم و شبکه عصبی آن را به تابع ارزش نگاشت می‌کند. اگر Q(st, a) برابر با rt + λ \* maxa’(Q(st+1, a’)) باشد، ما این را با یک شبکه عصبی تقریب می‌زنیم: Q(st, a) = rt + λ \* maxa’(Qhat(st+1, a’)). تابع F(xt, w, a) برابر با rt + λ \* maxa’(F(xt+1, w, a’)) خواهد بود و تابع هزینه (Loss Function) به صورت Lt = [rt + λ \* maxa’(F(xt+1, w, a’)) - F(xt, w, a)] تعریف می‌شود. هدف، کمینه کردن این تابع هزینه است.

در DQN از epsilon-greedy استفاده می‌کنیم، به این معنا که اکشنی را انتخاب می‌کنیم که بالاترین ارزش را دارد و stateی که بیشترین مقدار q را دارد استفاده می‌کنیم. اما ممکن است در بهینه محلی گیر کنیم، بنابراین به صورت تصادفی (رندوم) عمل می‌کنیم تا فضای جستجو را کامل بررسی کنیم. سپس به تدریج مقدار epsilon را کم می‌کنیم تا کمتر تصادفی عمل کنیم. در ابتدا فرآیند جستجو (exploration) است و سپس به بهره‌برداری (exploitation) می‌رسد تا به stateهایی برویم که ارزش‌های بالاتری دارند.

در DQN معمولاً از reply buffer استفاده می‌کنیم. در الگوریتم‌های gradient descent که مبتنی بر بهینه‌سازی محلی هستند و بهینه‌سازی گلوبال را در نظر نمی‌گیرند، مهم است که داده‌ها به صورت مستقل و با توزیع یکسان (IID) باشند، یعنی داده‌ها به هم ربطی نداشته باشند. به همین دلیل داده‌ها را shuffle می‌کنیم تا الگویی از رفتار داده به الگوریتم ندهیم. چیزی که بسیار مهم است، shuffle شدن داده‌ها توسط DQN است. اگر داده‌ها shuffle نشوند، الگوریتم در یک الگوی خاص گیر می‌کند و در بهینه محلی متوقف می‌شود.

ما در reply buffer تجربیات خود را ذخیره می‌کنیم. تجربه به معنای state، reward، next state و action است. این‌ها را ذخیره می‌کنیم و با یک اندازه‌ای مثلاً 10000 تجربه، هر بار 64 تا از آن‌ها را به صورت رندوم و به صورت minibatch انتخاب می‌کنیم. این 64 داده ممکن است از هر جایی از آن 10000 داده باشد تا خاصیت IID حفظ شود و در بهینه محلی گیر نکنیم و training پایداری داشته باشیم. اگر این کار را نکنیم، training ما ناپایدار خواهد شد.

تقریب‌زننده‌های F(xt, w, a) و f(xt+1, w, a') متفاوت هستند. چون برای پایدار کردن training است، ما نیاز به یک شبکه هدف (target network) و یک شبکه سیاست (policy network) داریم. شبکه هدف با استفاده از شبکه سیاست آپدیت می‌شود. به این صورت که: Wt\*(1-tau) + Wp\*tau = Wt. این یعنی شبکه هدف با استفاده از وزن‌های شبکه سیاست به صورت بسیار کند آپدیت می‌شود. این شبکه هدف باعث می‌شود که وزن‌ها پایدار باشند و training پایدار باشد.

با استفاده از device مشخص می‌کنیم که اگر روی GPU باشد به cuda برود و اگر روی CPU باشد روی همان اجرا شود.

در اینجا، replay memory به عنوان یک حافظه استفاده می‌شود که تجربیات ما را در آن ذخیره می‌کنیم. از یک transition به عنوان یک tuple استفاده می‌کنیم که هر زمان که نیاز است، اطلاعات خاصی از آن tuple را بازیابی کنیم و به عنوان state استفاده کنیم. replay memory توابع مختلفی دارد: تابع initialization که حافظه را برای ما مقداردهی اولیه می‌کند، تابع push که یک transition خاص را به حافظه اضافه می‌کند، تابع sample که یک batch از حافظه را به صورت رندوم انتخاب می‌کند (با اندازه batch\_size=64 مثلاً)، و len که تعداد transition های موجود در حافظه را نگه می‌دارد و در صورت نیاز، تجربه‌های قدیمی را حذف می‌کند، مثلاً اگر حافظه بیش از 10000 تجربه داشته باشد.

برای تابع هزینه (loss function) از Huber Loss استفاده می‌کنیم که مزایایی مانند دقت و عدم حساسیت زیاد به نویز را دارد. این تابع در صورتی که ارور کم باشد، مشابه Min Squared Error عمل می‌کند و در صورتی که ارور بزرگ باشد، مشابه Min Absolute Error عمل می‌کند.

Training DQN اغلب ناپایدار است و برای پایدارسازی آن، تدابیری اتخاذ می‌شود. در کلاس DQN، معمولاً از دو شبکه عصبی استفاده می‌شود: یک شبکه target و یک شبکه policy network. این شبکه‌ها به صورت linear هستند، با لایه‌هایی که ورودی‌ها را به صورت linear دریافت می‌کنند، سپس با استفاده از تابع relu فعال‌سازی می‌گذرند و در نهایت به یک لایه خطی دیگر می‌روند. تعداد 128 مشاهدات با observation ما به تعداد action ها داده می‌شود و خروجی ما اندازه action های موجود است، ما می‌خواهیم که Q را به تعداد action های خود تقریب بزنیم.

**پارامترهای آموزش**

Batchsize128

Gamma0.9

Epsilon\_start 0.9 شروع مقدار اپسیلون برای استراتژی ایپسیلون-گریدی

Eps\_end:0.05 پایان مقدار اپسیلون برای استراتژی ایپسیلون-گریدی

Lr:0.1نرخ یادگیری برای به‌روزرسانی وزن‌های شبکه

Epsilon\_decay : 1000 میزان کاهش تدریجی اپسیلون از 0.9 به 0.05

Tau: 0.05 نرخ به‌روزرسانی وزن‌های شبکه هدف به وزن‌های شبکه خود

**توضیح کد**

State, info = env.reset() بازنشانی محیط و بازگرداندن مشاهدات اولیه و اطلاعات.

len(state) تعداد مشاهدات (observation) در وضعیت کنونی.

Dqn(n\_observation, n\_action) مقدار دهی اولیه شبکه DQN با تعداد مشاهدات و تعداد اقدامات مشخص، انتقال به CUDA و همگام‌سازی وزن‌های شبکه خود و شبکه هدف. استفاده از بهینه‌ساز Adam

Memory = 10000اختصاص حافظه برای ذخیره تجربیات با ظرفیت ۱۰۰۰۰.

- Select\_actionپیاده‌سازی استراتژی ایپسیلون-گریدی برای انتخاب عمل. ابتدا به صورت تصادفی عمل می‌کند و سپس با گذر زمان به استفاده از شبکه خود (policy network) منتقل می‌شود. استفاده از `torch.no\_grad` برای جلوگیری از به‌روزرسانی غیرمنتظره پارامترهای شبکه در حین انتخاب عمل.

**حلقه آموزش به صورت زیر است:**

برای بهینه‌سازی از optimize model استفاده می‌شود.

اگر طول memory کمتر از batch size باشد، یک مقدار خالی برمی‌گردانیم.

در غیر این صورت، از حافظه یک batch size را به صورت تصادفی انتخاب کرده و آن را به عنوان transition ذخیره می‌کنیم.

transition را به صورت دسته‌ای (batch) به دست می‌آوریم که شامل وضعیت‌ها و اقدامات است.

- از تابع batch\_next\_state استفاده می‌کنیم تا وضعیت بعدی را برای دسته بگیریم. می‌دانیم که وقتی می‌خواهیم وضعیت را بگیریم، باید توجه داشته باشیم که وضعیت نهایی نباشد (s-is not none).

فقط وضعیت‌های غیر نهایی را از Batch.next state به دست می‌آوریم و آن‌ها را با non final mask ادغام می‌کنیم.

State batch action reward را کنار هم قرار می‌دهیم و به صورت دسته‌ای (batch) به policy می‌دهیم تا الگوریتم به‌روزرسانی را انجام دهیم.

تابع gather را صدا می‌زنیم تا از policy-next دو خروجی را بگیریم. اگر batch ۱۲۸ تایی باشد، دو عمل داریم و از آن‌ها انتخاب می‌کنیم.

State-action-value را به دست می‌آوریم و next-state-values را که با استفاده از torch.zeroes می‌سازیم و مقدار آن‌ها را برای batch size قرار می‌دهیم.

به target-net وضعیت‌های non final next state را می‌دهیم و ماکسیمم مقادیر را به دست می‌آوریم.

non-final-mask را به دست می‌آوریم و آن را به next state value اضافه کرده و سپس به روزرسانی می‌کنیم. سپس به expected state action value می‌رسیم و huber loss یا smoothlnlosss را به آن می‌دهیم.

مقدار state-action-value و expected-action-value (تارگت و چیزی که انتظار داریم) را به هم نزدیک می‌کنیم.

به optimize zero grad می‌دهیم و loss backward را به گرادیان می‌دهیم.

اگر اندازه بیشتر از ۱۰۰ باشد، کلیپ کنت گرادیان را اعمال می‌کنیم تا از انفجار گرادیان جلوگیری کنیم.

در مرحله آموزش، اگر cuda داشتیم، ۶۰۰ اپیزود را انتخاب می‌کنیم و اگر cpu داشتیم، ۵۰ اپیزود را انتخاب می‌کنیم. اول محیط را ریست می‌کنیم و به یک وضعیت شروعی می‌رویم و آن را به دستگاه (device) منتقل می‌کنیم و تعداد تجربیات را انتخاب می‌کنیم.

تابع select-action(state) که ابتدا به صورت تصادفی و سپس به صورت greedy عمل می‌کند را فراخوانی می‌کنیم.

عملی را به environment می‌دهیم و مشاهده‌ها، پاداش، terminate را به دست می‌آوریم

اگر terminate true باشد به معنای پایان است، در غیر این صورت به وضعیت بعدی می‌رویم و تجربه را به حافظه (memory) اضافه می‌کنیم.

اگر terminate false باشد، برای مشاهده وضعیت بعدی (next-state-observation)، policy را اعمال می‌کنیم که به معنای ادامه تلاش است و تجربه‌ای که به دست آورده‌ایم شامل وضعیت، اقدام، وضعیت بعدی و پاداش را به حافظه (memory) اضافه می‌کنیم.

وضعیت را برابر با next-state قرار می‌دهیم که برابر با مشاهده ماست و به optimize فراخوانی می‌دهیم تا روند به‌روزرسانی policy را انجام دهد.

وزن‌های تارگت نتورک (target-net) را از Target-net-state-dict بروزرسانی می‌کنیم تا وزن‌های جدید را بگیریم.

وزن‌های تارگت نتورک (policy-net) را از policy-net-state-dict بروزرسانی می‌کنیم تا وزن‌های policy را بگیریم.

با استفاده از tau ، وزن‌های تارگت نتورک (Target-net-state-dict) را بارگیری می‌کنیم تا تارگت‌های جدید شوند.

اگر به پایان اپیزود (done) رسیدیم، done را true قرار می‌دهیم و مدت زمان اپیزود را افزایش می‌دهیم، نتایج را برای ما نمودار می‌کنیم و متوقف نمی‌کنیم تا اپیزود جدید را شروع کنیم.

اولین عملیات، یک اقدام انجام می‌دهیم، می‌تواند تصادفی یا بر اساس policy باشد، از محیط نمونه برداری می‌کنیم و آن را در حافظه پاسخ (Reply-memory) قرار می‌دهیم، optimize را انجام داده و عملیات را به policy net منتقل می‌کنیم، سپس target-net را به روزرسانی می‌کنیم و این فرآیند تا آخر ادامه می‌یابد.

# Replay Memory

class ReplayMemory(object):

    def \_\_init\_\_(self, capacity):

        self.memory = deque([], maxlen=capacity)

    def push(self, \*args):

        """Save a transition"""

        self.memory.append(Transition(\*args))

    def sample(self, batch\_size):

        return random.sample(self.memory, batch\_size)

    def \_\_len\_\_(self):

        return len(self.memory)

# DQN Model

class DQN(nn.Module):

    def \_\_init\_\_(self, n\_observations, n\_actions):

        super(DQN, self).\_\_init\_\_()

        self.fc1 = nn.Linear(n\_observations, 128)

        self.fc2 = nn.Linear(128, 64)

        self.fc3 = nn.Linear(64, n\_actions)

    def forward(self, x):

        x = torch.relu(self.fc1(x))

        x = torch.relu(self.fc2(x))

        return self.fc3(x)

# DQNAgent

class DQNAgent:

    def \_\_init\_\_(self, env):

        self.env = env

        self.state\_size = env.size \* env.size

        self.action\_size = 5  # 4 movement actions + 1 shoot action

        self.memory = ReplayMemory(MEMORY\_CAPACITY)

        self.gamma = GAMMA

        self.epsilon = EPS\_START

        self.epsilon\_decay = EPS\_DECAY

        self.epsilon\_min = EPS\_END

        self.learning\_rate = LR

        self.policy\_net = DQN(self.state\_size, self.action\_size).to(device)

        self.target\_net = DQN(self.state\_size, self.action\_size).to(device)

        self.target\_net.load\_state\_dict(self.policy\_net.state\_dict())

        self.target\_net.eval()

        self.optimizer = optim.Adam(self.policy\_net.parameters(), lr=self.learning\_rate)

        self.steps\_done = 0

    def remember(self, state, action, reward, next\_state, done):

        self.memory.push(state, action, next\_state, reward, done)

    def select\_action(self, state):

        sample = random.random()

        eps\_threshold = EPS\_END + (EPS\_START - EPS\_END) \* np.exp(-1. \* self.steps\_done / EPS\_DECAY)

        self.steps\_done += 1

        if sample > eps\_threshold:

            with torch.no\_grad():

                return self.policy\_net(state).max(1)[1].view(1, 1)

        else:

            return torch.tensor([[random.randrange(self.action\_size)]], device=device, dtype=torch.long)

    def optimize\_model(self):

        if len(self.memory) < BATCH\_SIZE:

            return

        transitions = self.memory.sample(BATCH\_SIZE)

        batch = Transition(\*zip(\*transitions))

        non\_final\_mask = torch.tensor(tuple(map(lambda s: s is not None, batch.next\_state)), device=device, dtype=torch.bool)

        non\_final\_next\_states = torch.cat([s for s in batch.next\_state if s is not None]).view(-1, self.state\_size)

        state\_batch = torch.cat(batch.state).view(-1, self.state\_size)

        action\_batch = torch.cat(batch.action).view(-1, 1)

        reward\_batch = torch.cat(batch.reward)

        done\_batch = torch.tensor(batch.done, device=device, dtype=torch.bool)

        state\_action\_values = self.policy\_net(state\_batch).gather(1, action\_batch)

        next\_state\_values = torch.zeros(BATCH\_SIZE, device=device)

        with torch.no\_grad():

            next\_state\_values[non\_final\_mask] = self.target\_net(non\_final\_next\_states).max(1)[0]

        expected\_state\_action\_values = reward\_batch + (self.gamma \* next\_state\_values \* (~done\_batch))

        loss = nn.functional.smooth\_l1\_loss(state\_action\_values, expected\_state\_action\_values.unsqueeze(1))

        self.optimizer.zero\_grad()

        loss.backward()

        self.optimizer.step()

        # Update target network

        for target\_param, policy\_param in zip(self.target\_net.parameters(), self.policy\_net.parameters()):

            target\_param.data.copy\_(target\_param.data \* (1.0 - TAU) + policy\_param.data \* TAU)

    def learn(self, num\_episodes):

        episode\_rewards = []

        cumulative\_rewards = []

        for i\_episode in range(num\_episodes):

            state = self.env.reset()

            total\_reward = 0

            state = torch.tensor(state, dtype=torch.float32, device=device).unsqueeze(0)

            for t in count():

                action = self.select\_action(state)

                next\_state, reward, done = self.env.step(action.item())

                next\_state = torch.tensor(next\_state, dtype=torch.float32, device=device).unsqueeze(0)

                reward = torch.tensor([reward], device=device)

                self.remember(state, action, reward, next\_state, done)

                state = next\_state

                total\_reward += reward.item()

                self.optimize\_model()

                if done:

                    episode\_rewards.append(total\_reward)

                    print(f"Episode {i\_episode + 1} finished after {t + 1} steps with total reward {total\_reward}")

                    break

            cumulative\_rewards.append(np.sum(episode\_rewards))  # Accumulate total rewards

            if i\_episode % 10 == 0:

                self.target\_net.load\_state\_dict(self.policy\_net.state\_dict())

        return episode\_rewards, cumulative\_rewards

# Hyperparameters

BATCH\_SIZE = 64

GAMMA = 0.9

EPS\_START =1.0

EPS\_END = 0.05

EPS\_DECAY = 1000

TAU = 0.0001

LR = 0.1

MEMORY\_CAPACITY = 10000

**بررسی پارامتر ها اموزش و مقایسه آن ها**

در اینجا ما با یک محیط پیچیده در ارتباط هستیم زیرا شلیک تیر و کشتن WUMPS و وجود چاله ها و خورده شدن توسط WUMPS بر پیچیدگی محیط ما می افزاید. پیش بینی میکنم که عملکرد DQN که قادر است تابع Q را به صورت تقریبی یاد بگیرد که در محیط‌های پیچیده و با ابعاد بالا مانند Wumpus بسیار موثر است. همانطور که مشاهده میشود پاداش تجمعی در حال افزایش است یعنی عامل در حال یادگیری است و با افزایش اپیزود ها عماکرد ان بهتر هم می شود البته بستگی به Epsilon\_Decay هم دارد زیرا با افزایش ان باعث میشود خاصیت رندوم بود ان بیشتر شده و عمل رندوم تر بررسی کند. همانطور که مشاهده می شود در هنگامی که کاملا روی یک مقدار ماکسیمم reward تنظیم شده ممکن عامل اشتباه تصمیم بگیرد و موجب دریافت پاداش منفی بزرگ می شود مانند افتادن در چاله یا مواجهه با Wumpus در محیط بازی. در اینجا ما Epsilon\_Decay را یک عدد بین 500 تا 100 تعریف کردیم زیرا اگر epsilon\_decay خیلی بالا باشد، عامل ممکن است قبل از اینکه به اندازه کافی محیط را کاوش کرده باشد، به بهره‌برداری روی بیاورد. این ممکن است منجر به عملکرد ضعیف‌تری شود زیرا عامل هنوز به خوبی همه جوانب محیط را یاد نگرفته است. در مقابل، یک epsilon\_decay پایین‌تر به عامل زمان بیشتری برای اکتشاف می‌دهد که می‌تواند منجر به جمع‌آوری اطلاعات بهتر و در نهایت بهبود عملکرد شود.

