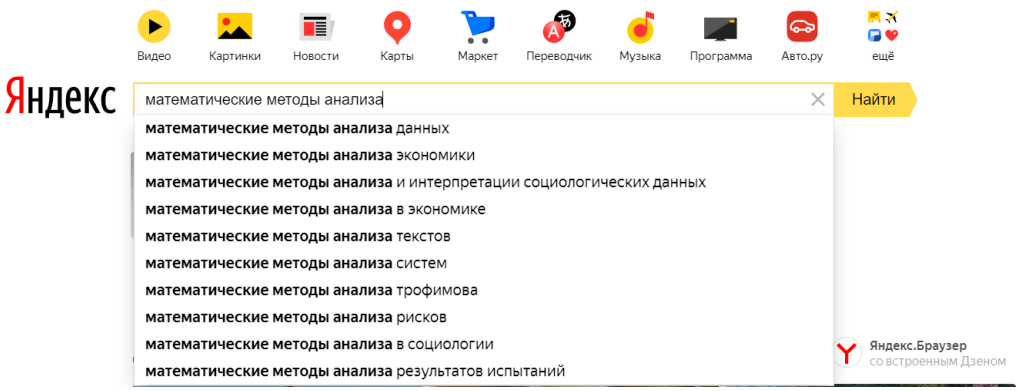
Языковое моделирование. Генерация текста.

Математические методы анализа текстов осень 2021

Попов Артём Сергеевич

3 октября 2021 г.

Поисковые подсказки



Постановка задачи языкового моделирования

Языковая модель решает одну из двух задач:

- оценить вероятность появления произвольной последовательности слов $(w_1, \ldots, w_n)$ в тексте, $w_i \in W$
- оценить вероятность появления произвольного слова w_n после произвольной последовательности слов $(w_1, \ldots, w_{n-1})$, $w_i \in W$

Задачи являются взаимозаменяемыми в силу цепного правила (chain rule) и формулы условной вероятности:

$$p(w_1, \ldots, w_n) = p(w_n|w_1, \ldots, w_{n-1}) \ldots p(w_2|w_1)p(w_1)$$

$$p(w_n|w_1, \ldots, w_{n-1}) = \frac{p(w_1, \ldots, w_n)}{p(w_1, \ldots, w_{n-1})}$$

Марковское правило

Проблема: с ростом n число всевозможных последовательностей растёт $\Rightarrow$ большое число вероятностей $p(w_1, \ldots, w_n)$ будет близко к нулю.

Упростим модель, воспользовавшись марковским свойством:

$$p(w_n|w_1, \ldots, w_{n-1}) \approx p(w_n|w_{n-k}, \ldots, w_{n-1}), \quad k \ll n$$

Таким образом:

$$p(w_1, \ldots, w_n) = \prod_{i=1}^{n} p(w_i|w_{i-k}, \ldots, w_{i-1})$$

Обучающая выборка и критерии качества

Для обучения модели будем использовать неразмеченный корпус D.

Критерии качества:

- Любая метрика классификации на отложенной выборке для задачи предсказания следующего слова по предыдущим
- Правдоподобие (может оцениваться по отложенной выборке):

$$\mathcal{L}(D) = \sum_{d \in D} \log p(d)$$

- Перплексия (может оцениваться по отложенной выборке):

$$\mathcal{P}(D) = \exp\left(-\frac{1}{|D|}\mathcal{L}(D)\right)$$

Приложения языкового моделирования

Задачи, связанные с генерацией данных:

- Поисковые подсказки
- Автодополнение кода
- Генерация текста
- Генерация данных, имеющих последовательную структуру

Постобработка результатов для задач с сложным выходом:

- Исправление опечаток
- Распознавание символов
- Распознавание речи
- Машинный перевод

Простейшие n-граммные модели

Используем численную оценку вероятности для фиксированного k:

$$p(w_n|w_{n-k},\dots,w_{n-1}) = \frac{c\left(w_{n-k},\dots,w_n\right)}{c\left(w_{n-k},\dots,w_{n-1}\right)},$$

где $c(w_i,\dots,w_{i+k})$ – число последовательностей $w_i,\dots,w_{i+k}$ в корпусе D.

Обучение модели — запоминание статистик появления последовательностей.

