

Фаза 2 • Неделя 3 • Понедельник

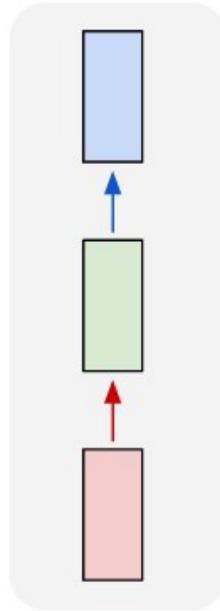
Рекуррентные нейронные сети • Recurrent Neural Networks

# Сегодня

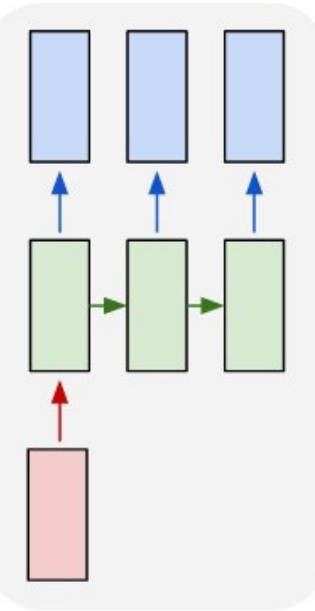
- сфера применения рекуррентных сетей
- архитектура рекуррентных слоев
  - Recurrent Neural Cell, Long Short Term Memory Cell, Gated Recurrent Unit Cell
- embedding-слой
- реализация сети-классификатора текстов

