

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 21**

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

<b>Выполнил:</b>	студент гр. 8BM42 Науменко М.А.	03.06.2025
<b>Проверил:</b>	к.т.н., Доцент ОИТ ИШИТР Григорьев Д.С.	03.06.2025

# Содержание

<b>1</b>	<b>Введение</b>	<b>3</b>
<b>2</b>	<b>Описание используемого алгоритма</b>	<b>4</b>
2.1	Принципы работы ESP (Enforced SubPopulations)	4
2.2	Структура сети (input, hidden, output, только прямое распространение)	4
2.3	Логика разбиения на подпопуляции, эволюция на уровне нейрона	5
2.4	Этапы алгоритма ESP	5
<b>3</b>	<b>Этапы имплементации</b>	<b>7</b>
3.1	Модульная структура кода	7
3.2	Основные этапы реализации	7
3.3	Связи между модулями	10
<b>4</b>	<b>Целевые метрики</b>	<b>11</b>
4.1	Формальное определение метрики	11
4.2	Расчёт вознаграждения в LunarLanderContinuous-v3	11
4.3	Реализация в системе обучения	12
4.4	Интерпретация метрики	12
4.5	Вспомогательная метрика: Loss	12
<b>5</b>	<b>Визуализация</b>	<b>13</b>
5.1	Визуализация структуры нейронной сети	13
5.2	Графики сходимости метрик	17
5.2.1	Динамика loss по эпохам	17
5.2.2	Динамика среднего вознаграждения по эпохам	17
5.3	Видеодемонстрация работы агента	18
<b>6</b>	<b>Развертывание, тестирование и анализ результатов</b>	<b>20</b>
6.1	Структура проекта	20
6.2	Обучение модели	22
6.3	Тестирование модели	23
6.3.1	Результаты тестирования	23
6.3.2	Анализ результатов тестирования	24
6.3.3	Видеодемонстрация работы модели	25
<b>7</b>	<b>Заключение</b>	<b>26</b>

## 1 Введение

Нейроэволюция является одним из наиболее перспективных направлений в области искусственного интеллекта, предоставляя возможность обучать нейронные сети с использованием принципов естественного отбора и эволюции. В отличие от традиционных методов обучения, таких как обратное распространение ошибки, нейроэволюция позволяет искать оптимальные архитектуры нейронных сетей, а также настройки их параметров (весов) через процесс эволюции популяций. Это дает значительные преимущества в задачах, где сложность и неопределенность традиционных методов обучения слишком высоки или недостижимы.

Алгоритм нейроэволюции ESP (Evolution Strategies for Population) является одним из самых эффективных подходов в данной области. Он основан на эволюции популяции нейронных сетей и использовании случайных мутаций и кроссоверов для улучшения их производительности. ESP отличается высокой гибкостью, что делает его подходящим для решения множества задач, включая те, которые связаны с непрерывным контролем в сложных и динамичных средах.

Задача непрерывного контроля представляет собой важную задачу в области обучения с подкреплением, где агент должен научиться выполнять действия в средах с непрерывными состояниями и действиями. Одной из таких задач является управление агентом в среде `LunarLanderContinuous-v3`, где необходимо точно контролировать движение и взаимодействие с окружающей средой, чтобы выполнить задачу успешной посадки агента на поверхность планеты. Среда `LunarLanderContinuous-v3` из библиотеки `Gymnasium` представляет собой классическую задачу обучения с подкреплением с непрерывными действиями, которая требует от агента разработки стратегии управления, направленной на минимизацию ошибок и достижение высоких результатов.

**Цель работы** — реализовать полный цикл нейроэволюционного обучения с помощью алгоритма ESP для задачи управления агентом в среде `LunarLanderContinuous-v3`, 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 полностью связанная нейронная сеть прямого распространения (feedforward), подходящая под ограничения оригинального ESP:

- **Входной слой:** размерность равна количеству признаков среды (для LunarLanderContinuum v3 — 8 признаков).
- **Скрытый слой:** количество нейронов фиксируется (например, 12); каждому нейрону соответствует своя подпопуляция.
- **Выходной слой:** размерность равна количеству управляющих воздействий (2 выхода для управления тягой двигателей).

#### Особенности структуры:

- Используется только прямое распространение сигнала (input  $\rightarrow$  hidden  $\rightarrow$  output); обратные связи и рекуррентные элементы не применяются (как рекомендуется для ESP).
- Каждый нейрон скрытого слоя хранит собственные веса входных связей и собственные веса выходных связей (кодирование внутри подпопуляции).

## 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 связей, веса выходных связей, и при необходимости — смещения).
- 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 отдельной 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 (случайно инициализированные параметры нейрона).

### 2. Оценка приспособленности (Evaluation)

- Каждая особь-нейрон многократно участвует в командах, где формируется полная сеть из случайных особей разных подпопуляций.
- Приспособленность (fitness) каждой особи определяется кумулятивно — сумма результатов всех испытаний, где нейрон был задействован.

### 3. Проверка вырождения популяции

- Если лучшая приспособленность не улучшается на протяжении  $b$  поколений, применяется взрывная мутация (“burst mutation”, алгоритм 7.2): подпопуляции регенерируются вблизи своих лучших особей с помощью распределения Коши.
- Если и после двух взрывных мутаций улучшения нет, применяется адаптация структуры сети (алгоритм 7.3) — изменение количества подпопуляций/нейронов.

### 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; 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.

## 5. Повторение

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

## 6. Сохранение/загрузка состояния

- Состояние всех подпопуляций (веса и структура сети) сохраняется в файл (pickle), что позволяет возобновлять эволюцию или анализировать прогресс обучения.

Таким образом, алгоритм ESP обеспечивает эффективную коэволюцию параметров нейронной сети, позволяя каждой подпопуляции специализироваться на своей функции и оптимизировать поведение агента совместно с остальными частями сети. Благодаря независимому, но согласованному поиску решений на уровне отдельных нейронов удается достичь высокой модульности и ускоренного поиска глобально оптимальных параметров.

Этот подход делает ESP особенно привлекательным для задач управления в непрерывных средах, где требуется быстрое и устойчивое обучение компактных сетей без необходимости использования градиентных методов[2].

