Реферат

Целью данной работы является построение программного и пользовательского интерфейсов для того, чтобы пользователь без опыта в программировании, но с наличием опыта в предметной области получил возможность взаимодействовать с сущзествующим модулем нейросететвого моделирвоания. Описание модуля: На вход: набор шаблонов для распознавания, количество эпох, параметры нейронной сети, номер модели мемристора и значения параметров модели. На выходе: набор временных срезов значений весов (состояний мемристоров) по эпохам. Модуль фиттинга: Реализован алгоритм случайного поиска и метод Хука-Дживса для уточнения. На вход: номер модели, верхние и нижние оценки параметров модели, набор точек ВАХа, набор точек напряжения от времени, кол-во итераций случайного поиска и максимальное количество шагов для метода Хука-Дживса. На выходе: номер модели, найденные значения параметров, исходная ВАХ и модельная ВАХ.

Обзор используемых технологий.

1. MongoDB
   MongoDB — документоориентированная система управления базами данных с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных. Написана на языке C++. Используется в веб-разработке, в частности, в рамках JavaScript-ориентированного стека MEAN.
2. Kotlin (spring framework)

   Kotlin (Ко́тлин) — статически типизированный, объектно-ориентированный язык программирования, работающий поверх Java Virtual Machine и разрабатываемый компанией JetBrains. Также компилируется в JavaScript и в исполняемый код ряда платформ через инфраструктуру LLVM. Язык назван в честь острова Котлин в Финском заливе, на котором расположен город Кронштадт.

   Авторы ставили целью создать язык более лаконичный и типобезопасный, чем Java, и более простой, чем Scala. Следствием упрощения по сравнению со Scala стали также более быстрая компиляция и лучшая поддержка языка в IDE. Язык полностью совместим с Java, что позволяет java-разработчикам постепенно перейти к его использованию; в частности, в Android язык встраивается с помощью Gradle, что позволяет для существующего android-приложения внедрять новые функции на Kotlin без переписывания приложения целиком.

Spring Framework (или коротко Spring) — универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET.

Первая версия была написана Родом Джонсоном, который впервые опубликовал её вместе с изданием своей книги «Expert One-on-One Java EE Design and Development» (Wrox Press, октябрь 2002 года).

Фреймворк был впервые выпущен под лицензией Apache 2.0 license в июне 2003 года. Первая стабильная версия 1.0 была выпущена в марте 2004. Spring 2.0 был выпущен в октябре 2006, Spring 2.5 — в ноябре 2007, Spring 3.0 в декабре 2009, и Spring 3.1 в декабре 2011. Текущая версия — 5.2.4.

Несмотря на то, что Spring не обеспечивал какую-либо конкретную модель программирования, он стал широко распространённым в Java-сообществе главным образом как альтернатива и замена модели Enterprise JavaBeans. Spring предоставляет бóльшую свободу Java-разработчикам в проектировании; кроме того, он предоставляет хорошо документированные и лёгкие в использовании средства решения проблем, возникающих при создании приложений корпоративного масштаба.

Между тем, особенности ядра Spring применимы в любом Java-приложении, и существует множество расширений и усовершенствований для построения веб-приложений на Java Enterprise платформе. По этим причинам Spring приобрёл большую популярность и признаётся разработчиками как стратегически важный фреймворк.

3. TypeScript(react framework)
   TypeScript — язык программирования, представленный Microsoft в 2012 году и позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript[3][4][5][6][7][8].

   Разработчиком языка TypeScript является Андерс Хейлсберг (англ. Anders Hejlsberg), создавший ранее Turbo Pascal, Delphi и C#.

   Спецификации языка открыты и опубликованы в рамках соглашения Open Web Foundation Specification Agreement (OWFa 1.0)[9].

TypeScript является обратно совместимым с JavaScript и компилируется в последний. Фактически, после компиляции программу на TypeScript можно выполнять в любом современном браузере или использовать совместно с серверной платформой Node.js. Код экспериментального компилятора, транслирующего TypeScript в JavaScript, распространяется под лицензией Apache. Его разработка ведётся в публичном репозитории через сервис GitHub[10].

TypeScript отличается от JavaScript возможностью явного статического назначения типов, поддержкой использования полноценных классов (как в традиционных объектно-ориентированных языках), а также поддержкой подключения модулей, что призвано повысить скорость разработки, облегчить читаемость, рефакторинг и повторное использование кода, помочь осуществлять поиск ошибок на этапе разработки и компиляции, и, возможно, ускорить выполнение программ.

