

# Глубокое обучение и вообще

Ульянкин Филипп

**Посиделка 11:** рекуррентные нейронные сетки

# Agenda

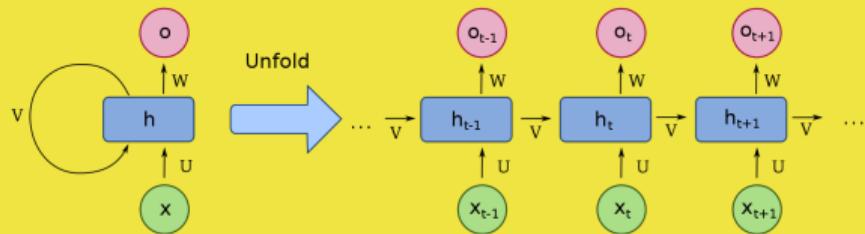
- Рекуррентные нейросети, простая RNN-ячейка
- Снова про взрывы и затухания градиентов
- LSTM и GRU ячейки
- Языковые модели

Довольно много слайдов и идей для объяснения я взял у Кати Лобачёвой

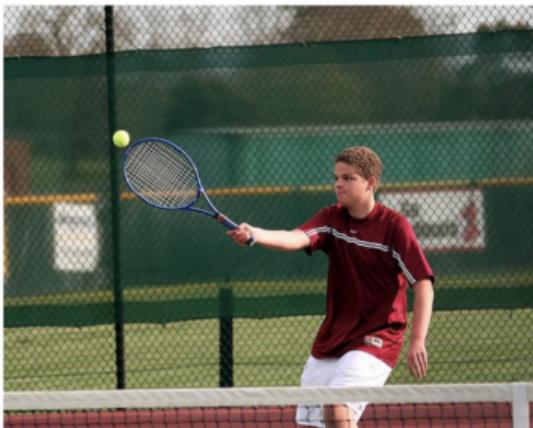
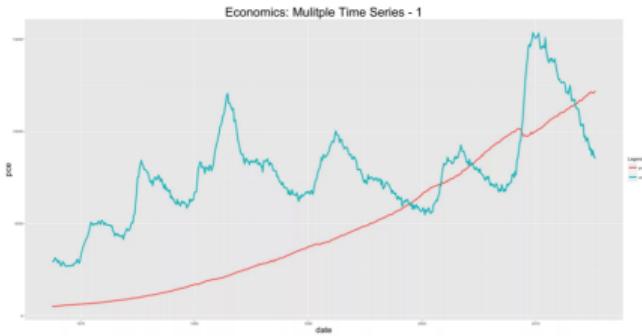
<https://www.coursera.org/learn/intro-to-deep-learning/home/week/5>

<https://github.com/tipt0p>

# Рекуррентные нейронные сети



# Последовательности всюду

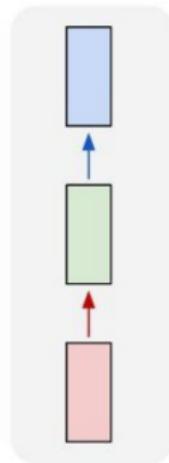


a man is playing tennis on a tennis court

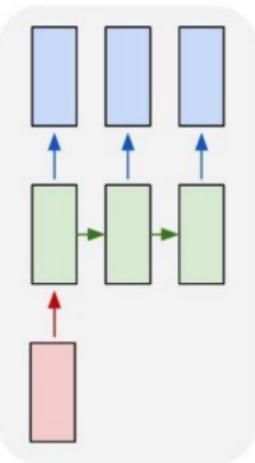
He dismissed the idea  
when the network is primed  
with a real sequence  
the samples mimic  
the writer's style

# Какой бывает работа с последовательностями

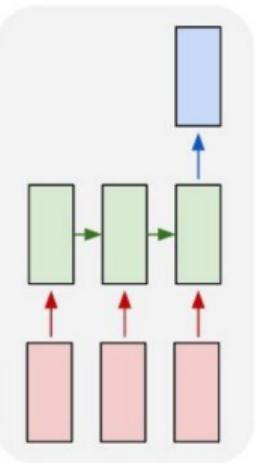
one to one



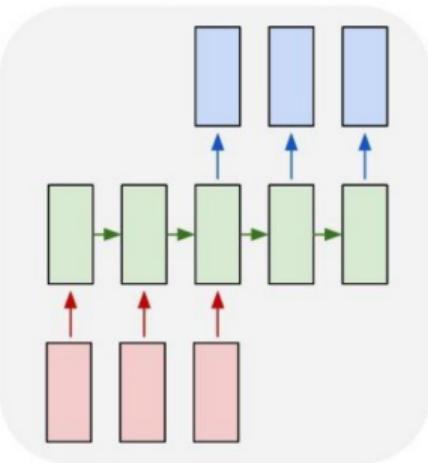
one to many



many to one



many to many



many to many

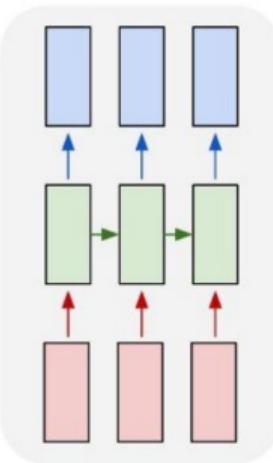


Image in  
Label out

Image in  
Words out

Words in  
Sentiment out

English in  
Portuguese out

Video In  
Labels out

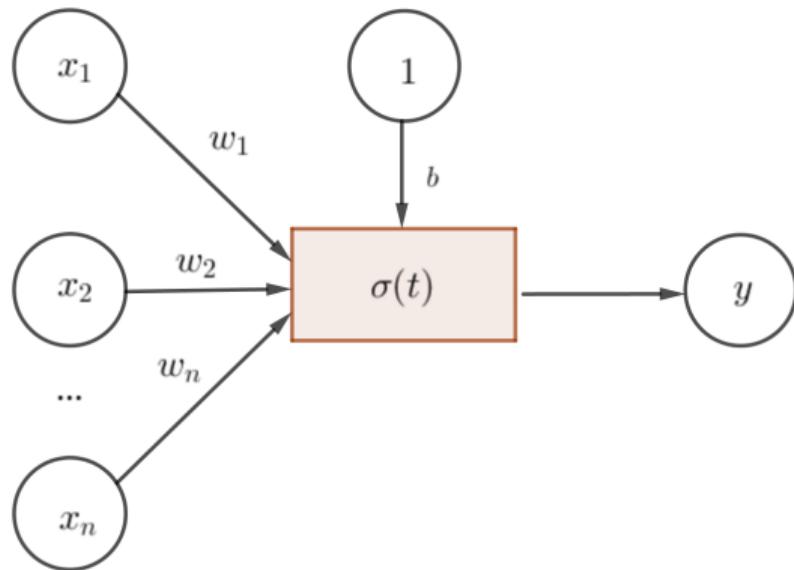
# От регрессии к нейрону

$$y_i = b + w \cdot x_i$$

⇓

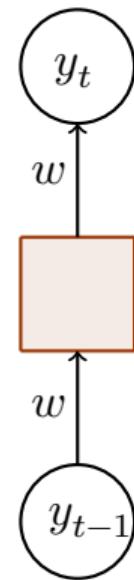
$$h_i = b + w \cdot x_i$$

$$y_i = f(h_i)$$



# От авторегрессии к нейрону

$$y_t = b + w \cdot y_{t-1}$$



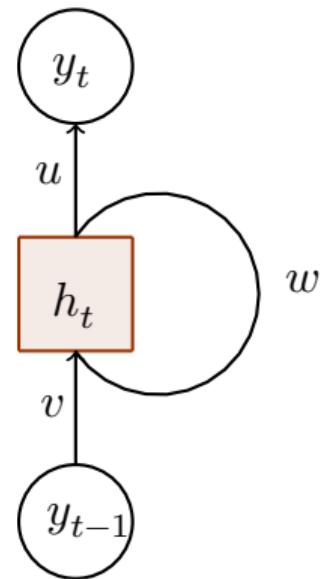
# От авторегрессии к нейрону

$$y_t = b + w \cdot y_{t-1}$$



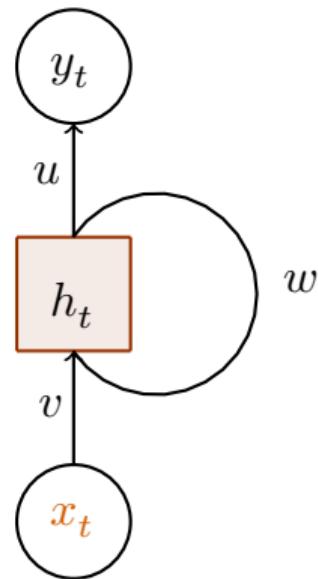
$$h_t = f_h(b_h + w \cdot h_{t-1} + v \cdot y_{t-1})$$

$$y_t = f_y(b_y + u \cdot h_t)$$

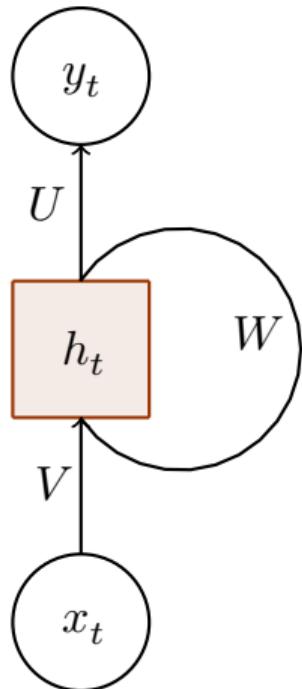


# От авторегрессии к нейрону

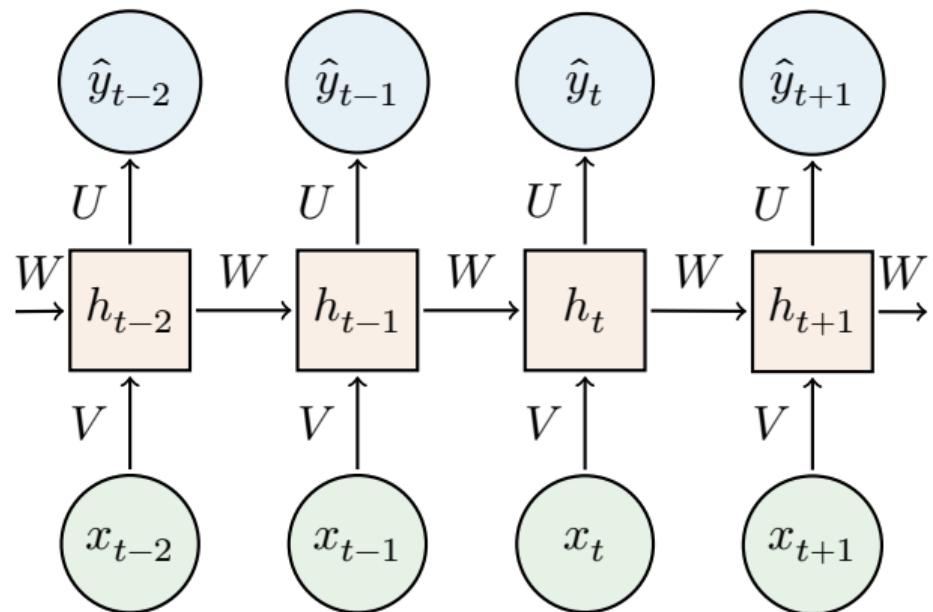
$$h_t = f_h(b_h + w \cdot h_{t-1} + v \cdot \textcolor{brown}{x}_t)$$
$$y_t = f_y(b_y + u \cdot h_t)$$



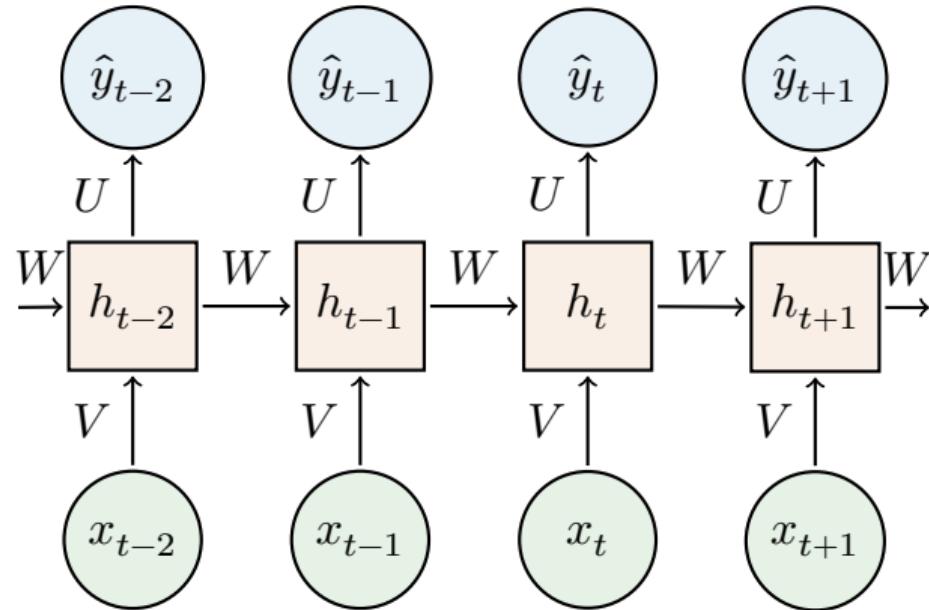
# Рекурентная сеть



Можно  
перерисовать



# Рекуррентная сеть



$$h_t = f_h(Vx_t + Wh_{t-1} + b_h) \quad \hat{y}_t = f_y(Uh_t + b_y)$$

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

# Рекурентная сеть

- На вход поступает последовательность (текст, видео, картинка, временной ряд)
- Нейрон просматривает её, на каждом шаге он взаимодействует сам собой и с последовательностью
- На каждом шаге нейрон использует одни и те же функции активации и веса
- Можно сказать, что у RNN есть память (скрытое состояние  $h_t$ ), в которой хранится информация о предыдущих элементах последовательности

