Министерство образования Республики Беларусь
Учреждение образования
“БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ”

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина:

РЕФЕРАТ
по курсу “Машинное обучение”
на тему

Градиентный спуск, методы оценивания градиента

Выполнил
магистрант группы 456241
Герчик А. В.

Проверил
доцент кафедры информатики
Боброва Н. Л.

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Машинное обучение (МО) представляет собой один из ключевых подходов в разработке интеллектуальных систем, позволяющий компьютерам обучаться на основе данных и улучшать свои результаты без прямого программирования.

Одной из центральных задач МО является нахождение оптимальных параметров моделей, таких как веса в линейной регрессии или параметры нейронных сетей. Для решения этой задачи применяется градиентный спуск - итеративный метод оптимизации, который используется для минимизации функции потерь.

Градиентный спуск лежит в основе большинства алгоритмов машинного обучения и играет важную роль в обучении глубоких нейронных сетей. Он использует информацию о градиенте функции потерь, указывающем направление наибольшего увеличения значения, чтобы корректировать параметры в направлении минимизации ошибки. В зависимости от того, как именно оценивается градиент, существуют различные модификации алгоритма, такие как стохастический градиентный спуск, пакетный (mini-batch) метод и адаптивные варианты, включая Adam, Adagrad и RMSprop.

Кроме классических методов, возникающие проблемы - такие как шум в данных или высокая размерность пространства параметров требуют внедрения специализированных методов оценивания и приближения градиентов.

Например, численное дифференцирование, метод конечных разностей и подходы, использующие автоматическое дифференцирование, обеспечивают гибкость и точность при работе с градиентом в сложных системах. Эти методы и их правильное применение оказывают значительное влияние на качество и скорость сходимости моделей.

В настоящей работе будут рассмотрены основы градиентного спуска, его модификации и методы оценивания градиента. Анализируя особенности различных подходов, исследуется их применимость в разных задачах машинного обучения и влияние на процесс оптимизации.

1 ГРАДИЕНТНЫЙ СПУСК. МАТЕМАТИЧЕСКИЕ ОСНОВЫ

1.1 Основы градиентного спуска

Градиентный спуск - это итеративный метод оптимизации, который позволяет минимизировать функцию, соответствующую ошибке или потере в моделях машинного обучения. Его идея заключается в обновлении параметров модели таким образом, чтобы в каждой итерации приближаться к минимальному значению функции потерь.

Для объяснения градиентного спуска рассмотрим задачу минимизации функции потерь $L(\theta)$, где θ - вектор параметров модели. Если функция $L(\theta)$ дифференцируема, то направление наибольшего возрастания функции определяется её градиентом $\nabla L(\theta)$. Чтобы найти минимум, необходимо изменять параметры θ в противоположном направлении от градиента:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla L(\theta_t)$$

где:

θ_t - значения параметров на t-й итерации,

η - шаг обучения (learning rate), определяющий, на сколько изменяются параметры за одну итерацию,

$\nabla L(\theta_t)$ - градиент функции потерь, вычисленный в точке θ_t.

Шаг обучения η - это важный гиперпараметр. Если он слишком велик, алгоритм может "перепрыгнуть" минимум, а если слишком мал - процесс обучения будет медленным или застрянет в локальном минимуме.

На рисунке 1.1 изображена работа метода при выбранном малом шаге. Если выбранный шаг слишком маленький потребуется много итераций чтобы добраться до минимума.

