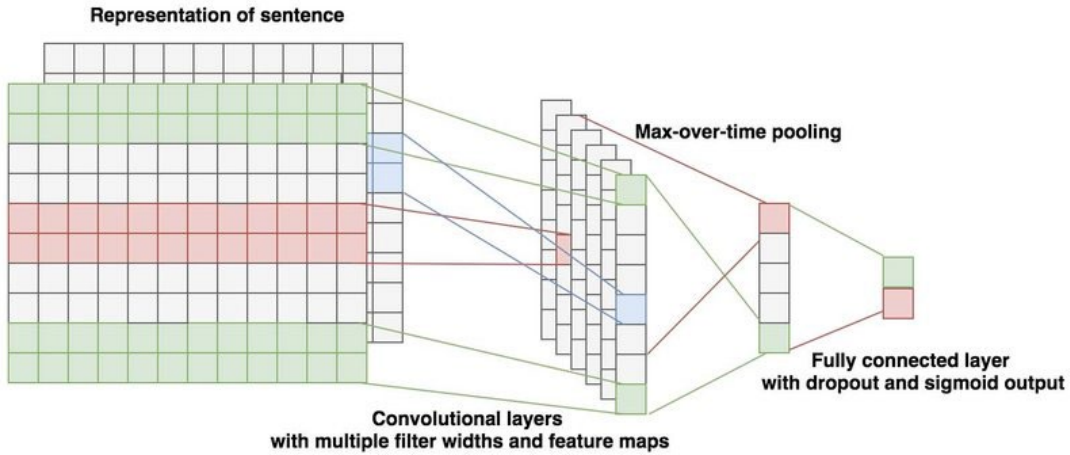
RNN. LSTM. GRU. SEQ2SEQ

РЕКУРРЕНТНЫЕ СЕТИ

Работаем с последовательностями

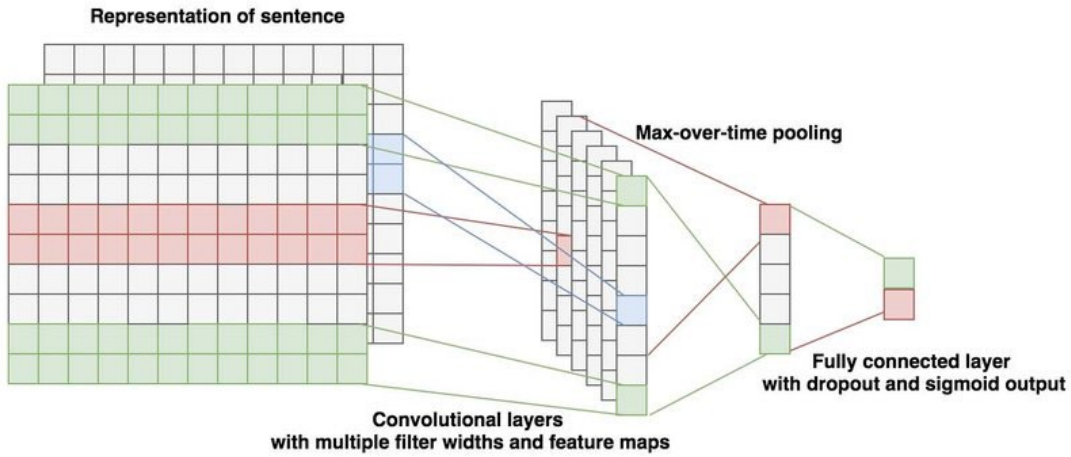
РЕКУРРЕНТНЫЕ СЕТИ

- Как мы работаем с текстами?
- Мы умеем в эмбеддинги для конкретных слов: word2vec, fasttext
- Можно их усреднять (эмбеддинги слов → предложение)
- Или воспользоваться CNN:



РЕКУРРЕНТНЫЕ СЕТИ

- В чем недостаток одномерных CNN?



РЕКУРРЕНТНЫЕ СЕТИ

- Идея: попробуем читать текст так, как это делает человек
- То есть, последовательно
- И чем дальше читаем, тем лучше понимаем, о чем текст

РЕКУРРЕНТНЫЕ СЕТИ

- Идея: попробуем читать текст так, как это делает человек
- То есть, последовательно
- И чем дальше читаем, тем лучше понимаем, о чем текст
- То есть, нам нужно дополнительно хранить информацию о предыдущих инпутах

- Имеем последовательность (слов): $x_1, x_2, x_3, \dots, x_n$
- Читаем слева направо
- Имеем h_t - накопленную информацию о прочитанном (вектор)

РЕКУРРЕНТНЫЕ СЕТИ

- Имеем последовательность (слов): $x_1, x_2, x_3, \dots, x_n$
- Читаем слева направо
- Имеем h_t - накопленную информацию о прочитанном (вектор)
- $h_t = f(W_{xh}x_t + W_{hh}h_{t-1})$
- Если хотим что-то выдавать на каждом шаге: $o_t = f_o(W_{ho}h_t)$