**تطابق با داده‌های آموزش**:

وزن‌ها (یا پارامترهای شبکه) باید به گونه‌ای تنظیم شوند که شبکه بتواند داده‌های آموزشی را بهتر تطابق دهد و مدل به داده‌های جدید به درستی پاسخ دهد.

**کاهش خطا**

هدف اصلی آموزش شبکه‌های عمیق، کاهش خطا در پیش‌بینی خروجی‌های شبکه است. وزن‌ها در فرایند بهینه‌سازی (مثل گرادیان کاهشی) برای کمینه کردن تابع خطا (loss function) به کار می‌روند.

**منظم‌سازی (Regularization)**:

وزن‌ها می‌توانند نقش منظم‌سازی را ایفا کنند، به این معنی که با تنظیم وزن‌ها به گونه‌ای که مقادیر بزرگی نگیرند، از بیش‌برازش (overfitting) جلوگیری کنند.

**ساختار شبکه**:

وزن‌ها تعیین‌کننده‌ی ساختار شبکه هستند. مقادیر وزن‌ها تعیین می‌کنند که هر نورون چه مقدار از ورودی‌ها را چگونه ترکیب و به لایه بعد منتقل کند.

**پیچیدگی مدل**:

تنظیم درست وزن‌ها می‌تواند به افزایش پیچیدگی یا سادگی مدل کمک کند. وزن‌های مناسب باعث می‌شوند که مدل قادر به یادگیری الگوهای پیچیده‌تر باشد و همچنین از لحاظ محاسباتی موثرتر باشد.

**تاثیر مقدار TAU**

TAUبرای به روزرسانی تدریجی پارامترهای هدف (target network) از پارامترهای شبکه اصلی (policy network) استفاده می‌شود. این عمل به جلوگیری از نوسانات ناخواسته و بهبود استحکام آموزش کمک می‌کند.

مقدار کمتر TAU (مانند 0.001) منجر به به روزرسانی کندتر پارامترهای هدف می‌شود که می‌تواند کمک کند تا آموزش بهتری انجام شود ولی ممکن است به نتایج آموزش آهسته‌تری منجر شود.

مقدار بالاتر TAU (مانند 0.1) به روزرسانی سریع‌تر پارامترهای هدف را ایجاد می‌کند، اما ممکن است باعث نوسانات زیادی در آموزش شود و به پدیده‌هایی مانند بیش‌برازش (overfitting) منجر شود.

مقدار batch\_size در آموزش شبکه‌های عصبی، از جمله در الگوریتم‌های یادگیری تقویتی مانند DQN، تأثیرات متعددی دارد:

**تاثیر بر سرعت آموزش**:

انتخاب اندازه مناسب برای batch\_size می‌تواند به سرعت آموزش کمک کند. برای مثال، استفاده از batch\_size بزرگتر می‌تواند باعث بهره‌گیری بیشتر از قدرت محاسباتی GPU یا CPU شود و به دستیابی به آموزش سریع‌تر کمک کند.

**کاهش واریانس گرادیان**:

استفاده از batch\_size بزرگتر می‌تواند کمک کند تا واریانس گرادیان کاهش یابد که می‌تواند به بهبود پایداری آموزش و کاهش نوسانات در آموزش منجر شود.

**حافظه مصرفی**:

انتخاب batch\_size بزرگتر می‌تواند باعث افزایش حافظه مصرفی در طول آموزش شود. از آنجا که برای هر دوره آموزش، داده‌های بیشتری در حافظه نگه‌داری می‌شوند.

**برای DQN model1**

# DQN Model

class DQN(nn.Module):

    def \_\_init\_\_(self, n\_observations, n\_actions):

        super(DQN, self).\_\_init\_\_()

        self.fc1 = nn.Linear(n\_observations, 128)

        self.fc2 = nn.Linear(128, 64)

        self.fc3 = nn.Linear(64, n\_actions)

    def forward(self, x):

        x = torch.relu(self.fc1(x))

        x = torch.relu(self.fc2(x))

        return self.fc3(x)

**tau=0.001 and neuron model1**

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 1000

TAU = 0.001

LR = 0.0001

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -26.609

Episode 75 finished after 8 steps with total reward 144

Episode 76 finished after 7 steps with total reward 145

Episode 77 finished after 8 steps with total reward 144

Episode 78 finished after 7 steps with total reward 145

Episode 79 finished after 10 steps with total reward 142

Episode 80 finished after 7 steps with total reward 145

Episode 81 finished after 8 steps with total reward 144

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

کشتن Wumpus: عامل موفق به کشتن Wumpus شده و به پاداش حداکثری 145 دست یافته است. این نشان‌دهنده توانایی عامل در استفاده بهینه از پیکان و درک درست محیط است.

کاهش ریسک: در اپیزودهای پایانی، عامل به جای جستجوی پرخطر و کسب پاداش‌های بالا (مثل 145)، ریسک کمتری را متحمل شده و به طور مکرر پاداش 95 را کسب کرده است. این به دلیل امتیازات منفی است که در صورت مواجهه با Wumpus یا افتادن در چاله‌ها کسب می‌کرده است.

پاداش تجمعی: پاداش تجمعی نوسانات زیادی دارد و شکل W را به خود گرفته است. این نشان‌دهنده نوسانات عملکرد عامل در طی اپیزودها است که از تلاش‌های پر ریسک به سمت استراتژی‌های پایدارتر تغییر کرده است.

میانگین پاداش در هر اپیزود: این میانگین با الگویی مشابه تابع لگاریتمی افزایش یافته است که نشان‌دهنده یادگیری تدریجی عامل و بهبود عملکرد آن در طول زمان است.

سود و فایده دانستن این اطلاعات

ارزیابی عملکرد عامل: تحلیل میانگین پاداش و پاداش تجمعی به ما اجازه می‌دهد عملکرد عامل را ارزیابی کنیم و بفهمیم که آیا استراتژی‌های یادگیری بهینه بوده‌اند یا خیر.

تنظیم پارامترها: دانستن شکل نوسانات پاداش‌ها و الگوی لگاریتمی میانگین پاداش به ما کمک می‌کند پارامترهای مدل را تنظیم کنیم تا به نتایج بهتری برسیم.

مدیریت ریسک: تحلیل نتایج نشان می‌دهد که عامل در برخی اپیزودها ریسک کمتری را پذیرفته و به نتایج پایدارتری دست یافته است. این اطلاعات به ما کمک می‌کند در طراحی استراتژی‌های یادگیری بهتر ریسک را مدیریت کنیم.

پیش‌بینی عملکرد آینده: با تحلیل الگوهای عملکرد گذشته، می‌توانیم پیش‌بینی کنیم که عامل در آینده چگونه عمل خواهد کرد و آیا نیاز به تغییر استراتژی‌های یادگیری وجود دارد یا خیر.

**tau=0. 1 and neuron model1**

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 1000

TAU = 0.1

LR = 0.0001

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -100.553

Episode 992 finished after 6 steps with total reward 95

Episode 993 finished after 6 steps with total reward 95

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 7 steps with total reward 94

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

عدم کشتن Wumpus: عامل موفق به کشتن Wumpus نشده و به پاداش حداکثری (145) دست نیافته است. این نشان‌دهنده ضعف عامل در استفاده بهینه از پیکان و درک درست محیط است.

کاهش ریسک: عامل به جای جستجوی پرخطر و کسب پاداش‌های بالا، ریسک کمتری را متحمل شده و به طور مکرر پاداش 95 را کسب کرده است. این به دلیل امتیازات منفی است که در صورت مواجهه با Wumpus یا افتادن در چاله‌ها کسب می‌کرده است.

پاداش تجمعی: پاداش تجمعی نوسانات زیادی دارد و شکل W را به خود گرفته است. این نشان‌دهنده نوسانات عملکرد عامل در طی اپیزودها است که از تلاش‌های پر ریسک به سمت استراتژی‌های پایدارتر تغییر کرده است.

میانگین پاداش در هر اپیزود: این میانگین با الگویی مشابه تابع لگاریتمی افزایش یافته است که نشان‌دهنده یادگیری تدریجی عامل و بهبود عملکرد آن در طول زمان است.

**Epsilondecay=500**

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 500

TAU = 0.001

LR = 0.0001

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -103.213

Episode 821 finished after 7 steps with total reward 145

Episode 993 finished after 6 steps with total reward 95

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

بر اساس خروجی‌های نمایش داده شده، عامل DQN عملکرد زیر را داشته است:

پاداش میانگین در هر اپیزود: -103.213 (نشان‌دهنده وجود ریسک‌های بالاتر و مواجهه بیشتر با خطرات است)

اپیزود 821: عامل موفق به کسب پاداش حداکثری 145 شده است.

اپیزودهای آخر: عامل به صورت پایدار به کسب پاداش 95 پرداخته که نشان‌دهنده عدم موفقیت در کشتن Wumpus و جمع‌آوری طلا است.

عدم کشتن Wumpus: عامل موفق به کشتن Wumpus نشده و به پاداش حداکثری دست نیافته است. این نشان‌دهنده ضعف عامل در استفاده بهینه از پیکان و درک درست محیط است.

کاهش ریسک: عامل به جای جستجوی پرخطر و کسب پاداش‌های بالا، ریسک کمتری را متحمل شده و به طور مکرر پاداش 95 را کسب کرده است. این به دلیل امتیازات منفی است که در صورت مواجهه با Wumpus یا افتادن در چاله‌ها کسب می‌کرده است.

پاداش تجمعی: پاداش تجمعی نوسانات زیادی دارد و شکل W را به خود گرفته است. این نشان‌دهنده نوسانات عملکرد عامل در طی اپیزودها است که از تلاش‌های پر ریسک به سمت استراتژی‌های پایدارتر تغییر کرده است.

میانگین پاداش در هر اپیزود: این میانگین با الگویی مشابه تابع لگاریتمی افزایش یافته است که نشان‌دهنده یادگیری تدریجی عامل و بهبود عملکرد آن در طول زمان است.

تحلیل دقیق‌تر پاداش‌ها

افزایش و کاهش پاداش تجمعی بعد از 100 اپیزود: پاداش تجمعی در ابتدا افزایش یافته، سپس کاهش پیدا کرده و مجدداً در اپیزود 380 افزایش یافته است. این نوسانات نشان‌دهنده تغییرات استراتژی‌های عامل در طول زمان و تلاش برای بهبود عملکرد است.

میانگین پاداش به شکل لگاریتمی: میانگین پاداش در هر اپیزود به صورت تابع لگاریتمی افزایش یافته، که نشان‌دهنده بهبود عملکرد عامل در طول زمان است. در نزدیکی اپیزود 400 کاهش میانگین پاداش رخ داده و سپس مجدداً افزایش یافته است، که ممکن است به دلیل تغییرات در استراتژی‌های عامل باشد.

**EPS\_DECAY = 2000**

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 2000

TAU = 0.001

LR = 0.0001

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -80.411

Episode 988 finished after 6 steps with total reward 95

Episode 989 finished after 6 steps with total reward 95

Episode 990 finished after 6 steps with total reward 95

Episode 991 finished after 6 steps with total reward 95

Episode 992 finished after 6 steps with total reward 95

Episode 993 finished after 6 steps with total reward 95

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

عدم کشتن Wumpus: عامل موفق به کشتن Wumpus نشده و به پاداش حداکثری دست نیافته است. این نشان‌دهنده ضعف عامل در استفاده بهینه از پیکان و درک درست محیط است.

کاهش ریسک: عامل به جای جستجوی پرخطر و کسب پاداش‌های بالا، ریسک کمتری را متحمل شده و به طور مکرر پاداش 95 را کسب کرده است. این به دلیل امتیازات منفی است که در صورت مواجهه با Wumpus یا افتادن در چاله‌ها کسب می‌کرده است.

پاداش تجمعی: پاداش تجمعی نوساناتی داشته اما به شکل نرمی افزایش داشته و سپس کاهش یافته است. این نشان‌دهنده بهبود تدریجی عملکرد عامل در طول زمان و تغییر استراتژی‌ها برای بهبود عملکرد است.

میانگین پاداش در هر اپیزود: میانگین پاداش در هر اپیزود به صورت تابع لگاریتمی افزایش یافته است که نشان‌دهنده یادگیری تدریجی عامل و بهبود عملکرد آن در طول زمان است.

تحلیل دقیق‌تر پاداش‌ها

افزایش و کاهش پاداش تجمعی: پاداش تجمعی به طور نرمی افزایش یافته و سپس کاهش یافته است. این نوسانات نشان‌دهنده تغییرات استراتژی‌های عامل در طول زمان و تلاش برای بهبود عملکرد است.

میانگین پاداش به شکل لگاریتمی: میانگین پاداش در هر اپیزود به صورت تابع لگاریتمی افزایش یافته، که نشان‌دهنده بهبود عملکرد عامل در طول زمان است. این افزایش تدریجی میانگین پاداش نشان‌دهنده یادگیری و بهبود عملکرد عامل در مواجهه با محیط است.

**LR = 0.0001 ,Gamma=0.9**

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.9

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 2000

TAU = 0.001

LR = 0.0001

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -55.117

Episode 970 finished after 7 steps with total reward 145

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 7 steps with total reward 94

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

کشتن Wumpus: در برخی اپیزودها عامل موفق به کشتن Wumpus شده و به پاداش حداکثری 145 دست یافته است. این نشان‌دهنده یادگیری موفق عامل در بعضی از شرایط است.

کاهش ریسک: بعد از دستیابی به پاداش بالا، عامل به جای جستجوی پرخطر، ریسک کمتری را متحمل شده و به طور مکرر پاداش 95 را کسب کرده است. این به دلیل امتیازات منفی است که در صورت مواجهه با Wumpus یا افتادن در چاله‌ها کسب می‌کرده است.

پاداش تجمعی: پاداش تجمعی به شکل V به خود گرفته، ابتدا کاهش یافته و سپس افزایش داشته است. این نشان‌دهنده یادگیری تدریجی و تنظیم استراتژی‌های عامل در طول زمان است.

میانگین پاداش در هر اپیزود: میانگین پاداش در هر اپیزود به صورت تابع لگاریتمی افزایش یافته است که نشان‌دهنده بهبود عملکرد عامل در طول زمان است.

تحلیل دقیق‌تر پاداش‌ها

پاداش تجمعی: پاداش تجمعی نوسان داشته و به شکل V به خود گرفته است. ابتدا کاهش یافته و سپس افزایش یافته است. این نشان‌دهنده تلاش عامل برای پیدا کردن استراتژی‌های بهتر و تنظیم رفتارها برای بهبود عملکرد است.

