turtlebot3_machine_learning

Rough overview of turtlebot3_d3qn → turtlebot3_PERd3qn

Understanding the code : commented with link provided

https://github.com/laphisboy/RL_fall/blob/master/fall_week_8/edit5_turtlebot3_dqn_stage.py (minor fixes not yet updated)

https://github.com/laphisboy/RL_fall/blob/master/fall_week_8/edit6_turtlebot3_dqn_stage.py (fixes for modules python not available in python2.7)

What has been added for PER?

Quick peek at burger's environment code

1. Append Memory

```
def appendMemory(self, state, action, reward, next_state, done):
    #(edit5)
    td_error = reward + self.discount_factor * np.max(self.target_model.predict(next_state.reshape(1, len(next_state)))) - self.model.predict(state.reshape(1, len(state)))

    p = np.abs(td_error + 0.0000001) ** self.alpha
    #(edit5)

    self.memory.append((state, action, reward, next_state, done))

    #(edit5)
    self.priority.append(p)
    #(edit5)
```

1. Calculate error 2. calculate P 3. Append P seperately

$$p_i = \left| r + \gamma \max_{a'} Q(s_{i+1}, a'; \boldsymbol{w}^-) - Q(s_i, a_i; \boldsymbol{w}) \right| \quad P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Basic overview

Quick peek at burger's environment code

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

2. Retrieve Batch

```
def get_PER_batch(self):
    p_sum = np.sum(self.priority)
    prob = self.priority / p_sum

    sample_indices = choices(range(len(prob)), k=self.batch_size, weights=prob)

    importance = (1/prob) * (1/len(self.priority))
    importance = np.array(importance)[sample_indices]
    samples = np.array(self.memory)[sample_indices]

    return samples, importance
```

1. Retrieve according to amount of prioritization
2. "importance" is for importance sampling

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)$$

Basic overview

Quick peek at burger's environment code

3. Train with IS

```
i = importance[i] ** self.beta

self.model.fit(X_batch, Y_batch, batch_size=self.batch_size, epochs=1, verbose=0, sample_weight=importance_batch)
```

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^{\beta}$$

Basic overview

Quick peek at DQN code

4. Because trying to do python3 in python2.7 …

```
from itertools import repeat as _repeat
import operator
from bisect import bisect as _bisect

def accumulate(iterable, func=operator.add):
    it = iter(iterable)
    total = next(it)
    yield total
    for element in it:
        total = func(total, element)
        yield total

def choices(population, weights=None, cum_weights=None, k=1):
    n = len(population)
    if cum_weights is None:
        if weights is None:
            _int = int
            n += 0.0
            return [population[_int(random.random() * n)] for i in _repeat(None, k)]
        cum_weights = list(accumulate(weights))
    elif weights is not None:
        raise TypeError('Cannot specify both weights and cumulative weights')
    if len(cum_weights) != n:
        raise TypeError('The number of weights does not match the population')
    bisect = _bisect
    total = cum_weights[-1] + 0.0
    hi = n - 1
    return [population[bisect(cum_weights, random.random() * total, 0, hi)] for i in _repeat(None, k)]
```

Added modified source code for itertools.accumulate and random.choices