Сети Жизни vs Deep Learning

- 1. Введение
- 2. Блеск и нищета нейросетей
- 3. Облачная ИТ архитектура и микросервисы
- 4. Асинхронные коммуникации и персистентная рекурсия
- 5. Аналогия микросервисной архитектуры и архитектуры организмов
- 6. Физические ограничения ИТ моделей
- 7. Архитектура сетей жизни vs Deep Learning
- 8. Современные ИТ технологии и социальная деструкция
- 9. Заключение
- 10. Дополнение: Асинхронное логика и асинхронное программирование

1. Введение

Данный текст написан по материалам семинара Нейросети 2.0, так как, из-за ограниченности времени и формата семинара, а также сложности предмета, невозможно донести всю полноту своей позиции. Под нейросетями в данной материале подразумеваются современные математические модели с глубоким обучением.

В ходе выступления на семинаре я постоянно называл нейросети в их текущей ипостаси калькулятором, хранилищем и даже гадостью – это была жесткая реакция, как противовес тому, что некоторые участники семинара постоянно пытаются одушевить нейросети и приписать ИТ системам, построенным на базе нейросетей некую субъектность - “ведем диалог с нейросетью”, “нейросеть подошла к решению задачи творчески” и т.д.

Современные решения на базе нейросети в своей основе являются **огромными статическими наборами параметров**, которые формируются и изменяются только на этапе обучения нейросети, а **на этапе использования остаются неизменными**, а значит, запрос непосредственно к нейросети с некоторыми заданными параметрами всегда будет выдавать один и тот же результат, вне зависимости от истории общения с ней (от контекста). Например при общении с чат-ботом, построенным на базе NLP (natural language processing, обработка естественного языка), вся вариабельность диалога и учет контекста общения (запоминания фактов и переменных диалога) выполняется не нейросетью, а машиной состояний, которая интегрирована с ней или же парой слоев дополнительной нейросети построенной поверх мощной векторной модели. Но эти пару слоев не запоминают факты и переменные, а лишь **обеспечивают динамическое улучшение распознавания предложений**.

И это нормальный подход для бизнес задач, так как транзакции в ходе любой деятельности всегда запоминаются в базах данных, поверх которых построена машина состояний организации, а значит любые нейросети будут встраиваться бизнесом в такую машину состояний, в которой и хранится текущий контекст.

С другой стороны, задача рисования картинки по текстовому описанию, которую в качестве примера привел Максим, это задача другого класса, так сказать, задача на креативность и творчество, поэтому я решил посмотреть как устроена архитектура такой системы и какие нейросети там используются. **Генеративные модели — это сети без учителя (нет необходимости вручную готовить размеченные наборы данных), для обучения используется вторая (как правило сверточная) сеть-дискриминатор. Две нейросети соревнуются, одна пытается сформировать изображение максимально близкое к изображениям в заданном обучающем датасете, а вторая пытается определить является ли входное изображение “истинным” или сгенерированным. Начальный сигнал генерации задается рандомным образцом пикселей.**

Речь идет о мультимодальной генеративной нейросети, которая создает рисунок по текстовому описанию.

Источник: Мультимодальные нейронные сети, как искусство

Источник: Как нейронные сети рисуют картины

Казалось бы вот она встреча двух нейросетей, о которой столько говорилось на семинаре! Но ведь речь шла о совсем другой архитектуре. Лучше всех мысль об ансамбле нейросетей выразил Андрей Шел. - цитирую: “***Нужно иметь ввиду взаимодействия разных, понятие другого здесь очень важно, разных нейросетей, причём не на уровне только внешних интерфейсов, но и на уровне их внутреннего устройства, потому что некоторые узлы, некоторое непустое множество узлов, оно присутствует и в одной и в другой нейросети, ну и в 3-й, 4-й и так далее***. “

Однако, организация взаимодействия разных видов нейросетей потребует другие алгоритмы, другую методологию, другие языки программирования, другие библиотеки. В разговоре с опытным разработчиком в этой сфере (который десятилетия в теме), я спросил “Почему используете Python (этот язык плохо приспособлен для организации параллельных вычислений и асинхронного программирования) для проектов в области AGI (Общий Искусственный Интеллект) ?”. Ответ был: “Что-нибудь сначала создадим, а потом будем разбираться”. Для справки - все основные разработки в области AI и ML ведутся в Python.

Глава 3, возможно, будет сложной для неспециалистов, я постарался максимально просто изложить состояние дел в современной ИТ архитектуре.

С точки зрения ИТ, рекурсия и фрактал — это понятия относящиеся к архитектуре фон Неймана, когда имеется программный автомат, который шаг за шагом последовательно запускает функции и процедуры.

Но современная ИТ архитектура - это асинхронная парадигма (Asynchronous Paradigm), асинхронная логика и функция обратного вызова (Callback), ниже, в разделе 3, будет показан генезис этой архитектуры.

В этой архитектуре то, что верхнеуровнево воспринимается как рекурсия, на самом деле, на уровне программного кода является наиболее часто исполняемые замкнутые последовательности асинхронных функций, связанные через функции обратного вызова. Со стороны наблюдателя может показаться, что это рекурсия, но это не так. С фрактальностью ситуация сложнее, об этом позже.

2. Блеск и нищета нейросетей

2.1 ***Нейросети могут обрабатывать данные и сигналы, которые недоступны напрямую человеку. Нам придется решать сложнейшие задачи визуализации и представления данных, чтобы задачи, которые решают нейросети стали доступными к обработке человеком.*** Data Science сегодня — основной инструмент обработки экспериментальных данных.

2.2 ***Скорость обработки информации современными компьютерами, кремниевыми "мозгами", несомненно важное качество, однако, использование органическими мозгами массово параллельных операций в потрясающих масштабах, значительно нивелирует это превосходство.*** Например если рассмотреть зрительный тракт, то он представляет из себя около 1 млн нервных волокон по которым информация о текущей световой активности по всей поверхности сетчатки моментально и параллельно передается в зрительные зоны мозга.

Длина зрительного тракта 0.05 метра, длительность одного нервного импульса 0.002 сек, скорость распространения пяти импульсов в миелинизированном нервном волокне 120 м/сек. Время передачи изображения с сетчатки глаза 5 * 0.002 + 0.05/120 = 0.01 с.

Передачи изображения с матрицы 1920*1080 пикселей по шине в оперативную память выполняется последовательно. Для 32 разрядной шины с тактовой частотой 2 Ггц время составит: 1920*1080*2 /32/2 000 000 000 = 0.00004 с.

Разница в несколько сотен раз, но не будем забывать, что и дальнейшая обработка в органике будет выполняться в массово параллельном режиме, а к "кремниевым мозгам" понадобится подключить мощнейшие графические видеокарты. Видеокарты обеспечивают распараллеливание, но это всего лишь примитивное перемножение матриц.

2.4 ***Фаза обучения нейросети и фаза использования нейросети разорваны, а обучение всегда требует огромных вычислительных мощностей и времени.*** Например, в NLP алгоритмах в качестве словарей активно используются массивные векторные модели. Нейросети построенные на базе таких векторных моделей будут всегда зависеть от текстов, которые использовались для их построения. Даже в случае обучения без учителя, дополнительное обучение на этапе эксплуатации, как правило невозможно.

