kaspersky

Kaspersky Neuromorphic Platform

© 2025 АО "Лаборатория Касперского"

Содержание

- О Kaspersky Neuromorphic Platform
  - Комплект поставки
  - Аппаратные и программные требования
  - Сторонние библиотеки
- Архитектура Kaspersky Neuromorphic Platform
  - Формат описания нейронной сети
    - Структура файла, описывающего группы нейронов
    - Структура файла, описывающего группы синапсов
  - Механизм обмена данными в Kaspersky Neuromorphic Platform
  - Форматы данных, обрабатываемые платформой
  - О бэкенде Kaspersky Neuromorphic Platform
    - Поддерживаемые бэкенды
    - О нейроморфном процессоре Алтай-1
    - О работе бэкенда для CPU
- Установка и удаление платформы
  - Установка deb-пакетов
  - Установка пакетов для разработки на языке программирования Python
  - Сценарий сборки проекта платформы или прикладного решения
    - Загрузка и распаковка архива с исходным кодом платформы
    - Определение параметра Boost_ROOT в ОС Windows
    - Настройка параметров сборки проекта прикладного решения на C++
    - Конфигурация сборки платформы или прикладного решения в ОС Windows
  - Удаление платформы
- Компоненты платформы для использования на C++
  - Библиотека neuron_traits
  - Библиотека synapse_traits
  - Библиотека core
    - Класс UID
    - Функция uid_hash
    - Класс continuously_uid_generator
    - Класс TagMap
    - Структура BaseData
    - Класс Population
    - Класс Projection
    - Класс Backend
    - Класс Device
    - Класс MessageBus
    - Класс MessageEndpoint
    - Класс Subscription
    - Пространство имен tags
    - Пространство имен messaging
  - Библиотека meta
  - Библиотека devices
  - Фреймворк для C++
    - Пространство имен sonata
    - Пространство имен coordinates

- Пространство имен synchronization
- Пространство имен storage
- Пространство имен population
- Пространство имен projection
- Пространство имен input
- Пространство имен output
- Пространство имен monitoring
- Класс Network
- Класс Model
- Класс BackendLoader
- Класс ModelLoader
- Класс ModelExecutor

Решение типовых задач на C++

- Добавление нового типа нейрона
- Добавление нового типа синапса
- Исполнение нейронной сети, созданной из проекций и популяций

Компоненты платформы для использования на Python

- Модуль neuron_traits
- Модуль synapse_traits
- Модуль core
  - Класс UID
  - Класс uid_hash
  - Класс UUID
  - Класс continuously_uid_generator
  - Класс TagMap
  - Структура BaseData
  - Набор классов Population
  - Набор классов Projection
  - Класс Backend
  - Структура BackendWrapper
  - Класс MessageBus
  - Класс MessageEndpoint
  - Набор классов Subscription
  - Пространство имен messaging
- Python-фреймворк
  - Пакет ANN2SNN
  - Тернарные слои пакета ANN2SNN
  - Пространство имен sonata
  - Пространство имен input
  - Пространство имен output
  - Класс BackendLoader
  - Класс Network
  - Класс Model
  - Класс SpikeMessageObserver
  - Класс ImpactMessageObserver
  - Класс ModelLoader
  - Класс ModelExecutor

Решение типовых задач на Python

- Исполнение нейронной сети, загруженной на бэкенд вручную
- Сценарий исполнения нейронной сети из тернарных слоев для классификации изображений
  - Создание и обучение нейронной сети из тернарных слоев для классификации изображений
  - Запуск инференса нейронной сети из тернарных слоев для классификации изображений
- Сценарий исполнения регрессионной модели с обучаемыми блоками, кодирующие и декодирующие данные
  - Создание и обучение регрессионной модели
  - Запуск инференса регрессионной модели

Использование Kaspersky Neuromorphic Platform API

Участие в разработке платформы

Лицензирование платформы

О предоставлении данных

Приложение. Параметры и пример YAML-файла, содержащего конфигурацию нейроморфного процессора Алтай-1

Глоссарий

- Artificial Neural Network (ANN)
- Spiking Neural Network (SNN)
- Бэкенд
- Бэкенд AltAI ANN2SNN
- Бэкенд AltAI SNN
- Вес синапса
- Внешняя среда
- Вычислитель
- Модификатор
- Нейрон
- Период ППС
- Плотная последовательность спайков (ППС)
- Популяция
- Проекция
- Синапс
- Синаптическое воздействие
- Соединение
- Спайк
- Такт
- Тик
- Фреймворк
- Функция нейрона
- Функция синапса

Информация о стороннем коде

Уведомления о товарных знаках

О Kaspersky Neuromorphic Platform

Kaspersky Neuromorphic Platform (далее также KNP, платформа) – это программная платформа, предназначенная для разработки импульсных нейронных сетей, их обучения и исполнения на разных вычислителях.

С помощью Kaspersky Neuromorphic Platform вы можете:

- Создавать импульсные нейронные сети (англ. Spiking Neural Network, SNN) и обучать их на различных типах входных данных (например, телеметрии, событиях, изображениях, 3D-данных, звуковых и тактильных данных).
- Преобразовывать искусственные нейронные сети (англ. Artificial Neural Network, ANN) в импульсные и обучать их.
- Оптимизировать структуру загруженных нейронных сетей.
- Проводить прикладные исследования в области классификации входных данных и других областях применения 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 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. neuromorphic processors).

Для решения вышеперечисленных задач вы можете использовать языки C++ и Python®. Платформа поддерживает CPU, а также нейроморфный процессор Алтай-1, предназначенный для энергоэффективного исполнения нейронных сетей в разных классах интеллектуальных устройств.

Комплект поставки

Kaspersky Neuromorphic Platform имеет несколько вариантов комплекта поставки. Вы можете выбрать тот вариант, который наиболее подходит для ваших целей использования платформы.

Для сборки платформы или прикладного решения также поставляется Docker-образ `knp-build-image`. Docker-образ необходимо скачать с репозитория Docker Hub.

Комплект поставки Kaspersky Neuromorphic Platform для разработки прикладных решений на Linux

В комплект поставки входит следующие файлы:

- Архив `pkg-deb.zip` со следующими пакетами:

- `knp-cpu-single-threaded-backend_<knp_version>_amd64.deb` – deb-пакет с бинарным кодом однопоточного бэкенда для CPU.
- `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb` – deb-пакет с бинарным кодом многопоточного бэкенда для CPU.
- `knp-cpp-framework_<knp_version>_amd64.deb` – 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 пакеты для каждого deb-пакета с бинарным кодом на C++:
  - `knp-cpu-single-threaded-backend-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb`.
  - `knp-cpu-multi-threaded-backend-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb`.
  - `knp-cpp-framework-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpp-framework_<knp_version>_amd64.deb`.

    В каждый dev-deb пакет входят следующие файлы:
    - Заголовочные файлы.
    - Модули CMake для сборки платформы на C++.
    - Статические версии библиотек платформы.
- `knp-python3-framework_<knp_version>_amd64.deb` – deb-пакет с Python-фреймворком для работы на бэкенде `AltAI SNN`.
- `knp-examples_<knp_version>_amd64.deb` – deb-пакет с примерами использования Kaspersky Neuromorphic Platform.
- `knp-documentation_<knp_version>_all.deb` – deb-пакет со справочным руководством по API.
- `knp-unspecified_<knp_version>_amd64.deb` – deb-пакет со служебными файлами CMake.
- `knp-altai-backend_<motiv_version>_amd64.deb` – deb-пакет с бинарным кодом бэкенда AltAI SNN.

  deb-пакет предоставляется компанией ООО "Мотив НТ".
- `knp-altai-backend-dev_<motiv_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-altai-backend_<motiv_version>_amd64.deb`.

  dev-deb пакет предоставляется компанией ООО "Мотив НТ".
- `knp_ann2snn.deb` – deb-пакет с пакетом `ANN2SNN`, бинарным кодом бэкенда AltAI ANN2SNN для нейроморфного процессора Алтай-1 и компонентом для размещения нейронной сети на нейроморфном процессоре Алтай-1.

  deb-пакет предоставляется компанией ООО "Мотив НТ".

где:

- `<knp_version>` – версия Kaspersky Neuromorphic Platform;
- `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`.

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

В комплект поставки Kaspersky Neuromorphic Platform входят следующие файлы:

- Архив `pkg-deb.zip` с deb-пакетом `knp-python3-framework_<knp_version>_amd64.deb`, содержащим Python-фреймворк для работы на бэкенде `AltAI SNN`.
- Архив `pkg-python.zip` с whl-пакетом `knp-<knp_version>-py3-none-any.whl`, содержащий Python-фреймворк для работы на бэкенде `AltAI SNN`.
- `knp_ann2snn.deb` – deb-пакет с пакетом `ANN2SNN`, бинарным кодом бэкенда `AltAI ANN2SNN` для нейроморфного процессора Алтай-1 и компонентом для размещения нейронной сети на нейроморфном процессоре Алтай-1.

  deb-пакет предоставляется компанией ООО "Мотив НТ".
- `knp_ann2snn-<motiv_version>-py3-none-any.whl` – whl-пакет с пакетом `ANN2SNN`, бинарным кодом бэкенда `AltAI ANN2SNN` для нейроморфного процессора Алтай-1 и компонентом для размещения нейронной сети на нейроморфном процессоре Алтай-1.

  Этот пакет можно установить на компьютер с процессором архитектуры x86.

  whl-пакет предоставляется компанией ООО "Мотив НТ".
- `knp_ann2snn_aarch64-<motiv_version>-py3-none-any.whl` – whl-пакет с пакетом `ANN2SNN` и бинарным кодом бэкенда `AltAI ANN2SNN` для нейроморфного процессора Алтай-1.

  Этот пакет можно установить на компьютер с процессором архитектуры ARM.

  whl-пакет предоставляется компанией ООО "Мотив НТ".

  где:

  - `<knp_version>` – версия Kaspersky Neuromorphic Platform;
  - `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`.

Комплект поставки Kaspersky Neuromorphic Platform для работы с исходным кодом платформы

Комплект поставки может быть использован для разработки прикладных решений на C++ на Linux и в ОС Windows.

Вы можете получить файлы с исходным кодом Kaspersky Neuromorphic Platform путем клонирования репозитория платформы. Вы также можете получить файлы с исходным кодом в виде архива.

Комплект поставки включает следующие файлы:

- Исходный код 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 стороннем коде `NOTICE.txt` на английском языке.
- Файл с правилами участников репозитория `CONTRIBUTING.md` на английском языке.
- Файл с условиями участия в репозитории `CLA.md` на английском языке.

Аппаратные и программные требования

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

Минимальные аппаратные требования:

- центральный процессор: совместимый процессор Intel Core i5 и выше;
- нейроморфный процессор (при необходимости): Алтай-1;
- 8 ГБ оперативной памяти;
- Свободного пространства на жестком диске:
  - 1 ГБ для установки Kaspersky Neuromorphic Platform.
  - 10 ГБ для сборки платформы или прикладного решения.

Поддерживаемые операционные системы:

- Debian GNU/Linux 12.5 и выше;
- Ubuntu 22.04 LTS и выше;
- Windows 7;
- Windows 10.

Вы можете использовать устройство под управлением любой другой операционной системы из семейства Linux, если дистрибутив операционной системы содержит библиотеку Boost версии 1.81 и выше.

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

- Visual Studio 2022.

  Это программное обеспечение требуется установить для сборки платформы или прикладного решения в ОС Windows.

- Boost 1.81 и выше

  При работе в ОС Windows рекомендуется установить библиотеку, скомпилированную предварительно для Visual Studio 2022 (версия компилятора – 14.3).

- CMake 3.25 и выше.
- При установке deb-пакетов с компонентами платформы для работы на бэкенде AltAI ANN2SNN должно быть установлено следующее программное обеспечение:
  - 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.
- При установке whl-пакета или deb-пакета с Python-фреймворком для работы на бэкенде AltAI SNN на устройстве должен быть установлен Python 3.10 и выше.
- При установке whl-пакета с Python-фреймворком для работы на бэкенде `AltAI ANN2SNN` на устройстве должно быть установлено следующее программное обеспечение:
  - NumPy 1.24.3.
  - TensorFlow 2.13.1.
  - Loguru 0.7.2.
  - PyYAML 6.0.1.
  - NetworkX 3.1.
  - Matplotlib 3.6.3.
  - tqdm 4.66.5.
- При установке deb-пакета с Python-фреймворком для работы на бэкенде `AltAI ANN2SNN` на устройстве должно быть установлено следующее программное обеспечение:
  - python3-pip.
  - python3-numpy.
  - python3-yaml.
  - python3-networkx.
  - python3-matplotlib.
  - python3-loguru.
  - python3-tqdm.
  - TensorFlow 2.13.1.

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.

Архитектура Kaspersky Neuromorphic Platform

Kaspersky Neuromorphic Platform включает в себя следующие компоненты:

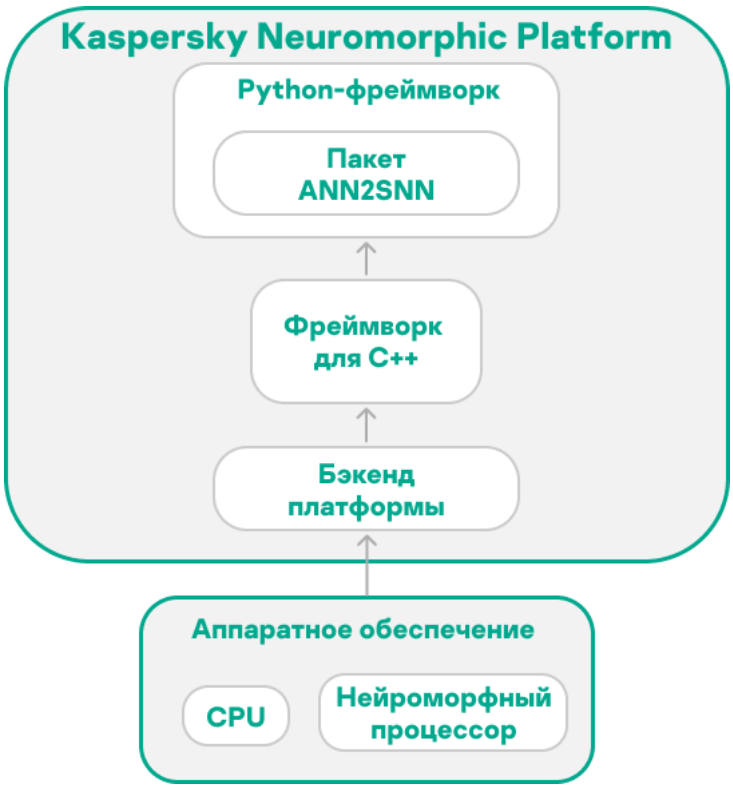
- *Python-фреймворк*. Предназначен для создания импульсных нейронных сетей, а также преобразования искусственных нейронных сетей в импульсные. Созданные импульсные нейронные сети обучаются методами синаптической пластичности и могут быть исполнены на центральном процессоре, нейроморфном процессоре Алтай-1 или его программном эмуляторе. Искусственные нейронные сети, преобразованные в импульсные, обучаются методом обратного распространения ошибки с учетом ограничений нейроморфного процессора Алтай-1. Полученные в результате нейронные сети адаптированы для исполнения на нейроморфном процессоре Алтай-1 или его программном эмуляторе.

  Python-фреймворк включает в себя модули, функциональность которых пересекается с функциональностью фреймворка для C++, а также пакет `ANN2SNN`, необходимый для преобразования искусственных нейронных сетей в импульсные и их исполнения на бэкенде AltAI ANN2SNN.

