kaspersky

Kaspersky Neuromorphic Platform

Содержание

О Kaspersky Neuromorphic Platform

Kaspersky Neuromorphic Platform (далее также платформа) – это программно-аппаратная платформа, предназначенная для эмуляции импульсных нейронных сетей и поддержки их выполнения на разных вычислителях.

С помощью Kaspersky Neuromorphic Platform вы можете:

- Создавать и обучать нейронные сети на различных типах входных данных (например, телеметрия, события, изображения, 3D-данные и звуковые и тактильные данные).
- Оптимизировать структуру и гиперпараметры загруженных нейронных сетей.
- Проводить прикладные исследования в области классификации входных данных и других областях применения импульсных нейронных сетей.
- Разрабатывать новые топологии нейронных сетей (например, импульсные аналоги сверточных нейронных сетей, предполагающих свертку по пространству и времени).
- Разрабатывать новые модели синаптической пластичности.
- Реализовывать новые модели нейронов.
- Реализовывать прикладные решения на основе нейроморфных импульсных нейронных сетей и искусственного интеллекта в области робототехнических манипуляторов, интернета вещей, беспилотных систем, человеко-машинного взаимодействия, носимых устройств и оптимизационного планирования.
- Реализовывать прикладные решения на устройствах с пониженным энергопотреблением или с использованием нейроморфных процессоров (англ. Neural Processing Unit, NPU).

Для решения вышеперечисленных задач вы можете использовать языки C++ и Python®. Платформа поддерживает CPU. Фреймворк Kaspersky Neuromorphic Platform можно также использовать как бекенд для PyNN API.

Комплект поставки

Комплект поставки Kaspersky Neuromorphic Platform зависит от области применения платформы. Kaspersky Neuromorphic Platform может быть использован для работы с нейронными сетями (например, для реализации новых моделей нейронов), а также совместной разработки платформы.

Комплект поставки Kaspersky Neuromorphic Platform для разработки прикладных решений

В комплект поставки входят следующие пакеты:

- deb-пакеты с бинарным кодом бекендов, включая код бекенда для нейроморфного процессора Алтай-2.
- deb-пакет с бинарным кодом фреймворка для C++, содержащий:
  - Бинарный код библиотек платформы.
  - Файл с условиями использования платформы LICENSE.txt на английском языке.

    - Файл с информацией о платформе README.md на английском языке.
- dev-deb пакет для deb-пакета с бинарным кодом на C++. В dev-deb пакет входят следующие файлы:
    - Заголовочные файлы.
    - Модули CMake для сборки платформы на C++.
    - Статические версии библиотек платформы.
- deb-пакет с Python-фреймворком.
- deb-пакет со следующими файлами:
    - Документацией по использованию платформы.
    - Справочным руководством по API.

Комплект поставки Kaspersky Neuromorphic Platform для разработки прикладных решений на языке программирования Python

В комплект поставки Kaspersky Neuromorphic Platform входят следующие файлы:

- deb-пакет с Python-фреймворком.
- whl-пакет для установки с платформы PyPI с помощью менеджера пакетов pip.

Комплект поставки Kaspersky Neuromorphic Platform с исходным кодом платформы

Kaspersky Neuromorphic Platform поставляется в виде архива, в который входят следующие файлы:

- Исходный код платформы.
- Документация по использованию платформы.
- Справочное руководство по API.
- Файл с информацией о платформе README.md на английском языке.
- Файл с условиями использования платформы LICENSE.txt на английском языке.
- Файл с информацией о платформе (Release Notes).
- Файл с информацией о стороннем коде NOTICES.txt на английском языке.
- Файл с правилами участников репозитория CONTRIBUTING.md на английском языке.
- Файл с условиями участия в репозитории Contributor License Agreement.docx на английском языке.

Вы можете получить файлы архива путем клонирования репозитория платформы.

Аппаратные и программные требования

Для функционирования Kaspersky Neuromorphic Platform компьютер должен удовлетворять следующим минимальным требованиям.

Список поддерживаемых процессоров:

- центральный процессор: совместимый процессор Intel Core i5 и выше;
- нейроморфный процессор Алтай-2.

Минимальные аппаратные требования:

- 1 ядро;
- 8 ГБ оперативной памяти;
- 10 ГБ свободного пространства на жесткой диске.

Процессор должен поддерживать следующие расширения:

- Advanced Vector Extensions (avx);
- Advanced Vector Extensions 2 (avx2).

Поддерживаемые операционные системы:

- Debian GNU/Linux 12.5 и выше;
- Ubuntu 22.04 LTS и выше.

Вы можете использовать устройство под управлением любой другой операционной системы из семейства Linux, если дистрибутив операционной системы содержит библиотеку Boost версии 1.80 и выше.

Для работы с платформой на устройстве должны быть установлено следующее программное обеспечение:

- Python 3.8;
- TensorFlow 2.4.1 и выше;
- Keras 2.3.1 и выше;
- NumPy 2.4.1 и выше.

  Версия библиотеки NumPy должна соответствовать версии библиотеки TensorFlow.

О бекенде Kaspersky Neuromorphic Platform

Бекенд Kaspersky Neuromorphic Platform предоставляет стандартизированный интерфейс для работы с вычислителем. Бекенд платформы позволяет запускать эмуляцию работы импульсной нейронной сети на различных вычислительных устройствах.

Физически бекенд платформы разделяется на библиотеки и компилируется в несколько независимых друг от друга бекендов. Каждый такой бекенд позволяет выполнять эмуляцию на разных типах устройств.

Бекенд Kaspersky Neuromorphic Platform выполняет следующие функции:

- Запускает импульсные нейронные сети на различных вычислителях.
- Выполняет статические проверки корректности построения импульсной нейронной сети.
- Предоставляет общую кодовую базу для реализации вычислителей.

Обмен данными в Kaspersky Neuromorphic Platform

Kaspersky Neuromorphic Platform обеспечивает обмен данными между объектами библиотеки поддержки бекендов, а также между объектами платформы и объектами внешней среды. Обмен данными между объектами библиотеки поддержки бекенда Kaspersky Neuromorphic Platform осуществляется через сообщения.

Обмен данными между популяциями и проекциями

Нейроны популяции генерируют сообщения, содержащие спайки, на которые может быть подписано более одной проекции. Спайки популяций передаются проекциям синапсов.

После получения спайков бекенд вычисляет синаптические воздействия. Проекции отправляют сообщения со значениями синаптических воздействий постсинаптическим популяциям нейронов.

Обмен данными между бекендом и внешней средой

Бекенд принимает данные от внешней среды в виде спайков. Спайки могут быть переданы в бекенд в синхронном режиме (по запросу пользователя). Для получения спайков в синхронном режиме вызывается блокирующий метод источника данных.

Сообщения от бекенда во внешнюю среду могут передаваться подписчикам по сигналу бекенда или в синхронном режиме по запросу пользователя.

Обмен данными между нейронной сетью и внешней средой

Из внешней среды в нейронную сеть данные поступают через каналы ввода, которые преобразуют эти данные в спайки.

Во внешнюю среду нейронная сеть передает данные через каналы вывода, подключенные к указанным популяциям и преобразующие спайки в данные. Передача данных, например потенциала мембраны, возможна также через подсистему мониторинга.

Поддерживаемые бекенды

В Kaspersky Neuromorphic Platform поддерживаются следующие типы бекендов:

- Бекенды для CPU:
  - *Однопоточный бекенд для CPU*. Бекенд работает только в одном потоке. Вы можете использовать однопоточный бекенд для отладки платформы и работы нейронной сети.
  - *Многопоточный бекенд для CPU*. Бекенд работает в нескольких потоках одновременно, что обеспечивает повышение скорости расчета нейронной сети. Вы можете использовать многопоточный бекенд для эмуляции импульсной нейронной сети.
- Бекенды для нейроморфного процессора Алтай-2:
  - *SNN*. Бекенд используется для эмуляции импульсных нейронных сетей.

    На рисунке ниже представлена схема взаимодействия компонентов платформы при использовании бекенда SNN. Импульсная нейронная сеть создается с помощью компонентов платформы и обучается методом синаптической пластичности. Далее нейронная сеть размещается с помощью утилиты placer на нейроморфном процессоре Алтай-2. Вы можете запускать исполнение импульсных нейронных сетей как на программном эмуляторе в составе бекенда SNN, так и непосредственно на аппаратном обеспечении нейроморфного процессора Алтай-2.

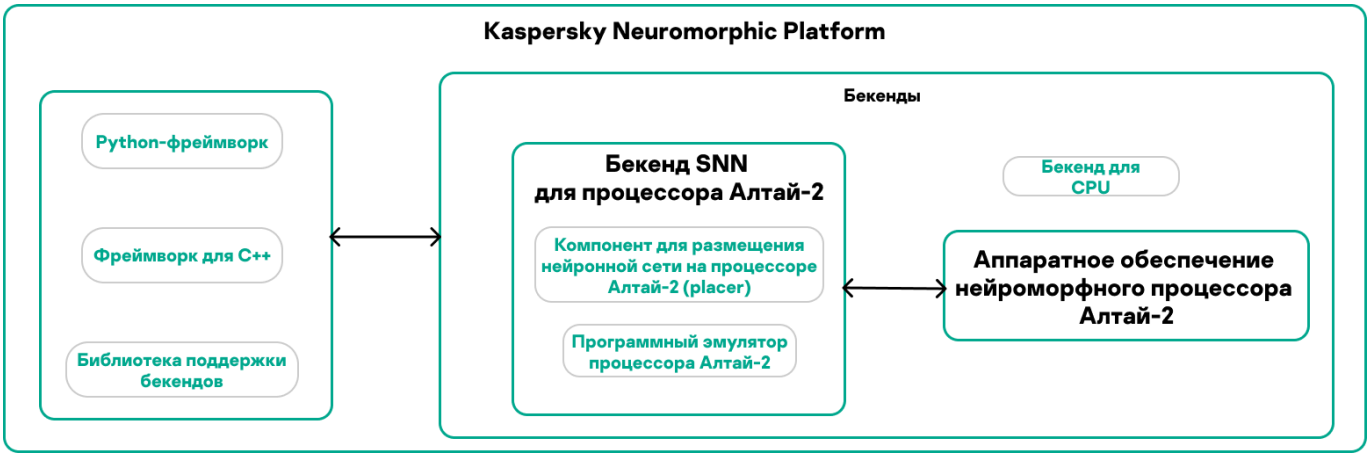


Схема взаимодействия платформы с бекендом SNN

  - *ANN2SNN*. Бекенд используется для эмуляции нейронных сетей, преобразованных в импульсные. Обучение нейронной сети производится обратным распространением ошибки с квантованием весов.

    На рисунке ниже представлена схема взаимодействия компонентов платформы при использовании бекенда ANN2SNN. Нейронная сеть создается с помощью тернарных слоев библиотеки ANN2SNN, реализованной как часть Python-фреймворка, и обучается путем обратного распространения ошибки с квантованием весов. Далее нейронная сеть размещается с помощью утилиты placer на нейроморфном процессоре Алтай-2. Размещенная нейронная сеть может быть исполнена как на программном эмуляторе в составе бекенда ANN2SNN, так и непосредственно на аппаратном обеспечении нейроморфного процессора Алтай-2.

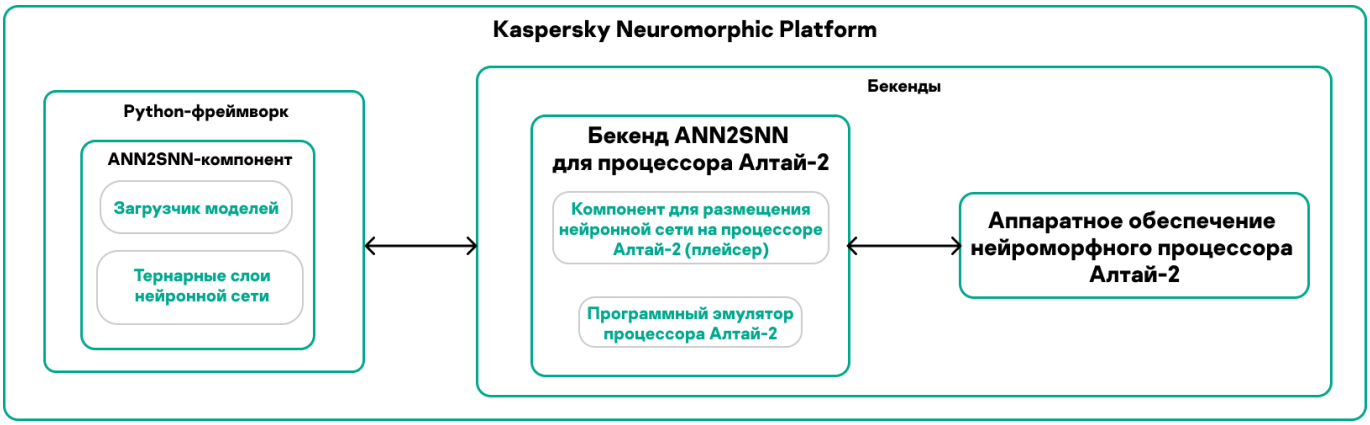


Схема взаимодействия компонентов платформы в случае использования библиотеки ANN2SNN

О работе бекенда для CPU

Работа бекенда для CPU основывается на обмене сообщениями между проекциями и популяциями. Объекты библиотеки поддержки бекендов также могут обмениваться сообщениями с объектами внешней среды. Для обмена сообщений бекенд предоставляет шину сообщений `MessageBus`.

Для инициализации бекенда требуется предварительно загрузить проекции, содержащие синапсы, и популяции, содержащие нейроны. В процессе инициализации бекенда загруженные проекции синапсов подписываются на спайки от связанных популяций или каналов ввода, а загруженные популяции подписываются на синаптические воздействия от связанных проекций.

Цикл работы бекенда состоит из следующих этапов:

1. Каналы ввода отправляют данные в шину сообщений `MessageBus`.
2. Бекенд вычисляет функцию популяции нейронов.
3. Популяции нейронов отправляют спайки с результатами вычислений в шину сообщений `MessageBus` с помощью точек подключения `MessageEndpoint`.
4. Шина сообщений `MessageBus` отправляет данные, полученные от популяций, в проекции или каналы вывода.
5. Бекенд вычисляет функцию проекции синапсов.
6. Проекции синапсов отправляют синаптические воздействия с результатами вычислений в шину сообщений `MessageBus` с помощью точек подключения `MessageEndpoint`.
7. Шина сообщений `MessageBus` отправляет данные, полученные от проекций, в связанные популяции нейронов.

О нейроморфном процессоре Алтай-2

Нейроморфный процессор Алтай-2 – это сверхбольшая интегральная схема (СБИС) с вычислительной архитектурой, которая имитирует работу биологических сенсорных и нервных систем. Нейроморфный процессор Алтай-2 предназначен для исполнения (инференса) импульсных нейронных сетей. В архитектуре процессора Алтай-2 реализован принцип "вычисления рядом с памятью", позволяющий избежать характерных для архитектуры фон Неймана избыточных энергозатрат на передачу данных между блоком памяти и вычислительным ядром. Нейроморфный процессор Алтай-2 обладает низким энергопотреблением.

Элементарной единицей импульсной нейронной сети является нейрон, у которого есть набор входных сигнальных линий (дендриты) и выходной сигнальной линией (аксон). Состояние нейрона описывается нейронным потенциалом. Если в процессе исполнения импульсной нейронной сети потенциал нейрона превышает заданный порог, то нейрон испускает по своему аксону спайк и отправляет его в заданные целевые нейроны. Спайк может быть как положительным, так и отрицательным. При моделировании нейрона также учитывается механизм утечки, предназначенный для релаксации нейронного потенциала.

Нейроны импульсной нейронной сети связаны друг с другом посредством синапсов. Каждому синапсу может быть присвоено значение веса, на которое изменяется потенциал нейрона при получении спайка от нейрона, связанного с этим синапсом.

Основной структурной единицей архитектуры нейроморфного процессора Алтай-2 является *ядро*. Ядро объединяет в себе группу нейронов и память для хранения их параметров. Функцию нейронов в ядре выполняет конечный автомат, который моделирует поведение нейронов.

Синхронизация работы всех нейронов импульсной нейронной сети осуществляется с помощью сигнала *тик*, после подачи которого потенциалы всех нейронов последовательно обновляются всеми ядрами и при необходимости испускают спайки. За тик ядро выполняет 262 144 синаптических операций. При этом частота тика не превышает 2000 Гц. Одна синаптическая операция выполняется за один такт синхросигнала с частотой 600 МГц.

Нейроморфный процессор Алтай-2 имеет регулярную структуру ядер в виде прямоугольной сети, в которой каждое ядро непосредственно связано со своими 4 соседями. Для передачи спайков между нейронами разных ядер реализован механизм маршрутизации таких сигналов по сети ядер. Соответственно, при выполнении функции нейрона к потенциалу моделируемого нейрона могут быть добавлены значения нейронных потенциалов, отправленных нейронами из других ядер.

В нейроморфном процессоре Алтай-2 предусмотрена возможность масштабирования за счет увеличения количества СБИС. По вопросам масштабирования вам нужно обратиться к специалистам "Лаборатории Касперского".

В таблице ниже приведены общие сведения о нейроморфном процессоре Алтай-2 и его составляющих:

Общие сведения о процессоре Алтай-2

Свойство	Ограничение
СБИС	
Площадь кристалла	6 кв. мм
Технология изготовления	28 нм
Максимальное количество нейроморфных ядер	256
Количество нейроморфных ядер в прототипе	16

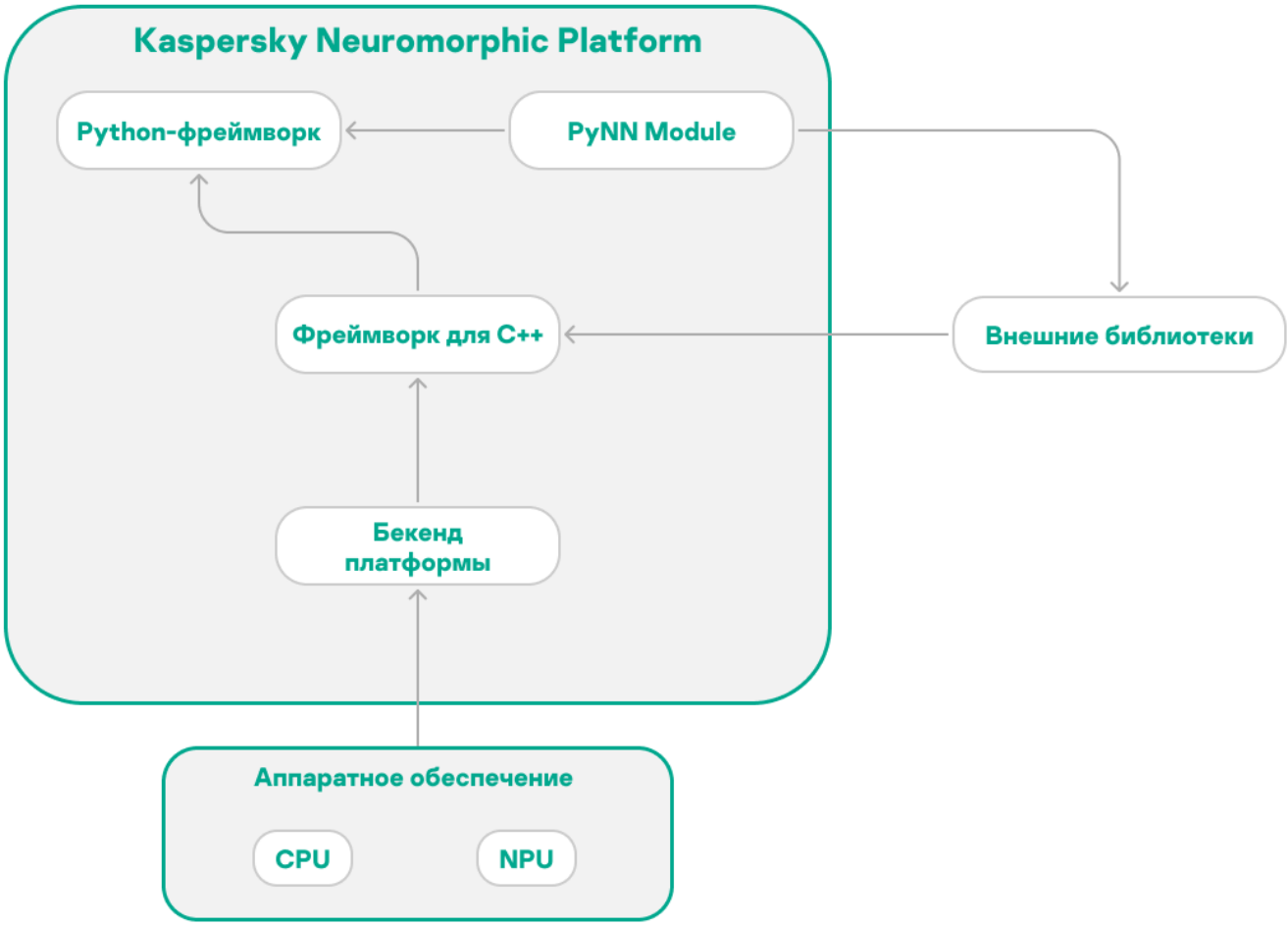
Максимальное количество моделируемых нейронов	131 072
Количество моделируемых нейронов в прототипе	8192
Ядро	
Количество моделируемых нейронов	512
Количество входных линий	512
Диапазон значений потенциала нейрона	От -32 768 до 32 767 Потенциал нейрона хранится в виде 16-битной величины.
Диапазон значений веса синапса	Вес синапса хранится в виде 8-битовой беззнаковой величины.
Вариативность весов на нейрон	4
Количество синаптических операций	9 Каждая синаптическая связь между моделируемым нейроном и входной линией характеризуется одной из девяти синаптических операций, выбором которых можно управлять воздействием этой синаптической связи на нейрон. Возможны следующие синаптические операции: • Пустая операция. Во время этой операции не происходит никаких изменений. • Прибавить вес N, где N – это номер веса в диапазоне от 1 до 4. • Вычесть вес N, где N – это номер веса в диапазоне от 1 до 4.
Количество синаптических операций за тик	262 144
Частота тика	Не более 2000 Гц
Энергопотребление ядра	Не более 4 мВт
Объем памяти ядра	88 КБ
Объем памяти ядра для хранения параметров нейрона	16 КБ

Архитектура Kaspersky Neuromorphic Platform

Kaspersky Neuromorphic Platform включает в себя следующие компоненты:

- *Бекенд*. Запускает эмуляцию работы импульсной нейронной сети на различных вычислительных устройствах (например, центральных процессорах).
- *Фреймворк для C++*. Обеспечивает управление бекендами.
- *Python-фреймворк*. Обеспечивает управление бекендами.
- *PyNN Module*. Обеспечивает интерфейс к Python-фреймворку.