# Рекуррентные нейронные сети

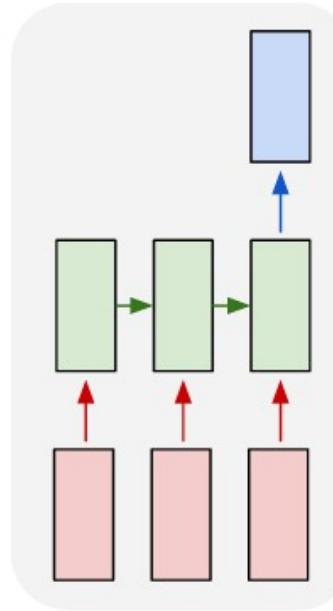
one to one



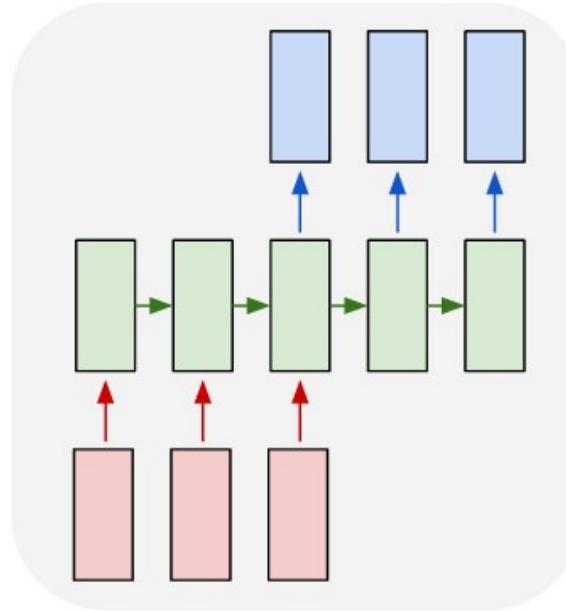
one to many



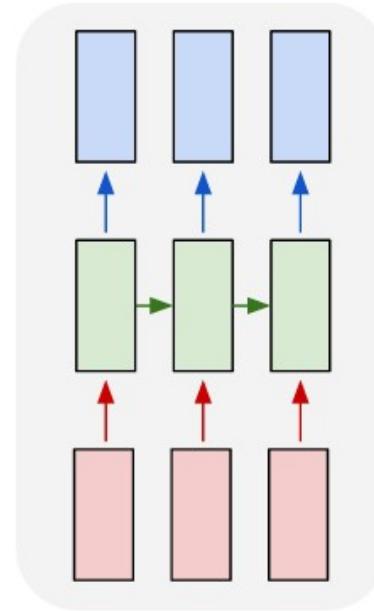
many to one



many to many



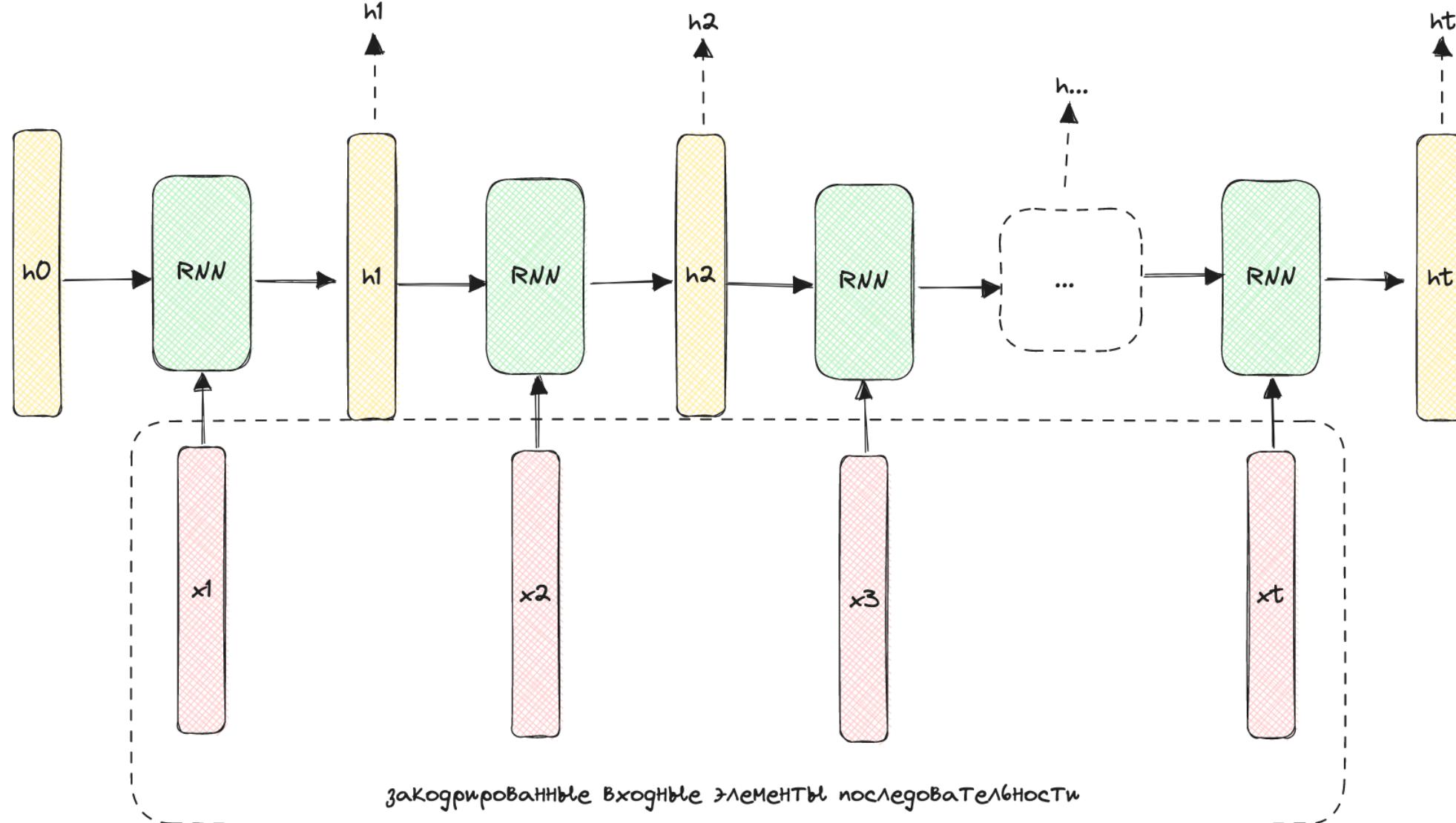
many to many



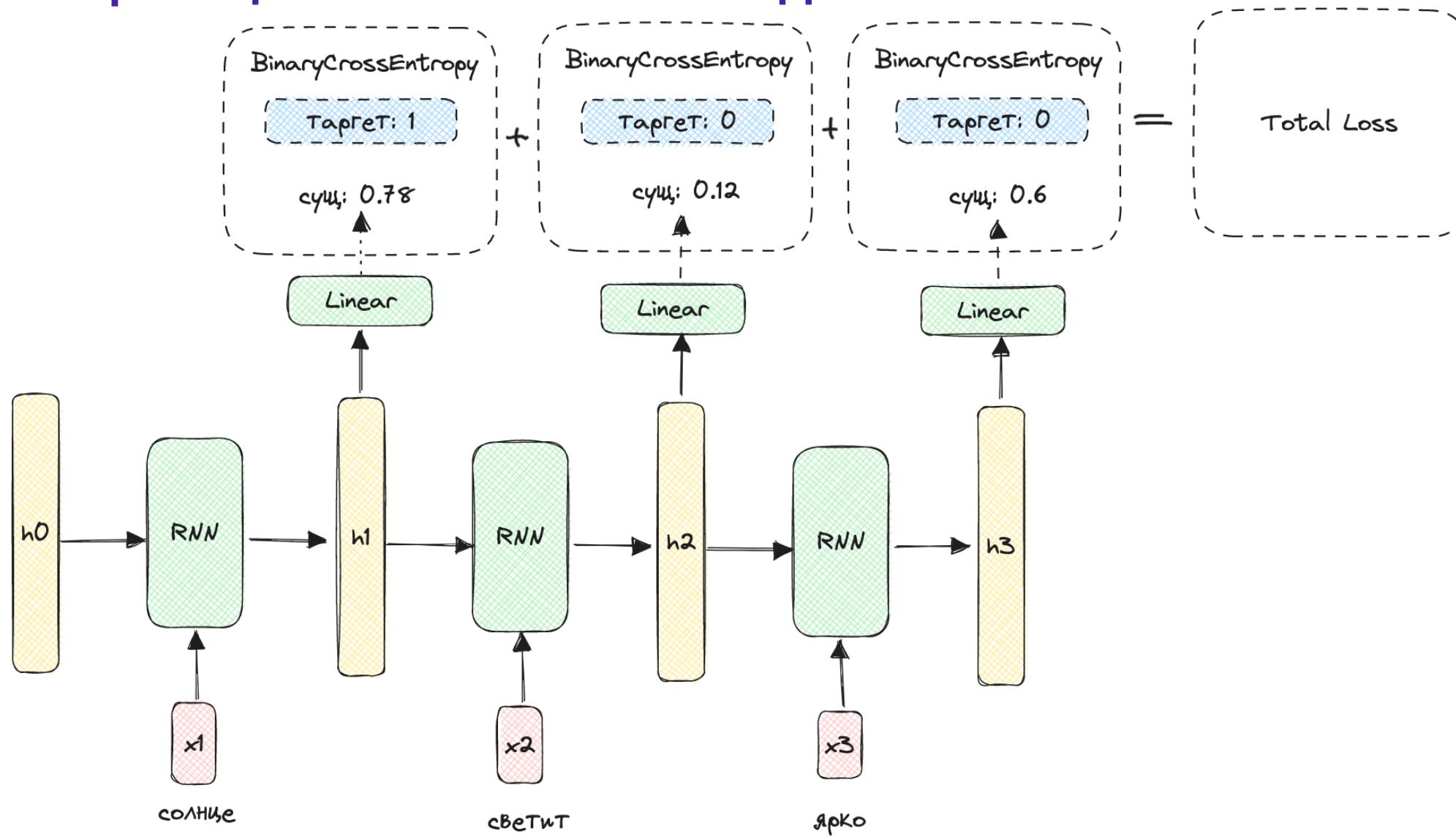
# Применения RNN

- Генерация текста
- Классификация текста
- Определение частей речи / POS (part of speech) tagging 
- Обнаружение именованных сущностей / NER (named entity recognition) 
- Seq2seq подход / sequence to sequence 

