ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
Факультет Санкт-Петербургская школа физико-математических и компьютерных наук

Винниченко Максим Юрьевич

ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ЯДЕРНЫХ МОДЕЛЕЙ ГЛУБОКОГО ОБУЧЕНИЯ

Выпускная квалификационная работа
по направлению подготовки 01.04.02 Прикладная математика и информатика
образовательная программа «Программирование и анализ данных»

Рецензент
руководитель подразделения
инструментов машинного обучения
Яндекс.технологии

_____
В.А. Ершов

Научный руководитель
кандидат физико-математических наук,
доцент

_____
И.Е. Куралёнок

Санкт-Петербург 2021

Оглавление

Аннотация

Недавние исследования в машинном обучении изучают связь между нейронными сетями и ядерными методами. Для бесконечно широких сетей можно вычислить ядро, пользуясь только архитектурой нейронной сети. С помощью полученного ядра задача машинного обучения сводится к линейной регрессии. Однако, процесс вычисления нейросетевых ядер аналитически плохо масштабируется: время обращения ядра растёт кубически от размера обучающей выборки. В этой работе предлагаются изучаются к решению этой проблемы.

Ключевые слова: нейронные сети, ядерные методы, линейная регрессия, экспонента матрицы.

Abstract

Our method suggests a new point of view on neural network optimization. We investigate the scalability of neural kernel methods to larger images, datasets. We approximate neural kernels with random features. The random features are sampled by concatenating outputs of thousands of small neural networks. We train a linear model over the random features, by sequential kernel optimization on batches of the data. This allows us to achieve state-of-the-art performance on a specific image classification task among kernel methods. Also, our approach has a huge potential for scaling to a large parallel compute.

Key words: neural networks, kernel methods, kernel regression, random features, scalability, gradient boosting, matrix exponential.

Введение

Ядерные методы

Ядерные модели основаны на обучении линейных моделей [14]. В линейной модели каждый признак вносит вклад в предсказание модели пропорционально его весу. Недостатком линейных моделей, является то, что они рассматривают каждый признак независимо от значений других признаков.

Ядерный трюк [1] является одним из вариантов решения этой проблемы. Он позволяет перевести признаки объектов в другое гильбертово пространство, которое назовём $\varphi(x)$, обучить там линейную модель. Для обучения такой модели достаточно уметь вычислять попарные скалярные произведения $k(x, y) = \dot{\langle}\varphi(x), \varphi(y)\rangle$. Функцию $k(x, y)$ называют ядром.

Недавние исследования в области нейронных сетей изучают формализм бесконечно широкой сети [8, 23, 16]. Оказывается, что обучение бесконечно широкой сети градиентным спуском эквивалентно обучению ядерной модели [23]. Это наблюдение позволило улучшить точность ядерных методов в задаче классификации изображений [16]. В частности, точность ядерных методов на наборе CIFAR-10 достигла 90%, что сопоставимо с точностью нейросетевых моделей. Однако, у нейросетевых ядерных методов есть существенных недостаток: они плохо масштабируются на большие размеры выборок.

Предмет исследования

В данной работе изучается проблема масштабируемости нейросетевых ядерных методов [25], [4] в задаче классификации изображений.

Мотивация

В случае нейросетевых ядерных методов, аналитическое вычисление ядра имеет вычислительную сложность $\mathcal{O}(n^2p^2)$, где n – размер обучающей выборки, p – число пикселей в изображении. Далее обращают матрицу ядра, сложность этой операции $\mathcal{O}(n^3)$. А также, сложность вычисления предсказания на одном объекте пропорциональна размеру обучающей выборки.

Эти ограничения не позволяет работать напрямую с большими размерами обучающих выборок.

Для решения этой проблемы существуют следующие 2 подхода. Вместо аналитического вычисления ядра можно использовать случайные признаки [18], [2]. Также можно обучаться на небольших случайных подмножествах обучающей выборки, называемых батчами, и потом комбинировать результаты обучения на этих батчах [**?**]. Назовём этот подход батчингом. Батчинг позволяет заменить обращение всей матрицы ядра на обращение маленьких подматриц. В данной работе рассматриваются оба подхода.

Цели и задачи

Целью данной работы является разработка масштабируемого ядерного метода для классификации изображений.

Для достижения данной цели были выделены следующие задачи:

- Обобщение метода Kernel Regression для случая необратимой матрицы ядра.
- Уменьшение вычислительной сложности обучения с помощью техники батчинг.
- Приближение аналитического ядра случайными признаками.
- Оценка качества метода на наборах данных CIFAR-10, Tiny ImageNet, ImageNet-R.

Структура работы

Работа устроена следующим образом. В главе 1 проведён краткий обзор литературы. В главе теория содержатся теоретические знания, необходимые для понимания работы. Также в главе теория доказывается, что линейную оптимизацию непрерывным аналогом градиентного спуска можно делать без обращения матрицы. В главе методы описывается метод машинного обучения, предложенный в данной работе. В главе данные описываются наборы

данных, использованные в использованные данной работе. В главе результаты представлены численные результаты сравнения метода с аналитическим вычислением ядра на наборе данных CIFAR-10, а также численные результаты сравнения с обучением нейронной сети ResNet18 на наборах данных Tiny ImageNet и ImageNet-R.

1. Обзор литературы

1.1. Нейронные сети

Впервые понятие искусственной нейронной сети было введено в 1946-ом году У. Маккалоком и У. Питтсом [13]. Они предложили математическую модель для описания нейронных сетей на основе логических схем.

Эти идеи несколько лет позже развил Ф. Розенблатт, который предложил в 1957 году модель [19] перцептрона для моделирования нейрона. Вход перцептрона и его веса задаются векторами $x = (x_1, x_2, \ldots, x_n), w = (w_1, w_2, \ldots, w_n)$ соответственно, где $x_i \in \mathbb{R}, w_i \in \mathbb{R}$. Выход претцептрона вычисляется как взвешенная сумма входов, к которой применяется функция активации σ: $perceptron(x) = \sigma(\sum x_i \cdot w_i)$. Далее будем называть перцептрон нейроном.

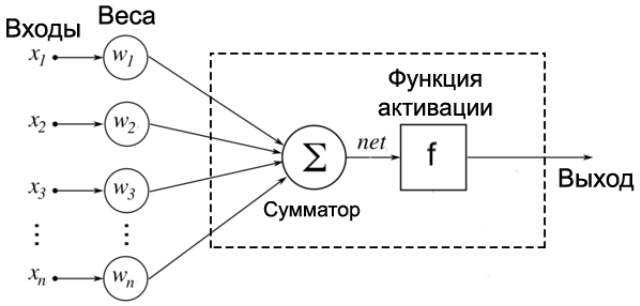

Рис. 1.1: Пример перцептрона [27]

В нейронной сети сигнал передаётся от предыдущего слоя нейронов к следующему. Математически такая модель записывается как:

$$x_i^l = \sigma\left(\sum_k w_{ik}^l x_k^{l-1}\right)$$

, где

$w_{i\star}^l$ – веса i-ого нейрона слоя l,

x_i^l – выход i-ого нейрона слоя l,

f – функция активации.