# Backpropagation through time

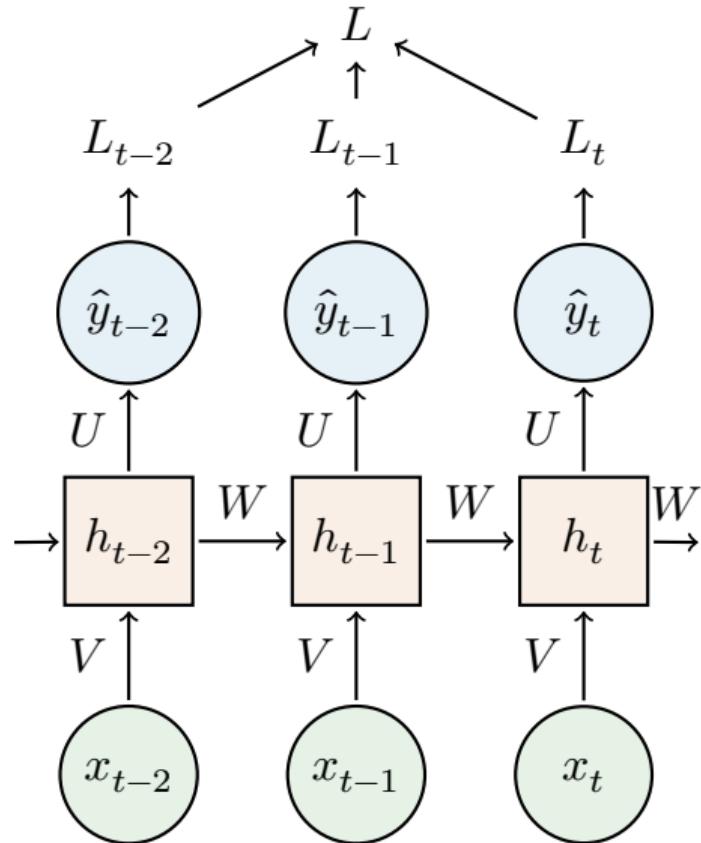
- Проблема состоит в том, что в работе сетки появилось новое измерение: время
- На каждом шаге сетка взвешивает свой предыдущий опыт и новую информацию, получается, что при обучении, мы должны брать производную назад во времени

# Backpropagation through time



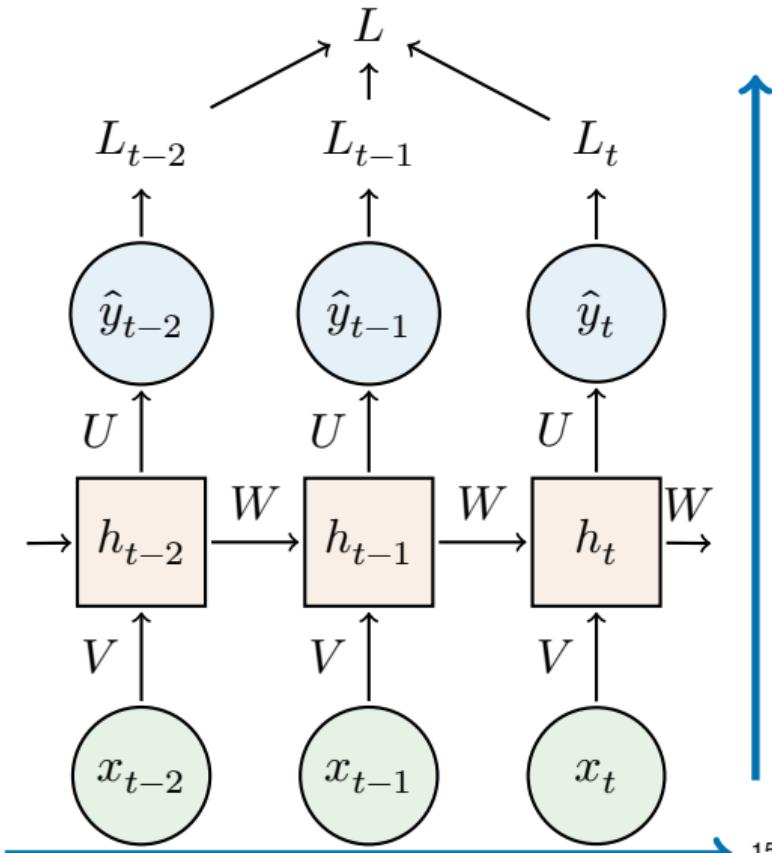
# Backpropagation through time

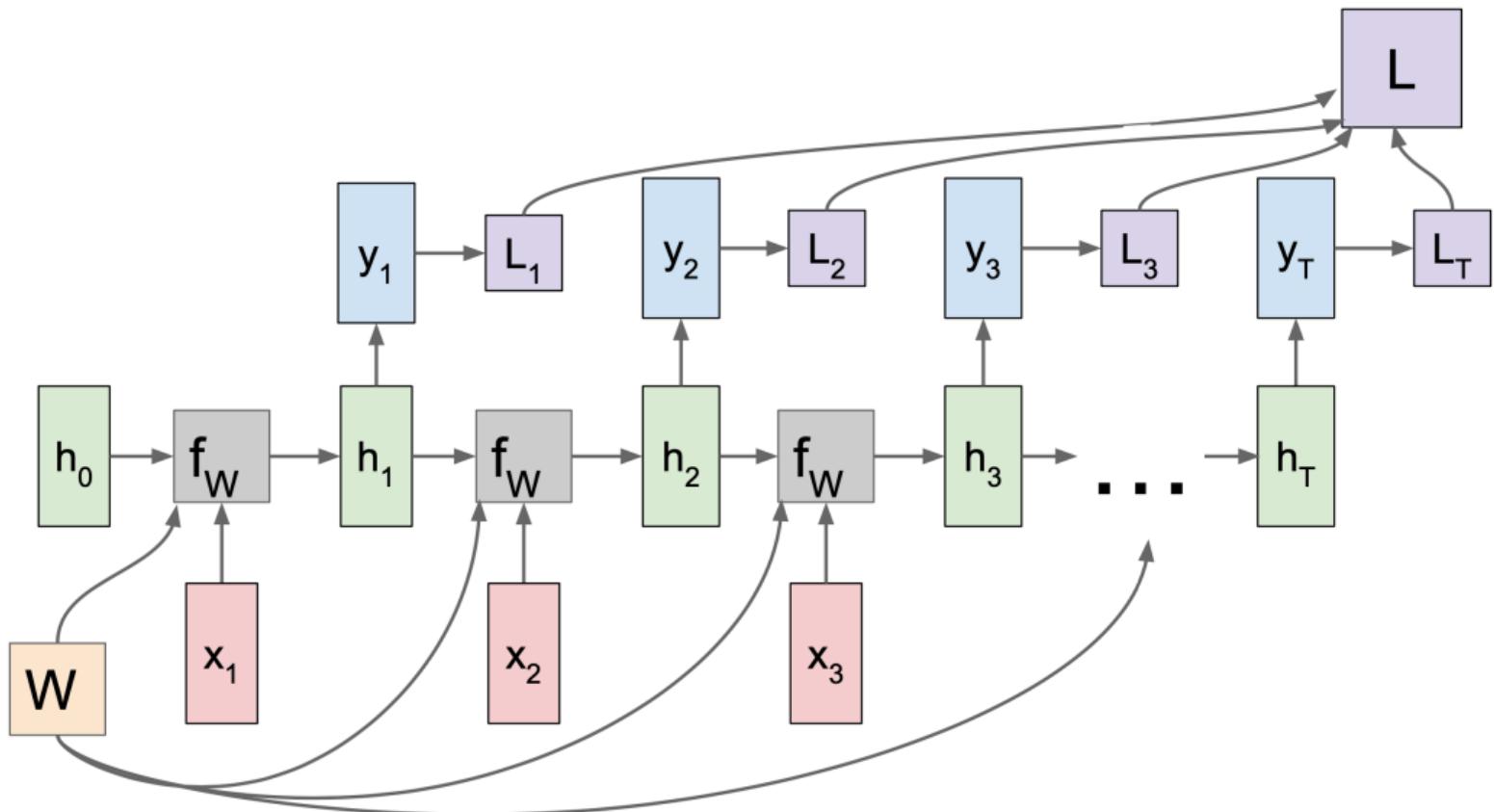
- $y_t$  — настоящее значение
- $\hat{y}_t$  — прогноз
- $L_t(y_t, \hat{y}_t)$  — функция потерь
- $L = \sum_t L_t(y_t, \hat{y}_t)$  — ошибка по всей последовательности



# Backpropagation through time

Forward pass:  
 $h_t, \hat{y}_t, L_t, L$





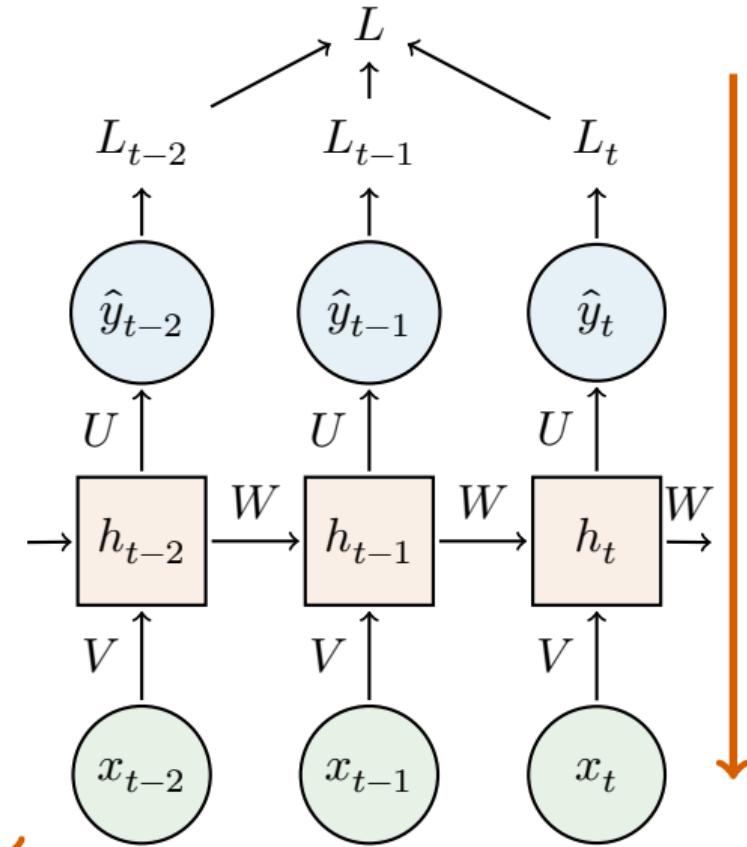
# Backpropagation through time

Forward pass:

$$h_t, \hat{y}_t, L_t, L$$

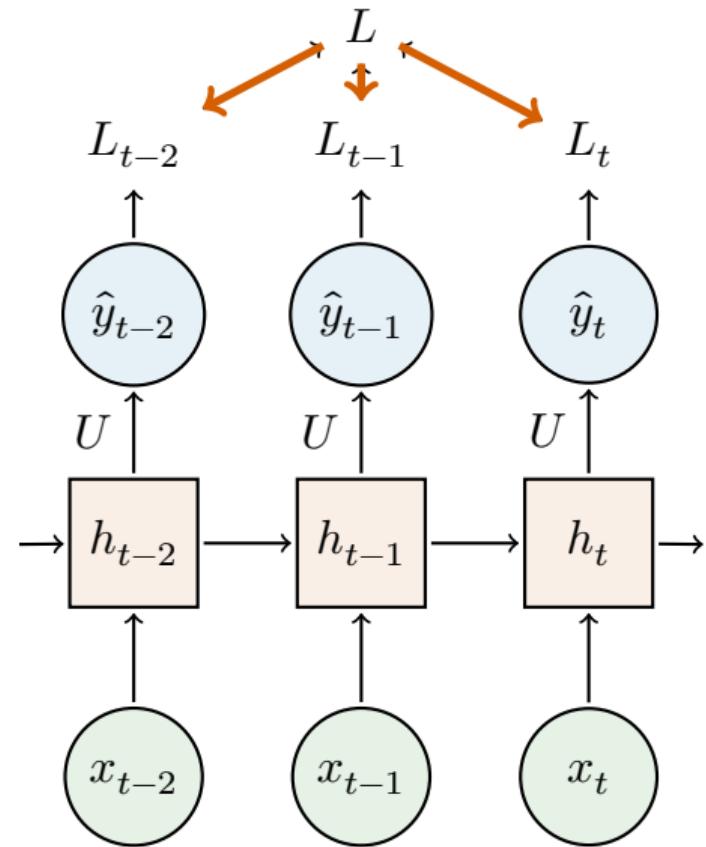
Backward pass:

$$\frac{\partial L}{\partial U}, \frac{\partial L}{\partial V}, \frac{\partial L}{\partial W}$$