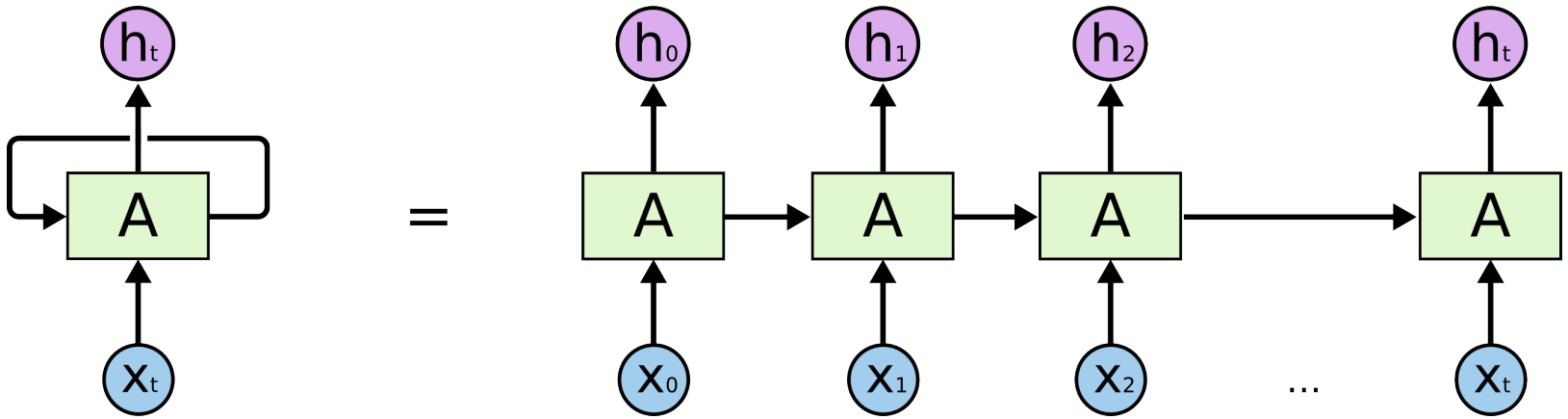
РЕКУРРЕНТНЫЕ СЕТИ

- Имеем последовательность (слов): $x_1, x_2, x_3, \dots, x_n$
- x_i - либо one-hot вектор, либо готовое векторное представление слова (любой предобученный эмбеддинг)

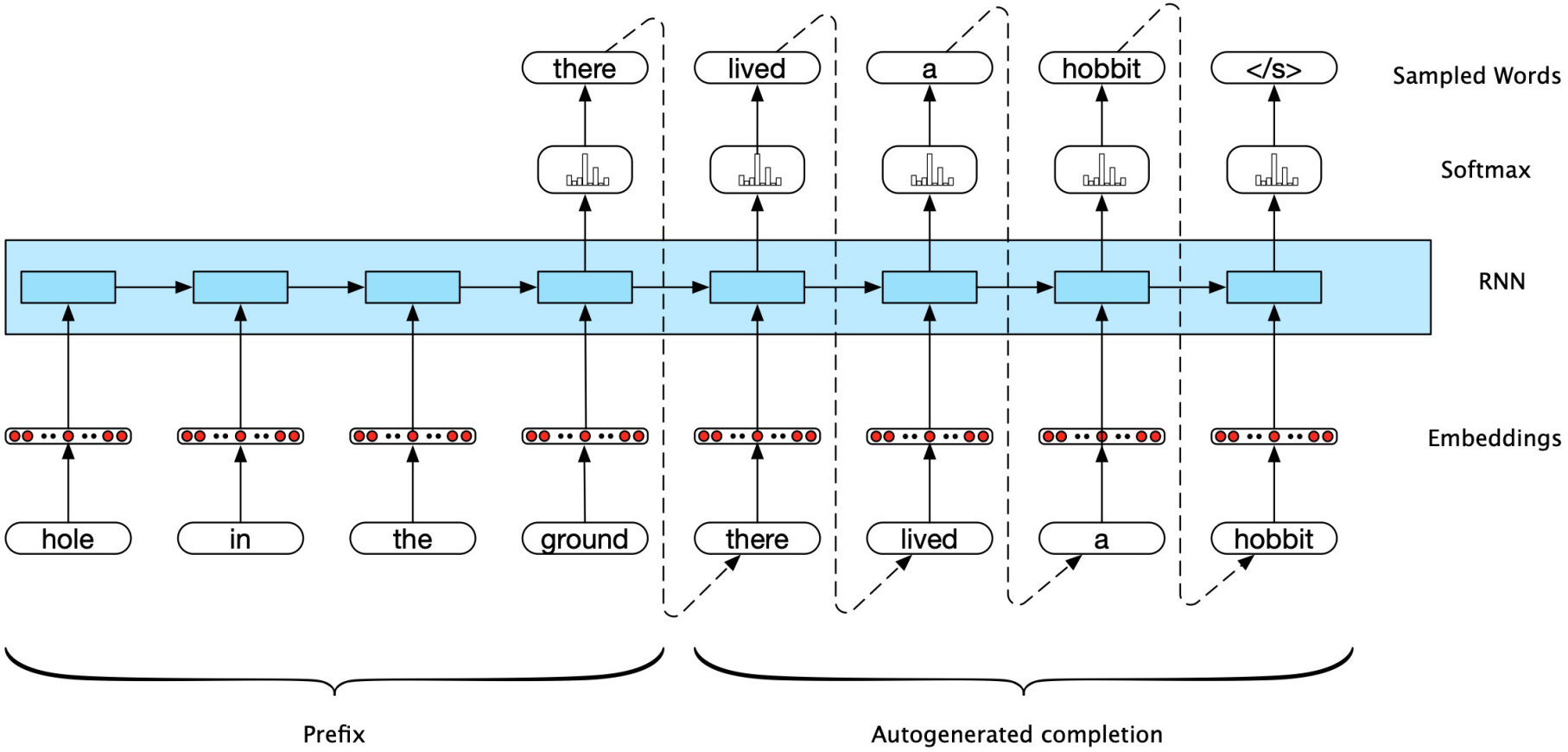
Типичные задачи:

- Генерация текстов (то есть, на каждом o_t предсказываем следующее слово)
- POS-tagging (RNNMorph?)

