

Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Алтайский государственный технический университет им. И.И. Ползунова»

Факультет (институт) Информационных технологий
Кафедра Прикладная математика
Направление Программная инженерия
Направленность (профиль) Разработка программно-информационных систем

УДК 004.942

УТВЕРЖДАЮ
Заведующий кафедрой

Е. Г. Боровцов
(инициалы, фамилия)
" " 2024 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

МД 09.04.04.08.000 ПЗ

обозначение документа

Разработка многоуровневой распределенной системы анализа
больших данных биоподобными сенсорно-моторными алгоритмами с
целью определения аномалий
тема магистерской диссертации

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Студент группы 8ПИ-21 Потапов Даниил Петрович
№ группы фамилия, имя, отчество

Научный руководитель
доцент каф. ПМ, к. ф-м. н. С. М. Старолетов
должность, учёная степень инициалы, фамилия

Барнаул 2024

Реферат

Целью данной работы является:

- изучение архитектуры кортикальных слоев и схемы движения сигналов между кортикальными слоями;
- написание программной системы для моделирования процессов, протекающих в неокортексе.

Объем работы – 152 страницы, включающих 48 рисунков, 20 используемых источников.

Ключевые слова: нейронные сети, НТМ, иерархическая темпоральная память, биоподобные алгоритмы, сенсорно-моторные алгоритмы, распределенные системы.

The abstract

By the purpose of the work it was:

- study of the architecture of cortical layers and the pattern of signal movement between cortical layers;
- development software system for modeling processes occurring in the neocortex.

Size of work – 152 pages includes 48 figures, 20 used sources.

Keywords: neural networks, НТМ, hierarchical temporal memory, biosimilar algorithms, sensory-motor algorithms, distributed systems.

					<i>МД 09.04.04.08.000 ПЗ</i>			
<i>Изм</i>	<i>Лист</i>	<i>№ докум</i>	<i>Подп.</i>	<i>Дата</i>	<i>Разработка многоуровневой распределенной системы анализа больших данных биоподобными сенсорно-моторными алгоритмами с целью определения аномалий</i>	<i>Лист</i>	<i>Лист</i>	<i>Листов</i>
<i>Разр.</i>		<i>Потапов Д. П.</i>	<i>Потапов</i>	<i>20.06.24</i>		<i>У</i>	<i>2</i>	<i>152</i>
<i>Руководитель</i>		<i>Старолетов С.М.</i>	<i>Старолетов</i>	<i>20.06.24</i>		<i>АлтГТУ ФИТ 8ПИ - 21</i>		
<i>Н. контр</i>		<i>Потупчик А. И.</i>	<i>Потупчик</i>	<i>20.06.24</i>				
<i>Зав. каф.</i>		<i>Боровцов Е. Г.</i>	<i>Боровцов</i>	<i>20.06.24</i>				

Содержание

Введение	4
1 Описание предметной области	6
1.1 Строение нервной системы.....	6
1.2 Передача сигналов в ЦНС.....	14
1.3 Теория «тысячи мозгов»	22
1.3.1 Обзор теории.....	22
1.3.2 Серво-моторная обработка информации	26
1.3.3 Обучение	29
1.3.4 Отличия от искусственных нейронных сетей	30
1.4 Большие данные и аномалии	34
2 Математическая модель.....	37
2.1 Модель НТМ нейрона и одного кортикального слоя.....	37
2.2 Многоуровневая модель кортикальной колонки.....	39
2.3 Модель перехода состояний нейронов	42
2.4 Обучение.....	44
2.5 Объем хранения данных	45
2.6 Выявление аномалий	47
3 Реализация.....	49
3.1 Ядро системы	49
3.1.1 Структура хранения данных	49
3.1.2 Переход между состояниями	50
3.1.3 Обучение	53
3.2 Модуль визуализации.....	56
3.2.1 Связь с ядром системы.....	56
3.2.2 Интерфейс	60
Заключение	64
Список использованных источников	65
Приложение А. Задание на выполнение ВКР	Ошибка! Закладка не определена.
Приложение Б. Исходный код системы.....	70

Введение

Исследование колончатой организации неокортекса по теории Джеффа Хоккинса является актуальным направлением для понимания работы человеческого мозга.

Хоккинс предложил модель, основанную на идее, что неокортекс состоит из колонок - вертикальных структур, которые работают параллельно и координируют свою активность для обработки информации. По его мнению, именно колонки - основные строительные блоки неокортекса, влияющие на нашу способность перерабатывать сложную информацию и формировать понимание мира [1].

Исследования колончатой организации неокортекса в рамках данной теории помогают углубить наши знания о различных аспектах работы мозга, таких как восприятие, память, решение проблем и формирование представлений. Понимание, как точно работают колонки, может помочь нам в разработке новых методов лечения нейрологических заболеваний, развитии искусственного интеллекта и создании более эффективных компьютерных алгоритмов.

Кроме того, исследования колончатой организации неокортекса могут пролить свет на механизмы возникновения нейрологических расстройств, таких как эпилепсия, шизофрения или болезнь Паркинсона. Понимание отклонений в структуре и функционировании колонок может помочь в разработке новых методов диагностики и лечения таких заболеваний.

Исследования по теории Хоккинса также могут внести вклад в развитие мозговых компьютерных интерфейсов, позволяющих людям с ограниченными физическими возможностями контролировать устройства прямо с помощью своих мыслей.

В целом, исследование колончатой организации неокортекса по Хоккинсу имеет большую актуальность для нас не только с точки зрения фундаментальных научных знаний о мозге, но и с практической перспективы для развития новых технологий и методов лечения.

Когда речь идет о поиске аномалий в данных, анализ данных играет решающую роль. Аномальные данные могут быть необычными или неожиданными, и их обнаружение является важным заданием во многих областях, таких как финансы, кибербезопасность, медицина и т.д.

Подход, основанный на идеях колонок, также может быть полезным при анализе данных и поиске аномалий. Поскольку колонки неокортекса занимаются обнаружением и предсказанием паттернов, они могут помочь в поиске неправильных или необычных образцов в данных.

1 Описание предметной области

1.1 Строение нервной системы

Задача центральной нервной системы (ЦНС) — после получения информации произвести за доли секунды ее оценку и принять соответствующее решение за счет способности головного мозга хранить и воспроизводить в нужный момент ранее поступившую информацию. Мыслительная способность осуществляется в результате анализа и синтеза нервных импульсов в высших центрах головного мозга и составляет высшую нервную деятельность человеческого организма [2]. Разделы ЦНС указаны на Рисунок 1.1.

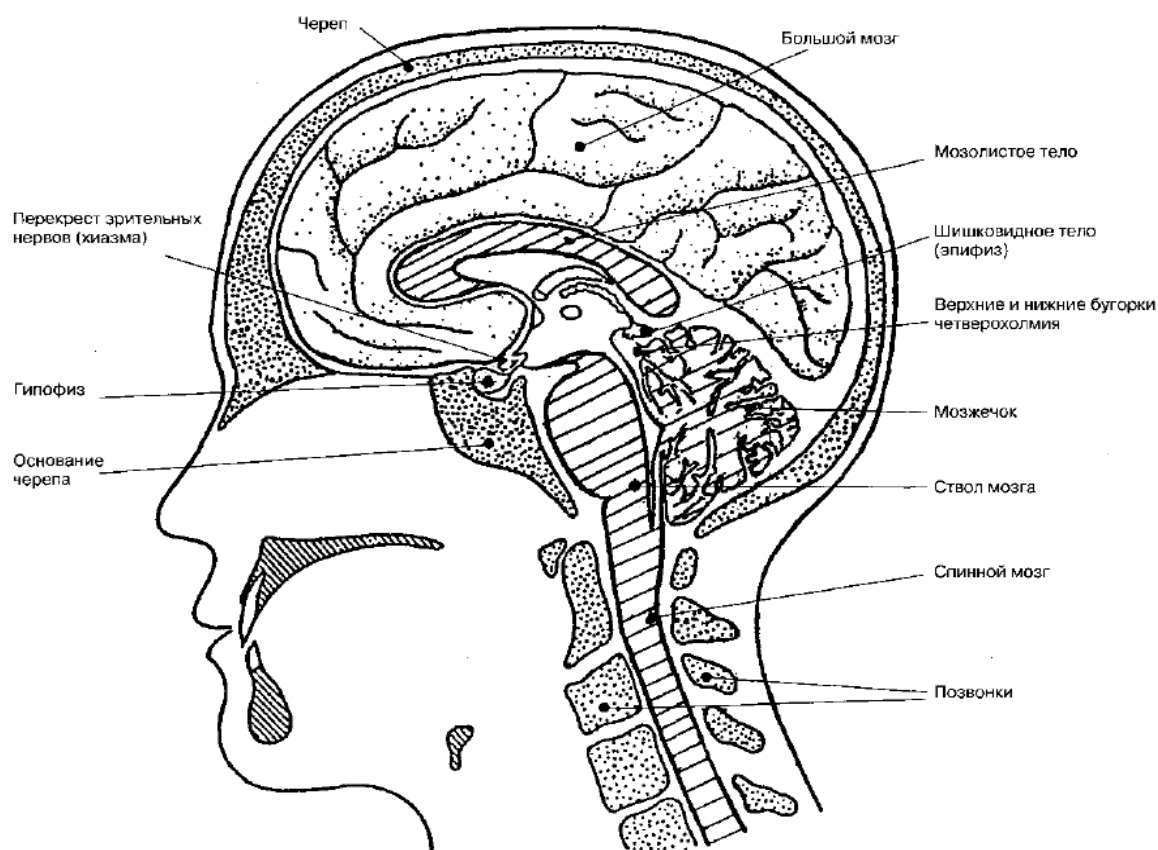


Рисунок 1.1 – Разрез головы по Шаде [3]

На Рисунок 1.2 выделены основные структуры головного мозга.

На клеточном уровне кора мозга представляет собой грандиозную структуру — согласно недавним подсчетам, она содержит от 10 до 100 млрд нейронов. Снаружи мозг выглядит как сложное складчатое образование с множеством выступов и углублений, занимающее верхнюю половину полости черепа. Если бы

удалось аккуратно развернуть обширную мантию, которую представляет собой кора мозга, получился бы покров площадью около одного квадратного метра, состоящий из шести слоев. В каждом слое находится бесчисленное множество нейронов с ветвящимися отростками; нейроны окружены вспомогательными клетками. Этот слоистый покров мозга подразделяется на миллионы вертикальных колонок. Таким образом, мы имеем дело с колоссальной шестислойной трехмерной системой. Каждый слой коры мозга выполняет специализированную функцию — осуществляет вход, выход или внутренние коммуникации в пределах коры [4].

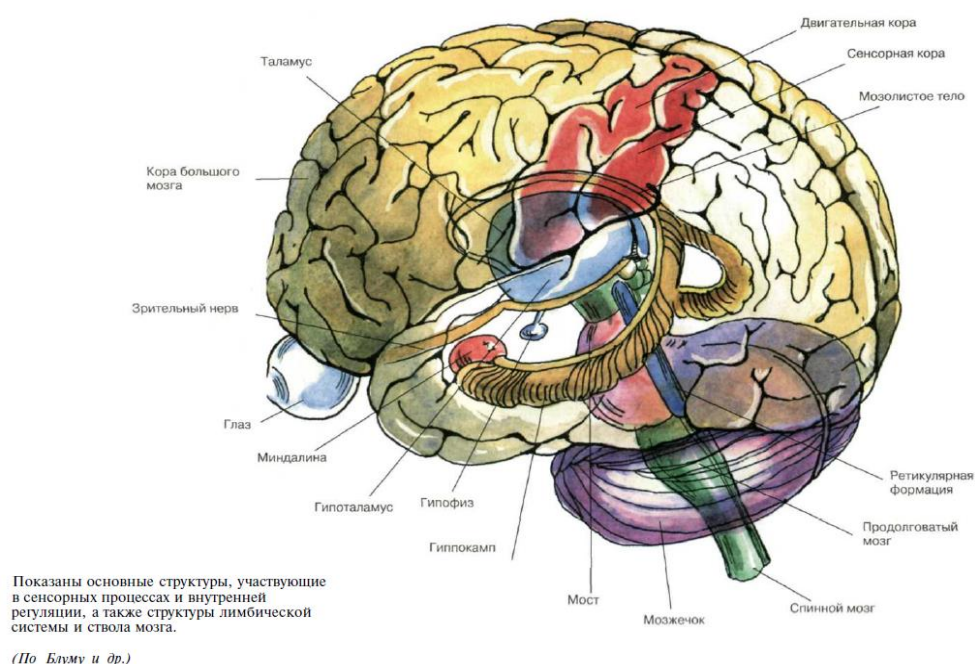


Рисунок 1.2 – Основные структуры головного мозга по Блуму [3]

Вероятнее всего, для описания вертикально расположенных нейронов слово «колонка» первым употребил Экономо. А о вертикальной модели работы коры и ее функциональной организации первый предположил Лоренте де Но. Маунткэсл обнаружил колончатую организацию во всех топографических и цитоархитектонических отделах соматосенсорной коры (Организация соматосенсорной коры показана на Рисунок 1.3). Так же он выделил несколько свойств колончатой организации в соматосенсорной коре:

1. Каждая локальная колонка характеризуется своими статическими параметрами места и модальности, а ряды колонок с одной и той же дерматомной

специализацией и ряды с одной и той же модальной специализацией расположены под прямым углом друг к другу.

2. Группы колонок этой матрицы специализированно связаны с упорядоченными группами колонок в других областях.

3. Данная область коры управляется своим специфическим таламокортикальным входом в такой мере, что те участки, в которых обработка сенсорных сигналов происходит, как полагают, с наибольшей точностью, - участки для кисти, ступни и лица – изолированы от некоторых кортико-кортикальных и каллозальных входов.

4. Колончатая организация вполне совместима с тем, что проекции формы тела частично сдвинуты и несколько перекрываются (Проекции формы тела показаны на Рисунок 1.4) [5].

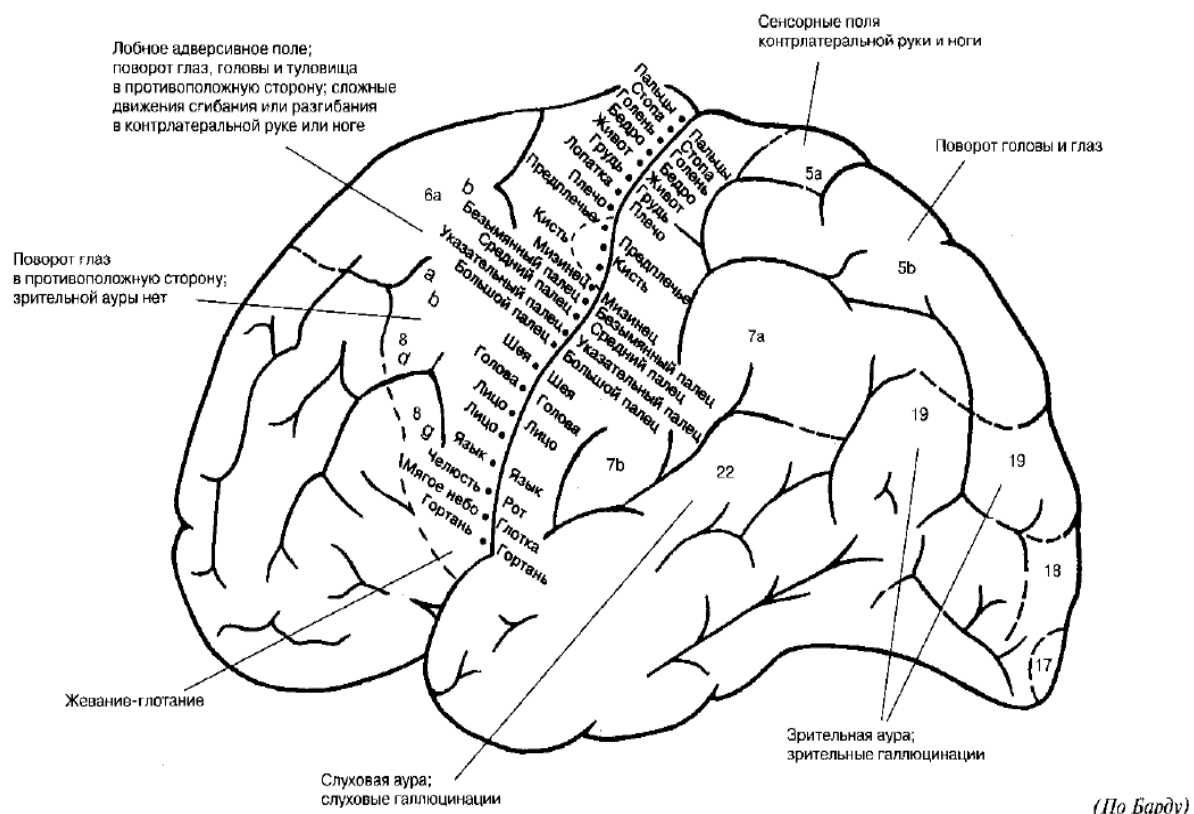


Рисунок 1.3 – Соматическая организация моторной и сенсорной областей коры человека по Барду [3]

Диаметр небольших колонок нейронов в коре мозга составляет примерно 1 мм [4]. В одной колонке находится около 110 клеток. Причем эта цифра остается

практически неизменной в разных областях неокортекса и у разных видов млекопитающих [5].

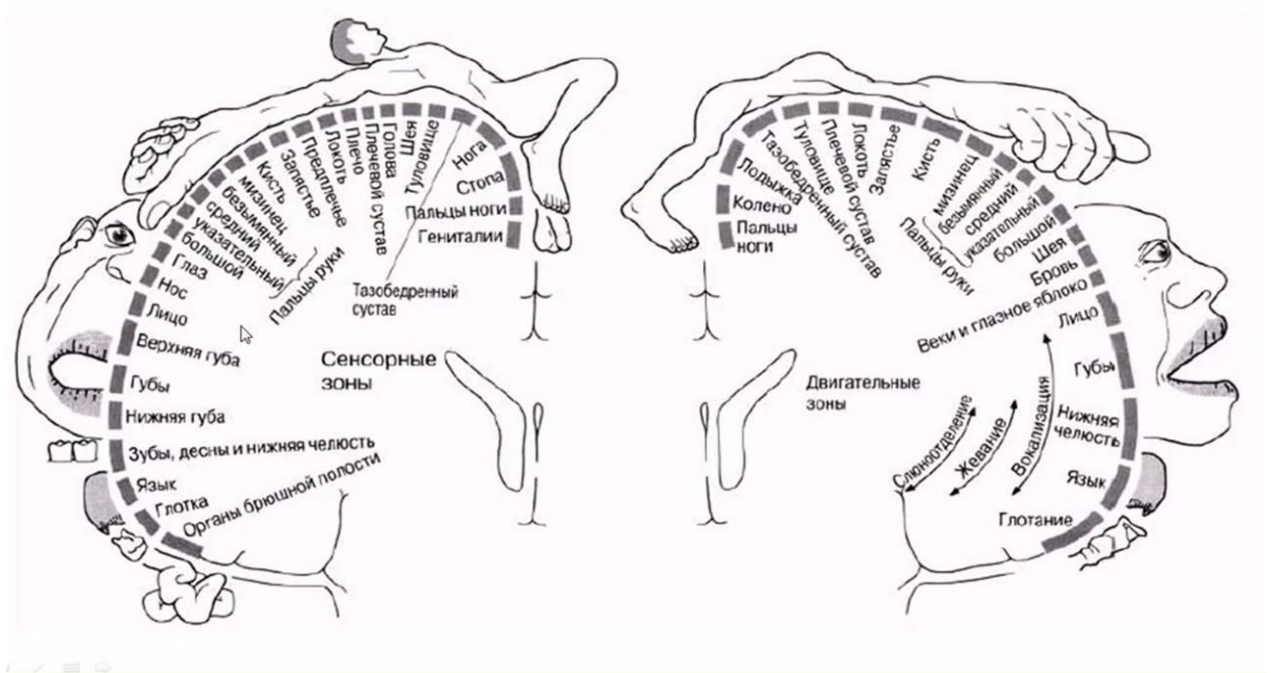


Рисунок 1.4 – Двигательный и сенсорный гомункулус Пенфилда [3]

Шесть горизонтальных слоев коры головного мозга образуют кортикальные колонки, вертикальные цилиндрические срезы, представленные на Рисунок 1.5.

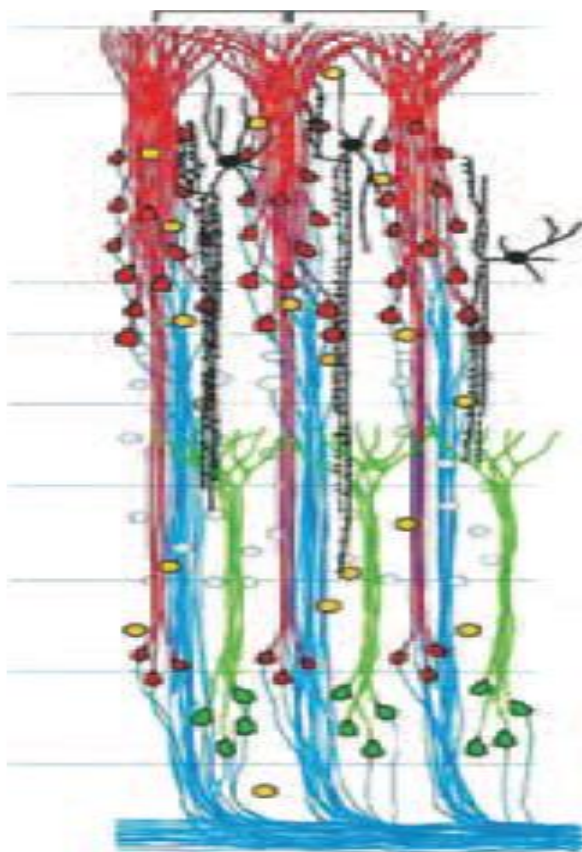


Рисунок 1.5 – Шесть основных слоев коры головного мозга, поперечный разрез [4]

Колонки можно объединить в гиперколонки, которые могут быть частью еще больших скоплений. Таким образом, кора головного мозга имеет как горизонтальную структуру, образованную шестью слоями клеток, так и вертикальную — колонки, гиперколонки и в конечном счете целые специализированные области.

Нужно заметить, что эти шесть слоев нумеруются (римскими цифрами), начиная сверху, где находится слой I, и дальше вниз до слоя VI, как на Рисунок 1.6. Названия слоев приведены на Рисунок 1.7.

I слой коры головного мозга состоит в основном из дендритов (входящие волокна), которые так тесно уложены и соединены между собой, что этот слой иногда называют «фиброзная сеть», слой переплетенных дендритов. Нейроны на Рисунок 1.6 названы «пирамидными», потому что их тела выглядят как микроскопические пирамиды [4].

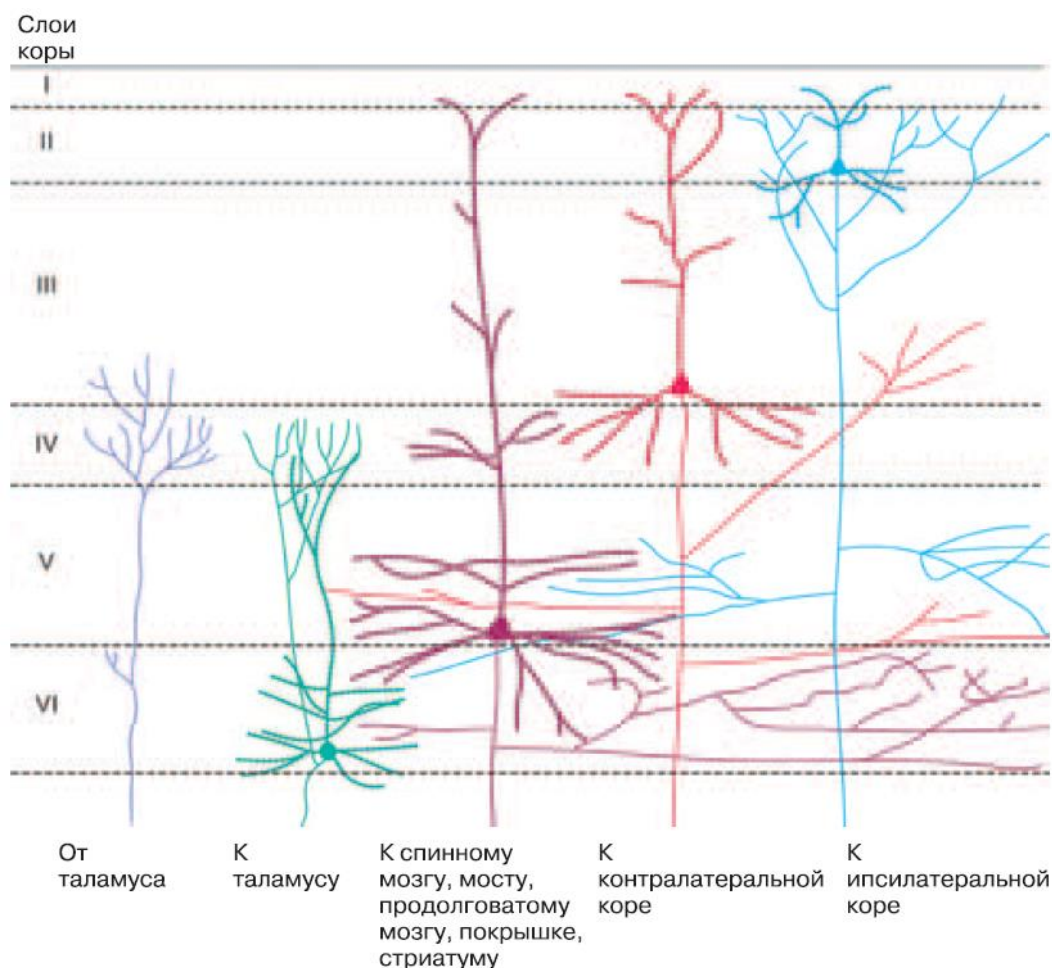


Рисунок 1.6 – Схематический рисунок шести слоев серого вещества коры головного мозга [4]

Согласно «нейронной доктрине», сформулированной С. Рамон-и-Кахалем, нервные клетки — нейроны — являются основными структурными и функциональными единицами нервной системы. Эта доктрина базируется на следующих основных положениях:

- Каждый нейрон является анатомической единицей. Это означает, что нейрон представляет собой клетку, в которой, как и в других клетках, имеется ядро и цитоплазма. Снаружи нервная клетка окружена оболочкой — плазматической мембраной, или плазмалеммой. В цитоплазме нейрона содержатся органеллы общего значения: эндоплазматический ретикулум, рибосомы, митохондрии и т. п., а также специальные органеллы: нейрофибриллы, построенные из белковых молекул длинные тонкие опорные нити, и тигроидное вещество, или вещество Ниссля, представляющее собой участки цитоплазмы с большим содержанием рибосом.

- Каждый нейрон является генетической единицей. Развиваясь из эмбриональной нервной клетки — нейробласта, — расположенной в нервной трубке или в ганглионарной пластинке, каждый нейрон содержит генетически запрограммированный код, определяющий специфику его строения, метаболизма и связей с соседними нейронами. Примеры представлена на Рисунок 1.8. Основные связи нейронов генетически запрограммированы. Однако это не исключает возможности модификации нейронных связей в процессе индивидуального развития при обучении и формировании различных навыков.

- Каждый нейрон является функциональной единицей. Иными словами, каждый нейрон представляет собой ту элементарную структуру, которая способна воспринимать раздражение и возбуждаться, а также передавать возбуждение в форме нервного импульса соседним нейронам или иннервируемым органам и мышцам.

- Каждый нейрон представляет собой поляризационную единицу, т. е. он проводит нервный импульс только в одном направлении. В силу этого отростки нейрона подразделяются на дендриты, которые проводят возбуждение к телу нейрона, и аксон, или нейрит, проводящий возбуждение от тела клетки.

- Каждый нейрон есть рефлекторная единица. Нейрон является элементарной составной частью той или иной рефлекторной дуги, по которой осуществляется проведение импульсов в нервной системе от рецепторов, воспринимающих средовые воздействия, до эффекторных органов, участвующих в ответной реакции на эти воздействия.

- Каждый нейрон является патологической единицей. Любая часть нервной клетки и ее отростков, отделенная путем повреждения от ее тела, погибает и подвергается распаду, или дегенерации. Хотя различные нейроны по-разному реагируют на повреждение, тем не менее при достаточно обширном повреждении цитоплазмы или ядра любого нейрона он погибает [6].

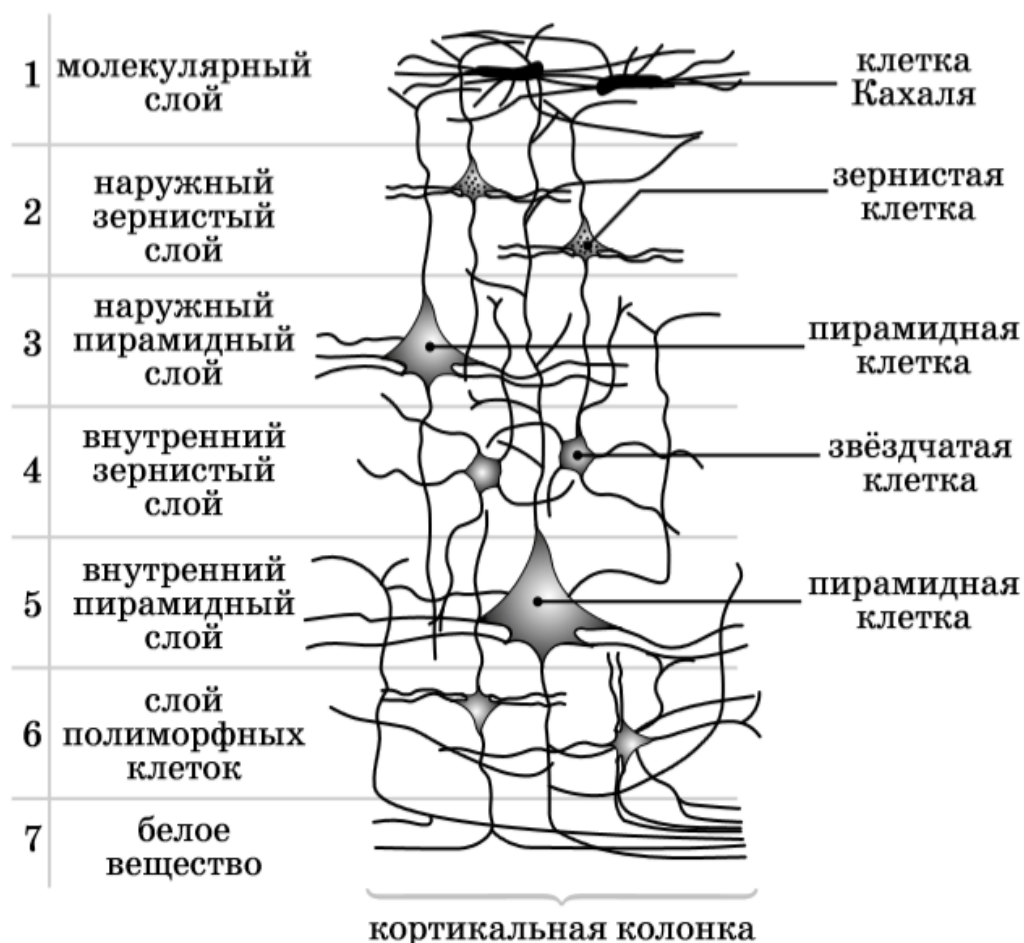


Рисунок 1.7 – Наименование слоев с указанием типов клеток

Погибшие нейроны не возмещаются. В случае их гибели после рождения число нейронов не может быть восполнено. Тем не менее при повреждении аксона его восстановление возможно путем роста отростка и воссоздания утраченных им в результате повреждения связей. Это наблюдается в периферической нервной системе при повреждении нервов.

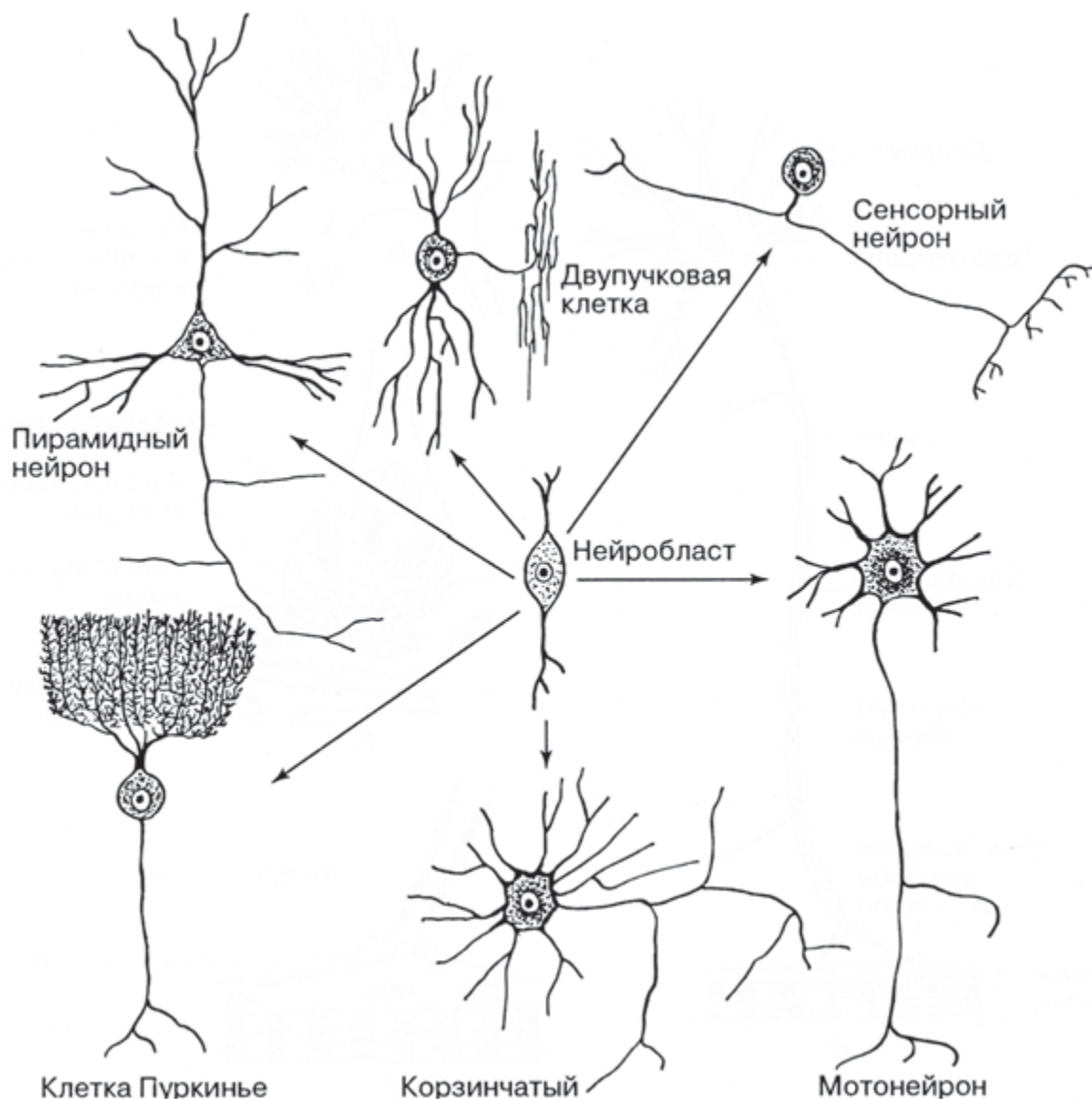


Рисунок 1.8 – Формирование разных типов нейронов из нейробластов [6]

1.2 Передача сигналов в ЦНС

Функционирование нервной системы связано с восприятием и обработкой разнообразной сенсорной информации, а также информационным обменом между различными частями организма и внешней средой. Передача информации между нервными клетками осуществляется в форме нервных импульсов. Нервные импульсы возникают в сенсорных нейронах как результат активации их воспринимающих структур, называемых рецепторами. Сами рецепторы активируются различными изменениями во внутренней среде организма и в окружающей его внешней среде. Сенсорные нейроны передают возникшие в рецепторах импульсы в спинной и головной мозг. Здесь происходит активация

других нейронов и передача нервных импульсов в конечном итоге на мотонейроны, локализованные в определенных отделах спинного и головного мозга (Рисунок 1.9).