Модели с небольшой «граммностью» имеют специальные названия:

- $k = 0$ — униграммная модель, $p(w_1,\dots,w_3) = p(w_3)p(w_2)p(w_1)$
- $k = 1$ — биграммная модель, $p(w_1,\dots,w_3) = p(w_3|w_2)p(w_2|w_1)p(w_1)$
- $k = 2$ — триграммная модель, $p(w_1,\dots,w_3) = p(w_3|w_2,w_1)p(w_2|w_1)p(w_1)$

Проблема первого токена в биграммной модели

Предложение, для которого хотим оценить вероятность:

$$d = (i, will, be, back)$$

Возникают проблемы с первым токеном в биграммной модели, необходимо заводить отдельную структуру для хранения униграмм первой позиции:

$$p(d) = p(back|be)p(be|will)p(will|i)p(i)$$

Чтобы оставаться в «классе биграмм», добавим фиктивный токен <start>:

$$p(d) = p(back|be)p(be|will)p(will|i)p(i|<start>)$$

Сглаживание Лапласа (add-k smoothing)

В n-граммной модели большое количество нулевых вероятностей.

Если в последовательности любой длины хотя бы одна k-грамма отсутствует в обучении, вероятность последовательности будет нулевой.

Для уменьшения числа нулевых вероятностей можно использовать сглаживание Лапласа:

$$p(w_n|w_{n-k},\dots,w_{n-1}) = \frac{c\left(w_{n-k},\dots,w_n\right)+\alpha}{c\left(w_{n-k},\dots w_{n-1}\right)+|W|\alpha}$$

Сглаживание Лапласа может сильно повлиять на порядок вероятностей и форму распределения.

Интерполяционное сглаживание (Jelinek-Mercer smoothing)

Вероятность оценивается смесью из нескольких n-граммных моделей разного порядка:

$$p(w_n|w_{n-k},\dots,w_{n-1}) = \sum_{c=1}^{k} \lambda_i \hat{p}(w_n|w_{n-c},\dots,w_{n-1})$$

$$\sum_{c=1}^{k} \lambda_c = 1, \quad \text{обычно } k \approx 3$$

Откат последовательности (Katz smoothing)

Если не получается применить модель высокого порядка, пробуем применить модель меньшего порядка с понижающим множителем:

$$p(w_n|w_{n-k},\dots,w_{n-1}) = \\ = \begin{cases} \alpha(w_{n-k},\dots,w_n)\hat{p}(w_n|w_{n-k},\dots,w_{n-1}), & \text{если } c(w_{n-k},\dots,w_n) > 0 \\ \beta(w_{n-k},\dots,w_n)p(w_n|w_{n-k+1},\dots w_{n-1}), & \text{иначе} \end{cases}$$

где $\alpha(\dots)$ и $\beta(\dots)$ выбираются исходя из нормировки вероятностей.

Влияние сглаживания: модельный пример

Построение биграммной модели без учёта сглаживания:

```
counts = np.random.zipf(a=3, size=(100, 100)) - 1
probs = counts / counts.sum(axis=1)[:, None]
```

Функция для вычисления вероятностей триграмм по биграммной модели:

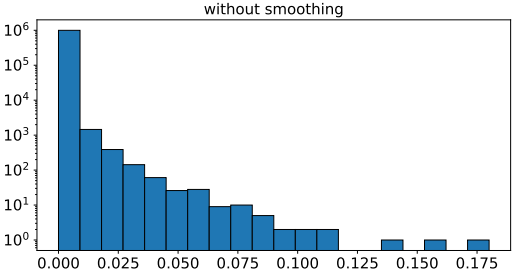
```
def get_trigram_probs(probs):
    return (probs[:, :, None] * probs[None, :, :]).ravel()
```

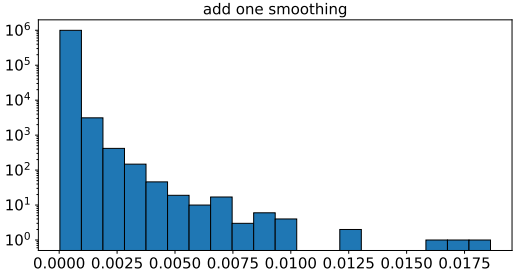
Влияние сглаживания: модельный пример

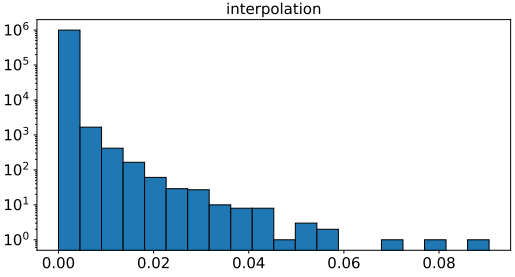
Задание моделей с разным сглаживанием:

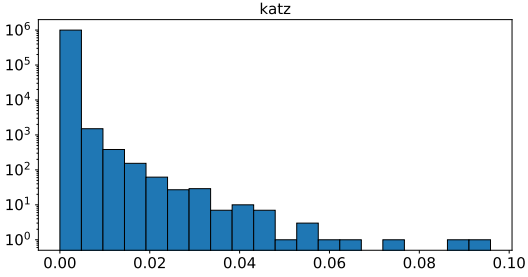
```
# сглаживание лапласа
addk_counts = counts + 1
addk_probs = addk_counts / addk_counts.sum(axis=1))[:, None]

# интерполяционное сглаживание
uni_probs = counts.sum(axis=1) / counts.sum()
interpolation_probs = 0.7 * probs + 0.3 * uni_probs

# сглаживание через откат
katz_counts = 0.7 * probs + 0.3 * uni_probs * np.int8(probs == 0)
katz_probs = katz_counts / katz_counts.sum(axis=1)[:, None]
```

Влияние сглаживания: общая картина

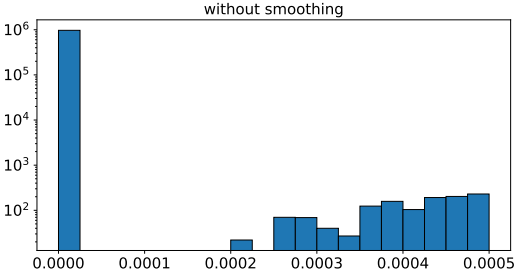


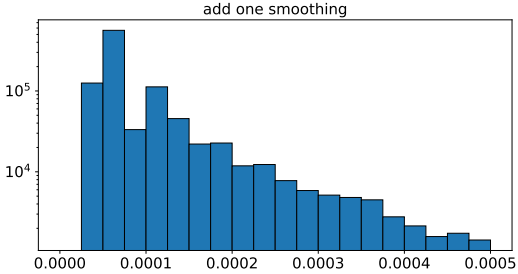


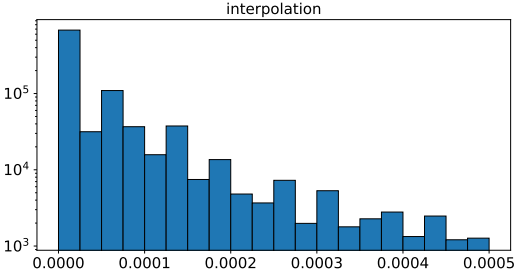


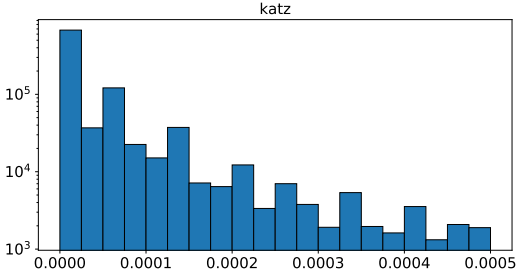


Влияние сглаживания: вблизи нуля









Другие виды сглаживания

Мы рассмотрели далеко не все виды сглаживания[1]:

- Good-Turing estimate
- Witten-Bell smoothing
- Absolute discounting
- Kneser-Ney smoothing — один из самых популярных способов, сглаживание зависит от вариативности контекстов

[1] N-граммным моделям посвящена отдельная глава в учебнике Jurafsky и Martin: ссылка.

Задача исправления опечаток (spellchecking)

Необходимо по входной строке получить её исправленный вариант.

Задача большая и разнообразная:

- ошибки пропуска/вставки символов
- ошибки склейки/расклейки слов
- ошибки неправильного написания слов
- неверная раскладка
- написание транслитерацией

Из-за сложности задачи, часто лучше тренировать модели, устойчивые к опечаткам, чем пытаться исправить опечатки на этапе предобработки.

Модель шумного канала для исправления опечаток

- Исходное слово w подаётся в канал и искажается в слово $\hat{w}$.
- Хотим восстановить исходное слово w по $\hat{w}$.
- Пусть $p(v)$ – языковая модель, $p(\hat{w} \mid v)$ – модель канала:

$$w = \operatorname*{argmax}_{u \in W} p(u \mid \hat{w}) = \operatorname*{argmax}_{u \in W} \frac{p(u, \hat{w})}{p(\hat{w})} = \operatorname*{argmax}_{u \in W} p(u)\ p(\hat{w} \mid u)$$

Простейшая модель шумного канала — ненулевые вероятности имеют слова, которые находятся на расстоянии Левенштейна $\leqslant t$ от заданного слова.

Расстояние Левенштейна — минимальное количество операций вставки, удаления и замены символа, необходимых для превращения одной последовательности символов в другую.

Алгоритм исправления опечаток

1. Разбиваем строку на слова, обрабатываем каждое слово отдельно.
2. Если слова нет в словаре, генерируем для него кандидатов на исправление, используя заданное число элементарных операций .
3. Удаляем кандидатов, не входящих в словарь.
4. При помощи языковой модели выбираем лучшего кандидата.

На последнем шаге можно использовать языковую модель любого порядка или специально обученный классификатор.

