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 3

на тему: Реализация алгоритма ESP для непрерывного контроля в среде Lunar Lander

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

Томск – 2025

Содержание

1 Цель работы

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

- Разработка модульного и воспроизводимого кода без использования сторонних реализаций NE/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 ESP (Enforced SubPopulations)

Алгоритм ESP (Enforced SubPopulations), предложенный Фаустино Гомесом[1], относится к классу коэволюционных алгоритмов эволюции весов искусственных нейронных сетей (ИНС). В отличие от классических эволюционных подходов, где целиком эволюционируют параметры всей сети, в ESP отдельные подпопуляции отвечают за оптимизацию весов каждого нейрона скрытого слоя.

Основные черты ESP:

- **Использование вещественного кодирования:** каждый генотип содержит веса всех входных и выходных связей своего нейрона.
- **Коэволюция:** оптимизация происходит параллельно для каждой подпопуляции, что способствует специализации нейронов (разделение функций, решаемых отдельными нейронами, за счет их индивидуального обучения).
- **Формирование команды:** для каждой попытки (trial) формируется сеть из случайно выбранных представителей разных подпопуляций; таким образом, особи оцениваются в различных "командах", что снижает вероятность локального экстремума.

2.2 Структура сети (input, hidden, output, только прямое распространение)

В работе используется однослойная рекуррентная нейронная сеть:

- **Входной слой:** размерность равна количеству признаков среды (для LunarLanderContinuo v3 — 8 признаков).
- **Скрытый слой:** количество нейронов фиксируется (в данной реализации — 10); каждому нейрону соответствует своя подпопуляция.

- **Выходной слой:** размерность равна количеству управляющих воздействий (4 выхода для управления тягами двух боковых и главного двигателей, а также возможность бездействия).

2.3 Логика разбиения на подпопуляции, эволюция на уровне нейрона

- Каждому нейрону скрытого слоя соответствует своя подпопуляция (подмножество особей), каждая особь — это вектор параметров (веса входных, скрытых ивыходных связей).
- Для оценки пригодности особей формируются команды — сети, собранные из случайно выбранных представителей всех подпопуляций.
- Оценка нейрона (особи) происходит кумулятивно: его фитнес — сумма результатов всех команд, в которых он участвовал (каждая особь должна быть использована не менее заданного числа раз, например, 10).
- Благодаря раздельной эволюции нейронов достигается специализация — нейроны постепенно начинают решать разные подзадачи управления.

2.4 Этапы алгоритма ESP

Алгоритм ESP состоит из следующих этапов (см. алгоритм 7.1 из лекции):

1. **Инициализация**
   - Задаётся число скрытых нейронов h.
   - Для каждого нейрона создаётся своя подпопуляция из n особей. Параметры нейронов случайно инициализируются.
2. **Оценка приспособленности (Evaluation)**
   - Каждая особь-нейрон многократно участвует в командах, где формируется полная сеть из случайных особей разных подпопуляций.
   - Приспособленность (fitness), полученная в ходе каждого испытания, где нейрон был задействован суммирутся.
   - Оценивание проводится, пока каждый нейрон не будет использован в, как минимум, 10 попытках.
3. **Проверка вырождения популяции**
   - Если лучшая приспособленность не улучшается на протяжении b поколений, применяется взрывная мутация (“burst mutation”, алгоритм 7.2): подпопуляции перегенерируются вблизи своих лучших особей с помощью распределения Коши.

- Если и после двух взрывных мутаций улучшения нет, происходит изменение структуры сети (алгоритм 7.3).

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

   - Для каждой подпопуляции рассчитывается средний фитнес каждой особи (суммарный фитнес делится на число испытаний).
   - Особей сортируют по убыванию приспособленности; особи, выходящие за пределы размера популяции, удаляются.
   - Скрещиваются 1/4 лучших особей (одноточечный кроссовер), потомки добавляются в конец подпопуляции.
   - Для нижней половины популяции применяется мутация с распределением Коши.

5. **Повторение**

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

3 Этапы имплементации

Реализация алгоритма ESP для задачи управления в среде `LunarLanderContinuous-v3` была выполнена на языке Python без использования сторонних реализаций алгоритма нейроэволюции.

3.1 Модульная структура кода

- `esp` — реализует логику алгоритма;
- `network` — отвечает за архитектуру сети;
- `utils и visualization` — визуализация, сохранение/загрузка модели;
- `esplander` — обеспечивает запуск обучения, тестирования и визуализации через аргументы командной строки.

