# Рекуррентные нейронные сети

Виктор Китов

v.v.kitov@yandex.ru

# Работа с последовательностями

- Данные в виде последовательности
  - динамика цен на акции
  - динамика действий посетителя веб-сайта
  - динамика погоды
  - предложения - последовательности слов
  - речь - последовательность звуков
  - видео - последовательность кадров
- Для текстов: необходимо представление входов небольшой фикс. длины
  - Word2Vec, glove, и др.
- Возможно использование сверточных нейросетей над последовательностями
  - но свертка имеет ограниченную область видимости в историю
- Решение: рекуррентные нейросети (Recurrent neural net, RNN)
  - помнят (в теории) всю историю

# Реккурентные нейросети

- Входная последовательность $\mathbf{x_{i:j}} := \mathbf{x_i}, ...\mathbf{x_j}$, $\mathbf{x_i} \in \mathbb{R}^{d_{in}}$.
- RNN выдает вектор фикс. размера $\widehat{\mathbf{y}}_\mathbf{n} \in \mathbb{R}^{d_{out}}$ :

$$\widehat{\mathbf{y}}_\mathbf{n} = RNN(\mathbf{x_{1:n}})$$

- Варьируя $n$ получаем отображения RNN* из последовательности в последовательность:

$$\widehat{\mathbf{y}}_{\mathbf{1:n}} = RNN^*(\mathbf{x_{1:n}})$$
$$\widehat{\mathbf{y}}_\mathbf{i} = RNN(\mathbf{x_{1:i}})$$

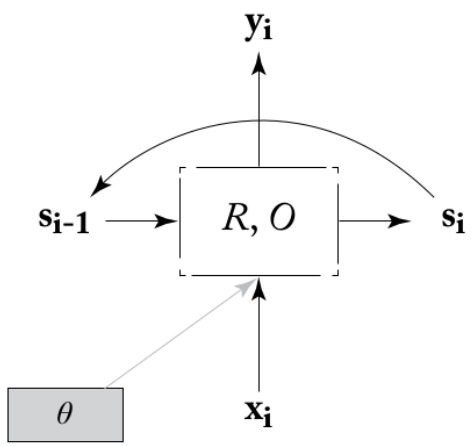
- Т.к. RNN сжимает всю историю $\mathbf{x_{1:n}}$ в вектор фикс. размера $\mathbf{y_n}$, его можно подавать как вектор признаков др. модели
  - например MLP

# Модель рекуррентной нейросети

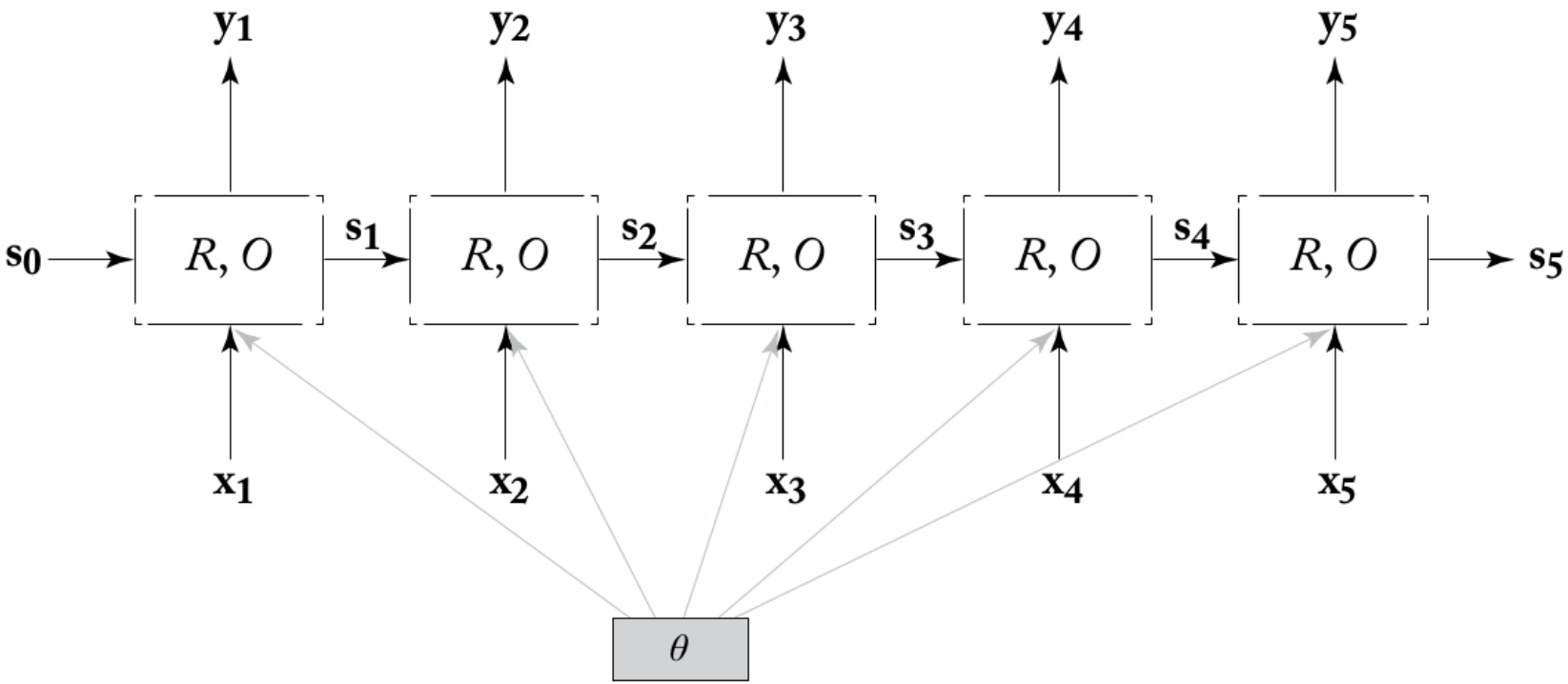
$$RNN^*(\mathbf{x_{1:n}}, \mathbf{s_0}) = \mathbf{y_{1:n}}$$
$$\widehat{\mathbf{y}}_\mathbf{i} = O(\mathbf{s_i})$$
$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i})$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}}, \mathbf{y_i} \in \mathbb{R}^{d_{out}}, \mathbf{s_i} \in \mathbb{R}^{d_{state}}$$

