

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Инженерная школа информационных технологий и робототехники

Отделение информационных технологий

09.04.01 «Информатика и вычислительная техника»

Исследование возможностей платформ Yandex Datasphere и VK Cloud ML
Platform на задаче классификации рукописных цифр

КУРСОВОЙ ПРОЕКТ

по дисциплине:

Облачные технологии

Исполнитель:

студент группы

8BM22

Ямкин Н.Н.

16.03.2023

Руководитель:

преподаватель

Ботыгин И.А.

Томск – 2023

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ	2
ВВЕДЕНИЕ	3
ТРЕБОВАНИЯ К ОБЛАЧНОЙ ИНФРАСТРУКТУРЕ	6
БЫСТРОДЕЙСТВИЕ СИСТЕМЫ	7
ПРОГРАММНЫЙ КОД	8
АЛГОРИТМ ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ	11
ПРАКТИЧЕСКАЯ ЧАСТЬ	13
YANDEX DATASPHERE	14
Основной функционал	14
Тестирование быстродействия	19
Тарификация	20
Особенности платформы	21
VK CLOUD ML PLATFORM	22
Основной функционал	22
Тестирование быстродействия	28
Тарификация	29
Особенности платформы	31
СРАВНИТЕЛЬНЫЙ АНАЛИЗ	32
ЗАКЛЮЧЕНИЕ	33
СПИСОК ИСТОЧНИКОВ	34

ВВЕДЕНИЕ

Сегодня обработка больших объемов информации является критически важной в различных областях, включая бизнес, науку, технологии, медицину, образование и в многих других сферах деятельности. Существует огромное количество данных, создаваемых и собираемых каждый день, и умение эффективно обрабатывать эти данные может дать конкурентное преимущество в любой области.

Процессинг больших объемов данных позволяет находить скрытые закономерности и тренды, выявлять причинно-следственные связи, строить прогнозы и принимать более осознанные решения. Например, в бизнесе анализ данных может помочь в принятии решений о маркетинге, оптимизации производства, улучшении качества продукции и т.д. В медицине анализ больших данных может помочь выявить причины заболеваний и разработать новые методы лечения.

Таким образом, обработка больших объемов информации имеет большое значение в современном мире, и способность анализировать, обрабатывать и использовать данные становится все более важной.

Однако, этот процесс требует значительных вычислительных ресурсов. Вот несколько причин, почему это происходит:

1. Большой объем данных: обработка и хранение больших объемов данных требует много вычислительной мощности. Например, при обработке медицинских изображений, анализе данных социальных медиа или биологических данных может потребоваться обработка терабайтов информации. Чем больше данных, тем больше вычислительных ресурсов нужно для их обработки.

2. Сложность алгоритмов: анализ данных может потребовать сложных алгоритмов и моделей машинного обучения, которые могут быть вычислительно затратными. Некоторые алгоритмы машинного обучения,

такие как нейронные сети, требуют большого количества вычислительных ресурсов для обучения моделей и предсказания результатов.

3. Скорость обработки: результаты анализа должны быть получены в разумные сроки. Чтобы обеспечить быструю обработку данных, могут быть использованы параллельные вычисления или распределенные системы.

Оптимальная компьютерная система позволяет снизить издержки на содержание системы и сохраняет качество работы на приемлемом уровне.

На сегодняшний день покупка собственного вычислительного оборудования обойдется дорого и не факт, что оно будет использовано эффективно.

Использование облачных сервисов может облегчить задачу анализа большого объема информации, поскольку облачные провайдеры могут предоставлять вычислительные мощности и хранилище данных в соответствии с требованиями пользователя.

Облачные сервисы могут быть предпочтительнее, чем локальные решения, по нескольким причинам:

1. Масштабируемость: облачные сервисы обычно предоставляют масштабируемость вычислительных ресурсов, что позволяет легко увеличивать или уменьшать мощность серверов в зависимости от текущих потребностей. Это позволяет пользователям обрабатывать большие объемы данных без необходимости покупки или аренды нового оборудования.

2. Доступность: пользователи могут обращаться к своим данным из любой точки мира, где есть доступ к Интернету, что позволяет эффективно работать с удаленными командами или коллегами. Кроме того, облачные сервисы могут обеспечить более высокий уровень доступности и надежности данных, чем локальные решения.

3. Гибкость: клиенты могут легко переключаться между различными инструментами и сервисами для обработки данных, такими как базы данных, системы хранения, инструменты анализа и т.д. Это может быть полезно для

тех, кто работает с различными типами данных или имеет различные потребности в обработке и анализе данных.

4. **Экономичность:** использование облачных сервисов может быть экономически более выгодным, чем приобретение и поддержка собственной инфраструктуры для обработки больших объемов данных. Данные сервисы могут предоставлять определенные скидки или пакетные предложения для клиентов, которые используют большое количество ресурсов.

5. **Безопасность:** уровень компетенций сотрудников облачного провайдера обычно выше, чем у сотрудников компаний-клиентов. Кроме того, провайдер использует оборудование и ПО промышленного уровня, что повышает безопасность и надёжность ИТ-систем.

Цель работы: используя программную реализацию полносвязной нейросети, решающей задачу распознавания цифр в датасете MNIST, провести сравнительный анализ облачных сервисов, предназначенных для работы с большими данными.

Задачи:

1. Составить требования к облачной инфраструктуре;
2. Выяснить, от чего зависит быстродействие программного кода;
3. Написание программы на языке Python, решающей задачу классификации рукописных цифр;
4. Исследование возможностей сервиса Yandex Datasphere;
5. Исследование возможностей сервиса VK Cloud ML Platform;
6. Сравнительный анализ работы сервисов.

ТРЕБОВАНИЯ К ОБЛАЧНОЙ ИНФРАСТРУКТУРЕ

Ранее было сказано, что облачные технологии позволяют эффективно работать с большим количеством информации.

Эффективная обработка данных – это способность компьютерной системы обрабатывать данные быстро, точно и с минимальным количеством ошибок и неисправностей.

Таким образом, будем сравнивать рассматриваемые облачные инфраструктуры между собой по следующим критериям:

1. Быстрая скорость работы: система должна быть способной быстро обрабатывать данные, чтобы обеспечить быстрый отклик и минимальное время простоя.

2. Экономическая эффективность: система должна быть способной обрабатывать данные с минимальными издержками.

3. Совместимость: система должна быть совместимой с другими программами и устройствами, что позволяет эффективно обмениваться данными и управлять всеми потоками данных в компании или организации.

4. Удобство использования: система должна быть простой в использовании и управлении, чтобы сократить затраты на обучение персонала и снизить вероятность ошибок при обработке данных.

Важный момент: последние три пункта по умолчанию предоставляются облачным провайдером, но на скорость работы может повлиять сам клиент.

БЫСТРОДЕЙСТВИЕ СИСТЕМЫ

В данном разделе разберемся, посредством чего пользователь может повлиять на скорость работы программного кода.

Скорость выполнения программы Python зависит от многих факторов, включая:

- **Размер входных данных:** скорость может зависеть от размера входных данных. Чем больше данных необходимо обработать, тем дольше будет выполняться программа.
- **Оптимизация кода** может существенно ускорить выполнение программы. Некоторые приемы оптимизации кода включают в себя использование нужных библиотек, более совершенных алгоритмов. Чем «чище» программный код, тем он быстрее.
- **Характеристики компьютера:** процессор и объем оперативной памяти напрямую влияют на быстроту обработки.

Процессор является одним из наиболее важных компонентов, влияющих на скорость выполнения программы Python. Чем мощнее процессор, тем быстрее может выполняться код.

Объем оперативной памяти также может существенно влиять на скорость выполнения программы Python. Если оперативной памяти недостаточно для хранения данных, используемых программой, компьютер может начать использовать виртуальную память, что приведет к замедлению выполнения программы.

Исследуем быстродействие на собственной и библиотечной (Keras) реализации программы распознавания рукописных цифр, используя разные вычислительные ресурсы, предоставляемые провайдерами.

Размер входных данных задан заранее и не изменяем.

ПРОГРАММНЫЙ КОД

Необходимо реализовать полносвязную нейронную сеть для распознавания рукописных цифр от 0 до 9 в выборке MNIST.

Датасет MNIST – это набор данных, который состоит из 70 000 изображений рукописных цифр от 0 до 9, размером 28 на 28 пикселей. Этот датасет часто используется для обучения и тестирования алгоритмов компьютерного зрения, таких как распознавание образов и классификация изображений.

