

Stepik. Neural networks and NLP. 6. Transfer learning - Part 1

6.1 Контекстуализированные представления и перенос знаний

Привет! Поздравляю всех кто досмотрел до этого момента! Эта лекция, этот модуль — последние в нашем курсе. Итак, давайте приступим к нашей лекции. Как вы помните из предыдущего модуля, 2018 год, в сфере [NLP](#), стал достаточно переломным моментом. Во-первых, концептуальное понимание того, как лучше всего представлять слова и предложения (чтобы учитывать их семантику, какой-то лингвистический смысл) достаточно сильно поменялось. Во-вторых, именно в этом году были предложены мощные и (что самое главное) выложенные с уже предобученными весами, модели, которые можно было скачать и использовать в своих прикладных задачах. И эти модели работали очень и очень хорошо. Этот момент был назван "моментом и ImageNet в NLP"^[1] — со ссылкой на то, как несколько лет назад подобные разработки в [transfer learning](#) ускорили развитие компьютерного зрения. И в этой лекции мы с вами попробуем отследить основные шаги развития transfer learning в области NLP с 2018 года по текущий момент. Сразу скажу, что эта лекция носит достаточно обзорный характер и мы не будем пытаться разобрать все малейшей тонкости всех модификаций всех недавно вышедших архитектур, а, скорее, рассмотрим основные концепты и основные идеи. Для начала, напомню, что основная идея transfer learning состоит в следующем. Мы предобучаем нашу модель на большом [корпусе](#), а затем [файнтюним](#) (fine-tune) предобученные веса сети. Берём сравнительно небольшой размеченный корпус и дообучаем сеть на решение интересующей нас задачи (например, на решение [задачи классификации](#) текстов или на [машинный перевод](#) или для решения какой-то другой задачи). В NLP часто применяется следующая схема: на этапе предобучения модель пытается предсказывать пропущенные слова в тексте (то есть, решается знакомая вам [задача языкового моделирования](#)), а на этапе дообучения полученные веса модели файнтюнятся с использованием размеченных данных и модель учится решать некоторую задачу. Выбор задачи языкового моделирования достаточно логичен и понятен, для решения этой задачи нам не нужны размеченные данные. Достаточно взять большой корпус текстов и поочерёдно заменять некоторой маской слова и пытаться их предсказывать. Кроме того, в процессе такого обучения модель приобретёт знания о том, как устроен язык. Например, она научиться понимать, какие слова встречаются с другими словами рядом или — какие устойчивые словосочетания существуют в языке. Вообще, эта идея — использовать semi-supervised learning для текстов появилась достаточно давно — гораздо раньше 2018 года. Например, были попытки считать некоторые статистики на уровне слов и фраз с использованием

неразмеченных данных, а затем использовать полученные данные как фичи для решения задач [обучения с учителем](#) (например, для классификации текстов). Тем не менее, до недавнего времени не было такого бума и такой популярности transfer learning для работы с текстами. Давайте же рассмотрим основные шаги, основные вехи, появление которых ознаменовало расцвет transfer learning в области обработки текстов.

[1] [NLP's ImageNet moment has arrived](#), 12 July 2018

Существует несколько способов работы с предобученной моделью:

1. Не менять веса предобученной сети (**feature extraction**). Например, мы можем добавить "наверх" предобученной модели линейный слой и обучить только его, не меняя веса предобученной сети (в таких случаях говорят, что веса предобученной сети "заморожены").
2. Менять веса предобученной сети (**fine-tuning**). В таком случае веса предобученной сети используются в качестве инициализации для downstream модели. Менять веса предобученной сети можно по-разному:
 1. Заморозить все слои предобученной сети и обучить только верхние "надстроенные" слои на решение конкретной задачи.
 2. Заморозить все слои и размораживать их постепенно.

Основная интуиция заключается в том, что одновременное обучение всех слоев сети сразу на данных другого домена может привести к нестабильности. Вместо этого мы обучаем слои индивидуально, чтобы дать им время адаптироваться к новой задаче и/или данным.

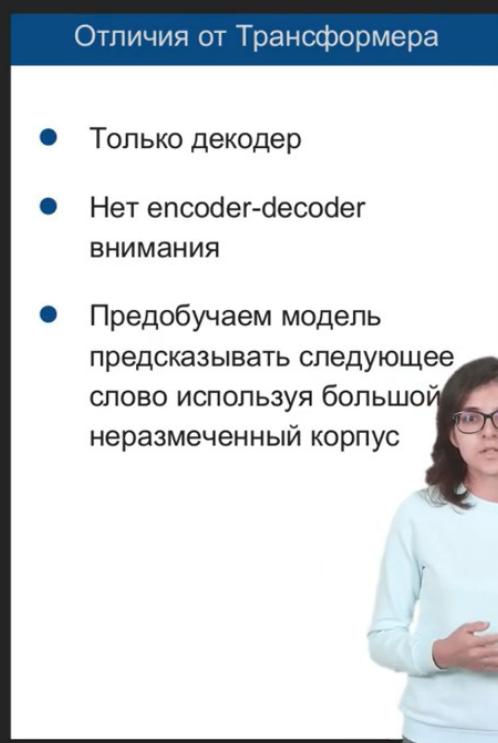
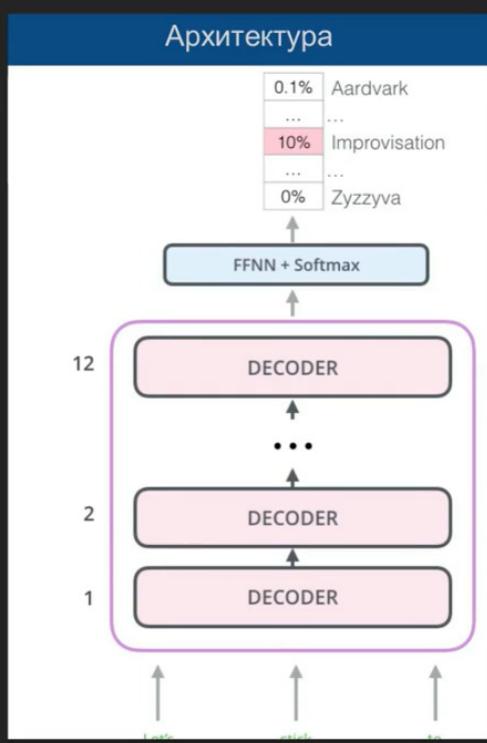
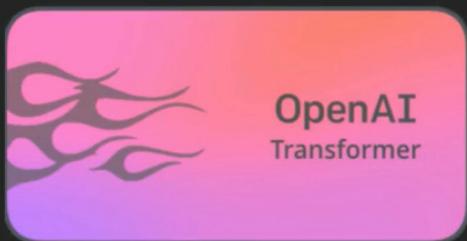
3. Не замораживать слои предобученной сети, но использовать различные (небольшие) learning rates для обучения разных слоев. Мы хотим использовать более маленькие значения learning rate, чтобы избежать перезаписи полезной информации. Более низкий learning rate особенно важен при обучении нижних слоев сети (поскольку они собирают более общую информацию), на ранних этапах обучения (поскольку модели все еще необходимо адаптироваться к целевому распределению) и на поздних этапах обучения (когда модель близка к тому, чтобы сойтись). Чтобы поддерживать более низкие скорости обучения на ранних этапах обучения, можно использовать треугольный график изменения learning rate (triangular learning rate schedule), который также называют "прогревом" скорости обучения (learning rate warm-up).
4. Не замораживать слои, но использовать регуляризацию для того, чтобы веса оставались близкими к весам предобученной сети.

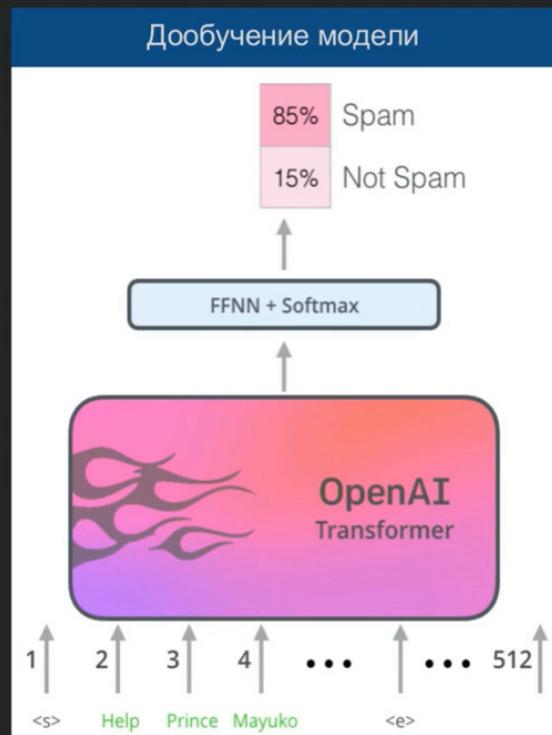
Начнём двигаться в хронологическом порядке. Итак, в июне 2018 года OpenAI анонсировал выход так называемого "[OpenAI трансформера](#)". Авторы этой модели обнаружили, что для реализации идеи [transfer learning](#) совершенно не нужна вся архитектура [трансформера](#), а

достаточно ограничиться только [декодером](#). Основная идея этой модели достаточно стандартна. Мы хотим предобучить языковую модель на большом неразмеченном [корпусе](#), так, чтобы дальше её можно было адаптировать для решения конкретных других задач. Ну, например, для решения [задачи классификации](#), суммаризации, чего угодно ещё. Если говорить про архитектуру — то, по сути, OpenAI трансформер — это декодер [ванильного](#) трансформера. В нём [стякается](#) 12 слоёв — 12 [self-attention](#) модулей. Полученную архитектуру учили решать [задачу языкового моделирования](#): предсказывать следующее слово при известном контексте. При этом, фича декодера с маскировкой будущих токенов (то есть — тех токенов, которые расположены правее от текущего токена) оказалась довольно полезной. Отлично, с предобучением OpenAI трансформеров всё очень просто — берём декодер обычного трансформера и учим его предсказывать следующее слово при известном контексте. Что же происходит с дообучением? После того, как наша модель научилась более-менее понимать язык, начнём использовать её для решения других задач. Например, будем классифицировать тексты (рассмотрим самый простой пример). Давайте представим, что мы хотим научиться разделять наши письма на 2 папки: "спам" и "не спам". Пристроим к нашему трансформеру всего один [линейный слой](#) и добавим [софтмакс](#). А теперь будем дообучать нашу модель с использованием небольшой размеченной коллекции писем. И мы обнаружим, что уже после нескольких эпох качество классификации окажется достаточно хорошим. Этот результат оказался настолько крутым^[1,2,3,4], что его попробовали адаптировать и для решения многих других задач. Так, трансформер отлично справляется и с более сложными задачами. Например, с задачей определения близости двух предложений по смыслу, или с задачей ответа на вопрос с несколькими вариантами ответа. И, при этом, для дообучения можно использовать сравнительно небольшой размеченный корпус, а предобученную модель достаточно обучить всего один раз. Ну, или даже проще — просто скачать готовые веса из интернета и начать сразу дообучать свою модель. Звучит достаточно круто, не правда ли? Но, тем не менее, всё-таки какие-то минусы и недостатки модели остаются за кадром. В чём же проблема? Как вы помните, OpenAI трансформер учится предсказывать только следующее слово и прячет все слова, находящиеся правее текущего токена. А что — если попробовать учиться предсказывать не только следующее (а, может быть, даже и предыдущее слово), а пойти ещё дальше? Например, заменять маской произвольное слово и предсказывать это произвольное слово внутри предложения.

