Карпов В.Э.

ChatGPT и аналогичные “болталки”: от имитации диалога к имитации разумности

Предисловие автора

В 2023 году началась настоящая истерия по поводу невиданных успехов ИИ, который наконец-то появился и может теперь решать все мыслимые и немыслимые задачи. Речь о ChatGPT от OpenAI. Причем, чем более далек от технической области был человек, тем с большим энтузиазмом он рассуждал об этом. Попытки хоть немного помочь разобраться с этим вопросом, убедить критически взглянуть на суть происходящего, привели к выводу, что надо просто рассказать о том, как все это устроено. Причем рассказать языком, понятным гуманитарию, обладающему познаниями в области математики и информатики хотя бы на школьном уровне. В том же 2023 году была прочитана лекция на майском семинаре "Этические проблемы ИИ" под названием "Болталка GPT. От имитации диалога к имитации разумности". Далее по мотивам этого выступления был написан текст, ставший "технической" частью публикации

Огородов Д.В., Карпов В.Э. ChatGPT и аналогичные “болталки”: от имитации диалога к имитации разумности. Информационные технологии в юридической деятельности: теле-право: монография. М.: Проспект, 2024. С. 47–75.

Этот текст, являющийся вполне самодостаточным трудом, и приводится ниже. Текст действительно написан для гуманитариев и действительно не требует для понимания специальных знаний. Я постарался объяснить все это а) доходчиво, "на пальцах" и б) стараясь быть все-таки технически корректным.

Оглавление

1. Как устроены “болталки”

Цель этого раздела – дать общее понимание того, как устроены пресловутые генеративные системы, т.е. системы, которые по запросу человека что-то выдают: текст, музыку, изображения, программы и т.п. В основном мы будем говорить о том, что представляют собой “говорилки” или “болталки” – программы, умеющие “гладко строить фразы” (чат (chat) – ‘это и есть болталка, ничего уничижительного). И не более того. Иное дело, что это – сложные, качественные болталки, главной деталью которых являются сложные языковые модели. Если такое понимание будет получено, если станет понятно, что внутри таких систем, то многие вопросы, кажущиеся сегодня актуальными, отпадут сами собой, просто потеряют смысл. По крайне мере, рассуждения о влиянии, об опасностях и вызовах, о рынке, о перспективах, о философских и правовых аспектах и многое другое предстанут в ином свете.

Дальше будет минимум математики и совсем немного информатики. Неизбежной платой за популярность стиля изложения стали определенная некорректность, нестрогость, неточность, неполнота и отчасти эклектичность. Но когда текст рассчитан на гуманитарную аудиторию, на это приходится идти (в надежде на то, что коллеги-технари этот текст не прочитают).

Хороших болтаток не так много. Это прежде всего пресловутый ChatGPT [GPT-3.5, 2023], GigaChat от Сбербанка [AIRI, 2023a] и YaChat от Яндекса [YaGPT, 2023]. Их мало в силу дороговизны их производства: нужно очень много ресурсов. Почему – станет ясно из дальнейшего изложения.

План изложения таков: сначала будет немного об истории вопроса (это важно для понимания того, что основы были заложены еще давно). Далее будет рассказано о сердце этих болталок – языковых моделях и о том, как работают системы, умеющие складно выстраивать слова. В конце поговорим немного о том, как “пишутся” картины.

1.1. Предыстория вопроса

В 1950 г. Алан Тьюринг в работе “Вычислительная техника и интеллект” [Turing, 1950] сформулировал свой знаменитый тезис, определяющий то, какая система может быть названа интеллектуальной. В упрощенной форме этот тезис сводится к следующему: *“Если эксперт, проводя диалог с анонимным собеседником, ошибочно принимает ЭВМ за человека, то такую искусственную систему следует считать интеллектуальной”*. И это воспринималось бесспорным в силу своей фундаментальности. Правда, философы подозревали это и раньше. Известный деист и энциклопедист Дени Дидро в своей Мысли

№ XX в 1746 г. писал: “… не по движениям и звукам, а по содержанию идей, по логической последовательности фраз и связи рассуждений должно заключать, что данное существо мыслит; если бы нашелся попугай, отвечающий на все вопросы, я заявил бы не колеблясь, что он мыслящее существо...” [Дидро, 1986].

Однако в 1966 г. случился казус: то ли в шутку, то ли всерьез, но Дж. Вейценбаум из лаборатории ИИ МИТ вместе со своим аспирантом пишут программу программа “Элиза” (Elize). Эта программа, реализуя т.н. фатический диалог, напрочь опровергла тезис Тьюринга.

Фатический диалог – это видимость беседы, общение без смысла, без понимания. Основные принципы ведения такого “разговора”: (1) построение ответов на основе чисто формального преобразования услышанного и (2) перефразирование сообщения человека, превращение утверждений в вопросы.

Принцип действия программы прост: выдается первоначальная фраза (затравка) и далее происходит сбор ответов. Построение дальнейшего взаимодействия происходит на основе образцов, которые содержатся в собранных предложениях (метод *сопоставления с образцом*, см. [Поспелов, 1982]).

Образец из предложения	**Ответ машины**
...я испытываю…	Сколько времени вы это испытываете?
...гнев...	А что вы испытываете сейчас?
...мать...	Расскажите подробнее о своей семье
...любовь...	Вас пугают эмоции?
...ад и т.п...	Пожалуйста, не употребляйте такие слова
...секс...	Это представляется важным

Вокруг этой программы и ее вариаций существует много мифов и легенд, на основе такого фатического диалога создавались программы психологической помощи, строились различного рода теории (что, надо сказать, несколько удивляло и пугало самого Вейценбаума, см., например, [Вейценбаум, 1982]). Так, в 1977 г. программа Parry (автор – Кеннет Колби (Kenneth Colby)) прекрасно осуществила мистификацию психиатров: в ходе эксперимента большинство из них сочло, что вело диалог со вполне реальным параноиком. А в 1996 г. очередной клон “Элизы” (автор – Грег Гарви (Greg Garvey)), использовался для моделирования католического исповедника. Более того, с 1990 г. много

лет проводился ежегодный конкурс Хью Лебнера (Hugh Loebner) на наиболее интеллектуальную программу с призовым фондом в $100’000. Серебряная медаль и $25’000 полагались программе, которой удастся мистифицировать, по меньшей мере, половину жюри. А золотая медаль и $100’000 предназначалась программе, способной пройти т.н. ”строгий тест Тьюринга”, в ходе которого реплики арбитра будут вводиться не с клавиатуры, а с помощью микрофона и видеокамеры [The Loebner Prize, 2018]. Кстати, глядя на это безобразие великий Марвин Мински объявил альтернативный конкурс: “*$100 будет выплачено тому, кто найдет способ положить конец "контрпродуктивной ежегодной шумихе"”*.

Подобного рода программы (чат-боты) иногда были весьма изощренными, они использовали базы справочных материалов, применяли разные технологии обработки текста, но в целом их создание было крайне трудоемким занятием групп энтузиастов. И так было до тех пор, пока за это дело не взялись всерьез.

1.2. Постановка задачи

Итак, представим себе, что перед нами стоит задача создания “болталку” – системы, способную “гладко строить фразы”. Или даже тексты. Или вести диалог.

В качестве способа ее решения выберем реализацию давней мечты человечества – загрузить в машину огромный массив текстов (“Не пропадать же добру”). При этом желательно, во-первых, свести участие человека в процессе создания такой системы к минимуму, т.е. чтоб машина училась сама (для того и будет ей дан весь накопленный текстовый массив). И во-вторых, не привлекать к этому процессу лингвистов (долго, сомнительно, сложно).

Основные “технические” идеи тоже естественны и понятны:

1. Строить ответы надо постепенно, формируя последовательно наиболее подходящие и “осмысленные” слова шаг за шагом. (формировать ответ сразу, целиком сложно. Для этого надо было бы построить модель, разбираться с семиотикой и т.п., а мы договорились лингвистов не использовать).

2. Будем полагать, что “смысл” текста определяется взаимным расположением слов в предложениях (иного варианта без лингвистов просто не остается, все , что есть – это просто играть со словами, как набором букв, и со статистикой).