Набор MNIST был создан в 1998 году и стал одним из самых популярных датасетов в машинном обучении. Он содержит 60 000 изображений для обучения и 10 000 изображений для тестирования [2].

Каждое изображение в выборке представлено в виде матрицы пикселей размером 28x28. Каждый пиксель может принимать значение от 0 до 255, где 0 – это белый цвет, а 255 - черный цвет. Изображения уже предобработаны, чтобы уменьшить шум и улучшить контраст.

Кроме того, каждое изображение сопровождается меткой (label), которая указывает, какая цифра представлена на изображении. Например, метка 0 означает, что на изображении изображена цифра 0, метка 1 - цифра 1, и так далее.

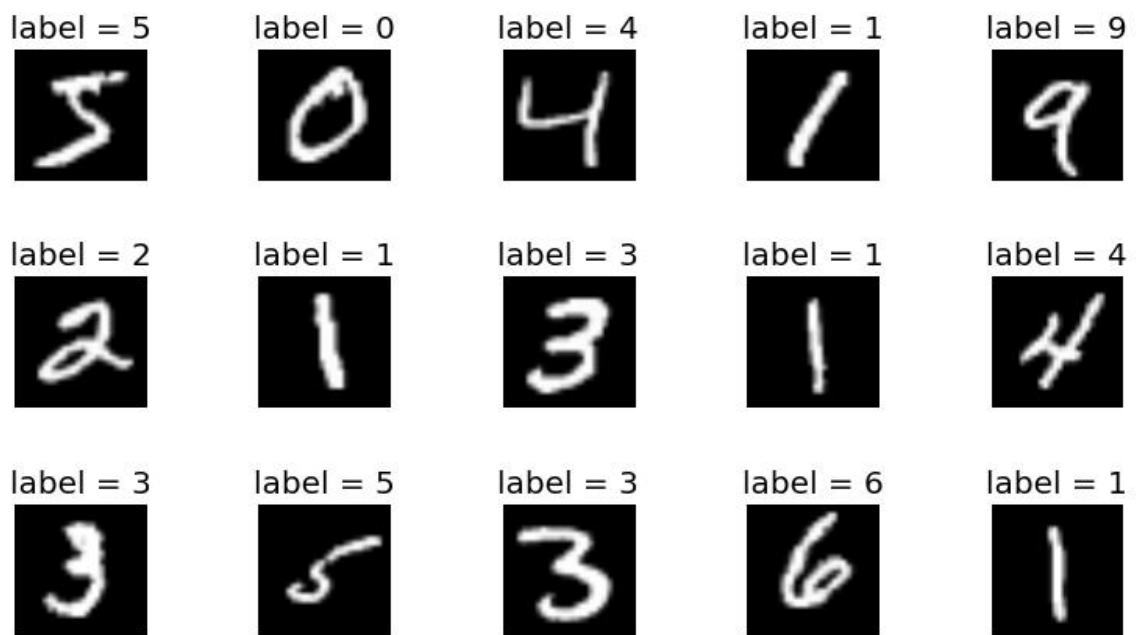


Рисунок 1 – Датасет MNIST [2]

Нейронная сеть состоит из 3 слоев: входного, скрытого и выходного. На вход подается вектор размерностью 785×1 (28 пикселей \times 28 пикселей + смещение). На скрытом слое 29 нейронов (с учётом смещения), а на выходном слое 10 нейронов, где каждый нейрон распознаёт одну определенную цифру (см. рисунок 2).

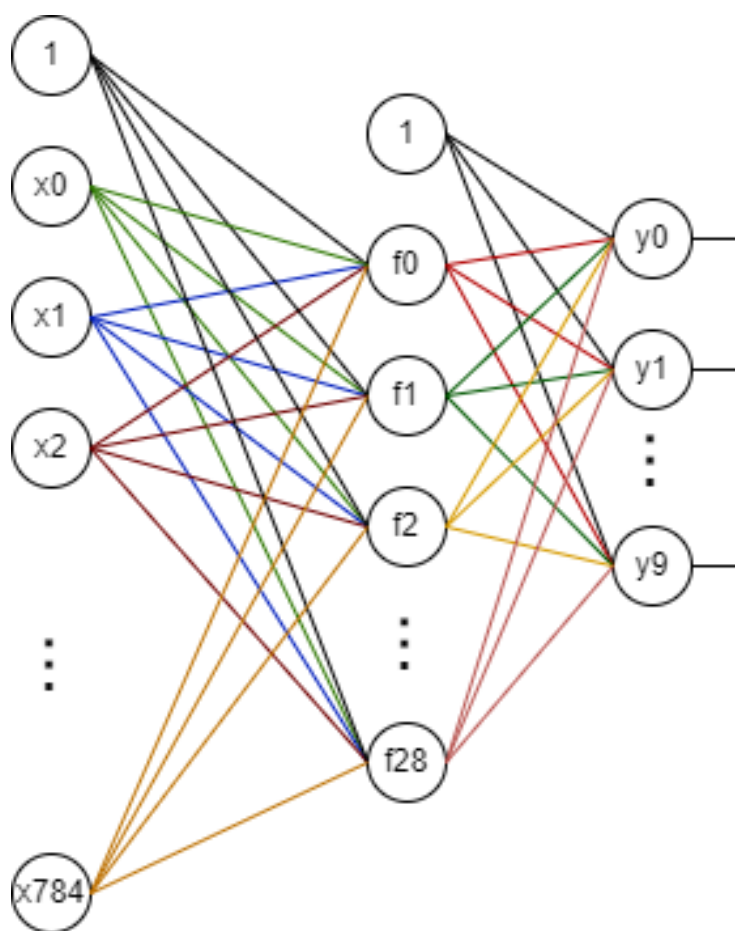


Рисунок 2 – Топология создаваемой нейросети

Для обучения нейросети используем алгоритм обратного распространения ошибки.

Листинг реализации нейросети приведен в Приложении А (авторская реализация без использования библиотеки Keras) и в Приложении Б (реализация с использованием библиотеки Keras).

АЛГОРИТМ ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ

Алгоритм обратного распространения ошибки определяет стратегию подбора весов многослойной сети с применением градиентных методов оптимизации. Поскольку целевая функция, обычно определяемая как сумма квадратичных разностей между фактическими и ожидаемыми выходными значениями, является непрерывной, градиентные методы оптимизации являются эффективными при обучении сети [1].

Основная идея алгоритма заключается в том, чтобы находить градиент (производную) функции потерь по параметрам нейронной сети и использовать его для обновления весов во всех слоях сети.

Процесс обратного распространения ошибки начинается с подачи входных данных на входной слой нейронной сети, после чего они проходят через все слои, пока не достигнут выходной слой, который выдаст предсказание модели. Затем вычисляется значение функции потерь, которая описывает, насколько сильно модель ошибается в предсказании.

Далее, происходит обратное распространение ошибки, при котором градиент функции потерь по каждому весу в сети вычисляется с помощью цепного правила дифференцирования. Этот градиент затем используется для обновления весов во всех слоях нейронной сети с помощью алгоритма градиентного спуска, который позволяет минимизировать значение функции потерь и улучшить качество предсказаний модели.

Квадратичная ошибка целевой функции для одного нейрона будет вычислена по формуле:

$$J(W, b, x, y) = \frac{1}{2} \|h_{W,b}(x) - y\|^2 \quad (1.1)$$

Задача — минимизировать $J(W, b)$. Для обучения нейронной сети необходимо инициализировать каждый параметр $W_{ij}^{(l)}$ и $b_i^{(l)}$ малыми случайными величинами, близкими к нулю, а затем применить алгоритм оптимизации (упоминавшийся выше градиентный спуск).

Так как $J(W, b)$ не является выпуклой функцией, то градиентный спуск восприимчив к локальным оптимумам. Каждая итерация градиентного спуска обновляет параметры таким образом:

$$\begin{aligned} W_{ij}^+ &= W_{ij} - \alpha \frac{\partial}{\partial W_{ij}} J(W, b) \\ b_i^+ &= b_i - \alpha \frac{\partial}{\partial b_i} J(W, b), \end{aligned} \tag{1.2}$$

где α – скорость обучения (learning rate).

