Полигармонический каскад

Бахвалов Ю.Н., к.т.н., независимый исследователь, г. Череповец

Аннотация.

В статье представлена глубокая архитектура машинного обучения 'полигармонический каскад' – последовательность пакетов полигармонических сплайнов [3], где каждый слой строго обоснован через теорию случайных функций и принципы индифферентности [2]. Это позволяет аппроксимировать нелинейные функции любой сложности с сохранением глобальной гладкости и вероятностной интерпретации.

Применительно к полигармоническому каскаду предложен метод обучения, альтернативный градиентному спуску: вместо прямой оптимизации коэффициентов решается единая на батче глобальная линейная система по значениям функций в фиксированных «созвездиях» узлов. Это даёт синхронизированные обновления всех слоёв, сохраняет вероятностную интерпретацию отдельных слоёв и теоретическую согласованность с исходной моделью из первой статьи цикла [2], хорошо масштабируется: все вычисления сводятся к 2D-матричным операциям, эффективно исполняемым на GPU. Демонстрируется быстрое обучение без переобучения на MNIST.

Ключевые слова: машинное обучение, регрессия, случайная функция, полигармонический сплайн, пакет полигармонических сплайнов, полигармонический каскад, дифференцирование.

Polyharmonic cascade

Bakhvalov Yu.N., Ph.D., Independent Researcher, Cherepovets

Abstract.

This paper presents a deep machine learning architecture called the 'polyharmonic cascade' – a sequence of packages of polyharmonic splines [3], where each layer is rigorously justified using random function theory and principles of indifference [2]. This enables approximation of nonlinear functions of any complexity while preserving global smoothness and probabilistic interpretation.

For the polyharmonic cascade, an alternative learning method to gradient descent is proposed: instead of directly optimizing the coefficients, a single global linear system (dual Gauss–Newton/Tikhonov step) is solved on the function values in fixed "constellations" of nodes, all in a batch. This ensures synchronized updates across all layers, preserves the probabilistic interpretation of individual layers and theoretical consistency with the original model from the first paper in the series [2],

and scales well: all computations are reduced to 2D matrix operations, efficiently executed on a GPU. Fast learning without overfitting on MNIST is demonstrated.

Keywords: machine learning, regression, random function, polyharmonic spline, polyharmonic spline package, polyharmonic cascade, differentiation.

Статья посвящена масштабированию решения, полученного в [2] для аппроксимации многомерных нелинейных функций неограниченной сложности. В работе [2] было показано, что регрессионная задача машинного обучения может быть решена (и иметь точное решение) на основе теории случайных функций [12] и принципов индифферентности. Полученные в [2] выражения с некоторыми нюансами можно понимать, как разновидность полигармонического сплайна специального вида (сплайн тонкой пластины [10], [5]).

В работе [3] было показано, что если в решении из [2] задать некоторое фиксированное количество ключевых точек (названное “созвездием”), то в рамках этого созвездия можно сразу объединить большое количество функций (полигармонических сплайнов) в виде пакета (понятие, введенное в [3]), который можно эффективно вычислять и дифференцировать.

Также в [3] было показано, что решение из [2] имеет ограничения применимости, связанные как с резким ростом вычислительной сложности для больших обучающих выборок, так и с получением неоптимального результата (в определенных случаях, разобранных в [3]). Эти же ограничения относятся и к пакетам таких функций.

Для снятия этих ограничений в [3] было предложено (и приведены обоснования) объединять последовательность пакетов полигармонических сплайнов в виде многослойной вычислительной структуры. Также были описаны выражения как для прямого вычисления такой структуры (последовательно от первого слоя к последнему), так и для её дифференцирования (двигаясь по цепному правилу в обратном порядке от последнего слоя к первому).

Все выражения были получены в виде операций над матрицами, что позволяет их относительно легко реализовать и эффективно вычислять с использованием GPU.

Такую вычислительную структуру будем называть полигармоническим каскадом, как последовательным каскадным соединением пакетов полигармонических функций.

Очевидно, что возникает прямая аналогия между полигармоническим каскадом и алгоритмами искусственных многослойных нейронных сетей, как по структуре (последовательных слоёв), так и по предназначению. Однако,

если считать такую вычислительную структуру (последовательность полигармонических пакетов функций) одной из разновидностей многослойных нейронных сетей (пусть и специфического вида) и дать ей название, например, “полигармоническая нейронная сеть” и т.п., то это название может породить неверные ассоциации касательно существа и принципов работы алгоритма.

Между полигармоническим каскадом и алгоритмами искусственных многослойных нейронных сетей есть существенные отличия. Это касается его математического представления и обоснования. Базовые математические выражения, лежащие в основе полигармонического каскада, могут быть выведены из теории случайных функций ([12] и [2]) и принципов индифферентности. Математически полигармоническая сплайн-функция значительно отличается от (обычно принятой) математической модели нейрона (а полигармонический пакет от слоя нейронов).

Хотя композиции гауссовых процессов и ядерных функций исследовались в работах по глубоким гауссовым процессам (Damianou & Lawrence, 2013 [7]) и глубокому ядерному обучению (Wilson et al., 2016 [16]), в отличие от них, полигармонический каскад сохраняет строгое вероятностное обоснование каждого слоя через принципы симметрии и индифферентности. Полигармонические сплайны представляют собой канонический класс радиально-базисных функций с оптимальными свойствами гладкости и инвариантности (Buhmann 2003 [6]; Duchon 1977 [9]). В отличие от произвольных RBF (например, гауссовых), они не вводят априорного предпочтения к низкочастотным компонентам, что критически важно при отсутствии априорной информации. Искусственное объединение в пакет на основе заданного “созвездия” произвольных радиально-базисных функций (помимо полигармонического сплайна) может вызвать осцилляции и привести к неработоспособности алгоритма (зависит от спектрального разложения радиально-базисной функции, что было показано в [2]).

Но, наверное, главная особенность полигармонического каскада в отличном от нейронных сетей, но эффективном методе обучения. Дело в том, что провести напрямую по коэффициентам уравнений (без каких-либо ухищрений) процедуру его обучения методом градиентного спуска невозможно. Хотя не возникает никаких затруднений в том, чтобы описать для полигармонического каскада граф вычислений (в том числе используя какую-либо библиотеку машинного обучения), найти градиенты ошибок для всех его коэффициентов уравнений, но использование любой оптимизационной процедуры, основанной на движении вдоль градиента практически не даст никакого результата (если применять оптимизацию непосредственно к настраиваемым коэффициентам уравнений). Пространство настраиваемых параметров полигармонического каскада является крайне неудачным для

выполнения в нем оптимизационной процедуры, основанной на градиентном спуске.

Однако эффективная процедура обучения полигармонического каскада (с большим количеством слоев) существует и будет разобрана в этой статье.