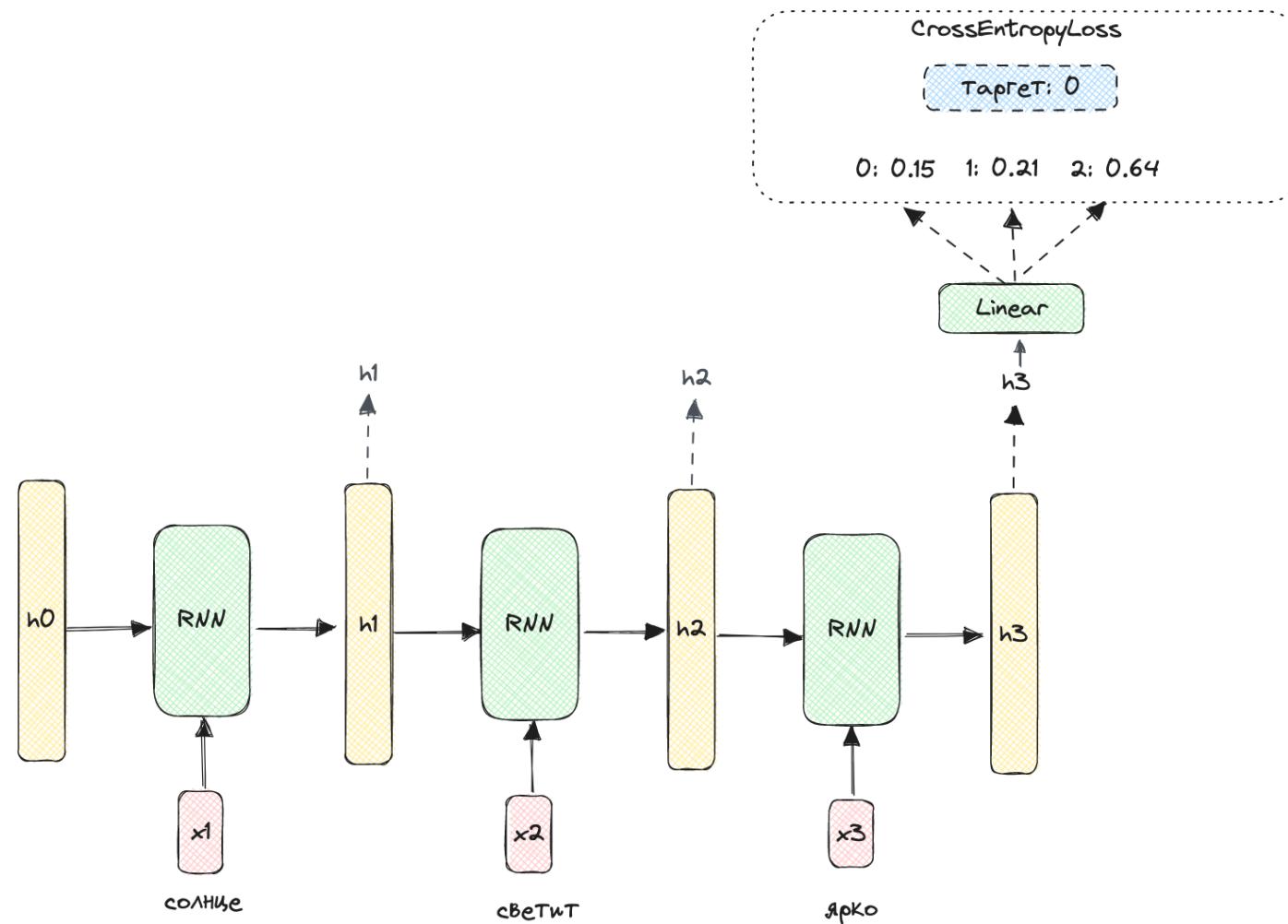
# Общая схема работы



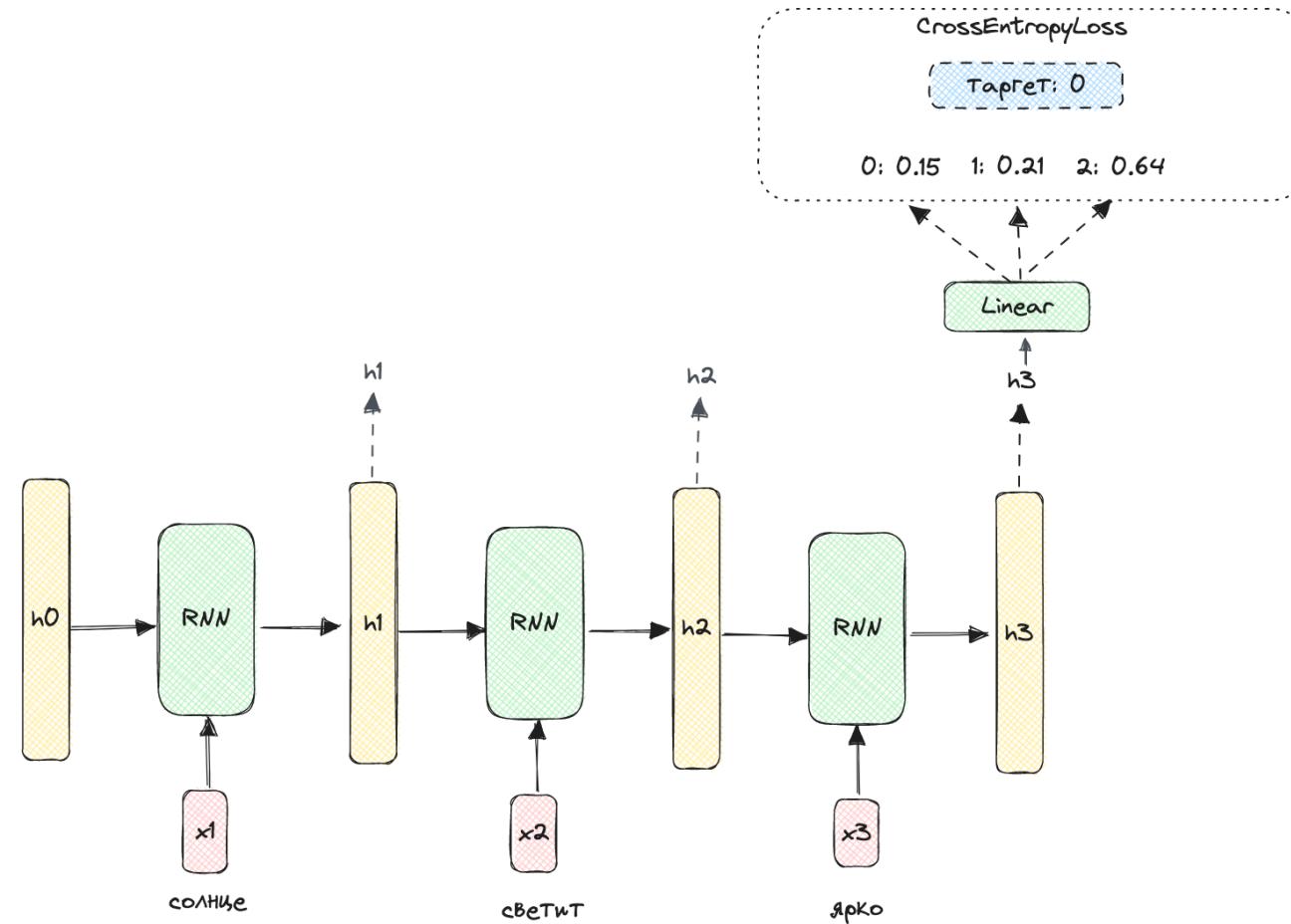
# Классификация элементов последовательности



# Классификация последовательности

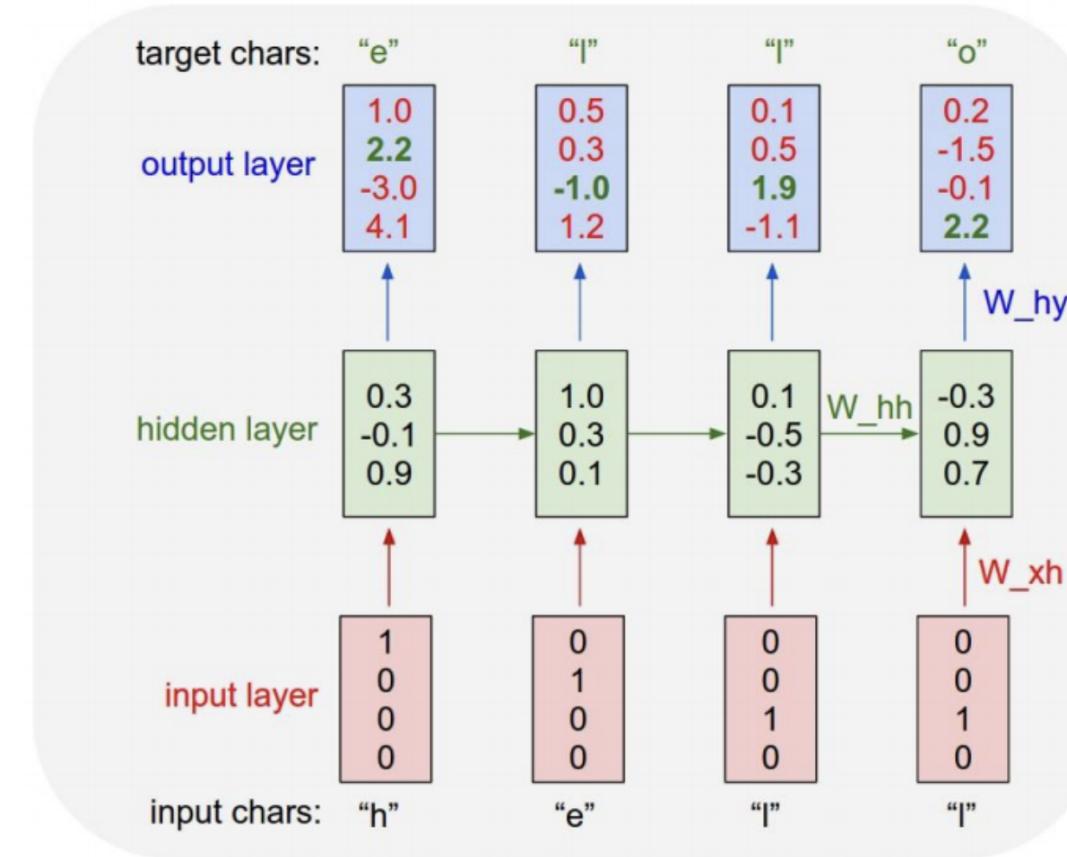


# Классификация последовательности

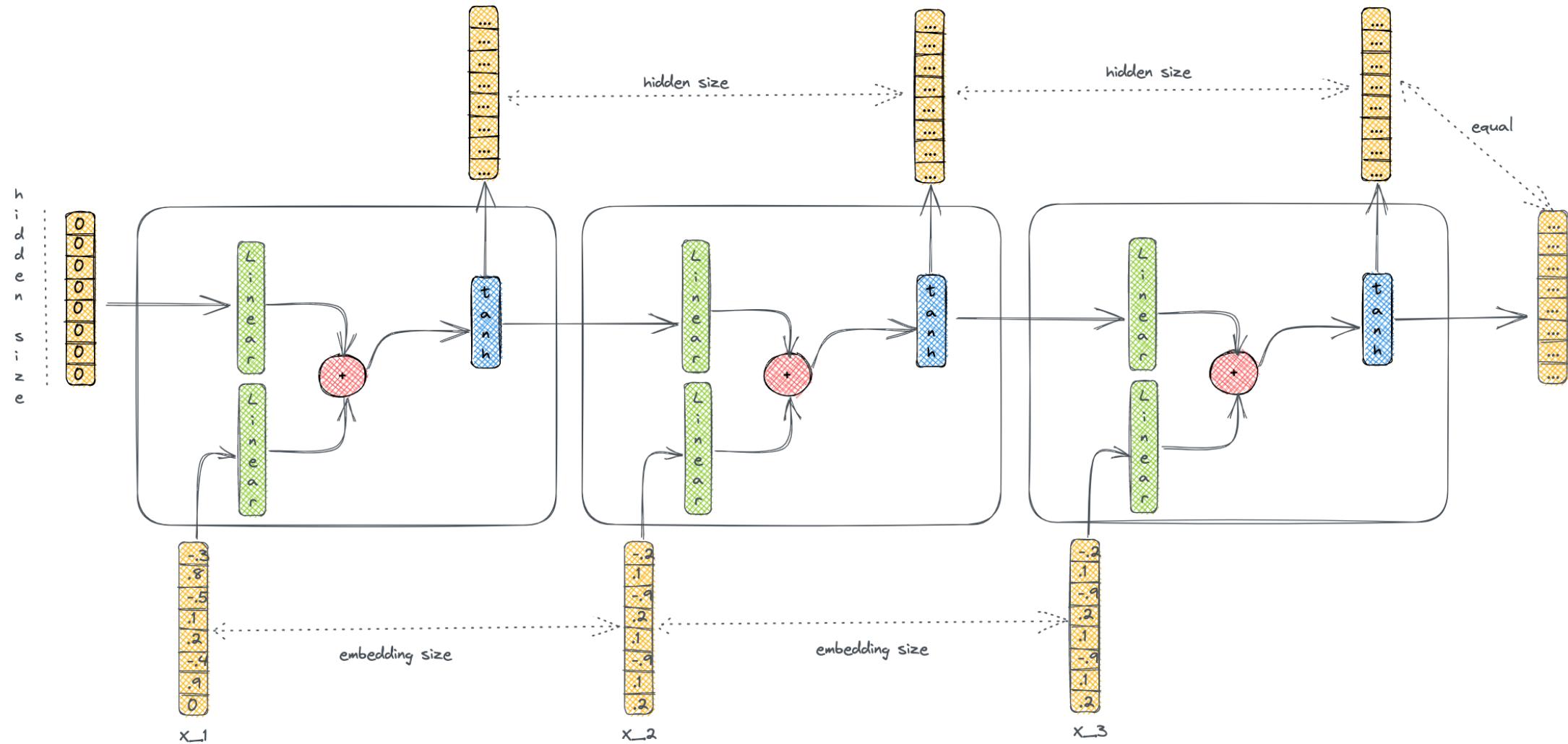


Тут можно и объединить выходы со всех ячеек (все  $h_n$  векторов), об этом чуть позже.