# Backpropagation through time

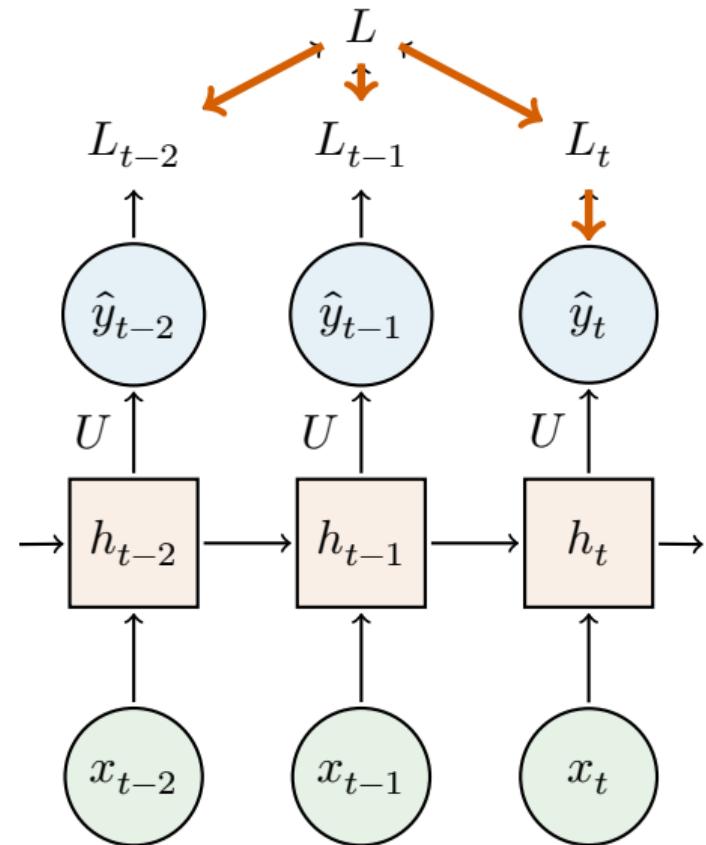
$$\frac{\partial L}{\partial U} = \sum_{i=0}^T \frac{\partial L_i}{\partial U}$$



# Backpropagation through time

$$\frac{\partial L}{\partial U} = \sum_{i=0}^T \frac{\partial L_i}{\partial U}$$

$$\frac{\partial L_t}{\partial U} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial U}$$



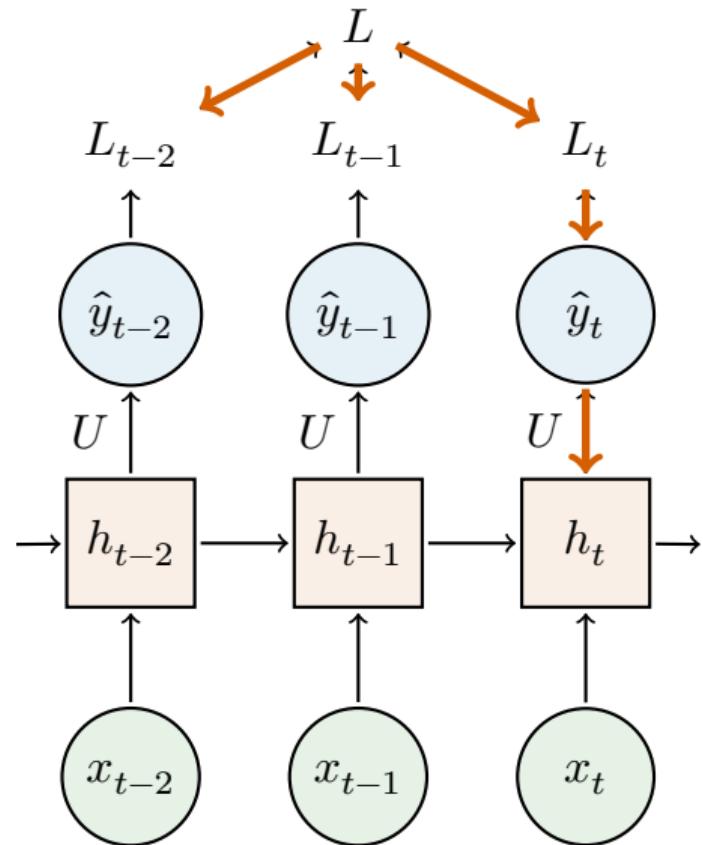
# Backpropagation through time

$$\frac{\partial L}{\partial U} = \sum_{i=0}^T \frac{\partial L_i}{\partial U}$$

$$\frac{\partial L_t}{\partial U} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial U}$$

$$h_t = f_h(b_h + W \cdot h_{t-1} + V \cdot x_t)$$
$$\hat{y}_t = f_y(b_y + U \cdot h_t)$$

Параметр  $U$  в одном месте!



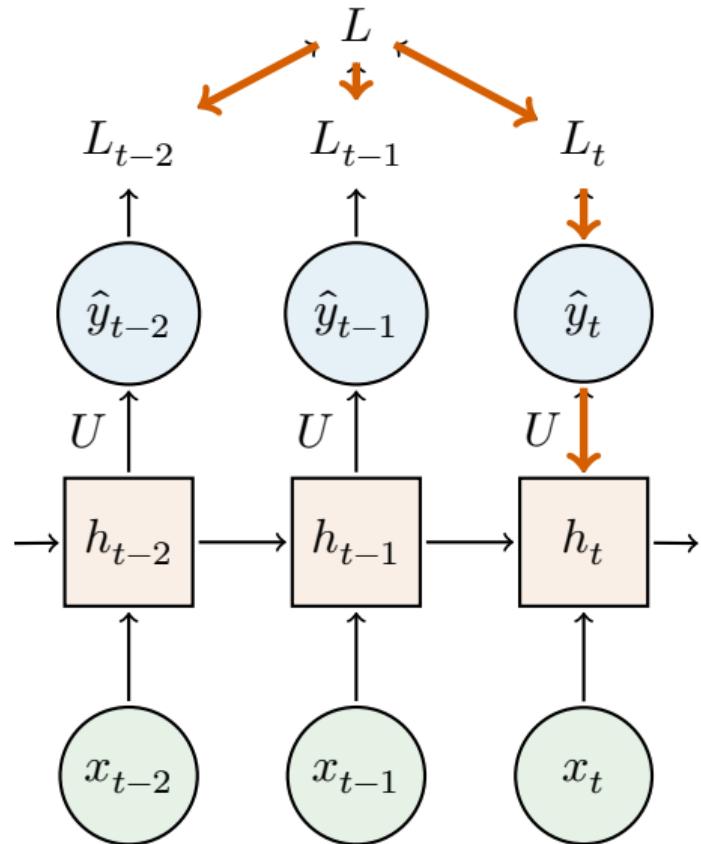
# Backpropagation through time

$$\frac{\partial L}{\partial U} = \sum_{i=0}^T \frac{\partial L_i}{\partial U}$$

$$\frac{\partial L_t}{\partial U} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot f'_y(\dots) \cdot h_t$$

$$h_t = f_h(b_h + W \cdot h_{t-1} + V \cdot x_t)$$

$$\hat{y}_t = f_y(b_y + U \cdot h_t)$$



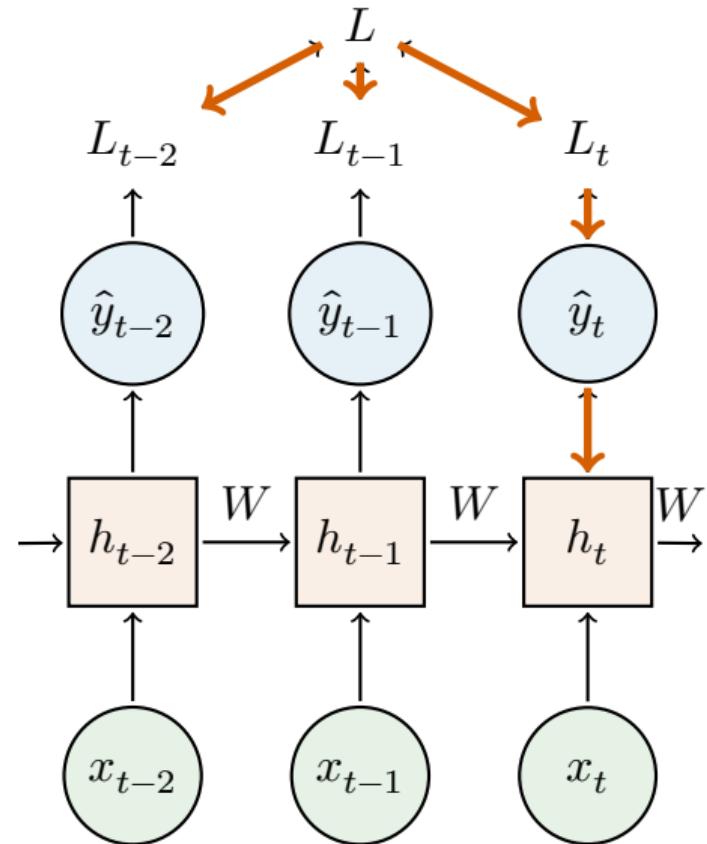
# Backpropagation through time

$$\frac{\partial L}{\partial W} = \sum_{i=0}^T \frac{\partial L_i}{\partial W}$$

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial W}$$

$$h_t = f_h(b_h + W \cdot h_{t-1} + V \cdot x_t)$$
$$\hat{y}_t = f_y(b_y + U \cdot h_t)$$

$W$  фигурирует в нескольких местах!



# Backpropagation through time

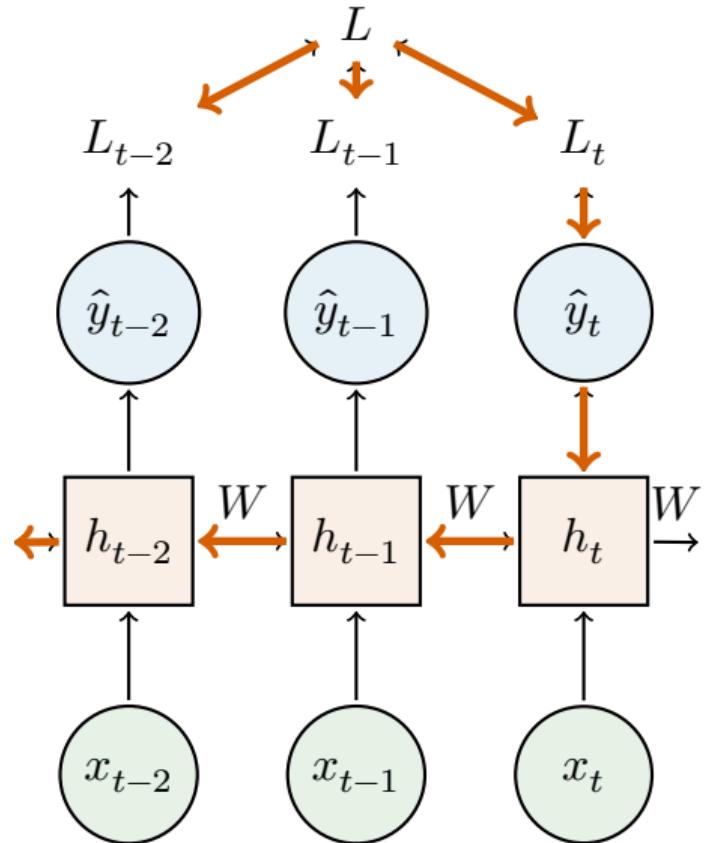
$$\frac{\partial L}{\partial W} = \sum_{i=0}^T \frac{\partial L_i}{\partial W}$$

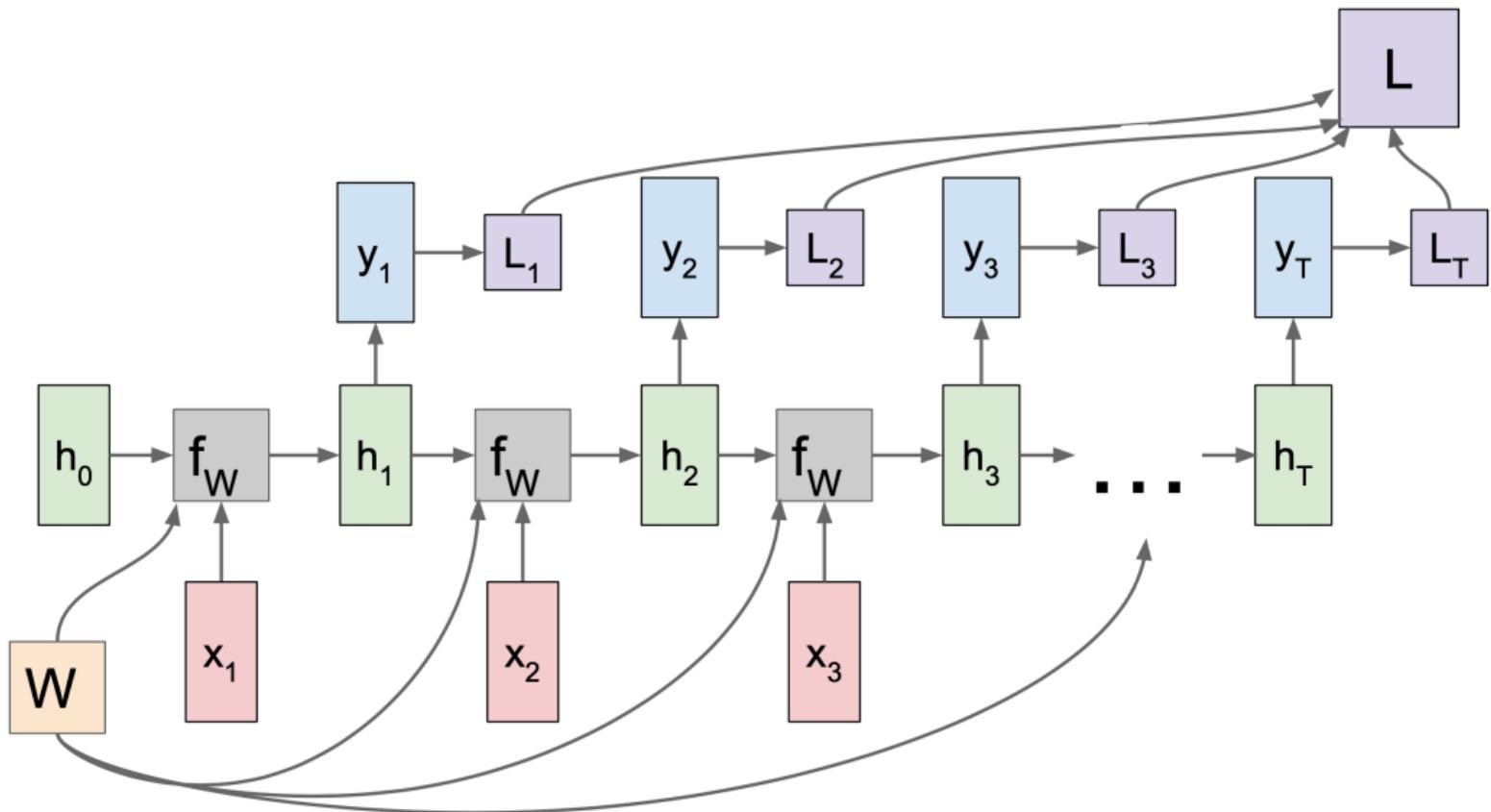
$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial W}$$

