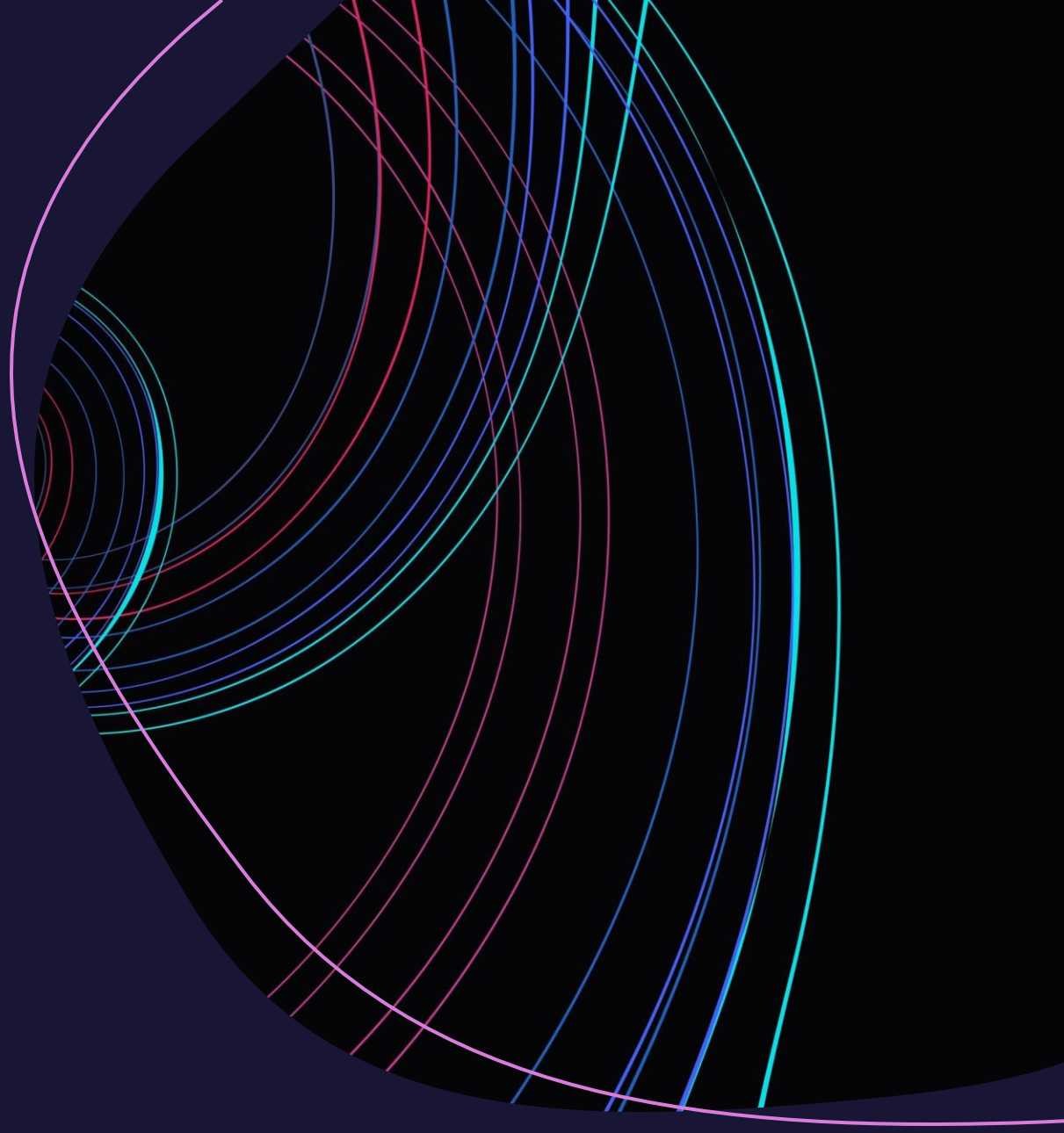
PacMan

- Voinescu David-Ioan
- Ghena-Ionescu Alexandru
- Rapcea Catalin
- Uta Mario Ernest
- Milosi Elias



