Анализ текстов

Ульянкин Филипп

21 сентября 2019 г.

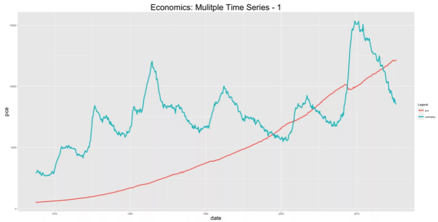
Рекурентные нейронные сетки

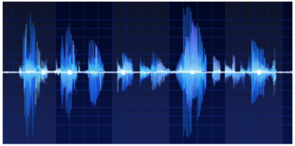
Agenda

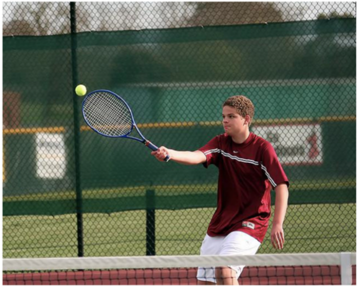
- Рекурентные нейросети
- Дописываем Евгения Онегина

Рекурентные нейронные сети

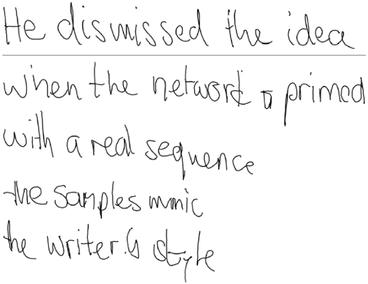
Последовательности







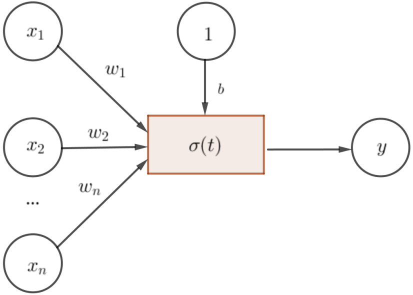
a man is playing tennis on a tennis court



Рекурентная сеть

- Каждый нейрон взаимодействует сам с собой
- На вход поступает последовательность (текст, видео, картинка, временной ряд), один и тот же нейрон просматривает её
- Впоследствие можно использовать эту сетку для генерации новых последовательностей (текстов, видео и тп)

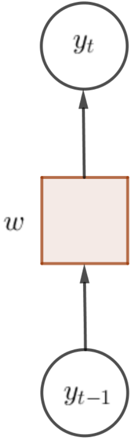
От регрессии к нейрону



$$
\begin{aligned}
y_i =& b + w \cdot x_i \\
& \Downarrow \\
h_i =& b + w \cdot x_i \\
y_i =& f(h_i)
\end{aligned}
$$

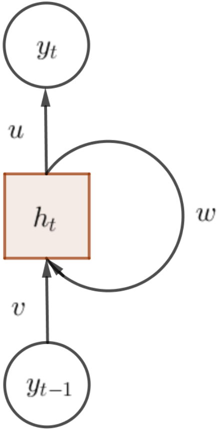
От авторегрессии к нейрону

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



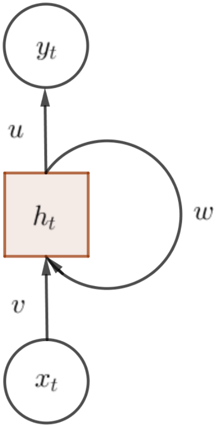
От авторегрессии к нейрону

$$
\begin{aligned}
y_t =& b + w \cdot y_{t-1} \\
&\Downarrow \\
h_t =& f_h(b_h + w \cdot h_{t-1} + v \cdot y_{t-1}) \\
y_t =& f_y(b_y + u \cdot h_t)
\end{aligned}
$$

