

LSTM и генерация текста

Маша Шеянова, masha.shejanova@gmail.com

RNN, LSTM

Вспомним обычную полносвязную нейросеть

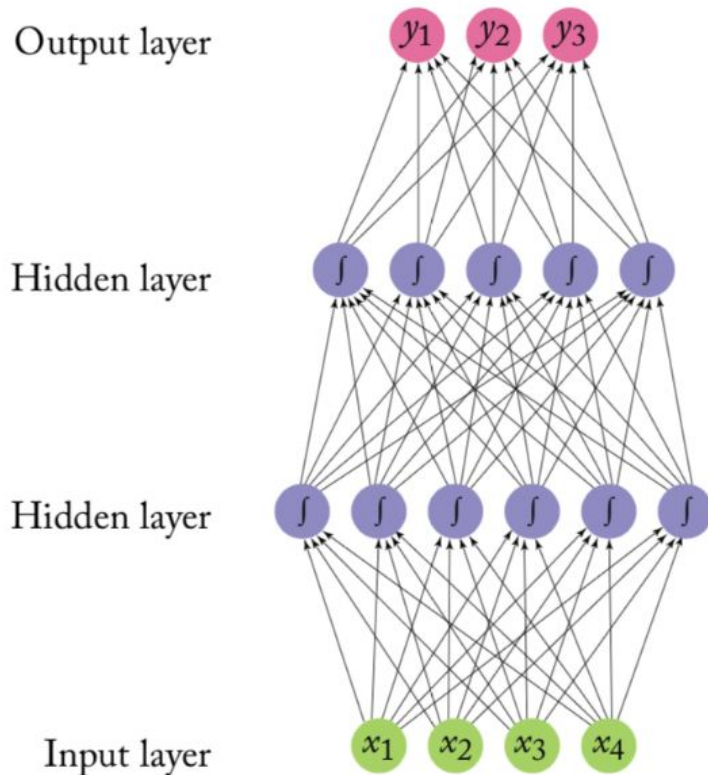
x — входные данные (признаки); W — веса

$$h_1 = f_1(W_1 * x + b_1)$$

$$h_2 = f_2(W_2 * h_1 + b_2)$$

$$y_{\text{pred}} = f_3(W_3 * h_2 + b_3)$$

$$y_{\text{pred}} = f_3(W_3 * f_2(W_2 * h_1 + b_2) + b_3)$$



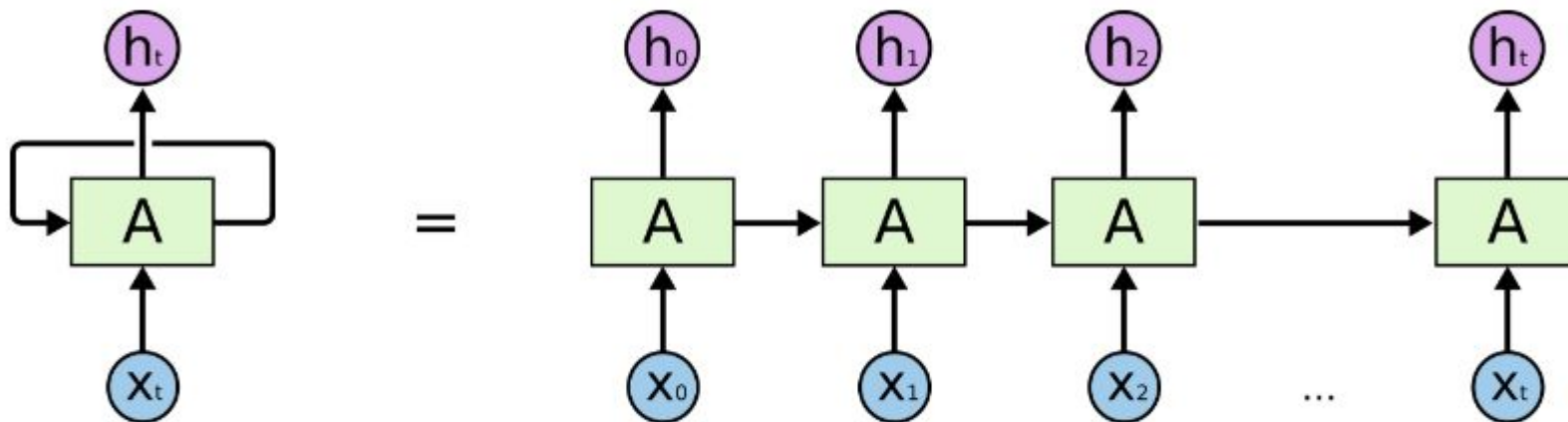
RNN (Recurrent Neural Network)

