ԲՈՎԱՆԴԱԿՈՒԹՅՈՒՆ

ՆԵՐԱԾՈՒԹՅՈՒՆ

Արհեստական բանականությամբ համակարգերի զարգացման հիմնական տարրերից մեկը ամրապնդմամբ ուսուցումն է. մեքենայական ուսուցման մեթոդ, որը ոգեշնչված է կենդանի օրգանիզմների վարքագծային հոգեբանությամբ, որը թույլ է տալիս գործակալներին օպտիմալ որոշումներ կայացնել տարբեր միջավայրերում:

Այս աշխատանքի համար մենք կկենտրոնանանք Q-learning ալգորիթմի կիրառման վրա FrozenLake խաղային միջավայրի համատեքստում: FrozenLake-ը դասական ամրապնդման ուսուցման խնդիր է, որտեղ գործակալը պետք է գտնի օպտիմալ ճանապարհը սառեցված լճի միջով՝ խուսափելով վտանգավոր անցքերից և հասնել թիրախային կետին: Այս միջավայրում Q-learning ալգորիթմի օգտագործումը մեզ թույլ է տալիս ուսումնասիրել գործակալի ուսուցման գործընթացը, անորոշության պայմաններում որոշումներ կայացնելու նրա կարողությունը և յուրաքանչյուր փորձառության հետ բարելավել իր կատարումը: Նախագիծը ներառում է եռափուլ հետազոտությունների իրականացում (տեղեկությունների հավաքում, մշակում, վերլուծություն):

Այս աշխատանքը կներառի Q-learning ալգորիթմի նկարագրությունը, դրա կիրառումը ամրապնդող ուսուցման խնդիրներում և FrozenLake-ում այս մեթոդի կիրառմամբ փորձերի արդյունքների վերլուծություն: Մենք կդիտարկենք ալգորիթմի արդյունավետությունը տարբեր պայմաններում, հարմարվելու և շրջակա միջավայրի բարդությունները հաղթահարելու կարողությունը: Ի վերջո, մեր նպատակն է ոչ միայն հասկանալ Q-learning-ի հիմունքները, այլ նաև բացահայտել դրա կիրառելիությունը իրական աշխարհի խնդիրների համատեքստում, որտեղ օպտիմալ որոշումներ կայացնելը կարևոր նշանակություն ունի:

ԱԼԳՈՐԻԹՄԻ ՆԿԱՐԱԳՐՈՒԹՅՈՒՆԸ

Q-learning-ը ամրապնդմամբ ուսուցման ամենատարածված մեթոդներից մեկն է, որն օգտագործվում է գործակալին սովորեցնել անորոշության պայմաններում օպտիմալ որոշումներ կայացնել: Q-learning-ի հիմնական գաղափարը գործակալին վարժեցնելն է այնպիսի գործողություններ ընտրելու համար, որ առավելագույնի հասցնի այդ գործողությունների կատարման արդյունքում ստացված ընդհանուր պարգևը:

Q-learning ալգորիթմում գործակալը փոխազդում է շրջակա միջավայրի հետ՝ որոշումներ կայացնելով և հետադարձ կապ ստանալով իր գործողությունների համար պարգևների տեսքով: Գործակալի նպատակն է գտնել գործողության ռազմավարություն, որը երկարաժամկետ հեռանկարում կբերի ընդհանուր պարգևի առավելագույնին: Այս նպատակին հասնելու համար գործակալն օգտագործում է Q ֆունկցիան, որը գնահատում է յուրաքանչյուր հնարավոր գործողության օգտակարությունը շրջակա միջավայրի յուրաքանչյուր վիճակում:

Q-learning ալգորիթմը թարմացնում է Q ֆունկցիայի արժեքները՝ հիմնվելով հետադարձ կապի վրա և թույլ է տալիս գործակալին ժամանակի ընթացքում բարելավել իր գործողությունների ռազմավարությունը: Q արժեքները թարմացվում են հետևյալ բանաձևով.

$$Q(s,a) = Q(s,a) + \alpha * (r + \gamma * \max(Q(s',a')) - Q(s,a))$$

Որտեղ:

- Q(s, a) - Q ֆունկցիայի արժեքը s վիճակի և a գործողության համար,
- α - ուսուցման արագություն, որը որոշում է նոր տեղեկատվության կարևորությունը գոյություն ունեցող արժեքների համեմատ,
- r - պարգև, որը ստացվել է s վիճակում a գործողությունը կատարելու համար,
- γ - զեղչի գործոն, որը որոշում է ապագա պարգևների կարևորությունը ներկա պահին պարգևների համեմատ,

- s' - նոր վիճակ, որը ստացվել է a գործողությունից հետո,
- a' - հաջորդ գործողությունը ընտրված է նոր s' վիճակում:

Q-ուսուցման ալգորիթմը շարունակում է թարմացնել Q ֆունկցիայի արժեքները, մինչև որ գործակալը հասնի գործողության օպտիմալ ռազմավարության կամ ավարտի որոշակի թվով ուսուցման կրկնություններ:

ՄԻՋԱՎԱՅՐԻ ՆԿԱՐԱԳՐՈՒԹՅՈՒՆԸ և ՍԱՀՄԱՆԱՓԱԿՈՒՄՆԵՐԸ

Միջավայրի նկարագրությունը:

FrozenLake-ը դասական ամրապնդման ուսուցման խնդիր է, որը 4x4 չափի ցանց է, որտեղ յուրաքանչյուր բջիջ կարող է լինել չորս վիճակներից մեկում՝ «F» (սառեցված լիճ), «H» (վտանգավոր անցք), «S» (սկիզբ) և «G» (նպատակակետ): Գործակալի նպատակն է ապահով ճանապարհորդել «S» մեկնարկային կետից մինչև «G» թիրախային կետը՝ միաժամանակ խուսափելով «H» փոսերից: Գործակալը կարող է կատարել չորս հնարավոր գործողություն՝ շարժվել դեպի վեր, վար, ձախ և աջ:

Սահմանափակումներ:

1. Միջավայրի վիճակներ. FrozenLake միջավայրը բաղկացած է սահմանափակ թվով վիճակներից, որոնցից յուրաքանչյուրը ներկայացնում է որոշակի դիրք 4x4 ցանցում: Սա սահմանափակում է վիճակների տարածքը, որը գործակալը կարող է ուսումնասիրել:

2. Գործակալի գործողություններ. Գործակալը կարող է կատարել միայն չորս հնարավոր գործողություն՝ շարժվել վեր, վար, ձախ և աջ: Այս սահմանափակումը սահմանում է գործողության տարածքը, որը հասանելի է գործակալի կողմից հետազոտության համար:

3. Պարգևներ և տույժեր. FrozenLake միջավայրում գործակալը ստանում է պարգև միայն «G» նպատակակետին հասնելու դեպքում և տուգանք՝ «H» փոսերում հայտնվելիս: Այլ վիճակներում պարգևատրման բացակայությունը կարող է դժվարացնել գործակալի համար սովորելը և պահանջում է հավասարակշռություն շրջակա միջավայրի ուսումնասիրության և պարգևներ ստանալու միջև:

4. Միջավայրի պատահականություն. FrozenLake-ում միջավայրը կարող է պատահական լինել, ինչը նշանակում է, որ գործակալի գործողության արդյունքը կարող

է կախված լինել արտաքին գործոններից կամ պատահական իրադարձություններից: Սա ազդում է ուսուցման կայունության վրա և պահանջում է լրացուցիչ ռազմավարություններ՝ պատահականությունը կառավարելու համար:

Հաշվի առնելով այս սահմանափակումները՝ գործակալը պետք է արդյունավետ կերպով ուսումնասիրի շրջակա միջավայրը՝ սովորելով գործողության օպտիմալ ռազմավարությունը՝ հաշվի առնելով հնարավոր պարգևները և ռիսկերը:

PYTHON-ՈՎ ԾՐԱԳՐԱՅԻՆ ԿՈԴԻ ԻՐԱԿԱՆԱՑՈՒՄԸ

```
import random

import numpy as np

import gym

import time

import json


def train_agent(epsilon, num_epoch):

    start_epsilon = epsilon

    start_time = time.time()


    env = gym.make("FrozenLake-v1", is_slippery=False)


    q_table = np.zeros((env.observation_space.n, env.action_space.n))


    alpha = 0.1

    gamma = 0.99

    epsilon_decay = 0.999


    for _ in range(num_epoch):

        state = env.reset()[0]


        while True:

            if random.uniform(0, 1) < epsilon:

                action = env.action_space.sample()
```

```
        else:
            action = np.argmax(q_table[state])

        next_state, reward, terminated, truncated, _ = env.step(action)

        if reward == 0.0:
            reward = -10.0 if terminated else -1.0

        q_table[state][action] = (
            (1 - alpha) * q_table[state][action] +
            alpha * (reward + gamma * np.max(q_table[next_state]))
        )

        if terminated or truncated:
            break

        state = next_state

    if epsilon > 0.1:
        epsilon *= epsilon_decay

print("Training finished.\n")

end_time = time.time()
execution_time = end_time - start_time
```

```
print("Execution time:", execution_time, "seconds \n")


env = gym.make("FrozenLake-v1", is_slippery=False, render_mode='human')
state = env.reset()[0]
render_start_time = time.time()

while True:
    action = np.argmax(q_table[state])
    next_state, reward, done, _, _ = env.step(action)
    state = next_state

    if time.time() - render_start_time > 3:
        print("Rendering takes too long. Unable to solve with these parameters.")
        break

    if done:
        break

render_time = time.time() - render_start_time

env.close()

return {
    "epsilon": start_epsilon,
```

```
        "num_epochs": num_epoch,
        "execution_time": execution_time,
        "render_time": render_time
    }

if __name__ == "__main__":
    epsilon_values = [0.1, 0.3, 0.5, 0.7, 0.9, 1]
    num_epochs_values = [30, 50, 70, 100, 300, 500, 700, 1000, 5000, 10000]

    results = []

    for epsilon in epsilon_values:
        for num_epochs in num_epochs_values:
            print(f"Training with epsilon={epsilon} and num_epochs={num_epochs}")
            results.append(train_agent(epsilon, num_epochs))

    with open('training_results.json', 'w') as f:
        json.dump(results, f, indent=4)

    print("Results saved to training_results.json")
```