- *Фреймворк для C++*. Предназначен для создания импульсных нейронных сетей и их обучения методами синаптической пластичности. Полученные в результате нейронные сети могут исполняться на центральном процессоре, нейроморфном процессоре Алтай-1 или его программном эмуляторе.

- *Бэкенд*. Запускает обучение и исполнение импульсной нейронной сети на различных вычислительных устройствах (например, на нейроморфном процессоре Алтай-1). Платформа предоставляет несколько независимых друг от друга бэкендов, каждый из которых позволяет обучать и исполнять импульсные нейронные сети на соответствующем типе устройства.

На рисунке ниже представлена схема взаимодействия компонентов Kaspersky Neuromorphic Platform.



Архитектура Kaspersky Neuromorphic Platform

Формат 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. 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.

На рисунке ниже представлена схема загрузки объекта нейронной сети из формата SONATA и сохранение топологии сети в формат SONATA.



Загрузка и сохранение нейронной сети

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
`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
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.

Механизм обмена данными в Kaspersky Neuromorphic Platform

При работе с исходным кодом необходимо учитывать особенности механизма обмена данными в платформе.

Kaspersky Neuromorphic Platform обеспечивает обмен данными между популяциями и проекциями, а также между объектами платформы и объектами внешней среды. Обмен данными между популяциями и проекциями 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 бэкенд вычисляет синаптические воздействия. Проекции отправляют сообщения со значениями синаптических воздействий постсинаптическим популяциям нейронов.

Обмен данными между бэкендом и внешней средой

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

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

Если требуется, вы можете реализовать и использовать асинхронный режим для обмена данными между бэкендом и внешней средой.

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.

Если требуется, вы можете реализовать и использовать собственные объекты платформы для обмена данными между нейронной сетью и внешней средой.

Форматы данных, обрабатываемые платформой

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

- Заголовок файла, содержащий следующую информацию:
  - версию 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:
  - шагом исполнения нейронной сети, на котором были сгенерированы спайки;
  - индексами нейронов, сгенерировавших спайки на этом шаге.

С помощью функций 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.

О бэкенде Kaspersky Neuromorphic Platform

Платформа предоставляет несколько независимых друг от друга бэкендов для исполнения импульсных нейронных сетей на различных вычислительных устройствах.

Бэкенды Kaspersky Neuromorphic Platform выполняют следующие функции:

- Запускают импульсные нейронные сети на различных вычислителях.
- Выполняют статические проверки корректности построения импульсной нейронной сети.
- Обеспечивают обмен данными между нейронной сетью и внешней средой.

Для реализации бэкендов вы можете использовать следующие компоненты платформы:

- *Компонент core*. Предоставляет классы (например, классы проекций и популяций), необходимые для взаимодействия с бэкендом. Компонент поддерживается для языков программирования C++ и Python.
- *Компонент devices*. Предоставляет набор интерфейсов для взаимодействия с физическими устройствами. Компонент поддерживается для языка программирования C++.
- *Компонент neuron_traits*. Содержит свойства нейронов. Компонент поддерживается для языков программирования C++ и Python.
- *Компонент synapse_traits*. Содержит свойства моделей синапсов. Компонент поддерживается для языков программирования C++ и Python.
- *Компонент meta*. Предоставляет шаблоны общего назначения. Компонент поддерживается для языка программирования C++.

Для реализации бэкендов не используйте компонент фреймворка, чтобы избежать возникновения циркулярных зависимостей между фреймворком и бэкендами.

Поддерживаемые бэкенды

В Kaspersky Neuromorphic Platform поддерживаются следующие типы бэкендов:

- Бэкенды для CPU:
  - *Однопоточный бэкенд для CPU*. Бэкенд работает только в одном потоке. Вы можете использовать однопоточный бэкенд для отладки платформы и работы нейронной сети. Интерфейс к бэкенду реализован в классе `SingleThreadedCPUBackend`.
  - *Многопоточный бэкенд для CPU*. Бэкенд работает в нескольких потоках одновременно, что обеспечивает повышение скорости расчета нейронной сети. Вы можете использовать многопоточный бэкенд для обучения и исполнения импульсной нейронной сети. Интерфейс к бэкенду реализован в классе `MultiThreadedCPUBackend`.
- Бэкенд для исполнения импульсных нейронных сетей на нейроморфном процессоре Алтай-1 (далее также "бэкенд AltAI SNN"):

  Интерфейс к бэкенду реализован в классе `AltAIBackend`. Класс `AltAIBackend` предоставляет стандартные функции, предоставляемые базовым классом `Backend`.

  В качестве типа устройства вы можете выбрать как аппаратный процессор нейроморфного процессора Алтай-1, так и его программный эмулятор. Для аппаратного процессора используется фиксированная конфигурация ядер. При выборе программного эмулятора в качестве типа устройства вы можете изменить параметры конфигурации ядер нейроморфного процессора Алтай-1 с помощью метода программного эмулятора `load_core_grid_params()`, а также задать расположение портов ввода и вывода в конфигурации ядер с помощью метода `make_standart_north_port()`.

  Метод `load_core_grid_params()` необходимо вызвать перед инициализацией бэкенда.

На рисунке ниже представлена схема взаимодействия компонентов платформы при использовании бэкенда `AltAIBackend`. Импульсная нейронная сеть создается с помощью компонентов платформы. Обучение нейронной сети необходимо выполнить и обучается методом синаптической пластичности. На данный момент бэкенд `AltAIBackend` поддерживает нейроны по модели `AltAILIF` и синапсы по модели `DeltaSynapse`, при этом веса синапса должны быть целочисленными и находиться в диапазоне от `-255` до `255`.

Выполнение обучения импульсной нейронной сети доступно на центральном процессоре методом синаптической пластичности. Если требуется, вы можете реализовать и использовать собственные методы обучения и бэкенды.

Далее нейронная сеть проверяется на соответствие ограничениям нейроморфного процессора Алтай-1 и переводится в формат, поддерживаемый процессором для ее исполнения. Исполнение импульсной нейронной сети возможно как на программном эмуляторе, так и непосредственно на аппаратном процессоре Алтай-1.

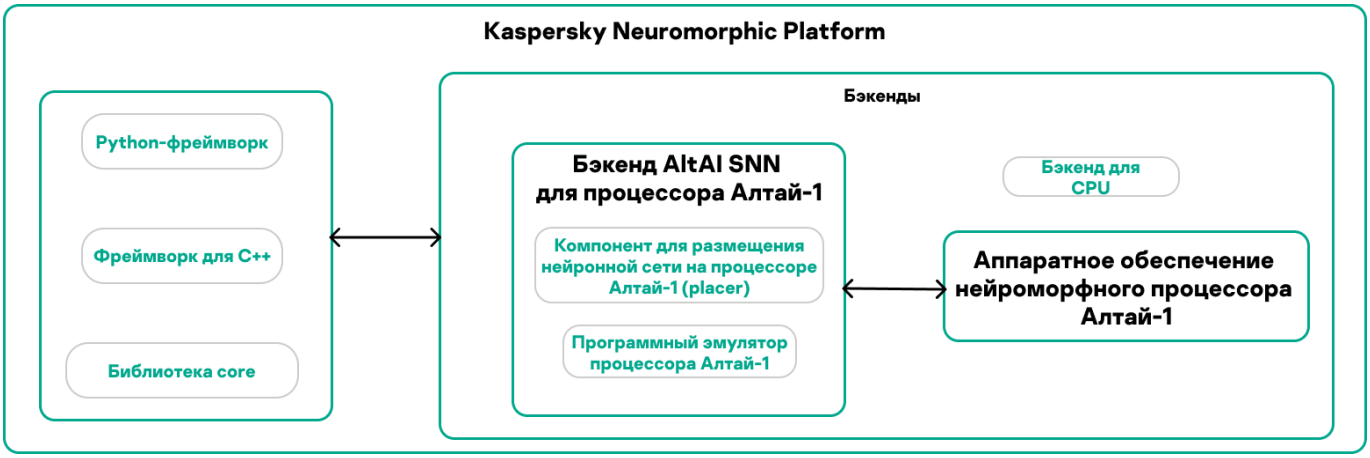


Схема взаимодействия платформы с бэкендом AltAI SNN

- Бэкенды для обучения и исполнения искусственных нейронных сетей, преобразованных в импульсные (далее также "бэкенды AltAI ANN2SNN"):
  - *Бэкенд аппаратного обеспечения нейроморфного процессора Алтай-1*;
  - *Бэкенд программного эмулятора нейроморфного процессора Алтай-1*.

Обучение нейронной сети производится обратным распространением ошибки с квантованием весов.

На рисунке ниже представлена схема взаимодействия компонентов платформы при использовании бэкендов `AltAI ANN2SNN`. Нейронная сеть создается с помощью тернарных слоев пакета `ANN2SNN`, реализованной как часть Python-фреймворка, и обучается путем обратного распространения ошибки с квантованием весов. Далее нейронная сеть размещается с помощью утилиты `placer` для ее исполнения на нейроморфном процессоре Алтай-1. Во время размещения нейронная сеть проверяется на соответствие ограничениям нейроморфного процессора и переводится в формат, поддерживаемый процессором для ее исполнения.

Бэкенды AltAI ANN2SNN реализованы в пакете, отдельном от бэкендов для CPU и бэкенда AltAI SNN.

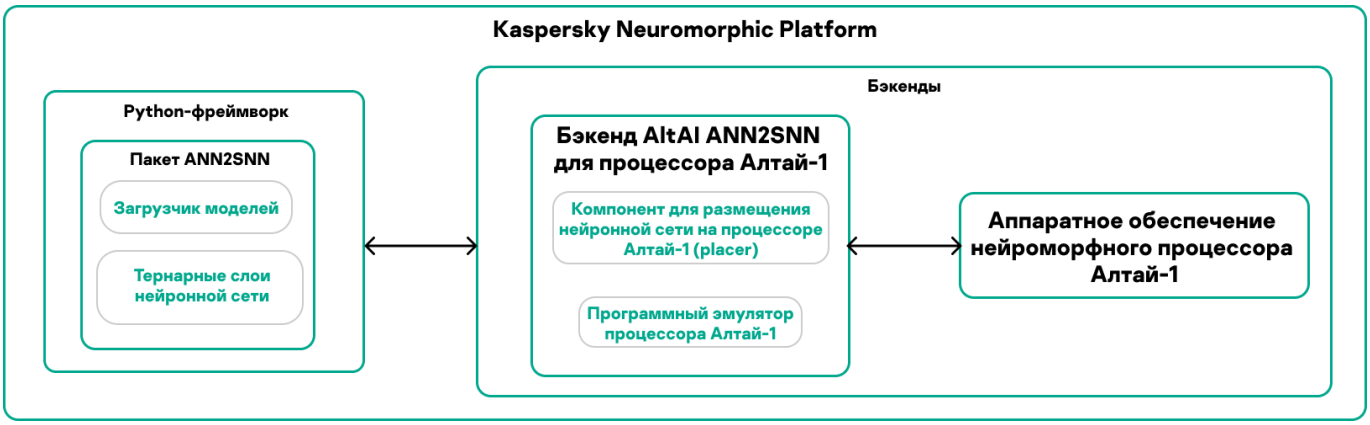


Схема взаимодействия компонентов платформы при использовании пакета AltAI ANN2SNN

О нейроморфном процессоре Алтай-1

Нейроморфный процессор Алтай-1 – это сверхбольшая интегральная схема (СБИС) с вычислительной архитектурой, которая имитирует работу биологических сенсорных и нервных систем. Нейроморфный процессор Алтай-1 предназначен для исполнения (инференса) импульсных нейронных сетей, созданных с помощью компонентов платформы или с помощью тернарных слоев пакета ANN2SNN. В архитектуре процессора Алтай-1 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-1 обладает низким энергопотреблением.

Для исполнения импульсной нейронной сети необходимо подключить объединительную плату с нейроморфным процессором Алтай-1 к компьютеру с помощью интерфейсов `PCI-E` или `USB`.

Элементарной единицей импульсной нейронной сети является нейрон, у которого есть набор входных сигнальных линий (дендриты) и выходная сигнальная линия (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-1 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 испускаются 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-1 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-1 предусмотрена возможность масштабирования за счет увеличения количества СБИС. По вопросам масштабирования вам нужно обратиться к специалистам АО "Лаборатория Касперского".

В таблице ниже приведены общие сведения о нейроморфном процессоре Алтай-1 и его составляющих:

Общие сведения о процессоре Алтай-1

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-битной 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 которой 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

О работе бэкенда для CPU

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

Цикл работы бэкенда состоит из следующих этапов:

1. Каналы ввода отправляют данные в шину сообщений `MessageBus`.
2. Бэкенд 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. Бэкенд 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.

Установка и удаление платформы

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

Для разработки прикладных решений на Linux вы можете установить платформу одним из следующих способов:

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

При работе с исходным кодом платформы на Linux или в ОС Windows вы можете собрать платформу или прикладное решение на C++. Сборка платформы может быть использована для разработки Kaspersky Neuromorphic Platform.

Установка deb-пакетов

Для разработки прикладных решений необходимо установить следующие пакеты:

- deb-пакеты с бинарным кодом бэкендов для CPU;
- deb-пакет с бинарным кодом бэкенда AltAI SNN;
- deb-пакет с бинарным кодом фреймворка для C++;
- dev-deb пакеты для каждого deb-пакета с бинарным кодом на C++;
- deb-пакет с Python-фреймворком для работы на бэкенде `AltAI SNN`;
- deb-пакет с пакетом `ANN2SNN` и бинарным кодом бэкенда AltAI ANN2SNN;
- deb-пакет с примерами использования Kaspersky Neuromorphic Platform;
- deb-пакет со справочным руководством по API.

На компьютере должно быть установлено программное обеспечение, которое указано в аппаратных и программных требованиях.

Установка deb-пакетов выполняется на Linux.

Чтобы установить deb-пакеты:

1. Для установки deb-пакетов с бинарным кодом бэкендов для CPU и бэкенда `AltAI SNN` запустите с root-правами команды:

   ```
   dpkg -i <path-to-deb-package>/knp-cpu-single-threaded-backend_<knp_version>_amd64.deb
   dpkg -i <path-to-deb-package>/knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb
   dpkg -i <path-to-deb-package>/knp-altai-backend_<motiv_version>_amd64.deb
   ```

   где:

- `<path-to-deb-package>` – путь к deb-пакету;
- `<knp_version>` – версия Kaspersky Neuromorphic Platform;
- `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`.

Во время установки пакета `knp-altai-backend_<motiv_version>_amd64.deb` примите Условия использования бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`. Согласие с Условиями использования является обязательным условием для установки пакета. Если вы не принимаете Условия использования, установка пакета будет прервана.

2. Для установки deb-пакета с бинарным кодом фреймворка для C++ запустите с root-правами команду:

```
dpkg -i <path-to-deb-package>/knp-cpp-framework_<knp_version>_amd64.deb
```

3. Для установки dev-deb пакетов запустите с root-правами команды:

```
dpkg -i <path-to-deb-package>/knp-cpu-multi-threaded-backend-dev_<knp_version>_amd64.deb
dpkg -i <path-to-deb-package>/knp-cpu-single-threaded-backend-dev_<knp_version>_amd64.deb
dpkg -i <path-to-deb-package>/knp-altai-backend-dev_<motiv_version>_amd64.deb
dpkg -i <path-to-deb-package>/knp-cpp-framework-dev_<knp_version>_amd64.deb
```

Во время установки пакета `knp-altai-backend-dev_<motiv_version>_amd64.deb` примите Условия использования бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`. Согласие с Условиями использования является обязательным условием для установки пакета. Если вы не принимаете Условия использования, установка пакета будет прервана.

