Тверской государственный технический университет

Интеллектуальные информационные системы

Рекуррентные нейронные сети

Материалы курса доступны по ссылке:
https://github.com/AndreyShpigar/ML-course

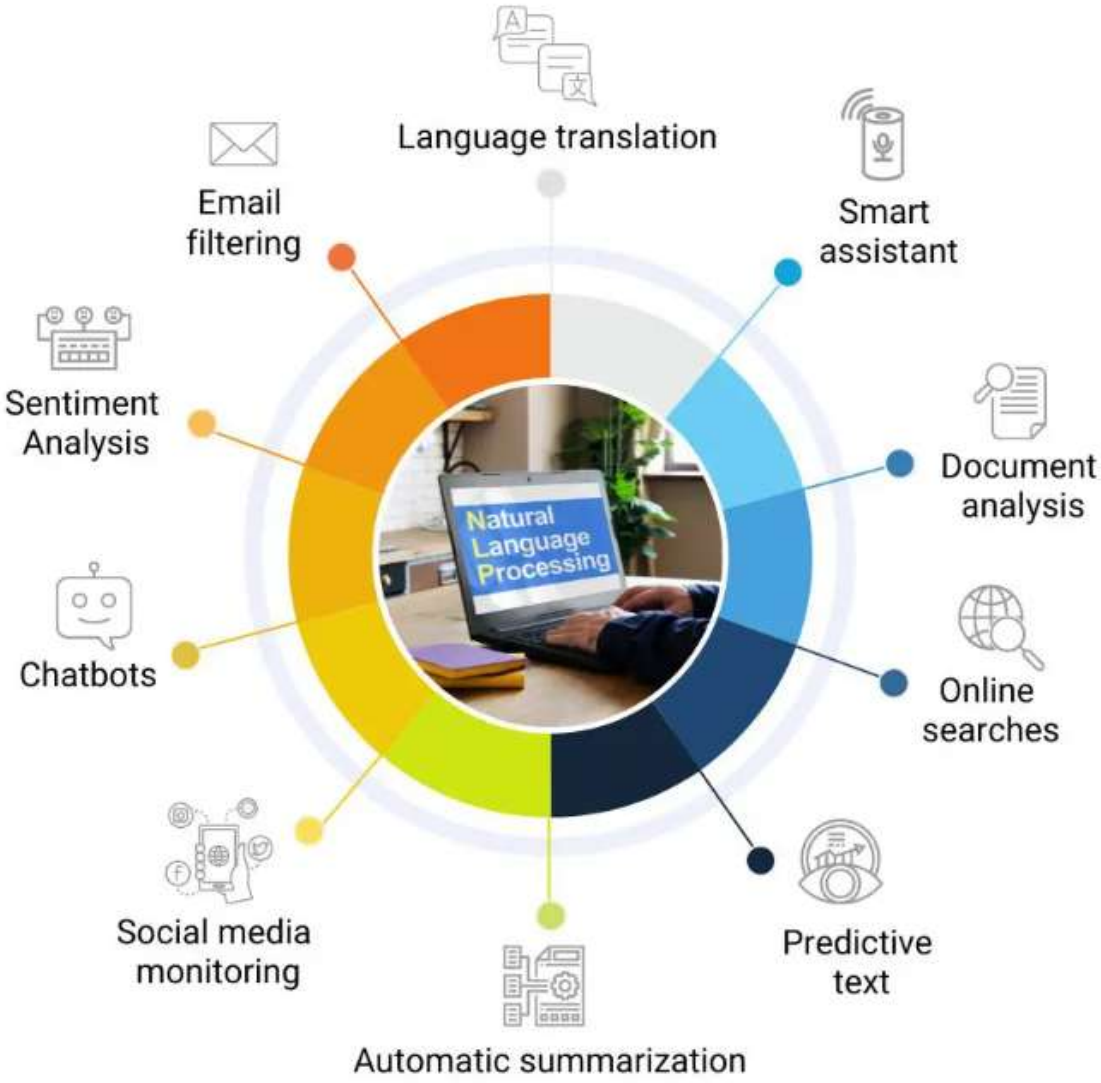
2024 г.

Задачи обработки последовательностей

Natural Language Processing (NLP) – анализ текстов, написанных на естественных языках:

- Классификация текста, анализ тональности документов, поиск релевантных документов по запросу и их ранжирование
- Синтез и распознавание речи
- Машинный перевод
- Генерация текста
- Диалоговые системы и чат-боты
- Суммаризация обращений

Applications of
Natural Language Processing

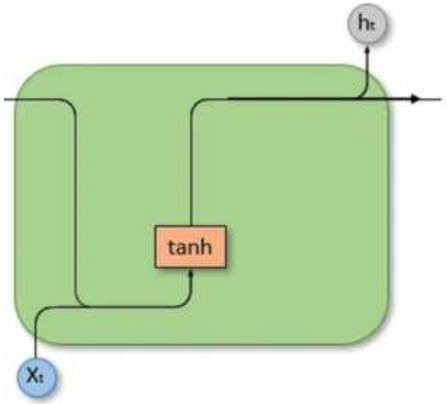


Задачи обработки последовательностей

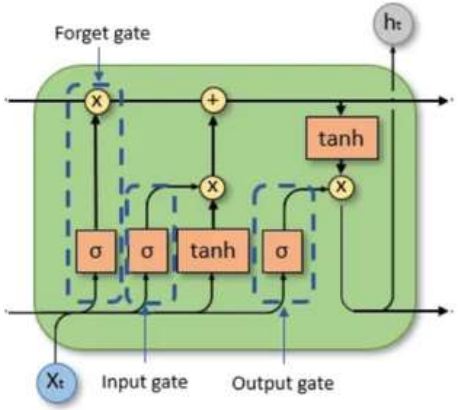
Временные ряды – значения меняющихся во времени признаков, полученные в некоторые моменты времени

- Прогнозирование временных рядов как самостоятельная задача – прогнозирование продаж, спроса, трафика, стоимости и множество других
- Обработка сигналов
- Поиск аномалий
- Интерпретация генома и задачи биоинформатики