$$h_t = f_h(b_h + W \cdot h_{t-1} + V \cdot x_t)$$

$$\hat{y}_t = f_y(b_y + U \cdot h_t)$$

$$\frac{\partial h_t}{\partial W} - ???$$





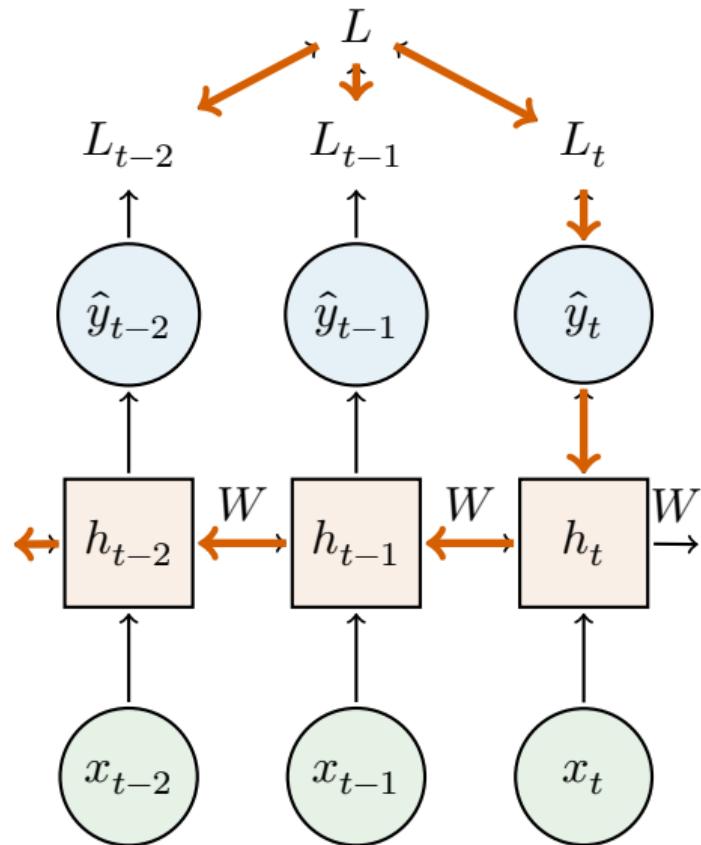
# Backpropagation through time

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial W}$$

$$h_t = f_h(b_h + W \cdot h_{t-1} + V \cdot x_t)$$

$$\hat{y}_t = f_y(b_y + U \cdot h_t)$$

$$\frac{\partial h_t}{\partial W} = \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \cdot \frac{\partial h_{t-1}}{\partial W} + \dots$$



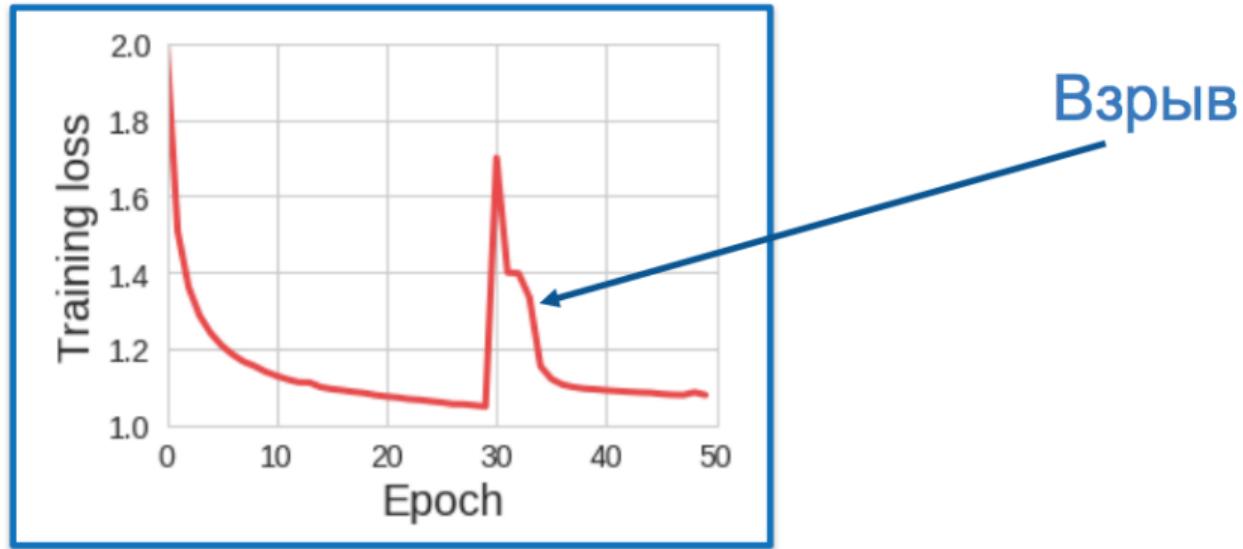
# Vanishing and Exploding

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

$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 < 1 \quad \Rightarrow \quad \text{Затухание градиентов}$

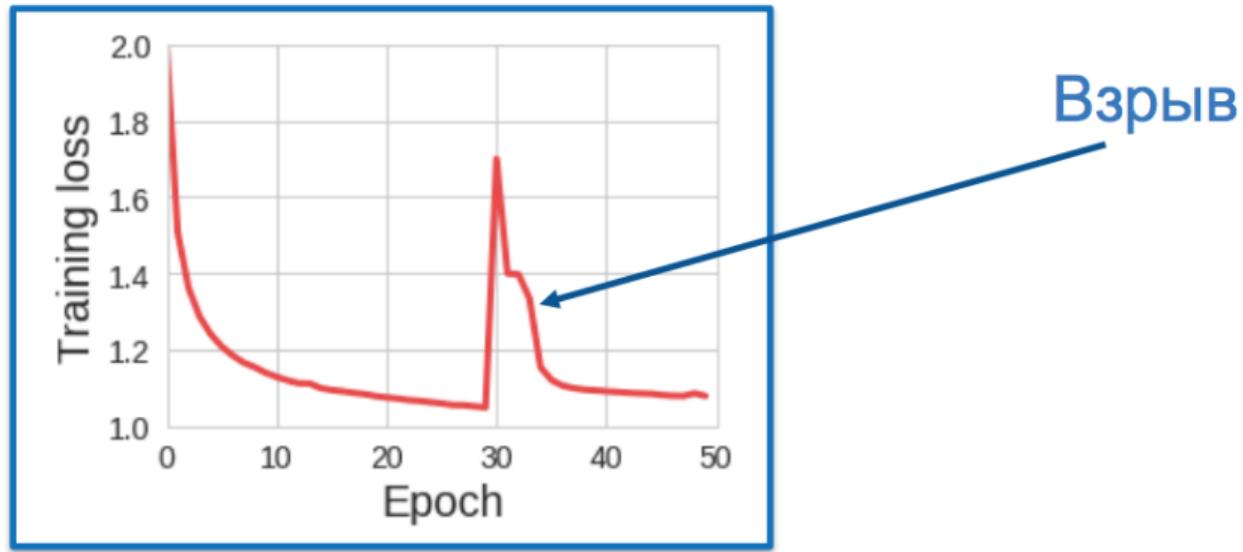
$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 > 1 \quad \Rightarrow \quad \text{Взрыв градиентов}$

# Как понять, что градиент взорвался?



Что делать со взрывом?

# Как понять, что градиент взорвался?



Что делать со взрывом?  $\Rightarrow$  ограничить градиент

# Ограничение градиентов (Gradient clipping)

Если

$$g = \frac{\partial L}{\partial w}, \quad \|g\|_2 > threshold$$

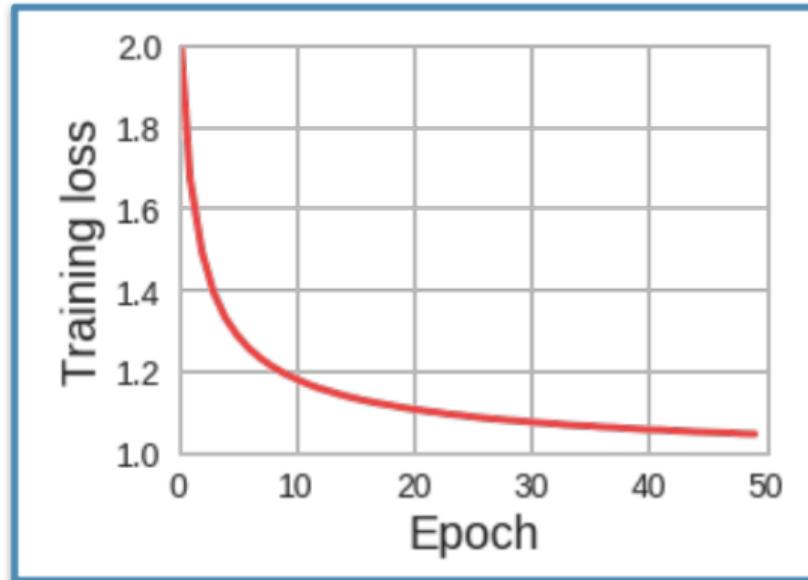
тогда

$$g = \frac{threshold}{\|g\|} \cdot g$$

Приём простой и работает :)

<https://arxiv.org/abs/1211.5063>

# Как понять, что градиент затухает?



Толи обучение сошлось, толи градиенты очень маленькие. Нельзя понять, что градиент затухает, не поправив это и не получив улучшение качества.

# Как предотвратить затухание градиентов?

- Аккуратная инициализация
- Skip connections
- Модели с гейтами: LSTM, GRU
- Регуляризация
- ...

## Изучим градиенты подробнее

$$h_t = f_h(b_h + W \cdot h_{t-1} + V \cdot x_t) = f_h(pr_t)$$

$$\hat{y}_t = f_y(b_y + U \cdot h_t)$$

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

## Изучим градиенты подробнее

$$\begin{aligned} h_t &= f_h(b_h + W \cdot h_{t-1} + V \cdot x_t) = f_h(pr_t) \\ \hat{y}_t &= f_y(b_y + U \cdot h_t) \end{aligned}$$

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

$$\frac{\partial h_i}{\partial h_{i-1}} = \frac{\partial h_i}{\partial inp_i} \cdot \frac{\partial inp_i}{\partial h_{i-1}} = diag(f'_h(pr_t)) \cdot ?$$

## Изучим градиенты подробнее

$$h_t = f_h(b_h + W \cdot h_{t-1} + V \cdot x_t) = f_h(pr_t)$$

$$\hat{y}_t = f_y(b_y + U \cdot h_t)$$

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

$$\frac{\partial h_i}{\partial h_{i-1}} = \frac{\partial h_i}{\partial inp_i} \cdot \frac{\partial inp_i}{\partial h_{i-1}} = diag(f'_h(pr_t)) \cdot \textcolor{brown}{W}$$

# Аккуратная инициализация (2015)

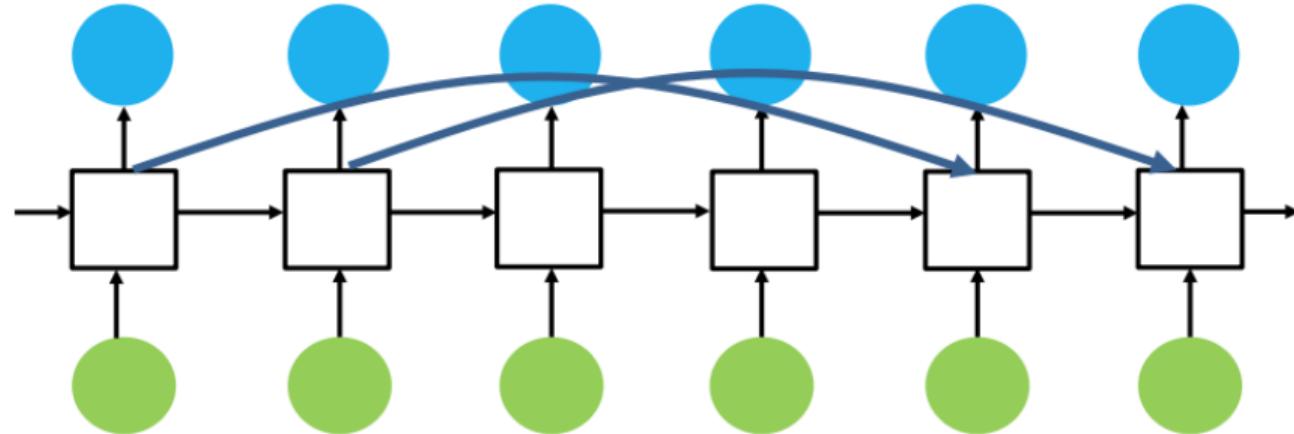
$$\frac{\partial L_t}{\partial W} \propto \sum_{k=0}^t \left( \prod_{i=k+1}^t \frac{\partial h_i}{\partial h_{i-1}} \right) \cdot \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_i}{\partial h_{i-1}} = \frac{\partial h_i}{\partial inp_i} \cdot \frac{\partial inp_i}{\partial h_{i-1}} = diag(f'_h(pr_t)) \cdot W$$

- Функция активации  $f_h(pr_t)$  не должна способствовать затуханию градиентов
- Если  $W$  ортогональная матрица, тогда  $W^T W = I$ ,  $W^{-1} = W^T$  и произведение  $\prod_i W_i$  не будет взрываться и затухать
- Инициализируем  $W$  ортогональной матрицей