4. Для установки deb-пакета с Python-фреймворком для работы на бэкенде `AltAI SNN` запустите с root-правами команду:

```
dpkg -i <path-to-deb-package>/knp-python3-framework_<knp_version>_amd64.deb
```

5. Для установки deb-пакета с пакетом `ANN2SNN` и бинарным кодом бэкенда `AltAI ANN2SNN` запустите с root-правами команду:

```
dpkg -i <path-to-deb-package>/knp_ann2snn.deb
```

Во время установки пакет `knp_ann2snn.deb` примите Условия использования бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`. Согласие с Условиями использования является обязательным условием для установки пакета. Если вы не принимаете Условия использования, установка пакета будет прервана.

6. Для установки deb-пакета с примерами использования платформы запустите с root-правами команду:

```
dpkg -i <path-to-deb-package>/knp-examples_<knp_version>_amd64.deb
```

7. Для установки deb-пакета со справочным руководством по API запустите с root-правами команду:

```
dpkg -i <path-to-deb-package>/knp-documentation_<knp_version>_all.deb
```

Установка пакетов для разработки на языке программирования Python

Для разработки на языке программирования Python необходимо установить следующие deb- или whl-пакеты:

- deb-пакеты:
  - `knp-python3-framework_<knp_version>_amd64.deb` – deb-пакет с Python-фреймворком для работы на бэкенде AltAI SNN.
  - `knp_ann2snn.deb` – deb-пакет с пакетом `ANN2SNN`, бинарным кодом бэкенда AltAI ANN2SNN для нейроморфного процессора Алтай-1 и компонентом для размещения нейронной сети на нейроморфном процессоре Алтай-1.
- whl-пакеты при установке с платформы PyPI:
  - `knp-<knp_version>-py3-none-any.whl` – whl-пакет с Python-фреймворком для работы на бэкенде `AltAI SNN`.
  - `knp_ann2snn-<motiv_version>-py3-none-any.whl` – whl-пакет с пакетом `ANN2SNN`, бинарным кодом бэкенда `AltAI ANN2SNN` для нейроморфного процессора Алтай-1 и компонентом для размещения нейронной сети на нейроморфном процессоре Алтай-1.

    > Этот пакет устанавливается на компьютеры с процессором архитектуры x86.

  - `knp_ann2snn_aarch64-<motiv_version>-py3-none-any.whl` – whl-пакет с пакетом `ANN2SNN` и бинарным кодом бэкенда `AltAI ANN2SNN` для нейроморфного процессора Алтай-1.

    > Этот пакет устанавливается на компьютеры с процессором архитектуры ARM.

  где:
  - `<knp_version>` – версия Kaspersky Neuromorphic Platform;
  - `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`.

Установка пакетов выполняется на Linux.

Чтобы установить deb-пакеты для разработки на языке программирования Python:

1. Для установки deb-пакета с Python-фреймворком для работы на бэкенде `AltAI SNN` запустите с root-правами команду:

   ```
   dpkg -i <path-to-deb-package>/knp-python3-framework_<knp_version>_amd64.deb
   ```

   где:
   - `<path-to-deb-package>` – путь к deb-пакету;

- `<knp_version>` – версия Kaspersky Neuromorphic Platform.

2. Для установки deb-пакета с пакетом `ANN2SNN` и бинарным кодом бэкенда `AltAI ANN2SNN` запустите с root-правами команду:

   ```
   dpkg -i <path-to-deb-package>/knp_ann2snn.deb
   ```

   Во время установки пакета `knp_ann2snn.deb` примите Условия использования бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`. Согласие с Условиями использования является обязательным условием для установки пакета. Если вы не принимаете Условия использования, установка пакета будет прервана.

Для установки whl-пакетов с платформы PyPI необходимо использовать менеджер пакетов `pip3`.

Чтобы установить whl-пакеты:

1. Для установки whl-пакета с Python-фреймворком для работы на бэкенде `AltAI SNN` запустите команду:

   ```
   pip3 install --break-system-packages knp-<knp_version>-py3-none-any.whl
   ```

   где `<knp_version>` – версия Kaspersky Neuromorphic Platform.

2. Для установки whl-пакета с пакетом `ANN2SNN` и бинарным кодом бэкенда `AltAI ANN2SNN` выполните одно из следующий действий:

   - Если требуется установить пакет на компьютере с процессором архитектуры x86, запустите команду:

     ```
     pip3 install --break-system-packages knp_ann2snn-<motiv_version>-py3-none-any.whl
     ```

   - Если требуется установить пакет на компьютере с процессором архитектуры ARM, запустите команду:

     ```
     pip3 install --break-system-packages knp_ann2snn_aarch64-<motiv_version>-py3-none-any.whl
     ```

   где `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`.

Сценарий сборки проекта платформы или прикладного решения

Вы можете собрать платформу или прикладное решение на C++. Сборка платформы может быть использована при разработке Kaspersky Neuromorphic Platform.

При разработке прикладных решений вы можете установить нужные deb-пакеты вместо сборки прикладного решения.

> Для сборки платформы или прикладного решения вы также можете использовать Docker-образ `knp-build-image`, входящий в комплект поставки платформы.

Сценарий сборки платформы или прикладного решения состоит из следующих этапов:

1. **Получение исходного кода платформы**

   Вы можете получить исходный код платформы одним из следующих способов:

   - Скачать архив с исходным кодом из репозитория платформы и распаковать его в рабочую директорию.
   - Склонировать репозиторий платформы.

2. **Определение параметра Boost_ROOT в ОС Windows**

   В ОС Windows определите параметр `Boost_ROOT`. Для этого создайте переменную окружения или укажите путь к установленной библиотеке Boost в файле `CMakePresets.json`, расположенном в корневой директории с исходным кодом платформы.

   Вы также можете определить путь к установленной библиотеке Boost как параметр вызова CMake с помощью следующей команды:

   ```
   cmake -DBOOST-ROOT=<путь к установленной библиотеке Boost>
   ```

3. **Настройка параметров сборки прикладного решения**

   В случае сборки прикладного решения опишите процесс сборки в файле `CMakeLists.txt`. Укажите директорию с программой, сборку которой необходимо выполнить с помощью функции `add_subdirectory`, укажите исполняемый файл с помощью функции `add_executable`, а также укажите библиотеки платформы, которые необходимо подключить к проекту, с помощью функции `target_link_libraries`. Вы можете использовать следующие библиотеки:

   - `neuron_traits`
   - `synapse_traits`
   - `core`
   - `meta`
   - `devices`
   - `backends`
   - `framework`

4. **Конфигурация сборки платформы или прикладного решения**

   Если вы собираете платформу или прикладное решение на Linux, выполните конфигурацию сборки с помощью системы сборки CMake. Подробнее о конфигурации сборки с помощью CMake см. документацию CMake.

   Если вы собираете платформу или прикладное решение в ОС Windows, выполните конфигурацию сборки в Visual Studio. Для этого откройте проект платформы или прикладного решения, выбрав нужный файл `CMakeLists.txt`, и выполните настройку кеша. В случае сборки прикладного решения необходимо выбрать файл `CMakeLists.txt` прикладного решения. В случае сборки платформы необходимо выбрать файл `CMakeLists.txt`, расположенный в корневой директории с исходным кодом платформы.

   > Первая конфигурация сборки с помощью CMake может выполняться слишком долго и завершаться с сетевыми ошибками. В случае возникновения сетевых ошибок выполните конфигурацию сборки еще раз.

5. **Запуск сборки**

Запустите сборку платформы или прикладного решения. Подробнее о запуске сборки проектов см. документацию CMake или документацию используемой интегрированной среды разработки.

Загрузка и распаковка архива с исходным кодом платформы

Чтобы загрузить и распаковать архив с исходным кодом платформы:

1. Скачайте архив с исходным кодом из 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>
   ```

Определение параметра Boost_ROOT в ОС Windows

Перед сборкой платформы или прикладного решения в ОС Windows требуется определить параметр `Boost_ROOT`. В качестве значения параметра необходимо указать путь к установленной библиотеке Boost. Для этого вам нужно указать его в параметре `Boost_ROOT` в файле `CMakePresets.json`, расположенном в корневой директории с исходным кодом платформы.

Вы также можете создать переменную окружения для параметра `Boost_ROOT`.

Чтобы создать переменную окружения в ОС Windows:

1. В ОС Windows нажмите комбинацию клавиш **Windows**+**R**.
2. Введите `sysdm.cpl` и нажмите на клавишу **Enter**.
3. В открывшемся окне выберите вкладку **Дополнительно** и нажмите на кнопку **Переменные среды**.
4. В открывшемся окне создайте переменную `Boost_ROOT` и укажите путь к библиотеке Boost в качестве значения переменной.

Настройка параметров сборки проекта прикладного решения на C++

Для сборки прикладного решения необходимо указать параметры сборки в файле `CMakeLists.txt`. 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 названия проекта.

set(CMAKE_CXX_STANDARD 17)

Указание C++ 17 в качестве используемого языкового стандарта.

add_subdirectory(<директория_с_исходным_кодом_платформы>)

Указание директории, включающей программу, сборку которой требуется выполнить.

add_executable("${PROJECT_NAME}" <файл_программы>)

Указание исполняемого файла.

target_link_libraries("${PROJECT_NAME}" PRIVATE <подключаемые_библиотеки_платформы>)

Определение библиотек Kaspersky Neuromorphic Platform, которые будут подключены к вашему проекту. Вы можете подключить библиотеки `backends`, `neuron trits`, `synapse traits`, `core`, `meta`, `devices` и `framework`.

Перед сборкой проекта в ОС Windows необходимо определить параметр `Boost_ROOT`.

Чтобы настроить параметры сборки проекта прикладного решения на C++:

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

   Пример 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("${PROJECT_NAME}" main.cpp)
   # Подключает фреймворк и библиотеку однопоточного бэкенда для CPU к проекту digits-recognition.
   target_link_libraries("${PROJECT_NAME}" PRIVATE KNP::BaseFramework::Core KNP::Backends::CPUSingleThreaded)
   ```

   При необходимости вы можете загрузить фреймворк динамически.

4. Сохраните изменения в файле `CMakeLists.txt`.

Конфигурация сборки платформы или прикладного решения в ОС Windows

В ОС Windows вы можете выполнить конфигурацию сборки платформы или прикладного решения с помощью Visual Studio.

Чтобы выполнить конфигурацию сборки платформы или прикладного решения в ОС Windows:

1. В ОС Windows запустите Visual Studio.
2. На вкладке **Файл** выберите **Открыть** → **CMake**.
3. В открывшемся окне выберите корневой файл `CMakeLists.txt`.

В случае сборки прикладного решения выберите файл `CMakeLists.txt` прикладного решения. При сборке платформы выберите файл `CMakeLists.txt`, расположенный в корневой директории с исходным кодом платформы.

4. На вкладке **Проект** выберите **Настроить кэш**.

   CMake начнет работы по конфигурации сборки.

После того, как CMake завершит работы по конфигурации, вам нужно запустить сборку.

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.

Удаление deb-пакетов

Для удаления deb-пакетов рекомендуется использовать пакетный менеджер `apt`.

Чтобы удалить deb-пакеты:

1. Для удаления deb-пакетов с бинарным кодом бэкендов для CPU и бэкенда `AltAI SNN` запустите с root-правами команды:

   ```
   apt purge knp-cpu-single-threaded-backend
   apt purge knp-cpu-multi-threaded-backend
   apt purge knp-altai-backend
   ```

2. Для удаления deb-пакета с бинарным кодом фреймворка для C++ запустите с root-правами команду:

   ```
   apt purge knp-cpp-framework
   ```

3. Для удаления dev-deb пакетов запустите с root-правами команды:

   ```
   apt purge knp-cpu-multi-threaded-backend-dev
   apt purge knp-cpu-single-threaded-backend-dev
   apt purge knp-altai-backend-dev
   apt purge knp-cpp-framework-dev
   ```

4. Для удаления deb-пакета с Python-фреймворком для работы на бэкенде `AltAI SNN` запустите с root-правами команду:

   ```
   apt purge knp-python3-framework
   ```

5. Для удаления deb-пакета с пакетом `ANN2SNN` и бинарным кодом бэкенда `AltAI ANN2SNN` запустите команду:

```
apt purge knp_ann2snn
```

6. Для удаления deb-пакета с примерами использования платформы запустите с root-правами команду:

```
apt purge knp-examples
```

7. Для удаления deb-пакета со справочным руководством по API запустите с root-правами команду:

```
apt purge knp-documentation
```

Удаление пакетов для разработки прикладных решений на Python

Для удаления deb-пакетов, используемых для разработки прикладных решений на Python, рекомендуется использовать пакетный менеджер `apt`.

Чтобы удалить deb-пакеты:

1. Для удаления deb-пакета с Python-фреймворком для работы на бэкенде `AltAI SNN` запустите с root-правами команду:

```
apt purge knp-python3-framework
```

2. Для удаления deb-пакета с пакетом `ANN2SNN` и бинарным кодом бэкенда `AltAI ANN2SNN` запустите с root-правами команду:

```
apt purge knp_ann2snn
```

Для удаления whl-пакетов необходимо использовать пакетный менеджер `pip3`.

Чтобы удалить whl-пакеты:

1. Для удаления whl-пакета с Python-фреймворком для работы на бэкенде `AltAI SNN` запустите команду:

```
pip3 uninstall knp-<knp_version>-py3-none-any.whl
```

   где `<knp_version>` – версия Kaspersky Neuromorphic Platform.

2. Для удаления whl-пакета с пакетом `ANN2SNN` и бинарным кодом бэкенда `AltAI ANN2SNN` выполните одно из следующих действий:

   - Если требуется удалить пакет с компьютера с процессором архитектуры x86, запустите команду:

```
pip3 uninstall knp_ann2snn-<motiv_version>-py3-none-any.whl
```

   - Если требуется удалить пакет с компьютера с процессором архитектуры ARM, запустите команду:

```
pip3 uninstall knp_ann2snn_aarch64-<motiv_version>-py3-none-any.whl
```

где `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`.

Удаление исходного кода платформы

Чтобы удалить исходный код платформы,

удалите рабочую директорию, в которую был распакован архив с исходным кодом платформы.

Компоненты платформы для использования на C++

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

Библиотека neuron_traits

Библиотека `neuron_traits` 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 `AltAILIF`.

  Нейрон по модели `AltAILIF` представляет собой нейрон-интегратор с утечкой (англ. Leaky Integrate-and-Fire Neuron). Нейрон характеризуется потенциалом мембраны и пороговыми значениями потенциала мембраны.

  Значение потенциала мембраны изменяется в соответствии со значением синаптического воздействия, поступающего от связанного синапса. Если потенциал мембраны нейрона достигает положительного или отрицательного порогового значения, нейрон генерирует положительный или отрицательный спайк соответственно. В зависимости от заданных атрибутов нейрона после превышения одного из порогов потенциал мембраны либо принимает значение величины, на которую был превышен порог, либо сбрасывается до заданного значения.

  Значение потенциала мембраны также изменяется линейно на значение утечки `potential_leak_`, которое в свою очередь может быть как положительным, так и отрицательным.

