# Инициализация полигармонического каскада, запуск и проверка

Бахвалов Ю.Н., к.т.н., ООО “ЛИС”, г. Череповец

**Аннотация.**

Статья посвящена вопросам, связанным с начальной инициализацией полигармонического каскада (модели машинного обучения с данной архитектурой), а также его запуску и первичному тестированию. Предложен один из вариантов выбора точек созвездий, который может подойти не только для универсальной их инициализации, но и для оптимизации работы каскада в целом, значительно упростив часть вычислений. Рассмотрен вопрос выбора начальных значений полигармонических функций в точках созвездий. Предложен вариант инициализации, позволяющий при необходимости неограниченно увеличить количество пакетов в каскаде, обеспечивая их работоспособность. Практически проверена работоспособность каскада с увеличением количества содержащихся в нём пакетов, проведено тестирование на датасетах MNIST, Higggs Boson и Epsilon.

**Ключевые слова:** машинное обучение, полигармонический сплайн, пакет полигармонических сплайнов, полигармонический каскад.

## Initialization of a polyharmonic cascade, launch and testing

Bakhvalov Yu.N., Ph.D., LLC "LIS", Cherepovets

**Abstract.**

The article is devoted to issues related to the initial initialization of a polyharmonic cascade (machine learning models with this architecture), as well as its launch and primary testing. One of the options for choosing constellation points is proposed, which can be suitable not only for their universal initialization, but also for optimizing the operation of the cascade as a whole, significantly simplifying some of the calculations. The issue of choosing the initial values of polyharmonic functions at constellation points is considered. An initialization option is proposed that allows, if necessary, to almost unlimitedly increase the number of packages in the cascade, ensuring their operability. The operability of the cascade with an increase in the number of packages contained in it was practically verified, testing was carried out on the MNIST, Higgs Boson and Epsilon datasets.

**Keywords**: machine learning, polyharmonic spline, polyharmonic spline package, polyharmonic cascade.

Данная статья посвящена вопросам начальной инициализации полигармонического каскада, архитектуры машинного обучения, теоретические основы которой были изложены в [1], [2] и [3]. Также в статье рассмотрены результаты проведенных с ним тестов.

В работе [1] показано как регрессионная задача машинного обучения может быть решена на основе теории случайных функций [4]. Полученное решение представляет собой разновидность полигармонического сплайна специального вида (сплайн тонкой пластины) [7] и [8]. Была установлена теоретическая связь полигармонического сплайна с теорией случайных функций при решении регрессионной задачи машинного обучения. Получена математическая конструкция для универсального решения задачи многомерной аппроксимации.

Работа [2] посвящена масштабированию решения, полученного в [1]. Показано, что одна и та же такая математическая конструкция хорошо подходит для описания и вычисления сразу большого количества функций. Но в этом случае эти функции должны быть заданы через набор значений в одних и тех же ключевых точках. Набор специально определенных таких точек был назван “созвездием”, а множество функций, описываемых таким механизмом, было названо “пакетом” полигармонических сплайнов. Также было обосновано объединение таких пакетов в виде каскада в более сложную вычислительную систему. Предложены эффективные процедуры её вычисления и дифференцирования в виде матричных операций.

В статье [3] предложен метод обучения полигармонического каскада (выводится как решение оптимизационной задачи). Метод не является градиентным спуском, хотя и требует выполнения процедуры дифференцирования по цепному правилу, двигаясь через пакеты в каскаде в обратном порядке, как один из этапов. В итоге, в статье [3] описывается полигармонический каскад и принципы его работы (как архитектура машинного обучения) в виде полноценного законченного алгоритма.

Однако, за скобками рассмотренных работ осталась процедура начальной инициализации полигармонического каскада. С практической точки зрения этот вопрос чрезвычайно важен. Кроме того, выбор созвездий имеет значение и с точки зрения теории (этот выбор может упростить исполняемые операции, значительно изменив вычислительную нагрузку, что будет показано ниже).

Чтобы рассмотреть процедуру инициализации полигармонического каскада, сначала кратко опишем работу самого каскада и процедуру его обучения. Обоснование всех этих действий и операций были выполнены в работах [2] и [3], здесь же приведем их без обоснования, какие они были получены в окончательном виде.

Заимствуем рисунок схемы работы каскада из работы [3].

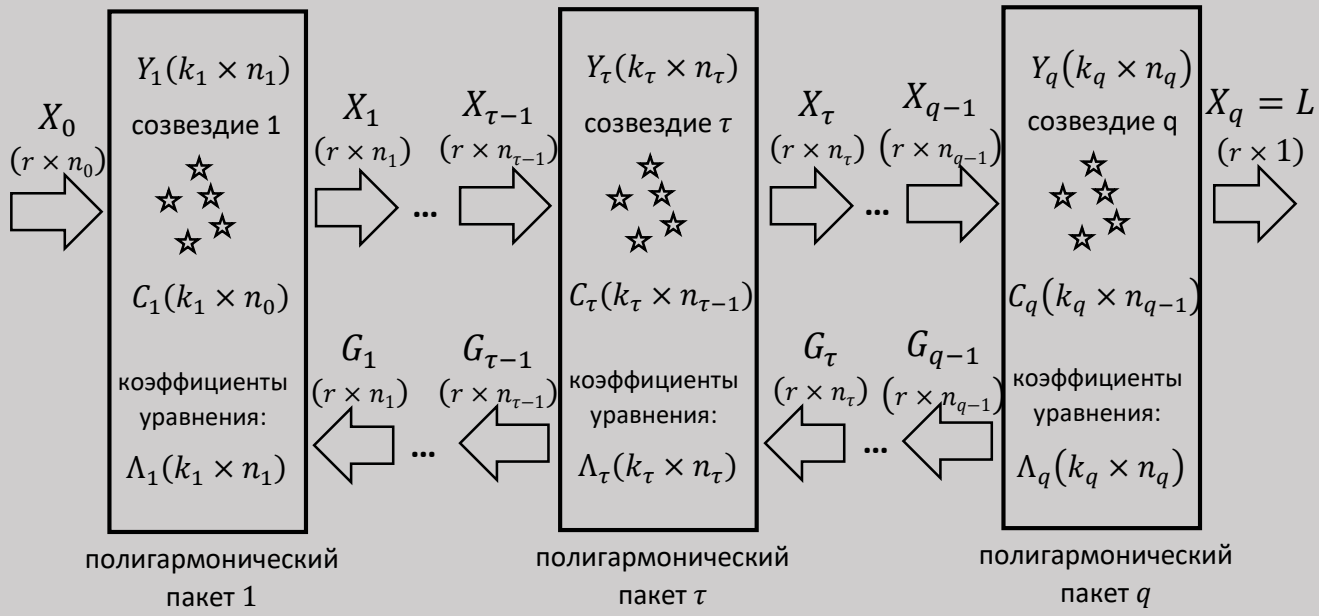


Рисунок 1. Полигармонический каскад

На рисунке 1 изображен полигармонический каскад, состоящий из последовательных пакетов (в количестве $q$). Непосредственно на схеме изображен первый и последний пакеты, а также некоторый промежуточный пакет с номером $\tau$. Количество входов и выходов пакетов задано с помощью значений $n_0, n_1, \dots, n_{q-1}$, а количество точек в созвездиях с помощью значений $k_1, k_2, \dots, k_q$. Количество обрабатываемых векторов в одном батче на схеме обозначено как $r$ (размер батча).

Обработка данных каскадом выглядит следующим образом.

На вход первого пакета приходит батч данных на обработку в виде матрицы $X_0$ размерностью $(r \times n_0)$. В результате обработки $X_0$ на выходе первого пакета будет получена матрица $X_1$ размерностью $(r \times n_1)$, которая поступит на вход второго пакета, где затем будет получена матрица $X_2$ и т.д. На выходе каскада будет получена матрица $X_q$ размерностью $(r \times 1)$, которая в [2] и [3] также еще обозначается как вектор $L$.

Как видно по размерности $L$ $(r \times 1)$ , рассматривается вариант, когда каскад имеет лишь один выход (последний из полигармонических пакетов вычисляет только одну функцию). В работе [3] был сначала подробно разобран именно такой случай, а затем представлены варианты, как его можно трансформировать в каскад с несколькими выходами. В текущей статье, чтобы рассмотреть вопросы инициализации каскада, достаточно разбора каскада с одним выходом (вычисляющий одну функцию). Трансформация каскада до варианта с несколькими выходами в данном контексте принципиально ничего

не меняет и может быть выполнена теми же способами, которые описаны в статье [3].

Каждый полигармонический пакет в каскаде описывается с помощью трех матриц. Для полигармонического пакета с номером $\tau$ это будут матрицы $C_\tau, Y_\tau$ и $\Lambda_\tau$. Матрица $C_\tau$ описывает точки созвездия, матрица $Y_\tau$ задает значения полигармонических функций (входящих в пакет) в этих точках. А матрица $\Lambda_\tau$ содержит коэффициенты уравнений, которые непосредственно используются при вычислениях (но матрица $\Lambda_\tau$ может быть найдена по известным $C_\tau$ и $Y_\tau$).

Чтобы выполнить этап обучения, нужно после обработки батча двигаться по каскаду в обратном направлении и вычислить последовательно матрицы производных от $G_{q-1}$ до $G_1$. Затем для каждого из пакетов независимо может быть вычислена соответствующая ему матрица $\Omega$. Все матрицы $\Omega_1, \Omega_2, \dots, \Omega_q$ складываются (они получатся одинаковой размерности $(r \times r)$ для всех пакетов). Решается система уравнений и находится вектор $B$ размером $r$, который затем может быть применен к каждому пакету независимо, чтобы пересчитать матрицу значений $Y_\tau$, а в соответствии с ней можно обновить матрицу коэффициентов $\Lambda_\tau$.

Опишем теперь все перечисленные действия более подробно и в виде операций над матрицами, которые были получены в работах [2] и [3].

Сначала покажем процедуру вычисления некоторым произвольным полигармоническим пакетом с номером $\tau$ матрицы $X_\tau$ из входной матрицы $X_{\tau-1}$.

Первым шагом находим матрицу квадратов всевозможных расстояний между строчками векторами в $X_{\tau-1}$ и точками созвездия. Обозначим эту матрицу как $M_\tau$.

$$M_\tau = N_{\tau x} J_{1,k_\tau} + J_{r,1} N_{\tau c}^T - 2 X_{\tau-1} C_\tau^T \ , \qquad (1)$$

где

$$N_{\tau x} = (X_{\tau-1} \circ X_{\tau-1}) J_{n_{\tau-1},1} \ ,$$

$$N_{\tau c} = (C_\tau \circ C_\tau) J_{n_{\tau-1},1} \ ,$$

$\circ$ - произведение Адамара,

$J_{1,k_\tau}$ – вектор строка из единиц размера $k_\tau$,

$J_{r,1}$ – вектор столбец из единиц размера $r$,

$J_{n_{\tau-1},1}$ - вектор столбец из единиц размера $n_{\tau-1}$,

$C_\tau$ – матрица точек созвездия $\tau$ – *го* пакета.

Затем из матрицы $M_\tau$ получаем матрицу $K_\tau$ той же размерности $(r \times k_\tau)$ путем преобразования всех её элементов:

$$k_{ip}^{\tau} = \frac{1}{2} m_{ip}^{\tau}\left(\ln\left(m_{ip}^{\tau}\right) - 2b\right) + c\ , \tag{2}$$

где

$b$ и $c$ – коэффициенты, значения которых оценивались в [1],

$k_{ip}^{\tau}$ – элемент матрицы $K_\tau$ (не путать с $k_\tau$),

$m_{ip}^{\tau}$ – элемент матрицы $M_\tau$,

$$i = \overline{1, r}\ , p = \overline{1, k_\tau}\ .$$

Далее находим $X_\tau$ как матричное произведение $K_\tau$ и $\Lambda_\tau$ :

$$X_\tau = K_\tau \Lambda_\tau \tag{3}$$

Таким образом, обработка данных любым пакетом заключается в выполнении операций (1), (2) и (3). Как видно, здесь нигде не используется матрица $Y_\tau$ (содержащая значения функций в точках созвездия). Однако именно её значения являются предметом оптимизации алгоритма обучения, разобранного в [3].

Но, зная матрицу $Y_\tau$, вычислить $\Lambda_\tau$ достаточно легко:

$$\Lambda_\tau = U_\tau Y_\tau \tag{4}$$

Где $U_\tau$ представляет собой квадратную матрицу, полностью определяемую заданным в пакете созвездием (размерность которой $(k_\tau \times k_\tau)$ определяется количеством точек в созвездии). Если созвездие (как принято в алгоритме обучения, разобранном в [3]) фиксировано и при обучении остается неизменным, то и $U_\tau$ требует лишь однократного вычисления при инициализации каскада.

Опишем операции вычисления $U_\tau$ по известной матрице созвездия $C_\tau$.

Сначала найдем матрицу всех возможных квадратов расстояний между точками созвездия по отношению друг к другу, которую обозначим как $M_\tau^{(C)}$.

$$M_\tau^{(C)} = N_{\tau c} J_{1,k_\tau} + J_{k_\tau,1} N_{\tau c}^T - 2 C_\tau C_\tau^T\ , \tag{5}$$

где

$$N_{\tau c} = (C_\tau \circ C_\tau) J_{n_{\tau-1},1},$$

$J_{1,k_\tau}$ – вектор строка из единиц размера $k_\tau$,

$J_{k_\tau,1}$ – вектор столбец из единиц размера $k_\tau$.

Тогда матрица $U_\tau$ будет:

$$U_\tau = \left(K_\tau^{(C)} + \sigma_\tau^2 E\right)^{-1}, \quad (6)$$

где

матрица $K_\tau^{(C)}$ вычисляется из $M_\tau^{(C)}$ с помощью (2),

$\sigma_\tau^2$ – дисперсии случайных величин,

(более подробно их смысл разобран в [1])

E – единичная матрица.

