МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**Институт информационных технологий, математики и механики**

**Кафедра алгебры, геометрии и дискретной математики**

Направление подготовки: «Фундаментальная информатика и информационные технологии»

Профиль подготовки: «Когнитивные системы»

**МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ**

**Тема:**

**«Анализ и модификация метода**

**CoGradient Descent»**

Выполнил:

студент группы 382006-3м

Репин В.И.

Научный руководитель:

доктор физ.-мат. наук

профессор кафедры АГиДМ

Золотых Н.Ю.

Нижний Новгород  
2022

Содержание

[Введение 3](#_Toc104825127)

[1. Обзор предметной области 4](#_Toc104825128)

[2. Цель работы 6](#_Toc104825129)

[3. Задача билинейной оптимизации 7](#_Toc104825130)

[4. Метод CoGradient Descent 8](#_Toc104825131)

[4.1. Описание и обоснование метода 8](#_Toc104825132)

[4.2. Модификации метода 12](#_Toc104825133)

[4.3. Практические применения и достигнутые результаты 14](#_Toc104825134)

[5. Реализация метода CoGD 18](#_Toc104825135)

[5.1. Алгоритм работы метода 18](#_Toc104825136)

[5.2. Описание реализованных функций 20](#_Toc104825137)

[6. Численные эксперименты 23](#_Toc104825138)

[6.1. Проверка работоспособности метода 23](#_Toc104825139)

[6.2. Тестирование модификаций 25](#_Toc104825140)

[7. О сходимости метода 32](#_Toc104825141)

[Заключение 33](#_Toc104825142)

[Список литературы 34](#_Toc104825143)

[Приложение A: Исходный код программы 36](#_Toc104825144)

Введение

Одним из часто применяемых алгоритмов в машинном обучении и компьютерном зрении является градиентный спуск и его различные модификации. Наиболее широко он применяется в алгоритме обратного распространения ошибки при обучении параметров нейронных сетей минимизацией целевой функции потерь.

Усовершенствования данного метода обычно заключались в более стабильной или быстрой сходимости [Кингма и др., 1][Дозат и др., 3], при этом практически всегда используются частные производные по переменным, что не учитывает возможных внутренних зависимостей между ними. Во многих прикладных задачах переменная, на которую наложено ограничение на разреженность, сходится быстрее, чем те, что без ограничений [Ву и др., 2].

Одной из таких является билинейная задача оптимизации, часто применяющаяся в алгоритмах компьютерного зрения и не только. Такая модель позволяет эффективно разделить смешанные в одном наблюдении переменные. Как примеры можно привести освещение и цвет объекта, его форму и тень или опознанный объект и его позицию на изображении.

В недавно предложенном методе CoGradient Descent [Чжан и др., 4] для решения таких задач учитывается особенность асинхронной сходимости переменных, оригинальный подход, корректирующий градиент связанных переменных при различной их сходимости. Полученные авторами тестовые результаты по некоторым параметрам превосходят современные методы решения на практике.

В то же время теоретическая сторона данного метода остаётся не исследованной, не проведён анализ сходимости метода CoGD и нет строгого доказательства. Эти отсутствующие элементы очень полезны в контексте надёжных и объяснимых алгоритмов машинного и глубокого обучения. В изложенной ниже работе метод детально проанализирован и проведены некоторые исследования. Также предложены модификации, затем проверенные экспериментально.

1. Обзор предметной области

Сама задача билинейной оптимизации в базовом виде была поставлена в [Майрал и др. 5], в качестве проблемы нахождения базисного словаря, с помощью которого могли бы быть представлены сигналы-наблюдения из заданного набора, при этом желательно было достичь разреженности коэффициентов представления. В этой работе задача решается итерационно: для каждого образца сигнала с помощью алгоритма LARS было найдено разложение в текущем словаре, а затем словарь итерационно обновлялся с этим разложением.

Со временем были найдены и другие практические применения билинейной модели, например удаление шума с изображения [Абдельхамед и др., 7] или удаление размытия [Янг и др. 6], хотя и с немного отличающейся интерпретацией, где ограничение на переменную коэффициентов разложения было в виде квадратичной формы со специальной матрицей кв.ф. Также выявилась возможность применять модель для обрезки нейросетей в работе [Ляо и др. 8].

Существующие решения проблемы были, в основном, направлены на разделение задачи на более простые и решения их алгоритмом Alternating Direction Method of Multipliers (ADMM), представленным в работе [Бойд и др., 9]. Подобные подходы можно наблюдать в работах [Хейде и др., 10], [Янг и др.. 11], обе из которых посвящены свёрточному разреженному кодированию изображений.

При этом в приведённых выше методах не учитывается взаимодействие между разделяемыми переменными, всегда происходит оптимизация их по отдельности. Данное эвристическое предположение, что оптимизируемые переменные друг от друга не зависят, приводит к более ранней сходимости по переменной с ограничением на разреженность и недостаточно оптимальному решению в целом. Это и было принято во внимание в работе [Чжан и др., 4], представившей метод CoGradient Descent.

Также можно заметить, что ранее в литературе билинейная модель не была использована непосредственно при обучении нейронных сетей, но при модификации модели слоя пакетной нормализации получается в точности билинейная модель.

Таким образом, подход рассматриваемого метода является новым, экспериментальным, а некоторые эмпирические допущения, используемые в его работе в совокупности с хорошими практическими результатами, делают теоретический анализ особенно интересным. Для понимания того, как производится данный анализ для других методов оптимизации, были рассмотрены уже упомянутые [Кингма и др., 1][Дозат и др., 3], рассказывающие об алгоритме ADAM, модификации градиентного спуска, где такие действия проводились.

1. Цель работы

Необходимо провести теоретический и практический анализ метода CoGradient Descent, а также сделать попытку найти строгое доказательство его сходимости, пусть и при некоторых ограничениях.

Для достижения данной цели нужно решить следующие задачи:

* Определить задачу билинейной оптимизации и её особенности
* Детально разобрать метод CoGradient Descent и его возможные практические применения
* Реализовать метод CoGradient Descent для базовой задачи оптимизации и предложить модификации
* Провести численные эксперименты для определения границ применимости
* Протестировать модификации метода
* Рассмотреть возможность доказательства сходимости метода

1. Задача билинейной оптимизации

Рассматривается оптимизация следующей целевой функции (**базовая версия задачи билинейной оптимизации**):

; (1)

где – наблюдение, которое может быть описано с помощью переменных и , что и представляет собой разделение наблюдения на два фактора. При этом на переменную накладывается ограничение на разреженность в виде нормы с параметром, чтобы избежать переобучения модели. На переменную может быть наложено более специфическое ограничение регуляризации, например норма каждого отдельного столбца не должна превышать единицы, что используется в задаче свёрточного разреженного кодирования.

В целом модель может быть по-разному подстроена под практические задачи, что будет продемонстрировано далее. Также стоит упомянуть обобщенную билинейную модель, введённую в работе [Йокойя и др., 12], где наблюдение представлено в виде суммы произведения пары матриц с добавлением шума, а при решении модель разбивается на две, схожие с базовой.

Одно из возможных решений данной проблемы, как уже было указано – алгоритм ADMM, разделяющий задачу на более простые подзадачи для упрощения. Также есть и стандартный подход – градиентный спуск и его модификации. В самом простом виде:

С различными коэффициентами скорости обучения Несмотря на все обыкновенные модификации градиентного спуска базовая идея остаётся неизменной – оптимизация одной переменной при фиксации значений другой, происходящая итеративно.

Рассмотрим и обоснуем алгоритм, основанный на предпосылке, что переменные всё же взаимодействуют между собой при формировании наблюдения.

1. Метод CoGradient Descent
   1. Описание и обоснование метода

В выражении частной производной по для билинейной задачи (без учёта регуляризации), можно увидеть, что градиент по будет затухать при приближении к нулю, что несомненно случится из-за наложенного ограничения на разреженность Таким образом, слишком раннее схождение может вызывать преждевременное схождение , эта асинхронность мешает качественно проводить обучение модели.

(2)

Таким образом, желательно синхронизировать сходимость переменных, исправлять , когда его быстрая сходимость начинает мешать поиску решения. Для этого сделаем предположение, что зависит от и тогда целевая функция оптимизации преобразуется в . Вычислим полученную частную производную по , пренебрегая слагаемыми регуляризации:

Второе слагаемое – взятие сложной производной при зависимости матричных переменных. Дальнейшие вычисления проводим поэлементно для :

Преобразуя данное выражение:

Обозначив за , подставим формулу 2 в выражение выше:

где .

