МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Инженерная школа информационных технологий и робототехники
Отделение информационных технологий
Направление: 09.04.01 Искусственный интеллект и машинное обучение

ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

по дисциплине: Нейроэволюционные вычисления

Вариант 14

на тему: Реализация алгоритма нейроэволюции H-ESP для непрерывного контроля среды Lunar Lander

Выполнил:	студент гр. 8ВМ42 Мабах А.С.	09.06.2025
Проверил:	к.т.н., Доцент ОИТ ИШИТР Григорьев Д.С.	09.06.2025

Томск – 2025

Содержание

1 Введение

Цель работы — реализовать полный цикл нейроэволюционного обучения с помощью алгоритма H-ESP для задачи управления агентом в среде `LunarLander-v3`, соблюдая следующие требования:

- Разработка модульного и воспроизводимого кода без использования сторонних реализаций H-ESP;
- Детальная визуализация структуры сети и динамики обучения на каждом этапе;
- Обоснование и анализ используемых целевых метрик, обеспечивающих объективную оценку успешности обучения.

2 Описание используемого алгоритма

2.1 Принципы работы H-ESP (Hierarchical Enforced SubPopulations)

Алгоритм H-ESP (Hierarchical Enforced SubPopulations), предложенный Фаустино Гомесом как вариант развития ESP.Нововведением является использование эволюции уровня сетей (L1), а не только нейронов (L2).

2.2 Структура сети

В работе используется рекуррентная нейронная сеть (RNN), которая обрабатывает последовательные данные с сохранением внутреннего состояния подходящая под алгоритм H-ESP:

- **Входной слой:** Полносвязный слой (Linear), преобразующий входные данные в скрытое пространство.
- **Рекуррентный слой:** Обрабатывает предыдущее скрытое состояние.
- **Выходной слой:** Преобразует скрытое состояние в выходные данны

2.3 Этапы алгоритма H-ESP

Алгоритм H-ESP состоит из следующих этапов:

1. **Инициализация**
   - Задаётся число скрытых нейронов h.
   - Формируется N случайных сетей с h скрытыми нейронами.
2. **Оценка приспособленности (Evaluation)**
   - Уровень нейронов. Если полученная при оценивании сеть лучше, чем худшая сеть из L2, то она добавляется в L2.
   - Уровень нейронных сетей. Формируется NN случайных сетей с hh скрытыми нейронами.
     - Определяется приспособленность еще не оцененных сетей.
     - Если приспособленность сети лучше, чем приспособленность лучшей ИНС (искусственной нейронной сети) из L1, то нейроны из этой сети добавляются в подпопуляции.
3. **Проверка вырождения популяции**
   - Если лучшая приспособленность не улучшается на протяжении b поколений, применяется взрывная мутация (“burst mutation”): подпопуляции перегенерируются вблизи своих лучших особей с помощью распределения Коши.

- Если и после двух взрывных мутаций улучшения нет, применяется адаптация структуры сети — изменение количества подпопуляций/нейронов.

4. **Скрещивание (Crossover) и отбор (Selection)**

   - Уровень нейронов
     - Для каждой подпопуляции рассчитывается средний фитнес каждой особи (суммарный фитнес делится на число испытаний).
     - Особей сортируют по убыванию приспособленности; лишние особи (выходящие за пределы размера популяции) удаляются.
     - Лучшие особи (обычно 1/4) скрещиваются между собой (одноточечный кроссовер), потомки добавляются в конец подпопуляции.
     - Для нижней половины популяции применяется мутация с распределением Коши.
   - Уровень нейронных сетей. Каждая ИНС скрещивается с более приспособленной с использованием h-точечного кроссинговера («понейронно»). Потомки мутируют.

5. **Повторение**

   - Шаги 2–4 повторяются до выполнения критерия остановки (например, достижение целевого качества или максимального числа эпох).

3 Этапы имплементации

Реализация алгоритма H-ESP для задачи управления в среде `LunarLanders-v3` была выполнена на языке Python с использованием только стандартных научных библиотек. (`numpy`, `gymnasium`) и полностью авторской логики без сторонних реализаций нейроэволюции. Оригинальный алгоритм был несколько переработан для оптимизации под конкретную задачу.