- в обычной нейросети, работая с отдельными словами, мы будем каждый раз подавать ей эмбединг (вектор) конкретного слова
- но работая с каждым словом в тексте, **мы хотим знать контекст**
- можно конкатенировать вектор текущего и предыдущего слова, но одного мало, а все слова в предложении — это слишком
- решение: **кодировать одним вектором весь предыдущий контекст**

Как? На каждом шаге нейросеть получает **вектор текущего слова** и **вектор выдачи нейросети** с предыдущего шага. Конкатенирует их, и дальше работает как обычная нейросеть.

RNN

В отличие от обычной нейросети, они получают на вход не только данные, но и выход предыдущей клетки RNN.



An unrolled recurrent neural network.

LSTM (Long Short Term Memory)

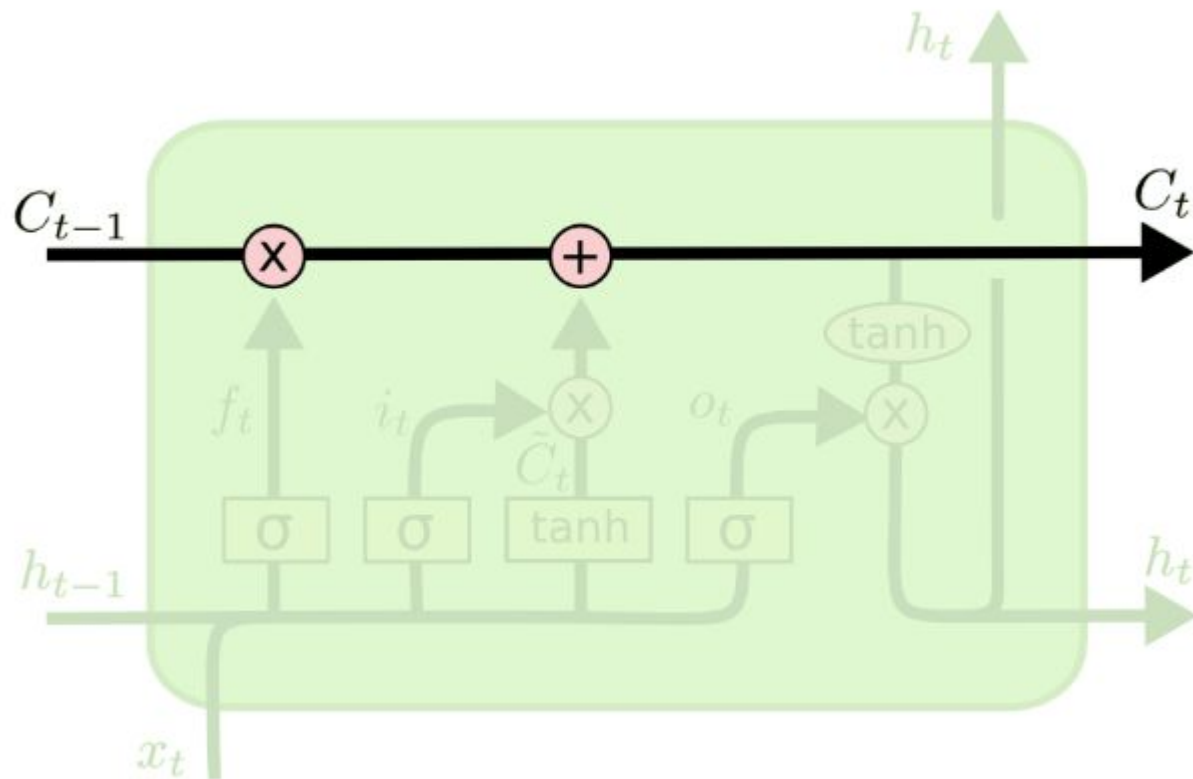
Проблемы с простой RNN:

- важная контекстная информация слишком быстро затирается новой
- с другой стороны, нет механизма, чтобы забывать ненужную информацию (например, забыть предыдущее предложение)

