Оглавление

ВВЕДЕНИЕ

В последнее время широкое растространение получила концепция реализации полупроводниковых сетей не как программного обеспечения, работающего на базе фон Неймановских процессоров, а в виде самостоятельных полупроводниковых устройств. Основной проблемой, возникающей при реализации нейронной сети как программного обеспечения, является разделение программы и памяти. Это критично для работы нейронов, так как от скорости, с которой мы можем прочитать значение веса синапса, зависит и скорость работы самого нейрона. Кроме того, бинарная фон Неймановская архитектура плохо подходит для хранения состояния нейрона и весов, которые в идеале должны иметь возможность непрерывного изменения.

На данный момент лучшим выходом из ситуации считается реализация нейронов на основе мемристоров и имитирующих их структур. Мемристор, являясь элементом с памятью, может иметь сотни устойчивых состояний сопротивления, которые мы можем устанавливать по желанию. Таким образом один мемристор может реализовывать один синапс, заменяя в этом качестве несколько сотен транзисторов.

Этот подход сопряжен с определенными трудностями, так как программное обеспечение для моделирования нейронных сетей такой структуры ещё не разработано. Существующие фреймворки и библиотеки, в основном, направлены на оптимизацию работы нейросети на каком-то отдельном процессоре (как TensorFlow). В связи с этим перед нами встала задача разработки собственного программного обеспечения, способного моделировать работу таких нейронных сетей.

О разрабатываемом ПО

Разрабатываемая на нынешний момент программа должна стать отправной точкой решения этой задачи. В ходе работы внимание было уделено поддержанию логичности и структурированности кода, согласованию принципа работы системы и её реального прототипа и удержанию баланса между этими целями и простотой и понятностью кода. Главным же принципом разработки служил принцип KISS («Не усложняй, придурок»).

В рамках программы нейронная сеть представляется в иерархическом виде. Главными строительными блоками являются классы, реализующие функционал нейрона, слоя нейронов, нейронной сети, таймера, логгера и эмулятора потока данных из внешнего мира.

