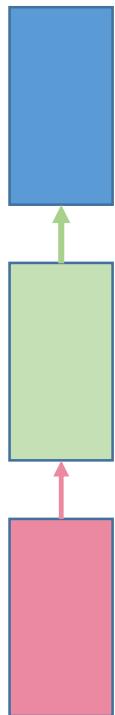


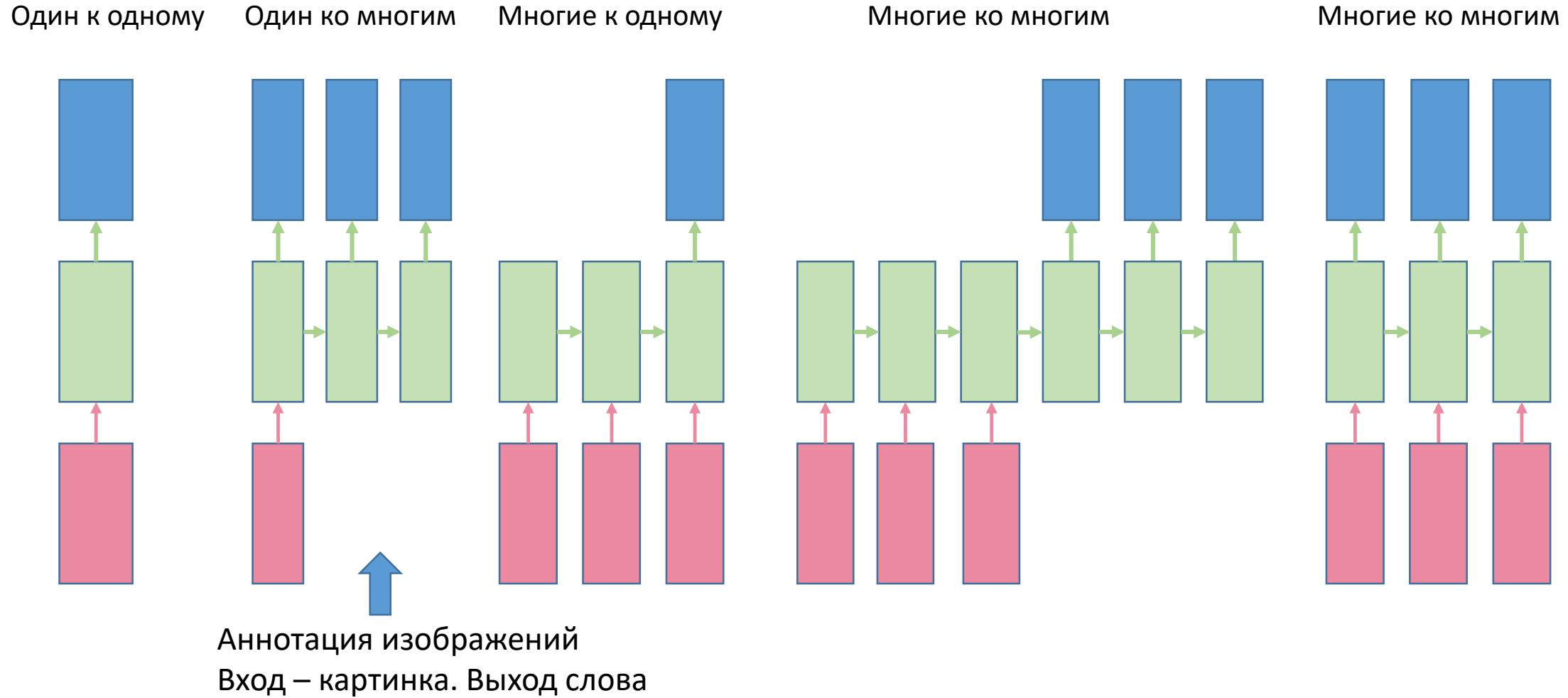
Рекуррентные нейронные сети

Сеть прямого распространения

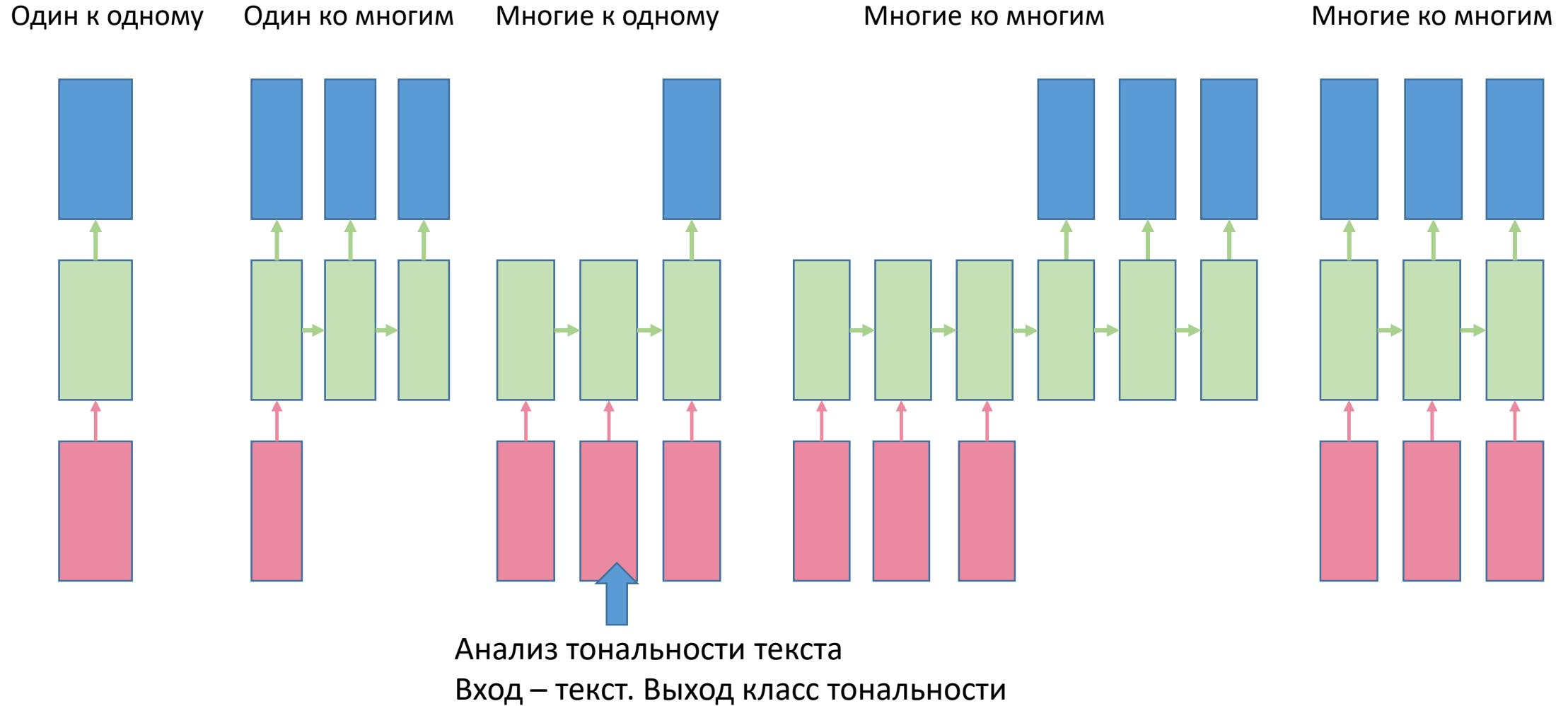
Один к одному



Рекуррентная нейросеть

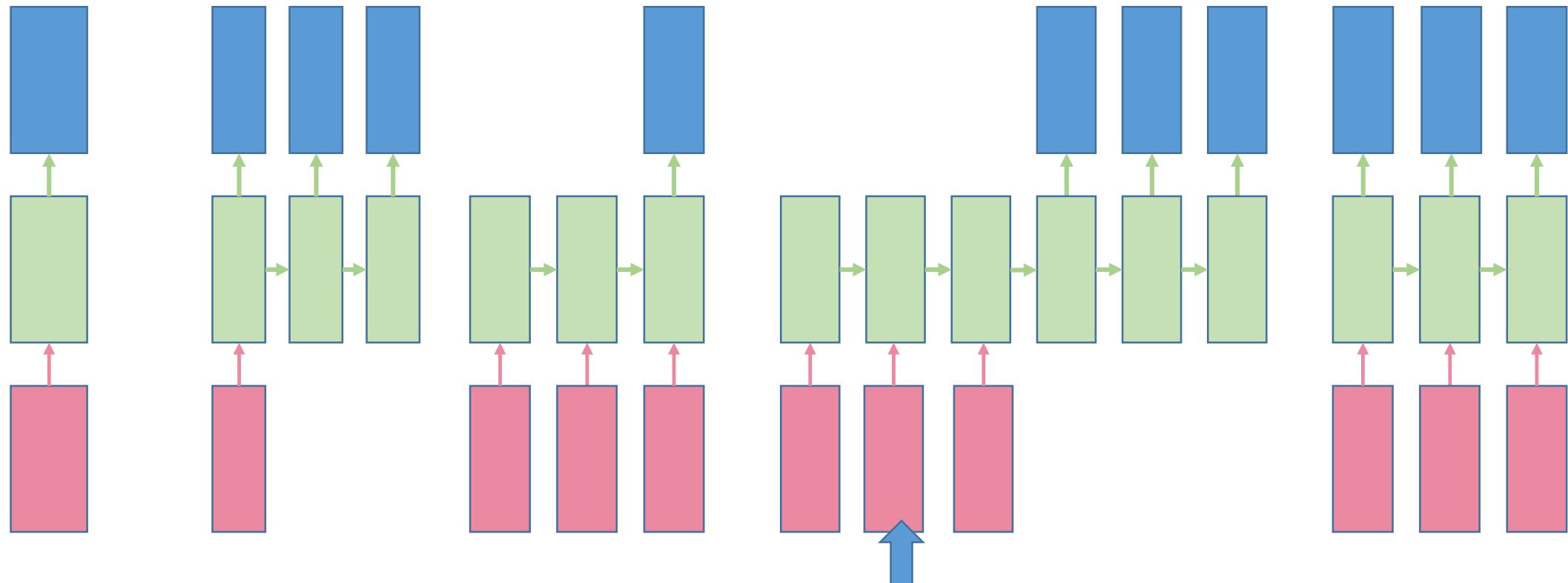


Рекуррентная нейросеть



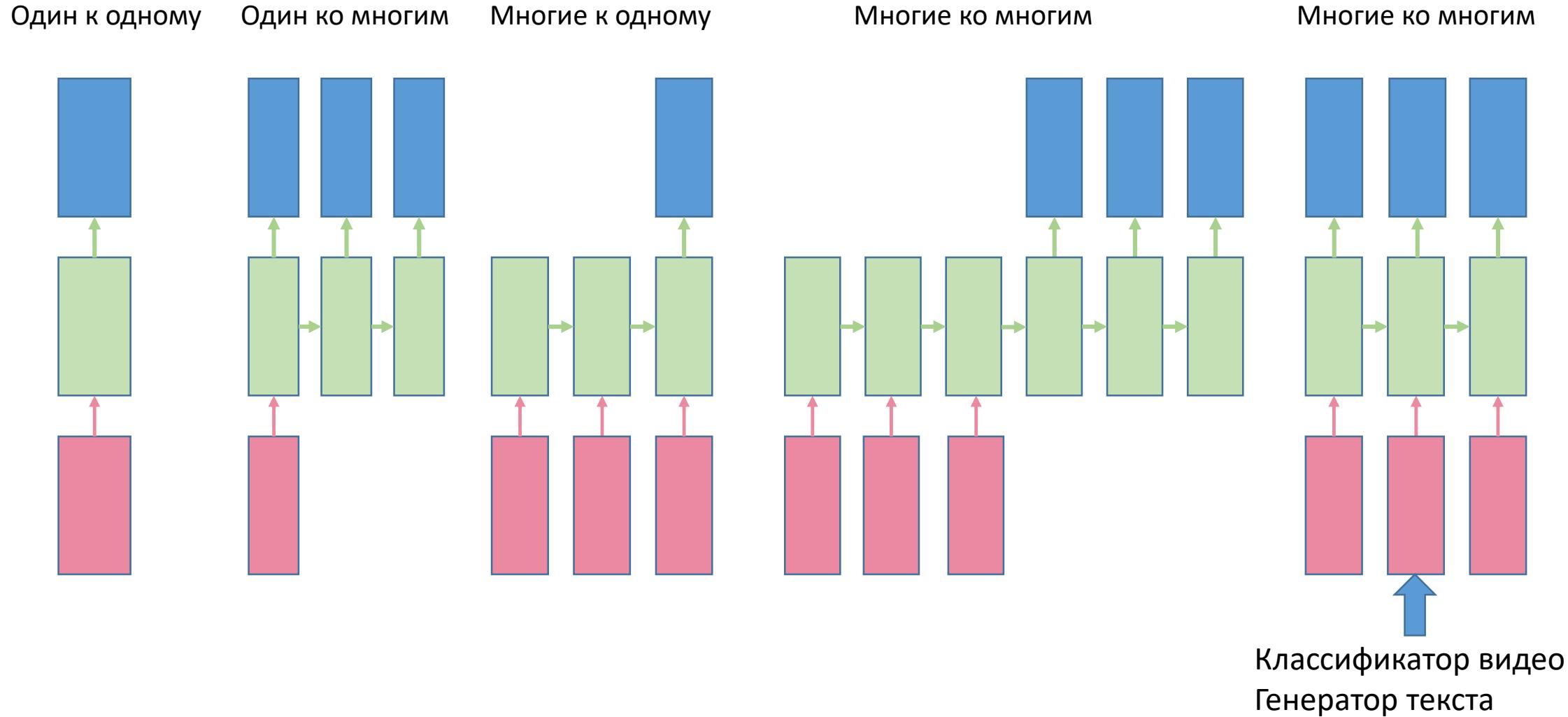
Рекуррентная нейросеть

Один к одному Один ко многим Многие к одному Многие ко многим Многие ко многим



Переводчик, чат боты
Вход – текст. Выход - текст

Рекуррентная нейросеть



Рекуррентная формула

Сеть подает на вход в момент времени t выход $t-1$

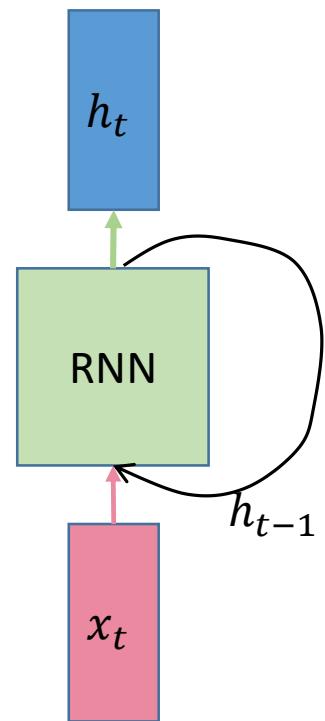
$$h_t = f_W(h_{t-1}, x_t)$$

Новое состояние

Старое состояние

Вход на шаге t

Функция с параметрами W



Рекуррентная формула

Сеть подает на вход в момент времени t выход $t-1$

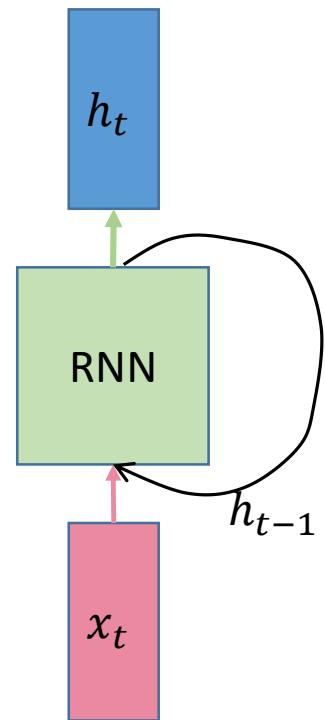
$$h_t = f_W(h_{t-1}, x_t)$$

Новое состояние

Старое состояние

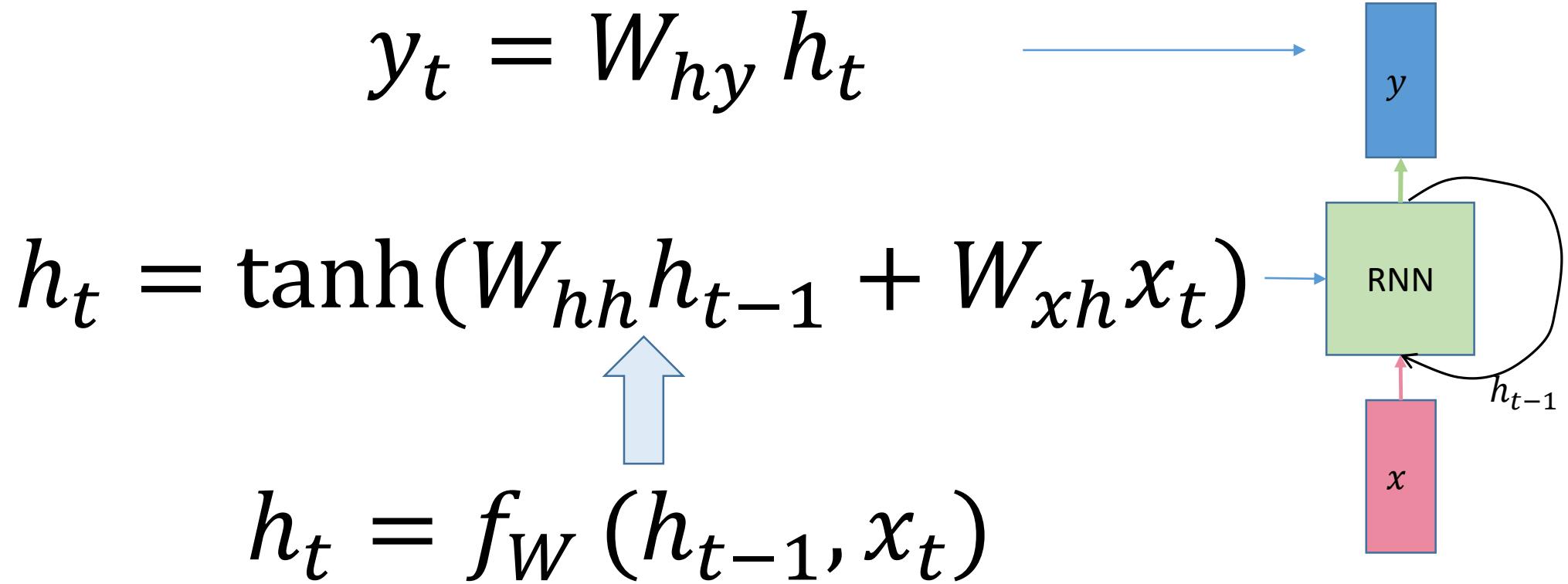
Вход на шаге t

Функция с параметрами W

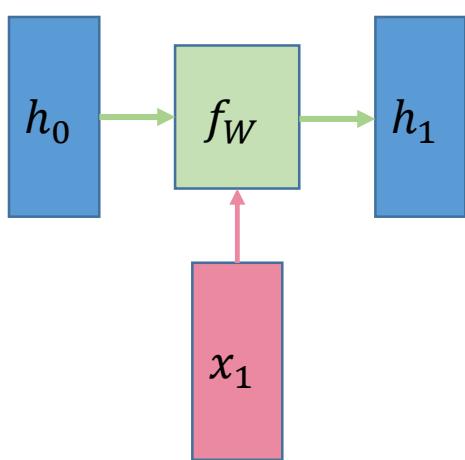