Представим полигармонический каскад в виде схемы. Пусть у нас есть каскад из q последовательно соединенных полигармонических пакетов (рисунок 1). На схеме изображены первый и последний пакеты в каскаде, а также некоторый полигармонический пакет из середины с номером τ.

На вход первого пакета приходит порция данных на обработку в виде матрицы X_0 размерностью $(r \times n_0)$, каждая строчка которой это независимый вектор на обработку размером n_0 (что является количеством входов вычислительной системы и мерностью функции, которая в итоге полигармоническим каскадом вычисляется). Количество самих векторов в порции данных (в данном случае мы ее рассматриваем как батч для обучения) равняется r.

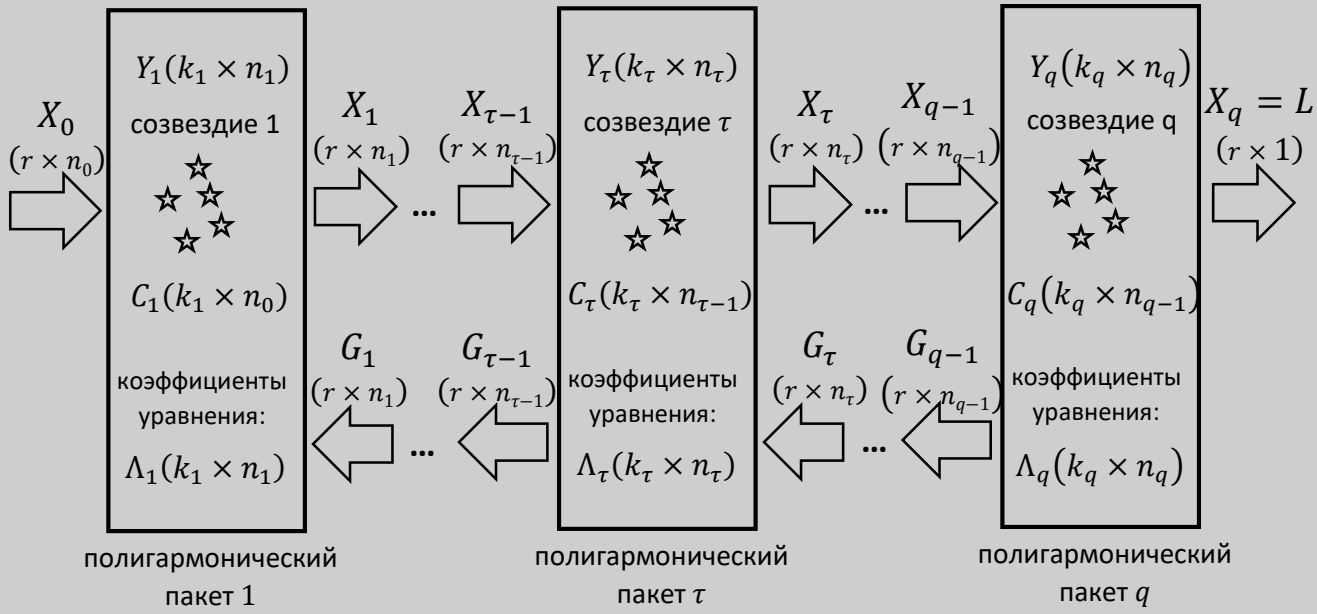

Рисунок 1. Полигармонический каскад

На выходе первого полигармонического пакета получаем матрицу X_1 размерностью $(r \times n_1)$, где n_1 – количество выходов первого полигармонического пакета (количество функций, который он вычисляет). Эта матрица поступает на вход второго полигармонического пакета и т.д. Для пакета с номером τ это будут соответственно матрица $X_{\tau-1}$ на входе и X_τ на выходе.

На выходе каскада с последнего пакета будет получена матрица X_q размерностью $(r \times 1)$, которая фактически будет являться вектором размером

r, который обозначим как $L = (l_1, l_2, \dots, l_r)$, каждый компонент которого l_i будет значением итогового результата обработки всем полигармоническим каскадом *i-ой* строки исходной матрицы X_0. Т.е. рассматриваем каскад, который имеет всего один выход на последнем слое и в целом вычисляет единственную функцию. После разбора алгоритма обучения для этого случая, рассмотрим, как алгоритм можно модифицировать, если выходов несколько.

Также полигармонический каскад позволяет пройтись теперь по нему в обратную сторону и вычислить матрицы производных $G_1, G_2, \dots, G_{q-1}$ (процедура была описана в [3]), что очень похоже на алгоритм обратного распространения ошибки в нейронных сетях ([1] и [13]). Однако в данном случае речь идет не про функцию ошибки, а о производных от самой вычисляемой полигармоническим каскадом функции. Во-вторых, эти матрицы производных вычисляются не для коэффициентов уравнений (в настройке которых и состоит итоговая цель), а по значениям с выходов пакетов, т.е. по значениям матриц $X_1, X_2, \dots, X_{q-1}$.

Таким образом, элементы, например, матрицы G_τ можем записать как $g_{it}^{(\tau)} = \frac{dl_i}{dx_{it}^{(\tau)}}$, где $\tau = \overline{1, q-1}$, $i = \overline{1, r}$, $t = \overline{1, n_\tau}$.

Соответственно размеры матриц $G_1, G_2, \dots, G_{q-1}$ будут полностью совпадать с размерами матриц $X_1, X_2, \dots, X_{q-1}$.

На рисунке не были обозначены матрицы G_0 и G_q для соответствующих им X_0 и X_q. Матрица G_0 может быть вычислена, но далее она в описываемом алгоритме не используется. Матрица же G_q использоваться будет, но очевидно, что она будет размерностью $(r \times 1)$ и представляет из себя вектор размером r состоящий из единиц (поскольку $dx_{i1}^{(q)} = dl_i$ в ее случае будут равны).

Каждый полигармонический пакет на схеме описан с помощью трех матриц. Для пакета с номером τ это будут матрицы Y_τ, C_τ и Λ_τ. Матрица C_τ размером $(k_\tau \times n_{\tau-1})$ описывает точки созвездия в пакете, матрица $Y_\tau(k_\tau \times n_\tau)$ – наборы значений в этих точках, а матрица $\Lambda_\tau(k_\tau \times n_\tau)$ представляет собой набор коэффициентов, которые непосредственно используются при вычислениях (хотя она и выводится через C_τ и Y_τ).

Опишем теперь все выражения, полученные в [2] и [3], для выполнения всех представленных на схеме и связанных с ними вычислений.

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

Первым шагом находим матрицу всевозможных расстояний между строчками векторами в $X_{\tau-1}$ и точками созвездия (обозначим ее как M_τ).

