LSTM - Сети с Долгой краткосрочной памятью

Преподаватель: Герард Костин

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

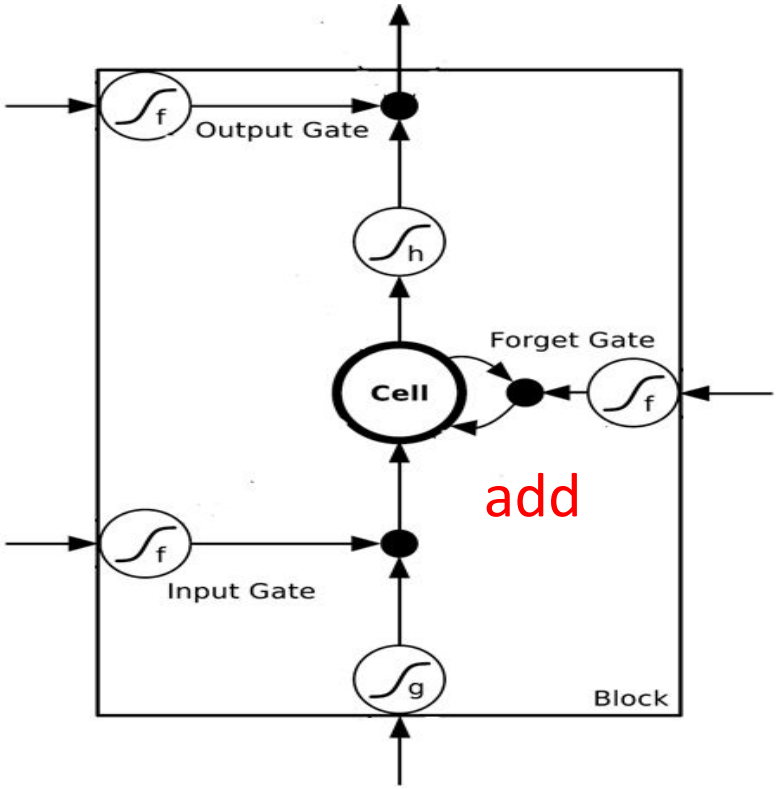
Проблемы RNN

- Ошибки, распространяющиеся в обратном направлении, умножаются на каждом слое, что приводит к экспоненциальному убыванию (если производная мала) или росту (если производная большая).
- Очень сложно реализовать обучение глубоких сетей или простых RNN сетей на большой временной глубине
- Очень сложно выучить зависимости на большом расстоянии, такие как согласование субъект-глагол.

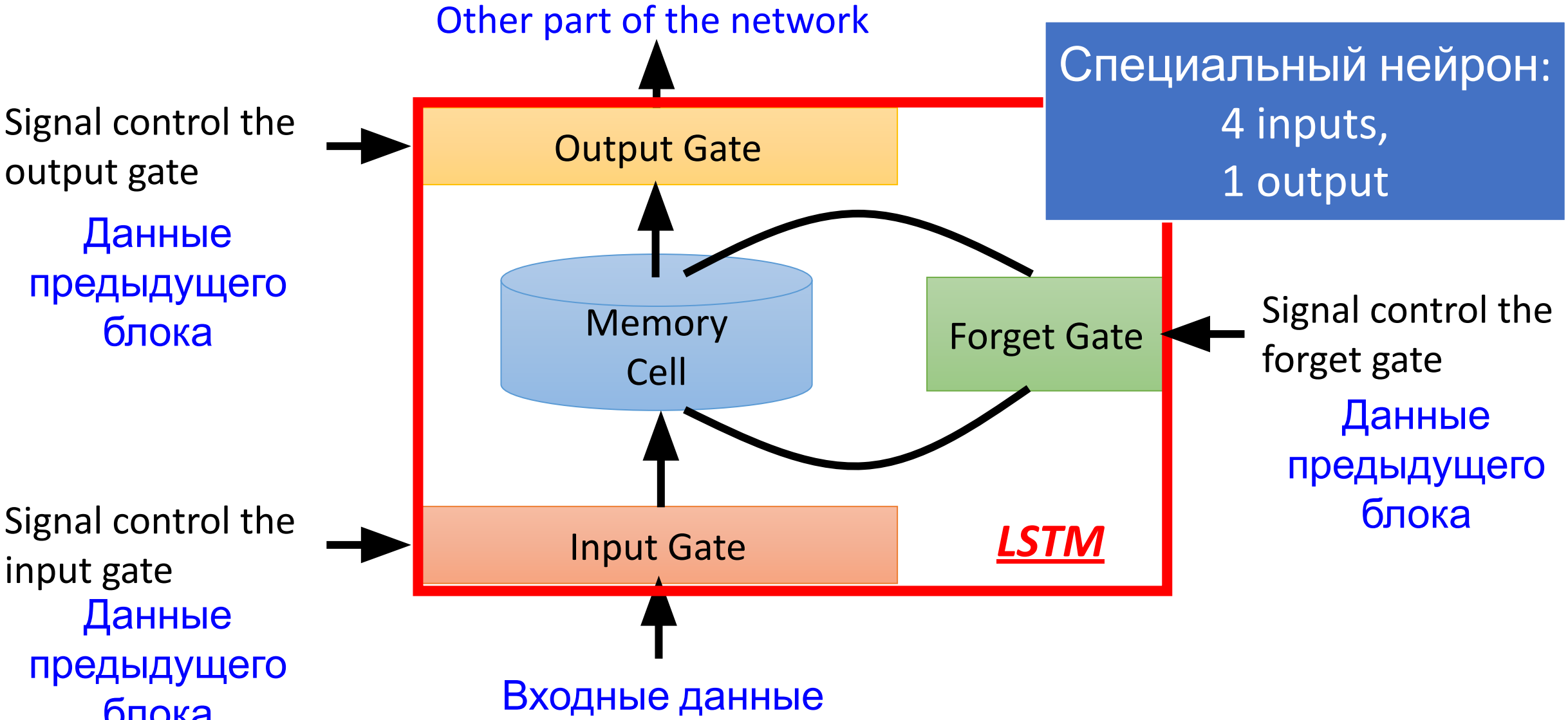
Как побороть проблемы RNN

- Long Short-term Memory (LSTM)
  - Могут работать с исчезающим градиентом (НЕ! gradient explode)
    - Память и Вход складываются (***added)***
    - Информация никогда не теряется, если вентиль “забвения” закрыт

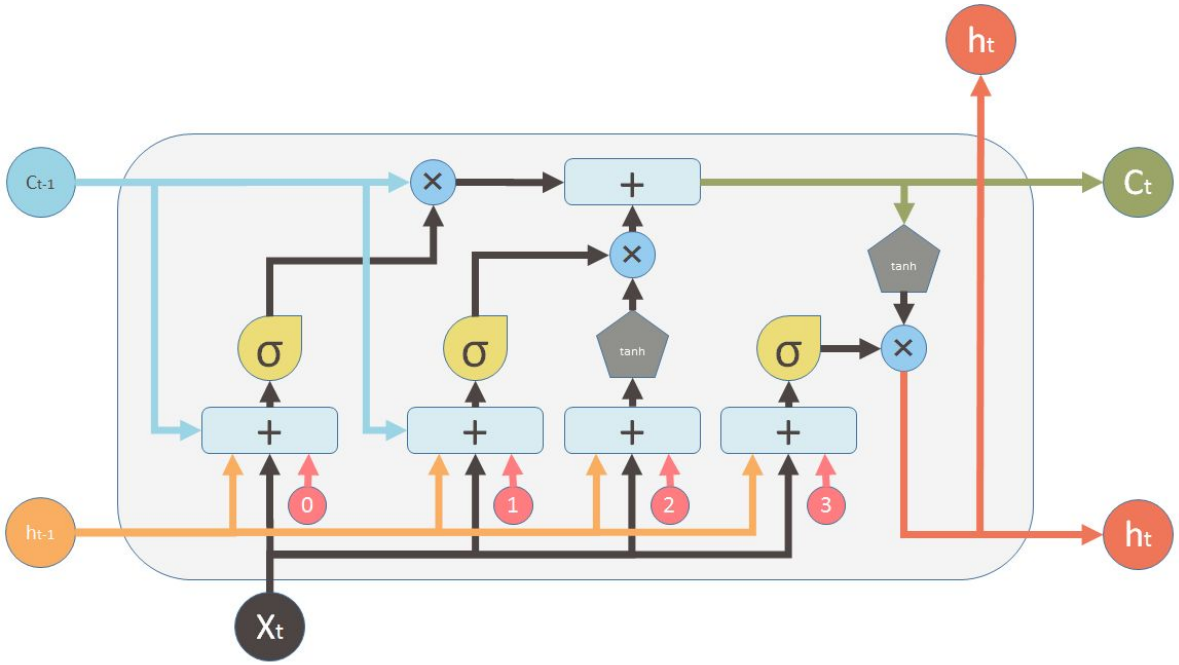
Нет Gradient vanishing
(Если вентиль “забвения” открыт.)

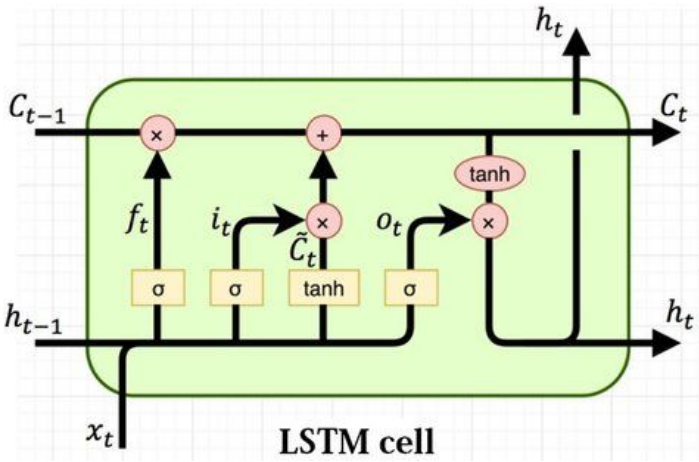


Long Short-term Memory (LSTM)



Long Short-term Memory (LSTM)





$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

Inputs:

- Xt — Input vector
- Ct-1 — Memory from previous block
- ht-1 — Output of previous block

outputs:

- Ct — Memory from current block
- ht — Output of current block

Nonlinearities:

- σ — Sigmoid
- tanh — Hyperbolic tangent

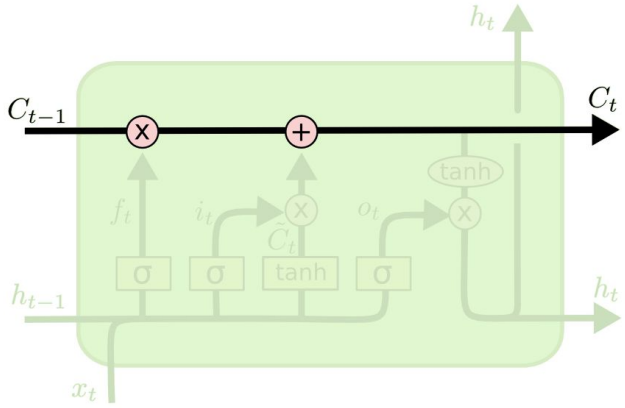
Vector operations:

- × — Element-wise multiplication
- \+ — Element-wise Summation / Concatenation

Bias: 0

Cell State

- Обрабатывает вектор C_t который имеет ту же размерность, что и скрытый слой, h_t
- Информация может быть добавлена или удалена из этого вектора состояния через шлюзы забвения и ввода.

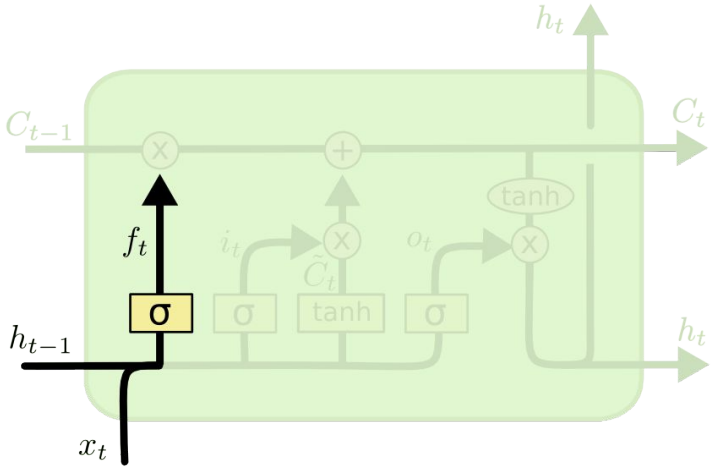


Cell State Пример

- Хотите запомнить лицо и порядок подлежащего существительного, чтобы его можно было проверить на соответствие объекту и глаголу, когда оно в конечном итоге встретится.
- Шлюз забвения удалит существующую информацию о предыдущем предмете при обнаружении нового.
- Входной шлюз «добавляет» информацию о новом предмете.