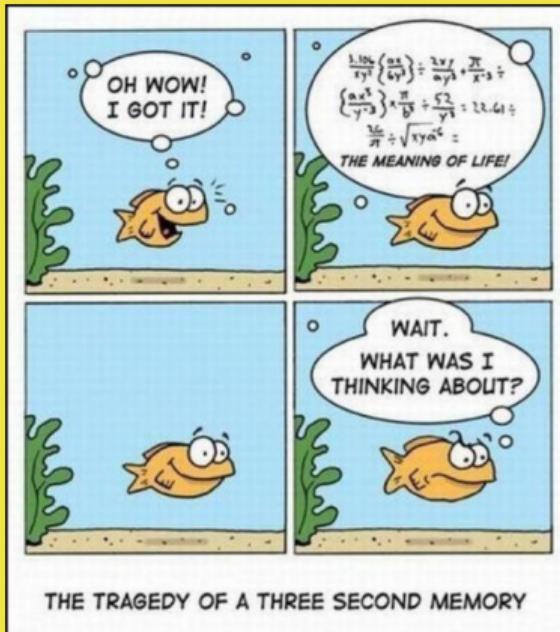
<https://arxiv.org/abs/1504.00941>

# Skip-connection



- Более короткие маршруты для градиентов позволяют избежать затухания, похожая идея использовалась в ResNet

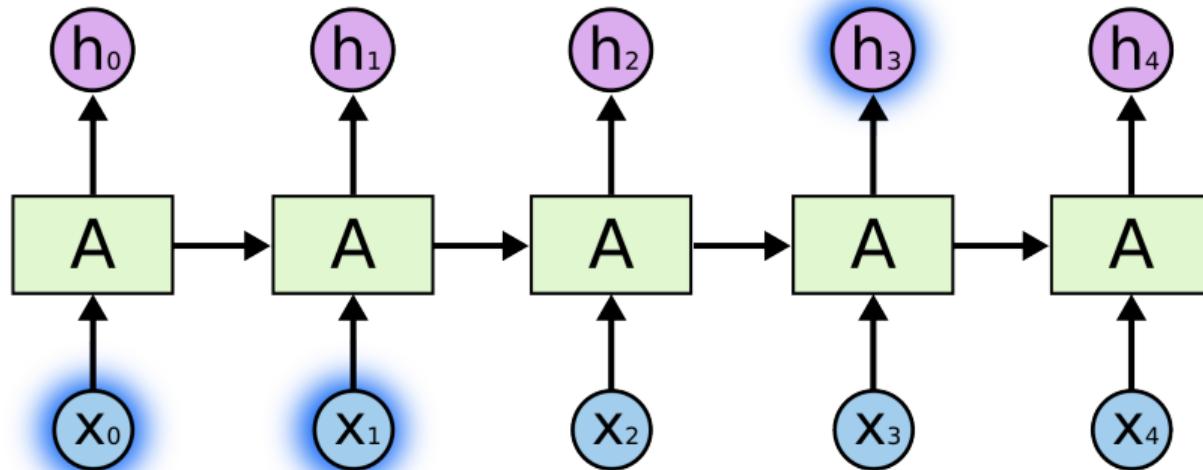
# LSTM (long short-term memory)



## Короткая память

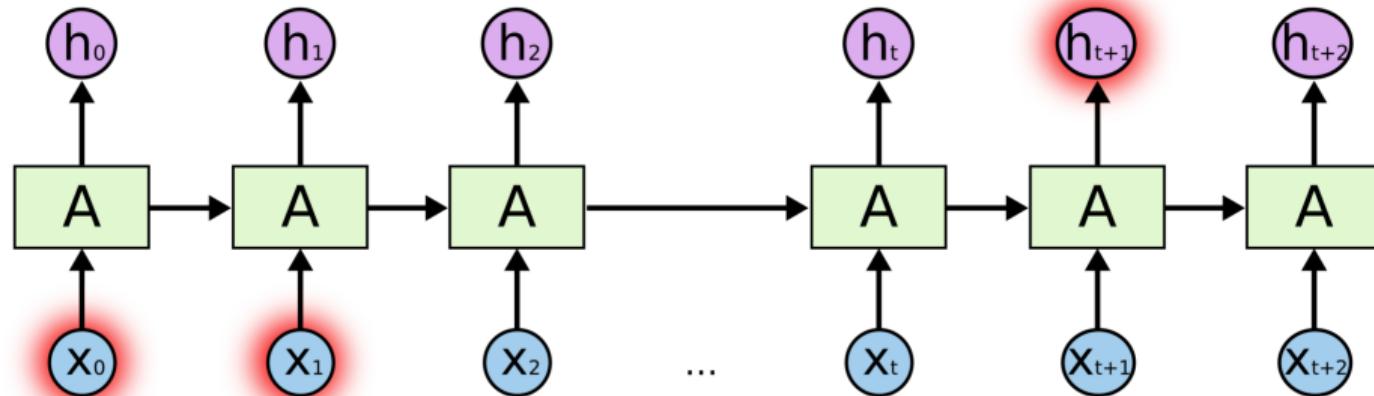
- Привлекательность RNN в том, что они потенциально умеют связывать предыдущую информацию с текущей
- При backpropagation текущие градиенты прорасыываются во времени назад
- Если градиент не взрывается, он постепенно затухает, получается что влияние текущего слоя не может проброситься во времени слишком далеко назад
- Влияние текущего слоя затухает экспоненциально по мере удаления и мешает обычным RNN находить в данных "далёкие" зависимости

# Короткая память



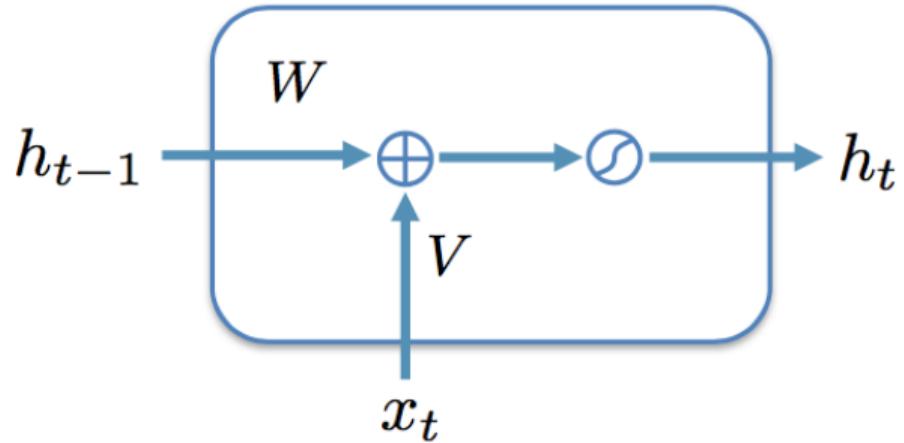
Облака плывут по небу

# Короткая память



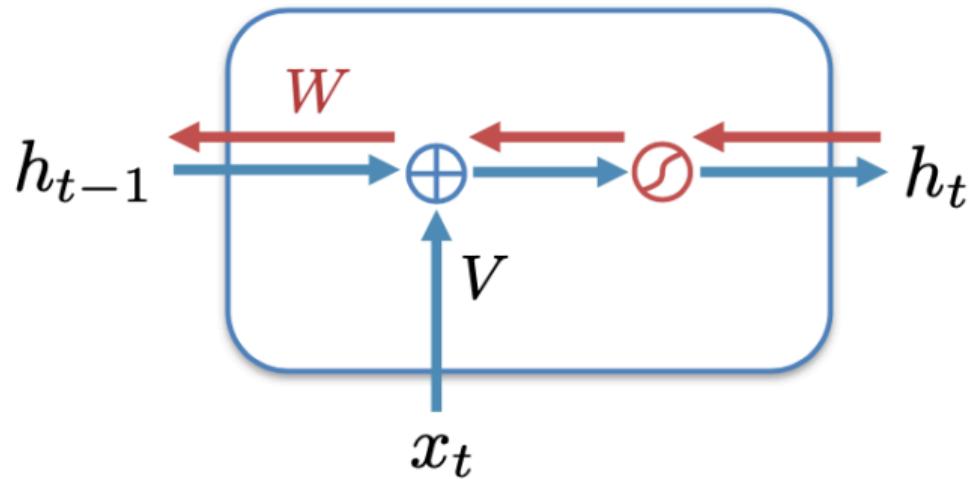
Я вырос во Франции... Я бегло говорю по-французски

# Простейшая RNN



$$h_t = \tilde{f}(Vx_t + Wh_{t-1} + b_h)$$

# Простейшая RNN



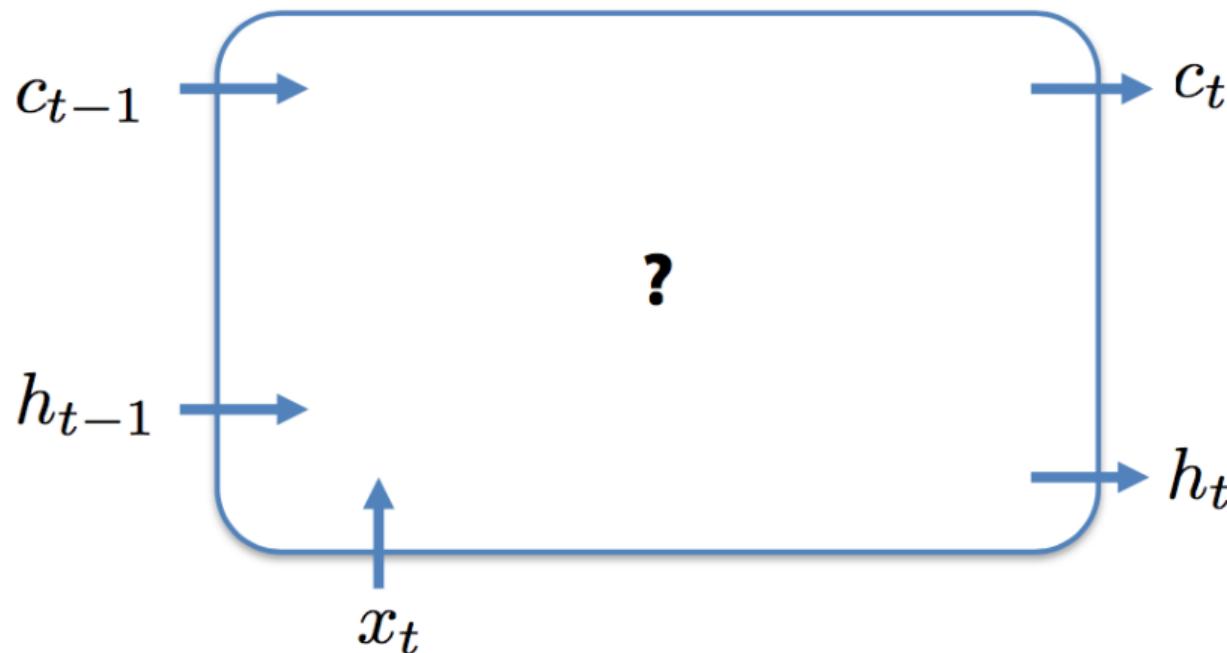
$$h_t = \tilde{f}(Vx_t + Wh_{t-1} + b_h)$$

## Долгая краткосрочная память (1997)

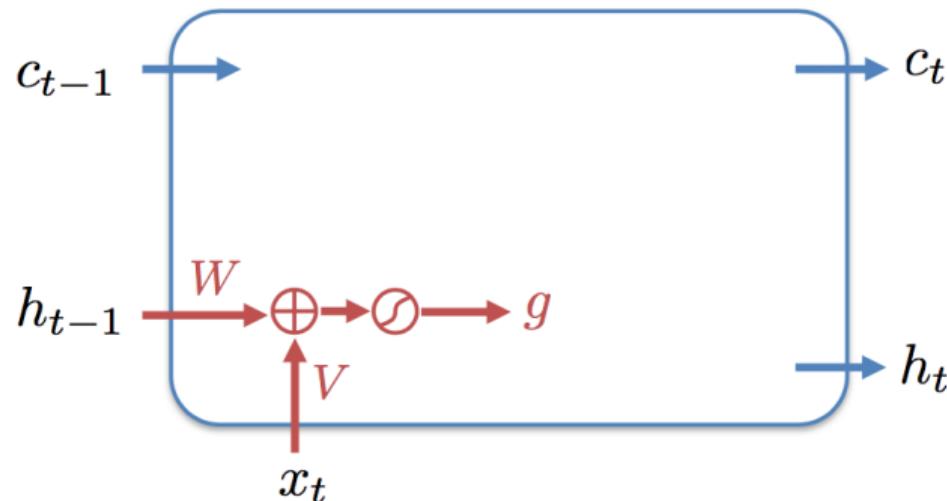
- В простой RNN-ячейке градиенты затухают из-за сложных преобразований
- Нужен короткий путь для градиентов
- Давайте разобьём память на долгосрочной и краткосрочную и получим LSTM

[https://www.researchgate.net/publication/13853244\\_Long\\_Short-term\\_Memory](https://www.researchgate.net/publication/13853244_Long_Short-term_Memory)