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

где

$$N_{\tau x} = (X_{\tau-1} \circ X_{\tau-1}) J_{n_{\tau-1},1} \,, \quad (2)$$

$$N_{\tau c} = (C_\tau \circ C_\tau) J_{n_{\tau-1},1} \,, \quad (3)$$

$\circ$ - произведение Адамара,

J_{1,k_τ} – вектор строка из единиц размера k_τ,

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

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

C_τ – матрица точек созвездия τ – *го* пакета.

Фактически выражение (1) это записанная в матричном виде теорема косинусов.

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

$$k_{ip}^\tau = m_{ip}^\tau \left(\ln\left(m_{ip}^\tau\right) - b\right) + c \,, \quad (4)$$

где

b и c – коэффициенты, значения которых оценивались в [2],

k_{ip}^τ – элемент матрицы K_τ (не путать с k_τ),

m_{ip}^τ – элемент матрицы M_τ,

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

Выражение (4) по сравнению с его использованием в [2] и [3] чуть упрощено. Убран множитель $\frac{1}{2}$ перед выражением а также двойка перед коэффициентом b. Это ничего по существу не меняет, поскольку сами коэффициенты b и c можно взять по-разному. В вычислительных экспериментах при использовании (4) неплохо себя показали значения *b=10*, *c=1000*.

Далее находим X_τ как матричное произведение K_τ и Λ_τ :

$$X_\tau = K_\tau \Lambda_\tau \quad (5)$$

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

Для начал найдем матрицу Θ_τ. Получается она путем поэлементного преобразования из матрицы M_τ которая была найдена в (1) с помощью выражения:

$$\theta_{ip}^{\tau} = \ln\left(m_{ip}^{\tau}\right) - b + 1 \,, \tag{6}$$

где

b – тот же коэффициент из (4),

θ_{ip}^{τ} – элемент матрицы Θ_τ ,

m_{ip}^{τ} – элемент матрицы M_τ,

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

Затем найдем матрицу Ψ_τ:

$$\Psi_\tau = \Theta_\tau \circ (G_\tau \Lambda_\tau^T) \tag{7}$$

Матрицы Θ_τ и Ψ_τ получатся размером $(r \times k_\tau)$.

И теперь выразим $G_{\tau-1}$:

$$G_{\tau-1} = X_{\tau-1} \circ \left(\Psi_\tau J_{k_\tau,1} J_{1,n_{\tau-1}}\right) - \Psi_\tau C_\tau \,, \tag{8}$$

где

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

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

Таким образом в полигармоническом каскаде с помощью выражений (1) – (5) осуществляются вычисления в прямом направлении, а с помощью (6) – (8) вычисления производных в обратном.

Но прежде чем перейти к описанию алгоритма обучения, покажем, что вычисления в прямом направлении можно представить и иным способом. В выражении (5) используется матрица Λ_τ, которая в свою очередь определяется через C_τ и Y_τ. Покажем каким образом она вычисляется.

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

$$M_\tau^{(C)} = N_{\tau c} J_{1,k_\tau} + J_{k_\tau,1} N_{\tau c}^T - 2 C_\tau C_\tau^T \,, \tag{9}$$

где

$N_{\tau c} = (C_\tau \circ C_\tau) J_{n_{\tau-1},1}$ (то же выражение что и (3)),

J_{1,k_τ} – вектор строка из единиц размера k_τ,

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

Определим матрицу U_τ как:

$$U_\tau = \left(K_\tau^{(C)} + \sigma_\tau^2 E\right)^{-1}, \tag{10}$$

где

матрица $K_\tau^{(C)}$ вычисляется из $M_\tau^{(C)}$ с помощью (4),

σ_τ^2 – дисперсии случайных величин,

(более подробно их смысл разобран в [2])

E – единичная матрица.

Значение σ_τ^2 играет роль “коэффициента регуляризации”, превращая задачу интерполяции в задачу аппроксимации и обратно (в зависимости от значений) при вычислении функций по точкам созвездия внутри пакета. Поэтому ее значение может зависеть от того, как именно или по какому принципу были выбраны точки внутри созвездия. Если исключается совпадение точек внутри созвездия, а сами они расположены между собой равномерно, то σ_τ^2 можно взять близким или равным нулю. В этом случает внутри пакета по отношению к значениям в Y_τ будет выполняться задача интерполяции, что впрочем, совсем не отменяет возможность выполнения аппроксимации по отношению к обучающей выборке полигармоническим каскадом в целом.

Матрица Λ_τ вычисляется как произведение U_τ и Y_τ:

$$\Lambda_\tau = U_\tau Y_\tau \tag{11}$$

Используя (5) и (11) получим:

$$X_\tau = K_\tau \Lambda_\tau = K_\tau U_\tau Y_\tau \tag{12}$$

Если ввести дополнительно матрицу H_τ:

$$H_\tau = K_\tau U_\tau \tag{13}$$

Тогда можем (12) записать как:

$$X_\tau = H_\tau Y_\tau \tag{14}$$

Будем в рамках описываемого алгоритма работы и обучения полигармонического каскада считать расположение в пакетах точек созвездий (матрицы C_τ) некоторым образом заданными при инициализации и в дальнейшем неизменными (будем считать их значения константами).

Хотя теоретически значения в матрицах $C_1, C_2, .., C_q$ можно было бы также подвергнуть в процессе обучения настройке. Но как показали вычислительные эксперименты, их фиксация не лишает полигармонический каскад возможности обучаться. Алгоритму их инициализации было посвящено отдельное исследование, результаты которого будут изложены в отдельной статье. В фиксации матриц C_τ есть также дополнительное преимущество, заключающееся в том, что в этом случае матрица U_τ из (10) для каждого полигармонического пакета также всегда остается фиксированной, а выражения (9) и (10) требуют лишь единственного однократного выполнения. Тем не менее, алгоритмы, в которых значения в матрицах $C_1, C_2, .., C_q$ также подвергаются изменениям в процессе обучения, требуют дополнительного исследования.

Итак, выражения (12) – (14), которыми можно подменить выражение (5), показывают, что полигармонический каскад может быть представлен таким образом (хотя и чуть более громоздким, с привлечением матриц U_τ), что непосредственно коэффициенты уравнений (матрицы Λ_τ) исключаются из вычислений и их роль начинают выполнять значения функций (матрицы Y_τ) в точках созвездий.

При таком (12) – (14) представлении и условии фиксации матриц C_τ, получается, что обучение полигармонического каскада сведется лишь к тому, чтобы подобрать значения матриц $Y_1, Y_2, .., Y_q$, или, иными словами, правильно указать значения функций полигармонических пакетов в точках созвездий.

Теперь, если обозначить весь каскад, как вычислитель некоторой функции F, то для вычисления некоторого значения на выходе l_i вектора L , каскад можно представить как:

$$l_i = F\left(x_i^{(0)}, Y_1, Y_2, .., Y_q\right), \tag{15}$$

где

$x_i^{(0)}$ – i-*я* строка матрицы X_0

Таким образом в (15) представляем матрицы $Y_1, Y_2, .., Y_q$ в качестве дополнительных входных параметров, как бы поступающих на вход каскада, а сам каскад в таком случае как фиксированную функцию.

