Table of Contents

Приложения рекуррентных нейронных сетей

- Прогнозирование временных рядов
- Управление технологическими процессами
- Классификация текстов или их фрагментов
- Анализ тональности документа/предложений/слов
- Машинный перевод
- Распознавание речи
- Синтез речи
- Синтез ответов на вопросы, разговорный интеллект
- Генерация подписей к изображениям
- Генерация рукописного текста
- Интерпретация генома и другие задачи биоинформатики
- ...

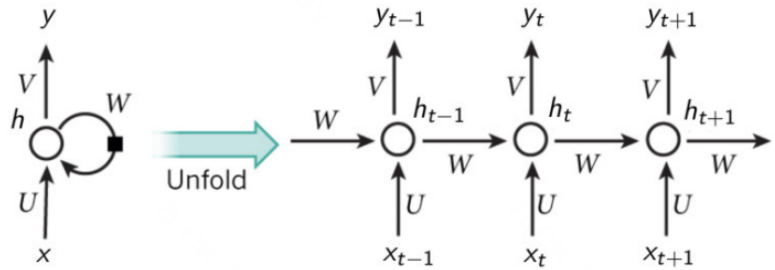
Общие положения

x_t — входной вектор в момент времени t

h_t — вектор скрытого состояния в момент времени t

y_t — выходной вектор (иногда $y_t = h_t$)

$$h_t = \sigma_h(Ux_t + Wh_{t-1})$$
$$y_t = \sigma_y(Vh_t)$$



Обучение сети

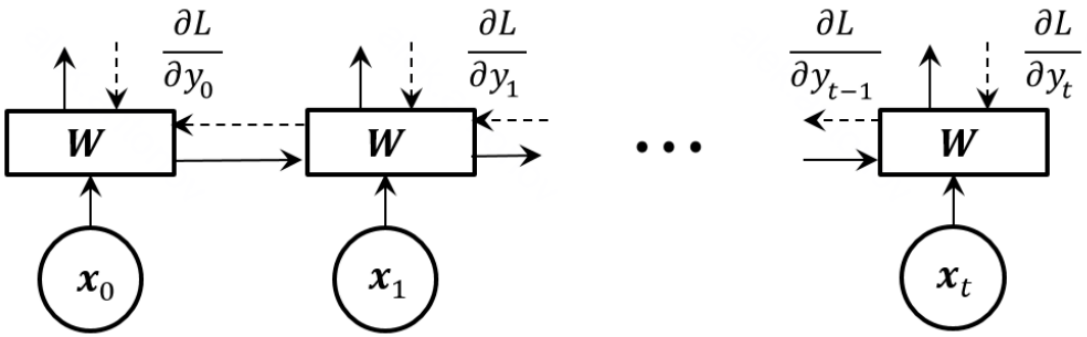
$$\sum_{t=0}^{T} \mathcal{L}_t(U, V, W) \to \min_{U,V,W}$$

$\mathcal{L}_t(U, V, W) = \mathcal{L}(y_t(U, V, W))$ – потеря от предсказания y_t

При Backpropagation получается, что

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial y_t} \frac{\partial y_t}{\partial h_t} \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

Обучение сети



Из-за большого количества произведений могут возникать проблемы потери точности, затухания или взрыва градиентов.

Борьба с затуханием и взрывом

- Gradient clipping
- Усложнение архитектуры: LSTM/GRU
- Регуляризация $\frac{\partial h_i}{\partial h_{i-1}} \to 1$ (не популярно)
- Truncated Backpropagation Through Time - ограничивать количество множителей (не популярно)

Gradient clipping

Идея в ограничении норм градиентов:

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

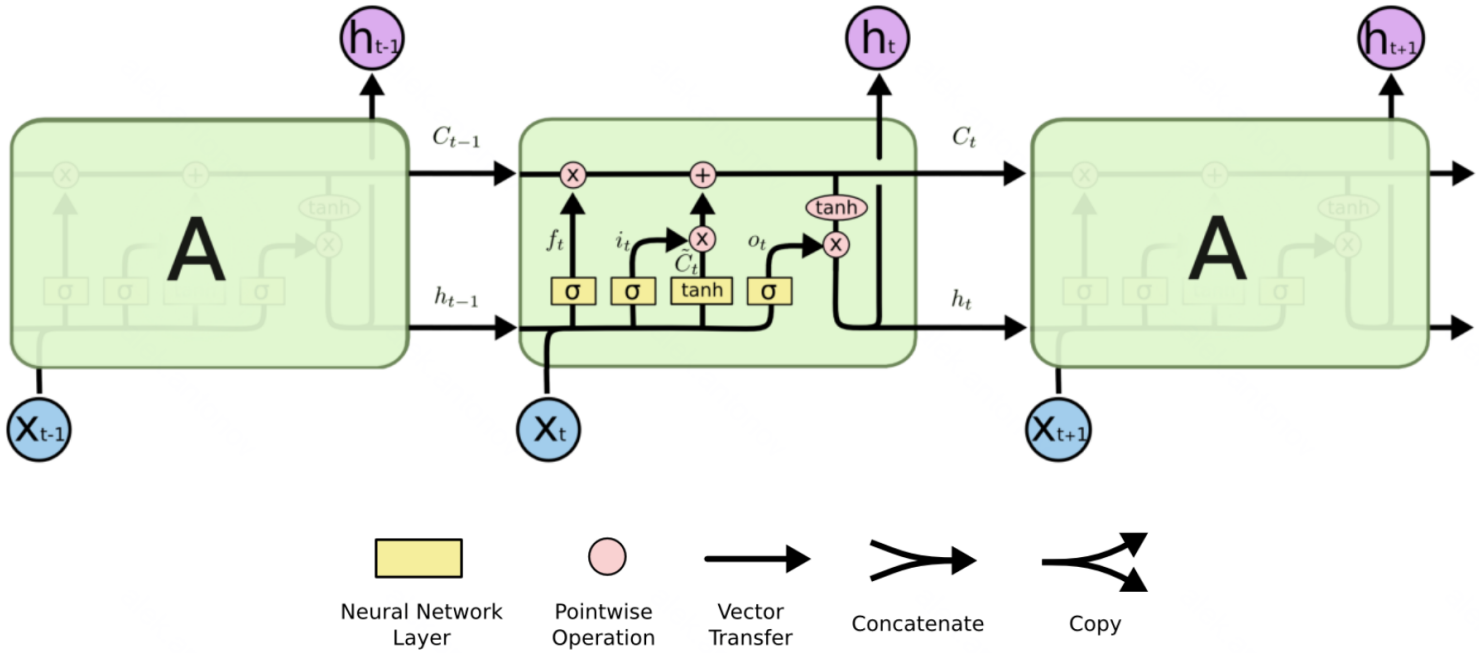
$\hat{g} \leftarrow \frac{\partial \mathcal{L}_t}{\partial W}$
if $||\hat{g}|| \geq$ *threshold* **then**
    $\hat{g} \leftarrow \frac{\textit{threshold}}{||\hat{g}||}\hat{g}$
end if

Используется для борьбы со взрывом.

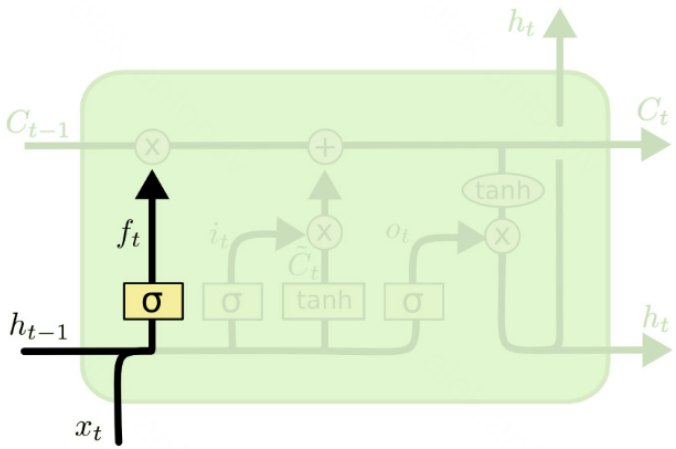
Table of Contents

Long short-term memory (LSTM)

