УДК 004.051, 004.042, 004.021 Дата подачи статьи: 19.06.19

DOI: 10.15827/0236-235X.128.639-649 2019. Т. 32. № 4. С. 639–649

Проектирование быстрой программной реализации специализированной нейросетевой архитектуры

с разреженными связями

*Ю.С. Федоренко* 1*, аспирант,* [*fedyura11235@mail.ru*](mailto:fedyura11235@mail.ru)

1 *Московский государственный технический университет им. Н.Э. Баумана, г. Москва, 105005, Россия*

Статья посвящена разработке быстрой программной реализации специализированной нейросетевой архитектуры. Конструирование признаков является важнейшим этапом в решении любой задачи ма- шинного обучения. Алгоритмы ручного отбора признаков в настоящее время теряют свою популяр- ность в ряде задач, уступая глубоким нейросетям. Однако применение глубоких моделей ограничено в задачах онлайнового (динамического) обучения, поскольку они не способны обучаться в режиме ре- ального времени. Кроме того, их использование в высоконагруженных системах затруднительно из-за вычислительной сложности.

В одной из работ автором совместно с коллективом была предложена архитектура нейронной сети, позволяющая осуществлять автоматический подбор признаков и при этом обучаться в режиме реаль- ного времени. Однако специфическая разреженность связей в этой архитектуре затрудняет ее реализа- цию на базе стандартных библиотек для работы с глубокими нейросетями. Поэтому было принято ре- шение сделать собственную реализацию предложенной архитектуры.

В статье рассмотрены структуры данных и алгоритмы, разработанные при написании программной реализации. Подробно описан процесс обработки примеров с позиции программной системы при пред- сказании и обучении модели. Для более полного описания особенностей реализации системы приве- дены UML-диаграммы классов, последовательностей и активности.

Проведены эксперименты по сравнению быстродействия созданной реализации и реализаций ана- логичной нейросетевой архитектуры на базе библиотек для работы с глубокими нейросетями. Анализ показал, что разработанная реализация работает на порядок быстрее реализаций на базе фреймворков для глубокого обучения. Такое ускорение связано с тем, что она оптимизирована под конкретную нейросетевую архитектуру в отличие от библиотек, рассчитанных на работу с широким классом нейронных сетей. Также экспериментальный анализ показал, что разработанная реализация нейросети работает всего на 20–30 % медленнее, чем простая логистическая регрессия с хешированием призна- ков, что позволяет использовать ее в высоконагруженных системах.

***Ключевые слова****: отбор признаков, глубокие нейронные сети, категориальные переменные, разре- женные связи, модульная арифметика, переиспользование индексов, бенчмарки.*

Постоянно растущий объем данных требует создания алгоритмов для их обработки. По этой причине в ряде областей задачи машин- ного обучения и интеллектуального анализа данных становятся все более распространен- ными и актуальными. Важное место в них за- нимает извлечение признаков, поскольку ис- пользование слабо релевантных признаков резко снижает эффективность любых алгорит- мов. И если еще 10–20 лет назад был распро- странен ручной подбор признаков [1], то сей- час наиболее перспективными являются нейросетевые алгоритмы, осуществляющие ав- томатический подбор признаков в процессе ре- шения задачи. Это позволяет сократить ручной объем работы для исследователей, а в ряде слу- чаев и повысить качество работы [2]. Однако глубокие нейронные сети сложны с вычисли-

тельной точки зрения, что затрудняет их при- менение в высоконагруженных системах с жесткими требованиями по времени ответа.

Кроме того, глубокие нейронные сети плохо подходят для решения задач онлайнового (ди- намического) обучения, поскольку они ста- тичны и не могут обучаться в режиме реаль- ного времени. По этой причине в подобных за- дачах легкие модели (например, логистическая регрессия) зачастую выигрывают у тяжеловес- ных, поскольку позволяют обновлять пара- метры в режиме реального времени. Однако столь простые модели требуют ручного под- бора признаков, что отнимает много времени у исследователей, и порой необходимо привле- чение к работе экспертов заданной предметной области. В работе [3] авторами была предло- жена нейронная сеть, которая осуществляет ав-

томатизированный подбор признаков, поддер- живая при этом обновление параметров в ре- жиме реального времени. Данная статья посвя- щена вопросам программной реализации этой архитектуры.

# Описание архитектуры

Рассматриваемая модель представляет со- бой нейронную сеть с одним скрытым слоем. Отличие от традиционного персептрона заклю- чается в разреженности архитектуры: нейроны скрытого слоя связаны только с частью нейро- нов входного слоя. Выходной слой нейронной сети состоит из одного нейрона с сигмоидаль- ной функцией активации, которая выдает веро- ятность для заданного набора входных пара- метров.