میانگین پاداش به شکل لگاریتمی: میانگین پاداش در هر اپیزود به صورت تابع لگاریتمی افزایش یافته، که نشان‌دهنده یادگیری تدریجی و بهبود عملکرد عامل در طول زمان است. این بهبود تدریجی میانگین پاداش نشان‌دهنده یادگیری و بهبود عملکرد عامل در مواجهه با محیط است عامل DQN با نرخ یادگیری پایین‌تر (0.0001) و فاکتور تخفیف 0.9 نشان داده است که می‌تواند به نتایج بهتری دست یابد. با این حال، نیاز به بهبود استراتژی‌های یادگیری و مدیریت بهتر ریسک‌ها وجود دارد تا به پاداش‌های بالاتر دست یابد. تحلیل نتایج و تنظیم پارامترها می‌تواند به بهبود عملکرد عامل کمک کند.

پیشنهادات برای بهبود

افزایش نرخ یادگیری: افزایش نرخ یادگیری می‌تواند به عامل کمک کند تا سریع‌تر بهبود یابد و پاداش‌های بالاتری کسب کند.

تنظیم اپسیلون: کاهش اپسیلون به صورت تدریجی‌تر می‌تواند به عامل کمک کند تا به طور مداوم بهبود یابد و ریسک‌های بیشتری را بپذیرد.

تنظیم شبکه عصبی: افزایش تعداد لایه‌ها یا نورون‌ها در شبکه عصبی می‌تواند به عامل کمک کند تا پیچیدگی‌های بیشتری را درک کند و به عملکرد بهتری دست یابد..

**Gamma=0.9, alpha=0.01**

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.9

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 2000

TAU = 0.001

LR = 0.01

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -124.922

Episode 986 finished after 7 steps with total reward 145

Episode 987 finished after 7 steps with total reward 145

Episode 988 finished after 7 steps with total reward 145

Episode 989 finished after 7 steps with total reward 94

Episode 990 finished after 7 steps with total reward 145

Episode 991 finished after 7 steps with total reward 145

Episode 992 finished after 7 steps with total reward 145

Episode 993 finished after 18 steps with total reward 83

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 7 steps with total reward 145

Episode 996 finished after 7 steps with total reward 145

Episode 997 finished after 7 steps with total reward 145

Episode 998 finished after 9 steps with total reward 143

Episode 999 finished after 7 steps with total reward 145

Episode 1000 finished after 37 steps with total reward 64

کشتن Wumpus: عامل در اپیزودهای مختلفی موفق به کشتن Wumpus شده و به پاداش حداکثری 145 دست یافته است. این نشان‌دهنده یادگیری موفق عامل در برخی اپیزودها است.

کاهش ریسک: عامل پس از دستیابی به پاداش بالا، به جای جستجوی پرخطر، ریسک کمتری را متحمل شده و به پاداش‌های پایدارتر مانند 95 و 145 دست یافته است.

پاداش تجمعی: پاداش تجمعی نوسان دارد و به شکل نرم کاهش و سپس افزایش داشته است. این نشان‌دهنده تنظیم استراتژی‌های عامل برای بهبود عملکرد در طول زمان است.

میانگین پاداش در هر اپیزود: میانگین پاداش در هر اپیزود به صورت تابع لگاریتمی x افزایش یافته، که نشان‌دهنده یادگیری تدریجی عامل و بهبود عملکرد آن در طول زمان است.

تحلیل دقیق‌تر پاداش‌ها

پاداش تجمعی: پاداش تجمعی نوسان دارد، ابتدا کاهش یافته و سپس افزایش یافته است. این نشان‌دهنده تلاش عامل برای یافتن استراتژی‌های بهتر و تنظیم رفتارهای خود برای بهبود عملکرد است.

میانگین پاداش به شکل لگاریتمی: میانگین پاداش در هر اپیزود به صورت تابع لگاریتمی x افزایش یافته است. این نشان‌دهنده یادگیری تدریجی و بهبود عملکرد عامل در طول زمان است.

پیشنهادات برای بهبود

افزایش نرخ یادگیری: ممکن است افزایش نرخ یادگیری به عامل کمک کند تا سریع‌تر بهبود یابد و پاداش‌های بالاتری کسب کند.

تنظیم اپسیلون: کاهش اپسیلون به صورت تدریجی‌تر می‌تواند به عامل کمک کند تا به طور مداوم بهبود یابد و ریسک‌های بیشتری را بپذیرد.

تنظیم شبکه عصبی: افزایش تعداد لایه‌ها یا نورون‌ها در شبکه عصبی می‌تواند به عامل کمک کند تا پیچیدگی‌های بیشتری را درک کند و به عملکرد بهتری دست یابد..

**LR = 0.1, EPS\_DECAY=100**

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.9

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 100

TAU = 0.001

LR = 0.1

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -282.023

Episode 305 finished after 9 steps with total reward 143

Episode 993 finished after 9 steps with total reward 92

Episode 994 finished after 14 steps with total reward 87

Episode 995 finished after 8 steps with total reward 93

Episode 996 finished after 10 steps with total reward 91

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 8 steps with total reward 93

Episode 999 finished after 10 steps with total reward 91

Episode 1000 finished after 8 steps with total reward 93

کشتن Wumpus: عامل در برخی اپیزودها موفق به کشتن Wumpus شده و به پاداش‌های بالای 143 دست یافته است. این نشان‌دهنده توانایی عامل در دستیابی به پاداش حداکثری در برخی اپیزودها است.

کاهش ریسک: عامل در ادامه اپیزودها ریسک کمتری را پذیرفته و به پاداش‌های پایدارتر مانند 87 تا 95 دست یافته است. این نشان می‌دهد که عامل استراتژی کم‌ریسک‌تری را دنبال کرده تا از امتیاز منفی جلوگیری کند.

پاداش تجمعی: پاداش تجمعی نوسان زیادی داشته و به شدت کاهش یافته است. این نشان‌دهنده ناپایداری عملکرد عامل در طول زمان است.

میانگین پاداش در هر اپیزود: میانگین پاداش ابتدا افزایش و سپس کاهش و بعد به صورت کمی افزایش داشته است. این نشان‌دهنده تغییرات استراتژی و عملکرد عامل در طول زمان است.

تحلیل دقیق‌تر پاداش‌ها

پاداش تجمعی: پاداش تجمعی نوسان داشته و همگرا نشده است. این نشان می‌دهد که عامل در طول زمان نتوانسته است به یک استراتژی پایدار و بهینه دست یابد.

میانگین پاداش به صورت تابع لگاریتمی: میانگین پاداش ابتدا افزایش یافته، سپس کاهش یافته و بعد کمی افزایش یافته است. این نشان‌دهنده تلاش عامل برای یافتن استراتژی‌های بهتر و تنظیم رفتارهای خود برای بهبود عملکرد است.

عامل DQN در برخی اپیزودها به نتایج مناسبی دست یافته است، اما عملکرد کلی آن ناپایدار بوده و به یک استراتژی پایدار و بهینه دست نیافته است. تنظیم پارامترها و بهبود استراتژی‌های یادگیری می‌تواند به بهبود عملکرد عامل کمک کند.

پیشنهادات برای بهبود

افزایش نرخ یادگیری: ممکن است افزایش نرخ یادگیری به عامل کمک کند تا سریع‌تر بهبود یابد و پاداش‌های بالاتری کسب کند.

تنظیم اپسیلون: کاهش اپسیلون به صورت تدریجی‌تر می‌تواند به عامل کمک کند تا به طور مداوم بهبود یابد و ریسک‌های بیشتری را بپذیرد.

تنظیم شبکه عصبی: افزایش تعداد لایه‌ها یا نورون‌ها در شبکه عصبی می‌تواند به عامل کمک کند تا پیچیدگی‌های بیشتری را درک کند و به عملکرد بهتری دست یابد.

استفاده از تکنیک‌های پیشرفته‌تر: تکنیک‌های پیشرفته‌تری مانند Double DQN یا Prioritized Experience Replay می‌توانند به بهبود عملکرد عامل کمک کنند.

**LR = 0.1, EPS\_DECAY = 1000** # Hyperparameters

BATCH\_SIZE = 64

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 1000

TAU = 0.001

LR = 0.1

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -378.034

Episode 201 finished after 8 steps with total reward 144

Episode 988 finished after 7 steps with total reward 94

Episode 989 finished after 8 steps with total reward 93

Episode 990 finished after 9 steps with total reward 92

Episode 991 finished after 14 steps with total reward 87

Episode 992 finished after 8 steps with total reward 93

Episode 993 finished after 2 steps with total reward -1001

Episode 994 finished after 9 steps with total reward 92

Episode 995 finished after 7 steps with total reward 94

Episode 996 finished after 7 steps with total reward 94

Episode 997 finished after 10 steps with total reward 91

Episode 998 finished after 16 steps with total reward 85

Episode 999 finished after 4 steps with total reward -1003

Episode 1000 finished after 12 steps with total reward 89

کشتن Wumpus

عامل در برخی اپیزودها موفق به کشتن Wumpus شده است، اما به بالاترین پاداش ممکن دست نیافته است. برای مثال، در اپیزود 201 با 8 قدم به پاداش 144 رسیده است.

عامل پر ریسک:

در ادامه، عامل ریسک‌های زیادی را نپذیرفته و پاداش‌های منفی زیادی دریافت کرده است. این نشان می‌دهد که عامل به جای پیگیری استراتژی‌های پرریسک، به دنبال کاهش ریسک بوده است.

پاداش تجمعی:

پاداش تجمعی نوسان زیادی داشته و کاهش شدیدی را تجربه کرده است. این نشان‌دهنده ناپایداری عملکرد عامل در طول زمان است.

میانگین پاداش در هر اپیزود:

میانگین پاداش ابتدا افزایش یافته، سپس به صورت کمی ثابت شده و با یک افزایش نرم همراه بوده است. این نشان‌دهنده تغییرات استراتژی و تلاش‌های عامل برای بهبود عملکرد است.

تحلیل دقیق‌تر پاداش‌ها

نوسانات پاداش‌ها:

پاداش‌ها بر حسب اپیزود تغییرات زیادی داشته و به طور مداوم کم و زیاد شده‌اند. این نشان‌دهنده ناپایداری در تصمیم‌گیری و یادگیری عامل است.

اپیزودهای 993 و 999 با پاداش‌های منفی شدید (-1001 و -1003) نشان می‌دهند که عامل در برخی مواقع به دلیل پذیرش ریسک‌های نامناسب، امتیاز منفی زیادی دریافت کرده است.

پاداش تجمعی:

نمودار پاداش تجمعی کاهش شدیدی داشته است. این نشان‌دهنده عدم توانایی عامل در دستیابی به یک استراتژی پایدار و بهینه است.

میانگین پاداش:

میانگین پاداش ابتدا افزایش یافته، سپس کمی ثابت شده و بعد با یک افزایش نرم همراه بوده است. این نشان‌دهنده تلاش‌های عامل برای بهبود عملکرد و یافتن استراتژی‌های بهتر است.

مدیریت ریسک:

تحلیل نتایج نشان می‌دهد که عامل در برخی اپیزودها ریسک‌های نامناسبی را پذیرفته و امتیاز منفی زیادی دریافت کرده است. این اطلاعات به ما کمک می‌کند تا در طراحی استراتژی‌های یادگیری بهتر، ریسک را مدیریت کنیم.

پیش‌بینی عملکرد آینده:

با تحلیل الگوهای عملکرد گذشته، می‌توانیم پیش‌بینی کنیم که عامل در آینده چگونه عمل خواهد کرد و آیا نیاز به تغییر استراتژی‌های یادگیری وجود دارد یا خیر.

عملکرد عامل DQN ناپایدار بوده و نتوانسته است به یک استراتژی پایدار و بهینه دست یابد. تنظیم پارامترها و بهبود استراتژی‌های یادگیری می‌تواند به بهبود عملکرد عامل کمک کند.

پیشنهادات برای بهبود

افزایش نرخ یادگیری:

ممکن است افزایش نرخ یادگیری به عامل کمک کند تا سریع‌تر بهبود یابد و پاداش‌های بالاتری کسب کند.

تنظیم اپسیلون:

کاهش اپسیلون به صورت تدریجی‌تر می‌تواند به عامل کمک کند تا به طور مداوم بهبود یابد و ریسک‌های بیشتری را بپذیرد.

تنظیم شبکه عصبی:

افزایش تعداد لایه‌ها یا نورون‌ها در شبکه عصبی می‌تواند به عامل کمک کند تا پیچیدگی‌های بیشتری را درک کند و به عملکرد بهتری دست یابد.

استفاده از تکنیک‌های پیشرفته‌تر:

تکنیک‌های پیشرفته‌تری مانند Double DQN یا Prioritized Experience Replay می‌توانند به بهبود عملکرد عامل کمک کنند.

**Normalization Reward**

# Hyperparameters

BATCH\_SIZE = 64

GAMMA = 0.9

EPS\_START = 0.9

EPS\_END = 0.05

EPS\_DECAY = 200

TAU = 0.01

LR = 0.0001

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Episode 994 finished after 6 steps with total reward 0.009499999789113645

Episode 995 finished after 7 steps with total reward 0.009399999791639857

Episode 996 finished after 6 steps with total reward 0.009499999789113645

Episode 997 finished after 7 steps with total reward 0.009399999791639857

Episode 998 finished after 6 steps with total reward 0.009499999789113645

Episode 999 finished after 6 steps with total reward 0.009499999789113645

Episode 1000 finished after 6 steps with total reward 0.009499999789113645

    normalized\_rewards = [sum(episode\_rewards[:i+1]) / (i + 1) for i in range(len(episode\_rewards))]

مفهوم نرمالیزه کردن پاداش‌ها

نرمالیزه کردن پاداش‌ها به معنای مقیاس‌بندی پاداش‌ها به یک محدوده خاص (معمولاً بین 0 و 1) یا تغییر مقیاس آن‌ها به گونه‌ای است که مقادیر پاداش‌ها در محدوده قابل مقایسه‌ای قرار گیرند. این فرآیند می‌تواند از طریق میانگین‌گیری، استانداردسازی یا دیگر روش‌های آماری انجام شود.

مزایای نرمالیزه کردن پاداش‌ها

بهبود پایداری آموزش:

نرمالیزه کردن پاداش‌ها باعث می‌شود که مقادیر پاداش‌ها در یک محدوده مشخص قرار گیرند و از نوسانات شدید جلوگیری شود. این امر به بهبود پایداری و همگرایی آموزش کمک می‌کند.

شتاب دادن به همگرایی:

زمانی که پاداش‌ها نرمالیزه شوند، عامل می‌تواند الگوهای موجود در محیط را سریع‌تر شناسایی کند و به استراتژی‌های بهینه دست یابد. این امر باعث می‌شود که آموزش عامل سریع‌تر همگرا شود.

کاهش تاثیر پاداش‌های خیلی بزرگ یا خیلی کوچک:

پاداش‌های بزرگ یا کوچک می‌توانند باعث نوسانات شدید در فرآیند آموزش شوند. نرمالیزه کردن پاداش‌ها باعث می‌شود که تاثیر این پاداش‌ها کاهش یابد و آموزش به صورت یکنواخت‌تری پیش برود.

سهولت در تنظیم پارامترها:

وقتی پاداش‌ها نرمالیزه شده باشند، تنظیم پارامترهای مدل (مانند نرخ یادگیری و تخفیف پاداش‌ها) آسان‌تر می‌شود، زیرا مقدار پاداش‌ها در یک محدوده مشخص قرار دارند.

اداش تجمعی:

پاداش تجمعی به صورت لگاریتمی افزایش داشته است. این نشان می‌دهد که عامل به تدریج در حال یادگیری است و با افزایش تعداد اپیزودها، عملکرد بهتری از خود نشان می‌دهد.

همگرایی پاداش‌ها:

مقادیر پاداش در انتهای اپیزودها به مقدار ثابتی نزدیک شده‌اند (حدود 0.0095). این نشان‌دهنده همگرایی خوب عامل است و به معنای آن است که عامل به یک استراتژی بهینه نزدیک شده است.

نوسانات پاداش‌ها:

با وجود همگرایی، نوساناتی در انتها مشاهده می‌شود. این نوسانات ممکن است به دلیل پیچیدگی‌های محیط یا تغییرات تصادفی باشد که عامل با آن‌ها مواجه می‌شود.

نرمالیزه کردن پاداش‌ها به بهبود پایداری و همگرایی عامل کمک کرده و باعث شده است که آموزش به صورت یکنواخت‌تری پیش برود. این فرآیند همچنین باعث کاهش تاثیر پاداش‌های خیلی بزرگ یا خیلی کوچک شده و تنظیم پارامترهای مدل را آسان‌تر کرده است. با این حال، نوسانات در انتهای آموزش نشان می‌دهد که همچنان نیاز به بهبود و تنظیم دقیق‌تر پارامترها وجود دارد.

