RENJU

Андрей Стороженко

Higher School of Economics

2017

Аннотация

В последнее время сфера машинного обучения развивается крайне стремительно. Создание искусственного интеллекта для различных игр давно интересует ученых в этой области. Стоит вспомнить матч DeepBlue (разработка IBM) против Гарри Каспарова в шахматы в 1996-1997, недавние сражения сильнейших игроков в го против AlphaGo (разработка DeepMind). Одни из популярных на сегодняшний день разделов в машинном обучении — глубокое обучение и обучение с подкреплением, а в рамках рассматриваемой специфики широко применяется метод Монте-Карло. В данном проекте мы изучили эти темы и применили их, написав программу, которая играет в рендзю. За основу был взят алгоритм AlphaGo, как сильнейший алгоритм для сложных настольных игр.

I. Введение

В рамках поставленной цели возникает необходимость решить следующие задачи:

1. Изучение теории машинного обучения (gradient descent, neural networks, deep learning, reinforcement learning, Monte-Carlo tree search).
2. Создание модели для игры в рендзю[1] на примере AlphaGo.
3. Сбор данных.
4. Обучение алгоритма.
5. Создание графического интерфейса программы.

Для решения поставленных задач мы использовали следующие программные решения:

- Основной язык — Python 3.5. Выбор обусловлен крайне удобными библиотеками для работы с данными и с линейной алгеброй (numpy), на которой построены используемые нами алгоритмы машинного обучения, стремительным развитием самого языка.
- Нейронные сети мы обучали с помощью TensorFlow (достаточно простое и эффективное решение). Для этого использовалась библиотека Keras.
- Для улучшения скорости вычислений были задействованы вычисления на GPU (NVIDIA GeForce GTX 1060) с помощью технологии NVIDIA CUDA.
- Модуль tkinter для создание простого и удобного интерфейса программы.

[1] Для того, чтобы сконцентрироваться на работе над алгоритмом, мы использовали упрощенную версию игры: без ограничений на ходы

II. Алгоритм для игры в го

Наше решение использует методы, описанные в статье про AlphaGo [2]. DeepMind в своем алгоритме применили следующие решения.

Была создана сверточная нейронная сеть policy, которая по состоянию доски возвращала вероятность хода для каждой клетки, и обучена на партиях профессиональных игроков. Сеть policy была дообучена с помощью обучения с подкреплением: на каждой итерации проводилась игра с одной из предыдущих версий сети и менялись параметры текущей сети.

Была создана сверточная сеть value, которая оценивала позицию, то есть предсказывала, сколько партий выиграет каждая сторона, если оба игрока придерживаются вероятностной стратегии, задаваемой policy сетью. Для ускорения разбора по дереву использовался более быстрый аналог policy сети — нейросеть rollout с меньшим числом

[2] `https://storage.googleapis.com/deepmind-media/alphago/AlphaGoNaturePaper.pdf`

параметров.

Процесс симуляции для выбора хода заключался в следующем. Алгоритм начинал строить дерево из текущей позиции. Каждую впервые посещенную позицию он оценивал сетью value, после чего запускалась быстрая игра с использованием сети rollout и запоминался результат. При выборе хода из посещенной ранее позиции использовались вероятности сети policy, оценки сети value и количество выигранных и проигранных партий при игре сетью rollout. По окончании симуляции возвращался ход, который выбирался наибольшее число раз.

III. Модель

Наш алгоритм состоит из двух основных частей: обученные на готовых данных нейронные сети и разбор по дереву. Из сетей мы использовали аналоги policy и rollout, описанные в статье.

i. Policy и rollout сети

Сеть policy = $\pi_\theta(s)$ и rollout = $r_\theta(s)$ принимают на вход позицию s в виде матрицы размера $15 \times 15 \times 4$ из нулей и единиц. Первый слой содержит единицы в занятых клетках, второй слой — в клетках, занятых игроком, который ходит в текущей позиции, третий — занятых его противником. Четвертый слой полностью состоит из единиц. Обе сети возвращают вектор размерности 225, где в компоненте, соответствующей ходу a, стоит вероятность $\pi_\theta(s,a)$ (и $r_\theta(s,a)$ соответственно) сделать этот ход.

Сеть π_θ состоит из четырех сверточных слоев со сверткой размера 5×5 и трех полносвязных. Также присутствуют два слоя *MaxPooling* с окном 2×2. Всего в сети около 15 миллионов обучаемых параметров. Сеть r_θ состоит из трех сверточных слоев, трех полносвязных и слоя *MaxPooling* такого же размера. Всего менее 700 тысяч параметров.