Учтём вид произведения

И сделаем следующее предположение: зависимость между присутствует только для i-го столбца матрицы **.** Это позволяет значительно упростить вид третьего множителя:

Теперь, перемножив две матрицы выше, можно получить выражение для матрицы, след которой нужно найти:

Подводя промежуточный итог рассуждений, мы получили выражение для частной производной по от целевой функции с учётом зависимости :

Используем это для корректировки нового значения x в методом градиентного спуска. Вернёмся к значению от нового для j-й компоненты вектора, учитывая зависимость .

Учитываем, что из выражения для обновления методом градиентного спуска можно выразить :

Это можно подставить в предыдущее выражение для корректированного значения и избавиться от слагаемого:

Теперь, когда все вычисления проведены, мы можем перейти от поэлементного варианта к полноценной матричной записи корректированной переменной:

При этом символом обозначается поэлементное произведение векторов (произведение Адамара). Уже сейчас видно, что для корректировки значения на текущем шаге, нужно вычислить следующее обычным способом, а затем заменить его на полученную комбинацию. При этом мы не можем контролировать степень влияния через коэффициент , он необходим для контроля воздействия в основном методе (каким бы он ни был). Поэтому введём гиперпараметры для контроля алгоритма следующим образом:

– гомогенное полиномиальное ядро.

(3)

Введя гиперпараметры таким образом, мы можем усилять влияние полиномиально через степень , и ослаблять его линейно коэффициентом (константа 2 внесена в него). Основное преимущество гиперпараметра неочевидно, оно заключается в том, что мы можем существенно увеличить по модулю большие элементы вектора и уменьшить малые. Почему это важно? Дело в поэлементном произведении векторов

Скалярное произведение, эффект оказываемый на переменную это:

Первый множитель – некое подобие невязки. Нельзя считать это невязкой в полном смысле, по отношению к решению задачи – ведь там ещё есть регуляризация, которая решение меняет. В то же время длина этого вектора снижает воздействие тем сильнее, чем ближе к решению текущие Причём для любого компонента вектора это воздействие одинаково. Второй множитель прямо показывает, насколько сильно в данной точке, для данного проявляется асинхронность сходимости. Чем больше это значение – тем сильнее будет воздействие на данную компоненту. Соответственно, возводя эти воздействия в степень, мы добиваемся более сильного действия на по тем компонентам, где асинхронность больше. Этот эффект улучшает работу алгоритма, а слишком большие значения нормируются через сразу для всего вектора, усиление компонент при этом сохраняется. Направление определяется знаком косинуса, который при возведении в квадрат всего выражения (в оригинальном методе) исчезает.

Полностью обоснован способ корректировки переменной , теперь осталось разобраться в какой момент следует провести корректировку.

Вводится характеристика для определения того, сошлась ли переменная с пороговым значением , которое можно по-разному задавать для различных практических задач:

В целом выражение выше должно возвращать единицу, если переменная активна в итерационном процессе и её градиент не затухает.

Ф. 5 Итоговое правило алгоритма CoGD

Условие для корректировки легко объяснить словами: в случае асинхронности сходимости со стороны , то есть ограничение разреженности по перешло свой нижний порог , разрежен и в достаточной степени сошёлся, а наоборот, мы корректируем преимущественно в тех компонентах, которые асинхронность вызывают. Нормы при выполнении данного условия существенно различны.

Уже упоминалось, что базовая версии задачи может быть незначительно модифицирована, соответственно вычисление **,** основанное на нужно также менять. (пример – добавление константы перед нормой в задаче свёрточного разреженного кодирования).

Конечно, на практике нельзя узнать зависимость в явном виде, чтобы посчитать частную производную для корректировки. Это выражение приближается отношениями переменных между итерациями алгоритма (эпохами обучения):

Если близко к 0, то значение производной фиксируется единицей.

Метод можно применять с любой модификацией градиентного спуска. Даже с любым методом, позволяющим получать следующее решение и производить пересчёт от нового корректированного значения, ведь совершенно не важен способ получения значения, возврат всё равно происходит по правилу обновления градиентного спуска.

На этом метод можно считать полностью описанным и обоснованным.

* 1. Модификации метода

Первое, что было замечено при анализе метода – некоторое несоответствие теоретического вывода с финальным результатом. Вывод был произведён, основываясь на следующем положении:

Что, очевидно, неверно, поскольку рассматривается задача минимизации целевой функции. – параметр скорости обучения, не может быть отрицательным в исходном выводе, поскольку тогда вся идея метода не соответствует происходящему – мы не осуществляем возврат назад (backtracking – при описании идеи в оригинальной статье), а продвигаемся дальше, усиляя асинхронность. Из-за этой путаницы со знаком вышло так, что итоговая формула включает в себя , хотя не должна. Вывод метода в прошлом подразделе основан на предпосылке, что мы *замедляем* сходимость и запускаем градиентный спуск в обратную сторону. Это полностью согласуется с идеей, несколько раз описанной и даже представленной на рисунке в оригинальной работе. При таком рассмотрении пропадает из итоговой формулы для пересчёта (он же , к которому мы возвращаемся с учётом зависимости).

Для этой модификации есть совершенно иная, третья возможность, если ход в обратную сторону не подразумевался, а нужен просто пересчёт от в продолжение сходимости и в прямой ход, но с тем же рассмотрением зависимости. Тогда следующее выражение будет основой вывода метода:

Дальше по выводу получается, что превратится в а итоговая формула преобразуется в . Это тоже необходимо протестировать, хотя данная реализация совершенно противоречит исходной идее.

Вторая отмеченная проблема – при введении полиномиального ядра не сохраняется знак скалярного произведения, возводимого в чётные степени (чаще всего в квадрат – большие степени нестабильны – известная проблема данного ядерного преобразования). При этом тестовые результаты на практических задачах не ухудшались при , как будет показано в следующем подразделе. Это странное обстоятельство также необходимо рассмотреть и проверить альтернативную версию метода.

Далее можно отметить довольно замысловатый способ определения асинхронности. Здесь нужно вводить два пороговых гиперпараметра для отслеживания норм **,** сильно зависящих от текущей задачи.Или одну, если порог для брать как медианное значение выборки норм на каждой итерации. Способ понятен: пока норма превышает определённое пороговое значение, мы корректируем , если он опускается по норме ниже медианы. (или ещё одного порогового значения). Предлагается альтернативное условие суждения об асинхронности, с введением обозначения:

Что достигается при использовании такого условия? Исходное положение метода – борьба с асинхронностью, должна быть исследована в динамике изменения , но отслеживаются только конкретные значения на каждой итерации. Здесь как раз и происходит оценка в динамике. Первая часть условия позволяет избежать ситуации, когда уже практически не меняется, и отношение во второй части из-за этого будет слишком большим, это может происходить на практике, что было замечено при первых пробах, и без данного условия метод может выйти из уже почти достигнутого решения. Также здесь блокируется случай, когда вообще растёт по норме. Вторая часть условия интуитивна, выражает асинхронность сходимости переменных.

Все эти модификации должны быть исследованы и проверены на предмет работоспособности и улучшения работы метода.

* 1. Практические применения и достигнутые результаты

В этом небольшом разделе рассмотрим применения метода на практике и полученные результаты, свидетельствующие об актуальности вопроса.

*Свёрточное разреженное кодирование:*

изображение.

cловарь, конкатенация матриц**,** обучаемых ядер свёртки.

–набор признаков, разреженный вектор с коэффициентами для кодирования из избыточного словаря .

При этом на матрицы внутри наложено следующее ограничение:

Подробно о преимуществах данного подхода рассказано в работе [Янг и др., 16]

Это ограничение не позволяет использовать итеративные методы безусловной оптимизации, поэтому задача решается в основном методом ADMM, разделяющим её на подзадачи с более простыми ограничениями. Примеры этого можно найти в [Хейде и др., 10], [Парик и др. 15]. Но это никак не мешает между итерациями применить корректировку с помощью CoGD по уже заданному правилу. Результаты показали улучшение в задаче восстановления частей изображения (Inpainting) для двух различных датасетов, обычно использующихся для сравнений. На рис. 1 представлены результаты, измеренные по стандартным меркам качества изображений PSNR (Peak Signal-to-Noise Ratio) и SSIM (Structural SIMilarity). Определение и сравнение этих метрик было описано в работе [Ален и др., 17].