Forget Gate

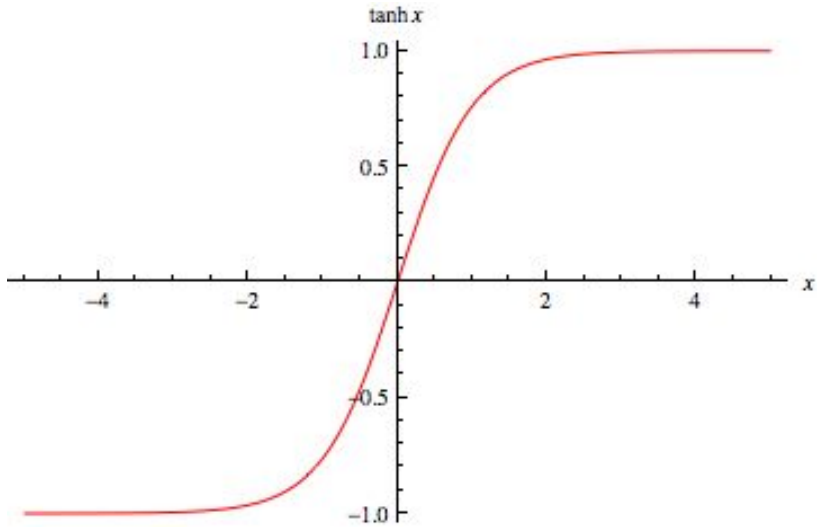
- Forget gate вычисляет значение 0-1, используя логистическую функцию вывода сигмоида из входных данных, x_t, аи текущее скрытое состояние, h_t:
- Мультипликативно сочетается с состоянием ячейки, «забывая» информацию, когда вентиль выводит что-то близкое к 0.



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

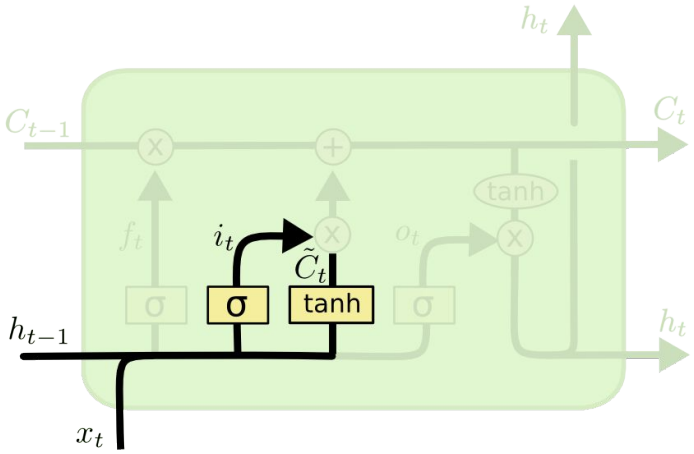
Hyperbolic Tangent Активация

- Tanh можно использовать в качестве альтернативной нелинейной функции сигмовидной логистической выходной функции (0-1).
- Используется для вывода пороговых значений от –1 до 1.



Input Gate

- Во-первых, определяет, какие записи в состоянии ячейки обновлять, вычисляя сигмоидальный выход 0-1.
- Затем определяет, какую сумму добавить / вычесть из этих записей, вычислив функцию выхода tanh (со значением от –1 до 1) входного и скрытого состояния.
-

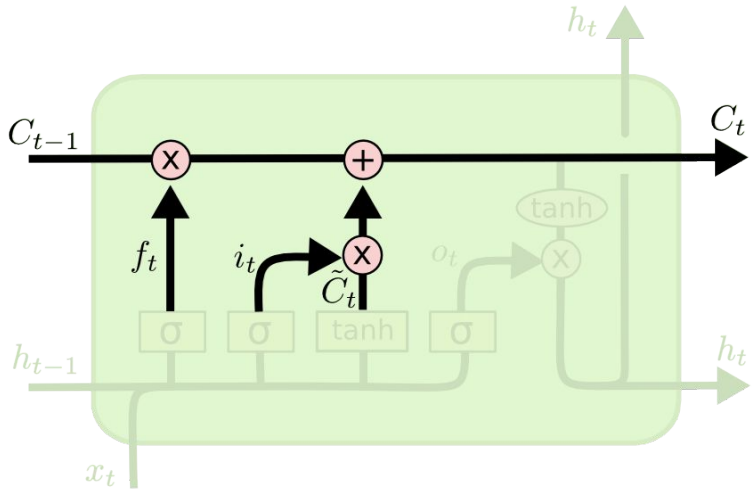


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Обновление Cell State

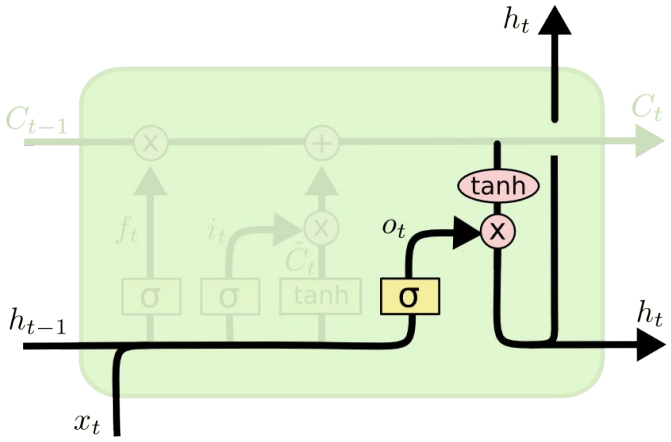
- Состояние ячейки обновляется с помощью покомпонентного векторного умножения, чтобы «забыть», и векторного сложения для «ввода» новой информации.



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

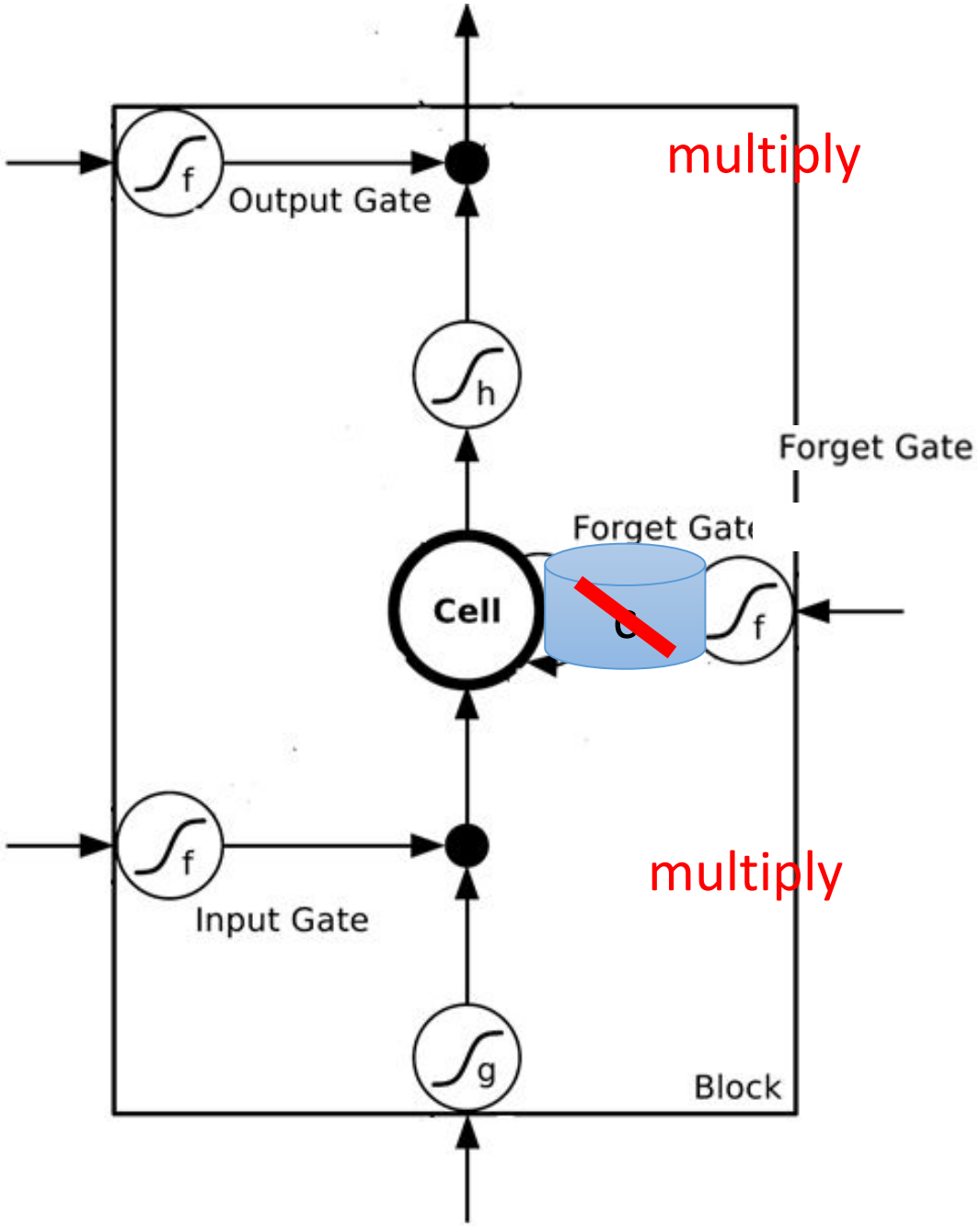
Output Gate

- Скрытое состояние обновляется на основе «отфильтрованной» версии состояния ячейки, масштабируемой от –1 до 1 с использованием tanh.
- Выходной вентиль вычисляет сигмовидную функцию входа и текущего скрытого состояния, чтобы определить, какие элементы состояния ячейки нужно «выводить».