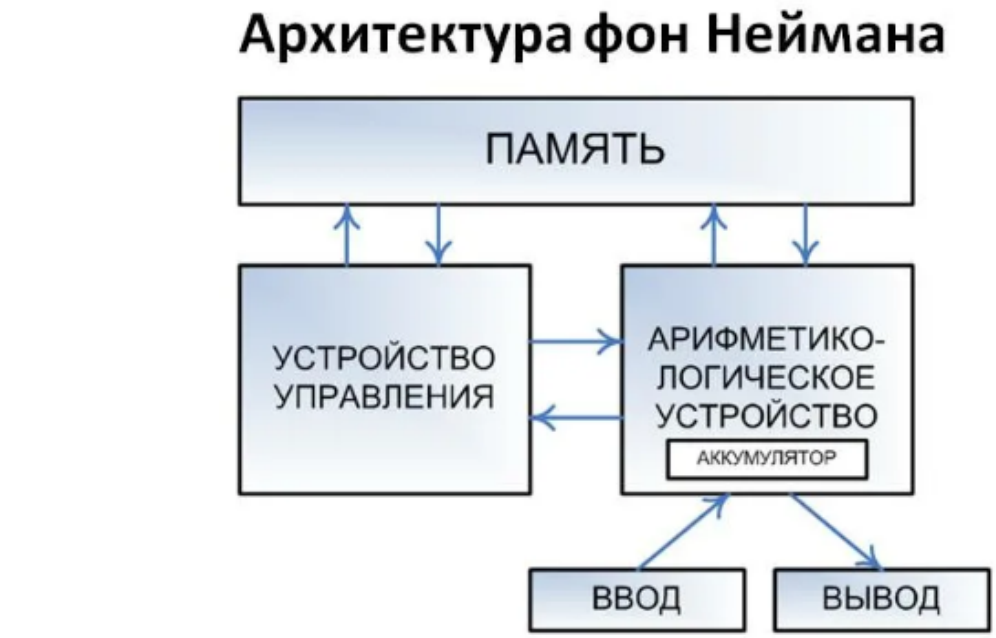
Итак, вывод: современную нейросеть можно рассматривать как чистую функцию (которая всегда возвращает одно и тоже значение при одинаковых входных

параметрах и не обладает побочными эффектами). А значит для обеспечения какой-либо содержательной логики нейросеть всегда должна интегрироваться с машиной состояний. Эта машина состояний, т.е. по сути творчество разработчиков сервиса и создает эффект и ощущение того что нейросеть (калькулятор :D) “общается” с пользователем.

3. Облачная ИТ архитектура и микросервисы

ВданномразделепоказанаэволюцияИТархитектурыотпрограммногоавтомата, выполненного в виде монолитного исполняемого модуля операционной системы (ОС), до современного асинхронного приложения на базе микросервисной архитектуры.

3.1 В основе построения всех современных компьютеров лежит архитектура фон Неймана, суть которой состоит в последовательном выполнении команд, которые хранятся в оперативной памяти и оперируют с данными хранящимися там же.

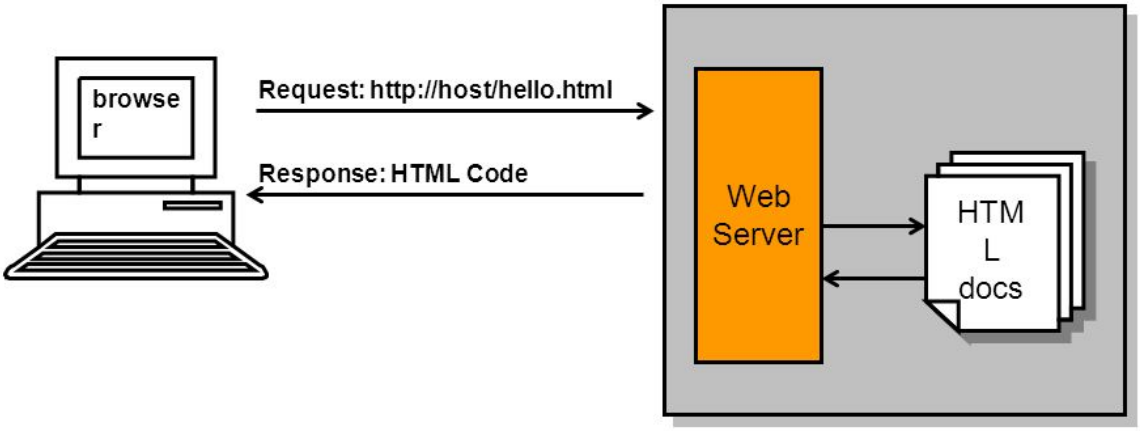


Принципы фон Неймана

- Принцип однородности памяти
- Принцип адресности
- Принцип программного управления
- Принцип двоичного кодирования

3.2 Появление сети Интернет привело к созданию *архитектуры всемирной паутины* — World Wide Web, представляющей из себя гипертекстовые документы, размещенные на веб-серверах и связанные друг с другом с помощью гиперссылок. Конечно, вся эта конструкция, состоит из огромного числа программных модулей и компьютеров, функционирующих в архитектуре фон Неймана, но давайте на мгновение забудем об этом и рассмотрим ее верхнеуровнево. Тогда основными компонентами этой архитектуры будут: веб-страница, веб-сервер, клиентский веб-браузер, сеть Интернет и HTTP протокол. ***Основной акцент сделаем не на структуре сети, а на процессах протекающие в ней.***

WEB архитектура



Основной цикл взаимодействия выглядит следующим образом:

Пользователь через web-браузер запрашивает документ, указывая его адрес в сети (URL), служба доменов (DNS) определяет web-сервер на котором расположен документ и пересылает запрос ему. Модуль маршрутизации web-сервера разбирает URL и выбирает контроллер (controller), ответственный за обработку данного типа запросов и передает запрос ему. Контроллер подготавливает необходимые данные и пересылает их модулю представления (view), который форматирует их в виде HTML документа. Полученная web-страница пересылается в web-браузер пользователя.

Приэтомсерверпараллельноможетобслуживатьмножестворазличных запросов, которые он получает асинхронно от различных пользователей. Было бы неприемлемо, если бы посетители сайта выстраивались в очередь один за другим, как в магазине на кассе. Обслуживание очередного запроса выполняется отдельным процессом операционной системы (fork) — этот объект довольно массивный, он долго стартует при запуске. По этой причине web-сервер после старта инициализирует пул (pool) таких процессов и, далее, выделяет “горячий” процесс из пула для обслуживания очередного входящего запроса (request).

Такимобразом,webархитектурапредставляетизсебякоммуникационнуюсетьв которой **обмен документом между web-сервером и web-браузером происходит в**

синхронном режиме, но сами **запросы от web-браузеров на web-сервер поступают асинхронно**, обслуживание запросов ***происходит в параллельном режиме*** в различных процессах ОС. Сами документы могут быть довольно большого размера, а периоды времени между запросами очередного документа от пользователем тоже достаточно значительные. Ведь человеку нужно время, чтобы ознакомиться со страницей и сделать следующий клик.

3.3 SOAP (от англ. Simple Object Access Protocol) — простой протокол доступа к объектам) — это протокол обмена структурированными сообщениями в распределенной вычислительной среде. ***Данная архитектура представляет собой протокол взаимодействия с web-сервисом и предназначен для обмена сообщениями между приложениями в формате XML (расширяемый язык разметки данных). Концептуально — это такая же коммуникационная сеть что и гипертекст, но она предназначен для обмена сообщениями не между web-серверами и пользователями web-страниц, а между различными приложениями корпорации, где для каждого приложения разрабатывается web-сервис и публикуется на сервисной шине.*** Протокол SOAP упакован в HTML, для его развертывания используется та же самая WEB инфраструктура.

К вопросу о фракталах — сравните архитектуру компьютера (приведена ниже) и архитектуру корпоративный сервисной шины (ESB).

