Рекуррентные сети

Лекция 5

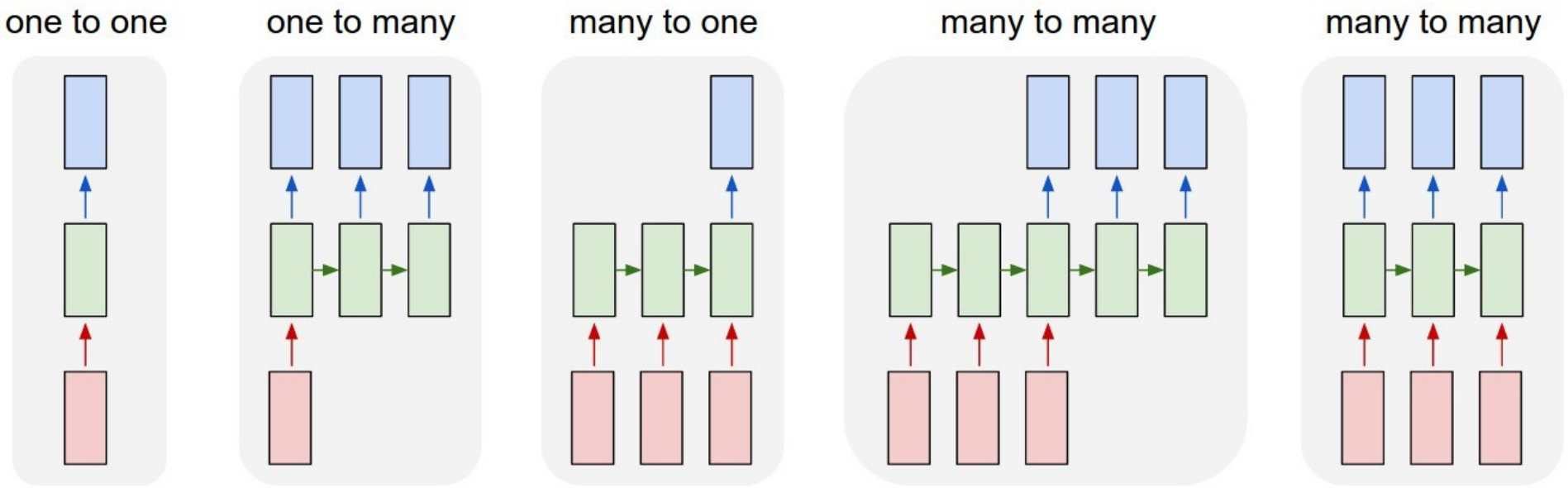
План лекции

- Обработка последовательностей
- Simple (vanilla) RNN
- LSTM
- GRU

Обработка последовательностей

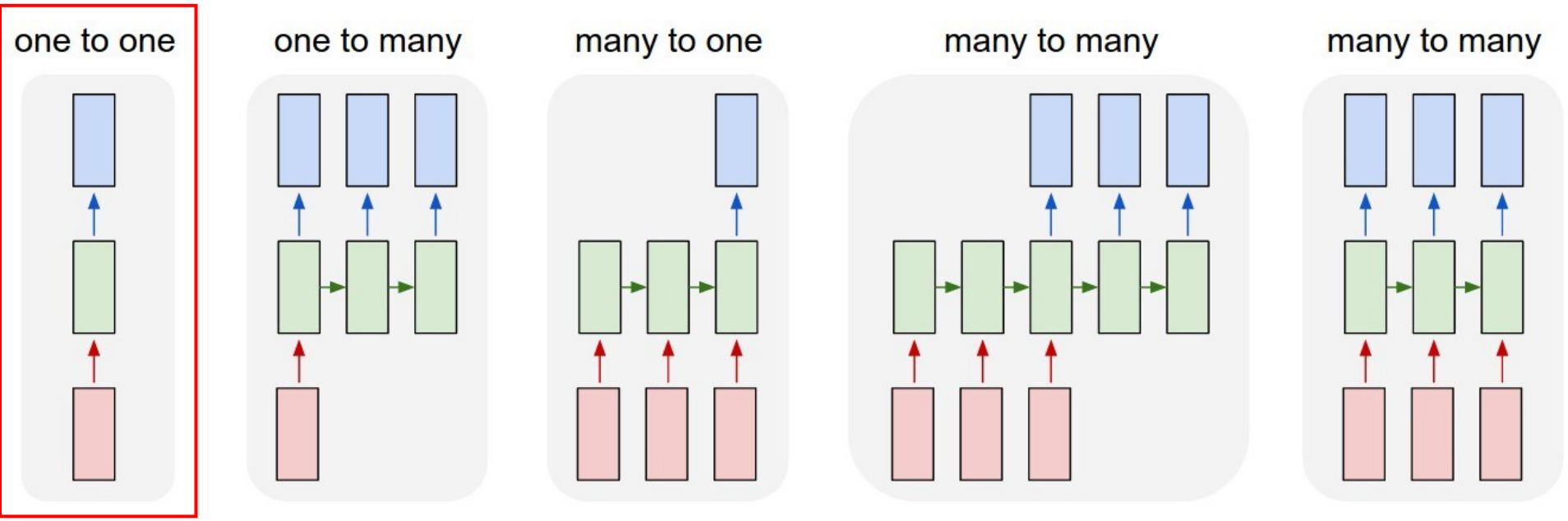
- Сети прямого распространения (Feed-Forward Neural Network, FFNN), свёрточные сети (Convolutional Neural Network, CNN):
  - Вход: вектор фиксированной размерности (изображение, текст)
  - Выход: вектор фиксированной размерности (классы)
  - Каждый вход независим от предыдущего
- Последовательности:
  - временны*е* ряды (акции, датчики)
  - текст
  - речь
  - музыка
  - видео

Задачи обработки последовательностей



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

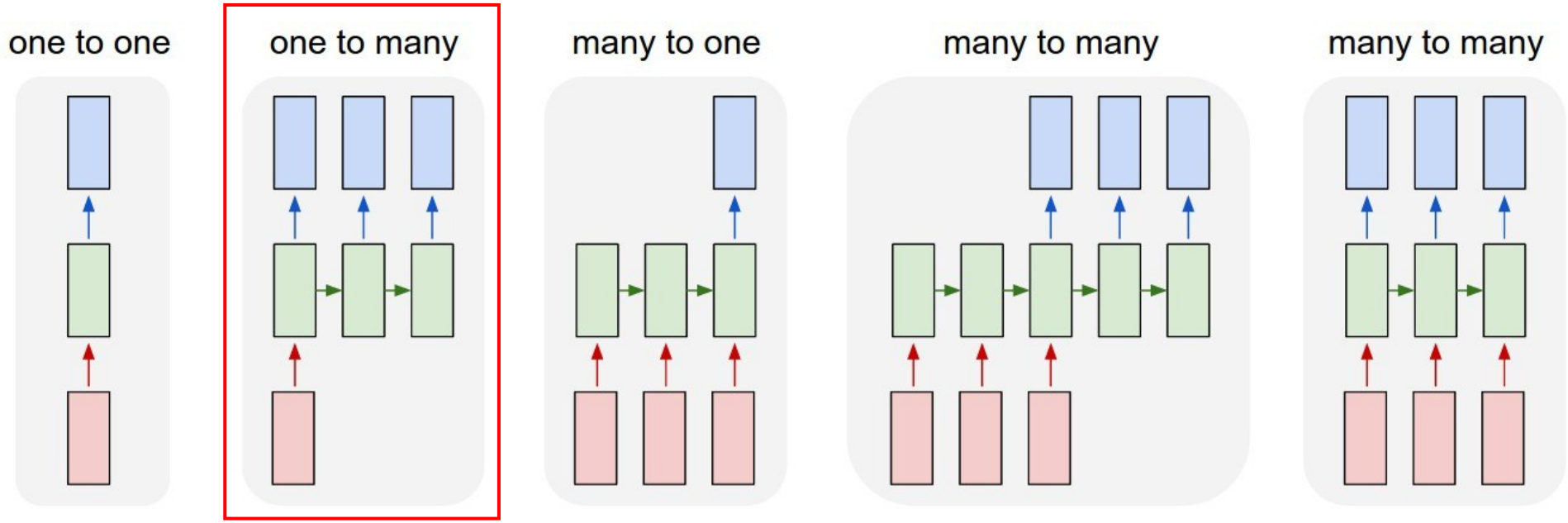
Задачи обработки последовательностей



- **One to one**:
  - вход – вектор фиксированной размерности
  - выход – вектор фиксированной размерности
  - пример: классификация изображений

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

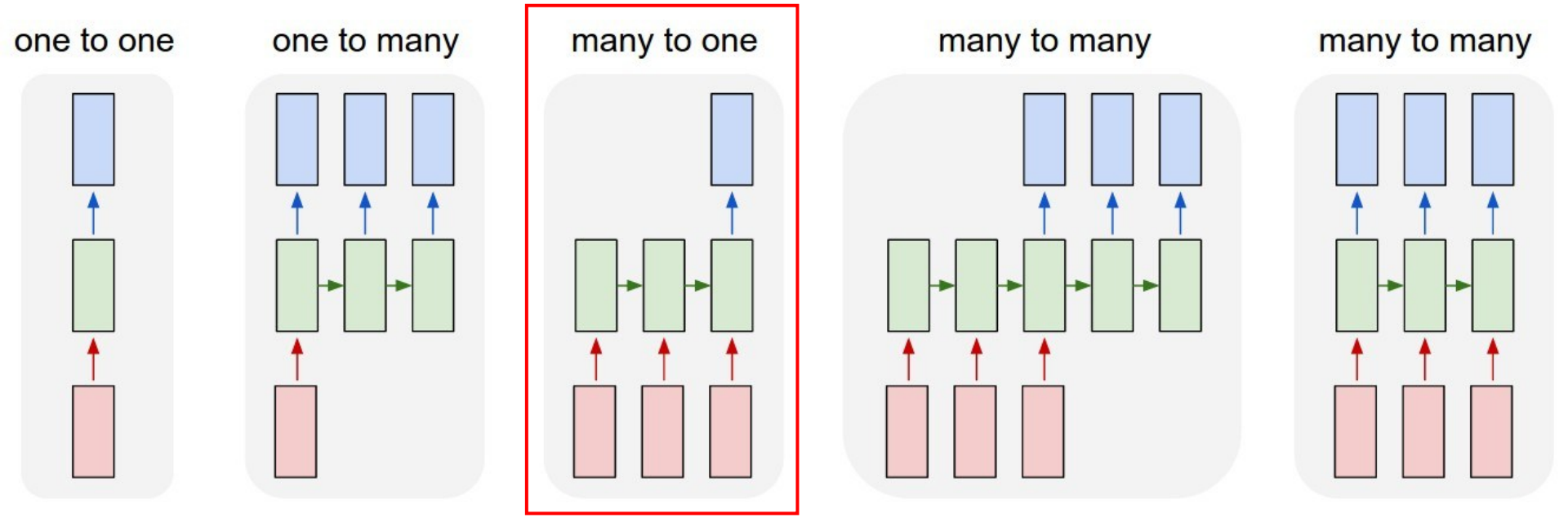
Задачи обработки последовательностей



- **One to many**:
  - вход – вектор фиксированной размерности
  - выход – последовательность
  - пример: генерация описания изображений (image captioning)