  В атрибутах экземпляра нейрона вы можете указать базовое значение потенциала мембраны, значение потенциала мембраны после сброса, положительное и отрицательное пороговое значение потенциала, значение утечки, а также задать флаги, управляющие поведением потенциала после превышения порогов, флаг сброса потенциала нейрона в начале каждого временного шага и флаг, управляющий поведением утечки в зависимости от текущего знака потенциала нейрона.

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

  Нейрон по модели `BLIFATNeuron` представляет собой нейрон-интегратор с утечкой и адаптивным порогом, генерирующий 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 своего порогового значения.

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

Подробнее о свойствах нейронов см. API-документацию Kaspersky Neuromorphic Platform. Если требуется, при работе с исходным кодом вы можете реализовать и использовать собственные модели нейронов.

Библиотека synapse_traits

Библиотека `synapse_traits` 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 спайков от пресинаптической популяции синапсы генерируют сообщение, содержащее значения синаптических воздействий. 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 неформированных 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: достижением максимально допустимого расстояния ISImax после последнего спайка в ППС, форсированным срабатыванием или приходом 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 интервалу времени с момента 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

Подробнее о свойствах синапсов см. API-документацию Kaspersky Neuromorphic Platform. Если требуется, при работе с исходным кодом вы можете реализовать и использовать собственные модели синапсов.

Библиотека core

Библиотека `core` предоставляет поддержку бэкендов. Библиотека `core` предоставляет классы, необходимые для взаимодействия с бэкендом и выполнения прикладных задач.

Таблица ниже содержит описание компонентов библиотеки.

Компоненты библиотеки core

Компонент	Описание

`Backend`	Базовый класс для конкретных реализаций бэкендов.
`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 к шине сообщений.
`Population`	Шаблонный класс, реализующий контейнер нейронов одной модели.
`Projection`	Шаблонный класс, реализующий контейнер синапсов одного типа.
`Subscription`	Класс, реализующий подписку на сообщения.
`TagMap`	Класс, реализующий интерфейс к словарю тегов.
`UID`	Класс, определяющий уникальные идентификаторы.
`uid_hash`	Функция, реализующая хеширование UID.
`messaging`	Пространство имен, содержащее интерфейсы к сообщениям.
`tags`	Пространство имен, содержащее названия и типы значений известных тегов.

На рисунке ниже представлена схема взаимодействия компонентов 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, реализованные в классах пространства имен `messaging`. Сообщения 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`.

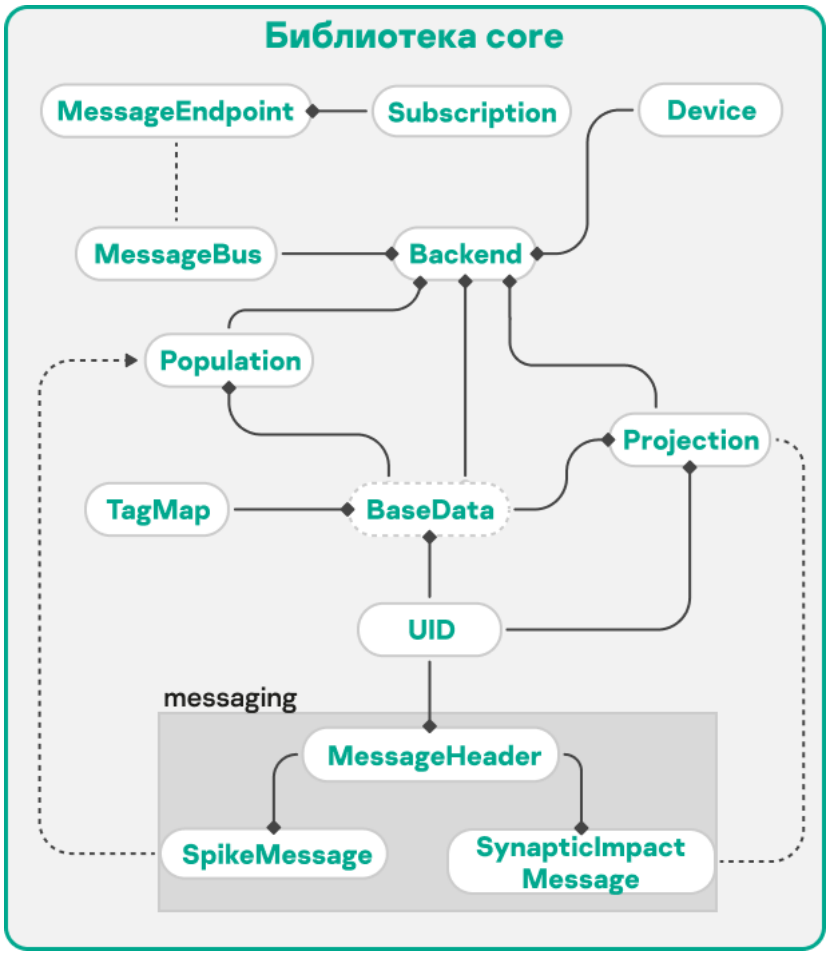


Схема взаимодействия компонентов библиотеки core

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 `core` и объектов фреймворка. Например, базовые данные `BaseData` используются в объектах классов `Backend`, `Device`, `Population` и `Projection`.

Базовые данные `BaseData` включают в себя следующие атрибуты:

- `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 `neuron_traits`.

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

Каждая популяция нейронов содержит следующие атрибуты:

- Базовые данные популяции.
- Атрибуты нейронов популяции.

Класс Projection

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

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 изменять параметры синапсов в составе проекции, а также передавать данные проекции на бэкенд и получать их от бэкенда.

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.
- Реестр соединений синапсов проекции с пресинаптическими и постсинаптическими нейронами.

  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.
- 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 бэкендов. Каждый бэкенд реализуется в виде отдельной библиотеки, с помощью которой вы можете:

- запускать обучение и исполнение 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.

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

Каждый объект класса `Backend` содержит следующие атрибуты:

- Базовые данные бэкенда.
- Шину сообщений. С помощью шины сообщений бэкенд обеспечивает обмен сообщениями между объектами библиотеки `core`, а также обмен сообщениями от модификаторов к нейронам и синапсам.

- Точку подключения к шине сообщений.
- Статус инициализации бэкенда.
- Статус исполнения нейронной сети на бэкенде.
- Устройства бэкенда.
- Номер шага исполнения нейронной сети.

Класс Device

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

Наследники класса переопределяют функции, 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 бэкенда.

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 `core` могут подписаться на сообщения или отправить их.

Класс `MessageBus` предоставляет функции, 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`, который используется для фильтрации сообщений. Шина сообщений может иметь несколько точек подключения.

С помощью функций класса `MessageEndpoint` вы можете настроить обмен сообщениями между объектами бэкенда 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 tags

В пространстве имен `tags` реализованы названия и типы значений известных тегов.

> Имена и типы значений известных тегов реализованы в пространстве имен `tags` для того, чтобы между бэкендом и фреймворком для C++ не возникало циркулярной зависимости.

Пространство имен messaging

Пространство имен `messaging` реализует интерфейсы для доступа к сообщениям. Сообщение содержит данные, которыми обмениваются объекты бэкенда, 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 бэкенде. Таблица ниже содержит описание поддерживаемых типов сообщений бэкенда.

Структуры сообщений

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. Индексы нейронов являются уникальными внутри популяции.
`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, от которого был получен спайк синапсом проекции. Индекс нейрона является уникальным внутри популяции. • `postsynaptic_neuron_index_` – индекс постсинаптического нейрона, которому будет передано значение синаптического воздействия для изменения значений его атрибутов. Индекс нейрона является уникальным внутри популяции. Вектор `impacts_` может содержать более одного набора перечисленных атрибутов для передачи синаптических воздействий нескольким нейронам одной популяции.

Библиотека meta

Библиотека `meta` представляет собой набор шаблонов общего назначения. Шаблоны библиотеки используются, например, внутри функций бэкенда, обеспечивающих интроспекцию.

Библиотека devices

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

Функции библиотеки используются в функциях бэкендов для получения информации об устройствах. Для выбора устройства, на котором будет исполняться нейронная сеть, необходимо сначала вызвать функцию бэкенда, которая предоставляет типы вычислителей. Типы вычислителей являются наследниками класса `Device` и обладают своим набором функций и свойств. Для выбора конкретного устройства из предоставленных необходимо вызвать функцию `select_devices()`, передав в нее уникальные идентификаторы устройств или указатели на экземпляры класса `Device`.

Информацию об устройстве (например, о потребляемой мощности) вы можете получить с помощью методов объекта, который обеспечивает интерфейс к устройству.

Если требуется, при работе с исходным кодом вы можете реализовать и использовать собственные устройства. Для этого в рамках бэкенда реализуйте класс устройства и предоставьте его интерфейс, а также реализуйте функцию загрузки устройства, которая возвращает указатель на экземпляр класса `Device`.

В Kaspersky Neuromorphic Platform на текущий момент реализованы интерфейсы для центрального процессора и нейроморфного процессора Алтай-1. При использовании нейроморфного процессора Алтай-1 вы можете выбрать в качестве типа устройства как аппаратный процессор, так и его программный эмулятор. Для аппаратного процессора используется фиксированная конфигурация ядер. При выборе программного эмулятора в качестве типа устройства вы можете изменить параметры конфигурации ядер нейроморфного процессора Алтай-1 с помощью метода `load_core_grid_params()`, а также задать расположение портов ввода и вывода в конфигурации ядер с помощью метода `make_standart_north_port()`.

Фреймворк для C++

Фреймворк для C++ предназначен для создания импульсных нейронных сетей и их обучения методами синаптической пластичности. Полученные в результате нейронные сети могут исполняться на центральном процессоре, нейроморфном процессоре Алтай-1 или его программном эмуляторе.

Фреймворк для C++ представляет собой независимые блоки, которые обеспечивают управление бэкендами, преобразуют представление нейронной сети в разные форматы и выбирают оптимальную нейронную сеть для запуска. Во фреймворке для C++ реализованы функции, не относящиеся непосредственно к обучению и исполнению 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 бэкенды.
- Выполнять обучение и исполнение импульсных нейронных сетей.
- Вводить и выводить данные нейронных сетей.
- Контролировать выполнение обучения и исполнения импульсных нейронных сетей.
- Реализовывать специальные алгоритмы.

Фреймворк для C++ реализован в пространстве имен `framework`. Таблица ниже содержит описание компонентов фреймворка для C++.

Компоненты фреймворка для C++

Компонент		Описание
`coordinates`		Пространство имен, которое содержит пространственные координаты нейрона.
`io`		Пространство имен, которое содержит пространства имен `input`, `output` и `storage`.
	`input`	Пространство имен, которое содержит интерфейсы для доступа к каналу ввода и преобразователю данных, поступающих из внешней среды.
	`output`	Пространство имен, которое содержит интерфейсы для доступа к каналам вывода.
	`storage`	Пространство имен, которое содержит функции загрузки и сохранения сообщений `SpikeMessage`.
`monitoring`		Пространство имен, которое содержит интерфейс к наблюдателю.
`population`		Пространство имен, которое содержит функции для генерации популяций нейронов.
`projection`		Пространство имен, которое содержит функции для генерации проекций синапсов и генерации синапсов со значениями атрибутов по умолчанию.
`sonata`		Пространство имен, которое содержит функции загрузки и сохранения нейронной сети в формате `SONATA`.
`synchronization`		Пространство имен, которое содержит функцию для получения актуального состояния нейронной сети с бэкенда.

`BackendLoader`	Класс, реализующий загрузчик динамических библиотек.
`Model`	Класс, реализующий модель.
`ModelExecutor`	Класс, реализующий исполнитель модели.
`ModelLoader`	Класс, реализующий загрузчик модели.
`Network`	Класс, реализующий объект нейронной сети.

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

В Kaspersky Neuromorphic Platform объект класса `Network` может быть загружен с помощью функции пространства имен `sonata`. Отображение объекта класса `Network` также может быть задано с помощью набора координат (произвольных или задаваемых пользователем) из пространства имен `coordinates`. Состояние объекта `Network` может быть актуализировано с бэкенда `Backend` с помощью функции пространства имен `synchronization`.

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. Сообщения `SpikeMessage` поступают в каналы ввода и выгружаются из каналов вывода с помощью функций пространства имен `storage`.

Вы можете исполнить объект класса `Model` с помощью функции `start()` объекта класса `ModelExecutor`. Объект класса `ModelExecutor` также содержит объект класса `MessageObserver` из пространства имен `monitoring`, который обрабатывает полученные сообщения в соответствии с заданной функцией, и объект класса `ModelLoader`, с помощью функций которого вы можете загружать модель на конкретный экземпляр бэкенда. Для исполнения модели на экземпляре бэкенда объекту класса `ModelExecutor` передается указатель на экземпляр бэкенда, полученный от объекта класса `BackendLoader`.

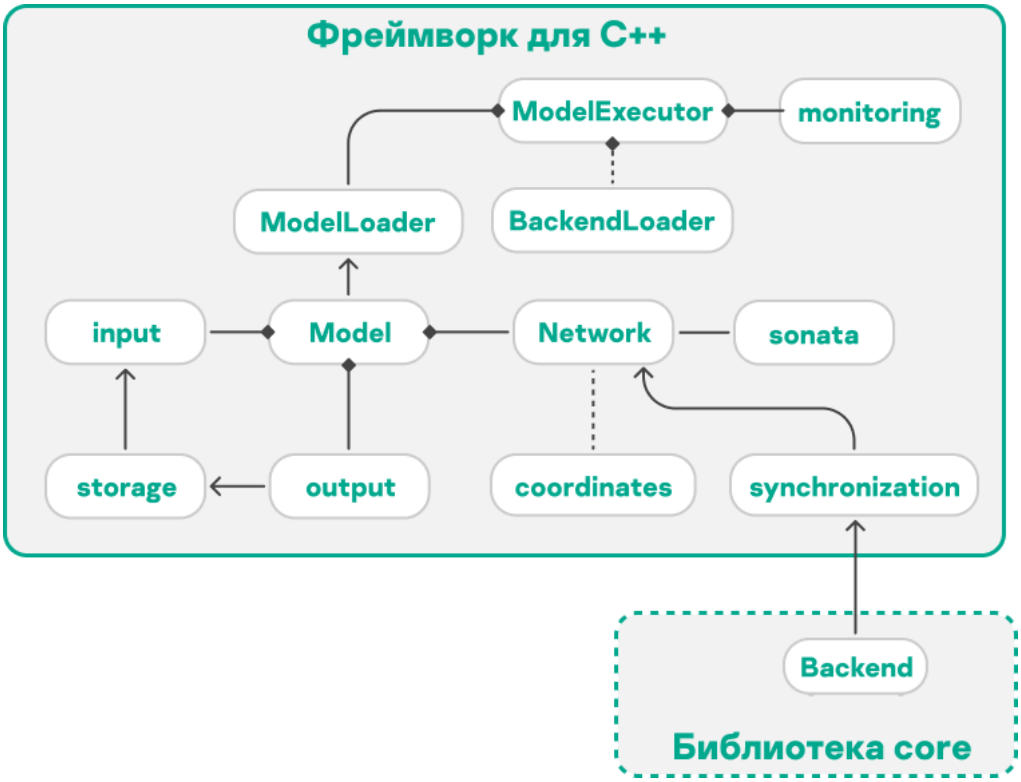


Схема взаимодействия компонентов фреймворка для C++

Пространство имен sonata

В пространстве имен `sonata` реализованы функции для загрузки и сохранения нейронной сети в формате `SONATA`. Пространство имен предоставляет следующие функции:

- `load_network()` – для загрузки нейронной сети. Топология нейронной сети должна быть описана в формате `SONATA`.
- `save_network()` – для сохранения топологии нейронной сети в формате `SONATA`.

