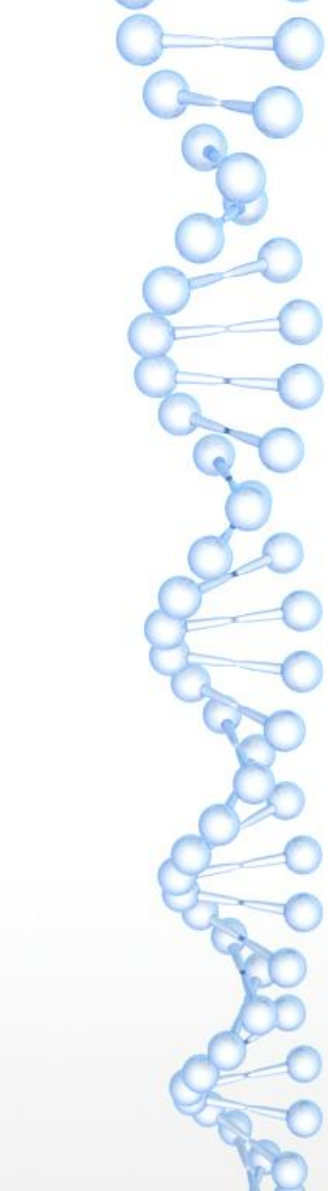
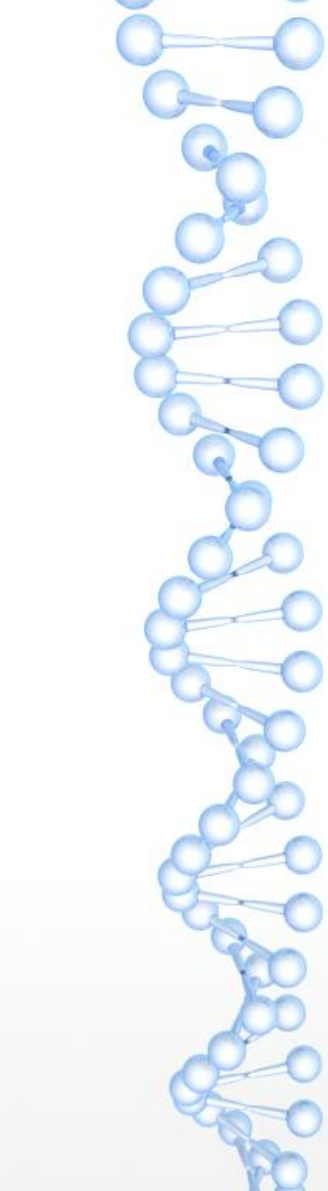


Обзорное исследование подходов к решению задачи Reinforcement Learning

Работу выполнили студенты BD-21:
Богомолов Эмиль, Петряйкин Фёдор, Серов Николай

https://github.com/zetyquickly/NN_Project-sphere

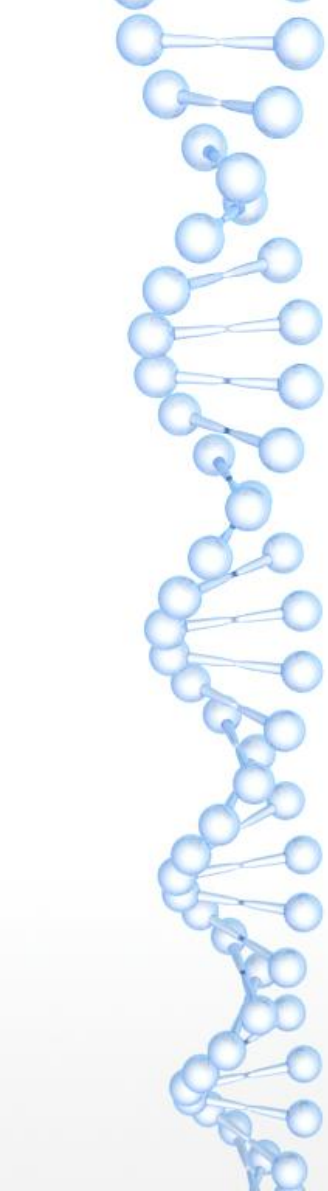


Цели работы

- Рассмотреть несколько подходов обучения с подкреплением: Q-learning, evolution strategy, genetic algorithm
- Реализовать алгоритмы с использованием нейронных сетей
- Опробовать полученные знания на примере игры из Atari

