строится векторное представление («мешок слов») и все данные (сырые данные с сайта, текст, фильтрованный текст, исходные документы и «мешок слов») сохраняются в хранилище.

Выбранная архитектура СКА позволяет её модернизировать и функционально наращивать в процессе эксплуатации. Так, планируется нарастить компонент «хранилище» и дополнить компонент «библиотека модулей» векторными алгоритмами и алгоритмами нейровычислений, что существенно расширит область применения СКА и позволит получать информацию в интересах сотрудников университета, государственных и частных организаций.

Список использованных источников:

1. Data Never Sleeps 6.0 [Электронный ресурс] / Режим доступа: https://www.domo.com/learn/data-neversleeps-6 Дата доступа: 18.01.2019.

2. Пилецкий, И. И. Аналитический комплекс анализа данных из открытых интернет источников / И. И. Пилецкий, В. А. Прытков, Н. А. Волорова // BIG DATA Advanced Analytics: collection of materials of the fourth international scientific and practical conference, Minsk, Belarus, May 3 – 4, 2018 – Minsk, BSUIR, 2018. – P. 193 – 199.

3. Прытков, В. А. Анализ репозитория университета с использованием графовой базы данных / В. А. Прытков, И. И. Пилецкий, Н. А. Волорова // BIG DATA Advanced Analytics: collection of materials of the fourth international scientific and practical conference, Minsk, Belarus, May 3 – 4, 2018. – Minsk, BSUIR, 2018. – P. 177 – 183.

СРАВНЕНИЕ ВАРИАЦИЙ ГРАДИЕНТНОГО СПУСКА НА ПРИМЕРЕ ЗАДАЧИ РАСПОЗНАВАНИЯ СИМВОЛОВ

Евжик Д.А., Подвальников Д.С., Тишковский М.А.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Анисимов В.Я. – к.ф.-м.н.,доцент

В данной работе приведен сравнительный анализ вариаций градиентного спуска, который демонстрирует применимость отдельного метода для задачи распознавания символов. Результаты исследования в данной работе позволяют сделать выбор наиболее подходящего по скорости и качеству обучения метода при решении задач.

В настоящее время можно заметить глубокий интерес к области машинного обучения. Как известно, для построения методов машинного обучения используются средства методов оптимизации для нахождения минимума целевой функции в некоторой области конечномерного векторного пространства.

В ходе исследования были проанализированы три популярных метода нахождения локального минимума функции. Обычный, пакетный и стохастический градиентный спуск.

В данной работе был рассмотрен набор данных *notMNIST* [1], который состоит из изображений размерностью 28×28 первых 10 букв латинского алфавита.

Градиентный спуск – это метод нахождения локального минимума функции путем движения по вектору антиградиента. Этот метод является очень распространенным в машинном обучении для нахождения минимума функции потерь. В начале работы алгоритма задается начальное приближение минимума, затем вычисляется значение градиента функции потерь на всей обучающей выборке, последним шагом является сложение начального приближения с полученным значением градиента со знаком минус. Стохастический градиентный спуск является вариацией обычного градиентного спуска. Его отличие заключается в том, что вычисление значения градиента производится не на всей обучающей выборке, а на одном случайном элементе, засчет этого скорость обучения значительно возрастает. Пакетный градиентный спуск схож со стохастическим, но вместо одного экземпляра из выборки берется какая-то ее часть, несколько экземпляров, что позволяет добиться как хорошей скорости обучения, так и более плавного движения к минимуму функции.

Описанные три вариации градиентного спуска были применены для обучения нейронной сети на наборе данных *notMNIST* размером 10000 элементов. Архитектура нейронной сети состояла из пяти скрытых слоев с 1024, 1024, 512, 256 и 128 нейронами на каждом слое соответственно. На всех скрытых слоях в качестве функции активации была использована *relu* [2] (кусочно-линейная функция), на выходном слое – *softmax* [3] (обобщение логистической функции для многомерного случая). В результате при обучении на 10 эпохах получились следующие графики, показывающие зависимость ошибки обучения от эпохи.

