

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

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

Содержание

1	Введение	3
2	Описание используемого алгоритма	3
2.1	Принципы работы ESP (Enforced SubPopulations)	3
2.2	Структура сети (input, hidden, output, только прямое распространение)	4
2.3	Логика разбиения на подпопуляции, эволюция на уровне нейрона	4
2.4	Этапы алгоритма ESP	5
3	Этапы имплементации	6
3.1	Модульная структура кода	6
3.2	Основные этапы реализации	7
3.3	Связи между модулями	8
4	Целевые метрики	9
5	Визуализация	9
5.1	Визуализация структуры нейронной сети	10
5.2	Графики сходимости метрик	13
5.2.1	Динамика <code>loss</code> по эпохам	13
5.2.2	Динамика среднего вознаграждения по эпохам	13
5.3	Видеодемонстрация работы агента	14
6	Развертывание, тестирование и анализ результатов	16
6.1	Структура проекта	16
6.2	Обучение модели	16
6.3	Тестирование модели	17
6.3.1	Результаты тестирования	17
6.3.2	Анализ результатов тестирования	18
6.3.3	Видеодемонстрация работы модели	18
7	Заключение	19
	Приложение. Листинги кода	21
	<code>esp_lander.py</code>	21
	<code>esp.py</code>	23
	<code>network.py</code>	26
	<code>utils.py</code>	28
	<code>visualizations.py</code>	28
	<code>model_example.pkl</code>	30

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 се-

тей (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 LunarLanderContinuous3 — 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 удастся 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 (отбор, кроссовер, мутация) 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 задаются размеры входного, скрытого и выходного слоя. Для каждого нейрона скрытого слоя создаётся подпопуляция, в которой каждая особь — это вектор вещественных весов (сумма весов входов + весов выходов). Все веса инициализируются случайно с малой дисперсией, что соответствует практике эволюционного обучения.

Пример (фрагмент кода):

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

Формирование сети и эволюционных команд: Для каждой попытки эволюции (trial) случайным образом из каждой подпопуляции выбирается по одной особи, из которых собирается скрытый слой сети. Сеть динамически формируется функцией `assemble_network`, где веса входов и выходов каждого нейрона берутся из выбранных особей подпопуляций.

Оценка приспособленности (фитнес-функция): Каждая сформированная сеть тестируется в среде `LunarLanderContinuous-v3`, где накапливается суммарное вознаграждение за эпизод (основная целевая метрика). Особенность — оценка проводится в “командах”: один и тот же нейрон участвует в разных комбинациях, что способствует специализации и диверсификации решений.

Пример (фрагмент кода):

```
1 for trial in range(self.subpop_size):
2     hidden_indices = [trial] * self.hidden_size
3     network = self.assemble_network(hidden_indices)
4     # ... тестирование сетивсреде
```

Эволюционные операции: отбор, кроссовер и мутация

- **Отбор** реализуется через турнирную селекцию для каждой подпопуляции: случайным образом выбирается несколько особей, и лучшая копируется в следующее поколение.

- **Кроссовер** (одноточечный): пары особей обмениваются частями своих векторов весов с вероятностью, заданной гиперпараметром.
- **Мутация**: к весам особей добавляется небольшое случайное отклонение (гауссовский шум), что поддерживает генетическое разнообразие и предотвращает преждевременную сходимость.

Пример (фрагмент кода):

```

1  # Кроссовер
2  if np.random.rand() < self.crossover_rate:
3      a, b = subpop[i], subpop[(i+1) % self.subpop_size]
4      point = np.random.randint(1, len(a))
5      child1 = np.concatenate([a[:point], b[point:]])
6      child2 = np.concatenate([b[:point], a[point:]])

```

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

Для оценки эффективности обучения был выбран показатель среднего суммарного вознаграждения за эпизод, получаемого агентом в среде `LunarLanderContinuous-v3`. Эта метрика идеально соответствует задаче, так как она напрямую отражает успех посадки космического аппарата: чем выше вознаграждение, тем лучше агент управляет процессом посадки, минимизируя использование топлива и корректно реагируя на изменения в окружении.

Среднее вознаграждение рассчитывается для каждой особи в популяции через несколько эпизодов, и на основе этой метрики оценивается пригодность (fitness) сети. Для вычисления метрики используется следующий фрагмент кода:

Фрагмент кода:

```
1 mean_reward = np.mean(fitness[0]) # Вознаграждение
```

Кроме того, для отслеживания прогресса обучения рассчитывается `loss`, который является отрицательным значением вознаграждения (то есть `loss = -reward`). Это позволяет визуализировать обучение и отслеживать сходимость модели, где снижение `loss` соответствует улучшению поведения агента:

Фрагмент кода:

```
1 loss = -mean_reward # Loss
```

Таким образом, метрика среднего вознаграждения используется для оценки успешности обучения и оптимизации стратегии агента, а `loss` служит вспомогательной метрикой для визуализации и анализа динамики процесса обучения.

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 ($\text{input} \rightarrow \text{hidden} \rightarrow \text{output}$), 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 “текущей” сети — она собирается из первых особей каждой подпопуляции на данной эпохе. Такой способ типичен для алгоритмов класса ESP: в процессе коэволюции не существует уникальной “лучшей” сети для каждой эпохи, а фиксированный способ сборки (например, по первым особям) обеспечивает наглядную и объективную демонстрацию хода эволюции архитектуры.

На каждом изображении:

- Толщина и цвет линии отражают величину и знак весового коэффициента между соответствующими нейронами (более яркие и толстые линии соответствуют весам с большим модулем).
- Входные нейроны показаны синим цветом, скрытые — оранжевым, выходные — зелёным.
- По мере обучения становится заметно, как отдельные связи усиливаются, формируются функциональные “пути”, а часть нейронов специализируется.

