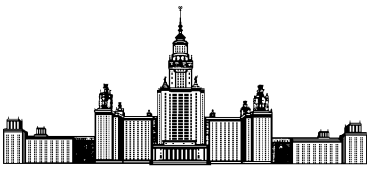
Московский государственный университет имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики
Кафедра Суперкомпьютеров и Квантовой Информатики

КУРСОВАЯ РАБОТА

Исследование применимости алгоритма дифференциальной эволюции для процесса обучения нейронной сети

Выполнил:
студент 3 курса 323 группы
Бобко Никита Александрович

Научный руководитель:
доцент, кандидат физ.-мат. наук
Попова Нина Николаевна

Москва, 2019

Содержание

1 Введение

Нейронные сети являются очень популярным инструментом анализа данных в последнее время, особенно, когда речь заходит о задачах классификации изображений, анализа человеческой речи и ее распознавания. Это те проблемы, в которых нам бы хотелось, чтобы компьютеры показывали поведение, похожее на человеческое. Нейронные сети, а особенно сверточные, хорошо себя показали в решении такого рода проблем.

Поэтому эффективное обучение нейронных сетей в наше время считается актуальной проблемой, которая обычно сопровождается сложными вычислениями. Алгоритм дифференциальной эволюции зарекомендовал себя как эффективный эвристический алгоритм поиска глобального минимума.

Цель настоящей работы исследовать потенциал алгоритма дифференциальной эволюции для процесса обучения нейронных сетей.

2 Обзор существующих работ

В этом разделе приведем обзор других работ, которые ведутся в данном направлении. В частности можно взглянуть на работу Uber AI labs [1], на основе кодовой базы которой выполнена текущая работа.

В своей работе Uber AI labs исследует потенциал обучения нейронной сети, которая учится играть в Atari Frostbite и также пытается обучить 3-х мерную модель гуманоида ходить. Для процесса обучения нейронной сети используются различные эволюционные алгоритмы. Результатом своей работы Uber AI labs представила реализацию 4-х различных эволюционных алгоритмов для решения описанных задач. Забегая вперед, скажем, что нам «удалось» получить результат лучше, чем у двух алгоритмов представленных в [1].

В другой существующей работе [5] рассматривается возможность построения сверточных нейронных сетей с помощью алгоритма дифференциальной эволюции. Авторы ставят перед собой цель исследовать потенциал алгоритма дифференциальной эволюции для автоматического построения и улучшения структуры и параметров глубоких сверточных сетей. Результатом этой работы стал гибридный алгоритм дифференциальной эволюции, который позволяет решить поставленную задачу.

3 Постановка задачи

Постановка задачи формулируется следующим образом: разработать алгоритм дифференциальной эволюции и попробовать обучить нейронную сеть, так чтобы она набирала как можно большее количество очков в игре Atari Frostbite.

3.1 Правила игры Atari Frostbite

Нижние 2/3 экрана заполнены водой с четырьмя рядами ледяных глыб, плывущих в горизонтальном направлении. Игрок делает свои ходы, прыгая с одного ряда на другой и

пытается избежать различных противников в виде крабов и птиц. Если игрок упадет в воду или встретит противника, то потеряет одну единицу жизни. Всего игроку изначально дается 4 единицы жизни. При достижении счетчиком жизни нуля, игра заканчивается поражением и начинается заново.

В верхней части экрана находится побережье, где игрок должен построить иглу. Каждый раз, когда игрок прыгает на глыбу, ряд, в котором находится глыба, меняет цвет с белого на голубой и иглу строится на +1 ледяной блок. После того, как игрок пропрыгал все ряды глыб на экране (напоминаем в игре всего 4 ряда ледяных глыб), то все глыбы обратно меняют цвет с голубого на белый и игрок может начать заново прыгать по ним.

