Movie Recommendation System

Group 16
111550075 顏名柔
111550108 吳佳諭
111550119 蔡承倢

Contents

Introduction

- To provide movies based on user preferences.
- Why this is important?
    - **For users**, they can find the movies they may enjoy, leading to a better watching experience. Also, this can save the users' time in searching moveis they like.
    - **For platform**, if they can provide the movies that match the user's preerence, users are more likely to stay in this platform, bringing the platform a higher revenue.

Related work

- Open Source:

  1. Paper: Deep Reinforcement Learning based Recommend System using stratified sampling
  2. Work: Netflix

  -> Try to use RL (i.e. DQN) to build our own recommendation system

- Difference: Beside DQN, we also use two traditional method, such as

  1. Collaborative Filtering

  2. Content-Based Filtering

Dataset – MovieLens 100K movie ratings (ml-100k)

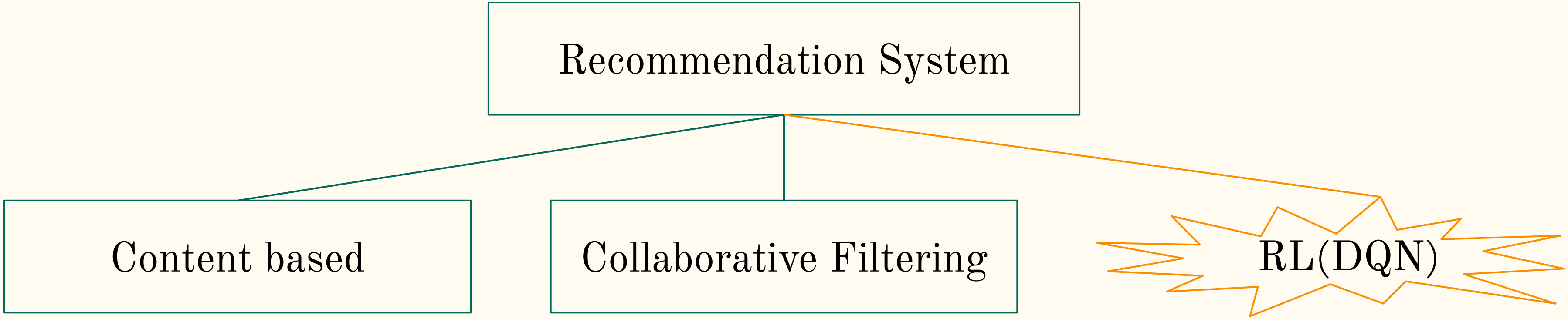
- 100,000 ratings (1-5) from 943 users on 1682 movies
- https://grouplens.org/datasets/movielens/100k/
- ml-100k (only list the files we used)
  |– u.data
  |– u.item
  |– u.user

Dataset – MovieLens 100K movie ratings (ml-100k)

Data	Description	Columns
u.data	The full u data set.	user id | item id | rating | timestamp
u.item	Information about the items (movies).	movie id | movie title | release date | video release date | IMDb URL | unknown | Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War | Western |
u.user	Information about the users.	user id | age | gender | occupation | zip code

Baseline – Why to choose them

- Use three algorithms and compare their results
    - Content based, Collaborative filtering, DQN
- Where does these baselines come from & why to choose them
    - Content based and collaborative filtering are traditional ways.
    - DQN use Neural Network to give better performance.



Baseline – Implementation: content based

- Content-based: Calculate the similarity between movies and recommend the movies that is similar to the user's movie.

```
similarity_matrix = []
for i in range(1683):
    similarity_matrix.append([])
    for j in range(1683):
        if(i==1682 or j==1682):
            similarity_matrix[i].append(0)
            continue
        a, b, c = 0, 0, 0
        for k in range(5, 24):
            a += int(movies[i][k]) * int(movies[j][k])
            b += int(movies[i][k]) * int(movies[i][k])
            c += int(movies[j][k]) * int(movies[j][k])
        if(b*c==0): similarity = a      # avoid devide by zero
        else: similarity = a / (math.sqrt(b) * math.sqrt(c))
        similarity_matrix[i].append(similarity)
```

