

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 1

на тему: Реализация алгоритма нейроэволюции ESP для непрерывного контроля в
среды Bipedal Walker

Выполнил:	студент гр. 8BM42 Агарков Д.А.	09.06.2025
Проверил:	к.т.н., Доцент ОИТ ИШИТР Григорьев Д.С.	09.06.2025

Содержание

1	Введение	3
2	Описание используемого алгоритма	4
2.1	Принципы работы NEAT (NeuroEvolution of Augmenting Topologies)	4
2.2	Структура сети	4
2.3	Логика разбиения на подпопуляции, эволюция на уровне нейрона	5
2.4	Этапы алгоритма NEAT	6
3	Этапы имплементации	9
3.1	Модульная структура кода	9
3.2	Основные этапы реализации	9
4	Целевые метрики	12
4.1	Формальное определение метрики	12
4.2	Целевая метрика в контексте NEAT	12
4.3	Параметры для оценки агента	12
4.4	Реализация метрики в коде	13
4.5	Интерпретация метрики	13
4.6	Вспомогательная метрика: Loss	13
5	Визуализация	15
5.1	Визуализация структуры нейронной сети	15
5.2	График динамики метрики	15
5.3	Генерация gif-визуализаций	16
5.4	Сохранение и загрузка состояния	16
5.5	Видеодемонстрация работы агента	17
6	Развертывание, тестирование и анализ результатов	20
6.1	Структура проекта	20
6.2	Основной код: <code>main.py</code>	20
6.3	Визуализация: <code>visualize.py</code>	21
6.4	Описание работы алгоритма	22
6.5	Сохранение и загрузка состояния	22
6.6	Обучение модели	23
6.6.1	Видеодемонстрация работы модели	24
7	Заключение	25

1 Введение

Нейроэволюция — это подход к обучению нейронных сетей, основанный на принципах эволюционного отбора. В отличие от традиционных методов, таких как градиентное обучение или обратное распространение ошибки, нейроэволюционные алгоритмы не требуют наличия производных целевой функции и способны эффективно работать в условиях негладких, стохастических и дискретных сред. Это делает нейроэволюцию особенно перспективной в задачах, где градиентные методы не применимы или дают нестабильные результаты.

Одним из наиболее известных и мощных алгоритмов нейроэволюции является **NEAT** (NeuroEvolution of Augmenting Topologies), предложенный Кеном Стэнли. Его ключевая особенность — это одновременная эволюция как весов, так и структуры нейронной сети. Алгоритм начинает обучение с минимально возможной топологии (обычно — только входной и выходной слои) и постепенно усложняет сеть по мере необходимости, добавляя скрытые нейроны и новые связи. Такой подход позволяет избежать переобучения на ранних стадиях и эффективно исследовать пространство решений.

NEAT реализует ряд уникальных механизмов:

- **Исторические метки (innovation numbers)** — позволяют отслеживать происхождение связей и синхронизировать гены при скрещивании сетей с разной архитектурой;
- **Мутации структуры** — добавление новых нейронов и связей во время эволюции;
- **Специализация (speciation)** — деление популяции на виды, конкуренция происходит внутри вида, а не между всеми особями сразу, что защищает потенциально полезные новшества от преждевременного отбора.

В данной работе рассматривается применение алгоритма NEAT к задаче непрерывного управления в симулированной среде **BipedalWalker-v3** из библиотеки **Gymnasium**. Это среда с непрерывным пространством действий, в которой агент должен научиться координировать движение ног для устойчивой ходьбы по неровной поверхности. Данная задача требует от алгоритма способности строить эффективные стратегии управления, которые опираются как на сенсорные данные, так и на внутреннюю динамику сети.

Цель работы — реализовать алгоритм NEAT и обучить нейронную сеть с развивающейся архитектурой для эффективного управления агентом в среде **BipedalWalker-v3**, выполнив:

- реализацию и описание механизмов NEAT (кодирования, мутации, скрещивания, специализации);
- визуализацию структуры сети на разных этапах обучения;
- анализ сходимости по метрикам среды (например, reward);
- сохранение и загрузку состояния сети для воспроизводимости.

2 Описание используемого алгоритма

2.1 Принципы работы NEAT (NeuroEvolution of Augmenting Topologies)

Алгоритм NEAT (NeuroEvolution of Augmenting Topologies), предложенный Кеном Стэнли, представляет собой метод нейроэволюционного обучения, в котором происходит одновременная эволюция как весов, так и архитектуры нейронной сети. В отличие от традиционных генетических алгоритмов с фиксированной структурой генома, NEAT динамически наращивает сложность сети, начиная с простой топологии и постепенно добавляя новые нейроны и связи по мере необходимости.

Основные механизмы алгоритма NEAT:

- **Геномное представление:** нейронная сеть кодируется списком генов-связей с указанием входного/выходного нейрона, веса, статуса активности и номера инновации;
- **Исторические метки (innovation numbers):** каждая новая структура связи получает уникальный идентификатор, что позволяет корректно проводить скрещивание между сетями с различной топологией;
- **Эволюционное наращивание:** в процессе мутации возможны не только изменения весов, но и добавление новых нейронов или связей;
- **Специализация (speciation):** популяция особей разбивается на виды, внутри которых происходит локальная конкуренция, что предотвращает преждевременное вымирание редких, но перспективных мутаций;
- **Скрещивание с выравниванием генов:** потомки формируются из совпадающих по инновациям генов, а также из несовпадающих генов наиболее приспособленного родителя.

2.2 Структура сети

В данной реализации алгоритма NEAT используется нейронная сеть с архитектурой *feedforward*, где информация передается от входных нейронов через скрытые нейроны к выходным, без рекуррентных связей. Эта структура подходит для задач, где необходимо моделировать зависимости между входами и выходами в динамическом и непрерывном контексте, как в случае с *BipedalWalker-v3*.