3.1 Модульная структура кода

Код организован модульно, что облегчает повторное использование и дальнейшее расширение:

- **Модуль** `Model` — реализует основные эволюционные операции и логику подпопуляций.
- **Модуль** `Network` — отвечает за архитектуру сети.
- **Вспомогательные модули** — визуализация, сохранение/загрузка весов, утилиты командной строки.
- **Главный исполняемый файл** — обеспечивает запуск обучения, тестирования и визуализации через аргументы командной строки.

3.2 Основные этапы реализации

Инициализация

В оригинальном алгоритме инициализация состоит в создании случайных сетей и их добавлении в подпопуляции L2, если они оказываются более приспособленными. Также в процессе инициализации определяются признаки для каждой сети. В данной реализации же инициализация происходит через создание случайных сетей для уровня L1, каждая из которых оценивается в среде и добавляется в популяцию. Здесь создается только один уровень, в отличие от оригинала, где можно увидеть более сложную структуру с уровнями L1 и L2, с добавлением нейронов в зависимости от их приспособленности.
Реализация подпопуляции в коде:

```
    self.L1 = []

        for _ in range(L1_size):
            net = ReccurentNetwork(self.input_dim, self.output_dim, hidden_units)
            self.evaluate_network(net)
            self.L1.append(net)
```

Оценка приспособленности

В оригинальном алгоритме оценка приспособленности заключается в расчетах на основе симуляции сети и различных пробежек по поколениям. Каждый нейрон оценивается на основе своих попыток при определении приспособленности, что является более теоретическим процессом. В коде же оценка приспособленности происходит через вычисление общего вознаграждения на нескольких эпизодах в игре. Каждая сеть, проходя через среду, получает свою оценку на основе общего вознаграждения, что делает этот процесс более специфичным и привязанным к задаче, в отличие от теоретического подхода, который используется в оригинальном алгоритме.
Каждая сеть проходит оценку через функцию evaluate_network, где вычисляется её приспособленность на основе суммы вознаграждений за несколько эпизодов.
Реализация оценки приспособленности в коде:

```
    def evaluate_network(self, network, episodes=5, render=False):
        total_reward = 0
        for _ in range(episodes):
            state, _ = self.env.reset()
            network.reset_states()
            done = False
            while not done:
                action = np.argmax(network.predict(state))
                state, reward, terminated, truncated, _ =
                    self.env.step(action)
                done = terminated or truncated
                total_reward += reward

        fitness = total_reward / episodes
        network.fitness = fitness
```

Рекомбинация подпопуляций

В оригинальном алгоритме рекомбинация происходит внутри подпопуляций нейронов с использованием их признаков и приспособленности для создания новых сетей. В коде этот процесс происходит через выбор двух лучших сетей из L1 для скрещивания, с последующим использованием кроссинговера и мутации. Кроссинговер, как и в оригинале, происходит между сетями, но в коде это делает случайный выбор двух сетей, что влияет на разнообразие в процессе. В оригинале акцент больше сделан на подпопуляции, где каждая группа имеет свои особенности, а в коде это более примитивный выбор между лучшими особями. Рекомбинация происходит через выбор двух лучших сетей из L1, которые затем проходят процесс кроссинговера и мутации.
Реализация рекомбинации:

```
    def recombine(self):
        new_pop = []
        sorted_L1 = sorted(self.L1, key=lambda x: x.fitness, reverse=True)
        topk = sorted_L1[:max(2, len(sorted_L1)//3)]

        while len(new_pop) < self.population_size:
            p1 = random.choice(topk)
            p2 = random.choice(topk)
            if p2 is p1:
                continue
            child = p1.crossover(p2)

            child.mutate(L2_pool=self.L2)

            new_pop.append(child)
        return new_pop
```