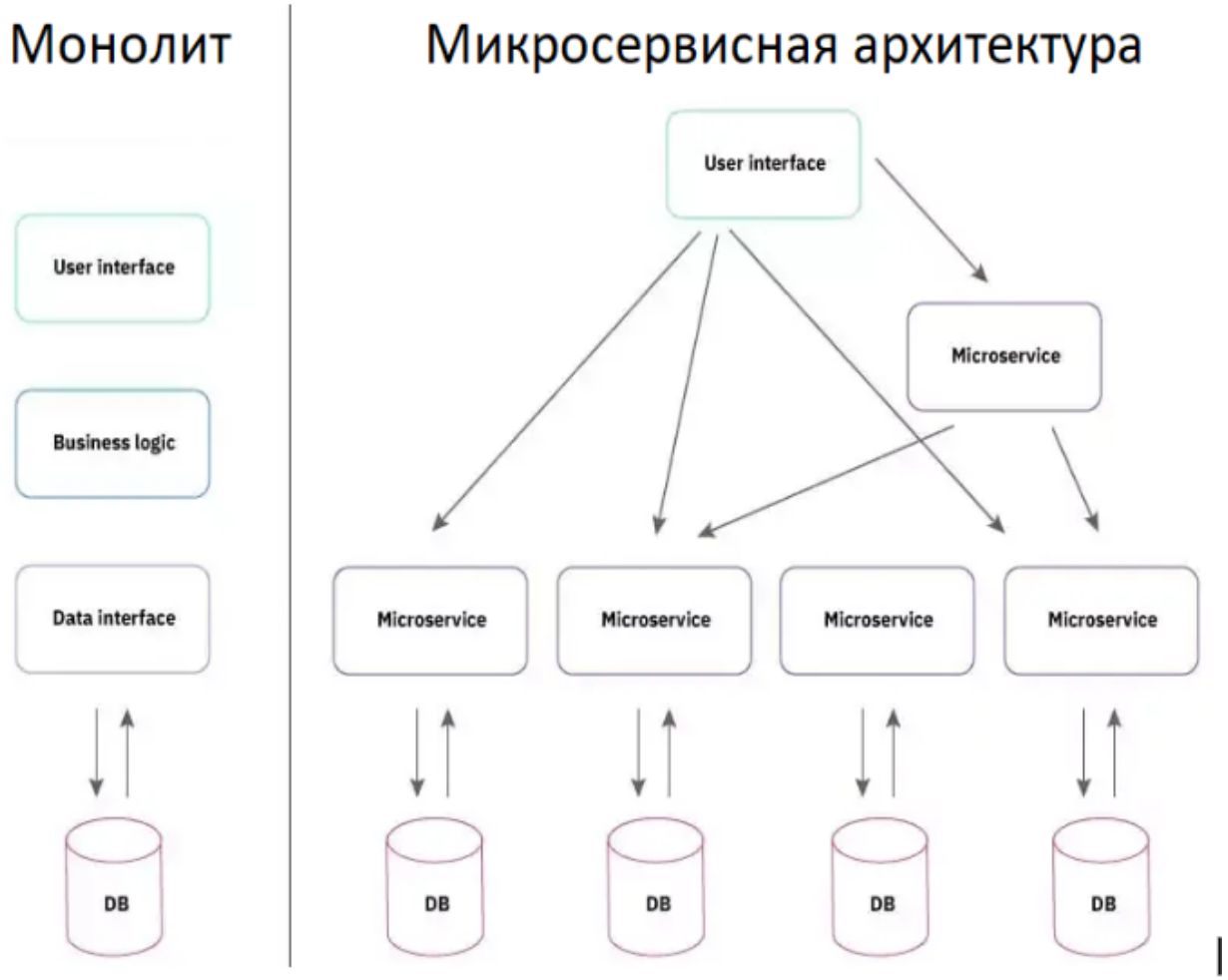
3.4 ***RESTful интерфейс и микросервисная архитектура*** это следующий шаг в эволюции коммуникационных ИТ архитектур.

Главное различие микросервисов и шины в том, что ***ESB была создана в контексте интеграции отдельных приложений***, чтобы получилось единое корпоративное распределенное приложение. ***А микросервисная архитектура предполагает проектирование приложений с разбиением их на микросервисы. Микросервисы — это маленькие автономные сервисы, работающие вместе и спроектированные вокруг бизнес-домена.***

Например, процедуру входа и регистрации на сайте можно разместить в один микросервис, процедуру оплаты в другой и так далее.

Это позволяет эффективно решать проблемы:

- независимое развертывания - не нужно заменять на сервере все приложение полностью, достаточно заменить необходимый микросервис;
- масштабирование нагрузки - при росте нагрузки можно выделить ресурсы точно для необходимого микросервиса;
- эффективная декомпозиция приложения - микросервисы слабо связаны и их разработка и отладка может вестись независимыми группами разработки и с использованием различных инструментов программирования;
- и другие.



REST (от англ. Representational State Transfer — «передача репрезентативного состояния» или «передача „самоописываемого“ состояния») — архитектурный стиль взаимодействия компонентов распределенного приложения в сети. ***Другими словами, REST — это набор правил того, как программисту организовать написание кода серверного приложения, чтобы все системы легко обменивались данными и приложение можно было легко масштабировать.***

Главным из этих правил является “***Отсутствие состояния”*** - протокол взаимодействия между клиентом и сервером требует соблюдения следующего условия: в период между запросами клиента никакая информация о состоянии клиента на сервере не хранится (Stateless protocol или «протокол без сохранения состояния»). Все запросы от клиента должны быть составлены так, чтобы сервер получил всю необходимую информацию для выполнения запроса. Состояние сессии при этом сохраняется на стороне клиента. ***Именно это правило является основой для обеспечения слабой связности микросервисов.***

Отметим, что если SOAP - это программный интерфейс, разработчики технологии все еще пытаются имитировать обмен сообщениями по протоколу HTTP, как удаленный вызов одного программного модулями другим (RMI - Remote Method Invocation), то REST - это уже ясно выраженная концепция обмена информационными сообщениями между микросервисами.

REST-интерфейсыещеназываютAPI-интерфейсами.API(ApplicationProgramming Interface) — программный интерфейс приложения.

В качестве примера, можно привести API-интерфейс Центробанка, который возвращает валютные курсы. В качестве входного запроса мы перечисляем нужные нам валюты, а в качестве ответа получаем их курсы. Нас не интересует как устроены системы ЦБ. Мы строим свое приложение, например, чат-бот, которое в нужный нам момент должно получить курсы, чтобы запросы наших пользователей были удовлетворены. Сервисы ЦБ также не волнует состояние и устройство наших систем.

3.5 Тема контейнеризации на первый взгляд несколько в стороне от вышеизложенного, но это только кажется. Если в предыдущих пунктах мы рассматривали ИТ архитектуру с точки зрения парадигмы ее проектирования, то контейнеризация - это взгляд с точки зрения ее использования, фактически это инструменты для превращения операционной системы в облако.

Контейнеризацияисистемыуправленияконтейнерамиосновныеэлементы облачной архитектуры. Контейнер позволяет упаковать приложение со всеми зависимостями и быстро развернуть контейнер для использования - либо на продуктовом сервере, либо на компьютерах команды разработки. ***Как правило в контейнере устанавливаются один или несколько микросервисов некоторой доменной области***, что позволяет эффективно решать задачи бесперебойного обслуживания и масштабирование нагрузки - в нужный момент запускается необходимое количество контейнеров и выделяются нужные ресурсы.

Docker — программное обеспечение ***для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации***, контейнеризатор приложений. ***Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть развернут на***

любой Linux-системе с поддержкой контейнеризации, а также предоставляет набор команд для управления этими контейнерами.

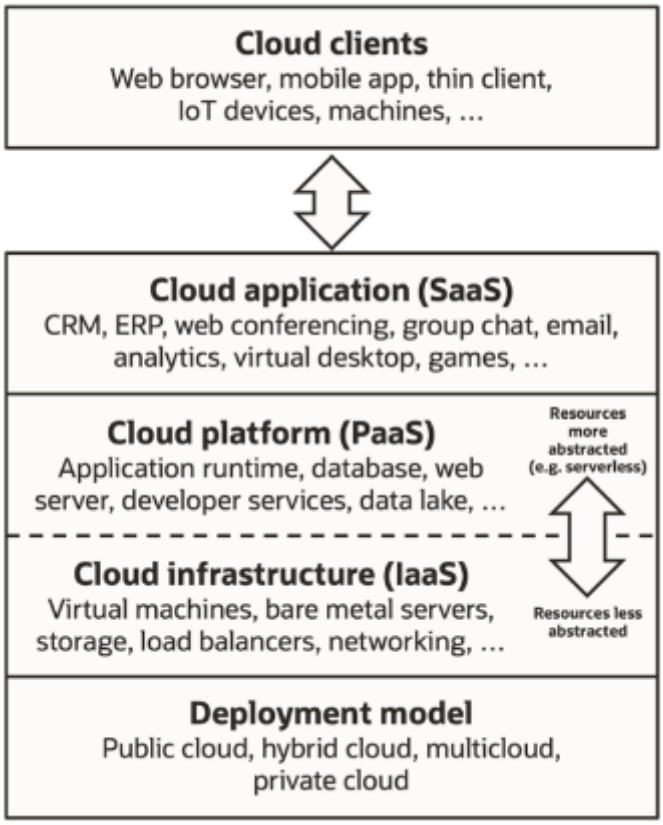
Kubernetes (от др.-греч. κυβερνήτης — «кормчий», «рулевой», часто также используется нумероним k8s) — открытое программное обеспечение для оркестровки контейнеризированных приложений — автоматизации их развертывания, масштабирования и координации в условиях кластера.