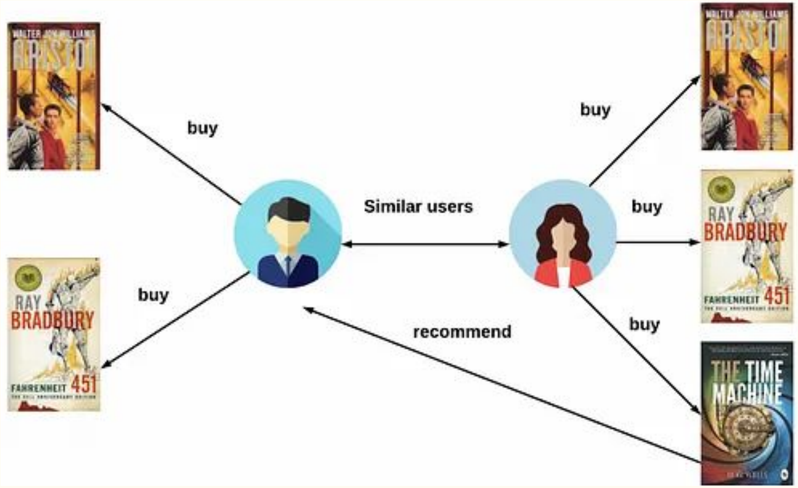
ref: Introduction to recommender systems

Baseline – Implementation: collaborative filtering

- Collaborative filtering: Relies on user-item interactions. It predict rating of items based on the user with similar preferences and recommends items that those like-minded users have enjoyed.

```python
# trains the SVD model with the best parameters
best_epochs = gs.best_params["rmse"]["n_epochs"]
best_factors = gs.best_params["rmse"]["n_factors"]
svd_sol_best = SVD(verbose=True, n_epochs=best_epochs, n_factors = best_factors)
cross_validate(svd_sol_best, data, measures=['RMSE', 'MAE'], cv=3, verbose=True)

# makes predictions based on the given movie for the given user
score = []
for iid in range(1682):
    pred_best = svd_sol_best.predict(uid, iid, r_ui=9, verbose=True)
    score.append(pred_best.est)
```



ref: [Introduction to recommender systems]

Baseline – Implementation: DQN

- DQN: neural network
  - 3 layers
  - batch_size = 32

```
class Net(nn.Module):
    '''
    The Neural Network, calculate Q value for each state
    '''
    def __init__(self, num_action, hidden_layer_size = 50):
        super(Net, self).__init__()
        self.input_state = 25
        self.num_action = num_action
        self.fc1 = nn.Linear(self.input_state, 32)           # input layer
        self.fc2 = nn.Linear(32, hidden_layer_size)          # hidden layer
        self.fc3 = nn.Linear(hidden_layer_size, num_action) # output layer

    def forward(self, state):
        '''...
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))
        q_value = self.fc3(x)
        return q_value
```

Baseline – Limitation

- What are the possible limitation it will encounter on your tasks
  - Content based: filter bubble, large memory requirement
  - Collaborative filtering: cold start, need enough data, popular bias
  - DQN: design an appropriate reward function, over fitting

Main Approach 1: Content based – Introduction

- Using the genre of each movie, recommend five movies that are the most simimlar to the movies that the user watched.
- We use Cosine Similarity.

That is, similarity between A and B is $\dfrac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \cdot \sqrt{\sum\limits_{i=1}^{n} B_i^2}}$,

where Ai and Bi are the genre of the movie.

Main Approach 1: Content based – Introduction

Movie/ Genre	Action	Comedy	Drama
A	1	0	1
B	0	1	1
C	0	1	0

Movie-Genre matrix.

Movie/ Movie	A	B	C
A	-	0.5	0
B	0.5	-	0.707
C	0	0.707	-

Similarity matrix.

With the similarity matrix, we can know “how” similar is the movie i to movie j. If the user watched movie B, we can guess that the user may also like movie C because it is the most similar to movie B.

Main Approach 1: Content based – Implementation

- Input: movie name. Output: five recommand movies.

