DEEP LEARNING

Введение

Святослав Елизаров, Коваленко Борис

12 марта 2016

Высшая школа экономики

ОСНОВНЫЕ ПОНЯТИЯ

ПОСТАНОВКА ЗАДАЧИ

Пусть Ω – множество всех объектов. Обозначим через X некоторое подмножество этого множества, $X \subset \Omega$

Множество Y – это множество значений целевого признака.

Функция $\tilde{f} : \Omega \to Y$ ставит в соответствие каждому объекту некоторое значение $y \in Y$.

ПОСТАНОВКА ЗАДАЧИ

Дано:

- Множество X
- Значения функции $\tilde{f}$ на множестве X

Задача: Предсказать значения $\tilde{f}$ для всего множества Ω, или, другими словами, восстановить функцию f. Восстановленную функцию будем обозначать просто f.

Такая задача называется **обучением с учителем** или **supervised learning**

ПОСТАНОВКА ЗАДАЧИ

Задачу можно переформулировать языком математической статистики:

Пусть Ω – пространство всех объектов. Обозначим через X некоторое его подмножество, $X \subset \Omega$

Множество Y – значений целевого признака (например, для классификации это множество меток).

$P(Y|X)$ – условное распределение целевого признака на множестве объектов.

Необходимо восстановить $P(Y|X)$. Восстановленное распределение будем, по традиции, обозначать $Q(Y|X)$

ПОСТАНОВКА ЗАДАЧИ

В случае если значения функции $\tilde{f}$ на множестве X неизвестны, то такая задача называется **обучением без учителя** или **unsupervised learning**

ПОСТАНОВКА ЗАДАЧИ

В данном курсе мы столкнёмся со множеством частных случаев каждой из этих задач:

С учителем:

1. Классификация
2. Регрессия
3. Сегментация изображений
4. ...

Без учителя:

1. Word or sentence embeddings
2. Кластеризация
3. Style transfer
4. ...

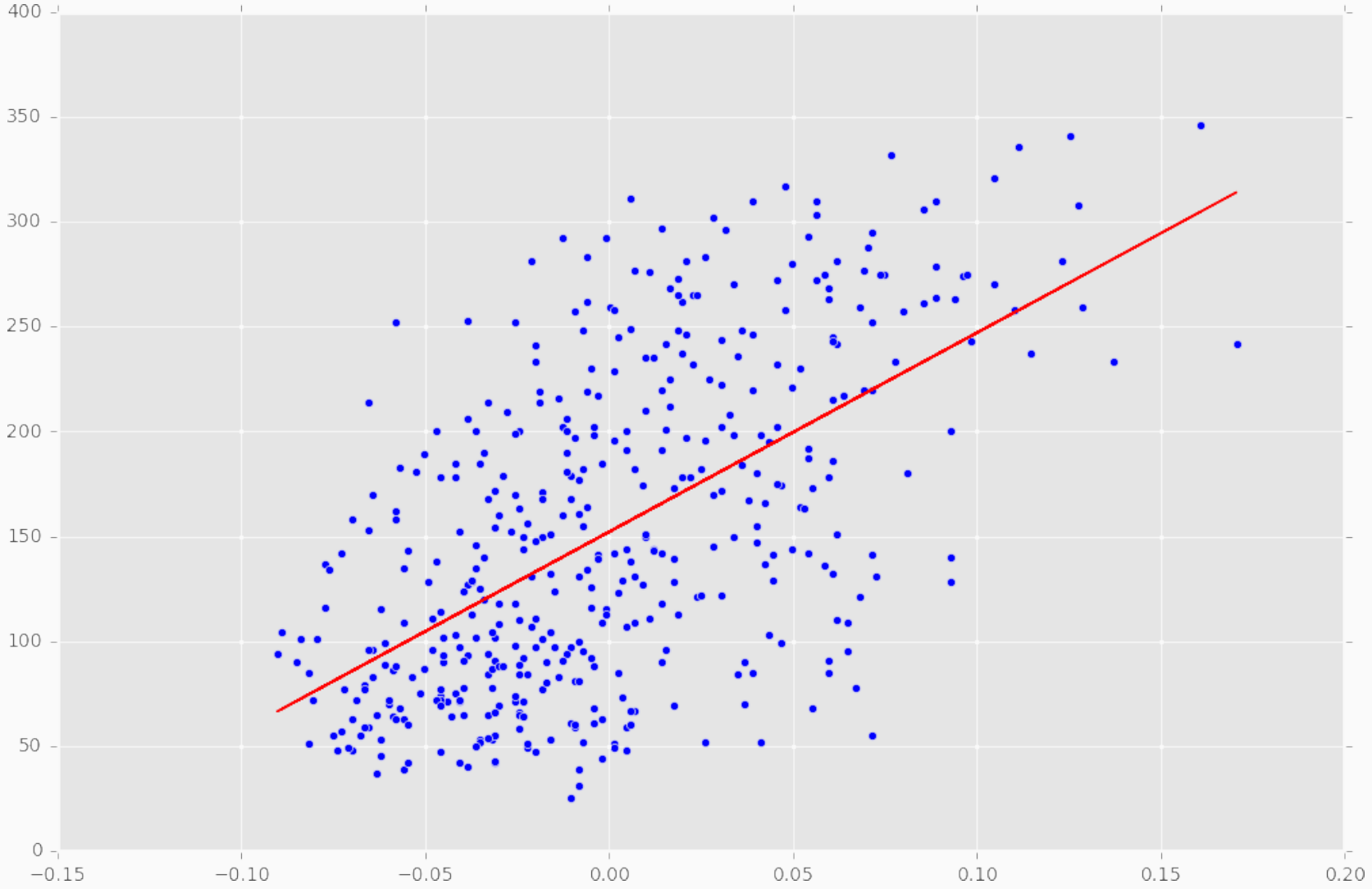
ЛИНЕЙНЫЕ МОДЕЛИ

ЛИНЕЙНАЯ РЕГРЕССИЯ

Простейшая линейная регрессия:

$$f(x) = wx + b$$

ПРИМЕР ПАРНОЙ РЕГРЕССИИ



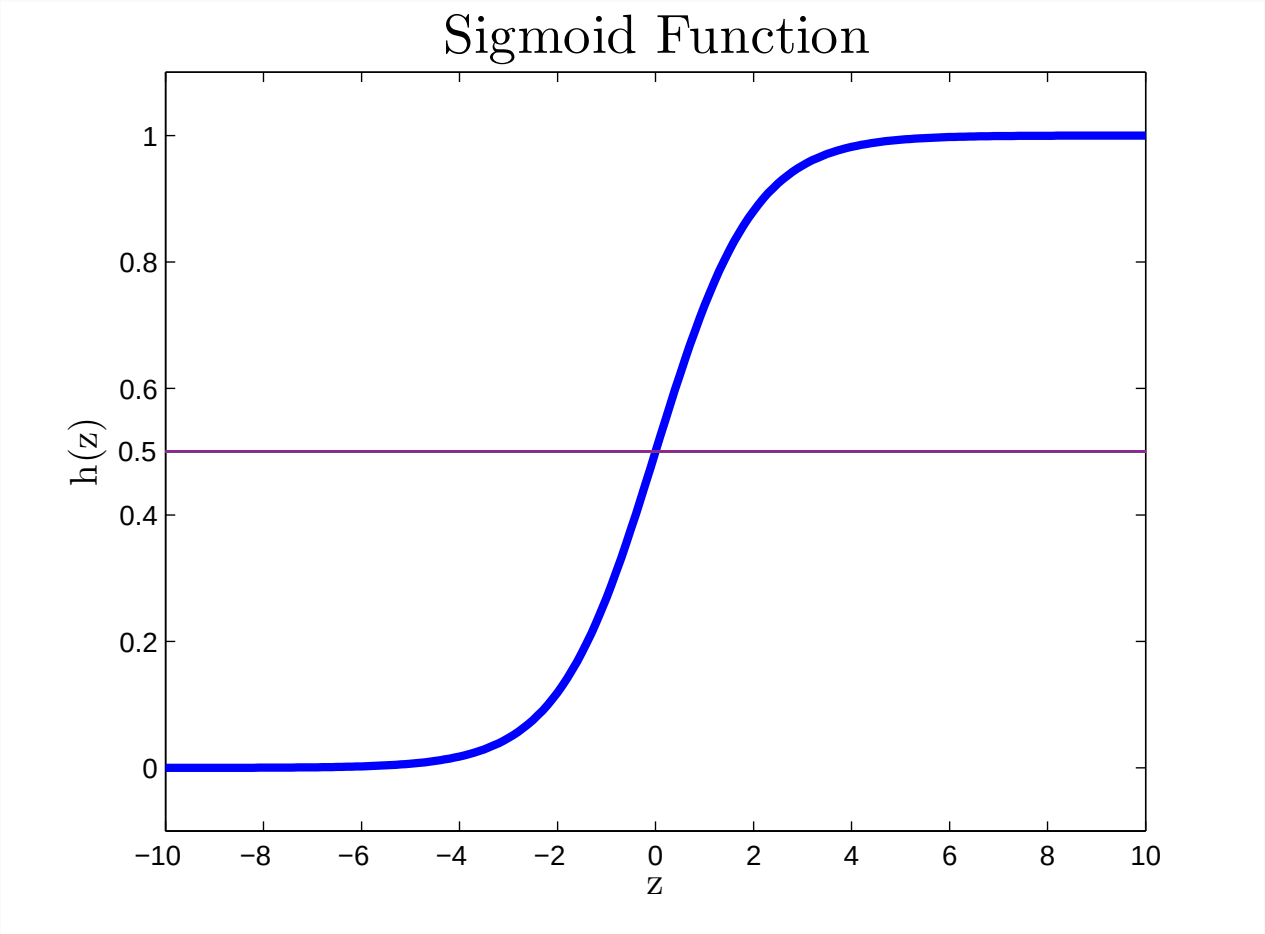
Логистическая регрессия

Логистическая регрессия:

$$f(x) = \sigma(wx + b)$$

Где $\sigma(x) = \frac{1}{1-e^{-x}}$

Функция $\sigma(a)$ называется **логистическим сигмоидом** (logistic sigmoid)



Если мы хотим перейти в другое пространство, где выборка разделима линейно, мы можем добавить полиномиальные признаки. Например x^2:

$$f(x) = \sigma(w_1 x + w_2 x^2 + b)$$

Опишем в общем виде модель линейной регрессии:

$$f(x) = x'w^T = \langle x', w \rangle$$

где

- $x = (x_1, x_2, \ldots, x_n) \in \Omega$
- $x' = 1 \cup x$, x' – это вектор-строка x, первым (с индексом 0) элементом которой назначена константа 1. В дальнейшем будем подразумевать под x вектор данной конструкции.
- $w = (w_0, w_1, \ldots, w_n)$

ФУНКЦИИ ПОТЕРЬ

multiclass svm (hinge loss)

$$L_i = \sum_{j \neq y_i} max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

MULTICLASS SVM (HINGE LOSS)

Пример:

Получен вектор скоров для задачи классификации: $[1.2, 5, 1.6]$, параметр $\Delta = 2$, истинный класс 1, loss для данного объекта равен

$$L_i = max(0, 5 - 1.2 - 2) + max(0, 1.6 - 1.2 - 2)$$

L2 Регуляризация

$$R(W) = \sum_i \sum_j w_{i,j}^2$$

CROSS ENTROPY LOSS

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W)$$

Cross entropy loss

$$L_i = -\log(\frac{e^{f(y_i)}}{\sum_m e^{f(y_m)}})$$

Функция softmax переводит скоры в ”вероятности”:

$$f_i(z) = \frac{e^{f(z_i)}}{\sum_m e^{f(z_m)}}$$

CROSS ENTROPY LOSS

Кросс энтропия :

$$H(p, q) = -\sum_{x} p(x) \log q(x) = H(p) + D_{KL}(p||q)$$

Данная функция потерь минимизирует KL дивергенцию между истинным и полученным распределением для объектов.

Истинное распределение - $[0, ..., 1, ..., 0]$

Предсказанное распределение - $[0.05, ..., 0.6, ..., 0.1]$

CROSS ENTROPY LOSS

Пример:

Получен вектор скоров для задачи классификации: $[1.2, 5, 1.6]$, истинный класс 1

- Получим вектор ”вероятностей”с помощью софтмакса - $[0.021, 0.947, 0.032]$
- $L_i = -\log(0.021) = 3.86$

В чем отличия SVM Multiclass и Cross entropy функции потерь?

