DEEP LEARNING

Неделя 3

Святослав Елизаров, Борис Коваленко, Артем Грачев

28 октября 2017

Высшая школа экономики

ОБУЧЕНИЕ НЕЙРОННЫХ СЕТЕЙ

ОБУЧЕНИЕ ЛИНЕЙНОЙ РЕГРЕССИИ

Для всех описанных ранее линейных моделей функция потерь была выпуклой. Выпуклая функция обладает множеством замечательных свойств, наиболее важными из которых для нас являются:

1. Функция непрерывна и дифференцируема на всём интервале за исключением не более чем счётного множества точек и дважды дифференцируема почти всюду.
2. Локальный минимум является глобальным.

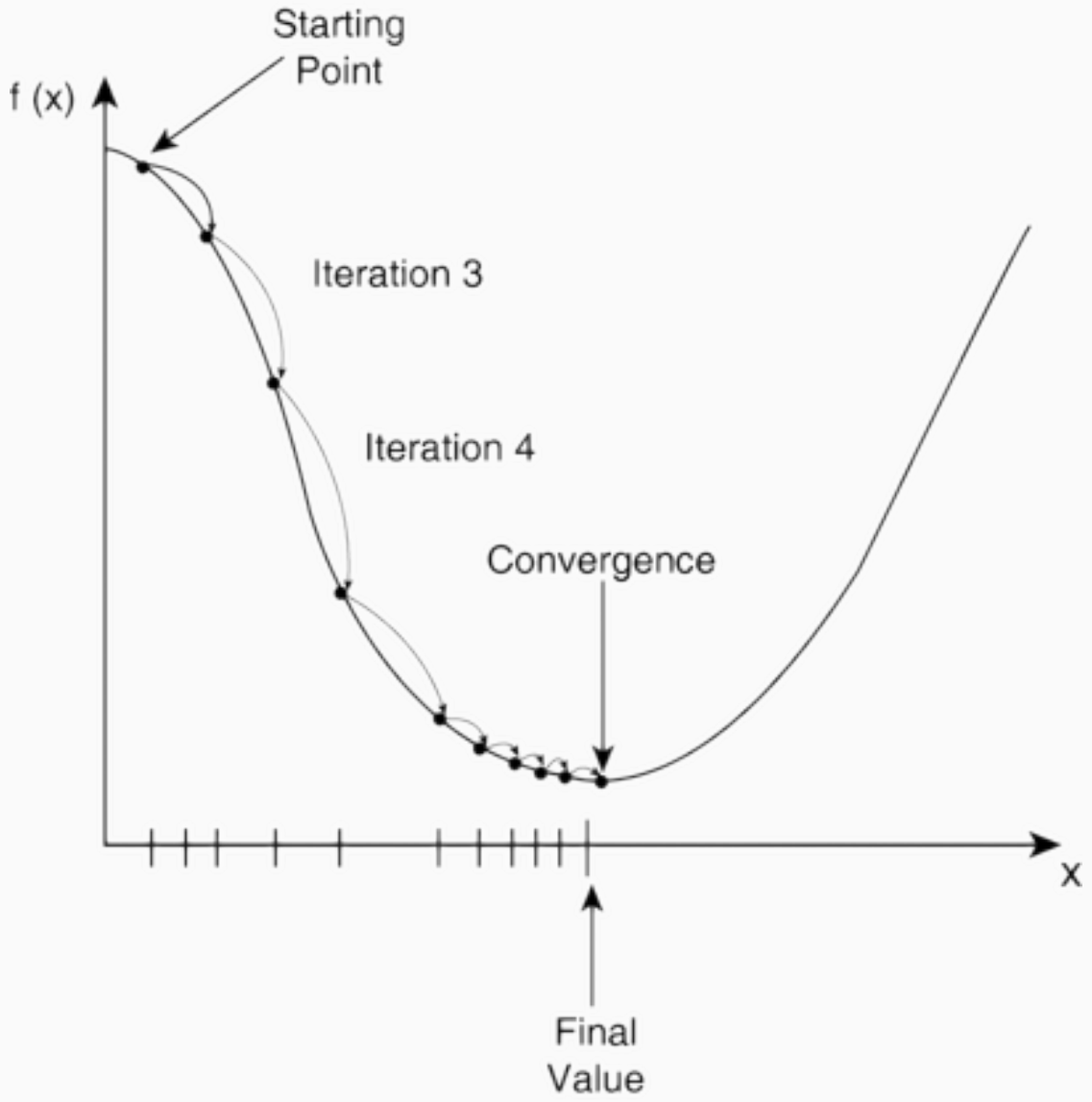
Таким образом мы можем применять основанные на вычислении градиента методы, не боясь застрять в локальном минимуме. Так же важным является то, что мы может вычислить Гессиан, если необходимо.

ГРАДИЕНТНЫЙ СПУСК

Градиент – обобщение производной на многомерный случай. Это вектор, показывающий направления роста функции и по модулю равный скорости роста. Обозначается $\nabla f(x)$.

Градиентный спуск – простейший метод численной оптимизации: суть метода в последовательном движении в направлении противоположном градиенту.

ГРАДИЕНТНЫЙ СПУСК



ГРАДИЕНТНЫЙ СПУСК

$$\theta_{n+1} = \theta_n - \lambda \nabla f(\theta)$$

Где θ_n – вектор параметров функции f на итерации n.

λ – learning rate, может быть как константой, так и функцией от номера итерации.

НЕВЫПУКЛЫЙ СЛУЧАЙ

Для искусственных нейронных сетей в общем виде требование выпуклости функции потерь не соблюдается (почему?). При обучения сети нам предстоит столкнуться со следующими проблемами:

1. Наличе множества локальных минимумов
2. Множество глобальных минимумов
3. Седловые точки
4. Миллионы параметров

Долгое время эти причины не позволяли использовать модели на основе искусственных нейронных сетей на практике.

НАЛИЧЕ МНОЖЕСТВА ЛОКАЛЬНЫХ МИНИМУМОВ

В статье Kenji Kawaguchi, 2016. Deep Learning without Poor Local Minima доказано, что для линейной глубокой сети (три и более слоёв) справедливы следующие утверждения:

- Каждый локальный минимум является глобальным
- Каждая критическая точка, не являющаяся глобальным минимумом, является седловой точкой

НАЛИЧЕ МНОЖЕСТВА ЛОКАЛЬНЫХ МИНИМУМОВ

В статье Kenji Kawaguchi, 2016. Deep Learning without Poor Local Minima доказано, что для линейной глубокой сети (три и более слоёв) справедливы следующие утверждения:

- Каждый локальный минимум является глобальным
- Каждая критическая точка, не являющаяся глобальным минимумом, является седловой точкой

Эти же утверждения справедливы для сетей с активацией типа Relu, в предположении что карты активации не зависят от входа (нереалистичное предположение)

НАЛИЧЕ МНОЖЕСТВА ЛОКАЛЬНЫХ МИНИМУМОВ

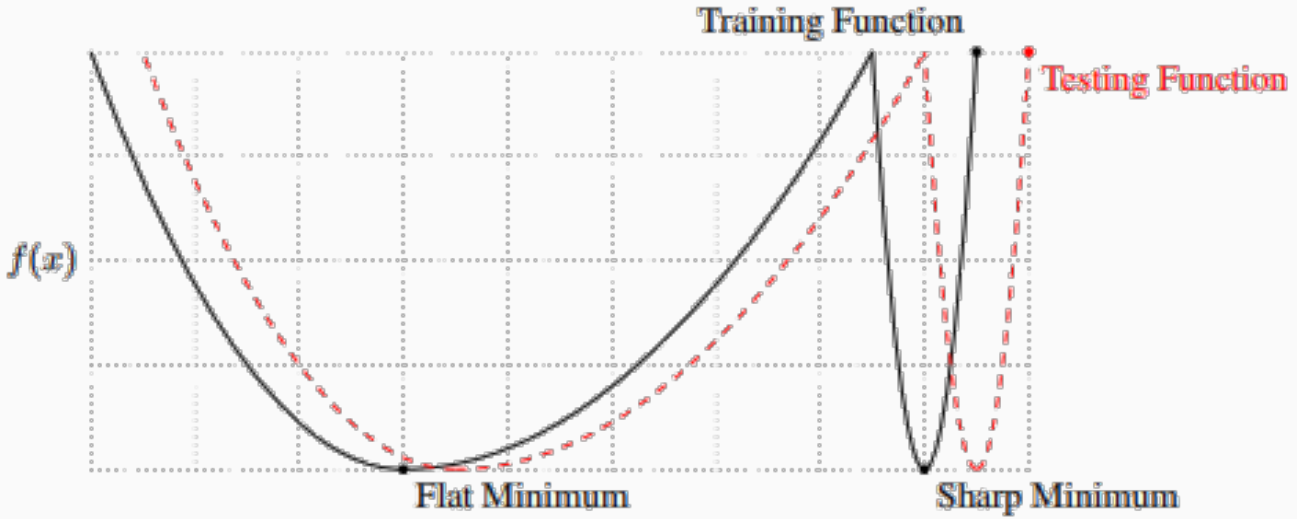
В статье LeCun et al., 2014. The Loss Surfaces of Multilayer Networks проводится исследовние ландшафта функции потерь и показывается, что вероятность попадания в плохой локальный минимум убывает с увеличением количества слоёв.