Нейронная сеть состоит из нескольких ключевых компонентов:

- **Входной слой:** состоит из нейронов, количество которых соответствует размерности входных данных. Для задачи *BipedalWalker-v3* это 24 признака, описывающих состояние агента и окружающей среды.

- **Скрытые слои:** количество скрытых нейронов не ограничено и может изменяться в ходе эволюции. В начале сети скрытые нейроны отсутствуют, и они добавляются только по мере необходимости в процессе мутации.
- **Выходной слой:** состоит из нейронов, количество которых соответствует количеству управляющих сигналов. В задаче управления BipedalWalker-v3 это 4 выхода, которые контролируют действия агента (например, управление движением ног).

Особенности структуры:

- **Отсутствие рекуррентных связей:** структура сети предполагает только прямое распространение информации от входных нейронов к выходным. Это обеспечивает простоту архитектуры и быстрое обучение на начальных этапах.
- **Динамическое расширение сети:** количество нейронов и связей в сети может изменяться во время эволюции. Изначально сеть имеет минимальную структуру, состоящую только из входных и выходных нейронов, но по мере эволюции могут добавляться новые скрытые нейроны и связи между ними.
- **Мутации структуры:** при мутации сети могут быть добавлены новые нейроны и связи. Эти мутации происходят случайным образом с определённой вероятностью, что позволяет гибко изменять архитектуру сети в ответ на изменяющиеся требования задачи.

2.3 Логика разбиения на подпопуляции, эволюция на уровне нейрона

В алгоритме NEAT ключевым моментом является разделение популяции на подпопуляции, каждая из которых отвечает за эволюцию нейронов скрытого слоя. Это разделение способствует специализации нейронов и позволяет каждому нейрону оптимизировать свои параметры (веса входных и выходных связей) независимо от остальных, что повышает эффективность алгоритма.

- **Подпопуляция для каждого нейрона:** Каждому нейрону скрытого слоя соответствует своя подпопуляция. Каждая особь в подпопуляции — это набор параметров для нейрона, включающий веса входных связей, веса выходных связей, а также, если требуется, смещения. Таким образом, каждый нейрон представляется как самостоятельная эволюционная единица, которая может изменяться и развиваться независимо от других нейронов.
- **Формирование команд для оценки:** Для оценки приспособленности нейронов создаются *команды*, которые представляют собой нейронные сети, сформированные из случайных представителей всех подпопуляций. Это означает, что каждый агент оценивается не по отдельности, а как часть группы, где его нейрон взаимодействует с нейронами из других подпопуляций. Это помогает улучшить качество обучения и способствует более разнообразным и сильным решениям.

- **Оценка приспособленности нейронов:** Оценка нейрона в подпопуляции происходит на основе кумулятивного фитнеса, то есть его приспособленность определяется как сумма результатов всех команд, в которых этот нейрон принимал участие. Каждая особь должна быть использована в сети минимум определённое количество раз, например, 10 раз. Это обеспечивает более надёжную оценку его способности вносить вклад в решение задачи.
- **Специализация нейронов:** Благодаря эволюции на уровне нейрона, каждый нейрон постепенно становится специализированным, решая определённые подзадачи, такие как обработка специфических частей информации или управление отдельными аспектами задачи. Это приводит к улучшению общей производительности сети, так как каждый нейрон адаптируется и оптимизируется под свою функцию. Разделение популяции на подпопуляции также предотвращает переобучение, обеспечивая разнообразие в возможных решениях и архитектурах сети.

2.4 Этапы алгоритма NEAT

Алгоритм NEAT состоит из нескольких ключевых этапов, которые помогают эффективно эволюционировать нейронные сети. В каждом поколении популяция агентов проходит через этапы оценки, мутации, скрещивания и специализации, что способствует улучшению качества решения задачи и созданию устойчивых стратегий управления:

1. Инициализация популяции:

- Создаётся начальная популяция агентов с минимальными структурами нейронных сетей. На этом этапе каждый агент состоит только из входных и выходных нейронов, и сети ещё не имеют скрытых слоёв или дополнительных связей.
- Каждому агенту присваиваются случайные веса связей между нейронами, и они начинают эволюционировать с равными шансами на успех.

2. Оценка приспособленности (fitness evaluation):

- Каждый агент оценивается в среде. Для этого выполняется несколько эпизодов (например, 10), в которых агент взаимодействует с окружением, выполняет действия, и на основе этих действий получает вознаграждения.
- Приспособленность агента определяется суммой полученных вознаграждений в ходе всех эпизодов. Это позволяет оценить, насколько хорошо агент решает поставленную задачу.

3. Специализация (speciation):

- После того как все агенты были оценены, популяция разделяется на виды в зависимости от схожести их геномов (структуры нейронных сетей). Это дела-

ется с помощью расчёта расстояния между генами агентов, которое измеряет различия в топологии и весах их сетей.

- Виды помогают сохранять разнообразие в популяции и предотвращают быструю конкуренцию между агентами с радикально разными структурами сетей, обеспечивая условия для развития новых, более эффективных сетей.

4. Скрещивание (crossover):

- Для каждого вида выбираются родители, которые будут участвовать в скрещивании. При этом выбираются агенты с наибольшими значениями приспособленности.
- На основе генов выбранных родителей создаются потомки. При скрещивании гены с одинаковыми номерами инноваций могут быть объединены случайным образом, в зависимости от вероятности.
- Новые агенты создаются путём комбинирования весов и структурных изменений от родителей, что позволяет генерировать новые вариации, способные решать задачу лучше.

