УДК 004.421
ББК 32.811
О57

Ярослав Омельяненко
О57 Эволюционные нейросети на языке Python / пер. с анг. В. _. Яценкова. – М.: ДМК Пресс, 2020. – 310 с.: ил.

ISBN

Эта книга дает всестороннее представление о нейроэволюции – подходе к обучению искусственных нейронных сетей, который использует эволюционные алгоритмы, чтобы упростить процесс решения сложных задач в таких областях, как игры, робототехника и моделирование естественных процессов.

Читатель начнет знакомство с ключевыми концепциями и методами нейроэволюции, написав несложный код на языке Python, а затем получит практический опыт работы с популярными библиотеками Python и научится решать распространенные и нестандартные прикладные задачи, используя алгоритмы на основе нейроэволюции. Речь пойдет о том, как адаптировать методы нейроэволюции к существующим проектам нейронных сетей для повышения эффективности обучения и принятия решений; в завершение будет рассказано о топологиях нейронных сетей и о том, как нейроэволюция позволяет развивать сложную топологию из простейшей базовой структуры.

Издание предназначено для специалистов в области машинного обучения и искусственного интеллекта, которые стремятся реализовать алгоритмы нейроэволюции с нуля. Наличие базовых знаний в области глубокого обучения и нейронных сетей, а также программирования на языке Python обязательно.

УДК 004.421
ББК 32.811

ISBN (анг.) 9781838824914
ISBN (рус.) 978-5-97060-854-8

Оглавление

Предисловие

Когда традиционные методы глубокого обучения приблизились к пределу своих возможностей, все больше и больше исследователей начали искать альтернативные подходы к обучению *искусственных нейронных сетей* (artificial neural network, ANN).

Глубокое машинное обучение чрезвычайно эффективно для распознавания образов, но не позволяет выполнять задачи, требующие понимания контекста или незнакомых данных. Многие исследователи, включая Джеффа Хинтона, отца современной концепции глубокого машинного обучения, согласны с тем, что существующий подход к проектированию систем искусственного интеллекта больше не в состоянии справиться с насущными проблемами.

В этой книге мы обсуждаем жизнеспособную альтернативу традиционным методам глубокого машинного обучения – нейроэволюционные алгоритмы. *Нейроэволюция* – это семейство методов машинного обучения, которые используют эволюционные алгоритмы для облегчения решения сложных задач, таких как игры, робототехника и моделирование естественных процессов. Нейроэволюционные алгоритмы имитируют процесс естественного отбора. Очень простые искусственные нейронные сети могут стать очень сложными. Конечным результатом нейроэволюции является оптимальная топология сети, которая делает модель более энергоэффективной и удобной для анализа.

В этой книге вы узнаете о различных нейроэволюционных алгоритмах и получите практические навыки по их использованию для решения различных типовых задач информатики – от классического обучения с подкреплением до создания агентов для автономной навигации по лабиринту. Кроме того, вы узнаете, как можно применить нейроэволюцию для обучения глубоких нейронных сетей при создании агента, способного играть в классические игры Atari.

Цель этой книги – дать вам ясное представление о методах нейроэволюции, проводя различные эксперименты с пошаговым руководством. Она содержит практические примеры в таких областях, как игры, робототехника и моделирование естественных процессов, с использованием реальных наборов данных, чтобы помочь вам лучше понять исследуемые методы. После прочтения этой книги у вас будет все необходимое для применения методов нейроэволюции к другим задачам, по аналогии с представленными экспериментами.

При написании этой книги я стремился дать вам знания о передовых технологиях, которые являются жизненно важной альтернативой традиционному глубокому обучению. Я надеюсь, что применение нейроэволюционных алгоритмов в ваших проектах позволит вам элегантно и эффективно решать проблемы, которые казались неразрешимыми.

Для кого написана эта книга

Эта книга предназначена для практиков машинного обучения, исследователей глубокого обучения и энтузиастов искусственного интеллекта, которые стремятся реализовать нейроэволюционные алгоритмы с нуля. Вы узнаете, как применять эти алгоритмы к различным задачам реального мира. Вы узнаете, как методы нейроэволюции могут оптимизировать процесс обучения искусственных нейронных сетей. Вы познакомитесь с основными понятиями нейроэволюции и получите необходимые практические навыки, чтобы использовать ее в своей работе и экспериментах. Знание Python, а также основ глубокого обучения и нейронных сетей является обязательным.

О чем эта книга

Глава 1 знакомит с основными понятиями генетических алгоритмов, такими как генетические операторы и схемы кодирования генома.

В *главе 2* обсуждаются практические аспекты методов нейроэволюции. В этой главе рассказано о достоинствах и недостатках популярных библиотек Python, которые предоставляют реализации алгоритма NEAT и его расширений.

Глава 3 – это место, где вы начинаете экспериментировать с алгоритмом NEAT, реализуя решение для классической проблемы информатики.

В *главе 4* продолжаются эксперименты, связанные с классическими проблемами информатики в области обучения с подкреплением.

В *главе 5* вы продолжаете эксперименты с нейроэволюцией, пытаясь создать решатель, способный найти выход из лабиринта. Вы узнаете, как реализовать симуляцию робота с набором датчиков для обнаружения препятствий и контроля его положения в лабиринте.

В *главе 6* вы воспользуетесь практическим опытом, полученным при создании решателя лабиринта в предыдущей главе, чтобы встать на путь создания более совершенного решателя.

Глава 7 знакомит вас с передовыми методами нейроэволюции. Вы узнаете о схеме косвенного кодирования генома, в которой используются *сети, производящие составные паттерны* (Compositional Pattern Producing Network, CPPN) для кодирования топологий крупномасштабных искусственных нейросетей.

В *главе 8* вы узнаете, как выбрать конфигурацию модульной сетчатки, которая лучше всего подходит для конкретной проблемной области.

В *главе 9* мы обсуждаем, как широко распространенная в природе стратегия коэволюции может быть перенесена в область нейроэволюции.

Глава 10 представляет вам концепцию глубокой нейроэволюции, которую можно использовать для обучения *глубоких искусственных нейронных сетей* (Deep Artificial Neural Network, DNN).

В *главе 11* рассказано, как начать работу над прикладной задачей, как настроить гиперпараметры нейроэволюционного алгоритма, как использовать передовые инструменты визуализации и какие показатели можно использовать для анализа выполнения алгоритма.

Глава 12 обобщает все, что вы узнали в этой книге, и дает рекомендации о том, как продолжить свое самообразование.

Как получить максимальную отдачу от этой книги

Для работы с примерами, представленными в этой книге, необходимы практические навыки программирования на языке Python. Для лучшего понимания исходного кода предпочтительно использовать IDE, которая поддерживает подсветку синтаксиса Python и ссылки внутри кода. Если у вас нет специализированной среды разработки, вы можете использовать инструмент разработки Microsoft Visual Studio Code. Он бесплатный и кросс-платформенный, и вы можете свободно скачать его по адресу https://code.visualstudio.com.

Python и большинство библиотек, которые мы обсуждаем в этой книге, являются кросс-платформенными и совместимы с Windows, Linux и macOS. Все описанные в книге эксперименты выполняются из командной строки, поэтому ознакомьтесь с приложением консоли терминала, установленным в выбранной вами ОС.

Для выполнения эксперимента, описанного в главе 10, вам необходим доступ к современному ПК с графическим ускорителем Nvidia GeForce GTX 1080Ti или выше. Этот эксперимент также лучше проводить в среде Ubuntu Linux. Ubuntu – это современная и мощная ОС на базе Linux. Навык работы с ней вам очень пригодится.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте **www.dmkpress.com** на странице с описанием соответствующей книги.

Скачивание цветных иллюстраций ***Если надо???***

Мы предоставляем файл PDF с цветными изображениями скриншотов и рисунков, используемых в этой книге. Вы можете скачать его по адресу: **https://static.packt-cdn.com/downloads/ 9781838824914_ColorImages.pdf.**

Условные обозначения и соглашения, принятые в книге

В книге используются следующие типографские соглашения.

Курсив используется для смыслового выделения новых терминов, адресов электронной почты, а также имен и расширений файлов.

`Моноширинный шрифт` применяется для листингов программ, а также в обычном тексте для обозначения имен переменных, функций, типов, объектов, баз данных, переменных среды, операторов и ключевых слов.

`Моноширинный полужирный шрифт` используется для обозначения команд или фрагментов текста, которые пользователь должен ввести дословно без изменений.

`Моноширинный курсив` – для обозначения в исходном коде или в командах шаблонных меток-заполнителей, которые должны быть заменены соответствующими контексту реальными значениями.

ЧАСТЬ I

Основы эволюционных вычислительных алгоритмов и методов нейроэволюции

В первой части книги представлены основные понятия эволюционных вычислений и обсуждаются особенности алгоритмов на основе нейроэволюции и перечень библиотек Python, которые могут использоваться для их реализации. Вы познакомитесь с основами методов нейроэволюции и получите практические рекомендации о том, как начать свои эксперименты. Здесь также приведено краткое знакомство с менеджером пакетов Anaconda для Python в рамках настройки вашей среды разработки.

Эта часть книги состоит из следующих глав:

- главы 1 «Обзор методов нейроэволюции»;
- главы 2 «Библиотеки Python и настройка среды разработки».