НАЛИЧЕ МНОЖЕСТВА ЛОКАЛЬНЫХ МИНИМУМОВ

	Training Accuracy		Testing Accuracy	
Name	SB	LB	SB	LB
F_1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F_2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C_1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C_2	$99.99\% \pm 0.04\%$	$99.99\% \pm 0.01\%$	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C_3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.30\%$	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C_4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$

НАЛИЧЕ МНОЖЕСТВА ЛОКАЛЬНЫХ МИНИМУМОВ

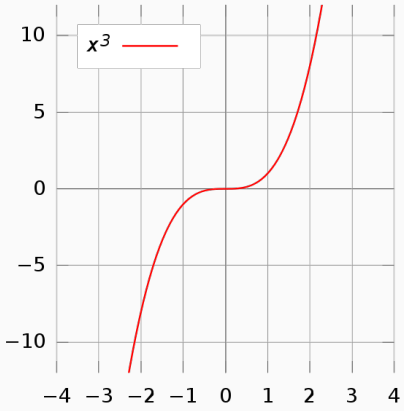
1. В статье Hochreiter, S., Schmidhuber, J. (1997). Flat minima. Neural Computation проводится исследование ландшафта функции потерь глубокой нейронной сети и вводятся понятия плоского и острого минимумов.
2. Исследование Keskar et al. (2016). On large-batch training for deep learning: Generalization gap and sharp minima показывает взаимосвязь между размером минибатча и вероятностью попасть в sharp minimum
3. В работе L Dinh et al. (2017) Sharp Minima Can Generalize For Deep Nets вводится строгое определение для двух типов минимумов и показано, что в некоторых случаях сети с Relu-активациями не теряют обобщающей способности даже при попадании в sharp minimum


Training Function
Testing Function
$f(x)$
Flat Minimum
Sharp Minimum

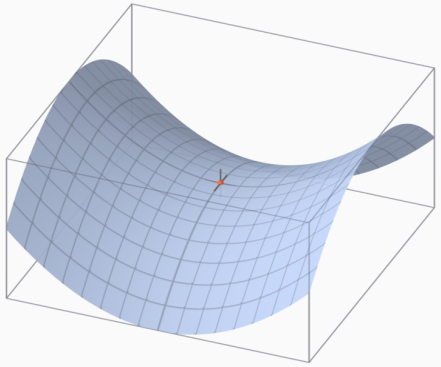
СЕДЛОВЫЕ ТОЧКИ

Седловой называется такая критическая точка функции, которая не является её экстремумом. Достаточное (но не необходимое!) условие того, что точка седловая: Гессиан в этой точке является неопределённой квадратичной формой.

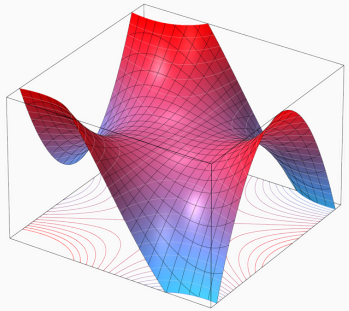
СЕДЛОВЫЕ ТОЧКИ



(a) $f(x) = x^3$



(b) $f(x) = x_1^2 - x_2^2$



(c) $f(x) = x_1^3 - 3x_1x_2^2$

СЕДЛОВЫЕ ТОЧКИ

Идеи?

СЕДЛОВЫЕ ТОЧКИ

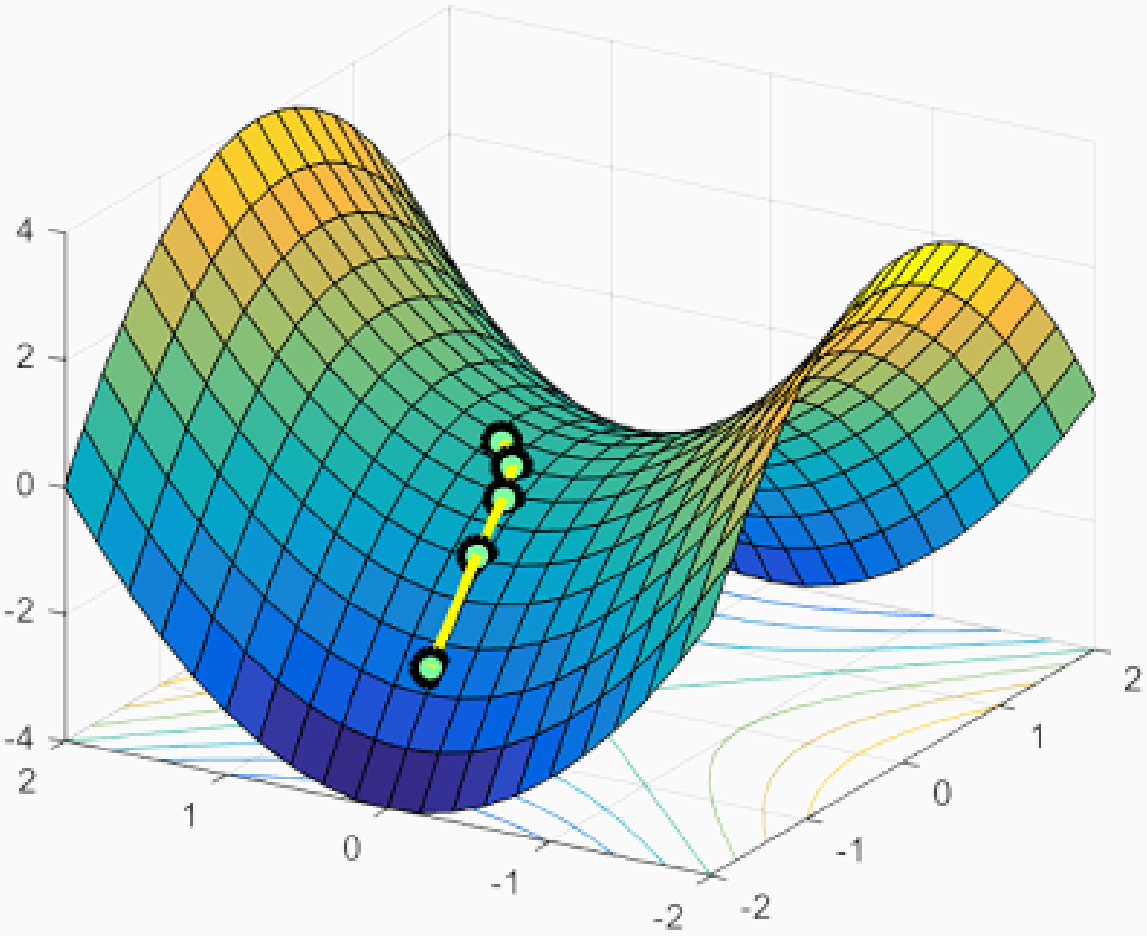
Добавим шум к градиенту. Если представить шарик, который катится по поверхности, то велика вероятность, что он скатится с седловой точки, если будет двигаться с небольшими случайными флуктуациями.

$$\theta_{n+1} = \theta_n - \lambda \nabla f(\theta) + \epsilon$$

Где $\epsilon \sim N(0, 1)$, т.е. Гауссовский шум.

Rong Ge et al., 2015. Escaping From Saddle Points – Online Stochastic Gradient for Tensor Decomposition

СЕДЛОВЫЕ ТОЧКИ



МЕТОДЫ ОПТИМИЗАЦИИ

МИЛЛИОНЫ ПАРАМЕТРОВ

- Модели на основе искусственных нейронных сетей могут иметь сотри тысяч или даже миллионы параметров.
- Чтобы обучить такую модель потребуется существенный объём данных.

Существуют теоретические оценки, но они сильно завышены и не могут быть применены на практике.

VANILLA SGD

Ищем локальный экстремум, идем вдоль градиента

$$W_{i+1} = W_i - \lambda \nabla L(W_i, X, y)$$

При большом наборе обучающих данных алгоритм будет работать крайне медленно. Более того, данные могут просто не поместиться в память.

На практике производят корректировку коэффициентов сети с использованием градиента, который аппроксимируются градиентом функции потерь, вычисленной только на случайном подмножестве обучающей выборки (mini batch).

VANILLA SGD

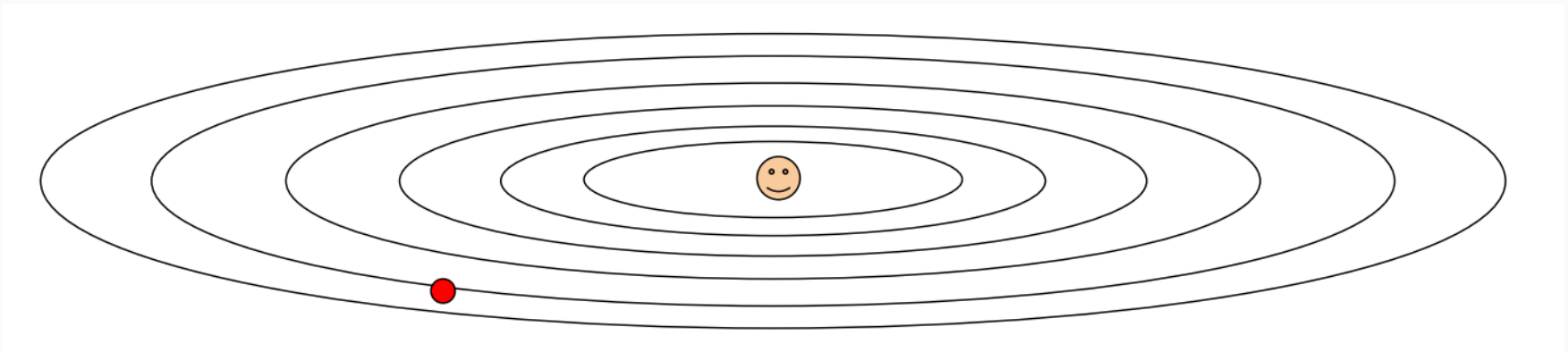
Количество объектов для вычисления градиента выбирается исходя из объема памяти который имеется (максимально заполняем память) или выбирается с помощью кросс-валидации.

