Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра суперкомпьютеров и квантовой информатики

Никуленков Михаил Романович

Оптимизация эволюционного метода поиска архитектур глубоких нейросетей с использованием сжатия моделей

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Научный руководитель:
к.ф.-м.н., доцент
Н.Н. Попова

Москва, 2022

Аннотация

Число задач, требующих нейросетевой обработки, неуклонно растет . Увеличиваются объемы обрабатываемых данных. Вследствие этого усложняются архитектуры нейросетей, что влечет за собой большие временные затраты на их построение и обучение.

Работа посвящена оптимизации методов автоматического построения архитектур глубоких нейросетей. Один из способов решения задачи определения хорошей архитектуры – использование эволюционных методов поиска архитектур, предполагающих обучение популяции нейронных сетей с целью оценки их качества. Одним из таких методов является CoDeepNEAT: благодаря способу представления генотипов, выделяющему отдельные популяции для слоев сети и схем соединения слоев, он дает возможность выполнять поиск архитектур глубоких нейросетей. Для оптимизации эволюционного поиска возможно использование MorphNet – способа автоматической оптимизации нейронных сетей, основанного на изменении размеров слоев имеющейся архитектуры. Предлагается использовать MorphNet для оптимизации популяций нейросетей, возникающих на этапах эволюционного метода CoDeepNEAT, путем их сжатия, тем самым ускоряя поиск за счет сокращения времени, необходимого на оценку качества сетей.

Предложенный подход реализован и исследован на задаче классификации изображений на кластере «Polus».

Исследования предложенного подхода показали, что разработанный метод позволяет добиться ускорения по сравнению с методом CoDeepNEAT без существенных потерь в точности генерируемых сетей.

Содержание

1 Введение

Задачи глубокого обучения требуют подходящей для них нейросетевой архитектуры. Нахождение хорошей архитектуры может быть непростой задачей. В последние годы активно развиваются методы автоматического построения нейросетей, особенно глубоких нейросетей. В этом случае временные затраты возрастают еще больше.

Для автоматизации процесса построения архитектур можно использовать нейроэволюционные методы. Они переносят задачу нахождения наилучшей архитектуры в область естественного отбора: нейронные сети представляются в виде генотипов, конкурируют между собой, скрещиваются и мутируют. Таким образом находится наиболее подходящая для решения задачи архитектура.

Актуальным является разработка методов, позволяющих сократить временные расходы на построение нейросетей. Методам поиска архитектур может потребоваться много времени для решения задачи.

Эволюционный метод поиска архитектур предполагает обучение составляющих популяцию нейронных сетей с целью оценки их приспособленности к решению задачи. MorphNet — способ автоматической оптимизации нейронных сетей, основанный на изменении размеров слоев архитектуры с помощью регуляризации, вызывающей разреженность. Предлагается использовать MorphNet для оптимизации популяций нейросетей, возникающих на этапах нейроэволюции. Сжатие архитектуры сетей, возникающих по ходу поиска, может помочь ускорить процесс. Подход MorphNet позволяет оптимизировать нейронную сеть по целевому ресурсу, в качестве которого выступает количество операций с плавающей точкой. Применение соответствующего регуляризатора способно ускорить обучение оптимизированной сети.

1.1 Цель работы

Целью работы является оптимизация методов автоматического построения архитектур глубоких нейросетей. В работе рассматривается применение метода MorphNet [1] для оптимизации построения архитектур глубоких сетей с помощью нейроэволюционного метода CoDeepNEAT [2] применительно к задаче классификации изображений с помощью сверточных нейронных сетей. В качестве критерия оптимизации выступает время работы нейроэволюционного метода.

1.2 Постановка задач

Задачами магистерской диссертации являются:

1. Обзор существующих методов поиска и оптимизации архитектур нейронных сетей.
2. Исследование стратегий применения регуляризации в процессе эволюционного отбора архитектур с целью выбора наиболее эффективной стратегии.
3. Разработка эволюционного метода с использованием регуляризационного сжатия.
4. Программная реализация предложенного метода.
5. Экспериментальное исследование разработанного метода.

1.3 Основные определения

Основные определения и сокращения, используемые в работе:

- Модель нейронной сети – стуктурное и конфигурационное описание нейронной сети (ее топология, набор гиперпараметров, весовых коэффициентов).
- Архитектура нейронной сети – топологическая структура нейросетевой модели (схема соединения нейронов, слоев).
- Процесс нейроэволюции – процесс оптимизации архитектуры нейронных сетей с помощью эволюционных алгоритмов.
- Сжатие архитектуры нейронной сети – уменьшение количества нейронов в слоях глубокой нейронной сети, описываемой архитектурой.
- Пространство поиска архитектур – пространство всех возможных конфигураций архитектур, которые могут быть найдены в процессе поиска архитектур.
- FLOPN (floating points operations number) – количество операций с плавающей точкой, необходимое для вычисления результата по входным данным нейронной сетью.

1.4 Структура работы

В разделе 2 дан обзор основных подходов к оптимизации архитектур нейронных сетей. В разделе 3 описаны способы применения регуляризационной оптимизации в процессе поиска архитектур с помощью генетического алгоритма, дано сравнение предложенных способов и вывод об их применимости. В разделе 4 дано доволнительное описание предложенного метода. В разделе 5 описана выполненная программная реализация. В разделе 6 приведено экспериментальное исследование предложенного метода. В разделе 7 формулируются основные результаты работы.

2 Обзор основных подходов к оптимизации архитектур нейронных сетей

Под оптимизацией архитектур понимается задача определения оптимальной с точки зрения установленного критерия архитектуры нейронной сети, предназначенной для решения задачи. В качестве таких критериев могут выступать, например: максимальная точность; минимальное значение FLOPN при сохранении точности исходной архитектуры.

Можно выделить два общих подхода к оптимизации архитектур:

1. Автоматическое построение архитектуры
   (a) Цель: найти оптимальную архитектуру для решения задачи.
   (b) Исходные данные: пространство поиска.
2. Оптимизация существующей архитектуры.
   (a) Цель: оптимизация заданной архитектуры нейронной сети с точки зрения необходимого критерия.
   (b) Исходные данные: архитектура нейронной сети.

Рассмотрим популярные методы из каждого подхода.

2.1 Автоматическое построение архитектуры

2.1.1 Grid Search

Grid Search – простейший метод поиска архитектур. Метод берет свое начало в задаче оптимизации гиперпараметров нейронной сети. В нем пространство возможных комбинаций компонент сети (иными словами – пространство возможных архитектур) представляется в виде точек на дискретной сетке. С помощью кросс-валидации исследуется производительность каждой из комбинаций. Комбинация с наилучшей производительностью представляет собой оптимальную архитектуру.

Метод grid search охватывает все комбинации, поэтому он способен найти наилучшую точку в заданном пространстве возможных архитектур. Однако, метод имеет значительный недостаток – он очень медленный [4]. Проверка каждой комбинации требует значительных временных затрат.

На на рис. 1 изображен простейший пример сетки в методе Grid Search: рассматривается нейронная сеть с двумя сверточными слоями с фиксированными размерами ядер. Пространство комбинаций определяется количеством фильтров на слой.

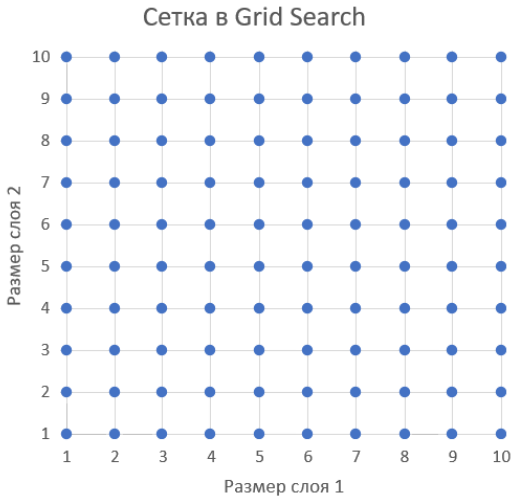


Рис. 1: Пример сетки в Grid Search для пространства из двух компонент.

2.1.2 Random Search

Метод Random Search берет свое начало в Grid Search, но отличается тем, что вместо анализа всех точек на сетке, он проверяет только случайно выбранное подмножество этих точек. Чем меньше подмножество, тем быстрее метод дает результат. Однако, с уменьшением рассматриваемого множества точность результата также понижается. Рассматривая больше сетей, метод приближается к Grid Search, но становится медленнее [4].

На рис. 2 изображен простейший пример сетки в методе Random Search: рассматривается нейронная сеть с двумя сверточными слоями с фиксированными размерами ядер. Пространство комбинаций определяется количеством фильтров на слой.