Глава 1

Обзор методов нейроэволюции

Концепция *искусственной нейронной сети* (artificial neural networks, ANN), которую мы дальше для удобства будем называть просто *нейросетью*, основана на структуре человеческого мозга. Существует устойчивое убеждение, что если суметь достоверно скопировать эту сложнейшую структуру, то можно создать *искусственный интеллект* (artificial intellect, AI). Мы все еще на пути к достижению этой цели. Хотя нам по силам создание специализированных AI-агентов, мы еще далеки от создания универсального искусственного интеллекта.

В этой главе вы познакомитесь с понятием искусственных нейросетей и двумя методами, которые мы можем использовать для их обучения (градиентный спуск с обратным распространением ошибки и нейроэволюция), чтобы нейросеть научилась приближаться к целевой функции. Тем не менее мы сосредоточимся в основном на обсуждении семейства алгоритмов на основе нейроэволюции. Вы узнаете о реализации эволюционного процесса, основанного на естественной эволюции, и познакомитесь с наиболее популярными алгоритмами нейроэволюции: NEAT, HyperNEAT и ES-HyperNEAT. Мы также обсудим методы оптимизации, которые можно использовать для поиска окончательных решений, и проведем сравнение между алгоритмами приближения к цели и алгоритмом поиска новизны. К концу этой главы вы будете иметь полное представление о внутреннем устройстве алгоритмов нейроэволюции и будете готовы применить эти знания на практике.

В этой главе мы рассмотрим следующие темы:

- эволюционные алгоритмы и нейроэволюционные методы;
- обзор алгоритма NEAT;
- NEAT на основе гиперкуба;
- HyperNEAT с развивающимся субстратом;
- метод оптимизации на основе поиска новизны.

1.1 Эволюционные алгоритмы и нейроэволюционные методы

Термин «искусственная нейронная сеть» обозначает граф *узлов*, соединенных *связями*, где каждая из связей имеет определенный *вес*. Узел нейросети является своего рода пороговым оператором, который позволяет сигналу проходить дальше только после срабатывания определенной функции активации. Это отдаленно напоминает принцип, по которому организованы нейроны головного мозга. Как правило, процесс обучения нейросети состоит из выбора подходящих значений веса для всех связей в сети. Таким образом, нейросеть способна аппроксимировать любую функцию и может рассматриваться как *универсальный аппроксиматор*, который определяется теоремой универсальной аппроксимации.

Чтобы познакомиться с доказательством теоремы универсальной аппроксимации, прочтите следующие статьи:

- *Cybenko G.* (1989) Approximations by Superpositions of Sigmoidal Functions, Mathematics of Control, Signals, and Systems, 2 (4), 303–314;
- *Leshno Moshe, Lin Vladimir Ya., Pinkus Allan, Schocken Shimon* (January 1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks. 6 (6): 861–867. doi:10.1016/S0893-6080(05)80131-5 (**https://www.sciencedirect.com/science/ article/abs/pii/S0893608005801315?via%3Dihub**);
- *Kurt Hornik* (1991). Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 4 (2), 251–257. doi:10.1016/0893-6080(91)90009-T (**https://www. sciencedirect.com/science/article/abs/pii/089360809190009T?via%3Dihub**);
- *Hanin B.* **(2018). Approximating Continuous Functions by ReLU Nets of Minimal Width. arXiv preprint arXiv:1710.11278 (https://arxiv.org/abs/1710.11278**).

За последние 70 лет было придумано много методов обучения нейросетей. Однако наиболее популярная техника, получившая известность в последнем десятилетии, была предложена Джеффри Хинтоном. Она основана на обратном распространении ошибки прогнозирования через сеть с различными методами оптимизации, построенными на основе градиентного спуска функции потерь по отношению к весам связей между узлами сети. Этот метод демонстрирует выдающуюся эффективность обучения глубоких нейронных сетей для задач, связанных главным образом с распознаванием образов. Однако, несмотря на присущие ему достоинства, он имеет существенные недостатки. Один из этих недостатков заключается в том, что для усвоения чего-то полезного из определенного набора данных требуется огромное количество обучающих образцов. Другим существенным недостатком является фиксированная архитектура нейросети, созданная экспериментатором вручную, что приводит к неэффективному использованию вычислительных ресурсов. Это связано с тем, что значительное количество сетевых узлов не уча-

ствует в процессе вывода. Кроме того, методы обратного распространения имеют проблемы с передачей полученных знаний в другие смежные области.

Наряду с методами обратного распространения применяются очень многообещающие эволюционные алгоритмы, которые могут решать вышеупомянутые проблемы. Эти основанные на биологии методы черпают вдохновение из теории эволюции Дарвина и используют принципы эволюции видов для создания искусственных нейронных сетей. Основная идея нейроэволюции состоит в том, чтобы создавать нейросеть с помощью стохастических методов поиска, основанных на популяции. Используя эволюционный подход, можно разработать оптимальные архитектуры нейронных сетей, которые точно решают конкретные задачи. В результате могут быть созданы компактные и энергоэффективные сети с умеренными требованиями к вычислительной мощности. Процесс эволюции реализуется путем применения генетических операторов (мутация, кроссовер) к популяции хромосом (генетически кодированные представления нейросетей или решений) на протяжении многих поколений. Большие надежды на этот метод основаны на том, что в природных биологических системах каждое последующее поколение становится все более приспособленным к внешним обстоятельствам, которые можно выразить целевой функцией, то есть они становятся лучшими приближениями целевой функции.

Далее мы обсудим основные понятия генетических алгоритмов. Вам достаточно будет иметь умеренный уровень понимания принципов работы генетических алгоритмов.

1.1.1 Генетические операторы

Генетические операторы находятся в самом сердце каждого эволюционного алгоритма, и от них зависит результативность любого нейроэволюционного алгоритма. Существует два основных генетических оператора: *мутация* и *кроссовер* (*рекомбинация*).

В этой главе вы узнаете об основах генетических алгоритмов и о том, как они отличаются от обычных алгоритмов, в которых для обучения нейросети используются методы обратного распространения ошибок.

Оператор мутации

Оператор мутации выполняет важную роль сохранения генетического разнообразия популяции в процессе эволюции и предотвращает остановку в локальных минимумах, когда хромосомы организмов в популяции становятся слишком похожими. Эта мутация изменяет один или несколько генов в хромосоме в соответствии с вероятностью мутации, определенной экспериментатором. Вводя случайные изменения в хромосому *решателя* (solver), мутация позволяет эволюционному процессу исследовать новые области в пространстве поиска возможных решений и находить все лучшие и лучшие решения на протяжении поколений.

На рис. 1.1 показаны распространенные типы операторов мутации.

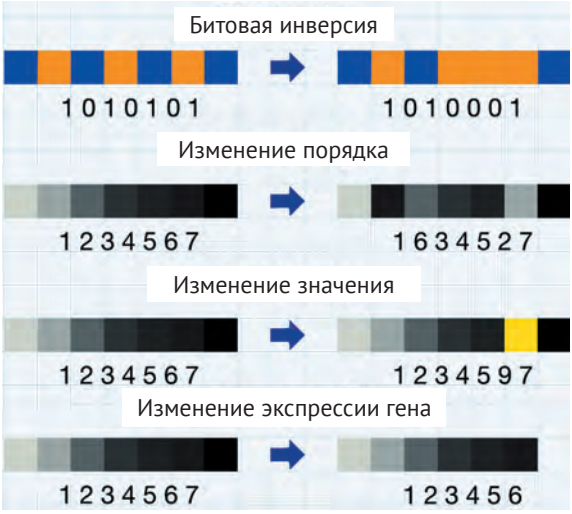

Рис. 1.1. Типы операторов мутации

Точный тип оператора мутации зависит от вида генетического кодирования, используемого конкретным генетическим алгоритмом. Среди различных типов мутаций, с которыми мы сталкиваемся, можно выделить следующие основные варианты:

- **битовая инверсия**: инвертируется случайно выбранный бит (бинарное кодирование);
- **изменение порядка**: изменение положения двух случайно выбранных генов в геноме (кодирование пермутации);
- **изменение значения**: к рабочему гену в случайной позиции добавляется небольшое значение (кодирование значения);
- **изменение экспрессии гена**: выбранный случайным образом ген добавляется в генотип / удаляется из генотипа (структурное кодирование).

Генотипы могут быть закодированы с использованием схем генетического кодирования с фиксированной и переменной длиной хромосомы. Первые три мутации могут быть применены к обоим типам схем кодирования. Последняя мутация может происходить только в генотипах, которые были закодированы с использованием переменной длины.

Оператор кроссовера

Оператор кроссовера (рекомбинации) позволяет нам стохастически генерировать новые поколения (решения) из существующих популяций путем рекомбинации генетической информации от двух родителей для генерации потомка. Таким образом, доли хороших решений от родительских организмов могут быть объединены и потенциально могут привести к лучшему потомству. Как правило, после операции кроссовера полученное потомство подвергается мутации перед добавлением в популяцию следующего поколения.

Различные операторы кроссовера показаны на рис. 1.2.