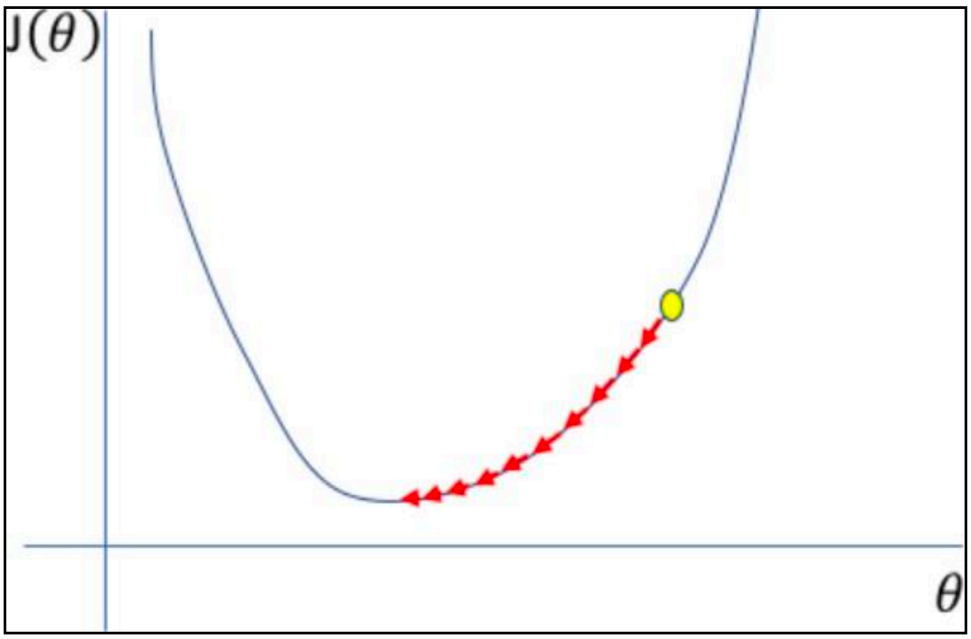

Рисунок 1.1 - Работа алгоритма с малым шагом

На рисунке 1.2 изображена работа метода при выбранном большом шаге. Если выбранный шаг слишком большой высока вероятность промахнуться мимо минимума.

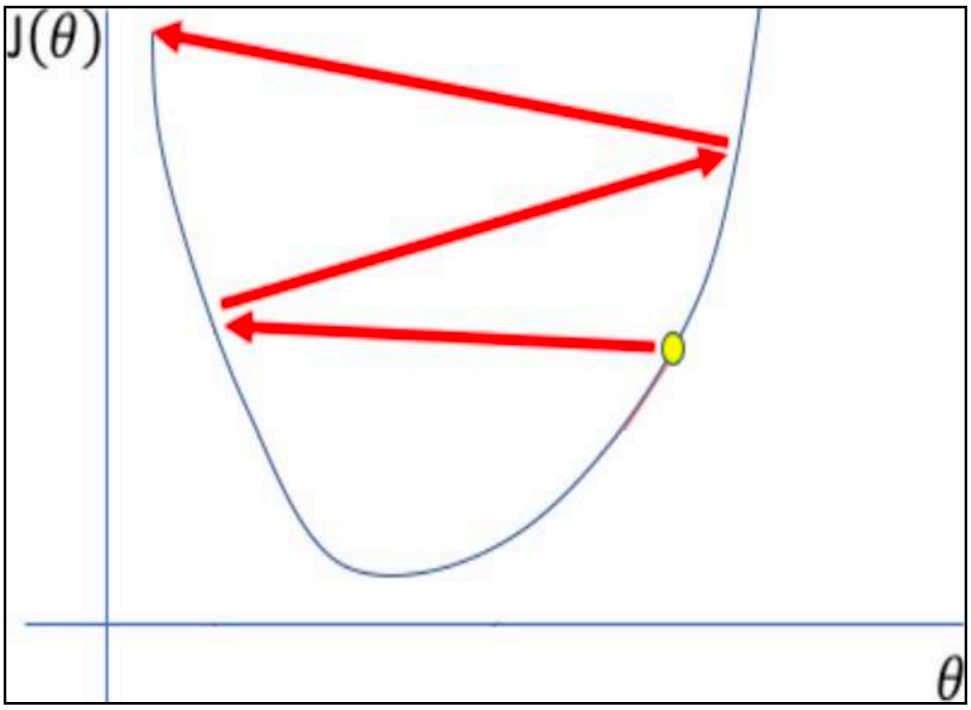

Рисунок 1.2 - Работа алгоритма с большим шагом

На рисунке 1.3 изображена работа метода при хорошо подобранном шаге[1].