Обычно $O(\mathbf{s}) \equiv \mathbf{s}$, $d_{state} = d_{out}$, $s_0 = \mathbf{0}$.



# Развернутая нейросеть (unrolled RNN)




$$\begin{aligned} s_4 &= R(s_3, x_4) = R(R(s_2, x_3), x_4) \\ &= R(R(R(s_1, x_2), x_3), x_4) = R(R(R(R(s_0, x_1), x_2), x_3), x_4) \end{aligned}$$

# Обучение

Обучение: развернуть RNN и использовать неизменность весов для разных $t$.

- называется **backpropagation through time** (BPTT)
- на практике: развернуть RNN для всех не пересекающихся подпоследовательностей фикс. длины из длинной последовательности.

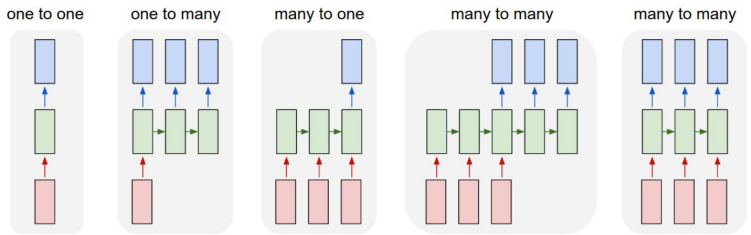
```
init s_0

for i in 0,1,...n/k-1:
    ŷ_{ki+1:ki+k} = RNN*(x_{ki+1:ki+k}, s_{ki})
    calculate loss Σ_{j=ki+1}^{ki+k} L(ŷ_j, y_j)
    backpropagate gradients, update weights
```

- Распарралелить вычисления в рамках последовательности нельзя - последующее состояние зависит от предыдущего.

# Архитектуры обычной и реккурентных нейросетей

Архитектуры обычной и реккурентных нейросетей:




- **one to one:** классическая классификация и регрессия в ML.
- **one to many:** описание изображений (image captioning), генерация текстов по заданной теме.
- **many to one:** классификация текстов, например определение тональности (sentiment analysis).
- **many to many:** машинный перевод, суммаризация длинного текста.
- **synced many to many:** разметка частей речи, определение событий на видео, распознавание речи.