На рисунке ниже представлена схема взаимодействия компонентов Kaspersky Neuromorphic Platform.



Архитектура Kaspersky Neuromorphic Platform

Компоненты платформы для использования на C++

Этот раздел содержит описание компонентов для использования Kaspersky Neuromorphic Platform на языке программирования C++.

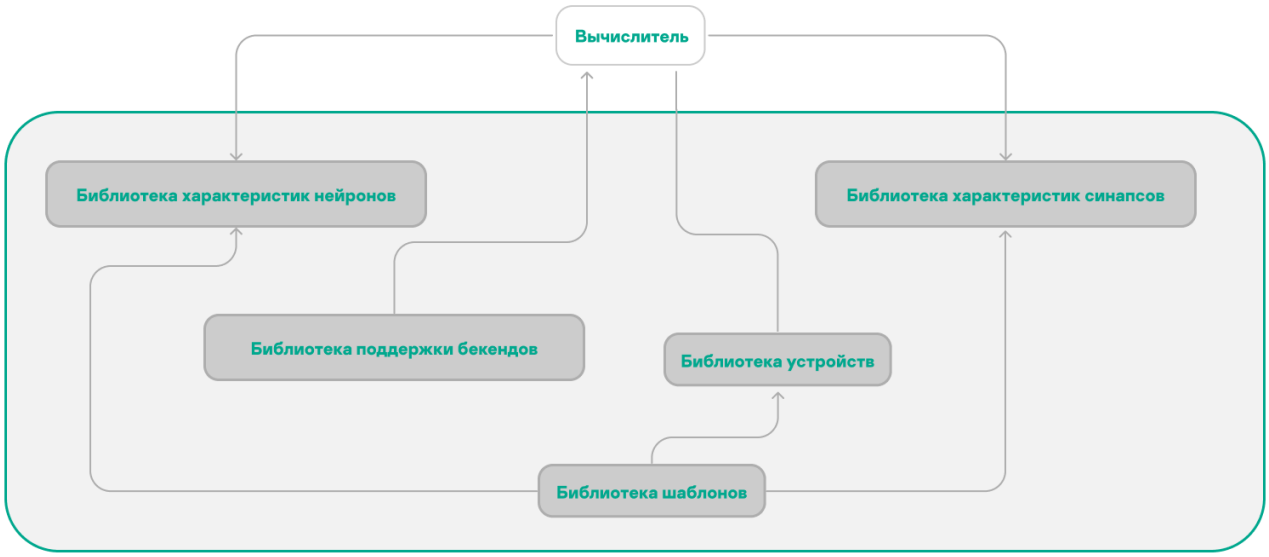
Компоненты бекенда платформы для C++

Для реализации бекендов вы можете использовать следующие библиотеки платформы на языке программирования C++:

- *Библиотека поддержки бекендов*. Реализует основные объекты и шаблоны вычислителя.
- *Библиотека устройств*. Предоставляет набор интерфейсов для взаимодействия с физическими устройствами.
- *Библиотека характеристик нейронов*. Содержит наборы свойств и типизацию популяций моделей нейронов.
- *Библиотека характеристик синапсов*. Содержит набор классов, определяющих поведение синапса.
- *Библиотека шаблонов*. Предоставляет шаблоны общего назначения.

На рисунке ниже представлена схема взаимодействия библиотек платформы.

В Kaspersky Neuromorphic Platform вычислитель хранит в себе интерфейс устройства, а также экземпляры объектов библиотеки поддержки бекендов (например, проекции и популяции), необходимые для выполнения прикладных задач на вычислителе. Шаблоны популяций и проекций нейронной сети должны быть специализированы типами нейронов и синапсов, описанными в библиотеках характеристик нейронов и синапсов.



Взаимодействие библиотек для C++ в Kaspersky Neuromorphic Platform

Библиотека поддержки бекендов

В библиотеке поддержки бекендов реализуются интерфейсы к объектам, необходимым для работы бекендов и выполнения прикладных задач. Библиотека содержит пространство имен `core`.

Таблица ниже содержит описание объектов библиотеки.

Объект	Описание
Backend	Базовый класс для конкретных реализаций бекендов.
BaseData	Структура, определяющая набор базовых данных.
continuously_uid_generator	Класс, реализующий генератор уникальных идентификаторов.
Device	Базовый класс устройств.
MessageBus	Класс, реализующий интерфейс к шине сообщений.
MessageEndpoint	Класс, реализующий интерфейс к точке подключения.
Messaging	Пространство имен, определяющее интерфейсы к сообщениям.
Population	Шаблонный класс, реализующий контейнер нейронов одной модели.
Projection	Шаблонный класс, реализующий контейнер синапсов одного типа.
Subscription	Класс, реализующий подписку на сообщения.
synapse_access	Пространство имен, реализующее соединения и интерфейс для доступа к их реестру.
TagMap	Класс, реализующий интерфейс к словарю тегов.
UID	Класс, определяющий уникальные идентификаторы.
uid_hash	Функция, реализующая хеширование UID.

На рисунке ниже представлена схема взаимодействия объектов библиотеки.

В Kaspersky Neuromorphic Platform структура `BaseData` содержит базовую информацию об объекте и включает в себя `UID` и `TagMap`. Структура `BaseData` используется в классах `Projection`, `Population` и `Backend`, соответственно каждый из них обладает идентификатором `UID` и набором тегов и их значениями `TagMap`.

Классы `Population` и `Projection` принимают и отправляют сообщения из пространства имен `Messaging`, которые содержат в своем заголовке UID отправителя.

Класс `Backend` содержит классы `Device` и `MessageBus`. Шина сообщений `MessageBus` создает точки подключения `MessageEndpoint` и взаимодействует с ними в процессе обмена сообщениями. Каждая точка подключения `MessageEndpoint` содержит контейнер подписок класса `Subscription`.

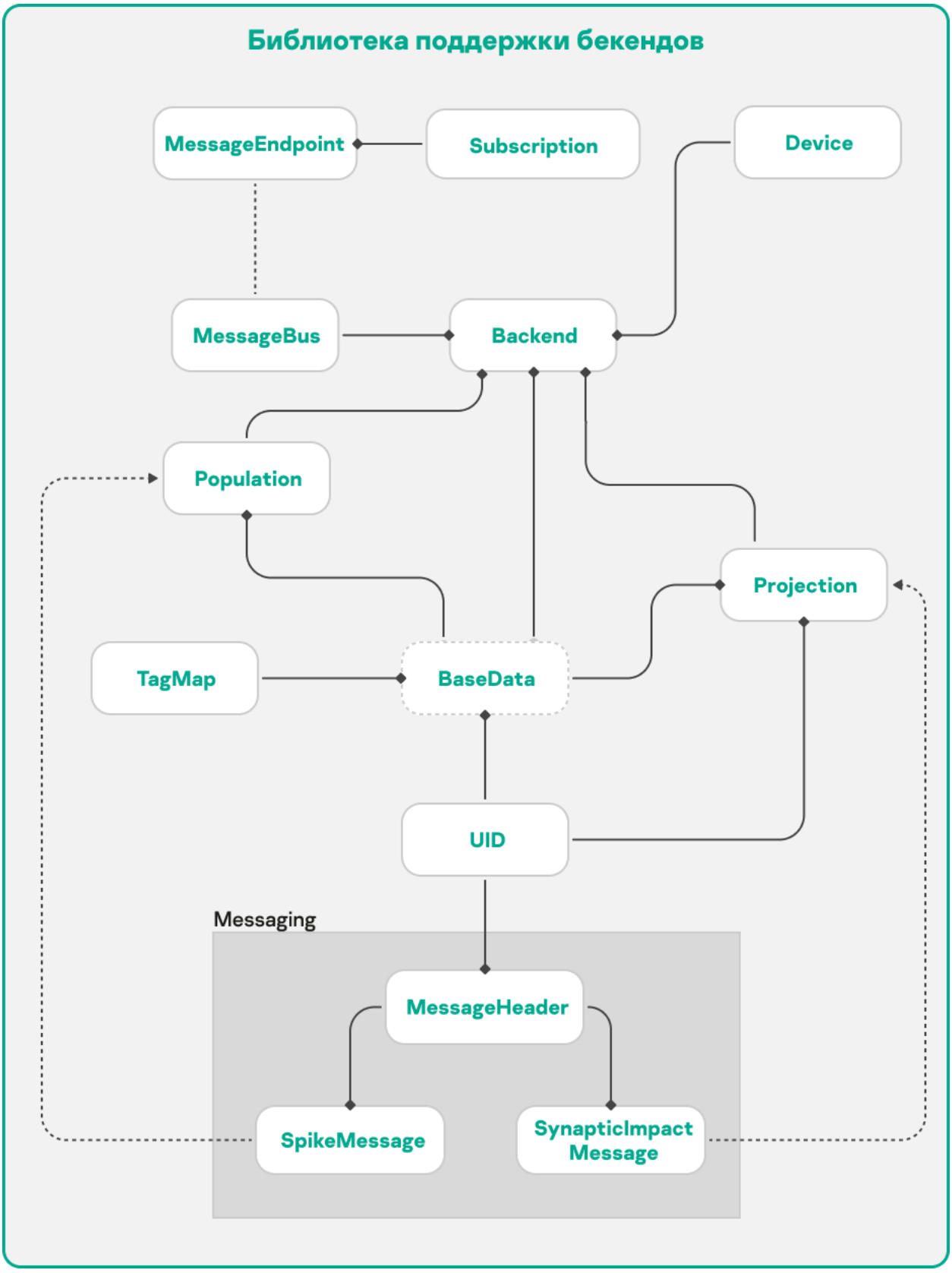


Схема взаимодействия объектов библиотеки поддержки бекендов

Класс UID

Объект класса `UID` является частью структуры `BaseData` и представляет собой идентификатор объекта. Идентификатор является уникальным.

Функция uid_hash

Функция `uid_hash` реализует функцию хеширования идентификаторов объектов.

Класс continuously_uid_generator

Класс `continuously_uid_generator` реализует генератор уникальных идентификаторов (англ. UID). Идентификатор отображается в виде строки, разбитой на группы с помощью дефисов, и представляет собой 128-битное число. Идентификаторы генерируются последовательно с шагом, равным 1.

Вы можете использовать класс `continuously_uid_generator` для отладки платформы и исполнения нейронной сети.

Класс TagMap

Объект класса `TagMap` является составляющей частью структуры `BaseData` и представляет собой ассоциативный массив, включающий ноль или более записей типа "ключ-значение тега", где ключом является имя тега. Имена тегов имеют строковый тип и являются уникальными.

Структура BaseData

Структура `BaseData` определяет набор базовых данных, общих для нескольких объектов библиотеки поддержки бекендов и объектов фреймворка. Например, базовые данные `BaseData` используются в объектах классов `Backend`, `Device`, `Population` и `Projection`.

Базовые данные `BaseData` включают в себя следующее:

- `uid_` – уникальный идентификатор объекта.
- `tags_` – ассоциативный массив `TagMap`, состоящий из тегов, используемых объектом, и их значений.

Класс Population

Шаблонный класс `Population` реализует контейнер, который содержит нейроны, принадлежащие одной модели. При использовании шаблона класса `Population` вы можете указать один из типов нейронов, поддерживаемых библиотекой характеристик нейронов.

С помощью класса `Population` вы можете передать данные популяций на бекенд для вычисления их атрибутов. После вычисления новые атрибуты популяций хранятся на бекенде. Чтобы передать новые атрибуты от бекенда в соответствующие объекты популяций, вы можете вызвать синхронизацию. Если вы вручную изменили атрибуты популяции до вызова синхронизации, вы можете заново передать данные популяции на бекенд для вычисления новых атрибутов. Подробнее о методах популяций и бекендов см. API-документацию Kaspersky Neuromorphic Platform.

Каждая популяция нейронов содержит следующие атрибуты:

- Базовые данные популяции.
- Тип нейронов, которые хранятся в популяции.

  Тип нейрона определяет набор атрибутов и функцию этого нейрона. Функция нейрона реализуется для конкретного типа нейрона на бекенде. При получении синаптического воздействия функция нейрона изменяет значения его атрибутов и возвращает состояние генерации спайка.
- Атрибуты нейронов популяции. Состояние нейронов в популяции.

Класс Projection

Шаблонный класс `Projection` реализует контейнер, который содержит синапсы одного из типов, поддерживаемых библиотекой характеристик синапсов.

Объект класса `Projection` представляет собой набор соединений нейронов пресинаптической и постсинаптической популяций. Конструирование проекции осуществляется с помощью генератора соединений.

С помощью класса `Projection` вы можете передавать данные проекции на бекенд и получать их от бекенда.

Kaspersky Neuromorphic Platform обеспечивает обмен данными между популяциями нейронов и проекциями синапсов через сообщения. Проекция синапсов получает сообщения со спайками от пресинаптической популяции и отправляет сообщения с синаптическими воздействиями в постсинаптическую популяцию.

Каждая проекция синапсов содержит следующие атрибуты:

- Базовые данные проекции.
- Уникальный идентификатор пресинаптической популяции.
- Уникальный идентификатор постсинаптической популяции.
- Значение типа `boolean` для обозначения возможности изменения веса синапса. Если изменение веса синапса заблокировано, то атрибут имеет значение `true`, иначе имеет значение `false`.
- Атрибуты группы синапсов.
- Значение типа `boolean` для обозначения актуальности реестра соединений. Если реестр соединений актуален, то атрибут имеет значение `true`, иначе имеет значение `false`.
- Атрибуты, используемые синапсами одного типа.

Класс Backend

Класс `Backend` является базовым для реализаций бекендов. Каждый бекенд реализуется в виде отдельной библиотеки, с помощью которой вы можете:

- запускать эмуляцию работы импульсной нейронной сети на различных устройствах;
- считывать выходные спайки, полученные от вычислителя;

- получать внутренние данные для мониторинга работы нейронной сети;
- выгружать внутренние данные мониторинга работы нейронной сети, полученные от вычислителя.

Каждый объект класса `Backend` содержит следующие атрибуты:

- Базовые данные бекенда.
- Шину сообщений. С помощью шины сообщений бекенд обеспечивает обмен сообщениями между объектами библиотеки поддержки бекендов, а также обмен сообщениями от модификаторов к нейронам и синапсам.
- Точку подключения.
- Статус инициализации бекенда.
- Статус исполнения нейронной сети на бекенде.
- Устройства бекенда.
- Номер шага.

Класс Device

Класс `Device` является базовым для классов устройств, поддерживаемых библиотекой устройств.

Наследники класса переопределяют методы, с помощью которых вы можете получить общую информацию об объектах классов-наследников. Например, вы можете получить информацию о типе или энергопотреблении конкретного устройства. Данные об устройствах класс `Device` получает от бекенда.

Каждый объект класса `Device` содержит базовые данные для конкретного устройства.

Класс MessageBus

Класс `MessageBus` реализует интерфейс для доступа к шине сообщений, которая обеспечивает прием, распределение и доставку сообщений между подключенными к ней объектами. Объект класса `MessageBus` предоставляет множество точек подключения, с помощью которых объекты библиотеки поддержки бекендов могут подписаться на сообщения или отправить их.

Класс `MessageBus` предоставляет методы, которые возвращают используемые точки подключения. Если вы удалите точку подключения, то передача сообщений с помощью этой точки подключения прекратится.

Использование шины сообщений позволяет реализовать принцип "издатель-подписчик". Издатель отправляет сообщения через посредника, в качестве которого выступает шина сообщений. Объект подписки фильтрует сообщения и отправляет их подписчикам. Подобный подход позволяет избежать создания большого количества IPC-каналов, поскольку издатель и подписчики не связаны напрямую.

Класс MessageEndpoint

Класс `MessageEndpoint` реализует интерфейс для доступа к точке подключения, которая обеспечивает подписку на сообщения и их отправку с помощью шины сообщений.

Объект класса `MessageEndpoint` хранит в себе набор объектов класса `Subscription`, которые используются для фильтрации сообщений. Шина сообщений может иметь несколько точек подключения.

С помощью методов класса `MessageEndpoint` вы можете настроить обмен сообщениями между объектами бекенда и фреймворка. Для передачи сообщений требуется предварительно добавить объект подписки в объект точки подключения.

Если вы удалите точку подключения, то связанные объекты подписки также будут удалены. Передача сообщений с помощью этой точки подключения прекратится.

Класс Subscription

Класс `Subscription` реализует функциональность подписки на сообщения. Объект подписки используется для хранения сообщений определенного типа до их запроса получателем. В объекте подписки определены получатели хранящихся сообщений.

Подписки хранятся в точках подключения к шине сообщений. Если объект точки подключения, который хранит объект подписки, будет удален, то объект подписки будет также удален. Передача сообщений, определенных этой подпиской, прекратится.

Каждый объект подписки содержит следующие атрибуты:

- Вектор сообщений для передачи получателю.
- Идентификатор получателя сообщений.
- Идентификаторы отправителей сообщений.

Пространство имен Messaging

Пространство имен `Messaging` реализует интерфейсы для доступа к сообщениям. Сообщение содержит данные, которыми обмениваются объекты бекенда, а также данные, отправляемые объектами, которые идентифицируются по UID. Обмен сообщениями обеспечивается с помощью шины сообщений `MessageBus`.

Каждое сообщение содержит следующие атрибуты:

- `MessageHeader` – заголовок сообщения, содержащий следующую информацию:
  - `sender_uid_` – идентификатор объекта, отправившего сообщение;
  - `send_time_` – время отправки сообщения.
- Тело сообщения, состоящее из совокупности передаваемых атрибутов. Состав передаваемых атрибутов зависит от типа сообщения.

