The Cross-Entropy Method

한양대학교 최솔비

RL methods 의 분류

- Model-free or model-based
- Value-based or policy-based
- On-policy or off-policy

Model-free / model-based

- model-free : 현재의 observation값 또는 이를 연산한 값이 그대로 action 이 됨.
- model-based : 미래의 observation이나 reward 를 예측하여 action 을 결정함.
- 복잡하고 observations 이 많은 환경에서는 적절한 모델을 세우기 어렵기 때문에 주로 model-free를 사용하며, 결정론적이고 제한된 환경에서는 model-based method 를 사용할 수 있다.

Value-based / policy-based

- policy-based : 가능한 다음 action들에 대한 확률 분포(policy)를 가지고 다음 action을 결정한다.
- value-based : 가능한 다음 action들을 모두 연산해 본 후, best value를 가져오는 action으로 결정한다.

On-policy / off-policy

- off-policy : agent가 이전 episodes 혹은 이전 agent의 data를 학습하여 policy prediction 한다.
- on-policy : 현재 agent 의 episodes 로 policy prediction 한다.

Cross-entropy method

- cross-entropy method 는
  model-free, policy-based, on-policy method 이다.
- cross-entropy는 agent가 수행할 다음 action에 대한 확률 분포(policy)을 예측해야 한다.

Practical cross-entropy

$$Observation\ s \Rightarrow trainable\ func(NN) \Rightarrow\ Policy\ \pi(a|s)$$

- current observation(input)을 가지고 probability distribution(output)을 도출하는 nonlinear function을 구하기 위해 neural network를 사용한다. (classification 문제와 유사)
- 이렇게 구한 확률 분포를 가지고 random number 를 생성하여 다음 action을 결정한다.

Practical cross-entropy(cont.)

- Cross-entropy method는 agent 의 일생동안 episodes 를 기억하고, 이 중 rewards 의 합이 높은 episodes 만을 학습하는 방법이다.

> 1. N 번의 episodes 를 실행한다.
> 2. 모든 episodes 의 total rewards 를 각각 계산한 후, reward boundary를 결정한다. (percentage)
> 3. boundary 이하의 reward를 가진 episodes를 버린다.
> 4. 남아있는 episodes(elite episodes)로 agent를 학습시킨다. (observations이 input, issued actions 이 desired output)
> 5. 좋은 결과가 나올 때까지 1부터 반복.

CartPole 예제로 알아보기

- 1개의 히든 레이어
- 128개 히든 뉴런
- 활성화 함수는 ReLU 사용
- hyperparameter 는 랜덤하게 초기화
  (어차피 cross-entropy 는 수렴이 빨리 됨)

CartPole 예제로 알아보기

```
HIDDEN_SIZE = 128
BATCH_SIZE = 16
PERCENTILE = 70
```

- `BATCH_SIZE` : iteration 마다 실행하는 episodes 수
- `PERCENTILE` : reward boundary (30퍼만 남김)

CartPole 예제로 알아보기(Net class 정의)

```
class Net(nn.Module):
        def __init__ (self, obs_size, hidden_size, n_acti
                super(Net, self).__init__()
                self.net = nn.Sequential(
                        nn.Linear(obs_size, hidden_size),
                        nn.ReLU(),
                        nn.Linear(hidden_size, n_actions)
                        ) # softmax 생략
        def forward(self, x):
                return self.net(x)
```

- network의 output 이 action에 대한 확률 분포이기 때문에 원래는 마지막에 softmax 함수를 써주어야 함 (softmax 는 정규화 시키는 함수)
- $$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \ for\ j = 1, ..., K.$$

CartPole 예제로 알아보기(cont.)

- 뒤의 `nn.CrossEntropyLoss` 클래스에서 softmax 와 cross-entropy loss 를 함께 계산해준다. softmax 계산은 지수화, cross-entropy loss 는 로그화가 들어가기 때문에 함께 계산하면 과정이 numerically stable 해진다.
- `CrossEntropyLoss` 메서드에는 반드시 정규화 되지 않은 raw data 를 넣어주어야 한다.
- 그렇기 때문에 만일 확률분포(policy)를 얻고 싶다면 output 에 직접 softmax 함수를 취해야 한다.