Планируется, что в силу полной обратной совместимости адаптация существующих приложений на новый язык программирования может происходить поэтапно, путём постепенного определения типов.

На момент релиза представлены файлы для восприятия расширенного синтаксиса TypeScript для Vim и Emacs, а также плагин для Microsoft Visual Studio.

Одновременно с выходом спецификации разработчики подготовили файлы с декларациями статических типов для некоторых популярных JavaScript-библиотек, среди которых jQuery.

React (иногда React.js или ReactJS) — JavaScript-библиотека[5] с открытым исходным кодом для разработки пользовательских интерфейсов.

React разрабатывается и поддерживается Facebook, Instagram и сообществом отдельных разработчиков и корпораций

React может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как MobX, Redux и GraphQL.

Нейроморфные системы.

Это — процессор, работа которого основана на принципах действия человеческого мозга. Такие устройства моделируют работу нейронов и их отростков — аксонов и дендритов — отвечающих за передачу и восприятие данных. Связи между нейронами образуются за счет синапсов — специальных контактов, по которым транслируются электрические сигналы.

Одна из задач нейроморфных устройств — ускорить обучение сверточных нейронных сетей для распознавания изображений. Системам искусственного интеллекта на базе этой технологии не нужно обращаться к массивному хранилищу с тренировочными данными по сети — вся информация постоянно содержится в искусственных нейронах. Такой подход дает возможность реализовывать алгоритмы машинного обучения локально. Поэтому ожидается, что нейроморфные чипы найдут применение в мобильных устройствах, IoT-гаджетах, а также дата-центрах.

Мемристоры.

Мемри́стор (от англ. memory — память, и англ. resistor — электрическое сопротивление) — пассивный элемент в микроэлектронике, способный изменять своё сопротивление в зависимости от протекавшего через него заряда (интеграла тока за время работы). Может быть описан как двухполюсник с нелинейной вольт-амперной характеристикой, обладающий гистерезисом.

На текущий момент мы рассматриваем однослойную нейроморфную сеть состоящую из 1T1R мемристорного кроссбара и нейронов. Мемристоры по сути выступают в качестве синапсов. Есть разные математические модели мемристоров, которые можно подстраивать под экспериментальные данные. Этой задачей занимается модуль фиттинга. Задав модель мемристора, её параметры и параметры нейроморфной сети можно запустить процесс моделирования работы сети. Этим занимается модуль нейросетевого моделирования.

Описание микросервиса.

Структура:

1. Контракты для десериализации объектов существующего API.
    - Юнит тесты на десериалищации из json’ов, приложенных в примере

- Концептуально разделить read/write сущности, либо подумать над тем, как логически разделить эти два типа доступов, либо может быть можно один контракт использовать.
- Формат проекта – библиотека классов

2. Клиент, взаимодействующий непосредственно с API:
   - Клиент не знает НИЧЕГО о том, что происходит уровнем выше, только получение/отправка данных в формате контрактов
   - Проконсультироваться про взаимодействие между контрактами API и внутренним форматом данных (мэппинг будет на уровне сервиса)
   - Нужно ли хранить данные из API в бд?(скорее всего нет)
   - Вытекает из предыдущего: кажется, что нужно настроить мэппинг между контрактами API и внутренним форматом данных.(Актуальный вопрос, но для backend в целом, клиента это не касается)
   - Клиент API и сервер приложения делить на проекты или нет?
   - Подключение логгера
   - Юнит тесты
   - Интеграционные тесты
3. Изолированная работа с бд:
   - Чисто CRUD с DataObject и ничего лишнего
   - Логгирование
4. Сервис, где заключена основная бизнес логика:
   - Попробовать использовать DDD
   - Мэппинг DataObject и ApiContracts
5. Сам API:
   - Подключение сервиса через DI
   - Swagger
   - HealthChecks
   - Четкое документирование, возвращаемый код в соответствии описанием кодов

Слои данных в целом в проекте (от самого сырого до пользовательского):

1. Контракты API
2. Внутренняя модель данных сервиса
3. DataObjects – модели доступа к бд
4. Контракты для UI (учитывая простоту моделей, возможно, избыточно)

Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Aufgezählt+ Ebene:1 + Ausgerichtetan: 1,9 cm + Einzug bei: 2,54 cm
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Aufgezählt+ Ebene:1 + Ausgerichtetan: 1,9 cm + Einzug bei: 2,54 cm

5. Контракты и модели данных UI (учитывая простоту моделей, можно схлопнуть до одного уровня)

Ответственности:

1. Клиент для работы с API
2. Repository для изолированния работы с бд
3. Service, который будет использовать клиент, описанный выше через DI
4. Непосредственно "ручки" API