Вход нейронный сети из L слоёв, пронумерованных от 1 до L, можно задать как x^0, а выходом будет x^L.

Обозначим нейронную сеть как $f(x, W)$, где x – вход нейронной сети, $W = (w^1, w^2, \ldots, w^L)$ – веса нейронной сети.

1.1.1. Обучение нейронных сетей

Пусть известно $\mathcal{X}$ – множество объектов. $\mathcal{Y}$ – множество ответов. Существует целевая зависимость $y(x) : \mathcal{X} \to \mathcal{Y}$, значения которой известны на конечном множестве объектов $\{x_1, x_2, \ldots, x_n\} \subset \mathcal{X}$. Также задана дифференцируемая функция потерь $\mathcal{L}(\hat{y}, y)$, которая штрафует за ошибку предсказания $\hat{y}$ на объекте с правильным ответом y. Обозначим значения функции y на соответствующих объектах как $y_i = y(x_i)$.

Задачу обучения нейронной сети $f(x_i, W)$ можно сформулировать как задачу минимизации эмпирического риска:

$$\arg\min_W \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x_i, W), y_i)$$

.

Так как и функция потерь, и нейронная сеть дифференцируемы, можно вычислить градиент $\frac{\partial \mathcal{L}}{\partial W}$. Поэтому для поиска оптимальных можно воспользоваться методом градиентного спуска или другими схожими методами [26].

1.2. Свёрточные сети

Свёрточные сети были применены к задаче классификации изображений ещё в статье Gradient-based learning applied to document recognition [6] 1998 года. Однако, особый интерес ним начался в 2012 году после победы сети AlexNet [11] в соревновании ImageNet.

Свёрточная нейронная сеть обычно представляет собой чередование свёрточных слоёв (convolutional layers), применений функций активации, субдискретизиризующих слоёв (subsampling layers) и наличие полносвязных слоёв (fully connected layers) на выходе [6].

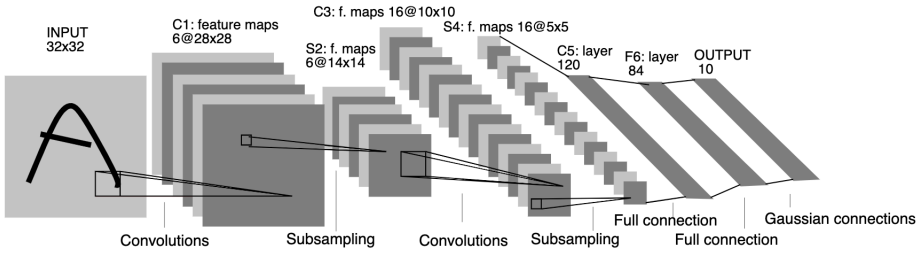

Рис. 1.2: Архитектура свёрточной сети LeNet5 [6]

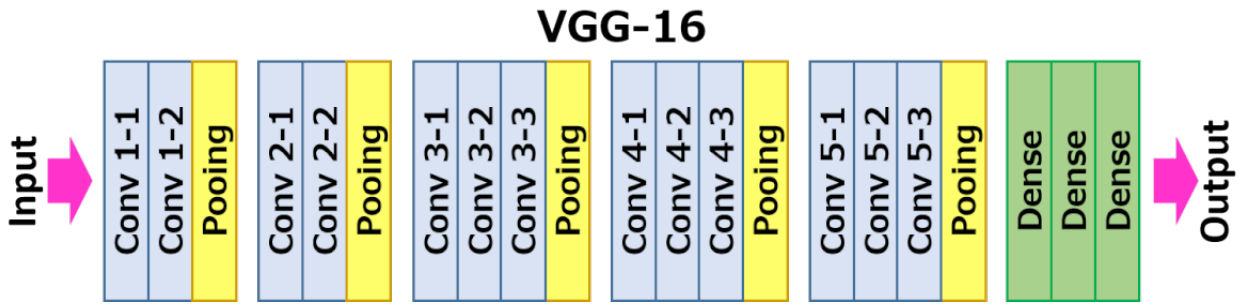

Рис. 1.3: Архитектура свёрточной сети: VGG16 [21] Свёрточные слои обозначены как Conv, полносвязные слои - Dense, субдискретизиризующие – Pooling.

1.2.1. Архитектура ResNet

При использовании архитектур свёрточных сетей, указанных выше в примерах, была замечена следующая проблема: с наращиванием глубины точность классификации сначала возрастает, а потом существенно деградирует.

Возникает вопрос а можем ли мы как-то обойти ограничение и нагромоздить больше слоёв без потери точности?

В статье Deep Residual Learning for Image Recognition [3] для решения этой проблемы предложили ввести добавить остаточный блок (residual block). Пускай слой учит зависимость $y = \mathcal{H}(x)$. Идея блока в том, чтобы разбить

$\mathcal{H}(x)$ на 2 составляющие: остаточную часть $\mathcal{F}(x)$ и тождественное преобразование $identity(x) = x$:

$$\mathcal{H}(x) = \mathcal{F}(x) + x$$

Утверждается, что остаточную часть $\mathcal{F}(x)$ легче выучить свёрточным слоем, чем учить оригинальное отображение $\mathcal{H}(x)$.

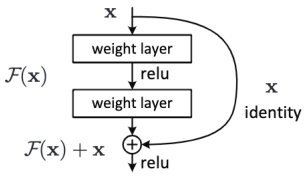

Рис. 1.4: Иллюстрация остаточного блока

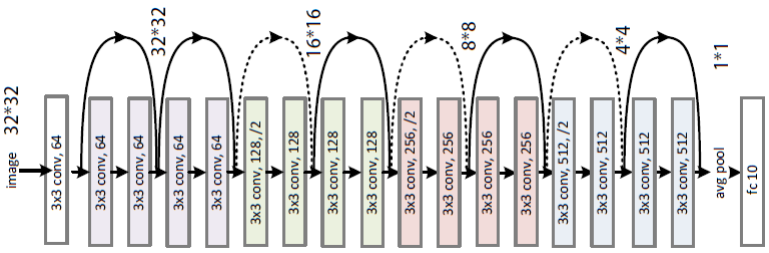

Рис. 1.5: Пример архитектуры свёрточной сети ResNet18 [3]

1.3. Ядерный трюк

Ядерный трюк был применён к задаче классификации изображений в статье Support Vetor Networks [1] в 1995 году. В этой статье предлагают метод обучения линейных моделей для задачи классификации изображений. Метод находит оптимальную разделяющую плоскость между 2мя классами.