3. Слабость применяемых механизмов преодолеть объемом обучающей выборки (ошибки нивелируются, переход от количества к качеству и т.п.).

Итак, начинаем создавать. Предположим, что есть некая начальная фраза, а нам нужно, чтоб система построил ее продолжение. Скажем, введя *“У попа была собака.”* Хорошо обученная система должна бы выдать продолжение типа “*Он её убил*”. Для этого мы создадим некий Большой Черный Ящик (БЧЯ), куда загрузим много текстов, проанализировав которые система и выдаст ответ. Условно это изображено на Рис. 1.

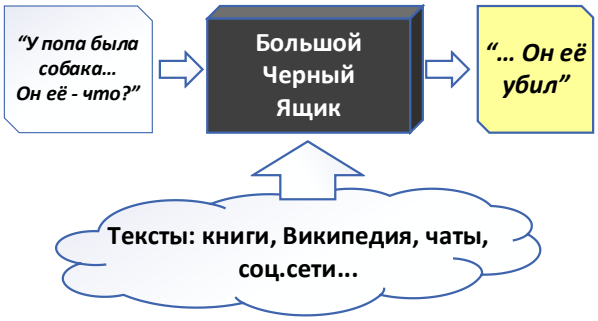


Рис. 1. Концептуальная схема

Возникает вопрос, что будет происходить с загруженными в БЧЯ текстами.

1.3. Игра в слова

1.3.1. Формальный лингвистический подход

Если бы мы согласились привлечь лингвистов, то они начали бы анализировать фразу. Для начала они выделили бы из *"У попа была собака, он ее любил…“* лексемы и построили бы их характеристики, получив, например, такое:

"Собакой" => { found:+, common: од, lemma: СОБАКА, Grammems: "С жр,тв,ед" }

Далее начались бы этапы синтаксического, а то и семантического (смыслового) анализа, см. Рис. 2.

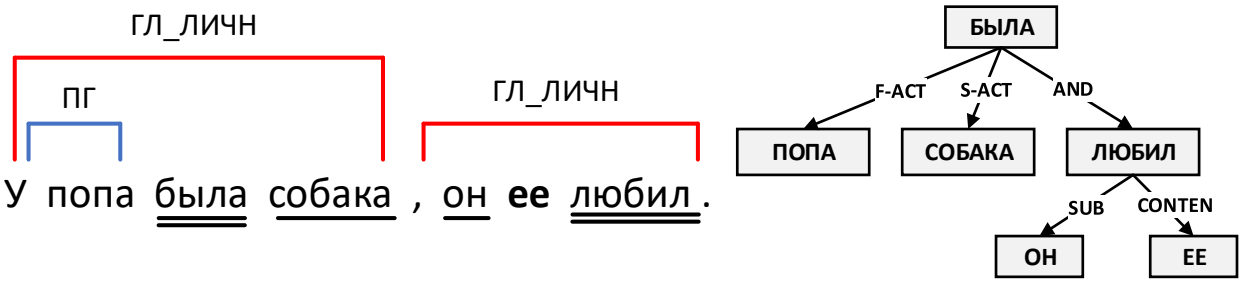


Рис. 2. Примеры результатов анализа с сайта aot.ru [AOT, 2023]

А потом дошли бы и до прагматического анализа. Это все конечно, верно и формально правильно, но слишком ресурсозатратно и трудоемко.

Загружаемые в БЧЯ тексты все равно должны как-то укладываться, формализоваться. Пусть не так, как у лингвистов, но что-то должно быть. Это что-то, определяющую форму представления текста внутри БЧЯ называется языковой моделью.

1.3.2. Языковые модели

Языковая модель – это алгоритм или способ кодирования и хранения текстов. Основная задача языковой модели – определить наиболее вероятное слово (между другими словами, в начале, в конце – не важно). Таких моделей много. Одни просты, другие изощренны. *GPT* – это тоже название модели. Если на эту модель повесить интерфейсные модули, то может получиться “болталка” – те же ChatGPT-4, GigaChat или YaChat. Но главное – это понять, что представляет собой эти языковые модели. Начнем с простых.

Языковая модель “Мешок слов (Bag of Words)”. Мы приводим ее для иллюстрации того, что желание поиграть со статистикой без учета смысла приводит к странным результатам.

Итак, пусть имеются следующие предложения (наш корпус текстов, на котором происходит обучение): (1) “ПОП ИМЕТЬ СОБАКА”, (2) “СОБАКА СЪЕСТЬ МЯСО”, (3) “ПОП СОБАКА УБИТЬ”.

Очевидно, что это – наш исходный текст про попа и собаку, но только несколько упрощенный. Далее можно ”играть” в простую статистику, вычисляя частоты появления слов, соотносить количество повторений в предложениях к общему количеству предложений и т.д. Пропустим всю эту разнообразную математику, представив сразу самый значимый результат – таблицу, в которой определена информационная значимость каждого слова в предложении. Это – т.н. параметр TF*IDF, который рассчитывается, исходя из значений количества повторений слова в предложении, количестве слов в предложении, количестве предложений, содержащих слово.

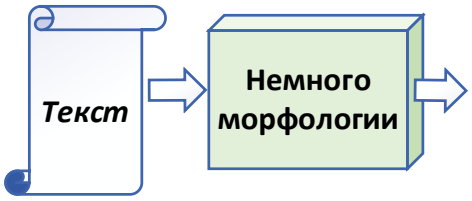
Табл. 1. Важность слов. TF*IDF

	“ПОП”	“ИМЕТЬ”	**“СОБАКА”**	“СЪЕСТЬ”	“МЯСО”	“УБИТЬ”
Предл.1	0.10	0.27	**0.00**	0.00	0.00	0.00
Предл.2	0.00	0.00	**0.00**	0.27	0.27	0.00
Предл.3	0.10	0.00	**0.00**	0.00	0.00	0.27

Обратите внимание, что ключевое слово “собака” с точки зрения такой модели имеет нулевую важность. Иными словами, важность считается, а смысла нет.

Псевдоассоциации. Эта языковая модель тоже относится к категории статистических, но в ней делается попытка получить какую-то осмысленность. В основе модели лежит предположение, что взаимное расположение лексем (слов) в тексте может что-то значить. Чем ближе друг к другу слова в предложении, тем сильнее их

“ассоциативная” связь. Для удобства можно определять степень близости числом от 0 до 1. Если взять некий текст и посчитать эти степени близости между всеми парами слов, то могут получиться интересные результаты. Для эффективности в этой модели используется морфологический анализ, т.е. для каждого слова указываются его характеристики – часть речи, число, род, падеж и т.п. В итоге мы получаем большую таблицу, строки и столбцы которой соответствуют словам – нашему словарю, а элементы таблицы – степени близости.



Слово	*Женщина*	*Любить*	*Врач*	*Собака*	...	*Война*
Женщина	0	0.2	0.01	0.0001		0.12
Любить	0.2	0	0.01	0.04		0.007
Врач		0.01	0	0.006		0.081
Собака	0.0001	0.04	0.006	0		0.005
...						
Война	0.12	0.007	0.081	0.005		0

Рис. 3. Псевдоассоциации

Зная морфологические характеристики, можно определить, с чем ассоциируются у того или иного автора интересующие слова. Например, слово “женщина”. Проанализировав “Поэтический сборник 97” Окуджавы Б.Ш., мы увидим, что с этим словом наиболее ассоциируются глаголы “любить”, “плакать”, “подарить”, “красть”, “гореть”, “потерять”, “очаровать”, “излететь”. А у Пришвина М.М., согласно “Кладовой солнца”, – это “бывать” и “нападать”. Ассоциации Шукшина В.М. в его паре “До третьих петухов” + “Печки-лавочки” уже не так поэтичны: “смешливый”, “беременная”, “болтливость”, “врач”. Могут быть и более сложные варианты. Например, можно определять, что ассоциируется с парой слов *“жизнь+смерть”*. У Окуджавы – это “короткий”, “прекрасный”, “странный”, “славный”, “геройский”, у Пришвина – “дикий”, “несчастная”, а у Шукшина объединяет эту пару слово “мелочь”.