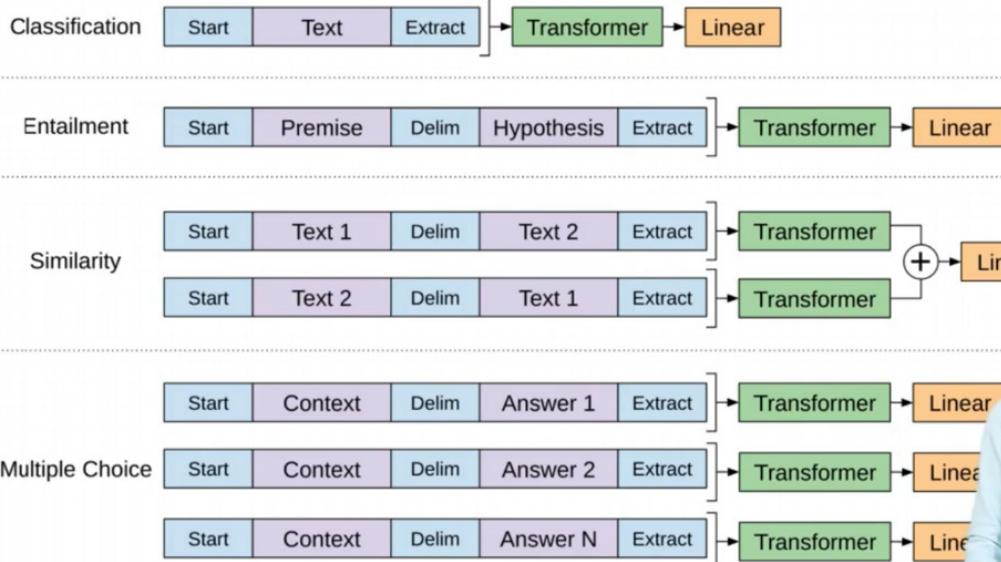
- [1] Peters M. E. et al. [Deep contextualized word representations](#) //arXiv preprint arXiv:1802.05365. – 2018.
- [2] Akbik A., Blythe D., Vollgraf R. [Contextual string embeddings for sequence labeling](#) //Proceedings of the 27th International Conference on Computational Linguistics. – 2018. – С. 1638-1649.
- [3] Baevski A. et al. [Cloze-driven pretraining of self-attention networks](#) //arXiv preprint arXiv:1903.07785. – 2019.
- [4] [The State of Transfer Learning in NLP](#), 18 August 2019

Основная цель: предобучить языковую модель на большом неразмеченном корпусе, чтобы ее можно было адаптировать для решения конкретной задачи (классификации, суммаризации, перевода, вопросно-ответный поиск и т.д.)





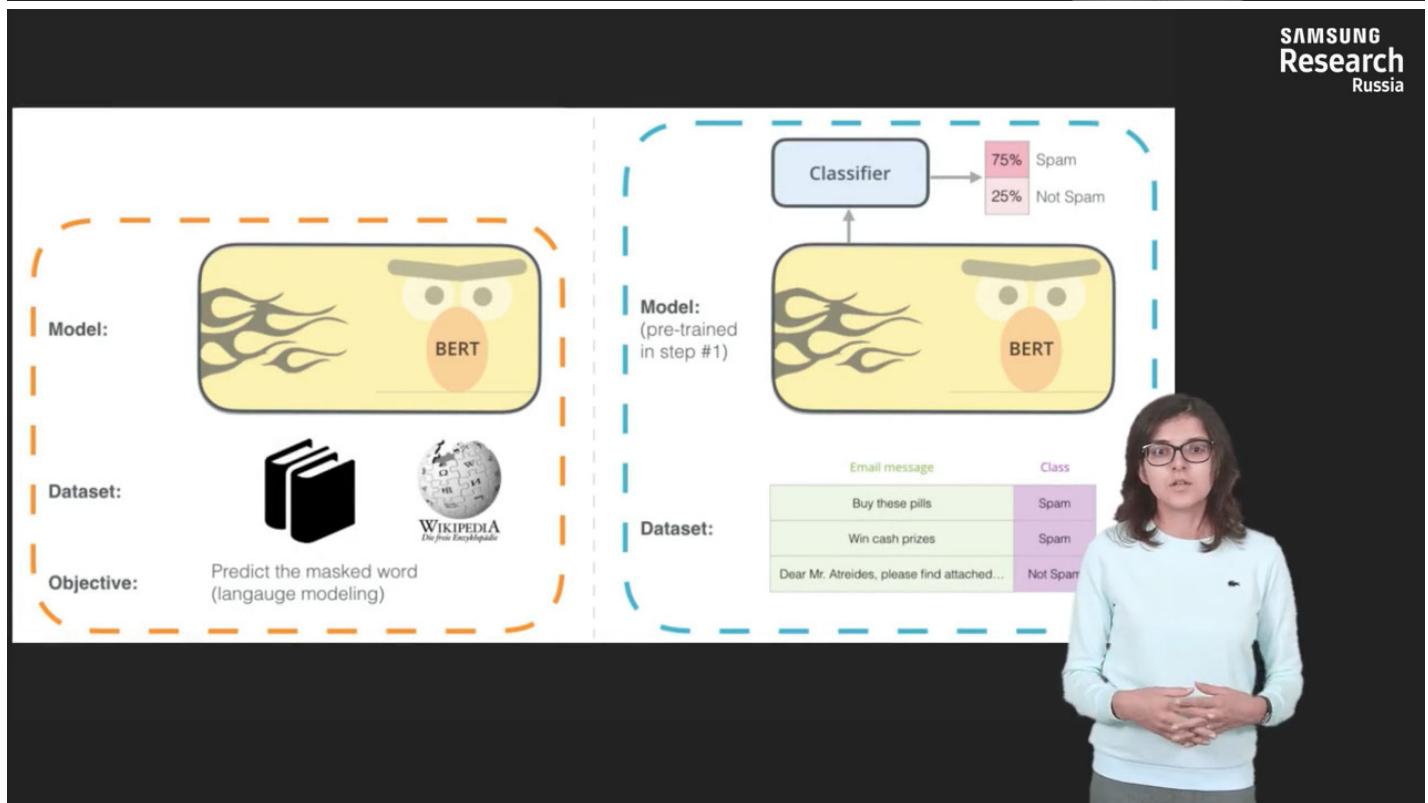
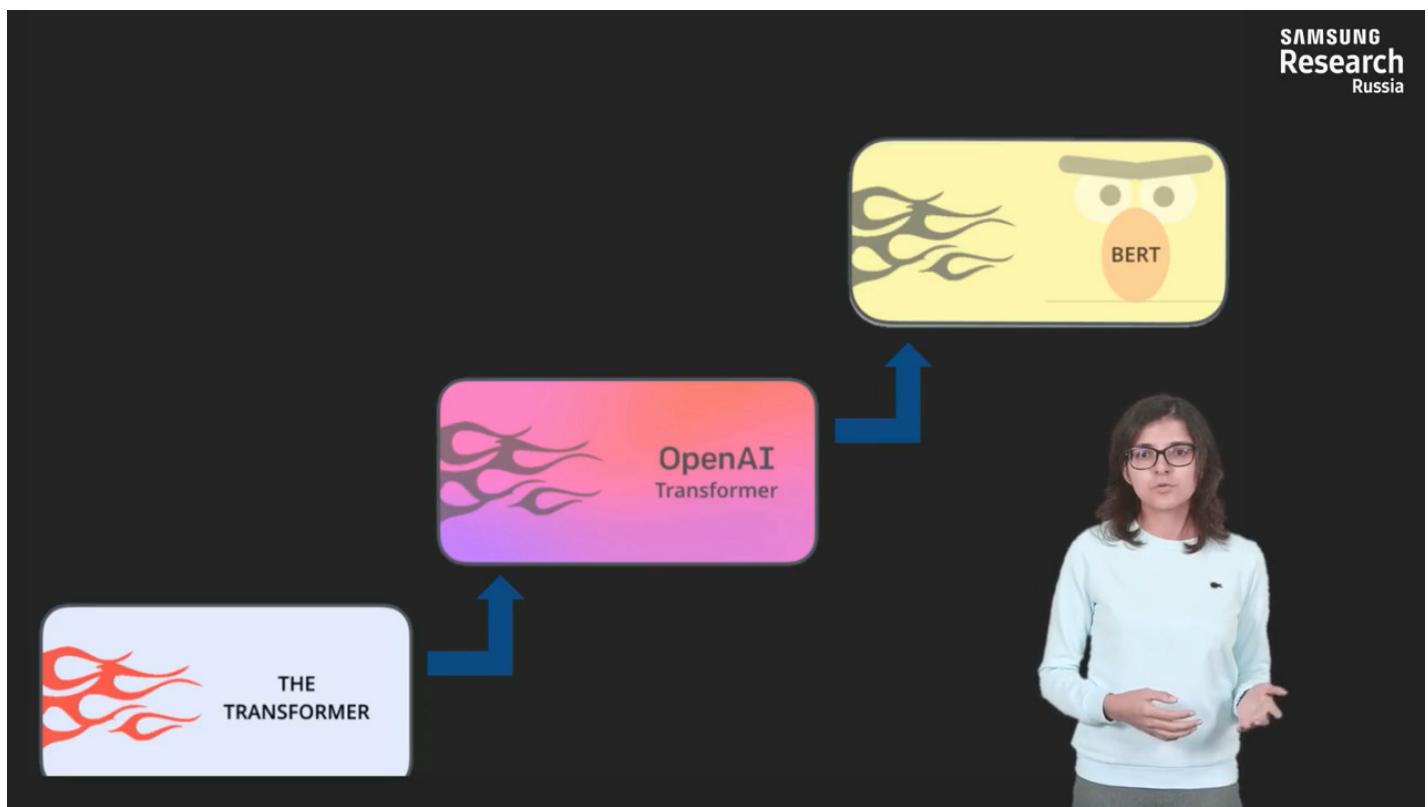
Формат входных данных для разных задач