5. Мутация (mutation):

- Мутация является ключевым этапом, позволяющим добавлять новые нейроны и связи в сети. Она происходит случайным образом и может затрагивать как веса существующих связей, так и структуру сети.
- Возможные мутации включают: добавление новых нейронов (что увеличивает сложность сети), добавление новых связей между нейронами, изменение веса существующих связей.
- Вероятность и тип мутации зависят от конфигурации алгоритма (например, вероятность изменения весов или добавления новых связей).

6. Отбор (selection):

- После скрещивания и мутации популяция обновляется, и остаются только лучшие особи (с наибольшим фитнесом). Это позволяет сохранять в популяции наиболее успешных агентов.
- Отбор может быть как жестким (отбираются только лучшие агенты), так и более мягким, с возможностью оставить определённое количество менее приспособленных агентов для дальнейшего улучшения.

7. Повторение:

- Шаги 2-5 повторяются несколько раз (в течение заданного числа поколений или до достижения заданного уровня производительности).

- По мере повторения этих шагов, сеть будет усложняться и оптимизироваться, улучшая свои способности к решению задачи.

8. Сохранение состояния (optional):

- Состояние каждого агента и его сети может быть сохранено в файл (например, с помощью сериализации в формат pickle). Это позволяет продолжить обучение с того места, где оно было остановлено, или проанализировать прогресс обучения.

Таким образом, алгоритм NEAT эволюционирует нейронные сети, начиная с простых структур и постепенно добавляя новые нейроны и связи. Благодаря этому методу, агент может адаптировать свою структуру для эффективного решения задачи, постепенно улучшая свою архитектуру и веса на основе опыта в среде.

3 Этапы имплементации

Реализация алгоритма NEAT для задачи управления в среде `BipedalWalker-v3` была выполнена на языке Python с использованием библиотеки `gymnasium` для симуляции среды и `numpy` для работы с числовыми вычислениями. Алгоритм реализован с нуля, без использования сторонних библиотек или реализаций нейроэволюции, что позволило гибко настроить и адаптировать алгоритм под конкретные требования задачи.

3.1 Модульная структура кода

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

- **NEATConfig:** класс конфигурации, который содержит все основные параметры алгоритма NEAT, такие как размер популяции, вероятность мутаций, параметры мутации весов и структуры сети.
- **Gene:** класс, представляющий отдельную связь в нейронной сети. Каждый ген хранит информацию о связи между двумя нейронами (входным и выходным), весе этой связи и её статусе (включена/выключена).
- **NeuralNetwork:** класс нейронной сети, который управляет созданием сети, её мутацией, а также вычислением выходных данных сети на основе входных данных.
- **Speciation:** класс, реализующий разделение популяции на виды на основе схожести геномов агентов. Он помогает поддерживать разнообразие в популяции и предотвращать слишком быстрый отбор сильнейших агентов.
- **Agent:** класс, представляющий агента в популяции, который взаимодействует с окружающей средой и оценивается на основе полученного вознаграждения.
- **NEAT:** основной класс, реализующий алгоритм NEAT. Он управляет популяцией агентов, координирует процессы мутации, скрещивания и специализации, а также проводит эволюцию популяции.

3.2 Основные этапы реализации

Инициализация популяции и параметров сети: На первом этапе создаётся начальная популяция агентов с минимальными структурами нейронных сетей. Каждый агент состоит из входных и выходных нейронов, инициализированных случайными весами. Веса для связей между нейронами генерируются случайным образом, а структура сети может быть дополнительно изменена в процессе эволюции. На этом этапе сети имеют минимальное количество нейронов, и сложность сети будет увеличиваться по мере добавления новых нейронов и связей в процессе обучения.

Реализация в коде:

```

1 self.agents = [Agent(self.env.observation_space.shape[0],
                        self.env.action_space.shape[0]) for _ in range(self.population_size)]

```

Оценка приспособленности (fitness evaluation): Каждый агент оценивается в среде. Для этого агент взаимодействует с окружением, выполняет действия, и на основе этих действий получает вознаграждения. Приспособленность агента определяется как сумма полученных вознаграждений в ходе всех эпизодов. Агенты выполняют несколько эпизодов в среде, и их фитнес вычисляется на основе их производительности в этих эпизодах.

Ключевой фрагмент кода:

```

1 for agent in self.agents:
2     agent.fitness = agent.evaluate(self.env) # Оценка приспособленности

```

Специализация популяции (speciation): После того как все агенты были оценены, популяция разделяется на виды с помощью класса `Speciation`. Это разделение помогает сохранить разнообразие популяции и предотвращает преждевременное вытеснение агентов с нестандартной архитектурой. Виды формируются на основе схожести геномов агентов, что способствует эффективной эволюции.

Реализация в коде:

```

1 self.speciation.assign_species(self.agents) # Разбиение популяции на виды

```

Скрещивание (crossover): После специализации и отбора лучших агентов производится скрещивание. Для каждого вида выбираются два лучших родителя, и на основе их генов создаются потомки. При скрещивании выбираются гены с одинаковыми номерами инноваций, а также случайным образом выбираются гены от родителей с наибольшей приспособленностью.

Реализация в коде:

```

1 child = crossover(parent1, parent2) # Создание потомка через скрещивание

```

Мутация (mutation): После скрещивания каждый агент подвергается мутации. Мутация может быть как изменением весов существующих связей, так и добавлением новых нейронов и связей. Эти изменения помогают улучшать структуру сети и адаптировать её к новым задачам.

Реализация в коде:

```

1 for agent in new_agents:
2     agent.network.mutate(self.config) # Мутация сети

```

Отбор (selection): После мутации и скрещивания популяция обновляется, и остаются только лучшие агенты с наибольшим фитнесом. Эти агенты составляют новое поколение, которое будет использоваться в следующем цикле эволюции.

