

# Рекуррентные нейронные сети

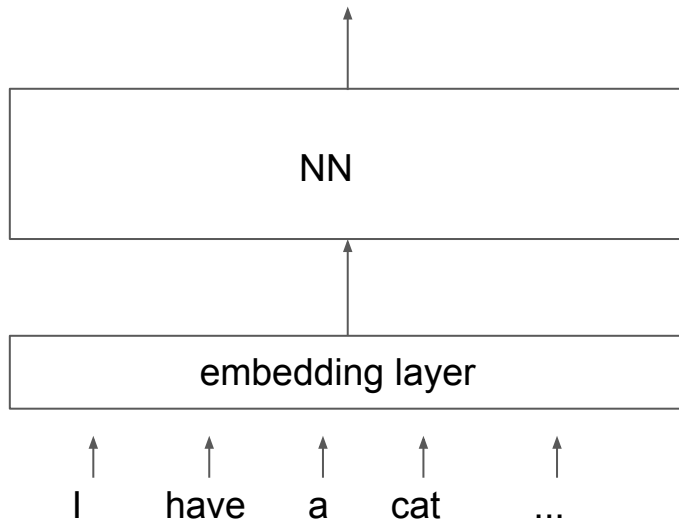
# Классификация текстов

# Задача классификации

Когда имеем малое  
количество данных

Output (class,  
e.g. genre, or  
emotion)

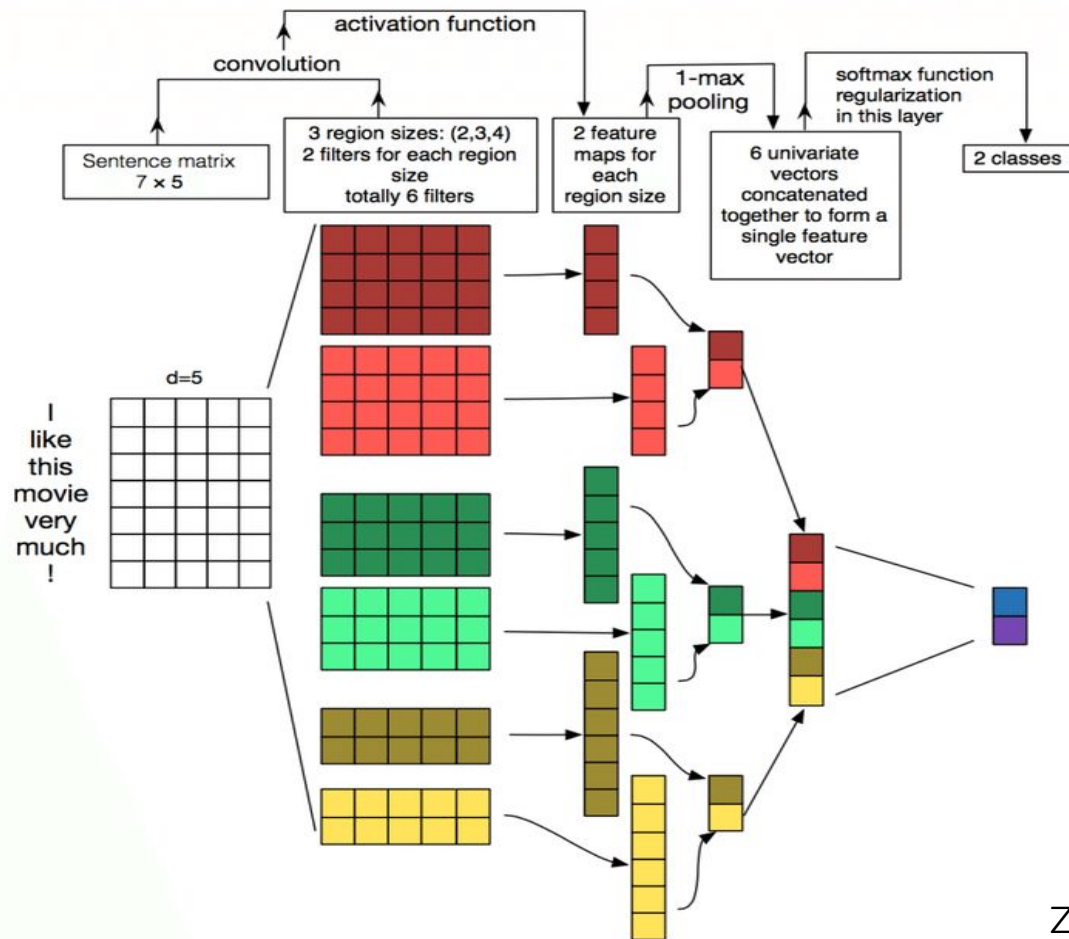
например,  
усреднить все  
эмбединги



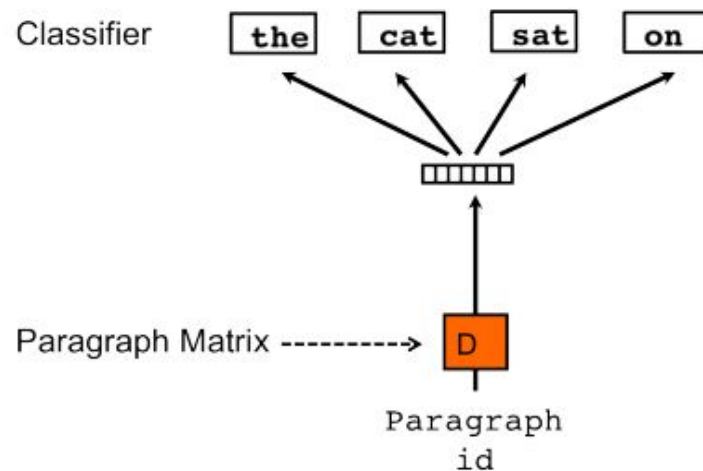
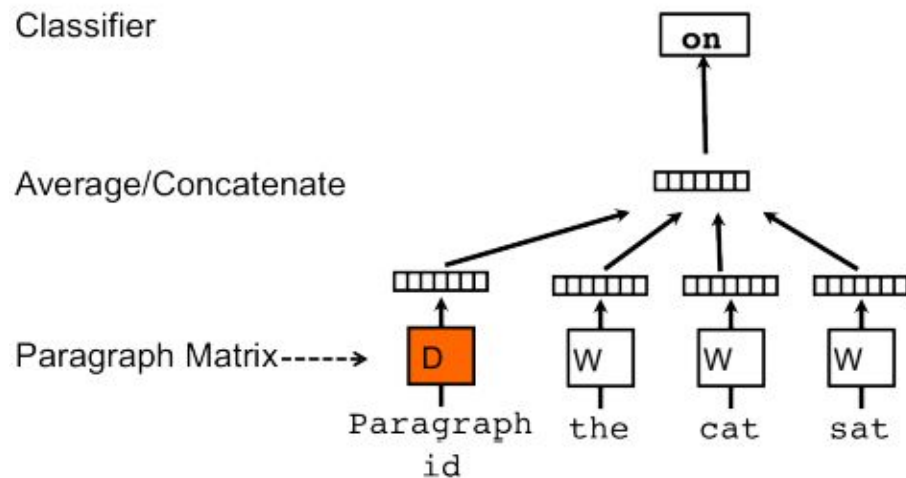
pre-trained

"I have a cat ..."

