Проект «Учимся играть в нарды»

Павел Хрушков, 1 *курс ПМИ ФКН ВШЭ*

Цели и задачи проекта

Основная цель проекта – с помощью машинного обучения создать стратегию, которая бы могла уверенно выигрывать в нарды у *случайной* стратегии, а в дальнейшем – и у человека (скажем, новичка игры в нарды).

Под стратегией понимается класс, который по набору возможных ходов на текущий момент умеет определять наиболее оптимальный. Случайная стратегия равновероятно выбирает из всех возможных ходов единственный. Таким образом, с помощью случайной стратегии можно играть по правилам, но к победе над любой нетривиальной стратегией (а тем более над человеком) она никогда не приведёт.

Тем не менее, случайная стратегия позволяет оценивать «успешность» другой по количеству выигрышей у неё в N партиях.

Первой задачей было написать класс, который бы, основываясь на

- значениях зар;
- текущем положении доски;
- цвете игрока,

мог бы генерировать все возможные *ходы*. Под ходом понимается список всех передвижений фишек, каждое из которых задается двумя числами: с какой и на какую позицию передвинуть фишку.

Следующий шаг – создание *арбитра*, который принимает две стратегии: одна играет за белых, другая – за черных. Арбитр может, сыграв N партий, определить вероятность победы белых при игре с чёрными.

Стратегии

Случайная стратегия

Для проверки корректности работы арбитра и случайной стратегии была посчитана вероятность победы случайной стратегии над случайной. Математические факты говорят о том, что эта вероятность должна быть равна 50%. Арбитр выдал результат 49,62%, что хорошо сходится с законом больших чисел.

Линейная регрессия

Метод заключается в том, чтобы по положению доски генерировать некоторое количество факторов (в работе были использованы такие: сколько фишек каждого цвета уже выкинуто из дома, сколько белых/чёрных фишек стоит на i-ой позиции, сколько подряд клеток занято фишками одного цвета), и на основе этих факторов $x_1, ..., x_k$ построить линейную зависимость вероятности победы $p(x_1, ..., x_k)$ от $x_1, ..., x_k$:

$$p(x_1, ..., x_k) = a_1 x_1 + ... + a_k x_k .$$

Задача: найти параметры $a_1, ..., a_k$. Было решено сделать следующее: сгенерировать dataset и обучить линейную регрессию с помощью пакета scikit-learn в Python. Вероятность победы белых по положению доски высчитывалась следующим образом: из данного положения доски было сыграно *k* партий и посчитано, в каких *m* из них выигрывали белые. Вероятностью победы белых при этом положении доски было число *m*/*k*. В итоге были сгенерированы параметры, и модель, основанная на линейной регрессии, побеждала у случайной стратегии в 92% случаев.

Time-Difference Learning

Эта модель основывалась на том желании, чтобы после очередного хода игрока вероятность выигрыша не сильно изменялась. Пусть мы ищем зависимость $p(x_1, ..., x_k) = a_1x_1 + ... + a_kx_k$. Обозначим за X_k набор параметров после *k*-ого хода в партии. Тогда мы хотим минимизировать функцию $(p(X_{k+1}) - p(X_k))^2$ после каждого хода, изменяя соответствующим образом параметры $a_1, ..., a_k$. Причем принималось, что вероятность *p* после последнего хода равна 0, если мы проиграли, и 1 в противном случае.

Из этих соображений и выбирались параметры линейной модели с помощью метода градиентного спуска. Результаты оценки «успешности»: выигрывает случайную стратегию в 85% случаев, но оказывается хуже модели, основанной на линейной регрессии в 72% партий.

Двухслойные нейронные сети

Используя уже собранный dataset, можно обучить нейронную сеть. В проекте была обучена сеть с двумя скрытыми слоями размером 10 каждый и функцией активации $\sigma(x)$ – сигмоид. В качестве алгоритма обучения нейронной сети был выбран алгоритм Kingma, Diederik и Jimmy Ba. Сеть обучалась с помощью scikit-learn 0.18.dev0 в Python.

Результаты «успешности» стратегии, использующей нейронные сети:

	Случайная стратегия	Линейная регрессия	Time-Difference Learning
Процент выигрыша	96%	88%	96%

Такое странное распределение вероятностей можно объяснить тем, что Time-Difference Learning «изучил» промахи случайной стратегии и способен составить достойную конкуренцию только этой стратегии. Линейная же регрессия находила закономерную зависимость между положением фишек на доске и вероятностью победы.

Трёхслойные нейронные сети

С теми же параметрами, что и у двуслойной сети, но с добавленным третьим слоем размера 10, можно построить трехслойную сеть и обучить на том же dataset. Ниже приведена таблица сравнения трехслойной нейронной сети с другими стратегиями:

	Случайная стратегия	Линейная регрессия	Time-Difference Learning	2-слойная нейронная сеть
Процент выигрыша	63%	42%	57%	31%

Как видно, попытка просто добавить один слой не увенчалась успехом. Вообще, от количества слоев нейронной сети зависит её «интеллектуальность». Результат же обучения нейронной сети зависит от размера dataset, и, по всей видимости, 100 000 данных не хватило для обучения трехслойной сети на хорошем уровне.

Визуализация

В ходе проекта была сделана визуализация игры в нарды, чтобы и человек мог сразиться со стратегией. Причем стратегия передается как аргумент к классу `Player`, т.е. созданный класс визуализации не зависит от стратегии (главное, чтобы она наследовала `IStrategy`).