**برای DQN model2**

# DQN Model

class DQN(nn.Module):

    def \_\_init\_\_(self, n\_observations, n\_actions):

        super(DQN, self).\_\_init\_\_()

        self.fc1 = nn.Linear(n\_observations, 24)

        self.fc2 = nn.Linear(24, 24)

        self.fc3 = nn.Linear(24, n\_actions)

    def forward(self, x):

        x = torch.relu(self.fc1(x))

        x = torch.relu(self.fc2(x))

        return self.fc3(x)

BATCH\_SIZE = 128

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 1000

TAU = 0.0001 or 0.001

LR = 0.0001

MEMORY\_CAPACITY = 10000

**Tau=****0.001**

برای حالت 0.001 عملکرد خیلی خوبی از خود نشان داد و با بهبود عملکرد آموزش الگوریتم به خوبی جواب داد. و در پاداش تجمعی به صورت خوبی در حال افزایش است و با افزایش مقدار اپیزود ها به صورت خوبی عملکرد الگوریتم در حال بهبود است. و نوسان ندارد. در نمودار episode per reward میتوانیم مشاهده کنیم که در هملن ابتدا خیلی خوب در حال یادگیری است. ودر ادامه میبینیم که به مقدار ماکزیمم مقدار همگرا میشه.

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 1000

TAU = 0.001

LR = 0.0001

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -55.277

مقدرا میانگین منفی است که نامطلبوب است ولی با افزایش مقدار episode ها ما عملکرد بهتری را شاهد خواهیم بود همچنین به مقدار ماکزیمم رسیدیم منتها میبینیم که مقدار ماکزیمم 145 با کشتن wumps به دست می اید ولی ما به مقدار 95 هگرا میشویم:

با توجه به گزارش داده شده، می‌توانیم بررسی کنیم که در تعداد زیادی از قسمت‌ها، عامل برای دستیابی به مقدار بالاتر به دنبال کشتن Wumpus نبوده اند. این نشان می‌دهد که قسمت ۵۴۲، عامل توانسته است به‌درستی از این مزیت استفاده کند و در نتیجه، مقدار ۱۴۵ را دریافت کند. از قسمت بعدی، می‌بینیم که عامل تنها به مقدار ۹۵ دست یافته است، که این نشان می‌دهد که تا این مرحله دیگر تلاشی برای کشتن Wumpus انجام نداده است.

نتیجه:

پاداش و هزینه: عامل می‌داند که کشتن Wumpus به واحد ۵۰ امتیاز پاداش می‌دهد، اما اگر در تلاش برای کشتن Wumpus شکست بخورد، به واحد ۱۰۰۰ امتیاز هزینه دارد. این اختلاف بسیار بزرگ میان پاداش و هزینه، باعث می‌شود که عامل به دقت بسیار زیادی در انتخاب تصمیماتش برای کشتن Wumpus نگران باشد.

ریسک بالا: با توجه به اینکه هزینه شکست در اینجا بسیار بالاست، عامل می‌تواند تصمیم بگیرد که این ریسک را نپذیرد و به جای آن به جمع‌آوری امتیازات کمتر اما با ریسک کمتر بپردازد. به عبارت دیگر، عامل می‌تواند به دنبال راهکارهایی بگردد که از طریق آن‌ها امتیازات را افزایش دهد بدون اینکه به این ریسک بزرگ بپردازد.

تغییرات محیطی: محیط Wumpus World ممکن است پویا باشد و شرایط و موقعیت‌های Wumpus و سایر عوامل محیطی می‌توانند تغییر کنند. این تغییرات می‌توانند باعث شوند که عامل از پیش‌بینی دقیق و استفاده بهینه از اقداماتش برای کشتن Wumpus برای بهبود عملکرد خود صرفنظر کند.

بنابراین، عامل ممکن است تصمیم بگیرد که به دنبال راهکارهایی برای بهبود عملکرد خود با ریسک کمتر باشد و از تلاش برای کشتن Wumpus پرهیز کند.

Episode 542 finished after 7 steps with total reward 145

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

**Tau=0.00001**

در این حالت ما نتوانستیم خیلی خوب اموزش دهیم همگرا شد ولی مقدار تجمعی و مقدار میانگین نوسان دار و کاهش دار شد و نتوانست برای این مقدار خوب عمل کند. دلیل اصلی کاهش کیفیت آموزش و عدم همگرایی مدل DQN در حالتی که `Tau=0.00001` است، ممکن است به عوامل زیر بازگردد:

کوچک بودن مقدار TAU

مقدار بسیار کوچکی برای TAU مانند 0.00001 می‌تواند باعث شود که فرایند به‌روزرسانی وزن‌های شبکه هدف (target\_net) خیلی آهسته انجام شود. این موضوع می‌تواند منجر به کندی در فرایند همگرایی شود، به‌ویژه اگر انتخاب TAU به طور نامناسب باشد.

عدم تطابق مناسب بین policy\_net و target\_net

فرایند به‌روزرسانی وزن‌های شبکه هدف به طور کامل باید به تطابق مناسبی بین وزن‌های شبکه سیاست (policy\_net) و شبکه هدف (target\_net) منجر شود. اگر TAU به‌طور نامناسبی تنظیم شود، ممکن است تطابق درستی بین این دو شبکه ایجاد نشود و عملکرد شبکه DQN به‌طور کامل بهینه نشود.

**Tau=0.1**

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 1000

TAU = 0.1

LR = 0.0001

MEMORY\_CAPACITY = 10000

در حالت tau=0.001 نسب به حالت اولیه افت داشتیم و مقدار پاداش تجمعی کاهش زیادی داشت و توانسته wumps را بکشد ولی به مقدار ماکزیمم 145 نتوانسه دست پیدا کند یعنی هم ریسک نکرده هم موقع ریسک کردن نتوانسته مسیر بهینه را برگزیند و به مقدار ماکزیمم خود برسد همچین نمودار پاداش تجمعی نشان داد دارای نوسان هستیم.

در حالتی که TAU=0.001 است، می‌بینیم که عملکرد الگوریتم DQN نسبت به حالت اولیه به شدت کاهش یافته است. میانگین پاداش در هر اپیزود برابر است که نشان از عملکرد ناپایدار مدل دارد.

مقدار نامناسب برای TAU: انتخاب مقدار TAU=0.001 ممکن است نامناسب باشد و باعث نشدن به بهبود کافی در هدف یادگیری شود. TAU که مقداری بسیار کوچکتر از حالت استاندارد TAU=0.001 در مقایسه با TAU=0.001) منجر به کاهش ناپایداری در هدف یادگیری می‌شود و ممکن است باعث نتوانایی در بهبود عملکرد مدل شود.)

نوسان در پاداش‌ها: نمودار پاداش تجمعی نشان دهنده نوسان‌های زیاد در هر اپیزود است. این نوسان‌ها می‌توانند نشان دهنده عدم پایداری و نامناسب بودن شرایط یادگیری باشند، که ممکن است به دلیل انتخاب عمل‌ها توسط مدل با توجه به ε-greedy وابسته به احتمالات تصادفی باشد.

عدم رسیدن به ماکزیمم پاداش 145: عدم توانایی در رسیدن به ماکزیمم پاداش 145 ممکن است به علت ناپایداری در یادگیری باشد. این موضوع معمولاً به عواملی مانند نوسان‌های زیاد در پاداش‌ها و عملکرد نامناسب مدل DQN برمی‌گردد که می‌تواند نتیجه انتخاب مقدار نامناسب برای پارامترهای یادگیری یا نوع معماری شبکه عصبی باشد.

به طور کلی، انتخاب پارامترهای یادگیری مناسب (مانند TAU, ε, learning rate و غیره) بسیار مهم است و باید با آزمون و خطا تنظیم شوند تا به یک عملکرد پایدار و بهینه در یادگیری برسیم.

**Epsilon-decay**

**Epsilon-decay=2000**

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 2000

TAU = 0.001

LR = 0.0001

MEMORY\_CAPACITY = 10000

نشان دهنده احتمال انتخاب عمل تصادفی توسط عامل است، از 1 (کاوش کامل) به 0.01 (بهره‌برداری کامل) کاهش می‌یابد.

فرمول استفاده شده برای کاهش ε به شکل زیر است:

        eps\_threshold = EPS\_END + (EPS\_START - EPS\_END) \* np.exp(-1. \* self.steps\_done / EPS\_DECAY)

که در آن ε\_start=1، ε\_end=0.01 و steps\_done تعداد گام‌هایی است که عامل انجام داده است.

میانگین پاداش در اپیزودها: میانگین پاداش در هر اپیزود برابر با است که به نسبت حالت‌های قبلی با دیگر مقادیر epsilon-decay بهبود داشته است. این نشان می‌دهد که استفاده از مقدار 2000 برای epsilon-decay منجر به یادگیری بهتر و بهره‌برداری مناسب از محیط شده است.

پاداش تجمعی: در این حالت، عملکرد مدل بدون نوسان و افت ناپذیر عمل کرده است، که نشان از پایداری و قدرت الگوریتم DQN در یادگیری طولانی‌مدت محیط است.

قابلیت رسیدن به ماکزیمم پاداش: در اپیزود 830، مدل توانسته به پاداش ماکزیمم 145 برسد که نشان از توانایی عملکردی خوب و قابلیت این الگوریتم در محیط موجود دارد که توانسته wumps را بکشد و ماکزیمم را پیدا کند ولی در ادامه ریسک نکرده که امتیاز منفی بگیرد و با عامل بدون ریسک برای دست یابی به مقدار 95 که ماکزیمم حالت بدون ریسک است اقدام کرده.

نوسان در پاداش: در برخی از اپیزودها که پاداش 95 بوده است، مشاهده می‌شود که مدل در چند مورد تصمیمات نامناسبی گرفته است. با این حال، این نوسان به نحوی کنترل شده است که به کلی عملکرد الگوریتم را تحت تأثیر قرار نداده است. و نسبت به بقیه حالات عملکرد بهتری داشته.

سیر تحول Epsilon و عملکرد الگوریتم:

در ابتدا، ε بسیار بالا است و عامل به طور کامل در حال کاوش است و احتمال انتخاب عمل تصادفی بسیار بالاست.

با گذشت زمان مقدار steps\_done افزایش می‌یابد، ε به تدریج کاهش می‌یابد و عامل بیشتر بهره‌برداری از دانشی که در فرآیند کاوش به دست آورده، می‌پردازد.

این فرآیند به ادامه یافتن آموزش، منجر به بهبود عملکرد کلی الگوریتم و افزایش احتمال رسیدن به حالت‌های پاداش بالا می‌شود.

با توجه به این تحلیل، استفاده از Epsilon-decay=2000 به عنوان یک پارامتر مهم در الگوریتم DQN است.

|  |
| --- |
|  |

Episode 830 finished after 7 steps with total reward 145

Episode 992 finished after 6 steps with total reward 95

Episode 993 finished after 6 steps with total reward 95

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward

**Epsilon decay=500**

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 500

TAU = 0.001

LR = 0.0001

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Episode 797 finished after 7 steps with total reward 145

Episode 992 finished after 6 steps with total reward 95

Episode 993 finished after 4 steps with total reward -1003

Episode 994 finished after 4 steps with total reward -1003

Episode 995 finished after 4 steps with total reward -1003

Episode 996 finished after 4 steps with total reward -1003

Episode 997 finished after 4 steps with total reward -1003

Episode 998 finished after 4 steps with total reward -1003

Episode 999 finished after 4 steps with total reward -1003

Episode 1000 finished after 4 steps with total reward -1003

میانگین پاداش در اپیزودها

میانگین پاداش در هر اپیزود:

نوسان در مقادیر پاداش: در اپیزودهای نهایی()، پاداش‌ها به شدت منفی هستند که باعث افت شدید در میانگین پاداش می‌شود.

توانایی در کشتن Wumpsرا دارد ولی نتایج نامطلوب است نتایج.

الگوریتم توانسته است در برخی از اپیزودها (مانند اپیزود 992) وامپ‌ها را با موفقیت نابود کند، اما در اکثر اپیزودها (از 990 تا 1000) با پاداش منفی مواجه شده است. این نشان می‌دهد که الگوریتم در تصمیم‌گیری‌های خود ناکارآمد بوده است و به نتایج منفی منجر شده است.

نوسان و ناپایداری در پاداش تجمعی:

نوسان در پاداش: نمودار تجمعی پاداش نشان می‌دهد که عملکرد الگوریتم پر از نوسانات بوده و تاثیرات منفی بسیاری را تجربه کرده است. این نوسانات نشان از عدم پایداری در یادگیری و عدم توانایی الگوریتم در ایجاد یک الگوی ثابت برای تصمیم‌گیری در محیط است.

میانگین پاداش:

مقدار منفی میانگین پاداش: از آنجا که الگوریتم در بسیاری از اپیزودها به پاداش منفی مواجه شده است، میانگین پاداش به شدت منفی شده است. این نشانگر این است که الگوریتم نتوانسته است یک استراتژی مؤثر برای جلوگیری از پاداش‌های منفی توسعه دهد و به دنبال بهبود عملکرد در مدت زمان طولانی نبوده است.

عملکرد الگوریتم با استفاده از Epsilon-decay=500 به نتایج مطلوبی نرسیده است. نوسانات شدید در پاداش و عدم پایداری در نمودار تجمعی، همراه با میانگین پاداش منفی، نشان از ناکارآمدی استراتژی اعمال شده توسط الگوریتم دارد. بهبودهای لازم در انتخاب استراتژی‌ها و پارامترهای مربوطه می‌تواند به بهبود عملکرد و افزایش پایداری در یادگیری منجر شود.

**Gamma=0.9**

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.9

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 2000

TAU = 0.001

LR = 0.0001

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Episode 127 finished after 53 steps with total reward 99

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

Average reward per episode after 1000 episodes: -144.024

میانگین پاداش

الگوریتم در این اپیزودها به پاداش 95 همگرا شده است، به این معنی که در تمام این اپیزودها، پاداش به این مقدار رسیده است. این نشان می‌دهد که الگوریتم به یک استراتژی خاص برای اجرای اقدامات در محیط رسیده و ممکن است از انجام اقدامات با ریسک بالا (مانند کشتن وامپ‌ها که می‌تواند به پاداش منفی 1003 منجر شود) اجتناب کرده باشد.

میانگین پاداش:

میانگین پاداش نشان می‌دهد که الگوریتم هنوز در حال یادگیری است و نیاز به بهبود استراتژی‌های خود دارد تا به میانگین پاداش مثبت و بهتری دست یابد.

نمودار تجمعی پاداش:

از آنجایی که نمودار تجمعی پاداش در ابتدا کمی کاهش داشته، به دلیل این است که الگوریتم در ابتدا به دنبال اکتشاف و یادگیری استراتژی‌های مناسب بوده است که ممکن است به انجام اقدامات با پاداش کم‌تر منجر شده باشد. با گذر زمان و بهبود استراتژی‌ها، الگوریتم به سمت افزایش پاداش حرکت کرده و نمودار تجمعی پاداش به روند افزایشی میل کرده است.

عملکرد الگوریتم در این حالت نشان می‌دهد که الگوریتم به یک استراتژی معقول برای اجرای اقدامات در محیط رسیده است، اما هنوز به ماکزیمم پاداش ممکن نرسیده و نیاز به بهبود بیشتر دارد. ادامه دادن به یادگیری و بهبود استراتژی‌های کنونی می‌تواند به تحقق این هدف کمک کند و باعث بهبود عملکرد کلی الگوریتم شود.

**Learning rate**

**gamma=0.9 and alpha=0.01**

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -199.661

Episode 430 finished after 15 steps with total reward 137

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

با وجود اینکه الگوریتم توانسته وامپ‌ها را کشته و به پاداش 95 همگرا شود، اما به ماکزیمم پاداش 145 که با کشتن وامپ‌ها در دسترس است، نرسیده است. این ممکن است به دلیل استراتژی‌های انتخاب شده توسط الگوریتم باشد که ممکن است از انجام اقدامات با ریسک بالا اجتناب کرده و به انتخاب استراتژی‌های ایمن‌تر و کم‌ریسک تر می‌پردازد. نمودار تجمعی پاداش رشد داشته است که نشان می‌دهد الگوریتم بهبود یافته و استراتژی‌های بهتری را در طول یادگیری اجرا می‌کند. این بهبود نشان دهنده توانایی الگوریتم در یادگیری و بهره‌گیری بهینه از محیط است. با توجه به نتایج ارائه شده، عملکرد الگوریتم به طور کلی بهبود یافته است اما هنوز می‌تواند در بهینه‌سازی استراتژی‌های خود پیشرفت کند تا به ماکزیمم پاداش ممکن دست یابد. ادامه دادن به یادگیری و بهبود استراتژی‌های کنونی می‌تواند به تحقق این هدف کمک کند و باعث بهبود عملکرد کلی الگوریتم شود.