Смысл $\sigma_\tau^2$ был более подробно разобран в [1]. В рамках полигармонического каскада $\sigma_\tau^2$ может быть взято равным нулю (или значением близким к нулю, если нужно избежать получения сингулярной матрицы при вычислении (6), что впрочем соответствует какому-то очень неудачному выбору созвездия, когда некоторые его точки близки друг к другу или совпадают).

Если каскад выполняет обработку батча не только для того чтобы получить значения на выходе, а как этап процедуры обучения, то $U_\tau$ используется также для вычисления промежуточной матрицы $H_\tau$, которая потребуется в дальнейшем ($K_\tau$ в (7) это матрица, найденная в (2)).

$$H_\tau = K_\tau U_\tau \quad (7)$$

Теперь разберем процедуру пересчета производных. Как по известной матрице $G_\tau$ вычислить $G_{\tau-1}$.

Сначала вычисляется матрица $\Theta_\tau$. Получается она путем поэлементного преобразования из матрицы $M_\tau$ которая была найдена в (1) с помощью выражения:

$$\theta_{ip}^\tau = \ln\left(m_{ip}^\tau\right) - 2b + 1, \quad (8)$$

где

$b$ – тот же коэффициент из (4),

$\theta_{ip}^\tau$ – элемент матрицы $\Theta_\tau$ ,

$m_{ip}^\tau$ – элемент матрицы $M_\tau$,

$i = \overline{1,r}$ , $p = \overline{1,k_\tau}$ .

Затем находится матрица $\Psi_\tau$:

$$\Psi_\tau = \Theta_\tau \circ (G_\tau \Lambda_\tau^T) \quad (9)$$

Матрицы $\Theta_\tau$ и $\Psi_\tau$ получатся размером $(r \times k_\tau)$.

И теперь можно найти $G_{\tau-1}$:

$$G_{\tau-1} = X_{\tau-1} \circ \left(\Psi_\tau J_{k_\tau,1} J_{1,n_{\tau-1}}\right) - \Psi_\tau C_\tau \,, \quad (10)$$

где

$J_{k_\tau,1}$ – вектор столбец из единиц размера $k_\tau$,

$J_{1,n_{\tau-1}}$ - вектор строка из единиц размера $n_{\tau-1}$.

После того, как для всех полигармонических пакетов в каскаде были получены матрицы $H_1, H_2, \dots, H_q$ из выражения (7) и матрицы $G_1, G_2, \dots, G_q$ из выражения (10) (кроме матрицы $G_q$, которая не вычисляется, а просто берется состоящей из единиц, размерностью $(r \times 1)$), вычисляется набор матриц $\Omega_1, \Omega_2, \dots, \Omega_q$.

$$\Omega_\tau = (H_\tau H_\tau^T) \circ (G_\tau G_\tau^T) \quad (11)$$

Как видно, выражение (11) может быть выполнено для любого пакета независимо (вычисляться параллельно). Здесь нет никакой привязки к последовательности, как они расположены друг за другом.

Затем матрицы $\Omega_1, \Omega_2, \dots, \Omega_q$ складываются и вычисляется вектор *B*.

$$\mathrm{B} = \left(\Omega_1 + \Omega_2 + \cdots + \Omega_q + \alpha E\right)^{-1} \Delta L \quad (12)$$

где $\Delta L = L^* - L$,

$L$ – вектор значений на выходе каскада,

$L^*$ - вектор желаемых значений на выходе каскада,

$\alpha$ – коэффициент, регулирующий точность решения системы уравнений, фактически регулирующий скорость обучения (чем его значение больше, тем скорость обучения медленнее)

Найти вектор *B* в (12) можно через вычисление обратной матрицы, либо решив систему уравнений. После чего можно найти изменение матриц значений полигармонических функций в точках созвездий $Y_\tau$ для любого из пакетов:

$$\Delta Y_\tau = H_\tau^T \left( G_\tau \circ \left( \mathrm{B} J_{1,n_\tau} \right) \right), \qquad (13)$$

где

$J_{1,n_\tau}$ – вектор строка из единиц размера $n_\tau$

А затем через (4) можно найти матрицы коэффициентов уравнений $\Lambda_\tau$.

Операции (13) и (4) так же, как и (11) для каждого пакета могут быть выполнены независимо (вычисляться параллельно).

Таким образом, выражения (1) – (13) описывают как работу полигармонического каскада, так и предложенный в работе [3] метод его обучения. Но что оказалось не разобранным в рамках такого описания и относится к начальной инициализации?

Если рассмотреть, что из себя представляет начальная инициализация полигармонического каскада, то её можно разделить на три составляющих.

Первая составляющая инициализации, это структура каскада, то как устроена последовательность пакетов, из которых он состоит, сколько входов и сколько выходов у каждого из пакетов (что в рассмотренной схеме описывается набором значений $n_0, n_1, \dots, n_{q-1}$). Если пакеты идут последовательно друг за другом, то очевидно, что количество выходов предыдущего пакета должно совпадать с количеством входов следующего. Именно такой вариант каскада рассматривался в работе [3] при выводе алгоритма обучения. Однако полученные в итоге выражения (1) – (13) носят локальный характер и относятся к операциям каждого из пакетов по отдельности (за исключением выражения (12), которое наоборот, относится ко всему каскаду целиком). Следовательно, этот же алгоритм обучения может быть применен для любого однонаправленного полигармонического каскада произвольной структуры. Теоретически каскад может разделяться на ветки (количество пакетов в которых может быть различным), которые затем могут объединяться снова и т.д. Однако в рамках данной работы такие варианты не изучались.

Вторая составляющая инициализации, это задание для каждого из пакетов своего созвездия, которое представляет из себя набор ключевых точек, по значениям в которых будут строиться все функции внутри пакета. В описанной схеме работы каскада (в [2] и [3]) созвездия представлены в виде матриц $C_\tau$, как будто они уже были известны или кем-то заданы заранее. Очевидно, что размерность пространства, в котором должны быть заданы точки созвездия должна совпадать с количеством входов пакета (размерностью функций, которые пакет вычисляет). Число же самих точек внутри созвездия никак теоретически не ограничено и не связано с

количеством выходов в пакете. Количество точек в созвездиях для последовательности пакетов на схеме обозначено как набор значений $k_1, k_2, \dots, k_q$. Соответственно, размерность матриц $C_\tau$ равна $(k_\tau \times n_{\tau-1})$. Созвездия, это один из ключевых элементов в работе полигармонических пакетов. Очевидно, что формирование матриц созвездий $C_\tau$, задание их значений, должно быть крайне важным этапом в начальной инициализации каскада.

Также очевидно, что точки созвездия должны быть приближенно расположены в той же области пространства, в которой будут находится вектора значений, поступающих на вход пакета с выходов предыдущего пакета или из обучающей выборки, если пакет в каскаде первый (в данном случае значения из обучающей выборки могут быть каким-либо образом принудительно нормированы).

Третьей составляющей при инициализации будем считать заданные начальные значения полигармонических функций в точках созвездий. Т.е. в разобранной схеме это будут начальные значения матриц $Y_\tau$. Непосредственные коэффициенты уравнений вычисляемых в пакетах функций (матрицы $\Lambda_\tau$) так же, как и в работе [3] будем считать вторичными, поскольку они определяются через точки созвездий и заданные значения в них (выражение (4)). Размерность матриц $Y_\tau$ равна $(k_\tau \times n_\tau)$ т.к. определяется количеством точек в созвездии и количеством выходов в пакете (количеством вычисляемых пакетом функций).

Разберем теперь более подробно составляющие инициализации полигармонического каскада. В соответствии с предложенным в [3] алгоритмом работы и обучения каскада, первые две составляющие, структура и созвездия задаются на этапе инициализации, после чего остаются фиксированными (в процессе обучения выполняется только настройка матриц значений в точках созвездий $Y_\tau$). Такая фиксация (как показали вычислительные эксперименты) не ограничивает способность каскада обучаться. Однако вопрос о возможности настройки не только значений в точках созвездий, но и самих точек в процессе обучения, остался не исследованным и требует дальнейшего изучения.

К выбору структуры каскада и количеству входящих в него пакетов можно подойти из тех же принципов, как при выборе количества слоёв в многослойных нейронных сетях. Чем больше пакетов в каскаде и чем большее количество промежуточных функций они вычисляют, тем более сложную нелинейную функцию каскад способен воспроизвести в целом.

Хотя теоретически даже один полигармонический пакет способен воспроизвести функцию неограниченной сложности, это будет подразумевать

неограниченное увеличение точек в его созвездии. В работе [2] были разобраны недостатки использования только одного пакета по сравнению с их каскадным соединением. Поэтому возможно лучшим вариантом, если требуется нарастить сложность каскада (количество настраиваемых параметров), предполагается ограничивать размеры созвездий в пакетах, но увеличивать количество вычисляемых в них функций (выходов пакетов) и количество самих пакетов (слоёв каскада).

Ключевыми элементами как при инициализации, так и в процессе функционирования полигармонического каскада являются созвездия. Однако способов, каким образом их можно было бы задать, может быть придумано большое количество.

Ниже будет описан один из предлагаемых вариантов выбора созвездий, процедура, которая с одной стороны обладает универсальностью и позволяет легко задать созвездие для любого пакета, с другой стороны позволяет упростить некоторые из выполняемых операций и снизить вычислительную нагрузку. Также эта процедура была испытана на практике и продемонстрировала, что инициализируемые с её помощью полигармонические каскады работоспособны и могут обучаться.

Однако, предлагаемый ниже алгоритм, это лишь один из работоспособных универсальных вариантов инициализации. Не исключено, что могут существовать другие алгоритмы формирования созвездий, которые могут дать преимущество перед предложенным алгоритмом при решении ряда задач или в целом. Формирование созвездий может быть темой дальнейших исследований.

**Формирование созвездий, предлагаемый вариант.**

Какие качества можно было бы ожидать от универсальной процедуры формирования созвездий? Во-первых, она должна подходить для описания созвездия в пространстве любой размерности (в соответствии с количеством входов в пакете). Во-вторых, можно предположить (как гипотезу), что наилучшим вариантом (поскольку заранее неизвестно какие функции пакет должен вычислять) будет случай, когда точки созвездия в этом многомерном пространстве будут расположены некоторым образом симметрично. Если брать в расчет эти соображения, то кандидатами на роль созвездий (которые в первую очередь требуют проверки), будут такие расположения точек, которые образуют вершины правильных многомерных многогранников. Одним из относительно простых в плане формирования и использования является гипероктаэдр, хотя и варианты с вершинами других типов многогранников могут быть кандидатами.

В текущей работе, в рамках вычислительных экспериментов был реализован алгоритм автоматического формирования созвездий при инициализации каскада в виде первой начальной точки созвездия с нулевыми координатами и вершинами гипероктаэдров вокруг неё на единичном расстоянии.

На рисунке 2 проиллюстрированы варианты формирования созвездий от одномерного до четырехмерного пространства. Далее по такому же принципу можно задать созвездие для пространства любой размерности. Если размерность пространства равна $n$, то созвездие будет содержать $2n + 1$ точек.

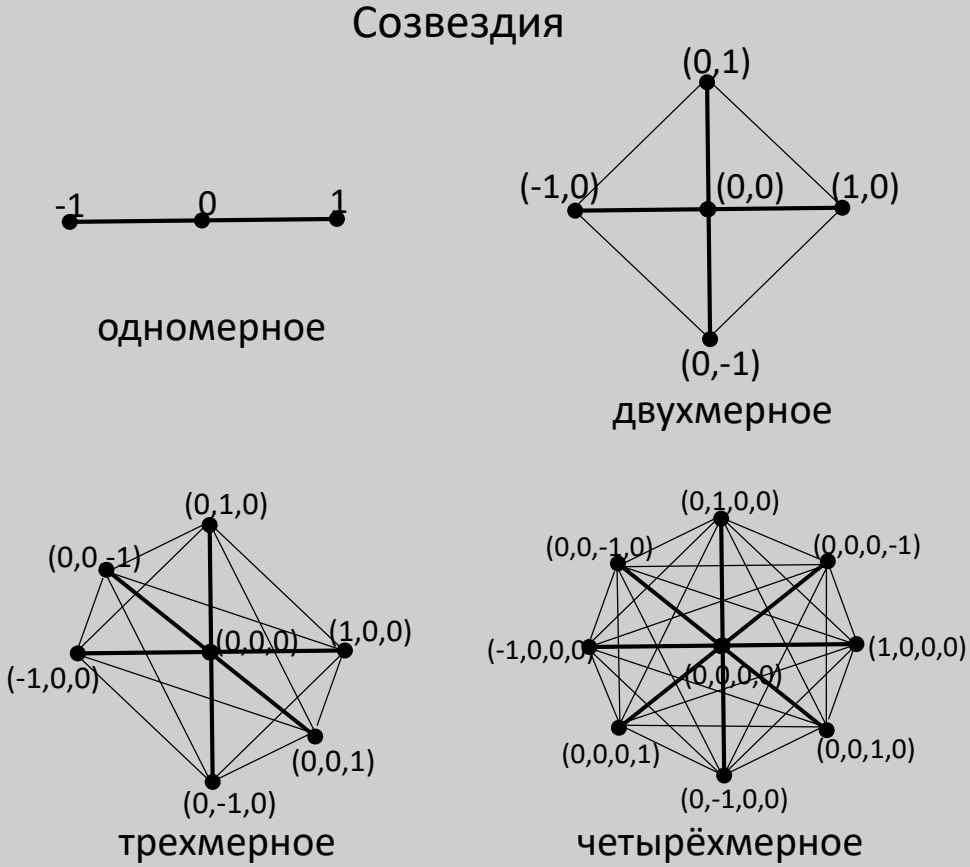


Рисунок 2. Принципы формирования созвездий на основе гипероктаэдров.