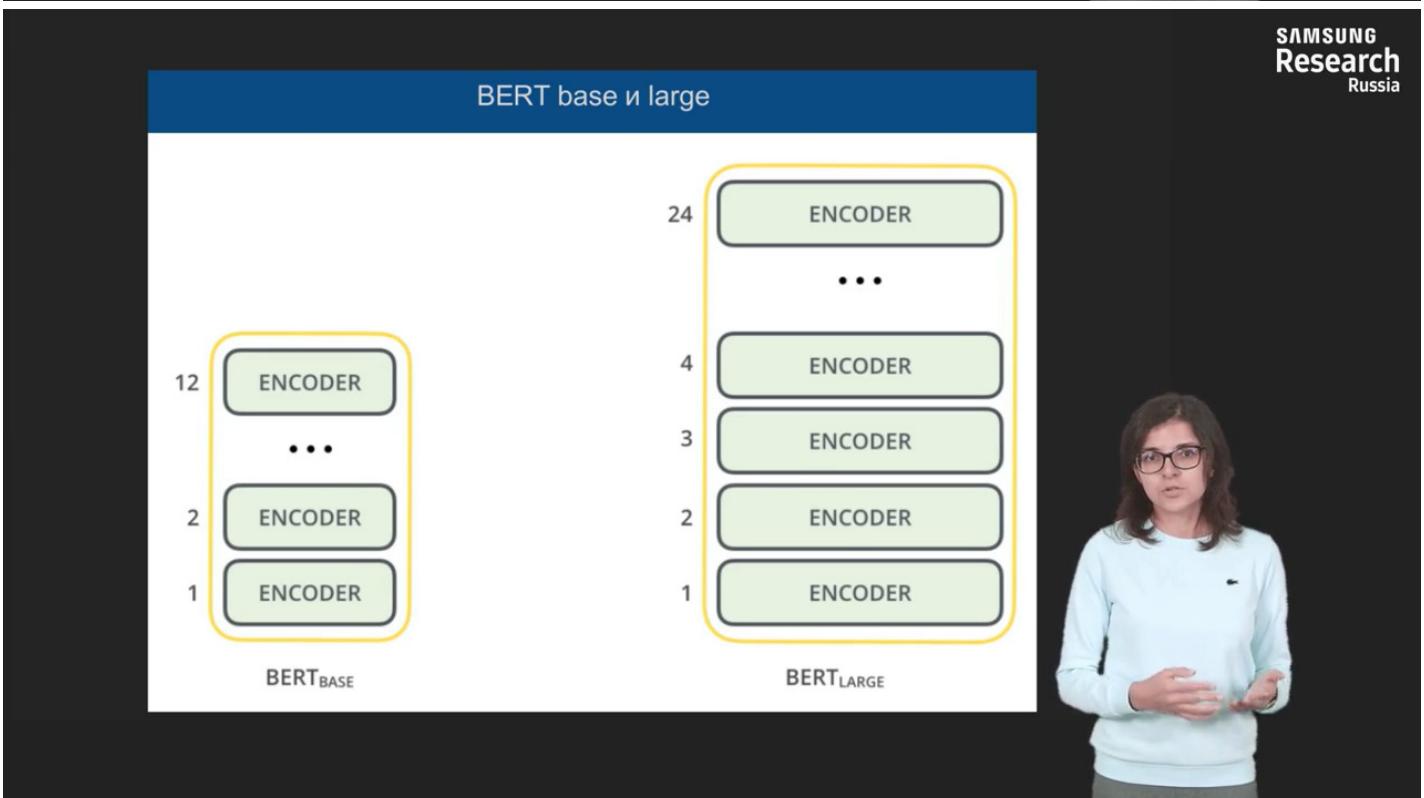
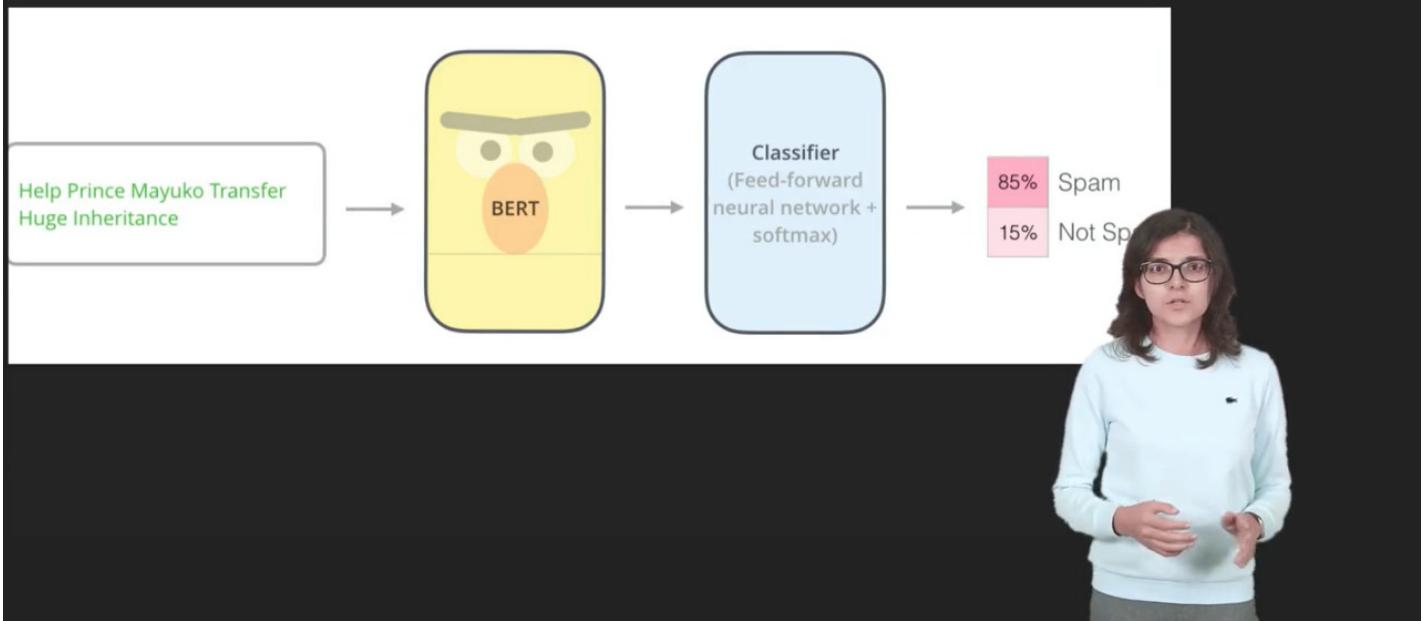


"Легко!" — сказали авторы известной в последнее время модели под названием "BERT".^[1] [BERT](#) расшифровывается как "Bi-directional Encoder Representations from Transformers", то есть — двунаправленное представление на основе трансформеров. Обратите внимание, что здесь опять будет использоваться стандартная архитектура [трансформера](#). Основная идея здесь, в принципе, та же самая: мы будем предобучать BERT на [задаче языкового моделирования](#) и

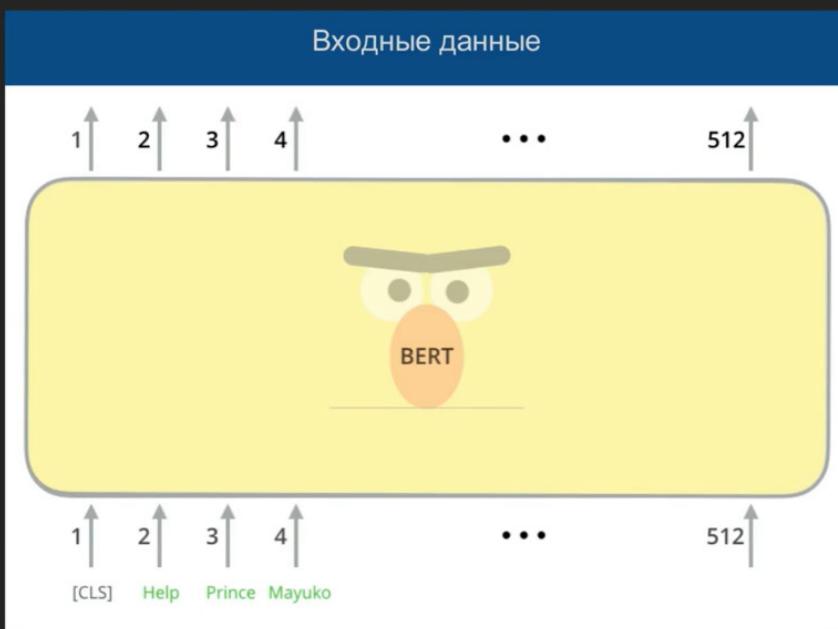
некоторой ещё чуть более усложнённой задаче, а затем будем дообучать нашу сеть на решение какой-либо задачи. Например, на решение уже привычной нам [задачи классификации](#) на два класса: на "спам" и "не спам". Но сначала давайте поговорим про архитектуру BERT, а уже потом обсудим, как происходит предобучение и дообучение. Итак, BERT — это, по факту, [настёканные](#) модули из [ванильного](#) трансформера. Существуют две популярные модификации, различающиеся только размером — это "BERT Base" и "BERT Large". С помощью BERT Large были достигнуты state of the art результаты на момент выпуска статьи про BERT. При этом, BERT Base сравним по размеру с OpenAI трансформером. Он так же состоит из 12 [self-attention](#) модулей, а BERT Large, в то же время, уже имеет в 2 раза больше self-attention модулей, и, к тому же — 16, а не 12 голов в multi-headed [attention](#). Для сравнения, в стандартном ванильном трансформере их было всего 8. Итак, с архитектурой мы разобрались. Что же принимает BERT на вход? На вход BERT принимает последовательность токенов, которые проходят через весь стэк. На каждом слое работает self-attention, затем результаты проходят через полно связанный слой и, далее, подаются в следующий блок — в следующий encoder. На текущий момент рассказа про BERT может создаться впечатление, что никаких принципиальных отличий BERT от [OpenAI трансформера](#) нет. Но, на самом деле, это неправда. Основная фишка BERT — в его предобучении. В процессе предобучения BERT тренируется решать не одну задачу языкового моделирования, а целых две задачи. Первая состоит в уже привычном нам языковом моделировании, но здесь языковое моделирование несколько видоизменено и называется маскированным языковым моделированием. BERT маскирует 15% входных токенов и пытается их предсказать. Основное отличие от OpenAI трансформера — в том, что тут мы предсказываем не следующий токен, а любой токен в любом произвольном месте предложения. Какой токен заменить маской — выбирается произвольно. Отлично! Но решение такой задачи не научит BERT осознавать взаимосвязи между предложениями или вытаскивать долгосрочные зависимости и смысл из текста. А что, если добавить к предобучению ещё одну задачу? На вход будем подавать 2 предложения (A и B) и нужно будет определить, является ли B логичным продолжением A.

[1] Devlin J. et al. [Bert: Pre-training of deep bidirectional transformers for language understanding](#) //arXiv preprint arXiv:1810.04805. – 2018.





	THE TRANSFORMER	BERT	Base BERT	Large BERT
Количество энкодеров	6		12	24
Юниты в полносвязном слое	512		768	1024
Attention Heads	8		12	16

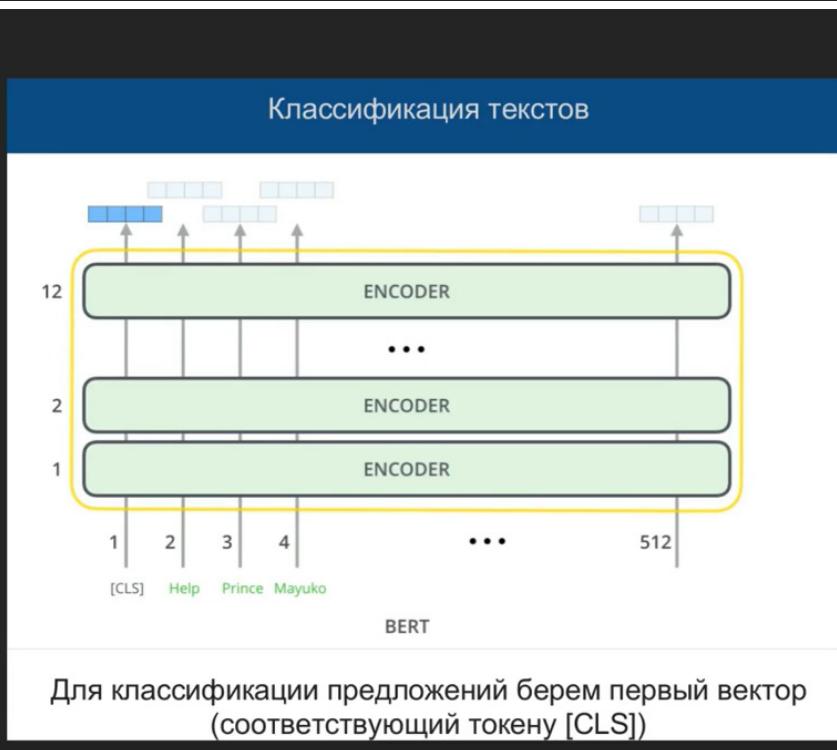
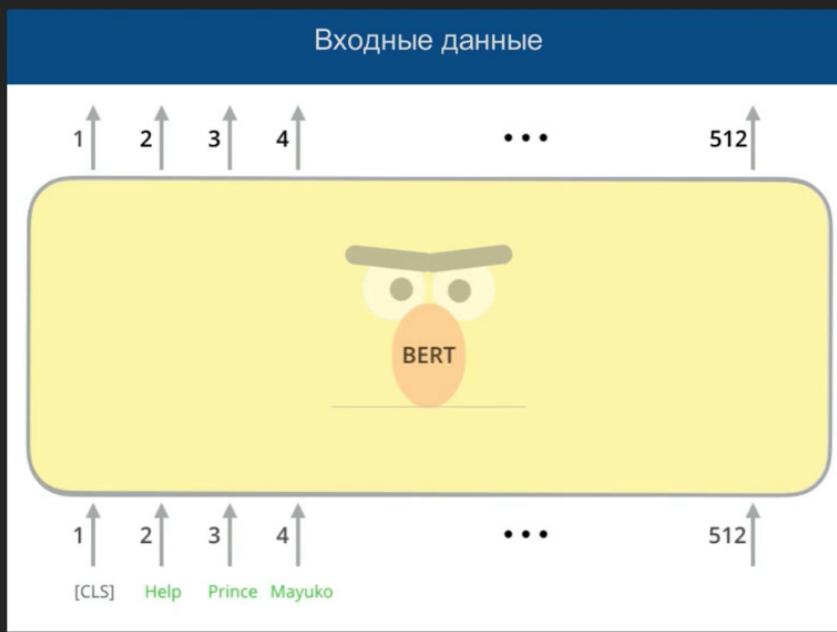


- «Маскированное» языковое моделирование
- Есть два предложения А и Б, нужно определить, является ли Б логичным продолжением А?