# LSTM

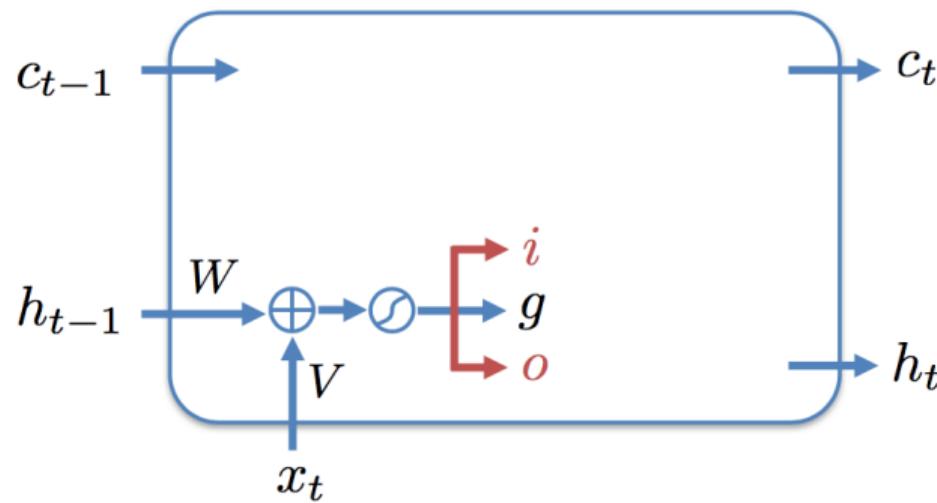


# LSTM



$$g_t = \tilde{f}(V_g x_t + W_g h_{t-1} + b_g)$$

# LSTM

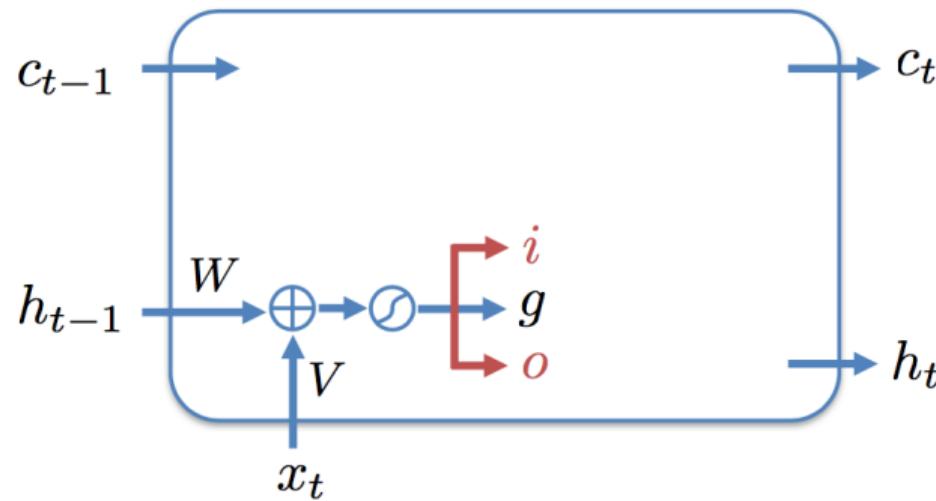


$$g_t = \tilde{f}(V_g x_t + W_g h_{t-1} + b_g)$$

$$i_t = \sigma(V_i x_t + W_i h_{t-1} + b_i)$$

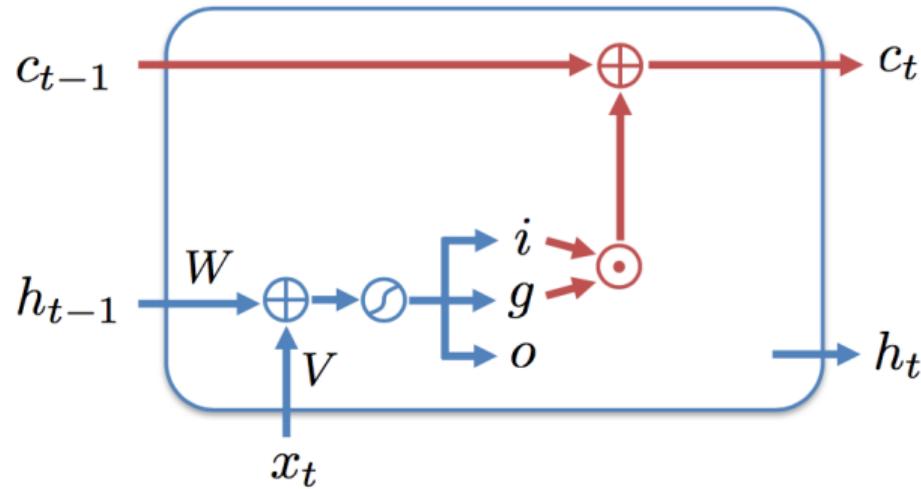
$$o_t = \sigma(V_o x_t + W_o h_{t-1} + b_o)$$

# LSTM



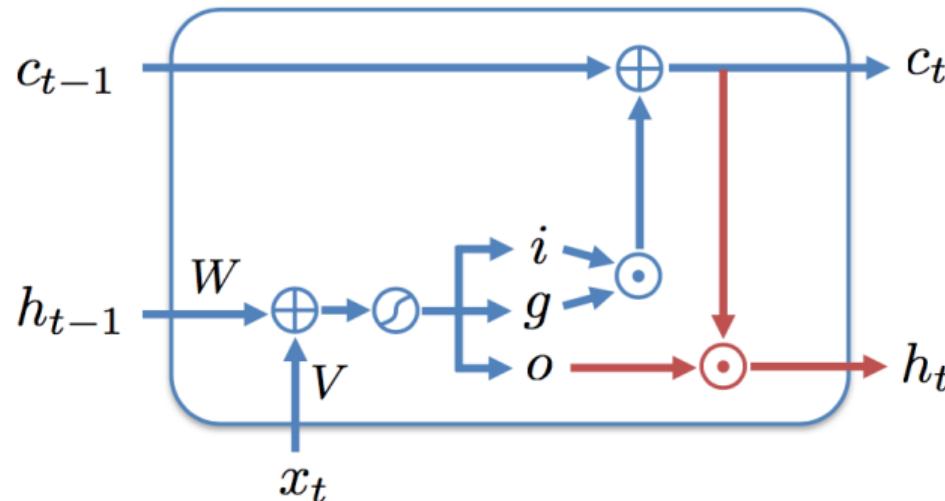
$$\begin{pmatrix} g_t \\ i_t \\ o_t \end{pmatrix} = \begin{pmatrix} \tilde{f} \\ \sigma \\ \sigma \end{pmatrix} (Vx_t + Wh_{t-1} + b)$$

# LSTM



$$\begin{pmatrix} g_t \\ i_t \\ o_t \end{pmatrix} = \begin{pmatrix} \tilde{f} \\ \sigma \\ \sigma \end{pmatrix} (Vx_t + Wh_{t-1} + b) \quad c_t = c_{t-1} + i_t \cdot g_t$$

# LSTM



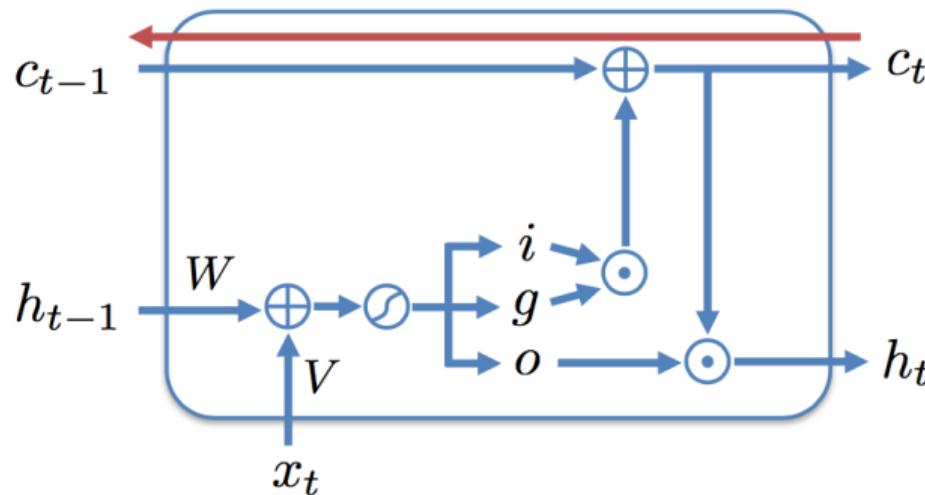
$$\begin{pmatrix} g_t \\ i_t \\ o_t \end{pmatrix} = \begin{pmatrix} \tilde{f} \\ \sigma \\ \sigma \end{pmatrix} (Vx_t + Wh_{t-1} + b)$$

$$c_t = c_{t-1} + i_t \cdot g_t$$
$$h_t = o_t \cdot \tilde{f}(c_t)$$

# LSTM

- Ключевой элемент LSTM это состояние ячейки (cell state,  $c_t$ ). Она проходит напрямую через цепочку, участвуя лишь в нескольких линейных операциях. Информация может легко течь по ней не подвергаясь преобразованиям.
- Фильтры (gates) контролируют поток информации и могут удалять лишнюю. Они состоят из слоя сигмоидальной нейронной сети и операции умножения. Сигмоида возвращает числа от 0 до 1, говоря какую долю информации нужно сохранить.
- В LSTM три таких фильтра контролируют состояние ячейки. Часть забывается, часть берётся из нового входа. Все эти манипуляции делают ячейки очень гибкими.

# LSTM



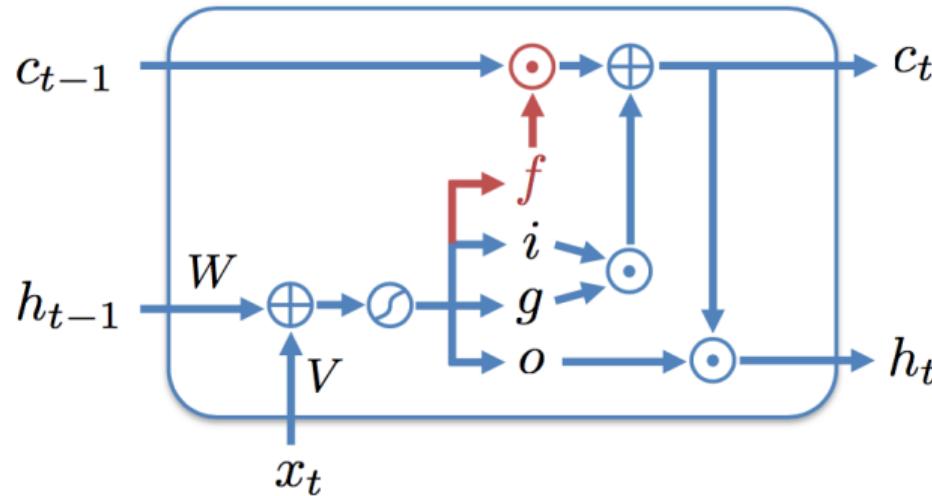
$$c_t = c_{t-1} + i_t \cdot g_t \quad \frac{\partial h_t}{\partial h_{t-1}} \rightarrow \frac{\partial c_t}{\partial c_{t-1}} = 1$$

Gradients do not vanish!

# LSTM

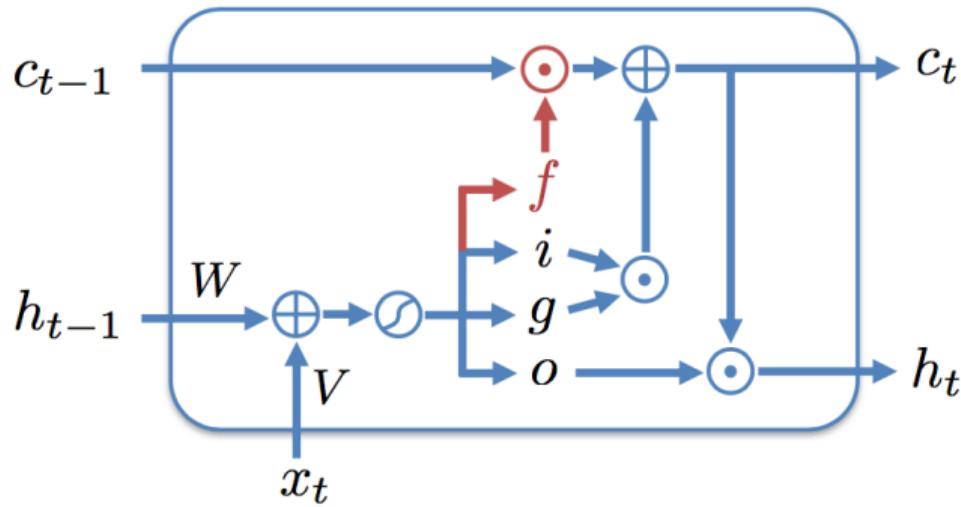
- В рекурсивном вычислении состояния ячейки нет никакой нелинейности. Обычно это называют **каруселью константной ошибки**
- Ошибка в LSTM пропагируется без изменений и скрытые состояния, если LSTM сама не решит их перезаписать, могут не меняться довольно долго
- Такое устройство ячейки решает проблему исчезающих градиентов, однако проблема взрывающихся градиентов остаётся  $\Rightarrow$  **ограничение градиентов**

# LSTM с забыванием (1999)



$$\begin{pmatrix} g_t \\ i_t \\ o_t \\ \textcolor{red}{f}_t \end{pmatrix} = \begin{pmatrix} \tilde{f} \\ \sigma \\ \sigma \\ \sigma \end{pmatrix} (Vx_t + Wh_{t-1} + b) \quad c_t = \textcolor{red}{f}_t \cdot c_{t-1} + i_t \cdot g_t$$
$$h_t = o_t \cdot \tilde{f}(c_t)$$

# LSTM с забыванием (1999)



$$f_t = \sigma(V_f x_t + W_f h_{t-1} + b_f) \quad c_t = f_t \cdot c_{t-1} + i_t \cdot g_t$$

$$\frac{\partial c_t}{\partial c_{t-1}} = f_t \quad \rightarrow \quad \text{High initial } b_f$$

## Распределение обязанностей

- $c$  — это вектор памяти, и мы хотим в него записать какую-то новую информацию, которая пришла из  $x$
- $g$  — это вектор, который представляет собой эту новую информацию
- $i$  — это гейт, который отвечает за то, какую долю информации мы берем
- $o$  — это гейт, который отвечает за то, какую долю информации мы выдаем
- $h$  — это вектор, который отвечает за то, что мы будем выдавать
- $f$  — это гейт, который отвечает за забывание