**برای batch size=128**

# Hyperparameters

BATCH\_SIZE = 128

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 1000

TAU = 0.001

LR = 0.0001

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 8 steps with total reward 93

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

**برای batch size=64**

# Hyperparameters

BATCH\_SIZE = 64

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 1000

TAU = 0.001

LR = 0.0001

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Episode 994 finished after 2 steps with total reward -1001

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

**برای batch size=32**

# Hyperparameters

BATCH\_SIZE = 32

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 1000

TAU = 0.001

LR = 0.0001

MEMORY\_CAPACITY = 10000

|  |
| --- |
|  |

Episode 989 finished after 6 steps with total reward 95

Episode 990 finished after 7 steps with total reward 145

Episode 991 finishe Episode 992 finished after 6 steps with total reward 95

Episode 993 finished after 6 steps with total reward 95

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95d after 6 steps with total reward 95

# ب. عملکرد :Policy

# پاداش تجمعی در اپیزودها برای هر دوعاملQ-Learning و DQN ترسیم کنید. چگونه عملکرد در طول زمان بهبود می یابد؟

# میانگین پاداش در هر اپیزود برای هر دوعامل پس از 1000 اپیزود مقایسه کنید. کدام الگوریتم عملکرد بهتری داشت؟

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -429.221

First episode where agent consistently finds gold for consecutive episodes: 656

Q-learning Episode 728/1000, Total Reward: 145

Q-learning Episode 989/1000, Total Reward: 95

Q-learning Episode 990/1000, Total Reward: 95

Q-learning Episode 991/1000, Total Reward: 95

Q-learning Episode 992/1000, Total Reward: 95

Q-learning Episode 993/1000, Total Reward: 95

Q-learning Episode 994/1000, Total Reward: 95

Q-learning Episode 995/1000, Total Reward: 95

Q-learning Episode 996/1000, Total Reward: 95

Q-learning Episode 997/1000, Total Reward: 95

Q-learning Episode 998/1000, Total Reward: 95

Q-learning Episode 999/1000, Total Reward: 95

Q-learning Episode 1000/1000, Total Reward: 95

**DQN**

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: 29.059

Episode 893 finished after 7 steps with total reward 145

Episode 978 finished after 6 steps with total reward 95

Episode 979 finished after 6 steps with total reward 95

Episode 980 finished after 6 steps with total reward 95

Episode 981 finished after 6 steps with total reward 95

Episode 984 finished after 7 steps with total reward 94

Episode 985 finished after 6 steps with total reward 95

Episode 986 finished after 6 steps with total reward 95

Episode 987 finished after 6 steps with total reward 95

Episode 988 finished after 6 steps with total reward 95

Episode 989 finished after 6 steps with total reward 95

Episode 990 finished after 6 steps with total reward 95

Episode 991 finished after 6 steps with total reward 95

Episode 992 finished after 6 steps with total reward 95

Episode 993 finished after 6 steps with total reward 95

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 7 steps with total reward 94

بهبود عملکرد عامل در طول زمان بهبود عملکرد عامل در طول زمان می‌تواند با بررسی نمودارهای پاداش تجمعی، میانگین پاداش در هر قسمت، و نقاط عطف خاص که تغییرات قابل توجهی در عملکرد رخ می‌دهد، تحلیل شود. در زیر توضیحات دقیقی در این مورد آمده است:

عامل یادگیری Q: قسمت‌های اولیه (1-655):

فاز کاوش: در قسمت‌های اولیه، عامل به طور عمده محیط را کاوش می‌کند به دلیل مقدار بالای اپسیلون. این موضوع منجر به عملکرد ناپایدار با پاداش‌های کم و گاهی منفی می‌شود، زیرا عامل هنوز در حال یادگیری عملکردهای بهینه است. چالش‌های یادگیری: عامل با چالش‌هایی در یافتن طلا و اجتناب از خطرات به طور مداوم روبرو می‌شود، که منجر به پاداش‌های نوسانی می‌شود. قسمت‌های میانی (656-789):

بهبود عملکرد: پس از قسمت 656، بهبود قابل توجهی در عملکرد مشاهده می‌شود. عامل شروع به یافتن طلا به طور مداوم می‌کند، که نشان‌دهنده این است که شروع به یادگیری سیاست موثر کرده است. افزایش پاداش: پاداش تجمعی شروع به افزایش به طور پیوسته می‌کند به دلیل بهینه‌سازی عملکردهای عامل. قسمت‌های بعدی (790-1000):

عملکرد پایدار: از قسمت 790 به بعد، عامل عملکرد پایدار و مداومی را نشان می‌دهد. برای مثال، کسب پاداش کلی 95 در هر قسمت از قسمت‌های 990 تا 1000. پایداری سیاست: عامل به طور موثر یاد گرفته است که چگونه محیط را کاوش کند، طلا را پیدا کند و از خطرات اجتناب کند، که منجر به پاداش‌های بالا به طور مداوم می‌شود.

عامل DQN: قسمت‌های اولیه (1-600):

کاوش و یادگیری: مشابه عامل یادگیری Q، عامل DQN در ابتدا به طور گسترده‌ای محیط را کاوش می‌کند. با این حال، به دلیل استفاده از بازیابی تجربیات و شبکه‌های هدف، عامل شروع به یادگیری به طور کارآمدتر می‌کند. بهبودهای اولیه: عامل نشان می‌دهد که عملکرد آن به طور پیوسته در مقایسه با عامل یادگیری Q زودتر افزایش می‌یابد، که نشان‌دهنده این است که سیاست بهینه را سریع‌تر یاد می‌گیرد. قسمت‌های میانی (601-900):

بهبود پایدار: عملکرد عامل DQN در طول این قسمت‌ها به طور پیوسته بهبود می‌یابد. استفاده از بازیابی تجربیات در پایدار کردن فرآیند یادگیری کمک می‌کند، و نوسانات مشاهده شده در عامل یادگیری Q را کاهش می‌دهد. پاداش‌های بالاتر: پاداش‌های تجمعی ادامه دارد به افزایش می‌یابد، که نشان‌دهنده توانایی رو به افزایش عامل در کسب پاداش‌های بالا است. قسمت‌های بعدی (901-1000):

عملکرد بهینه: تا زمانی که عامل به قسمت‌های 978 تا 1000 می‌رسد، به طور مداوم پاداش‌های بالا (95 در هر قسمت) را کسب می‌کند. این نشان می‌دهد که عامل به طور موثر یک سیاست بهینه را یاد گرفته است و می‌تواند به طور قابل اعتماد از آن استفاده کند. سیاست پایدار: عملکرد بیشتر پایدار و مداوم است، که نشان می‌دهد عامل DQN نه تنها عملکردهای بهینه را یاد گرفته است بلکه آن‌ها را به طور موثر حفظ کرده است. خلاصه عامل یادگیری Q:

برای رسیدن به عملکرد پایدار، زمان بیشتری می‌برد. پس از قسمت 656، بهبود قابل توجهی نشان می‌دهد، با پاداش‌های بالا و پایدار از قسمت 790 به بعد مشاهده می‌شود. عامل DQN:

نسبت به عامل یادگیری Q، بهبود سریع‌تر و مداوم‌تری را نشان می‌دهد. پاداش‌های بالا را زودتر کسب می‌کند و آن‌ها را به طور قابل اعتمادتر حفظ می‌کند، که بازتاب کارایی DQN در یادگیری و پایدار کردن سیاست بهینه است.

نتیجه‌گیری عملکرد عامل در طول زمان از طریق ترکیبی از کاوش و یادگیری از تجربه بهبود می‌یابد. عامل یادگیری Q بهبود تدریجی را نشان می‌دهد، با نقاط عطف قابل توجه در قسمت‌های 656 و 790. در مقابل، عامل DQN به طور سریع‌تر و مداوم‌تر بهبود می‌یابد، که برتری‌های بازیابی تجربیات و شبکه‌های هدف در پایدار کردن فرآیند یادگیری و دستیابی به سطوح بالاتر عملکرد در زودترین زمان ممکن در فرآیند آموزش را برجسته می‌کند.

میانگین پاداش:

میانگین پاداش در هر اپیزود پس از 1000 اپیزود:

اپیزودهای ابتدایی با مشکلات زیادی روبرو شده‌اند که منجر به پاداش‌های منفی شده است.

از اپیزود 656 به بعد، عملکرد به طور قابل توجهی بهبود یافته و پاداش‌های مثبت بیشتری کسب شده است.

اپیزودهای موفقیت‌آمیز:

اولین اپیزودی که عامل به طور پیوسته طلا را پیدا کرده 656

در اپیزود 728، عامل به پاداش 145 دست یافته است. که توانسته wumps را بکشه و به ماکزیمم پاداش دست یابد.

در اپیزودهای 989 تا 1000، عامل به طور مداوم پاداش 95 را کسب کرده است.

نمودارها:

نمودار پاداش برای هر اپیزود نشان می‌دهد که پس از اپیزود 656 بهبود چشمگیری در عملکرد رخ داده است.

نمودار پاداش تجمعی نشان می‌دهد که ابتدا کاهشی بوده و پس از اپیزود 656 افزایشی شده است.

نمودار میانگین پاداش برای هر اپیزود نیز نشان می‌دهد که میانگین پاداش به طور که تا656 بهصورت لگاریتمی و بعد ان به صورت صعودی و صاف در حال افزایش است.

DQN:

میانگین پاداش:

میانگین پاداش در هر اپیزود پس از 1000 اپیزود:

عامل در اپیزودهای ابتدایی با مشکلات کمتری مواجه شده و توانسته به پاداش‌های مثبت بیشتری دست یابد.

اپیزودهای موفقیت‌آمیز:

اپیزود 893: پاداش 145 کسب شده است. توانسته wups را بکشد.

در اپیزودهای 987 تا 1000، عامل به طور مداوم پاداش 95 را کسب کرده است.

عامل از اپیزود 180 به بعد تقریبا به طور مداوم به طلا دست یافته است.

نمودارها:

نمودار پاداش برای هر اپیزود نشان می‌دهد که عامل از اپیزود 180 به بعد به طور مداوم به طلا می‌رسد، ولی گاهی تصمیمات اشتباهی دارد که منجر به پاداش‌های منفی می‌شود.

نمودار پاداش تجمعی ابتدا کاهش و سپس افزایش داشته و در اپیزود170 مسیر مجدداً کاهش یافته و سپس افزایش زیادی داشته است.

نمودار میانگین پاداش برای هر اپیزود به صورت لگاریتمی افزایش یافته ولی در اپیزود 170 کمی کاهش داشته و سپس دوباره افزایش یافته است. باتوجه به نمودار ها متوجه میشویم که DQN عملکرد بهتری داشته و در اپیزود 1000 میبینیم نزدیک 0 است میانگین پاداش در هر اپیزود ولی در QL عملکرد پایین تری داشته و مقدار ان در اپیزود 1000 کمتر است و یادگیری بهتری داشته DQN

سرعت یادگیری: DQN به وضوح سریع‌تر از Q-Learning policy بهینه را یادگرفته است. عامل DQN در اپیزود 180 به طور مداوم به طلا دست یافته در حالی که Q-Learning در اپیزود 656 به این سطح از عملکرد رسیده است.

پایداری عملکرد: هر دو عامل در اپیزودهای پایانی به طور مداوم پاداش 95 را کسب کرده‌اند، ولی DQN به طور کلی در یادگیری و دستیابی به پاداش‌های مثبت پایدارتر عمل کرده است.

پاداش تجمعی و میانگین پاداش: نمودارهای پاداش تجمعی و میانگین پاداش نشان می‌دهند که عملکرد DQN به مرور زمان بهتر شده و پس از یادگیری، به طور پایدار پاداش‌های مثبت بیشتری کسب کرده است.

عامل DQN عملکرد بهتری نسبت به عامل یادگیری Q داشت: عامل DQN به طور قابل توجهی عملکرد بهتری نسبت به عامل یادگیری Q از نظر میانگین پاداش در هر قسمت پس از 1000 قسمت داشت. پاداش میانگین مثبت برای عامل DQN (29.059) در مقایسه با پاداش میانگین منفی برای عامل یادگیری Q (-429.221)، اثربخشی الگوریتم DQN در یادگیری و پایدار کردن سیاست‌های بهینه را نشان می‌دهد. توانایی عامل DQN در کسب و حفظ پاداش‌های بالا به طور مداوم در قسمت‌های بعدی، عملکرد برتر و کارایی آن در یادگیری وظیفه را برجسته می‌کند. نتیجه‌گیری الگوریتم DQN بهتر از الگوریتم یادگیری Q عمل کرد، که این موضوع با پاداش میانگین بالاتر در هر قسمت پس از 1000 قسمت ثابت شده است. بهبودهای ایجاد شده در روش DQN، مانند بازیابی تجربیات و شبکه‌های هدف، به یادگیری پایدار و موثرتر کمک کرد، که منجر به عملکرد بهتر در کل و پاداش‌های بالاتر شد.

# بحث کنید که چگونه نرخ اکتشاف اپسیلون بر فرایند تاثیر میگذارد وقت اپسیلون بالا بود درمقابل وقتی کم بود چه چیز را مشاهده کردید؟

پاداش تجمعی در طول زمان

برای هر دو عامل Q-learning و DQN، پاداش تجمعی در طول اپیزودها پیگیری شده است. داده‌ها نشان می‌دهند که چگونه عملکرد عامل‌ها با پیشرفت هر اپیزود بهبود می‌یابد.

عامل Q-learning:

پاداش تجمعی پس از اپیزود 656 ابتدا به طور قابل توجهی افزایش می‌یابد.

تا انتهای 1000 اپیزود، عامل Q-learning به طور مداوم طلا را پیدا می‌کند که نشان از بهبود قابل توجه در عملکرد است.

به عنوان مثال، از اپیزود 990 تا 1000، عامل Q-learning به طور مداوم به پاداش کلی 95 در هر اپیزود می‌رسد.

عامل DQN:

عامل DQN به طول اپیزودها بهبود مستمری را نشان می‌دهد و به نقطه‌ای می‌رسد که اپیزودها را با کمترین مراحل و پاداش‌های بالا به اتمام می‌رساند.

به عنوان مثال، از اپیزود 978 تا 1000، عامل DQN به طور مداوم به پاداش کلی 95 در هر اپیزود می‌رسد.

میانگین پاداش در هر اپیزود

عامل Q-learning: میانگین پاداش در هر اپیزود پس از 1000 اپیزود است.

عامل DQN: میانگین پاداش در هر اپیزود پس از 1000 اپیزود است.

مقایسه عملکرد:

عامل DQN نسبت به عامل Q-learning در میانگین پاداش در هر اپیزود پس از 1000 اپیزود عملکرد بهتری داشته است.

میانگین پاداش بالاتر عامل DQN نشان می‌دهد که این عامل توانسته است بهتر و به صورت مداوم‌تر پاداش‌های بالاتری را نسبت به عامل Q-learning کسب کند.

تأثیر نرخ اکتشاف اپسیلون

نرخ اپسیلون در Q-learning کنترل می‌کند میزان مبادله بین اکتشاف و بهره‌برداری است:

**اگر rand() < epsilon**

این به این معنی است که اگر یک عدد تصادفی کمتر از اپسیلون (یک عدد کوچک مثبت) باشد، در این صورت:

یک عمل تصادفی انجام دهید

عامل باید یک عمل تصادفی انجام دهد. این قسمت کاوش است که در آن عامل سعی می‌کند اعمال جدیدی را امتحان کند تا ببیند آیا این اعمال بهتر هستند یا نه.

. else

اگر عدد تصادفی کمتر از اپسیلون نباشد، در این صورت:

.بهترین عمل را بر اساس جدول Q انجام دهید (عمل حریصانه)

عامل باید عملی را انجام دهد که به نظر او بر اساس جدول Q بهترین است. این قسمت بهره‌برداری است که در آن عامل از آنچه آموخته است برای انجام بهترین عمل استفاده می‌کند.

اپسیلون یک عدد کوچک مثبت (معمولاً بین 0 و 1) است که تعیین می‌کند چقدر از وقت‌ها عامل باید کاوش کند (عمل تصادفی انجام دهد) و چقدر از وقت‌ها باید بهره‌برداری کند (بهترین عمل را انجام دهد). اگر اپسیلون بالا باشد، عامل بیشتر کاوش می‌کند و اگر پایین باشد، بیشتر بهره‌برداری می‌کند.

کاوش به عامل کمک می‌کند تا با انجام اعمال جدید، اطلاعات بیشتری در مورد محیط خود به دست آورد. این می‌تواند به کشف استراتژی‌های جدیدی که ممکن است به پاداش‌های بیشتر منجر شوند کمک کند.