$$L(W) = \frac{1}{N} \sum_{i}^{N} L_i(W, x_i, y_i)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i}^{N} \nabla_W L_i(W, x_i, y_i)$$

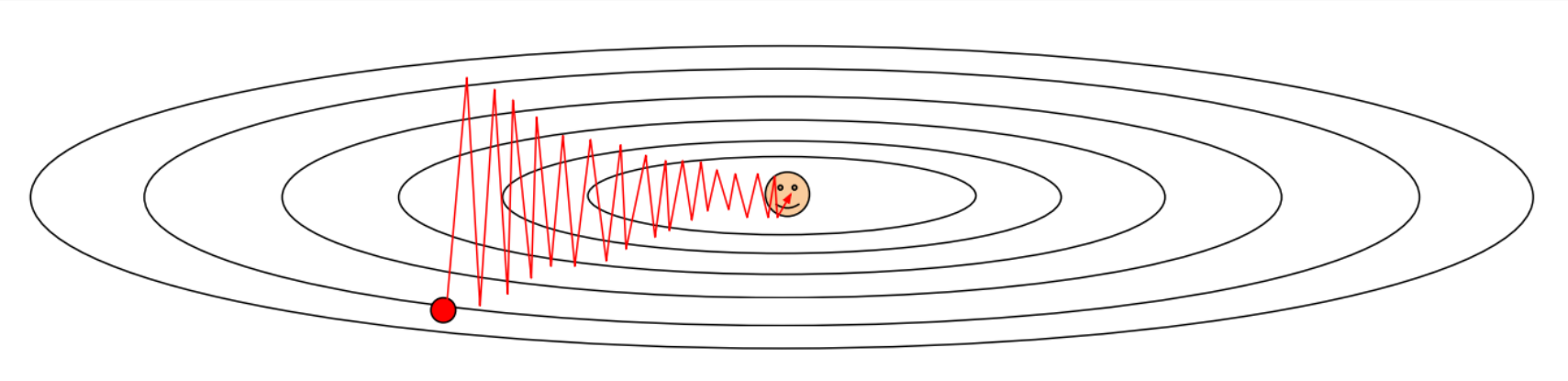
VANILLA SGD

Что будет с Vanilla SGD если линии уровня функции потерь выглядят так:



VANILLA SGD

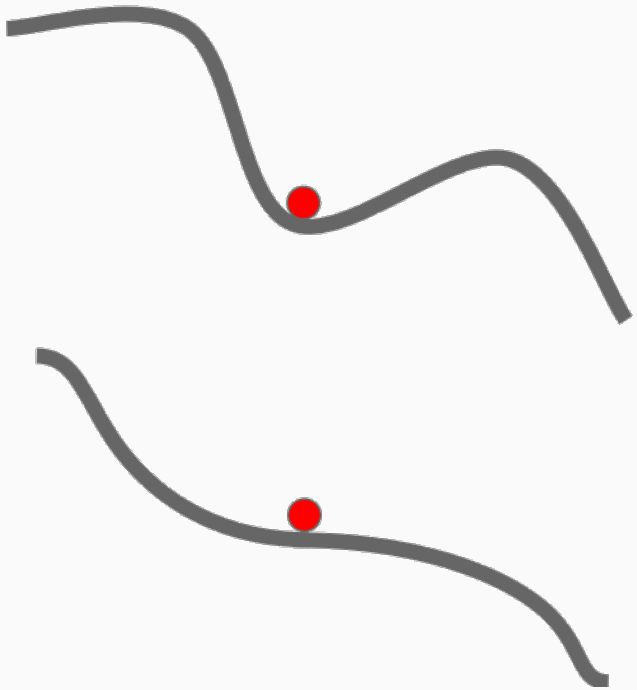
Что будет с Vanilla SGD если линии уровня функции потерь выглядят так:



Как можно улучшить алгоритм?

VANILLA SGD

Не забываем о локальных минимумах и седловых точках!



Как можно улучшить алгоритм?

ИМПУЛЬСНЫЙ МЕТОД

Если в случае градиентного спуска мы представляли человека, спускающегося с высокой горы, то в случае импульсного метода с горы скатывается тяжелый железный шар. На направление и скорость движения шара влияет не только тот рельеф, который он преодолевает в данный момент, но и его предыдущее состояние.

SGD:

$$W_{t+1} = W_t - \lambda \nabla L(W_t, ...)$$

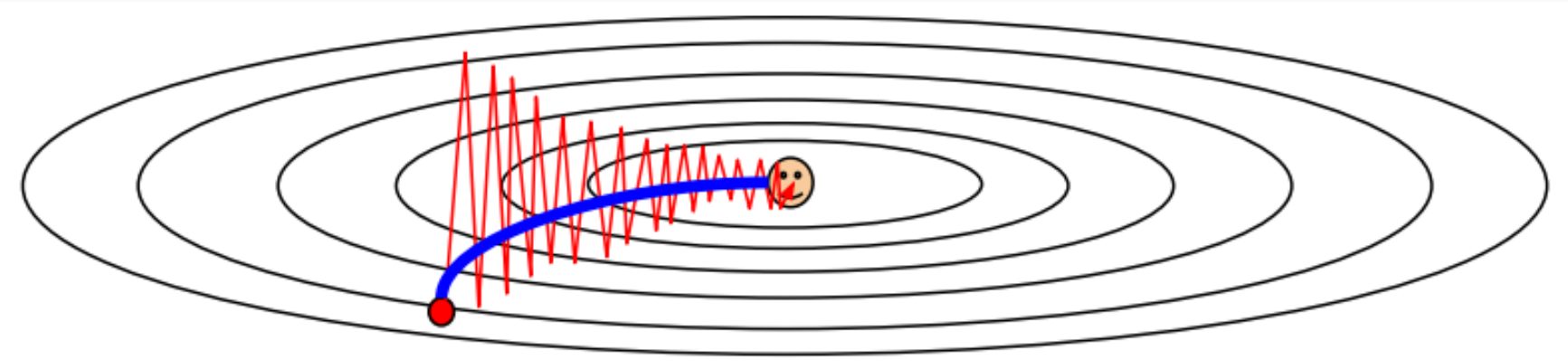
Импульсный метод

$$v_{t+1} = \rho v_t + \nabla L(W_t, ...)$$
$$W_{t+1} = W_t - \alpha v_{t+1}$$

ИМПУЛЬСНЫЙ МЕТОД

Метод даёт существенный прирост в скорости сходимости. Более подробно пр метод можно прочесть в онлайн-журнале distill.pub: Gabriel Goh 2017. Why Momentum Really Works

SGD + MOMENTUM



ADAGRAD

$$Cache_{t+1} = Cache_t + \nabla L(W_t, ...)$$

$$W_{t+1} = W_t - \frac{\lambda \nabla L(W_t, ...)}{\sqrt{Cache_{t+1} + 1e^{-10}}}$$

Маштабирование шага для каждого параметра. Редкие признаки получаю больше внимания при оптимизации.

Какие могут быть проблемы? Как их решить?

RMSPROP

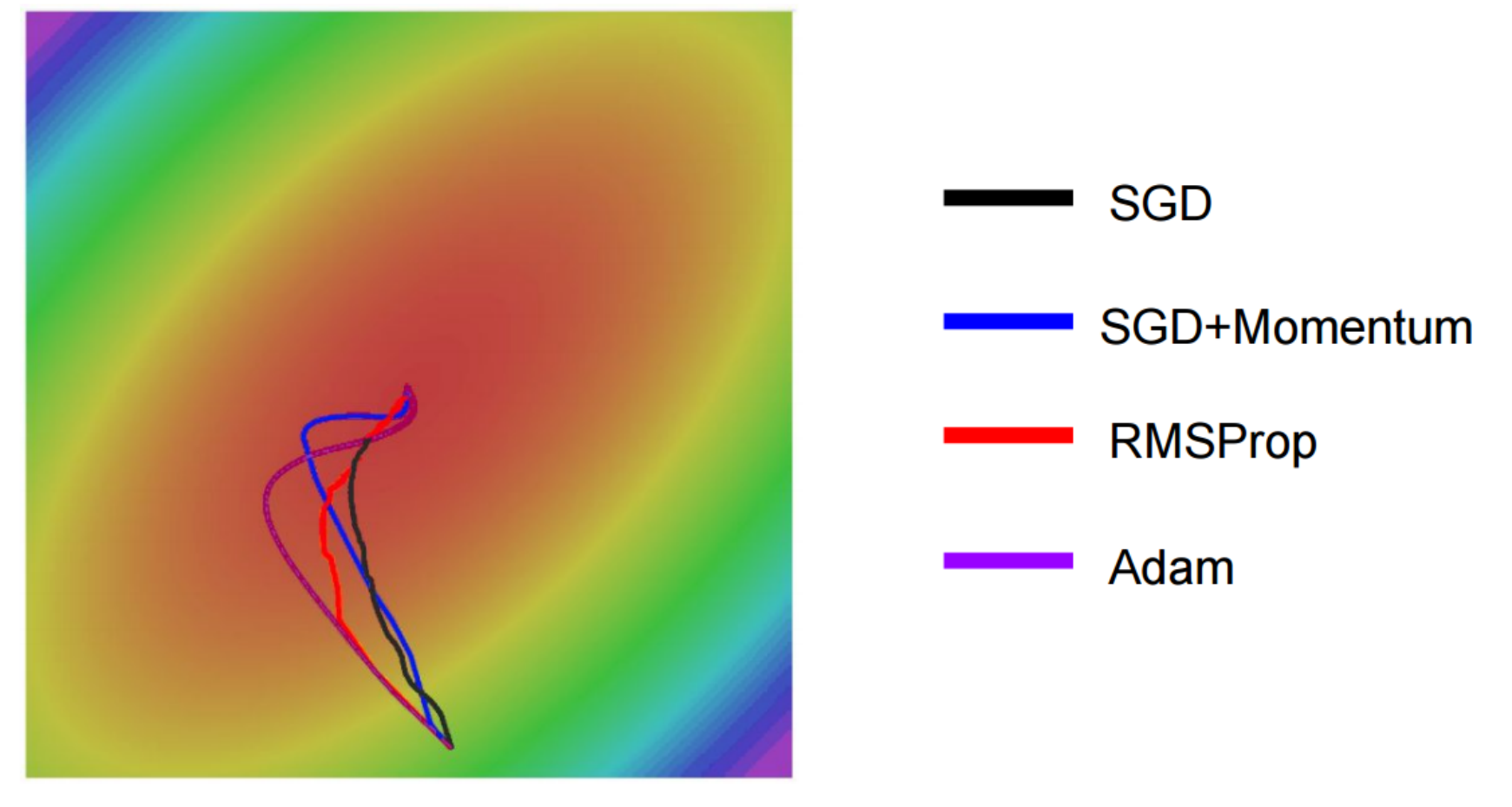
$$Cache_{t+1} = \gamma Cache_t + (1 - \gamma)\nabla L(W_t, ...)$$

$$W_{t+1} = W_t - \frac{\lambda \nabla L(W_t, ...)}{\sqrt{Cache_{t+1} + 1e^{-10}}}$$