# Варианты применения RNN

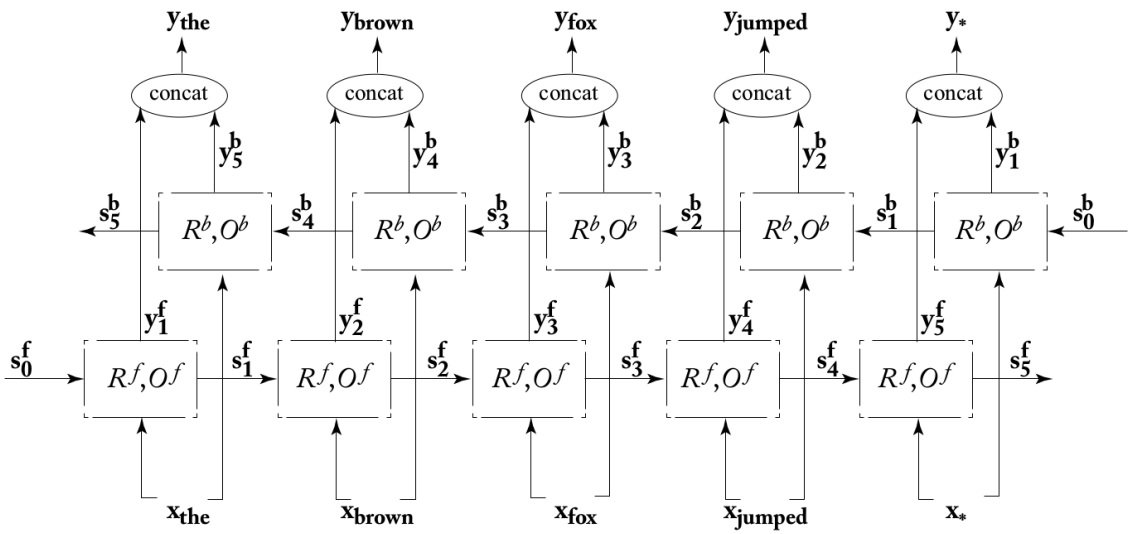
- **Acceptor**: выдает итоговый $\widehat{\mathbf{y}}_\mathbf{n}$.
  - пример: прочитать комментарий и указать его полярность.
- **Encoder**: закодировать последовательность в виде высокоразмерного $\widehat{\mathbf{y}}_\mathbf{n}$
  - машинный перевод: перевод выполняется декодирующей RNN, стартующей из $\mathbf{s_0} = \widehat{\mathbf{y}}_\mathbf{n}$.
  - описание текста вектором фикс. длины $\widehat{\mathbf{y}}_\mathbf{n}$, на основе которого будет производиться суммаризация текста
- **Transducer**: по $x_1, ...x_n$ выдать $y_1, ...y_n$.
  - разметка частей речи, предсказание следующего слова (языковое моделирование), текст->речь, речь->текст
  - Ф-ция потерь:

$$\mathcal{L}\left(\widehat{\mathbf{y}}_{\mathbf{1:n}}, \mathbf{y}_{\mathbf{1:n}}\right) = \sum_{i=1}^{n} L\left(\widehat{\mathbf{y}}_\mathbf{i}, \mathbf{y}_\mathbf{i}\right)$$

# Содержание

# Двунаправленная RNN

- Двунаправленная рекуррентная нейросеть (bidirectional RNN) состоит из 2x RNN:
  - forward RNN $\left(R^f, O^f\right)$ с состоянием $\mathbf{s_i^f}, i = \overline{1,n}$
  - backward RNN $\left(R^b, O^b\right)$ с состоянием $\mathbf{s_i^b}$, $i = \overline{1,n}$
- Forward RNN идет слева-направо $\mathbf{x_1}, \mathbf{x_2} \ldots \mathbf{x_n}$.
- Backward RNN идет справа-налево $\mathbf{x_n}, \mathbf{x_{n-1}} \ldots \mathbf{x_1}$.



