Embeddings and RNN

Щербаков Максим

16 октября 2025

Обработка естественного языка

Обработка естественного языка (Natural Language Processing, NLP) включает задачи, связанные с текстами на естественном языке. Это задачи понимания языка и генерации текстов, а некоторые из них охватывают оба аспекта.

Векторные представления слов

- BOW
- Tf-idf
- LSA

Эмбеддинги слов

- Контекстные эмбеддинги
- Word2Vec
- GloVe, FastText

One-hot Encoding

Создадим словарь фиксированного размера с примерами:

Слово	**Векторное представление**
cat	[1, 0, 0, …, 0]
dog	[0, 1, 0, …, 0]
…	…
mother	[0, 0, 0, …, 1, …, 0] (i-я координата)

Например, если размер словаря равен 50 000, каждая позиция соответствует одному слову.

Недостатки one-hot encoding

- Векторы слов не отражают смысл слова, нельзя измерить «похожесть» двух слов по значению.
- Векторы разрежены и требуют большого объема памяти.
- Размер словаря ограничен, слова вне словаря не обрабатываются.
- При изменении размера словаря все векторы нужно пересчитывать заново.

Bag Of Words

Словарь:

Индекс	**Слово**
1	a
2	and
14	are
145	cat
257	dog
678	is
1537	sleeping

Предложение	**Вектор BoW**
1. a cat and a dog are sleeping	[2, 1, 0, …, 1, 0, …, 0, 1, 0, …, 1, 0, …, 0, …, 1, …, 0]
2. a dog is walking	[1, 0, 0, …, 0, 0, …, 0, 0, 0, …, 1, 0, …, 1, …, 0, …, 0]
indexes	1 2 14 145 257 678 1537

Наследуются все недостатки one-hot encoding:

- Векторы предложений не очень хорошо отражают смысл предложения. Порядок слов не учитывается;
- Векторы довольно разрежены, требуют много лишней памяти;
- Фиксированный размер словаря. Слова, не попавшие в словарь, не могут быть обработаны.
- При изменении размера словаря нужно пересчитывать векторы заново.

Еще один недостаток BoW — различные слова могут иметь разную важность для текста.

TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) — статистическая мера важности слова в документе с учётом частоты его использования в данном документе и обратной частоты встречаемости этого слова во всех документах корпуса.

Частота термина (TF)

$$\text{TF}(t, d) = \frac{\text{число вхождений } t \text{ в } d}{\text{общее число слов в } d}$$

Обратная частота документа (IDF)

$$\text{IDF}(t) = \log\left(\frac{N}{n_t}\right)$$

где N — общее число документов, n_t — количество документов, содержащих термин t.

Формула TF-IDF

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

Для слова «a» в двух предложениях:

1. a cat and a dog are sleeping
2. a dog is walking

Общее число документов $N = 2$.

Шаг 1: Рассчитаем TF (частоту термина)

- В предложении 1: слово «a» встречается 2 раза из 7 слов

$$\text{TF}(a, d_1) = \frac{2}{7} \approx 0.286$$

- В предложении 2: слово «a» встречается 1 раз из 4 слов

$$\text{TF}(a, d_2) = \frac{1}{4} = 0.25$$

Шаг 2: Рассчитаем IDF (обратную частоту по документам) Слово «a» есть в обоих документах, значит количество документов с этим словом $n_a = 2$.

$$\text{IDF}(a) = \log \frac{N}{n_a} = \log \frac{2}{2} = \log 1 = 0$$

Шаг 3: Рассчитаем TF-IDF

$$\text{TF} - \text{IDF}(a, d_1) = \text{TF}(a, d_1) \times \text{IDF}(a) = 0.286 \times 0 = 0$$

$$\text{TF} - \text{IDF}(a, d_2) = \text{TF}(a, d_2) \times \text{IDF}(a) = 0.25 \times 0 = 0$$

Для слова «walking»:

$$\text{TF}(walking, d_1) = 0$$

$$\text{TF}(walking, d_2) = \frac{1}{4} = 0.25$$

$$\text{IDF}(walking) = \log \frac{N}{n_{walking}} = \log \frac{2}{1} = \log 2 \approx 0.693$$

$$\text{TF-IDF}(walking, d_1) = 0 \times 0.693 = 0$$

$$\text{TF-IDF}(walking, d_2) = 0.25 \times 0.693 \approx 0.173$$

Плюсы

- Векторы имеют больший смысл, чем при BoW;
- Возможность решать такие задачи, как ранжирование документов и выделение ключевых слов;

Недостатки

- Векторы довольно разрежены;
- Фиксированный размер словаря.
- При изменении коллекции документов векторы нужно пересчитывать.

