ITMO more than a UNIVERSITY

Современная теория информации

Лекция 6. Алгоритмы сжатия, используемые в архиваторах

Беляев Евгений Александрович
eabelyaev@itmo.ru

Содержание лекции

ИТМО

1. Метод "стопка книг".
2. Метод скользящего словаря LZ-77.
3. Алгоритм LZ-78 $\approx$ LZW;
4. Предсказание по частичному совпадению (PPM);
5. Преобразование Барроуза-Уиллера;
6. Нейросетевое моделирование источника и арифметическое кодирование

- Пусть $X = \{0, 1, ..., M-1\}$ – алфавит источника, на выходе которого наблюдается последовательность $x_1, x_2, ...$
- Предполагаем, что первой букве предшествуют все буквы алфавита.

1. Интервальное кодирование.
   - Вместо буквы x_n передаётся количество букв (длина интервала) r_n между текущим и предыдущим появлением данной буквы.
   - Пример. $abc|cabbbabbac \to 0, 3, 3, 0, 0, 3, 1, 0, 2, 9$.
2. "Стопка книг"[1][2].
   - Вместо буквы x_n передаётся количество ***различных*** букв d_n между текущим и предыдущим появлением данной буквы.
   - Пример. $abc|cabbbabbac \to 0, 2, 2, 0, 0, 1, 1, 0, 1, 2$.
   - Пример. $cba|cabbbabbac \to 2, 1, 2, 0, 0, 1, 1, 0, 1, 2$.

[1] Рябко Б.Я., Сжатие информации с помощью стопки книг // Проблемы передачи информации, 1980.

[2] J.Bentley et al., A locally adaptive data compression scheme, 22nd Alteron Conference on Communication Control and Computing, 1984.

- IF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CAN
- В ASCII пробел это 32, а буквам A,...,Z соответствуют числа 65...90.
- Результат: ...zyx...ZYX...BA..._...|74, 72, 35, 88, 73, 3, 72, 71, 80, 1, 81, 86, 6, 76, 4, 3, 6, 86, 3, 10, 10, 3, 3, 6, 87, 83, 9, 6, 6, 7, 3, 8, 81, 9, 9, 9, 9, 7, 2, 5, 3, 10, 8, 3, 10, 10, 3, 13, 6, 13.
- Применим к $\{d_i\}$ монотонный код:

$$mon(n) \to \underbrace{111..0}_{unar\,(\lfloor \log_2 n \rfloor + 1)} \underbrace{bin(n - 2^{\lfloor \log_2 n \rfloor})}_{\lfloor \log_2 n \rfloor \ бит}.$$

$$mon(21) \to \underbrace{11110}\,\underbrace{10101}.$$

- Получим 332 бита.

- Кодер LZ-77[3] хранит в памяти скользящий словарь, объёмом W (последние W букв).
- После кодирования символов $x_1, ..., x_{n-1}$ кодер ищет в словаре как можно более длинную последовательность (слово) из новых букв $x_n, ..., x_{n+l}$.
  - Если не нашлось, то декодеру передаётся $0|bin(x_n)$.
  - Если нашлось, то декодеру передаётся $1|bin(d)|vlc(l)$, где
    - d – расстояние от текущей позиции до начала найденного в словаре слова. Кодируется равномерным кодом длины $\lceil \log W \rceil$.
    - l – длина найденного слова. Кодируется неравномерным кодом, например, унарным.

[3] J. Ziv, A. Lempel, Compression of individual sequnces via variable rate coding, IEEE Transactions on Information Theory, 1977.

Метод скользящего словаря (LZ-77). Пример, когда $vlc(x) = mon(x)$.

ИТМО

Шаг	Флаг	$(x_1, ..., x_n)$	W	d	l	Код	Биты
0	0	I	0	-	0	0 bin(I)	9
1	0	F	1	-	0	0 bin(F)	9
2	0	_	2	-	0	0 bin(_)	9
3	0	W	3	-	0	0 bin(W)	9
4	0	E	4	-	0	0 bin(E)	9
5	1	_	5	2	1	1 010 0	5
6	0	C	6	-	0	0 bin(C)	9
7	0	A	7	-	0	0 bin(A)	9
8	0	N	8	-	0	0 bin(N)	9
9	1	N	9	0	1	1 0000 0	6
10	0	O	10	-	0	0 bin(O)	9
11	0	T	11	-	0	0 bin(T)	9
12	1	_	12	6	1	1 0110 0	6
13	0	D	13	-	0	0 bin(D)	9
14	1	O	14	3	1	1 0011 0	6
15	1	_	15	2	1	1 0010 0	6

Шаг	Флаг	$(x_1, ..., x_n)$	W	d	l	Код	Биты
16	1	A	16	8	1	1 01000 0	7
17	0	S	17	-	0	0 bin(S)	9
18	1	_WE_	18	15	4	1 01111 11000	11
19	1	W	22	2	1	1 00010 0	7
20	1	O	23	8	1	1 01000 0	7
21	0	U	24	-	0	0 bin(U)	9
22	0	L	25	-	0	0 bin(L)	9
23	1	D	26	12	1	1 01100 0	7
24	1	_WE_	27	8	4	1 01000 11000	11
25	1	S	31	13	1	1 01101 0	7
26	0	H	32	-	0	0 bin(H)	9
27	1	OULD_	33	9	5	1 001001 11001	12
28	1	DO_AS_WE_	38	24	9	1 011000 1110001	14
29	1	CAN	47	40	3	1 101000 101	10
Всего бит:							257

- Использовать код Хаффмана или арифметическое кодирование для $vlc(x)$.
- Ограничить максимальное I, $W = 2^w$ для более удобной организации памяти[4].
- Хранение словаря в виде древовидной структуры[5].