Для каждого момента времени используется
одна функция и одна матрица весов

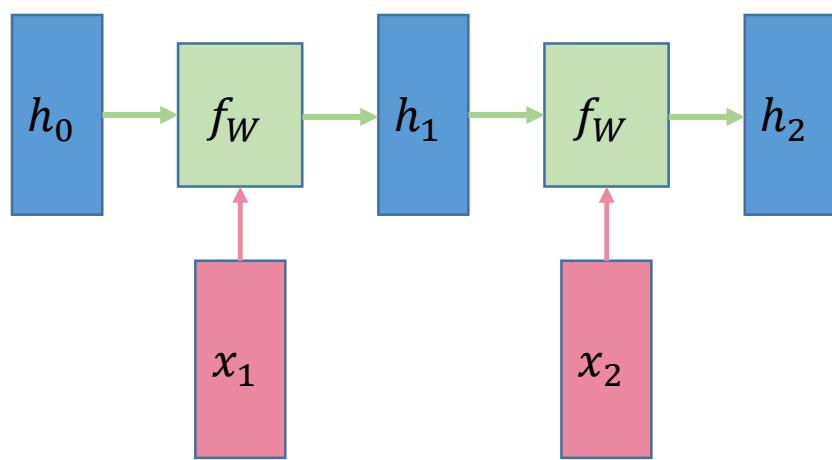
Базовая рекурнтная сеть



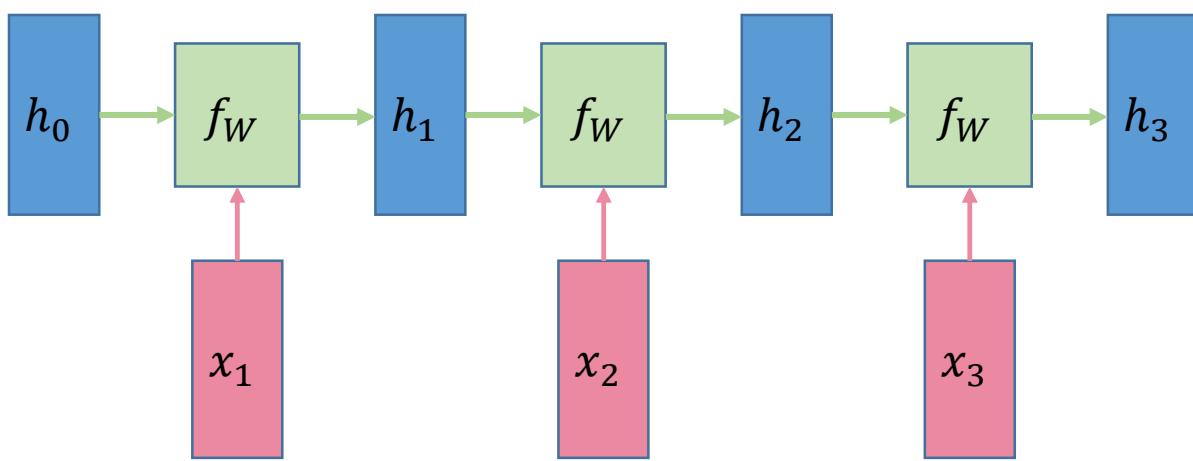
RNN вычислительный граф



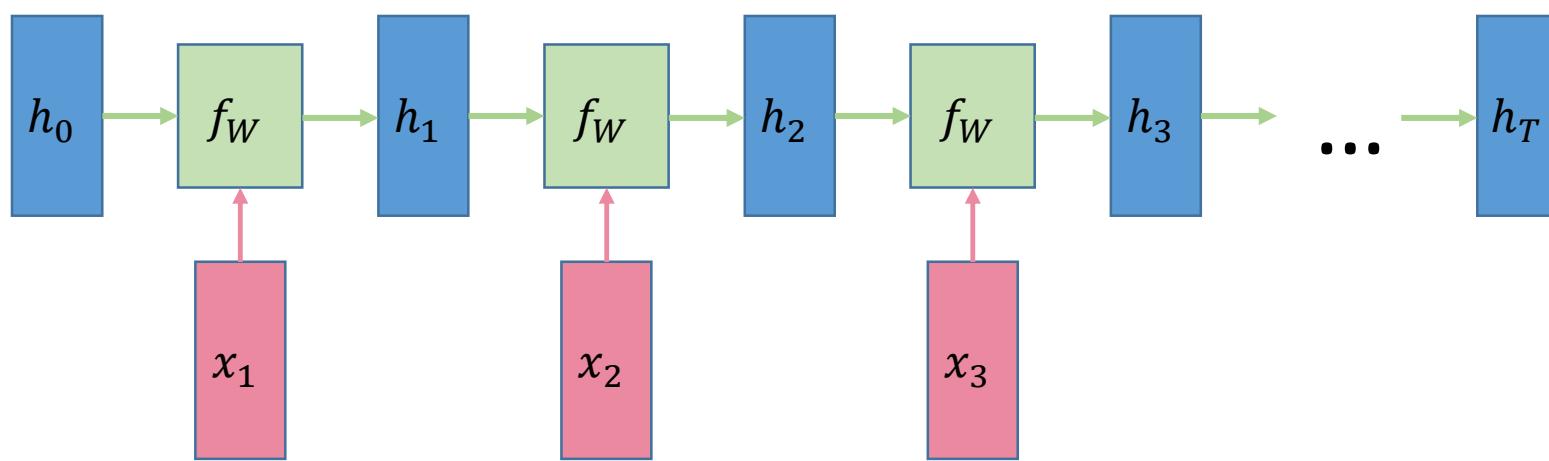
RNN вычислительный граф



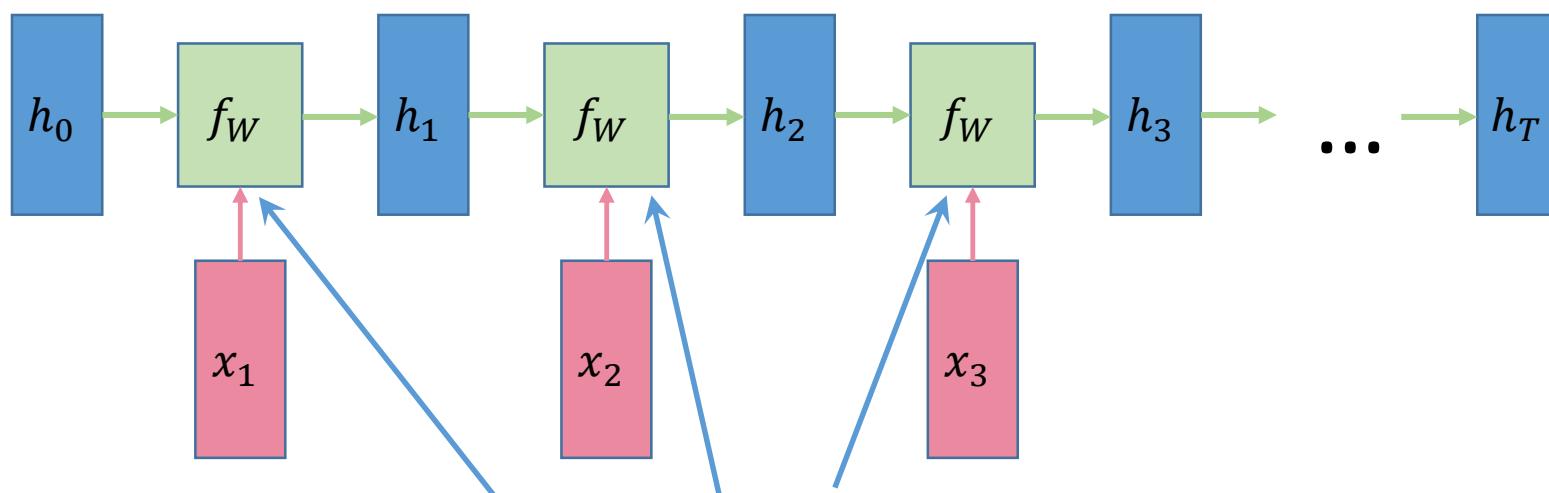
RNN вычислительный граф



RNN вычислительный граф

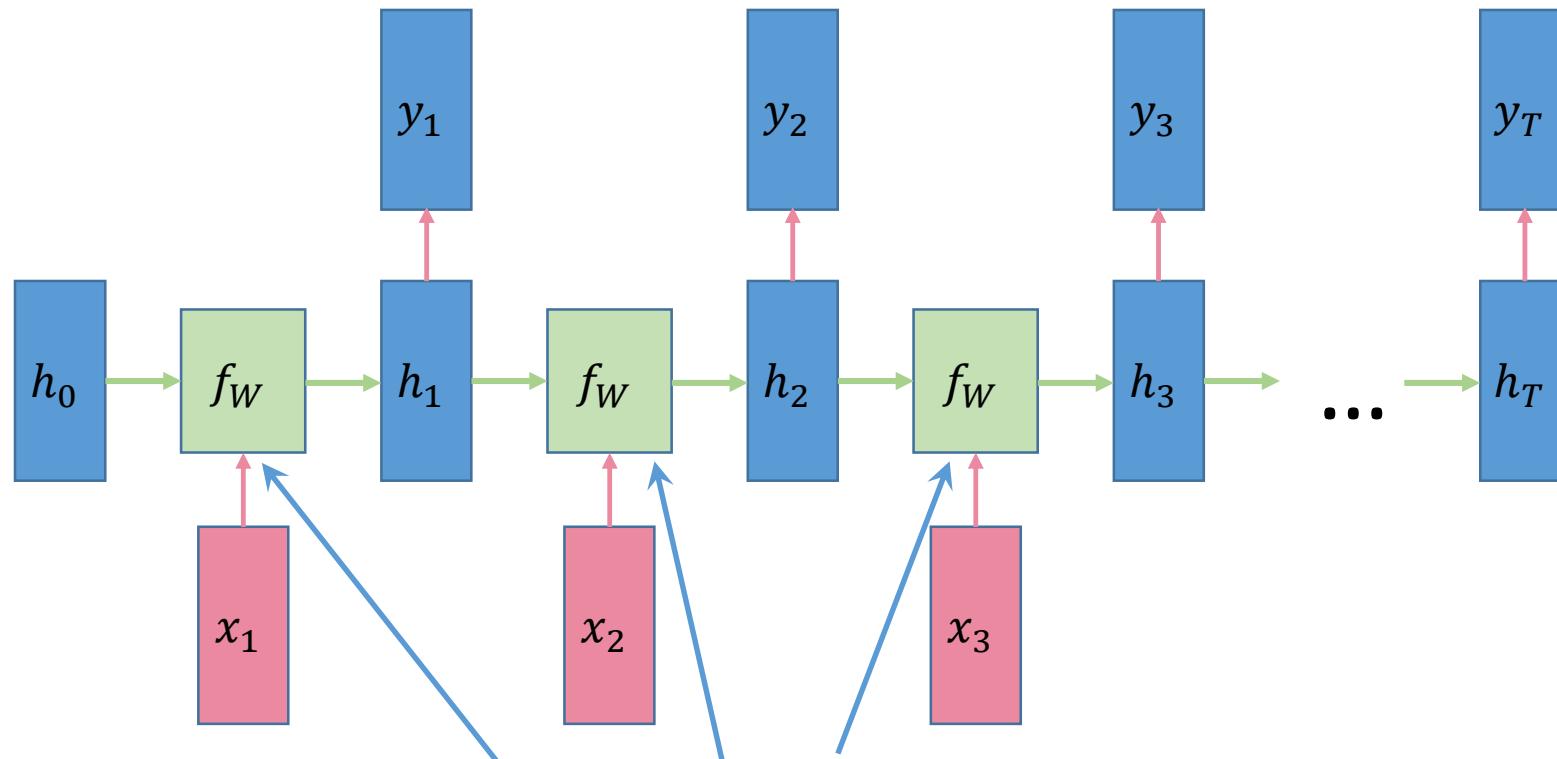


RNN вычислительный граф



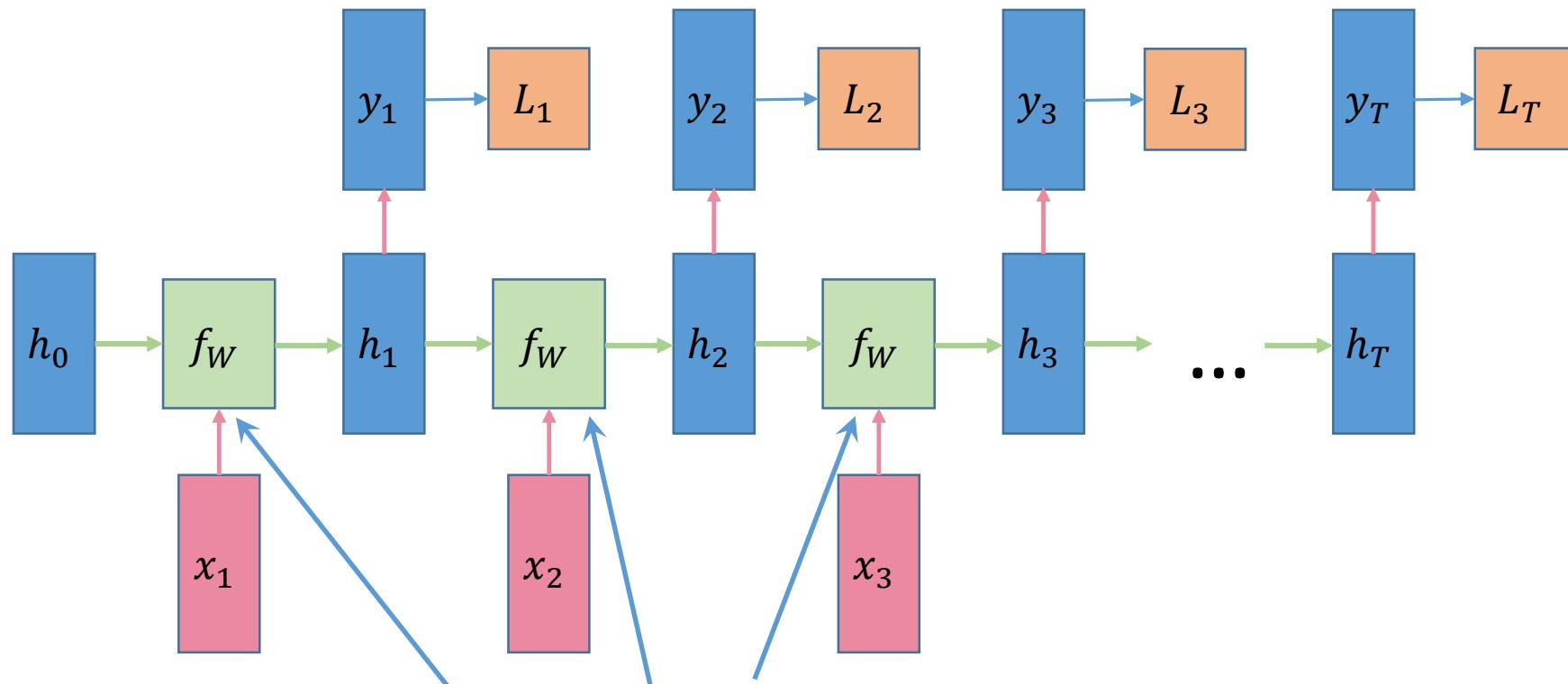
Одна матрица W на каждом шаге

Вычислительный граф. Многие ко многим



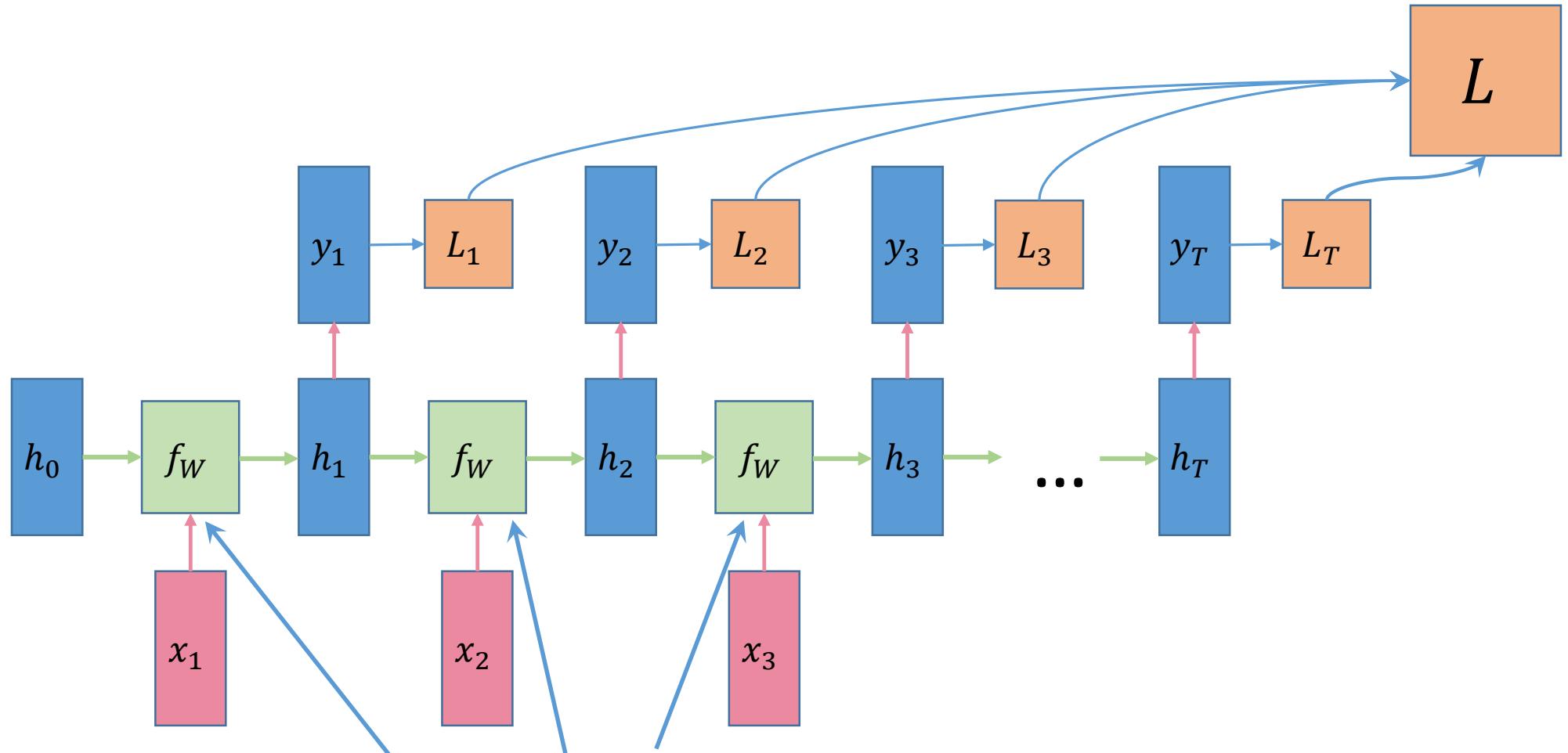
Одна матрица W на каждом шаге

Вычислительный граф. Многие ко многим



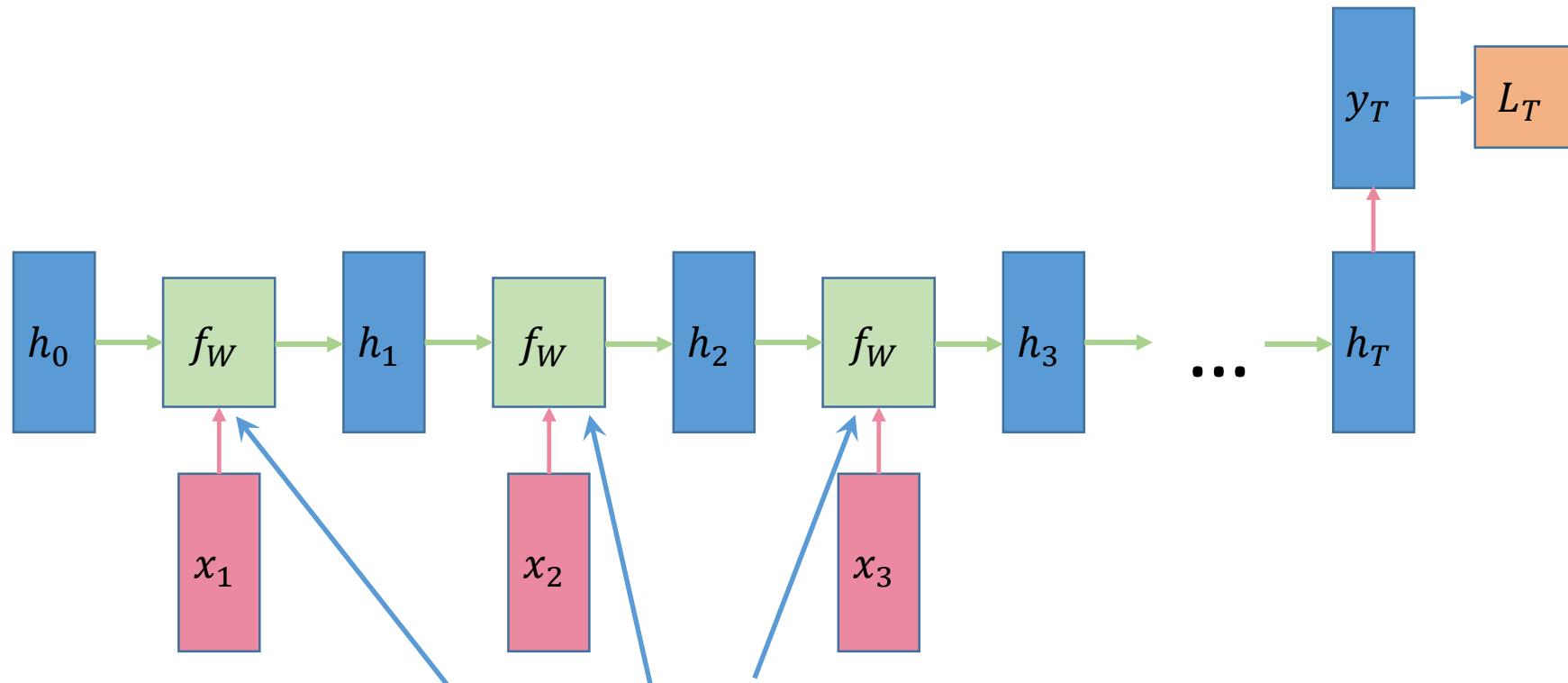
Одна матрица W на каждом шаге

Вычислительный граф. Многие ко многим



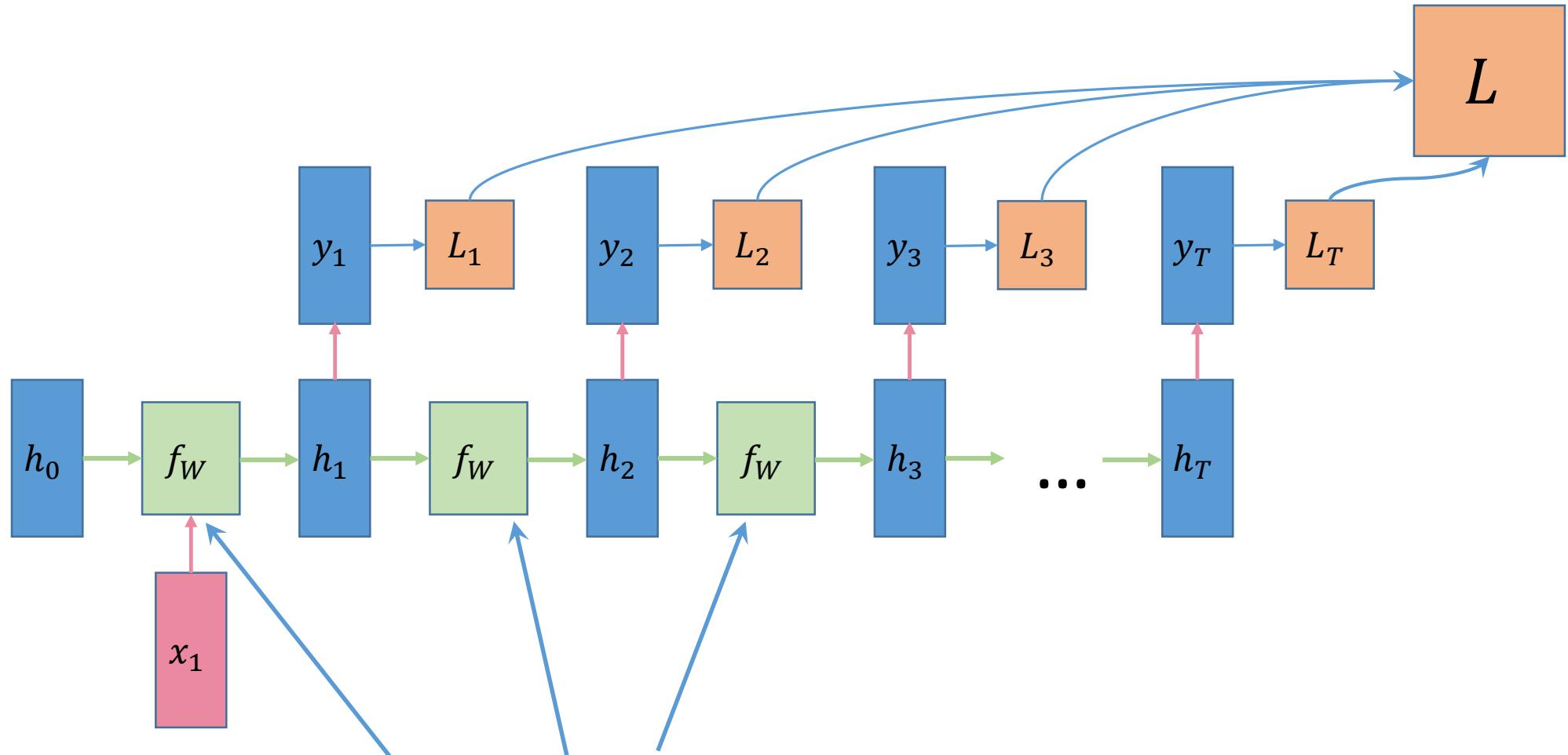
Одна матрица W на каждом шаге

Вычислительный граф. Многие к одному



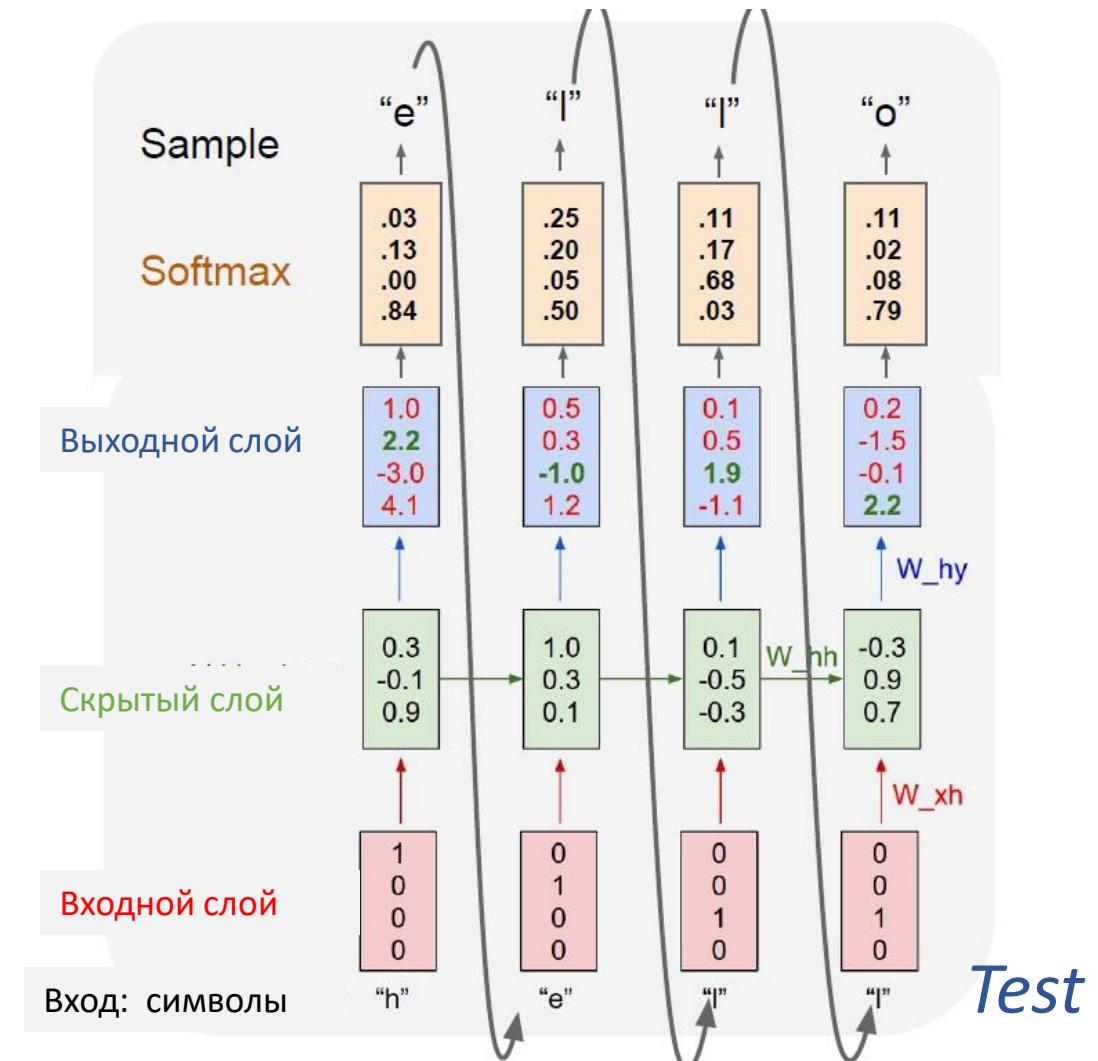
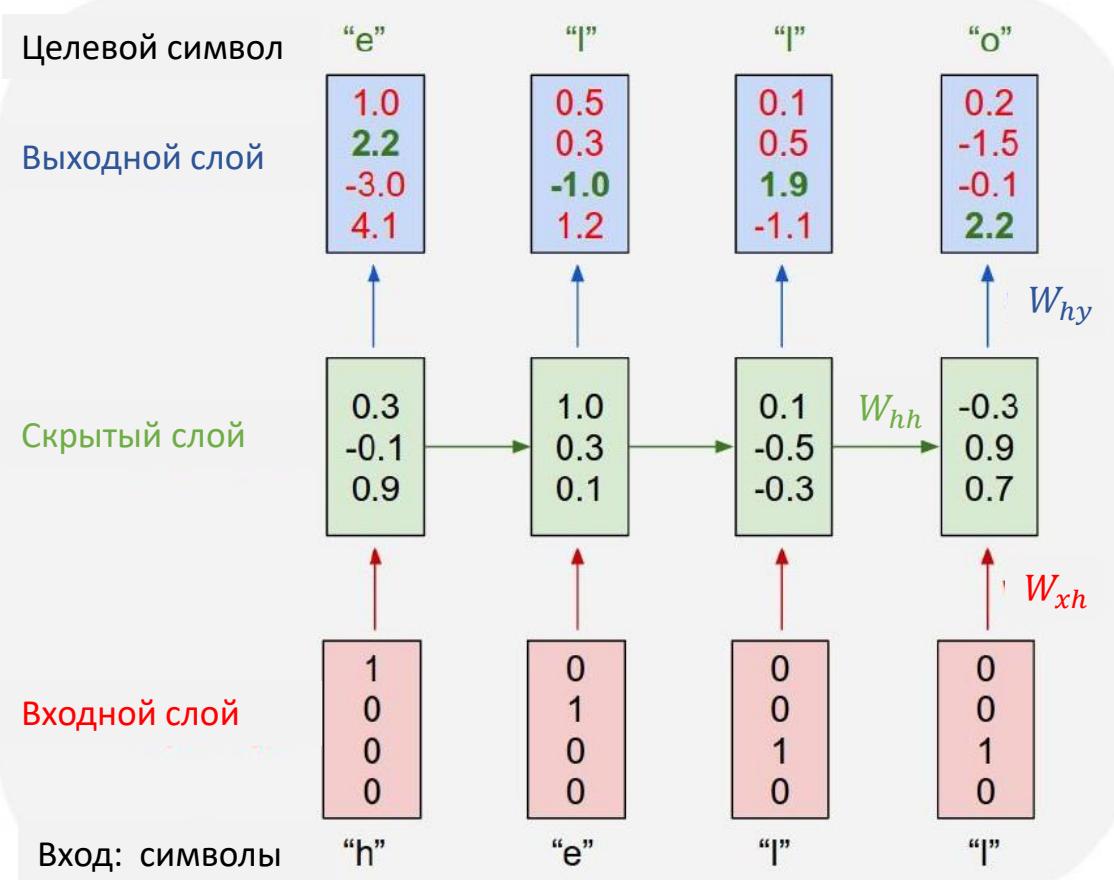
Одна матрица W на каждом шаге