На рассмотренной выше схеме (рисунок 1) количество входов для пакета с номером $\tau$ было обозначено как $n_{\tau-1}$. Значит количество точек, которое обозначается как $k_\tau$, в созвездии (по предлагаемой схеме) будет равно:

$$k_\tau = 2n_{\tau-1} + 1 \qquad (14)$$

Однако в дальнейших рассуждениях, рассмотрим выражения, относящиеся только к одному, некоторому пакету с номером $\tau$ (и соответственно они подойдут к пакету с любым номером), и чтобы не

усложнять выражения обозначим количество входов этого пакета просто как $n$ (фактически считая $n = n_{\tau-1}$).

Опишем теперь, как будет выглядеть такое созвездие в виде матрицы $C_\tau$.

$$C_\tau = \begin{bmatrix} 0_{1\times n} \\ -E_n \\ E_n \end{bmatrix}, \quad (15)$$

где

$E_n$ – единичная матрица размерностью $(n \times n)$,

$0_{1\times n}$ – строка, состоящая из нулей размером $n$ .

Очевидно, чтобы полигармонический каскад с заданными таким образом созвездиями мог полноценно работать, значения, которые поступают на вход всех пакетов должны приблизительно находится не слишком далеко от диапазона $(-1{,}1)$. Выполнить это требование для первого пакета в каскаде можно выполнив нормировку входных данных. Также можно подобрать значения полигармонических функций в точках созвездий (матрицы $Y_\tau$) при инициализации пакетов, чтобы в начале процедуры обучения каскад был работоспособен.

Возникает вопрос, не произойдет ли где-то в процессе обучения выход за диапазон расположения созвездий (которые фиксированы) значений с предыдущих пакетов в каскаде? Нигде в алгоритме обучения из работы [3] такая опасность напрямую не контролируется, значения на выходе пакетов никак не ограничены. Однако, как показали вычислительные эксперименты, полигармонический каскад (при правильном подборе коэффициента $\alpha$ из (12) и разобранной далее в текущей статье процедуре инициализации) вне зависимости от количества пакетов ведет себя устойчиво. Обучение по алгоритму из [3] заставляет настраиваемые полигармонические функции из разных пакетов вести себя согласованно друг с другом как единое целое.

Рассмотрим теперь более подробно как выбор созвездия, основанного на гипероктаэдре в виде (15) может отразиться на исполняемых операциях (1) – (13).

В первую очередь значительно упростится выражение (1), если в него вместо $C_\tau$ подставить выражение (15):

$$M_\tau = N_{\tau x} J_{1,k_\tau} + [0_{r\times 1} \quad 1 + 2X_{\tau-1} \quad 1 - 2X_{\tau-1}], \quad (16)$$

где

$$N_{\tau x} = (X_{\tau-1} \circ X_{\tau-1}) J_{n,1},$$

$0_{r\times 1}$ – вектор столбец из нулей размера $r$,

$J_{1,k_\tau}$ – вектор строка из единиц размера $k_\tau$,

$J_{n,1}$ - вектор столбец из единиц размера $n$.

Наиболее сложным в вычислительном плане в (1) было матричное произведение $X_{\tau-1}C_\tau^T$. Как видно, при использовании созвездия (15), выражение (1) преобразовавшись в (16) радикально упростилось.

Выражение (5) также перетерпит существенные изменения.

$$M_\tau^{(C)} = N_{\tau c}J_{1,k_\tau} + J_{k_\tau,1}N_{\tau c}^T - 2C_\tau C_\tau^T =$$

$$= \begin{bmatrix}0_{1\times 2n+1}\\ J_{2n,2n+1}\end{bmatrix} + [0_{2n+1\times 1} \quad J_{2n+1,2n}] - 2\begin{bmatrix}0 & 0_{1\times n} & 0_{1\times n}\\ 0_{n\times 1} & E_n & -E_n\\ 0_{n\times 1} & -E_n & E_n\end{bmatrix}, \qquad (17)$$

где

$0_{1\times 2n+1}$ – строка из нулей размером $2n$+1,

$J_{2n,2n+1}$ – матрица из единиц размерностью $(2n \times 2n+1)$,

$0_{2n+1\times 1}$ – столбец из нулей размером $2n$+1,

$J_{2n+1,2n}$ – матрица из единиц размерностью $(2n+1 \times 2n)$,

$0_{1\times n}$ – строка из нулей размера $n$,

$0_{n\times 1}$ – столбец из нулей размера $n$.

Обозначим через $P_{2n}$ матрицу:

$$P_{2n} = \begin{bmatrix}0_{n\times n} & E_n\\ E_n & 0_{n\times n}\end{bmatrix} \qquad (18)$$

Тогда:

$$M_\tau^{(C)} = \begin{bmatrix}0 & J_{1,2n}\\ J_{2n,1} & 2J_{2n}\end{bmatrix} - 2\begin{bmatrix}0 & 0_{1\times 2n}\\ 0_{2n\times 1} & E_{2n} - P_{2n}\end{bmatrix}, \qquad (19)$$

где $E_{2n}$ – единичная матрица размерностью $(2n \times 2n)$,

$J_{2n}$ – матрица из единиц размерностью $(2n \times 2n)$.

В итоге:

$$M_\tau^{(C)} = \begin{bmatrix}0 & J_{1,2n}\\ J_{2n,1} & 2J_{2n,2n} - 2E_{2n} + 2P_{2n}\end{bmatrix} \qquad (20)$$

Посмотрим, что получилось в (20).

По главной диагонали (т.е. в первом элементе первой строки и в тех элементах, где $E_{2n}$ из (20) принимает значение равное 1) $M_\tau^{(C)}$ равна 0.

В первой строке и в первом столбце из (20), кроме первого элемента главной диагонали, $M_\tau^{(C)}$ принимает значение равное 1.

В тех элементах, где $P_{2n}$ из (20) принимает значение равное единице $M_\tau^{(C)}$ принимает значение равное 4.

Все остальные элементы матрицы $M_\tau^{(C)}$ равны 2.

Таким образом, элементы матрицы (20) могут принимать только четыре варианта значений – 0, 1, 2, 4. Воспользуемся этим при вычислении матрицы $K_\tau^{(C)}$ (которая вычисляется из $M_\tau^{(C)}$ через (2)). Обозначим через $k_0, k_1, k_2, k_4$ значения функции (2) от значений 0, 1, 2, 4.

$$k_0 = \lim_{x\to 0}(0.5x(\ln(x) - 2b) + c) = \mathrm{c} \tag{21}$$

$$k_1 = 0.5(\ln(1) - 2b) + c = \mathrm{c} - b \tag{22}$$

$$k_2 = (\ln(2) - 2b) + c \tag{23}$$

$$k_4 = 2(\ln(4) - 2b) + c \tag{24}$$

Выразим теперь $K_\tau^{(C)}$ используя (20) – (24).

$$K_\tau^{(C)} = \begin{bmatrix} k_0 & k_1 J_{1,2n} \\ k_1 J_{2n,1} & (k_0 - k_2)E_{2n} + (k_4 - k_2)P_{2n} + k_2 J_{2n} \end{bmatrix}, \tag{25}$$

Используя (25) теперь можно вычислить $U_\tau$ в (6). Но сначала запишем как будет выглядеть выражение $K_\tau^{(C)} + \sigma_\tau^2 E$, обратную матрицу от которого необходимо будет найти.

$$K_\tau^{(C)} + \sigma_\tau^2 E = \begin{bmatrix} k_0 + \sigma_\tau^2 & k_1 J_{1,2n} \\ k_1 J_{2n,1} & (k_0 - k_2 + \sigma_\tau^2)E_{2n} + (k_4 - k_2)P_{2n} + k_2 J_{2n} \end{bmatrix} \tag{26}$$

Значение $\sigma_\tau^2$ для созвездия (15) можно взять равное нулю.

Чтобы вычислить $U_\tau$ найдем обратную матрицу от (26). Поскольку это блочная матрица из 4-х блоков, воспользуемся формулой Фробениуса, которую можно представить так:

$$\begin{bmatrix} K_1 & K_2 \\ K_3 & K_4 \end{bmatrix}^{-1} = \begin{bmatrix} K_1^{-1} + K_1^{-1}K_2 S^{-1} K_3 K_1^{-1} & -K_1^{-1}K_2 S^{-1} \\ -S^{-1}K_3 K_1^{-1} & S^{-1} \end{bmatrix},$$

где $S = K_4 - K_3 K_1^{-1} K_2$

Для выражения (26) блоки $K_1, K_2, K_3, S$ будут:

$$K_1 = k_0 + \sigma_\tau^2$$

$$K_2 = k_1 J_{1,2n}$$

$$K_3 = k_1 J_{2n,1}$$

$$S = (k_0 - k_2 + \sigma_\tau^2)E_{2n} + (k_4 - k_2)P_{2n} + k_2 J_{2n} - k_1 J_{2n,1} \frac{1}{k_0 + \sigma_\tau^2} k_1 J_{1,2n} =$$

$$= (k_0 - k_2 + \sigma_\tau^2)E_{2n} + (k_4 - k_2)P_{2n} + \left(k_2 - \frac{{k_1}^2}{k_0 + \sigma_\tau^2}\right) J_{2n} \quad (27)$$

Введем коэффициенты:

$$a_1 = k_0 - k_2 + \sigma_\tau^2 \quad (28)$$

$$a_2 = k_4 - k_2 \quad (29)$$

$$a_3 = k_2 - \frac{{k_1}^2}{k_0 + \sigma_\tau^2} \quad (30)$$

Тогда $S$ можно записать как:

$$S = a_1 E_{2n} + a_2 P_{2n} + a_3 J_{2n} \quad (31)$$

Матрица $U_\tau$ по формуле Фробениуса получится:

$$U_\tau = \begin{bmatrix} \frac{1}{k_0 + \sigma_\tau^2} + \frac{k_1^2}{(k_0 + \sigma_\tau^2)^2} J_{1,2n} S^{-1} J_{2n,1} & -\frac{k_1}{k_0 + \sigma_\tau^2} J_{1,2n} S^{-1} \\ -\frac{k_1}{k_0 + \sigma_\tau^2} S^{-1} J_{2n,1} & S^{-1} \end{bmatrix} \quad (32)$$

Представим матрицу $S^{-1}$ как:

$$S^{-1} = b_1 E_{2n} + b_2 P_{2n} + b_3 J_{2n} \quad (33)$$

Но матричное произведение (31) и (33) тогда должно дать единичную матрицу $E_{2n}$ (размерностью $2n \times 2n$).

$$E_{2n} = (a_1 E_{2n} + a_2 P_{2n} + a_3 J_{2n})(b_1 E_{2n} + b_2 P_{2n} + b_3 J_{2n}) =$$

$$= a_1 b_1 E_{2n} + a_1 b_2 P_{2n} + a_1 b_3 J_{2n} + a_2 b_1 P_{2n} + a_2 b_2 E_{2n} + a_2 b_3 J_{2n} +$$

$$+ a_3 b_1 J_{2n} + a_3 b_2 J_{2n} + a_3 b_3 2n J_{2n} = (a_1 b_1 + a_2 b_2)E_{2n} + (a_1 b_2 + a_2 b_1)P_{2n} +$$

$$+ (a_1 b_3 + a_2 b_3 + a_3 b_1 + a_3 b_2 + 2n a_3 b_3) J_{2n} \quad (34)$$

Тогда из (34) получаем систему уравнений:

$$\begin{cases} a_1 b_1 + a_2 b_2 = 1 \\ a_1 b_2 + a_2 b_1 = 0 \\ a_1 b_3 + a_2 b_3 + a_3 b_1 + a_3 b_2 + 2n a_3 b_3 = 0 \end{cases} \quad (35)$$

Если решить эту систему, то получим:

$$b_1 = \frac{a_1}{a_1^2 - a_2^2} \tag{36}$$

$$b_2 = -\frac{a_2}{a_1^2 - a_2^2} \tag{37}$$

$$b_3 = -\frac{a_3}{(a_1 + a_2 + 2na_3)(a_1 + a_2)} \tag{39}$$

Поскольку матрица $S^{-1}$ выражена в виде (33), как взвешенная сумма единичной матрицы, матрицы $P_{2n}$ из (18) и матрицы состоящей из единиц, то сумма всех элементов любой строки или столбца матрицы $S^{-1}$ будет равняться $b_1 + b_2 + 2nb_3$. Сумма же всех элементов матрицы $S^{-1}$ целиком будет равна $2n(b_1 + b_2 + 2nb_3)$.

Этими свойствами можно воспользоваться, если подставить $S^{-1}$ в (32).

Введем дополнительные коэффициенты $u_1$ и $u_2$.

$$u_1 = \frac{1}{k_0 + \sigma_\tau^2} + \frac{k_1^2}{(k_0 + \sigma_\tau^2)^2} J_{1,2n} S^{-1} J_{2n,1} =$$

$$= \frac{1}{k_0 + \sigma_\tau^2} + \frac{k_1^2}{(k_0 + \sigma_\tau^2)^2} 2n(b_1 + b_2 + 2nb_3) \tag{40}$$

$$u_2 = -\frac{k_1}{k_0 + \sigma_\tau^2}(b_1 + b_2 + 2nb_3) \tag{41}$$

Тогда $U_\tau$, используя (32) – (33) и (36) – (41) можно выразить как:

$$U_\tau = \begin{bmatrix} u_1 & u_2 J_{1,2n} \\ u_2 J_{2n,1} & b_1 E_{2n} + b_2 P_{2n} + b_3 J_{2n} \end{bmatrix} \tag{42}$$

Как видно из полученного выражения (42), для получения матриц $U_\tau$ из (6) при использовании созвездий (15) на основе гипероктаэдров нет необходимости прямого вычисления как матрицы $M_\tau^{(C)}$ так и матрицы $K_\tau^{(C)}$, нет необходимости напрямую обращать матрицу в (6). Вместо этого можно сначала вычислить $k_0, k_1, k_2, k_4$ из (21) – (24), затем найти коэффициенты $a_1, a_2, a_3$ через (28) – (30), что позволит найти $b_1, b_2, b_3$ из (36) – (39) и наконец найти $u_1$ и $u_2$ из (40) – (41). Это всё позволит синтезировать $U_\tau$ через (42). В итоге количество вычислений радикально сокращается и не зависит от размера созвездия (15) и таким же образом не зависит от количества входов пакета (мерности аппроксимируемых пакетом функций).