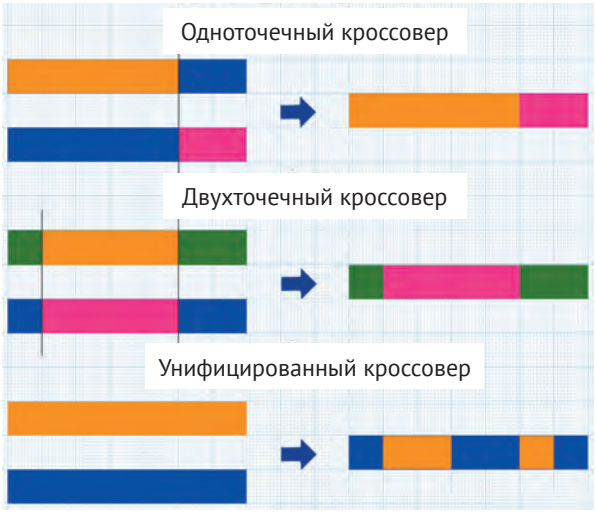

Рис. 1.2. Типы операторов кроссовера (рекомбинации)

Различные типы операторов кроссовера также зависят от схемы кодирования генома, используемого конкретными алгоритмами, но наиболее распространенными являются следующие:

- **одноточечный кроссовер**: выбирается случайная точка пересечения, часть генома от начала до точки пересечения копируется в потомство от одного родителя, а остаток копируется от другого родителя;
- **двухточечный кроссовер**: случайным образом выбираются две точки пересечения, часть генома от начала до первой точки копируется из первого родителя, часть между первой и второй точками пересечения копируется из второго родителя, и остаток копируется из первого родителя;
- **унифицированный кроссовер**: гены случайным образом копируются из первого или второго родителя.

1.1.2 Схемы кодирования генома

Одним из наиболее важных решений при разработке нейроэволюционного алгоритма является выбор генетического представления нейронной сети, которое может быть реализовано следующими способами:

- стандартная мутация (см. раздел «Оператор мутации»);
- операторы комбинирования (см. раздел «Оператор кроссовера»).

В настоящее время существуют две основные схемы кодирования генома: прямая и косвенная. Рассмотрим каждую схему более подробно.

Прямое кодирование генома

Прямое кодирование генома пытались использовать в методах нейроэволюции для создания нейросетей, относящихся к сетям с *фиксированной топологией*, – то есть топология сети определяется только экспериментатором. Здесь гене-

тический код (генотип) реализуется как вектор действительных чисел, представляющих силу (вес) связей между узлами сети.

Эволюционные операторы изменяют значения вектора весовых коэффициентов с помощью оператора мутации и объединяют векторы родительских организмов с помощью оператора кроссовера для получения потомства. Позволяя легко применять эволюционные операторы, упомянутый метод кодирования в то же время имеет некоторые существенные недостатки. Один из его основных недостатков заключается в том, что топология сети определяется экспериментатором с самого начала и остается постоянной для всех поколений за время выполнения эксперимента. Этот подход противоречит естественному эволюционному процессу, при котором в ходе эволюции изменяются не только свойства, но и физическая структура организмов, что позволяет охватывать максимально широкое пространство поиска и находить оптимальные решения. На рис. 1.3 показан эволюционный процесс.

Рис. 1.3. Эволюционный процесс

Чтобы устранить недостатки методов с фиксированной топологией, Кеннет О. Стэнли предложил метод *нейроэволюции с развитием топологии* (neuroevolution of augmenting topologies, NEAT). Основная идея этого алгоритма заключается в том, что эволюционные операторы применяются не только к вектору с весами всех связей, но и к топологии созданной нейронной сети. Таким образом, путем генерации популяций организмов проверяются различные топологии с различными весами связей. Мы обсудим особенности алгоритма NEAT позже в этой главе.

Алгоритм NEAT демонстрирует выдающуюся эффективность в самых разных задачах – от традиционного обучения с подкреплением до управления сложными автономными персонажами в компьютерных играх – и стал одним из самых популярных эволюционных алгоритмов за всю историю машинного обучения. Тем не менее он принадлежит к семейству алгоритмов прямого кодирования, которое ограничивает его использование до развития только нейросетей небольшого размера, где пространство параметров ограничено максимум тысячами связей. Это связано с тем, что каждая связь кодируется непосредственно в генотипе, и при большом количестве закодированных связей вычислительные требования значительно возрастают. Это делает невозможным использование алгоритма для развития больших нейронных сетей.

Косвенное кодирование генома

Чтобы преодолеть проблемы размерности в подходе с прямым кодированием, Кеннет О. Стэнли предложил метод *косвенного кодирования*, который основан на кодировании фенотипа посредством генома в ДНК. Он основан на том факте, что физический мир построен вокруг геометрии и закономерностей (структурных паттернов), где естественные симметрии встречаются повсюду. Таким образом, размер кода любого физического процесса может быть значительно уменьшен путем повторного использования определенного набора блоков кодирования для одной и той же структуры, которая повторяется много раз. Предложенный метод, называемый *нейроэволюцией с развитием топологии на основе гиперкуба* (hypercube-based neuroevolution of augmenting topologies, HyperNEAT), предназначен для построения крупномасштабных нейронных сетей с использованием геометрических закономерностей. HyperNEAT использует *сети, производящие составные паттерны* (compositional pattern producing network, CPPN), для представления связей между узлами как функции декартова пространства. Мы обсудим HyperNEAT более подробно далее в этой главе.

1.1.3 Коэволюция

В природе популяции разных видов часто одновременно развиваются во взаимодействии друг с другом. Этот тип межвидовых отношений называется *коэволюцией*. Коэволюция является мощным инструментом естественной эволюции, и неудивительно, что она привлекла внимание разработчиков нейроэволюционных алгоритмов. Существует три основных типа коэволюции:

- *мутуализм*, когда два или более вида мирно сосуществуют и взаимно выигрывают друг от друга;
- *конкурентная коэволюция*:
  - *хищничество*, когда один организм убивает другой и потребляет его ресурсы;
  - *паразитизм*, когда один организм использует ресурсы другого, но не убивает его;
- *комменсализм*, когда представители одного вида получают выгоды, не причиняя вреда и не принося выгоды другим видам.

Исследователи изучили существующие стратегии коэволюции и выявили их плюсы и минусы. В этой книге мы представим нейроэволюционный алгоритм, который использует принцип комменсализма для поддержания двух одновременно развивающихся групп: совокупности возможных решений и совокупности целевых функций кандидатов. Мы обсудим алгоритм *эволюции решений и приспособленности* (solution and fitness evolution, SAFE) в главе 9.

1.1.4 Модульность и иерархия

Другим важным аспектом организации естественных когнитивных систем является модульность и иерархия. При изучении человеческого мозга нейробиологи обнаружили, что это не монолитная система с однородной структурой, а сложная иерархия модульных структур. Кроме того, из-за ограни-

чений скорости распространения сигнала в биологических тканях структура мозга обеспечивает *принцип локальности*, когда связанные задачи обрабатываются геометрически смежными структурами в мозге. Эти особенности природных систем не остались без внимания исследователей нейроэволюции и реализованы во многих эволюционных алгоритмах. В главе 8 мы обсудим создание модульных нейросетей с использованием алгоритма на основе нейроэволюции.

1.2 Обзор алгоритма NEAT

Метод NEAT предназначен для уменьшения размерности пространства поиска параметров посредством постепенного развития структуры нейросети в процессе эволюции. Эволюционный процесс начинается с популяции маленьких, простых геномов (семян) и постепенно увеличивает их сложность с каждым новым поколением.

Геномы семян имеют очень простую топологию: доступны (экспрессированы) только входные, выходные и смещающие нейроны. На начальном этапе скрытые узлы отсутствуют, чтобы гарантировать, что поиск решения начинается в пространстве параметров (весов связей) с наименьшим числом измерений. С каждым новым поколением вводятся дополнительные гены, расширяющие пространство поиска решения, представляя новое измерение, которое ранее не существовало.

Таким образом, эволюция начинается с поиска в небольшом пространстве, которое можно легко оптимизировать, и при необходимости добавляет новые измерения. При таком подходе сложные фенотипы (решения) могут быть обнаружены постепенно, шаг за шагом, что намного эффективнее, чем запуск поиска непосредственно в обширном пространстве окончательных решений. Естественная эволюция использует похожую стратегию, время от времени добавляя новые гены, которые делают фенотипы более сложными. В биологии этот процесс постепенного усложнения называется *комплексным расширением*.

Основная цель метода NEAT – минимизировать сложность структуры генома – касается не только конечного продукта, но и всех промежуточных поколений организмов. Таким образом, эволюция топологии сети приводит к значительному выигрышу в производительности за счет сокращения общих решений для пространства поиска. Например, многомерное пространство окончательного решения возникает только в конце эволюционного процесса. Еще одна существенная особенность алгоритма заключается в том, что каждая структура, представленная в геноме, является предметом последующих оценок пригодности в будущих поколениях. Кроме того, во время эволюционного процесса выживут только полезные структуры. Другими словами, структурная сложность генома всегда оправдана целями.

1.2.1 Схема кодирования NEAT

Схема генетического кодирования NEAT разработана таким образом, чтобы можно было легко сопоставлять соответствующие гены во время процесса

спаривания, когда к двум родительским геномам применяется оператор кроссовера. Геном NEAT является линейным представлением схемы связей кодированной нейронной сети, как показано на рис. 4.1.