Пространство имен coordinates

В пространстве имен `coordinates` реализованы 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 система координат для двухмерного и трехмерного пространства, а также полярная и сферическая системы координат.

Если требуется, вы можете реализовать и использовать собственные системы координат.

Пространство имен synchronization

В пространстве имен `synchronization` реализована функция `get_network_copy()` для получения актуального состояния нейронной сети с бэкенда. Например, вы можете использовать эту функцию для сохранения параметров нейронной сети при изменении качества ее предсказаний или через каждые несколько шагов ее исполнения. Вы можете сравнить качество сохраненных копий нейронной сети для выбора наиболее оптимальной для ее загрузки на устройство.

Пространство имен storage

Пространство имен `storage` содержит пространство имен `native`, в котором реализованы функции для загрузки и сохранения сообщений `SpikeMessage`. Пространство имен предоставляет следующие функции:

- `load_messages_from_json` – для загрузки сообщений из JSON-файла.
- `load_messages_from_h5` – для загрузки сообщений из HDF5-файла.
- `save_messages_to_json` – для сохранения сообщений в JSON-файл.
- `save_messages_to_h5` – для сохранения сообщений в HDF5-файл.

Пространство имен population

Пространств имен `population` содержит пространства имен `creators` и `neurons_generators`.

Пространство имен creators

В пространстве имен creators реализованы функции для генерации популяций. Пространство имен предоставляет следующие функции:

- `from_container` – для генерации популяции нейронов определенного типа из контейнера, содержащего атрибуты нейронов.
- `make_random` – для генерации популяции нейронов определенного типа со случайными значениями атрибутов.
- `make_default` – для генерации популяции нейронов определенного типа со значениями атрибутов по умолчанию.
- `make_clone` – для генерации популяции нейронов определенного типа, значения атрибутов которых скопированы с атрибутов заданного нейрона.

Пространство имен neurons_generators

В пространстве имен `neurons_generators` реализованы функции и интерфейсы для генерации нейронов. Пространство имен предоставляет следующие функции и интерфейсы:

- Функцию `from_container` – для генерации нейронов определенного типа из контейнера, содержащего атрибуты нейронов.
- Класс `MakeRandom` – для генерации нейронов определенного типа со случайными значениями атрибутов.
- Функцию `make_default` – для генерации нейронов определенного типа со значениями атрибутов по умолчанию.
- Функцию `make_clone` – для генерации нейронов определенного типа, значения атрибутов которых скопированы с атрибутов заданного нейрона.

Пространство имен projection

Пространство имен `projection` содержит пространства имен `creators`, `synapse_generators` и `parameters_generators`.

В пространстве имен также реализована функция `connect_populations` для связи двух проекций. В результате функция создает проекцию, связывающую эти две популяции, и добавляет ее в объект нейронной сети. С помощью этой функции вы можете связать между собой две популяции одним из следующих образов:

- В результирующей проекции связи между нейронами популяций произвольны.

- В результирующей проекции каждый нейрон пресинаптической популяции связан с каждым нейроном постсинаптической популяции.

Пространство имен creators

В пространство имен `creators` реализованы функции для генерации проекций синапсов. Пространство имен предоставляет следующие функции:

- `all_to_all` – генерирует проекцию синапсов, соединяющих каждый нейрон пресинаптической популяции с каждым нейроном постсинаптической популяции.
- `one_to_one` – генерирует проекцию синапсов, соединяющих нейроны пресинаптической и постсинаптической популяций в отношении один-к-одному.
- `from_container` – генерирует проекцию синапсов из контейнера, содержащего атрибуты синапсов и индексы пресинаптического и постсинаптического нейронов.
- `from_map` – генерирует проекцию синапсов из объекта `std::map`, содержащего атрибуты синапсов в качестве значения и индексы пресинаптического и постсинаптического нейронов в качестве ключа.
- `fixed_probability` – генерирует проекцию синапсов, соединяющих каждый нейрон пресинаптической популяции с каждым нейроном постсинаптической популяции. Синапсы генерируются с заданной вероятностью.
- `index_based` – генерирует проекцию синапсов, соединяющих нейроны популяций, если функция генерации синапса возвращает ненулевое `std::optional` значение. В функцию генерации синапса передаются индексы пресинаптического и постсинаптического нейронов.
- `fixed_number_post` – генерирует проекцию синапсов, соединяющих каждый нейрон пресинаптической популяции с заданным количеством случайных нейронов постсинаптической популяции.
- `fixed_number_pre` – генерирует проекцию синапсов, соединяющих каждый нейрон постсинаптической популяции с заданным количеством случайных нейронов пресинаптической популяции.
- `clone_projection` – генерируется проекцию, синапсы которой скопированы из другой проекции. Исходная и созданная проекции могут быть разных типов.

Пространство имен synapse_generators

В пространстве имен `synapse_generators` реализованы функции и интерфейсы для генерации синапсов. Пространство имен предоставляет следующие функции и интерфейсы:

- Функцию `all_to_all` – генерирует синапсы между каждым нейроном пресинаптической популяции и каждым нейроном постсинаптической популяции.
- Функцию `one_to_one` – генерирует синапсы между нейронами пресинаптической и постсинаптической популяций в отношении один-к-одному.
- Функцию `from_container` – генерирует синапсы из контейнера, содержащего атрибуты синапсов и индексы пресинаптического и постсинаптического нейронов.
- Класс `FromMap` – генерирует синапсы из объекта `std::map` или `std::unordered_map`, содержащего атрибуты синапсов в качестве значения и индексы пресинаптического и постсинаптического нейронов в качестве ключа.

- Класс `FixedProbability` – генерирует синапсы между каждым нейроном пресинаптической популяции и каждым нейроном постсинаптической популяции. Синапсы генерируются с заданной вероятностью.
- Функцию `index_based` – генерирует синапсы между нейронами популяций, если функция генерации синапса возвращает ненулевое `std::optional` значение. В функцию генерации синапса передаются индексы пресинаптического и постсинаптического нейронов.
- Класс `FixedNumberPost` – генерирует синапсы между каждым нейроном пресинаптической популяции и заданным количеством случайных нейронов постсинаптической популяции.
- Класс `FixedNumberPre` – генерирует синапсы между каждым нейроном постсинаптической популяции и заданным количеством случайных нейронов пресинаптической популяции.
- Функцию `clone_projection` – генерируется синапсы, атрибуты которого скопированы с другого синапса. Исходный и созданный синапс могут быть разных типов.

Пространства имен parameters_generators

В пространстве имен реализованы функция `default_synapse_gen` и класс `CopySynapseGen`. Функция `default_synapse_gen` генерирует синапс со значениями атрибутов по умолчанию. Класс `CopySynapseGen` реализует генератор синапсов, который копирует атрибуты у заданного синапса.

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 функций пространства имен `input` преобразователь данных приводит значения во входных отрезках к типу значений `boolean`. Если входное значение содержит спайк, функция возвращает значение `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 функций 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 к шине сообщений, с помощью которой преобразованные данные отправляются подписчикам.
- Преобразователь данных в сообщения `SpikeMessage`.

Пространство имен output

В пространство имен `output` реализован интерфейс для доступа к каналу 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.

Пространство имен monitoring

Пространство имен `monitoring` содержит 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 к шине сообщений, 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 функции, с помощью которых вы можете добавлять и удалять популяции и проекции нейронной сети, а также соединять их между собой.

Класс Model

Класс `Model` реализует модель, хранящую нейронную сеть и каналы ввода и вывода и связывающую их друг с другом. Для исполнения модели на разных бэкендах вы можете загрузить объект модели в объект класса `ModelExecutor` и вызвать функцию исполнения модели `start()`.

С помощью функций класса `Model` вы можете управлять объектом нейронной сети. Например, вы можете добавить в объект нейронной сети каналы ввода и вывода, связанные с определенными проекциями.

Каждый объект класса `Model` содержит следующие атрибуты:

- Базовые данные модели.

- Нейронную сеть.
- Идентификаторы каналов ввода и вывода.

Класс BackendLoader

Класс `BackendLoader` реализует загрузчик динамических библиотек, возвращающий указатель на экземпляр загруженного бэкенда. Указатель на экземпляр бэкенда передается в исполнитель модели `ModelExecutor` для ее исполнения на бэкенде.

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

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

Класс ModelLoader

Класс `ModelLoader` реализует загрузчик моделей, который загружает модель на нужный экземпляр бэкенда. С помощью функций класса `ModelLoader` вы можете загружать модель на бэкенд, а также получать сведения о каналах ввода и вывода, используемых в модели.

Каждый объект класса `ModelLoader` содержит следующие атрибуты:

- Базовые данные загрузчика моделей.
- Экземпляр бэкенда, на который требуется загрузить модель.
- Карту канала ввода, содержащую идентификатор, хешированный идентификатор и преобразователь входных данных.
- Каналы ввода и вывода.

Класс ModelExecutor

Класс `ModelExecutor` реализует исполнитель моделей. С помощью класса `ModelExecutor` вы можете исполнить загруженную модель на разных бэкендах. Для исполнения модели класс `ModelExecutor` использует загрузчик моделей `ModelLoader`, а также указатель на экземпляр бэкенда, полученный от объекта класса `BackendLoader`.

Каждый объект класса `ModelExecutor` содержит следующие атрибуты:

- Базовые данные исполнителя модели.
- Загрузчик моделей.
- Наблюдатели.

Решение типовых задач на C++

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

Дополнительно вы можете найти примеры использования платформы в директории `/usr/src/knp/examples` после установки пакета `knp-examples` или в репозитории платформы. В директории с каждым примером содержится файл `README.md`, в котором представлено описание примера.

Список примеров для C++:

- `mnist-client` – пример классификации изображений рукописных чисел из базы данных MNIST.
- `save-load-networks` – пример сохранения и загрузки нейронной сети в формате `SONATA`.
- `simple-network` – пример создания и исполнения нейронной сети, загруженной на бэкенд автоматически.

Добавление нового типа нейрона

На текущий момент Kaspersky Neuromorphic Platform поддерживает наборы свойств и типизацию популяций по моделям нейронов `AltAILIF`, `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. Из рабочей директории, в которую был распакован архив с исходный кодом платформы, перейдите в директорию `neuron-trait-library/include/knp/neuron-traits/` и создайте в ней 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>

{
    // Указание количества 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();
    // Указание значения по умолчанию, к которому стремится мембранный потенциал для
    // тормозных синапсов, основанных на проводимости.
    constexpr static double reverse_inhibitory_potential = -0.3;
    // Указание значения 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>

{
    // Указание количества 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_;
    // Указание порогового значения мембранного потенциала нейрона.
    double activation_threshold_ = default_values<BLIFATNeuron>::activation_threshold_;
    // Указание динамического порогового значения 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 бэкендах перегруженный метод запуска.

   Например, для бэкенда 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. Из рабочей директории, в которую был распакован архив с исходный кодом платформы, перейдите в директорию `synapse-traits-library/include/knp/synapse-traits/` и создайте заголовочный файл для нового типа синапса.

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

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


template <>
struct synapse_parameters<DeltaSynapse>

{
    // Указание атрибутов 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 бэкендах перегруженный метод запуска.

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

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 бэкенд для CPU 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 бэкенде 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++
   // ...
   // Реализация функции, генерирующей 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++
   // ...
   // Реализация функции, генерирующей 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};
   };

   // Реализация функции, генерирующей 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++

// ...
// Создание объекта 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++

// ...
// Создание объекта 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++

// ...
// Создание объекта 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
};

// Получение 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 бэкенд.

   Пример загрузки популяции и проекций на бэкенд:

```
C++

// ...
Backend backend;
// ...
// Загрузка на бэкенд созданного объекта популяции.
backend.load_populations({population});

// Загрузка на бэкенд созданных объектов проекций.
backend.load_projections({input_projection, loop_projection});
```

9. Определите объект точки подключения к шине сообщений.

   Пример создания точки подключения к шине сообщений:

C++

```
// ...
// Создание точки подключения к шине сообщений.
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++

```
// ...
// Создание произвольного UID канала ввода.
knp::core::UID in_channel_uid;

// Создание произвольного UID канала вывода.
knp::core::UID out_channel_uid;

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

// Создание подписки 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)
{
    // Отправка спайков 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();
    // Получение спайков, отправленных популяцией на канал вывода.
    auto output = endpoint.unload_messages<knp::core::messaging::SpikeMessage>(out_channel_uid);

    // Запись индексов 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.

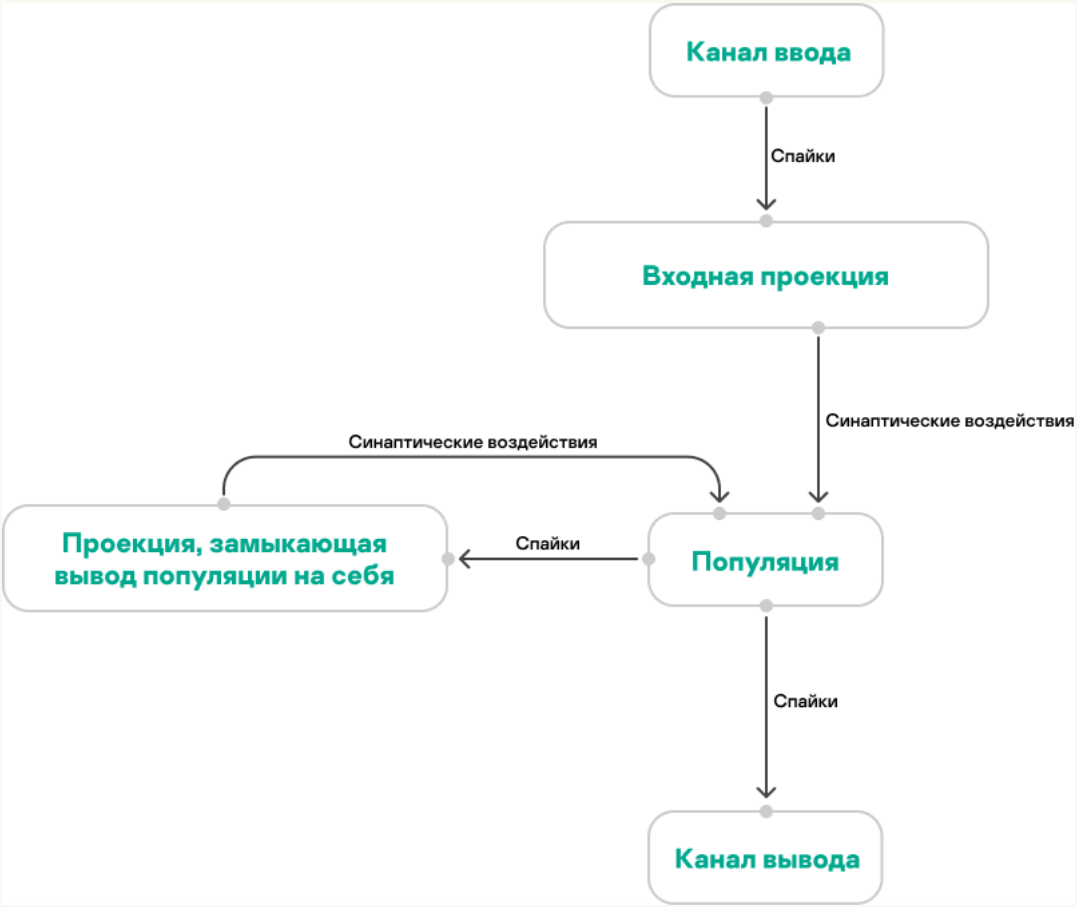


Схема исполнения нейронной сети

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.