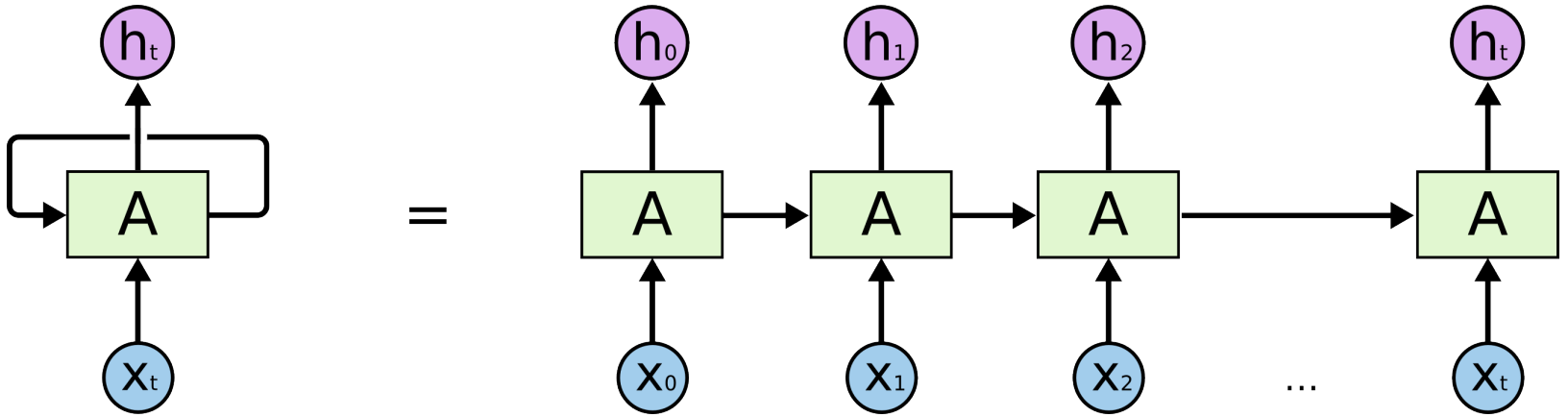
РЕКУРРЕНТНЫЕ СЕТИ



РЕКУРРЕНТНЫЕ СЕТИ

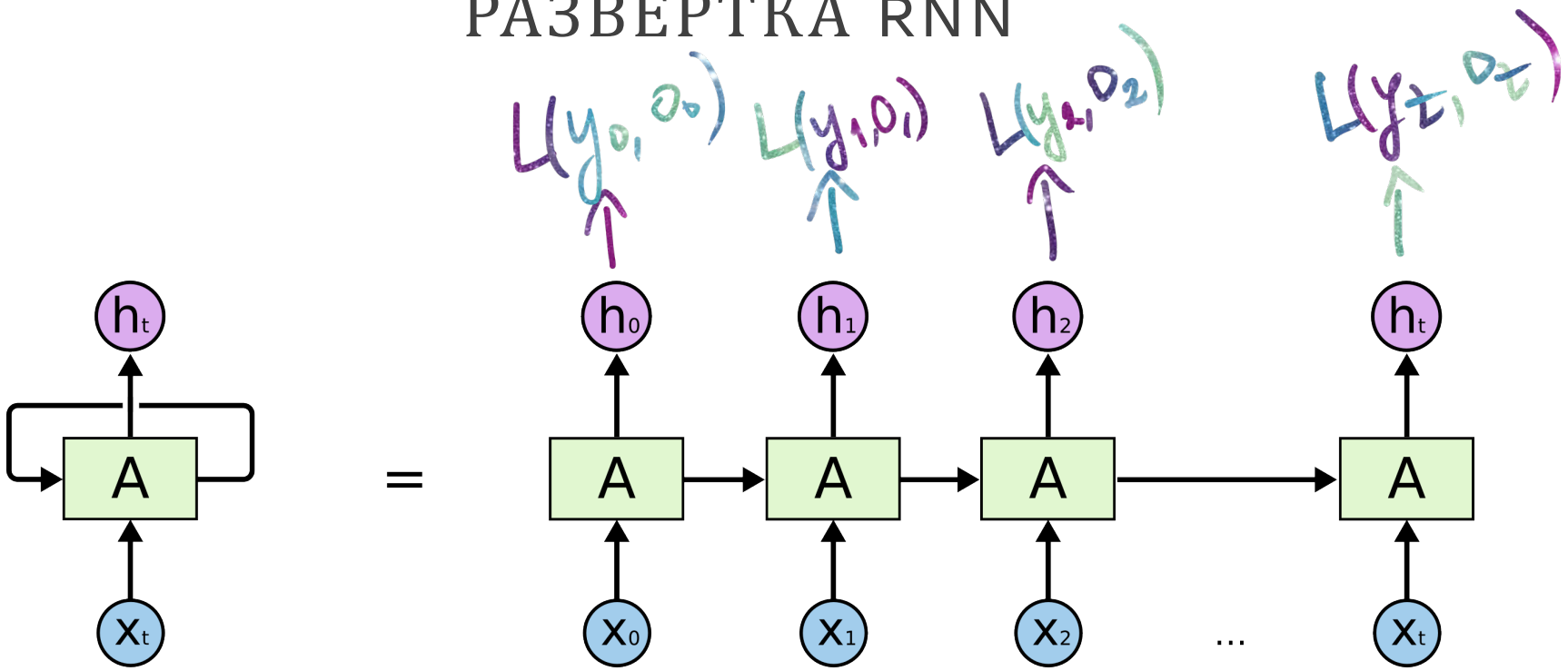


РАЗВЕРТКА RNN



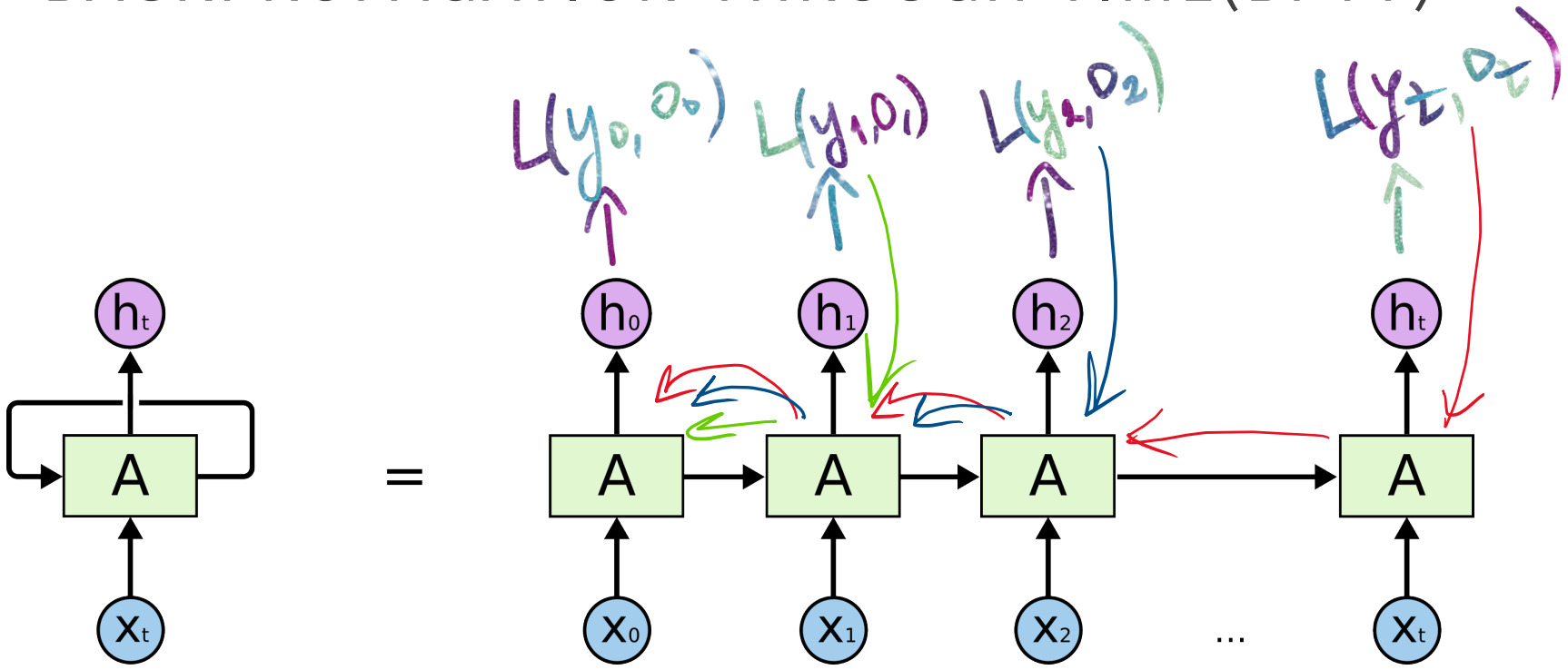
$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1})$$
$$o_t = f_o(W_{ho}h_t)$$

РАЗВЕРТКА RNN



$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1})$$
$$o_t = f_o(W_{ho}h_t)$$

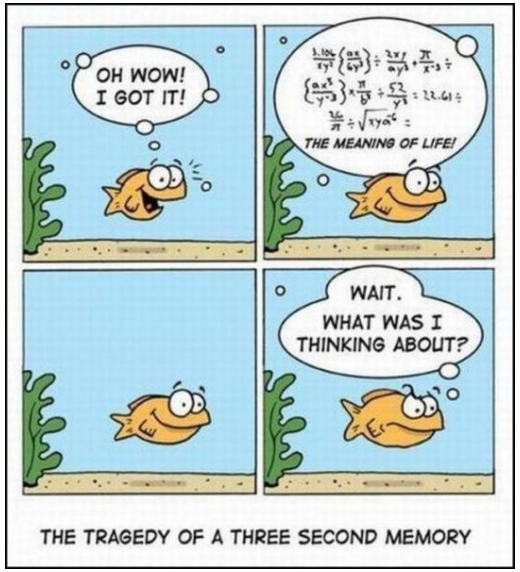
BACKPROPAGATION THROUGH TIME(BPTT)