# CNN



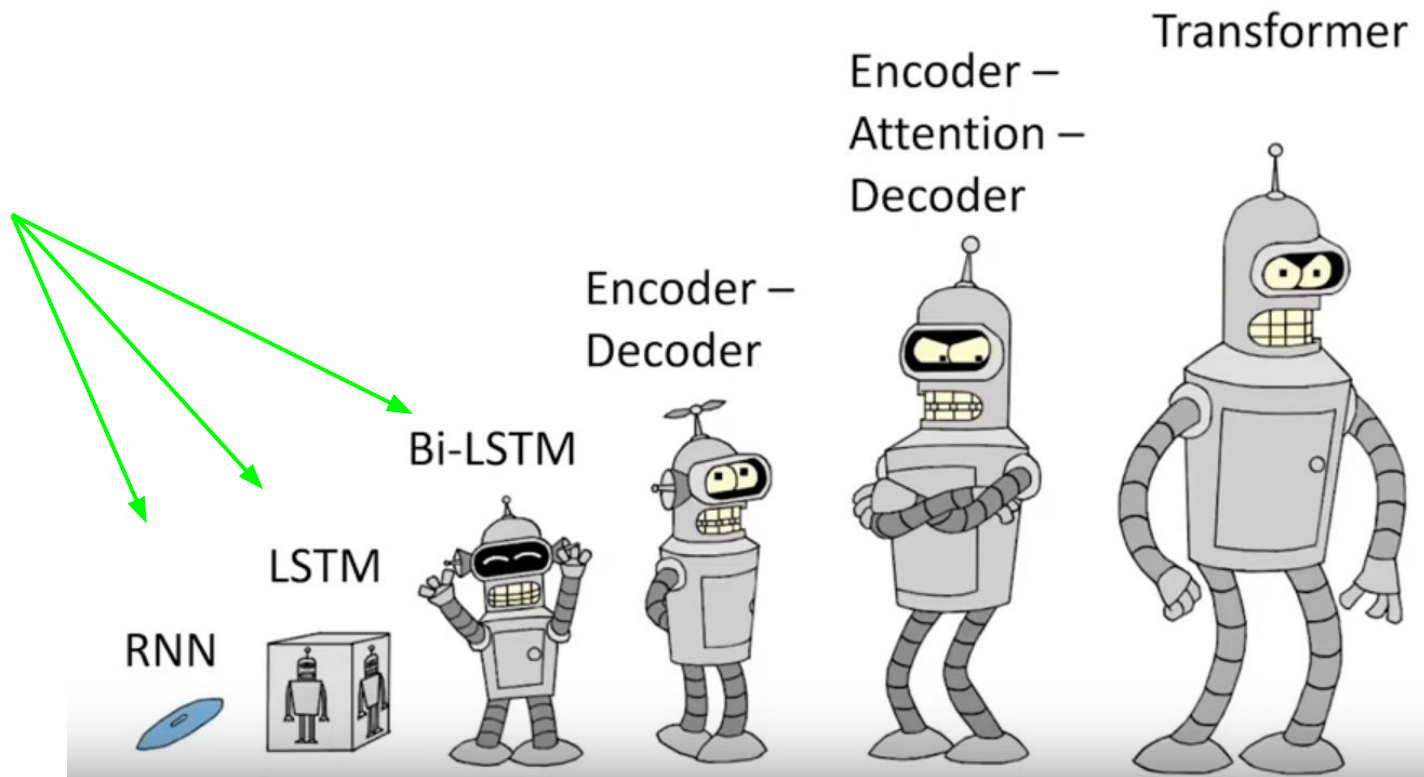
# Doc2Vec



# Классификация текстов с помощью RNN

Зачем нужен еще один тип сетей?

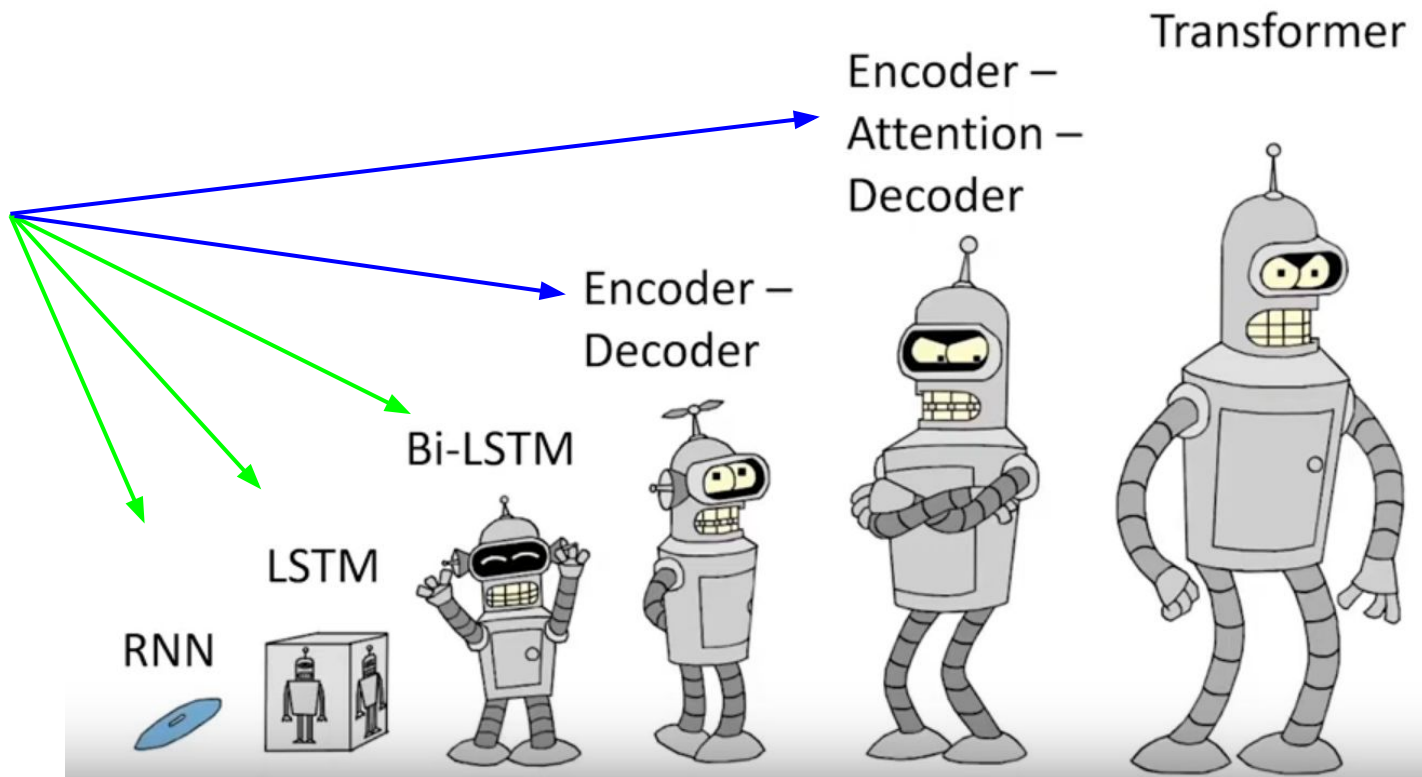
# Наш путь



Картинка из лекции  
Бурцева М. С.