Примеры визуализации структуры сети на разных эпохах обучения:

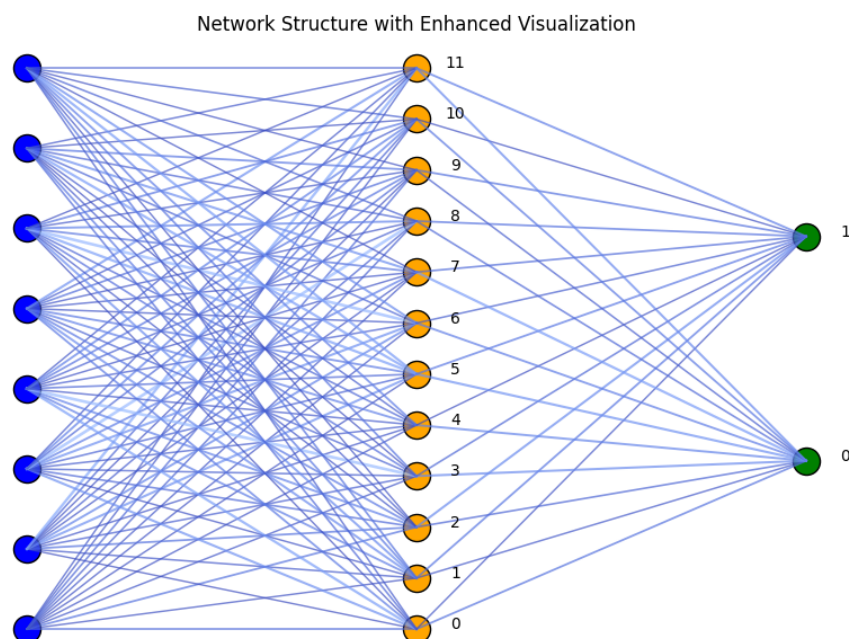


Рис. 1: Эпоха 1. Исходная структура сети: веса малы по модулю и равномерно распределены. Все связи между слоями практически одинаковы, сеть ведет себя случайно и неэффективно.

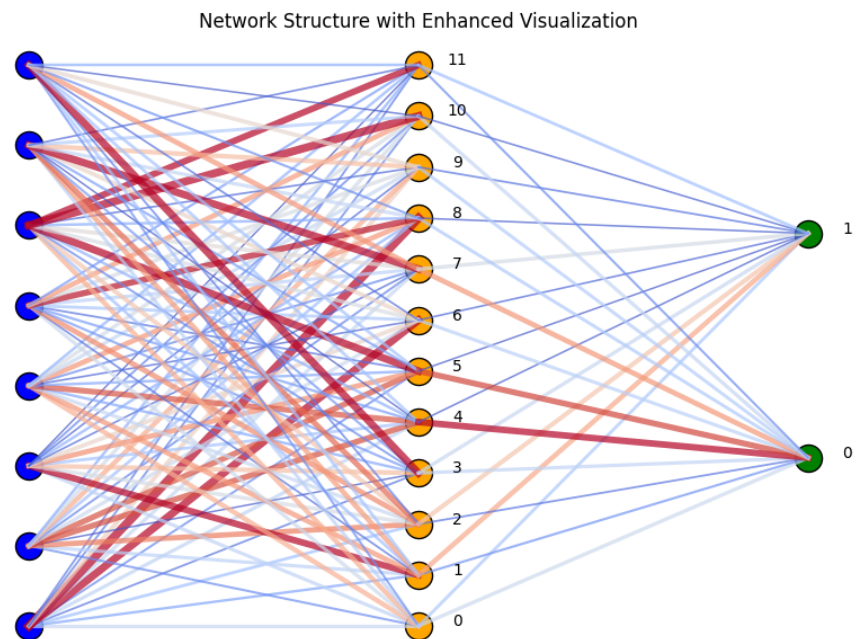


Рис. 2: Эпоха 250. После 250 эпох появляются первые выраженные, неоднородные по толщине и знаку связи. Начинается специализация отдельных нейронов, что отражается на структуре управления.

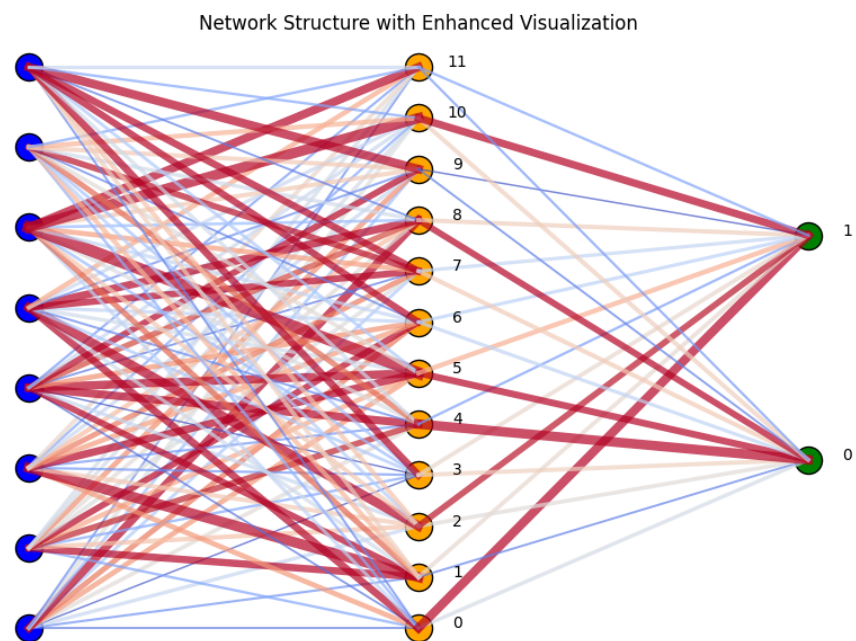


Рис. 3: Эпоха 500. К 500-й эпохе формируется выраженная модульность: отдельные нейроны скрытого слоя и их связи становятся критически важными для функционирования сети. Усиливается разница в роли нейронов.