Вычислительный граф. Один ко многим

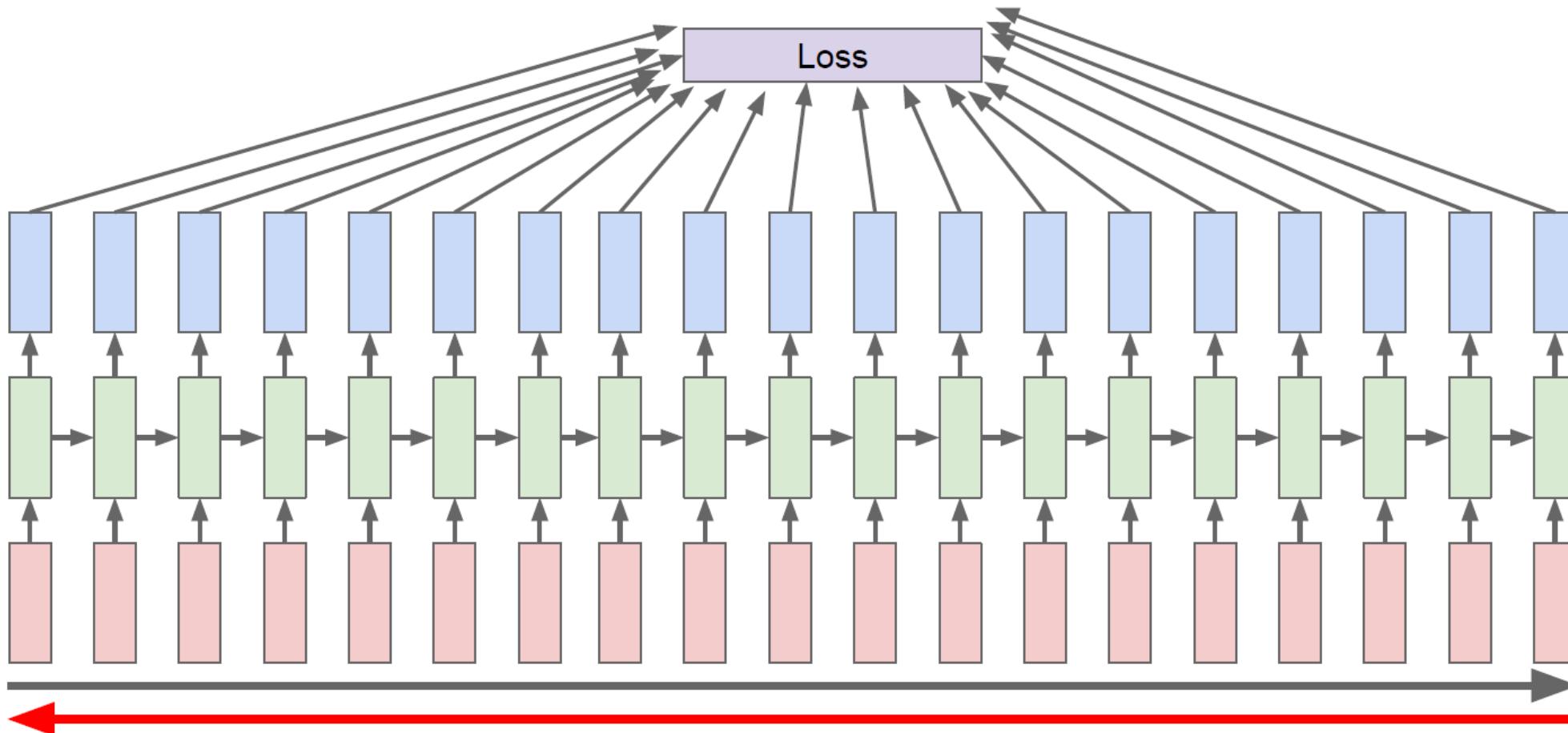


Одна матрица W на каждом шаге

Character-level. Языковая модель

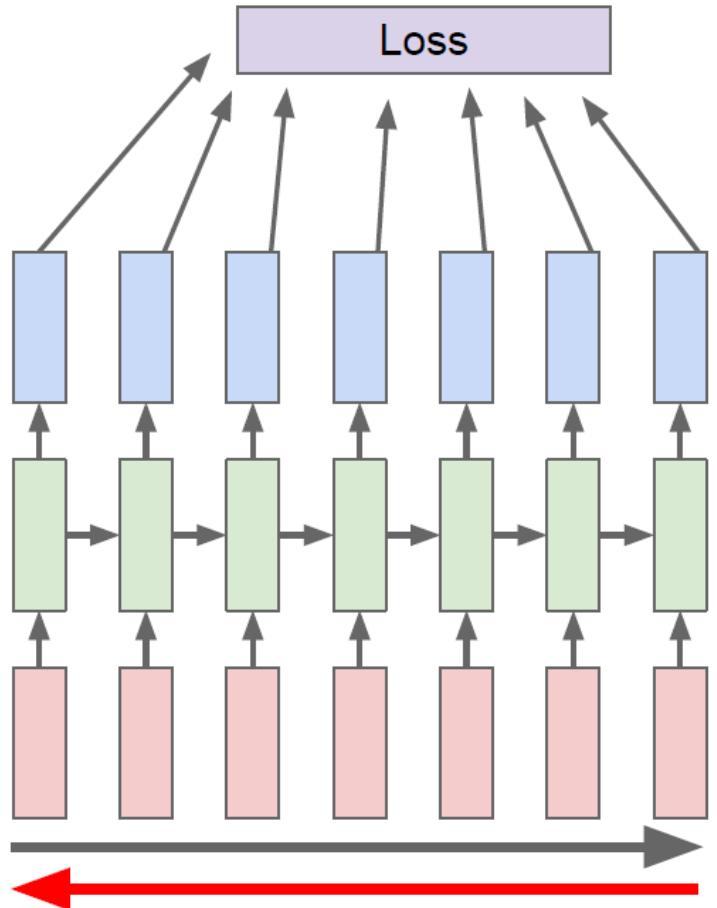


Backpropagation по времени



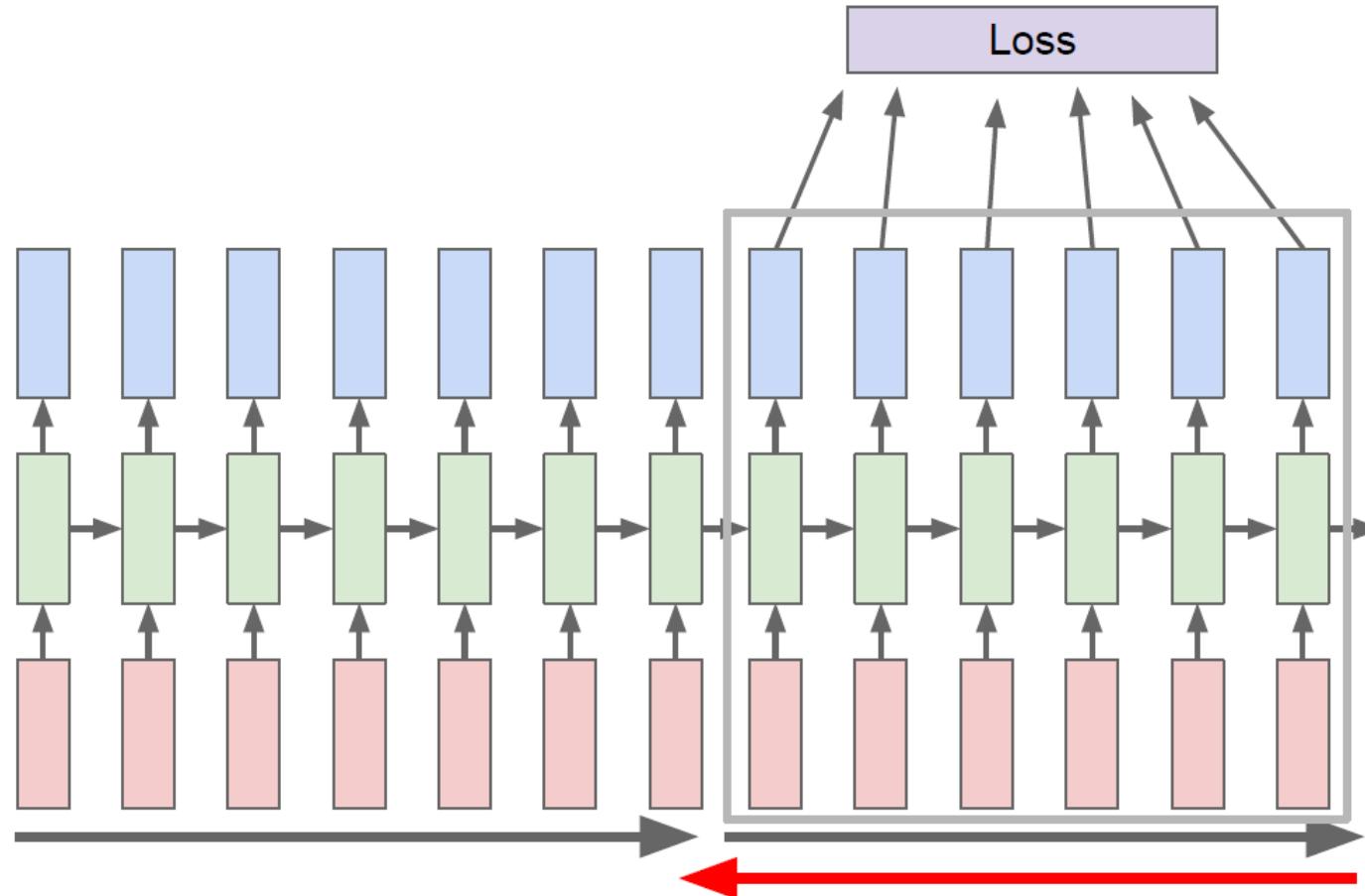
Прямой и обратный проход по всей последовательности

Truncated Backpropagation по времени



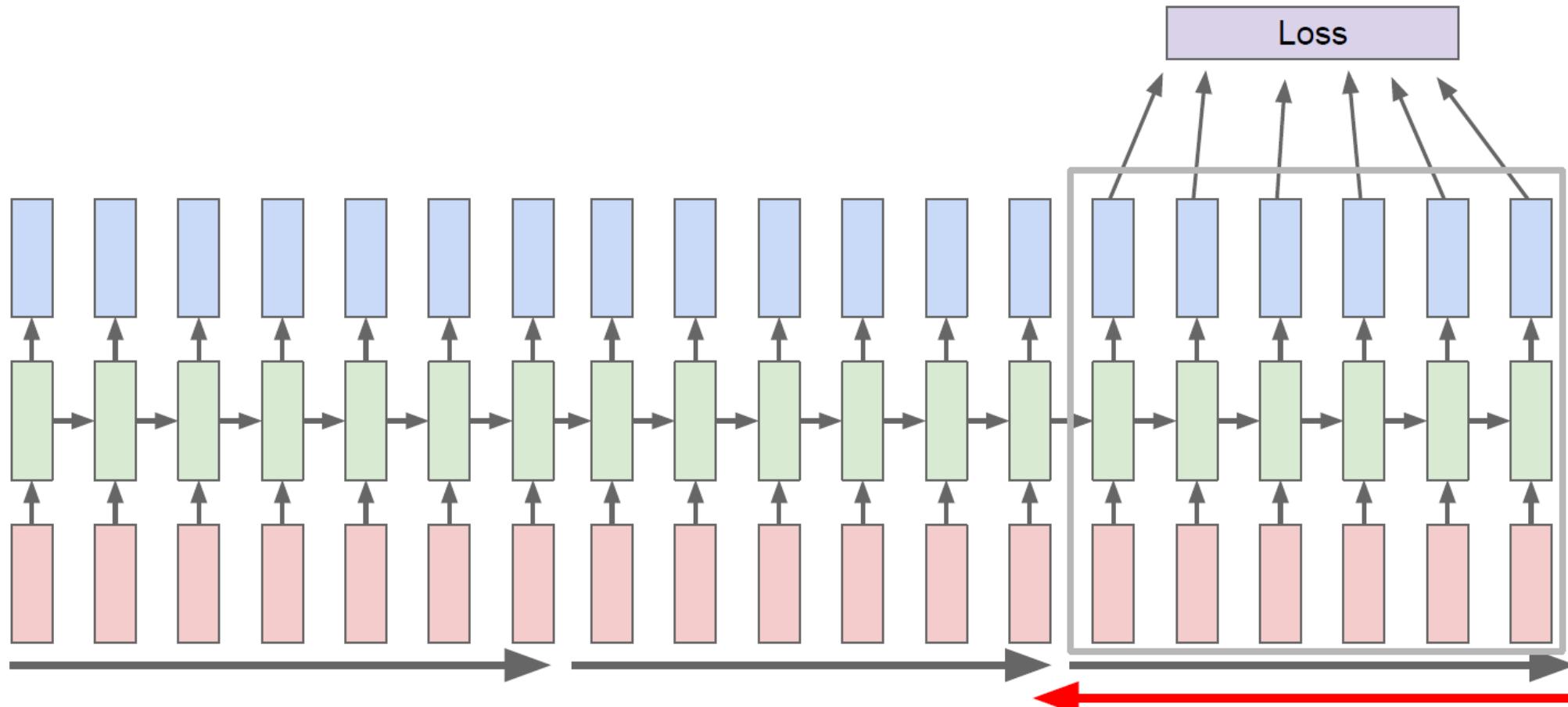
Прямой и обратный проход
только по кусочку всей
последовательности

Truncated Backpropagation по времени

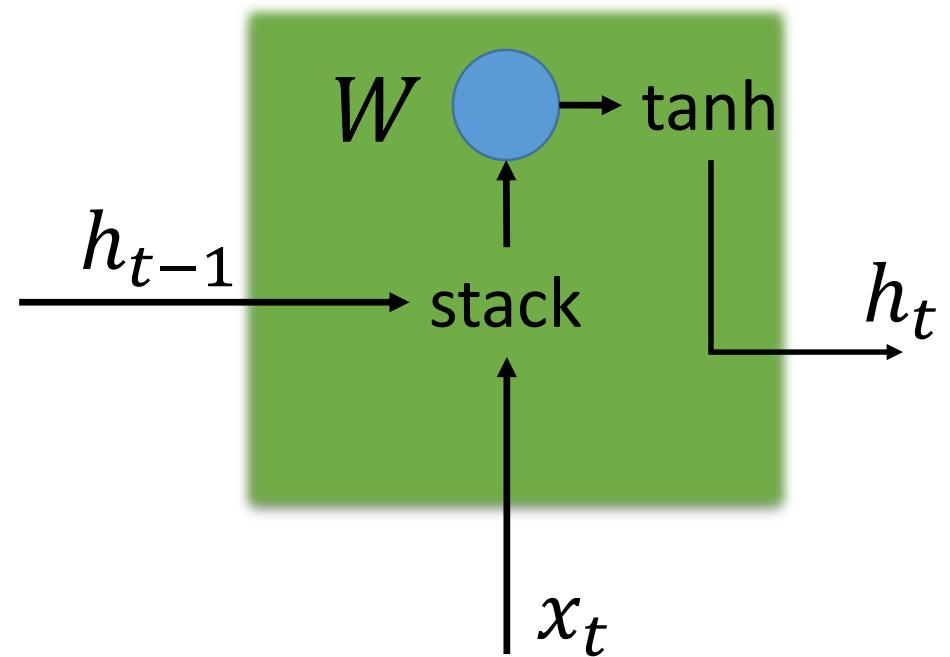


Сохраняем состояния
предыдущего
кусочка и делаем
прямой и обратный
проход по
следующему

Truncated Backpropagation по времени



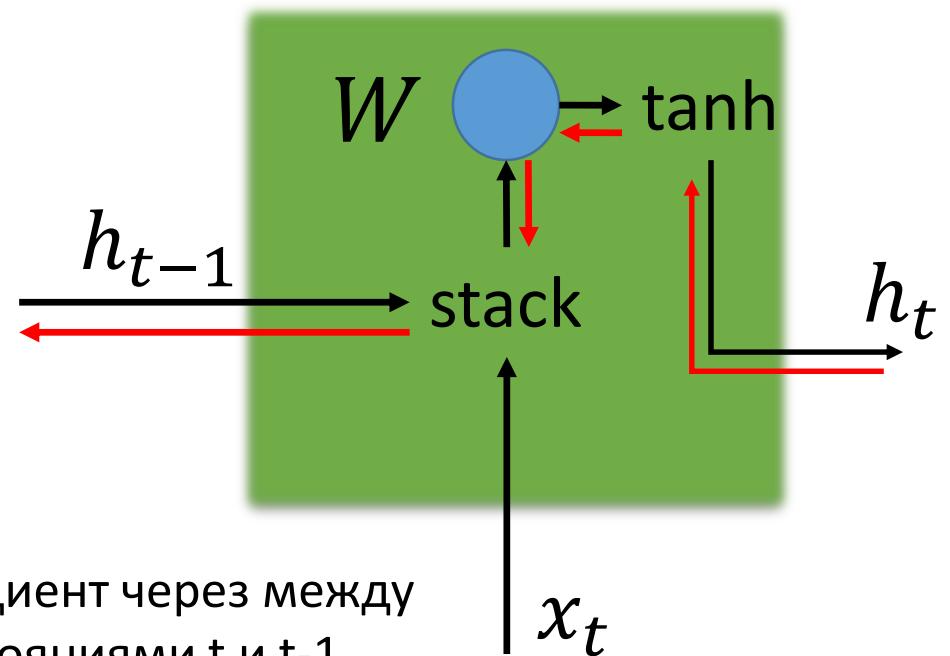
Градиент через RNN



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh((W_{hh} \quad W_{xh}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) \\ &= \tanh(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) \end{aligned}$$

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

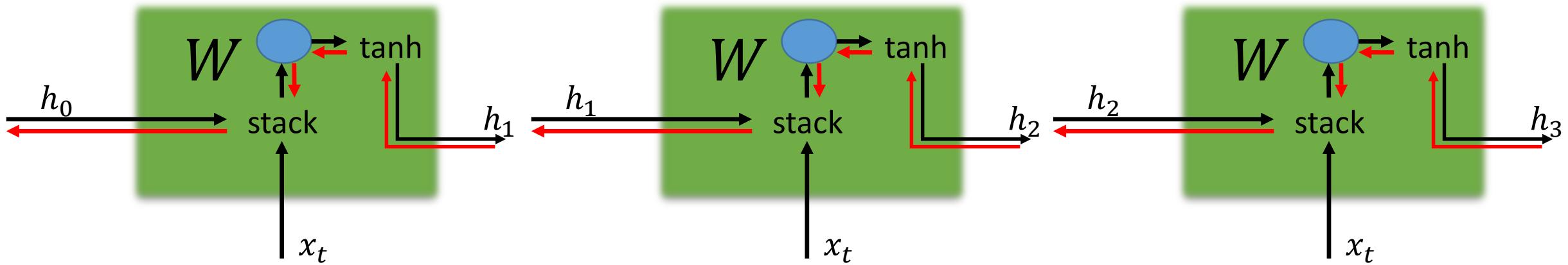
Градиент через RNN



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh((W_{hh} \quad W_{xh}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) \\ &= \tanh(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) \end{aligned}$$

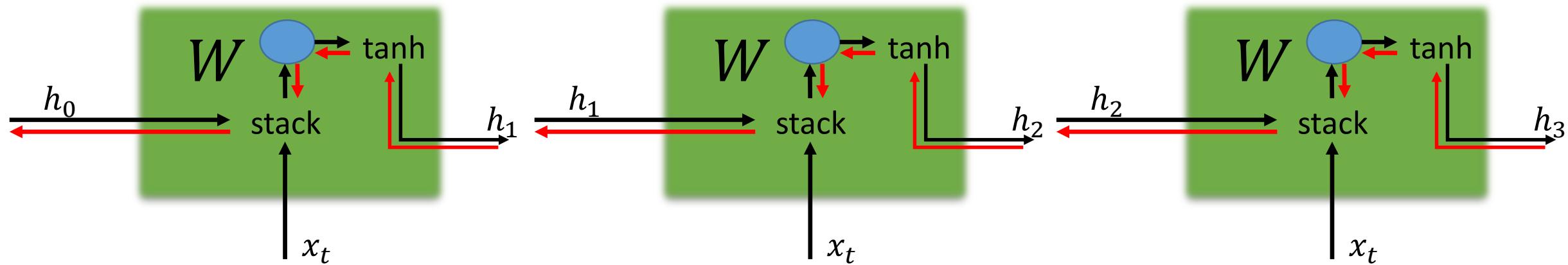
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Градиент через RNN



Вычисление градиента для состояния 0 будет приводить к многократному умножению на матрицу весов W

Градиент через RNN



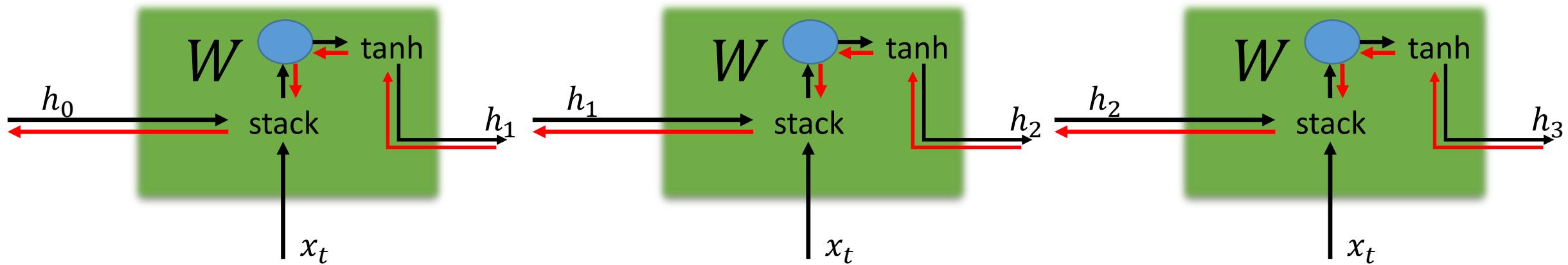
Вычисление градиента для состояния 0 будет приводить к многократному умножению на матрицу весов W

Значения матрицы больше 1:

Exploding gradient

Значения матрицы меньше 1:
Vanishing gradient

Градиент через RNN



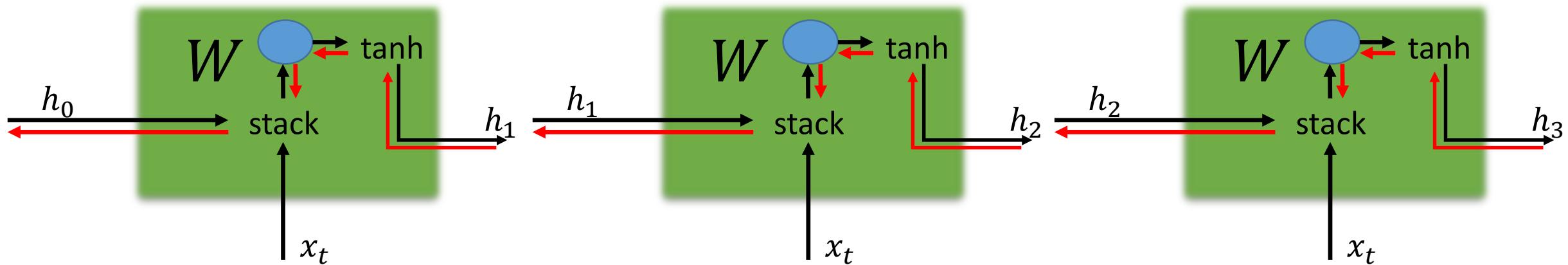
Вычисление градиента для состояния 0 будет приводить к многократному умножению на матрицу весов W

Значения матрицы больше 1:
Exploding gradient



```
torch.nn.utils.clip_grad_norm( model.parameters() , 1)
```

Градиент через RNN



Вычисление градиента для состояния 0 будет приводить к многократному умножению на матрицу весов W

Значения матрицы меньше 1:
Vanishing gradient

Нужно менять архитектуру RNN

Long Short Term Memory (LSTM)

Vanilla RNN

$$h_t = \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

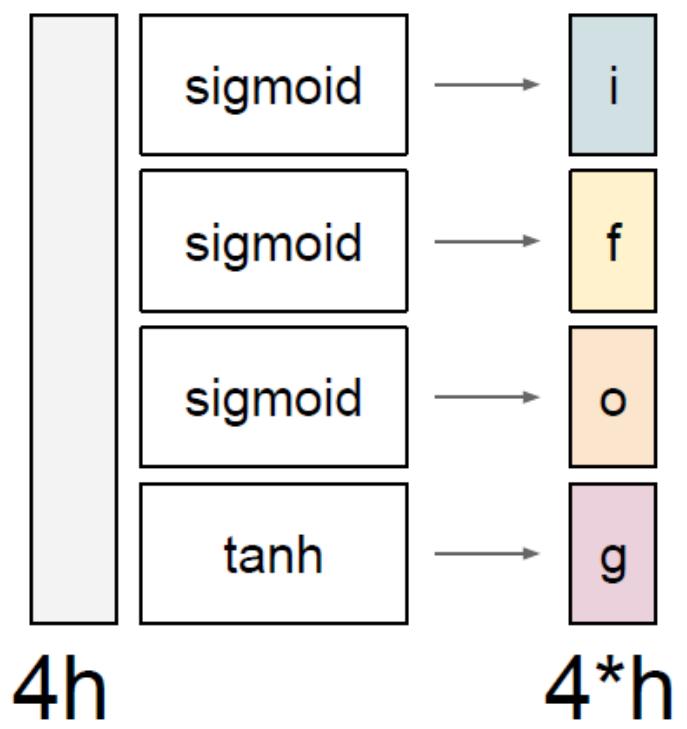
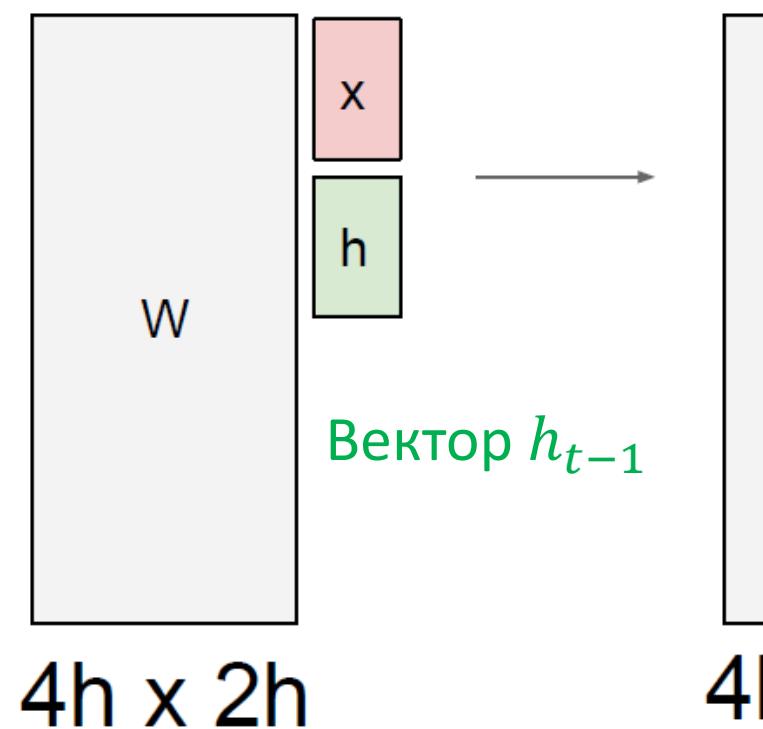
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory”,
Neural Computation
1997