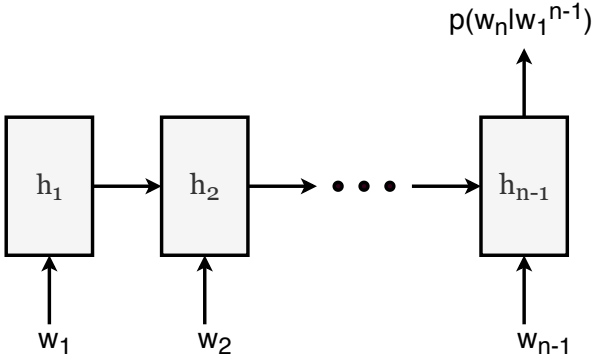
[1] Глава в Juravsky, Martin: ссылка
[2] Спеллчекер Норвига: ссылка

Резюме по n-граммным моделям

- Быстро обучаются
- Требуют много памяти для хранения
- Нет смысла использовать большие k
- Можно использовать для построения автодополнения, генерации признаков или решения задач в формате шумного канала
- Не нужно использовать для генерации текста
- Есть готовые реализации в библиотеке nltk
- Есть готовые реализации спеллчекеров с поддержкой языковых моделей: библиотека Jamspell, `levenshtein_corrector` в библиотеке deeppavlov

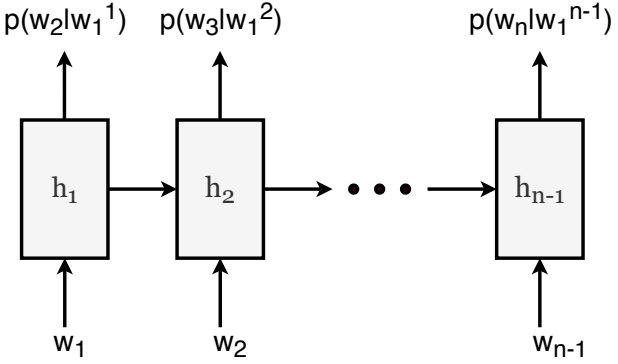
Использование нейросети для языкового моделирования

Идея 1: моделировать $p(w_n = w | w_1, \ldots, w_{n-1})$ при помощи нейросети для обработки последовательности (RNN, CNN, Transformer).



Использование нейросети для языкового моделирования

Идея 2: использовать «однонаправленные» модели и делать предсказания на всех выходах, i-ый выход соответствует $p(w_{i+1} = w | w_1, \dots w_i)$.



Общий алгоритм работы нейросети для языкового моделирования

1. На вход подаётся последовательность токенов:

$$w_1^{n-1} = \{w_1, \ldots, w_{n-1}\}, \quad w_i \in W$$

2. Каждый токен преобразуется в его эмбеддинг:

$$v_1^{n-1} = Embedding(w_1^{n-1}) = \{v_1, \ldots, v_{n-1}\}$$

3. Эмбеддинги подаются в слои для обработки последовательности:

$$h_1^{n-1} = model(v_1^{n-1}) = \{h_1, \ldots, h_{n-1}\}$$

4. К выходам на заданных позициях применяется линейный слой:

$$o_i = Uh_i + b$$

5. Функционал для обучения:

$$-\sum_{i=1}^{n-1} \log p(w = w_{i+1}|w_1^i) = -\sum_{i=1}^{n-1} \log \operatorname*{softmax}_{w \in W} o_{tw}|_{w=w_{i+1}}$$

Однонаправленный архитектуры: RNN и трансформер

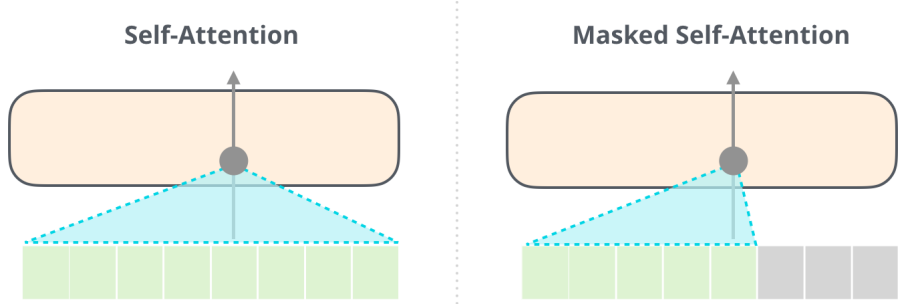
Рекуррентные сети по умолчанию являются однонаправленными.
Как сделать однонаправленным трансформер?

[1]Alammar; The Illustrated GPT-2: ссылка

Однонаправленный архитектуры: RNN и трансформер

Рекуррентные сети по умолчанию являются однонаправленными. Как сделать однонаправленным трансформер?