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

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

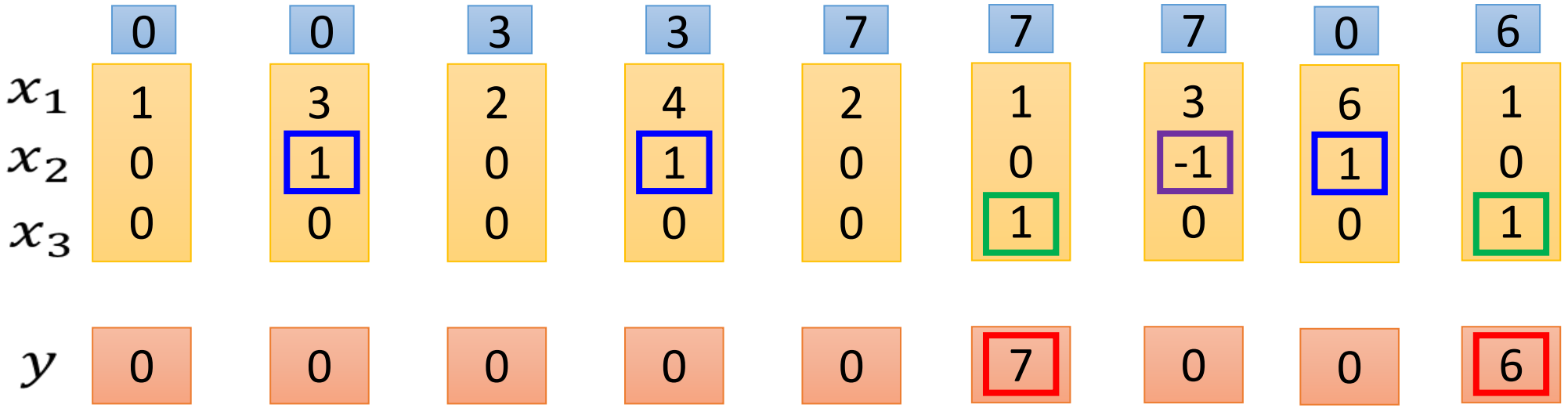


Функция активации - sigmoid function

Между 0 и 1

За счет этого определяется на сколько вентиль “забвения” пропускает информацию

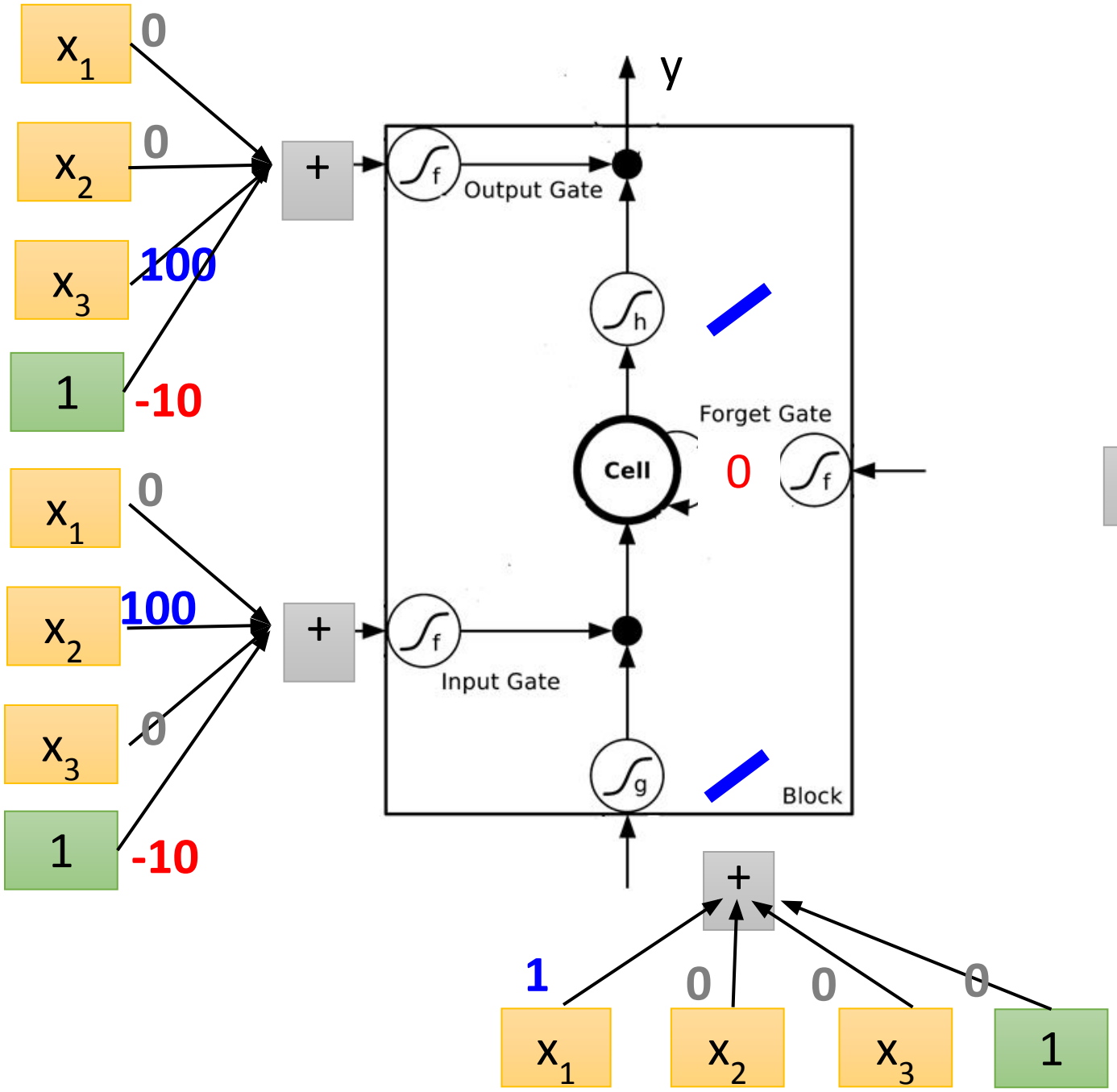
LSTM - Разбор

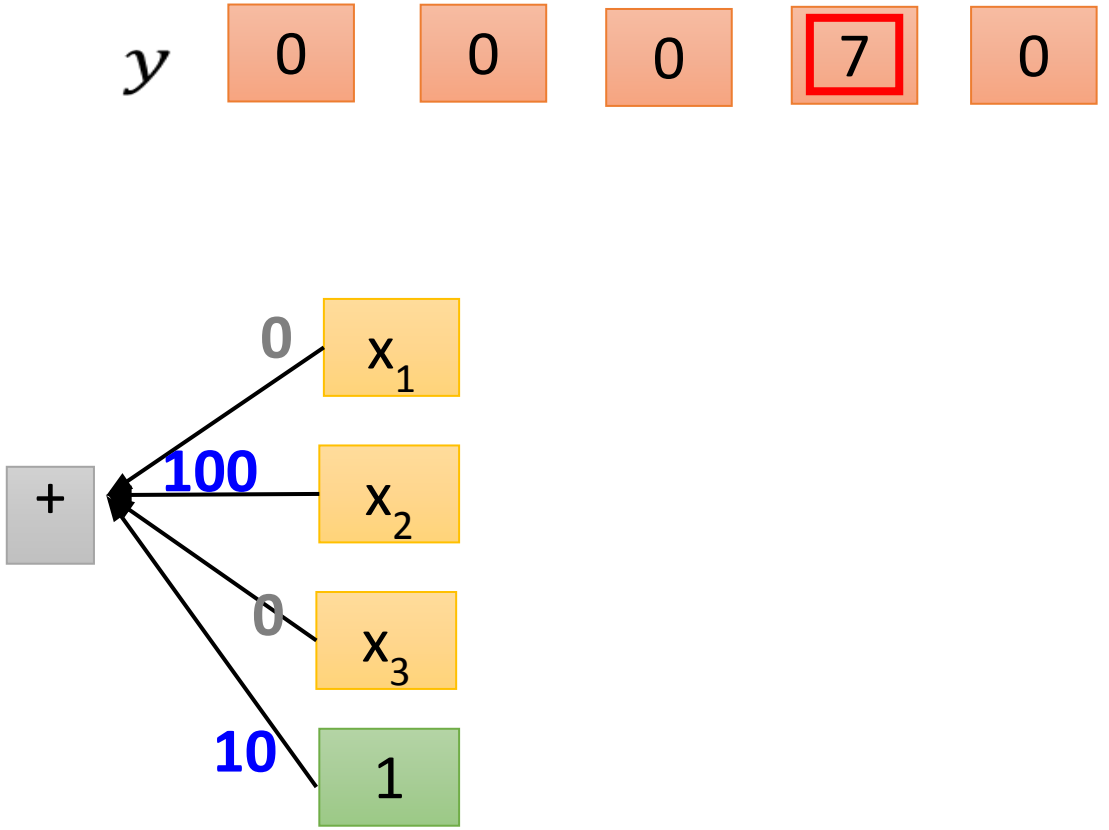


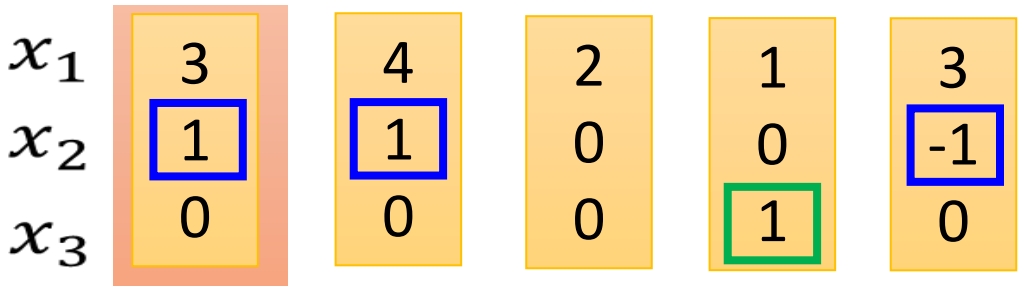
Когда $x_2 = 1$, добавляем x_1 в память

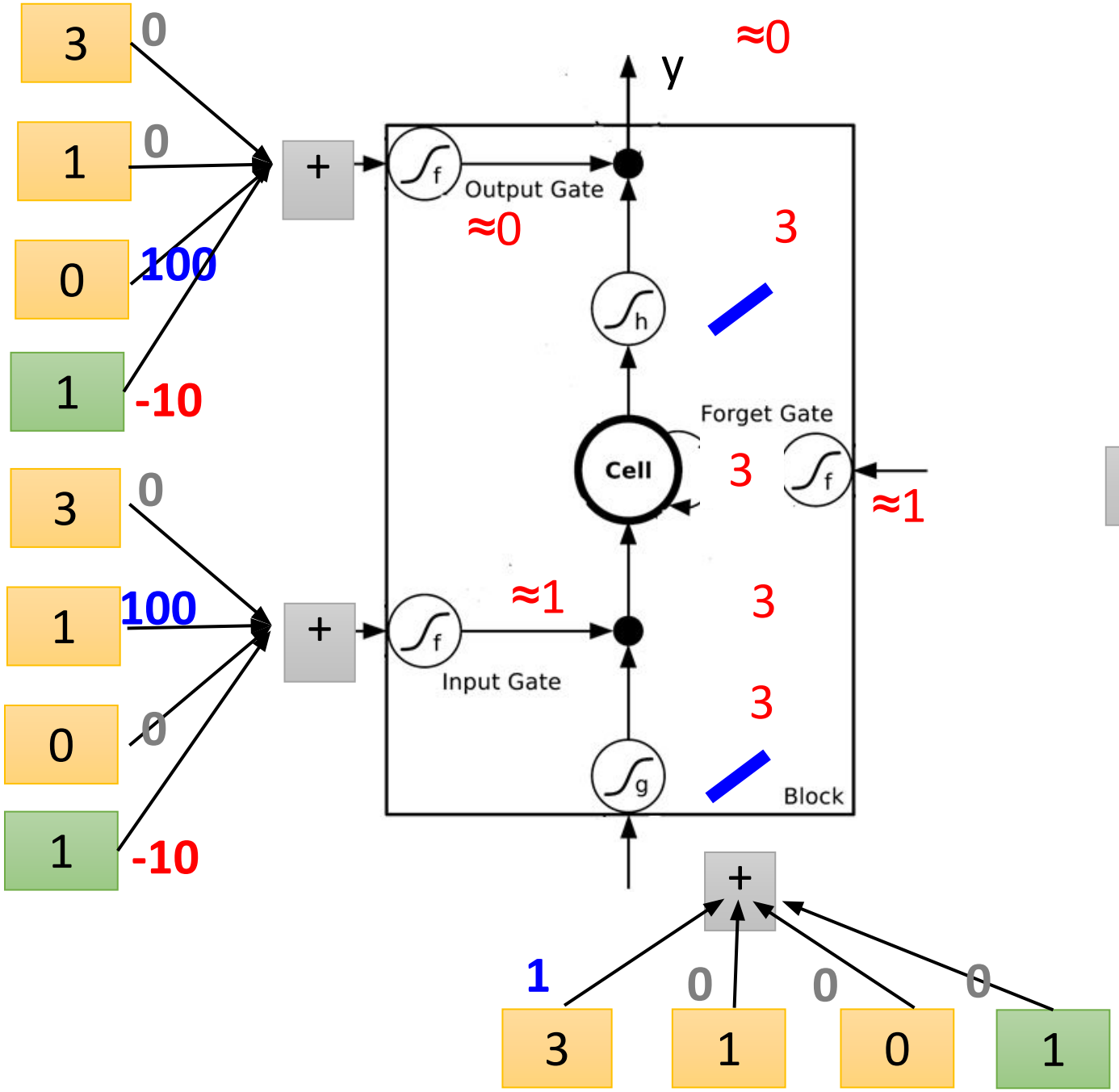
Когда $x_2 = -1$, очищаем память

Когда $x_3 = 1$, выводим данные в память

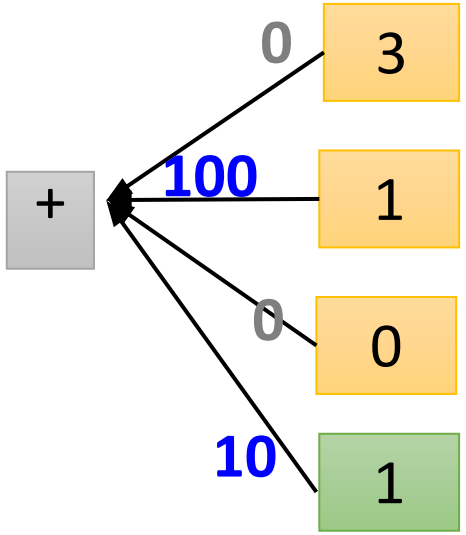

x1
0
x2
0
x3
100
1
-10
+
f
Output Gate
y
h
Forget Gate
Cell
0
f
x1
0
x2
100
x3
0
1
-10
+
f
Input Gate
g
Block
+
1
0
0
0
x1
x2
x3
1