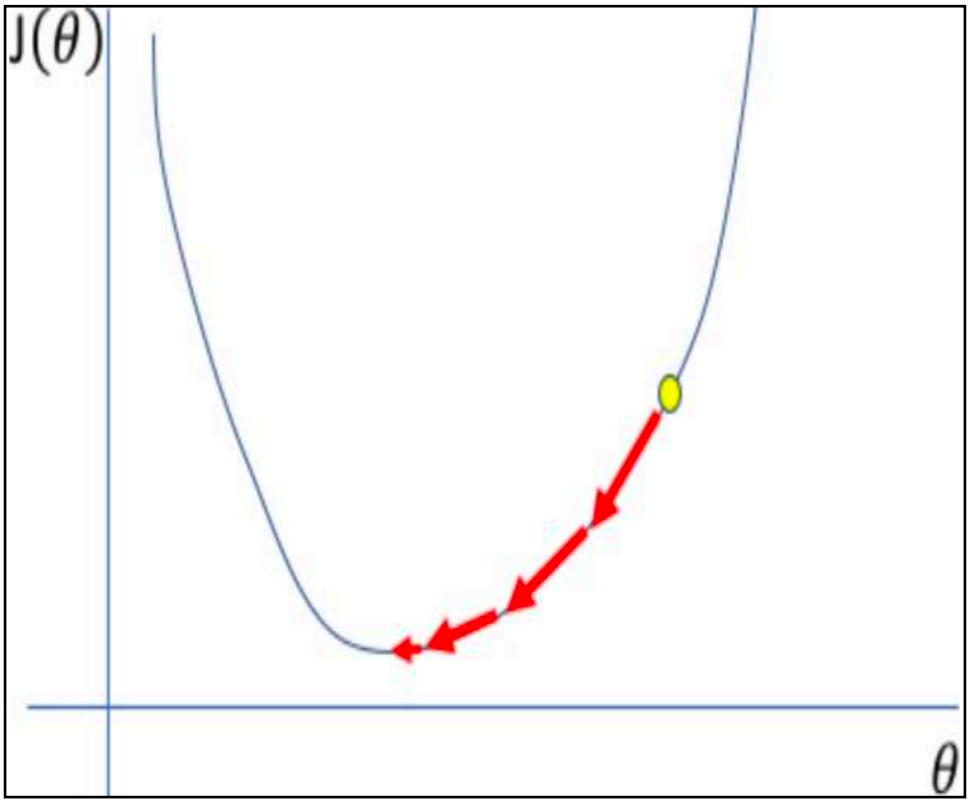

Рисунок 1.3 - Работа алгоритма с хорошо подобранным шагом

1.2 Различные варианты градиентного спуска

1.2.1 Стандартный пакетный (Batch) градиентный спуск

Этот вариант вычисляет градиент, используя все обучающие данные на каждом шаге, что может быть неэффективно для больших наборов данных.

Формула такого варианта градиентного спуска:

$$\theta_{t+1} = \theta_t - \eta \cdot 1/n \sum n \nabla L_i(\theta_t)$$

На рисунке 1.4 представлен пакетный градиентный спуск.

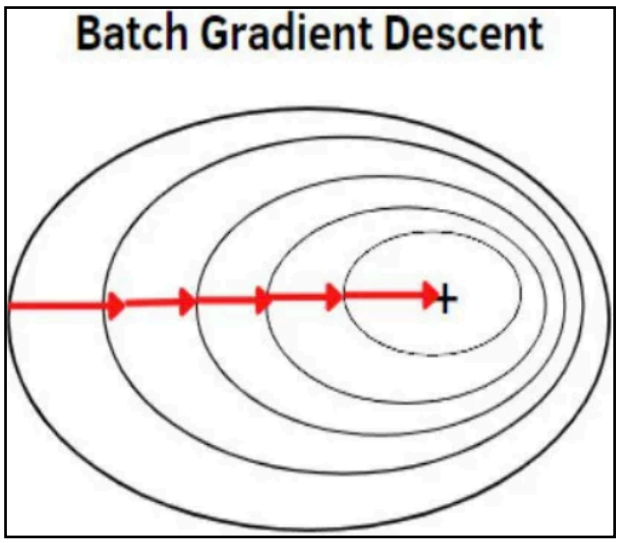

Рисунок 1.4 - Пакетный градиентный спуск

1.2.2 Стохастический (SGD) градиентный спуск

Данный вариант вычисляет градиент на основе одного обучающего примера, выбранного случайным образом, на каждом шаге, что делает алгоритм более быстрым и способным к обновлению модели онлайн.

Формула такого варианта градиентного спуска:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla L_i(\theta_t)$$

На рисунке 1.5 представлен стохастический градиентный спуск.