На вход модель принимает заданное коли- чество признаков. Сами признаки категориаль- ные, и для их подачи на вход применяется уни- тарное кодирование [4]. Каждому признаку от- водится одинаковый диапазон значений во входном векторе. Схема архитектуры пред- ставлена на рисунке 1.

Специфичность данной архитектуры заклю- чается в том, что она имеет разреженные связи с определенной структурой. И это дает ей ряд преимуществ.

# Особенности разреженной нейросетевой архитектуры

Исследования работы головного мозга по- казывают, что биологические нейроны коди- руют информацию в разреженном и распреде- ленном видах. Согласно оценкам, процент ак- тивных нейронов в один и тот же момент времени колеблется от 1 до 4. Это соответ- ствует балансу между разнообразием возмож- ных представлений и небольшим потребле- нием энергии. Традиционные сети прямого распространения без использования *L*1 регуля- ризации не обладают таким свойством. Напри- мер, при использовании сигмоидальной функ- ции активации нейроны после начальной ини- циализации имеют устойчивое состояние на середине между режимами насыщения. Это выглядит неестественно с биологической точки зрения и вредит оптимизации на основе градиентного спуска.

Разреженные представления имеют не- сколько преимуществ [5]. В контексте рассмат- риваемой задачи стоит выделить эффективное представление данных переменного размера. Различные входы могут содержать разное ко- личество информации, и потому их более удобно представлять в виде структур с пере- менным размером. Стоит отметить, что слиш-



**relu12**

feature\_1

**relu1N**

Нейроны

скрытого слоя, соединенные с нейронами, отвечающими за 2 входных признака

feature\_2

**relu2N**

σ

pr

**relu1**

feature\_N

**relu2**

Нейроны

скрытого слоя, соединенные с нейронами, отвечающими за 1 входной признак

**reluN**

*Рис. 1. Схема предложенной нейросетевой архитектуры*

*Fig. 1. The proposed neural network architecture*

ком высокая степень разреженности может приводить к деградации модели, поскольку она сокращает ее емкость. Однако на сегодняшний день известно, что глубокие нейронные сети часто содержат избыточное число параметров (что приводит к усложнению вычислений и ро- сту потребления ресурсов), поэтому их можно значительно упростить без существенной по- тери качества [6]. Помимо вычислительной из- быточности, большое количество параметров зачастую ухудшают обобщающую способ- ность моделей, делая их более подверженными переобучению.

Следовательно, можно удалить много пара- метров нейронной сети без существенного ухудшения (а порой и с улучшением) произво- дительности. Разумеется, такие изменения при- водят к возникновению разреженных архитек- тур. Помимо пониженных требований к памяти (нужно хранить меньшее число параметров), сокращение числа параметров позволяет упро- стить итоговые вычисления и уменьшить время предсказания, что играет роль в высоко- нагруженных системах или в системах, работа- ющих со слабым аппаратным обеспечением.

Есть различные стратегии сокращения па- раметров нейронных сетей. Стратегии усече- ния весов были предложены еще Лекуном в ра- боте [7], и они же остаются наиболее популяр- ными до настоящего времени. Относительно недавно в работе [8] был предложен алгоритм сокращения количества связей, основанный на похожести нейронов. Стратегия прореживания нейронной сети может быть также встроена в обучение модели. Еще один подход заключа- ется в обучении маленькой модели, которая по своему поведению будет имитировать боль- шую модель [9]. Кроме того, в некоторых рабо- тах предлагается обучать глубокие модели, но с меньшим числом параметров. Оставшиеся параметры при этом должны предсказываться на основании уже обученных.

В предложенной нейросети разреженность уже заложена в саму архитектуру с учетом ре- шаемой задачи (предсказание поведения поль- зователей в Интернете). Это осложняет ее реа- лизацию на базе таких нейросетевых фрейм- ворков, как Pytorch, Tensorflow, Theano и т.д. Разреженность связей можно реализовать при помощи обнуления некоторых весовых коэф- фициентов или путем использования модулей Sparse. Однако полученная реализация получа- ется в несколько раз медленнее, чем простая логистическая регрессия, что осложняет пере- ход на нее в высоконагруженных системах. По

этой причине для применения данной архитек- туры в задаче предсказания поведения пользо- вателей в Интернете было принято решение разработать самостоятельную программную реализацию, оптимизированную непосред- ственно под данную архитектуру. Далее рассматривается концепция программной реа- лизации и описываются разработанные алго- ритмы, позволяющие обеспечить быстродей- ствие предложенной нейронной сети.