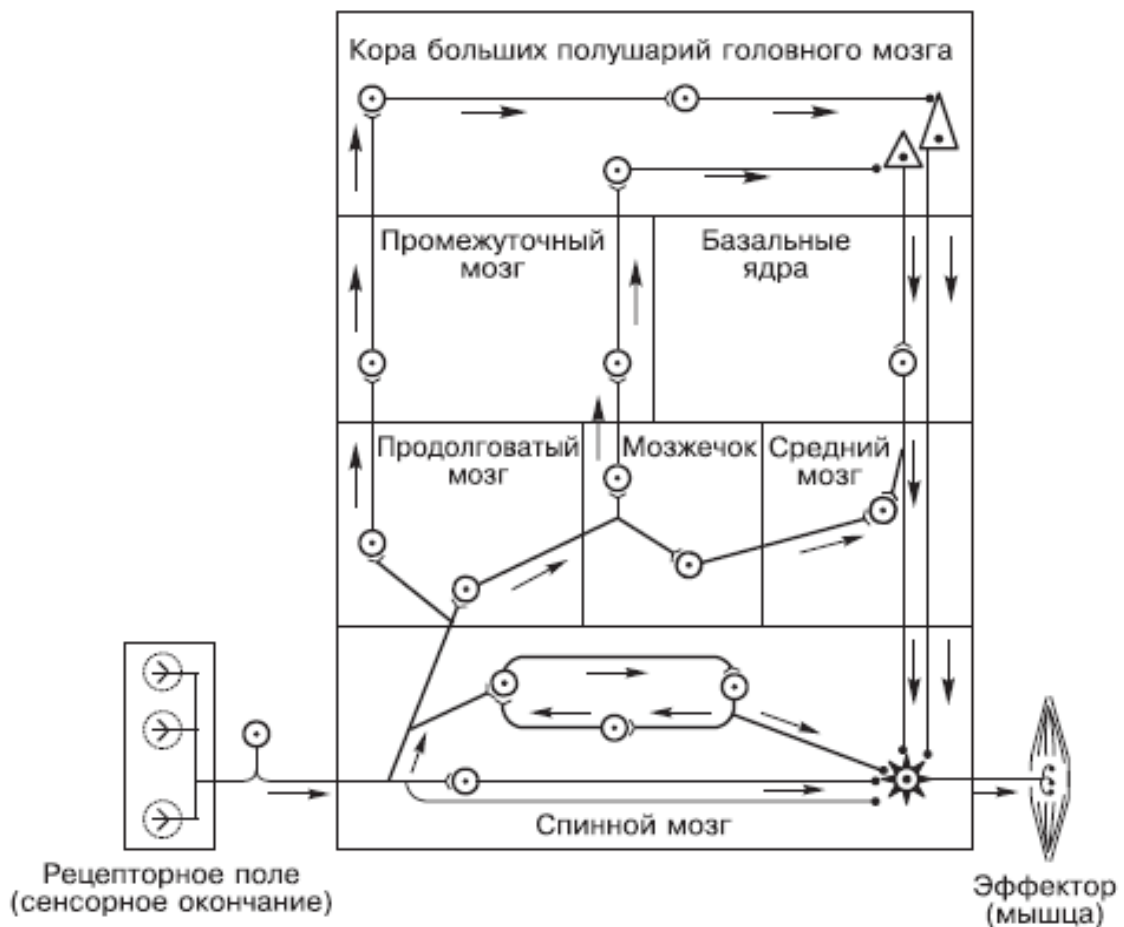


Рисунок 1.9 – Иерархия организации нейронных связей в нервной системе

Мотонейроны вступают в контакт с различными эффекторными (исполнительными) образованиями, такими как мышцы, железы, кровеносные сосуды, которые под влиянием поступающих нервных импульсов изменяют свою работу, повышая или снижая ее уровень. Посредством связей, обеспечивающих передачу нервных импульсов между нервными клетками, осуществляется избирательное объединение (интеграция) рецепторного аппарата и эффекторного аппарата, реализующего ответную реакцию организма. Нервная система обладает также памятью — способностью хранить и накапливать значимую для организма информацию, получаемую из внешней и внутренней среды [6].

Отростки нервной клетки неравнозначны в функциональном отношении, так как одни из них проводят раздражение к телу нейрона — это дендриты, и только

один отросток — нейрит (аксон) — проводит раздражение от тела нервной клетки и передает его либо на другие нейроны, либо на эффекторные структуры (в частности, на мышечные волокна). Функциональные части нейрона показаны на Рисунок 1.10.

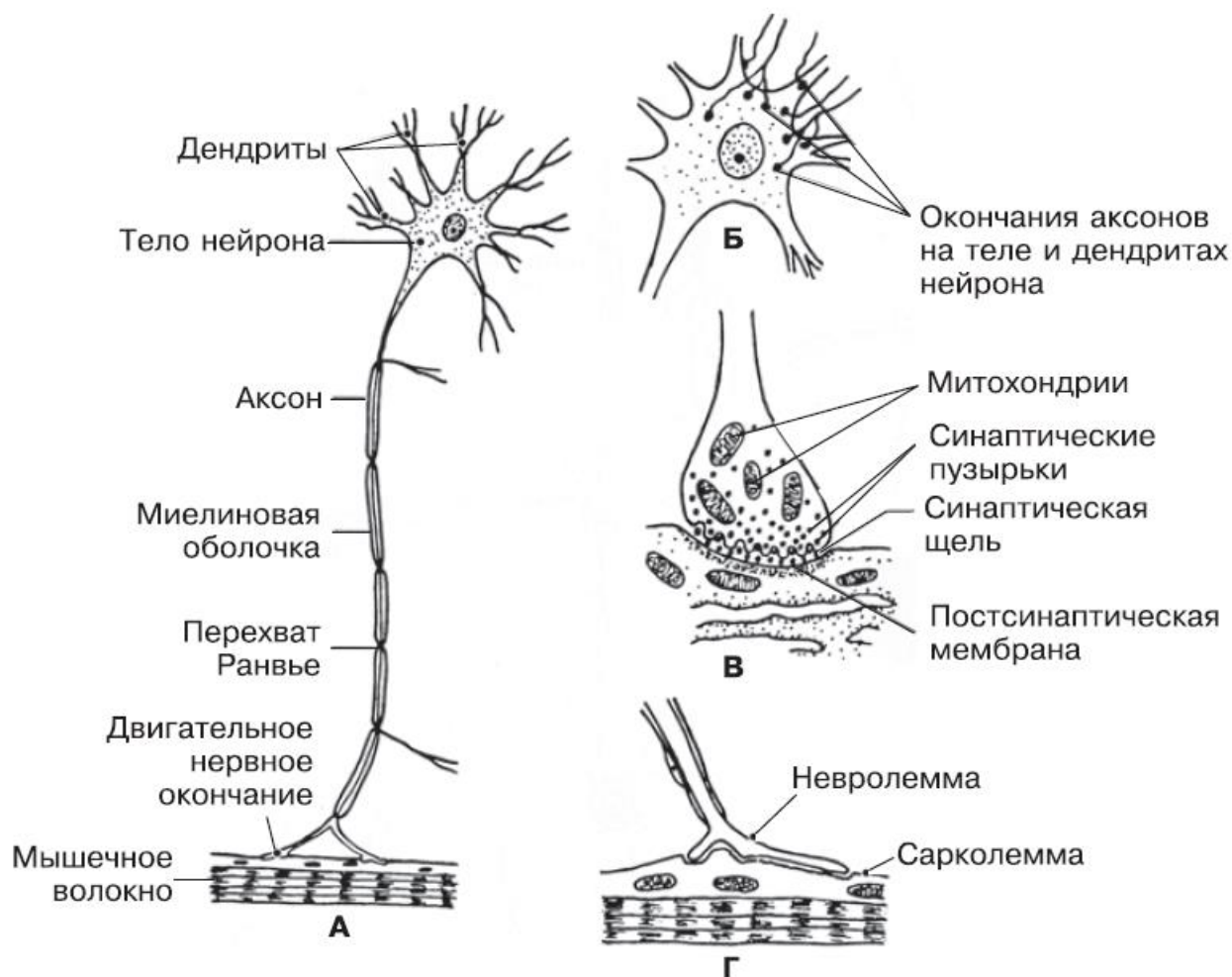


Рисунок 1.10 – А. Строение нейрона. Б. Тело нейрона. В. Синапс. Г. Нервное окончание [6]

Благодаря разветвлению аксона возбуждение от одного нейрона одновременно передается многим нервным клеткам. В результате осуществляется распределение поступающей с нервными импульсами информации между многими нейронами, что составляет один из элементов аналитической деятельности нервной системы. Функциональная разнородность отростков нервной клетки обеспечивает направленную передачу нервного возбуждения. Мультиполярность многих нейронов создает условия для одновременного восприятия и обработки каждым

нейроном различных потоков информации, что лежит в основе синтетической деятельности нервной системы.

Синапс — это контактное соединение одного нейрона с другим. В его формировании принимает участие аксон одного нейрона, образующий окончания на дендритах или теле другого нейрона. Посредством синапса нервный импульс передается от одного нейрона к другому. Передача возбуждения осуществляется при участии специальных веществ-передатчиков [6].

Исследования Лоренто де Но показали, что основной единицей активности в неокортексе служит вертикально расположенная группа клеток с множеством связей между этими клетками по вертикальной оси и малым числом в горизонтальном направлении [5].

Джефф Хоккинс в своей статье [1] рассматривает работу сенсорной коры. А именно, рассматривается устройство колонок в неокортексе мозга и то, как они обеспечивают способность к обучению структуры окружающего мира.

Хоккинс предлагает теорию, основанную на предположении о том, что помимо участия в формировании представлений о мире, неокортекс также играет важную роль в образовании идентификации знакомых объектов и концепций. Он объясняет, что колонки образуют сеть нейронов, которые совместно с другими регионами мозга участвуют в формировании сложных представлений о мире, а также в процессе обучаются новым структурам и понятиям.

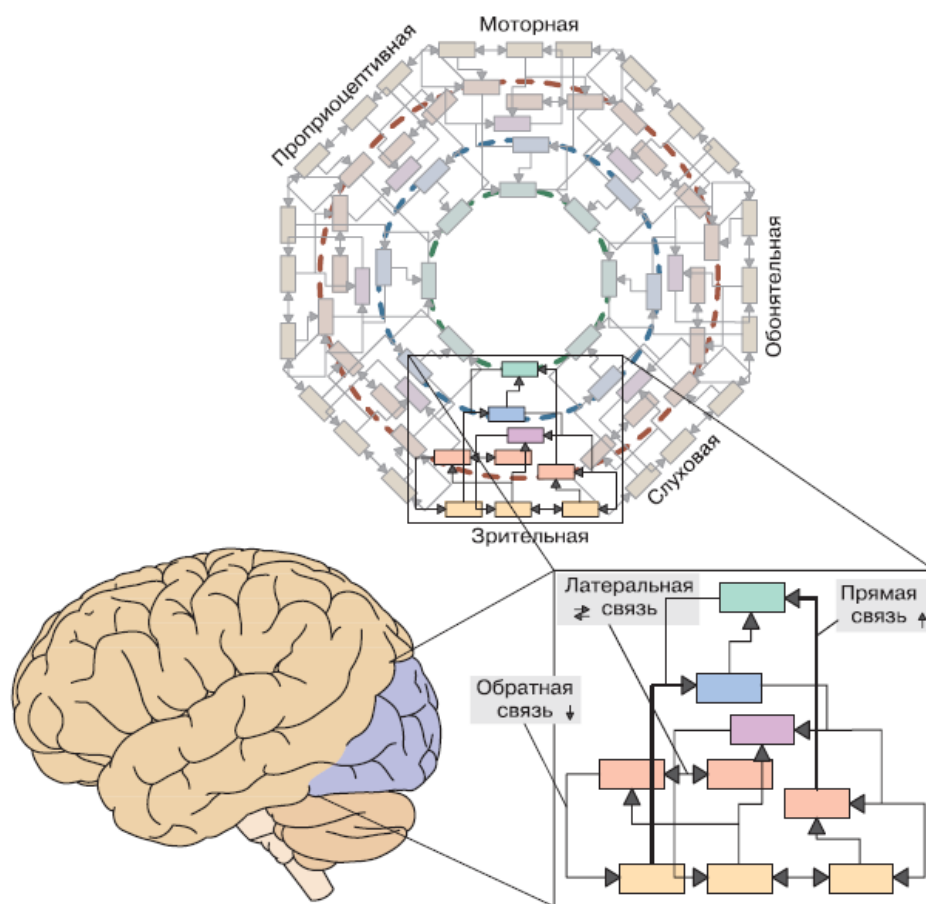


Рисунок 1.11 – Иерархия и связи на примере зрительного сектора

Очень хорошо объясняет алгоритм работы мозга психотерапевт Андрей Курпатов. Фрагмент из его книги «Чертоги разума» [13] приведен ниже.

«Сейчас я попытаюсь очень просто, буквально на пальцах объяснить, каким образом наш мозг занимается преобразованием фактов реальности в объяснения, которые и образуют ложную карту нашего с вами мировоззрения.

Мир, который нас окружает, представляется нам наполненным различными объектами, Вы можете видеть перед собой, например, стол, кресло, окно, кофейную чашку или книгу. И вы уверены, что видите эти объекты целиком.

На самом деле, это не так. Посмотрите сейчас на свою руку. Да, буквально — взгляните на неё! С ней же всё нормально? Пять пальцев — правильно? Ногти, линии на ладони, кожные складки в районе фаланг. Всё правильно, ничего не путаю? Нормальная такая рука, правда?

А теперь попытайтесь понять, что делает ваш взгляд, когда вы смотрите на свою кисть. Даже чтобы пересчитать пальцы на своей руке, вам необходимо несколько раз перевести взгляд с одного пальца на другой. А ещё — на ногти, на фаланги, линии, костяшки пальцев...

Быть может, вы уже заметили сосуды, просвечивающие из-под кожи, морщинки, зоны покраснения, волосы, заусеницы в основании ногтевого ложа.

Сколько раз вы перевели взгляд с одной точки на другую? Очень много! И я уж не говорю о множественных саккадах, которые мы сознательно заметить не можем. Вы осознали только намеренное передвижение взгляда от одной области вашей кисти к другой.

Теперь на секунду закройте глаза и представьте себе свою руку. Получилось? Посмотрите на свою руку ещё раз и сравните с воображаемой — они похожи, А теперь ещё раз переведите взгляд с пальца на палец, с линий на складки, с ногтей на сосуды... Теперь не очень похожи, правда? Похожи, но не совсем то, правильно?

Итак, правда состоит в том, что в каждый конкретный момент времени вы видите очень малую часть того, что перед вами. Буквально микроскопическую часть, кусочек кусочка. Но это же противоречит нашему предыдущему утверждению, что мир вокруг нас наполнен целыми предметами!

Вы видите свою руку целиком, кружку, которая, на столе, вы тоже видите целиком, а не фрагменты. Наконец, эту книгу тоже видите сразу всю, полностью, правильно? При этом ваш вам сосредоточен на конкретном слове... Как такое может быть?

Это масштабная мистификация, за которую и отвечает кора нашего мозга. Давайте представим сейчас (в очень упрощённом виде), как это работает. В коре нашего мозга шесть слоёв больших кортикальных колонок — вертикальная совокупность нервных клеток. Но, вообще говоря, каждая кортикальная колонка — это модуль, каждый из которых, в свою очередь, тоже состоит из колонок. Самые маленькие кортикальные колонки (их ещё называют мини-колонками) состоят из 80-120 нейронов, которые имеют одинаковое предназначение.

Чем ниже уровень и чем меньше кортикальная колонка — тем более простую информацию она обрабатывает.

Поступающая в мозг информация, миновав глубинные структуры мозга, где происходит первичный анализ информации, поступает на нижний уровень коры. Здесь идентифицируются самые элементарные вещи — линии, цвет, тоны и сила звука, эффект соприкосновения (при осязании).

А далее всё происходит, как в игре "Тетрис". Информационные сигналы, обработанные в низшем слое коры, поступают выше — на следующий слой.

При каждом переходе со слоя на слой информация обобщается, то есть что-то отбрасывается, что-то взаимно уничтожается, схлопывается, соединяется, модифицируется. И полученный результат, вырвавшись на самый верх, фиксируется как целостный образ.

Теперь вернёмся к руке. Когда вы на неё смотрите, происходит следующее: фотоны света, отражённые от конкретного участка кожи, попадают на сетчатку вашего глаза, превращаются в цифровой сигнал и передаются (причём по двум путям — короткому и длинному) в нижние отделы коры головного мозга.

Эти отделы способны увидеть лишь линии, чёрточки, цвета и т. д. с маленького участка кожи. Всё, эта информация прошла первичную обработку и побежала вверх. Тут вы чуть переводите взгляд, и снова такая же история — информация пошла к коре, а дальше от одного уровня коры к другому. Все эти бесчисленные потоки информации сходятся на самом верху в наружных слоях коры головного мозга.

Итак, вот как это выглядит:

- нижние слои коры хранят самую примитивную визуальную информацию, и соответствующие клетки реагируют тогда, когда она совпадает с поступившей информацией;
- верхние слои коры, напротив, хранят цельные образы — те самые, которые, как нам казалось, мы видим перед собой: стол, кресло, кружка, книга и т. д.

Но и это ещё не всё! Теперь верхние слои коры берут контроль на себя — они начинают диктовать нижним слоям, какую информацию принимать в расчёт, а какую отбрасывать.

То есть, если я уже идентифицировал некий предмет (клетки верхних слоёв коры активизировали во мне некий целостный визуальный образ), то я, по сути, перестаю его воспринимать. Да, отлетающие от него фотоны продолжают бомбардировать сетчатку моего глаза, но эта информация блокируется уже на нижних уровнях коры. Тут и так знают, что мне следует видеть, и не надо им лишний раз морочить голову!

Да, если кора не справилась и опознала предмет неверно, то дальше она, возможно, скорректируется. Но это-то и забавно: она смогла увидеть предмет, опознать его, подумать о нём и т.д., хотя на самом деле его там не было, там был какой-то другой предмет!

Так случается, например, что вы видите на улице знакомого человека (допустим, какого-нибудь Витю Петрова), а потом вдруг понимаете, что обознались. То есть информация от этого объекта сначала быстро промчалась вверх по вашей коре, нашла подходящий образ («Петров Витя»), а у вас возникло стойкое ощущение, что вы видите перед собой «того самого» человека — Витю.

И только если противоречивая информация продолжает и продолжает поступать, вы, наконец, начинаете испытывать некоторое замешательство, затем недоумение, смущение...

И вдруг снова отчётливо видите, что перед вами «совсем другой человек»! Информация, топтавшаяся до сего момента на нижних уровнях, пробила наконец заслонку, созданную верхними слоями коры, поднялась к ним, и там нашёлся новый образ для той же самой ситуации!

По этой же причине в сумерках вы можете принять одёжную вешалку за притаившегося в углу человека, а некий предмет на земле — за какое-то животное. Информация поступила в мозг, обработалась, активизировала образ верхнего слоя коры — и у вас возникает эффект узнавания. Вы вглядываетесь дополнительно,

получаете новую информацию, происходит новая итерация обработки сигналов, и вы понимаете, что ошиблись.».

1.3 Теория «тысячи мозгов»

1.3.1 Обзор теории

Несмотря на все достижения нейробиологии, мы мало продвинулись в решении ее главного вопроса: какова биологическая природа интеллекта?

В 1979 году вышла статья «Мысли о мозге» Фрэнсиса Крика. Она была опубликована на русском языке в сборнике [14]. Крик заметил, что ученые десятилетиями собирали данные о мозге. Они узнали очень много фактов, но никто не придумал, как собрать их во что-то значимое. Мозг подобен гигантской головоломке из тысяч кусочков. По словам Крика, мозг является загадкой не из-за отсутствия достаточного количества данных, а потому, что ученые не знают, как расположить части этого пазла. За сорок лет прошедших с тех пор, как Крик написал свое эссе, было сделано много значительных открытий о мозге, но в целом его наблюдение по-прежнему верно. Как интеллект возникает из клеток в голове, до сих пор остается загадкой. По мере того как с каждым годом появляется все больше кусочков головоломки, начинает казаться, что мы отдаляемся от понимания мозга, а не приближаемся к нему [12, 14].

В книге «1000 мозгов. Новая теория интеллекта» Джефф Хоккинс рассказывает о том, для чего нейроны коры головного мозга объединились в странные сообщества под названием «кортикальные колонки». В неокортексе их сто пятьдесят тысяч. Джефф Хокинс и его команда установили, что каждая колонка, составляющая кору головного мозга, создает собственную модель мира. В итоге образуется не одна модель, а тысячи, и наше восприятие — это коллективное решение кортикальных колонок, принятое ими путем голосования. Кортикальные колонки в разрезе неокортекса представлены на Рисунок 1.12.

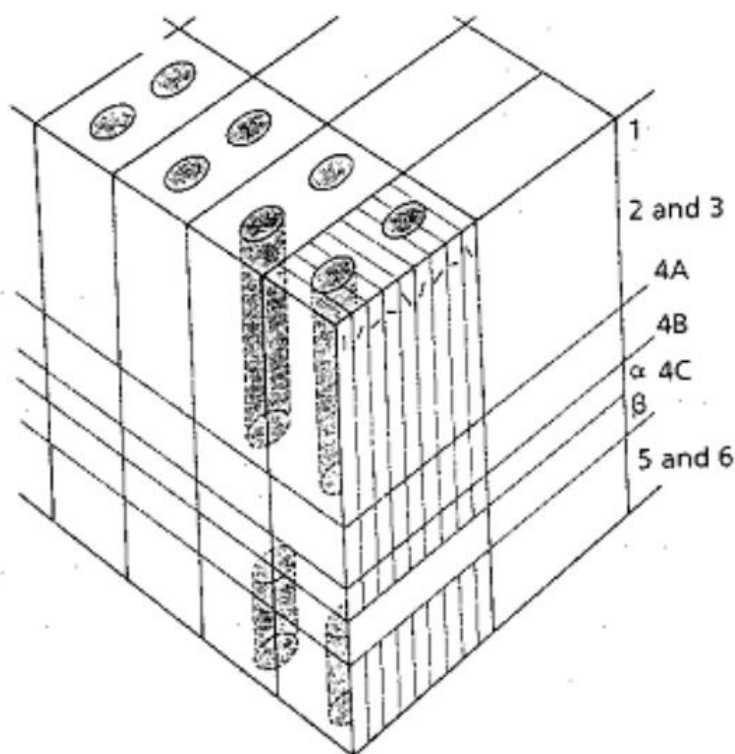


Рисунок 1.12 – Кортикальные колонки в разрезе слоев неокортекса

Новый взгляд на деятельность головного мозга авторы идеи назвали теорией «тысячи мозгов». Они утверждают, что не искусственные нейронные сети, а открытые ими законы работы неокортекса лягут в основу развития искусственного интеллекта в будущем [12].

Статья Джеффа Хоккинса "Why Neurons Have Thousands of Synapses, A Theory of Sequence Memory in Neocortex" [7] исследует вопрос, почему нейроны имеют тысячи синапсов и как это связано с памятью последовательностей в неокортексе.

Хоккинс предлагает теорию, основанную на предположении, что информация в мозге представлена последовательностями активаций нейронных групп. Согласно его модели, каждый нейрон может обрабатывать и хранить небольшую часть последовательности, а вся последовательность кодируется через синаптические связи на уровне сетей нейронов.

В статье поясняется, почему тысячи синапсов, соединяющих один нейрон, играют важную роль в обработке информации о последовательностях. В частности, Хоккинс объясняет, что благодаря множеству синапсов, нейрон может

обнаруживать и запоминать временные шаблоны активаций (паттерны), что важно для контекстного понимания последовательностей.

Статья также обсуждает роль синаптических связей и пластичности нервных сетей в формировании и сохранении памяти. Хоккинс обращает внимание на то, что каждая синаптическая связь может иметь различные веса и изменяться в зависимости от активности нейронов. Это позволяет мозгу гибко адаптироваться и улучшать эффективность обработки информации.

В статье также разбираются различные модели и эксперименты, подтверждающие исследования Хоккинса. Он предлагает не только теоретическую основу, но и конкретные эмпирические примеры, чтобы подкрепить свою модель памяти.

В следующей по порядку статье «A Theory of How Columns in the Neocortex Enable Learning the Structure of the World» [1] предлагается продолжение исследования модели неокортекса. Сохраняется принцип того, что неокортикальные регионы организованы в столбцы и слои. Связи между слоями проходят преимущественно перпендикулярно поверхности, что предполагает столбчатую функциональную организацию. Некоторые слои имеют дальние возбуждающие боковые связи, что позволяет предположить взаимодействие между колонками. Боковые связи показаны горизонтальными синими стрелочками на Рисунок 1.13.

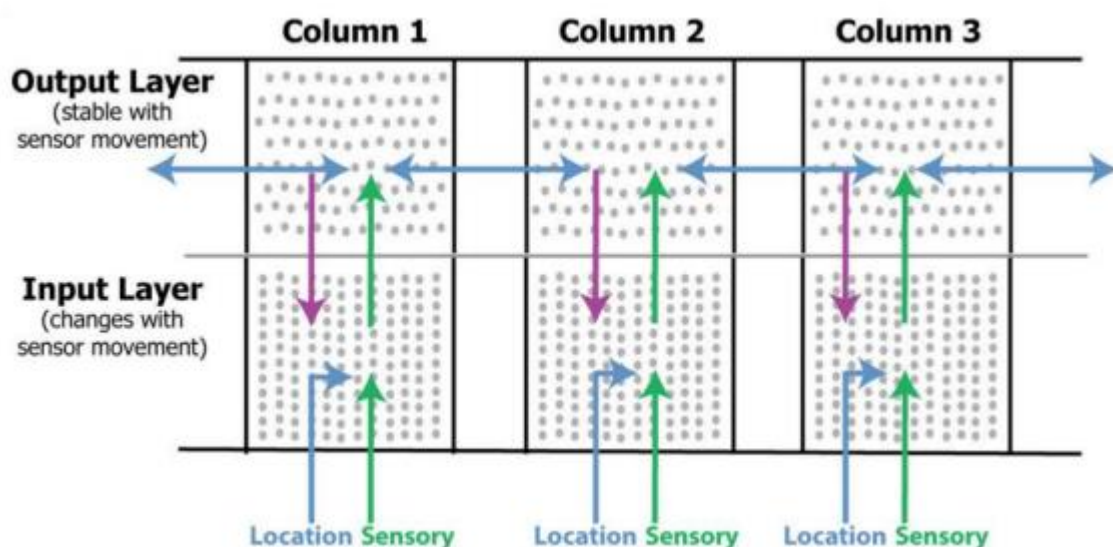


Рисунок 1.13 – Взаимодействие нескольких кортикальных колонок

Подобные модели взаимодействия существуют во всех регионах, но их точная роль остается загадкой. Делается акцент на многослойной структуре, которая обеспечивает надежное обучение и распознавание объектов. Каждый столбец интегрирует изменяющиеся входные данные с течением времени для изучения полных прогнозных моделей наблюдаемых объектов. Возбуждающие боковые связи между столбцами позволяют сети быстрее выводить объекты на основе частичного знания соседних столбцов. Поскольку столбцы интегрируют входные данные во времени и пространстве, сеть изучает модели сложных объектов, которые выходят далеко за пределы рецептивного поля отдельных ячеек. Сетевая модель представляет новую функцию кортикальным столбцам. Предполагается, что представление местоположения относительно воспринимаемого объекта рассчитывается в особых слоях каждого столбца. Сигнал местоположения подается в качестве входных данных в сеть, где он объединяется с сенсорными данными. Модель содержит два слоя и один или несколько столбцов. Моделирование показывает, что с помощью правил обучения, подобных Хеббиану, небольшие сети с одним столбцом могут научиться распознавать сотни объектов. Многоколоночные сети распознают объекты со значительно меньшим количеством движений сенсорных рецепторов. Учитывая повсеместное распространение столбчатых и ламинарных паттернов связности по всему неокортексу, есть основания полагать,

что столбцы и регионы обладают более мощными возможностями распознавания и моделирования, чем предполагалось ранее [1].

Некоторые ученые, например, Ричард Докинз, считают, что дело даже не в том, что Хокинс недооценивает мощь искусственного интеллекта. Хоккинс считает, что большинство современных исследований идут в неверном направлении. Правильный путь, по его мнению, состоит в том, чтобы понять, как работает мозг, позаимствовать его методы и значительно ускорить их [12].

1.3.2 Серво-моторная обработка информации

Долгое время считалось, что информация поступает в неокортекс через «сенсорные области», поднимается и опускается по иерархии областей и, наконец, опускается в «моторную кору». Там она проецируется на нейроны спинного мозга, которые двигают мышцами и конечностями. На текущий момент известно, что это не совсем так. В каждой исследованной области ученые обнаружили клетки, которые проецируются на какие-либо старые части мозга, связанные с движением. Например, зрительные области, получающие входные данные от глаз, посылают сигнал в ту старую часть мозга, которая отвечает за движение глаз. А слуховые области, получающие информацию от ушей, проецируются на ту часть мозга, которая двигает головой. Движение головы изменяет то, что вы слышите, подобно тому как движение глаз изменяет то, что вы видите.

Имеющиеся доказательства указывают на то, что сложная схема, встречающаяся повсюду в неокортексе, выполняет сенсомоторную задачу. Нет исключительно моторных или сенсорных областей.

Таким образом, неокортекс — это орган интеллекта. Лист нервной ткани размером с салфетку, разделенный на десятки областей. Есть области, отвечающие за зрение, слух, осязание и речь. Есть области, которые не так легко обозначить, — они отвечают за продумывание и планирование на высоком уровне. Эти области соединены друг с другом пучками нервных волокон. Некоторые связи между областями иерархические, что позволяет предположить, что информация передается из области в область упорядоченным образом, подобно блок-схеме. Но есть и другие

связи между областями, в которых, по-видимому, мало порядка, — это позволяет предположить, что, информация распространяется в разные части одновременно. Все области, независимо от того, какую функцию выполняют, похожи друг на друга.

Это же утверждал и Маунткасл: «Не может быть никаких сомнений в доминирующем влиянии дарвиновской революции середины девятнадцатого века на представления о структуре и функциях нервной системы. Идеи Спенсера, Джексона, Шеррингтона и многих, кто последовал за ними, основывались на эволюционной теории, согласно которой мозг развивается в филогенезе путем последовательного добавления большего количества частей. Согласно этой теории, каждое новое добавление или расширение сопровождалось разработкой более сложного поведения и в то же время подвергалось регулированию более старые и примитивные части и предположительно более примитивное поведение, которое они контролируют» [5].

В этих трех предложениях Маунткасл говорит о том, что мозг увеличивался в процессе эволюции, добавляя новые части поверх старых. Старые части управляют более примитивным поведением, в то время как новые создают более сложное. Хотя большая часть мозга увеличилась за счет добавления новых частей поверх старых, неокортекс вырос не настолько, чтобы занимать 70% нашего мозга. Неокортекс стал большим, создав множество копий одного и того же – базовой схемы.

Неокортекс начинался с малого, но затем становился больше, не создавая ничего нового, а копируя базовую схему снова и снова. По мере роста неокортекс увеличивается по площади, но не по толщине. Маунткасл утверждал, что хотя неокортекс человека намного больше, чем у крысы или собаки, все они состоят из одного и того же элемента, просто у нас больше копий этого элемента [12]. Сравнение неокортексов разных видов млекопитающих представлено на Рисунок 1.14.

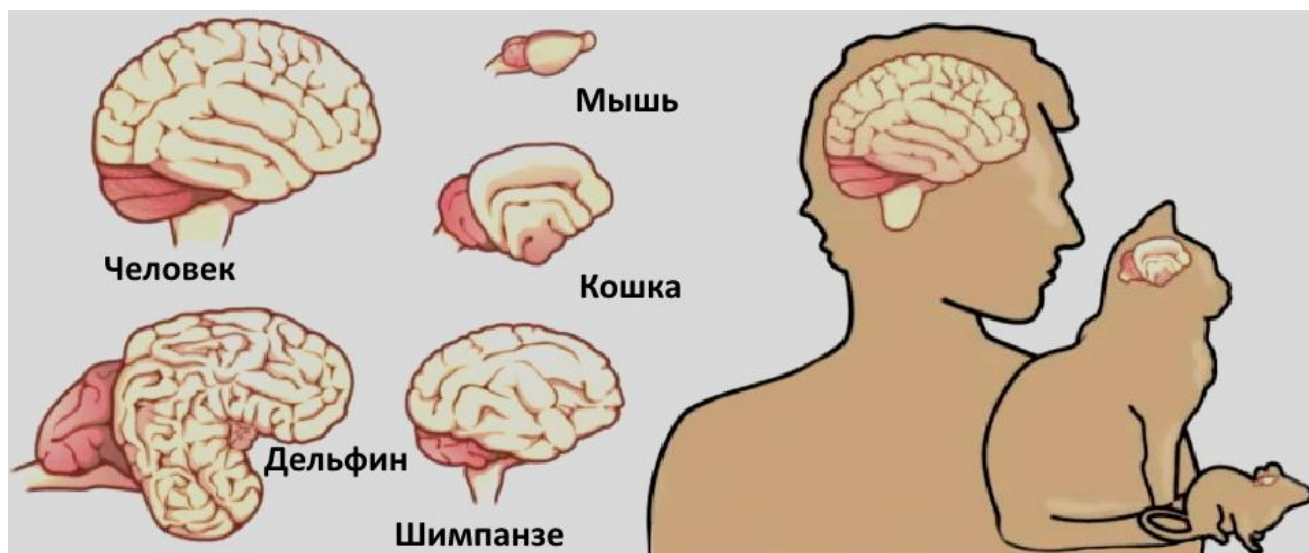


Рисунок 1.14 – Сравнение неокортексов разных видов млекопитающих [17]

«Короче говоря, в моторной коре головного мозга нет ничего по сути моторного, как и сенсорного в сенсорной коре головного мозга. Таким образом, выяснение режима работы локальной модульной схемы в любом месте неокортекса будет иметь большое обобщающее значение» [5].

В этих двух предложениях Маунткэсл резюмирует основную идею, изложенную в эссе. Он говорит, что каждая часть неокортекса работает по одному и тому же принципу. Все, что мы считаем интеллектом, — от зрения до прикосновения, языка и мышления на высоком уровне, — в основе своей одно и то же.

Неокортекс разделен на десятки областей, каждая из которых выполняет свою функцию. Если посмотреть на неокортекс снаружи, то не получится увидеть области; здесь нет разграничений, так же как на спутниковом снимке не видны политические границы между странами. Если же посмотреть на неокортекс в разрезе, можно увидеть сложную и детализированную структуру. Однако детали выглядят одинаково независимо от того, в какую область коры головного мозга смотреть. Участок коры головного мозга, отвечающий за зрение, выглядит как участок коры головного мозга, отвечающий за осязание, который выглядит как участок коры головного мозга, отвечающий за язык.

Маунткасл предположил, что причина, по которой области выглядят одинаково, заключается в том, что все они делают одно и то же. То, что отличает их, — это не внутренние функции, а связи. Если соединить область коры с глазами, получится зрение; если соединить ту же область с ушами, то получится слух; а если соединить эти области с другими областями, то получится высшее мышление, такое как язык. Затем Маунткасл указывает, что если получится обнаружить основную функцию любой части неокортекса, то станет понятно, как он работает в целом [12].

Фундаментальная единица неокортекса, единица интеллекта, — это кортикальная колонка. Если посмотреть на поверхность неокортекса, то кортикальная колонка занимает около одного квадратного миллиметра. Она проходит через всю толщину в 2,5 миллиметра; таким образом, ее объем — 2,5 кубических миллиметра. Согласно этому определению, в неокортексе человека насчитывается примерно сто пятьдесят тысяч кортикальных колонок. Можно сказать, что кортикальная колонка похожа на маленький кусочек тонких спагетти. Человеческий неокортекс подобен ста пятидесяти тысячам коротеньких кусочков спагетти, расположенных вертикально друг рядом с другом.

Кортикальные колонки не видны под микроскопом. За редким исключением, между ними нет видимых границ. Ученые знают, что они существуют, потому что все клетки в одной колонке будут реагировать на одну и ту же часть сетчатки или на один и тот же участок кожи, но затем все клетки в следующей колонке будут реагировать на другую часть сетчатки или другой участок кожи. Группа ответов — это то, что определяет колонка. Это наблюдается во всех частях неокортекса [5, 12].

1.3.3 Обучение

Мозг запоминает много вещей. У человека есть постоянные воспоминания (например, о том, где он вырос), а также временные (например, о том, что он ел на ужин вчера вечером). Также он обладает базовыми знаниями (например, как открыть дверь или как правильно написать слово, «словарь»). Все это хранится с помощью синапсов, связей между нейронами.

Основная идея в методе обучения мозга заключается в том, что каждый нейрон имеет тысячи синапсов, которые соединяют его с тысячами других нейронов. Если два нейрона активизируются одновременно, то связь между ними укрепляется. Когда мы чему-то учимся, связи укрепляются, а когда что-то забываем, связи ослабевают. Эта основная идея была предложена Дональдом Хеббом в 1940-х годах, и сегодня ее называют хеббовским обучением.

В течение многих лет считалось, что связи между нейронами в мозге взрослого человека фиксированы. Считалось также, что обучение связано с увеличением или уменьшением силы синапсов. Именно так по-прежнему происходит обучение в большинстве искусственных нейронных сетей.

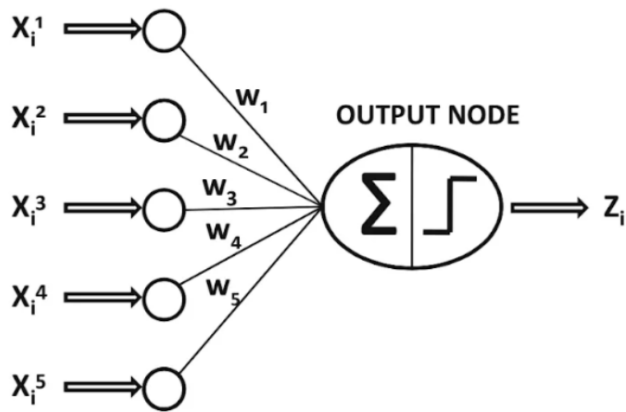
Однако за последние несколько десятилетий ученые обнаружили, что во многих частях мозга, включая неокортекс, образуются новые синапсы, а старые исчезают. Каждый день многие синапсы на отдельном нейроне будут исчезать, и их заменят новые. Таким образом, большая часть обучения происходит за счет формирования новых связей между нейронами, которые ранее не были связаны. Забывание происходит тогда, когда старые или неиспользуемые соединения полностью удаляются.

Связи в нашем мозге хранят модель мира, которую мы узнали благодаря опыту. Каждый день мы испытываем что-то новое и добавляем новые знания в модель, формируя новые синапсы. Нейроны, которые активны в любой момент, представляют наши текущие мысли и восприятие [12].

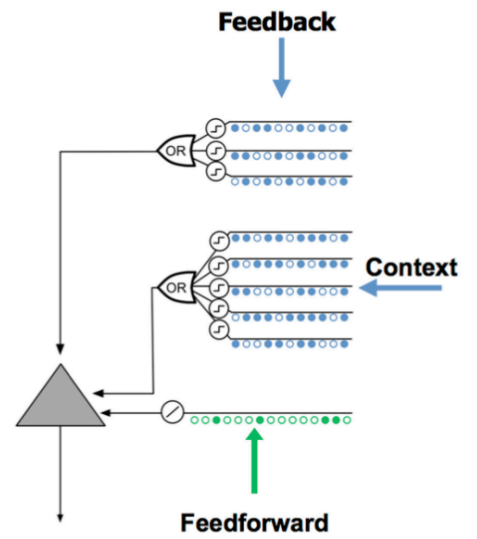
1.3.4 Отличия от искусственных нейронных сетей

Как было сказано ранее, одно из отличий – современные искусственные сети отталкиваются от «силы» синапса. В теории «Тысячи мозгов» считается, что достаточно наличия синапса, а его «сила» не влияет на точность предсказания.