Обновление уровня сетей

В оригинальном алгоритме обновление уровня сетей включает добавление новых сетей в L2, а также их оценку и возможную замену худших. Это происходит на основе приспособленности, когда нейроны из сетей с высокой приспособленностью добавляются в L2, что влияет на будущие поколения. В коде же обновление происходит через добавление нейронов в L2 с учетом их приспособленности и замены худших нейронов, если появляется новый более подходящий нейрон. Здесь больше внимания уделяется замене нейронов в списке L2, чем в оригинальном алгоритме, где добавление новых сетей в подпопуляции более структурировано.
Обновление уровня сетей происходит через добавление лучших нейронов в L2, если их приспособленность выше определённого уровня, и через замену худших нейронов.
Реализация обновления сетей:

Реализация операций:

```
    for i in range(len(self.networks)):
        if self.network_fitness[i] == -float('inf'):
            self.network_fitness[i] = self.evaluate_network(self.networks[i],
                env)

        best_net_idx = np.argmax(self.network_fitness)
        best_net = self.networks[best_net_idx]
        for i in range(self.hidden_size):
            neuron_weights = torch.cat([best_net.fc1.weight[i],
                best_net.fc1.bias[i], best_net.recurrent.weight[i]])
```

```
            worst_idx = np.argmin(self.subpop_fitness[i])
            self.subpopulations[i][worst_idx] = neuron_weights
```

Рекомбинация сетей

В оригинальном алгоритме рекомбинация сетей основана на слиянии сетей с лучшими признаками и их приспособленностью. Это может подразумевать более общий подход к слиянию сетей. В коде же процесс рекомбинации реализован через кроссинговер между двумя сетями, с выбором нейронов, принадлежащих к одному из родителей. Этот процесс более конкретен, так как каждый нейрон из скрытых слоев или выходных нейронов копируется случайным образом из одной из двух сетей. Это делает процесс более направленным на случайность и изменчивость в составе сетей, что отличается от абстрактного описания в оригинальном алгоритме.
В коде рекомбинация сетей осуществляется через кроссинговер между двумя сетями с использованием случайного выбора нейронов из разных сетей.
Реализаци рекомбинации сетей в коде:

```
    def crossover(self, other):
            child = ReccurentNetwork(self.input_dim, self.output_dim,
                self.hidden_units)
            for i in range(self.hidden_units):
                parent = self.hidden_neurons[i] if random.random() < 0.5 else
                    other.hidden_neurons[i]
                child.hidden_neurons[i] = parent.clone()
            for i in range(self.output_dim):
                parent = self.output_neurons[i] if random.random() < 0.5 else
                    other.output_neurons[i]
                child.output_neurons[i] = parent.clone()
            return child
```

Логика адаптации:

```
    for i in range(self.num_subpops):
        best_idx = np.argmax(self.subpop_fitness[i])
        best_neuron = self.subpopulations[i][best_idx]
        self.subpopulations[i] = [self.cauchy_mutation(best_neuron,
            self.mutation_scale)
                                  for _ in range(self.subpop_size)]
```

4 Целевые метрики

4.1 Формальное определение метрики

Основной целевой метрикой является **среднее суммарное вознаграждение за эпизод** (average episodic reward), вычисляемое как:

$$R_{\text{avg}} = \frac{1}{N} \sum_{i=1}^{N} R_i$$

где:

- N — количество эпизодов оценки
- R_i — суммарное вознаграждение за i-й эпизод

4.2 Расчёт вознаграждения в LunarLanderContinuous-v3

В среде `LunarLander-v3` вознаграждение формируется по сложной формуле, учитывающей физические параметры посадки:

$$R = R_{\text{position}} + R_{\text{velocity}} + R_{\text{angle}} + R_{\text{contact}} + R_{\text{landing}} + R_{\text{fuel}} + R_{\text{time}}$$

Компоненты вознаграждения:

1. **Позиция** (R_{position}):
$$-100\sqrt{(x - x_{\text{target}})^2 + (y - y_{\text{target}})^2}$$
Штраф за удаление от целевой зоны посадки

2. **Скорость** (R_{velocity}):
$$-100\,(|v_x| + |v_y|)$$
Штраф за высокую горизонтальную (v_x) и вертикальную (v_y) скорость