Предположим, что мы хотим так изменить матрицы $Y_1, Y_2, .., Y_q$, чтобы при том же $x_i^{(0)}$ на входе, получить новое значение на выходе, равное l_i^* . Если мы смогли найти такие изменения матриц $\Delta Y_1, \Delta Y_2, .., \Delta Y_q$, что удалось выполнить это условие, тогда это можно записать как:

$$l_i^* = F\left(x_i^{(0)}, Y_1 + \Delta Y_1, Y_2 + \Delta Y_2, .., Y_q + \Delta Y_q\right) \tag{16}$$

Поскольку функция *F* дифференцируема по любому элементу из любой матрицы $Y_1, Y_2, .., Y_q$, то (16) можно представить как:

$$F\left(x_i^{(0)}, Y_1 + \Delta Y_1, Y_2 + \Delta Y_2, .., Y_q + \Delta Y_q\right) =$$

$$= F\left(x_i^{(0)}, Y_1, Y_2, .., Y_q\right) + \sum_{\tau=1}^{q} \sum_{p=1}^{k_\tau} \sum_{t=1}^{n_\tau} \left(\frac{dl_i}{dy_{pt}^{(\tau)}} \Delta y_{pt}^{(\tau)}\right) + e_i\,, \tag{17}$$

где

$y_{pt}^{(\tau)}$ – элементы матрицы Y_τ,

$\Delta y_{pt}^{(\tau)}$ – элементы матрицы ΔY_τ,

e_i – ошибка.

Таким образом мы заменили функцию в (16) суммой значения функции в (15) с касательной гиперплоскостью ее сечения в $x_i^{(0)}$. Если *F* непрерывна и дифференцируема (а функция, создаваемая полигармоническим каскадом именно такая), то при бесконечно малых $\Delta y_{pt}^{(\tau)}$, такое представление будет абсолютно верным. Но при больших значениях $\Delta y_{pt}^{(\tau)}$ появится ошибка, которая в (17) обозначена как e_i .

Но как можно вычислить $\frac{dl_i}{dy_{pt}^{(\tau)}}$ из (17)?

Выражение (14), которое записано в виде произведения матриц можно представить в эквивалентном виде как выражение для их элементов:

$$x_{it}^{(\tau)} = \sum_{p=1}^{k_\tau} h_{ip}^{(\tau)} y_{pt}^{(\tau)}\,, \tag{18}$$

$x_{it}^{(\tau)}$ – элементы матрицы X_τ,

$h_{ip}^{(\tau)}$ – элементы матрицы H_τ,

$y_{pt}^{(\tau)}$ – элементы матрицы Y_τ.

Из (18) можно получить, что $\frac{dx_{it}^{(\tau)}}{dy_{pt}^{(\tau)}} = h_{ip}^{(\tau)}$.

Как уже было сказано ранее $\frac{dl_i}{dx_{it}^{(\tau)}} = g_{it}^{(\tau)}$ (элемент матрицы G_τ).

Тогда мы можем выразить:

$$\frac{dl_i}{dy_{pt}^{(\tau)}} = \frac{dl_i}{dx_{it}^{(\tau)}} \frac{dx_{it}^{(\tau)}}{dy_{pt}^{(\tau)}} = g_{it}^{(\tau)} h_{ip}^{(\tau)} \tag{19}$$

Используя (16), (17) и (19), а также составив эти уравнения для обработки каскадом последовательно всех строк $x_1^{(0)}, x_2^{(0)}, \dots, x_r^{(0)}$ матрицы X_0 получим систему из *r* уравнений:

$$\begin{cases} \sum_{\tau=1}^{q} \sum_{p=1}^{k_\tau} \sum_{t=1}^{n_\tau} \left(g_{1t}^{(\tau)} h_{1p}^{(\tau)} \Delta y_{pt}^{(\tau)} \right) + e_1 = l_1^* - F\left(x_1^{(0)}, Y_1, Y_2, .., Y_q \right) \\ \dots \\ \sum_{\tau=1}^{q} \sum_{p=1}^{k_\tau} \sum_{t=1}^{n_\tau} \left(g_{rt}^{(\tau)} h_{rp}^{(\tau)} \Delta y_{pt}^{(\tau)} \right) + e_r = l_r^* - F\left(x_r^{(0)}, Y_1, Y_2, .., Y_q \right) \end{cases} \tag{20}$$

Или иначе:

$$\begin{cases} \sum_{\tau=1}^{q} \sum_{p=1}^{k_\tau} \sum_{t=1}^{n_\tau} \left(g_{1t}^{(\tau)} h_{1p}^{(\tau)} \Delta y_{pt}^{(\tau)} \right) + e_1 = l_1^* - l_1 = \Delta l_1 \\ \dots \\ \sum_{\tau=1}^{q} \sum_{p=1}^{k_\tau} \sum_{t=1}^{n_\tau} \left(g_{rt}^{(\tau)} h_{rp}^{(\tau)} \Delta y_{pt}^{(\tau)} \right) + e_r = l_r^* - l_r = \Delta l_r \end{cases}, \tag{21}$$

где

Δl_i – элемент $\Delta L = L^* - L$,

L^* - вектор желаемых значений l_i^* на выходе каскада.

Получив выражение (21) теперь можно сформулировать оптимизационную задачу: поиск таких минимальных значений $\Delta y_{pt}^{(\tau)}$ (т.е. постараться изменить значения в точках созвездий, которые описываются матрицами $Y_1, Y_2, .., Y_q$, наименьшим образом) которые минимизируют значения ошибок $e_1, e_2, \dots, e_r$ при условии выполнения уравнений (21). Т.е. нужно соблюсти баланс между минимизацией ошибок с одной стороны, но с другой стороны сами значения $\Delta y_{pt}^{(\tau)}$ желательно подобрать минимально возможными.

Можно обратить внимание, что значения ошибок $e_1, e_2, \dots, e_r$ при таком подходе уже состоят из двух компонентов, играя как роль ошибок значений на выходе каскада по сравнению с желаемыми l_i^* так и роль ошибок (величины которых неизвестны) в представлении модели выражением (17) . Поэтому приоритетно минимизировать $e_1, e_2, \dots, e_r$ до нуля за одну итерацию может лишь навредить процессу обучения.

Достигнуть все перечисленные цели позволяет целевая функция вида:

$$\frac{1}{\alpha}\sum_{i=1}^{r} e_i^2 + \sum_{\tau=1}^{q}\sum_{p=1}^{k_\tau}\sum_{t=1}^{n_\tau}\left(\Delta y_{pt}^{(\tau)}\right)^2 \to \min \qquad (22)$$

В (22) был добавлен коэффициент α который позволяет регулировать приоритет между минимизацией ошибок и значений $\Delta y_{pt}^{(\tau)}$.