Псевдоассоциации способны и порождать тексты. Пусть нам необходимо разукрасить или разнообразить некий текст “*От огорчения умер имярек*”, *”…И, смеясь, пошел имярек прочь…”*. Построив матрицу псевдоассоциаций на основе “Муму” Тургенева И.С., мы определим, какие наречия наиболее близки к словам “умер” и “смеясь”. В результате получим предложения *“От огорчения СКОРО умер имярек”* и *“…И, ТЯЖЕЛО смеясь, пошел имярек дальше”*. По такому же принципу можно создавать самые настоящие “тургеневские” (согласно “Муму”) фразы. Поверхностный синтаксический анализ текста “Муму” показывает, что самая частая синтаксическая

структура – это *(П→С→Г→ПРЕДЛ→(П, С))*. Далее просто. Берем начальное слово (прилагательное *П*), ищем наиболее близко ассоциированное с ним существительное *С*, потом глагол *Г* и т.д. Иногда получается вполне убедительно:

“Богатырская сила подействовала через крепкую думу”, “Богатырская сила подействовала через старшую приживалку”, “Такова ходила молва о богатырской силе немой”, “Такова ходила сила через старшую приживалку умильную”, “О силе второпях и между”.

Прежде, чем рассматривать другие, более осмысленные модели, сделаем важное отступление. Необходимо описать базовый механизм, который применяется в более изощренных языковых моделях. Это – те самые пресловутые искусственные нейронные сети. Именно они и реализуют вычислительные процедуры языковых моделей.

1.3.3. Основной инструмент. Искусственные нейронные сети

Это – основной инструмент для работы с языковыми (да и не только) моделями. О нейронных сетях (ИНС) говорится очень много и разнообразно. Вплоть до того, что между терминами искусственный интеллект и ИНС ставится знак равенства. Но мы попробуем донести здесь одну простую мысль: если мы разберемся в том, как устроена ИНС, то многие вопросы отпадут сами по себе. История НС очень важна и показательна. Опустив предысторию вопроса (1943 г., МакКалок У. и Питтс У.), можно считать, что начало ИНС было положено в 1957 г., когда Розенблатт Ф.Ф. пишет рабочий отчет “Персептрон. Воспринимающий и познающий автомат” [Rosenblatt, 1957]. В отчете описывалось устройство, состоящее из искусственных нейронов и решающее задачу распознавания. Так, как делает это, по мнению Рознблатта, мозг. В те далекие времена исследователи старались апробировать свои идеи не только не бумаге или в программном виде, но на реальном железе. Из уважения к Розенблатту на Рис. 4 приведена общая концепция персептрона и схема его элемента.

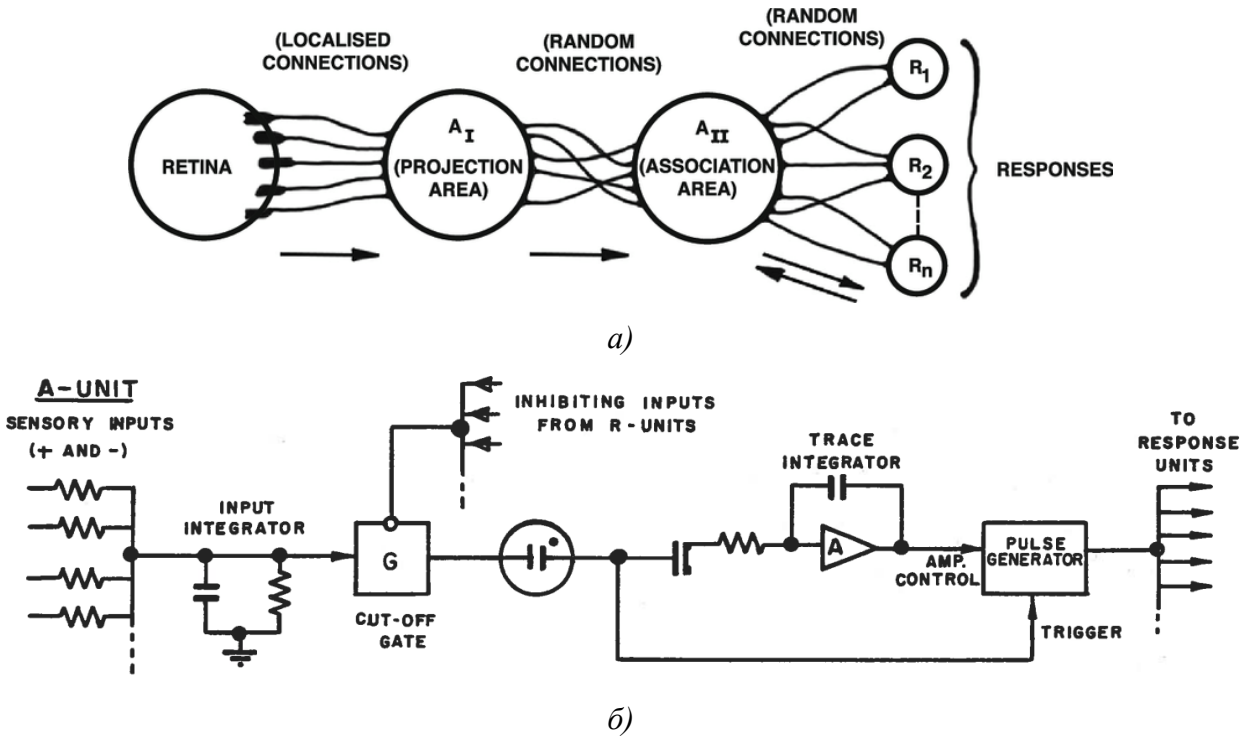


Рис. 4. а) первоначальная концепция персептрона Ф. Розенблатта: искусственные нейроны имитируют работу мозга, преобразуя входные данные на сетчатке в ответы [Rosenblatt, 1958], б) схема ассоциативного элемента [Rosenblatt, 1957]

Если отбросить рассуждения об имитации работы мозга и т.п., мы получим в итоге следующее. ИНС – это хороший, мощный инструмент, пригодный к применению в самых разных областях (обучение, распознавание, управление и т.п.). При этом инструмент устроен крайне просто. Есть набор входных величин (чисел) X и есть набор выходных чисел *Y*. Между *X* и *Y*, в скрытых (внутренних) слоях расположены искусственные нейроны, которые делают очень простую операцию – они суммируют приходящие на их входы сигналы. Результат суммирования отправляется дальше, на вход следующего элемента. Весь фокус в том, что это не просто суммирование, но суммирование с весами: каждый входной сигнал умножается на некий коэффициент. На Рис. 5 эти коэффициенты обозначены как w. Персептрон работает так. Берется набор входных чисел *X*, подается на вход персептрона. Дальше сигналы умножаются и складываются, проходя через все элементы, и на выходе мы получаем набор чисел *Y*. Если это то, что нам нужно, то все хорошо. А если на выходе не то, что мы ожидали, то изменяются значения коэффициентов *w* так, чтобы персептрон в итоге отреагировал как надо.

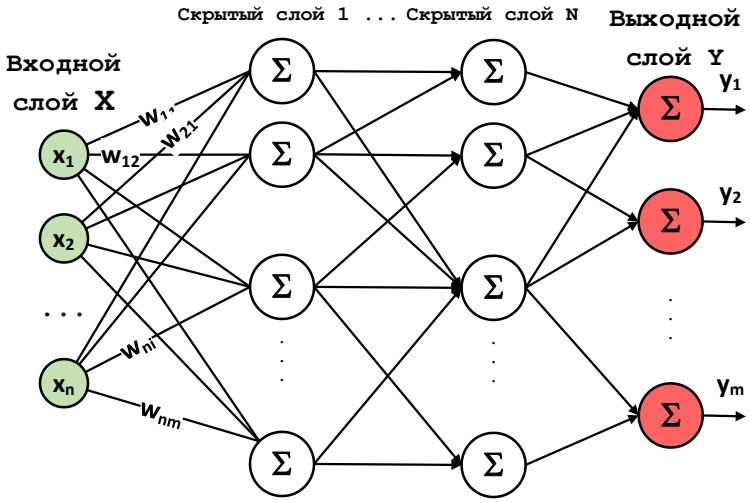


Рис. 5. Формальная схема персептрона

Это и есть обучение персептрона по примерам: мы подаем на вход системы много пар *(X,Y)* и подстраиваем веса так, чтобы персептрон на каждый входной набор реагировал нужным образом. Например, входом *X* могут быть изображения, и тогда каждый элемент входного набора x_i – это точка с картинки (и элементов *X* тогда очень много). А на выходе *Y* мы ожидаем увидеть числа, которые будем интерпретировать как, например, категория, к которой относится это изображение (y_1 – собака, y_2 – кошка и т.п.). Итак, обучающая выборка здесь – это множество изображений (*X*), каждое из которых снабжено заранее указанием того, как надо отреагировать на этот вход (*Y*). Далее мы возьмем неизвестную (на ней мы не обучались) картинку, подадим на вход, а персептрон выдаст ответ, тот, который определен сформировавшимися в процессе обучениями весами.