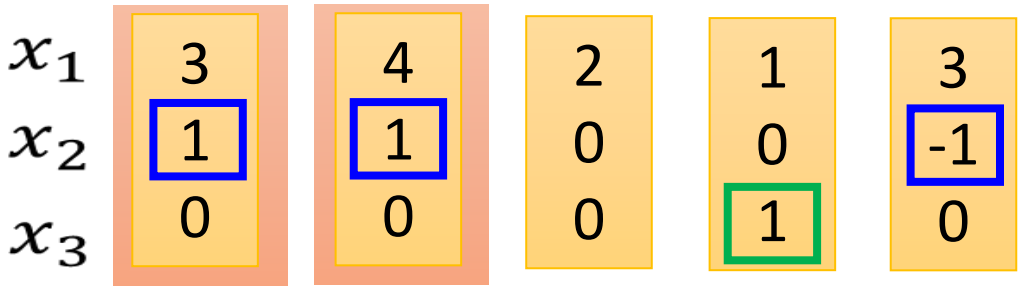

y
0
0
0
7
0
0
x1
100
x2
0
x3
10
1
+

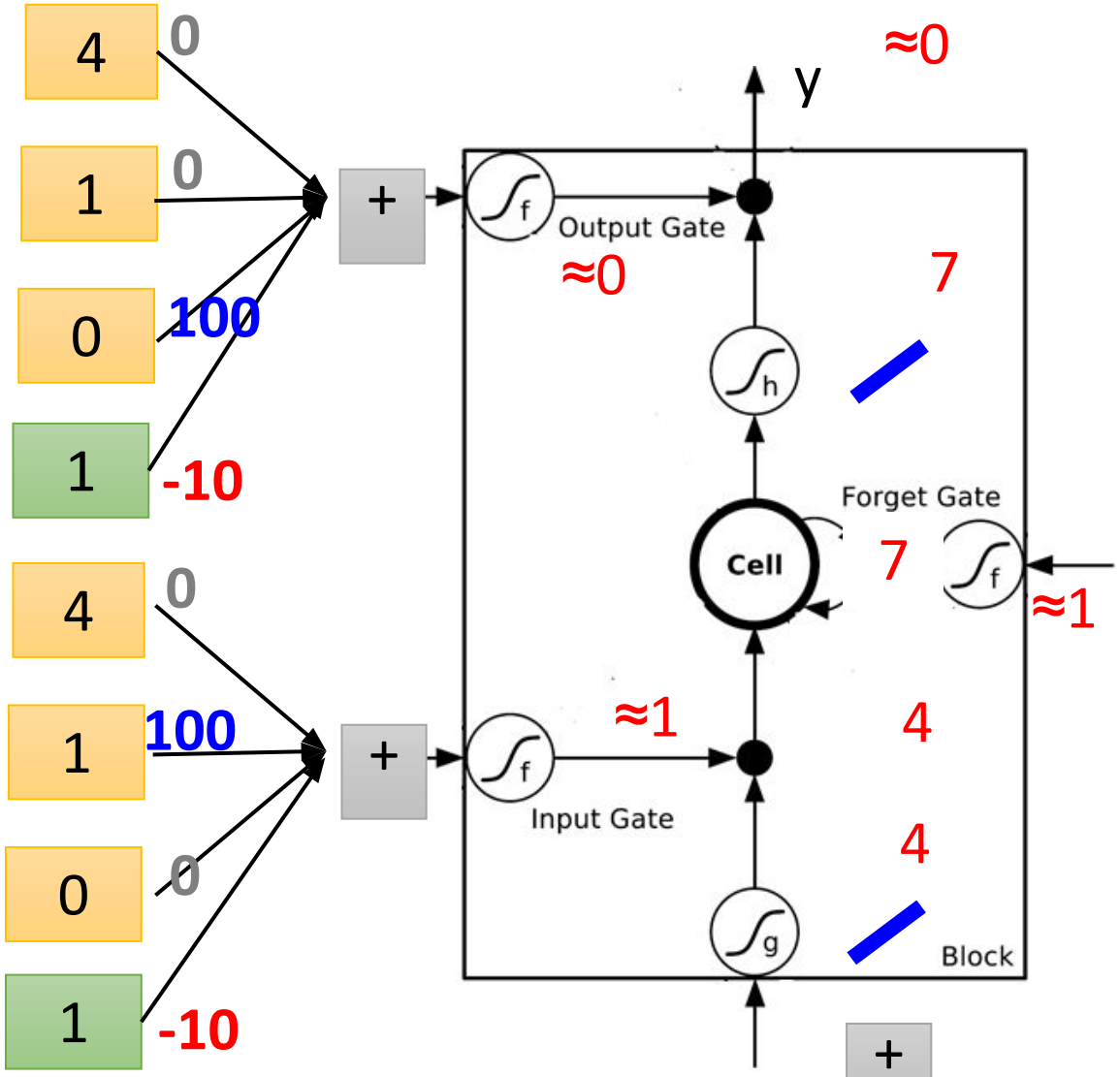

x1
x2
x3
3
1
0
4
1
0
2
0
0
1
0
1
3
-1
0


3
0
1
0
0
100
1
-10
y
≈0
+
f
Output Gate
≈0
3
h
Forget Gate
Cell
3
f
≈1
3
0
1
100
+
f
≈1
Input Gate
3
0
0
3
1
-10
g
Block
+
1
0
0
0
3
1
0
1

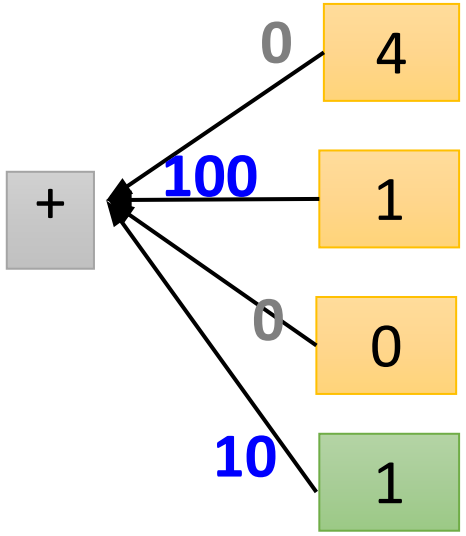

y
0
0
0
7
0

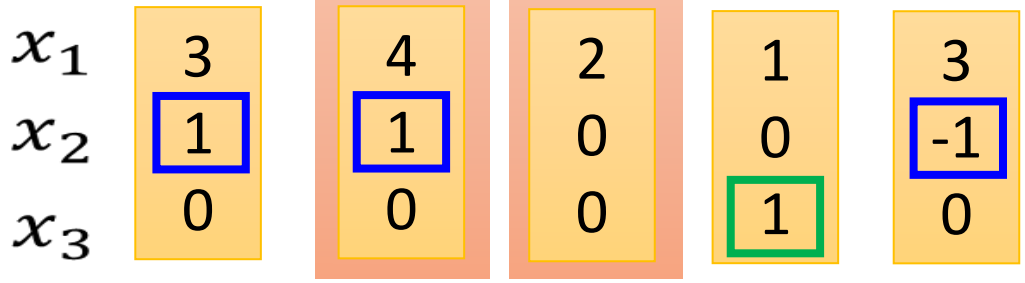

0
3
+
100
1
0
0
10
1

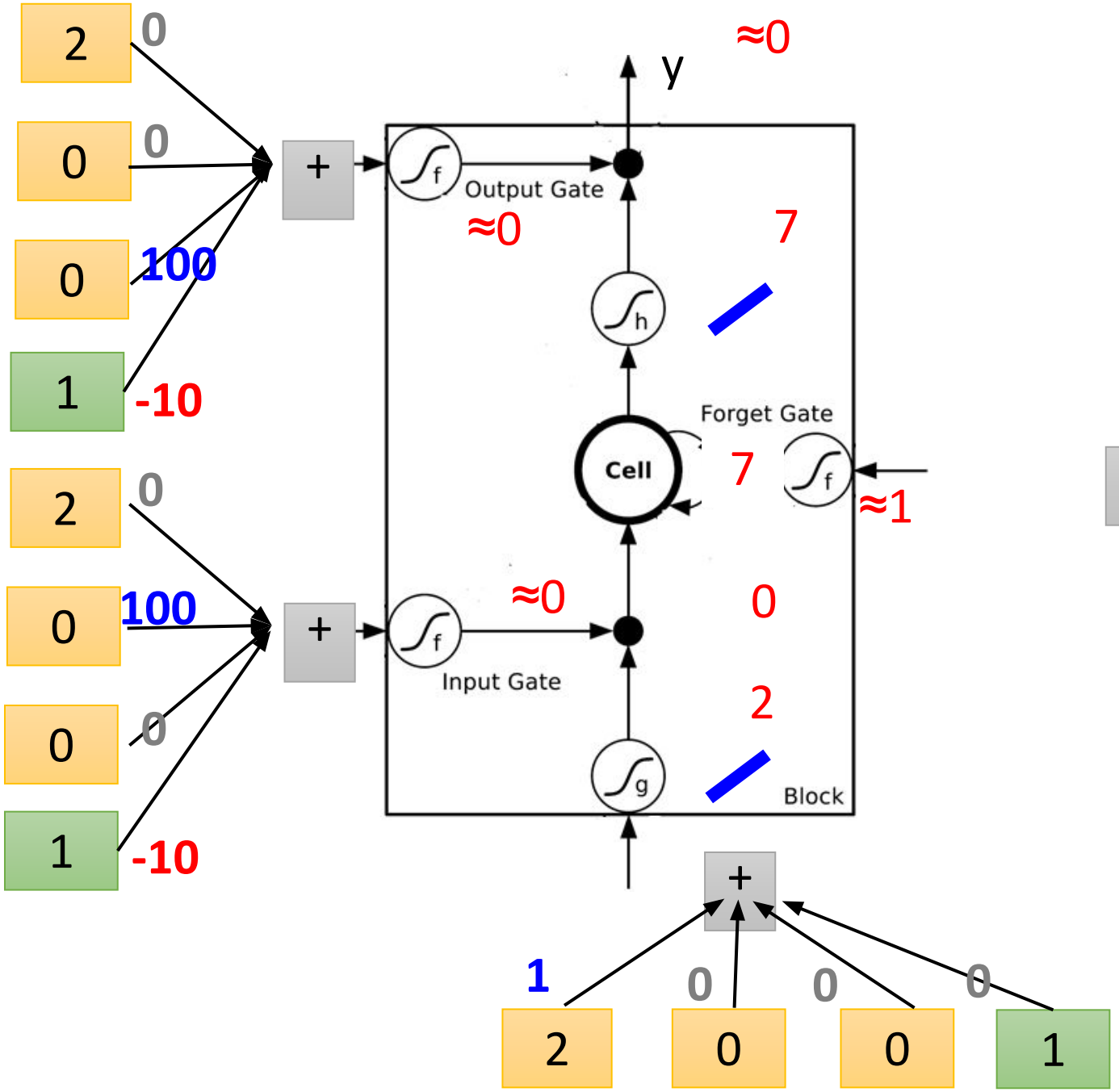

x_1
x_2
x_3
3
1
0
4
1
0
2
0
0
1
0
1
3
-1
0


4
1
0
1
0
100
-10
+
y
≈0
f
Output Gate
≈0
7
h
Forget Gate
Cell
7
f
≈1
4
1
0
1
100
0
-10
+
f
≈1
Input Gate
4
4
g
Block
+
1
0
0
0
4
1
0
1