# Двунаправленная RNN

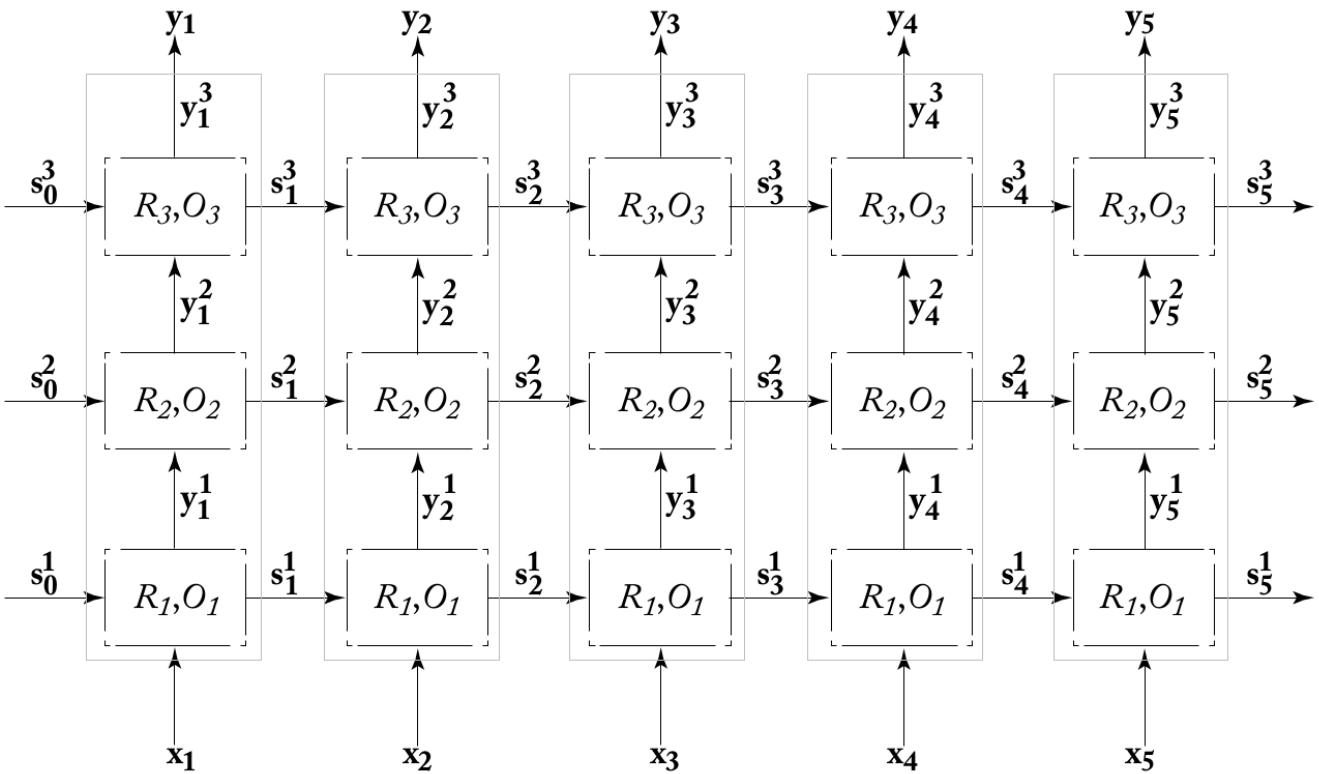
- В каждый момент $i$ имеем 2 состояния:
  1. $\mathbf{s_i^f} = F_1(\mathbf{x_1}, \mathbf{x_2}...\mathbf{x_i})$
  2. $\mathbf{s_i^b} = F_2(\mathbf{x_i}, \mathbf{x_{i+1}}, ...\mathbf{x_n})$
- Для каждого $i$ выдаем их конкатенацию, которую подаем на вход др. модели (MLP)

$$biRNN(\mathbf{x_{1:n}}, i) = \widehat{\mathbf{y}}_\mathbf{i} = [\widehat{\mathbf{y}}_\mathbf{i}^\mathbf{f}; \widehat{\mathbf{y}}_\mathbf{i}^\mathbf{b}] = [RNN^f(\mathbf{x_{1:i}});\ RNN^b(\mathbf{x_{n:i}})]$$

$$biRNN^*(\mathbf{x_{1:n}}) = \mathbf{y_{1:n}} = [biRNN(\mathbf{x_{1:n}}, 1); ...; biRNN(\mathbf{x_{1:n}}, n)]$$

- Двунаправленная RNN эффективно размечает последовательность (режим Transducer), например по тексту разметить части речи.
  - т.к. выход учитывает контекст и справа, и слева.

# Многослойная RNN (stacked RNN)




- Выход предыдущего слоя - вход для следующего.
- На практике работают точнее однослойных RNN.
- Можно использовать многослойную двунаправленную RNN.

# Содержание

# Bag-of-words RNN

Bag-of-words RNN:

$$s_i = s_{i-1} + x_i$$
$$y_i = s_i$$

- $x_i$: вход
- $s_i$: скрытый слой
- $y_i$: выход

Порядок входов не имеет значения, поэтому на практике не применяется.