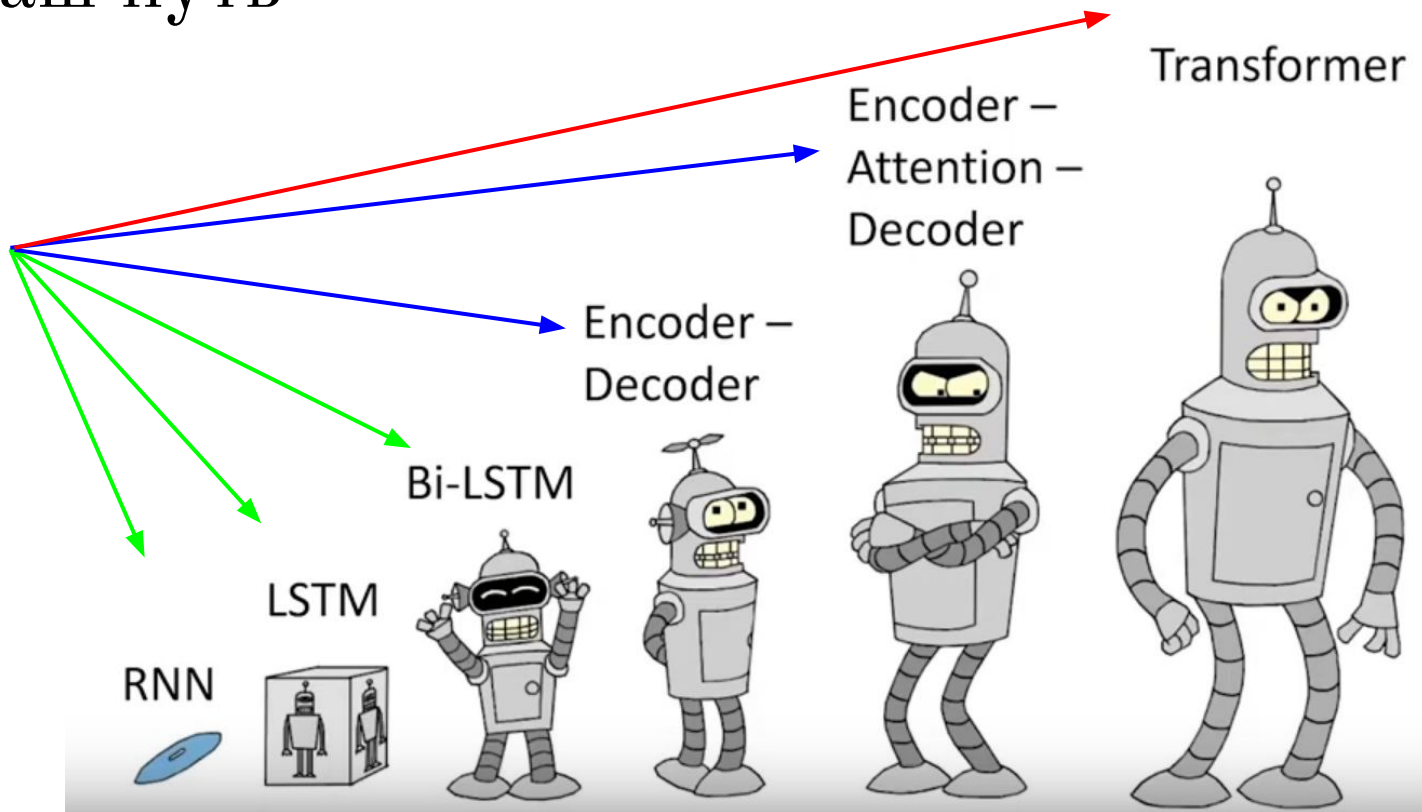


# Наш путь



Картинка из лекции  
Бурцева М. С.

# Наш путь



Картинка из лекции  
Бурцева М. С.

# Зачем нужен еще один тип сетей?

Примеры последовательностей в данных:

- Видео
- Аудио / музыка
- Текст

# Sequence tagging

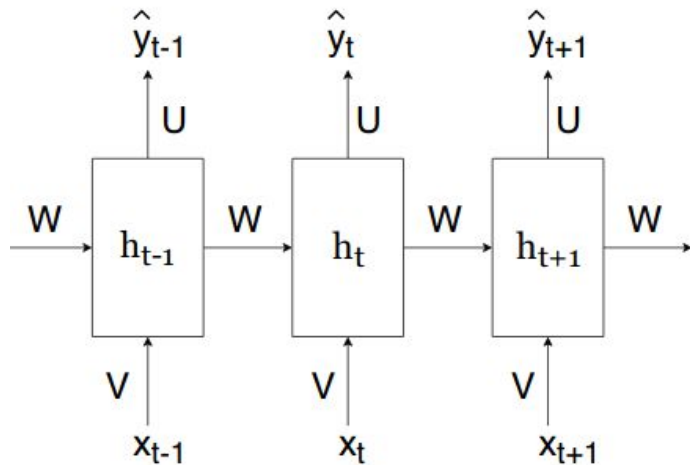
Дано:

- ▶  $D$  — множество размеченных последовательностей  $(x, y)$
- ▶  $x = \{x_1, \dots, x_n\}$  — последовательность входных объектов
- ▶  $y = \{y_1, \dots, y_n\}$  — последовательность выходных векторов
- ▶  $x_i \in X, y_i \in Y$

Необходимо: по входной последовательности предсказать элементы выходной последовательности

$$\hat{Y} = \arg \max_Y p(Y|X)$$

# RNN

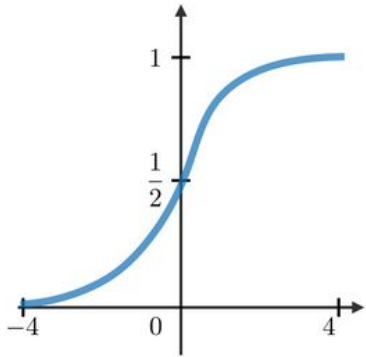
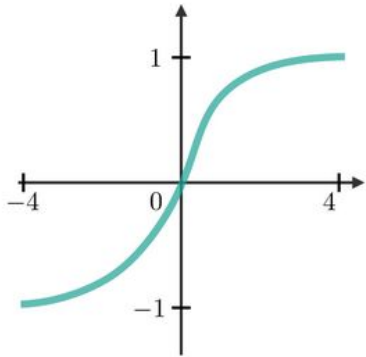
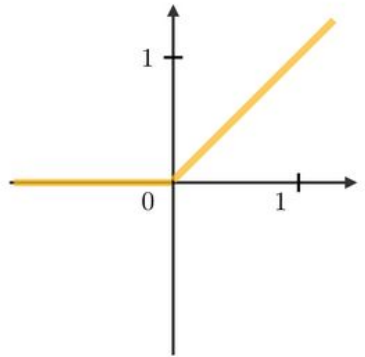


$h_t$  — скрытое состояние  
в момент  $t$

$$h_t = f(Vx_t + Wh_{t-1} + b)$$

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

# Activation functions

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

# RNN

Обучение сети — минимизация суммарных потерь:

$$\sum_{t=1}^n \mathcal{L}_t(y_t, \hat{y}_t) \rightarrow \min_{V, U, W, b, \hat{b}}$$

# Пример

$$h_t = f(Vx_t + Wh_{t-1} + b)$$

$$|h| = 10, |x| = 30$$

От какого количества обучаемых весов зависит  $h_5$ ?

От какого количества обучаемых весов зависит  $h_2$ ?