# Диаграмма классов системы

Диаграмма основных классов реализован- ной библиотеки представлена на рисунке 2. Класс feature\_t используется для описания под- держиваемых признаков в модели, level\_t – для описания уровня модели, который может со- стоять из нескольких признаков (в таком слу- чае применяется класс hashing trick [10]), а levels\_holder\_t описывает конфигурацию мо- дели целиком (модель состоит из набора уров- ней). Помимо конфигурации, у модели есть набор настроек (к примеру, количество элемен- тов, отводимых на представление каждого признака), который описывается классом lr\_settings\_t. В результате класс самой нейросе- тевой модели наследует от класса настроек и класс конфигурации модели. Сама нейросете- вая модель описывается двумя классами: класс neural\_net\_t позволяет только использовать мо- дель (загрузив ее предварительно из дампа), а класс neural\_net\_fitter\_t – обучать. Такое разде- ление дает возможность программным компо- нентам, которые только используют обучен- ную модель, не хранить множество дополни- тельных параметров, связанных с настройками обучения. Из рисунка 2 видно, что класс lr\_settings\_t оказывается включенным дважды. Для разрешения этой ситуации используется виртуальное наследование. Также с классом neural\_net\_fitter\_t жестко связан объект опти- мизатора (композиция), то есть он создается в конструкторе класса neural\_net\_fitter\_t. Разде- ление оптимизаторов и самих моделей дает по- тенциальную возможность использовать одни и те же оптимизаторы для обучения разных мо- делей.

# Структуры данных и алгоритмы создания нейросети

Для реализации выбран язык программиро- вания C++, так как он включает в себя множе- ство современных инструментов программиро-

**Logger**

-m\_filename: string

-m\_log\_level: int

+Logger()

+log(log\_level: int, msg: string)

+setLogLevel(log\_level: int)

**lr\_settings\_t**

**levels\_holder\_t**

-m\_levels: vector<level\_t>

-m\_is\_valid: bool

#logger: Logger\*

+reset(levels: vector<vector<feature\_t>>)

+contains(feature: EFeature): bool

#unique\_key(event: vector<vector<int>>): vector<int>

-reset\_meta\_level()

**level\_t**

-m\_features: vector<feature\_t>

UNREGISTERED UNREGISTERED UNREGISTERED UNREGISTERED UNREGISTERED

-m\_typed: bool

+level\_t()

+reset(features: feature\_t, typed: bool): bool

+features(): vector<features>

+for\_each\_key(e: vector<EFeature>, level\_num: int)

+feature\_from\_event(e: vector<EFeature>): feature\_t

#m\_I: int

#m\_level\_I: vector<int>

+lr\_settings\_t(num\_levels: int)

+N(): int

**feature\_t**

-m\_feature: EFeature

+feature\_t(f: EFeature)

+feature(): EFeature

+create\_by\_name(name: string): feature\_t

**lr\_fitter\_settings\_t**

#m\_a: double #m\_lambda: double #m\_epoch\_count: int #...

+lr\_fitter\_settings\_t(num\_levels: int)

+update(rhs: lr\_fitter\_settings\_t)

**neural\_net\_t**

#m\_num\_hidden: int

UNREGISTERED UNREGISTERED UNREGISTERED UNREGISTERED UNREGISTERED

#m\_hidden\_weights: vector<vector<float>> #m\_output\_weights: vector<float> #m\_structure: vector<vector<int>>

+predict(e: vector<int>): float

+load()

+size()

«enumeration»

**gd\_algorithm\_t**

Vanilla Adam RmsProp

...

«enumeration»

**EFeature**

eAge eGeo eBanner eContents

eUserTopics

...

**neural\_net\_fitter\_t**

-m\_processed\_files: int

-m\_gd\_optimizer: gd\_optimizer\_t

+m\_logloss: float

+neural\_net\_fitter\_t(f: vector<vector<feature\_t>>, name: string)

-N: int

-b1: float

-u: vector<float>

-v: vector<float>

**gd\_optimizer\_t**

+createNNStructure()

+update(e: vector<int>, y: int): float

+fit(): bool

+ready\_to\_fit(): bool

+gd\_optimizer\_t(N: int, gd\_algo: gd\_algorithm\_t, b1: float)

+step(g: double, lr: double, i: int): double

+addEventsCount()

-countCorrection(i: int)

*Рис. 2. Диаграмма классов реализованной библиотеки Fig. 2. The implemented library class diagram*

UNREGISTERED UNREGISTERED UNREGISTERED UNREGISTERED UNREGISTERED

вания, таких как лямбда-функции, move-семан- тика, умные указатели и т.д., сохраняя при этом хорошую производительность [11].