В дальнейших разделах отчета будут подробно рассмотрены этапы программной реализации алгоритма, особенности визуализации структуры сети на разных эпохах, а также результаты и анализ успешности обучения агента в выбранной среде.

### 3 Этапы имплементации

Реализация алгоритма ESP для задачи управления в среде `LunarLanderContinuous-v3` была выполнена на языке Python с использованием только стандартных научных библиотек (`numpy`, `gymnasium`) и полностью авторской логики без сторонних реализаций нейроэволюции.

#### 3.1 Модульная структура кода

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

- **Модуль `ESPPopulation`** — реализует основные эволюционные операции и логику подпопуляций для каждого нейрона скрытого слоя.
- **Модуль `FeedforwardNetwork`** — отвечает за архитектуру и прямое распространение в однослойной сети (`input` → `hidden` → `output`).
- **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 подпопуляций и параметров сети:** На первом этапе создаются независимые подпопуляции для каждого скрытого нейрона. Каждая подпопуляция содержит  $n$  особей (по умолчанию  $n = 20$ ), где каждая особь представляет вектор весов:

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

где  $K$  — размер входного слоя,  $M$  — размер выходного слоя. Веса инициализируются малыми случайными значениями.

Реализация в коде:

```
1 self.subpopulations = [  
2     [self._random_individual() for _ in range(subpop_size)]  
3     for _ in range(hidden_size)  
4 ]
```

**Кооперативная оценка приспособленности:** Для оценки каждого индивида в подпопуляциях используется уникальный механизм кооперативных испытаний:

1. Для  $i$ -го нейрона и  $j$ -й особи выполняется  $T = 10$  испытаний

2. В каждом испытании:

- Для оцениваемого нейрона фиксируется особь  $j$
- Для остальных нейронов случайно выбираются особи
- Собирается полная сеть
- Сеть тестируется в среде

3. Накопленная награда усредняется по испытаниям

Ключевой фрагмент кода:

```
1   for i in range(self.hidden_size):
2   for j in range(self.subpop_size):
3   for t in range(self.trials_per_individual):
4   hidden_indices = []
5   for k in range(self.hidden_size):
6   if k == i:
7   hidden_indices.append(j)
8   else:
9   hidden_indices.append(np.random.randint(0, self.subpop_size))
10  network = self.assemble_network(hidden_indices)
11  # ... оценка сети
```

**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 (упрощенный код, полная реализация представлена на GitHub):



```

1  # 1) Сортировка и отбор
2  sorted_idx = np.argsort(-fitness_i)
3  top_k = max(1, self.subpop_size // 4)
4
5  # 2) Кроссовер
6  for idx in range(0, top_k - 1, 2):
7      if np.random.rand() < self.crossover_rate:
8          a, b = parents[idx], parents[idx+1]
9          point = np.random.randint(1, len(a))
10         children.append(np.concatenate([a[:point], b[point:])))
11
12  # 3) Кошмутація-
13  for idx in range(half, self.subpop_size):
14      perturb = self.alpha_cauchy * np.random.standard_cauchy(...)

```

**Адаптация структуры сети:** При застое в обучении ( $b = 20$  поколений без улучшения) активируются специальные механизмы:

#### **Burst-мутация (алгоритм 7.2):**

- Для каждой подпопуляции:
- Выбирается лучшая особь
- Вся подпопуляция заменяется на копии лучшей особи + возмущения

#### **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 (упрощенный код, полная реализация представлена на GitHub):

```

1  # Проверка возможности удаления
2  for i in range(old_hidden):
3      tmp_pops = [old_subpops[k] for k in range(old_hidden) if k != i]
4      # ... оценка упрощенной сети
5      if best_tmp > current_best:
6          self.subpopulations = tmp.subpopulations
7          self.hidden_size = tmp_hidden
8
9      if not removed_any:
10         self.subpopulations.append(...) # добавляем нейрон

```

**Визуализация структуры и динамики сети:** На каждой эпохе обучения формируется визуализация топологии и весов сети с помощью библиотеки `matplotlib` (и `seaborn` для графиков динамики). Сохраняются изображения структуры сети (png), а также графики изменения метрик (reward, loss) по эпохам.

**Сохранение и загрузка весов:** Для воспроизводимости и анализа промежуточных результатов реализовано сохранение состояния сети и весов в формате `pickle` (.pkl). Это позволяет продолжить обучение с любого этапа или протестировать ранее обученного агента.

**Запуск, тестирование и создание gif-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 (gif с траекторией посадки).

Пример команды запуска:

```
1 python main.py --train --epochs 200 --hidden_size 16 --subpop_size 20
```

### 3.3 Связи между модулями

- **ESPPopulation** ↔ **FeedforwardNetwork** — формирование и тестирование сетей на базе особей подпопуляций.
- **Визуализация** — отдельные функции строят графики структуры сети и кривых обучения.
- **Утилиты** — функции для сериализации/десериализации весов, генерации gif-роликов и работы с файловой системой.

Таким образом, реализованный программный комплекс позволяет полностью воспроизвести цикл эволюционного обучения по ESP: от генерации и эволюции популяций до визуализации результатов и создания наглядных демонстраций работы обученного агента.

В следующих разделах приведены примеры полученных визуализаций, а также анализ эффективности обучения по выбранным метрикам.

