ОГЛАВЛЕНИЕ

Глава 1. Оптимизация гиперпараметров

Краткое содержание

Недавний интерес к сложным и дорогостоящим в вычислительном отношении моделям машинного обучения с большим количеством гиперпараметов, таким как фреймворки автоматизированного машинного обучения (AutoML) и глубокие нейронные сети, привел к возрождению исследований по оптимизации гиперпараметров (Hyperparameter Optimization - HPO).

В этой главе мы приводим обзор наиболее известных подходов к HPO. Сначала мы обсудим методы оптимизации функций черного ящика, основанные на безмодельных методах и байесовской оптимизации. Поскольку высокая вычислительная потребность многих современных приложений машинного обучения делает оптимизацию чистого черного ящика чрезвычайно дорогостоящей, далее мы сосредоточимся на современных методах с высокой точностью, которые используют (намного) более дешевые варианты функции черного ящика для приблизительной оценки качества настроек гиперпараметров. Наконец, мы посмотрим на нерешенные задачи и будущие направления исследований.

1.1 Введение

Каждая система машинного обучения имеет гиперпараметры, и самая основная задача в автоматизированном машинном обучении (AutoML) заключается в автоматической настройке этих гиперпараметров для оптимизации производительности. В особенности относительно недавно появившиеся глубокие нейронные сети в решающей степени зависят от широкого спектра гиперпараметрических вариантов архитектуры, регуляризации и оптимизации нейронной сети.

Автоматическая оптимизация гиперпараметров (HPO) имеет несколько важных вариантов использования:

- сократить человеческие усилия, необходимые для применения машинного обучения. Это особенно важно в контексте AutoML;
- повысить производительность алгоритмов машинного обучения (путем адаптации их к решаемой задаче);
- улучшить воспроизводимость и объективность научных исследований. Автоматический HPO явно более воспроизводим, чем ручной поиск.

Проблема оптимизации гиперпараметров имеет долгую историю, уходящую корнями к 1990-м годам. Также достаточно рано было установлено, что для разных наборов данных лучше работают различные конфигурации гиперпараметров. В отличие от этого, относительно недавним является понимание того, что HPO можно использовать для адаптации конвейеров общего назначения к конкретным областям деятельности. В настоящее время также широко признано, что настроенные гиперпараметры работают лучше, чем настройки по умолчанию, предоставляемые распространенными библиотеками машинного обучения.

Из-за растущего использования машинного обучения в компаниях, HPO также представляет значительный коммерческий интерес и играет там все большую роль, будь то во внутренних инструментах компании, как часть облачных сервисов машинного обучения или как SaaS сама по себе.

HPO сталкивается с рядом проблем, которые делают ее трудной задачей на практике:

• Оценка функций может быть чрезвычайно дорогостоящей для больших моделей (например, при глубоком обучении), сложных конвейеров машинного обучения или больших наборов данных.

• Пространство конфигурации часто является сложным (включающим смесь непрерывных, категориальных и условных гиперпараметров) и многомерным. Кроме того, не всегда ясно, какие из гиперпараметров алгоритма необходимо оптимизировать и в каких диапазонах.

• Обычно у нас нет доступа к градиенту функции потерь по отношению к гиперпараметрам. Кроме того, другие свойства целевой функции, часто используемые в классической оптимизации, такие как выпуклость и гладкость, обычно неприменимы.

• Невозможно напрямую оптимизировать обобщающую способность, поскольку обучающие наборы данных имеют ограниченный размер.

Эта глава структурирована следующим образом. Сначала мы формально определим проблему HPO и обсудим ее варианты (Раздел 1.2). Затем мы обсудим алгоритмы оптимизации черного ящика для решения HPO (Раздел 1.3). Далее мы сосредоточимся на современных методах с высокой точностью, которые позволяют использовать HPO даже для очень дорогих моделей, используя приблизительные показатели производительности, которые дешевле, чем полная оценка модели (Раздел 1.4). Затем мы приведем обзор наиболее важных систем

гиперпараметрической оптимизации и приложений для AutoML (Раздел 1.5) и закончим главу обсуждением открытых проблем (Раздел 1.6).

1.2 Постановка задачи

Пусть A обозначает алгоритм машинного обучения с N гиперпараметрами. Обозначим через Λ_n область значений n-го гиперпараметра, а общее пространство конфигураций гиперпараметров как $\boldsymbol{\Lambda} = \Lambda_1 \times \Lambda_2 \times \ldots \Lambda_N$. Вектор гиперпараметров обозначается как $\lambda \in \boldsymbol{\Lambda}$, а алгоритм A инициализированный вектором λ, обозначим как A_λ.

Область значений гиперпараметра может быть вещественной (например, скорость обучения), целочисленной (например, количество слоев), бинарной (например, использовать ли раннюю остановку или нет) или категориальные (например, выбор оптимизатора). Для целочисленных и вещественных гиперпараметров возможные области значений в основном ограничены по практическим причинам, за некоторыми исключениями [12, 113, 136].

Кроме того, конфигурационное пространство может включать обусловленность, т. е. гиперпараметр является релевантным только в том случае, если другой гиперпараметр (или некоторая комбинация гиперпараметров) принимает определенное значение. Обусловленные пространства гиперпараметров принимают вид ориентированных ациклических графов. Такие обусловленные пространства встречаются, например, в автоматизированной настройке конвейеров машинного обучения, где выбор между различными алгоритмами предварительной обработки и машинного обучения моделируются как категориальные гиперпараметры, задача известная как полный выбор модели (Full Model Selection - FMS) или как задача комбинированного выбора алгоритма и оптимизации гиперпараметров (Combined Algorithm Selection and Hyperparameter optimization - CASH) [30, 34, 83, 149]. Такие пространства возникают также при оптимизации архитектуры нейронной сети: например, количество слоев может быть целочисленным гиперпараметром, а гиперпараметры для слоя i будут активны только в том случае, если глубина сети не менее i [12, 14, 33].

Имея набор данных D, наша цель состоит в том, чтобы найти

$$\lambda^* = \underset{\lambda \in \boldsymbol{\Lambda}}{\operatorname{argmin}} E(D_{train}, D_{valid})_{\sim D}\, V(L, A_\lambda, D_{train}, D_{valid}) \quad (1.1)$$

где функция $V(L, A_\lambda, D_{train}, D_{valid})$ измеряет потери модели, сгенерированной алгоритмом A с гиперпараметрами λ на обучающих данных D_{train} и проверенной на тестовых данных D_{valid}. На практике у нас есть доступ только к конечному набору данных $D \sim D$ и, следовательно, необходимо аппроксимировать ожидание в уравнении 1.1.

Популярные варианты функции проверки $V(\cdot, \cdot, \cdot, \cdot)$ включают использование контрольной выборки и перекрестной проверки ошибки для заданной пользователем функции потерь (например, коэффициент ошибочной классификации);
см. Bischl et al. [16] для обзора вариантов такой проверки. Также было предложено несколько стратегий для сокращения времени оценки: можно проверять алгоритмы машинного обучения только на подмножестве разбиений для кроссвалидации [149], только на подмножестве данных [78, 102, 147], либо использовать только небольшое количество итераций; мы обсудим некоторые из этих стратегий более подробно в разд. 1.4. Недавняя работа по многозадачной [147] и многоисточниковой [121] оптимизации вводит дополнительные вычислительно дешевые вспомогательные задачи, которые можно вычислить вместо уравнения 1.1. Они могут предоставить вычислительно дешево информацию, для помощи HPO, но при этом не обязательно тренируют модель машинного обучения на интересующем наборе данных и поэтому не дают пригодную для использования модель в качестве побочного продукта.

1.2.1 Альтернативы оптимизации: Оркестровка и маргинализация

Решение вышеуказанного уравнения с помощью одного из методов, описанных в остальной части этой главы, обычно требует подгонки алгоритма машинного обучения A к нескольким гиперпараметрическим векторам λ_t. Вместо применения к ним argmin оператора можно либо построить ансамбль (который направлен на минимизацию потерь для заданного протокола проверки), либо интегрировать все гиперпараметры (если рассматриваемая модель является вероятностной моделью). Мы ссылаемся на Guyon et al. и ссылки в нем для сравнения выбора частотной и байесовской моделей.

Выбор только одной конфигурации гиперпараметров может быть слишком расточительно, в случае когда HPO идентифицировало много хороших конфигураций, а объединение их в ансамбль может повысить производительность. Это особенно полезно в AutoML системах с большим пространством конфигурации (например, в задачах FMS или CASH), где хорошие конфигурации могут быть очень разнообразными, что увеличивает потенциальную выгоду от объединения. Программа Automatic Frankensteining (Автоматический Франкенштейнинг) в целях дальнейшего повышения производительности использует HPO для обучения модели суммирования используя в качестве входных данных результаты моделей, также найденных с помощью HPO; затем модели 2-го уровня объединяются с использованием традиционной стратегии оркестровки.

Методы, обсуждавшиеся до сих пор, применяли оркестровку после процедуры HPO. Хотя на практике они повышают производительность, базовые модели в этом случает не оптимизированы под оркестровку. Однако также возможна прямая оптимизация для моделей, которая максимально улучшила бы существующий ансамбль.

Наконец, при работе с байесовскими моделями часто можно объединять гиперпараметры алгоритма машинного обучения, например, используя максимизацию доказательств, усреднение байесовской модели, выборку среза или эмпирический Байес.

1.2.2 Многокритериальная оптимизация

В практических приложениях часто необходимо найти компромисс между двумя или более целями, такими как производительность модели и потребление ресурсов или функции множественных потерь. Потенциальные решения могут быть получены двумя способами. Во-

первых, если известно ограничение на вторичный показатель производительности (например, максимальное потребление памяти), проблема может быть сформулирована как задача ограниченной оптимизации. Мы обсудим использование ограничений в байесовской оптимизации в разд. 1.3.2.4 Во-вторых, и в более общем плане, можно применить многоцелевую оптимизацию для поиска фронта Парето, набора конфигураций, которые являются оптимальным компромиссом между целями в том смысле, что для каждой конфигурации на фронте Парето нет другой конфигурации, которая работает лучше, по крайней мере, для одной цели, и не хуже для всех других целей. Затем пользователь может выбрать конфигурацию с помощью фронта Парето.

1.3 Оптимизация гиперпараметров черного ящика

В общем случае, любой метод оптимизации черного ящика может быть применен к HPO. Из-за невыпуклой природы проблемы обычно предпочтительны алгоритмы глобальной оптимизации, но некоторая локальность в процессе оптимизации полезна для того, чтобы добиться прогресса в рамках оценок нескольких функций, которые обычно доступны. Сначала мы обсудим методы HPO черного ящика без моделей, а затем опишем методы байесовской оптимизации черного ящика.

1.3.1 Методы оптимизации черного ящика без моделей

Поиск по сетке - это самый базовый метод HPO, также известный как полный факториальный дизайн. Пользователь задает конечный набор значений для каждого гиперпараметра, и поиск по сетке производит поиск по декартову произведению этих наборов. Это решение страдает от проклятия размерности, поскольку требуемое количество оценок функций растет экспоненциально с увеличением размерности конфигурационного пространства. Дополнительная проблема поиска по сетке заключается в том, что увеличение разрешения дискретизации существенно увеличивает требуемое количество оценок функций.

Простой альтернативой поиску по сетке является случайный поиск. Как следует из названия, случайный поиск выбирает конфигурации случайным образом до тех пор, пока не будет исчерпан определенный бюджет на поиск. Этот способ работает лучше, чем поиск по сетке, в случае, когда некоторые гиперпараметры намного важнее других (свойство, которое выполняется во многих случаях). Интуитивно понятно, что при

поиске с фиксированным бюджетом при котором можно выполнить только B оценок функции, количество различных значений, которые поиск по сетке может позволить себе оценить для каждого из N гиперпараметров является только $B^{1/N}$, тогда как случайный поиск будет исследовать B различных значений для каждого; см. рисунок ниже.

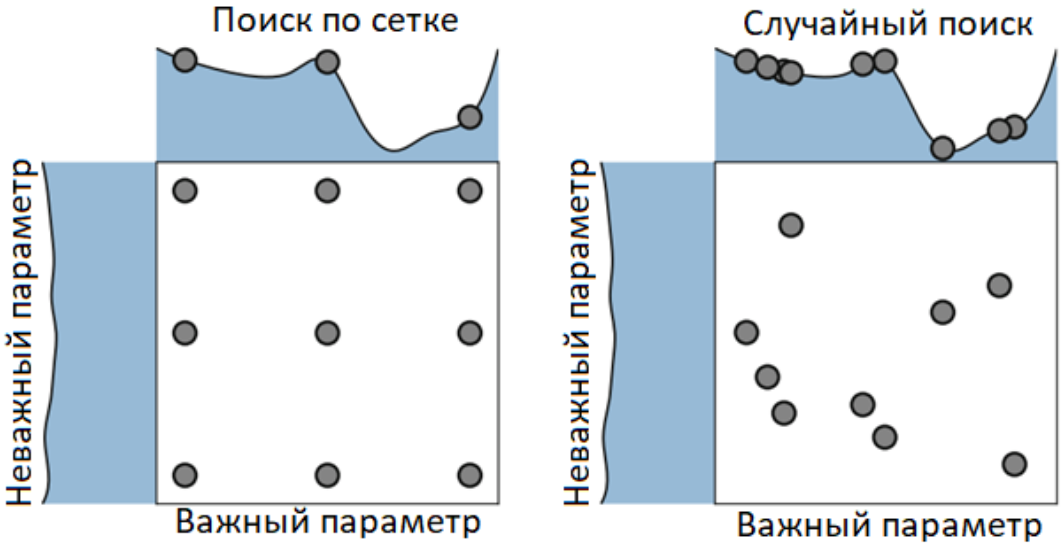

Рис 1. Сравнение поиска по сетке и случайного поиска для минимизации функции с одним важным и одним неважным параметром. Этот рисунок основан на иллюстрации на рис. 1 Бергстра и Бенжио

Дополнительные преимущества по сравнению с поиском по сетке включают более простое распараллеливание (поскольку рабочим процессам не нужно взаимодействовать друг с другом, а отказывающие рабочие процессы не оставляют дыр в дизайне) и гибкое распределение ресурсов (поскольку можно добавить произвольное количество случайных точек к дизайну случайного поиска, чтобы все равно получить дизайн случайного поиска; но для поиска по сетке то же самое неверно).

Случайный поиск является полезной отправной точкой, поскольку он не делает никаких предположений относительно оптимизации алгоритма машинного обучения и, при наличии достаточного количества ресурсов, ожидаемо достижение им производительности, сколь угодно близкой к оптимальной. Таким образом, чередование случайного поиска с более сложными стратегиями оптимизации позволяет гарантировать достижение минимального значения сходимости, а также добавляет исследование областей, с помощью которых можно улучшить поиск на основе модели. Случайный поиск также является полезным методом для инициализации процесса поиска, поскольку он исследует все пространство конфигурации

и, таким образом, часто находит настройки с приемлемой производительностью. Однако это не универсальное решение, часто ему требуется гораздо больше времени, чем методам управляемого поиска, для того чтобы определить одну из наиболее эффективных конфигураций гиперпараметров: например, при выборке без замены из пространства конфигурации с N булевыми гиперпараметрами с хорошей и плохой настройками для каждого и без эффектов взаимодействия, ожидаемо потребуется 2^{N-1} оценки функции для нахождения оптимума, в то время как управляемый поиск мог бы найти оптимум за N + 1 оценок функции следующим образом: начиная с произвольной конфигурации, следует совершить проход по гиперпараметрам и изменять по одному за раз, сохраняя получившуюся конфигурацию, если производительность улучшилась, и откатывая изменение, если улучшения не произошло.

Методы, основанные на популяции, такие как генетические алгоритмы, эволюционные алгоритмы, эволюционные стратегии и оптимизация роя частиц, представляют собой алгоритмы оптимизации, которые поддерживают популяцию, т.е. набор конфигураций, и улучшают эту популяцию путем применения локальных возмущений (так называемых мутаций) и комбинаций различных членов популяции (так называемый кроссовер) для получения нового поколения улучшенных конфигураций. Эти методы концептуально просты, могут обрабатывать различные типы данных, и параллельны, поскольку совокупность из N членов может быть оценена параллельно на N машинах.

Одним из наиболее известных методов, основанных на популяции, является эволюционная стратегия адаптации ковариационной матрицы (CMA-ES); эта простая эволюционная стратегия выбирает конфигурации из многомерной гауссовой модели, среднее значение которой и ковариация обновляются в каждом поколении на основе успеха особей популяции. CMA-ES является одним из наиболее конкурентоспособных алгоритмов оптимизации blackbox, регулярно доминирующим в соревновании по оптимизации черного ящика (Black-Box Optimization Benchmarking - BBOB).

1.3.2 Байесовская оптимизация

Байесовская оптимизация - это современная архитектура оптимизации для глобальной оптимизации дорогостоящих функций «черного ящика», которая недавно получила распространение в HPO благодаря получению новых современных результатов в настройке глубоких нейронных сетей для классификации изображений, распознавания речи и нейронных сетей языкового моделирования, а также демонстрируя широкую применимость к задачам в различных областях. Для подробного ознакомления с байесовской оптимизацией отсылаем к превосходным учебным пособиям Шахриари и Броху.

В этом разделе мы сначала даем краткое введение в байесовскую оптимизацию, представляем альтернативные суррогатные модели, используемые в ней, описываем расширения для условных и ограниченных конфигурационных пространств, а затем обсуждаем несколько важных приложений для оптимизации гиперпараметров.

Многие недавние достижения в области байесовской оптимизации больше не рассматривают HPO как «черный ящик», например, многофункциональный HPO (см. раздел 1.4), байесовская оптимизация с метаобучением (см. главу 2) и байесовская оптимизация с учетом структуры конвейера. Кроме того, многие недавние разработки в области байесовской оптимизации не нацелены непосредственно на HPO, но часто могут быть легко применены к HPO, такие как новые функции сбора данных, новые модели и ядра, а также новые схемы распараллеливания.

1.3.2.1 Байесовская оптимизация в кратком изложении

Байесовская оптимизация – это итеративный алгоритм с двумя ключевыми компонентами: вероятностной суррогатной моделью и функцией сбора данных, позволяющей решить, какую точку оценивать следующей. На каждой итерации суррогатная модель подгоняется ко всем наблюдениям целевой функции, сделанным до сих пор. Затем функция сбора данных, которая использует прогнозирующее распределение вероятностной модели, определяет полезность различных точек-кандидатов, комбинируя исследования и использование. По сравнению с оценкой дорогостоящей функции «черного ящика», функция сбора данных обходится дешевле для вычисления и, следовательно, может быть тщательно оптимизирована.

Хотя существует много функций сбора данных, ожидаемое улучшение (expected improvement - EI):

$$E[ℸ(\lambda)] = E(\max(f_{min} - y, 0)] \quad (1.2)$$

является распространенным выбором, поскольку его можно вычислить в закрытой форме, если прогноз модели y для конфигурации λ соответствует нормальному распределению:

$$E[ℸ(\lambda)] = (f_{min} - \mu(\lambda))\Phi\left(\frac{f_{min} - \mu(\lambda)}{\sigma}\right) + \sigma\varphi\left(\frac{f_{min} - \mu(\lambda)}{\sigma}\right) \quad (1.3)$$

где φ(·) и Φ(·) представляют собой функции стандартной нормальной плотности и стандартного нормального распределения, а f_{min} – наилучшее наблюдаемое значение на данный момент.

На рис. 1.2 показана байесовская оптимизация, оптимизирующая игрушечную функцию.