Мотивация LSTM: сеть должна долго помнить предысторию (контекст), какой именно - сеть выучивает самостоятельно
Вводится вектор C_t - вектор долгого контекста



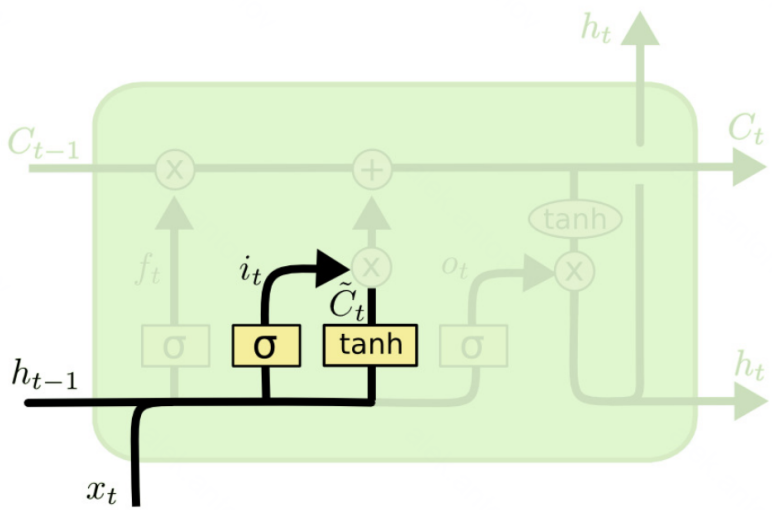
LSTM - forget gate



$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big)$$
$$i_t = \sigma\big(W_i \cdot [h_{t-1}, x_t] + b_i\big)$$
$$o_t = \sigma\big(W_o \cdot [h_{t-1}, x_t] + b_o\big)$$
$$\tilde{C}_t = \mathrm{th}\big(W_c \cdot [h_{t-1}, x_t] + b_c\big)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$h_t = o_t \odot \mathrm{th}(C_t)$$

Фильтр забывания (forget gate) с параметрами W_f, b_f решает какие координаты вектора контекста C_{t-1} надо запомнить.

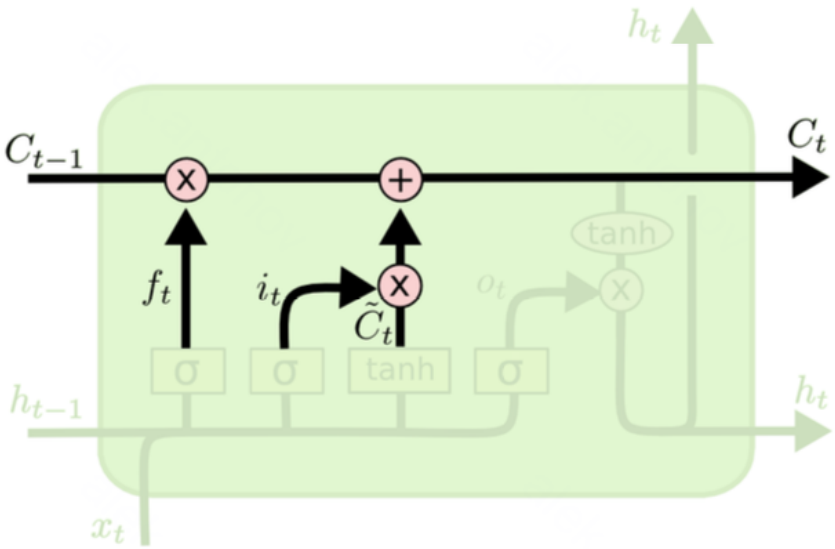
LSTM - input gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)$$
$$\tilde{C}_t = \text{th}\left(W_c \cdot [h_{t-1}, x_t] + b_c\right)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$h_t = o_t \odot \text{th}(C_t)$$