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

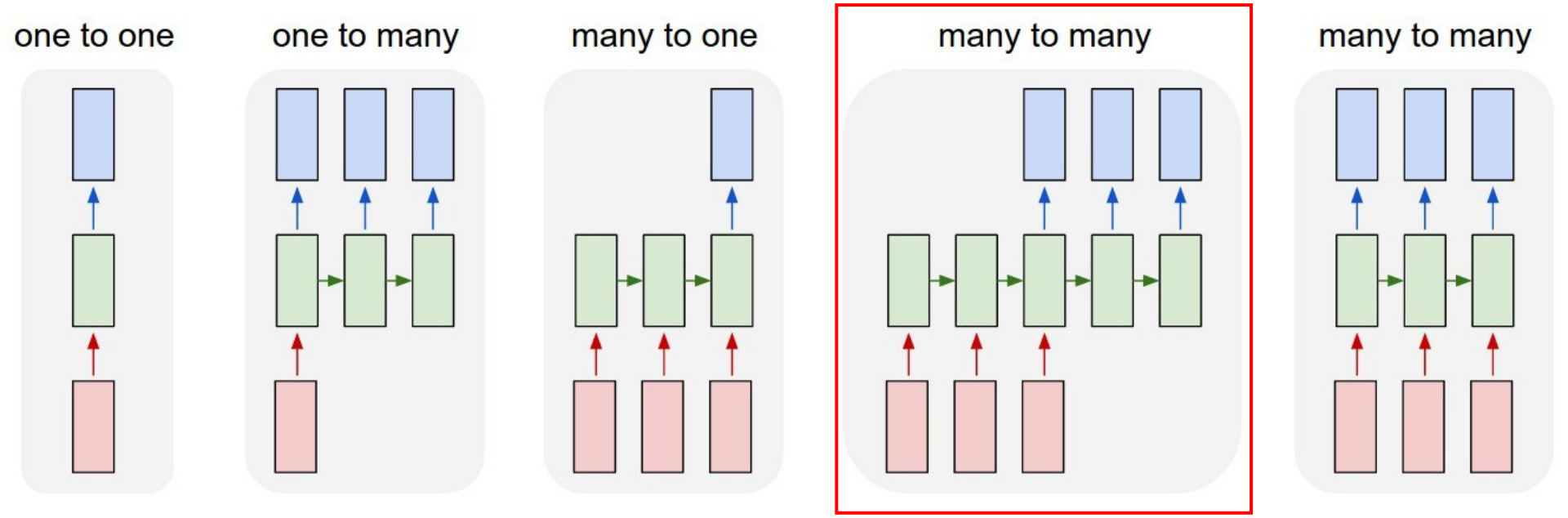
Задачи обработки последовательностей



- **Many to one**:
  - вход – последовательность
  - выход – вектор фиксированной размерности
  - пример: анализ тональности (sentiment analysis)

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

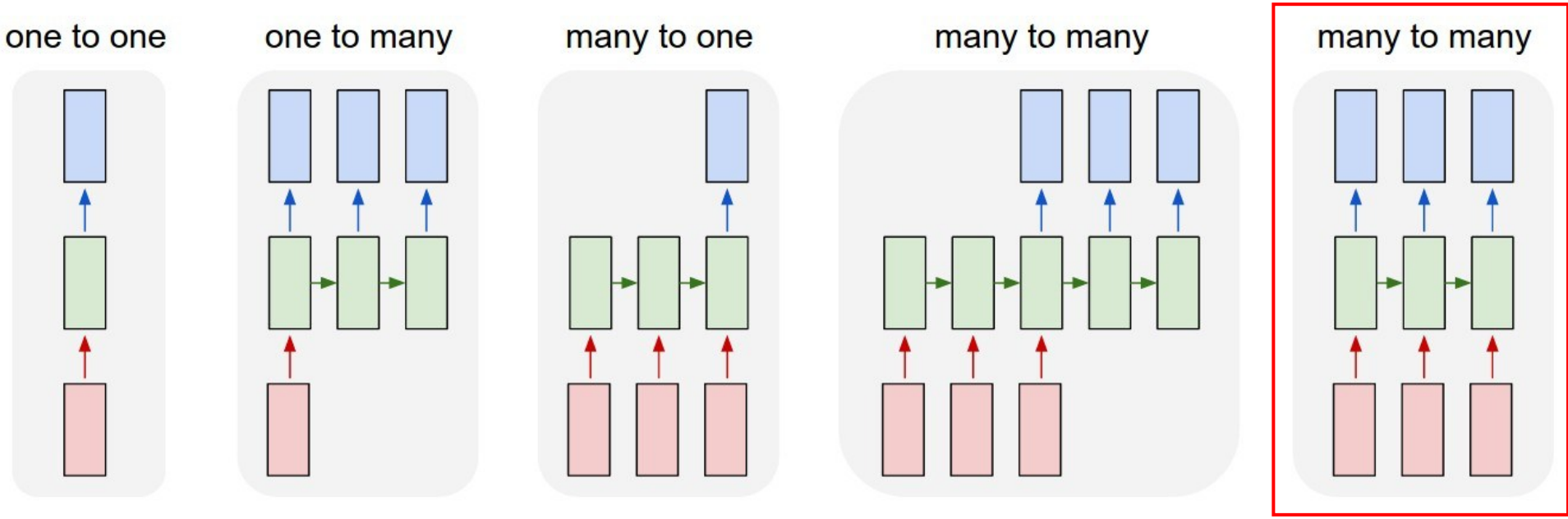
Задачи обработки последовательностей



- **Many to many**:
  - вход – последовательность
  - выход – последовательность
  - пример: машинный перевод (machine translation)
  - пример: диалоговые системы (conversational agents)

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Задачи обработки последовательностей



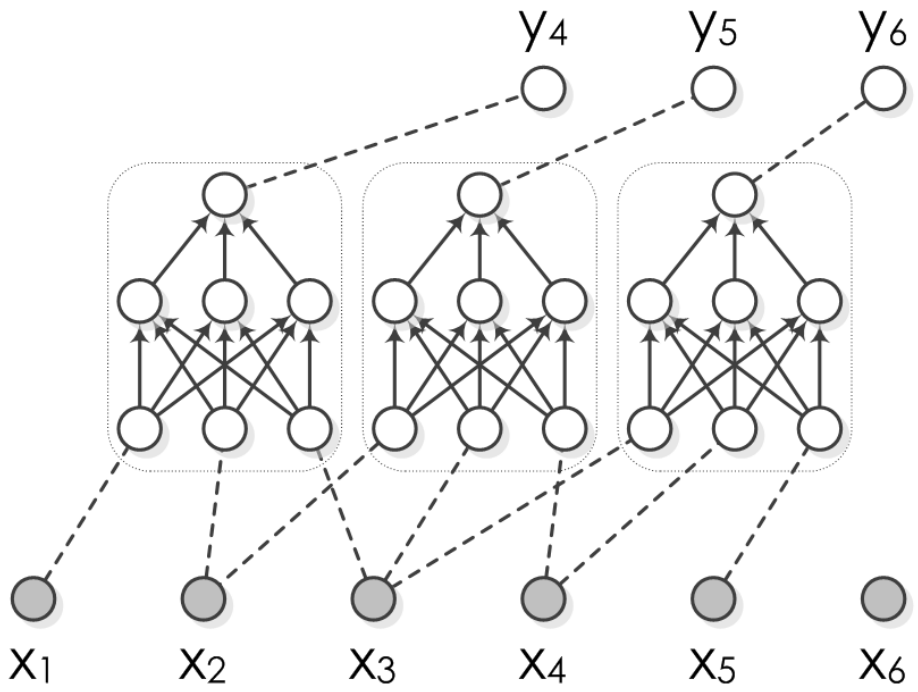
- **Synchronized many to many**:
  - вход – последовательность
  - выход – последовательность
  - пример: разметка кадров видеопотока (video frame classification)
  - пример: извлечение именованных сущностей (named entity recognition)

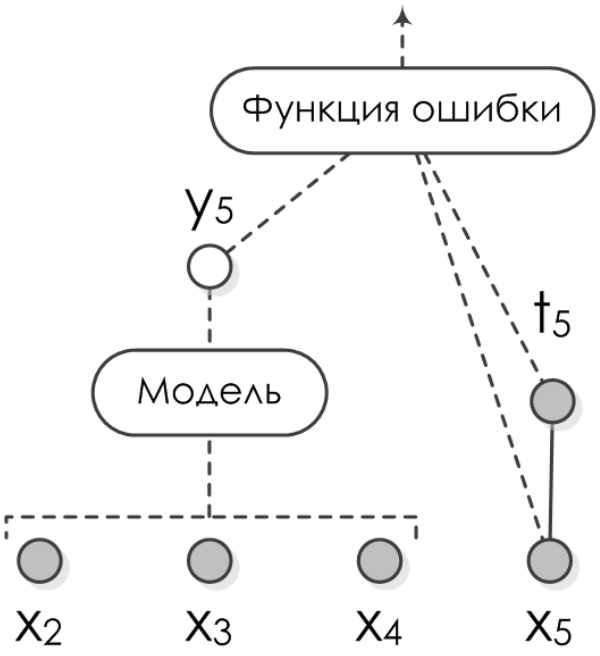
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Скользящее окно