1. Load data and construct the similarity matrix (**`similarity_matrix`**).

```
similarity_matrix = []
for i in range(1683):
    similarity_matrix.append([])
    for j in range(1683):
        if(i==1682 or j==1682):
            similarity_matrix[i].append(0)
            continue
        a, b, c = 0, 0, 0
        for k in range(5, 24):
            a += int(movies[i][k]) * int(movies[j][k])
            b += int(movies[i][k]) * int(movies[i][k])
            c += int(movies[j][k]) * int(movies[j][k])
        if(b*c==0): similarity = a      # avoid devide by zero
        else: similarity = a / (math.sqrt(b) * math.sqrt(c))
        similarity_matrix[i].append(similarity)
```

Main Approach 1: Content based – Implementation

2. User enter a movie. Use a list (**`user_movie`**) to record the movie genre that the user like.

```
for i in range(19):
    user_movie[i] = (user_movie[i] * (n_movie-1) + int(movies[watched_movie_id][i+5])) / n_movie
```

3. Calculate the similarity between the user's prefer movie and other movies.

4. Output five movies that has the largest similarity value. (random)

```
Please enter a movie name (enter 0 to exit): Toy Story
You may like the following 5 movies:
                    Aladdin and the King of Thieves || similarity = 1.0000
                                            Aladdin || similarity = 0.8660
                                     Goofy Movie, A || similarity = 0.8660
                                              Balto || similarity = 0.8165
                                    Grand Day Out, A || similarity = 0.8165
```

Main Approach 2: SVD – Introduction

- Based on the input movie, estimate scores of other movies, and then recommend five that get the highest scores.

- We use singular value decomposition(SVD) to predict ratings.

That is, $V = UM^T$

❖ V: Original rating matrix $V \in R^{n \times m}$
(Vij: user i's rating of movie j)
❖ U: User feature matrix $U \in R^{f \times n}$
❖ M: Movie feature matrix $M \in R^{f \times m}$

- And then we define "Cost Function": p(Ui, Mj) is the prediction of rating.

$$E = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} I_{ij}(V_{ij} - p(U_i, M_j))^2 + \frac{k_u}{2} \sum_{i=1}^{n} \|U_i\|^2 + \frac{k_m}{2} \sum_{j=1}^{m} \|M_j\|^2$$

Main Approach 2: SVD – Introduction

Movie/ User	Avengers 3	Transformer 3
John	5	4
Amy	4	1
Jack	0	2

Rating Matrix (n * m)

Movie feature Matrix (n * f)

Movie/ Genre	Avengers 3	Transformer 3
Drama	3	-2
Animation	3	5

User/ Genre	John	Amy	Jack
Drama	3	5	3
Animation	4	0	3

User feature Matrix (f * m)

Main Approach 2: SVD – Implementation

- Input: user ID and movie name. Output: five recommend movie.

1. Load data and users enter their id and movies.
2. Choose movies that belongs to the same genres as given movie to be training data .(train_movie)
3. Create a dataset that can be use in svd function.

```
# Create a Reader object with a rating scale of 1 to 5
reader = Reader(rating_scale=(1, 5))

# Create a Dataset object from the train_rating data using the Reader object
data = Dataset.load_from_df(pd.DataFrame({'user_id': [int(x[0]) for x in train_rating],
                                          'anime_id': [int(x[1]) for x in train_rating],
                                          'rating': [int(x[2]) for x in train_rating]}),
                            reader)
```

Main Approach 2: SVD – Implementation

- Input: user ID and movie name. Output: five recommend movie.

4. Use GridSearchCV to choose the best parameter for SVD.

```
param_grid = {"n_epochs": [5, 10], "lr_all": [0.002, 0.005], "n_factors": [50, 100,150]}
gs = GridSearchCV(SVD, param_grid, measures=["rmse", "mae"], cv=3, joblib_verbose=1, n_jobs=2)
gs.fit(data)
algo = gs.best_estimator["rmse"]
trainset = data.build_full_trainset()
algo.fit(trainset)
predictions = algo.test(trainset.build_testset())
accuracy.rmse(predictions)
```

5. Use surprise to train SVD.

```
# trains the SVD model with the best parameters
best_epochs = gs.best_params["rmse"]["n_epochs"]
best_factors = gs.best_params["rmse"]["n_factors"]
svd_sol_best = SVD(verbose=True, n_epochs=best_epochs, n_factors = best_factors)
cross_validate(svd_sol_best, data, measures=['RMSE', 'MAE'], cv=3, verbose=True)
```