1.3.2.2 Сурогатные модели (Surrogate Models)

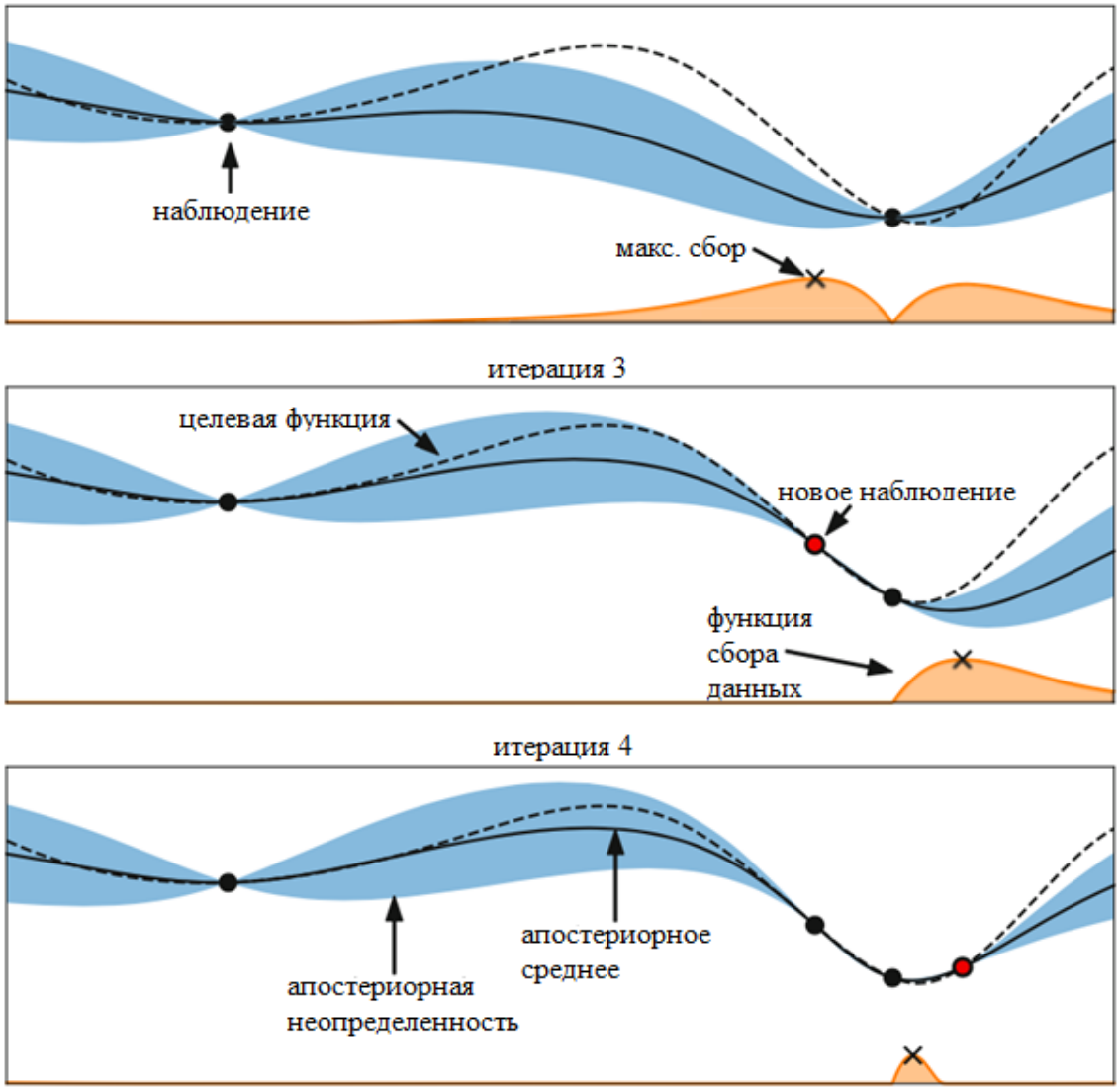

Рис. 1.2 Иллюстрация байесовской оптимизации для одномерной функции. Наша цель состоит в том, чтобы минимизировать пунктирную линию, используя суррогат гауссовского процесса (прогнозы показаны черной линией, а синяя область представляет неопределенность), максимизируя функцию сбора данных, представленную нижней оранжевой кривой. (Верх) Значение сбора данных низкое вокруг наблюдений, а самое высокое значение сбора данных находится в точке, где значение прогнозируемой функции низкое, а неопределенность прогнозирования относительно высока. (Посередине) В то время как слева от нового наблюдения все еще существует большая дисперсия, прогнозируемое среднее значение справа намного ниже, и следующее наблюдение проводится там. (Внизу) Хотя неопределенности относительно местоположения истинного максимума почти не осталось, следующая

оценка выполняется там из-за ожидаемого улучшения по сравнению с лучшей точкой на данный момент.

Традиционно байесовская оптимизация использует гауссовы процессы для моделирования целевой функции из-за их выразительности, гладких и хорошо откалиброванных оценок неопределенности и вычислимости прогнозного распределения в замкнутой форме.

Гауссовский процесс G $(m(\lambda), k(\lambda, \lambda')$ полностью определяется средним значением m(λ) и ковариационной функцией k(λ, λ'), хотя обычно предполагается, что средняя функция постоянна. в байесовской оптимизации. Прогнозы среднего значения и дисперсии μ(·) и $\sigma^2(\cdot)$ для случая без шума могут быть получены следующим образом:

$$\mu(\lambda) = \mathbf{k}^{\mathrm{T}} * \mathbf{K}^{-1}\mathbf{y}, \quad \sigma^2(\lambda) = k(\lambda, \lambda) - \mathbf{k}^{\mathrm{T}}_{*} \mathbf{K}^{-1}\mathbf{k}_{*}, \tag{1.4}$$

где k_* обозначает вектор ковариаций между λ и всеми предыдущими наблюдениями, K – ковариационная матрица всех ранее оцененных конфигураций, а y – наблюдаемые значения функции. Качество гауссовского процесса зависит исключительно от функции ковариации. Распространенным выбором является ядро Mátern 5/2 с его гиперпараметрами, интегрированными с помощью Марковской цепи Монте-Карло.

Одним из недостатков стандартных гауссовских процессов является то, что они масштабируются кубически по количеству точек данных, что ограничивает их применимость, когда можно позволить себе много вычислений функций (например, при большом количестве параллельных исполнителей или когда вычисления функций дешевы из-за использования более низкой точности). Этого кубического масштабирования можно избежать с помощью масштабируемых приближений гауссовского процесса, таких как разреженные гауссовы процессы. Они аппроксимируют полный гауссовский процесс, используя только подмножество исходного набора данных в качестве исходных точек для построения матрицы ядра K. Хотя они позволили байесовской оптимизации с GPs масштабироваться до десятков тысяч точек данных для оптимизации параметров рандомизированного SAT-решателя, есть критические замечания по поводу калибровки их оценок неопределенности, и их применимость к стандартному HPO не была протестирована.

Другим недостатком гауссовых процессов со стандартными ядрами является их плохая масштабируемость до больших размеров. В результате было предложено множество расширений для эффективной обработки внутренних свойств конфигурационных пространств с большим количеством гиперпараметров, таких как использование случайных вложений, использование гауссовых процессов на разбиениях конфигурационного пространства, цилиндрических ядер и аддитивных ядер.

Поскольку некоторые другие модели машинного обучения являются более масштабируемыми и гибкими, чем гауссовские процессы, также проводится большое количество исследований по адаптации этих моделей к байесовской оптимизации. Во-первых, (глубокие) нейронные сети – это очень гибкие и масштабируемые модели. Самый простой способ применить их к байесовской оптимизации – это использовать их в качестве экстрактора признаков для предварительной обработки входных данных, а затем использовать выходные данные конечного скрытого слоя в качестве базовых функций для байесовской линейной регрессии. Более сложная, полностью байесовская обработка весов сети также возможна при использовании байесовской нейронной сети, обученной с помощью стохастического градиента Гамильтониана Монте-Карло. Нейронные сети, как правило, быстрее, чем гауссовы процессы для байесовской оптимизации после $\sim$250 вычислений функций, что также допускает крупномасштабный параллелизм. Гибкость глубокого обучения также может обеспечить байесовскую оптимизацию для более сложных задач. Например, вариационный автокодер может вычислить функции для встраивания сложных входных данных (таких как структурированные конфигурации автоматизированного статистика, см. главу 9) в вектор с вещественным значением, так чтобы обычный гауссовский процесс мог обрабатывать его. Для байесовской оптимизации с несколькими источниками архитектура нейронной сети, построенная на машинах факторизации, может включать информацию о предыдущих задачах, а также быть расширена для решения проблемы CASH.

Другой альтернативной моделью байесовской оптимизации являются случайные леса. В то время как GPs (гауссовы процессы) работают лучше, чем случайные леса, на небольшой числовой конфигурации пространств , случайные леса изначально обрабатывают большие, категориальные и

условные конфигурационные пространства, где стандартные GPs (гауссовы процессы) плохо работают. Более того, вычислительная сложность случайных лесов намного лучше масштабируется по многим точкам данных: в то время как вычислительная сложность подбора и прогнозирования дисперсий с GPs (гауссовыми процессами) для n точек данных масштабируется как O (n3) и O (n2), соответственно, для случайных лесовы, масштабирование по n составляет только O(n log n) и O(log n) соответственно. Благодаря этим преимуществам фреймворк SMAC для байесовской оптимизации со случайными лесами позволил использовать известные фреймворки AutoML Auto-WEKA и Auto-sklearn (которые описаны в главах. 4 и 6).

Вместо моделирования вероятности p(y|λ) наблюдений y при заданных конфигураций λ дерева оценок Парзена (TPE) моделирует функции плотности p(λ|y < α) и p(λ|y ≥ α). Учитывая процентиль α (обычно устанавливается равным 15%), наблюдения делятся на хорошие наблюдения и плохие наблюдения, и для моделирования двух распределений используются простые одномерные окна Парзена. Отношение $p(\lambda|y<\alpha)/p(\lambda|y \geq \alpha)$ связано с функцией получения ожидаемых улучшений и используется для предложения новых конфигураций гиперпараметров. TPE использует дерево оценок Парзена для условных гиперпараметров и продемонстрировал хорошую производительность на структурированных задачах HPO, является концептуально простым и естественным образом распараллеливается. Он также лежит в основе фреймворка AutoML Hyperoptsklearn (который описан в главе 5).

Наконец, отметим, что существуют также подходы, основанные на суррогатах, которые не следуют парадигме байесовской оптимизации: Hord использует детерминированный суррогат RBF, а Harmonica использует метод сжатого зондирования, оба для настройки гиперпараметров глубоких нейронных сетей.

1.3.2.3 Описание конфигурационного пространства (Configuration Space Description

Байесовская оптимизация изначально была разработана для оптимизации функций с вещественными значениями, ограниченных рамками. Однако для многих гиперпараметров машинного обучения, таких

как скорость обучения в нейронных сетях или регуляризация в машинах опорных векторов, обычно оптимизируют показатель степени экспоненциального члена, чтобы описать, что его изменение, например, с 0,001 до 0,01, будет иметь такое же большое влияние, как и изменение с 0,1 до 1. от 0,1 до 1. Метод, известный как искажение входных данных, позволяет автоматически изучать такие преобразования в процессе оптимизации, заменяя каждое входное измерение двумя параметрами бета-распределения и оптимизируя их.

Одним из очевидных ограничений box-constraints (ограничения рамки, определяют верхнюю и нижнюю границы веса активов.) является то, что пользователь должен определить их заранее. Чтобы избежать этого, можно динамически расширять пространство конфигурации. В качестве альтернативы, алгоритм в стиле оценки распределения TPE способен работать с бесконечными пространствами, на которые помещается (обычно гауссовский) априор.

Целые числа и категориальные гиперпараметры требуют специальную обработку, но их можно довольно легко интегрировать в обычную байесовскую оптимизацию путем небольшой адаптации ядра и процедуры оптимизации (см. раздел 12.1.2 в [58], а также [42]). Другие модели, такие как машины факторизации и случайные леса, также естественным образом могут обрабатывать эти типы данных.

Условные гиперпараметры по-прежнему являются активной областью исследований (см. главы 5 и 6 для описания пространств условной конфигурации в последних системах AutoML). Они могут быть обработаны изначально древовидными методами, такими как случайные леса и оценкой Парзена на деревьях (TPE), но из-за многочисленных преимуществ гауссовских процессов по сравнению с другими моделями также было предложено несколько ядер для структурированных конфигурационных пространств.

1.3.2.4 Байесовская оптимизация с ограничениями (Constrained Bayesian Optimization)

В реалистичных сценариях часто необходимо удовлетворять ограничениям, таким как потребление памяти, время обучения, время прогнозирования, точность сжатой модели, потребление энергии или просто не допускать сбоев во время процедуры обучения.

Ограничения могут быть скрыты тем, что доступно только бинарное наблюдение (успех или неудача). Типичными примерами в AutoML являются ограничения по памяти и времени, позволяющие обучать алгоритмы в общей вычислительной системе и гарантировать, что конфигурация одного медленного алгоритма не использует все время, доступное для HPO (см. также главы. 4 и 6).

Ограничения также могут быть просто неизвестны, что означает, что мы можем наблюдать и моделировать вспомогательную функцию ограничения, но узнаем о нарушении ограничения только после оценки целевой функции. Примером этого является время прогнозирования машины опорных векторов, которое может быть получено только путем ее обучения, поскольку оно зависит от количества опорных векторов, выбранных во время обучения.

Простейший подход к моделированию нарушенных ограничений заключается в определении значение штрафа (по крайней мере, такого же, как наихудшее из возможных наблюдаемых значений потерь) и использовать его в качестве наблюдения за неудачными запусками. Более продвинутые подходы моделируют вероятность нарушения одного или нескольких ограничений и активно ищут конфигурации с низкими значениями потерь, которые вряд ли нарушат какое-либо из заданных ограничений.

Фреймворки байесовской оптимизации, использующие теоретико-информационные функции сбора данных, позволяют отделить оценку целевой функции от ограничений, чтобы динамически выбирать, какие из них оценивать следующими. Это становится выгодным, когда оценка интересующей функции и ограничений требует совершенно разного количества времени, например, при оценке производительности глубокой нейронной сети и потреблении памяти.

1.4 Оптимизация с мульти-точностью

Увеличивающиеся размеры датасетов и все более сложные модели являются серьезным препятствием в HPO, поскольку это приводит к проблемам с производительностью для модели черного ящика. Подбор одной конфигурации гиперпараметров на больших наборах данных в настоящее время может легко превышать несколько часов и занимать до нескольких дней.

Таким образом, распространенные методы ускорения подбора заключаются в проверке конфигурации алгоритма/гиперпараметра на небольшом подмножестве данных, его обучении только на нескольких итерациях, использовании только подмножества признаков, одной или нескольких частей при кросс-валидации, изображений с пониженной дискретизацией в компьютерном зрении. Методы с мульти-точностью превращают такие ручные эвристики в формальные алгоритмы, используя так называемые аппроксимации с низкой точностью (low-fidelity approximations) для минимизации фактической функции потерь. Эти приближения вводят компромисс между точностью оптимизации и временем выполнения, при этом на практике полученное ускорение часто перевешивает ошибку аппроксимации.

Сначала мы рассмотрим методы, которые в процессе работы моделируют кривую обучения алгоритма и могут остановить процедуру обучения, если прогнозируется, что использование дополнительных ресурсов не поможет. Во-вторых, мы обсудим простые методы, которые выбирают только один вариант из конечного набора заданных конфигураций алгоритмов/гиперпараметров. В-третьих, мы обсудим multi-fidelity методы, которые могут активно решать, какой класс приближений предоставит наибольшую информацию о поиске оптимальных гиперпараметров. Мы также ссылаемся на главу 2 (в которой обсуждается, как можно использовать multi-fidelity методы для разных наборов данных) и главу 3 (в которой описываются low-fidelity приближения для поиска нейронной архитектуры).

1.4.1 Прогнозирование ранней остановки на основе кривой обучения

Мы начинаем этот раздел о multi-fidelity методах в HPO с методов, которые оценивают и моделируют кривые обучения в процессе своей работы, а затем решают, добавлять ли дополнительные ресурсы или остановить процедуру обучения для заданной конфигурации гиперпараметра. Примерами кривых обучения являются точность одной и той же конфигурации, обученной на увеличивающихся подмножествах набора данных, или точность итерационного алгоритма, измеряемая для каждой итерации (или каждой i-й итерации, если вычисление является дорогостоящим).

Экстраполяция кривой обучения используется в контексте *прогностического завершения*, то есть используется некоторая модель для экстраполяции уже имеющейся части кривой обучения, и процесс обучения останавливается, если прогнозируется, что данная конфигурация не достигнет наилучшей производительности, полученной до сих пор в процессе оптимизации. Каждая кривая обучения моделируется как взвешенная комбинация из 11 параметрических функций из различных научных областей. Параметры этих функций и их веса выбираются с помощью цепей Маркова Монте Карло, чтобы свести к минимуму отклонения от наблюдаемой части кривой. Это дает вероятностное распределение, которое позволяет остановить обучение на основе вероятности того, что модель не превзойдет самую лучшую имеющуюся модель/конфигурацию. При оптимизации нейронных сетей критерий прогностического завершения в сочетании с байесовской оптимизацией позволил снизить частоту ошибок по сравнению с байесовской оптимизацией "черного ящика". В среднем этот метод ускорил оптимизацию в два раза и позволил найти (на тот момент) наиболее эффективную нейронную сеть для CIFAR-10 (без искусственного увеличения объема данных).

Хотя описанный выше метод ограничен отсутствием обмена информацией между различными конфигурациями гиперпараметров, этого можно достичь, используя упомянутые параметрические функции в качестве выходного уровня байесовской нейронной сети [80]. Таким образом, параметры и веса базисных функций и, следовательно, полная кривая обучения могут быть предсказаны для произвольных конфигураций гиперпараметров. Альтернативно можно использовать предыдущие кривые обучения в качестве новых базисных функций для экстраполяции. В то

время как результаты экспериментов являются неубедительными в отношении того, превосходит ли предлагаемый метод заранее заданные параметрические функции, отсутствие необходимости определять их вручную является явным преимуществом.

Байесовская оптимизация *замораживания-оттаивания* - это полная интеграция кривых обучения в процесс моделирования и выбора байесовской оптимизации. Вместо завершения процесса модели машинного обучения обучаются еще в течение нескольких итераций, а затем *замораживаются*. Байесовская оптимизация может затем решить *разморозить* одну из замороженных моделей, что означает продолжить ее обучение. В качестве альтернативы, метод также может принять решение о запуске новой конфигурации. Замораживание-оттаивание моделирует работу конвергентного алгоритма с регулярным гауссовским процессом и вводит специальную ковариационную функцию, соответствующую экспоненциально затухающим функциям, для моделирования кривых обучения с гауссовскими процессами для каждой кривой обучения.

1.4.2 “Бандитские” алгоритмы выбора

В этом разделе мы описываем методы, которые пытаются определить наилучший алгоритм из заданного конечного набора алгоритмов на основе оценок их производительности; ближе к концу мы также обсудим потенциальные комбинации со стратегиями адаптивной конфигурации. Мы фокусируемся на вариантах “бандитских” стратегий *последовательного деления пополам* и *Hyperband*, поскольку они показали высокую производительность, особенно для оптимизации алгоритмов глубокого обучения. Строго говоря, некоторые из методов, которые мы обсудим в этом подразделе, также моделируют кривые обучения, но они не предоставляют средств выбора новых конфигураций на основе этих моделей.

Однако сначала мы кратко опишем историческую эволюцию методов выбора алгоритмов с мульти-точностью. В 2000 году Петрак отметил, что простое тестирование различных алгоритмов на небольшом подмножестве данных является мощным и дешевым механизмом выбора алгоритма. Более поздние подходы использовали итеративные схемы исключения алгоритмов для удаления конфигураций гиперпараметров, если они плохо работают с подмножествами данных, если они работают значительно хуже, чем группа наиболее эффективных конфигураций, если они работают в *k*-

раз (*k* — указанный пользователем коэффициент) хуже, чем наилучшая конфигурация, или если даже оптимистичная оценка производительности для конфигурации хуже, чем у наилучшей конфигурации. Аналогично, можно отказаться от конфигураций гиперпараметров, если они плохо работают на одном или нескольких фолдах при кросс-валидации. Наконец, Джеймисон и Талвалкар предложили использовать алгоритм *последовательного деления пополам*, первоначально представленный Карни и др. для HPO.