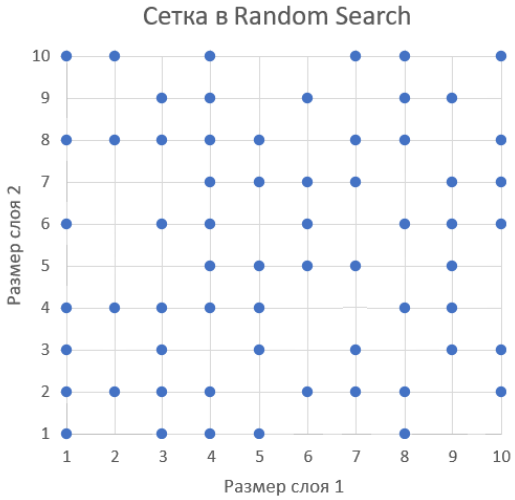


Рис. 2: Пример сетки в Random Search для пространства из двух компонент.

2.1.3 Байесовская оптимизация

Байесовская оптимизация, имеющая историю применения к задаче определения оптимальных гиперпараметров, также применима к задаче поиска архитектур [5]. В контексте поиска архитектур целевая функция ставит соответствие между архитектурой и ее точностью после обучения в течении нескольких эпох.

Пусть определены:

1. Пространство поиска A.
2. Целевая функция $f(a) \to [0, 1]$.

Цель: найти такую архитектуру $a \in A$, при которой $f(a)$ максимальна.

Назовем "суррогатом" целевой функции вероятностную модель $P(score|architecture)$. На каждой итерации байесовская оптимизация использует суррогат для моделирования целевой функции на основе ранее рассмотренных архитектур и значений их точности. С помощью суррогата выбирается следующая архитектура для оценки путем максимизации ожидаемого значения точности.

Баесовская оптимизация представляет собой итеративный процесс, которому предшестует выбор подлежащей оценке группы архитектур из A, выступающей в качестве исходных данных для суррогата. Структура итерации баесовской оптимизации:

1. Найти лучшую сеть $a \in A$ с точки зрения суррогата.
2. Вычислить $f(a)$
3. Обновить параметры суррогата с учетом новых результатов.

Определение следующей архитектуры для оценки путем максимизации предсказания суррогата и оценка предсказанной архитектуры часто требуют значительных временных затрат и усложняют применение байесовской оптимизации в контексте поиска архитектур.

2.1.4 Обучение с подкреплением

Обучение с подкреплением или reinforcement learning может лежать в основе стратегии поиска архитектур. Например, в известной работе [6] обучение с подкреплением было применено к задаче поиска архитектур на наборе данных CIFAR-10. В результате была создана архитектура, по точности конкурирующая с лучшей архитектурой, разработанной вручную. Точность созданной архитектуры составила 96.35%, на 0.09% превзойдя ручную архитектуру.

Идея заключается постепенном в обучении сети-контроллера, которая должна генерировать архитектуры. Рассмотрим схему итерации в рассматриваемом подходе к поиску архитектур:

1. Сгенерировать архитектуру с помощью контроллера.
2. Обучать сгенерированную архитектуру в течении нескольких эпох.
3. Оценить производительность сгенерированной архитектуры.
4. Обновить параметры контроллера в соотстветствии с полученной оценкой.

Таким образом, метод, использующий описанный подход, должен отвечать на следующие вопросы:

1. Как спроектировать сеть-контроллер?
2. Как должен выглядеть процесс оптимизации сети-контроллера?

В контексте поиска архитектур нейронных сетей в качестве сетей-контроллеров могут служить рекуррентные сети, так как они создают последовательность выходных значений. В работе [6] эти последовательности предлагается декодировать для создания архитектуры нейронной сети, которая затем подлежит обучению и оценке.

Агентом будем называть алгоритм, который умеет анализировать состояние среды и совершать в ней действия. Функцией вероятности оптимальности действия будем называть функцию, которая показывает насколько то или иное действие оптимально в текущем состоянии среды. Наградой будем называть ответ среды на действия агента. Рассмотрим процесс обучения с подкреплением, изображенный на рис. 3:

1. Агент предпринимает действия в соответствии с функцией вероятности оптимальности действия.
2. Среда подвергается манипуляциям, генерируется новое состояние среды.
3. На основании награды агент обновляет функцию оптимальности действия.
4. Повторить с новым состоянием среды и обновленной функцией оптимальности действия.

Таким образом, в качестве агента в задаче поиска архитектур выступает сеть-контроллер, оперирующая в среде-архитектуре. Награда определяется производительностью сгенерированной сети.

Одним из способов обновления функцией вероятности оптимальности действия является REINFORCE ([6], [7]) – это градиентный алгоритм, который использовался в задаче поиска архитектур для оптимизации контроллеров [6]. Градиент функции вероятности оптимальности действия REINFORCE оптимизирует целевую функцию на

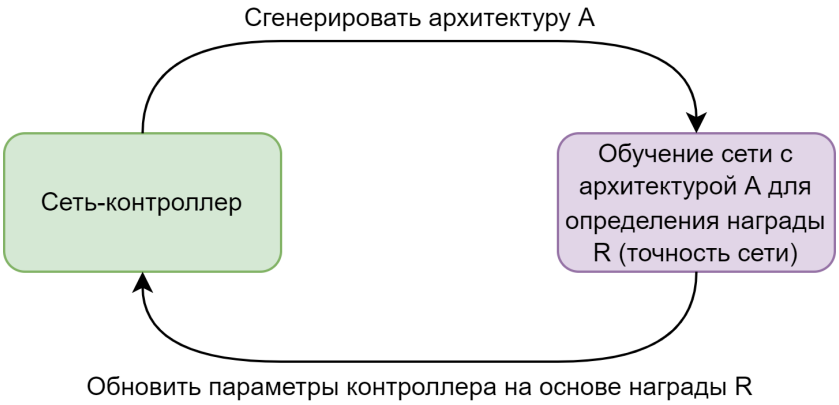


Рис. 3: Процесс поиска архитектуры с помощью контроллера.

основании ответа среды таким образом, чтобы максимизировать общее вознаграждение агента.

2.1.5 Нейроэволюция

Процесс поиска архитектур может выполняться с использованием генетических алгоритмов. Генетические алгоритмы – это класс эволюционнных алгоритмов. Эволюционные алгоритмы стремятся воспроизвести процесс эволюции популяций животных, наблюдаемый в природе, применяя его для оптимизации функций. Популяция в генетических алгоритмах определяется наборами генов (генотипами), зачастую выраженными в виде массивов чисел. Пусть имеется исходная популяция решений. Общий процесс генетического алгоритма выглядит следующим образом:

1. Оценка приспособленности каждого решения. Оценка приспособленности определяется функцией приспособленности, которая в свою очередь зависит от задачи.
2. Отбор решений, наиболее приспособленных для создания потомства. Существуют различные схемы отбора: например, ранжированный отбор, когда для создания потомства выбираются наиболее приспособленные особи; ранжированный выбор с элементом случайности, когда с вероятностью к созданию потомства могут быть допущены менее приспособленные особи.
3. Создание потомства и применение случайных мутаций. Причем потомство не обязательно должно иметь двух родителей – гены можно смешивать и сопоставлять различными способами. Будущее поколение также подвергается случайным мутациям, вносящим изменения в гены. Вероятность мутаций должна быть определена.
4. Повтор процесса с новым поколением решений.

CoDeepNEAT [2] (Cooperative Deep Neuroevolution of Augmenting Topologies) – эволюционный метод оптимизации топологии, компонент и гиперпараметров глубинных

сетей.

Этот алгоритм относится к семейству нейроэволюционных алгоритмов поиска архитектур. Свое идейное начало он берет в методах: 1) NEAT [3] (Neuroevolution of Augmenting Topologies), предлагающим способ кодирования топологии сети в генотип для возможности скрещивания сетей на основании их приспособленности (точности) и применения мутаций; также важной особенностью этого алгоритма является то, что в нем предлагается мера разбиения особей из популяции на виды (группы) на основании схожести их топологий (особи конкурируют только внутри своей группы); 2) DeepNEAT [2] является обобщением идеи NEAT на глубинные сети; главное отличие от NEAT в том, что звенья топологии – это слои в глубинной сети (в то время как в NEAT это отдельные нейроны).

Подходы NEAT и DeepNEAT объединяет то, что в обоих из них генотип кодирует сеть целиком. CoDeepNEAT является следующим шагом в развитии идеи DeepNEAT. В CoDeepNEAT выделяются популяции "модулей" (иначе говоря, слоев сети) и "шаблонов" (схем соединения слоев; звено шаблона указывает на вид модулей, а не на конкретный модуль). Эти две популяции эволюционируют раздельно. На рис. 4 продемонстрован процесс составления сети из популяций модулей и шаблонов.