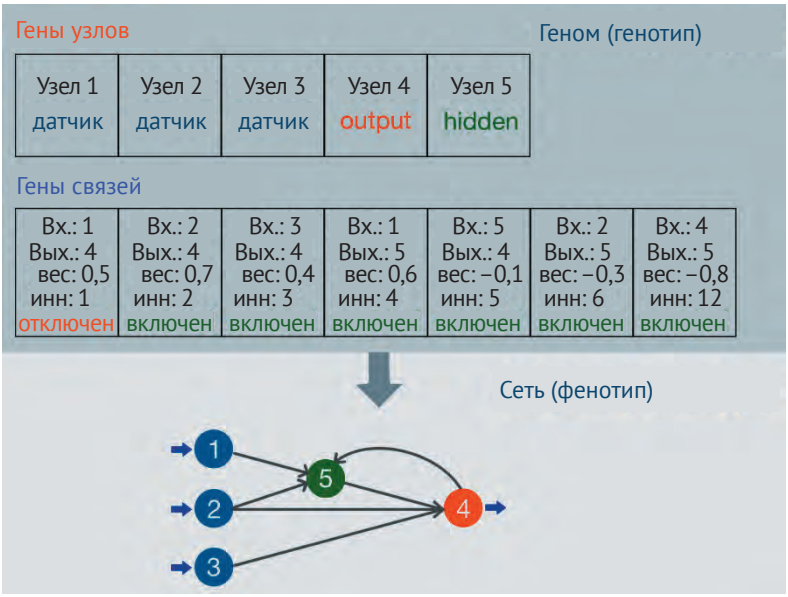

Рис. 1.4. Схема генома NEAT

Каждый геном представлен в виде списка *генов связей*, которые кодируют связи между узлами нейронной сети. Кроме того, существуют *гены узлов*, которые кодируют информацию о сетевых узлах, такую как идентификатор узла, тип узла и тип функции активации. Ген связи кодирует следующие параметры связи между узлами:

- идентификатор входного узла;
- идентификатор выходного узла;
- сила (вес) связи;
- бит, который указывает, включена (экспрессирована) связь или нет;
- номер обновления, который позволяет сопоставлять гены во время рекомбинации.

На нижней части рис. 4.1 представлена схема того же генома в виде ориентированного графа.

1.2.2 Структурные мутации

Специфический для NEAT оператор мутации может изменить силу (вес) связи и структуру сети. Существует два основных типа структурных мутаций:

- добавление новой связи между узлами;
- добавление нового узла в сеть.

Структурные мутации алгоритма NEAT схематически изображены на рис. 1.5.

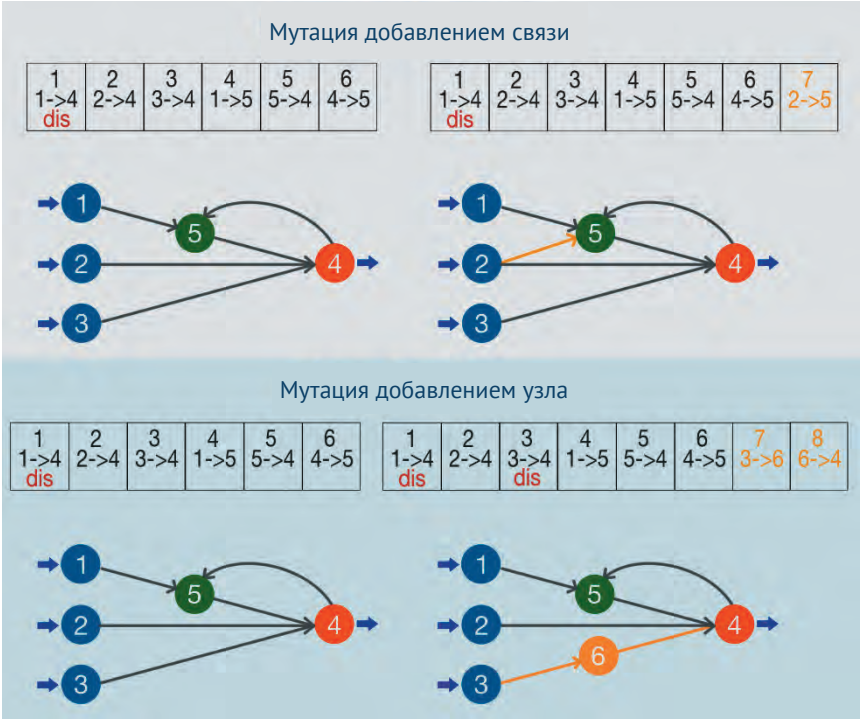

Рис. 1.5. Структурные мутации в алгоритме NEAT

Когда оператор мутации применяется к геному NEAT, вновь добавленному гену (гену связи или гену узла) присваивается очередной номер обновления. В ходе эволюционного процесса геномы организмов в популяции постепенно становятся больше, и образуются геномы различных размеров. Этот процесс приводит к тому, что в одинаковых позициях в геноме находятся разные гены связей, что делает процесс сопоставления между генами одного и того же происхождения чрезвычайно сложным.

1.2.3 Кроссовер с номером обновления

В эволюционном процессе есть неочевидная информация, которая точно говорит нам, какие гены должны совпадать между геномами любого организма в топологически разнообразной популяции. Это информация, при помощи которой каждый ген сообщает нам, от какого предка он был получен.

Гены связей с одним и тем же историческим происхождением представляют одну и ту же структуру, несмотря на то что, возможно, имеют разные значения весов. Историческое происхождение генов в алгоритме NEAT отражено в увеличивающихся номерах обновлений, которые позволяют нам отслеживать хронологию структурных мутаций.

В то же время в процессе кроссовера потомки наследуют номера обновлений генов от родительских геномов. Таким образом, номера обновлений конкретных генов никогда не меняются, что позволяет сопоставить сходные гены из разных геномов в ходе кроссовера. Номера обновлений совпадаю-

щих генов одинаковы. Если номера обновлений не совпадают, ген принадлежит непересекающейся или избыточной части генома, в зависимости от того, находится ли его номер обновления внутри или вне диапазона других родительских номеров. Непересекающиеся или избыточные гены представляют собой структуры, которых нет в геноме другого родителя и которые требуют особой обработки во время фазы кроссовера. Таким образом, потомство наследует гены, которые имеют одинаковые номера обновлений. Они случайным образом выбираются у одного из родителей. Потомство всегда наследует непересекающиеся или избыточные гены от родителей с самой высокой степенью приспособленности. Эта особенность позволяет алгоритму NEAT эффективно выполнять рекомбинацию генов с использованием линейного кодирования генома без необходимости проведения сложного топологического анализа.

На рис. 1.6 показан пример кроссовера (рекомбинации) между двумя родителями с использованием алгоритма NEAT. Геномы обоих родителей выравниваются при помощи номера обновления (число в верхней части ячейки гена связи). После этого производится потомство путем случайного выбора от любого из родителей генов связи с совпадающими номерами обновлений – это гены с номерами от одного до пяти. Наконец, от любого из родителей безусловным образом добавляются непересекающиеся и избыточные гены, которые выстраиваются по нарастанию номера обновления.

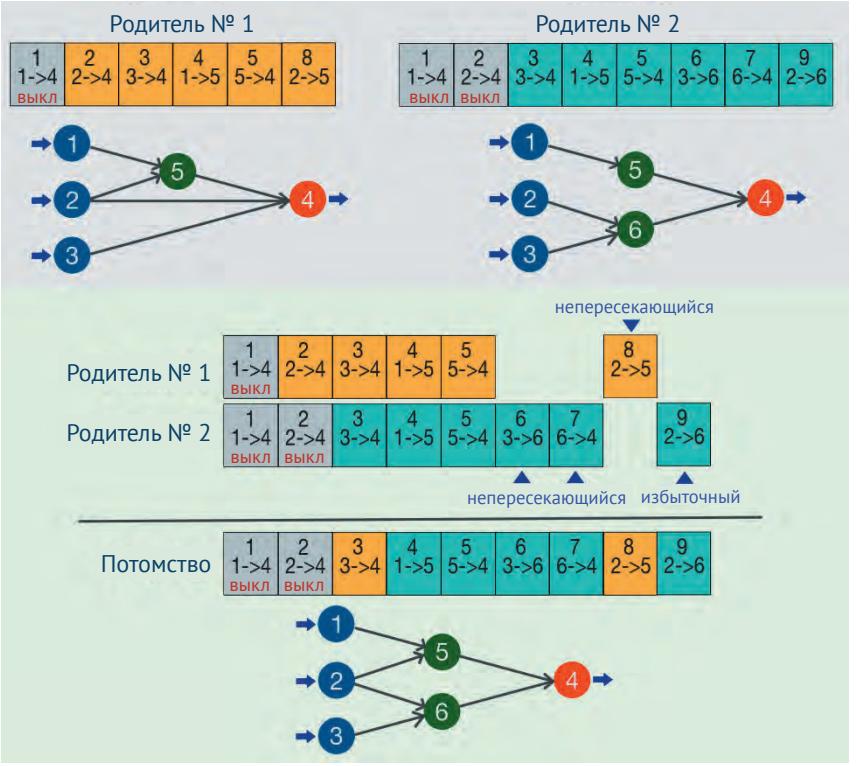

Рис. 1.6. Кроссовер (рекомбинация) алгоритма NEAT

1.2.4 Видообразование

В процессе эволюции организмы могут из поколения в поколение создавать разнообразные топологии, но они не могут производить и поддерживать собственные топологические обновления. Меньшие сетевые структуры оптимизируются быстрее, чем большие, что искусственно уменьшает шансы на выживание потомка генома после добавления нового узла или связи. Таким образом, недавно дополненные топологии испытывают *отрицательное эволюционное давление* из-за временного снижения приспособленности организмов в популяции. В то же время новые топологии могут содержать инновации, которые в конечном итоге приводят к выигрышному решению, и было бы жалко их потерять. Для решения проблемы временного снижения пригодности в алгоритм NEAT была введена концепция видообразования. *Видообразование* ограничивает круг организмов, которые могут спариваться, вводя узкие ниши, когда организмы, принадлежащие к одной и той же нише, в фазе кроссовера конкурируют только друг с другом, а не со всеми организмами в популяции. Видообразование реализуется путем деления популяции так, чтобы организмы с похожей топологией принадлежали к одному и тому же виду, то есть за счет *кластеризации геномов по видам*. Обобщенная форма алгоритма видообразования представлена на рис. 1.7.