Main Approach 2: SVD – Implementation

6. Calculate predicted ratings of each movie based on user id.

```
# makes predictions based on the given movie for the given user
score = []
for iid in range(1682):
    pred_best = svd_sol_best.predict(uid, iid, r_ui=9, verbose=True)
    score.append(pred_best.est)
```

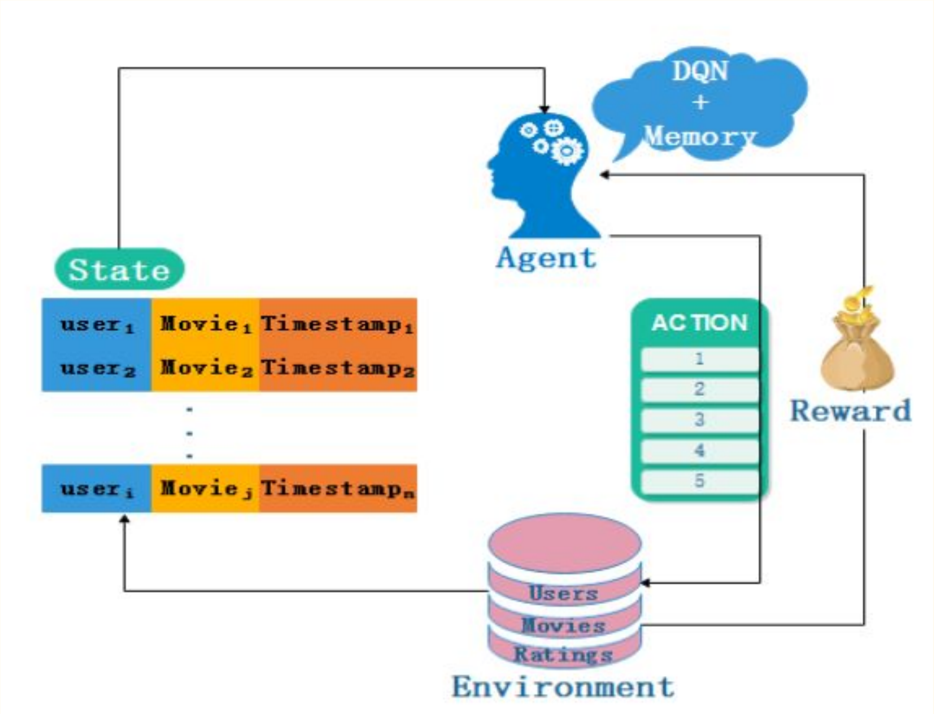
7. Output five movies with the highest predicted ratings.

```
# choose the top 5 similar movie
top_5_indices = sorted(range(len(score)), key=lambda i: (-score[i], i), reverse=True)[:5]

#Output
for i in top_5_indices:
    name = dataset.find_movie_name(i)
    est_score = score[i]
    print("%60s || Score = %.4f" %(name, est_score))
```

Main Approach 3: DQN

Use the neural network to calculate Q value for each state. The agent will learn in the environment by doing some actions and receive the rewards.



https://iopscience.iop.org/article/10.1088/1757-899X/466/1/012110/pdf

Main Approach 3: DQN – Introduction

DQN:

Combines the Q-Learning algorithm with deep neural networks.

- **Input**: user ID and movie name.
- **Output**: one recommend movie name.
- State = (User ID, Movie ID, Gender, Age, Occupation, Zip code)
- Action: The rating of the movie. [1,2,3,4,5]
  → We will recommend the movie with the highest action(score).
- Reward: the difference value between real score and predicted score by the DQN agent.

Main Approach 3: DQN – Introduction

- Detail of the Q learning algorithm.

(1)Q value estimation function

$$\underbrace{Q(S_t, A_t)}_{\text{New Q-value estimation}} \leftarrow \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}} + \underbrace{\alpha}_{\text{Learning Rate}}[\underbrace{\underbrace{\underbrace{R_{t+1}}_{\text{Immediate Reward}} + \underbrace{\gamma max_a Q(S_{t+1}, a)}_{\text{Discounted Estimate optimal Q-value of next state}}}_{\text{TD Target}} - \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}}}_{\text{TD Error}}]$$

(2)The Bellman's equation

$$Q(s, a; \theta) = r + \gamma \max_{a'} Q(s', a'; \theta')$$

Equation 1: Bellman's Equation for the DQN algorithm.