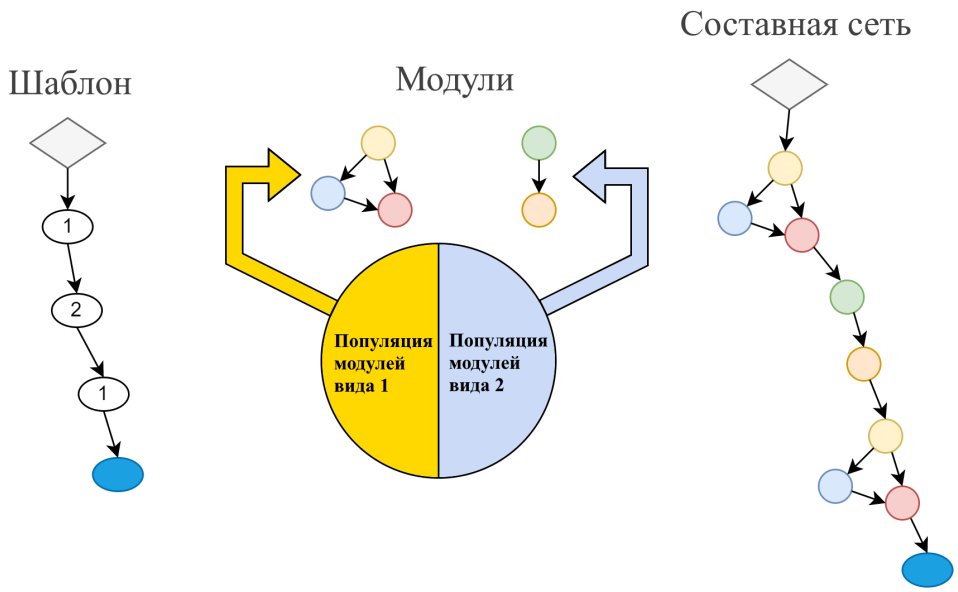


Рис. 4: Составление сети из модулей и шаблона.

Популяция сетей строится на основании популяций модулей и шаблонов, которые в свою очередь внутри себя разделены на виды. При построении сети по шаблону генотип из вида, на который указывает звено, выбирается случайно. Если разные звенья одной схемы указывают на один вид модулей, используется один и тот же случайный генотип

из вида.

Оценивается приспособленность особи (составной сети), схемы и модули снабжаются информацией о приспособленности составных сетей, о том, где они были использованы. На основании этой информации строятся их собственные меры приспособленности для конкуренции.

Отличительной особенностью семейства NEAT-подобных методов является то, что в них используется разделение популяции сетей на группы. В работе [3] предлагается мера схожести архитетур, на основании которой происходит разделение популяции на группы похожих архитектур, называемые видами. Конкуренция между сетями осуществляется только внутри своего вида. Эта особенность вносит разнообразие в общую популяцию сетей, дает возможность новым, еще не успевшим себя показать архитектурам развиться и не быть уничтоженными на этапе отбора.

На рис. 5 проиллюстрирован схематичный пример разделения на виды и отбора в популяции из девяти сетей. Каждой сети соответствует численное значение в пределах $0 - 1000$, определяющее ее степень приспособленности. Отбор осуществляется только внутри вида.

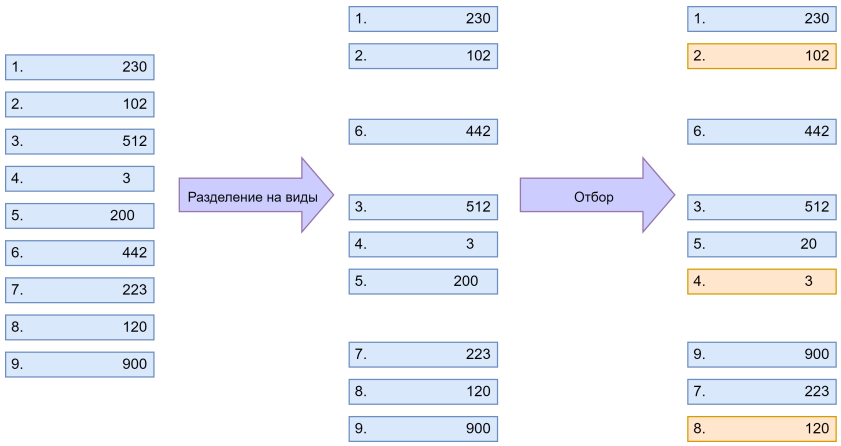


Рис. 5: Пример разделения на виды и последующего отбора популяции из девяти сетей. Сети, помеченные оранжевым цветом, не прошли отбор внутри своих видов.

2.2 Оптимизация существующей архитектуры

2.2.1 Network morphism

Байесовскую оптимизацию можно использовать также для оптимизации существующей архитектуры. Рассмотрим пример. В работе [8] предлагается подход к оптимизации архитектуры нейронной сети с помощью использования эволюционных алгоритмов, байесовской оптимизации и поиска восхождением к вершине (hill-climbing). Стратегия

восхождения работает путем итеративного преобразования исходного решения для создания новых решений, пока не будет найдено наилучшее решение. На рис. 6 изображена схема описанной стратегии.

Таким образом, метод на каждой итерации с помощью байесовской оптимизации, рассмотренной в разделе «Байесовская оптимизация», генерирует набор измененных форм решения, полученного на предыдущей итерации. Лучшая модификация выбирается в качестве нового решения, затем процесс повторяется.

Семейство методов, использующих баесовскую оптимизацию в задачах поиска/оптимизации архитектуры имеют один общий недостаток - максимизация предсказания суррогата и оценка предсказанной архитектуры требуют значительных временных затрат, усложняя такое применение байесовской оптимизации.

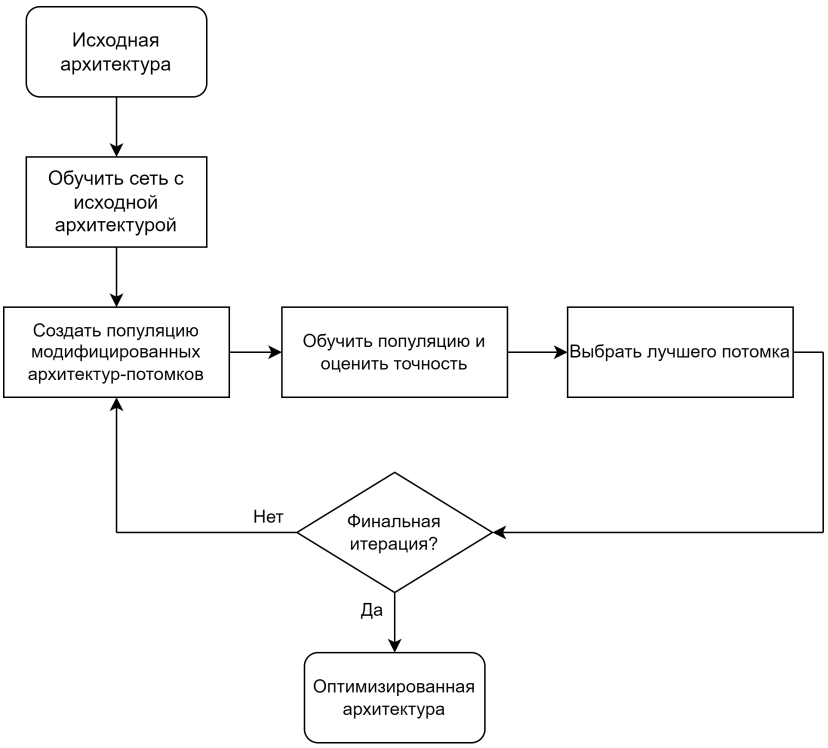


Рис. 6: Схема процесса байесовской оптимизации архитектуры.

2.2.2 Group Lasso

Одним из способов оптимизации архитектуры является использование регуляризации. Регуляризация может вызывать разреженность в весах нейронной сети. Если все веса, которые соответствуют выходам одного нейрона, близки к нулю, то можно этот нейрон вычеркнуть из архитектуры, изменив ее. Таким образом можно сжать сеть, сократив количество ее параметров и FLOPN.