С помощью сообщений происходит обновление пользовательских структур и синхронизация изменений на бекенде. Таблица ниже содержит описание поддерживаемых типов сообщений бекенда.

Тип сообщения	Описание	Структура сообщения

`SpikeMessage`	Сообщение от популяции нейронов к проекции синапсов, содержащее спайк. На сообщения, содержащие спайки, может быть подписано более одной проекции.	`header_` – заголовок сообщения, содержащий UID популяции, нейроны которой сгенерировали спайки, и время генерации спайков. `neuron_indexes_` – индексы нейронов популяции, сгенерировавших спайки.
`SynapticImpactMessage`	Сообщение от проекции синапсов к популяции нейронов, содержащее синаптическое воздействие.	`header_` – заголовок сообщения, содержащий UID проекции и время отправки сообщения. `postsynaptic_population_uid_` – UID постсинаптической популяции. `presynaptic_population_uid_` – UID пресинаптической популяции. `is_forcing_` – значение типа `boolean` для обозначения использования пластичности в проекции синапсов. Если в проекции синапсов используется синаптическая пластичность, то атрибут имеет значение `true`, иначе имеет значение `false`. `impacts_` – вектор, содержащий следующие атрибуты: • `connection_index_` – индекс соединения. • `impact_value_` – значение синаптического воздействия. • `synapse_type_` – тип синапса. • `presynaptic_neuron_index_` – индекс пресинаптического нейрона. • `postsynaptic_neuron_index_` – индекс постсинаптического нейрона, которому будет передано значение синаптического воздействия для изменения значений его атрибутов.

Пространство имен synapse_access

Пространство имен `synapse_access` реализует соединения и интерфейс для доступа к их реестру.

Структура Connection

Структура `Connection` определяет соединение синапса с пресинаптическим и постсинаптическим нейроном и включает в себя в следующие атрибуты:

- `from_` – индекс пресинаптического нейрона.
- `to_` – индекс постсинаптического нейрона.
- `index_` – индекс синапса.

Класс Index

Класс `Index` реализует реестр соединений. С помощью методов класса `Index` проекция может найти синапсы, связанные с пресинаптическими или постсинаптическими нейронами с заданными индексами. В свою очередь вы можете получить информацию о соединениях с помощью методов проекции.

Запись о соединении синапса с пресинаптическим и постсинаптическим нейронами в реестр выполняется проекцией. Обновление реестра зависит от поведения функции проекции и может выполняться автоматически после любого изменения проекции (например, удаления синапса из проекции) или в момент, когда проекция используется для вычисления.

Библиотека устройств

Библиотека устройств реализует и предоставляет набор интерфейсов, которые вы можете использовать для взаимодействия с физическими устройствами.

При запуске эмуляции импульсной нейронной сети библиотека устройств используется для получения информации об устройствах (например, параметров CPU), на которых работают реализованные бекенды. Информацию об устройстве (например, о потребляемой им мощности) вы можете получить с помощью методов объекта, который обеспечивает интерфейс к устройству.

Выбор устройства для эмуляции нейронной сети осуществляется с помощью методов бекенда.

В Kaspersky Neuromorphic Platform на текущий момент реализован интерфейс для центрального процессора.

Библиотека характеристик нейронов

Библиотека характеристик нейронов представляет собой набор атрибутов нейронов. Вы можете использовать атрибуты нейронов, представленных в библиотеке, например, при построении популяций.

На текущий момент библиотека поддерживает наборы свойств для следующих нейронов:

- Нейрон по модели `BLIFATNeuron` (далее также "BLIFAT-нейроны").

  Нейрон по модели `BLIFATNeuron` представляет собой нейрон-интегратор с утечкой и адаптивным порогом, генерирующих серию спайков (англ. Bursting Leaky Integrate-and-Fire with Adaptive Threshold Neuron). BLIFAT-нейрон характеризуется потенциалом мембраны и пороговым значением потенциала мембраны.

  Значение потенциала мембраны изменяется в соответствии со значением синаптического воздействия, поступающего от связанного синапса. Если потенциал мембраны нейрона достигает порогового значения, нейрон начинает генерировать серию спайков с определенным интервалом. При этом после генерации спайка потенциал мембраны стремится к базовому значению, а пороговое значение изменяется в соответствии с инкрементом.

  Если к нейрону не поступают сообщения от связанного синапса в течение определенного времени, потенциал мембраны и его пороговое значение экспоненциально стремятся к базовым значениям.

Вы можете указать базовое значения потенциала мембраны, базовое пороговое значение, значение инкремента, интервал времени, при достижении которого значения потенциала и порогового значения потенциала мембраны начинают стремиться к базовым, продолжительность серии спайков и интервал генерации спайков в атрибутах экземпляра нейрона.

- Нейрон по модели `SynapticResourceSTDPBLIFATNeuron`.

  Нейрон по модели `SynapticResourceSTDPBLIFATNeuron` представляет собой BLIFAT-нейрон, который характеризуется свободным синаптическим ресурсом, пороговым значением свободного синаптического ресурса и стабильностью.

  Если значение свободного синаптического ресурса по модулю превышает пороговое значение, то этот свободный синаптический ресурс распределяется поровну между всеми пластичными пресинаптическими связями нейрона. После этого значение свободного синаптического ресурса нейрона становится равным `0`. Если значение свободного синаптического ресурса по модулю меньше порогового значения, то нейрон присваивает себе синаптический ресурс от связанных синапсов до достижения порогового значения.

  Значение стабильности нейрона изменяется в соответствии с синапатической пластичностью, используемой синапсом, и влияет на ее значение пластичности.

Если требуется, вы можете реализовать и использовать собственные модели нейронов.

Библиотека характеристик синапсов

Библиотека характеристик синапсов представляет собой набор атрибутов синапсов. Вы можете использовать синапсы, представленные в библиотеке, например, при построении проекций.

На текущий момент библиотека поддерживает наборы свойств для следующих синапсов:

- Синапс по модели `DeltaSynapse` (далее также "дельта-синапс").

  После получения спайка от пресинаптической популяции синапс генерирует сообщение, содержащее значение синаптического воздействия. Проекция синапсов отправляет синаптическое воздействие популяции для изменения мембранного потенциала постсинаптического нейрона. При этом отправка сообщения проекцией происходит без временной задержки.

  Отправляемое синаптическое воздействие определяется значением веса дельта-синапса. Вы можете указать вес синапса в атрибутах его экземпляра.

- Синапс по модели `AdditiveSTDPDeltaSynapse`.

  Синапс по модели `AdditiveSTDPDeltaSynapse` представляет собой дельта-синапс с синаптической пластичностью, зависимой от времени получения спайка. Если спайк от пресинаптической популяции поступает в проекцию перед генерацией пресинаптического спайка постсинаптическим нейроном, значение веса синапса увеличивается. Если пресинаптический спайк поступает на синапс после генерации спайка постсинаптическим нейроном, значение веса синапса уменьшается.

  Вес синапса определяет значение синаптического воздействия, отправляемого в постсинаптическую популяцию.

  Изменение веса синапса рассчитывается по формуле, представленной на рисунке ниже, где:

  - *j* соответствует пресинаптическому нейрону.
  - *i* соответствует постсинаптическому нейрону.
  - *W* соответствует функции синаптической пластичности.
  - *n* соответствует порядковому номеру пресинаптического нейрона.

- f соответствует порядковому номеру постсинаптического нейрона.
- t_i^n соответствует времени получения синапсом спайков от пресинаптического нейрона.
- t_j^f соответствует времени генерации спайка постсинаптическим нейроном.
- Δw_j соответствует значению изменения веса синапса после получения спайка от пресинаптического нейрона.

$$\Delta w_j = \sum_{f=1}^{N} \sum_{n=1}^{N} W(t_i^n - t_j^f)$$

Формула для расчета изменения веса синапса по модели `AdditiveSTDPDeltaSynapse`

- Синапс по модели `SynapticResourceSTDPDeltaSynapse`.

  Синапс по модели `SynapticResourceSTDPDeltaSynapse` представляет собой дельта-синапс с синаптической пластичностью, зависимой от синаптического ресурса. В свою очередь значение синаптического ресурса зависит от используемой синаптической пластичности:

  - Хеббовская пластичность.

    Значение синаптического ресурса зависит от периода плотной последовательности спайков (далее также "ППС"). *ППС* – это последовательность генерируемых нейроном нефорсированных спайков, в которой все соседние спайки разделены временем не больше, чем *ISImax*, где *ISImax* – это максимальное допустимое расстояние между спайками в ППС. В свою очередь *периодом ППС* является интервал времени между моментом начала Хеббовской пластичности перед первым спайком в ППС и одним из следующих моментов времени: достижение максимально допустимого расстояния ISImax после последнего спайка в ППС, форсированное срабатывание или приход спайка на дофаминовый синапс.

    При получении хотя бы одного спайка в период ППС синаптический ресурс изменяется на величину текущей Хеббовской пластичности. Значение Хеббовской пластичности зависит от переменной стабильности нейрона и рассчитывается по формуле, представленной на рисунке ниже, где:

    - d_H соответствует Хеббовской пластичности.
    - *s* соответствует переменной стабильности нейрона. В момент первого спайка в период ППС значение стабильности меняется на некоторую константу. Если стабильность уже отрицательная, она не меняется в меньшую сторону.

$$d_H = \overline{d_H} \min(2^{-s}, 1)$$

Формула для расчета Хеббовской пластичности

  - Безусловная пластичность.

    При ненулевой безусловной пластичности значение синаптического ресурса уменьшается на значение пластичности при получении спайка. Значение безусловной пластичности зависит от переменной стабильности нейрона и рассчитывается по формуле, представленной на рисунке ниже, где:

    - d_H соответствует безусловной пластичности.
    - *s* соответствует переменной стабильности нейрона. В момент первого спайка в период ППС значение стабильности меняется на некоторую константу. Если стабильность уже отрицательная, она не меняется в меньшую сторону.

$$d_H = \overline{d_H} \min(2^{-s}, 1)$$

Формула для расчета безусловной пластичности

- Дофаминовая пластичность.

  Дофаминовая пластичность применима, когда сумма всех дофаминовых синапсов, получивших спайк в текущий шаг исполнения нейронной сети, не равна нулю. В случае дофаминовой пластичности значение синаптического ресурса изменяется по формуле, представленной на рисунке ниже, где:

  - *D* соответствует сумме дофаминовых синапсов.
  - *s* соответствует переменной стабильности нейрона.

$$D\min\left(2^{-s}, 1\right)$$

Формула для расчета синаптического ресурса при дофаминовой пластичности

В свою очередь сумма дофаминовых синапсов влияет на переменную стабильности нейрона. Если в момент форсированного срабатывания пришло *дофаминовое наказание* (сумма дофаминовых синапсов меньше нуля), то значение стабильности нейрона уменьшается на значение *rD*, где *r* соответствует некоторой константе, а *D* – дофаминовому наказанию.

Если в момент форсированного срабатывания пришло *дофаминовое вознаграждение* (сумма дофаминовых синапсов выше нуля), то значение стабильности нейрона увеличивается на значение *rD*, где *r* соответствует некоторой константе, а *D* – дофаминовому вознаграждению. Иначе значение стабильности изменяется на значение, представленное на рисунке ниже, где:

- *D* соответствует сумме дофаминовых синапсов.
- *ISImax* соответствует максимально допустимому расстоянию между спайками в ППС.
- t_{TSS} соответствует времени с первого спайка в последнем ППС.

$$D\max\left(2 - \frac{|t_{TSS} - ISI_{max}|}{ISI_{max}}, -1\right)$$

Формула для расчета изменения стабильности нейрона при дофаминовом вознаграждении

Вес синапса определяет значение синаптического воздействия, отправляемого в постсинаптическую популяцию и связан со значением синаптического ресурса формулой, представленной на рисунке ниже, где:

- *w* соответствует весу синапса.
- *W* соответствует синаптическому ресурсу.
- $\boldsymbol{w}_{\min}$ соответствует некоторому случайно заданному минимальному значению веса.
- $\boldsymbol{w}_{max}$ соответствует некоторому случайно заданному максимальному значению веса.

$$\boldsymbol{w} = \boldsymbol{w}_{\min} + \frac{(\boldsymbol{w}_{max} - \boldsymbol{w}_{min}) * max(W, 0)}{\boldsymbol{w}_{max} - \boldsymbol{w}_{min} + max(W, 0)}$$

Формула для расчета изменения веса в зависимости от синаптического ресурса

Если требуется, вы можете реализовать и использовать собственные модели синапсов.

Библиотека шаблонов

Библиотека шаблонов представляет собой набор шаблонов общего назначения. Шаблоны библиотеки используются, например, внутри методов бекенда, обеспечивающих интроспекцию.

Компоненты фреймворка для C++

Фреймворк Kaspersky Neuromorphic Platform, реализованный на языке программирования C++, представляет собой независимые блоки, которые обеспечивают управление бекендами, преобразуют представление нейронной сети в разные форматы и выбирают оптимальную нейронную сеть для запуска. Во фреймворке для C++ реализованы функции, не относящиеся непосредственно к эмуляции импульсных нейронных сетей.

С помощью фреймворка для C++ Kaspersky Neuromorphic Platform вы можете выполнять следующие задачи:

- Загружать структуру нейронной сети из внешних форматов.
- Динамически загружать различные бекенды.
- Выполнять эмуляции импульсных нейронных сетей на различных бекендах.
- Вводить и выводить данные нейронных сетей.
- Контролировать выполнение эмуляции.
- Реализовывать специальные алгоритмы.

Таблица ниже содержит описание объектов фреймворка для C++.

Объекты фреймворка для C++

Объект	Описание
BackendLoader	Класс, реализующий загрузчик динамических библиотек.
Coordinates	Набор классов, реализующих пространственные координаты нейрона.
input	Пространство имен, в котором реализованы интерфейсы для доступа к каналу ввода и преобразователю данных, поступающих из внешней среды.
MessageObserver	Класс, реализующий наблюдателя.
Model	Класс, реализующий модель.
ModelExecutor	Класс, реализующий исполнитель модели.
Network	Класс, реализующий объект нейронной сети.
output	Пространство имен, в котором реализованы интерфейсы для доступа к каналам вывода.

На рисунке ниже представлена схема взаимодействия объектов фреймворка для C++.

В Kaspersky Neuromorphic Platform объект класса `Network` может включать в себя набор координат (произвольных или задаваемых пользователем) из классов `Coordinates`. Объект класса `Network`, а также контейнеры каналов ввода и вывода, реализованных в пространствах имен `input` и `output` соответственно, хранятся в объекте класса `Model`. Каналы ввода и вывода используются для преобразования данных, поступающих в проекции нейронной сети, а также для преобразования сообщений популяций нейронной сети в один из предусмотренных форматов данных.

Вы можете исполнить объект класса `Model` с помощью метода `run()` объекта класса `ModelExecutor`. Объект класса `ModelExecutor` также содержит объект класса `MessageObserver`, который обрабатывает полученные сообщения в соответствии с заданной функцией, и экземпляр класса `BackendLoader`, с помощью методов которого вы можете загружать динамические библиотеки.

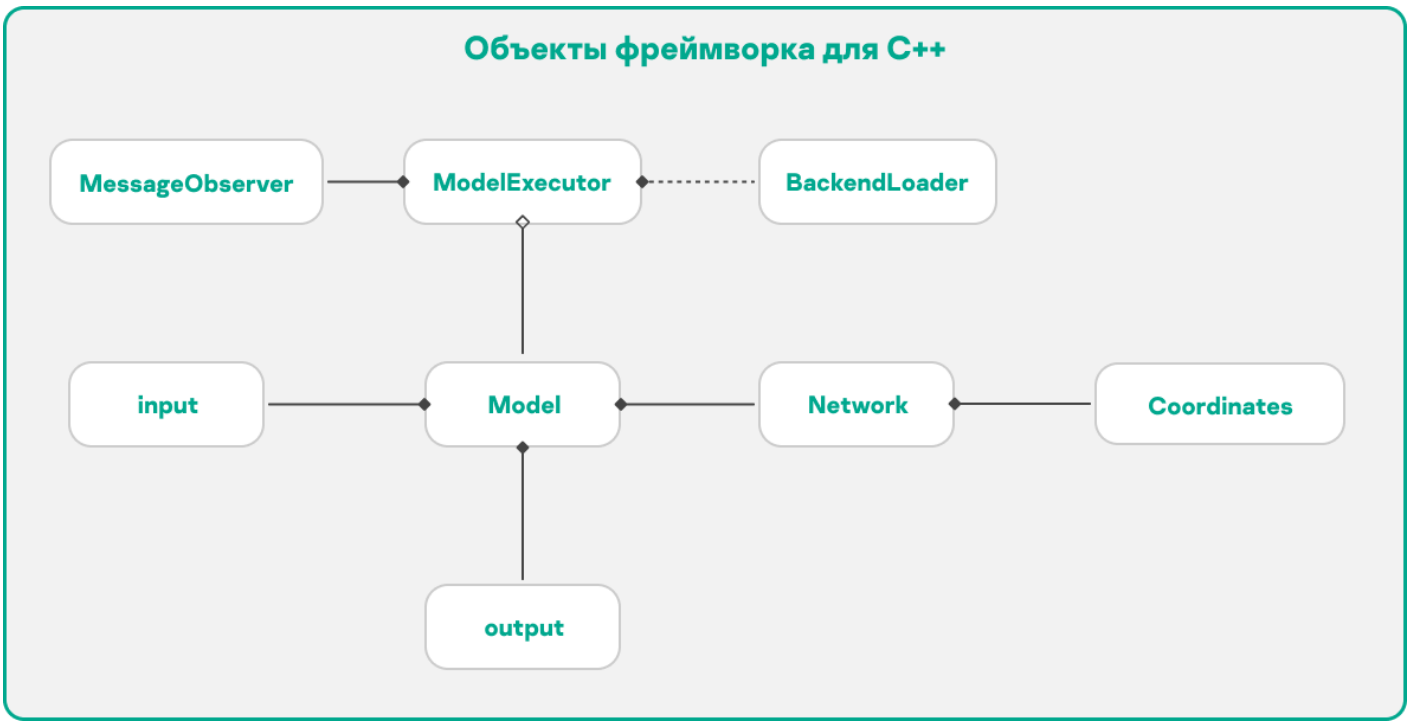


Схема взаимодействия объектов фреймворка для C++

Пространство имен input

В пространстве имен `input` реализованы интерфейсы для доступа к каналу ввода и преобразователю данных, поступающих из внешней среды. Данные могут поступать в Kaspersky Neuromorphic Platform как в синхронном (по запросу пользователя), так и в асинхронном режиме (по вызову обработчика).

Класс SequenceConverter

Класс `SequenceConverter` реализует преобразователь данных, который принимает данные из потока `istream` и делит поступившие данные на отрезки определенной длины. С помощью методов пространства имен `input` преобразователь данных приводит значения во входных отрезках к типу значений `boolean`. Если входное значение содержит спайк, метод возвращает `true`. Объект класса `SequenceConverter` возвращает список индексов входных значений, которые содержат спайки, в виде сообщения `SpikeMessage`.

Каждый объект класса `SequenceConverter` содержит следующие атрибуты:

- Поток данных `istream`, от которого поступают данные.
- Функцию приведения входных значений к типу значений `boolean`.
- Длину отрезка входных данных.

Класс IndexConverter

Класс `IndexConverter` реализует преобразователь данных, который принимает данные из потока `istream` и преобразует последовательность индексов поступивших значений, содержащих спайки, в сообщение `SpikeMessage`.

Каждый объект класса `IndexConverter` содержит следующие атрибуты:

- Поток данных `istream`, от которого поступают данные.
- Символ, с помощью которого индексы значений разделены в сообщении `SpikeMessage`.

Класс InputChannel

Класс `InputChannel` реализует канал ввода, который принимает сообщения `SpikeMessage` от преобразователя данных. С помощью методов класса `InputChannel` преобразованные данные могут быть переданы подписчикам через точки подключения при наличии подписки.

Каждый объект класса `InputChannel` содержит следующие атрибуты:

- Базовые данные канала ввода.
- Точку подключения, с помощью которой преобразованные данные отправляются подписчикам.
- Преобразователь данных в сообщения `SpikeMessage`.