Архитектура нейросети задается статически и хранится в двумерном массиве. Входные дан- ные поступают в виде набора из *L* признаков, каждый из которых имеет *N* различных значе- ний. Таким образом, входное пространство значений разбито на *L* диапазонов, каждый из которых соответствует одному признаку. С учетом архитектуры сети количество нейро-

по номеру набора признаков *li* быстро получать затронутые номера нейронов скрытого слоя, была создана матрица соединений нейронной сети. В этой матрице каждому номеру признака на входном слое соответствует список затрону- тых номеров нейронов скрытого слоя. Для нейронной сети, где *L* = 5 (5 различных катего- риальных признаков), данная матрица выгля- дит так, как показано в таблице 1.

*Таблица 1*

**Пример матрицы соединений**

нов скрытого слоя равняется

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Признак** | **Нейрон** | | | | |
| **1** | **2** | **3** | **4** | **5** |
| 0 | 0 | 5 | 6 | 7 | 8 |
| 1 | 1 | 5 | 9 | 10 | 11 |
| 2 | 2 | 6 | 9 | 12 | 13 |
| 3 | 3 | 7 | 10 | 12 | 14 |
| 4 | 4 | 8 | 11 | 13 | 14 |

*L*  ( *L*  1)  *L* 

2

**для пяти признаков**

*Table 1*

 *L*  (*L*  1) . Поскольку сеть не является полно-



2

связной, хранить полную матрицу весов (с ну- левыми весами в позициях, где нет связей) расточительно, поэтому в разработанной реа- лизации хранятся только ненулевые коэффици- енты. Благодаря фиксированной архитектуре можно заранее рассчитать размер такой мат- рицы с ненулевыми коэффициентами. Чтобы

**The example of a matrix of coupling for 5 features**

Для формирования такой матрицы сначала заполняется вспомогательный двумерный мас- сив соединений для нейронов скрытого слоя (для каждого нейрона скрытого слоя содер- жится массив с номерами связанных с ним нейронов входного слоя). Количество строк равняется количеству нейронов скрытого слоя, а количество столбцов – числу связей каждого нейрона (для данной архитектуры оно равно 1 или 2). Для нейронной сети из пяти признаков такая вспомогательная структура имеет вид, представленный в таблице 2.

*Таблица 2*

**Вспомогательная матрица**

**для нейронной сети с пятью признаками**

*Table 2*

**A utility matrix for a neural network with 5 features**

Такая структура данных заполняется на ос- нове модульной арифметики. Приведем алго- ритм ее заполнения (рис. 3).

Сложность такого алгоритма составляет *O*(*n*2), где *n* – количество уровней в модели. Учитывая, что *n* не превышает нескольких де- сятков, это не является проблемой. Кроме того, создание архитектуры нейросети делается только один раз за всю работу сети, поэтому доля затраченного на это времени крайне не- значительна. Чтобы из такой вспомогательной матрицы получить основную матрицу связей, достаточно однократного прохода по циклу. Алгоритм 2 описывает формирование матрицы соединений нейронной сети (рис. 4).

Сложность данного алгоритма составляет *O*(*n*3), где *n* – количество уровней модели (по- скольку количество скрытых нейронов сети пропорционально квадрату числа уровней в модели). Однако и это не является проблемой по тем же причинам, что и в алгоритме 1. В итоге для каждого значения входного при- знака можно получить номера связанных с ним нейронов скрытого слоя. Поскольку каждый нейрон входного слоя связан ровно с *L* нейро- нами скрытого слоя, веса нейронов скрытого слоя можно хранить в матрице размерности *N**L*×*L*. В случае полносвязной сети аналогич- ного размера для хранения весов скрытого слоя потребовалась бы матрица размера

*L*  (*L*  1)

*N*  *L* 

|  |  |  |
| --- | --- | --- |
| **№ нейрона скрытого**  **слоя** | **№ нейрона 1 входного**  **слоя** | **№ нейрона 2 входного**  **слоя** |
| 0 | 0 | - |
| 1 | 1 | - |
| 2 | 2 | - |
| 3 | 3 | - |
| 4 | 4 | - |
| 5 | 0 | 1 |
| 6 | 0 | 2 |
| 7 | 0 | 3 |
| 8 | 0 | 4 |
| 9 | 1 | 2 |
| 10 | 1 | 3 |
| 11 | 1 | 4 |
| 12 | 2 | 3 |
| 13 | 2 | 4 |
| 14 | 3 | 4 |

, то есть в *L*/2 раз больше. При

2

*L* = 20 признакам получается экономия памяти в 10 раз.