CartPole 예제로 알아보기 (iterate_batches 함수)

```
Episode = namedtuple('Episode',
                     field_names = ['reward','steps'])
EpisodeStep = namedtuple('EpisodeStep',
                     field_names = ['observation','action'])
```

- `Episode`의 `steps` 에는 `EpisodeStep`들이 들어감.
- `batch` 리스트 안에 `Episode`들이 들어감.

```
def iterate_batches(env, net, batch_size):
        # batch 리스트 안에 episode 들이 들어가게 됨.
        batch = []
        # current episode reward
        episode_reward = 0.0
        # current episode steps
        episode_steps = []
        obs = env.reset()
        sm = nn.Softmax(dim = 1)
```

CartPole 예제로 알아보기 (iterate_batches 함수)

```
    while True:
        obs_v = torch.FloatTensor([obs])
        act_probs_v = sm(net(obs_v))
        act_probs = act_probs_v.data.numpy()[0]
```

- observation을 network input 으로 넣어주기 위해 batch(2-dimentional tensor)로 만든다(브라켓 한번 더 씌움).
- `act_probs_v` 는 tensor 이므로, tensor.data 를 array로 만들어 받아와서 접근한다. input과 같은 구조의 2차원 배열이 되므로 0번째 element 가 원하는 vector 이다.

```
# 이 확률로 action 선택한다.
action = np.random.choice(len(act_probs), p = act_probs)
next_obs, reward, is_done, _ = env.step(action)
# 기록
episode_reward += reward
episode_steps.append(
        EpisodeStep(observation = obs, action = action))

# episode 하나가 끝남.
if is_done:
    batch.append(        #기록
        Episode(reward = episode_reward,
                        steps = episode_steps))
    # 초기화
    episode_reward = 0.0
    episode_steps = []
    next_obs = env.reset()
    if len(batch) == batch_size:
        yield batch      # 배치가 하나 끝나면 일단 함수를 빠져나감
        # 다음 함수 호출 시엔 여기서부터 진행
        batch = []

obs = next_obs
```

cf. Generator 와 yield

- 파이썬에서 방대한 data 를 처리 할 때, list 방식으로는 메모리에 모두 올리지 못하기 때문에 iterator 혹은 generator 방식으로 하게 된다.
- iterator 는 `next()` 를 통해 순차적으로 값을 가져오는 object 이다.
- generator function 은 중간에 iterator 를 심을 수 있는 함수이다.
- `yield` 키워드 다음 값을 return 하게 되며, 이후 generator 의 `next()` 호출 시에 그 다음 줄 부터 실행하게 된다.
- 메모리 절약 , lazy evaluation.

CartPole 예제로 알아보기

- agent의 network training과 episodes의 생성(data gathering)이 동시에 진행된다.
- 한 배치(16 episodes)가 끝날 때 마다 yield 를 통해 함수 밖으로 빠져나간 후, caller 에서 gradient descent method로 네트워크를 학습시킨다.
- 다음 generator function(`iterate_batches()`) call 때에는 개선된 network를 사용하게 된다.

CartPole 예제로 알아보기 (filter_batch 함수)

```
def filter_batch(batch, percentile):
        rewards = list(map(lambda s : s.reward, batch))
        reward_bound = np.percentile(rewards, percentile)
        reward_mean = float(np.mean(rewards))

        train_obs = []
        train_act = []
        for example in batch: # 각각의 에피소드에 대해서
                if example.reward < reward_bound:
                        continue
                # iterable 객체의 elements 를 append
                train_obs.extend(map(
lambda step : step.observation, example.steps))
                train_act.extend(map(
lambda step : step.action, example.steps))

        train_obs_v = torch.FloatTensor(train_obs)
        train_act_v = torch.LongTensor(train_act)
        return train_obs_v, train_act_v,
                reward_bound, reward_mean
```

CartPole 예제로 알아보기 (main 함수)

```
if __name__ == "__main__":
        env = gym.make("CartPole-v0")
        obs_size = env.observation_space.shape[0]
        n_actions = env.action_space.n

        net = Net(obs_size, HIDDEN_SIZE, n_actions)
        # loss function(type of objective function) 지정.
        objective = nn.CrossEntropyLoss()
        optimizer = optim.Adam(
                params = net.parameters(), lr = 0.01)
```

```
for iter_no, batch in enumerate(
                  iterate_batches(env, net, BATCH_SIZE)):

    # 각각의 batch에 대해 elite episodes 의 data 만 받아 온다.
    obs_v, acts_v, reward_b, reward_m = filter_batch(
                                        batch, PERCENTILE)
    optimizer.zero_grad()
    # filtered data 가운데 observations를 네트워크에 input 한다.
    action_scores_v = net(obs_v)
    # output 과 desired output을 손실(목적)함수에 넣는다.
    loss_v = objective(action_scores_v, acts_v)
    # loss value 를 back propagation 후 weight 갱신.
    loss_v.backward()
    optimizer.step()

    print("%d: loss = %.3f, reward_mean=%.1f, reward_boun
% (iter_no, loss_v.item(), reward_m, reward_b))

    if reward_m > 199:
        print("Solved!")
        break
```