Добиться группового прилижения значения весов можно с помощью использования регуляризатора Group Lasso [9], выраженного в виде слагаемого, добавляющегося к функции потерь: $L(\bar{\theta}, \bar{x}, \bar{y}) + \lambda R(\bar{\theta})$, где L – функция потерь, R – регуляризатор, $\bar{\theta}$ – набор весовых коэффициентов сети, $\bar{x}, \bar{y}$ – набор обучающих данных, λ – сила регуляризации. В регуляризаторе Group Lasso веса, соотвествующие одному нейрону, помещаются в одну Lasso группу g и коллективно сдвигаются к нулю:

$$R(\bar{\theta}) = \sum_{g \in G} \sqrt{|g|} \|x\|_2.$$

Проблемой при использовании регуляризации для поиска компактной архитектуры является определение оптимального набора гиперпараметров. Одним из наиболее важных гиперпараметров является сила регуляризации λ. Выбор правильной силы регуляризации является ключевым моментом при обучении. При слишком большой силе модель вообще не будет обучаться, при слишком низкой разреженность в весах не возникнет и не удастся узнать, какие нейроны следует отбросить.

Методы, применяющие регуляризацию для изменения архитектуры, имеют следующие особенности:

1. Весовые коэффициенты исходной сети бесполезны для новой сети.
2. Может потребоваться поиск оптимальной силы регуляризации.
3. Возможно снижение точности за счет сжатия исходной сети.

2.2.3 MorphNet

MorphNet [1] – способ автоматической оптимизации архитектуры нейронных сетей. Оптимизация достигается изменением размеров слоев сети. Оптимизация учитывает целевой ресурс (прим. FLOPN, количество параметров). На рис. 7 продемонстрирован пример оптимизации сверточной нейронной сети методом MorphNet.

MorphNet поочередно сжимает и расширяет архитектуру:

1. На этапе сжатия применяется регуляризатор, способствующий возникновению разреженности. Регуляризатор использует идею группировки весов, рассмотренную в разделе "Group Lasso". На этапе расширения количество нейронов во всех слоях равномерно увеличивается в $\omega \geq 1$ раз.
2. Этап расширения позволяет перераспределить нагрузку внутри сети, придав более полезным слоям пропорционально больший размер. Этап расширения применяется по усмотрению – MorphNet может быть использован в том числе только для сжатия сети.

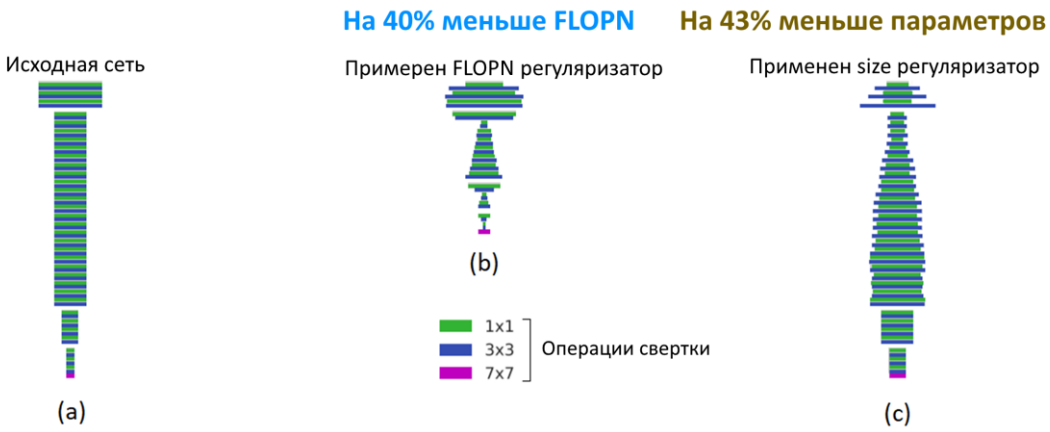


Рис. 7: Пример оптимизации сверточной нейронной сети методом MorphNet в работе [1]. Полученные сети дают одинаковую производительность с исходной сетью, но используют меньше целевого ресурса.

Пусть $O_{1:M}$- архитектура сверточной сети с кол-вом сверточных слоев M. На этапе сжатия ставится цель минимизировать функцию потерь $L(\theta)$ с применением регуляризации:

$$\min_{\theta}(L(\theta) + \lambda F(O_{1:M})),$$

где $F(O_{1:M}) \leq \zeta$ - регуляризатор, содержащий оценку целевого ресурса, на который накладывается ограничение ζ. Авторами MorphNet предложены [1] различные регуляризаторы, построенные на основе как оценке FLOPN модели, так и на количестве ее параметров.

В работе [10] приводится сравнение различных регуляризационных техник сжатия архитектуры. В работе делается вывод, что из всех рассмотренных методов MorphNet дает наибольшее сокращение FLOPN при наименьшем снижении точности сжатой сети.

2.3 Выводы

Нейроэволюция и обучение с подкреплением – активно развивающиеся и наиболее популярные способы поиска архитектур. Рассмотрение различных подходов к оптимизации архитектуры нейронной сети указывает на возможное их применение в комбинации с методами поиска архитектур. Логичным такое применение представляется именно в генетических алгоритмах – если вносить изменение в популяцию, можно попробовать добиться ускорения процесса поиска за счет дополнительной модификации популяции на еще на этапе ее оценки.

Из рассмотренных методов оптимизации архитектуры подходящими для такого применения представляются регуляризацонные методы: баесовские методы требуют очень больших временных затрат, что противоречит задаче ускорения поиска архитектур.

Среди регуляризационных методов выделяется MorphNet как дающий наибольшее снижение FLOPN архитектуры при наименьшем снижении ее точности [10].

3 Способы применения регуляризационной оптимизации в CoDeepNEAT

3.1 Применимость в рассматриваемой задаче

Возможна оптимизация нейросетевых моделей, возникающих во время эволюционного процесса в методе CoDeepNEAT, с использованием регуляризатора MorphNet. Сжатие сетей, генерируемых эволюционным алгоритмом, позволяет ускорить их обучение, необходимое для оценки точности.

3.2 Особенности CoDeepNEAT

Подход CoDeepNEAT хорошо справляется с задачей поиска архитектур глубоких нейросетей, однако, он имеет нетривиальную внутреннюю структуру: в отличии от большинства нейроэволюционных методов, популяция сетей в нем определяется двумя независимыми друг от друга популяциями модулей и шаблонов. Элементы шаблонов указывают на вид модулей. Используется случайный модуль из вида.

Схемы использующие один модуль могут перераспределить нагрузку внутри себя по разному. Соответственно после этапа обучения это будут уже разные модули. Возникают вопросы. Как оценить приспособленность исходного модуля? Как выбрать новый размер исходного модуля?

Также возможна ситуация, в которой несколько особей, реализующих в популяции некоторый шаблон, в результате оптимизации избавились от части своих слоев. В данном случае старому шаблону будут соответствовать несколько новых, уменьшенных шаблонов. Как выбрать новую структуру шаблона? Для ответа на эти вопросы необходимо сначала наиболее общим образом определить стратегии применения разреживающей регуляризации в разделенных популяциях:

1. Не менять исходные модули, и, как следствие, не менять шаблоны. Каждая схема копирует к себе исходный модуль и изменяет его под себя. Исходному модулю сообщается информация об эффективности его “производных” модулей.
2. Менять исходный модуль, вносить изменения в шаблоны.

3.3 Эффект коэффициента регуляризации

Определяющий параметр регуляризационного сжатия нейросетевых моделей – коэффициент регуляризации. От выбора коэффициента зависит сила сжатия, и, соответ-

ственно, точность полученной в результате архитектуры.

Использование регуляризации для сжатия моделей имеет свои особенности, отличающиеся от стандартного применения регуляризации для избежания переобучения. Данное применение регуляризации ставит себе целью сокращение использования целевого ресурса (FLOPN) за счет деградации точности модели. В идеальном случае стоит выполнить поиск коэффициента регуляризации, который удовлетворял бы требованиям по необходимому сокращению целевого ресурса и допустимой деградации точности.

Методы регуляризационной оптимизации не подразумевают использование сети сразу после оптимизации: сеть придется обучать заново. Авторы подхода подчеркивают [1], что значения весов у "оставшихся в живых" нейронов, полученные в ходе оптимизации, будут бесполезны после сжатия слоев сети. Применение регуляризации для ускорения нейроэволюционного процесса накладывает ограничения на выбор коэффициента: задача сокращения времени работы не позволяет рассмотреть множество коэффициентов, так как оценка коэффициента требует время на сжатие и последующее полноценное обучение сети. В связи с этим предлагается выбрать некоторую точку отсчета, установить зависимость между архитектурой и используемой для её сжатия силой регуляризации.

Авторы подхода MorphNet предлагают выполнять поиск коэффициента регуляризации в пределах нескольких порядков по логарифмической шкале вокруг значения $\frac{1}{initial\_cost}$, где $initial\_cost$ – значение FLOPN исходной архитетуры. Для всех сжимаемых моделей предлагается установить единое положение на логарифмической шкале. Таким образом, сила регуляризации будет однозначно определяться значением целевого ресурса модели.

3.4 Эволюция сжатых сетей

Опишем наивный подход, не подразумевающий использование информации о том, каким образом была изменена структура сети в результате применения MorphNet.