Порядок создания нейронной сети с учетом вспомогательных объектов изображен на диа-

**Алгоритм 1**. Заполнение вспомогательной матрицы для нейронной сети **Исходные параметры**: количество входных уровней levels\_num **Результат**: вспомогательная матрица соединений

count = 0;

i = 0 **до тех пор, пока** i < levels\_num **выполнять**

добавить в матрицу строку номер count с текущим номером уровня i;

инкрементировать счетчик count; перейти к следующему уровню;

**конец**

i = 0 **до тех пор, пока** i < levels\_num **выполнять**

j = i + 1 **до тех пор, пока** j < levels\_num **выполнять**

добавить в матрицу строку номер count, содержащую 2 текущих номера уровня i и j; инкрементировать счетчик count;

перейти к следующему уровню внутреннего обхода;

**конец**

перейти к следующему уровню внешнего обхода;

**конец**

*Рис. 3. Алгоритм 1*

*Fig. 3. Algorithm 1*

**gd\_optimizer\_t**

|  |  |
| --- | --- |
| **Алгоритм 2**. Заполнение основной матрицы соединений нейронной сети  **Исходные параметры**: Вспомогательная матрица соединений connections, Количество нейронов скрытого слоя num\_hidden  **Результат**: основная матрица связей  i = 0 **до тех пор, пока** i < num\_hidden **выполнять**  j = 0 **до тех пор, пока** j < числа нейронов входного слоя, связанных с i-м нейроном скрытого слоя (размер connections[i]) **выполнять**  получить номер нейрона входного слоя level = connections[i][j];  к строке level (соответствует связям набора входных признаков с номером  level) матрицы связей добавить номер нейрона скрытого слоя i;  перейти к следующему нейрону входного слоя, с которым связан i-й нейрон скрытого слоя;  **конец**  перейти к следующему нейрону скрытого слоя;  **конец**  *Рис. 4. Алгоритм 2*  *Fig. 4. Algorithm 2* | |
| грамме последовательности (рис. 5), на кото- рой видно, что при создании объекта нейросети сначала инициализируется объект настроек, за- тем создается конфигурация уровней, после чего инициализируются параметры обучения модели. Затем по вышеописанному алгоритму формируется нейронная сеть с заданной струк- турой, после чего создается объект оптимиза- тора. При создании объекта оптимизатора ис-  пользуются параметры обучения модели. | **Последовательность обработки примера нейросетью**  Обработка примеров нейронной сетью про- изводится в следующей последовательности:   * расчет номеров затронутых индексов (в силу разреженности данных для одного при- мера затрагиваются лишь несколько индексов); * получение номера уровня (это делается по номеру индекса, так как на каждый признак |



**neural\_net\_t**

**lr\_settings\_t levels\_holder\_t**

**lr\_fitter\_settings\_t**

1 : Инициализировать 2 : Настройки модели

3 : Задать уровни согласно

4 : Конфигурация уровней модели

5 : Инициализировать

6 : Настройки процедуры обучения

7 : Задать структуру модели 8 : Получить размер уровня

9 : Размер уровня

«create» 10 : Создать

**gd\_optimizer\_t**

11 : Инициализировать

12 : Запросить настройки обучения

13 : Настройки обучения

14 : Готовый оптимизатор

*Рис. 5. Диаграмма последовательности создания объекта нейронной сети*

*Fig. 5. The sequence diagram of neural network object creation*

отводится одинаковое количество значений);

* получение номеров нейронов скрытого слоя, с которыми связан данный уровень (с ис- пользованием матрицы, изображенной в таб- лице 1);
* расчет взвешенной суммы для нейрона выходного слоя;
* переиспользование рассчитанного набо- ра индексов для предсказания и обновления.

Процесс суммирования нейронов скрытого слоя изображен на рисунке 6. Основная осо- бенность суммирования заключается в том, что оно производится не по нейронам скрытого слоя, а по весам. Сначала рассчитывается вклад во все нейроны скрытого слоя от первого при- знака, затем – от второго и т.д. Такой способ суммирования позволяет получить ускорение за счет локального расположения в памяти сла- гаемых (весов).