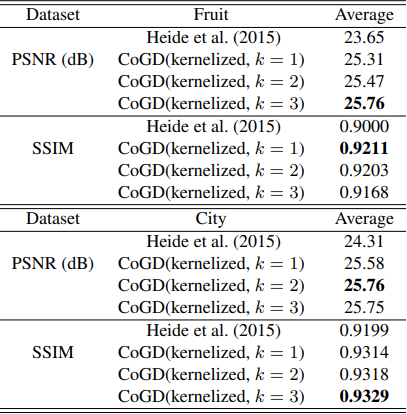


Рис. 1 Результаты применения CoGD - СРК. [4]

*Обрезка весов свёрточной нейронной сети:*

Это достаточно актуальная задача, решение которой позволяет сжать и упростить нейросеть для практического использования без понижения качества её работы.

Хотя изначально этот вопрос решался с довольно простыми условиями на норму свёрточных ядер [Ли и др., 18] или жадным алгоритмом [Луо и др., 19], но теперь применяется и билинейная модель с маской **,** на которую наложено ограничение на разреженность. [Хе и др., 20], [Хуанг и др., 21], [Линь и др., 22].

Модель представлена следующим образом:

- карта признаков на слое .

– веса свёртки на слое

соответствующий элемент маски (softmask – значения могут быть от 0 до 1).

- функция активации, ⊗ - операция свёртки

Задача билинейной оптимизации вводится далее:

Причём, если после обучения для соответствующей свёртки значение в маске около нуля – свёртка считается бесполезной и может быть обрезана.

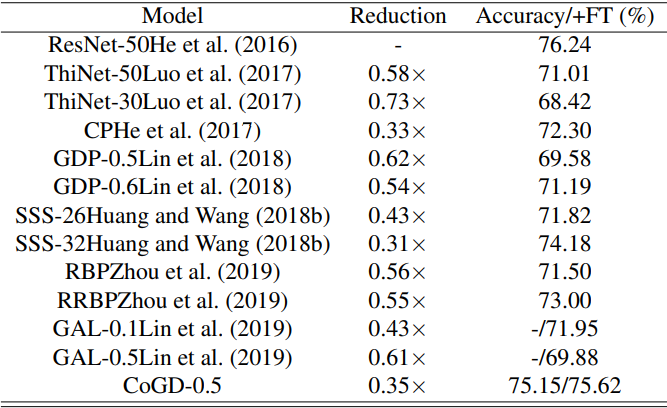


Рис. 2 Применение различных способов обрезки к ResNet на датасете ImageNet [4]

*Обучение нейронной сети:*

Это применение оказалось наиболее тяжёлым для понимания. Понятно, что применить CoGD можно только в тех задачах, где присутствует билинейная модель. При обычных обстоятельствах этого в обучении CNN нет.

Однако, для одного конкретного слоя, где применяется пакетная нормализация перед активацией, можно применить коррекцию CoGD, рассматривая параметр сжатия как **,** а веса на текущем слое как **:**

*-* среднее и дисперсия текущего пакета.

И тогда можно обновлять веса согласно следующей билинейной модели:

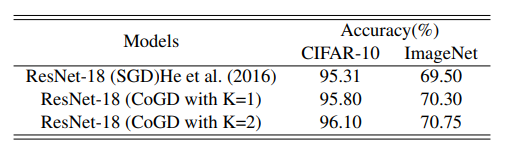


Рис. 3 Обучение нейросети с внедрением CoGD на слое пакетной нормализации. [4]

Можно заметить, что во всех случаях применение метода обеспечивало результаты лучше, чем были достигнуты ранее другими подходами. Это ясно прослеживается во всех результатах, полученных авторами публикации, [Чжан и др., 4] на которой основана данная работа. Тем более интересны обнаруженные неувязки, исправление которых может сделать этот подход ещё лучше.

Здесь заканчивается первая часть работы, которая описывает решаемую задачу, показывает новизну и результативность метода и предполагаемые модификации. Теперь можно перейти к практической части, где можно будет оценить его работу и справедливость предложенных исправлений.

1. Реализация метода CoGD
   1. Алгоритм работы метода

Ниже приведён оригинальный алгоритм CoGD на псевдокоде, отмечены места, где он модифицирован и сами модификации.

INPUT: ,,

OUTPUT:

IF < AND THEN (1)

FOR := 0 TO N

IF THEN

(2)

(3)

RETURN

*Примечания к алгоритму:*

Операция - произведение Адамара, поэлементное умножение векторов. Под операцией подразумевается деление каждого столбца числителя на соответствующий элемент знаменателя. Это можно назвать поэлементным делением, если в качестве знаменателя рассмотреть матрицу из конкатенированных M раз строк:

Параметр может быть нефиксированным и вычисляться как медианное значение выборки норм во внешнем алгоритме. Это может быть эффективно реализовано через отсортированный массив и существенно не замедляет алгоритм. Оптимальное значение для сильно зависит от конкретной задачи, а именно от её размерности.

Теперь рассмотрим все предлагаемые модификации по очереди:

1. Альтернативное условие применения:

IF AND THEN

Это требует введения по крайней мере одного нового параметра (но за счёт избавления от двух других). Главное преимущество данного условия – отслеживание изменения норм переменных в динамике. Поскольку целью являются именно нормы переменных, а не их приращение, не рассматривается выражение вида. Ранее не было сказано другое очевидное преимущество: этот параметр намного меньше привязан к конкретной задаче. Отношение приростов – величина безразмерная и относительная, тогда как из оригинального метода строго привязано к норме , которая может быть существенно разной для просто разных по размеру задач. Можно использовать тот же что и в контроле переменной dadx, но на практике может оказаться, что его нужно сделать меньше. Почему именно меньше? Первое условие с эпсилон должно быть менее строгим, потому что оно отсеивает всё применение метода в целом, тогда как второе применение эпсилона отсеивает применение по конкретной компоненте вектора, оно может быть более строгим, следовательно, эпсилон большим.

1. Сохранение знака скалярного произведения и его ограничение сверху:

У данного изменения нет определённого обоснования, но его отсутствие непонятно. Само по себе возведение в степень не является тождественным преобразованием, к тому же возведённые в квадрат элементы теряют свой знак, но при этом не ухудшается работа метода. Это загадочное обстоятельство нужно хотя бы проверить самостоятельно и, возможно, объяснить.

Касательно того же объекта – авторы оригинального метода позаботились о том, чтобы он не вызывал расходимость в конце работы исходного – за счёт ограничения на приближение к нулю и установление единицами соответствующих столбцов матрицы dadx. Опасность расхождения с другой стороны, в самом начале его работы, у них отсутствует, ведь есть верхний порог и метод не будет применяться каждую итерацию. Это изменение вызвано сугубо практическими наблюдениями с альтернативным условием, когда некоторые элементы вектора были слишком велики после возведения в степень за счёт большого значения а велико оно обычно в самом начале работы метода. Без контроля этого аспекта оказывается, что альтернативное условие может применяться постоянно, а норма резко вырасти.

Обе эти модификации можно уместить в одну:

IF THEN

Параметр me должен быть достаточно большим, как минимум на порядок большим, чем .

1. Исправление финальной формулы согласно теоретическому выводу:

Здесь всё достаточно прямолинейно: Из выражения для корректированного вовсе исчезает Данное изменение обосновано в первой части работы.

* 1. Описание реализованных функций

Первая серия алгоритмов была реализована исключительно для более глубокого понимания его особенностей. Сразу было ясно, что на настоящих прикладных задачах его будет крайне тяжело реализовать, потому что детали реализации и внедрения были совсем непонятны и описаны поверхностно. Хотелось хотя бы проверить, что всё сделано правильно и результаты совпадают с полученными авторами оригинального метода для базовой задачи. Соответственно, был реализован именно этот метод для самой по себе скалярной задачи, но с возможностью корректировки в матричном виде.

В качестве основных алгоритмов, предоставляющих для корректировки, выберем некоторые наиболее применяемые на практике: стохастический градиентный спуск (SGD), Momentum и ADAM.

Подробное описание работы данных алгоритмов выходит за рамки данной работы, можно только сказать, что для реализации использовались описания, приведённые в оригинальных работах: [Кингма и др., 1] для ADAM и [Румельхарт и др., 13] для Momentum.