Выбор среды для экспериментов

- Статья DeepMind с их результатами
- Выбор пал на BreakOutDeterministic-v4



Обучение

- Каждая стратегия обучалась в течение 1 дня,
  1 поток CPU 3.30 ГГц, GPU GTX 960
- На вход подается разность текущего и предыдущего экрана
- Архитектура сети (нелинейность ReLU):

- Свертка 3 → 16 (kernel_size=8, stride=4)
- Свертка 16 → 32 (kernel_size=4, stride=2)
- Линейный слой 13824 → 160
- Линейный слой 160 → 4

Первый подход: Q-learning

- Базируется на уравнении Беллмана (см. лекцию 12):

$$Q^{\pi}(s,a) = r + \gamma Q^{\pi}(s', \pi(s'))$$

- Нейронная сеть минимизирует разницу
- с помощью Huber-loss:

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

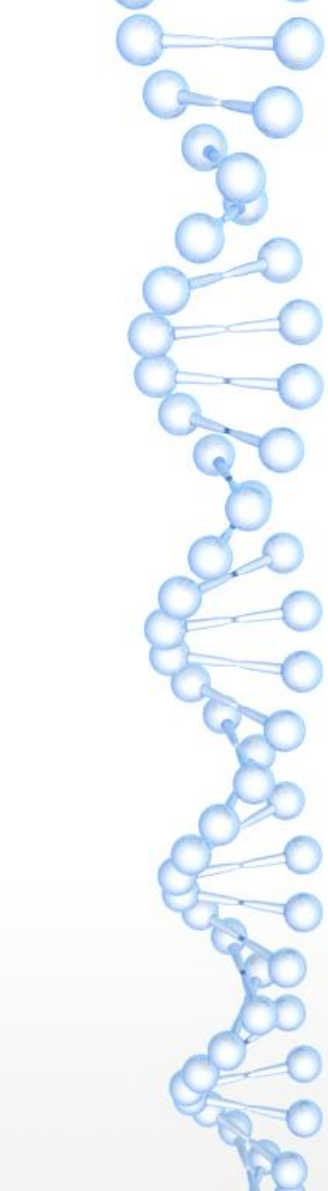
Первый подход: Q-learning

- Во время обучения заполняется deque из четвёрок

$$(s, a, r, s')$$

- Состояние – это последовательность из последних 4 кадров, приведённых к одному каналу

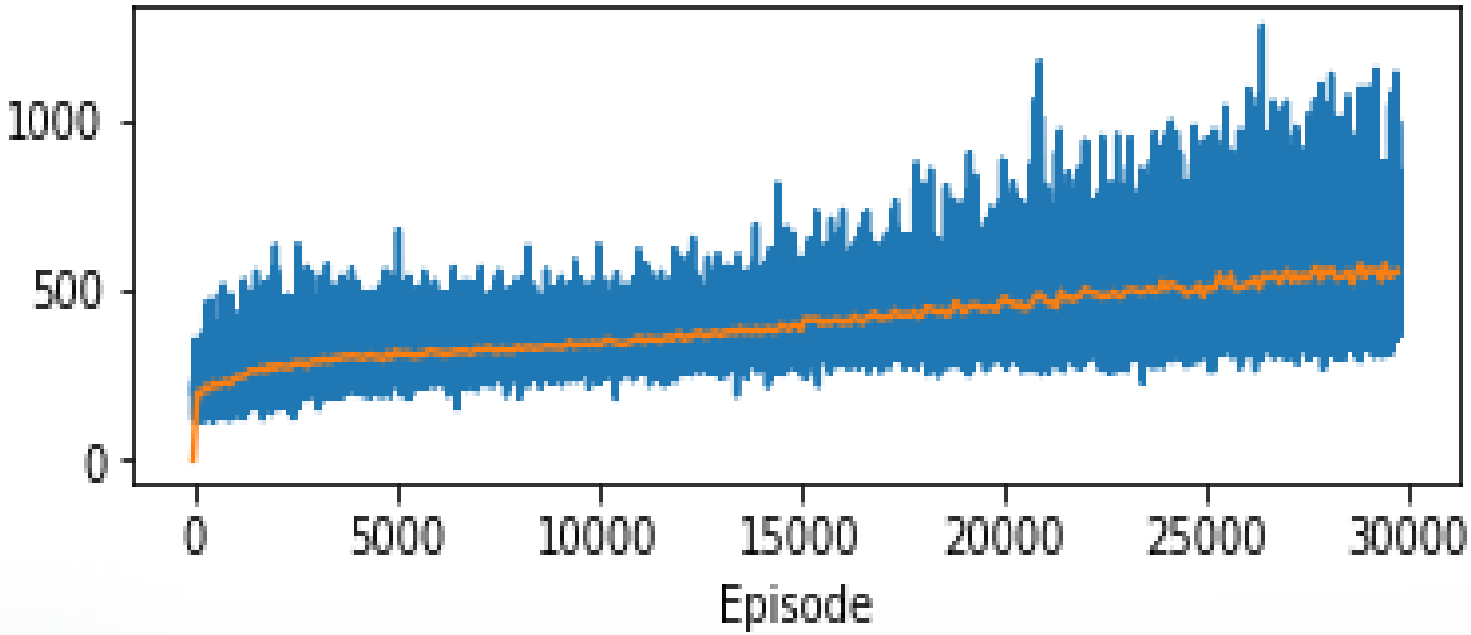
- Действие выбирается epsilon-жадной стратегией с затухающей экспоненциально случайностью