# Сеть Элмана

Сеть Элмана (Elman net, simple RNN):

$$\mathbf{s_i} = g_s\left(W_s\mathbf{s_{i-1}} + V_s\mathbf{x_i} + \mathbf{b_s}\right)$$
$$\mathbf{y_i} = g_y(W_y\mathbf{s_i} + \mathbf{b_y})$$

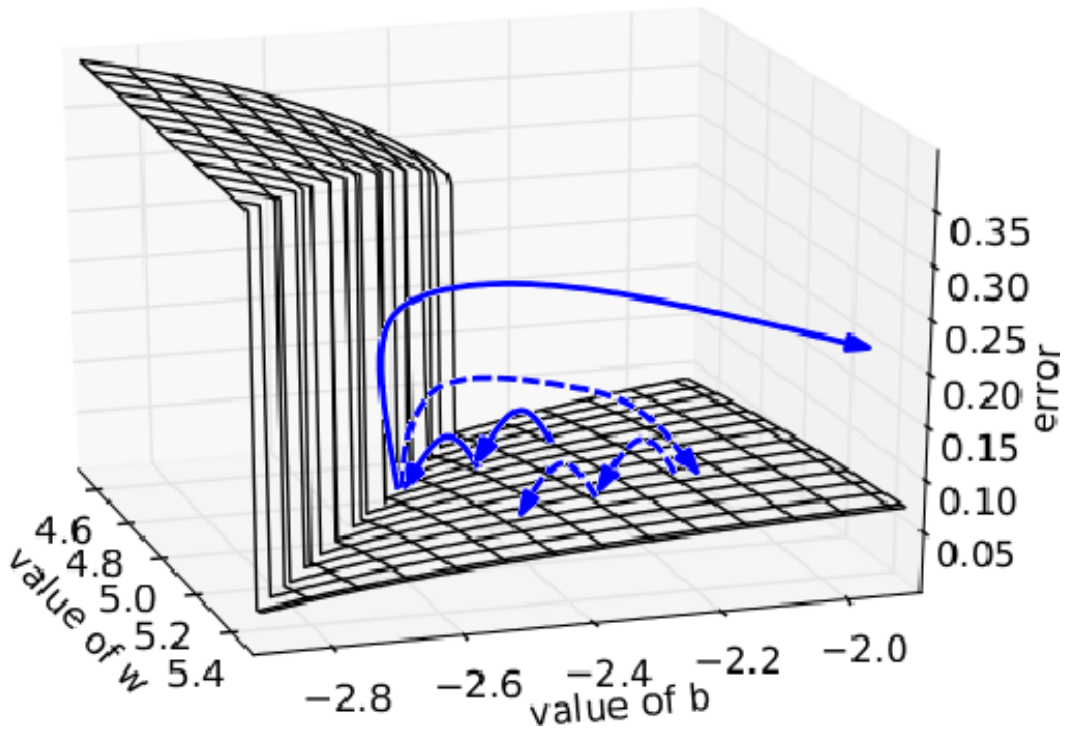
- $\mathbf{x_i}$: вход
- $\mathbf{s_i}$: скрытый слой
- $\mathbf{y_i}$: выход
- $W_s, V_s, W_y$: матрицы параметров
- $\mathbf{b_s}, \mathbf{b_y}$: векторы параметров
- $g_s(\cdot), g_y(\cdot)$: функции активации

# Свойства сети Элмана

- Сеть Элмана чувствительна к порядку входов.
- Зависимость $s_i \leftarrow s_{i-1} \leftarrow s_{i-1} \leftarrow \cdots \leftarrow s_0$ через $W_s$.
- Из-за рекуррентной зависимости от $W_s$ возможны:
  - проблема взрывающегося градиента (exploding gradient) при $|\lambda| \gg 1$
  - проблема затухающего градиента (vanishing gradient) при $|\lambda| \approx 0$

# Проблема взрывающегося градиента

Exploding gradient problem:



# Проблема взрывающегося градиента

Решение проблемы взрывающегося градиента:

- добавить регуляризацию
- обрезать норму градиента по порогу

$$\text{если } \|\nabla_w L(\widehat{y}_i, y_i)\| \leq t: \quad w \to w - \varepsilon \nabla_w L(\widehat{\mathbf{y}}_\mathbf{i}, \mathbf{y}_\mathbf{i})$$

$$\text{если } \|\nabla_w L(\widehat{y}_i, y_i)\| > t: \quad w \to w - \varepsilon \frac{t}{\|\nabla_\theta L(\widehat{\mathbf{y}}_\mathbf{i}, \mathbf{y}_\mathbf{i})\|} \nabla_\theta L(\widehat{\mathbf{y}}_\mathbf{i}, \mathbf{y}_\mathbf{i})$$