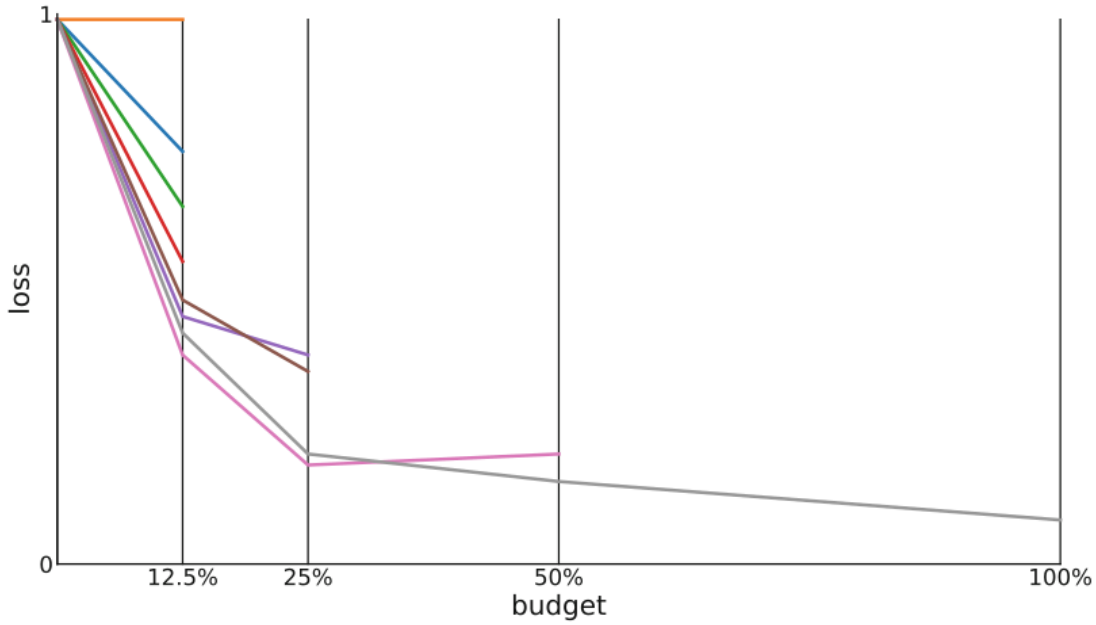

Рис. 1.3 Иллюстрация последовательного деления пополам для 8 алгоритмов/конфигураций. После оценки всех алгоритмов с использованием 1/8 от общего бюджета половина из них отбрасывается, а *budget* — бюджет , выделенный на оставшиеся алгоритмы, удваивается. *Loss* здесь обозначает функцию потерь.

Последовательное деление пополам — чрезвычайно простая, но мощная и поэтому популярная стратегия выбора алгоритма мульти-точности: для заданного начального бюджета запрашиваются все алгоритмы для этого бюджета; затем удаляется та половина, которая показала худшие результаты, далее удваивается бюджет 2 и операция последовательно повторяется до тех пор, пока не останется только один алгоритм. Этот процесс проиллюстрировано на рис. 1.3.

Джеймисон и Талвалкар [69] сравнили несколько распространенных бандитских методов и обнаружили, что последовательное деление пополам хорошо работает как с точки зрения количества требуемых итераций, так и с точки зрения необходимого времени вычислений, что алгоритм теоретически превосходит единую стратегию распределения бюджета, если алгоритмы сходятся благоприятно, и что предпочтительнее многих известных бандитских стратегий из литературы, таких как UCB и EXP3.

Хотя последовательное деление пополам является эффективным подходом, он страдает от компромисса между бюджетом и количеством конфигураций. Учитывая общий бюджет, пользователь должен заранее решить, следует ли попробовать множество конфигураций и назначить только небольшой бюджет для каждой, или попробовать только несколько конфигураций и назначить им больший бюджет.

Назначение слишком малого бюджета может привести к преждевременному прекращению работы хороших конфигураций, а назначение слишком большого бюджета может привести к слишком долгому запуску плохих конфигураций и, следовательно, к пустой трате ресурсов.

HyperBand [90] представляет собой стратегию хеджирования, предназначенную для решения этой проблемы при выборе из случайно выбранных конфигураций. Он делит общий бюджет на несколько комбинаций количества конфигураций и бюджета для каждой, а затем вызывает последовательное деление пополам в качестве подпрограммы для каждого набора случайных конфигураций. Из-за стратегии хеджирования, которая включает запуск некоторых конфигураций только с максимальным бюджетом, в худшем случае HyperBand занимает не более чем в

постоянную константу раз больше времени, чем ванильный случайный поиск с максимальным бюджетом. На практике было показано, что из-за использования дешевых оценок с низкой точностью HyperBand лучше обычного случайного поиска и байесовской оптимизации черного ящика для подмножеств данных, подмножеств признаков и итерационных алгоритмов, таких как стохастический градиентный спуск для глубоких нейронных сетей.

Несмотря на успех HyperBand для глубоких нейронных сетей, это очень ограничивает неадаптированность предложения конфигурации стратегии к оценке функций. Чтобы преодолеть это ограничение, подход BOHB [33] сочетает байесовскую оптимизацию и HyperBand для достижения лучшего из обоих миров: высокая производительность в любое время (быстрые улучшения в начале за счет использования низкой точности в HyperBand) и высокую конечную производительность (хорошая производительность в долгосрочной перспективе за счет замены случайного поиска HyperBand байесовской оптимизацией). BOHB также эффективно использует параллельные ресурсы и работает с проблемными областями в диапазоне от нескольких до многих десятков гиперпараметров. Компонент байесовской оптимизации BOHB напоминает TPE [12], но отличается тем, что использует многомерные оценки плотности ядра. Он подходит только для модели с наивысшей точностью, для которой не менее $|\Lambda| + 1$ оценка была выполнена (количество гиперпараметров плюс один). Таким образом, первая модель BOHB настроена на самую низкую точность, и со временем модели, обученные на более высокой точности, берут верх, но все еще используют более низкую точность при последовательном делении пополам. Эмпирически было показано, что BOHB превосходит несколько современных методов HPO для настройки машин опорных векторов, нейронных сетей и алгоритмов обучения с подкреплением, включая большинство методов, представленных в этом разделе [33]. Также были предложены дополнительные подходы к объединению HyperBand и байесовской оптимизации [15, 151].

Множественные оценки точности также можно комбинировать с HPO другими способами. Вместо переключения между более низкой точностью и максимальной точностью можно выполнить HPO на подмножестве исходных данных и извлечь наиболее эффективные конфигурации, чтобы

использовать их в качестве основы для HPO на полном наборе данных [152]. Чтобы ускорить решение проблемы CASH, также можно итеративно удалять целые алгоритмы (и их гиперпараметры) из пространства конфигурации из-за низкой производительности на небольших подмножествах наборов данных [159].

1.4.3 Адаптивный выбор точности

Все методы в предыдущем подразделе следуют заранее определенному расписанию точности. В качестве альтернативы можно активно выбирать, какие точности оценивать с учетом предыдущих наблюдений, чтобы предотвратить неправильную спецификацию расписания.

Многозадачная байесовская оптимизация [147] использует многозадачный гауссовский процесс для моделирования производительности связанных задач и автоматического изучения корреляции задач в процессе оптимизации. Этот метод может динамически переключаться между более дешевыми задачами с низкой точностью и дорогостоящей целевой задачей с высокой точностью на основе теоретико-информационной функции сбора данных с учетом затрат. На практике предлагаемый метод начинает исследовать пространство конфигурации на более дешевой задаче и переключается на более дорогое пространство конфигурации только на более поздних этапах оптимизации, примерно вдвое сокращая время, необходимое для HPO. Многозадачную байесовскую оптимизацию также можно использовать для переноса информации из предыдущих задач оптимизации, и мы можем обратимся к гл. 2 для получения дополнительной информации.

Многозадачная байесовская оптимизация (и методы, представленные в предыдущем подразделе) требуют предварительной спецификации набора точности. Это может быть неоптимальным, так как они могут быть указаны неправильно [74, 78] и потому что количество точностей, которые можно обработать, невелико (обычно пять или меньше). Следовательно, и для того, чтобы использовать обычно плавную зависимость от точности (такой как, например, размер данных используемого подмножества), часто дает лучшие результаты, если трактовать зависимость как непрерывную (и,

например, выбирать непрерывный процент от полного набора данных для оценки конфигурации), компенсируя прирост информации и время, необходимое для оценки [78]. Чтобы использовать знания предметной области о том, что производительность обычно улучшается с увеличением количества данных, с убывающей отдачи, для подмножеств данных может быть построено специальное ядро [78].

Это обобщение многозадачной байесовской оптимизации повышает производительность и может обеспечить ускорение в 10–100 раз по сравнению с байесовской оптимизацией черного ящика. Вместо использования теоретико-информационной функции сбора данных байесовская оптимизация с функцией сбора данных с верхней доверительной границей (UCB) также может быть расширена до множественной точности [73, 74]. В то время как первый такой подход, MFGP-UCB [73], требовал предварительных определений точности, более поздний алгоритм BOCA [74] отменил это требование. BOCA также применялся для оптимизации с более чем одной непрерывной точностью, и мы ожидаем, что HPO для более чем одной непрерывной точности будет представлять дальнейший интерес в будущем.

Вообще говоря, методы, которые могут адаптивно выбирать свою точность, очень привлекательны и более эффективны, чем концептуально более простые методы, основанные на бандитах, которые обсуждались в разд. 1.4.2, но на практике мы предупреждаем, что для успешного выбора точности требуются надежные модели. Когда модели недостаточно надежны (поскольку у них еще недостаточно данных для обучения или из-за несоответствия моделей), эти методы могут тратить слишком много времени на оценку более высокой точности, а более надежные расписания с фиксированным бюджетом, обсуждаемые в разд. 1.4.2 обеспечивает лучшую производительность при фиксированном ограничении времени.

1.5 Приложения к AutoML

В этом разделе мы предоставляем исторический обзор наиболее важных систем гиперпараметрической оптимизации и приложений для автоматизированного машинного обучения.

Поиск по сетке использовался для оптимизации гиперпараметров с 1990-х годов [71, 107] и уже поддерживался ранними инструментами машинного обучения в 2002 году [35]. Первыми методами адаптивной оптимизации, примененными к HPO, были жадный поиск в глубину [82] и поиск по шаблону [109], оба улучшились по сравнению с конфигурациями гиперпараметров по умолчанию, а поиск по шаблону также улучшился по сравнению с поиском по сетке. Генетические алгоритмы были впервые применены для настройки двух гиперпараметров C и γ из RBF SVM в 2004 году [119] и привели к улучшению производительности классификации за меньшее время, чем поиск по сетке. В том же году эволюционный алгоритм был использован для обучения композиции из трех различных ядер для SVM, гиперпараметров ядра и совместного выбора подмножества признаков; обученная комбинация ядер смогла превзойти каждое отдельное оптимизированное ядро. Схожий по духу генетический алгоритм также был использован в 2004 году для выбора признаков и гиперпараметров SVM или нейронной сети [129].

CMA-ES впервые был использован для оптимизации гиперпараметров в 2005 году [38], для оптимизации гиперпараметров SVM C и γ, шкалы длины ядра l_i для каждого измерения входных данных и полной матрицы поворота и масштабирования. Гораздо позже было продемонстрировано, что CMA-ES является отличным выбором для параллельного HPO, превосходя самые современные инструменты байесовской оптимизации при параллельной оптимизации 19 гиперпараметров глубокой нейронной сети на 30 графических процессорах [91].

В 2009 году Эскаланте и др. [30] расширили задачу HPO до задачи полного выбора модели, которая включает выбор алгоритма предварительной обработки, алгоритма выбора признаков, классификатора и всех их гиперпараметров. Получив возможность построить конвейер машинного обучения из нескольких готовых алгоритмов машинного обучения с помощью HPO, авторы эмпирически обнаружили, что могут применять свой метод к любому набору данных, поскольку для этого не требуется знание домена, и продемонстрировали применимость своего

подхода к различным областям [32, 49]. Предложенный ими метод - выбор модели роя частиц (PSMS) - использует модифицированный оптимизатор роя частиц для обработки пространства условных конфигураций. Чтобы избежать перебора, PSMS был расширен с помощью специальной стратегии ансамблирования, которая объединяет лучшие решения из нескольких поколений [31]. Поскольку оптимизация роя частиц изначально была разработана для работы с непрерывными конфигурационными пространствами, PSMS позже была также расширена, чтобы использовать генетический алгоритм для оптимизации структуры трубопровода и использовать оптимизацию роя частиц только для оптимизации гиперпараметров каждого трубопровода [145].

Насколько нам известно, первое применение байесовской оптимизации для HPO относится к 2005 году, когда Фролих и Зелл [39] использовали онлайн гауссовский процесс вместе с EI для оптимизации гиперпараметров SVM, добившись ускорения в 10 раз (классификация, 2 гиперпараметра) и 100 раз (регрессия, 3 гиперпараметра) по сравнению с поиском по сетке. Tuned Data Mining [84] предложил настраивать гиперпараметры полного конвейера машинного обучения с помощью байесовской оптимизации; в частности, использовался один фиксированный конвейер и настраивались гиперпараметры классификатора, а также порог классификации для каждого класса и веса классов.

В 2011 году Бергстра и др. [12] первыми применили байесовскую оптимизацию для настройки гиперпараметров глубокой нейронной сети, превзойдя как ручной, так и случайный поиск. Кроме того, они продемонстрировали, что TPE обеспечивает лучшую производительность, чем подход, основанный на гауссовских процессах. TPE, а также байесовская оптимизация со случайными лесами также были успешно применены для совместного поиска нейронной архитектуры и оптимизации гиперпараметров [14, 106].

Еще один важный шаг в применении байесовской оптимизации к HPO был сделан Snoek и др. в 2012 году в работе Practical Bayesian Optimization of Machine Learning Algorithms [140], в которой описаны несколько трюков для HPO на основе гауссовских процессов, реализованных в системе Spearmint, и получен новый современный результат для оптимизации гиперпараметров глубоких нейронных сетей.

Независимо от парадигмы полного выбора модели, Auto-WEKA [149] (см. также главу 4) введена задача комбинированного выбора алгоритма и оптимизации гиперпараметров (CASH), в которой выбор алгоритма классификации моделируется как категориальная переменная, гиперпараметры алгоритма моделируются как условные гиперпараметры, а байесовская система оптимизации SMAC на основе случайного леса [59] используется для совместной оптимизации в результирующей 786-мерное конфигурационное пространство.

В последние годы методы с множественной точностью стали очень популярны, особенно в глубоком обучении. Сначала, используя аппроксимации низкой верности на основе подмножеств данных, подмножеств признаков и коротких прогонов итерационных алгоритмов, Hyperband [90] продемонстрировал превосходство над методами байесовской оптимизации "черного ящика", которые не учитывали эти низкие верности. Наконец, совсем недавно, в работе BOHB: Robust and Efficient Hyperparameter Optimization at Scale, опубликованной в 2018 году, Фолкнер и другие [33] представили надежную, гибкую и распараллеливаемую комбинацию байесовской оптимизации и Hyperband, которая существенно превосходит как Hyperband, так и черную байесовскую оптимизацию для широкого круга задач, включая настройку машин опорных векторов, различных типов нейронных сетей и алгоритмов обучения с подкреплением.

На момент написания статьи мы даем следующие рекомендации относительно того, какие инструменты мы бы использовали в практических приложениях HPO:

- Если применимы множественные критерии достоверности (т.е. если возможно определить существенно более дешевые версии интересующей целевой функции, так что производительность для них примерно соответствует производительности для полной целевой функции, представляющей интерес), мы рекомендуем BOHB [33] как надежный, эффективный, универсальный и распараллеливаемый метод оптимизации гиперпараметров по умолчанию.
- В случае получения множественной верности не применяется:
    - Если все гиперпараметры имеют вещественное значение и можно позволить себе только несколько десятков оценок функций, мы рекомендуем использовать инструмент

байесовской оптимизации на основе гауссовского процесса, такой как Spearmint [140].

- Для больших и условных конфигурационных пространств мы предлагаем либо SMAC на основе случайного леса [59], либо TPE [14] из-за их доказанной высокой производительности в таких задачах [29].
- Для чисто вещественнозначных пространств и относительно дешевых целевых функций, для которых можно позволить себе более сотни оценок, мы рекомендуем CMA-ES [51].

1.6 Открытые проблемы и направления будущих исследований

Мы завершаем эту главу обсуждением открытых проблем, текущих исследований вопросы и потенциальные дальнейшие события, которые, как мы ожидаем, окажут влияние на ВПО[1] в будущем. Примечательно, что, несмотря на их актуальность, мы опускаем обсуждения важности гиперпараметров и определения конфигурационного пространства, поскольку они подпадают под сферу метаобучения и могут быть найдены в главе 2.

1.6.1 Контрольные показатели и сопоставимость

Учитывая широту существующих методов HPO, естественный вопрос заключается в том, каковы сильные и слабые стороны каждого из них. Чтобы обеспечить справедливое сравнение между различными подходами к HPO, сообществу необходимо разработать и согласовать общий набор критериев, который со временем расширяется по мере появления новых вариантов HPO, таких как оптимизация с высокой точностью. В качестве конкретного примера для
как это могло бы выглядеть, мы хотели бы упомянуть платформу COCO (сокращение от comparing continuous optimizers), которая предоставляет инструменты тестирования и анализа для непрерывной оптимизации и используется в качестве рабочей платформы для ежегодного тестирования оптимизации черного ящика (BBOB) [11]. Усилия в аналогичном направлении в HPO уже привели к созданию библиотеки оптимизации гиперпараметров (HPOlib [29]) и коллекции эталонных тестов специально

для байесовских методов оптимизации [25]. Однако ни одна из них не получила такой популярности, как платформа COCO.

Кроме того, сообществу нужны четко определенные показатели, но в настоящее время в разных работах используются разные показатели. Одним из важных аспектов, по которому оценки различаются, является то, сообщают ли они о производительности в наборе проверки, используемом для оптимизации, или в отдельном наборе тестов. Первый помогает изучить силу оптимизатора изолированно, без шума, который добавляется при оценке при переходе от проверки к набору тестов; с другой стороны, некоторые оптимизаторы могут привести к большему переоснащению, чем другие, что может быть диагностировано только с помощью набора тестов. Другим важным аспектом, по которому оценки различаются, является то, сообщают ли они о результатах после определенного количества оценок функций или по истечении определенного промежутка времени. Последний учитывает разницу во времени между оценкой различных конфигураций гиперпараметров и включает накладные расходы на оптимизацию, и, следовательно, отражает то, что требуется на практике; однако первый более удобен и способствует воспроизводимости, давая одинаковые результаты независимо от используемого оборудования. Поэтому, чтобы повысить воспроизводимость, особенно для исследований, требующих времени, следует опубликовать реализацию.

Мы отмечаем, что при использовании новых контрольных показателей важно проводить сравнение с надежными базовыми показателями, что является еще одной причиной, по которой методы HPO должны публиковаться с сопровождающей реализацией. К сожалению, не существует общей библиотеки программного обеспечения, как, например, доступно в исследованиях глубокого обучения, которая реализует все основные строительные блоки [2, 117]. В качестве простого, но эффективного базового уровня, который может быть тривиально включен в эмпирические исследования, Джеймисон и Рехт [68] предлагают сравнить с различными уровнями распараллеливания случайного поиска, чтобы продемонстрировать
ускорение по сравнению с обычным случайным поиском. При сравнении с другими методами оптимизации важно сравнивать с надежной реализацией, поскольку, например, было показано, что более простые версии байесовской оптимизации дают более низкую производительность [79, 140, 142].

1.6.2 Оптимизация на основе градиента

В некоторых случаях (например, машины опорных векторов методом наименьших квадратов и нейронные сети) возможно получить градиент критерия выбора модели по отношению к некоторым гиперпараметрам модели. В отличие от blackbox HPO, в этом случае каждое вычисление целевой функции приводит к целому гиперградиентному вектору вместо одного значения с плавающей точкой, что позволяет ускорить HPO.