Описание используемых методов и их параметров:

* *numgrad(f,x,h = 1e-9) –* численный градиент в точке x для функции f;
* *CoGD( -* метод CoGradient Descent;

*g\_der,* - предвычисленное ;

*cur, prev, - ,* полученные итерацией основного алгоритма;

*alpx, alpA, -* пороги для суждения об асинхронности сходимости;

*lr3, k)* - – гиперпараметры алгоритма.

* *conv\_cond(x, A, alpx, alpA) –* функция, определяющая асинхронность сходимости. Может быть модифицирована для динамических порогов.
* *SGD( -* Стохастический градиентный спуск;

*f,* - целевая функция;

*st,-* стартовые значения для переменных;

*lr1,lr2, -* соответственно;

*num\_it, -* ограничение по числу итераций*;*

*use\_cogd=False,* использовать ли CoGD;

*g\_der=None, alpx=None,alpA=None,beta=None,k = None) –* параметры CoGD.

* *Momentum( -* метод оптимизации Momentum;

*f, st, lr1,lr2, num\_it, -* параметры имеют аналогичный методу SGD смысл;

*mnt, -* коэффициент импульса (параметр Momentum)

*use\_cogd=False, -* использовать ли CoGD;

*g\_der=None, alpx=None,alpA=None,beta=None, k =None) –* параметры CoGD.

* *Adam( -* метод оптимизации ADAM;

*f, st, lr1,lr2, num\_it, -* параметры имеют аналогичный методу SGD смысл;

*beta1,beta2, -* параметры Adam

*use\_cogd=False, -* использовать ли CoGD;

*g\_der=None, alpx=None,alpA=None,beta=None, k =None) -* параметры CoGD.

Первая реализация CoGD и соответствующих алгоритмов была предназначена для функций скалярных переменных, чтобы яснее увидеть работу метода на двумерной плоскости.

Но также есть и полностью матричная реализация, подходящая и для прикладных задач (если бы было ясно, как встроить её в окружающую модель). Именно на этой реализации будут проведены все тесты и сравнения, она же будет изучаться в третьем разделе. Для данной реализации внешним методом служит только Adam.

* *ConvCond\_bm(A, x, alpA, alpx) –* функция, определяющая асинхронность сходимости для матричных переменных.
* *R(m, p) –* вычисление нормы вектора m порядка p. Для матриц используется норма Фробениуса.
* *med\_sorted(m) –* медиана отсортированного списка. Служебная функция.
* *CoGD\_bm( -* метод CoGradient Descent для билинейной задачи.

*A\_prev, A\_cur, x\_prev, x\_cur, lr2, alpA, alpx, lr3, k, -* стандартные параметры

*dradrx, -* новый параметр для альтернативного условия применения

*newcond, poschange, keepsign, -* флаги для применения модификаций

*maxthresh = 2000, minthresh = 1e-4) –* пороговые значения для контроля c и

* *Adam\_bm( -* внешний метод, значения которого корректируются

*b, -* вектор-наблюдение размерности Mx1

*x\_len, -* размерность вектора Nx1.

*lamb,* - параметр при ограничении на разреженность **.** Три этих параметра полностью задают билинейную задачу.

*lr1, lr2, A0,x0, beta1, beta2, num\_it, -* cтандартные параметры Adam

*use\_cogd=False, newcond=False, poschange=False, keepsign =False, alpA=None, dradrx=None, lr3=None, k =None) –* параметры CoGD

1. Численные эксперименты
   1. Проверка работоспособности метода

Первый набор тестов относится к скалярной версии методов и проверяется поэтому на функции двух аргументов.

Для теста используем функцию, которую можно привести к матричному виду формулы 1, а именно функцию Била [Доккьюн и др., 14]. Выглядит она сама по себе следующим образом:

Добавим ограничение на разреженность и ограничение на , затем представим функцию в нужном виде:

Где ,

Мы можем решать задачу в переменных основным методом, находя градиент по ним, а корректировать отдельно с помощью CoGD в переменных **,** поскольку менять нам нужно будет только . При этом получится и испытать метод на неодномерной задаче и отрисовать результат.

Решение этой задачи найдено в пакете Wolfram Mathematica и составляет:

На всех изображениях ниже оно отмечено, чтобы можно было оценить близость полученного решения. Тестовые параметры можно найти в приложении А, где производился запуск методов. Начальная точка [1.5,1.8].

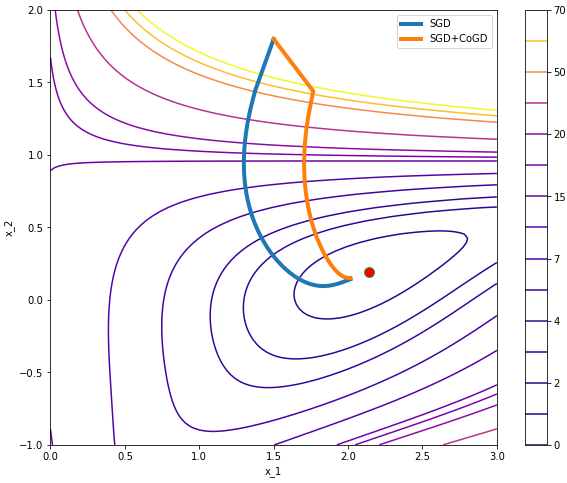


Рис. 4 SGD и CoGD на тестовой функции

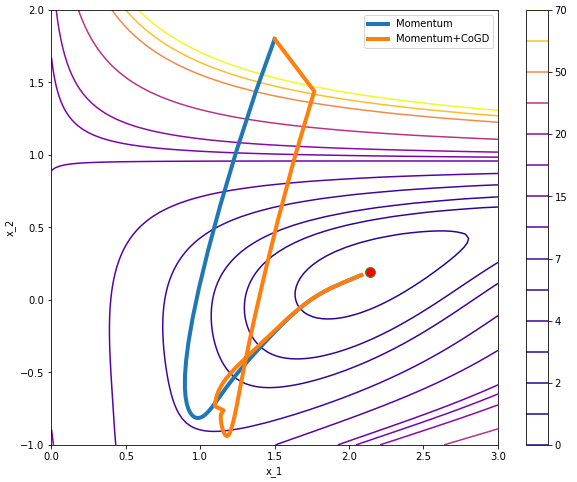


Рис. 5 Momentum и CoGD на тестовой функции

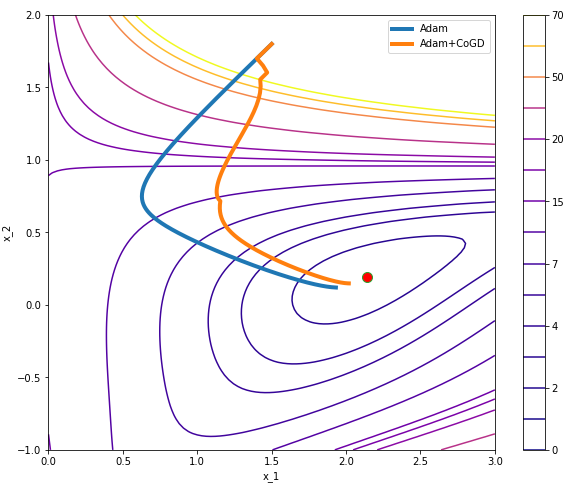


Рис. 6 Adam и CoGD на тестовой функции

Отлично видно, что в каждом случае метод сошёлся и путь оптимизации в среднем короче с внедрением CoGradient Descent, чем без него. Также эти результаты практически полностью совпадают с ожидаемыми для данной функции, параметров и стартовой точки. Для дальнейших экспериментов был выбран метод Adam.

* 1. Тестирование модификаций

Сначала определим, как именно будет сравниваться работа методов. Были рассмотрены различные варианты, в основном была изучена работа [Бейранванд и др., 23]. Есть график сходимости: на оси абсцисс – итерации метода, а на оси ординат – достигнутое значение целевой функции. Но это показывает результат лишь для одного запуска и для одной стартовой точки. Поэтому применяем подход, отображённый на рис. 7. Идея заключается в том, чтобы найти минимально достигнутое значение для того метода, с которым ведётся сравнение, в нашем случае – Adam. Далее мы определяем, на какой итерации каждый из методов достиг эпсилон окрестности данной точки. Разницу между этими значениями уже можно сравнить. В случае если метод не достигает этого диапазона за предоставленные итерации хотя бы в одном запуске – считаем, что он не сработал. Это полезно в том случае, если сравнить по итоговому значению целевой функции не удаётся – они слишком близки.