Пространство имен output

В пространство имен `output` реализованы интерфейс для доступа к каналам вывода и преобразователи сообщений `SpikeMessage`. Kaspersky Neuromorphic Platform может передавать данные как в синхронном режиме (по запросу пользователя), так и в асинхронном режиме (по вызову обработчика).

Класс OutputChannel

Класс `OutputChannel` реализует канал вывода, который принимает набор сообщений, содержащих спайки, от связанной популяции за несколько тактов исполнения нейронной сети. Сообщения `SpikeMessage` могут быть преобразованы в один из следующих форматов данных, определяемых преобразователями сообщений:

- Вектор значений типа `boolean`.
- Вектор значений, в котором каждое значение соответствует количеству генерации спайков нейроном.
- Вектор нейронов, сгенерировавших спайки, из последовательности сообщений `SpikeMessage`.

Если эти форматы данных вам не подходят, вы можете реализовать собственную функцию, которая будет преобразовывать полученные сообщения в нужный вам формат данных.

Класс ConvertToSet

Класс `ConvertToSet` реализует преобразователь данных, который получает сообщения `SpikeMessage` из объекта класса `OutputChannel` и преобразует их в вектор нейронов, сгенерировавших спайки.

Каждый объект класса `ConvertToSet` содержит размер вектора нейронов.

Функтор converter_bitwise

Функтор `converter_bitwise` реализует преобразователь данных, который получает сообщения `SpikeMessage` и преобразует их в вектор значений типа `boolean`, в котором значение `true` соответствуют наличию спайка.

Функтор converter_count

Функтор `converter_count` реализует преобразователь данных, который получает сообщения `SpikeMessage` и преобразует их в вектор значений. Каждое значение вектор соответствует количеству генерации спайков нейроном.

Набор классов Coordinates

Набор классов `Coordinates` реализует координаты точки в n-мерном пространстве. Координаты хранятся в объекте тега.

Координаты могут быть использованы для отображения нейронной сети. С помощью координат вы также можете задать атрибуты нейронной сети. Например, на основе расстояния между координатами нейронов вы можете выявить задержку сигнала, который передается между этими нейронами.

Координаты могут изменяться произвольно. Во фреймворке для C++ реализованы декартова и полярная системы координат.

Класс MessageObserver

Класс `MessageObserver` реализует наблюдателя, который подписывается на некоторый набор сущностей и определенный тип сообщений. В конце каждого такта исполнения нейронной сети наблюдатель получает сообщения, пришедшие от сущностей по подписке, и обрабатывает их в соответствии с заданной функцией.

Каждый объект класса `MessageObserver` содержит следующие атрибуты:

- Точку подключения, которая обеспечивает подписку на сообщения сущностей;
- Функцию для обработки полученных сообщений;
- Идентификатор наблюдателя.

Класс Network

Класс `Network` реализует объект нейронной сети. Объект нейронной сети хранит базовые данные, популяции, проекции и стратегии генерации нейронов и синапсов, а также обеспечивает общий интерфейс для доступа к нейронам и синапсам нейронной сети. В нейронной сети все популяции связаны проекциями.

Класс содержит методы, с помощью которых вы можете добавлять и удалять популяции и проекции нейронной сети.

Класс Model

Класс `Model` реализует модель, связывающую и хранящую нейронную сеть и каналы ввода и вывода. Для исполнения модели на разных бекендах вы можете загрузить объект модели в объект класса `ModelExecutor` и вызвать метод исполнения модели `run()`.

С помощью методов класса `Model` вы можете управлять объектом нейронной сети.

Каждый объект класса `Model` содержит следующие атрибуты:

- Базовые данные модели.
- Нейронную сеть.
- Идентификаторы каналов ввода и вывода.

Класс ModelExecutor

Класс `ModelExecutor` реализует исполнитель моделей. С помощью класса `ModelExecutor` вы можете исполнить загруженную модель на разных бекендах.

Каждый объект класса `ModelExecutor` содержит следующие атрибуты:

- Базовые данные исполнителя модели.
- Загрузчик динамических библиотек.
- Экземпляр бекенда, на котором требуется исполнить модель.
- Модель для исполнения.
- Карту канала ввода, содержащую идентификатор, хешированный идентификатор и преобразователь входных данных.
- Каналы ввода и вывода.

Класс BackendLoader

Класс `BackendLoader` реализует загрузчик динамических библиотек, возвращающий указатель на экземпляр загруженного бекенда.

С помощью методов класса `BackendLoader` вы можете загружать динамические библиотеки и проверять, является ли загруженная библиотека библиотекой бекенда.

Класс `BackendLoader` реализован с помощью библиотеки Boost:DLL.

Компоненты платформы для использования на Python

Этот раздел содержит описание компонентов для использования Kaspersky Neuromorphic Platform на языке программирования Python.

Компоненты бекенда платформы для Python

Для реализации бекендов вы можете использовать следующие библиотеки платформы на языке программирования Python:

- *Библиотека поддержки бекендов*. Реализует основные объекты и шаблоны вычислителя.
- *Библиотека характеристик нейронов*. Содержит наборы свойств и типизацию популяций моделей нейронов.
- *Библиотека характеристик синапсов*. Содержит набор классов, определяющих поведение синапса.

На рисунке ниже представлена схема взаимодействия библиотек платформы.

В Kaspersky Neuromorphic Platform вычислитель хранит в себе экземпляры объектов библиотеки поддержки бекендов (например, проекции и популяции), необходимые для выполнения прикладных задач на вычислителе. Шаблоны популяций и проекций нейронной сети должны быть специализированы типами нейронов и синапсов, описанными в библиотеках характеристик нейронов и синапсов.

Библиотека поддержки бекендов

В библиотеке поддержки бекендов реализуются интерфейсы к объектам, необходимым для работы бекендов и выполнения прикладных задач. Библиотека содержит пространство имен `core`.

Таблица ниже содержит описание объектов библиотеки.

Объект	Описание
Backend	Базовый класс для конкретных реализаций бекендов.
BaseData	Структура, определяющая набор базовых данных.
continuously_uid_generator	Класс, реализующий генератор уникальных идентификаторов.
MessageBus	Класс, реализующий интерфейс к шине сообщений.
MessageEndpoint	Класс, реализующий интерфейс к точке подключения.
Messaging	Пространство имен, определяющее интерфейсы к сообщениям.
Population	Набор классов, реализующих контейнеры нейронов одной модели.
Projection	Набор классов, реализующих контейнеры синапсов одного типа.
Subscription	Набор классов, реализующих подписку на сообщения.
TagMap	Класс, реализующий интерфейс к словарю тегов.
UID	Класс, определяющий уникальные идентификаторы.
uid_hash	Класс, реализующий хеширования UID.

UUID	Класс, реализующий интерфейс к уникальному идентификатору библиотеки Boost:Uuid.

На рисунке ниже представлена схема взаимодействия объектов библиотеки.

В Kaspersky Neuromorphic Platform структура `BaseData` содержит базовую информацию об объекте и включает в себя `UID` и `TagMap`. Структура `BaseData` используется в классах `Projection`, `Population` и `Backend`, соответственно каждый из них обладает идентификатором `UID` и набором тегов и их значениями `TagMap`.

Классы `Population` и `Projection` принимают и отправляют сообщения из пространства имен `Messaging`, которые содержат в своем заголовке UID отправителя. Обмен сообщений осуществляется с помощью методов класса `Backend`, в котором хранятся классы `MessageBus` и `MessageEndpoint`. Шина сообщений `MessageBus` создает точки подключения `MessageEndpoint` и взаимодействует с ними в процессе обмена сообщениями. Каждая точка подключения `MessageEndpoint` содержит контейнер подписок класса `Subscription`.

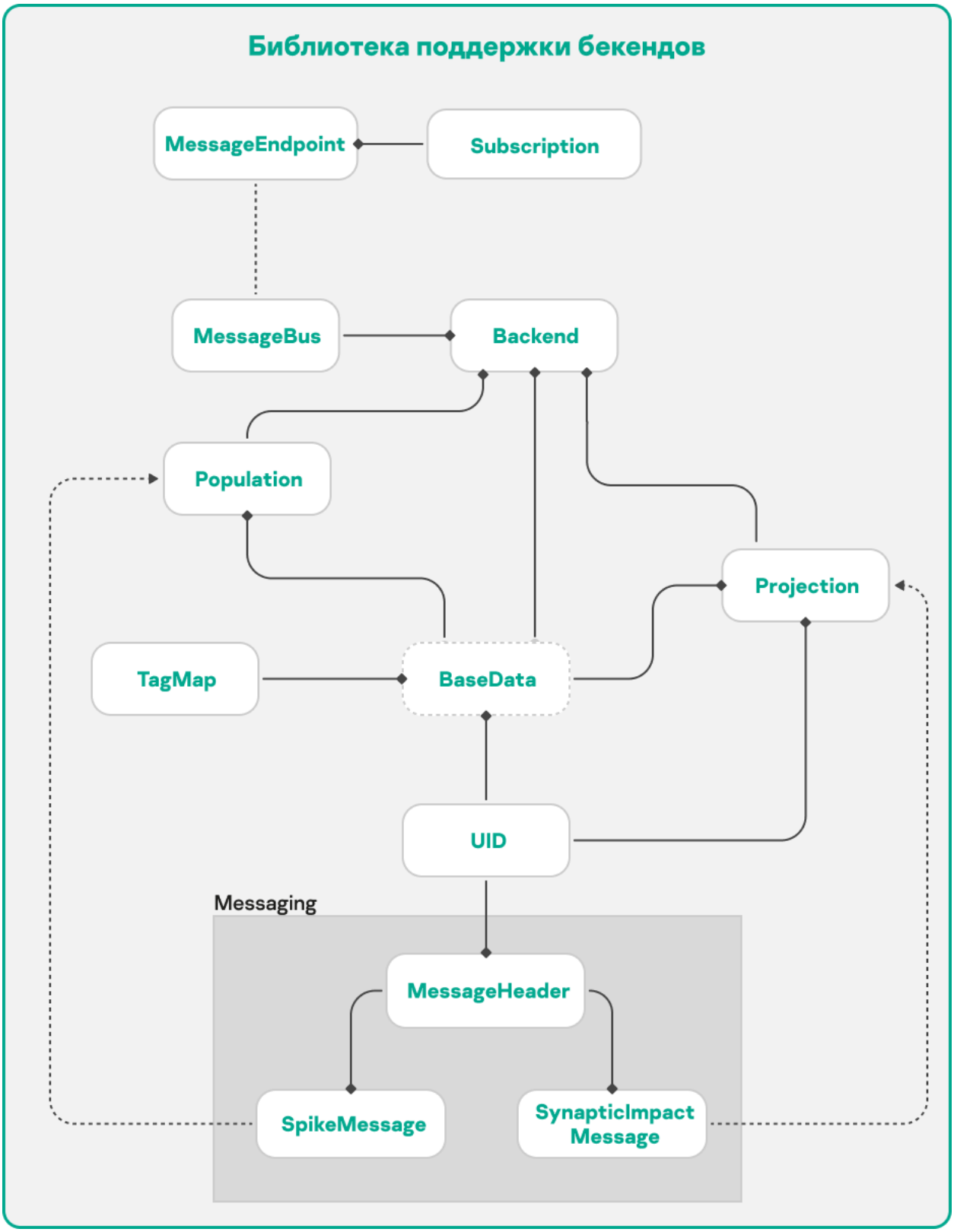


Схема взаимодействия объектов библиотеки поддержки бекендов

Класс UID

Объект класса `UID` является частью структуры `BaseData` и представляет собой уникальный идентификатор объекта.

Если требуется, вы можете сгенерировать случайный UID или создать уникальный идентификатор UUID. Класс UID также предоставляет логические операторы и операторы сравнения и присваивания.

Класс uid_hash

Класс `uid_hash` реализует функцию хеширования идентификаторов объектов.

Класс UUID

Класс `UUID` реализует интерфейс к уникальному идентификатору библиотеки Boost:Uuid. Каждый уникальный идентификатор имеет следующие свойства:

- Размер уникального идентификатора в байтах.
- Функцию, возвращающую значение `true`, если уникальный идентификатор состоит исключительно из нулей.
- Один из следующих вариантов идентификатора:
  - `NCS` – вариант, зарезервированный для обратной совместимости с форматом UUID Apollo Network Computing System 1.5.
  - `RFC_4122` – вариант, описанный в RFC 4122.
  - `MICROSOFT` – вариант, зарезервированный для обратной совместимости с ранними GUID Microsoft Windows.
  - `FUTURE` – вариант, зарезервированный для использования в будущем.

Класс continuously_uid_generator

Класс `continuously_uid_generator` реализует генератор уникальных идентификаторов (англ. UID). Идентификатор отображается в виде строки, разбитой на группы с помощью дефисов, и представляет собой 128-битное число. Идентификаторы генерируются последовательно с шагом, равным 1.

Вы можете использовать класс `continuously_uid_generator` для отладки платформы и исполнения нейронной сети.

Класс TagMap

Объект класса `TagMap` является составляющей частью структуры `BaseData` и представляет собой ассоциативный массив, включающий ноль или более записей типа "ключ-значение тега", где ключом является имя тега. Имена тегов имеют строковый тип и являются уникальными.

Структура BaseData

Структура `BaseData` определяет набор базовых данных, общих для нескольких объектов библиотеки поддержки бекендов и объектов фреймворка. Например, базовые данные `BaseData` используются в объектах классов `Backend`, `Population` и `Projection`.

Базовые данные `BaseData` включают в себя следующее:

- `uid_` – уникальный идентификатор объекта.
- `tags_` – ассоциативный массив `TagMap`, состоящий из тегов, используемых объектом, и их значений.

Набор классов Population

Набор классов `Population` реализует контейнеры, каждый из которых содержит нейроны, принадлежащие одной модели. При использовании набора классов `Population` вы можете указать один из типов нейронов, поддерживаемых библиотекой характеристик нейронов. Каждая популяция нейронов имеет уникальный идентификатор.

С помощью набора классов `Population` вы можете управлять нейронами в составе популяции. Вы также можете загрузить популяции на бекенд для вычисления нейронной сети. Полученные свойства популяций хранятся на бекенде. Чтобы передать новые свойства от бекенда в соответствующие объекты популяций, вы можете вызвать синхронизацию. Если вы вручную изменили свойства популяции до вызова синхронизации, вы можете заново загрузить популяцию на бекенд для вычисления нейронной сети. Подробнее о методах популяций и бекендов см. API-документацию Kaspersky Neuromorphic Platform.

Набор классов Projection

Набор классов `Projection` реализует контейнеры, каждый из которых содержит синапсы одного из типов, поддерживаемых библиотекой характеристик синапсов. Каждая проекция синапсов имеет уникальный идентификатор.

В классах проекций хранятся синапсы, представляющие собой соединения между нейронами пресинаптической и постсинаптической популяций. Идентификаторы этих популяций хранятся в проекции.

С помощью набора классов `Projection` вы можете управлять синапсами в составе проекции, а также передавать соединения между нейронами на бекенд и получать их от бекенда обратно.

Класс Backend

Класс `Backend` является базовым для реализаций бекендов. Каждый бекенд реализуется в виде отдельной библиотеки, с помощью которой вы можете:

- запускать эмуляцию работы импульсной нейронной сети на различных устройствах;
- получать сообщения, содержащие спайки и/или синаптические воздействия, для заданной точки подключения;
- получать внутренние данные для мониторинга работы нейронной сети;
- выгружать внутренние данные мониторинга работы нейронной сети, полученные от вычислителя.

Каждый объект класса `Backend` имеет следующие свойства:

- Уникальный идентификатор бекенда.

- Шину сообщений. С помощью шины сообщений бекенд обеспечивает обмен сообщениями между объектами библиотеки поддержки бекендов, а также обмен сообщениями от модификаторов к нейронам и синапсам.
- Точку подключения.
- Статус исполнения нейронной сети на бекенде.

Класс MessageBus

Класс `MessageBus` реализует интерфейс для доступа к шине сообщений, которая обеспечивает прием, распределение и доставку сообщений между подключенными к ней объектами. Объект класса `MessageBus` предоставляет множество точек подключения, с помощью которых объекты библиотеки поддержки бекендов могут подписаться на сообщения или отправить их.

Класс `MessageBus` предоставляет методы, с помощью которых вы можете создать точки подключения, а также передать сообщения через шину сообщений. Если вы удалите точку подключения, то передача сообщений с помощью этой точки подключения прекратится.

Использование шины сообщений позволяет реализовать принцип "издатель-подписчик". Издатель отправляет сообщения через посредника, в качестве которого выступает шина сообщений. Объект подписки фильтрует сообщения и отправляет их подписчикам. Подобный подход позволяет избежать создания большого количества IPC-каналов, поскольку издатель и подписчики не связаны напрямую.

Класс MessageEndpoint

Класс `MessageEndpoint` реализует интерфейс для доступа к точке подключения, которая обеспечивает подписку на сообщения и их отправку с помощью шины сообщений.

Объект класса `MessageEndpoint` хранит в себе идентификатор получателя сообщений и ключ класса подписки, который используется для фильтрации сообщений. Шина сообщений может иметь несколько точек подключения.

С помощью методов класса `MessageEndpoint` вы можете настроить обмен сообщениями между объектами бекенда и фреймворка с заданными идентификаторами. Для передачи сообщений требуется предварительно добавить ключ класса подписки в объект точки подключения.

Если вы удалите точку подключения, то связанные классы подписки также будут удалены. Передача сообщений с помощью этой точки подключения прекратится.

Набор классов Subscription

Набор классов `Subscription` реализует функциональность подписки на сообщения. Класс из группы подписок `Subscription` используется для хранения сообщений определенного типа до их запроса получателем. С помощью методов набора классов `Subscription` вы можете определить получателей хранящихся сообщений.

Ключи классов подписки хранятся в точках подключения к шине сообщений. Если объект точки подключения, который хранит ключ класса, будет удален, то соответствующий класс подписки будет также удален. Передача сообщений, определенных этим классом, прекратится.

Пространство имен Messaging

Пространство имен `Messaging` реализует интерфейсы для доступа к сообщениям. Сообщение содержит данные, которыми обмениваются объекты бекенда, а также данные, отправляемые объектами, которые идентифицируются по UID. Обмен сообщениями обеспечивается с помощью шины сообщений `MessageBus`.

Каждое сообщение содержит следующие свойства:

- `MessageHeader` – заголовок сообщения, содержащий следующую информацию:
  - `sender_uid` – идентификатор объекта, отправившего сообщение;
  - `send_time` – время отправки сообщения.
- Тело сообщения, состоящее из совокупности передаваемых атрибутов. Состав передаваемых свойств зависит от типа сообщения.

С помощью сообщений происходит обновление пользовательских структур и синхронизация изменений на бекенде. Таблица ниже содержит описание поддерживаемых типов сообщений бекенда.

Тип сообщения	Описание	Структура сообщения
`SpikeMessage`	Сообщение от популяции нейронов к проекции синапсов, содержащее спайк. На сообщения, содержащие спайки, может быть подписано более одной проекции.	`header` – заголовок сообщения, содержащий UID популяции, нейроны которой сгенерировали спайки, и время генерации спайков. `neuron_indexes` – индексы нейронов популяции, сгенерировавших спайки.
`SynapticImpactMessage`	Сообщение от проекции синапсов к популяции нейронов, содержащее синаптическое воздействие.	`header` – заголовок сообщения, содержащий UID проекции и время отправки сообщения. `postsynaptic_population_uid_` – UID постсинаптической популяции. `presynaptic_population_uid_` – UID пресинаптической популяции. `is_forcing_` – значение типа `boolean` для обозначения использования пластичности в проекции синапсов. Если в проекции синапсов используется синаптическая пластичность, то атрибут имеет значение `true`, иначе имеет значение `false`. `impacts_` – вектор, содержащий следующие свойства: • `connection_index_` – индекс соединения. • `impact_value_` – значение синаптического воздействия. • `synapse_type_` – тип синапса.

		• `presynaptic_neuron_index_` – индекс пресинаптического нейрона. • `postsynaptic_neuron_index_` – индекс постсинаптического нейрона, которому будет передано значение синаптического воздействия для изменения значений его свойств.