При предсказании сначала производится расчет затронутых индексов и значений нейро- нов скрытого слоя. Эти значения используются при получении предсказания. Во время обуче- ния сначала предсказывается вероятность для данного примера, затем рассчитывается функ- ция потерь, на основании которой вычисля- ются градиенты и обновляются коэффициенты. При этом переиспользуются индексы и значе- ния коэффициентов на скрытом слое, получен-



**F1**

**0 5 6 7 8**

**1 5 9 10 11**

**F2**

**0**

**6 7 8**

**F3**

**2 6 9 12 13**

**F4**

**3 7 10 12 14**

**Структура нейросети**

**4 8 11 13 14**

**F5**

**Входной слой**

**Матрица весов скрытого**

**слоя**

*Рис. 6. Организация суммирования для нейронов скрытого слоя Fig. 6. The summation process for hidden layer neurons*

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

**10**

**11**

**12**

**13**

**14**

**Скрытый слой**

ные при обучении модели. Последнее позво- ляет значительно сократить время, затрачивае- мое моделью на обработку одного примера.

Основные элементы процедуры обучения изображены на диаграмме активности (рис. 7). Отметим, что некоторые этапы расчета гради- ентов выполняются параллельно. После обуче- ния модели на каждом примере производится расчет метрики кросс-энтропия. В данной мет- рике суммарная ошибка на выборке складыва- ется из ошибок по отдельным примерам, по- этому ее удобно использовать в задачах онлай- нового обучения для контроля качества работы модели. Если по какой-либо причине не уда- лось получить предсказание для данного при- мера, то и обучение также не производится.

# Сравнение скорости обработки примеров с другими реализациями

После разработки вышеописанной архитек- туры были проведены измерения для сравне- ния времени ее работы с реализациями, сделан- ными на базе уже имеющихся библиотек. В сравнениях участвовали четыре реализации.

1. Собственная реализация логистической регрессии с хешированием составных призна- ков [10]. Использовались 15 комбинаций при- знаков.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
| **F1** |
| **F2** |
| **F3** |
| **F4** |
| **F5** |



|  |  |
| --- | --- |
| **Получить предсказание модели**  Успешно  **Взять список выставленных индексов из predict**  **Рассчитать значение функции потерь**  **Выставить нулевое значение вероятности**  UNREGISTERED  **Обновить веса скрытого слоя**  **Обновить пороги скрытого слоя Обновить веса выходного слоя**  **Рассчитать значение метрик**  **Вернуть значение вероятности**  *Рис. 7. Диаграмма активности процедуры обучения модели по одному примеру Fig. 7. The activity diagram of model training using one sample* | |
| 1. Вышеописанная реализация нейронной сети с использованием простых признаков на входе. 2. Реализация предложенной архитектуры нейронной сети на базе библиотеки Lasagne с использованием разреженных матриц [12]. 3. Реализация предложенной архитектуры на базе Pytorch [13]. На момент проведения экс- периментов была доступна версия pytorch 0.4.1 (дата релиза – июль 2018 года) без поддержки автоматического расчета градиентов для разре- женных матриц. Поэтому реализация данной архитектуры была произведена на базе плот-   ных матриц. | В эксперименте производились измерения для разной размерности, отводимой на каждый признак. Количество коэффициентов, отводи- мых на признак, изменялось от 28 до 216. Всего использовалось 15 признаков. Таким образом, общий размер входного вектора изменялся от 4 тысяч до 1 млн признаков. Замерялось время получения предсказаний для 100 тысяч приме- ров и обновления модели на 100 тысяч при- меров (1 итерация). Замеры производились на сервере с процессором Intel Xeon CPU E5-2667  3.30 GHz и оперативной памятью Micron 36KSF2G72PZ-1 1333 MHz (0.8ns) объемом  256 Гб. |

**predict 1x2, 100K примеров update 1x2, 100K примеров**



100

90

80

70

60

**время, сек**

50

40

30

20

10

0

8 9 10 11 12 13 14 15 16

**Размерность признака, 2^x**

Реализация лог. регрессии x 10 Разработанная реализация x 10 Реализация на базе Lasagne

Реализация на базе Pytorch

100

90

80

70

60

**время, сек**

50

40

30

20

10

0

8 9 10 11 12 13 14 15 16

**Размерность признака, 2^x**

Реализация лог. регрессии x 10 Разработанная реализация x 10 Реализация на базе Lasagne

Реализация на базе Pytorch

*Рис. 8. Сравнение времени предсказания для 100 тысяч примеров*

*Fig. 8. Prediction time comparison for 100 thousands samples*

*Рис. 9. Сравнение времени обучения моделей для 100 тысяч примеров*

*Fig. 9. Training time comparison for 100 thousands samples*

Результаты измерений для предложенной архитектуры приведены на рисунках 8 и 9.