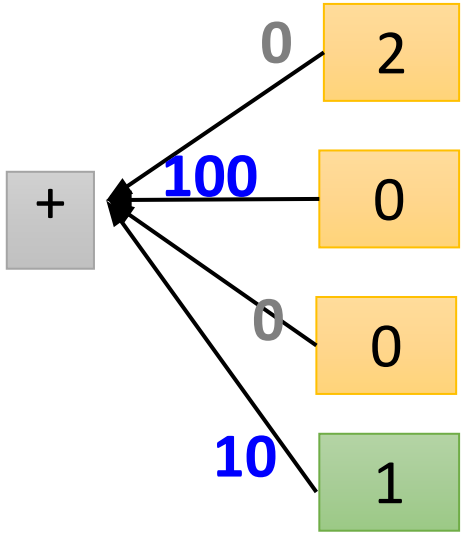

y
0
0
0
7
0

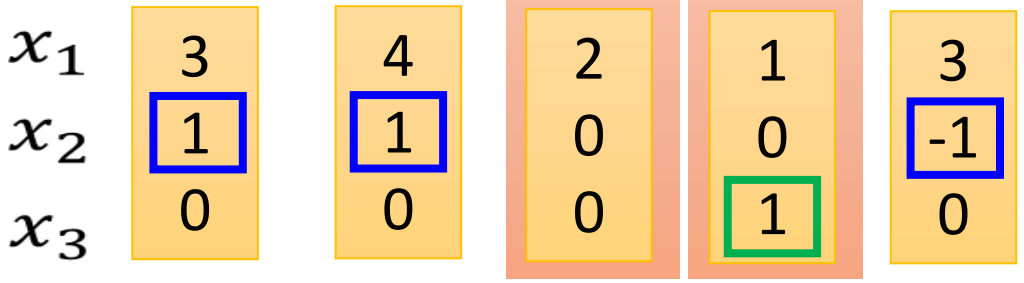

0
4
+
100
1
0
0
10
1

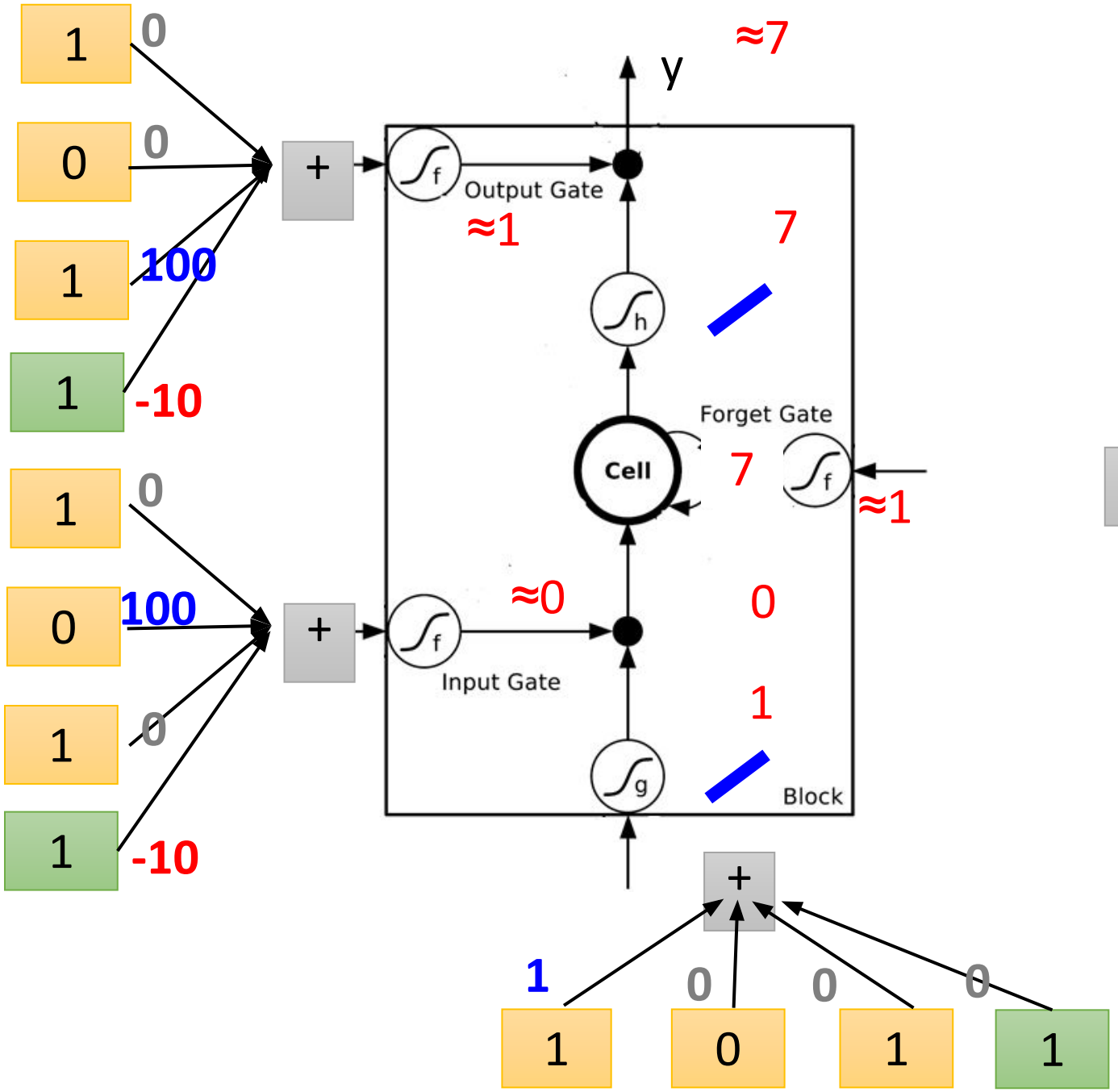

x_1
x_2
x_3
3
1
0
4
1
0
2
0
0
1
0
1
3
-1
0


2
0
0
1
0
0
100
-10
+
y
≈0
f
Output Gate
≈0
7
h
Cell
Forget Gate
7
f
≈1
2
0
0
1
100
0
-10
+
f
Input Gate
≈0
0
2
g
Block
+
1
0
0
0
2
0
0
1

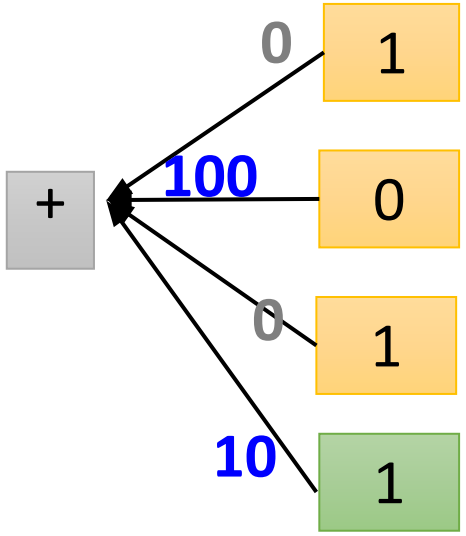

y
0
0
0
7
0

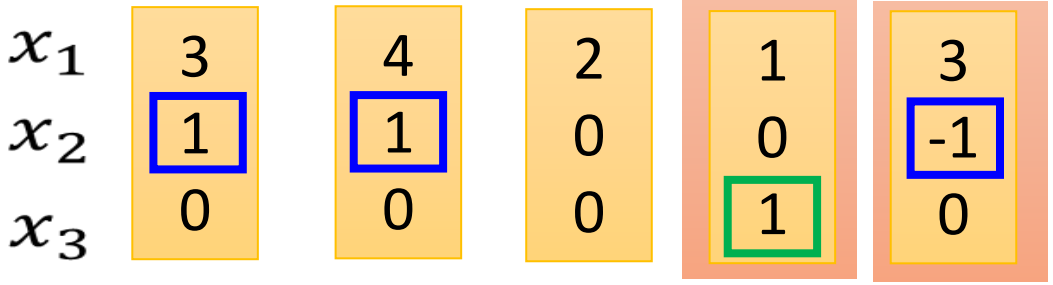

0
2
+
100
0
0
0
10
1

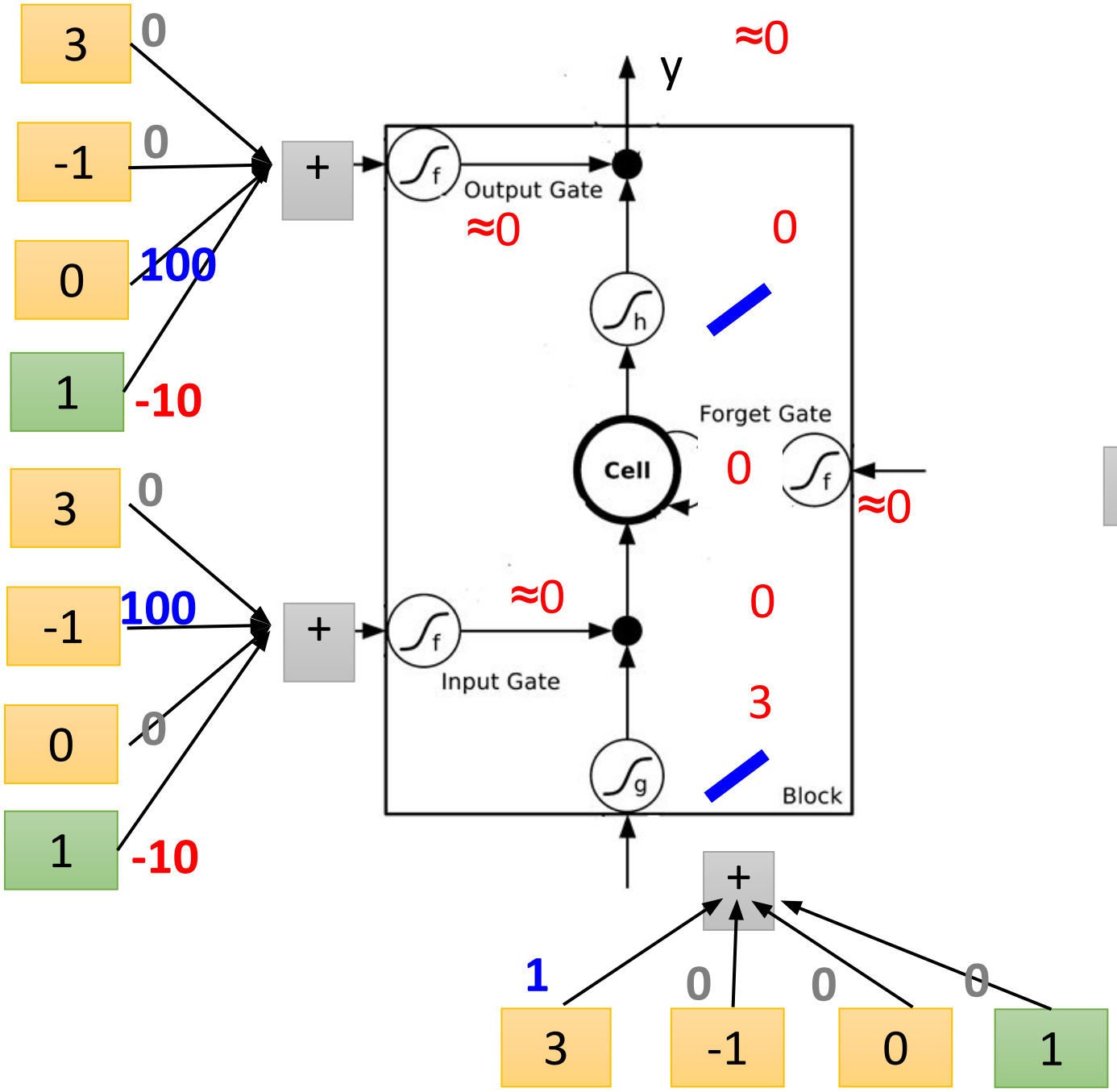

x_1
x_2
x_3
3
1
0
4
1
0
2
0
0
1
0
1
3
-1
0


1
0
0
0
1
100
1
-10
+
y
≈7
f
Output Gate
≈1
7
h
Forget Gate
Cell
7
f
≈1
1
0
0
100
1
0
1
-10
+
f
≈0
0
Input Gate
1
g
Block
+
1
0
0
0
1
0
1
1

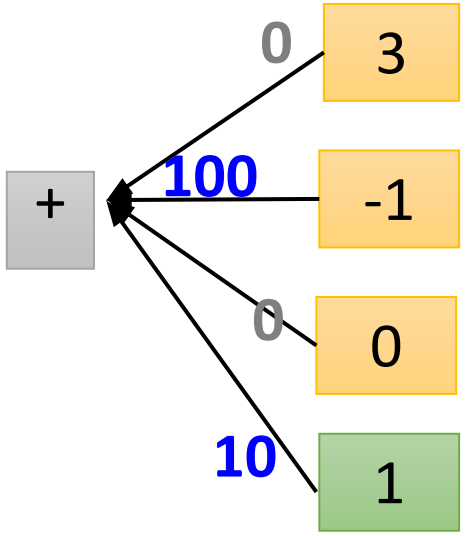

y
0
0
0
7
0

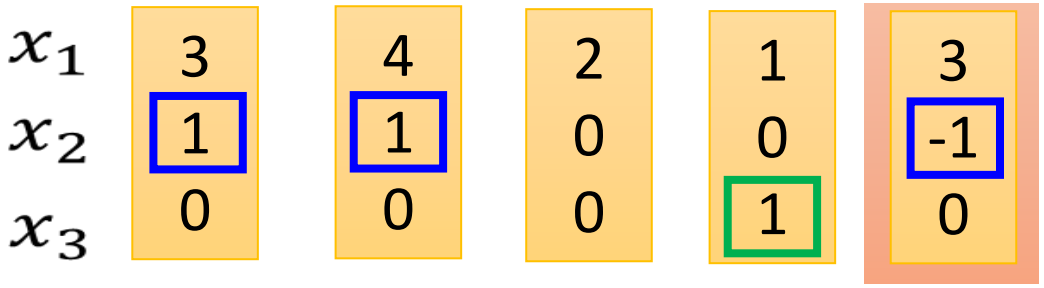

0
1
+
100
0
0
1
10
1


x_1
x_2
x_3
3
1
0
4
1
0
2
0
0
1
0
1
3
-1
0


3
0
-1
0
100
0
1
-10
+
y
≈0
f
Output Gate
≈0
0
h
Forget Gate
Cell
0
f
≈0
3
0
-1
100
+
≈0
0
f
Input Gate
0
3
0
0
1
-10
g
Block
+
1
0
0
0
3
-1
0
1