# Проблема затухающего градиента

- Инициализировать $W_s = I, b_s = 0$, $g_s = ReLU,\ LeakyReLU$.
- Добавить регуляризацию на несильное отклонение от ортогональности

$$\left\| W_s^T W_s - I \right\|_F^2 \text{ либо } \left\| W_s^T W_s - I \right\|_F^2 + \left\| W_s W_s^T - I \right\|_F^2$$

  т.к. у ортогональных матриц все $|\lambda_i| = 1$[1].
- Разделить состояние на быстро и медленно меняющиеся компоненты: $s_t = [s_t^{slow}; s_t^{fast}]$
  - $s_t^{slow} = \alpha s_{t-1}^{slow} + (1-\alpha) W_{slow} x_t,\ \alpha \lesssim 1$
  - $s_t^{fast} = \sigma\left(V_x x_t + V_{fast} s_{t-1}^{fast} + V_{slow} s_t^{slow}\right)$

[1] Докажите.

# Содержание

# Проблема обычной RNN

- Проблема обычной RNN: вся память перезаписывается на каждой итерации.
  - поэтому RNN быстро забывает прошлое (или градиент взрывается)
- Важно помнить прошлое глубоко в истории:
  - автоматические ответы на вопросы
  - машинный перевод
  - суммаризация текстов

# Проблема обычной RNN

- Проблема обычной RNN: вся память перезаписывается на каждой итерации.
  - поэтому RNN быстро забывает прошлое (или градиент взрывается)
- Важно помнить прошлое глубоко в истории:
  - автоматические ответы на вопросы
  - машинный перевод
  - суммаризация текстов
- Решение: использование вентилей

# Вентили

- Пусть **s**-старое состояние, **x**-новый вход, **s**$'$-новое состояние, $s, s', x \in \mathbb{R}^n$.
- Вентиль $g \in \{0,1\} \in \mathbb{R}^n$ контролирует позиции, которые нужно обновить:
- Пример ($\odot$ - поэлементное умножение):

$$\underbrace{\begin{bmatrix}8\\11\\3\\7\\5\\15\end{bmatrix}}_{\mathbf{s}'} \leftarrow \underbrace{\begin{bmatrix}0\\1\\0\\0\\0\\1\end{bmatrix}}_{\mathbf{g}} \odot \underbrace{\begin{bmatrix}10\\11\\12\\13\\14\\15\end{bmatrix}}_{\mathbf{x}} + \underbrace{\begin{bmatrix}1\\0\\1\\1\\1\\0\end{bmatrix}}_{(\mathbf{1}-\mathbf{g})} \odot \underbrace{\begin{bmatrix}8\\9\\3\\7\\5\\8\end{bmatrix}}_{\mathbf{s}}$$

- Какие вентили перекрывать? нужно их настраивать.
- Кусочно-постоянный вентиль не сможем оптимизировать.
  - поэтому используем гладкий вентиль $\mathbf{g} = \sigma(f(\mathbf{x}, \mathbf{s}, \theta))$
  - $\theta$: настраиваемые параметры
  - $f$: любая гладкая функция

# Популярные RNN с вентилями

Популярные RNN с вентилями:

- Long short-term memory (LSTM)
  - "сеть долгой кратковременной памяти"
- Gated recurrent unit (GRU)

RNN с вентилями работают лучше на больших данных чем простые RNN.

# Long short-term memory (LSTM)

$$\mathbf{f_t} = \sigma\left(W_f \mathbf{x_t} + U_f \mathbf{h_{t-1}} + \mathbf{b_f}\right) \quad \text{forget gate}$$

$$\mathbf{i_t} = \sigma\left(W_i \mathbf{x_t} + U_i \mathbf{h_{t-1}} + \mathbf{b_i}\right) \quad \text{input gate}$$

$$\mathbf{o_t} = \sigma\left(W_o \mathbf{x_t} + U_o \mathbf{h_{t-1}} + \mathbf{b_0}\right) \quad \text{output gate}$$

$$\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot tanh\left(W_c \mathbf{x_t} + U_c \mathbf{h_{t-1}} + \mathbf{b_c}\right) \quad \text{inner state}$$

$$\mathbf{h_t} = \mathbf{o_t} \odot tanh\left(\mathbf{c_t}\right) \quad \text{observed output}$$