(3)Loss function

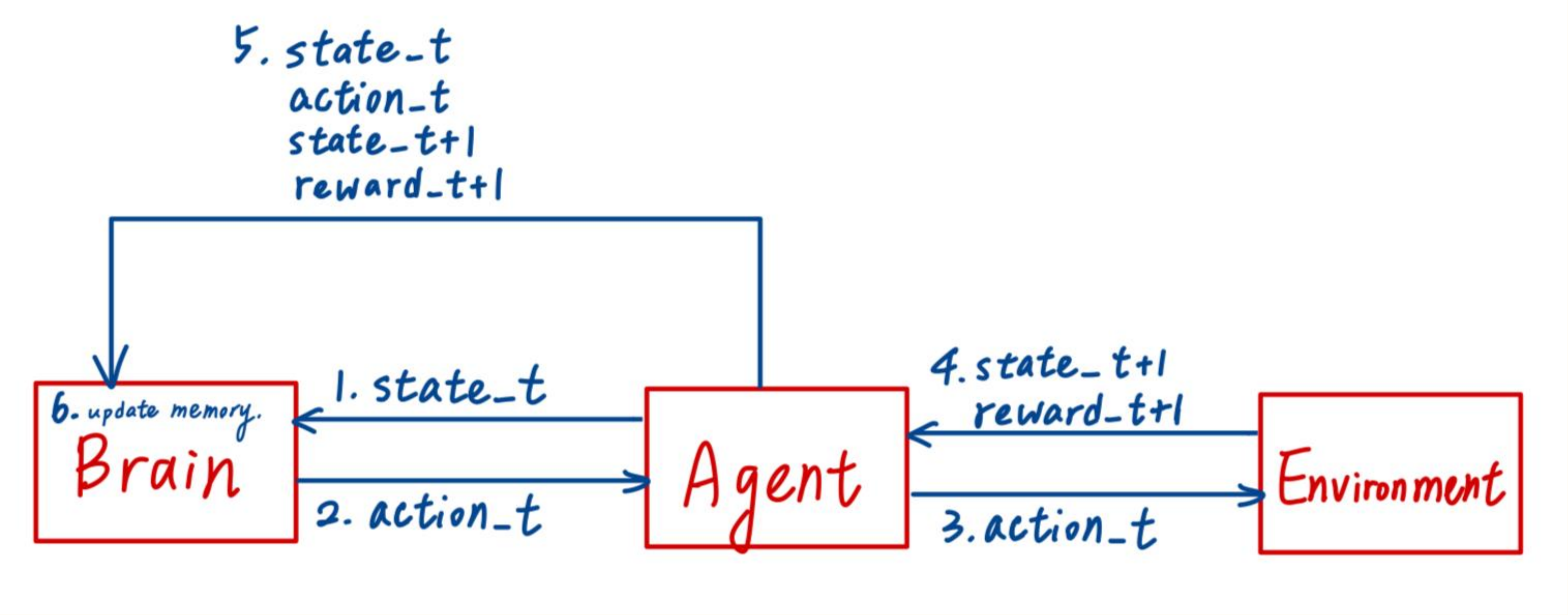
$$L(\theta) = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta))^2]$$

Equation 2: Loss function for the DQN algorithm.

ref: Introduction to Reinforcement Learning. Part 3: Q-Learning with Neural Networks, Algorithm DQN

Main Approach 3: DQN – Implementation

- The flow chart of DQN.python:



Main Approach 3: DQN – Implementation

1. Load data and buile up the environment.

```
databaserating = dataset.load_data()
databaseuser = dataset.load_user()
databasemovie = dataset.load_movie()

def __init__(self):
    self.action_space = spaces.Discrete(5,start=1)
    self.observation_space = spaces.Box(low=np.array([1,1,0,10,1,999]), high=np.array([943,1682,1,60,21,99999]), dtype=np.int64)
    num_states = self.observation_space.shape[0]
    num_actions = self.action_space.n
    self.agent = Agent(num_states, num_actions)
```

2. execute the “run” function inside the environment.