В итоге ((21) и (22)) была получена задача квадратичного программирования с системой ограничений в виде равенств. Эта задача решается методом множителей Лагранжа.

При беглом взгляде на (21) и (22) могут возникнуть сомнения в результативности предложенного подхода. Каждое уравнение в (21) будет содержать количество коэффициентов равное количеству настраиваемых параметров во всем полигармоническом каскаде. А количество самих уравнений равно r – количеству обучаемых примеров в батче. Если, например, в батче тысяча примеров, то чтобы просто записать коэффициенты из (21) в виде матрицы то потребуется памяти в тысячу раз больше чем количество настроечных параметров в самом полигармоническом каскаде. Также, можно предположить, что потребуются значительные вычислительные ресурсы для нахождения при таком подходе решения. Однако, как будет показано дальше, такие опасения неверны.

Запишем функцию Лагранжа, обозначив ее $\hat{L}$, чтобы была разница в обозначении по сравнению с уже использованным ранее L в качестве вектора значений на выходе каскада. Множители Лагранжа обозначим вместо λ_i как β_i, а последовательность их значений как вектор B, чтобы не путать с матрицей Λ и её элементами, которая уже используется.

Функция Лагранжа:

$$\hat{L}\left(e_1, e_2, \dots, e_r, \Delta y_{11}^{(1)}, \dots, \Delta y_{pt}^{(\tau)}, \dots, \Delta y_{k_q n_q}^{(q)}, \beta_1, \beta_2, \dots, \beta_r\right) =$$
$$= \frac{1}{\alpha}\sum_{i=1}^{r} e_i^2 + \sum_{\tau=1}^{q}\sum_{p=1}^{k_\tau}\sum_{t=1}^{n_\tau}\left(\Delta y_{pt}^{(\tau)}\right)^2 +$$

$$+\sum_{i=1}^{r}\beta_i\left(\Delta l_i - e_i - \sum_{\tau=1}^{q}\sum_{p=1}^{k_\tau}\sum_{t=1}^{n_\tau}\left(g_{it}^{(\tau)}h_{ip}^{(\tau)}\Delta y_{pt}^{(\tau)}\right)\right) \quad (23)$$

Если в (23) принять $\frac{d\hat{L}}{d\beta_i} = 0$, то получим снова систему уравнений (21).

Если принять $\frac{d\hat{L}}{de_i} = 0$, то найдем, как будут связаны между собой в решении значения e_i и β_i.

$$e_i = \frac{\alpha}{2}\beta_i \quad (24)$$

Если принять $\frac{d\hat{L}}{d\Delta y_{pt}^{(\tau)}} = 0$, то получим:

$$\Delta y_{pt}^{(\tau)} = \frac{1}{2}\sum_{i=1}^{r}\beta_i g_{it}^{(\tau)}h_{ip}^{(\tau)} \quad (25)$$

Соединив вместе (21), (24) и (25) получим систему уравнений:

$$\begin{cases} \sum_{\tau=1}^{q}\sum_{p=1}^{k_\tau}\sum_{t=1}^{n_\tau}\left(g_{1t}^{(\tau)}h_{1p}^{(\tau)}\sum_{i=1}^{r}\beta_i g_{it}^{(\tau)}h_{ip}^{(\tau)}\right) + \alpha\beta_1 = 2\Delta l_1 \\ \dots \\ \sum_{\tau=1}^{q}\sum_{p=1}^{k_\tau}\sum_{t=1}^{n_\tau}\left(g_{rt}^{(\tau)}h_{rp}^{(\tau)}\sum_{i=1}^{r}\beta_i g_{it}^{(\tau)}h_{ip}^{(\tau)}\right) + \alpha\beta_r = 2\Delta l_r \end{cases} \quad (26)$$

Таким образом решение оптимизационной задачи будет представлять из себя решение системы уравнений (26), чтобы найти $\beta_1, \beta_2, \dots, \beta_r$, используя которые в (25) можно вычислить любое $\Delta y_{pt}^{(\tau)}$, т.е. найти матрицы изменений значений в точках созвездий $\Delta Y_1, \Delta Y_2, .., \Delta Y_q$.

В (26) можно поменять последовательности суммирования местами и записать систему уравнений в виде:

$$\begin{cases} \sum_{i=1}^{r}\beta_i\sum_{\tau=1}^{q}\left(\sum_{p=1}^{k_\tau}h_{1p}^{(\tau)}h_{ip}^{(\tau)}\right)\left(\sum_{t=1}^{n_\tau}g_{1t}^{(\tau)}g_{it}^{(\tau)}\right) + \alpha\beta_1 = 2\Delta l_1 \\ \dots \\ \sum_{i=1}^{r}\beta_i\sum_{\tau=1}^{q}\left(\sum_{p=1}^{k_\tau}h_{rp}^{(\tau)}h_{ip}^{(\tau)}\right)\left(\sum_{t=1}^{n_\tau}g_{rt}^{(\tau)}g_{it}^{(\tau)}\right) + \alpha\beta_r = 2\Delta l_r \end{cases} \quad (27)$$

Но (27) можно записать в виде операций над матрицами:

$$\left(\sum_{\tau=1}^{q}\left((H_\tau H_\tau^T)\circ(G_\tau G_\tau^T)\right)+\alpha E\right)\mathrm{B}=2\Delta L\,, \tag{28}$$

где

E – единичная матрица размером r,

B – вектор столбец $\beta_1,\beta_2,\dots,\beta_r$.

Кстати, именно возможность преобразования из (26) в (27) и (28) позволяет в рассматриваемой задаче радикально сократить и размеры памяти, требуемые для вычислений, так и сам объем вычислений для решения по сравнению с решением задачи квадратичного программирования, когда коэффициенты в уравнениях системы ограничений заданы произвольным образом.

Обозначим выражение под знаком суммы в (28) в виде отдельной матрицы:

$$\Omega_\tau=(H_\tau H_\tau^T)\circ(G_\tau G_\tau^T) \tag{29}$$

Каждая матрица Ω_τ связана только со своим полигармоническим пакетом (со своим слоем в полигармоническом каскаде) и для каждого из них может быть вычислена независимо. Но не смотря на то, что каждый из пакетов может иметь разное количество и входов и выходов и разное количество точек в созвездиях (разные размерности Y_τ) все матрицы $\Omega_1,\Omega_2,\dots,\Omega_q$ будут одной и той же размерности $(r\times r)$.