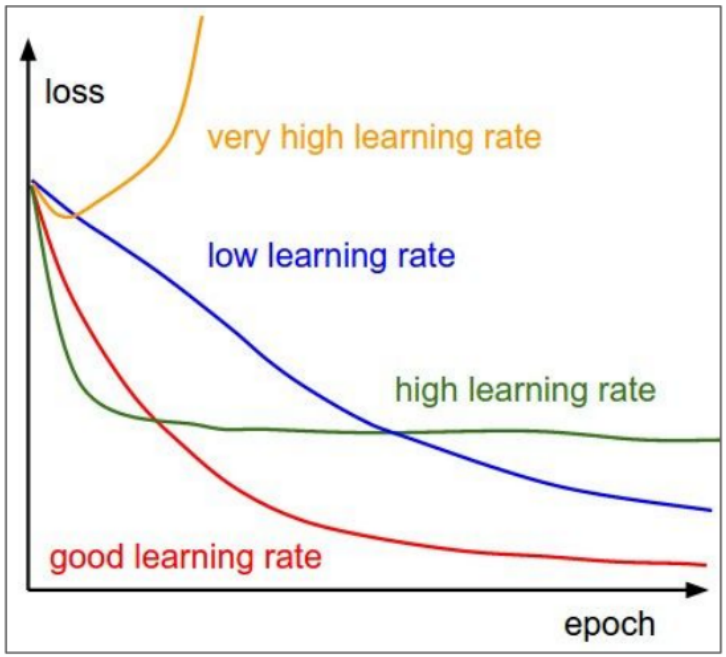
γ - Параметр затухания для истории градиентов, учитываем только окно недавних градиентов

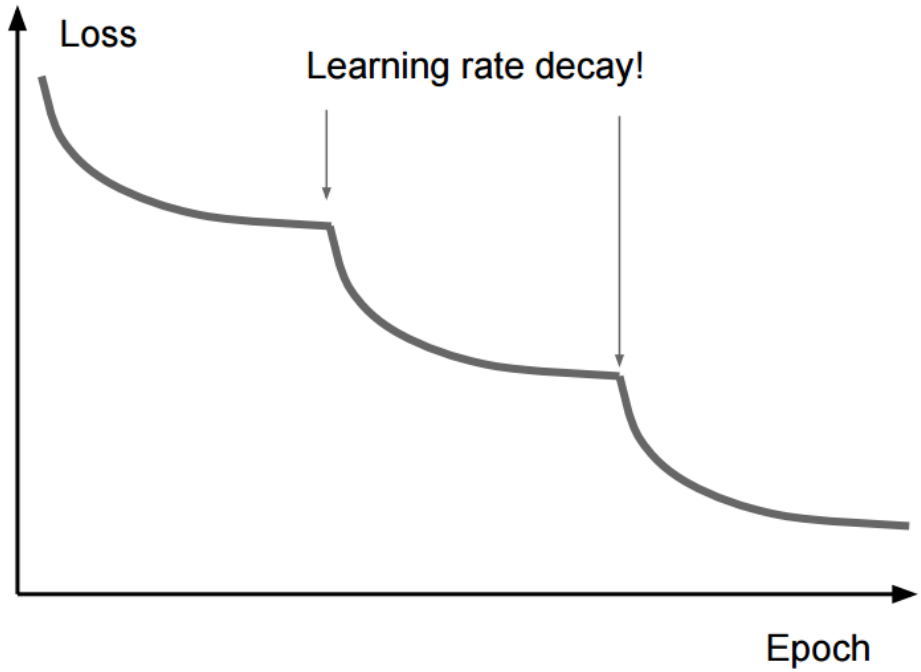
$$Adam = RMSProp + Momentum$$

RMSPROP



RMSPROP


loss
very high learning rate
low learning rate
high learning rate
good learning rate
epoch


Loss
Learning rate decay!
Epoch

ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

Остаётся решить как мы будем вычислять градиент. Необходимо найти некоторый универсальный способ представления функций, удобный для вычисления частных производных.

Вычислительным графом (computational graph) называется направленный ациклический граф в вершинах которого находятся операции из которых состоит исходная функция. Направление в графе отражает зависимость значений одних вершин от других.

ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

Вычислительные графы позволяют:

- повторно использовать промежуточные результаты
- транслировать описанные функции в реализации на разных языках

Вычислительные графы используются в большинстве современных библиотек для deep learining.

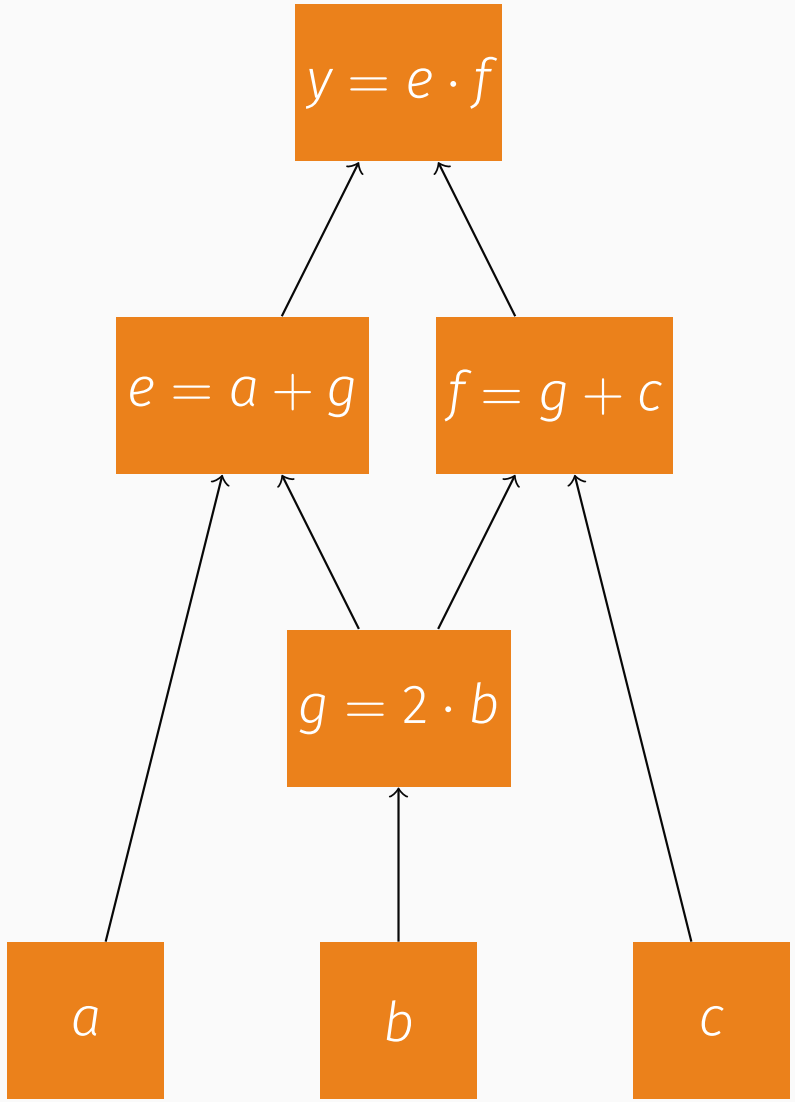
ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