Фильтр входных данных (input gate) с параметрами W_i, b_i решает какие координаты вектора контекста надо обновить. Модель нового контекста с параметрами W_c, b_c формирует вектор $\tilde{C}_t$ с информацией о новом контексте.

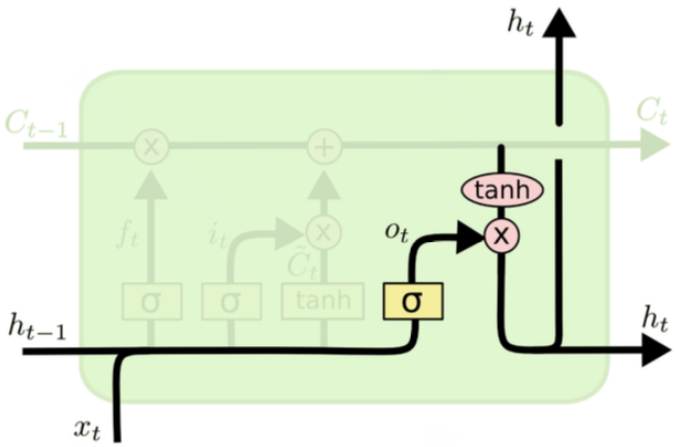
LSTM - новый вектор контекста



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)$$
$$\tilde{C}_t = \mathrm{th}\left(W_c \cdot [h_{t-1}, x_t] + b_c\right)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$h_t = o_t \odot \mathrm{th}(C_t)$$

Новый вектор контекста C_t формируется как смесь старого вектора контекста C_{t-1} с фильтром f_t и нового вектора контекста $\tilde{C}_t$ с фильтром i_t. Настраиваемых параметров нет.

LSTM - output gate

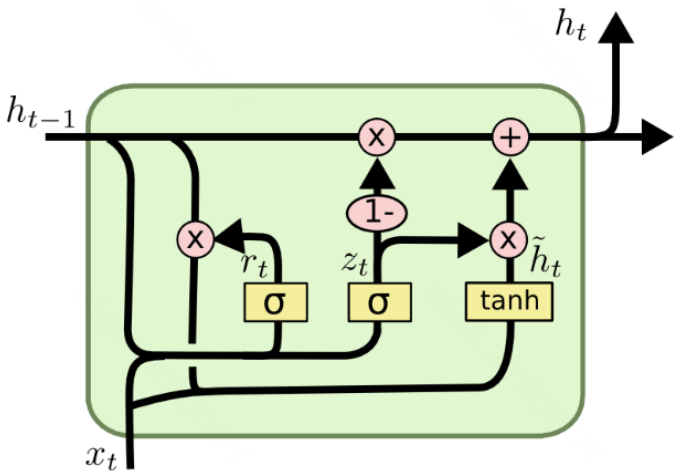


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)$$
$$\tilde{C}_t = \text{th}\left(W_c \cdot [h_{t-1}, x_t] + b_c\right)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$h_t = o_t \odot \text{th}(C_t)$$

Фильтр выходных данных (output gate) с параметрами W_o, b_o решает какие координаты **нового** вектора контекста C_t пойдут дальше.

Выходной сигнал h_t формируется из вектора контекста C_t с помощью нелинейного преобразования и фильтра o_t.