3. **Угол наклона** (R_{angle}):
$$-100|\theta|$$
Штраф за отклонение от вертикального положения (θ — угол в радианах)

4. **Контакт с поверхностью** (R_{contact}):
$$+10 \cdot (\text{leg1\_contact} + \text{leg2\_contact})$$
Награда за касание посадочными опорами

5. **Успешная посадка** (R_{landing}):
$$\begin{cases} +200 & \text{если } v_y > -1 \text{ м/с и } |\theta| < 0.2 \text{ рад} \\ -100 & \text{в противном случае} \end{cases}$$

6. **Расход топлива** (R_{fuel}):

$$-0.3 \cdot (\text{main\_engine} + 0.03 \cdot \text{side\_engine})$$

Штраф за использование основного и боковых двигателей

7. **Временной штраф** (R_{time}):

$$-0.3 \cdot t$$

Штраф за каждый шаг симуляции (t)

4.3 Реализация в системе обучения

В системе обучения метрика рассчитывается следующим образом:

```
best = max(n.fitness for n in self.L1)
avg = np.mean([n.fitness for n in self.L1])

fitness = total_reward / episodes
network.fitness = fitness
```

4.4 Интерпретация метрики

- **Успешная посадка**: $R_{\text{avg}} \geq 200$
- **Приемлемый результат**: $50 \leq R_{\text{avg}} < 200$
- **Неудачная посадка**: $R_{\text{avg}} < 0$
- **Рекорд среды**: $R_{\text{avg}} \approx 300$ (оптимальная посадка)

5 Визуализация

Важной частью анализа эволюционного обучения является наглядная визуализация развития структуры нейронной сети и динамики ключевых метрик. В ходе экспериментов автоматически сохранялись скриншоты архитектуры сети на различных этапах (через фиксированный интервал эпох), а также строились графики изменения целевых показателей.

5.1 Визуализация структуры нейронной сети

На рисунках ниже представлены архитектуры реккурентной нейронной сети с одним скрытым слоем, эволюционирующей с помощью алгоритма H-ESP для задачи управления посадкой в среде `LunarLander-v3`. Входными данными для сети служат восемь параметров состояния среды (позиция, скорость, угол, контакт с поверхностью и др.), скрытый слой состоит из 12 нейронов, выходной слой — из 4 нейрона, отвечающих за одно из возможных действий.

Каждая визуализация строится по "лучшей" сети — она собирается из лучших особей каждой подпопуляции на данной эпохе.

На каждом изображении (рис. **??**):

- **Цвет и толщина связей** отражают знак и величину весового коэффициента:
  - Зеленые линии — положительные веса (активирующие связи)
  - Красные линии — отрицательные веса (ингибирующие связи)
  - Пунктрирные линии - реккурентная связь
  - Толщина линии пропорциональна абсолютной величине веса $|w|$
- **Нормализация**: Для устранения влияния выбросов толщина и насыщенность масштабируются по 95-му перцентилю модулей весов
- **Узлы сети**:
  - Серые — входные нейроны
  - Синие — реккурентный слой
  - Оранжевый — выходные нейроны

  Пример визуализации структуры сети:

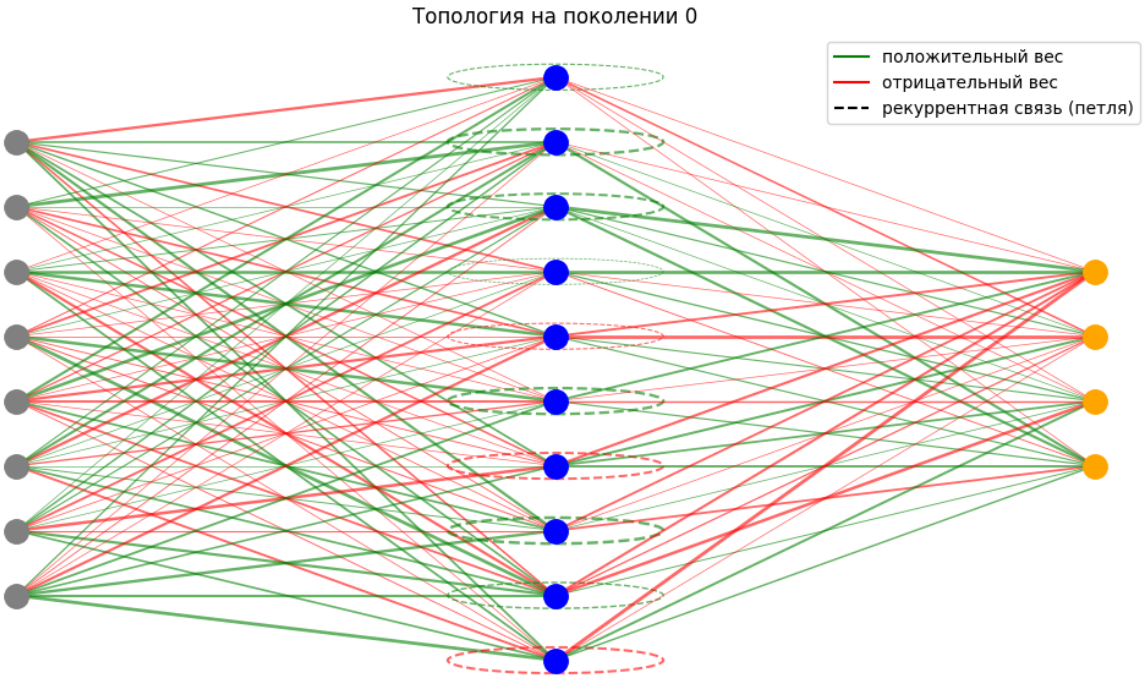

Рис. 1: Визуализизация структуры сети после инициализации

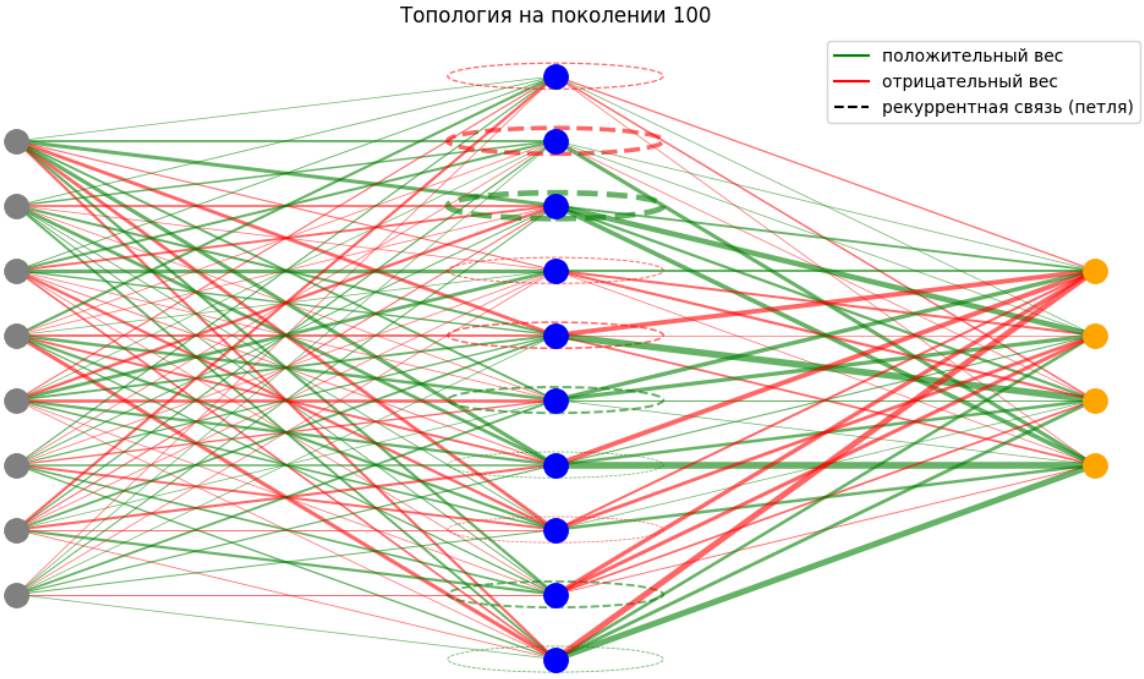

Рис. 2: Визуализизация структуры сети на 100 поколении