12. Сохраните созданный файл.

Если требуется, соберите прикладное решение с помощью CMake.

Компоненты платформы для использования на Python

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

Модуль neuron_traits

Модуль `neuron_traits` представляет собой набор атрибутов нейронов. Вы можете использовать атрибуты нейронов, представленных в модуле, например, при построении популяций.

На текущий момент модуль 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, генерирующий 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, при работе с исходным кодом вы можете реализовать и использовать собственные модели нейронов.

Модуль synapse_traits

Модуль `synapse_traits` представляет собой набор атрибутов синапсов. Вы можете использовать синапсы, представленные в модуле, например, при построении проекций.

На текущий момент модуль поддерживает наборы свойств для следующих синапсов:

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

  После получения спайков от пресинаптической популяции синапсы генерируют сообщение, содержащее значения синаптических воздействий. 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 `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 неформированных 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.

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

    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.

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

    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, при работе с исходным кодом вы можете реализовать и использовать собственные модели синапсов.

Модуль core

Модуль `core` предоставляет поддержку бэкендов. Модуль `core` предоставляет классы, необходимые для взаимодействия с бэкендом и выполнения прикладных задач.

Таблица ниже содержит описание компонентов модуля.

Компоненты модуля core

Компонент	Описание
`Backend`	Базовый класс для конкретных реализаций бэкендов.
BackendWrapper	Структура, реализующая обертку для базового класса Backend.
`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 к шине сообщений.
`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.
`messaging`	Пространство имен, содержащее интерфейсы к сообщениям.

На рисунке ниже представлена схема взаимодействия компонентов модуля.

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, реализованные в классах пространства имен `messaging`. Сообщения содержат в своем заголовке UID отправителя. Обмен сообщений осуществляется с помощью функций 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`.

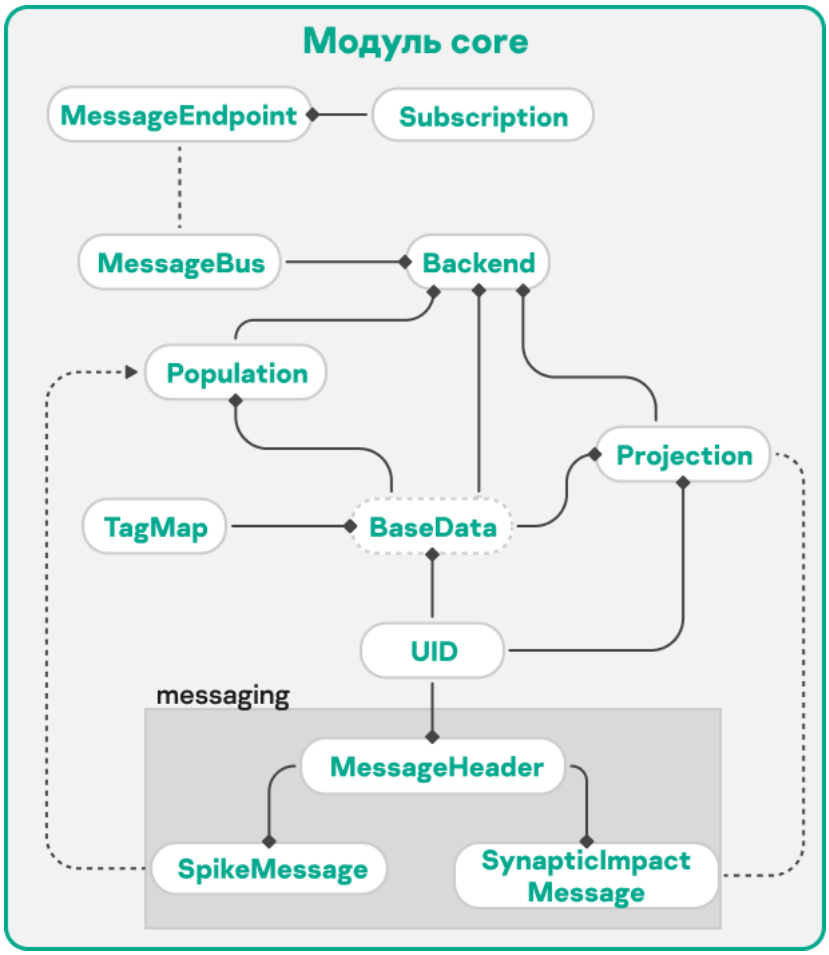


Схема взаимодействия компонентов модуля core

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 модуля `core` и объектов фреймворка. Например, базовые данные `BaseData` используются в объектах классов `Backend`, `Population` и `Projection`.

Базовые данные `BaseData` имеют следующие свойства:

- Уникальный идентификатор объекта.
- 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 модулем `neuron_traits`. Каждая популяция нейронов имеет уникальный идентификатор.

С помощью набора классов `Population` вы можете управлять нейронами в составе популяции. Вы также можете загрузить популяции на бэкенд для вычисления нейронной сети. Полученные свойства популяций хранятся на бэкенде. Чтобы передать новые свойства от бэкенда в соответствующие объекты популяций, вы можете вызвать синхронизацию. Если вы вручную изменили свойства популяции до вызова синхронизации, вы можете заново загрузить популяцию на бэкенд для вычисления нейронной сети.

Набор классов Projection

Набор классов `Projection` реализует контейнеры, каждый из которых содержит синапсы одного из типов, поддерживаемых модулем `synapse_traits`. 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 бэкенд и получать их от бэкенда обратно.

Класс Backend

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

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

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

Каждый объект класса `Backend` имеет следующие свойства:

- Уникальный идентификатор бэкенда.
- Шину сообщений. С помощью шины сообщений бэкенд обеспечивает обмен сообщениями между объектами модуля `core`, а также обмен сообщениями от модификаторов к нейронам и синапсам.
- Точку подключения к шине сообщений.
- Статус исполнения нейронной сети на бэкенде.

Структура BackendWrapper

Структура `BackendWrapper` реализует обертку для базового класса `Backend` для использования его объектов на языке программирования Python.

С помощью структуры `BackendWrapper` вы можете управлять поддержкой синаптической пластичности для объекта класса `Backend`, получать информацию о поддерживаемых экземпляром бэкенда нейронах, синапсах, популяциях и проекциях, а также об используемых устройствах и точках подключений к шине сообщений. Вы также можете загружать в экземпляр бэкенда и удалять из него нужные проекции синапсов и популяции нейронов для исполнения нейронной сети и управлять обучением нейронной сети.

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 к шине сообщений, с помощью которых объекты модуля `core` могут подписаться на сообщения или отправить их.

Класс `MessageBus` предоставляет функции, 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 функций класса `MessageEndpoint` вы можете настроить обмен сообщениями между объектами бэкенда 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 функций 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 бэкенда, 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 бэкенде. Таблица ниже содержит описание поддерживаемых типов сообщений бэкенда.

Структуры сообщений

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. Индексы нейронов являются уникальными внутри популяции.
`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, от которого был получен спайк синапсом проекции. Индекс нейрона является уникальным внутри популяции. • `postsynaptic_neuron_index` – индекс постсинаптического нейрона, которому будет передано значение синаптического воздействия для изменения значений его атрибутов. Индекс нейрона является уникальным внутри популяции. Вектор `impacts` может содержать более одного набора перечисленных атрибутов для передачи синаптических воздействий нескольким нейронам одной популяции.

Python-фреймворк

Python-фреймворк предназначен для создания импульсных нейронных сетей, а также преобразования искусственных нейронных сетей в импульсные. Созданные импульсные нейронные сети обучаются методами синаптической пластичности и могут быть исполнены на центральном процессоре, нейроморфном процессоре Алтай-1 или его программном эмуляторе. Искусственные нейронные сети, преобразованные в импульсные с помощью тернарных слоев пакета `ANN2SNN`, обучаются методом обратного распространения ошибки с учетом ограничений нейроморфного процессора Алтай-1. Полученные в результате нейронные сети адаптированы для исполнения на нейроморфном процессоре Алтай-1 или его программном эмуляторе.

Python-фреймворк представляет собой независимые блоки, которые обеспечивают управление бэкендами, преобразуют представление нейронной сети в разные форматы и выбирают оптимальную нейронную сеть для запуска. В Python-фреймворке реализованы функции, не относящиеся непосредственно к обучению и исполнению импульсных нейронных сетей.

С помощью Python-фреймворка Kaspersky Neuromorphic Platform вы можете динамически загружать различные бэкенды.

Python-фреймворк реализован в модуле `python-framework`. Таблица ниже содержит описание компонентов Python-фреймворка.

Компоненты Python-фреймворка

Компонент	Описание
`ANN2SNN`	Пакет, реализующий тернарные слои нейронной сети.
`input`	Пространство имен, в котором реализован интерфейс для доступа к каналу ввода.
`output`	Пространство имен, в котором реализован интерфейс для доступа к каналу вывода.
`sonata`	Пространство имен, которое содержит функции загрузки и сохранения нейронной сети в формате `SONATA`.
`BackendLoader`	Класс, реализующий загрузчик динамических библиотек.
`ImpactMessageObserver`	Класс, реализующий наблюдатель за сообщениями `SynapticImpactMessage`.
`Model`	Класс, реализующий модель.
`ModelExecutor`	Класс, реализующий исполнитель моделей.
`ModelLoader`	Класс, реализующий загрузчик модели.
`Network`	Класс, реализующий объект нейронной сети.
`SpikeMessageObserver`	Класс, реализующий наблюдатель за сообщениями `SpikeMessage`.

Пакет ANN2SNN

Пакет `ANN2SNN` предназначен для создания нейронных сетей, которые можно преобразовать в импульсные нейронные сети и разместить для их исполнения на нейроморфном процессоре Алтай-1. Пакет `ANN2SNN` реализован на языке программирования Python.

Для создания нейронных сетей необходимо использовать специальные тернарные слои, в которых учитываются ограничения, характерные для нейроморфного процессора Алтай-1. 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 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 нейронная сеть проверяется на соответствие ограничениям процессора. Проверка нейронной сети производится относительно параметров конфигурации нейроморфного процессора, указанных в YAML-файле. В результате размещения выдается информация о количестве ядер, необходимых для запуска нейронной сети на нейроморфном процессоре Алтай-1. Нейронная сеть переводится в формат, поддерживаемый процессором для ее дальнейшего запуска на аппаратном процессоре или его программном эмуляторе. Если нейронную сеть невозможно исполнить на нейроморфном процессоре из-за превышения допустимого количества ядер, выдается соответствующая ошибка.

> В нейроморфном процессоре Алтай-1 предусмотрена возможность масштабирования за счет увеличения количества СБИС. По вопросам масштабирования вам нужно обратиться к специалистам "Лаборатории Касперского".

Вы можете провести примерную оценку количества ядер, необходимых для исполнения нейронной сети, 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 оценки количества ядер

Количество ядер, необходимых для размещения полносвязного слоя `TernaryDense`, зависит от количества входных нейронов в слое. Если количество входных нейронов не больше 512, то вы можете рассчитать необходимое количество ядер по формуле, представленной 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 полносвязных слоев нейронной сети.
- *U* соответствует количеству целевых нейронов.

$$x = \sum_{i=1}^{n} \frac{U}{512}$$

Формула оценки количества ядер для слоя TernaryDense

Если количество входных нейронов больше 512, то вы можете рассчитать необходимое количество ядер по формуле, представленной 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 полносвязных слоев нейронной сети.
- *U* соответствует количеству целевых нейронов.
- *N* соответствует количеству входных нейронов для слоя.

$$x = \sum_{i=1}^{n} \frac{U}{512} * (\frac{N}{512} + 1)$$

Формула оценки количества ядер для слоя TernaryDense

Для нейронных сетей, созданных на базе пакета `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 пакета ANN2SNN

В пакете `ANN2SNN` поддерживаются следующие тернарные слои:

Классы пакета ANN2SNN, реализующие тернарные слои.

Класс	Описание
`TernaryDense`	Реализует 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-1.
`TernaryConv1D`	Реализует 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`.
`TernaryConv2D`	Реализует 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`.
`TernaryLocallyConnected1D`	Реализует 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`.
`TernaryLocallyConnected2D`	Реализует 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`.
`TernaryDepthwiseConv1D`	Реализует 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`.
`TernaryDepthwiseConv2D`	Реализует 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`.
`MaxPool1D`	Поддерживается стандартная реализация слоя `MaxPool1D` 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.
`MaxPool2D`	Поддерживается стандартная реализация слоя `MaxPool2D` 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.
`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`.
`InputLayer`	Поддерживается стандартная реализация входного слоя `InputLayer` библиотеки `Keras`. Входные данные должны иметь бинарный тип.
`Flatten`	Поддерживается стандартная реализация слоя `Flatten` библиотеки `Keras`.
`Reshape`	Поддерживается стандартная реализация слоя `Reshape` библиотеки `Keras`.
`Concatenate`	Поддерживается стандартная реализация слоя `Concatenate` библиотеки `Keras`.

Для размещения нейронной сети на нейроморфном процессоре Алтай-1 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`).

Пространство имен sonata

В пространстве имен `sonata` реализованы функции для загрузки и сохранения нейронной сети в формате `SONATA`. Пространство имен предоставляет следующие функции:

- `load_network()` – для загрузки нейронной сети. Топология нейронной сети должна быть описана в формате `SONATA`.
- `save_network()` – для сохранения топологии нейронной сети в формате `SONATA`.

Пространство имен input

В пространстве имен `input` реализован интерфейс для доступа к каналу ввода `InputChannel`. Канал ввода принимает сообщения `SpikeMessage` от генератора сообщений. С помощью функций класса `InputChannel` полученные сообщения могут быть переданы подписчикам через точки подключения при наличии подписки.

Пространство имен output

В пространстве имен `output` реализован интерфейс для доступа к каналу вывода `OutputChannel`. С помощью функций класса `OutputChannel` вы можете читать сообщения из буфера сообщений, а также обновлять его сообщениями, полученными от точек подключения.

Класс BackendLoader

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

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

Класс Network

Класс `Network` реализует объект нейронной сети. Объект нейронной сети хранит 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.

С помощью функций класса `Network` вы можете добавлять и удалять популяции и проекции нейронной сети.

Каждый объект класса `Network` имеет следующие свойства:

- Количество популяций в нейронной сети.
- Количество проекций в нейронной сети.

Класс Model

Класс `Model` реализует модель, хранящую нейронную сеть и каналы ввода и вывода и связывающую их друг с другом. Для исполнения модели на разных бэкендах вы можете загрузить объект модели в объект класса `ModelExecutor` и вызвать функцию исполнения модели `start()`.

Класс SpikeMessageObserver

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

Класс ImpactMessageObserver

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

Класс ModelLoader

Класс `ModelLoader` реализует загрузчик моделей, который загружает модель на нужный экземпляр бэкенда. С помощью функций класса `ModelLoader` вы можете загружать модель на бэкенд, а также получать сведения о каналах ввода и вывода, используемых в модели.

Класс ModelExecutor

Класс `ModelExecutor` реализует исполнитель моделей. С помощью класса `ModelExecutor` вы можете исполнить загруженную модель на разных бэкендах. Для исполнения модели класс `ModelExecutor` использует указатель на экземпляр бэкенда, полученный от объекта класса `BackendLoader`.

Решение типовых задач на Python

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

Для Python предоставлен пример mnist-client, содержащий пример классификации изображений рукописных чисел из базы данных MNIST.