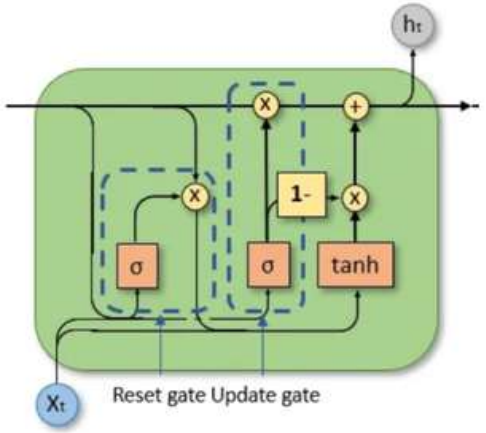
RNN



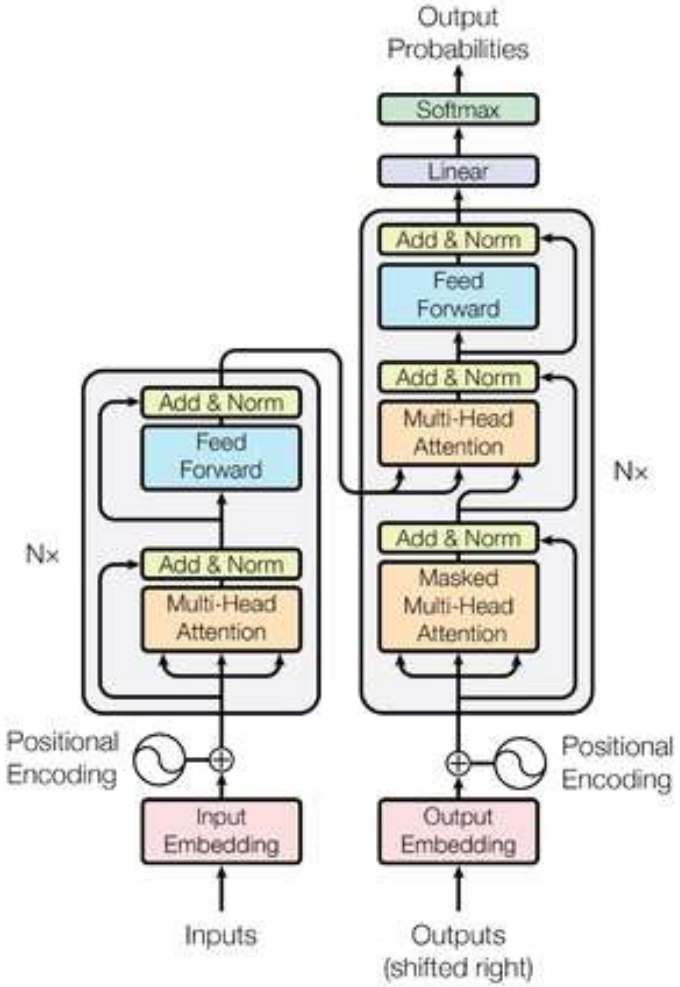
LSTM



GRU

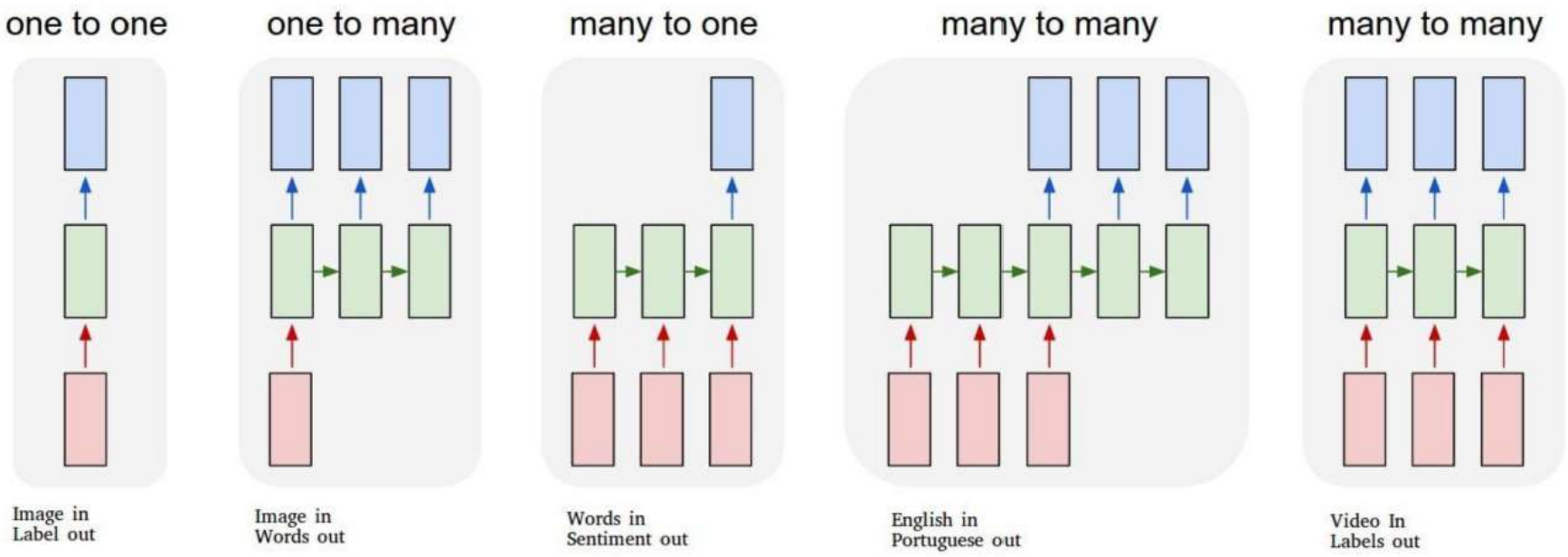


Transformers



Способы работы с последовательностями:

1. One-to-one
2. One to many
3. Many to one
4. Many to many (синхронизированный или нет)



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

Many-to-one

Было сложно, мне понравилось

↓

Позитивный

One-to-many

→ Пингвин в шляпе

Many-to-many, не синхронизированный

I seem to have overfitted

↓

Кажется, я переобучился

Many-to-many, синхронизированный

Будет	сложно,	вам	понравится
↓	↓	↓	↓
глагол	наречие	местоимение	глагол

Источник: https://education.yandex.ru/handbook/ml/article/nejroseti-dlya-raboty-s-posledovatelnostyami