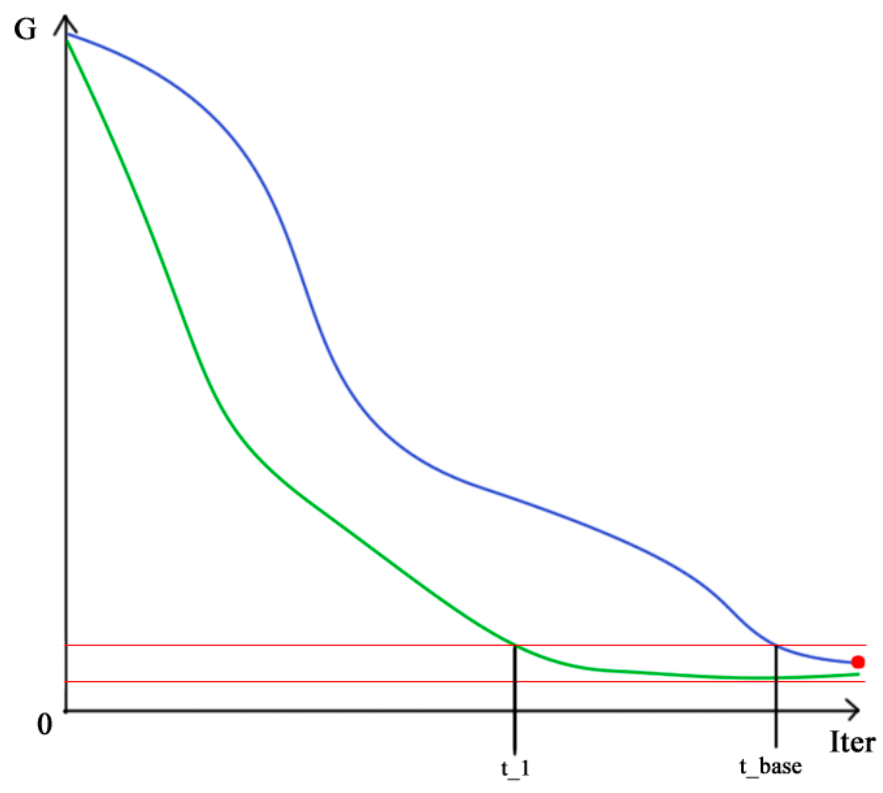


Рис. 7 Сравнение сходимости методов

В качестве тестовых задач теперь используем три разных вектора разных размеров, сгенерированных случайно. Задача полностью описывается двумя параметрами: вектор и размерность вектора . Эти размерности составляют размерность матрицы

Задачи размера 3x8, 8x25, 20x100. Каждый запуск содержит три метода: базовый Adam, Adam с внедрением оригинального CoGD, и Adam с модифицированным тем или иным способом CoGD. В каждом отдельном запуске старт находится в точке **,** их инициализация выполнена равномерно распределёнными числами в диапазоне [0,1). Все параметры, которые можно сохранять между методами остаются неизменными и оптимальными для исходного Adam. Если какой-то параметр изменяется, то будет приведена причина и изменённые значения.

*Исправление финальной формулы:*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 3x8 (300it) | | 8x25 (800it) | | 20x100 (2000it) | |
|  | t\_b-t | G | t\_b-t | G | t\_b-t | G |
| Adam | - | 5.635764 | - | 9.582050 | - | 16.299148 |
| Adam+CoGD | 104 | 5.643602 | 52 | 9.605140 | - | 28.529716 |
| Adam+v1CoGD | 98 | 5.662173 | 149 | 9.640232 | - | 29.659038 |
| Adam+v2CoGD | - | 6.882171 | - | 11.696880 | - | 16.180941 |

Таблица 1 Эксперименты с финальной формулой на различных размерах задачи

В таблице 1 отражены результаты экспериментов.

На рис. 8-10 изображены соответствующие запуски для каждого размера задач. В большинстве случаев график нарисован без первых итераций, значение целевой функции там слишком велико.

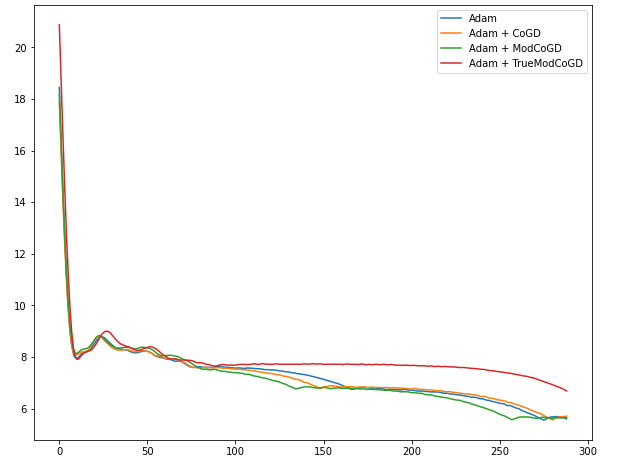


Рис. 8 Исправление формулы. 3x8

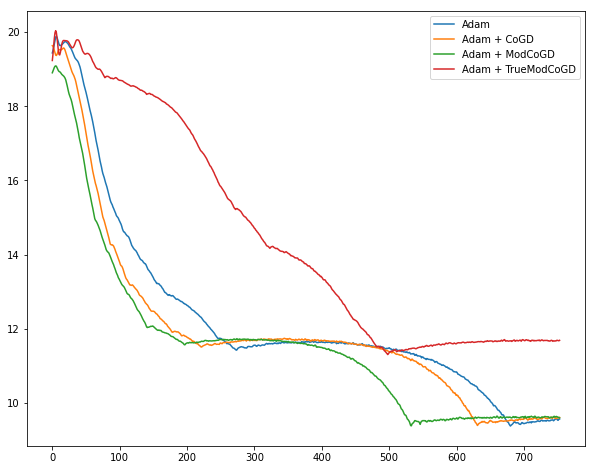


Рис. 9 Исправление формулы. 8x25

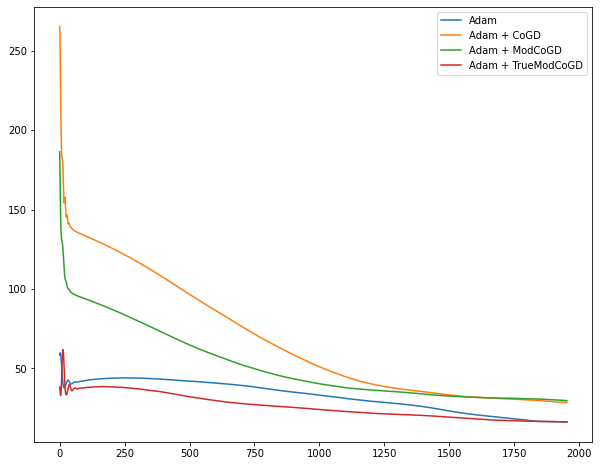


Рис. 10 Исправление формулы. 20x100

Итак, результаты очень запутанные. И оригинальный метод (как он описан и представлен в работе) и модифицированный по исходному замыслу работают корректно только для задач малого размера. То есть замысел замедлить сходимость с учётом зависимости для практических задач не сработает, их размер скорее похож на размер третьей тестовой задачи. Получается, что для лучшего результата всё же нужно выводить метод с предпосылкой ускорения сходимости с зависимостью, а верная формула в таком случае будет:.

Дальнейшие эксперименты будем проводить только на крупной задаче размера 50x100 с данной формулой обновления, внедряя другие модификации и проверяя их работоспособность.

*Cохранение знака:*

|  |  |
| --- | --- |
|  | G |
| Adam | 48.07704957 |
| Adam+CoGD | 34.22517829 |
| Adam+ModCoGD | 42.77637423 |

Таблица 2 Эксперименты по сохранению знака

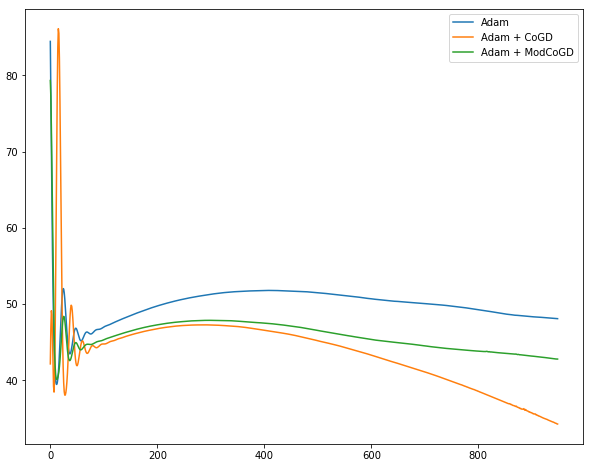


Рис. 11 Сравнение работы метода с модификацией сохранения знака

До этого во всех экспериментах было использовано k = 2, в том числе и здесь. Сохранение знака ухудшило результат, но это не единственная проверка, которую можно провести. Можно использовать степень k = 1, тогда как в оригинальном методе это значение по умолчанию сохраняет знак, то мы сделаем наоборот и уберём его. Согласно гипотезе, это должно улучшить работу метода.