Так же, современные нейронные сети используют терминологию биологии, но у них нет цели повторить в точности процессы, протекающие в неокортексе. Например, они не учитывают дендриты и дендритные спайки (Рисунок 1.15).



а) Персептрон



б) Пример HTM нейрона

Рисунок 1.15 – Сравнение персептрона и HTM нейрона

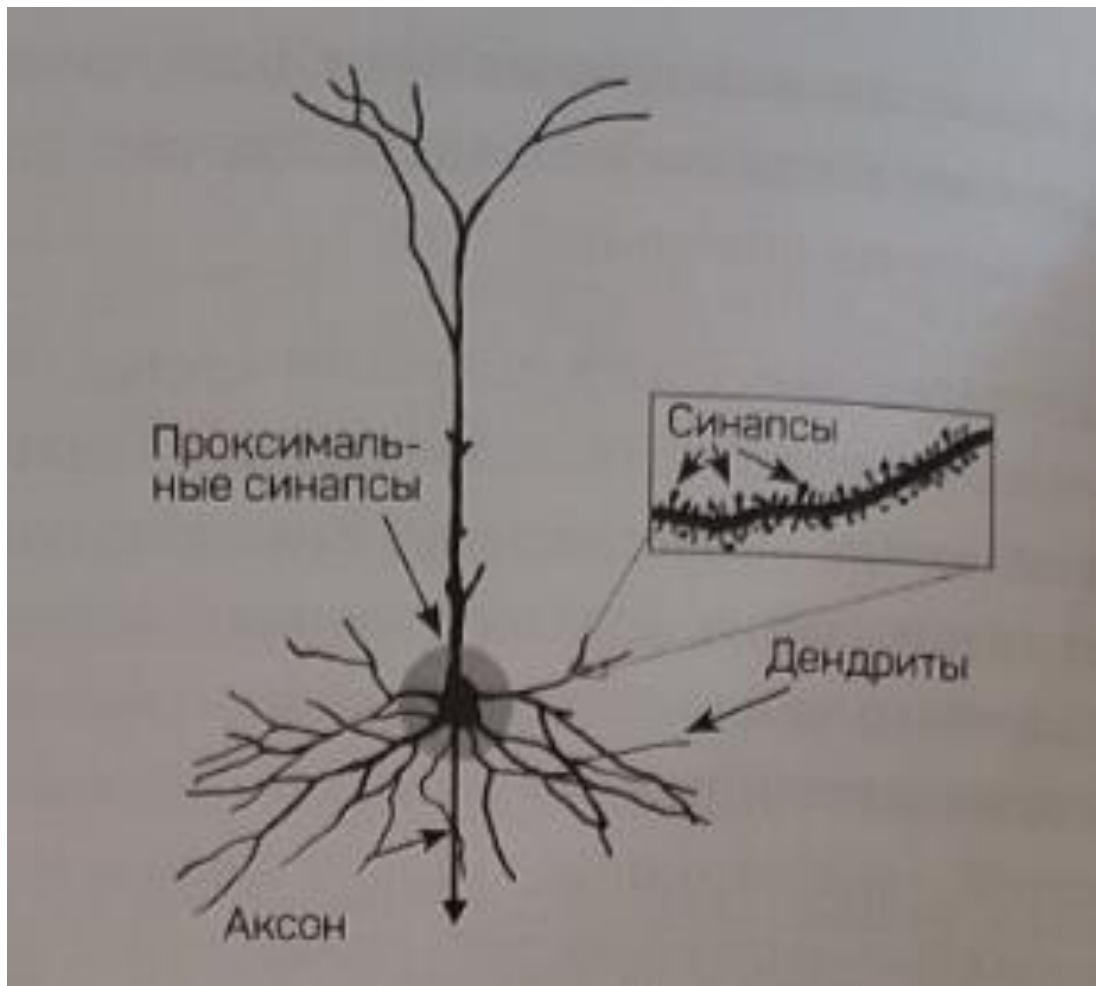


Рисунок 1.16 – Модель нейрона с дендритами и синапсами

На Рисунок 1.16 изображен наиболее распространенный в неокортексе тип нейронов. Подобные нейроны имеют тысячи, иногда десятки тысяч синапсов, расположенных вдоль, ветвей дендритов. Некоторые из дендритов находятся рядом с телом клетки (в нижней части изображения), а некоторые — далеко от нее (сверху). На вставке показан увеличенный вид одной ветви дендрита, видно насколько малы и плотно «упакованы» синапсы.

Каждый бугорок вдоль дендрита представляет собой один синапс. Так же выделена область вокруг тела клетки. Синапсы в этой области называются проксимальными. Если проксимальные синапсы получают достаточное количество входных данных, то нейрон будет активироваться. Спайк начинается в теле клетки и распространяется к другим нейронам через аксон. Аксон не виден на этом рисунке, стрелкой показано то место, где он должен находиться. Это классический вид нейрона, если рассматривать проксимальные синапсы и тело клетки.

Как ни странно, менее 10% синапсов клетки находятся в проксимальной области. Остальные 90% находятся слишком далеко, чтобы вызвать спайк. Если входной сигнал поступает в один из этих дальних синапсов, как показано на Рисунок 1.16, то он почти не влияет на тело клетки. Все, что могли сказать исследователи, так это то, что эти синапсы выполняли какую-то модулирующую роль. В течение многих лет никто не знал, за что отвечают 90% синапсов в неокортексе.

Примерно с 1990 года ситуация изменилась. Ученые обнаружили новые типы спайков, которые перемещаются вдоль дендритов. Раньше мы знали только об одном типе: начинающемся в теле клетки и перемещающемся по аксону, чтобы достичь других клеток. Теперь стало известно, что были и другие спайки: перемещающиеся вдоль дендритов. Один тип дендритного спайка происходит тогда, когда группа из двадцати или около того синапсов, расположенных друг рядом с другом на ветви дендрита, получает входные данные одновременно. Как только возникает дендритный всплеск, он перемещается вдоль дендрита до тех пор, пока не

достигнет тела клетки. Попад туда, он повышает тонус клетки, но не настолько, чтобы активировать ее.

Нейрон остается в этом подвешенном состоянии в течение некоторого времени, прежде чем вернуться к нормальному. Долгое время ученые не могли понять, для чего нужны дендритные спайки, если они не приводят к активации.

Не зная, для чего нужны дендритные спайки, исследователи искусственного интеллекта используют имитированные нейроны, у которых их нет. У них также нет дендритов и синапсов.

Хокинс считает, что дендритные спайки создают предсказания. Дендритный спайк возникает тогда, когда набор синапсов, расположенных близко друг к другу на отдаленном дендрите, получает входные данные одновременно. И это означает, что нейрон распознал паттерн активности в некоторых других нейронах. Когда обнаруживается паттерн активности, возникает дендритный спайк, который повышает напряжение в теле клетки, переводя ее в так называемое прогностическое состояние. Затем нейрон настраивается на спайк. Это похоже на то, как бегун, услышавший «На старт... внимание...», готовится бежать. Если нейрон в прогностическом состоянии впоследствии получает достаточно входных данных для создания потенциального спайка действия, то клетка активируется немного раньше, чем если бы нейрон не находился в прогностическом состоянии. Предположим, что есть десять нейронов, которые распознают один и тот же паттерн на своих проксимальных синапсах. Это похоже на то, как десять бегунов на стартовой линии ждут одного и того же сигнала, чтобы начать забег. Один бегун, слышит «На старт... внимание...» и предвкушает, что забег вот-вот начнется. Когда раздается сигнал «Марш», он начинает бежать быстрее, чем бегуны, которые не слышали предыдущих сигналов и, соответственно, не подготовились к забегу. Увидев, что первый бегун вырвался вперед, другие бегуны сдаются и даже не стартуют. Такого рода конкуренция происходит во всем неокортексе [12].

В каждой мини-колонке несколько нейронов реагируют на один и тот же входной сигнал. Они подобны бегунам на стартовой линии — все ждут одной и той

же команды. Они готовы в любой момент начать действовать, если поступит ожидаемый сигнал. Однако если один или несколько нейронов находятся в состоянии прогнозирования, то, согласно нашей теории, активизируются только они, в то время как другие нейроны подавляются. Таким образом, когда поступает неожиданный входной сигнал, одновременно срабатывают несколько нейронов. Если входные данные предсказаны, то активными становятся только нейроны в прогностическом состоянии. Это интересное наблюдение о неокортексе: неожиданные входные сигналы вызывают гораздо большую активность, чем ожидаемые.

1.4 Большие данные и аномалии

Анализ больших данных - это процесс извлечения ценных знаний и информации из обширных и разнообразных наборов данных. Эти данные обычно слишком большие или сложные для обработки с помощью традиционных методов. Анализ больших данных и теория 1000 мозгов тесно взаимосвязаны. Анализ больших данных позволяет нам собирать и обрабатывать огромные объемы данных, которые могут быть сгенерированы множеством людей.

Теория «1000 мозгов» может помочь в анализе больших данных, предоставляя основу для разработки новых алгоритмов и методов машинного обучения. Вот несколько способов, которыми эта теория может быть применена в этом контексте:

- Иерархическая обработка: Теория «1000 мозгов» предполагает, что мозг обрабатывает информацию иерархически, от простых функций к более сложным. Это представление может быть использовано для разработки алгоритмов машинного обучения, которые могут эффективно обрабатывать большие объемы данных, разбивая их на более мелкие, управляемые части.

- Репрезентации памяти на основе предсказаний: Теория «1000 мозгов» также подчеркивает роль памяти на основе предсказаний в обработке информации мозгом. Это может привести к созданию новых методов для хранения и извлечения данных, которые могут улучшить производительность алгоритмов машинного обучения на больших объемах данных.

- Адаптивные алгоритмы: Теория «1000 мозгов» предполагает, что мозг постоянно адаптируется и учится. Это может служить основой для разработки адаптивных алгоритмов машинного обучения, которые могут подстраиваться под изменяющиеся данные и улучшать свою производительность с течением времени.

В целом, теория «1000 мозгов» обеспечивает ценную основу для исследования и разработки новых подходов к анализу больших данных. А реализация биоподобной модели на основе этой теории подходит для определения аномалий в данных [15].

Например, датчики являются одним из источников больших данных. Поскольку датчики проникают в нашу повседневную жизнь, мы наблюдаем экспоненциальный рост доступности потоковых данных временных рядов.

Во многом благодаря развитию Интернета вещей (IoT) и подключенных источников данных в реальном времени у нас теперь есть огромное количество приложений с датчиками, которые производят важные данные, которые меняются со временем. Эффективный анализ этих потоков может предоставить ценную информацию для любого варианта использования и приложения.

Обнаружение аномалий в потоковых данных в реальном времени имеет практическое и важное применение во многих отраслях. Такие варианты использования, как профилактическое обслуживание, предотвращение мошенничества, обнаружение неисправностей и мониторинг, можно найти во многих отраслях, таких как финансы, информационные технологии, безопасность, медицина, энергетика, электронная коммерция, сельское хозяйство и социальные сети.

Обнаружение аномалий может дать полезную информацию в критических сценариях, но надежных решений пока не существует. С этой целью предлагается новое и надежное решение для решения проблем, связанных с обнаружением аномалий в реальном времени. Аномалией будем считать момент времени, когда поведение системы необычно и значительно отличается от предыдущего нормального поведения. Аномалия может означать негативное изменение в системе,

например колебание частоты вращения турбины реактивного двигателя, возможно, указывающее на скорый отказ. Аномалия также может быть положительной, например, аномально большое количество кликов на странице нового продукта, что означает более высокий, чем обычно, спрос.

В любом случае аномалии в данных идентифицируют аномальное поведение с потенциально полезной информацией. Аномалии могут быть пространственными, при этом отдельный экземпляр данных может считаться аномальным по отношению к остальным данным, независимо от того, где он находится в потоке данных.

Аномалия также может быть временной. или контекстуальной, если временная последовательность данных имеет значение; т. е. экземпляр данных является аномальным только в определенном временном контексте, но не иначе. Временные аномалии часто незаметны и их трудно обнаружить в реальных потоках данных. Обнаружение временных аномалий в практических приложениях ценно, поскольку они могут служить ранним предупреждением о проблемах в базовой системе [16].

2 Математическая модель

2.1 Модель НТМ нейрона и одного кортикального слоя

Основная единица хранения информации в кортикальной колонке – синапс, показывающий, есть связи или нет (Рисунок 2.1).

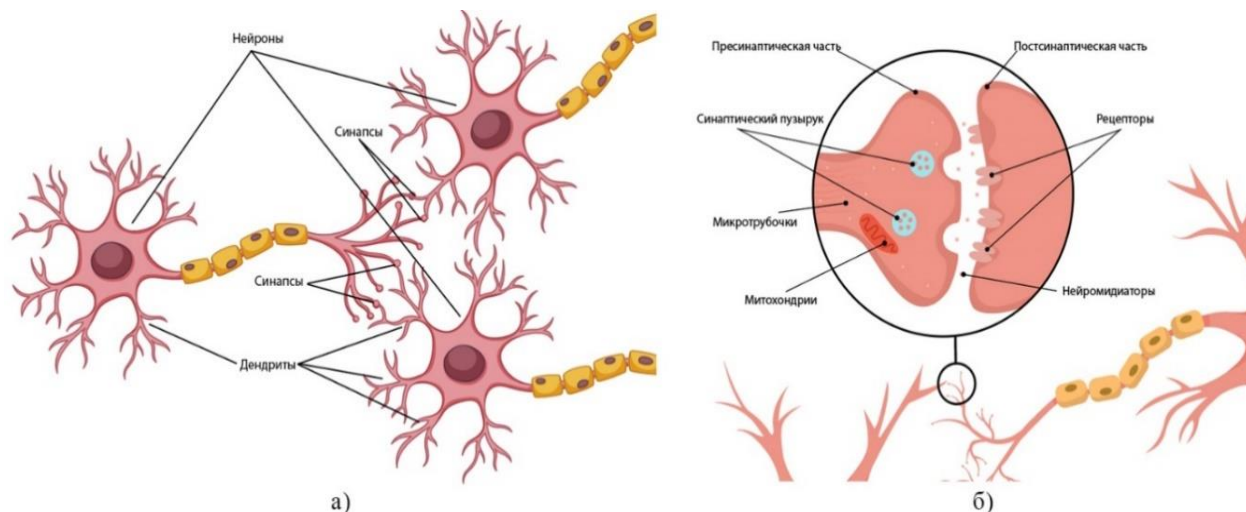


Рисунок 2.1– Модель нейрона (а) и структура синапса (б) [18]

С биологической точки зрения устройство колонки неокортекса можно представить в виде схемы, показанной на Рисунок 2.2.

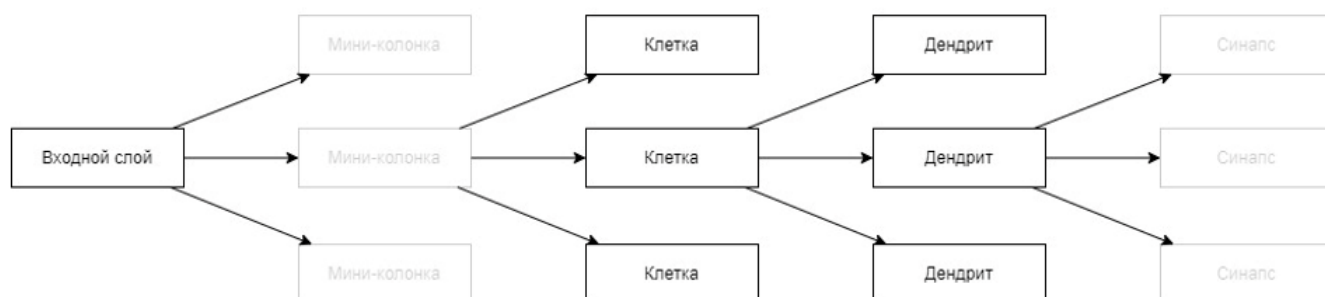


Рисунок 2.2 – Модель одного кортикального слоя

Более бледное изображение мини-колонки обусловлено тем, что это условная единица. Организация клеток внутри колонки может не соответствовать физическому расположению в столбец, но важно понимать, эти клетки относятся к одной условной группе, поэтому клетки организованы в «мини-колонки» для удобства работы с ними в рамках модели. Для синапса похожее обоснование, синапс – соединение дендрита и аксона, по сути, условная единица, не имеющая физического проявления (но только в рамках данной модели).

С точки зрения хранения данных, структура будет иметь вид, представленный на Рисунок 2.3 [19].

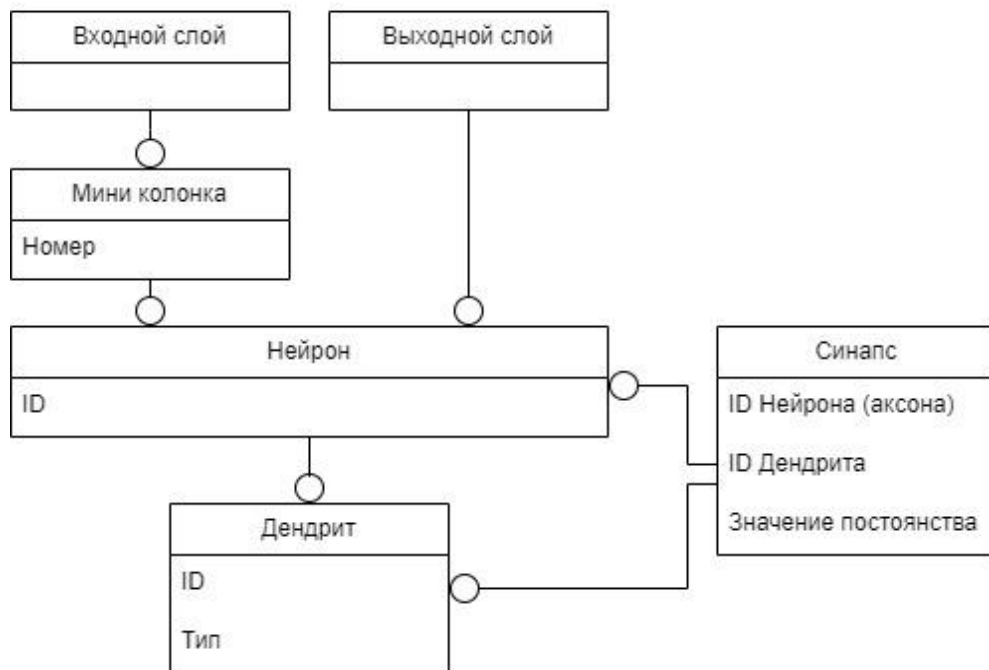


Рисунок 2.3 – Реляционная схема данных

Синапсы, по сути, представлены парой чисел – значение постоянства и вес синапса. Значение постоянства – непрерывная величина от 0 до 1, указывающая на то, на сколько прочная установлена связь и есть ли она вообще. Так же существует пороговое значение $threshold \in [0; 1]$, которое говорит о том, существует ли синапс. Если значение постоянства больше порогового значения, то вес синапса равен единице, иначе нулю. Образование синапса представлено на Рисунок 2.4. В процессе обучения значения постоянства синапсов меняются, что влияет на «стойкость» синапсов. Если какой-то паттерн повторяется из раза в раз, то значение постоянства будет увеличиваться, что будет говорить о том, что паттерн распознается верно [8].

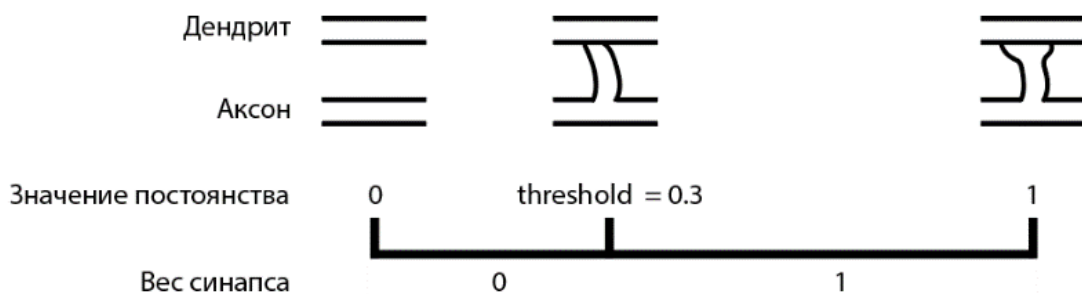


Рисунок 2.4 – Образование синапса

Первоначальная инициализация значений синапсов целесообразно проводить значениями, полученными распределением Пуассона. Опытным путем было выяснено, что строить распределение нужно с центром в точке 0.12, тогда лишь некоторые значения будут превышать заданное *threshold* и образовывать синапс.

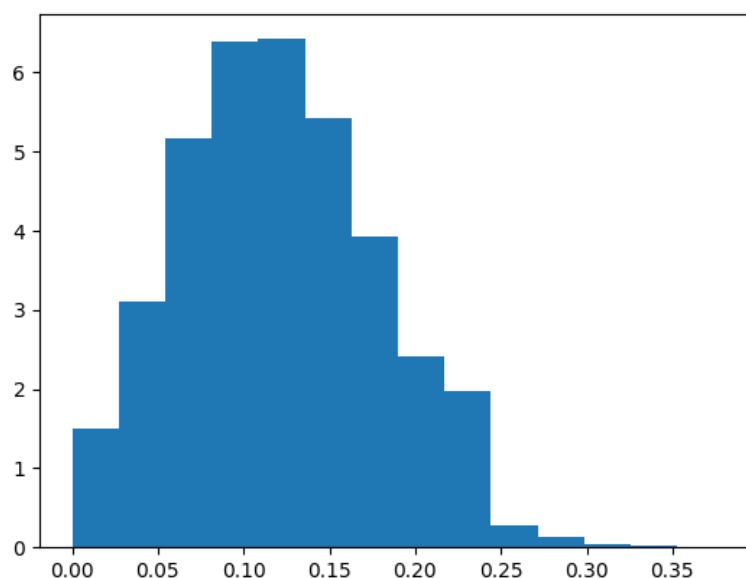


Рисунок 2.5 – Гистограмма значений постоянства синапсов, полученных с помощью распределения Пуассона.

2.2 Многоуровневая модель кортикальной колонки

Общая модель системы приводится на Рисунок 2.6.

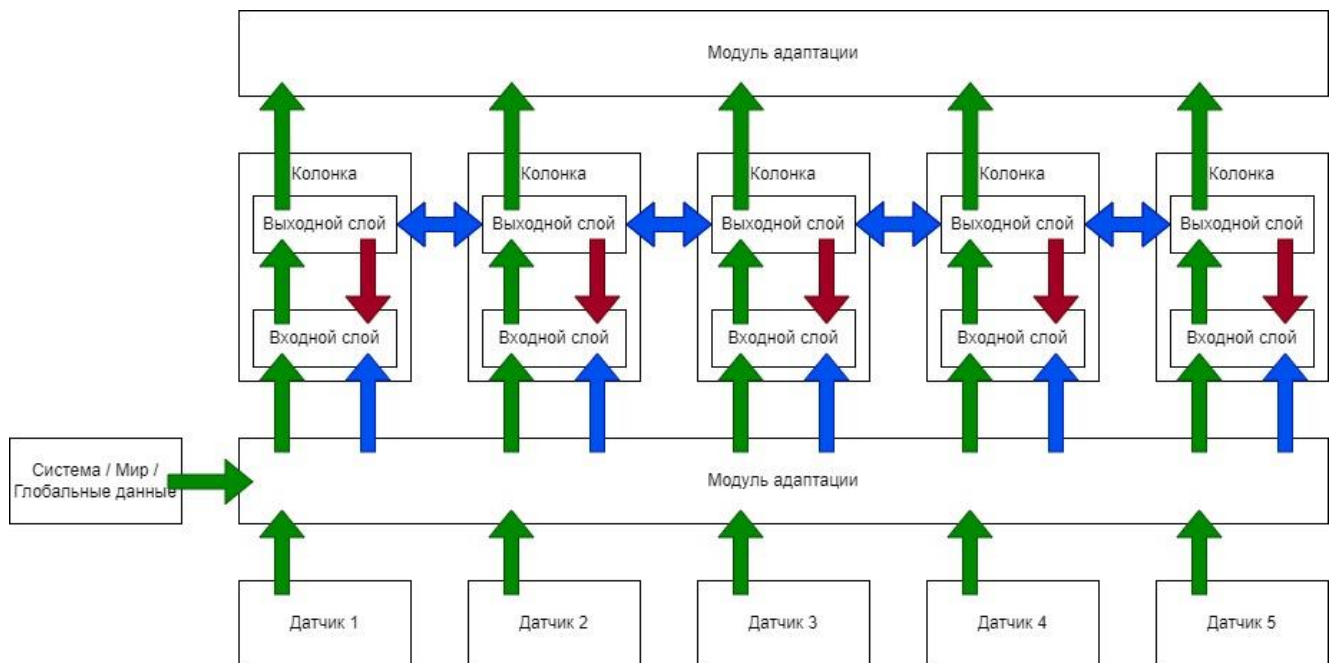


Рисунок 2.6 – Общая модель двухуровневой распределенной системы.

Цветовая индикация:

1. Зеленый - прямые сигналы, которые идут по проксимальным дендритам.

Служат для непосредственной активации нейронов;

2. Синий – боковые (латеральные) сигналы, идут по базальным дендритам.

Служат для деполяризации клеток, то есть предвещают активацию. Клетки с боковой связью активируются быстрее и пресекают активацию клеток, у которых такой связи нет;

3. Красный – обратная связь, апикальные дендриты. это обратная связь от

внешнего слоя. Если клетка во внешнем слое была активирована, то она может предсказать активацию клеток входного слоя, деполяризовав их по апикальному дендриту.

При этом следует отметить, что колонки могут общаться не только с соседними колонками в рамках одной области неокортекса, но и с колонками в других областях (Рисунок 2.7).



Рисунок 2.7 – Взаимодействие колонок с другими областями неокортекса

Причем, может быть даже такая ситуация, что связи колонок образуют полносвязный граф (Рисунок 2.8).

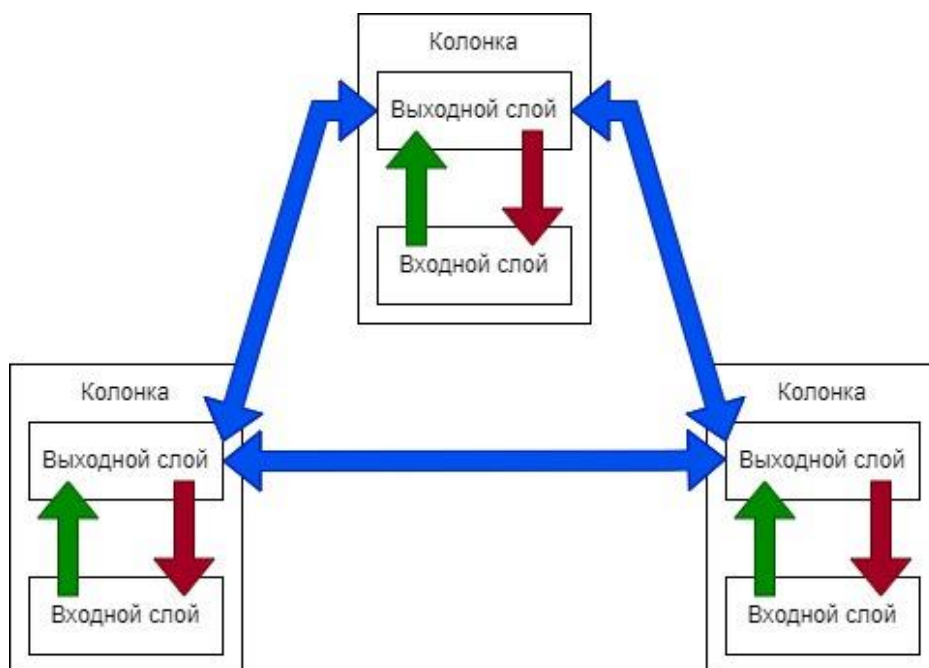


Рисунок 2.8 – Взаимодействие нескольких колонок

Как правило, на один «датчик» приходится несколько кортикальных колонок, которые в свою очередь «общаются» между собой и «принимают» коллективное решение о распознавании паттерна активации. Принимая этот аспект во внимание, можно преобразовать схему таким образом, как показано на Рисунок 2.9.

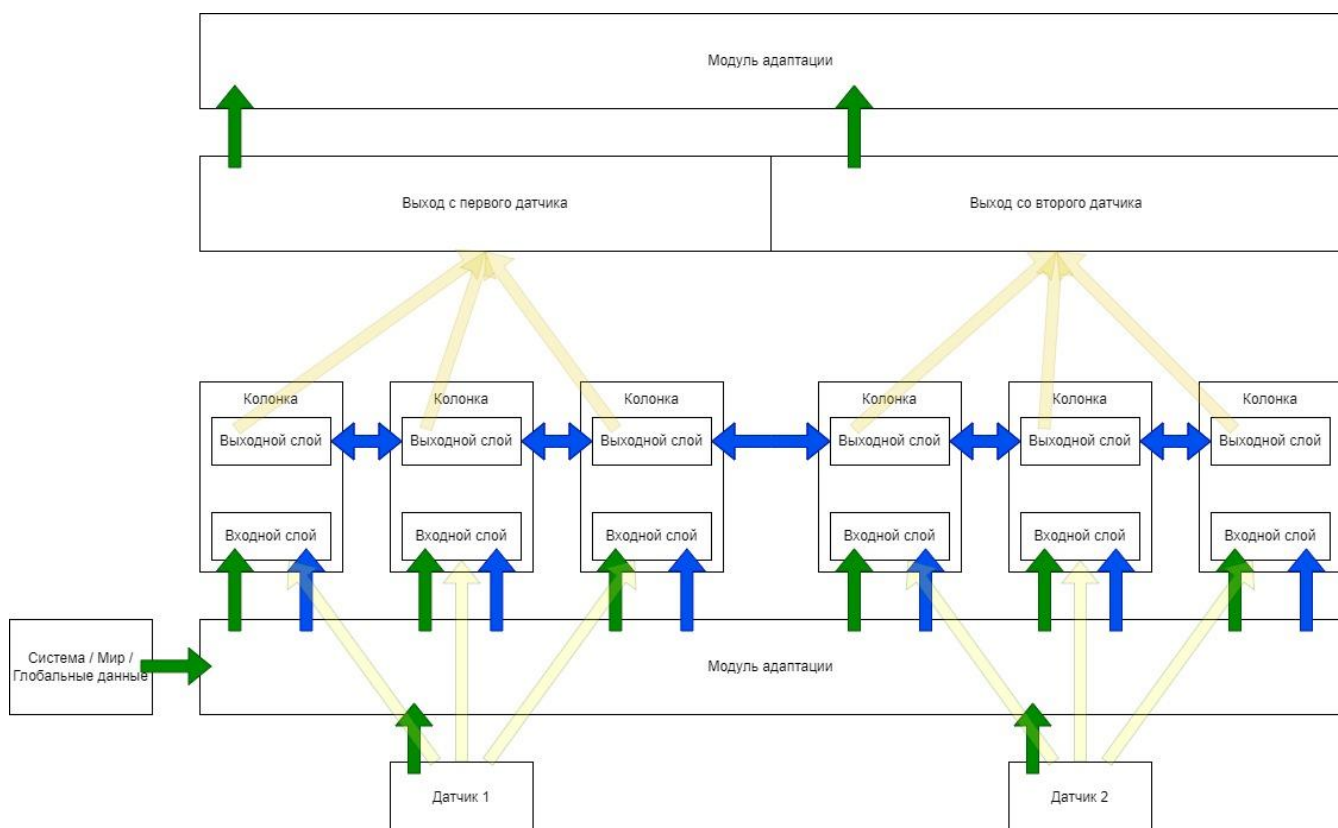


Рисунок 2.9 – Схема с несколькими колонками на один датчик

2.3 Модель перехода состояний нейронов

Нейрон может находиться в трех состояниях: неактивный, деполяризованный, активный. Состояния целесообразно хранить в отдельных структурах данных. Для деполяризованных клеток можно использовать бинарную матрицу π^{in} , заполняется матрица по формуле (2.1).

$$\pi_{ij}^{in} = \begin{cases} 1, & \text{если } \exists_a (L \cdot \dot{D}^{ijd,in} \geq \theta_b^{in}) > 0, \\ 0 & \text{иначе} \end{cases} \quad (2.1)$$

где i, j – условные координаты нейрона;

d – дендрит;

L – сигнал местоположения;

$\dot{D}^{ijd,in}$ – бинарная матрица существующих синапсов на d -ом дендрите ij -ого нейрона;

θ_b^{in} – порог активации дендрита.

То есть будут предсказаны те клетки, у которых есть хотя бы один дендрит, у которого есть достаточное количество синапсов с входным сигналом.

Переход нейрона в стадию активности происходит по формуле (2.2).

$$a_{ij}^{in} = \begin{cases} 1, \text{если } j \in W^{in} \text{ and } \pi_{ij}^{in} > 0 \\ 1, \text{если } j \in W^{in} \text{ and } \sum_i \pi_{ij}^{in} = 0 \\ \text{иначе } 0 \end{cases}, (2.2)$$

где a_{ij}^{in} – бинарная матрица с активными клетками;

j – номер мини-колонки;

W^{in} – множество столбцов, на которые поступил прямой сигнал.

То есть активными станут предсказанные ранее клетки, на которые поступил прямой сигнал. И активными станут те мини-колонки, на которые поступил прямой сигнал, но ни одна клетка в мини-столбце не была предсказана.

Активные клетки входного слоя посылают прямой сигнал в выходной слой для активации последнего. Активация клеток выходного слоя будет происходить по формулам ((2.3), ((2.4), ((2.5) и (2.6).

$$o_k^{out,t} = \sum_{i,j} I[f_{ijk} \geq \theta_c^{out}] a_{ij}^{in,t}, (2.3)$$

где k – индекс клетки выходного слоя;

$o_k^{out,t}$ – количество клеток, которые посылают прямой сигнал на k -ю клетку выходного слоя;

I – индикаторная функция;

f_{ijk} – значение постоянства синапса между ij -ой клеткой входного слоя и k -ой клеткой выходного слоя;

θ_c^{out} – порог активации синапса между слоями;

$a_{ij}^{in,t}$ – активные клетки на шаге t .

$$W^{out,t} = \{k | o_k^{out,t} \geq \theta_p^{out}\}, (2.4)$$

где $W^{out,t}$ – клетки выходного слоя с прямым сигналом на шаге t ;

θ_p^{out} – пороговое количество клеток, которые должны посылать сигнал в выходной слой на k -ю клетку, чтобы сигнал стал достаточно сильным.

$$a_i^{out,t} = \begin{cases} 1, \text{если } i \in W^{out,t} \text{ and } p_i^{out,t-1} \geq \xi_{t-1}^{out} \\ \text{иначе } 0 \end{cases}, (2.5)$$

где $a_i^{out,t}$ – активные клетки выходного слоя на шаге t ;

$p_i^{out,t-1}$ – количество активных базальных дендритов на предыдущем шаге;

ξ_{t-1}^{out} – пороговое количество базальных дендритов, необходимое для активации.

$$p_i^{out,t-1} = \sum_d I[\tilde{A}^{out} \cdot \tilde{D}^{id,out} \geq \theta_b^{out}], \quad (2.6)$$

где $\tilde{A}^{out} \cdot \tilde{D}^{id,out}$ – существующие синапсы между выходным слоем и дендритом;

θ_b^{out} – порог активации дендрита.

Другими словами, выходная клетка станет активной, если у нее достаточно прямых и базальных связей. Если количество клеток с боковой поддержкой меньше некоторого s , то $\xi_{t-1}^{out} = 0$.

2.4 Обучение

Для обучения выбирается паттерн выходного слоя, который будет укрепляться при поступлении новых сенсорных сигналов и сигналов местоположения. Будут образовываться новые синапсы, которые способствуют активации этого паттерна и уничтожаться те, которые дают ложные срабатывания.

Таким образом, обучение дендритных сегментов будет происходить по формуле (2.7).

$$\Delta D^{ijd,in} = p_{in}^+ \dot{D}^{ijd,in} \circ L^t - p_{in}^- \dot{D}^{ijd,in} \circ (1 - L^t), \quad (2.7)$$

где $\Delta D^{ijd,in}$ – коэффициент, на который изменится значение постоянства синапса;

p_{in}^+ – коэффициент для вознаграждения «верного» синапса;

$\dot{D}^{ijd,in}$ – бинарная матрица с синапсами дендрита;

L^t – сигнал местоположения в момент времени t ;

p_{in}^- – коэффициент для уменьшения «не верного» синапса.

Аналогичным образом происходит обучение синапсов между входным и выходным слоем и между клетками выходного слоя и дендритами выходного слоя. Единственное отличие – разные коэффициенты.

2.5 Объем хранения данных

На емкость сети из кортикальных колонок влияет четыре параметра:

1. Репрезентативное пространство сети
2. Количество мини-столбцов
3. Количество нейронов в выходном слое
4. Количество кортикальных колонок

Если учитывать, что диаметр кортикальных колонок варьируется от 300 до 600 мкм, а диаметр мини-колонки от 30 до 60 мкм, то можно предположить, что кортикальный столбец содержит от 150 до 250 мини-колонок.

Таким образом, сеть из 150 мини-колонок, 16 клеток на мини-колонку и одновременно 10 активных клетках может содержать порядка 10^{15} сенсорных представлений и 16^{10} представлений местоположения.

Если выходной нейрон соединяется со слишком большим количеством входных нейронов, он может быть ошибочно активирован шаблоном, по которому он не обучался. Таким образом, пропускная способность сети ограничена размером выходного слоя.

Например, колонка с 4096 клетками в выходном слое способна запомнить до 400 объектов, график приводится в статье [1] (Рисунок 2.10). По вертикале отмечена точность распознавания, а по горизонтали - количество объектов. Исследуя график, можно сделать вывод, что увеличение количества мини-колонок входного слоя положительно влияет на емкость одной кортикальной колонки.