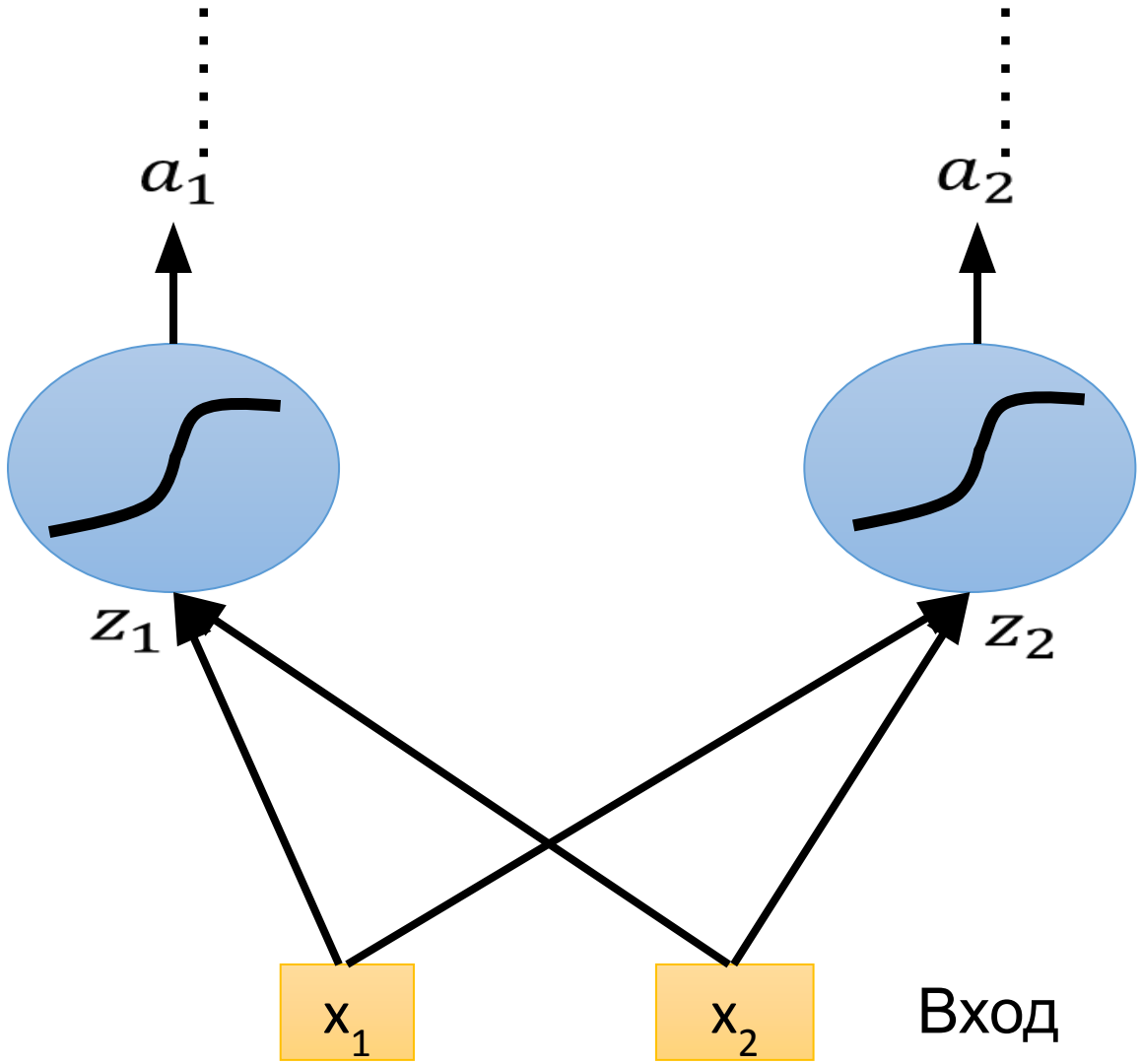

y
0
0
0
7
0

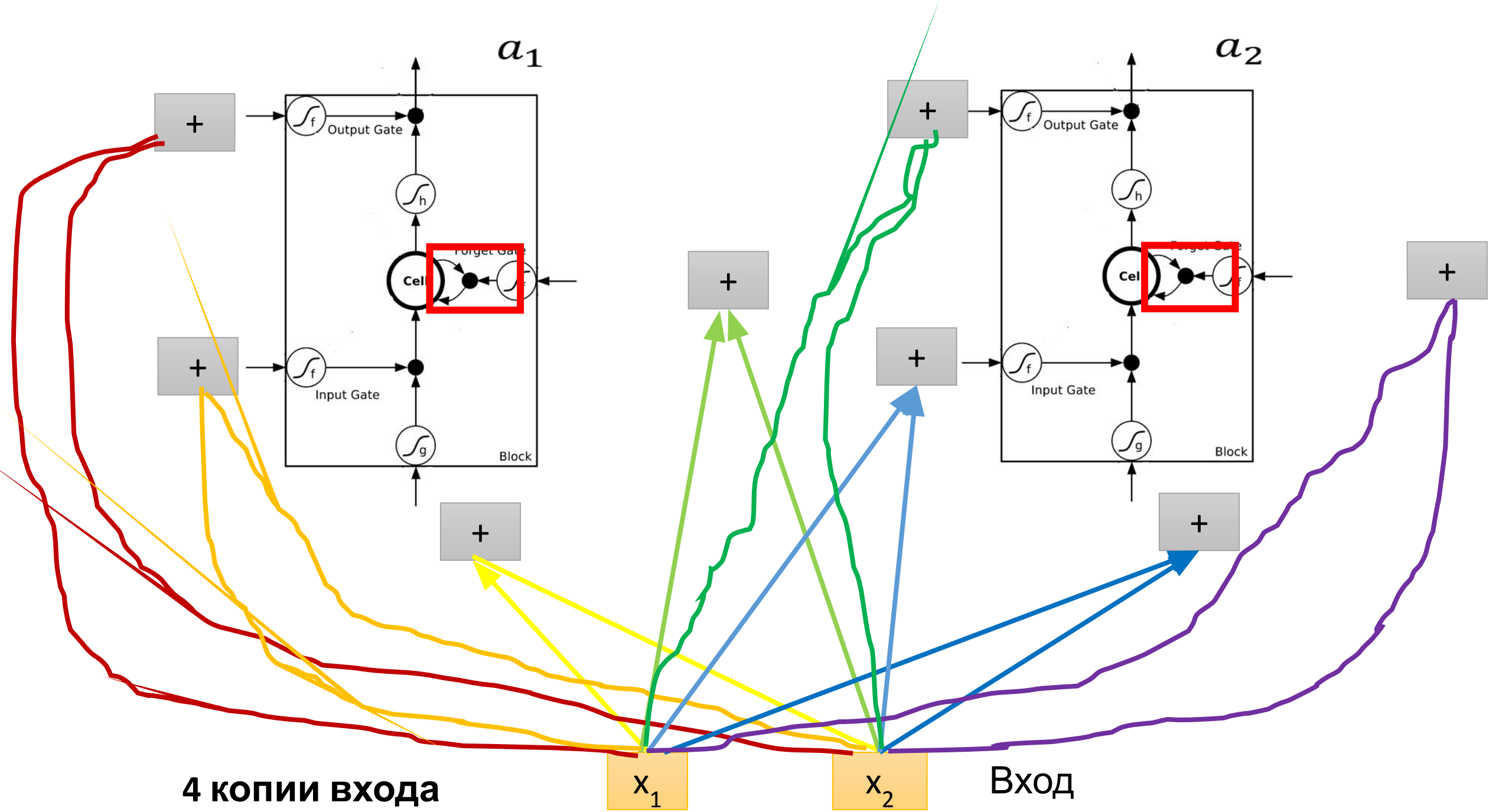

0
3
+
100
-1
0
0
10
1


x_1
x_2
x_3
3
1
0
4
1
0
2
0
0
1
0
1
3
-1
0

Исходная сеть:

 Заменяем обычный нейрон на LSTM

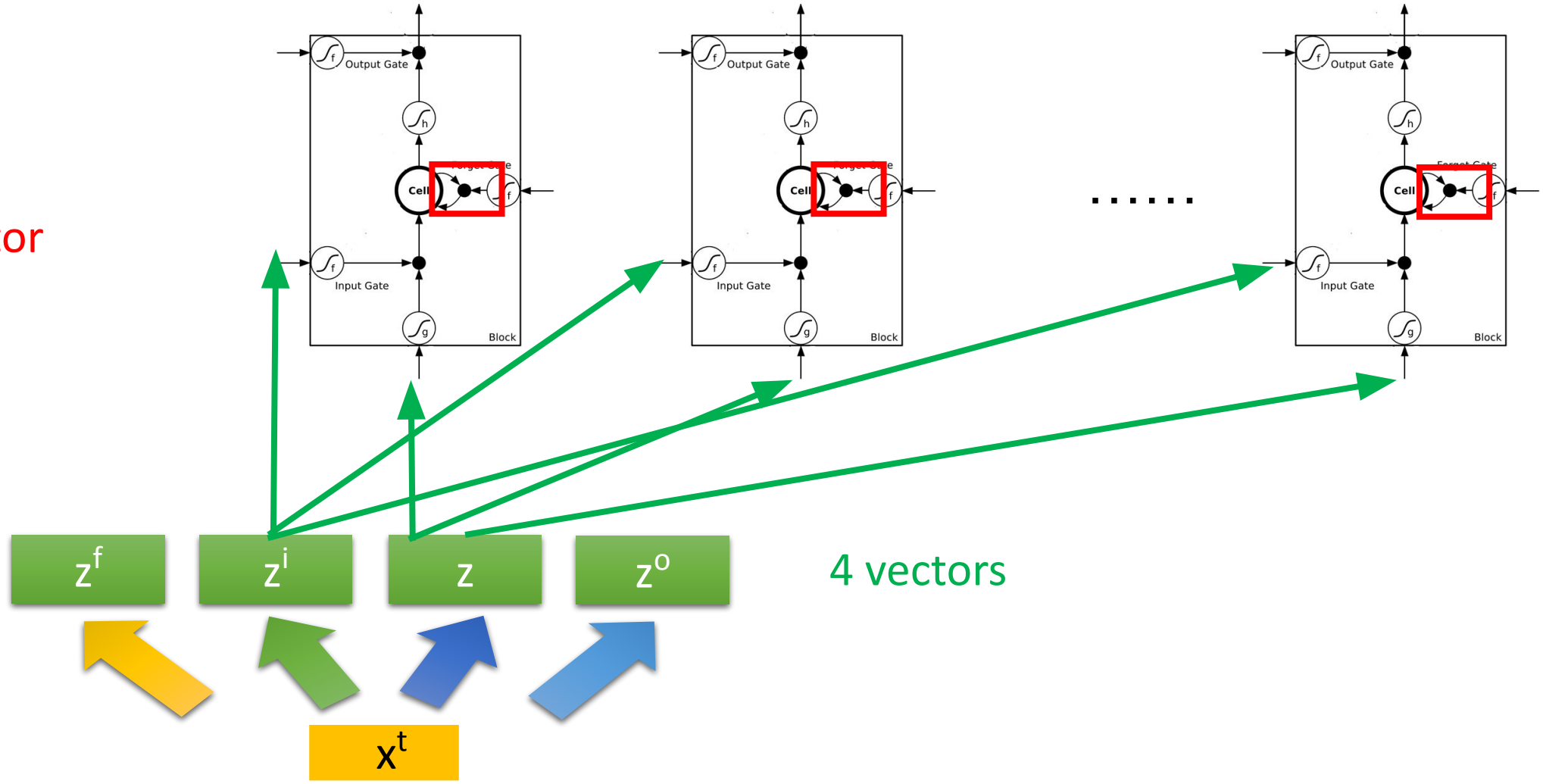



a_1
a_2
Output Gate
Input Gate
Block
Cel
+
x_1
x_2
Вход
4 копии входа

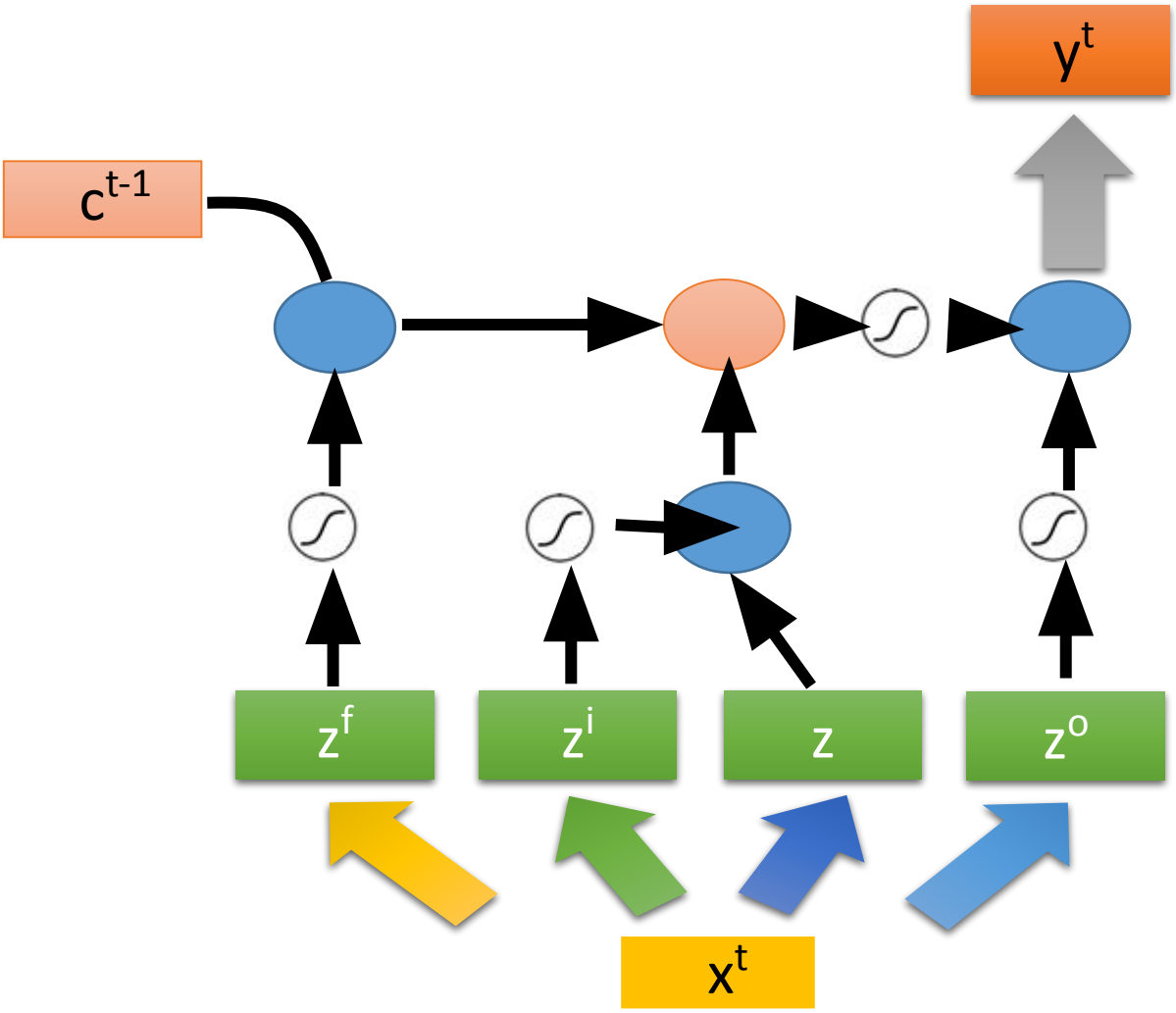
LSTM

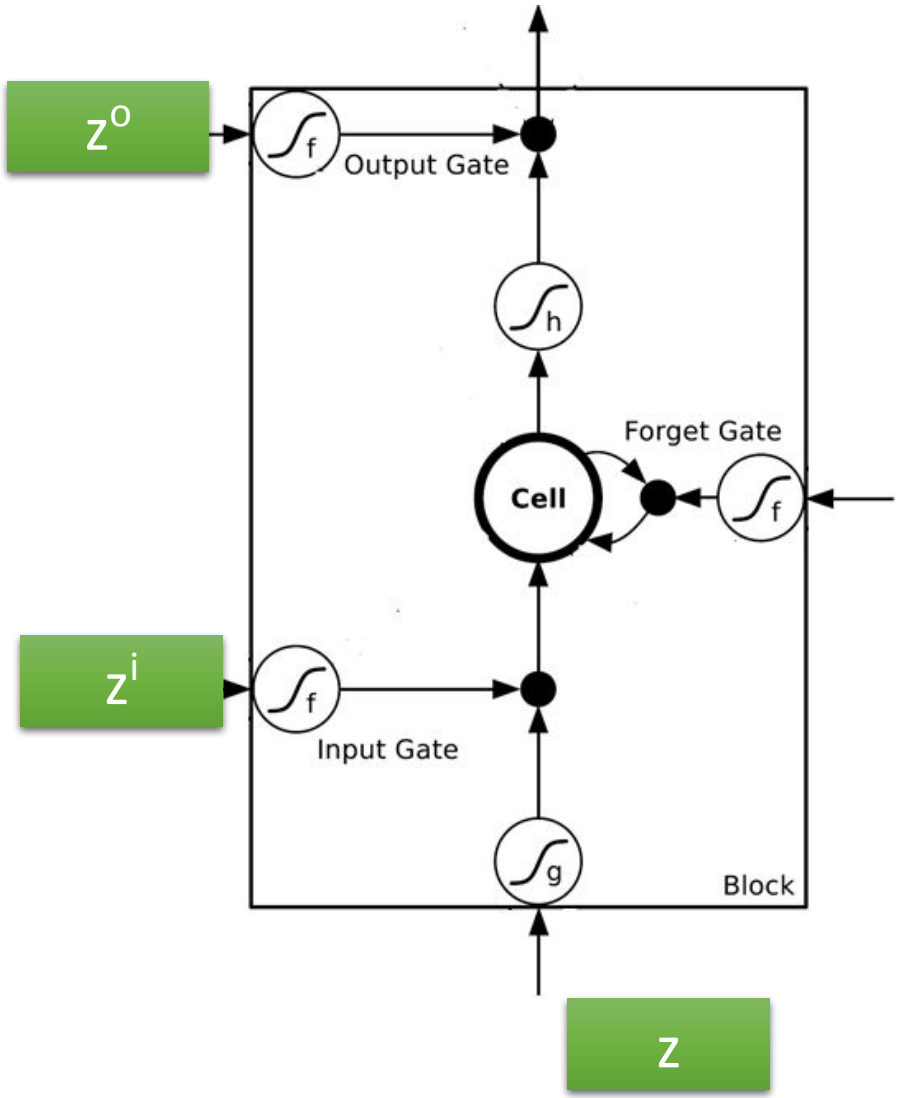
c^{t-1}

vector



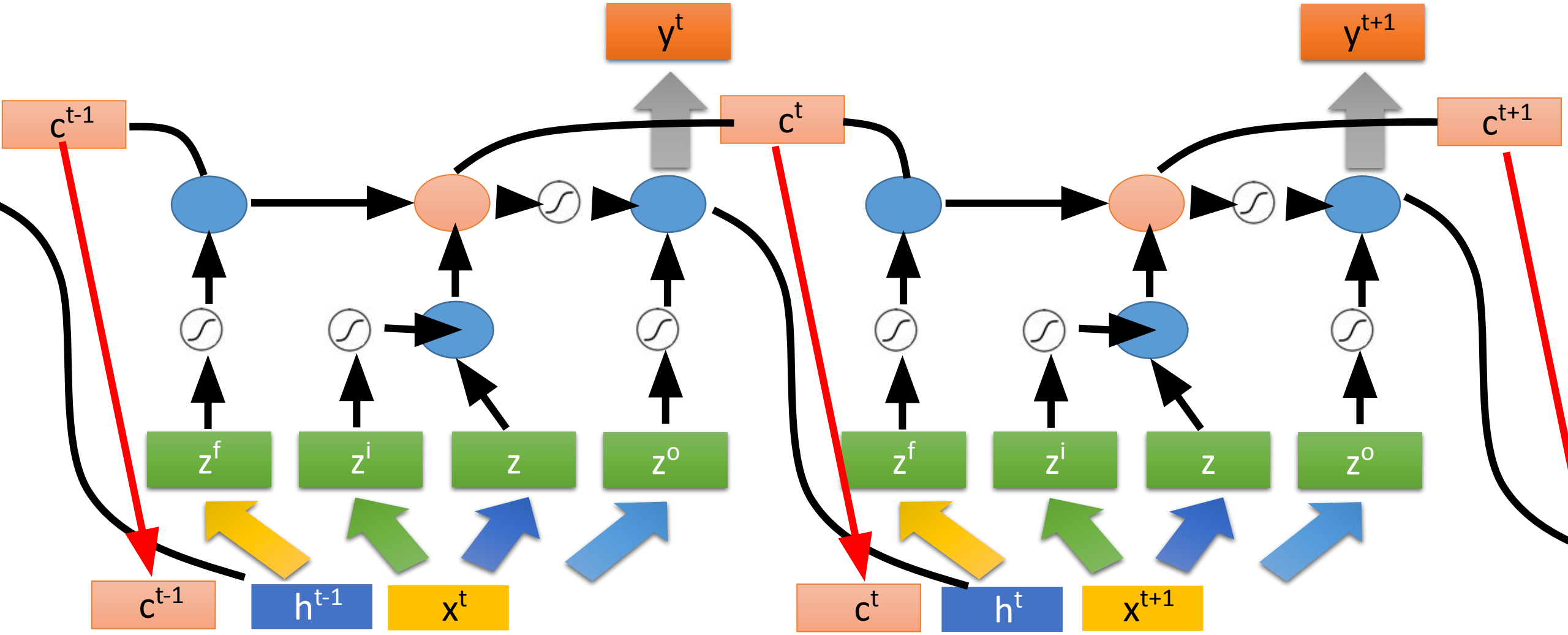
LSTM


y^t
c^{t-1}
z^f
z^i
z
z^o
x^t

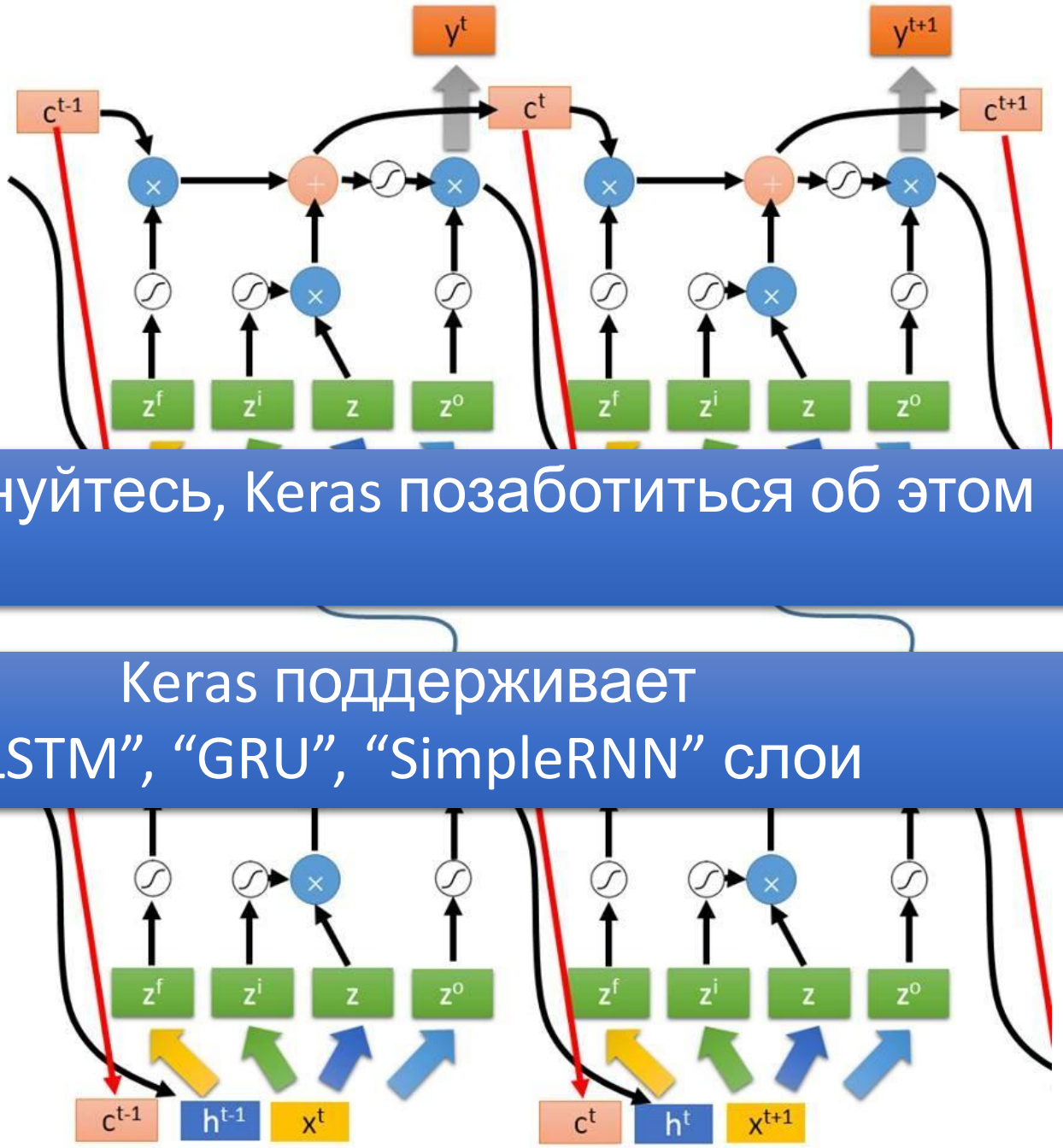

z^o
Output Gate
z^i
Input Gate
Forget Gate
Cell
Block
z^f
z

LSTM

Extension: “peephole”