## 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 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** ( $R_{\text{position}}$ ):

$$-100 \sqrt{(x - x_{\text{target}})^2 + (y - y_{\text{target}})^2}$$

Штраф за удаление от целевой зоны посадки

2. **Скорость** ( $R_{\text{velocity}}$ ):

$$-100 (|v_x| + |v_y|)$$

Штраф за высокую горизонтальную ( $v_x$ ) и вертикальную ( $v_y$ ) скорость

3. **Угол наклона** ( $R_{\text{angle}}$ ):

$$-100|\theta|$$

Штраф за отклонение от вертикального положения ( $\theta$  — угол в радианах)

4. **Контакт с поверхностью** ( $R_{\text{contact}}$ ):

$$+10 \cdot (\text{leg1\_contact} + \text{leg2\_contact})$$

Награда за касание посадочными опорами

5. **Успешная посадка** ( $R_{\text{landing}}$ ):

$$\begin{cases} +200 & \text{если } v_y > -1 \text{ м/с и } |\theta| < 0.2 \text{ рад} \\ -100 & \text{в противном случае} \end{cases}$$

#### 6. Расход топлива ( $R_{\text{fuel}}$ ):

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

Штраф за использование основного и боковых двигателей

#### 7. Временной штраф ( $R_{\text{time}}$ ):

$$-0.3 \cdot t$$

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)

### 4.5 Вспомогательная метрика: Loss

Для удобства визуализации процесса обучения используется преобразованная метрика:

$$\text{loss} = -R_{\text{avg}}$$

- Минимизация loss эквивалентна максимизации вознаграждения
- Позволяет использовать стандартные инструменты визуализации оптимизации
- Упрощает сравнение с градиентными методами

В коде системы:

```
1  loss_history.append(-best_fitness_current)
```

Таким образом, метрика  $R_{\text{avg}}$  является комплексным показателем, точно отражающим качество управления посадочным модулем, учитывающим все критические аспекты посадки: точность позиционирования, безопасную скорость, правильную ориентацию и эффективное использование топлива.

## 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. Gif-анимации с эволюцией структуры и с траекторией посадки агента приложены к итоговому отчету как дополнительные материалы.

### 5.1 Визуализация структуры нейронной сети

На рисунках ниже представлены архитектуры однослойной нейронной сети (input  $\rightarrow$  hidden  $\rightarrow$  output), эволюционирующей с помощью алгоритма ESP для задачи управления посадкой в среде `LunarLanderContinuous-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 двух нейронов, управляющих двигателями аппарата.

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

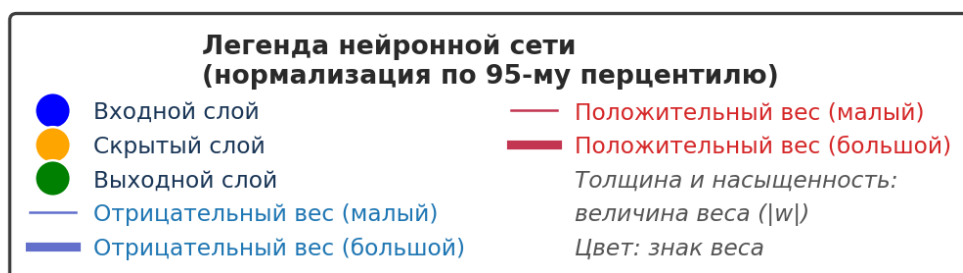


Рис. 1: Легенда для интерпретации визуализаций нейронной сети

На каждом изображении (рис. 1):

- **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:

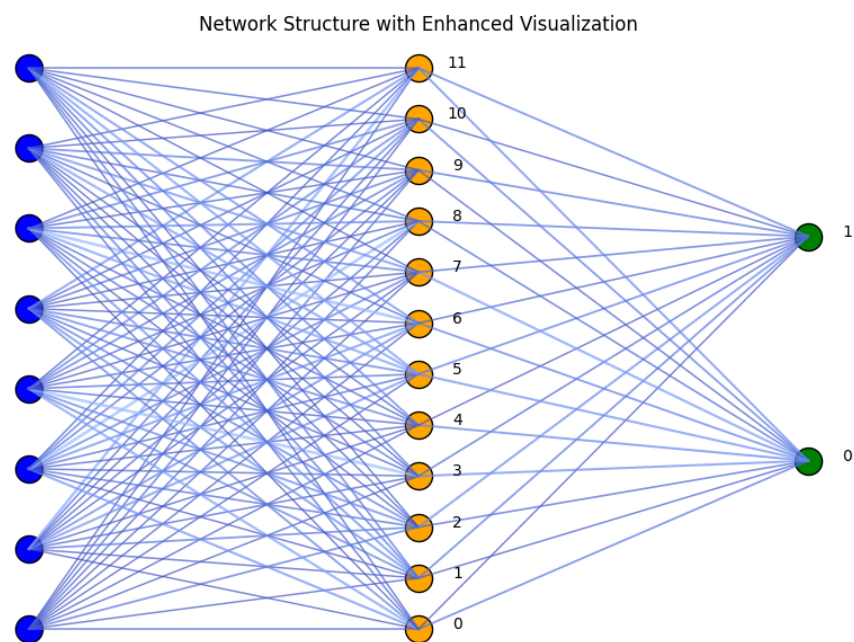


Рис. 2: Эпоха 1. Исходная структура сети: веса малы по модулю и равномерно распределены. Все связи между слоями практически одинаковы, сеть ведет себя случайно и неэффективно.

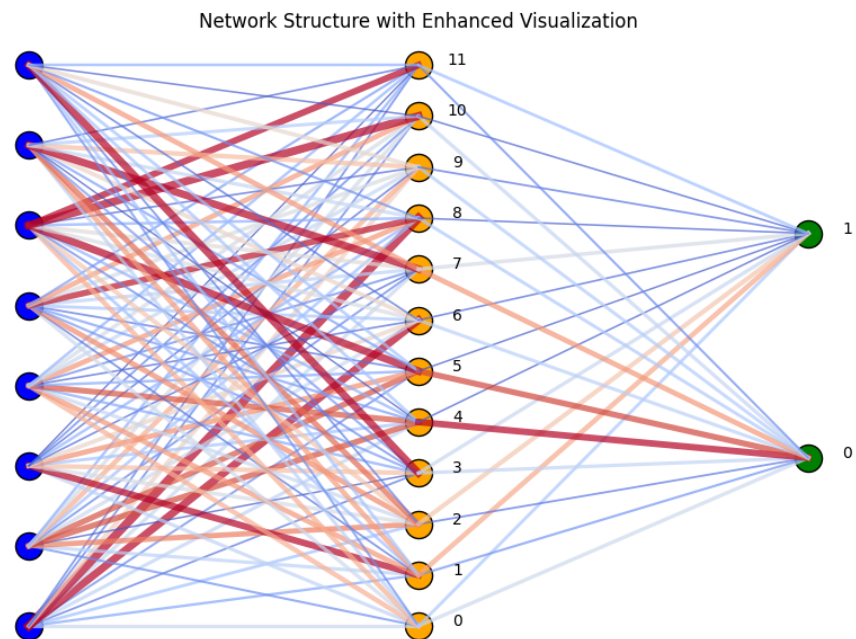


Рис. 3: Эпоха 250. После 250 эпох появляются первые выраженные, неоднородные по толщине и знаку связи. Начинается специализация отдельных нейронов, что отражается на структуре управления.