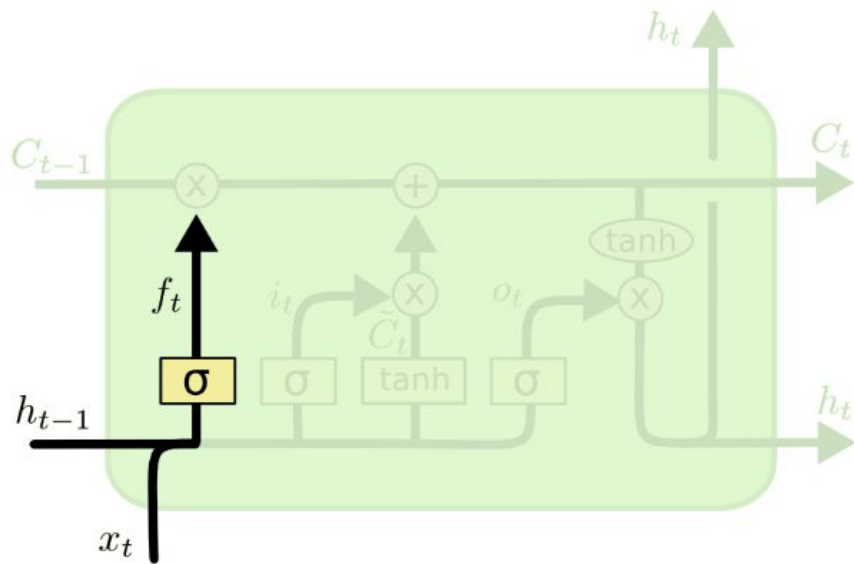
Решение — **два** контекстных вектора:

- долговременной памяти (слабо изменяется от клетки к клетке)
- кратковременной памяти (выдача предыдущей клетки)

Вектор долговременной памяти



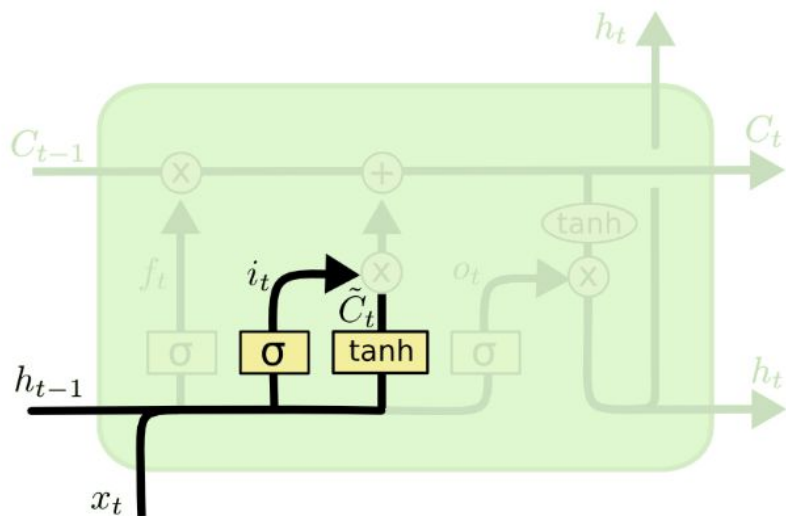
Выбираем, что “забыть”



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

Запоминаем новое

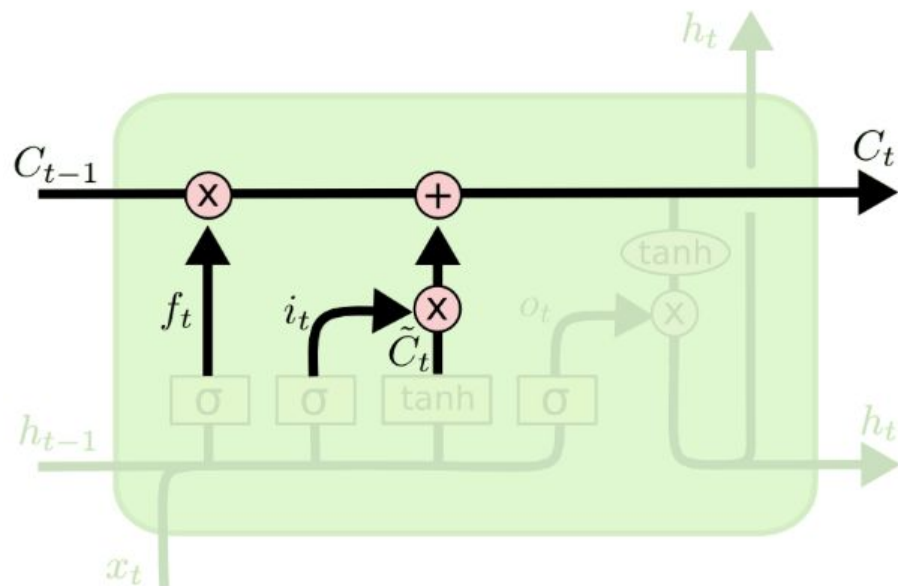
- i_t — выбираем, что запомнить
- \tilde{C}_t — то, что можно будет запомнить