Алгоритм 1: Кластеризация геномов по видам

Вход: Популяция организмов (*Population*) и известные виды (*Species*)

Результат: Организмы кластеризованы по видам. При необходимости созданы новые виды.

```
foreach genome ∈ Population do
    foreach S ∈ Species do
        if genome.IsCompatible(S) then
            // Добавляем подходящий геном в текущие виды
            S.AddGenome(genome);
        else if S is the last known species then
            // Создаем новые виды для данного генома
            S_new ⟵ create_new_species(genome) ;
            // Добавляем новые виды в список известных видов
            Species ⟵ Species ∪ S_new ;
```

Рис. 1.7. Обобщенная форма алгоритма видообразования

Метод NEAT позволяет создавать сложные нейросети, способные решать различные задачи оптимизации управления, а также другие проблемы обучения без учителя. Благодаря способности топологии нейросети эволюционировать посредством усложнения и видообразования полученные решения, как правило, имеют оптимальную производительность обучения и логического вывода.

Результирующая топология растет строго в соответствии с проблемой, которая должна быть решена, без каких-либо лишних скрытых слоев, вводимых

обычными методами проектирования топологии нейросетей, обучаемых на основе обратного распространения ошибки.

Подробнее прочитать про алгоритм NEAT можно в оригинальной статье по адресу: **http://nn.cs.utexas.edu/downloads/papers/stanley.phd04.pdf.**

1.3 NEAT на основе гиперкуба

Интеллект является продуктом мозга, а человеческий мозг как структура сам является продуктом естественной эволюции. Такая сложная структура развивалась в течение миллионов лет под давлением суровых условий и в то же время конкурировала за выживание с другими живыми существами. В результате возникла чрезвычайно развитая система со многими слоями, модулями и триллионами связей между нейронами. Структура человеческого мозга является нашей путеводной звездой и помогает нам в создании систем искусственного интеллекта. Однако как мы можем постичь всю сложность человеческого мозга с помощью наших несовершенных инструментов?

Изучая мозг человека, нейробиологи обнаружили, что его пространственная структура играет важную роль во всех задачах восприятия и познания – от зрения до абстрактного мышления. Было найдено много сложных геометрических структур, таких как клетки коры мозга, которые помогают нам в инерциальной навигации, и кортикальные столбцы, которые связаны с сетчаткой глаза для обработки зрительных образов. Было доказано, что структура мозга позволяет нам эффективно реагировать на образы в сигналах, поступающих от сенсориума[1], с помощью обособленных нейронных структур, которые активируются определенными образами на входах. Эта особенность мозга позволяет ему использовать чрезвычайно эффективный способ представления и обработки всего разнообразия входных данных, полученных из окружающей среды. Наш мозг превратился в набор эффективных механизмов распознавания и обработки образов, которые активно применяют повторное использование нейронных модулей, тем самым значительно сокращая количество необходимых нейронных структур. Это стало возможным только благодаря сложной модульной иерархии и пространственной интеграции различных частей.

Иными словами, биологический мозг реализует сложные иерархические и пространственно-ориентированные процедуры обработки данных. Это вдохновило исследователей нейроэволюции на внедрение аналогичных методов обработки данных в области искусственных нейронных сетей. При проектировании подобных систем необходимо учитывать следующие проблемы:

- огромное количество входных признаков и параметров обучения, которые требуют построения крупномасштабных нейросетей;
- эффективное представление естественных геометрических закономерностей и симметрий, которые наблюдаются в физическом мире;

[1] Совокупность органов чувств (источников биологических сигналов) и клеток головного мозга, непосредственно принимающих эти сигналы. – *Прим. перев.*

- эффективная обработка входных данных благодаря внедрению принципа локальности, то есть когда пространственно- и семантически смежные структуры данных обрабатываются модулями взаимосвязанных нейронных блоков, которые занимают одну и ту же компактную область всей сетевой структуры.

В этом разделе вы узнали о методе HyperNEAT на основе гиперкуба, который был предложен Кеннетом О. Стэнли для решения различных задач с использованием многомерных геометрических структур. Далее мы рассмотрим *сети, производящие составные паттерны* (CPPN).

1.3.1 Сети, производящие составные паттерны

Алгоритм HyperNEAT расширяет исходный алгоритм NEAT, вводя новый тип схемы кодирования косвенного генома под названием CPPN. Этот тип кодирования позволяет представлять паттерны связей нейросети фенотипа как функцию его геометрии.

HyperNEAT сохраняет паттерн связей нейронной сети фенотипа в виде четырехмерного гиперкуба, где каждая точка кодирует связь между двумя узлами (то есть координатами исходного и целевого нейронов), а CPPN рисует в нем различные паттерны. Другими словами, CPPN вычисляет четырехмерную функцию, которая определяется следующим образом:

$$w = CPPN\left(x_1, y_2, x_2, y_2\right)$$

Здесь исходный узел находится в точке (x_1, y_2), а целевой узел – в точке (x_1, y_2). На этом этапе CPPN возвращает вес для каждой связи между каждым узлом в сети фенотипа, который представлен в виде сетки. По соглашению, связь между двумя узлами не *экспрессируется* (не участвует в процессах генома), если величина веса связи, вычисленная с помощью CPPN, меньше минимального порогового значения ($w_{\min}$). Таким образом, паттерн связей, созданный CPPN, может представлять произвольную топологию сети. Паттерн связей может использоваться для кодирования крупномасштабных нейросетей путем обнаружения закономерностей в обучающих данных и может повторно использовать тот же набор генов для кодирования повторений. Паттерн связей, созданный CPPN, принято называть *субстратом* (substrate).

На рис. 1.8 показана интерпретация паттерна геометрической связности на основе гиперкуба.

В отличие от традиционных нейросетевых архитектур, CPPN может использовать различные функции активации для своих скрытых узлов и тем самым отражать различные геометрические закономерности. Например, для представления повторений можно использовать тригонометрический синус, а для обеспечения локальности в определенной части сети (то есть симметрии вдоль оси координат) – гауссову кривую. Таким образом, схема кодирования CPPN может компактно представлять паттерны с различными геометрическими закономерностями, такими как симметрия, повторение, повторение с закономерностями и т. д.

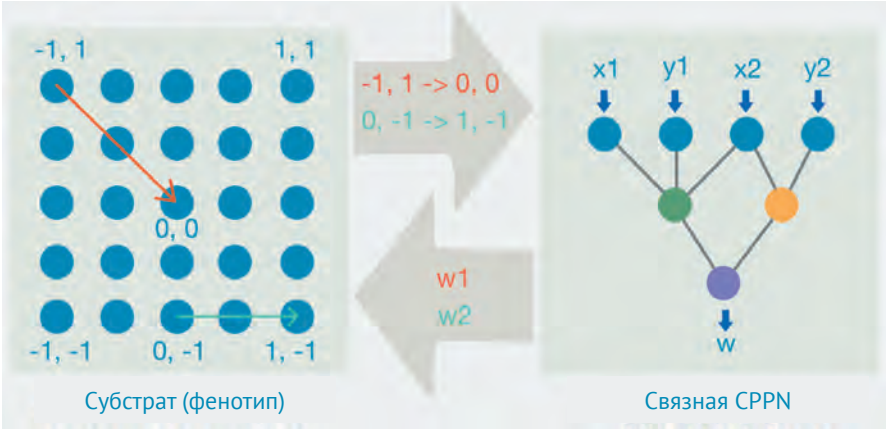

Рис. 1.8. Интерпретация паттерна геометрической связности на основе гиперкуба

1.3.2 Конфигурация субстрата

Компоновка сетевых узлов в субстрате, к которому подключается CPPN, может принимать различные формы, которые лучше всего подходят для решения конкретных задач. Экспериментатор отвечает за выбор подходящей компоновки для достижения оптимальной производительности. Например, выходные узлы, которые управляют радиальным объектом, таким как механизм с шестью шагающими конечностями, удобнее всего располагать в круговой форме, чтобы паттерн связей мог быть представлен в полярных координатах.

