Практическая работа

Использование моделей для методов искусственного интеллекта

Начнём с марковской цепи для МППР, лежащего в основе большинства алгоритмов обучения с подкреплением. Для оценивания стратегии используется уравнение Беллмана. В основном применяется два подхода к решению МППР: итерация по ценности и итерация по стратегии (политике).

Марковская цепь описывает последовательность событий, обладающую марковским свойством. Она состоит из множества допустимых состояний S = {s0, s1, ... , sm} и матрицы переходов T(s, s'), элементами которой являются вероятности перехода из состояния s в состояние s'. Марковское свойство означает, что будущее состояние процесса зависит только от его текущего состояния и не зависит от прошлых состояний. Иными словами, состояние процесса в момент t + 1 зависит только от состояния в момент t.

МППР основан на марковской цепи и включает агента и процесс принятия решений. Разработаем МППР и вычислим функцию ценности при оптимальной стратегии.

Помимо множества допустимых состояний S = {s0, s1, ... , sm}, в определение МППР входит множество действий A = {a0, a1, ... , an}, модель переходов T(s, a, s'), функция вознаграждения R(s) и коэффициент обесценивания ***γ***. Матрица переходов T(s, a, s') содержит вероятности выбора действия **a** в состоянии **s**, которое переводит процесс в состояние **s'**. Коэффициент обесценивания ***γ*** определяет компромисс между вознаграждениями в ближайшем и отдалённом будущем.

Действия, выбираемые на разных шагах, необязательно совпадают. Обычно они зависят от состояния. Решаем МППР, т. е. находим оптимальную стратегию в реальных ситуациях. Функция ценности стратегии измеряет, насколько агенту выгодно находиться в каждом состоянии при следовании данной стратегии: чем выше ценность, тем лучше состояние. МППР считается решенным, если найдена оптимальная стратегия.

На практике применим подход – оценивание стратегии. Оценивание стратегии – итеративный алгоритм. Оценку начинаем с произвольных ценностей, а затем итеративно улучшаем их, опираясь на уравнение математического ожидания Беллмана, добиваясь сходимости. На каждой итерации ценность состояния ***s*** при следовании стратегии **π** обновляется по формуле:

$$V(s) := \sum_{a} \pi(s,a)\left[R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s')\right]$$

- здесь **π(s, a)** обозначает вероятность выбора действия **a** в состоянии **s** при следовании стратегии **π;**
T(s, a, s') – вероятность перехода из состояния **s** в состояние **s'** в результате выбора действия **a**;
R(s, a) – вознаграждение, полученное в состоянии **s** при выборе действия **a**.

Остановить итеративный процесс обновления можно двумя способами. Первый – задать фиксированное число итераций, скажем 1000 или 10 000. Второй – задать пороговое значение (обычно 0.0001 или что-то в этом роде) и прекращать процесс, когда ценности всех состояний изменяются на величину, меньшую порога.

Функция оценивания стратегии выполняет следующие действия:

- инициализирует ценности всех состояний нулями;

- обновляет ценности в соответствии с уравнением математического ожидания Беллмана;

- вычисляет максимальное изменение ценностей по всем состояниям;

- если максимальное изменение больше порога, то процесс обновления продолжается. В противном случае оценивание завершается, и возвращаются ценности, вычисленные на последней итерации.

Поскольку алгоритм оценивания стратегии приближенный, результат может отличаться от полученного с помощью обращения матрицы, но нам и не нужна большая точная функции ценности.

Оценивание стратегии используется для предсказания результатов стратегии, а не для решения задач управления!

Найти оптимальную стратегию можно, применив алгоритм итерации по ценности. Его идея примерно такая же, как в алгоритме оценивания стратегии – итеративный алгоритм. В начале работы ценности состояний произвольны, а затем обновляются с применением уравнения оптимальности Беллмана, пока не сойдутся. На каждой итерации вместо вычисления математического ожидания (среднего) ценности по всем действиям выбирается действие, при котором ценность оказывается максимальной:

$$V^*(s) := \max_a \left[R(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s') \right]$$

- здесь **V*(s)** обозначает оптимальную ценность состояния, т. е. ценность при следовании оптимальной стратегии; **T(s, a, s')** – вероятность перехода из состояния ***s*** в состояние ***s***' при выборе действия ***a***;
R(s, a) – вознаграждение, полученное в состоянии **s** при выборе действия ***a***.

Вычислив оптимальные ценности, легко можно получить оптимальную стратегию:

$$\pi^*(s) := \operatorname{argmax}_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

Алгоритм итерации по ценности находит оптимальную функцию ценности, итеративно применяя уравнение оптимальности Беллмана.

Далее показана форма уравнения оптимальности Беллмана, которую можно использовать, когда вознаграждения, начисляемые средой, зависят от нового состояния.