Но можно пойти и дальше. Рассмотрим для чего нужна сама матрица $U_\tau$.

Матрица $U_\tau$ используется для того чтобы трансформировать значения полигармонических функций $Y_\tau$ в точках созвездия в коэффициенты самих полигармонических функций (требуемых для их вычисления) $\Lambda_\tau$ в (4). Также матрица $U_\tau$ используется в процессе обучения для формирования матрицы $H_\tau$ в (7), которая затем используется в (11).

Представим матрицу $Y_\tau$ в виде:

$$Y_\tau = \begin{bmatrix} y_1 \\ Y_1 \\ Y_2 \end{bmatrix}, \tag{43}$$

где

$y_1$ – первая строка матрицы $Y_\tau$,

$Y_1, Y_2$ – матрицы, объединяющие в себе по $n$

следующих строк матрицы $Y_\tau$

Обозначим как:

$$Y_{1+2} = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}, Y_{2+1} = \begin{bmatrix} Y_2 \\ Y_1 \end{bmatrix}, y_s = J_{1,2n} Y_{1+2} \tag{44}$$

Матрицы $Y_{1+2}$ и $Y_{2+1}$ объединяют в себе матрицы $Y_1$ и $Y_2$ но в разном порядке. Строка $y_s$ представляет собой сумму столбцов $Y_{1+2}$.

Совместим теперь (4), (42), (43), (44).

$$\Lambda_\tau = U_\tau Y_\tau = \begin{bmatrix} u_1 y_1 + u_2 J_{1,2n} Y_{1+2} \\ u_2 J_{2n,1} y_1 + (b_1 E_{2n} + b_2 P_{2n} + b_3 J_{2n}) Y_{1+2} \end{bmatrix} =$$

$$= \begin{bmatrix} u_1 y_1 + u_2 y_s \\ b_1 Y_{1+2} + b_2 Y_{2+1} + J_{2n,1}(u_2 y_1 + b_3 y_s) \end{bmatrix} \tag{45}$$

Представим теперь матрицу $K_\tau$ , которая находится через (2) из матрицы $M_\tau$(которая в свою очередь определяется через (2) или теперь (16)) в следующем виде:

$$K_\tau = [\hat{k}_1 \quad K_1 \quad K_2] \tag{46}$$

где

$\hat{k}_1$ – первый вектор столбец матрицы $K_\tau$,

$K_1, K_2$ – матрицы, объединяющие в себе по $n$

следующих столбцов матрицы $K_\tau$

Обозначим как:

$$K_{1+2} = [K_1 \quad K_2]\,, \quad K_{2+1} = [K_2 \quad K_1]\,, \quad \hat{k}_s = K_{1+2}J_{2n,1} \tag{47}$$

Матрицы $K_{1+2}$ и $K_{2+1}$ объединяют в себе матрицы $K_1$ и $K_2$ но в разном порядке. Столбец $\hat{k}_s$ представляет собой сумму строк $K_{1+2}$.

Совместим теперь (7), (42), (46), (47).

$$H_\tau = K_\tau U_\tau = \left[u_1\hat{k}_1 + u_2\hat{k}_s \quad b_1K_{1+2} + b_2K_{2+1} + \left(u_2\hat{k}_1 + b_3\hat{k}_s\right)J_{1,2n}\right] \tag{48}$$

Таким образом, используя (45) и (48) можно напрямую вычислять $\Lambda_\tau$ и $H_\tau$ из $Y_\tau$ и $K_\tau$ (причем сложность вычислений по сравнению с (4) и (7) существенно сокращается). Для этого необходимо лишь предварительно (единственный раз при инициализации каскада) определить коэффициенты $b_1, b_2, b_3, u_1, u_2$ для каждого из пакетов. Необходимость в отдельном вычислении и хранении в памяти матриц $U_\tau$ (так же как и матриц $M_\tau^{(C)}$ и $K_\tau^{(C)}$) отпадает.

И, наконец, рассмотрим выражение (10).

Если представить $\Psi_\tau$ как:

$$\Psi_\tau = [\psi_1 \quad \Psi_1 \quad \Psi_2] \tag{49}$$

где

$\psi_1$ – первый столбец матрицы $\Psi_\tau$,

$\Psi_1, \Psi_2$ – матрицы, объединяющие в себе по $n$

следующих столбцов матрицы $\Psi_\tau$

Тогда выражение (10), подставив в него (15), можно записать как:

$$G_{\tau-1} = X_{\tau-1} \circ \left(\Psi_\tau J_{k_\tau,1}J_{1,n_{\tau-1}}\right) + [\Psi_1 - \Psi_2] \tag{50}$$

Как результат, устраняем из (10) наиболее сложную в вычислительном плане операцию матричного умножения.

В итоге, если теперь проанализировать выражения (14) – (50), то можно сделать вывод, что использование процедуры формирования созвездий на основе гипероктаэдров (+точка в начале координат) обеспечивает не только универсальный способ формирования созвездий для любого из пакетов, но и позволяет значительно упростить вычисления. Нет необходимости хранения в памяти и использования матриц созвездий $C_\tau$ в явном виде. Вместо этого можно использовать выражения (16) и (50), в которые трансформируются выражения (1) и (10).

Также не требуется при инициализации вычислять и хранить $U_\tau$ для возможности выполнения (4) и (7). Вместо этого можно вычислить (однократно при инициализации) через выражения (21) – (24), (28) – (30) и (36) – (41) коэффициенты $b_1, b_2, b_3, u_1, u_2$. А затем вместо (4) и (7) использовать (45) и (48).

Таким образом, использование гипероктаэдов в качестве созвездий может дать значительное преимущество в вычислительном плане. Впрочем, можно предположить, что использование других многомерных симметричных фигур в качестве созвездий также может привести к оптимизации вычислений, но вместо (16), (45), (48), (50) получатся другие выражения. Это может быть темой дальнейших исследований.

Хочется обратить внимание, что в ходе вычислительных экспериментов, при реализации алгоритмов в коде с использованием библиотеки Pytorch (хотя возможно это было лишь особенностью конкретной реализации) выражения (16), (45), (48), (50) несмотря на то, что требуют гораздо меньше математических операций, начинают давать преимущество в скорости исполнения перед (1), (4), (7), (10), только когда количество входов в пакете (и соответственно размеры матриц) становится больше чем несколько сотен (наблюдается при исполнении как на CPU, так и на GPU). Возможно, это связано с гораздо более сильной относительной оптимизацией операций матричного умножения как в аппаратной, так и в программной части по сравнению с другими операциями (более простыми). Например, разделение матрицы $\Psi_\tau$ на части в (49) и их вычитание в (50) может выполняться медленнее чем перемножение $\Psi_\tau$ на $C_\tau$ в (10) если их размеры менее сотен строк. Но это говорит о том, что потенциально в будущем полигармонический каскад может быть реализован более эффективно, чем текущая версия кода, используемая в вычислительных экспериментах, результаты которых представлены в этой статье ниже.

Рассмотрим теперь следующий важный вопрос (касающийся инициализации полигармонического каскада), это начальные значения в точках созвездий, т.е. начальные значения матриц $Y_\tau$. Их выбор будет определять, какие функции будут вычисляться от пакета к пакету в каскаде в самом начале при его создании.

Одним из важных этапов при обучении полигармонического каскада является процедура последовательного вычисления (от последнего пакета к первому) матриц с частными производными $G_\tau$. Хотя значения производных не используются напрямую для оптимизации (как при градиентном спуске), они важны для получения систем уравнений.

Если же, например, присвоить в каком либо пакете всем значениям в матрице $Y_\tau$ одно и то же фиксированное значение (не обязательно ноль), то все функции на выходе этого пакета всегда будут возвращать это фиксированное значение (или очень близкое к нему) а их производные будут везде равняться нулю (или будут к нему близки). В итоге, для всех пакетов, которые стоят перед пакетом с номером $\tau$ , все матрицы производных обнулятся, и эти пакеты не смогут обучаться (изменить свои матрицы $Y_\tau$), что прямо следует из (13). Впрочем, это создаст проблему лишь на первой итерации, после которой значения в $Y_\tau$ могут быть изменены и на следующей итерации к обучению подключится весь каскад. Однако, если взять такую неудачную матрицу значений $Y_\tau$ уже для двух любых пакетов в каскаде, то это приведет каскад в неработоспособное состояние.

Из этих соображений следует, что $Y_\tau$ должны состоять из отличающихся друг от друга элементов. В качестве начальных $Y_\tau$ нельзя брать нулевую или состоящую из единиц матрицы более чем в одном из пакетов.

Но в каком диапазоне должны лежать начальные значения $Y_\tau$? Очевидно, что они должны лежать в том же диапазоне значений, в котором расположены точки созвездия в следующем пакете. А поскольку в качестве созвездий решено было взять гипероктаэдры, вершины которых расположены на единичном расстоянии от начала координат, то получается, что начальные значения элементов $Y_\tau$ можно брать в диапазоне примерно от -1 до 1, но так, чтобы длины строк-векторов матрицы $Y_\tau$ имели длину не слишком отличающуюся от единицы.

Исходя из подобных соображений можно предложить процедуру инициализации $Y_\tau$. Эти матрицы могут быть заполнены случайными значениями от -1 до 1, но с ограничением на длину получившихся векторов строк в матрице (как вариант, их можно сначала заполнить случайными значениями, а потом для всех строк выполнить нормализацию).

Но почему бы тогда просто не взять значения в точках созвездий $Y_\tau$ такими, чтобы на выходе пакета они совпали с созвездием следующего пакета? Теоретически такое вполне можно сделать. Однако, в случае с описанной в этой статье процедурой инициализации созвездий на основе гипероктаэдров, такое возможно только в том случае, когда идущие друг за другом пакеты имеют одинаковое количество входов и выходов. Тем не менее, этим вполне можно воспользоваться, пусть и не для всего полигармонического каскада целиком.

Предположим, что в полигармоническом каскаде есть фрагмент из последовательности пакетов, начиная с номера $\tau$ и до номера $t$, таких, что количество входов и выходов в каждом из них совпадает.

$$n_{\tau-1} = n_\tau = n_{\tau+1} = \cdots = n_{t-1} = n_t = n$$

Но тогда, если пользоваться предложенной в этой статье процедурой инициализации созвездий, они во всех пакетах будут одинаковые, с одинаковым количеством точек.

$$k_\tau = k_{\tau+1} = \cdots k_{t-1} = k_t = 2n + 1 = k$$

Обозначим количество входов или выходов каждого из этих пакетов просто как $n$. А количество точек созвездий в каждом из них как $k$.

Тогда получим схему, изображенную на рисунке 3.

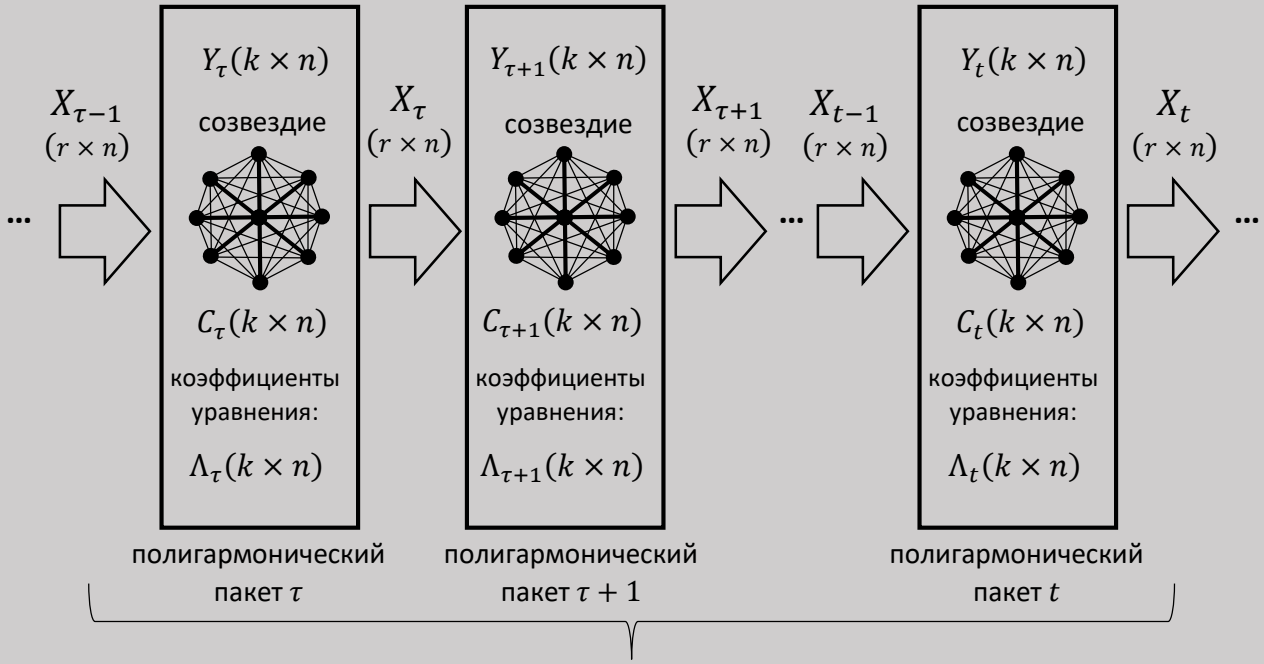


Рисунок 3. Фрагмент полигармонического каскада

В этом случае при инициализации такой последовательности пакетов можно взять матрицы $Y_\tau$ равными $C_\tau$.

$$Y_\tau = Y_{\tau+1} = \cdots = Y_t = C_\tau = \cdots = C_t \tag{50}$$

Если выбрать $\sigma_\tau^2$ из (6) или (28), (30) равным нулю, а вектора строки из $X_{\tau-1}$ будут лежать в области точек созвездия пакета $\tau$, то $X_{\tau-1}$ пройдет в результате обработки через все пакеты (в том виде, как они будут получены после инициализации) до $X_t$ практически не изменившись ($X_t \approx X_{\tau-1}$).