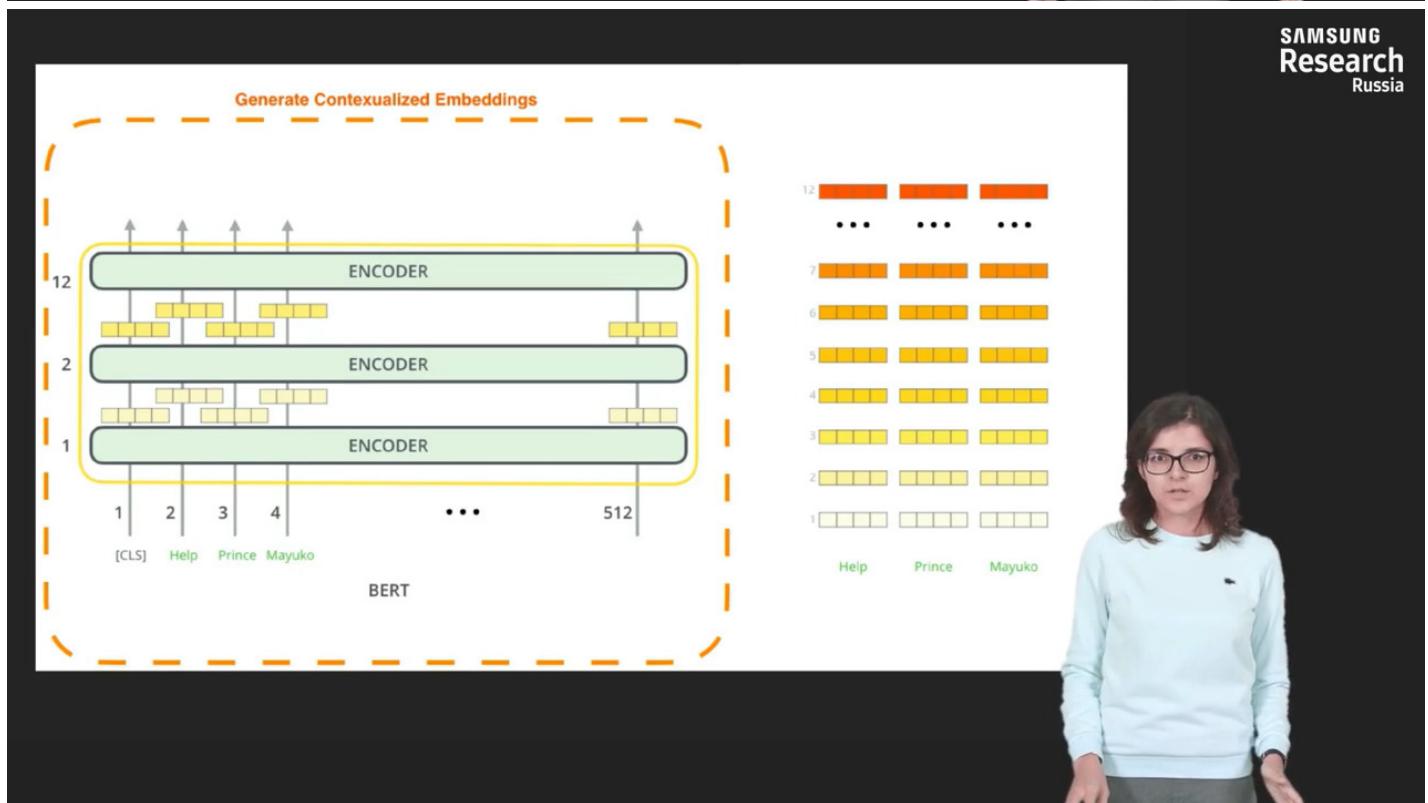
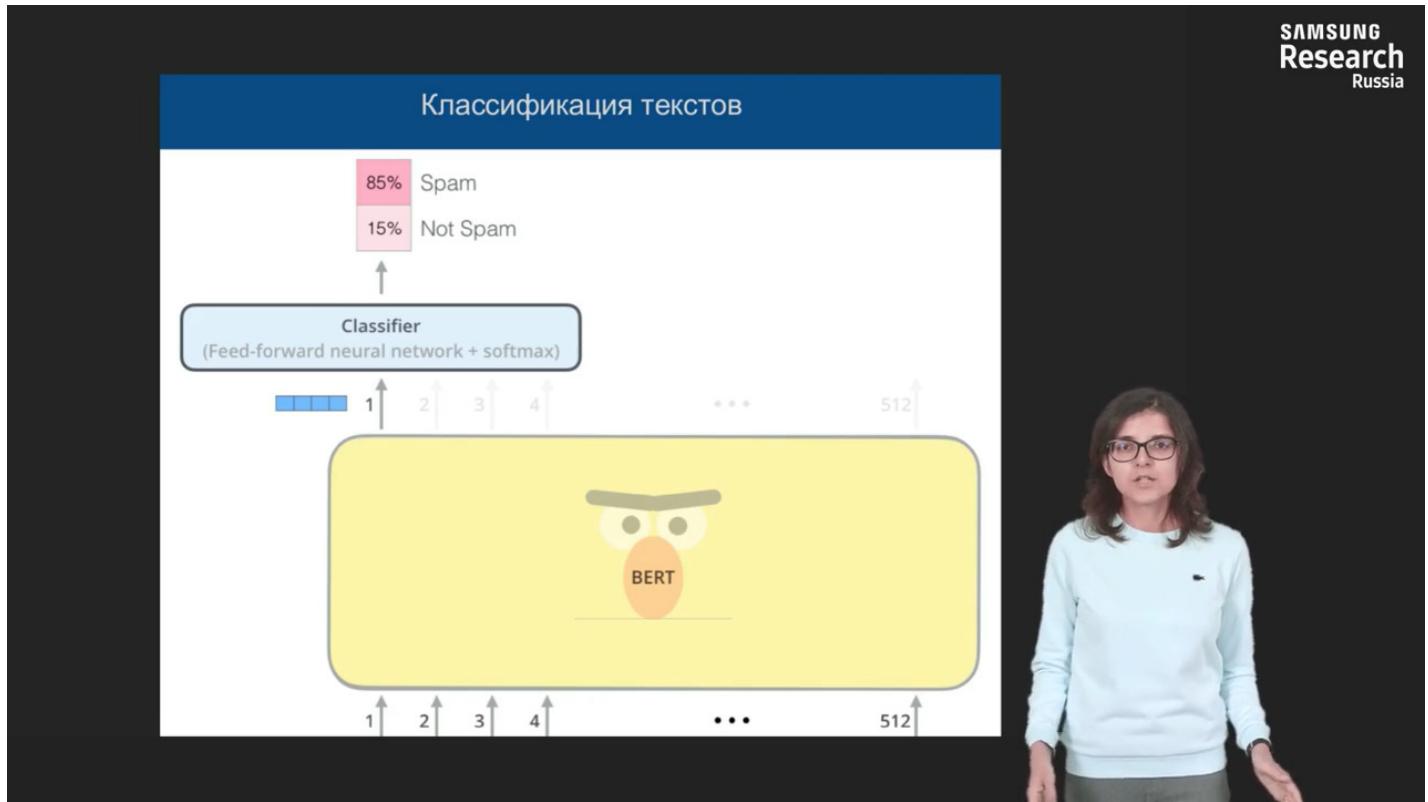


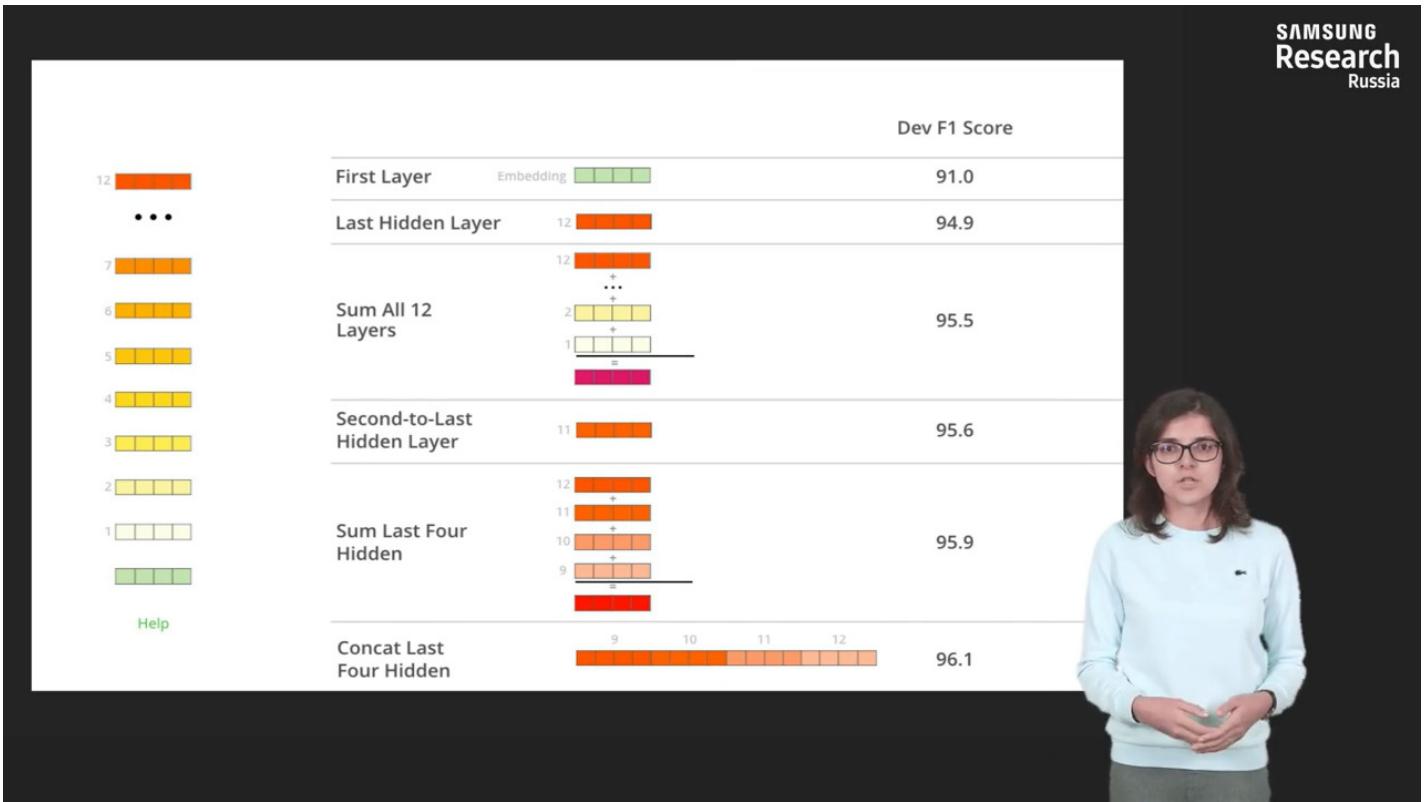
Для того, чтобы удобно работать с обеими задачами, [BERT](#) нужен особенный формат входных данных. Так — последовательность входных токенов должна начинаться с метки "CLS", а, в случае если мы подаём на вход два предложения, то токены первого предложения отделим от токенов второго меткой "SEP". В чём же смысл этой таинственной метки "CLS"? Давайте представим, что процесс предобучения окончен и мы дообучаем модель — например, хотим классифицировать письма на "спам" и "не спам" — уже знакомая нам задача. Мы подали в BERT последовательность из N токенов, а на выходе мы получим N векторов. Но, в качестве фичей для дальнейшей классификации, мы будем использовать только единственный, первый вектор — тот самый, относящийся к метке "CLS". Это может звучать немного странно и непривычно. Тем не менее, BERT учится именно в этом векторе концентрировать информацию обо всей последовательности. В статье про BERT предлагается, в качестве классификатора на этапе дообучения, использовать всего один [линейный слой с софтмаксом](#). Также, как и [OpenAI трансформер](#), BERT можно дообучить на решение многих других задач, не только на решение [задачи классификации](#). Например — можно научить BERT работать с вопросно-ответным поиском или переводить предложения с французского на английский, либо на любой другой язык. Кроме того, векторы, которые нам выдаёт предобученный BERT, можно использовать в качестве [эмбеддингов](#). Остаётся вопрос — какие именно векторы брать? Можно использовать векторы с последнего слоя (кажется довольно логичный вариант). Но, как оказалось, существуют более эффективные схемы. Например, можно суммировать векторы со всех 12, или, например, последних четырёх слоёв. Авторы статьи утверждают, что наилучшего качества, при решении задачи [извлечения именованных сущностей](#) с помощью эмбеддингов (а именно этой задачей проверялось, какие варианты комбинаций эмбеддингов с разных слоёв лучше