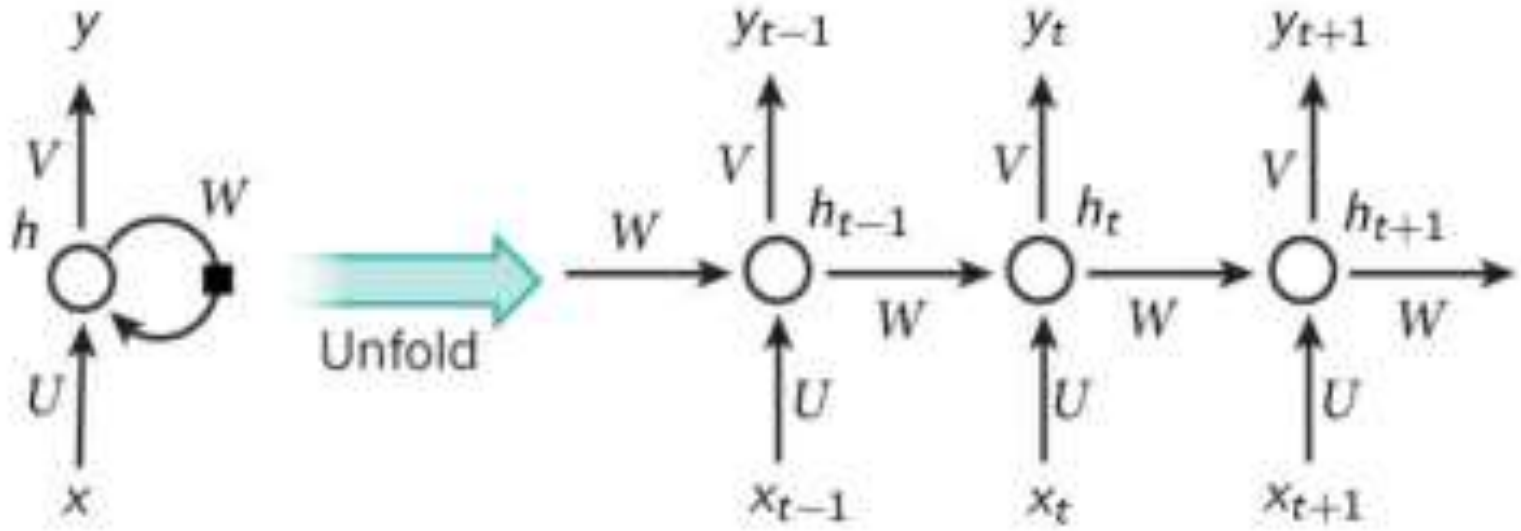
Задача обработки последовательностей

x_t - входной вектор в момент t

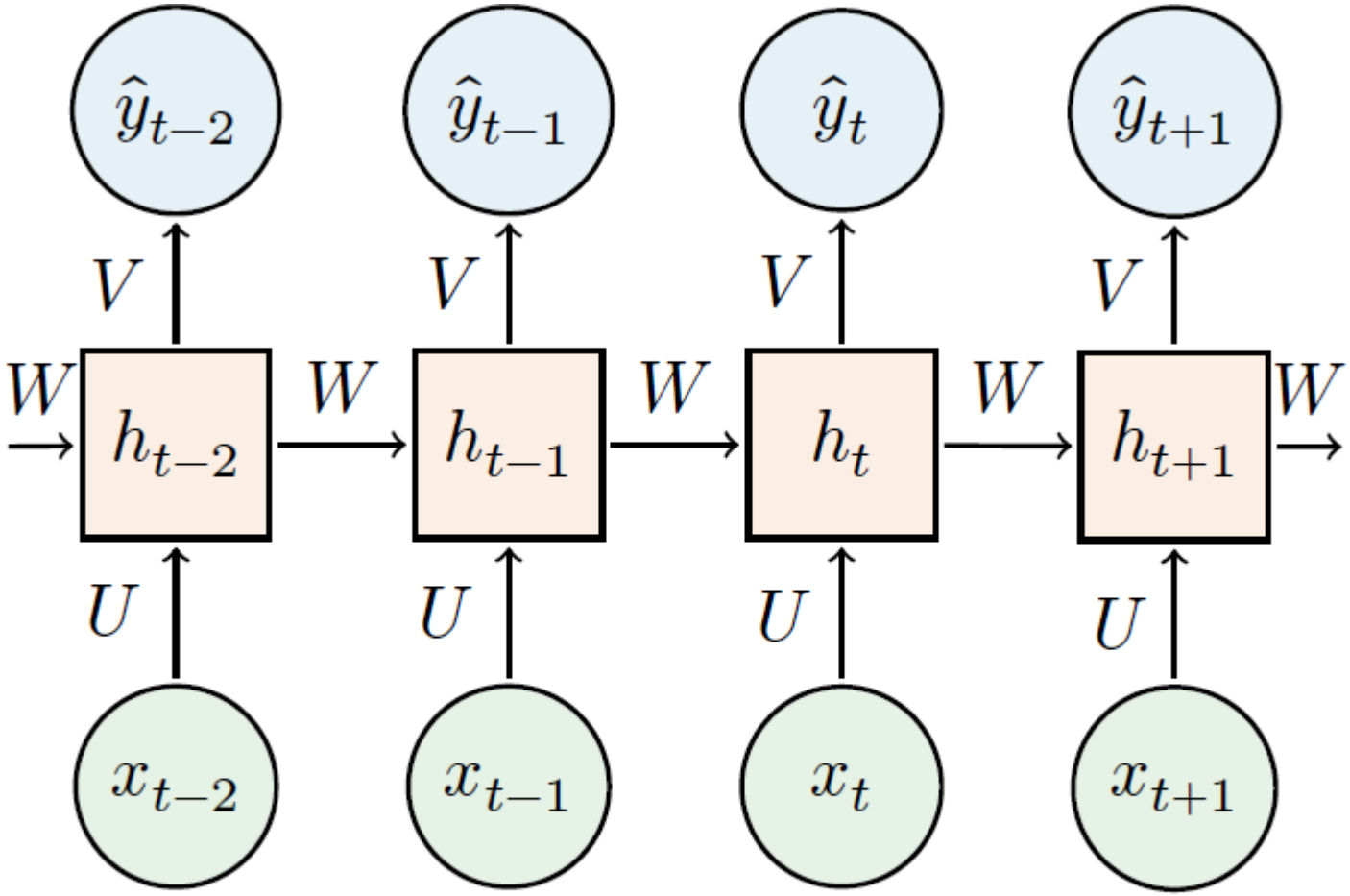
h_t - вектор скрытого состояния в момент t