Существует несколько распространенных типов конфигурации субстрата, которые обычно используются с HyperNEAT (рис. 1.9):

- **двухмерная сетка**: регулярная сетка узлов сети в двухмерном декартовом пространстве с центром в (0, 0);
- **трехмерная сетка**: регулярная сетка узлов сети в трехмерном декартовом пространстве с центром в (0, 0, 0);
- **сэндвич**: две двухмерные плоские сетки с узлами входа и выхода, где один слой может выстраивать связи в направлении другого;
- **круговая**: упорядоченная радиальная структура, которая подходит для определения закономерностей в полярных координатах.

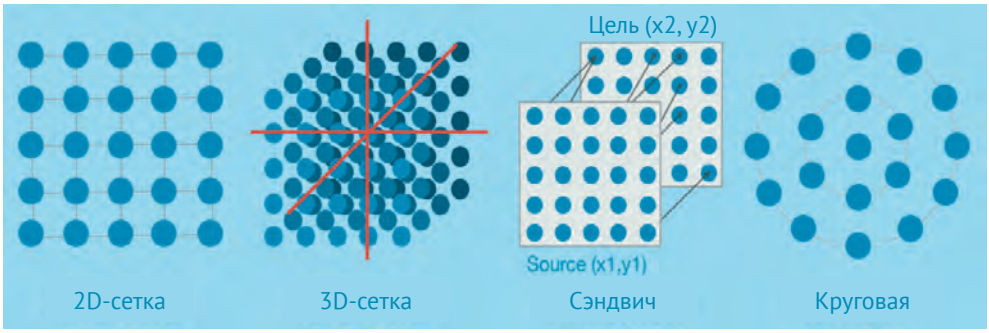

Рис. 1.9. Примеры конфигурации слоев субстрата

1.3.3 CPPN с развивающимися связями и алгоритм HyperNEAT

Метод называется HyperNEAT, потому что он использует модифицированный алгоритм NEAT для развития CPPN, представляющих объемные структуры в гиперпространстве. Каждая экспрессированная (доступная для генетических операций) точка паттерна, находящаяся в *гиперкубе* (hypercube), представляет собой связь между двумя узлами графа нижнего измерения (субстрата). Таким образом, размерность гиперпространства в два раза больше размерности нижележащего графа наименьшего измерения. Далее в главе 8 мы рассмотрим некоторые примеры, в которых используются двумерные паттерны связей.

Обобщенная форма алгоритма HyperNEAT представлена на рис. 1.10.

Алгоритм 2: Общая форма алгоритма HyperNEAT

```
begin
    1. Выбор нужной конфигурации субстрата (слои узлов и сопоставление входов/выходов);
    2. Инициализация популяции минимальной CPPN со случайными весами связей;
    repeat
        foreach организм ∈ популяция do
            3. Запросить CPPN организма о весе каждой возможной связи в субстрате,
               представляющем его фенотип. Если абсолютное значение выхода превышает
               пороговую величину, создать связь с весом, масштабированным
               пропорционально выходному значению;
            4. Использовать субстрат как фенотип ANN в целевой области, чтобы оценить
               пригодность найденного решения;
        5. Воспроизвести CPPN организмов в популяции при помощи NEAT;
    until решение найдено;
```

Рис. 1.10. Обобщенная форма алгоритма HyperNEAT

Любой ген связи или ген узла, который был добавлен к CPPN во время ее эволюции, приводит к открытию нового глобального измерения вариаций паттернов связей через субстрат фенотипа (новые признаки). Каждая модификация генома CPPN представляет новый способ изменения всего паттерна связей. Кроме того, ранее разработанные CPPN могут быть использованы в качестве основы для создания паттернов связей для субстрата с более высоким разрешением, чем то, которое использовалось для его обучения. Это позволяет нам получить рабочее решение прежней проблемы при любом разрешении, возможно, без ограничения верхнего предела. Вышеупомянутые свойства сделали HyperNEAT мощным инструментом для развития крупномасштабных искусственных нейронных сетей, имитирующих биологические объекты.

Больше информации о методе HyperNEAT вы можете найти по адресу **https://eplex.cs.ucf.edu/papers/stanley_alife09.pdf**.

1.4 HyperNEAT с развиваемым субстратом

Метод HyperNEAT основан на идее, что геометрические структуры живого мозга могут быть полноценно представлены искусственными нейронными сетями с узлами, расположенными в определенных пространственных местоположениях. Таким образом, нейроэволюция получает значительные преимущества и позволяет обучать крупномасштабные нейросети для сложных задач, что было невозможно с обычным алгоритмом NEAT. В то же время хотя подход HyperNEAT и основан на структуре естественного мозга, ему по-прежнему не хватает пластичности процесса естественной эволюции. Позволяя эволюционному процессу разрабатывать различные паттерны связей между узлами сети, метод HyperNEAT накладывает жесткое ограничение на то, где размещаются узлы. Экспериментатор должен определить расположение узлов сети с самого начала, и любое неверное предположение, сделанное исследователем, снизит эффективность эволюционного процесса.

Размещая сетевой узел в определенном месте на субстрате, экспериментатор создает непреднамеренное ограничение на паттерн весов, создаваемый CPPN. Это ограничение затем мешает CPPN, когда она пытается закодировать геометрические закономерности мира природы в топографию решающей нейросети (фенотип). В данном случае паттерн связей, создаваемый CPPN, должен идеально совпадать со слоем субстрата, который определен экспериментатором; возможны только связи между сетевыми узлами, которые определены заранее. Такое ограничение приводит к ненужным ошибкам аппроксимации, которые портят результат. Если позволить CPPN разрабатывать паттерны связей для узлов, которые находятся в немного других местах, это может оказаться более эффективным подходом.

1.4.1 Плотность информации в гиперкубе

Почему мы должны начинать с наложения подобных ограничений на расположение узлов? Разве не было бы лучше, если бы неявные подсказки, полученные из паттернов связей, стали бы указаниями к тому, где разместить следующий узел, чтобы лучше представить естественные закономерности физического мира?

Области с одинаковыми весами связей кодируют небольшое количество информации и, следовательно, имеют небольшое функциональное значение. В то же время области с огромными градиентами значений веса являются чрезвычайно информационными. Такие области могут получить выгоду от размещения дополнительных узлов сети для более точного кодирования природного процесса. Как вы помните из нашего обсуждения алгоритма HyperNEAT, связь между двумя узлами в субстрате можно представить в виде точки в четырехмерном гиперкубе. Главная идея предложенного алгоритма ES-HyperNEAT состоит в том, чтобы разместить больше гиперточек в тех областях гиперкуба, где обнаруживаются бо̀льшие вариации весов связей. В то же время в областях с меньшим разбросом весов связей размещается меньшее количество гиперточек.

Размещение узлов и образование связей между ними может быть продиктовано изменением веса связей, которые создаются эволюционирующей CPPN для данной области субстрата. Другими словами, для принятия решения о размещении следующего узла в субстрате нет необходимости в дополнительной информации, кроме той, что мы уже получаем от CPPN, кодирующей паттерны связей сети. Основным руководящим принципом для алгоритма определения топографии субстрата становится плотность информации.

Размещение узла в фенотипе нейросети отражает информацию, которая закодирована в паттернах связей, созданных CPPN.

1.4.2 Квадродерево как эффективный экстрактор информации

Для представления гиперточек, которые кодируют веса связей внутри гиперкуба, алгоритм ES-HyperNEAT использует квадродерево. *Квадродерево* – это древовидная структура данных, в которой каждый внутренний узел имеет ровно четыре дочерних узла. Эта структура данных была выбрана благодаря присущим ей свойствам, что позволяет ей представлять двумерные области на разных уровнях детализации. С помощью квадродерева можно организовать эффективный поиск в двухмерном пространстве, разбив любую интересующую область на четыре подобласти, и каждая из них становится листом дерева, а корневой (родительский) узел представляет исходную (декомпозированную) область.

Используя метод извлечения информации на основе квадродерева, метод ES-HyperNEAT итеративно ищет новые связи между узлами в двумерном пространстве субстрата нейросети, начиная с узлов ввода и вывода, которые были предварительно определены экспериментатором. Этот метод намного эффективнее в вычислительном отношении, чем поиск непосредственно в четырехмерном пространстве гиперкуба.

На рис. 1.11 показан пример извлечения информации с использованием квадродерева.

