Караев Александр Дмитриевич
магистрант
Караева Дария Александровна
магистрант
ФГАОУ ВО «Московский физико-технический
институт (государственный университет)»
г. Долгопрудный, Московская область

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКОГО АЛГОРИТМА ДЛЯ ИМИТАЦИИ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА В ИГРЕ

Аннотация: *в данной статье рассмотрен пример использования нейронной сети для создания интеллектуального бота для игры 2048. Использована модифицированная версия генетического алгоритма, подобраны оптимальные параметры для достижения высоких результатов. Показано, что написанная программа может достигать лучших результатов, нежели среднестатистический игрок.*

Ключевые слова: *интеллектуальный бот, игры, нейронные сети, генетический алгоритм, машинное обучение.*

В последнее время широкую популярность приобрели задачи искусственного интеллекта в играх (см., например, [1]). Причем нередко возникают ситуации, когда обучение с учителем – наиболее распространенная модель – неприменимо по тем или иным причинам: игра ведётся пошагово, причем нельзя однозначно сказать, был ли конкретный шаг «правильным» или нет. Также, во многих играх есть элемент случайности, в связи с которым бессмысленно тренировать модель на фиксированных входных данных. И наконец, элемент «пошаговости» превращает попытки выиграть в перебор по пространству возможных ходов, которое может являться большим деревом.

В данной работе рассматривается случай, объединяющий в себе все вышеописанные проблемы, а также методика, позволяющая успешно обучить ИИ в такой ситуации.

В качестве модели была выбрана модификация известной компьютерной игры 2048. Опишем кратко правила данной игры.

Имеется поле размера 4x4, каждая клетка которого либо пуста, либо содержит плитки со значением $2^N, N > 0$. В начале игры поле содержит 2 плитки номинала «2» или «4» каждая. Будем называть *ходом* нажатие одной из стрелок, после которого все плитки сдвигаются в соответствии с указанным направлением. Если при сдвиге две плитки одного номинала «налетают» друг на друга, они слипаются в одну с номиналом, равным сумме номиналов и к суммарному счёту прибавляется это значение. После каждого хода в свободном месте поля появляется плитка с номиналом 2 (вероятность 90.9%) или с номиналом 4 (вероятность 9.1%). Игра заканчивается, если игрок не может сделать ход. В оригинальной версии игра выиграна, если игрок получает плитку 2048, в нашей модификации – игра продолжается до полного проигрыша.

Данная игра, с точки зрения исследования, хороша тем, что содержит относительно малое количество параметров (16 – текущие номиналы плиток, текущий счёт не рассматривался как параметр) и всего 4 параметра, влияющих на очередной ход – вероятность хода в одном из четырёх направлений.

В качестве архитектуры модели выступает нейронная сеть с одним скрытым слоем. Количество нейронов в каждом слое определялось параметрами модели: входной слой имел столько нейронов, сколько параметров имела модель и выходной – сколько требовалось для определения хода. Скрытый слой содержал фиксированное количество параметров. В качестве передаточной функции был выбран гиперболический тангенс.

Итоговая нейронная сеть работала по следующей схеме:

1. Получили вектор размерности N (количество параметров модели).

2. Подействовали на него матрицей, определяющей переход с входного слоя на скрытый.

3. Подействовали гиперболическим тангенсом на полученный вектор.

4. Повторили пункты 2–3 для последующего слоя.

5. Получили M параметров, определяющих следующий ход модели.

Обычно для обучения нейронной сети применяют алгоритм обратного распространения ошибки (см [3]), но он работает только в задаче обучения с учителем. В нашей же ситуации, мы не можем указать нейронной сети на её ошибки, так как в данной игре нет понятия «правильный» ход, поэтому back-propagation не применим.

В связи с этим было решено использовать генетический алгоритм, описанный, например, в [2; 6] – эвристический алгоритм поиска, используемый в задачах оптимизации (например, [4]). Грубо говоря, это «умный» перебор всех допустимых значений параметров с целью получения максимального положительного результата.

Основа генетического алгоритма (см [5]) – возможность формализовать задачу таким образом, чтобы её решение могло быть закодировано генотипом – вектором генов. Генами в данной работе выступали параметры нейронной сети (значения элементов матриц).