$$y_n = f(x_{n-1}, \dots, x_{n-l})$$

$$x_n = f(x_{n-1}, \dots, x_{n-l})$$

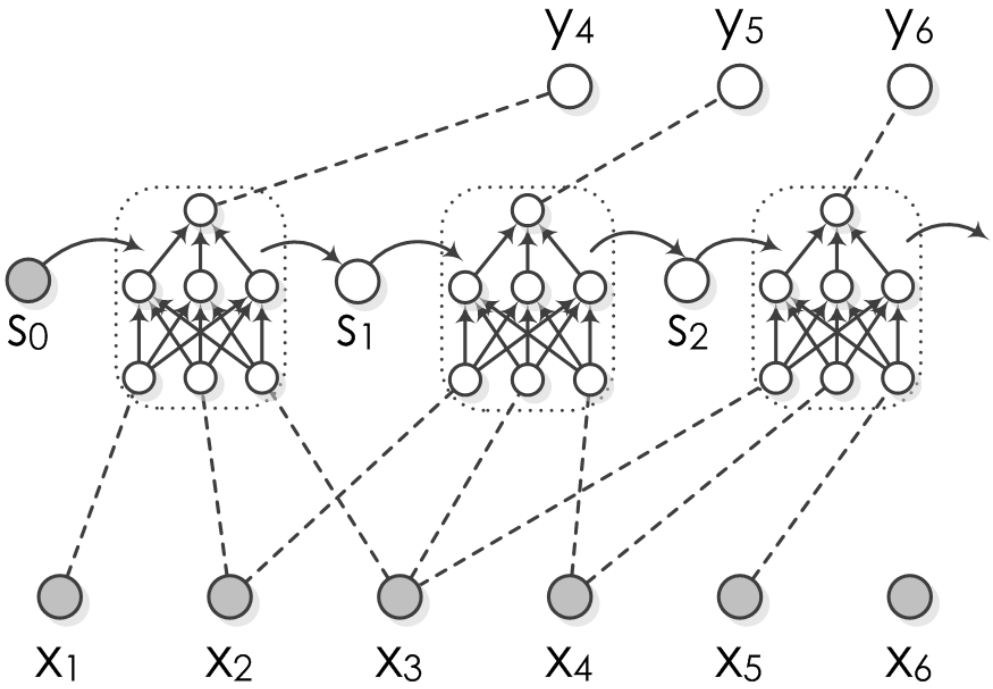


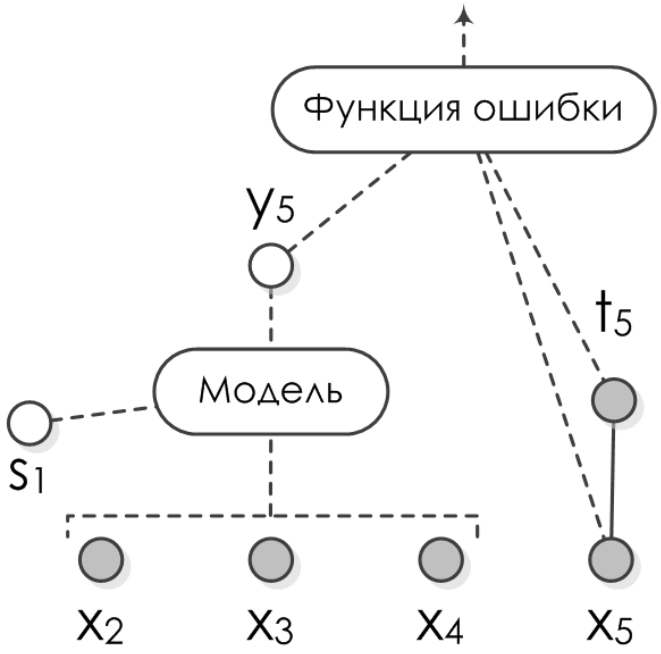


Но входные окна независимы друг от друга (x_1 никак не влияет на x_4)

Рекуррентные сети: идея (Recurrent Neural Network, RNN)

$$y_n = f(x_{n-1}, \dots, x_{n-l}, s_{n-l-1}) \qquad s_i = h(x_i, \dots, x_{i+l}, s_{i-1})$$

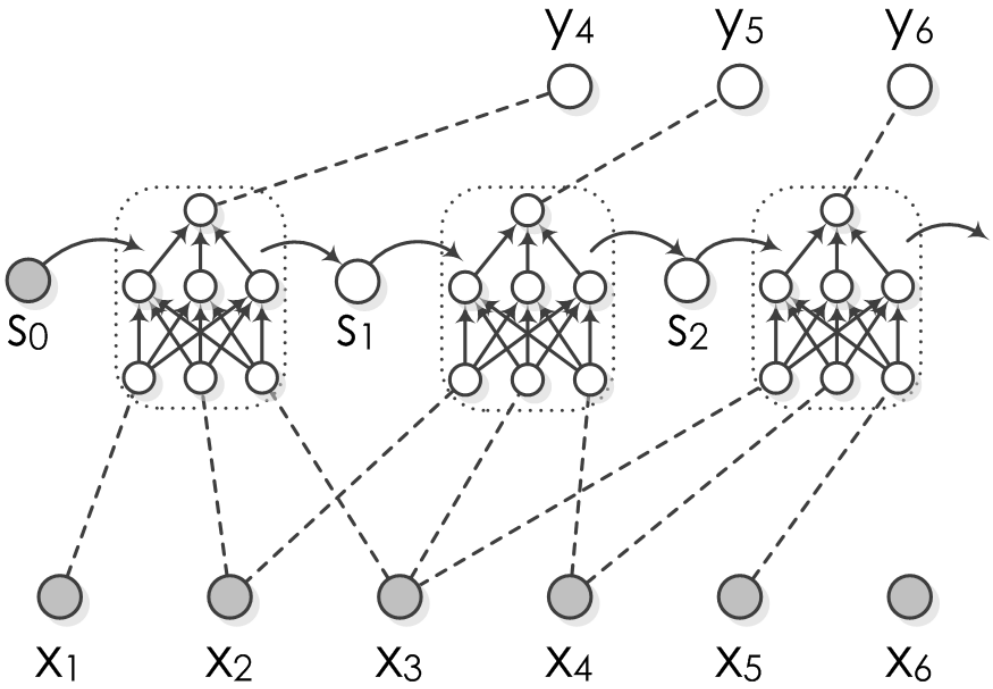


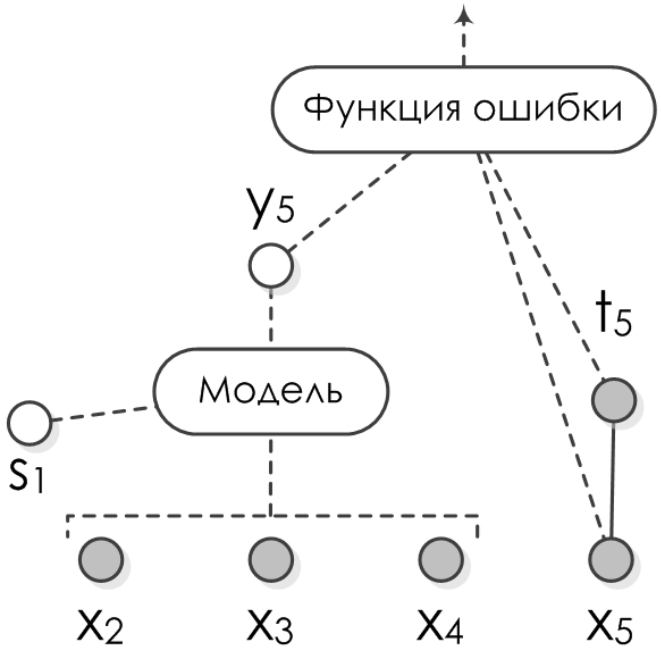


Как обучать сети с циклами?

Рекуррентные сети: алгоритм обратного распространения ошибки

$$y_6 = f(x_3, x_4, x_5, s_2) = f\big(x_3, x_4, x_5, h(x_2, x_3, x_4, s_1)\big) =$$
$$= f\left(x_3, x_4, x_5, h\big(x_2, x_3, x_4, h(x_1, x_2, x_3, s_0)\big)\right)$$



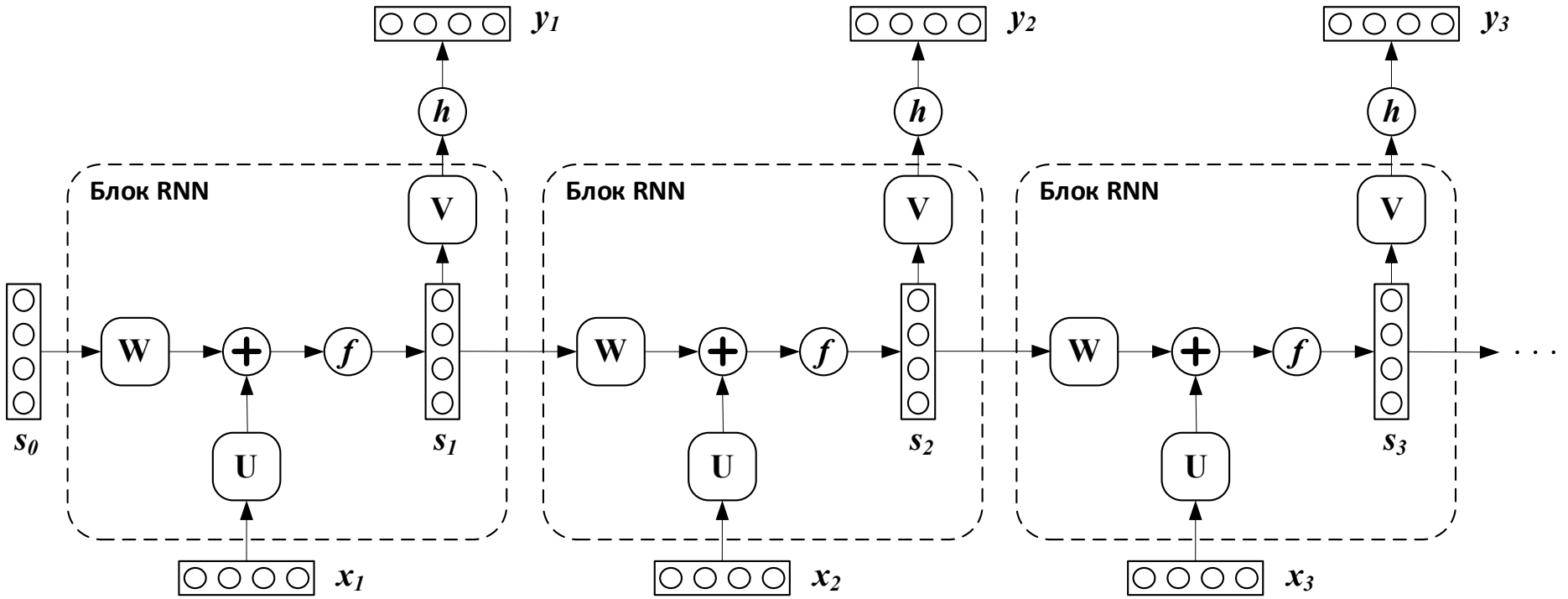


Очень глубокая сеть со множеством общих (shared) весов

Рекуррентные сети: Simple (Vanilla) RNN

$$\vec{s}_t = f\left(\boldsymbol{W}\vec{s}_{t-1} + \boldsymbol{U}\vec{x}_t + \vec{b}\right) \qquad \vec{y}_t = h(\boldsymbol{V}\vec{s}_t + \vec{c})$$