LSA (Латентный семантический анализ)

Идея: использовать сингулярное разложение матрицы документы-слова. Заметим что при ”срезе” диагональной матрицы точность снижается, но смысл сохраняется.

Плюсы

- Векторы имеют смысл;
- Возможность уменьшать размер эмбеддингов без существенной потери качества.

Латентный семантический анализ

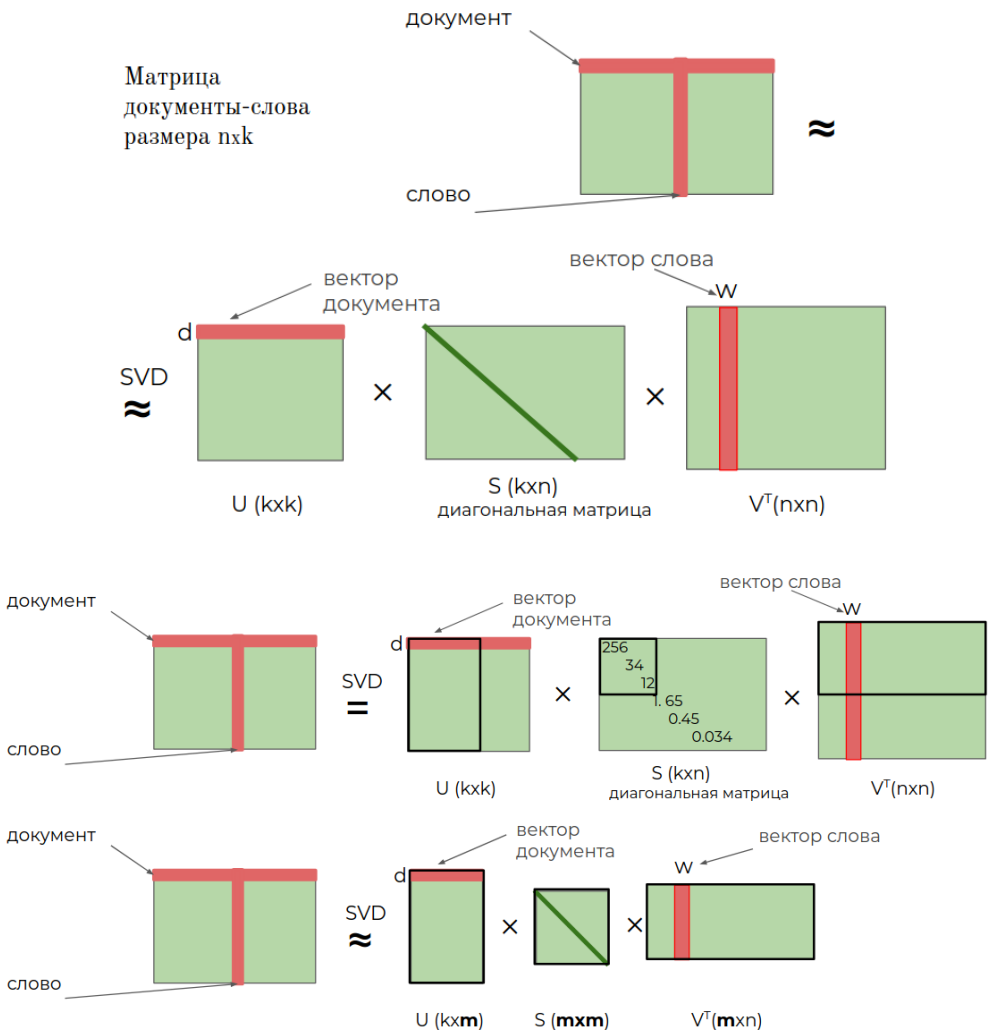


Figure 1: Иллюстрация LSA

Недостатки

- Большая вычислительная сложность при большом количестве документов;
- Фиксированный размер словаря;
- При изменении коллекции документов векторы нужно пересчитывать;
- Вероятностная модель метода не соответствует реальности.