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

Обновляем долговременную память

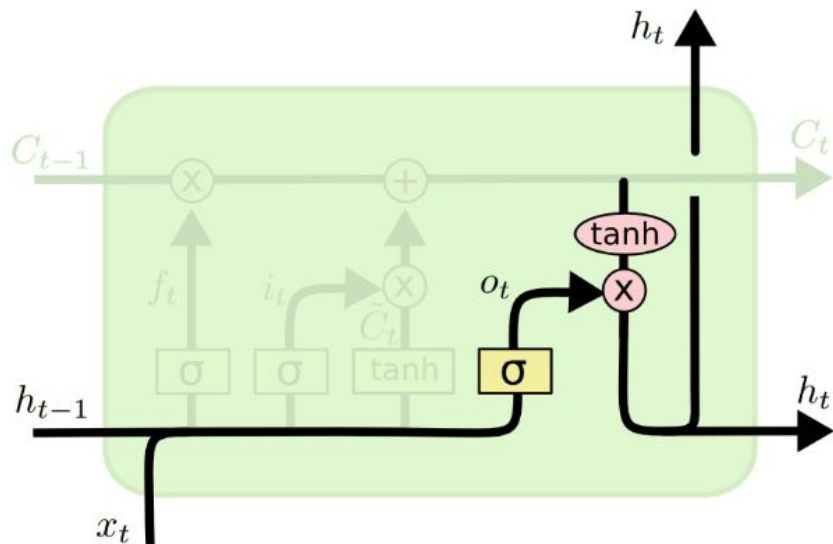
(и больше не трогаем)



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

Получаем результат для текущего шага

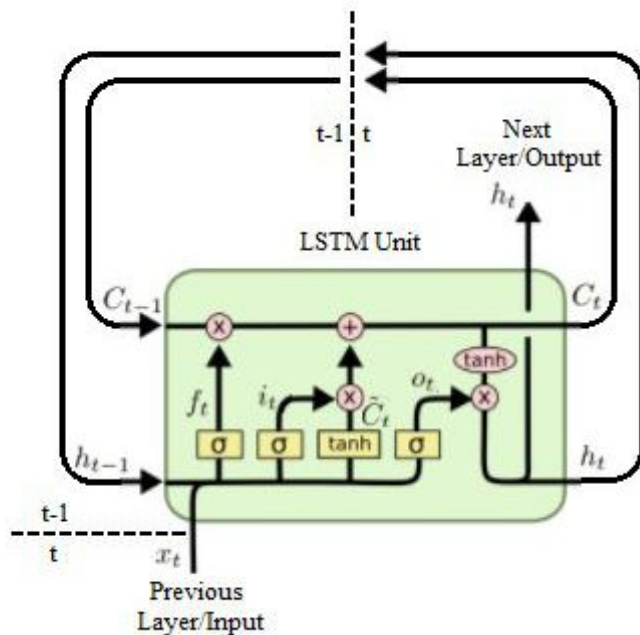
выход нейросети == кратковременная память



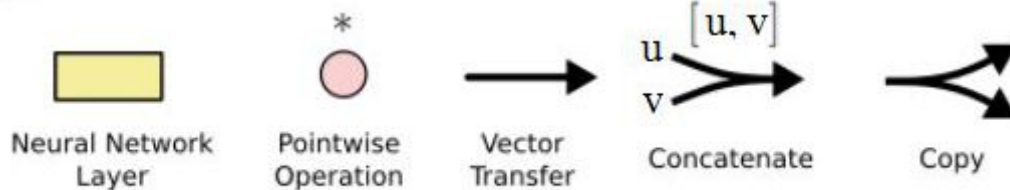
$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