DevOps (акроним от англ. development & operations) — методология автоматизации технологических процессов сборки, настройки и развертывания программного обеспечения.

DevOps ***методология предполагает активное взаимодействие специалистов по разработке со специалистами по информационно-технологическому обслуживанию и взаимную интеграцию их технологических процессов друг в друга для обеспечения высокого качества программного продукта.*** DevOps предназначен для эффективной организации создания и обновления программных продуктов и услуг. Методология основана на идее тесной взаимозависимости создания продукта и эксплуатации программного обеспечения, которая прививается команде как культура создания продукта.

Cloud computing - облачные вычисления - ***это доступность ресурсов компьютерной системы по требованию, особенно хранилища данных (облачное хранилище) и вычислительной мощности, без прямого активного управления со стороны пользователя. Большие облака часто имеют функции, распределенные по нескольким местоположениям, каждое местоположение является центром обработки данных.*** Облачные вычисления основаны на совместном использовании ресурсов для достижения согласованности и обычно используют модель "оплаты по мере использования", которая может помочь в снижении капитальных затрат, но также может привести к непредвиденным операционным расходам для неосведомленных пользователей.

Облачная архитектура



4. Асинхронные коммуникации и персистентная рекурсия

В синхронном коде выполнение функций происходит в том месте, где они были вызваны, и в тот момент, когда происходит вызов. В асинхронном коде всё по-другому. Вызов функции не означает, что она отработает прямо здесь и сейчас. Более того, мы не знаем, когда она отработает.

Синхронный подход в случае большого числа медленных операций ввода-вывода (обмен со внешней средой - с дисками, с сетью интернет, с печатным устройство)) очень неэффективно использует ресурсы системы. Асинхронный же код продолжает свою работу во время любых медленных операций, в том числе при обращении к микросервисам, скорость ответа от которых может, как и любые другие операции ввода-вывода, быть очень медленной.

Другими словами, код никогда не блокируется на операциях ввода-вывода (IO), но может узнать об их завершении. Правильно написанные асинхронные программы (в тех ситуациях, где это нужно) значительно эффективнее синхронных. Иногда это настолько критично, что синхронная версия просто не справляется с задачей.

Упорядоченность асинхронных операций обеспечивается в первую очередь через функцию обратного вызова - Callback, обычно это последний аргумент в вызове асинхронной функции. Эта функция вызывается после завершения асинхронной операции. Исполняющая подсистема языка программирования никогда не ждет завершения асинхронной функции, после ее запуска она переходит к обработке следующей функции.

В примерах ниже специально выбран самый ранний синтаксис, чтобы подчеркнуть значение ***функции обратного вызова****.* ***Это гораздо более значимый феномен, чем понятие классической рекурсии, которая оставляет нас в парадигме последовательного программирования архитектуры фон Неймана***

Важнейшее преимущество асинхронной неблокируюшей логики в сочетании с техникой корутин (coroutine), состоит в том, что в классических нагруженных системах (при использовании распараллеливания с использованием процессов ОС), список доступных процессов обычно быстро исчерпывается, если запущенные в процессах обмены с внешними сервисами синхронные и/или медленные. Заметим

же, что переключения между процессами ОС само по себе довольно ресурсоемкое.

Резюмирую - без использование техники асинхронного программирования построить современную высоконагруженную микросервисную систему (единорога, о котором все мечтают ж-)) невозможно или очень сложно.

Пример классическое рекурсии в архитектуре фон Неймана:

```
let cnt = 0;

function sendMsg1() {
  console.log('Msg1');
  cnt++;
  if (cnt < 10) sendMsg1();
}

sendMsg1();
```

Пример персистентной рекурсии в асинхронной архитектуре:

```
function startMsg(callback) {
  console.log('Start');
  callback();
}
const stopMsg = function() {
  return function() {
     console.log('Stop');
  }
}
```

```
const sendMsg1 = function(callback) {
  return function() {
    console.log('msg1');
    callback();
  }
}
const sendMsg2 = function(callback) {
  return function() {
    console.log('msg2');
    callback();
  }
}
startMsg(sendMsg1(
        sendMsg1(
          sendMsg1(
            sendMsg1(stopMsg()))))));
startMsg(sendMsg1(
        sendMsg2(
          sendMsg1(
            sendMsg2(stopMsg()))))));
```

Несложно заметить разницу, если классическая рекурсия разворачивается на этапе выполнения, то перманентная рекурсия в асинхронной логике декларативно формируется на этапе описания цепочек функций. Любая цепочка функций может

быть декларирована и в нужный момент активирована.

При этом не составит никакого труда описать любое дерево из функций (которые, суть будущие действия) или целую сеть с обратными связями. Единственное отличие от живых сетей - в живых сетях эти “функции” и сети создает сама Жизнь, а в нашем примере это пока делается руками программиста.

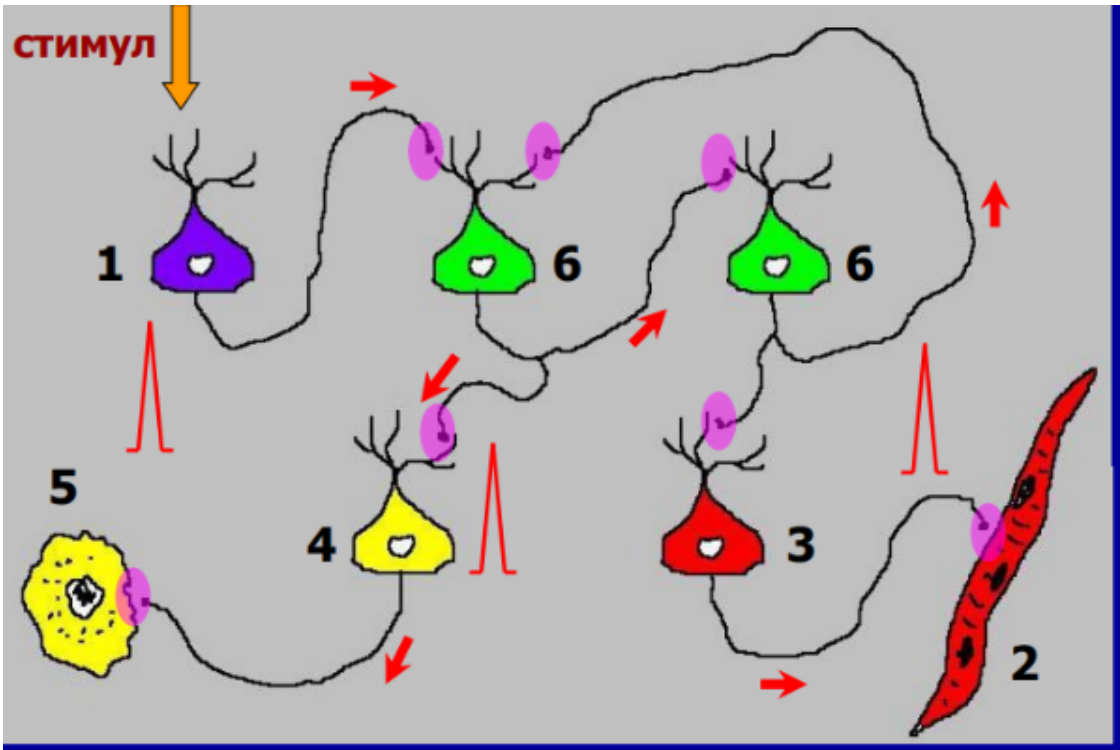
Совершенно очевидно что такое архитектурное решение гораздо ближе, к коммуникативным сетям жизни, легко увидеть аналогию между самскарами (следами действий) в теории Будды о Сансаре и этими декларативными цепочками функций.