Строго говоря, персептрон Розенблатта – это т.н. нелинейный аппроксиматор. Он обучается на каждый входной набор отвечать нужным образом, т.е. преобразовывать *X* в *Y*. Архитектур таких систем существует огромное количество, и называются они уже по-разному, и связи между элементами могут идти самым причудливым образом, разветвляясь, возвращаясь, пропуская что-то и т.п. А главная проблема – это собственно настройка весов. Эти веса и есть т.н. параметры системы. Их много, и их подбор занимает огромное количество времени. Эта трудоемкость и привела к тому, что расцвет этих технологий наступил тогда, когда появились суперкомпьютеры, способные обучать сети с очень большим числом параметров.

1.3.4. Начальный этап. Представление слов

По-прежнему будем разбирать наш текст про попа и собаку. Для начала разбиваем его на предложения:

sentences = ['у попа была собака.', 'он ее любил.', 'собака съела кусок мяса.', 'поп ее убил.']

Далее составляем список лексем:

words = {'.': 0, 'ее': 1, 'собака': 2, 'убил': 3, …, 'любил': 8, 'он': 9, 'была': 10, 'попа': 11, 'у': 12}

И здесь возникает первая трудность: слов получается очень много. Особенно, если учесть, что мы берем слова со всеми их различными словоформами (по-прежнему, лингвистика – это не наше). Словарь даже 500’000 слов будет слишком велик, поэтому используется следующий трюк. Слово разбивается на фрагменты – наиболее часто повторяемые последовательности букв. Это даже не слоги. Фрагменты далее кодируются и слово превращается в набор чисел. Эта процедура называется токенизацией.

“токенизация”=['ток','ени','зация']=[3041,337,4504]

Теперь возникает следующий вопросы – что делать дальше с этими словами и как их все-таки хранить?

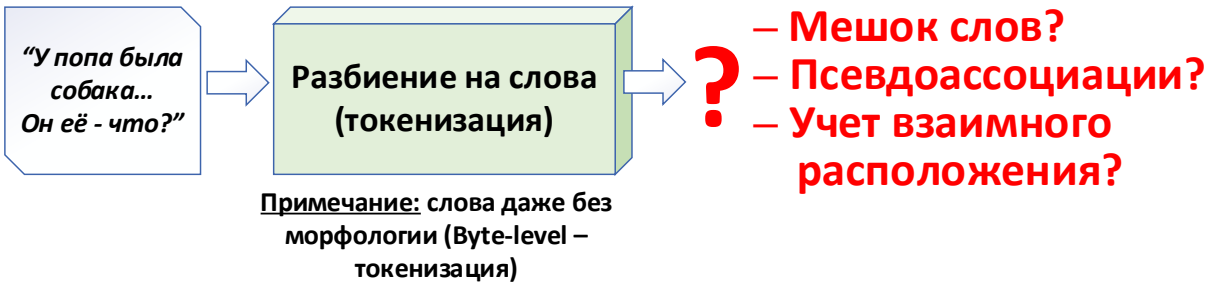


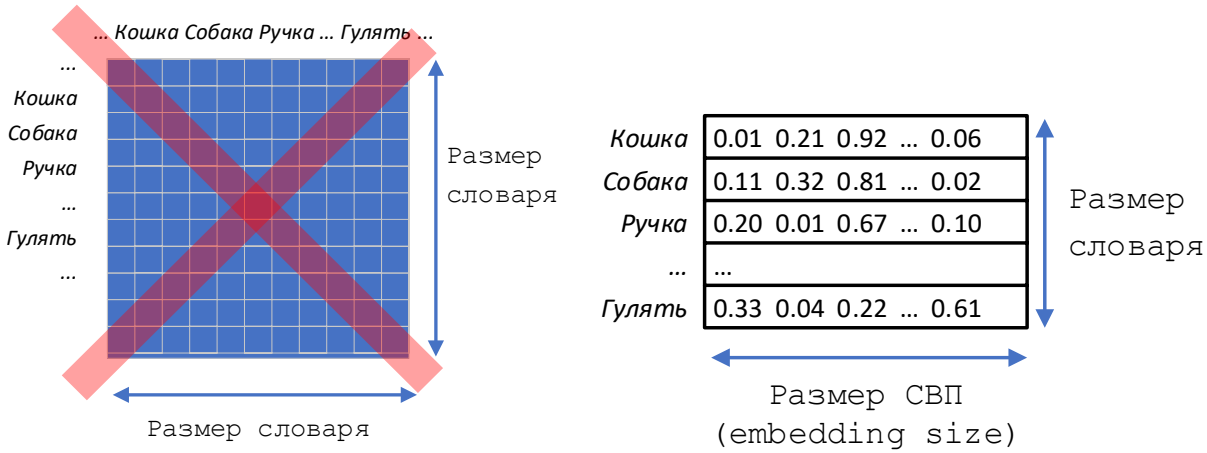
Рис. 6. Начальная процедура анализа текста

Как псевдоассоциации, так и модель “Мешок слов” крайне неудобны. Во-первых, получаются слишком большие словари (приняв для русского языка количество слов N=500'000, мы должны хранить N*N=500'000*500'000 чисел). Во-вторых, мы по-прежнему не понимаем, где мог бы появиться смысл, как учитывать контекст и т.п.

Удобное представление слов. Было бы хорошо представлять слова в виде набора (вектора) чисел.

1. С числами удобно работать.
2. Числа (элементы вектора) могли бы нести какой-то смысл, определять характеристики слова.

Такое “волшебное” кодирование называется часто эмбеддингом (*embedding*), но вместо этого жуткого термина мы будем использовать иной – сжатое векторное представление (СВП).



а) *б)*

*Рис. 7. Словари. а) плохой вариант с прямым хранением слов, размер = N*N? б) хранение в виде сжатого векторного представления, размер = N*длина_вектора_СВП*

Пофантазируем дальше. Если бы мы могли создать СВП для каждого слова, то:

1. Было бы компактно (размер = 500'000*размерность_вектора_СВП, при этом количество элементов в СВП было бы небольшим, например, порядка нескольких десятков).
2. Мы бы знали о самом слове многое (элементы СВП что-нибудь да значили бы, представляли бы какие-нибудь характеристики).
3. Можно было бы делать с такими словами в виде СВП что-то полезное: определять близость слов, вычислять, находить что-то и т.п.)

Представим себе, что СВП содержит числа-признаки, определяющие некоторые осмысленные характеристики слова типа живое ли оно, является ли человеком, каков пол, какая часть речи и т.п. И тогда мы получим очень красивую языковую модель, см. Рис. 8.

Примечание. Все почему-то любят этот пример, в котором можно "вычислять", вычитая из королей мужчин и складывая перья. Поэтому автор и приводит эту картинку ниже.