Ядерный трюк состоит в следующем. Можно сначала перевести объекты в другое гильбертово пространство, которое назовём $\varphi(x)$, где потом обучить линейную модель. Для обучения такой модели достаточно уметь вычислять

попарные скалярные произведения $k(x, y) = \dot{\langle}\varphi(x), \varphi(y)\rangle$. Ядром называют функцию $k(x, y)$.

Например [9], пусть задан набор точек на плоскости $\{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$, которые хотим научиться разделять на 2 класса. Но, к сожалению, эти точки нельзя разделить линейной моделью – прямой. Чтобы обойти эту проблему, переведём точки в пространство $\varphi((a, b)) = (1, a, b, a^2 + b^2)$, положив их на параболу. Тогда ядро будет записано как функция $k(x, y) = 1 + x_a y_a + x_b y_b + ||x||^2||y||^2$, где координаты точек x и y это (x_a, x_b) и (y_a, y_b) соответственно. Утверждается, в пространстве φ уже точки хорошо разделимы линейной моделью.

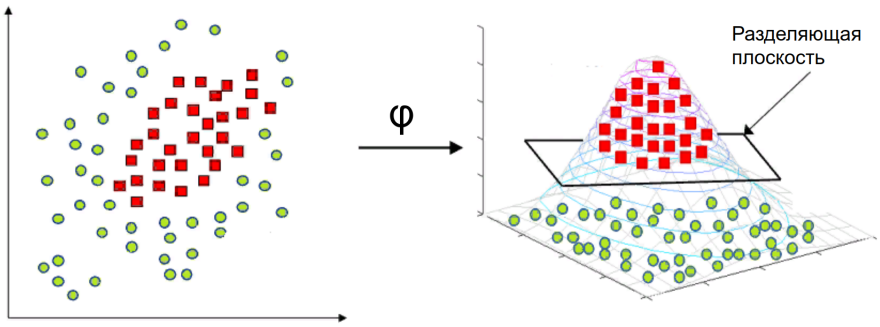

Рис. 1.6: Мотивационный пример для ядерного метода.

Ядерный трюк подробнее описан в разделе "Ядерный трюк" главы теория. Однако, метод оптимизации из статьи Support Vector Networks [1] не используется в данной работе, поэтому его подробное описание опущено.

Метод Kernel Ridge [20] применяет ядерный трюк в задаче регрессии. Для его применения водятся двойственные переменные, которые задают веса объектов. Вместо оптимизации весов признаков, оптимизируются веса объектов. Более формально метод Kernel Ridge описан в разделе "Двойственная задача" главы Теория.

1.4. Нейросетевые ядерные методы

В статье Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent [23] изучают связь между ядерными методами и нейронными сетями. Оказывается, что обучение бесконечно широкой сети эквивалентно обучению линейной модели, признаками которой является градиент этой нейронной сети. Аналогично, обучение нейросетевого гауссовского процесса, эквивалентно обучению только последнего слоя соответствующей широкой нейронной сети. Также в статье изучают динамику изменения параметров во время обучения градиентным спуском. Траектория изменения вектора весов линейной модели описывается линейным дифференциальным уравнением.

В статье "Neural Kernels without tangents" [16] тоже исследуется связь между нейронными сетями и ядерными методами. Однако, исследование происходит с более практической стороны: исследуется насколько верна аналогия между нейронными сетями и ядерными моделями верна на практике.

В статье проводится параллель между композиционными ядрами и свёрточными сетями. Для этого вводится следующая нотация. Набор изображений $x_1, x_2, \ldots, x_n$ можно описать как многомерный тензор $T[i, j, k, c]$, где i – номер изображения, (j,k) – координаты пикселя, c – номер цвета. Тогда можно начать с тензора ядра $K[i, j, k, l, m, n] = \langle T[i, j, k], T[l, m, n] \rangle$. Начальный тензор содержит корреляции всех пар пикселей. Рассмотрим случай бесконечно широкой сети. Следующие операции можно выразить аналитически, как преобразования тензора ядра: свёртка, активация, операции average pool [11]. Причём, операциями average pool можно уменьшить размер тензора ядра до $n \times 1 \times 1 \times n \times 1 \times 1$, что соответствует привычной матрице ядра.

Применение метода Kernel Ridge к полученной матрице ядра позволило им достичь точности классификации 89.8% на наборе данных CIFAR-10. Этот метод является state-of-the-art по точности классификации CIFAR-10 среди ядерных методов.

Проблемой метода является высокая вычислительная сложность. Метод имеет сложность $\mathcal{O}(n^3 + n^2p^2)$, где n – размер обучающей выборки, p – число пикселей в изображении.

2. Теория

2.1. Постановка задачи

Задача машинного обучения состоит в следующем. Известно $\mathcal{X}$ – множество объектов. $\mathcal{Y}$ – множество ответов. Существует целевая зависимость $y(x) : \mathcal{X} \to \mathcal{Y}$, значения которой известны на конечном множестве объектов $\{x_1, x_2, \ldots, x_n\} \subset \mathcal{X}$. Обозначим значения функции y на соответствующих объектах как $y_i = y(x_i)$. Задача машинного обучения состоит в построении оценки целевой зависимости $f(x)$, которая приближает $y(x)$ на всём множестве $\mathcal{X}$. Такой оценкой можно пользоваться для предсказания значения $y(x)$ для неизвестных объектов.

В случае задачи регрессии множество $\mathcal{Y}$ является либо множеством вещественных чисел $\mathbb{R}$, либо множеством вещественных векторов $\mathbb{R}^D$.

В случае задачи классификации множество $\mathcal{Y}$ имеет вид $\{1, 2, \ldots, D\}$, где значения $1, 2, \ldots, D$ – номера соответствующих классов. Часто оказывается, что проводить оптимизацию для построения $f(x)$ напрямую не удобно. Поэтому сводят задачу классификации к регрессии. Это можно делать следующим образом. Представим y_i как вектор вида $v(y_i) = (-1, -1, \ldots, 1, \ldots, -1)$, где значение на позиции y_i равно 1, на остальных позициях значения равны -1. Это позволяет решать задачу регрессии с векторными целевыми значениями. Для оценки номера класса объекта достаточно выбрать позицию с максимальным значением в оценке такого вектора.

2.2. Линейная регрессия

Линейная регрессия оценивает целевую зависимость линейной моделью. Формально, предсказание линейной модели с весам θ на объекте x имеет вид:

$$f(x, \theta) = \sum_{i=1}^{m} x_i \cdot \theta_i = \langle x, \theta \rangle$$

.