После того, как все 15 ледяных блоков, необходимых для построения иглу, собраны, игрок должен вернуться обратно на побережье и войти в иглу, это позволит игроку перейти на следующий уровень. На каждом уровне враги и ледяные блоки двигаются быстрее чем на предыдущем уровне, тем самым усложняя игру.

Каждый из уровней обязан быть закончен не более чем за 45 секунд, иначе игрок теряет одну единицу жизни. Чем быстрее закончен уровень, тем больше дополнительных очков начисляется игроку.

Цель игры: набрать как можно большее количество очков, до того как персонаж умрет.

4 Метод решения задачи

В качестве метода решения выбран алгоритм дифференциальной эволюции. Этот алгоритм принадлежит к классу стохастических алгоритмов минимизации произвольной функции, не требующий знания градиента этой функции, что делает этот алгоритм очень удобным для применения в ряде задач.

4.1 Алгоритм дифференциальной эволюции

Алгоритм дифференциальной эволюции был разработан Рэйнером Сторном и Кеннетом Прайсом, впервые опубликован ими в 1995 году [2]. Из интересного также можно отметить, что этому алгоритму удалось занять 3-е место в «First International Contest on Evolutionary Computation» (1stICEO), который проходил в Нагое, мае 1996 [3].

Формально, пусть имеем $f : \mathbb{R}^n \to \mathbb{R}$ – функция, которую требуется минимизировать (заметим, что максимизация может быть выполнена рассмотрением функции $h = -f$). Градиент f не извесен. Цель: найти решение m, для которого $f(m) \leq f(p)$ для любого p в области определения, что означает, что m является глобальным минимумом.

Популяцией будем называть некоторое конечное множество $M = \{ x \mid x \in \mathbb{R}^n \}$.

У алгоритма дифференциальной эволюции есть три настраиваемых параметра:
$P \in [0, 1]$ – «вероятность скрещивания»
$F \in [0, 2]$ – некоторый весовой коэффициент

$|M|$ – число элементов в популяции

Пусть $x \in \mathbb{R}^n$ один из кандидатов на решение в популяции. Тогда алгоритм дифференциальной эволюции будет выглядеть следующим образом:

- Проинициализировать все x случайными значениями в области поиска минимума f
- До тех пор пока критерий остановки не выполнен (критерием остановки могут быть различные условия, такие как: число итераций, допустимое значение функции f, ...)
    - Для каждого x из популяции выполнить:
        - Выбрать 3 случайных a, b и c из популяции, они должны быть отличны друг от друга, а также от x
        - Выбрать случайное число $R \in \{1, ..., n\}$
        - Сгенерировать вектор $y \in \mathbb{R}^n$ следующим образом: Пусть $r_i \sim U(0, 1)$ – случайная величина с равномерным распределением. Если $r_i < P \times R$ или $i = R$, то $y_i = a_i + F \times (b_i - c_i)$, иначе $y_i = x_i$
        - Если оказалось, что $f(y) < f(x)$, то заменить в популяции x на y

4.2 Архитектура нейронной сети

Игра представляет из себя 3-х мерную матрицу размерности $160 \times 210 \times 3$ (Игра имеет разрешение 160×210 и еще одно одно измерение размерностью 3 для RGB). Прежде чем подать это изображение на вход нейронной сети каждый кадр уменьшается до размеров 84×84 пикселей и конвертируется из цветного изображения в изображение в градациях серого. Дальше 4 последовательных кадра «склеиваются» и получается 3-х мерная матрица размерности $84 \times 84 \times 4$, которая уже подается на вход нейронной сети.

Архитектура нейронной сети выглядит следующим образом:

- $84 \times 84 \times 4$ входных нейрона
- Сверточный слой с 16 фильтрами, размером ядра 8×8 и шагом 4
- Сверточный слой с 32 фильтрами, размером ядра 4×4 и шагом 2
- Выравнивающий слой (flatten layer)
- Полносвязный слой с 256 выходами
- Полносвязный слой с числом выходов, равным количеству действий, которые можно делать в игре Atari Frostbite

Такая архитектура была выбрана на основе [1].