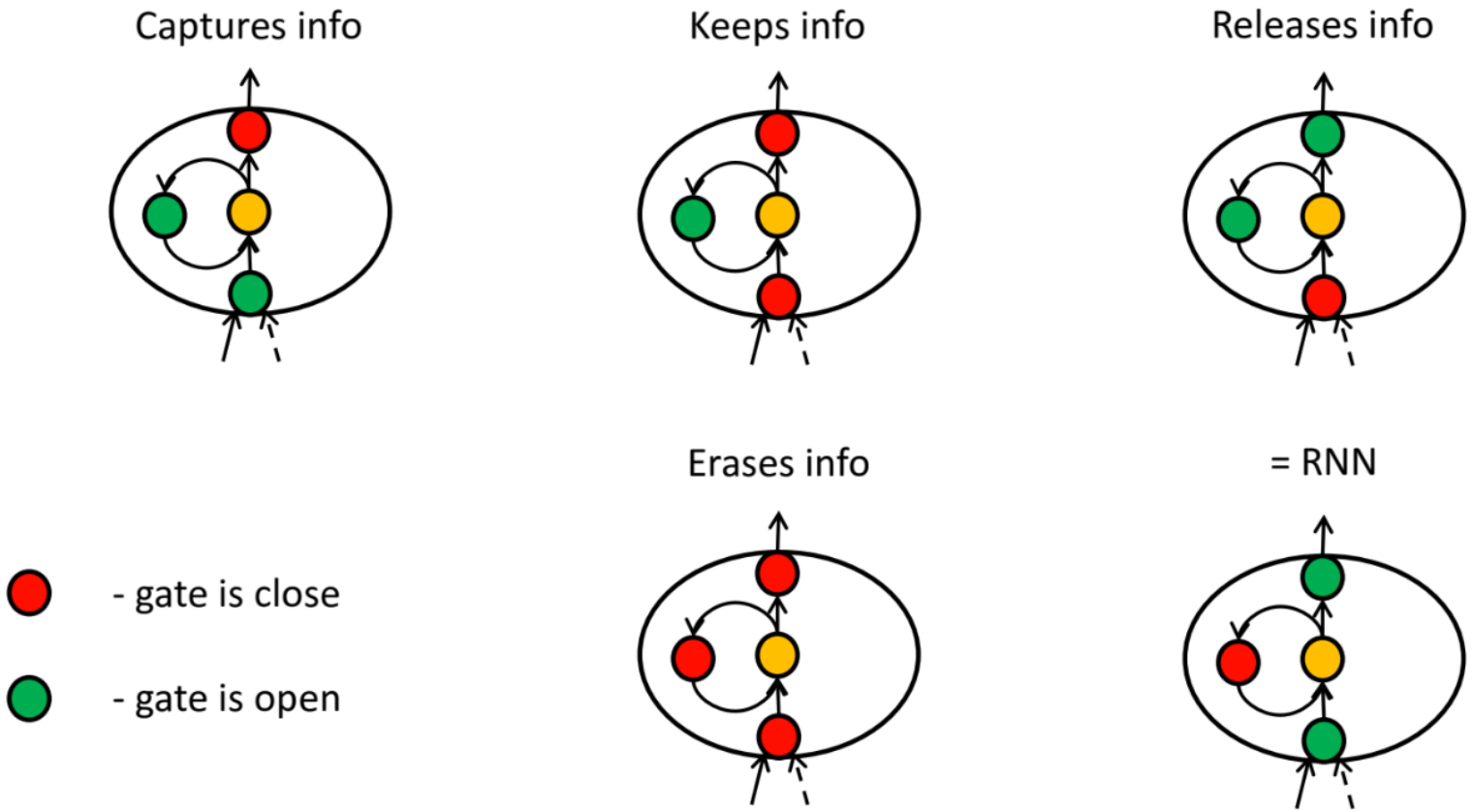
$\mathbf{x_t}$-вход, $\mathbf{h_t}$ -выход.

Параметры:

- матрицы: $W_f, U_f, W_i, U_i, W_o, U_o, W_c, U_c$
- вектора: $b_f, b_i, b_o, b_c$
- инициализация: $c_0, h_0$

# Иллюстрация

Вход-внизу, память-в середине, выход-наверху.