Маклорин и др. [99] описали процедуру вычисления точных градиентов производительности валидации по отношению ко всем непрерывным гиперпараметрам нейронной сети путем обратного распространения через всю процедуру обучения стохастического градиентного спуска с импульсом (используя новый алгоритм, экономящий память). Способность эффективно обрабатывать множество гиперпараметров с помощью методов, основанных на градиенте, позволяет создать новую парадигму гиперпараметризации модели для получения гибкости по сравнению с классами моделей, регуляризацией и методами обучения. Маклорин и др. продемонстрировал применимость HPO на основе градиента ко многим задачам HPO высокой размерности, таким как оптимизация скорости обучения нейронной сети для каждой итерации и слоя в отдельности, оптимизация гиперпараметра шкалы инициализации веса для каждого слоя в нейронной сети, оптимизация штрафа l2 для каждого отдельного параметра в логистической регрессии, и изучение совершенно новых обучающих наборов данных. В качестве небольшого недостатка, обратное распространение через всю процедуру обучения происходит за счет удвоения временной сложности процедуры обучения. Описанный метод также может быть обобщен для работы с другими алгоритмами обновления параметров [36]. Чтобы преодолеть необходимость обратного распространения через всю процедуру обучения, последующая работа позволяет выполнять обновления гиперпараметров в отношении отдельного набора проверки, чередующегося с процессом обучения [5, 10, 36, 37, 93].

Недавние примеры основанной на градиенте оптимизации гиперпараметров простой модели [118] и структур нейронных сетей (см. гл. 3) демонстрируют многообещающие результаты, превосходящие самые современные байесовские оптимизационные модели. Несмотря на высокую специфику модели, тот факт, что оптимизация гиперпараметров на основе

градиента позволяет настраивать несколько сотен гиперпараметров, может позволить существенно улучшить HPO.

1.6.3 Масштабируемость

Несмотря на недавние успехи в области многомерной оптимизации, все еще существуют проблемы машинного обучения, которые не были напрямую решены с помощью HPO из-за их масштаба, и которые могут потребовать новых подходов. Здесь масштаб может означать как размер конфигурационного пространства, так и затраты на оценку отдельных моделей. Например, до сих пор не было никаких работ по HPO для глубоких нейронных сетей на наборе данных ImageNet challenge [127], в основном из-за высокой стоимости обучения даже простой нейронной сети на этом наборе данных. Будет интересно посмотреть, позволят ли методы, выходящие за рамки представления "черного ящика" из раздела 1.3, такие как методы мультиверсии, описанные в разделе 1.4, градиентные методы или методы метаобучения (описанные в гл. 2), решить такие проблемы. В гл. 3 описаны первые успехи в обучении строительных блоков нейронных сетей на небольших наборах данных и их применение к ImageNet, однако гиперпараметры процедуры обучения по-прежнему задаются вручную. Учитывая необходимость параллельных вычислений, мы с нетерпением ждем появления новых методов, в полной мере использующих возможности крупномасштабных вычислительных кластеров. Хотя существует много работ по параллельной байесовской оптимизации [12, 24, 33, 44, 54, 60, 135, 140], за исключением нейронных сетей, описанных в разделе 1.3.2.2 [141], до сих пор ни один метод не продемонстрировал масштабируемость до сотен рабочих. Несмотря на их популярность, за единственным исключением HPO, примененного к глубоким нейронным сетям [91],3 популяционные подходы еще не были продемонстрированы для оптимизации гиперпараметров на наборах данных, превышающих несколько тысяч точек данных. В целом, мы ожидаем, что потребуются более сложные и специализированные методы, оставляющие позади[2] для дальнейшего масштабирования гиперпараметров для решения интересных задач. проблем.

1.6.4 Избыточная подгонка и обобщение

Открытой проблемой в HPO является чрезмерная подгонка. Как отмечалось в постановке задачи (см. раздел 1.2), обычно мы имеем только ограниченное количество точек данных, доступных для расчета валидационной потери, которую необходимо оптимизировать, и, таким образом, не обязательно оптимизировать для обобщения на невидимые тестовые точки данных. Аналогично переподгонке алгоритма машинного обучения к обучающим данным, эта проблема заключается в переподгонке гиперпараметров к ограниченному валидному множеству; это также было продемонстрировано экспериментально [20, 81].

Простой стратегией для уменьшения переподгонки является использование различной перестановки обучающего и валидационного множеств для каждой оценки функции; было показано, что это улучшает эффективность обобщения при настройке SVM, как при стратегии удержания, так и при стратегии перекрестной валидации [95]. Выбор окончательной конфигурации может быть дополнительно усилен, если выбирать ее не по наименьшему наблюдаемому значению, а по наименьшему среднему прогнозируемому значению модели гауссовского процесса, используемой в байесовской оптимизации [95].

Другой возможностью является использование отдельного удерживающего набора для оценки конфигураций, найденных HPO, чтобы избежать смещения в сторону стандартного валидационного набора [108, 159]. Различные аппроксимации показателей обобщения могут привести к различным показателям тестирования [108], и есть сообщения о том, что несколько стратегий повторной выборки могут привести к измеримым различиям в производительности HPO машин опорных векторов [150].

Другой подход к борьбе с переборкой[3] может заключаться в поиске стабильных оптимумов вместо резких оптимумов объективной[4] функции [112]. Идея заключается в том, что для стабильных оптимумов значение функции вблизи оптимума не меняется при незначительных возмущениях гиперпараметров, в то время как для острых оптимумов оно меняется. Стабильные оптимумы приводят к лучшему обобщению при применении найденных гиперпараметров к новому, невидимому набору точек данных (т.е. тестовому набору). Было показано, что функция приобретения, построенная на основе этого, лишь немного перестраивается для HPO машины опорных векторов, в то время как обычная байесовская оптимизация демонстрирует сильную перестройку[5] [112].

Другими подходами к борьбе с избыточной подгонкой являются методы ансамблей и байесовские методы, представленные в разделе 1.2.1. Учитывая все эти различные методы, не существует общепринятой методики, как наилучшим образом избежать чрезмерной подгонки, и пользователю остается выяснить, какая стратегия лучше всего подходит для его конкретной проблемы HPO. Мы отмечаем, что наилучшая стратегия может быть различной для разных задач HPO проблемы.

1.6.5 Строительство пайплайна произвольного размера

Все методы HPO, которые мы обсуждали до сих пор, предполагают конечный набор компонентов для конвейеров машинного обучения или конечное максимальное число слоев в нейронных сетях. Для конвейеров машинного обучения (см. Системы AutoML, описанные в части II этой книги) может быть полезно использовать более одного алгоритма предварительной обработки объектов и динамически добавлять их, если это необходимо для решения задачи, увеличивая пространство поиска на гиперпараметр, чтобы выбрать подходящий алгоритм предварительной обработки и его собственные гиперпараметры. В то время как пространство поиска для стандартных инструментов оптимизации черного ящика могло бы легко включать несколько дополнительных таких препроцессоров (и их гиперпараметров) в качестве условных гиперпараметров, неограниченное количество из них было бы трудно поддерживать

Одним из подходов к более естественной обработке конвейеров произвольного размера является инструментарий оптимизации конвейеров с древовидной структурой (TPOT [115], см. также главу 8), который использует генетическое программирование и описывает возможные конвейеры с помощью грамматики. TPOT использует многоцелевую оптимизацию, чтобы сбалансировать сложность конвейера с производительностью, чтобы избежать создания излишне сложных конвейеров.

Другой парадигмой создания конвейера является использование иерархического планирования; недавний ML-Plan [101, 108] использует иерархические сети задач и демонстрирует конкурентоспособную производительность по сравнению с Auto-WEKA [149] и Auto sklearn [34].

Пока что эти подходы не всегда превосходят системы AutoML с фиксированной длиной пайплайн, но более крупные трубопроводы могут обеспечить большее улучшение. Аналогично, поиск по нейронной архитектуре приводит к сложным конфигурационным пространствам, и мы ссылаемся на главу 3 для описания методов их решения.

Глава 2. Метаобучение

Хоакин Ваншорен (Joaquin Vanschoren)

Аннотация

Метаобучение, или обучение обучения, - это наука о систематическом наблюдении за тем, как различные подходы к машинному обучению выполняют широкий спектр задач обучения, а затем обучаются на основе этого опыта (или метаданных), чтобы осваивать новые задачи намного быстрее, чем это возможно в противном случае. Это не только значительно ускоряет и улучшает разработку пайплайнов машинного обучения или нейронных архитектур, но и позволяет нам заменить алгоритмы, разработанные вручную, новейшими подходами, изученными на основе данных. В этой главе мы даем обзор последних разработок в этой увлекательной и постоянно развивающейся области.

2.1 Введение

Когда мы осваиваем новые навыки, мы редко – если вообще когда–либо - начинаем с нуля. Мы начинаем с навыков, приобретенных ранее в похожих задачах, переиспользуем подходы, которые хорошо работали раньше, и фокусируемся на том, что, вероятно, стоит попробовать, основываясь на опыте [82]. С каждым приобретенным навыком освоение новых навыков становится все проще, требуя меньше примеров и меньше проб и ошибок. Иначе говоря, мы учимся тому, как учиться в разных задачах. Аналогично, при построении моделей машинного обучения для конкретной задачи мы часто опираемся на опыт работы с похожими задачами или используем наше (часто косвенное) понимание поведения методов машинного обучения, чтобы помочь сделать правильный выбор.

Задача метаобучения состоит в том, чтобы извлекать уроки из предыдущего опыта систематическим образом, основанным на данных. Во-первых, нам нужно собрать метаданные, которые описывают предыдущие задачи обучения и ранее обученные модели. Они включают в себя те конфигурации алгоритмов, которые использовались для обучения моделей, включая настройки гиперпараметров, составы пайплана и/или архитектуры сетей, результаты оценки модели, такие как точность и время обучения, параметры обученной модели, такие как обученные веса нейронной сети, а

также измеримые свойства модели, также известные как мета-характеристики. Во-вторых, нам нужно обучиться на этих предыдущих данных, извлечь и перенести знания, которые помогут в поисках оптимальных моделей для новых задач. Эта глава содержит краткое описание различных подходов для эффективного мета-обучения.

Термин мета-обучение покрывает любой тип обучения, основанный на предыдущем опыте с другими задачами. Чем сильнее схожи эти предыдущие задачи, тем большее количество мета-данных мы можем использовать, и определение схожести задач будет ключевой всеобъемлющей задачей. Возможно, нет необходимости говорить о том, что бесплатный сыр есть только в мышеловке. Если новая задача представляет собой совершенно несвязанные явления или случайный шум, использование предыдущего опыта не будет эффективно. К счастью, в реальных задачах существует большое количество возможностей извлечь уроки из предыдущего опыта.

В оставшейся части этой главы мы классифицируем методы мета-обучения на основе типа мета-данных, которые они используют, от самых общих до наиболее специфичных для конкретной задачи. Во-первых, в разделе 2.2 мы рассматриваем, как обучаться исключительно на оценках моделей. Эти методы могут быть использованы для рекомендации общеполезных конфигураций и пространств поиска конфигураций, а также для передачи знаний из эмпирически аналогичных задач. В разделе. 2.3, мы обсуждаем, как мы можем охарактеризовать задачи, чтобы более явно выразить сходство задач и построить метамодели, которые изучают взаимосвязи между характеристиками данных и эффективностью обучения. Наконец, в разделе 2.4 описывается, как мы можем передавать параметры обученной модели между задачами, которые по своей сути схожи, например, использовать одни и те же входные характеристики, что позволяет переносить обучение и многоразовое обучение помимо прочих.

Обратите внимание, что, хотя многозадачное обучение (одновременное обучение нескольких связанных задач) и коллективное обучение (построение нескольких моделей для одной и той же задачи) часто могут быть эффективно объединены с системами метаобучения, сами по себе они не предполагают изучения предыдущего опыта выполнения других задач.

Эта глава основана на совсем недавней обзорной статье [176].

2.2 Обучение на основе оценок моделей

Рассмотрим, что у нас есть доступ к предыдущим задачам $t_j \in T$, набору всех известных задач, а также набору алгоритмов обучения, полностью определенных их конфигурациями $\theta_i \in \boldsymbol{\Theta}$; здесь $\boldsymbol{\Theta}$ представляет собой дискретное, непрерывное или смешанное пространство конфигурации, которое может охватывать настройки гиперпараметров, компоненты пайплайна и / или компоненты архитектуры сетей. $\mathbf{P}$ - это набор всех предыдущих скалярных оценок $P_{i,j} = P(\theta_i, t_j)$ конфигурации θ_i для задачи t_j в соответствии с предопределенной мерой оценки, например точностью, и методом оценки модели, например кросс-валидацией. $\mathbf{P}$new - это набор известных оценок P_i, новых для новой задачи tnew. Теперь мы хотим обучить метаученика L, который прогнозирует рекомендуемые конфигурации $\boldsymbol{\Theta}^*$new для новой задачи tnew. Мета-ученик обучается на основе метаданных $\mathbf{P} \cup \mathbf{P}$new. $\mathbf{P}$ обычно собирается заранее или извлекается из хранилищ метаданных. $\mathbf{P}$new изучается с помощью самой техники мета-обучения итеративным способом, иногда с предварительным запуском с начальной $\mathbf{P}'$new, сгенерированной другим методом.

2.2.1 Рекомендации, не зависящие от задачи

Во-первых, представьте, что у вас нет доступа к каким-либо оценкам на tnew, следовательно, $P_{new} = \emptyset$. Тогда мы все еще можем изучить функцию $f : \boldsymbol{\Theta} \times T \rightarrow \{\theta_k*\}$, $k = 1..K$, что дает набор рекомендуемых конфигураций, независящих от tnew. Эти θ_k* затем могут быть оценены заново для выбора наилучшего из них или для предварительного запуска дальнейших подходов к оптимизации, таких как те, которые обсуждаются в разделе 2.2.3.

Такие подходы часто приводят к ранжированию, т.е. упорядоченному множеству θ_k*. Обычно это делается путем дискретизации $\boldsymbol{\Theta}$ в набор возможных конфигураций θ_i, также называемый портфолио, оцениваемый по большому количеству задач t_j. Затем мы можем отранжировать каждую задачу, например, используя показатели успеха, AUC или значительные

выигрыши. Однако чаще всего желательно, чтобы одинаково хорошие, но более быстрые алгоритмы оценивались выше, и было предложено множество методов для компромисса между точностью и временем обучения. Затем мы можем объединить эти рейтинги по отдельным задачам в глобальный рейтинг, например, путем вычисления среднего ранга по всем задачам. Когда недостаточно данных для построения общего рейтинга, можно рекомендовать подмножества конфигураций на основе наиболее известных конфигураций для каждой предыдущей задачи или возвращать квазилинейные рейтинги.

Чтобы найти наилучшее решение $\theta *$ для задачи t_{new}, никогда ранее не встречавшейся, можно использовать простой метод, работающий в любое время, — выбрать K лучших конфигураций [21], пройтись по списку и оценить каждую конфигурацию по t_{new} по очереди. Эта оценка может быть остановлена после достижения заданного значения K, лимита времени или при обнаружении достаточно точной модели. В условиях ограниченного времени было показано, что многокритериальное ранжирование (включая время обучения) сходятся к почти оптимальным моделям намного быстрее [1, 134] и обеспечивают прочную основу для сравнения алгоритмов [1, 85]. Подход, сильно отличающийся от описанного выше, заключается в том, чтобы сначала подобрать дифференцируемую функцию $f_j(\theta_i) = P_{i,j}$ для всех предыдущих оценок конкретной задачи t_j , а затем использовать градиентный спуск для поиска оптимизированной конфигурации $\theta_j *$ на предыдущую задачу [186]. Предполагая, что некоторые из задач t_j будут аналогичны t_{new}, эти $\theta_j *$ будут полезны для теплого старта подходов байесовской оптимизации с быстрым запуском.

2.2.2 Дизайн пространства конфигурации

Предыдущие оценки также могут быть использованы для изучения лучшего пространства конфигурации $*$. Хотя это снова не зависит от t_{new}, это может радикально ускорить поиск оптимальных моделей, поскольку исследуются только наиболее важные области конфигурационного пространства. Это имеет решающее значение, когда вычислительные ресурсы ограничены, и оказалось важным фактором при практическом сравнении систем AutoML [33]. Во-первых, в функциональном подходе ANOVA [67] гиперпараметры считаются важными, если они объясняют большую часть различий в производительности алгоритма для данной задачи. В [136] это было исследовано с использованием 250 000 экспериментов OpenML с 3 алгоритмами в 100 наборах данных. 38 J. Vanschoren Альтернативный подход заключается в том, чтобы сначала узнать оптимальную настройку гиперпараметра по умолчанию, а затем определить важность гиперпараметра как повышение производительности, которое может быть достигнуто путем настройки гиперпараметра вместо того, чтобы оставить его на этом значении по умолчанию. Верно, даже

несмотря на то, что гиперпараметр может вызывать много различий, он также может иметь одну конкретную настройку, которая всегда приводит к хорошей производительности. В [120] это было сделано с использованием около 500 000 экспериментов OpenML на 6 алгоритмах и 38 наборах данных. Значения по умолчанию изучаются совместно для всех гиперпараметров алгоритма путем первоначального обучения суррогатных моделей для этого алгоритма для большого количества задач. Затем выполняется выборка многих конфигураций, и конфигурация, минимизирующая средний риск для всех задач, является рекомендуемой конфигурацией по умолчанию. Наконец, важность (или возможность настройки) каждого гиперпараметра оценивается путем наблюдения за тем, сколько улучшений можно получить, настроив его. В [183] значения по умолчанию обучаются независимо от других гиперпараметров. и определяются как конфигурации, наиболее часто встречающиеся в топ-K конфигураций для каждой задачи. В случае, когда оптимальное значение по умолчанию зависит от мета-функций (например, количества обучающих экземпляров или функций), изучаются простые функции, включающие эти мета-функции. Затем статистический тест определяет, можно ли безопасно оставить гиперпараметр с этим значением по умолчанию, исходя из потери производительности, наблюдаемой, когда гиперпараметр (или набор гиперпараметров) не настраивается, в то время как все остальные параметры настраиваются. Это было оценено с использованием 118 000 экспериментов OpenML с 2 алгоритмами (SVM и Random Forests) в 59 наборах данных.

2.2.3 Передача конфигурации

Если мы хотим предоставить рекомендации для конкретной задачи t_{new}, нам нужна дополнительная информация о том, насколько t_{new} похож на предыдущие задачи t_j. Один из способов сделать это — оценить ряд рекомендуемых (или потенциально случайных) конфигураций на t_{new}, получив новое свидетельство P_{new}. Если затем мы заметим, что оценки $P_{i,new}$ подобны $P_{i,j}$, то t_j и t_{new} можно считать внутренне подобными на основе эмпирических данных. Мы можем включить эти знания для обучения мета-обучаемого, который предсказывает рекомендуемый набор конфигураций $*$ новых для t_{new}. Более того, каждое выбранное θ_{new} $*$ можно оценить и включить в P_{new}, повторяя цикл и собирая больше эмпирических данных, чтобы узнать, какие задачи похожи друг на друга.

2.2.3.1 Относительные ориентиры

Первая мера сходства задач учитывает относительные (попарные) различия в производительности, также называемые относительными ориентирами, $RL_{a,b,j} = P_{a,j} —P_{b,j}$ между двумя конфигурациями θ_a и θ_b на конкретной задаче t_j[53]. Активное тестирование [85] использует их следующим образом: оно стартует с лучшей в глобальном масштабе конфигурации (см. раздел 2.2.1), называет ее θлучшей и продолжает в стиле турнира. В каждом раунде он выбирает «конкурента» θ_c, который наиболее убедительно превосходит θ_{best} в аналогичных задачах. Мета-обучение считает задачи похожими, если относительные ориентиры всех оцениваемых конфигураций аналогичны, т. е. если конфигурации работают одинаково как на t_j, так и на t_{new}, тогда задачи считаются аналогичными. Затем он оценивает конкурента θ_c, получая $P_{c,new}$, обновляет сходство задач и повторяет. Ограничение этого метода заключается в том, что он может учитывать только те конфигурации θ_i, которые были оценены во многих предыдущих задачах.