$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1})$$
$$o_t = f_o(W_{ho}h_t)$$

ПРОБЛЕМЫ С ГРАДИЕНТАМИ

- Сигнал теряется по мере прохождения
- Не факт, что получится обучить зависимость финального вектора h_t от первых слов в тексте
- Градиенты перемножаются: умножение множества слагаемых нестабильно
- Можно либо словить 0, либо взрыв градиента
- 0 означает, что градиенты не будут течь, и зависимость потеряется

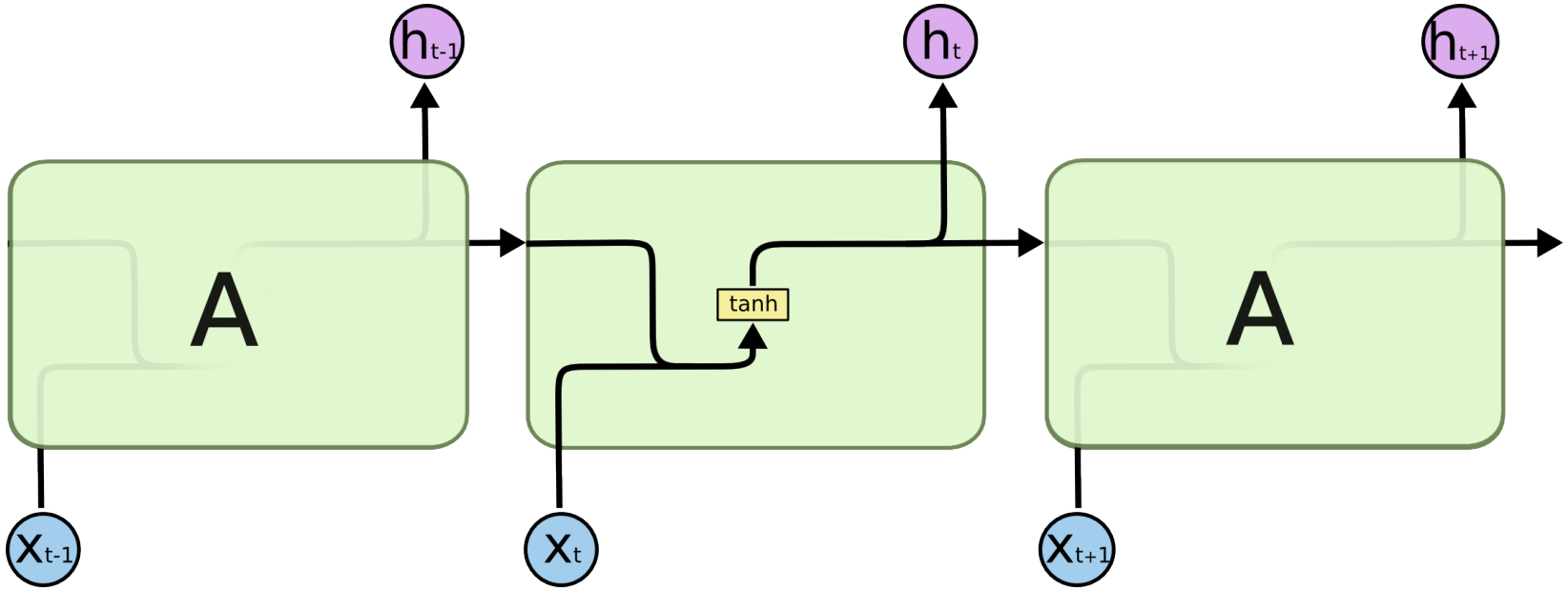


LSTM (LONG SHORT-TERM MEMORY)