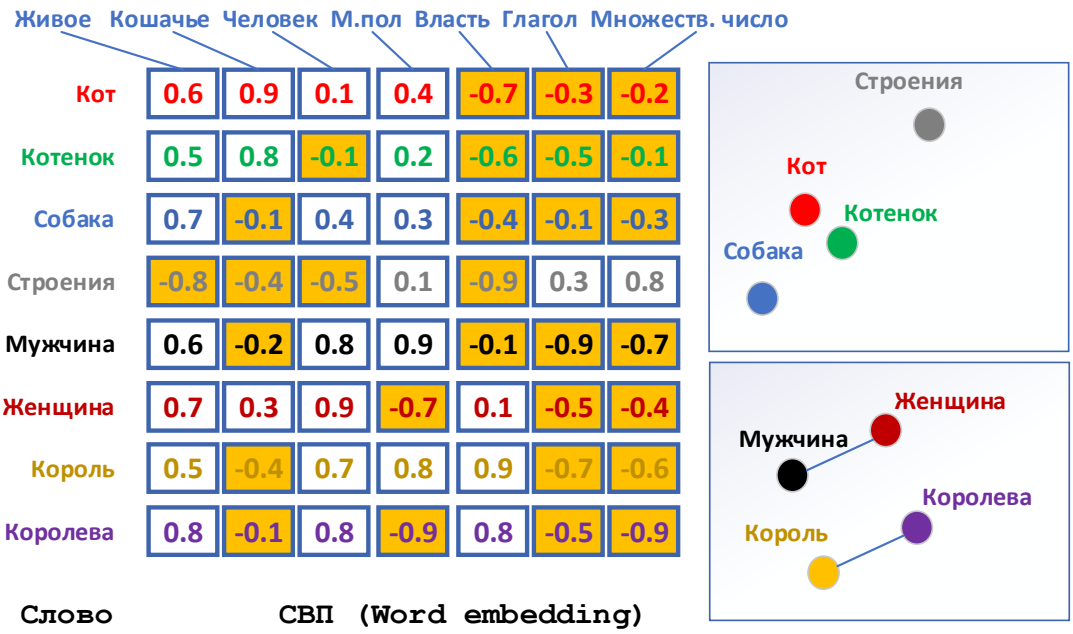


Рис. 8. Пример представления слов в виде СВП. Слева – сам словарь, справа – гипотетическое изображение СВП как точек в некотором признаковом пространстве

Как только мы получаем красивую и удобную математическую форму (вектор – это красиво), мы тут же можем начать вычислять. Например, такая форма позволит нам осуществить поиск слова, вводя запросы в виде “формул” и получая самую близкую итоговую точку-слово:

”король” - “мужчина”+”женщина” => “королева”

”двуногое” - “перья”+”ногти” => “человек”

Тем и хороши векторы, что их можно складывать и вычитать. Проблема только в том, что вручную создать СВП для каждого слова – это неподъемная задача.

1.3.5. Как формируется СВП

Итак, наша задача заключается в том, чтобы создать систему, которая сама сформирует СВП, т.е. определит значения и смысл элементов СВП, пусть даже и не такие, как представляет себе человек. И пусть эта система учится сама. Но чтобы система могла сделать такое, нужен какой-то критерий, иначе непонятно, как и на чем учиться, чтоб было понятно, для какой задачи нужен этот СВП.

Этой задачей является сформировать такое СВП, которое могло бы по контексту определить вероятность появления слова. Контекст – это слова, расположенные слева и справа от искомого. Если задать окружение слова, то хорошее СВП укажет нам, что должно находится внутри этого контекста. Точнее, на основе СВП для каждого слова из словаря должна определиться вероятность его появления в этом контексте. Условно это изображено на Рис. 9.

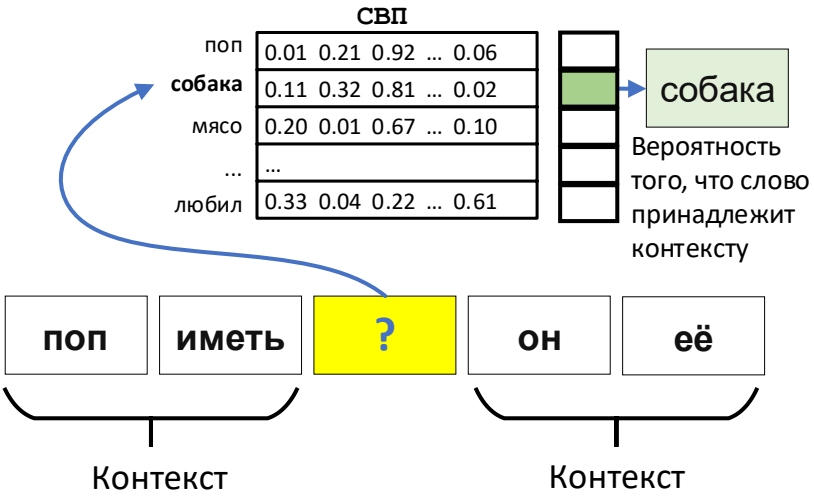


Рис. 9. Хорошее СВП нужно для предсказания

Наиболее популярной разновидностью методов преобразования слов в числа (построения СВП или *Word Embedding*) является модель *Word2Vec*, в основе которой лежит нейронная сеть. Напомним, что языковая модель – это способ кодирования и хранения текстов для определения наиболее вероятных слов. Мы рассмотрим далее именно модель *Word2Vec*, т.к. она, во-первых, проста и наглядна, а во-вторых, многие другие модели используют во многом аналогичные подходы.

Наша задача заключается в том, чтобы понять, что и как надо подавать на вход системы, чтобы она научилась (сама, с помощью нейросети) определять вероятности появления слов в контексте.

1. **Исходный текст**: *“У попа была собака, он ее любил”*.
2. Строим словарь: {*“у”*, *“попа”*, *“была”*, *“собака”*, *“,”*, *“он”*, *“ее”*, *“любил”*}.

2. Нас интересует слово “*собака”*.

3. Контекст – слова слева и справа от интересующего нас слова. Точнее, контекстов получается много: *(“у попа была” – **собака** – “он ее любил”) = (“у попа была” – **собака**), (“попа была” – **собака**), (“была” – **собака**), (**собака** – “он ее ”), (**собака** – “он ее любил”)…*

Как происходит разбиение на слова и формирование словаря, мы уже говорили. Разберемся со входом нашей системы. Будем кодировать слова очень простым способом, указывая позицию слова в словаре. Такое кодирование называется представлением слов в формате *one-hot* (переводить это не будем):

	Слова в предложении					
Слово из словаря	“поп”	“иметь”	“собака”	“он”	“ее”	“любить”
“поп”	**1**	0	0	0	0	0
“собака”	**0**	**0**	**1**	**0**	**0**	**0**
“любить”	0	0	0	0	0	**1**

Итак, нас по-прежнему интересует слово “собака” и его контекст. Повторим, что контекст слова – это слова слева и справа от него на какую-нибудь глубину. Допустим, что эта глубина равна двум (по два слова слева и справа).

Входное слово “собака” будет представлено так: *X = [0, 0, 0, **1**, 0, 0, 0].*

Контекст (слова *“у”* и *“любить”* в него уже не входят)*: Y = [0, 1, 1, 1, 1, 1, 0].*

Алгоритм *Word2Vec* ищет все предложения с входным словом *X* и контекстом *Y* вокруг него. Причем ищет во всех загружаемых в систему текстах. *Y* является тем вектором, с которым сравниваются результаты выходного слоя.

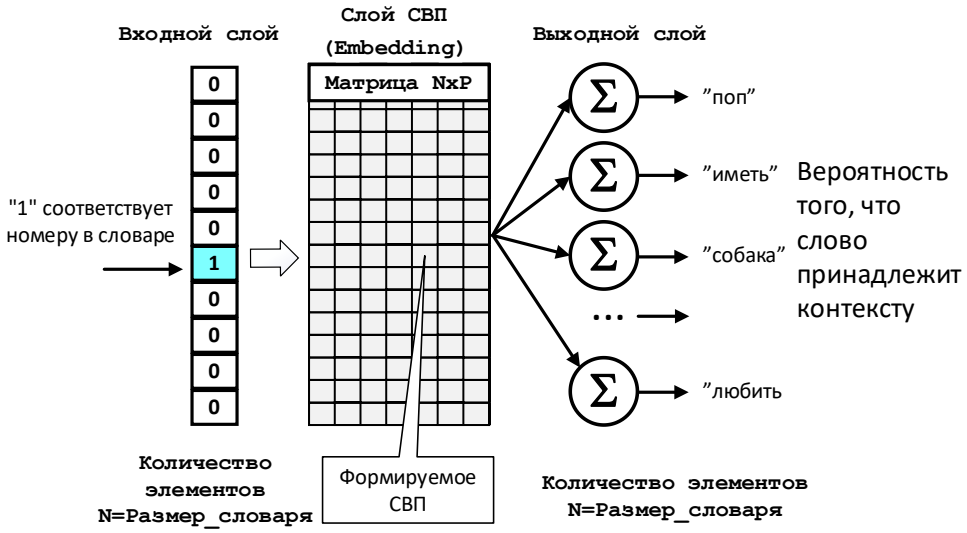


Рис. 10. Формирование СВП с помощью нейронной сети. Здесь P – размерность СВП

Интересно, что при таком способе кодирования можно менять местами *X* и *Y*. Можно использовать механизм *Skip-gram***,** где входе – слово, а на выход – контекст, можно использовать *CBOW* (Continuous bag-of-words), где на входе – контекст, а на выходе – предсказанное слово.