Например возьмём функцию $y = (a + 2b)(2b + c)$

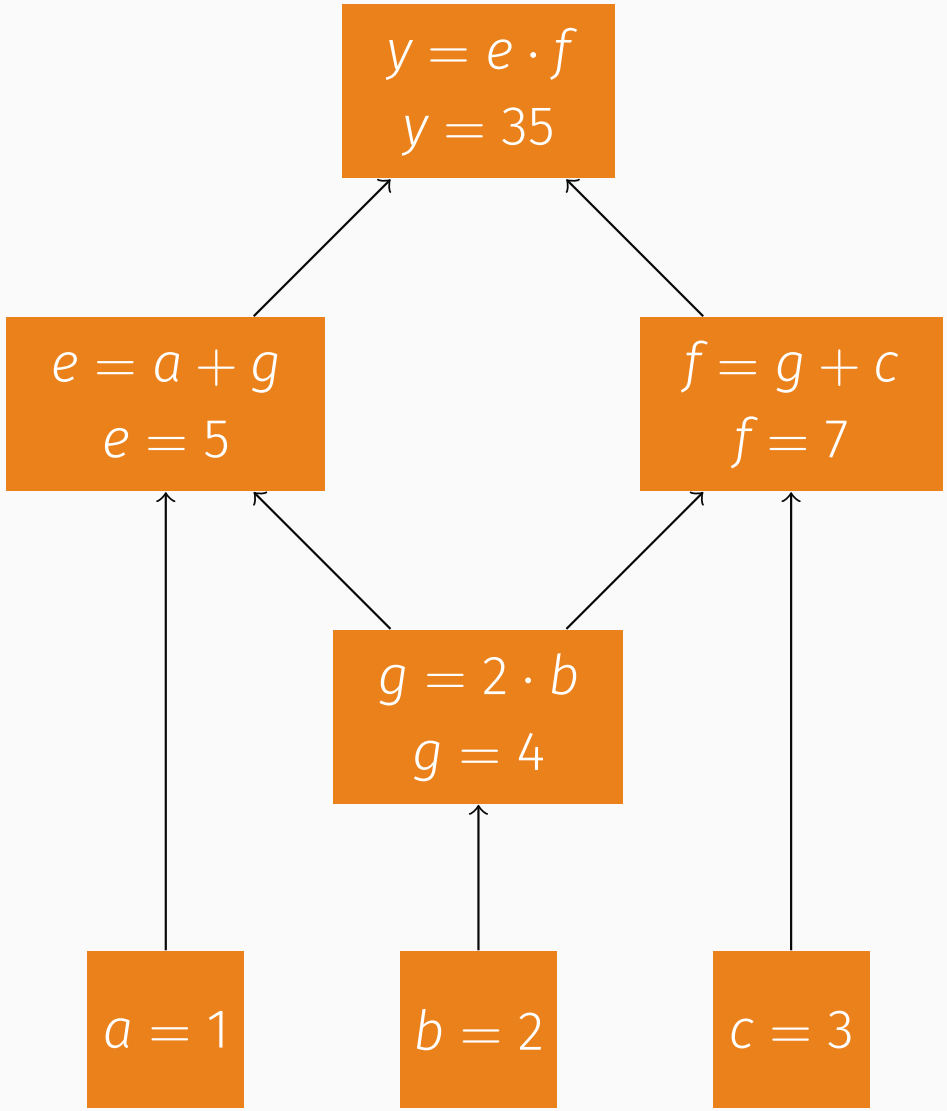
Она состоит из четырёх операций, следовательно в графе будет четыре вершины (и три входа). Выпишем их:

$g = 2 \cdot b$
$e = a + g$
$f = g + c$
$y = e \cdot f$

ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

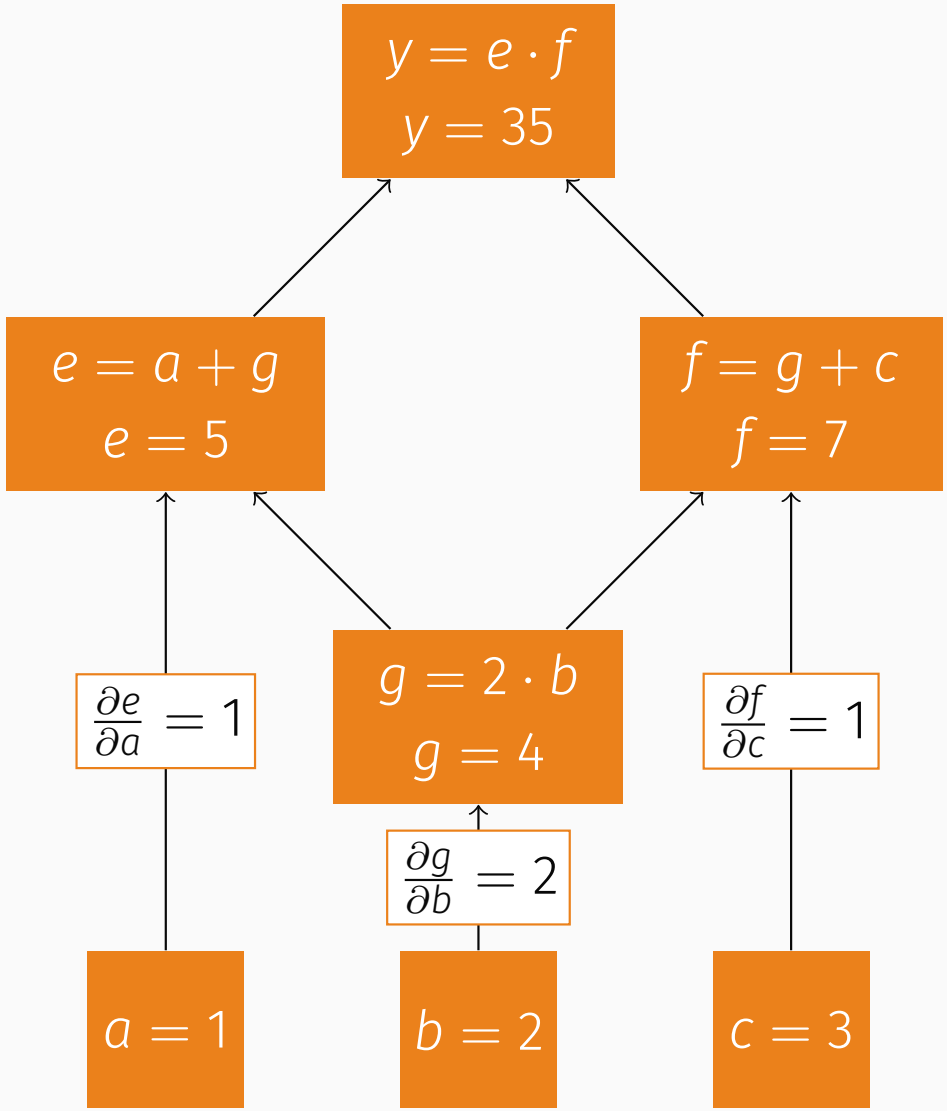


ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

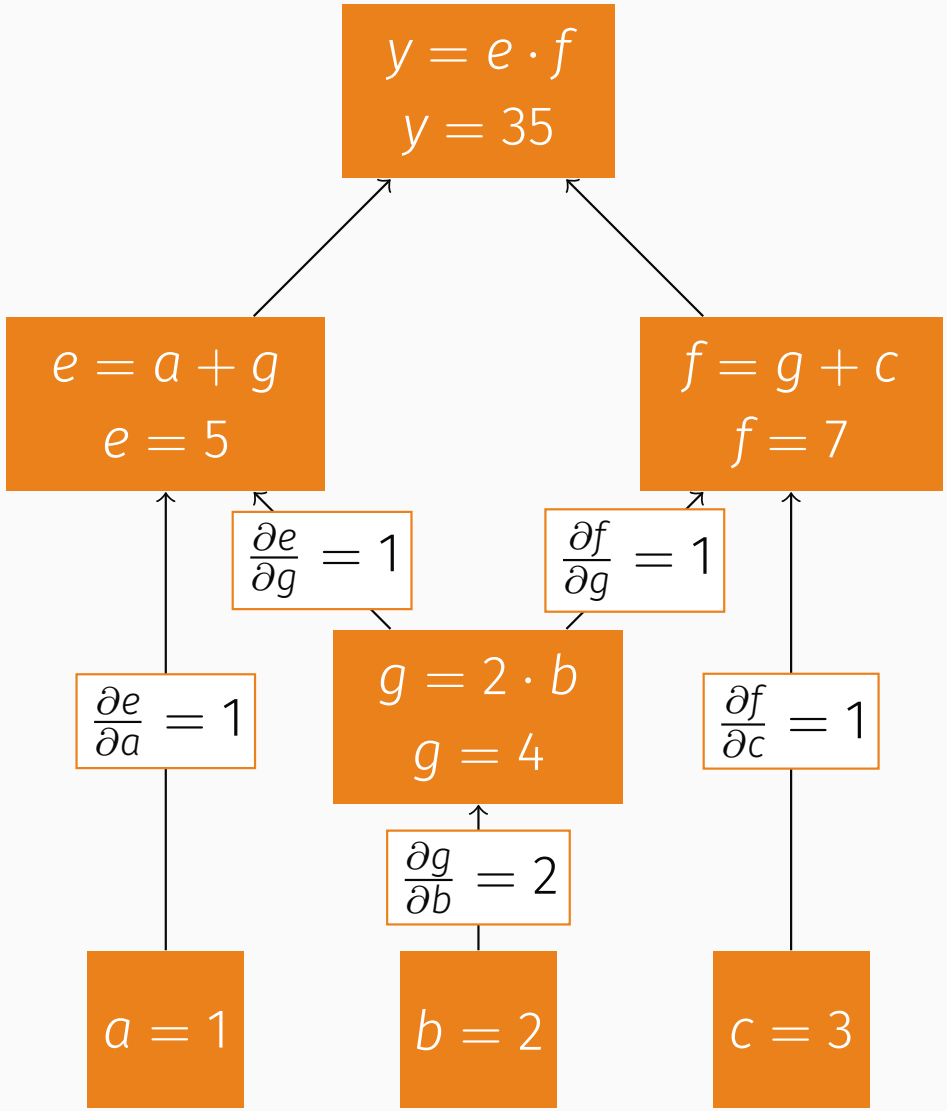
Как видно из примера, значения узла g было рассчитано один раз, но использовалось дважды.

Теперь вычислим градиент функции y.

