ADS5037_강화학습
기말과제 보고서

과목	**ADS5037 강화학습**
이름	**김한비**
소속	**데이터사이언스융합과**
학번	**2021712203**

DQFD 논문 리뷰:

강화학습에서 사용되는 DQN(Deep Q-Learning)의 경우, 전반적으로는 좋은 성능을 보이나, 좋은 성능을 내기위해서는 학습 과정에서 많은 데이터와 시간이 소요된다. 따라서, 본 논문에서는 사전에 학습된 데이터를 이용하여 사전 학습을 시킴으로써, 적은 양의 데이터로 좋은 성능을 낼수 있는 DQfD(Deep Q-Learning from Demonstraion)을 소개하고 있다.

DQfD는 어떤 주어진 환경에서 얻어진 데이터들을 통해 학습을 시작하기 전에, 데모 데이터(Demonstration Data: 동일한 실험 환경에서 전반적으로 좋은 결과를 얻어 질 수 있게 얻어진 state, action, reward 등의 데이터 값)를 통한 사전 학습을 진행한다. 사전 학습은 두 가지 장점을 가지고 있다. 첫째로, 주어진 상태(State)에 대해서 이미 좋은 행동(Action)을 알고있기 때문에, 추후 같은 상태에서 해당 행동을 통해 더 큰 보상(Reward)를 기대 할 수 있다. 또한, 사전 학습을 통해 차후 Q-Network의 가중치를 미리 최적의 상태로 변경해놓은 상태에서 시작 할 수 있다.

보통 DQN의 경우, 각 에피소드의 상태, 행동 및 보상 등을 Replay Buffer에 저장하고, Replay Buffer의 크기가 어느정도 초과할 경우, 처음 저장되었던 상태을 제거하며 추후 상태를 업데이트 하는 반면, DQfD에서는 먼저 저장된 데모 데이터가 이미 좋은 성능을 보이는 데이터임으로, 데이터를 우선적으로 제거하기 보단, 비율을 설정해두어 차후 훈련이 잘 된 후 제거될수있게 설정하는 것의 중요성 또한 강조하고 있다. 또한, Q-Network 업데이트에 사용될 Mini-Batch를 선정하는 과정에서 역시 Random Sampling보다는 데모 데이터와 차후 얻어진 데이터를 조합하여 사용하는 것을 권장하고 있다.

사전 학습 과정에서 데모 데이터를 모방하여 Value Function을 Bellman Equation에 만족하게 업데이트하기 위해서 총 네 가지 Loss Function을 Q-Network에 적용한다. 이는, the 1-step double Q-Learning Loss, N-step Double Q-Learning Loss, Supervised Large Margin Classification Loss, 그리고 L2 Regularization Loss이다. 앞서 소개한 네 가지 Loss Function을 Q-Network의 가중치(Weight)와 절편(Bias)을 최적화하는데 사용된다. 각 각 Loss Function이 사용되는 이유는 다음과 같다.

첫째로, the 1-step double Q-Learning Loss은 데모 데이터로부터 얻어진 각 상태/행동 등을 그 다음 상태와 보상과 비교하여 Q-Network을 업데이트하며 이는 DQN의 Q-Network를 최적화하는데 사용되는 Loss Function과 동일하다. 다음으로, Supervised Large Margin Classification Loss는 데모 데이터로부터 얻어진 행동은, 전반적으로 좋은 결과 및 보상을 얻는 행동이기 때문에, 같은 상태에서 얻어진 다른 행동에 비해 좋은 행동임을 이미 알고있다. 따라서, 가중치를 더 주어서 다음에 해당 상태에 이동하였을 때, 데모 데이터로부터 얻은 행동을 수행할수있게 해준다. N-step Double Q-Learning Loss의 경우, 데모 데이터에서 모방할 수 있는 상태 및 행동에는 데모 데이터가 작을수록 그 제약이 크다. 적은 데이터에서 최대한 Q-Network를 최적화하기 위해, n-step을 정하여 해당 n-step이 진행되었을 때, Q-value와 보상들을 통해 Q-Network를 최적화시켜준다. 마지막으로, L2 Regularization Loss은 적은 데모 데이터로부터 업데이트된 가중치가 과적합이 나는 것을 방지하기 위한 Loss Function이다. N-step Double Q-Learning Loss, Supervised Large Margin

Classification Loss, 그리고 L2 Regularization Loss의 경우, 전체 Loss Function을 구하기전에 상수값(Lambda)를 이용하여 해당 Loss Function이 전체 Loss Function에 미칠 영향을 조절할수있다. 이를 이용하여, 데모 데이터를 이용한 사전 학습이 끝나고, 완전히 새로운 훈련이 시작될시, Supervised Large Margin Classification Loss의 상수값을 0으로 변경하여, 해당 Loss Function의 사용을 끝낼수있다.