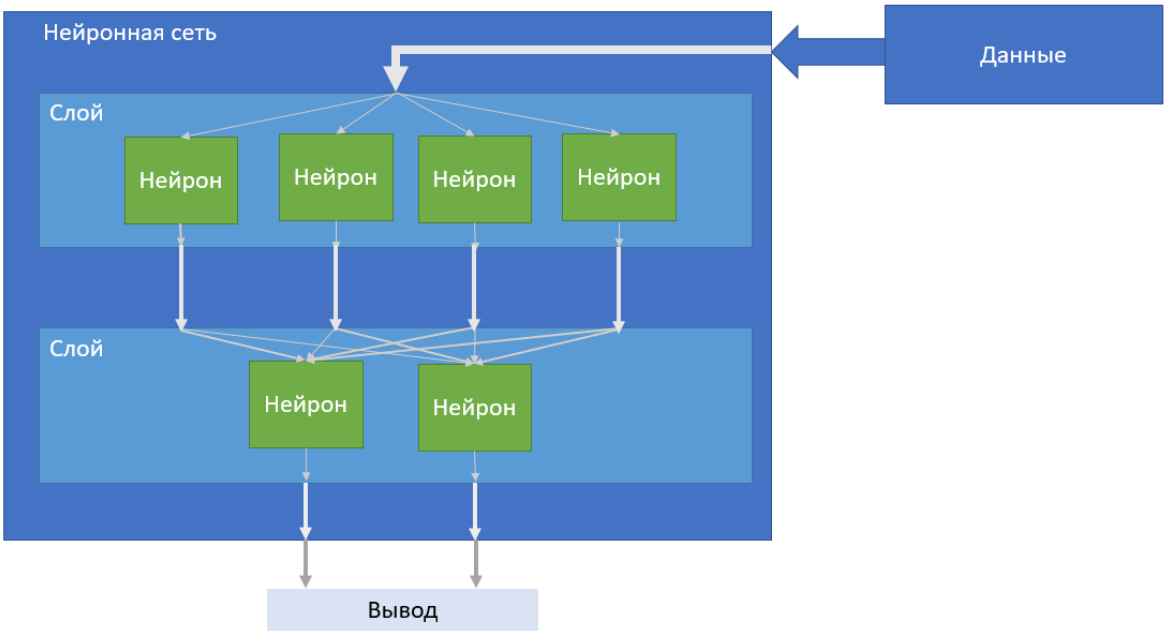


Рисунок 1 - пример структуры объектов в нейронной сети

Основным принципом взаимодействия компонентов нейронной сети является иерархичность и разделение обязанностей. Пользователь взаимодействует только с объектом, представляющим нейронную сеть. Его задачей является получение и обработка данных из внешнего мира и связь слоев между собой, а так же инициализация и изменение параметров модели.

Объект слоя работает как контейнер для нейронов, осуществляя их координацию, латеральное ингибирование, распределяя входные сигналы на входы всех нейронов и передавая их выходные значения в модель для дальнейшей обработки.

Объект нейрона хранит в себе информацию о весах синапсов, времени прихода спайков по синапсам и входном и выходном уровнях.

Таким образом обработка данных производится конвеерным методом, и каждый объект рассматривает вложенные как «черные ящики» (за небольшими исключениями, обусловленными условностями постоения программной архитектуры и обеспечения быстродействия), благодаря чему при необходимости сеть сможет работать с разнородными объектами: например, объединять нейроны модели МакКаллока и STDP, потенцирующие и ингибирующие, с сигмоидальной или дельта активационной функцией, или обычные слои и резервуарные.

MODEL.PY

Timer

В начале работы программы создаётся один объект класса Timer, который затем передаётся как аргумент в инициализации всех синхронизируемых объектов. Служит для имитации асинхронной работы программы.

clock

Хранит текущий отсчёт времени, полученный по AER

listeners

Список всех синхронизируемых объектов

__init__()

Инициализируем clock=0 и listeners=[]

set_time(time)

Устанавливает clock в значение time

reset()

Устанавливает clock=0 и вызывает метод reset всех синхронизируемых объектов

add_listener(listener)

Добавляет listener в список синхронизируемых объектов

remove_listener(listener)

Убирает listener из списка синхронизируемых объектов

announce()

Вызывает метод set_clock(clock) всех синхронизируемых объектов

SyncedObject

Класс, от которого наследуются все классы объектов, требующих синхронизации и имитации асинхронного поведения. Определяет методы и свойства для взаимодействия с таймером.

clock

Хранит текущий отсчёт времени, полученный от Timer

__init__({‘timer’: Timer})

В качестве аргумента при инициализации объекта должен быть передан объект класса Timer. Вызывается метод add_listener Timer с аргументом – ссылкой на текущий объект, таким образом каждый объект класса, унаследованного от SyncedObject, будет добавлен в listeners Timer и обновляться при подаче нового события по AER

update_clock(time)

Устанавливает в clock значение time. Служит для того, чтобы быть перегруженным дочерними классами, для логики работы которых необходимо совершать действия по тактовому импульсу.

reset()

Устанавливает clock = 0. Служит для того, чтобы быть перегруженным дочерними классами, для логики работы которых необходимо регулярно очищать память

Logger(SyncedObject)

Класс, служащий для сохранения состояний нейронов и правильных ответов учителя в ходе работы сети. Необходим для расчёта правильности работы сети в конце прогонки данных.

logs

Словарь, куда записываются все логи. Имеет структуру вида:

{‘timestamps’: [] – массив для записи отметок времени, если потребуется соотнести запись со значением времени внутри модели

[Object]: []… - массивы для записи состояний объектов в каждый момент времени

watches

Список всех объектов, за которыми происходит слежение, и атрибутов, значение которых необходимо сохранить.

