В качестве эксперта была взята политика, обученная с помощью RLOO с entropy_reg_coef = 0.05.

Создан датасет data/expert_dataset.csv формата:

x,x_dot,theta,theta_dot,action

-0.005635776091367006,0.00684911897405982,0.04081037640571594,-0.024575045332312584,0.0

-0.005498793907463551,-0.18883360922336578,0.04031887650489807,0.28069958090782166,1.0

-0.009275466203689575,0.0056907134130597115,0.045932866632938385,0.0010007602395489812,1.0

из 99289 сэмплов.

Обучена в supervised-стиле на criterion = nn.CrossEntropyLoss() необученная политика.

При валидации показала результаты:

Episode 0: reward = 500.0

Episode 1: reward = 500.0

Episode 2: reward = 500.0

Episode 3: reward = 500.0

Episode 4: reward = 500.0

Episode 5: reward = 500.0

Episode 6: reward = 500.0

Episode 7: reward = 500.0

Episode 8: reward = 500.0

Episode 9: reward = 500.0

И на самой симуляции видно, что каретка катается нормально, а столбик не падает :)

Случаи, когда BC может плохо работать:

1) Мало данных

   Попробуем из сгенерированного датасета взять лишь 10% данных (data/expert_dataset_10.csv, models/model_bc_10.pth). Результаты существенно ухудшились на некоторых эпизодах:

   Episode 0: reward = 500.0

   Episode 1: reward = 500.0

   Episode 2: reward = 153.0

   Episode 3: reward = 211.0

   Episode 4: reward = 500.0

   Episode 5: reward = 500.0

Episode 6: reward = 500.0

Episode 7: reward = 500.0

Episode 8: reward = 500.0

Episode 9: reward = 177.0

Это демонстрирует зависимость BC от объёма и разнообразия демонстраций

2) Измененные начальные состояния

```
# измененный угол шеста, чтобы показать, что BC не обобщается на новые состояния
# obs[2] += np.random.uniform(-1000000, 1000000)
```

Однако, даже с таким очень сильным отклонением качество не упало:

Episode 0: reward = 500.0

Episode 1: reward = 500.0

Episode 2: reward = 500.0

Episode 3: reward = 500.0

Episode 4: reward = 500.0

Episode 5: reward = 500.0

Episode 6: reward = 500.0

Episode 7: reward = 500.0

Episode 8: reward = 500.0

Episode 9: reward = 500.0

Даже при экстремальных начальных углах агент быстро выравнивает шест

3) Шум в состояние во время симуляции

```
obs_t_noisy = obs_t + torch.randn_like(obs_t) * 0.5
logits = policy(obs_t_noisy)
```

Шум уже “подкашивает” все эпизоды:

Episode 0: reward = 25.0

Episode 1: reward = 108.0

Episode 2: reward = 18.0

Episode 3: reward = 122.0

Episode 4: reward = 41.0

Episode 5: reward = 108.0

Episode 6: reward = 19.0

Episode 7: reward = 19.0

Episode 8: reward = 22.0

Episode 9: reward = 19.0

Повышение шума быстро деградирует политику, BC чувствителен к сенсорному шуму, т.к. не обучался корректировать ошибки

Возьмем шум поменьше

```
obs_t_noisy = obs_t + torch.randn_like(obs_t) * 0.1
logits = policy(obs_t_noisy)
```

У меньшему шуму уже хорошо устойчиво

Episode 0: reward = 500.0

Episode 1: reward = 500.0

Episode 2: reward = 500.0

Episode 3: reward = 500.0

Episode 4: reward = 500.0

Episode 5: reward = 500.0

Episode 6: reward = 500.0

Episode 7: reward = 500.0

Episode 8: reward = 500.0

Episode 9: reward = 392.0