y_t - выходной вектор

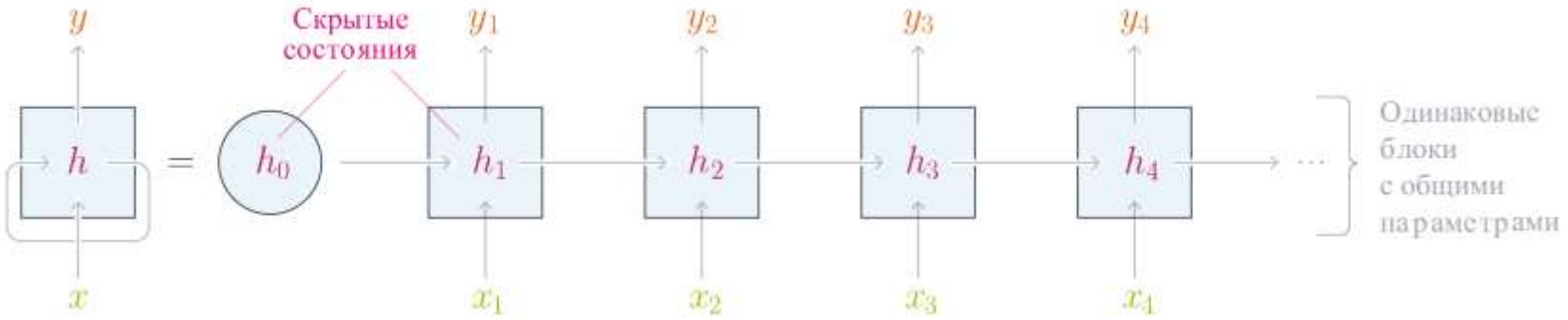
$$h_t = \sigma_h\left(U_{x_t} + W_{h_{t-1}}\right)$$
$$y_t = \sigma_y(V_{h_t})$$



Рекуррентную сеть можно рассматривать, как несколько копий одной и той же сети, каждая из которых передает информацию последующей копии



$$h_t = f_h(Ux_t + Wh_{t-1} + b_h) \qquad \hat{y}_t = f_y(Vh_t + b_y)$$



h_t - вектор скрытого состояния в момент t – «внутренняя память» - для хранения информации о предыдущих элементах последовательности. На каждом дискретном шаге в сеть подаются данные, при этом происходит обновление скрытого состояния:

$h_t = \tanh(h_{t-1}W_1 + x_tW_2)$

По скрытому состоянию предсказывается выходной сигнал:

$$y_t = h_tW_3$$

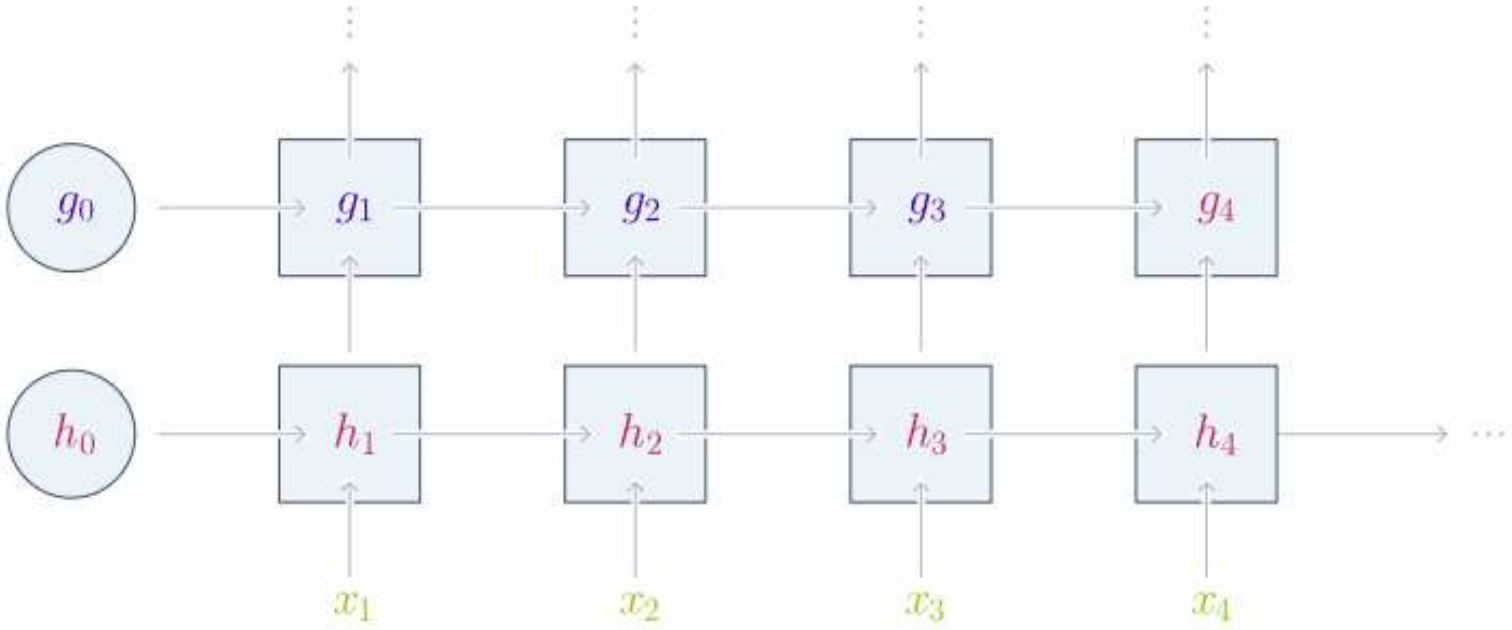
Веса W_i одинаковых на всех итерациях, то есть очередные x_t и h_{t-1} подаются на вход одного и того же слоя, зацикленного на себе

Функция потерь – суммарное отклонение по всем выходным сигналам y_t:

$$\sum_{t=0}^{T} \mathscr{L}_t(U, V, W) \to \min_{U,V,W}$$

$$\mathscr{L}_t(U, V, W) = \mathscr{L}\big(y_t(U, V, W)\big) - \text{потеря от предсказания } y_t$$

Источник: https://education.yandex.ru/handbook/ml/article/nejroseti-dlya-raboty-s-posledovatelnostyami