3.2 Основные этапы реализации

Инициализация подпопуляций и параметров сети. На первом этапе создаются независимые подпопуляции для каждого скрытого нейрона. Каждая подпопуляция содержит n особей, где каждая особь представляет вектор весов:

$$\vec{w}_i = [w_{i1}^{\text{in}}, \ldots, w_{iK}^{\text{in}}, w_{i1}^{\text{hiden}}, \ldots, w_{iT}^{\text{hiden}}, w_{i1}^{\text{out}}, \ldots, w_{iM}^{\text{out}}]$$

где K – размер входного слоя, T – размер скрытого слоя, M – размер выходного слоя. Веса задаются случайными значениями.

Реализация в коде:

```
    self.subpopulations = [
            [self._random_individual() for _ in range(subpop_size)]
            for _ in range(hidden_size)
    ]
```

Оценка приспособленности. Для оценки каждого индивида в подпопуляциях используется механизм испытаний:

```
    def evaluate(self, env: gym.Env, n_episodes: int = 1, render: bool = False):
        for i in range(self.hidden_size):
            self.cum_fitness[i].fill(0.0)
            self.count_trials[i].fill(0)

        for i in range(self.hidden_size):
            for j in range(self.subpop_size):
                for t in range(self.trials_per_individual):
```

```
                hidden_indices = []
                for k in range(self.hidden_size):
                    hidden_indices.append(j if k == i else np.random.randint(0,
                        self.subpop_size))
                network = self.assemble_network(hidden_indices)
                total_rewards = []
                for ep in range(n_episodes):
                    obs, _ = env.reset()
                    done = False
                    episode_reward = 0.0
                    hidden_state = None
                    while not done:
                        action, hidden_state = network.forward(obs, hidden_state)
                        obs, reward, terminated, truncated, _ = env.step(action)
                        episode_reward += reward
                        done = terminated or truncated
                        if render:
                            env.render()
                    total_rewards.append(episode_reward)
                avg_reward = np.mean(total_rewards)
                self.cum_fitness[i][j] += avg_reward
                self.count_trials[i][j] += 1

        avg_fitness = []
        for i in range(self.hidden_size):
            avg = self.cum_fitness[i] / np.maximum(self.count_trials[i], 1)
            avg_fitness.append(avg)
        return avg_fitness
```

Эволюционные операции. После оценки выполняется цикл эволюционных операций для каждой подпопуляции:

1. Селекция и кроссовер (алгоритм 7.1):

- Сортировка особей по убыванию fitness
- Выбор top-25% (лучшая четверть) как родительский пул
- Парное скрещивание с вероятностью $P_{\text{cross}} = 0.5$
- Одноточечный кроссовер
- Замена худших особей потомками

2. Мутации:

- *Коши-мутация* для нижней половины популяции:

$$\vec{w} \leftarrow \vec{w} + \alpha \cdot \text{Cauchy}(0, 1)$$

Реализация операций:

```
    def select_and_breed(self, avg_fitness: list[np.ndarray]):
        for i in range(self.hidden_size):
            subpop = self.subpopulations[i]
            fitness_i = avg_fitness[i]
            sorted_idxs = np.argsort(-fitness_i)
            subpop_sorted = [subpop[idx].copy() for idx in sorted_idxs]
            top_k = max(1, self.subpop_size // 4)
            parents = subpop_sorted[:top_k]
            children = []
            for idx in range(0, top_k - 1, 2):
                if np.random.rand() < self.crossover_rate:
                    a, b = parents[idx], parents[idx + 1]
                    point = np.random.randint(1, len(a))
                    children.append(np.concatenate([a[:point], b[point:]]))
                    children.append(np.concatenate([b[:point], a[point:]]))
                else:
                    children.append(parents[idx].copy())
                    children.append(parents[idx + 1].copy())
            if top_k % 2 == 1:
                children.append(parents[-1].copy())
            m = len(children)
            keep_count = max(0, self.subpop_size - m)
            retained = subpop_sorted[:keep_count]
            subpop_new = retained + children
            half = self.subpop_size // 2
            for idx in range(half, self.subpop_size):
                perturb = self.alpha_cauchy *
                    np.random.standard_cauchy(size=subpop_new[idx].shape)
                subpop_new[idx] += perturb
            for idx in range(self.subpop_size):
                if np.random.rand() < self.mutation_rate:
                    subpop_new[idx] += np.random.randn(*subpop_new[idx].shape) *
                        0.01
            self.subpopulations[i] = subpop_new
```

Адаптация структуры сети. При застое в обучении активируются специальные механизмы:

Burst-мутация (алгоритм 7.2):

- Для каждой подпопуляции выбирается лучшая особь;
- Вся подпопуляция заменяется на копии лучшей особи + возмущения.