Воспользовавшись эти свойством, можно при необходимости наращивать количество пакетов в каскаде до неограниченного количества (сохраняя его работоспособность и способность к обучению после инициализации).

В завершение, хотелось бы сказать о дополнительных возможностях оптимизации вычислений в процессе обучения. Рассмотрим из каких строчек

будет состоять матрица $K_\tau$ из (3) для первого из пакетов в каскаде т.е. матрица $K_1$. Каждая её строчка будет целиком определяться соответствующим ей вектором из батча. В следующей эпохе этот вектор попадет в другой батч, но соответствующая ему строка в $K_1$ снова будет той же самой. Значит можно предварительно обработать всю входную часть обучающей выборки и найти всевозможные строчки матрицы $K_1$. А обучающие батчи можно сразу подавать на вход в виде готовых матриц $K_1$ в выражение (3) или (48) минуя (16) и (2). Но можно пойти еще дальше и выполнить ту же схему сразу для матрицы $H_1$ из (7) или (48). Тогда необходимость в $K_1$ при обучении отпадает, а выражение (3) можно заменить на:

$$X_1 = H_1 \mathrm{Y}_1 \tag{51}$$

Также дополнительную оптимизацию вычислений можно получить при нахождении вектора B в (12) через решение системы линейных уравнений. Матрица $\left(\Omega_1 + \Omega_2 + \cdots + \Omega_q + \alpha E\right)$ из (12) всегда будет симметричной и положительно определенной. Следовательно, для оптимизации вычислений можно использовать разложение Холецкого.

Итак, с учетом теоретических положений из [1], [2] и [3] и предложенных принципов инициализации полигармонического каскада из данной статьи, в итоге, были разобраны все элементы, необходимые для реализации данной архитектуры машинного обучения в коде, для её запуска и проверки работоспособности.

**Запуск и проверка.**

Ниже представлены результаты проведенных вычислительных экспериментов. В качестве платформы для исследований использовался компьютер со следующими характеристиками: процессор Intel Core i9-10920X, видеокарта NVIDIA GeForce RTX 3070 8Гб. Вычисления, связанные с полигармоническим каскадом, выполнялись на видеокарте.

Параметры *b* и *c* из (2) (и как следствие в (21) - (24)) использовались равными *b=5, c=400*. Во всех экспериментах вычисления проводились с использованием типа данных float32.

Исходный код, необходимый для воспроизведения экспериментов, доступен в публичном репозитории GitHub: https://github.com/xolod7/polyharmonic-cascade.git [12] и заархивирован на Zenodo с постоянным идентификатором DOI: https://doi.org/10.5281/zenodo.16811633 [13].

## MNIST

Для проведения первой серии экспериментов был выбран набор данных MNIST ([9] с загрузкой через библиотеку torchvision). Обучающая часть выборки содержит 60 тысяч примеров, тестовая - 10 тысяч.

Данные использовались в оригинальном виде без предварительного преобразования, аугментации или выделения каких-либо признаков по изображению. При обучении никак не учитывалось, что перед нами структура в виде изображения. На вход каскада просто поступали вектора из 784 параметров (“мешок пикселей”).

Из самой постановки задачи в таком виде, видно, что цель экспериментов с MNIST не состояла в получении результатов, близких к SOTA моделям для данного теста. Цель была в проверке принципиальной работоспособности полигармонического каскада (как архитектуры машинного обучения), его способности обучаться по одному и тому же алгоритму обучения [3] вне зависимости от конфигурации каскада (даже экстремальной), пусть и на относительно небольшом наборе данных. Т.е., цель состояла в том, чтобы проверить способность полигармонического каскада к масштабированию и устойчивость лежащих в его основе алгоритмов при любой конфигурации каскада (датасет MNIST был здесь взят в качестве рабочего примера).

Масштабирование на 10 выходов осуществлялось по процедуре, описанной в [3] путем повторения структуры каскада на каждый выход. При таких условиях проверка принципиальной работоспособности полигармонического каскада с помощью MNIST дополнительно выглядит более тщательной по сравнению с прогнозированием какого-то одного показателя на выходе), поскольку стоит лишь одному классификатору из десяти обучиться неверно, как это сразу скажется на общем результате. Размер батча во всех экспериментах с MNIST был выбран 2000.

### Эксперимент (MNIST) 1.

Использовался полигармонический каскад 784-100-20-20-10 из четырех пакетов. Количество настраиваемых параметров в каскаде около 1.6 млн. Обучение проходило в течение 10 эпох. Значение $\alpha$ из (12) взято равное 200.

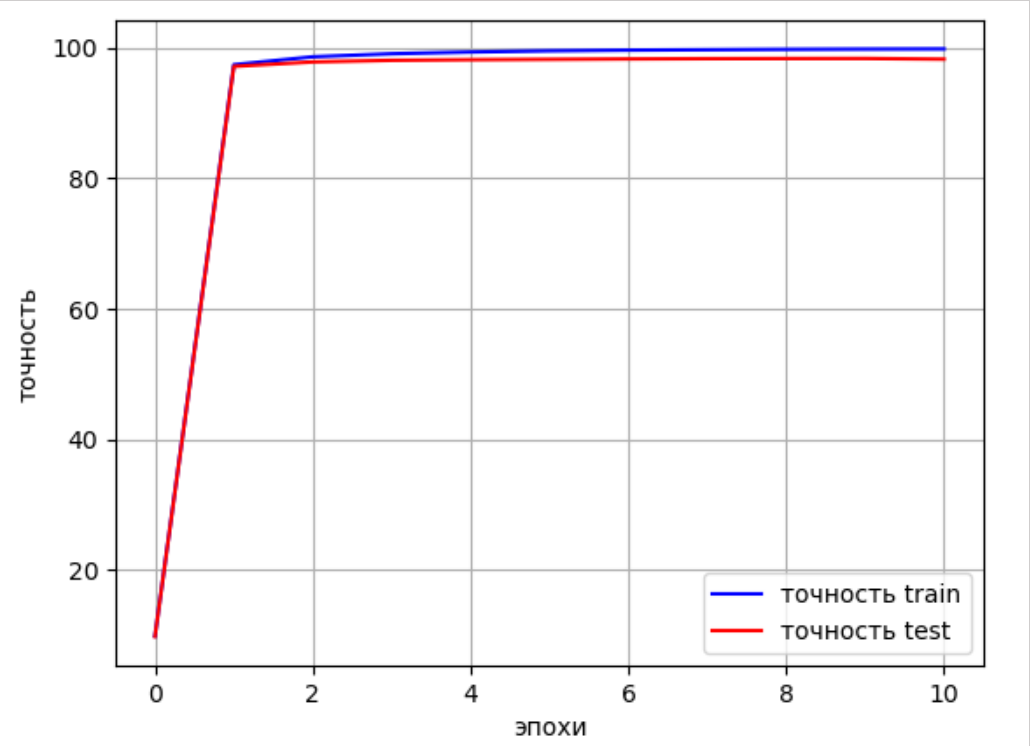


Рисунок 4. Эксперимент 1. График обучения на датасете MNIST.

Время выполнения эпохи составило около 1.4 – 1.5 секунд. После первой эпохи была достигнута точность 97.14% на тестовом множестве.

Если рассмотреть верхнюю часть графика более подробно:

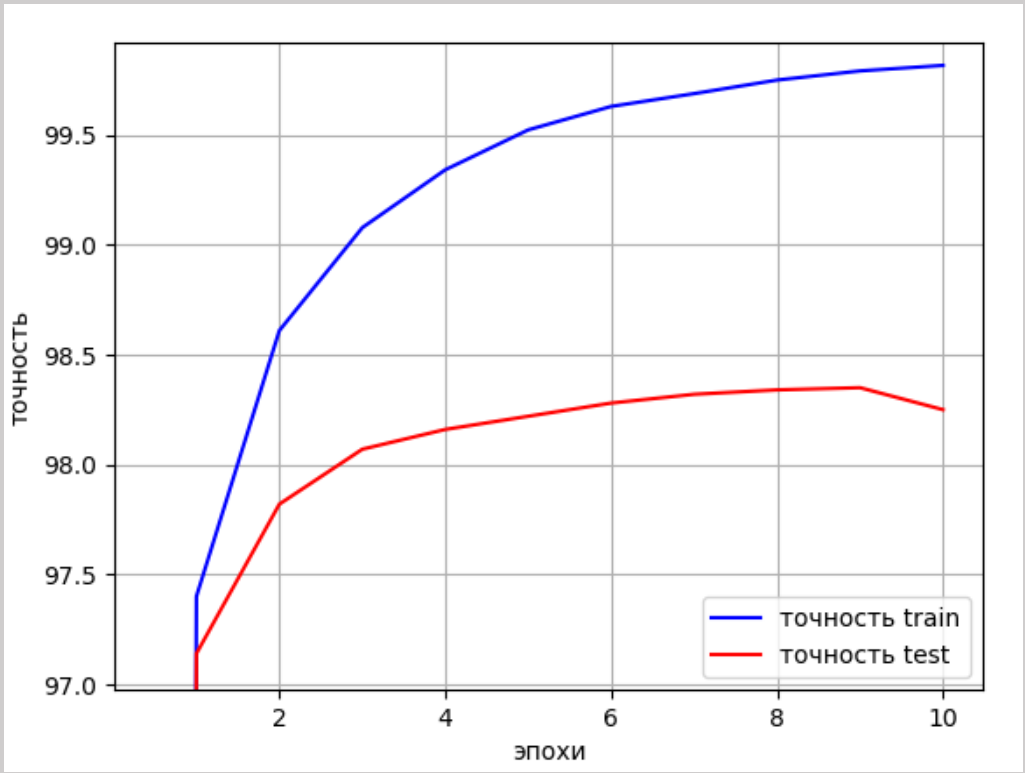


Рисунок 5. Эксперимент 1. График обучения на датасете MNIST (в масштабе).

В течении 10-ти эпох точность по тестовой выборке достигла 98.35%. Впрочем, при каждом запуске это значение может отличаться (но стабильно больше 98.1%).

В итоге, первый эксперимент подтвердил применимость полигармонического каскада как архитектуры машинного обучения, а также работоспособность предложенного в [3] метода его обучения и процедуры инициализации с использованием гипероктаэдров, рассмотренной в данной статье. Полученный результат 98.35% находится вполне на уровне других методов (при сравнении с различными методами обучения [9]) для теста MNIST при условии, что никак, ни прямо, ни косвенно через особенности архитектуры (как для сверточных сетей) не учитывалось, что происходит обработка именно изображений (что входные параметры отвечают за расположенные друг относительно друга пиксели).

Эксперимент (MNIST) 2.

Чтобы проверить масштабируемость алгоритмов, нарастим количество настраиваемых параметров в каскаде до 75.74 млн (увеличение по сравнению с экспериментом 1 почти в 50 раз). Такое количество даст каскад из 5-ти слоев 784-1000-1000-1000-1000-10. Значение $\alpha$ возьмем как в предыдущем случае равным 200.

Выполним обучение в течение 10 эпох.

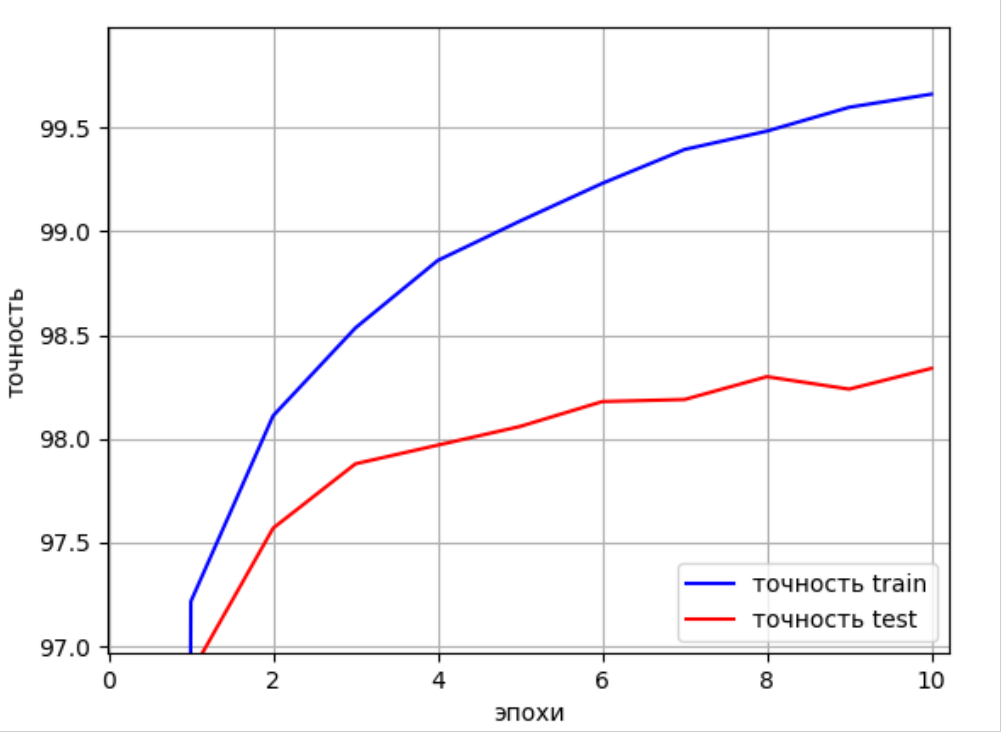


Рисунок 6. Эксперимент 2 (MNIST). Обучение 10 эпох.

Время выполнения эпохи увеличилось до 12 - 17 сек (по сравнению с экспериментом 1 увеличение примерно в 10 раз, хотя количество параметров увеличилось почти в 50 раз). Динамика изменения точности получилась близкая к тому, что было и в эксперименте 1.

Посмотрим на динамику обучения при увеличении количества эпох до ста.