4.3 Как происходит обучение и взаимодействие алгоритма дифференциальной эволюции с нейронной сетью

Так как в алгоритме дифференциальной эволюции в общем случае идет минимизация некоторой функции $f(x)$, где $x \in \mathbb{R}^n$, то надо выбрать функцию f и установить биективное отображение между весами нейронной сети и пространством $\mathbb{R}^n$.

В качестве f выбрана функция, равная количеству очков, которое набирает нейронная сеть в игре Atari Frostbite за фиксированное время (если в течении этого времени нейронная сеть успела проиграть, то считаем то количество очков, которое она успела набрать до проигрыша). В нашем случае это число равно 5000 итераций взаимодействия с библиотекой gym (именно эта библиотека используется для эмуляции игры Atari Frostbite). Это число выбрано таким образом, чтобы быть достаточно большим, чтобы количество очков, которое можно набрать, не ограничивалось сверху временем, но и не таким большим, чтобы вычисление f занимало слишком много времени.

Таким образом, осталось только установить биективное соответствие между $\mathbb{R}^n$ и весами нейронной сети. Это делается тривиальным образом, просто методом упорядочивания весов нейронной сети. Сеть имеет $n = 1009058$ параметров для обучения.

В качестве значений параметров P, F алгоритма дифференциальной эволюции были выбраны 0.9 и 0.5 соответственно. Размер популяции варьировался в разных экспериментах

5 Вычислительный эксперимент предложенного алгоритма

Вычислительный эксперимент проводился на платформе Google Cloud. На следующих характеристиках:

Процессор: Intel(R) Xeon(R) CPU
Число физических ядер: 12
Число виртуальных ядер: 24
Частота процессора: 2 Ггц
ОЗУ: 90 ГБ

После небольшого размышления стало понятно, что нейронная сеть сможет достаточно хорошо обучиться прыгать по глыбам, избегая противников и не падая в воду. Интерес представляло то, «догадается» ли нейронная сеть зайти в иглу, чтобы перейти на следующий уровень или она продолжит прыгать по глыбам. Но так как каждый раунд в игре не может длится более 45 секунд, то это должно было стимулировать нейронную сеть «обучиться» заходить в иглу.

Понять, «обучилась» ли нейронная сеть заходить в иглу по количеству набранных ею очков, довольно просто. Если число очков больше 540, то это значит, что сети удалось «зайти» в иглу. Это связано с тем, что такое количество очков нельзя набрать просто прыгая по

глыбам, 45 секунд успеют истечь раньше, а за заход в иглу как раз дается ровно 540 очков.

Сначала вычислительный эксперимент проводился для количества рабочих равного 200. Из трех вычислительных экспериментов нейронной сети удавалось найти решение, в котором она «обучалась» заходить в иглу, в одном из трех первых поколений популяции. Была выдвинута гипотеза, что во всех этих трех экспериментах нам просто повезло. Так как рабочих 200 и один из них мог случайным образом попасть в правильное решение еще на самом старте обучения.

Таким образом после этого были проведены два эксперимента с 40 рабочими, чтобы снизить вероятность случайного попадания в «правильное» решение с самых первых поколений. В этих двух экспериментах нейронная сеть, действительно, «попала» в правильное решение не сразу, и лишь по прошествии порядка 150 поколений обе нейронные сети «догадалась» зайти в иглу.