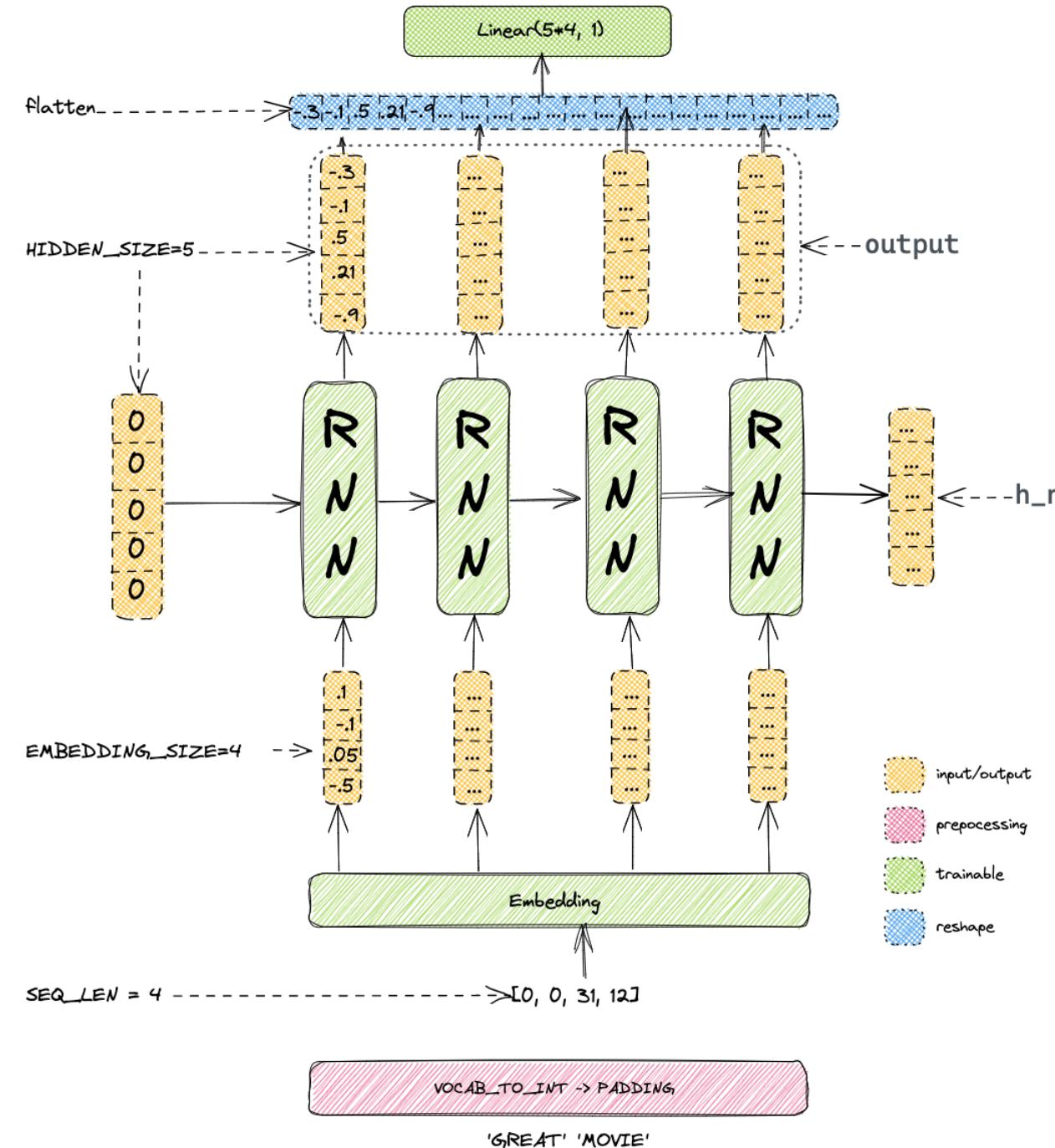
# Генерация • Character-level model



# Обычная рекуррентная ячейка



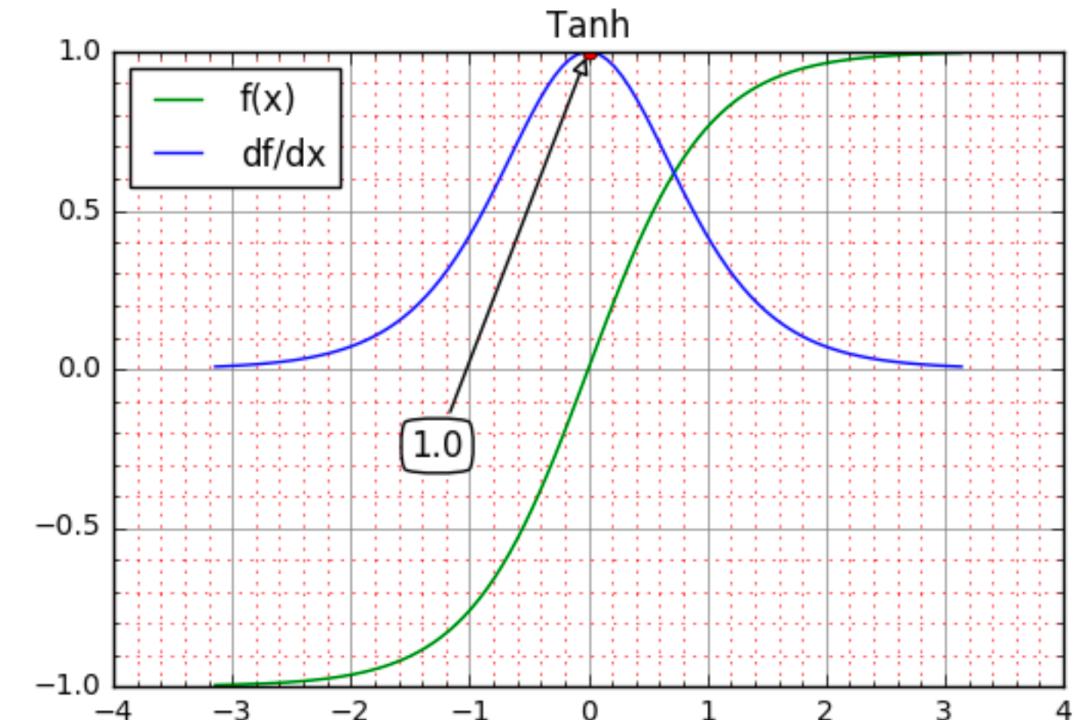
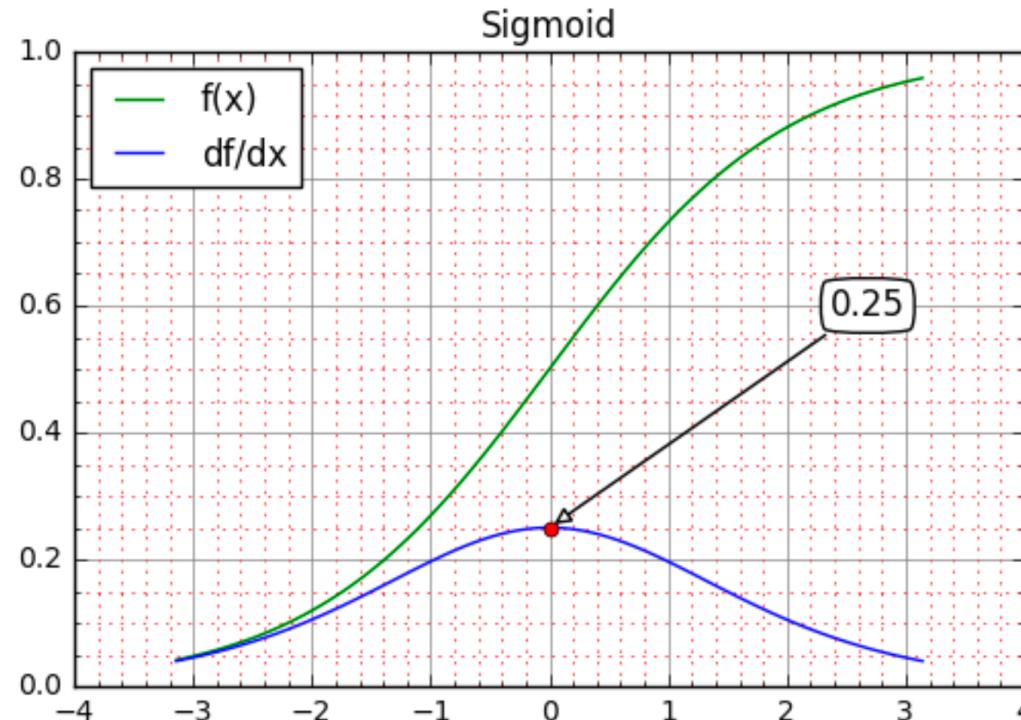
# Общая схема