Результаты данного эксперимента в виде зависимости ошибки от порядкового номера эпохи представлены на рисунке 1, рисунке 2 и рисунке 3.

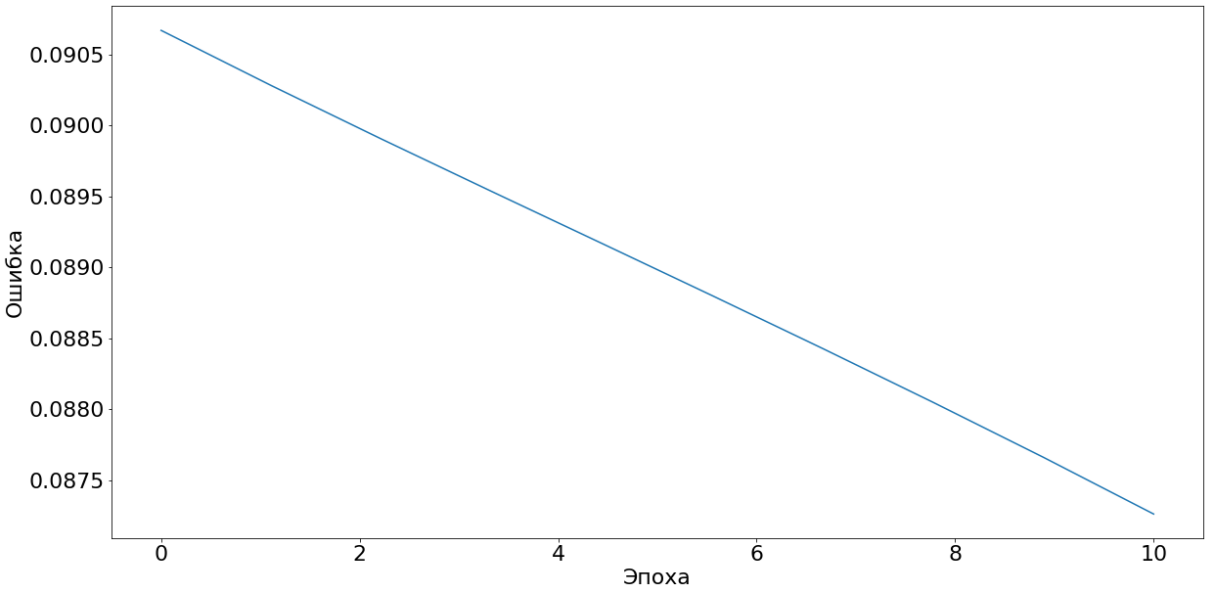

Рисунок 1 – Изменение ошибки от эпохи при испоользовании обычного градиентного спуска