__init__()

Инициализация свойств объекта начальными значениями.

add_watch(watch, attr = ‘output’,)

Записывает в массив watches пару (watch, attr, blocking). attr – строка с названием свойства объекта, значение которого необходимо сохранять.

remove_watch(watch)

Слежение за объектом watch прекращается.

collect_watches()

В списки событий всех объектов, за которыми происходит слежение, в словаре watches добавляется значение выбранного свойства

get_logs(watch)

Передаёт сохранённый список значений свойства объекта watch

update_clock(time)

Обновляет clock и вызывает collect_watches

SpikeNetwork(SyncedObject)

feed

Объект-эмулятор потока данных с камеры DataFeed (см. camera_feed.py/DataFeed)

frame

Номер входного события с камеры

layers

Список объектов-слоёв нейросети

logger

Объект-логгер (см. Logger)

teacher

Объект, используемый для хранения номера показываемой траектории. После каждого события в свойство output заносится массив вида: [1j, …, 1, 1j, …, 1j] где индекс единицы совпадает с номером показываемой в данный момент траектории, а все остальные позиции заняты мнимыми единицами. Когда хоть на одном нейроне выходного слоя есть спайк, массив заносится в логгер.

raw_data

Текстовое представление пришедших по AER данных

__init__()

инициализация всех свойств, также в цикле инициализируются слои и нейроны. Структура сети передаётся в виде ключ-значение ‘structure’: (n_1, n_2,…, n_m), где n_i – число нейронов в i-том слое. При создании сети все веса нейронов инициализируются нулями

set_param_set(param_set)

Смена параметров нейронов для уже запущенной сети. Все параметры иерархически передаются вниз в методы set_param_set слоёв из списка layers

set_feed(feed)

Смена потока данных

reset()

Сброс количества пришедших событий, происходит в начале каждого каждого набора траекторий после замены весов у нейронов.

next()

Подача на вход первого слоя нейронов (метод update объекта Layer) следующего события из потока данных. Затем выходные спайки первого слоя передаются на второй, и так далее. Выходные значения последнего слоя возвращаются из функции.

get_journals_from_layer(layer = -1)

Формирует массив массивов спайков с указанного слоя, по умолчанию – с выходного.

calculate_fitness()

При помощи функции get_journals_from_layer формируется матрица R, такая, что

$$R = \begin{bmatrix} r_{0,0} & \cdots & r_{0,m} \\ \vdots & \ddots & \vdots \\ r_{n,0} & \cdots & r_{n,m} \end{bmatrix}$$

В n строках матрицы записаны состояния n нейронов указанного слоя в момент одного из m входных событий. Таким образом, в j-том столбце имеется от 0 до n значений «0» и от 0 до n значений «1», соответствующих не сработавшим и сработавшим в момент прихода j-того события нейронам.

Затем получаются записи состояний объекта-учителя teacher из logger и преобразовываются в матрицу

$$C = \begin{bmatrix} c_{0,0} & \cdots & c_{0,n} \\ \vdots & \ddots & \vdots \\ c_{m,0} & \cdots & c_{m,n} \end{bmatrix}$$

Элемент $c_{i,j}$ принимает значение 1, если в момент времени i показывалась траектория с номером j, или 1j (мнимая единица), в противном случае.

Затем производится матричное умножение R и C, результат заносится в матрицу n*n F, такую, что $Re(F_{i,j})$ – равно числу раз, когда нейрон i срабатывал при показе траектории j, а $Im(F_{i,j})$ – равно числу раз, когда нейрон i срабатывал на прочие траектории.

Затем считается значение фитнесс-функции алгоритму, аналогичному следующему:

1. Выбрать в первой строке матрицы F максимальное по действительной части значение $F_{i,j}$
2. Добавить к значению score $F_{i,j}$
3. Передать на начало алгоритма матрицу F’, равную матрицу F с удалёнными строкой i и столбцом j

Затем значение score преобразуется по следующей формуле и возвращается функцией:

$$score = \left(\frac{Re(score)}{Re(score) + Im(score) + 1} - \frac{1}{n}\right) 1/* Re(score)$$

Таким образом, значение score учитывает как общее количество верных спайков, выданных всеми нейронами, так и отношение числа правильных срабатываний к неправильным. Слагаемое +1 в знаменателе дроби предохраняет от деления на 0, в то время как слагаемое $-\frac{1}{n}$ служит, чтобы отобразить превышение количества правильных срабатываний над тем, что было бы при случайном выборе.