Идея подхода заключается в предварительном сжатии всех сетей, подлежащих оценке в рамках нейроэволюционного алгоритма. В данном подходе генотипы модулей и шаблонов не затрагиваются, но их конкуренция осуществляется с помощью информации об эффективности оптимизированных составных сетей. Эволюционный метод нацеливается на отбор сетей, которые могут быть оптимизированы лучшим образом.

Опишем некоторые необходимые для понимания подхода процедуры.
Procedure $optimize(individual)$
Вход: сеть ($individual$)

Выход: оптимизированная с точки зрения использовании целевого ресурса сеть

Procedure $measureFitness(individual)$
Вход: сеть для оценки приспособленности ($individual$).
Выход: значение приспособленности особи.

Procedure $applyGeneticOperators(fitnesses, individuals, modules, blueprints)$
Вход: значения приспособленности особей из популяции ($fitnesses$), генотипы сетей из популяции ($individual$), популяции модулей и шаблонов ($modules$, $blueprints$).
Выход: популяции модулей и шаблонов, из которых может быть составлено следующее поколение особей.

Данный способ применения MorphNet на этапе эволюционного алгоритма можно описать следующим образом:
Procedure $makeEvolutionStep(individuals, modules, blueprints)$
Вход: популяция сетей ($individuals$), популяции модулей и шаблонов ($modules$, $blueprints$).
Выход: популяции модулей и шаблонов, из которых может быть составлено следующее поколение особей.

Общий алгоритм:

Algorithm 1: *makeEvolutionStep*

```
Data: individuals, modules, blueprints
for all individual ∈ individuals do
    optimize(individual);
    fitnesses[individual] ← measureFitness(individual);
end
applyGeneticOperators(fitnesses, individuals, modules, blueprints);
```

3.5 Наследование измененных генотипов

Ранее был рассмотрен подход к использованию регуляризационной оптимизации, не подразумевающий какое-либо изменение исходных популяций модулей и шаблонов. Такая стратегия изменяет исходную задачу нейроэволюции, перенося ее в область отбора архитектур, которые могут быть наилучшим образом оптимизированы с помощью регуляризации. К тому же, такой подход накладывает вынужденное ограничение – чтобы оставаться в рамках измененной задачи, необходимо выполнять оптимизацию всех сетей на каждом поколении генетического алгоритма. Это продиктовано тем, что в общем случае не обязательно будет наблюдаться соотвествие между мерами приспособленности оптимизированных и неоптимизированных сетей. Учет структурных изменений модулей и шаблонов поможет вернуться к исходной задаче нейроэволюции, и сделает подход более гибким, позволив выполнять сжимающую оптимизацию, например, раз в

несколько поколений генетического алгоритма.

Для определения способов изменения модулей и шаблонов необходимо учитывать несколько факторов:

1. Один и тот же модуль может использоваться в разных особях из популяции. Слои, составляющие этот модуль, могут быть оптимизованы сетями по-разному.
2. Особь может использовать один и тот же модуль в разных местах внутри своей структуры. Соответственно, после оптимизации одна сеть может содержать несколько по-разному измененных экземпляров одного и того же модуля.
3. В результате оптимизации группа слоев может быть полностью удалена, что может быть отражено в генотипе шаблона. При этом несколько сетей из популяции могут иметь один и тот же шаблон, измененный по-разному.

Имеем список модулей, использованных в текущей популяции. Каждому элементу этого списка соответствует набор особей, в которых этот модуль использовался, и набор соотвествующих значений приспособленности особей.

Упорядочим особей в порядке убывания их приспособленности. Выберем для всех модулей, использованных в первой особи, их локальную версию для этой особи в качестве окончательной. Если особь содержит несколько версий одного и того же модуля, следует выбрать наиболее затратную с точки зрения использования целевого ресурса (FLOPN) версию модуля. Пометим эти модули. Проделаем то же самое со второй особью, пропуская ранее помеченные модули. И так далее со всеми оставшимися особями.

Опишем некоторые необходимые процедуры.
Procedure $sortByFittness(info_{individuals}, fitness)$
Вход: массив с информацией о состояниях модулей в особях ($info_{individuals}$), значения приспособленности особей из популяции ($fitnesses$)
Выход: отсортированный по убыванию значения приспособленности массив $info_{individuals}$.

Procedure $mostCostyState(states)$
Вход: набор состояний модуля ($states$).
Выход: наиболее большое с точки зрения использования целевого ресурса (FLOPN) состояние модуля.

Алгоритм изменения модулей:

Algorithm 2: *applyModuleOptimization*

Data: $fitnesses$, $info_{individuals}$, $modules$
$sortByFittness(info_{individuals}, fitnesses)$;
$markedModules \leftarrow \emptyset$;
for $(individualNum, moduleStates) \in info_{individuals}$ **do**
  for $(moduleNum, states) \in moduleStates[individualNum]$ **do**
    if *not* $moduleNum \in markedModules$ **then**
      $modules[moduleNum] \leftarrow mostCostyState(states)$;
    end
  end
  $markedModules \leftarrow markedModules \cup info[modulesNums]$
end

3.6 Выбор оптимальной стратегии

Проведем экспериментальное сравнение двух предложенных стратегий. Параметры экспериментов описаны в разделе "Экспериментальные исследования предложенного подхода". Для серии из 4-х экспериментов в 7 поколений генетического алгоритма рассмотрим изменение точности лучшей сети по поколениям. На рис. 8-9 изображено изменение точности лучшей сети по поколениям для наборов данных MNIST и CIFAR-10.

Из рис. 8-9 видно, что на наборах данных CIFAR-10 и MNIST для метода с наследованием генотипов наблюдается нестабильное поведение точности, не соответствующее желаемому поведению генетического алгоритма, нацеленного на поэтапное улучшение точности результата. Метод без наследования генотипов показал наглядное улучшение точности. Исходя из этого был выбран метод без наследования генотипов.

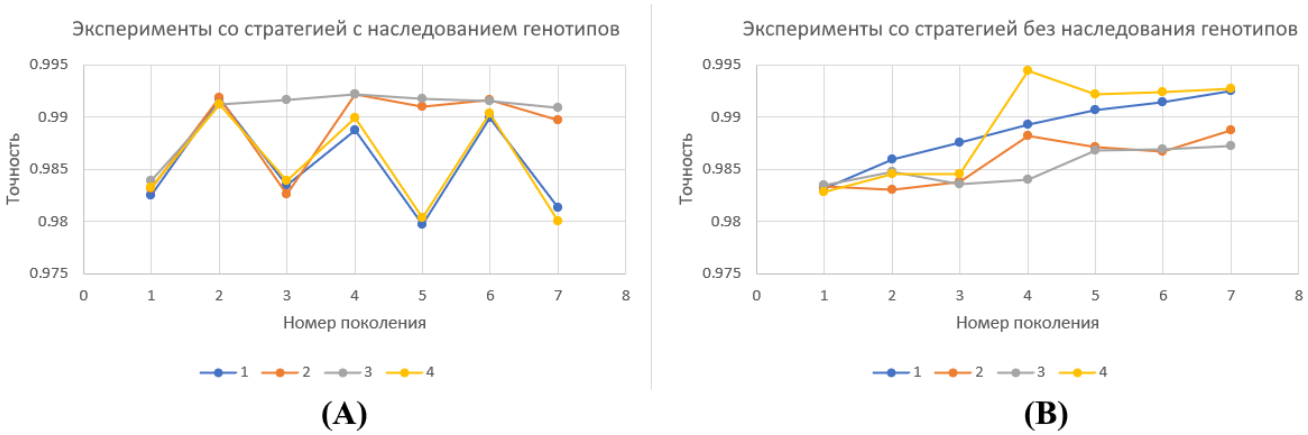


Рис. 8: MNIST. Точность лучшей сети по поколениям. A – метод с наследованием генотипов, B – метод без наследования генотипов по четырем экспериментам. Эксперименты определяются цветами своих графиков.