2.2.3.2 Суррогатные модели

Более гибким способом передачи информации является построение суррогатных моделей s*j* (θ*i*) = *Pi, j* для всех предыдущих задач *tj* , обученных с использованием всех доступных P. Затем можно определить сходство задач в терминах ошибки между s*i* (θ*i*) и *Pi, new*: если суррогатная модель для *tj* может генерировать точные прогнозы для *tnew*, то эти задачи по сути схожи. Обычно это делается в сочетании с байесовской оптимизацией, чтобы определить следующий θ*i*.

Wistuba и др. [187] обучают суррогатные модели на основе гауссовых процессов (GPs) для каждой предыдущей задачи плюс одну для новой и объединяют их во взвешенную нормализованную сумму с прогнозируемым средним μ, определяемым как взвешенная сумма отдельных μ (полученных из предыдущих задачи *t*). Веса μ ’s вычисляются с использованием средневзвешенного значения ядра Надараи-Уотсона, где каждая задача представлена в виде вектора относительных ориентиров, а квадратичное ядро Епанечникова [104] используется для измерения сходства между относительными ориентирами векторов *tj* и *tnew*. Чем больше *tj* похож на *tnew*, тем больше вес s*j*, увеличивая влияние суррогатной модели для *tj*.

Фейрер и др. [45] предлагают объединить прогнозирующие распределения отдельных гауссовых процессов, что снова делает объединенную модель гауссовым процессом. Веса вычисляются в соответствии с независимым байесовским ансамблем Лакоста и др. [81], который взвешивает предикторы в соответствии с оценкой их эффективности обобщения.

Метаданные также могут быть переданы в функции сбора данных, а не в суррогатной модели [187]. Суррогатная модель обучается только для *Pi, new*, но следующий θ*i* для оценки обеспечивается функцией сбора данных, которая представляет собой средневзвешенное значение ожидаемого улучшения [69] для *Pi*, *new* и прогнозируемых улучшений для всех предыдущих *Pi, j*. Веса предыдущих задач снова могут быть определены с помощью точности суррогатной модели или с помощью относительных ориентиров. Вес ожидаемого компонента улучшения постепенно увеличивается с каждой итерацией по мере сбора большего количества доказательств *Pi*, новых.

2.2.3.3 Многозадачное обучение с теплым запуском

Другой подход к соотнесению предыдущих задач *tj* заключается в изучении представления совместной задачи с использованием P предварительных оценок. В [114] для конкретной задачи байесовской линейной регрессии [20] суррогатные модели sj(θiz) обучаются в новой конфигурации θ^z, изучаемой прямой нейронной сетью *NN*(θi), которая обучает подходящее базовое расширение θ^z исходной конфигурации θ, в котором линейные суррогатные модели могут точно предсказывать Pi, новые. Суррогатные модели предварительно обучены на метаданных OpenML, чтобы обеспечить предварительный запуск для оптимизации *NN*(θi) в условиях многозадачного обучения. Более ранняя работа по многозадачному обучению [166] предполагала, что у нас уже есть набор ‘похожих’ исходных задач *tj*. Он передает информацию между этими *tj* и t*new* путем построения совместной модели GP для байесовской оптимизации, которая изучает и использует точную взаимосвязь между задачами. Однако изучение совместного GP, как правило, менее масштабируемо, чем создание одного GP для каждой задачи. Спрингенберг и др. [161] также предполагают, что задачи связаны и похожи, но изучают взаимосвязь между задачами в процессе оптимизации с использованием

байесовских нейронных сетей. Таким образом, их метод представляет собой своего рода гибрид двух предыдущих подходов. Головин и др. [58] предполагают порядок последовательности (например, время) между задачами. Он создает стек регрессоров GP, по одному на задачу, обучая каждый GP остаткам относительно регрессора под ним. Следовательно, каждая задача использует стоящие перед ней задачи для определения своих приоритетов.

2.2.3.4 Другие методики

Многорукие бандиты, [139] обеспечивают еще один подход к поиску исходных задач tj, наиболее связанных с t_{new}[125]. В этой аналогии каждый t_j - это одна рука, и (стохастическое) вознаграждение за выбор (выполнение) конкретной предшествующей задачи (arm) определяется в терминах ошибки в прогнозах байесовского оптимизатора на основе GP, который моделирует предыдущие оценки tj как зашумленные измерения и объединяет их с существующие оценки по tnew. Однако кубическое масштабирование графического процессора делает этот подход менее масштабируемым.

Другой способ определить сходство задач - взять существующие оценки Pi, j, использовать Выборку Томпсона [167] для получения оптимального распределения p^j_{max}, а затем измерения KL-дивергенции [80] между p^j_{max} и p^{new}_{max} [124]. Затем эти распределения объединяются в смешанное распределение, основанное на сходствах, и используются для построения функции сбора данных, которая предсказывает следующую наиболее перспективную конфигурацию для оценки. Пока что оценена только настройка 2 гиперпараметров SVM с использованием 5 задач.

Наконец, дополнительный способ использовать P - это рекомендовать, какие конфигурации не следует использовать. После обучения суррогатных моделей для каждой задачи мы можем посмотреть, какие *tj* наиболее похожи на t*new*, а затем использовать s*j* (θ*i*), чтобы обнаружить области Q, где производительность, по прогнозам, будет низкой. Исключение этих регионов может ускорить поиск более эффективных. Wistuba и др. [185], делают это, используя меру сходства задач, основанную на коэффициенте ранговой корреляции Кендалла тау [73] между рангами, полученными путем ранжирования конфигураций θ*i* с использованием P*i, j* и P*i, new*, соответственно.

2.2.4 Кривые обучения

Мы также можем извлекать метаданные о самом процессе обучения, например о том, насколько быстро улучшается производительность модели по мере добавления дополнительных обучающих данных. Если обучение разделено на этапы s_t, как правило, с добавлением фиксированного количества обучающих примеров на каждый этап, то производительность $P(\theta_i,t_j,s_t)$ конфигурации θ в задаче t_j после этапа s_t может быть измерена, что позволит построить кривую обучения по этапам времени s_t. Как обсуждалось в главе 1, кривые обучения также используются для ускорения оптимизации гиперпараметров для данной задачи. В метаобучении информация о кривой обучения передается между задачами.

При оценке конфигурации для новой задачи t_{new} мы можем прекратить обучение после определенного количества итераций $r < t$ и использовать частично наблюдаемую кривую обучения, чтобы предсказать, насколько хорошо конфигурация будет работать на всем наборе данных, на основе предыдущего опыта с другими задачами. и принять решение о продолжении обучения. Это может значительно ускорить поиск хорошей конфигурации.

Один из подходов состоит в том, чтобы предположить, что аналогичные задачи будут давать аналогичные кривые обучения. Во-первых, расстояние между задачами задачами на основе того, насколько похожи частичные кривые обучения: $dist(t_a,t_b)=f(P_{i,a,t},P_{i,b,t}),t=1,...r$. Затем найдите K наиболее похожих задач $t_{1....k}$ и используйте их полные кривые обучения, чтобы предсказать, насколько хорошо конфигурация будет работать с новым полным набором данных. Сходство задач можно измерить, сравнивая формы локальных кривых во всех опробованных конфигурациях, а прогнозы можно делать, сопоставляя «ближайшую» полную кривую с новой локальной кривой [83, 84]. Этот подход также успешен в сочетании с активным тестированием [86] и может быть дополнительно ускорен с помощью многоцелевых методов оценки, включающих время обучения [134].

Интересно, что хотя несколько методов нацелены на прогнозирование кривых обучения при поиске нейронной архитектуры (см. главу 3), ни один из этих методов пока не использует кривые обучения, ранее наблюдаемые в других задачах.

2.3 Обучение на основе свойств задачи

Еще одним богатым источником метаданных являются характеристики поставленной задачи (мета-функции). Каждая задача $t_j \in T$ может быть описана вектором $m(t_j)=(m_{j,1.....},m_{j,k})$, состоящим из K метапризнаков, где $m_{j,k} \in M$— множество всех известных метапризнаков. Это можно использовать для мер сходства, определяя меру сходства задач на основе (например) евклидова расстояния между $m(t_i)$ и $m(t_j)$, чтобы мы могли передавать информацию из наиболее похожей задачи в новую задачу t_{new}. Кроме того, вместе с предыдущей оценкой P мы можем обучить мета-обучаемого L прогнозировать производительность $P_{i,new}$ конфигурации θ_i в новой задаче t_{new}.

2.3.1 Мета-функции

В таблице 2.1 представлен краткий обзор наиболее часто используемых метафункций и краткое объяснение того, почему они указывают на производительность модели. Там, где это возможно, мы также показываем формулы, используемые для их расчета. Более полные обзоры можно найти в литературе [26,98,130,138,175].

Чтобы построить вектор метапризнаков $m(t_j)$, эти метапризнаки необходимо выбрать и дополнительно обработать. Исследования метаданных OpenML показывают, что оптимальный набор метафункций зависит от приложения [17]. Многие мета-признаки рассчитываются на основе отдельных признаков или комбинаций признаков и должны быть суммированы с помощью сводной статистики (max, min, μ, σ, quartiles, $q_{1...4}$) или гистограмм [72]. Их необходимо систематически извлекать и агрегировать [117]. Также важно нормализовать все мета-признаки [9], выполнить выбор признаков [172] или использовать методы уменьшения размерности (например, PCA) [17] при вычислении сходства задач. При изучении метамодели также могут использоваться реляционные метаобучения [173] или методы рассуждений на основе прецедентов [63, 71, 92].

В дополнение к этим общим мета-признакам было сформулировано множество более специфических мета-признаков. Для потоковых данных можно использовать потоковые ориентиры [135, 137], для данных временных рядов можно рассчитать коэффициент автокорреляции или

наклон регрессионной модели [7, 121, 147], а для неконтролируемых задач данные можно группировать различными способами и извлекать свойства этих кластеров [159]. Информация, относящаяся к предметной области, также может использоваться во многих приложениях [109, 156].

2.3.2 Мета-функции обучения

Вместо того, чтобы вручную определять метафункции, мы также можем изучить совместное представление для групп задач. Один из подходов заключается в построении метамодели, которая, учитывая другие мета-функции задачи M, обучается на метаданных производительности P или $f:M\rightarrow M'$ для создания представления мета-функцииM′, похожего на ориентир. Sun и Pfahringer [165] делают это, оценивая набор предопределенных конфигураций θ_i для всех предыдущих задач t_j и генерируя бинарную мета-функцию m для каждой попарной комбинации конфигураций θ_a и θ_b, обозначая, является ли производительность θ лучше, чем θ, отсюда $m'(t_j)=(m_{j,a,b},m_{j,a,c},m_{j,b,c}..)$. Чтобы вычислить $m_{new,a,b}$, из каждой попарной комбинации (a, b) изучаются метаправила, каждое из которых предсказывает, будет ли θ_a лучше, чем θ_b в задаче t_j, учитывая его дополнительную метахарактеристику $m(t_j)$.

Мы также можем изучить совместное представление $f:P\times\Theta\rightarrow M'$, полностью основанное на доступных метаданных P. Ранее в разделе 2.2.3 мы обсуждали, как использовать нейронные сети с прямой связью [114] для этой цели. Если задачи используют одно и то же входное пространство, например, это изображения с одинаковым разрешением, глубокое метрическое обучение также можно использовать для изучения представления мета-признаков, например, с использованием сиамских (Siamese) сетей [75].

Таблица 2.1

Обзор часто используемых мета-функций. Группы сверху вниз: простые, статистические, теоретико-информационные, сложные, основанные на моделях и ориентиры. Непрерывные объекты X и целевой объект Y имеют среднее значение μ_X, stdev σ_X, дисперсию σ^2_X.Категориальные объекты X и класс C имеют категориальные значения pi, условные вероятности $p_{i|j}$, совместные вероятности $p_{I,j}$, предельные вероятности p_i + =å$_j$ p_{ij}, энтропия H(x)=− =å$_j$ p_i +log$_2$(p_{i+}).

Название	Формула	Обоснование	Разновидность
Количество экземпляров	n	Скорость, масштабируемость [99]	p/n, log(n), log(n/p)
Количество свойств	p	Проклятие размерности [99]	log(p), % categorical
Классы	c	Сложность, дисбаланс [99]	соотношение min/maj класс
Отсутствующие значения	m	Эффекты вменения [70]	% отсутствует
Число выбросов	o	Шумность данных [141]	o/n

Коэффициент перекоса	$E(X-\mu X)^3/\sigma X3$	Нормальность характеристик [99]	min,max,μ,σ ,q1, q3
Куртозис	$E(X-\mu X)^4/\sigma X^4$	Нормальность характеристик [99]	min,max,μ,σ ,q1, q3
Корреляция	ρX1 X2	Взаимозависимость характеристик [99]	min,max,μ,σ ,ρXY [158]
Ковариация	covX1 X2	Взаимозависимость характеристик [99]	min,max,μ,σ ,covXY
Концентрация	tx_1x_2	Взаимозависимость характеристик [72]	min,max,μ,σ ,τXY
Разреженность	sparsity(X)	Степень дискретности	min,max,μ,σ
Гравитация	gravity(X)	Межклассовая дисперсия	
ANOVA p-значение	$p_{valx1x2}$	избыточность функций	PvalXY[158]

Коэфф. вариации	σY /μY	Вариации цели [158]	
PCA ρλ1		Дисперсия в первом ПК [99]	
Перекос PCA		Перекос первого ПК [48]	PCA kurtosis [48]
PCA 95%		Внутренняя размерность [9]	
Вероятность класса	P(C)	Дисбаланс классов [99]	min,max,μ,σ
Энтропия класса	H(C)	Важность признака [99]	
Норма. энтропия		Информативность признака [26]	min,max,μ,σ
Взаимная информация.	MI(C,X)	Внутренняя размерность [99]	min,max,μ,σ
Коэффициент неопределенности.		Важность признака [3]	min,max,μ,σ
Эквив. кол-во подвигов		Внутренняя размерность [99]	

Соотношение шум/сигнал		Зашумленность данных [99]	
дискриминация Фишера.		Классы делимости c1 , c2 [64]	См. [64]
Объем перекрытия		Перекрытие распределения классов [64]	См. [64]
Вариативность концепции		Сложность задачи [180]	См. [180]
Согласованность данных		Качество данных [76]	См. [76]
Количество узлов, листьев	|η|, |ψ |	Сложность концепции [113]	Глубина дерева
Длина ветви		Сложность концепции [113]	min,max,μ,σ
Узлы на функцию	|ηX |	Важность признака [113]	min,max,μ,σ
Листья на класс	|ψc | / |ψ|	Сложность класса [49]	min,max,μ,σ
Оставляет соглашение	nψi /n	Разделяемость классов [16]	min,max,μ,σ

Получение информации		Важность признака [16]	min,max,μ,σ , gini
Ландмаркер(1NN)	$P(q_{1NN}, t_j)$	разреженность данных	элита 1NN
Ландмаркер (дерево)	$P(q_{Tree}, t_j)$	делимость данных	пень, рандомное дерево
Ландмаркер (Лин)	$P(q_{Lin}, t_j)$	линейная делимость	дискриминант Лин
Ландмаркер (NB)	$P(q_{NB}, t_j)$	Независимость признаков [115]	Больше моделей [14, 88]
Относительный LM	$P_{a,j} - P_{b,j}$	Производительность зондирования	
Подвыборка LM	$P(\theta_i, t_j, s_t)$	Производительность зондирования	

Они обучаются путем передачи данных двух разных задач в две сети-близнецы и использования различий между прогнозируемой и наблюдаемой производительностью Pi, new в качестве сигнала ошибки. Поскольку параметры модели обеих сетей связаны в сиамскую сеть, две очень похожие задачи отображаются на одни и те же области в латентном пространстве мета-функций. Они могут быть использованы для быстрого запуска байесовской оптимизации гиперпараметров [75] и поиска нейронной архитектуры [2].

2.3.3 Оптимизация с теплым стартом на основе похожих задач

Мета-функции самый естественный способ оценить сходство задач и инициализировать процедуры оптимизации на основе перспективных конфигураций для аналогичных задач. Это сродни тому, как эксперты начинают ручной поиск хороших моделей, учитывая опыт выполнения соответствующих задач.

Во-первых, запуск алгоритма генетического поиска в областях поискового пространства с многообещающими решениями может значительно ускорить конвергенцию к хорошему решению. Гомес и соавт. [59] рекомендуют начальные конфигурации путем нахождения k наиболее похожих предыдущих задач tj на основе расстояния L1 между векторами m(tj) и m (tnew), где каждое m (tj) включает 17 простых и статистических мета-признаков. Для каждой из k наиболее похожих задач наилучшая конфигурация оценивается на tnew и используется для инициализации алгоритма генетического поиска (оптимизация роя частиц), а также поиска по Табу. Рейф и др. [129] придерживаются очень похожего подхода, используя 15 простых, статистических и привязанных к местности мета-функций. Они используют метод прямого отбора, чтобы найти наиболее полезные мета-признаки, и запускают стандартный генетический алгоритм (GAlib) с модифицированной операцией гауссовой мутации. Также были опробованы варианты активного тестирования (см. раздел 2.2.3), использующие мета-функции [85, 100], но они показали себя не лучше, чем подходы, основанные на относительных ориентирах.

Кроме того, подходы к оптимизации на основе моделей могут значительно выиграть от первоначального набора перспективных конфигураций. SCoT [9] обучает единственную суррогатную модель ранжирования $f : M \times Q \rightarrow R$, предсказывая ранг θi в задаче tj . M содержит 4 мета-функции (3 простых и одна на основе PCA). Суррогатная модель обучена на всех рейтингах, в том числе на tnew. Ранжирование используется потому, что шкала оценочных значений может сильно отличаться в разных задачах. Регрессия GP преобразует ранги в вероятности для выполнения байесовской оптимизации, и каждый новый Pi, new используется для переобучения суррогатной модели после каждого шага.

Шиллинг и др. [148] используют модифицированный многослойный персептрон в качестве суррогатной модели формы sj(θi, m(tj), b(tj))=Pi, j, где m(tj) - это мета-признаки, а b(tj) - вектор j двоичных признаков, которые равны 1, если мета-экземпляр это от tj и 0 в противном случае.Многослойный персептрон использует модифицированную функцию активации, основанную на машинах факторизации [132] на первом уровне, направленную на изучение скрытого представления для каждой задачи для моделирования сходства задач. Поскольку эта модель не может представлять неопределенности, ансамбль из 100 многослойных персептронов обучается получать средства прогнозирования и моделировать отклонения.

Обучение одной суррогатной модели на основе всех предыдущих метаданных часто менее масштабируемо. Йогатама и Манн [190] также строят единую байесовскую суррогатную модель, но включают только задачи, аналогичные t*new*, где сходство задач определяется как евклидово расстояние между векторами метаобъектов, состоящими из 3 простых метаобъектов. Значения Pi, j стандартизированы, чтобы преодолеть проблему различных масштабов для каждого tj. Суррогатная модель изучает гауссовский процесс с определенной комбинацией ядер во всех экземплярах.

Фойрер и др. [48] предлагают более простой и масштабируемый метод, который запускает байесовскую оптимизацию путем сортировки всех предварительных задач tj, аналогично [59], но включая 46 простых, статистических и ориентирующих метафункций, а также H (C). T лучших конфигураций на г наиболее похожих задачах используются для теплого запуска суррогатной модели. При этом перебирается гораздо больше гиперпараметров, чем в предыдущих работах, включая шаги предварительной обработки. Этот подход с теплым стартом был также использован в более поздней работе [46], которая подробно обсуждается в гл. 6.

Наконец, можно также использовать совместную фильтрацию, чтобы рекомендовать перспективные конфигурации [162]. По аналогии, задачи t j (пользователи) предоставляют оценки (Pi, j) для конфигураций θi (элементов), а методы матричного факторизации используются для прогнозирования неизвестных значений Pi, j и рекомендации наилучших

конфигураций для любой задачи. Важной проблемой здесь является проблема холодного запуска, поскольку факторизация матрицы требует, по крайней мере, некоторых оценок на t*new*. Янг и др. [189] используют D-оптимальный дизайн эксперимента для выборки начального набора оценок P*i, new*. Они прогнозируют как прогнозируемую производительность, так и время выполнения, чтобы рекомендовать набор конфигураций "теплого пуска", которые являются одновременно точными и быстрыми. Мисир и Себаг [102, 103] используют мета-функции для решения проблемы холодного запуска. Фуси и др. [54] также используют мета-функции, следуя той же процедуре, что и [46], и используют вероятностный матричный подход факторизации, который позволяет им выполнять байесовскую оптимизацию для дальнейшей оптимизации своих конфигураций конвейера θi . Этот подход дает полезные скрытые вложения как задач, так и конфигураций, в которых байесовская оптимизация может быть выполнена более эффективно.