[4] LZFG

[5] E.Fiala, Data compression with finite windows. Communication of the ACM, 1989.

- На первом шаге кодер LZW[6] формирует словарь из *esc*-символа, который помещается в ячейку с номером $i = 0$, объём словаря $C \leftarrow 1$.
- На последующих шагах кодер ищет самое длинное слово в словаре, совпадающее с началом подлежащей кодированию последовательности $x_n, \ldots, x_{n+l}$.
  - Если не нашлось, то декодеру передаётся $\underbrace{00..0}_{\lceil \log(C-1) \rceil} | bin(x_n)$.
  - Если нашлось, то
    - декодеру передаётся $\underbrace{bin(i)}_{\lceil \log(C-1) \rceil},$ где i – номер найденного слова.
    - словарь дополняется новым словом, которое получается дописыванием к слову с номером i буквы x_{n+l+1}, $C \leftarrow C + 1$.
    - Декодер сможет добавить x_{n+l+1} только на следующем шаге. Поэтому кодер не использует последнее слово словаря.

[6] J. Ziv, A. Lempel, Compression of individual sequences via variable-rate coding, IEEE Transactions on Information Theory, 1980.

Шаг	Словарь	i	Код	Биты
0	esc	–	–	–
1	I	0	bin(I)	0+8=8
2	F	0	bin(F)	log(1)+8=8
3	_	0	0bin(_)	$\lceil \log(2) \rceil + 8 = 9$
4	W	0	00bin(W)	$\lceil \log(3) \rceil + 8 = 10$
5	E	0	00bin(E)	$\lceil \log(4) \rceil + 8 = 10$
6	_C	3	011	3
7	C	0	000bin(C)	3+8=11
8	A	0	000bin(A)	3+8=11
9	N	0	000bin(N)	3+8=11
10	NO	0	000bin(N)	4+8=12
11	O	0	0000bin(O)	4+8=12
12	T	0	0000bin(T)	4+8=12
13	_D	3	0011	4
14	D	0	0000bin(D)	4+8=12
15	O_	11	1011	4

Шаг	Словарь	i	Код	Биты
16	_A	3	0011	4
17	AS	8	1000	4
18	S	0	00000bin(S)	5+8
19	_W	3	00011	5
20	WE	4	00100	5
21	E_	5	00101	5
22	_WO	19	10011	5
23	OU	11	01011	5
24	U	0	00000bin(U)	5+8=13
25	L	0	00000bin(L)	5+8=13
26	D_	14	01110	5
27	_WE	19	10011	5
28	E_S	21	10101	5
29	SH	18	10010	5

Шаг	Словарь	i	Код	Биты
30	H	0	00000bin(H)	5+8=13
31	OUL	23	10111	5
32	LD	25	11001	5
33	D_D	26	11010	5
4	DO	14	001110	6
35	O_A	15	001111	6
36	AS_	17	010001	6
37	_WE_	27	011011	6
38	_CA	6	000110	6
39	AN	8	001000	6
40	N	9	001001	6
Всего бит:				291

- Избыточность LZ-77 с увеличением W определяется как:

$$\bar{R} \approx H_\infty(X) + \frac{\log \log W}{\log W}$$

- Избыточность LZW с увеличением C определяется как:

$$\bar{R} \approx H_\infty(X) + \frac{\log \log C}{\log C}$$

Контекстное адаптивное арифметическое кодирование. Основная идея.

ИТМО

- Если распределение вероятностей вида $\hat{p}(x_{t+1}|x_1, ..., x_t)$ возможно оценить с достаточной точностью, то арифметический кодер может использовать эти вероятности для достижения максимального сжатия, т.е., среднюю длину кодового слова, близкую к $H(X|X^\infty)$.
- Каждое условие соответствует источнику без памяти, к которому может быть применено универсальное кодирование.
- Избыточность адаптивного кодирования приближается к нулю с ростом длины кодируемой последовательности. Поэтому в каждый контекст должно попасть достаточное количество символов, т.е., контекстов не должно быть слишком много.
- Эффективность такого кодирования сильно зависит от метода оценки вероятностей.

Определение. Строка $\boldsymbol{s} = \boldsymbol{x}_{t-d+1}^{t}$ длины $d \leq D$, предшествующая x_{t+1}, является контекстом для x_{t+1}, если $\boldsymbol{s}$ уже появлялось в $\boldsymbol{x}_1^{t-1}$.