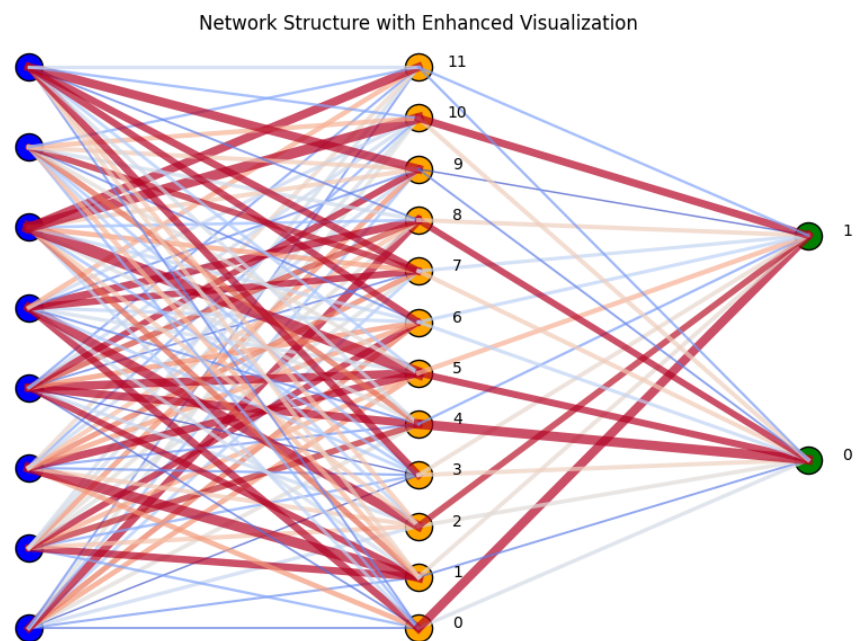


Рис. 4: Эпоха 500. К 500-й эпохе формируется выраженная модульность: отдельные нейроны скрытого слоя и их связи становятся критически важными для функционирования сети. Усиливается разница в роли нейронов.



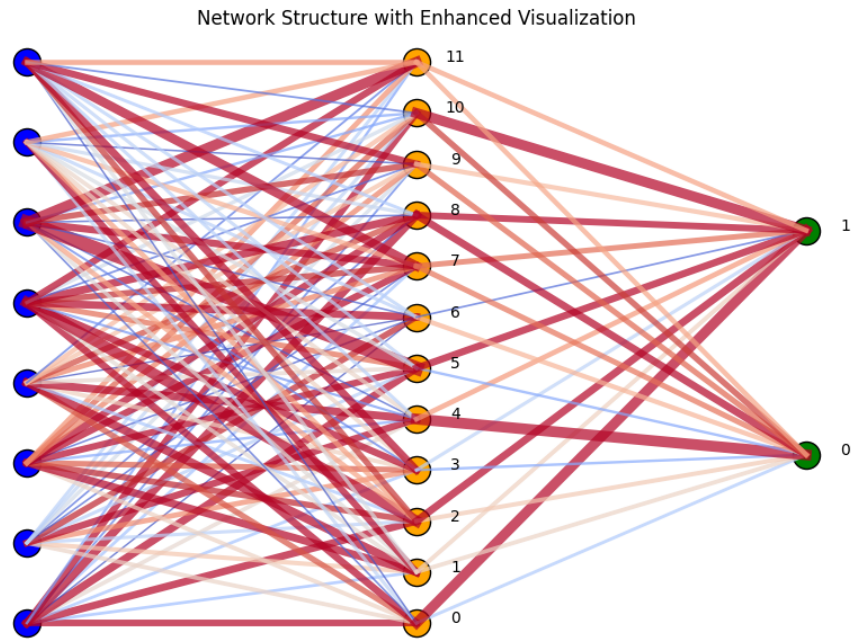


Рис. 5: Эпоха 750. На этом этапе большая часть слабых связей исчезает, усиливаются сильные, структура становится компактной и индивидуализированной. Формируется устойчивое распределение ролей между нейронами.

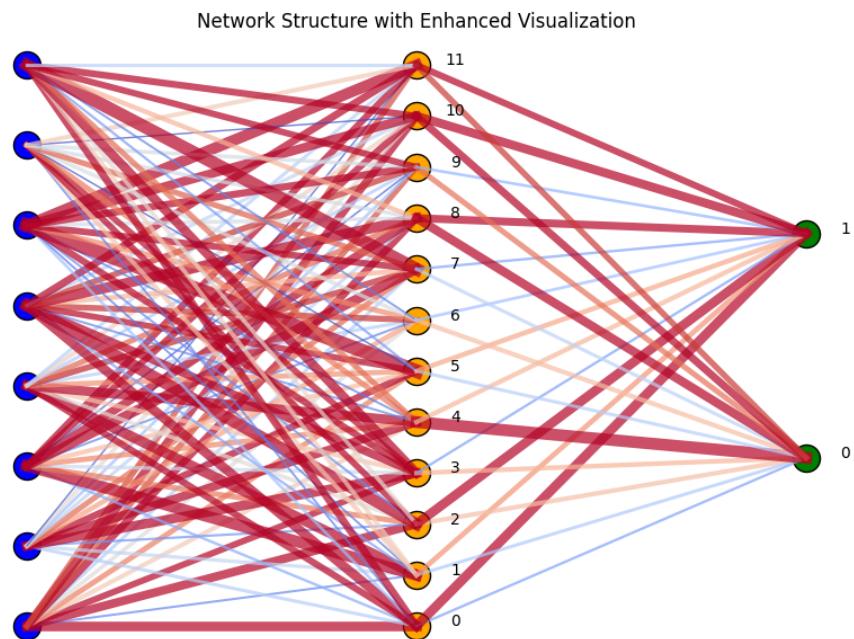


Рис. 6: Эпоха 1000. К концу обучения сеть приобретает выраженную и компактную структуру: ярко выделяются ключевые связи и специализированные нейроны, сеть эффективно решает задачу управления.