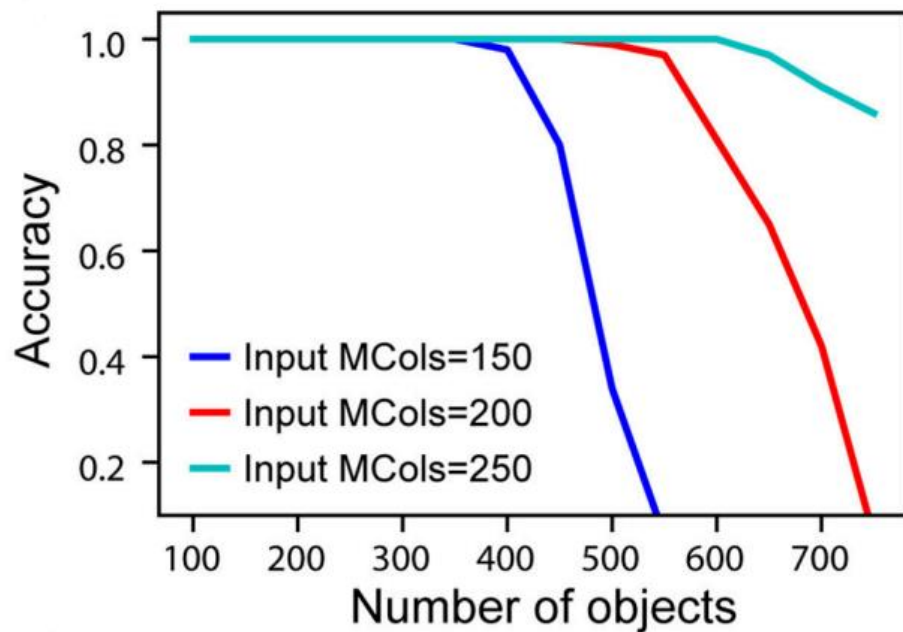


Рисунок 2.10 - Зависимость количества объектов, которые может запомнить колонка, от количества мини-колонок

Таким же образом и увеличение количества клеток выходного слоя позволяет увеличить емкость кортикальной колонки (Рисунок 2.11).

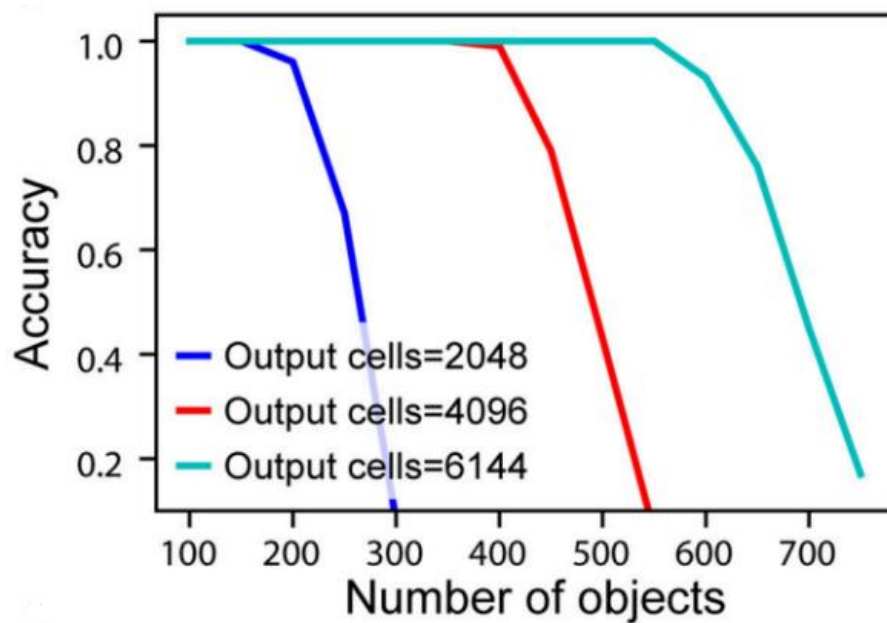


Рисунок 2.11 – Зависимость количества объектов, которые может запомнить колонка, от количества клеток в выходном слое

Увеличение количества кортикальных колонок тоже увеличивает емкость сети в целом.

Считается, что каждый столбец коры изучает модель «своего» мира, того, что он может ощущать. В одном столбце изучается структура многих объектов и поведение, которое можно применить к этим объектам. Благодаря локальным базальным и дальним корково-кортикальным связям колонки, воспринимающие один и тот же объект, могут разрешить неоднозначность.

2.6 Выявление аномалий

Для системы, выявляющей аномалии, можно выделить следующие требования:

1. Прогнозы необходимо делать онлайн; т. е. алгоритм должен идентифицировать состояние x_t как нормальное или аномальное перед получением последующего x_{t+1} .
2. Алгоритм должен обучаться непрерывно без необходимости сохранять весь поток.
3. Алгоритм должен работать без присмотра и в автоматическом режиме, т.е. без меток данных или ручной настройки параметров.
4. Алгоритмы должны адаптироваться к динамичной среде и дрейфу концепций, поскольку основная статистика потока данных часто нестационарна.
5. Алгоритмы должны обнаруживать аномалии как можно раньше.
6. Алгоритмы должны минимизировать ложноположительные и ложноотрицательные результаты.

Модель кортикальной памяти представляет собой теоретическую основу для последовательного обучения в коре. Реализации работают в режиме реального времени и хорошо работают для задач прогнозирования. Модель постоянно изучает и моделирует пространственно-временные характеристики своих входных данных, но не моделирует аномалии напрямую и не выдает готовую для использования оценку аномалий.

На Рисунок 2.12(a) показан обзор процесса детектирования аномалий. В каждый момент времени входные данные x_t подаются в стандартную сеть НТМ. Происходит процесс деполяризации и активации клеток. Затем вычисляется ошибки

прогнозирования S_t . После, используя вероятностную модель S_t , вычисляется L_t — вероятность того, что система находится в аномальном состоянии. Порог этой вероятности определяет, будет ли обнаружена аномалия.

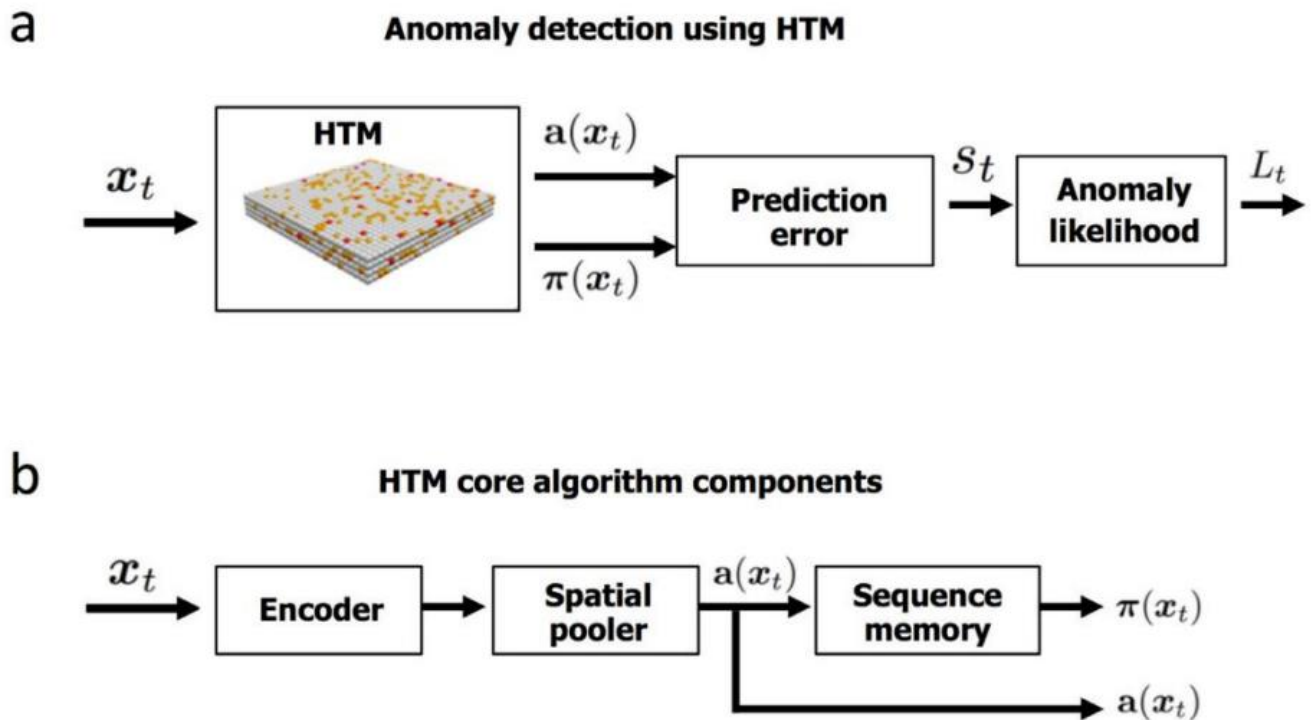


Рисунок 2.12 – Поиск аномалий с помощью иерархической темпоральной памяти. На Рисунок 2.12(б) показаны основные компоненты и представления алгоритма. Текущий входной сигнал x_t подается на модуль адаптации данных (encoder), а затем на процесс разреженного пространственного объединения. Результирующий вектор $a(x_t)$ представляет собой разреженный двоичный вектор, представляющий текущий входной сигнал. Сердцем системы является компонент памяти последовательности. Этот компонент моделирует временные закономерности в $a(x_t)$ и выводит прогноз в форме другого разреженного вектора $\pi(x_t)$. Таким образом, $\pi(x_t)$ является предсказанием для $a(x_{t+1})$ [16].

3 Реализация

3.1 Ядро системы

Ядро системы реализовано на функциональном языке Erlang, который позволяет работать с сопоставлениями и легковесными процессами.

3.1.1 Структура хранения данных

Для хранения данных слоев будем использовать словарь (ключ - значение), один слой – один словарь.

Для входного слоя ключом будет являться разряд входных бинарных данных, значение – мини-колонка, тоже словарь. В словаре мини-колонок ключом является guid клетки, а значение – словарь дендритов. Дендриты, в свою очередь, представляют из себя тоже словарь, где ключ – номер разряда латерального сигнала, а значение – синапс.

Синапс – структура из guid синапса, значения постоянства и веса.

Выходной слой – словарь, ключ – номер разряда, значение – пара, состоящая из guid клетки и словаря дендритных ветвей. Структура дендрита выходного слоя аналогична структуре дендрита входного слоя.

Дендритная ветвь выходного слоя имеет размер выходного слоя. Потому что латеральная поддержка в этом слое ссылается сама на себя.

В отдельных словарях FeedForward (зеленые стрелочки на Рисунок 2.6) и FeedBack (красные стрелочки на Рисунок 2.6) хранятся синапсы между клетками входного слоя и выходного. При этом, для удобства хранится номер разряда внешнего словаря. Структура хранения приводится в $\text{map} \lll \text{range}, \text{id}(\text{in}) \gg, \ll \text{range}, \text{id}(\text{out}) \gg, \text{permanenceValue} \gg, (3.1)$.

$\text{map} \lll \text{range}, \text{id}(\text{in}) \gg, \ll \text{range}, \text{id}(\text{out}) \gg, \text{permanenceValue} \gg, (3.1)$

Для отображения состояния предсказания клеток на текущей итерации будем использовать аналогичную по структуре схему данных. Но хранить в ней будем те данные, которые имеют отношение к предсказанным клеткам. А именно мини-колонки, в которых предсказанные клетки, сами предсказанные клетки и дендриты,

которые привели к активации. Дополнительно для каждой клетки будем хранить признак активного апикального дендрита (guid самого дендрита, если он есть).

Для отображения состояния активности клеток на текущей итерации будем использовать следующую структуру – словарь по мини-колонкам, где в значении хранится список guid активных клеток. При этом не нужно хранить дендриты, потому что они уже хранятся в словарь с предсказанными клетками. Следует заметить, что если клетка активная, но ее нет в словаре с предсказанными клетками, значит она была активирована в следствие того, что в мини-колонке не оказалось предсказанных клеток и был активирован весь столбец [19].

При этом все данные считаются глобальными в рамках процесса и хранятся в глобальной области Process Dictionary, чтобы любая функция могла получить данные в любой момент времени без передачи параметров в функции и увеличения стека вызовов.

3.1.2 Переход между состояниями

Точка входа в процесс активации – получение сигнала местоположения. Сигнал представляет из себя набор чисел. Размер сигнала местоположения соответствует размеру дендрита нейрона входного слоя. Происходит перебор всех клеток входного слоя с целью выявления дендритов, которые приводят к деполяризации клеток. При определении так же учитывается наличие апикального дендрита, так как он понижает порог активации дендрита (Рисунок 3.1).

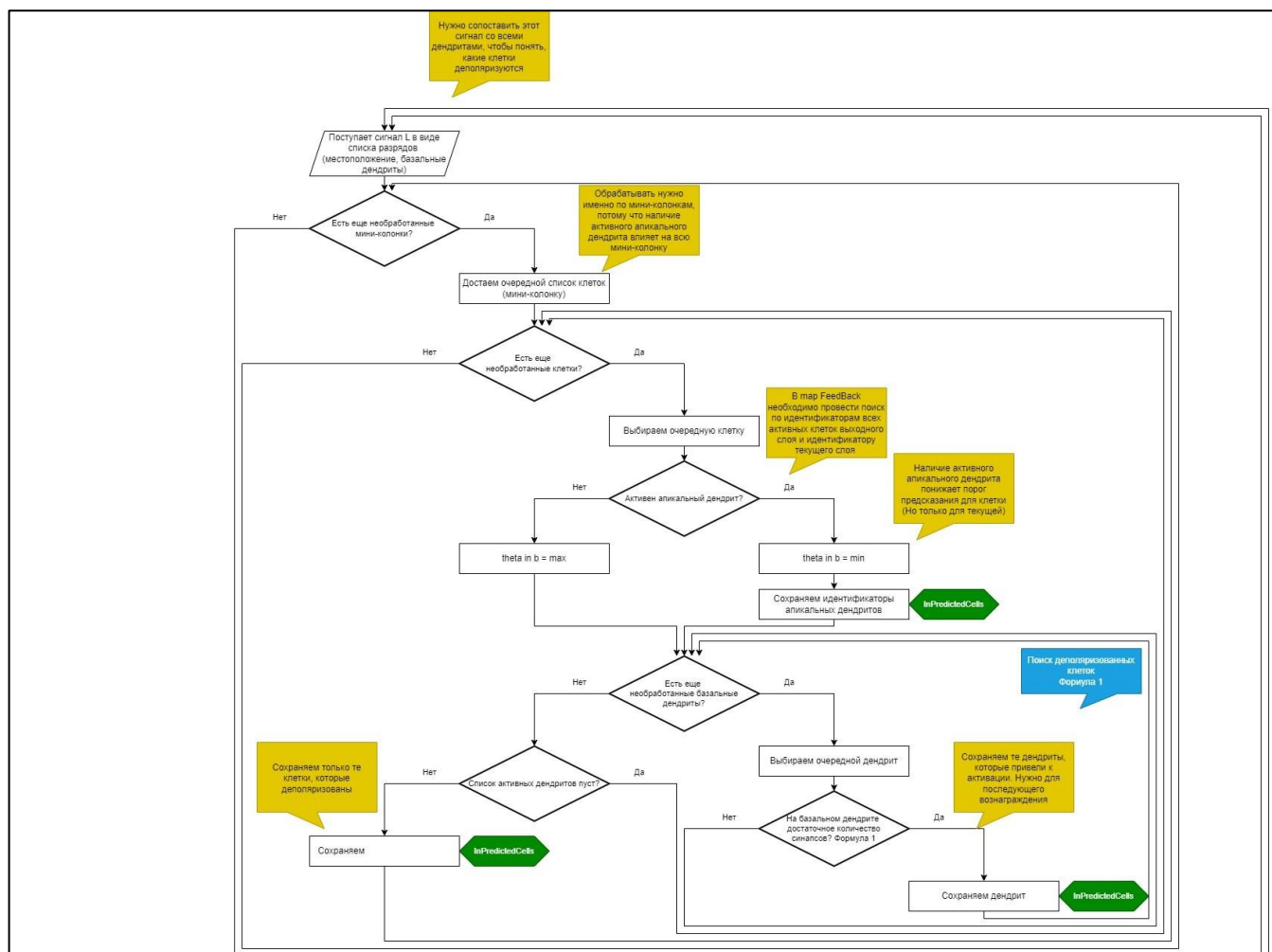


Рисунок 3.1 – Алгоритм предсказания. Обработка сигнала местоположения

Следующий этап – поступление прямого сигнала, который представляет из себя набор чисел, соответствующих номерам мини-колонок. Активируются те клетки, которые были ранее деполяризованы. Причем если в мини-колонке есть деполяризованные клетки с активным апикальным дендритом, то активируются только они. Если же в мини-колонке не было предсказанных клеток, а на нее поступил прямой сигнал, то станут активными все клетки мини-колонки (Рисунок 3.2).

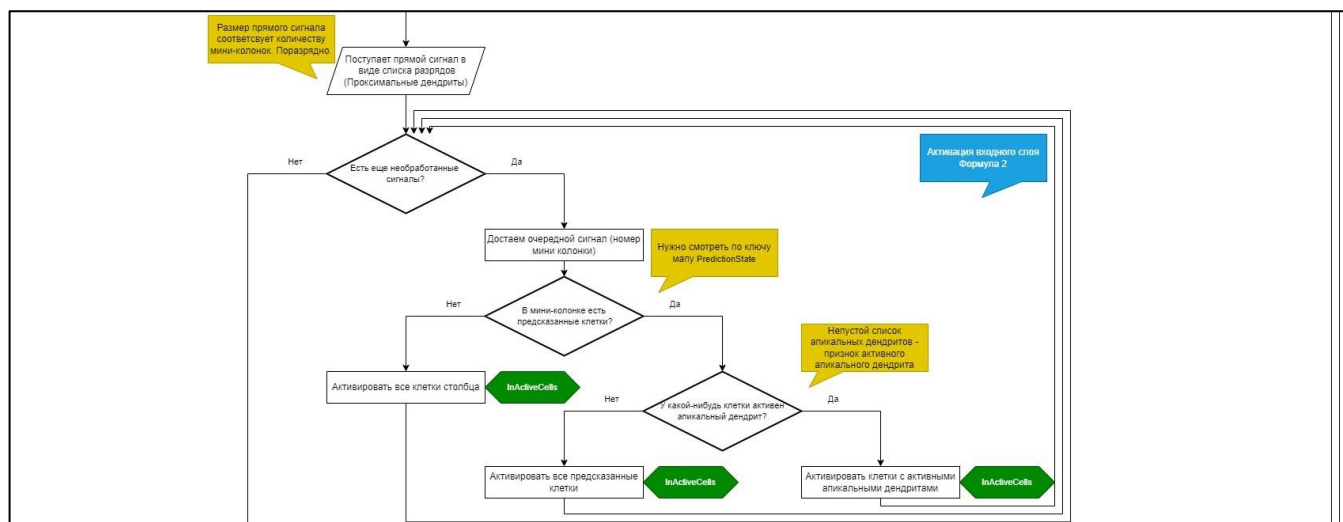


Рисунок 3.2 – Алгоритм предсказания. Обработка прямого сигнала

Активация клеток входного слоя приводит к активации клеток выходного слоя. При этом учитывается боковая поддержка от соседних кортикальных колонок и паттерн активации на предыдущем временном шаге. Но если из-за слабой поддержки не удастся активировать достаточное количество клеток в выходном слое, то боковая поддержка не учитывается (Рисунок 3.3).

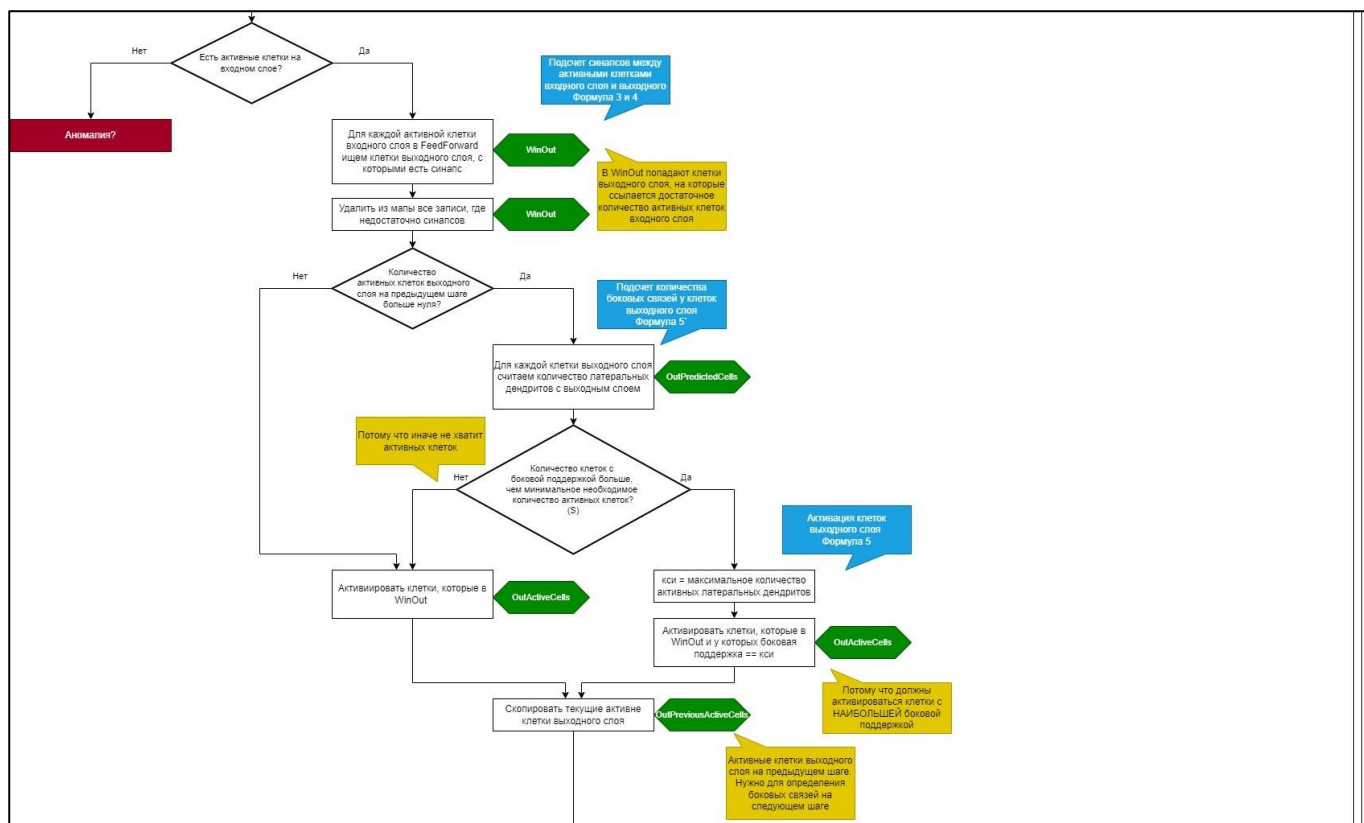


Рисунок 3.3 - Алгоритм предсказания. Активация выходного слоя

3.1.3 Обучение

Ядро работает в трех режимах:

1. Активное обучение
2. Пассивное обучение
3. Отсутствие обучения

Активное обучение подразумевает, что мы заранее знаем, о каком объекте идет речь, и устанавливаем соответствующий паттерн активации в выходной слой. Тем самым при обучении будут усиливаться связи, которые приводят к активации этого паттерна (Рисунок 3.4).

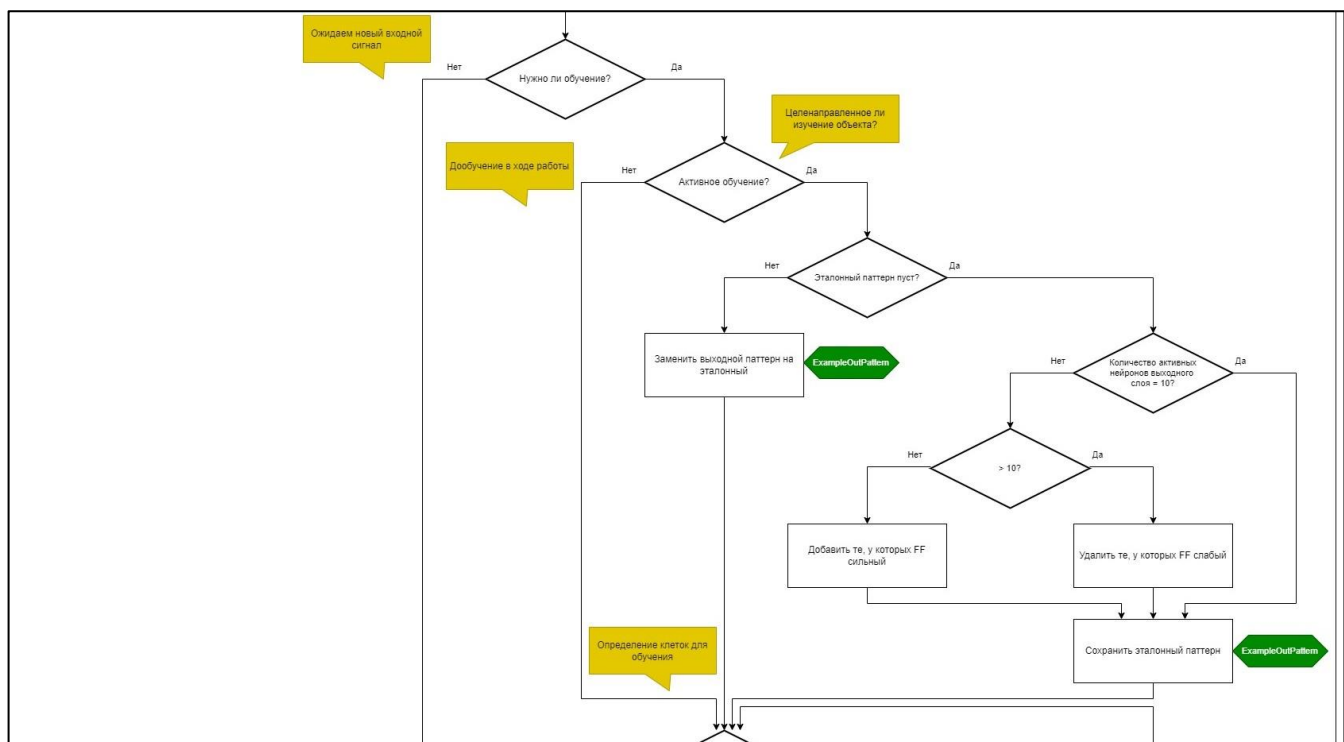


Рисунок 3.4 – Алгоритм обучения. Установка паттерна выходного слоя

При обучении входного слоя, вознаграждаются те дендриты, которые привели к деполяризации клеток, которые в последствии стали активными. Если дендрит привел к деполяризации, а клетка не стала активной, то этот дендрит «наказывается» (Рисунок 3.5).

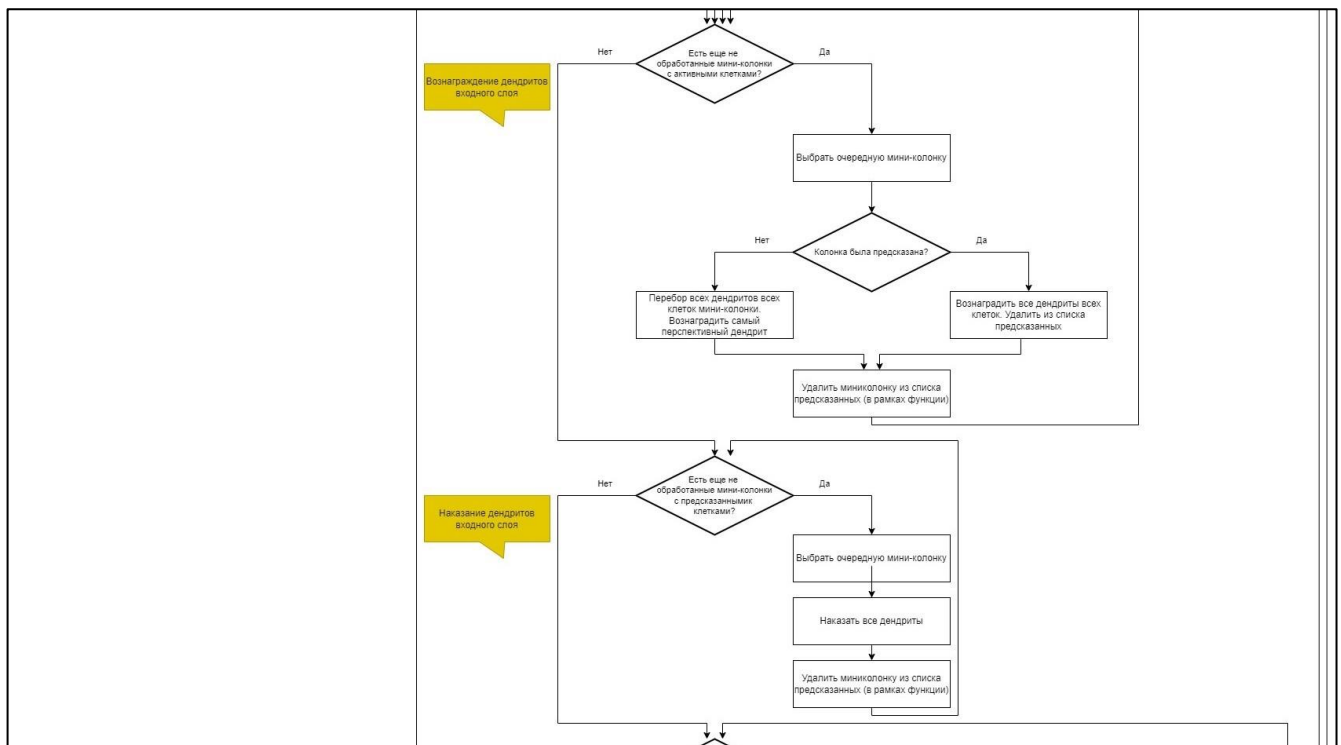


Рисунок 3.5 - Алгоритм обучения. Обновление синапсов дендритов

Похожим образом происходит обновление синапсов между слоями и синапсов боковой поддержки (Рисунок 3.6).

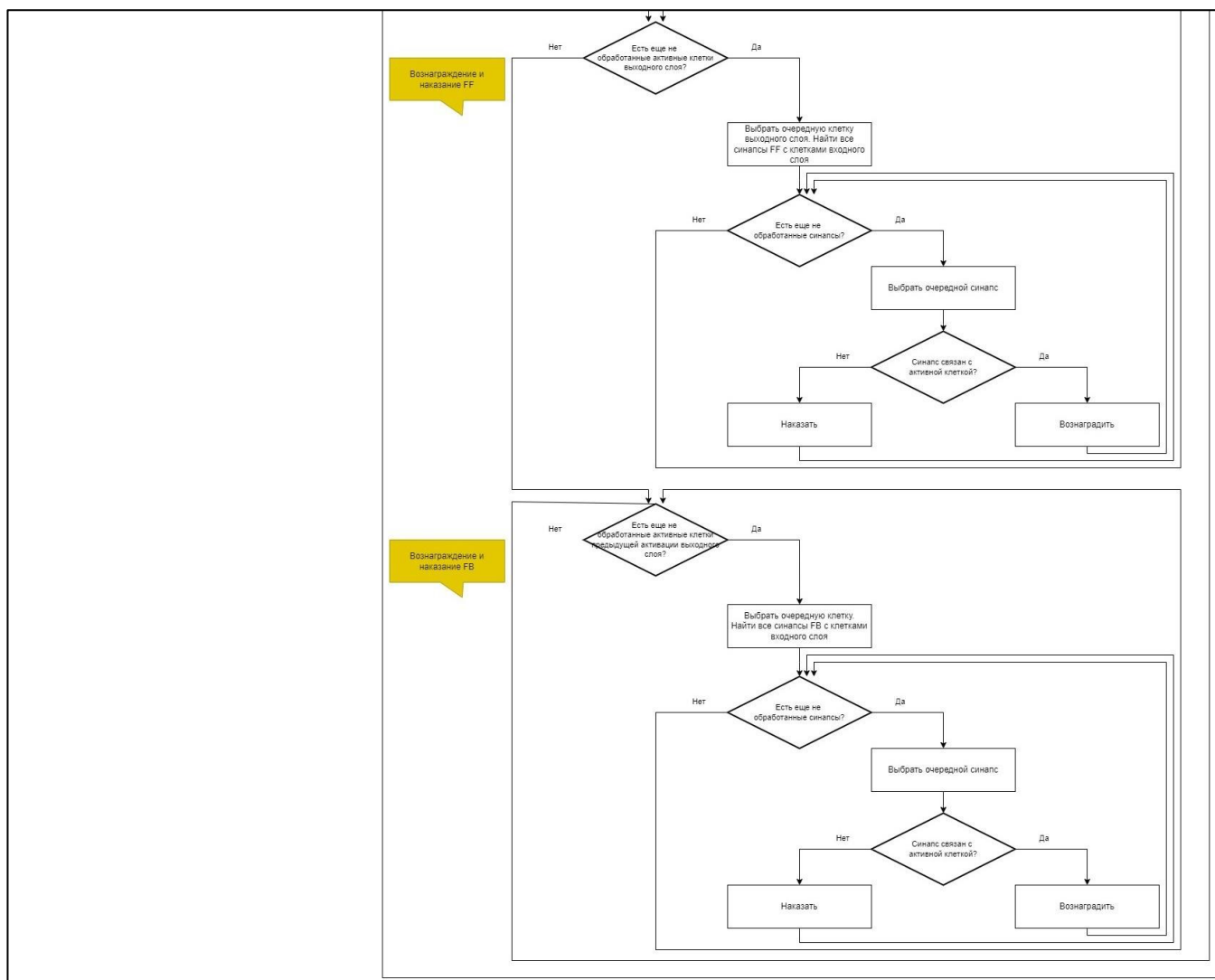


Рисунок 3.6 - Алгоритм обучения. Вознаграждение связей между слоями

3.2 Модуль визуализации

3.2.1 Связь с ядром системы

Для передачи информации между ядром системы, реализованном на Erlang, и модулем визуализации, реализованном на .NET WPF, был реализован механизм передачи сообщений. Передача сообщений происходит по схеме «клиент-сервер», где сервером является модуль визуализации, потому что предполагается, что можно анализировать данные с нескольких колонок. Для передачи используются TCP-сокеты. Отправка происходит построчно. Была разработана грамматика и синтаксический анализатор (Рисунок 3.7).



Рисунок 3.7 – Схема отправки структуры данных

Передача данных начинается с объявления типа передаваемой структуры (Рисунок 3.8), затем отправляется сами данные структуры, затем отправляется терминальный символ «END», обозначающий завершение передачи данных.

Есть возможность отправлять как базовые типы структур – словари и списки, так и уникальные, относящиеся к предметной области, – синапсы, дендриты (Рисунок 3.9). Уникальные структуры, по сути, состоят из простых базовых структур, но здесь важна последовательность, в которой будут отправлены данные.

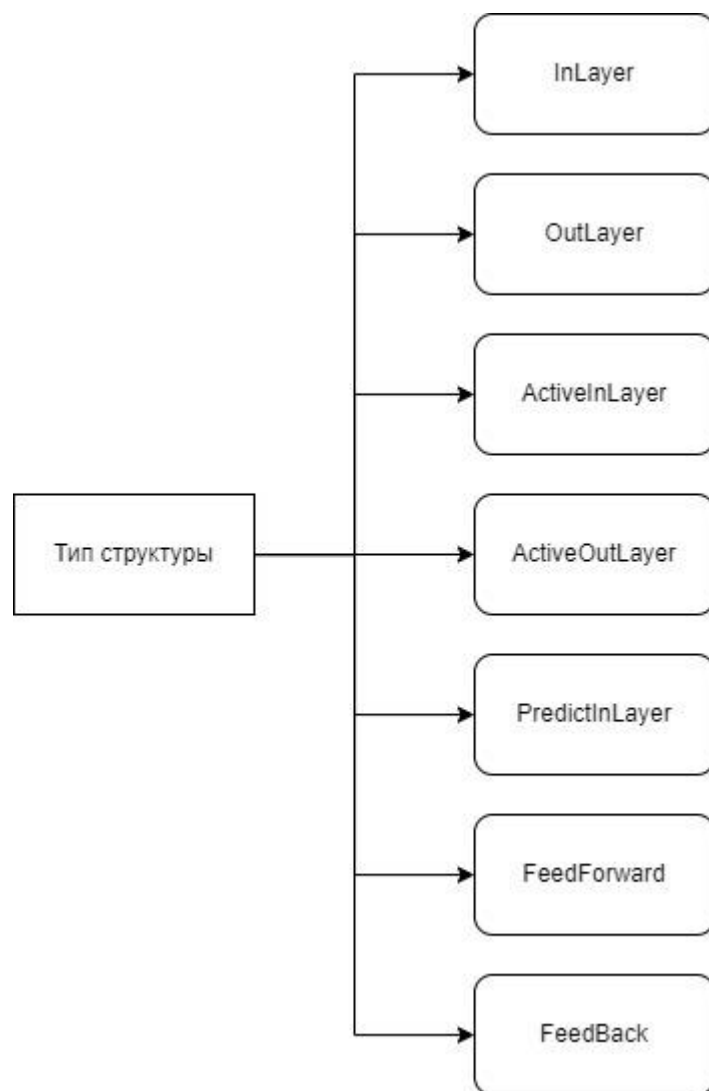


Рисунок 3.8 – Типы структур предметной области

В качестве отправляемой структуры (самих данных) могут выступать типы, представленные на Рисунок 3.9. Map и List – базовые типы, схемы представлены на Рисунок 3.10 и Рисунок 3.11 соответственно.