ПРАКТИЧЕСКАЯ ЧАСТЬ

Существует три основных модели обслуживания в облачных вычислениях:

- IaaS (Infrastructure as a Service) – инфраструктура как услуга: IaaS-провайдер предоставляет виртуальную инфраструктуру, такую как серверы, сетевые устройства, хранилище данных и другие ресурсы, которые могут быть использованы для создания собственных приложений и услуг. Пользователи могут управлять своими виртуальными машинами и приложениями на основе предоставленных ресурсов.

- PaaS (Platform as a Service) – платформа как услуга: PaaS-провайдер предоставляет средства разработки, развертывания и управления приложениями, такие как программное обеспечение для разработки, базы данных, системы управления контентом и другие сервисы. Пользователи могут разрабатывать свои приложения на основе предоставленных средств и развертывать их на инфраструктуре провайдера.

- SaaS (Software as a Service) – программное обеспечение как услуга: SaaS-провайдер предоставляет полноценные приложения, такие как CRM (управление взаимоотношениями с клиентами), ERP (системы планирования ресурсов предприятия), веб-приложения и другие, которые доступны через Интернет. Пользователи могут использовать приложения без необходимости управлять инфраструктурой или платформой, на которой они работают.

Каждая из этих моделей имеет свои преимущества и недостатки, и выбор модели зависит от требований конкретного приложения или услуги. IaaS обычно используется при разработке собственных приложений или для работы с приложениями, которые требуют большого количества ресурсов, PaaS может быть полезен для разработки и развертывания приложений с использованием стандартных средств разработки, а SaaS предоставляет полноценное приложение, доступное через Интернет.

YANDEX DATASPHERE

Основной функционал

Yandex DataSphere — среда для ML-разработки полного цикла. С помощью данной среды пользователи могут использовать вычислительные ресурсы (CPU и GPU) для обработки больших объемов данных. Этот сервис позволяет работать с информацией различных форматов, включая структурированные и неструктурированные данные, и использовать различные инструменты для их анализа и обработки, такие как Apache Hadoop, Apache Spark, Apache Flink, и другие [3].

Одним из преимуществ является то, что пользователи могут сами выбрать настройки вычислительного кластера в зависимости от своих потребностей, чтобы оптимизировать производительность и стоимость использования сервиса. Также сервис обладает огромным количеством предустановленных библиотек и опцией сохранения состояния ноутбука, что делает его ещё более удобным.

Модель обслуживания Yandex Datasphere – это PaaS (Platform as a Service) т.е. платформа-сервис, предоставляющая готовую инфраструктуру и инструменты для разработки, тестирования и развертывания приложений без необходимости управления нижележащей инфраструктурой и операционными системами.

В качестве IDE DataSphere предоставляет Jupyter Notebook.

Сервис предоставляет функционал для совместной разработки через «Сообщества». Сообщества — способ организации групповой работы. Сообщества определяют область видимости проектов и ресурсов DataSphere.

Проекты — основное рабочее место пользователя в DataSphere. В проектах хранятся состояние интерпретатора, переменные, установленное ПО и прочая информация [3].

После создания платёжного аккаунта достаточно зайти в DataSphere и нажать большую синюю кнопку «Создать проект». Далее откроется интерфейс (см. рисунок 3, рисунок 4, рисунок 5):

cloud

ID ru-central1-a

Тимофей Ямкин
Timaaka47@yandex.ru

Открыть проект в JupyterLab

Создать ресурс

Еще ▾

Обзор Участники 1 Настройки

Запущенные операции

Остановить исполнения в IDE

Синхронные операции
Нет запущенных операцийФоновые операции
Нет запущенных фоновых операцийВ очереди
Нет операций в очередиИнформация об инстансах ноды
будет здесь, когда вы создадите ноду.[Подробнее](#)

Доступное хранилище проекта

9.48 ГБ

Занято 0.52 ГБ / 10 ГБ ?

Рисунок 3 – Рабочая область проекта

В Yandex Datasphere есть возможность запустить продолжительные операции (например, обучение моделей) в фоновом режиме, т.е. асинхронно (см. рисунок 3). При этом пользователь может продолжать работу с ноутбуком. Стоит учесть, что фоновые операции могут начать исполняться не сразу, а само исполнение может быть дольше, чем обычные операции. Однако фоновые операции тарифицируются в 2 раза дешевле обычных. Для запуска фоновой операции необходимо указать в ячейке комментарий *#pragma async*.

Изменить размер хранилища



Занято 0.53 ГБ из 10 ГБ

Размер хранилища
проекта

10 10 4096

Итоговая стоимость: Бесплатно ?

Отмена

Изменить размер

Рисунок 4 – Хранилище объекта

Каждый проект DataSphere имеет хранилище, в рамках которого хранение данных не тарифицируется. При увеличении квоты на размер проекта объем хранилища свыше 10 ГБ оплачивается отдельно. Максимальный размер хранилища 4ТБ.

Хранение данных внутри датасетов оплачивается отдельно.

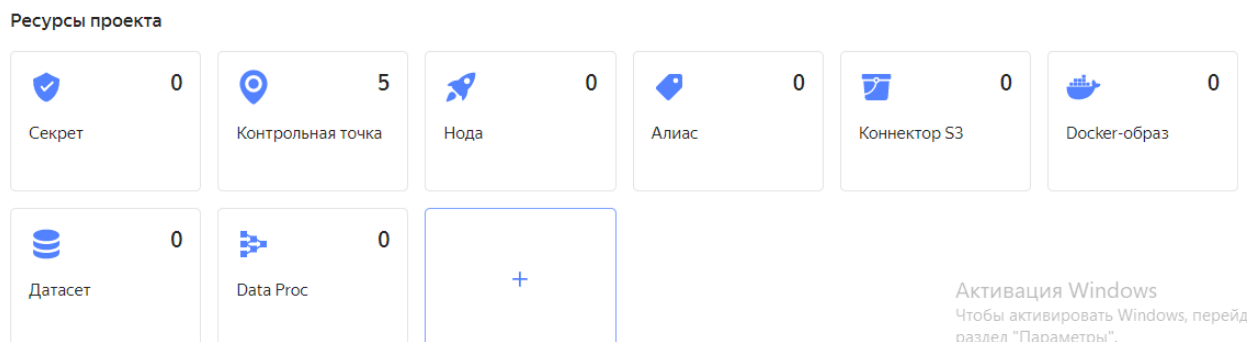


Рисунок 5 – Рабочая область проекта. Ресурсы

Ресурсы DataSphere — объекты, которые создаются или используются в проектах: датасеты, Docker-образы, ноды и другие.

В проектах DataSphere можно использовать следующие типы ресурсов [3]:

- **Датасет** — это механизм хранения информации, который предоставляет быстрый доступ к большим объемам данных. Датасеты позволяют хранить до 4 ТБ, при этом доступ к данным будет быстрее, чем к основному хранилищу проекта.

Создание и наполнение датасета происходит во время инициализации. После инициализации датасет нельзя изменить, он будет доступен только для чтения. Если вы хотите добавить файлы в датасет, создайте его заново.

- Чтобы безопасно хранить ключи, пароли и другую приватную информацию, DataSphere предоставляет специальный тип ресурсов — секреты. **Секрет** — это пара ключ-значение, в которой значение хранится в зашифрованном виде.

Созданные секреты можно использовать в коде ячеек как переменные окружения, чтобы безопасно подключаться к источникам данных и системам контроля версий, а также хранить в них ключи, которые необходимы для создания других ресурсов, например, подключений S3.

- **Docker-образы** — окружение операционной системы, в котором собран произвольный набор ПО, библиотек, переменных окружения и конфигурационных файлов.

Вы можете самостоятельно настроить окружение проекта DataSphere с помощью Docker-образа, собрав в него произвольный набор ПО, библиотек, переменных окружения и конфигурационных файлов. Docker-образ, примененный для проекта, будет использоваться при запуске кода во всех его ноутбуках.

- Вы можете управлять подключением к объектному хранилищу S3 в интерфейсе DataSphere на странице проекта с помощью ресурса **Коннектор S3**.

S3 или Simple Storage Service — сервис, где хранятся цифровые данные большого объема. Работает по одноименному протоколу.

- **Контрольные точки** — сохраненное состояние ноутбука: код ячеек, вывод и значения переменных, а также данные хранилища проекта. Контрольные точки версионизируются.

- **Ноды** — сервисы, развернутые для эксплуатации обученных моделей. Сторонние сервисы могут обращаться к нодам по API.