Перед возвратом значения score нейроны выходного слоя переупорядочиваются таким образом, чтобы индекс каждого нейрона в слое соответствовал номеру наиболее хорошо угадываемой им траектории.

get_weights()

Возвращает массив возвращаемых методами get_weights() слоёв значений

set_random_weights()

Вызывает метод set_random_weights() всех слоев

set_weights(weights)

Weights – список списков весов нейронов.

Передаёт i-тому слою из списка слоёв i-тый элемент списка weights.

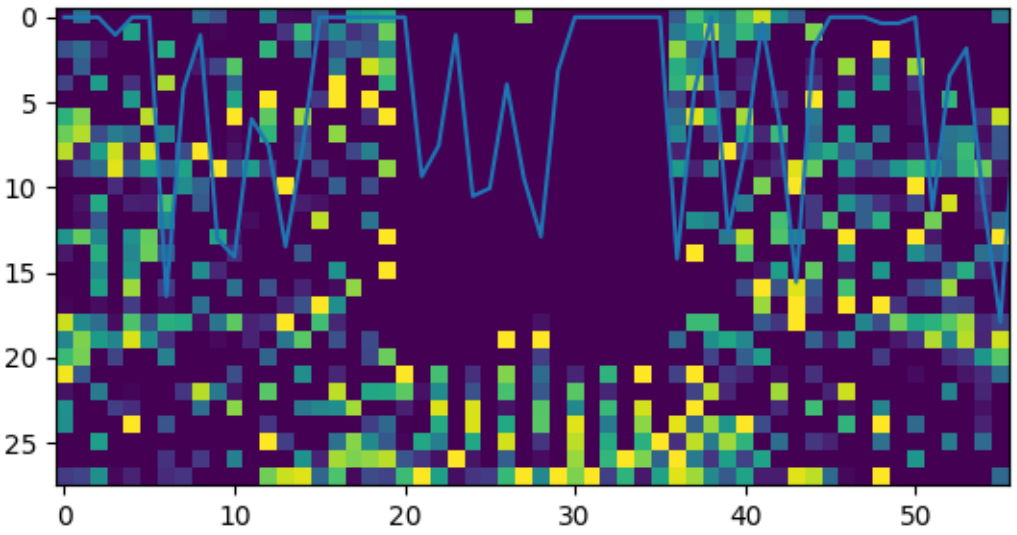
mutate(*weights)

Weights – несколько список списков весов нейронов.

Передаёт i-тому слою из списка слоёв i-тый элемент каждого элемента weights.

save_attention_maps()

Сохраняет карты внимания нейронов выходного слоя в виде изображений. На карте внимания каждый пиксель примерно соответствует положению пикселя входной матрицы, и цветом отображается вес этого пикселя. Так как сигнал с каждого пикселя может передаваться по синапсам разной полярности, в зависимости от того, включился он или выключился, каждому пикселю входной матрицы соответствуют два соседних по горизонтали пикселя на карте внимания (ломаная линия – артефакт).


0
5
10
15
20
25
0
10
20
30
40
50

Layer(SyncedObject)

Базовый класс, от которого наследуются классы слоев.

layer_number

Номер слоя в модели, используется для нумерации синапсов

neuron_count

Количество нейронов в слое

neurons

Список объектов нейронов

output

Список выходных значений нейронов

__init__()

Инициализирует необходимые свойства класса

update(input_spikes)

Передаёт список адресов синапсов, по которым пришли спайки в данный момент времени, в каждый нейрон из neurons, и записывает в output значения output нейронов

get_synapses()