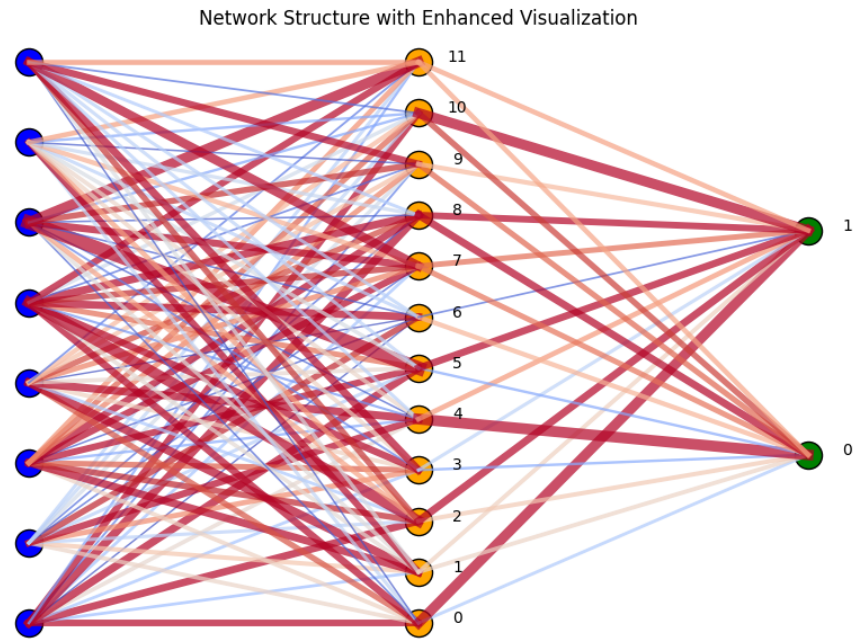


Рис. 4: Эпоха 750. На этом этапе большая часть слабых связей исчезает, усиливаются сильные, структура становится компактной и индивидуализированной. Формируется устойчивое распределение ролей между нейронами.

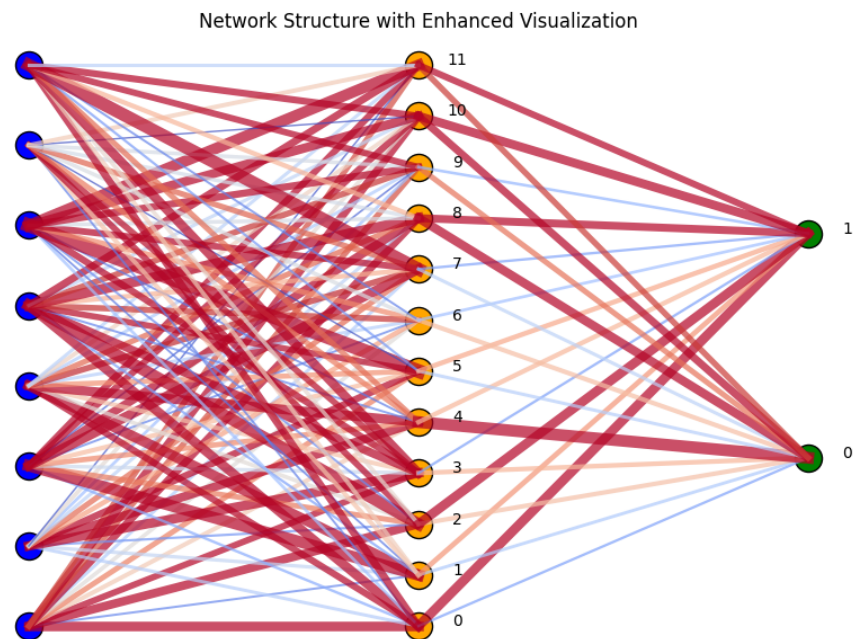


Рис. 5: Эпоха 1000. К концу обучения сеть приобретает выраженную и компактную структуру: ярко выделяются ключевые связи и специализированные нейроны, сеть эффективно решает задачу управления.

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

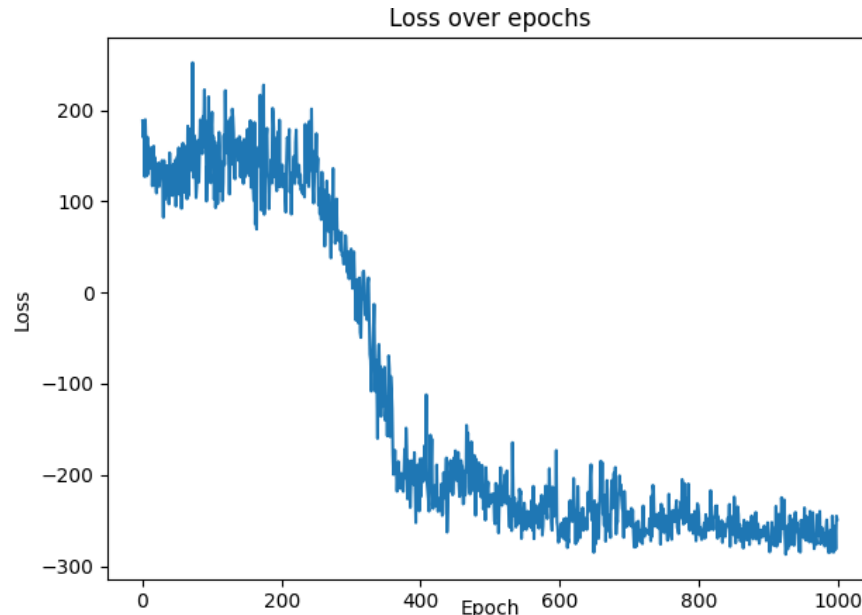


Рис. 6: Loss по эпохам: постепенное снижение `loss` указывает на улучшение поведения агента и успешную эволюцию популяции.