Подход второй: Q-learning

- Оптимизация Huber loss производится на семплированном нескоррелированном подмножестве памяти
- Во время обучения сосуществуют две сети одинаковой архитектуры. Одна обучается, другая обновляет веса каждые несколько шагов алгоритма

Q-learning: резульаты



Средний reward = 11

Второй подход: evolution strategy

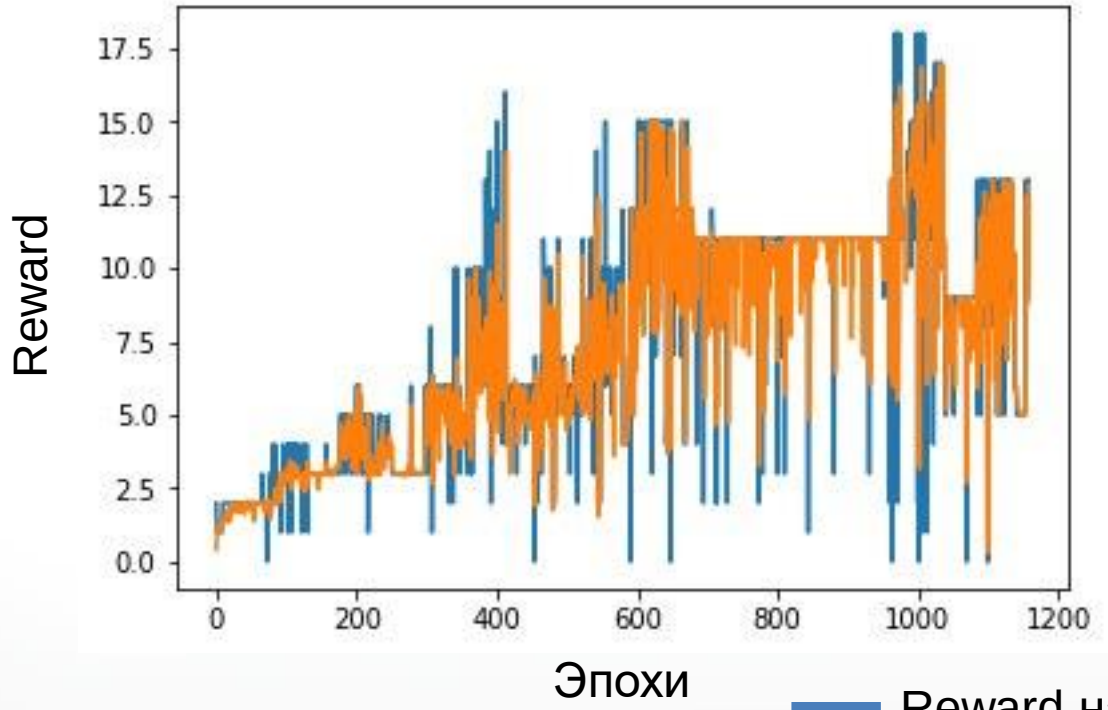
Algorithm 1 Evolution Strategies

1: **Input:** Learning rate α, noise standard deviation σ, initial policy parameters θ_0
2: **for** $t = 0, 1, 2, \ldots$ **do**
3: Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
4: Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
5: Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
6: **end for**