Предположим, что все матрицы (29) для каждого пакета в каскаде были вычислены, тогда из (28) и (29) мы можем вывести как получить B:

$$\mathrm{B}=2\left(\Omega_1+\Omega_2+\cdots+\Omega_q+\alpha E\right)^{-1}\Delta L \tag{30}$$

Полученный в (30) вектор B затем можно использовать независимо в каждом полигармоническом пакете чтобы вычислить все элементы $\Delta y_{pt}^{(\tau)}$ из матрицы ΔY_τ используя выражение (25). Таким образом вектор B (и процедура его вычисления (30)) будет выполнять синхронизирующую роль между различными пакетами в каскаде в процессе обучения, делая изменения всех настроечных параметров из разных пакетов зависимыми от изменения друг друга.

Рассмотрим (25) подробнее. Из выражения следует, что элемент $\Delta y_{pt}^{(\tau)}$ матрицы ΔY_τ с индексами (p,t) равен скалярному произведению p-го столбца

матрицы H_τ и t-го столбца матрицы G_τ совместно с вектором B. Следовательно (25) можно представить в виде операций с матрицами:

$$\Delta Y_\tau = \frac{1}{2} H_\tau^T \left(G_\tau \circ \left(\mathrm{B} J_{1,n_\tau} \right) \right), \tag{31}$$

где

J_{1,n_τ} – вектор строка из единиц размера n_τ

Можно обратить внимание, что если в (30) убрать множитель 2 перед выражением и одновременно в (31) убрать $\frac{1}{2}$, то итоговый результат не изменится.

Дальше находим новые значения Y_τ, прибавляя к ним ΔY_τ и обновляем Λ_τ через выражение (11).

Итак, опишем теперь полностью получившийся алгоритм по пунктам.

1. Инициализация полигармонического каскада. Выбор количества пакетов, количества функций в них, задание точек созвездий (описываемые матрицами C_τ) и начальных значений в них (описываемые матрицами Y_τ). Вычисление для каждого пакета своей матрицы U_τ используя (9), (4), (10). Вычисление в (11) начальных значений Λ_τ. Выбор коэффициента α, влияющего на скорость обучения, который будет использоваться в (30). Если взять α слишком большим, то обучение может сильно замедлится. Если слишком маленьким, работа алгоритма может нарушиться. (процедура инициализации C_τ и Y_τ, в рамках данной статьи не рассматривается, чему посвящено отдельное исследование)
2. Разделение обучающей выборки на батчи и последовательное обучение на них по следующим далее пунктам.
3. Последовательная обработка данных батча пакетами в каскаде начиная с первого и до последнего с помощью выражений (1) – (5), сохранение матриц X_τ и M_τ, а также вычисление в (13) и сохранение для каждого из пакетов матрицы H_τ. Вычисление на выходе каскада вектора ΔL (ошибка между желаемыми значениями и полученным результатом).
4. Задание матрицы G_q как вектора столбца состоящего из единиц. Вычисление в обратном направлении матриц производных $G_{q-1},..,G_1$ с помощью выражений (6) – (8).
5. Вычисление для каждого пакета матрицы Ω_τ с помощью (29). Сложение всех этих матриц в одну и вычисление вектора B в (30). В выражении (30) используется обращение матрицы, но в принципе, чтобы получить B достаточно решить систему линейных уравнений.

6. Вычисление для каждого пакета ΔY_τ с помощью полученного в предыдущем пункте вектора B в (31). Обновление матрицы Y_τ и вычисление новых коэффициентов уравнений в пакете Λ_τ в (11).
7. Повторение пунктов 3-7 для следующего батча и т.д.

Алгоритм можно несколько оптимизировать, найдя матрицы Θ_τ с помощью (6) еще в пункте 3 и сохранить именно их вместо M_τ. А матрицу K_τ в п.3 найти не через (4) а по выражению:

$$K_\tau = M_\tau \circ \left(\Theta_\tau - J_{r,k_\tau}\right) + c \qquad (32)$$

где J_{r,k_τ} – матрица единиц размерностью $(r \times k_\tau)$

Тогда в п.4 матрицы Θ_τ уже будут в наличии и (6) вычислять не потребуется.

Таким образом в пунктах 1-7 были изложены основные принципы алгоритма обучения полигармонического каскада. Все выражения получились в виде операций над двумерными матрицами, следовательно, такой алгоритм может быть эффективно реализован на GPU.

В отличие от приближённых методов масштабирования ядерных машин, таких как метод Нистрома (Williams & Seeger, 2001 [15]) или быстрые мультипольные методы (Beatson & Greengard, 1997 [4]), предложенный каскад сохраняет точную форму ядра на каждом уровне, не жертвуя глобальной гладкостью ради вычислительной эффективности.

Предложенный итеративный алгоритм формально напоминает метод Гаусса–Ньютона (Dennis & Schnabel, 1996 [8]), поскольку на каждой итерации нелинейная система линеаризуется и решается задача наименьших квадратов. Однако ключевое отличие заключается в выборе пространства оптимизации: вместо прямой настройки коэффициентов разложения (что нарушило бы вероятностную интерпретацию), мы оптимизируем значения скрытых функций в фиксированных точках созвездия. Аналогично, штраф за изменения этих значений можно рассматривать как обобщение регуляризации по Тихонову (Tikhonov & Arsenin, 1977 [14]), применённой не к весам, а к промежуточным состояниям модели.

Однако, о чем было сказано в начале, на выходе каскада в последнем слое был всего один выход, т.е. каскад в целом вычислял лишь единственную функцию. На практике же машинного обучения, может потребоваться вычислять значения на нескольких выходах, например, в задачах классификации, где каждый выход соответствует своему классу.

Адаптировать алгоритм под ситуацию с несколькими выходами можно с помощью различных подходов.

Очевидным первым шагом является поставить на выходе пакет с несколькими выходами. Пусть их количество является s. Тогда $X_q = L$ будет являться матрицей размерностью $(r \times s)$. Такой же размерности тогда должны быть и матрицы G_q и ΔL, но что уже напрямую не вписывается в предложенный алгоритм.

Одним из вариантов возможного решения может быть задание на выходе полигармонического каскада некоторой целевой функции. Ее значения таким образом подменят собой тот единственный выход каскада, под который разработан алгоритм. Т.е. значения этой функции породят рассчитанные и желаемые вектора L и L^* и как следствие ΔL требуемой размерностью $(r \times 1)$. Производные же этой функции по X_q заполнят значениями матрицу G_q размерностью $(r \times s)$. Этот подход, как показали вычислительные эксперименты работоспособен, но он менее эффективен и замедляет обучение, чем те, которые будут описаны ниже.

После вычислительных экспериментов были найдены две модификации алгоритма, позволяющие адаптировать предложенный алгоритм обучения к полигармоническому каскаду с несколькими выходами.