5. Аналогия микросервисной архитектуры и архитектуры организмов

Итак, какжеустроенсовременныйцифровой Левиафан? Современноеоблачное приложение представляет собой набор микросервисов некоторой доменной области. Каждый микросервис реализован как асинхронный код, который содержит набор прослушивателей, связанных как с другими микросервисами, так и с удаленными сервисами (через REST API) из любой точки мира. Прослушиватели, при поступлении заданного сигнала, инициируют выполнение либо программной функции, функции исполнительного устройства, либо обрабатывают данные переданные датчиком.

Совершенно очевидна аналогия между архитектурой такого распределенного приложения, и архитектурой живого организма, который состоит из различных автономных органов, желез и рецепторов, связанных между собой сообщениями передающимися по нервным и/или гуморальным каналам.

Если проводить аналогию дальше, то синапсы - области связи отростков различных нейронов, дендритов и аксонов (области выделенные розовым на рисунке ниже), аналогичны функциям обратного вызова (Callbacks) в асинхронной логике.



Передача сигнала к следующей клетке происходит в особых структурах – **синапсах** (центральных, нервно-мышечных, вегетативных; *на схеме их 7*).

Сигнал по нейрону (вернее – по его мембране) передается в виде коротких электрических импульсов – **потенциалов действия** (**ПД**, длительность 1-2 мс, амплитуда около 100 мВ).

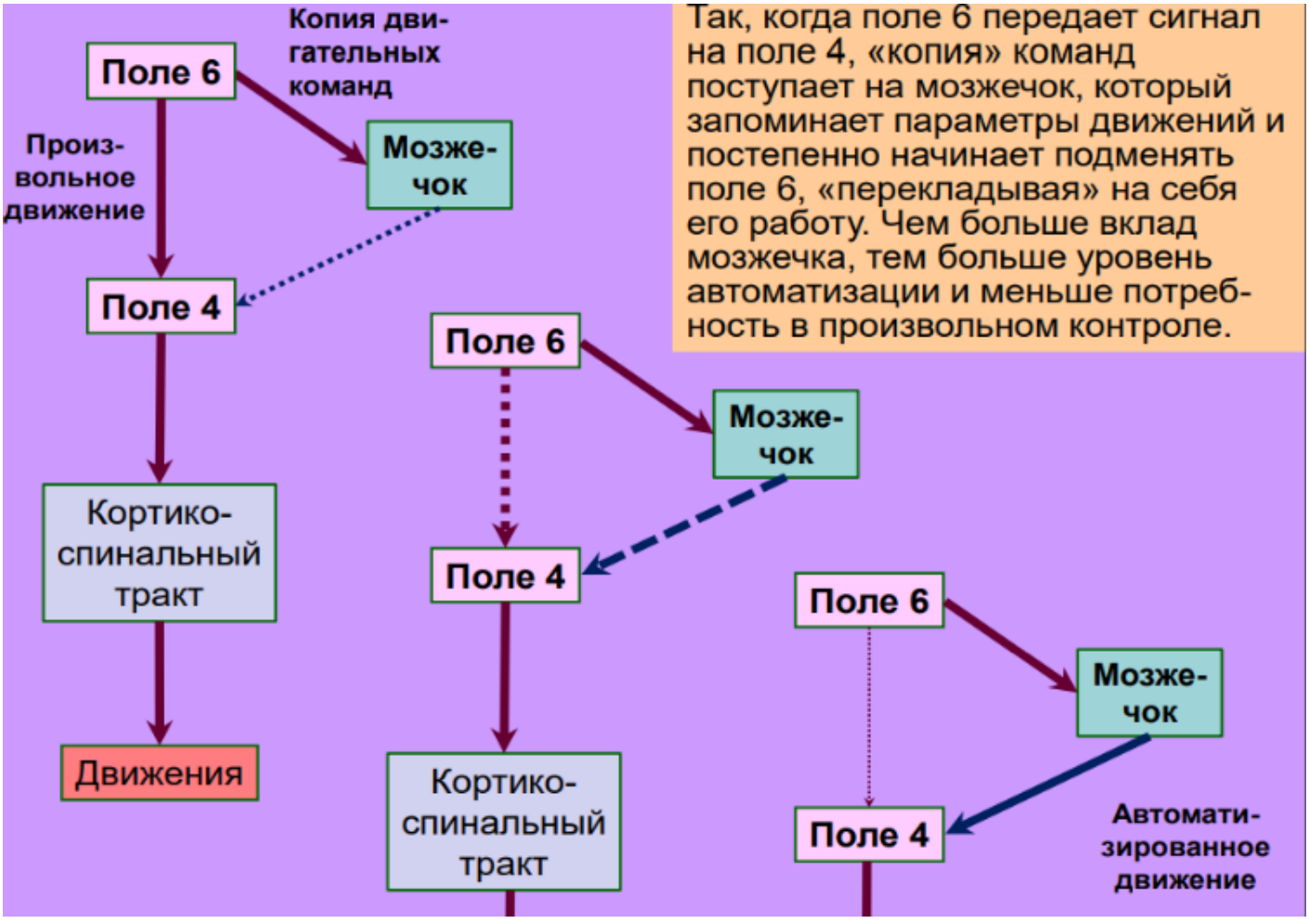
Сигнал от нейрона к следующей клетке передается за счет выделения из окончания аксона особого вещества («**медиатора**»), которое воздействует на активность клетки-мишени.

За счет произвольного контроля мы можем реализовать (заучивать) совершенно новые движения; это огромный плюс.
Но существует и минус: произвольное управление движениями «тормозит» другие высшие функции коры (точнее, конкурирует с ними).

Такое торможение снижает способность оперативно реагировать на изменения условий среды, и в ходе эволюции появился еще один тип движений – автоматизированные: при многократных повторах произвольного движения происходит запоминание его параметров.

Базальные ганглии: запоминание параметров программы (последовательности движений).

Мозжечок: запоминание параметров конкретных движений.



Так, когда поле 6 передает сигнал на поле 4, «копия» команд поступает на мозжечок, который запоминает параметры движений и постепенно начинает подменять поле 6, «перекладывая» на себя его работу. Чем больше вклад мозжечка, тем больше уровень автоматизации и меньше потребность в произвольном контроле.

Источник: **д.б.н. профессор Дубынин В.А. «СЕНСОРНЫЕ и ДВИГАТЕЛЬНЫЕ СИСТЕМЫ МОЗГА», 2020 Альбертович.**

6. Физические ограничения ИТ моделей

7.1 ***Основное ограничение современных ИТ систем связано со скоростью света — скорость электрического импульса в проводнике равно ⅔ скорости света.*** Казалось бы огромная величина, но вспомним, что основная масса используемых компьютеров работает в классической архитектуре фон Неймана, это значит что операции процессоров всегда выполняются последовательно. Устройства же с физическими параллельными вычислениями пока дороги и широко не распространены.

Поэтому для увеличения мощности вычислительной системы в настоящее время широко используется, так называемое, горизонтальное масштабирование - распараллеливание вычислений на множестве компьютеров объединенных быстрым каналом TCP/IP.

На этом пути возникают проблемы, а именно - Теорема CAP:

Если система C (Consistency - согласованность данных в узлах распределенной системы) , то либо A (Availability - гарантия ответа при отсутствия гарантии совпадения ответов от разных узлов),) либо P (Partition tolerance - устойчивость к разделению узлов).

Источник: CAP-теорема простым, доступным языком

CA - все согласовано и быстро. Но нет горизонтального масштабирования. Только увеличивать мощность кластера.

CP - все хорошо, но ответ может быть долгим. Все горизонтально отмасштабировано, но время будет тратится на согласование данных в кластерах.

AP - отклик быстрый, все горизонтально отмасштабировано, но данные в узлах рассогласованы.

А это значит, что точную, быструю и обслуживающую в моменте “бесконечное” количество агентов глобальную компьютерную модель построить в принципе невозможно.

Помните господа марксисты и товарищи большевики обещали все научно запланировать и управлять рационально? Так ведь это в принципе невозможно, а это значит на этом пути нас ждет только упрощение, усечения системы. И никакой искусственный интеллект тут не поможет, он ещё и самый тормознутый в этой схеме.

Впрочем это ещё не самое интересное. Гораздо интереснее как же господа глобалисты будут обновлять соответствие между моделью и реальностью. Тут никакой Интернет вещей не поможет. Пока параметры модели попадут в ИТ систему, пока перестроятся модели АИ, она станет не актуальной. Значит будут гонять заезженную пластинку и люди будут плясать под неё ?