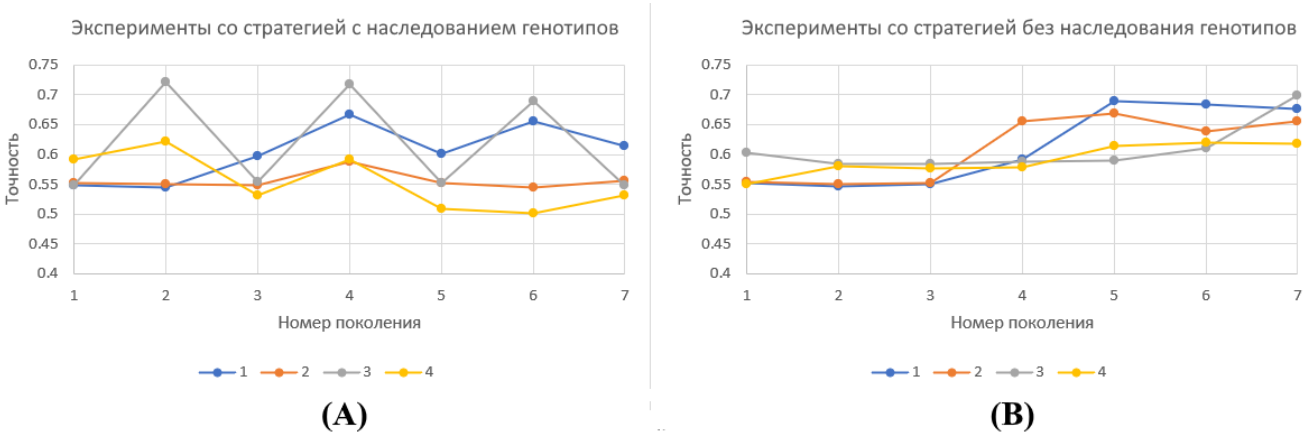


Рис. 9: CIFAR-10. Точность лучшей сети по поколениям. A – метод с наследованием генотипов, B – метод без наследования генотипов по четырем экспериментам. Эксперименты определяются цветами своих графиков.

4 Предлагаемый метод

В разделе "Выбор оптимальной стратегии"сделан вывод о применимости метода, описанного в разделе "Эволюция сжатых сетей"для ускорения поиска архитектур. Этот метод рассматривается в работе в качестве основного.

4.1 Описание метода

Схема предложенного метода представлена на рис. 10. Рассмотрим структуру итерации эволюционного алгоритма:

1. Генерация популяции сетей-особей из генотипов
2. Вычисление функции приспособленности особей
   (a) Сжатие сетей из популяции с помощью MorphNet
   (b) Тренировка сжатых сетей, оценка точности
3. Отбор
4. Скрещивание
5. Мутация

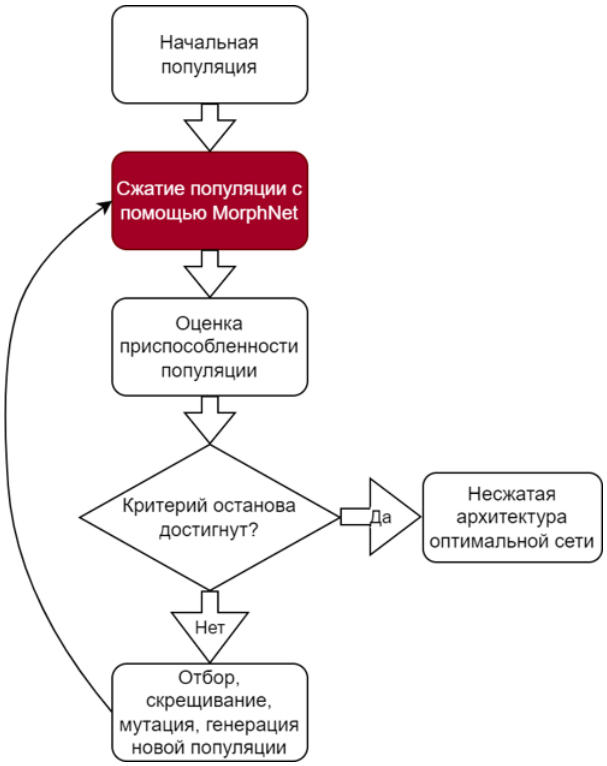


Рис. 10: Схема предложенного метода.

4.2 Оценка эффективности

Приведем оценку эффективности предложенного метода с учетом следующих допущений:

1. Размер популяции m.
2. Каждая сеть обучается в течении n эпох перед оценкой.
3. Перед обучением проводится k эпох сжатия MorphNet.
4. За одну эпоху сжатия FLOPN уменьшается в $a \geq 1$ раз.
5. Время, необходимое на обучение и оценку сети, прямо пропорционально ее значению FLOPN и определяется коэффициентами c и d соответственно. Пусть f – значение FLOPN сети. Время на обучение в одну эпоху – $f \cdot c$, время на оценку сети на валидационном наборе данных – $f \cdot d$.

Тогда ускорение метода при оценке популяции относительно CoDeepNEAT (при условии одинаковой популяции архитектур) согласно описанной модельной конфигурации составит:

$$\frac{\sum_{i=1}^{m} f_i \cdot (cn+d)}{\sum_{i=1}^{m} f_i ck + \frac{f_i}{a^k} \cdot (cn+d)} = \frac{\sum_{i=1}^{m} f_i \cdot (cn+d)}{\sum_{i=1}^{m} f_i \cdot (ck + \frac{cn+d}{a^k})} = \frac{cn+d}{ck + \frac{cn+d}{a^k}}.$$

Таким образом, ускорение будет наблюдаться при следующем соотношении сжатия и обучения:

$$ck < (cn + d) \cdot \frac{a^k - 1}{a^k}, \text{ или}$$

$$\frac{1}{a^k} < 1 - \frac{ck}{cn + d}.$$

5 Реализация предложенного подхода

Предложенный метод требует задания размеров популяций модулей, шаблонов и сетей, задания возможных конфигураций отбираемых сетей, выбора количества поколений генетического алгоритма и степени параллелизма этапа оценки популяции. В реализации использовались:

1. Система программирования Python.
2. Библиотеки машинного обучения Keras, TensorFlow.
3. MorphNet.
4. CoDeepNEAT на основе AutoML библиотеки NNI (Neural Network Intelligence) [13], поддерживающей параллельную обработку.

Реализация представляет собой программу, структурно разделенную на реализации функции приспособленности, базовых функций генетического алгоритма и распределения работы. Схема реализации представлена на рис. 11.

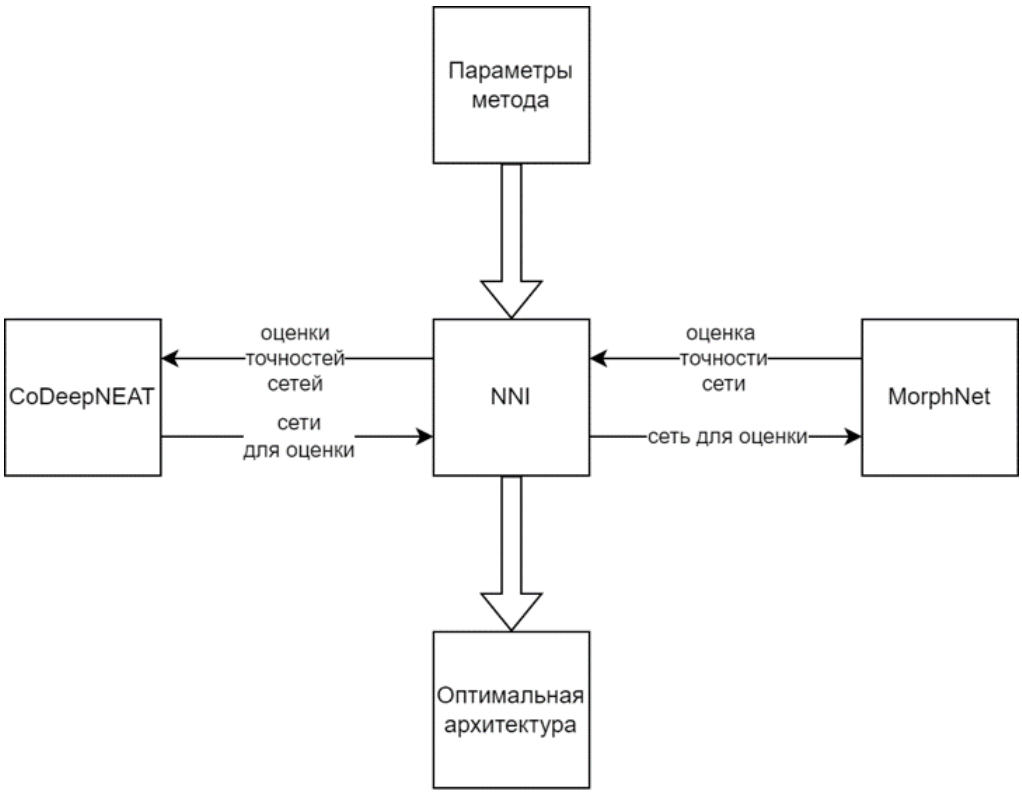


Рис. 11: Схема реализации предложенного метода.