На графиках видно, что разработанная реа- лизация нейронной сети работает практически так же, как логистическая регрессия, и на поря- док быстрее реализации на базе Lasagne. Реа- лизация на базе Pytorch при увеличении раз- мера вектора начинает работать значительно медленнее, поскольку она основана на плот- ных, а не на разреженных матрицах. Разрабо- танная архитектура легко расширяема до под- держки не только пар, но и троек, четверок признаков и т.д. Однако в нейронной сети, в ко- торой каждый нейрон скрытого слоя связан с тремя нейронами на входном слое, время пред- сказания и обучения значительно возрастает.

# Заключение

В целом можно сделать вывод, что благо- даря описанным оптимизациям разработанная реализация позволяет получить ускорение при- мерно в 10 раз по сравнению с реализацией, написанной на уже готовых нейросетевых фреймворках. Таким образом, использование разработанной реализации целесообразно на практике, особенно в высоконагруженных си- стемах, работающих в режиме реального вре- мени, и только в том случае, когда каждый нейрон скрытого слоя связан с небольшим ко- личеством нейронов на входе. В противном случае предпочтительно использовать тради- ционные плотные матрицы.

***Литература***

* 1. Guyon I., Gunn S., Nikravesh M., Zadeh L. Feature Extraction: Foundations and Applications. Studies in Fuzziкness and Soft Computing. Springer, 2008, 773 p.
  2. Бенджио И., Гудфеллоу Я., Курвилль А. Глубокое обучение; [пер. с англ. А. Слинкина]. М.: ДМК-Пресс, 2018. 652 с.
  3. Fedorenko Yu.S., Gapanyuk Yu.E. The neural network with automatic feature selection for solving problems with categorical variables. Proc. XX Int. Conf. Neuroinformatics. Springer, 2018, vol. 799, pp. 129–135.
  4. Харрис С.Л., Харрис Д.М. Цифровая схемотехника и архитектура компьютера; [пер. с англ.]. М.: ДМК-Пресс, 2018. 792 с.
  5. Glorot X., Bordes A., Bengio Y. Deep Sparse Rectifier Neural Networks. Proc. Intern. Conf. PMLR, 2011, pp. 315–323.
  6. Molchanov D., Ashukha A., Vetrov D. Variational dropout sparsifies deep neural networks. Proc. 34th Intern. Conf. PMLR, 2017, pp. 2498–2507.
  7. LeCun Y., Denker J., Solla S. Optimal brain damage. Proc. Conf. NIPS, 1989, vol. 2, pp. 598–605.
  8. Srinivas S., Venkatesh B. Data-free parameter pruning for Deep Neural Networks. 2015. URL: https://arxiv.org/abs/1507.06149 (дата обращения: 11.05.2019).
  9. Bucila С., Caruana R., Niculescu-Mizil A. Model compression. Proc. ACM SIGKDD Intern. Conf., USA, 2006, pp. 535–541. DOI: 10.1145/1150402.1150464.
  10. ISOCPP: C++ 11 Overview. URL: https://isocpp.org/wiki/faq/cpp11 (дата обращения: 11.05.2019).
  11. Weinberger K., Dasgupta A., Langford J., Smola A., Attenberg J. Feature hashing for large scale mul- titask learning. Proc. 26th ICML, Canada, 2009, pp. 1113–1120. DOI: 10.1145/1553374.1553516.
  12. Lasagne: Docs, Welcome to Lasagne. URL: https://lasagne.readthedocs.io/en/latest/ (дата обращения: 11.05.2019).
  13. Pytorch. Tensors and Dynamic neural networks in Python with strong GPU acceleration. URL: <http://pytorch.org/>(дата обращения: 11.05.2019).

Software & Systems Received 19.06.19

DOI: 10.15827/0236-235X.128.639-649 2019, vol. 32, no. 4, pp. 639–649

**The development of fast software implementation of specialized neural network architecture with sparse connections**

***Yu.S. Fedorenko*** 1*, Postgraduate Student,* [*fedyura11235@mail.ru*](mailto:fedyura11235@mail.ru)

1 *Bauman Moscow State Technical University, Moscow, 105005, Russian Federation*

**Abstract.** The paper is devoted to the development of fast software implementation of a specialized neural network architecture. Feature engineering is one of the most important stages in solving machine learning tasks. Nowadays, the algorithms of handcrafted feature selection lose their popularity, giving way to deep neural networks. However, application of deep models is limited in online learning tasks as they are not able to learn in real time. Besides, their using is difficult in high-loaded systems due to significant computational complex- ity.