Первая модификация алгоритма обучения основана на следующих правилах:

1. Для каждого обучающего примера из батча случайным образом выбирается только один из выходов полигармонического каскада и значение l_i берется от него. Желаемое значение l^* также берется аналогично для этого же выхода.
2. Матрицы производных $G_1, G_2, \dots, G_{q-1}$ находятся только по выбранным выходам. Т.е. первая строка этих матриц – это производные по выходу полигармонического каскада, который был выбран для первого обучающего примера. Вторая строка матриц – производные, вычисленные по выходу, который был выбран для второго обучающего примера и т.д.

При таком подходе один и тот же обучающий пример может быть включен (хотя не обязательно) в один и тот же батч несколько раз, если каждый раз у него был выбран разный выход каскада.

Этот подход можно формализовать следующим образом.

Зададим G_q как бинарную матрицу:

$$G_q \in \{0,1\}^{r \times s}, \text{где для всех } i \in \{1, \dots, r\} \qquad (33)$$

$$\exists! \, j \in \{1, \dots, s\}, \text{такой что её элемент } g_{ij}^{(q)} = 1$$

Т.е. G_q такая матрица, где каждая строка содержит ровно один элемент, равный 1, а остальные 0. Остальные матрицы $G_1, G_2, \dots, G_{q-1}$ вычисляются по тем же выражениям, что и раньше.

Матрицы L (на выходе каскада) и L^* (из обучающей выборки) будут размерностью $(r \times s)$. Получить из них ΔL размерностью $(r \times 1)$ можно следующим образом:

$$\Delta L = \left((L^* - L) \circ G_q\right) J_{s,1} \tag{34}$$

где $J_{s,1}$ – вектор столбец из единиц размером s

Полученный вектор ΔL используется затем в алгоритме так же как и раньше. Далее все действия в алгоритме обучения аналогичны.

Рассмотрим теперь второй вариант модификации алгоритма, позволяющий решать задачу обучения полигармонического каскада, когда у него несколько выходов.

Вычисление каскадом значений на каждом из своих выходов можно считать отдельной независимой функцией и по каждой из них выполнять отдельное независимое обучение. Причем структура каскада, количество пакетов, сколько у каждого из них входов и выходов, заданные в них созвездия (либо количество точек в них) можно считать для всех функций одинаковыми. В этом случае, чтобы масштабировать систему на несколько выходов, достаточно просто перейти от операций с двумерными матрицами к трехмерным тензорам, добавив к ним еще одну координату. Такой подход особенно эффективен при исполнении операций на GPU.

Причем не все матрицы в каскаде нужно превращать в трехмерные тензоры. Исходная матрица X_0 очевидно останется двумерной $(r \times n_0)$. Также и матрицы $C_1, C_2, .., C_q$ можно оставить двумерными (к примеру, если взять произвольную C_τ , то она будет иметь размерность $(k_\tau \times n_{\tau-1})$ как и раньше). Такими же останутся и матрицы U_τ размерностью $(k_\tau \times k_\tau)$. Хотя теоретически и их (C_τ и U_τ) можно превратить в трехмерные тензоры, если для вычисления каскадом каждой из независимых функций по какой-то причине хотим задать в пакетах индивидуальные созвездия.

Допустим, что так же, как и предыдущем разобранном случае количество выходов каскада равно s. Тогда размерность тензоров Λ_τ и Y_τ будет равной $(s \times k_\tau \times n_\tau)$. Тензоры X_τ, начиная с выхода первого пакета будут размерностью $(s \times r \times n_\tau)$.

На выходе каскада будет получен тензор $L = X_q$ размерностью $(s \times r \times 1)$, в то время как часть батча, соответствующая желаемым значениям на

выходе, L^* будет двумерной матрицей размерностью $(r \times s)$, следовательно ее нужно будет транспонировать и превратить в тензор $(s \times r \times 1)$ и затем найти разность между ними - тензор ΔL. В то же время G_q нужно взять тензором размерности $(s \times r \times 1)$ состоящим из единиц.

Все вычисления при обучении происходят по тому же алгоритму и тем же выражениям что и для случая с одним выходом, только $H_\tau, G_\tau, \Omega_\tau$ будут трехмерными тензорами. В выражении (30) нужно будет найти s обратных матриц, либо решить систему уравнений s раз, результатом чего будет тензор B размерностью $(s \times r \times 1)$.

Ну и соответственно дальше для каждого пакета через (31) находятся тензоры ΔY_τ размерностью $(s \times k_\tau \times n_\tau)$. Обновляем Y_τ и Λ_τ.

В данном контексте, когда говорим об операциях, где смешиваются одновременно двумерные матрицы и трехмерные тензоры, например, операция матричного умножения двумерной матрицы U_τ размерностью $(k_\tau \times k_\tau)$ и трехмерного тензора Y_τ размерностью $(s \times k_\tau \times n_\tau)$ в (11) , подразумеваем, что происходят s независимых матричных умножений матрицы U_τ и каждого из s компонентов тензора Y_τ (к слову в библиотеке машинного обучения PyTorch такое совмещение в операциях тензоров разной размерности происходит автоматически, поэтому при переходе от части двумерных матриц к трехмерным тензорам код для вычислений остается неизменным).

Таким образом, были рассмотрены две модификации алгоритма обучения, для полигармонического каскада, когда у него несколько выходов. Второй вариант при проведении вычислительных экспериментов показал себя более эффективным и приводил к более быстрому и точному обучению, несмотря на то, что он требует фактически дублирования всего каскада и его независимого обучения для каждого выхода (на GPU это весьма эффективно реализуется). Преимуществом первого подхода является возможность его использования в случае, когда количество выходов каскада требуется очень большим, когда рассматривать каждый из выходов как отдельную независимую функцию не выглядит эффективным.

В рамках проведенных вычислительных экспериментов полигармонический каскад, как архитектура машинного обучения, включая описанный в данной статье алгоритм, был реализован на языке Python с использованием библиотеки PyTorch.

Было подтверждено, что предложенный в статье алгоритм подходит для обучения полигармонического каскада, состоящего как из всего нескольких,

так и включающего десятки последовательных полигармонических пакетов (от нескольких сотен, до десятков миллионов настроечных параметров).

Ниже дано описание примера обучения полигармонического каскада на наборе данных MINST [11] (был загружен через библиотеку torchvision).

При обучении данным MINST никак не учитывалось, что перед нами структура в виде изображения 28x28 (как в сверточных сетях). Просто на вход поступали вектора, содержащие 784 параметра (784 входа каскада).

Для обучения данному датасету использовался полигармонический каскад, состоящий из четырех последовательных пакетов. Первый пакет принимал на вход 784 параметра и вычислял на входе 100 функций. Второй пакет принимал на вход 100 параметров и вычислял 20 функций. Третий пакет принимал на вход 20 параметров и вычислял 20 функций. И четвертый пакет принимал 20 параметров на вход и вычислял одну функцию. Вся структура с помощью второго метода модификации была масштабирована на 10 выходов.