2-1 take the action from the agent

```
action = self.agent.get_action(state, episode)
```

Main Approach 3: DQN – Implementation

2-2 We define the next state as the next movie the user would watch according to the database.

```
next_movie = dataset.find_next_movie(current_userid, current_movieid)

if (next_movie == 1700):
    state = torch.from_numpy(observation).type(torch.FloatTensor)  # NumPy變數轉換成Pytorch的張量
    state = torch.unsqueeze(state, 0)
    observation[0] = observation[0]+1
    user_info = dataset.find_userinfo(observation[0])
    observation[2] =  user_info[2]
    observation[3] = user_info[1]
    observation[4] = user_info[3]
    observation[5] = user_info[4]

else:
    observation[1] = next_movie

observation_next =  observation
```

Main Approach 3: DQN – Implementation

2-3 calculate the reward using the reward function

2-4 send (state, action, state_next, reward) to the agent

```
if done:
    torch.save(self.agent.brain.model.state_dict(), "cj_DQN.pt")
    ret.append(r)
    break
else:
    state_next = observation_next
    state_next = torch.from_numpy(state_next).type(torch.FloatTensor)
    state_next = torch.unsqueeze(state_next, 0)
    r += reward

self.agent.memorize(state, action, state_next, reward)

self.agent.update_q_function()

state = state_next
```

```
real_rating = dataset.find_rating(current_userid,current_mo

if(real_rating == "none"):
    k = dataset.find_average_rating(int(current_userid))
    d = abs(k - int(action))
    complete_episodes = 0

else:
    d = abs(int(real_rating) - int(action))

reward = torch.FloatTensor([0.0])
if d == 0:
    reward = torch.FloatTensor([3.0])
    complete_episodes = complete_episodes + 3
elif d == 1:
    reward = torch.FloatTensor([1.0])
    complete_episodes = complete_episodes + 1
elif d == 2:
    reward = torch.FloatTensor([0.0])
    complete_episodes = complete_episodes
elif d == 3:
    reward = torch.FloatTensor([-1.0])
    complete_episodes = 0
elif d == 4:
    reward = torch.FloatTensor([-3.0])
    complete_episodes = 0
```

Main Approach 3: DQN – Implementation

3. Agent: build brain, update Q function, and decide the action.

```
class Agent:
    def __init__(self, num_states, num_actions):
        self.brain = Brain(num_states, num_actions)  #agent的腦袋

    def update_q_function(self): #更新Q函數
        self.brain.replay()

    def get_action(self, state, episode): #決定動作
        action = self.brain.decide_action(state, episode)
        return action

    def memorize(self, state, action, state_next, reward): #將state,action,state_next,reward存入memory
        self.brain.memory.push(state, action, state_next, reward)
```

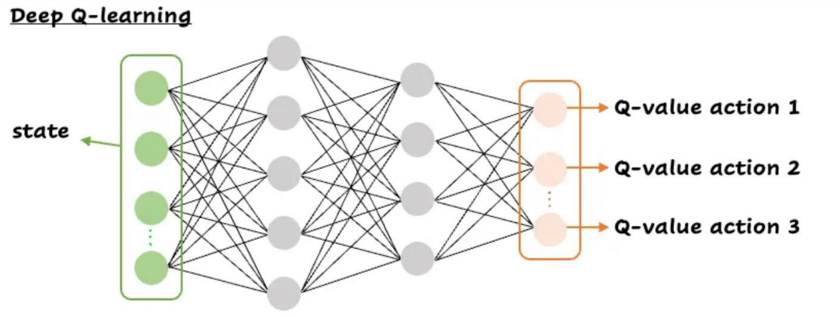
Main Approach 3: DQN – Implementation

4. Brain: The neural network itself, where the Q learning algorithm takes place.

- Input: states, the information with both the movie and the user.
- Output: the Q_value for each action. (Action: the rating of the movie.)

```python
def __init__(self, num_states, num_actions):
    self.num_actions = num_actions
    self.memory = ReplayMemory(CAPACITY) #make a replay buffer.

    #Build the neural network.
    self.model = nn.Sequential()
    self.model.add_module('fc1', nn.Linear(num_states, 32)) #input layer
    self.model.add_module('relu1', nn.ReLU())
    self.model.add_module('fc2', nn.Linear(32, 32)) #hidden layer
    self.model.add_module('relu2', nn.ReLU())
    self.model.add_module('fc3', nn.Linear(32, num_actions)) #output layer
```



Relation between Q-learning and deep Q-learning: the table is replaced by a neural network, where the input layer contains information about the state, and the outputs are Q-values for every action. Image by author.

ref: Techniques to Improve the Performance of a DQN Agent

Main Approach 3: DQN – Result

- x_axis : episode
- y_axis : the sum of the rewards.