Решаем проблему с затуханием градиентов

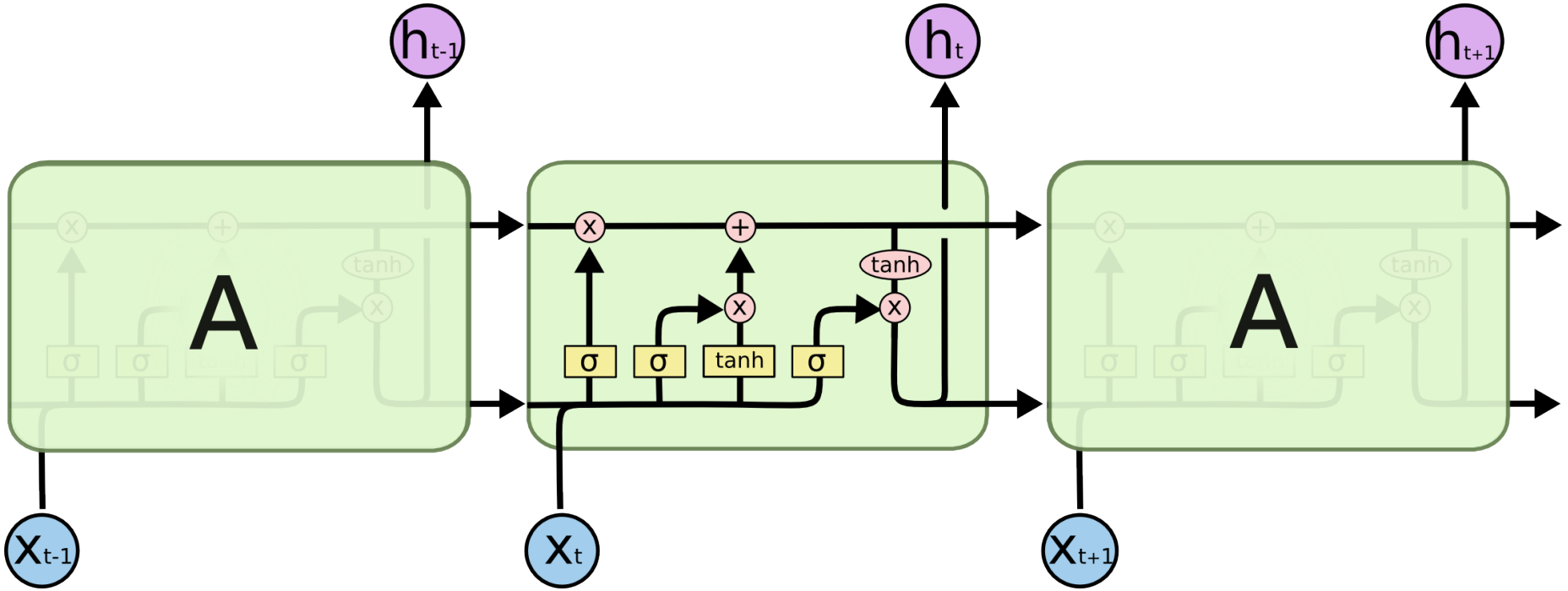
LSTM

- Обычное устройство RNN:



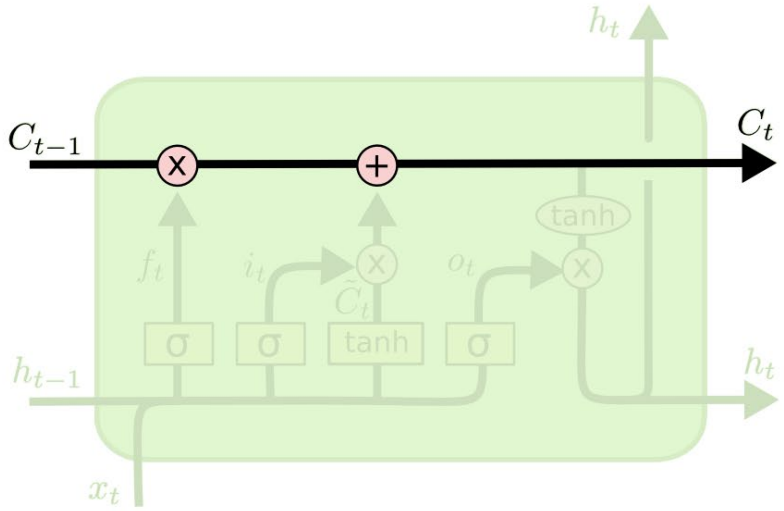
LSTM

- Давайте дополнительно будем пробрасывать значения без изменений



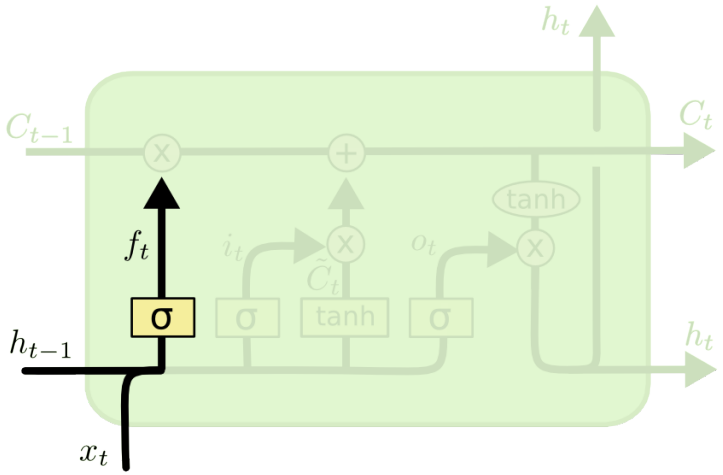
LSTM

- У каждой ячейки теперь будет свое состояние ячейки (cell state)
- К этому состоянию мы что-то будем добавлять или убирать