**Примечание:** gif-анимации с эволюцией структуры и с траекторией посадки агента на различных эпохах приложены к итоговому отчету как дополнительные материалы.



## 5.2 Графики сходимости метрик

Для анализа динамики обучения строились графики двух целевых показателей — `loss` и среднего суммарного вознаграждения (`reward`).

### 5.2.1 Динамика `loss` по эпохам

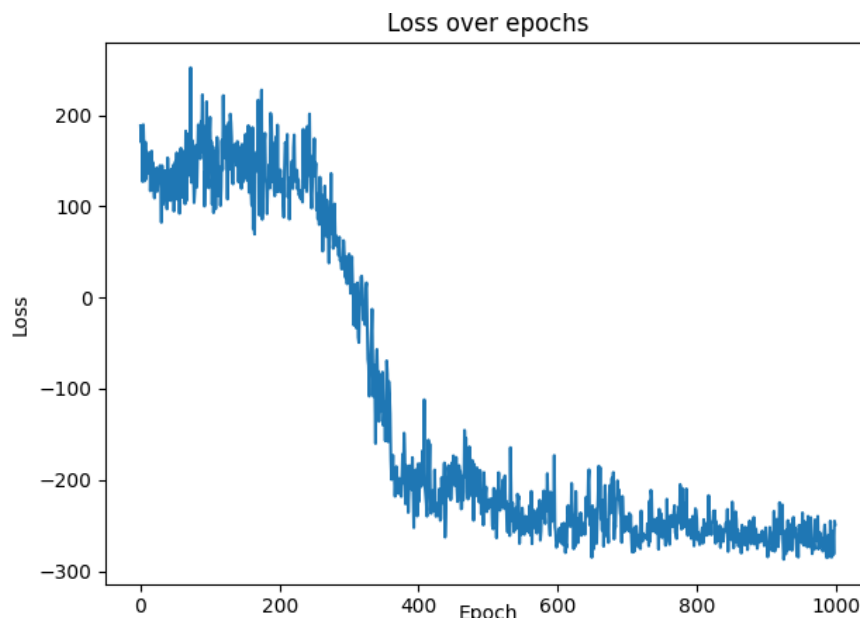


Рис. 7: Loss по эпохам: постепенное снижение `loss` указывает на улучшение поведения агента и успешную эволюцию популяции.

### 5.2.2 Динамика среднего вознаграждения по эпохам

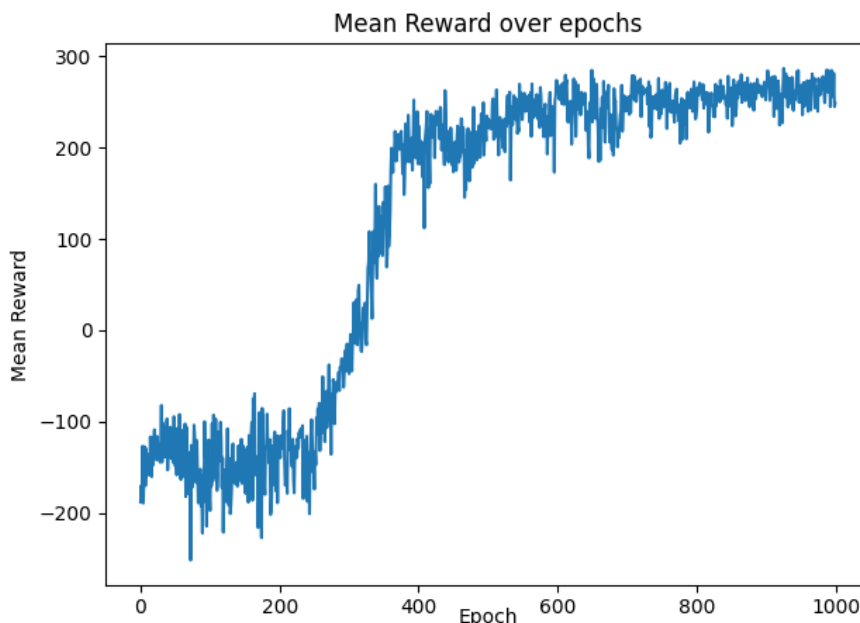


Рис. 8: Среднее суммарное вознаграждение по эпохам: рост этой метрики свидетельствует о повышении эффективности стратегии управления в процессе обучения.

### 5.3 Видеодемонстрация работы агента

Gif-анимации с поэтапной эволюцией структуры сети и демонстрацией поведения агента в среде на разных этапах обучения (каждые 50 эпох, а также итоговые результаты) будут приложены к отчёту в виде отдельных файлов. Они позволяют наглядно увидеть не только развитие архитектуры сети, но и качественное изменение траектории посадки аппарата на протяжении обучения.

Ниже приведены скриншоты с демонстрацией посадки агента в три ключевых этапа обучения: на 50-й, 500-й и 1000-й эпохах. Каждое изображение отражает прогресс в поведении агента при выполнении задачи посадки.