6 Экспериментальные исследования предложенного подхода

6.1 Подход к сравнению нейроэволюционных методов

Чтобы свести к минимуму влияние случайности, являющейся неотъемлемой харакеристикой эволюционных алгоритмов, каждая группа экспериментов использует одну и ту же заранее сгенерированные популяции модулей, шаблонов и сетей в качестве стартового положения процесса эволюции. Учитывая, что количество поколений в эволюционном процессе зафиксировано и одинаково в соответствующих экспериментах, сравнение предлагаемого подхода со стандартным CoDeepNEAT [2] можно провести путем анализа средней точности сетей из последнего поколения.

Сравниваются две группы экспериментов с одним и тем же набором данных, первая использует предложенный подход, вторая использует обычный CoDeepNEAT. Популяции сетей, принадлежащих к одной и той же группе объединяются в единую смешанную популяцию; вычисляется средняя точность по смешанной популяции. Для каждой группы экспериментов вычисляется среднее время, потребовавшееся для рассмотрения фиксированного числа поколений.

Ключевыми значениями для оценки производительности предлагаемого метода являются его ускорение и деградация точности. Ускорение вычисляется как $\frac{t_a}{t_b}$, где t_a – среднее время по группе экспериментов с CoDeepNEAT, а t_b – среднее время по группе экспериментов с предложенным подходом. Деградация точности вычисляется как $acc_a - acc_b$, где acc_a – средняя точность по смешанной популяции из экспериментов с CoDeepNEAT, а acc_b – средняя точность по смешанной популяции из экспериментов с предложенным подходом. Отрицательное значение деградации точности означает, что предложенный метод дал лучшие результаты, чем исходный CoDeepNEAT.

6.2 Параметры экспериментов и конфигурации

Кластерная конфигурация Для экспериментального исследования предлагаемого подхода использовалась следующая кластерная конфигурация:

- 1 узел кластера Polus cluster с 2 x IBM POWER8 процессорами, поддерживающими до 160 потоков, 256GB RAM, 2 x Nvidia Tesla P100 [12].

Наборы данных Использовались следующие наборы данных:

- MNIST
- CIFAR-10
- CIFAR-100 (аугментация в 3 раза)
- Tiny ImageNet (100 классов, аугментация в 4 раза)

Параметры эволюции Эксперименты на MNIST и CIFAR-10 проводились на 7-ми поколениях, эксперименты на CIFAR-100 и Tiny ImageNet проводились на 4-х поколениях. Каждая группа состоит из 4 экспериментов, размер популяции во всех экспериментах - 12 сетей.

Конфигурация MorphNet Проводилась одна эпоха сжатия сети по полному набору тренировочных данных. В качестве силы регуляризации выбиралось значение $\frac{1}{init\_FLOPN}$, где $init\_FLOPN$ это значение FLOPN несжатой сети. Порог зануления весовых коэффициентов устанавливался как 0.1 для MNIST и CIFAR-10 и как 0.05 для CIFAR-100 и Tiny ImageNet.

Пространство возможных конфигураций архитектур Настройка пространства возможных конфигураций архитектур (или пространства поиска) проводилась вручную методом проб и ошибок, учитывался предыдущий опыт авторов в решении смежных задач. Пространство поиска для MNIST и CIFAR-10 состоит из: слои Conv2D с $16-68$ фильтрами (возможные размеры ядра $[1, 3, 5]$) с допустимыми смежными MaxPooling2D слоями (параметр pool size устанавливался как 2) или Dropout слоями (параметр dropout rate $0 - 0.5$); предвыходной слой Dense размером $32 - 256$ (с ReLu активацией).

Пространство поиска для CIFAR-100 и Tiny ImageNet состоит из: слои Conv2D с $16 - 128$ фильтрами (возможные размеры ядра $[2, 4, 7, 9, 11, 13]$)) с допустимыми смежными MaxPooling2D слоями (параметр pool size устанавливался как 2) или Dropout слоями (параметр dropout rate $0-0.5$); предвыходные два слоя Dense размером $32-128$ (с ReLu активацией).

6.3 Результаты экспериментов

Среднее время и точность для каждого набора данных представлены в таб. 1-2. Сравнение времени и точности на разных наборах данных представлено на рис. 12.

Таблица 1: Средняя точность и время обычного CoDeepNEAT для разных наборов данных

	MNIST	CIFAR-10	CIFAR-100	Tiny ImageNet
Время, мин.	116.1	111.79	95.6	490.22
Точность, %	98.95	66.57	30.03	11.75

Таблица 2: Средняя точность и время предложенного метода для разных наборов данных

	MNIST	CIFAR-10	CIFAR-100	Tiny ImageNet
Время, мин.	78.6	61.29	41.21	228.18
Точность, %	98.93	64.96	33.2	12.14

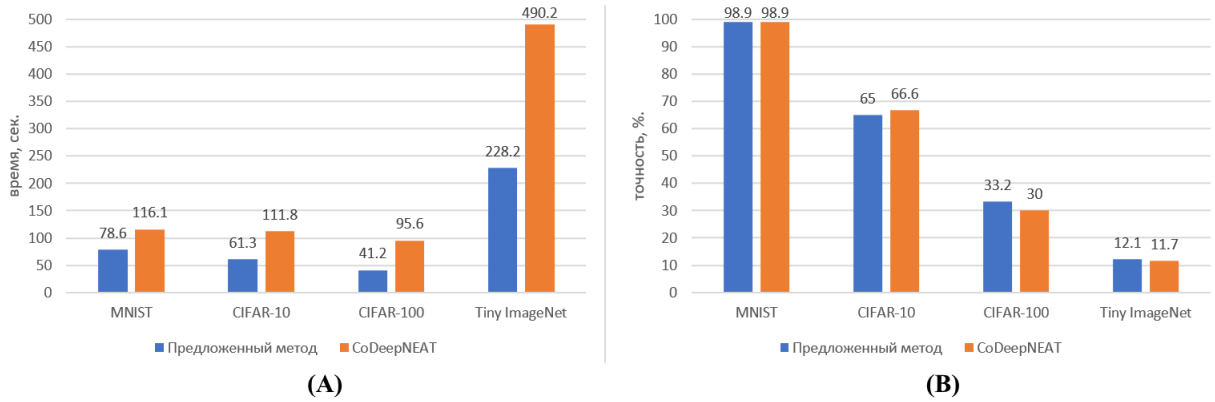


Рис. 12: (A) – сравнение времени на разных наборах данных. (B) - сравнение точности на разных наборах данных.

Из диаграмм на рис. 12 видно, что предложенный метод позволил ускорить поиск архитектур на всех рассмотренных наборах данных. При этом метод не оказал значительного влияния на точность результата.

Ускорение и деградация точности для каждого набора данных представлены в таб. 3:

Таблица 3: Ускорение и деградация точности на разных наборах данных

	MNIST	CIFAR-10	CIFAR-100	Tiny ImageNet
Ускорение	1.48	1.82	2.31	2.14
Дегр. точности, %	0.02	1.61	-3.16	-0.38

Отрицательное значение деградации точности означает, что предложенный метод превзошел CoDeepNEAT по точности. Из таб. 3 видно, что лучший результат был получен на наборе данных CIFAR-10: метод позволил ускорить поиск архитектур в 2.31 раза относительно CoDeepNEAT и не привел к ухудшению точности результата.

Значение функции потерь и изменение точности в процессе обучения лучшей архитектуры для каждого набора данных представлены на рис. 13-16 (значения измерялись по валидационной выборке):