2.3.4 Метамодели

Мы также можем изучить сложную взаимосвязь между мета-функциями задачи и полезностью конкретных конфигураций, построив мета-модель L, которая рекомендует наиболее полезные конфигурации Θ^*_{new} с учетом мета-функций M новой задачи t_{new}. Существует богатый объем более ранних работ [22, 56, 87, 94] по построению метамоделей для выбора алгоритма [15, 19, 70, 115] и рекомендации по гиперпараметрам [4,79, 108, 158]. Эксперименты показали, что усиленные и упакованные деревья часто дают наилучшие прогнозы, хотя многое зависит от точных используемых мета-признаков [72, 76].

2.3.4.1 Ранжирование

Метамодели также могут генерировать рейтинг из K наиболее перспективных конфигураций. Один из подходов заключается в построении мета-модели k-nearest neighbor (kNN) для предсказания того, какие задачи похожи, а затем ранжирования лучших конфигураций на этих похожих задачах [23, 147]. Это похоже на работу, рассмотренную в разделе 2.3.3, но без привязки к последующему подходу оптимизации. Метамодели, специально предназначенные для ранжирования, такие как деревья предсказательной кластеризации [171] и деревья ранжирования меток [29],

также хорошо себя зарекомендовали. Approximate Ranking Tree Forests (ART Forests) [165], ансамбли быстрых ранжирующих деревьев, оказываются особенно эффективными, поскольку они имеют "встроенный" выбор мета-функций, хорошо работают, даже если доступно мало предыдущих задач, а ансамблирование делает метод более надежным. autoBagging [116] ранжирует рабочие процессы Bagging, включая четыре различных гиперпараметра Bagging, используя ранжировщик на основе XGBoost, обученный на 140 наборах данных OpenML и 146 мета-функциях. Lorena et al. [93] рекомендует конфигурации SVM для задач регрессии, используя мета-модель kNN и новый набор мета-функций, основанный на сложности данных.

2.3.4.2 Прогнозирование производительности

Мета-модели также могут напрямую предсказывать производительность, например, точность или время обучения, конфигурации для данной задачи, учитывая ее мета-характеристики. Это позволяет нам оценить, будет ли конфигурация достаточно интересной для оценки в любой процедуре оптимизации. В ранних работах использовалась линейная регрессия или регрессоры на основе правил для прогнозирования производительности дискретного набора конфигураций и последующего их соответствующего ранжирования [14, 77]. Гуэрра и др. [61] обучают мета-регрессор SVM для каждого алгоритма классификации, чтобы предсказать его точность при настройках по умолчанию для новой задачи t_{new} с учетом ее мета-функций. Рейф и др. [130] обучают аналогичный мета-регрессор большему количеству метаданных, чтобы предсказать его оптимизированную производительность. Дэвис и др. [32] вместо этого используют многослойный метаучитель на основе персептрона, предсказывающий производительность конкретной конфигурации алгоритма.

Вместо прогнозирования прогностической производительности мета-регрессор также может быть обучен предсказывать время обучения алгоритма/прогнозирования, например, используя регрессор SVM, обученный на мета-признаках [128], сам настроенный с помощью генетических алгоритмов [119]. Янг и др. [189] прогнозируют время выполнения конфигурации с использованием полиномиальной регрессии, основываясь только на количестве экземпляров и функций. Хаттер и др.

[68] предоставляют общий трактат о прогнозировании времени выполнения алгоритма в различных областях.

Большинство из этих метамоделей генерируют многообещающие конфигурации, но на самом деле не настраивают конфигурации под t_{new}. Вместо этого прогнозы могут быть использованы для предварительного запуска или руководства любым другим методом оптимизации, который допускает всевозможные комбинации метамоделей и методов оптимизации. Действительно, некоторые работы, обсуждаемые в разделе 2.3.3, можно рассматривать как использование метамодели, основанной на расстоянии, для разогрева байесовской оптимизации [48, 54] или эволюционных алгоритмов [59, 129]. В принципе, здесь можно было бы использовать и другие метамодели.

Вместо изучения взаимосвязи между мета-функциями характеристиками задачи и производительностью конфигурации можно также построить суррогатные модели, предсказывающие производительность конфигураций для конкретных задач [40]. Затем можно узнать, как комбинировать эти прогнозы для каждой задачи для предварительного запуска или руководства методами оптимизации для новой задачи t_{new} [45, 114, 161, 187], как обсуждалось в разделе 2.2.3. Хотя метаособенности также можно использовать для объединения прогнозов для каждой задачи на основе сходства задач, в конечном счете более эффективно собирать новые наблюдения$P_{i,new}$, поскольку они позволяют нам уточнять оценки сходства задач с каждым новым наблюдением [47, 85, 187].

2.3.5 Создание конвеера

При создании целых конвейеров машинного обучения [153] количество вариантов конфигурации резко возрастает, что делает еще более важным использование предыдущего опыта. Можно управлять пространством поиска, накладывая на конвейер фиксированную структуру, полностью описываемую набором гиперпараметров. Затем можно использовать наиболее перспективные конвейеры для аналогичных задач, чтобы запустить байесовскую оптимизацию [46, 54].

Другие подходы дают рекомендации для определенных этапов трубопровода [118, 163] и могут быть использованы в более крупных подходах к строительству трубопровода, таких как планирование [55, 74,

105, 184] или эволюционные методы [110, 164]. Нгуен и др. [105] создают новые конвейеры, используя поиск по лучу, ориентированный на компоненты, рекомендованные мета-учеником, и сам обучается на примерах успешных предыдущих конвейеров. Билаллиет и др. [18] предсказывают, какие методы предварительной обработки рекомендуются для данного алгоритма классификации. Они строят метамодель для каждого целевого алгоритма классификации, который, учитывая t_{new} мета-признаки, предсказывает, какой метод предварительной обработки следует включить в конвейер. Аналогичным образом, Шенфельд и др. [152] строят метамодели, предсказывающие, когда алгоритм предварительной обработки улучшит точность или время выполнения конкретного классификатора.

AlphaD3M [38] использует подход самостоятельного обучения с подкреплением, в котором текущее состояние представлено текущим конвейером, а действия включают добавление, удаление или замену компонентов конвейера. Поиск по дереву Монте-Карло (MCTS) генерирует конвейеры, которые оцениваются для обучения рекуррентной нейронной сети (LSTM), которая может прогнозировать производительность конвейера, в свою очередь, создавая вероятности действий для MCTS в следующем раунде. Описание состояния также включает в себя метафизические характеристики текущей задачи, что позволяет нейронной сети обучаться между задачами. Mosaic [123] также генерирует конвейеры с использованием MCT, но вместо этого использует основанный на результатах подход для выбора перспективных конвейеров.

2.3.6 Настраивать или не настраивать?

Чтобы уменьшить количество оптимизируемых параметров конфигурации и сэкономить ценное время оптимизации в условиях ограниченного времени, мета-модели также были предложены, чтобы предсказывать, стоит ли настраивать данный алгоритм с учетом мета-признаков поставленной задачи [133] и сколько улучшений мы можем ожидать от настройки конкретного алгоритма по сравнению с дополнительными временными затратами [144]. Более целенаправленные исследования конкретных алгоритмов обучения привели к созданию мета-моделей, предсказывающих, когда необходимо настраивать метод опорных векторов [96], какие гиперпараметры по умолчанию хорошо подходят для

метода опорных векторов с учетом поставленной задачи (включая интерпретируемые метамодели) [97], и как настроить деревья решений [95].

2.4 Обучение с помощью предыдущих моделей

Последний тип метаданных, с помощью которых мы можем обучать, — это сами предыдущие модели машинного обучения, т. е. их структура и параметры обученной модели. Короче говоря, мы хотим обучить мета-ученика L, который узнаёт, как обучить (базового) ученика l_{new} для новой задачи t_{new}, имея подобные задачи $t_j \in T$ и соответствующие оптимизированные модели $l_i \in L$, где L — пространство всех возможных моделей. Ученик l_j обычно определяется его параметрами модели $W = \{w_k\}, k = 1 \ldots K$ и/или его конфигурацией $\theta_i \in \Theta$.

2.4.1 Трансферное обучение

В трансферном обучении [170] мы берем модели, обученные на одной или нескольких исходных задачах , и используем их в качестве отправных точек для создания модели на аналогичной целевой задаче . Это можно сделать, заставив целевую модель быть структурно или иным образом похожей на исходную модель(и). Это общеприменимая идея, и подход трансферного обучения был предложен для методов ядра [41, 42], параметрической байесовской модели [8, 122, 140], байесовских сетей [107], кластеризации [168] и обучения с подкреплением [36, 62]. Однако нейронные сети особенно подходят для трансферного обучения, потому что и структура, и параметры исходных моделей могут использоваться в качестве хорошей инициализации для целевой модели, что дает предварительно обученную модель, которую затем можно дополнительно настроить, используя доступные обучающие данные, на [11, 13, 24, 169]. В некоторых случаях может потребоваться изменить исходную сеть перед трансферным обучением [155]. Мы сосредоточимся на нейронных сетях в оставшейся части этого раздела.

Было показано, что особенно большие датасеты с изображениями, такие как ImageNet [78], дают предварительно обученные модели, которые особенно хорошо переносятся на другие задачи [37, 154]. Однако, также было показано, что этот подход не работает, когда целевая задача не так уж похожа [191]. Вместо того, чтобы надеяться, что предварительно обученная

модель «случайно» хорошо переносится на новую проблему, мы можем целенаправленно наполнить мета-учеников индуктивной предвзятостью (извлеченной из многих похожих задач), которая позволит им обучаться на новых задачах гораздо быстрее, как мы обсудим ниже.

2.4.2 Мета-обучение для нейронных сетей

Ранний подход к мета-обучению заключается в создании рекуррентных нейронных сетей (RNN), способных изменять собственные веса [149, 150]. Во время обучения они используют собственные веса в качестве дополнительных входных данных и наблюдают за своими ошибками, чтобы научиться модифицировать веса для новой задачи. Обновление весов определяется в параметрической форме, дифференцируемо от начала до конца и может совместно оптимизировать и сеть, и алгоритм обучения с использованием градиентного спуска, но при этом его очень сложно обучать. В более поздних работах использовалось обучение с подкреплением для разных задач, чтобы адаптировать поиск стратегии [151] или скорости обучения для градиентного спуска [31] к поставленной задаче.

Вдохновленные мыслью, что обратное распространение — маловероятный механизм обучения для человеческого мозга, Бенжио и др. [12] заменили обратное распространение ошибки на простые вдохновленные биологией параметрические правила (или развитые правила [27]) для обновления синаптических весов. Параметры оптимизируются, например, с использованием градиентного спуска или эволюции через набор входных задач. Рунарссон и Йонссон [142] заменили эти параметрические правила однослойной нейронной сетью. Санторо и др. [146] вместо этого использовали нейронную сеть с дополнительной памятью, чтобы научиться хранить и извлекать «воспоминания» о предыдущих классификационных задачах. Хохрайтер и др. [65] использовали сеть долгой краткосрочной памяти [66] в качестве мета-обучения для обучения многослойных перцептронов.

Андрухович и др. [6] также заменили оптимизатор, например, стохастический градиентный спуск, сетью долгой краткосрочной памяти, обученной для выполнения нескольких предыдущих задач. Потеря мета-ученика (оптимизатора) определяется как сумма потерь базовых учеников

(оптимизируемых) и оптимизируется с помощью градиентного спуска. На каждом этапе мета-обучаемый выбирает обновление весов по оценкам, максимально снижая потери оптимизатора на основе веса обученной модели с предыдущего шага, а также текущий градиент производительности. Потом работа обобщает этот подход, обучая оптимизатор на синтетических функциях с использованием градиентного спуска [28]. Это позволяет мета-ученикам оптимизировать оптимизируемого, даже если у них нет доступа к градиентам.

Параллельно Ли и Малик [89] предложили фреймворк для оптимизации обучения с точки зрения обучения с подкреплением. Он представляет какой-либо конкретный алгоритм оптимизации как линию поведения, а затем изучает эту линию поведения с помощью управляемого поиска. Последующая работа [90] показывает, как использовать этот подход для обучения алгоритмов оптимизации для (мелких) нейронных сетей.

Область поиска нейронной архитектуры включает в себя множество других методов, которые строят модель производительности нейронной сети для конкретной задачи, например, с использованием байесовской оптимизации или обучения с подкреплением (см. гл. 3 для подробного обсуждения). Однако большинство этих методов (пока) не обобщаются на задачи и поэтому здесь не обсуждаются.

2.4.3 Дообучение на малых выборках

Особенно сложной проблемой метаобучения является обучение точной глубокой модели, с использованием всего нескольких обучающих примеров, учитывая предыдущий опыт работы с очень похожими задачами, для которых у нас есть большие обучающие выборки. Это называется - обучение на малых выборках. У людей есть врожденная способность делать это, и мы хотим построить агенты машинного обучения, которые могут делать то же самое [82]. Частным примером этого является классификация «K-shot N-way», в которой нам дается много примеров (например, изображений) определенных классов (например, объектов) и хотим обучить новый классификатор I_{new}, способный классифицировать N новых классов, для каждого из которых есть K примеров.

Используя предыдущий опыт, мы можем, например, изучить представление общих признаков всех задач, начать обучение I_{new} заново с

лучшей инициализацией параметров W_{init} модели и получить индуктивное смещение, которое помогает оптимизировать модель параметры, так что I_{new} можно обучить намного быстрее, чем в других случаях.

Предыдущая работа над обучением с первого раза в значительной степени основывалась на функциях, разработанных вручную. [10, 43, 44, 50]. Однако с помощью мета-обучения мы надеемся найти общую черту представления всех задач от начала до конца.

Виньялс и др. [181] утверждают, что, чтобы учиться на очень небольшом количестве данных, нужно обратиться к непараметрическим моделям (таким как k-ближайших соседей), которые используют память компонента, а не изучение многочисленных параметров модели. Их мета-учитель Matching Network реализует идею компонента памяти в нейронной сети. Он изучает общее представление для размеченных примеров и сопоставляет каждый новый тестовый экземпляр с примерами из памяти, используя косинусную меру. Сеть обучается на мини-пакетах, в каждом из которых всего несколько примеров для каждой задачи.

Снелл и др. [157] предлагают прототипы сетей, которые отображают примеры в p-мерное векторное пространство, в котором примеры для каждого конкретного класса близки друг к другу. Затем вычисляется прототип (средний вектор) для каждого класса. Новые тестовые экземпляры сопоставляются с одним и тем же векторным пространством, а метрика расстояния используется для создания softmax-функции над всеми классами. Рен и др. [131] распространяют этот подход на полуконтролируемое обучение.

Рави и Ларошель [126] используют мета-обучение на основе LSTM для изучения правил обновления учителя нейросети. С каждым новым примером учащийся

возвращает текущий градиент и потери в мета-обучающий модуль LSTM, который затем обновляет параметры модели $\{w_k\}$ учащегося. Мета-обучаемый проходит обучение по всем предыдущим задачам.

Модельно-независимое мета-обучение (MAML) [51], с другой стороны, не пытается узнавать правило обновления, но вместо этого изучает инициализацию параметров модели W_{init}, которая лучше обобщает аналогичные задачи. Начиная со случайного $\{w_k\}$, он итеративно выбирает пакет предыдущих задач, и для каждой он обучает учащегося на K примерах вычислять градиент и потери (на тестовой выборке). Затем он распространяет метаградиент для обновления весов $\{w_k\}$ в том

направлении, в котором их было бы легче обновить. Другими словами, после каждой итерации веса $\{w_k\}$ становятся лучше W_{init} для запуска тонкой настройки любой из задач. Финн и Левин [52] также утверждают, что MAML способен аппроксимировать любой алгоритм обучения при использовании достаточно глубокой полносвязной сети ReLU при определенных потерях. Они также пришли к выводу, что инициализация MAML более устойчива к переобучению на небольших выборках и её обобщение шире, чем обобщение подходов метаобучения, основанных на LSTM.

REPTILE [106] представляет собой аппроксимацию MAML, которая выполняет стохастический градиент спуска за K итераций по заданной задаче, а затем постепенно перемещает инициализацию весов в направлении весов, полученных после K итераций. Мысль заключается в том, что каждая задача, вероятно, имеет более одного набора оптимальных весов {w∗ i}, а цель состоит в том, чтобы найти W_{init}, близкий хотя бы к одному из {w∗ i} для каждой задачи. Наконец, мы также можем получить мета-обучение из black-box нейронной сети. Санторо и др. [145] предлагают нейронные сети с расширенной памятью (MANN), которые обучают нейронную машину Тьюринга (NTM) [60], нейронную сеть с дополненными возможностями памяти, как мета-ученик. Затем этот мета-ученик может запомнить информацию о предыдущих задачах и использовать ее для обучения учащегося I_{new}.

SNAIL [101] представляет собой универсальную архитектуру мета-учителя, состоящую из чередующихся слоев временной свертки и причинно-следственных связей. Сверточные сети изучают общий вектор признаков для учебных экземпляров (изображений) для агрегирования информации из прошлого опыта. Слои каузального внимания узнают, какие фрагменты информации выбирать из накопленного опыта для обобщения на новые задачи.

В целом, пересечение глубокого обучения и метаобучения - особая благодатная почва для новаторских идей, и мы ожидаем, что эта область со временем станут намного более важными.

2.4.4 Обучение без размеченных данных

Мета-обучение, безусловно, не ограничивается (полу)контролируемыми задачами и успешно применяется для решения таких разнообразных задач, как обучение с подкреплением, активное

обучение, оценка плотности и рекомендации по элементам. Базовый ученик может быть без присмотра, в то время как мета-ученик находится под присмотром, но другие комбинации конечно также возможны.

Дуан и др. [39] предлагают сквозной подход к обучению с подкреплением (RL),состоящий из быстрого алгоритма RL для конкретной задачи, который руководствуется универсальным медленным алгоритмом мета-RL. Задачи связаны через Марковские процессы принятия решений (MDP - Markov Decision Process). Алгоритм мета-RL моделируется как RNN, который получает наблюдения, действия, награды и флаги завершения. Активации RNN содержат состояние быстрого RL-учителя, а веса RNN изучаются путем наблюдения производительности учителя при выполнении задач.

Параллельно Wang [182] также предложил использовать алгоритм глубокого RL для обучения RNN, на основе результатов предыдущих интервалов.

Панг и др. [112] предлагают мета-обучающий подход к активному обучению (AL).

base-learner может быть любым бинарным классификатором, а meta-learner — глубокой RL сетью, состоящей из глубокой нейронной сети, которая изучает представление проблемы AL между задачами и сеть политик, которая изучает оптимальную политику, параметризованную как веса в сети. Мета-ученик получает текущее состояние (немаркированный набор баллов и состояние базового классификатора) и вознаграждение (производительность базового классификатора) и выдает вероятность запроса, т. е. какие точки в немаркированном наборе для дальнейшего рассмотрения.

Рид и др. [127] предлагают метод дооубучения на малых выборках для оценки плотности (DE). Цель состоит в том, чтобы узнать распределение вероятностей по небольшому количеству изображений на определенную тему (например, рукописное письмо), которое можно использовать для создания изображений на ту же тему этого понятия или вычисления вероятности того, что изображение относится к этой теме. Подход использует авторегрессионные модели изображений, которые факторизуют совместное распределение на попиксельные коэффициенты. Обычно они основаны на большой выборке изображений на конкретную тему. Обучающий модуль на основе MAML, напротив, обучается на

небольшой выборке изображений на конкретную тему и большой выборке на различные другие темы.