Логика мутации:

```
    def burst_mutation(self):
        for i in range(self.hidden_size):
            avg_i = self.cum_fitness[i] / np.maximum(self.count_trials[i], 1)
            best_idx = int(np.argmax(avg_i))
            best_vector = self.subpopulations[i][best_idx]
            new_subpop = []
            for _ in range(self.subpop_size):
                perturb = self.alpha_cauchy *
                    np.random.standard_cauchy(size=best_vector.shape)
                new_subpop.append(best_vector + perturb)
            self.subpopulations[i] = new_subpop
        for i in range(self.hidden_size):
            self.cum_fitness[i].fill(0.0)
            self.count_trials[i].fill(0)
```

Адаптация архитектуры (алгоритм 7.3):

- Последовательная проверка нейронов на значимость
- Если удаление нейрона улучшает fitness - он удаляется
- Если ни один нейрон не удалён - добавляется новый нейрон

Логика адаптации:

```
    def adapt_structure(self, env: gym.Env, n_episodes: int = 1):
        removed_any = True
        while removed_any:
            removed_any = False
            current_best = self._compute_global_best_fitness()
            old_subpops = [list(sp) for sp in self.subpopulations]
            old_hidden = self.hidden_size
            for i in range(old_hidden):
                tmp_pops = [old_subpops[k] for k in range(old_hidden) if k != i]
                tmp_hidden = old_hidden - 1
```

```
            tmp = ESPPopulation(
                self.input_size, tmp_hidden, self.output_size,
                self.subpop_size, self.trials_per_individual,
                self.alpha_cauchy, self.stagnation_b,
                self.mutation_rate, self.crossover_rate
            )
            tmp.subpopulations = [[ind.copy() for ind in sp] for sp in
                tmp_pops]
            best_tmp = tmp._compute_global_best_fitness_from_avg(
                tmp.evaluate(env, n_episodes=n_episodes)
            )
            if best_tmp > current_best:
                print(f"  {i}: {current_best:.3f} - {best_tmp:.3f}")
                self.subpopulations = tmp.subpopulations
                self.hidden_size = tmp_hidden
                removed_any = True
                break
    if not removed_any:
        old_h = self.hidden_size
        self.hidden_size += 1
        self.subpopulations.append([
            np.random.randn(self.input_size + self.hidden_size +
                self.output_size) * 0.1
            for _ in range(self.subpop_size)
        ])
        for i in range(old_h):
            new_subpop = []
            for vec in self.subpopulations[i]:
                ih = vec[:self.input_size]
                hh = vec[self.input_size:self.input_size+old_h]
                ho = vec[self.input_size+old_h:]

                new_hh = np.concatenate([hh, np.random.randn(1) * 0.1])
                new_ho = np.concatenate([ho,
                    np.random.randn(self.output_size) * 0.1])
                new_vec = np.concatenate([ih, new_hh, new_ho])
                new_subpop.append(new_vec)
            self.subpopulations[i] = new_subpop

    self.cum_fitness = [np.zeros(self.subpop_size) for _ in
        range(self.hidden_size)]
    self.count_trials = [np.zeros(self.subpop_size, dtype=np.int32) for _ in
        range(self.hidden_size)]
```

Визуализация структуры и динамики сети. На каждой эпохе обучения формируется визуализация топологии сети, а также графики изменения метрики по эпохам.

Сохранение и загрузка весов. Для воспроизводимости и анализа промежуточных результатов реализовано сохранение состояния сетив формате pickle. Это позволяет продолжить обучение с любого этапа или протестировать ранее обученного агента.

Запуск, тестирование и создание визуализаций. Сценарий запуска программы реализован через аргументы командной строки: можно запустить обучение (`-train`), протестировать готовую сеть (`-test`), либо создать отдельную визуализацию структуры сети или работы агента.