Принцип работы алгоритма состоит в следующем: задаётся функция приспособленности (fitness function), которая определяет, насколько «хорош» данный генотип, а также берётся набор генотипов с произвольными начальными данными – популяция. Далее в цикле, пока не будет достигнут оптимальный результат последовательно производятся следующие операции:

1. Размножение (скрещивание) – генерация новых генотипов на основе старых. Берётся два (обычно) генотипа-родителя, на их основе генерируется новый генотип – ребёнок и добавляется к исходному набору.

2. Мутирование – произвольные биты (значения) некоторых генотипов изменяются на некоторую величину с небольшой вероятностью.

3. Вычисляется приспособленность каждого генотипа.

4. Селекция – формирование нового поколения на основе приспособленности генотипов старого поколения.

Для обучения бота игре в 2048 использовалась нейронная сеть с размером входного слоя в 16 параметров, скрытого – 5 параметров и выходного – 4.

В качестве значений выступали соответственно матрицы 16*5 и 5*4, что в сумме даёт 100 параметров архитектуры. Один из вариантов кодирования каждого параметра – использование чисел с плавающей запятой. Однако, большинство языков программирования работают с числами размера 32/64 бита, что является лишними накладными расходами. Также, для применения генетического алгоритма нужно уметь оперировать с битами, а формат чисел с плавающей запятой подразумевает, что различные биты отвечают совершенно за разные свойства числа (мантисса, экспонента, знак). В связи с этим было решено протестировать программу на собственном формате чисел произвольной битности – UniformFloat<N>.

Данный формат хранит число с плавающей запятой в знаковом типе размера N бит, а его значение – это значение знакового типа, деленное на 2^{N-1} – максимально допустимое значение данного типа. В итоге, UniformFloat<N> может хранить 2^N различных рациональных чисел, равномерно распределенных в интервале от –1 до 1 (нормировка на единицу по умолчанию). Отметим, что данный формат хорошо показал себя в тестах для некоторых значений N, что будет показано ниже.

В дальнейшем, будем называть *агентом* генотип и нейронную сеть, ему соответствующую, а набор агентов – *поколением* (или популяцией).

Итоговый геном имел $100 * N$ бит данных. Реализация генетического алгоритма была упрощена для ускорения его работы. А именно: размножение и селекция были объединены в одну стадию отсеивания – после просчёта приспособленности один наихудший геном заменялся на копию наилучшего. Мутирование представляло собой смену значения одного произвольного бита с некоторой вероятностью. В качестве функции приспособленности выступает средний результат, который набрал агент за одну стадию тестирования.

Тестирование проходило следующим образом.

Каждый агент играл фиксированное количество игр, а средний результат по играм являлся показателем успешности. Усреднение требовалось для более грамотной оценки и для предотвращения «случайных» успехов.

Перед очередным ходом на вход нейронной сети подавалось текущее состояние поля, на выходе получались 4 числа – «вес» хода в каждую сторону.

В соответствии с максимальным весом производилась попытка походить. Если ход невозможен, брался следующий по величине вес и повторялись те же действия.

Для каждого генотипа игра заканчивалась в том случае, если он не может походить или он превысил искусственное ограничение на количество шагов, необходимое для более плавного обучения: агенты учились сначала на небольших по длительности играх, затем на более сложных. Заметим, что ограничение зависит от номера поколения, и, как правило, растет вместе с ним.

После того, как всё поколение отыграло некоторое количество игр, производилось отсеивание и мутация.

Данный алгоритм повторялся фиксированное количество раз (от 5000 до 20000), после чего оценивалась эффективность обучения.

Итоговая реализация тестировалась при размере популяции в 100 особей, различных размерах скрытого слоя сети, а также вероятности мутации одного случайного бита каждого агента на каждой итерации. Кроме того, были рассмотрены следующие типы для кодирования значений нейронной сети: UniformFloat<8>, UniformFloat<16> и float (32 bit). Дополнительно было проведено тестирование для случая линейной зависимости ограничения количества шагов от номера популяции и для зависимости вида квадратного корня.