# Пример

$$h_t = f(Vx_t + Wh_{t-1} + b)$$

$$|h| = 10, |x| = 30$$

$$V \in \mathbb{R}^{10 \times 30}, W \in \mathbb{R}^{10 \times 10}, b \in \mathbb{R}^{10 \times 1}$$

# RNN: градиенты

Обучаемые параметры:  $V, U, W, b, \hat{b}$

Градиент по  $W$ :

$$\frac{d\mathcal{L}_t}{dW} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{dh_t}{dW}$$

# RNN: градиенты

Обучаемые параметры:  $V, U, W, b, \hat{b}$

Градиент по  $W$ :

$$\frac{d\mathcal{L}_t}{dW} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \boxed{\frac{dh_t}{dW}}$$

$$\boxed{\frac{dh_t}{dW}} = \sum_{k=1}^t \left( \prod_{i=k+1}^t \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

# RNN: проблемы

Взрыв градиента:

$$\prod_{i=k+1}^t \frac{\partial h_i}{\partial h_{i-1}} \rightarrow \infty$$

Затухание градиента:

$$\prod_{i=k+1}^t \frac{\partial h_i}{\partial h_{i-1}} \rightarrow 0$$

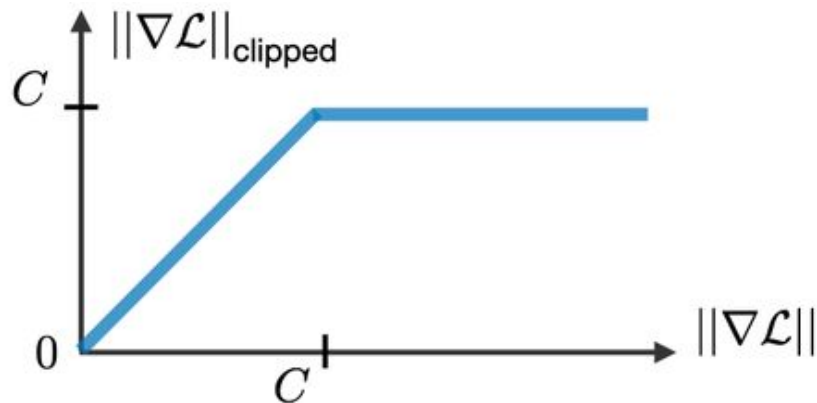
Популярные способы борьбы с взрывом/затуханием:

- ▶ Gradient clipping (против взрыва)
- ▶ Модели LSTM и GRU (против затухания)

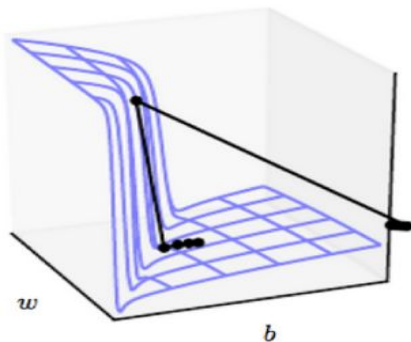
# Gradient clipping

Ограничение нормы градиента:

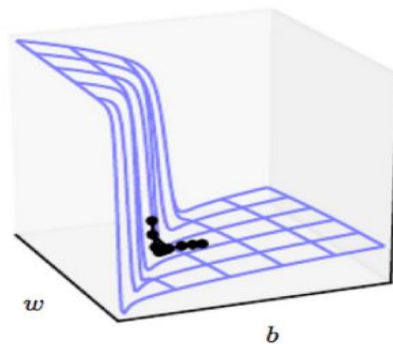
$$\text{if } \|\nabla \mathcal{L}\|_{\text{clipped}} \geq C : \\ \nabla \mathcal{L} \leftarrow \frac{C}{\|\nabla \mathcal{L}\|} \cdot \nabla \mathcal{L}$$



Without clipping

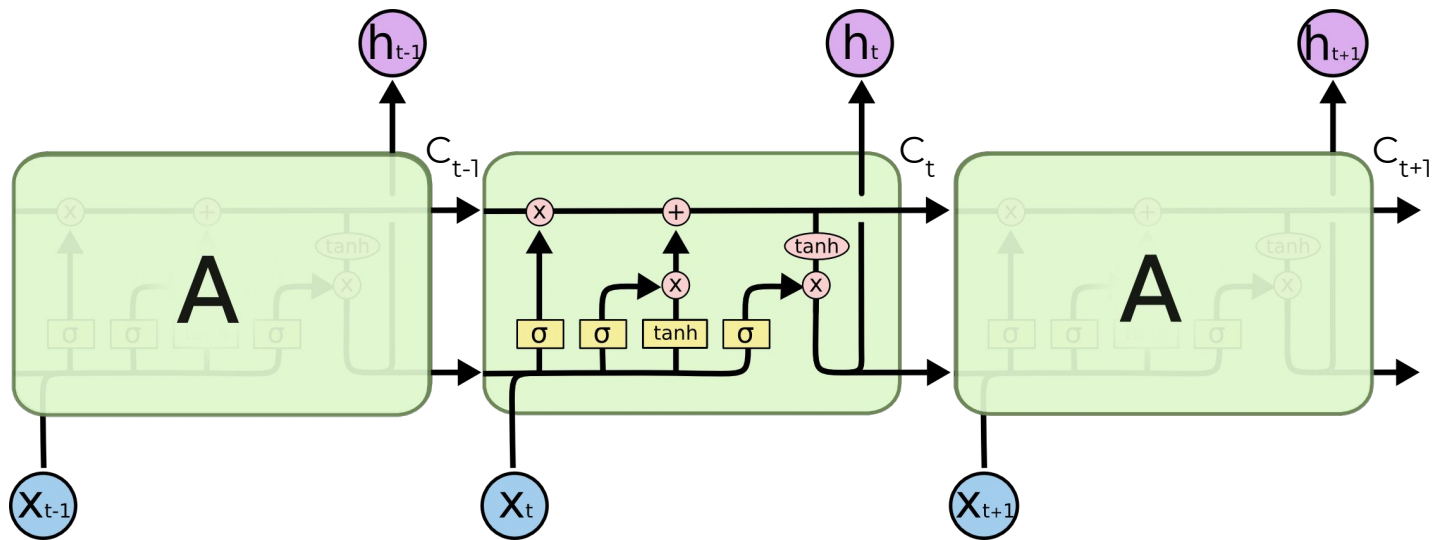


With clipping



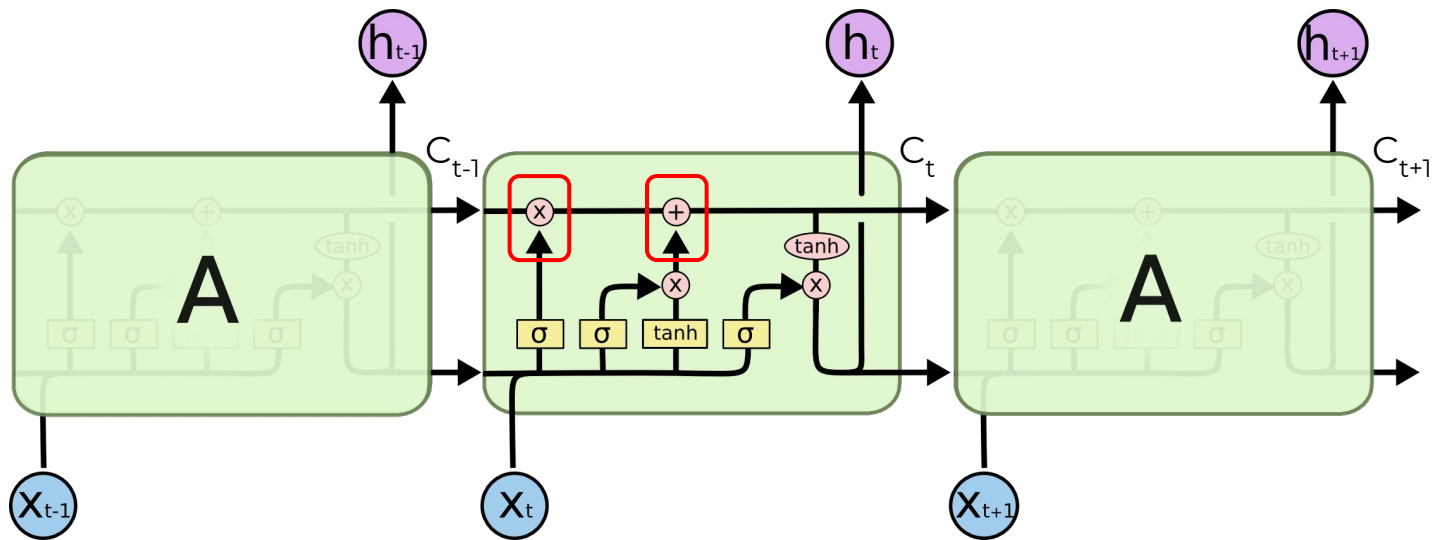
# Long Short Term Memory Cell (LSTM)