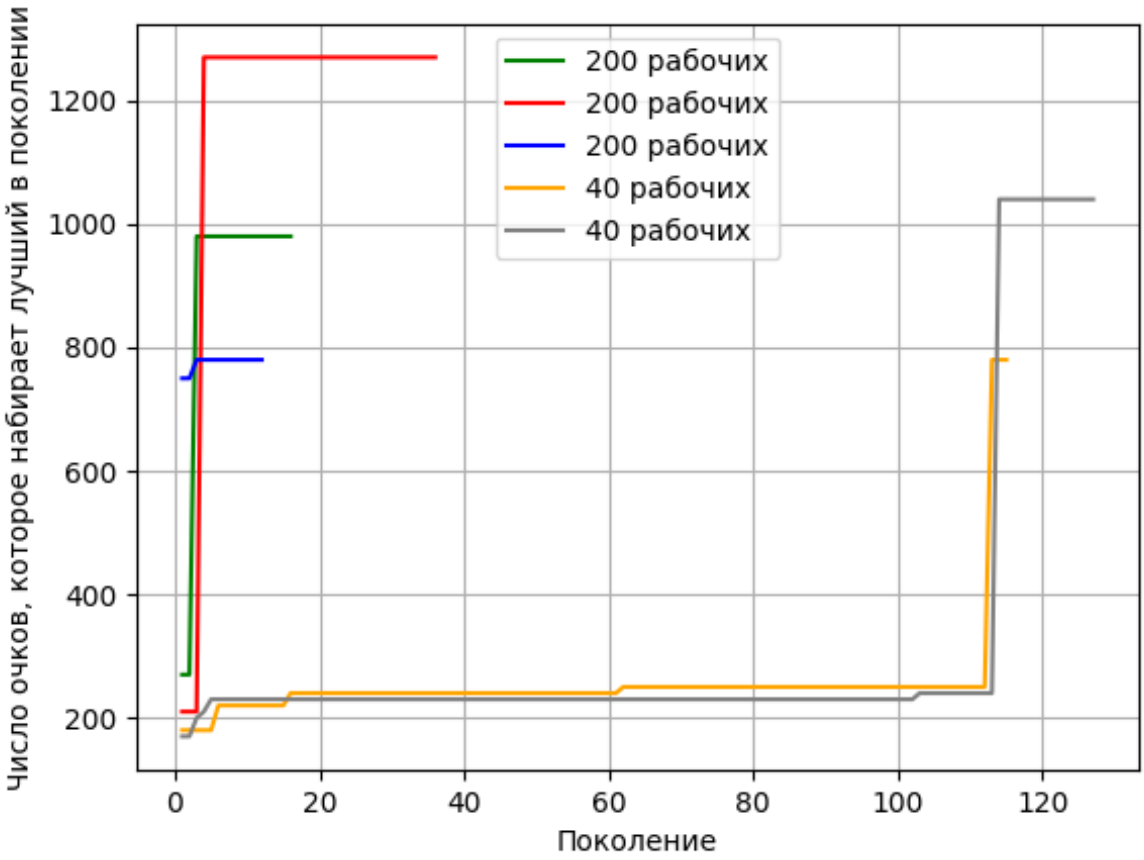


Рис. 1: Зависимость количества набираемых очков от поколения в 5-ти различных экспериментах