- **Алиасы** — «надстройка» для публикации сервисов. Алиасы позволяют распределять нагрузку между нодами и обновлять развернутые сервисы во время работы.

- **Data Proc** — это управляемый сервис для обработки больших данных на основе Apache Hadoop и Apache Spark. Data Proc позволяет пользователям создавать кластеры на основе этих технологий с минимальными затратами и начинать работу с обработкой больших объемов

данных без необходимости устанавливать и настраивать соответствующее программное обеспечение на собственном оборудовании.

После создания проекта жмем кнопку «Открыть проект в Jupyter Lab» для того, чтобы перейти в ноутбук.

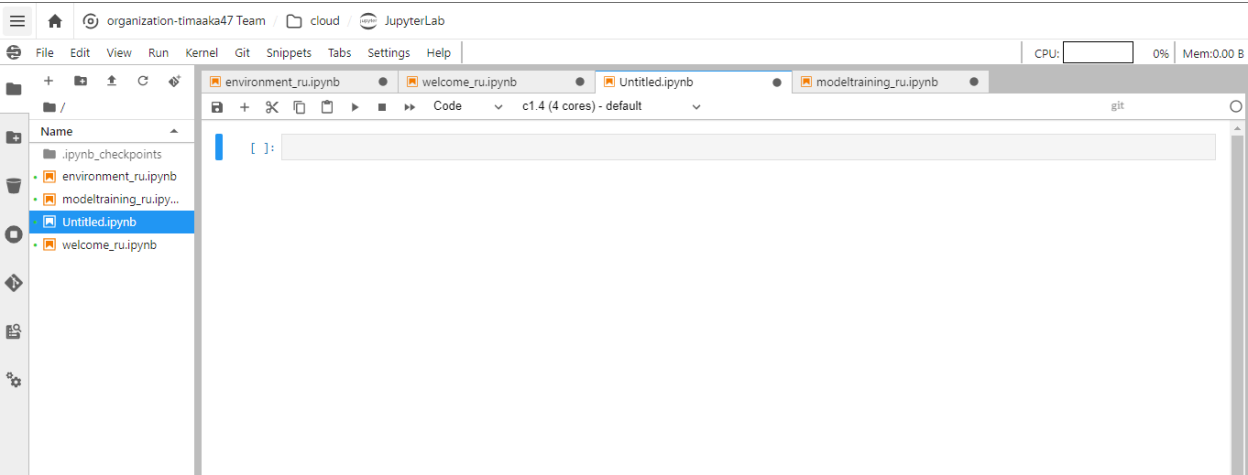


Рисунок 6 – Ноутбук

Важная деталь: каждая ячейка может быть исполнена на любой аппаратной конфигурации из предлагаемых.

Имя	Количество vCPU	Количество GPU	RAM, ГБ
Конфигурации с vCPU на платформе Intel Ice Lake			
c1.4 (по умолчанию)	4	0	32
c1.8	8	0	64
c1.32 (доступна по запросу в техническую поддержку)	32	0	256
c1.80 (доступна для юридических лиц по запросу в техническую поддержку)	80	0	640
Конфигурации с vCPU на платформе Intel Broadwell и GPU NVIDIA® Tesla® V100			
g1.1 (доступна по запросу в техническую поддержку или при остатке на счете не менее 500 ₽)	8	1	от 48 до 96
g1.2 (доступна по запросу в техническую поддержку)	16	2	от 96 до 192
g1.4 (доступна по запросу в техническую поддержку)	32	4	от 192 до 384
Конфигурации с vCPU на платформе AMD EPYC™ и GPU NVIDIA® Ampere® A100			
g2.mig	4	1/8	16
g2.1 (доступна по запросу в техническую поддержку)	28	1	119
g2.2 (доступна по запросу в техническую поддержку)	56	2	238
g2.4 (доступна по запросу в техническую поддержку)	112	4	476
g2.8 (доступна по запросу в техническую поддержку)	224	8	952

Рисунок 7 – Конфигурация вычислительных ресурсов

Тестирование быстродействия

Обучим две реализации полносвязной нейросети (код приведен в приложении) и замерим время их обучения на различных аппаратных конфигурациях.

Результаты приведены ниже.

```
#!/g2.mig
%%time
a.learning(X_train,y_train,learning_rate = 0.1,epochs = 1)

CPU times: user 2min 59s, sys: 5min 20s, total: 8min 20s
Wall time: 2min 5s
'Обучение выполнено!'
```

Рисунок 8 – Авторская реализация нейросети на #!g2.mig конфигурации

```
a = Network()

#!/c1.32
%%time
a.learning(X_train,y_train,learning_rate = 0.1,epochs = 1)

CPU times: user 4min 30s, sys: 6min 38s, total: 11min 8s
Wall time: 1min 46s
'Обучение выполнено!'
```

Рисунок 10 – Авторская реализация нейросети на #!c1.32 конфигурации

```
[25]: a.network_accuracy(X_train,y_train)

[25]: 94.055

[26]: a.network_accuracy(X_test,y_test)

[26]: 93.82000000000001
```

Рисунок 12 – Точность авторской реализации нейросети (количество эпох – 2)

```
[8]: #!g2.mig
%%time
batch_size = 1 # количество подаваемых картинок за раз
epochs = 1

model.compile(loss="MSE", optimizer=keras.optimizers.SGD(learning_rate=0.1), metrics=["accuracy"])
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs) # обучаем НС

60000/60000 [=====] - 68s 1ms/step - loss: 0.0321 - accuracy: 0.8262
CPU times: user 1min 31s, sys: 11.1 s, total: 1min 42s
Wall time: 1min 33s
[8]: <keras.callbacks.History at 0x7f19c01c1700>
```

Рисунок 9 – Реализация нейросети на Keras на #!g2.mig конфигурации

```
[11]: #!c1.32
%%time
batch_size = 1 # количество подаваемых картинок за раз
epochs = 1

model.compile(loss="MSE", optimizer=keras.optimizers.SGD(learning_rate=0.1), metrics=["accuracy"])
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs) # обучаем НС

60000/60000 [=====] - 28s 470us/step - loss: 0.0319 - accuracy: 0.8277
CPU times: user 39 s, sys: 10 s, total: 49.1 s
Wall time: 46 s
[11]: <keras.callbacks.History at 0x7f45470e640>
```

Рисунок 11 – Реализация нейросети на Keras на #!c1.32 конфигурации

```
[9]: score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])

Test loss: 0.01670234650373459
Test accuracy: 0.9126999974250793
```

Рисунок 13 – Точность реализации нейросети на Keras (количество эпох – 1)

Таблица 1 – Быстродействие программного кода на разных аппаратных конфигурациях

	Авторская реализация нейросети	Реализация с помощью Keras
Время (конфигурация #!g2.mig)	2 минуты 5 секунд	1 минута 33 секунды
Время (конфигурация #!c1.32)	1 минута 46 секунд	46 секунд
Точность на тестовой выборке, %	93,82 (количество эпох обучения – 2)	91,27 (количество эпох обучения – 2)

Таблица 2 – Характеристики аппаратных конфигураций

	Количество CPU	Количество GPU	RAM, Гб
g2.mig	4	1/8	16
c1.32	32	0	256

В таблице 1 и 2 можно увидеть, как ускоряется исполнение программы при применении различных вычислительных конфигураций.

Тарификация

При работе с сервисом DataSphere вы платите за фактическое использование вычислительных ресурсов — посекундно тарифицируется время вычисления [3]. То есть если ячейка закончила свои вычисления и CPU/GPU больше не загружен, то вы ничего не платите. Вычисление общей стоимости происходит с помощью юнитов. Чем мощнее конфигурация, тем больше юнитов в секунду будет назначено, стоимость одного юнита 0.0012 рублей/секунда.

Более подробная тарификация приведена ниже.