5.2.2 Динамика среднего вознаграждения по эпохам

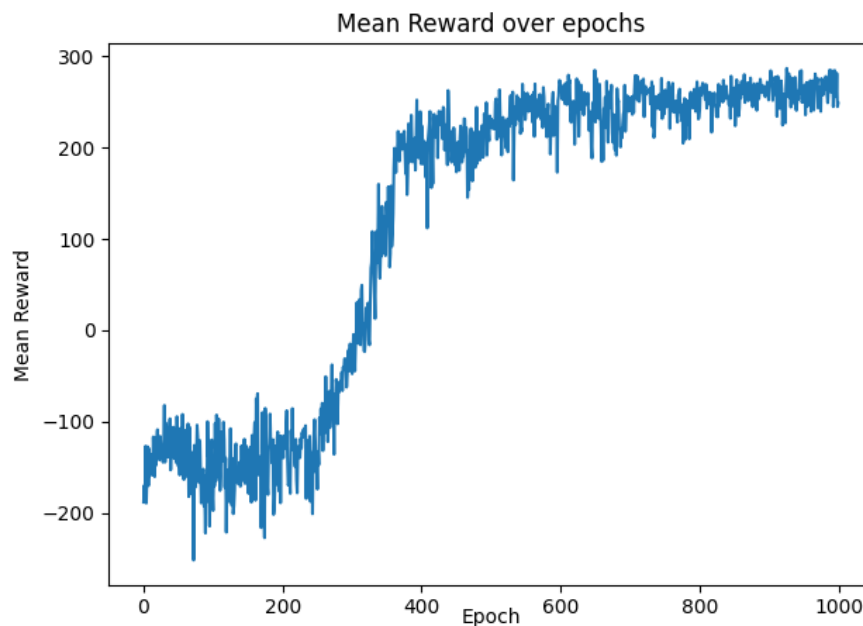


Рис. 7: Среднее суммарное вознаграждение по эпохам: рост этой метрики свидетельствует о повышении эффективности стратегии управления в процессе обучения.

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.

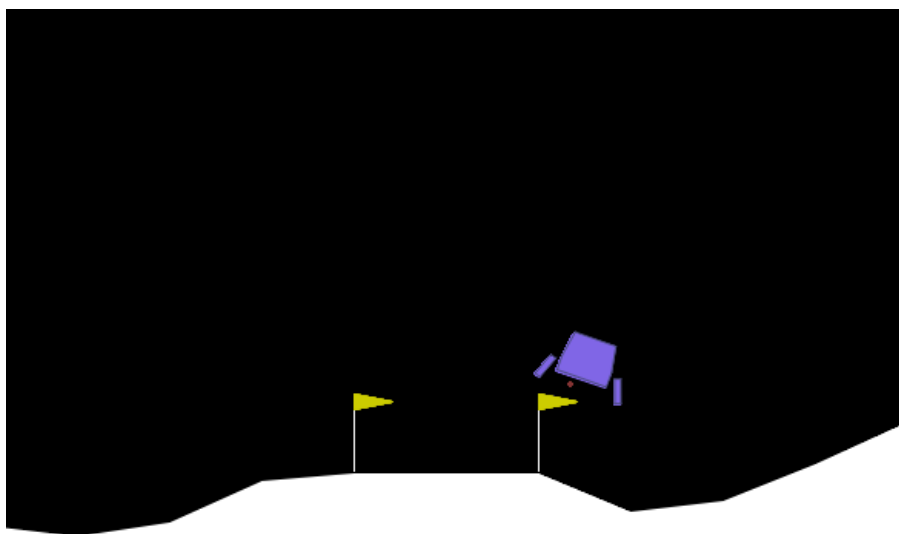


Рис. 8: Полет на 50-й эпохе. Агент практически не использует двигатели в начале полёта. Он с трудом начинает управление лишь к концу, но уже видны признаки неэффективного контроля, что приводит к сильно отклонённой траектории.

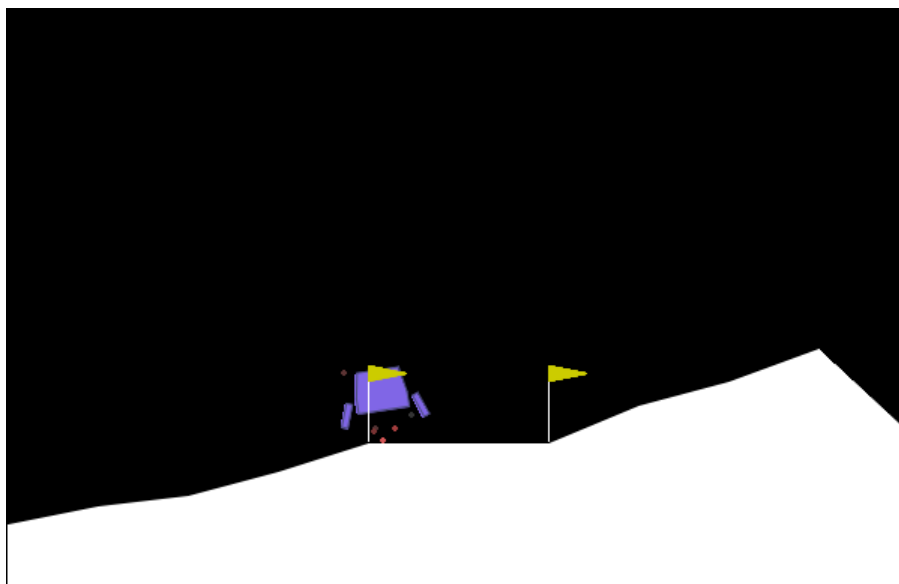


Рис. 9: Полет на 500-й эпохе. Оба двигателя начинают активно использоваться. Агент корректно снижает высоту, но всё ещё имеет проблемы с точностью посадки: хотя снижение становится более плавным, посадка происходит не в обозначенную область. Сеть уже научилась базовому управлению, но не идеально.