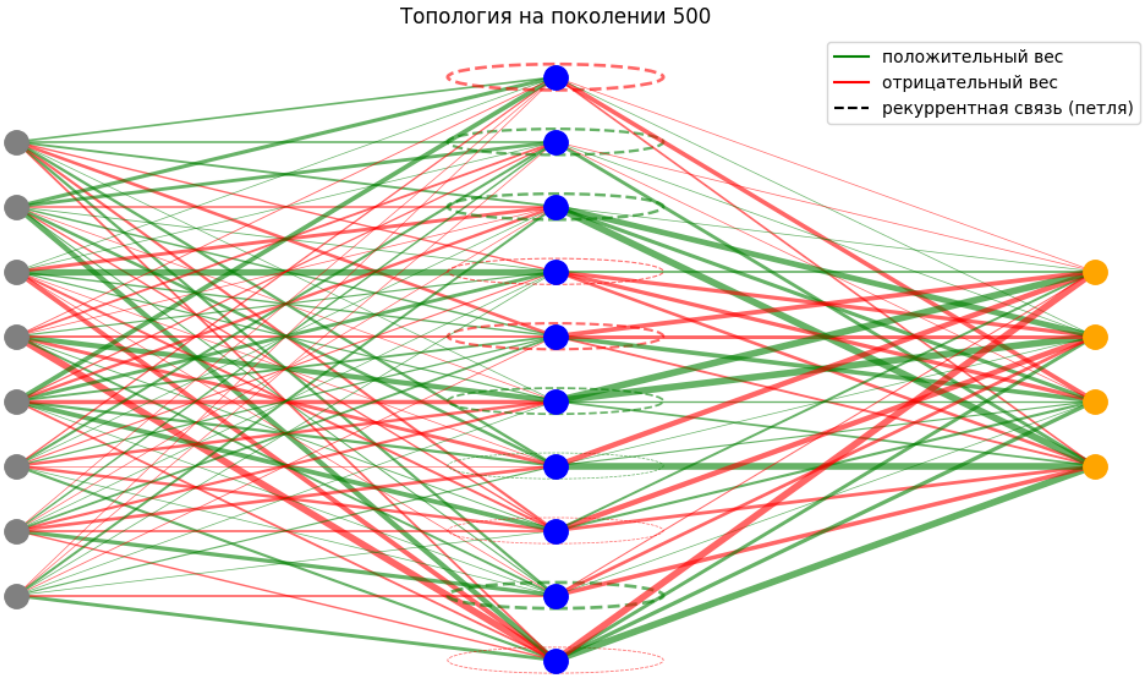

Рис. 3: Визуализизация структуры сетина 500 поколении

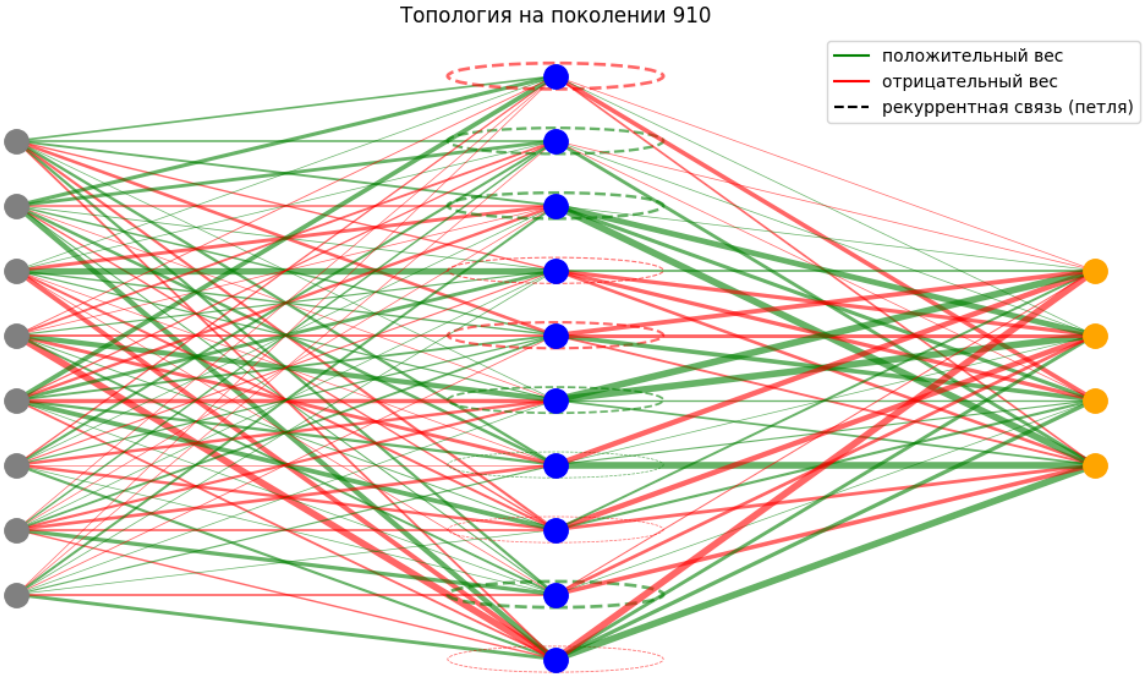

Рис. 4: Визуализизация структуры сети на 910 поколении

5.2 Графики сходимости метрик