# Проблемы RNN

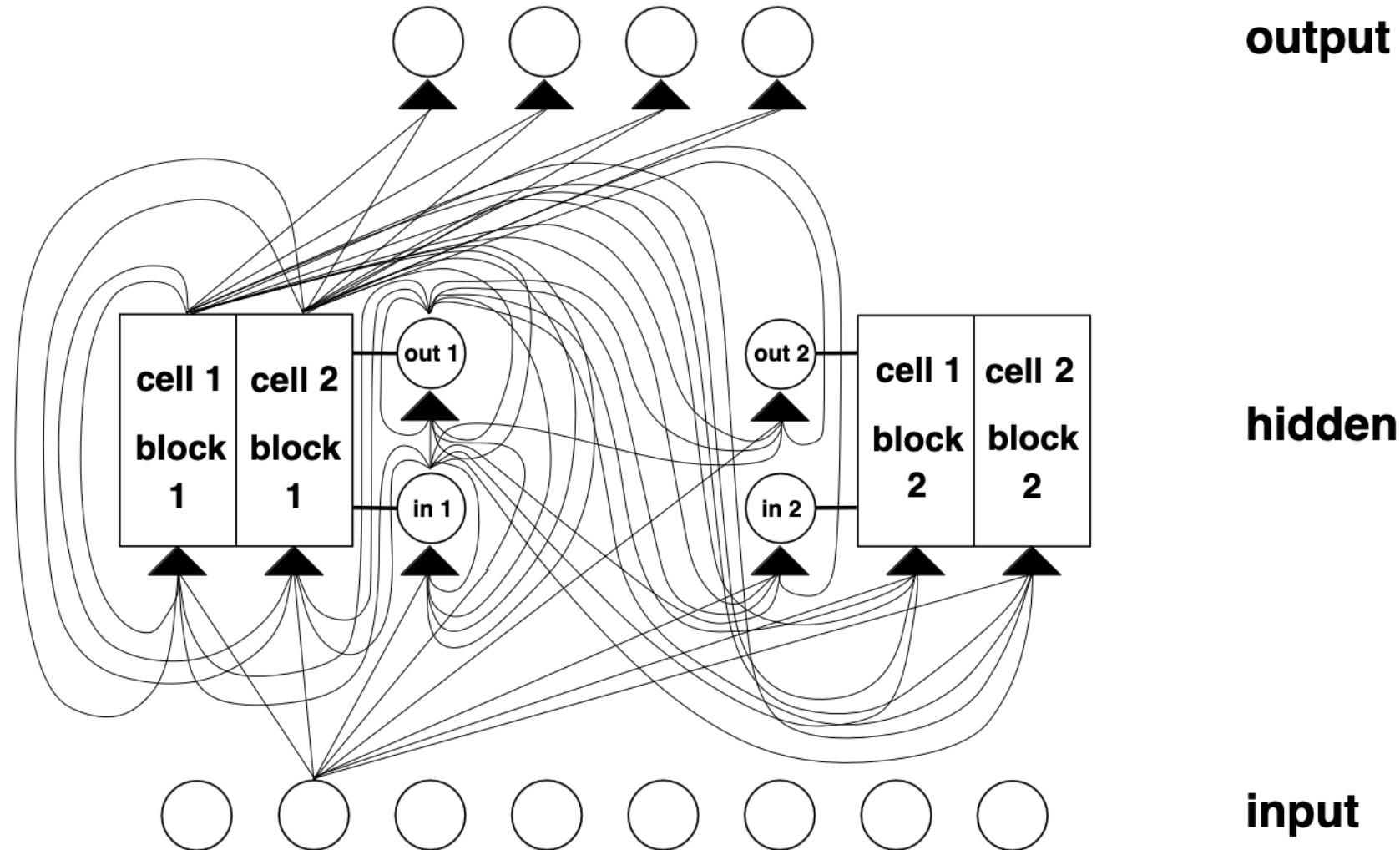
- Короткая память (быстро теряет контекст)
- Затухающие градиенты
- Развитие – LSTM-архитектура