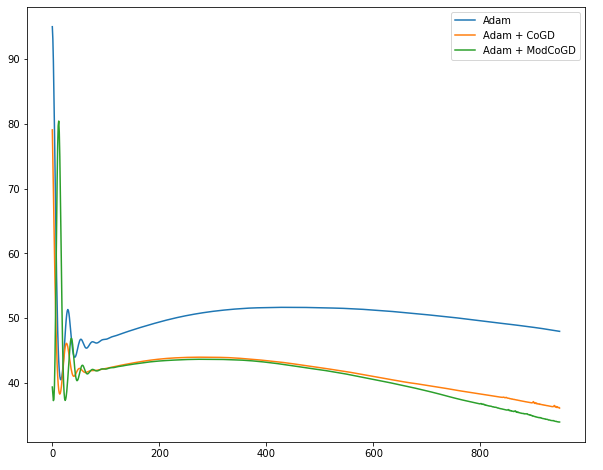


Рис. 12 Отбрасывание знака при k = 1

Это означает, что метод действительно работает лучше, если знак не сохранять. У данного факта может быть следующее объяснение: поскольку на наложено ограничение разреженности, то компоненты в среднем стремятся к нулю. При уменьшении компоненты мы можем рассмотреть применение метода: . В случае если   
 , мы получаем, что применение хода метода приближает к нулю только при положительном , знак которого полностью определяется скалярным произведением, рассмотренным в первой части работы.

*Альтернативное условие применения:*

|  |  |
| --- | --- |
|  | G |
| Adam | 19.141705 |
| Adam+CoGD | 18.961615 |
| Adam+ModCoGD | 15.382397 |

Таблица 3 Эксперименты с альтернативным условием (изменённый LR)

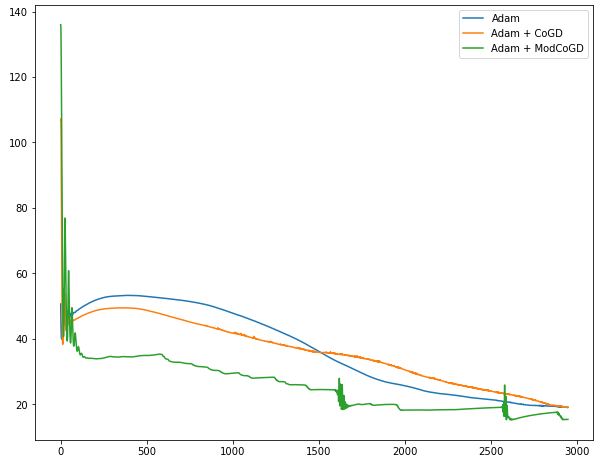


Рис. 13 Сравнение работы метода с альтернативным условием (изменённый LR)

Это один из тех случаев, когда пришлось уменьшить в два раза параметр , поскольку альтернативное условие применяется чаще, чем оригинальное. Без изменения наблюдается следующая картина:

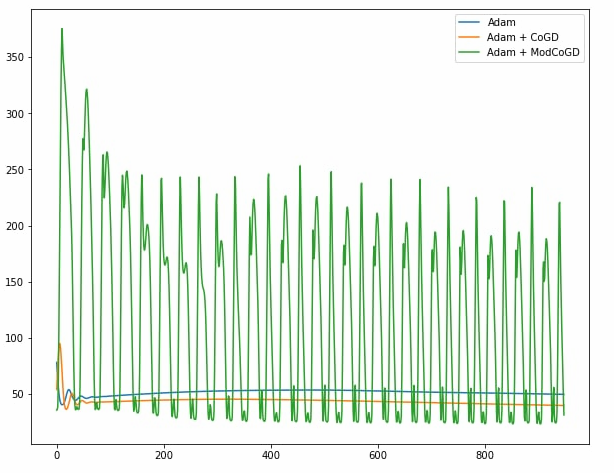


Рис. 14 Сравнение работы метода с альтернативным условием (одинаковый LR)

Итак, изменённое условие в целом работает лучше, на рисунке 13 чётко видны моменты применения метода, когда сходимость по замедляется и мы «подталкиваем» его в сторону нуля преимущественно по тем компонентам, где это происходит. Динамическое отслеживание состояния норм позволяет применять метод чаще и более локально.

1. О сходимости метода

В выводе и применении метода есть множество допущений, каждое из которых может помешать доказательству. Перечислим их:

1. При взятии обеих частных производных для коррекции игнорируются слагаемые регуляризации.
2. Предпосылка, что только там, где они взаимодействуют при перемножении в билинейной задаче. То есть **.**
3. Точная производная заменяется отношением приращений между итерациями (эпохами).

Допущения 1 и 2 позволяют нам представить результат в упрощенном виде и в целом нет никаких преград, чтобы избавиться от них и провести полное вычисление (кроме существенного замедления работы).

От допущения 3 избавиться не получится, мы не имеем и не можем найти зависимость в таком виде, который позволял бы взять производную.

В конце концов сам метод применяется нерегулярно, на некоторых отдельных итерациях, а выбор производится по специальному условию с гиперпараметрами. Это соображение наводит на мысль, что асимптотика и характер сходимости исходного метода не могут измениться. Ход внешнего метода нарушается выборочно и при определённом выборе параметров даже довольно редко. Применение его слишком часто с любым условием даёт картину, похожую на рис. 14.

Очень легко оценить, максимальное расстояние, на которое метод может сдвинуть с запланированной траектории на шаге t. Учитываем, что есть финальное ограничение сверху на уже возведённые в степень элементы **:** me – константа, для ограничения.

Итак, главный вопрос, ответ на который показывает, сойдётся ли комбинированный метод: выдержит ли основной алгоритм сдвиг на данное расстояние, осуществляемый регулярно? Именно поэтому от частоты применения метода напрямую зависит его сходимость.

Заключение

В ходе работы подробно изучен метод CoGradient Descent. Были предложены существенные модификации, некоторые из которых смогли объяснить непонятные аспекты его работы, а некоторые – улучшить результат на больших задачах. Объяснены странности и нестыковки, возникающие при изучении оригинальной статьи. Проведена серия экспериментов, подтвердивших поставленные гипотезы о том, как функционирует метод.

Определено, что замедление сходимости путём возврата к предыдущим итерациям работает только на задачах малого размера. В то же время на реальных задачах необходимо сдвигать компоненты в сторону нуля. Эксперименты для этой модификации положили конец сомнениям относительно способа работы метода.

Совершенно точно, отслеживание асинхронности в динамике даёт лучшие результаты, чем измерение норм на отдельных итерациях. Параметр для этого условия проще выбрать, а само условие более интуитивно описывает положение, выдвинутое авторами исходного метода.

В целом цели работы были достигнуты, а также можно определить дальнейшие продвижения в этой теме. Определение оптимальных параметров по умолчанию или их динамическое изменение, а также развитие теоретической части, где можно будет доказать сходимость. Желательно проверить введённые модификации на реальной задаче и сравнить результат.