- Дополнительный путь течения информации (состояние сети,  $C_t$ )



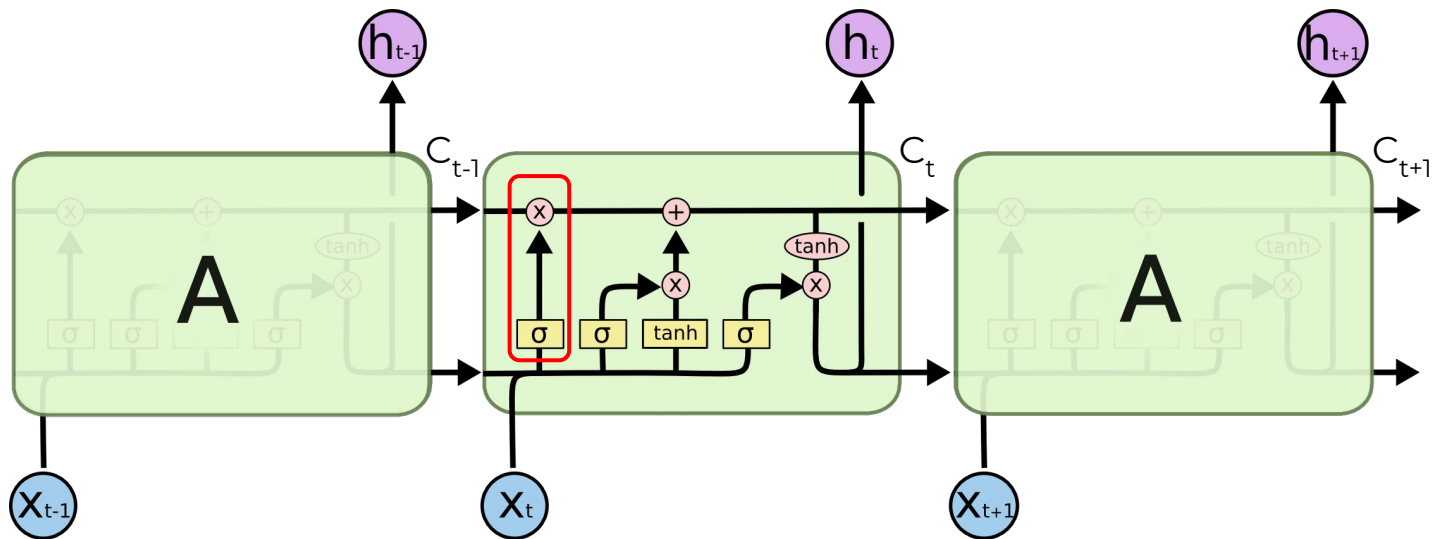
# Long Short Term Memory Cell (LSTM)

- Состояние сети обновляется в каждой клетке (очень аккуратно, с возможностью не обновляться вообще)



# Long Short Term Memory Cell (LSTM)

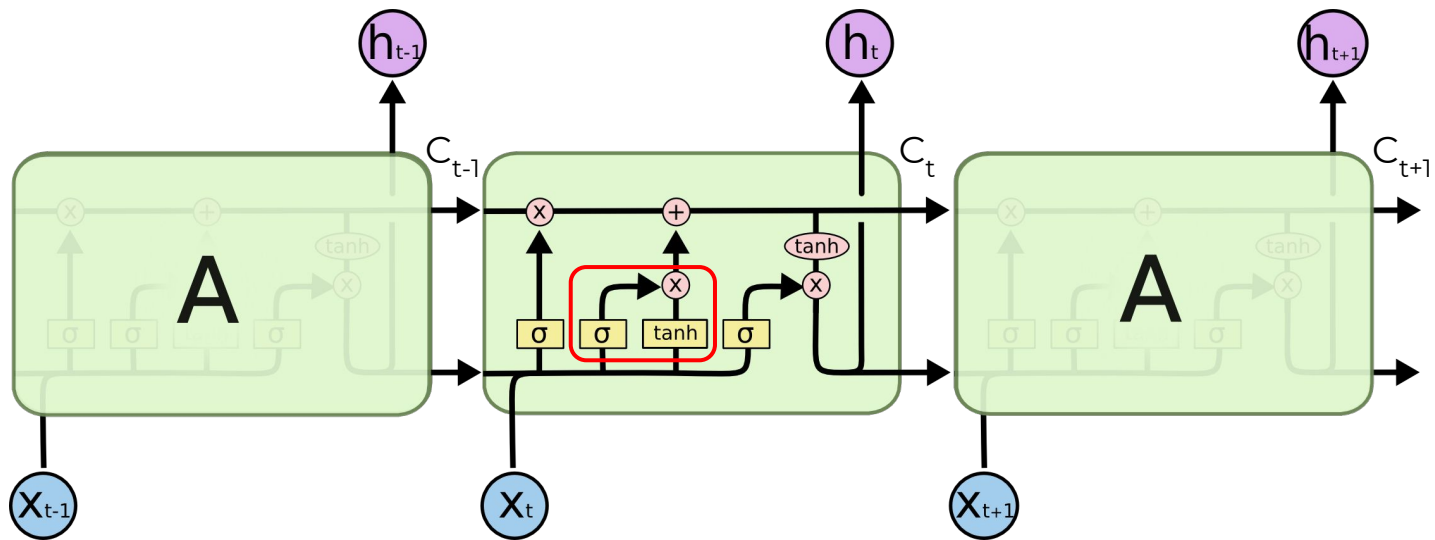
- Forget gate: часть информации забывается (умножение на результаты сигмоды)





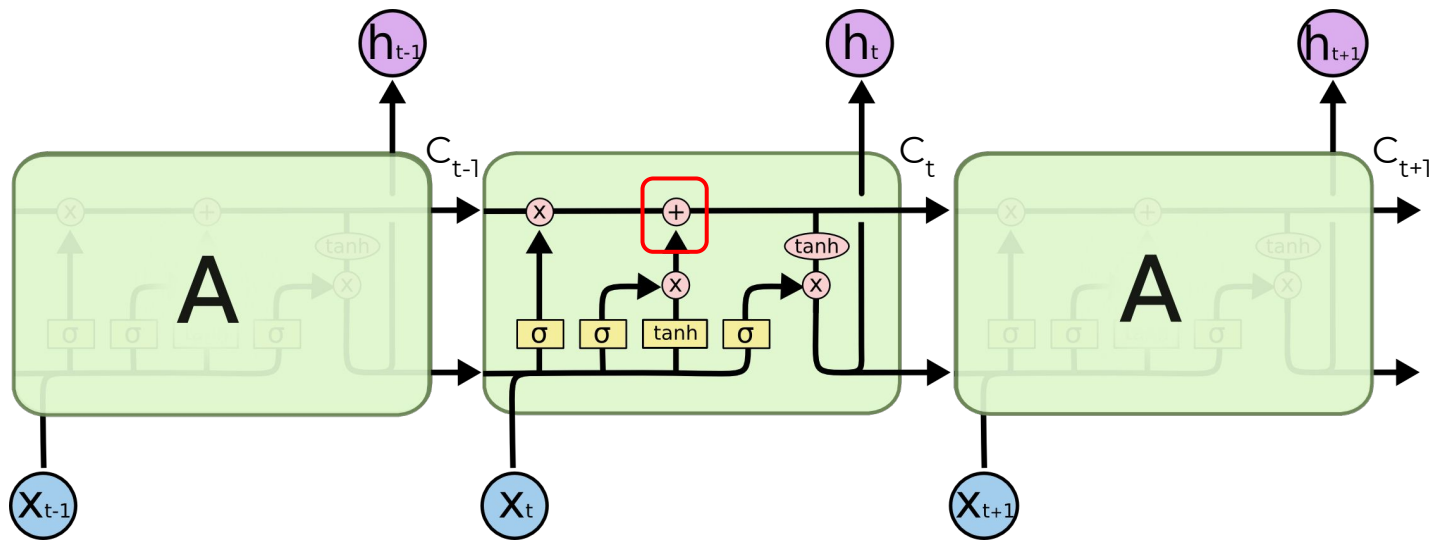
# Long Short Term Memory Cell (LSTM)