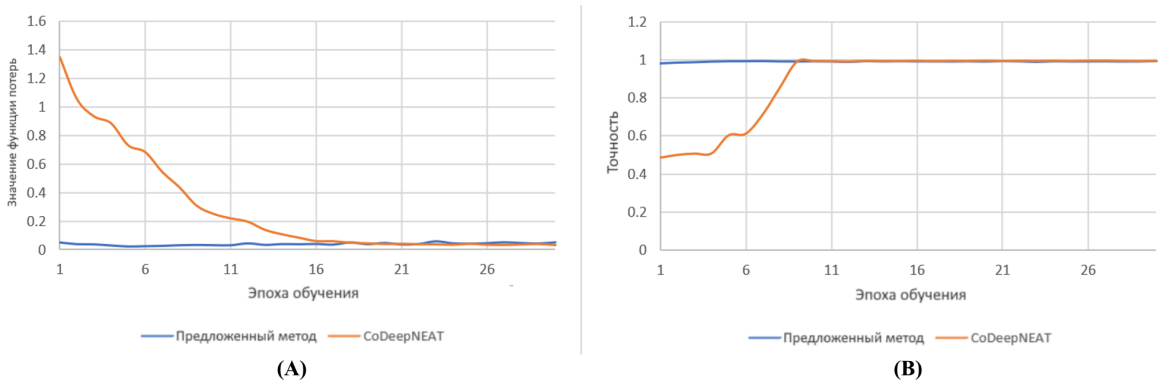
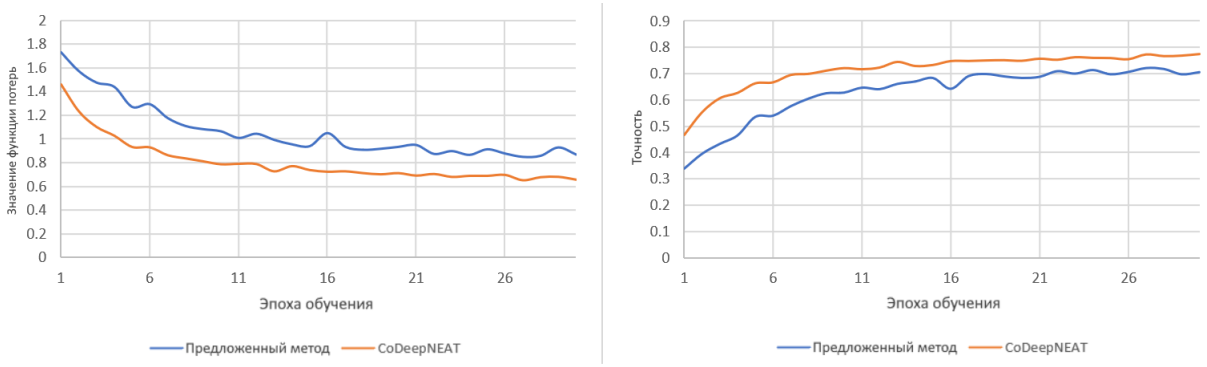
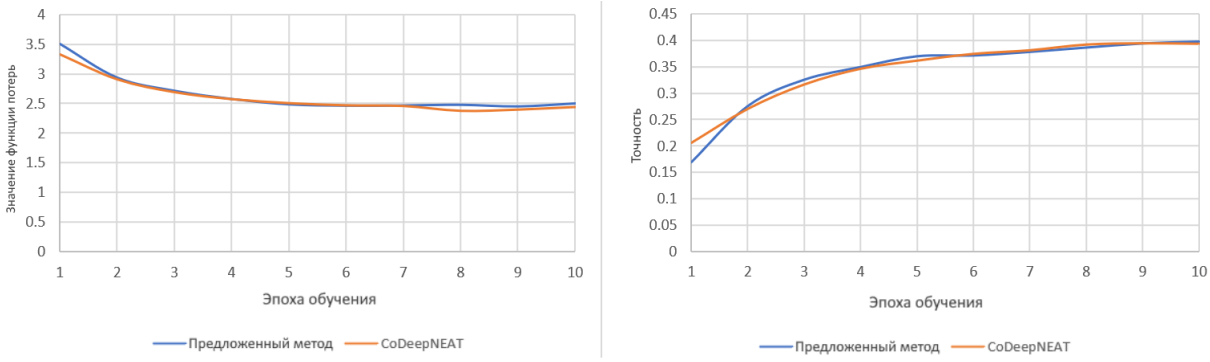


Рис. 13: Процесс обучения на MNIST. (A) – значение функции потерь, (B) - точность.



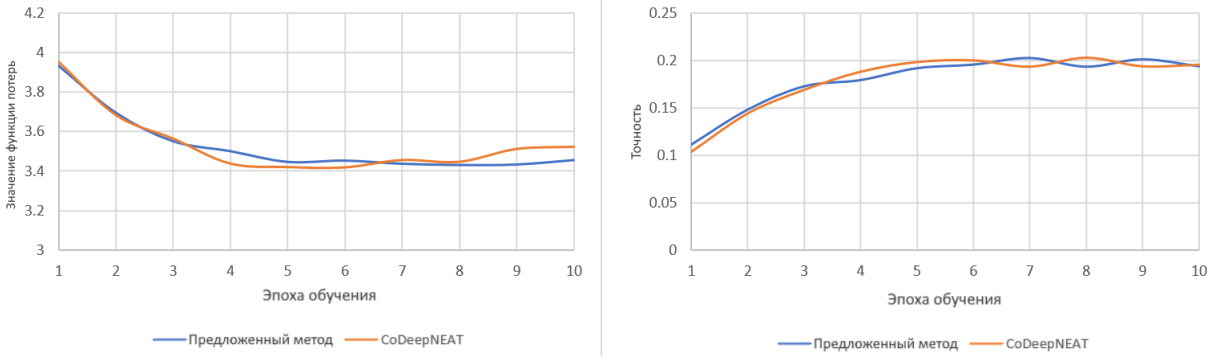
(A) **(B)**

Рис. 14: Процесс обучения на CIFAR-10. (A) – значение функции потерь, (B) - точность.



(A) **(B)**

Рис. 15: Процесс обучения на CIFAR-100. (A) – значение функции потерь, (B) - точность.



(A) **(B)**

Рис. 16: Процесс обучения на Tiny ImageNet. (A) – значение функции потерь, (B) - точность.

6.4 Выводы

Предложенный метод может использоваться для поиска архитектур глубоких нейронных сетей. Он позволил ускорить поиск архитектур относительно CoDeepNEAT в 1.9 раз в среднем по экспериментам на рассмотренных наборах данных без существенного влияния на точность результата.

7 Заключение

В работе получены следующие результаты:

1. Проведен обзор методов поиска и оптимизации архитектур нейронных сетей.
2. Предложен эволюционный метод поиска архитектур глубоких нейронных сетей, использующий регуляризационное сжатие популяции. Сжатие выполняется в процессе эволюции.
3. Реализованы и исследованы две стратегии регуляризационной оптимизации в процессе эволюционного отбора с обособленными популяциями модулей и шаблонов.
4. Проведено экспериментальное исследование предложенного метода на наборах данных MNIST, CIFAR-10, CIFAR-100 и Tiny ImageNet. Показано, что метод позволяет ускорить процесс нейроэволюции без существенных потерь в точности генерируемых сетей.

Результаты докладывались на конференции “Тихоновские чтения 2021” [11]. Статья, описывающая результаты работы, подана на конференцию “Суперкомпьютерные дни в России 2022”.

8 Список литературы

[1] Gordon A., Eban E., Nachum O., et al. MorphNet: Fast Simple Resource-Constrained Structure Learning of Deep Networks // IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2018. DOI:10.1109/CVPR.2018.00171.

[2] Miikkulainen R., Liang J., Meyerson E., et al. Evolving deep neural networks // Artificial Intelligence in the Age of Neural Networks and Brain Computing. Elseiver 2019. P.293-312. DOI:10.1016/B978-0-12-815480-9.00015-3

[3] Stanley K.O., Miikkulainen R. Evolving neural networks through augmenting topologies // Evolutionary computation. MIT Press. 2002. Vol. 10 P.99–127. DOI:10.1162/artl.2009.15.2.15202.

[4] Liashchynskyi P., Liashchynskyi P. Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS // Ternopil National Economic University. 2019. CoRR:1912.06059.

[5] Kandasamy K., Neiswanger W., Schneider J., Poczos B., and P Xing E. Neural architecture search with bayesian optimisation and optimal transport // Advances in Neural Information Processing Systems (NeurIPS). 2018. P.2016–2025.

[6] Zoph B., Quoc V. Le. Neural Architecture Search with Reinforcement Learning // The International Conference on Learning Representations (ICLR). 2017.

[7] Pham H., Melody Y. Guan, Zoph B., Quoc V. Le, Dean J. Efficient Neural Architecture Search via Parameter Sharing // 35th International Conference on Machine Learning. 2018. PMLR 80:4095-4104.

[8] Kwasigroch A., Grochowski M., Mikolajczyk M. et al. Deep neural network architecture search using network morphism // 24th International Conference on Methods and Models in Automation and Robotics (MMAR). 2019. DOI:10.1109/MMAR.2019.8864624.

[9] Scardapane S., Comminiello D., Hussain A., Uncini A. Group Sparse Regularization for Deep Neural Networks // 2017. DOI:10.1016/j.neucom.2017.02.029.

[10] Shafiee M. J., Hryniowski A., Li F., Qiu Lin Z., Wong A. State of Compact Architecture Search For Deep Neural Networks // Waterloo Artificial Intelligence Institute, University of Waterloo. 2019. CoRR:abs/1910.06466.

[11] Никуленков М.Р., Попова Н.Н., Хамитов К.Г. Анализ эффективности применения регуляризационной оптимизации для ускорения метода поиска архитектур глубоких нейросетей // Тихоновские чтения. 2021. https://cs.msu.ru//news/3573.

[12] Спецификации кластера Polus // 2020. http://hpc.cs.msu.su/polus.

[13] Neural Network Intelligence // 2020. https://www.microsoft.com/en-us/research/project/neural-network-intelligence/.