Пример. THE_CAT_IN_THE_CAR_ATE_THE_RAT

t	буква	контекст
16	A	THE_C
27	R	THE_

1. Выполняется поиск контекста $\boldsymbol{s} = \boldsymbol{x}_{t-d+1}^{t}$ наибольшей длины d, не превышающей D.
2. Для всех возможных значений символа x_{t+1} вычисляются оценки условных вероятностей символа при известном контексте $\boldsymbol{s}$.
3. Значение символа x_{t+1} кодируется арифметическим кодом в соответствии с вычисленной условной вероятностью.

- D – параметр алгоритима.
- Вероятность того, что символ $x_{t+1} = a$ после контекста $\boldsymbol{s} = \boldsymbol{x}_{t-d+1}^{t}$ может быть оценена как
$$\hat{p}_t(a|\boldsymbol{s}) = \frac{\tau_t(\boldsymbol{s}, a)}{\tau_t(\boldsymbol{s})}.$$
- Если буква для данного контекста не встречалась, т.е. $\tau_t(\boldsymbol{s}, a) = 0$, то арифметический кодер не сможет использовать $\hat{p}_t(a|\boldsymbol{s}) = 0$. Поэтому используется *esc*-символ.
- Если $\hat{p}_t(a|\boldsymbol{s}) = 0$, то передаётся *esc*-символ и контекст укорачивается на одну букву.

1: Находим наибольшее d такое, что $\hat{p}_t(\boldsymbol{x}_{t-d+1}^t) > 0$, $d \leq D$.
2: Выбираем контекст $\boldsymbol{s} \leftarrow \boldsymbol{x}_{t-d+1}^t$.
3: **while** $\hat{p}_t(x_{t+1}|\boldsymbol{s}) = 0$ **do**
4: Кодируем *esc* в соответствии с $\hat{p}_t(esc|\boldsymbol{s})$.
5: Уменьшаем длину контекста: $d \leftarrow d - 1, \boldsymbol{s} \leftarrow \boldsymbol{x}_{t-d+1}^t$.
6: **end while**
7: **if** $d > 0$ **then**
8: Кодируем x_{t+1} в соответствии с $\hat{p}_t(x|\boldsymbol{s})$.
9: **else**
10: **if** $\hat{p}_t(x) > 0$ **then**
11: Кодируем x_{t+1} в соответствии с $\hat{p}_t(x)$
12: **else**
13: Передаём *esc* и кодируем x_{t+1} в соответствии с равномерным распределением на не встречавшихся в $\boldsymbol{x}_1^t$ буквах.
14: **end if**
15: **end if**

Алгоритм A:

$$\hat{p}_t(a|\#) = \frac{\tau_t(\boldsymbol{a})}{t+1}; \qquad \hat{p}_t(esc|\#) = \frac{1}{t+1}, \tau_t(\boldsymbol{a}) > 0$$

$$\hat{p}_t(a|\boldsymbol{s}) = \frac{\tau_t(\boldsymbol{s}, a)}{\tau_t(\boldsymbol{s})+1}; \quad \hat{p}_t(esc|\boldsymbol{s}) = \frac{1}{\tau_t(\boldsymbol{s})+1}, \tau_t(\boldsymbol{s}, \boldsymbol{a}) > 0$$

Алгоритм D:

$$\hat{p}_t(a|\boldsymbol{s}) = \frac{\tau_t(\boldsymbol{s}, a) - 1/2}{\tau_t(\boldsymbol{s})}; \quad \hat{p}_t(esc|\boldsymbol{s}) = \frac{M_t(\boldsymbol{s})}{2\tau_t(\boldsymbol{s})}, \tau_t(\boldsymbol{s}, \boldsymbol{a}) > 0,$$

где $M_t(\boldsymbol{s})$ – число различных букв, появившихся в последовательности длины t вслед за контекстом $\boldsymbol{s}$.

- Пусть $\boldsymbol{s} = (s_d, ..., s_1)$ – контекст длины d.
- Если $\hat{p}(x_{t+1}|\boldsymbol{s}) = 0$, то передаётся *esc*-символ.
- После получения *esc* декодер всё еще не знает x_{t+1}, но он знает, какие символы не могут быть на этой позиции. Это символы, которые следовали за контекстом $\boldsymbol{s}$, когда он появлялся ранее. Это знание позволяет уточнять вероятность $\hat{p}(x_{t+1}|\boldsymbol{s}') = (s_{d-1}, ..., s_1)$.
- Список исключаемых символов растёт, если *esc* снова передаётся.
- Исключение некоторых символов увеличивает оценку вероятности для оставшихся символов, т.е., увеличивает сжатие.

Пример.

- Рассмотрим контекст $\boldsymbol{s}$ ="*кор*" за которым следует буква "*т*", причем "*корт*" ранее не встречалось. При этом предположим, что ранее после $\boldsymbol{s}$ встречались "*а*" и "*с*".
- Передаём *esc* и укорачиваем контекст до $\boldsymbol{s}$ ="*ор*".
- Тогда τ'_t(ор)=τ_t(ор)-τ_t(ора)-τ_t(орс).