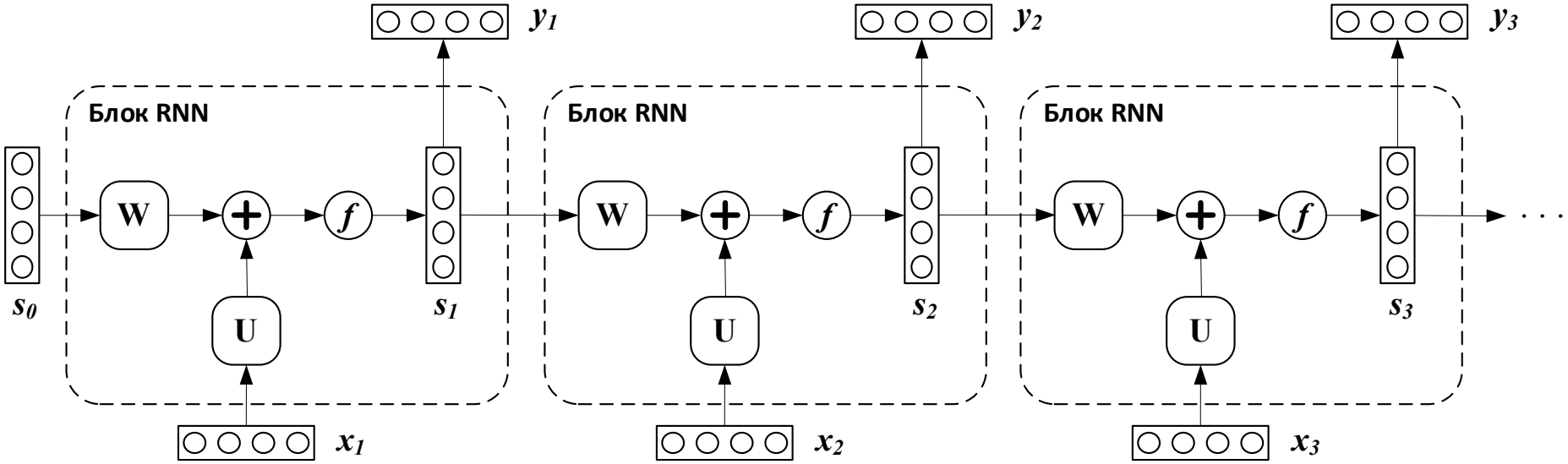
f: σ, tanh, ReLU $\qquad$ h: softmax



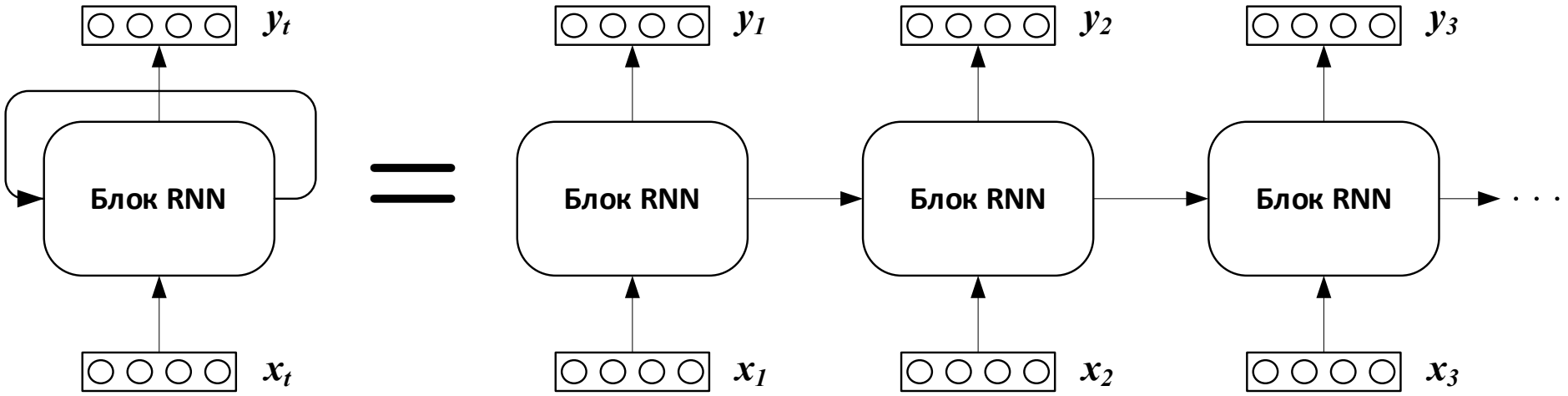
Сеть Элмана: Elman Jeffrey L. Finding Structure in Time // Cognitive Science, 1990

Рекуррентные сети: Simple RNN

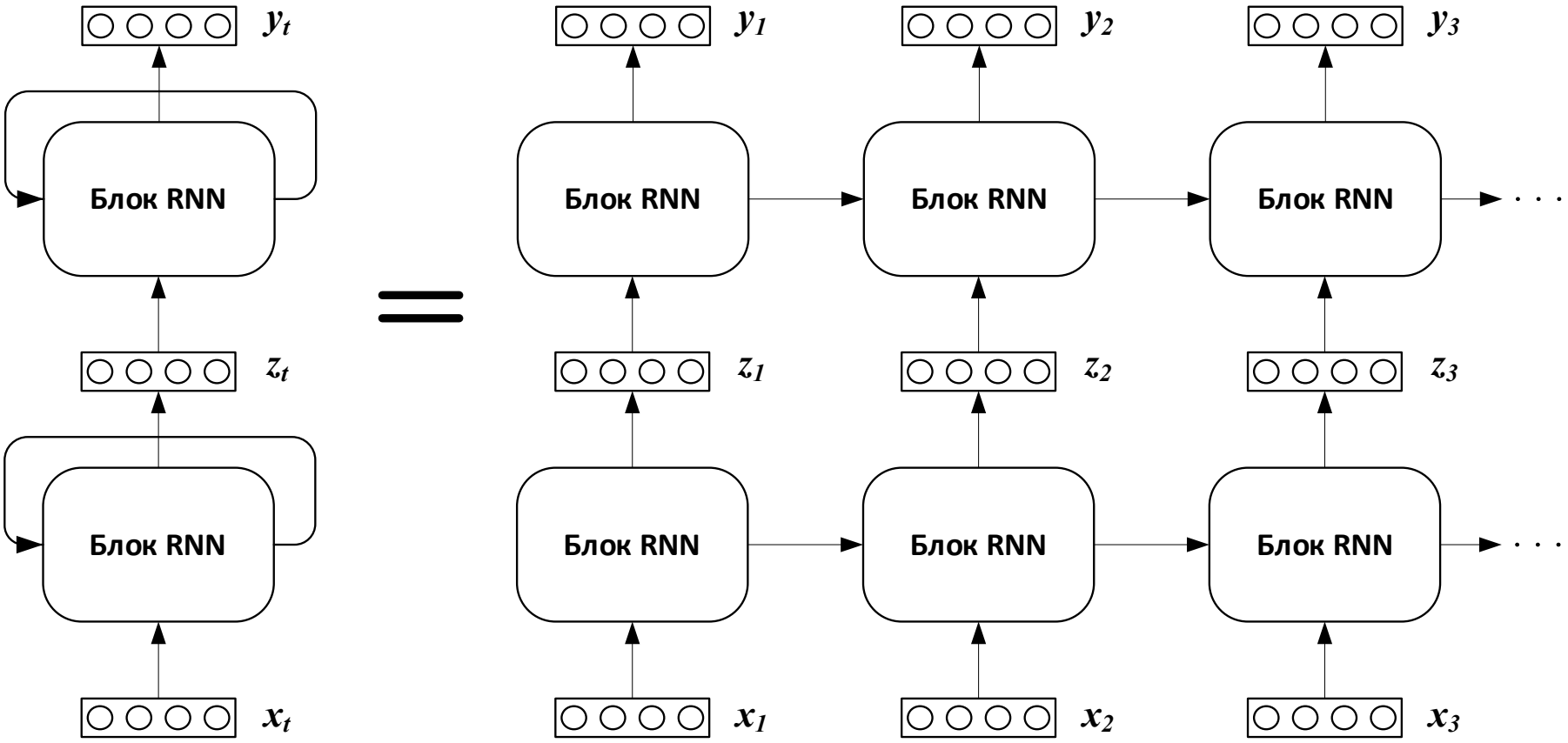
$$\vec{s}_t = f\left(\boldsymbol{W}\vec{s}_{t-1} + \boldsymbol{U}\vec{x}_t + \vec{b}\right) \qquad \vec{y}_t = \vec{s}_t \qquad f\text{: tanh, ReLU}$$