Cuprins

Introducere

- **Pacman** e un joc în care trebuie să navighezi printr-un labirint, colectând pastile pentru a acumula puncte, evitând în același timp fantomele care te urmăresc. Obiectivul principal este să golești labirintul fără să fii prins, avansând astfel la următorul nivel și obținând un scor cât mai mare.
- **`"MsPacman-v4"`** face parte din **Arcade Learning Environment (ALE)** care oferă o implementare a jocului **Ms. Pac-Man**, cu următoarele caracteristici și detalii tehnice:
  - **Stările** sunt reprezentate sub formă de cadre video capturate din joc, oferind o descriere vizuală a mediului. Acestea conțin informații despre: **poziția lui Ms. Pac-Man, poziția fantomelor, obiectele din labirint rămase, starea item-urilor speciale**
  - **Spațiul de acțiuni** este reprezentat de cele **5 acțiuni** posibile pe care le poate efectua Pac-Man: **stop si deplasare in cele 4 directii**

Introducere

Jocul atribuie recompense bazate pe progresul jucătorului:

- **Recompense pozitive:**
  - Colectarea unei pastile oferă o recompensă mică pozitivă.
  - Colectarea unui fruct (item bonus) oferă o recompensă mai mare.
  - Consumul fantomelor după activarea unui power-up oferă o recompensă semnificativă.
- **Recompense negative:**
  - Pierderea unei vieți după ce Pac-Man este prinsă de o fantomă.
  - Recompensa finală reflectă scorul total acumulat la finalul unui episod.

Un episod se termină în două cazuri:

- Pac-Man își pierde toate viețile.
- Toate pastilele din labirint au fost colectate (nivel completat).
- **Scopul agentului:** Maximiza scorul acumulat într-un episod.

'Pseudo' Value Iteration

Value iteration este un algoritm fundamental de RL si programare dinamica. Acesta este folosit pentru a calcula politica optima si value function-ul pentru Procesele Decizionale Markov, calculand iterativ ecuatia lui Bellman

Jocul Pacman este mult prea complex pentru a aplica value iteration clasic, implementarea prezentata este ‘de creatie propie’ pentru a putea fi folosit.

'Pseudo' Value Iteration

Fata de value iteration clasic, nu stim toate starile jocului (nu stim chiar niciuna), fapt ce ne forteaza sa *exploram*

La inceputul fiecarui episode, pacman joaca o runda, initial complet aleator, pentru a afla cat mai multe stari. Pe parcursul antrenari, runda incepe sa respecte din ce in mai mult policy-ul

```
while not done:

    if s not in best_policy or random()<epsilon:
        a = env.action_space.sample()
        rand+=1
        #print("A mutat random")
        while not a:
            a = env.action_space.sample()
    else:
        #print("A gandit")
        think+=1
        a=best_policy[s]

    next_state, reward, done, _  = env.step(a)
    next_state=tuple(next_state.flatten())
    total_reward+=reward
    known_paths[s][a]={"reward":reward,"next_state":next_state}

    if next_state not in known_paths:
        known_paths[next_state]={}
        known_paths[next_state][0]={"reward":0,"next_state":0}

    s=next_state
    if done:
        break

print(f"Scorul dupa explorare {total_reward}")
print(f'Miscari gandite {think}\nMiscari aleatoare {rand}')
```

'Pseudo' Value Iteration

Dupa explorare, aplicam algoritmul de value iteration pe starile deja cunoscute(starile inainte de explorarea din episodul current). Ulterior aplicam policy improvement pentru a determina noua politica.

Rulam acesti pasi pentru fiecare episod, luand in calcul noile stari descoperite cu fiecare explorare

```
while maxChange > tol:
    numIters += 1
    maxChange = 0.0
    for s in states:
        bestActionValue = -np.inf
        for action in known_paths[s]:

            if known_paths[s][action]['next_state'] not in value_function:
                value_function[known_paths[s][action]['next_state']]=0

            value_for_thisAction = 0.0

            if known_paths[s][action]['next_state'] ==0:
                value_for_thisAction =  known_paths[s][action]['reward']
            else:
                value_for_thisAction = known_paths[s][action]['reward'] + gamma * value_function[known_paths[s][action]['next_state']]

            if value_for_thisAction > bestActionValue:
                bestActionValue = value_for_thisAction

        if s in value_function:
            maxChange = max(maxChange, abs(value_function[s] - bestActionValue))

        value_function[s] = bestActionValue


print(f"Value iteration converged after {numIters} steps\n")
policy = policy_improvement(states, nA, value_function, gamma)
return value_function, policy
```

```
def policy_improvement (states , nA , value_function, gamma=0.9):
    policy={}
    for s in states:
        value_per_action = np.zeros(shape=(nA,))
        for action in known_paths[s]:

            action_val =  known_paths[s][action]['reward'] + gamma * value_function[known_paths[s][action]["next_state"]]
            value_per_action[action] = action_val

        best_action = np.argmax(value_per_action)
        policy[s] = best_action

    return policy
```