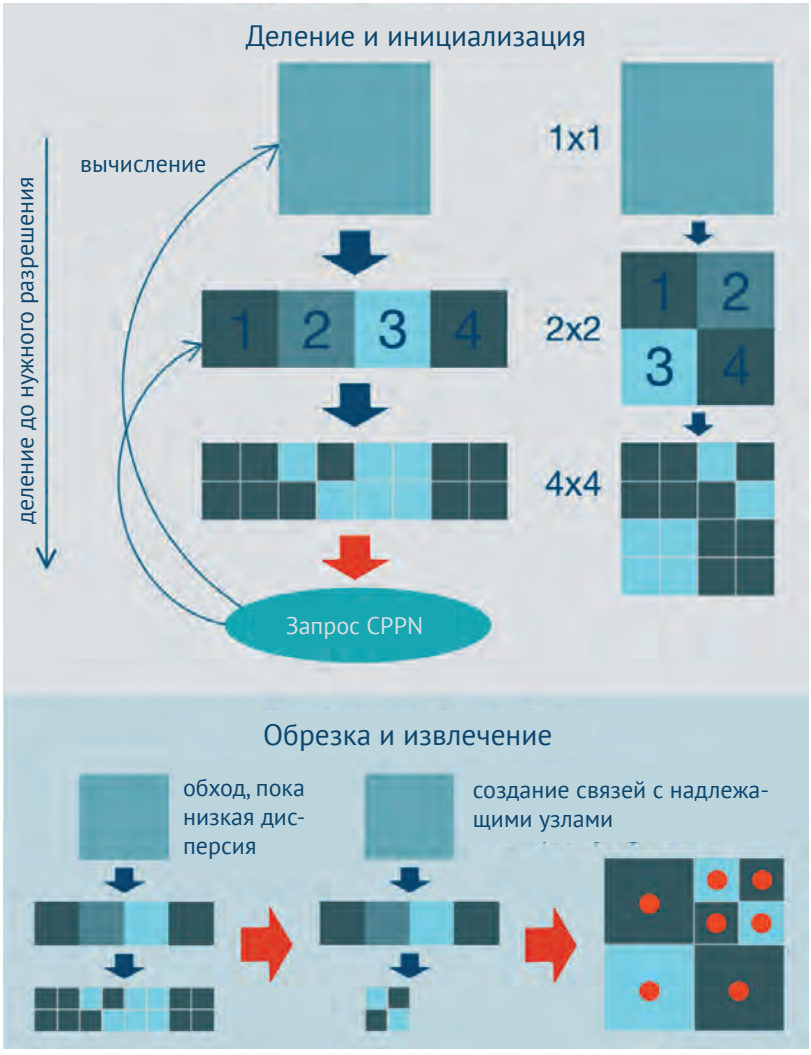

Рис. 1.11. Пример извлечения информации с использованием квадродерева

Алгоритм поиска на основе квадродерева работает в два основных этапа:

1. **Деление и инициализация**. На этом этапе квадродерево создается путем рекурсивного деления исходного пространства субстрата, занимающего область от (−1, −1) до (1, 1). Деление останавливается, когда достигается желаемая глубина дерева. От этого неявно зависит, сколько подпространств помещается в начальное пространство подложки (разрешение инициализации). После этого для каждого узла дерева с центром в (a, b) запрашивается CPPN с аргументами (a, b, x_i, y_i) для определения весов связей. Когда веса связей для k конечных узлов конкретного узла p квадродерева найдены, дисперсию этого узла можно рассчитать по следующей формуле:

$$\sigma^2 = \sum_{i=1}^{k} \left(\bar{\omega} - \omega_i \right)^2$$

Здесь $\bar{\omega}$ – средний вес соединения между k конечных узлов и ω_i – вес соединения с конкретным конечным узлом. Рассчитанное значение дисперсии является эвристическим индикатором наличия информации в конкретной области подложки. Если это значение выше, чем конкретный порог деления (определяющий желаемую плотность информации), то для соответствующего квадрата можно повторить стадию деления. Таким образом алгоритм может обеспечить требуемую плотность информации. Посмотрите на верхнюю часть рис. 1.11, чтобы увидеть, как выполняются деление и инициализация с использованием квадродерева.

2. **Обрезка и извлечение**. Чтобы гарантировать, что в областях с высокой плотностью информации (большой разброс весов) будет экспрессировано большее количество связей (и узлов в субстрате), для созданного на предыдущем этапе квадродерева выполняется процедура обрезки и извлечения. Для квадродерева выполняют поиск в глубину, пока дисперсия текущего узла не станет меньше порога дисперсии σ_t^2 или пока не окажется, что у узла нет дочерних элементов (нулевая дисперсия). Для каждого отвечающего этим условиям узла экспрессирована связь с соответствующим центром (x, y), и каждый родительский узел уже определен либо экспериментатором, либо найден при предыдущем запуске этих двух этапов (то есть среди скрытых узлов, которые уже были созданы методом ES-HyperNEAT). Фаза обрезки и извлечения наглядно представлена на нижней части рис. 1.11.

1.4.3 Алгоритм ES-HyperNEAT

Алгоритм ES-HyperNEAT начинается с определяемых пользователем входных узлов и подробно исследует исходящие из них связи, направленные ко вновь экспрессированным скрытым узлам. Экспрессия паттернов исходящих связей и размещения скрытых узлов в пространстве субстрата выполняется с использованием метода извлечения информации из квадродерева, который мы описали ранее. Процесс извлечения информации применяется итеративно до тех пор, пока не будет достигнут желаемый уровень плотности представления информации или пока в гиперкубе не перестанет обнаруживаться новая информация. После этого результирующая сеть соединяется с определенными пользователем выходными узлами путем экспрессии паттернов входящих подключений, направленных к выходам. Для этого мы также используем извлечение информации из квадродерева. В конечной (экспрессированной) сети остаются только те скрытые узлы, которые имеют путь как к входному, так и к выходному узлу.

Теперь у нас определено множество узлов и связей внутри субстрата фенотипа нейросети. Может быть выгодно удалить некоторые узлы из сети, введя дополнительную стадию обрезки. На этой стадии мы сохраняем только точки в пределах определенной полосы и удаляем точки на краю полосы. Делая полосы шире или уже, CPPN может управлять плотностью закодированной информации. Для получения более подробной информации о стадии обрезки,

пожалуйста, прочтите статью про ES-HyperNEAT (**https://eplex.cs.ucf.edu/papers/risi_alife12.pdf**).

Обобщенная форма алгоритма ES-HyperNEAT представлена на рис. 1.12.

Алгоритм 3: ES-HyperNEAT

```
Параметры : initialDepth, maxDepth, varianceThreshold, bandThreshold, iterationLevel,
            divisionThreshold
Вход      : CPPN, InputPositions, OutputPositions
Выход     : Connections, HiddenNodes

begin
    // Связи вход -> скрытый узел
    foreach input ∈ InputPositios do
        // Анализ паттерна исходящих связей
        root ⟵ DivisionAndInitialization(input.x, input.y, true);
        // Построение квадродерева и сохранение подходящих связей
        PruningAndExtraction(input.x, input.y, inputConnections, root, true);
        foreach c ∈ inputConnections do
            node ⟵ Node(c.x2, c.y2);
            if node ∉ HiddenNodes then
                HiddenNodes ⟵ HiddenNodes ∪ node;
    // Связи скрытый узел -> скрытый узел
    UnexploredHiddenNodes ⟵ HiddenNodes;
    for i ← 1 to iterationLevel do
        foreach hidden ∈ UnexploredHiddenNodes do
            root ⟵ DivisionAndInitialization(hidden.x, hidden.y, true);
            PruningAndExtraction(hidden.x, hidden.y, hidnConnections, root, true);
            foreach c ∈ hidnConnections do
                node ⟵ Node(c.x2, c.y2);
                if node ∉ HiddenNodes then
                    HiddenNodes ⟵ HiddenNodes ∪ node;
            // Удаление исследованных узлов
            UnexploredHiddenNodes ⟵ HiddenNodes − UnexploredHiddenNodes;
    // Связи скрытый узел -> выход
    foreach ouput ∈ OutputPositions do
        // Анализ входящих связей по отношению к выходу (output)
        root ⟵ DivisionAndInitialization(output.x, output.y, false);
        PruningAndExtraction(output.x, output.y, outputConnections, root, false);
        /* Здесь не создаются новые узлы, потому что все скрытые узлы, подключенные
           ко входу/выходу, уже экспрессированы */
    connections ⟵ inputConnections ∪ hidnConnections ∪ outputConnections;
    Удаление узлов и связей, не занятых в построении путей от входов к выходам.
```

Рис. 1.12. Обобщенная форма алгоритма ES-HyperNEAT

Алгоритм ES-HyperNEAT использует все преимущества методов NEAT и HyperNEAT и предоставляет еще более мощные новые функции, в том числе:

- автоматическое размещение скрытых узлов внутри субстрата для точного соответствия паттернам связи, которые выражены развитыми CPPN;
- позволяет нам гораздо проще создавать модульные нейросети фенотипа благодаря специфической способности сразу начать эволюционный поиск с уклоном в сторону локальности (благодаря особому устройству исходных архитектур CPPN);
- с помощью ES-HyperNEAT можно уточнить существующую структуру фенотипа нейросети, увеличив число узлов и связей в субстрате в процессе эволюции – в противоположность методу HyperNEAT, где количество узлов субстрата предопределено.

Алгоритм ES-HyperNEAT позволяет нам использовать исходную архитектуру HyperNEAT без изменения генетической структуры части NEAT. Это позволяет нам решать проблемы, которые трудно решить с помощью алгоритма HyperNEAT из-за сложностей с предварительным созданием соответствующей конфигурации субстрата.