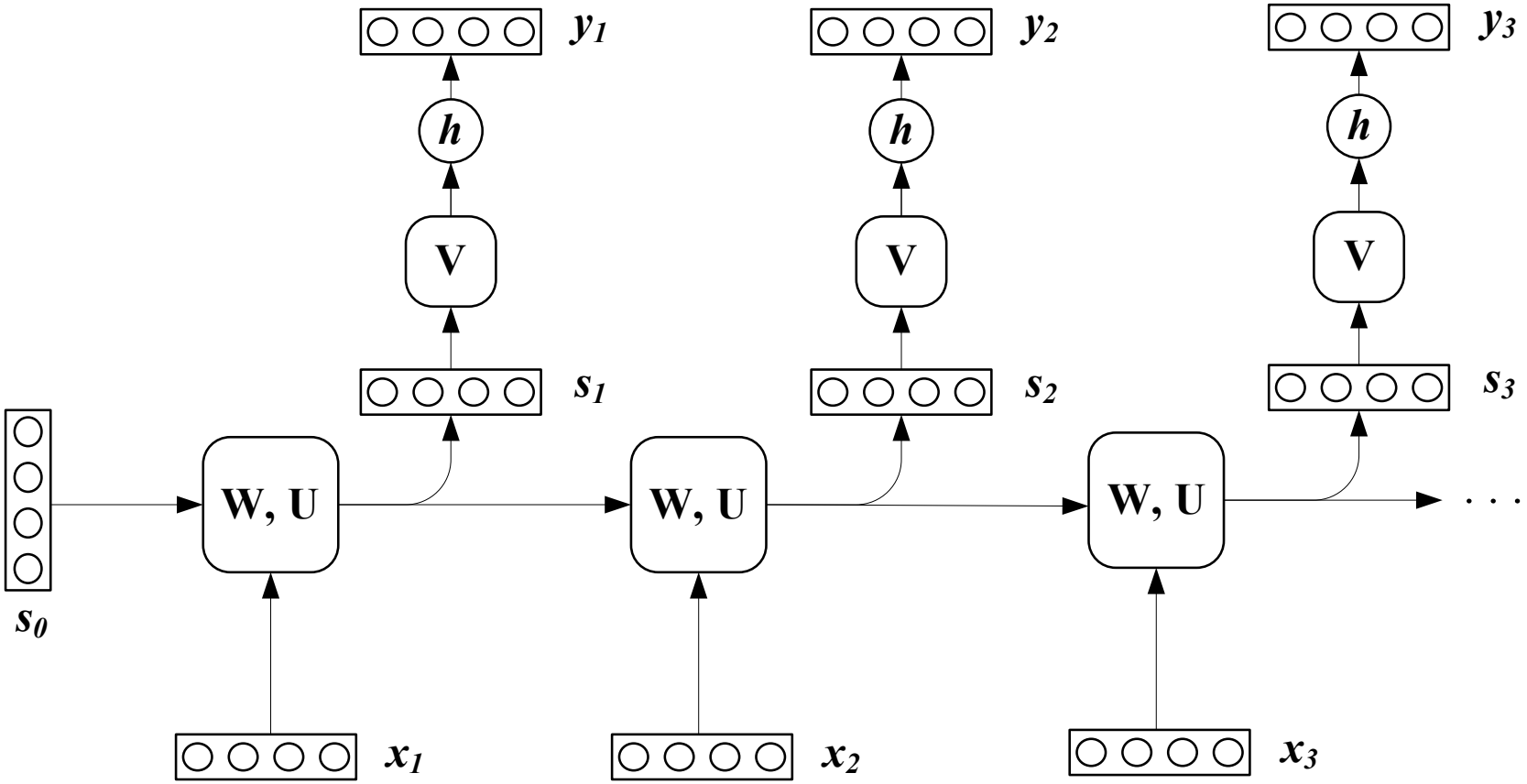
Рекуррентные сети: Simple RNN



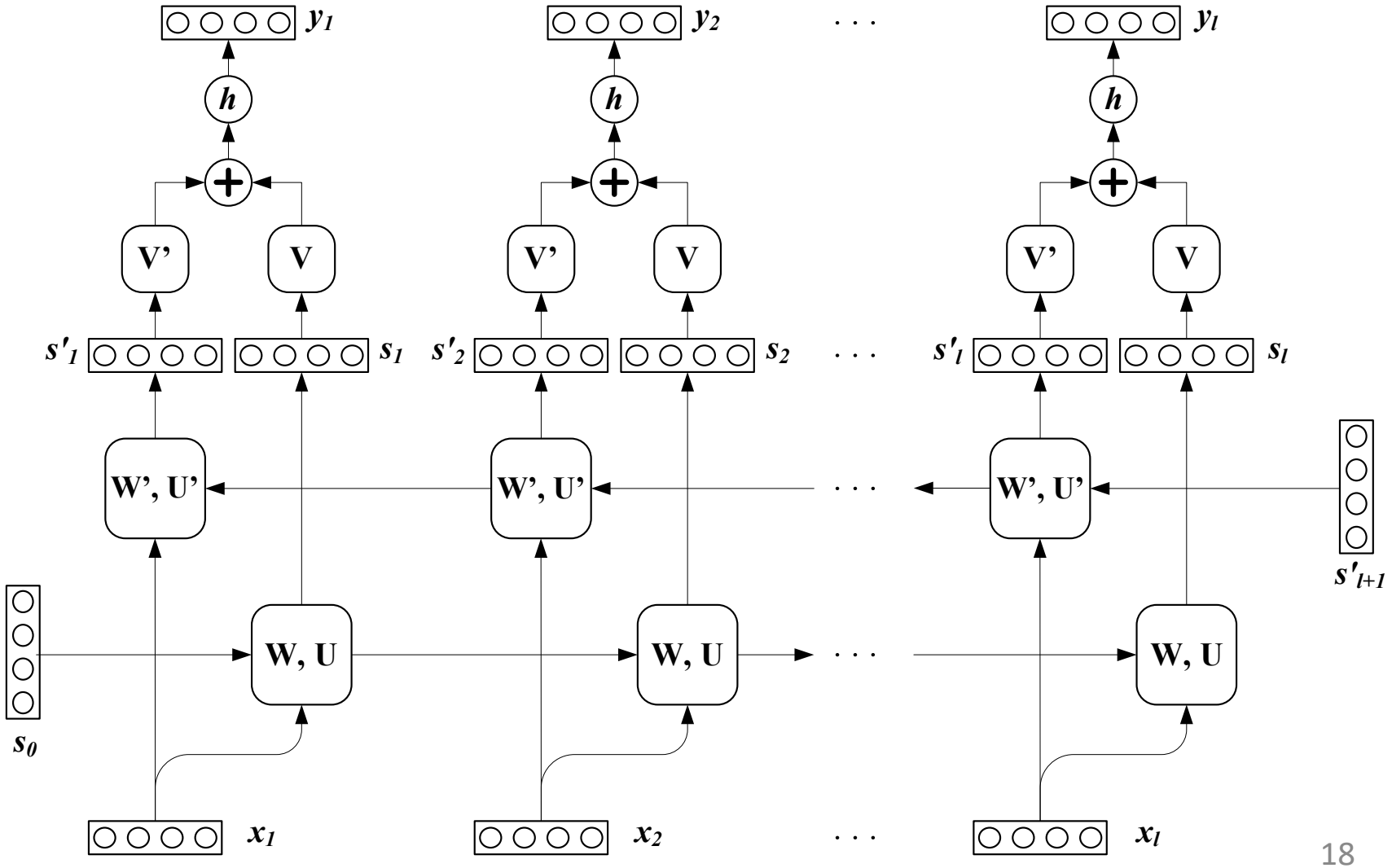
Рекуррентные сети: двухслойная RNN



Двунаправленные рекуррентные сети (Bidirectional RNN)



Двунаправленные рекуррентные сети (Bidirectional RNN)



Двунаправленные рекуррентные сети (Bidirectional RNN)

$$\vec{s}_t = f\left(\boldsymbol{W}\vec{s}_{t-1} + \boldsymbol{U}\vec{x}_t + \vec{b}\right)$$

$$\overrightarrow{s'}_t = f\left(\boldsymbol{W}'\overrightarrow{s'}_{t+1} + \boldsymbol{U}'\vec{x}_t + \overrightarrow{b'}\right)$$

$$\vec{y}_t = h\left(\boldsymbol{V}\vec{s}_t + \boldsymbol{V}'\overrightarrow{s'}_t + \vec{c}\right)$$

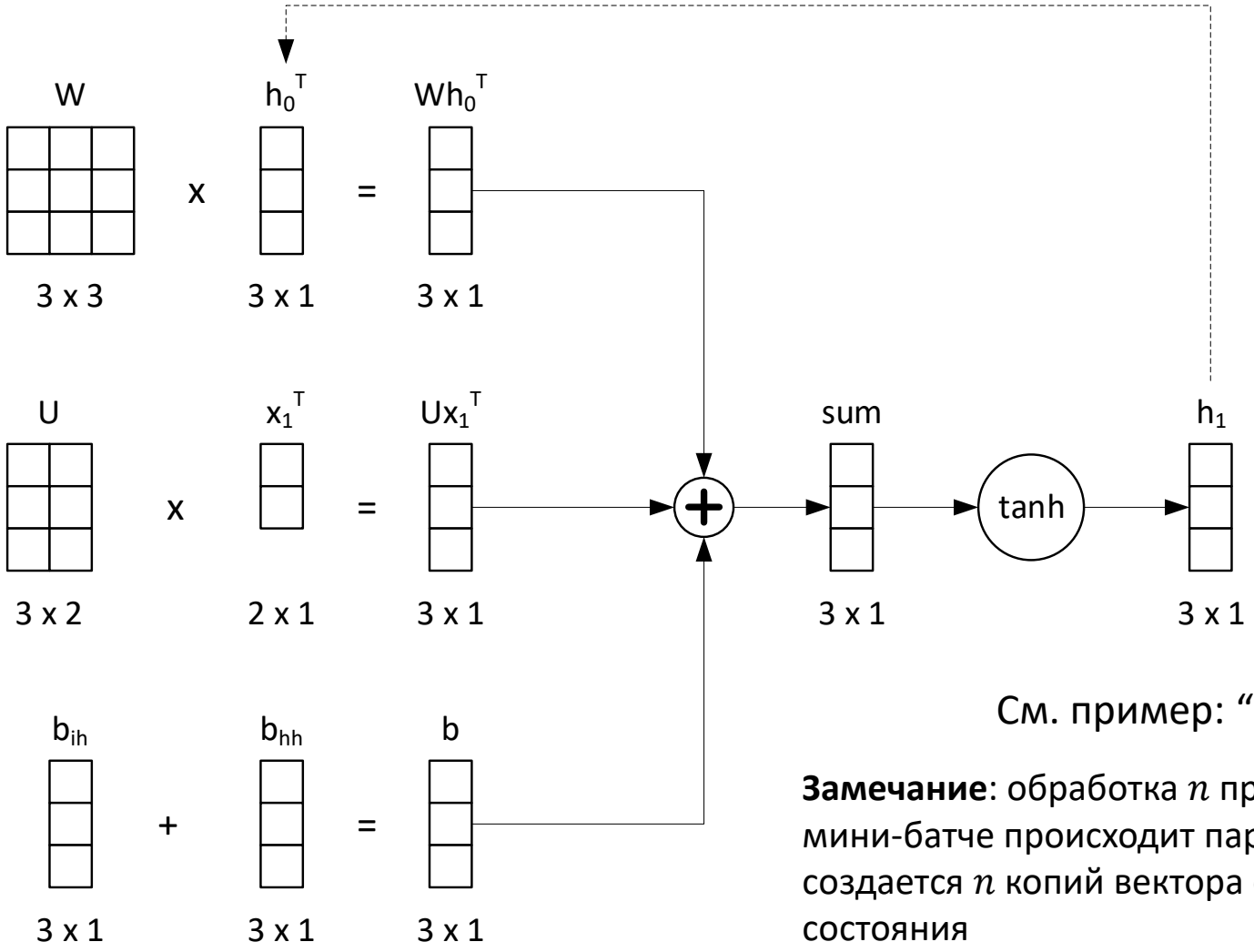
Рекуррентные сети: Simple RNN – PyTorch

```
torch.nn.RNN(
    input_size,           # размерность входа
    hidden_size,          # размерность скрытого состояния
    num_layers=1,         # количество слоёв
    nonlinearity='tanh',  # нелинейность: 'tanh', 'relu'
    bias=True,            # свободные коэффициенты
    batch_first=False,    # расположение батча во входе
    dropout=0,            # вероятность дропаута
                          # (кроме последнего слоя)
    bidirectional=False   # двунаправленность
)
```

https://pytorch.org/docs/stable/generated/torch.nn.RNN.html

Рекуррентные сети: Simple RNN – PyTorch

$$\vec{s}_t = f\left(\boldsymbol{W}\vec{s}_{t-1} + \vec{b}_{hh} + \boldsymbol{U}\vec{x}_t + \vec{b}_{ih}\right) \qquad \vec{y}_t = \vec{s}_t \qquad f\text{: tanh, ReLU}$$



См. пример: “rnn.ipynb”

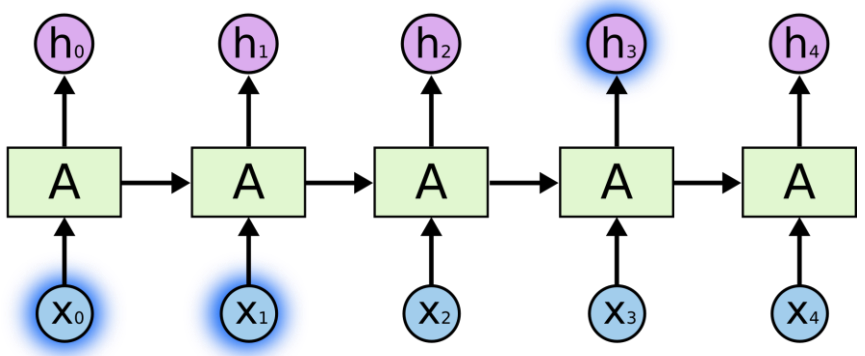
Замечание: обработка n примеров в мини-батче происходит параллельно – создается n копий вектора скрытого состояния