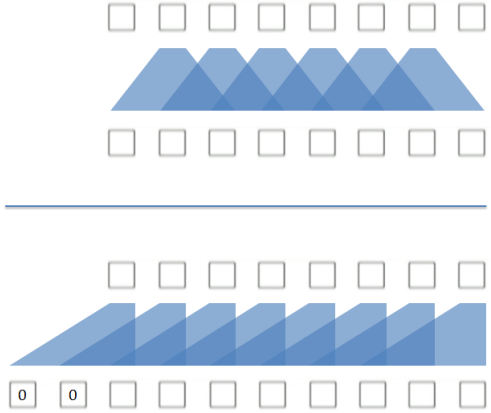
Внимание в трансформере должно смотреть только на предыдущие токены. Это соответствует архитектуре декодировщика трансформера:



[1] Alammar; The Illustrated GPT-2: ссылка

Однонаправленные архитетуры: CNN

В свёрточной сети используем свёртки только по левому контексту ячейки:



[1]Dauphin et al (ICML 2017); Language Modeling with Gated Convolutional Networks

Модели GPT (generative pre-training)

GPT это:

1. популярные архитектуры для языкового моделирования
2. конкретные модели, обученные openAI (GPT1–3)

Устройство моделей GPT:

- на входе и выходе — BPE токены (Byte pair encoding)
- архитектура всех моделей GPT — несколько блоков декодировщика трансформера с минимальными изменениями
- сеть обучается на задачу языкового моделирования по большому корпусу текстов

[1] Radford et al; Improving Language Understanding by Generative Pre-Training (2018)
[2] Radford et al; Language Models are Unsupervised Multitask Learners (2019)
[3] Brown et al; Language Models are Few-Shot Learners; (2020)

Byte-pair encoding (BPE)

Byte-pair encoding — способ работы с большим словарём.

Алгоритм BPE (обучение):

1. Исходный словарь — множество символов корпуса, исходный набор правил — пустое множество
2. На каждой итерации добавляем в словарь самую часто совстречаемую в корпусе пару двух элементов словаря a, b и правило $\{a\ b \to ab\}$

Алгоритм BPE (применение): последовательно применяем каждое из полученных правил.

[1] Gage (C Users Journal 1994), A New Algorithm for Data Compression

[2] Sennrich et al (ACL 2016), Neural Machine Translation of Rare Words with Subword Units

BPE на примере

Проведём 5 итераций обучения алгоритма на предложении «she sells seashells by the seashore»

1. s h e | s e l l s | s e a s h e l l s | b y | t h e | s e a s h o r e

BPE на примере

Проведём 5 итераций обучения алгоритма на предложении «she sells seashells by the seashore»

1. s h e | s e l l s | s e a s h e l l s | b y | t h e | s e a s h o r e
   {s h → sh}
2. sh e | s e l l s | s e a sh e l l s | b y | t h e | s e a sh o r e

BPE на примере

Проведём 5 итераций обучения алгоритма на предложении «she sells seashells by the seashore»

1. s h e | s e l l s | s e a s h e l l s | b y | t h e | s e a s h o r e
   {s h → sh}
2. sh e | s e l l s | s e a sh e l l s | b y | t h e | s e a sh o r e
   {s h → sh, s e → se}
3. sh e | se l l s | se a sh e l l s | b y | t h e | se a sh o r e

BPE на примере

Проведём 5 итераций обучения алгоритма на предложении «she sells seashells by the seashore»

1. s h e | s e l l s | s e a s h e l l s | b y | t h e | s e a s h o r e
   {s h → sh}
2. sh e | s e l l s | s e a sh e l l s | b y | t h e | s e a sh o r e
   {s h → sh, s e → se}
3. sh e | se l l s | se a sh e l l s | b y | t h e | se a sh o r e
   {s h → sh, s e → se, l l → ll}
4. sh e | se ll s | se a sh e ll s | b y | t h e | se a sh o r e

BPE на примере

Проведём 5 итераций обучения алгоритма на предложении «she sells seashells by the seashore»

1. s h e | s e l l s | s e a s h e l l s | b y | t h e | s e a s h o r e
   {s h → sh}
2. sh e | s e l l s | s e a sh e l l s | b y | t h e | s e a sh o r e
   {s h → sh, s e → se}
3. sh e | se l l s | se a sh e l l s | b y | t h e | se a sh o r e
   {s h → sh, s e → se, l l → ll}
4. sh e | se ll s | se a sh e ll s | b y | t h e | se a sh o r e
   {s h → sh, s e → se, l l → ll, se a → sea}
5. sh e | se ll s | sea sh e ll s | b y | t h e | sea sh o r e