# Комментарии

- $\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \ldots$ обеспечивает более долгую память (но все равно не бесконечную - см. memory networks)
- Рекомендуется инициализация $\mathbf{b_f} \geq \mathbf{1}$
  - вначале сеть пытается запоминать все

# Gated recurrent unit (GRU)

$$\mathbf{z_t} = \sigma\left(W_z\mathbf{x_t} + U_z\mathbf{h_{t-1}} + \mathbf{b_z}\right)$$
$$\mathbf{r_t} = \sigma\left(W_r\mathbf{x_t} + U_r\mathbf{h_{t-1}} + \mathbf{b_r}\right)$$
$$\mathbf{h_t} = (1 - z_t) \odot \mathbf{h_{t-1}} + \mathbf{z_t} \odot tanh\left(W_h\mathbf{x_t} + U_h\left(\mathbf{r_t} \odot \mathbf{h_{t-1}}\right) + \mathbf{b_h}\right)$$

$\mathbf{x_t}$ - вход, $\mathbf{z_t}$ - forget gate, $\mathbf{r_t}$ - input gate, $\mathbf{h_t}$ - выход.
Параметры:

- матрицы: $W_z, U_z, W_r, U_r, W_h, U_h$
- вектора: $\mathbf{b_z}, \mathbf{b_r}, \mathbf{b_h}$
- инициализация: $\mathbf{h_0}$

# Комментарии

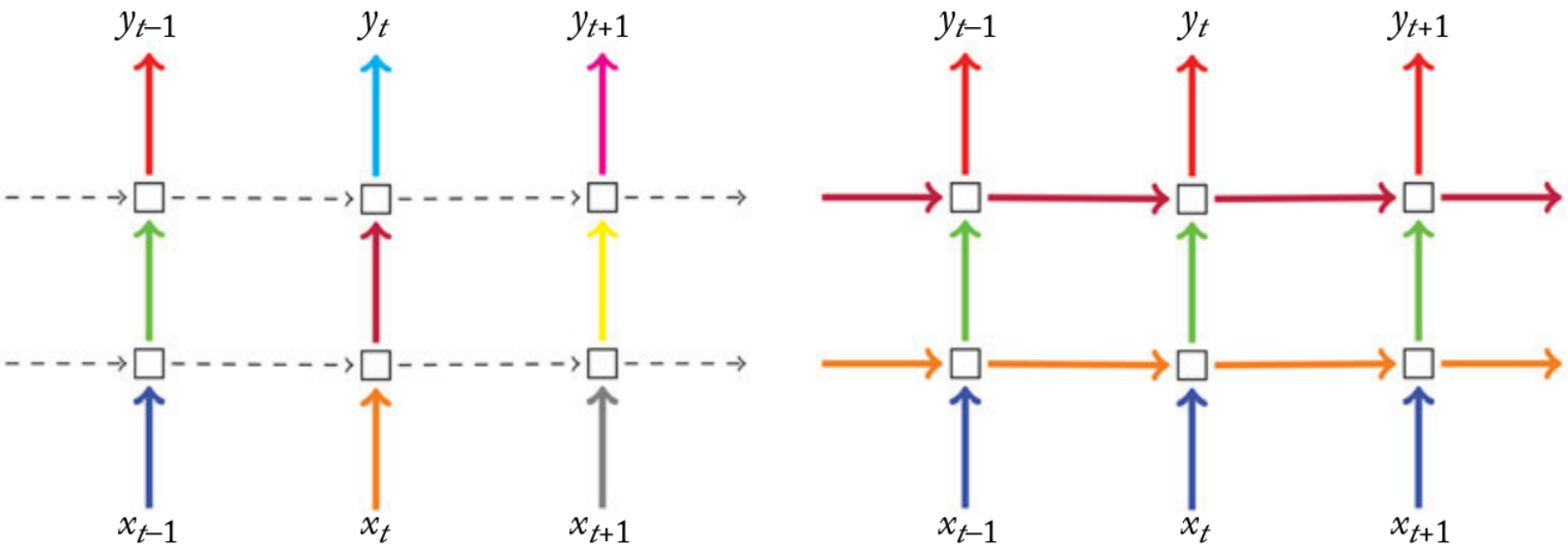
- По сравнению с LSTM GRU:
  - имеет 2, а не 3 вентиля
  - память и выход - то же самое
- GRU содержит меньше параметров и меньше переобучается.

# Dropout в RNN

- Проблема: dropout во времени нарушает временные связи.
- Решения
  1. (Pham et al. [2013]): применять dropout к нереккурентным связям.
  2. (Gal [2015], variational RNN dropout): для каждой последовательности $x_{1:T}$ регенерировать и фиксировать маску.
     - маска не меняется от времени.
     - маска регенерируется для другой последовательности.

# Иллюстрация вариантов dropout в RNN

Иллюстрация dropout в решениях 1 и 2.



# Заключение

- Рекуррентные нейросети - удобная модель для обработки и генерации последовательных данных.
- Специфические проблемы настройки рекуррентных нейросетей:
  - взрывающийся градиент
    - решение: обрезка градиента по норме
  - затухающий градиент
    - решение: использование архитектур с вентилями.