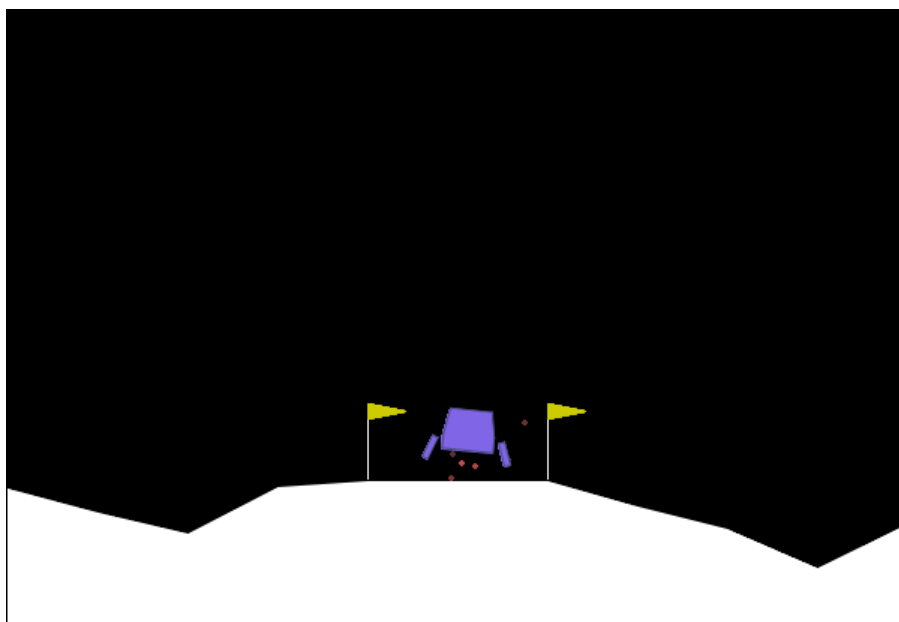


Рис. 10: Полет на 1000-й эпохе. Агент успешно осваивает точную посадку. Он плавно снижает высоту и приземляется практически в центре указанной зоны, минимизируя действия и используя оба двигателя для эффективного контроля. Это отражает успешное обучение и оптимизацию стратегии управления.

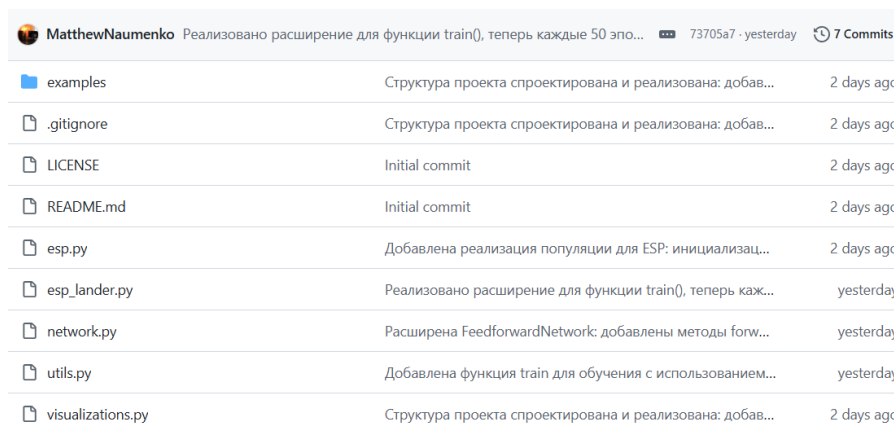
Эти скриншоты иллюстрируют прогресс, который агент демонстрирует в ходе обучения: от неэффективного использования двигателей и нестабильной траектории в начале, до точной и плавной посадки в конце. Эти изменения отражают улучшение структуры сети и рост её способности к контролю.

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 был структурирован следующим образом:



MatthewNaumenko 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. 11: Структура проекта: все основные компоненты, включая модули для обучения, тестирования, визуализации и утилит.

Проект был размещён на GitHub. В процессе разработки было выполнено 9 коммитов и создан пул-реквест в основную ветку `main`. Также был создан файл `.gitignore` для исключения ненужных файлов из репозитория. Ссылка на репозиторий: <https://github.com/MatthewNaumenko/esp-lunarlander>.

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, включая среднее вознаграждение (*Mean reward*) и значение потерь (*loss*) для каждой эпохи. Например, на первых эпохах обучение выглядит следующим образом:

```
(.venv) PS C:\Users\Матвей\PycharmProjects\ESP> python esp_lander.py --train --epochs 1000 --hidden_size 12 --subpop_size 20
Epoch 1: Mean reward -188.49, loss 188.49
Epoch 2: Mean reward -170.02, loss 170.02
Epoch 3: Mean reward -183.24, loss 183.24
Epoch 4: Mean reward -126.97, loss 126.97
Epoch 5: Mean reward -189.53, loss 189.53
Epoch 6: Mean reward -157.76, loss 157.76
Epoch 7: Mean reward -127.70, loss 127.70
Epoch 8: Mean reward -170.43, loss 170.43
Epoch 9: Mean reward -129.84, loss 129.84
Epoch 10: Mean reward -159.51, loss 159.51
Epoch 11: Mean reward -135.73, loss 135.73
Epoch 12: Mean reward -150.62, loss 150.62
Epoch 13: Mean reward -153.88, loss 153.88
Epoch 14: Mean reward -158.53, loss 158.53
Epoch 15: Mean reward -116.85, loss 116.85
Epoch 16: Mean reward -139.17, loss 139.17
```

Рис. 12: Скриншот командной строки с результатами обучения на первых эпохах. Программа выводит среднее вознаграждение и значение loss для каждой эпохи. Эти метрики позволяют отслеживать прогресс обучения.