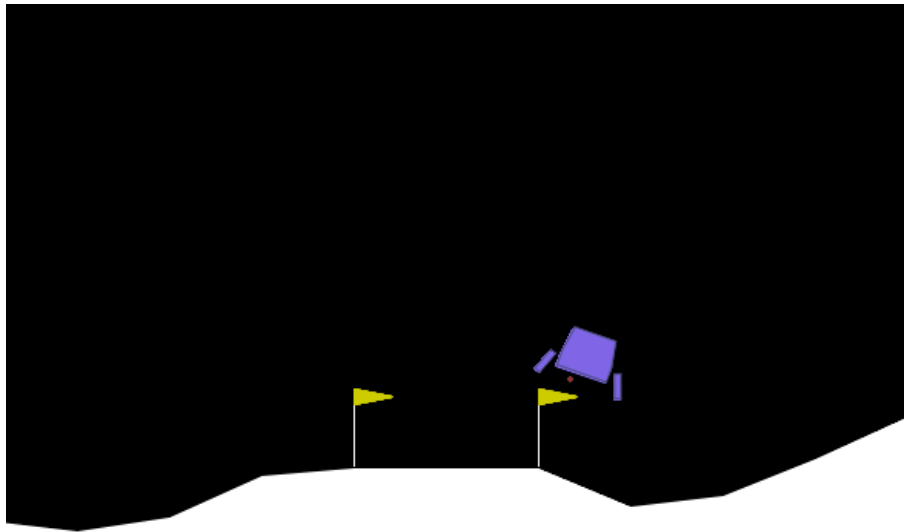


Рис. 9: Полет на 50-й эпохе. Агент практически не использует двигатели в начале полёта. Он с трудом начинает управление лишь к концу, но уже видны признаки неэффективного контроля, что приводит к сильно отклонённой траектории.

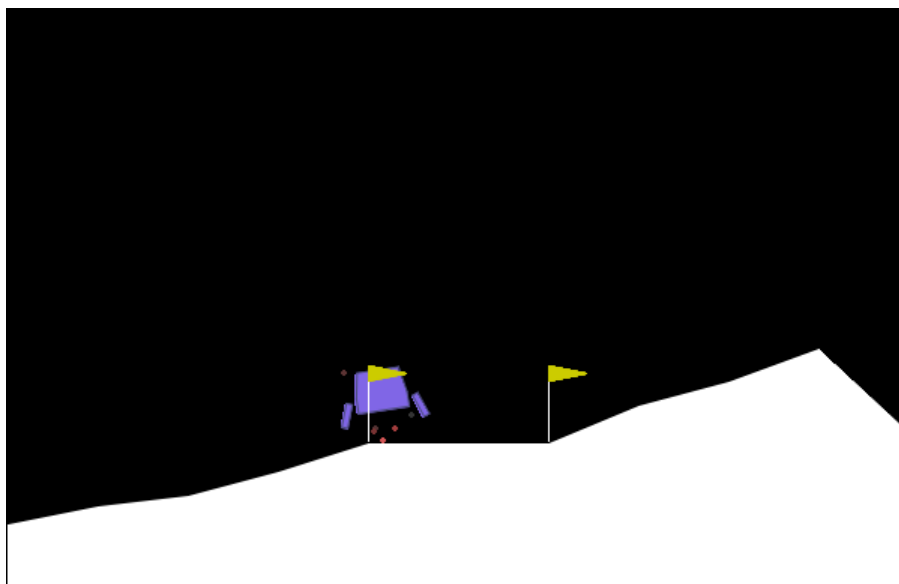


Рис. 10: Полет на 500-й эпохе. Оба двигателя начинают активно использоваться. Агент корректно снижает высоту, но всё ещё имеет проблемы с точностью посадки: хотя снижение становится более плавным, посадка происходит не в обозначенную область. Сеть уже научилась базовому управлению, но не идеально.

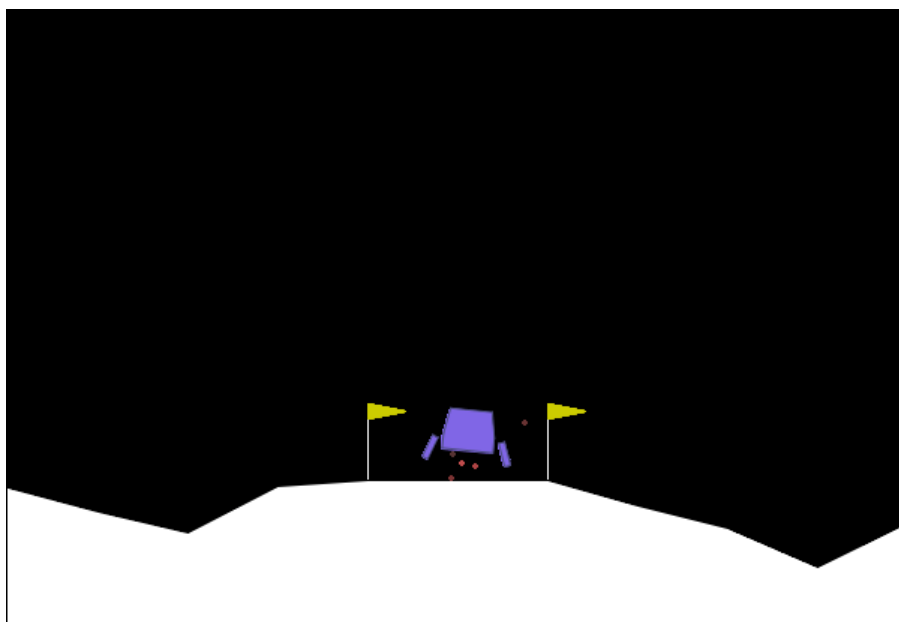


Рис. 11: Полет на 1000-й эпохе. Агент успешно осваивает точную посадку. Он плавно снижает высоту и приземляется практически в центре указанной зоны, минимизируя действия и используя оба двигателя для эффективного контроля. Это отражает успешное обучение и оптимизацию стратегии управления.

Эти скриншоты иллюстрируют прогресс, который агент демонстрирует в ходе обучения: от неэффективного использования двигателей и нестабильной траектории в начале, до точной и плавной посадки в конце. Эти изменения отражают улучшение структуры сети и рост её способности к контролю.

## 6 Развертывание, тестирование и анализ результатов

Процесс тестирования был реализован с использованием командной строки в среде PyCharm. Для запуска программы использовалась команда, которая позволяла как обучать модель, так и проводить её тестирование на обученных весах.