بهره‌برداری به عامل کمک می‌کند تا از دانشی که تا کنون کسب کرده است برای به دست آوردن پاداش‌های بیشتر استفاده کند. این معمولاً به معنی انتخاب عملی است که در جدول Q بالاترین مقدار را دارد.

**تعادل بین کاوش و بهره‌برداری:**

یک چالش اساسی در یادگیری تقویتی تعادل بین کاوش و بهره‌برداری است. سیاست Epsilon Greedy این تعادل را با استفاده از پارامتر اپسیلون برقرار می‌کند، به طوری که گاه‌گاهی عامل کاوش می‌کند و بیشتر اوقات بهره‌برداری می‌کند.

با استفاده از سیاست Epsilon Greedy، عامل می‌تواند به تدریج دانش خود را بهبود دهد و در نهایت به یک سیاست بهینه برای به دست آوردن بیشترین پاداش برسد.

به طور کلی، تنظیم مناسب مقدار گاما بستگی به مسئله خاص دارد. در مسائلی که پاداش‌های بلندمدت اهمیت بیشتری دارند، معمولاً از مقدار گامای بزرگتر استفاده می‌شود و در مسائلی که پاداش‌های کوتاه‌مدت مهم‌تر هستند، از مقدار گامای کوچکتر استفاده می‌شود.

در الگوریتم Q-learning، اکتشاف (Exploration) و بهره‌برداری (Exploitation) دو جنبه کلیدی هستند که باید بین آن‌ها تعادل برقرار شود.

اکتشاف به معنای امتحان کردن اقدامات جدید و ناشناخته برای جمع‌آوری اطلاعات بیشتر درباره محیط است. این کار به عامل کمک می‌کند که:

از افتادن در دام انتخاب‌های محلی بهینه جلوگیری کند.

اطلاعات کاملی درباره تمامی اقدامات ممکن و نتایج آن‌ها کسب کند.

راه‌حل‌های جدید و احتمالا بهتری را کشف کند.

بهره‌برداری به معنای استفاده از اطلاعات فعلی برای انتخاب بهترین اقدام ممکن است. در این روش، عامل بر اساس دانش فعلی خود اقداماتی را انجام می‌دهد که بیشترین پاداش را به همراه دارند. بهره‌برداری به عامل کمک می‌کند که:

در کوتاه‌مدت پاداش‌های بیشتری کسب کند.

از دانش موجود به بهترین شکل ممکن استفاده کند.

در الگوریتم‌های یادگیری تقویتی، لازم است بین اکتشاف و بهره‌برداری تعادل برقرار شود. یکی از روش‌های رایج برای این کار، استفاده از سیاست epsilon-greedy است که به این صورت عمل می‌کند:

با احتمال ε (epsilon) عامل اکتشاف می‌کند و یک اقدام تصادفی انتخاب می‌کند.

- با احتمال عامل بهره‌برداری می‌کند و بهترین اقدام ممکن بر اساس جدول Q را انتخاب می‌کند.

در ابتدای فرآیند یادگیری، مقدار ε بالا تنظیم می‌شود تا عامل بیشتر اکتشاف کند و اطلاعات بیشتری کسب کند. به مرور زمان و با افزایش دانش عامل، مقدار ε کاهش می‌یابد تا عامل بیشتر بهره‌برداری کند و از دانش خود برای کسب پاداش‌های بیشتر استفاده کند.

اکتشاف به معنای انجام اقدامات تصادفی یا کم‌تر شناخته شده است که می‌تواند به کشف اطلاعات جدید و یادگیری بهتر منجر شود. از آنجایی که در این حالت عامل ریسک می‌پذیرد، می‌توان از مقادیر کمتری برای k استفاده کرد. این کار باعث می‌شود احتمال انتخاب اقدامات با ارزش Q پایین‌تر افزایش یابد.

بهره‌برداری به معنای استفاده از اطلاعات فعلی و دانش به دست آمده است تا اقداماتی که به احتمال زیاد بیشترین پاداش را دارند را انتخاب کنیم. در این حالت، می‌توان از مقادیر بالاتری برای k استفاده کرد. این باعث می‌شود اقدامات با ارزش Q بالاتر انتخاب شوند و احتمال انتخاب اقدامات با ارزش Q پایین‌تر کاهش یابد.

فرمول معمول برای محاسبه احتمال انتخاب یک اقدام ai در حالت s به صورت زیر است:

p(ai|s) احتمال انتخاب اقدام ai در حالت s است.

Q(s,ai) ارزش Q برای اقدام ai در حالت s است.

kیک ثابت است که بسته به میزان اکتشاف یا بهره‌برداری می‌تواند مقادیر مختلفی داشته باشد. اگر بخواهیم اکتشاف کنیم، می‌توانیم k را کم و اگر بخواهیم بهره‌برداری کنیم، می‌توانیم k را بیشتر در نظر بگیریم.

اپسیلون بالا (بیشترین اکتشاف):

در ابتدای فرآیند آموزش، ارزش بالای اپسیلون باعث می‌شود که عامل بیشتر اکتشاف کند که به او کمک می‌کند تا جفت‌های حالت-عمل را که منجر به پاداش‌های بالاتر می‌شوند، کشف کند.

عامل Q-learning ابتدا دشواری‌هایی در پیدا کردن پاداش‌های بهینه را داشت که این موضوع در پاداش‌های پایین در اپیزودهای اولیه نشان داده شده است.

اپسیلون پایین (بیشترین بهره‌برداری):

با پیشرفت آموزش، کاهش اپسیلون باعث می‌شود که عامل از پاداش‌های شناخته شده بهره ببرد و عملکرد خود را استحکام بخشد.

هر دو عامل در نهایت به استفاده از استراتژی‌های یادگرفته شده برای دستیابی به پاداش‌های بالا به صورت مداوم می‌پردازند، که در اپیزودهای بعدی مشاهده می‌شود.

مشاهدات:

هنگامی که اپسیلون بالا بود، هر دو عامل رفتار بیشتری در اکتشاف نشان دادند که ابتدا منجر به عملکرد نامنظم شد، اما به عامل‌ها امکان می‌دهد تا اطلاعات مفیدی درباره محیط جمع‌آوری کنند.

هنگامی که اپسیلون پایین بود، هر دو عامل عملکرد پایدار و مطمئن‌تری را نشان دادند و از استراتژی‌های یادگرفته شده برای دستیابی به پاداش‌های بالا بهره بردند.

پارامترهای تاثیر گذار در نرخ اکتشاف اپسیلون:epsilon\_decay, epsilon\_start, start\_end می باشد

فرمول نرخ اکتشاف اپسیلون:

        eps\_threshold = EPS\_END + (EPS\_START - EPS\_END) \* np.exp(-1. \* self.steps\_done / EPS\_DECAY)

مجموعه یک

# Hyperparameters

BATCH\_SIZE = 64

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 200

TAU = 0.001

LR = 0.001

MEMORY\_CAPACITY = 10000

مجموعه دوم

# Hyperparameters

BATCH\_SIZE = 64

GAMMA = 0.99

EPS\_START = 1.0

EPS\_END = 0.01

EPS\_DECAY = 800

TAU = 0.001

LR = 0.001

MEMORY\_CAPACITY = 10000

**برای مدل یک DQN**

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: 29.059

Episode 893 finished after 7 steps with total reward 145

Episode 991 finished after 6 steps with total reward 95

Episode 992 finished after 6 steps with total reward 95

Episode 993 finished after 6 steps with total reward 95

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 7 steps with total reward 94

میانگین پاداش:

میانگین پاداش در هر اپیزود پس از 1000 اپیزود: 29.059

عامل در اپیزودهای ابتدایی با مشکلات کمتری مواجه شده و توانسته به پاداش‌های مثبت بیشتری دست یابد.

اپیزودهای موفقیت‌آمیز:

اپیزود 893: پاداش 145 کسب شده است.

در اپیزودهای 987 تا 1000، عامل به طور مداوم پاداش 95 را کسب کرده است.

عامل از اپیزود 180 به بعد تقریبا به طور مداوم به طلا دست یافته است.

نمودارها:

نمودار پاداش برای هر اپیزود نشان می‌دهد که عامل از اپیزود 180 به بعد به طور مداوم به طلا می‌رسد، ولی گاهی تصمیمات اشتباهی دارد که منجر به پاداش‌های منفی می‌شود.

نمودار پاداش تجمعی ابتدا کاهش و سپس افزایش داشته و در اپیزود170 مسیر مجدداً کاهش یافته و سپس افزایش زیادی داشته است.

نمودار میانگین پاداش برای هر اپیزود به صورت لگاریتمی افزایش یافته ولی در اپیزود 170 کمی کاهش داشته و سپس دوباره افزایش یافته است.

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: 19.414

Episode 992 finished after 6 steps with total reward 95

Episode 993 finished after 6 steps with total reward 95

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

ژ نمودار پاداش برای هر اپیزود از اپیزود 205 به بعد به طور مداوم به طلا می‌رسد، اما گاهی اوقات تصمیمات نادرستی اتخاذ می‌کند که باعث کاهش پاداش می‌شود، ممکن است این تصمیمات مربوط به عواملی مانند عدم کشف Wumpus باشد که منجر به کاهش پاداش تجمعی می‌شود. در اپیزودهایی که از 150 تا 180 هستند، پاداش به طور ثابتی می‌ماند، سپس در اپیزودهای بین 180 تا 205، کاهشی وجود دارد و بعداً دوباره افزایش پیدا می‌کند. نمودار میانگین پاداش نیز به صورت لگاریتمی افزایش داشته است، اما نزدیک اپیزود 200، یک کاهش کوچک داشته و سپس دوباره افزایشی نشان داده است و به نزدیکی صفر رسیده است.

تاثیر نرخ اکتشاف اپسیلون بر فرایند یادگیری در مجموعه 1 و مجموعه 2 DQN

در اینجا به بررسی تاثیر نرخ اکتشاف اپسیلون بر فرایند یادگیری دو مجموعه DQN می‌پردازیم:

مجموعه 1

نرخ اکتشاف اپسیلون:

شروع اپسیلون: 1.0

پایان اپسیلون: 0.01

کاهش اپسیلون: 200

نتایج و مشاهده‌ها:

اکتشاف اولیه: با شروع اپسیلون بالا (1.0)، عامل بیشتر به اکتشاف می‌پردازد. این منجر به جمع‌آوری داده‌های متنوعی می‌شود که برای بهبود مدل یادگیری در مراحل بعدی مفید است.

کاهش سریع اپسیلون: با کاهش اپسیلون به مقدار 0.01 و نرخ کاهش سریع (200)، عامل به سرعت به سمت بهره‌برداری از دانسته‌های خود می‌رود. این باعث می‌شود که عامل در مراحل پایانی آموزش، عملکرد بهتری از خود نشان دهد و بازدهی بالاتری داشته باشد.

عملکرد نهایی: میانگین بازدهی در هر اپیزود پس از 1000 اپیزود برای مجموعه 1 برابر با 29.059 است، که نشان‌دهنده عملکرد نسبتاً خوب عامل در یافتن پاداش‌ها و تصمیم‌گیری بهینه است.

مجموعه 2

نرخ اکتشاف اپسیلون:

شروع اپسیلون: 1.0

پایان اپسیلون: 0.01

کاهش اپسیلون: 800

نتایج و مشاهده‌ها:

اکتشاف اولیه: با شروع اپسیلون بالا (1.0)، عامل بیشتر به اکتشاف می‌پردازد. این منجر به جمع‌آوری داده‌های متنوعی می‌شود که برای بهبود مدل یادگیری در مراحل بعدی مفید است.

کاهش آهسته‌تر اپسیلون: با کاهش اپسیلون به مقدار 0.01 و نرخ کاهش آهسته‌تر (800)، عامل به تدریج به سمت بهره‌برداری از دانسته‌های خود می‌رود. این باعث می‌شود که عامل در مراحل میانی و پایانی آموزش، عملکرد پایدارتری داشته باشد و بازدهی بهتری از خود نشان دهد.

عملکرد نهایی: میانگین بازدهی در هر اپیزود پس از 1000 اپیزود برای مجموعه 2 برابر با 19.414 است، که نشان‌دهنده عملکرد خوب عامل است، اما کمتر از مجموعه 1.

مقایسه عملکرد دو مجموعه

مجموعه 1: با کاهش سریع‌تر اپسیلون، عامل توانسته است به سرعت از اکتشاف به بهره‌برداری منتقل شود و عملکرد بهتری از خود نشان دهد. این منجر به بازدهی بالاتر و پایدارتری در مراحل پایانی آموزش شده است.

مجموعه 2: با کاهش آهسته‌تر اپسیلون، عامل به تدریج به سمت بهره‌برداری از دانسته‌های خود رفته است. این باعث شده است که عملکرد عامل پایدار باشد، اما بازدهی نهایی کمی کمتر از مجموعه 1 باشد.

نتیجه‌گیری

نرخ اکتشاف اپسیلون تأثیر مهمی در فرایند یادگیری دارد. شروع با اپسیلون بالا باعث اکتشاف بیشتر و جمع‌آوری داده‌های متنوع می‌شود.

کاهش سریع‌تر اپسیلون (مجموعه 1) به عامل کمک می‌کند تا سریع‌تر به بهره‌برداری از دانسته‌های خود بپردازد و عملکرد بهتری داشته باشد.

کاهش آهسته‌تر اپسیلون (مجموعه 2) به عامل اجازه می‌دهد که به تدریج و با ثبات بیشتری به سمت بهره‌برداری برود، اما بازدهی نهایی ممکن است کمتر از کاهش سریع‌تر باشد.

**برای مدل 2 DQN**

نتایج اول

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: 53.266

Episode 992 finished after 6 steps with total reward 95

Episode 993 finished after 6 steps with total reward 95

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

نتایج دوم

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: 8.06

Episode 877 finished after 7 steps with total reward 145

Episode 993 finished after 6 steps with total reward 95

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 6 steps with total reward 95

**مقایسه:**

**EPS\_DECAY = 200**

پاداش تجمعی صاف و صعودی است، به این معنی که عامل به تدریج بهبود یافته و پاداش‌ها به صورت یکنواخت افزایش یافته‌اند.

از اقدامات پر ریسک اجتناب کرده و پاداش‌های کمتری اما پایدارتری دریافت کرده است. توانسته به پاداش 95 همگرا شود اما موفق به کسب حداکثر پاداش نشده است.

اکتشاف و بهره ‌برداری: مقدار EPS\_DECAY = 200 نشان‌دهنده این است که عامل نسبتاً سریع از حالت اکتشاف به حالت بهره‌برداری منتقل می‌شود. در نتیجه، عامل در ابتدا اقدامات مختلف را آزمایش می‌کند و سپس به سرعت به استفاده از تجربیات قبلی و انتخاب بهترین اقدامات براساس اطلاعات موجود می‌پردازد. این منجر به کاهش اقدامات پر ریسک و همگرایی سریع‌تر به پاداش‌های پایدار می‌شود.

**EPS\_DECAY = 800**

عامل توانسته پاداش‌های بالاتری (مثلاً 145) دریافت کند که نشان‌دهنده موفقیت در اکتشاف است. پاداش تجمعی به صورت صعودی بالا رفته که نشان‌دهنده بهبود عملکرد عامل است. نوسانات بیشتری در پاداش‌ها مشاهده شده که نشان‌دهنده اقدامات ریسکی بیشتر است. میانگین پاداش نهایی کمتر از مجموعه اول است. میانگین پاداش در هر اپیزود پس از 1000 اپیزود: 8.06

BATCH\_SIZE:

در هر سه مجموعه مقدار 64 بوده که تعادلی مناسب بین حجم داده‌ها و سرعت یادگیری ایجاد کرده است.

GAMMA:

در هر سه مجموعه مقدار 0.99 بوده که نشان‌دهنده اهمیت بالای پاداش‌های آینده است.

EPS\_START و EPS\_END

بر اساس تحلیل‌های انجام شده، مقدار EPS\_DECAY = 200 بهترین عملکرد را ارائه می‌دهد. این تنظیمات به عامل اجازه می‌دهد تا به تعادل مناسبی بین اکتشاف و بهره‌برداری دست یابد. عامل در ابتدا به کشف وضعیت‌ها و اقدامات جدید می‌پردازد و سپس به سرعت به استفاده از تجربیات قبلی و انتخاب بهترین اقدامات بر اساس اطلاعات موجود می‌پردازد. این منجر به کاهش اقدامات پر ریسک و همگرایی سریع‌تر به پاداش‌های پایدار می‌شود. در مقابل، مقادیر بسیار پایین یا بسیار بالای EPS\_DECAY منجر به عملکرد ضعیف‌تر و نوسانات بیشتر در پاداش‌ها می‌شوند.