Библиотека характеристик нейронов

Библиотека характеристик нейронов представляет собой набор атрибутов нейронов. Вы можете использовать атрибуты нейронов, представленных в библиотеке, например, при построении популяций.

На текущий момент библиотека поддерживает наборы свойств для нейрона по модели `BLIFATNeuron` (далее также "BLIFAT-нейроны").

Нейрон по модели `BLIFATNeuron` представляет собой нейрон-интегратор с утечкой и адаптивным порогом, генерирующих серию спайков (англ. Bursting Leaky Integrate-and-Fire with Adaptive Threshold Neuron). BLIFAT-нейрон характеризуется потенциалом мембраны и пороговым значением потенциала мембраны.

Значение потенциала мембраны изменяется в соответствии со значением синаптического воздействия, поступающего от связанного синапса. Если потенциал мембраны нейрона достигает порогового значения, нейрон начинает генерировать серию спайков с определенным интервалом. При этом после генерации спайка потенциал мембраны стремится к базовому значению, а пороговое значение изменяется в соответствии с инкрементом.

Если к нейрону не поступают сообщения от связанного синапса в течение определенного времени, потенциал мембраны и его пороговое значение экспоненциально стремятся к базовым значениям.

Вы можете указать базовое значения потенциала мембраны, базовое пороговое значение, значение инкремента, интервал времени, при достижении которого значения потенциала и порогового значения потенциала мембраны начинают стремиться к базовым, продолжительность серии спайков и интервал генерации спайков в атрибутах экземпляра нейрона.

Если требуется, вы можете реализовать и использовать собственные модели нейронов.

Библиотека характеристик синапсов

Библиотека характеристик синапсов представляет собой набор атрибутов синапсов. Вы можете использовать синапсы, представленные в библиотеке, например, при построении проекций.

На текущий момент библиотека поддерживает наборы свойств для синапса по модели `DeltaSynapse` (далее также "дельта-синапс").

После получения спайка от пресинаптической популяции синапс генерирует сообщение, содержащее значение синаптического воздействия. Проекция синапсов отправляет синаптическое воздействие популяции для изменения мембранного потенциала постсинаптического нейрона. При этом отправка сообщения проекцией происходит без временной задержки.

Отправляемое синаптическое воздействие определяется значением веса дельта-синапса. Вы можете указать вес синапса в атрибутах его экземпляра.

Если требуется, вы можете реализовать и использовать собственные модели синапсов.

Компоненты Python-фреймворка

Фреймворк Kaspersky Neuromorphic Platform, реализованный на языке программирования Python, представляет собой независимые блоки, которые обеспечивают управление бекендами, преобразуют представление нейронной сети в разные форматы и выбирают оптимальную нейронную сеть для запуска. Во фреймворке для Python реализованы функции, не относящиеся непосредственно к эмуляции импульсных нейронных сетей.

С помощью фреймворка для Python Kaspersky Neuromorphic Platform вы можете динамически загружать различные бекенды.

Таблица ниже содержит описание объектов Python-фреймворка.

Объекты Python-фреймворка

Объект	Описание
ANN2SNN	Библиотека, реализующая тернарные слои нейронной сети.
BackendLoader	Класс, реализующий загрузчик динамических библиотек.

Библиотека ANN2SNN

Библиотека ANN2SNN предназначена для создания нейронных сетей, которые можно преобразовать в импульсные нейронные сети и разместить на нейроморфном процессоре Алтай-2 для их исполнения. Библиотека ANN2SNN реализована на языке программирования Python.

Для создания нейронных сетей необходимо использовать специальные тернарные слои, в которых учитываются ограничения, характерные для нейроморфного процессора Алтай-2. Тернарные слои, используемые для создания нейронной сети, соответствуют слоям библиотеки Keras, со следующими основными отличиями:

- Входными и выходными данными каждого из слоев нейронной сети являются бинарные тензоры соответствующей размерности.
- Все веса слоев нейронной сети находятся в диапазоне {-1,0,1}.
- Значение смещения в слое нейронной сети находится в диапазоне от -32 768 до 32 767.

При обучении нейронной сети, созданной с помощью тернарных слоев, стоит учитывать следующие особенности:

- В случае наличия небинаризованных данных для обучения и/или валидации необходимо сначала их привести к бинарному виду. Для это можно использовать алгоритмы кодирования и декодирования или создать нейросетевые обучаемые блоки, кодирующие и декодирующие данные.
- При обучении моделей бинарной классификации методом градиентного спуска для дифференцируемости функции потерь необходимо указать сигмоидную функцию (англ. sigmoid function) в качестве функции

активации выходного слоя. После проведения обучения нужно заменить сигмоидноую активацию выходного слоя на активацию с функцией Хевисайда (англ. Heaviside function).

- При обучении моделей многоклассовой классификации методом градиентного спуска для дифференцируемости функции потерь необходимо указать многомерную логистическую функцию (англ. softmax function) в качестве функции активации выходного слоя. При размещении такой нейронной сети на нейроморфном процессоре Алтай-2 выходной вектор будет автоматически преобразован к бинарному виду, реализующему индикатор максимальной компоненты значения выходного вектора.

Сериализация и десериализация нейронных сетей проводится стандартными методами, предоставляемыми библиотекой Keras, с указанием параметра `custom_objects` в виде словаря тернарных слоев и функции активации heaviside.

Во время размещения нейронной сети на нейроморфном процессоре нейронная сеть проверяется на соответствие ограничениям процессора. В результате размещения выдается информация о количестве ядер, необходимых для запуска нейронной сети на нейроморфном процессоре Алтай-2. Если нейронную сеть невозможно разместить на нейроморфном процессоре, выдается соответствующая ошибка. Вы можете провести примерную оценку количества ядер, необходимого для размещения нейронной сети на нейроморфном процессоре Алтай-2, по формуле, представленной на рисунке ниже, где:

- *x* соответствует необходимому количеству ядер. После расчета количества ядер по формуле округлите результат в большую сторону.
- *i* соответствует порядковому номеру слоя нейронной сети.
- *n* соответствует количеству слоев нейронной сети.
- *w* соответствует первой размерности тензора слоя нейронной сети.
- *h* соответствует второй размерности тензора слоя нейронной сети.
- *c* соответствует третьей размерности тензора слоя нейронной сети.

$$x = \sum_{i=1}^{n} \frac{w_i * h_i * c_i}{512}$$

Формула для примерной оценки необходимого количества ядер

Для нейронных сетей, созданных на базе библиотеки ANN2SNN, возможны следующие варианты инференса:

- Инференс за количество тиков, равное количеству слоев в нейронной сети. При этом варианте инференса за тик исполнения нейронной сети вычисляется только один слой нейронной сети, а обработанные данные передаются в последующий слой. Время инференса прямо пропорционально количеству слоев нейронной сети.
- Инференс за один тик. При этом варианте инференса каждый слой нейронной сети получает данные от предыдущего слоя и обрабатывает их. Время инференса составляет около 1,5 мс.

Тернарные слои библиотеки ANN2SNN

В библиотеке ANN2SNN поддерживаются следующие тернарные слои:

- **Слой TernaryDense**

Класс, реализующий полносвязный слой, аналогом которого является слой `Dense` библиотеки Keras. При работе с полносвязным слоем рекомендуется указать количество нейронов слоя (параметр `units`) равным или меньше 512. Если у нейрона слоя более 512 связей, то такой нейрон будет представлен в виде нескольких нейронов на нейроморфном процессоре Алтай-2.

- **Слой TernaryConv1D**

  Класс, реализующий одномерный сверточный слой, аналогом которого является слой `Conv1D` библиотеки Keras. При работе с этим слоем рекомендуется установить следующие ограничения:

  - Размер ядра (параметр `kernel_size`) – не более 20.
  - Количество фильтров (параметр `filters`) – не более 64.

- **Слой TernaryConv2D**

  Класс, реализующий двумерный сверточный слой, аналогом которого является слой `Conv2D` библиотеки Keras. При работе с этим слоем рекомендуется следовать следующим ограничениям:

  - Не использовать ядра большого размера (параметр `kernel_size`). Оптимальными считаются ядра с размерами [2, 2], [3, 3] и [4, 4].
  - Ограничить размер входных изображений до 128x128 и число каналов до 10.
  - Ограничить количество фильтров (параметр `filters`) до 32.

- **Слой TernaryLocallyConnected1D**

  Класс, реализующий одномерный локально-связный слой, аналогом которого является слой `LocallyConnected1D` библиотеки Keras. При работе с этим слоем рекомендуется установить следующие ограничения:

  - Размер ядра (параметр `kernel_size`) – не более 20.
  - Количество фильтров (параметр `filters`) – не более 64.

- **Слой TernaryLocallyConnected2D**

  Класс, реализующий двумерный локально-связный слой, аналогом которого является слой `LocallyConnected2D` библиотеки Keras. При работе с этим слоем рекомендуется следовать следующим ограничениям:

  - Не использовать ядра большого размера (параметр `kernel_size`). Оптимальными считаются ядра с размерами [2, 2], [3, 3] и [4, 4].
  - Ограничить размер входных изображений до 128x128 и число каналов до 10.
  - Ограничить количество фильтров (параметр `filters`) до 32.

- **Слой TernaryDepthwiseConv1D**

Класс, реализующий одномерный сверточный слой, аналогом которого является слой `DepthwiseConv1D` библиотеки Keras. При работе с этим слоем рекомендуется установить следующие ограничения:

- Размер ядра (параметр `kernel_size`) – не более 20.
- Количество фильтров (параметр `filters`) – не более 64.

- **Слой TernaryDepthwiseConv2D**

  Класс, реализующий двумерный сверточный слой, аналогом которого является слой `DepthwiseConv2D` библиотеки Keras. При работе с этим слоем рекомендуется следовать следующим ограничениям:

  - Не использовать ядра большого размера (параметр `kernel_size`). Оптимальными считаются ядра с размерами [2, 2], [3, 3] и [4, 4].
  - Ограничить размер входных изображений до 128x128 и число каналов до 10.
  - Ограничить количество фильтров (параметр `filters`) до 32.

- **Слой MaxPool1D**

  Поддерживается стандартная реализация соответствующего слоя библиотеки Keras. Для уменьшения количества оборудования нейроморфного процессора, необходимого для исполнения нейронной сети, рекомендуется отказаться от использования этого слоя путем увеличения параметра `stride` в сверточных слоях.

- **Слой MaxPool2D**

  Поддерживается стандартная реализация соответствующего слоя библиотеки Keras. Для уменьшения количества оборудования нейроморфного процессора, необходимого для исполнения нейронной сети, рекомендуется отказаться от использования этого слоя путем увеличения параметра `stride` в сверточных слоях.

- **Слой BatchNormalization**

  Поддерживается стандартная реализация нормирующего слоя `BatchNormalization` библиотеки Keras. Параметр масштабирования `scale` не поддерживается и должен быть определен как `false`.

  Слой `BatchNormalization` должен идти после слоя `Activation`.

- **Слой InputLayer**

  Поддерживается стандартная реализация входного слоя `InputLayer` библиотеки Keras. Входные данные должны иметь бинарный тип.

- **Слой Flatten**

Поддерживается стандартная реализация соответствующего слоя библиотеки Keras.

- Слой Reshape

  Поддерживается стандартная реализация соответствующего слоя библиотеки Keras.

- Слой Concatenate

  Поддерживается стандартная реализация соответствующего слоя библиотеки Keras.

Для размещения нейронной сети на нейроморфном процессоре Алтай-2 в качестве функции активации каждого слоя необходимо указать функцию `heaviside`. При обучении нейронной сети методом градиентного спуска используется кусочно-линейная аппроксимация сигмоидной функции (англ. `hard sigmoid function`).

Класс BackendLoader

Класс `BackendLoader` реализует загрузчик динамических библиотек, возвращающий указатель на экземпляр загруженного бекенда.

С помощью методов класса `BackendLoader` вы можете загружать динамические библиотеки и проверять, является ли загруженная библиотека библиотекой бекенда.

Сторонние библиотеки и используемые форматы

Этот раздел содержит информацию о сторонних библиотеках, форматах данных и структуры нейронной сети, используемых Kaspersky Neuromorphic Platform.

Сторонние библиотеки

Kaspersky Neuromorphic Platform использует следующие библиотеки:

- `Boost`. Библиотеки используются для реализации классов и структур (например, `UID`, `BackendLoader`).
- `FlatBuffers`. Библиотека используется для сериализации в Kaspersky Neuromorphic Platform.
- `GoogleTest`. Библиотека используется для тестов платформы.
- `Intel PCM`. Библиотека используется для получения от CPU метаданных устройства (например, данные об энергопотреблении).
- `spdlog`. Библиотека используется для логирования работы Kaspersky Neuromorphic Platform.

Форматы данных, обрабатываемые платформой

Для хранения входных и выходных данных в виде временного ряда используется базовый формат Kaspersky Neuromorphic Platform. Файл базового формата содержит следующую информацию:

- Заголовок файла, содержащий следующую информацию:
  - версия формата файла;
  - массив идентификаторов UID отправителей данных для каждого шага исполнения нейронной сети (опционально);
  - массив тегов.
- Входные или выходные данные представляют собой временной ряд, состоящий из списка структур со следующей информацией:
  - шаг исполнения нейронной сети, на котором были сгенерированы спайки;
  - индексы нейронов, сгенерировавших спайки на этой шаге.

С помощью методов Kaspersky Neuromorphic Platform вы можете преобразовать файлы базового формата в сообщения `SpikeMessage`.

Сообщения формата KNP сериализуются в следующие нотации:

- HDF5 – основная нотация для хранения данных в бинарном виде;
- JSON – нотация объектов JavaScript, которая может быть использована для отладки работы нейронной сети.

Помимо базового формата входные данные также могут храниться и быть переданы в виде CSV-файлов, в которых содержатся индексы нейронов, сгенерировавших спайки. CSV-файлы с входными данными могут быть переданы в объекты класса `IndexConverter` для их дальнейшей обработки в каналах ввода.

Форматы описания нейронной сети

В Kaspersky Neuromorphic Platform используется формат SONATA для описания топологии нейронной сети.

В формате SONATA нейронная сеть представляется в виде графа, состоящего из вершин, соответствующих нейронам, и ребер, соединяющих вершины графа и соответствующих синапсам. Вершины графа могут быть объединены в несколько популяций нейронов, которые связывается между собой проекцией синапсов (совокупностью ребер графа). Нейронам и синапсам присваиваются типы соответствующих популяций и проекций. Параметры, назначенные типам нейронов или синапсов, автоматически присваиваются всем нейронам или синапсам этих типов. Кроме того, нейронам, синапсам и их типам могут быть назначены атрибуты, которые определяют различные аспекты нейронной сети (например, положение, тип, параметры модели).

Популяция нейронов состоит из одной или нескольких групп нейронов с единообразной табличной структурой. Популяции сериализуются в HDF5-файлы и связываются между собой CSV-файлом, описывающим типы нейронов и атрибуты, применимые к этим типам. Группы нейронов представляются в виде групп HDF5-файлов, содержащих наборы данных, длины которых соответствуют количеству нейронов в группах. Данные в HDF5-файлах хранятся в бинарном виде.

Аналогично нейронам, синапсы определяются в проекциях, хранящихся в HDF5-файлах. В этих файлах описываются атрибуты для каждого синапса. Проекция синапсов состоит из одной или нескольких групп синапсов с единообразной табличной структурой. Каждый HDF5-файл связан с CSV-файлом, в котором определены типы синапсов и атрибуты, применимые к синапсам этих типов.

Примеры структур HDF5-файлов:

- **Структура файла, описывающего группы нейронов**

Ниже приведен пример структуры HDF5-файла, описывающего группы нейронов.

```
< имя_файла >.h5
    nodes
        < название_популяции >
            node_type_id
            node_id
            node_group_id
            node_group_index
            < группа_нейронов >
                < атрибут_группы >
                ...
                dynamics_params
                    < динамический_атрибут >
                    ...
```

В таблице ниже приведены элементы данных в файле, описывающем группы нейронов.

Элементы данных файла, описывающего группы нейронов

Элемент данных	Описание
< имя_файла >.h5	Имя файла, описывающего группы нейронов.
nodes	Обязательный элемент для определения групп.
< название_популяции >	Обязательный элемент, содержащий название популяции. Популяций может быть несколько.
node_type_id	Обязательный элемент, содержащий набор данных с массивом идентификаторов типов нейронов популяции.
node_id	Обязательный элемент, содержащий набор данных с индексами нейронов популяции.
node_group_id	Обязательный элемент, содержащий набор данных с индексами групп, к которым относятся нейроны популяции.
node_group_index	Обязательный элемент. содержащий набор данных с индексами нейронов в соответствующих группах.
< группа_нейронов >	Обязательный элемент, содержащий идентификатор группы нейронов.
< атрибут_группы >	Один или несколько атрибутов группы. Атрибуты не являются обязательными и зависят от типа нейрона. Каждый из атрибутов представляет собой набор данных.
dynamics_params	Группа необязательных динамических атрибутов.
< динамический_атрибут >	Один или несколько динамических атрибутов. Каждый из атрибутов представляет собой набор данных со значениями атрибутов для каждого нейрона в группе.

- Структура файла, описывающего группы синапсов

Ниже приведен пример структуры HDF5-файла, описывающего группы синапсов.

```
< имя_файла >.h5
    edges
        < название_проекции >
            source_node_id
            edge_group_id
            edge_group_index
            target_node_id
            edge_type_id
            indices
                source_to_target
                    node_id_to_range
                    range_to_edge_id
                target_to_source
                    node_id_to_range
                    range_to_edge_id
            < группа_синапсов >
                delay
                syn_weight
                dynamics_params
```

В таблице ниже приведены элементы данных в файле, описывающем группы синапсов.

Элементы данных файла, описывающего группы синапсов

Элемент данных	Описание
< имя_файла >.h5	Имя файла, описывающего группы синапсов.
edges	Обязательный элемент для определения групп.
< название_проекции >	Обязательный элемент, содержащий название проекции. Проекций может быть несколько.
source_node_id	Обязательный элемент, содержащий набор данных с идентификаторами пресинаптических нейронов для каждого синапса проекции.
edge_group_id	Обязательный элемент, содержащий набор данных с идентификаторами групп, к которым относятся синапсы проекции.
edge_group_index	Обязательный элемент, содержащий набор данных с индексами синапсов в соответствующих группах.
target_node_id	Обязательный элемент, содержащий набор данных с идентификаторами постсинаптических нейронов для каждого синапса проекции.
edge_type_id	Обязательный элемент, содержащий набор данных с идентификаторами типа каждого синапса проекции.
indices	Необязательный элемент для индексации синапсов по пресинаптическим и постсинаптическим нейронам.
source_to_target	Элемент, описывающий синапсы по пресинаптическим нейронам.
target_to_source	Элемент, описывающий синапсы по постсинаптическим нейронам.
node_id_to_range	Элемент, содержащий набор данных с идентификаторами пресинаптических нейронов для каждого синапса проекции.
range_to_edge_id	Элемент. содержащий набор данных с идентификаторами постсинаптических нейронов для каждого синапса проекции.

< группа_синапсов >	Обязательный элемент, содержащий идентификатор группы нейронов.
delay	Необязательный элемент, содержащий набор данных со значениями задержек синапсов группы.
syn_weight	Необязательный элемент, содержащий набор данных со значениями весов синапсов группы.
dynamics_params	Необязательные динамические атрибуты.

Параметры конфигурации нейронной сети прописываются в JSON-файлах. Подробнее о формате SONATA, см. документацию SONATA.

Установка и удаление платформы

Этот раздел содержит пошаговые инструкции по установке и удалению Kaspersky Neuromorphic Platform.

Для разработки прикладных решений вы можете установить платформу следующими способами:

- Установить deb-пакеты.
- Установить пакеты для разработки прикладных решений на языке программирования Python.

Вы также можете загрузить архив с исходным кодом платформы и собрать проект для разработки прикладных решений на C++.

Установка deb-пакетов

Раздел находится в разработке.

Установка пакетов для разработки на языке программирования Python

Раздел находится в разработке.

Загрузка и распаковка архива с исходным кодом платформы

Перед сборкой проекта для разработки прикладных решений необходимо скачать и распаковать архив с исходным кодом платформы.