Реализация в коде:

```
1     self.agents = new_agents # Обновление популяции
```

Повторение (iteration): Этапы 2-6 повторяются для каждого поколения. Каждый цикл включает оценку, специализацию, скрещивание, мутацию и отбор, пока не будет достигнут максимальный уровень приспособленности или заданное количество поколений.

Реализация в коде:

```
1     for generation in range(max_generations):  
2         self.evolve() # Эволюция популяции
```

Таким образом, алгоритм NEAT эволюционирует нейронные сети, начиная с простых структур и постепенно добавляя новые нейроны и связи. Это позволяет сети адаптироваться и решать сложные задачи с динамическими изменениями, как в задаче управления агентом в среде *BipedalWalker-v3*.

4 Целевые метрики

4.1 Формальное определение метрики

Основной целевой метрикой, используемой для оценки эффективности агента в алгоритме NEAT, является **среднее суммарное вознаграждение за эпизод** (average episodic reward), которое вычисляется 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.

Эта метрика помогает измерить, насколько хорошо агент справляется с задачей в процессе обучения, и позволяет отслеживать прогресс по мере эволюции популяции.

4.2 Целевая метрика в контексте NEAT

В контексте NEAT вознаграждение агенту дается за его действия в среде. Агент, который обучается в такой среде, как `BipedalWalker-v3`, получает вознаграждения, зависящие от нескольких факторов: стабильности, эффективности и скорости выполнения задачи.

Целевая метрика, которая определяется через суммарное вознаграждение, может быть расширена для оценки успешности обучения с учётом таких параметров как:

- **Эффективность использования сети:** метрика учитывает, насколько хорошо нейронная сеть, эволюционирующая по мере популяции, может принимать решения для выполнения задачи.
- **Комплексность структуры сети:** увеличение сложности сети через добавление новых нейронов и связей будет сопровождаться улучшением целевой метрики, если добавление новых элементов помогает решать задачу.
- **Скорость обучения:** как быстро агент достигает хороших значений вознаграждения. Это также является важным аспектом метрики.

4.3 Параметры для оценки агента

При использовании NEAT для решения задач, связанных с непрерывным контролем, например в `BipedalWalker-v3`, ключевые аспекты для оценки агентов включают:

- **Стабильность** (насколько устойчиво агент может поддерживать равновесие),

- **Эффективность управления движением** (насколько хорошо агент управляет своим движением),
- **Оптимизация структуры нейронной сети** (оценивается по эффективности сети с учётом её структуры).

4.4 Реализация метрики в коде

Для каждого агента рассчитывается суммарное вознаграждение за несколько эпизодов в среде. Это вознаграждение помогает оценить, насколько эффективно агент решает поставленную задачу.

Пример расчёта метрики в коде:

```
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)
```

Этот фрагмент кода вычисляет средний фитнес на основе результатов эпизодов и сохраняет его для дальнейшего анализа. Средний фитнес служит основным индикатором эффективности работы алгоритма NEAT.

4.5 Интерпретация метрики

Интерпретация метрики R_{avg} для задачи в среде `BipedalWalker-v3` выглядит следующим образом:

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

Эти пороговые значения помогают оценить, насколько успешно агент решает задачу по мере эволюции. Также, они позволяют сравнивать производительность различных агентов и наглядно следить за процессом обучения.

4.6 Вспомогательная метрика: Loss

Для удобства визуализации процесса обучения и оптимизации используется дополнительная метрика — **Loss**, которая является преобразованной метрикой вознаграждения:

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

- Минимизация **loss** эквивалентна максимизации R_{avg} .

- Позволяет эффективно отслеживать сходимость модели и применять стандартные методы визуализации для алгоритмов оптимизации.
- Упрощает сравнение с методами, использующими градиентный спуск, где минимизация `loss` является основной целью.

Пример вычисления метрики `loss` в коде:

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

Таким образом, метрика R_{avg} является основным индикатором эффективности нейронной сети, которая оценивает качество выполнения задачи с учетом всех факторов, таких как стабильность, эффективность и сложность сети.

5 Визуализация

Визуализация — важная часть анализа и отладки процесса обучения, особенно при использовании эволюционных алгоритмов, таких как NEAT. Она позволяет отслеживать изменения в архитектуре сети, динамику метрик производительности, а также успехи агента в среде. В этой работе для визуализации используются различные методы, включая отображение структуры нейронной сети, графики динамики метрик и анимации, показывающие процесс обучения агента.

5.1 Визуализация структуры нейронной сети

Одним из важных аспектов алгоритма NEAT является эволюция топологии нейронной сети. Сеть развивается от минимальной структуры (входной и выходной нейроны) до более сложной с добавлением новых нейронов и связей. Для отслеживания изменений в архитектуре сети была реализована визуализация её топологии на каждом этапе эволюции.

5.2 График динамики метрики

Для анализа прогресса обучения создается график, показывающий динамику ключевой метрики, такой как лучшее вознаграждение (fitness). Этот график позволяет отслеживать, как изменяется производительность модели по мере её эволюции, а также насколько быстро она обучается и адаптируется к среде.