Multiple-layer LSTM

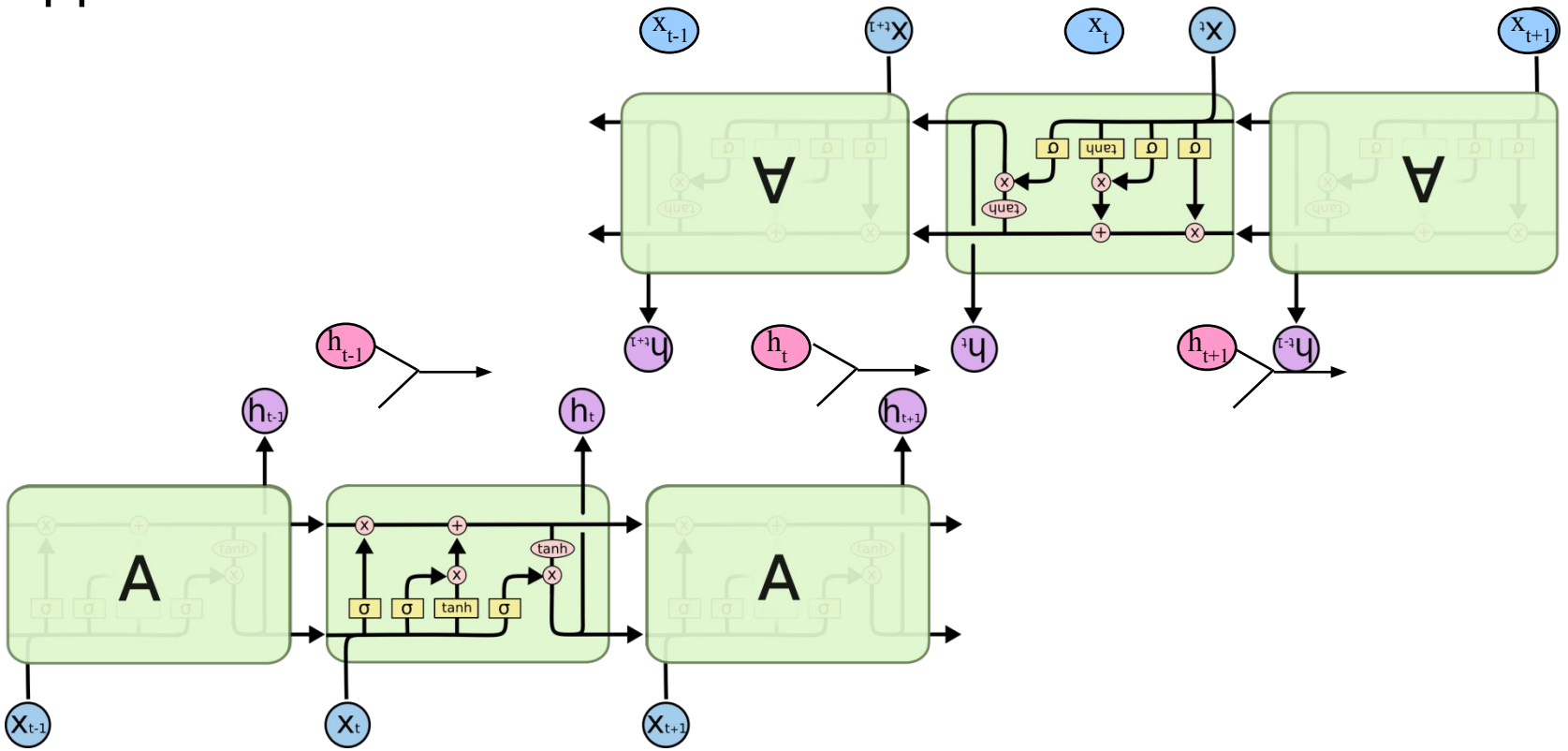


Не волнуйтесь, Keras позаботиться об этом

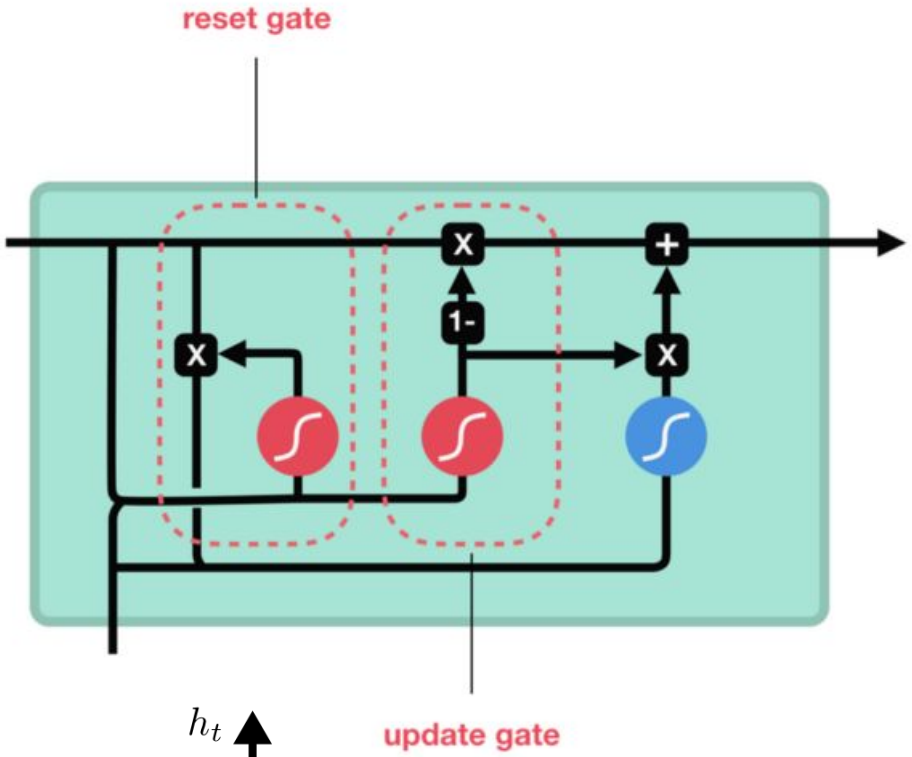
Keras поддерживает
“LSTM”, “GRU”, “SimpleRNN” слои

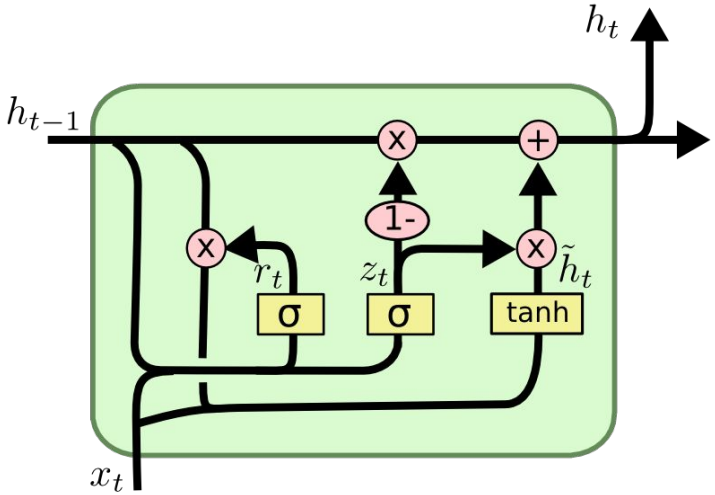
Bi-directional LSTM (Bi-LSTM)

- Отдельные LSTM обрабатывают последовательность вперед и назад, а также скрытые слои на каждом временном шаге объединяются для формирования выходных данных ячейки ».



GRU





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

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

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

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

Update Gate
Шлюз обновления действует аналогично шлюзу забвения и входа LSTM. Он решает, какую информацию выбросить и какую новую добавить.

Reset Gate
Шлюз сброса - это еще один шлюз, который используется, чтобы решить, сколько прошлой информации нужно забыть.
В GRU меньше тензорных операций; поэтому они обучаются немного быстрее, чем LSTM.

Неизвестно, какая из архитектур лучше. Исследователи и инженеры обычно пытаются определить, этот момент эксперементальным путем..

Варианты применения LSTM

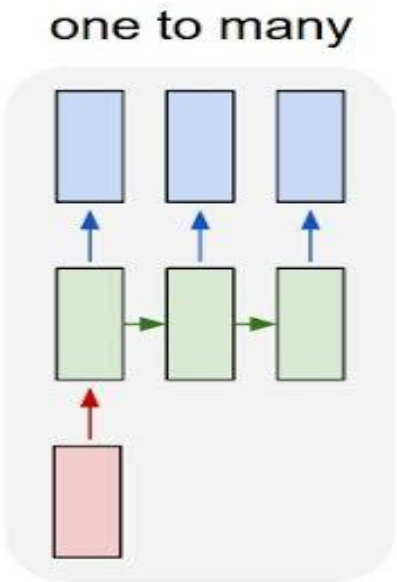
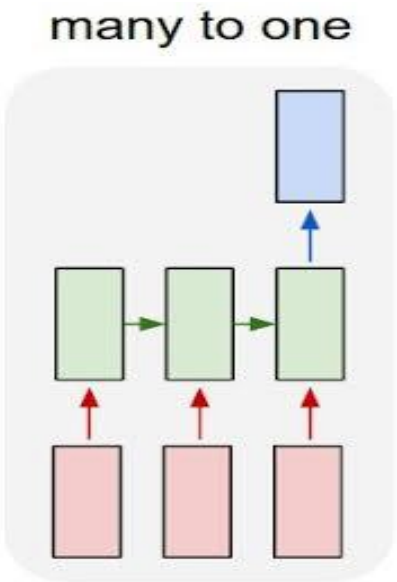
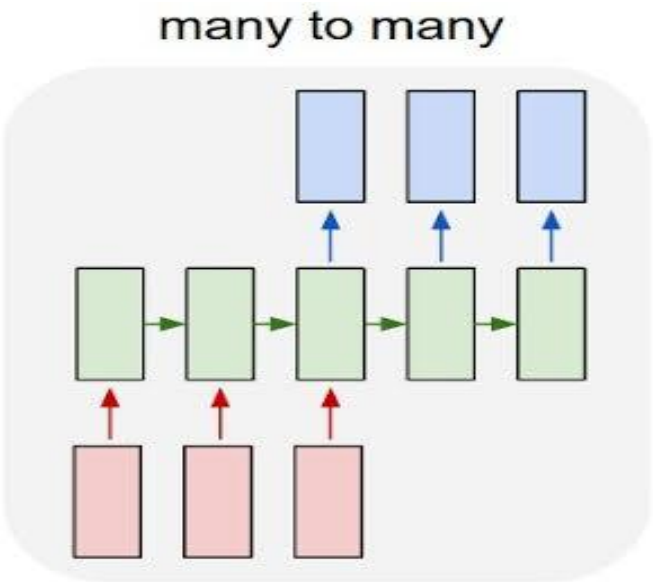


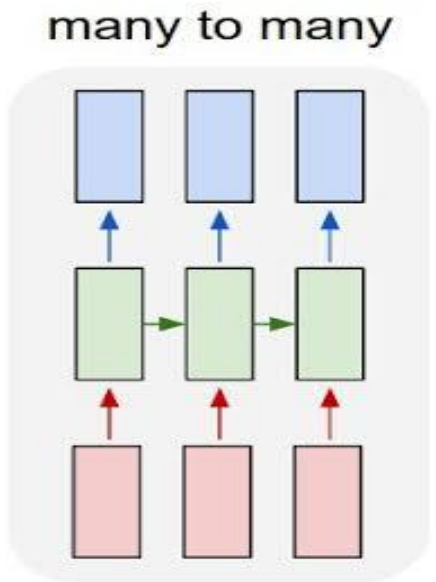
Image Captioning



Video Activity Recog
Text Classification



Video Captioning
Machine Translation



POS Tagging
Language Modeling