Обучение на основе временных различий (TD-обучение), как и обучение методом Монте-Карло, является безмодельным алгоритмом. Напомним, что при обучении методом Монте-Карло Q-функция обновляется в конце эпизода. Но главное достоинство TD-обучения состоит в том, что Q-функция обновляется на каждом шаге эпизода.

В нашем примере рассмотрим популярный TD-метод – Q-обучение. Это алгоритм обучения с разделенной стратегией. В нем Q-функция обновляется по следующей формуле:

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

- где **Q(s, a)** – состояние, в которое переходит среда после выполнения действия ***a*** в состоянии **s**;
r – начисленное вознаграждение; **α** – скорость обучения; **γ** – коэффициент обесценивания.

Компонент **max Q(s', a')** означает, что поведенческая стратегия жадная, т. е. для генерирования обучающих данных выбирается действие с наибольшим значением Q-функции в состоянии ***s***. В методе Q-обучения действия выбираются в соответствии с *ε-жадной* стратегией.

На шаге *ε-жадная* стратегия принимает параметр ε от 0 до 1 и количество возможных действий |A|.
С вероятностью ε/|A| действие выбирается произвольное действие, а с вероятностью 1 – ε + ε/|A| действие – с наибольшей ценностью пары «состояние–действие». Выполняется Q-обучение:

инициализируем таблицу значений Q-функции нулями;
в каждом эпизоде агент выбирает действие, следуя ε-жадной стратегии.

После каждого шага Q-функция обновляется;

выполняем n_episode эпизодов;
получаем оптимальную стратегию на основе оптимальной Q-функции.

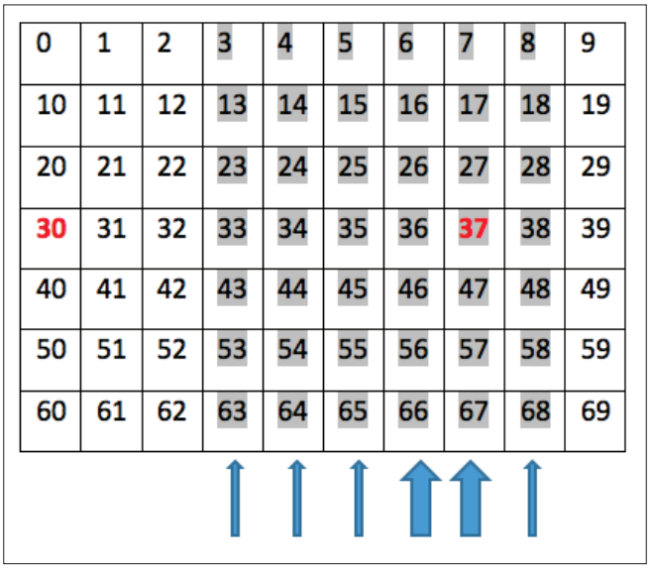
Оптимальная стратегия показывает, что из начального состояния 36 агент делает шаг вверх в состояние 24, затем идет вправо до состояния 35 и, делая шаг вниз, достигает цели.

Как видим, алгоритм Q-обучения оптимизирует Q-функцию, обучаясь на опыте, сгенерированном другой стратегией. Это очень похоже на управление методом Монте-Карло с разделенной стратегией. Разница в том, что Q-функция обновляется после каждого шага, а не в конце эпизода. Это лучше работает в окружающих средах, где эпизоды длинные и ждать, пока эпизод завершится, неэффективно. На каждом шаге Q-обучения (или любого другого TD-метода) мы получаем новую информацию об окружающей среде и используем эту информацию для немедленного обновления ценностей. В нашем примере для нахождения оптимальной стратегии понадобилось всего 500 эпизодов.

Для нахождения оптимальной стратегии понадобилось около 100 эпизодов. Мы можем вывести на график длину каждого эпизода. Также можно графически представить, как изменялась величина вознаграждения.

Займемся сложной окружающей средой Windy Gridworld - на сетке, в которой действует внешняя сила, сдувающая агента из некоторых ячеек. Применим TD-метод для поиска оптимальной стратегии в этой среде.

Среда Windy Gridworld развернута на пространственной сетке 7×10, изображенной на рисунке ниже:



Агент может двигаться вверх, вправо, вниз и влево. В примере он находится в ячейке 30, а цель – попасть в ячейку 37, в этот момент эпизод заканчивается. За каждый сделанный шаг агенту начисляется вознаграждение (-1). Поэтому требуется добраться до цели как можно раньше.

Сложность этой среды в том, что в столбцах с 4 по 9 дует ветер. Если агент оказывается в любой ячейке из этих столбцов, то его будет сдувать вверх. Сила ветра в седьмом и восьмом столбцах = 2, а в четвертом, пятом, шестом и девятом = 1. Например, если агент попытается сделать шаг вправо из состояния 43, то окажется в состоянии 34. Сделав шаг влево из состояния 48, агент окажется в состоянии 37. Сделав шаг вверх из состояния 67, он окажется в состоянии 37, потому что ветер сдует его еще на две ячейки вверх. Если же он сделает шаг вниз из состояния 27, то окажется в состоянии 17, потому что ветер сдувает на две ячейки вверх, пресекая попытку спуститься.