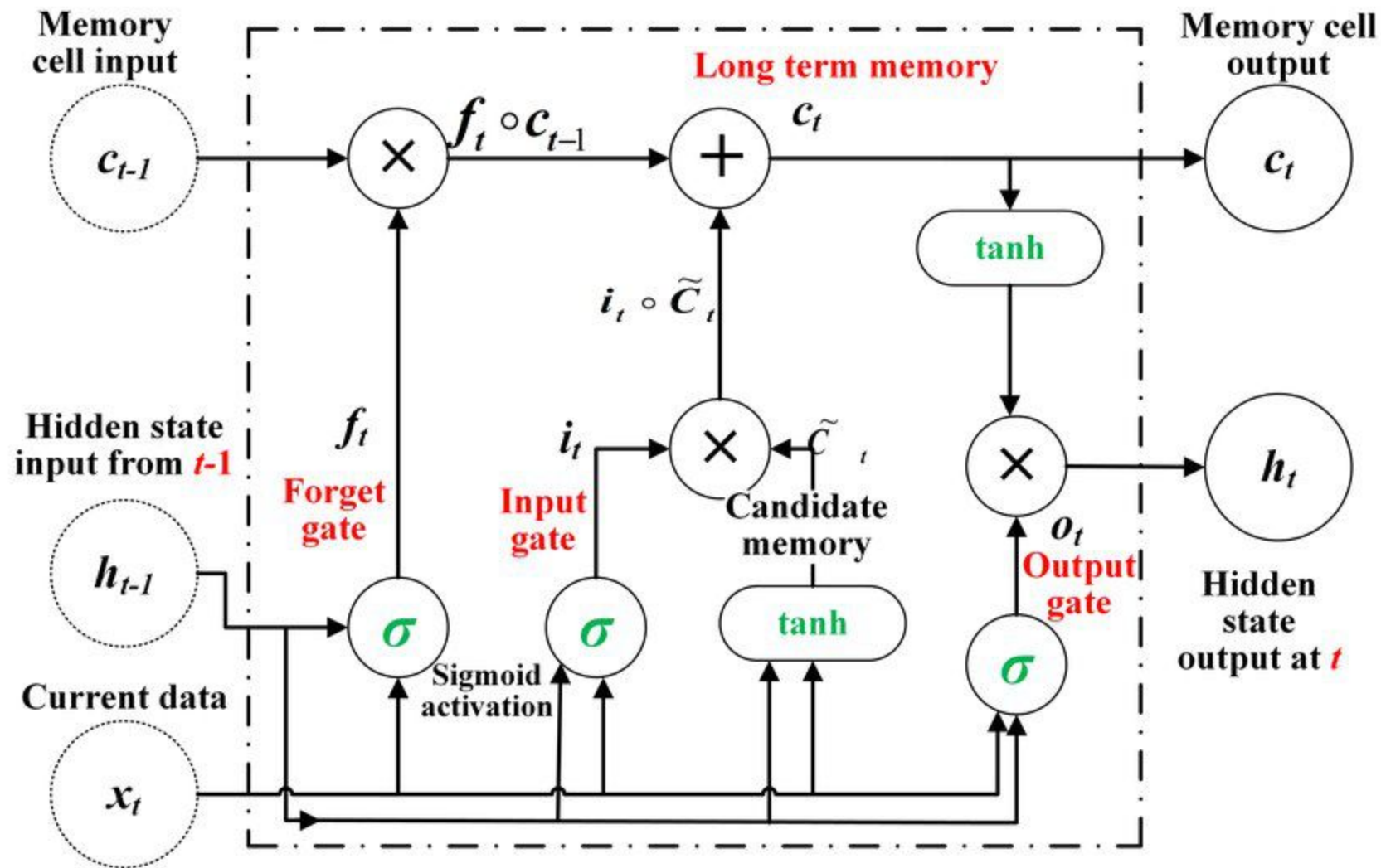
$$h_t = o_t * \tanh(C_t)$$

полная LSTM клетка

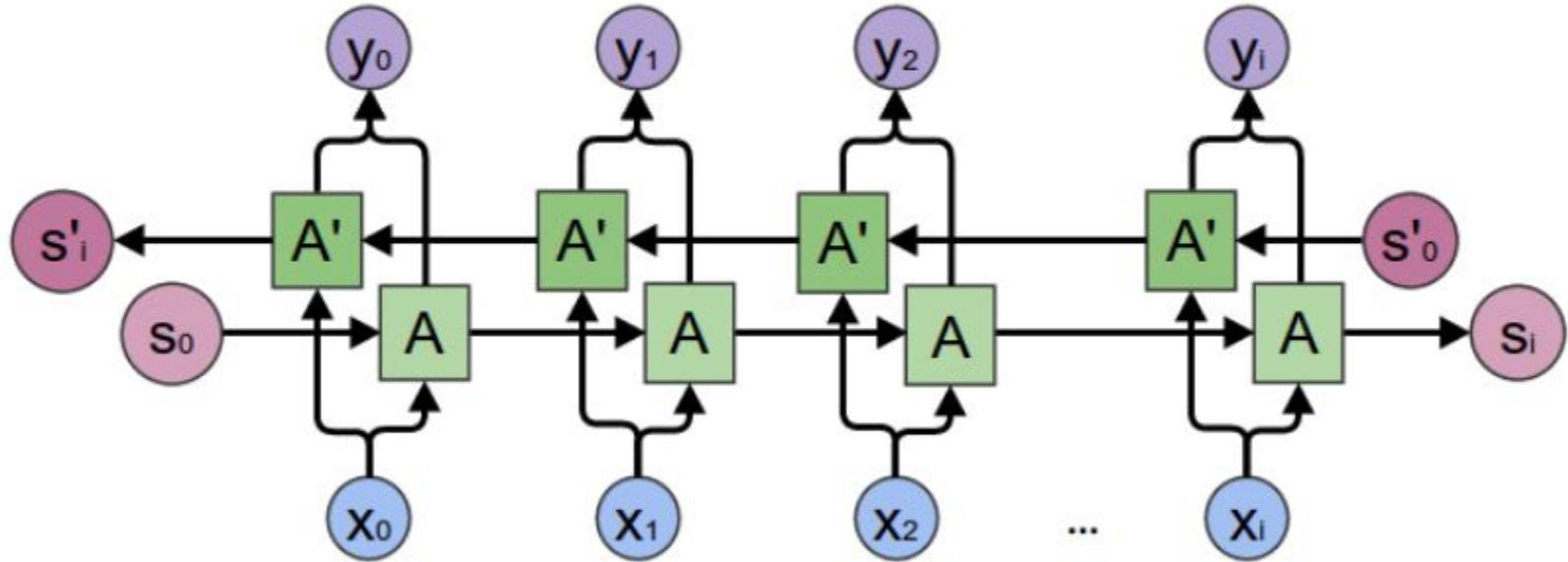


$$\begin{aligned}
 f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
 i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
 \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
 C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
 o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
 h_t &= o_t * \tanh(C_t)
 \end{aligned}$$





Bi-LSTM (Bidirectional LSTM)



Языковые модели

Что умеет языковая модель

- предсказать самое вероятное следующее слово
 - поезд прибыл на _____
- в более общем случае, дать распределение вероятностей для следующего слова
 - поезд прибыл на ... (вокзал: 0.5; юг: 0.1; север: 0.1, ...)
- сравнить вероятности последовательностей
 - поезд прибыл на вокзал
 - поезд прибыл на отдых

Приложения

Выбрать лучший вариант среди возможных:

- спеллчекинг
- распознавание речи
- распознавание символов
- фарзовый машинный перевод

Предложить хорошее продолжение

- автодополнение
- генерация текста