работают), [сконкатенировав](#) векторы с последних четырёх слоёв и использовав полученные вектора в качестве эмбеддингов, можно достичь наилучших результатов. Отлично! Но почему эти эмбеддинги так хорошо работают? Давайте вспомним проблему омонимии. В стандартном [word2vec](#) (или в [GloVe](#), или в [FastText](#)) омонимичные слова (например, слово "косой" из предложения "косил косой косой косой") будут иметь одинаковый эмбеддинг, хотя смысл и даже часть речи у этих слов "косой", три раза встретившихся в предложении, будет совершенно разной. BERT же выдаёт эмбеддинг слова в контексте — то есть, для одного и того же слова, употреблённого в разных контекстах, мы можем получить разные векторы. Получается так, потому, что проходя через [self-attention](#) блоки BERT, слово выступает в роли query, key и value несколько раз, что дополнительно обогащает итоговый вектор слова информацией о взаимосвязях его с другими словами в предложении, о его взаимосвязях с контекстом. Возникает вопрос: а был ли BERT первой моделью, которая умеет выдавать контектуализированные эмбеддинги? Оказывается, что нет.



Для классификации предложений берем первый вектор
(соответствующий токену [CLS])



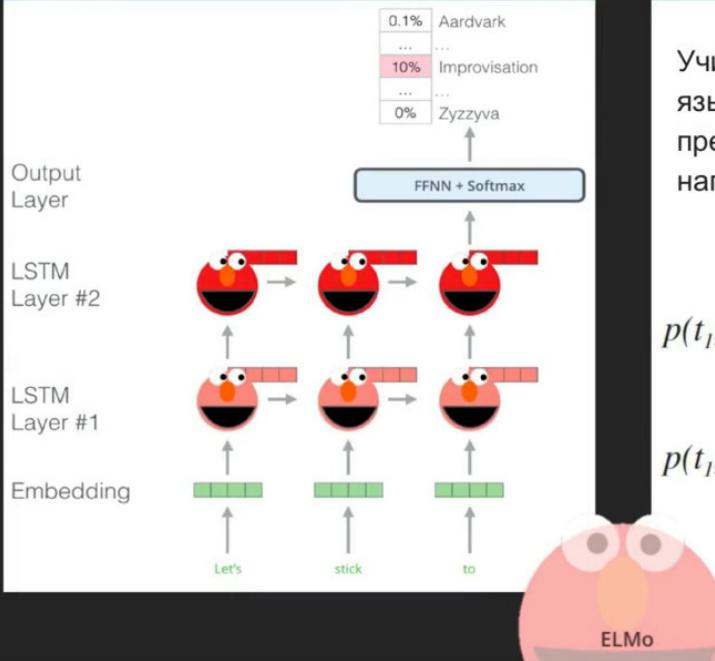


Теперь давайте немного отойдём от [self-attention](#) и вспомним, как работает [LSTM](#). Именно двунаправленный LSTM и лежит в основе популярной модели, которая выдаёт хорошие контекстуализированные [эмбеддинги](#). Эта модель называется ELMo.^[1] [ELMo](#) расшифровывается как "Embeddings from Language Models". Модель смотрит на всё предложение, прежде чем присвоить слову какое-то [векторное представление](#). ELMo, точно так же, как и ранее рассмотренные модели, учится решать [задачу языкового моделирования](#), но уже не с помощью архитектуры на основе [трансформера](#), а с помощью LSTM (а точнее, даже двух LSTM, которые смотрят в разные стороны). Давайте чуть подробнее про это. Одна LSTM смотрит вперёд и учится предсказывать следующее слово при наличии контекста, то есть, всех слов, которые расположены левее того, которое нужно предсказать. А вторая LSTM смотрит назад и предсказывает предыдущее слово, зная все слова, стоящие правее от текущего слова. Такая LSTM называется двунаправленной (или, по-английски, bi-directional). Итоговое векторное представление слова получается путём конкатенации скрытых состояний из обеих частей LSTM. Можно конкатенировать эти скрытые состояния разными способами. Например, вектора скрытых состояний можно домножить на веса и, затем, [сконкатенировать](#) (или даже суммировать) в один вектор. Выбор варианта комбинации скрытых слоёв двух частей LSTM, скорее, относится к инженерным задачам, некоторым инженерным хакам, подбору эвристик. Так что сейчас мы не будем останавливаться на этом подробно.

[1] Peters M. E. et al. Deep contextualized word representations //arXiv preprint arXiv:1802.05365. – 2018.

Архитектура

SAMSUNG
Research
Russia



Bi-directional LSTM

Учим двунаправленную LSTM
языковому моделированию:
предсказываем слова в обоих
направлениях:

$$p(t_1, t_2, \dots, t_N) = \prod_{k=1}^N p(t_k | t_1, t_2, \dots, t_{k-1})$$

$$p(t_1, t_2, \dots, t_N) = \prod_{k=1}^N p(t_k | t_{k+1}, t_{k+2}, \dots, t_N)$$

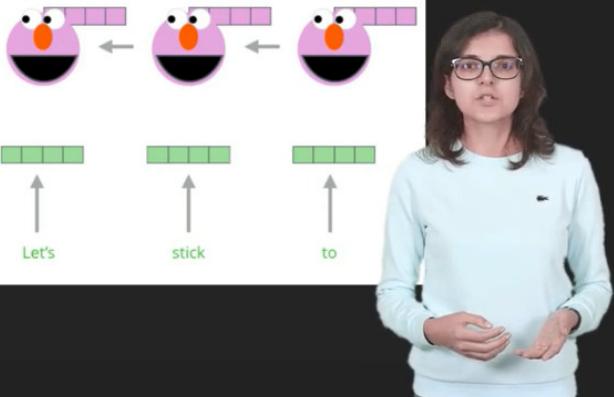
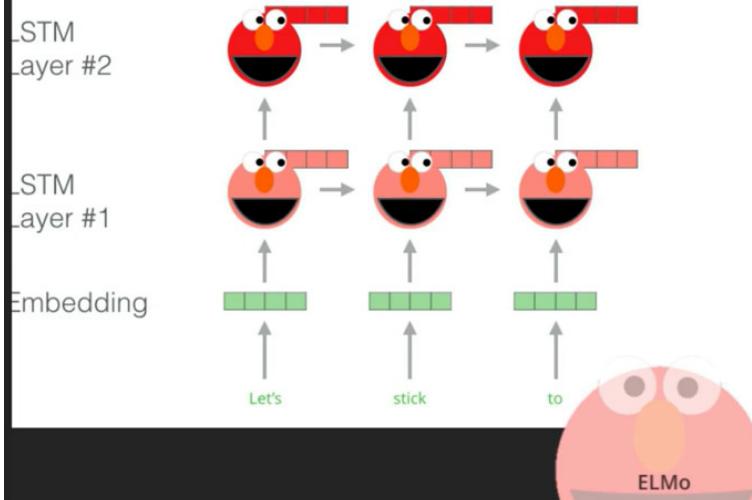


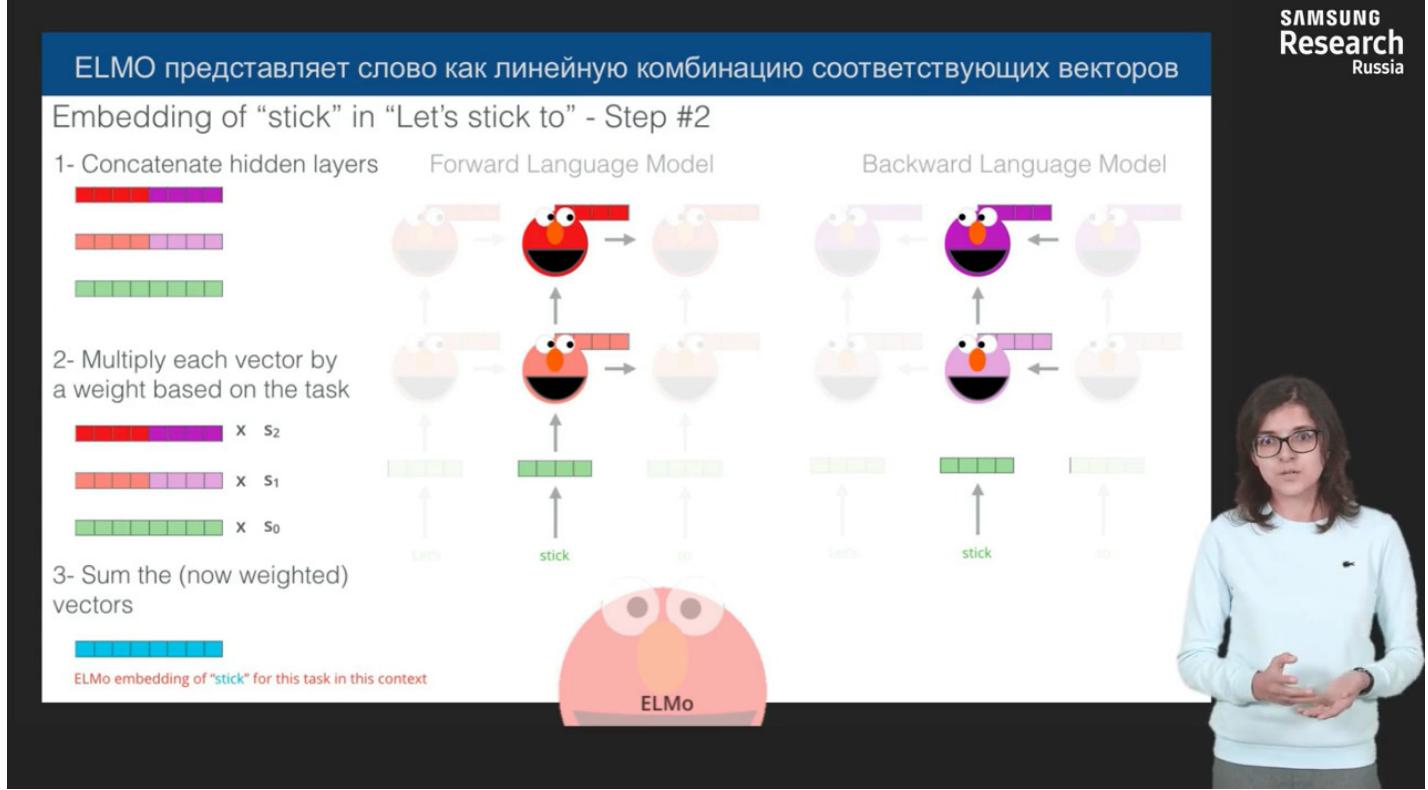
SAMSUNG
Research
Russia

Embedding of "stick" in "Let's stick to" - Step #1

Forward Language Model

Backward Language Model





Итак, мы разобрали [нейронные сети](#), которые появились сравнительно недавно, но, тем не менее, уже стали, в некотором роде, классическими. Теперь давайте пойдём дальше и узнаем, что же происходило позднее, были ли какие-то принципиальные усовершенствования в архитектурах после таких популярных [BERT](#) и [ELMo](#). А также давайте обсудим, что на текущий момент является state of the art (сотами) для многих задач в области обработки текста. Чтобы понять, какие улучшения были придуманы и какое было развитие у идей BERT, ELMo, [OpenAI трансформера](#), нужно, для начала, сформулировать какие проблемы стояли перед исследователями и инженерами^[4] при работе с ванильным [трансформером](#) (ну или с BERT). Например, одна проблема: иногда бывает так, что слово в предложении связано с каким-то термином, который встречался в предыдущем абзаце текста, или даже на предыдущей странице книги.^[5] Такие зависимости называются долговременными. По умолчанию, трансформер умеет работать с контекстом фиксированного размера и все фрагменты обрабатывает отдельно.^[1] Соответственно, некоторые зависимости могут быть не учтены. Особенно речь идёт про долговременные зависимости. Кроме того, текст бьётся на фрагменты произвольно, что может приводить к тому, что предложение разделится на два кусочка и первый кусок попадёт в один фрагмент, а второй кусок — в другой фрагмент, что явно не очень хорошо. Чтобы преодолеть эти проблемы, сотрудники Google придумали новый трансформер и назвали его Transformer-XL.^[3] Основные две идеи в этой архитектуре следующие. Первая — это рекуррентный механизм на основе сегментов, а вторая — это relative positional encoding. Давайте рассмотрим обе эти идеи и чуть подробнее про них поговорим. Итак, идея номер один: рекуррентный механизм на уровне сегментов. Давайте