- `enumerate` 은 count와 리스트 element 를 전달

CartPole 예제로 알아보기

- `reward_m` 이 199보다 크면 반복문을 종료한다.
- CartPole 게임의 타임리밋은 200(reward) 이며, 지난 100개 episodes 의 reward 평균이 195 이상이라면 문제를 해결했다고 생각할 수 있다.
- 학습이 잘 되었다고 판단되었을때 학습을 종료한다.

결과 화면

```
0: loss = 0.696, reward_mean=19.3, reward_bound=21.5
1: loss = 0.684, reward_mean=27.7, reward_bound=30.5
2: loss = 0.667, reward_mean=33.8, reward_bound=40.5
3: loss = 0.661, reward_mean=47.6, reward_bound=57.0
4: loss = 0.648, reward_mean=46.2, reward_bound=60.5
5: loss = 0.631, reward_mean=41.6, reward_bound=44.5
....
28: loss = 0.571, reward_mean=171.4, reward_bound=199.0
29: loss = 0.562, reward_mean=161.4, reward_bound=200.0
30: loss = 0.574, reward_mean=167.2, reward_bound=200.0
31: loss = 0.565, reward_mean=167.5, reward_bound=200.0
32: loss = 0.563, reward_mean=156.1, reward_bound=200.0
33: loss = 0.575, reward_mean=154.4, reward_bound=199.5
34: loss = 0.564, reward_mean=170.2, reward_bound=200.0
35: loss = 0.563, reward_mean=193.9, reward_bound=200.0
36: loss = 0.573, reward_mean=185.1, reward_bound=200.0
37: loss = 0.562, reward_mean=199.7, reward_bound=200.0
Solved!
```

Cross-Entropy method 의 한계

다른 예제 : FrozenLake Game

- FrozenLake 는 4* 4 격자 위에서 start 에서 출발해 goal 까지 이동하는 게임이다.
- 중간에 hole 과 만나면 에피소드가 종료된다.
- reward 는 goal 에 도착했을 때에만 '1' 받을 수 있다.
- 한번에 전후좌우로 한칸씩 이동할 수 있지만, 33%의 확률로만 정확한 곳에 이동하며 나머지 경우에는 가고자 하는 방향의 오른쪽(33%) 혹은 왼쪽(33%)으로 미끄러진다.

```
SFFF       (S: starting point, safe)
FHFH       (F: frozen surface, safe)
FFFH       (H: hole, fall to your doom)
HFFG       (G: goal, where the frisbee is located)
```

다른 예제 : FrozenLake Game

- 이 게임을 cross entropy method(Cartpole 의 학습 코드)로 학습시킨다면 적절히 수렴하지 못해 agent 의 수행능력이 개선되지 않는다.

1. CartPole 에서는 실행시간이 늘어날수록 reward 가 늘어남. FrozenLake 은 goal 에 도착하여 reward 를 받기전에는 action이 '얼마나 좋은지'를 판단할 수 없음.
2. CartPole 에피소드들의 reward는 정규분포에 가까움. FrozenLake 는 에피소드 reward 가 0 또는 1뿐이며, 0인 에피소드가 대부분이다. 따라서 cartpole 처럼 percentage 로 elite episodes를 고르는 방식(cross entropy)이 잘 되지 않는다.
3. 불확정성(미끄러짐)

Cross-entropy method 의 한계

- 에피소드가 유한해야 하며 짧으면 더 좋다.
- 에피소드들의 total rewards의 분포가 적절해야 한다.
- agent가 '잘하고 있는지'를 알 수 있는 중간 지표가 있어야 좋다.

FrozenLake 를 Cross-entropy 방식으로 풀기 위해선...

- 성공한 episode (reward 1)을 얻기 위해 batch 사이즈를 늘린다.
- reward에 discount factor 를 적용한다 : 짧은 경로로 도착 하는 것이 더 좋은 reward 를 받게 된다.
- elite episode를 더 오래 저장하며 학습시킨다. (희귀하므로)
- 오래 학습시킨다. (50% 성공하려면 5000 iterations)

끝