Алгоритм A:

$$\begin{aligned} \hat{p}_t(a|\boldsymbol{s}) &= \frac{\tau_t(\boldsymbol{s}, a)}{\tau_t(\boldsymbol{s}) - \tau_t^*(\boldsymbol{s}) + 1}; \\ \hat{p}_t(esc|\boldsymbol{s}) &= \frac{1}{\tau_t(\boldsymbol{s}) - \tau_t^*(\boldsymbol{s}) + 1}, \tau_t(\boldsymbol{s}, \boldsymbol{a}) > 0 \end{aligned}$$

где $\tau_t(\boldsymbol{s})$ и $\tau_t(\boldsymbol{s}, a)$ – число строк $\boldsymbol{s}$ и $(\boldsymbol{s}, a)$, соответственно, которые содержатся в $\boldsymbol{x}_1^t$, $\tau_t^*(\boldsymbol{s})$ – число букв, которые не могут следовать за $\boldsymbol{s}$.

Если $\tau_t(\boldsymbol{s}, a) = 0$, но $\tau_t(\boldsymbol{s}', a) > 0$, то

$$\hat{p}_t(a|\boldsymbol{s}) = \frac{1}{(\tau_t(\boldsymbol{s}) - \tau_t^*(\boldsymbol{s}) + 1)} \hat{p}_t(a|\boldsymbol{s}')$$

где $\boldsymbol{s}'$ на одну букву короче $\boldsymbol{s}$.

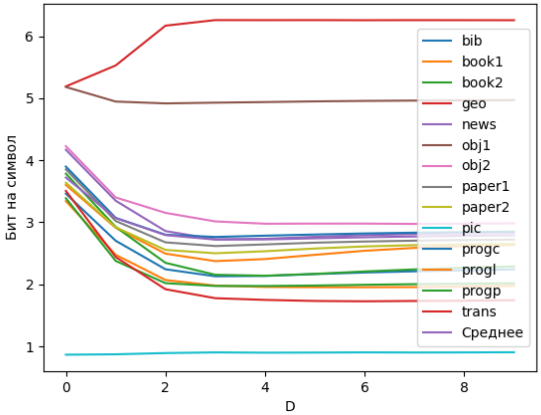
Шаг	Буква	Контекст	$\tau_t(\boldsymbol{s})$	$p_t(esc\|\boldsymbol{s})$	$p_t(x\|\boldsymbol{s})$
1	I	#	0	1	1/256
2	F	#	1	1/2	1/255
3	_	#	2	1/3	1/254
4	W	#	3	1/4	1/253
5	E	#	4	1/5	1/252
6	_	#	5		1/6
7	C	_	1,6	1/2×1/6’	1/251
8	A	#	7	1/8	1/250
9	N	#	8	1/9	1/249
10	N	#	9		1/10
11	O	N	1,10	1/2×1/9’	1/248
12	T	#	11	1/12	1/247
13	_	#	12		2/13
14	D	_	2,13	1/3×1/12’	1/246
15	O	#	14		1/15
16	_	O	1,15	1/2	3/15’
17	A	_	3,16	1/4	1/14’
18	S	A	1,17	1/2×1/16’	1/245
19	_	#	18		4/19

Шаг	Буква	Контекст	$\tau_t(s)$	$p_t(esc\|s)$	$p_t(x\|s)$
20	W	_	4		1/5
21	E	_W	1		1/2
22	_	_WE	1		1/2
23	W	_WE_	1,1,1,5	1/2×1'×1'	2/5'
24	O	_W	2,2,23	1/3×1'	2/22'
25	U	O	2,24	1/3×1/18'	1/244
26	L	#	25	1/26	1/243
27	D	#	26		1/27
28	_	D	1,27	1/2	6/25'
29	W	_	6		3/7
30	E	_W	3		2/4
31	_	_WE	2		2/3
32	S	_WE_	2,2,2,7,31	1/3×1'×1'×1/3'	1/23'
33	H	S	1,32	1/2×1/25'	1/242
34	O	#	33		3/34
35	U	O	3		1/4
36	L	OU	1		1/2
37	D	OUL	1		1/2
38	_	OULD	1		1/2
39	D	OULD_	1,1,1,1,8	1/2×1'×1'×1'	1/4'

Шаг	Буква	Контекст	$\tau_t(s)$ [7]	$p_t(esc\|s)$	$p_t(x\|s)$
40	O	_D	1		1/2
41	_	_DO	1		1/2
42	A	_DO_	1		1/2
43	S	_DO_A	1		1/2
44	_	DO_AS	1		1/2
45	W	O_AS_	1		1/2
46	E	_AS_W	1		1/2
47	_	AS_WE	1		1/2
48	C	S_WE_	1,3	1/2	1/3'
49	A	_WE_C	1		1/2
50	N	WE_CA	1		1/2

Всего бит = 250 бит (232 бит для PPMD).

[7] $\tau_t(s)$ – число появлений контекста на предыдущих шагах



- На практике[8] достаточно $1 \leq D \leq 5$.