коротко опишем, в чём заключается основной смысл. Во время обучения, представления, вычисленные для предыдущего сегмента, будут фиксироваться и кэшироваться для повторного использования, и этот расширенный контекст модель будет учитывать, когда она обрабатывает следующий сегмент. Это дополнительное соединение увеличивает максимально возможную длину зависимости в N раз, где N — это глубина сети. И, поскольку теперь контекстная информация может перетекать через границы сегмента, это обеспечивает хорошее качество работы трансформера. Кроме того, этот механизм повторения также решает проблему фрагментации контекста, обеспечивая необходимый контекст для токенов в передней части нового сегмента. Звучит очень круто, и если мы посмотрим на схему, то можно убедиться, что теперь у долговременных зависимостей есть гораздо больше шансов не потеряться и сыграть свою роль в процессе обучения. При этом, у такого подхода всё-таки есть несколько проблем. Например, наивное применение такого рекуррентного механизма не работает совершенно, поскольку, как минимум, позиционные кодировки не являются когерентными при повторном использовании предыдущих сегментов.^[2] Например, давайте рассмотрим старый сегмент с контекстными позициями: "0", "1", "2", "3". Когда новый сегмент обрабатывается, у нас есть позиции "0", "1", "2", "3", "0", "1", "2", "3" — для двух объединённых сегментов, где семантика каждого идентификатора позиции не работает для всей последовательности. Для решения этой проблемы предлагается использовать второй основной концепт из статьи про Transformer-XL — новая схема позиционного кодирования. Она делает возможным механизм повторения, который мы рассмотрели только что, который называется "рекуррентным механизмом на уровне сегментов". Кроме того, в отличие от других схем позиционного кодирования, предложенная авторами Transformer-XL формулировка использует фиксированные [эмбеддинги](#) с обучаемыми трансформациями вместо обучаемых эмбеддингов. И такая схема гораздо лучше обобщается на более длинные последовательности во время тестирования. Когда оба эти подходы объединены, Transformer-XL получает гораздо более длинный эффективный контекст, чем стандартный [ванильный](#) трансформер. Из ещё очевидных плюсов Transformer-XL стоит отметить его быструю скорость работы на этапе тестирования. Авторы статьи отмечают, что им удалось добиться ускорения в 1800 раз при сравнении с ванильным трансформером (на этапе тестирования). Теперь давайте посмотрим на схему и обратим внимание на то, как работает ванильный трансформер и Transformer-XL. Для стандартного трансформера мы видим, что каждый фрагмент вычисляется каждый раз заново. А для Transformer-XL не нужно повторное вычисление, то есть мы видим, что ускорение действительно может быть достаточно большим. Если говорить про остальные преимущества Transformer-XL, то, конечно, стоит заметить то, что Transformer-XL работает с контекстом, который примерно на 80% длиннее чем у [RNN](#), и на 450% процентов длиннее чем у ванильного трансформера. И, также, Transformer-XL гораздо более точен при прогнозировании выборки на длинных последовательностях — как раз из-за моделирования долгосрочных зависимостей. Также, на коротких последовательностях, он тоже работает лучше за счёт решения проблемы фрагментации контекста. Той самой проблемы, когда наше предложение

может разбиться на два куска, и эти кусочки отнесутся к двум разным фрагментам. Но является ли Transformer-XL наилучшей моделью на текущий момент для решения задач обработки текстов? Как оказывается, нет. State of the art (sota), на текущий момент, является сетка под названием [GPT-2](#).

[1] Mikolov T. et al. [Distributed representations of words and phrases and their compositionality](#) //Advances in neural information processing systems. – 2013. – С. 3111-3119.

[2] [Transformer-XL: Unleashing the Potential of Attention Models](#), January 29, 2019, Zhilin Yang and Quoc Le, Google AI

[3] Dai Z. et al. [Transformer-xl: Attentive language models beyond a fixed-length context](#) //arXiv preprint arXiv:1901.02860. – 2019.

[4] Young T. et al. [Recent trends in deep learning based natural language processing](#) //ieee Computational intelligenCe magazine. – 2018. – Т. 13. – №. 3. – С. 55-75.

[5] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. [Neural machine translation by jointly learning to align and translate](#). arXiv preprint arXiv:1409.0473 (2014).

Ограничения ванильного Трансформера

SAMSUNG
Research
Russia

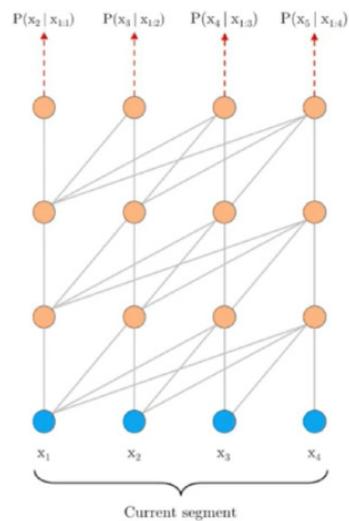
- Контекст фиксированной длины
- Проблема фрагментации контекста: разные части одного предложения могут оказаться в разных фрагментах

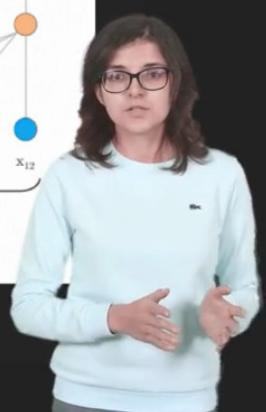
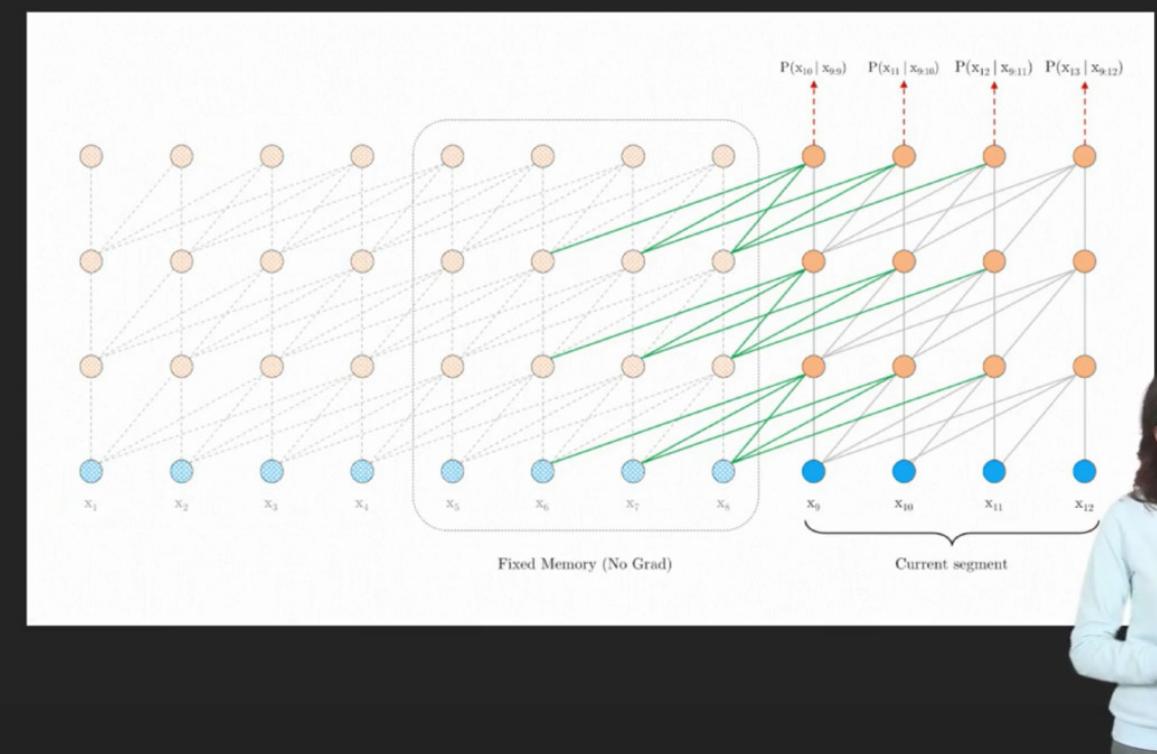
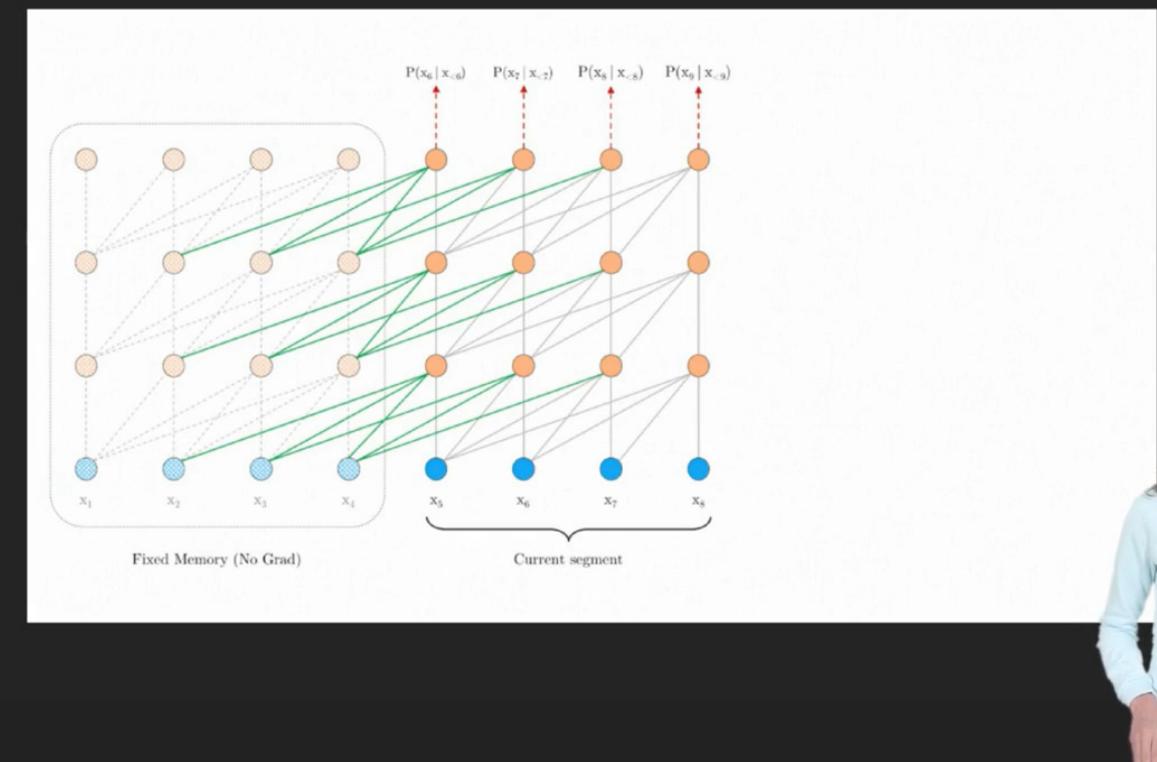
Решение