Контекстные эмбеддинги

- Контекстные эмбеддинги строятся с учётом контекста, в котором встречается слово.
- Смысл слова определяется окружающими словами, а не только самим словом.
- Пример: рассмотрим предложения с пропусками и кандидаты на их место:
    1. Маша ездит на _____

2. Колесо _____ было проколото
3. У _____ красивая белая рама

- Разные кандидаты (велосипед, мотоцикл, машина, лошадь) подходят или не подходят в зависимости от контекста.

Идея контекстных векторов слов

- Контекстные векторы слов — это строки матрицы, где каждая строка соответствует слову, а значение зависит от контекста.
- Таким образом, векторы отражают смысл слова в конкретном контексте.
- Векторы можно сравнивать по косинусному расстоянию или среднеквадратичной ошибке (MSE) для оценки смысловой близости.

Преимущества и недостатки

Плюсы

- Векторы начинают отражать смысл слов, а не только их частотность.
- Можно сравнивать слова на схожесть.

Минусы

- Векторы всё ещё довольно разрежены и требуют много памяти.
- Размер словаря ограничен, слова вне словаря не обрабатываются.
- При изменении словаря нужно пересчитывать векторы заново.
- Эмбеддинги для редких слов менее информативны.

Word2Vec

Мы будем учить нейросеть по слову предсказывать слова, которые могут находиться в контексте (стоять вокруг этого слова).

Наш датасет — набор текстов. Мы будем идти по датасету скользящим окном размера 5, и в каждом положении окна по центральному слову учить нейросеть предсказывать слова, находящиеся в текущем окне.

На векторах word2vec можно проводить векторную арифметику:

$$v(\text{king}) - v(\text{man}) + v(\text{woman}) \approx v(\text{queen})$$

Преимущества

- Векторы отражают смысл слов;
- Размерность векторов не зависит от размера словаря;
- При добавлении документов векторы можно дообучить.

Недостатки

- Фиксированный размер словаря. При изменении размера словаря документов векторы нужно пересчитывать;
- Для редких слов эмбеддинги получаются неоптимальными;
- Слова, имеющие один корень, обрабатываются нейросетью по-разному. eat, eater, eating

Skip-Gram — предсказание слов контекста по центральному слову
CBOW — предсказание центрального слова по словам контекста