Размеры моделей GPT

	Размер корпуса	Словарь	Число слоёв	Размер слоя	Число параметров
GPT	7k books	40k	12	768	$\leqslant$ 117M
GPT2 small	40 GB	50k	12	768	117M
GPT2 medium	40 GB	50k	24	1024	345M
GPT2 large	40 GB	50k	36	1280	762M
GPT2 extra large	40 GB	50k	48	1600	1542M
GPT-3 small	$\approx$ 1TB	??	12	768	125M
GPT-3 medium	$\approx$ 1TB	??	24	1024	350M
GPT-3 large	$\approx$ 1TB	??	24	1536	760M
GPT-3 13B	$\approx$ 1TB	??	40	5140	13.0B

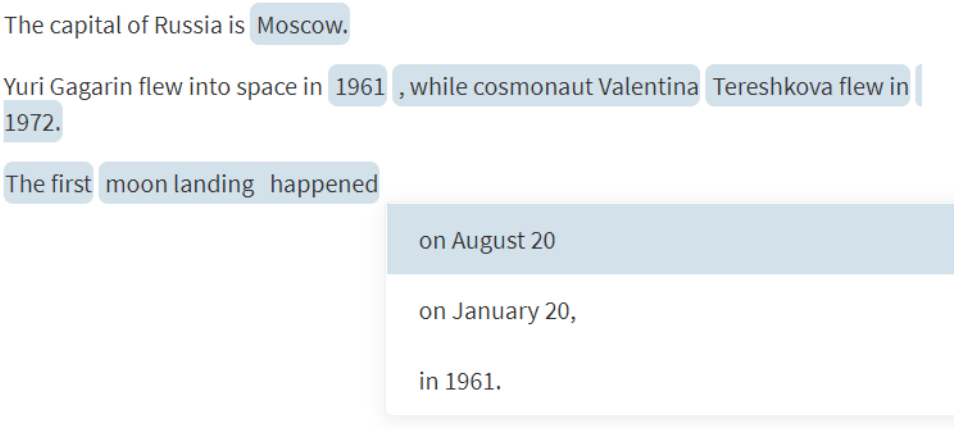
Что умеет GPT: AI Dungeon

AI Dungeon (ссылка) — текстовая игра, в которой созданием мира, реакцией на любые действия игрока занимается GPT3.

One night you have a dream. In the dream, the earth begins to shake and you see an opening in the ground. ***You see a huge train underground.*** *It is an engine made of old iron with a small cab on the front. Smoke comes out of the stack and you think to yourself "I must be near a great big machine that creates a lot of smoke."* ***The train stops. People start to come out of it.*** *They are all heavily dressed in thick clothing and carry weapons. "That's strange,"you think to yourself, "People shouldn't be arriving at the train when it is still underground."* ***All people are in a hurry somewhere. They knock you off your feet.*** *You try to get up, but a very large man nearly breaks your neck with one blow from his fist. You see more arriving everyday. You begin to wonder: "What is going on".* ***Suddenly, you understand everything. This is the metro at rush hour.*** *This is the busiest time of day and night. Every day hundreds of people come through here. They have their own agenda and own goals. However, at this moment you see something burning in the distance.*

Что умеет GPT: zero-shot learning

За счёт обучения на большом объёме данных GPT можно использовать в режиме zero-shot learning, например, как вопросно-ответную систему:



[1]Интерфейс: ссылка

Использование нейросетевых языковых моделей

Можно использовать напрямую для задач генерации текста и автодополнения.

Можно использовать для любых других задач связанных с генерацией текста, применяя подходы:

- Обучение без подготовки (zero-shot learning)
- Обучение с небольшой подготовкой (few-shot learning)
- Использование выходов с промежуточных слоёв для контекстных эмбеддингов (context embeddings)
- Дообучение модели (fine-tuning)

Алгоритм поэлементной генерации последовательности

Вход модели — токен <START> или токены уже написанного текста.

Повторяем до тех пор, пока не будет достигнута максимальная длина последовательности или не будет сгенерирован токен <END>:

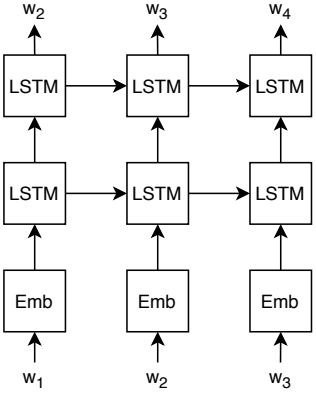
1. Получение из модели логитов $q(w|w_1^i)$
2. Получение из логитов распределения $p(w|w_1^i)$
3. Выбор нового токена w_{i+1} из полученного распределения

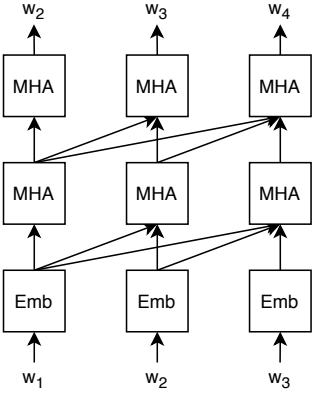
Эффективный пересчёт вероятностей при генерации

Как не пересчитывать $q(w|w_1, \ldots, w_i)$ с нуля на каждом шаге генерации?.