Possible reason:

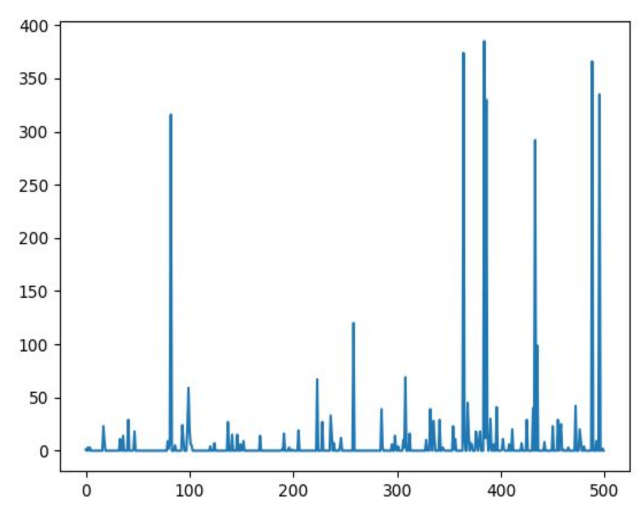
(1) Overfitting → DDQN

(2) Wrong definition of state, next_state, action or even their relation.

(3)Lack of movie related elements in the state.

```
100%|████████████████████████████████████| 500/500
Enter your user ID (1-943): 219
Please enter a movie name (enter 0 to exit): Four Rooms
you may like  Antonia's Line
Please enter a movie name (enter 0 to exit): Antonia's Line
you may like  Toy Story
Please enter a movie name (enter 0 to exit): Toy Story
you may like  Taxi Driver
```

rewards



episode

Evaluation Metric

- RMSE: We use RMSE to measure the difference between predicted ratings and actual ratings.
- For SVD, it already computes the predicted rating. For content-based, we define predicted rating = similarity * 5. With these predicted ratings, we can calculate the RMSE to evaluate the algorithm's performance.

  ** Multiple by 5 because the rating is between 1-5

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(r_{ui} - \hat{r}_{ui})^2}$$

Where r_{ui} is the actual rating of user u for item i, and $\hat{r}_{ui}$ is the predicted rating.

Result & Analysis – Type experiment

- In SVD, there are 2 type can be changed: learning rate and factors.

1. Learning Rate(lr_all):

   We test for three different learning rate:

```
param_grid = {"n_epochs": [5, 10], "lr_all": [0.2, 0.5], "n_factors": [50, 100,150]}
```

```
param_grid = {"n_epochs": [5, 10], "lr_all": [0.02, 0.05], "n_factors": [50, 100,150]}
```

```
param_grid = {"n_epochs": [5, 10], "lr_all": [0.002, 0.005], "n_factors": [50, 100,150]}
```

Result & Analysis – Type experiment

1. Learning Rate(lr_all):

➢ Learning Rate = [0.2, 0.5]

True Romance	Score = 3.8155
Striptease	Score = 3.8183
Somewhere in Time	Score = 3.8450
Johnny Mnemonic	Score = 3.8451
Marked for Death	Score = 3.8610

➢ Learning Rate = [0.02, 0.05]

Striptease	Score = 3.8527
Johnny Mnemonic	Score = 3.9027
Glimmer Man, The	Score = 3.9046
Sliver	Score = 3.9076
Preacher's Wife, The	Score = 3.9464

➢ Learning Rate = [0.002, 0.005]

Striptease	Score = 3.9989
Johnny Mnemonic	Score = 4.0201
Cliffhanger	Score = 4.0404
Time to Kill, A	Score = 4.0422
Cape Fear	Score = 4.0430

Analysis: When the range of learning rate become more precise, the rating would be higher.

Result & Analysis – Type experiment

- In SVD, there are 2 type can be changed: learning rate and factors.