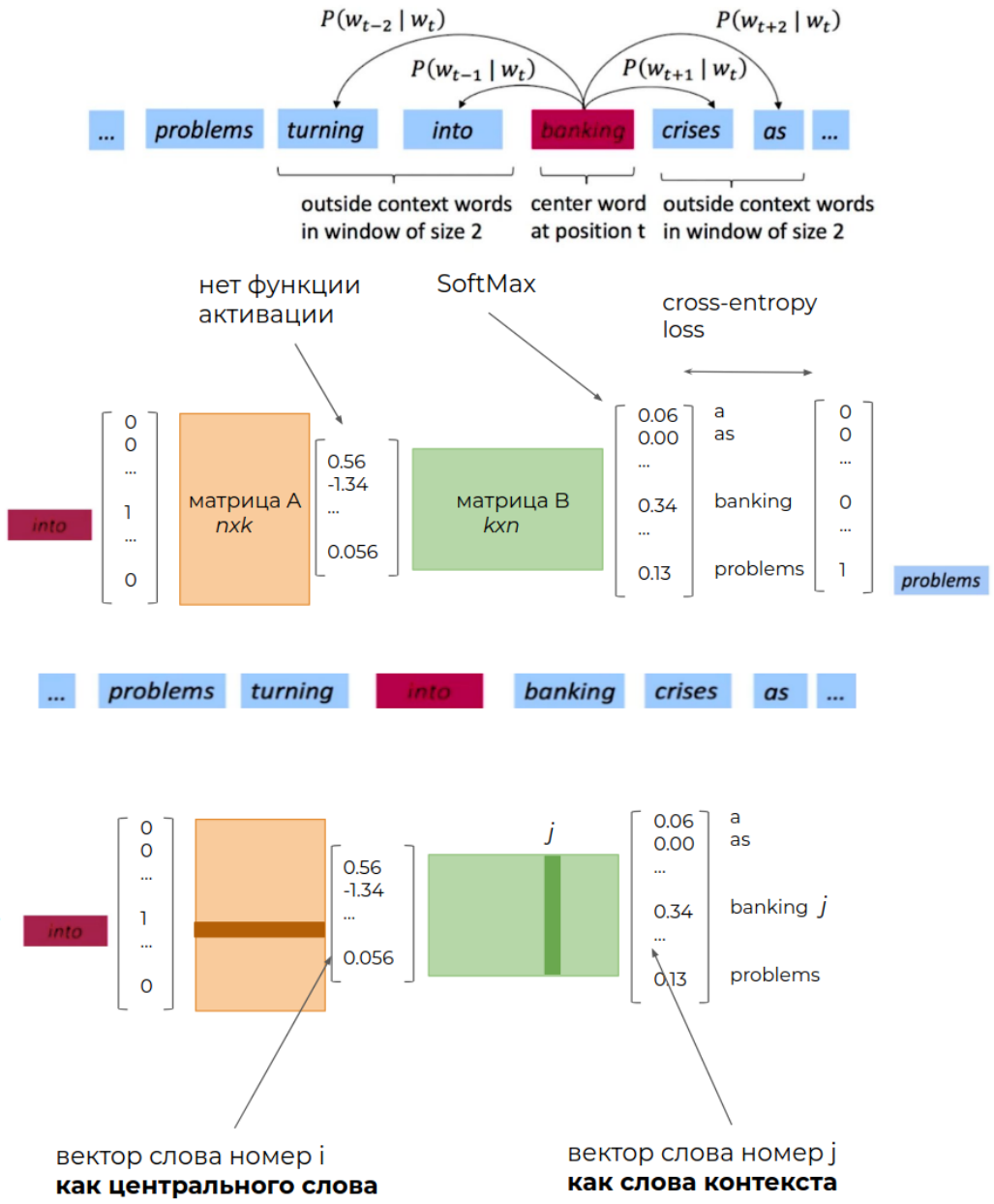


Figure 2: Иллюстрация Word2Vec

FastText

Идея — будем строить векторы для частей слов, а не для целых слов.

- Делим слова на n-граммы по буквам: apple = <ap, ppl, ple, le>
- Учим векторы для n-грамм;
- Вектор слова получаем как сумму векторов его n-грамм.

Плюсы

- Можно получить более адекватные эмбеддинги для редких и неизвестных слов;

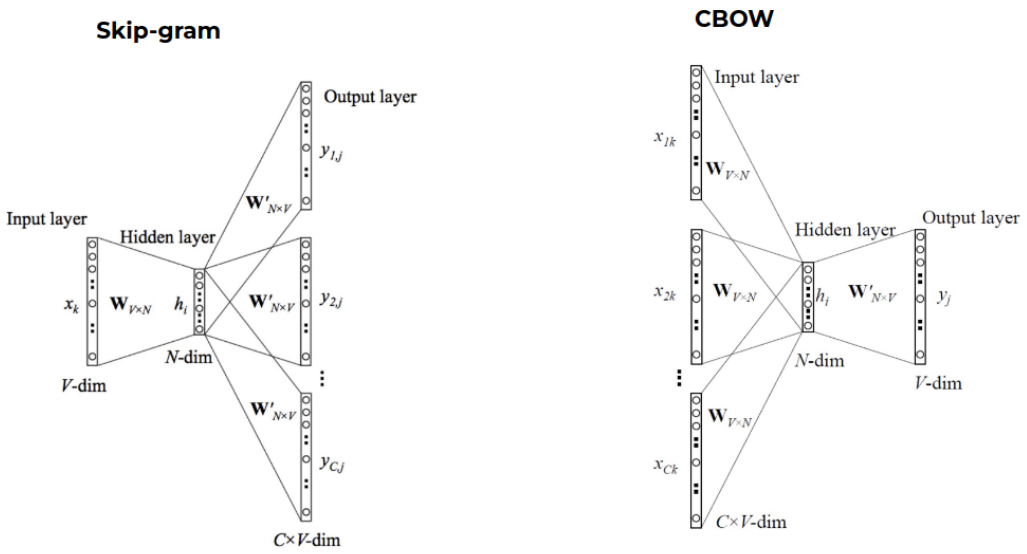


Figure 3: Иллюстрация Skip-Gram и CBOW

Недостатки

- n-грамм может быть очень много. Требуется больше вычислительных ресурсов.