Вероятность последовательности

Пусть $w_{1:n} = w_1, \dots, w_m$ – последовательность слов.

Точная оценка вероятности этой последовательности — **цепное правило**:

$$P(X_1, \dots, X_n) = P(X_1)P(X_2|X_1)\dots P(X_n|X_1, \dots, X_{n-1}))$$

Вероятность следующего слова:

$$P(X_n|X_1, \dots, X_{n-1})) = \frac{P(X_1, \dots, X_n)}{P(X_1, \dots, X_{n-1}))}$$

Но оценить $P(w_k|w_{1:k-1})$ не легче!

Цепи Маркова

Мы пользуемся **марковским предположением**:

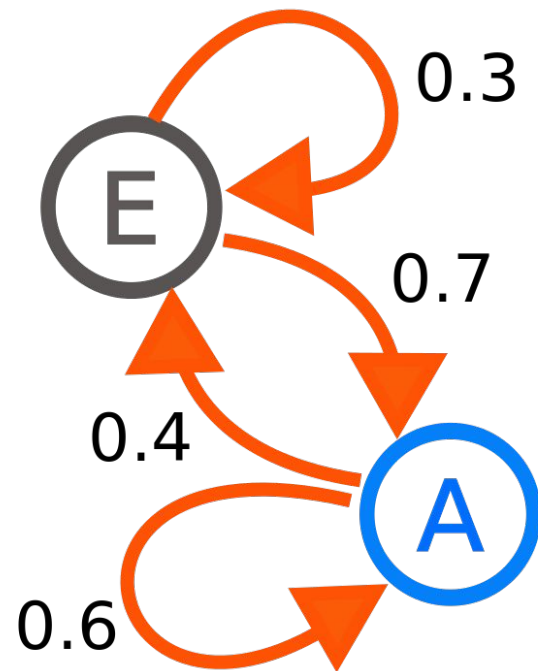
текущее состояние зависит лишь от конечного числа
предыдущих состояний

Иными словами:

$$P(w_i | w_1 \dots w_{i-1}) \approx P(w_i | w_{i-n+1} \dots w_{i-1})$$

Цепи Маркова

- направленный граф
- вершины — “состояния” (в нашем случае, слова или символы)
- на рёбрах — вероятность перехода из одного состояния в другое



От цепей Маркова к RNN

Ограничение марковских цепей на N-граммах:

- маленькое N — слишком несвязная
 - я приехал наконец на питоне писать приятно
- большое N — слишком сильно подстраивается под обучающий корпус

Если данных много, лучше обучить RNN.