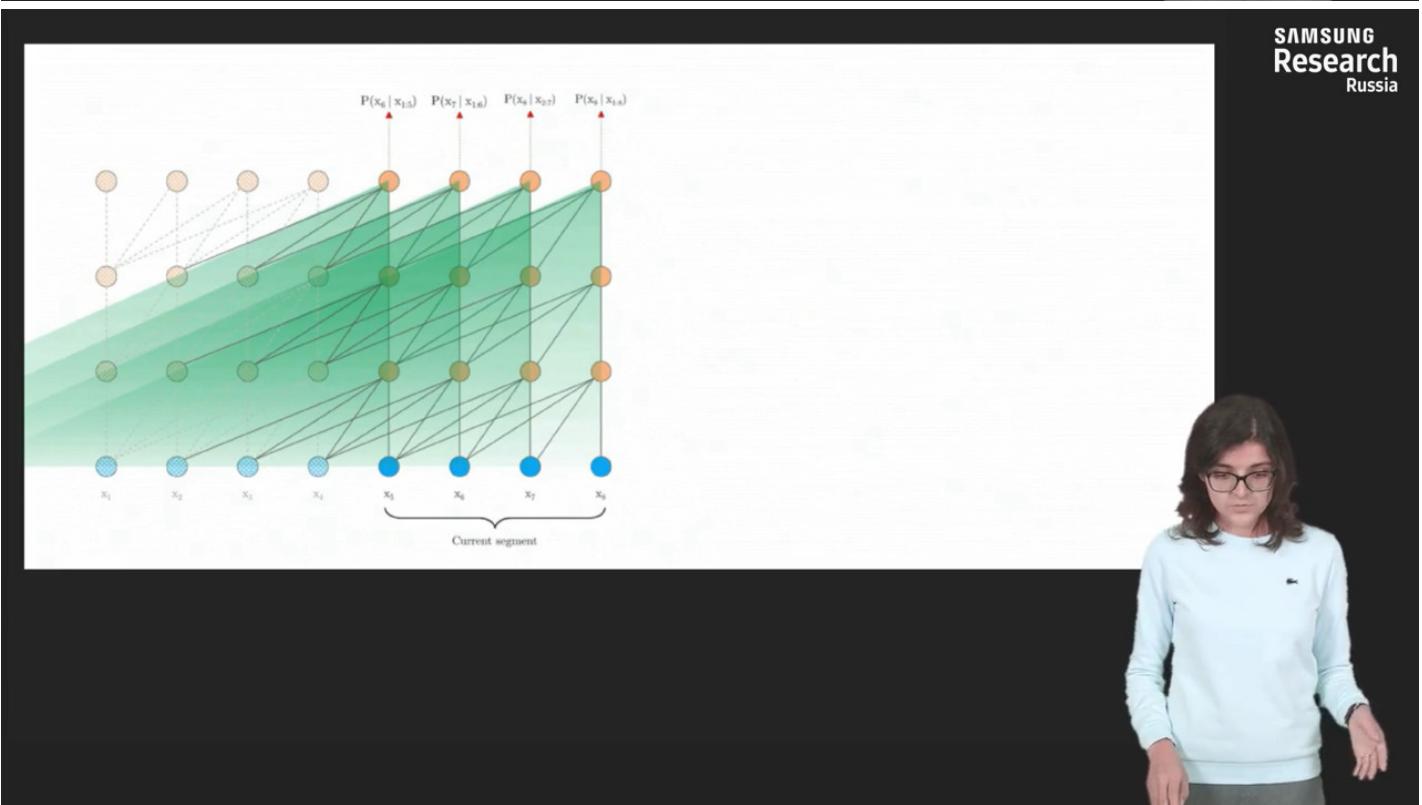
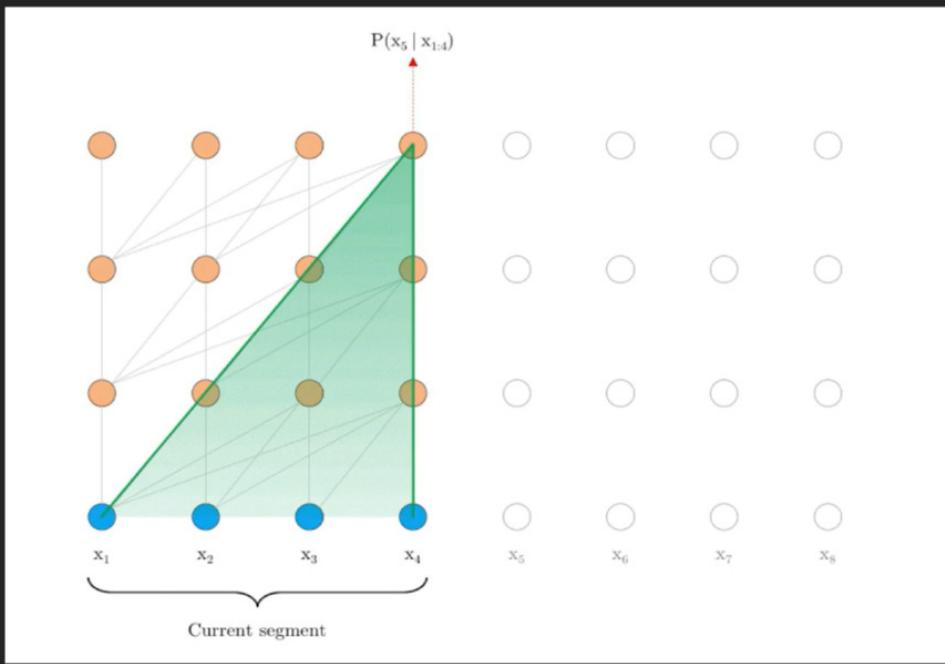
- Рекуррентный механизм на уровне сегментов (segment-level recurrence mechanism)
- Relative Positional Encoding

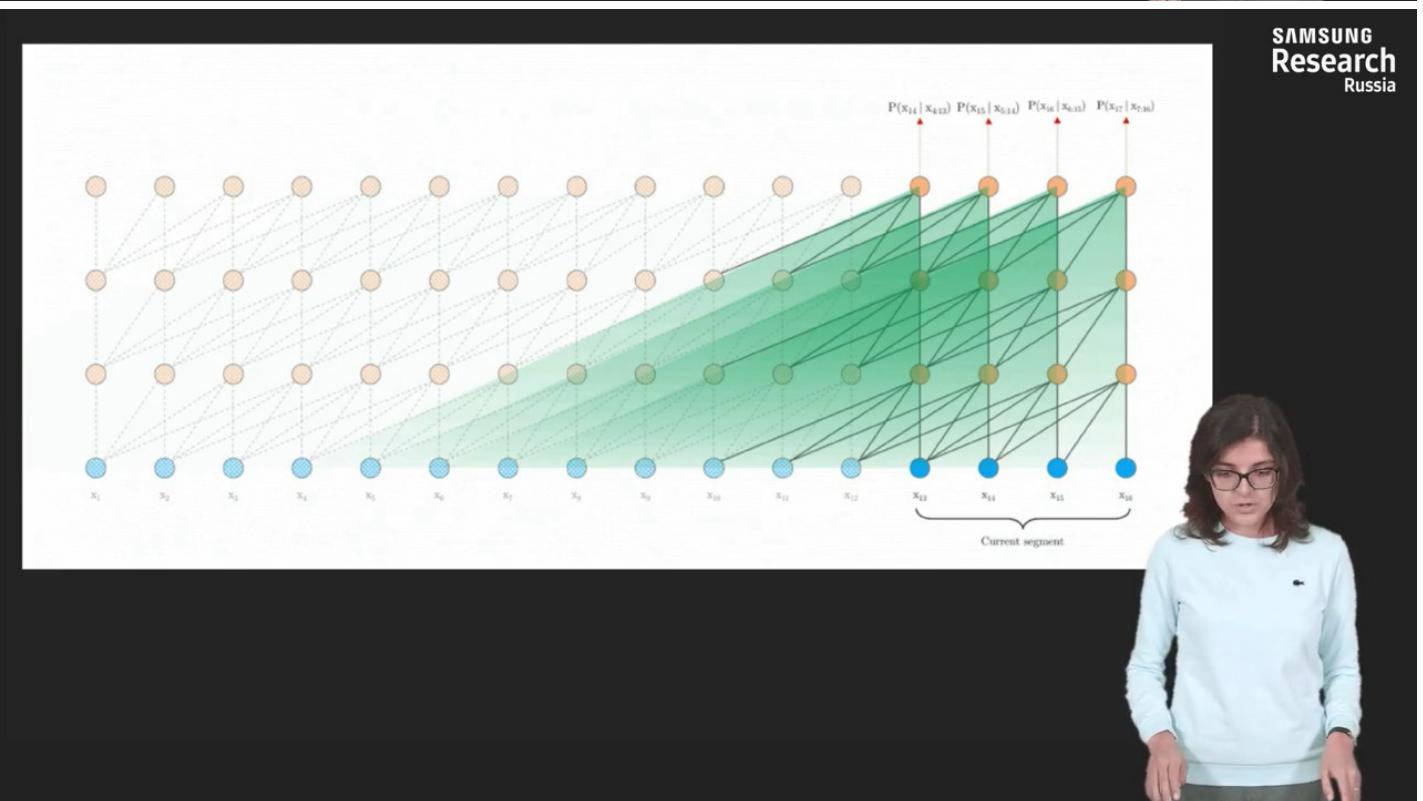
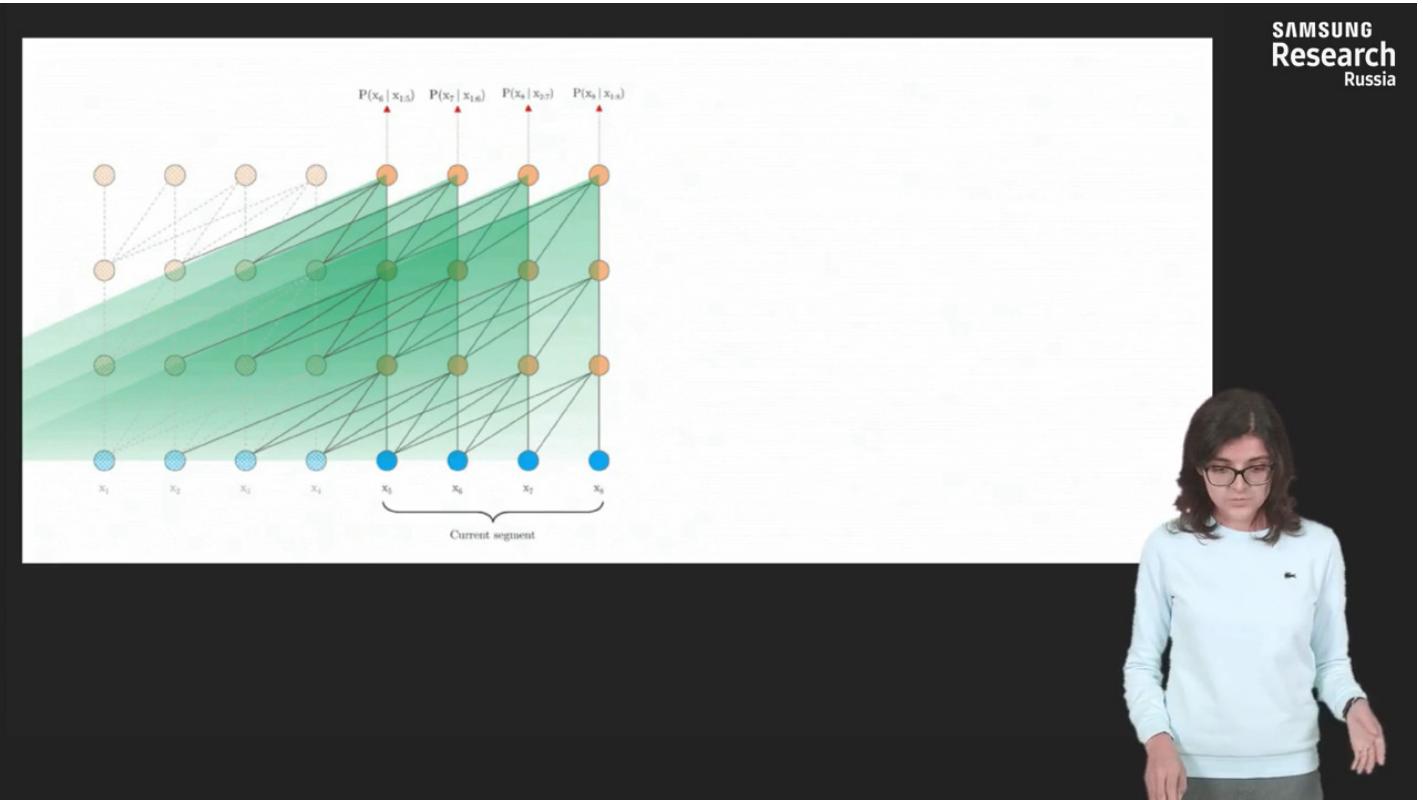


SAMSUNG
Research
Russia









Преимущества

- Трансформер-XL работает с контекстом, который примерно на 80% длиннее, чем у RNN и на 450% длиннее, чем у ванильного трансформера.
- Transformer-XL до 1800+ раз быстрее, чем ванильный трансформер во время тестирования
- Transformer-XL имеет более точен при прогнозировании выборки из-за моделирования долгосрочных зависимостей и решения проблемы фрагментации контекста.