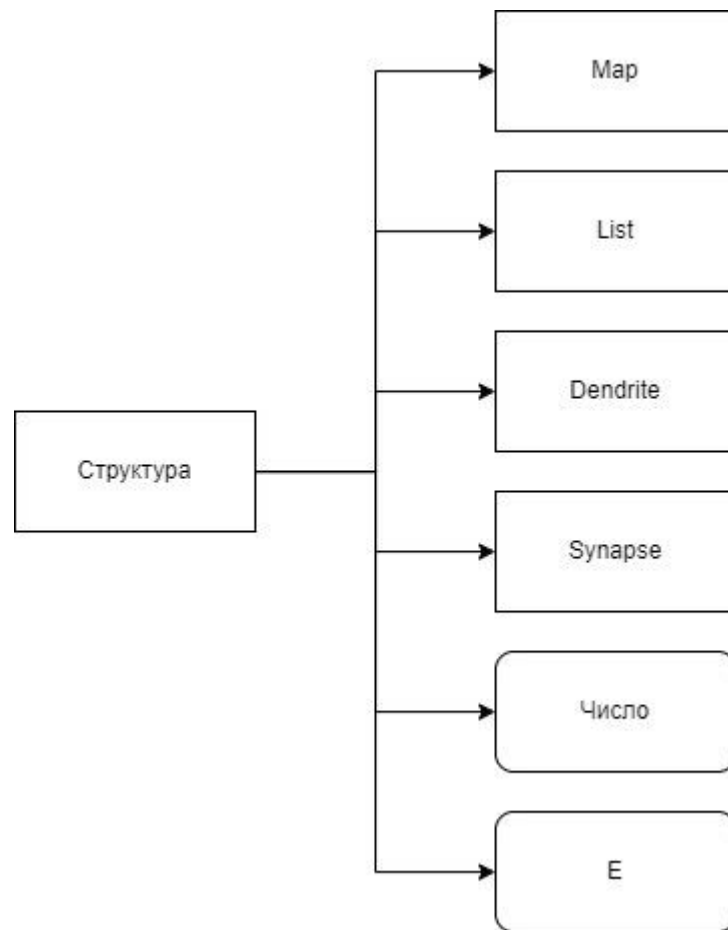


Рисунок 3.9 – Отправляемые типы данных



Рисунок 3.10 – Структура словаря

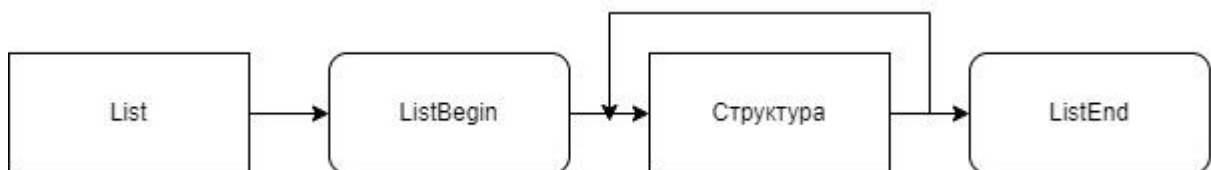


Рисунок 3.11 – Структура списка

Синапс – уникальная структура, состоит из трех чисел – guid синапса, значение постоянства и вес (Рисунок 3.12).

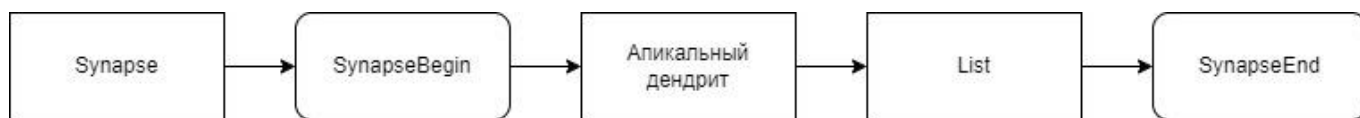


Рисунок 3.12 – Структура синапса

Апикальный дендрит – тоже уникальная структура, которая может принимать два значения. Либо число – guid клетки, к которой тянется дендрит, либо значение «false» - признак того, что дендрита нет (Рисунок 3.13).

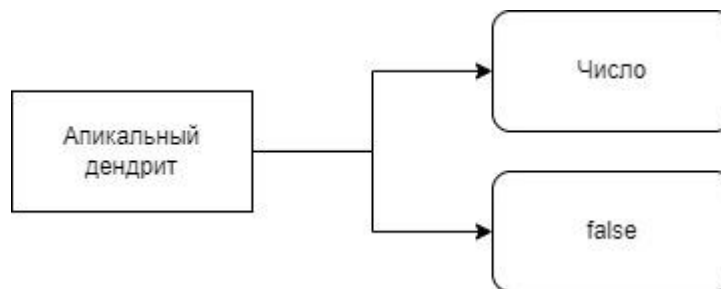


Рисунок 3.13 – Структура признака апикального дендрита

3.2.2 Интерфейс

В интерфейсе модуля визуализации (Рисунок 3.14) есть функционал, позволяющий работать с сигналами – редактирование и отправка прямого сигнала и сигнала местоположения (Рисунок 3.15).

Так же есть наглядное представление данных, благодаря которому можно анализировать процессы, протекающие во время работы системы. А именно:

1. Анализ состояния клеток входного слоя (Рисунок 3.16)
2. Анализ состояния дендритов клеток входного слоя (Рисунок 3.17)
 - 2.1.Наличие синапсов
 - 2.2.Наличие активных дендритов
3. Анализ связей между слоями (FeedForward, FeedBack) (Рисунок 3.18)
4. Анализ активности выходного слоя (Рисунок 3.19)



Рисунок 3.14 – Интерфейс разработанного программного обеспечения

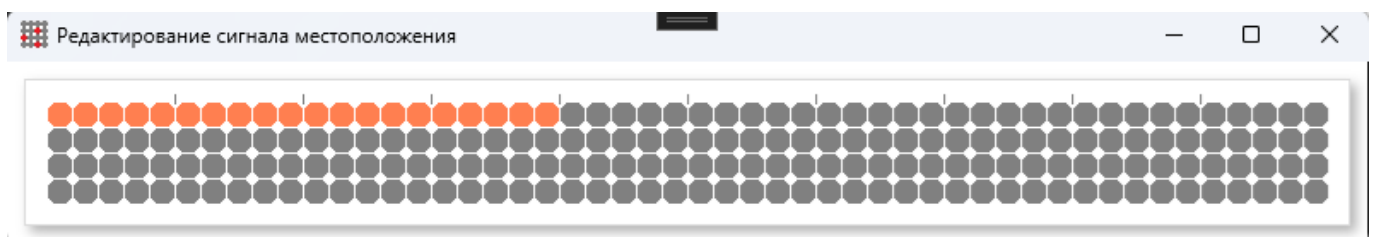


Рисунок 3.15 – Окно редактирования сигнала

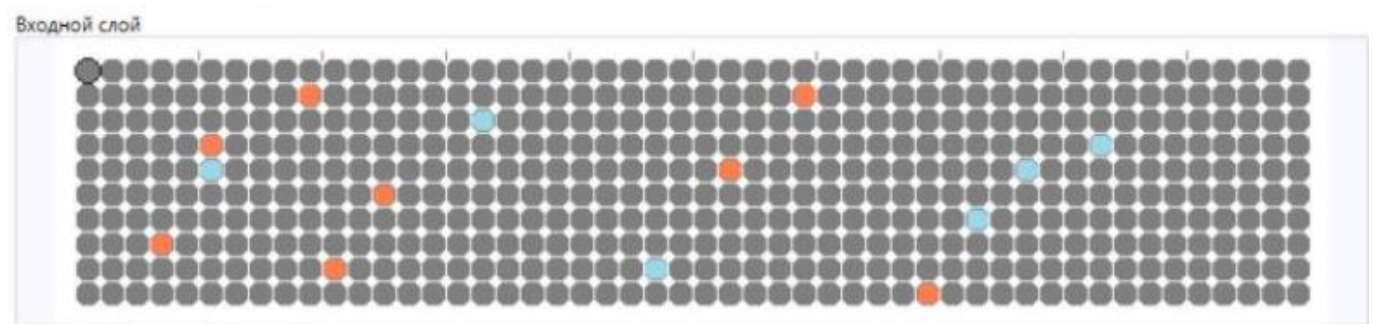


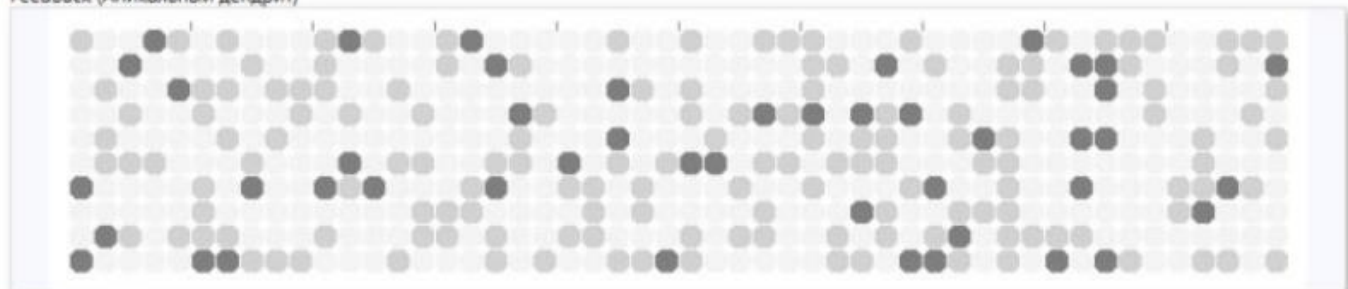
Рисунок 3.16 – Активность входного слоя

Базальные дендриты



Рисунок 3.17 – Дендриты одной клетки входного слоя

FeedBack (Апикальный дендрит)



FeedForward

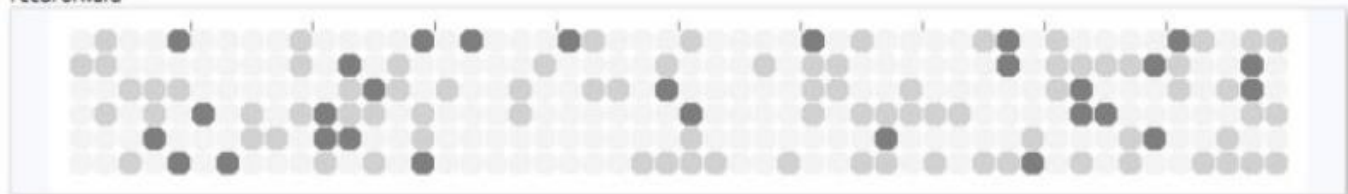


Рисунок 3.18 – Связи между слоями

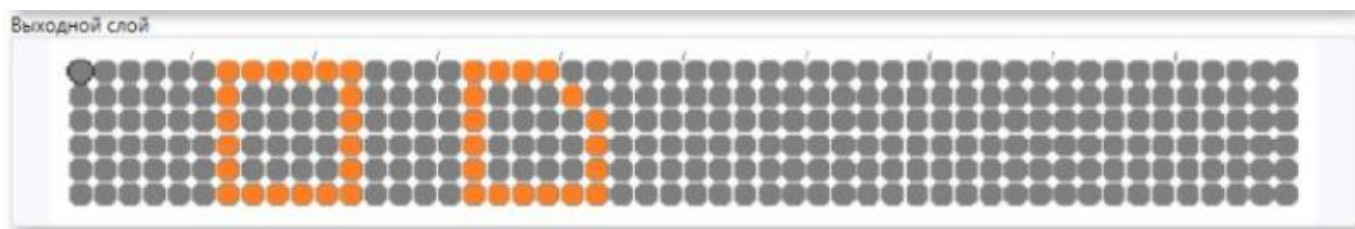


Рисунок 3.19 – Активность выходного слоя

Для реализации использовался паттерн проектирования MVVM, схема взаимодействия компонентов представлена на Рисунок 3.20.

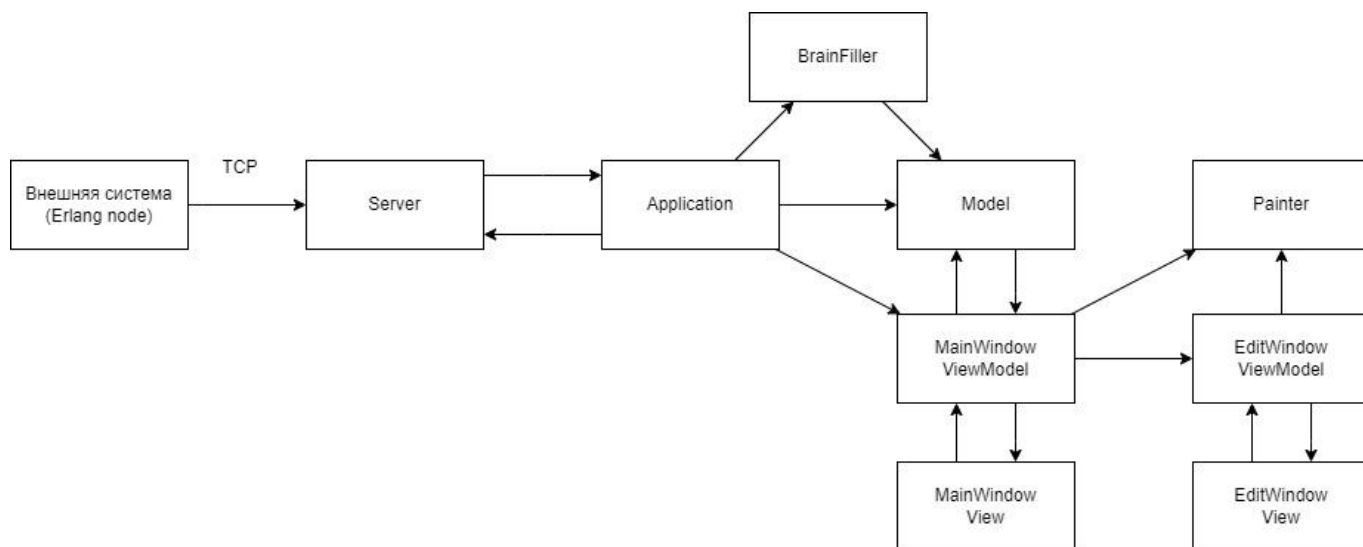


Рисунок 3.20 – Взаимодействие компонентов в модуле визуализации

Заключение

В ходе работы была изучена литература и научные статьи Джеффа Хоккинса и исследовательской группы Numenta. В соответствии с изложенной теорией было реализовано программное обеспечение, состоящее из двух модулей – ядро системы, делающее предсказание, и модуль визуализации для наглядности представления данных.

В целом, разработанная система является многообещающим инструментом для анализа больших данных с целью обнаружения аномалий. Система может быть использована в различных областях, таких как здравоохранение, финансы и производство, для улучшения процессов принятия решений и повышения эффективности.

В будущем данное исследование будет расширено в следующих направлениях:

1. Реализация модуля адаптации данных. Системы для преобразования данных с различных датчиков в разряженный бинарный вектор и разложения их на две составляющих – данные местоположения и сенсорные данные.
2. Изучение новых сенсорно-моторных алгоритмов и представления неокортексом данных местоположения.
3. Изучение назначения различных типов нейронов в неокортексе, таких как Grid Cells.
4. Разработка новых методов визуализации и интерпретации данных для улучшения понимания результатов анализа и выявления аномалий.
5. Интеграция системы с другими системами анализа данных для создания комплексных решений для анализа больших данных.

Список использованных источников

1. Hawkins, J. A Theory of How Columns in the Neocortex Enable Learning the Structure of the World / J. Hawkins, S. Ahmad, Y. Cui // Front. Neural Circuits. – 2017. – Vol. 11.
2. Дорогина, О. И. Нейрофизиология : учебное пособие / О. И. Дорогина ; М-во науки и высш. образования Рос. Федерации, Урал. федер. ун-т. — Екатеринбург : Изд-во Урал. ун-та, 2019. — 100 с.
3. Атлас «Нервная система человека. Строение и нарушения». Под редакцией В.М.Астапова и Ю.В. Микадзе. 4-е издание, перераб. и доп. — М.: ПЕР СЭ, 2004. — 80 с.
4. Мозг, познание, разум: введение в когнитивные нейронауки [Электронный ресурс] : в 2 ч. Ч. 1 / под ред. Б. Баарса, Н. Гейдж ; пер. с англ. под ред. проф. В. В. Шульговского. — Эл. изд. — Электрон. текстовые дан. (1 файл pdf : 552 с.). — М. : БИНОМ. Лаборатория знаний, 2014.
5. Эделмен, Дж. Разумный мозг: Пер. с англ. / Дж. Эделмен, В. Маунткасл, Перевод Алексеенко Н. Ю.; Под ред. и с предисл. Е. Н. Соколова. – М.: Мир, 1981. – 135 с. с ил.
6. Козлов, В. И. Анатомия нервной системы : учебное пособие для студентов / В. И. Козлов, Т. А. Цехмистренко. — 3-е изд., электрон. — М. : Лаборатория знаний, 2022. — 216 с.
7. Hawkins, J. Why neurons have thousands of synapses, a theory of sequence memory in neocortex / J. Hawkins, S. Ahmad // Frontiers in neural circuits. – 2016. – Vol. 10.
8. Потапов, Д. П. Проектирование биоподобной модели память-предсказание по Хоккинсу / Потапов Д.П., Целебровский О.Б., Старолетов С.М. // Современные цифровые технологии : материалы II Всероссийской научнопрактической конференции (01 июня 2023 г.) / под общ. ред. А.А.Беушев, А.С. Авдеев, Е.Г. Боровцов, А.Г. Зрюмова ; АлтГТУ им. И. И. Ползунова. – Барнаул : АлтГТУ, 2023. – 243 - 247 с.

9. НТМ. GitHub / Д. П. Потапов, О. Б. Целебровский [Электронный ресурс]. – URL: <https://github.com/sablist99/НТМ> (Дата обращения 23.05.2023)
10. Гусельников, В. И. Электрофизиология головного мозга. — М.: Высшая школа, 1976 – 423 с. с ил. и табл.
11. Antic, S. D. The decade of the dendritic NMDA spike / S.D. Antic, W.L. Zhou, A.R. Moore, S.M. Short, K.D. Ikonomu // J Neurosci Res. – 2010. – 2991 – 3001 с.
12. Хокинс, Дж. 1000 мозгов. Новая теория интеллекта / Джефф Хокинс, [пер. с англ. С. Черникова]. – СПб.; Портал, 2024. – 368с.
13. Курпатов, А. Чертоги разума. Убей в себе идиота! / Андрей Курпатов. – СПб: ООО «Дом Печати издательства Книготорговли «Капитал», 2020. – 416 с., с ил.
14. Мозг / Д. Хьюбел, Ч. Стивенс, Э. Кэндел [и др.]; Пер. с англ./ Перевод Алексеенко Н. Ю.; Под ред. и с предисл. П. В. Симонова. – М.: Мир, 1984. – 280 с. с ил.
15. Старолетов, С. М. Обзор современного состояния кортикальных алгоритмов и их применение для анализа сигналов в реальном времени / С. М. Старолетов // Системный администратор. – 2022. – № 11(240). – С. 82-87. – EDN INFWEA.
16. Unsupervised real-time anomaly detection for streaming data / S. Ahmada, A. Lavina, S. Purdya, Z. Agha // Neurocomputing. – 2017. – №267. – 134 – 147с.
17. Как спят дельфины — исследование сна дельфинов / SiriusSib [Электронный ресурс]. – URL: <https://siriusib.ru/kak-spyat-delfiny-issledovanie-sna-delfinov/> (Дата обращения 16.06.2024)
18. Библиотека изображений Freepik [Электронный ресурс]. – URL: freepik.com (Дата обращения 23.05.2023)
19. Потапов, Д. П. Проектирование многоуровневой распределенной системы анализа биоподобными сенсорно-моторными алгоритмами / Д. П. Потапов, С. М. Старолетов // Современные цифровые технологии : материалы III


Всероссийской научнопрактической конференции (03 июня 2024 г.) / АлтГТУ им. И. И. Ползунова. – Барнаул : АлтГТУ, 2024. – Неопубл. материалы.

20. ThousandBrains. GitHub / Д. П. Потапов [Электронный ресурс]. – URL: <https://github.com/sablist99/ThousandBrains> (Дата обращения 19.06.2024)

Приложение А

Задание на выполнение ВКР

Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Алтайский государственный технический университет им. И.И. Ползунова»

УТВЕРЖДАЮ
Заведующий кафедрой ПМ

подпись Боровцов Е. Г.
Ф.И.О

ЗАДАНИЕ

на выполнение магистерской диссертации
бакалаврской работы, дипломной работы (дипломного проекта), магистерской диссертации

по направлению подготовки (специальности)

Программная инженерия

студенту группы 8ПИ-21 Потапову Даниилу Петровичу
фамилия, имя, отчество

Тема: Разработка многоуровневой распределенной системы анализа больших данных биоподобными сенсорно-моторными алгоритмами с целью определения аномалий

Утверждено приказом ректора от 20.11.23 № Л-3332

Срок выполнения работы 21.06.24

Задание принял к исполнению 
подпись Д. П. Потапов
инициалы, фамилия

БАРНАУЛ 2024

1 Исходные данные

Задание на выполнение, техническая документация на языки программирования Erlang, .NET, научные статьи исследовательской группы Numenta, ресурсы internet

2 Содержание разделов ВКР и календарный график её выполнения

Наименование разделов работы и их содержание	Трудоемкость, %	Срок выполнения	Консультант (фамилия, инициалы, подпись)
1 Предметная область	30		Старолетов С. М.
2 Проектирование	30		Старолетов С. М.
3 Реализация	40		Старолетов С. М.

3 Научно-библиографический поиск

3.1 По научно-технической литературе просмотреть РЖ «Кибернетика», «Программное обеспечение» и научно-технические журналы «Программирование», «Программная инженерия», «Информационные технологии» за последние 10 лет.

3.2 По нормативной литературе просмотреть указатели государственных и отраслевых стандартов за последний год.

3.3 Патентный поиск провести за 4 года по странам Россия

Руководитель


подпись

С. М. Старолетов

инициалы, фамилия

Приложение Б

Исходный код системы

Ядро системы. Модуль ActivateInCells.

```
%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 19. май 2024 15:56
%%%-----
-module('ActivateInCells').
-author("Potap").

%% API
-export([getActiveCells/1]).

% Получение списка Guid всех клеток из карты (из мини-колонки) через итератор
% Iterator - итератор для перебора клеток
% ActiveCells - out переменная, список Guid активных клеток
getCellsFromMapByIterator(Iterator, ActiveCells) ->
case maps:next(Iterator) of
  none -> ActiveCells;
  {CellGuid, _Value, NewIterator} ->
    getCellsFromMapByIterator(NewIterator, lists:append(ActiveCells, [CellGuid]))
end.

% Функция возвращает список CellsGuid с активным апикальным дендритом в мини-колонке
getCellWithActiveApicalDendriteByRangesHelper(PredictedCellsInMiniColumnIterator, ActiveCells) ->
case maps:next(PredictedCellsInMiniColumnIterator) of
  none -> ActiveCells;
  {_CellGuid, {noActiveApicalDendrite, _}, NewIterator} ->
    getCellWithActiveApicalDendriteByRangesHelper(NewIterator, ActiveCells);
  {CellGuid, {_OutCellRange, _}, NewIterator} ->
    getCellWithActiveApicalDendriteByRangesHelper(NewIterator, lists:append(ActiveCells,
[CellGuid]))
end.

getCellWithActiveApicalDendriteByRanges(PredictedCellsInMiniColumn, ActiveCells) ->
case PredictedCellsInMiniColumn of
  {ok, PredictedCells} ->
    getCellWithActiveApicalDendriteByRangesHelper(maps:iterator(PredictedCells), ActiveCells);
  _ -> ActiveCells
end.

% Получение списка активных клеток в мини-колонке
% RangeOfColumn - номер мини-колонки (разряд), которая выстрелила
```

```

% ActiveCells - out переменная, содержит информации об активных клетках. Map<разряд колонки,
[Guid активных клеток]>
% PredictedCells - Информация о деполяризованных клетках. Берется из глобальных данных
getActiveCellsInMiniColumn(RangeOfColumn, ActiveCells, PredictedCells) ->
% Колонка была ранее предсказана?
case 'CommonFunctions':hasMiniColumnInPredict(RangeOfColumn) of
% Если нет - активируем все клетки
false -> maps:put(RangeOfColumn,
getCellsFromMapByIterator(
maps:iterator( % Передаем в функцию итератор по мини-колонке
maps:get(RangeOfColumn, % Получаем мини-колонку по разряду
'GlobalDataService':getInLayer() % Получаем исходную структуру данных
)), []), ActiveCells);
% Если да - проверяем, есть ли активный апикальный дендрит у предсказанных клеток
true ->
case getCellWithActiveApicalDendriteByRanges(maps:find(RangeOfColumn, PredictedCells), []) of
% Если нет - активируем все предсказанные клетки
[] -> maps:put(RangeOfColumn, getCellsFromMapByIterator(
maps:iterator( % Передаем в функцию итератор по мини-колонке
maps:get(RangeOfColumn, PredictedCells) % Получаем мини-колонку по разряду
), []), ActiveCells);
% Если да - активируем клетки с активным апикальным дендритом
Value -> maps:put(RangeOfColumn, Value, ActiveCells)
end
end.

```

```

% Перебор всех выстреливших мини-колонок
% Первый аргумент - входной сигнал (список разрядов)
% ActiveCells - out переменная, содержит информации об активных клетках. Map<разряд колонки,
[Guid активных клеток]>
% PredictedCells - Информация о деполяризованных клетках. Берется из глобальных данных
getActiveCellsHelper([], ActiveCells, _PredictedCells) ->
% Когда обработан весь FeedForward
ActiveCells;
getActiveCellsHelper([RangeOfColumnWithFeedForward | TFeedForward], ActiveCells, PredictedCells)
->
% Обрабатываем очередной сигнал из FeedForward
getActiveCellsHelper(TFeedForward,
getActiveCellsInMiniColumn(RangeOfColumnWithFeedForward, ActiveCells, PredictedCells),
PredictedCells).

```

```

% Функция возвращает активные клетки. Данные упакованы в иерархию, аналогичную структуре
хранения данных
% FeedForward - входной сигнал (список разрядов)
getActiveCells(FeedForward) ->
getActiveCellsHelper(FeedForward, #{} , 'GlobalDataService':getInPredictedCells()).

```

Ядро системы. Модуль ActivateOutCells.

```

%% %%-----

```



```

%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 20. май 2024 0:43
%%%-----
-module('ActivateOutCells').
-author("Potap").

%% API
-export([getOutActiveCells/0]).

-include("Model/BrainSettings.hrl").

% Описание алгоритма
% Берем клетку выходного слоя. И для каждой такой клетки перебираем активные клетки
% входного слоя.
% По двум guid получаем синапс из FeedForward.
% Если синапс есть, то сохраняем его.
% Когда кончились активные клетки, смотрим на количество сохраненных синапсов.
% Если их достаточное количество, то сохраняем range колонки и список синапсов, приведших к
% активации

% В этой функции происходит формирование списка существующих синапсов между выходной
% клеткой и активными клетками ОДНОЙ МИНИ-КОЛОНКИ входного слоя.
% ActiveCellGuid - текущая активная клетка, для которой ищем синапс с выходной клеткой
% OutCellGuid - текущая выходная клетка, для которой ищем синапсы с активными клеткой
% Synapses - out параметр, результирующая мапа, содержащая существующие синапсы
getSynapsesListBetweenInActiveCellsFromOneMiniColumnAndOutCell([], _OutCellRange, Synapses) -
>
    Synapses;
getSynapsesListBetweenInActiveCellsFromOneMiniColumnAndOutCell([ActiveCellGuid | ActiveCells],
OutCellRange, Synapses) ->
    case 'CommonFunctions':existSynapseBetweenLayersByFGTR(ActiveCellGuid, OutCellRange,
'GlobalDataService':getFeedForward()) of
        {true, _Value} ->
            getSynapsesListBetweenInActiveCellsFromOneMiniColumnAndOutCell(ActiveCells, OutCellRange,
lists:append(Synapses, [{ActiveCellGuid, OutCellRange}]));
        false ->
            getSynapsesListBetweenInActiveCellsFromOneMiniColumnAndOutCell(ActiveCells, OutCellRange,
Synapses)
    end.

% В этой функции происходит формирование списка существующих синапсов между выходной
% клеткой и активными клетками входного слоя
% Вместе с этим проверяется, а достаточно ли этих синапсов для активации
getSynapsesListBetweenInActiveCellsAndOutCell(ActiveCellsIterator, OutCellRange, Synapses) ->
    case maps:next(ActiveCellsIterator) of

```

```

% Проверяем - а достаточно ли синапсов у текущей выходной клетки
none ->
case 'CommonFunctions':getListSize(Synapses) >= ?THETA_OUT_P of
    true -> Synapses;
    false -> []
end;
{_Range, ActiveCellsList, NewCellIterator} ->
getSynapsesListBetweenInActiveCellsAndOutCell(NewCellIterator, OutCellRange,
getSynapsesListBetweenInActiveCellsFromOneMiniColumnAndOutCell(ActiveCellsList,
OutCellRange, Synapses))
end.

% В этой функции формируется список выигрышных клеток выходного слоя (тех клеток, у
которых достаточно синапсов с активными клетками входного слоя)
getOutWinCells(OutCellsIterator, OutWin) ->
case maps:next(OutCellsIterator) of
    none -> OutWin;
    {Range, {_OutCellGuid, _}, NewCellIterator} ->
        case
getSynapsesListBetweenInActiveCellsAndOutCell(maps:iterator('GlobalDataService':getInActiveCells())
, Range, []) of
    % Если связь не прочная или ее вообще нет
    [] -> getOutWinCells(NewCellIterator, OutWin);
    % Если есть достаточно синапсов, то объявляем клетку выигрышной
    Synapses -> getOutWinCells(NewCellIterator, maps:put(Range, Synapses, OutWin))
end
end.

% Получение выигрышных клеток выходного слоя
getOutWinCells() ->
getOutWinCells(maps:iterator('GlobalDataService':getOutLayer()), #{}).

% Выборка активных клеток с учетом латерального сигнала
getActiveCellsWithKsi(WinOutIterator, Ksi, ActiveList) ->
case maps:next(WinOutIterator) of
    none -> ActiveList;
    {Range, List, NewWinOutIterator} ->
        case 'CommonFunctions':getListSize(List) of
            Size -> case Size >= Ksi of
                true -> getActiveCellsWithKsi(NewWinOutIterator, Ksi, lists:append(ActiveList, [Range]));
                false -> getActiveCellsWithKsi(NewWinOutIterator, Ksi, ActiveList)
            end
        end
    end
end.

getActiveCellsWithKsi(WinOut, Ksi) ->

```

```
getActiveCellsWithKsi(maps:iterator(WinOut), Ksi, []).
```

```
% Функция возвращает максимальное количество активных базальных дендритов
% TODO Узнать, как записать значение вызова в функцию и использовать его дальше. При этом
не потеряв функциональный стиль
getKsi(PredictedCellsIterator, Ksi) ->
case maps:next(PredictedCellsIterator) of
  none -> Ksi;
  {_Range, List, NewCellIterator} ->
    case 'CommonFunctions':getListSize(List) of
      Size -> case Size > Ksi of
        true -> getKsi(NewCellIterator, Size);
        false -> getKsi(NewCellIterator, Ksi)
      end
    end
  end
end.

getKsi(PredictedCells) ->
getKsi(maps:iterator(PredictedCells), 0).
```

```
% Получение активных клеток выходного слоя.
% С учетом прямого и латерального сигнала
getOutActiveCells([]) ->
  getActiveCellsWithKsi(getOutWinCells(), 0);
getOutActiveCells(undefined) ->
  getActiveCellsWithKsi(getOutWinCells(), 0);
getOutActiveCells(OutPreviousActivation) ->
  case 'PredictCells':getPredictedCellsInOutputLayer(OutPreviousActivation,
maps:iterator('GlobalDataService':getOutLayer()), #{ }) of
    PredictedCells ->
      % Количество клеток с боковой поддержкой больше, чем минимальное количество активных
клеток
      case maps:size(PredictedCells) >= ?S of
        true -> getActiveCellsWithKsi(getOutWinCells(), getKsi(PredictedCells));
        false -> getActiveCellsWithKsi(getOutWinCells(), 0)
      end
    end
  end.

getOutActiveCells() -> getOutActiveCells('GlobalDataService':getOutPreviousActivation()).
```

Ядро системы. Модуль BrainInit.

```
%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
```

```

%%% Created : 16. май 2024 0:59
%%%-----
-module('BrainInit').
-author("Potap").

-include("Model/BrainSettings.hrl").
-include("Model/Model.hrl").

%% API
-export([initializeGlobalData/0, getSynapse/0, getInLayer/0]).

% Определение наличия связи в зависимости от поргового значения
getPermanenceWeight(Value) when Value >= ?PERMANENCE_WEIGHT_BORDER ->
    true;
getPermanenceWeight(_Value) ->
    false.

% Функция создания синапса
getSynapseHelper(Value) ->
    case Value < ?POTENTIAL_SYNAPSE_BORDER of
    true -> none;
    false ->
        #synapse{
            guid = 'MyMath':getId(),
            permanenceValue = Value,
            permanenceWeight = getPermanenceWeight(Value)}
        end.

getSynapse() ->
    getSynapseHelper('MyMath':getPoissonValue()).

% Функция добавления синапса в дендрит. Добавляет в дендрит только в том случае, если
% вернулся "синапс" из которого потенциально может вырасти настоящий синапс Определяется
% порговым значением.
addSynapseInDendrite(_Key, Synapse, Dendrite) when Synapse == none ->
    Dendrite;
addSynapseInDendrite(Key, Synapse, Dendrite) ->
    maps:put(Key, Synapse, Dendrite).

% Функция создания дендрита клетки любого слоя
% CurrentSynapseNumber - счетчик, количество синапсов на текущем дендрите.
% Dendrite - out переменная, в которой копится результат (Map)
% SynapseCount - количество синапсов на дендрите (меняется в зависимости от типа слоя)
getDendriteHelper(CurrentSynapseNumber, Dendrite, SynapseCount) when CurrentSynapseNumber ==
SynapseCount ->
    Dendrite;
getDendriteHelper(CurrentSynapseNumber, Dendrite, SynapseCount) ->

```

```
getDendriteHelper(CurrentSynapseNumber + 1, addSynapseInDendrite(CurrentSynapseNumber,
getSynapse(), Dendrite), SynapseCount).
```

```
getDendrite(SynapseCount) -> getDendriteHelper(0, #{} , SynapseCount).
```

```
% Функция создания клетки для любого слоя
```

```
% CurrentDendrite - счетчик, количество дендритов на текущей клетке.
```

```
% Cell - out переменная, в которой копится результат (Map)
```

```
% DendriteCount
```

```
getCellHelper(CurrentDendrite, Cell, _SynapseCount) when CurrentDendrite == ?D ->
Cell;
```

```
getCellHelper(CurrentDendrite, Cell, SynapseCount) ->
```

```
getCellHelper(CurrentDendrite + 1, maps:put('MyMath':getId(), getDendrite(SynapseCount), Cell),
SynapseCount).
```

```
getCell(SynapseCount) -> getCellHelper(0, #{} , SynapseCount).
```

```
% Функция создания мини-колонки входного слоя
```

```
getInMiniColumnHelper(CurrentCell, MiniColumn) when CurrentCell == ?M ->
```

```
MiniColumn;
```

```
getInMiniColumnHelper(CurrentCell, MiniColumn) ->
```

```
getInMiniColumnHelper(CurrentCell + 1, maps:put('MyMath':getId(), getCell(?N_EXT),
MiniColumn)).
```

```
getInMiniColumn() -> getInMiniColumnHelper(0, #{}).
```

```
% Функция создания входного слоя одной колонки
```

```
getInLayerHelper(CurrentMiniColumn, Layer) when CurrentMiniColumn == ?N_IN ->
```

```
Layer;
```

```
getInLayerHelper(CurrentMiniColumn, Layer) ->
```

```
getInLayerHelper(CurrentMiniColumn + 1, maps:put(CurrentMiniColumn, getInMiniColumn(),
Layer)).
```

```
getInLayer() -> getInLayerHelper(0, #{}).
```

```
% Функция создания выходного слоя одной колонки
```

```
getOutLayerHelper(CurrentCell, Layer) when CurrentCell == ?N_OUT ->
```

```
Layer;
```

```
getOutLayerHelper(CurrentCell, Layer) ->
```

```
getOutLayerHelper(CurrentCell + 1, maps:put(CurrentCell, {'MyMath':getId(), getCell(?N_OUT)},
Layer)).
```

```
getOutLayer() -> getOutLayerHelper(0, #{}).
```

```

% Обход клеток выходного слоя
getSynapsesBetweenLayersHelper([], _CurrentInCell, ResultMap) ->
    ResultMap;
getSynapsesBetweenLayersHelper([CurrentOutCell | TOut], CurrentInCell, ResultMap) ->
    getSynapsesBetweenLayersHelper(TOut, CurrentInCell, addSynapseInDendrite({CurrentInCell,
CurrentOutCell}, getSynapse(), ResultMap)).

% Обход клеток входного слоя
getSynapsesBetweenLayers([], _To, ResultMap) ->
    ResultMap;
getSynapsesBetweenLayers([CurrentFrom | FromT], To, ResultMap) ->
    getSynapsesBetweenLayers(FromT, To, getSynapsesBetweenLayersHelper(To, CurrentFrom,
ResultMap)).

% Функция возвращает карту с синапсами между входным и выходным слоем по ключу
{CurrentInCell, CurrentOutCell}
getFeedForward() -> getSynapsesBetweenLayers('GlobalDataService':getAllInCells(),
'GlobalDataService':getAllOutCells(), #{}).

% Функция возвращает карту с синапсами между выходным и входным слоем по ключу
{CurrentOutCell, CurrentInCell}
getFeedBack() -> getSynapsesBetweenLayers('GlobalDataService':getAllOutCells(),
'GlobalDataService':getAllInCells(), #{}).

% Перебор клеток в мини-колонке
getAllInCellsByCells(CellIterator, AllInCells, ColumnRange) ->
    case maps:next(CellIterator) of
        none -> AllInCells;
        {CellGuid, _DendriteMap, NewCellIterator} ->
            getAllInCellsByCells(NewCellIterator, lists:append(AllInCells, [{ColumnRange, CellGuid}]),
ColumnRange)
    end.

% Перебор мини-колонок во входном слое
getAllInCellsByMiniColumns(MiniColumnIterator, AllInCells) ->
    case maps:next(MiniColumnIterator) of
        none -> AllInCells;
        {ColumnRange, CellMap, NewMiniColumnIterator} ->
            getAllInCellsByMiniColumns(NewMiniColumnIterator,
            getAllInCellsByCells(maps:iterator(CellMap), AllInCells, ColumnRange))
    end.

% Функция возвращает номер мини-колонки и Guid всех клеток входного слоя
getAllInCells() -> getAllInCellsByMiniColumns(maps:iterator('GlobalDataService':getInLayer()), []).