LSTM

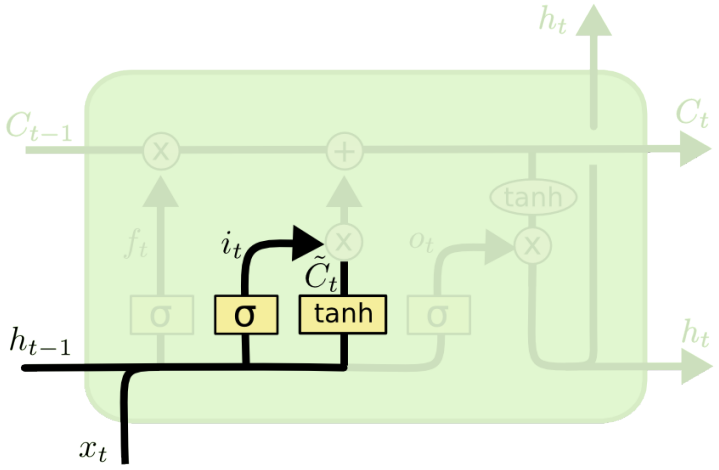
- Часть информации нужно выкидывать: мы не хотим помнить совсем все, а то на все памяти не хватит
- Давайте заведем отдельную матрицу весов и будем пробрасывать сигмоиду про «забыть»: 0 – забудь, 1 – помни
- Это будут наши врата забвения (gate)



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

LSTM

- Часть информации нужно обновлять: решить, что обновляем и как
- Еще один гейт решает, какие значения мы обновляем
- Матрица $\tilde{C}_t$ содержит в себе значения, на которые нужно обновить выбранное



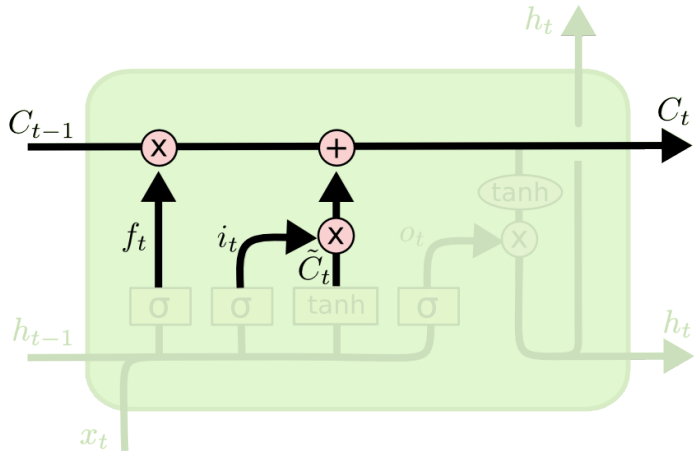
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

LSTM

- Как это все взаимодействует:

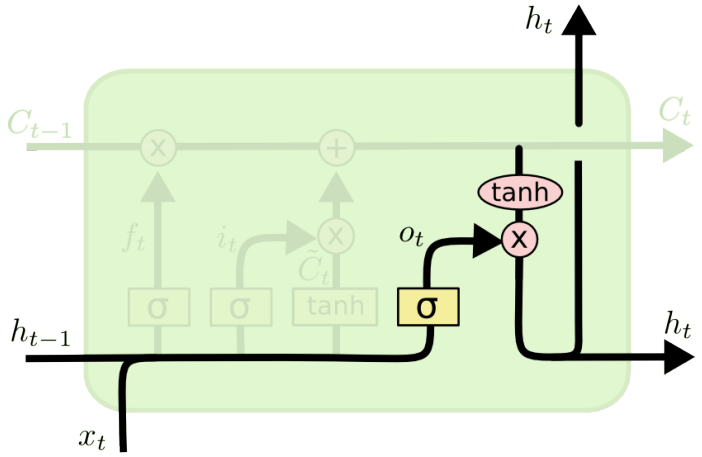
1. Перемножаем старое состояние на значения гейта забывания и забываем че-то (зануляем)
2. Перемножаем значения гейта инпута и матрицу новых значений, чтобы получить только те, которые нужно обновить
3. Прибавляем новые значения к старым



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

LSTM

- Наконец, нам нужно еще что-то выдать в промежуточный аутпут.
- Нам нужна сигмоида, которая решит, какие части состояния мы выплевываем
- А потом прогоняем состояние через tanh и уже выдаем окончательный ответ, перемножив на результат сигмоиды



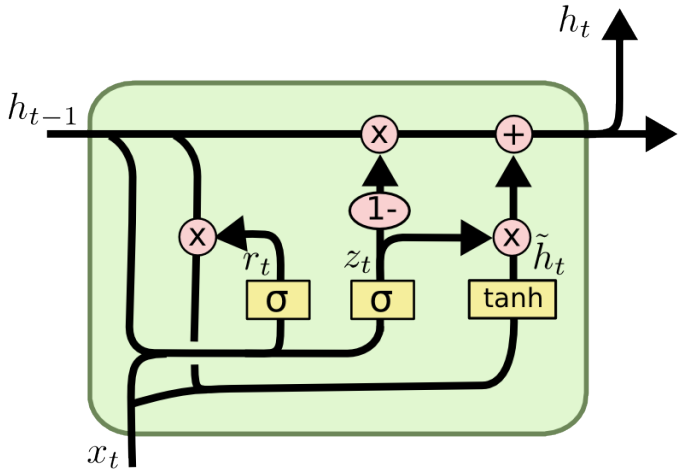
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

GRU

GRU

- Gated Recurrent Unit (GRU) – это только слегка модифицированная версия LSTM
- Позволяет сразу выплевывать ответы без дополнительной матрицы весов
- Следовательно, работает чуточку быстрее, но не намного
- Качество и скорость LSTM & GRU сопоставимы