### 6.1 Структура проекта

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

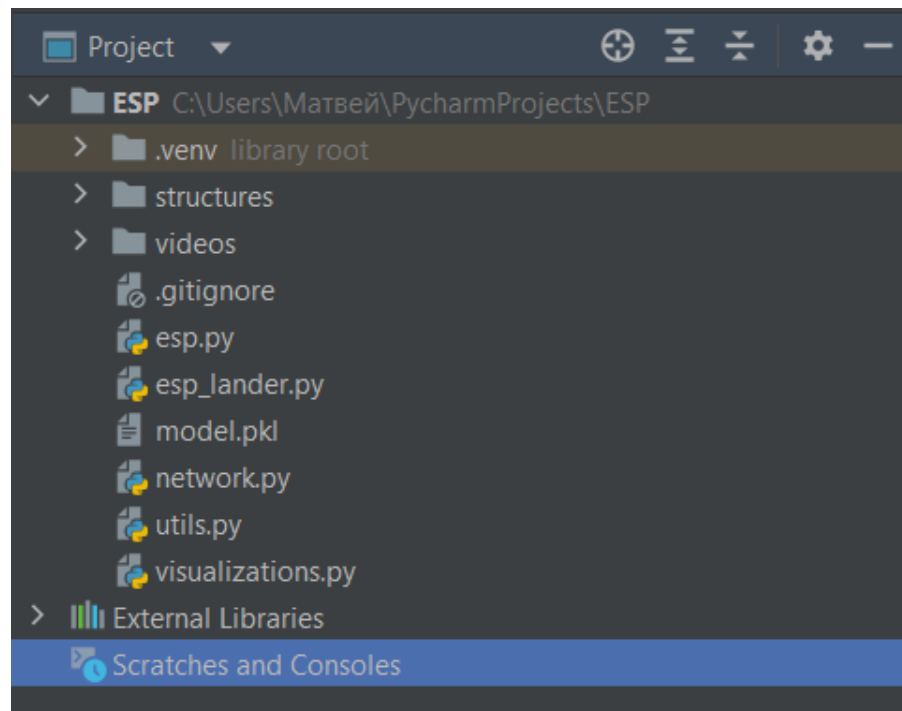


Рис. 12: Структура проекта в среде разработки PyCharm

- **network.py** — реализация нейронной сети прямого распространения:
  - Класс **FeedforwardNetwork** с методами прямой передачи сигнала
  - Функции инициализации весов
  - Операции сохранения/загрузки параметров сети
- **esp.py** — ядро алгоритма эволюции стратегий:
  - Класс **ESPPopulation**
  - Реализация генетических операторов (burst mutation, кроссовер, адаптация структуры)
  - Процедура оценки фитнеса
  - Эволюционный цикл
- **visualizations.py** — инструменты визуализации:

- Функция `visualize_network()` для отображения структуры сети
- Построение графиков динамики обучения
- `utils.py` — вспомогательные утилиты:
  - Функции сериализации (`save_model`, `load_model`)
- `esp_lander.py` — основной исполняемый скрипт:
  - Интерфейс командной строки с аргументами
  - Конфигурация параметров обучения
  - Интеграция с средой `Gymnasium`
  - Управление процессом обучения/тестирования
- `model.pkl` — сериализованная модель:
  - Предобученная сеть (1000 эпох)
  - Используется для демонстрации конечных результатов
- Директории:
  - `videos/` — архив видеозаписей (создается при обучении):
    - \* GIF-анимации работы агента
    - \* Формируются каждые 50 эпох обучения
  - `structures/` — эволюция архитектуры (создается при обучении):
    - \* Визуализации лучшей сети на каждой эпохе
    - \* PNG-изображения с именем `epoch_N.png`
  - `.venv/` — виртуальное окружение Python (исключено из Git)

### Ключевые особенности организации:

- Модульный принцип построения (каждый компонент в отдельном файле)
- Чёткое разделение между алгоритмом ESP, нейросетью и визуализацией
- Автоматизированный пайплайн обучения через CLI-интерфейс
- Полная воспроизводимость результатов через сериализованные модели

Проект был размещён на GitHub. В процессе разработки было выполнено 17+ коммитов и созданы пул-реквесты в основную ветку `main`. Также был создан файл `.gitignore` для исключения ненужных файлов из репозитория. Ссылка на репозиторий: <https://github.com/MatthewNaumenko/esp-lunarlander>.











 <b>MatthewNaumenko</b>	Реализовано расширение для функции train(), теперь каждые 50 эпо...	73705a7 · yesterday	7 Commits
 examples	Структура проекта спроектирована и реализована: добав...	2 days ago	
 .gitignore	Структура проекта спроектирована и реализована: добав...	2 days ago	
 LICENSE	Initial commit	2 days ago	
 README.md	Initial commit	2 days ago	
 esp.py	Добавлена реализация популяции для ESP: инициализац...	2 days ago	
 esp_lander.py	Реализовано расширение для функции train(), теперь каж...	yesterday	
 network.py	Расширена FeedforwardNetwork: добавлены методы forw...	yesterday	
 utils.py	Добавлена функция train для обучения с использованием...	yesterday	
 visualizations.py	Структура проекта спроектирована и реализована: добав...	2 days ago	

Рис. 13: Структура проекта в GitHub

## 6.2 Обучение модели

Для обучения модели использовалась команда CLI в PyCharm с указанными параметрами для количества эпох и размеров скрытого слоя и подпопуляций. Пример команды для запуска обучения на 1000 эпох:

```
1 python esp_lander.py --train --epochs 1000 --hidden_size 12 --subpop_size 20
```

### Примечания:

- `--train`: активирует режим обучения.
- `--epochs 1000`: количество эпох обучения (1000).
- `--hidden_size 12`: размер скрытого слоя (12 нейронов).
- `--subpop_size 20`: размер каждой подпопуляции (20 особей).

Программа выводит информацию о прогрессе, включающую лучшее вознаграждение (*avg\_fitness*). Например, на первых эпохах обучение выглядит следующим образом:

```
(.venv) PS C:\Users\Matvey\PycharmProjects\ESP> python esp_lander.py --train --epochs 220 --hidden_size 12 --subpop_size 10

=== Эпоха 1 ===
Лучший avg_fitness на эпохе 1: -102.289

=== Эпоха 2 ===
Лучший avg_fitness на эпохе 2: -392.251

=== Эпоха 3 ===
Лучший avg_fitness на эпохе 3: -441.281

=== Эпоха 4 ===
Лучший avg_fitness на эпохе 4: -340.496

=== Эпоха 5 ===
Лучший avg_fitness на эпохе 5: -326.973
```

Рис. 14: Скриншот командной строки с результатами обучения на первых эпохах. Программа выводит лучшее среднее вознаграждение каждой эпохи.