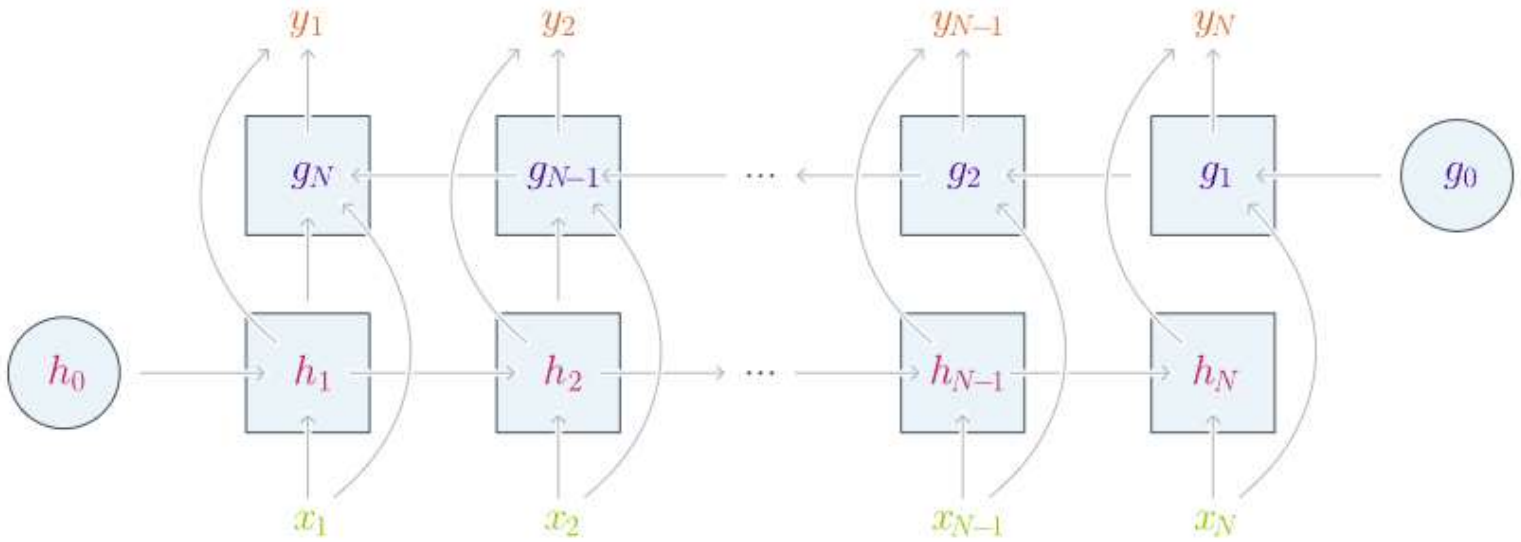
Глубокая RNN — несколько рекуррентных слоев:

- Первый слой RNN (первая сеть) принимает на вход исходную последовательность
- Второй слой RNN – принимает выходы первой сети
- Третий слой RNN – принимает выходы второй сети
- И так далее

Источник: https://education.yandex.ru/handbook/ml/article/nejroseti-dlya-raboty-s-posledovatelnostyami

Bidirectional RNN

Стандартная RNN учитывает только предыдущий контекст. Но, например, слово в предложении связано не только с предыдущими, но и с последующими словами. Для таких случаев используется двунаправленная рекуррентная сеть (BRNN) – состоящая из прямой (элементы подаются от первого к последнему) и обратной (элементы подаются в обратном порядке) рекуррентных сетей.



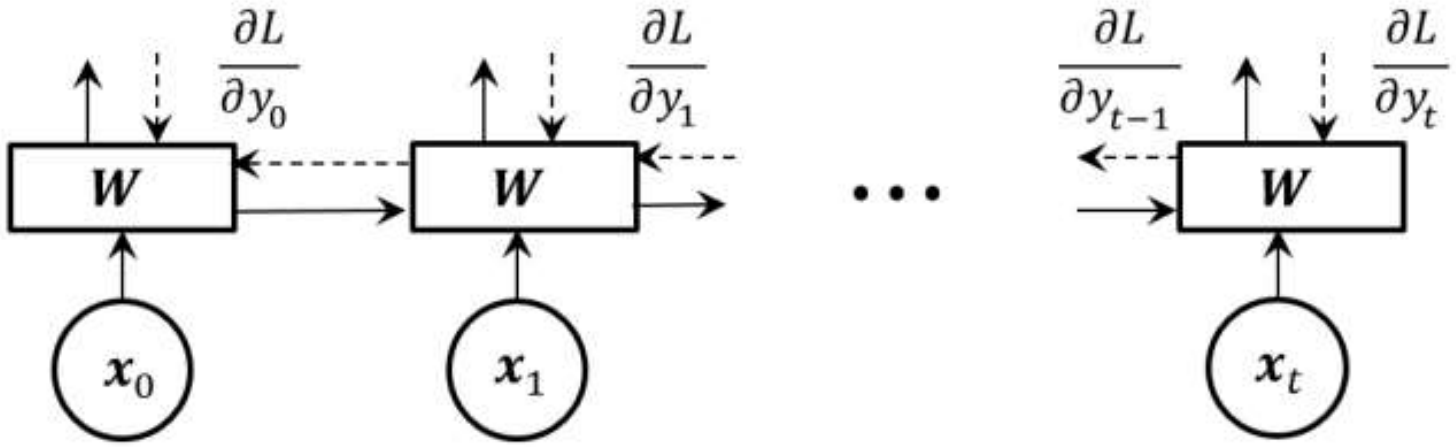
$$y_n = [h_n, g_n]\, w + b$$

Конкатенация

Источник: https://education.yandex.ru/handbook/ml/article/nejroseti-dlya-raboty-s-posledovatelnostyami

Backpropagation Through Time (BPTT)

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\sum_{k=0}^{t}\left(\prod_{i=k+1}^{t}\frac{\partial h_i}{\partial h_{i-1}}\right)\frac{\partial h_k}{\partial W}$$



Для предотвращения затухания/взрыва градиентов:

частные производные должны стремиться к 1: $\frac{\partial h_i}{\partial h_{i-1}} \to 1$