DQfD 논문에서 소개된 알고리즘과 기말 과제 코드 비교 및 설명:

Algorithm 1 Deep Q-learning from Demonstrations.

Inputs: $\mathcal{D}^{replay}$: initialized with demonstration data set, θ: weights for initial behavior network (random), θ': weights for target network (random), τ: frequency at which to update target net, k: number of pre-training gradient updates
for steps $t \in \{1, 2, \ldots k\}$ **do**
    Sample a mini-batch of n transitions from $\mathcal{D}^{replay}$ with prioritization
    Calculate loss $J(Q)$ using target network
    Perform a gradient descent step to update θ
    if $t \bmod \tau = 0$ **then** $\theta' \leftarrow \theta$ **end if**
end for
for steps $t \in \{1, 2, \ldots\}$ **do**
    Sample action from behavior policy $a \sim \pi^{\epsilon Q_\theta}$
    Play action a and observe (s', r).
    Store (s, a, r, s') into $\mathcal{D}^{replay}$, overwriting oldest self-generated transition if over capacity
    Sample a mini-batch of n transitions from $\mathcal{D}^{replay}$ with prioritization
    Calculate loss $J(Q)$ using target network
    Perform a gradient descent step to update θ
    if $t \bmod \tau = 0$ **then** $\theta' \leftarrow \theta$ **end if**
    $s \leftarrow s'$
end for

```
q = Qnet()
q_target = Qnet()
q_target.load_state_dict(q.state_dict())
optimizer = optim.Adam(q.parameters(), lr=learning_rate)

memory = ReplayBuffer()
demon = get_demo_traj()

for i in range(len(demon)):
    for j in range(len(demon[i])):
        done_mask = 0.0 if demon[i][j][4] else 1.0
        memory.put_pretrain((demon[i][j][0], demon[i][j][1],
                             demon[i][j][2], demon[i][j][3], done_mask ))
```

1번

1. Qnet()에 저장된 Q-Network를 각각 q와 q_target에 저장하고, q_target의 가중치를 q와 동일하게 맞춰준다.
2. Replay Buffer를 불러오고, demo_traj에 저장된 데모 데이터를 각각 Replay Buffer에 저장

```
# Do pretrain
for k in range(1000):
    DQfDAgent.pretrain(self, q, q_target, memory, optimizer)
    if k % 100 == 0:
        q_target.load_state_dict(q.state_dict())
```

2,3,6번

1. 총 1000번의 사전 학습을 진행하며 100번 마다 q_target의 가중치를 q의 가중치로 업데이트.

```
def pretrain(self, q, q_target, memory, optimizer):
    s, a, r, s_prime, done_mask = memory.sample(batch_size)
    q_out = q(s)
    q_a = q_out.gather(1, a)
    max_q_prime = q_target(s_prime).max(1)[0].unsqueeze(1)
    target = r + gamma * max_q_prime * done_mask
    loss = F.mse_loss(q_a, target)
    je_loss = sum(q_out.max(1)[0].unsqueeze(1) + abs(q_out.max(1)[1].unsqueeze(1) - a) - q_a)
    loss = loss + 0.1 * je_loss[0]
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

3,4,5번

1. Replay Buffer에서 batch_size만큼 Mini Batch 샘플링
2. J(Q) Loss Function을 구하고 해당 Loss Function을 Gradient Descent로 최적화.

```
while not done:
    a = q.sample_action(torch.from_numpy(s).float())
    s_prime, r, done, info = env.step(a)
    done_mask = 0.0 if done else 1.0
    memory.put((s, a, r , s_prime, done_mask))
    s = s_prime
```

8,9,10,11번

1. 250 에피소드가 진행되고 각 에피소드는 앞서 구한 Q-Network에 의해서 Action이 결정
2. 해당 Action을 통해 Next State, Reward, Done의 정보를 Replay Buffer에 저장

```
if memory.size() > 400:
    for i in range(10):
        s, a, r, s_prime, done_mask = memory.sample(batch_size)

        q_out = q(s)
        q_a = q_out.gather(1, a)
        max_q_prime = q_target(s_prime).max(1)[0].unsqueeze(1)
        target = r + gamma * max_q_prime * done_mask
        loss = F.mse_loss(q_a, target)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

if e % 20 == 0:
    q_target.load_state_dict(q.state_dict())
```

12,13,14,15,16,17번

1. Buffer Size가 일정 크기(400)를 초과할시 새로운 훈련 진행.
2. 해당 훈련 과정은 앞선 사전 학습과 동일하나, J(E) Loss는 사용하지 않는다.