Пример команды запуска:

```
python main.py --train --epochs 500 --hidden_size 10 --subpop_size 10
```

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 `LunarLanderContinuous-v3` вознаграждение формируется по 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:

```
1        avg_fitness = pop.evaluate(env, n_episodes=args.episodes_per_eval,
             render=False)
2        best_fitness_current = pop._compute_global_best_fitness_from_avg(avg_fitness)
3        reward_history.append(best_fitness_current)
```

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 структуры нейронной сети

- **Цвет и толщина связей** отражают знак и величину весового коэффициента:
  - Синие линии — отрицательные веса (ингибирующие связи)
  - Красные линии — положительные веса (активирующие связи)
  - Толщина линии пропорциональна абсолютной величине веса $|w|$
- **Нормализация**: Для устранения влияния выбросов толщина и насыщенность масштабируются по 95-му перцентилю модулей весов
- **Узлы сети**:
  - Синие — входные нейроны
  - Оранжевые — скрытые нейроны
  - Зелёные — выходные нейроны
- По мере обучения наблюдается:
  - Усиление ключевых связей
  - Формирование устойчивых функциональных “путей”
  - Специализация отдельных нейронов
  - Упрощение структуры через отмирание слабых связей

  Примеры визуализации структуры сети на разных эпохах обучения:

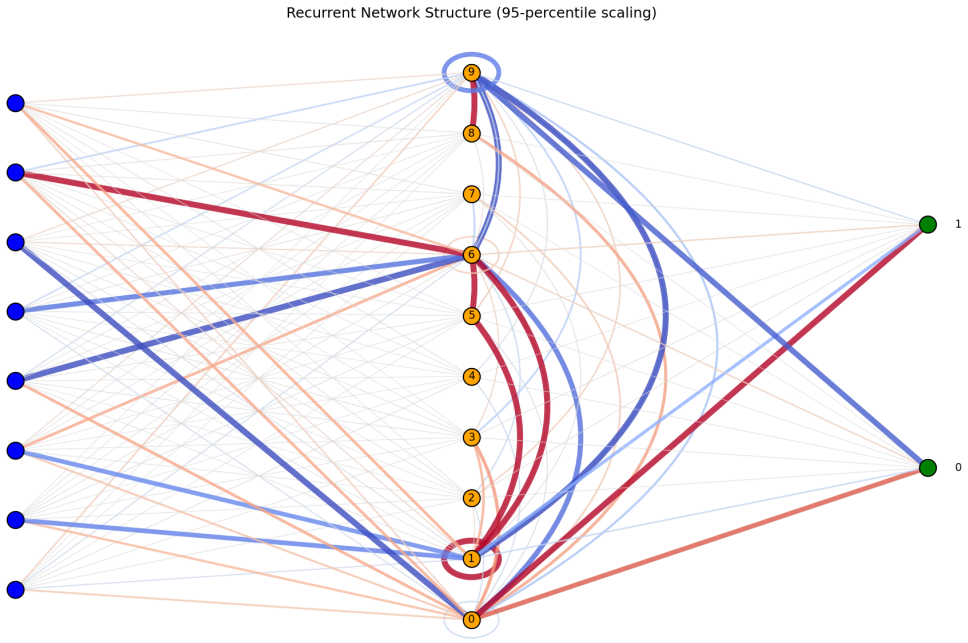

Рис. 1: Эпоха 1. Исходная структура сети: веса малы по модулю и равномерно распределены. Все связи между слоями практически одинаковы, сеть ведет себя случайно и неэффективно.