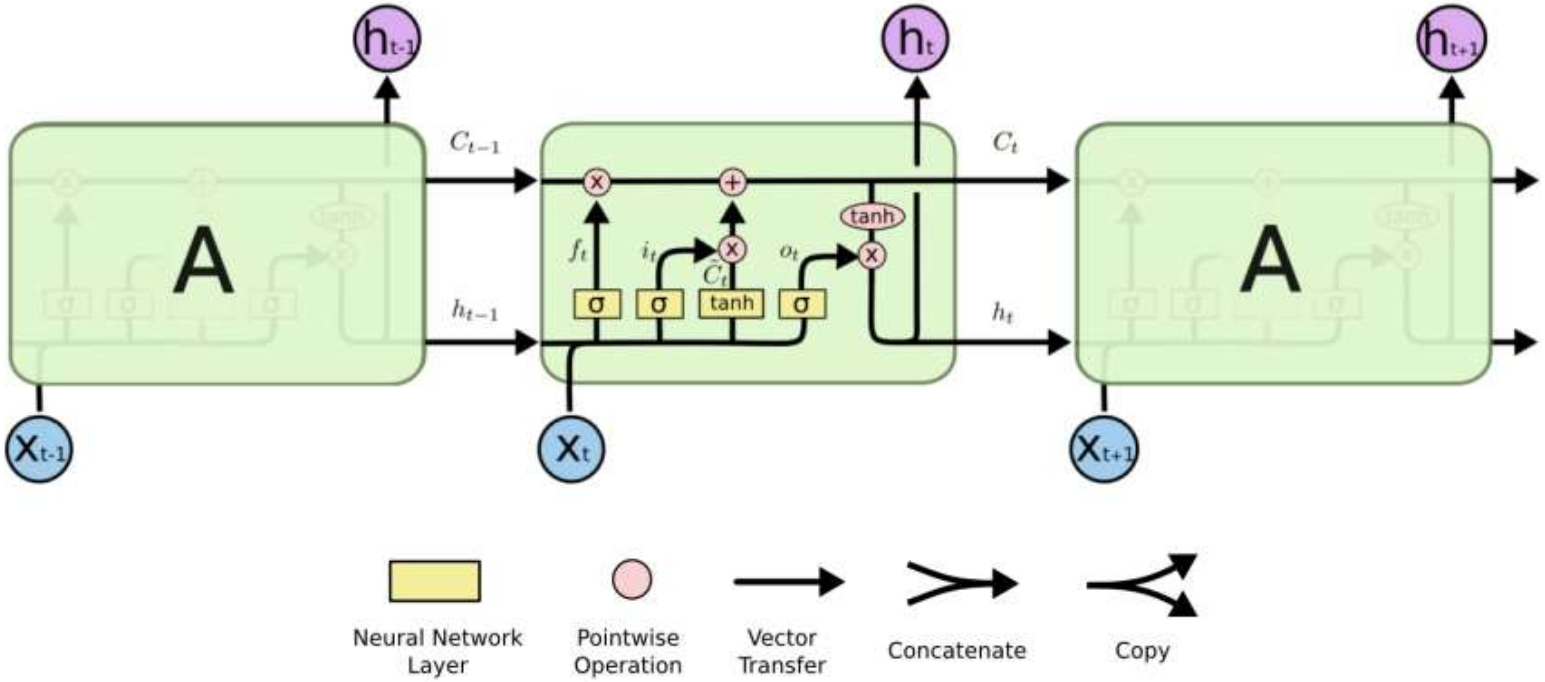
Достигается за счет выбора функций активации

Источник: http://www.machinelearning.ru/wiki/images/7/71/Voron-ML-DeepLearning-slides.pdf

Сети долгой кратковременной памяти
Long Short-term memory (LSTM)

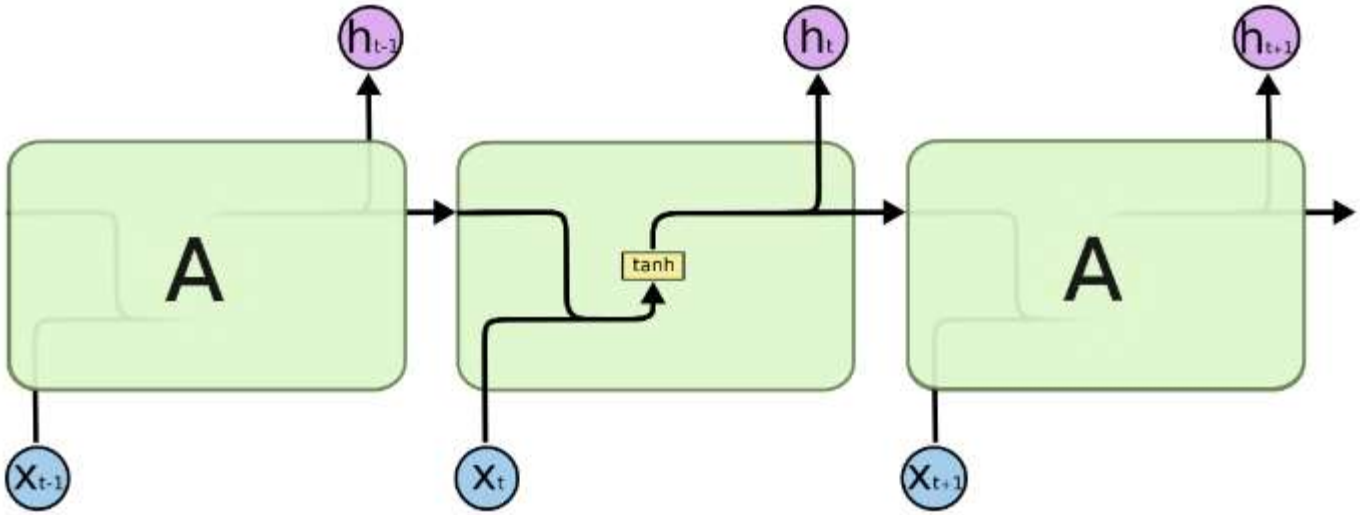
Мотивация LSTM: сеть должна сама долго помнить контекст, какой именно – сеть должна определить сама.

Вводится вектор C_t состояния сети в момент времени t. Разделяются векторы кратковременной и долговременной памяти.