Итак, мы получили самый важный результат: имея слово, модель выдает вероятности слов его окружения (контекста):

“собака” => “у”, “папа”, “была”, “он”, “ее”, “любил”.

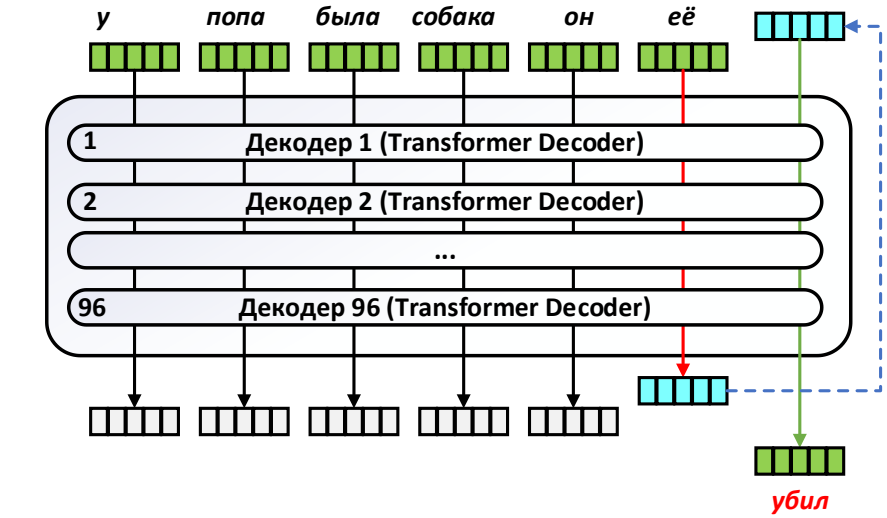
Все остальное проще.

1. Для учета позиции слова в тексте можно при кодировании добавить к этому слову еще один атрибут – его номер.
2. Ясно, как можно последовательное разворачивать цепочки слов (определяя самое вероятное продолжение).
3. Можно реализовать “механизм внимания” (чтоб было понятно, что *“он”* относится к слову поп*, а “ее”* – к собаке).
4. Можно придумать различные стратегии разворачивания последовательностей слов.
5. И так далее.

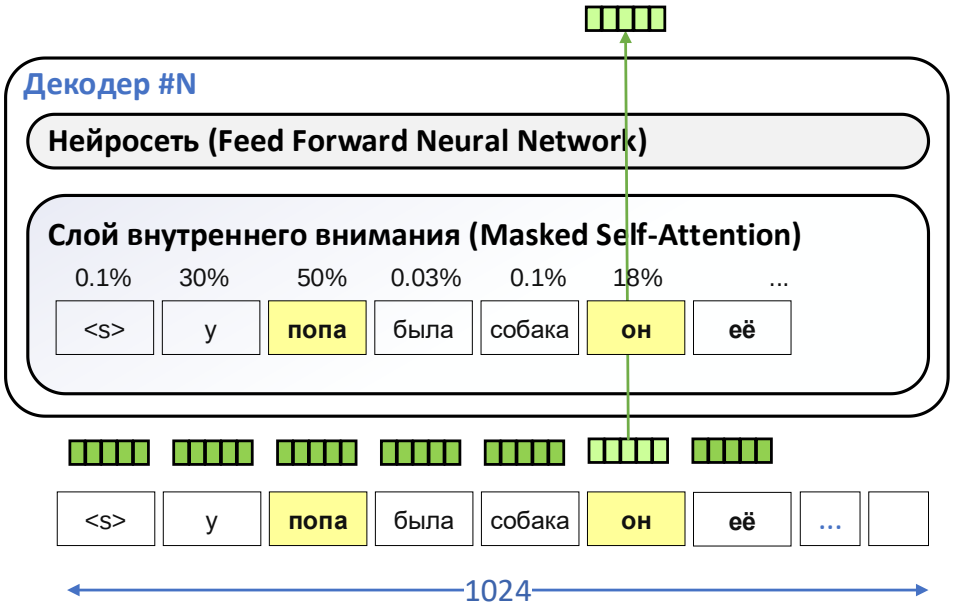
Подытожим. Принцип работы всех этих чатов примерно одинаков и все зависит от того, как сформированы СВП. Сам же процесс генерации продолжения текста выглядит так:

1. Слова входного текста кодируются в последовательности чисел – список токенов.
2. Список токенов проходит через линейный слой и превращается в список СВП (*embeddings*, очень похоже на *Word2Vec*).
3. К каждому СВП добавляется указание его позиции в предложении.
4. Список СВП проходит через несколько одинаковых блоков – декодеров (*Transformer Decoder Block*).
5. После того как список СВП пройдёт через последний блок, СВП, соответствующий последнему слову в предложении, отправляется обратно на вход сети, и мы получаем распределение вероятностей следующего слова-токена.
6. Из этого распределения выбираем следующий (например, наиболее вероятный) токен.
7. Добавляем этот токен к входному тексту и повторяем шаги 1-6.

Это условно изображено на Рис. 11. Обратите внимание на количество блоков декодеров. В модели *GPT-3* их 96. Это очень много. Каждый блок – это большая нейросеть, с огромным количеством настраиваемых (обучаемых) параметров.



а)



б)

Рис. 11. а) общая схема формирования следующего слова в предложении: прохождение через 96 однотипных блоков – декодеров, каждый декодер – это нейросеть, б) устройство декодера

Чем больше декодоров, тем более “тонкие”, “косвенные” зависимости между словами может найти и учесть система. Чем шире входной вектор (1024 – это тоже очень много), тем больший контекст можно охватить. Только платой за это является большое количество вычислительных ресурсов и времени, необходимых для обучения всего этого хозяйства.

Множество улучшений. Вокруг базовой модели выстраивается целая инфраструктура, множество механизмов, улучшающих ее работу. К таким полезным механизмам относятся, например, распознавание устойчивых сочетаний слов (Phrase Learning). Например, “New+York”, “Нижний+новгород”... и т.д. А еще полезно

избавляться от слишком часто повторяющихся слов (Subsampling). Предлоги, союзы в контексте не всегда информативны (большая частота встречаемости слова в корпусе увеличивает вероятность того, что оно не имеет информативной ценности).

Стили генерации и фильтрация. Отдельно стоят механизмы, отвечающие за стратегии продолжения текста, включая определения того, что ответ пошел не туда.

Итак, при генерации текста система последовательно определяет вероятности следующих слов. Самый простой вариант выбора – это взять следующим наиболее вероятное слово. Этот стиль называется *сэмплированием* (Sample). Но опыты показывают, что при этом увеличиваются повторы и как ни странно, увеличивается вероятность несвязности текста. Есть сэмплирование с ограничением маловероятных исходов (Filtering). Здесь выбор следующего происходит так же, как и в предыдущем случае, но с помощью некоторых параметров мы заранее убираем все маловероятные слова. А еще можно попробовать оценить заранее всю получающуюся цепочку. Это – т.н. *лучевой поиск* (Beamsearch), при котором выбирается не следующее самое вероятное слово, а цепочка слов, у которой общая вероятность наибольшая.

Оценка потенциальной цепочки лежит и в основе фильтрации текстов. Если в цепочке обнаруживается некоторое “запрещенное” слово (которое может быть и самым вероятным), то цепочка отбрасывается.