Для краткости обозначений представим объекты $x_1, x_2, \ldots, x_n$ и соответ-

ствующие ответы в матричном виде:

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \qquad Y = \begin{pmatrix} y(x_1) \\ y(x_2) \\ \vdots \\ y(x_n) \end{pmatrix}$$

В матричной форме: $Y = y(X), \quad f(X) = X\theta$

Теперь найдём веса, которые минимизируют функцию потерь $\frac{1}{2}||f(X) - y(X)||^2$ на известных объектах.

Воспользуемся формулой $\langle a, b\rangle' = \langle a', b\rangle + \langle a, b'\rangle$:

$\frac{1}{2}||X\theta - y||_2^2\ ' = \frac{1}{2}\langle X\theta - Y, X\theta - Y\rangle\ ' = \frac{1}{2}\left(\langle X, X\theta - Y\rangle + \langle X\theta - Y, X\rangle\right) = \frac{1}{2} \cdot 2\langle X, X\theta - Y\rangle = X^T(X\theta - Y)$

Минимум функции потерь достигается в нуле производной $X^T(X\theta - Y) = 0$. Поэтому оптимальное значение параметра:

$$\theta_{opt} = (X^T X)^{-1} X^T Y$$

2.3. Двойственная задача

Начнём со следующего наблюдения.

Пусть веса линейной модели θ можно разложить по базису $x_1, x_2, ..., x_n$:

$$\theta = \sum_{i=1}^{n} x_i \alpha_i = X^T \alpha$$

Тогда для вычисления f(z) для произвольного z нужно знать только скалярные произведения и коэффициенты α:

$$f(z) = \langle z, \sum_{i=1}^{n} x_i \alpha_i\rangle = \sum \alpha_i \langle z, x_i\rangle$$

.

Более того для поиска оптимальных коэффициентов достаточно знать только попарные скалярные произведения. Подставим разложение $\theta = X^T\alpha$ в условие минимума функции потерь $X^T(X\theta - Y) = 0$:

$$X^T(XX^T\alpha - Y) = 0$$

Решение:

$$\alpha = \left(XX^T\right)^{-1} Y$$

Для предсказания на произвольном наборе объектов Z подставим решение в линейную модель:

$$f(Z) = Z\theta = ZX^T\alpha = ZX^T\left(XX^T\right)^{-1} Y$$

Заметим, что матрицы ZX^T и XX^T состоят из попарных скалярных произведений, например:

$$XX^T = \begin{pmatrix} \langle x_1, x_1\rangle & \langle x_1, x_2\rangle & \dots & \langle x_1, x_n\rangle \\ \langle x_2, x_1\rangle & \langle x_2, x_2\rangle & & \langle x_2, x_n\rangle \\ \vdots & & \ddots & \vdots \\ \langle x_n, x_1\rangle & \langle x_n, x_2\rangle & \dots & \langle x_n, x_n\rangle \end{pmatrix}$$

Замечание: если матрица XX^T не обратима, то используют матрицу $XX^T + \beta E$, где параметр β соответствует L_2 регуляризации. Такой метод называют Kernel Ridge [20].

2.4. Ядерный трюк

Простой линейной модели часто оказываться недостаточно для точной оценки целевой зависимости. Одним из возможных решений этой проблемы является переход в другое пространство признаков. Причём, для применения ядерного трюка нам достаточно знать только попарные скалярные произведения в другом пространстве признаков.

Рассмотрим гильбертово пространство $\varphi(\mathcal{X}) = \{\varphi(x)|x \in \mathcal{X}\}$, где определено скалярное произведение $k(z, x) = \langle\varphi(z), \varphi(x)\rangle$.

Пусть $X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$

Обозначим: $\varphi(X) = \begin{pmatrix} \varphi(x_1) \\ \varphi(x_2) \\ \vdots \\ \varphi(x_n) \end{pmatrix}$

Также введём обозначения: $K = \varphi(X)\ \varphi(X)^T$, $K(Z, X) = \varphi(Z)\ \varphi(X)^T$

Замечание: матрицу K называют ядром.

Выпишем решение двойственной задачи линейной регрессии в пространстве $\varphi(\mathcal{X})$:

$f(Z) = \varphi(Z)\varphi(X)^T \left(\varphi(X)\varphi(X)^T\right)^{-1} Y = K(Z, X)K^{-1}Y$

$\alpha = K^{-1}Y$

Замечание: для обучения модели используем только значения функции $k(\cdot, \cdot)$ и Y.

2.5. Градиентный спуск с непрерывным временем

Задачу линейной оптимизации можно решать градиентном спуском с непрерывным временем. Обычный градиентный спуск является приближённым решением дифференциального уравнения методом Эйлера. Вместо приближённого решения дифференциального уравнения в данной работе рассматриваются точное решение.

Запишем уравнения на градиентный спуск в пространстве $\varphi(x)$:

$$f_t(X) = \varphi(X)\theta_t$$

$$\dot{\theta}_t = -\eta \frac{dL(f_t(X))}{d\theta}$$

,

где $L(F) = \frac{1}{2}||F - Y||_2$

Замечание: $f_t(X)$ – непрерывный аналог предсказания модели после t итераций градиентного спуска, θ_t – параметры линейной модели после t итераций градиентного спуска, η – шаг градиентного спуска.

Решим эти уравнения и найдём $f_t(X)$:

$$\dot{\theta}_t = -\eta \frac{dL(f_t(X))}{d\theta} = -\eta \left\langle \frac{\partial f_t(X)}{\partial \theta}, \frac{\partial L}{\partial f} \right\rangle = -\eta \ \varphi^T(X)(f_t(X) - Y)$$

$$\dot{f}_t(X) = \varphi(X)\dot{\theta}_t = -\eta \ \varphi(X)\varphi^T(X) \ (f_t(X) - Y)$$

$$\frac{d(f_t(X) - Y)}{dt} = -\eta K(f_t(X) - Y)$$

$$f_t(X) - Y = e^{-\eta K t}(f_0(X) - Y)$$

$$f_t(X) = (e^{-\eta K t} - E)(-Y) + e^{-\eta K t} f_0(X)$$

На практике удобно выбирать $\theta_0 = 0$ так что $f_0(X) = 0$.
В этом случае решение уравнения принимает вид:

$$f_t(X) = (e^{-\eta K t} - E)(-Y)$$

$$\theta_t = \varphi^T(X) K^{-1}(e^{-\eta K t} - E) \ (-Y)$$

Аналогично можно ввести двойственные переменные α_t:

$$\alpha_t = K^{-1}(e^{-\eta K t} - E) \ (-Y)$$

Предсказание модели на произвольных объектах Z имеет вид:

$$f_t(Z) = \varphi(Z)\theta_t = K(Z, X) \ K^{-1}(e^{-\eta K t} - E)(-Y)$$

2.6. Ненужность обращения матрицы

В этой секции сведём вычисление значения $K^{-1}(e^{-\eta K t} - E)$ к вычислению экспоненты матрицы.