Конфигурация	Количество юнитов в конфигурации	Цена за 1 секунду вычислений, вкл. НДС
c1.4 (4 vCPU, 0 GPU)	4	0,0048 Р
c1.8 (8 vCPU, 0 GPU)	8	0,0096 Р
c1.32 (32 vCPU, 0 GPU)	32	0,0384 Р
c1.80 (80 vCPU, 0 GPU)	80	0,0960 Р
g1.1 (8 vCPU, 1 GPU V100)	72	0,0864 Р
g1.2 (16 vCPU, 2 GPU V100)	144	0,1728 Р
g1.4 (32 vCPU, 4 GPU V100)	288	0,3456 Р
g2.mig (4 vCPU, 1/8 GPU A100)	18	0,0216 Р
g2.1 (28 vCPU, 1 GPU A100)	116	0,1392 Р
g2.2 (56 vCPU, 2 GPU A100)	232	0,2784 Р
g2.4 (112 vCPU, 4 GPU A100)	464	0,5568 Р
g2.8 (224 vCPU, 8 GPU A100)	928	1,1136 Р

Рисунок 14 – Тарификация вычислительных ресурсов

Ресурс	Цена за 1 ГБ в месяц, вкл. НДС
Объем хранилища проекта, до 10 ГБ	Не тарифицируется
Объем хранилища проекта, сверх 10 ГБ	11,9100 Р

Рисунок 16 – Тарификация хранилища

Конфигурация	Количество юнитов в конфигурации	Цена за 1 секунду вычислений, вкл. НДС
c1.4 (4 vCPU, 0 GPU)	2	0,0024 Р
c1.8 (8 vCPU, 0 GPU)	4	0,0048 Р
c1.32 (32 vCPU, 0 GPU)	16	0,0192 Р
c1.80 (80 vCPU, 0 GPU)	40	0,0480 Р
g1.1 (8 vCPU, 1 GPU V100)	36	0,0432 Р
g1.2 (16 vCPU, 2 GPU V100)	72	0,0864 Р
g1.4 (32 vCPU, 4 GPU V100)	144	0,1728 Р
g2.mig (4 vCPU, 1/8 GPU A100)	9	0,0108 Р
g2.1 (28 vCPU, 1 GPU A100)	58	0,0696 Р
g2.2 (56 vCPU, 2 GPU A100)	116	0,1392 Р
g2.4 (112 vCPU, 4 GPU A100)	232	0,2784 Р
g2.8 (224 vCPU, 8 GPU A100)	464	0,5568 Р

Рисунок 15 – Тарификация фоновых операций

Тип хранилища	Цена за 1 ГБ в месяц, вкл. НДС
Датасет	11,91 Р

Рисунок 17 – Тарификация датасета

При этом, при создании платежного аккаунта, пользователю доступен грант на 4000 рублей и 2 месяца для тестирования возможностей платформы.

Особенности платформы

Таблица 3 – Особенности платформы Yandex Datasphere

Ценовая политика	Оплачивается только фактическое время вычислений. Единый платеж (не нужно дополнительно подключать и оплачивать GPU).
Аппаратная конфигурация	Можно максимально оптимизировать вычисления, подобрав для каждой ячейки с кодом нужную вычислительную конфигурацию
Время жизни окружения	Бесконечно

Версионирование	Код, переменные и диск версионироваться средствами платформы
Смена CPU/GPU	Данные сохраняются
Окружение	Заранее предустановлены все необходимые библиотеки для анализа данных
Дополнительно	1. Доступен фоновый режим вычислений 2. Исчерпывающая инструкция по работе с сервисом

VK CLOUD ML PLATFORM

Основной функционал

VK Cloud ML Platform – это сервис облачных вычислений, предоставляемый компанией VK, который позволяет пользователям создавать, обучать и развертывать модели машинного обучения в облачной среде. Этот сервис обеспечивает пользователям инструменты для работы с большими объемами данных и выполнения сложных вычислительных задач, таких как анализ данных, обучение нейронных сетей и машинное обучение [4].

С помощью VK Cloud ML Platform пользователи могут разрабатывать и запускать свои модели машинного обучения, не заботясь о необходимости настройки инфраструктуры и управления ресурсами. Пользователи могут использовать готовые библиотеки машинного обучения, такие как TensorFlow и PyTorch, и настраивать свои собственные модели, используя языки программирования Python, R и другие. Все вычисления выполняются в облачной среде VK, что обеспечивает масштабируемость, гибкость и высокую доступность вычислительных ресурсов.

По модели обслуживания, VK Cloud ML Platform можно отнести к категории PaaS (Platform as a Service), т.е. платформа-сервис, предоставляющая готовую инфраструктуру и инструменты для разработки,

тестирования и развертывания приложений без необходимости управления нижележащей инфраструктурой и операционными системами.

Платформа состоит из нескольких компонент и их комбинаций [4]:

1. Jupyter Hub – инструмент для работы с Jupyter Notebook в многопользовательском режиме. Позволяет быстро создать Jupyter Notebook для проведения экспериментов.

2. MLflow – инструмент для решения задач трекинга и версионирования экспериментов, ML моделей.

3. MLflow + Jupyter Hub – готовое к работе окружение с JupyterHub, предназначенное для управления жизненным циклом ML-моделей.

4. MLflow Deploy – удобная среда для упаковки ML-моделей и их автоматического развертывания в облаке. Компонент интегрирован с MLflow и JupyterHub.

Создадим инстанс Jupyter Hub. Переходим в раздел ML Platform и нажимаем кнопку «Создать инстанс».

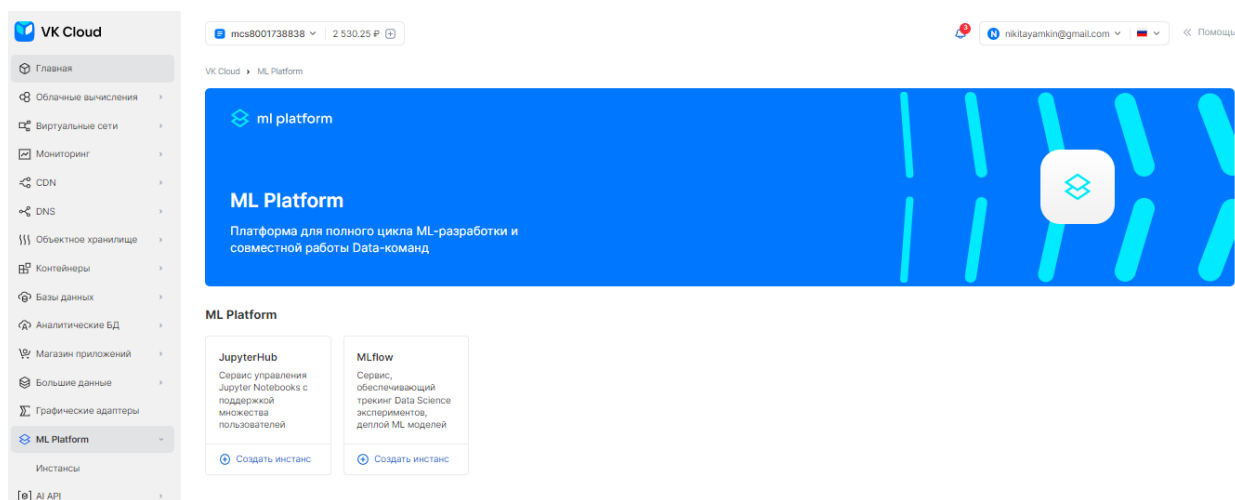


Рисунок 18 – Главная страница раздела ML Platform

Попадем на страницу для создания нового инстанса, где предоставляется возможность задать его вычислительные характеристики, а также настройки доступа к нему (логин и пароль).

В области справа отображается тарификация для выбранной конфигурации инстанса.

Создание нового инстанса

✓ Тип инстанса

2 Конфигурация

3 Выбор сети

4 Завершение

Имя инстанса

mlp-5046

Тип виртуальной машины

Standard-4-8

4 CPU

8 ГБ RAM

Зона доступности

Москва (GZ1)

Настройки диска

Размер диска

-

100

ГБ

+

Тип диска ⓘ

SSD

High-IOPS SSD

Доменное имя

Выбор доменного имени

ml-platform-3ba4995a27f57a

Пароль доступа

Имя пользователя

yamkin_nikita

Пароль пользователя

A123456a

Сгенерировать

ⓘ

Сохраните пароль. Возможность восстановления пароля не предусмотрена. Допустимо использовать только разрешенные символы.