Long Short Term Memory (LSTM)

Вектор x



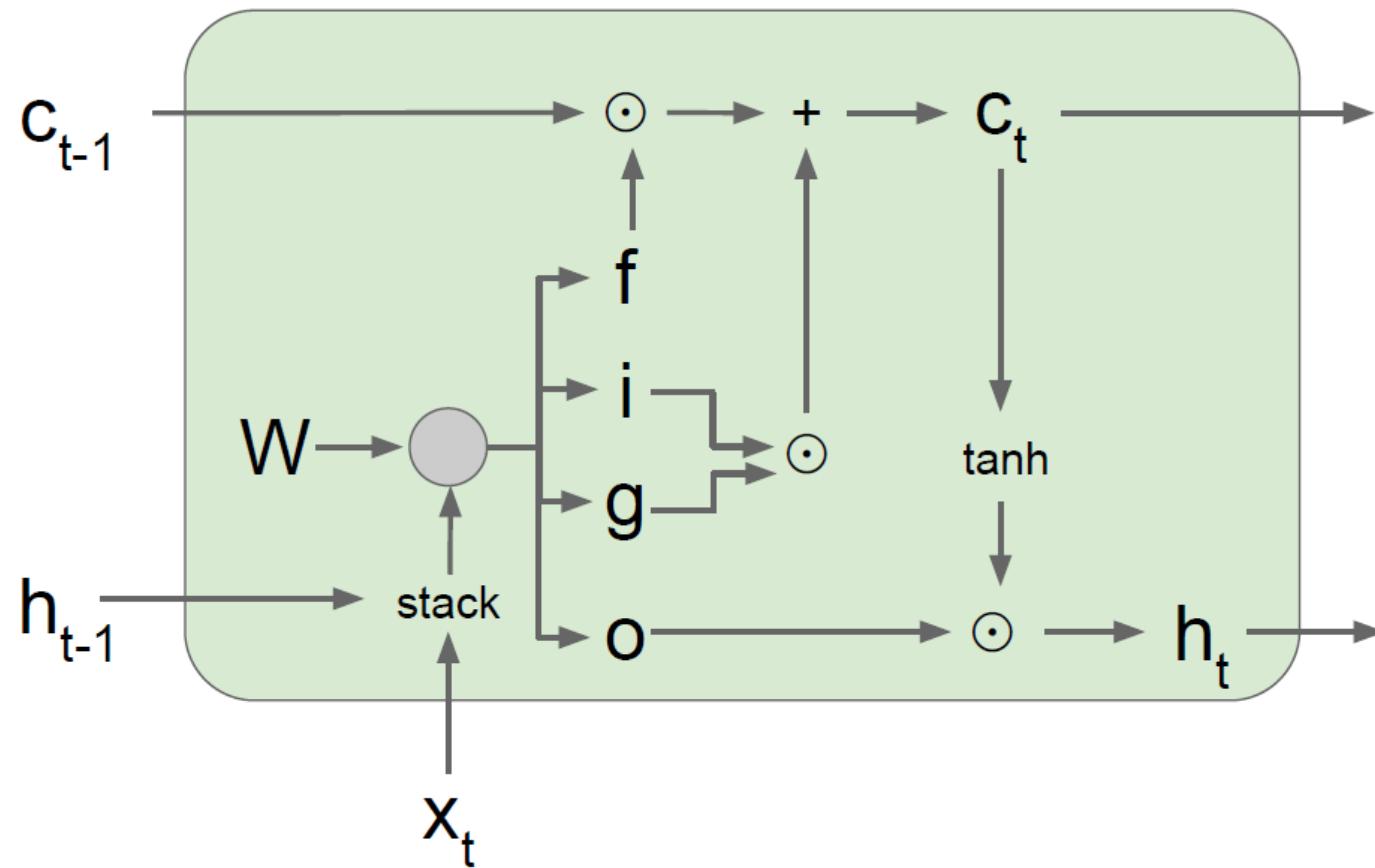
- f: Forget gate, Забывать ли состояние элемента
- i: Input gate, Писать ли состояние элемента
- g: Gate gate, Сколько писать в элемент
- o: Output gate, Сколько выводить из элемента

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

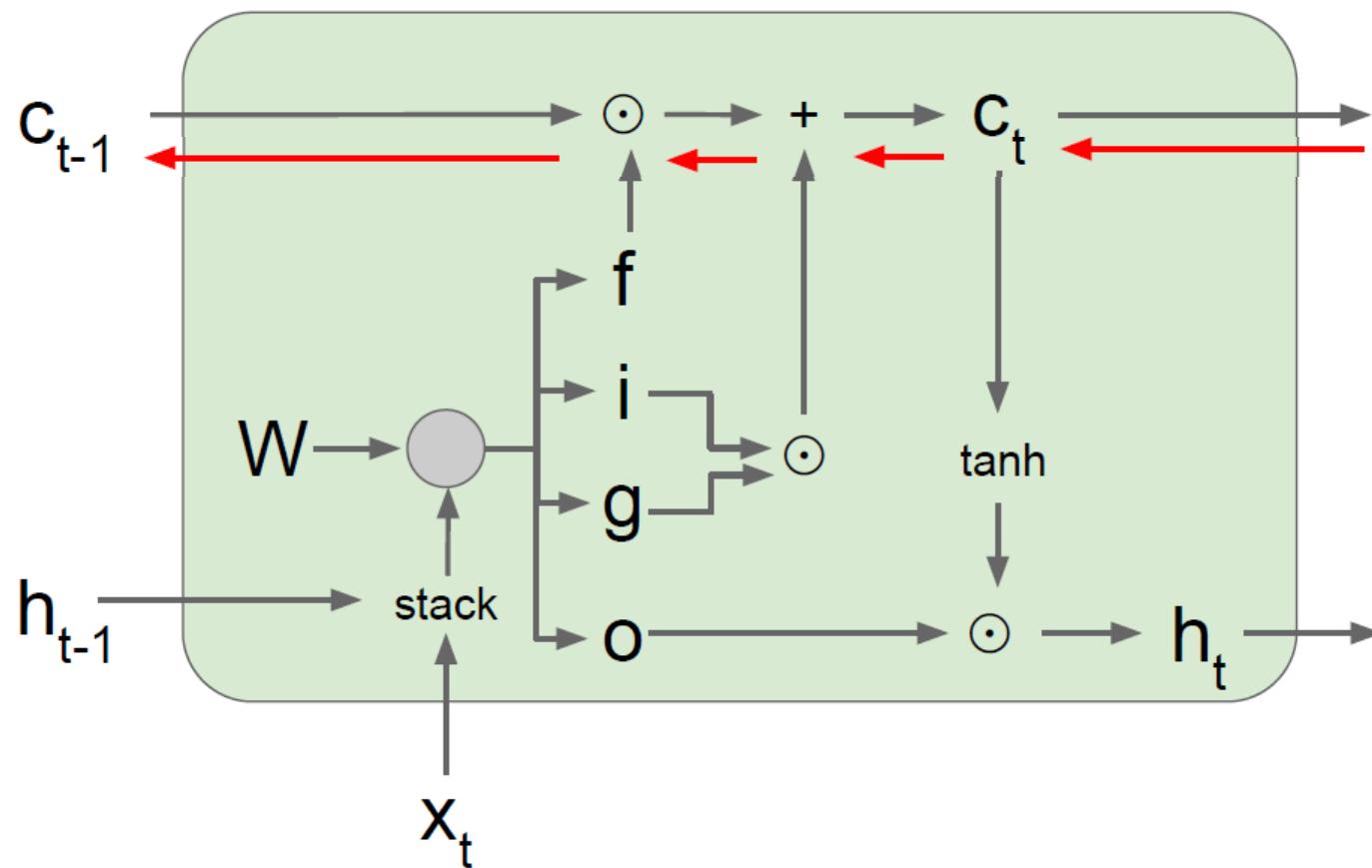
$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

LSTM. Gradient Flow



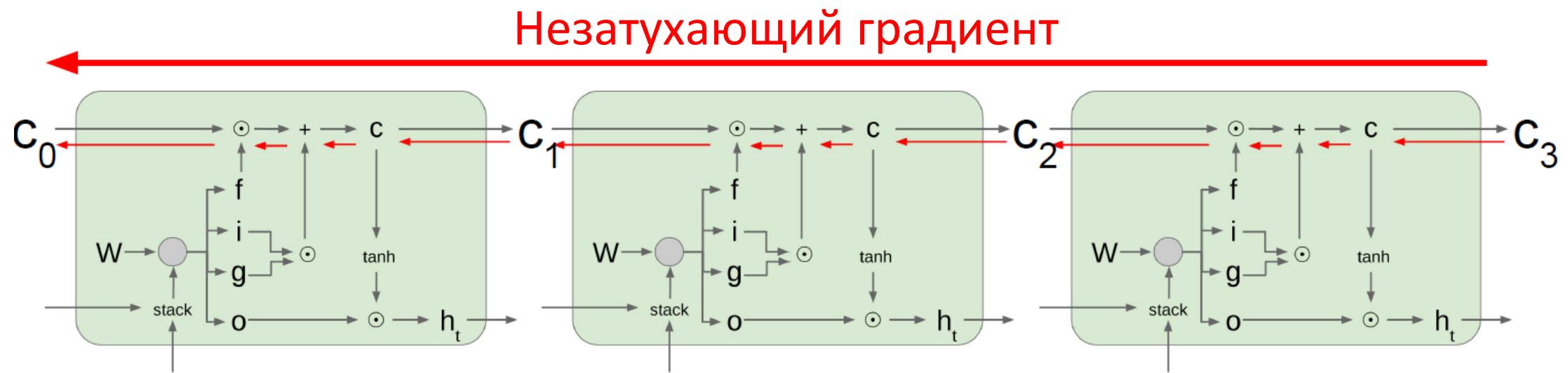
Обратный проход от c_t до c_{t-1} перемножаются только с f без матрицы W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

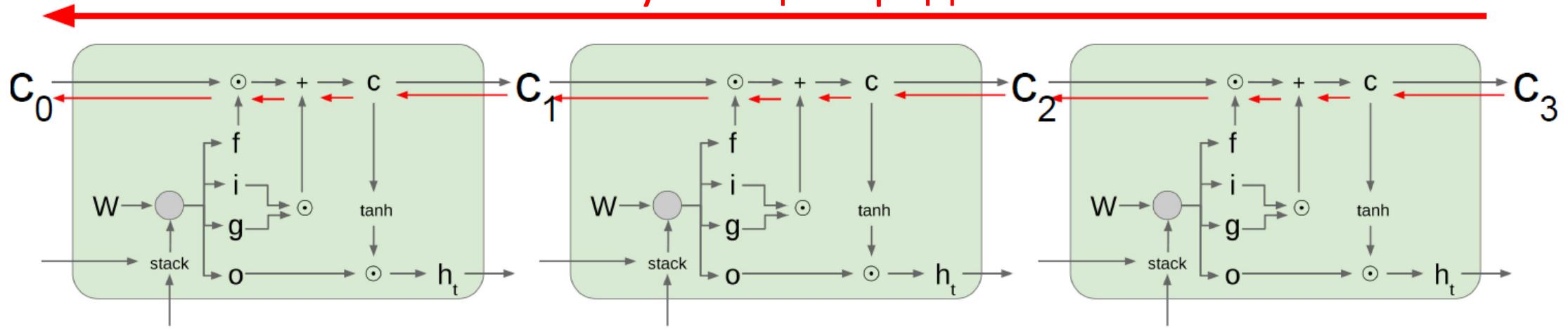
$$h_t = o \odot \tanh(c_t)$$

LSTM. Gradient Flow

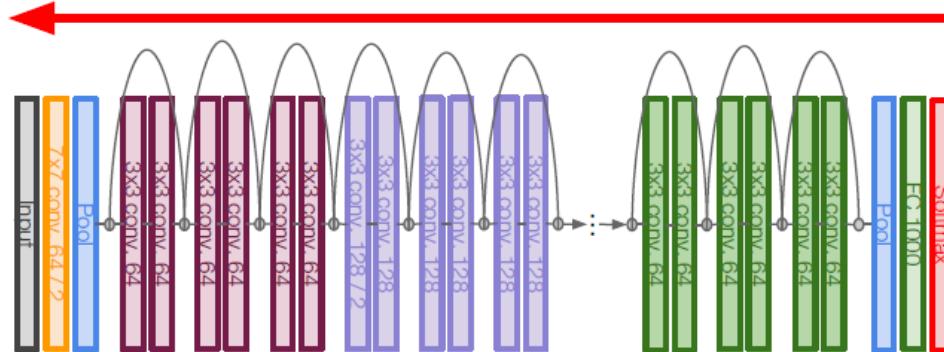


LSTM. Gradient Flow

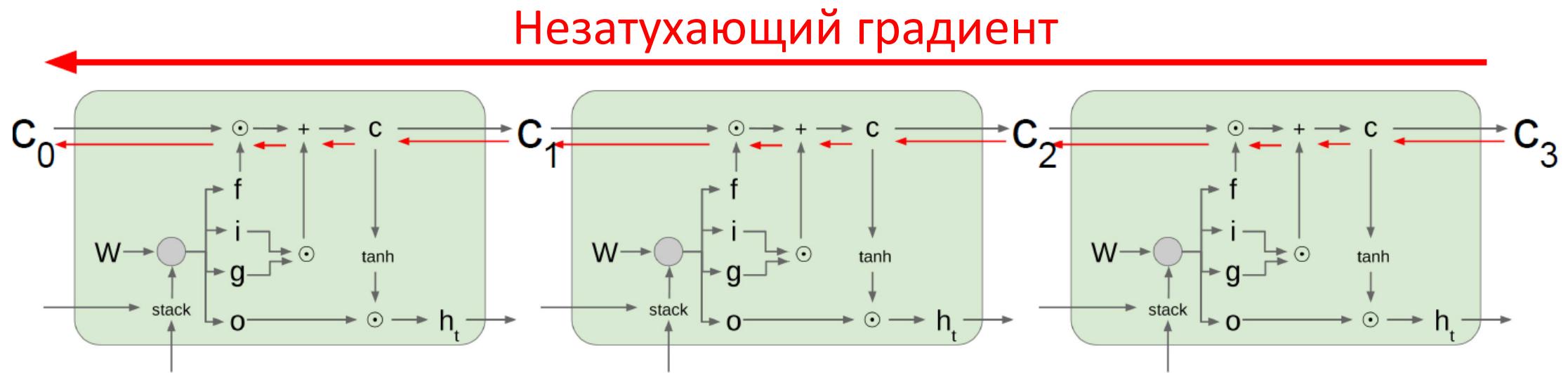
Незатухающий градиент



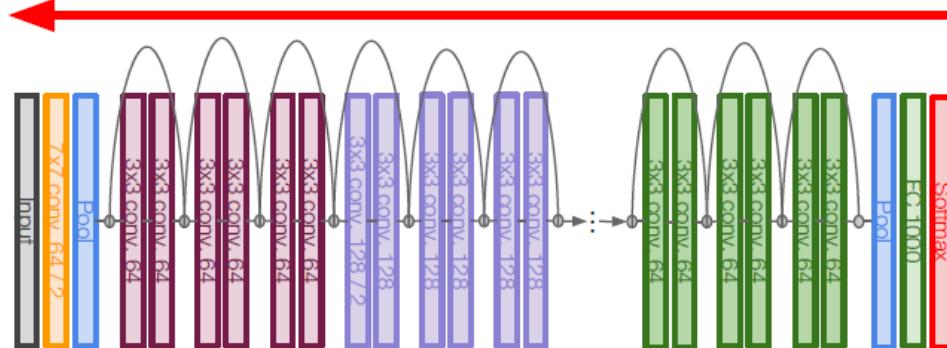
ResNet



LSTM. Gradient Flow



ResNet



Highway Networks

$$g = T(x, W_T)$$

$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

Srivastava et al, "Highway Networks",
ICML DL Workshop 2015

GRU RNN

GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation, Cho et al. 2014*]

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$

$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$

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

Image Captioning

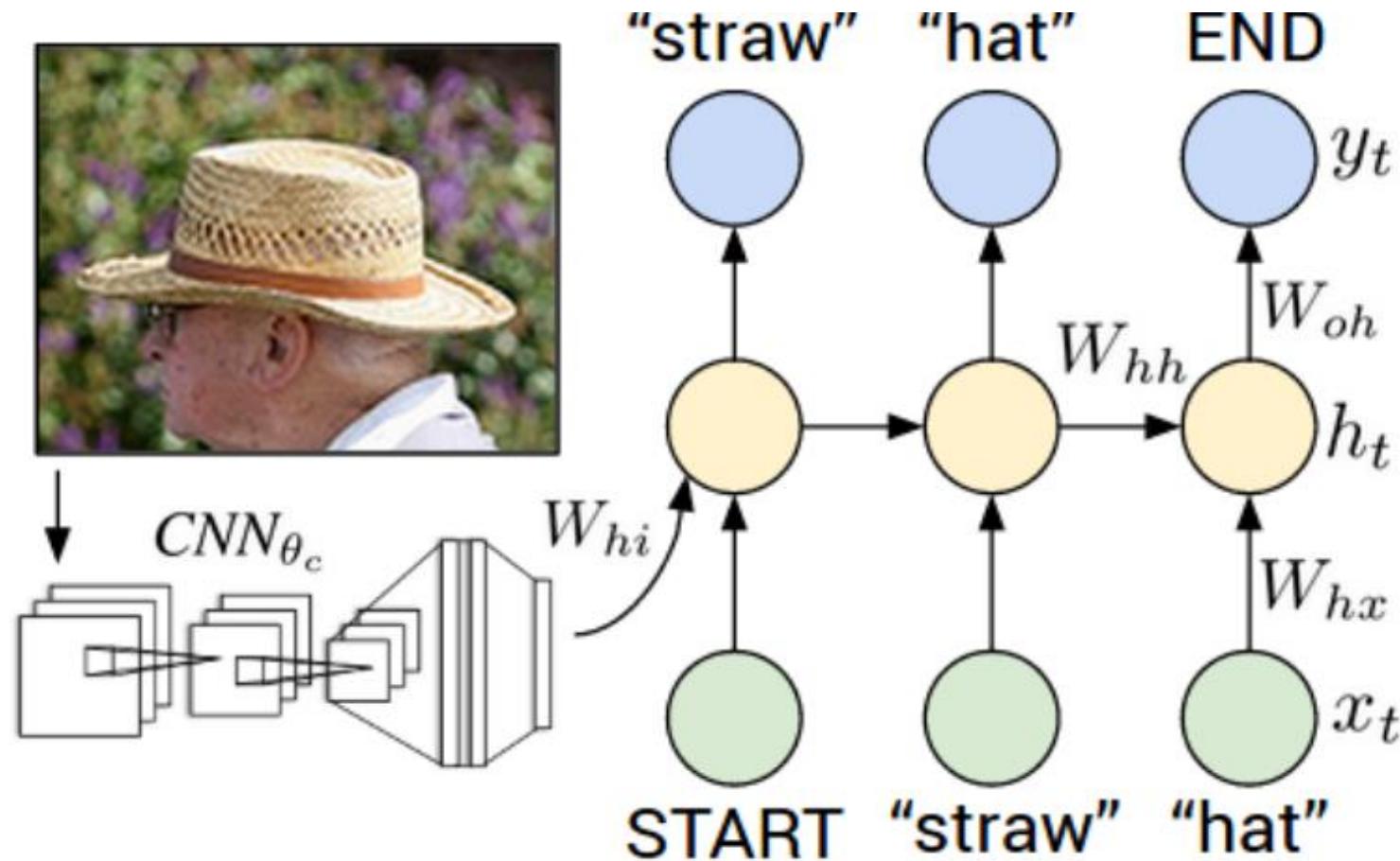
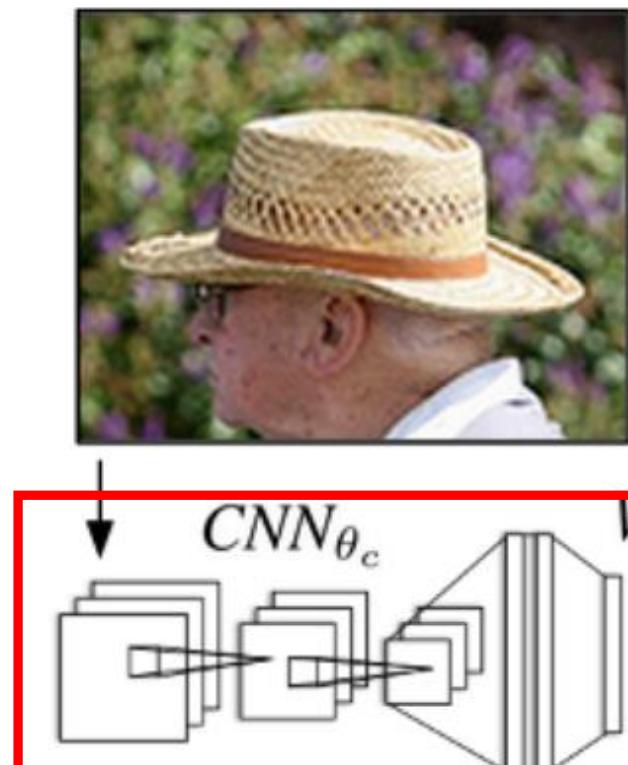
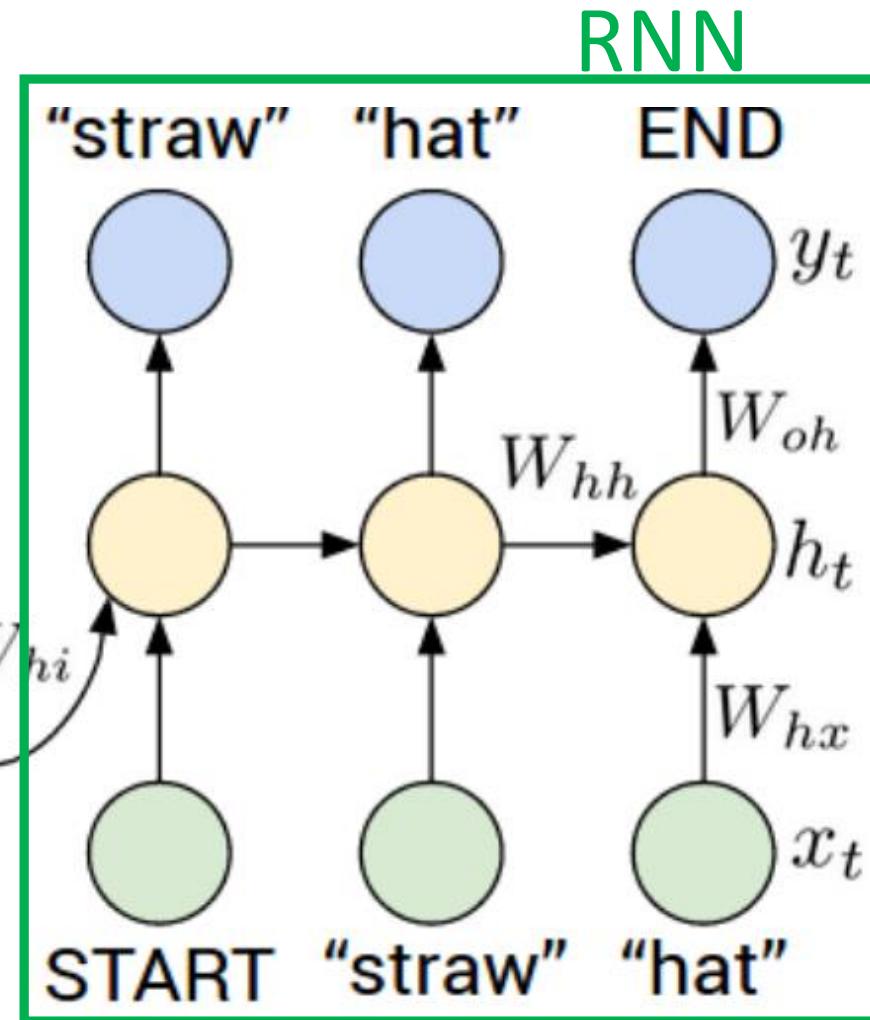