Доопределим множитель $K^{-1}(e^{-\eta K t} - E)$ для случая необратимых K как

сумму ряда:

$$K^{-1}(e^{-\eta Kt} - E) = -\eta t \left(\frac{E}{1!} + \frac{-\eta Kt}{2!} + \frac{(-\eta Kt)^2}{3!} + \dots \right)$$

.

Можно доказать методом подстановки, что решения дифференциального уравнения остаются верными для необратимых K. Для краткости доказательства методом подстановки опущены.

Для вычисления этого множителя нам понадобится экспонента матрицы. Известно, что

$$\exp(M) = \frac{E}{0!} + \frac{M}{1!} + \frac{M^2}{2!} + \dots$$

Аналогично определим значение выражения $M^{-1}(e^M - E)$:

$$M^{-1}(e^M - E) = \frac{E}{1!} + \frac{M}{2!} + \frac{M^2}{3!} + \dots$$

Теорема 2.1.

$$\exp \begin{pmatrix} M & E \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} e^M & M^{-1}\left(e^M - E\right) \\ 0 & E \end{pmatrix}$$

Лемма 2.2. *Для $k \in \mathbb{N}$ верно:*

$$\begin{pmatrix} M & E \\ 0 & 0 \end{pmatrix}^k = \begin{pmatrix} M^k & M^{k-1} \\ 0 & 0 \end{pmatrix}$$

Лемма доказывается по индукции. Доказательство опущено.

Доказательство. Доказательство теоремы.

$$\exp \begin{pmatrix} M & E \\ 0 & 0 \end{pmatrix} = \frac{\begin{pmatrix} E & 0 \\ 0 & E \end{pmatrix}}{0!} + \frac{\begin{pmatrix} M & E \\ 0 & 0 \end{pmatrix}^1}{1!} + \frac{\begin{pmatrix} M & E \\ 0 & 0 \end{pmatrix}^2}{2!} + \frac{\begin{pmatrix} M & E \\ 0 & 0 \end{pmatrix}^3}{3!} + \dots =$$

$$= \frac{\begin{pmatrix} E & 0 \\ 0 & E \end{pmatrix}}{0!} + \frac{\begin{pmatrix} M & E \\ 0 & 0 \end{pmatrix}}{1!} + \frac{\begin{pmatrix} M^2 & M^1 \\ 0 & 0 \end{pmatrix}}{2!} + \frac{\begin{pmatrix} M^3 & M^2 \\ 0 & 0 \end{pmatrix}}{3!} + \dots =$$

$$= \begin{pmatrix} e^M & M^{-1}\left(e^M - E\right) \\ 0 & E \end{pmatrix}$$

□

Воспользовавшись теоремой, сведём вычисление $K^{-1}(e^{-\eta Kt} - E)$ к вычислению экспоненты матрицы:

$$-\eta t \exp \begin{pmatrix} -\eta tK & E \\ 0 & 0 \end{pmatrix} = -\eta t \begin{pmatrix} e^{-\eta tK} & (-\eta tK)^{-1}\left(e^{-\eta tK} - E\right) \\ 0 & E \end{pmatrix} =$$

$$= \begin{pmatrix} -\eta t\ e^{-\eta tK} & K^{-1}\left(e^{-\eta tK} - E\right) \\ 0 & E \end{pmatrix}$$

Получили множитель $K^{-1}(e^{-\eta Kt} - E)$ в правом верхнем углу матрицы. Известно, что экспоненту матрицы численно можно вычислять как предел: $\left(E + \frac{M}{2^k}\right)^{2^k} \xrightarrow[k\to\infty]{} \exp(M)$. Вместо обращения матрицы, достаточно численно оценить значение этого предела с помощью бинарного возведения матрицы в степень.

3. Методы

3.1. Оптимизация с помощью экспоненты матрицы

Научимся обучать линейную модель в пространстве $\varphi(\mathcal{X})$ с помощью градиентного спуска с непрерывным временем. Для начала рассмотрим поиск значений двойственных переменных. Их можно найти даже в случае бесконечномерных векторов в пространстве $\varphi(\mathcal{X})$, так как для их вычисления достаточно попарных скалярных произведений.

Из предыдущей главы известно, что значения двойственных переменных α_t вычисляются по формуле:

$$\alpha_t = K^{-1}(e^{-\eta K t} - E)\ (-Y)$$

Множитель $K^{-1}(e^{-\eta K t} - E)$ вычисляется через экспоненту матрицы:

$$\begin{pmatrix} \cdot & K^{-1}(e^{-\eta K t} - E) \\ \cdot & \cdot \end{pmatrix} = -\eta t \exp \begin{pmatrix} -\eta t K & E \\ 0 & 0 \end{pmatrix}$$

.

Для вычисления экспоненты небольших матриц используется функция scipy.expm. Для больших матриц используется реализации метода удвоения аргумента [22] на PyTorch.

Выбраны значения параметров $\eta = 10^5$, $t = 1$.

Зная значения двойственных переменных, можно вычислить значения весов линейной модели: $\theta_t = \varphi(X)^T \alpha_t$

3.2. Батчинг

Алгоритмы, описанные в этой секции, основаны на методе градиентного бустинга [5]. Суть градиентного бустинга состоит в следующем. Итеративно обучается ансамбль слабых регрессоров, предсказание ансамбля является взвешенной суммой предсказаний слабых регрессоров. На каждой итерации обучается один слабый регрессор и добавляется в ансамбль.

Опишем обучение слабого регрессора, используемого в данной работе. На каждой итерации выбираем m непересекающихся подмножеств $X_1 \sqcup X_2 \sqcup \ldots \sqcup$

$X_m \sqcup X_{leftover} = X$. В качестве слабого регрессора $f_i(Z)$ выступает среднее регрессоров, обученных на подмножествах $X_1, X_2, ..., X_m$. Итоговая линейная модель является взвешенной суммой слабых регрессоров.

Формализуем:

Обозначим $K_j = K(X_j, X_j)$

$F_i(Z)$ – предсказание ансамбля после i-й итерации бустинга, $F_0(Z) = 0$

Для простоты обозначений вычислим остатки на соответствующих объектах: $Y\_res_{ij} = y(X_j) - F_{i-1}(X_j)$.

Слабый регрессор:

$$f_i(Z) = \frac{1}{m} \sum_{j=1}^{m} K(Z, X_j)\ K_j^{-1}(e^{-\eta K_j} - E)(-Y\_res_{ij})$$

Шаг бустинга:

$$F_i = F_{i-1} + \beta \cdot f_i$$

β – параметр алгоритма.

Замечание: Взвешенную сумму слабых регрессоров можно хранить неявно в виде значений двойственных переменных или в виде значений весов линейной модели. Это возможно из-за линейности используемых моделей.