'Pseudo' Value Iteration

Dupa ce rulam algoritmul timp de cateva episoade (20 mai exact), testam algoritmul si obtinem scorul de..... >2100 :D (Algoritmul aleatoriu are un average de 250).

Value Iteration nu este totusi cea mai buna varianta pentru Pacman, algoritmul mancand foarte mult RAM, jocul fiind prea complex. Algoritmii mai complexi de RL sunt mult mai potriviti pentru acest caz

DEEP Q-LEARNING

- Ce este DEEP Q-LEARNING?

O metoda avansata de Reinforcement Learning care combina Q-LEARNING cu retele neuronale convolutionale.

Permite luarea deciziilor in medii mai complexe, cum ar fi jocurile, unde starile sunt reprezentate de imagini.

- De ce nu am folosit Q-LEARNING simplu?

In Q-LEARNING clasic, tabelul Q devine impracticabil de mare pentru jocuri complexe.

Q-Learning simplu nu poate generaliza între stări similare, ceea ce înseamnă că trebuie să exploreze fiecare stare posibilă pentru a învăța.

Arhitectura si fluxul de lucru al algoritmului

- Cadrele video sunt transformate in imagini grayscale normalizate pentru a reduce complexitatea inputului.
- Agentul exploreaza mediul pentru a colecta perechi stare-acțiune-recompensă.
- Strategia epsilon-greedy este utilizată pentru a echilibra explorarea și exploatarea.
- Valorile Q sunt actualizate folosind TD error. Politica învățată este aplicată pentru a observa performanța agentului.

Avantajele folosirii DEEP Q-LEARNING

- Deep Q-Learning foloseste o retea neuronala pentru a aproxima functia Q, permitand invatarea in spatii de stari continue sau foarte mari.
- Q-Learning simplu nu are mecanisme pentru procesarea de intrari complexe, cum sunt imaginile jocului. Retelele neuronale convoluționale (CNN) utilizate în Deep Q-Learning extrag automat caracteristicile relevante.
- Deep Q-Learning implementeaza strategii avansate, cum ar fi epsilon-greedy cu scadere graduala, pentru a gasi un echilibru intre explorare si exploatare, ceea ce imbunatateste viteza de invatare.

Rezultatele algoritmului

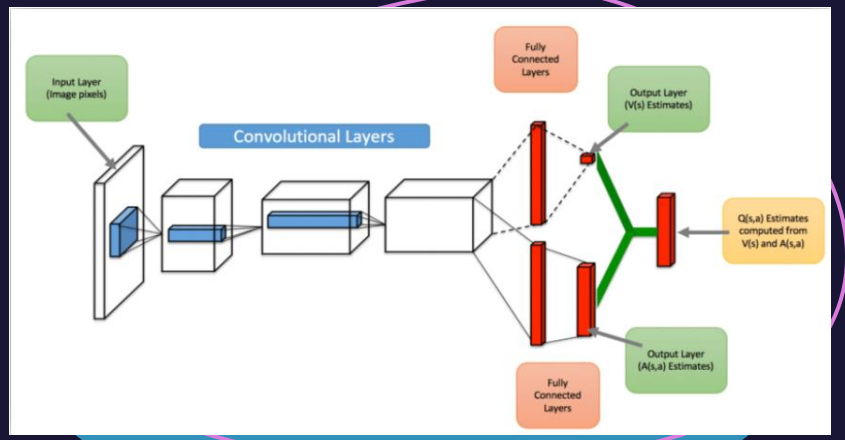
- Dupa rularea aproximativa a 500 episoade, algoritmul obtine un scor mediu de 500-600 si cel mai bun scor de 2200.
- Timpul de rulare al acestor episoade a fost aproximativ o ora si 20 minute (pe o masina virtuala cu un procesor Ryzen 7 6800H).