- можно учить предсказывать следующее слово
- можно считать косинусное расстояние до следующего слова
- а в генерации текста можно ...

Пример для текста

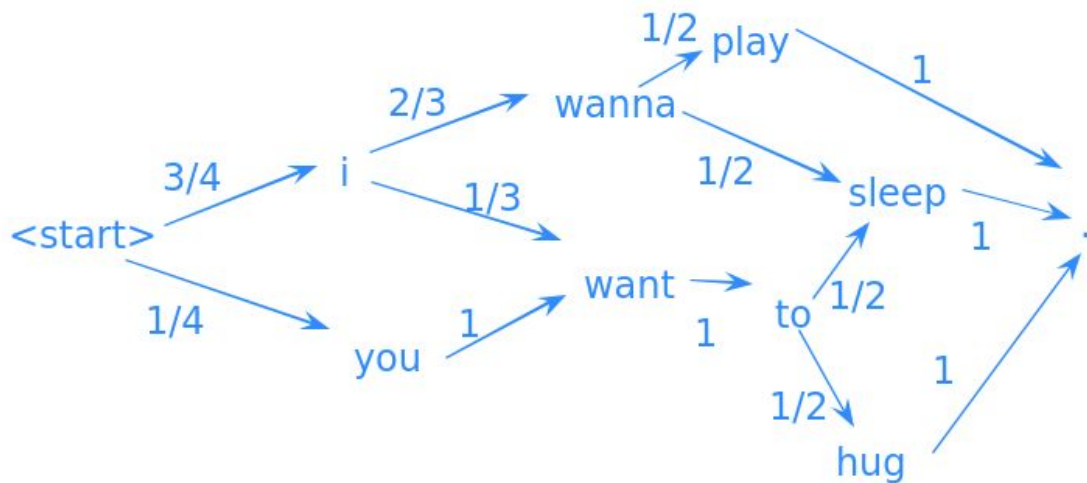
корпус:

<start> i wanna sleep .

<start> i wanna play .

<start> i want to sleep .

<start> you want to hug .



Цепи Маркова на N-граммах

Переходим к n-граммам: $P(w_{i+1}|w_{1:i}) \approx P(w_{i+1}|w_{i-n:i})$, то есть, учитываем $n - 1$ предыдущее слово. Т.е. используем Марковские допущения о длине запоминаемой цепочки.

Модель

- униграм: $P(w_k)$
- биграмм: $P(w_k|w_{k-1})$
- триграмм: $P(w_k|w_{k-1}w_{k-2})$

Генерация текста (NLG)

Зачем?

- развлечение
- развлечение
- ещё раз развлечение
- генерация сюжетов в играх (тоже развлечение)
- чат-бот для психологической помощи?

Во-первых, это весело!



Ветхий Алгоритм @alg_testament · Jul 24

эта процедура должна работать не так, отец мой, отец мой, не бесчести имени Бога твоего.

Translate Tweet



1



33



94



Ветхий Алгоритм @alg_testament · Jul 22

Сигналы, генерируемые терминалом, возникают, когда пользователь входит в дом Господень

Translate Tweet



39



103



Ветхий Алгоритм @alg_testament · Jul 22

И поставлю Себе священника верного; он будет обладать правами суперпользователя.

Translate Tweet



1



82



217



Как?

- самое простое: марковская цепь
- поинтереснее: RNN / LSTM с предсказанием следующего слова
- ещё интереснее: RNN → GAN

GAN (generative adversarial network)