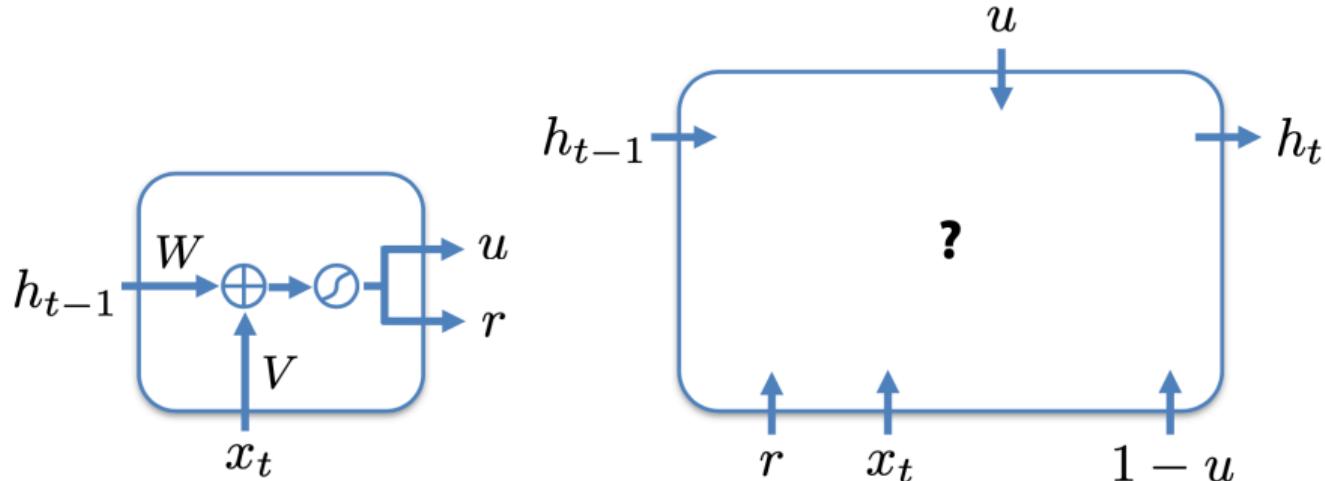
# GRU



## А вот бы весов поменьше бы

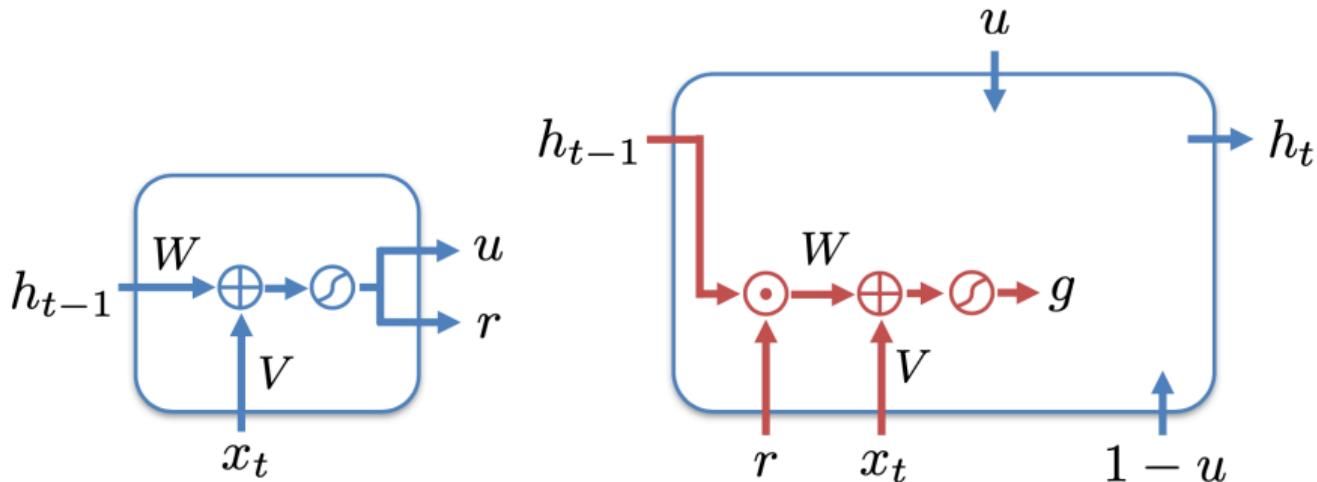
- В 2015 году придумали GRU-ячейку
- Придумали с желанием сохранить память в ячейке, но упростить LSTM
- Учится и применяется на 20-50% быстрее, результаты в большинстве случаев похожие

# GRU-ячейка



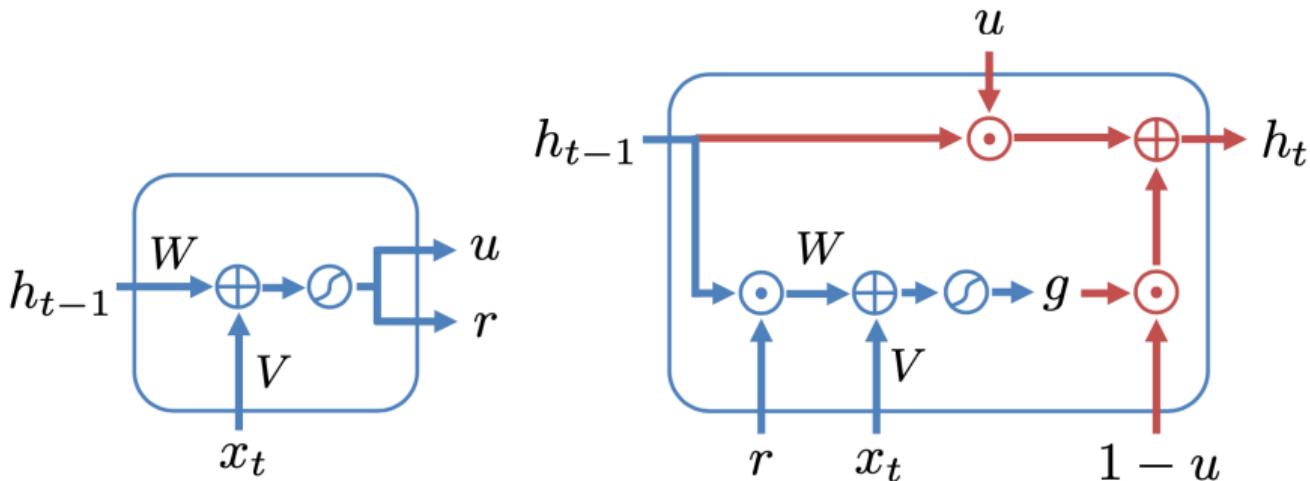
$$\begin{pmatrix} r_t \\ u_t \end{pmatrix} = \sigma(Vx_t + Wh_{t-1} + b)$$

# GRU-ячейка



$$\begin{pmatrix} \textcolor{red}{r}_t \\ u_t \end{pmatrix} = \sigma(Vx_t + Wh_{t-1} + b)$$
$$g_t = \tilde{f}(V_g x_t + W_g(h_{t-1} \cdot \textcolor{red}{r}_t) + b_g)$$

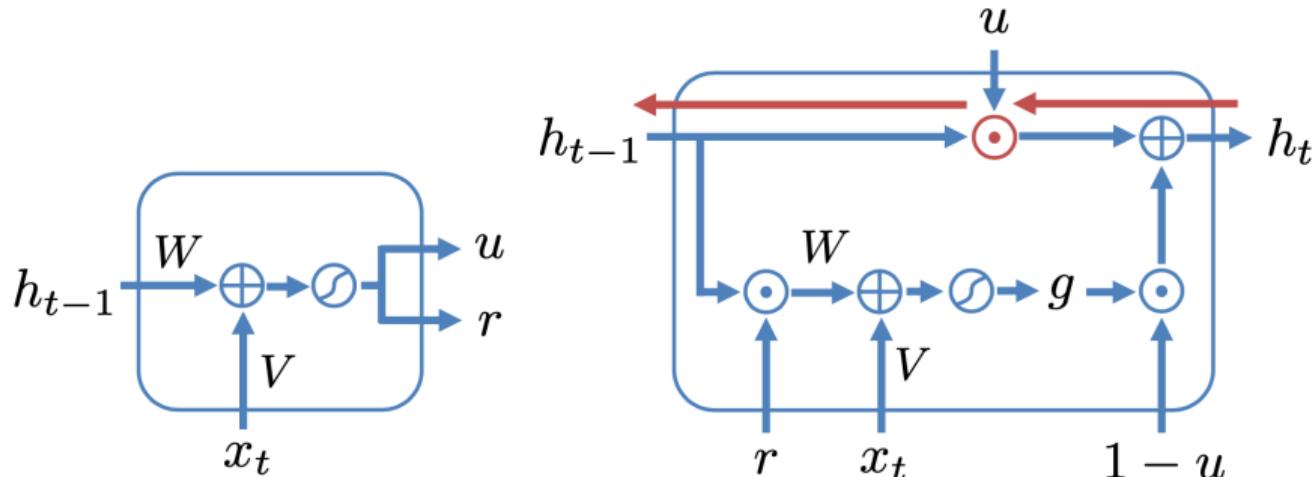
# GRU-ячейка



$$\begin{pmatrix} r_t \\ u_t \end{pmatrix} = \sigma(Vx_t + Wh_{t-1} + b)$$

$$g_t = \tilde{f}(V_g x_t + W_g(h_{t-1} \cdot r_t) + b_g)$$
$$h_t = (1 - u_t) \cdot g_t + u_t \cdot h_{t-1}$$

# GRU-ячейка



$$u_t = \sigma(V_u x_t + W_u h_{t-1} + b_u) \quad h_t = (1 - u_t) \cdot g_t + u_t \cdot h_{t-1}$$

$$\frac{\partial h_t}{\partial h_{t-1}} = u_h + (1 - u_h) \cdot \frac{\partial g_h}{\partial h_{h-1}} \quad \rightarrow \quad \text{High initial } b_u$$

# Помоги Google найти новую ячейку (2015)

- LSTM и GRU придуманы из головы, нет гарантии их оптимальности
- Google решил сделать перебор 10000 разных ячеек
- Нашли новые ячейки похожие на LSTM и GRU

| 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 | <b>4.380 (79.83)</b> |
| 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:

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

MUT2:

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

MUT3:

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

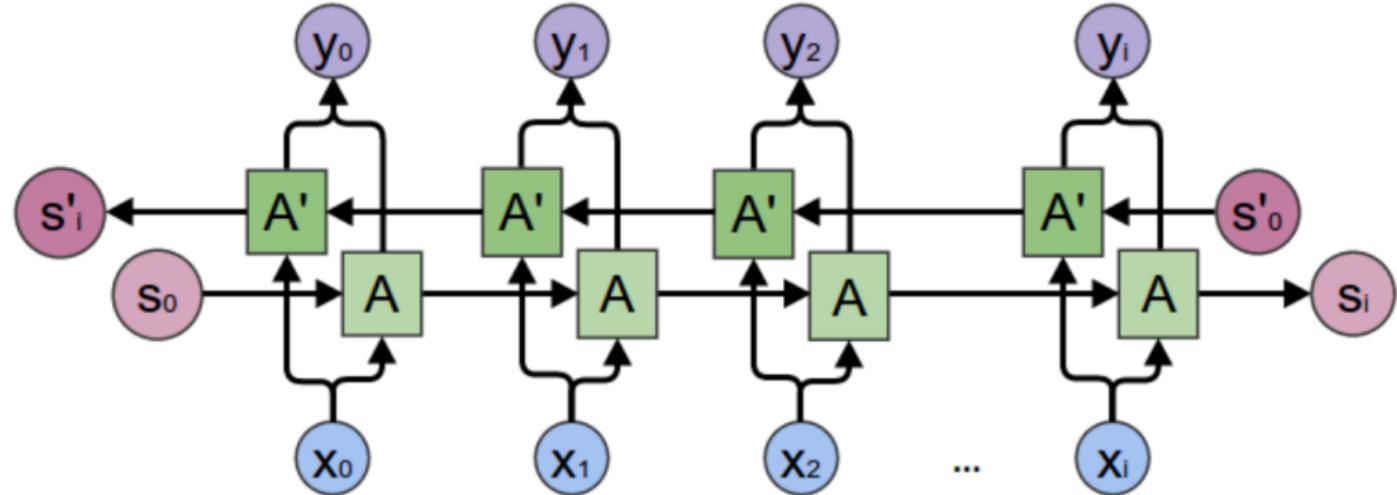
# Двунаправленные рекуррентные сети



# Двунаправленные рекуррентные сети

- RNN, LSTM и GRU ячейки можно стакать точно так же, как линейные слои.
- Все ячейки можно делать двунаправленными. Мы обсудили направление слева-направо, но можем запустить параллельную ветку, которая будет делать то же самое справа-налево.
- Это возможно не для всех задач, потому что иногда мы хотим генерировать (и у нас просто нет правого конца).

# Двунаправленные рекуррентные сети



## Двунаправленные рекуррентные сети

- Часто RNN к концу последовательности забывает о том, с чего всё начиналось, последние элементы последовательности всегда будут важнее первых
- Давайте один слой будет читать последовательность слева направо, а второй справа налево. Разумеется это можно делать только для тех последовательностей, которые даны нам целиком (предложения, аудио) и нельзя для тех, которые мы видим только в прошлое (валютный курс).
- Для каждого элемента получаем скрытое состояние, отражающее его контекст и слева и справа.