Следующий шаг

Предыдущий шаг

Рисунок 19 – Первая страница создания инстанса

Создание нового инстанса

✓ Тип инстанса > ✓ Конфигурация > **3 Выбор сети** > 4 Завершение

Сеть

Создать новую сеть

Адрес подсети

10.0.0.0/24

Например, 10.0.0.0/24

Ключ виртуальной машины

id_rsa

Создать инстанс

Предыдущий шаг

Рисунок 20 – Вторая страница создания инстанса

Ежемесячная плата

7 202 Р ⓘ

Поминутная плата

0.19 Р

Прайс лист

[Посмотреть цены](#)

Рисунок 21 – Автоматически рассчитанная тарификация

После того, как произвели конфигурацию, нажимаем «Создать инстанс» и ждем некоторое время. Как только инстанс будет готов к использованию, перед вами появится страница с его общими данными.

Общие данные


Параметры инстанса	Значение
Имя	mip-5046
ID	db611001-822b-4ba8-b7f7-427adc6687b9
Тип инстанса	 JupyterHub
Статус	● Инстанс активен
Виртуальные CPU, шт.	4
Объем оперативной памяти, МБ	8192
Объем всех дисков, ГБ	100
Зона доступности	Москва (GZ1)
DNS-имя	https://mi-platform-3ba4995a27f57a.ml.msk.vkcs.cloud
IP-адрес	95.163.208.43
Дата создания	14 марта 2023 20:27


Рисунок 22 – Общие данные инстанса

VK Cloud > ML Platform > Инстансы

Инстансы

 Скрыть квоты

Инстансы 1 из 10 шт	CPU 4 из 32 шт	RAM 8 из 52 ГБ	Объем 100 из 400 ГБ	Диски 1 из 32 шт	IP 1 из 7 шт
------------------------	-------------------	-------------------	------------------------	---------------------	-----------------

 Если вы хотите увеличить квоты, напишите в [техническую поддержку](#)

X

☐ [+ Добавить](#)

Инстансы	Тип	IP-адрес	DNS-имя	
<input type="checkbox"/> ● mip-5046	 JupyterHub	95.163.208.43	https://mi-platform-3ba4995a27f57a.ml.msk.vkcs.cloud	Standard-4-8-80 ...

Рисунок 23 – Созданный инстанс в личном кабинете

Для того, чтобы получить доступ к IDE, необходимо перейти по ссылке, указанной в поле «DNS-имя» на странице с общими данными, и ввести свои учетные данные, которые были указаны в процессе создания.

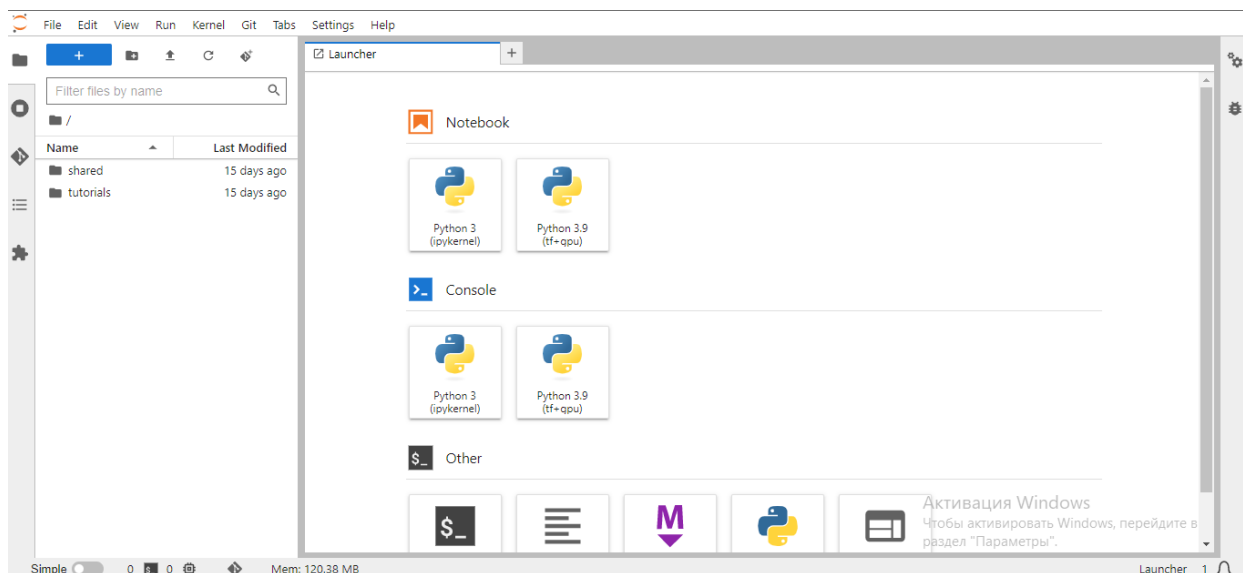


Рисунок 24 – Домашняя страница ноутбука

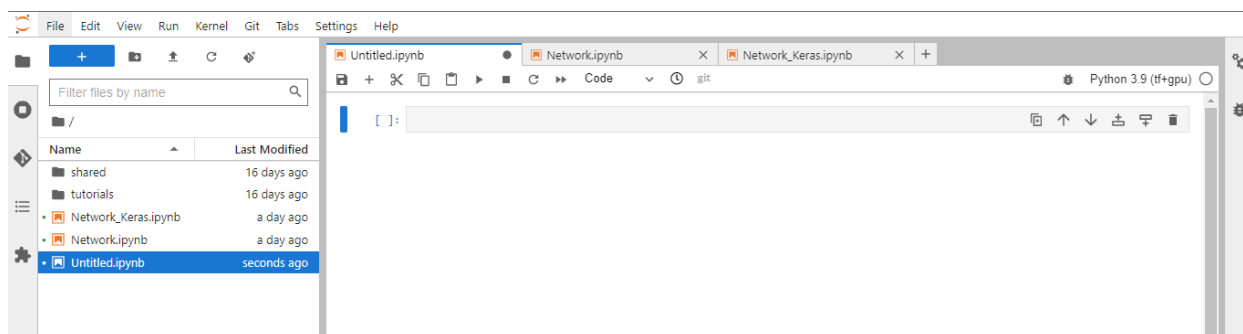


Рисунок 25 – Ноутбук

Пользователю будут доступны два ядра на выбор: Python 3 (ipykernel) и Python 3.9 (tf + gpu). Отличие между ними в том, что во втором варианте предустановлена библиотека Tensorflow и предоставляется возможность использования GPU (оплачивается отдельно).

Важной особенностью данного ноутбука является возможность отладки кода в ячейке с помощью дебаггера.

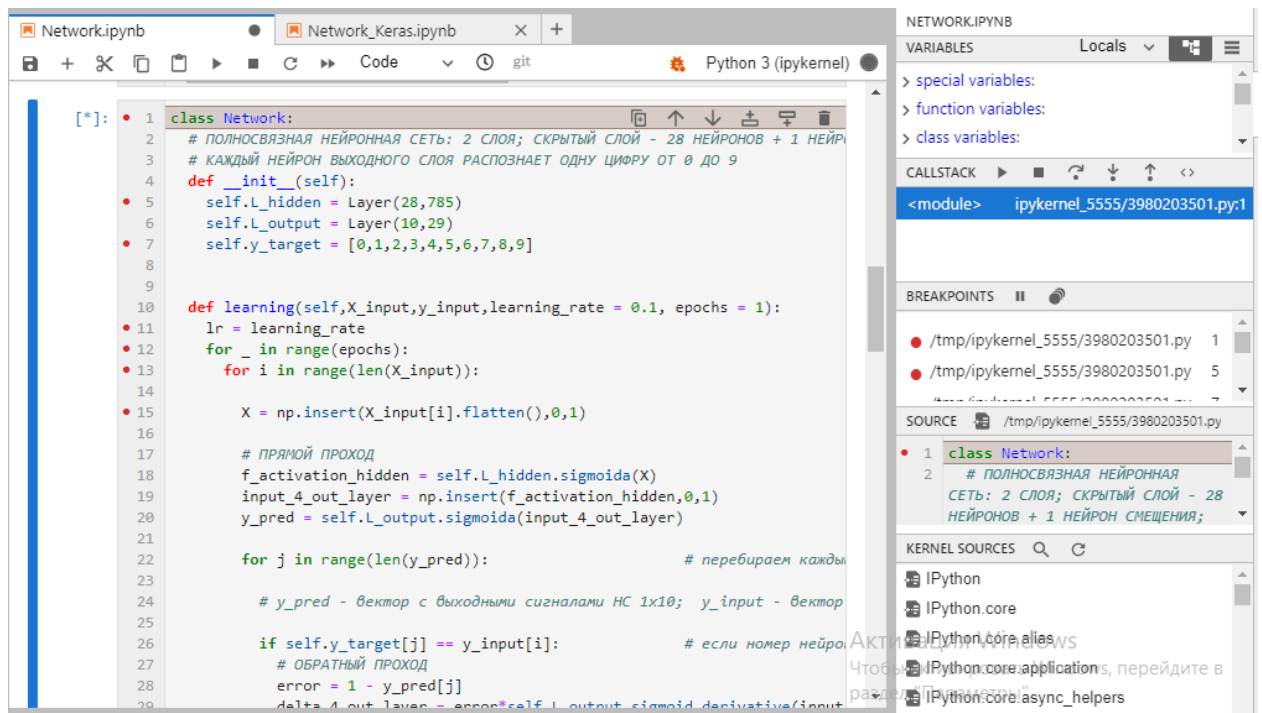


Рисунок 26 – Отладка ячейки кода с помощью встроенного дебаггера

Тестирование быстродействия

Обучим две реализации полносвязной нейросети (код приведен в приложении) и замерим время их обучения на различных аппаратных конфигурациях.