Возвращает список синапсов слоя. Синапс представляется строкой вида «номер слоя в 16-ричном представлении с незначащими нулями до ширины строки в 2 знака», «номер синапса в 16-ричном представлении слое с незначащими нулями до ширины строки в 2 знака».

get_weights()

Возвращает список весов нейронов слоя.

set_random_weights()

Вызывает метод set_random_weights всех нейронов слоя

set_weights(weights)

weights – список весов нейронов

Для каждого нейрона из списка neurons передаёт i-тому нейрону i-тый элемент списка weights

mutate(*weights)

weights – несколько списков весов нейронов

Для каждого нейрона из списка neurons передаёт i-тому нейрону i-тый элемент каждого элемента weights

set_param_set(param_set)

Передает новые параметры нейронов во все нейроны слоя

rearrange_neurons(new_order)

Принимает на вход список new_order. i-тый элемент списка – номер j позиции, куда нужно переместить i-тый нейрон. Используется для того, чтобы переставить нейроны в выходном слое на положения с номерами, равными номерам траекторий, лучше всего ими угадываемых.

STDPLayer(Layer)

Класс слоя нейросети с временно-зависимой пластичностью синапсов

wta

Булева переменная, обозначает, используется ли Winner-Takes-It-All модель

__init__()

Заполняет список neurons объектами STDPNeuron в количестве neuron_count и устанавливает режим wta (вкл или выкл)

update()

После того, как срабатывает update родительского класса, происходит латеральное ингибирование нейронов. Ингибируются все нейроны, кроме тех, которые сработали на этом такте и находятся в периоде рефрактерности.

Neuron(SyncedObject)

Базовый класс, описывающий нейрон. Конкретные модели наследуются от него. Нейрон не разделяется на пре- и пост- нейрон.

input_level

Уровень сигнала на входе нейрона

output_level

Уровень сигнала на выходе нейрона

learn

Булева переменная: обучается ли нейрон

param_set

Набор параметров, определяющий работу нейрона

randmut

Шанс на замену значения веса синапса случайным значением при работе метода mutate

weights

Словарь весов синапсов нейрона. Имеет структуру вида

{$synapse_1$: int, $synapse_2$: int…}, где $synapse_i$ – строка, соответствующая некому синапсу.

update()

Не описывается в родительском классе, т.к. поведение нейронов в разных моделях совершенно разное, но присутствует для обозначения структуры.

set_random_weights()

Значения в словаре weights заменяются на случайные между минимальным и максимальным значениями веса

set_param_set(param_set)

Заменяет параметры нейрона на новый набор параметров

mutate(*weights)

Weights: n словарей в формате весов синапсов (см. weights). Для каждого нейрона с шансом randmut будет установлен случайный вес между минимальным и максимальным, а если нет – то один из соответствующих весов, поданных на вход. Так как нейроны выходного слоя выстраиваются в одном порядке во всех случаях (см.

SpikeNetwork.calculate_fitness), перемешиваются веса нейронов, выполняющих одну и ту же функцию.

STDPNeuron(SyncedObject)

refractory

Булева переменная: находится ли нейрон в периоде рефрактерности

ltp_synapses

Словарь, содержащий пары синапс: время последнего срабатывания. Используется для определения тех синапсов, которые сработали вовремя для того, чтобы их вес увеличился

t_spike

Момент времени прихода последнего спайка

t_last_spike

Момент времени прихода предпоследнего спайка

inhibited_by

Используется для латерального ингибирования и рефрактерности, указывает время, до которого нейрон неактивен.

inhibited_on

Момент времени, когда нейрон был ингибирован

__init__()

Генерируется функция активации нужного вида. Ссылка на функцию помещается в свойство activation_function. В данный момент это дельта-функция с пороговым значением param_set[‘i_thres’]. Свойства устанавливаются в значение по умолчанию:

t_spike = 0,

t_last_spike = 0,

inhibited_by = -1,

inhibited_on = -1.

reset()

Восстанавливаются значения по умолчанию свойств, ltp_synapses сбрасывается.

update(synapses)

Если установленное в свойстве clock время меньше, чем значение inhibited_by, то есть нейрон находится в периоде рефрактерности или ингибирования, обработка прекращается

Значение выходного уровня сбрасывается в 0, а флаг рефрактерности refractory сбрасывается в False.