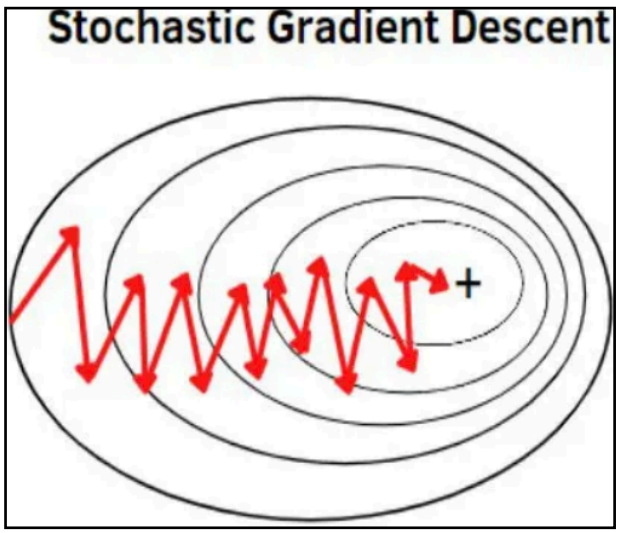

Рисунок 1.5 - Стохастический градиентный спуск

1.2.3 Мини-пакетный (Mini-Batch) градиентный спуск

В свою очередь, данный вариант является компромиссом между двумя предыдущими методами - вычисляет градиент на основе небольшой выборки данных (мини-пакета), что позволяет более эффективно использовать ресурсы и обеспечивает более стабильное схождение[2].

На рисунке 1.6 представлен мини-пакетный градиентный спуск.

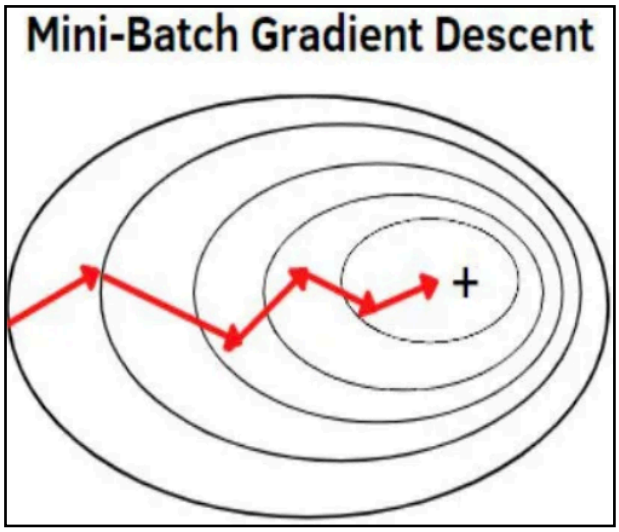

Рисунок 1.6 - Мини-пакетный градиентный спуск

1.2.4 Adagrad

Adagrad адаптирует шаг обучения для каждого параметра модели на основе предыдущих значений градиентов. Он усиливает шаг обучения для редких признаков и уменьшает для часто встречающихся, что делает его полезным для задач с разреженными данными (например, NLP или рекомендательные системы) [3].

Правило обновления:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla L(\theta_t)$$

где:

$G_T = \sum_{i=1}^{t} \nabla L(\theta_i)^2$ - сумма квадратов всех предыдущих градиентов,

ϵ - небольшая константа для избежания деления на ноль.

При использовании данного подхода может возникнуть проблема, а именно со временем накопление квадратов градиентов может сделать шаг обучения слишком маленьким, и процесс оптимизации остановится.

1.2.5 RMSProp

RMSprop улучшает Adagrad, используя экспоненциальное затухание для градиентов. Вместо простого накопления всех предыдущих градиентов используется скользящее среднее, что позволяет шагам обучения оставаться на разумном уровне в течение всего процесса. Это делает RMSprop особенно эффективным для задач с нелинейными функциями потерь и больших нейронных сетей [4].

Правило обновления:

$$\theta_{t+1} = \theta_t - \frac{\eta}{E\,|\,\nabla L(\theta_t)^2\,| + \epsilon} \cdot \nabla L(\theta_t)$$

где:

$E\,|\,\nabla L(\theta_t)^2\,|$ - экспоненциально взвешенное среднее квадратов градиентов,

ϵ - небольшая константа для стабилизации.