“Black box” алгоритм: вместо расчета градиентов в Q-Learning мы максимизируем reward. Направление движения – оценка градиента по параметрам

+: Может эффективно исполняться параллельно

Второй подход: evolution strategy



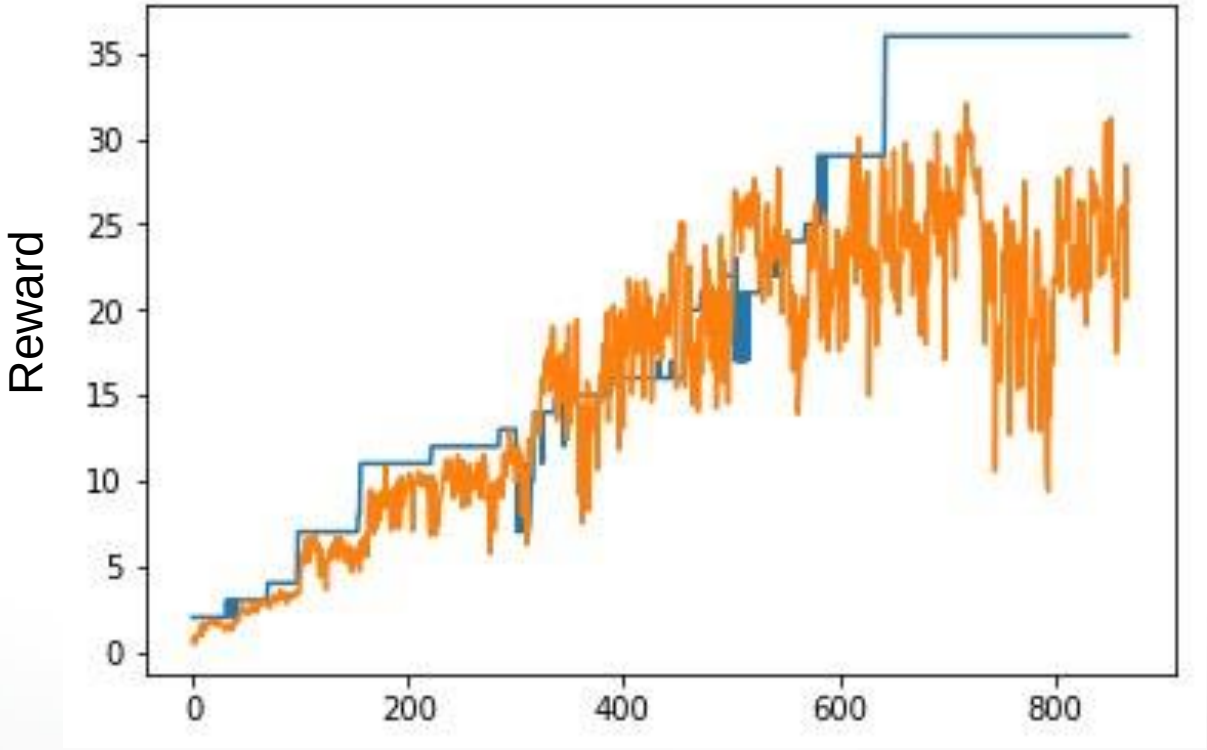
Третий подход: genetic algorithm

Algorithm 1 Simple Genetic Algorithm

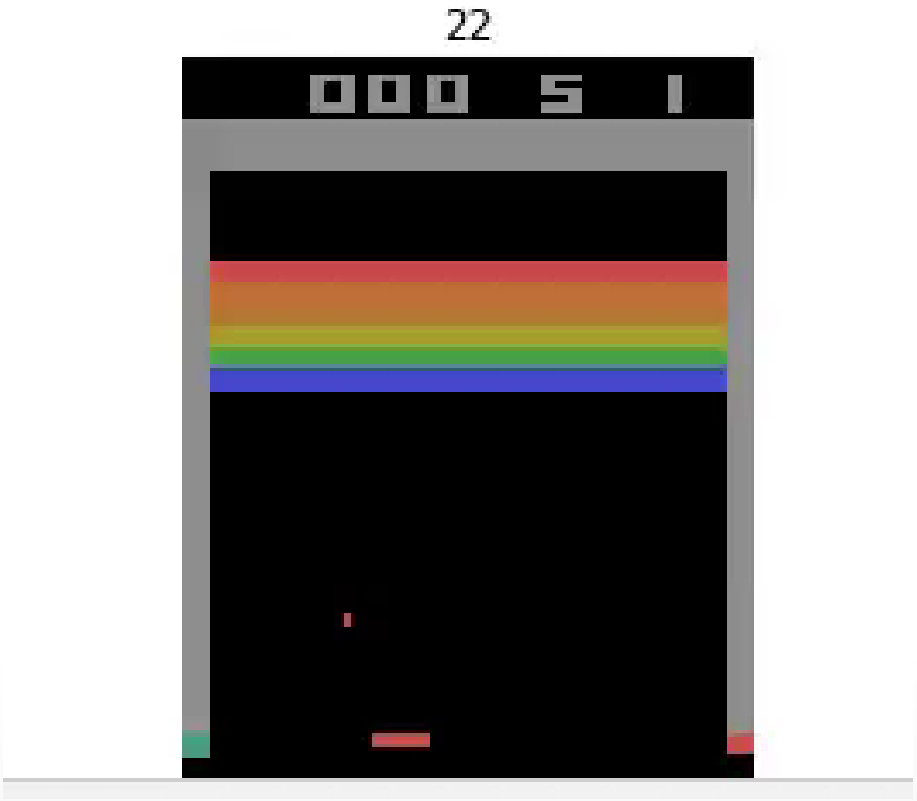
Input: mutation function ψ, population size N, number of selected individuals T, policy initialization routine ϕ, fitness function F.
for $g = 1, 2..., G$ generations **do**
  for $i = 1, ..., N-1$ in next generation's population **do**
    if $g = 1$ **then**
      $\mathcal{P}_i^{g=1} = \phi(\mathcal{N}(0, I))$ {initialize random DNN}
    else
      $k = \text{uniformRandom}(1, T)$ {select parent}