Обе сети обучались на готовых данных. Из партии создавались пары (s, v), где s — позиция в нужном размере, а v — вектор размера 225, где на месте сделанного в позиции хода стоит 1, а во всех остальных координатах 0. Эта пара добавлялась в обучающую выборку. После этого происходила трансформация пары: позиция случайным образом поворачивалась и сдвигалась на случайный целый вектор в пределах доски. Номер сделанного хода также менялся, и новая пара добавлялась в выборку. Это необходимо для того, чтобы сеть могла корректно обрабатывать позиции, когда игра начинается не в центральной клетке (а в выборке все партии начинаются в цетре доски).

Всего каждая сеть была обучена на около 200 миллионах пар. Качество обучения π_θ и r_θ в метрике «доля правильных ответов» составило около 40%.

ii. Дерево разбора Монте-Карло

Когда игра находится в позиции s и алгоритм должен сделать ход, начинается разбор по дереву. Вершины дерева — это возможные позиции как для белых, так и для черных. Текущей позиции соответствует корень дерева s_{root}. В каждой посещенной вершине записывается цвет вершины, сразу предсказываются вероятности $\pi_\theta(s,a)$ для всех действий, создается массив для подсчета количества выборов каждого из возможных ходов $N(s,a)$ и для суммы результатов симуляций игр при выборе хода для каждого из ходов $R(s,a)$. Если у текущей вершины нет действий, которые совершались ранее, вычисляется π_θ и выбирается возможное действие с максимальной вероятностью, после чего запускается симуляция с помощью сети r_θ. Если же вершина уже была обследована, новое действие подбирается по следующей формуле:

$$a = \mathsf{argmax}_{a'}[R(s,a') > 0.85N(s,a')] +$$

$$+ [R(s,a') > 0.25N(s,a')] + \frac{R(s,a') + 10\pi_\theta(s,a')}{1 + N(s,a')}.$$

Таким образом, будет выбран ход с большей вероятностью $\pi_\theta(s,a)$, с большим количеством выигрышей и с меньшим количеством посещений (отвечает за исследование новых ходов). Однако, эта величина может быть отрицательна для всех ходов, и будет выбрано занятое поле (для него эта величина всегда ноль). В этом случае необходимо выбрать действие еще раз по формуле

$$a = \mathsf{argmax}_{a'} \frac{\pi_\theta(s,a')}{1 + N(s,a')}.$$

Зависимости от вероятностей и посещений аналогичны. Игра с помощью сети r_θ происходит следующим образом: в каждой позиции ход выбирается случайно с вероятностным распределением, пропорциональным $r_\theta(s)$. При этом, если в позиции возможно выиграть, сразу выбирается выигрышный ход. Одна симуляция заканчивается в следующих случаях: победа черных ($r = +1$), победа белых ($r = -1$), ничья или глубина от корня дерева до позиции превысила 10 ($r = 0$). Далее всем посещенным вершинам в дереве, по которым проходила симуляция, передается награда r. В зависимости от цвета соответствующее значение $R(s, a)$ изменяется.

По окончании разбора ход из выбирается следующим образом: если можно выиграть сразу, делается выигрышный ход. В остальных случаях имеется два варианта. При наличии такого действия a, что $N(s_{root}, a) > 50$ и $R(s_{root}, a) > 0.5N(s_{root}, a)$, из всех таких ходов выбирается один с наибольшим отношением R/N (если один из часто посещаемых ходов имеет хорошую «ценность», имеет смысл выбрать его). Если такого хода не нашлось, делается ход, который выбирался при поиске по дереву наибольшее число раз.

IV. Результаты

Проводилось заключительное соревнование с другими алгоритмами[3] *Lolita*, *John*, *Hardy*, *Elena*[4] по следующим правилам:

1. Каждый играет с каждым одну партию за белых и одну за черных.
2. За победу начисляется 3 балла, ничья стоит 1 балл.
3. В случае ошибки времени исполнения игроку засчитывается поражение.
4. Время на ход ограничено сверху 10 секундами.
5. Игра не может длиться более 60 ходов. В случае превышения ограничения партия заканчивается ничьей.

Вышеописанное решение *Pierre* показало следующий результат:

Player	P	H	L	J	E	Pts
Pierre	-	6	6	6	6	24
Hardy	0	-	3	4	6	13
Lolita	0	3	-	4	4	11
John	0	1	1	-	3	5
Elena	0	0	1	3	-	4

V. Итоги

В рамках работы над прокетом были изучены следующие темы: градиентный спуск и реализация различных его вариантов; нейросети и их обучение; обучение с подкреплением; метод поиска по дереву Монте-Карло.

Рассмотренные темы позволили нам реализовать алгоритм для игры в рендзю. Он играет на уровне с человеком, выиграть его достаточно сложно. При этом не использовалось никакой теории о игре в рендзю. Результаты соревнования показывают, что выбранный алгоритм уверенно выигрывает и у других схожих решений.

[3] Написаны другими участниками проекта
[4] Имена ботов сохранены