Image Captioning



CNN

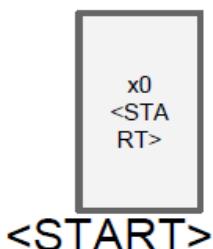




test image

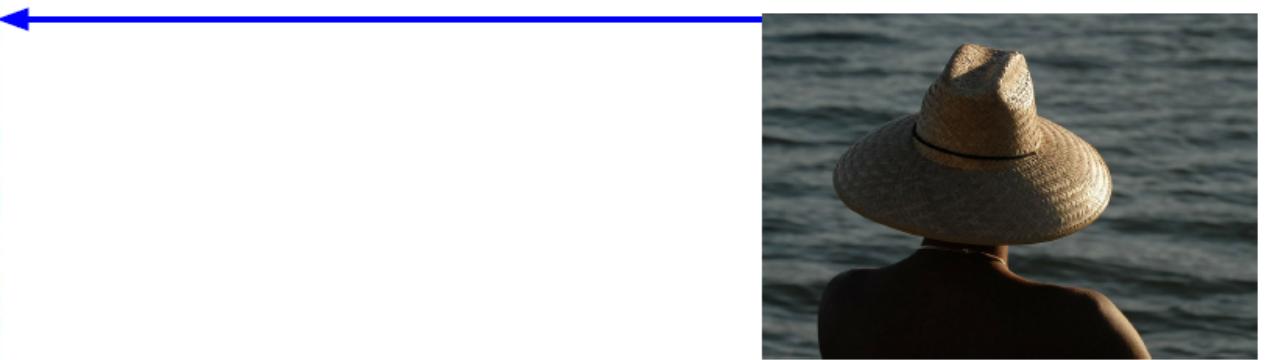


test image

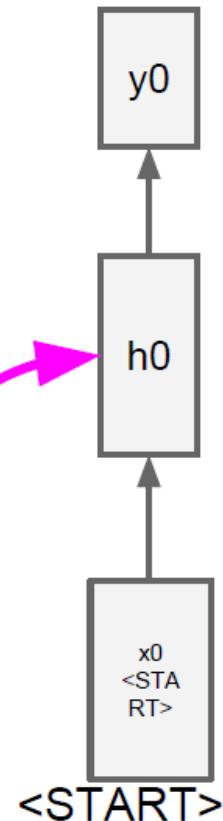




V



test image



Wih

Рекуррентная функция

$$h = \tanh(Wxh * x + Whh * h + Wih * v)$$



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

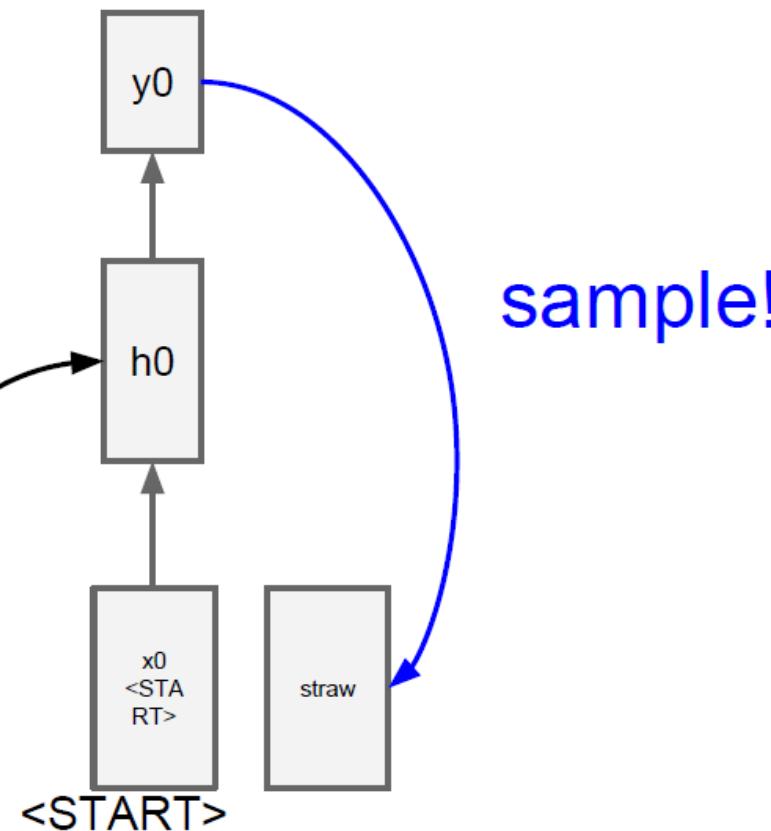
conv-512

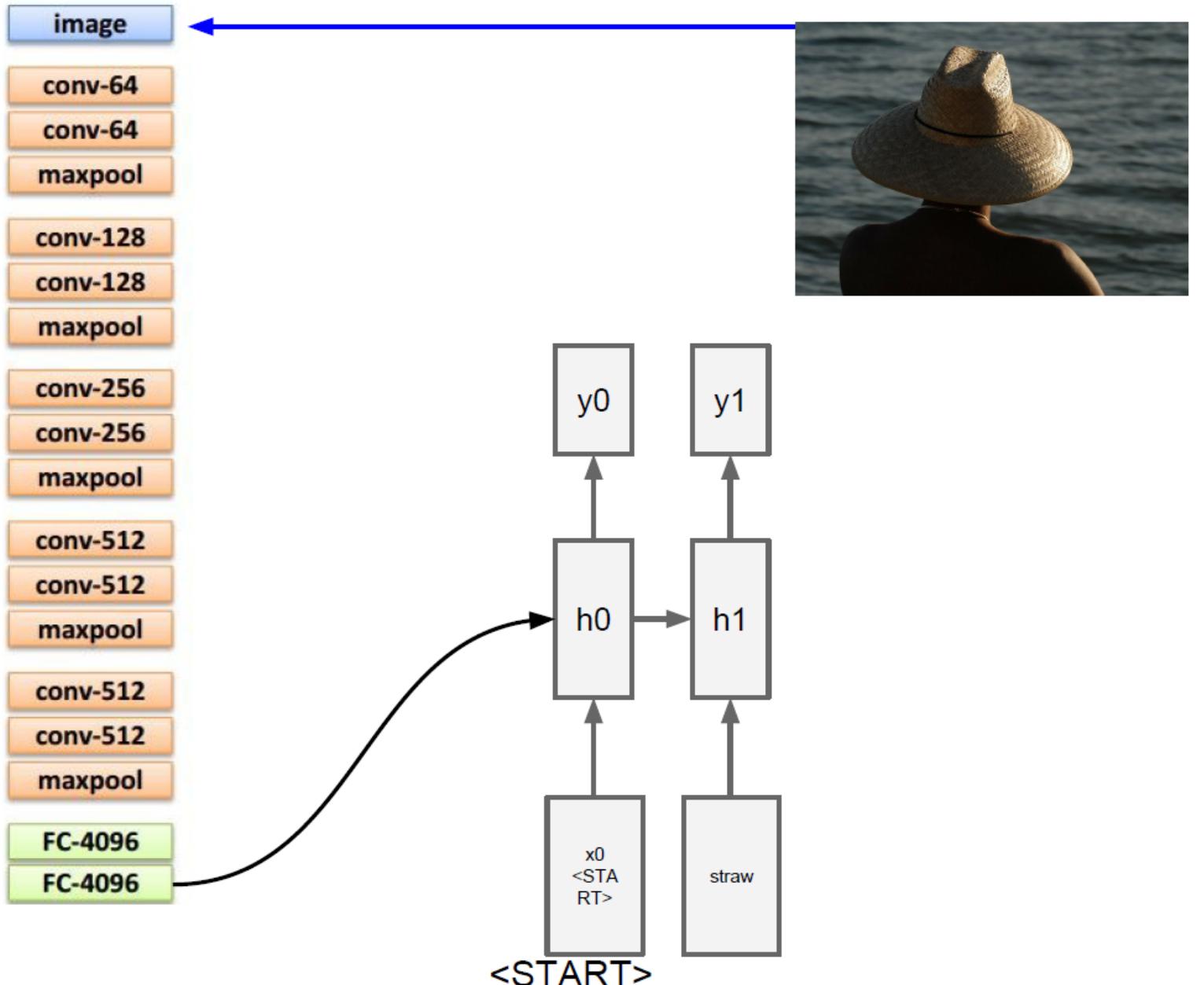
conv-512

maxpool

FC-4096

FC-4096





test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

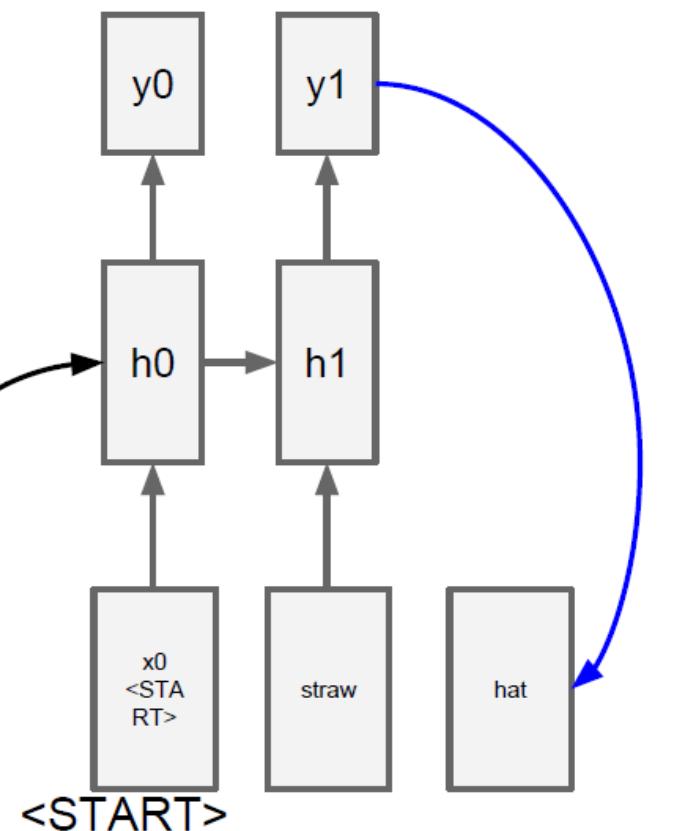
conv-512

conv-512

maxpool

FC-4096

FC-4096

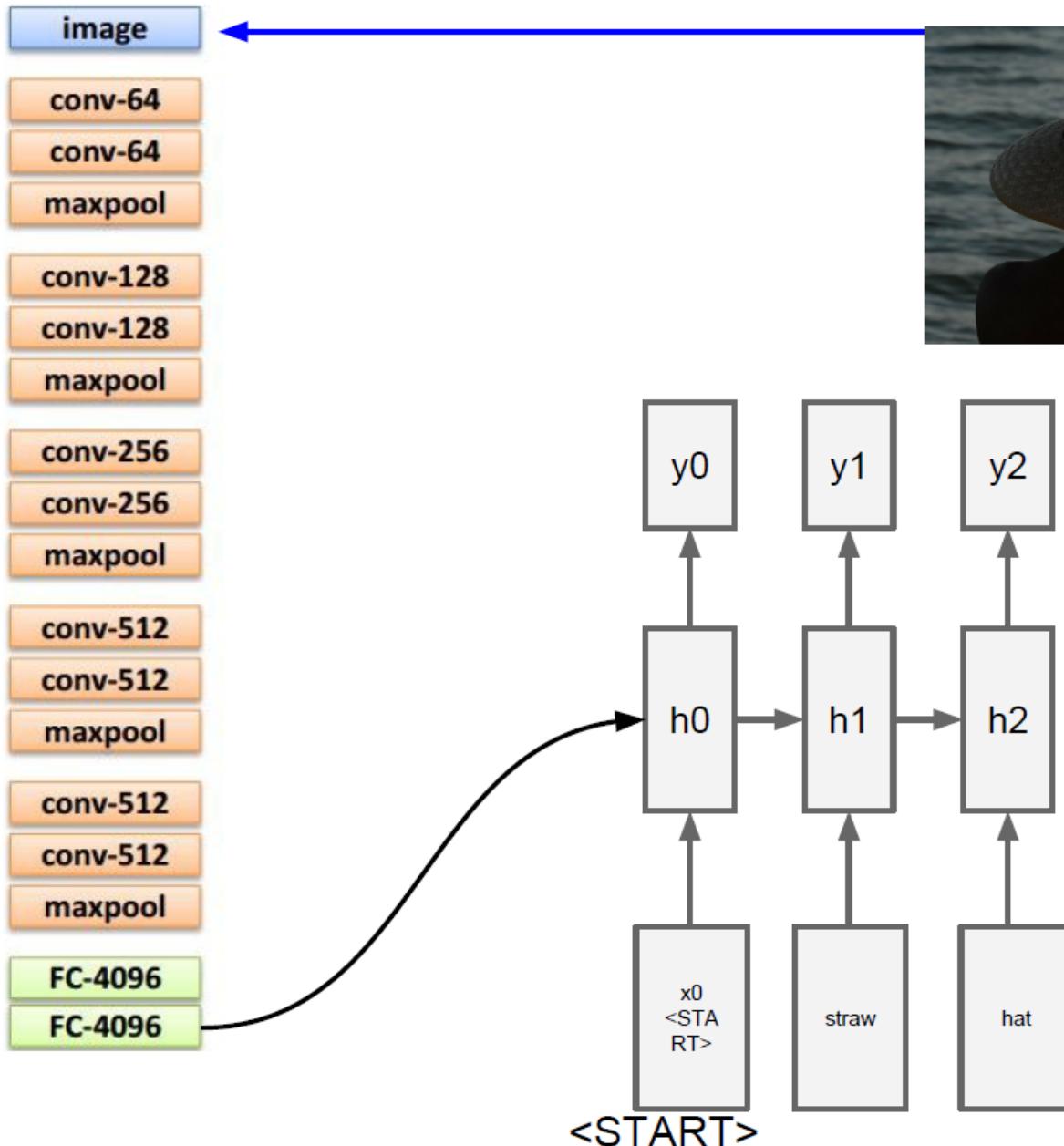


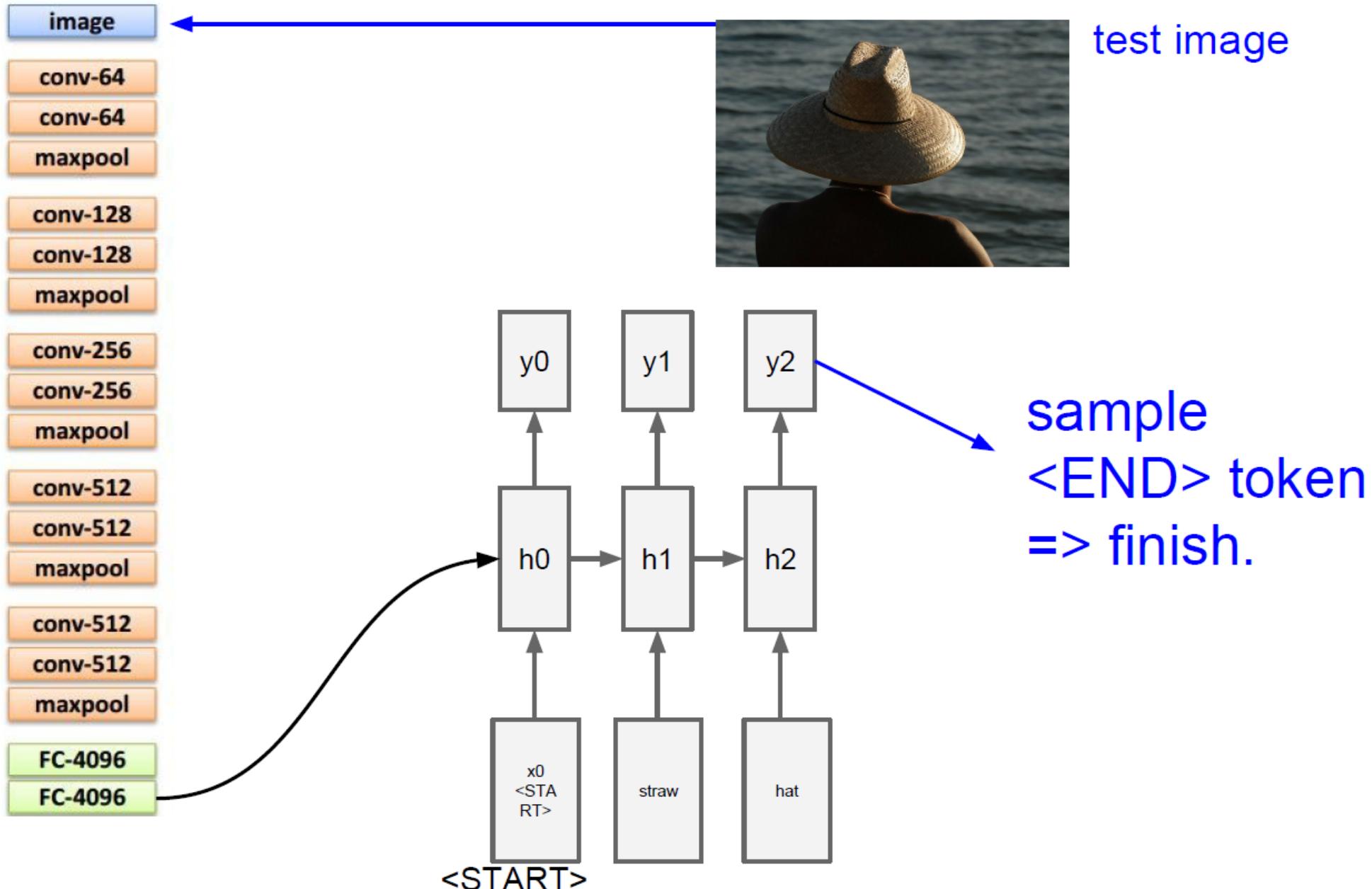
test image

sample!



test image





Пример работы. Image Captioning



A cat sitting on a suitcase on the floor



A cat is sitting on a tree branch



A dog is running in the grass with a frisbee



A white teddy bear sitting in the grass



Two people walking on the beach with surfboards



A tennis player in action on the court



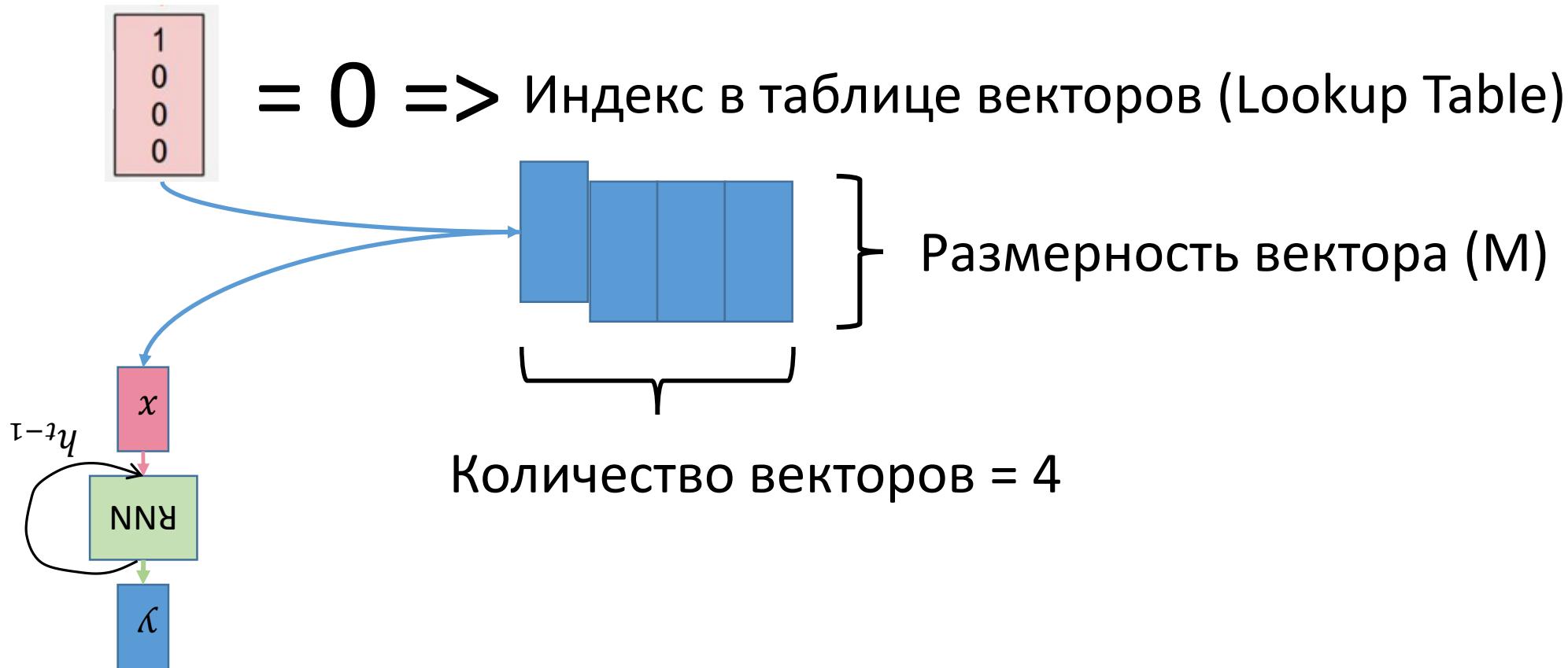
Two giraffes standing in a grassy field



A man riding a dirt bike on a dirt track

Практика. Embedding.

- Входной вектор удобно представлять не как one-hot-encoding



Практика. Embedding.

```
class torch.nn.Embedding(num_embeddings, embedding_dim, padding_idx=None,  
max_norm=None, norm_type=2, scale_grad_by_freq=False, sparse=False) [source] 
```

A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

Parameters:

- **num_embeddings** (*int*) – size of the dictionary of embeddings
- **embedding_dim** (*int*) – the size of each embedding vector
- **padding_idx** (*int, optional*) – If given, pads the output with zeros whenever it encounters the index.
- **max_norm** (*float, optional*) – If given, will renormalize the embeddings to always have a norm lesser than this
- **norm_type** (*float, optional*) – The p of the p-norm to compute for the max_norm option
- **scale_grad_by_freq** (*boolean, optional*) – if `True`, gradient w.r.t. weight matrix will be a sparse tensor. See Notes for more details regarding sparse gradients.

Variables:

weight (*Tensor*) – the learnable weights of the module of shape (num_embeddings, embedding_dim)

Shape:

- Input: LongTensor (N, W), N = mini-batch, W = number of indices to extract per mini-batch

`class torch.nn.LSTM(*args, **kwargs)` [source]

Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$\begin{aligned} i_t &= \text{sigmoid}(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\ f_t &= \text{sigmoid}(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\ g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\ o_t &= \text{sigmoid}(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\ c_t &= f_t * c_{(t-1)} + i_t * g_t \\ h_t &= o_t * \tanh(c_t) \end{aligned}$$

where h_t is the hidden state at time t , c_t is the cell state at time t , x_t is the hidden state from the previous layer at time t or input_t for the first layer, and i_t , f_t , g_t , o_t are the input, forget, cell, and output gates, respectively.

Parameters:

- **input_size** – The number of expected features in the input x
- **hidden_size** – The number of features in the hidden state h
- **num_layers** – Number of recurrent layers.
- **bias** – If `False`, then the layer does not use bias weights b_{ih} and b_{hh} . Default: `True`

- **batch_first** – If `True`, then the input and output tensors are provided as (batch, seq, feature)
- **dropout** – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
- **bidirectional** – If `True`, becomes a bidirectional RNN. Default: `False`

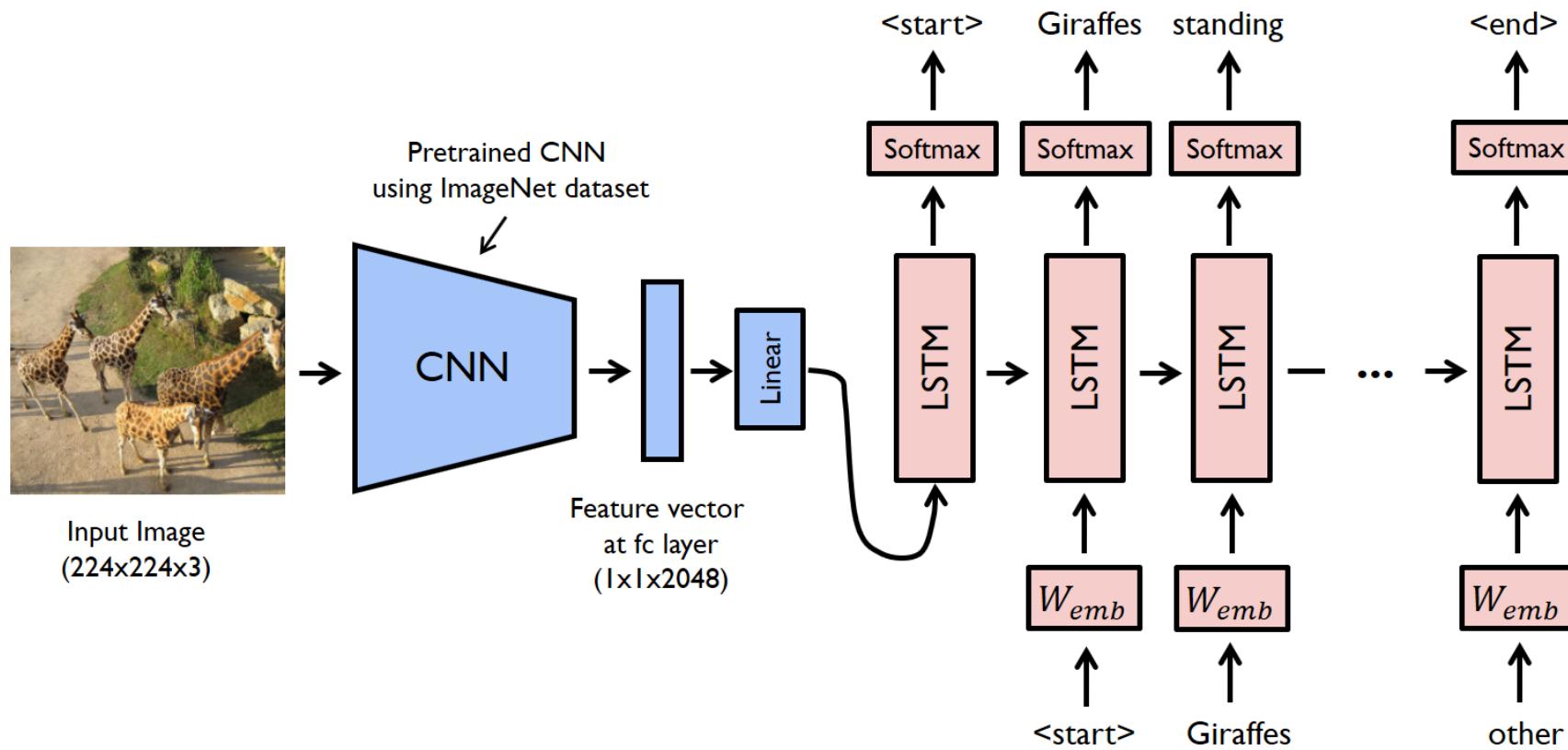
Inputs: $\text{input}, (\text{h}_0, \text{c}_0)$

- **input** (seq_len, batch, input_size): tensor containing the features of the input sequence. The input can also be a packed variable length sequence. See [`torch.nn.utils.rnn.pack_padded_sequence\(\)`](#) for details.
- **h_0** (num_layers * num_directions, batch, hidden_size): tensor containing the initial hidden state for each element in the batch.
- **c_0** (num_layers * num_directions, batch, hidden_size): tensor containing the initial cell state for each element in the batch.