- Input gate: новая информация формируется



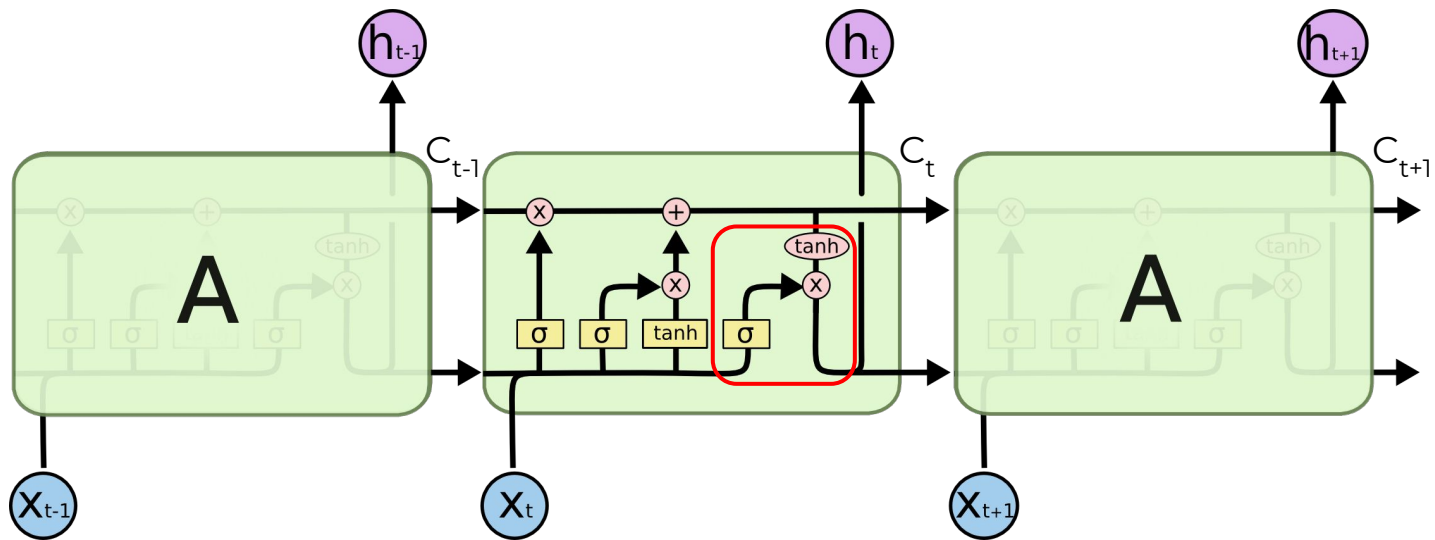
# Long Short Term Memory Cell (LSTM)

- Input gate: новая информация прибавляется к состоянию сети



# Long Short Term Memory Cell (LSTM)

- Output: новое состояние сети скрещивается с  $h_{t-1}$  и формирует  $h_t$



# Long Short Term Memory Cell (LSTM)

$$z_t = [h_{t-1}, x_t]$$

$$f_t = \sigma(W_f \cdot z_t + b_f)$$

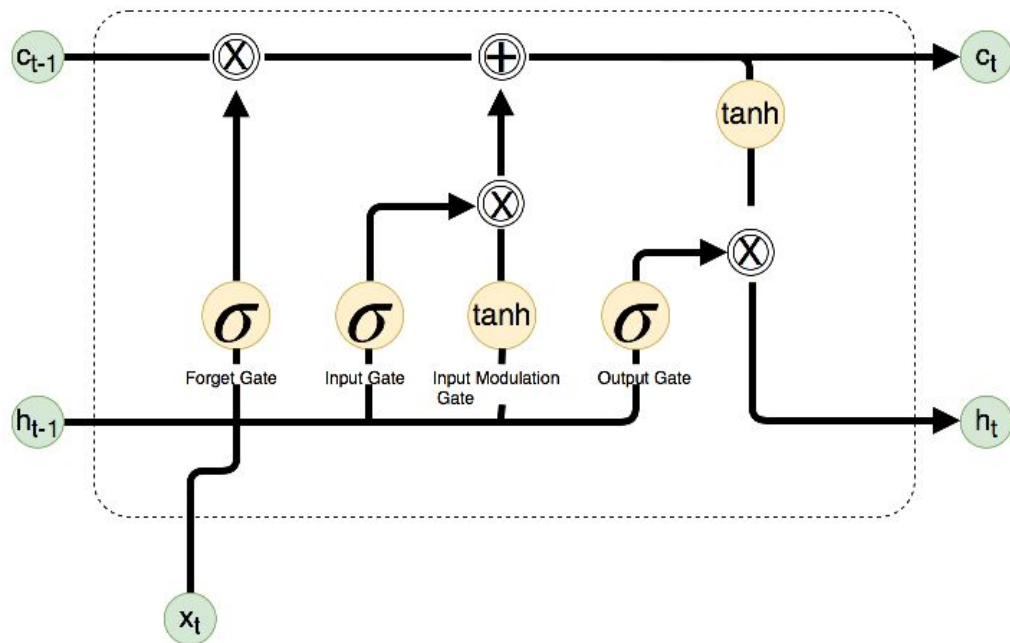
$$i_t = \sigma(W_i \cdot z_t + b_i)$$

$$\hat{C}_t = \tanh(W_c \cdot z_t + b_c)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{C}_t$$

$$o_t = \sigma(W_o \cdot z_t + b_o)$$

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



# Gated Recurrent Unit (GRU)

**Update gate:** controls what parts of hidden state are updated vs preserved

$$\mathbf{u}^{(t)} = \sigma \left( \mathbf{W}_u \mathbf{h}^{(t-1)} + \mathbf{U}_u \mathbf{x}^{(t)} + \mathbf{b}_u \right)$$

**Reset gate:** controls what parts of previous hidden state are used to compute new content

$$\mathbf{r}^{(t)} = \sigma \left( \mathbf{W}_r \mathbf{h}^{(t-1)} + \mathbf{U}_r \mathbf{x}^{(t)} + \mathbf{b}_r \right)$$