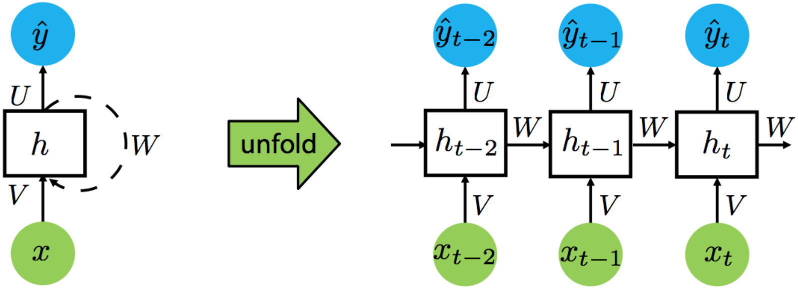


От авторегрессии к нейрону

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



Рекурентная сеть



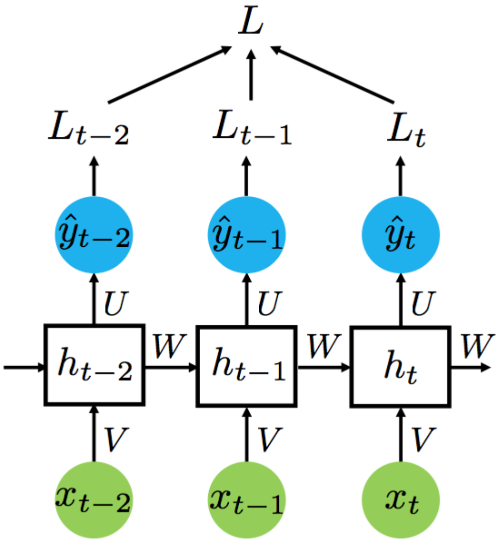
$$h_t = f_h(Vx_t + Wh_{t-1} + b_h) \qquad \hat{y}_t = f_y(Uh_t + b_y)$$

Рекурентная сеть

- **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

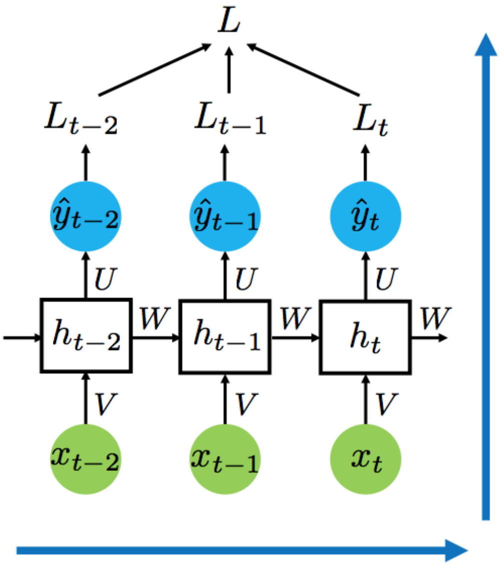
Рекурентная сеть

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



Forward pass:

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

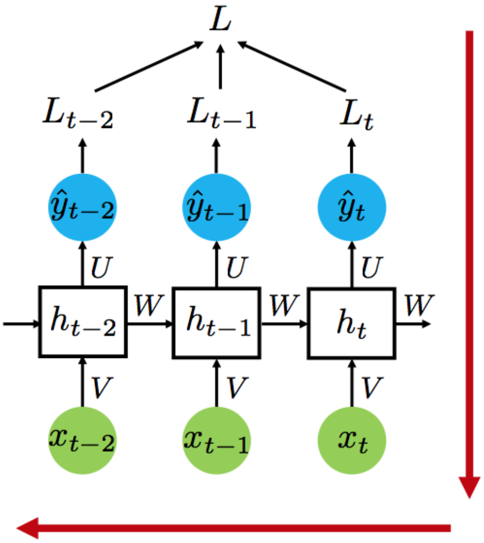


Forward pass:

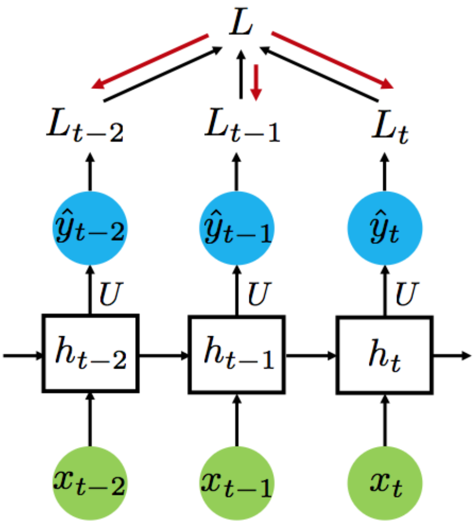
$$h_t,\ \hat{y}_t,\ L_t,\ L$$

Backward pass:

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



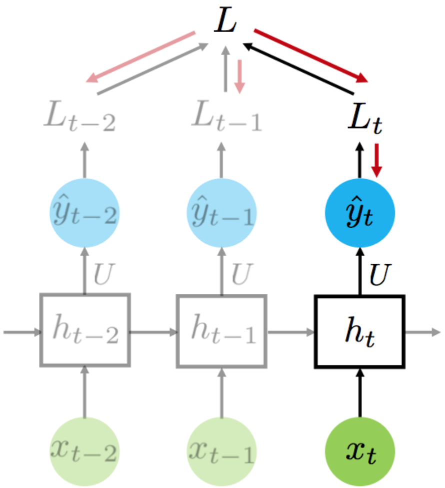
$$\frac{\partial L}{\partial U} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial U}$$



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

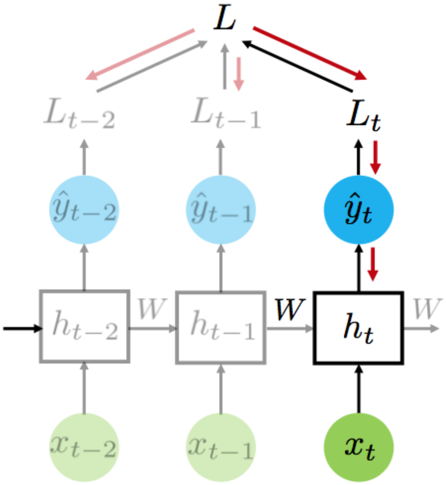
$$\frac{\partial L_t}{\partial U} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial U}$$

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



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

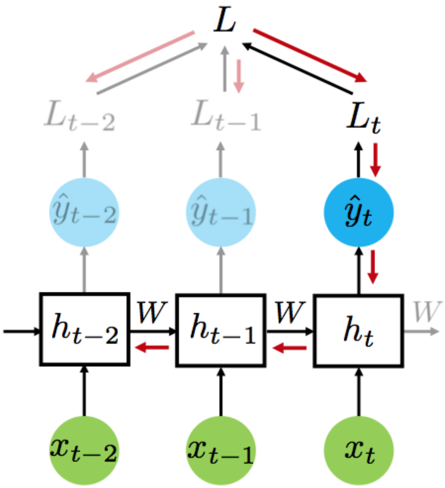
$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W}$$



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

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

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



$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \boxed{\left(\frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W} + \ldots \right)}$$