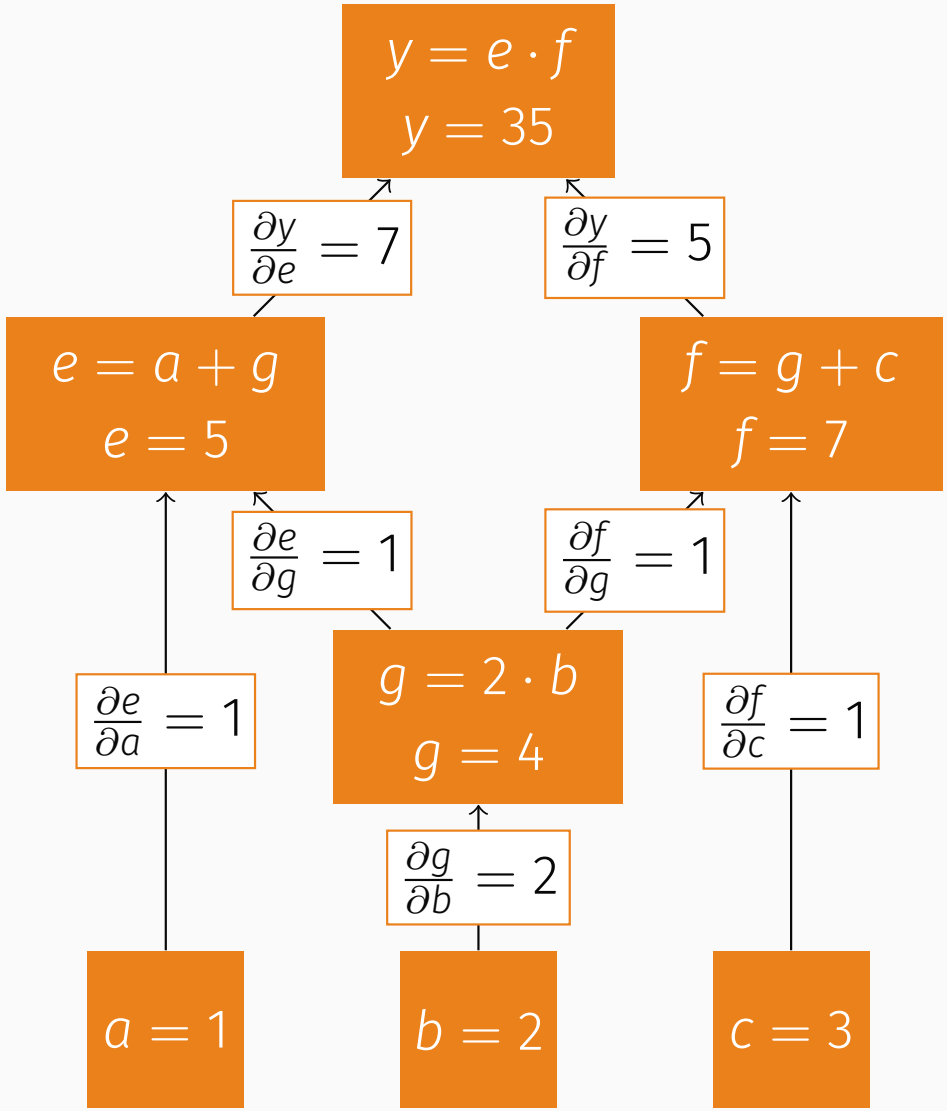
ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

Теперь воспользуемся цепным правилом и вычислим $\frac{\partial y}{\partial a}$, $\frac{\partial y}{\partial b}$ и $\frac{\partial y}{\partial c}$:

$$\frac{\partial y}{\partial a} = \frac{\partial y}{\partial e} \cdot \frac{\partial e}{\partial a} = 7$$

$$\frac{\partial y}{\partial b} = \frac{\partial y}{\partial e} \cdot \frac{\partial e}{\partial g} \cdot \frac{\partial g}{\partial b} + \frac{\partial y}{\partial f} \cdot \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial b} = 24$$

$$\frac{\partial y}{\partial c} = \frac{\partial y}{\partial f} \frac{\partial f}{\partial c} = 5$$

Какова сложность этого алгоритма?

ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

Что с этим делать?

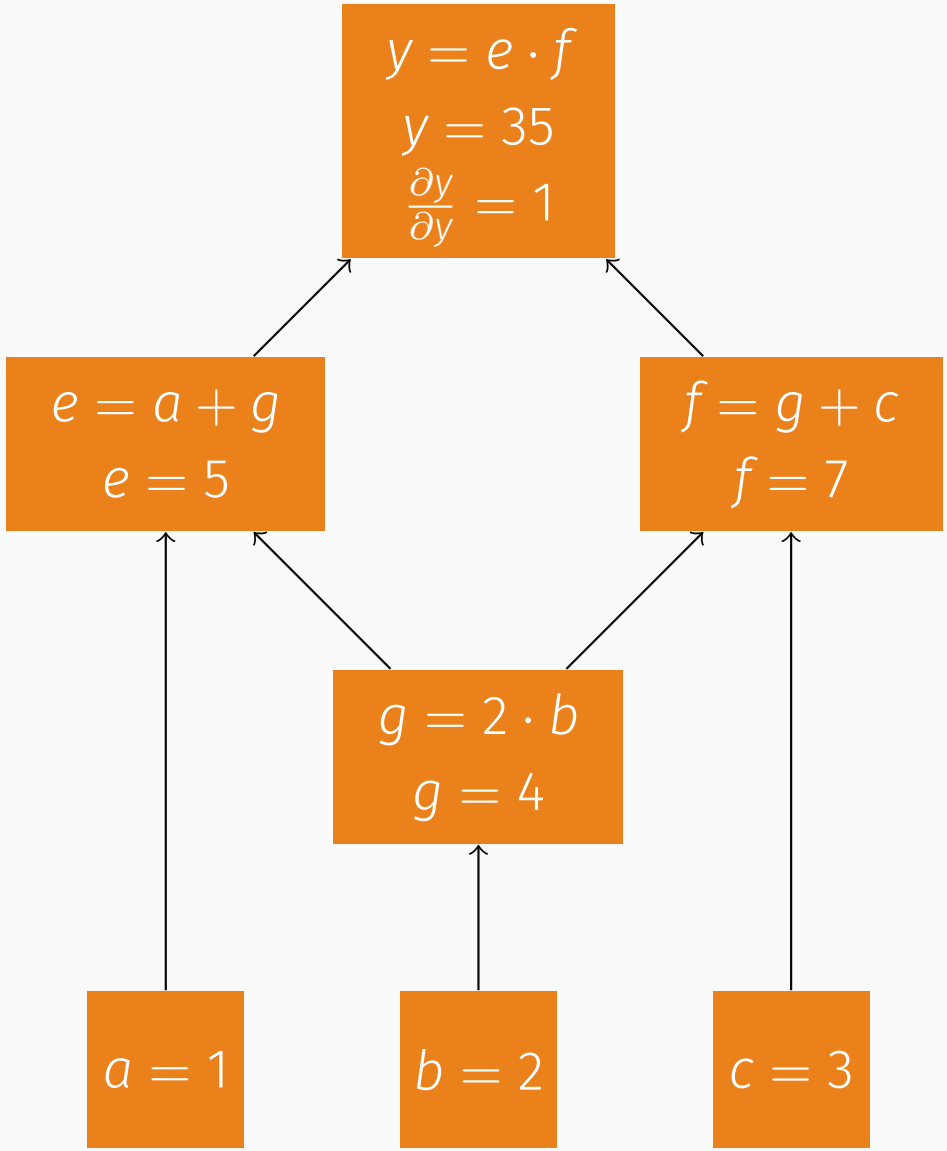
ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

Что с этим делать? Применим динамическое программирование!

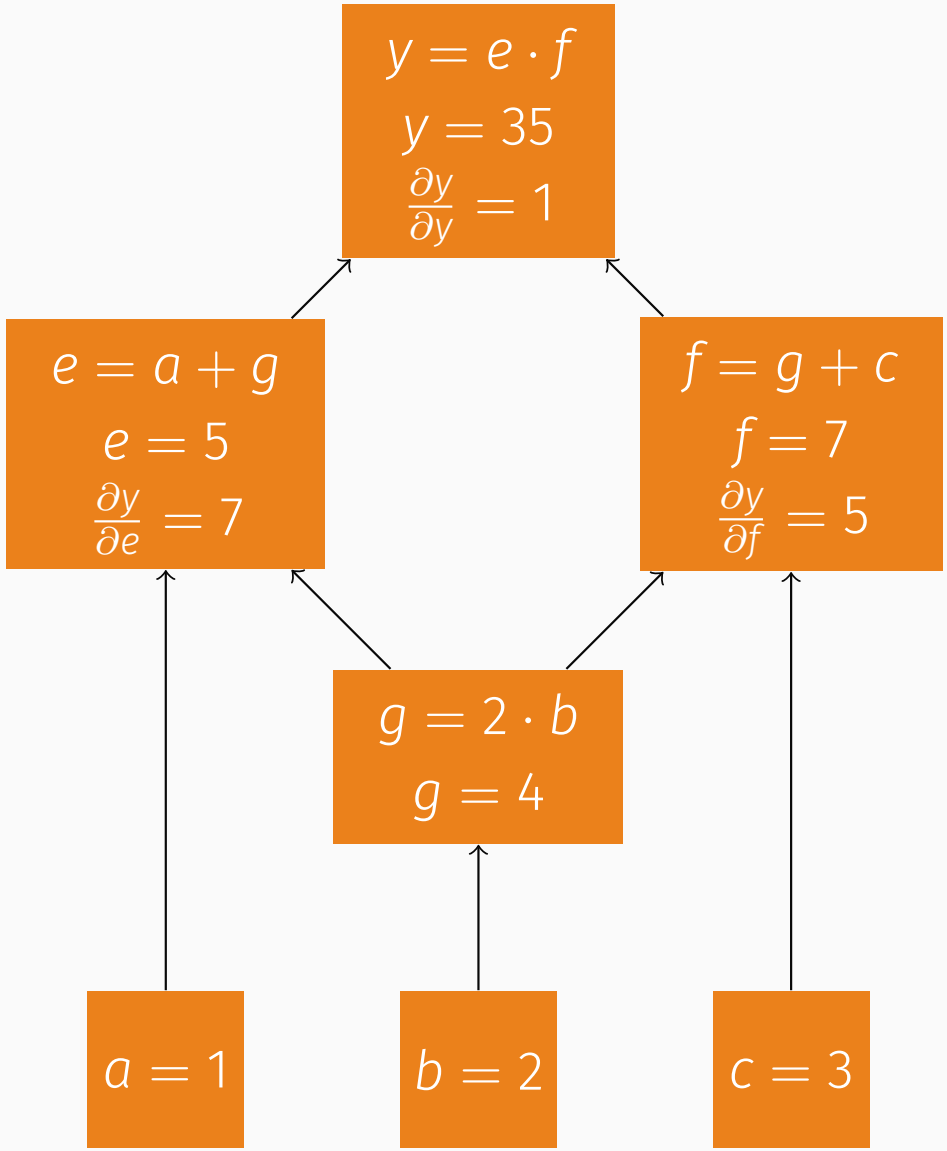
Будем считать частные производные с конца, используя полученную на предбудущих шагах информацию для вычисления значений.