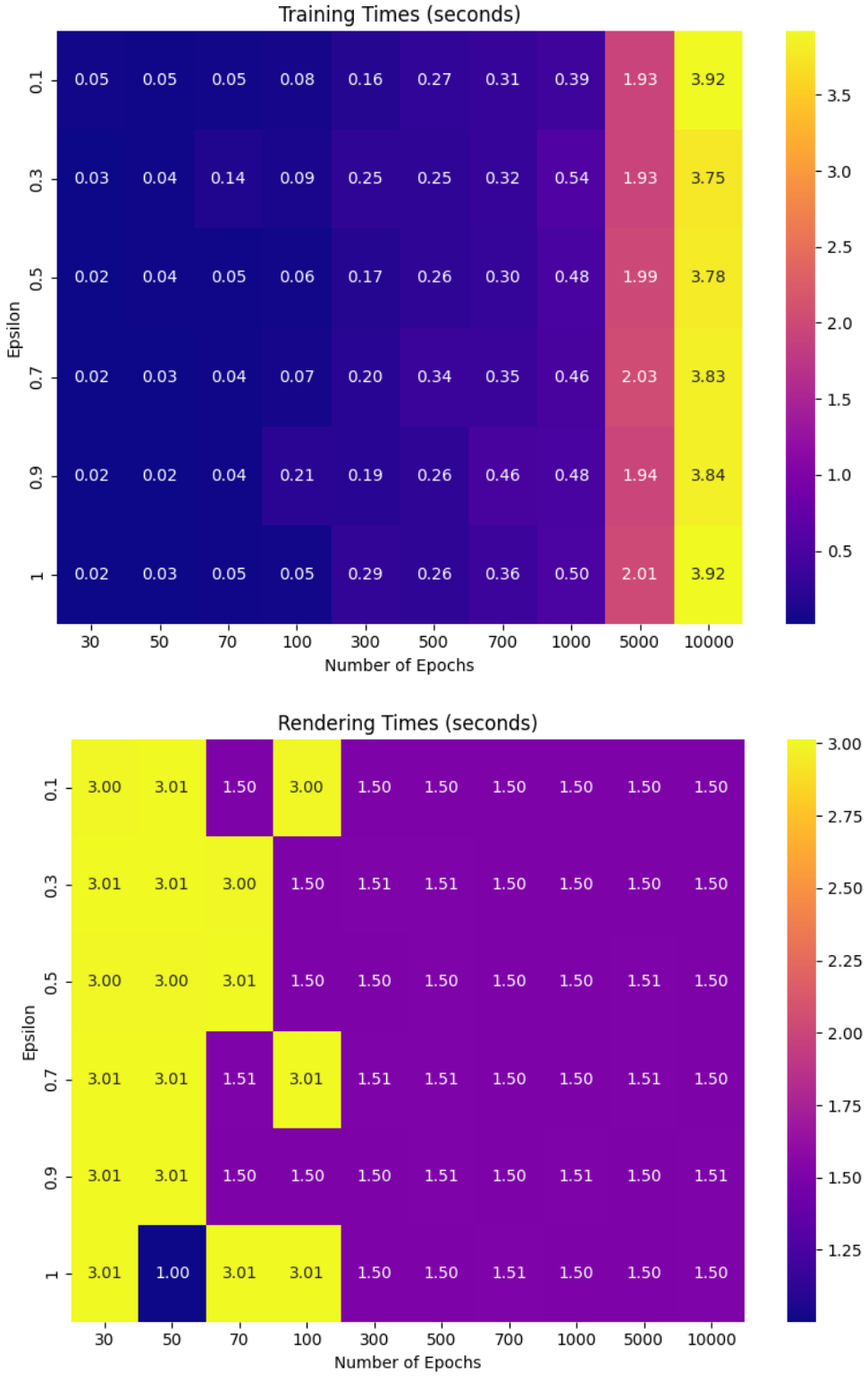
Վիզուալ ներկայացման համար

```
import json
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the results from the JSON file
with open('training_results.json', 'r') as f:
    results = json.load(f)

# Extract unique epsilon and num_epochs values
epsilon_values = sorted(set(result["epsilon"] for result in results))
num_epochs_values = sorted(set(result["num_epochs"] for result in results))

# Create matrices to store training times and rendering times
training_times_matrix = np.zeros((len(epsilon_values), len(num_epochs_values)))
rendering_times_matrix = np.zeros((len(epsilon_values), len(num_epochs_values)))

# Fill the matrices with training times and rendering times
for result in results:
    epsilon_index = epsilon_values.index(result["epsilon"])
    num_epochs_index = num_epochs_values.index(result["num_epochs"])
    training_times_matrix[epsilon_index, num_epochs_index] = result["execution_time"]
    rendering_times_matrix[epsilon_index, num_epochs_index] = result["render_time"]
```

```
# Create a figure with two subplots for training times and rendering times
fig, axes = plt.subplots(1, 2, figsize=(18, 6))

# Plot training times
sns.heatmap(training_times_matrix, ax=axes[0], annot=True, fmt=".2f", cmap="plasma",
            xticklabels=num_epochs_values, yticklabels=epsilon_values)
axes[0].set_xlabel('Number of Epochs')
axes[0].set_ylabel('Epsilon')
axes[0].set_title('Training Times (seconds)')

# Plot rendering times
sns.heatmap(rendering_times_matrix, ax=axes[1], annot=True, fmt=".2f", cmap="plasma",
            xticklabels=num_epochs_values, yticklabels=epsilon_values)
axes[1].set_xlabel('Number of Epochs')
axes[1].set_ylabel('Epsilon')
axes[1].set_title('Rendering Times (seconds)')

plt.tight_layout()
plt.show()
```