# RNN • Взрывающиеся и затухающие градиенты

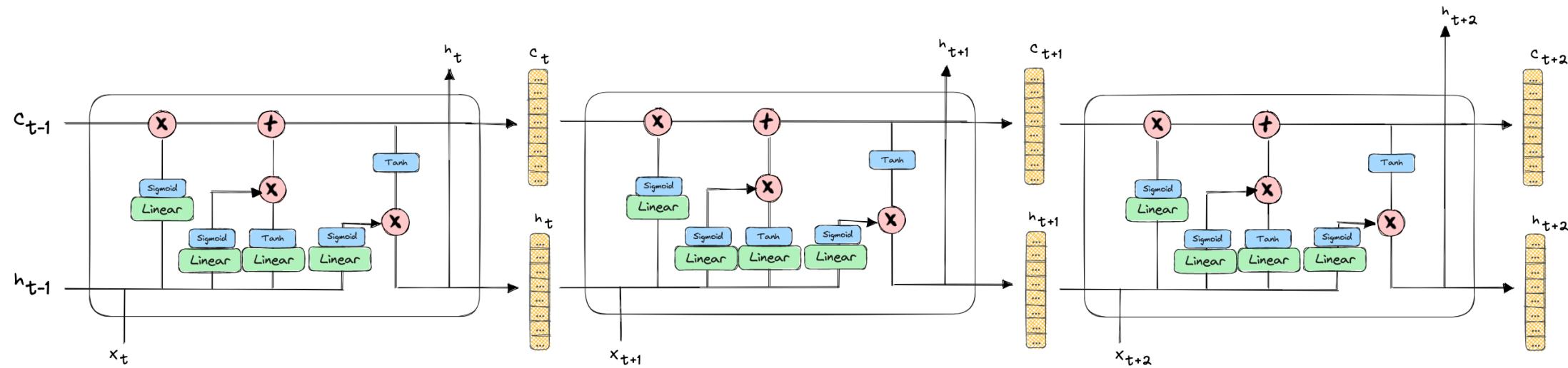


# LSTM • Long Short Term Memory

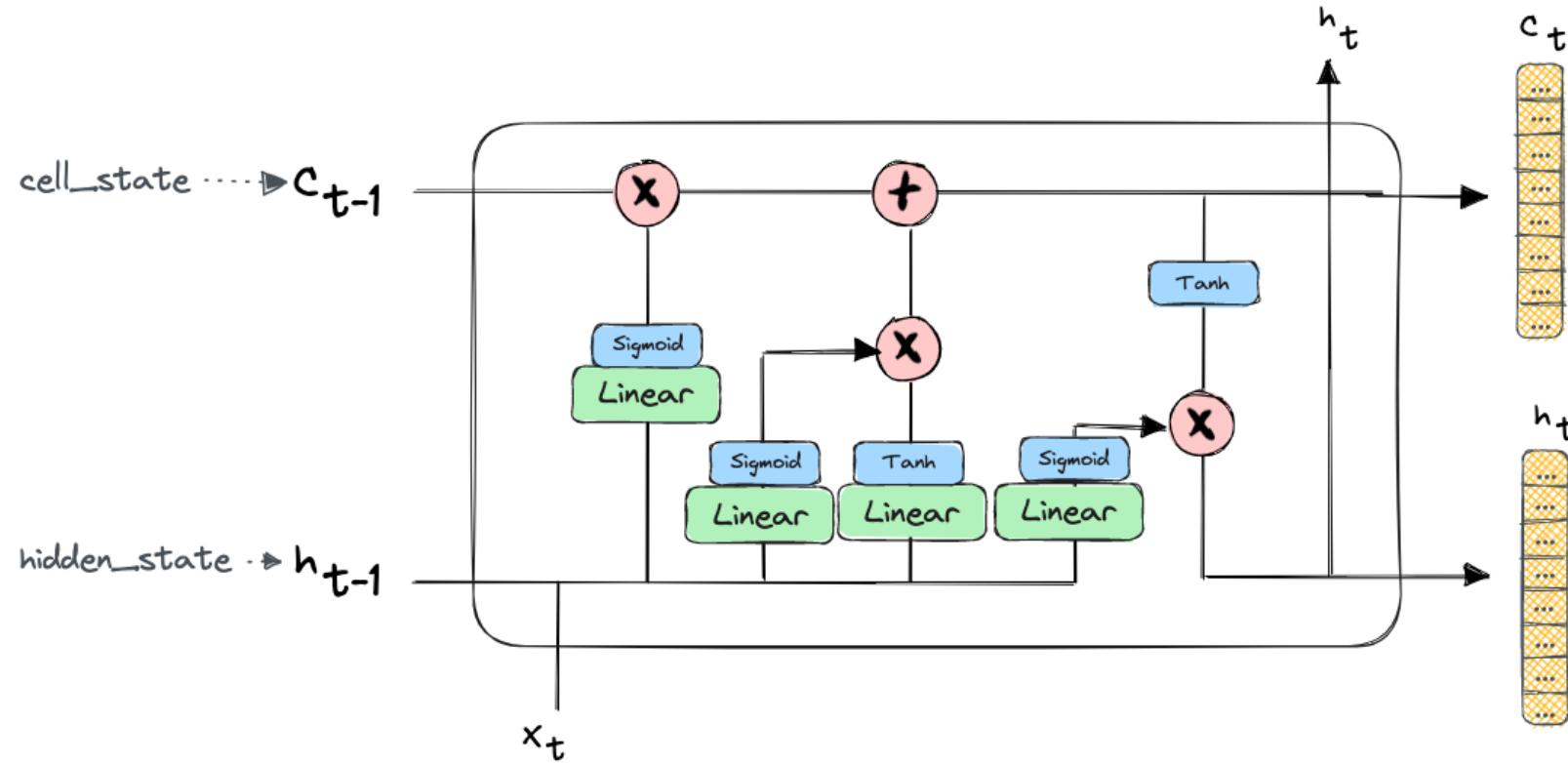
# Картина, которая все объясняет



# LSTM • Long short-term memory

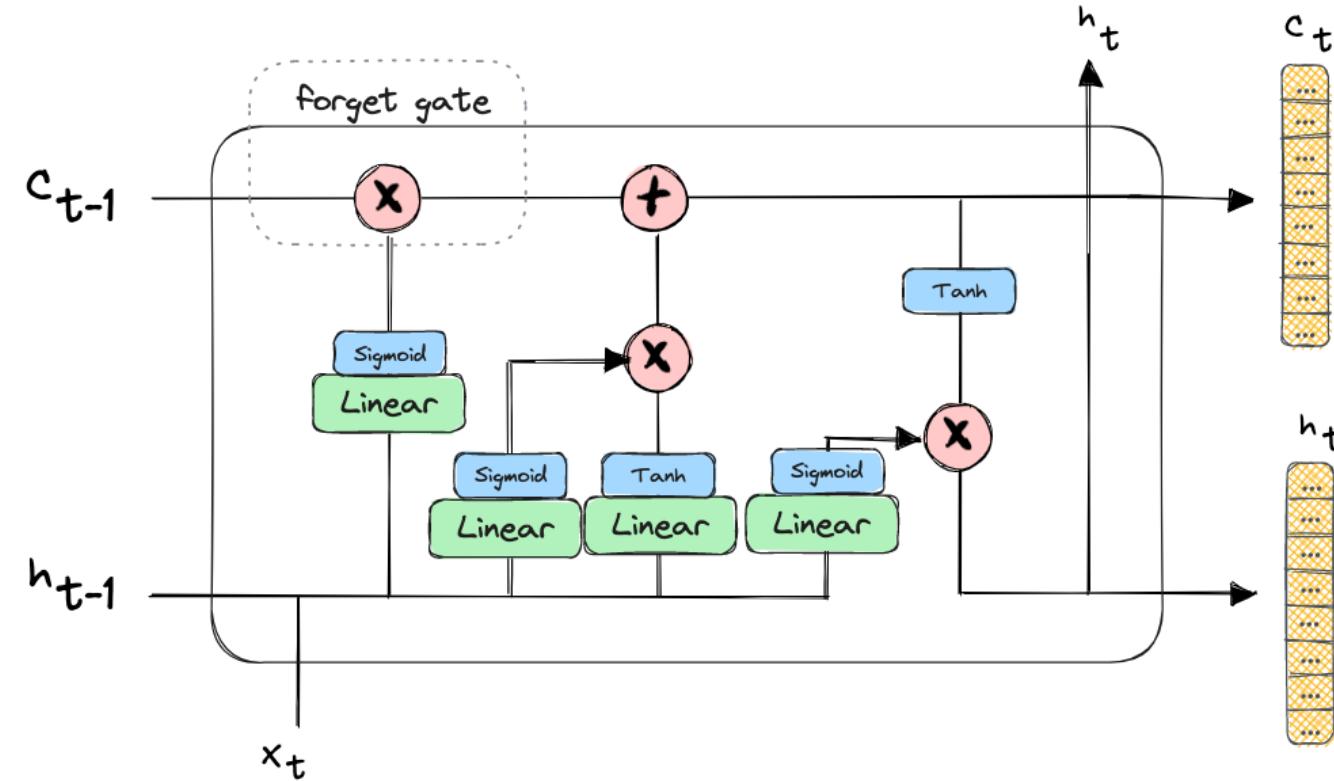


# LSTM • Long short-term memory

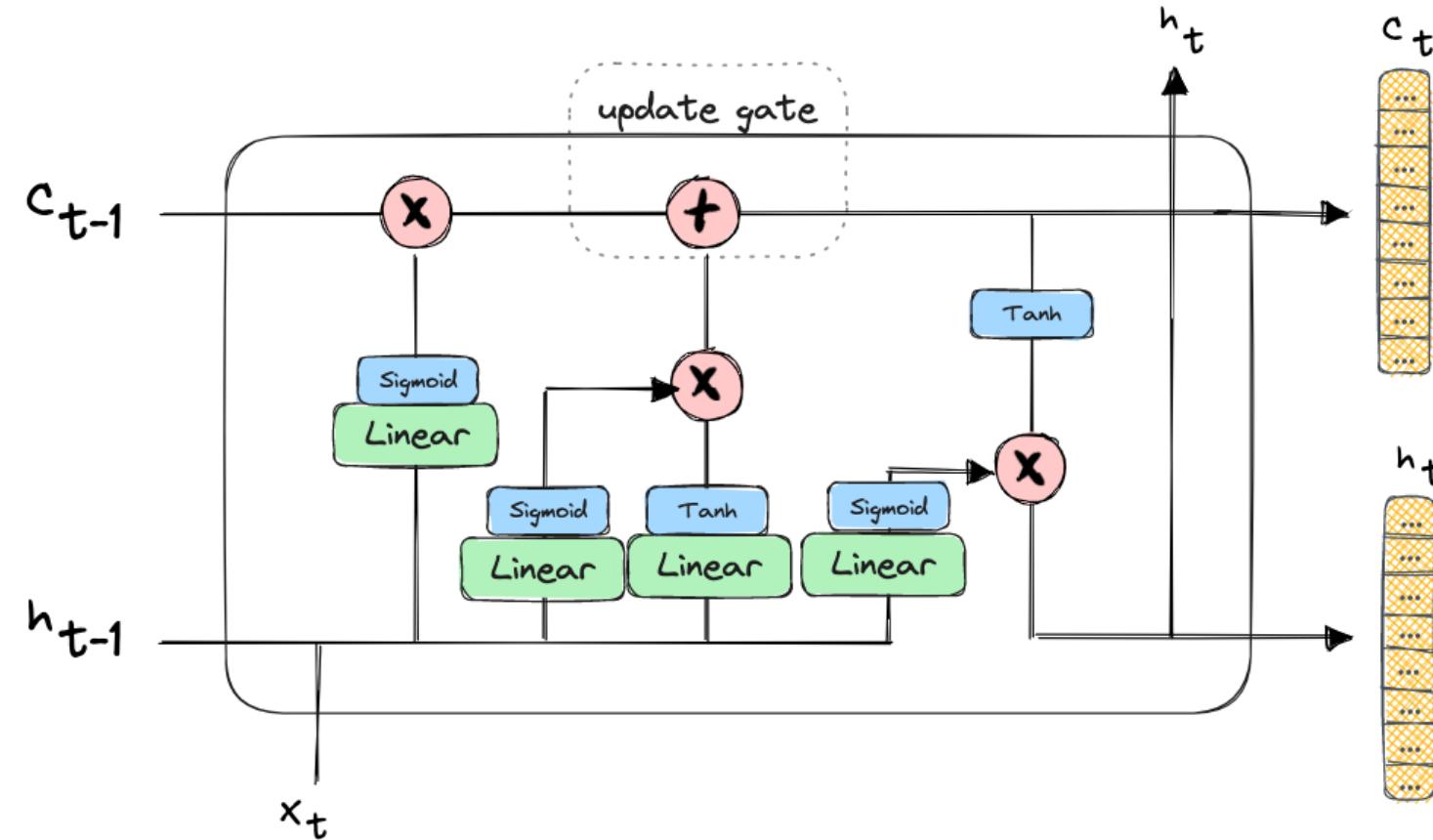


$C$  – cell state,  $h$  – hidden state,  $x$  – input