3.2.1. Алгоритм оценки значений двойственных переменных

Для этого алгоритма достаточно уметь считать только матрицу ядра.

Алгоритм находит **alpha** – оценку двойственных переменных.

```
def find_alpha(X, Y, learning_rate, beta, n_iter, batch_size):
    K_full = K(X, X)
    alpha = 0
    for _ in range(n_iter):
        Y_res = Y - K_full * alpha

        d_alpha = 0
        batch_indices = get_batches(|X|, batch_size, coverage=2/3)
        for idx in batch_indices:
            K = K(X[idx], X[idx])
```

```
            d_alpha[idx] = K^(-1)(exp(- learning_rate K) - E) * (-
                Y_res[idx])
        d_alpha /= len(batch_indices)

        alpha += beta * d_alpha
    return alpha
```

3.2.2. Алгоритм оценки весов модели

Для этого алгоритма нужно уметь представлять вектора X в пространстве $\varphi(X)$. Алгоритм находит **theta** – веса линейной модели.

```
def find_theta(X, Y, learning_rate, beta, n_iter, batch_size):
    theta = 0
    for _ in range(n_iter):
        d_theta = 0
        batch_indices = get_batches(|X|, batch_size, coverage=2/3)
        for idx in batch_indices:
            K = K(X[idx], X[idx])
            Y_res = Y[idx] - phi(X[idx]) * theta
            d_theta += phi(X[idx])^T * K^(-1)(exp(- learning_rate K)
                - E) * (- Y_res)
        d_theta /= len(batch_indices)

        theta += beta * d_theta
    return theta
```

3.3. Архитектура моделей

Нейросетевые модели, используемые в данной работе, основаны свёрточных нейронных [24] сетях из семейства Myrtle [17]. Примером такой сети является сеть Myrtle5:

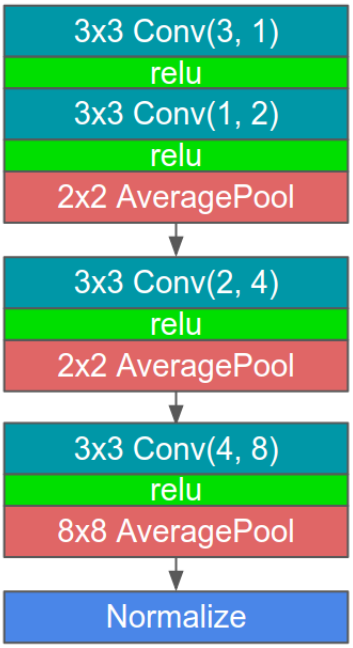

Рис. 3.1: Архитектура Myrtle5

В скобках указаны количества входящих и выходящих фильтров соответственно, которые выбраны в данной работе.

Нотация:

- 3x3 Conv(a, b) обозначает свёртку с фильтрами размера 3x3, a – число входящих фильтров, b - число выходящих фильтров.
- relu обозначает применение функции активации $\sqrt{2}\max(0, x)$.
- AveragePool обозначает операцию average pooling.
- Normalize обозначает операцию нормирования вектора $normalize(v) = \frac{v}{||v||_2}$.

Небольшое количество фильтров выбрано для того, чтобы сделать вычисление свёрток достаточно быстрым сэмплирования большого количества выходов нейронной сети. Данной работе количество сэмплируемых выходов на один объект достигает 800 тыс.

Также в данной работе используются архитектуры Myrtle7 и Myrtle10, которые представлены на рисунке ниже. Эти сети имеют большую глубину, чем Myrtle5, что делает их обучение более долгим, однако даёт больше качество классификации.

Рис. 3.2: Архитектуры Myrtle7 и Myrtle10 соответственно

3.4. Вычисление ядра с помощью случайных признаков

Определим скалярное произведение $k(z, x)$ как математическое ожидание: $k(z, x) = E_w\langle g(z, w), g(x, w)\rangle$. Тут $g(x, w)$ – выход нейронной сети с параметрами w на объекте x.

Ядро, заданное этим скалярным произведением, соответствует ядру нейросетевого гауссовского процесса для случая бесконечно широких сетей.

Это математическое ожидание можно оценим как:

$$\hat{k}(z, x) = \frac{1}{l}\sum_{i=1}^{l}\langle g(z, w_i), g(x, w_i)\rangle$$

.

В данной работе порядок параметра l составляет десятки и сотни тысяч.

Ядро оценённое таким образом сохраняет свою практическую применимость и для сетей небольшой ширины. В данной работе используем оценку математического ожидания $\hat{k}(z, k)$ для сетей Myrtle5, Myrtle7, Myrtle10.

Зафиксируем последовательность весов $W = (w_1, w_2, \ldots, w_l)$. Тогда можно составить большую нейросетевую модель G, выход которой будет конкатенацией выходов небольших моделей:

$G(x, W) = \frac{1}{\sqrt{l}}(g(x, w_1), g(x, w_2), \ldots, g(x, w_l))$

Более того модель $G(x, W)$ задаёт гильбертово пространство:

$$\hat{\varphi}(x) = G(x, W)$$

.

В этом гильбертовом пространстве можно обучить линейную модель. Из-за конечномерности этого пространства можно обучать не только двойственные переменные, но и веса линейной модели, что позволяет обучаться на больших обучающих выборках.

3.5. ZCA

Ядерные методы чувствительны к структуре пространства объектов. Предобработка ZCA [10] позволяет сделать пространство объектов более удобным для обучения, что повышает точность ядерного метода на наборе дан-

ных CIFAR-10 [16]. ZCA ортогонализует набор данных, но в отличие от PCA сохраняет фазу данных. Изображениями после преобразования ZCA остаются изображениями, на которых часто можно увидеть тот же объект.

Опишем ZCA подробнее:

Пусть даны объекты, представленные виде векторов $x_1, x_2, \ldots, x_n$.

Алгоритм ZCA состоит 4х шагов:

- Нормализация $x'_i = \frac{x_i - \overline{x_i}}{std(x_i)}$, где

  $\overline{v}$ – среднее вектора v,

  $std(v) = ||v - \overline{v}||$ – стандартное отклонение вектора v.

- Вычисление матрицы ковариаций:

$$C = X^T X$$

  , где X – матрица из нормализованных объектов:

$$X = \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \end{pmatrix}$$

- Применение сингулярного разложения:

  Симметричную матрицу C можно разложить в сингулярное разложение:

$$C = V D^2 V^T$$

  , где V – унитарная матрица, D – положительная диагональная матрица.