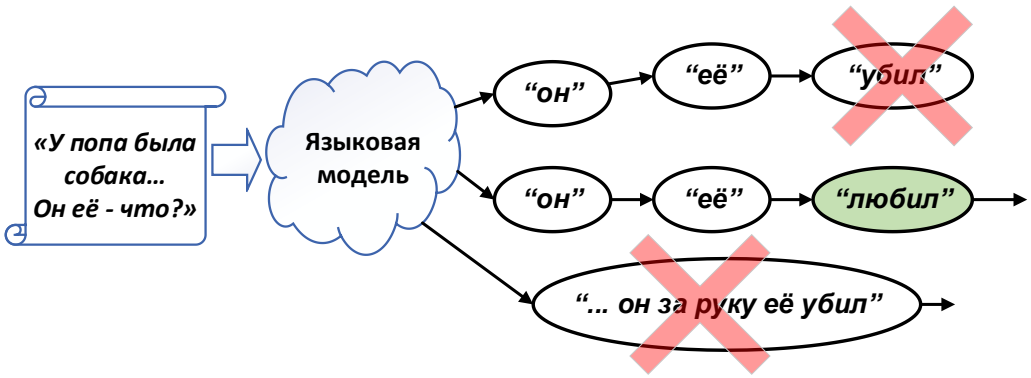


Рис. 12. Процесс фильтрации. Оценка цепочек. Если в них появляются нехорошие слова, то модель пробует следующий вариант

2. Изобразительное творчество

Одним из наиболее ярких проявлений генеративности является создание изображений по их текстовому описанию. Мы, разумеется, не будем обсуждать правовые,

этические и искусствоведческие аспекты этого явления. Поговорим просто о том, как это устроено на примере системы DALL-E 2 [DALL-E mini, 2023].

Итак, на входе – текстовое описание, на выходе – изображение. Основная идея очевидна: следует векторизовать и текст (много текстов), и изображения (много изображений). Обучающая выборка для таких систем – это множество самых разных изображений (DALL-E 2 обучалось на 650 миллионах изображений) и их текстовые описания. Иными словами, каждый обучающий пример – это файл с изображением (картина, например, или фото) и текст ("*картина «Купание красного коня». Автор – Петров-Водкин, холст, масло, мальчик на коне, водоем*"). Далее надо сопоставить или связать между собой СВП текста и СВП изображения. Как строится СВП текста, мы разбирали выше. СВП изображения формируется таким же "бессмысленным" образом. Только если текст представлялся собой множеством кусков (слогов, токенов) слов, то для кодирования изображений составными частями будет фрагменты изображений (скажем, части размером 32x32 точек).

Смысл обучения – в сопоставлении изображений и их описания. Условно это показано на Рис. 13.

	СВП изображений							
СВП описания						…		
Фото собаки	0.03	0.85	0.2	0.01	0.1		0.06	0.01
Фото кота	0.95	0.2	0.1	0.3	0.02		0.02	0.2
Бегущий человек	0.01	0.01	0.9	0.002	0.1		0.5	0.3
Натюрморт с рыбой	0.001	0.002	0.0001	0.8	0.6		0.7	0.01
…								
Картина "Девочка на шаре"	0.2	0.24	0.3	0.5	0.32		0.3	0.9

Рис. 13. Сходство текста и изображения

Важнейшим компонентом DALL-E 2 является нейросеть CLIP (12 миллиардов параметров). Интересно, что основой текстового представления в ней является знакомая модель *Word2Vec*.

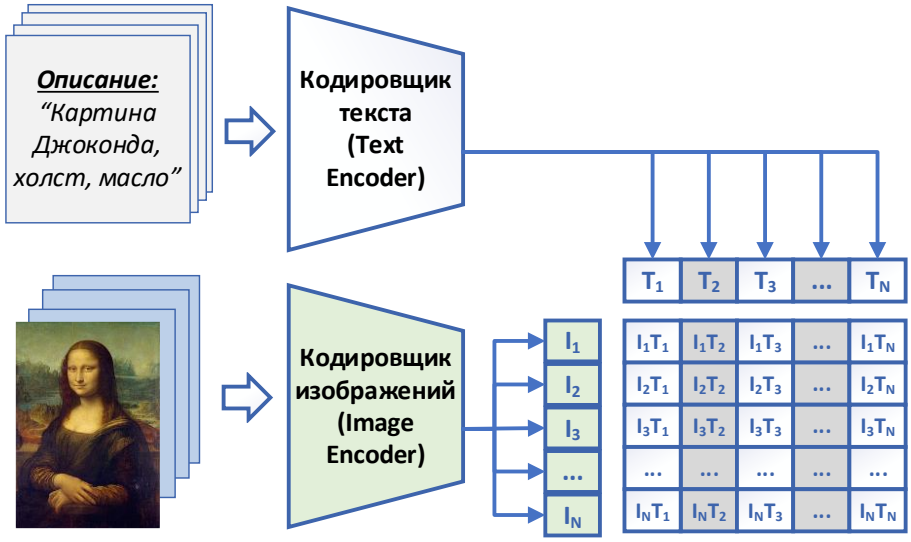


Рис. 14. Сопоставление текстового описания с изображениями

Главная задача этой системы не в том, чтобы точно идентифицировать и запоминать пару текст-изображение, а учиться находить пару среди других, которую с наибольшей вероятностью можно было считать правильным (опять поиск наиболее вероятного фрагмента или слова).

Далее мы берем текстовый запрос пользователя и, сопоставляя его СВП описаний обучающей выборки, извлекаем уже СВП изображений. И дальше “куски” изображений превращаем в итоговую картинку.

Конечно, все не так просто. Надо не только сформировать СВП текста и изображений (этим занимается CLIP), но еще превратить результаты сопоставления в изображение (этим занимается система GLIDE), а потом сделать эту картинку хорошо выглядящей. Но в целом последовательность генерации изображения по описанию такова:

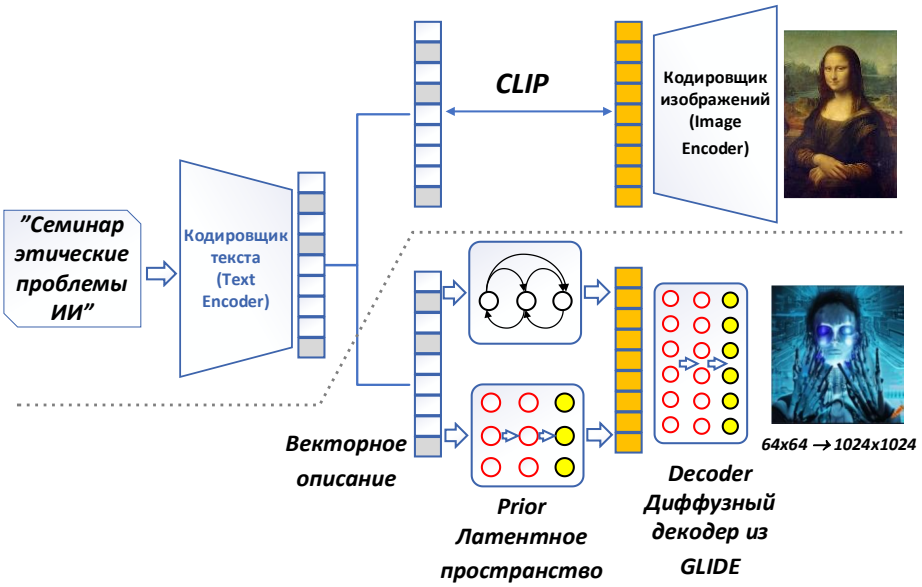
1. Word2Vec превращает слова описания в СВП, которыми оперирует нейросеть CLIP.

2. Векторы отправляются в латентное пространство (Prior). В Prior СВП используются, для подбора векторов будущего изображения (CLIP).