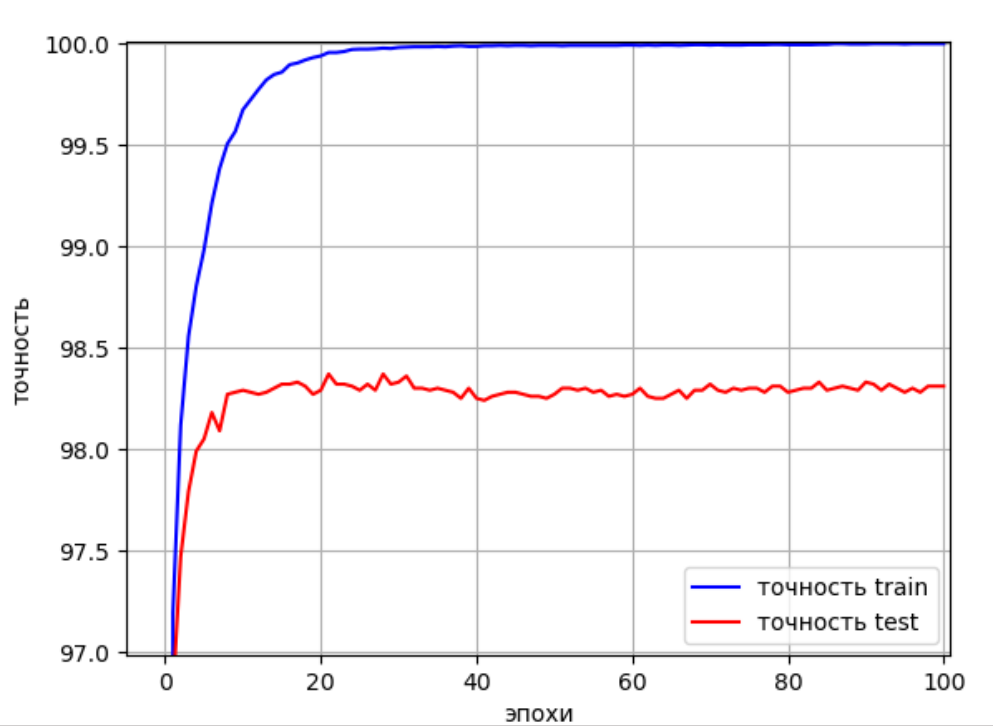


Рисунок 7. Эксперимент 2 (MNIST). Обучение 100 эпох.

Точность на обучающем множестве достигла почти 100%. Точность на тестовой выборке стабилизировалась в диапазоне 98.25-98.3%. Какого-то заметного процесса переобучения при продолжении эпох для данной задачи не наблюдается.

Возможно каскад с 75 млн. параметров избыточен для решения задачи и просто “запоминает” часть обучающих примеров вместо обобщения. Попробуем специально замедлить скорость обучения, присвоив $\alpha = 10000$ и выполнить 300 эпох.

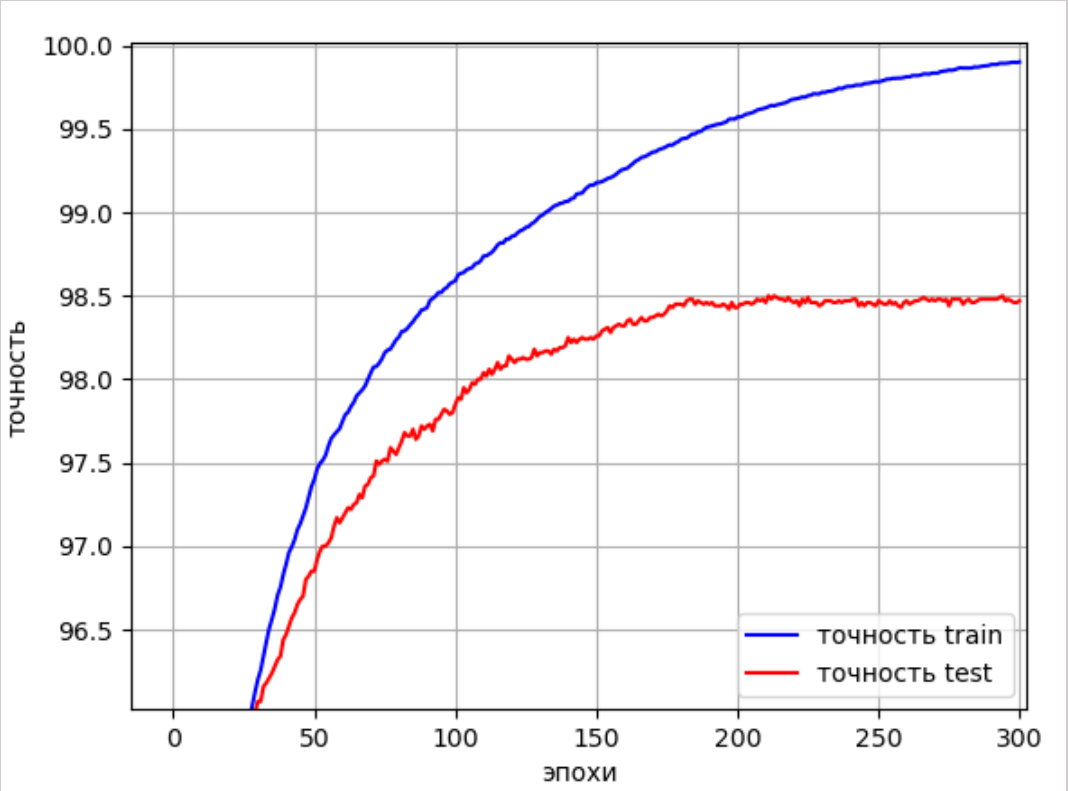


Рисунок 8. Эксперимент 2 (MNIST). Специально замедленное обучение.

Как видно, это благотворно сказалось на качестве и каскад превысил значение точности 98.45%.

Эксперимент (MNIST) 3.

Теперь попробуем проверить масштабируемость полигармонического каскада и алгоритма его обучения другим способом. Вместо увеличения количества входов и выходов в пакетах, значительно увеличим количество самих пакетов в каскаде.

Возьмем каскад, состоящий из 100 слоёв 784-100-100-…-100-100-10.

У всех пакетов, кроме первого и последнего будет по 100 входов и 100 выходов. В этом случае можно использовать инициализацию (50). Получившийся каскад будет содержать 21.27 млн. параметров. Значение $\alpha$ возьмем равным 50.

Выполним обучение в 10 эпох.

Время выполнения одной эпохи составило примерно 25.5 сек. После первой эпохи была получена точность примерно в 97%. Затем, еще через несколько эпох точность стабилизировалась в районе 98%.

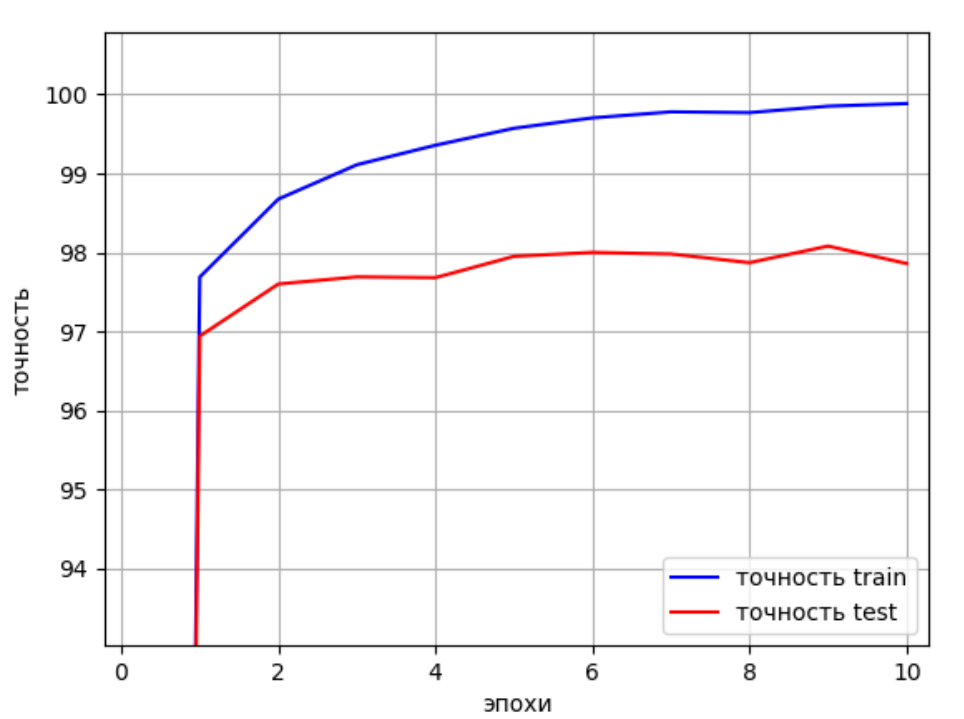


Рисунок 9. Эксперимент 3 (MNIST). Обучение каскада из 100 слоёв.

Возможно каскад из ста слоёв не самая оптимальная модель для решения задачи классификации MNIST, но полученный результат свидетельствует, что при наращивании количества пакетов в полигармоническом каскаде, он остается работоспособным, может быть проинициализирован по предложенной в данной статье процедуре и способен обучаться по тому же самому алгоритму без каких-либо принципиальных изменений (может потребоваться лишь подбор более оптимального параметра $\alpha$).

Эксперимент (MNIST) 4.

Теперь возьмём каскад, состоящий из 500 слоев 784-100-25-25-…-25-10. Получается каскад из еще большого числа пакетов чем ранее, но большинство из них содержит только по 25 входов и выходов.

Получившийся каскад будет содержать 7.95 млн. параметров. Значение $\alpha$ возьмем равным 2000. Выполним обучение в 10 эпох.

Как видно из рисунка, при обучении каскад показал качественно такую же динамику изменения точности, как и в первом эксперименте, достигнув значительного прогресса в первую же эпоху.

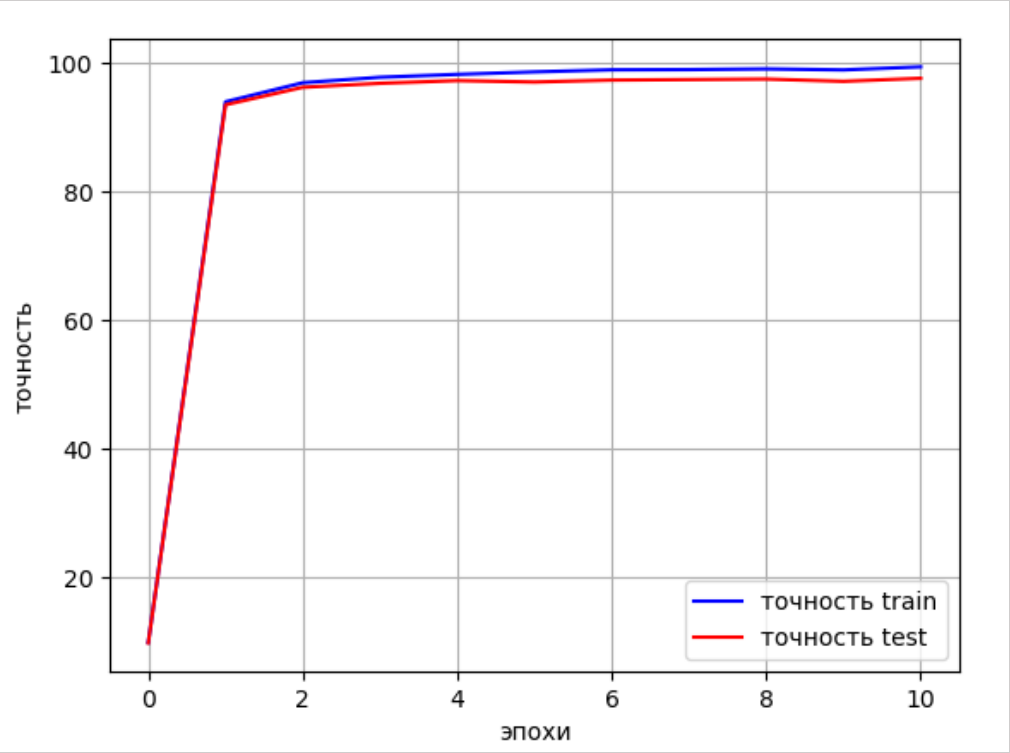


Рисунок 10. Эксперимент 4 (MNIST). Обучение каскада из 500 слоёв.

Посмотрим более детально на показатели точности на графике.

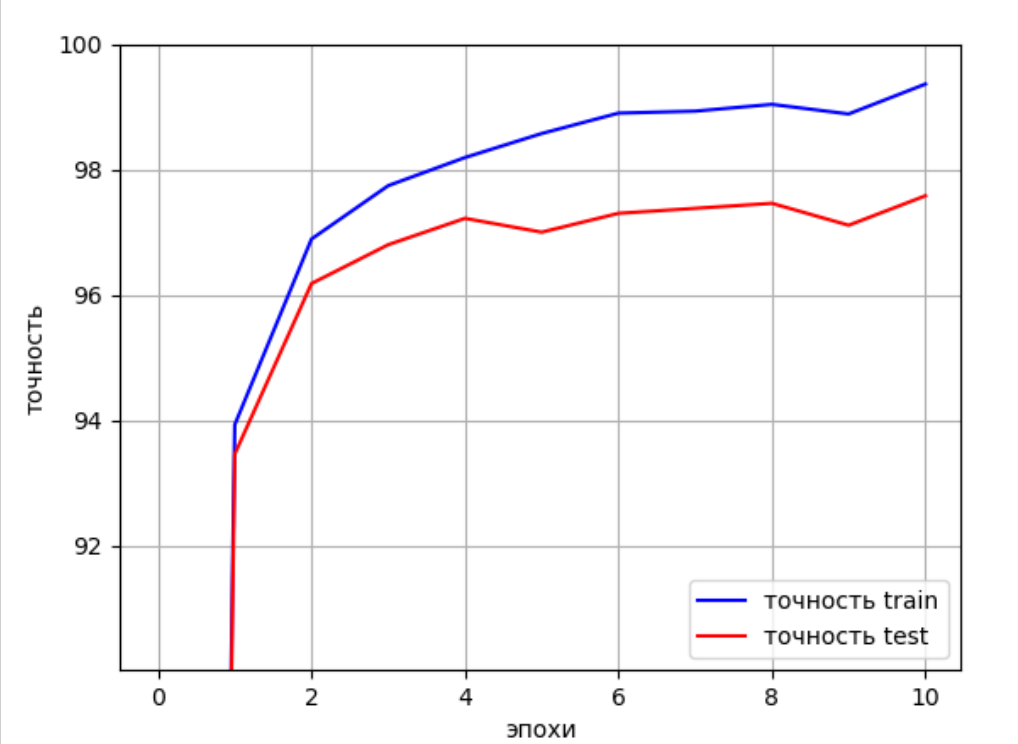


Рисунок 11. Эксперимент 4 (MNIST). Обучение каскада из 500 слоёв (график в масштабе).