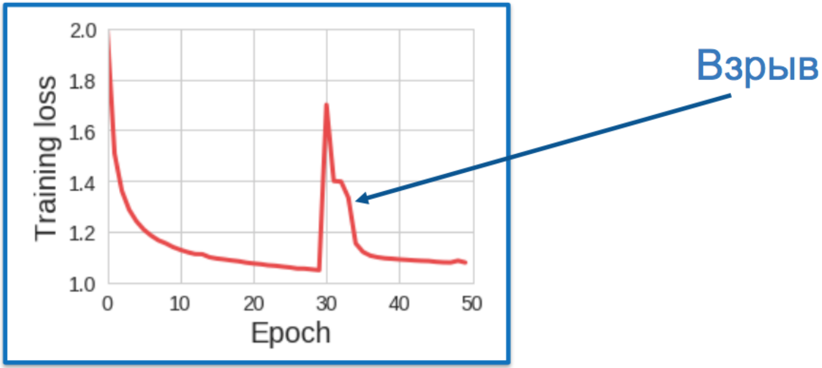
Vanishing and Exploding

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

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

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

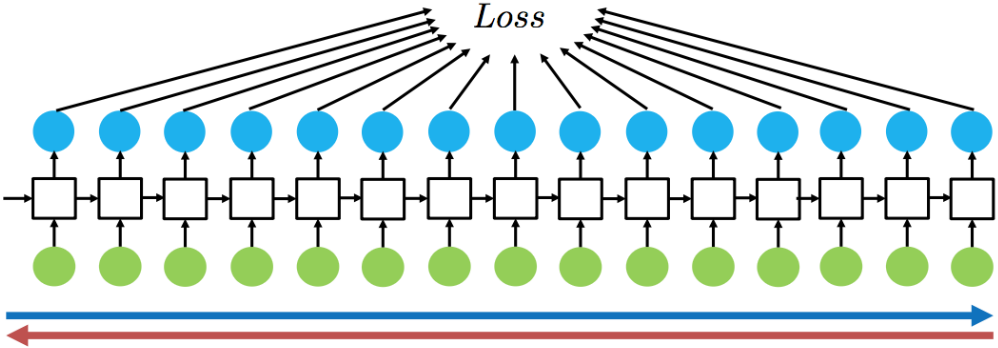
Как понять, что градиент взорвался?



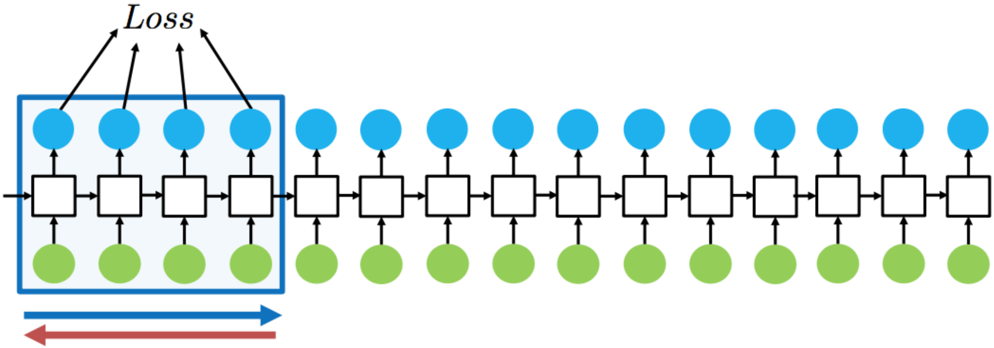
Как предотвратить взрыв?

- Поставить порог, которые не будет пробиваться градиентом (gradient clipping)
- Обучать сетку не целиком, а по кусочкам
- Аккуратно инициализировать веса
- Делать skip-connection
- Придумать специальную архитектуру