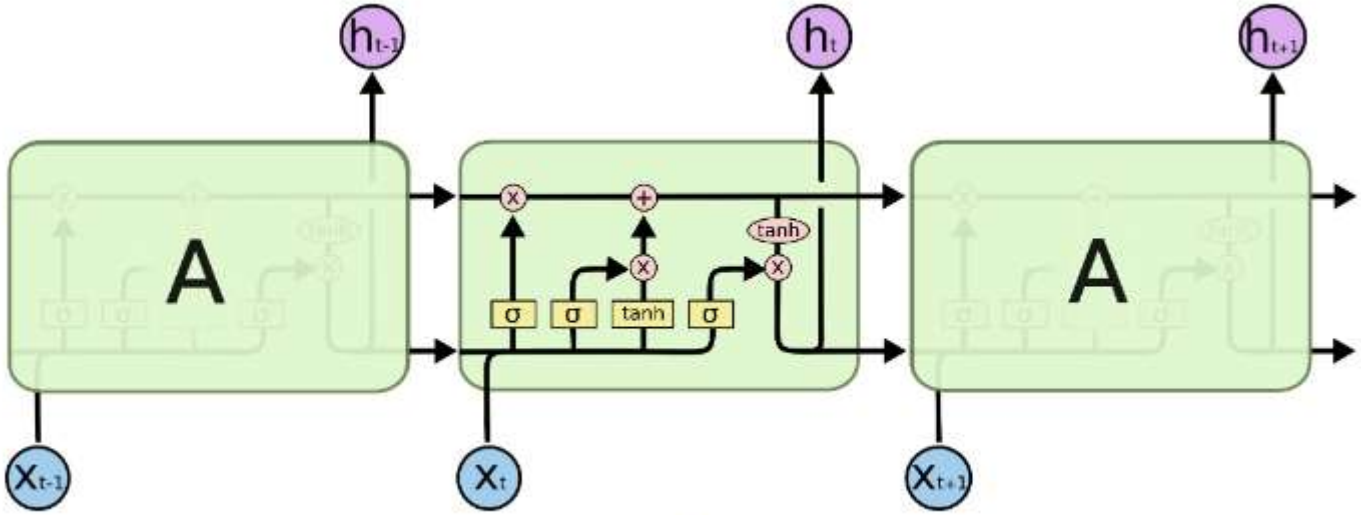


Источник: https://www.bioinf.jku.at/publications/older/2604.pdf

- LSTM частично решает проблему исчезновения и взрыва градиентов в процессе обучения
- Все RNN можно представить в виде цепочки повторяющихся блоков (линейный слой + функция активации). В LSTM повторяющийся блок имеет более сложную структуру, состоящую не из одного, а из четырех слоев. Появляется понятие состояния блока (cell state, c_n)
- Cell state используется в роли внутренней (закрытой) информации LSTM-блока. Скрытое состояние h_t передается наружу, как в следующий блок так и в следующий слой или выход сети.
- LSTM может добавлять или удалять определенную информацию из cell state с помощью специальных механизмов – gates



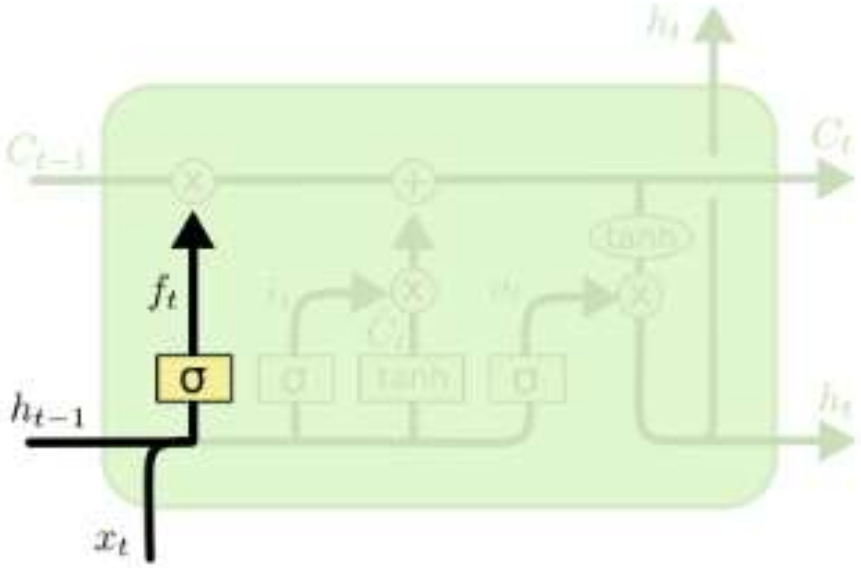
The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.

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

- Forget gate (вентиль забывания) – на основе предыдущего скрытого состояния h_{t-1} и нового входа x_t определить, какую долю информации из c_{n-1} (состояния предыдущего блока) стоит пропустить дальше, а какую забыть

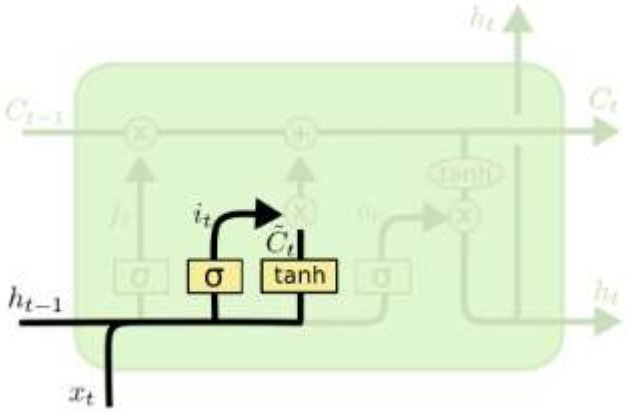


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- Следующий шаг – определить что вносится в cell state

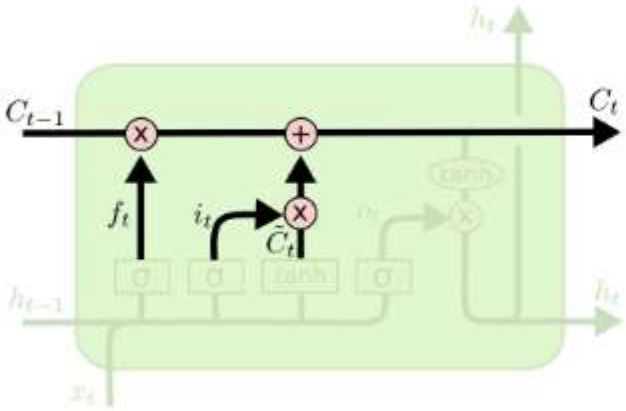
Источник: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Input gate (вентиль входного состояния) – решает, какие слагаемые надо «забыть», а какие внести