2. Number of factors:

   We test for three different factors:

```
svd_sol_best = SVD(verbose=True, n_epochs=best_epochs, n_factors = 50)
```

```
svd_sol_best = SVD(verbose=True, n_epochs=best_epochs, n_factors = 100)
```

```
svd_sol_best = SVD(verbose=True, n_epochs=best_epochs, n_factors = 150)
```

Result & Analysis – Type experiment

2. Number of factors:

➢ Number of factors = 50

```
      True Romance || Score = 4.0347
  Glimmer Man, The || Score = 4.0572
 Somewhere in Time || Score = 4.0576
            Sliver || Score = 4.0588
 Shadow Conspiracy || Score = 4.0792
```

➢ Number of factors = 100

```
      True Romance || Score = 3.8330
        Striptease || Score = 3.8413
            Sliver || Score = 3.8759
 Somewhere in Time || Score = 3.8783
    Chain Reaction || Score = 3.9157
```

➢ Number of factors = 150

```
      True Romance || Score = 3.7949
            Sliver || Score = 3.8450
   Johnny Mnemonic || Score = 3.8478
 Somewhere in Time || Score = 3.8483
  Marked for Death || Score = 3.8867
```

Analysis: When number of factors is increasing, the ratings are decreasing.

->It may be due to the model becoming too complex, resulting in overfitting.

Result & Analysis – Discussion and Analysis

- We use the same input, test for 1, 5, 10 times separately, and then compare RMSE.

➢ Content-Based

Filtering/ times	Content-Based
1	8.261599924202647
5	9.487659487650069
10	11.08391490881054

Analysis: When number of testcase is increasing, RMSE are also increasing.

-> Less Accuracy

Result & Analysis – Discussion and Analysis

- We use the same input, test for 1, 5, 10 times separately, and then compare RMSE.

➢ SVD

Filtering/ times	SVD
1	3.9288043553128413
5	3.858081586068965
10	3.913781415118725

Analysis: When number of testcase is increasing, shows some fluctuation, but the difference is not significant.

Result & Analysis – Discussion and Analysis

- Compare Content-based filtering with Collaborative filtering:

Filtering/ times	Content-Based	Collaborative
1	8.261599924202647	3.9288043553128413
5	9.487659487650069	3.858081586068965
10	11.08391490881054	3.913781415118725

Analysis: The RMSE for content-based filtering ranges from 8 to 11, while for collaborative filtering, it is concentrated between 3.8 and 3.9.

-> Collaborative filtering seems more stable?

Result & Analysis – Discussion and Analysis

- Possible Reason:

  Content-Based calculate similarity according to movie genre.

  -> Users might type in different kind of movie everytime.

  -> RMSE is sensitive to genre of input movie

  Our guess: If we input movies with the same genre everytime, RMSE for content-based might lower.

Result & Analysis – Discussion and Analysis

- To prove our guess: We used the top recommendation result as the input for the next iteration, conducting repeated tests.

Filtering/ times	Content-Based	Collaborative
1	2.013194072167833	3.857669464038458
5	1.4141995167287948	3.7697541396635095
10	3.217187094744785	3.791153271628686

1. RMSE for content-based become smaller!->Content-Based is sensitive to input data.
2. It didn't have much impact on collaborative filtering.->Reflect user-item interaction.

Limitation

- What are the possible limitation it will encounter on your tasks
  - Content based: filter bubble, large memory requirement
  - Collaborative filtering: cold start, need enough data, popular bias
  - DQN: design an appropriate reward function, over fitting

Practical use

- Users can get recommendations on the terminal.
- Applied to website, app, or LLM in the future.
  - streaming service, such as Netflix, friDay.
  - build an app or recommendation system.
  - Use it as a dataset to train a ChatBot.

Thank You

Github link & Reference

Github: https://github.com/chia-yuu/AI-final-project

Reference:

Collaborative Filtering
Recommendation system
RL based recommend system
Introduction to recommender systems
SVD
小川雄太郎 (2019)。《實戰人工智慧之深度強化學習|使用PyTorch x Python》。許郁文譯。臺北：碁峰資訊。
DQN

Contribution

Member	Contribution	Proportion
111550075 顏名柔	SVD, Evaluation, PPT, recording	30%
111550108 吳佳諭	Prepare the dataset, Content based, DQN, PPT, recoreding	40%
111550119 蔡承倢	Prepare the dataset, DQN, PPT, recording	30%