Gated Recurrent Unit (GRU)



$$z_t = \sigma\big(W_z \cdot [h_{t-1}, x_t] + b_z\big)$$
$$r_t = \sigma\big(W_r \cdot [h_{t-1}, x_t] + b_r\big)$$
$$\tilde{h}_t = \text{th}\big(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h\big)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Используется только состояние h_t, вектор контекста C_t не вводится.

Фильтр обновления (update gate) вместо входного и забывающего.

Фильтр перезагрузки (reset gate) r_t решает какую часть памяти нужно перенести дальше с прошлого шага.

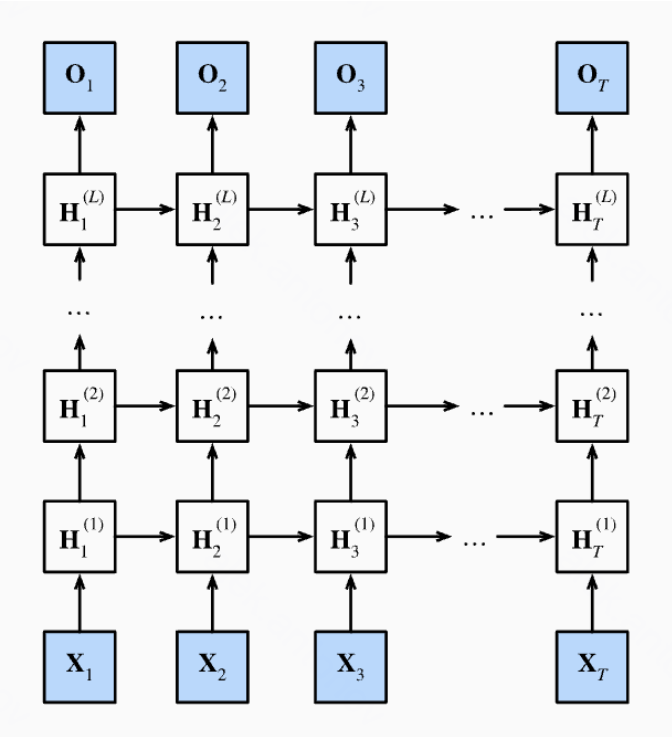
Как LSTM и GRU решают проблему?

Теперь h_t очень сложно (нелинейными преобразованиями) зависит от h_{t-1}, но вместо этого появляется $\frac{\partial C_t}{\partial C_{t-1}} = f_t$.
f_t - это сигмоида, которая всё ещё может быть около нуля.
Однако, **этого можно избежать**: инициализировать b_f большими значениями. Это будет гарантировать близкое к единице значение прозводной по крайней мере на первых итерациях.
Поскольку в самом начале обучения у нас выучиваются полезные вещи, то и далее будет происходить также.

Table of Contents

Deep RNN

Идея в подачи выходов одной рекуррентной сети на вход другой. Обычно используется до 5 слоёв.