Также, для сравнения с игрой реальных людей был разработан игровой модуль для записи статистики и добавлены результаты двух игроков.

В качестве статистики популяции на каждой итерации собиралась следующая информация:

– успех лучшего агента по популяции;

– средний успех по популяции;

– максимально возможный успех при заданном ограничении количества шагов.

Перейдем к результатам тестирования. Напомним, что с ростом популяции росло количество допустимых шагов, поэтому на ранних стадиях жизни почти все агенты не успевали проиграть за фиксированное количество шагов.

Также, результаты игроков *Player 1*, *Player 2* начиная с некоторой популяции становятся константой – так как они проиграли за фиксированное количество шагов и свой счёт улучшить с ростом популяции не могут (их результат статичен).

В дальнейшем, под *возрастом* популяции будет подразумеваться количество прошедших селекций и отборов генетического алгоритма.

На рисунке 1 представлены результаты тестирования для некоторых параметров. Из графика можно сделать вывод, что поставленная в работе цель достигнута – «бот» успешно обошёл среднестатистических игроков в 1.5 раза, причем результат его обучения вышел на некую асимптотику.

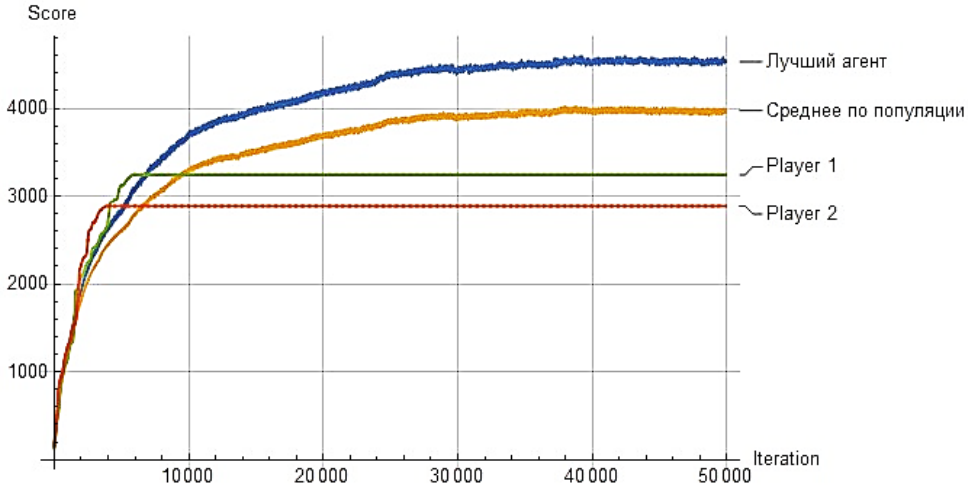

Рис. 1. Зависимость успеха от возраста популяции.
Использование UniformFloat<8>, вероятность мутации 0.03

Если проанализировать дальнейшие графики, будет видно, что при корректировке параметров алгоритма можно получить более высокие результаты, однако это выходит за рамки данной работы.

Для проверки эффективности использования введенного типа UniformFloat<N> проводилось сравнение результатов тестирования для указанного типа и стандартного float. Вопреки ожиданиям, использование типа float показало себя не хуже введенного нами, что отражено на рисунке 2. Несмотря на то, что смена отдельных бит в float может привести к бесконечности или NaN, можно предположить, что популяция быстро отсеивала такие результаты.

Тем не менее, введенный тип имеет в 4 раза меньший размер, чем стандартный float, что делает его значительно более эффективным по памяти при достаточно хороших результатах игры.