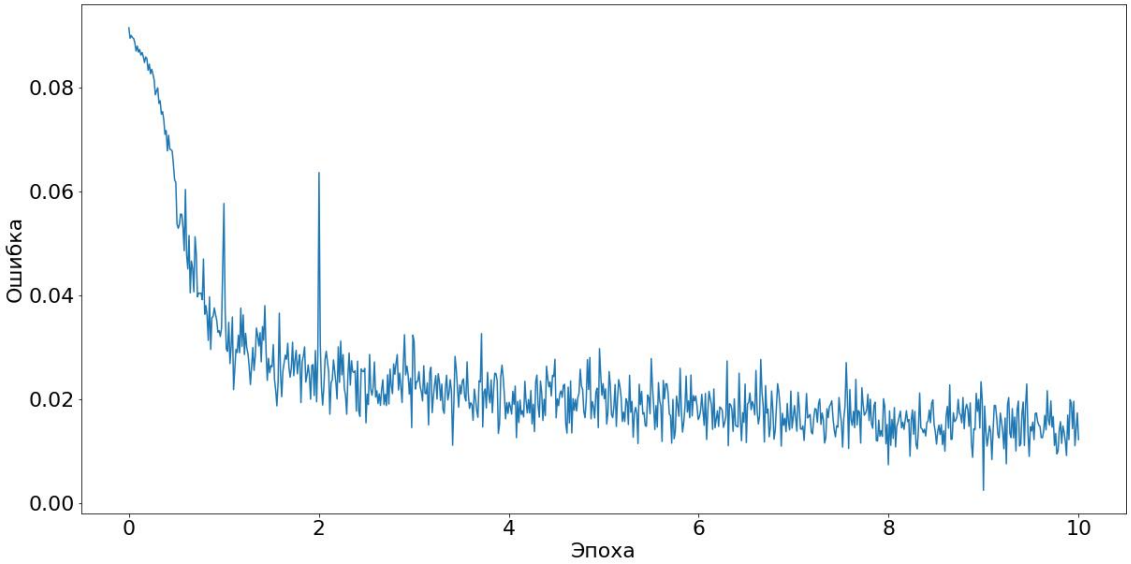

Рисунок 2 – Изменение ошибки от эпохи при использовании пакетного градиентного спуска

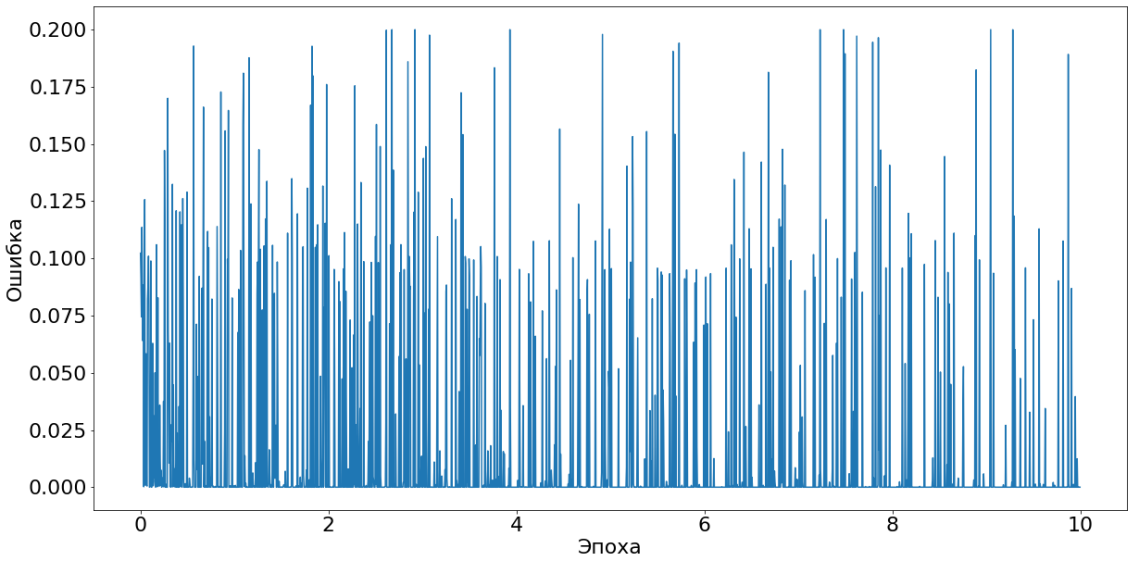

Рисунок 3 – Изменение ошибки от эпохи при использовании стохастического градиентного спуска

В результате применения обычного градиентного спуска в среднем на одну эпоху тратилось в среднем 93 мс, при этом точность предсказания на тестовой выборке составила 0.32. Из этого видно, что 10 эпох не достаточно для достижения высокой точности. При использовании пакетного градиентного спуска с размером пакета 128 элементов среднее время обучения одной эпохи составило 153 мс, качество обучения – 0.78. Для стохастического градиентного спуска на обучение одной эпохи потребовалось 61 с, качество обучения составило 0.84. Для обучения использовался шаг равный 0.5. Для оценки качеств модели использовалась метрика *accuracy* [4].