```

```

% Перебор клеток в выходном слое.
% Да, дубль функции, но причины для изменения разные
getAllOutCellsHelper(CellIterator, AllOutCells) ->
  case maps:next(CellIterator) of
    none -> AllOutCells;
    {Range, {CellGuid, _Cell}, NewCellIterator} ->
      getAllOutCellsHelper(NewCellIterator, lists:append(AllOutCells, [{Range, CellGuid}])))
  end.

% Функция возвращает Guid всех клеток выходного слоя
getAllOutCells() -> getAllOutCellsHelper(maps:iterator('GlobalDataService':getOutLayer()), []).

```

```

initializeGlobalData() ->
  put(?ID, 0),
  'GlobalDataService':putInLayer(getInLayer()),
  'GlobalDataService':putOutLayer(getOutLayer()),
  'GlobalDataService':putAllInCells(getAllInCells()),
  'GlobalDataService':putAllOutCells(getAllOutCells()),
  'GlobalDataService':putFeedForward(getFeedForward()),
  'GlobalDataService':putFeedBack(getFeedBack()),
  ok.

```

Ядро системы. Модуль BrainService.

```

%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 23. май 2024 17:35
%%%-----
-module('BrainService').
-author("Potap").

%% API
-export([initializeBrain/0, sendExternalSignal/1, sendFeedForwardSignal/1]).

% Заполнение слоев синапсами, создание связей между синапсами
initializeBrain() ->
  'BrainInit':initializeGlobalData().

% Отправка сигнала местонахождения (L)
sendExternalSignal(Signal) ->
  'GlobalDataService':putInPredictedCells('PredictCells':getPredictedCellsInInputLayer(Signal)),
  ok.

% Отправка сенсорного (прямого) сигнала
sendFeedForwardSignal(Signal) ->
  'GlobalDataService':putInActiveCells('ActivateInCells':getActiveCells(Signal)),

```



```
'GlobalDataService':putOutPreviousActivation('GlobalDataService':getOutActiveCells()), %
Сохранение активации выходного слоя
'GlobalDataService':putOutActiveCells('ActivateOutCells':getOutActiveCells()),
'Education':train(),
ok.
```

Ядро системы. Модуль CommonFunctions.

```
%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 19. май 2024 16:06
%%%-----
-module('CommonFunctions').
-author("Potap").

%% API
-export([existSynapseInMap/2, existActiveApicalDendriteByCellGuid/1,
  hasMiniColumnInPredict/1, hasActiveApicalDendriteInPredict/1, getListSize/1,
  getMiniColumnsWithActiveApicalDendrite/0,
  existActiveApicalDendriteByRanges/1, existSynapseBetweenLayersByRanges/3,
  findSynapseInMapByFRTG/3,
  findSynapseInMapByFRTR/4, existSynapseBetweenLayersByFRTG/3,
  existSynapseBetweenLayersByFGTR/3]).

-include("Model/Model.hrl").

% Проверка существования синапса
existSynapseInMap(Range, SynapsesMap) ->
case maps:find(Range, SynapsesMap) of
  {ok, Value} ->
    case Value#synapse.permanenceWeight of
      true -> {1, Value};
      false -> 0
    end;
  _ -> 0
end.

% Проверка существования синапса между клетками слоев
% По рэнджу клетки входного слоя и guid клетки выходного слоя
findSynapseInMapByFRTG(FromCellRange, ToCellGuid, Iterator) ->
case maps:next(Iterator) of
  none -> none;
  {{ {FromCellRange, _}, {_, ToCellGuid} }, Value, _NewIterator} ->
    {ok, Value};
  {_, _, NewIterator} -> findSynapseInMapByFRTG(FromCellRange, ToCellGuid, NewIterator)
end.

% Проверка существования синапса между клетками слоев
```

```

% По guid клетки входного слоя и рэнджу клетки выходного слоя
findSynapseInMapByFGTR(FromCellGuid, ToCellRange, Iterator) ->
  case maps:next(Iterator) of
    none -> none;
    {{_, FromCellGuid}, {ToCellRange, _}}, Value, _NewIterator} ->
      {ok, Value};
    {_, _, NewIterator} -> findSynapseInMapByFGTR(FromCellGuid, ToCellRange, NewIterator)
  end.

% Проверка существования синапса между клетками слоев
% По рэнджу клетки входного слоя и рэнджу клетки выходного слоя
% TODO Если ищем фидбэк от выходного слоя, то мало найти синапс. Нужно найти ВСЕ и
% смотреть на их устойчивость
findSynapseInMapByFRTR(FromCellRange, ToCellRange, Iterator, Synapses) ->
  case maps:next(Iterator) of
    none -> Synapses;
    {{{FromCellRange, _}, {ToCellRange, _}}, Value, NewIterator} ->
      findSynapseInMapByFRTR(FromCellRange, ToCellRange, NewIterator, lists:append(Synapses,
[Value]));
    {_, _, NewIterator} -> findSynapseInMapByFRTR(FromCellRange, ToCellRange, NewIterator,
Synapses)
  end.

% TODO Подумать, как параметризовать три функции ниже
existSynapseBetweenLayersByFRTG(FromCellRange, ToCellGuid, SynapsesMap) ->
  case findSynapseInMapByFRTG(FromCellRange, ToCellGuid, maps:iterator(SynapsesMap)) of
    {ok, Value} ->
      case Value#synapse.permanenceWeight of
        true -> {true, Value};
        false -> false
      end;
    _ -> false
  end.

existSynapseBetweenLayersByFGTR(FromCellGuid, ToCellRange, SynapsesMap) ->
  case findSynapseInMapByFGTR(FromCellGuid, ToCellRange, maps:iterator(SynapsesMap)) of
    {ok, Value} ->
      case Value#synapse.permanenceWeight of
        true -> {true, Value};
        false -> false
      end;
    _ -> false
  end.

checkSynapsePermanenceWeight([]) ->
  false;
checkSynapsePermanenceWeight([Synapse | T]) ->
  case Synapse#synapse.permanenceWeight of

```

```

true -> {true, Synapse};
false -> checkSynapsePermanenceWeight(T)
end.

```

```

existSynapseBetweenLayersByRanges(FromRange, ToRange, SynapsesMap) ->
  checkSynapsePermanenceWeight(findSynapseInMapByFRTR(FromRange, ToRange,
maps:iterator(SynapsesMap), [])) ).

```

```

% Проверка на наличие апикального дендрита у клетки входного слоя по Guid
existActiveApicalDendriteByCellGuid(_ToInCellGuid, []) ->
  false;
existActiveApicalDendriteByCellGuid(_ToInCellGuid, undefined) ->
  false;
existActiveApicalDendriteByCellGuid(ToInCellGuid, [ActiveOutCellRange | H]) ->
  case existSynapseBetweenLayersByFRTG(ActiveOutCellRange, ToInCellGuid,
'GlobalDataService':getFeedBack()) of
    {true, _Value} -> {true, ActiveOutCellRange};
    false -> existActiveApicalDendriteByCellGuid(ToInCellGuid, H)
  end.
existActiveApicalDendriteByCellGuid(ToInCellGuid) ->
  existActiveApicalDendriteByCellGuid(ToInCellGuid, 'GlobalDataService':getOutActiveCells()).

```

```

% Проверка на наличие мини-колонки в предсказанных
hasMiniColumnInPredict(RangeOfColumnWithFeedForward) ->
  case maps:find(RangeOfColumnWithFeedForward, 'GlobalDataService':getInPredictedCells()) of
    {ok, _Value} -> true;
    _ -> false
  end.

```

```

hasActiveApicalDendriteInPredictByMiniColumn(Iterator) ->
  case maps:next(Iterator) of
    none -> false;
    {_CellGuid, {?NoActiveApicalDendrite, _}, NewIterator} ->
hasActiveApicalDendriteInPredictByMiniColumn(NewIterator);
    {_CellGuid, {_ApicalDendrite, _}, _NewIterator} -> true
  end.

```

```

% Определение наличия апикального дендрита хотя бы у одной клетки в мини-колонке
hasActiveApicalDendriteInPredict(LayerIterator, Ret) ->
  case maps:next(LayerIterator) of
    none -> Ret;
    {_Range, MiniColumn, NewIterator} ->
    hasActiveApicalDendriteInPredict(NewIterator,
hasActiveApicalDendriteInPredictByMiniColumn(maps:iterator(MiniColumn)))
  end.

```

```

hasActiveApicalDendriteInPredict(LayerIterator) ->
  hasActiveApicalDendriteInPredict(LayerIterator, false).

```

```

% Проверка на наличие апикального дендрита у клетки входного слоя по номеру мини-колонок
existActiveApicalDendriteByRanges(_ToRange, []) ->
    false;
existActiveApicalDendriteByRanges(_ToRange, undefined) ->
    false;
existActiveApicalDendriteByRanges(ToRange, [ActiveOutCellRange | H]) ->
    case existSynapseBetweenLayersByRanges(ActiveOutCellRange, ToRange,
'GlobalDataService':getFeedBack()) of
        {true, _Value} -> true;
        false -> existActiveApicalDendriteByRanges(ToRange, H)
    end.
existActiveApicalDendriteByRanges(ToRange) ->
    existActiveApicalDendriteByRanges(ToRange, 'GlobalDataService':getOutActiveCells()).

% Получение списка мини-колонок, у которых есть апикальный дендрит
getMiniColumnsWithActiveApicalDendrite(LayerIterator, TargetColumns) ->
    case maps:next(LayerIterator) of
        none -> TargetColumns;
        {Range, _MiniColumnMap, NewIterator} ->
            case existActiveApicalDendriteByRanges(Range) of
                true -> getMiniColumnsWithActiveApicalDendrite(NewIterator, lists:append(TargetColumns,
[Range]));
                false -> getMiniColumnsWithActiveApicalDendrite(NewIterator, TargetColumns)
            end
        end
    end.

getMiniColumnsWithActiveApicalDendrite() ->
    getMiniColumnsWithActiveApicalDendrite(maps:iterator('GlobalDataService':getInLayer()), []).

% Количество элементов в списке
getListSize([]) -> 0;
getListSize(List) ->
    getListSize(List, 0).

getListSize([], Count) ->
    Count;
getListSize([_ | L], Count) ->
    getListSize(L, Count + 1).
Ядро системы. Модуль DevelopersScripts.
%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 23. май 2024 17:47
%%%-----
-module('DevelopersScripts').

```

```

-author("Potap").

%% API
-export([script_0/0, script_1/1, script_1_withHardCode/0, script_2/1, printToFileThousandSynapses/0,
script_3/0]).

% Инициализация синапсов во всех структурах
script_0() ->
  'BrainService':initializeBrain().

% Отправка сигнала местоположения, получение предсказанных клеток
script_1(Signal) ->
  'BrainService':sendExternalSignal(Signal),
  'GlobalDataService':getInPredictedCells().

% Отправка сигнала местоположения, получение предсказанных клеток
script_1_withHardCode() ->
  'BrainService':sendExternalSignal([11, 12, 23, 14, 35, 36, 37, 30, 39, 40]),
  'GlobalDataService':getInPredictedCells().

% Отправка прямого сигнала, получение активных клеток выходного слоя
script_2(List) ->
  'BrainService':sendFeedForwardSignal(List),
  'GlobalDataService':getOutActiveCells().

% Вывод министолбцов входного слоя с активными апикальными дендритами
script_3() ->
  'CommonFunctions':getMiniColumnsWithActiveApicalDendrite().

% Вывод на экран 1000 синапсов
printToFileThousandSynapses(Count, Synapses) when Count == 1000 ->
  Synapses;
printToFileThousandSynapses(Count, Synapses) ->
  printToFileThousandSynapses(Count + 1, lists:append(Synapses, ['BrainInit':getSynapse()])).

printToFileThousandSynapses() ->
  'HelpFunctions':listWriteToFile('ThousandSynapses.tb', printToFileThousandSynapses(0, [])).
Ядро системы. Модуль GlobalDataService.
%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 23. май 2024 17:50
%%%-----
-module('GlobalDataService').
-author("Potap").

```

```

%% API
-export([
    putInLayer/1, getInLayer/0, putOutLayer/1, getOutLayer/0, putAllInCells/1, getAllInCells/0,
    putAllOutCells/1, getAllOutCells/0, putFeedForward/1, getFeedForward/0, putFeedBack/1,
    getFeedBack/0,
    putInPredictedCells/1, getInPredictedCells/0, putOutActiveCells/1, getOutActiveCells/0,
    putInActiveCells/1, getInActiveCells/0, putOutPreviousActivation/1, getOutPreviousActivation/0,
    putSenderMode/1, getSenderMode/0,
    putLocationSignal/1, getLocationSignal/0, appendLocationSignal/1,
    putSensorySignal/1, getSensorySignal/0, appendSensorySignal/1,
    putEducationMode/1, getEducationMode/0]).

-include("Model/Model.hrl").

% TODO Добавить общую проверку на undefined

putInLayer(Layer) ->
    put(?InLayer, Layer).

getInLayer() ->
    get(?InLayer).


putOutLayer(Layer) ->
    put(?OutLayer, Layer).

getOutLayer() ->
    get(?OutLayer).


putAllInCells(Cells) ->
    put(?AllInCells, Cells).

getAllInCells()->
    get(?AllInCells).


putAllOutCells(Cells) ->
    put(?AllOutCells, Cells).

getAllOutCells()->
    get(?AllOutCells).


putFeedForward(Synapses) ->
    put(?FeedForward, Synapses).

```

```
getFeedForward()->  
  get(?FeedForward).
```

```
putFeedBack(Synapses) ->  
  put(?FeedBack, Synapses).
```

```
getFeedBack()->  
  get(?FeedBack).
```

```
putInPredictedCells(Cells) ->  
  put(?InPredictedCells, Cells).
```

```
getInPredictedCells()->  
  get(?InPredictedCells).
```

```
putInActiveCells(Cells) ->  
  put(?InActiveCells, Cells).
```

```
getInActiveCells()->  
  get(?InActiveCells).
```

```
putOutActiveCells(Cells) ->  
  put(?OutActiveCells, Cells).
```

```
getOutActiveCells()->  
  get(?OutActiveCells).
```

```
putOutPreviousActivation(OutActiveCells) ->  
  put(?OutPreviousActivation, OutActiveCells).
```

```
getOutPreviousActivation()->  
  get(?OutPreviousActivation).
```

```
putSenderMode(Mode) ->  
  put(?SenderMode, Mode).
```

```
getSenderMode()->  
  get(?SenderMode).
```



```
putLocationSignal(Signal) ->
    put(?LocationSignal, Signal).
```

```
getLocationSignal() ->
    get(?LocationSignal).
```

```
appendLocationSignal(Signal) ->
    put(?LocationSignal, lists:append(getLocationSignal(), [Signal])).
```

```
putSensorySignal(Signal) ->
    put(?SensorySignal, Signal).
```

```
getSensorySignal() ->
    get(?SensorySignal).
```

```
appendSensorySignal(Signal) ->
    put(?SensorySignal, lists:append(getSensorySignal(), [Signal])).
```

```
putEducationMode(Mode) ->
    put(?EducationMode, Mode).
```

```
getEducationMode()->
    get(?EducationMode).
```

Ядро системы. Модуль HelpFunctions.

```
%%%-----
```

```
%%% @author Potap
```

```
%%% @copyright (C) 2024, <COMPANY>
```

```
%%% @doc
```

```
%%%
```

```
%%% @end
```

```
%%% Created : 18. май 2024 19:00
```

```
%%%-----
```

```
-module('HelpFunctions').
```

```
-author("Potap").
```

```
%% API
```

```
-export([mapWriteToFile/2, listWriteToFile/2, getIntegerFromString/1, printGlobalData/0]).
```

```
-include("Model/ProjectSettings.hrl").
```

```
-include("Model/Model.hrl").
```

```
% Функция вывода Map в файл с заданным именем
```

```
% TODO Рассмотреть идею - сначала формировать большую строку, а затем разом писать ее в файл
```

```

mapWriteToFile(File, Map) ->
case Map of
  undefined ->
    error;
  _ ->
    {ok, S} = file:open(?FileDirectory ++ File, write),
    maps:foreach(
      fun(Key, Value) ->
        if
          Value#synapse.permanenceWeight == false -> none;
          true -> file:write_file(?FileDirectory ++ File, io_lib:fwrite("~p -> ~p \n", [Key, Value]), [append])
        end
      end, Map),
    file:close(S)
end.

```

% Функция вывода List в файл

```

listWriteToFile(File, List) ->
case List of
  undefined ->
    error;
  _ ->
    {ok, S} = file:open(?FileDirectory ++ File, write),
    lists:foreach(
      fun(Value) ->
        file:write_file(?FileDirectory ++ File, io_lib:fwrite("~p\n", [Value]), [append])
      end, List),
    file:close(S)
end.

```

% Функция выводит в файлы все глобальные данные

```

printGlobalData() ->
'HelpFunctions':mapWriteToFile('InLayer.tb', 'GlobalDataService':getInLayer()),
'HelpFunctions':mapWriteToFile('OutLayer.tb', 'GlobalDataService':getOutLayer()),
'HelpFunctions':mapWriteToFile('FeedForward.tb', 'GlobalDataService':getFeedForward()),
'HelpFunctions':mapWriteToFile('FeedBack.tb', 'GlobalDataService':getFeedBack()),
'HelpFunctions':mapWriteToFile('InActiveLayer.tb', 'GlobalDataService':getInActiveCells()),
'HelpFunctions':mapWriteToFile('InPredictedCells.tb', 'GlobalDataService':getInPredictedCells()),
'HelpFunctions':listWriteToFile('OutActiveLayer.tb', 'GlobalDataService':getOutActiveCells()),
'HelpFunctions':listWriteToFile('PreviousOutActiveLayer.tb',
'GlobalDataService':getOutPreviousActivation()).

```

% Функция получает на вход строку.

% Если получилось преобразовать в число, то вернет это число.

% Иначе вернет false

```

getIntegerFromString(X) ->
case string:to_integer(X) of
  {A, B} ->
    if
      % Если получилось распарсить и такм не оказалось посторонних символов
      length(B) == 0 -> A;

```

```

% Если есть мусор, то это не число
true -> false
end;
% Не получилось распарсить
_ -> false
end.

```

Ядро системы. Модуль Main.

```

%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 14. май 2024 2:00
%%%-----
-module('Main').
-author("Potap").

%% API
-export([main/0]).

```

```

main() ->
'VisualisationClient':initializeSocket(),
'VisualisationClient':runReceiver().

```

Ядро системы. Модуль MyMath.

```

%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 18. май 2024 15:29
%%%-----
-module('MyMath').
-author("Potap").

-include("Model/BrainSettings.hrl").
-include("Model/Model.hrl").

```

```

%% API
-export([getPoissonValue/0, getId/0, getGuid/0]).

```

```

factorial(0) -> 1;
factorial(N) when N > 0 -> N * factorial(N - 1).

```

```

poisson(Lambda, K) ->
(math:pow(Lambda, K) * math:exp(-Lambda)) / factorial(K).

```

```

% Функция получения числа через распределение Пуассона
% Коэффициенты подобраны опытным путем

```

```
getPoissonValue() -> round(poisson(rand:uniform() * 7, 1) * ?ROUND_EPSILON) /
?ROUND_EPSILON.
```

```
% GUID
```

```
getGuid() ->
```

```
<<A:32, B:16, C:16, D:16, E:48>> = rand:bytes(16),
list_to_binary(io_lib:format("~8.16.0b~4.16.0b~4.16.0b~12.16.0b",
[A, B, C band 16#0fff, D band 16#3fff bor 16#8000, E])).
```

```
getId() ->
```

```
put(?ID, get(?ID) + 1),
```

```
get(?ID).
```

Ядро системы. Модуль PredictCells.

```
%%%-----
```

```
%%% @author Potap
```

```
%%% @copyright (C) 2024, <COMPANY>
```

```
%%% @doc
```

```
%%%
```

```
%%% @end
```

```
%%% Created : 19. май 2024 14:57
```

```
%%%-----
```

```
-module('PredictCells').
```

```
-author("Potap").
```

```
%% API
```

```
-export([getPredictedCellsInInputLayer/1, getPredictedCellsInOutputLayer/3]).
```

```
-include("Model/BrainSettings.hrl").
```

```
-include("Model/Model.hrl").
```

```
% В этой функции определяется есть ли на дендрите достаточное количество синапсов,
соответствующих входному сигналу
```

```
% Первый аргумент - Входной сигнал (L - для входного слоя, Выходной слой - для выходного
слоя)
```

```
% SynapseMap - Мапа со значениями постоянства синапсов. Берется из входного слоя
(Map<Range, Synapse>)
```

```
% SynapseCount - Количество существующих синапсов (соответствующих входному сигналу)
```

```
% Theta - Порог активации дендрита
```

```
passedThetaInBThreshold([], _SynapsesMap, SynapseCount, Theta) when SynapseCount >= Theta ->
```

```
% Если порог активации дендрита пройден
```

```
true;
```

```
passedThetaInBThreshold([], _SynapsesMap, _SynapseCount, _Theta) ->
```

```
% Если порог активации НЕ пройден, то дендрит не добавляется
```

```
false;
```

```
passedThetaInBThreshold([Range|_T], SynapsesMap, SynapseCount, Theta) ->
```

```
% Рекурсивная обработка входных сигналов
```

```
case 'CommonFunctions':existSynapseInMap(Range, SynapsesMap) of
```

```

{1, _Value} -> passedThetaInBThreshold(T, SynapsesMap, SynapseCount + 1, Theta);
0 -> passedThetaInBThreshold(T, SynapsesMap, SynapseCount, Theta)
end.

```

```

% В этой функции собираем активные дендриты клетки
% Signal - передается дальше для сравнения
% DendriteIterator - итератор для перебора карты с дендритами
% ActiveDendrites - out переменная, список для сохранения идентификаторов активных дендритов
% Theta - Порог активации дендрита
findActiveDendrites(Signal, DendriteIterator, ActiveDendrites, Theta) ->
case maps:next(DendriteIterator) of
  % Когда перебрали все дендриты в карте - возвращаем список GUID активных дендритов
  none -> ActiveDendrites;
  {DendriteGuid, SynapsesMap, NewDendriteIterator} ->
    case passedThetaInBThreshold(Signal, SynapsesMap, 0, Theta) of
      % Если на дендрите есть достаточно синапсов, то добавляем GUID дендрита в список
      активных
      true -> findActiveDendrites(Signal, NewDendriteIterator, lists:append(ActiveDendrites,
[DendriteGuid]), Theta);
      % Иначе просто переходим на следующую итерацию
      false -> findActiveDendrites(Signal, NewDendriteIterator, ActiveDendrites, Theta)
    end
end.

```

```

% В этой функции определяем предсказанные клетки
% Signal - передается дальше для сравнения
% CellIterator - итератор для перебора карты с клетками
% PredictedCells - карта, где ключ - Guid предсказанной клетки, значение - список Guid дендритов,
которые привели к предсказанию
% TODO Подумать, как избежать дублирования кода в case по поиску апикального дендрита
findPredictedCells(Signal, CellIterator, PredictedCells) ->
case maps:next(CellIterator) of
  % Когда перебрали все клетки в карте (мини-столбце). Сразу вернем количество объектов в
мапе, потому что сопоставление по пустоте недопустимо
  none -> {map_size(PredictedCells), PredictedCells};
  {CellGuid, DendriteMap, NewCellIterator} ->
    % Определяем, есть ли активные дендриты на клетке
    case 'CommonFunctions':existActiveApicalDendriteByCellGuid(CellGuid) of
      % Есть апикальный дендрит
      {true, OutActiveCellRange} ->
        case findActiveDendrites(Signal, maps:iterator(DendriteMap), [], ?THETA_IN_B_MIN) of
          % Если функция вернула пустой список, значит нет активных дендритов, а значит, клетка
не станет предсказанной. Просто переходим к следующей итерации
          [] -> findPredictedCells(Signal, NewCellIterator, PredictedCells);
          % Иначе возвращен список активных дендритов, значит клетка предсказана (Так как
достаточно одного активного дендрита)

```

```

        ActiveDendrites -> findPredictedCells(Signal, NewCellIterator, maps:put(CellGuid,
{OutActiveCellRange, ActiveDendrites}, PredictedCells))
    end;
    % Нет апикального дендрита
    false ->
        case findActiveDendrites(Signal, maps:iterator(DendriteMap), [], ?THETA_IN_B_MAX) of
            % Если функция вернула пустой список, значит нет активных дендритов, а значит, клетка
не станет предсказанной. Просто переходим к следующей итерации
            [] -> findPredictedCells(Signal, NewCellIterator, PredictedCells);
            % Иначе возвращен список активных дендритов, значит клетка предсказана (Так как
достаточно одного активного дендрита)
            ActiveDendrites -> findPredictedCells(Signal, NewCellIterator, maps:put(CellGuid,
{?NoActiveApicalDendrite, ActiveDendrites}, PredictedCells))
        end
    end
end.

```

```

% В этой функции определяем мини-колонки с деполяризованными клетками
findMiniColumnWithPredictedCells(Signal, MiniColumnIterator, MiniColumns) ->
    case maps:next(MiniColumnIterator) of
        % Когда перебрали все клетки в мапе (мини-столбце)
        none -> MiniColumns;
        {ColumnRange, CellMap, NewMiniColumnIterator} ->
            case findPredictedCells(Signal, maps:iterator(CellMap), #{}) of
                % Если функция вернула пустую мапу, значит нет предсказанных клеток в мини-колонке.
Просто переходим к следующей итерации
                {0, _} -> findMiniColumnWithPredictedCells(Signal, NewMiniColumnIterator, MiniColumns);
                {_, PredictedCells} -> findMiniColumnWithPredictedCells(Signal, NewMiniColumnIterator,
maps:put(ColumnRange, PredictedCells, MiniColumns))
            end
        end
    end.

```

```

% Функция возвращает предсказанные клетки и дендриты, которые привели к деполяризации.
Данные упакованы в иерархию, аналогичную структуре хранения данных
getPredictedCellsInInputLayer(Signal) ->
    findMiniColumnWithPredictedCells(Signal, maps:iterator('GlobalDataService':getInLayer()), #{}).

```

```

% Функция возвращает предсказанные клетки в выходном слое, основываясь на активных клетках
предыдущего шага
% Первый аргумент - сигнал, активные клетки с предыдущего шага - мапа
getPredictedCellsInOutputLayer([], _OutCellsIterator, PredictedCells) ->
    PredictedCells;
getPredictedCellsInOutputLayer(ActiveCellsInOutputLayerOnPreviousTimeStep, OutCellsIterator,
PredictedCells) ->
    case maps:next(OutCellsIterator) of

```

```

none -> PredictedCells;
{Range, {_OutCellGuid, DendriteMap}, NewCellIterator} ->
  case findActiveDendrites(ActiveCellsInOutputLayerOnPreviousTimeStep,
maps:iterator(DendriteMap), [], ?THETA_OUT_B) of
  [] -> getPredictedCellsInOutputLayer(ActiveCellsInOutputLayerOnPreviousTimeStep,
NewCellIterator, PredictedCells);
  ActiveDendrites ->
getPredictedCellsInOutputLayer(ActiveCellsInOutputLayerOnPreviousTimeStep, NewCellIterator,
maps:put(Range, ActiveDendrites, PredictedCells))
  end
end.

```

Ядро системы. Модуль VisualisationClient.

```

%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 28. май 2024 22:05
%%%-----
-module('VisualisationClient').
-author("Potap").

%% API
-export([sendInformMessage/2, sendIntegerValue/2, sendFloatValue/2, sendBoolValue/2,
initializeSocket/0, runReceiver/0]).

-include("Model/ServerSettings.hrl").
-include("Model/Commands.hrl").

% Команды отправки разных типов данных
sendInformMessage(Socket, Message) ->
  gen_tcp:send(Socket, Message ++ "\n").

sendIntegerValue(Socket, Value) ->
  gen_tcp:send(Socket, integer_to_list(Value) ++ "\n").

sendFloatValue(Socket, Value) ->
  gen_tcp:send(Socket, float_to_list(Value, [{decimals, 4}]) ++ "\n").

sendBoolValue(Socket, Value) ->
  case Value of
    false -> gen_tcp:send(Socket, "false" ++ "\n");
    true -> gen_tcp:send(Socket, "true" ++ "\n")
  end.

% Создание сокета {ok, Socket} = gen_tcp:connect({127, 0, 0, 1}, 8888, [binary, {active, false}]).
initializeSocket() ->
  {ok, Socket} = gen_tcp:connect(?IP_ADDRESS, ?PORT, [binary, {active, false}, {packet, line}]),

```



```
put(socket, Socket).
```

```
% Бесконечный прием сообщений
```

```
runReceiver() ->
```

```
case gen_tcp:recv(get(socket), 0) of
```

```
{ok, Command} ->
```

```
'VisualisationSender':handleCommand(lists:delete($\n, lists:delete($\r, binary_to_list(Command))));
```

```
{error, Reason} ->
```

```
Reason
```

```
end,
```

```
runReceiver().
```

Ядро системы. Модуль VisualisationSender.

```
%%%-----
```

```
%%% @author Potap
```

```
%%% @copyright (C) 2024, <COMPANY>
```

```
%%% @doc
```

```
%%%
```

```
%%% @end
```

```
%%% Created : 02. июнь 2024 13:29
```

```
%%%-----
```

```
-module('VisualisationSender').
```

```
-author("Potap").
```

```
%% API
```

```
-export([handleCommand/1]).
```

```
-include("Model/Commands.hrl").
```

```
-include("Model/Model.hrl").
```

```
-include("Model/SenderMode.hrl").
```

```
-include("Model/EducationMode.hrl").
```

```
-include("Model/ServerSettings.hrl").
```

```
% TODO Написать проверку на существование сокета
```

```
% Обработка сообщений, полученных от модуля визуализации
```

```
handleCommand(Command) ->
```

```
% Если нам прислали число, то определяем для чего оно пришло. Иначе это буквенная команда
```

```
case 'HelpFunctions':getIntegerFromString(Command) of
```

```
false ->
```

```
case Command of
```

```
?NeedBrainInitialize ->
```

```
'BrainService':initializeBrain(),
```

```
sendInLayer(),
```

```
sendFeedBack(),
```

```
sendFeedForward(),
```

```
sendOutLayer();
```

```
?NeedBrainPrint ->
```

```
'HelpFunctions':printGlobalData();
```

```

?LocationSignalBegin ->
    % Устанавливаем режим для распознавания сигнала местоположения и обнуляем старый
    сигнал
    'GlobalDataService':putSenderMode(?LocationSignalMode),
    'GlobalDataService':putLocationSignal([]);
?LocationSignalEnd ->
    % Сбрасываем режим, отправляем сигнал мозгу и данные в модуль визуализации
    'GlobalDataService':putSenderMode(?NoneSenderMode),
    'BrainService':sendExternalSignal('GlobalDataService':getLocationSignal()),
    sendPredictInLayer();

?SensorySignalBegin ->
    % Устанавливаем режим для распознавания сенсорного сигнала и обнуляем старый сигнал
    'GlobalDataService':putSenderMode(?SensorySignalMode),
    'GlobalDataService':putSensorySignal([]);
?SensorySignalEnd ->
    % Сбрасываем режим, отправляем сигнал мозгу и данные в модуль визуализации
    'GlobalDataService':putSenderMode(?NoneSenderMode),
    'BrainService':sendFeedForwardSignal('GlobalDataService':getSensorySignal()),
    sendActiveInLayer(),
    sendActiveOutLayer();

?EducationModeBegin ->
    'GlobalDataService':putSenderMode(?EducationCommandMode);

_ ->
    io:format("Поступила неизвестная команда: ~p", [Command])

end;
% У нас число
_ ->
    case 'GlobalDataService':getSenderMode() of
        % Заполняем сигнал местоположения
        ?LocationSignalMode ->
            'GlobalDataService':appendLocationSignal('HelpFunctions':getIntegerFromString(Command));

        % Заполняем сенсорный сигнал
        ?SensorySignalMode ->
            'GlobalDataService':appendSensorySignal('HelpFunctions':getIntegerFromString(Command));

        % Кстанавливаем режим обучения модели
        ?EducationCommandMode ->
            'GlobalDataService':putEducationMode('HelpFunctions':getIntegerFromString(Command)),
            'GlobalDataService':putSenderMode(?NoneSenderMode);

        _ -> error
    end
end.

```

```

sendInLayer() ->
'VisualisationClient':sendInformMessage(get(socket), ?StructureName_InLayer),
sendData(get(socket), 'GlobalDataService':getInLayer()),
'VisualisationClient':sendInformMessage(get(socket), ?StructureEnd).

sendOutLayer() ->
'VisualisationClient':sendInformMessage(get(socket), ?StructureName_OutLayer),
sendData(get(socket), 'GlobalDataService':getOutLayer()),
'VisualisationClient':sendInformMessage(get(socket), ?StructureEnd).

sendPredictInLayer() ->
'VisualisationClient':sendInformMessage(get(socket), ?StructureName_PredictInLayer),
sendData(get(socket), 'GlobalDataService':getInPredictedCells()),
'VisualisationClient':sendInformMessage(get(socket), ?StructureEnd).

sendActiveInLayer() ->
'VisualisationClient':sendInformMessage(get(socket), ?StructureName_ActiveInLayer),
sendData(get(socket), 'GlobalDataService':getInActiveCells()),
'VisualisationClient':sendInformMessage(get(socket), ?StructureEnd).

sendActiveOutLayer() ->
'VisualisationClient':sendInformMessage(get(socket), ?StructureName_ActiveOutLayer),
sendData(get(socket), 'GlobalDataService':getOutActiveCells()),
'VisualisationClient':sendInformMessage(get(socket), ?StructureEnd).

sendFeedForward() ->
'VisualisationClient':sendInformMessage(get(socket), ?StructureName_FeedForward),
sendData(get(socket), 'GlobalDataService':getFeedForward()),
'VisualisationClient':sendInformMessage(get(socket), ?StructureEnd).

sendFeedBack() ->
'VisualisationClient':sendInformMessage(get(socket), ?StructureName_FeedBack),
sendData(get(socket), 'GlobalDataService':getFeedBack()),
'VisualisationClient':sendInformMessage(get(socket), ?StructureEnd).

sendData(Socket, Structure) ->
if
  is_map(Structure) -> sendMap(Socket, Structure);
  is_list(Structure) -> sendList(Socket, Structure);
  is_record(Structure, synapse) -> sendSynapse(Socket, Structure);
  is_integer(Structure) -> 'VisualisationClient':sendIntegerValue(Socket, Structure);
  true -> sendSimpleData(Socket, Structure)
end.

sendMap(Socket, Map) ->
'VisualisationClient':sendInformMessage(Socket, ?MarkerMapBegin),
maps:foreach(
  fun(Key, Value) ->

```

```

    sendData(Socket, Key),
    sendData(Socket, Value)
end, Map),
'VisualisationClient':sendInformMessage(Socket, ?MarkerMapEnd).

sendList(Socket, List) ->
'VisualisationClient':sendInformMessage(Socket, ?MarkerListBegin),
lists:foreach(
  fun(Value) ->
    sendData(Socket, Value)
  end, List),
'VisualisationClient':sendInformMessage(Socket, ?MarkerListEnd).

sendSynapse(Socket, Synapse) ->
'VisualisationClient':sendInformMessage(Socket, ?MarkerSynapseBegin),
'VisualisationClient':sendIntegerValue(Socket, Synapse#synapse.guid),
'VisualisationClient':sendFloatValue(Socket, Synapse#synapse.permanenceValue),
'VisualisationClient':sendBoolValue(Socket, Synapse#synapse.permanenceWeight),
'VisualisationClient':sendInformMessage(Socket, ?MarkerSynapseEnd).

sendSimpleData(Socket, SimpleData) ->
case SimpleData of
  {noActiveApicalDendrite, List} ->
    'VisualisationClient':sendInformMessage(Socket, ?MarkerDendriteBegin),
    'VisualisationClient':sendInformMessage(Socket, ?MarkerFalse),
    sendList(Socket, List),
    'VisualisationClient':sendInformMessage(Socket, ?MarkerDendriteEnd);

  {ApicalDendrite, [H | T]} ->
    'VisualisationClient':sendInformMessage(Socket, ?MarkerDendriteBegin),
    'VisualisationClient':sendIntegerValue(Socket, ApicalDendrite),
    sendList(Socket, [H | T]),
    'VisualisationClient':sendInformMessage(Socket, ?MarkerDendriteEnd);

  {{Key1, Key2}, {Key3, Key4}} ->
    'VisualisationClient':sendInformMessage(Socket, ?MarkerFeedBegin),
    'VisualisationClient':sendIntegerValue(Socket, Key1),
    'VisualisationClient':sendIntegerValue(Socket, Key2),
    'VisualisationClient':sendIntegerValue(Socket, Key3),
    'VisualisationClient':sendIntegerValue(Socket, Key4),
    'VisualisationClient':sendInformMessage(Socket, ?MarkerFeedEnd);

  {CellGuid, Map} ->
    'VisualisationClient':sendInformMessage(Socket, ?MarkerOutColumnBegin),
    'VisualisationClient':sendIntegerValue(Socket, CellGuid),
    sendMap(Socket, Map),
    'VisualisationClient':sendInformMessage(Socket, ?MarkerOutColumnEnd);

  _ -> 'VisualisationClient':sendInformMessage(Socket, ?MarkerUndefined)
end.

```

Ядро системы. Модуль BrainSettings.

```

%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 07. июнь 2024 22:36
%%%-----
-author("Potap").

```

```

-define(N_C, 1).
-define(N_IN, 50). % По статье - 150
-define(M, 10). % По статье - 16
-define(D, 10). %
-define(N_OUT, 300). % По статье - 4096
-define(N_EXT, 200). % По статье - 2400
-define(S, 4). % По статье - 40
-define(PERMANENCE_WEIGHT_BORDER, 0.35).
-define(POTENTIAL_SYNAPSE_BORDER, 0.17).
-define(PERMANENCE_THRESHOLD, 0.3).
-define(ROUND_EPSILON, 10000).