Для анализа динамики обучения строились график среднего суммарного вознаграждения.

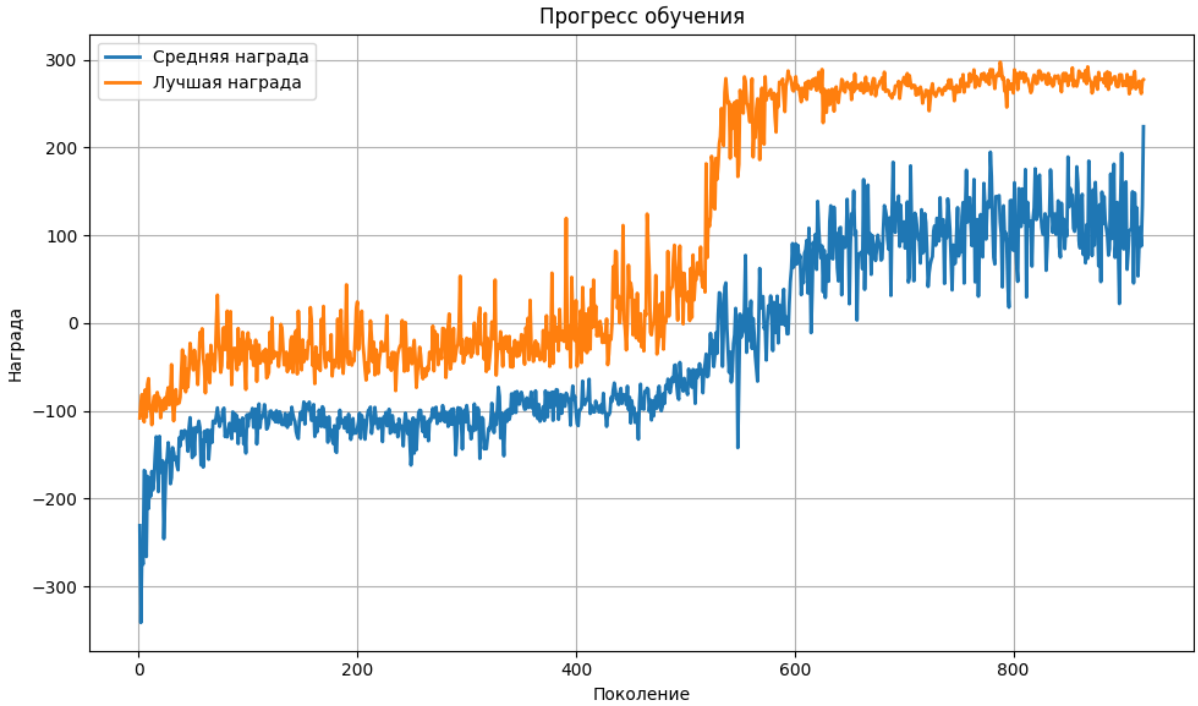

Рис. 5: График среднего суммарного вознаграждения и лучшего суммарного вознаграждения