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

После окончания обучения был выполнен процесс тестирования, и результаты по каждому из пяти эпизодов были следующими:

```
1 Test episode 1, reward: 286.66
2 Test episode 2, reward: 302.70
3 Test episode 3, reward: 260.61
4 Test episode 4, reward: 273.05
5 Test episode 5, reward: 312.64
```

```
(.venv) PS C:\Users\Матвей\PycharmProjects\ESP> python esp_lander.py --test --load_weights model.pkl
Test episode 1, reward: 286.66
Test episode 2, reward: 302.70
Test episode 3, reward: 260.61
Test episode 4, reward: 273.05
Test episode 5, reward: 312.64
```

Рис. 13: Скриншот результатов тестирования модели: вывод среднего вознаграждения за эпизод в ходе теста.

6.3.2 Анализ результатов тестирования

На ранних этапах обучения, в частности на первых эпохах, значение reward было значительно ниже (около -180). Это свидетельствует о том, что на старте агент не мог эффективно взаимодействовать с окружением. Такие значения reward в тестах означают, что агент вообще не выполнял задачи корректно: например, он мог падать или терять управление, и как следствие, не зарабатывал очки. Это нормальный процесс для обучающегося агента, который поначалу не знает, как действовать в среде и использует случайное управление.

Пример значений reward на первых этапах:

- -180 — это очень низкое вознаграждение, которое указывает на то, что агент ещё не обучился, его действия далеки от оптимальных, а система сильно штрафует за неправильное поведение.
- На этом этапе агент может совершать слишком резкие или неправильные маневры, тратить топливо без необходимости, а также не контролировать посадку.

К концу обучения (после 1000 эпох) значение reward уже стабильно повышается, и в финальных эпизодах мы видим значения от 260 до 312. Это гораздо более высокие результаты, что означает улучшение стратегии управления и способность адаптироваться к среде. Важно заметить, что значения в диапазоне 250–300 являются отличными для этой задачи, так как они показывают, что агент научился стабильно управлять посадкой.

Таким образом, на основе этих значений можно судить, что агент:

- На первых этапах обучения имел низкое качество управления и высокие потери.
- По мере обучения он начал демонстрировать стабильное улучшение в выполнении задачи, что подтверждается постепенным увеличением значения reward.
- На конечных этапах, с результатами порядка 300, агент мог точно управлять посадкой и всегда попадал в центр целевой зоны.

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 (260–310) 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).

Приложение. Листинги кода

В данном приложении приведены основные модули реализованного программного комплекса.

esp_lander.py

Листинг 1: esp_lander.py

```
1  import argparse
2  import os
3  import numpy as np
4  import gymnasium as gym
5  from esp import ESPPopulation
6  from visualizations import visualize_network, plot_metric
7  from utils import save_network, load_network
8
9  def record_landing_gif(network, epoch, video_dir="videos"):
10     import os
11     os.makedirs(video_dir, exist_ok=True)
12     env = gym.make("LunarLanderContinuous-v3", render_mode="rgb_array_list") #
        для новых gymnasium
13     obs, _ = env.reset()
14     done = False
15     frames = []
16     while not done:
17         frame = env.render()
18         while isinstance(frame, list) or (isinstance(frame, np.ndarray) and frame.ndim >
            3):
19             frame = frame[0]
20             frames.append(frame)
21         action = network.forward(obs)
22         obs, reward, terminated, truncated, _ = env.step(action)
23         done = terminated or truncated
24     env.close()
25     import imageio
26     gif_path = f"{video_dir}/lander_epoch_{epoch+1:04d}.gif"
27     imageio.mimsave(gif_path, [frame for frame in frames], fps=30)
28     print(f"Saved landing gif: {gif_path}")
29
30     def train(args):
31         env = gym.make('LunarLanderContinuous-v3')
32         pop = ESPPopulation(
33             input_size=8,
```

```

34 hidden_size=args.hidden_size,
35 output_size=2,
36 subpop_size=args.subpop_size
37 )
38 reward_history = []
39 loss_history = []
40 os.makedirs(args.struct_dir, exist_ok=True)
41 for epoch in range(args.epochs):
42     # Оценка по ( одному эпизоду для каждого индивида )
43     fitness = pop.evaluate(env, n_episodes=args.episodes_per_eval)
44     # Средний фитнес по всем особям прокси ( : по первой подпопуляции )
45     mean_reward = np.mean(fitness[0])
46     reward_history.append(mean_reward)
47     # В данном случае loss = -reward
48     loss = -mean_reward
49     loss_history.append(loss)
50     print(f"Epoch {epoch+1}: Mean reward {mean_reward:.2f}, loss {loss:.2f}")
51     # Визуализация структуры сети
52     net = pop.get_current_network()
53     visualize_network(net, f"{args.struct_dir}/epoch_{epoch+1:04d}.png")
54     # Эволюция
55     pop.select(fitness)
56     pop.crossover_and_mutate()
57     if (epoch + 1) % 50 == 0:
58         net = pop.get_current_network()
59         record_landing_gif(net, epoch)
60         # Сохранить веса
61         save_network(pop.get_best_network(), args.save_weights)
62         # Графики
63         plot_metric(reward_history, "Mean Reward", os.path.join(args.struct_dir,
64             "reward_curve.png"))
64         plot_metric(loss_history, "Loss", os.path.join(args.struct_dir, "loss_curve.png"))
65         env.close()
66
67     def test(args):
68         env = gym.make('LunarLanderContinuous-v3', render_mode='human')
69         net = load_network(args.load_weights)
70         for ep in range(args.test_episodes):
71             obs, _ = env.reset()
72             done = False
73             total_reward = 0
74             while not done:
75                 action = net.forward(obs)