  Тогда итоговые признаки будут равны $\mathbf{X_{zca}} = \mathbf{X V} \frac{\sqrt{\mathbf{n-1}}}{\mathbf{D}+\epsilon} \mathbf{V^T}$

ZCA обладает следующим замечательным свойством. Оно полностью декоррелирует признаки при $\epsilon = 0$ и ортонормированной матрице V:

Посмотрим на сингулярное разложение X: $X = UDV^T$

$X^TX = VDU^TUDV^T = VD^2V^T$ (Это тоже сингулярное разложение матрицы ковариаций X^TX)
$X_{zca} = XV\frac{\sqrt{n-1}}{D}V^T = UDV^TV\frac{\sqrt{n-1}}{D}V^T = \sqrt{n-1}UV^T$
$X_{zca}^TX_{zca} = (\sqrt{n-1}UV^T)^T(\sqrt{n-1}UV^T) = (n-1)VU^TUV^T = (n-1)E$
$\mathbf{X_{zca}^TX_{zca} = (n-1)E}$

Сложность вычисления ZCA можно существенно уменьшить для случая больших изображений. Даже для цветных изображений размера 224×224, размер матрицы ковариаций равен 150528×150528, что занимает 90GB памяти.

Эту проблему можно обойти следующей оптимизацией:

- Оценим матрицы V и D явно как элементы сингулярного разложения X.

$$X = UDV^T$$

.

- Вместо точного разложения вычислим разложение приближённо, ограничившись k наибольшими элементами диагонали D:

$$X = U_kD_kV_k^T$$

.

- Для вычисления сингулярного разложения воспользуемся алгоритмом $randomized\_svd$ [7]. Сложность алгоритма $randomized\_svd$ $\mathcal{O}(nP\log(k)+(n+P)k^2)$.

В итоге сложность ZCA стала теперь понизилась с $O(np\min\{n,p\})$ до $\mathcal{O}(npk+(n+p)k^2)$, где k – константа в районе нескольких тысяч.

4. Данные

4.1. CIFAR-10

CIFAR-10 состоит 60 тыс. маленьких цветных изображений размера 32x32 пикселя. Каждое из изображений принадлежит одному из 10-ти классов: самолёт, автомобиль, птица, кошка, олень, собака, лягушка, лошадь, корабль, грузовик. Каждому классу принадлежит 6000 изображений.

Набор данных разделён 2 группы: тренировочная выборка из 50000 примеров и тестовая выборка из 10000 примеров.

4.2. Tiny ImageNet

Tiny ImageNet 110000 тыс. картинок цветных изображений размера 64x64 пикселя. Каждое изображение принадлежит одному из 200 классов. Набор данных сбалансированный: каждому классу соответствует 550 изображений. Примеры классов: пивная бутылка, стакан, божья коровка, золотая рыбка, дорожный знак...

Набор данных разделён 2 группы: тренировочная выборка из 100000 примеров и валидационная выборка из 10000 примеров.

4.3. Imagenet-R

Imagenet-R состоит картинок 30000 картинок из набора данных ImageNet. Картинки разделены на 200 классов. Отмечу, что набор данных не сбалансированный, также он не разделён на тренировочную и обучающую выборки.

Примеры классов: белая акула, молоток, буритто, футбольный мяч...

4.4. Предобработка данных

Все наборы данных были предобработаны преобразованием ZCA [15].

Матрица преобразования была вычислена на тренировочной выборке и применена ко всему набору данных.

Картинки из Imagenet-R были дополнительно масштабированы до размера 256x256 и обрезаны до размера 224x224. Так же Imagenet-R был разделён на тренировочную и тестовую выборки в отношении 9:1.

5. Результаты

5.1. Результаты на подмножествах CIFAR-10

Точность классификации на подмножествах размера 1280		
Ядро	Эмпирическое ядро	Аналитическое*
Myrtle5	64.2 ± .3	61.9 ± .7
Myrtle7	65.3 ± .8	-
Myrtle10	**66.6 ± .8**	64.4 ± .5

Таблица 5.1: Точность классификации (accuracy) на случайных подмножествах CIFAR-10, состоящих 1280 объектов. После знака ± приведено стандартное отклонение. Значения оценены по 5 случайным подмножествам. Колонка аналитическое* соответствует результатам статьи "Neural Kernels Without Tangents".

Для каждого объекта, участвовавшего в подмножестве, сэмплировался вектор случайных переменных размера 400 тыс. при вычисления эмпирического ядра. Значения для аналитического ядра взяты из статьи "Neural Kernels Without Tangents".

5.2. Результаты на всём CIFAR-10

Тут приведено сравнение различных ядер на всём CIFAR-10:

Точность классификации на CIFAR-10			
Ядро	Эмпирическое	Аналитическое	Аналитическое*
Myrtle5	-	-	85.8
Myrtle7	85.1	-	86.6
Myrtle10	86.1	-	87.5
Myrtle10 Gaussian Kernel	-	**88.0**	**88.2**

Таблица 5.2: Точность классификации (accuracy) на CIFAR-10. К аналитическому ядру, взятому из материалов статьи "Neural Kernels Without Tangents", применён алгоритм оценки двойственных переменных. Колонка аналитическое* соответствует результатам той же статьи.

Тут для каждого объекта, участвовавшего в подмножестве, сэмплировался вектор случайных переменных размера 800 тыс. при вычисления эмпирического ядра.

Различие в качестве классификации между алгоритмом оценки двойственных переменных переменных и методом Kernel Ridge Regression, который используется в статье "Neural Kernels Without Tangents", небольшое. Так же в той статье используется ряд трюков для повышения качества классификации, один из которых повышает качество классификации с помощью Kernel Ridge Regression на 0.3%. В данной работе этот трюк не используется.

Для единообразия используется алгоритм оценки двойственных переменных из предыдущего раздела. Алгоритм оценки весов модели показывает схожие результаты, однако для части аналитических ядер невозможно вычислить все веса явно из-за бесконечной размерности соответствующих гильбертовых пространств.

5.3. Результаты на Tiny-imagenet и Imagenet-R

Точность классификации на Tiny ImageNet		
	ядро Myrtle11	ResNet18
без аугментаций	**40.0**	31.8
с аугментацией	**43.4**	36.5
Точность классификации на Imagenet-R		
	ядро Myrtle15	ResNet18
без аугментаций	**24.4**	16.0
с аугментацией:	**22.7**	22.1

Таблица 5.3: Точности классификации на Tiny ImageNet и Imagenet-R

Используются следующие аугментации данных из библиотеки torchvision [12]:

- RandomCrop(56) и RadomHorizontalFlip – на Tiny-Imagenet
- RandomCrop(196) – на ImageNet-R

. Замечу, что аугментация RadnomCrop вырезает случайный кусок из заданного изображения, и поэтому качество классификации может упасть после применения этой аугментации.

Также замечу, что эти наборы в несколько раз крупнее CIFAR-10. Вычисление аналититических ядер на этих наборах данных заняло бы тысячи GPU часов. Поэтому сравнение с аналитическим ядром на этих наборах данных не проводилось.