Другими словами, мы будем последовательно применять $\frac{\partial y}{\partial \cdot}$ к каждому узлу.

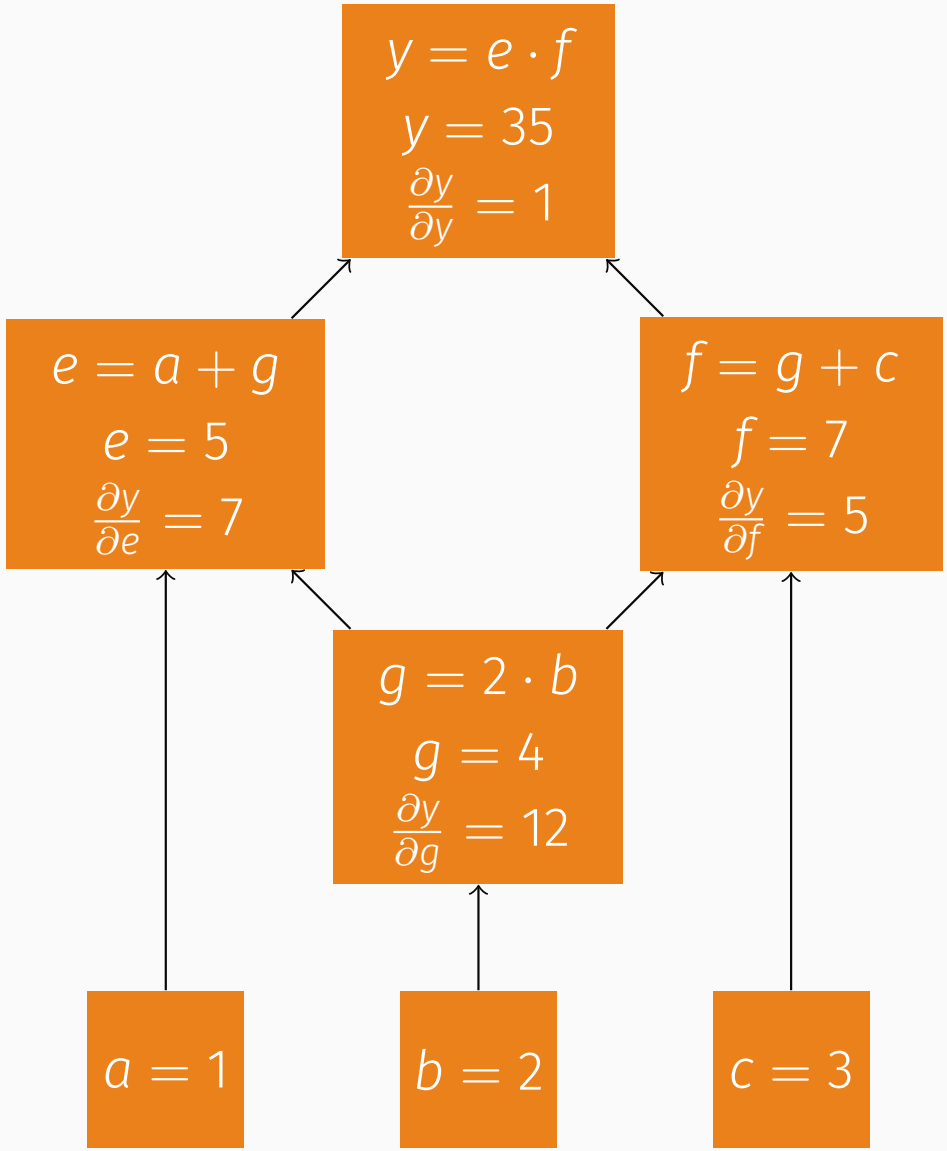
ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



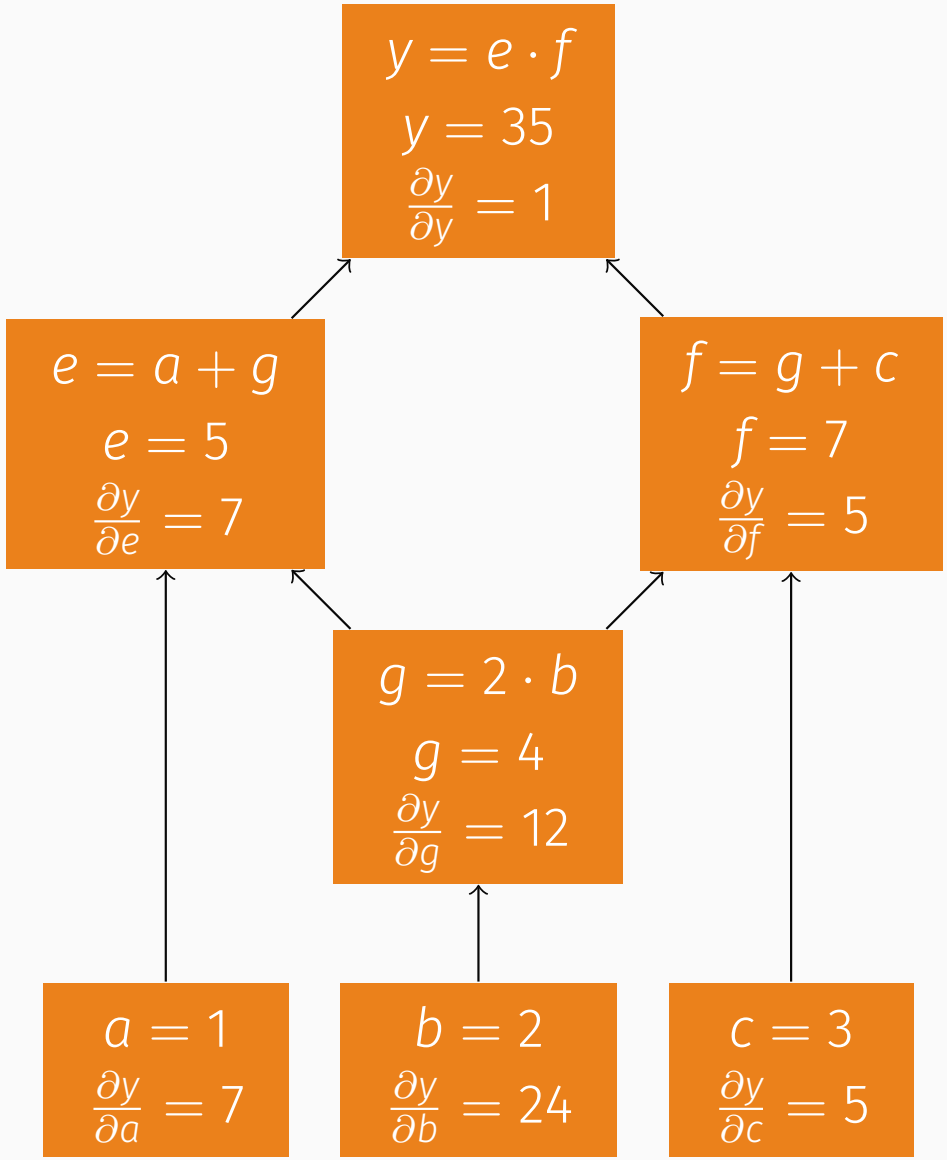
ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

Таким образом мы смогли сразу получить *все* необходимые частные производные.

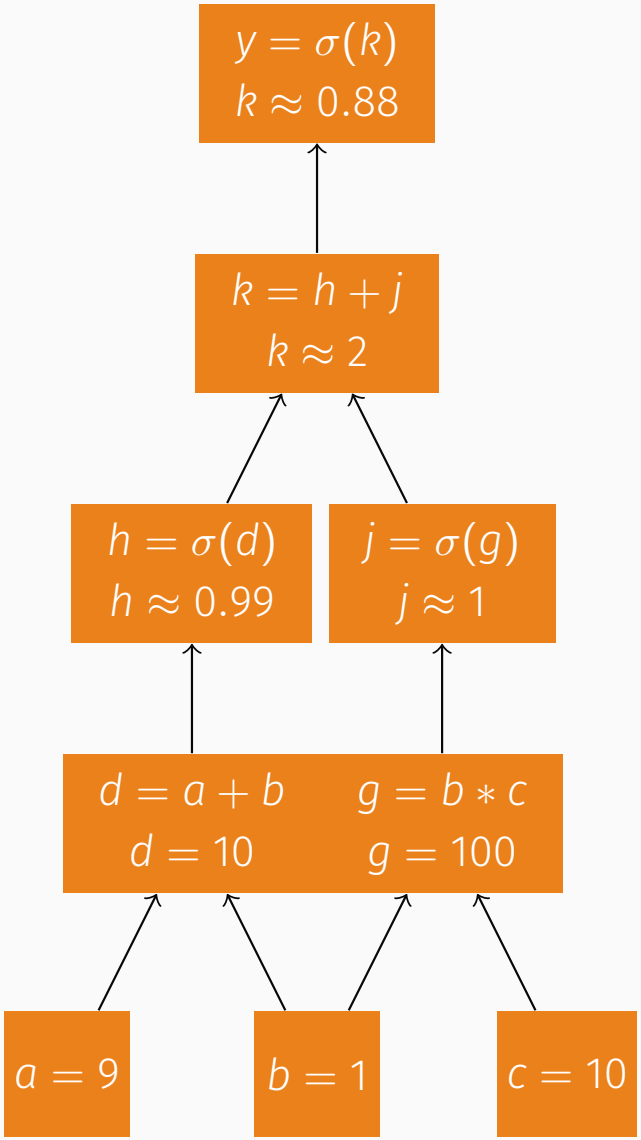
Данный подход называется алгоритмом **обратного распространения ошибки** (error backpropagation).