Наконец, Вартак и соавторы. [178] обращаются к проблеме холодного старта при матричной факторизации. Они предлагают архитектуру глубокой нейронной сети, которая обучает (базовую) нейронную сеть, чьи смещения корректируются на основе информации о задаче. При этом структура и веса рекомендателей нейронной сети остаются фиксированными, мета-обучение узнает, какие смещения использовать на основе каждой user story.

Все эти недавние новые разработки показывают, что часто бывает полезно взглянуть на проблемы через призму мета-обучения и найти новые, основанные на данных подходах и заменить созданных вручную базовых учеников.

2.5 Заключение

Возможности мета-обучения проявляются по-разному и могут быть охвачены с использованием широкого спектра методов обучения. Каждый раз, когда мы пытаемся
исследовать новую задачу, независимо от того, успешна или нет, мы приобретаем полезный опыт, который можем использовать для изучения новых задач. Мы никогда не должны начинать полностью с нуля. Вместо этого мы должны систематически собирать наш «учебный опыт», чтобы создавать на его основе системы AutoML, которые, затем, совершенствуются, помогая нам решать новые проблемы обучения еще более эффективно. Чем больше новых задач похожих на старые, тем больше мы можем использовать предшествующий опыт, до такой степени, что большая часть исследований может быть уже сделана. Способность компьютерных систем хранить практически бесконечное количество предыдущего опыта обучения (в виде метаданных) открывает широкий диапазон возможностей для использования предыдущего опыта совершенно по-новому. Наша цель[6] - научиться решать любую задачу - дает нам гораздо больше возможностей, чем знание того, как учиться
любая конкретная задача.

Глава 3. Поиск нейронной архитектуры

Томас Элскен, Ян Хендрик Метцен и Фрэнк Хаттер

Аннотация

За последние годы глубокое обучение позволило добиться значительного прогресса в решении различных задач - распознавание изображений и речи и машинный перевод. Одним из важнейших аспектов этого прогресса являются новые нейронные архитектуры. Используемые в настоящее время архитектуры в основном разрабатывались вручную специалистами, что является трудоемким и подверженным ошибкам процессом. Из-за этого растет интерес к автоматизированным методам поиска нейронной архитектуры. Мы предоставляем обзор существующих работ в этой области исследований и классифицируем их в соответствии с тремя измерениями: пространство поиска, стратегия поиска и стратегия оценки эффективности.

3.1 Введение

Успех глубокого обучения в перцептивных задачах во многом обусловлен автоматизацией процесса разработки свойств: иерархические средства извлечения свойств изучаются сквозным способом на основе данных, а не разрабатываются вручную. Однако этот успех сопровождался растущим спросом на архитектурную инженерию, где все более сложные нейронные архитектуры разрабатываются вручную. Поиск нейронной архитектуры (NAS), процесс автоматизации проектирования архитектуры, таким образом, является логичным следующим шагом в автоматизации машинного обучения. NAS можно рассматривать как подобласть AutoML и имеет значительное совпадение с оптимизацией гиперпараметров и мета-обучением (которые описаны в главах. 1 и 2 этой книги соответственно).

Мы классифицируем методы для NAS в соответствии с тремя измерениями: пространство поиска, стратегия поиска и стратегия оценки производительности:

• **Пространство поиска**. Пространство поиска определяет, какие архитектуры могут быть представлены в принципе. Объединение предварительных знаний о свойствах, хорошо подходящих для выполнения задачи, может уменьшить размер поискового пространства и упростить поиск. Однако это также приводит к человеческому отклонению, которое может помешать поиску новых архитектурных строительных блоков, выходящих за рамки текущих человеческих знаний.

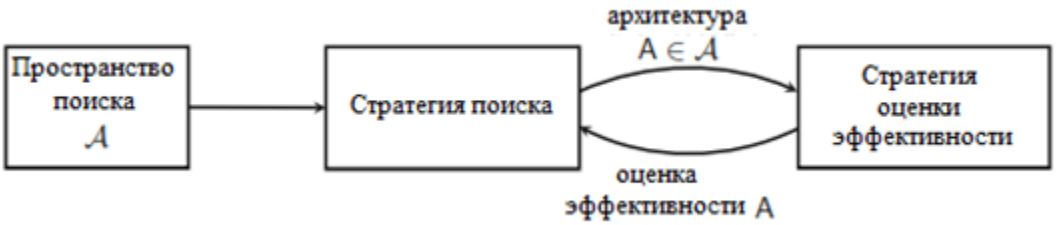

Рис. 3.1 Абстрактная иллюстрация методов поиска нейронной архитектуры. Стратегия поиска выбирает архитектуру A из предопределенного пространства поиска *A*. Архитектура передается стратегии оценки производительности, которая возвращает оцененную производительность A для стратегии поиска

• **Стратегия поиска**. Стратегия поиска подробно описывает, как исследовать пространство поиска. Это включает в себя классический компромисс между исследованием и эксплуатацией, поскольку, с одной стороны, желательно быстро находить хорошо работающие архитектуры, в то время как, с другой стороны, следует избегать преждевременного сближения в области неоптимальных архитектур.

• **Стратегия оценки эффективности**. Целью NAS обычно является поиск архитектур, которые обеспечивают высокую производительность прогнозирования для незнакомых данных. Оценка производительности относится к процессу оценки этой производительности: самый простой вариант - выполнить стандартное обучение и проверку архитектуры на данных, но это, к сожалению, требует больших вычислительных затрат и ограничивает количество архитектур, которые могут быть исследованы. Поэтому многие недавние исследования

сосредоточены на разработке методов, которые снижают стоимость этих оценок производительности.

Мы ссылаемся на рис. 3.1 для иллюстрации. Глава также структурирована в соответствии с этими тремя аспектами: мы начинаем с обсуждения поисковых пространств в разделе 3.2, рассматриваем стратегии поиска в разделе 3.3 и описываем подходы к оценке эффективности в разделе 3.4. В заключение мы даем обзор будущих направлений в разделе 3.5.

Эта глава основана на совсем недавней обзорной статье [23].

3.2 Пространство поиска

Пространство поиска определяет, какие нейронные архитектуры в принципе может обнаружить подход NAS. Теперь мы обсудим общие пространства поиска из недавних работ.

Относительно простым пространством поиска является пространство нейронных сетей с цепной структурой, как показано на рис. 3.2.

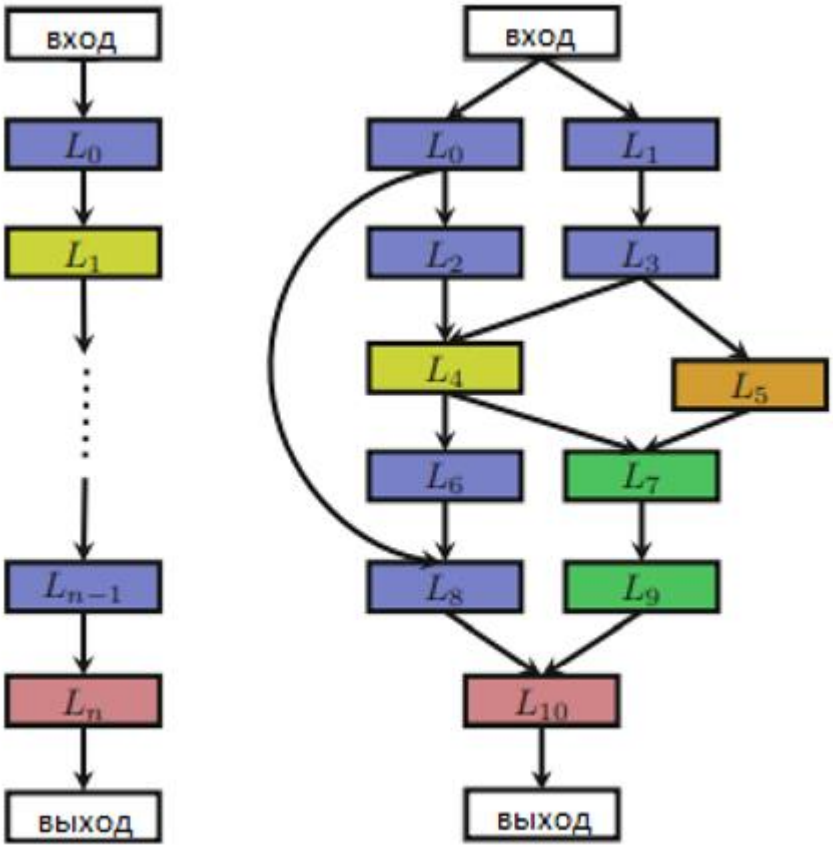

Рис.3.2 Иллюстрация различных архитектурных пространств. Каждый узел на графиках соответствует слою в нейронной сети, например, сверточному или объединяющему слою. Различные типы слоев визуализируются разными цветами. Ребро от слоя *Li* к слою *Lj* обозначает, что *Lj* получает выходные данные *Li* в качестве входных данных. Слева: элемент пространства, структурированного цепочкой. Справа: элемент более сложного пространства поиска с дополнительными типами слоев и множеством ответвлений и пропусков соединений.

Архитектура нейронной сети с цепной структурой A может быть записана как последовательность из n слоев, где $i-$ й слой L_i получает свои входные данные от слоя $i\ -\ 1$, а его выходные данные служат входными данными для слоя $i\ +\ 1$, т.е. $A = L_n\ \circ \ldots\ L_1\ \circ\ L_0$. Затем пространство поиска параметризуется: (i) (максимальным) количеством слоев n (возможно, неограниченным); (ii) типом операции, которую может выполнять каждый слой, например, объединение, свертка или более продвинутые типы слоев, такие как разделяемые по глубине свертки или расширенные свертки.; и (iii) гиперпараметры, связанные с операцией, например, количество фильтров, размер ядра и шаги для сверточного слоя или просто количество единиц для полностью подключенных сетей. Обратите внимание, что параметры из (iii) обусловлены (ii), следовательно, параметризация пространства поиска является не фиксированной длиной, а скорее условным пространством.

Недавние работы над NAS включают современные элементы дизайна, известные по архитектуре ручной работы, такие как пропускные соединения, которые позволяют строить сложные, многоотраслевые сети, как показано на рис. 3.2 (справа). В этом случае входные данные слоя i могут быть формально описаны как функция $gi(L_{i-1}^{out},\ldots,L_0^{out})$ объединяющая выходные данные предыдущего слоя. Использование такой функции приводит к значительно большему количеству степеней свободы. Частными случаями этих многоотраслевых архитектур являются (i) сети с цепной структурой (путем установки $g_i\ (L_{i-1}^{out},\ldots,L_0^{out}\ ) = L_{i-1}^{out}$), (ii) остаточные сети i−1 0 i−1, где суммируются выходные данные предыдущего уровня $(g_i\ (L_{i-1}^{out}\ ,\ldots,L_0^{out}\ ) = L_{i-1}^{out}\ ,\ldots,L_j^{out}\ ,j\ <\ i)$, (iii) плотные сети, где выходные данные предыдущего уровня объединены $\left(gi\ (L_{i-1}^{out}\ ,\ldots,L_0^{out}) =\ concat\ (L_{i-1}^{out}\ ,\ldots,L_0^{out}\ )\right)$.

Мотивированные архитектурой ручной работы, состоящей из повторяющихся мотивов, Зоф и др. и Чжун и др. предлагают искать такие мотивы, названные соответственно ячейками или блоками, а не целые архитектуры. Зоф и др. оптимизируют ячейки двух разных типов: нормальную ячейку, которая сохраняет размерность входных данных, и уменьшенную ячейку, которая уменьшает пространственное измерение. Затем создается окончательная архитектура путем укладки этих ячеек предопределенным образом, как показано на рис.3.3.

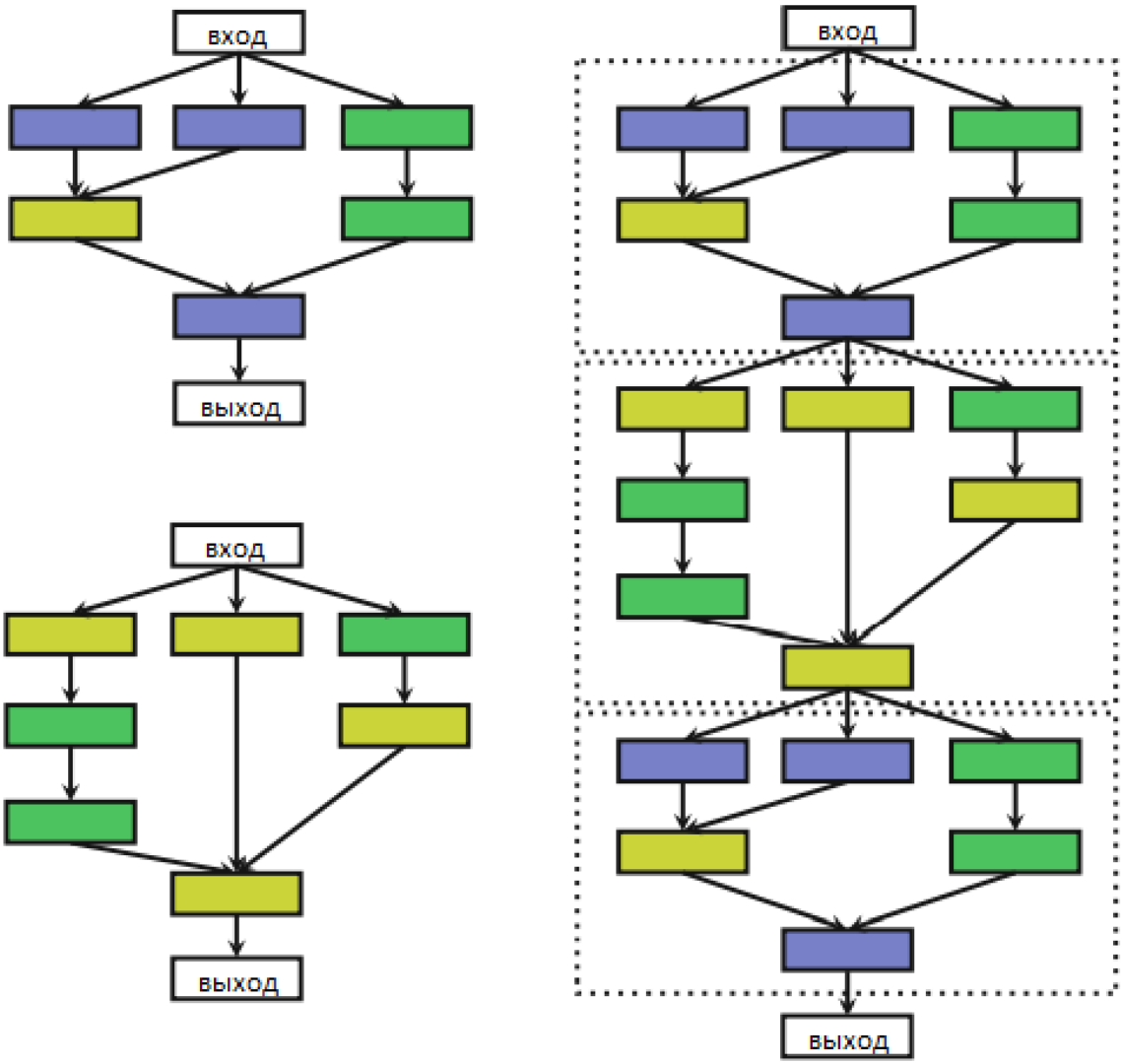

Рис.3.3 Иллюстрация пространства поиска ячейки. Слева: две разные ячейки, например, нормальная ячейка (вверху) и уменьшенная ячейка (внизу). Справа: архитектура, построенная путем последовательного размещения ячеек. Обратите внимание, что ячейки также можно комбинировать более сложным образом, например, в пространствах с несколькими ответвлениями, просто заменяя слои ячейками

Это пространство поиска имеет два основных преимущества по сравнению с теми, которые обсуждались выше:

1. Размер пространства поиска резко сокращается, поскольку ячейки могут быть сравнительно небольшими. Например, Зоф и др. оценивают семикратное ускорение по сравнению с их предыдущей работой при достижении лучшей производительности.

2. Ячейки можно легче перенести в другие наборы данных, адаптировав количество ячеек, используемых в модели. Действительно,

Зоф и др. переносят ячейки, оптимизированные на CIFAR-10, в ImageNet и достигают самой современной производительности.

Вследствие этого, это пространство поиска на основе ячеек также успешно использовалось во многих более поздних работах. Однако при использовании пространства поиска на основе ячеек возникает новый дизайнерский выбор, а именно, как выбрать метаархитектуру: сколько ячеек следует использовать и как они должны быть соединены для построения фактической модели? Например, Зоф и др. строят последовательную модель из ячеек, в которой каждая ячейка получает выходные данные двух предыдущих ячеек в качестве входных данных, в то время как Cai и др. используют высокоуровневую структуру хорошо известных вручную разработанных архитектуры, такие как DenseNet, и используют свои ячейки в этих моделях. В принципе, ячейки могут быть объединены произвольно, например, в пределах пространства с несколькими ответвлениями, описанного выше, путем простой замены слоев ячейками. В идеале, метаархитектура должна быть оптимизирована автоматически как часть NAS; в противном случае это может легко заканчиться тем, что занимаются разработкой метаархитектуры, и поиск ячейки становится чрезмерно простым, если большая часть сложности уже учтена метаархитектурой.

Одним из шагов в направлении оптимизации метаархитектур является иерархическое пространство поиска, введенное Лиу и др., которое состоит из нескольких уровней мотивов. Первый уровень состоит из набора примитивных операций, второй уровень из различных мотивов, которые соединяют примитивные операции с помощью прямых ациклических графов, третий уровень мотивов, которые кодируют, как соединять мотивы второго уровня, и так далее. Пространство поиска на основе ячеек можно рассматривать как частный случай этого иерархического пространства поиска, где количество уровней равно трем, мотивы второго уровня соответствуют ячейкам, а третий уровень представляет собой жестко закодированную метаархитектуру.

Выбор пространства поиска в значительной степени определяет сложность задачи оптимизации: даже для случая пространства поиска, основанного на одной ячейке с фиксированной метаархитектурой, проблема оптимизации остаётся (i) непрерывной и (ii) относительно многомерной (поскольку более сложные модели имеют тенденцию лучшей производительности, что приводит к широкому выбору дизайна). Мы

отмечаем, что архитектуры во многих пространствах поиска могут быть записаны в виде векторов фиксированной длины; например, пространство поиска для каждой из двух ячеек Зоф и др. может быть записано в виде 40-мерного пространства поиска с категориальными измерениями, каждое из которых выбирает между небольшим количеством различных строительных блоков и входных данных. Аналогично, неограниченные пространства поиска могут быть ограничены максимальной глубиной, что приводит к созданию пространств поиска фиксированного размера с (потенциально многими) условными измерениями.

В следующем разделе мы обсудим стратегии поиска, которые хорошо подходят для такого рода поисковых пространств.

3.3 Стратегии поиска

Для исследования пространства нейронных архитектур можно использовать множество различных стратегий поиска, включая случайный поиск, байесовскую оптимизацию, эволюционные методы, обучение с подкреплением (RL) и методы, основанные на градиенте. Исторически сложилось так, что эволюционные алгоритмы уже использовались многими исследователями для разработки нейронных архитектур (а часто и их весов) десятилетия назад [см., например, 2, 25, 55, 56]. Яо [67] приводит обзор литературы по работам, выполненным ранее 2000 года.