Заключение

Разработан масштабируемый ядерный метод для классификации изображений. Показано, что метод масштабируется на наборы данных крупнее CIFAR-10: Tiny ImageNet, состоящий из 110 тыс. картинок размера 64x64, Imagent-R состоящий из 30000 картинок, которые были отмасштабированы до размера 224x224.

Для достижения этой цели использованы следующие техники: случайные признаки и батчинг, а также их комбинации. Случайные признаки позволяют уменьшить время вычисления ядра. Они даже показывают результаты, которые лучше аналитических ядер на небольших подмножествах CIFAR-10. Батчинг же позволяет достичь сходных результатов с Kernel Ridge Regression, но имеет лучшие перспективы для масштабирования на большие наборы данных и большие вычислительные мощности.

Оценка ядра случайными признаками превосходит аналитическое ядро на небольших подмножества CIFAR-10. Также метод превосходит свёрточную нейронную сеть ResNet18 при обучении на Tiny ImageNet, и он показывает сходные результаты с ResNet18 на наборе данных ImageNet-R.

Следующий ряд технических приёмов, представленный в работе, сделал возможным на практике обучение обучение только последнего слоя глубокой сети:

- использование конкатенации маленьких сетей вместо одной широкой сети
- решение дифференциального уравнения через экспоненту матрицы
- батчинг и бустинг

Представленный в работе подход даёт новый взгляд на проблему оптимизации с помощью нейронных сетей. Инициализация широкой нейронной сети уже сэмплируют признаки, советующие ядру, дающему высокую точность классификации. Однако, для обучения на этих случайных признаках нужны сотни тысяч итераций градиентного спуска. В данной работе это делается напрямую через решение дифференциального уравнения.

Список литературы

[1] Cortes Corinna, Vapnik Vladimir. Support-vector networks // Machine learning. — 1995. — Vol. 20, no. 3. — P. 273–297.

[2] Daniely Amit, Frostig Roy, Singer Yoram. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity // Advances In Neural Information Processing Systems. — 2016. — P. 2253–2261.

[3] Deep residual learning for image recognition / Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun // Proceedings of the IEEE conference on computer vision and pattern recognition. — 2016. — P. 770–778.

[4] Exact Gaussian processes on a million data points / Ke Wang, Geoff Pleiss, Jacob Gardner et al. // Advances in Neural Information Processing Systems. — 2019. — P. 14622–14632.

[5] Friedman Jerome H. Stochastic gradient boosting // Computational statistics & data analysis. — 2002. — Vol. 38, no. 4. — P. 367–378.

[6] Gradient-based learning applied to document recognition / Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner // Proceedings of the IEEE. — 1998. — Vol. 86, no. 11. — P. 2278–2324.

[7] Halko Nathan, Martinsson Per-Gunnar, Tropp Joel A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions // SIAM review. — 2011. — Vol. 53, no. 2. — P. 217–288.

[8] Jacot Arthur, Gabriel Franck, Hongler Clément. Neural tangent kernel: Convergence and generalization in neural networks // Advances in neural information processing systems. — 2018. — P. 8571–8580.

[9] Kernel method. — Access mode: `https://en.wikipedia.org/wiki/Kernel_method` (online; accessed: 2021-06-04).

[10] Krizhevsky Alex, Hinton Geoffrey et al. Learning multiple layers of features from tiny images. — 2009.

[11] Krizhevsky Alex, Sutskever Ilya, Hinton Geoffrey E. Imagenet classification with deep convolutional neural networks // Advances in neural information processing systems. — 2012. — Vol. 25. — P. 1097–1105.

[12] Marcel Sébastien, Rodriguez Yann. Torchvision the Machine-Vision Package of Torch // Proceedings of the 18th ACM International Conference on Multimedia. — MM '10. — New York, NY, USA : Association for Computing Machinery, 2010. — P. 1485–1488. — Access mode: https://doi.org/10.1145/1873951.1874254.

[13] McCulloch Warren S, Pitts Walter. A logical calculus of the ideas immanent in nervous activity // The bulletin of mathematical biophysics. — 1943. — Vol. 5, no. 4. — P. 115–133.

[14] Montgomery Douglas C, Peck Elizabeth A, Vining G Geoffrey. Introduction to linear regression analysis. — John Wiley & Sons, 2012. — Vol. 821.

[15] Nasonov Andrey, Chesnakov Konstantin, Krylov Andrey. CNN Based Retinal Image Upscaling Using Zero Component Analysis // The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences. — 2017. — Vol. 42. — P. 27.

[16] Neural kernels without tangents / Vaishaal Shankar, Alex Fang, Wenshuo Guo et al. // International Conference on Machine Learning / PMLR. — 2020. — P. 8614–8623.

[17] Page David. How to Train Your ResNet 4: Architecture // Myrtle.ai. — 2012. — online; accessed: https://myrtle.ai/learn/how-to-train-your-resnet-4-architecture/ (online; accessed: 15.06.2019).

[18] Rahimi Ali, Recht Benjamin. Random features for large-scale kernel machines // Advances in neural information processing systems. — 2008. — P. 1177–1184.

[19] Rosenblatt Frank. The perceptron: a probabilistic model for information storage and organization in the brain. // Psychological review. — 1958. — Vol. 65, no. 6. — P. 386.

[20] Saunders Craig, Gammerman Alexander, Vovk Volodya. Ridge regression learning algorithm in dual variables. — 1998.

[21] Simonyan Karen, Zisserman Andrew. Very deep convolutional networks for large-scale image recognition // arXiv preprint arXiv:1409.1556. — 2014.

[22] Walz Guido. Computing the matrix exponential and other matrix functions // Journal of computational and applied mathematics. — 1988. — Vol. 21, no. 1. — P. 119–123.

[23] Wide neural networks of any depth evolve as linear models under gradient descent / Jaehoon Lee, Lechao Xiao, Samuel Schoenholz et al. // Advances in neural information processing systems. — 2019. — P. 8570–8581.

[24] The history began from alexnet: A comprehensive survey on deep learning approaches / Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic et al. // arXiv preprint arXiv:1803.01164. — 2018.

[25] A kernel theory of modern data augmentation / Tri Dao, Albert Gu, Alexander J Ratner et al. // Proceedings of machine learning research. — 2019. — Vol. 97. — P. 1528.

[26] A theoretical framework for back-propagation / Yann LeCun, D Touresky, G Hinton, T Sejnowski // Proceedings of the 1988 connectionist models summer school. — Vol. 1. — 1988. — P. 21–28.

[27] Нейронные сети, перцептрон. — Access mode: `https://neerc.ifmo.ru/wiki/index.php?title=Нейронные_сети,_перцептро` (online; accessed: 2021-06-04).