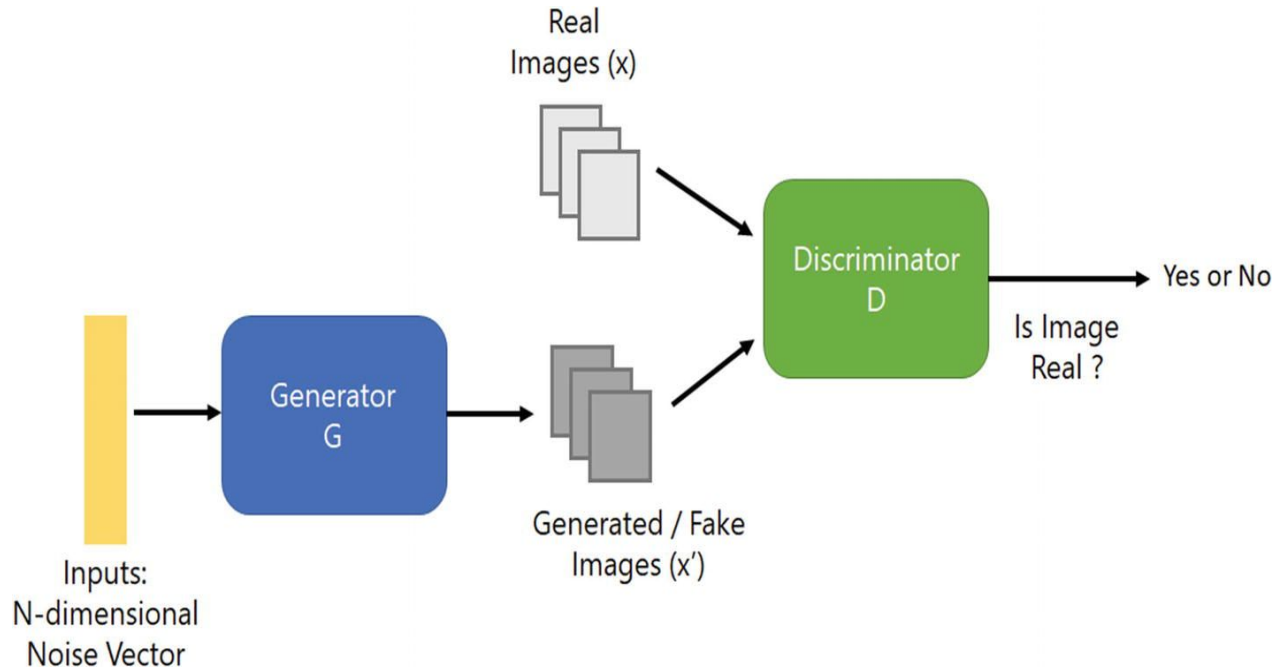
Две нейросети: “генератор” и “дискриминатор”.

- генератор порождает какой-то текст (изначально случайный)
- дискриминатор получает:
 - либо сгенерированный текст
 - либо из “человеческий” (из корпуса)
- ... и пытается определить какой ему текст дали (бинарная классификация)
- функция потерь дискриминатора — отличил ли истинный от “подделки”
- функция потерь генератора — “получилось ли обмануть дискриминатор”
- обучаются обе, но для генерации нужна первая

GAN

Вообще, GANы
пришли из
картинок, где
достигли

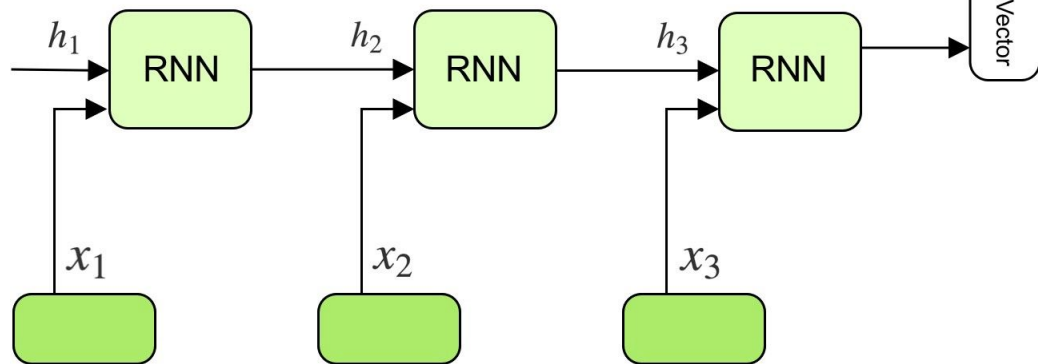
впечатляющих
результатов.



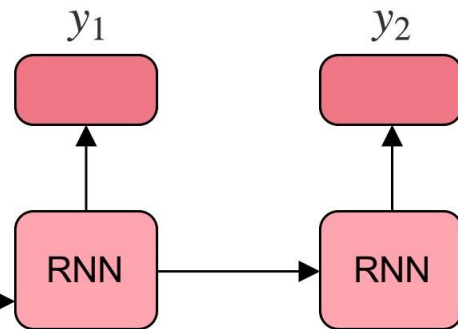
Что дальше?

seq2seq

Encoder



Decoder



seq2seq + attention

Transformers