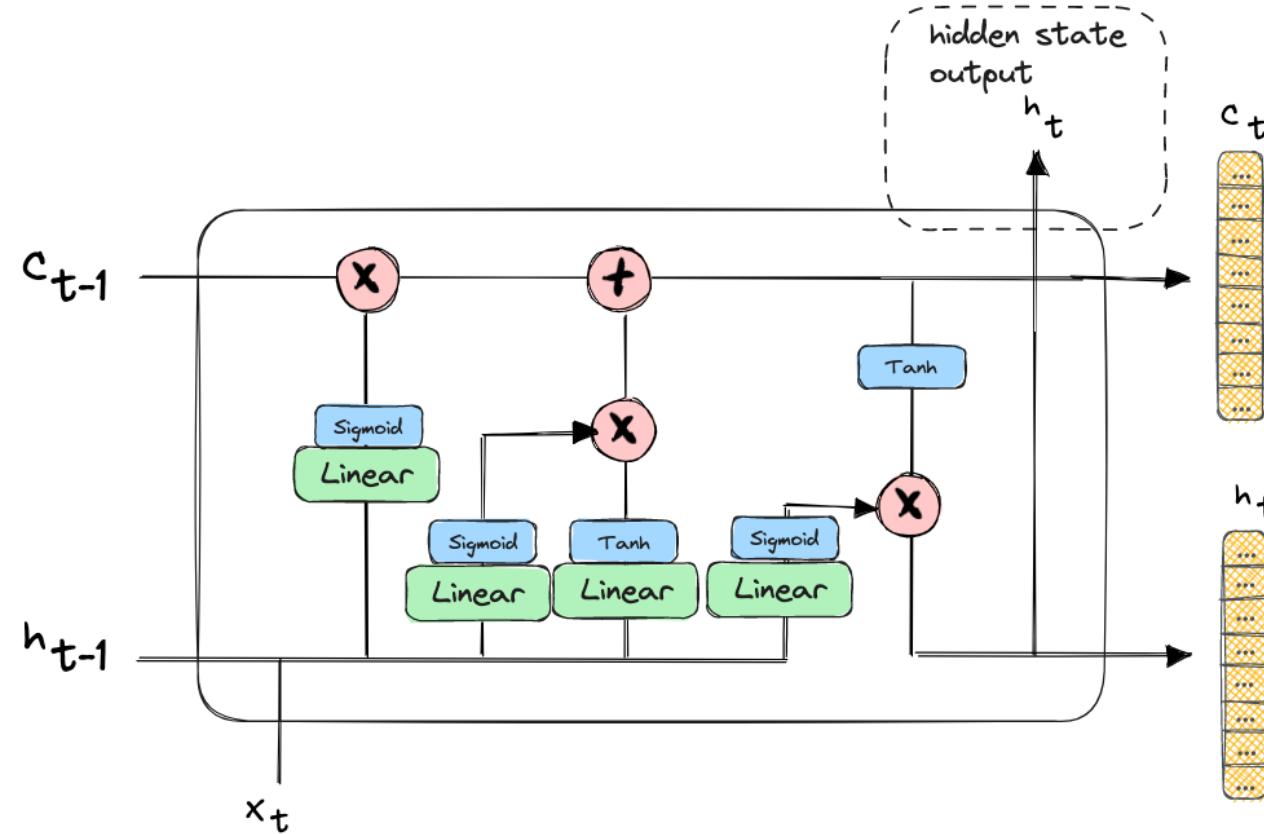
# LSTM • Long short-term memory



# LSTM



# LSTM



# Если попроще, то

## LSTM

---

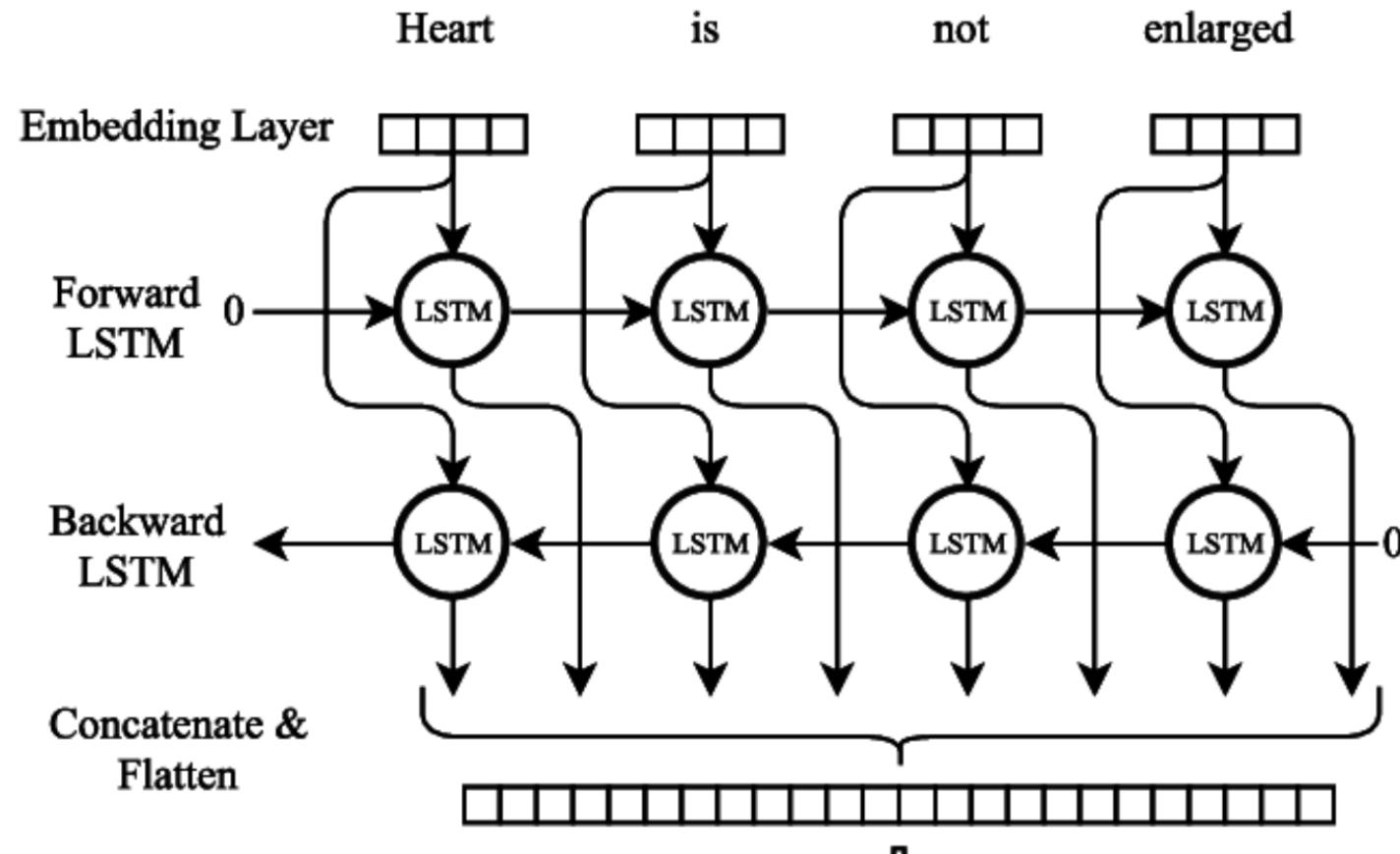
CLASS `torch.nn.LSTM(*args, **kwargs)` [\[SOURCE\]](#)

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

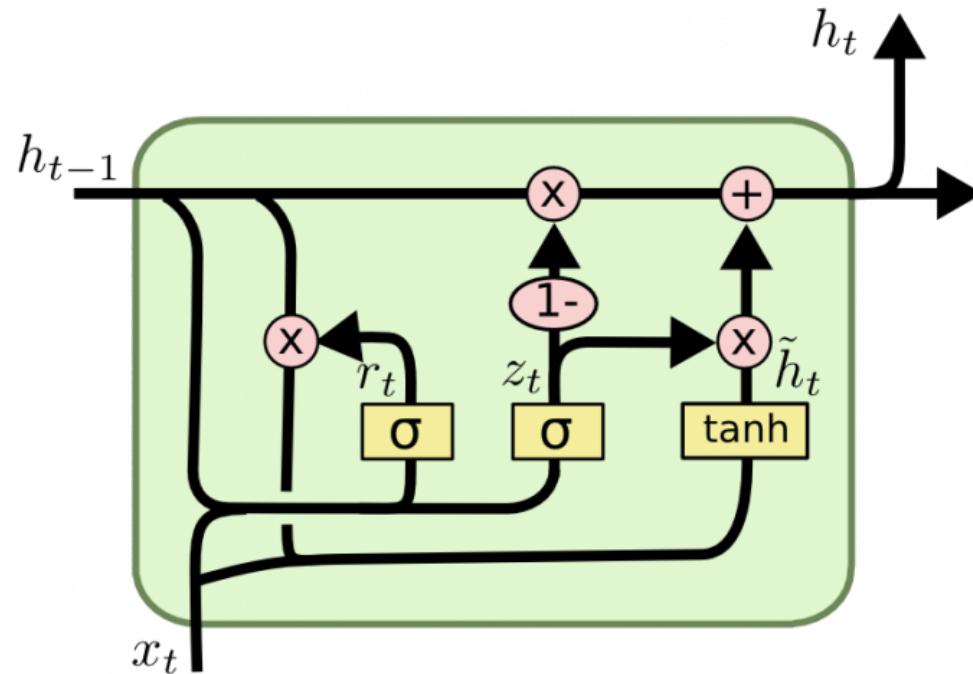
For each element in the input sequence, each layer computes the following function:

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

# Bidirectional LSTM • BiSLTM



# GRU • Gated Recurrent Unit



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

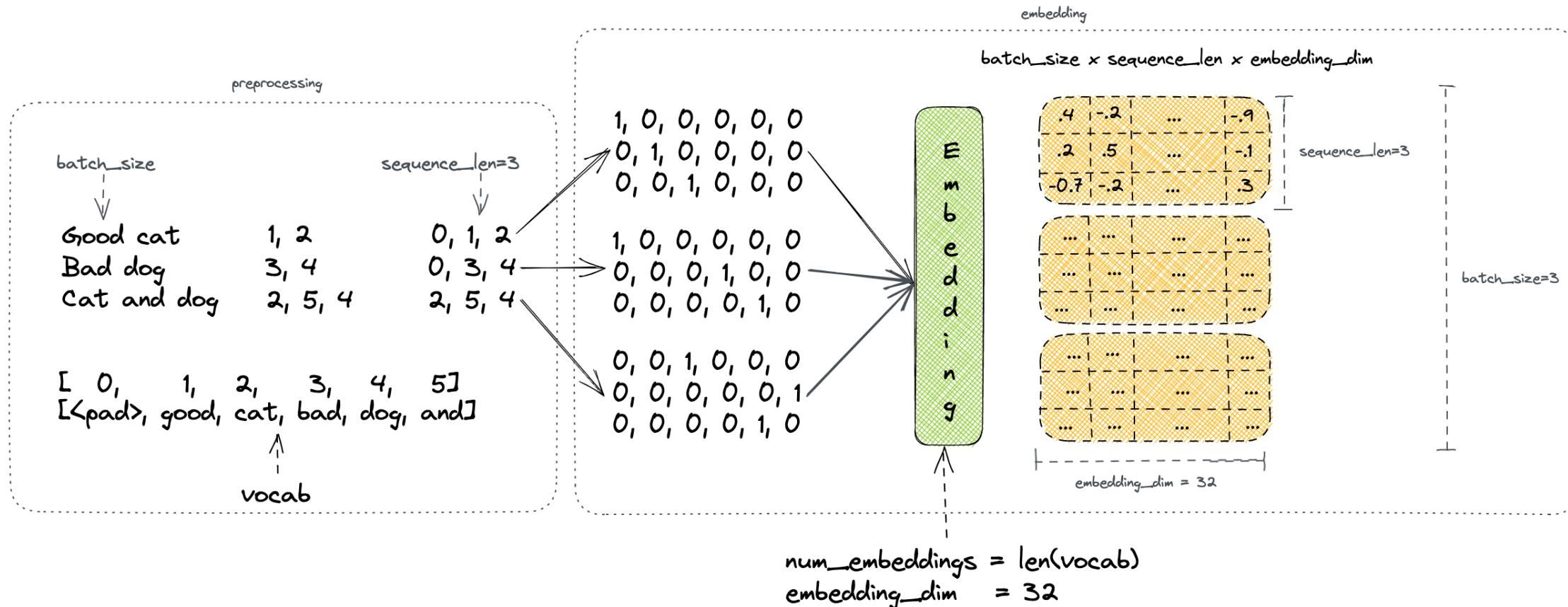
$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

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

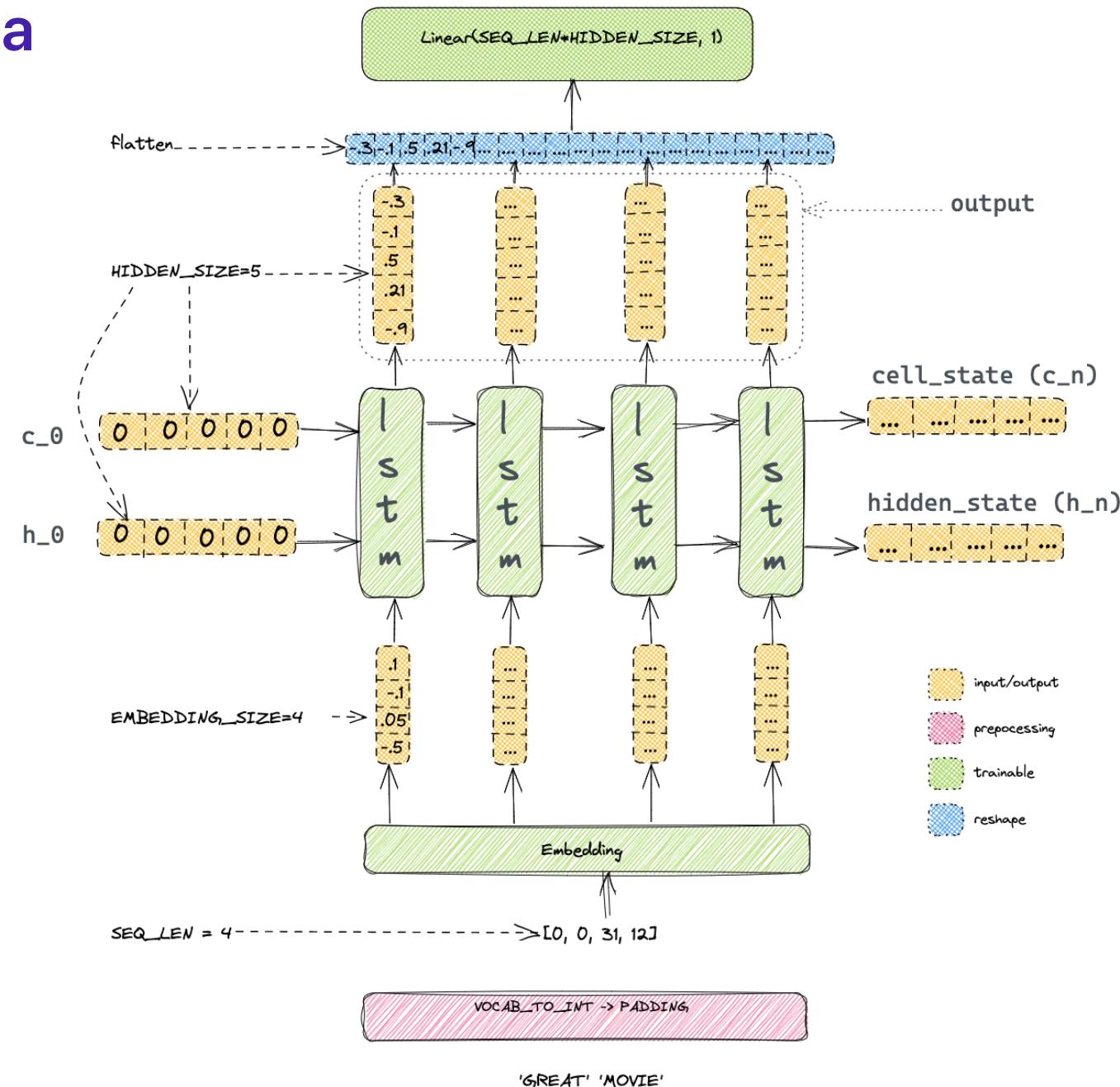
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

# Embedding-слой

# Embedding-слой



# Общая схема



# Итоги

- основная задача рекуррентных сетей - сохранять память
- рекуррентные сети могут работать с последовательностями
  - длинные последовательности приводят к проблемам с градиентами
  - длинные последовательности ведут к «забыванию» контекста
- в паре с рекуррентными сетями часто используют Embedding-слой, его задача – превратить каждый элемент последовательности в вектор заданной длины

Самые популярные варианты:

- RNN
- LSTM
- GRU

# Итоги

Каждая из архитектур может быть:

- двунаправленной
- многослойной
- двунаправленной и многослойной

Все это – параметры конкретного слоя в pytorch

```
nn.LSTM(  
    input_size      = ... # размер вектора, выходящего из эмбеддинга  
    hidden_size     = ... # внутренняя размерность Linear слоя  
    num_layers      = ... # число слоев  
    bidirectional  = ... # двунаправленная или односторонняя  
)
```