С учетом всех заданных созвездий общее число настроечных параметров в каскаде (учет всех матриц Y_τ) составило около 1.6 миллиона.

Обучение происходило с помощью видеокарты NVIDIA GeForce RTX 3070.

График обучения представлен на рисунке 2.

Из графика видно, что полигармонический каскад обучился практически сразу же на первой эпохе, достигнув показателя точности 97.52%. Затем показатель точности на тестовом множестве после 2-3 эпох стабилизировался. Это говорит об отсутствии какого-либо выраженного процесса переобучения по мере продолжения процесса (по крайней мере в данной задаче).

После 10-ти эпох точность составила 98.3%.

При повторении обучения, в зависимости от случайных значений в начальной инициализации, итоговые значения могут немного расходиться примерно в диапазоне 98.2 – 98.4%. Но график обучения визуально, остается практически одинаковый.

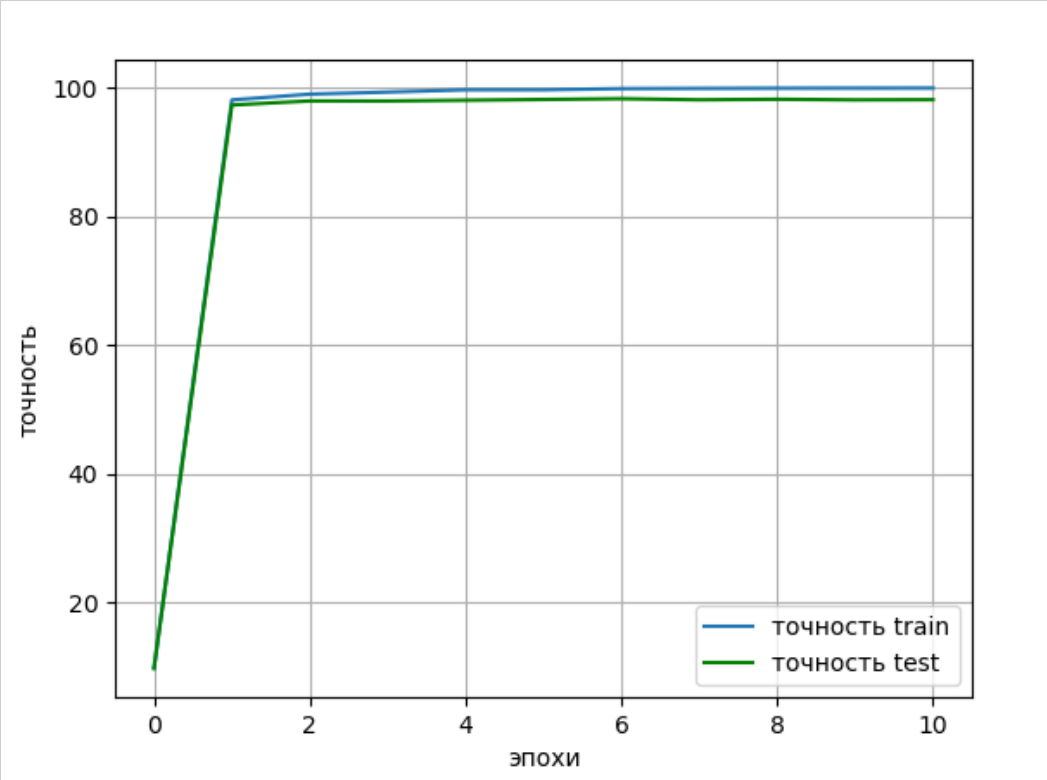

Рисунок 2. График обучения набору данных MNIST

Время обучения на эпоху составило около 1.5 сек.

Заключение. Предложенный метод обучения не является лишь вычислительным приёмом, а логическим продолжением вероятностной парадигмы, заложенной в первой работе цикла [2]. Отказ от градиентного спуска по коэффициентам λ обусловлен их статусом как множителей Лагранжа, а не свободных параметров. Переход к оптимизации по значениям функций в созвездии позволяет сохранить теоретическую целостность модели даже в глубокой композиции. Это делает полигармонический каскад архитектурой, в которой каждый слой и процедура обучения имеют вероятностное обоснование, выводимое из первых принципов.

Автор благодарит анонимных рецензентов за ценные отзывы о более ранних версиях этой работы.

Список литературы:

1. *Барцев С. И., Охонин В. А.* Адаптивные сети обработки информации. Красноярск : Ин-т физики СО АН СССР, 1986. Препринт N 59Б. - 20 с.
2. Бахвалов Ю. Н. 2025. Решение регрессионной задачи машинного обучения на основе теории случайных функций. PREPRINTS.RU. https://doi.org/10.24108/preprints-3113020

3. Бахвалов Ю. Н. 2025. Пакеты полигармонических сплайнов, их объединение, эффективные процедуры вычисления и дифференцирования. PREPRINTS.RU. https://doi.org/10.24108/preprints-3113111
4. Beatson, R. K., & Greengard, L. (1997). A short course on fast multipole methods. In Wavelets, Multilevel Methods and Elliptic PDEs (pp. 1–37). Oxford University Press.
5. 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
6. Buhmann, M. D. (2003). Radial Basis Functions: Theory and Implementations. Cambridge University Press.
7. Damianou, A., & Lawrence, N. (2013). Deep Gaussian Processes. In Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics (AISTATS), 207–215.
8. Dennis, J. E., & Schnabel, R. B. (1996). Numerical Methods for Unconstrained Optimization and Nonlinear Equations. SIAM.
9. Duchon, J. (1977). Splines minimizing rotation-invariant semi-norms in Sobolev spaces. In Constructive Theory of Functions of Several Variables (pp. 85–100). Springer.
10. Harder R.L. and Desmarais R.N.: Interpolation using surface splines. Journal of Aircraft, 1972, Issue 2, pp. 189−191
11. 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
12. Пугачев В.С., Теория случайных функций и её применение к задачам автоматического управления. Изд. 2-ое, перераб. и допол. - М.: Физматлит, 1960.
13. Rumelhart D.E., Hinton G.E., Williams R.J., Learning Internal Representations by Error Propagation. In: Parallel Distributed Processing, vol. 1, pp. 318 - 362. Cambridge, MA, MIT Press. 1986
14. Тихонов А. Н., Арсенин В. Я. Методы решения некорректных задач. - 3-е изд., испр. - М.: Наука, 1986. - 288 с.
15. Williams, C. K. I., & Seeger, M. (2001). Using the Nyström method to speed up kernel machines. In Advances in Neural Information Processing Systems 13 (NIPS), 682–688.
16. Wilson, A. G., Hu, Z., Salakhutdinov, R., & Xing, E. P. (2016). Deep Kernel Learning. In Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS), 370–378