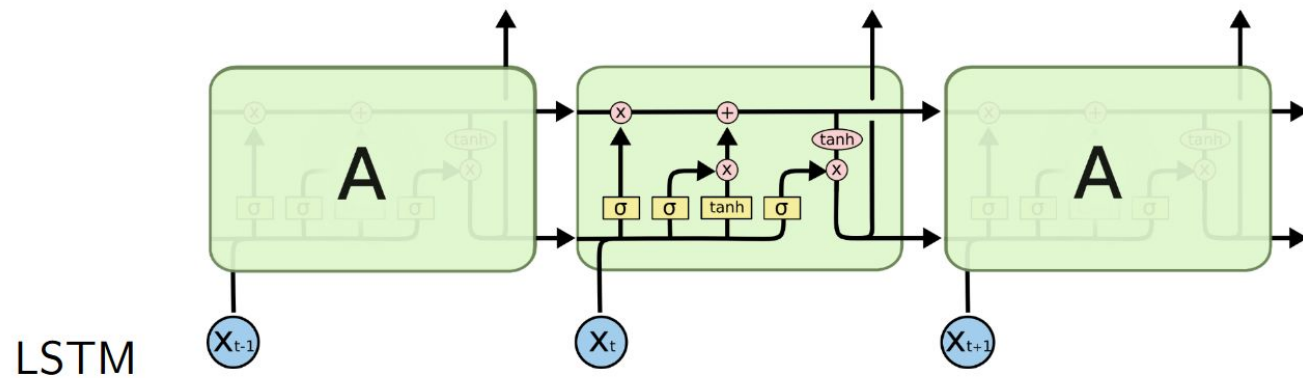
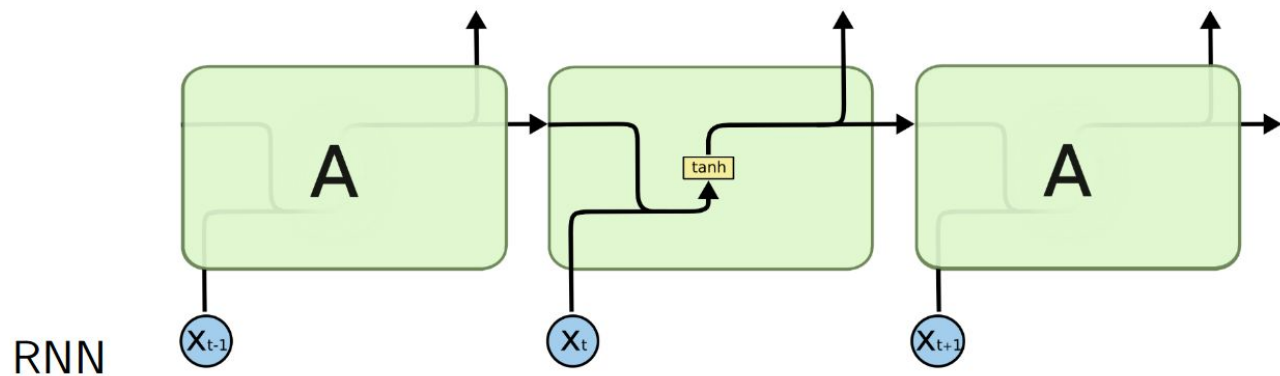
**New hidden state content:** reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

$$\tilde{\mathbf{h}}^{(t)} = \tanh \left( \mathbf{W}_h (\mathbf{r}^{(t)} \circ \mathbf{h}^{(t-1)}) + \mathbf{U}_h \mathbf{x}^{(t)} + \mathbf{b}_h \right)$$

**Hidden state:** update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$\mathbf{h}^{(t)} = (1 - \mathbf{u}^{(t)}) \circ \mathbf{h}^{(t-1)} + \mathbf{u}^{(t)} \circ \tilde{\mathbf{h}}^{(t)}$$

# LSTM и RNN



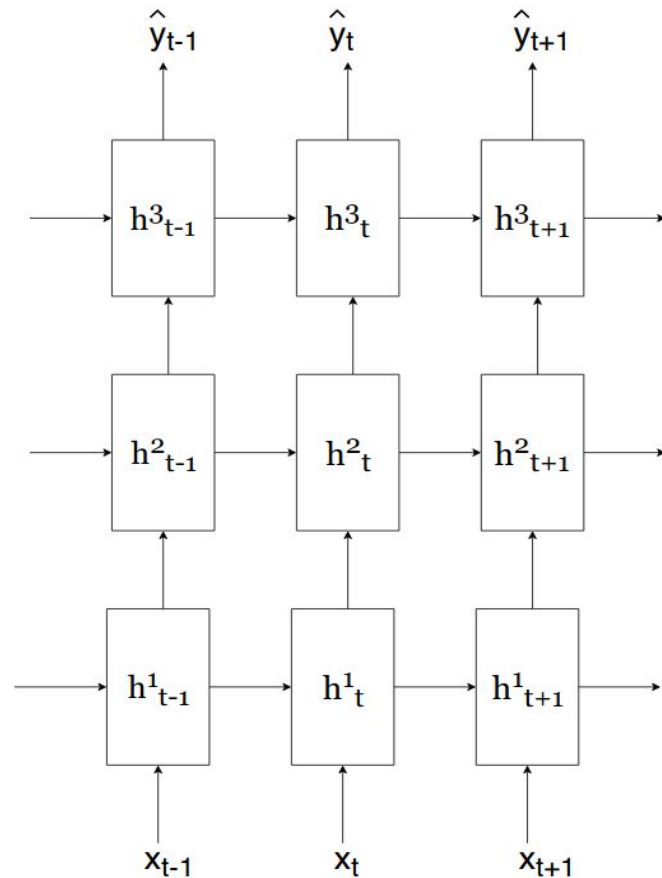
# Можно больше слоёв?

$$h_t^1, C_t^1 = LSTM(h_{t-1}^1, C_{t-1}^1, x_t)$$

$$h_t^2, C_t^2 = LSTM(h_{t-1}^2, C_{t-1}^2, h_t^1)$$

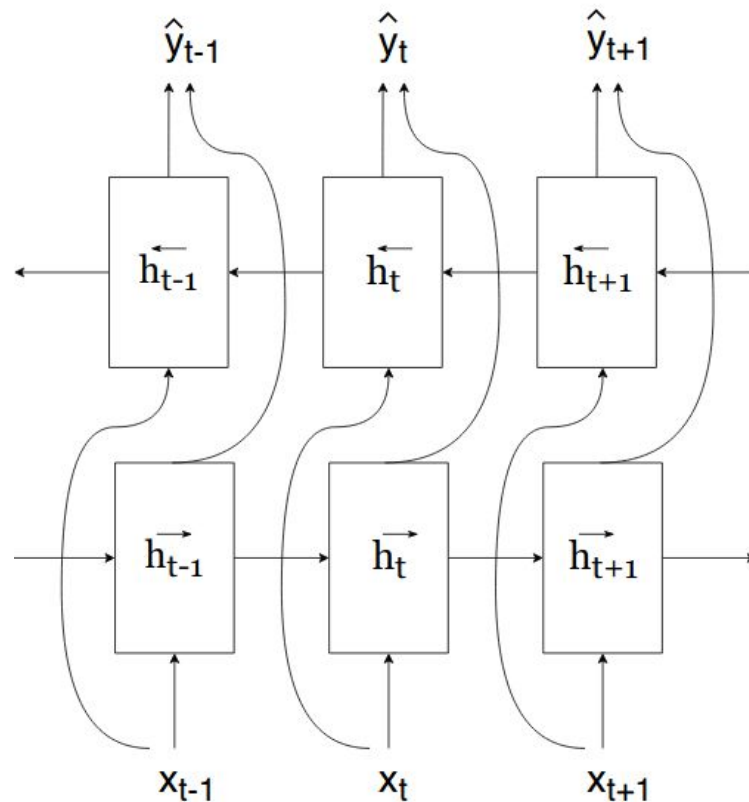
$$h_t^3, C_t^3 = LSTM(h_{t-1}^3, C_{t-1}^3, h_t^2)$$