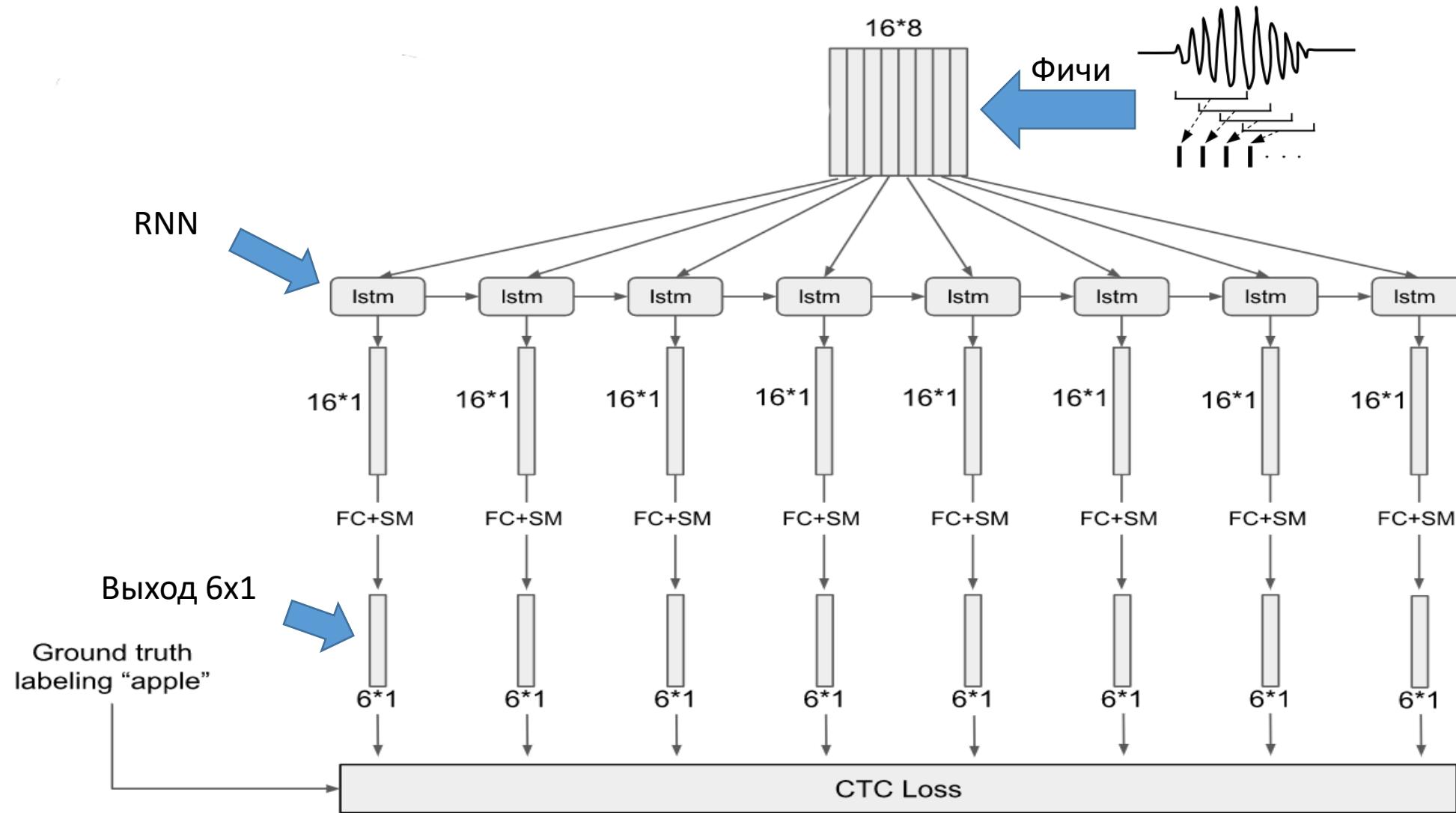
Outputs: $\text{output}, (\text{h}_n, \text{c}_n)$

- **output** (seq_len, batch, hidden_size * num_directions): tensor containing the output features (h_t) from the last layer of the RNN, for each t . If a [`torch.nn.utils.rnn.PackedSequence`](#) has been given as the input, the output will also be a packed sequence.
- **h_n** (num_layers * num_directions, batch, hidden_size): tensor containing the hidden state for $t=\text{seq_len}$
- **c_n** (num_layers * num_directions, batch, hidden_size): tensor containing the cell state for $t=\text{seq_len}$

Практика. Image Captioning



CTC loss



CTC loss

- CTC loss – это “softmax” слой $p_l = \frac{\exp(x_l)}{\sum_k \exp(x_k)}$
- Количество выходов слоя на 1 больше, чем всего маркеров L
- Активация первых $|L|$ элементов слоя интерпретируется как вероятность
- Активация дополнительного юнита интерпретируется как отсутствие маркера. “blank”

CTC loss

- Для входной последовательности x длиной T
 - Задаем RNN с m входами, n выходами и w – вектор весов как непрерывное отображение $N_w: (R_m)^T \rightarrow (R_n)^T$
 - Тогда $y = N_w(x)$ – последовательность выходов RNN
- y_k^t - активация выходного элемента k в момент времени t
- y_k^t - вероятность про наблюдать маркер k в момент времени t
 - Определяет распределение по множеству L'^T последовательностей длины T над алфавитом $L' = L \cup \{blank\}$:
 - $p(\pi|x) = \prod_{t=1}^T y_{\pi_t}^t, \forall \pi \in L'^T.$
- Элементы L'^T - это пути π

CTC loss - пути



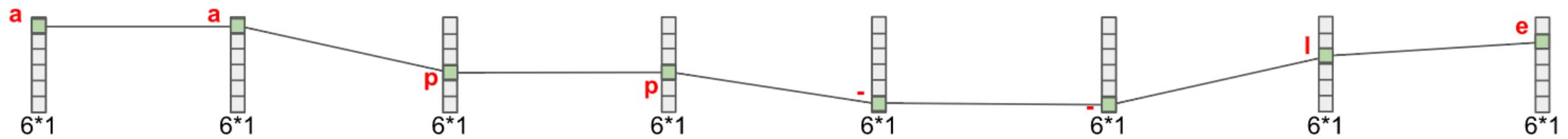
Path1: "ap-pl-ee"

B("ap-pl-ee")

Labeling: "apple"

$$p(\text{"ap-pl-ee"}) = y_a^1 \cdot y_p^2 \cdot y_-^3 \cdot y_p^4 \cdot y_l^5 \cdot y_-^6 \cdot y_e^7 \cdot y_e^8$$

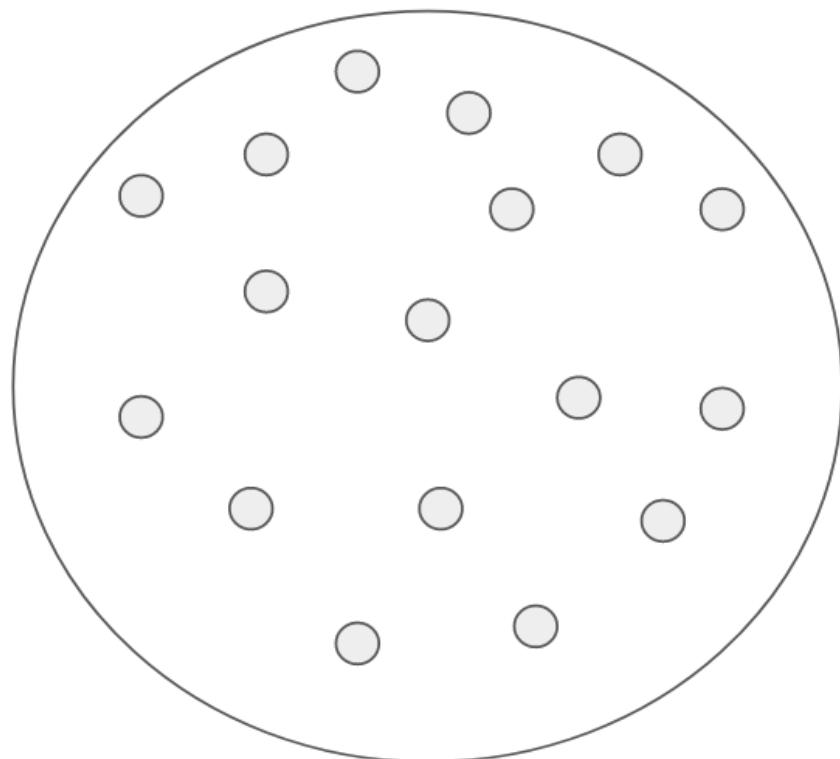
CTC loss - пути



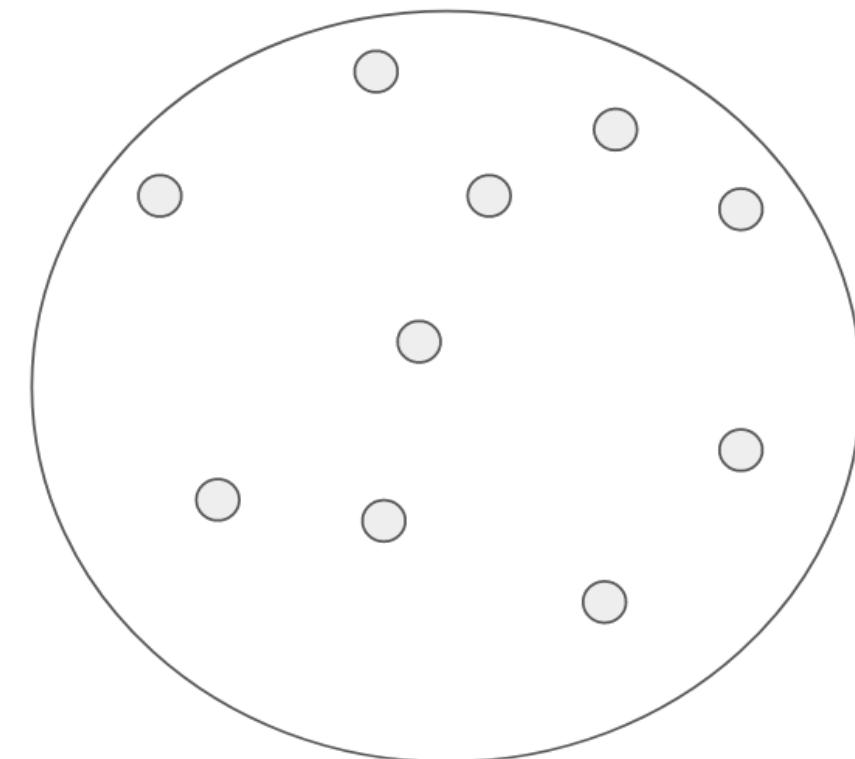
Path1: "ap-pl-ee" $\xrightarrow{\text{B("ap-pl-ee")}}$ Labeling: "apple"
 $p(\text{"ap-pl-ee"}) = y_a^1 \cdot y_p^2 \cdot y_{-}^3 \cdot y_p^4 \cdot y_l^5 \cdot y_{-}^6 \cdot y_e^7 \cdot y_e^8$

Path2: "aapp--le" $\xrightarrow{\text{B("aapp--le")}}$ Labeling: "aple"
 $p(\text{"aapp--le"}) = y_a^1 \cdot y_a^2 \cdot y_p^3 \cdot y_p^4 \cdot y_{-}^5 \cdot y_{-}^6 \cdot y_l^7 \cdot y_e^8$

CTC loss. Пути

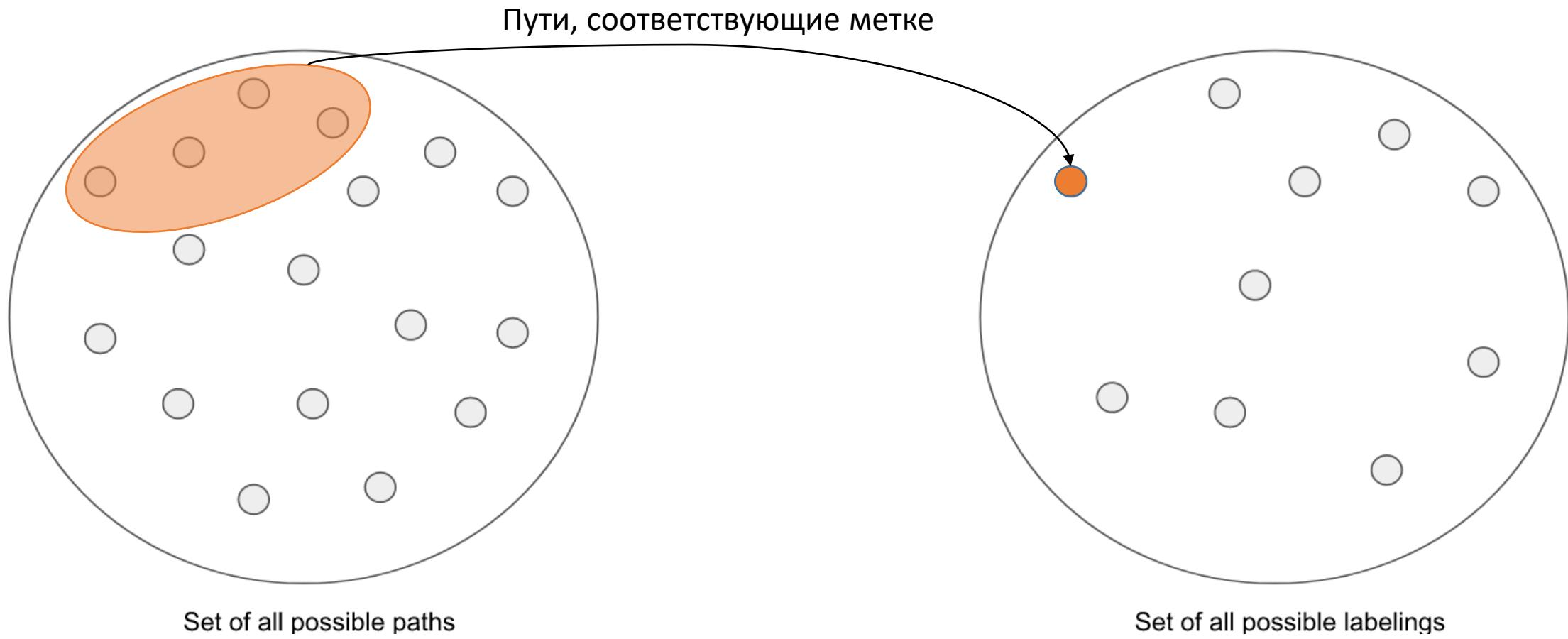


Set of all possible paths

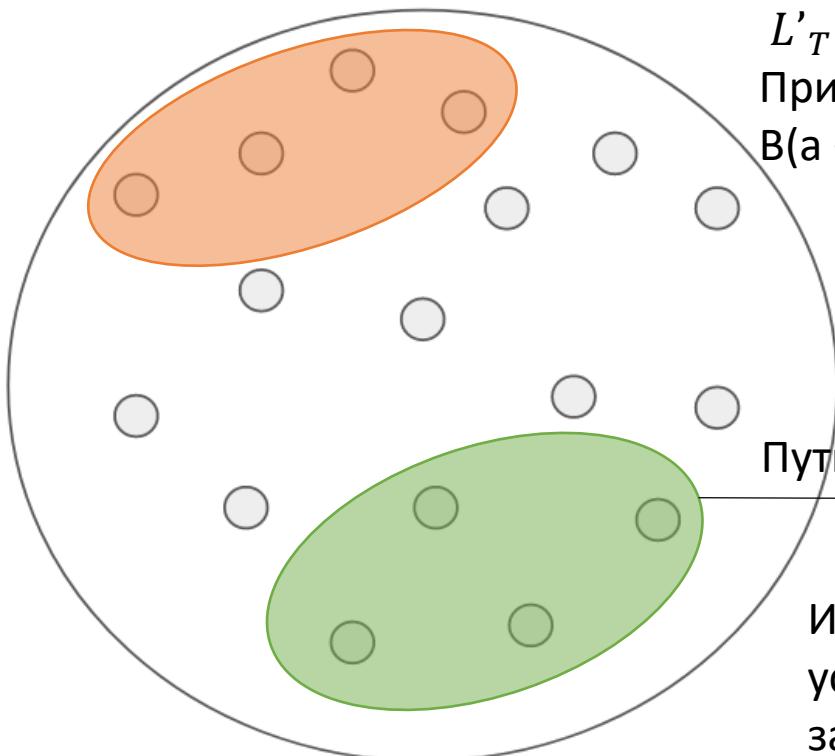


Set of all possible labelings

CTC loss. Пути



CTC loss. Пути



Мы хотим получить маппинг: B :

$$L'_T \rightarrow L^{\leq T}.$$

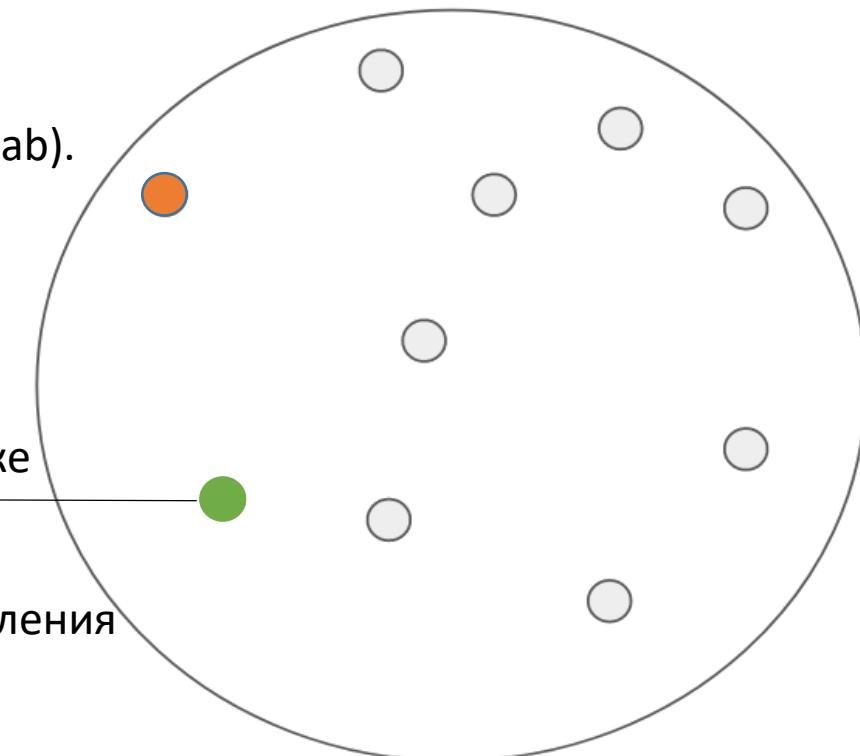
Пример:

$$B(a - ab-) = B(-aa - -abb) = aab).$$

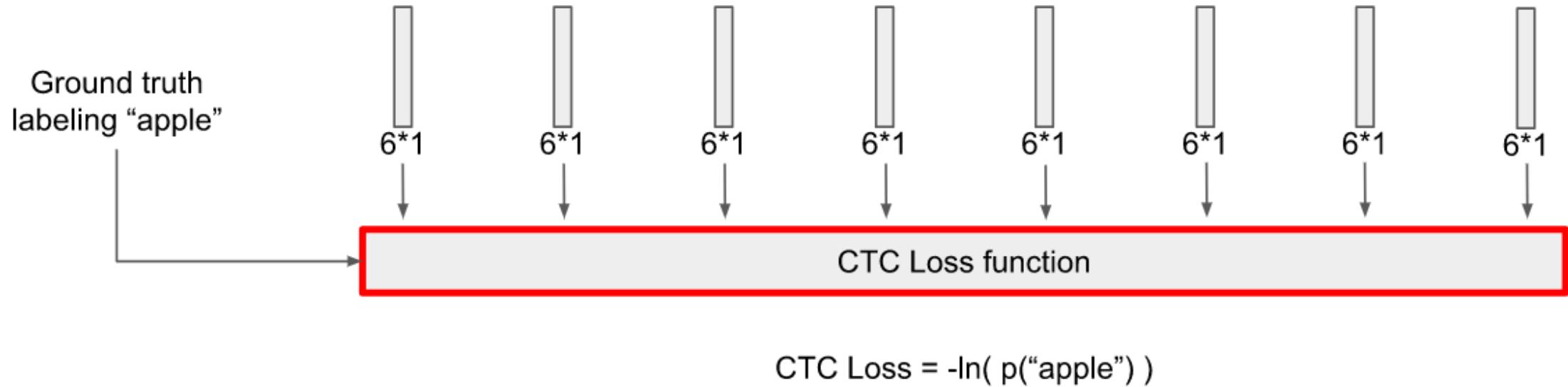
Пути, соответствующие метке

Используем B для определения
условной вероятности B
заданной метки $l \in L^{\leq T}$

$$p(l|x) = \sum_{X_\pi \in B^{-1}(l)} p(\pi|x).$$



CTC loss



- Вероятность слова – сумма вероятностей по всем возможным путям
- $6^8 = 1\ 679\ 616$ – возможных путей (случай из примера)
- Используем динамическое программирование для нахождения вероятности целевой последовательности

Нахождение возможных путей

- Аналогично прямому проходу и проходам в НММ мы рассчитываем α и β

$$\alpha_t(s) \stackrel{\text{def}}{=} \sum_{\substack{\pi \in N^T : \\ \mathcal{B}(\pi_{1:t}) = \mathbf{l}_{1:s}}} \prod_{t'=1}^t y_{\pi_{t'}}^{t'}$$

- Суммарная вероятность всех путей, чей префикс заканчивается символом в позиции s в момент времени t от начала последовательности

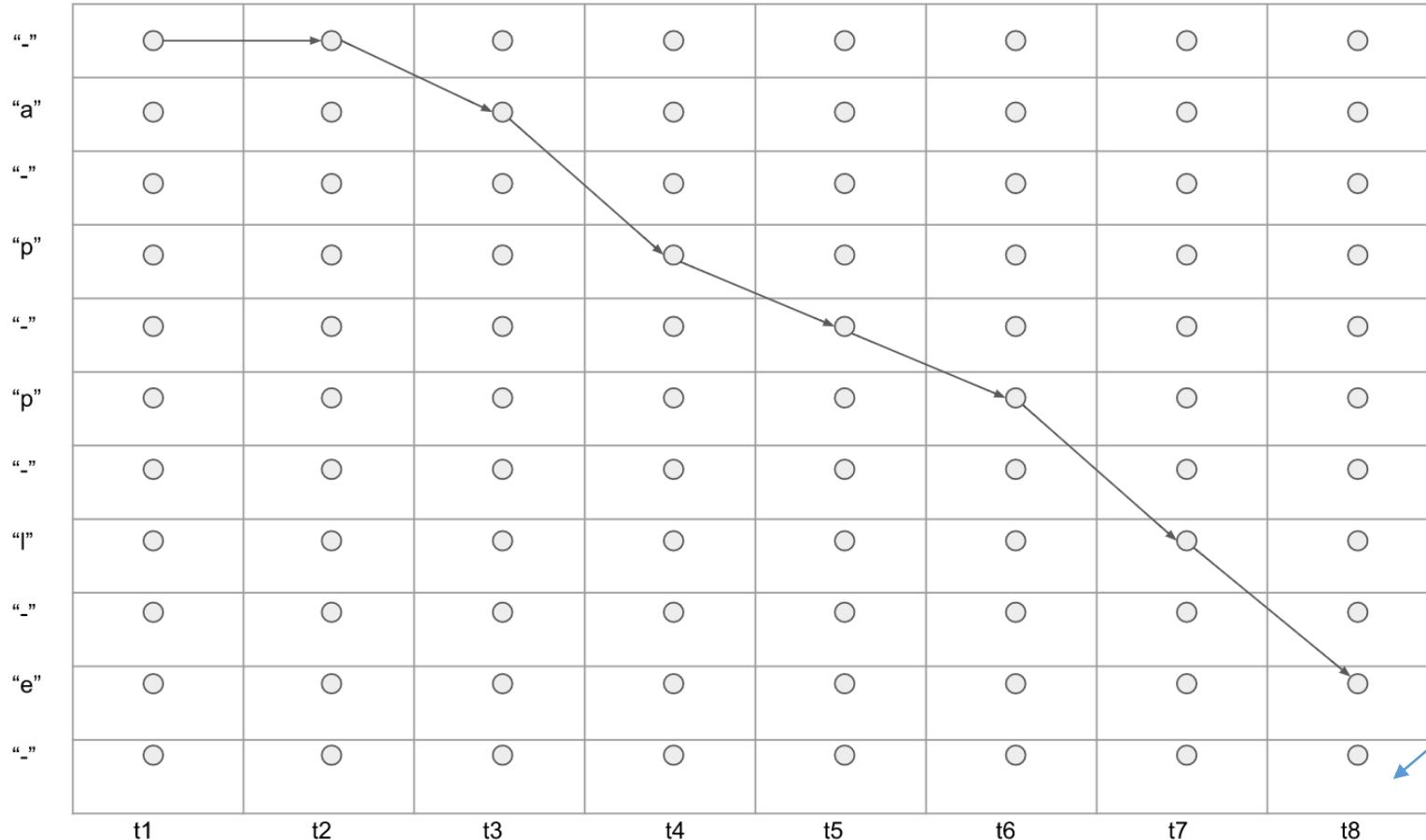
$$\beta_t(s) \stackrel{\text{def}}{=} \sum_{\substack{\pi \in N^T : \\ \mathcal{B}(\pi_{t:T}) = \mathbf{l}_{s:|\mathbf{l}|}}} \prod_{t'=t}^T y_{\pi_{t'}}^{t'}$$

- Суммарная вероятность всех путей, чей суффикс начинается с символом в позиции s в момент времени t

Нахождение возможных путей

Нахождение возможных путей

Пример, путь “—ар-ple” может быть отмапирован на маркер “apple” $B(\text{--ap-ple}) = \text{“apple”}$



Не обязательно
заканчиваем тут

Нахождение возможных путей

Пример: невозможный переход, нельзя предсказывать предыдущий символ.