Проблемы RNN

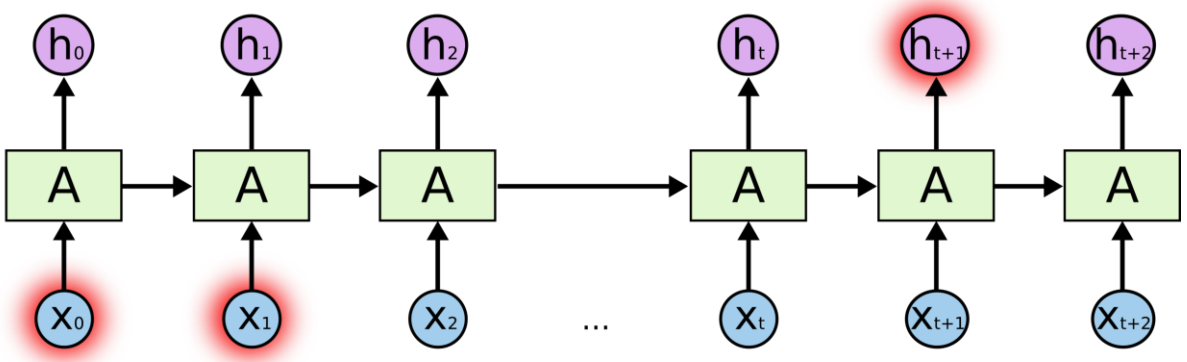
- Взрывающиеся градиенты (exploding gradients)
  - Если матрица весов увеличивает норму вектора градиента, то через T слоев норма увеличится экспоненциально от T
  - Решение: отсечение градиентов (gradient clipping)
- Затухающие градиенты (vanishing gradients)
  - Проблема долговременных зависимостей в данных (Long-Term Dependencies)

Проблемы RNN

- Кратковременные зависимости:
  - «Облака плывут по **небу**»



- Долговременные зависимости:
  - «Я рос во Франции… Я бегло говорю **по-французски**»

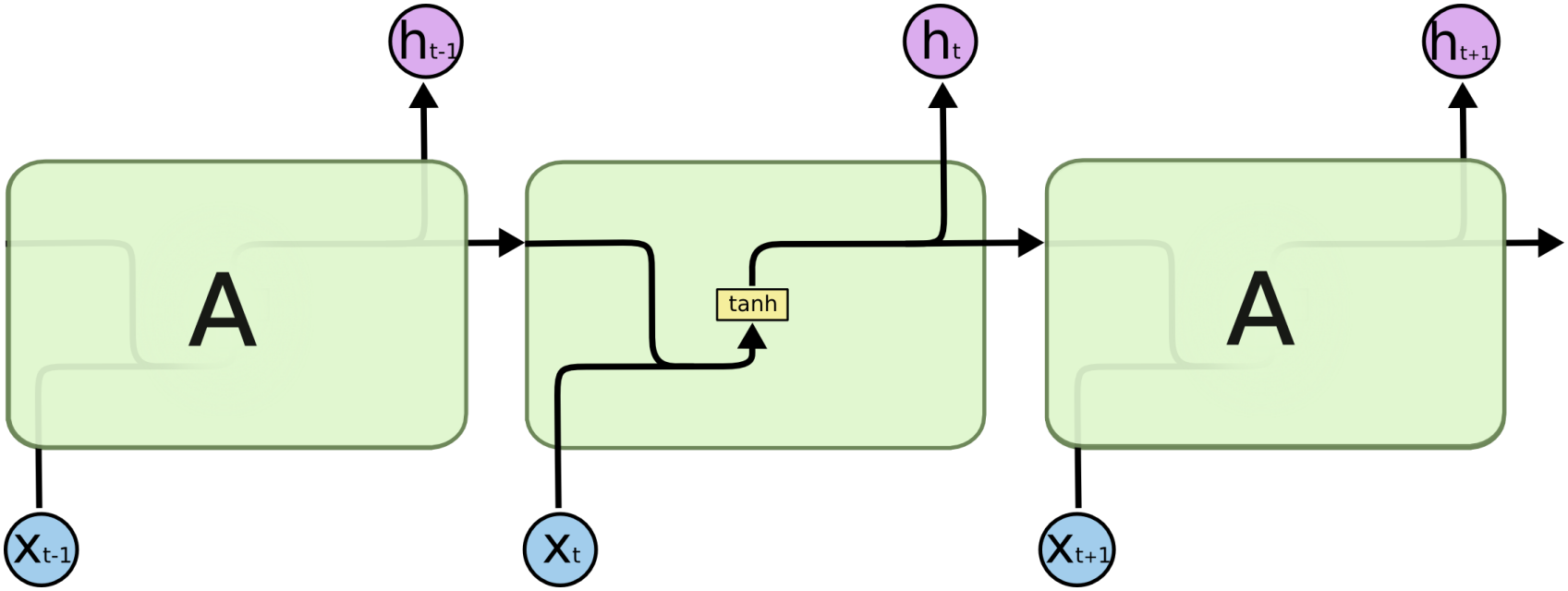


LSTM

- LSTM (Long Short-Term Memory) – долгая краткосрочная память
  - Hochreiter Sepp, Schmidhuber Jürgen. Long Short-Term Memory: Tech. Rep. FKI-207-95: Fakultät für Informatik, Technische Universität München, 1995
  - Hochreiter Sepp, Schmidhuber Jürgen. Long Short-Term Memory // Neural Computation. 1997. Vol. 9(8). P. 1735–1780

LSTM

- Simple RNN:

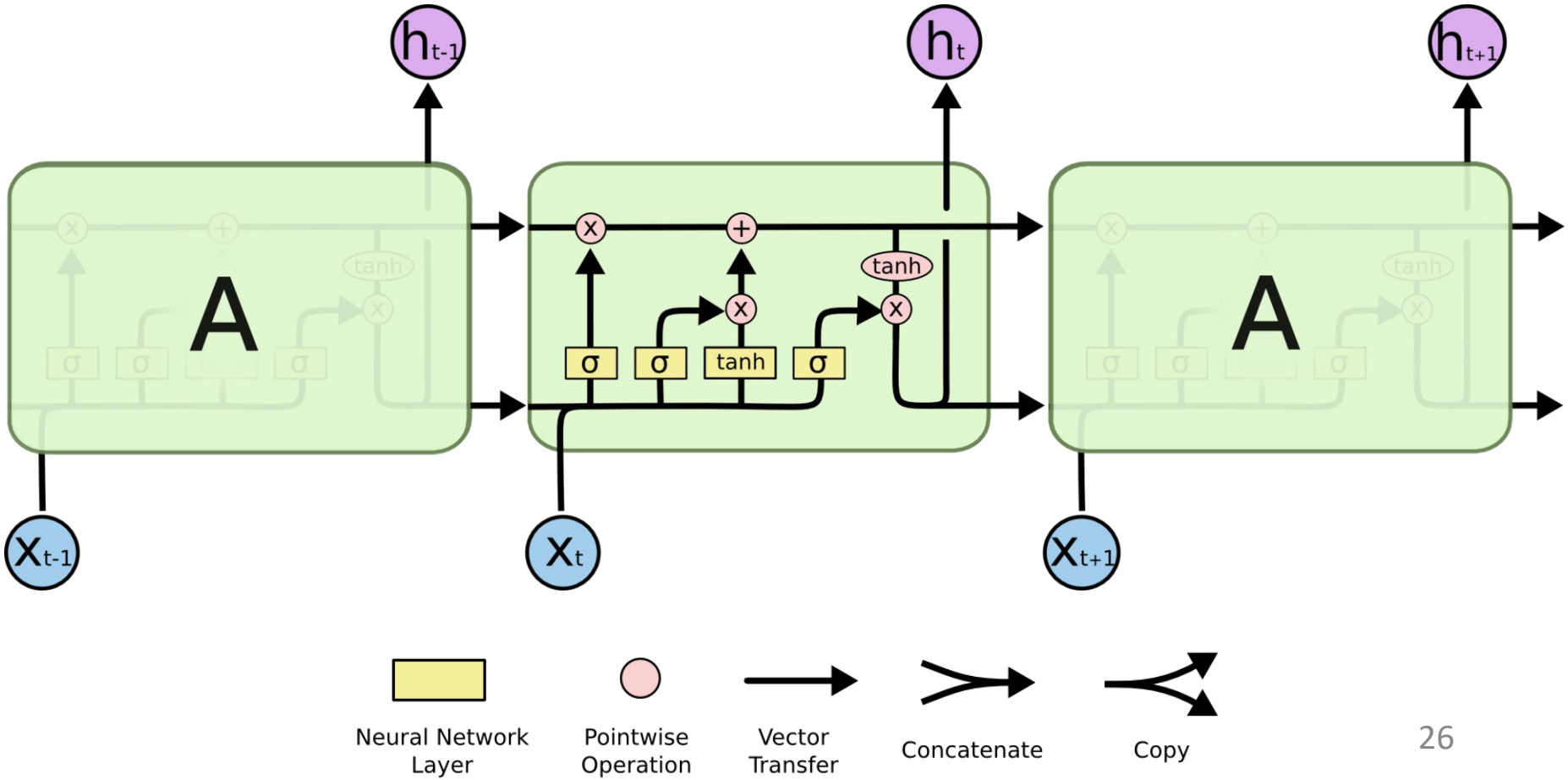


http://colah.github.io/posts/2015-08-Understanding-LSTMs

https://habr.com/ru/company/wunderfund/blog/331310

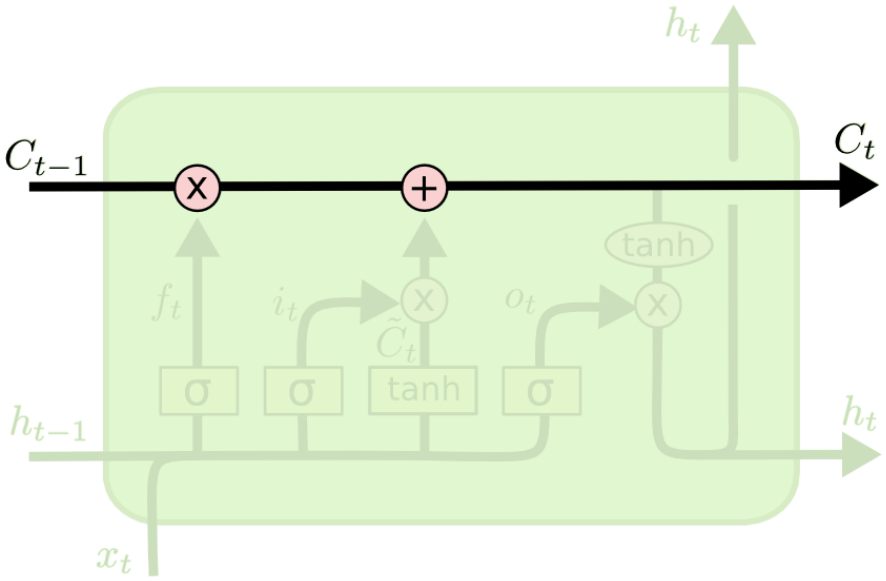
LSTM

- LSTM:


h_{t-1}
h_t
h_{t+1}
A
A
σ
σ
tanh
σ
tanh
X_{t-1}
X_t
X_{t+1}
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