[8] Данные получены Сергеем Козелько, 2021.

Пусть $\boldsymbol{x}$ – последовательность на выходе источника[9].

1. Все циклические сдвиги последовательности $\boldsymbol{x}$ сортируются в алфавитном порядке и формируют таблицу.
2. Результатом такого преобразования являются:
   - Последний столбец таблицы.
   - Номер строки, которая соответствует исходной последовательности.

[9] M.Burrows, D. Wheeler, A block-sorting lossless data compression algorithm, Report 124, Digital Systems Research Center, 1994

Прямое преобразование Барроуза-Уиллера. Пример.

0	A	NIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_	C
1	A	NNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_	C
2	A	S_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO	_
3	A	S_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO	_
4	C	ANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE	_
5	C	ANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE	_
6	D	O_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD	_
7	D	O_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT	_
8	D	_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOU	L
9	D	_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOU	L
10	E	_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_	W
11	E	_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_	W
12	E	_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_	W
13	E	_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_	W
14	F	WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CAN	I*
15	H	OULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_	S
16	I	F_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CA	N
17	L	D_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHO	U
18	L	D_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WO	U
19	N	IF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_C	A
20	N	NOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_C	A
21	N	OT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CA	N
22	O	T_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CAN	N
23	O	ULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_S	H
24	O	ULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_	W

25	O	_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_	D
26	O	_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_	D
27	S	HOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE	_
28	S	_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_	A
29	S	_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_	A
30	T	_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANN	O
31	U	LD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SH	O
32	U	LD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_W	O
33	W	E_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS	_
34	W	E_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF	_
35	W	E_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD	_
36	W	E_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS	_
37	W	OULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE	_
38	_	AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_D	O
38	_	AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_D	O
40	_	CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_W	E
41	_	CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_W	E
42	_	DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOUL	D
43	_	DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNO	T
44	_	SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_W	E
45	_	WE_CANIF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_A	S
46	_	WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CANI	F
47	_	WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_WE_WOUL	D
48	_	WE_WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_A	S
49	_	WOULD_WE_SHOULD_DO_AS_WE_CANIF_WE_CANNOT_DO_AS_W	E

Результат преобразования:

- Последний столбец:
  CC_____LLWWWWISNUUAANNHWDD_AAOOO_____OOEEDTESFDSE
- Номер строки = 16.

Свойства преобразования:

- В результате преобразования символы сгруппировались так, что они стали содержать серии одинаковых символов.
- Это свойство может быть использовано для увеличения степени сжатия.
- Необходимо передавать номер строки.

- Находим первый столбец таблицы путём сортировки последнего столбца.
- Восстанавливаем исходную строку символ за символом, с учетом того, что символу, находящемуся в последнем столбце предшествует символ в первом столбце (из-за лексикографического упорядочивания).

Обратное преобразование Барроуза-Уиллера. Пример.

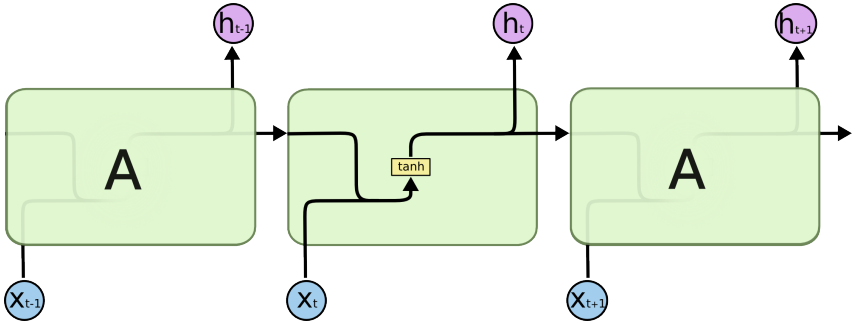
t	First	Last	t	First	Last	t	First	Last
0	A	C	17	L	U	34	W	_
1	A	C	18	L	U	35	W	_
2	A	_	19	N	A	36	W	_
3	A	_	20	N	A	37	W	_
4	C	_	21	N	N	38	_	O
5	C	_	22	O	N	39	_	O
6	D	_	23	O	H	40	_	E
7	D	_	24	O	W	41	_	E
8	D	L	25	O	D	42	_	D
9	D	L	26	O	D	43	_	T
10	E	W	27	S	_	44	_	E
11	E	W	28	S	A	45	_	S
12	E	W	29	S	A	46	_	F
13	E	W	30	T	O	47	_	D
14	F	I*	31	U	O	48	_	S
15	H	S	32	U	O	49	_	E
16	I	N	33	W	_			

- (N,16) $\rightarrow$ I, (I,14) $\rightarrow$ F, (F,46) $\rightarrow$ _. Пробелов много. Что делать?
- Так как на строке 46 пробел является 9-м по счёту, ищем 9-ю строку с пробелом в последнем символе, получаем (34,W),...