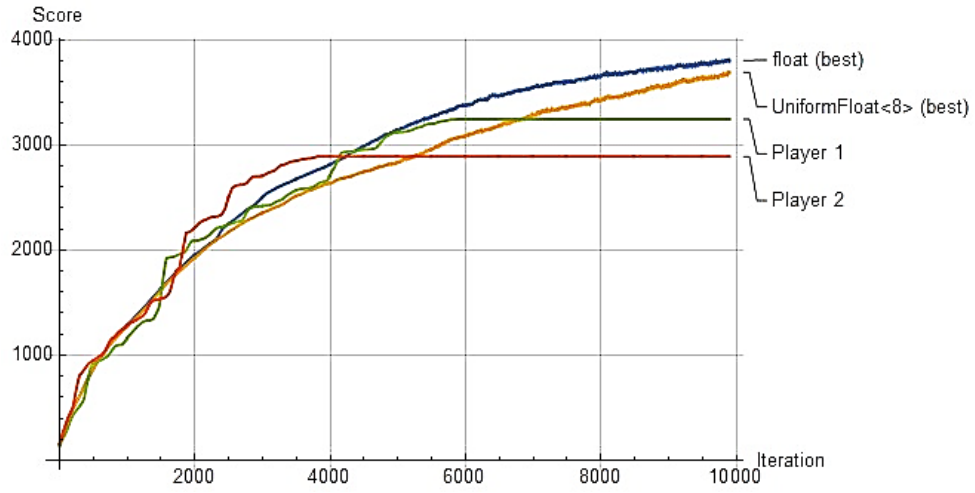

Рис. 2. Зависимость успеха от возраста популяции.
Сравнение результатов float и UniformFloat<8>
при вероятности мутации 0.03

Результаты сравнения UniformFloat<8> и UniformFloat<16> представлены на рисунке 3. 8-мибитный тип оказался более успешен, однако, вероятно, его 16-тибитному аналогу просто не хватило времени, чтобы мутировать. Также, асимптотически, расширенная версия имеет большую вариативность и теоретически может показать более высокие результаты при более длительном обучении.

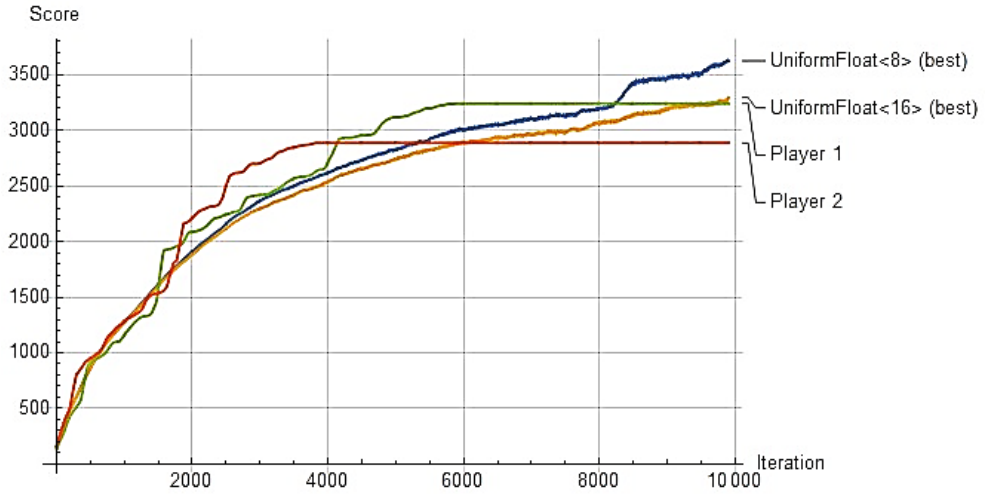

Рис. 3. Зависимость успеха от возраста популяции.
Сравнение результатов UniformFloat<8> и UniformFloat<16>
с вероятностью мутации 0.01

Отметим также, что при увеличении вероятности мутации скорость обучения возрастает, что отражено на рисунке 4. Однако, при выборе больших значений обучение может стать крайне нестабильным, что делает более разумным выбор небольших значений.