ВЫЧИСЛИТЕЛЬНЫЕ ПРОБЛЕМЫ

ПРИМЕР

- Вычислим градиент функции $y = \sigma(\sigma(a + b) + \sigma(b * c))$ при помощи метода обратного распространения ошибки
- Примем $a = 9, b = 1, c = 100$

ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ПРИМЕР

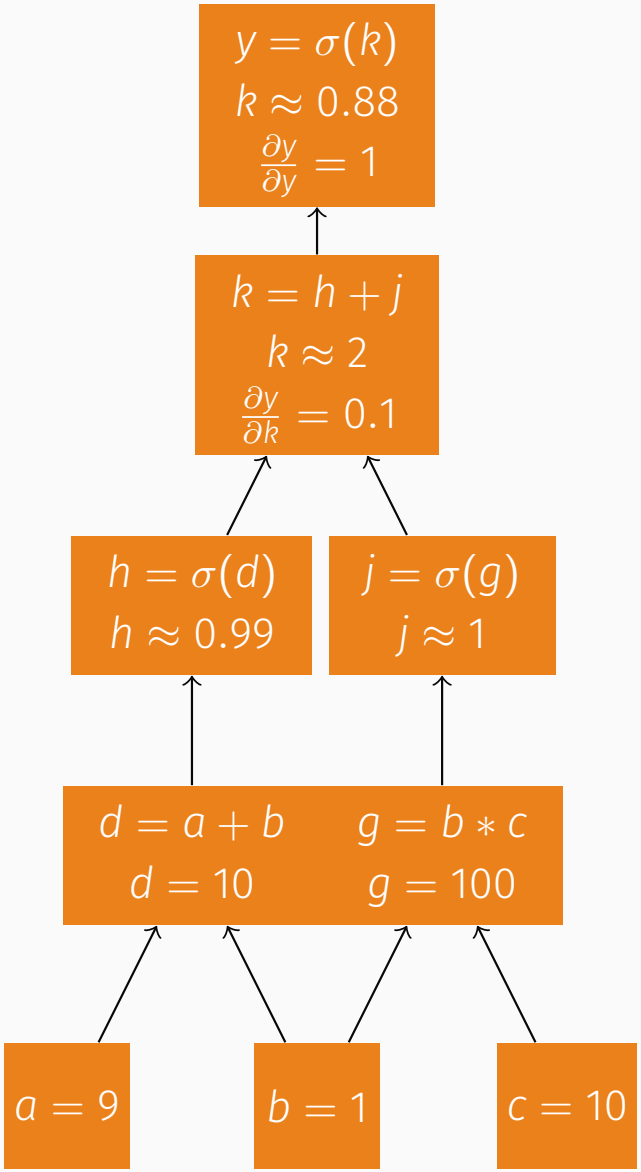
Найдём производную $\sigma(x)$ по x:

$$\sigma(x)' = \left(\frac{1}{1+e^{-x}}\right)' = \frac{0(1+e^{-x}) + 1(0+e^{-x})}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})^2}$$

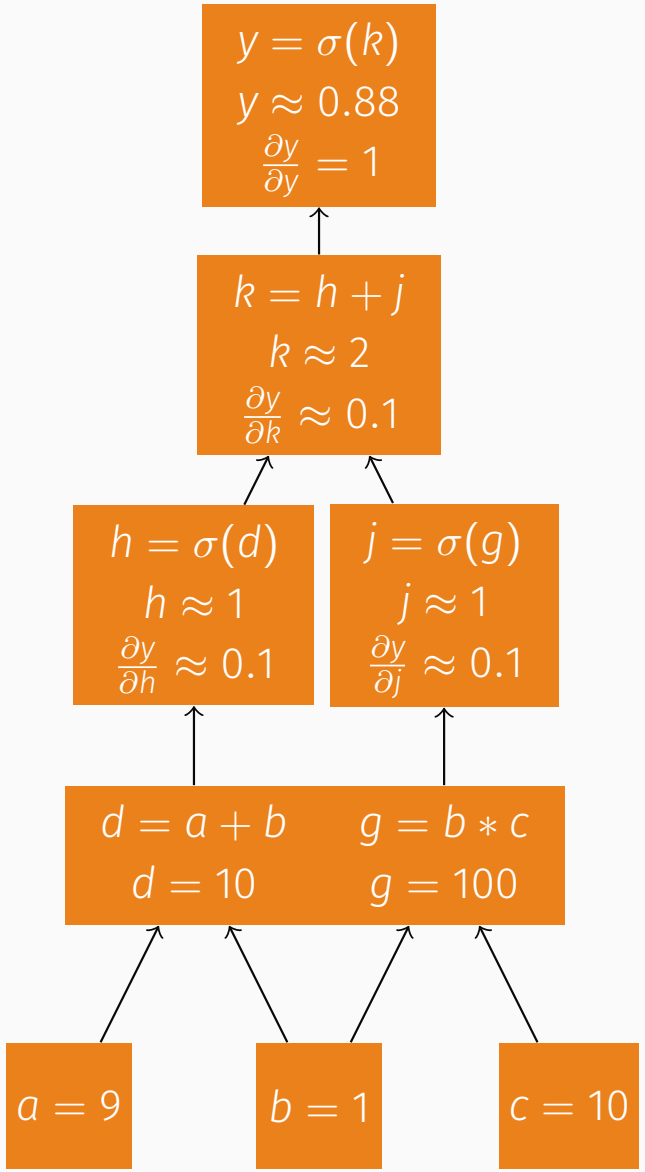
Или

$$\sigma(x)' = \sigma(x)(1-\sigma(x))$$

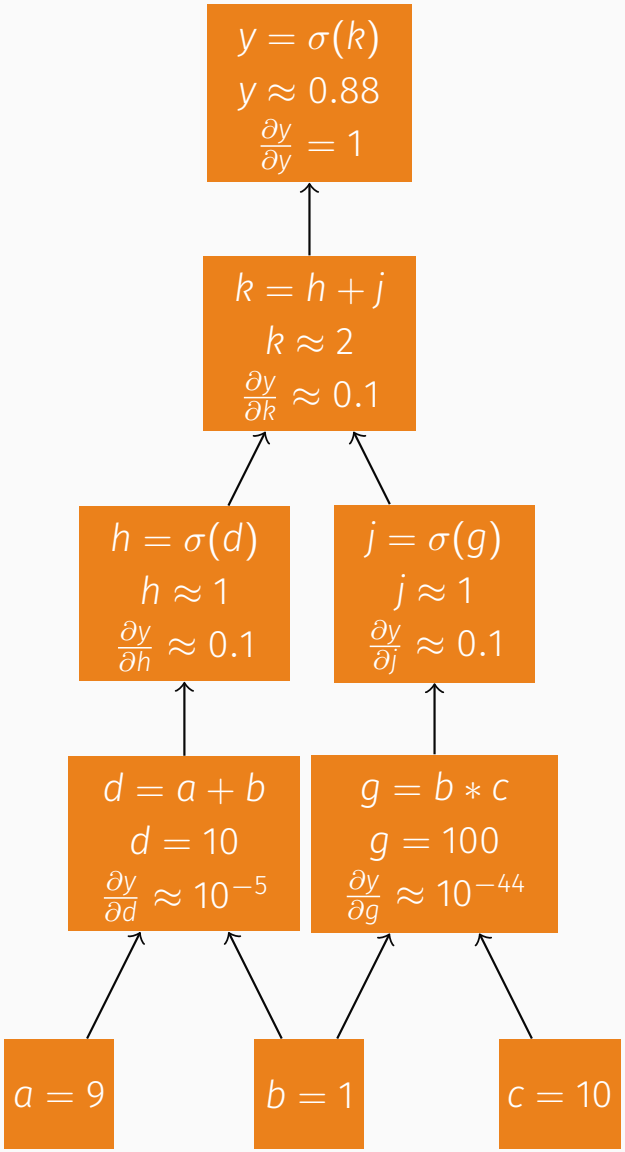
ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ПРИМЕР

Дальше можно не считать, очевидно, что:

$$\nabla y(a, b, c) \approx (0, 0, 0)$$

Если бы это была функция потерь нейронной сети, о чем бы это говорило?

ПРИМЕР

Данная проблема называется проблемой **исчезающего градиента** (vanishing gradient problem).

В рассмотренном примере использовалась функция $\sigma(x)$, максимальное значение, которое может принять её производная равно 0.25. В глубокой нейронной сети, использующей сигмоид в качестве функции активации между слоями, градиент с большой вероятностью будет исчезать.

Важно понимать как работает алгоритм обратного распространения и как ведут себя производные функций активации, которые вы используете!