GloVe (Global Vectors)

GloVe использует статистическую информацию о частоте встречаемости слов и фраз в тексте, чтобы улучшить обучение эмбеддингов редких слов. Подробнее можно почитать тут.

Эмбеддинги в общем смысле

Эмбеддинг — векторное представление объекта, которое отражает информацию об объекте. Выходы слоев моделей, обученных под какую-либо задачу, тоже можно считать эмбеддингами.

Иными словами, при обучении нейронной сети выход последнего слоя - тоже эмбеддинг.

RNN, GRU и LSTM

Отдельной темой доклада могло бы стать обучение RNN, и почему функцией активации должен быть Tanh, если коротко, то для длинных временных последовательность при вычислении градента получаем премножение большого количества производных функции активации, что ведеть либо к затуханию, либо к взрыву градиентов.

1 Слой рекуррентной нейросети

Рекуррентные нейронные сети (РНС) — это класс нейронных сетей, где связи между узлами образуют направленный граф вдоль временной последовательности.

1.1 Обновление вектора скрытого состояния и вычисление выхода слоя

На каждом временном шаге t рекуррентный слой обновляет свой вектор скрытого состояния h^t на основе предыдущего скрытого состояния h^{t-1} и входного вектора x^t. Выход слоя y затем вычисляется на основе обновленного скрытого состояния.

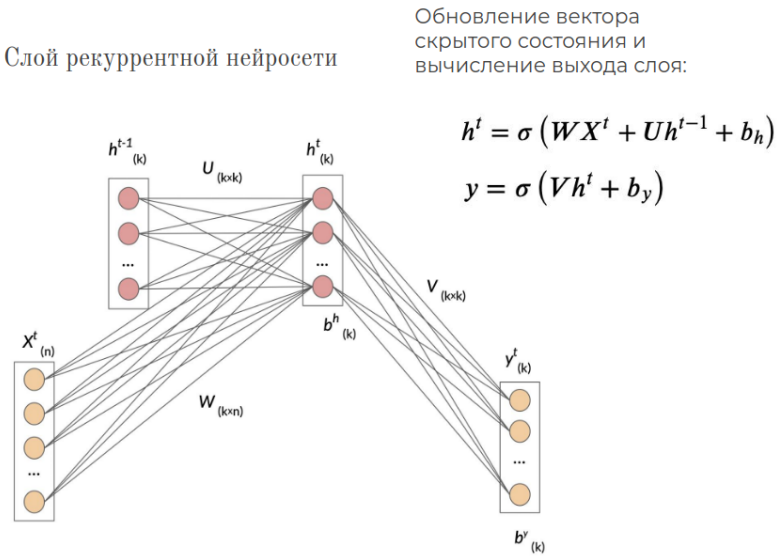


Figure 4: Схема работы рекуррентного слоя.

Математически это можно описать следующими формулами:

$$h^t = \sigma(WX^t + Uh^{t-1} + b_h)$$
$$y = \sigma(Vh^t + b_y)$$

где W, U, V — весовые матрицы, b_h, b_y — векторы смещения, а σ — функция активации.

2 LSTM (Long Short-Term Memory)

Долгая краткосрочная память (LSTM) — это особый тип архитектуры РНС, разработанный для лучшего запоминания и использования долгосрочных зависимостей в данных.

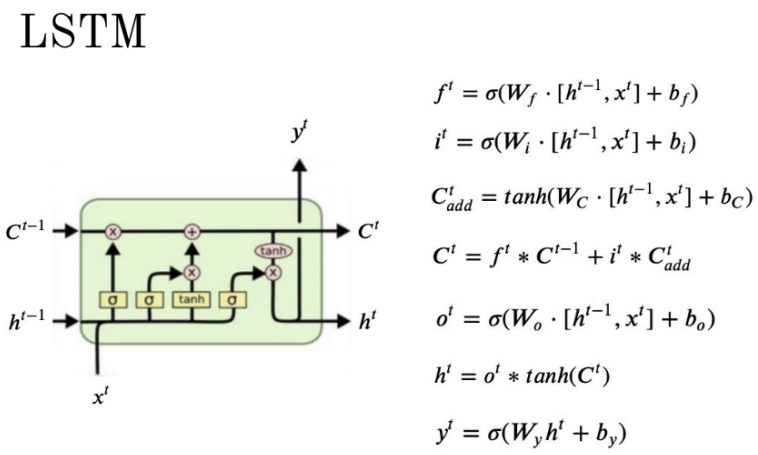


Figure 5: Архитектура ячейки LSTM.