Список литературы

1. D. P. Kingma, J. Ba Adam: A method for stochastic optimization // Computer Science. – 2014
2. J. Wu, L. Cong, Y. Wang, Q. Hu, C. Jian. Quantized convolutional neural networks for mobile devices // In CVPR. – 2016, DOI: 10.48550/arXiv.1512.06473
3. T. Dozat. Incorporating nesterov momentum into adam // In International Conference on Learning Representations, pages 1–8. – 2016
4. R. Wang, B. Zhang, L. Zhuo, Q. Ye, D. Doermann Cogradient Descent for Dependable Learning // arXiv preprint. – 2021, DOI: 10.48550/arXiv.2106.10617
5. J. Mairal, F. Bach, J. Ponce, G. Sapiro Online learning for matrix factorization and sparse coding // Journal of Machine Learning Research. - 2010. - №11(Jan). - pages 19–60
6. S. I. Young, A. T. Naman, B. Girod, D. Taubman; Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pages 5592-5601, 2019
7. A. Abdelhamed, M. A. Brubaker, M. S. Brown. Noise flow: Noise modeling with conditional normalizing flows. In ICCV, pages 3165–3173, 2019
8. Qiyu Liao, Dadong Wang, Hamish Holewa, Min Xu; Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019
9. S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers // Foundations and Trends in Machine Learning. - 2011 . - №3, Iss. 1. - pages. 1–122.
10. F. Heide, W. Heidrich, G. Wetzstein. Fast and flexible convolutional sparse coding. In CVPR, pages 5135–5143, 2015
11. L. Yang, C. Li, J. Han, C. Chen, Q. Ye, B. Zhang, X. Cao, W. Liu Image Reconstruction via Manifold Constrained Convolutional Sparse Coding for Image Sets // IEEE Journal of Selected Topics in Signal Processing. - 2017. - №11. - pages. 1072-1081
12. N. Yokoya, J. Chanussot, A. Iwasaki Generalized bilinear model based nonlinear unmixing using semi-nonnegative matrix factorization, 2012 IEEE International Geoscience and Remote Sensing Symposium, 2012, pages 1365-1368
13. D. Rumelhart, G. E. Hinton, R. J. Williams Learning representations by back-propagating errors // Nature. - 1986. - №323. - pages 533-536.
14. Y. Dokkyun, A. Jaehyun, J. Sangmin An Effective Optimization Method for Machine Learning Based on ADAM // Applied Sciences. - 2020. - №10
15. N. Parikh, S. Boyd, et al. Proximal algorithms. // Foundations and Trends in Optimization. – 2014. - №1: pages 127–239
16. J. Yang, J. Wright, T. S. Huang and Y. Ma, "Image Super-Resolution Via Sparse Representation," in IEEE Transactions on Image Processing, №19, pages. 2861-2873, Nov. 2010
17. H., Alain & Z. Djemel. Image quality metrics: PSNR vs. SSIM. Proceedings of the ICPR, pages 2366-2369, 2010
18. H. Li, A. Kadav, I. Durdanovic, H. Samet, H.P.Graf,. Pruning Filters for Efficient ConvNets. // arXiv preprint – 2016, DOI: 10.48550/arXiv.1608.08710
19. J.H. Luo, J. Wu, W. Lin, ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression., In proceedings of IEEE International Conference on Computer Vision (ICCV) pages 5068-5076, 2017, DOI: 10.1109/ICCV.2017.541
20. Y. He, X. Zhang, J. Sun. Channel pruning for accelerating very deep neural networks. In proceedings of IEEE International Conference on Computer Vision (ICCV), pages 1398–1406, 2017.
21. Z. Huang, N. Wang. Data-driven sparse structure selection for deep neural networks. In ECCV, pages 304–320, 2018, DOI: 10.48550/arXiv.1707.01213
22. S. Lin, R. Ji, C. Yan, B. Zhang, D. Doermann. Towards optimal structured CNN pruning via generative adversarial learning., In CVPR, 2019, DOI: 10.48550/arXiv.1903.09291
23. V. Beiranvand, W. Hare, Y. Lucet. Best Practices for Comparing Optimization Algorithms // Optimization and Engineering. – 2017. - №18. DOI: 10.1007/s11081-017-9366-1.

Приложение A: Исходный код программы

import numpy as np

import matplotlib.pyplot as plt

import bisect

beale\_mod = lambda x,y : (1.5 - x + x\*y)\*\*2 + (2.25 - x + x\*(y\*\*2))\*\*2 + (2.625 - x + x\*(y\*\*3))\*\*2 + abs(x) + y\*\*2

beale\_g\_der = lambda x,y : np.array([x - x\*y - 1.5,x-x\*y\*\*2-2.25, x-x\*y\*\*3-2.625])

A = lambda y: np.array([[1-y],[1-y\*\*2],[1-y\*\*3]])

x = lambda x: np.array([x])

def numgrad(f,x,h = 1e-9):

dim = len(x)

coords = []

for i in range(dim):

coords.append(np.linspace(x[i]-h, x[i]+h,3))

x\_grid = np.meshgrid(\*coords)

V = f(\*x\_grid)

ind = tuple([1]\*dim)

res = np.array([x[ind] / h for x in np.gradient(V)])

res[0], res[1] = res[1], res[0]

return(res)

def SGD(f, st, lr1,lr2, num\_it, use\_cogd=False, g\_der=None, alpx=None,alpA=None,beta=None,k = None):

cur = st

path = []

path.append(cur)

for \_ in range(num\_it):

grad = numgrad(f,cur)

cur = cur - np.multiply([lr2,lr1],grad)

if(use\_cogd):

CoGD(g\_der,cur,path[-1],alpx,alpA,beta,k)

path.append(cur)

return cur,path

def conv\_cond(x, A, alpx, alpA):

R\_A = np.linalg.norm([A], ord=2)\*\*2

R\_x = np.linalg.norm([x], ord=1)

return (R\_x < alpx) and (R\_A >= alpA)

def CoGD(g\_der, cur, prev, alpx, alpA, lr3, k):

if conv\_cond(cur[0],cur[1],alpx, alpA):

c = g\_der(\*cur)@(A(cur[1])-A(prev[1]))/(x(cur[0])-x(prev[0]))

#c = g\_der(\*cur)\*(cur[1]-prev[1])/(cur[0]-prev[0])

c = np.sign(c)\*abs(c)\*\*k

cur[0] = cur[0] + 0.001\*lr3\*c\*prev[0]

return cur

a,path = SGD(beale\_mod,[1.5,1.8],0.001,0.001,500)

b,path\_alt = SGD(beale\_mod,[1.5,1.8],0.001,0.001,250,True,beale\_g\_der,1.4,0.5,0.01,2)

x1, x2 = zip(\*path)

alt\_x1, alt\_x2 = zip(\*path\_alt)

plt.figure(figsize=(10,8))

plt.plot(x1,x2,linewidth = 4,label='SGD')

plt.plot(alt\_x1,alt\_x2,linewidth = 4,label='SGD+CoGD')

plt.plot(2.13829, 0.194514, marker="o", markersize=10, markerfacecolor="red")

X, Y = np.meshgrid(np.linspace(0, 3, 100), np.linspace(-1, 2, 100))

Z = beale\_mod(X, Y)

plt.contour(X, Y, Z,levels=[0,1,2,3,4,5,7,10,15,17,20,30,50,60,70],cmap='plasma');

plt.colorbar();

plt.xlabel("x\_1")

plt.ylabel("x\_2")

plt.legend()

plt.show()

def Momentum(f, st, lr1,lr2, mnt, num\_it, use\_cogd=False, g\_der=None, alpx=None,alpA=None,beta=None, k =None):

cur = st

delta = np.array([0,0])

path = []

path.append(cur)

for \_ in range(num\_it):

grad = numgrad(f,cur)

delta = mnt\*delta - np.multiply([lr2,lr1],grad)

cur = cur + delta

if(use\_cogd):

CoGD(g\_der,cur,path[-1],alpx,alpA,beta,k)

path.append(cur)

return cur,path

a,path = Momentum(beale\_mod,[1.5,1.8],0.001,0.001,0.8,200)

b,path\_alt = Momentum(beale\_mod,[1.5,1.8],0.001,0.001,0.8,200,True,beale\_g\_der,1.5,0.5,0.01,2)

x1, x2 = zip(\*path)

alt\_x1, alt\_x2 = zip(\*path\_alt)

plt.figure(figsize=(10,8))

plt.plot(x1,x2,linewidth = 4,label='Momentum')

plt.plot(alt\_x1,alt\_x2,linewidth = 4,label='Momentum+CoGD')

plt.plot(2.13829, 0.194514, marker="o", markersize=10, markerfacecolor="red")

X, Y = np.meshgrid(np.linspace(0, 3, 100), np.linspace(-1, 2, 100))