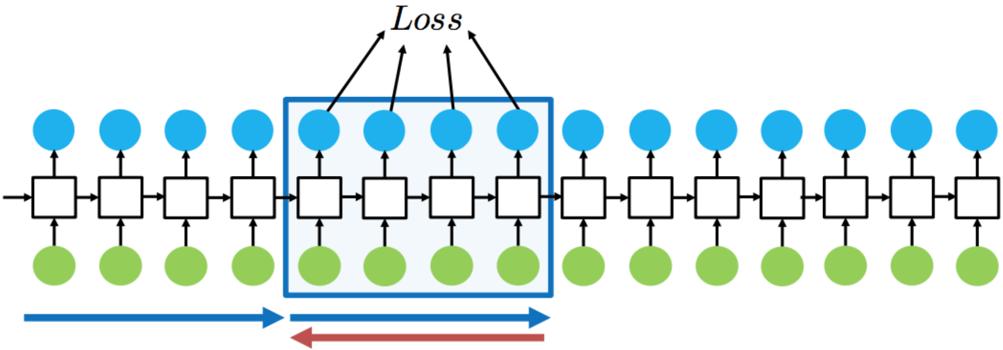
Урезанное обучение



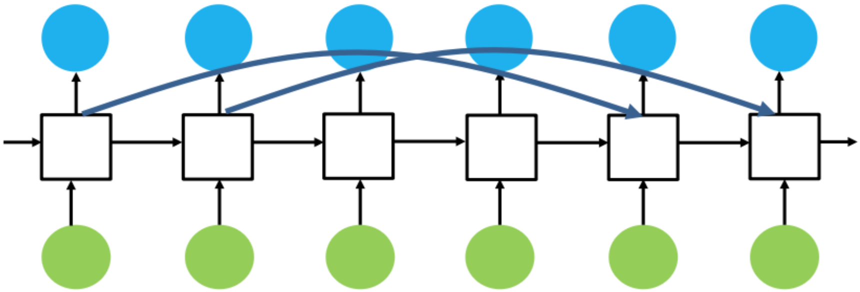
Урезанное обучение



Урезанное обучение



Skip-connection

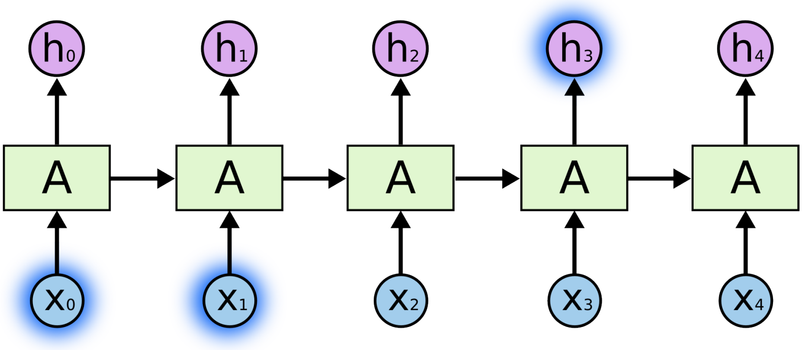


LSTM (long short-term memory)

Короткая память

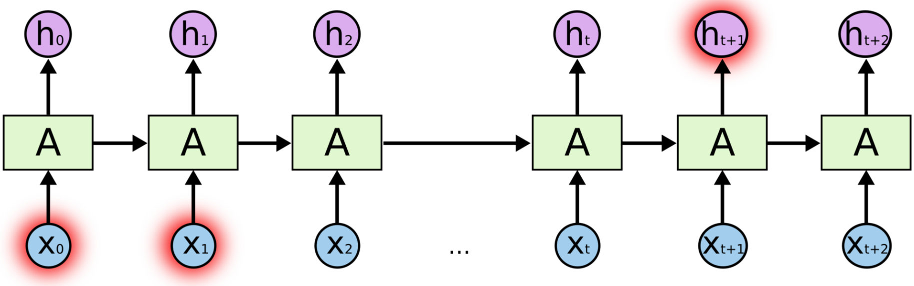
- Привлекательность RNN в том, что они потенциально умеют связывать предыдущую информацию с текущей
- При backpropagation текущие градиенты пробрасываются во времени назад
- Если градиент не взрывается, он постепенно затухает, получается что влияние текущего слоя не может пробросится во времени слишком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