Источник: Теорема CAP и глобальный ИИ.

7.2 ***Второе ограничение связано с достижением компонентами печатных схем предельного размера сопоставимого с размером атомов.*** Помните закон Мура — количество транзисторов на микрочипах вырастало в среднем в два раза каждые два года. Почему Закон Мура работал?

Долгое время Закон Мура работал как часы. Транзисторы уменьшались, их число росло, а мощность возрастала. А это, на секундочку рост по экспоненте, то есть очень быстро! Процессоры производят путем фотолитографии. Иными словами, лазер светит через трафарет, который называется маской, и процессор буквально выжигается на кремниевой подложке.

Так индустрия и развивалась: когда достигали предела разрешения лазера — меняли его на лазер с более короткой длиной волны.

Поначалу использовали дуговые ртутные лампы, а не лазеры, с длиной волны 436 нм — это синий свет. Потом освоили 405 нм — это фиолетовый. И наконец до 365 нм — ближний ультрафиолет. На этом эра ртутных ламп закончилась и началось использование ультрафиолетовых газовых лазеров. Сначала освоили 248 нм — средний ультрафиолет, а потом 193 нм — глубокий ультрафиолет или DUV. Такие лазеры давали максимальное разрешение в 50 нм и на какое-то время этого хватало. Но потом произошел переломный момент…

К 2006 году надо было осваивать техпроцесс в 40-45 нм. Разрешения лазеров было недостаточно. Гиганты Кремниевой Долины потратили сотни миллионов долларов для перехода на 157 нм (лазеры на основе фторид-кальциевой оптики), однако всё было впустую.

В 2000 году после пересечения порога в 100 нм из-за сильного уплотнения транзисторов, расстояние между ними стало настолько маленьким, что начались утечки тока! Грубо говоря, электрончики перескакивали из одного участка схемы в соседний — где их быть не должно. И портили вычисления… А также увеличилось паразитное энергопотребление.

Из-за этого пришлось поставить крест на росте тактовых частот. Если раньше частоты удваивались так же быстро, как транзисторы, то теперь прирост практически остановился.

В итоге, вопреки своим планам, Intel застрял на 14 нм техпроцессе, а тактовые частоты остановили свой рост. И примерно с 2010 года начались 10 интересных лет оптимизаций.

Если раньше прогресс обеспечивался брутальным уменьшением техпроцесса и прирост производительности давался легко, то теперь началась настоящая работа по допиливанию всего того, что человечество придумало за 40 предыдущих лет.

Люди стали искать инновации за пределами Закона Мура:

- ***Процессоры стали многоядерными и многопоточными.***
- Появилась масса со-процессоров, которые невероятно эффективно решают отдельные задачи: обработка фотографий, кодирование видео, нейронные движки, облачные вычисления. В конце концов, перенос вычислений на видеокарты.
- Люди наконец начали оптимизировать софт.
- А производителям железа пришлось ежегодно совершенствовать свою продукцию. Ведь просто новый процессор, не позволял продать новый ноутбук.

И вот прошло 10 лет, пока мы с горем пополам производили 14-ти, 10-ти, и даже 7-нанометровые процессоры. Произошло событие, которого все очень долго ждали. Мир перешел на экстремальную УФ-литографию. ***Длина волны лазера скакнула с 193 нм до 13,5 нм, что является крупнейшим скачком за всю историю создания процессоров. Технологию разрабатывали 81 год и только в 2020 она заработала в полную мощь.***

Ключевой момент технологии в том, что она позволит уменьшать техпроцесс вплоть до 1 нм, а это 10 атомов в толщину! И если вы считаете, что это невозможно, это не так. Компания IBM уже в этом году освоила 2 нм. Так, что 1 нм — это лишь дело техники.

Но, а что нас ждет за порогом в 1 нм? Как дальше повышать производительность?

Скорее всего мы полностью откажемся от текущей концепции центрального процессора, основанной на архитектуре Фон Неймана и перейдем на асинхронные нейроморфные процессоры, построенные по подобию человеческого мозга. Кстати, их разработкой занимается тоже Intel.

Источник: Что такое Закон Мура и как он работает теперь? Разбор

7. Архитектура сетей жизни vs Deep Learning

Итак из предыдущего раздела следуют, что ИТ уже десять лет как уперлись в стену - в невозможность эффективно наращивать производительность методом грубой силы, необходим переход к к массово параллельному и асинхронному программированию. Этот процесс на макроуровне давно идет и свидетельство этому - это развитая мировая экономика API, но основной рабочей лошадкой пока еще является автомат выполненный в архитектуре фон Неймана, хотя он уже и обвешан со всех сторон специализированными процессорами и графическими ускорителями.

Это значит, что в ИТ системах человечество все больше и больше будет переходить к архитектурам, которые присущи живым системам в которых ресурсы эффективно распределяются и концентрируются в узлах коммуникационной сети, обеспечивая необходимые производительность, надежность, качество и адаптивность системы, построенной в такой архитектуре. Примеры: API экономика, любой ее сервисный узел спроектированный на микросервисах, любая органическая или социальная система.

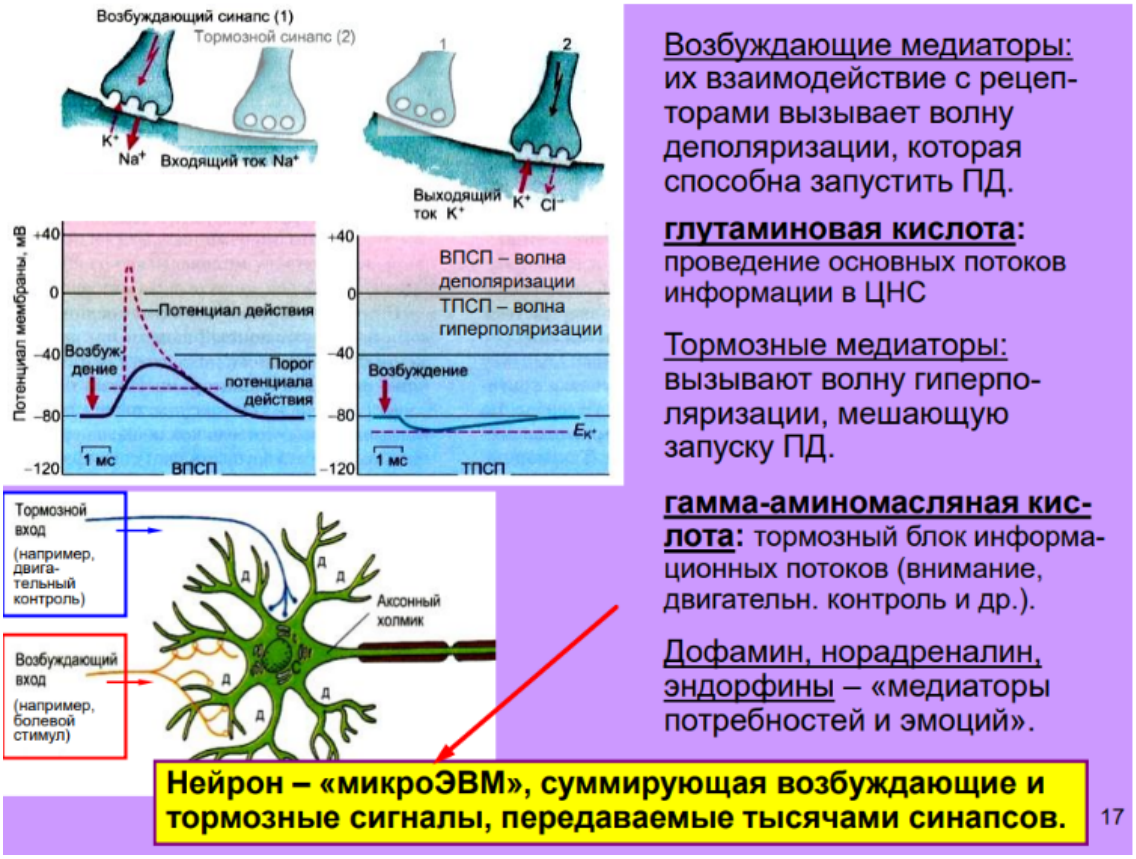
И здесь нам придется догонять Природу и еще многому у нее учиться. Чтобы показать это, сопоставим “наивысшее” достижение нашей цивилизации, так называемый искусственный интеллект, стоящий на технологиях Deep Learning и органические сети нашего мозга по различным критериям эффективности и по архитектуре.