1.2.6 Adam

Adam объединяет идеи из Adagrad и RMSprop и использует два экспоненциально затухающих средних: одно для первого момента (градиента) и другое для второго момента (квадрата градиента). Это позволяет добиться эффективной и стабильной оптимизации, особенно в задачах с большими и сложными нейронными сетями.

Adam успешно применим для обучения глубоких нейронных сетей, поскольку он сочетает быстрое сужение в начале обучения и стабилизацию на последних этапах [5].

Адаптивные алгоритмы, такие как Adagrad, RMSprop и Adam, значительно улучшили качество и скорость обучения моделей машинного обучения. Они позволяют обойти ограничения стандартного градиентного спуска, автоматически подстраивая шаг обучения для каждого параметра. Adam стал одним из самых популярных методов для обучения нейронных сетей благодаря балансу между скоростью и стабильностью сходимости.

2 МЕТОДЫ ОЦЕНИВАНИЯ ГРАДИЕНТА

Градиент - ключевой элемент в большинстве оптимизационных задач машинного обучения, таких как обучение нейронных сетей и регрессионных моделей. В идеале градиент вычисляется аналитически, однако это не всегда возможно, особенно в сложных или высокоразмерных системах. Именно здесь на помощь приходят методы оценивания градиента. В данном разделе будет рассмотрено, зачем необходимы такие методы, какие подходы существуют, их плюсы, минусы и применимость в разных задачах.

2.1 ОСНОВНЫЕ ПОДХОДЫ

Существует несколько методов, которые различаются по точности, сложности и вычислительным затратам. Ниже будут приведены три ключевых подхода.

2.1.1 Численное дифференцирование

Этот метод используется для аппроксимации градиента в точках, где аналитическое дифференцирование невозможно. Основная идея заключается в том, что мы приближенно находим производную по формуле:

$$\frac{dx}{df} \approx \frac{f(x+h) - f(x)}{h}$$

где:

h - небольшое положительное число, представляющее шаг.

f(x) - значение функции в точке x

Этот метод может быть неточным, если шаг h h слишком велик (приближение грубое) или слишком мал (возникают ошибки округления).

2.1.2 Метод конечных разностей

Метод конечных разностей является улучшенной версией численного дифференцирования. Существует два основных варианта: прямые разности и центральные разности.

Прямые разности (forward difference) - простой, но менее точный метод:

$$\nabla f(x) \approx \frac{f(x+h) - f(x)}{h}$$

Центральные разности (central difference) - более точны, так как учитывают информацию с двух сторон от точки x:

$$\nabla f(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Данный метод легко реализуем и подходит для произвольных функций, однако для каждой компоненты параметра требуется отдельный вызов функции, что делает метод медленным на больших данных [6].

2.1.3 Автоматическое дифференцирование (Autodiff)

Автоматическое дифференцирование - это метод, который точно вычисляет производные, комбинируя символическое и численное дифференцирование. Он не страдает от ошибок аппроксимации и может применяться даже для сложных функций, таких как нейронные сети.

Есть два ключевых подхода к автодифференцированию:

- Прямой режим (forward mode): Вычисляет производную по одному входу за раз. Полезен, если функция имеет больше входов, чем выходов.
- Обратный режим (reverse mode): Вычисляет все производные одновременно.

Это делает его идеальным для обучения нейронных сетей, где у нас много параметров и один выход (например - ошибка).

Если говорить в контексте разработки на языке программирования Python нельзя не упомянуть популярные библиотеки для автодифференцирования:

- TensorFlow
- PyTorch
- JAX

Данный подход предоставляет точность, гибкость и работает для сложных функций, однако, в свою очередь требует дополнительной памяти и не всегда прозрачен для пользователя.[7]

2.2 СРАВНЕНИЕ ПОДХОДОВ

В таблице 2.1 будет проведено сравнение вышеупомянутых подходов оценивания методов градиента.