Z = beale\_mod(X, Y)

plt.contour(X, Y, Z,levels=[0,1,2,3,4,5,7,10,15,17,20,30,50,60,70],cmap='plasma');

plt.colorbar();

plt.xlabel("x\_1")

plt.ylabel("x\_2")

plt.legend()

plt.show()

def Adam(f, st, lr1,lr2,beta1,beta2, num\_it, use\_cogd=False, g\_der=None, alpx=None,alpA=None,beta=None, k =None):

eps = 1e-15

cur = st

mt = np.array([0,0])

vt = np.array([0,0])

cb1 = beta1

cb2 = beta2

path = []

path.append(cur)

for \_ in range(num\_it):

grad = numgrad(f,cur)

mt = beta1\*mt + (1-beta1)\*grad

vt = beta2\*vt + (1-beta2)\*np.square(grad)

used\_mt = mt/(1-cb1)

used\_vt = vt/(1-cb2)

cb1\*=beta1

cb2\*=beta2

cur = cur - np.multiply([lr2,lr1],np.divide(used\_mt,np.add(np.sqrt(used\_vt),[eps,eps])))

if(use\_cogd):

CoGD(g\_der,cur,path[-1],alpx,alpA,beta,k)

path.append(cur)

return cur,path

a,path = Adam(beale\_mod,[1.5,1.8],0.05,0.05,0.9,0.999,200)

b,path\_alt = Adam(beale\_mod,[1.5,1.8],0.05,0.05,0.9,0.999,200,True,beale\_g\_der,1.4,0.5,0.01,2)

x1, x2 = zip(\*path)

alt\_x1, alt\_x2 = zip(\*path\_alt)

plt.figure(figsize=(10,8))

plt.plot(x1,x2,linewidth = 4,label='Adam')

plt.plot(alt\_x1,alt\_x2,linewidth = 4,label='Adam+CoGD')

plt.plot(2.13829, 0.194514, marker="o", markersize=10, markerfacecolor="red")

X, Y = np.meshgrid(np.linspace(0, 3, 100), np.linspace(-1, 2, 100))

Z = beale\_mod(X, Y)

plt.contour(X, Y, Z,levels=[0,1,2,3,4,5,7,10,15,17,20,30,50,60,70],cmap='plasma');

plt.colorbar();

plt.xlabel("x\_1")

plt.ylabel("x\_2")

plt.legend()

plt.show()

def ConvCond\_bm(A,x, alpA, alpx):

R\_A = np.linalg.norm(A, ord=2)\*\*2

R\_x = np.linalg.norm(x, ord=1)

return (R\_x < alpx) and (R\_A >= alpA)

def R(m,p):

return np.linalg.norm(m,ord=p)\*\*p

def CoGD\_bm(A\_prev, A\_cur, x\_prev,x\_cur,lr2,alpA, alpx, lr3, k, dradrx, newcond, poschange, keepsign,maxthresh = 2000,minthresh = 1e-4):

res = np.copy(x\_cur)

dRx = R(x\_cur,1)-R(x\_prev,1)

dRA = R(A\_cur,2)\*\*2-R(A\_prev,2)\*\*2

if (not newcond and ConvCond\_bm(A\_cur,x\_cur,alpA,alpx)) or (newcond and dRx > 1e-6 and abs(dRA/dRx) > dradrx):

dadx = np.divide(A\_cur-A\_prev,np.squeeze(x\_cur-x\_prev))

for i in range (dadx.shape[1]):

if (abs(x\_prev[i,0]) < minthresh or abs(dRx) < minthresh):

dadx[:,i] = 1

c = (2\*((A\_prev@x\_prev-b).T)@dadx).T

if(keepsign):

c = np.sign(c)\*np.power(np.absolute(c),k)

else:

c = np.power(np.absolute(c),k)

for i in range (c.shape[0]):

if abs(c[i,0]) > maxthresh:

c[i,0] = np.sign(c[i,0])\*maxthresh

if(poschange == 0):

res = x\_cur + lr3\*lr2\*np.multiply(c,x\_prev)

elif (poschange == 1):

res = x\_prev + lr3\*lr2\*np.multiply(c,x\_prev)

elif (poschange == 2):

res = x\_cur - lr3\*lr2\*np.multiply(c,x\_prev)

return res

def med\_sorted(m):

m\_size = len(m)

if m\_size%2 == 1:

return m[m\_size//2]

else:

return (m[m\_size//2 - 1]+m[m\_size//2])/2

def Adam\_bm(b, x\_len, lamb, lr1,lr2,A0,x0, beta1, beta2, num\_it, use\_cogd=False,

newcond=False,poschange=0,keepsign = False,alpA=None,dradrx=None,lr3=None, k =None):

eps = 1e-15

A = np.copy(A0)

x = np.copy(x0)

mt = np.array([np.zeros(A.shape),np.zeros(x.shape)],dtype=object)

vt = np.copy(mt)

cb1 = beta1

cb2 = beta2

G = []

AR = []

xR = []

TEST = []

for t in range(num\_it):

grad = np.array([2\*(A@x-b)@(x.T)+2\*A,2\*(A.T)@(A@x-b)+lamb\*np.sign(x)],dtype=object)

mt = beta1\*mt + (1-beta1)\*grad

vt = beta2\*vt + (1-beta2)\*np.square(grad)

used\_mt = mt/(1-cb1)

used\_vt = vt/(1-cb2)

cb1\*=beta1

cb2\*=beta2

A\_prev = np.copy(A)

x\_prev = np.copy(x)

A = A - lr1\*np.divide(used\_mt[0],np.add(np.sqrt(used\_vt[0]),np.full(used\_vt[0].shape,eps)))

x = x - lr2\*np.divide(used\_mt[1],np.add(np.sqrt(used\_vt[1]),np.full(used\_vt[1].shape,eps)))

bisect.insort(AR,R(A,2))

bisect.insort(xR,R(x,1))

TEST.append(R(x,1))

if(use\_cogd):

x = CoGD\_bm(A\_prev, A, x\_prev, x, lr2, alpA, med\_sorted(xR),lr3,k,dradrx,newcond,poschange,keepsign)

G.append(np.linalg.norm(b-A@x,ord=2)\*\*2 + lamb\*np.linalg.norm(x,ord=1)+np.linalg.norm(A,ord=2))

plt.plot(TEST)

return A,x,G

#b = np.array([[2,5,-3]]).T

#b = np.array([[0,10,7,3,2,5,-3,-5]]).T

#b = np.array([[5, 7, 0, 9, -6, -2, 0, 5, -3, 9, 7, -3, 3, 10, 8, -9, 7, -6, 0, -8]]).T

b = np.random.randint(-1,10,size=(50,1))

np.linalg.norm(b,ord=2)\*\*2

#x\_len = 2

x\_len = 100

alpA = 30

n\_it = 3000

st = 50

A0 = np.random.rand(len(b),x\_len)

x0 = np.random.rand(x\_len,1)

Ar,xr,Gr = Adam\_bm(b,x\_len,1,0.05,0.05,A0,x0,0.9,0.999,n\_it)

Ar\_cgd,xr\_cgd,Gr\_cgd = Adam\_bm(b,x\_len,1,0.05,0.05,A0,x0,0.9,0.999,n\_it,True,False,2,False,alpA,50,0.001,2)

Ar\_modcgd,xr\_modcgd,Gr\_modcgd = Adam\_bm(b,x\_len,1,0.05,0.05,A0,x0,0.9,0.999,n\_it,True,True,2,False,alpA,100,0.0005,2)

plt.figure(figsize=(10,8))

plt.show()

#Ar\_truemodcgd,xr\_truemodcgd,Gr\_truemodcgd = Adam\_bm(b,x\_len,1,0.05,0.05,A0,x0,0.9,0.999,n\_it,True,False,2,False,alpA,50,0.005,2)

plt.figure(figsize=(10,8))

plt.plot(Gr[st:-1],label ='Adam')

plt.plot(Gr\_cgd[st:-1],label = 'Adam + CoGD')

plt.plot(Gr\_modcgd[st:-1],label = 'Adam + ModCoGD')

#plt.plot(Gr\_truemodcgd[st:-1],label = 'Adam + TrueModCoGD')

plt.legend()

plt.show()

print('Adam min\_G:', Gr[-1])

print('Adam + CoGD min\_G:', Gr\_cgd[-1])

print('Adam + ModCoGD min\_G:', Gr\_modcgd[-1])

#print('Adam + TrueModCoGD min\_G:', Gr\_truemodcgd[-1])

eps = 1e-1

baseline = np.argmin(Gr)

print('Baseline-Adam',baseline)

print('CoGD',np.argwhere(Gr\_cgd<Gr[baseline]+eps)[0,0])

print('ModCoGD',np.argwhere(Gr\_modcgd<Gr[baseline]+eps)[0,0])

#print('TrueModCoGD',np.argwhere(Gr\_truemodcgd<Gr[baseline]+eps)[0,0])