ՍՏԱՑՎԱԾ ԱՐԴՅՈՒՆՔԵՐԻ ՎԵՐԼՈՒԾՈՒԹՅՈՒՆԸ և ՀԱՇՎԵՏՎՈՒԹՅՈՒՆ

Ստորև ներկայացված են ուսուցման և իրագործման ժամանակները կախված epsilon և number of epochs պարամետրերից (նկ. 1)։



Նկ. 1 ՈՒսուցման և իրագործման ժամանակները կախված պարամետրերից

Նկ.1-ից երևում է, որ միայն մի դեպքում է գործակալը ընկնում անցքի մեջ (epsilon = 1 և number of epochs = 50):

30 և 50 number of epochs-ի դեպքում գործակալը չի կարողանում լուծել խնդիրը, ինչի պատճառով 3 վարկյան հետո ստիպողաբար կանգնեցվում է նրա աշխատանքը: Նույնը տեղի է ունենում 70 և 100 number of epochs -ի դեպքերի 50%-ում:

Բոլոր այն դեպքերում երբ գործակալին հաջողվում է լուծել խնդիրը դա տևում է 1.50 կամ 1.51 վարկյան: Հստակ կապ չի գտնվել կատարման տևողության և պարամետրերի միջև, բացի այն որ epsilon = 0.1-ի դեպքում տևողությունը միշտ 1.50 վարկյան է:

Չնայած նրան, որ number of epochs-ի ավելացման հետ աճում է ուսուցման ժամանկը (մինչև 24.5 անգամ), գործակալի աշխատանքում տեսանելի փոփոխություններ չեն նկատվում:

ԵԶՐԱԿԱՑՈՒԹՅՈՒՆ

Այս կուրսային աշխատանքի ընթացքում իրականացվել է ուսումնասիրություն FrozenLake միջավայրում Q-learning ալգորիթմի օգտագործման վերաբերյալ՝ դրա արդյունավետությունն ու հնարավորություններն ուսումնասիրելու ամրապնդման ուսուցման խնդիրների համատեքստում: Ընդհանուր վերլուծությունը և ստացված արդյունքները թույլ են տալիս մի քանի հիմնական եզրակացություններ անել.

Նախ, Q-learning ալգորիթմի օգտագործումը FrozenLake միջավայրում ցույց է տալիս սահմանափակ և պատահական միջավայրում գործակալի գործողությունների օպտիմալ ռազմավարությունը սովորելու նրա կարողությունը: Սա հաստատվում է փորձարարական արդյունքներով, որոնք ցույց են տալիս գործակալի աշխատանքի աստիճանական բարելավում կրկնությունների քանակի աճով:

Երկրորդ, գործակալի ուսուցման գործընթացի վերլուծությունը թույլ է տալիս մեզ բացահայտել տարբեր պարամետրերի ազդեցությունը ալգորիթմի կատարման և արագության վրա: Սա օգնում է ավելի լավ հասկանալ, թե որ պարամետրերի ընտրության ռազմավարությունները կարող են առավել արդյունավետ լինել կոնկրետ առաջադրանքների և միջավայրերի համար:

Երրորդ, number of epochs = 300 բավարար է խնդիրը լուծելու համար, ամենաբարձր արդյունավետությունը լինում է epsilon = 0.1-ի դեպքում:

Վերջապես, հետազոտության արդյունքները ընդգծում են հավասարակշռության կարևորությունը շրջակա միջավայրի ուսումնասիրությունը և որոշումներ կայացնելու համար հայտնի տեղեկատվության օգտագործման հարցում: Սա թույլ է տալիս գործակալին հասնել գործողության օպտիմալ ռազմավարության:

ՕԳՏԱԳՈՐԾՎԱԾ ԳՐԱԿԱՆՈՒԹՅԱՆ ՑԱՆԿ

1. https://gymnasium.farama.org
2. https://chat.openai.com
3. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.
4. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.
5. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489.
6. Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.
7. Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. Journal of artificial intelligence research, 4, 237-285.
8. Hasselt, H. V. (2010). Double Q-learning. Advances in neural information processing systems, 23, 2613-2621.
9. Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). Mathematics for machine learning. Cambridge University Press.