Чтобы загрузить и распаковать архив с исходным кодом платформы:

1. Скачайте архив с исходным кодом с репозитория платформы.
2. Распакуйте в рабочую директорию архив платформы.

   Например, вы можете распаковать архив платформы с помощью следующей команды:

   ```
   tar xf <путь к архиву платформы> --directory <путь к рабочей директории>
   ```

Сборка проекта для разработки решений на C++

После загрузки и распаковки архива с исходным кодом платформы вы можете собрать проект для разработки прикладных решений на языке программирования C++ с помощью Kaspersky Neuromorphic Platform. Для этого необходимо указать параметры сборки проекта в корневом файле скрипта сборки CMakeLists.txt.

Корневой файл CMakeLists.txt должен содержать следующие команды:

- `cmake_minimum_required(VERSION 3.25)` – указание минимальной поддерживаемой версии CMake.

  Для сборки проекта на базе Kaspersky Neuromorphic Platform требуется CMake версии не ниже 3.25.

- `project(<название проекта>)` – указание параметров проекта.
- `set(CMAKE_CXX_STANDARD 17)` – указание C++ 17 в качестве используемого языкового стандарта.
- `add_subdirectory(<имя директории с исходным кодом платформы>)` – указание директории, включающая программу, сборку которых требуется выполнить.
- `add_executable("${PROJECT_NAME}" <файл программы>)` – указание исполняемого файла.
- `target_link_libraries("${PROJECT_NAME}" PRIVATE <подключаемые библиотеки платформы>)` – определение библиотек, которые будут подключены к вашему проекту.

Подробнее о сборке проектов с помощью CMake см. документацию CMake.

Чтобы собрать проект для разработки прикладных решений на C++:

1. Перейдите в рабочую директорию.
2. В рабочей директории создайте файл скрипта сборки CMakeLists.txt.
3. Откройте созданный файл в любом текстовом редакторе и укажите параметры сборки проекта.

   Примеры файла CMakeLists.txt:

   - **Пример файла CMakeLists.txt при загрузке динамических библиотек бекенда**

     CMakeLists.txt

     ```
     cmake_minimum_required(VERSION 3.25)
     // simple-network – название проекта
     project(simple-network)

     set(CMAKE_CXX_STANDARD 17)

     // Запускает CMake в директории KNP и добавляет все проекты из директории в
     сборку
     add_subdirectory(KNP)

     // Добавляет бинарный файл с исходным кодом в файл программы main.cpp
     add_executable("${PROJECT_NAME}" main.cpp)
     // Подключает фреймворк к проекту simple-network
     target_link_libraries("${PROJECT_NAME}" PRIVATE KNP::BaseFramework::Core)
     ```

   - **Пример файла CMakeLists.txt без загрузки динамических библиотек бекенда**

При сборке проекта без загрузки динамических библиотек требуется подключить библиотеки нужных бекендов с помощью метода `target_link_libraries`. Перечислите необходимые библиотеки через пробел как в приведенном примере.

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.25)
// digits-recognition – название проекта
project(digits-recognition)

set(CMAKE_CXX_STANDARD 17)

// Запускает CMake в директории KNP и добавляет все проекты из директории в сборку
add_subdirectory(KNP)

// Добавляет бинарный файл с исходным кодом в файл программы main.cpp
add_executable("${PROJECT_NAME}" main.cpp)
// Подключает фреймворк и библиотеку однопоточного бекенда для CPU к проекту digits-recognition.
target_link_libraries("${PROJECT_NAME}" PRIVATE KNP::BaseFramework::Core KNP::Backends::CPUSingleThreaded)
```

4. Сохраните изменения в файле CMakeLists.txt.

Удаление платформы

В зависимости от способа установки вы можете удалить платформу следующими способами:

- Удалить deb-пакеты.
- Удалить пакеты для разработки прикладных решений на языке программирования Python.
- Удалить директорию, в которую был распакован архив с исходным кодом платформы.

Участие в разработке платформы

Вы можете участвовать в разработке Kaspersky Neuromorphic Platform. Для этого необходимо получить исходный код с репозитория платформы.

Чтобы участвовать в разработке платформы:

1. Откройте терминал и с помощью команды `cd` перейдите в рабочую директорию.

   Вы можете использовать любой другой инструмент для работы с Git.

2. Для получения исходного кода платформы требуется выполнить следующие действия:

   a. В командной строке выполните следующую команду для загрузки репозитория платформы на устройство:

   ```
   git clone --recurse-submodules --remote-submodules <репозиторий платформы>
   ```

   где:

   - `--recurse-submodules` означает, что каждый подмодуль в репозитории платформы, включая вложенные подмодули, будет автоматически инициализирован и обновлен.
   - `--remote-submodules` означает, что для обновления подмодуля все клонированные подмодули будут использовать статус ветки его удаленного отслеживания.
   - `<репозиторий платформы>` соответствует веб-адресу репозитория платформы, полученному от сотрудников "Лаборатории Касперского".

   b. Настройте зависимости для работы с исходным кодом платформы.

3. Если требуется, загрузите удаленный репозиторий платформы в ваш локальный репозиторий. Для этого вы можете использовать команду `git pull`.

4. В командной строке выполните следующую команду для создания ветки, в которой вы будете вносить изменения в код платформы:

   ```
   git checkout -b <название ветки>
   ```

5. Внесите изменения в код платформы.

6. Зафиксируйте изменения.

   Например, вы можете использовать следующие команды для публикации изменений:

   ```
   git add <имя файла для публикации>
   git commit -m "<сообщение для публикации>"
   git push origin <название ветки>
   ```

7. Опубликуйте изменения в репозитории платформы.

Лицензирование платформы

Условия использования Kaspersky Neuromorphic Platform – это обязательное соглашение между вами и АО "Лаборатория Касперского", в котором изложены условия, на которых вы можете пользоваться платформой.

> Внимательно ознакомьтесь с Условиями использования перед началом работы с Kaspersky Neuromorphic Platform.

Вы можете ознакомиться с Условиями использования, прочитав документ LICENSE.txt, расположенном в репозитории платформы.

Если вы не согласны с Условиями использования, вы должны удалить и не использовать Kaspersky Neuromorphic Platform.

О предоставлении данных

Kaspersky Neuromorphic Platform не запрашивает, не хранит и не обрабатывает информацию, относящуюся к персональным данным.

Решение типовых задач на C++

Этот раздел содержит описание типовых пользовательских задач, выполняемых на языке программирования C++.

Добавление нового типа нейрона

На текущий момент Kaspersky Neuromorphic Platform поддерживает наборы свойств и типизацию популяций по моделям нейронов `BLIFATNeuron` и `SynapticResourceSTDPBLIFATNeuron`. Если требуется, вы можете добавить новый тип нейрона.

Вы можете использовать эту инструкцию при работе с исходным кодом платформы.

Чтобы добавить новый тип нейрона:

1. В директории `neuron-trait-library/include/knp/neuron-traits/` создайте заголовочный файл для нового типа нейрона.

2. В созданном заголовочном файле определите структуру типа нейрона и его шаблонные свойства (например, `default_values`, `neuron_parameters`).

   Пример определения BLIFAT-нейрона:

   neuron-trait-library/include/knp/neuron-traits/blifat.h

```
#include "type_traits.h"

namespace knp::neuron_traits

{
struct BLIFATNeuron;

template <>
struct default_values<BLIFATNeuron>

{
    // Задает количество шагов нейронной сети с шага последнего спайка по
умолчанию.
    constexpr static std::size_t n_time_steps_since_last_firing_ =
std::numeric_limits<std::size_t>::infinity();
    // Задает значение по умолчанию, к которому стремится мембранный потенциал для
тормозных синапсов, основанных на проводимости.
    constexpr static double reverse_inhibitory_potential = -0.3;
    // Задает значение по умолчанию, к которому стремится мембранный потенциал для
тормозных синапсов, основанных на токе.
    constexpr static double min_potential = -1.0e9;
    ...
};

template <>
struct neuron_parameters<BLIFATNeuron>

{
    // Задает количество шагов нейронной сети с шага последнего спайка.
    std::size_t n_time_steps_since_last_firing_ =
```

```
default_values<BLIFATNeuron>::n_time_steps_since_last_firing_;
    // Задает пороговое значение мембранного потенциала нейрона.
    double activation_threshold_ =
default_values<BLIFATNeuron>::activation_threshold_;
    // Задает динамическое пороговое значение мембранного потенциала, после
достижения которого нейрон генерирует спайк.
    double dynamic_threshold_ = default_values<BLIFATNeuron>::dynamic_threshold_;
    ...
};

}  // namespace knp::neuron_traits
```

3. Добавьте созданный тип нейрона в список нейронов, определенных в заголовочном файле `neuron-trait-library/include/knp/neuron-traits/all_traits.h`.

   Пример добавления BLIFAT-нейрона в список нейронов:

   `neuron-trait-library/include/knp/neuron-traits/all_traits.h`

   ```
   #include "blifat.h"

   namespace knp::neuron_traits

   {
   // Список типов нейронов, разделенных запятыми.
   #define ALL_NEURONS BLIFATNeuron
   ...
   }  // namespace knp::neuron_traits
   ```

4. Реализуйте в нужных бекендах перегруженный метод запуска.

   Например, для бекенда CPU реализуйте перегруженный метод запуска расчета популяций.

Добавление нового типа синапса

На текущий момент Kaspersky Neuromorphic Platform поддерживает наборы свойств и типизацию проекций по моделям синапсов `DeltaSynapse`, `AdditiveSTDPDeltaSynapse` и `SynapticResourceSTDPDeltaSynapse`. Если требуется, вы можете добавить новый тип синапса.

> Вы можете использовать эту инструкцию при работе с исходным кодом платформы.

Чтобы добавить новый тип синапса:

1. В директории `synapse-traits-library/include/knp/synapse-traits/` создайте заголовочный файл для нового типа синапса.

2. В созданном заголовочном файле определите структуру типа синапса и его шаблонные свойства (например, `synapse_parameters`).

   Пример определения дельта-синапса:

   `synapse-traits-library/include/knp/synapse-traits/delta.h`

```
#include "type_traits.h"

namespace knp::synapse_traits

{
struct DeltaSynapse;


template <>
struct default_values<DeltaSynapse>

{
    // Задает значение веса синапса по умолчанию.
    constexpr static float weight_ = 0.0F;
    // Задает значение задержки синапса по умолчанию.
    constexpr static uint32_t delay_ = 1;
    // Задает тип синапса по умолчанию.
    constexpr static OutputType output_type_ = OutputType::EXCITATORY;
};


template <>
struct synapse_parameters<DeltaSynapse>

{
    // Задает атрибуты синапса.
    synapse_parameters() : weight_(0.0F), delay_(1),
output_type_(knp::synapse_traits::OutputType::EXCITATORY) {}

    synapse_parameters(float weight, uint32_t delay,
knp::synapse_traits::OutputType type)
        : weight_(weight), delay_(delay), output_type_(type)

    // Вес синапса
    float weight_;
    // Задержка синапса
    std::size_t delay_;
    // Тип синапса на выходе
    knp::synapse_traits::OutputType output_type_;
};

}  // namespace knp::synapse_traits
```

3. Добавьте созданный тип синапса в список синапсов, определенных в заголовочном файле `synapse-traits-library/include/knp/synapse-traits/all_traits.h`.

   Пример добавления дельта-синапса в список синапсов:

`synapse-traits-library/include/knp/synapse-traits/all_traits.h`

```
#include "delta.h"

namespace knp::synapse_traits

{
// Список типов синапсов, разделенных запятыми.
#define ALL_SYNAPSES DeltaSynapse

...
}  // namespace knp::neuron_traits
```

4. Реализуйте в нужных бекендах перегруженный метод запуска.

   Например, для бекенда CPU реализуйте перегруженный метод запуска расчета проекций.

Исполнение нейронной сети, загруженной на бекенд автоматически

Этот раздел содержит инструкции по созданию нейронной сети, ее автоматической загрузке на бекенд и исполнению.

> Вы можете использовать эту инструкцию при разработке прикладных решений.

Чтобы автоматически загрузить нейронную сеть на бекенд и исполнить ее:

1. В директории вашего проекта создайте файл программы формата CPP, в котором будет реализована функция для создания и запуска нейронной сети.

2. В файле программы подключите заголовочные файлы, необходимые для исполнения нейронной сети, с помощью директивы `#include`.

   Если требуется, определите псевдонимы с помощью оператора `using`.

   Пример подключения заголовочных файлов для исполнения нейронной сети с популяцией BLIFAT-нейронов и проекцией дельта-синапсов на однопоточном бекенде для CPU:

   C++

   ```
   #include <knp/framework/io/out_converters/convert_set.h>
   #include <knp/framework/model_executor.h>
   #include <knp/framework/network.h>
   #include <knp/neuron-traits/blifat.h>
   #include <knp/synapse-traits/delta.h>

   #include <filesystem>


   using DeltaProjection = knp::core::Projection<knp::synapse_traits::DeltaSynapse>;
   using BLIFATPopulation = knp::core::Population<knp::neuron_traits::BLIFATNeuron>;
   ```

3. Реализуйте генератор синапсов с нужными вам свойствами или используйте готовый генератор из библиотеки фреймворка.

   Пример реализации генератора дельта-синапсов:

   C++

   ```
   // Реализована функция, генерирующая синапсы для проекции, которая будет связана
   // с каналом ввода. Функция хранится в переменной input_projection_gen.
   inline std::optional<DeltaProjection::Synapse> input_projection_gen(size_t
   /*index*/)
   {
       return DeltaProjection::Synapse{{1.0, 1,
   knp::synapse_traits::OutputType::EXCITATORY}, 0, 0};
   }


   // Реализована функция, генерирующая синапсы для проекции, которая будет замыкать
   ```

```
вывод
// популяции на себя. Функция хранится в переменной synapse_generator.
inline std::optional<DeltaProjection::Synapse> synapse_generator(size_t /*index*/)
{
    return DeltaProjection::Synapse{{1.0, 6,
knp::synapse_traits::OutputType::EXCITATORY}, 0, 0};
}
```

4. Реализуйте генератор нейронов с нужными вам свойствами или используйте готовый генератор из библиотеки фреймворка.

   Пример реализации генератора BLIFAT-нейронов:

```C++
// Реализована функция, генерирующая нейроны.
// Функция хранится в переменной neuron_generator.
inline knp::neuron_traits::neuron_parameters<knp::neuron_traits::BLIFATNeuron>
neuron_generator(size_t)
{
    return knp::neuron_traits::neuron_parameters<knp::neuron_traits::BLIFATNeuron>
{};
}
```

5. Создайте функцию, в которой будут созданы объекты, необходимые для исполнения нейронной сети.

   Пример создания функции `main`:

```C++
main(int argc, const char* const argv[])
{
    ...
```

6. В созданной функции создайте объект популяции и передайте в конструктор генератор нейронов.

   Пример создания объекта популяции:

```C++
    ...
    // Создает объект популяции с одним BLIFAT-нейроном
    BLIFATPopulation population{neuron_generator, 1};

    // Сохраняет UID популяции в переменной output_uid
    knp::core::UID output_uid = population.get_uid();
    ...
```

7. В созданной функции создайте объект проекции, который будет замыкать вывод популяции на себя. Передайте в конструктор идентификатор связанной популяции и генератор синапсов.

   Пример создания проекции, замыкающей вывод популяции на себя:

```C++
    ...
    // Создает объект проекции с одним дельта-синапсом, замыкающей вывод популяции
    // самой на себя
```

```
    DeltaProjection loop_projection =
            DeltaProjection{population.get_uid(), population.get_uid(),
synapse_generator, 1};
    ...
```

8. В созданной функции создайте объект входной проекции. Передайте в конструктор генератор синапсов и идентификатор связанной популяции.

   Пример создания входной проекции, связанной с каналом ввода и популяцией:

```
C++

    ...
    // Создает объект входной проекции с одним дельта-синапсом, которой
присваивается
    // нулевой идентификатор(knp::core::UID{false}). Проекция принимает спайки от
    // канала ввода и отправляет синаптическое воздействие в объект популяции.
    DeltaProjection input_projection = DeltaProjection{knp::core::UID{false},
population.get_uid(), input_projection_gen, 1};

    // Сохраняет UID входной проекции в переменной input_uid
    knp::core::UID input_uid = input_projection.get_uid();
    ...
```

9. В созданной функции создайте объект нейронной сети и передайте в него созданные проекции и популяцию.

   Пример создания объекта нейронной сети:

```
C++

    ...
    // Создает объект сети network
    knp::framework::Network network;

    // Добавляет созданный объект популяции population в объект нейронной сети
network
    network.add_population(std::move(population));

    // Добавляет созданный объект входной проекции input_projection в объект
    // нейронной сети network
    network.add_projection(std::move(input_projection));

    // Добавляет созданный объект проекции loop_projection, замыкающей вывод
    // популяции самой на себя, в объект нейронной сети network
    network.add_projection(std::move(loop_projection));
    ...
```

10. В созданной функции определите идентификаторы каналов ввода и вывода.

    Пример определения идентификаторов каналов ввода и вывода:

```
C++

    ...
    // Создает произвольные идентификаторы i_channel_uid и o_channel_uid для
каналов ввода и вывода
    knp::core::UID i_channel_uid, o_channel_uid;
    ...
```

11. В созданной функции создайте объект модели. Передайте в модель объект нейронной сети, идентификаторы каналов, популяции и входной проекции.

    Пример создания модели:

```C++
    ...
    // Создает объект модели model и передает в нее объект нейронной сети network
    knp::framework::Model model(std::move(network));

    // Передает в объект модели model созданный идентификатор канала ввода
    // i_channel_uid и идентификатор входной проекции input_uid.
    model.add_input_channel(i_channel_uid, input_uid);

    // Передает в объект модели model созданный идентификатор канала вывода
    // o_channel_uid и идентификатор популяции output_uid.
    model.add_output_channel(o_channel_uid, output_uid);
    ...
```

12. В созданной функции создайте функтор генерации спайков.

    Пример создания функтора генерации спайков:

```C++
    ...
    auto input_gen = [](knp::core::messaging::Step step) ->
knp::core::messaging::SpikeData
    {
        // Посылает спайки на шагах 0, 5, 10 и 15 исполнения нейронной сети
        if (step % 5 == 0)
        {
            knp::core::messaging::SpikeData s;
            s.push_back(0);
            return s;
        }
        return knp::core::messaging::SpikeData();
    };
    ...
```

13. В созданной функции укажите путь к экземпляру бекенда, который будет исполнять нейронную есть.

    Путь к бекенду должен соответствовать пути к динамической библиотеке бекенда. Если экземпляр бекенда находится не по указанному пути, измените путь к бекенду.

    Пример:

```C++
    ...
    // Заданный путь приведен в качестве примера. Укажите путь к нужной
    // динамической библиотеке бекенда на вашем локальном компьютере.
    auto backend_path = std::filesystem::path(argv[0]).parent_path().parent_path()
/ "lib" / "knp-cpu-single-threaded-backend";
    ...
```

14. В созданной функции создайте объект исполнителя модели. Передайте исполнителю модели объект модели, путь к бекенду, идентификатор канала ввода и функтор генерации спайков. Запустите исполнение модели.

    Пример создания объекта исполнителя модели и исполнения 20 шагов нейронной сети:

```C++
    ...
    // Создает объект исполнителя модели me. Передает в объект me объект модели model,
    // путь к бекенду backend_path, идентификатор канала ввода i_channel_uid и
    // функтор генерации спайков input_gen.
    knp::framework::ModelExecutor me(model, backend_path, {{i_channel_uid, input_gen}});

    // Инициализирует исполнитель модели me
    me.init();

    // Получает ссылку на объект канала вывода out_channel из
    // исполнителя моделей me по идентификатору канала вывода o_channel_uid
    auto &out_channel = me.get_output_channel(o_channel_uid);