```

```

76  obs, reward, terminated, truncated, _ = env.step(action)
77  total_reward += reward
78  done = terminated or truncated
79  print(f"Test episode {ep+1}, reward: {total_reward:.2f}")
80  env.close()
81
82  def visualize(args):
83      net = load_network(args.load_weights)
84      visualize_network(net, args.outfile)
85      print(f"Network structure saved as {args.outfile}")
86
87  if __name__ == "__main__":
88      parser = argparse.ArgumentParser(description="ESP for LunarLanderContinuous-v3")
89      parser.add_argument("--train", action="store_true", help="Train ESP")
90      parser.add_argument("--epochs", type=int, default=100)
91      parser.add_argument("--hidden_size", type=int, default=12)
92      parser.add_argument("--subpop_size", type=int, default=20)
93      parser.add_argument("--episodes_per_eval", type=int, default=1)
94      parser.add_argument("--struct_dir", type=str, default="structures")
95      parser.add_argument("--save_weights", type=str, default="model.pkl")
96      parser.add_argument("--load_weights", type=str, default="model.pkl")
97      parser.add_argument("--test", action="store_true", help="Test ESP")
98      parser.add_argument("--test_episodes", type=int, default=5)
99      parser.add_argument("--visualize_structure", action="store_true")
100     parser.add_argument("--epoch", type=int, default=1)
101     parser.add_argument("--outfile", type=str, default="network.png")
102     args = parser.parse_args()
103
104     if args.train:
105         train(args)
106     elif args.test:
107         test(args)
108     elif args.visualize_structure:
109         visualize(args)
110     else:
111         print("No mode specified. Use --train, --test or --visualize_structure.")

```

esp.py

Листинг 2: esp.py

```

1  import numpy as np
2  import gymnasium as gym
3  from network import FeedforwardNetwork

```

```

4
5 class ESPPopulation:
6     """Реализация популяции для
7         ESP: для каждого нейрона скрытого слоя - подпопуляция особи ( = вектор весов входов +
            веса выходов)
8     """
9
10    def __init__(self, input_size, hidden_size, output_size, subpop_size=20,
11                  mutation_rate=0.1, crossover_rate=0.5):
12        self.input_size = input_size
13        self.hidden_size = hidden_size
14        self.output_size = output_size
15        self.subpop_size = subpop_size
16        self.mutation_rate = mutation_rate
17        self.crossover_rate = crossover_rate
18
19        # Подпопуляция для каждого скрытого нейрона: каждый - (input_size + output_size) весов
20        self.subpopulations = [
21            [self._random_individual() for _ in range(subpop_size)]
22            for _ in range(hidden_size)
23        ]
24
25    def _random_individual(self):
26        # Вес входов скрытому нейрону + вес скрытого к каждому выводу
27        return np.random.randn(self.input_size + self.output_size) * 0.1
28
29    def assemble_network(self, hidden_indices):
30        """Собрать сеть
31            : по одному представителю из каждой подпопуляции (hidden_indices
32              - индекс в каждой подпопуляции)
33        """
34        # Becainput-hidden
35        w_ih = np.stack([self.subpopulations[i][hidden_indices[i]][:self.input_size] for
36                          i in range(self.hidden_size)])
37        # Becahidden-output
38        w_ho = np.stack([self.subpopulations[i][hidden_indices[i]][self.input_size:] for
39                          i in range(self.hidden_size)]).T
40        # w_ih: [hidden_size, input_size]
41        # w_ho: [output_size, hidden_size]
42        return FeedforwardNetwork(self.input_size, self.hidden_size, self.output_size,
43                                  w_ih, w_ho)
44
45    def evaluate(self, env, n_episodes=1, render=False):

```



```

41     """Оценить всех особей в подпопуляциях
42
43     """
44     fitness = [np.zeros(self.subpop_size) for _ in range(self.hidden_size)]
45
46     for trial in range(self.subpop_size):
47         # Для каждого hidden нейрона - выбираем trial индивида-
48         hidden_indices = [trial] * self.hidden_size
49         network = self.assemble_network(hidden_indices)
50         rewards = []
51         for ep in range(n_episodes):
52             obs, _ = env.reset()
53             total_reward = 0
54             done = False
55             while not done:
56                 action = network.forward(obs)
57                 obs, reward, terminated, truncated, _ = env.step(action)
58                 total_reward += reward
59                 done = terminated or truncated
60             if render:
61                 env.render()
62             rewards.append(total_reward)
63         for i in range(self.hidden_size):
64             fitness[i][trial] = np.mean(rewards)
65     return fitness
66
67     def select(self, fitness, tournament_k=3):
68         """Турнирный отбор для каждой подпопуляции
69
70         """
71         new_subpops = []
72         for subpop, fit in zip(self.subpopulations, fitness):
73             idxs = np.arange(self.subpop_size)
74             selected = []
75             for _ in range(self.subpop_size):
76                 tournament = np.random.choice(idxs, tournament_k, replace=False)
77                 best = tournament[np.argmax(fit[tournament])]
78                 selected.append(subpop[best].copy())
79             new_subpops.append(selected)
80         self.subpopulations = new_subpops
81
82     def crossover_and_mutate(self):
83         """Одноточечный кроссовер и мутация по Гауссу

```

```

84
85 """
86 for s, subpop in enumerate(self.subpopulations):
87     # Кроссовер
88     for i in range(0, self.subpop_size, 2):
89         if np.random.rand() < self.crossover_rate:
90             a, b = subpop[i], subpop[(i+1) % self.subpop_size]
91             point = np.random.randint(1, len(a))
92             child1 = np.concatenate([a[:point], b[point:]])
93             child2 = np.concatenate([b[:point], a[point:]])
94             subpop[i] = child1
95             subpop[(i+1) % self.subpop_size] = child2
96     # Мутация
97     for i in range(self.subpop_size):
98         if np.random.rand() < self.mutation_rate:
99             subpop[i] += np.random.randn(*subpop[i].shape) * 0.1
100
101 def get_best_network(self):
102     """Сеть излучшихособей
103
104     """
105     best_indices = [np.argmax([np.random.rand() for _ in subpop]) for subpop in
106                     self.subpopulations]
107     return self.assemble_network(best_indices)
108
109 def get_current_network(self):
110     """Сеть из первыхособей для
111         ( визуализации)
112     """
113     indices = [0 for _ in range(self.hidden_size)]
114     return self.assemble_network(indices)