Показатели точности оказались ниже, однако, как следует из результатов, каскад из 500 слоёв остается работоспособным и может обучаться по тому же самому алгоритму, как и каскад из 4-х слоёв. Пакеты в данном случае

соединены последовательно друг за другом, без использования дополнительных методик, таких как “skip connection”, которые применяются для создания глубоких структур у нейросетей. Возможно использование “skip connection” (или подобных методик) могло бы положительно сказаться и на работе полигармонического каскада, но в рамках данного исследования это не проверялось и может быть темой дальнейших исследований.

Дополнительно можно отметить, что время на эпоху в четвертом эксперименте увеличилось до 70 сек. Несмотря на то, что каскад здесь содержит меньшее количество параметров чем в эксперименте 2, каждая эпоха проходит значительно медленнее. Возможно это связано с особенностями исполнения на GPU, когда умножение более крупных матриц выполняется более эффективно с точки зрения выполнения количества элементарных математических операций в единицу времени.

*Промежуточный вывод.*

Эксперименты с датасетом MNIST подтвердили работоспособность полигармноческих каскадов с различного размера и сложности. Один и тот же алгоритм обучения применим как к каскаду с 4-мя, так и каскаду с 500 слоями (как к каскаду с 1.6 млн., так и к каскаду с 75 млн. настраиваемых параметров).

HIGGS

Теперь проверим способность полигармонического каскада обучаться на датасете большого размера со сложными нелинейными зависимостями и высоким уровнем шума.

Для этих целей был выбран набор данных HIGGS [5, 6], первоначально представленный в работе Baldi et al. [5] и доступный через репозиторий UCI Machine Learning Repository [6]. Он содержит 11 млн. примеров (первые 10.5 млн. использовались как обучающие, последние 500 тыс. как тестовые) с 28 входными признаками.

Было проведено два теста. В обоих случаях использовался каскад из 20 пакетов(слоёв) 28-200-200-…-200-200-1. У всех пакетов, кроме первого и последнего в таком каскаде получается по 200 входов и 200 выходов. Использовалась инициализация (50). Такой каскад будет содержит 21.27 млн. параметров. Значение $\alpha$ было взято равным 1000. Размер батча был выбран 14000.

Тест (Higgs) 1.

В первом тесте данные на обучение (в виде батчей) в каскад подавались в том же виде, как они представлены в датасете (логарифмирования признаков не проводилось), выполнялась лишь процедура нормировки по максимальному и минимальному значениям признаков, чтобы уложить значения входных признаков в диапазон (-1,1)).

Обучение проходило в течение 500 эпох.

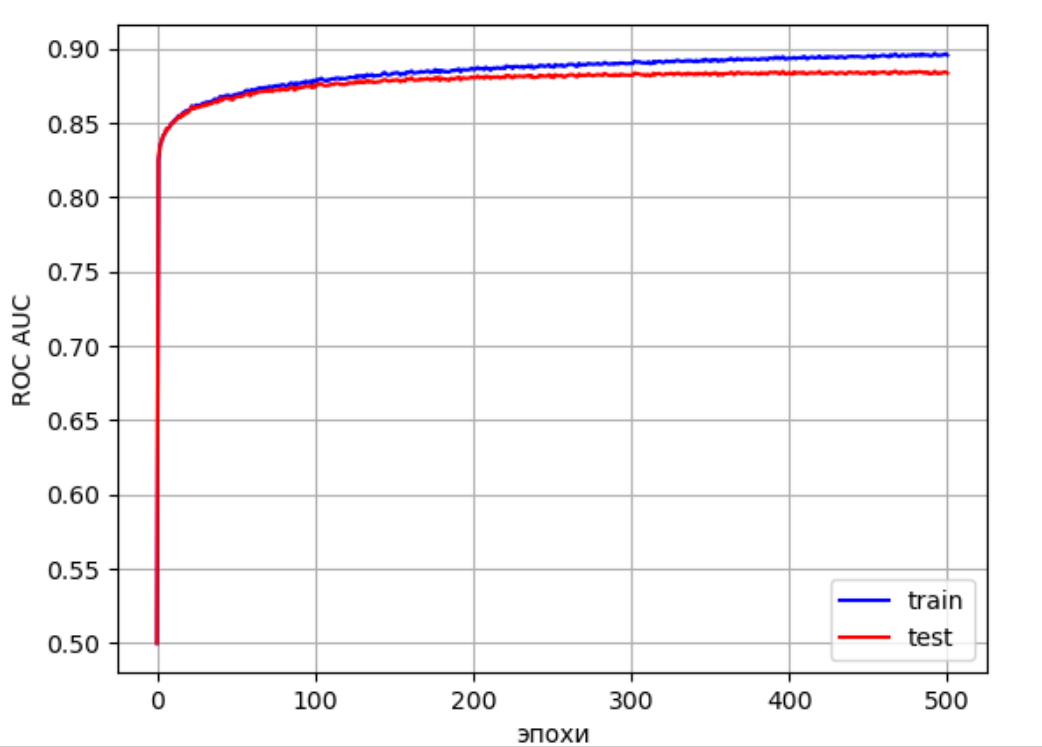


Рисунок 12. Тест 1. График обучения на датасете HIGGS.

Время на эпоху составило приблизительно 14.5 – 15 минут.

Как видно по графику, показатель ROC AUC плавно и схонхронно растет как для обучающего так и для тестового множества во всём процессе обучения.

Посмотрим более подробно на верхнюю часть графика.

Видно, что значение AUC равное 0.88 для тестового множества было достигнуто примерно на 160-той эпохе. В районе 500-той эпохи значение AUC для тестового множества колеблется в районе 0.884 – 0.885. Графики AUC как для обучающего так и для тестового множества продолжают расти (разница меджу ними примерно 0.012) и скорее всего если продолжить обучение, отметка 0.885 будет уверенно пройдена. Переобучения, когда AUC для обучающего множества продолжает расти а для тестового начинает падать, на представленной дистанции обучения не наблюдается.

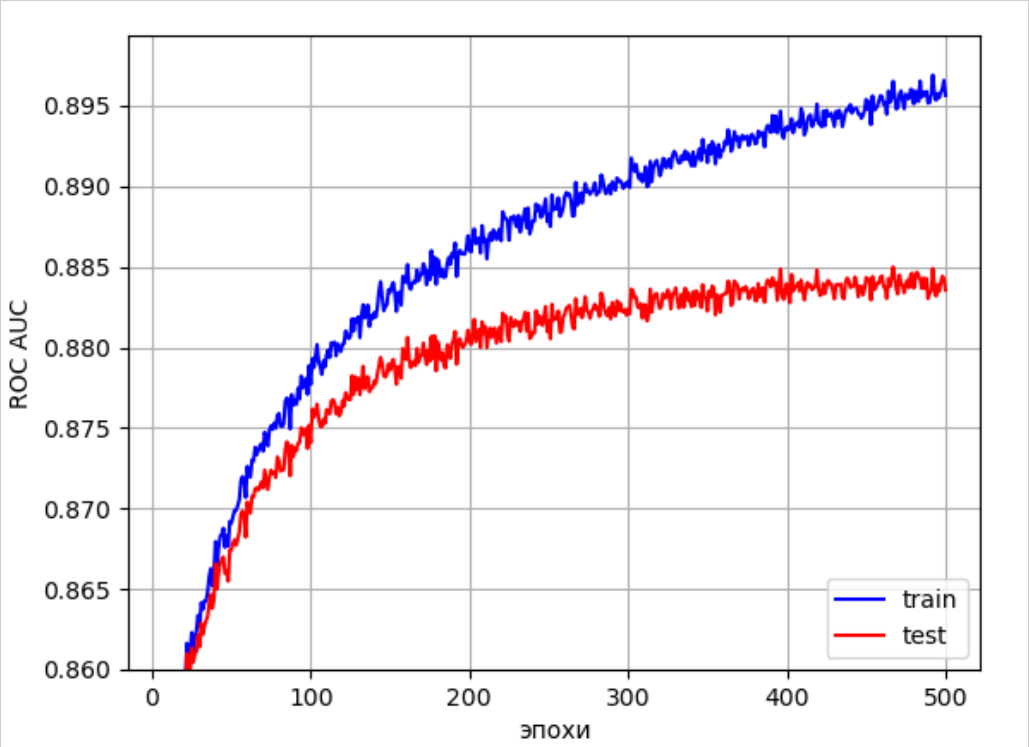


Рисунок 13. Тест 1. График обучения на датасете HIGGS (в масштабе).

Тест (Higgs) 2.

Тест 2 во всем повторяет тест 1, за исключением того, что часть признаков была предварительно обработана. От признаков с номерами 1, 6, 10, 14, 18, 22, 23, 24, 25, 26, 27, 28 был вычислен натуральный логарифм (функция np.log() библиотеки Numpy). От признака с номером 4 была вычислена функция np.log1p(). Цель заключалась в получении более равномерного распределения их значений.

В результате был получен график, изображенный на рисунке 14. При таком масштабе визуально он практически не отличается от графика обучения в тесте 1 (Рисунок 12).

Посмотрим более подробно на его верхнюю часть (Рисунок 15). Эта диаграмма также очень похожа на рисунок 13. Но если внимательно проанализировать, то видно, что изменения распределения значений входных признаков дает кратковременный эффект в скорости обучения приблизительно на первых 50-ти эпохах. Далее, к 500-там эпохам эффект становится обратным. К концу обучения AUC по тестовому множеству колеблется около значения 0.883. Однако, как и в первом тесте видно, что показатель AUC продолжает медленно, но увеличиваться и возможно добрался бы до отметки 0.885 если бы обучение было продолжено.

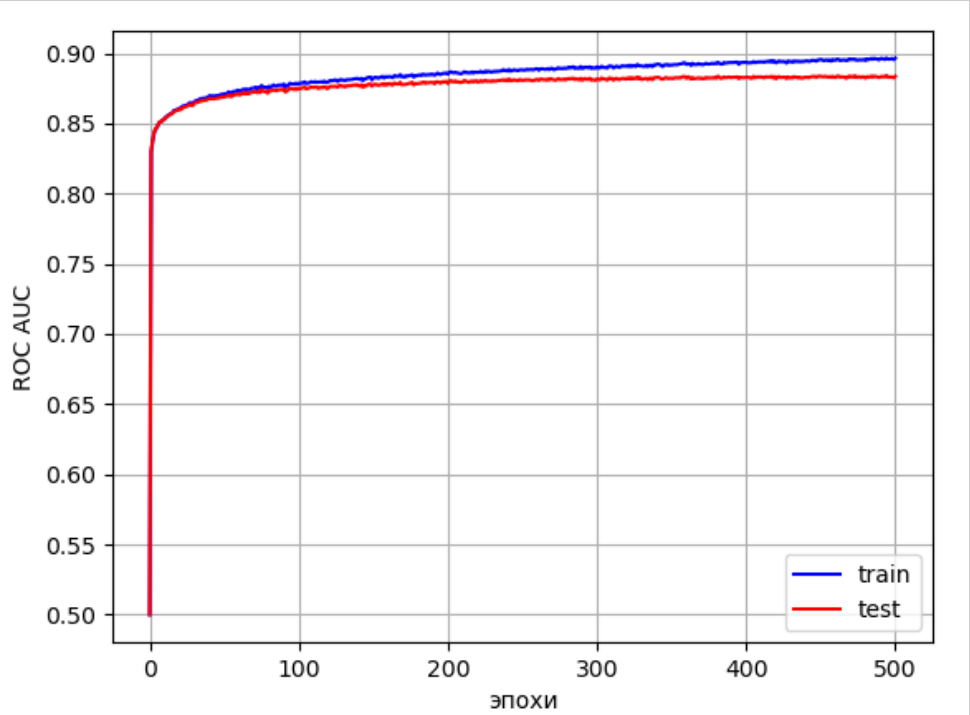


Рисунок 14. Тест 2. График обучения на датасете HIGGS.