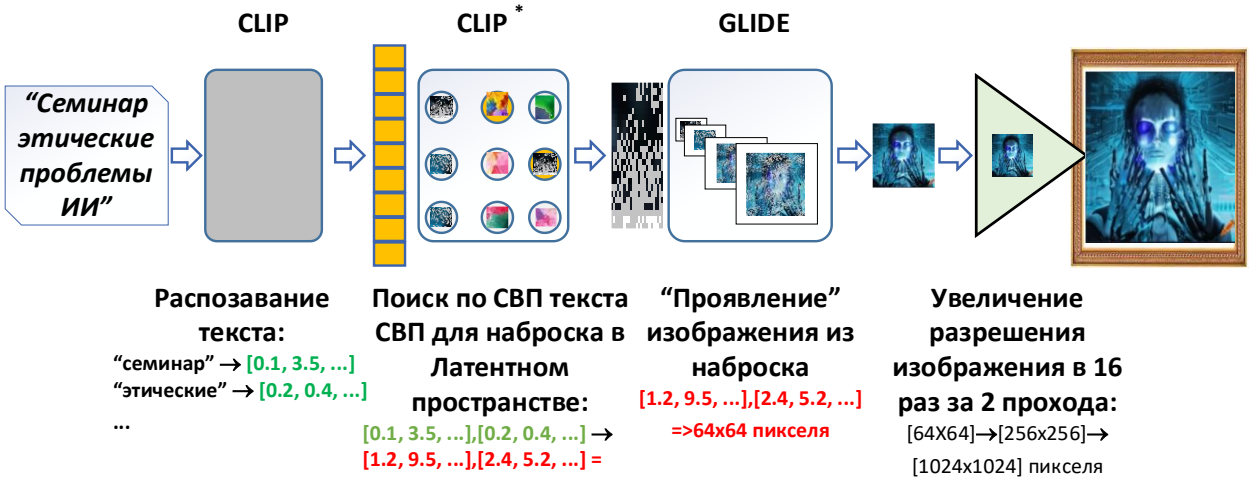
3. Диффузная модель (GLIDE) берёт текстовые и визуальные СВП и выдаёт картинку в разрешении 64x64.

4. Финальное изображение масштабируется до 1024 x 1024 пикселей с помощью двух дополнительных диффузных моделей.

Эта схема изображена на Рис. 15.



а)



б)

Рис. 15. Схема работы DALL-E. а) укрупненная, б) более подробная

Теперь становится понятным, почему такие системы иногда выдают странные или страшные изображения, как фильтруются результаты (на текстовом, описательном уровне, разумеется).

Для иллюстрации на Рис. 16 приведены результаты работы клона DALL-E 2 – системы DALL-E mini [DALL-E mini, 2023].

<table>
<tr>
<td> 
“Рыба на велосипеде”</td>
<td> 
“Юрист на велосипеде”</td>
</tr>
<tr>
<td> 
“Женщина на велосипеде”</td>
<td> 
“Рыба, юрист и женщина на велосипеде”</td>
</tr>
</table>

Рис. 16. Примеры генерации изображений программой DALL-E mini [DALL-E mini, 2023]

Дело, конечно, не в качестве, а в том, что мы понимаем, как это происходит и почему эти изображения выглядят именно так.

Справедливости ради отметим, что если задать те же запросы программе ”Кандинский” от Сбербанка, то мы получим более интересные результаты [AIRI, 2023b].

3. Заключение

Многие вопросы, которые любит обсуждать широкая публика, стали, надеемся, более понятными, а то и отпадают вовсе.

1. Ясно стоит за названием таких систем. Chat – это просто чат, болталка, то, что умеет складно выстраивать слова. GPT – это языковая модель: Generative Pre-trained Transformer, т.е. Генеративный Предобученный Трансформер. Генеративный, т.к. порождает тексты. Предварительно обученный – тоже понятно. Обучение – это основная процедура настройки весов. Трансформер – это просто общее название некоторой архитектуры нейронных сетей с прямыми связями. Их главная особенность в том, что они могут быть распараллелены, т.е. обрабатывать фрагменты независимо. Это крайне важно, т.к. текстов много и надо хоть как-то ускорить процесс обучения. Плата за такой

параллелизм тоже очевидна: эта модель может “забыть”, что было в начале ответа, она не способна выстраивать длительную логическую последовательность и т.п.

2. Ясно, что фразы типа “*что-то там является чат-ботом с искусственным интеллектом…*” тоже приобретают иное звучание (самое “интеллектуальное” во всей этой механике – это нелинейные аппроксиматоры, а о том, как такие системы играют в слова мы и разбирали в этой работе”.

3. Понятно, откуда берутся бессодержательные (хотя внешне хорошо выглядящие) ответы: что более вероятно, то и выдается. И понятно, почему система зачастую начинает убедительно “фантазировать”. Такая бессмысленность называется красивым термином “галлюцинации”.

4. Ясно, почему создание подобного рода систем – это удел крупных корпораций. С одной стороны, аналогичный чат можно создать самому (силами студенческой команды, лаборатории, ИП, ООО и т.п.). Основные механизмы известны, программные компоненты есть даже в открытом доступе. Результат получится быстро, но работать будет плохо. Настолько плохо, что не будет никакого смысла создавать такие системы “на коленке”. Причины очевидны:

А) Для обучения нужно много вычислительных ресурсов: суперкомпьютеры, кластеры и т.д. Для модели GPT-4 надо настроить 300 млрд. параметров.

Б) Нужно иметь большое количество начальных тестировщиков, причем, качественных. Для той же GPT-4 на первом этапе создавались тренировочные запросы, на которые реальные люди писали идеальный ответ (13’000 образцов). А на втором, более масштабном, этапе осуществлялась оценка качества ответов системы. Для этого использовалась “Модель поощрения”: на текстовый запрос GPT генерировал несколько ответов, которые затем реальные люди оценивали от лучших к худшим. И уже далее необходимо привлечь иметь огромное количество текущих, “обычных” тестировщиков-пользователей.

В) В этих моделях скрыто много “деталей” – механизмов, улучшающих, оптимизирующих и т.п. (красивым примером механизм “Оптимизации политики”, когда GPT сама оценивает ответ вместо человека). А этих деталей никто не раскрывает. Это те самые “ноу-хау”, без которых все будет плохо.

Здесь можно увидеть аналогии с созданием атомной бомбы в гараже (ну, хорошо, – если не бомбы, то космической ракеты): теория всем известна, но сделать это нереально: “тонкости”, “малозначительные детали” не известны, а еще ресурсов нужно очень много.

Список литературы

1. AIRI. GigaChat [Электронный ресурс]. URL: https://developers.sber.ru/portal/products/gigachat (дата обращения: 18.06.2023a).

2. AIRI. Kandinsky 2.1 от Сбера [Электронный ресурс]. URL: https://fusionbrain.ai/diffusion (дата обращения: 16.06.2023b).

3. GPT-3.5. BAI Chat [Электронный ресурс]. URL: https://chatbot.theb.ai (дата обращения: 11.06.2023).

4. Rosenblatt F. The perceptron. A perceiving and rocognizing automaton (project PARA). Report No. 85-460-1. , 1957. 33 с.

5. Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. // Psychol. Rev. 1958. Т. 65. С. 386–408.

6. Turing A. M. Computing machinery and Intellicence // Mind. 1950. Т. 54. № 236. С. 433–460.

7. YaGPT. Чат от Яндекса [Электронный ресурс]. URL: https://ya.ru/ (дата обращения: 18.06.2023).

8. АОТ. Автоматическая Обработка Текста [Электронный ресурс]. URL: http://aot.ru/ (дата обращения: 05.06.2023).

9. Вейценбаум Д. Возможности вычислительных машин и человеческий разум. От суждений к вычислениям. М.: Радио и связь, 1982.

10. Дидро Д. Философские мысли. Сочинения: в 2-х т. Т.1 Пер. с франц.; Сост., ред., вступит, статья и примеч. В.Н.Кузнецова. М.: Мысль, 1986. 592 с.

11. Поспелов Д. А. Фантазия или наука: на пути к искусственному интеллекту. М.: Наука, 1982.

12. The Loebner Prize [Электронный ресурс]. URL: https://www.ocf.berkeley.edu/~arihuang/academic/research/loebner.html (дата обращения: 05.06.2023).

13. DALL-E mini [Электронный ресурс]. URL: https://huggingface.co/spaces/dalle-mini/dalle-mini (дата обращения: 11.06.2023).