2.1 Ворота забывания (“forget gate”)

Этот вентиль решает, какую информацию следует отбросить из состояния ячейки. Он смотрит на h^{t-1} и x^t и выводит число от 0 до 1 для каждого числа в состоянии ячейки C^{t-1}.

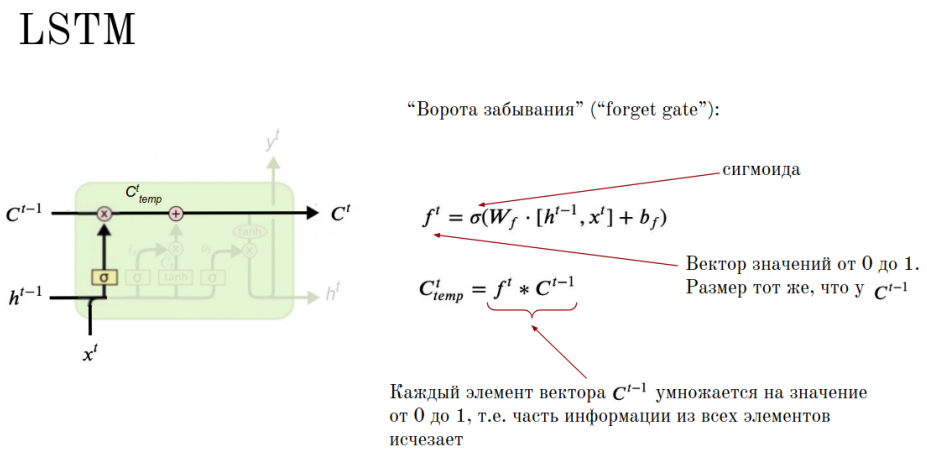


Figure 6: Работа ворот забывания.

$$f^t = \sigma(W_f \cdot [h^{t-1}, x^t] + b_f)$$
$$C^t_{temp} = f^t * C^{t-1}$$

Здесь σ — это сигмоидная функция. Каждый элемент вектора C^{t-1} умножается на значение от 0 до 1, что приводит к частичному или полному ”забыванию” информации.

2.2 Ворота входа (“input gate”)

Ворота входа определяют, какая новая информация будет сохранена в состоянии ячейки.

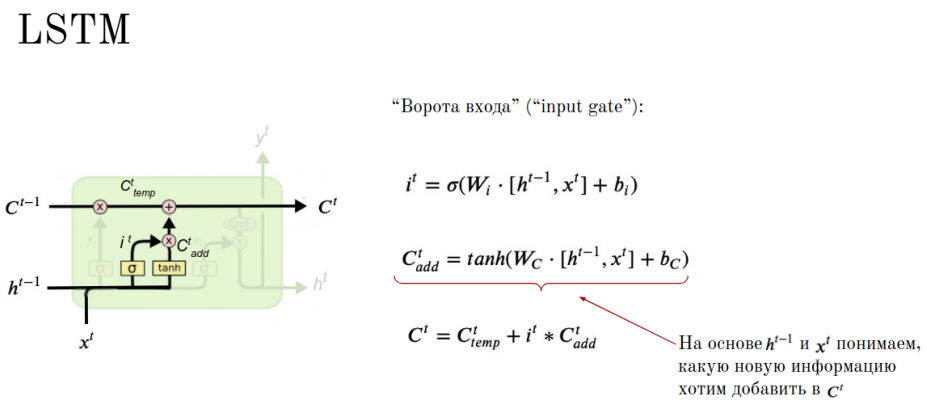


Figure 7: Работа ворот входа.

$$
\begin{aligned}
i^t &= \sigma(W_i \cdot [h^{t-1}, x^t] + b_i) \\
C^t_{add} &= \tanh(W_C \cdot [h^{t-1}, x^t] + b_C) \\
C^t &= C^t_{temp} + i^t * C^t_{add}
\end{aligned}
$$

На основе h^{t-1} и x^t определяется, какую новую информацию следует добавить в C^t.