LSTM: храним предыдущие состояния h_i, C_i для каждого слоя.

Трансформер: храним key, value на предыдущих позициях для всех слоёв.





Стратегии выбора токена

- Наиболее вероятный токен:

$$w_{i+1} = \arg\max_{w \in W} p(w|w_1^i)$$

- Семплилированный из распределения токен:

$$w_{i+1} \sim p(w|w_1^i)$$

Для автодополнения лучше использовать детерминированную стратегию.
При генерации длинного текста лучше использовать семплирование.

Детерминированная стратегия для длинной последовательности может привести к зацикливанию — повторению одной и той же n-граммы.

Получение распределения: $\mathrm{softmax}$ с температурой

Здесь и далее полагаем $q_w \equiv q(w|w_1^i)$.

- Стандартное преобразование — $\mathrm{softmax}$:

$$p(w|w_1^i) = \underset{w \in W}{\mathrm{softmax}}\, q_w = \frac{\exp(q_w)}{\sum_{u \in W} \exp(q_u)}$$

- Повышение/понижение температуры:

$$p(w|w_1^i) = \underset{w \in W}{\mathrm{softmax}} \left(\frac{q_w}{\tau} \right)$$

При малых τ распределение стремится к вырожденному.
При больших τ распределение стремится к равномерному.

Получение распределения: topK и topP

Для уменьшения шума будем занулять все элементы распределения кроме нескольких максимальных.

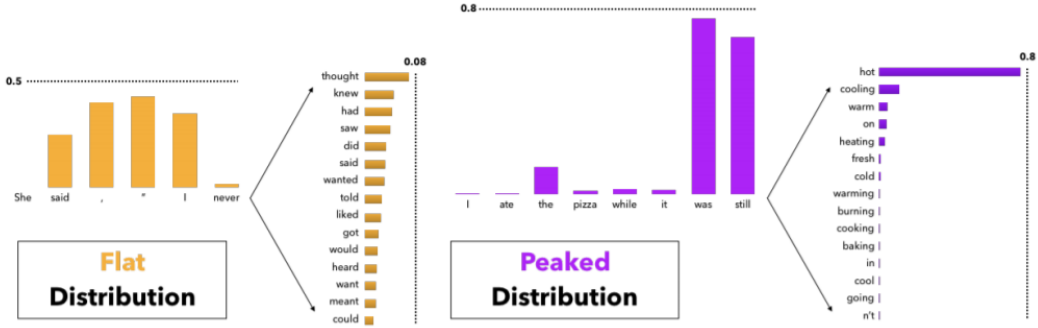
- TopK стратегия. V_k — k токенов с наибольшими вероятностями.
- TopP (nucleus) стратегия. V_p — наименьшее по количеству элементов множество токенов с суммой вероятностей больше p.

$$\hat{p}_w = \underset{w \in W}{\operatorname{softmax}}\, q_w$$

$$p(w|w_1^i) = \frac{\hat{p}_w \mathbb{I}[w \in V_k]}{Z}$$

[1] Holtzman et al (ICLR 2020); The Curious Case Of Neural Text Degeneration

Почему topP потенциально лучше topK



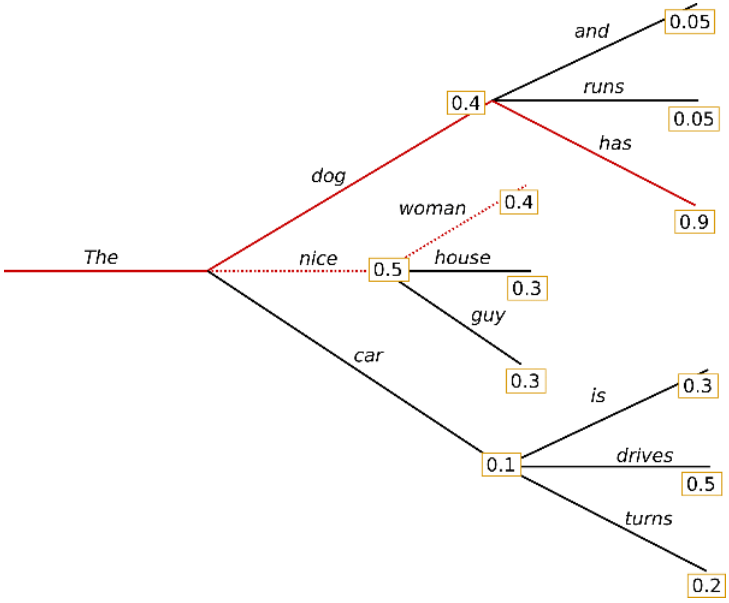
Beam search (лучевой поиск) для генерации последовательности

Вход модели — одна последовательность, состоящая из токена <START> или токенов уже написанного текста.