Подробнее ознакомиться с алгоритмом ES-HyperNEAT и механизмами, лежащими в его основе, можно в статье по адресу **https://eplex.cs.ucf.edu/papers/risi_alife12.pdf.**

1.5 Метод оптимизации поиском новизны

Большинство методов машинного обучения, включая эволюционные алгоритмы, основывают свое обучение на оптимизации целевой функции. Основная идея, лежащая в основе методов оптимизации *целевой функции* (objective function), заключается в том, что лучший способ улучшить производительность *решателя* (solver) – вознаграждать его за приближение к цели. В большинстве эволюционных алгоритмов близость к цели измеряется *приспособленностью* (fitness) решателя. Мера полезности организма определяется *функцией приспособленности* (fitness function), которая является метафорой эволюционного давления на организм для адаптации к окружающей среде. Согласно этой парадигме, наиболее приспособленный организм лучше приспособлен к окружающей среде и лучше всего подходит для поиска решения.

Хотя методы прямой оптимизации функций приспособленности хорошо работают во многих простых случаях, при решении более сложных задач они часто становятся жертвами *ловушки локального оптимума*. Сходимость к локальному оптимуму означает, что ни один локальный шаг в пространстве поиска не обеспечивает каких-либо улучшений в процессе оптимизации функции приспособленности. Традиционные генетические алгоритмы используют мутационные и островные механизмы, чтобы вырваться из таких локальных оптимумов. Однако, как мы выясним позже, выполняя эксперименты в данной книге, эти подходы не всегда работают при решении обманчивых проблем, или может потребоваться слишком много времени, чтобы найти успешное решение.

Многие реальные проблемы имеют такие обманчивые ландшафты функций приспособленности, которые не могут быть успешно пройдены с помощью процесса оптимизации, основанного исключительно на измерении близости

текущего решения к цели. В качестве примера мы можем рассмотреть задачу навигации по неизвестному городу с нерегулярным рисунком улиц. В такой задаче движение к месту назначения часто означает путешествие по обманчивым дорогам, которые уводят вас еще дальше, только чтобы привести вас к месту назначения после нескольких поворотов. Но если вы решите начать с дорог, которые лучше всего направлены на пункт назначения, они могут завести вас в тупик, в то время как пункт назначения находится прямо за стеной, но недоступен.

1.5.1 Поиск новизны и естественная эволюция

Рассматривая, как естественный отбор работает в физическом мире, мы видим, что движущей силой эволюционного разнообразия является *поиск новизны* (novelty search, NS). Другими словами, любой развивающийся вид получает непосредственные эволюционные преимущества по сравнению с конкурентами, находя новые модели поведения. Это позволяет ему более эффективно использовать окружающую среду. У естественной эволюции нет определенных целей, и она расширяет пространство поиска решений, вознаграждая за исследование и использование новых форм поведения. Новизну можно рассматривать как точку приложения усилий для многих скрытых творческих сил в мире природы, что позволяет эволюции разрабатывать еще более сложные модели поведения и биологические структуры.

Вдохновившись естественной эволюцией, Джоэл Леман предложил новый метод поисковой оптимизации для искусственного эволюционного процесса, названный поиском новизны. При использовании этого метода для поиска решения не определяется и не используется никакая конкретная функция приспособленности; вместо этого новизна каждого найденного решения вознаграждается непосредственно в процессе нейроэволюции.

Таким образом, новизна найденных решений направляет нейроэволюцию к конечной цели. Такой подход дает нам возможность использовать творческие силы эволюции, не зависящие от адаптивного давления, стремящегося приспособить решение к определенной нише.

Эффективность поиска новизны может быть продемонстрирована с помощью эксперимента по навигации в лабиринте, где поиск на основе близости к цели находит решение для простого лабиринта за гораздо большее количество шагов (поколений), чем поиск новизны. Кроме того, в случае сложного лабиринта с дезориентирующей конфигурацией поиск на основе близости к цели вообще не может найти решение. Мы обсудим эксперименты по навигации в лабиринте в главе 5.

1.5.2 Метрика новизны

Алгоритм поиска новизны использует метрику новизны для отслеживания уникальности поведения каждого нового организма. То есть *метрика новизны* – это мера того, насколько далеко новый организм находится от остальной части популяции в *пространстве поведения* (behavior space). Эффективная реализация метрики новизны должна позволить нам вычислить *разреженность* (sparseness) в любой точке пространства поведения. Любая область с более плотным скоплением посещенных точек менее нова и дает меньшее эволюционное вознаграждение.

Наиболее простой мерой разреженности в точке является среднее расстояние до *k*-ближайших соседей этой точки в пространстве поведения. Когда это расстояние велико, интересующий объект находится в разреженной области. В то же время более плотным областям присущи меньшие значения расстояний. Таким образом, разреженность в точке вычисляется по следующей формуле:

$$\rho(x) = \frac{1}{k}\sum_{i=0}^{k} dist(x, \mu_i)$$

Здесь μ_i – это *i*-й ближайший сосед *x*, определенный по метрике расстояния dist(x_i, μ_i). Метрика расстояния является предметно-ориентированной мерой *поведенческой разницы* между двумя индивидуумами.

Кандидаты из разреженных областей получают более высокие оценки новизны. Когда эта оценка превышает некоторый минимальный порог ρ_{min}, индивидуум в этой точке добавляется в архив лучших исполнителей, характеризующий распределение предыдущих решений в пространстве поведения. Нынешнее поколение населения в сочетании с архивом определяет, где был поиск раньше и где он будет происходить сейчас. Таким образом, градиент поиска направляется на новое поведение без какой-либо явной цели, просто максимизируя показатель новизны. Тем не менее поиск новизны по-прежнему основывается на прикладной информации, потому что изучение нового поведения требует всестороннего исследования области поиска.

Обобщенная форма алгоритма поиска новизны показана на рис. 1.13.

Алгоритм 4: Метод поиска новизны

Параметры : noveltyThreshold
Вход : Population, NoveltyArchive
Результаты : Каждый организм в популяции оценивается по метрике новизны, исходя из данных в *NoveltyArchive* и других организмов в *Population*. Если его оценка превышает noveltyThreshold, он добавляется в *NoveltyArchive*. Значение *Fitness* каждого организма обновляется на основе оценки новизны. Далее приспособленность (*Fitness*) может повлиять на решение о том, какой из организмов заслуживает участия в репродукции.

```
begin
    foreach organism ∈ Population do
        // Вычисляем оценку новизны для организма (organism) на основе
        // сочетания популяции (population) и архива новинок (NoveltyArchive)
        novelty ⟵ AvgKnnDistance(organism, Population, NoveltyArchive);
        if novelty > noveltyThreshold then
            // Добавляем organism в NoveltyArchive
            NoveltyArchive ⟵ NoveltyArchive ∪ organism;
            // Удаляем из архива новинок записи с наименьшим значением
            // оценки новизны, чтобы сохранить размер архива.
            PurgeNoveltyArchive(NoveltyArchive)
        /* Оценка новизны, сохраненная в качестве меры приспособленности, пригодится
           на следующем шаге эволюционного алгоритма, таком, как репродукция, когда
           более приспособленные организмы имеют больше шансов оставить потомство. */
        organism.Fitness ⟵ novelty
```

Рис. 1.13. Обобщенная форма алгоритма поиска новизны

Метод оптимизации поиска новизны позволяет эволюции искать решения в любом дезориентирующем пространстве и находить оптимальные решения. С помощью этого метода возможно реализовать дивергентную эволюцию, когда популяция вынуждена не останавливаться на конкретном нишевом решении (локальный оптимум) и должна исследовать все пространство решений. Как показали эксперименты, это очень эффективный метод поисковой оптимизации, несмотря на его нелогичный подход, который полностью игнорирует явную цель во время поиска. Более того, он может найти окончательное решение в большинстве случаев даже быстрее, чем традиционный поиск на основе целей, измеряющий приспособленность как расстояние от окончательного решения.

Подробное описание алгоритма можно найти по адресу **http://joellehman.com/lehman-dissertation.pdf**.

1.6 Заключение

В этой главе мы начали с обсуждения различных методов, которые используются для обучения искусственных нейронных сетей. Мы рассмотрели, чем традиционные методы градиентного спуска отличаются от методов, основанных на нейроэволюции. Затем мы представили один из самых популярных алгоритмов нейроэволюции (NEAT) и два способа его расширения (HyperNEAT и ES-HyperNEAT). Наконец, вы познакомились с методом поисковой оптимизации (поиск по новизне), способным найти решение для множества дезориентирующих проблем, которые не могут быть решены с помощью традиционных методов поиска на основе целей. Теперь вы готовы применить эти знания на практике после настройки необходимой среды моделирования, которую мы обсудим в следующей главе.

В следующей главе мы также рассмотрим доступные библиотеки, позволяющие проводить эксперименты с нейроэволюцией в виде кода на языке Python. Вы узнаете, как настроить рабочую среду и какие инструменты пригодятся для управления зависимостями в экосистеме Python.

1.7 Дополнительное чтение

Для более глубокого понимания тем, которые мы обсуждали в этой главе, изучите дополнительные материалы по следующим ссылкам:

- **NEAT**: **http://nn.cs.utexas.edu/downloads/papers/stanley.phd04.pdf**;
- **HyperNEAT**: **https://eplex.cs.ucf.edu/papers/stanley_alife09.pdf**;
- **ES-HyperNEAT**: **https://eplex.cs.ucf.edu/papers/risi_alife12.pdf**;
- **Novelty Search**: **http://joellehman.com/lehman-dissertation.pdf**.