Результаты приведены ниже.

```
[11]: %%time
a.learning(X_train,y_train,learning_rate = 0.1,epochs = 1)

CPU times: user 4min 45s, sys: 3min 8s, total: 7min 54s
Wall time: 1min 59s

[11]: 'Обучение выполнено!'
```

Рисунок 27 – Авторская реализация нейросети на ядре Python 3.9 (tf + gpu)

```
[8]: %time
batch_size = 1 # количество подаваемых картинок за раз
epochs = 1

model.compile(loss="MSE", optimizer=keras.optimizers.SGD(learning_rate=0.1), metrics=["accuracy"])
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs) # обучаем НС

60000/60000 [*****] - 68s 1ms/step - loss: 0.0314 - accuracy: 0.8325
CPU times: user 1min 21s, sys: 9.13 s, total: 1min 30s
Wall time: 1min 8s

[8]: <keras.callbacks.History at 0x7f41a75920a0>
```

Рисунок 28 – Реализация нейросети на Keras на ядре Python 3.9 (tf + gpu)

Таблица 4 – Быстродействие программного кода на разных аппаратных конфигурациях

	Авторская реализация нейросети	Реализация с помощью Keras
Время (ядро Python 3.9 (tf + gpu))	1 минута 59 секунд	1 минута 8 секунд
Точность на тестовой выборке, %	93,34 (количество эпох обучения – 2)	91,56 (количество эпох обучения – 2)

Таблица 5 – Характеристика инстанса

	Количество CPU	Количество GPU	RAM, Гб
ядро Python 3.9 (tf + gpu)	4	0	8

В таблице 4 и 5 можно увидеть, как ускоряется исполнение программы на более оптимальной программной реализации нейросети с помощью библиотеки Keras.

Тарификация

Оплата сервиса Cloud ML Platform доступна по модели pay-as-you-go – только за использованные ресурсы. Воспользуйтесь калькулятором для расчета стоимости [4].

Рассчитать стоимость

Выберите модуль

JupyterHub MLflow MLflow Production

CPU, шт. ☯

- 4 +

RAM, Гб

- 4 +

Количество инстансов, шт.

- 1 +

Объем диска, Гб

- 100 +

Тип диска ☯

SSD High-IOPS

Итоговый расчет ☯

Кол-во сервисов в данной конфигурации

- 1 +

Стоимость в месяц

6 242 ₽

Стоимость в день

208,07 ₽

Тестировать

Рисунок 27 – Пример расчета стоимости калькулятором для заданной конфигурации инстанса

X

Параметр	Значение	Цена за день (в том числе НДС)	Цена за месяц (в том числе НДС)
CPU, шт.	4 шт.	122,4 Р	3 672 Р
RAM	4 Гб	32 Р	960 Р
SSD	100 Гб	48 Р	1 440 Р
Количество инстансов в этой конфигурации:	1	-	-
Публичный IP	Да	5,67 Р	170 Р
Итог за 1 сервис(ов)		208,07 Р	6 242 Р

Рисунок 28 – Детализация расчета

Отдельно можно подключить графический процессор. Тарификация его использования приведена ниже.

Стоимость

Вы платите только за те ресурсы, которые используете.

Годовой план

Стоимость в час за виртуальную машину
с картой NVIDIA Tesla A100

175 Р

Период оплаты составляет
1 месяц

Минимальное использование
1 год

Рассчитать конфигурацию

Месячный план

Стоимость в час за виртуальную машину
с картой NVIDIA Tesla A100

194,4 Р

Период оплаты и минимальное использование составляют
1 месяц

Рассчитать конфигурацию

Часовой план

Стоимость в час за виртуальную машину
с картой NVIDIA Tesla A100

204,2 Р

Период оплаты и минимальное использование составляют
1 час

Рассчитать конфигурацию

Рисунок 29 – Тарификация GPU

При этом, при создании платежного аккаунта, пользователю доступен грант на 3000 рублей для тестирования возможностей платформы.

Особенности платформы

Таблица 6 – Особенности платформы VK ML Platform

Ценовая политика	Оплата только за использованные ресурсы. Нужно дополнительно подключать и оплачивать GPU.
Аппаратная конфигурация	Создается инстанс с заданной конфигурацией. Для того, чтобы изменить конфигурацию нужно создать новый инстанс. GPU подключается и оплачивается отдельно.
Время жизни окружения	12 часов
Версионирование	Код, переменные и диск версионируются компонентой MLflow, которую нужно оплачивать отдельно
Смена CPU/GPU	Данные не сохраняются
Окружение	Заранее предустановлены только некоторые библиотеки. Остальное нужно устанавливать самостоятельно.
Дополнительно	<ol style="list-style-type: none">1. Доступна отладка ячейки кода с помощью дебаггера2. Короткая и непонятная инструкция по работе с сервисом

СРАВНИТЕЛЬНЫЙ АНАЛИЗ

Таблица 7 – Сравнительные характеристики рассматриваемых платформ

	Yandex Datasphere	VK Cloud ML Platform	Google Colab
Модель обслуживания	PaaS	PaaS	SaaS
Ценовая политика	Оплачивается только фактическое время вычислений. Единый платеж (не нужно дополнительно подключать и оплачивать GPU).	Оплата только за использованные ресурсы. Нужно дополнительно подключать и оплачивать GPU.	Бесплатно (в том числе и GPU) или по подписке
Аппаратная конфигурация	Можно максимально оптимизировать вычисления, подобрав для каждой ячейки с кодом нужную вычислительную конфигурацию	Создается инстанс с заданной конфигурацией. Для того, чтобы изменить конфигурацию нужно создать новый инстанс. GPU подключается и оплачивается отдельно.	Бесплатно: 2vCPU Intel Xeon 2.2GHz GPU Nvidia Tesla K80
Время жизни окружения	Бесконечно	Бесконечно	12 часов
Версионирование	Код, переменные и диск версионизируются средствами платформы	Код, переменные и диск версионизируются компонентой MLflow, которую нужно оплачивать отдельно	Результаты нужно сохранять на диск вручную
Смена CPU/GPU	Данные сохраняются	Данные не сохраняются	Сессия заканчивается, теряются данные
Наличие библиотек	Заранее предустановлены все необходимые библиотеки для анализа данных	Заранее предустановлены только некоторые библиотеки. Остальное нужно устанавливать самостоятельно.	Нужно самостоятельно устанавливать все необходимые библиотеки. Импортированные в проект библиотеки отключаются в конце сессии
Безопасность	Логин + пароль + Секреты	Логин + пароль	Логин + пароль
Дополнительно	<ul style="list-style-type: none"> Доступен фоновый режим вычислений. Исчерпывающая инструкция по работе с сервисом 	<ul style="list-style-type: none"> Доступна отладка ячейки кода с помощью дебаггера. Короткая и непонятная инструкция по работе с сервисом 	—

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были исследованы возможности Yandex Datasphere и VK Cloud ML Platform – популярных отечественных сервисов для работы с большими данными. Результаты исследования занесены в Таблицу №7, в которой также были описаны характеристики другой популярной платформы Google Colaboratory.

Выбор того или иного облачного провайдера зависит от решаемой задачи.