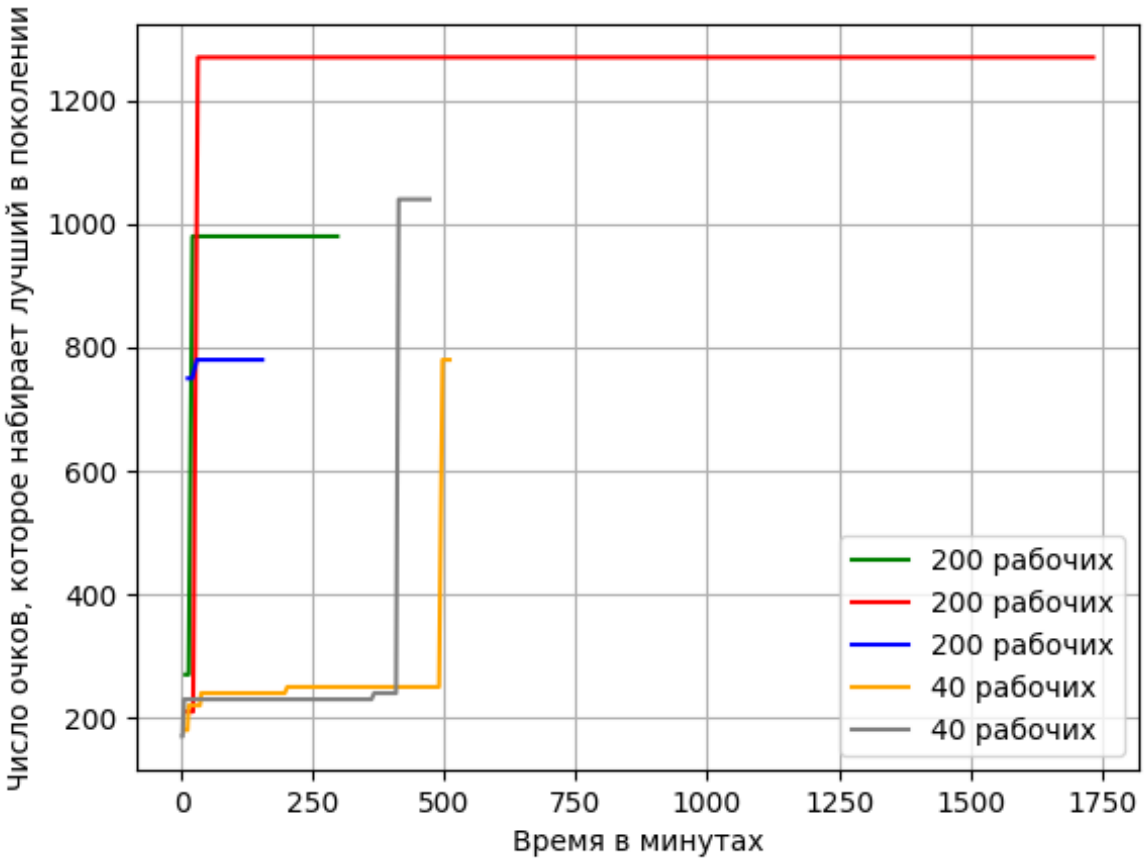


Рис. 2: Зависимость количества набираемых очков от времени в минутах в 5-ти различных экспериментах

Видео того, как работает обученная нейронная сеть, можно найти по адресу:
`https://youtu.be/VjWcr3XTsHU`

6 Описание программной реализации

Программная реализация построена на базе
`https://github.com/uber-research/deep-neuroevolution`.
Мою программную реализацию можно найти по адресу
`https://github.com/nikitabobko/deep-neuroevolution`.
Программа выполнена на языке Python 3.

Основные модули и их описание:

- `es_distributed/policies.py` содержит класс `Policy` и его наследника `ESAtariPolicy`. В этих классах реализованы политики обсчета нейронной сети и сама структура нейронной сети

- `es_distributed/dist.py` содержит два класса `CoolWorkerClient` и `CoolMasterClient`. Это классы в которых описано взаимодействие основного процесса и процессов-рабочих. Взаимодействие сделано через пересылки сообщений по сокетам по TCP соединению. (процесс-мастер запускается по адресу `127.0.0.1:9999`)

- `es_distributed/es.py` содержит две основные функции `run_master` и `run_worker`, которые выполняют процесс-мастер и процессы-рабочие соответственно.

- `scripts/viz.py` визуализирует файл сохраненный на i-ой итерации, чтобы можно было вживую увидеть то, как играет лучший индивид из поколения.

Программная реализация выглядит следующим образом: сначала запускается вышеупомянутый bash скрипт `scripts/local_run_exp.sh`. В этом скрипте можно указать количество рабочих. Далее этот скрипт запускает функцию `es_distributed.main.master`, в которой происходит fork всех дочерних процессов и поднимается сервер, через который в последствии будет идти общение мастера и рабочих.

Далее, мастер итерацию за итерацией будет выполнять `es_distributed.es.run_master`, где происходит алгоритм дифференциальной эволюции. Рабочим же распределяются задачи по подсчету количества очков, которые набирает отдельно взятый индивид из популяции.