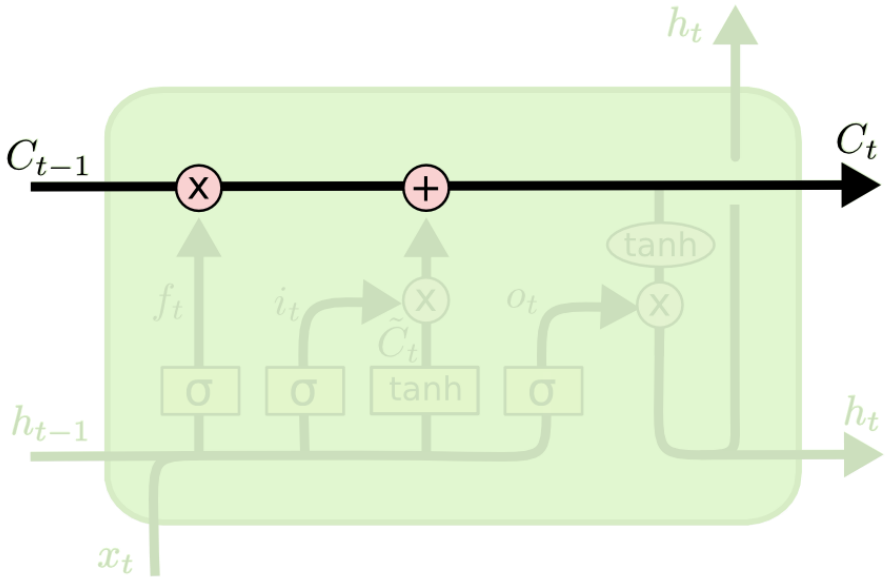
LSTM

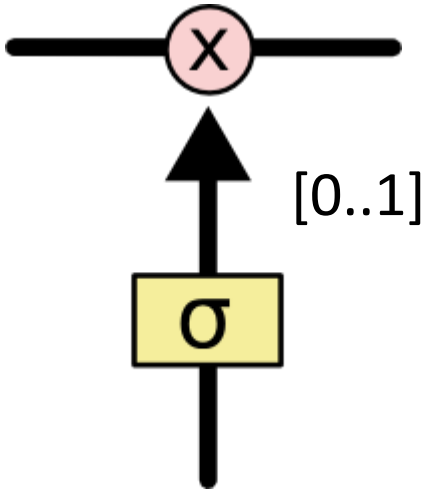
- *Cell state* (состояние ячейки):



LSTM

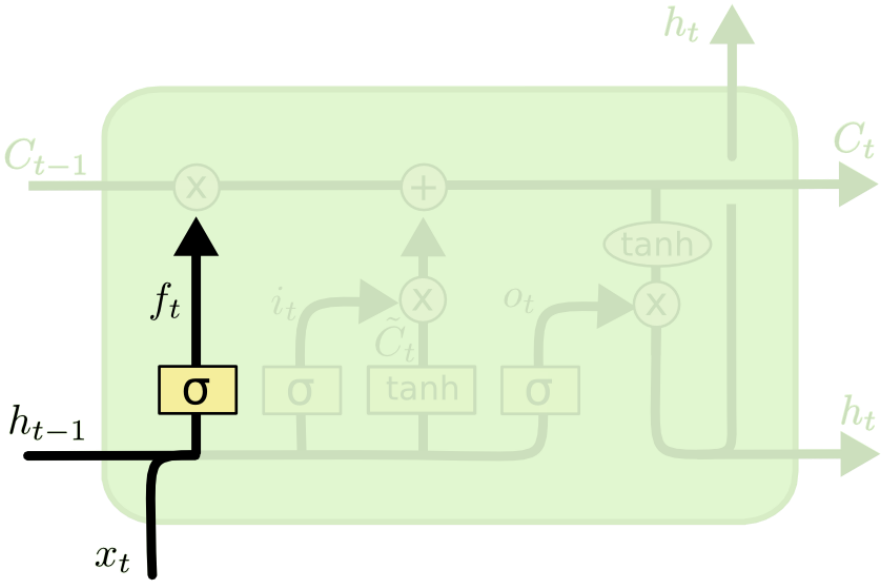
- *Gates* (вентили, фильтры, гейты) – контролируют состояние ячейки (обновляют и удаляют информацию):





LSTM

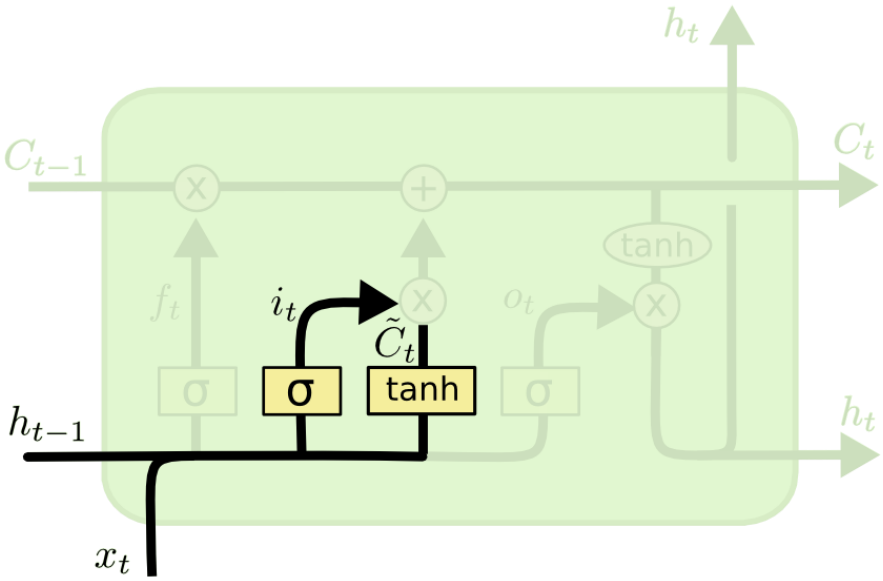
- *Forget gate* («вентиль забывания»):



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

LSTM

- *Input gate* (входной вентиль) и *candidate cell state* (кандидат в состояние ячейки):

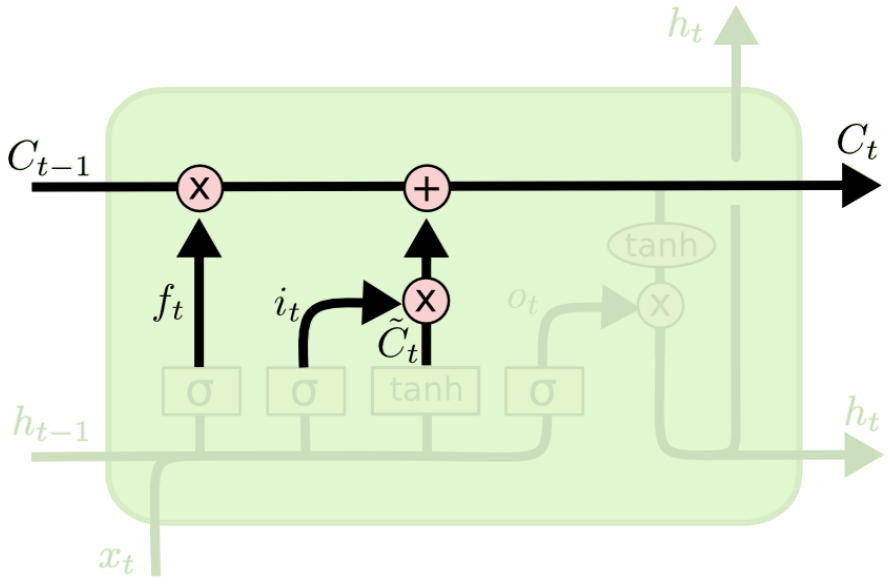


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

LSTM

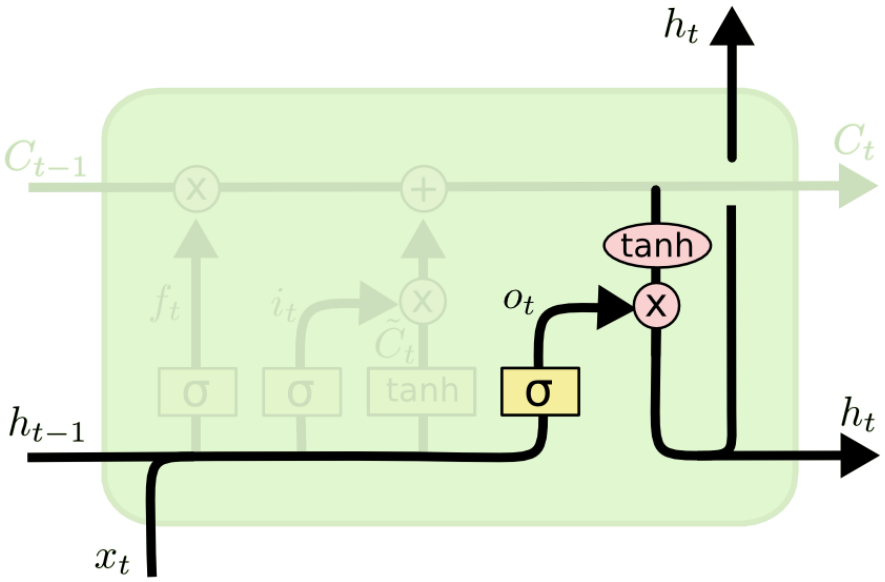
- Изменение состояния ячейки – забывание и обновление (“$*$” – поэлементное умножение):