В примере для среды определяются пространство наблюдений, ветреные участки и сила ветра, матрицы переходов и вознаграждений и начальное состояние.

Определим метод, который возвращает результат действия: вероятность (она всегда равна 1), новое состояние, вознаграждение (всегда равно –1) и признак завершения эпизода.

В примере вычисляется новое положение агента, зная текущее состояние, выбранное действие (шаг) и эффект ветра. При этом гарантируется, что новое положение не выйдет за пределы сетки. В конце проверяется, достиг ли агент конечной цели.

Определим метод, который предотвращает выход агента за пределы сеточного мира, и добавим метод для отображения сетки и политики агента.

В нашем примере решим задачу, применив TD-метод управления.

Решим МППР методом TD-обучения с единой стратегией – SARSA (State-Action-Reward-State-Action).

Как и Q-обучение, SARSA завязан на ценности пар «состояние–действие». Обновление Q-функции производится по следующей формуле:

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma\, Q(s', a') - Q(s, a))$$

здесь
s' – состояние, в которое переходит среда после выбора агентом действия a в состоянии s;
r – полученное при этом вознаграждение;
α – скорость обучения,
γ – коэффициент обесценивания.

В SARSA для обновления значения Q-функции мы выбираем следующее действие a', следуя ε-жадной стратегии. Найдем оптимальную стратегию взаимодействия со средой методом SARSA.

- Определим ε-жадную поведенческую стратегию.

- Зададим количество эпизодов и инициализируем два списка для хранения длин эпизодов и полученных в них вознаграждений.

- Определим функцию, реализующую алгоритм SARSA.

- Зададим коэффициент обесценивания 1, скорость обучения 0.5 и ε = 0.1

Выполним алгоритм SARSA с этими параметрами и выведем оптимальную стратегию.

На первом шаге функция sarsa выполняет следующие действия:

- инициализирует нулями таблицу значений Q-функции;

- в каждом эпизоде позволяет агенту следовать ε-жадной стратегии при выборе действия.

- на каждом шаге обновляет Q-функцию по формуле

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$

в которой a' выбирается, следуя $\boldsymbol{\varepsilon}$-жадной стратегии, и выполняет новое действие a' в новом состоянии s';

- прогоняет n эпизодов;

- строит оптимальную стратегию на основе оптимальной Q-функции.

Алгоритм SARSA оптимизирует Q-функцию, выбирая действие в соответствии с ε-жадной стратегией. Особенность алгоритма в том, что Q-функция обновляется небольшими приращениями на каждом шаге, а не в конце всего эпизода.

Это считается преимуществом для окружающих сред с длинными эпизодами, когда ждать завершения эпизода неэффективно. На каждом шаге SARSA мы получаем новую информацию о среде и сразу же используем ее для обновления ценностей. В нашем примере для нахождения оптимальной стратегии понадобилось всего 500 эпизодов.

Как видно из примера, для получения оптимальной стратегии хватило примерно 200 эпизодов. Чтобы убедиться в этом, построим графики длин и полных вознаграждений для каждого эпизода.

Определим два списка: для хранения длин эпизодов и полученных в них вознаграждений:

length_episode = [0] * n_episode

total_reward_episode = [0] * n_episode

В процессе обучения будем сохранять длину эпизода и вознаграждение в нем.

Можно увидеть, что длина эпизодов стабилизируется примерно после 200 эпизодов. Небольшие флуктуации связаны со случайным исследованием в ε-жадной стратегии.

Нарисуем график зависимости вознаграждения от времени.

Чем меньше значение ε, тем слабее флуктуации, являющиеся результатом случайного исследования в ε-жадной стратегии.

Организуем программный вариант выбора наилучшего набора гиперпараметров в обучении с подкреплением.

Качество набора измеряется по следующим показателям:

- среднее полное вознаграждение в нескольких первых эпизодах: мы хотим, чтобы вознаграждение стало большим как можно быстрее;

- средняя длина нескольких первых эпизодов: мы хотим, чтобы решение достигалось как можно быстрее;

- среднее вознаграждение на одном временном шаге в нескольких первых эпизодах: мы хотим, чтобы это вознаграждение достигло максимума как можно быстрее.

Теперь используем три потенциальных значения альфа= [0.4, 0.5, 0.6], и три потенциальных значения эпсилон= [0.1, 0.03, 0.01]. Будем рассматривать только первые 500 эпизодов.

Выполните поиск наилучшего набора гиперпараметров для обучения.
Текст скрипта Python есть в разделе дисциплины на сайте кафедры.

Сохраните результаты (графики, таблицы, программный код) в отчет, пришлите на почту преподавателю.