Размер:

1. Размер кремниевого элемента: 10-20 атомов = 5- 10 нм (нм - нанометр 10-9)
2. Размер тела нейрона: 5-130 мкм = 5000 - 130 000 нм.

Казалось бы органические мозги не могут составить никакой конкуренции кремниевым. Но это не так, сопоставление размера кремниевого вентиля и размера нейрона некорректно, так как каждый нейрон является специализированным микропроцессором, а не отдельным его элементом. К тому же содержащим службу технического обслуживания и ремонта - не нужны никакие специалисты DevOps и инженеры технической поддержки. Сопоставлять с нейроном нужно специализированный процессор - например видеокарту или хотя бы одно его ядро.

На приведенном ниже рисунке проф. В. А. Дубынин очень верно схватил самую суть архитектуры мозга, за одним малым исключением - нейроном это не микроЭВМ (он не обладает памятью), нейрон - специализированный коммуникационный микропроцессор. Поэтому сопоставление в лоб размеров нейрона и кремниевого транзистора не корректно.



Поэтому сопоставим эффективный размер видеокарты приходящийся на одно ядро, например MSI GeForce RTX 3070 - ее размеры 14 см x 33.5 см x 3 слота

1. 10 см / 6144 ядер CUDA = 16 мкм
2. Размер тела нейрона: 5-130 мкм

Скорость:

1. Тактовая частота шины - 3.5 ггц, ширина канала - 64- 128 бит
2. Длина аксона - 0.5 мкм - 1.5 м, скорость нервного импульса 0.5-130 м/сек , длительность импульса 150-500 мкс.

Источник: Аксон

1. Ширина канала у кремния : за одну секунду по шине может быть передано 28-64 Гбайт данных.
2. Нейрон: 100-200 импульсов с числом потенциальных коммутаций на синапсах аксона до 20 тысяч.

Энергоэффективность:

1. Видеокарта: MSI GeForce RTX 3070 - 8 ГБ GDDR6X. 17.4 млрд транзисторов. ядра CUDA - 6 144.   - потребляемая мощность - 170 ВТ
2. Нейронов в мозгу 86 Млрд - потребляемая мощность - 10 ВТ

1. 1 ядро CUDA - 27 милливатт (10-3)
2. 1 нейрон - 0.12 нановатт (10-9)

Источник: Анатомия видеокарты - устройство и назначение

Видим что кремниевые мозги недалеко ушли от органики в размерах и чудовищно проигрывают в энергоэффективности. С производительностью сложнее, так как принципы функционирования базовых элементов совершенно разные. Нейрон - это коммутатор сигналов, а ядро CUDA - программный автомат, который тасует данные в ячейках памяти. Напомню, что графические ускорители - это один из важнейших факторов в революции ИИ, произошедшей в 2000 годы - без них ИИ развивался бы намного медленнее и вряд ли нашел бы столь массовое применение, так как именно на видеоплатах производятся параллельные вычисления для обучения нейронной модели.

8. Современные ИТ технологии и социальная деструкция

8.1 Рассмотрим первый аспект развития коммуникаций и ИТ технологий - гиперцентрализация социума, которую эти технологии критически усиливают.

До тех пор пока в нашу жизнь не стали широко внедряться современные средства коммуникации и под их влиянием не трансформировались медиа (важнейшим из искусств является кино :-)) , коммуникативные сети в социуме состояли из двух качественно разных частей:

1. Связи первого рода: Аналог химических сетей в организме - семейные, родовые, общинные, клановые, этнические.
2. Связи второго рода: Аналог нервных сетей в организме - бюрократические сети госаппарата, торговые сети.

Первые, освященные веками, были медленными и требовали широкого согласования и обсуждения, выполнение ритуалов при принятии какого либо решения и отработке изменений в социуме. В рамках этих сетей жизнь двигалась по кругам освященных традицией, менялись лишь участники таких движений.

Вторые были намного более быстрыми. Бюрократические коммуникации были быстрыми и предназначались для трансляции в социум директив верхушки и контроля их выполнения. Они необходимы были для обеспечения замиренного пространства внутри социума, для его внутренней и внешней устойчивости.

Социум, который сумел выстроить эффективные сети второго рода, получал конкурентное преимущество, расширялся и ассимилировал (или уничтожал, или паразитировал на) своих соседей.

Источник: Л.Е. Гринин РАННИЕ ГОСУДАРСТВА И ИХ АНАЛОГИ В ПОЛИТОГЕНЕЗЕ: ТИПОЛОГИИ И СОПОСТАВИТЕЛЬНЫЙ АНАЛИЗ

До появления телеграфа и железных дорог коммуникации были медленные, поэтому государство и бюрократия минимально вмешивалось в структуру и процессы в социуме, опиравшиеся больше на коммуникации первого типа. Экономика состояла из локальных,

слабо связанных рынков. Большую роль, например в экономике России, играли монастырские хозяйства, аккумулировавшие излишки зерна в тучные годы и распределяющие его в худые.

Источник: Андрей Степаненко. О роли скупщика в истории

Ткань социума была живая, сеть была децентрализованная, все ресурсы - власть, владение землей и другими активами, управление хозяйством были распределены по этой сети, власть бюрократии была ограниченной. Географическая структура социума определялась богатством территорий ресурсами.

С появлением телеграфа, радиосвязи, железных дорог появляется товарное производство и начинают развиваться промышленные империи, морская империя Великобритания появилась на столетие раньше, а первое масштабное серийное производство - это Венецианский арсенал. Бюрократия начинает все больше и больше перетягивать управление локальными территориями на себя, но владение землей все еще остается у “местных” и это заставляет с ними считаться. Апофеозом торжества бюрократии был СССР, когда всех лишили собственности и власти, бюрократы могли планировать и директивно управлять огромной территорией из единого центра. Все это кончилось печально, советский голем - великолепный пример мертворожденной социальной структуры, который некоторое время выживал только на богатых руинах русской цивилизации. Низшие слои социума рассматривались в советской модели как ресурс экономики, который нужно было воспроизводить в силу слабой автоматизации и роботизации экономики.

Западная система оказалась более живучей и адаптивной, но сверхцентрализация (т.е. глобализация) ведет ее к такому же бесславному концу. Подробно расписывать не буду, если интересно внизу ссылки:

Источник: Глобализация. Есть ли свет в конце тоннеля ? - 1

Источник: ВОСПОМИНАНИЯ О БУДУЩЕМ. Вопросы и комментарии - 2.

Резюмирую:

С появлением ERP систем и глобальных ИТ коммуникаций государственная и корпоративная (ТНК) бюрократия получила инструмент мониторинга и управления в любой точке мира (где их власть не ограничена местными сообществами и традициями).

Та форма “искусственного интеллекта” и машинного обучения которая сегодня активно развивается, это еще один “полезный” инструмент в руках глобальных бюрократов и их хозяев, еще одна дополнительную возможность - устраняется человек-посредник в мониторинге бизнес среды и социальных сетей по формальным признакам, которые раньше невозможно было алгоритмизировать (социальные графы, намерения фраз, суммаризация

тестов, визуальные признаки), такой мониторинг теперь можно сделать максимально автоматизированным.

8.1 Рассмотрим второй аспект развития коммуникаций и ИТ технологий - разрушение и модификация социальных связей первого рода .

Кратко пройдусь и по данному вопросу. До появления массовых медиа производство контента было чрезвычайно трудозатратным, контент тщательно готовили к распространению, в основном письменное, да и печатное слово было доступно только верхним слоям социума. Нижние получали его в виде проповедей священников или как профессиональное тексты.

После появления коммерческих издательств, кино, радио и телевидения производство контента поставили на широкую ногу, качество его в основном значительно снизилось, но оставался сильный контроль за его содержанием поскольку сети его производства и распространения были все еще чрезвычайно централизованными и иерархическими (за исключением газеты “Искра” :-ж).