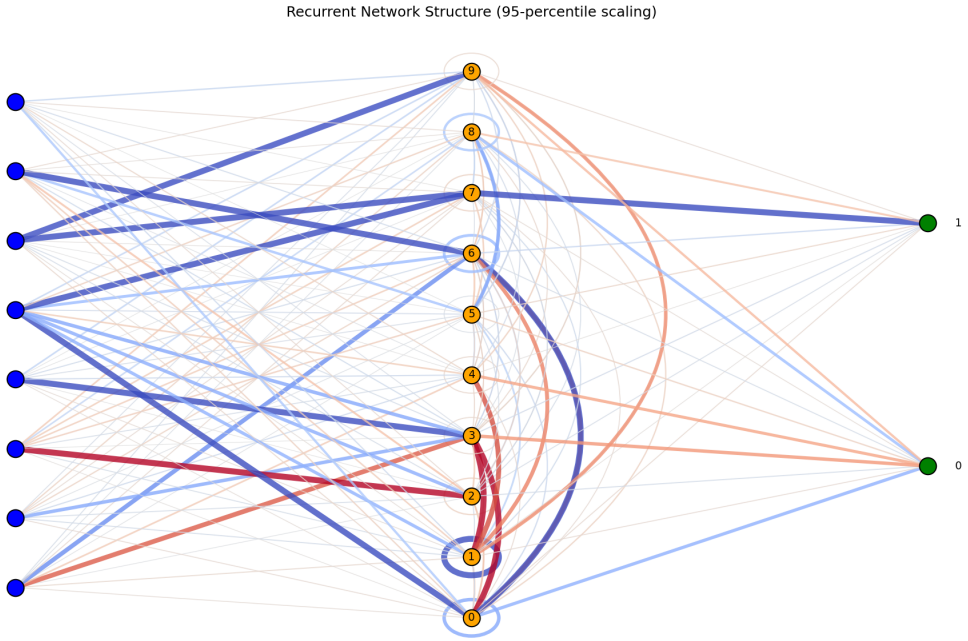

Рис. 2: Эпоха 100. Появляются выраженные, неоднородные по толщине и знаку связи. Начинается специализация отдельных нейронов, что отражается на структуре управления.

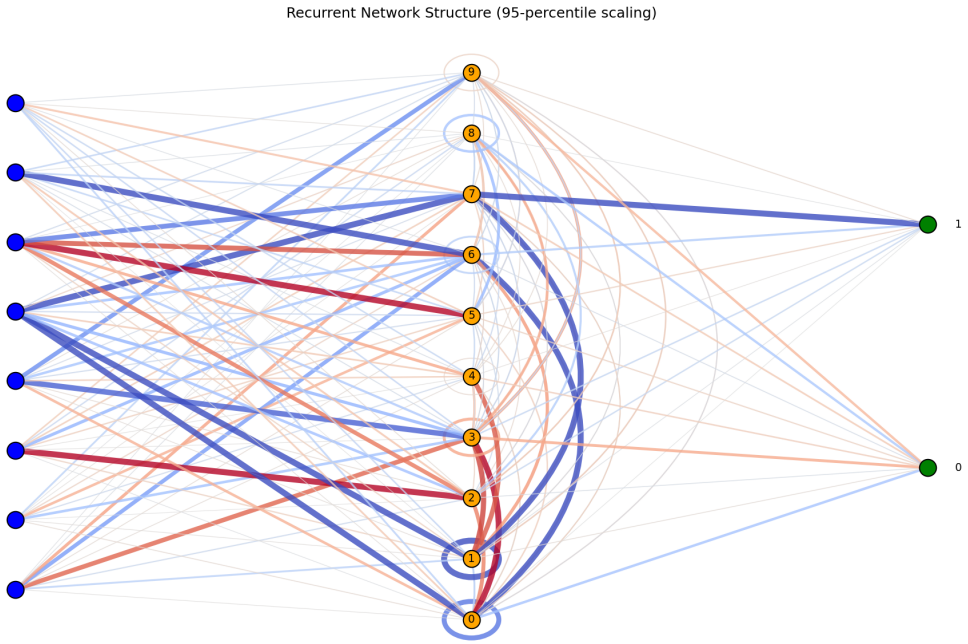

Рис. 3: Эпоха 150. Формируется модульность: отдельные нейроны скрытого слоя и их связи становятся критически важными для функционирования сети. Усиливается разница в роли нейронов.

5.1 Графики сходимости метрик

Для анализа динамики обучения строился график среднего суммарного вознаграждения (`reward`).