Повторяем до тех пор, пока не будет получено m законченных последовательностей (достигнута максимальная длина последовательности или был сгенерирован токен <END>):

1. Получение из модели логитов для всех имеющихся последовательностей
2. Получение из логитов распределений
3. Выбор новых m токенов для каждого из полученных распределений
4. Подсчёт вероятности каждой последовательности (chain-rule)
5. Отбор m наиболее вероятных последовательностей
6. Если получили законченную последовательность, не продолжаем её

Иллюстрация работы beam search



Ещё одна иллюстрация работы beam search

Свойства beam search

- Beam search выбирает наиболее вероятную последовательность, а не последовательность наиболее вероятных токенов
- Beam search необходим, если мы хотим выдавать пользователю несколько вариантов ответа
- Beam search с разумными m практически всегда позволяет получить более адекватные результаты, чем при $m = 1$
- Большие m в beam search приводит к практически одинаковым «безопасным» последовательностям
- При больших m большинство последовательностей будут короткими
- Beam search часто используют с детерминированными стратегиями выбора токена и температурой, реже с семплированием

Сравнение разных стратегий генерации

У некоторых методов перплексия меньше перплексии реального текста, аналог переобучения...

Method	Perplexity	Self-BLEU4	Zipf Coefficient	Repetition %	HUSE
Human	12.38	0.31	0.93	0.28	-
Greedy	1.50	0.50	1.00	73.66	-
Beam, b=16	1.48	0.44	0.94	28.94	-
Stochastic Beam, b=16	19.20	0.28	0.91	0.32	-
Pure Sampling	22.73	0.28	**0.93**	0.22	0.67
Sampling, t=0.9	10.25	0.35	0.96	0.66	0.79
Top-k=40	6.88	0.39	0.96	0.78	0.19
Top-k=640	13.82	**0.32**	0.96	**0.28**	0.94
Top-k=40, t=0.7	3.48	0.44	1.00	8.86	0.08
Nucleus p=0.95	**13.13**	**0.32**	0.95	0.36	**0.97**

Отличия в сети при генерации и обучении

Есть существенные отличия на этапах обучения и применения:

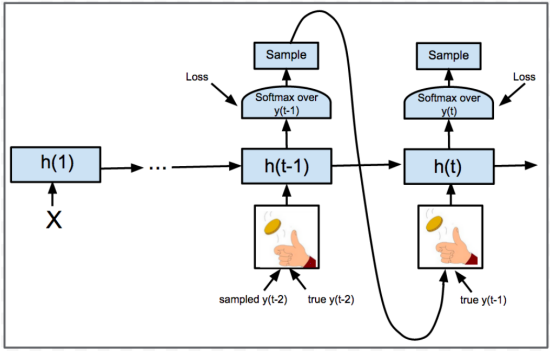
Этап	На входе ячейки
Обучение	Истинное w_i
Применение	Предсказанное $\hat{w}_i$

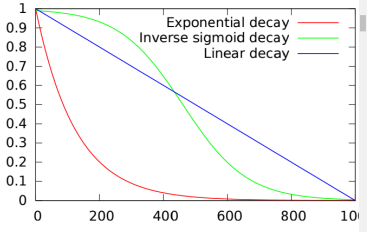
+ Модель быстро обучается обычно с хорошим качеством
– Модель плохо генерирует следующее слово для плохо сгенерированного предложения (таких случаев нет в обучении)

Beam search частично решает эту проблему!

Scheduled Sampling[1]

Выбираем с вероятностью ϵ_i истинное слово, иначе сгенерированное:





ϵ_i убывает с течением итераций по одному из трёх законов.

[1] Bengio et al (2015); Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks

Резюме по лекции

- Языковые модели позволяют оценить вероятность появления последовательности слов
- N-граммные языковые модели можно использовать для генерации признаков, простого автодополнения, для задач с шумным каналом (исправление опечаток)
- Нейросетевые языковые модели можно использовать для генерации текста и сложного автодополнения
- Для нейросетевых моделей выгодно использовать однонаправленные архитектуры
- Даже для хорошо обученной модели необходимо правильно подобрать гиперпараметры генерации текста