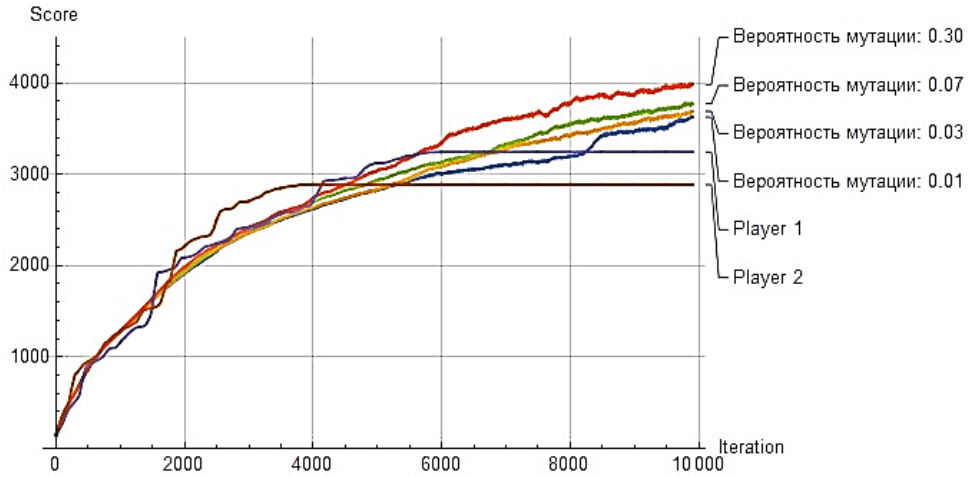

Рис. 4. Зависимость успеха от возраста популяции.
Сравнение результатов UniformFloat<8>
с различной вероятностью мутации

Наконец, проводилось сравнение результатов для разных размеров скрытого слоя сети. По результатам, представленным на рисунке 5 можно сделать вывод, что сеть большего размера учится дольше, но в теории она позже выйдет на асимптотику, так как имеет большую вариативность.

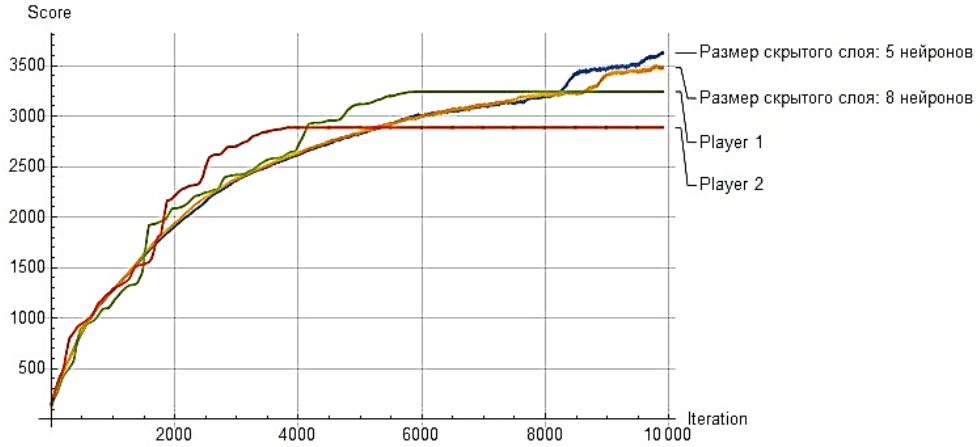

Рис. 5. Зависимость успеха от возраста популяции
для разных размеров скрытого слоя.
Вероятность мутации – 0.01

В заключение, отметим, что написанный нами интеллектуальный бот показывает высокие результаты, превосходящие среднестатистических игроков. Из полученных данных вытекает, что при достаточно длительном обучении и подборе параметров алгоритма можно добиться высокой эффективности программы.

Список литературы

1. Buckland M. Ai Techniques for Game Programmers / M.Buckland. – Boston: Premier Press, 2002.

2. Goldberg D. The Design of Innovation: Lessons from and for Competent Genetic Algorithms / D. Goldberg. – Norwell, MA: Kluwer Academic Publishers, 2002.

3. Rumelhart D.E. Learning representations by back-propagating errors / D.E. Rumelhart, G.E. Hinton, R.J. Williams // Nature. – 1986. – №323 (6088). – P. 533–536.

4. Schmitt L.M. Theory of Genetic Algorithms II: models for genetic operators over the string-tensor representation of populations and convergence to global optima

for arbitrary fitness function under scaling / L.M. Schmitt // Theoretical Computer Science. – 2004. – №310. – P. 181–231.

5. Генетический алгоритм // Wikipedia URL: https://ru.wikipedia.org/wiki/Генетический_алгоритм (дата обращения: 25.08.2017).

6. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы- 2-е изд. – М: Горячая линия-Телеком, 2008. – С. 452.