Deep RNN

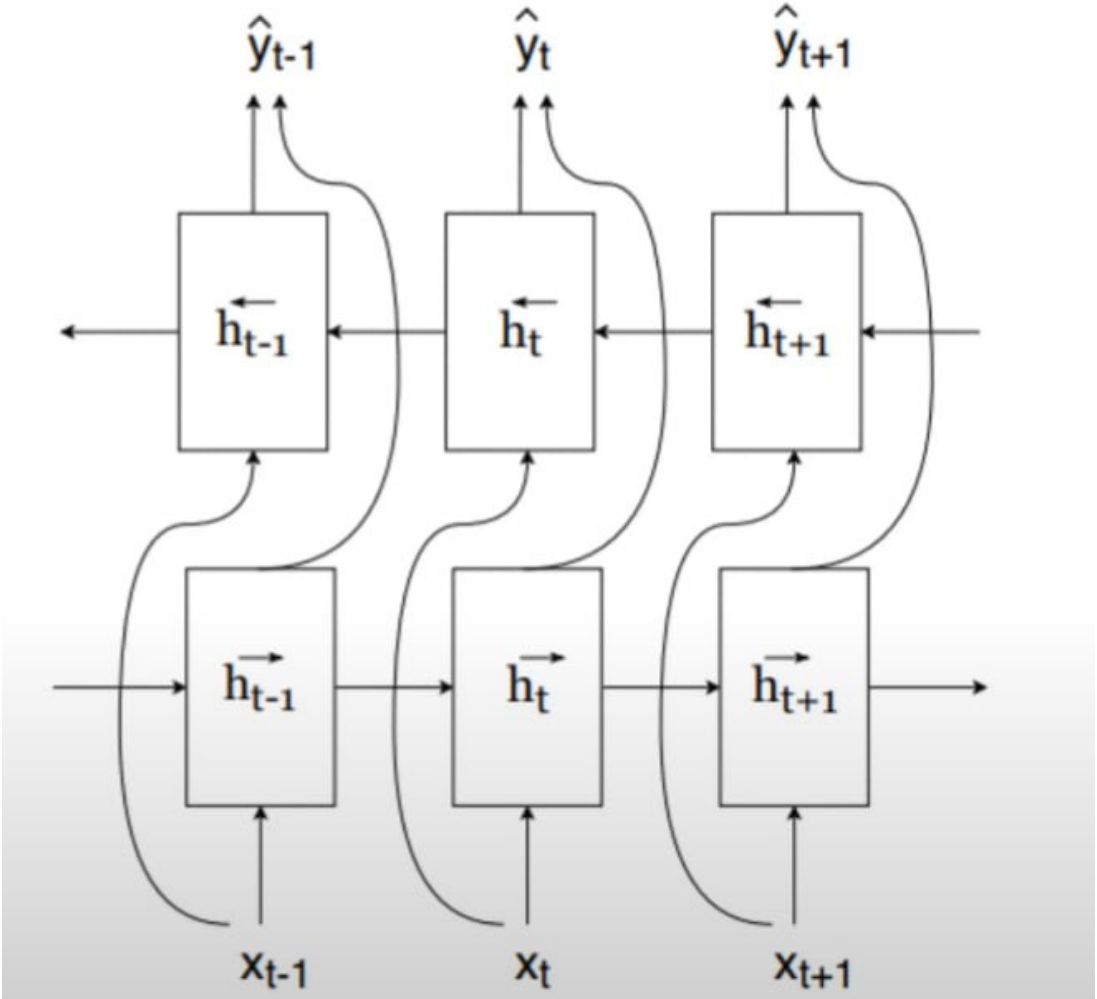
$$H_t^{(1)} = GRU(H_{t-1}^{(1)}, x_t)$$

$$H_t^{(2)} = GRU(H_{t-1}^{(2)}, H_t^{(1)})$$

$$H_t^{(3)} = GRU(H_{t-1}^{(3)}, H_t^{(2)})$$

$$\hat{y}_t = g(UH_t^{(3)} + \hat{b})$$

Bidirectional GRU (BiGRU)



$$\vec{h_t} = \vec{GRU}(\vec{h_{t-1}}, x_t)$$

$$\overleftarrow{h_t} = \overleftarrow{GRU}(\overleftarrow{h_{t+1}}, x_t)$$

$$h_t = BiGRU_t(x, \vec{h}, \overleftarrow{h}) = \left[\vec{h_t}, \overleftarrow{h_t}\right]$$

$$\hat{y}_t = g(Uh_t + \hat{b})$$

Table of Contents

Резюме и дополнительные сведения

- Рекуррентные сети - обучаемые преобразования входной последовательности в выходную (seq2seq)
- Приёмы, сделавшие возможным глубокое обучение:
  - продвинутые градиентные методы ускоряют сходимость
  - skip connections позволяет ещё сильнее упростить архитектуру LSTM
- Переход от feature engineering к architecture engineering
- Подбор архитектуры и гиперпараметров всё ещё искусство
- Иерархические рекуррентные сети используются для посимвольных последовательностей