- CC_____LLWWWWISNUUAANNHWDD_AAOOO_____OOEEDTESFDSE
- Будем писать *esc* каждый раз, когда появляется новый символ и число различных букв между предыдущим и текущим появлением буквы.
- Получим: *esc* 0 *esc* 0 0 0 0 0 *esc* 0 *esc* 0 0 0 *esc esc esc esc* 0 *esc* 0 2 0 *esc* 6 *esc* 0 9 5 0 *esc* 0 0 2 0 0 0 0 1 0 *esc* 0 4 *esc* 2 10 *esc* 4 2 3
- Применяем нумерационное кодирование:
  - Кодирование композиции: 35 бит.
  - Кодирование последовательности: 94 бита.
- Передача символов, соответствующих *esc*: $8 \times 15 = 120$ бит.
- Передача индекса исходной строки в преобразовании: 6 бит.

Итого: 255 бит.

С помощью RNN можно обрабатывать последовательности данных, например, предсказывать следующий символ по предыдущим. В RNN чтобы хранить информацию о предыдущих токенах вводится понятие скрытого состояния (hidden state).



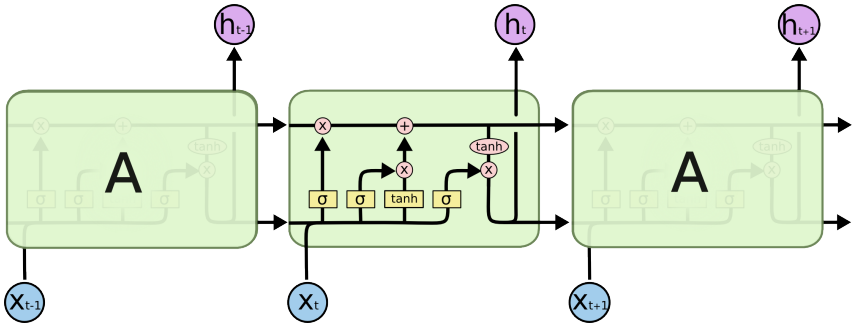
На каждом шаге в сеть подаются данные (например, эмбеддинг токена), при этом происходит обновление скрытого состояния:

$$h_t = \tanh(h_{t-1} W_1 + x_t W_2)$$

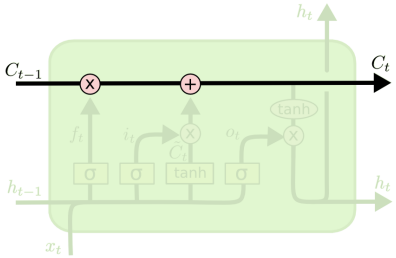
Далее по скрытому состоянию предсказывается выходной сигнал:

$$y_t = h_t W_3$$

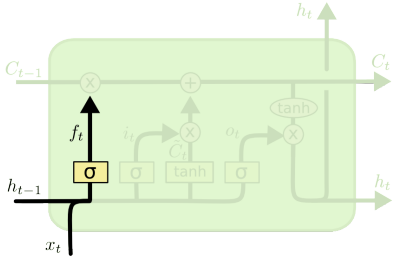
В RNN ошибки распространяются на много шагов назад посредством backpropagation. В этом процессе происходит многократное перемножение градиентов, и если собственные значения матрицы весов на каждом шаге превышают 1, величина градиентов будет экспоненциально увеличиваться (“взрываться”) по мере возрастания количества временных шагов.



Long short term memory (LSTM) архитектура частично решает проблемы классических RNN - взрывающиеся и затухающие градиенты. Специальные механизмы - вентили (gates), позволяют контролировать количество проходящей через них информации.

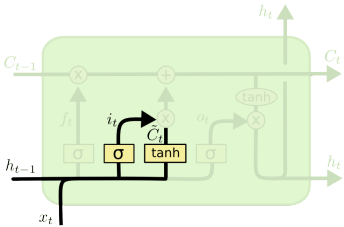


Кроме этого, в LSTM появляется понятие состояния блока (cell state). Он позволяет информации легко проходить через множество блоков.



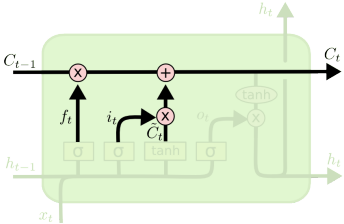
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Forget gate позволяет на основе предыдущего скрытого состояния h_{t-1} и нового входа x_t определить, какую долю информации из состояния предыдущего блока c_{t-1} стоит пропустить дальше, а какую забыть.

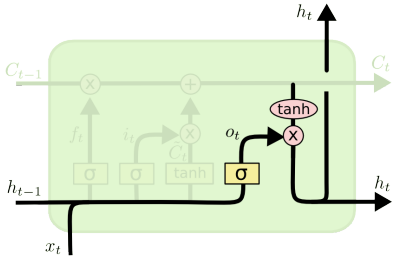


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Input gate позволяет решить какую информацию добавить в состояние блока от h_{t-1} и x_t. Таким образом, состояние блока будет обновлено:

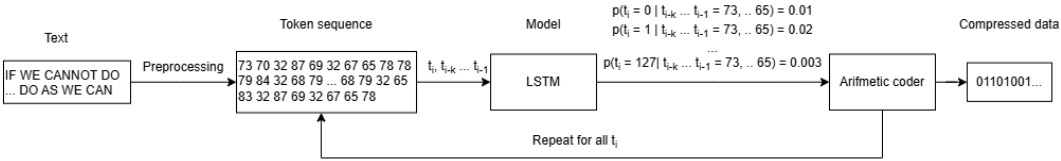


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Output gate отвечает на вопрос о том, сколько информации из c_t следует отдавать на выход из LSTM-блока.