Исполнение нейронной сети, загруженной на бэкенд вручную

Этот раздел содержит инструкции по созданию нейронной сети, ее загрузке на бэкенд для CPU и исполнению.

> Вы можете использовать эту инструкцию для разработки прикладных решений.

Чтобы автоматически загрузить нейронную сеть на бэкенд 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 бэкенде 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
   # Реализация функции, генерирующей 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
   # Реализация функции, генерирующей 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
   ```

```
# Реализация функции, генерирующей 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

    # ...
    # Создание объекта 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

    # ...
    # Создание объекта 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

    # ...
    # Создание объекта 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
    )

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

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

   Пример загрузки бэкенда и создания точки подключения:

```
Python
```

```
# ...
# Загрузка экземпляра однопоточного бэкенда для CPU с локального компьютера.
backend = BackendLoader().load(f'{pytestconfig.rootdir}/../lib/libknp-cpu-single-threaded-backend')

# Загрузка созданной популяции в экземпляр бэкенда.
backend.load_all_populations([population])
# Загрузка созданных проекций в экземпляр бэкенда.
backend.load_all_projections([input_projection, loop_projection])

# Инициализация бэкенда.
backend._init()
# Создание объекта 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

```
# ...
# Создание произвольных идентификаторов 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 бэкенд и точку подключения на сообщения от каналов ввода и вывода.

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

Python

```
# ...
# Подписка бэкенда на спайки от канала ввода.
backend.subscribe(SpikeMessage, input_uid, [in_channel_uid])
# Подписка точки 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

```
# ...
# Создание списка results, в который будут записываться индексы шагов со спайками.
results = []

for step in range(0, 20):
    # Отправка спайков на шагах 0, 5, 10 и 15 исполнения нейронной сети.
    if step % 5 == 0:
        # Отображение номера 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):
        # Отображение номера 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.

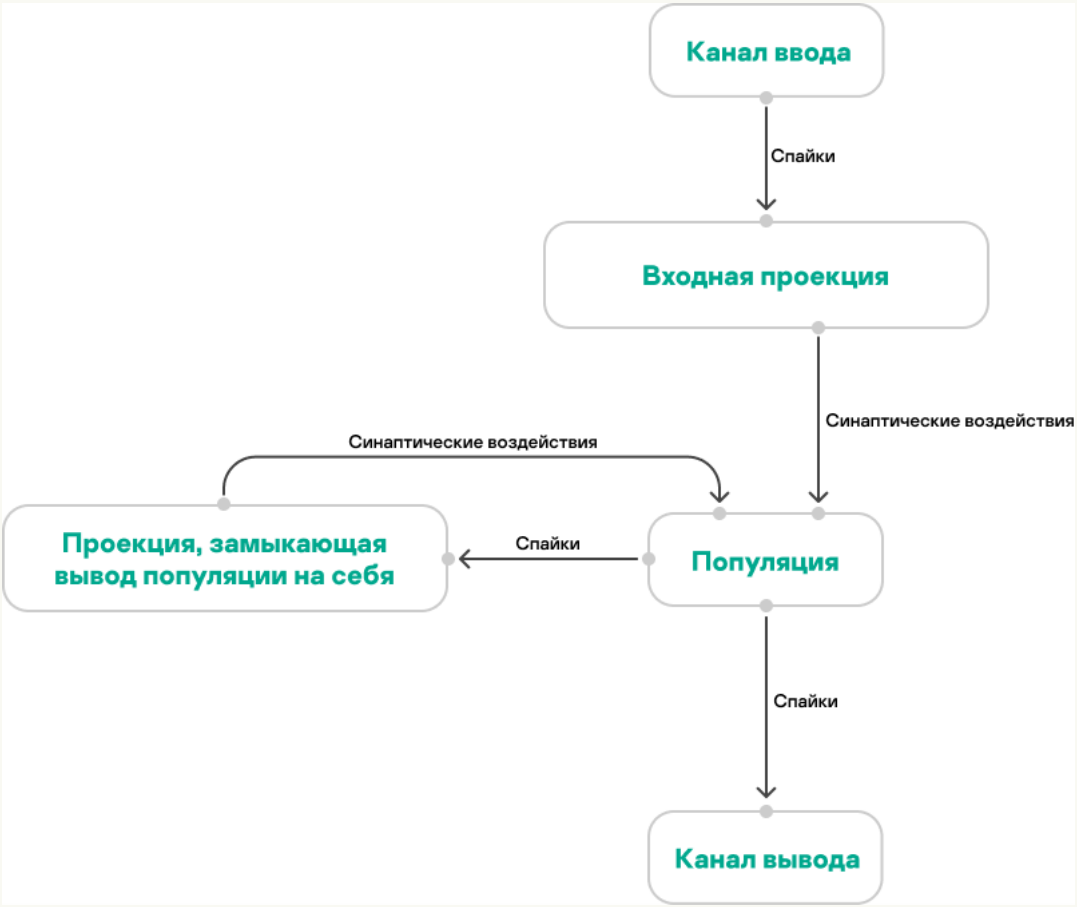


Схема исполнения нейронной сети

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.

13. Запустите программу.

Сценарий исполнения нейронной сети из тернарных слоев для классификации изображений

Вы можете использовать этот сценарий при разработке прикладных решений.

В этом разделе приводится пример создания, обучения, размещения нейронной сети для классификации изображений и ее исполнения на нейроморфном процессоре Алтай-1. Нейронная сеть состоит из тернарных слоев пакета `ANN2SNN`.

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 пакета `ANN2SNN`. В качестве функции активации выходного слоя укажите функцию `softmax`. Подготовьте данные для обучения нейронной сети. Приведите данные для обучения в бинарный вид. Обучите нейронную сеть и оцените результаты обучения.

2. **Размещение нейронной сети**

   Укажите параметры конфигурации нейроморфного процессора Алтай-1 в YAML-файле. Разместите нейронную сеть с помощью следующей команды:

   ```
   placer -p <NN_file> -l <placer_log> -o <altai_config> -hw <hw_config>
   ```

   В этой команде:

   - `<NN_file>` – путь к файлу с обученной нейронной сетью;
   - `<placer_log>` – путь к файлу, в который будут записываться журналы утилиты `placer`;
   - `<altai_config>` – путь к файлу, в который будет сохранена конфигурация для загрузки на нейроморфный процессор;
   - `<hw_config>` – путь к файлу, содержащему конфигурацию нейроморфного процессора. Если флаг `-hw` не указан, используется конфигурация нейроморфного процессора по умолчанию.

   Пример:

   ```
   placer -p mnist.alt -l mnist_placer.log -o mnist.json
   ```

3. **Запуск инференса нейронной сети**

Подключите объединительную плату с нейроморфным процессором Алтай-1 к компьютеру с помощью интерфейсов `PCI-E` или `USB`.

Укажите конфигурационный файл и компонент для исполнения нейронной сети. Нейронная сеть может быть исполнена как на аппаратном процессоре Алтай-1, так и на его 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 пакета `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 knp_ann2snn.altainn import TernaryConv2D, TernaryDense
   from knp_ann2snn.altainn import heaviside
   from knp_ann2snn.altainn import Clip

   from tensorflow import keras
   from keras.models import Sequential, load_model
   from keras.layers import Flatten
   from keras.callbacks import ModelCheckpoint

   import tensorflow as tf
   ```

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

   Пример создания нейронной сети с тремя сверточными слоями:

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

4. Подготовьте данные для обучения.

   Пример подготовки данных MNIST для обучения:

   ```Python
   # Загрузка данных MNIST для обучения.
   (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
   ```

```
# Кодирование целевых меток.
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

# Нормализация входных данных.
x_train = x_train / 255
x_test = x_test / 255
# Приведение входных данных 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
# Замена размерности данных (28, 28) на (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
# Сохранение обученной нейронной сети в файл mnist.h5.
model_chekpoint_callback = ModelCheckpoint(
    filepath="mnist.alt",
    monitor="accuracy",
    mode="max",
    save_best_only=True,
)
```

6. Обучите нейронную сеть.

   Пример обучения нейронной сети:

```Python
# Запуск обучения на тренировочных данных.
model.fit(x_train, y_train, epochs = 15, verbose = 1, callbacks=[model_chekpoint_callback])
# Загрузка обученной модели.
model = load_model(
    "mnist.alt",
    custom_objects = {
        "heaviside_mod": heaviside,
        "TernaryDense": TernaryDense,
        "TernaryConv2D": TernaryConv2D,
        "Clip": Clip
    }
)
# Оценка точности нейронной сети.
print(f"Точность модели: {model.evaluate(x_test, y_test, verbose = 1)[1]*100:.2f}%")
```

7. Запустите программу.

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-1. Нейронную сеть требуется предварительно создать с помощью тернарных слоев пакета `ANN2SNN`, обучить и разместить для исполнения на нейроморфном процессоре.

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

1. Подключите объединительную плату с нейроморфным процессором Алтай-1 к компьютеру с помощью интерфейсов `PCI-E` или `USB`.

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

3. 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 knp_ann2snn import Altai
import tensorflow as tf
import numpy as np
from tqdm import tqdm
```

4. Создайте функцию для подсчета точности модели.

   Пример функции для подсчета точности модели:

```python
def calculate_accuracy(array1,array2):

    correct_predictions = sum(a == b for a, b in zip(array1,array2))
    accuracy = correct_predictions / len(array1)

    return accuracy
```

5. Подготовьте данные для инференса.

   Пример подготовки данных MNIST для инференса:

```python
# Загрузка модели.
altai = Altai()
# Укажите путь к конфигурационному файлу для загрузки на нейроморфный процессор
# и компонент для исполнения нейронной сети:
# hw – для инференса нейронной сети на аппаратном процессоре Алтай-1
# gm – для инференса нейронной сети на программном эмуляторе.
# Если функция активации на последнем слое - softmax, необходимо ее указать.
altai.build("mnist.json", "gm", activation="softmax")


# Загрузка данных MNIST для инференса.
(_, _), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Нормализация входных данных.
x_test = x_test / 255

# Приведение входных данных в бинарный вид.
x_test[x_test > 0.5] = 1
x_test[x_test <= 0.5] = 0

x_test = x_test.reshape(x_test.shape + (1,))
```

6. Запустите инференс.

   Пример инференса:

```python
pred = []

for i in tqdm(range(x_test.shape[0])):
    # Преобразование входного вектора в спайки.
    altai.prepare_spikes(x_test[i])
```

```
    # Запуск генерации сигнала тик.
    altai.start_ticks(2)
    # Получение спайков от нейронной сети.
    spikes_idx = altai.get_spikes()
    # Берем -1 индекс из-за того, что интересует результат последнего тика.
    predicted_class = spikes_idx[-1]
    pred.append(predicted_class)

# Оценка точности нейронной сети.
print("Accuracy: ", calculate_accuracy(y_test, pred)* 100,"%")
```

7. Запустите программу.

Сценарий исполнения регрессионной модели с обучаемыми блоками, кодирующие и декодирующие данные

Вы можете использовать этот сценарий при разработке прикладных решений.

В этом разделе приводится пример создания, обучения, размещения регрессионной модели и ее исполнения на нейроморфном процессоре Алтай-1.

Сценарий исполнения регрессионной модели с обучаемыми блоками, кодирующие и декодирующие данные, состоит из следующих этапов:

1. **Создание и обучение регрессионной модели**

   Создайте регрессионную модель, состоящую из кодирующего блока, нейронной сети и декодирующего блока. Подготовьте данные для обучения модели. Обучите модель и оцените результаты обучения.

2. **Размещение нейронной сети**

   Укажите параметры конфигурации нейроморфного процессора Алтай-1 в YAML-файле. Разместите нейронную сеть с помощью следующей команды:

   `placer -p <NN_file> -l <placer_log> -o <altai_config> -hw <hw_config>`

   В этой команде:

   - `<NN_file>` – путь к файлу с обученной нейронной сетью;
   - `<placer_log>` – путь к файлу, в который будут записываться журналы утилиты `placer`;
   - `<altai_config>` – путь к файлу, в который будет сохранена конфигурация для загрузки на нейроморфный процессор;
   - `<hw_config>` – путь к файлу, содержащему конфигурацию нейроморфного процессора. Если флаг `-hw` не указан, используется конфигурация нейроморфного процессора по умолчанию.

   Пример:

   `placer -p snn_sin.keras -l sin_placer.log -o sin.json`

3. **Запуск инференса регрессионной модели**

   Подключите объединительную плату с нейроморфным процессором Алтай-1 к компьютеру с помощью интерфейсов `PCI-E` или `USB`.

Укажите конфигурационный файл и компонент для исполнения нейронной сети. Нейронная сеть может быть исполнена как на аппаратном процессоре Алтай-1, так и на его программном эмуляторе. Запустите инференс регрессионной модели.

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 пакета `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 knp_ann2snn.altainn import TernaryDense
   from knp_ann2snn.altainn import  heaviside
   from knp_ann2snn.altainn import Clip

   from tensorflow import keras
   from keras.models import Sequential, load_model
   from keras.callbacks import ModelCheckpoint
   from keras.layers import Dense


   import matplotlib.pyplot as plt
   import numpy as np
   ```

3. Создайте синусоиду и подготовьте данные для обучения.

   Пример подготовки данных:

   ```Python
   # Определение количества элементов в наборе данных.
   N = 4000
   # Определение индекса, до которого элементы набора данных будут использования для обучения.
   train_idx = 3000
   # Определение индекса, до которого элементы, начиная с индекса train_idx, образуют тестовые данные.
   test_idx = 4000
   # Определение количества элементов в пакете (англ. batch).
   elements_in_batch = 10
   # Определение размерности выхода кодирующего блока.
   encoder_output_shape = 25
   # Определение размерности выхода нейронной сети.
   snn_output_shape = 35
   # Определение размерности выхода декодирующего блока.
   decoder_output_shape = 1


   # Создание синусоиды.
   x = np.arange(N)
   arr = np.sin(x)
   ```

```
# Формирование тренировочных данных на основе elements_in_batch значений.
train_range = range(train_idx - elements_in_batch - 1)
x_train = np.array([arr[i : i + elements_in_batch] for i in train_range])
y_train = np.array([[arr[i + elements_in_batch + 1]] for i in train_range])


# Формирование тестовых данных на основе elements_in_batch значений.
test_range = range(train_idx, test_idx - elements_in_batch - 1)
x_test = np.array([arr[i : i + elements_in_batch] for i in test_range])
y_test = np.array([[arr[i + elements_in_batch + 1]] for i in test_range])
```

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

   Пример создания регрессионной модели:

Python

```
# Создание кодирующего блока с одним слоем, который преобразует данные в спайки.
encoder = Sequential(
    [
        Dense(
            encoder_output_shape,
            activation=heaviside,
            input_shape=(elements_in_batch,)
        )
    ]
)


# Создание импульсной нейронной сети.
snn = Sequential(
    [
        TernaryDense(
            snn_output_shape,
            activation=heaviside,
            input_shape=(encoder_output_shape,),
            use_bias=False
        )
    ]
)


# Создание декодирующего блока с одним слоем, который преобразует спайки в целевое значение.
decoder = Sequential(
    [
        Dense(
            decoder_output_shape,
            activation="linear",
            input_shape=(snn_output_shape,)
        )
    ]
)

# Создание модели 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

```
# Обучение модели на тренировочных данных.
checkpoint = ModelCheckpoint(
    "sin_model.keras",
    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=[])

model.save("sin_model.keras")
```

```
# Загрузка модели.
model = load_model(
    "sin_model.keras",
    custom_objects={
        "heaviside_mod": heaviside,
        "Clip": Clip,
        "TernaryDense": TernaryDense,
    },
)

# Получение предсказаний.
predict_test = model.predict(x_test)
predict_test = predict_test.flatten()

# Построение графиков по предсказанным и целевым значениям.
plt.plot(predict_test, "b")
plt.plot(y_test, "r", alpha=0.5)
plt.savefig("sin_train.png")
```

6. Сохраните кодирующий блок, нейронную сеть и декодирующий блок.