$$y_t = g(Uh_t^2 + \hat{b})$$



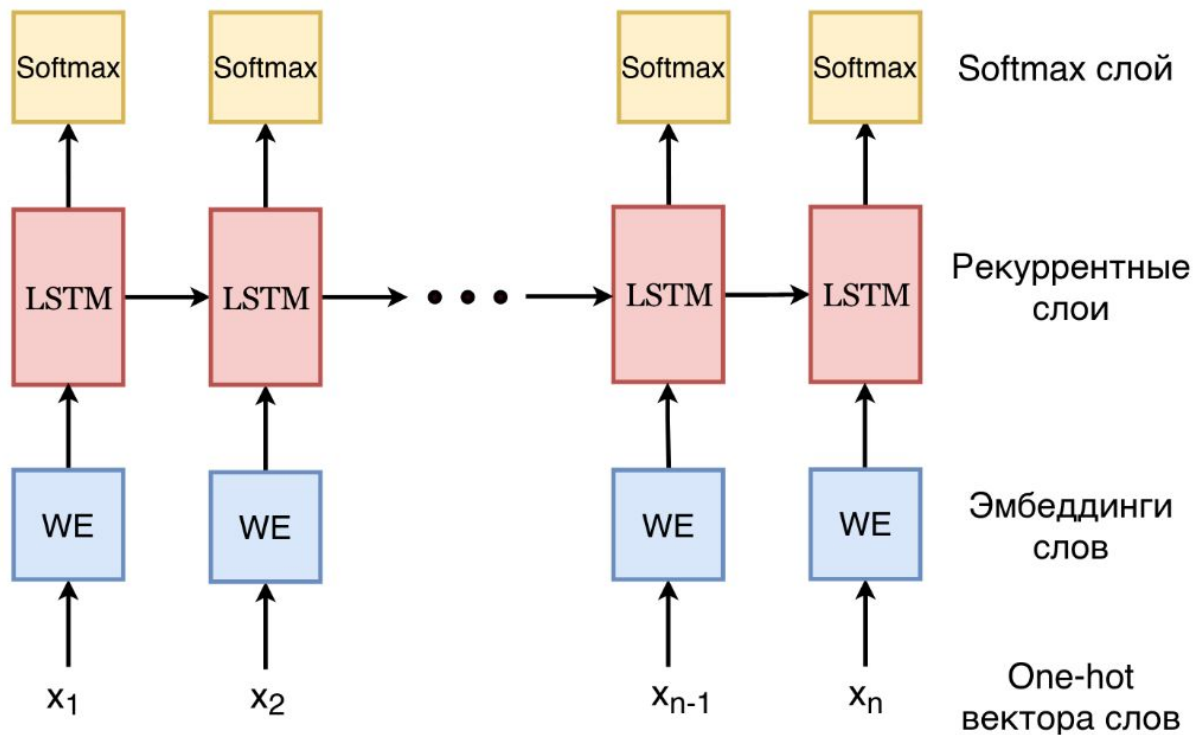
# Можно больше направлений?

$$\begin{aligned}\vec{h}_t, \vec{C}_t &= \overrightarrow{LSTM} \left( \vec{h}_{t-1}, \vec{C}_{t-1}, x_t \right) \\ \overleftarrow{h}_t, \overleftarrow{C}_t &= \overleftarrow{LSTM} \left( \overleftarrow{h}_{t+1}, \overleftarrow{C}_{t+1}, x_t \right) \\ y_t &= g \left( U \begin{bmatrix} \vec{h}_t, \overleftarrow{h}_t \end{bmatrix} + \hat{b} \right)\end{aligned}$$

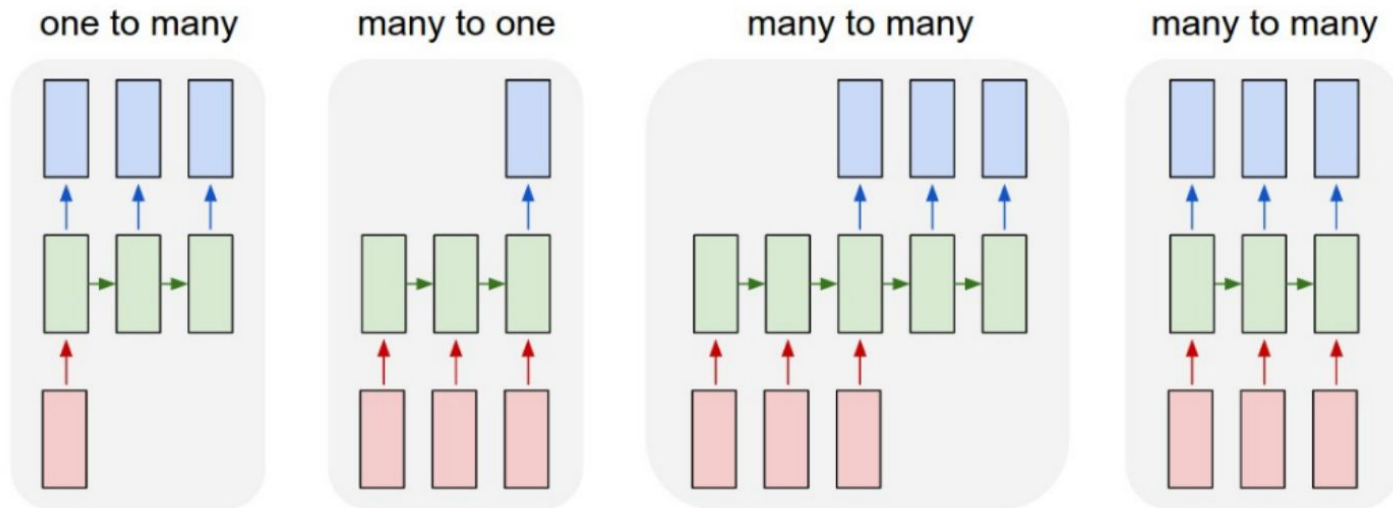




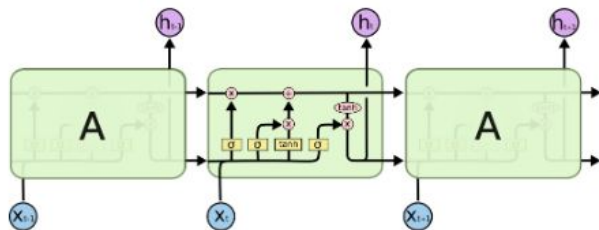
# Sequence tagging



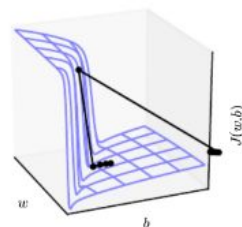
# Где? Как?



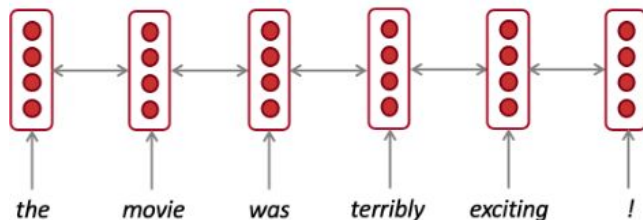
# Summary



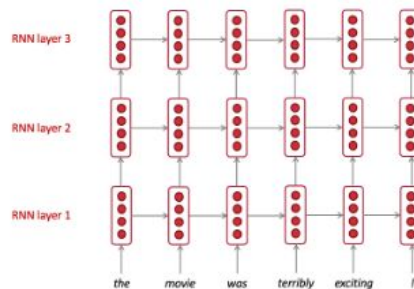
1. LSTMs are powerful but GRUs are faster



2. Clip your gradients



3. Use bidirectionality when possible



4. Multi-layer RNNs are powerful, but you might need skip/dense-connections if it's deep