CROSS ENTROPY LOSS

Пример:

Получен вектор скоров для задачи классификации: $[1.2, 5, 1.6]$, истинный класс 1

- Получим вектор ”вероятностей”с помощью софтмакса - $[0.021, 0.947, 0.032]$
- $L_i = -\log(0.021) = 3.86$

В чем отличия SVM Multiclass и Cross entropy функции потерь?

На практике получаемые результаты очень похожи

CROSS ENTROPY LOSS

Пример:

Получен вектор скоров для задачи классификации: $[1.2, 5, 1.6]$, истинный класс 1

- Получим вектор ”вероятностей”с помощью софтмакса - $[0.021, 0.947, 0.032]$
- $L_i = -\log(0.021) = 3.86$

В чем отличия SVM Multiclass и Cross entropy функции потерь?

На практике получаемые результаты очень похожи

Тема по прежнему не раскрыта - как найти хорошие W и b?

Теперь введём следующую модель:

$$f(x) = \psi\left(\sum_{i=0}^{N} w_i \phi_i(x)\right) = \psi\left(\langle w, \phi(x)\rangle\right)$$

Где w_i – веса (параметры модели) при i-м компоненте, w – вектор (матрица или тензор) весов.

Где ϕ_i – базисные функции (налагается требование дифференцируемости). При их помощи можно, например, добавить x^2 как признак.

$\phi(x)$ – вектор (матрица или тензор) значений базисных функций от x.

ψ – функция активации, так же должна быть дифференцируемой. В случае логистической регрессии это сигмоид.

Искусственные нейронные сети прямого распространения (feed-forward artificial neural networks) являются частным случаем этой модели.

- В качестве базисных функций $\phi_i(x)$ возьмём обобщённую линейную модель.
- Поступим так несколько раз
- Верхним индексом обозначаем уровень вложенности (самый ”глубокий”0)
- Совокупность элементов имеющих один верхний индекс принято называть ”слоем”

Например, так будет выглядеть формула, описывающая простейшую полносвязную сеть с одним скрытым слоем (размерности 3), решающую задачу бинарной классификации для $x \in R^3$:

$$y(x) = \sigma(w_1^2(w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3) + \cdots + w_3^2(w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3))$$

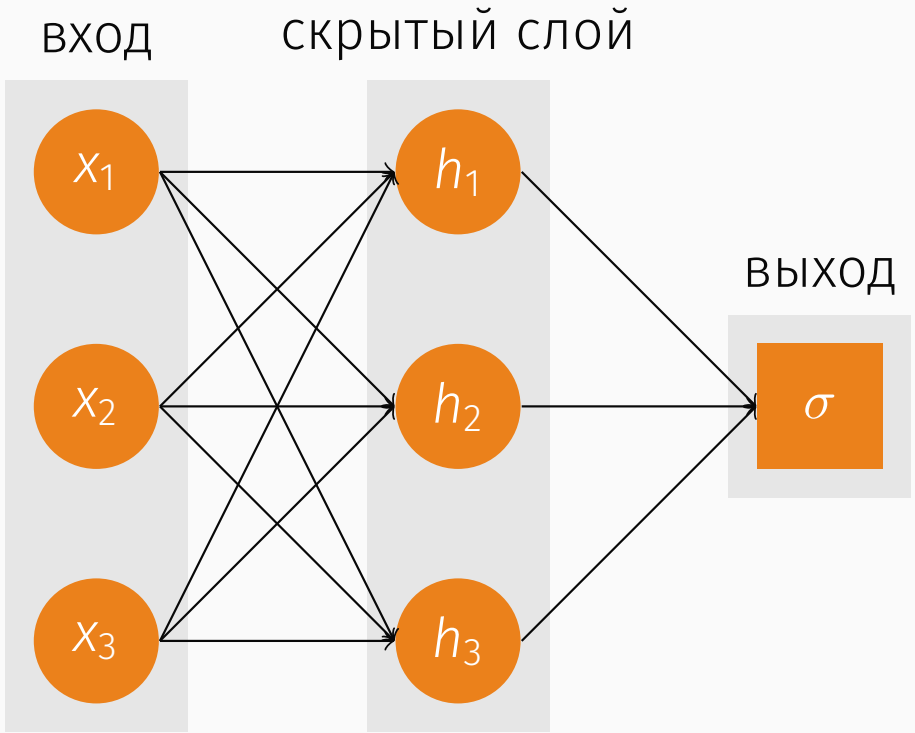
Или в матричном виде. Обратите внимание, что для удобства записи номер слоя теперь обозначен в нижнем индексе:

$$y(x) = \sigma(W_2^T(W_1^T x))$$

ГРАФ НЕЙРОННОЙ СЕТИ

Теперь, если мы представим каждое слагаемое как вершину в графе и проведём ребра между теми вершинами которые представлены в одной сумме с ненулевыми весами, то получим привычный граф искусственной нейронной сети.

ГРАФ НЕЙРОННОЙ СЕТИ



Где $h_i = w^1_{i1}x_1 + w^1_{i2}x_2 + w^1_{i3}x_3$

ОБУЧЕНИЕ НЕЙРОННЫХ СЕТЕЙ

Для всех описанных ранее линейных моделей функция потерь была выпуклой. Проверим это для логистической регрессии:

$$t \log\left(\sigma(ax+b)\right) + (1-t)\log\left(\sigma(ay+b)\right) \leq \log\left(\sigma(a(tx+(t-1)y)+b)\right)$$

$$\log\left(\left(\sigma(ax+b)\right)^{t}\left(\sigma(ay+b)\right)^{(1-t)}\right) \leq \log\left(\sigma(a(tx+(t-1)y)+b)\right)$$

$$\log\left(\frac{\left(\sigma(ax+b)\right)^{t}\left(\sigma(ay+b)\right)^{(1-t)}}{\sigma(a(tx+(t-1)y)+b)}\right) \leq 0$$

Так как $\forall x, \sigma(x) \in (0,1)$ выражение под логарифмом всегда лежит в этом интервале. Значения логарифма на этом интервале всегда отрицательные. Следовательно функция вогнутая.

Проделайте тоже самое для случая множества классов.

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.