GPT-2 (как следует из названия) — преемник сети под названием GPT. Модель GPT была создана сотрудниками OpenAI и она заимствует некоторые идеи из предыдущий их (достаточно популярной) работы под названием "Sentiment Neuron".^[1] Модель GPT-2 была обучена на уже знакомой вам [задаче языкового моделирования](#). Модель просто учились предсказывать следующее слово. Не было никаких двунаправленных сеток, не было никаких масок. То есть, модель просто предсказывает следующее слово, имея контекст. Но, при этом, она учились просто на огромнейшем датасете размером около 40 гигабайт (то есть, 40 Гб интернет-страниц, интернет-текста). И эта модель работает настолько хорошо, что она вызвала достаточно большой резонанс в сообществе. Выдвигались гипотезы, что с помощью такого мощного инструмента можно будет генерировать, например, фейковые новости или заполнять интернет некачественным контентом, генерировать спам, эмулировать присутствие человека в онлайн переписке — что угодно ещё, любая незаконная деятельность. Эти мысли пошли, собственно, от самих разработчиков GPT-2, от работников OpenAI, и они выразили свои опасения по поводу вредоносных приложений технологии, и поэтому, сначала, отказались релизить полную версию модели. Вместо этого выпустили вариант GPT-2 с меньшим количеством весов, но, при этом, опубликовали статью, выложили веса, выложили туториалы по тому, как нужно использовать модель. Так почему же эта модель так хороша? Давайте немного посмотрим на цифры. Полный вариант модели содержит полтора миллиарда весов и обучалась модель на датасете под названием "WebText".^[2] WebText состоит из восьми миллионов веб-страниц, и данные в этой выборке очень и очень разнообразные. Собственно, это одна из причин — почему с помощью GPT-2 удается добиться state-of-the-art результатов

при решении многих задач. Модель может выучить принципы работы [вопросно-ответных систем](#) или [машинного перевода](#), или систем для суммаризации текстов прямо из сырых текстов, пока она учится решать задачу языкового моделирования на сырых данных. То есть, иногда модель может работать достаточно неплохо даже без [файнтюнинга](#). Тем не менее, в статье про GPT-2 были показаны хорошие результаты для zero-shot learning^[4] только для одной задачи из восьми. Для остальных задач результаты работы модели без дообучения, хоть и не ужасные, но всё ещё далеки от идеальных. Задача, на которой zero-shot learning^[4] для GPT-2 всё-таки сработал — это задача понимания прочитанного текста (или "reading comprehension^[3]"). Датасет для этой задачи выглядел следующим образом — он состоял из пар "документ и диалог про этот документ". Документы относились к семи разным доменам и сетке нужно было научиться отвечать на вопросы по этому документу. Тем не менее, было отмечено, что, хоть GPT-2 и показал отличные результаты, он часто выучивал достаточно простые эвристики. Например, отвечать на вопрос "кто?" единственным встретившимся в тексте именем собственным. Теперь давайте посмотрим на примеры того, как хорошо работает GPT-2 после небольшого дообучения. Примеры на этом слайде я взяла с сайта OpenAI и там представлены несколько задач, на которых GPT-2 работает очень и очень хорошо. Здесь вы видите два вопроса. На первый сеть отвечает правильно, на второй — неправильно. Но, при этом, можете отметить, что во втором вопросе сеть всё-таки поняла что нужно ответить названием штата и смогла это сделать, хотя и назвала неправильный штат. Другой пример — здесь нужно продолжить несколько предложений наиболее логичным словом. И здесь сеть тоже справляется не идеально, но, при этом, она выдаёт правильный по домену ответ, то есть нужно было ответить каким-то словом, связанным с едой, и эту взаимосвязь сеть вполне смогла угадать. Следующий пример достаточно стандартный — это машинный перевод. И с этой задачей сеть GPT-2 справляется вполне хорошо.

[1] <https://openai.com/blog/unsupervised-sentiment-neuron/>

[2] <https://github.com/eukaryote31/openwebtext>

[3] <https://paperswithcode.com/task/reading-comprehension>

[4] <https://paperswithcode.com/task/zero-shot-learning>



ew AI fake text generator may be too dangerous to ... - The Guardian
<https://www.theguardian.com/.../elon-musk-backed-ai-writes-convincing-news-fiction>
 days ago · The Elon Musk-backed nonprofit company OpenAI declines to release research publicly for fear of misuse. The creators of a revolutionary AI system that can write news stories and works of fiction – dubbed “deepfakes for text” – have taken the unusual step of not releasing ...

openAI built a text generator so good, it's considered too dangerous to ...
<https://techcrunch.com/2019/02/17/openai-text-generator-dangerous/>
 ▾ hours ago · A storm is brewing over a new language model, built by non-profit artificial intelligence research company OpenAI, which it says is so good at ...

he AI Text Generator That's Too Dangerous to Make Public | WIRED
<https://www.wired.com/story/ai-text-generator-too-dangerous-to-make-public/>
 ▾ days ago · In 2015, car-and-rocket man Elon Musk joined with influential startup backer Sam Altman to put artificial intelligence on a new, more open ...

on Musk-backed AI Company Claims It Made a Text Generator ...
<https://gizmodo.com/elon-musk-backed-ai-company-claims-it-made-a-text-gener-183...>
 ▾ on Musk-backed AI Company Claims It Made a Text Generator That's Too Dangerous to Release · Rhett Jones · Friday 12:15pm · Filed to: OpenAI Filed to: ...

cientists have made an AI that they think is too dangerous to ...
<https://www.weforum.org/.../amazing-new-ai-churns-out-coherent-paragraphs-of-text/>
 ▾ days ago · Sample outputs suggest that the AI system is an extraordinary step forward, producing text rich with context, nuance and even something ...

ew AI Fake Text Generator May Be Too Dangerous To ... - Slashdot
<https://news.slashdot.org/.../new-ai-fake-text-generator-may-be-too-dangerous-to-rele...>
 ▾ days ago · An anonymous reader shares a report: The creators of a revolutionary AI system that can write news stories and works of fiction – dubbed ...

Top stories

OpenAI built a text generator so good, it's considered too dangerous to release <small>TechCrunch</small> <small>11 hours ago</small>	Elon Musk's AI company created a fake news generator it's too scared to make public <small>BGR.com</small> <small>9 hours ago</small>	The AI That Can Write A Fake News Story From A Handful Of Words <small>NDTV.com</small> <small>2 hours ago</small>

When Is Technology Too Dangerous to Release to the Public?
Slate · 2 days ago

Scientists Developed an AI So Advanced They Say It's Too Dangerous to Release
ScienceAlert · 6 days ago



- 1.5 миллиарда весов
- Обучался на датасете **WebText** (8 миллионов веб-страниц)
- Zero-shot learning работает (хотя и не всегда дает очень хорошие результаты)



EXAMPLES

Who wrote the book the origin of species?

Correct answer: Charles Darwin

Model answer: Charles Darwin

What is the largest state in the U.S. by land mass?

Correct answer: Alaska

Model answer: California



EXAMPLE

Both its sun-speckled shade and the cool grass beneath were a welcome respite after the stifling kitchen, and I was glad to relax against the tree's rough, brittle bark and begin my breakfast of buttery, toasted bread and fresh fruit. Even the water was tasty; it was so clean and cold. It almost made up for the lack of...

Correct answer: coffee

Model answer: food



EXAMPLE

French sentence:

Un homme a expliqué que l'opération gratuite qu'il avait subie pour soigner une hernie lui permettrait de travailler à nouveau.

Reference translation:

One man explained that the free hernia surgery he'd received will allow him to work again

Model translation:

A man told me that the operation gratuity he had been promised would not allow him to travel.



На этом мы заканчиваем обзор популярных в последнее время нейросетей и заканчиваем разговор про [transfer learning](#) в сфере обработки текстов. Мы обсудили пять популярных архитектур, составили своё представление о том, что же происходит в области обработки текстов в настоящее время и получили некоторое представление о том, какие архитектуры

сейчас популярны и какие основные концепты лежат в их основе. Для более подробного понимания темы я рекомендую ознакомиться с информацией, которая доступна по ссылкам с этого слайда. Информация из обзоров под названием "illustrated transformer^[1]" и "illustrated BERT^[2]" использовалась в этой лекции, но, для того чтобы более подробно понять, что же творится под капотом **BERT**, я рекомендую всё-таки ознакомиться с этим обзором и посмотреть на отличные иллюстрации, которые там представлены. Удачи в изучении материалов!

[1] <http://jalammar.github.io/illustrated-transformer/>

[2] <http://jalammar.github.io/illustrated-bert/>

[3] <http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture14-contextual-representations.pdf>

[4] <https://medium.com/mlreview/understanding-building-blocks-of-ulmfit-818d3775325b>

Что было в этой лекции

- OpenAI Transformer
- BERT
- ELMO
- Transformer-XL
- GPT и GPT-2

SAMSUNG
Research
Russia

Материалы

- <http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture13-contextual-representations.pdf>
- <http://jalammar.github.io/illustrated-transformer/>
- <http://jalammar.github.io/illustrated-bert/>
- <https://medium.com/mlreview/understanding-building-blocks-of-ulmfit-818d3775325b>

A woman with glasses and a light blue sweatshirt stands to the right of the presentation slide, looking towards the camera.

Что еще почитать

Список ссылок на статьи про архитектуры, рассмотренные в лекции:

1. OpenAI Transformer: [Improving Language Understanding by Generative Pre-Training](#).
2. ELMO: [Deep contextualized word representations](#)
3. BERT: [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#)
4. Transformer-XL: [Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context](#)
5. GPT-2: [Language Models are Unsupervised Multitask Learners](#)

Серия статей от [Jay Alammar](#) (отличные посты с очень понятными картинками):

1. [Illustrated Transformer](#)
2. [Illustrated BERT](#) (материалы отсюда были использованы в лекции)
3. [Illustrated GPT-2](#)

Еще немного блог-постов:

1. [ELMO от AllenNLP](#)
2. [Transformer-XL от Google AI](#)
3. [Простая статья на Medium про Transformer-XL](#)

Ссылки на код моделей:

1. [OpenAI Transformer](#)
2. [ELMO \(Tensorflow\)](#)
3. [BERT \(Tensorflow\)](#)
4. [Transformer-XL](#)
5. [GPT-2](#)
6. Обновление от того же автора по GPT-3: <http://jalammar.github.io/how-gpt3-works-visualizations-animations/>