2.3 Обновление вектора h^t и получение выхода ячейки y^t

Выходной вентиль решает, какая часть состояния ячейки будет на выходе.

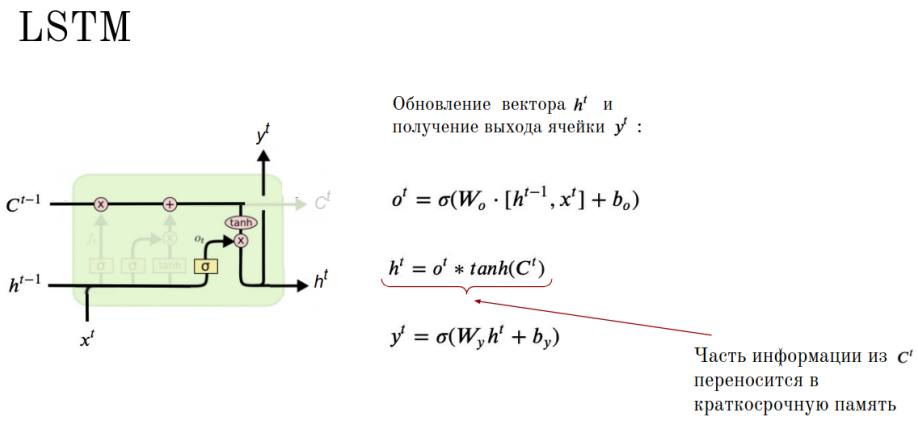


Figure 8: Обновление скрытого состояния и вычисление выхода.

$$
\begin{aligned}
o^t &= \sigma(W_o \cdot [h^{t-1}, x^t] + b_o) \\
h^t &= o^t * \tanh(C^t) \\
y^t &= \sigma(W_y h^t + b_y)
\end{aligned}
$$

Часть информации из C^t переносится в краткосрочную память, представленную вектором h^t, который также используется для получения выхода y^t.

3 GRU (Gated Recurrent Unit)

Управляемый рекуррентный блок (GRU) — это облегченный вариант LSTM, представленный в 2014 году.

3.1 Облегченный вариант LSTM

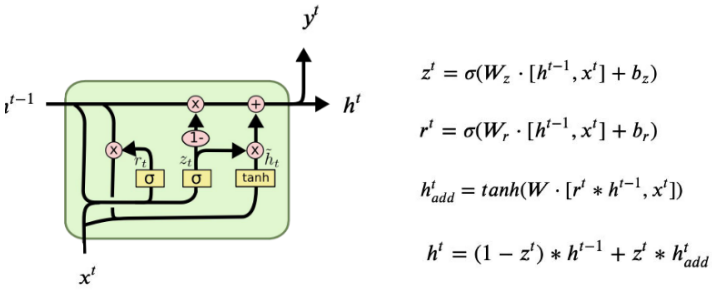
Архитектура GRU проще, чем у LSTM, и, следовательно, вычислительно более эффективна.

$$
\begin{aligned}
z^t &= \sigma(W_z \cdot [h^{t-1}, x^t] + b_z) \\
r^t &= \sigma(W_r \cdot [h^{t-1}, x^t] + b_r) \\
h^t_{add} &= \tanh(W \cdot [r^t * h^{t-1}, x^t]) \\
h^t &= (1 - z^t) * h^{t-1} + z^t * h^t_{add}
\end{aligned}
$$

Вентиль обновления z^t помогает решить, сколько информации из предыдущего скрытого состояния нужно сохранить. Вентиль сброса r^t определяет, как объединить новый вход с предыдущей информацией.

GRU

“Облегченный вариант” LSTM



$$z^t = \sigma(W_z \cdot [h^{t-1}, x^t] + b_z)$$

$$r^t = \sigma(W_r \cdot [h^{t-1}, x^t] + b_r)$$

$$h^t_{add} = tanh(W \cdot [r^t * h^{t-1}, x^t])$$

$$h^t = (1 - z^t) * h^{t-1} + z^t * h^t_{add}$$

Figure 9: Архитектура ячейки GRU.