Байесовская оптимизация отметила несколько ранних успехов в NAS с 2013 года, что привело к созданию ультрасовременных архитектур видения [7], высочайшей производительности CIFAR-10 без увеличения данных [19] и первых автоматически настраиваемых нейронных сетей, которые выиграли в конкурентной борьбе с наборами данных экспертов-людей. [41]. NAS стал основной темой исследований в сообществе машинного обучения после того, как Зоф и Ле [74] добились конкурентных результатов в тестах CIFAR-10 и Penn Treebank с помощью стратегии поиска, основанной на обучении с подкреплением. В то время как Zoph и Le [74] используют огромные вычислительные ресурсы для достижения этого результата (800 графических процессоров в течение трех-четырех недель), после их работы было быстро опубликовано множество методов, позволяющих снизить вычислительные затраты и добиться дальнейшего улучшения производительности. Сформулировать[7] NAS как проблему обучения с подкреплением (RL) [4, 71, 74, 75], генерацию нейронной архитектуры можно рассматривать как действие агента, причем пространство действий идентично пространству поиска. Вознаграждение агента основано на оценке производительности обученной архитектуры на невидимых данных (см. раздел 3.4). Различные подходы RL отличаются тем, как они представляют политику агента и как они ее оптимизируют: Zoph и Le [74] используют политику рекуррентной нейронной сети (RNN) для последовательной выборки строки, которая, в свою очередь, кодирует нейронную архитектуру. Первоначально они обучили эту сеть с помощью алгоритма градиента политики УСИЛЕНИЯ, но в последующей работе вместо этого использовали оптимизацию проксимальной политики (PPO) [75]. Бейкер и др. [4] используйте[8] Q-learning для обучения политики, которая последовательно выбирает тип слоя и соответствующие

гиперпараметры. Альтернативный взгляд на эти подходы заключается в последовательных процессах принятия решений, в которых политика последовательно выбирает действия для создания архитектуры, “состояние” среды содержит сводку действий, выбранных на данный момент, и (недисконтированное) вознаграждение получается только после окончательного действия. Однако, поскольку во время этого последовательного процесса не происходит никакого взаимодействия со средой (не наблюдается внешнего состояния и нет промежуточных вознаграждений), мы считаем более интуитивным интерпретировать процесс выборки архитектуры как последовательную генерацию одного действия; это упрощает проблему RL для многорукого бандита без состояния. проблема. Аналогичный подход был предложен Cai и др. [10], которые рассматривают NAS как последовательный процесс принятия решений: в их подходе состоянием является текущая (частично обученная) архитектура, наградой является оценка производительности архитектуры, а действие соответствует применению сохраняющих функцию мутаций., получивший название сетевых морфизмов [12, 63], см. также раздел 3.4, за которым следует этап обучения сети. Чтобы иметь дело с сетевыми архитектурами переменной длины, они используют двунаправленный LSTM для кодирования архитектур в представление фиксированной длины. Основываясь на этом закодированном представлении, сети актеров принимают решение о выбранном действии. Комбинация этих двух компонентов составляет политику, которая обучается от начала до конца с помощью алгоритма градиента политики УСИЛЕНИЯ. Мы отмечаем, что этот подход не позволит дважды посетить одно и то же состояние (архитектуру), так что от политики требуется сильное обобщение по пространству архитектуры.

Альтернативой использованию RL являются нейроэволюционные подходы, которые используют эволюционные алгоритмы для оптимизации нейронной архитектуры. Первый такой подход к проектированию нейронных сетей, о котором нам известно, появился почти три десятилетия назад: Миллер и др. [44] используют генетические алгоритмы для предложения архитектур и используют обратное распространение для оптимизации их весов. С тех пор многие нейроэволюционные подходы [2, 55, 56] используют генетические алгоритмы для оптимизации как нейронной архитектуры, так и ее весов; однако при масштабировании до современных нейронных архитектур с миллионами весов для задач

контролируемого обучения оптимизация веса на основе SGD более поздних нейроэволюционных подходов [22, 38, 43, 49, 50, 59, 66] поэтому снова используйте методы, основанные на градиенте, для оптимизации весов и используйте исключительно эволюционные алгоритмы для оптимизации сама нейронная архитектура. Эволюционные алгоритмы развивают совокупность моделей, т.е. набор (возможно, обученных) сетей; на каждом этапе эволюции отбирается по крайней мере одна модель из популяции, которая служит родительской для создания потомства путем применения к ней мутаций. В контексте NAS мутации - это локальные операции, такие как добавление или удаление слоя, изменение гиперпараметров слоя, добавление пропущенных соединений, а также изменение обучающих гиперпараметров. После обучения потомков оценивается их пригодность (например, производительность в проверочном наборе), и они добавляются в популяцию.

Нейроэволюционные методы отличаются тем, как они отбирают родителей, обновляют популяцию-связи и порождать потомство. Например, Real et al. [50], Real et al. [49] и Liu et al. [38] используют турнирную выборку [27] для выборки родителей, тогда как Elsken et al. [22] выбирают родителей из многоцелевого фронта Парето, используя обратную плотность. Реал и др. [50] удаляют худшую особь из популяции, в то время как Реал и др. [49] сочли полезным удалить самую старую особь (что уменьшает жадность), а Лю и др. [38] вообще не удаляют особей. Чтобы генерировать потомство, большинство подходов инициализируют дочерние сети случайным образом, в то время как Элскен и др. [22] используют ламаркианское наследование, то есть знания (в форме изученных весов) передаются от родительской сети к ее дочерним элементам с помощью сетевых морфизмов. Real и др. [50] также позволяют потомству наследовать все параметры своего родителя, на которые не влияет применяемая мутация; хотя это наследование не является строго функциональным, оно также может ускорить обучение по сравнению со случайной инициализацией. Более того, они также позволяют изменять скорость обучения, что можно рассматривать как способ оптимизации графика скорости обучения во время NAS.

Реал и др. [49] провели тематическое исследование, сравнивая RL, эволюцию и случайный поиск (RS) и пришли к выводу, что RL и эволюция работают одинаково хорошо с точки зрения конечной точности теста, при

этом эволюция имеет лучшую производительность в любое время и находит более компактные модели. В экспериментах оба подхода постоянно показывают лучшие результаты, чем RS,
но с довольно небольшим отрывом: RS достиг тестовой ошибки примерно в 4% на
CIFAR-10, в то время как RL и эволюция достигли приблизительно 3,5% (после "увеличения модели", при которой глубина и количество фильтров были увеличены; разница на реальном, не дополненном пространстве поиска составила около 2%). Разница была еще еще меньше для Лю и др. [38], которые сообщили о тестовой ошибке 3,9% на CIFAR-10 и top-1 ошибке валидации 21,0% на ImageNet для RS, по сравнению с 3,75% и 20,3% для их метода, основанного на эволюции, соответственно.

Байесовская оптимизация (БО, см., например, [53]) является одним из самых популярных методов для оптимизации гиперпараметров (см. также гл. 1 этой книги), но многие группы не применяли ее к NAS, поскольку типичные инструментальные средства BO основаны на гауссовских процессах и сосредоточены на низкоразмерных непрерывных оптимизационных задачах. Сверски и др [60] и Кандасами и др [31] вывели функции ядра для архитектурных пространств поиска, чтобы использовать классические методы БО на основе GP, но пока что без достижения новой современной производительности. Напротив, в нескольких работах используются древовидные модели (в частности, древовидные оценки Парзена [8] или случайные леса [30]) для эффективного поиска в очень высокоразмерных условных пространствах и достигают передовой производительности для широкого круга задач, оптимизируя как нейронные архитектуры, так и их гиперпараметры [7, 19, 41, 69]. Хотя полное сравнение отсутствует, есть предварительные свидетельства того, что эти подходы могут также превзойти эволюционные алгоритмы [33].

Пространства архитектурного поиска также были исследованы иерархически, например, в комбинации с эволюцией [38] или путем последовательной основанной на модели оптимизации [37]. Негриньо и Гордон [45] и Вистуба [65] используют древовидную структуру их пространства поиска и используют поиск по дереву Монте-Карло. Элскен и др. [21] предлагают простой, но хорошо работающий алгоритм подъема на холм, который обнаруживает высококачественные архитектуры путем жадного движения в направлении более эффективных архитектур, не требуя более сложных механизмов поиска.

В отличие от описанных выше методов оптимизации без градиента, Лю и др. [39] предлагают непрерывную релаксацию пространства поиска для обеспечения градиентной оптимизации: вместо того, чтобы фиксировать одну операцию o_i(например, свертку или объединение), которая должна быть выполнена на определенном слое, авторы вычисляют выпуклую комбинацию из набора операций $\{o_1, \ldots, o_m\}$. Более конкретно, для данного входного слоя x, выход слоя y вычисляется как $y = \sum_{i=1}^{m} \lambda_i o_i(x), \lambda_i \geq 0, \sum_{i=1}^{m} \lambda_i = 1$, где выпуклые коэффициенты λ_i эффективно параметризуют архитектуру сети. Лю и др. [39] затем оптимизируют веса сети и архитектуру сети, чередуя шаги градиентного спуска на обучающих данных для весов и на валидационных данных для параметров архитектуры, таких как λ. В конечном итоге, дискретная архитектура получается путем выбора операции i с $i = arg\,max_i \lambda_i$для каждого слоя. Шин и др. [54] и Ахмед и Торресани [1] также используют градиентную оптимизацию архитектур нейронных сетей, однако они рассматривают только оптимизацию гиперпараметров слоев или паттернов связности, соответственно.

3.4 Стратегия оценки производительности

Стратегии поиска, обсуждаемые в разд. 3.3, направлены на поиск архитектуры нейронной сети A, которая максимизирует некоторую меру производительности, например, точность на неизвестных ранее данных. Чтобы направлять свой процесс поиска, эти стратегии должны оценить производительность архитектуры A, которую они рассматривают. Самый простой способ сделать это - обучить A на
обучающих данных и оценить ее производительность на валидационных данных. Однако обучение с нуля каждой архитектуры, которую необходимо оценить, часто требует вычислительных затрат порядка тысяч дней работы GPU для NAS [49, 50, 74, 75].

Чтобы уменьшить эту вычислительную нагрузку, производительность может быть оценена на основе меньших точностей фактической производительности после полного обучения (также обозначаются как прокси метрики). Такие более низкие точности включают более короткое время обучения [69, 75], обучение на

подмножестве данных [34], на изображениях с меньшим разрешением [14] или с меньшим количеством фильтров на слой [49, 75]. Хотя эти аппроксимации с низкой точностью снижают вычислительные затраты, они также вносят погрешность в оценку, поскольку производительность, как правило, будет недооценивается. Это может не вызывать проблем до тех пор, пока стратегия поиска только полагается на ранжирование различных архитектур, и относительное ранжирование остается стабильным. Однако последние результаты показывают, что это относительное ранжирование может резко измениться. когда разница между “дешевыми” приближениями и "полной" оценкой становится слишком большой[69], что говорит в пользу постепенного увеличения точности [24, 35].

Другой возможный способ оценки производительности архитектуры основан на следующем экстраполяция кривой обучения [5, 19, 32, 48, 61]. Домхан и др. [19] предлагают экстраполировать начальные кривые обучения и прекращать те, которые, по прогнозам, будут иметь низкую производительность, чтобы ускорить процесс поиска архитектуры. Бейкер и др [5], Клейн и др [32], Равал и Миккулаинен [48], Сверский и др. [61] также рассматривают гиперпараметры архитектуры
для предсказания того, какие частичные кривые обучения являются наиболее перспективными. Обучение суррогатной модели для прогнозирования производительности новых архитектур также предложена Лю и др. [37], которые не используют экстраполяцию кривых обучения, а поддерживают прогнозирование производительности на основе свойств архитектуры/ячейки и экстраполяцию на
архитектуры/ячейки большего размера, чем использовались во время обучения. Основная проблема для прогнозирования производительности нейронных архитектур заключается в том, что для того, чтобы ускорить процесс поиска, хорошие предсказания в относительно большом пространстве поиска должны быть сделаны
на основе относительно небольшого количества оценок.

Другой подход к ускорению оценки производительности заключается в инициализации весов новых архитектур на основе весов других архитектур, которые были обучены ранее. Один из способов достижения этой цели, называемый сетевыми морфизмами [64], позволяет модифицировать архитектуру, оставляя при этом функцию, представленную сетью, неизменной [10, 11, 21, 22]. Это позволяет последовательно увеличивать пропускную способность сетей и сохранять

высокую производительность, не требуя обучения с нуля. Продолжение обучение в течение нескольких эпох также позволяет использовать дополнительные возможности, вносимые морфизмами сети. Преимуществом этих подходов является то, что они позволяют искать пространства без присущей верхней границы на размер архитектуры [21]; С другой стороны, строгие сетевые морфизмы могут только увеличить размер архитектуры и, таким образом, это может привести к появлению чрезмерно сложных архитектур. Эта проблема может быть решена путем использования приближенных сетевых морфизмов, которые позволяют уменьшать архитектуру [22].

Одномоментный поиск архитектуры является еще одним перспективным подходом для ускорения оценки производительности, который рассматривает все архитектуры как различные подграфы суперграфа (модель one-shot) и распределяет веса между архитектурами, которые имеют общие ребра в этом суперграфе [6, 9, 39, 46, 52]. При данном подходе требуется обучать только веса одной one-shot модели (одним из различных способов), а архитектуры (которые являются просто подграфами модели one-shot) могут затем быть оценены без отдельного обучения, наследуя обученные веса от одномоментной модели. Это значительно ускоряет оценку производительности архитектур, поскольку обучение не требуется (только оценка производительности на проверочных данных). Этот подход обычно имеет большую погрешность, так как недооценивает фактическую производительность архитектур. Тем не менее, он позволяет надежно ранжировать архитектуры, поскольку оценочная производительность сильно коррелирует с фактической производительностью [6]. Различные одномоментные методы NAS различаются тем, как обучается модель one-shot: ENAS [46] обучает RNN контроллер, который выбирает архитектуры из пространства поиска и обучает модель one-shot на основе приблизительных градиентов, полученных с помощью REINFORCE. DARTS [39] оптимизирует все веса одномоментной модели совместно с непрерывной релаксацией пространства поиска, полученной путем размещения смеси операций-кандидатов на каждом ребре одномоментной модели. Бендер и др. [6] обучают модель one-shot только один раз и показывают, что этого достаточно при стохастической деактивации частей этой модели во время обучения с помощью метода отбрасывания путей. В то время как ENAS и DARTS оптимизируют распределение по архитектурам во время обучения,

подход Бендера и других [6] можно рассматривать как использование фиксированного распределения. Высокая производительность, достигаемая подходом Бендера и других [6], указывает на то, что комбинация разделения веса и фиксированного (тщательно выбранного) распределения может (возможно, неожиданно) быть единственным необходимым ингредиентами для одномоментного NAS. С этими подходами связано мета-обучение гиперсетей, которое генерирует веса для новых архитектур и, таким образом, требует обучения только гиперсетей, но не самих архитектур [9]. Основное отличие здесь в том, что веса не являются строго общими, а генерируются общей гиперсетью (условно для выбранной архитектуры).

Общее ограничение одномоментного NAS заключается в том, что суперграф, определенный a-priori, ограничивает пространство поиска его подграфами. Более того, подходы, требующие, чтобы весь суперграф находился в памяти GPU во время поиска архитектуры, будут ограничены относительно небольшими суперграфами и пространствами поиска соответственно, и поэтому они обычно используются в сочетании с пространствами поиска на основе ячеек. Хотя подходы основанные на распределении веса, значительно сократили вычислительные ресурсы, требуемые для NAS (с тысяч до нескольких дней работы GPU), в настоящее время не совсем ясно, какие погрешности они вносят в поиск, если распределение выборки архитектур оптимизируется вместе с моделью one-shot. Например, начальное смещение в исследовании определенных частей пространства поиска больше, чем других, может привести к тому, что весовые коэффициенты одномоментной модели будут лучше адаптированы для этих архитектур, что в свою очередь усилит смещение поиска в эти части пространства поиска. Это может привести к преждевременной сходимости NAS и может быть одним из преимуществ фиксированного распределения выборки, используемого Бендером и другими [6]. В целом, более систематический анализ смещений, вносимых различными оценщиками производительности, был бы желательным направлением будущей работы.

3.5 Будущие направления

В этом разделе мы обсудим несколько текущих и будущих направлений исследований в области NAS. Большинство существующих работ посвящено NAS для классификации изображений. С одной стороны, это сложный эталон, так как много ручного проектирования было посвящено поиску архитектур, которые хорошо работают в этой области и которых не так легко превзойти NAS. С другой стороны, относительно легко определить хорошо подходящее пространство поиска, используя знания, полученные в результате ручной разработки. Это, в свою очередь, делает маловероятным, что NAS найдет архитектуры, которые существенно превосходят существующие, поскольку найденные архитектуры не могут принципиально отличаться друг от друга. Мы
таким образом считаем важным выйти за рамки проблем классификации изображений, применяя NAS к менее изученным областям. Заметными первыми шагами в этом направлении являются применение NAS к моделированию языка [74], моделированию музыки [48], восстановлению изображений [58] и сжатию сетей [3]; приложения к обучению с подкреплением, генеративные состязательные сети, семантическая сегментация или объединение сенсоров могут быть дальнейшими перспективными направлениями.

Альтернативным направлением является разработка методов NAS для многозадачных проблем [36, 42] и для многоцелевых задач [20, 22, 73], в которых показатели эффективности использования ресурсов используются в качестве целей наряду с эффективностью прогнозирования на невидимых данных. Аналогичным образом, было бы интересно расширить подходы RL/bandit, такие, как те, что обсуждались в разд. 3.3, для обучения политик, которые обусловлены состоянием, которое кодирует свойства задачи/требования к ресурсам (т.е. превращая постановку в контекстного бандита). Аналогичного направления придерживались Рамачандран и Ле [47] в расширении одномоментного NAS для генерации различных архитектур в зависимости от задачи или экземпляра "на лету". Более того, применение NAS для поиска архитектур, которые более устойчивы к неблагоприятным примерам [17] - интригующее недавнее направление.

С этим связаны исследования по определению более общих и гибких пространств поиска. Например, хотя пространство поиска на основе клеток обеспечивает высокую переносимость между различными задачами классификации изображений, оно в значительной степени основано на

человеческом опыте в области классификации изображений и не может быть легко обобщено на другие области, где жестко закодированная иерархическая структура (повторение одних и тех же ячеек несколько раз в цепной структуре) не применима (например, семантическая сегментация или обнаружение объектов). Пространство поиска, которое позволяет представлять и идентифицировать более общую иерархическую структуру таким образом, сделает NAS более широко применимым, см. первую работу Лю и др. в этом направлении. Более того, общие пространства поиска также основаны на заранее определенных строительных блоках, таких как различные виды сверток и объединение, но не позволяют идентифицировать новые строительные блоки на этом уровне; выход за рамки этого ограничения может существенно увеличить возможности NAS.

Сравнение различных методов для NAS осложняется тем, что измерения производительности архитектуры зависят от многих факторов, помимо самой архитектуры. Хотя большинство авторов сообщают о результатах на наборе данных CIFAR-10, эксперименты часто различаются в отношении пространства поиска, вычислительных возможностей, данных дополнения, процедур обучения, регуляризации и других факторов. Например, для CIFAR-10 производительность существенно повышается при использовании косинусного отжига графика скорости обучения [40], дополнения данных методом CutOut [18], MixUp [70] или комбинацией факторов [16], и регуляризации с помощью регуляризации Shake-Shake [26] или запланированным drop-path [75]. Поэтому можно предположить, что улучшения этих компонентов оказывают большее влияние на заявленные показатели производительности, чем лучшие архитектуры, найденные NAS. Таким образом, мы считаем определение общих эталонов критическим для справедливого сравнения различных методов NAS. Первым шагом в этом направлении является определение эталона для совместного поиска архитектуры и поиска гиперпараметров для полносвязной нейронной сети с двумя скрытыми слоями [33].В этом эталоне необходимо оптимизировать девять дискретных гиперпараметров, которые управляют как архитектурой, так и оптимизацией/регуляризацией. Все 62208 возможных комбинаций гиперпараметров были предварительно оценены, так что различные методы могут быть сравнены с небольшими вычислительными ресурсами. Тем не менее, пространство поиска все еще очень простое по сравнению с пространствами, используемыми большинством методов NAS. Было бы также интересно оценить методы

NAS не изолированно, а как часть полной системы AutoML с открытым исходным кодом, где гиперпараметры [41, 50, 69] и конвейер дополнения данных [16] оптимизируются вместе с NAS.

Хотя NAS достигла впечатляющей производительности, до сих пор она не дает представления о том, почему конкретные архитектуры работают хорошо и насколько похожи архитектуры, полученные в независимых прогонах. Желательно выявить общие мотивы, понять, почему эти мотивы важны для высокой производительности, и выяснить, обобщаются ли эти мотивы на различные проблемы.