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

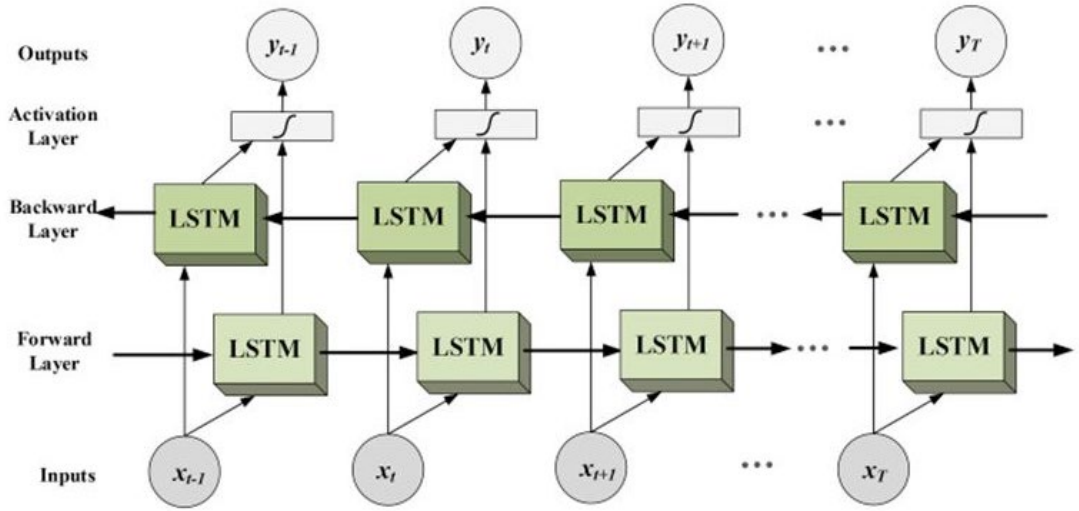
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Примерно все те же объяснения текстом

BI-LSTM

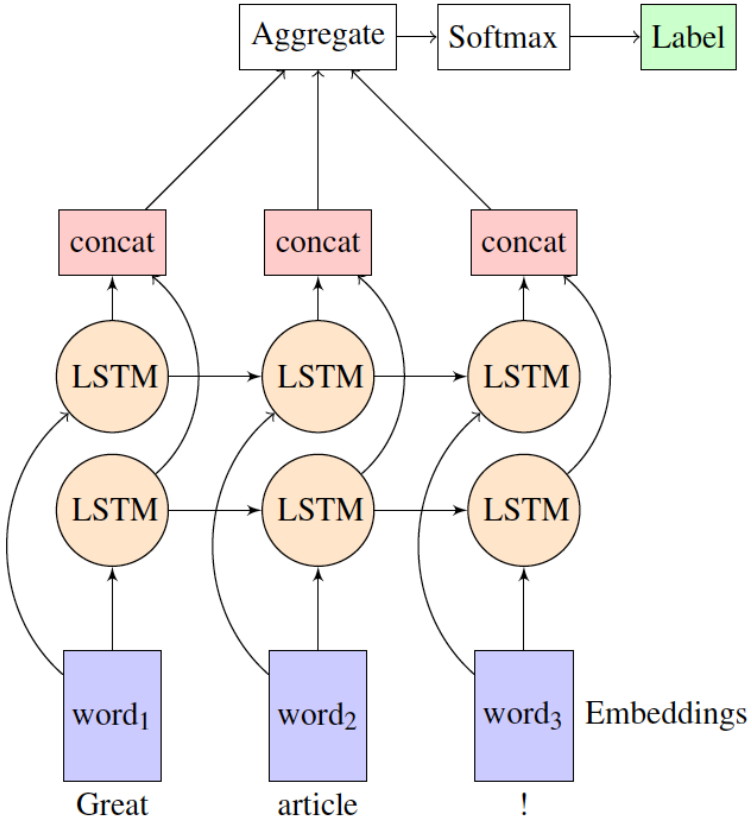
BI-LSTM

- Почему мы определяем часть речи только по предыдущим словам?
- Будем смотреть и на следующие слова



BI-LSTM

- Предсказание для слова строится по скрытым состояниям, учитывающим весь контекст
- Как их, кстати, соединить?