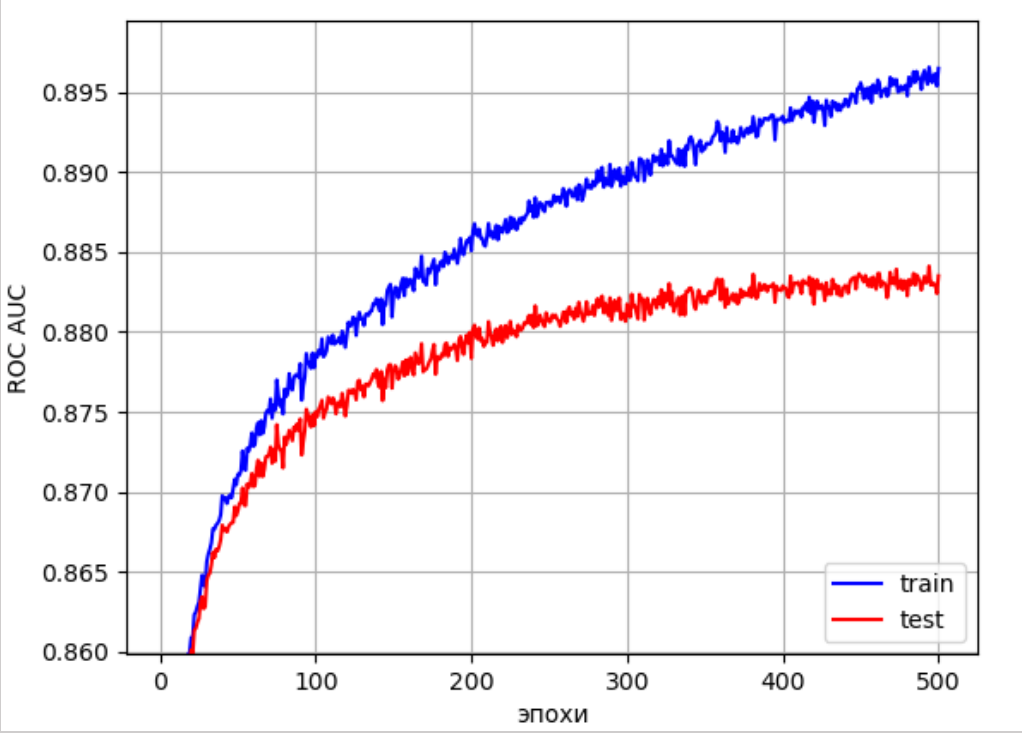


Рисунок 15. Тест 2. График обучения на датасете HIGGS (в масштабе).

Таким образом, по результатам тестов на датасете Higgs Boson, можно сделать вывод, что полигармонический каскад способен работать с большими выборками, содержащими сложные нелинейные зависимости и высокий

уровень шума, достигая сопоставимых результатов с нейросетями глубокого обучения (представленными в статье [5]).

Epsilon

Далее, для оценки эффективности полигармонического каскада в условиях высокоразмерных данных и крупных выборок был выбран датасет Epsilon [10] из PASCAL Large Scale Learning Challenge 2008. Этот энабор данных, содержит 500 тыс. примеров (первые 400 тыс. использовались как обучающие, последние 100 тыс. как тестовые) с 2000 признаков. Датасет был загружен через OpenML API (ID 45575, version=1) с помощью библиотеки scikit-learn.

Тест (Epsilon) 1.

Использовался полигармонический каскад 2000-3-20-20-1 небольшого размера из четырех пакетов. Количество настраиваемых параметров в каскаде 13004. Обучение проходило в течение 10 эпох. Значение $\alpha$ было взято равное 10.

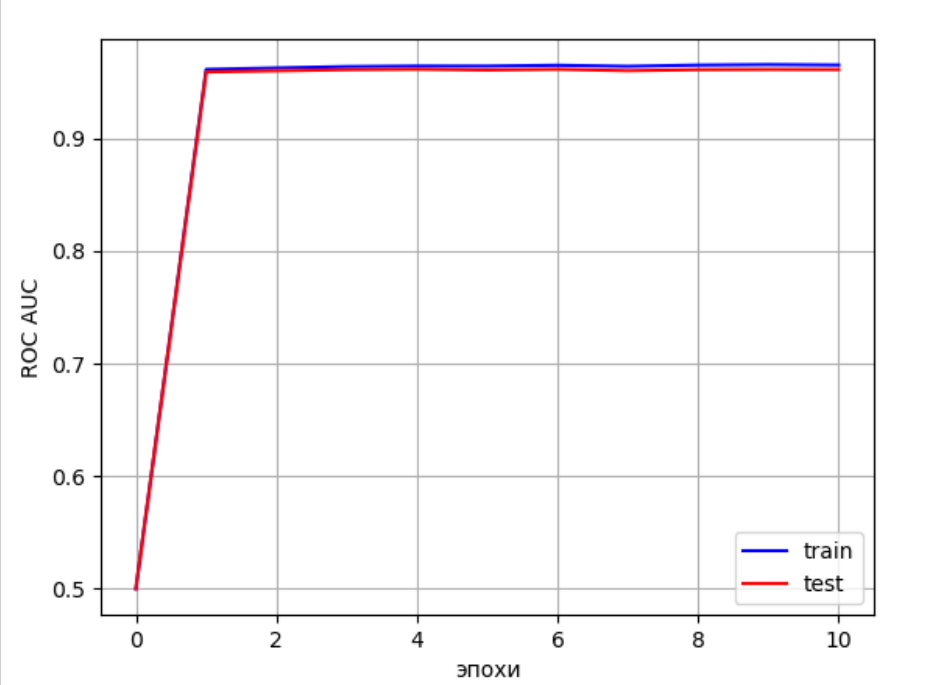


Рисунок 16. Тест 1. Обучение на датасете Epsilon.

Диаграмма обучения на датасете Epsilon похожа на графики обучения на датасете MNIST. Каскад фактически обучился на первой же эпохе. Тест подтвердил способность полигармонического каскада обучаться на крупных высокоразмерных выборках, быстро достигая высокого качества.

Далее, при продолжении эпох, какого-то ярко выраженного процесса переобучения не наблюдается.

Посмотрим на график более подробно.

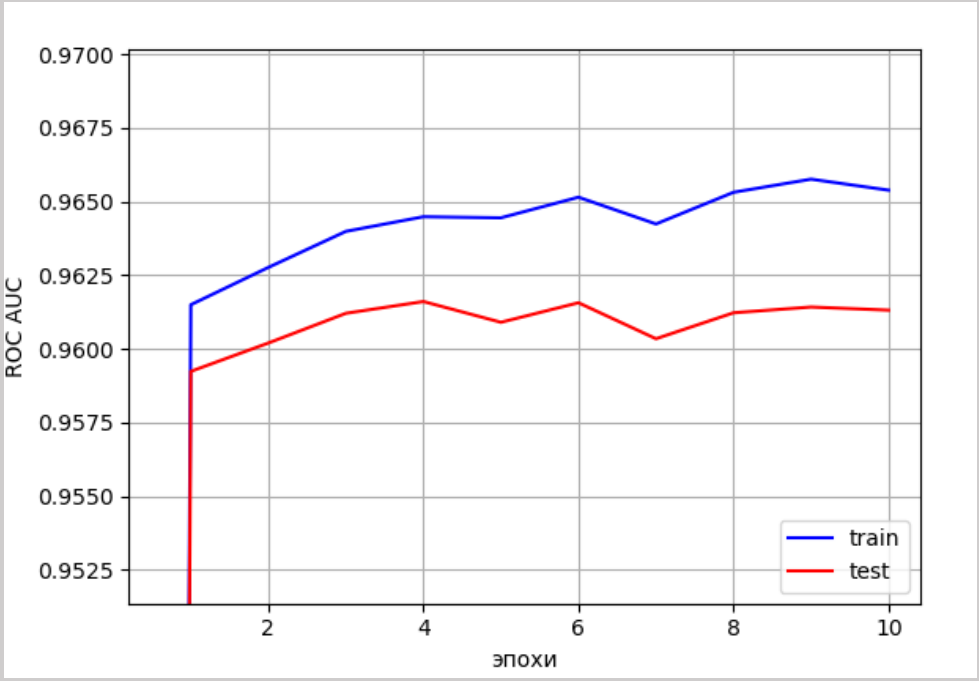


Рисунок 17. Тест 1. Обучение на датасете Epsilon (в масштабе).

После первой эпохи был достигнут показатель ROC AUC в 0.9592. Затем он вырос к четвертой эпохе до 0.9616, и несколько снизился и к десятой эпохе составляя 0.9613.

Время выполнения эпохи составило чуть больше 5 сек.

Тест (Epsilon) 2.

Увеличим теперь масштаб каскада до 2000-2000-2000-2000-2000-1 используя пять пакетов с 2000 входов и выходов в каждом (кроме последнего). Количество настраиваемых параметров увеличится до 32 млн.

Значение $\alpha$ специально возьмем большим, равным 20000, чтобы замедлить процесс сходимости обучения и более внимательно проследить за динамикой. Выполним 30 эпох.

Время обучения увеличилось до примерно 37-39 секунд на эпоху. Это примерно в восемь раз медленнее, чем в предыдущем тесте, хотя количество настраиваемых параметров в каскаде выросло примерно в две с половиной тысячи раз.

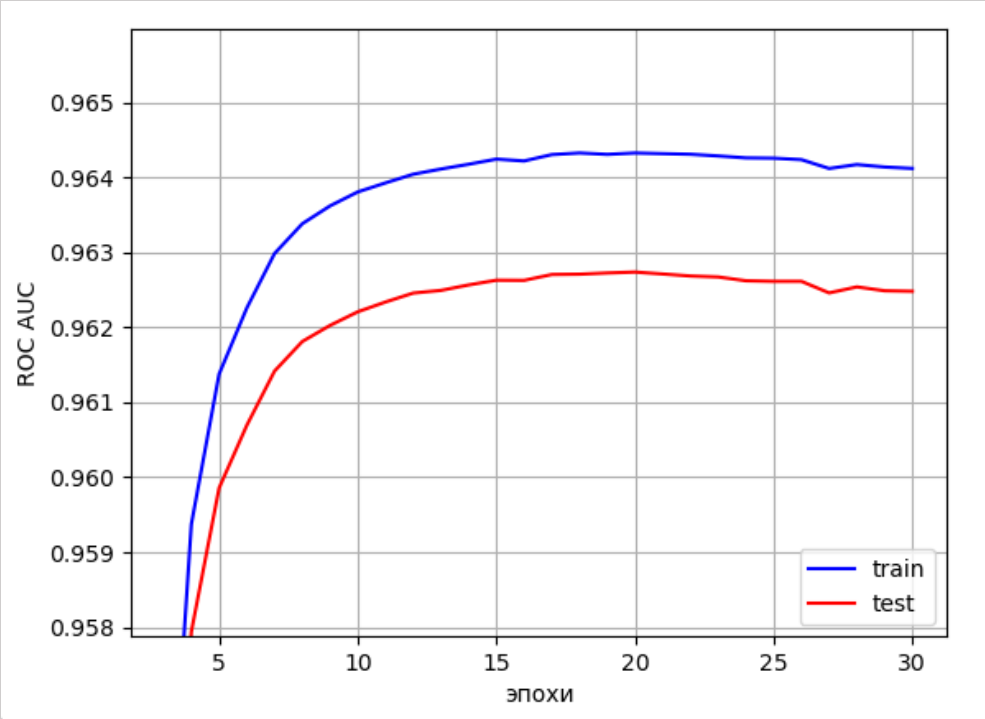


Рисунок 18. Тест 2. Обучение на датасете Epsilon.

В итоге, показатель ROC AUC достиг значения 0.96 на пятой эпохе, затем продолжил плавно расти и остановился на значении 0.9627 с 17-то по 21-ю эпоху. Затем началось плавное снижение, видимо стал проявлять себя процесс переобучения и ROC AUC снизился до 0.9624 к 30-той эпохе.

Тест подтвердил работоспособность полигармонического каскада с большим количеством параметров на датасете Epsilon, при его обучении по тому же алгоритму, как и каскада меньшего размера.

Полученное значение AUC в обоих тестах Epsilon получилось на достаточно высоком уровне (значительно выше, чем к примеру, в статье [11]).

**Заключение.**

Подведем итоги. В рамках данной работы был предложен вариант универсальной инициализации полигармонического каскада, выбор точек созвездий и начальных значений в них. Было показано, что следствием этого выбора может быть значительная оптимизация вычислений, необходимых для работы каскада. Практически проверена способность полигармонического каскада обучаться на датасетах MNIST, Higggs Boson и Epsilon. Подтверждена работоспособность алгоритмов обучения на каскадах различного размера, от 4-х до 500 слоёв. Подтверждена способность полигармонического каскада обучаться как на датасетах большого размера со сложными нелинейными зависимостями и уровнем шума (HIGGS), так и на датасетах с высокой размерностью входных данных (2000 признаков в Epsilon).

**Примечание:** Данное исследование проводилось независимо и вне рамок профессиональной деятельности в организации ООО “ЛИС”.

**Источники:**


1. Бахвалов Ю. Н. 2024. Решение регрессионной задачи машинного обучения на основе теории случайных функций. PREPRINTS.RU. https://doi.org/10.24108/preprints-3113020
2. Бахвалов Ю. Н. 2024. Пакеты полигармонических сплайнов, их объединение, эффективные процедуры вычисления и дифференцирования. PREPRINTS.RU. https://doi.org/10.24108/preprints-3113111
3. Бахвалов Ю. Н. 2025. Полигармонический каскад. PREPRINTS.RU. https://doi.org/10.24108/preprints-3113501
4. В.С. Пугачев, Теория случайных функций и её применение к задачам автоматического управления. Изд. 2-ое, перераб. и допол. — М.: Физматлит, 1960.
5. Baldi, P., Sadowski, P., Whiteson, D.: Searching for Exotic Particles in High-Energy Physics with Deep Learning. Nature Communications 5, 4308 (2014). https://doi.org/10.1038/ncomms5308
6. Dua, D., Graff, C.: UCI Machine Learning Repository (2019). University of California, Irvine, School of Information and Computer Sciences. [Доступ: July 31, 2025]. URL: https://archive.ics.uci.edu/ml/datasets/HIGGS
7. R.L. Harder and R.N. Desmarais: Interpolation using surface splines. Journal of Aircraft, 1972, Issue 2, pp. 189−191
8. Bookstein, F. L. (June 1989). "Principal warps: thin plate splines and the decomposition of deformations". IEEE Transactions on Pattern Analysis and Machine Intelligence. 11 (6): 567–585. doi:10.1109/34.24792
9. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. DOI:10.1109/5.726791
10. PASCAL Network of Excellence. Epsilon Dataset (PASCAL Large Scale Learning Challenge 2008) [Data set; ID 45575, Version 1]. OpenML, 2008. URL: https://www.openml.org/d/45575
11. Zhang, H., Si, S., & Hsieh, C. J. (2017). GPU-acceleration for Large-scale Tree Boosting. arXiv preprint arXiv:1706.08359
12. https://github.com/xolod7/polyharmonic-cascade.git
13. https://zenodo.org/records/16811633