      $\mathcal{P}_i^g = \psi(\mathcal{P}_k^{g-1})$ {mutate parent}
    end if
    Evaluate $F_i = F(\mathcal{P}_i^g)$
  end for
  Sort $\mathcal{P}_i^g$ with descending order by F_i
  if $g = 1$ **then**
    Set Elite Candidates $C \leftarrow \mathcal{P}_{1...10}^{g=1}$
  else
    Set Elite Candidates $C \leftarrow \mathcal{P}_{1...9}^g \cup \{\text{Elite}\}$
  end if
  Set Elite $\leftarrow$ Сеть с макс. reward прошлого поколения
  $\mathcal{P}^g \leftarrow [\text{Elite}, \mathcal{P}^g - \{\text{Elite}\}]$ {only include elite once}
end for
Return: Elite

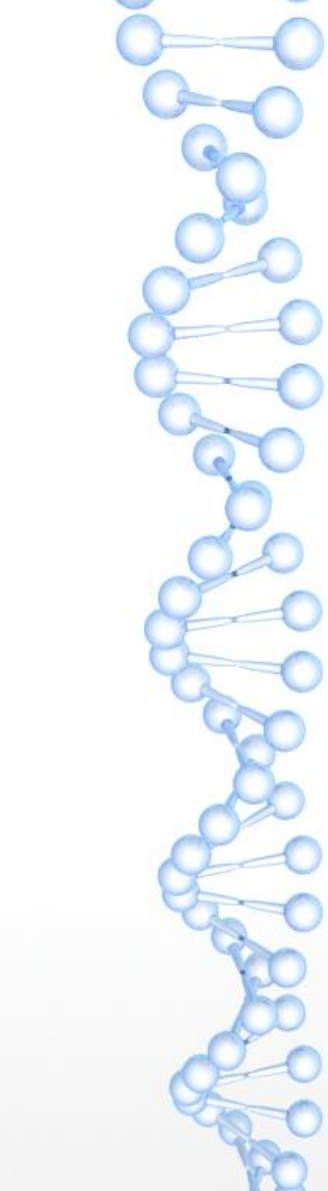
"Black box" алгоритм:
Вместо оценки градиентов мы просто заменяем популяцию наиболее перспективными претендентами

Третий подход: genetic algorithm



Третий подход: genetic algorithm





Спасибо за внимание!