SEQ2SEQ

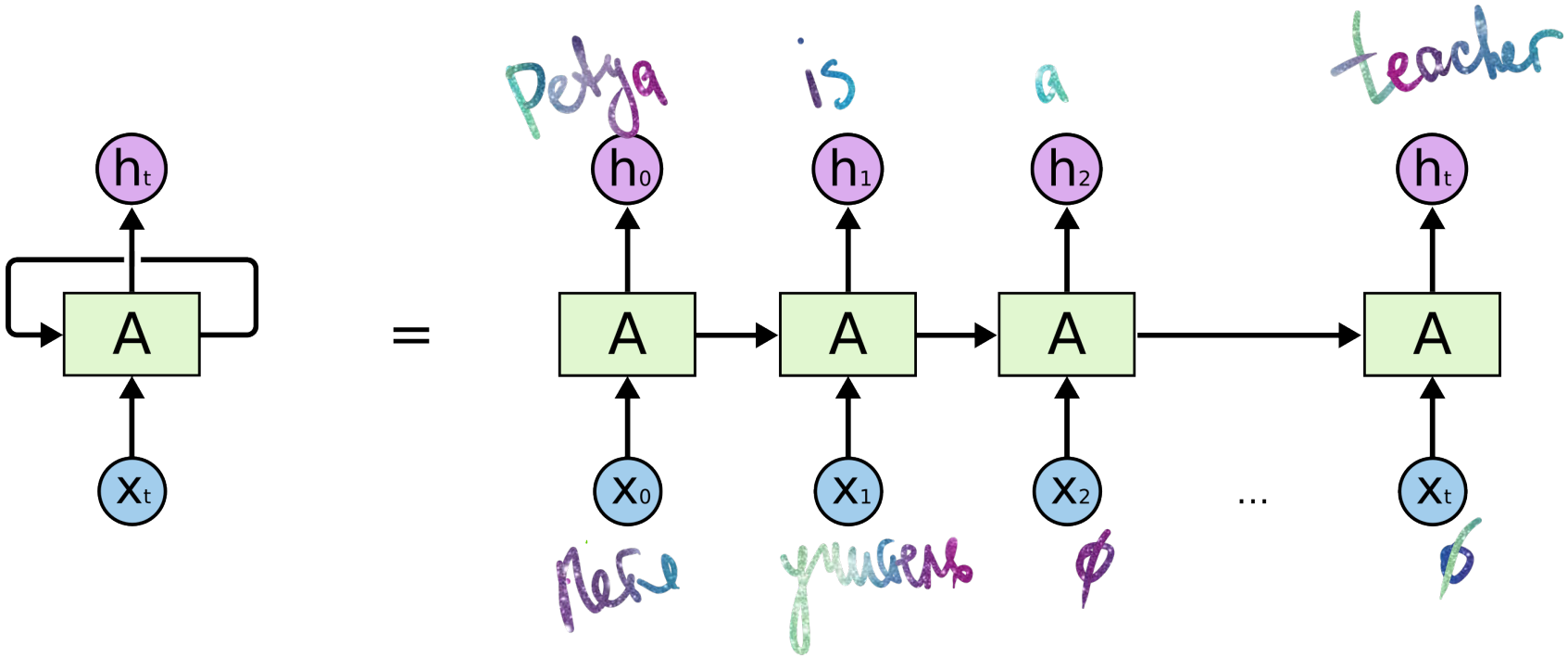
SEQ2SEQ

Sequence to Sequence

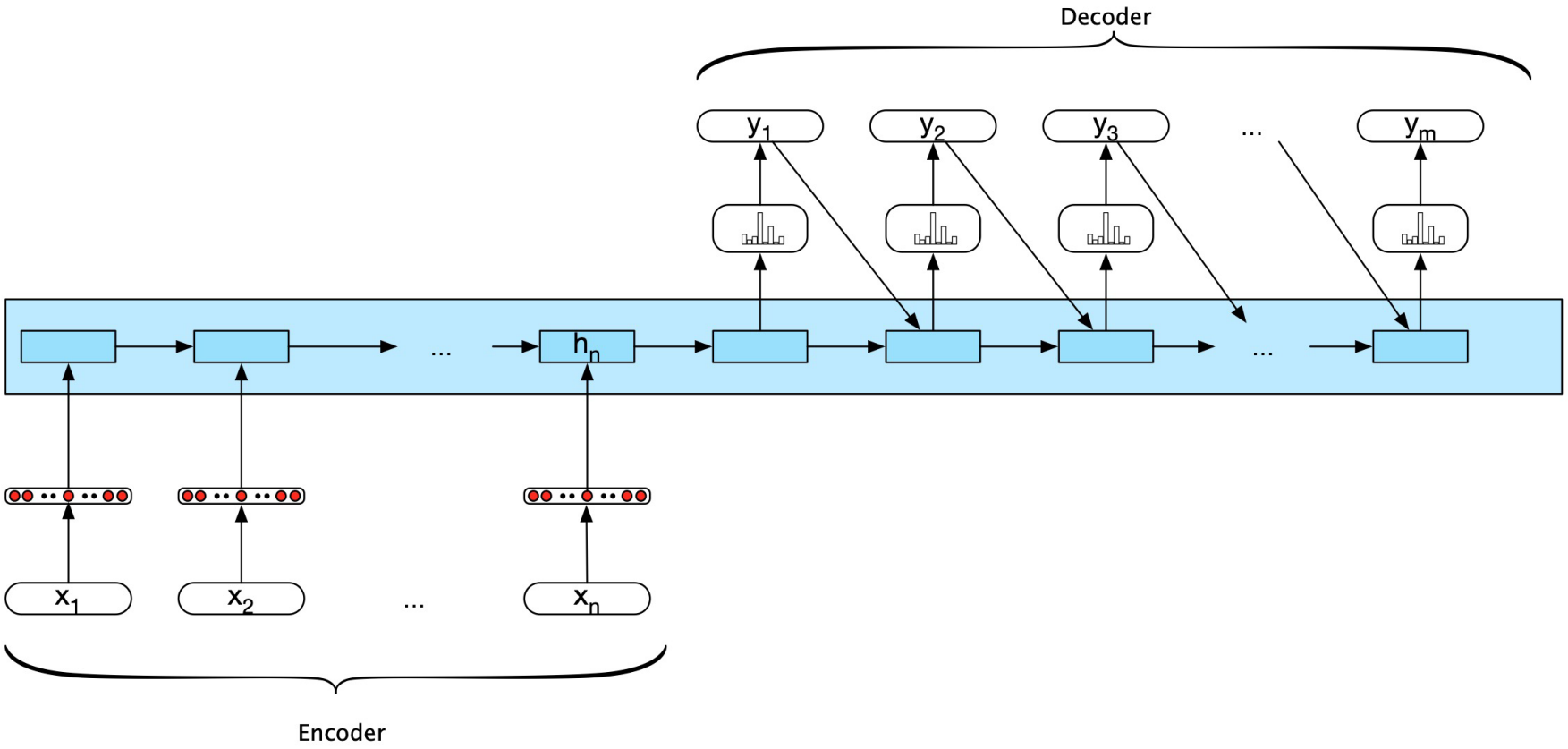
- Машинный перевод
- Суммаризация текста
- Генерация комментариев к коду
- Математические преобразования
- Смена стиля текста

SEQ2SEQ MACHINE TRANSLATION

- Что делать, если длины входного и выходного текстов разные?



SEQ2SEQ MACHINE TRANSLATION



SEQ2SEQ MACHINE TRANSLATION

- В конце входного текста ставим специальный токен <EOS>
- Прогоняем входной текст через RNN
- Скрытое состояние после всего текста — «контекст»
- Контекст передаётся в RNN, которая генерирует выходной текст
- Используется Beam Search

BEAM SEARCH

- Выбираем B вариантов для первого слова по максимальной вероятности
- Для каждого рассматриваем все возможные варианты для следующего слова, оставляем B наиболее вероятных вариантов
- И так далее