1. LSTM используется для вычисления вектора вероятностей следующего символа на основе предыдущих.
2. Этот вектор используется для кодирования текущего символа при помощи арифметического кодера.
3. Веса LSTM модели обновляются (обучаются) на основе уже обработанных символов (тоже самое происходит на стороне декодера).
4. В итоговый файл, кроме сжатых данных, в начало записывается длина оригинальной последовательности и используемый алфавит.
5. LSTM предсказывает всё лучше по мере роста числа обработанных символов.

Алгоритм	Затраты (бит)
Без сжатия	400
Двухпроходное кодирование, код Хаффмана	302
Нумерационное кодирование	283
Арифметическое кодирование, алгоритм A	293
Арифметическое кодирование, алгоритм D	283
LZ77	280
LZW	291
PPMA	250
PPMD	232
Преобразование Барроуза-Уиллера и метод "стопка книг"	255
Преобразование Барроуза-Уиллера и кодирование расстояний	235
Prediction-based modeling[10]	400
Prediction-based modeling без учета header	232

[10] A single-layer LSTM with 512 hidden units is used

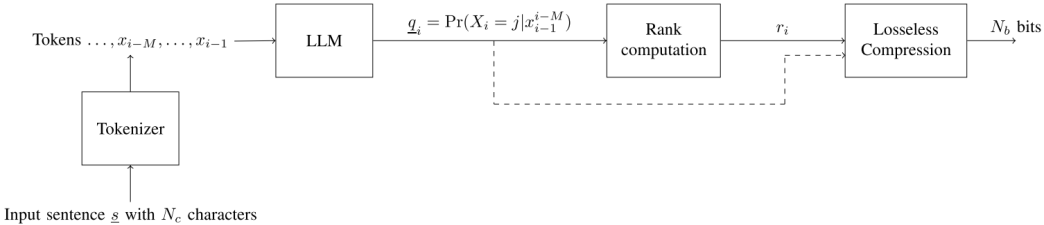
Size (bytes)	File name	Description
111,261	BIB[11]	ASCII text in UNIX "refer"format – 725 bibliographic references.
768,771	BOOK1	unformatted ASCII text – Thomas Hardy: Far from the Madding Crowd.
610,856	BOOK2	ASCII text in UNIX "troff"format – Witten: Principles of Computer Speech.
102,400	GEO	32 bit numbers in IBM floating point format – seismic data.
377,109	NEWS	ASCII text – USENET batch file on a variety of topics.
21,504	OBJ1	VAX executable program – compilation of PROGP.
246,814	OBJ2	Macintosh executable program – "Knowledge Support System".
53,161	PAPER1	UNIX "troff"format – Witten, Neal, Cleary: Arithmetic Coding for Data Compression.
82,199	PAPER2	UNIX "troff"format – Witten: Computer (in)security.
513,216	PIC	1728×2376 bitmap image (MSB first): text in French and line diagrams.
39,611	PROGC	Source code in C – UNIX compress v4.0.
71,646	PROGL	Source code in Lisp – system software.
49,379	PROGP	Source code in Pascal – program to evaluate PPM compression.
93,695	TRANS	ASCII and control characters – transcript of a terminal session.

[11] http://www.data-compression.info/Corpora/CalgaryCorpus/

Chunk	Compressor	Raw Compression Rate (%)				Adjusted Compression Rate (%)			
		enwik9	ImageNet	LibriSpeech	Random	enwik9	ImageNet	LibriSpeech	Random
∞	gzip	32.3	70.7	36.4	100.0	32.3	70.7	36.4	100.0
	LZMA2	23.0	57.9	29.9	100.0	23.0	**57.9**	29.9	100.0
	PNG	42.9	58.5	32.2	100.0	42.9	58.5	32.2	100.0
	FLAC	89.5	61.9	30.9	107.8	89.5	61.9	30.9	107.8
2048	gzip	48.1	68.6	38.5	100.1	48.1	68.6	38.5	100.1
	LZMA2	50.0	62.4	38.2	100.0	50.0	62.4	38.2	100.0
	PNG	80.6	61.7	37.6	103.2	80.6	61.7	37.6	103.2
	FLAC	88.9	60.9	30.3	107.2	88.9	60.9	**30.3**	107.2
	Transformer 200K	30.9	194.0	146.6	195.5	30.9	194.0	146.6	195.5
	Transformer 800K	21.7	185.1	131.1	200.1	21.9	185.3	131.3	200.3
	Transformer 3.2M	17.0	215.8	228.2	224.0	**17.7**	216.5	228.9	224.7
	Llama 2 (7B)	8.9	53.4	23.1	103.2	1408.9	1453.4	1423.1	1503.2
	Chinchilla 1B	11.3	62.2	24.9	108.8	211.3	262.2	224.9	308.8
	Chinchilla 7B	10.2	54.7	23.6	101.6	1410.2	1454.7	1423.6	1501.6
	Chinchilla 70B	**8.3**	**48.0**	**21.0**	100.8	14008.3	14048.0	14021.0	14100.8

12

[12] G. Deletang et al., Language Modeling Is Compression, International Conference on Learning Representations, 2024.