Короткая память



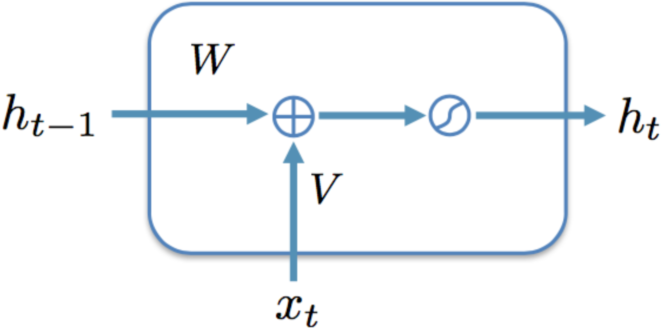
Облака плывут по небу

Короткая память



Я вырос во Франции… Я бегло говорю по-французски

Простейшая RNN

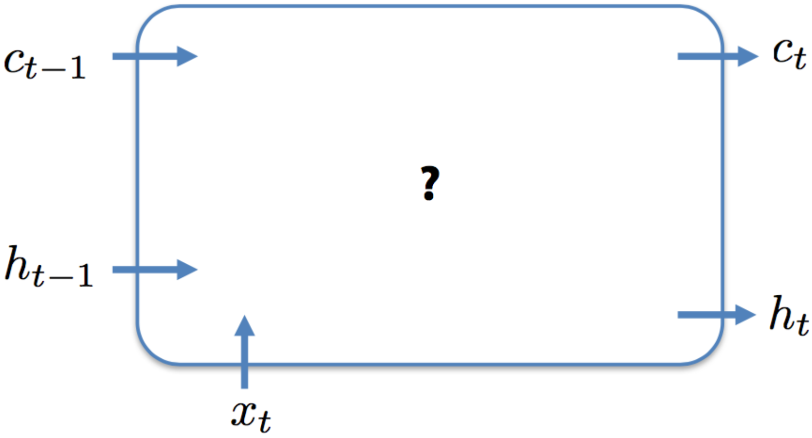


$$h_t = \tilde{f}(Vx_t + Wh_{t-1} + b_h)$$

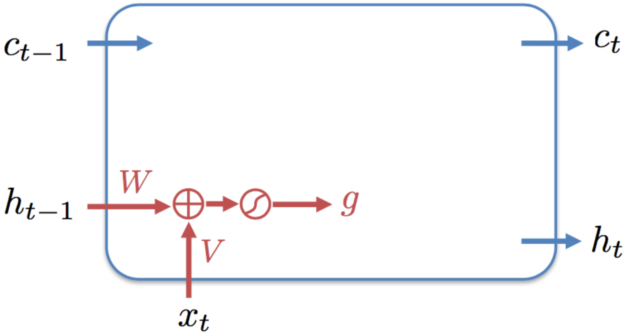
Долгая краткосрочная память

- Архитектура LSTM лечит эту проблему, она способна к обучению долговременным зависимостям.
- Внутри рекуррентной ячейки долгосрочная память моделируется явным образом. Конечно же нам из-за этого придётся учить больше параметров.
- Завораживающе! Так как же она выглядит?

LSTM

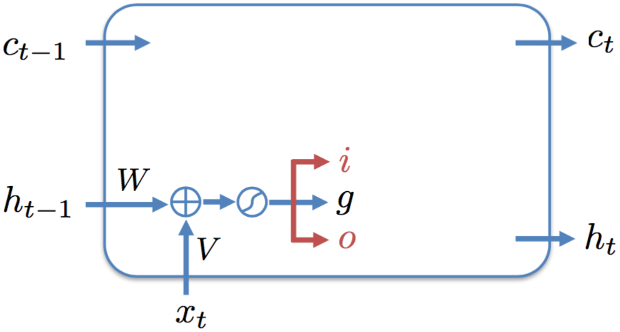


LSTM



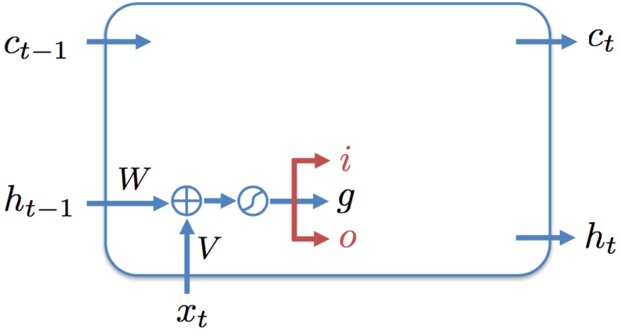
$$g_t = \tilde{f}(V_g x_t + W_g h_{t-1} + b_g)$$

LSTM



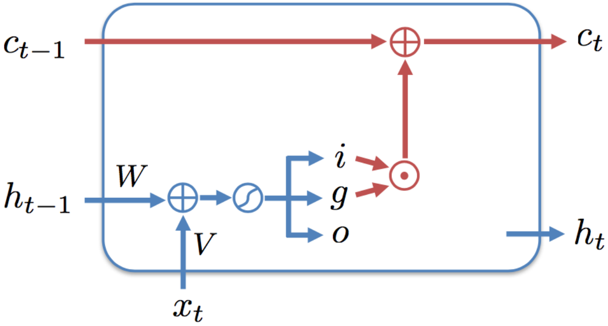
$$g_t = \tilde{f}(V_g x_t + W_g h_{t-1} + b_g)$$
$$i_t = \sigma(V_i x_t + W_i h_{t-1} + b_i)$$
$$o_t = \sigma(V_o x_t + W_o h_{t-1} + b_o)$$