    // Запускает исполнение модели на 20 шагов
    me.start([](size_t step) { return step < 20; });
    ...
```

[Шаги исполнения нейронной сети]

Исполнение нейронной сети производится циклично. Действия, описанные в цикле, повторяются каждые 5 шагов исполнения сети.

На первом шаге цикла канал ввода посылает спайки входной проекции.

На следующем шаге цикла входная проекция вычисляет синаптические воздействия и отправляет их популяции. После получения синаптического воздействия популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя.

Через 6 шагов цикла после получения спайков, отправленных входной проекцией, проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия, и передает воздействия популяции.

На рисунке ниже представлена схема исполнения нейронной сети.

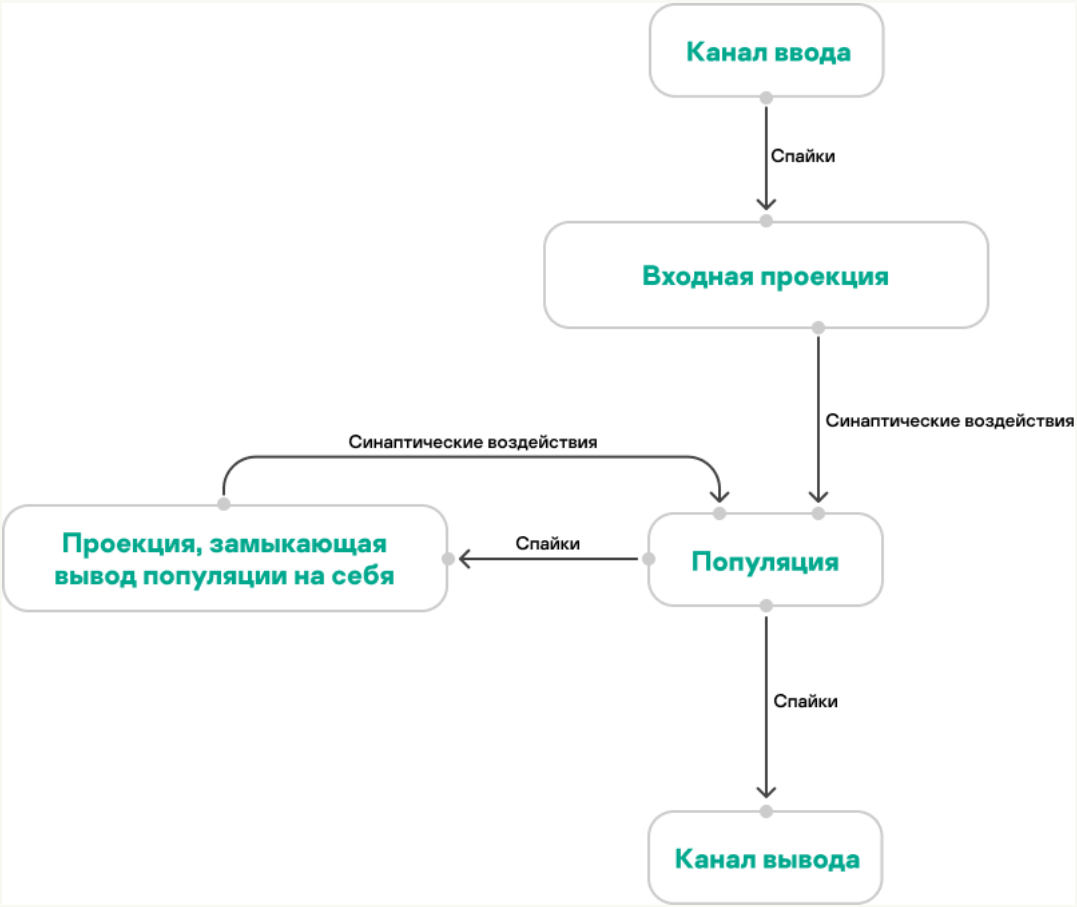


Схема исполнения нейронной сети

В этом примере нейронная сеть исполняется в рамках цикла со следующими шагами:

- *Шаг 0*. Канал ввода посылает спайки входной проекции. Шаг повторяется каждые 5 шагов цикла исполнения нейронной сети.

- *Шаг 1*. Входная проекция вычисляет синаптические воздействия и отправляет их популяции. После получения синаптического воздействия популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения сети.

- *Шаг 7*. Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных 6 шагов назад, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения нейронной сети.

- *Шаг 13*. Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных 6 шагов назад, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения нейронной сети.

- *Шаг 19*. Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных на шаге 13, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя.

15. При необходимости создайте вектор, в который будут записываться индексы шагов, на которых на канал вывода приходят спайки.

    Пример записи результатов работы модели:

```C++
    // Создает вектор results, в который будут записываться индексы шагов со
спайками
    std::vector<knp::core::messaging::Step> results;

    // Обновляет канал вывода
    const auto &spikes = out_channel.update();

    // Выделяет в памяти область для спайков
    results.reserve(spikes.size());

    // Записывает в вектор results индексы шагов, на которых на канал вывода
    // приходят спайки
    std::transform(
        spikes.cbegin(), spikes.cend(), std::back_inserter(results),
        [](const auto &spike_msg) { return spike_msg.header_.send_time_; });

    // Выводит через пробел индексы шагов, на которых на канал вывода приходят
спайки
    for (const auto &s : results) std::cout << s << " ";
    // Выводит символ новой строки и вызывает метод flush()
    std::cout << std::endl;
}
```

Исполнение нейронной сети, созданной из проекций и популяций

Этот раздел содержит инструкции по созданию нейронной сети из проекций и популяций, загрузке проекций и популяций на бекенд и исполнению нейронной сети.

Этот вариант построения и исполнения нейронной сети используется редко.
Вы можете использовать эту инструкцию при разработке прикладных решений.

Чтобы создать нейронную сеть вручную и исполнить ее:

1. В директории вашего проекта создайте файл программы формата CPP, в котором будет реализована функция для создания и запуска нейронной сети.

2. В файле программы подключите заголовочные файлы, необходимые для исполнения нейронной сети, с помощью директивы `#include`.

   Если требуется, определите псевдонимы с помощью оператора `using`.

   Пример подключения заголовочных файлов для исполнения нейронной сети с популяцией BLIFAT-нейронов и проекцией дельта-синапсов на однопоточном бекенде для CPU:

```C++
#include <knp/backends/cpu-single-threaded/backend.h>
#include <knp/core/population.h>
#include <knp/core/projection.h>
#include <knp/neuron-traits/blifat.h>
#include <knp/synapse-traits/delta.h>

#include <vector>


using Backend = knp::backends::single_threaded_cpu::SingleThreadedCPUBackend;
using DeltaProjection = knp::core::Projection<knp::synapse_traits::DeltaSynapse>;
using BLIFATPopulation = knp::core::Population<knp::neuron_traits::BLIFATNeuron>;
using Population =
knp::backends::single_threaded_cpu::SingleThreadedCPUBackend::PopulationVariants;
using Projection =
knp::backends::single_threaded_cpu::SingleThreadedCPUBackend::ProjectionVariants;
```

3. Реализуйте генератор нейронов с нужными вам свойствами или используйте готовый генератор из библиотеки фреймворка.

   Пример реализации генератора BLIFAT-нейронов:

```C++
...
// Реализована функция, генерирующая нейроны.
// Функция хранится в переменной neuron_generator.
auto neuron_generator = [](size_t index)
{
    return knp::neuron_traits::neuron_parameters<knp::neuron_traits::BLIFATNeuron>
{};
};
...
```

4. Реализуйте генератор синапсов с нужными вам свойствами или используйте готовый генератор из библиотеки фреймворка.

   Пример реализации генератора дельта-синапсов:

```C++
...
// Реализована функция, генерирующая синапсы для проекции, которая будет связана
// с каналом ввода. Функция хранится в переменной input_projection_gen.
DeltaProjection::SynapseGenerator input_projection_gen = [](size_t index) ->
std::optional<DeltaProjection::Synapse>
{
    return DeltaProjection::Synapse{{1.0, 1,
knp::synapse_traits::OutputType::EXCITATORY}, 0, 0};
};
```

```
// Реализована функция, генерирующая синапсы для проекции, которая будет замыкать
// вывод популяции на себя. Функция хранится в переменной synapse_generator.
DeltaProjection::SynapseGenerator synapse_generator = [](size_t index) ->
std::optional<DeltaProjection::Synapse>
{
    return DeltaProjection::Synapse{{1.0, 6,
knp::synapse_traits::OutputType::EXCITATORY}, 0, 0};
};
...
```

5. Создайте объект популяции и передайте в конструктор созданный генератор нейронов.

   Пример создания объекта популяции:

```
C++

...
// Создает объект популяции
BLIFATPopulation population{neuron_generator, 1};
```

6. Создайте объект проекции, которая будет замыкать вывод популяции на себя. Передайте в конструктор идентификатор связанной популяции и генератор синапсов.

   Пример создания проекции, замыкающей вывод популяции на себя:

```
C++

...
// Создает объект проекции, замыкающей вывод популяции на себя
Projection loop_projection = DeltaProjection{population.get_uid(),
population.get_uid(), synapse_generator, 1};
...
```

7. Создайте объект входной проекции. Передайте в конструктор генератор синапсов и идентификатор связанной популяции.

   Пример создания входной проекции, связанной с каналом ввода и популяцией:

```
C++

...
// Создает объект входной проекции, которой присваивается нулевой идентификатор
// (knp::core::UID{false}). Проекция принимает спайки от канала ввода и отправляет
// синаптическое воздействие в объект популяции.
Projection input_projection = DeltaProjection{knp::core::UID{false},
population.get_uid(), input_projection_gen, 1};

// Получает UID входной проекции, которая отправляет популяции синаптическое
воздействие
knp::core::UID input_uid = std::visit([](const auto &proj) { return proj.get_uid();
}, input_projection);
...
```

8. Загрузите популяцию и проекции на бекенд.

   Пример загрузки популяции и проекций на бекенд:

```C++
...
Backend backend;
...
// Загружает на бекенд созданный объект популяции
backend.load_populations({population});

// Загружает на бекенд созданные объекты проекций
backend.load_projections({input_projection, loop_projection});
```

9. Определите объект точки подключения.

   Пример создания точки подключения:

```C++
...
// Создает точку подключения
auto endpoint = backend.message_bus_.create_endpoint();
...
```

10. Подключите канал ввода к входной проекции, а также популяцию к каналу вывода. Задайте для каналов ввода и вывода уникальные идентификаторы и сформируйте для них сообщения.

    Пример подключения к каналам ввода и вывода:

```C++
...
// Создает произвольный UID канала ввода
knp::core::UID in_channel_uid;

// Создает произвольный UID канала вывода
knp::core::UID out_channel_uid;

// Создает подписку на спайки от канала ввода
backend.subscribe<knp::core::messaging::SpikeMessage>(input_uid, {in_channel_uid});

// Создает подписку на спайки от популяции с заданным UID для канала вывода
// с заданным UID
endpoint.subscribe<knp::core::messaging::SpikeMessage>(out_channel_uid,
{population.get_uid()});
...
```

11. Настройте передачу сообщений в нейронной сети и укажите параметры исполнения такта нейронной сети.

    Следующий пример исполнения нейронной сети состоит из 20 шагов:

```C++
...
std::vector<size_t> results;
for (size_t step = 0; step < 20; ++step)
{
    // Посылает спайки входной проекции от канала ввода на шагах 0, 5, 10 и 15
    if (step % 5 == 0)
    {
```

```
            knp::core::messaging::SpikeMessage message{{in_channel_uid, 0}, {0}};
            endpoint.send_message(message);
        }
        backend.step();
        endpoint.receive_all_messages();
        // Получает спайки, отправленные популяцией на канал вывода
        auto output = endpoint.unload_messages<knp::core::messaging::SpikeMessage>
(out_channel_uid);

        // Записывает индексы шагов, на которых на канал вывода приходят спайки
        if (!output.empty()) results.push_back(step);
    }
    ...
```

Шаги исполнения нейронной сети

Исполнение нейронной сети производится циклично. Действия, описанные в цикле, повторяются каждые 5 шагов исполнения сети.

На первом шаге цикла канал ввода посылает спайки входной проекции.

На следующем шаге цикла входная проекция вычисляет синаптические воздействия и отправляет их популяции. После получения синаптического воздействия популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя.

Через 6 шагов цикла после получения спайков, отправленных входной проекцией, проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия, и передает воздействия популяции.

На рисунке ниже представлена схема исполнения нейронной сети.

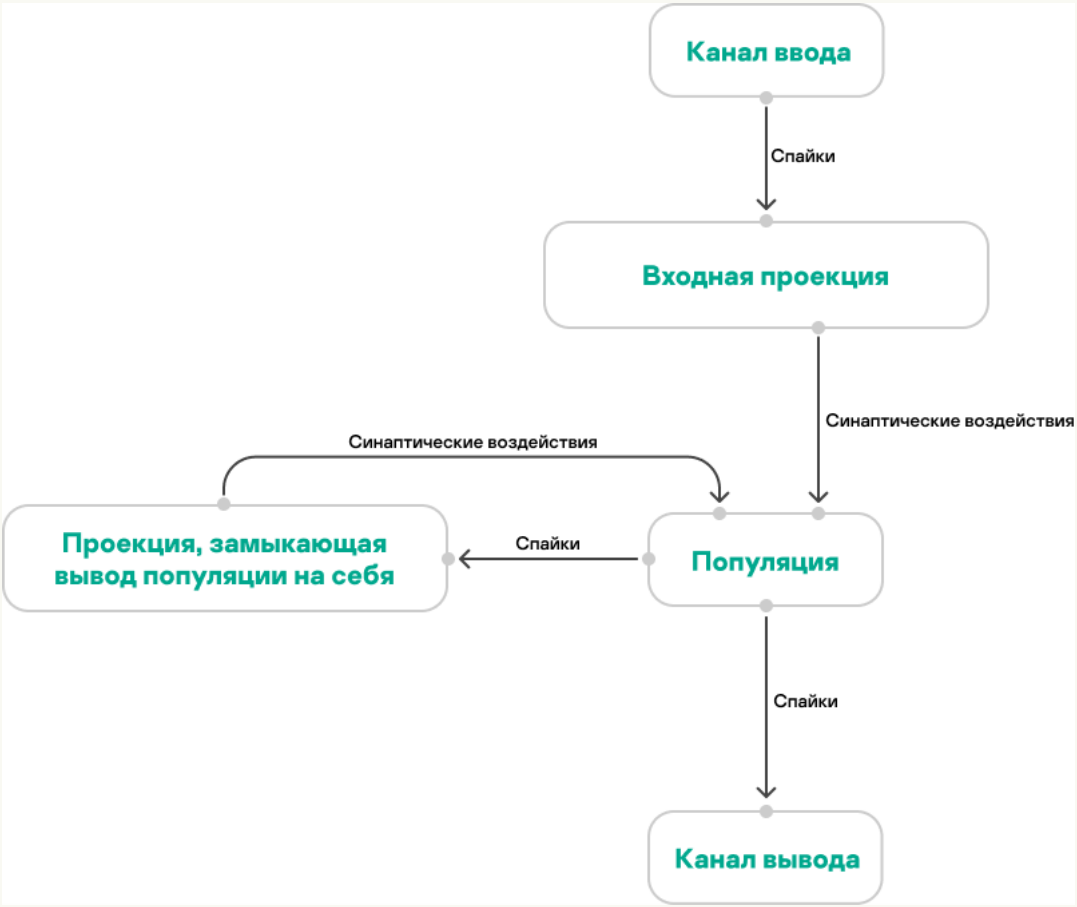


Схема исполнения нейронной сети

В этом примере нейронная сеть исполняется в рамках цикла со следующими шагами:

- *Шаг 0*. Канал ввода посылает спайки входной проекции. Шаг повторяется каждые 5 шагов цикла исполнения нейронной сети.

- *Шаг 1*. Входная проекция вычисляет синаптические воздействия и отправляет их популяции. После получения синаптического воздействия популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения сети.

- *Шаг 7*. Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных 6 шагов назад, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения нейронной сети.

- *Шаг 13*. Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных 6 шагов назад, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения нейронной сети.

- *Шаг 19*. Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных на шаге 13, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя.

Решение типовых задач на Python

Этот раздел содержит описание типовых пользовательских задач, выполняемых на языке программирования Python.

Исполнение нейронной сети, загруженной на бекенд вручную

Этот раздел содержит инструкции по созданию нейронной сети, ее загрузке на бекенд и исполнению.

> Вы можете использовать эту инструкцию для разработки прикладных решений.

Чтобы автоматически загрузить нейронную сеть на бекенд и исполнить ее:

1. В директории вашего проекта создайте файл программы с расширением .py, в котором будет реализована функция для создания и запуска нейронной сети.

2. В файле программы импортируйте объекты библиотек платформы, необходимые для исполнения нейронной сети.

   Если требуется, определите псевдонимы с помощью оператора `as`.

   Пример подключения объектов библиотек для исполнения нейронной сети с популяцией BLIFAT-нейронов и проекцией дельта-синапсов на однопоточном бекенде для CPU:

   ```Python
   from knp.base_framework._knp_python_framework_base_framework import BackendLoader
   from knp.core._knp_python_framework_core import UID, BLIFATNeuronPopulation,
   DeltaSynapseProjection, SpikeMessage
   from knp.neuron_traits._knp_python_framework_neuron_traits import
   BLIFATNeuronParameters
   from knp.synapse_traits._knp_python_framework_synapse_traits import
   DeltaSynapseParameters, OutputType
   ```

3. Реализуйте генератор нейронов с нужными вам свойствами или используйте готовый генератор из библиотеки фреймворка.

   Пример реализации генератора BLIFAT-нейронов:

   ```Python
   # Реализована функция, генерирующая BLIFAT-нейроны
   def neuron_generator(_):  # type: ignore[no-untyped-def]
       return BLIFATNeuronParameters()
   ```

4. Реализуйте генератор синапсов с нужными вам свойствами или используйте готовый генератор из библиотеки фреймворка.

   Пример реализации генератора дельта-синапсов:

   ```Python
   # Реализована функция, генерирующая дельта-синапсы для проекции, которая будет
   замыкать вывод
   ```

```
# популяции на себя
def synapse_generator(_):  # type: ignore[no-untyped-def]
    return DeltaSynapseParameters(1.0, 6, OutputType.EXCITATORY), 0, 0


# Реализована функция, генерирующая дельта-синапсы для проекции, которая будет
связана
# с каналом ввода
def input_projection_gen(_):  # type: ignore[no-untyped-def]
    return DeltaSynapseParameters(1.0, 1, OutputType.EXCITATORY), 0, 0
```

5. Создайте функцию, в которой будут созданы объекты, необходимые для исполнения нейронной сети.

   Пример создания функции main:

```
Python

def main():  # type: ignore[no-untyped-def]
    ...
```

6. В созданной функции создайте объект популяции и передайте в конструктор генератор нейронов.

   Пример создания объекта популяции:

```
Python

    ...
    # Создает объект популяции с одним BLIFAT-нейроном
    population = BLIFATNeuronPopulation(neuron_generator, 1)
    ...
```

7. В созданной функции создайте объект проекции, который будет замыкать вывод популяции на себя. Передайте в конструктор идентификатор связанной популяции и генератор синапсов.

   Пример создания проекции, замыкающей вывод популяции на себя:

```
Python

    ...
    # Создает объект проекции с одним дельта-синапсом, замыкающей вывод популяции
    # на себя
    loop_projection = DeltaSynapseProjection(population.uid, population.uid,
    synapse_generator, 1)
    ...
```

8. В созданной функции создайте объект входной проекции. Передайте в конструктор генератор синапсов и идентификатор связанной популяции.

   Пример создания входной проекции, связанной с каналом ввода и популяцией:

```
Python

    ...
    # Создает объект входной проекции с одним дельта-синапсом, которой
присваивается
    # нулевой идентификатор (UID(False)). Проекция принимает спайки от канала ввода
    # и отправляет синаптическое воздействие в объект популяции.
    input_projection = DeltaSynapseProjection(UID(False), population.uid,
```

```
    input_projection_gen, 1)

    # Сохраняет UID входной проекции в переменной input_uid
    input_uid = input_projection.uid
    ...
```

9. В созданной функции загрузите экземпляр бекенда и загрузите в него созданные популяцию и проекции. Создайте точку подключения.

   Пример загрузки бекенда и создания точки подключения:

```
Python

    ...
    # Загружает экземпляр однопоточного бекенда для CPU с вашего локального
компьютера
    backend = BackendLoader().load(f'{pytestconfig.rootdir}/../lib/libknp-cpu-
single-
    threaded-backend')