**برای QL**

  EPSILON = 0.9

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -429.221

First episode where agent consistently finds gold for consecutive episodes: 656

Q-learning Episode 728/1000, Total Reward: 145

Q-learning Episode 992/1000, Total Reward: 95

Q-learning Episode 993/1000, Total Reward: 95

Q-learning Episode 994/1000, Total Reward: 95

Q-learning Episode 995/1000, Total Reward: 95

Q-learning Episode 996/1000, Total Reward: 95

Q-learning Episode 997/1000, Total Reward: 95

Q-learning Episode 998/1000, Total Reward: 95

Q-learning Episode 999/1000, Total Reward: 95

Q-learning Episode 1000/1000, Total Reward: 95

**QL**

میانگین پاداش:

میانگین پاداش در هر اپیزود پس از 1000 اپیزود: -429.221

اپیزودهای ابتدایی با مشکلات زیادی روبرو شده‌اند که منجر به پاداش‌های منفی شده است.

از اپیزود 656 به بعد، عملکرد به طور قابل توجهی بهبود یافته و پاداش‌های مثبت بیشتری کسب شده است.

اپیزودهای موفقیت‌آمیز:

اولین اپیزودی که عامل به طور پیوسته طلا را پیدا کرده: 656

در اپیزود 728، عامل به پاداش 145 دست یافته است.

در اپیزودهای 989 تا 1000، عامل به طور مداوم پاداش 95 را کسب کرده است.

نمودارها:

نمودار پاداش برای هر اپیزود نشان می‌دهد که پس از اپیزود 656 بهبود چشمگیری در عملکرد رخ داده است.

نمودار پاداش تجمعی نشان می‌دهد که ابتدا کاهشی بوده و پس از اپیزود 656 افزایشی شده است.

نمودار میانگین پاداش برای هر اپیزود نیز نشان می‌دهد که میانگین پاداش به طور پیوسته در حال افزایش است.

برای EPSILON=0.8

|  |
| --- |
|  |

Q-learning Episode 790/1000, Total Reward: 145

Average reward per episode after 1000 episodes: -525.653

First episode where agent consistently finds gold for consecutive episodes: 791

Q-learning Episode 991/1000, Total Reward: 95

Q-learning Episode 992/1000, Total Reward: 95

Q-learning Episode 993/1000, Total Reward: 95

Q-learning Episode 994/1000, Total Reward: 95

Q-learning Episode 995/1000, Total Reward: 95

Q-learning Episode 996/1000, Total Reward: 95

Q-learning Episode 997/1000, Total Reward: 95

Q-learning Episode 998/1000, Total Reward: 95

Q-learning Episode 999/1000, Total Reward: 95

Q-learning Episode 1000/1000, Total Reward: 95

نمودار پاداش بر حسب اپیزود:

از اپیزود 790 به بعد، نمودار پاداش به 145 رسیده و سپس به ثبات در مقدار 95 رسیده است تا انتهای تعداد اپیزودها (1000). این نشان می‌دهد که مدل در این نقطه از تمرکز خود به طور قابل توجهی بهبود یافته است و به دنبال وظیفه پیدا کردن طلا به صورت مداوم موفقیت آمیز است.

نمودار پاداش تجمعی:

نمودار پاداش تجمعی نیز نشان می‌دهد که تا اپیزود 790 کاهشی وجود داشته، اما سپس با افزایش مستمر به یک مقدار ثابت از پاداش 95 رسیده است. این نشان می‌دهد که مدل بهبود یافته و توانسته است استحکام خود را در به دست آوردن طلا به طور مداوم تقویت کند.

نمودار میانگین هر پاداش در اپیزود:

این نمودار نشان می‌دهد که در ابتدا، پاداش هر اپیزود نوساناتی داشته است، که این ممکن است ناشی از این باشد که مدل هنوز به یک سیاست بهینه نرسیده است و در فرایند یادگیری مشکلاتی داشته است. اما پس از اپیزود 100، با افزایش نرمی و سپس کاهشی که در ابتدای 600 اتفاق می‌افتد، مدل به یک سیاست بهینه نزدیک شده و پاداش هر اپیزود به صورت صعودی و صاف به بالا می‌رود.

مدل Q-learning با انتخاب EPSILON=0.8 و بهره‌گیری از یادگیری غیرخطی، به طور کلی موفقیت آمیز بوده است. به طور خاص، از اپیزود 790 به بعد، مدل به پیدا کردن طلا به صورت مداوم موفق شده است که نشان از یادگیری قابل قبولی است. از دیدگاه نمودار پاداش تجمعی، مدل به تسلط بر یافتن طلا در محیط Wumpus World رسیده است و پایداری در عملکرد خود نشان داده است.

مقایسه

برای EPSILON = 0.8:

میانگین پاداش در هر اپیزود:

میانگین پاداش پس از 1000 اپیزود حدوداً -525.653 بوده است. این نشان می‌دهد که در ابتدا، مدل با نوساناتی مواجه بوده که ممکن است ناشی از استفاده از EPSILON بالا باشد که به تصادفی‌ترین حرکات منجر شود.

از اپیزود 790 به بعد، مدل به پاداش 145 رسیده و سپس به ثبات در مقدار 95 رسیده است، که نشان از یادگیری بهبود یافته و تمایل به انجام اقدامات مفید‌تر دارد.

نمودار پاداش تجمعی:

نمودار نشان می‌دهد که پس از اپیزود 790، مدل به طور مداوم پیشرفت داشته و به یک سطح پایدار از پاداش 95 رسیده است. این نمودار نشان می‌دهد که مدل توانسته است تاکتیک‌های بهتری را برای جمع‌آوری طلا اجرا کند.

نمودار میانگین پاداش در اپیزود:

این نمودار نشان می‌دهد که پس از یادگیری اولیه و تنظیم EPSILON، میانگین پاداش به طور مداوم بهبود یافته و به یک مقدار قابل قبول نزدیک شده است.

برای EPSILON = 0.9:

میانگین پاداش در هر اپیزود:

میانگین پاداش پس از 1000 اپیزود حدوداً -429.221 بوده است، که نسبت به EPSILON=0.8 بهبود داشته است. این نشان می‌دهد که EPSILON بالاتر باعث می‌شود عامل بیشتر به اکتشاف و کشف تاکتیک‌های بهتر بپردازد.

از اپیزود 656 به بعد، عملکرد به طور قابل توجهی بهبود یافته و پاداش‌های مثبت بیشتری کسب شده است.

نمودار پاداش تجمعی:

پس از اپیزود 656، نمودار پاداش تجمعی نشان می‌دهد که عملکرد به طور چشمگیری بهبود یافته و به سطح بالاتری از پاداش رسیده است. این نمودار نشان می‌دهد که با افزایش EPSILON، مدل به سرعت توانسته است به سیاست بهینه نزدیک شود.

نمودار میانگین پاداش در اپیزود:

مدل با استفاده از EPSILON بالا، نسبت به ابتدا به سرعت بهبود یافته و میانگین پاداش در اپیزودها به صورت پیوسته افزایش یافته است، نشان از تاثیر مثبت EPSILON بالا بر یادگیری دارد.

یر نرخ اکتشاف اپسیلون بر فرآیند یادگیری Q-learning با EPSILON=0.8 و EPSILON=0.9

در اینجا به بررسی تأثیر نرخ اکتشاف اپسیلون بر فرآیند یادگیری Q-learning در دو حالت مختلف (EPSILON=0.8 و EPSILON=0.9) می‌پردازیم:

**مقایسه**

EPSILON=0.9

نرخ اکتشاف اپسیلون:

اپسیلون اولیه: 0.9

نتایج و مشاهده‌ها:

اکتشاف بیشتر: با شروع اپسیلون نسبتاً بالا (0.9)، عامل بیشتر به اکتشاف محیط پرداخته است. این به معنای جمع‌آوری داده‌های متنوع از محیط و افزایش شانس یافتن مسیرهای بهینه است.

میانگین پاداش: میانگین پاداش در هر اپیزود پس از 1000 اپیزود برابر با است. این مقدار منفی نشان‌دهنده این است که عامل در مراحل اولیه آموزش با مشکلات زیادی مواجه بوده است.

بهبود عملکرد: از اپیزود 656 به بعد، عامل به طور پیوسته طلا را پیدا کرده و عملکردش بهبود یافته است. این نشان می‌دهد که پس از اکتشاف زیاد در مراحل اولیه، عامل توانسته است به تدریج به استراتژی‌های بهینه برسد.

پایداری در عملکرد: در اپیزودهای پایانی، عامل به طور مداوم پاداش 95 را کسب کرده است که نشان‌دهنده پایداری در عملکرد است.

EPSILON=0.8

نرخ اکتشاف اپسیلون:

اپسیلون اولیه: 0.8

نتایج و مشاهده‌ها:

اکتشاف کمتر: با شروع اپسیلون کمی پایین‌تر (0.8)، عامل کمی کمتر به اکتشاف محیط پرداخته است. این ممکن است باعث شود که عامل زودتر به استراتژی‌های بهره‌برداری از دانسته‌های خود بپردازد.

میانگین پاداش: میانگین پاداش در هر اپیزود پس از 1000 اپیزود برابر با -525.653 است. این مقدار نشان‌دهنده این است که عامل در مراحل اولیه آموزش با مشکلات بیشتری مواجه بوده است.

بهبود عملکرد: از اپیزود 790 به بعد، عملکرد عامل بهبود یافته و به طور پیوسته طلا را پیدا کرده است. این نشان می‌دهد که با کاهش تدریجی اپسیلون، عامل به تدریج به استراتژی‌های بهینه رسیده است.

پایداری در عملکرد: در اپیزودهای پایانی، عامل به طور مداوم پاداش 95 را کسب کرده است که نشان‌دهنده پایداری در عملکرد است.

مقایسه تأثیر دو نرخ اپسیلون

اکتشاف اولیه:

در هر دو حالت، با شروع اپسیلون نسبتاً بالا (0.9 و 0.8)، عامل به اکتشاف محیط پرداخته و داده‌های متنوع جمع‌آوری کرده است.

میانگین پاداش:

عامل با اپسیلون 0.9 میانگین پاداش بالاتری نسبت به عامل با اپسیلون 0.8 دارد. این نشان می‌دهد که اکتشاف بیشتر اولیه (اپسیلون 0.9) به عامل کمک کرده تا داده‌های بهتری برای یادگیری جمع‌آوری کند و در نهایت به عملکرد بهتری برسد.

بهبود عملکرد:

عامل با اپسیلون 0.9 زودتر از عامل با اپسیلون 0.8 به بهبود عملکرد دست یافته و از اپیزود 656 به بعد به طور پیوسته طلا را پیدا کرده است. در حالی که عامل با اپسیلون 0.8 از اپیزود 790 به بعد به این عملکرد رسیده است.

پایداری در عملکرد:

در هر دو حالت، عامل در اپیزودهای پایانی به طور مداوم پاداش 95 را کسب کرده است که نشان‌دهنده پایداری در عملکرد و یادگیری موفق است.

نتیجه‌گیری

اکتشاف بیشتر اولیه (EPSILON=0.9) به عامل کمک کرده است تا داده‌های متنوع‌تری جمع‌آوری کند و در نهایت به استراتژی‌های بهینه‌تر و میانگین پاداش بالاتری دست یابد.

اکتشاف کمتر اولیه (EPSILON=0.8) باعث شده است که عامل زودتر به بهره‌برداری از دانسته‌های خود بپردازد، اما میانگین پاداش نهایی کمتر از حالت اپسیلون 0.9 است.

# د)کارایی یادگیری

# چند اپیزود طول کشید تا عامل Q-Learning به طور مداورم و بدون افتادن در گودال یا خورده شدن توسط Wumps پیدا کند؟

عامل در اپیزود 656 توانسته به طور پیوسته به طلا دست پیدا کند. و بدون افتادن در گودال یا خورده شدن توسط Wumps پیدا کند.

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -429.221

First episode where agent consistently finds gold for consecutive episodes: 656

Q-learning Episode 728/1000, Total Reward: 145

Q-learning Episode 989/1000, Total Reward: 95

Q-learning Episode 990/1000, Total Reward: 95

Q-learning Episode 991/1000, Total Reward: 95

Q-learning Episode 992/1000, Total Reward: 95

Q-learning Episode 993/1000, Total Reward: 95

Q-learning Episode 994/1000, Total Reward: 95

Q-learning Episode 995/1000, Total Reward: 95

Q-learning Episode 996/1000, Total Reward: 95

Q-learning Episode 997/1000, Total Reward: 95

Q-learning Episode 998/1000, Total Reward: 95

Q-learning Episode 999/1000, Total Reward: 95

Q-learning Episode 1000/1000, Total Reward: 95

میانگین پاداش:

میانگین پاداش در هر اپیزود پس از 1000 اپیزود برابر با بوده است. این نشان می‌دهد که عامل در بسیاری از اپیزودها با مشکلاتی مانند افتادن در چاله‌ها یا خورده شدن توسط Wumpus مواجه شده است.

با این حال، در اپیزودهای پایانی به نظر می‌رسد که عامل توانسته بهبود یابد و پاداش‌های مثبت بیشتری کسب کند.

اپیزودهای موفقیت‌آمیز:

عامل در اپیزود 656 توانسته به طور پیوسته به طلا دست پیدا کند. این نشان می‌دهد که عامل پس از تمرین‌های متعدد، استراتژی مناسبی برای رسیدن به طلا پیدا کرده است.

اپیزودهای پایانی:

در اپیزودهای 989 تا 1000، عامل توانسته است به طور مداوم پاداش 95 را کسب کند که نشان می‌دهد عامل به استراتژی موفقیت‌آمیزی برای رسیدن به طلا و جلوگیری از خطرات دست یافته است.

نمودار پاداش برای هر اپیزود:

این نمودار نشان می‌دهد که عامل در ابتدا با مشکلاتی مواجه شده است و پاداش‌های منفی زیادی دریافت کرده است. با این حال، از اپیزود 656 به بعد، عملکرد عامل به طور قابل توجهی بهبود یافته و پاداش‌های مثبت بیشتری دریافت کرده است.

در اپیزود 728، عامل به پاداش 145 دست یافته است که نشان‌دهنده عملکرد بسیار موفقیت‌آمیز در آن اپیزود است.که توانسته wumps را بکشه و به ماکزیمم جایزه برسه.

نمودار پاداش تجمعی:

این نمودار نشان می‌دهد که پاداش تجمعی عامل در ابتدا کاهشی بوده است که نشان‌دهنده مشکلات و چالش‌های زیاد در اپیزودهای اولیه است.

پس از اپیزود 656، نمودار پاداش تجمعی افزایشی می‌شود که نشان‌دهنده بهبود عملکرد عامل و کسب پاداش‌های مثبت بیشتر است.

میانگین پاداش برای هر اپیزود:

این نمودار نشان می‌دهد که میانگین پاداش عامل به طور پیوسته در حال افزایش است. این افزایش از اپیزود 656 به بعد شیب بیشتری دارد که نشان‌دهنده بهبود قابل توجه عملکرد عامل است.

بهبود عملکرد عامل: عامل پس از تمرین‌های متعدد و تجربه‌های مختلف توانسته است به استراتژی مناسبی برای رسیدن به طلا و اجتناب از خطرات دست یابد. این بهبود به وضوح در نمودارهای پاداش تجمعی و میانگین پاداش برای هر اپیزود قابل مشاهده است.

تأثیر آموزش: در اپیزودهای اولیه، عامل مشکلات زیادی داشته است که منجر به پاداش‌های منفی شده است. اما با گذشت زمان و افزایش تجربه، عامل توانسته است این مشکلات را پشت سر بگذارد و به پاداش‌های مثبت دست یابد.

پیوستگی در عملکرد: در اپیزودهای پایانی، عامل به طور مداوم پاداش 95 را کسب کرده است که نشان‌دهنده پیوستگی در عملکرد موفقیت‌آمیز عامل است.

نتایج نشان می‌دهد که عامل توانسته است با استفاده از الگوریتم Q-learning به عملکرد موفقیت‌آمیزی دست یابد و به اهداف تعیین شده در محیط Wumpus World برسد.