Общие требования:

- Конфиги(например API_URL) хранить в env
- Юнит тесты
- Интеграционные тесты
- dockerfile
- helm chart

Neuromorphic Systems Interface

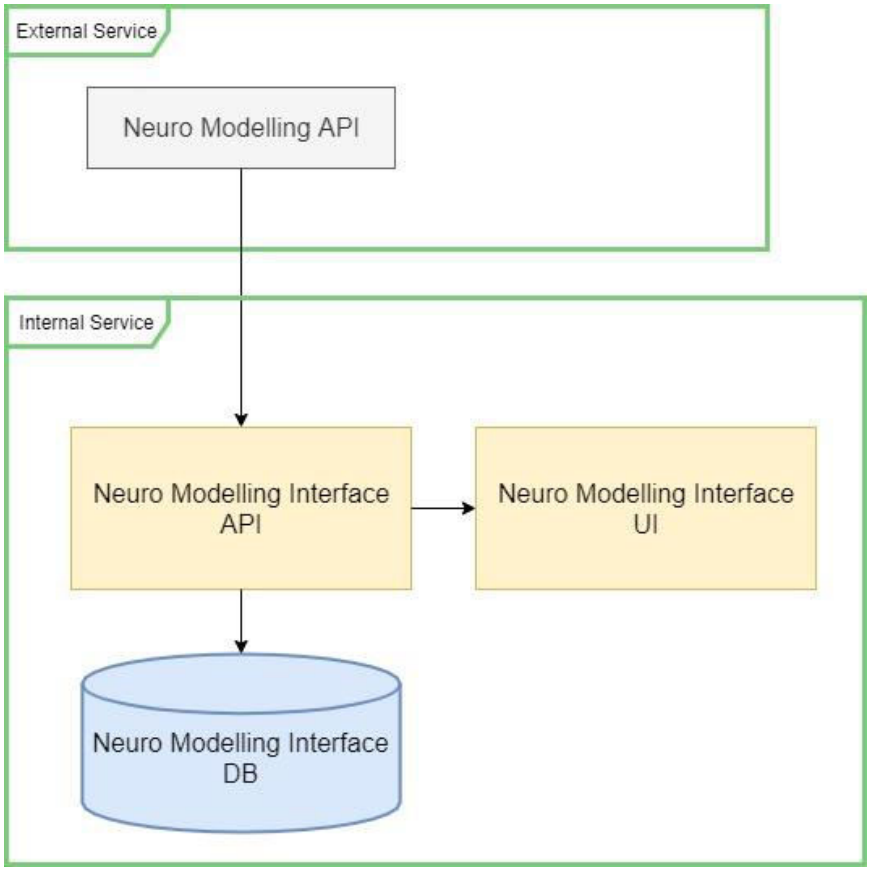
Solution Approach

Цель документа

Данный документ описывает архитектуру интрефейса для модуля нейросетевого моделирования.

Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Einzug: Links: 1,27 cm, Keine Aufzählungen oder Nummerierungen
Formatiert: Englisch (USA)
Formatiert: Englisch (USA)
Formatiert: Englisch (USA)

Схема потоков данных



External Service - существующий сервис для нейросетевого моделирования. С ним работа ведется как с черным ящиком.
Internal Service - сервис, который необходимо реализовать.

Схема вызовов

На текущий момент возможны следующие варианты:

1. Использование для взаимодействия backend-frontend только Rest API
   Плюсы:
   - Простота реализации
   - Возможность пользователя коммуницировать с сервером не только через UI, но и через Swagger, либо выполнять запросы с помощью утилиты curl (если, к примеру на операционной системе нет возможности использовать UI)

   Минусы:
   - Скорость меньше, чем при использовании web socket
2. Использование для взаимодействия backend-frontend только Web Socket
   Плюсы:

- Высокая скорость взаимодействия

Минусы:

- Более сложный в написании и поддержке код, как backend, так и frontend
- Нет возможности использовать swagger или выполнять запросы через curl

3. Использование для взаимодействия backend-frontend Web Socket, но также оставить rest endpoints

Плюсы:

- Сочетает в себе все плюсы использования технологий по отдельности

Минусы:

- Слишком много кода для написания, поддержки и отладки

В связи с тем, что высоких нагрузок на сервис не ожидается, был выбран первый вариант - использование только Rest API.