Таблица 2.1 - Сравнение методов оценивания градиента

Метод	**Точность**	**Вычислительные затраты**	**Применимость**
Численное дифференцирование	Низкая	Низкая	Простые функции, Черный ящик
Конечные разности	Средняя	Средняя	Общие задачи, Симуляции
Автоматическое дифференцирование	Высокая	Высокая	Глубокие нейтронные сети, Сложные модели

Методы оценивания градиента являются фундаментальными инструментами оптимизации в машинном обучении и играют ключевую роль в обучении моделей. В зависимости от структуры задачи, сложности функции потерь и доступных вычислительных ресурсов можно выбрать наиболее подходящий метод. В таблице 2.2 будет приведена инструкция, как верно выбрать метод оценивания.

Таблица 2.2 - Инструкция выбора метода оценивания

Ситуация	Рекомендуемый метод	Обоснование
Нужно быстро протестировать идею на простой функции	Численное дифференцирование	Быстрое и простое в реализации
Модель с ограниченным количеством параметров и без аналитической производной	Метод конечных разностей	Точный, хотя и затратный по вычислениям
Обучение нейронной сети или другой модели с большим числом параметров	Автоматическое дифференцирование	Высокая точность и эффективность на больших данных
Проверка правильности градиентов	Конечные разности или численное дифференцирование	Для тестирования корректности autodiff

С развитием вычислительных мощностей и алгоритмов автодифференцирование становится все более популярным и применимым в разных областях. Понимание сильных и слабых сторон каждого метода позволяет эффективно использовать их в соответствующих задачах.

В машинном обучении автоматическое дифференцирование стало де-факто стандартом, однако для небольших задач или валидации результатов методы

конечных разностей и численного дифференцирования по-прежнему остаются полезными.

Специалистам важно владеть этими методами, чтобы уметь адаптировать их под конкретные условия и находить оптимальные решения для сложных задач.

ВЫВОДЫ

В ходе работы были подробно рассмотрены основные аспекты градиентного спуска и методов оценивания градиента, которые являются центральными элементами оптимизации в машинном обучении. Эти инструменты позволяют моделям эффективно находить оптимальные параметры, минимизируя ошибку и улучшая качество предсказаний. Подведем основные выводы.

Градиентный спуск - ключевой алгоритм для обучения моделей, от простых регрессионных задач до сложных глубоких нейронных сетей. Были рассмотрены его вариации, включая стандартный, стохастический и мини-батч подходы. Каждый из них имеет свои плюсы: стандартный - более точен, стохастический - быстрее сходится, а мини-батч комбинирует их преимущества.

Адаптивные методы оптимизации (Adam, Adagrad и RMSprop) расширяют возможности градиентного спуска, позволяя автоматически регулировать шаг обучения в зависимости от параметров. Это делает оптимизацию более стабильной, особенно на сложных функциях с большим числом параметров, где традиционные алгоритмы могут сталкиваться с проблемами застревания в локальных минимумах.

Методы оценивания градиента, такие как численное дифференцирование, конечные разности и автоматическое дифференцирование, дают гибкость в задачах, где аналитические градиенты сложно получить. Численные методы полезны для небольших экспериментов и проверки моделей, но более эффективное решение в больших системах предоставляет автодифференцирование, ставшее стандартом в современных фреймворках машинного обучения.

Таким образом, выбор подходящего алгоритма оптимизации и метода оценки градиента зависит от конкретной задачи. При небольших объемах данных можно использовать численные методы, для крупных моделей - адаптивные алгоритмы с автодифференцированием. Компетентное использование этих инструментов позволяет разрабатывать модели, которые эффективно обучаются и обеспечивают высокое качество предсказаний.

Будущее этих методов связано с их интеграцией в новые области, такие как квантовые вычисления и распределенные системы, что открывает новые горизонты для применения машинного обучения.

В заключение, глубокое понимание принципов градиентного спуска и методов оценки градиентов является ключом к созданию эффективных моделей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Линейная регрессия / Н.Л. Боброва // БГУИР - 2024, С. 28-30.

[2] Что такое Градиентный спуск / И.С. Шаповал // AppTractor - 2024.

[3] Understanding the AdaGrad Optimization Algorithm: An Adaptive Learning Rate Approach / S. Brijesh // Medium - 2023.

[4] RMSProp / J.Huang // Cornell University - 2020.

[5] Adaptive Moment Estimation (ADAM) / S. Improve // MarketMuse - 2017.

[6] Numerical Optimization / J. Nocedal, S. Wrigh // UCI - 2006.

[7] Deep Learning / I. Goodfellow, Y. Bengio, A. Courville // MIT - 2016.