Для образовательных целей Google Colaboratory – самый приемлемый вариант, т.к. данная платформа предоставляет бесплатный доступ к мощным вычислительным ресурсам и удобна в использовании.

Недостатком Google Colaboratory является ограниченность вычислительных мощностей и хранилища данных, что делает практически невозможным запуск больших проектов в этой среде.

Для решения бизнес-задач, в соответствии с описанными ранее требованиями к облачной инфраструктуре, Yandex Datasphere – лучший вариант из рассмотренных. Разработка от Яндекса более гибкая с точки зрения использования аппаратных конфигураций и более дешёвая в использовании (оплата только за время выполнения), а сами условия тарификации просты и понятны. В то же время Yandex Datasphere имеет отличительные особенности (такие как: фоновый режим работы, сохранение состояния ноутбука и Секреты), которые значительно ускоряют и упрощают ML-разработку.

СПИСОК ИСТОЧНИКОВ

1. Градиент // Википедия. Свободная энциклопедия URL: <https://ru.wikipedia.org/wiki/Градиент> (дата обращения: 10.03.2023).
2. MNIST dataset // SkillFactory.Блог. URL: <https://blog.skillfactory.ru/glossary/mnist-dataset/> (дата обращения 11.03.2023)
3. Yandex Datasphere. Пошаговые инструкции // Yandex Cloud. URL: <https://cloud.yandex.ru/docs/datasphere/> (дата обращения: 13.03.2023).
4. ML Platform // VK Cloud. URL: <https://mcs.mail.ru/docs/ml/mlplatform> (дата обращения: 14.03.2023).

ПРИЛОЖЕНИЕ А

Авторская реализация полносвязной нейронной сети

```
import numpy as np
from random import random, randint
from keras.datasets import mnist
from matplotlib import pyplot as plt
import math
import pandas as pd

from tensorflow import keras
from tensorflow.keras import layers

from PIL import Image

(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = np.array(X_train, dtype = 'float64')
y_train = y_train

X_test = np.array(X_test, dtype = 'float64')
y_test = y_test

#### Нормализация данных. Приведем значение пикселей в диапазон от 0 до 1

X_train = X_train / 255
X_test = X_test / 255
```

```

class Layer:
    def __init__(self, n, m):
        # Каждый вход нейрона - пиксель картинки
        # Всего входов у нейрона 28x28 + 1 = 785
        # Всего входов 785; от каждого входа 28 весовых коэффициентов - на с
крытый слой
        # Всего 28 нейронов на скрытом слое, от каждого нейрона 10 весовых коэ
эффициентов на выходной слой

        self.W = np.random.rand(n, m) / 1000

# считаем взвешенную сумму
def calculate_sum(self, X_input):
    return np.dot(X_input, self.W.T)

# считаем функцию активации
def sigmoida(self, X_input):
    sum = self.calculate_sum(X_input)          # вектор 1x28
    sum_list = []
    for i in range(len(sum)):
        sum_i = 1 / (1 + math.exp(-sum[i]))
        sum_list.append(sum_i)
    return np.array(sum_list)

# считаем производную функции активации
def sigmoid_derivative(self, X_input):
    sigmoida = self.sigmoida(X_input)          # вектор 1x28
    sigmoid_derivative_list = []
    for i in range(len(sigmoida)):
        sigmoid_derivative_i = sigmoida[i] * (1 - sigmoida[i])
        sigmoid_derivative_list.append(sigmoid_derivative_i)
    return np.array(sigmoid_derivative_list)

```

```

class Network:
# ПОЛНОСВЯЗНАЯ НЕЙРОННАЯ СЕТЬ: 2 СЛОЯ; СКРЫТЫЙ СЛОЙ - 28 НЕЙРОНОВ + 1 НЕЙРОН СМЕЩЕНИЯ; ВЫХОДНОЙ СЛОЙ - 10 НЕЙРОНОВ
# КАЖДЫЙ НЕЙРОН ВЫХОДНОГО СЛОЯ РАСПОЗНАЕТ ОДНУ ЦИФРУ ОТ 0 ДО 9
    def __init__(self):
        self.L_hidden = Layer(28,785)          # 28x28 +1
        self.L_output = Layer(10,29)
        self.y_target = [0,1,2,3,4,5,6,7,8,9]
    def learning(self,X_input,y_input,learning_rate = 0.1, epochs = 1):
        lr = learning_rate
        for _ in range(epochs):
            for i in range(len(X_input)):
                X = np.insert(X_input[i].flatten(),0,1)
                # ПРЯМОЙ ПРОХОД
                f_activation_hidden = self.L_hidden.sigmoida(X)
                input_4_out_layer = np.insert(f_activation_hidden,0,1)
                y_pred = self.L_output.sigmoida(input_4_out_layer)
                for j in range(len(y_pred)):
                    if self.y_target[j] == y_input[i]:
                        # ОБРАТНЫЙ ПРОХОД
                        error = 1 - y_pred[j]
                        delta_4_out_layer = error*self.L_output.sigmoid_derivative(input_4_out_layer)[j]
                        sigmoid_derivative_for_L_hid_with_bias = np.insert(self.L_hidden.sigmoid_derivative(X),0,0.25)

                        delta_4_hid_layer = delta_4_out_layer*self.L_output.W[j]*sigmoid_derivative_for_L_hid_with_bias
                        delta_4_hid_layer_without_bias = delta_4_hid_layer[1:]
                        # МЕНЯЕМ ВЕСА ДЛЯ ОДНОГО НЕЙРОНА
                        self.L_output.W[j] += lr*delta_4_out_layer*input_4_out_layer
                        # МЕНЯЕМ ВСЕ ВЕСА СКРЫТОГО СЛОЯ 28x785;
                        self.L_hidden.W += lr*np.dot(delta_4_hid_layer_without_bias.reshape(28,1),X.reshape(1,785))

                    if self.y_target[j] != y_input[i]:
                        # ОБРАТНЫЙ ПРОХОД
                        error = 0 - y_pred[j]
                        delta_4_out_layer = error*self.L_output.sigmoid_derivative(input_4_out_layer)[j]
                        sigmoid_derivative_for_L_hid_with_bias = np.insert(self.L_hidden.sigmoid_derivative(X),0,0.25)

```

```

        delta_4_hid_layer = delta_4_out_layer*self.L_output.W[j]*sigmoid_derivative_for_L_hid_with_bias
        delta_4_hid_layer_without_bias = delta_4_hid_layer[1:]
        self.L_output.W[j] += lr*delta_4_out_layer*input_4_out_layer
        self.L_hidden.W += lr*np.dot(delta_4_hid_layer_without_bias.reshape(28,1),X.reshape(1,785))
    return ('Обучение выполнено!')

def network_output(self,X_input):
# подаем весь набор картинок
    y_pred_list = []
    for i in range(len(X_input)):
        X = np.insert(X_input[i].flatten(),0,1)
        # ПРЯМОЙ ПРОХОД
        f_activation_hidden = self.L_hidden.sigmoida(X)
        input_4_out_layer = np.insert(f_activation_hidden,0,1)
        y_pred = self.L_output.sigmoida(input_4_out_layer)
        y_pred_number = np.argmax(y_pred)
        y_pred_list.append(y_pred_number)
    return np.array(y_pred_list)

def network_accuracy(self,X_input,y_input):
    counter = 0
    y_pred = self.network_output(X_input)
    for i in range(len(y_pred)):
        if y_pred[i] == y_input[i]:
            counter += 1

    accuracy = counter / len(y_pred) * 100
    return accuracy

a = Network()
%%time
a.learning(X_train,y_train,learning_rate = 0.1,epochs = 1)
a.network_accuracy(X_train,y_train)
a.network_accuracy(X_test,y_test)

```

ПРИЛОЖЕНИЕ Б

Реализация полносвязной нейронной сети с помощью библиотеки Keras

```
# Model / data parameters
num_classes = 10          # количество цифр
input_shape = (28, 28, 1) # размер картинки

# Load the data and split it between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255          # нормализация
x_test = x_test.astype("float32") / 255            # нормализация

# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Flatten(),
        layers.Dense(28, activation="sigmoid"),
        layers.Dense(10, activation="sigmoid"),
    ]
)

model.summary()
```

```
%%time
batch_size = 1    # количество подаваемых картинок за раз
epochs = 1

model.compile(loss="MSE", optimizer=keras.optimizers.SGD(learning_rate=0.1
), metrics=["accuracy"])
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs)

score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```