1. Входной текст представляется как выход источника с алфавитом размером в 32000 слов (токенов) переменной длины.
2. Вероятность каждого следующего слова $\boldsymbol{x}_i$ длиной в b_i символов оценивается условная вероятность $p(\boldsymbol{x}_i | \boldsymbol{x}_{i-M}, ..., \boldsymbol{x}_{i-1})$ при помощью LLaMA модели, которая учитывает предыдущие M слов.
3. Энтропия оценивается через оценку условной собственной информации:

$$H(\boldsymbol{X}) \leq -\frac{\lim\limits_{N \to 0} \frac{1}{N} \sum\limits_{i}^{N} \log p(\boldsymbol{x}_i | \boldsymbol{x}_{i-M}, ..., \boldsymbol{x}_{i-1})}{E(b_i)}$$

COMPRESSION PERFORMANCE OF THE LLM ON THE TEXT8 DATASET, AS A FUNCTION OF ITS MEMORY (M)

M	N_c	N_t	H_{ub} (bpc)	$\rho_{\text{LLaMA+zlib}}$ file size (bits)	$\rho_{\text{LLaMA+TbyT}}$ (bpc)	$\rho_{\text{LLaMA+AC}}$ (bpc)
31	4,568,855	1,000,000	0.9139	1.3159	1.0425	0.9145
127	4,568,855	1,000,000	0.7511	1.1303	0.8847	0.752
255	4,568,855	1,000,000	0.7242	1.0985	0.859	0.725
511	4,568,855	1,000,000	0.7093	1.0812	0.845	0.7101

Таким образом, энтопия английского языка может быть оценена как 0.71 бит на символ[13].

[13] C. Valmeekam et al., LLMZip: Lossless Text Compression using Large Language Models, 2023

Архиватор	Настройки	Сжатие[14], байт
Uncompressed		3,141,622
compress		1,272,772
Info-ZIP	2.32 -9	1,020,781
gzip 1.3.5	-9	1,017,624
bzip2 1.0.3	-9	828,347
7-zip 9.12b		848,687
ppmd Jr1	-m256 -o16	740,737
ppmonstr J		675,485
ZPAQ v7.15	-method 5	659,709

[14] Все файлы сжимаются по-отдельности

№	Алгоритм	Параметры	Баллы
1	Код Хаффмана	Двухпроходный	10
2	Код Хаффмана	Двухпроходный, блоками по 2 символа	15
3	Код Хаффмана	Двухпроходный, с учетом предыдущего символа	15
4	Нумерационное кодирование	На основе арифметического кодирования	20
5	Арифметическое кодирование	Оценка вероятности A	15
6	Арифметическое кодирование	Оценка вероятности D	15
7	"Стопка книг"	Кодирование монотонным кодом	10
8	LZ77	$vlc(x)$ – монотонный код	10
9	LZ77	$vlc(x)$ – код Хаффмана	15
10	LZW		10
11	PPMA		20
12	PPMD		20
13	Барроуза-Уиллера	Сжатие методом "стопки книг"	20

Требования к программе и отчету

Требования к программе:

1. Написать кодер и декодер, которые работают как отдельные программы и запускаются с командной строки:
2. В качестве параметра на вход кодера подаётся имя сжимаемого файла. На выходе программа выдаёт файл со сжатыми данными: *encoder infile zipfile*.
3. На вход декодера подаётся файл со сжатыми данными и имя декодированного файла: *decoder zipfile decfile*. На выходе декодер выдаёт файл *decfile*, который идентичен исходному *infile* (бит в бит).
4. Алгоритмы, относящиеся непосредственно к кодированию и декодированию должны быть реализованы без использования сторонних библиотек.

На почту *eabelyaev@itmo.ru* прислать:

1. Отчет в pdf формате, который включает в себя:
   - ФИО студента, номер группы и задания.
   - Описание алгоритма кодирования и декодирования.
   - Для каждого из 14 файлов Calgary corpus вычислить оценки $H(X)$, $H(X|X)$, $H(X|XX)$, а также указать значения средних затрат на символ для реализованного алгоритма (в битах) и суммарный размер сжатого файла в байтах. Данные представить в виде таблицы.
   - Суммарное значение сжатого размера всех файлов в байтах.
2. Исходный код кодера и декодера.
3. Исполняемые программы кодера и декодера, готовыми к запуску на Windows 10.

ITMO more than a UNIVERSITY

Спасибо за внимание!