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

LSTM

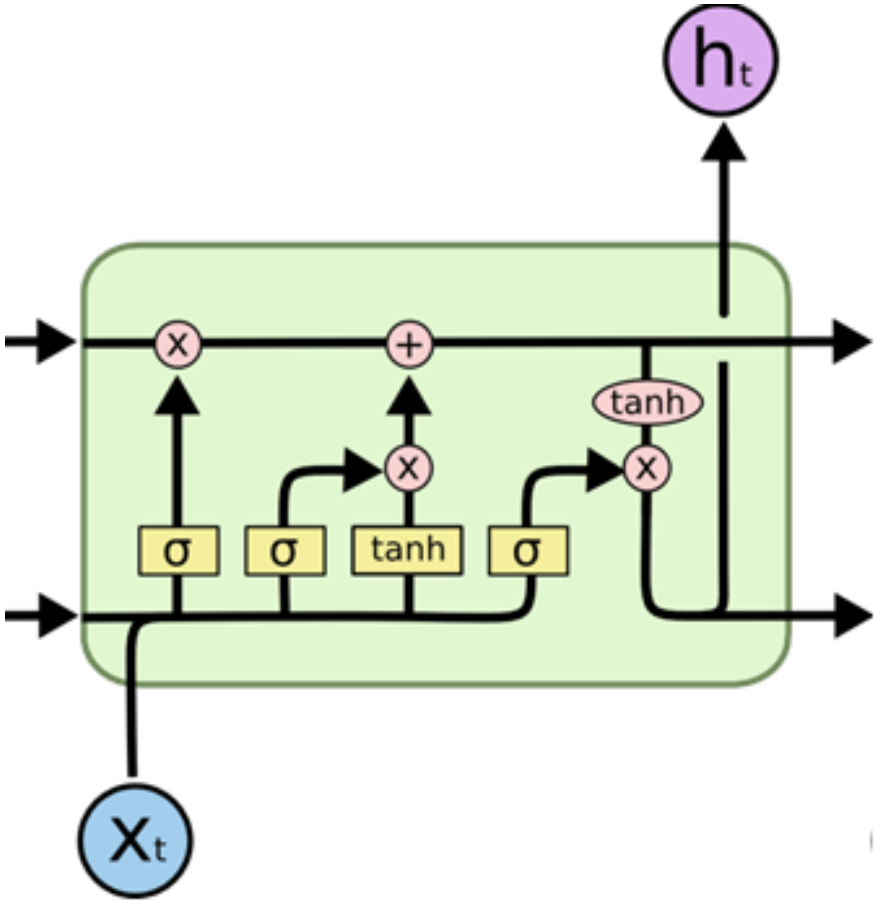
- *Output gate* (выходной вентиль) и выход ячейки:



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

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

LSTM



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

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

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

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

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

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

Карусель константной ошибки

- Пусть $f_t = 1$ (забывающий вентиль всегда пропускает сигнал), тогда:

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

$$\frac{dC_t}{dC_{t-1}} = 1$$

– состояние ячейки может сохранять свои значения сколько угодно долго, пока сама не «решит» их перезаписать – решение проблемы затухающих градиентов

- Свободный коэффициент b_f должен быть инициализирован значением ≥ 1 (в PyTorch – нет)

LSTM и Simple RNN

LSTM эквивалентна Simple RNN если:

- $f_t = 0$ (forget gate)
- $i_t = 1$ (input gate)
- $o_t = 1$ (output gate)
- (не считая tanh)

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

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

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

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

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

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

LSTM: реализация в PyTorch

```
torch.nn.LSTM(
    input_size,           # размерность входа
    hidden_size,          # размерность скрытого состояния
    num_layers=1,         # количество слоёв
    bias=True,            # свободные коэффициенты
    batch_first=False,    # расположение батча во входе
    dropout=0,            # вероятность дропаута
    bidirectional=False,  # двунаправленность
    proj_size=0           # выходной слой проекции
)
```

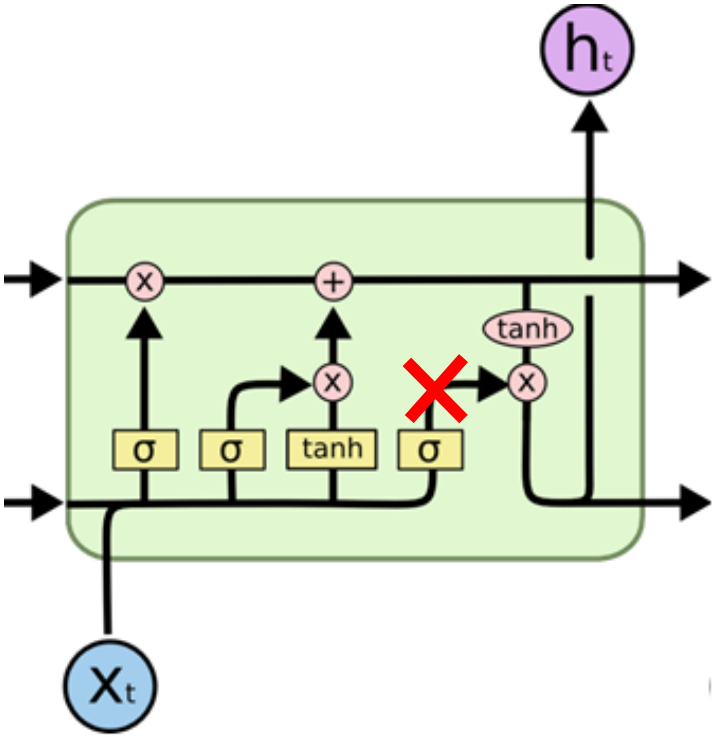
https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

См. “lstm_gru.ipynb”

Вариации LSTM

Peepholes («замочные скважины»):

- в случае, если $o_t \to 0$, вентили, управляющие состоянием ячейки, не имеют доступа к ней, поскольку $h_t \to 0$



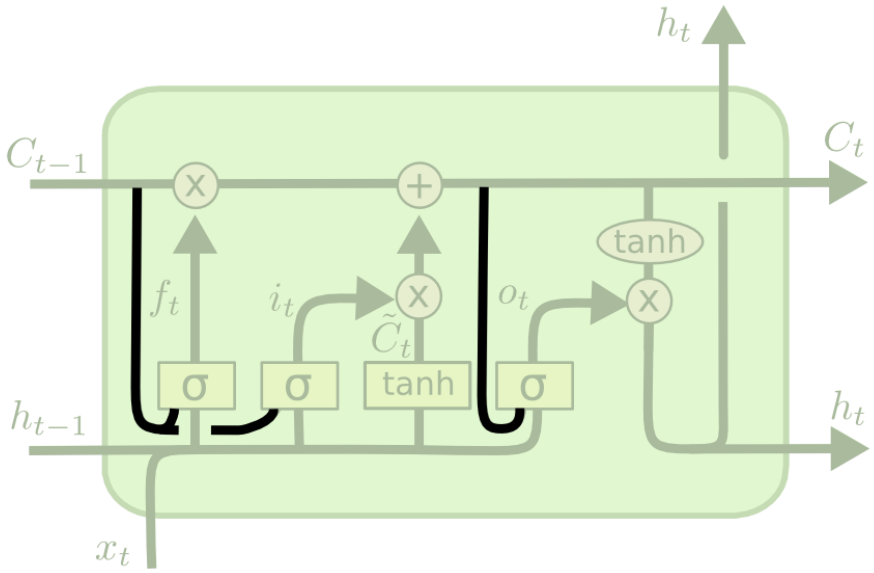
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

Вариации LSTM

Peepholes («замочные скважины»):

- в случае, если $o_t \to 0$, вентили, управляющие состоянием ячейки, не имеют доступа к ней



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

В PyTorch не реализованы

Вариации LSTM

Greff K., Srivastava R.K., Koutník J., Steunebrink B.R., Schmidhuber J. LSTM: A Search Space Odyssey // IEEE Transactions on Neural Networks and Learning Systems. 2016

- LSTM без входного вентиля
- LSTM без забывающего вентиля
- LSTM без выходного вентиля
- LSTM без функции активации σ на входном вентиле
- LSTM без функции активации σ на выходном вентиле
- LSTM без замочных скважин
- LSTM со связанными входным и забывающим вентилями

Вариации LSTM

Greff K., Srivastava R.K., Koutník J., Steunebrink B.R., Schmidhuber J. LSTM: A Search Space Odyssey // IEEE Transactions on Neural Networks and Learning Systems. 2016

Результаты:

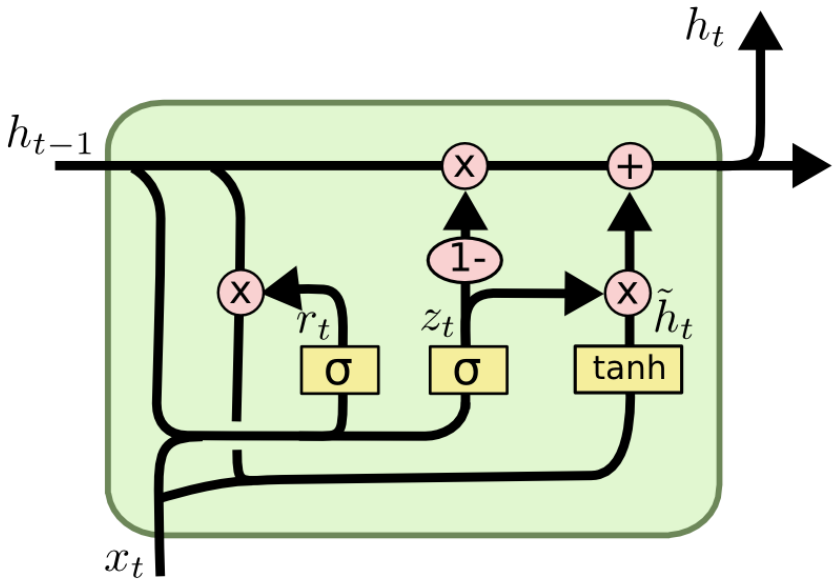
- ключевые компоненты – забывающий вентиль и функция активации на выходном вентиле
- ни один из вариантов не работает лучше базового
- более простые варианты, чем базовый, работают не хуже базового
- $\rightarrow$ возможность создания более простой архитектуры

GRU

- GRU (Gated Recurrent Unit) – управляемый рекуррентный блок
  - Cho K., van Merriënboer B., Bahdanau D., Bengio Y. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches // Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8). 2014
  - Cho K., van Merriënboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H., Bengio Y. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation // EMNLP-2014
- Simple RNN: 2 матрицы весов (U и W)
- LSTM: 8 матриц ($2 \times W_f$, $2 \times W_i$, $2 \times W_C$, $2 \times W_o$)
- GRU: 6 матриц ($2 \times W_z$, $2 \times W_r$, $2 \times W$)

GRU

- Скрытое состояние совмещено с состоянием ячейки
- Забывающий вентиль и входной вентиль объединены в вентиль обновления z (*update gate*)
- Добавлен вентиль перезагрузки r (*reset gate*)



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

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

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

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

GRU: реализация в PyTorch

```
torch.nn.GRU(
    input_size,           # размерность входа
    hidden_size,          # размерность скрытого состояния
    num_layers=1,         # количество слоёв
    bias=True,            # свободные коэффициенты
    batch_first=False,    # расположение батча во входе
    dropout=0,            # вероятность дропаута
    bidirectional=False   # двунаправленность
)
```

https://pytorch.org/docs/stable/generated/torch.nn.GRU.html

См. “lstm_gru.ipynb”