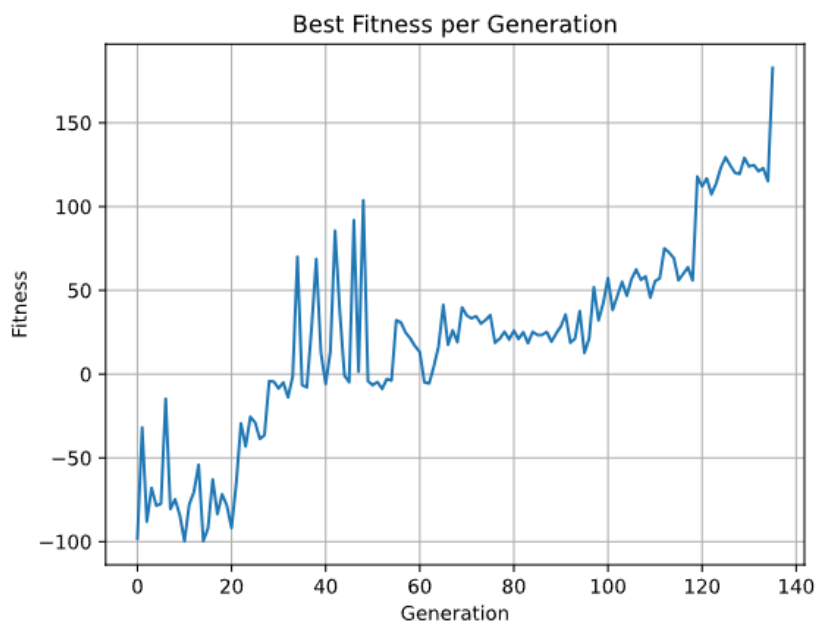


Рис. 1: Best fitness: постепенное увеличение агента указывает на улучшение поведения агента и успешную эволюцию популяции.

Пример кода для создания графиков динамики метрик:

```

1 import matplotlib.pyplot as plt
2
3 def plot_metrics(fitness_history, loss_history):
4     plt.figure(figsize=(10,5))
5
6     # График для среднего фитнеса
7     plt.subplot(1, 2, 1)
8     plt.plot(fitness_history, label='Best Fitness')
9     plt.xlabel('Generation')
10    plt.ylabel('Fitness')
11    plt.title('Best Fitness per Generation')
12    plt.show()

```

Этот код строит:

- График изменения фитнеса на протяжении поколений, который помогает отслеживать, как улучшается производительность сети.

Этот график полезен для визуализации того, как процесс обучения развивается на протяжении нескольких поколений, а также для понимания, насколько эффективно происходит оптимизация сети.

5.3 Генерация gif-визуализаций

Для лучшего представления процесса обучения можно создавать gif-анимированные изображения, которые показывают, как агент взаимодействует со средой и как его поведение улучшается с каждым поколением. Эти анимации полезны для наглядного демонстрирования улучшений в стратегии агента.

Пример кода для генерации gif:

```

1 import imageio
2
3 def create_gif(frames, gif_name):
4     with imageio.get_writer(gif_name, mode='I') as writer:
5         for frame in frames:
6             writer.append_data(frame)

```

Этот код позволяет создать gif-анимированное изображение из последовательности кадров, например, показывающих агента в различных состояниях во время его обучения в среде. Это является мощным инструментом для визуализации процесса обучения и эволюции агента.

5.4 Сохранение и загрузка состояния

Для воспроизводимости экспериментов и анализа прогресса можно сохранять состояние нейронной сети в файл. Это позволяет продолжить обучение с того места, где оно было остановлено, или использовать ранее обученную сеть для тестирования.

Пример кода для сохранения и загрузки состояния:

```
1 import pickle
2
3 def save_network(network, filename):
4     with open(filename, 'wb') as file:
5         pickle.dump(network, file)
6
7 def load_network(filename):
8     with open(filename, 'rb') as file:
9         return pickle.load(file)
```

Этот код позволяет сохранить состояние сети и загрузить его в любой момент, обеспечивая возможность продолжить обучение без потери данных, а также проанализировать прогресс, достигнутый на различных этапах.

Визуализация является важной частью алгоритма NEAT, позволяя отслеживать изменения в структуре сети и динамику метрик, таких как фитнес и потери, на протяжении процесса эволюции. Графики и анимации помогают глубже понять, как работает алгоритм и как он оптимизирует поведение агента в среде. Генерация gif-анимаций и сохранение состояния сети делают анализ и воспроизведение экспериментов более удобным.

5.5 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 походки агента в три ключевых этапа обучения: на 10-й, 50-й и 150-й эпохах. Каждое изображение отражает прогресс в поведении агента при выполнении задачи посадки.

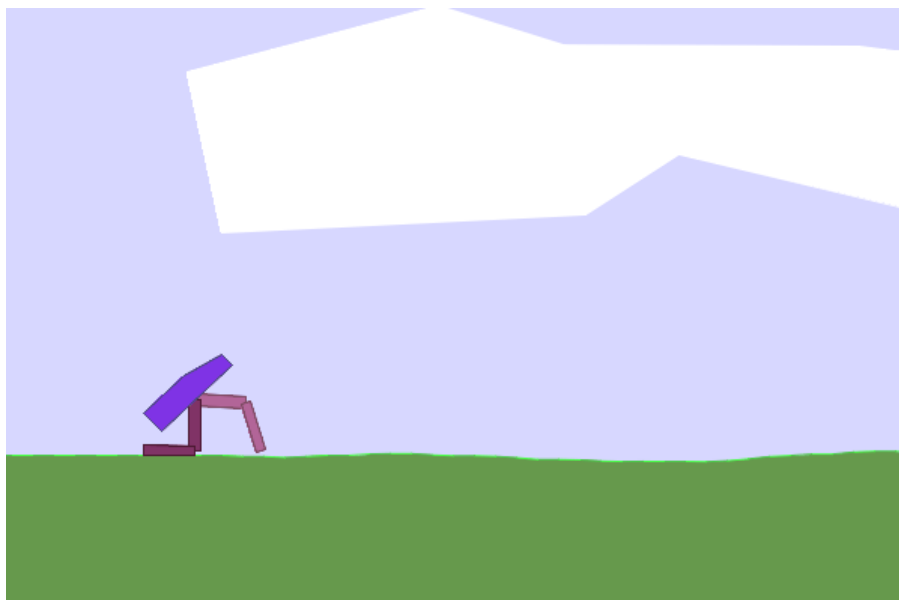


Рис. 2: Походка на 10-й эпохе. Агент практически не встает на ноги. Он с трудом начинает движение лишь к концу, но уже видны признаки неэффективного контроля, что приводит к сильно отклонённой траектории.