    # Загружает созданную популяцию в экземпляр бекенда
    backend.load_all_populations([population])
    # Загружает созданные проекции в экземпляр бекенда
    backend.load_all_projections([input_projection, loop_projection])

    # Инициализирует бекенд
    backend._init()
    # Создает объект точки подключения
    endpoint = backend.message_bus.create_endpoint()
    ...
```

10. В созданной функции определите идентификаторы каналов ввода и вывода.

    Пример определения идентификаторов каналов ввода и вывода:

```
Python

    ...
    # Создает произвольные идентификаторы in_channel_uid и out_channel_uid для
каналов
    # ввода и вывода
    in_channel_uid = UID()
    out_channel_uid = UID()
    ...
```

11. В созданной функции подпишите бекенд и точку подключения на сообщения от каналов ввода и вывода.

    Пример подписки бекенда и точки подключения на сообщения `SpikeMessage`:

```
Python

    ...
    # Подписывает бекенд на спайки от канала ввода
    backend.subscribe(SpikeMessage, input_uid, [in_channel_uid])
    # Подписывает точку подключения на спайки от канала вывода
    endpoint.subscribe(SpikeMessage, out_channel_uid, [population.uid])
    ...
```

12. В созданной функции исполните нейронную сеть. При необходимости создайте список, в который будут записываться индексы шагов, на которых на канал вывода приходят спайки.

    Пример исполнения 20 шагов нейронной сети:

```Python
    ...
    # Создает список results, в который будут записываться индексы шагов со
спайками
    results = []

    for step in range(0, 20):
        # Посылает спайки на шагах 0, 5, 10 и 15 исполнения нейронной сети
        if step % 5 == 0:
            # Отображает номер шага, на котором канал ввода отправил спайк входной
проекции
            print(f'STEP {step}')
            message = SpikeMessage((in_channel_uid, step), [0])
            endpoint.send_message(message)

        backend._step()
        messages_count = endpoint.receive_all_messages()
        output = endpoint.unload_messages(SpikeMessage, out_channel_uid)
        if len(output):
            # Отображает номер шага, на котором на канал вывода пришел спайк
            print(f'STEP {step}')
```

Шаги исполнения нейронной сети

Исполнение нейронной сети производится циклично. Действия, описанные в цикле, повторяются каждые 5 шагов исполнения сети.

На первом шаге цикла канал ввода посылает спайки входной проекции.

На следующем шаге цикла входная проекция вычисляет синаптические воздействия и отправляет их популяции. После получения синаптического воздействия популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя.

Через 6 шагов цикла после получения спайков, отправленных входной проекцией, проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия, и передает воздействия популяции.

На рисунке ниже представлена схема исполнения нейронной сети.

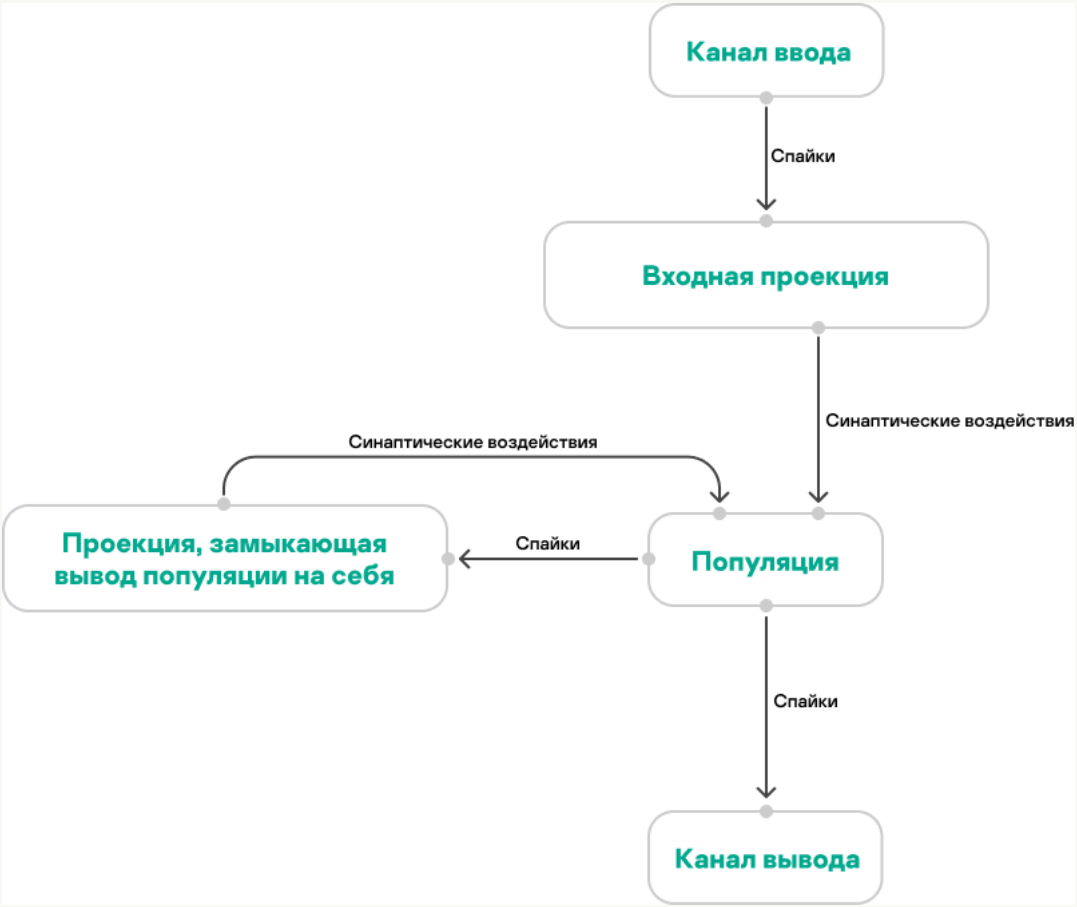


Схема исполнения нейронной сети

В этом примере нейронная сеть исполняется в рамках цикла со следующими шагами:

- *Шаг 0*. Канал ввода посылает спайки входной проекции. Шаг повторяется каждые 5 шагов цикла исполнения нейронной сети.

- *Шаг 1*. Входная проекция вычисляет синаптические воздействия и отправляет их популяции. После получения синаптического воздействия популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения сети.

- *Шаг 7*. Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных 6 шагов назад, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения нейронной сети.

- *Шаг 13*. Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных 6 шагов назад, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения нейронной сети.

- *Шаг 19*. Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных на шаге 13, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя.

Сценарий исполнения нейронной сети из тернарных слоев для классификации изображений

Вы можете использовать этот сценарий при разработке прикладных решений.

Сценарий исполнения нейронной сети из тернарных слоев для классификации изображений состоит из следующих этапов:

1. **Создание и обучение нейронной сети из тернарных слоев**

   Создайте нейронную сеть с помощью тернарных слоев библиотеки ANN2SNN. В качестве функции активации выходного слоя укажите сигмоидную функцию. Подготовьте данные для обучения нейронной сети. Приведите данные для обучения в бинарный вид. Обучите нейронную сеть и оцените результаты обучения.

2. **Размещение нейронной сети на нейроморфном процессоре Алтай-2**

   Разместите нейронную сеть на нейроморфном процессоре Алтай-2 с помощью следующей команды:

   ```
   placer.py -p <путь к файлу с обученной нейронной сетью> -l <путь к файлу, в который будут записываться журналы утилиты placer> -o <путь к конфигурационному файлу для загрузки на нейроморфный процессор>
   ```

   Например, `placer.py -p mnist.h5 -l mnist_placer.log -o mnist_altai.json`

3. **Запуск инференса нейронной сети**

   Укажите конфигурационный файл и компонент для исполнения нейронной сети. Нейронная сеть может быть исполнена как на аппаратном обеспечении нейроморфного процессора Алтай-2, так и на программном эмуляторе. Подготовьте данные для инференса и приведите их в бинарный вид. Запустите инференс.

Создание и обучение нейронной сети из тернарных слоев для классификации изображений

Этот раздел содержит инструкции по созданию нейронной сети для классификации изображений и ее обучению. Нейронная сеть создана с помощью тернарных слоев библиотеки ANN2SNN.

Чтобы создать и обучить нейронную сеть из тернарных слоев для классификации изображений:

1. В директории вашего проекта создайте файл программы с расширением .py, в котором будет создана и обучена нейронная сеть.

2. В файле программы импортируйте модули библиотек, необходимые для создания и исполнения нейронной сети.

   Если требуется, определите псевдонимы с помощью оператора `as`.

   Пример подключения объектов для создания и исполнения сверточной нейронной сети:

```Python
from altainn.ternary_tf2.layers import TernaryConv2D, TernaryDense
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from altainn.binarynet_tf2.layers import Clip
from altainn.ternary_tf2.ops import heaviside_mod as heaviside
import tensorflow as tf
```

3. Создайте нейронную сеть. Укажите сигмоидную функцию в качестве функции активации для выходного слоя.

   Пример создания нейронной сети с тремя сверточными слоями:

```Python
model = Sequential(
[TernaryConv2D(3, 2, activation=heaviside, input_shape=(28, 28, 1)),
TernaryConv2D(6, 3, activation=heaviside),
TernaryConv2D(6, 3, activation=heaviside),
Flatten(),
TernaryDense(10, activation="sigmoid")])
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=
["accuracy"])
```

4. Подготовьте данные для обучения.

   Пример подготовки данных MNIST для обучения:

```Python
# Загружает данные MNIST для обучения
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
# Кодирует целевые метки
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

# Нормализует входные данные
x_train = x_train / 255.0
x_test = x_test / 255.0
# Приводит входные данные в бинарный вид
x_train[x_train > 0.5] = 1
x_train[x_train <= 0.5] = 0

x_test[x_test > 0.5] = 1
x_test[x_test <= 0.5] = 0
# Меняет размерность данных из (28, 28) в (28, 28, 1) для правильной работы
сверточных слоев
x_train = x_train.reshape(x_train.shape + (1,))
x_test = x_test.reshape(x_test.shape + (1,))
```

5. Создайте Callback-функцию для сохранения нейронной сети.

   Пример Callback-функции:

```Python
#Сохраняет обученную нейронную сеть в файле mnist.h5
model_chekpoint_callback = ModelCheckpoint(
    filepath="mnist.h5",
    monitor="accuracy",
    mode="max",
    save_best_only=True
```

6. Обучите нейронную сеть.

   Пример обучения нейронной сети:

```Python
# Запускает обучение на тренировочных данных
model.fit(x_train, y_train, epochs = 15, verbose = 1, callbacks=
[model_chekpoint_callback])
# Загружает обученную модель
model = load_model("mnist.h5", custom_objects = {"heaviside_mod": heaviside,
"TernaryDense": TernaryDense,
    "TernaryConv2D": TernaryConv2D, "Clip": Clip})
#Оценивает точность нейронной сети
print(f"Точность модели: {model.evaluate(x_test, y_test, verbose = 1)
[1]*100:.2f}%")
```

Запуск инференса нейронной сети из тернарных слоев для классификации изображений

Этот раздел содержит инструкции по запуску инференса нейронной сети для классификации изображений на нейроморфном процессоре Алтай-2. Нейронную сеть требуется предварительно создать с помощью тернарных слоев библиотеки ANN2SNN, обучить и разместить на нейроморфном процессоре.

Чтобы запустить инференс нейронной сети из тернарных слоев для классификации изображений:

1. В директории вашего проекта создайте файл программы с расширением .py, в котором будет реализован инференс нейронной сети.

2. В файле программы импортируйте модули библиотек, необходимые для инференса нейронной сети.

   Если требуется, определите псевдонимы с помощью оператора as.

   Пример подключения объектов для инференса нейронной сети:

```Python
from python_altai.altai import Altai
import numpy as np
```

3. Укажите конфигурационный файл и компонент для исполнения нейронной сети.

   Пример конфигурации:

```python
# Для переменной PATH_TO_CONFIGURATION укажите путь к конфигурационному файлу
# для загрузки на нейроморфный процессор
# Для переменной MODE укажите одно из следующих значений:
# hw – для инференса нейронной сети на аппаратном обеспечении нейроморфного
процессора Алтай-2
# gm – для инференса нейронной сети на программном эмуляторе
PATH_TO_CONFIGURATION = 'mnist_altai.json'
MODE = 'gw'

# Загрузка конфигурации
altai_layer = Altai()
altai_layer.build(PATH_TO_CONFIGURATION, MODE)
```

4. Подготовьте данные для инференса.

   Пример подготовки данных MNIST для инференса:

```python
# Загружает данные MNIST для инференса
(_, _), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Нормализует входные данные
x_test = x_test / 255.0

# Приводит входные данные в бинарный вид
x_test[x_test > 0.5] = 1
x_test[x_test <= 0.5] = 0
```

5. Запустите инференс.

   Пример инференса:

```python
for x in x_test:
    # Преобразует входной вектор в спайки
    altai_layer.prepare_spikes(x)
    # Запускает генерацию сигнала тик
    altai_layer.start_ticks(9)
    # Получает спайки от нейронной сети
    predict = altai_layer.get_spikes()
```

Создание и обучение регрессионной модели

Этот раздел содержит инструкции по созданию регрессионной модели с обучаемыми блоками, кодирующие и декодирующие данные, и ее обучению. Модель создана с помощью тернарных слоев библиотеки ANN2SNN.

Вы можете использовать эту инструкцию при разработке прикладных решений.

Чтобы создать и обучить регрессионную модель:

1. В директории вашего проекта создайте файл программы с расширением .py, в котором будет создана и обучена модель.

2. В файле программы импортируйте модули библиотек, необходимые для создания и обучения модели.

   Если требуется, определите псевдонимы с помощью оператора `as`.

   Пример подключения объектов для создания и обучения регрессионной модели:

```Python
from altainn.ternary_tf2.layers import TernaryDense
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense
from altainn.ternary_tf2.ops import heaviside_mod as  heaviside
from altainn.binarynet_tf2.layers import Clip
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import ModelCheckpoint
import numpy as np
```

3. Создайте синусоиду и подготовьте данные для обучения.

   Пример подготовки данных:

```Python
# Создает синусоиду
x = np.arange(4000)
x = x * 0.1
arr = np.sin(x)

# Берет для обучения 10 значений, по которым прогнозируются следующие значения
# Формирует тренировочные данные
x_train = []
y_train = []
for i in range(0, 3000-11):
    x_train.append(arr[i:i+10])
    y_train.append(arr[i+11])

# Формирует тестовые данные
x_test = []
y_test = []

for i in range(3000, 4000-11):
    x_test.append(arr[i:i+10])
    y_test.append(arr[i+11])
x_train = np.array(x_train)
x_test = np.array(x_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

4. Создайте регрессионную модель. Для этого создайте кодирующий блок, импульсную нейронную сеть и декодирующий блок.

   Пример создания регрессионной модели:

```Python
# Создает кодирующий блок с одним слоем, который преобразует данные в спайки
encoder = Sequential([
    Dense(10, activation=heaviside, input_shape=(10,)),
```

```
])

# Создает импульсную нейронную сеть
snn = Sequential([
    TernaryDense(32, activation=heaviside, input_shape=(10,)),
    TernaryDense(32, activation=heaviside),
])

# Создает декодирующий блок с одним слоем, который преобразует спайки в целевое
значение
decoder = Sequential([
    Dense(1, activation="linear", input_shape=(32,))
])

# Создает модель с кодирующим блоком, нейронной сетью и декодирующим блоком
model = Sequential([encoder, snn, decoder])
model.compile(optimizer="adam", loss="mse")
```

5. Обучите модель.

Пример обучения модели:

Python

```
# Обучает модель на тренировочных данных
checkpoint = ModelCheckpoint('sin_model.h5', monitor='val_loss', verbose=1,
save_best_only=True, mode='min')
model.fit(x_train, y_train, epochs=50, validation_data=(x_test, y_test), callbacks=
[checkpoint])

# Загружает модель
model = load_model('sin_model.h5', custom_objects={'heaviside_mod': heaviside,
'Clip': Clip, "TernaryDense": TernaryDense})

# Получает предсказания
predict_test = model.predict(x_test)

# Строит графики по предсказанным и целевым значениям
plt.plot(predict_test, 'b')
plt.plot(y_test, 'r', alpha=0.5)
plt.show()
```

Использование Kaspersky Neuromorphic Platform API

В Kaspersky Neuromorphic Platform реализован интерфейс прикладного программирования (Application Programming Interface, API), который обеспечивает доступ к методам платформы.

ОТКРЫТЬ СПРАВОЧНОЕ РУКОВОДСТВО API для фреймворка для C++ (на английском языке)

С помощью Kaspersky Neuromorphic Platform API вы можете выполнять следующие действия:

- получать данные об используемых устройствах и бекендах;
- получать данные об объектах библиотеки поддержки бекендов;
- получать данные об объектах фронтенда платформы;
- управлять объектами библиотеки поддержки бекендов;
- создавать и обучать нейронные сети;
- управлять гиперпараметрами нейронных сетей;
- создавать новые модели нейронов;
- создавать новые топологии нейронных сетей;
- реализовывать прикладные решения.

Глоссарий

Бекенд

Программно-аппаратная часть платформы, обеспечивающая универсальный интерфейс к вычислителю и реализующая функции нейронов и синапсов, а также их модификаторов в терминах вычислителя.

Вес синапса

Значение сигнала, который синапс передает нейронам связанной популяции.

Вычислитель

Устройство или код, непосредственно выполняющий сеть.

Модификатор

Код, изменяющий поведение нейрона или синапса, например добавляющий синаптическую пластичность.

Нейрон

Узел нейронной сети, имеющий набор атрибутов. На основе атрибутов нейрона и последовательности входных сигналов происходит вычисление некоторой математической функции, результатом которого является синаптическое воздействие или его отсутствие.

Период ППС

Интервал времени между моментом начала Хеббовской пластичности перед первым спайком в ППС и одним из следующих моментов времени: достижение максимально допустимого расстояния (ISImax) после последнего спайка в ППС, форсированное срабатывание или приход спайка на дофаминовый синапс.

Плотная последовательность спайков (ППС)

Последовательность генерируемых нейроном неформированных спайков, в которой все соседние спайки разделены временем не большее, чем ISImax, где ISImax – это максимальное допустимое расстояние между спайками.

Популяция

Контейнер, содержащий набор нейронов и их функцию в соответствии с типом нейронов.

Проекция

Контейнер, содержащий набор синапсов одного типа, соединяющих нейроны пресинаптической и постсинаптической популяций.

Синапс

Место соединения нейронов, основная функция которого – передача сигнала от одного нейрона к другому. Каждый нейрон может быть связан с множеством синапсов.

Синаптическое воздействие

Значение, формируемое синапсом и предназначенное для изменения значений атрибутов нейронов.

Соединение

Связь синапса с пресинаптическим или постсинаптическим нейроном.

Спайк

Кратковременный потенциал действия, сгенерированный нейроном популяции в результате изменения атрибутов нейрона.

Такт

Квант времени синхросигнала, за который вычислитель ядра проводит одну операцию.

Тик

Квант времени, за который выполняется вычисление функции всех нейронов в ядре. За время тика все спайки, сгенерированные ядрами или поступившие на входные линии нейронов, доставляются до целевых нейронов.

Фреймворк

Программно-аппаратная часть платформы, определяющая ее структуру и обеспечивающая управление бекендами. В Kaspersky Neuromorphic Platform реализован фреймворк на языке программирования C++.

Функция нейрона

Система, описывающая и изменяющая поведение нейрона. Результатом функции является спайк или его отсутствие.

Функция синапса

Система, моделирующая поведение синапса. Результатом функции является синаптическое воздействие или его отсутствие.

Информация о стороннем коде

Информация о стороннем коде содержится в файле NOTICES.txt, расположенном в репозитории платформы.

Уведомления о товарных знаках

Зарегистрированные товарные знаки и знаки обслуживания являются собственностью их правообладателей.

LTS и Ubuntu являются зарегистрированными товарными знаками Canonical Ltd.

Intel и Core – товарные знаки Intel Corporation, зарегистрированные в Соединенных Штатах Америки и в других странах.

Linux – товарный знак Linus Torvalds, зарегистрированный в США и в других странах.

Python – товарный знак или зарегистрированный товарный знак Python Software Foundation.

Debian – зарегистрированный товарный знак Software in the Public Interest, Inc.