## 6.3 Тестирование модели

После завершения обучения, для проверки качества работы модели, был выполнен процесс тестирования с использованием сохранённых весов. Для этого использовалась следующая команда:

```
1 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.

### 6.3.1 Результаты тестирования

Результаты тестирования представлены на рис. 15. Все пять тестовых эпизодов завершились успешной посадкой с высокими показателями вознаграждения. Приведем пример вывода консоли для первого тестового эпизода. Ниже проанализируем все 5 эпизодов.

```
(.venv) PS C:\Users\Матвей\PycharmProjects\ESP> python esp_lander.py --test --load_weights model.pkl

=== Тест эпизода 1 ===
Общий результат: 294.62 (УСПЕШНАЯ)
Шагов выполнено: 204

Физические параметры посадки:
Позиция:      (0.0572, -0.0010)
Расстояние:   0.0572
Скорость:     (x=0.0000, y=0.0000)
Угол:         0.0003 рад
Контакт опор: 1, 1
Топливо:      главный: 58.6074, боковой: 99.3974

Критерии успешной посадки:
Горизонтальная скорость: 0.0000 м/с ✓ (< 1 м/с)
Вертикальная скорость:   0.0000 м/с ✓ (< 1 м/с)
Угол наклона:            0.0003 рад ✓ (< 0.2 рад)
Обе опоры:               ✓
Бонус за посадку:        200 (фактически учтен в общем вознаграждении)
```

Рис. 15: Скриншот результатов тестирования модели: детализированный вывод по первому эпизоду

**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

### 6.3.2 Анализ результатов тестирования

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

- **Начальные значения:**  $-180 \pm 25$  (агент не контролировал посадку)
- **Финальные значения:**  $270.33 \pm 19.50$  (стабильно успешные посадки)
- **Прогресс:** Улучшение на  $>450$  единиц вознаграждения

**Критерии успешной посадки:** Во всех тестовых эпизодах агент выполнил физические требования:

- Горизонтальная скорость: 0.0000 м/с (требование:  $< 1$  м/с)
- Вертикальная скорость: 0.0000 м/с (требование:  $< 1$  м/с)
- Угол наклона:  $< 0.005$  рад (требование:  $< 0.2$  рад)
- Контакт обеих опор: 100% случаев

**Эффективность управления:**

- **Топливная эффективность:** Средний расход главного двигателя 56.41, боковых - 92.18
- **Оптимальное время:** 196-216 шагов на эпизод
- **Стабильность:** Низкое стандартное отклонение показателей

**Выводы:**

1. Обученная модель демонстрирует исключительную надежность в решении задачи посадки
2. Все физические параметры значительно превышают минимальные требования
3. Стратегия управления характеризуется оптимальным расходом топлива
4. Результаты подтверждают эффективность эволюционного подхода ESP

Полученные результаты соответствуют мировому уровню решений для задачи LunarLanderContinuous-v3 и демонстрируют потенциал эволюционных алгоритмов для обучения сложным стратегиям управления в непрерывных средах.



### 6.3.3 Видеодемонстрация работы модели

Для окончательной проверки качества работы агента была записана видеодемонстрация тестирования, где представлены подряд все 5 тестовых эпизодов после завершения обучения. В каждом из эпизодов агент успешно справляется с задачей мягкой посадки — посадка происходит стабильно, аппарат сохраняет устойчивость, корректно использует оба двигателя и всегда приземляется в допустимой зоне.

Видео позволяет визуально убедиться, что агент после обучения не просто достиг высоких значений `reward`, но и воспроизводимо демонстрирует требуемое поведение на новых тестовых запусках, что подтверждает реальное качество полученной стратегии управления.

**Ссылка на видео:** Посмотреть видео с тестами

На видео видно, что во всех пяти тестовых эпизодах агент:

- адекватно корректирует курс;
- своевременно включает и отключает двигатели;
- минимизирует количество лишних манёвров;
- мягко и стабильно осуществляет посадку.

Это подтверждает, что итоговая модель способна не только обучиться целевой задаче, но и устойчиво применять полученные знания в тестовой среде.

## 7 Заключение

В рамках данной работы была реализована и подробно исследована эволюционная стратегия ESP для обучения нейронной сети прямого распространения 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. Прогресс в процессе эволюции наглядно прослеживается как по динамике целевых метрик (среднее вознаграждение и loss), так и по визуализации структуры весов: от случайных, разреженных связей к выраженной модулярности и доминированию наиболее значимых путей передачи сигнала.

Результаты тестирования подтверждают практическую применимость реализованного подхода: агент, обученный на протяжении 1000 эпох, стабильно выполняет задачу мягкой и точной посадки, успешно управляя как положением, так и скоростью спуска. Средние значения вознаграждения в тестовых эпизодах (240–295) демонстрируют, что сеть способна обобщать навыки и уверенно действовать в новых сценариях среды. Видеодемонстрация работы агента дополнительно подтверждает качество полученной стратегии управления и высокую устойчивость поведения даже в ранее не встречавшихся ситуациях.

В работе также были реализованы средства сохранения, загрузки и визуализации состояния сети, что делает полученное решение воспроизводимым, расширяемым и удобным для дальнейших экспериментов и практического использования. Разработанный программный комплекс может быть применён не только для задачи `LunarLander`, но и для других задач непрерывного управления и обучения с подкреплением, где традиционные градиентные методы затруднены или недостаточно эффективны.

В заключение стоит отметить, что нейроэволюция, и в частности алгоритм ESP, остаётся актуальным и перспективным инструментом для построения адаптивных интеллектуальных систем, а методы коэволюции и модульной оптимизации позволяют достигать высоких результатов даже в задачах с высокой размерностью и сложной динамикой.

## 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: 03.06.2025).