   Пример сохранения:

```python
# Сохранение кодирующего блока в файл encoder_sin.keras.
model.layers[0].save("encoder_sin.keras")
# Сохранение нейронной сети в файл snn_sin.keras.
model.layers[1].save("snn_sin.keras")
# Сохранение декодирующего блока в файл decoder_sin.keras.
model.layers[2].save("decoder_sin.keras")
```

7. Запустите программу.

Запуск инференса регрессионной модели

Этот раздел содержит инструкции по запуску инференса регрессионной модели с кодирующим и декодирующим блоком. Модель требуется предварительно создать, обучить и разместить для исполнения на нейроморфном процессоре.

Чтобы запустить инференс регрессионной модели:

1. Подключите объединительную плату с нейроморфным процессором Алтай-1 к компьютеру с помощью интерфейсов `PCI-E` или `USB`.

2. В директории вашего проекта создайте файл программы с расширением `.py`, в котором будет реализован инференс модели.

3. В файле программы импортируйте модули библиотек, необходимые для инференса модели

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

   Пример подключения объектов для инференса модели:

```python
from knp_ann2snn.altainn import TernaryDense
from knp_ann2snn.altainn import heaviside
from knp_ann2snn.altainn import Clip
from knp_ann2snn.python_altai import Altai

from keras.models import load_model
import matplotlib.pyplot as plt
```

```
from tqdm import tqdm
import numpy as np
```

4. Создайте синусоиду и подготовьте данные для инференса.

   Пример подготовки данных:

```Python
# Определение количества элементов в наборе данных.
N = 4000
# Определение индекса, до которого элементы набора данных будут использования для обучения.
train_idx = 3000
# Определение индекса, до которого элементы, начиная с индекса train_idx, образуют тестовые данные.
test_idx = 4000
# Определение количества элементов в пакете (англ. batch).
elements_in_batch = 10
# Определение размерности выхода кодирующего блока.
encoder_output_shape = 25
# Определение размерности выхода нейронной сети.
snn_output_shape = 35
# Определение размерности выхода декодирующего блока.
decoder_output_shape = 1


# Создание синусоиды.
x = np.arange(N)
arr = np.sin(x)


# Формирование тестовых данных на основе elements_in_batch значений.
test_range = range(train_idx, test_idx - elements_in_batch - 1)
x_test = np.array([arr[i : i + elements_in_batch] for i in test_range])
y_test = np.array([[arr[i + elements_in_batch + 1]] for i in test_range])
```

5. Загрузите модель, кодирующий и декодирующий блок.

   Пример загрузки модели, кодирующего и декодирующего блока:

```Python
altai = Altai()
# Укажите путь к конфигурационному файлу для загрузки на нейроморфный процессор
# и компонент для исполнения нейронной сети:
# hw – для инференса нейронной сети на аппаратном процессоре Алтай-1
# gm – для инференса нейронной сети на программном эмуляторе.
altai.build("sin.json","gm")


encoder = load_model(
    "encoder_sin.keras",
    custom_objects={
        "heaviside_mod": heaviside
        }
    )

snn_model = load_model(
    "snn_sin.keras",
    custom_objects={
        "TernaryDense": TernaryDense,
        "heaviside_mod": heaviside,
        "Clip": Clip
        }
    )

decoder = load_model("decoder_sin.keras")
```

6. Запустите инференс.

   Пример запуска инференса:

```
Python
predict = np.zeros(len(x_test))

# Запуск цикла предсказаний.

for i in tqdm(range(len(x_test))):

    # Предсказание кодирующего блока.
    encoded = encoder.predict_on_batch(x_test[i:i+1])

    # Предсказание нейронной сети.
    altai.prepare_spikes(encoded)
    altai.start_ticks(1)

    spikes_idx = altai.get_spikes()
    spikes_idx = spikes_idx[spikes_idx!=-2147483647]
    spikes = np.zeros(snn_output_shape)
    spikes[spikes_idx] = 1

    # Предсказание декодирующего блока.
    decoded = decoder.predict_on_batch(spikes.reshape(1,-1))

    # Обновление массива предсказаний.
    predict[i] = decoded
    predict = predict.reshape(len(predict))
    y_test = y_test.reshape(len(y_test))
```

7. Оцените результаты инференса. Для этого вы можете сравнить значения синусоиды с предсказаниями модели на графике.

   Пример построения графика для оценки результатов инференса:

```
Python

# Построение графиков.
plt.plot(predict, "b")
plt.plot(y_test, "r", alpha=0.5)
plt.savefig("sin_inference.png")
plt.show()
```

8. Запустите программу.

Использование Kaspersky Neuromorphic Platform API

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

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

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



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

Участие в разработке платформы

Вы можете участвовать в разработке Kaspersky Neuromorphic Platform. Для этого необходимо получить исходный код из 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. Например, в ОС Windows вы можете использовать Visual Studio.

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 <репозиторий_платформы>`

   где `<репозиторий_платформы>` соответствует веб-адресу репозитория платформы, полученному от сотрудников "Лаборатории Касперского".

   b. Установите Python-пакеты с помощью следующей команды:

   ```
   pip3 -r requirements.txt
   ```

   c. Если требуется, загрузите удаленный репозиторий платформы в ваш локальный репозиторий. Для этого вы можете использовать следующую команду:

   ```
   git pull
   ```

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

   `git checkout -b <название_ветки>`

4. Внесите изменения в код платформы.

5. 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>
   ```

6. 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. Kaspersky Neuromorphic Platform распространяется на условиях лицензии Apache 2.0.

Условия использования Kaspersky Neuromorphic Platform распространяются на исходный код платформы, а также следующие пакеты:

- `knp-cpu-single-threaded-backend_<knp_version>_amd64.deb` – deb-пакет с бинарным кодом однопоточного бэкенда для CPU.
- `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb` – deb-пакет с бинарным кодом многопоточного бэкенда для CPU.
- `knp-cpp-framework_<knp_version>_amd64.deb` – deb-пакет с бинарным кодом фреймворка для C++.
- `knp-cpu-multi-threaded-backend-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb`.
- `knp-cpu-single-threaded-backend-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpu-multi-threaded-backend_<knp_version>_amd64.deb`.
- `knp-cpp-framework-dev_<knp_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-cpp-framework_<knp_version>_amd64.deb`.
- `knp-python3-framework_<knp_version>_amd64.deb` – deb-пакет с Python-фреймворком для работы на бэкенде `AltAI SNN`.
- `knp-examples_<knp_version>_amd64.deb` – deb-пакет с примерами использования Kaspersky Neuromorphic Platform.
- `knp-documentation_<knp_version>_all.deb` – deb-пакет со справочным руководством по API.
- `knp-<knp_version>-py3-none-any.whl` – whl-пакет с Python-фреймворком для работы на бэкенде `AltAI SNN`.

  где `<knp_version>` – версия Kaspersky Neuromorphic Platform.

Вы можете ознакомиться с Условиями использования Kaspersky Neuromorphic Platform, прочитав документ `LICENSE.txt`, расположенный в следующих директориях:

- В корневой директории репозитория платформы.
- В директории `/usr/share/doc/knp` после установки пакета `knp-cpp-framework_<knp_version>_amd64.deb`.
- В директории `/usr/lib/python<Python version>/knp` после установки пакета `knp-<knp_version>-py3-none-any.whl`.
- В корневой директории Docker-образа `knp-build-image` для сборки платформы или прикладного решения.

  где:

  - `<Python version>` – версия установленного Python;

- `<knp_version>` – версия Kaspersky Neuromorphic Platform.

Условия использования бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN` – это обязательное соглашение между вами и ООО "Мотив НТ", в котором изложены условия, на которых вы можете пользоваться следующими пакетами:

- `knp-altai-backend_<motiv_version>_amd64.deb` – deb-пакет с бинарным кодом бэкенда `AltAI SNN` для нейроморфного процессора Алтай-1;
- `knp-altai-backend-dev_<motiv_version>_amd64.deb` – dev-deb пакет для deb-пакета `knp-altai-backend_<motiv_version>_amd64.deb`;
- `knp_ann2snn.deb` – deb-пакет с пакетом `ANN2SNN` и бинарным кодом бэкенда `AltAI ANN2SNN`;
- `knp_ann2snn-<motiv_version>-py3-none-any.whl` – whl-пакет с пакетом `ANN2SNN` и бинарным кодом бэкенда `AltAI ANN2SNN` для установки на компьютер с процессором архитектуры x86;
- `knp_ann2snn_aarch64-<motiv_version>-py3-none-any.whl` – whl-пакет с пакетом `ANN2SNN` и бинарным кодом бэкенда `AltAI ANN2SNN` для установки на компьютер с процессором архитектуры ARM.

  где `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`.

Вы можете ознакомиться с Условиями использования следующими способами:

- Во время установки deb-пакетов.
- В документе `LICENSE.txt`, расположенном в директории `/usr/share/doc/altai-backend/`, после установки пакетов `knp-altai-backend_<motiv_version>_amd64.deb` или `knp-altai-backend_<motiv_version>_amd64.deb`.
- В документе `LICENSE.txt`, расположенном в директории `/usr/share/doc/knp_ann2snn/`, после установки пакета `knp_ann2snn.deb`.
- В документе `LICENSE.txt`, расположенном в директории `/usr/lib/python<Python version>/site-packages/knp_ann2snn/`, после установки любого из whl-пакетов.

  где:

  - `<Python version>` – версия установленного Python;
  - `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`.

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

Если вы не согласны с Условиями использования, вы должны удалить и не использовать Kaspersky Neuromorphic Platform. Если вы не принимаете Условия использования бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN` во время установки deb-пакетов установка пакетов будет прервана.

О предоставлении данных

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

Приложение. Параметры и пример YAML-файла, содержащего конфигурацию нейроморфного процессора Алтай-1

Параметры конфигурационного файла необходимо задавать для исполнения нейронной сети на бэкенде `AltAI ANN2SNN`.

В конфигурационном файле задается информация о количестве СБИС, ядер, нейронов и синапсов в нейроморфном процессоре Алтай-1. Конфигурационный файл содержит следующие параметры:

- `chip` – параметр, включающий параметр `cores`. В параметре `cores` задается количество ядер в одной СБИС.
- `core` – группа параметров, описывающих ядро с помощью следующих параметров:
  - `neurons` – количество нейронов в ядре.
  - `axons` – количество входных линий.
  - `synapses` – количество синапсов в ядре.
- `row` – параметр, включающий параметр `chips`. В параметре `chips` задается количество СБИС в одном модуле объединительной платы нейроморфного процессора Алтай-1.
- `col` – параметр, включающий параметр `chips`. В параметре `chips` задается количество модулей в объединительной плате нейроморфном процессоре Алтай-1.
- `not_working_chips` – массив с номерам СБИС, которые не будут участвовать в исполнении нейронной сети на нейроморфном процессоре, например, из-за их отсутствия на модулях объединительной платы.

  Этот параметр заполняется квалифицированным техническим специалистом Заказчика, сотрудником "Лаборатории Касперского" или сертифицированным интегратором.

Ниже приведен пример YAML-файла, содержащего конфигурацию нейроморфного процессора Алтай-1 по умолчанию. В примере ниже объединительная плата нейроморфного процессора состоит из 4 модулей по 8 СБИС в каждом модуле. Каждая СБИС содержит 16 ядер, каждое из которых включает в себя по 512 нейронов, синапсов и входных линий.

hw_config.yaml

```yaml
chip:
  cores: 16
core:
  neurons: 512
  axons: 512
  synapses: 512
row:
  chips: 8
col:
  chips: 4
not_working_chips: []
```

Глоссарий

Artificial Neural Network (ANN)

Нейронная сеть на основе искусственных нейронов, выполняющих одномоментную суммацию входных сигналов, представленных числами и взвешенных согласно весам входных связей, и генерацию выходного числового сигнала согласно заданной функции активации.

Spiking Neural Network (SNN)

Нейронная сеть на основе искусственных нейронов, выполняющих временную суммацию входных сигналов, представленных импульсами (спайками) и взвешенных согласно весам входных связей, и генерацию выходного импульса согласно определенной пороговой функции активации.

Бэкенд

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

Бэкенд AltAI ANN2SNN

Бэкенд для обучения и исполнения искусственных нейронных сетей, преобразованных в импульсные, на нейроморфном процессоре Алтай-1.

Бэкенд AltAI SNN

Бэкенд для исполнения импульсных нейронных сетей на нейроморфном процессоре Алтай-1.

Вес синапса

Значение сигнала, который синапс передает нейронам связанной популяции.

Внешняя среда

Совокупность внешних по отношению к Kaspersky Neuromorphic Platform элементов, с которыми взаимодействует платформа.

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

Программная часть платформы, определяющая ее структуру и обеспечивающая управление бэкендами. 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 Kaspersky Neuromorphic Platform содержится в файлах `NOTICE.txt`, расположенных в следующих директориях:

- При работе с исходным кодом платформы в корневой директории репозитория платформы.
- При работе с Docker-образом `knp-build-image` в корневой директории Docker-образа.
- После установки пакета `knp-cpp-framework_<knp_version>_amd64.deb` в директории `/usr/share/doc/knp`.
- После установки пакета `knp-<knp_version>-py3-none-any.whl` в директории `/usr/lib/python<Python version>/knp`.

  где:

  - `<Python version>` – версия установленного Python;
  - `<knp_version>` – версия Kaspersky Neuromorphic Platform.

Информация о стороннем коде пакетов, поставляемых ООО "Мотив НТ", содержится в файле `NOTICE.txt`, расположенном в следующих директориях:

- `/usr/share/doc/altai-backend/` после установки пакетов `knp-altai-backend_<motiv_version>_amd64.deb` или `knp-altai-backend-dev_<motiv_version>_amd64.deb`.
- `/usr/share/doc/knp_ann2snn/` после установки пакета `knp_ann2snn.deb`.
- `/lib/python<Python version>/site-packages/knp_ann2snn/` после установки пакетов `knp_ann2snn-<motiv_version>-py3-none-any.whl` или `knp_ann2snn_aarch64-<motiv_version>-py3-none-any.whl`.

  где:

  - `<Python version>` – версия установленного Python;
  - `<motiv_version>` – версия бэкендов для нейроморфного процессора Алтай-1 и пакета `ANN2SNN`.

Уведомления о товарных знаках

Зарегистрированные товарные знаки и знаки обслуживания являются собственностью их правообладателей.

Apache является либо зарегистрированным товарным знаком, либо товарным знаком Apache Software Foundation.

Ubuntu и LTS являются зарегистрированными товарными знаками Canonical Ltd.

Docker и логотип Docker являются товарными знаками или зарегистрированными товарными знаками компании Docker, Inc. в США и/или других странах. Docker, Inc. и другие стороны могут также иметь права на товарные знаки, описанные другими терминами, используемыми в настоящем документе.

TensorFlow и любые связанные с ним обозначения являются товарными знаками Google LLC.

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.

Microsoft, Visual Studio и Windows являются товарными знаками группы компаний Microsoft.

JavaScript – зарегистрированный товарный знак компании Oracle и/или аффилированных компаний.

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

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