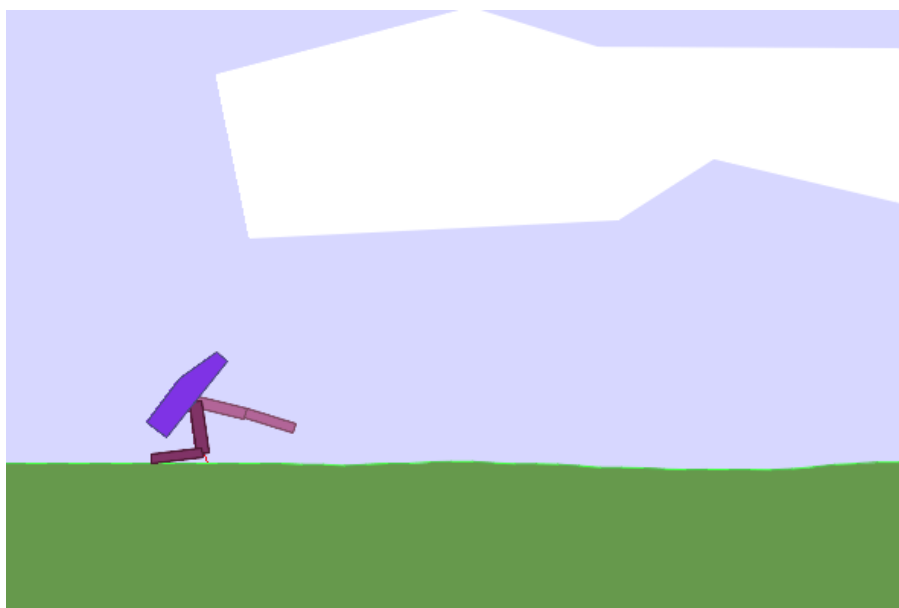


Рис. 3: Походка на 50-й эпохе. Обе ноги заметно начинают движение. Агент корректно поднимает ногу, чтобы двигаться, но всё ещё имеет проблемы с ходьбой.

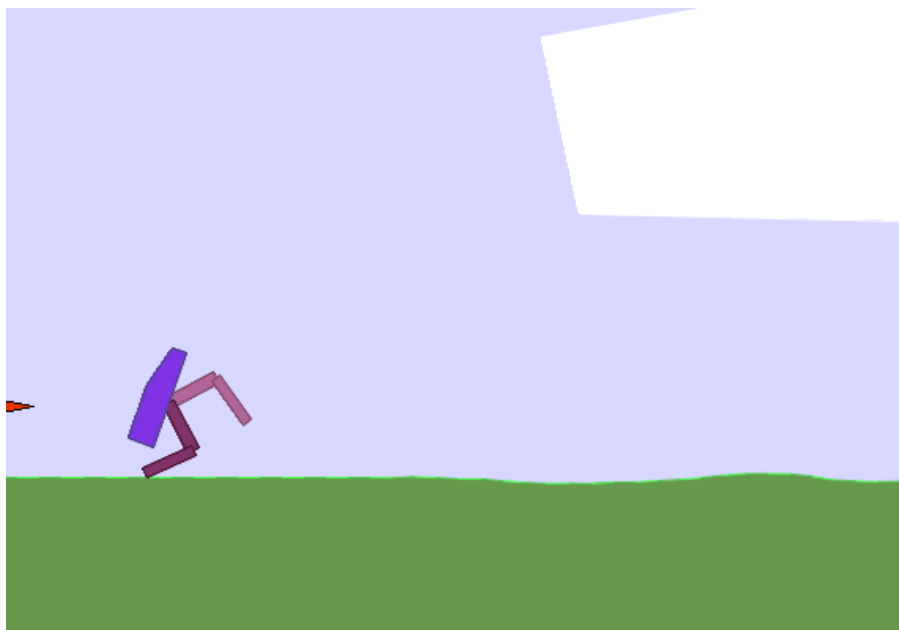


Рис. 4: Походка на 150-й эпохе. Агент успешно осваивает походку. Он аккуратно и уверенно шагает, переворотов нет.

Эти скриншоты иллюстрируют прогресс, который агент демонстрирует в ходе обучения: от неэффективного движения ног и нестабильной траектории в начале, до аккуратной и уверенной походки. 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 VS Code. 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:

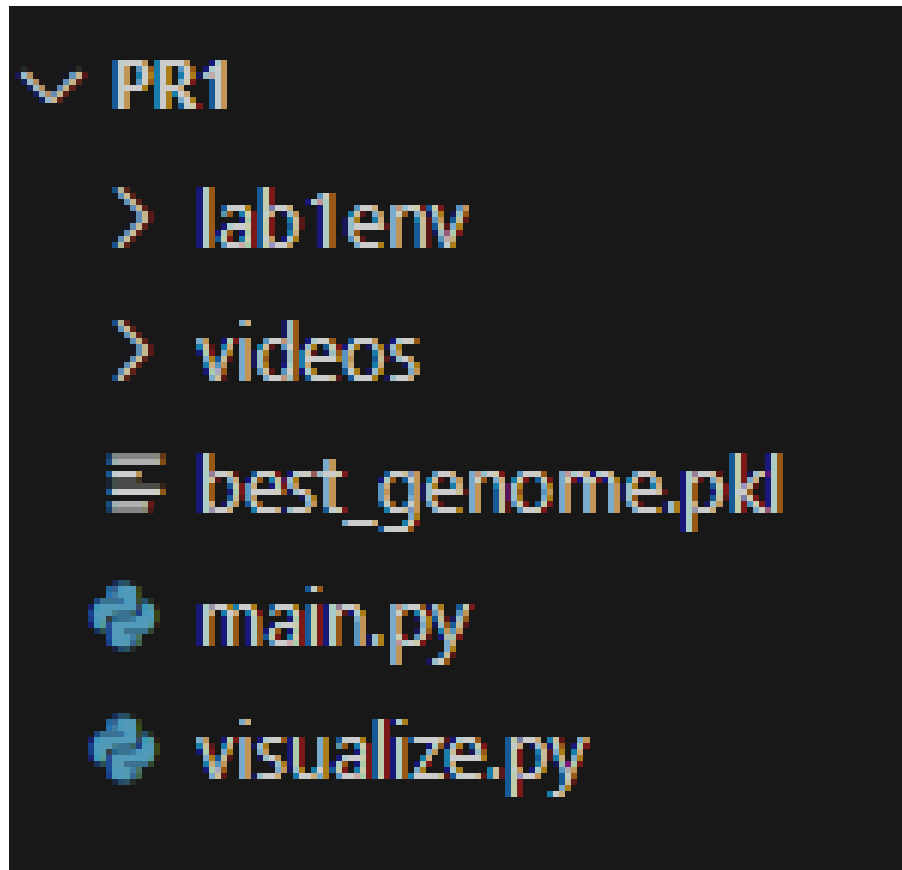


Рис. 5: Структура проекта в среде разработки VS Code

6.2 Основной код: `main.py`

`main.py` представляет собой основной модуль, который содержит реализацию алгоритма NEAT. Включает в себя классы для создания нейронных сетей, мутаций, скрещивания, специализации популяции и коэволюции.

- **NEATConfig:** класс конфигурации, который задает все параметры для эволюции, включая размер популяции, порог для разделения на виды, параметры мутаций и рекомбинации.
- **Gene:** класс для представления связи между нейронами, включая номер инновации, веса связи и статус (включена или выключена связь).

- **NeuralNetwork:** класс для нейронной сети, который управляет созданием сети, её мутацией, а также расчётом выходных данных на основе входных данных.
- **Speciation:** класс для разделения популяции на виды. Он помогает избежать преждевременного вытеснения агентов с новыми структурами.
- **Agent:** класс для представления агента, который взаимодействует с окружающей средой и оценивается по полученному вознаграждению.
- **NEAT:** основной класс, который управляет процессами эволюции популяции, включая мутацию, скрещивание и специализацию.

Пример кода для создания популяции агентов:

```
1 class NEAT:
2     def __init__(self, env, config):
3         self.env = env
4         self.config = config
5         self.population_size = config.population_size
6         self.agents = [Agent(self.env.observation_space.shape[0],
                               self.env.action_space.shape[0]) for _ in range(self.population_size)]
```

6.3 Визуализация: visualize.py

Файл `visualize.py` содержит функции для визуализации ключевых метрик, таких как фитнес, и отслеживания прогресса алгоритма NEAT. Используется библиотека `matplotlib` для построения графиков.

- **plot_metrics:** функция для построения графиков динамики изменения фитнеса за поколение. Это помогает наглядно оценить, как увеличивается производительность агентов по мере их эволюции.

Пример кода для визуализации фитнеса:

```
1 import matplotlib.pyplot as plt
2
3 def plot_metrics(fitness_history):
4     plt.figure(figsize=(10,5))
5
6     # График для среднего фитнеса
7     plt.subplot(1, 2, 1)
8     plt.plot(fitness_history, label='Best Fitness')
9     plt.xlabel('Generation')
10    plt.ylabel('Fitness')
11    plt.title('Best Fitness per Generation')
12    plt.show()
```

6.4 Описание работы алгоритма

1. Инициализация популяции: Создается начальная популяция агентов, каждый из которых имеет минимальную структуру нейронной сети. Популяция оценивается в среде `BipedalWalker-v3`, и каждый агент получает своё вознаграждение.

2. Специализация популяции: На основе схожести геномов агентов популяция разделяется на виды с помощью метода `Speciation`, что помогает сохранить разнообразие в популяции и избежать преждевременной конкуренции между агентами с различными топологиями.

3. Скрещивание и мутация: Процесс скрещивания комбинирует лучшие черты агентов с помощью одноточечного кроссовера, в то время как мутация позволяет добавлять новые нейроны и связи, а также изменять веса существующих связей.

4. Оценка приспособленности: Каждый агент оценивается по вознаграждению, полученному в ходе взаимодействия с окружающей средой. Это вознаграждение используется для вычисления фитнесса.

5. Эволюция: По мере выполнения этих шагов, популяция эволюционирует, улучшая свою способность к выполнению задачи.

6.5 Сохранение и загрузка состояния

Для воспроизводимости эксперимента реализована возможность сохранения состояния сети и агентов. Это позволяет продолжить обучение с того места, где оно было остановлено, или протестировать ранее обученную модель.

Пример кода для сохранения состояния:

```
1 import pickle
2
3 def save_network(network, filename):
4     with open(filename, 'wb') as file:
5         pickle.dump(network, file)
6
7 def load_network(filename):
8     with open(filename, 'rb') as file:
9         return pickle.load(file)
```

`bestgenome.pkl`:

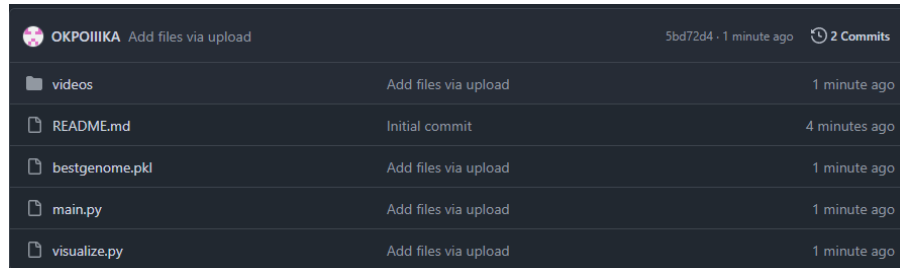
- Предобученная сеть (140 эпох)
- Используется для демонстрации конечных результатов

Директории:

- `videos/` — архив видеозаписей (создается при обучении):
 - MP4-анимация работы агента

- Визуализации лучшей сети
- PNG-изображения с именем `agent_N.png`
- `lab1env/` — виртуальное окружение Python (исключено из Git)

Проект был размещён на GitHub. Ссылка на репозиторий: https://github.com/OKPOIIIIKA/NEAT_PR.