Ce poate fi imbunatatit?

- Learning rate, gamma si batch size.
- Implementarea unui buffer de experiente prioritizate ar permite modelului sa invete mai eficient din tranzitiile relevante.
- Antrenarea si testarea pe mai multe configuratii ale jocului pentru a valida generalizarea modelului.

Duel Q

- Arhitectura:
- Functia generala:
- Utilizează o rețea neurală cu două fluxuri (unul pentru valoare și unul pentru avantaj), care sunt combinate pentru a calcula Q(s, a).



- Poate învăța mai eficient în medii unde diferențele dintre acțiuni nu sunt foarte mari, deoarece valoarea unei stări este mai importantă decât avantajul specific al fiecărei acțiuni.

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

Duel Q

- **Valoarea stării (V(s))**: Este estimarea valorii generale a stării, adică cât de bună este starea în general, fără a lua în considerare acțiunile specifice.

- **Avantajul acțiunii (A(s, a))**: Măsoară cât de bună este o anumită acțiune comparativ cu celelalte posibile acțiuni într-o stare dată.

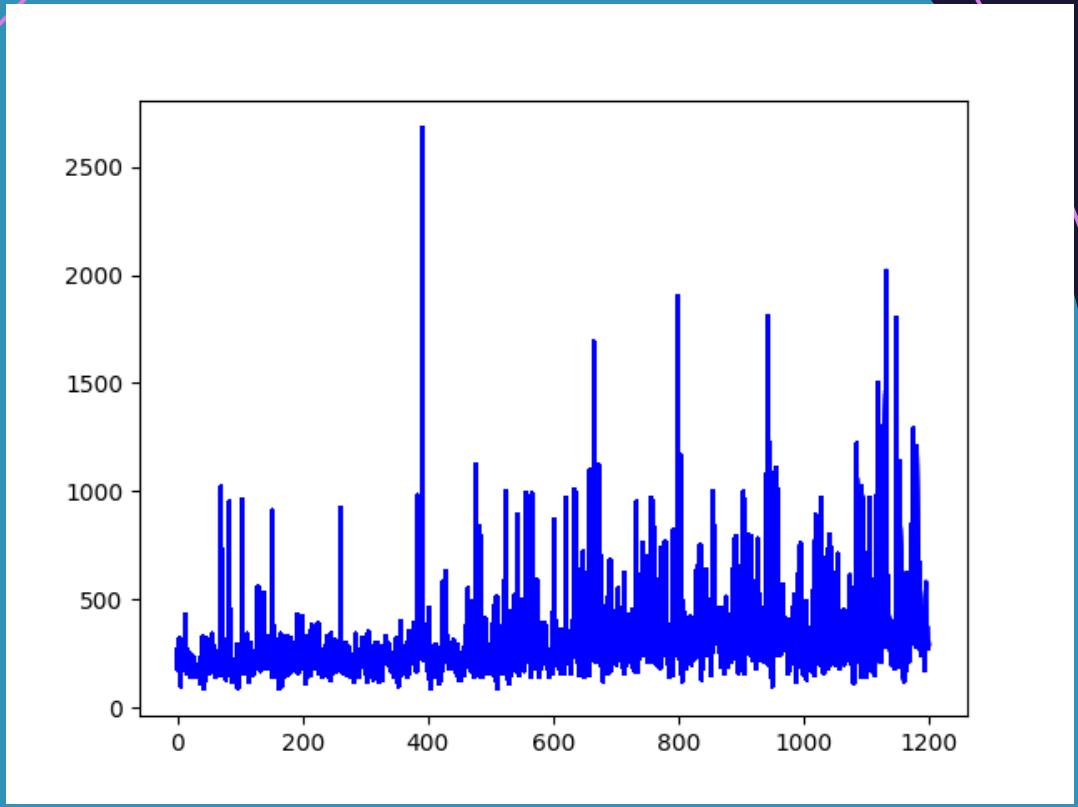
```
    def forward(self, x):  David Voinescu *
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = x.view( *shape: x.size(0), -1)
        x = torch.relu(self.fc(x))

        value = self.value(x)
        advantage = self.advantage(x)

        q_values = value + (advantage - advantage.mean(dim=1, keepdim=True))
        return q_values
```