Из полученных результатов можно сделать вывод о том, что обычный градиентный спуск работает гораздо медленнее при достижении минимальной ошибки в сравнении со стохастическим и пакетным градиентный спуском, однако он значительно лучше минимизирует функцию. Стохастический градиентный спуск лучше показывает себя на задачах, в которых для обучения требуется небольшое количество данных, для более сложных задач с большим набором данных в обучающей выборке достижение минимума может также оказаться долгим процессом. Для больших выборок подходит пакетный градиентный спуск, он быстрее в сравнении с обычным и в то же время быстрее сходится к минимуму в сравнении со стохастическим. Таким образом, в случае, когда достаточно найти значение близкое к минимуму за приемлемое время, имеет смысл использовать стохастический или пакетный градиентный спуск в зависимости от решаемой задачи.

Список использованных источников:

1. Набор данных notMNIST [Электронный ресурс] – Режим доступа: http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html
2. Функция relu [Электронный ресурс] – Режим доступа: https://arxiv.org/pdf/1803.08375.pdf
3. Функция softmax [Электронный ресурс] – Режим доступа: https://arxiv.org/pdf/1811.03378.pdf
4. Определение метрики accuracy в математической статистике [Электронный ресурс] – Режим доступа: http://gsp.humboldt.edu/OLM_2015/Courses/GSP_216_Online/lesson6-2/metrics.html

ОБЛАЧНАЯ ИНФРАСТРУКТУРА КАК КОД НА ПРИМЕРЕ СИСТЕМЫ ДЛЯ АВТОМАТИЗАЦИИ КОНТРОЛЯ ДОСТУПА К SAP

Киселёв Д.О.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Егорова Н.Г. – канд. техн. наук,
доцент каф. Информатики

В настоящее время использование облачных сервисов стало стандартом при разработке программного обеспечения. Будь это маленький сайт или большая промышленная система. Потребности в мощности ресурсов очень динамические: сегодня систему использует сто человек, через полгода - тысяча, следовательно, становится вопрос простого перехода из одной инфраструктуры на другую. В случае использования облачных сервисов и подхода «инфраструктура как код», данный процесс максимально упрощается.

Инфраструктура как код (англ. Infrastructure as code (IaC)) – это способ создания и управления ресурсами для центров хранения и обработки данных методом их описания в исходном коде. Данный подход приходит на замену настройке оборудования вручную или с помощью интерактивных инструментов. Для описания инфраструктуры чаще всего используется декларативный язык, так же оно может находится в системе контроля версий. Подходы IaC продвигаются для облачных вычислений, которые иногда называют инфраструктура как сервис (IaaS).

Преимущества IaC можно описать в трех направлениях: стоимость, скорость, риски. Снижение стоимости происходит как за счет более рационального использования вычислительных ресурсов, так и за счет снижения трудозатрат на их обслуживание, позволяя больше сфокусироваться на решении проблем бизнеса. Автоматизация инфраструктуры обеспечивает скорость за счет более быстрого выполнения при настройке инфраструктуры. Автоматизация убирает риск, связанный с человеческой ошибкой, что позволяет уменьшить время простоя и повысить надежность. Эти преимущества помогают корпоративному сегменту двигаться в направлении развития практик DevOps.

Архитектура системы для автоматизации и контроля доступа к SAP построена с использованием большого количества облачных сервисов, причем разных поставщиков. Изначально единственным способом создания нового окружения было исполнения шагов, описанных в текстовом документе вручную. Для того, чтобы развернуть новое окружение, тратилось очень много времени. Так же этот процесс был подвергнут ошибкам из-за разночтения инструкции. Соответственно стал вопрос об автоматизации развертывания инфраструктуры. Схема архитектуры представлена на рисунке 1.