# Достоинства RNN

- Может обрабатывать ввод любой длины
- Вычисление для шага  $t$  может (теоретически) использовать информацию из всей последовательности
- Вычисления работают очень долго, градиенты могут взрываться
- Получить информацию со всей последовательности очень сложно

# Языковые модели



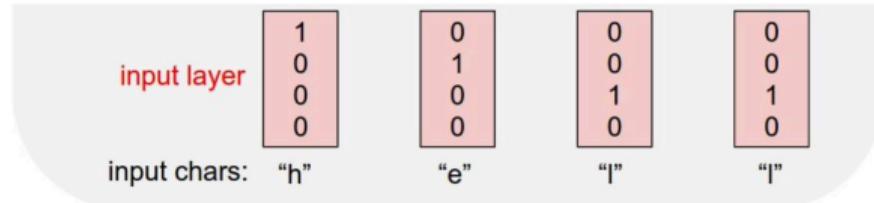
## Языковые модели

- У нас есть неразмеченный набор текстов
- Мы хотим научиться генерировать текст, т.е. хотим построить генеративную модель
- Чтобы нейросеть понимала, где начало, а где конец -- добавим специальные токены `<BOS>` (beginning of sequence) и `<EOS>` (ending of sequence)
- Будем учить модель выдавать на каждом шаге вероятность следующего токена

$$p(x_t | x_1, \dots, x_{t-1})$$

# Языковая модель

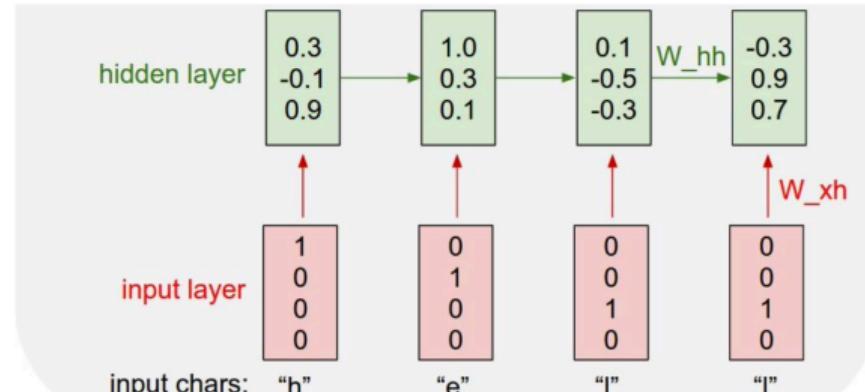
- Буквенная языковая модель (character-level Language Model)
- Словарь: [h,e,l,o]
- Пытаемся выучить слово "hello"



# Языковая модель

- Буквенная языковая модель (character-level Language Model)
- Словарь: [h,e,l,o]
- Пытаемся выучить слово "hello"

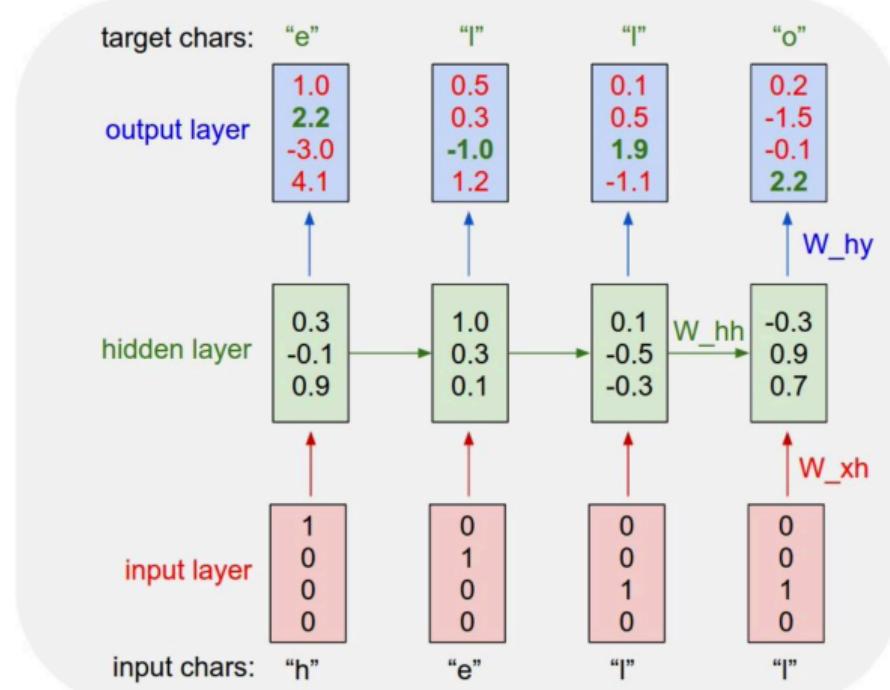
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



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

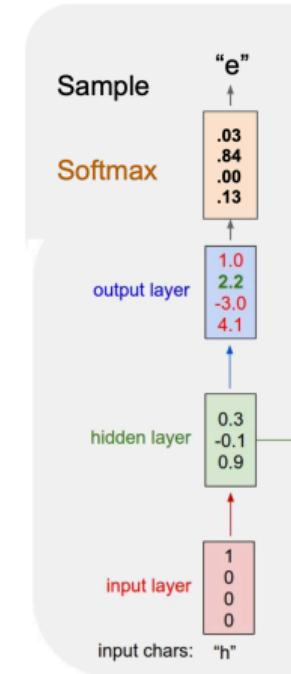
# Языковая модель

- Буквенная языковая модель (character-level Language Model)
- Словарь: [h,e,l,o]
- Пытаемся выучить слово "hello"



# Языковая модель

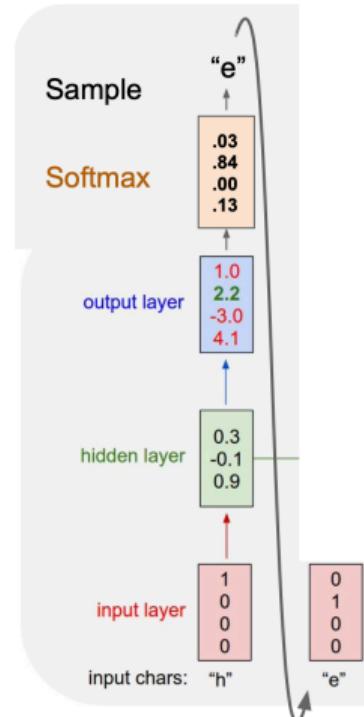
- Буквенная языковая модель (character-level Language Model)
- Словарь: [h,e,l,o]
- В тестовом режиме предсказываем следующие буквы



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

# Языковая модель

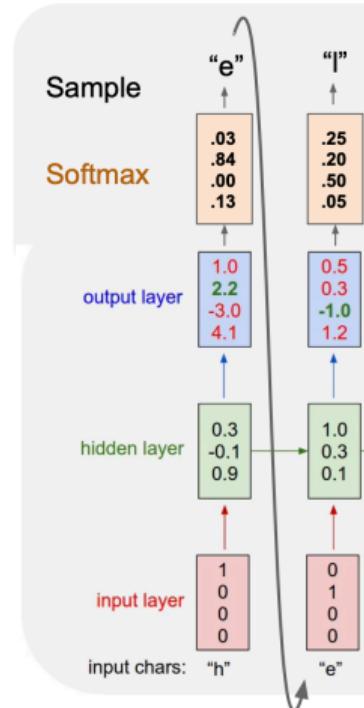
- Буквенная языковая модель (character-level Language Model)
- Словарь: [h,e,l,o]
- В тестовом режиме предсказываем следующие буквы



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

# Языковая модель

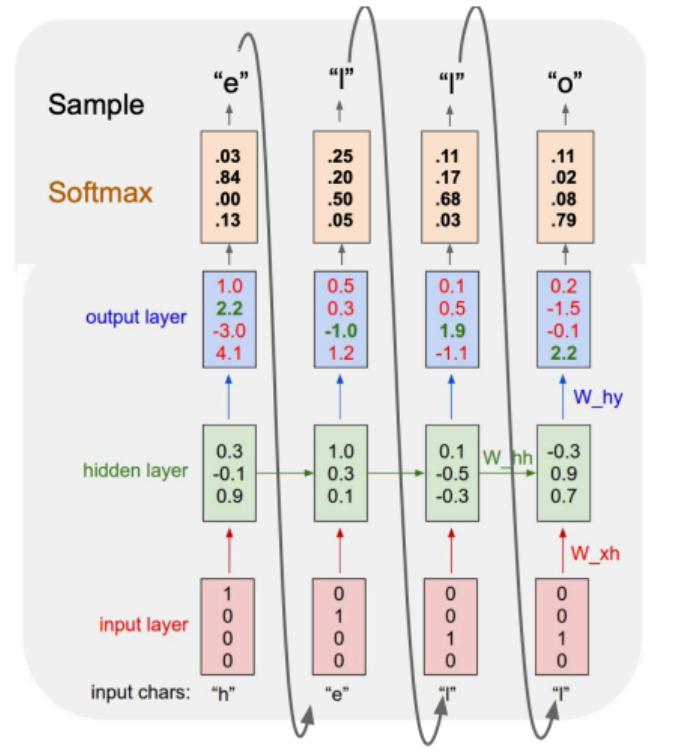
- Буквенная языковая модель (character-level Language Model)
- Словарь: [h,e,l,o]
- В тестовом режиме предсказываем следующие буквы



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

# Языковая модель

- Буквенная языковая модель (character-level Language Model)
- Словарь: [h,e,l,o]
- В тестовом режиме предсказываем следующие буквы



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

## Генерация текста

- Посыпаем на вход в модель  $\langle BOS \rangle$ , модель выдает нам распределение на первое слово при условии начального токена

$$p(x_1 | x_0 = \langle BOS \rangle)$$

- Сэмплируем токен  $x_1$  из этого распределения и посыпаем на вход в нейросетку, получаем распределение на второе слово при условии первых двух токенов

$$p(x_2 | x_0, x_1)$$

- Продолжаем, пока не выпадет токен  $\langle EOS \rangle$

## Генерация текста

- То же самое можно делать с CNN и трансформерами. Важно, чтобы модель могла кушать какую-то последовательность и выдавать распределение
- Мы можем хотеть делать генерацию не с <BOS>, а начиная с какого-то префикса — например,  $(x_1, x_2, x_3)$ . На самом деле, такая ситуация вообще ничем не отличается, переучивать не нужно. Мы просто загоняем <BOS> и эти токены в модель, а сэмплировать начинаем после следующего шага

# Температура сэмплирования

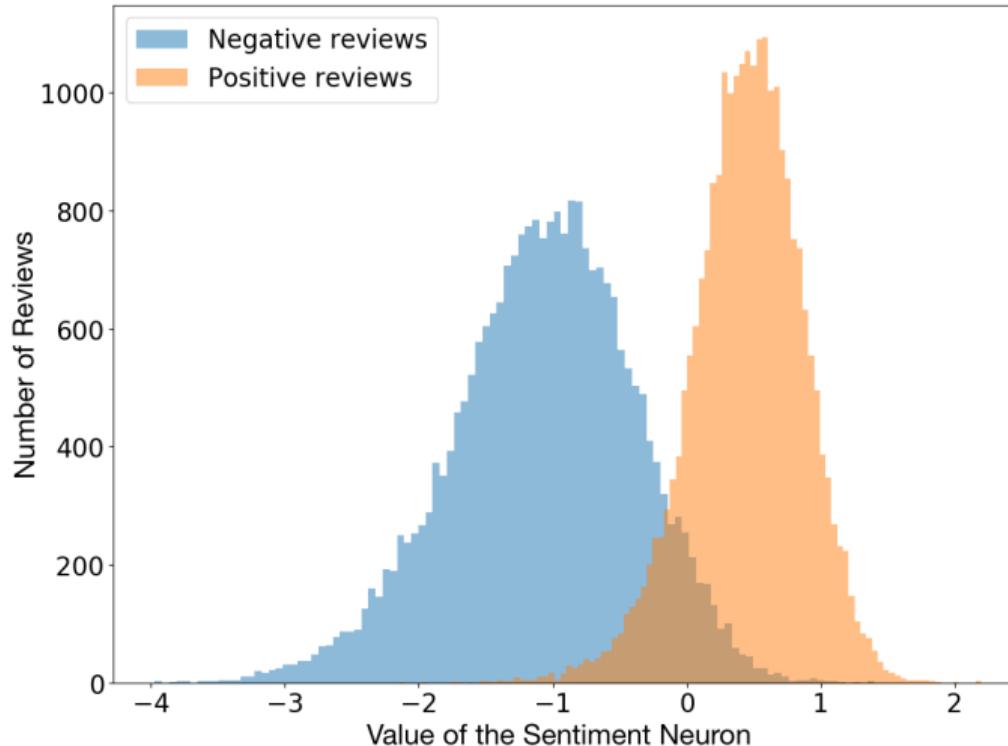
- Иногда в функцию softmax добавляют дополнительный параметр  $T$ , который называют температурой сэмплирования

$$\frac{e^{\frac{z_i}{T}}}{\sum_{k=1}^K e^{\frac{z_k}{T}}}$$

- Чем больше  $T$ , тем сильнее распределение похоже на равномерное
- Чем меньше  $T$ , тем ближе распределение к вырожденному

[https://fulyankin.github.io/deep\\_learning\\_masha\\_book/problem\\_set\\_04\\_logloss/problem\\_06.html](https://fulyankin.github.io/deep_learning_masha_book/problem_set_04_logloss/problem_06.html)

# Злобный нейрон



<https://openai.com/blog/unsupervised-sentiment-neuron/>

# Злобный нейрон

## SENTIMENT FIXED TO POSITIVE

I couldn't figure out the shape at first but it definitely does what it's meant to do. It's a great product and I recommend it highly

I couldn't figure out why this movie had been discontinued! Now I can enjoy it anytime I like. So glad to have found it again.

I couldn't figure out how to use the video or the book that goes along with it, but it is such a fantastic book on how to put it into practice!

I couldn't figure out how to use just one and my favorite running app. I use it all the time. Good quality, You cant beat the price.

I couldn't figure out how to attach these balls to my little portable drums, but these fit the bill and were well worth every penny.

## SENTIMENT FIXED TO NEGATIVE

I couldn't figure out how to use the product. It did not work. At least there was no quality control; this tablet does not work. I would have given it zero stars, but that was not an option.

I couldn't figure out how to set it up being that there was no warning on the box. I wouldn't recommend this to anyone.

I couldn't figure out how to use the gizmo. What a waste of time and money. Might as well throw away this junk.

I couldn't figure out how to stop this drivel. At worst, it was going absolutely nowhere, no matter what I did. Needless to say, I skim-read the entire book. Don't waste your time.

I couldn't figure out how to play it.

<https://openai.com/blog/unsupervised-sentiment-neuron/>