На графике видно, как средняя награда вышла из ямы случайных значений после примерно 50 эпохи. Затем с 100 эпохи наступило плато, которое длилось до 400 эпохи. После этого пошел резкий рост и примерно с 550 эпохи отдельные особи начали решать задачу. С 600 эпохи пошел медленный рост с прорывными поколениями и на 919 эпохе среднее занчение всей популяции превысило значение в 200 еденицы, что говорит о решение задачи и завершение обучения.

6 Развертывание, тестирование и анализ результатов

6.1 Структура проекта

Проект организован в виде набора Python-модулей и вспомогательных файлов. Основные компоненты:

- `network.py` — реализация нейронной сети прямого распространения:
  - Класс `RecurrentNetwork` с методами прямой передачи сигнала
  - Функции инициализации весов
  - Операции сохранения/загрузки параметров сети
- `model.py` — ядро алгоритма эволюции стратегий:
  - Класс `HESP`
  - Процедура оценки фитнеса

    * Эволюционный цикл
- `utils.py` — вспомогательные утилиты:
    * Визуализация сети
    * Построение графика средней награды
    * Запуск программы с сохраненными весами
- `train.py` — основной исполняемый скрипт:
    * Конфигурация параметров обучения
    * Интеграция с средой `Gymnasium`
    * Управление процессом обучения/тестирования

6.2 Анализ результатов тестирования

Динамика обучения: Сравнение с начальным этапом обучения показывает радикальное улучшение показателей:

- **Начальные значения**: -300
- **Финальные значения**: 220
- **Прогресс**: Улучшение на 520 единиц вознаграждения

Выводы:

1. Алгоритм успешно решает задачу, так как удовлетворяет заданному критерию в 200 еденицы награды
2. Динамика обучения – график показывает устойчивый рост награды с возможными колебаниями, что характерно для эволюционных методов
3. Лучшие особи – максимальная награда в популяции значительно выше 200, что говорит о наличии высокоэффективных решений.

Алгоритм Hierarchical ESP (H-ESP) успешно обучился решению поставленной задачи, стабильно достигая целевого уровня награды 200. Это подтверждает его эффективность в оптимизации нейронных сетей для заданной среды.

7 Заключение

В ходе лабораторной работы задача управления посадочным модулем в среде LunarLander
v2 успешно решалась с использованием измененного алгоритма Hierarchical ESP (H-ESP) в сочетании с рекуррентной нейронной сетью, содержащей один скрытый слой. Начав обучение со средним вознаграждением около -300 пунктов, что соответствовало полному отсутствию управляемой стратегии, система продемонстрировала устойчивую динамику улучшения показателей. К завершению обучения на 919-й эпохе было достигнуто среднее вознаграждение 230 пунктов, что свидетельствует о формировании эффективной стратегии посадки.

Применение H-ESP в сочетании с рекуррентной архитектурой сети позволило агенту не только освоить базовые навыки управления, но и выработать последовательную стратегию, учитывающую временные зависимости в процессе посадки. Особенно важно отметить, что рекуррентная структура с одним скрытым слоем оказалась достаточной для решения данной задачи, демонстрируя баланс между вычислительной эффективностью и способностью к обучению сложным паттернам управления. Полученные результаты подтверждают эффективность комбинации эволюционных методов с рекуррентными архитектурами для задач непрерывного управления, где требуется учет временных зависимостей в поведении агента.

Список использованной литературы

1. Лекция 7. Алгоритмы ESP и H-ESP. Томский политехнический университет, 2025.