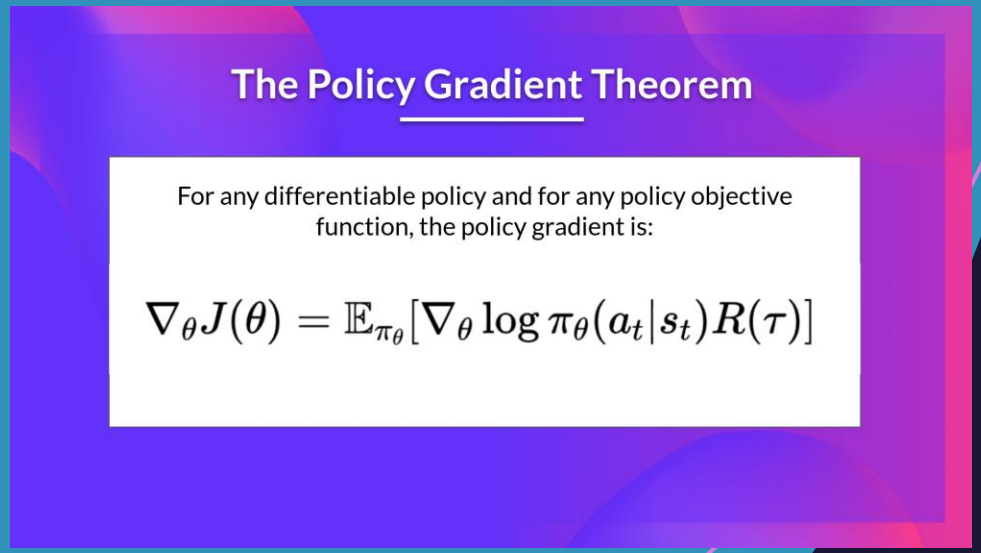
```
class DuelingDQN(nn.Module):  3 usages  David Voinescu *
    def __init__(self, action_space):  David Voinescu *
        super(DuelingDQN, self).__init__()
        self.conv1 = nn.Conv2d( in_channels: 4,  out_channels: 16, kernel_size=8, stride=4)
        self.conv2 = nn.Conv2d( in_channels: 16,  out_channels: 32, kernel_size=4, stride=2)
        self.fc = nn.Linear(32 * 432,  out_features: 128)
```

Duel Q



- Antrenat pe doar 1200 de episoade
- Durata variaza intre 15 secunde si 2 minute, in functie de frameskip)

Policy Gradient



Despre algoritm:

- Este o metodă din RL care ajusteaza direct parametrii unei politici pentru a maximiza recompensa cumulativa.
- Politica este reprezentata printr-o retea neuronala care ofera probabilitatile pentru fiecare actiune posibila.
- Agentul selectează acțiunile pe baza acestor probabilitati si foloseste gradientul pentru a actualiza reteaua.

Fluxul de lucru al programului

Preprocesare:

Observatiile mediului sunt transformate în imagini grayscale normalizate și redimensionate.

Antrenare:

Agentul colectează experiente prin explorare.

Calculam recompensa cumulativă folosind factorul de discount gamma.

Politica este ajustată prin metoda Policy Gradient și un termen de entropie care promoveazs explorarea.

Testare:

Agentul joaca Pac-Man, actiunile fiind selectate conform politicii invatate.

Avantaje ale algoritmului

Simplitate conceptuala:

Algoritmul ajusteaza direct probabilitatile actiunilor.

Adaptabilitate:

Poate fi folosit pentru o gama larga de jocuri si probleme.

Explorare crescuta:

Termenul de entropie încurajeaza agentul sa incerce strategii noi.

Dezavantaje si Rezultat

Esantionare :

- Necesita multe episoade pentru a invata o politica eficienta.

Sensibilitate la hiperparametri:

- Performanta depinde semnificativ de valori precum rata de învatare (lr) și factorul de discount (gamma).

Agentul a fost antrenat și testat pe mediul MsPacman-v4 din biblioteca OpenAI Gym.

Gamma: 0.89 (agentul prioritizează recompensele pe termen mediu).

Performanta a crescut gradual pe masura ce agentul a invatat sa evite fantomele si sa maximizeze punctajul.

Multumim pentru atentie!