```

network.py

Листинг 3: network.py

```

1 import numpy as np
2
3 class FeedforwardNetwork:
4     """
5     # Однослойная прямораспространённая сеть (input -> hidden -> output)
6     """
7
8     def __init__(self, input_size, hidden_size, output_size,

```

```

9 weights_input_hidden=None, weights_hidden_output=None):
10 self.input_size = input_size
11 self.hidden_size = hidden_size
12 self.output_size = output_size
13
14 # Инициализация весов если не переданы
15 if weights_input_hidden is None:
16 self.weights_input_hidden = np.random.randn(hidden_size, input_size) * 0.1
17 else:
18 self.weights_input_hidden = weights_input_hidden
19
20 if weights_hidden_output is None:
21 self.weights_hidden_output = np.random.randn(output_size, hidden_size) * 0.1
22 else:
23 self.weights_hidden_output = weights_hidden_output
24
25 def forward(self, x):
26 """Прямое распространение
27
28 """
29 h = np.tanh(self.weights_input_hidden @ x)
30 o = np.tanh(self.weights_hidden_output @ h)
31 return o
32
33 def get_weights(self):
34 return {
35     "input_size": self.input_size,
36     "hidden_size": self.hidden_size,
37     "output_size": self.output_size,
38     "weights_input_hidden": self.weights_input_hidden,
39     "weights_hidden_output": self.weights_hidden_output
40 }
41
42 @staticmethod
43 def from_weights(data):
44 return FeedforwardNetwork(
45     data['input_size'],
46     data['hidden_size'],
47     data['output_size'],
48     data['weights_input_hidden'],
49     data['weights_hidden_output']
50 )

```

utils.py

Листинг 4: utils.py

```
1  import pickle
2  import os
3
4  def save_network(network, filename):
5      data = network.get_weights()
6      with open(filename, "wb") as f:
7          pickle.dump(data, f)
8
9  def load_network(filename):
10     import network
11     with open(filename, "rb") as f:
12         data = pickle.load(f)
13     return network.FeedforwardNetwork.from_weights(data)
```

visualizations.py

Листинг 5: visualizations.py

```
1  import matplotlib.pyplot as plt
2  import seaborn as sns
3  import numpy as np
4  import matplotlib.cm as cm
5
6  def visualize_network(network, filename):
7      """Визуализация структуры сети
8         (input-hidden-output) соединениями.
9      """
10     plt.figure(figsize=(8, 6))
11     in_x, in_y = [0]*network.input_size, np.linspace(0, 1, network.input_size)
12     hid_x, hid_y = [0.5]*network.hidden_size, np.linspace(0, 1, network.hidden_size)
13     out_x, out_y = [1]*network.output_size, np.linspace(0.3, 0.7, network.output_size)
14
15     colormap = cm.get_cmap('coolwarm')
16
17     for i, (x0, y0) in enumerate(zip(in_x, in_y)):
18         for j, (x1, y1) in enumerate(zip(hid_x, hid_y)):
19             w = network.weights_input_hidden[j, i]
20             color = colormap(abs(w))
21             plt.plot([x0, x1], [y0, y1], color=color, alpha=0.7, lw=abs(w) * 3 + 1)
22
```

```

23 for i, (x0, y0) in enumerate(zip(hid_x, hid_y)):
24     for j, (x1, y1) in enumerate(zip(out_x, out_y)):
25         w = network.weights_hidden_output[j, i]
26         color = colormap(abs(w))
27         plt.plot([x0, x1], [y0, y1], color=color, alpha=0.7, lw=abs(w) * 3 + 1)
28
29 plt.scatter(in_x, in_y, s=300, label='Input', color='blue', edgecolors='black',
30             linewidths=1)
31 plt.scatter(hid_x, hid_y, s=300, label='Hidden', color='orange',
32             edgecolors='black', linewidths=1)
33 plt.scatter(out_x, out_y, s=300, label='Output', color='green',
34             edgecolors='black', linewidths=1)
35
36 for i, txt in enumerate(range(network.input_size)):
37     plt.text(in_x[i] - 0.05, in_y[i], str(txt), fontsize=10, ha='center',
38             color='white')
39
40 for i, txt in enumerate(range(network.hidden_size)):
41     plt.text(hid_x[i] + 0.05, hid_y[i], str(txt), fontsize=10, ha='center',
42             color='black')
43
44 for i, txt in enumerate(range(network.output_size)):
45     plt.text(out_x[i] + 0.05, out_y[i], str(txt), fontsize=10, ha='center',
46             color='black')
47
48 plt.axis('off')
49 plt.title('Network Structure with Enhanced Visualization')
50 plt.tight_layout()
51
52 plt.savefig(filename)
53 plt.close()
54
55 def plot_metric(metric_history, ylabel, filename):
56     plt.figure()
57     sns.lineplot(x=np.arange(len(metric_history)), y=metric_history)
58     plt.xlabel("Epoch")
59     plt.ylabel(ylabel)
60     plt.title(f"{ylabel} over epochs")
61     plt.tight_layout()
62     plt.savefig(filename)
63     plt.close()

```

`model_example.pkl`

Листинг 6: `model_example.pkl`

```
1  {  
2    "input_size": 8,  
3    "hidden_size": 12,  
4    "output_size": 2,  
5    "weights_input_hidden": numpy.ndarray,  
6    "weights_hidden_output": numpy.ndarray  
7  }
```