LSTM



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

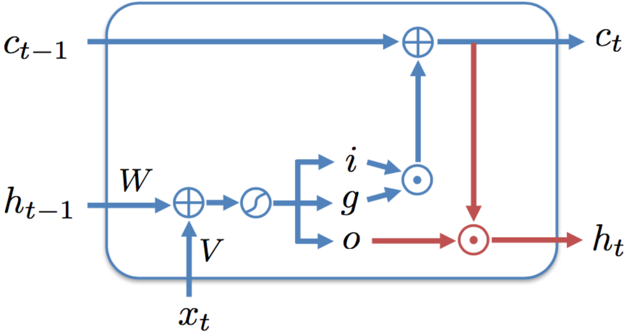
LSTM



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

$$c_t = c_{t-1} + i_t \cdot g_t$$

LSTM



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

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

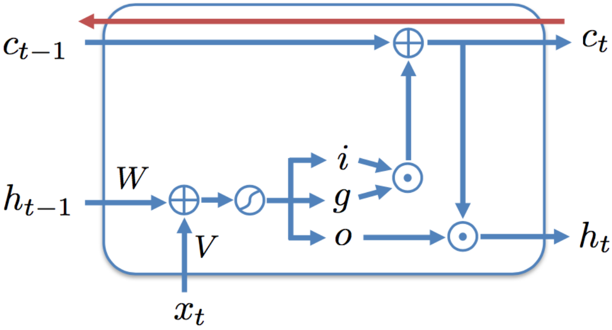
LSTM

- Ключевой элемент LSTM это состояние ячейки (cell state, c_t). Она проходит напрямую через цепочку, участвую 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



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

Gradients do not vanish!

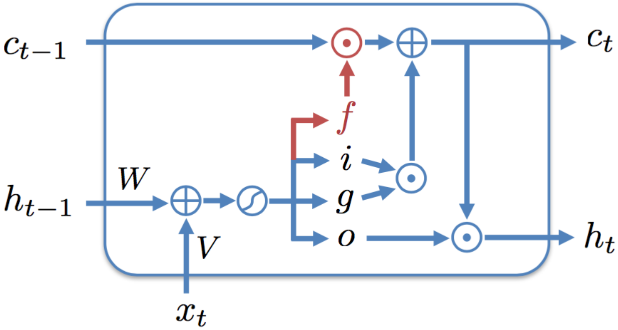
LSTM

- 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. Ошибка сама затухать не будет. Однако проблема взрывающихся градиентов остаётся.

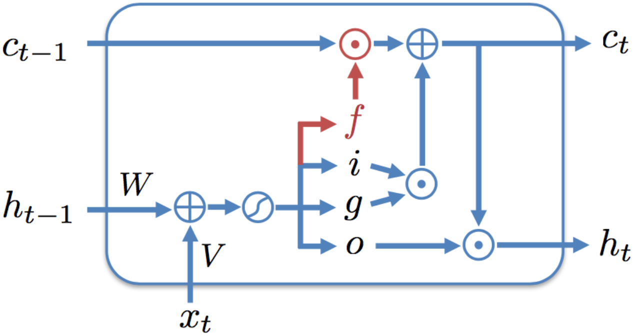
LSTM с забыванием



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

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

LSTM с забыванием



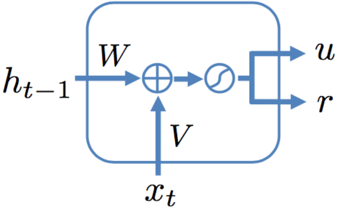
$$f_t = \sigma(V_f x_t + W_f h_{t-1} + b_f) \qquad c_t = f_t \cdot c_{t-1} + i_t \cdot g_t$$

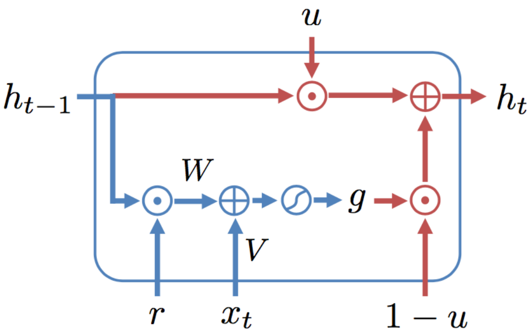
$$\frac{\partial c_t}{\partial c_{t-1}} = f_t \quad \Rightarrow \quad \text{High initial } b_f$$

А вот бы весов поменьше бы

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

GRU-ячейка





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

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

$$h_t = (1 - u_t) \cdot g_t + u_t \cdot h_{t-1}$$