5.1.1 Динамика среднего вознаграждения по эпохам

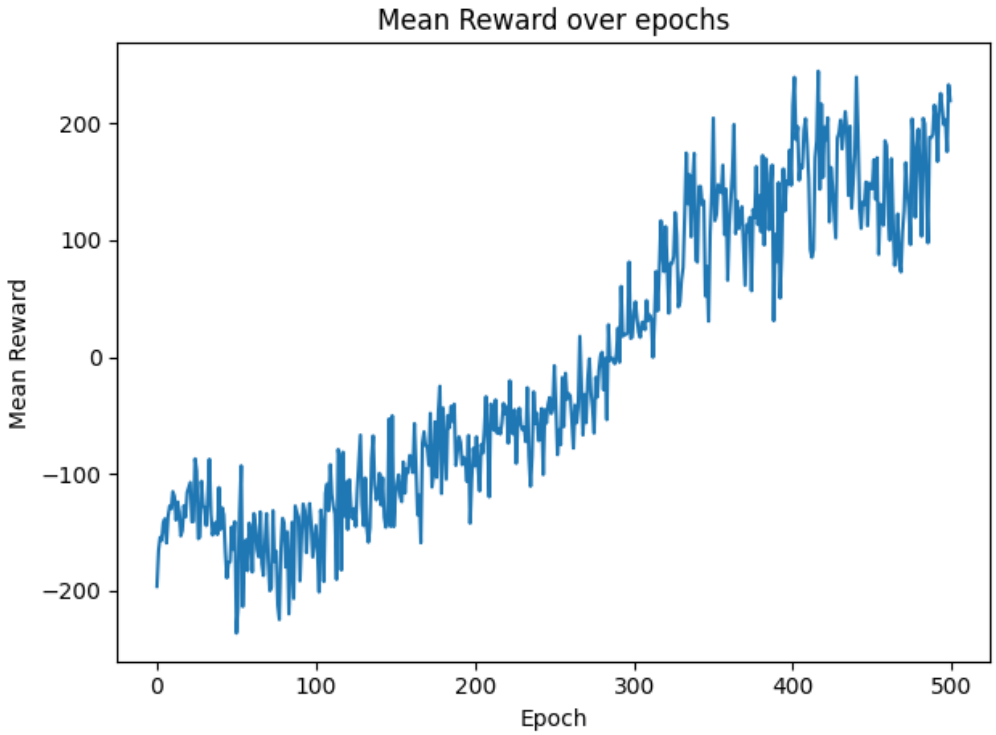

Рис. 4: Среднее суммарное вознаграждение по эпохам. Рост свидетельствует о повышении эффективности стратегии управления в процессе обучения.

6 Развертывание, тестирование и анализ результатов

6.1 Обучение модели

Для обучения модели используется команда CLI с параметрами для количества эпох и размеров скрытого слоя и подпопуляций. Пример команды для запуска обучения на 500 эпох:

```
python esp_lander.py --train --epochs 500 --hidden_size 10 --subpop_size 10
```

Примечания:

- `-train`: активирует режим обучения.
- `-epochs 500`: количество эпох обучения.
- `-hidden_size 10`: размер скрытого слоя.
- `-subpop_size 10`: размер каждой подпопуляции.

6.2 Тестирование модели

После завершения обучения, для проверки качества работы модели, выполняется процесс тестирования с использованием сохранённых весов. Для этого требуется выполнить команду:

```
python esp_lander.py --test --load_weights model.pkl
```

Где:

- `-test`: активирует режим тестирования.
- `-load_weights model.pkl`: указывает путь к файлу с весами модели, полученными в процессе обучения.

6.2.1 Результаты тестирования

Все пять тестовых эпизодов завершились успешной посадкой с высокими показателями вознаграждения. Приведем пример вывода консоли для первого тестового эпизода. Ниже проанализируем все 5 эпизодов.

Ключевые показатели успешности:

- **100% успешных посадок**: Все 5 тестовых эпизодов завершились с статусом "УСПЕШНАЯ"
- **Высокое вознаграждение**: Значения от 239.76 до 294.62 (среднее 270.33)
- **Точное позиционирование**: Среднее расстояние до цели 0.0373
- **Идеальная стабилизация**: Нулевые скорости и минимальный угол наклона

7 Заключение

В рамках данной работы была реализована и подробно исследована эволюционная стратегия ESP для обучения рекуррентной нейронной сети на задаче управления агентом в среде `LunarLanderContinuous-v3`. Был выполнен полный цикл разработки: от построения модульной архитектуры кода и создания собственного эволюционного алгоритма до визуализации структуры сети и анализа результатов тестирования.

Проведённые эксперименты показали, что метод ESP обеспечивает эффективное и устойчивое обучение агента сложным стратегиям управления без использования градиентных методов. За счёт коэволюции независимых подпопуляций удаётся достичь высокой специализации нейронов скрытого слоя и формирования компактной, адаптивной архитектуры сети.

Список использованной литературы

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

2. Such F. P., Madhavan V., Conti E. [и др.]. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning // arXiv preprint arXiv:1712.06567. — 2017. — URL: `https://arxiv.org/abs/1712.06567` (дата обращения: 29.05.2025).