In the one of previous articles, the author has proposed a neural network architecture with automatic feature selection and the ability to train in real time. However, specific sparsity of connections in this architecture complicates its implementation on the base of classic deep learning frameworks. Therefore, we decided to do our own implementation of the proposed architecture.

This paper considers data structures and algorithms developed when writing software implementation. It describes sample processing in details from the program system point of view during model predicting and training. For a more complete description of implementation details, there are UML classes, sequences and activity diagrams.

The performance of the developed implementation is compared with implementations of same architecture on the base of deep learning frameworks. The analysis has shown that the developed software works an order of magnitude faster than library-based implementations. Such acceleration is due to the fact that the developed implementation is optimized for a specific architecture, while the frameworks are designed to work with a wide class of neural networks. In addition, the benchmarks have shown that the developed implementation of a proposed neural network works only 20-30 percent slower than a simple logistic regression model with hand- crafted features. Thus, it can be used in high loaded systems.

**Keywords:** feature engineering, deep neural networks, categorical features, sparse connections, modular arithmetic, index reuse, benchmark.

***References***

* + 1. Guyon I., Gunn S., Nikravesh M., Zadeh L. *Feature Extraction: Foundations and Applications. Studies in Fuzziness and Soft Computing*. Springer, Berlin Heidelberg, 2008, 773 p.
    2. Bengio Y., Goodfellow I., Courville A. *Deep Learning*. MIT Press, 2016, 800 p. (Russ. ed.: A. Slinkin, Moscow, DMK-Press, 2018, 652 p.).
    3. Fedorenko Yu.S., Gapanyuk Yu.E. The Neural Network with Automatic Feature Selection for Solving Problems with Categorical Variables. *Proc. XX Int. Conf. Neuroinformatics*. Springer Publ., 2018, vol. 799, pp. 129–135.
    4. Harris S.L., Harris D.M. *Digital Design and Computer Architecture*. 2nd ed., Moscow, DMK-Press, 2018, 792 p.
    5. Glorot X., Bordes A., Bengio Y. Deep Sparse Rectifier Neural Networks. *Proc. Intern. Conf. PMLR*. 2011, pp. 315–323.
    6. Molchanov D., Ashukha A., Vetrov D. Variational Dropout Sparsifies Deep Neural Networks. *Proc. 34th Intern. Conf. on Machine Learning, PMLR*. 2017, pp. 2498–2507.
    7. LeCun Y., Denker J., Solla S. Optimal brain damage*. Proc. Conf. NIPS*, 1989, vol. 2, pp. 598–605.
    8. Srinivas S., Venkatesh B. *Data-Free Parameter Pruning for Deep Neural Networks*. 2015. Available at: <https://arxiv.org/abs/1507.06149>(accessed May 11, 2019).
    9. Bucila С., Caruana R., Niculescu-Mizil A. Model compression. *Proc. 12th ACM SIGKDD Intern. Conf. on Knowledge Discovery and Data Mining*. USA, 2006, pp. 535–541. DOI: 10.1145/1150402.1150464.
    10. *ISOCPP: C++ 11 Overview*. Available at: <https://isocpp.org/wiki/faq/cpp11>(accessed May 11, 2019).
    11. Weinberger K., Dasgupta A., Langford J., Smola A., Attenberg J. Feature hashing for large scale mul- titask learning. *Proc. 26th Intern. Conf. on Machine Learning, ICML*. Canada, 2009, pp. 1113–1120. DOI: 10.1145/1553374.1553516.
    12. *Lasagne: Docs, Welcome to Lasagne*. Available at: <https://lasagne.readthedocs.io/en/latest/>(accessed May 11, 2019).
    13. *Pytorch. Tensors and Dynamic Neural Networks in Python with Strong GPU Acceleration*. Available at: <http://pytorch.org/>(accessed May 11, 2019).

**Для цитирования**

Федоренко Ю.С. Проектирование быстрой программной реализации специализированной нейросетевой архитектуры с разреженными связями // Программные продукты и системы. 2019. Т. 32. № 4. С. 639–649. DOI: 10.15827/0236-235X.128.639-649.

**For citation**

Fedorenko Yu.S. The development of fast software implementation of specialized neural network architecture with sparse connections. *Software & Systems*. 2019, vol. 32, no. 4, pp. 639–649 (in Russ.). DOI: 10.15827/0236-235X.128.639-649.