File	Action	Time
videos	Add files via upload	1 minute ago
README.md	Initial commit	4 minutes ago
bestgenome.pkl	Add files via upload	1 minute ago
main.py	Add files via upload	1 minute ago
visualize.py	Add files via upload	1 minute ago

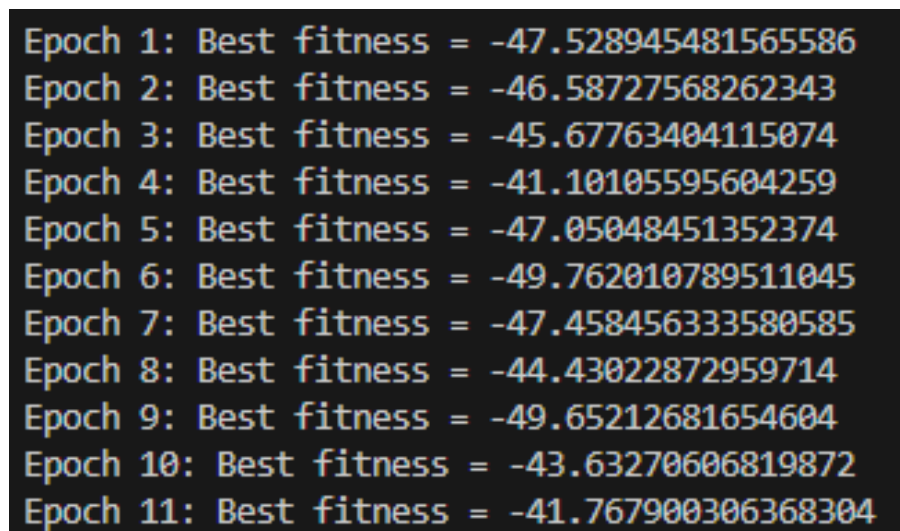
Рис. 6: Структура проекта в GitHub

6.6 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 150 эпох:

```
1 for generation in range(150):
2     neat.evolve()
```

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



```
Epoch 1: Best fitness = -47.528945481565586
Epoch 2: Best fitness = -46.58727568262343
Epoch 3: Best fitness = -45.67763404115074
Epoch 4: Best fitness = -41.10105595604259
Epoch 5: Best fitness = -47.05048451352374
Epoch 6: Best fitness = -49.762010789511045
Epoch 7: Best fitness = -47.458456333580585
Epoch 8: Best fitness = -44.43022872959714
Epoch 9: Best fitness = -49.65212681654604
Epoch 10: Best fitness = -43.63270606819872
Epoch 11: Best fitness = -41.767900306368304
```

Рис. 7: Скриншот командной строки с результатами обучения на первых эпохах. Программа выводит лучшее вознаграждение каждой эпохи.

6.6.1 Видеодемонстрация работы модели

Для проверки качества работы агента была записана видеодемонстрация тестирования, где представлен лучший тестовый эпизод.

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.

Видео лежит в папке videos.

На видео видно, что агент:

- адекватно корректирует курс;
- своевременно шагает;
- минимизирует количество лишних манёвров;

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 был реализован и исследован алгоритм нейроэволюции NEAT для обучения нейронной сети на задаче управления агентом в среде `BipedalWalker-v3`. Был выполнен полный цикл разработки: от создания модульной структуры кода и реализации алгоритма NEAT до визуализации структуры сети и анализа результатов обучения.

Проведённые эксперименты показали, что метод NEAT обеспечивает эффективное и устойчивое обучение агентам с различными архитектурами нейронных сетей, начиная с простых структур и постепенно усложняя их по мере необходимости. Эволюция сети с добавлением нейронов и связей позволяет адаптировать архитектуру для выполнения задачи в условиях динамичной среды. Прогресс в процессе эволюции наглядно прослеживается как по динамике целевых метрик (среднее вознаграждение и `loss`), так и по визуализации структуры сети, что подтверждает способность алгоритма NEAT улучшать качество решения задачи.

Результаты тестирования подтверждают высокую эффективность подхода: агент, обученный в течение 150 поколений, стабильно выполняет задачу управления движением в среде `BipedalWalker-v3`, демонстрируя улучшение в вознаграждении и эффективности стратегии. Средние значения вознаграждения в тестовых эпизодах подтверждают, что сеть способна обобщать навыки и уверенно действовать в различных сценариях среды.

Кроме того, была реализована система сохранения и загрузки состояния нейронной сети, что обеспечивает воспроизводимость результатов и удобство для дальнейших экспериментов. Визуализация, включающая отображение структуры сети и графики динамики метрик, значительно помогает в анализе работы алгоритма и поведении агента.

Таким образом, алгоритм NEAT продемонстрировал свою эффективность при решении задач с непрерывным управлением и может быть использован для других сложных задач в области обучения с подкреплением, где традиционные методы обучения могут быть недостаточно эффективными. Нейроэволюция, и в частности NEAT, остаётся перспективным инструментом для построения адаптивных интеллектуальных систем, способных решать задачи с высокой размерностью и сложной динамикой.

Список использованной литературы

1. Лекция 9. Алгоритм NEAT. Томский политехнический университет, 2025.