Не обязательно
заканчиваем тут

Нахождение возможных путей

Инициализация

$$\alpha_1(1) = y_b^1$$

$$\alpha_1(2) = y_{\mathbf{l}_1}^1$$

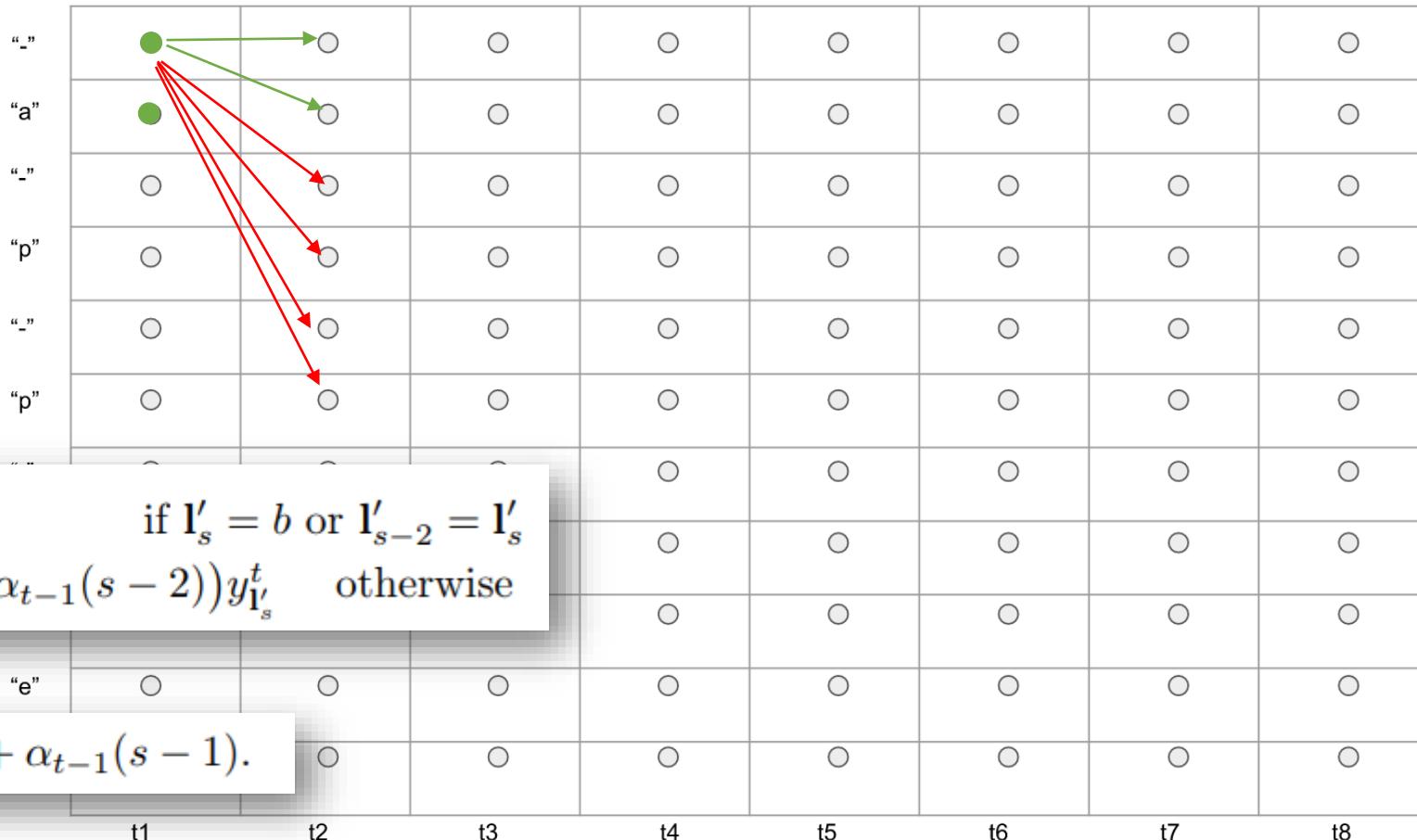
$$\alpha_1(s) = 0, \quad \forall s > 2$$

Начинаем либо b или с первого символа

	○	○	○	○	○	○	○	○
	○	○	○	○	○	○	○	○
	○	○	○	○	○	○	○	○
	○	○	○	○	○	○	○	○
	○	○	○	○	○	○	○	○
	○	○	○	○	○	○	○	○
	○	○	○	○	○	○	○	○
	○	○	○	○	○	○	○	○
	○	○	○	○	○	○	○	○
	○	○	○	○	○	○	○	○
t1	t2	t3	t4	t5	t6	t7	t8	