# کارایی یادگیری Q-Learning و DQN را مقایسه کنید. کدام یک policy بهینه را سریع تر یادگرفت؟

**DQN**

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: 29.059

Episode 893 finished after 7 steps with total reward 145

Episode 978 finished after 6 steps with total reward 95

Episode 979 finished after 6 steps with total reward 95

Episode 980 finished after 6 steps with total reward 95

Episode 981 finished after 6 steps with total reward 95

Episode 984 finished after 7 steps with total reward 94

Episode 985 finished after 6 steps with total reward 95

Episode 986 finished after 6 steps with total reward 95

Episode 987 finished after 6 steps with total reward 95

Episode 988 finished after 6 steps with total reward 95

Episode 989 finished after 6 steps with total reward 95

Episode 990 finished after 6 steps with total reward 95

Episode 991 finished after 6 steps with total reward 95

Episode 992 finished after 6 steps with total reward 95

Episode 993 finished after 6 steps with total reward 95

Episode 994 finished after 6 steps with total reward 95

Episode 995 finished after 6 steps with total reward 95

Episode 996 finished after 6 steps with total reward 95

Episode 997 finished after 6 steps with total reward 95

Episode 998 finished after 6 steps with total reward 95

Episode 999 finished after 6 steps with total reward 95

Episode 1000 finished after 7 steps with total reward 94

**QL**

|  |
| --- |
|  |

Average reward per episode after 1000 episodes: -429.221

First episode where agent consistently finds gold for consecutive episodes: 656

Q-learning Episode 728/1000, Total Reward: 145

Q-learning Episode 992/1000, Total Reward: 95

Q-learning Episode 993/1000, Total Reward: 95

Q-learning Episode 994/1000, Total Reward: 95

Q-learning Episode 995/1000, Total Reward: 95

Q-learning Episode 996/1000, Total Reward: 95

Q-learning Episode 997/1000, Total Reward: 95

Q-learning Episode 998/1000, Total Reward: 95

Q-learning Episode 999/1000, Total Reward: 95

Q-learning Episode 1000/1000, Total Reward: 95

**QL**

میانگین پاداش:

میانگین پاداش در هر اپیزود پس از 1000 اپیزود: -429.221

اپیزودهای ابتدایی با مشکلات زیادی روبرو شده‌اند که منجر به پاداش‌های منفی شده است.

از اپیزود 656 به بعد، عملکرد به طور قابل توجهی بهبود یافته و پاداش‌های مثبت بیشتری کسب شده است.

اپیزودهای موفقیت‌آمیز:

اولین اپیزودی که عامل به طور پیوسته طلا را پیدا کرده: 656

در اپیزود 728، عامل به پاداش 145 دست یافته است.

در اپیزودهای 989 تا 1000، عامل به طور مداوم پاداش 95 را کسب کرده است.

نمودارها:

نمودار پاداش برای هر اپیزود نشان می‌دهد که پس از اپیزود 656 بهبود چشمگیری در عملکرد رخ داده است.

نمودار پاداش تجمعی نشان می‌دهد که ابتدا کاهشی بوده و پس از اپیزود 656 افزایشی شده است.

نمودار میانگین پاداش برای هر اپیزود نیز نشان می‌دهد که میانگین پاداش به طور پیوسته در حال افزایش است.

DQN:

میانگین پاداش:

میانگین پاداش در هر اپیزود پس از 1000 اپیزود: 29.059

عامل در اپیزودهای ابتدایی با مشکلات کمتری مواجه شده و توانسته به پاداش‌های مثبت بیشتری دست یابد.

اپیزودهای موفقیت‌آمیز:

اپیزود 893: پاداش 145 کسب شده است.

در اپیزودهای 987 تا 1000، عامل به طور مداوم پاداش 95 را کسب کرده است.

عامل از اپیزود 180 به بعد تقریبا به طور مداوم به طلا دست یافته است.

نمودارها:

نمودار پاداش برای هر اپیزود نشان می‌دهد که عامل از اپیزود 180 به بعد به طور مداوم به طلا می‌رسد، ولی گاهی تصمیمات اشتباهی دارد که منجر به پاداش‌های منفی می‌شود.

نمودار پاداش تجمعی ابتدا کاهش و سپس افزایش داشته و در اپیزود170 مسیر مجدداً کاهش یافته و سپس افزایش زیادی داشته است.

نمودار میانگین پاداش برای هر اپیزود به صورت لگاریتمی افزایش یافته ولی در اپیزود 170 کمی کاهش داشته و سپس دوباره افزایش یافته است.

سرعت یادگیری: DQN به وضوح سریع‌تر از Q-Learning policy بهینه را یادگرفته است. عامل DQN در اپیزود 180 به طور مداوم به طلا دست یافته در حالی که Q-Learning در اپیزود 656 به این سطح از عملکرد رسیده است.

پایداری عملکرد: هر دو عامل در اپیزودهای پایانی به طور مداوم پاداش 95 را کسب کرده‌اند، ولی DQN به طور کلی در یادگیری و دستیابی به پاداش‌های مثبت پایدارتر عمل کرده است.

پاداش تجمعی و میانگین پاداش: نمودارهای پاداش تجمعی و میانگین پاداش نشان می‌دهند که عملکرد DQN به مرور زمان بهتر شده و پس از یادگیری، به طور پایدار پاداش‌های مثبت بیشتری کسب کرده است.

DQN عملکرد بهتری نسبت به Q-Learning داشته و توانسته است policy بهینه را سریع‌تر و پایدارتر یاد بگیرد. نمودارهای پاداش نیز نشان می‌دهند که DQN در اپیزودهای ابتدایی عملکرد بهتری داشته و به طور مداوم به اهداف خود رسیده است.

تحلیل کارایی یادگیری: Q-learning:

در ابتدا با پاداش‌های متوسط پایین مواجه شد، که نشان‌دهنده‌ی سختی در یادگیری سیاست‌های بهینه است. عملکرد به طور قابل توجهی در اپیزود 656 بهبود یافت، که نشان‌دهنده‌ی موفقیت اولیه و پیوسته در پیدا کردن طلا است. در اپیزودهای نهایی، پاداش‌های پایدار و بالایی (95) را کسب کرد، که نشان‌دهنده‌ی یادگیری موفقیت‌آمیز استراتژی‌های بهینه است. DQN:

طی اپیزودها، بهبود پایدار و مستحکم‌تری در عملکرد نشان داد. پاداش‌های بالا (145) را نسبتاً زودتر در اپیزود 893 کسب کرد. در اپیزودهای نهایی، پاداش‌های بالایی (95) را با کمترین گام‌ها حفظ کرد، که نشان‌دهنده‌ی یک فرآیند یادگیری موثرتر است. مقایسه و نتیجه‌گیری: سرعت یادگیری:

DQN سریع‌تر یاد گرفت، پاداش‌های بالا و عملکرد پایدار را زودتر نسبت به Q-learning کسب کرد. Q-learning طولانی‌تر طول کشید تا پایدار شود و پاداش‌های بالای پیوسته را کسب کند. میانگین پاداش:

DQN پس از 1000 اپیزود، میانگین پاداش بیشتری در هر اپیزود (29.059) نسبت به Q-learning (-429.221) داشت. کارایی:

DQN در یادگیری سیاست بهینه موثرتر بود، که با عملکرد بالاتر و پیوسته از ابتدا نشان داده شد. تأثیر نرخ کشف اپسیلون: Q-learning با EPSILON=0.8 و EPSILON=0.9:

اپسیلون بالا (بررسی بیشتر):

تشویق به بررسی، کمک می‌کند تا عامل زوج‌های حالت-عمل بیشتری را کشف کند. مراحل اولیه پاداش‌های متنوع و اغلب پایین‌تری را به دلیل بررسی اقدامات زیربهینه نشان می‌دهد. مثال: Q-learning با EPSILON=0.9 پس از 1000 اپیزود، میانگین پاداش -429.221 داشت، که نشان‌دهنده‌ی بررسی گسترده اما پاداش‌های فوری پایین‌تر است. اپسیلون پایین (استفاده بیشتر):

تمرکز بر استفاده از اطلاعات معروف، منجر به پاداش‌های پایدار و بالاتر می‌شود. همانطور که آموزش پیش می‌رود، کاهش اپسیلون عملکرد را پایدار می‌کند. مثال: عامل Q-learning در اپیزودهای نهایی، پاداش‌های پیوسته 95 را کسب کرد، که نشان‌دهنده‌ی استفاده موفقیت‌آمیز از استراتژی‌های یاد گرفته شده است. مشاهدات:

وقتی اپسیلون بالا بود، هر دو عامل عملکرد متغیرتری به دلیل اقدامات بررسی نشان دادند. همانطور که اپسیلون کاهش یافت، هر دو عامل عملکرد پایدار و بالاتری را نشان دادند، استفاده از سیاست‌های بهینه یاد گرفته شده را به طور موثر انجام دادند.

DQN نسبت به Q-learning، کارایی یادگیری برتری نشان داد. تأثیر نرخ کشف اپسیلون در تعادل بین بررسی و استفاده بود، که در نهایت منجر به بهبود و عملکرد پایدار با کاهش اپسیلون شد.

# ه)معماری مورد استفاده ازبرای عاملان DQN را شرح دهید. چرا این معماری را انتخاب کردید؟

بررسی معماری مورد استفاده برای عاملان DQN

معماری DQN مدل 1:

class DQN(nn.Module):

def \_\_init\_\_(self, n\_observations, n\_actions):

super(DQN, self).\_\_init\_\_()

self.fc1 = nn.Linear(n\_observations, 128)

self.fc2 = nn.Linear(128, 64)

self.fc3 = nn.Linear(64, n\_actions)

def forward(self, x):

x = torch.relu(self.fc1(x))

x = torch.relu(self.fc2(x))

return self.fc3(x)

این مدل شامل:

یک لایه‌ی ورودی با 128 نورون است که به تعداد مشاهدات ورودی وصل است.

یک لایه‌ی مخفی با 64 نورون.

یک لایه‌ی خروجی که به تعداد اقدامات محیط (actions) بستگی دارد.

معماری DQN مدل 2:

class DQN(nn.Module):

def \_\_init\_\_(self, n\_observations, n\_actions):

super(DQN, self).\_\_init\_\_()

self.fc1 = nn.Linear(n\_observations, 24)

self.fc2 = nn.Linear(24, 24)

self.fc3 = nn.Linear(24, n\_actions)

def forward(self, x):

x = torch.relu(self.fc1(x))

x = torch.relu(self.fc2(x))

return self.fc3(x)

این مدل شامل:

یک لایه‌ی ورودی با 24 نورون است که به تعداد مشاهدات ورودی وصل است.

یک لایه‌ی مخفی دیگر با 24 نورون.

یک لایه‌ی خروجی که به تعداد اقدامات محیط (actions) بستگی دارد.

دلیل انتخاب معماری‌ها:

مدل 1: این معماری با تعداد نورون‌های بیشتر در لایه‌های ورودی و مخفی، ظرفیت بیشتری برای یادگیری روابط پیچیده در داده‌ها دارد. این امر می‌تواند به بهبود عملکرد عامل در یادگیری سیاست بهینه منجر شود.

مدل 2: این معماری با تعداد نورون‌های کمتر، ساده‌تر و سبک‌تر است. این می‌تواند به یادگیری سریع‌تر اما با دقت کمتر منجر شود.

نتایج نشان دادند که افزایش تعداد نورون‌ها در لایه‌های مخفی مدل 1 بهبود قابل توجهی در عملکرد عامل DQN داشته است. به طور خاص:

مدل 1: به طور کلی میانگین پاداش‌ها و نرخ یادگیری سیاست بهینه در این مدل بهتر بوده است.

مدل 2: عملکرد خوبی داشته اما به طور مستمر و با دقت کمتر سیاست بهینه را یادگرفته است.

افزایش نورون‌ها: با افزایش تعداد نورون‌ها، مدل قابلیت بیشتری برای یادگیری الگوهای پیچیده‌تر دارد که می‌تواند به بهبود عملکرد در محیط‌های پیچیده‌تر منجر شود. این امر در نتایج بهبود یافته مدل 1 مشهود است.

زمان آموزش: مدل 2 به دلیل سادگی، ممکن است زمان آموزش کمتری نیاز داشته باشد، اما دقت کمتری نیز دارد.

کارایی نهایی: مدل 1 توانست سیاست بهینه را سریع‌تر و با دقت بیشتری یاد بگیرد، در حالی که مدل 2 در محیط‌های ساده‌تر ممکن است عملکرد قابل قبولی داشته باشد اما در محیط‌های پیچیده‌تر کم‌کارآمدتر باشد.

افزایش تعداد نورون‌ها در مدل‌های DQN، بهبود قابل توجهی در یادگیری سیاست بهینه و عملکرد نهایی عامل داشته است. این امر نشان می‌دهد که در محیط‌های پیچیده‌تر، استفاده از مدل‌هایی با ظرفیت بیشتر می‌تواند نتایج بهتری به همراه داشته باشد.

در مورد مدل اول

لایه‌های ورودی و مخفی:

لایه ورودی (fc1) این لایه یک لایه خطی (fully connected) است که تعداد نورون‌های آن برابر با تعداد مشاهدات (n\_observations)و خروجی آن 128 نورون است. این لایه وظیفه دارد تا داده‌های ورودی (مشاهدات محیط) را به یک فضای ویژگی با ابعاد بالاتر منتقل کند.

لایه مخفی اول (fc2): این لایه نیز یک لایه خطی است که ورودی آن 128 نورون و خروجی آن 64 نورون است. این لایه به مدل کمک می‌کند تا ویژگی‌های پیچیده‌تری را یاد بگیرد.

لایه مخفی دوم (fc3): این لایه خروجی نهایی مدل است که تعداد نورون‌های آن برابر با تعداد اقدامات (n\_actions) در محیط است. این لایه وظیفه دارد تا ارزش هر اقدام (Q-value) را برای هر حالت ورودی محاسبه کند.

2. توابع فعال‌سازی:

در هر لایه مخفی از تابع فعال‌سازی ReLU استفاده شده است. ReLU به مدل کمک می‌کند تا ویژگی‌های غیرخطی را یاد بگیرد و به طور موثر مسائل یادگیری عمیق را حل کند. عملکرد ReLU به این صورت است که هر مقدار منفی را صفر می‌کند و مقادیر مثبت را بدون تغییر باقی می‌گذارد.

مزایا:

ظرفیت یادگیری بالا:

با داشتن تعداد نورون‌های بیشتر در لایه‌های مخفی، مدل قابلیت یادگیری روابط پیچیده‌تر در داده‌ها را دارد. این امر بهبود عملکرد مدل را در محیط‌های پیچیده تضمین می‌کند.

پایداری و انعطاف‌پذیری:

استفاده از ReLU به مدل کمک می‌کند تا سریع‌تر همگرا شود و از مشکلات اشباع شدن که در توابع فعال‌سازی دیگر مانند sigmoid وجود دارد، جلوگیری کند.

عمومیت‌پذیری بهتر:

معماری عمیق‌تر با تعداد نورون‌های بیشتر، باعث می‌شود مدل بتواند ویژگی‌های مهم‌تری از داده‌ها استخراج کند که در نتیجه عملکرد بهتری در محیط‌های متفاوت دارد.

نحوه کار:

پیش‌پردازش و ورودی داده‌ها:

مشاهدات محیط به عنوان ورودی به لایه fc1 داده می‌شود. این لایه ویژگی‌های اولیه را استخراج می‌کند و به فضای ویژگی 128 بعدی منتقل می‌کند.

پردازش ویژگی‌ها:

خروجی لایه اول به عنوان ورودی به لایه دوم (fc2) داده می‌شود. در این لایه، ویژگی‌های استخراج شده به ویژگی‌های پیچیده‌تری تبدیل می‌شوند و در نهایت به یک فضای 64 بعدی منتقل می‌شوند.

محاسبه ارزش اقدامات:

در نهایت، ویژگی‌های خروجی از لایه دوم به لایه سوم (fc3) داده می‌شود. این لایه ارزش (Q-value) هر اقدام را برای حالت ورودی محاسبه می‌کند. خروجی این لایه یک بردار با تعداد عناصر برابر با تعداد اقدامات محیط است.

انتخاب اقدام:

عامل بر اساس مقادیر Q-value محاسبه شده توسط لایه خروجی، اقدام بهینه را انتخاب می‌کند. این انتخاب ممکن است بر اساس سیاست epsilon-greedy باشد که در آن با احتمال epsilon یک اقدام تصادفی و با احتمال 1-epsilon بهترین اقدام انتخاب می‌شود.

این معماری به دلیل داشتن تعداد نورون‌های بیشتر و استفاده از توابع فعال‌سازی موثر مانند ReLU، عملکرد بهتری در یادگیری سیاست بهینه و تصمیم‌گیری در محیط‌های پیچیده دارد. این مدل با بهره‌گیری از ظرفیت بالای خود، قادر به یادگیری و استخراج ویژگی‌های مهم از داده‌ها است که منجر به بهبود عملکرد کلی در محیط‌های مختلف می‌شود.

|  |
| --- |
|  |

**Layer: fc1**

* Size: torch.Size([128, 16])
* Values:
* Bias:

**Layer: fc2**

* Size: torch.Size([64, 128])
* Values:
* Bias:

**Layer: fc3**

* Size: torch.Size([5, 64])
* Values:
* Bias:

tensor([-0.5684, -0.5613, -0.5686, -0.5488, -2.6766])