```

```

-define(THETA_IN_B_MIN, 4). % В статье значение параметра не уточняется
-define(THETA_IN_B_MAX, 6).
-define(THETA_OUT_B, 18).
-define(THETA_OUT_C, 1).
-define(THETA_OUT_P, 3).

```

```

-define(P_IN_PLUS, 0.5).
-define(P_IN_MINUS, 0.1).
-define(P_FF_PLUS, 0.5).
-define(P_FF_MINUS, 0.1).
-define(P_OUT_PLUS, 0.5).
-define(P_OUT_MINUS, 0.1).

```

Ядро системы. Модуль Commands.

```

%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 07. июнь 2024 22:32
%%%-----
-author("Potap").

```

```

-define(MarkerMapBegin, "MapBegin").
-define(MarkerMapEnd, "MapEnd").
-define(MarkerListBegin, "ListBegin").
-define(MarkerListEnd, "ListEnd").
-define(MarkerDendriteBegin, "DendriteBegin").
-define(MarkerDendriteEnd, "DendriteEnd").

```

```

-define(MarkerSynapseBegin, "SynapseBegin").
-define(MarkerSynapseEnd, "SynapseEnd").
-define(MarkerFeedBegin, "FeedBegin").
-define(MarkerFeedEnd, "FeedEnd").
-define(MarkerOutColumnBegin, "OutColumnBegin").
-define(MarkerOutColumnEnd, "OutColumnEnd").
-define(MarkerUndefined, "UNDEFINED").
-define(MarkerFalse, "false").

-define(NeedBrainInitialize, "NeedBrainInitialize").
-define(NeedBrainPrint, "NeedBrainPrint").
-define(LocationSignalBegin, "LocationSignalBegin").
-define(LocationSignalEnd, "LocationSignalEnd").
-define(SensorySignalBegin, "SensorySignalBegin").
-define(SensorySignalEnd, "SensorySignalEnd").
-define(EducationModeBegin, "EducationModeBegin").

-define(StructureName_InLayer, "InLayer").
-define(StructureName_PredictInLayer, "PredictInLayer").
-define(StructureName_ActiveInLayer, "ActiveInLayer").
-define(StructureName_OutLayer, "OutLayer").
-define(StructureName_ActiveOutLayer, "ActiveOutLayer").
-define(StructureName_FeedForward, "FeedForward").
-define(StructureName_FeedBack, "FeedBack").
-define(StructureEnd, "END").

-define(TrueString, "true").
-define(FalseString, "false").
Ядро системы. Модуль EducationMode.
%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 12. июнь 2024 18:26
%%%-----
-author("Potap").

```

```

-define(NoEducationMode, 0).
-define(ActiveEducationMode, 1).
-define(PassiveEducationMode, 2).
Ядро системы. Модуль Model.
%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 16. май 2024 0:50
%%%-----

```

```

-author("Potap").

-record(synapse, { guid, permanenceValue, permanenceWeight :: boolean()}).

-define(InActiveCells, inActiveCells).
-define(InPredictedCells, inPredictedCells).
-define(InLayer, inLayer).

-define(OutActiveCells, outActiveCells).
-define(OutPreviousActivation, outPreviousActivation).
-define(OutLayer, outLayer).

-define(FeedForward, feedForward).
-define(FeedBack, feedBack).

-define(AllInCells, allInCells).
-define(AllOutCells, allOutCells).

-define(LocationSignal, locationSignal).
-define(SensorySignal, sensorySignal).

-define(SenderMode, senderMode).
-define(EducationMode, educationMode).

-define(NoActiveApicalDendrite, noActiveApicalDendrite).
-define(ID, id).

```

Ядро системы. Модуль ProjectSettings.

```

%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 07. июнь 2024 22:34
%%%-----
-author("Potap").

```

```

-define(FileDirectory, "tmp/").

```

Ядро системы. Модуль SenderMode.

```

%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 08. июнь 2024 0:55
%%%-----
-author("Potap").

```

```

-define(NoneSenderMode, 0).
-define(LocationSignalMode, 1).

```

```
-define(SensorySignalMode, 2).
-define(EducationCommandMode, 3).
```

Ядро системы. Модуль ServerSettings.

```
%%%-----
%%% @author Potap
%%% @copyright (C) 2024, <COMPANY>
%%% @doc
%%%
%%% @end
%%% Created : 07. июнь 2024 22:34
%%%-----
-author("Potap").
```

```
-define(IP_ADDRESS, {127, 0, 0, 1}).
-define(PORT, 8888).
```

Модуль визуализации. Класс BoolToVisibilityConverter.

```
using System.Globalization;
using System.Windows;
using System.Windows.Data;
```

```
namespace ThousandBrainsVisualisation.Converters
{
    public class BoolToVisibilityConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            try
            {
                var v = (bool)value;
                return v ? Visibility.Visible : Visibility.Collapsed;
            }
            catch (InvalidCastException)
            {
                return Visibility.Collapsed;
            }
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

Модуль визуализации. Класс BrushColorConverter.

```
using System.Globalization;
using System.Windows.Data;
using System.Windows.Media;
```

```
namespace ThousandBrainsVisualisation.Converters
{
```



```

public class BrushColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        if ((bool)value)
        {
            return Colors.Green;
        }
        return Colors.Red;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

Модуль визуализации. Класс BrainFiller.

```

using System.Globalization;
using ThousandBrainsVisualisation.Model;

namespace ThousandBrainsVisualisation.BrainFillerNS
{
    public delegate void BeginGetDataEventHandler();
    public delegate void EndGetDataEventHandler();

    public partial class BrainFiller(BrainModel brain)
    {
        // TODO Добавить в каждом switch сообщение об ошибке, если попали в default

        /*
        * Сделать универсальный механизм по заполнению любой структуры данных не получится.
        * Потому что у эрланга динамическая типизация, мы заранее не знаем, что лежит в данных и
какая вложенность.
        * При заполнении на шарпах нам это нужно знать.
        * Поэтому будем отслеживать уровень вложенности и в зависимости от заполняемой
структуры данных (InLayer, PredictCells и т.п.), будем
        * класть данные в определенные словари
        */

        private BrainModel brain = brain;
        private BrainFillerMode BrainFillerMode = BrainFillerMode.Wait;
        private DataStructureMode DataMode = DataStructureMode.None;
        private Stack<DataStructureMode> DataStructureStack = new();

        public event BeginGetDataEventHandler BeginGetData = delegate { };
        public event EndGetDataEventHandler EndGetData = delegate { };

        private int Map_CurrentLevel = 0;
        private Synapse Synapse = new();

```

```

private int? FeedKey_1;
private int? FeedKey_2;
private int? FeedKey_3;
private int? FeedKey_4;

// Структура InLayer
private int MapKey_InLayer_Level_1;
private int MapKey_InLayer_Level_2;
private int MapKey_InLayer_Level_3;
private int MapKey_InLayer_Level_4;
private Dictionary<int, Dictionary<int, Dictionary<int, Dictionary<int, Synapse>>>>
Map_InLayer_Level_1 = []; // Колонки
private Dictionary<int, Dictionary<int, Dictionary<int, Synapse>>> Map_InLayer_Level_2 = []; //
Клетки в колонке
private Dictionary<int, Dictionary<int, Synapse>> Map_InLayer_Level_3 = []; // Дендриты
private Dictionary<int, Synapse> Map_InLayer_Level_4 = []; // Синапсы

// Структура PredictInLayer
private int MapKey_PredictInLayer_Level_1;
private int MapKey_PredictInLayer_Level_2;
private Dictionary<int, Dictionary<int, Dendrites>> Map_PredictInLayer_Level_1 = [];
private Dictionary<int, Dendrites> Map_PredictInLayer_Level_2 = [];
private List<int> ActiveLateralDendrites = [];
private Dendrites ActiveDendrites = new();

// Структура ActiveInLayer
private int MapKey_ActiveInLayer_Level_1;
private Dictionary<int, List<int>> Map_ActiveInLayer_Level_1 = [];
private List<int> ActiveCellsIds = [];

// Структура OutLayer
private int MapKey_OutLayer_Level_1;
private int MapKey_OutLayer_Level_2;
private int MapKey_OutLayer_Level_3;
private Dictionary<int, (int, Dictionary<int, Dictionary<int, Synapse>>>) Map_OutLayer_Level_1 =
[];
private int CellGuid_OutLayer;
private Dictionary<int, Dictionary<int, Synapse>> Map_OutLayer_Level_2 = [];
private Dictionary<int, Synapse> Map_OutLayer_Level_3 = [];

// Структура ActiveOutLayer
private List<int> ActiveCellOutLayer = [];

// Структура FeedForward
private Dictionary<((int?, int?), (int?, int?)), Synapse> FeedForwardSynapses = [];

// Структура FeedBack
private Dictionary<((int?, int?), (int?, int?)), Synapse> FeedBackSynapses = [];

public void SendData(string? text)

```

```

{
    // Отладочная строка для отображения полученных данных
    //MainWindowViewModel.SynchronizedText += text + "\n";
    switch (BrainFillerMode)
    {
        // Проверяем какую структуру предметной области заполняем
        case BrainFillerMode.Wait:
            // До сих пор ничего не заполняли. Указываем только сейчас.
            SetMode(text);
            break;
        default:
            // Нам известно, какую структуру предметной области заполняем
            switch (DataMode)
            {
                // Проверяем, какую логическую структуру данных заполняем
                case DataStructureMode.None:
                    // Ожидаем указания заполняемой логической структуры данных
                    SetDataStructure(text);
                    break;

                case DataStructureMode.MapMode:
                    FillMap(text);
                    break;

                case DataStructureMode.ListMode:
                    FillList(text);
                    break;

                case DataStructureMode.SynapseMode:
                    FillSynapse(text);
                    break;

                case DataStructureMode.DendriteMode:
                    FillDendrite(text);
                    break;

                case DataStructureMode.OutColumnMode:
                    FillOutColumn(text);
                    break;

                case DataStructureMode.FeedMode:
                    FillFeed(text);
                    break;

                default:
                    break;
            }
            break;
    }
}

```

```

private void SetMode(string? text)
{
    BeginGetData();
    switch (text)
    {
        case InLayer:
            // Заполняем входной слой
            BrainFillerMode = BrainFillerMode.FillInLayer;
            break;

        case OutLayer:
            // Заполняем выходной слой
            BrainFillerMode = BrainFillerMode.FillOutLayer;
            break;

        case ActiveInLayer:
            // Заполняем активные клетки входного слоя
            BrainFillerMode = BrainFillerMode.FillActiveInLayer;
            break;

        case ActiveOutLayer:
            // Заполняем активные клетки выходного слоя
            BrainFillerMode = BrainFillerMode.FillActiveOutLayer;
            break;

        case PredictInLayer:
            // Заполняем предсказанные клетки входного слоя
            BrainFillerMode = BrainFillerMode.FillPredictInLayer;
            break;

        case FeedForward:
            // Заполняем связь входного слоя с выходным
            BrainFillerMode = BrainFillerMode.FillFeedForward;
            break;

        case FeedBack:
            // Заполняем связь выходного слоя с входным
            BrainFillerMode = BrainFillerMode.FillFeedBack;
            break;

        default:
            break;
    }
}

private void SetDataStructure(string? text)
{
    switch (text)
    {
        case MapBegin:

```

```
// Заполняем словарь/map/dictionary
// Это может быть не первая логическая структура, поэтому надо запомнить, что
заполняли ранее
```

```
    DataStructureStack.Push(DataMode);
    DataMode = DataStructureMode.MapMode;
    Map_CurrentLevel++;
    InitializeMap(Map_CurrentLevel);
    break;
```

```
case MapEnd:
    SaveMapValue();
    DataMode = DataStructureStack.Pop();
    Map_CurrentLevel--;
    break;
```

```
case ListBegin:
    DataStructureStack.Push(DataMode);
    DataMode = DataStructureMode.ListMode;
    InitializeList();
    break;
```

```
case ListEnd:
    SaveList();
    DataMode = DataStructureStack.Pop();
    break;
```

```
case SynapseBegin:
    DataStructureStack.Push(DataMode);
    DataMode = DataStructureMode.SynapseMode;
    Synapse = new();
    break;
```

```
case SynapseEnd:
    SaveSynapse();
    DataMode = DataStructureStack.Pop();
    break;
```

```
case DendriteBegin:
    DataStructureStack.Push(DataMode);
    DataMode = DataStructureMode.DendriteMode;
    ActiveDendrites = new();
    break;
```

```
case DendriteEnd:
    SaveDendrite();
    DataMode = DataStructureStack.Pop();
    break;
```

```

    case OutColumnBegin:
        DataStructureStack.Push(DataMode);
        DataMode = DataStructureMode.OutColumnMode;
        break;

    case OutColumnEnd:
        DataMode = DataStructureStack.Pop();
        break;

    case FeedBegin:
        DataStructureStack.Push(DataMode);
        DataMode = DataStructureMode.FeedMode;
        InitializeFeed();
        break;

    case FeedEnd:
        DataMode = DataStructureStack.Pop();
        break;

    case End:
        SendFilledStructure();
        BrainFillerMode = BrainFillerMode.Wait;
        EndGetData();
        break;

    default:
        break;
}
}

private void SendFilledStructure()
{
    switch (BrainFillerMode)
    {
        case BrainFillerMode.FillInLayer:
            brain.InLayer = Map_InLayer_Level_1;
            break;

        case BrainFillerMode.FillPredictInLayer:
            brain.PredictInLayer = Map_PredictInLayer_Level_1;
            break;

        case BrainFillerMode.FillActiveInLayer:
            brain.ActiveInLayer = Map_ActiveInLayer_Level_1;
            break;

        case BrainFillerMode.FillOutLayer:
            brain.OutLayer = Map_OutLayer_Level_1;

```

```

        break;

    case BrainFillerMode.FillActiveOutLayer:
        brain.ActiveOutLayer = ActiveCellOutLayer;
        break;

    case BrainFillerMode.FillFeedForward:
        brain.FeedForwardSynapses = FeedForwardSynapses;
        break;

    case BrainFillerMode.FillFeedBack:
        brain.FeedBackSynapses = FeedBackSynapses;
        break;

    default:
        break;
}
}

#region Initialize

private void InitializeFeed()
{
    FeedKey_1 = FeedKey_2 = FeedKey_3 = FeedKey_4 = null;
}

private void InitializeList()
{
    switch (BrainFillerMode)
    {
        case BrainFillerMode.FillPredictInLayer:
            ActiveLateralDendrites = [];
            break;

        case BrainFillerMode.FillActiveInLayer:
            ActiveCellsIds = [];
            break;

        case BrainFillerMode.FillActiveOutLayer:
            ActiveCellOutLayer = [];
            break;
    }
}

private void InitializeMap(int mapCurrentLevel)
{
    switch (BrainFillerMode)
    {
        case BrainFillerMode.FillInLayer:
            switch (Map_CurrentLevel)
            {

```

```

    case 1:
        Map_InLayer_Level_1 = [];
        break;
    case 2:
        Map_InLayer_Level_2 = [];
        break;
    case 3:
        Map_InLayer_Level_3 = [];
        break;
    case 4:
        Map_InLayer_Level_4 = [];
        break;
    default:
        break;
}
break;

case BrainFillerMode.FillPredictInLayer:
    switch (Map_CurrentLevel)
    {
        case 1:
            Map_PredictInLayer_Level_1 = [];
            break;
        case 2:
            Map_PredictInLayer_Level_2 = [];
            break;
        default:
            break;
    }
    break;

case BrainFillerMode.FillActiveInLayer:
    switch (Map_CurrentLevel)
    {
        case 1:
            Map_ActiveInLayer_Level_1 = [];
            break;
        default:
            break;
    }
    break;

case BrainFillerMode.FillOutLayer:
    switch (Map_CurrentLevel)
    {
        case 1:
            Map_OutLayer_Level_1 = [];
            break;
        case 2:
            Map_OutLayer_Level_2 = [];
            break;
    }

```



```

        case 3:
            Map_OutLayer_Level_3 = [];
            break;
        default:
            break;
    }
    break;

case BrainFillerMode.FillFeedForward:
    FeedForwardSynapses = [];
    break;

case BrainFillerMode.FillFeedBack:
    FeedBackSynapses = [];
    break;

    default:
        break;
}
}

#endregion

#region Fill

private void FillFeed(string? text)
{
    bool isNumber = int.TryParse(text, out var number);
    if (isNumber)
    {
        if (FeedKey_1 == null)
        {
            FeedKey_1 = number;
        }
        else
        {
            if (FeedKey_2 == null)
            {
                FeedKey_2 = number;
            }
            else
            {
                if (FeedKey_3 == null)
                {
                    FeedKey_3 = number;
                }
                else
                {
                    if (FeedKey_4 == null)
                    {
                        FeedKey_4 = number;

```

```

        }
    }
}
}
else
{
    SetDataStructure(text);
}
}

```

```

private void FillOutColumn(string? text)
{
    bool isCellGuid = int.TryParse(text, out var guid);
    if (isCellGuid)
    {
        CellGuid_OutLayer = guid;
    }
    else
    {
        SetDataStructure(text);
    }
}

```

```

private void FillList(string? text)
{
    switch (BrainFillerMode)
    {
        case BrainFillerMode.FillPredictInLayer:
            bool isGuid = int.TryParse(text, out var guid);
            if (isGuid)
            {
                ActiveLateralDendrites.Add(guid);
            }
            else
            {
                SetDataStructure(text);
            }
            break;

        case BrainFillerMode.FillActiveInLayer:
            bool isCellid = int.TryParse(text, out var cellId);
            if (isCellid)
            {
                ActiveCellsIds.Add(cellId);
            }
            else
            {
                SetDataStructure(text);
            }
    }
}

```

```

        break;

    case BrainFillerMode.FillActiveOutLayer:
        bool isActiveCellid = int.TryParse(text, out var activeCellId);
        if (isActiveCellid)
        {
            ActiveCellOutLayer.Add(activeCellId);
        }
        else
        {
            SetDataStructure(text);
        }
        break;
    }
}

private void FillDendrite(string? text)
{
    bool isApicalDendrite = int.TryParse(text, out var apicalDendrite);
    if (isApicalDendrite)
    {
        ActiveDendrites.ApicalDendrite = apicalDendrite;
    }
    else
    {
        if (text == False)
        {
            ActiveDendrites.ApicalDendrite = null;
        }
        else
        {
            SetDataStructure(text);
        }
    }
}

private void FillMap(string? text)
{
    // К нам пришли данные. Если распарсились - ключ.
    // Иначе - вложенная структура
    bool isNumber = int.TryParse(text, out var number);
    if (isNumber)
    {
        SaveMapKey(number);
    }
    else
    {
        SetDataStructure(text);
    }
}

```

```

private void FillSynapse(string? text)
{
    bool isSynapseId = int.TryParse(text, out var synapseId);
    if (isSynapseId)
    {
        Synapse.Id = synapseId;
    }
    else
    {
        // TODO Не кастится значение
        bool isPermanenceValue = float.TryParse(text, NumberStyles.Any,
CultureInfo.GetCultureInfo("fr-FR"), out var permanenceValue);
        if (isPermanenceValue)
        {
            Synapse.PermanenceValue = permanenceValue;
        }
        else
        {
            bool isWeight = bool.TryParse(text, out var weight);
            if (isWeight)
            {
                Synapse.Weight = weight;
            }
            else
            {
                SetDataStructure(text);
            }
        }
    }
}

#endregion

#region Save
private void SaveDendrite()
{
    switch (BrainFillerMode)
    {
        case BrainFillerMode.FillPredictInLayer:
            Map_PredictInLayer_Level_2.Add(MapKey_PredictInLayer_Level_2, ActiveDendrites);
            break;
    }
}

private void SaveSynapse()
{
    switch (BrainFillerMode)
    {
        case BrainFillerMode.FillInLayer:
            Map_InLayer_Level_4.Add(MapKey_InLayer_Level_4, Synapse);
            break;
    }
}

```

```

        case BrainFillerMode.FillOutLayer:
            Map_OutLayer_Level_3.Add(MapKey_OutLayer_Level_3, Synapse);
            break;

        case BrainFillerMode.FillFeedForward:
            FeedForwardSynapses.Add(((FeedKey_1, FeedKey_2), (FeedKey_3, FeedKey_4)),
Synapse);
            break;

        case BrainFillerMode.FillFeedBack:
            FeedBackSynapses.Add(((FeedKey_1, FeedKey_2), (FeedKey_3, FeedKey_4)), Synapse);
            break;
    }
}

private void SaveList()
{
    switch (BrainFillerMode)
    {
        case BrainFillerMode.FillPredictInLayer:
            ActiveDendrites.ActiveLateralDendrites = ActiveLateralDendrites;
            break;

        case BrainFillerMode.FillActiveInLayer:
            Map_ActiveInLayer_Level_1.Add(MapKey_ActiveInLayer_Level_1, ActiveCellsIds);
            break;
    }
}

private void SaveMapValue()
{
    switch (BrainFillerMode)
    {
        case BrainFillerMode.FillInLayer:
            switch (Map_CurrentLevel)
            {
                case 2:
                    Map_InLayer_Level_1.Add(MapKey_InLayer_Level_1, Map_InLayer_Level_2);
                    break;
                case 3:
                    Map_InLayer_Level_2.Add(MapKey_InLayer_Level_2, Map_InLayer_Level_3);
                    break;
                case 4:
                    Map_InLayer_Level_3.Add(MapKey_InLayer_Level_3, Map_InLayer_Level_4);
                    break;
                default:
                    break;
            }
            break;
    }
}

```

```

        case BrainFillerMode.FillPredictInLayer:
            switch (Map_CurrentLevel)
            {
                case 2:
                    Map_PredictInLayer_Level_1.Add(MapKey_PredictInLayer_Level_1,
Map_PredictInLayer_Level_2);
                    break;
                default:
                    break;
            }
            break;

        case BrainFillerMode.FillOutLayer:
            switch (Map_CurrentLevel)
            {
                case 2:
                    Map_OutLayer_Level_1.Add(MapKey_OutLayer_Level_1, (CellGuid_OutLayer,
Map_OutLayer_Level_2));
                    break;
                case 3:
                    Map_OutLayer_Level_2.Add(MapKey_OutLayer_Level_2, Map_OutLayer_Level_3);
                    break;
                default:
                    break;
            }
            break;

        default:
            break;
    }
}

private void SaveMapKey(int number)
{
    switch (BrainFillerMode)
    {
        case BrainFillerMode.FillInLayer:
            switch (Map_CurrentLevel)
            {
                case 1:
                    MapKey_InLayer_Level_1 = number;
                    break;
                case 2:
                    MapKey_InLayer_Level_2 = number;
                    break;
                case 3:
                    MapKey_InLayer_Level_3 = number;
                    break;
                case 4:
                    MapKey_InLayer_Level_4 = number;
                    break;
            }
        }
    }
}

```

```

        default:
            break;
    }
    break;

case BrainFillerMode.FillPredictInLayer:
    switch (Map_CurrentLevel)
    {
        case 1:
            MapKey_PredictInLayer_Level_1 = number;
            break;
        case 2:
            MapKey_PredictInLayer_Level_2 = number;
            break;
        default:
            break;
    }
    break;

case BrainFillerMode.FillActiveInLayer:
    switch (Map_CurrentLevel)
    {
        case 1:
            MapKey_ActiveInLayer_Level_1 = number;
            break;
        default:
            break;
    }
    break;

case BrainFillerMode.FillOutLayer:
    switch (Map_CurrentLevel)
    {
        case 1:
            MapKey_OutLayer_Level_1 = number;
            break;
        case 2:
            MapKey_OutLayer_Level_2 = number;
            break;
        case 3:
            MapKey_OutLayer_Level_3 = number;
            break;
        default:
            break;
    }
    break;

default:
    break;
}
}

```

```

        #endregion
    }
}
Модуль визуализации. Класс ClientOnServerSide.
using System.IO;
using System.Net.Sockets;

namespace ThousandBrainsVisualisation.Logic
{
    public class ClientOnServerSide
    {
        public Guid Id { get; }
        protected StreamWriter Writer { get; }
        protected StreamReader Reader { get; }
        protected Stream BinaryReader { get; }

        protected TcpClient Client { get; }
        protected Server Server { get; }

        public ClientOnServerSide(TcpClient tcpClient, Server serverObject)
        {
            Id = Guid.NewGuid();
            Client = tcpClient;
            Server = serverObject;
            BinaryReader = Client.GetStream();
            Reader = new StreamReader(BinaryReader);
            Writer = new StreamWriter(BinaryReader);
        }

        public async Task SendMessage(string? message)
        {
            await Writer.WriteLineAsync(message);
            await Writer.FlushAsync();
        }

        public async Task ProcessAsync()
        {
            try
            {
                string? binaryMessage;
                while (true)
                {
                    try
                    {
                        byte[] buffer = new byte[10240];
                        binaryMessage = await Reader.ReadLineAsync();
                        if (string.IsNullOrEmpty(binaryMessage))
                        {
                            continue;
                        }
                    }
                }
            }
        }
    }
}

```



```

        }
        Server.NextSymbol(binaryMessage);
        binaryMessage = null;
    }
    catch (Exception e)
    {
        //Server.NextSymbol(e.Message);
        break;
    }
}
}
catch (Exception e)
{
    // TODO написать обработчик с всплывающим окном или другим визуальным
отображением
    //Server.NextSymbol(e.Message);
}
finally
{
    Server.RemoveConnection(Id);
}
}
public void Close()
{
    Writer.Close();
    Reader.Close();
    Client.Close();
    BinaryReader.Close();
}
}
}
}

```

Модуль визуализации. Класс Painter.

```

using System.Drawing;
using System.Windows;
using System.Windows.Media.Imaging;
using ThousandBrainsVisualisation.Model;

namespace ThousandBrainsVisualisation.Logic
{
    public delegate void InCellsPaintedEventHandler();
    public delegate void OutCellsPaintedEventHandler();
    public delegate void FeedBackSynapsesPaintedEventHandler();
    public delegate void FeedForwardSynapsesPaintedEventHandler();
    public delegate void BasalDendritesPaintedEventHandler();
    public delegate void LocationSignalPaintedEventHandler();
    public delegate void SensorySignalPaintedEventHandler();

    public class Painter()
    {

```

```

    private static readonly SolidBrush UsuallyCellBrush = new(Color.Gray); // Тут есть синапс, но он
не активен
    private static readonly SolidBrush PotentialSynapseBrush = new(Color.LightGray); // Тут может
быть синапс
    private static readonly SolidBrush EmptyCellBrush = new(Color.WhiteSmoke); // Тут мтнапса
никогда возникнуть не сможет
    private static readonly SolidBrush PredictedCellBrush = new(Color.LightBlue);
    private static readonly SolidBrush ActiveCellBrush = new(Color.Coral);
    private static readonly int CellSize = 15;
    private static readonly int CellMargin = 12;
    private int ImageWidth => Brain.InLayer.Count * CellSize + CellMargin * 2;
    private int CellsInOneLineCount => (ImageWidth - CellMargin * 2) / CellSize;
    private int ImageDendriteHeight => (int)Math.Ceiling((decimal)Brain.LocationSignalSize /
CellsInOneLineCount) * CellSize + CellMargin * 2;
    private int ImageInLayerHeight => Brain.InLayer.FirstOrDefault().Value.Count * CellSize +
CellMargin * 2;
    private int ImageOutLayerHeight => (int)Math.Ceiling((decimal)Brain.OutLayer.Count /
CellsInOneLineCount) * CellSize + CellMargin * 2;

    private Tuple<int, int>? selectedInCell;
    private Tuple<int, int>? selectedOutCell;

    private readonly BrainModel brain;
    public BrainModel Brain => brain;

    #region Bitmaps
    private BitmapImage inLayerCells = new();
    public BitmapImage InLayerCells
    {
        get => inLayerCells;
        set
        {
            if (value != null)
            {
                inLayerCells = value;
                InCellsPainted();
            }
        }
    }

    private BitmapImage outLayerCells = new();
    public BitmapImage OutLayerCells
    {
        get => outLayerCells;
        set
        {
            if (value != null)
            {
                outLayerCells = value;
                OutCellsPainted();
            }
        }
    }

```

```
}  
}
```

```
private BitmapImage feedBackSynapsesBitmap = new();
```

```
public BitmapImage FeedBackSynapsesBitmap
```

```
{  
    get => feedBackSynapsesBitmap;  
    set  
    {  
        if (value != null)  
        {  
            feedBackSynapsesBitmap = value;  
            FeedBackSynapsesPainted();  
        }  
    }  
}
```

```
private BitmapImage feedForwardSynapsesBitmap = new();
```

```
public BitmapImage FeedForwardSynapsesBitmap
```

```
{  
    get => feedForwardSynapsesBitmap;  
    set  
    {  
        if (value != null)  
        {  
            feedForwardSynapsesBitmap = value;  
            FeedForwardSynapsesPainted();  
        }  
    }  
}
```

```
private BitmapImage feedForwardSignalBitmap = new();
```

```
public BitmapImage FeedForwardSignalBitmap
```

```
{  
    get => feedForwardSignalBitmap;  
    set  
    {  
        if (value != null)  
        {  
            feedForwardSignalBitmap = value;  
            SensorySignalPainted();  
        }  
    }  
}
```

```
private BitmapImage locationSignalBitmap = new();
```

```
public BitmapImage LocationSignalBitmap
```

```
{  
    get => locationSignalBitmap;  
    set  
    {
```

```

        if (value != null)
        {
            locationSignalBitmap = value;
            LocationSignalPainted();
        }
    }
}

private BitmapImage sensorySignalBitmap = new();
public BitmapImage SensorySignalBitmap
{
    get => sensorySignalBitmap;
    set
    {
        if (value != null)
        {
            sensorySignalBitmap = value;
            SensorySignalPainted();
        }
    }
}

private List<DendriteView> basalDendrites { get; set; } = [];
public List<DendriteView> BasalDendrites
{
    get => basalDendrites;
    set
    {
        if (value != null)
        {
            basalDendrites = value;
        }
    }
}
#endregion

#region ClickOnEllipse
public void SelectLocationSignalSynapse(int x, int y)
{
    int miniColumnKey = (x - CellMargin) / CellSize;
    int cellIndex = (y - CellMargin) / CellSize;
    int range = cellIndex * CellsInOneLineCount + miniColumnKey;

    // Если не получилось удалить, значит синапса там и не было. Поэтому добавляем
    if (!Brain.LocationSignal.Remove(range))
    {
        Brain.LocationSignal.Add(range);
    }
    DrawLocationSignal();
}

```

```

// TODO Написать функцию, преобразующую координаты в порядковый номер клетки
public void SelectSensorySignalMiniColumn(int x, int y)
{
    int miniColumnKey = (x - CellMargin) / CellSize;
    // TODO Painter не должен класть данные в модель
    // Если не получилось удалить, значит синапса там и не было. Поэтому добавляем
    if (!Brain.SensorySignal.Remove(miniColumnKey))
    {
        Brain.SensorySignal.Add(miniColumnKey);
    }
    DrawSensorySignal();
}

public void SelectInCell(int x, int y)
{
    int miniColumnKey = (x - CellMargin) / CellSize;
    int cellIndex = (y - CellMargin) / CellSize;
    BasalDendrites = [];
    if (Brain.InLayer.ContainsKey(miniColumnKey) && Brain.InLayer[miniColumnKey].Count >=
cellIndex)
    {
        // Этот замут нужен для того, чтобы обратиться к клетке не по ID, а по порядковому
номеру, который соответствует выводу на картинке
        // TODO не обрабатывается ситуация, если промазать и выйти за клетки
        var cell = Brain.InLayer[miniColumnKey].OrderBy(c =>
c.Key).ToList().ElementAt(cellIndex);
        selectedInCell = new Tuple<int, int>(miniColumnKey, cellIndex);
        DrawInCells();
        DrawFeedForwardSynapses(miniColumnKey, cell.Key);
        foreach (var dendrite in cell.Value.OrderBy(d => d.Key))
        {
            BasalDendrites.Add(
                new DendriteView()
                {
                    Info = GetDendriteInfo(miniColumnKey, cell.Key, dendrite.Key),
                    Image = GetDendriteImage(miniColumnKey, cell.Key, dendrite.Key, dendrite.Value)
                });
        }
        BasalDendritesPainted();
    }
}

public void SelectOutCell(int x, int y)
{
    int miniColumnKey = (x - CellMargin) / CellSize;
    int cellIndex = (y - CellMargin) / CellSize;
    int range = cellIndex * CellsInOneLineCount + miniColumnKey;
    if (Brain.OutLayer.ContainsKey(range))
    {
        selectedOutCell = new Tuple<int, int>(miniColumnKey, cellIndex);
        DrawOutCells();
    }
}

```

```

        DrawFeedBackSynapses(range);
    }
}
#endregion

#region Draw
// TODO На текущий момент механизм отрисовки не универсальный. Значения подобраны
так, чтобы корректно отображались 50 мини-колонок на экране 1920x1080
// TODO Обернуть в try/catch, потому что при неверной последовательности команд erlang
клеток может не быть
public void DrawInCells()
{
    Application.Current.Dispatcher.Invoke(() =>
    {
        using var bmp = new Bitmap(ImageWidth, ImageInLayerHeight);
        using var gfx = Graphics.FromImage(bmp);

        gfx.Clear(Color.White);

        int i = 0;
        foreach (var miniColumn in Brain.InLayer.OrderBy(c => c.Key))
        {
            int j = 0;
            foreach (var cell in miniColumn.Value.OrderBy(c => c.Key))
            {
                gfx.FillEllipse(
                    SelectBrushForInLayer(miniColumn.Key, cell.Key),
                    (CellSize * i) + CellMargin,
                    (CellSize * j) + CellMargin,
                    CellSize,
                    CellSize);
                j++;
            }
            i++;
        }
        DrawRisk(gfx);

        if (selectedInCell != null)
        {
            gfx.DrawEllipse(
                new Pen(Color.Black, 1),
                (CellSize * selectedInCell.Item1) + CellMargin,
                (CellSize * selectedInCell.Item2) + CellMargin,
                CellSize,
                CellSize);
        }

        InLayerCells = BitmapImageImageFromBitmap(bmp);
    });
}

```

```

public void DrawOutCells()
{
    Application.Current.Dispatcher.Invoke(() =>
    {
        using var bmp = new Bitmap(ImageWidth, ImageOutLayerHeight);
        using var gfx = Graphics.FromImage(bmp);

        gfx.Clear(Color.White);

        int i = 0;
        int j = 0;
        foreach (var miniColumn in Brain.OutLayer.OrderBy(c => c.Key))
        {
            if (i == CellsInOneLineCount)
            {
                i = 0;
                j++;
            }

            gfx.FillEllipse(
                SelectBrushForOutLayer(miniColumn.Key),
                (CellSize * i) + CellMargin,
                (CellSize * j) + CellMargin,
                CellSize,
                CellSize);
            i++;
        }
        DrawRisk(gfx);

        if (selectedOutCell != null)
        {
            gfx.DrawEllipse(
                new Pen(Color.Black, 1),
                (CellSize * selectedOutCell.Item1) + CellMargin,
                (CellSize * selectedOutCell.Item2) + CellMargin,
                CellSize,
                CellSize);
        }

        OutLayerCells = BitmapImageImageFromBitmap(bmp);
    });
}

```

```

private void DrawFeedBackSynapses(int range)
{
    Application.Current.Dispatcher.Invoke(() =>
    {

```

// ImageInLayerHeight - исходим из того, что нельзя нарисовать FeedBack до выходного

слоя

```

        using var bmp = new Bitmap(ImageWidth, ImageInLayerHeight);
        using var gfx = Graphics.FromImage(bmp);

```

```

gfx.Clear(Color.White);

var synapses = Brain.FeedBackSynapses.Where(f => f.Key.Item1.Item1 == range);

int i = 0;
foreach (var miniColumn in Brain.InLayer.OrderBy(c => c.Key))
{
    int j = 0;
    foreach (var cell in miniColumn.Value.OrderBy(c => c.Key))
    {
        gfx.FillEllipse(
            SelectBrushForFeed(synapses.Where(v => v.Key.Item2.Item1 == miniColumn.Key &&
v.Key.Item2.Item2 == cell.Key).FirstOrDefault().Value),
            (CellSize * i) + CellMargin,
            (CellSize * j) + CellMargin,
            CellSize,
            CellSize);
        j++;
    }
    i++;
}
DrawRisk(gfx);

FeedBackSynapsesBitmap = BitmapImageImageFromBitmap(bmp);
});
}

private void DrawFeedForwardSynapses(int miniColumnKey, int cellKey)
{
    Application.Current.Dispatcher.Invoke(() =>
    {
        int cellsCount = Brain.OutLayer.Count;
        // ImageOutLayerHeight - исходим из того, что нельзя нарисовать FeedForward до
ВЫХОДНОГО СЛОЯ
        using var bmp = new Bitmap(ImageWidth, ImageOutLayerHeight);
        using var gfx = Graphics.FromImage(bmp);

        gfx.Clear(Color.White);

        var synapses = Brain.FeedForwardSynapses.Where(f => f.Key.Item1.Item1 ==
miniColumnKey && f.Key.Item1.Item2 == cellKey);

        int i = 0;
        int j = 0;
        for (int s = 0; s < cellsCount; s++)
        {
            if (i == CellsInOneLineCount)
            {
                i = 0;
                j++;
            }
        }
    }
}

```



```

    }

    gfx.FillEllipse(
        SelectBrushForFeed(synapses.Where(v => v.Key.Item2.Item1 ==
s).FirstOrDefault().Value),
        (CellSize * i) + CellMargin,
        (CellSize * j) + CellMargin,
        CellSize,
        CellSize);
    i++;
}
DrawRisk(gfx);

FeedForwardSynapsesBitmap = BitmapImageImageFromBitmap(bmp);
});
}

public void DrawLocationSignal()
{
    Application.Current.Dispatcher.Invoke(() =>
    {
        using var bmp = new Bitmap(ImageWidth, ImageDendriteHeight);
        using var gfx = Graphics.FromImage(bmp);

        gfx.Clear(Color.White);

        int i = 0;
        int j = 0;
        for (int s = 0; s < Brain.LocationSignalSize; s++)
        {
            if (i == CellsInOneLineCount)
            {
                i = 0;
                j++;
            }

            gfx.FillEllipse(
                SelectBrushForLocationSignal(j * CellsInOneLineCount + i),
                (CellSize * i) + CellMargin,
                (CellSize * j) + CellMargin,
                CellSize,
                CellSize);
            i++;
        }
        DrawRisk(gfx);

        LocationSignalBitmap = BitmapImageImageFromBitmap(bmp);
    });
}