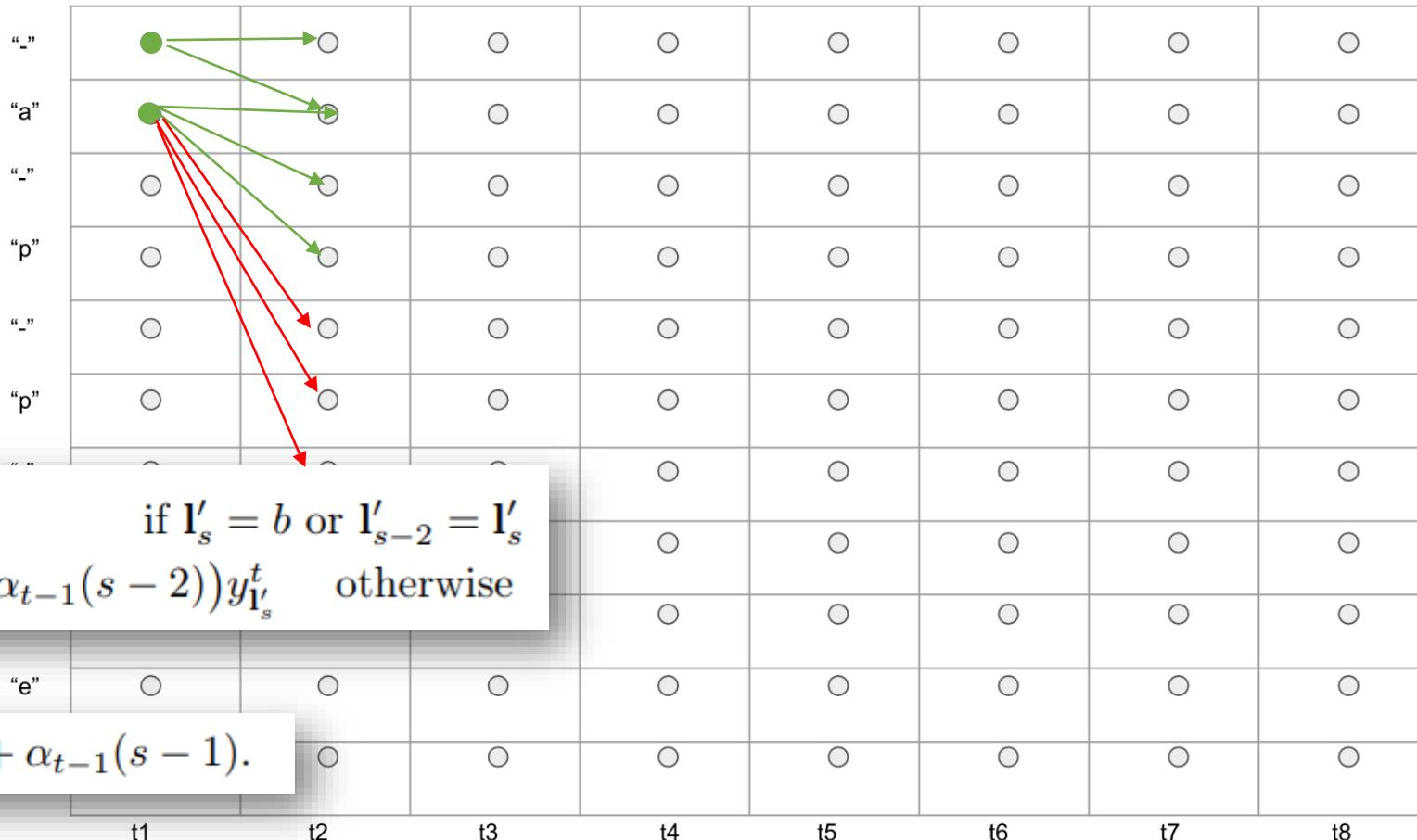
Нахождение возможных путей

Куда можно двигаться из этих точек



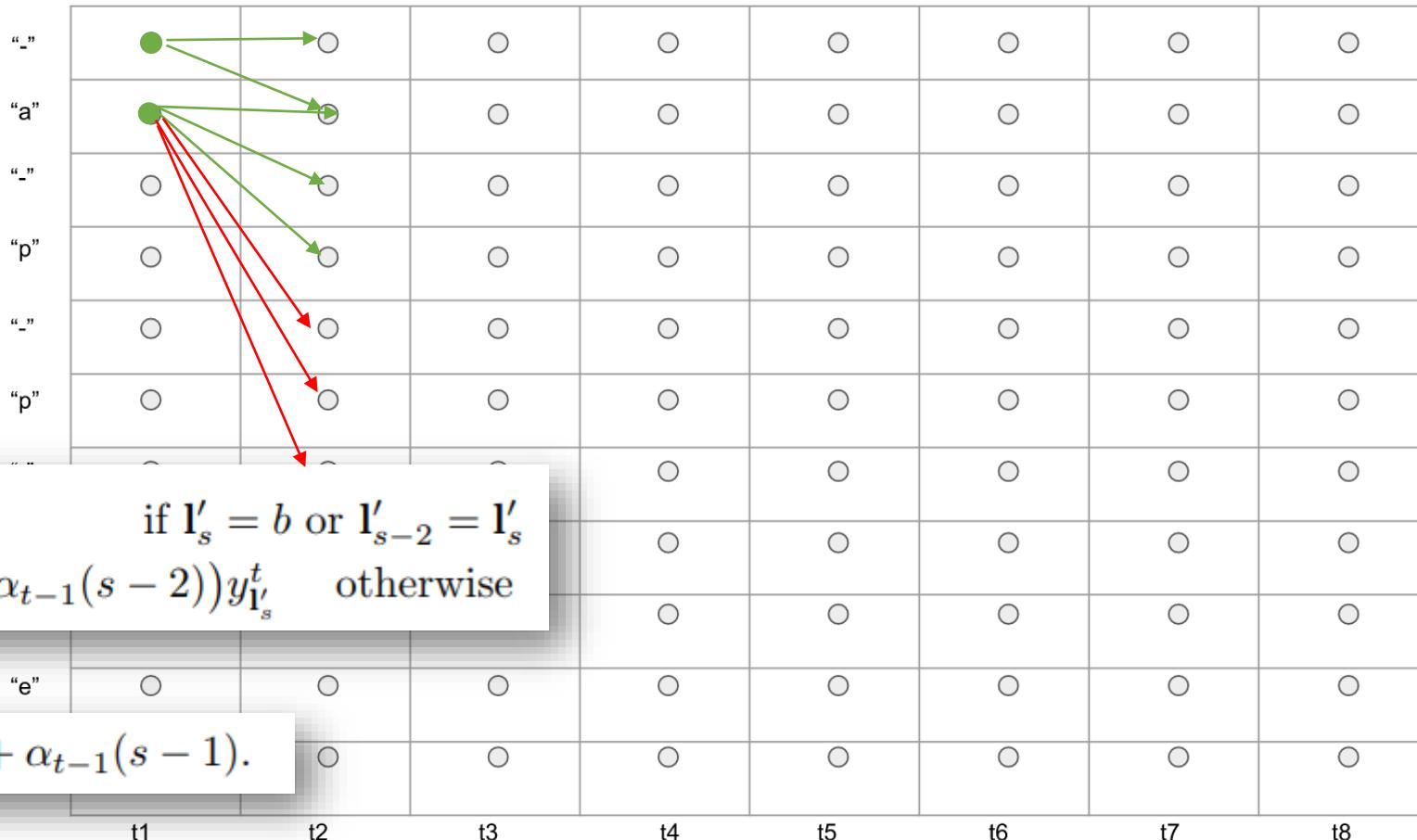
Нахождение возможных путей

Куда можно двигаться из этих точек

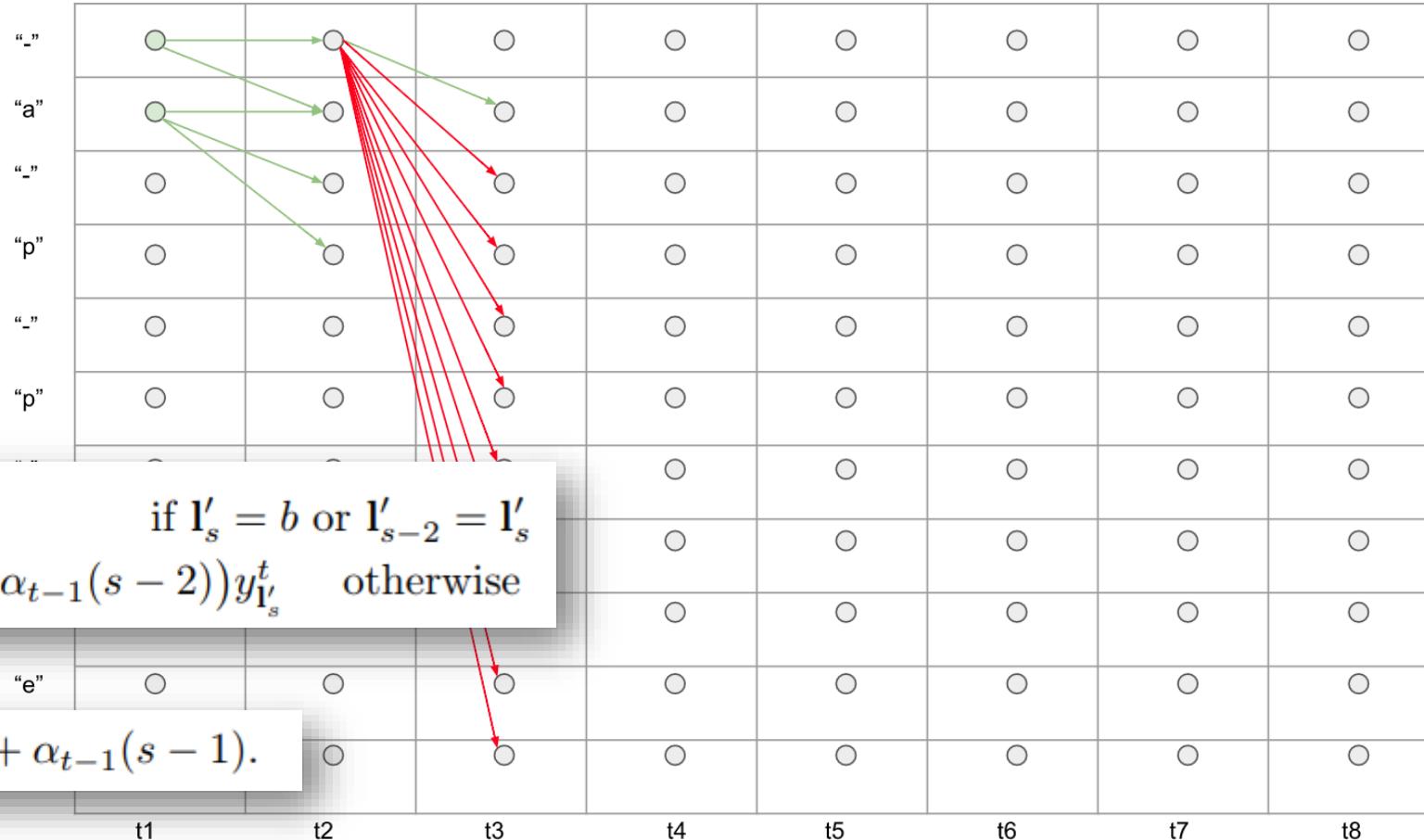


Нахождение возможных путей

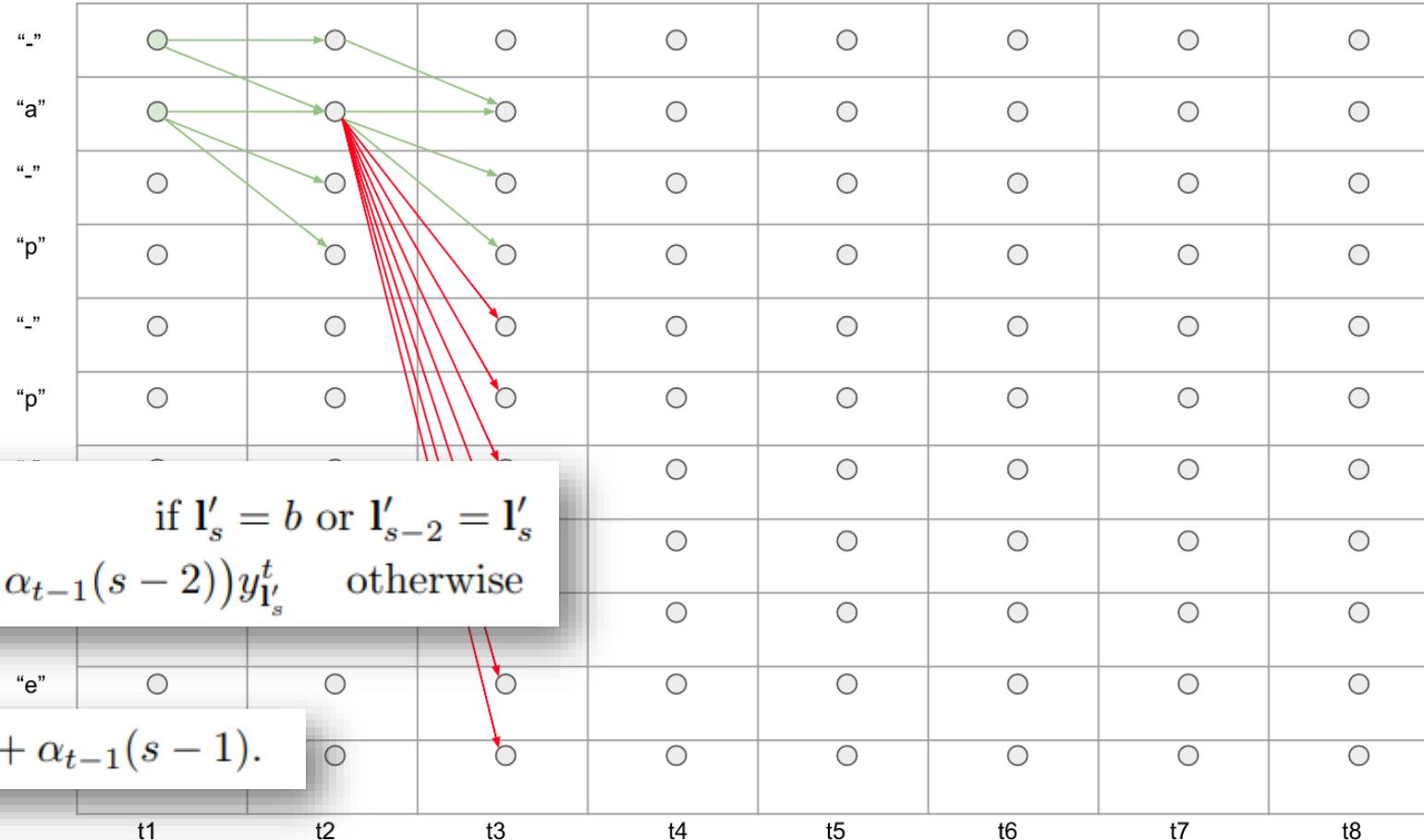
Куда можно двигаться из этих точек



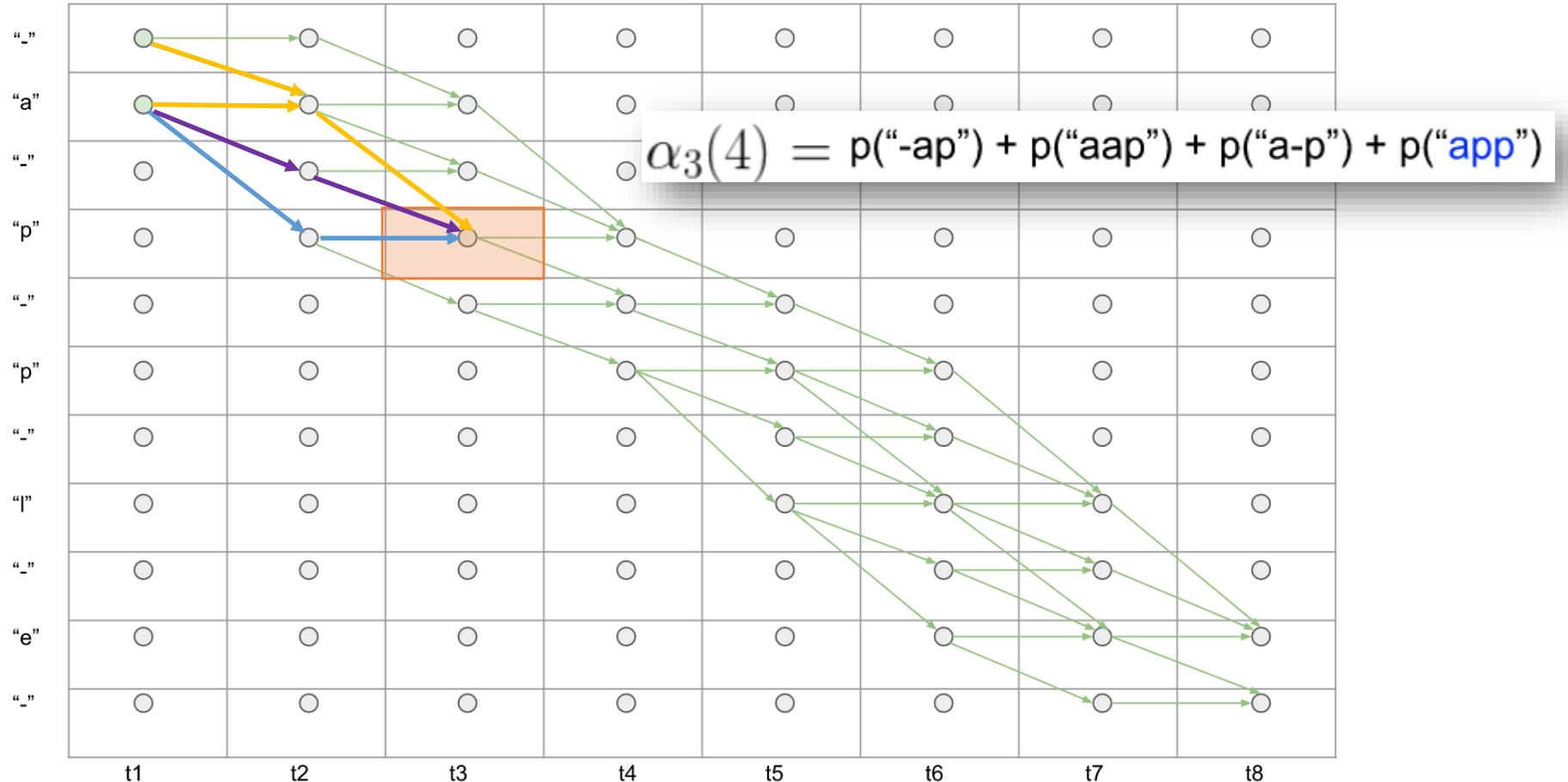
Нахождение возможных путей



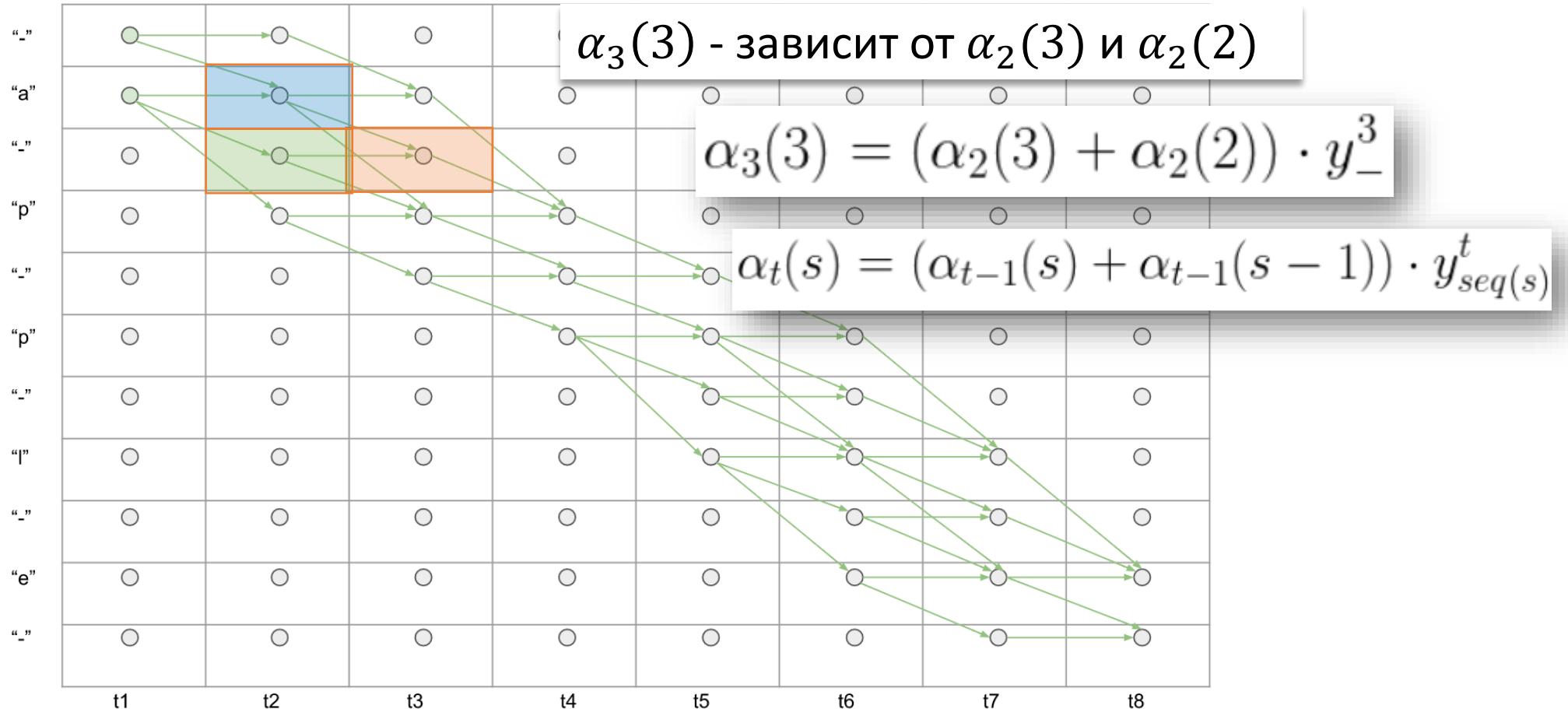
Нахождение возможных путей



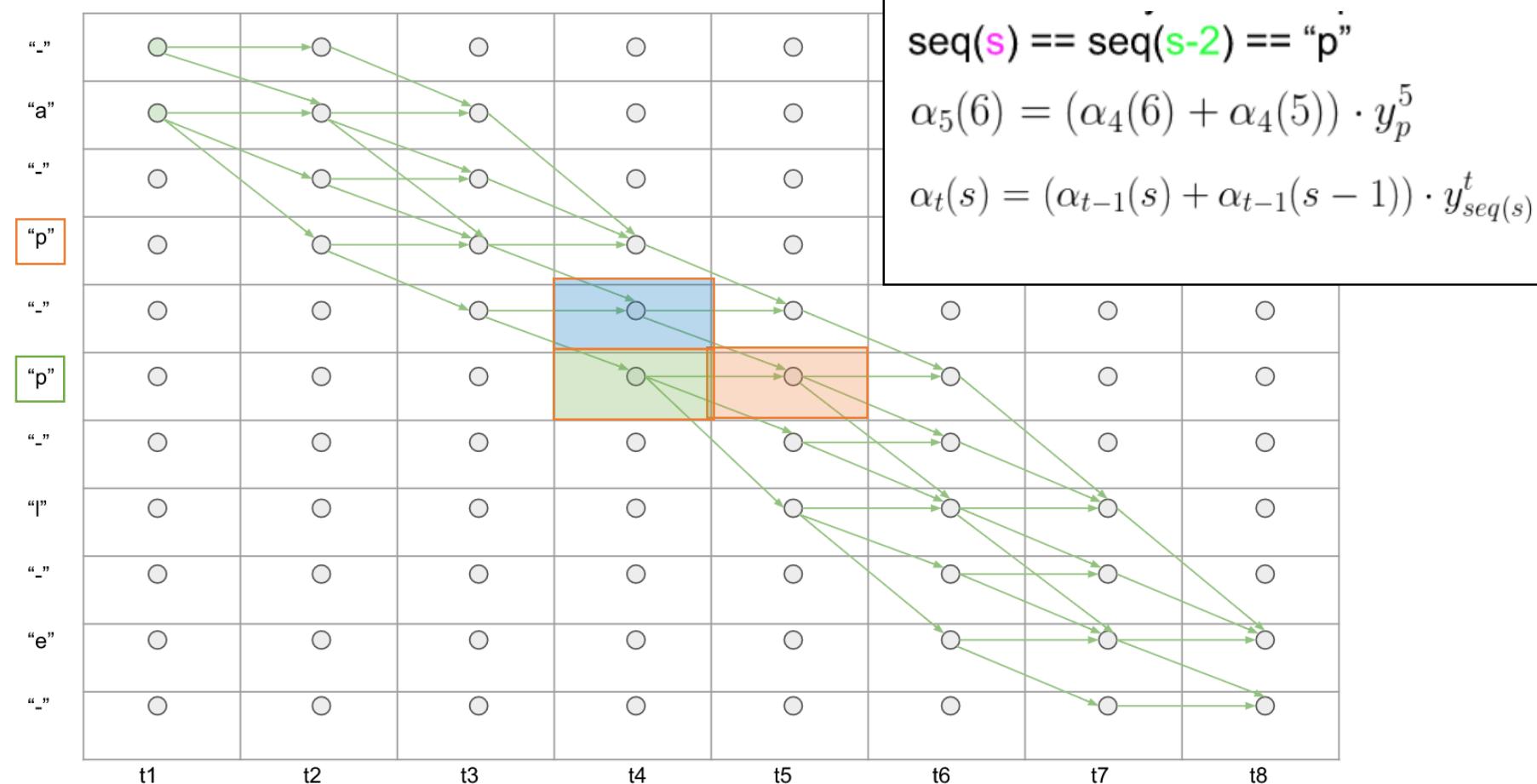
Нахождение возможных путей



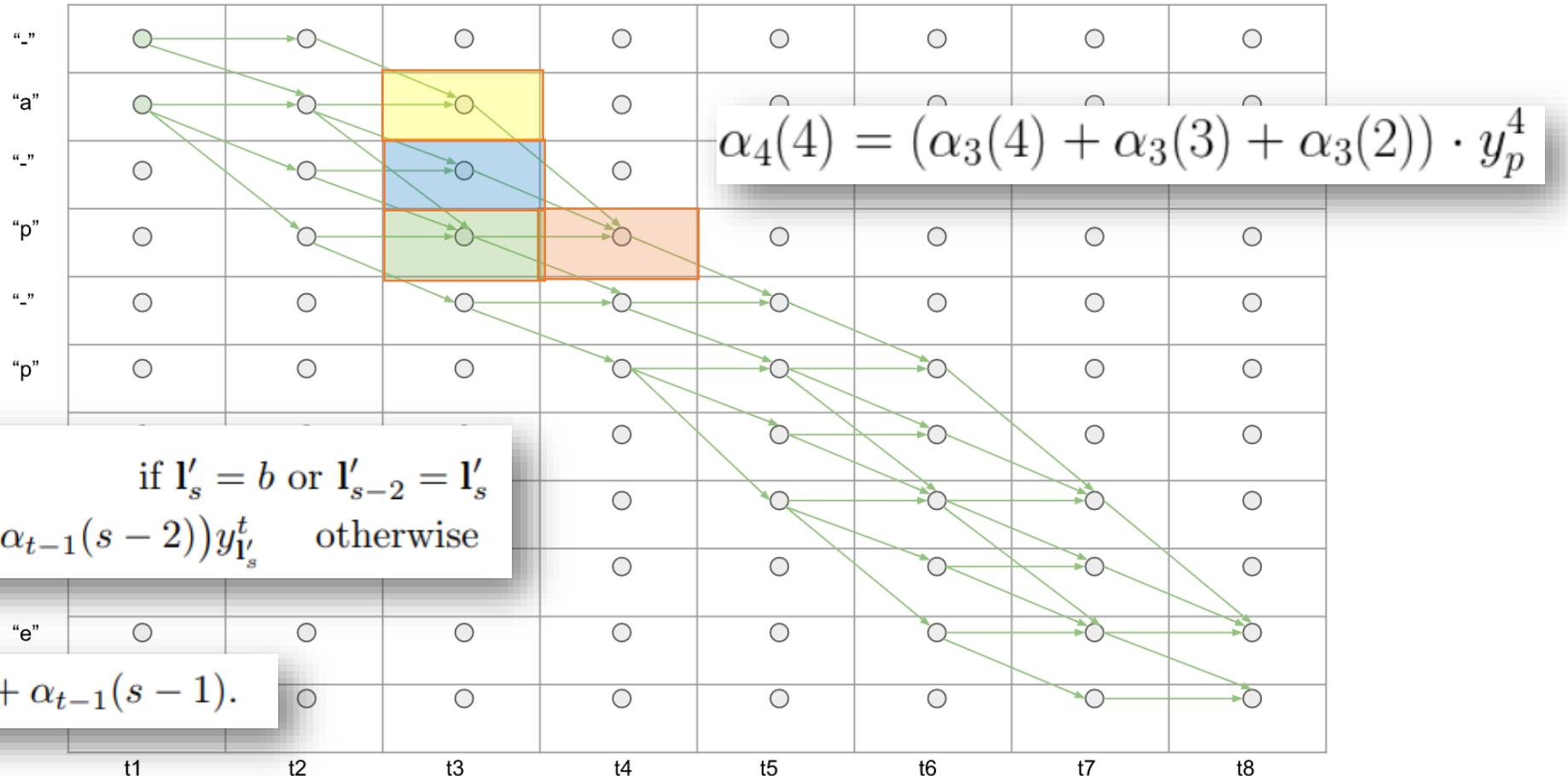
Нахождение возможных путей



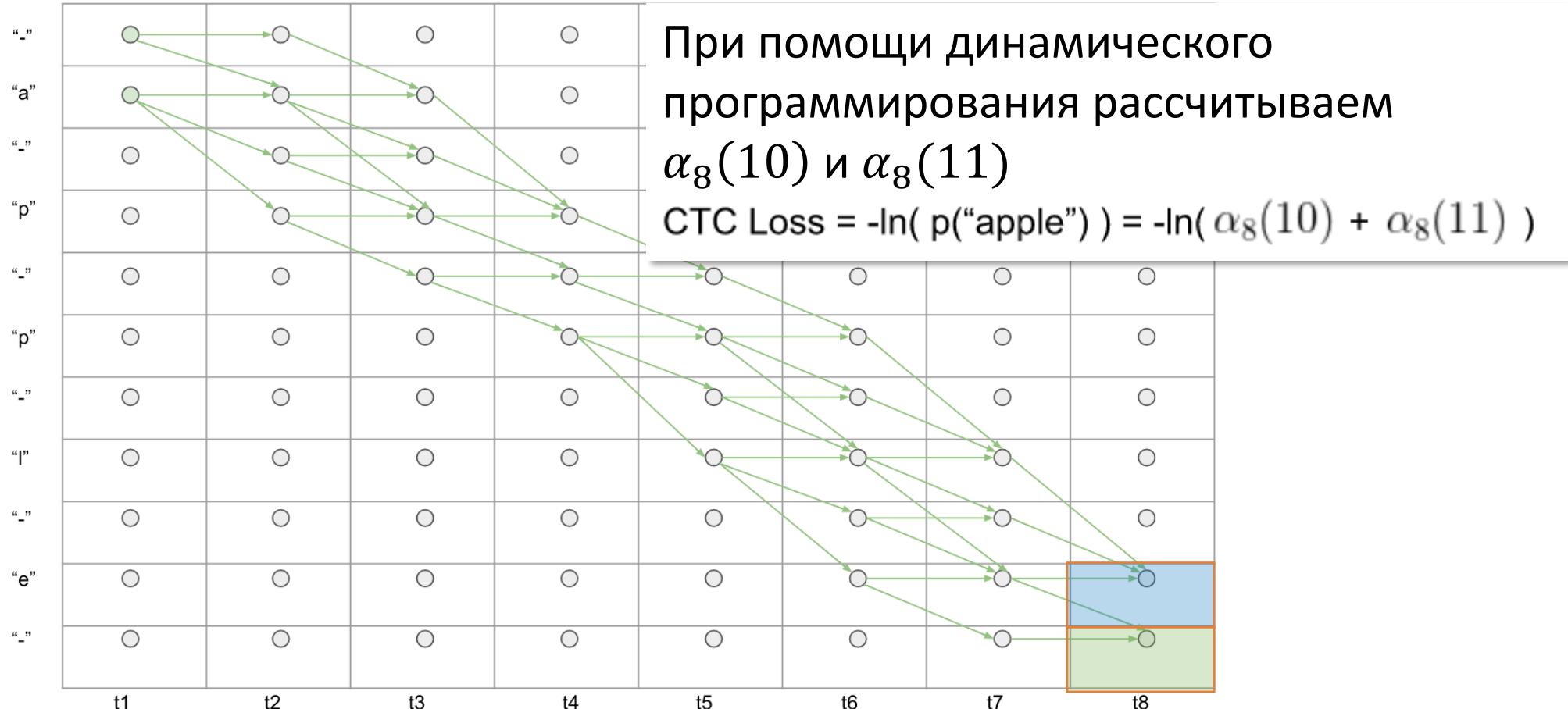
Нахождение возможных путей



Нахождение возможных путей



Нахождение возможных путей



Нахождение возможных путей. Обратный проход

- Делается аналогично прямому проходу для каждого момента времени и метки рассчитываем вероятность

$$\beta_t(s) \stackrel{\text{def}}{=} \sum_{\substack{\pi \in N^T : \\ \mathcal{B}(\pi_{t:T}) = \mathbf{l}_{s:|\mathbf{l}|}}} \prod_{t'=t}^T y_{\pi_{t'}}^{t'}$$

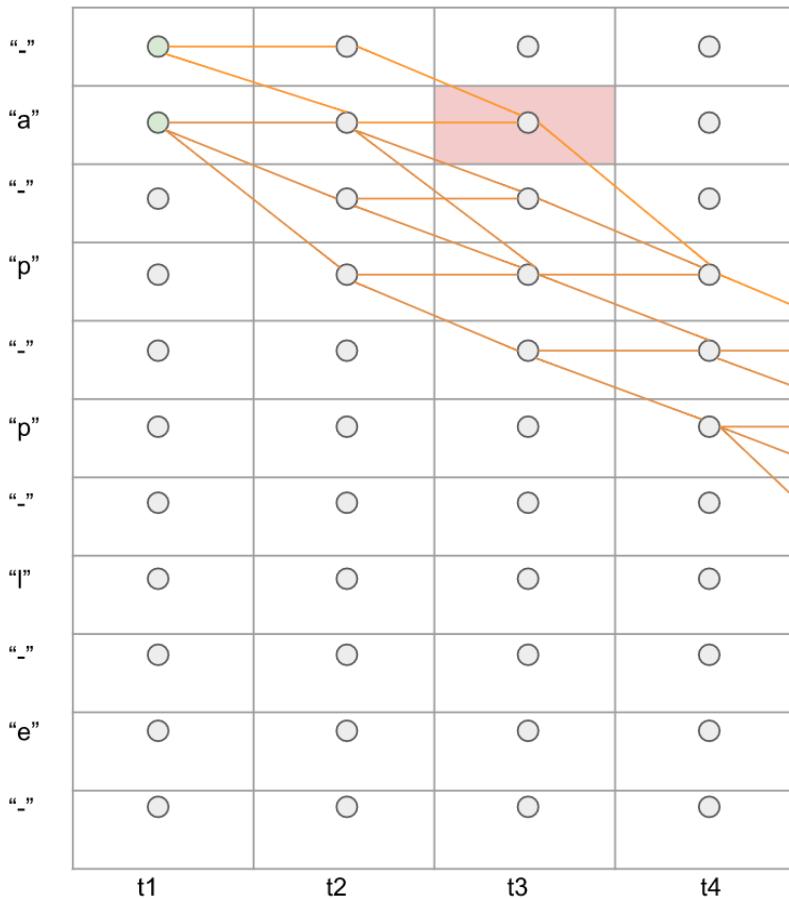
$$\beta_T(|\mathbf{l}'|) = y_b^T$$

$$\beta_T(|\mathbf{l}'| - 1) = y_{\mathbf{l}'_{[1]}}^T$$

$$\beta_T(s) = 0, \forall s < |\mathbf{l}'| - 1$$

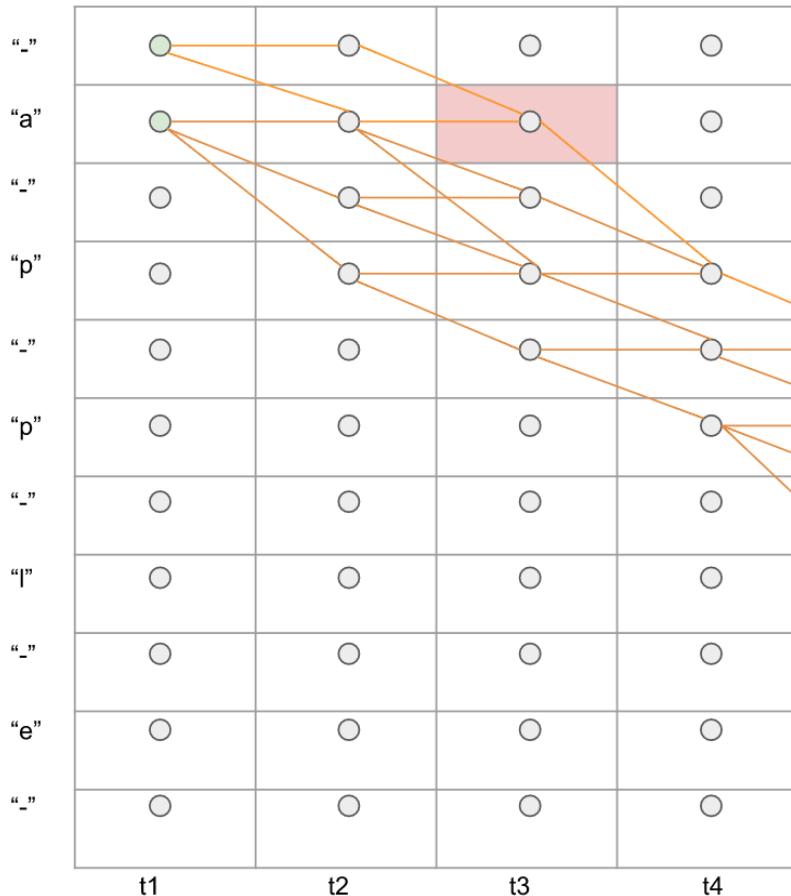
$$\beta_t(s) = \begin{cases} \bar{\beta}_t(s) y_{\mathbf{l}'_s}^t & \text{if } \mathbf{l}'_s = b \text{ or } \mathbf{l}'_{s+2} = \mathbf{l}'_s \\ (\bar{\beta}_t(s) + \beta_{t+1}(s+2)) y_{\mathbf{l}'_s}^t & \text{otherwise} \end{cases} \quad \bar{\beta}_t(s) \stackrel{\text{def}}{=} \beta_{t+1}(s) + \beta_{t+1}(s+1).$$

Вероятность путей для отдельной метки



$$\begin{aligned}
 \alpha_3(2) &= p("aa") + p("-aa") + p("aaa") = \\
 &= y_-^1 \cdot y_-^2 \cdot y_a^3 + y_-^1 \cdot y_a^2 \cdot y_a^3 + y_a^1 \cdot y_a^2 \cdot y_a^3 \\
 \beta_3(2) &= p("ap-ple") = y_a^3 \cdot y_p^4 \cdot y_-^5 \cdot y_p^6 \cdot y_l^7 \cdot y_e^8 \\
 \Rightarrow \alpha_3(2) \cdot \beta_3(2) &= y_-^1 \cdot y_-^2 \cdot y_a^3 \cdot y_a^3 \cdot y_p^4 \cdot y_-^5 \cdot y_p^6 \cdot y_l^7 \cdot y_e^8 + \\
 &+ y_-^1 \cdot y_a^2 \cdot y_a^3 \cdot y_a^3 \cdot y_p^4 \cdot y_-^5 \cdot y_p^6 \cdot y_l^7 \cdot y_e^8 \\
 &+ y_a^1 \cdot y_a^2 \cdot y_a^3 \cdot y_a^3 \cdot y_p^4 \cdot y_-^5 \cdot y_p^6 \cdot y_l^7 \cdot y_e^8
 \end{aligned}$$

Вероятность путей для отдельной метки

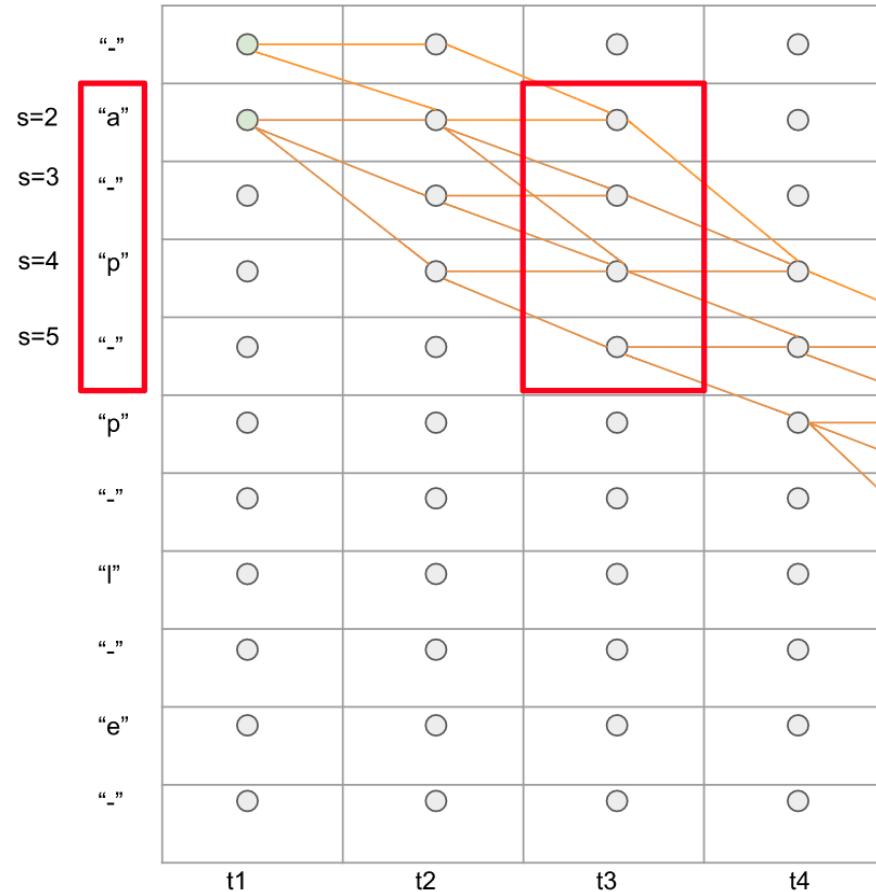


$$\alpha_3(2) \cdot \beta_3(2) = y_{-}^1 \cdot y_{-}^2 \cdot y_a^3 \cdot \boxed{y_a^3} \cdot y_p^4 \cdot y_{-}^5 \cdot y_p^6 \cdot y_l^7 \cdot y_e^8 + \\ + y_{-}^1 \cdot y_a^2 \cdot y_a^3 \cdot \boxed{y_a^3} \cdot y_p^4 \cdot y_{-}^5 \cdot y_p^6 \cdot y_l^7 \cdot y_e^8 \\ + y_a^1 \cdot y_a^2 \cdot y_a^3 \cdot \boxed{y_a^3} \cdot y_p^4 \cdot y_{-}^5 \cdot y_p^6 \cdot y_l^7 \cdot y_e^8 =$$

$$= (p("--aap-ple") + p("-aap-ple") + p("aaap-ple")) * y_a^3$$

$$\frac{\alpha_3(2) \cdot \beta_3(2)}{y_a^3} = (p("--aap-ple") + p("-aap-ple") + p("aaap-ple"))$$

Вероятность слова для времени



Суммарная вероятность
всех путей на шаге 3

$$p("apple") = \sum_{s=2}^5 \frac{\alpha_3(s) \cdot \beta_3(s)}{y_{seq(s)}^3}$$

Вероятность в любой
момент времени

$$p("apple") = \sum_{s=1}^{|seq|} \frac{\alpha_t(s) \cdot \beta_t(s)}{y_{seq(s)}^t}$$

$$\text{CTC Loss} = -\ln(p(\text{"apple"}))$$

Градиент

$$p(\text{"apple"}) = \sum_{s=1}^{|seq|} \frac{\alpha_t(s) \cdot \beta_t(s)}{y_{seq(s)}^t}$$

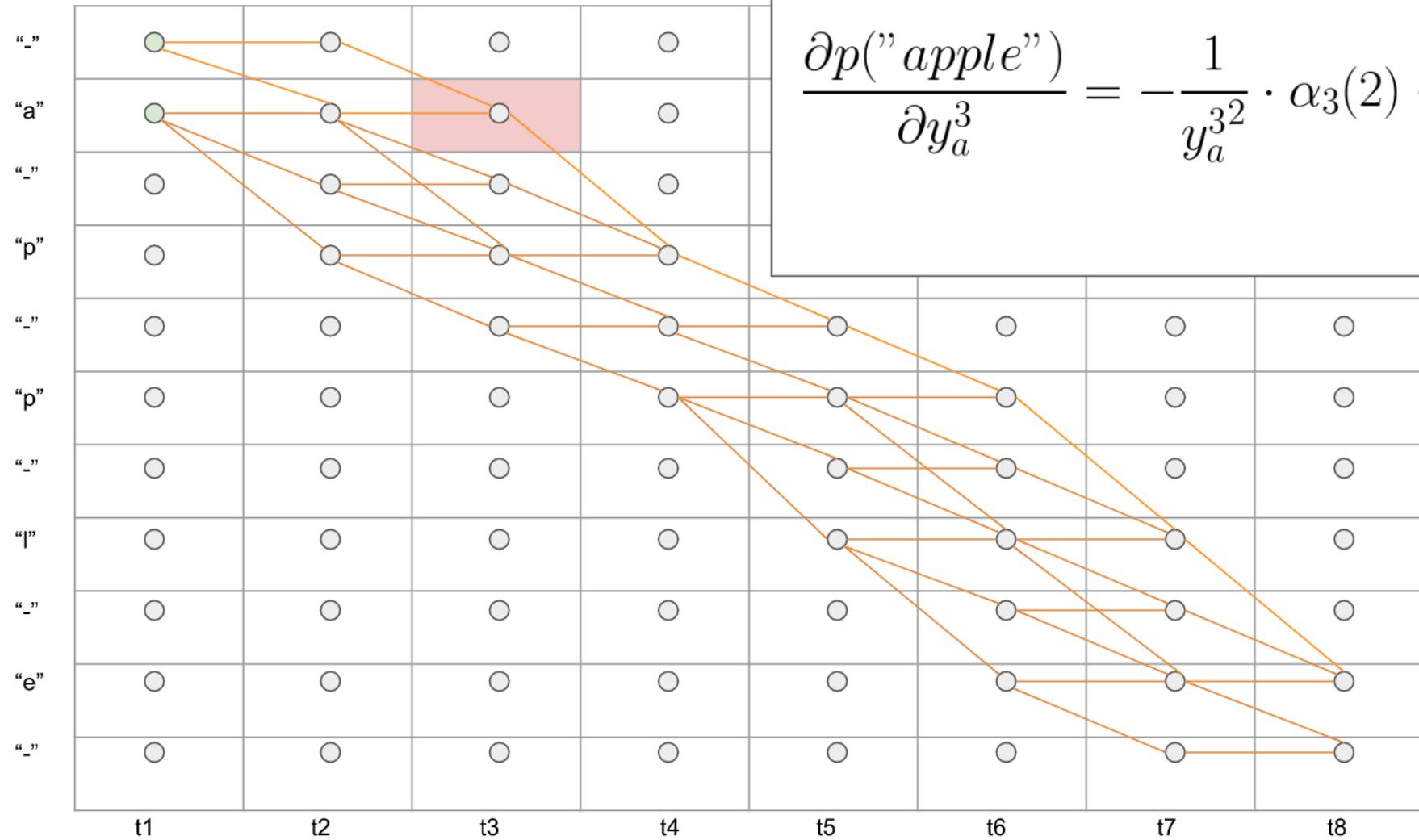
- Принимаем во внимание, что

$$\frac{\partial(-\ln(p(\text{"apple"})))}{\partial y_k^t} = -\frac{1}{p(\text{"apple"})} \cdot \boxed{\frac{\partial p(\text{"apple"})}{\partial y_k^t}}$$

- Чтобы обучить сеть нам нужно продифференцировать функцию ошибки по всем выходам сети y_k^t
- Рассматриваем только пути идущие через символ k в момент t

$$\frac{\partial p(\text{"apple"})}{\partial y_k^t} = -\frac{1}{y_k^{t^2}} \cdot \sum_{s:seq(s)=k} \alpha_t(s) \cdot \beta_t(s)$$

Градиент



Градиент