Реализация класса клиента для взаимодействия с Neuromophic API.
Требования к классу:

- Получение конфигурации: логина, пароля, url-адреса.
- Получение сериализатора из DI Container
- Получение логгера из DI Container
- Покрытие тестами
- Использование http client. Тут воможны варианты:

  1. Использовать отношение “является”, то есть написать класс – наследник для http client. В случае наследования необходимо, чтобы класс наследник удовлетворял следующим концептуальным критериям:

  а) Выполнение отношения “является”. То есть необходимо, чтобы производный класс представлял собой подмножество множества допустимых значений класса родителя, а не просто использовать его методы. К примеру, для класса GeometricFigure с единственным методом GetSquare() можно написать производный класс House. Метод получения площади для него будет актуален и полезен, однако отношение “является” не выполняется, что приносит диссонанс в концептуальную структуру класса. В этом примере лучше использовать отношение “включает” и использовать GeometricFigure как член класса House, который содержит информацию о геометрической фигуре, которая максимально похожа на фигуру дома. В данном же случае, предлагается сделать класс, который будет расширять функционал стандартного http client’а, то есть отношение “является” выполнятеся.

Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch

Б) Выполение принципа подстановки Барбары Лисков. Сам принцип гласит следующее: любой производный класс должен расширять функционал базового, а не замещать, а также любой производный класс может быть поставлен на место базового и это ничего не изменит в работе приложения. Несложно заметить, что этот приницип тесно переклекается с концепцией отношения “является”, соответственно также, как и отношение, принцип выполняется для класса.

4. Использовать отношение “включает”. При использовании данного отношения класс http client будет передаваться через DI Container, как и логгер.

В итоге был выбран вариант сделать http client членом класса.

Аргументы:

- При использовании наследования клиентскому коду будет доступен функционал по работе с http запросами, в то время как по всем современным принципам разработки не рекомендуется создавать классы с лишнимии методами. К примеру один из ключевых принципов разработки ПО – KISS(keep it short simple) включает в себя этот тезис.
- Хотя принцип подстановки и выполняется, при более подробном анализе класса, становится понятным, что не выполняется отношение “является”, поскольку реализуемый класс содержит не только функционал для работы с http запросами, но и сериализацию.

Для реализуемого класса также необходимо выполнение принципов сокрытия информации. Соответственно по итогу полученный класс должен:

- Инкапсулировать работу с http. То есть возращать клиентскому коду обработанные данные в виде моделей контрактов, никакой информации о кодах response и тому подобного методы класса возвращать не должны.
- Выполнять логгирование.
- Не содердать абсолютно никакой бизнес логики. Этот класс просто выполняет запросы, логгирует и возвращает значения. Вся логика должна быть уровнем выше.

Однако, в связи с тем, что существует несколько концептуально разных типов моделей: memristor, network, количество требуемых методов увеличивается вдвое. В совей книге Артур Ариэль описывает “магическое” число 7 +-2 и предлагает исопльзовать его для характеристики максимального количества методов в классе. Данное число взято не из воздуха, оно показывает, сколько примерно предметов может запомнить

Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Schriftart:(Standard) TimesNewRoman,Schriftartfarbe: Automatisch,Muster: Transparent
Formatiert: Russisch
Formatiert: Schriftart:(Standard) TimesNewRoman,Schriftartfarbe: Automatisch,Muster: Transparent
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch
Formatiert: Russisch

человек одновременно. Соответственно, в нашем случае, число методов явно будет больше верхней границы этого интервала – девяти. Также для концептуальной целостности и декомпозиции можно разделить класс client на два(или возможно больше) класса(ов). Остается только подобрать нужные элементы для декомпозиции. Для этого был выбран эмпирический путь. То есть сначала реализация с минимальным уровнем декомпозиции, возможно даже вообще без нее, а уже по ходу будет понятно, как нужно разделить класс. Может показаться, что это означает "пустить на самотек задачу", однако, согласно известному правилу "80/20", только 80% ПО нуждается в проектировке до начала разработки, остальные же 20 можно проектировать по ходу разработки.

План разработки клиента:

1 Класс ApiSettings с настройками для доступа к API

2 Добавление файла кофигурации, чтобы можно было менять их, не изменяя исходный код приложения

3 Подготовка кастомного сериализатора

4 Класс "прототип" для клиента

5 Насписание юнит тестов для класса, построение плана для декомпозиции класса

6 Реализация декомпозиции класса

7 Написание оставшихся юнит тестов

~~3.~~ 8 Тестирование и(или) написание интеграционных тестов--

Formatiert: Russisch

Formatiert: Russisch

Formatiert: Russisch

Formatiert: Russisch

Formatiert: Russisch

Formatiert: Russisch

Formatiert: Standard, Keine Aufzählungen oder Nummerierungen

Formatiert: Englisch (USA)