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

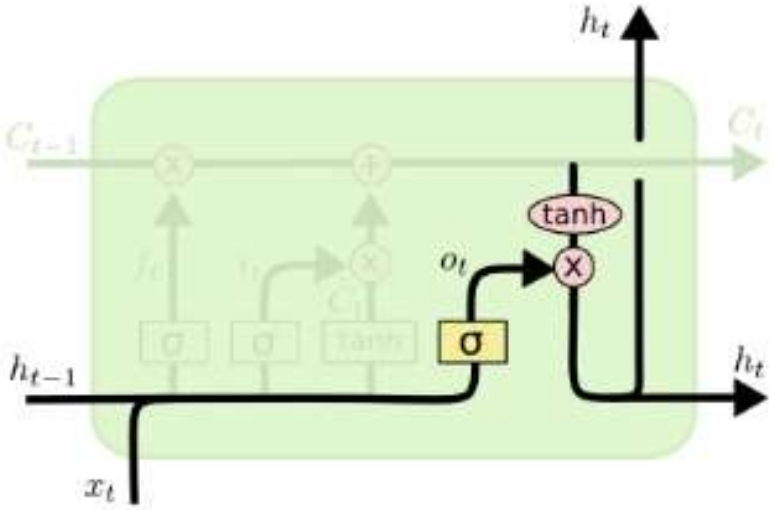
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



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

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

- Output gate (вентиль выходного состояния) – отвечает на вопрос о том, сколько информации из cell state следует отдавать на выход из LSTM-блока. Доля передаваемой информации o_t



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

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

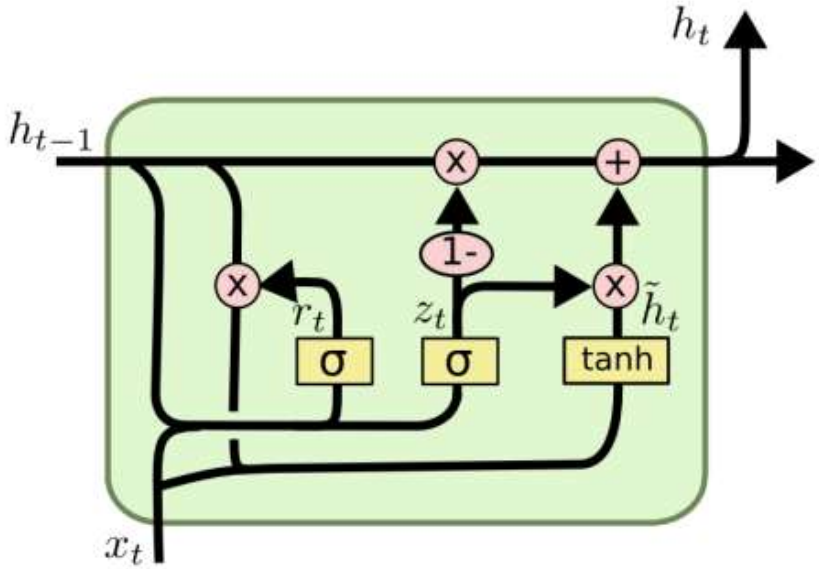
Источник: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit (GRU)

GRU является логическим упрощением LSTM с сохранением всех достоинств оригинальной архитектуры.

Используется только состояние h_t, вектор C_t не вводится.
Используется update gate – обобщение input gate и forget gate

Reset gate – решает, какую часть памяти нужно перенести дальше с прошлого шага



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t] + b_z\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t] + b_r\right)$$
$$\tilde{h}_t = \mathrm{th}\left(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h\right)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Источник: https://https://arxiv.org/pdf/1406.1078v3

Update gate – будем забывать только те значения, которые собираемся обновить (forget gate + input gate):

$$z_t = \sigma(h_{t-1}W_1^z + x_t W_2^z + b_z)$$

Reset gate – определяет, какую долю информации из h_{t-1} с прошлого шага надо «сбросить» и инициализировать заново:

$$r_t = \sigma(h_{t-1}W_1^r + x_t W_2^z + b_r)$$

Вычисляем потенциальное обновление для скрытого состояния:

$$\widetilde{h_n} = \tanh((r_t \odot h_{t-1})W_1^h + x_t W_2^h + b_h)$$

Решаем, что из старого забыть, а что из нового добавить:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \widetilde{h_t}$$

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

An Empirical Exploration of Recurrent Network Architectures

- LSTM и GRU придуманы эмпирически, неясно являются ли эти архитектуры оптимальными
- В ходе исследования, проведенного Rafal Jozefowicz, Ilya Sutskever (Google Inc.) и Wojciech Zaremba (NY University, Facebook) была проведена оценка более десяти тысяч различных архитектур RNN и LSTM. Были найдены некоторые архитектуры (ячейки), которые превосходят и LSTM и GRU на некоторых, но не на всех задачах

Arch.	5M-tst	10M-v	20M-v	20M-tst
Tanh	4.811	4.729	4.635	4.582 (97.7)
LSTM	4.699	4.511	4.437	4.399 (81.4)
LSTM-f	4.785	4.752	4.658	4.606 (100.8)
LSTM-i	4.755	4.558	4.480	4.444 (85.1)
LSTM-o	4.708	4.496	4.447	4.411 (82.3)
LSTM-b	4.698	4.437	4.423	**4.380 (79.83)**
GRU	4.684	4.554	4.559	4.519 (91.7)
MUT1	4.699	4.605	4.594	4.550 (94.6)
MUT2	4.707	4.539	4.538	4.503 (90.2)
MUT3	4.692	4.523	4.530	4.494 (89.47)

MUT1:

$$\begin{aligned} z &= \text{sigm}(W_{\text{xz}}x_t + b_z) \\ r &= \text{sigm}(W_{\text{xr}}x_t + W_{\text{hr}}h_t + b_r) \\ h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + \tanh(x_t) + b_{\text{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT2:

$$\begin{aligned} z &= \text{sigm}(W_{\text{xz}}x_t + W_{\text{hz}}h_t + b_z) \\ r &= \text{sigm}(x_t + W_{\text{hr}}h_t + b_r) \\ h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + W_{xh}x_t + b_{\text{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT3:

$$\begin{aligned} z &= \text{sigm}(W_{\text{xz}}x_t + W_{\text{hz}}\tanh(h_t) + b_z) \\ r &= \text{sigm}(W_{\text{xr}}x_t + W_{\text{hr}}h_t + b_r) \\ h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + W_{xh}x_t + b_{\text{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

Источник: https://proceedings.mlr.press/v37/jozefowicz15.pdf

Достоинства RNN архитектур:

- Способны учитывать контекст и последовательность данных
- Способны обрабатывать ввод любой длины
- Эффективны на задачах с временными зависимостями

Недостатки RNN архитектур:

- Вычислительная сложность, взрыв градиента
- Ограничения на долгосрочное запоминание, сложно получить информацию со всей последовательности