В аргументе synapses передаётся список сработавших синапсов нейрона. Каждый из них обрабатывается в цикле по алгоритму:

1. t_last_spike = t_spike, t_spike = clock. Если пришло несколько импульсов – для второго разница во времени будет уже нулевой.
2. input_level обновляется по формуле
$$input\_level = input\_level * e^{\frac{t_{spike} - t_{last\_spike}}{param\_set['t\_leak']}} + weights[synapse]$$
3. Для синапса synapse заносится новое значение в словарь ltp_synapses, равное clock+param_set['t_ltp']

Затем происходит обработка срабатывания нейрона.

1. Для output_level расчитывается новое значение по activation_function(input_level)
2. Если нейрон сработал, входной уровень сбрасывается в 0, флаг периода рефрактерности устанавливается в True, для inhibited_by расчитывается новое значение как сумма clock и param_set['t_refrac']
3. Если включено обучение, происходит прогон по всем записям в словаре ltp_synapses. Для синапсов, период долгосрочного потенцирования которых ещё не истёк, вызывается функция _synapse_inc_, для остальных _synapse_dec_.
4. Все значения в ltp_synapses сбрасываются

В конце функция возвращает выходной уровень нейрона.

_synapse_inc_(synapse)

Вес переданного как аргумент нейрона в словаре weights увеличивается на param_set['a_inc']. Вес не может стать меньше param_set['w_min']

_synapse_dec_(synapse)

Вес переданного как аргумент нейрона в словаре weights уменьшается на param_set['a_dec']. Вес не может стать больше param_set['w_max']

inhibit()

Если нейрон уже ингибирован, но при этом не срабатывал, его период ингибирования запускается заново, inhibited_by = param_set['t_inhibit'] + clock. Значение inhibited_on устанавливается равным clock

Если нейрон находится на периоде рефрактерности, но ещё не был латерально ингибирован, его период ингибирования продляется на период ингибирования param_set['t_inhibit']

CAMERA_FEED.PY

Файл содержит описание класса DataFeed, имитирующего поток данных с камеры.

DataFeed

cache

Кэш траекторий. Если траектория уже открывалась, она не загружается заново с диска, а подгружается из кэша

data

Содержимое файла с траекториями

index

Указывает, какой кусок data выдавать следующим

pixels

Служебный массив адресов пикселей камеры с матрицей 28*28. Они служат идентификаторами синапсов для нейронов входного слоя.

timer

Указатель на таймер.

__init__()

Все свойства устанавливаются в значения по умолчанию

load(source)

Загружает содержимое указанного файла траекторий либо из кэша, либо с диска, в data, устанавливает index в 0, вызывает timer.reset()

get_pixels

Возвращает массив адресов пикселей

__iter__, __next__

Служебные методы, обеспечивающие использование объекта в цикле for. При каждой итерации из data считываются и выдаются очередные 40 бит данных в виде строки с шестнадцатеричным числом

parse_aer(raw_data)

Возвращает строку с шестнадцатеричным числом, составленным из 17 старших бит raw_data, обозначающую адрес синапса, по которому пришёл сигнал от камеры, и число из 23 младших бит raw_data, равное времени события в микросекундах.

UTILITY.PY

Файл с константами и параметрами, для удобства собранными в одном месте.

АЛГОРИТМ РАБОТЫ СЕТИ

На блок-схеме приведён примерный алгоритм программы, проводящей многостадийное обучение нейросети на неком наборе входных данных. На практике к алгоритму могут добавляться дополнительные шаги, например, для прогонки значений входных параметров сети для нахождения оптимальных, или для визуализации результата обучения сети на тестовом наборе входных данных, как это сделано в программах sweep.py и test.py соответственно.