С появлением массовых ИТ платформ и коммуникаций ситуация радикально изменилась и коммуникации стали децентрализованными, производство “контента” стало дешевым и простым. Бюрократия драконовскими методами (различные законы и реестры регламентирующие содержание - и часто отнюдь не в массовых коммуникациях, а просто переписку в небольшом сообществе, массовый мониторинг и троллинг) пытается удержать под контролем гиперинформационный поток, но пока получается не очень. Джин анархии выбрался из бутылки.

Бюрократия вынуждена отказаться от информационных методов управления социумом (через модерируемый ею контент и программирование массового сознания на этом контенте), к управлению через директивы, алгоритмы и централизованные ИТ платформы. . Благодаря дешевизне производства и дистрибуции контента и всеобщей связанности социума через мобильные и компьютерные устройства, социальные сети и платформы распространения и потребления контента, благодаря таргетированной рекламе и другим инструментам появилась возможность быстро масштабировать любые культуры и идеи на весь мир, в том числе вирусы, навязывая их социуму. То что раньше было приемлемо в нишевом сообществе теперь легко навязывается всем как норма. Возникла даже новая профессия - инфлюенсеры (от англ. influence — «влиять»), а также культура отмены/деплатформизации (например, удаление президента США из Твиттера).

Огромная опасность из-за гиперинформационного, анархического и безграничного характера современных коммуникаций возникает и в сфере воспитания, так сказать, молодого поколения. Воспитать нормального человека сегодня можно только жестко ограничивая его доступ к этому информационному потоку, по крайней мере на этапе зарождения и становления личности.

А это значит, что необходим возврат к общинной форме социума, процессы жизнедеятельности в которой жестко регламентируются, что полностью противоречит основам рынка и общества гиперпотребления.

8.3 Реальная опасность тотальной деструкции и гибели социума.

Совсем коротко….

Современные ИТ технологии, в том числе нейросети, - это инструмент. Инструмент который может использоваться различными кланами и кликами с различными целями, как впрочем и любые другие знания, технологии и инструменты.

Но ИТ коммуникации вторгаются в самое сердце социальной ткани и последствия для всех нас могут быть печальными. Желание бюрократов придать субъектность нейросетям, это просто желание снять с себя ответственность за результаты своей деятельности. А результаты этой деятельности печальны во всем мире, глобальные и местные кланы в борьбе за Власть и ресурсы создали гиперцентрализованные, мертвые систем убивающие живой социум.

9. Заключение

В силу физических ограничений указанных в разделе 6, никакая централизованная система не способна учесть всего разнообразия культур, норм, правил и традиций локальных сообществ. Никакая централизованная система с самым продвинутым искусственным интеллектом не способна принимать эффективные и оптимальные решения для заданной локации.

Попытку навязать всем ИИ стоит рассматривать с точки зрения того, кто является выгодоприобретателем и кто несет ответственность за внедрения “черного ящика” и применяемые им решения на основе централизованных моделей, которые всегда не соответствуют реальности. Будет ли ИИ встраиваться в локальные культуры, усиливать и объединять их или это будет средство манипулирования массами, оболванивание и в конечном счете их уничтожения ? Очень удобно, еще вчера все это оправдывалось единственно верным учением (коммунизм) , сегодня все будет оправдываться именем мифического искусственного интеллекта.

Восхищаться же пока особо нечем, по сравнению с Природой решения крайне неэффективные и энергоемкие, впрочем с наведением ракет и снарядов вполне справляются.

Современные наработанные массовые инструментарии и модели не дают особых надежд на то что будут построены по настоящему самообучающиеся системы, выявляющие смыслы и работающие с ними. Все финансирование направлено на развитие наработанных инструментов, рассматриваемых как инструментарий Власти и захвата влияния над Миром.

10. Дополнение: Асинхронное логика и асинхронное программирование

10.1

Пример синхронного выполнения с объединением двух файлов через чтение и запись в другой файл с помощью синхронных функций readFileSync и writeFileSync, время выполнения кода = время чтения файла1 + время чтения файла2 + время записи нового файла:

```
import fs from 'fs';

const content1 = fs.readFileSync('./myfile1', 'utf-8');

const content2 = fs.readFileSync('./myfile2', 'utf-8');

fs.writeFileSync('./myfile-copy', content1 + content2 );
```

10.2

Пример асинхронного программирования. Задача объединения двух файлов с помощью асинхронных функций readFile и writeFile сводится к последовательному выполнению трех операций, так как записать новый файл мы можем лишь тогда, когда прочитаем данные первых двух. Упорядочить подобный код можно лишь одним способом: каждая последующая операция должна запускаться внутри callback’а предыдущей. Тогда мы построим нужную цепочку вызовов:

```
import fs from 'fs';

fs.readFile('./first', 'utf-8', (_error1, data1) => {

  fs.readFile('./second', 'utf-8', (_error2, data2) => {
```

```
    fs.writeFile('./new-file', `${data1}${data2}`, (_error3) => {
      console.log('File has been written');
    });
  });
});
```

10.3

Данный код гораздо эффективнее примера с синхронным чтением, так как во время чтения и записи файла процессор может выполнять другие команды, но его можно сделать еще более эффективным если читать оба исходных файла параллельно:

```
import fs from 'fs';

const state = {
  count: 0,
  results: [],
};

const tryWriteNewFile = () => {
  if (state.count !== 2) {
    return; // guard expression
  }
  fs.writeFile('./new-file', state.results.join(''), (error) => {
    if (error) {
      return;
    }
    console.log('finished!');
  });
```

```
};

console.log('first reading was started');

fs.readFile('./first', 'utf-8', (error1, data1) => {
  console.log('first callback');
  if (error1) {
    return;
  }
  state.count += 1;
  state.results[0] = data1;
  tryWriteNewFile();
});

console.log('second reading was started');

fs.readFile('./second', 'utf-8', (error2, data2) => {
  console.log('second callback');
  if (error2) {
    return;
  }
  state.count += 1;
  state.results[1] = data2;
  tryWriteNewFile();
});
```

Так как этот код асинхронный, результат работы каждой функции можно получить лишь внутри колбэков. Причём, порядок запуска колбэков мы не можем знать — всё зависит от того, какой файл прочитается быстрее. ***Для отслеживания состояния выполнения этих операций придётся ввести глобальное состояние (относительно этих операций), через которое мы будем отслеживать завершенность и в котором сохраним данные.*** И только когда все операции завершились — запишем новый файл. Кроме того, нам нужно

чётко разделять данные первого и второго файлов, так как запись в новый файл (в отличие от чтения) должна происходить в определенном порядке.

Источник: Асинхронное программирование. Параллельное выполнение операций.

Приведенный код в примерах можно сделать гораздо более компактным и наглядным используя синтаксис оператора async/await, ***но здесь специально выбран самый ранний синтаксис, чтобы подчеркнуть значение функции обратного вызова. Это гораздо более значимый феномен, чем рекурсия, которая оставляет нас в парадигме последовательного программирования архитектуры фон Неймана.***

10.4

Listener - прослушиватель:

Прослушиватель - это функция, которая связана с некоторым событием. Проще всего это продемонстрировать на примере пользовательского интерфейса в браузере:

```
<script>

 function countRabbits() {

   for(let i=1; i<=3; i++) {

     alert("Кролик номер " + i);

   }

 }

</script>

<input type="button" onclick="countRabbits()" value="Считать кроликов!">
```

В данном примере при каждом клике на кнопку вызывается функция которая выводит три сообщения о количестве кроликов. ***Таким образом листенер - это тот же самый Callback, но он связан не с функцией, а с некоторым элементом, который обычно предназначен для связи в внешним миром.***

Источник: Введение в браузерные события

10.5

На web-сервере главные прослушиватели располагаются в модуле маршрутизации запросов (которые сервер получает извне).

```
const express = require('express'),

app = express());

const host = '127.0.0.1';

const port = 7000;

app.get('/api/users', (req, res) => {...});

app.post('/api/users', (req, res) => {...});

app.put('/api/users', (req, res) => {...});

app.delete('/api/users', (req, res) => {...});

app.listen(port, host, () => console.log(`Server listens http://${host}:${port}`))
```