```

```

public void DrawSensorySignal()
{
    Application.Current.Dispatcher.Invoke(() =>
    {
        using var bmp = new Bitmap(ImageWidth, CellMargin * 2 + CellSize);
        using var gfx = Graphics.FromImage(bmp);

        gfx.Clear(Color.White);

        for (int s = 0; s < Brain.InLayer.Count; s++)
        {
            gfx.FillEllipse(
                SelectBrushForSensorySignal(s),
                (CellSize * s) + CellMargin,
                CellMargin,
                CellSize,
                CellSize);
        }
        DrawRisk(gfx);

        SensorySignalBitmap = BitmapImageImageFromBitmap(bmp);
    });
}

private void DrawRisk(Graphics g)
{
    for (int i = 1; i < CellsInOneLineCount / 5; i++)
    {
        g.DrawLine(new Pen(Color.Gray, 1), CellMargin + CellSize * 5 * i, 8, CellMargin + CellSize *
5 * i, 13);
    }
}
#endregion

#region Dendrite Handle
private BitmapImage GetDentriteImage(int miniColumnKey, int cellKey, int dendriteKey,
Dictionary<int, Synapse> dendrite)
{
    ;
    using var bmp = new Bitmap(ImageWidth, ImageDendriteHeight);
    using var gfx = Graphics.FromImage(bmp);

    gfx.Clear(Color.White);

    int i = 0;
    int j = 0;
    for (int s = 0; s < Brain.LocationSignalSize; s++)
    {
        if (i == CellsInOneLineCount)
        {
            i = 0;

```

```

        j++;
    }

    dendrite.TryGetValue(s, out Synapse? synapse);
    gfx.FillEllipse(
        SelectBrushForDendrite(synapse, s),
        (CellSize * i) + CellMargin,
        (CellSize * j) + CellMargin,
        CellSize,
        CellSize);
    i++;
}
DrawRisk(gfx);

// Todo Разобраться с повторяющимся вызовом
if (Brain.HasActiveLateralDendrite(miniColumnKey, cellKey, dendriteKey))
{
    gfx.DrawRectangle(
        new Pen(Color.Green, 3),
        1, 1, ImageWidth - 3, ImageDendriteHeight - 3);
}

return BitmapImageImageFromBitmap(bmp);
}

private string GetDendriteInfo(int miniColumnKey, int cellKey, int dendriteKey)
{
    string result = string.Empty;

    result += "Активность: " + (Brain.HasActiveLateralDendrite(miniColumnKey, cellKey,
dendriteKey) ? "Да " : "Нет ");
    result += "Количество существующих синапсов: " +
Brain.GetExistSynapseCountInDendrite(miniColumnKey, cellKey, dendriteKey) + " ";
    result += "Количество активных синапсов: " +
Brain.GetActiveSynapseCountInDendrite(miniColumnKey, cellKey, dendriteKey) + " ";

    return result;
}
#endregion

#region SelectBrush
private SolidBrush SelectBrushForFeed(Synapse? synapse)
{
    if (synapse == null)
    {
        return EmptyCellBrush;
    }
    else
    {
        if (synapse.Weight)
        {

```

```

        return UsuallyCellBrush;
    }
    else
    {
        return PotentialSynapseBrush;
    }
}

private SolidBrush SelectBrushForInLayer(int miniColumnKey, int cellKey)
{
    if (Brain.IsActiveCellInLayer(miniColumnKey, cellKey))
    {
        return ActiveCellBrush;
    }
    else
    {
        if (Brain.IsPredictCellInLayer(miniColumnKey, cellKey))
        {
            return PredictedCellBrush;
        }
        else
        {
            return UsuallyCellBrush;
        }
    }
}

private SolidBrush SelectBrushForDendrite(Synapse? synapse, int indexInLocation)
{
    if (synapse == null)
    {
        return EmptyCellBrush;
    }
    if (synapse.Weight)
    {
        if (Brain.LocationSignal.Contains(indexInLocation))
        {
            return ActiveCellBrush;
        }
        else
        {
            return UsuallyCellBrush;
        }
    }
    else
    {
        return PotentialSynapseBrush;
    }
}

```

```

    private SolidBrush SelectBrushForLocationSignal(int range) =>
Brain.LocationSignal.Contains(range) ? ActiveCellBrush : UsuallyCellBrush;

    private SolidBrush SelectBrushForSensorySignal(int range) =>
Brain.SensorySignal.Contains(range) ? ActiveCellBrush : UsuallyCellBrush;

    private SolidBrush SelectBrushForOutLayer(int miniColumnKey) =>
Brain.IsActiveCellOutLayer(miniColumnKey) ? ActiveCellBrush : UsuallyCellBrush;
    #endregion

    private static BitmapImage BitmapImageImageFromBitmap(Bitmap bmp)
    {
        using var memory = new System.IO.MemoryStream();

        bmp.Save(memory, System.Drawing.Imaging.ImageFormat.Png);
        memory.Position = 0;

        var bitmapImage = new BitmapImage();
        bitmapImage.BeginInit();
        bitmapImage.StreamSource = memory;
        bitmapImage.CacheOption = BitmapCacheOption.OnLoad;
        bitmapImage.EndInit();
        bitmapImage.Freeze();

        return bitmapImage;
    }

    public event InCellsPaintedEventHandler InCellsPainted = delegate { };
    public event OutCellsPaintedEventHandler OutCellsPainted = delegate { };
    public event FeedBackSynapsesPaintedEventHandler FeedBackSynapsesPainted = delegate { };
    public event FeedForwardSynapsesPaintedEventHandler FeedForwardSynapsesPainted = delegate { };
};

    public event BasalDendritesPaintedEventHandler BasalDendritesPainted = delegate { };
    public event LocationSignalPaintedEventHandler LocationSignalPainted = delegate { };
    public event SensorySignalPaintedEventHandler SensorySignalPainted = delegate { };

    public Painter(BrainModel brain) : this()
    {
        this.brain = brain;
    }
}

```

Модуль визуализации. Класс Server.

```
using System.Net;
```

```
using System.Net.Sockets;
```

```
namespace ThousandBrainsVisualisation.Logic
```

```
{
    public delegate void SendDataToApplicationEventHandler(string? text);
    public delegate void ChangeConnectionsCountEventHandler(int count);
    public class Server
```

```

{
    public Server()
    {
        TcpListener = new TcpListener(IPAddress.Any, 8888);
        Clients = [];
    }

    public event SendDataToApplicationEventHandler SendDataToApplication = delegate { };
    public event ChangeConnectionsCountEventHandler ChangeConnectionsCount = delegate { };

    protected TcpListener TcpListener { get; set; }
    protected IList<ClientOnServerSide> Clients { get; set; }

    public void RemoveConnection(Guid id)
    {
        ClientOnServerSide? client = Clients.FirstOrDefault(c => c.Id == id);
        if (client != null)
        {
            Clients.Remove(client);
        }
        client?.Close();
        ChangeConnectionsCount(Clients.Count);
    }

    public void Disconnect()
    {
        foreach (var client in Clients)
        {
            client.Close();
        }
        TcpListener.Stop();
    }

    public async Task ListenAsync()
    {
        try
        {
            TcpListener.Start();
            Console.WriteLine("Сервер запущен. Ожидание подключений...");

            while (true)
            {
                TcpClient tcpClient = await TcpListener.AcceptTcpClientAsync();

                ClientOnServerSide clientObject = new(tcpClient, this);
                Clients.Add(clientObject);
                ChangeConnectionsCount(Clients.Count);
                Task.Run(clientObject.ProcessAsync);
            }
        }
        catch (Exception ex)
    }

```

```

        {
            Console.WriteLine(ex.Message);
        }
        finally
        {
            Disconnect();
        }
    }

    public async Task SendDataToClient(string? data)
    {
        foreach(var client in Clients)
        {
            await client.SendMessage(data);
        }
    }

    public void NextSymbol(string? text)
    {
        SendDataToApplication(text);
    }
}

```

Модуль визуализации. Класс BrainCommands.

namespace ThousandBrainsVisualisation.Model

```

{
    public class BrainCommands
    {
        public const string LocationSignalBegin = "LocationSignalBegin";
        public const string LocationSignalEnd = "LocationSignalEnd";
        public const string SensorySignalBegin = "SensorySignalBegin";
        public const string SensorySignalEnd = "SensorySignalEnd";
        public const string NeedBrainInitialize = "NeedBrainInitialize";
        public const string NeedBrainPrint = "NeedBrainPrint";
    }
}

```

Модуль визуализации. Класс BrainFiller.

namespace ThousandBrainsVisualisation.BrainFillerNS

```

{
    public partial class BrainFiller
    {
        private const string InLayer = "InLayer";
        private const string OutLayer = "OutLayer";
        private const string ActiveInLayer = "ActiveInLayer";
        private const string ActiveOutLayer = "ActiveOutLayer";
        private const string PredictInLayer = "PredictInLayer";
        private const string FeedForward = "FeedForward";
        private const string FeedBack = "FeedBack";
        private const string MapBegin = "MapBegin";
    }
}

```

```

private const string MapEnd = "MapEnd";
private const string ListBegin = "ListBegin";
private const string ListEnd = "ListEnd";
private const string DendriteBegin = "DendriteBegin";
private const string DendriteEnd = "DendriteEnd";
private const string SynapseBegin = "SynapseBegin";
private const string SynapseEnd = "SynapseEnd";
private const string OutColumnBegin = "OutColumnBegin";
private const string OutColumnEnd = "OutColumnEnd";
private const string FeedBegin = "FeedBegin";
private const string FeedEnd = "FeedEnd";
private const string End = "END";
private const string False = "false";
}
}

```

Модуль визуализации. Класс BrainFillerMode.

```
namespace ThousandBrainsVisualisation.Model
```

```

{
    public enum BrainFillerMode
    {
        Wait,
        FillInLayer,
        FillOutLayer,
        FillActiveInLayer,
        FillActiveOutLayer,
        FillPredictInLayer,
        FillFeedForward,
        FillFeedBack
    }
}

```

Модуль визуализации. Класс BrainModel.

```
namespace ThousandBrainsVisualisation.Model
```

```

{
    public delegate void UpdateInCellsEventHandler();
    public delegate void UpdateOutCellsEventHandler();
    public delegate void SendLocationSignalEventHandler();
    public delegate void SendSensorySignalEventHandler();
    public delegate void SetNeedBrainInitializeEventHandler();
    public delegate void SetNeedBrainPrintEventHandler();

    public class BrainModel
    {
        #region Model
        private Dictionary<int, Dictionary<int, Dictionary<int, Dictionary<int, Synapse>>>>> inLayer;
        public Dictionary<int, Dictionary<int, Dictionary<int, Dictionary<int, Synapse>>>>> InLayer
        {
            get => inLayer;
            set
            {

```



```

        if (value != null)
        {
            inLayer = value;
            UpdateInCells();
        };
    }
}

```

```

private Dictionary<int, Dictionary<int, Dendrites>> predictInLayer;
public Dictionary<int, Dictionary<int, Dendrites>> PredictInLayer
{
    get => predictInLayer;
    set
    {
        if (value != null)
        {
            predictInLayer = value;
            UpdateInCells();
        };
    }
}

```

```

private Dictionary<int, List<int>> activeInLayer;
public Dictionary<int, List<int>> ActiveInLayer
{
    get => activeInLayer;
    set
    {
        if (value != null)
        {
            activeInLayer = value;
            UpdateInCells();
        };
    }
}

```

```

private Dictionary<int, (int, Dictionary<int, Dictionary<int, Synapse>>)> outLayer;
public Dictionary<int, (int, Dictionary<int, Dictionary<int, Synapse>>)> OutLayer
{
    get => outLayer;
    set
    {
        if (value != null)
        {
            outLayer = value;
            UpdateOutCells();
        };
    }
}

```

```
}
```

```
private List<int> activeOutLayer;  
public List<int> ActiveOutLayer  
{  
    get => activeOutLayer;  
    set  
    {  
        if (value != null)  
        {  
            activeOutLayer = value;  
            UpdateOutCells();  
        };  
    }  
}
```

```
private Dictionary<((int?, int?), (int?, int?)), Synapse> feedForwardSynapses;  
public Dictionary<((int?, int?), (int?, int?)), Synapse> FeedForwardSynapses  
{  
    get => feedForwardSynapses;  
    set  
    {  
        if (value != null)  
        {  
            feedForwardSynapses = value;  
        };  
    }  
}
```

```
private Dictionary<((int?, int?), (int?, int?)), Synapse> feedBackSynapses;  
public Dictionary<((int?, int?), (int?, int?)), Synapse> FeedBackSynapses  
{  
    get => feedBackSynapses;  
    set  
    {  
        if (value != null)  
        {  
            feedBackSynapses = value;  
        };  
    }  
}
```

```
#endregion Data
```

```
#region ExternalData  
private List<int> locationSignal = [];  
public List<int> LocationSignal  
{  
    get => locationSignal;
```

```

    set
    {
        if (value != null)
        {
            locationSignal = value;
        };
    }
}

private List<int> sensorySignal = [];
public List<int> SensorySignal
{
    get => sensorySignal;
    set
    {
        if (value != null)
        {
            sensorySignal = value;
        };
    }
}
#endregion

#region VM data
private bool needBrainInitialize = false;
public bool NeedBrainInitialize
{
    get => needBrainInitialize;
    set
    {
        needBrainInitialize = value;
        if (value == true)
        {
            SetNeedBrainInitialize();
        }
    }
}

private bool needSendLocationSignal = false;
public bool NeedSendLocationSignal
{
    get => needSendLocationSignal;
    set
    {
        needSendLocationSignal = value;
        if (value == true)
        {
            SendLocationSignal();
        }
    }
}

```

```
}
```

```
private bool needSendSensorySignal = false;
public bool NeedSendSensorySignal
{
    get => needSendSensorySignal;
    set
    {
        needSendSensorySignal = value;
        if (value == true)
        {
            SendSensorySignal();
        }
    }
}
```

```
private bool needBrainPrint = false;
public bool NeedBrainPrint
{
    get => needBrainPrint;
    set
    {
        needBrainPrint = value;
        if (value == true)
        {
            SetNeedBrainPrint();
        }
    }
}
#endregion
```

```
#region Checkers
```

```
public bool HasActiveLateralDendrite(int miniColumnKey, int cellKey, int dendriteKey) =>
PredictInLayer.ContainsKey(miniColumnKey)
                                &&
PredictInLayer[miniColumnKey].ContainsKey(cellKey)
                                &&
PredictInLayer[miniColumnKey][cellKey].ActiveLateralDendrites.Contains(dendriteKey);
```

```
//TODO Добавить условия
```

```
public int GetExistSynapseCountInDendrite(int miniColumnKey, int cellKey, int dendriteKey) =>
InLayer[miniColumnKey][cellKey][dendriteKey].Where(s => s.Value.Weight == true).Count();
public int GetActiveSynapseCountInDendrite(int miniColumnKey, int cellKey, int dendriteKey) =>
InLayer[miniColumnKey][cellKey][dendriteKey].Where(s => s.Value.Weight == true &&
LocationSignal.Contains(s.Key)).Count();
```

```
public bool IsActiveCellInLayer(int miniColumnKey, int cellKey) =>
ActiveInLayer.ContainsKey(miniColumnKey) && ActiveInLayer[miniColumnKey].Contains(cellKey);
```

```

    public bool IsPredictCellInLayer(int miniColumnKey, int cellKey) =>
PredictInLayer.ContainsKey(miniColumnKey) &&
PredictInLayer[miniColumnKey].ContainsKey(cellKey);
    public bool IsActiveCellOutLayer(int miniColumnKey) =>
ActiveOutLayer.Contains(miniColumnKey);
    #endregion

    //TODO Реализовать передачу настроек мозга по TCP
    public int LocationSignalSize = 200;

    public event UpdateInCellsEventHandler UpdateInCells = delegate { };
    public event UpdateOutCellsEventHandler UpdateOutCells = delegate { };
    public event SendLocationSignalEventHandler SendLocationSignal = delegate { };
    public event SendSensorySignalEventHandler SendSensorySignal = delegate { };
    public event SetNeedBrainInitializeEventHandler SetNeedBrainInitialize = delegate { };
    public event SetNeedBrainInitializeEventHandler SetNeedBrainPrint = delegate { };

    public BrainModel()
    {
        inLayer = [];
        predictInLayer = [];
        activeInLayer = [];
        outLayer = [];
        activeOutLayer = [];
        feedForwardSynapses = [];
        feedBackSynapses = [];
        LocationSignal = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19];
        SensorySignal = [];
    }
}

```

Модуль визуализации. Класс DataStructureMode.

```

namespace ThousandBrainsVisualisation.Model
{
    public enum DataStructureMode
    {
        None,
        MapMode,
        ListMode,
        DendriteMode,
        SynapseMode,
        OutColumnMode,
        FeedMode
    }
}

```

Модуль визуализации. Класс Dendrites.

```

namespace ThousandBrainsVisualisation.Model
{
    public class Dendrites

```

```

{
    public Dendrites()
    {
        ActiveLateralDendrites = [];
    }

    public int? ApicalDendrite { get; set; }
    public List<int> ActiveLateralDendrites { get; set; }
}
}

```

Модуль визуализации. Класс DendriteView.
using System.Windows.Media.Imaging;

```

namespace ThousandBrainsVisualisation.Model
{
    public class DendriteView
    {
        public string Info { get; set; }
        public BitmapImage Image { get; set; }
    }
}

```

Модуль визуализации. Класс EditSignalMode.

```

namespace ThousandBrainsVisualisation.Model
{
    public enum EditSignalMode
    {
        LocationSignal,
        SensorySignal
    }
}

```

Модуль визуализации. Класс Synapse.

```

namespace ThousandBrainsVisualisation.Model
{
    public class Synapse
    {
        public int Id { get; set; }
        public float PermanenceValue { get; set; }
        public bool Weight { get; set; }

        public Synapse() { }
    }
}

```

Модуль визуализации. Класс EditSignalWindow.

```

<Window x:Class="ThousandBrainsVisualisation.View.EditSignalWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

```

```

xmlns:viewmodel="clr-namespace:ThousandBrainsVisualisation.ViewModel"
d:DataContext="{d:DesignInstance Type=viewmodel:EditSignalViewModel}"
mc:Ignorable="d"
Title="Редактирование сигнала местоположения" SizeToContent="WidthAndHeight"
Icon="..\Resources/Icon.png">
<Grid>
<StackPanel Grid.Column="0" Margin="10 10 10 10">
<Border Background="GhostWhite" BorderBrush="Gainsboro" BorderThickness="1" >
<Border.Effect>
<DropShadowEffect Opacity="0.3" Color="Black" BlurRadius="10"/>
</Border.Effect>
<Image x:Name="SignalView" MouseDown="SignalViewMouseDown" Source="{Binding
SignalBitmap}" Stretch="None"></Image>
</Border>
<!--<StackPanel Grid.Column="0" Margin="10 10 0 10" Orientation="Horizontal"
HorizontalAlignment="Right">
<Button x:Name="CancelButton" Content="Отменить" Margin="10 0 0 0"
HorizontalAlignment="Right" VerticalAlignment="Top" Width="75" />
<Button x:Name="SaveButton" Content="Сохранить" Margin="10 0 0 0"
HorizontalAlignment="Right" VerticalAlignment="Top" Width="75" />
<Button x:Name="SaveAndSendButton" Content="Сохранить и отправить" Margin="10 0 0
0" HorizontalAlignment="Right" VerticalAlignment="Top" Width="140" />
</StackPanel-->
</StackPanel>
</Grid>
</Window>

```

Модуль визуализации. Класс EditSignalWindow.

```
using System.Windows;
```

```
using ThousandBrainsVisualisation.ViewModel;
```

```
namespace ThousandBrainsVisualisation.View
```

```
{
```

```
/// <summary>
```

```
/// Логика взаимодействия для EditLocationSignal.xaml
```

```
/// </summary>
```

```
public partial class EditSignalWindow : Window
```

```
{
```

```
public EditSignalWindow()
```

```
{
```

```
InitializeComponent();
```

```
}
```

```
private void SignalViewMouseDown(object sender, System.Windows.Input.MouseButtonEventArgs
```

```
e)
```

```
{
```

```
Point p = e.GetPosition(SignalView);
```

```
((EditSignalViewModel)DataContext).SelectSignalSynapse((int)p.X, (int)p.Y);
```

```
}
```

```
}
```

```
}
```

Модуль визуализации. Класс MainWindow.

```
<Window x:Class="ThousandBrainsVisualisation.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:local="clr-namespace:ThousandBrainsVisualisation.Converters"
  xmlns:viewmodel="clr-namespace:ThousandBrainsVisualisation.ViewModel"
  d:DataContext="{d:DesignInstance Type=viewmodel:MainWindowViewModel}"
  mc:Ignorable="d"
  Title="ThousandBrains" Height="845" Width="1700" Icon=" ../Resources/Icon.png">

  <Window.Resources>
    <Style TargetType="{x:Type Button}" x:Key="ImageButtonStyle">
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="{x:Type Button}">
            <ContentPresenter/>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>

    <local:BoolToVisibilityConverter x:Key="BoolToVisibilityConverter"/>
    <local:BrushColorConverter x:Key="ColorConverter"/>
  </Window.Resources>

  <Grid>
    <StackPanel>
      <Menu Height="25" VerticalAlignment="Top">
        <MenuItem Header="Управление">
          <MenuItem Header="Кортикальная колонка" >
            <MenuItem Header="Инициализация" Command="{Binding
BrainInitializeCommand}"></MenuItem>
            <MenuItem Header="Вывод в файл" Command="{Binding
BrainPrintCommand}"></MenuItem>
          </MenuItem>
          <MenuItem Header="Сигнал местоположения" >
            <MenuItem Header="Редактировать" Command="{Binding
EditLocationSignalCommand}"></MenuItem>
            <MenuItem Header="Отправить" Command="{Binding
SendLocationSignalCommand}"></MenuItem>
          </MenuItem>
          <MenuItem Header="Сенсорный сигнал" >
            <MenuItem Header="Редактировать" Command="{Binding
EditSensorySignalCommand}"></MenuItem>
            <MenuItem Header="Отправить" Command="{Binding
SendSensorySignalCommand}"></MenuItem>
          </MenuItem>
          <Separator />
        </MenuItem>
      </Menu>
    </StackPanel>
  </Grid>
```



```

    <MenuItem Header="Выход" ></MenuItem>
</MenuItem>
</Menu>

<Grid x:Name="MainGrid">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>

    <StackPanel Grid.Column="0" Margin="10 10 10 10">

        <StackPanel Orientation="Horizontal" Margin="0 0 0 10">
            <TextBlock Margin="0 0 10 0">Активное подключение</TextBlock>

            <Ellipse Width="10" Height="10" HorizontalAlignment="Left">
                <Ellipse.Fill>
                    <SolidColorBrush Color="{Binding HasActiveConnection,
Converter={StaticResource ColorConverter}}" />
                </Ellipse.Fill>
            </Ellipse>
        </StackPanel>

        <TextBlock>Входной слой</TextBlock>

        <Border Background="GhostWhite" BorderBrush="Gainsboro" BorderThickness="1" >
            <Border.Effect>
                <DropShadowEffect Opacity="0.3" Color="Black" BlurRadius="10"/>
            </Border.Effect>
            <Image x:Name="InLayerView" MouseDown="InLayerViewMouseDown"
Source="{Binding InLayerCells}" Stretch="None"></Image>
        </Border>

        <TextBlock>FeedBack (Апикальный дендрит)</TextBlock>

        <Border Background="GhostWhite" BorderBrush="Gainsboro" BorderThickness="1" >
            <Border.Effect>
                <DropShadowEffect Opacity="0.3" Color="Black" BlurRadius="10"/>
            </Border.Effect>
            <Image x:Name="FeedBackView" Source="{Binding FeedBackSynapsesBitmap}"
Stretch="None"></Image>
        </Border>

        <TextBlock>FeedForward</TextBlock>

        <Border Background="GhostWhite" BorderBrush="Gainsboro" BorderThickness="1" >
            <Border.Effect>
                <DropShadowEffect Opacity="0.3" Color="Black" BlurRadius="10"/>
            </Border.Effect>
            <Image x:Name="FeedForwardView" Source="{Binding FeedForwardSynapsesBitmap}"
Stretch="None"></Image>

```

```

</Border>

<TextBlock>Выходной слой</TextBlock>

<Border Background="GhostWhite" BorderBrush="Gainsboro" BorderThickness="1" >
    <Border.Effect>
        <DropShadowEffect Opacity="0.3" Color="Black" BlurRadius="10"/>
    </Border.Effect>
    <Image x:Name="OutLayerView" MouseDown="OutLayerViewMouseDown"
Source="{Binding OutLayerCells}" Stretch="None"></Image>
    </Border>
</StackPanel>

<StackPanel Grid.Column="1" Margin="10 10 10 10">

    <TextBlock>Базальные дендриты</TextBlock>

    <Border Background="GhostWhite" BorderBrush="Gainsboro" BorderThickness="1" >
        <Border.Effect>
            <DropShadowEffect Opacity="0.3" Color="Black" BlurRadius="10"/>
        </Border.Effect>
        <StackPanel>
            <ListView x:Name="Dendrites" ItemsSource="{Binding BasalDendrites}"
Height="700" ScrollViewer.VerticalScrollBarVisibility="Visible"
HorizontalContentAlignment="Center">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <StackPanel>
                            <TextBlock Text="{Binding Path=Info}"/>
                            <Image Source="{Binding Path=Image}"/>
                        </StackPanel>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </StackPanel>
    </Border>
</StackPanel>
</Grid>
</StackPanel>

<Border BorderBrush="Black" BorderThickness="1" Background="#80000000"
Visibility="{Binding Path=IsBusy, Converter={StaticResource BoolToVisibilityConverter}}">
    <Grid>
        <TextBlock Margin="0" TextWrapping="Wrap" Text="Обмен данными..."
HorizontalAlignment="Center" VerticalAlignment="Center" FontSize="20" FontWeight="Bold"
Foreground="#7EFFFFFF"/>
    </Grid>
</Border>
</Grid>
</Window>

```

Модуль визуализации. Класс MainWindow.

```
using System.Windows;
using ThousandBrainsVisualisation.ViewModel;
```

```
namespace ThousandBrainsVisualisation
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

            private void InLayerViewMouseDown(object sender,
System.Windows.Input.MouseButtonEventArgs e)
            {
                Point p = e.GetPosition(InLayerView);
                ((MainWindowViewModel)DataContext).SelectInCell((int)p.X, (int)p.Y);
            }

            private void OutLayerViewMouseDown(object sender,
System.Windows.Input.MouseButtonEventArgs e)
            {
                Point p = e.GetPosition(OutLayerView);
                ((MainWindowViewModel)DataContext).SelectOutCell((int)p.X, (int)p.Y);
            }
        }
    }
}
```

Модуль визуализации. Класс BaseViewModel.

```
using System.ComponentModel;
using System.Runtime.CompilerServices;
```

```
namespace ThousandBrainsVisualisation.ViewModel
{
    public class BaseViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = "")
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Модуль визуализации. Класс EditSignalViewModel.

```
using System.Windows.Media.Imaging;
using ThousandBrainsVisualisation.Logic;
```

```

using ThousandBrainsVisualisation.Model;

namespace ThousandBrainsVisualisation.ViewModel
{
    class EditSignalViewModel : BaseViewModel
    {
        public readonly BrainModel Brain;
        private readonly Painter Painter;

        public EditSignalMode Mode { get; set; }

        private BitmapImage signalBitmap = new();
        public BitmapImage SignalBitmap
        {
            get => signalBitmap;
            set
            {
                if (value != null)
                {
                    signalBitmap = value;
                    OnPropertyChanged();
                }
            }
        }

        private List<int> signal = [];
        public List<int> Signal
        {
            get => signal;
            set
            {
                if (value != null)
                {
                    signal = value;
                }
            }
        }

        public void SelectSignalSynapse(int x, int y)
        {
            switch (Mode)
            {
                case EditSignalMode.LocationSignal:
                    Painter.SelectLocationSignalSynapse(x, y);
                    break;

                case EditSignalMode.SensorySignal:
                    Painter.SelectSensorySignalMiniColumn(x, y);
                    break;
            }
        }
    }
}

```

```

public EditSignalViewModel(BrainModel brain, EditSignalMode mode)
{
    Brain = brain;
    Painter = new(brain);

    Mode = mode;

    switch (Mode)
    {
        case EditSignalMode.LocationSignal:
            Painter.LocationSignalPainted += () => SignalBitmap = Painter.LocationSignalBitmap;
            Painter.DrawLocationSignal();
            break;

        case EditSignalMode.SensorySignal:
            Painter.SensorySignalPainted += () => SignalBitmap = Painter.SensorySignalBitmap;
            Painter.DrawSensorySignal();
            break;
    }
}
}

```

Модуль визуализации. Класс MainWindowViewModel.

```

using System.Windows.Media.Imaging;
using ThousandBrainsVisualisation.Logic;
using ThousandBrainsVisualisation.Model;
using ThousandBrainsVisualisation.View;

namespace ThousandBrainsVisualisation.ViewModel
{
    public class MainWindowViewModel : BaseViewModel
    {
        public readonly BrainModel Brain;
        private readonly Painter Painter;

        private bool isBusy = false;
        public bool IsBusy
        {
            get => isBusy;
            set
            {
                isBusy = value;
                OnPropertyChanged();
            }
        }

        private bool hasActiveConnection = false;
        public bool HasActiveConnection
        {

```

```

    get => hasActiveConnection;
    set
    {
        hasActiveConnection = value;
        OnPropertyChanged();
    }
}

```

```

#region Bitmaps
private BitmapImage inLayerCells = new();
public BitmapImage InLayerCells
{
    get => inLayerCells;
    set
    {
        if (value != null)
        {
            inLayerCells = value;
            OnPropertyChanged();
        }
    }
}

```

```

private BitmapImage outLayerCells = new();
public BitmapImage OutLayerCells
{
    get => outLayerCells;
    set
    {
        if (value != null)
        {
            outLayerCells = value;
            OnPropertyChanged();
        }
    }
}

```

```

private BitmapImage feedBackSynapsesBitmap = new();
public BitmapImage FeedBackSynapsesBitmap
{
    get => feedBackSynapsesBitmap;
    set
    {
        if (value != null)
        {
            feedBackSynapsesBitmap = value;
            OnPropertyChanged();
        }
    }
}

```

```

}

private BitmapImage feedForwardSynapsesBitmap = new();
public BitmapImage FeedForwardSynapsesBitmap
{
    get => feedForwardSynapsesBitmap;
    set
    {
        if (value != null)
        {
            feedForwardSynapsesBitmap = value;
            OnPropertyChanged();
        }
    }
}

private List<DendriteView> basalDendrites { get; set; } = [];
public List<DendriteView> BasalDendrites
{
    get => basalDendrites;
    set
    {
        if (value != null)
        {
            basalDendrites = value;
            OnPropertyChanged();
        }
    }
}
#endregion

public void SelectInCell(int x, int y)
{
    Painter.SelectInCell(x, y);
}

public void SelectOutCell(int x, int y)
{
    Painter.SelectOutCell(x, y);
}

private void SendLocationSignal()
{
    IsBusy = true;
    Brain.NeedSendLocationSignal = true;
}

private void SendSensorySignal()
{
    IsBusy = true;
    Brain.NeedSendSensorySignal = true;
}

```

```

}

#region Commands
private RelayCommand? brainInitializeCommand;
public RelayCommand BrainInitializeCommand
{
    get
    {
        return brainInitializeCommand ??= new RelayCommand(obj =>
        {
            IsBusy = true;
            Brain.NeedBrainInitialize = true;
        });
    }
}

private RelayCommand? brainPrintCommand;
public RelayCommand BrainPrintCommand
{
    get
    {
        return brainPrintCommand ??= new RelayCommand(obj =>
        {
            //IsBusy = true;
            Brain.NeedBrainPrint = true;
        });
    }
}

private RelayCommand? sendLocationSignalCommand;
public RelayCommand SendLocationSignalCommand
{
    get
    {
        return sendLocationSignalCommand ??= new RelayCommand(obj =>
        {
            SendLocationSignal();
        });
    }
}

private RelayCommand? editLocationSignalCommand;
public RelayCommand EditLocationSignalCommand
{
    get
    {
        return editLocationSignalCommand ??= new RelayCommand(obj =>
        {

```



```

        EditSignalViewModel editSignalViewModel = new(Brain, EditSignalMode.LocationSignal);
        EditSignalWindow editSignal = new()
        {
            DataContext = editSignalViewModel
        };
        editSignal.ShowDialog();
    });
}
}

```

```

private RelayCommand? editSensorySignalCommand;
public RelayCommand EditSensorySignalCommand
{
    get
    {
        return editSensorySignalCommand ??= new RelayCommand(obj =>
        {
            EditSignalViewModel editSignalViewModel = new(Brain, EditSignalMode.SensorySignal);
            EditSignalWindow editSignal = new()
            {
                DataContext = editSignalViewModel
            };
            editSignal.ShowDialog();
        });
    }
}

```

```

private RelayCommand? sendSensorySignalCommand;
public RelayCommand SendSensorySignalCommand
{
    get
    {
        return sendSensorySignalCommand ??= new RelayCommand(obj =>
        {
            SendSensorySignal();
        });
    }
}
}
#endregion

```

```

public MainWindowViewModel(BrainModel brain)
{
    Brain = brain;
    Painter = new(brain);

    Brain.UpdateInCells += Painter.DrawInCells;
    Brain.UpdateOutCells += Painter.DrawOutCells;

    Painter.InCellsPainted += () => InLayerCells = Painter.InLayerCells;
}

```

```

        Painter.OutCellsPainted += () => OutLayerCells = Painter.OutLayerCells;
        Painter.FeedBackSynapsesPainted += () => FeedBackSynapsesBitmap =
Painter.FeedBackSynapsesBitmap;
        Painter.FeedForwardSynapsesPainted += () => FeedForwardSynapsesBitmap =
Painter.FeedForwardSynapsesBitmap;
        Painter.BasalDendritesPainted += () => BasalDendrites = Painter.BasalDendrites;
    }
}
}

```

Модуль визуализации. Класс RelayCommand.

using System.Windows.Input;

```

namespace ThousandBrainsVisualisation.ViewModel
{
    public class RelayCommand : ICommand
    {
        Action<object?> execute;
        Func<object?, bool>? canExecute;

        public event EventHandler? CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }
        public RelayCommand(Action<object?> execute, Func<object?, bool>? canExecute = null)
        {
            this.execute = execute;
            this.canExecute = canExecute;
        }
        public bool CanExecute(object? parameter)
        {
            return canExecute == null || canExecute(parameter);
        }
        public void Execute(object? parameter)
        {
            execute(parameter);
        }
    }
}

```

Модуль визуализации. Класс App.

```

<Application x:Class="ThousandBrainsVisualisation.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="View/MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>

```

Модуль визуализации. Класс App.

```
using System.Windows;
using ThousandBrainsVisualisation.BrainFillerNS;
using ThousandBrainsVisualisation.Logic;
using ThousandBrainsVisualisation.Model;
using ThousandBrainsVisualisation.ViewModel;

namespace ThousandBrainsVisualisation
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
        private BrainModel Brain;
        private MainWindowViewModel MainWindowViewModel;
        private BrainFiller BrainFiller;
        private Server Server;
        public App()
        {
            Brain = new();
            MainWindowViewModel = new(Brain);

            Brain.SendLocationSignal += SendLocationSignalToClient;
            Brain.SendSensorySignal += SendSensorySignalToClient;
            Brain.SetNeedBrainInitialize += SetNeedBrainInitialize;
            Brain.SetNeedBrainPrint += SetNeedBrainPrint;

            BrainFiller = new(Brain);
            BrainFiller.BeginGetData += SetIsBusy;
            BrainFiller.EndGetData += UnSetIsBusy;

            Server = new();
            Server.SendDataToApplication += SendDataToApplication;
            Server.ChangeConnectionsCount += SetHasActiveConnection;

            Task.Run(() => Server.ListenAsync());
        }

        protected override void OnActivated(EventArgs e)
        {
            base.OnActivated(e);
            MainWindow.DataContext = MainWindowViewModel;
        }

        private void SendDataToApplication(string? data)
        {
            BrainFiller.SendData(data);
        }

        private void SetHasActiveConnection(int count)
```

```

{
    // TODO обнулять данные, если нет колонок
    MainWindowViewModel.HasActiveConnection = count > 0;
}

private void SetIsBusy()
{
    MainWindowViewModel.IsBusy = true;
}

private void UnSetIsBusy()
{
    MainWindowViewModel.IsBusy = false;
}

// TODO Переделать на асинхронные методы
private void SetNeedBrainInitialize()
{
    Server.SendDataToClient(BrainCommands.NeedBrainInitialize);
    Brain.NeedBrainInitialize = false;
}

private void SetNeedBrainPrint()
{
    Server.SendDataToClient(BrainCommands.NeedBrainPrint);
    Brain.NeedBrainPrint = false;
}

private void SendLocationSignalToClient()
{
    Server.SendDataToClient(BrainCommands.LocationSignalBegin);
    foreach (int i in Brain.LocationSignal)
    {
        Server.SendDataToClient(i.ToString());
    }
    Server.SendDataToClient(BrainCommands.LocationSignalEnd);
    Brain.NeedSendLocationSignal = false;
}

private void SendSensorySignalToClient()
{
    Server.SendDataToClient(BrainCommands.SensorySignalBegin);
    foreach (int i in Brain.SensorySignal)
    {
        Server.SendDataToClient(i.ToString());
    }
    Server.SendDataToClient(BrainCommands.SensorySignalEnd);
    Brain.NeedSendSensorySignal = false;
}
}
}

```