Диаграмма UML описанной реализации выглядит следующим образом:

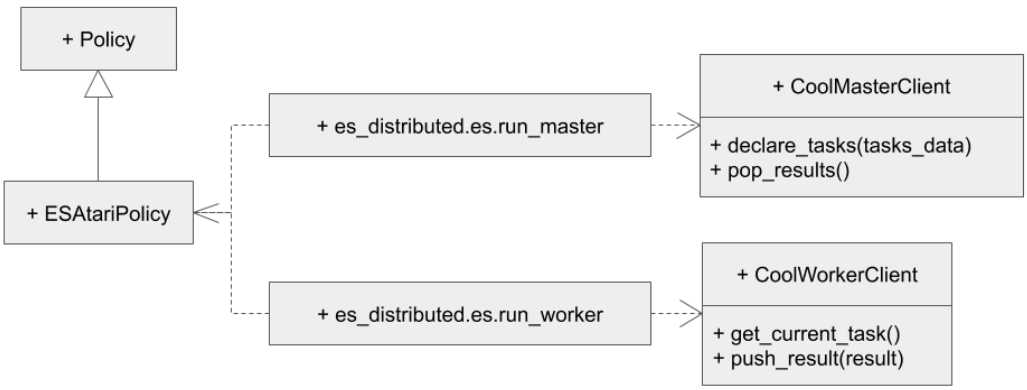


Рис. 3: UML диаграмма реализации

6.1 Установка и запуск

После того, как проект склонирован с помощью git:

```
git clone https://github.com/nikitabobko/deep-neuroevolution.git
```

Нужно создать виртуальную среду Python 3 и активировать ее:

```
python3 -m venv env
. env/bin/activate
```

Далее идет установка необходимых зависимостей:

```
pip install -r requirements.txt
```

И после этого можно запустить процесс обучения:

```
. scripts/local_run_exp.sh es configurations/frostbite_es.json
```

Чтобы запустить режим визуализации нужно выполнить:

```
python -m scripts.viz 'FrostbiteNoFrameskip-v4' <YOUR_H5_FILE>
```

7 Заключение

К сожалению, не до конца понятен потенциал алгоритма дифференциальной эволюции в применении к обучению нейронных сетей. Этот алгоритм является алгоритмом глобальной минимизации функции, и этому алгоритму действительно удалось выйти из локального минимума в экспериментах с количеством рабочих равному 40, несмотря на то, что популяция на протяжении 8.3 часов находилась в локальном минимуме.

Но удручающим является то, что в эксперименте с 200 рабочими алгоритм, проработав более 29 часов, так и не смог набрать большее количество очков, чем то, которое он набрал за первые полчаса эксперимента.

Тем не менее точно можно утверждать, что получившийся алгоритм оказался лучше алгоритмов Evolution Strategies (ES) и Novelty Search – Evolution Strategies (NS-ES) в работе [1], так как оба этих алгоритма ES и NS-ES так и не смогли «догадаться» войти в иглу.

В качестве дальнейшего направления было бы интересно исследовать эффективность обучения при различных выборах значений параметров алгоритма дифференциальной эволюции P и F. Таким образом в [3] утверждается, что выбор параметра F для каждого поколения случайным образом значительно улучшает сходимость алгоритма.

Список литературы

[1] *Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O. Stanley, Jeff Clune*. Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents

[2] *Rainer Storn, Kenneth Price*. Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces

[3] Differential Evolution (DE) for Continuous Function Optimization (an algorithm by Kenneth Price and Rainer Storn)

[4] *Magnus Erik Hvass Pedersen*. Good Parameters for Differential Evolution

[5] *Bin Wang, Yanan Sun, Bing Xue, Mengjie Zhang*. A Hybrid Differential Evolution Approach to Designing Deep Convolutional Neural Networks for Image Classification