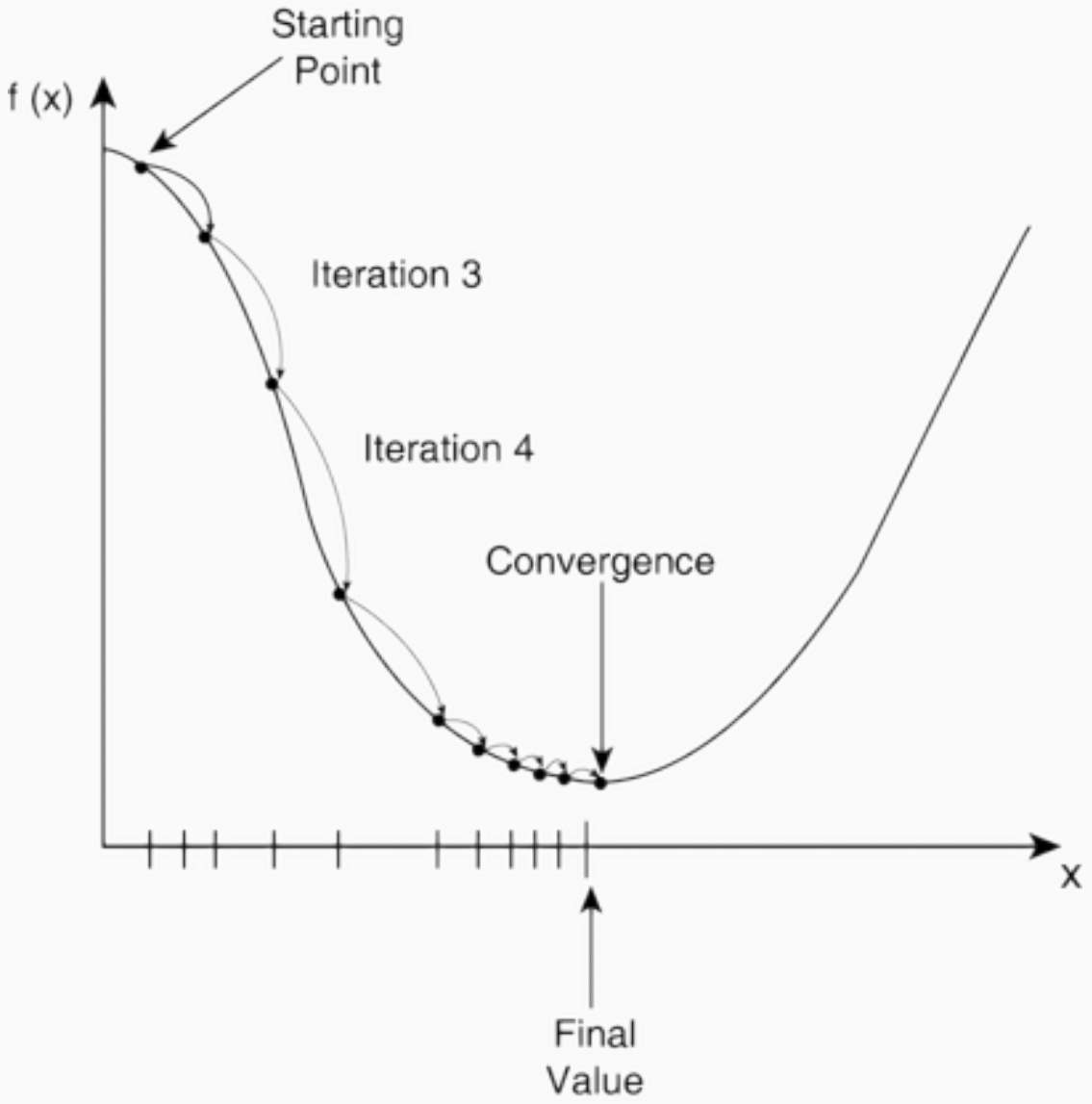
Таким образом мы можем применять основанные на вычислении градиента методы, не боясь застрять в локальном минимуме. Так же важным является то, что мы может вычислить Гессиан, если необходимо.

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

1. Все локальные минимумы примерно одинаковы (в некоторых случаях являются глобальными минимумами)
2. Чем глубже модель, тем меньше вероятность встретить плохой локальный минимум
3. Любая критическая точка, не являющаяся минимумом, является седловой

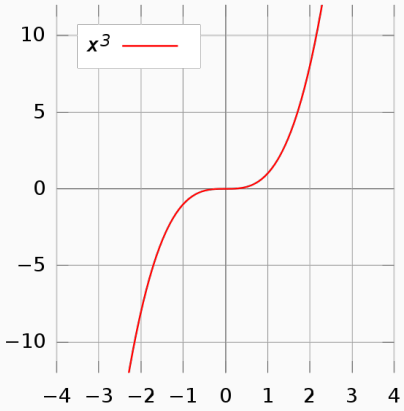
Статьи:

1. LeCun et al., 2014. The Loss Surfaces of Multilayer Networks
2. Kenji Kawaguchi, 2016. Deep Learning without Poor Local Minima

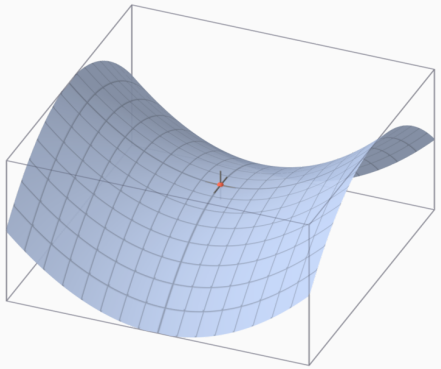
Седловые точки

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.

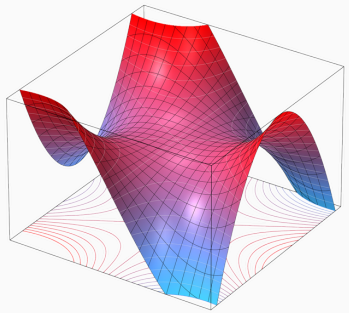
СЕДЛОВЫЕ ТОЧКИ



(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

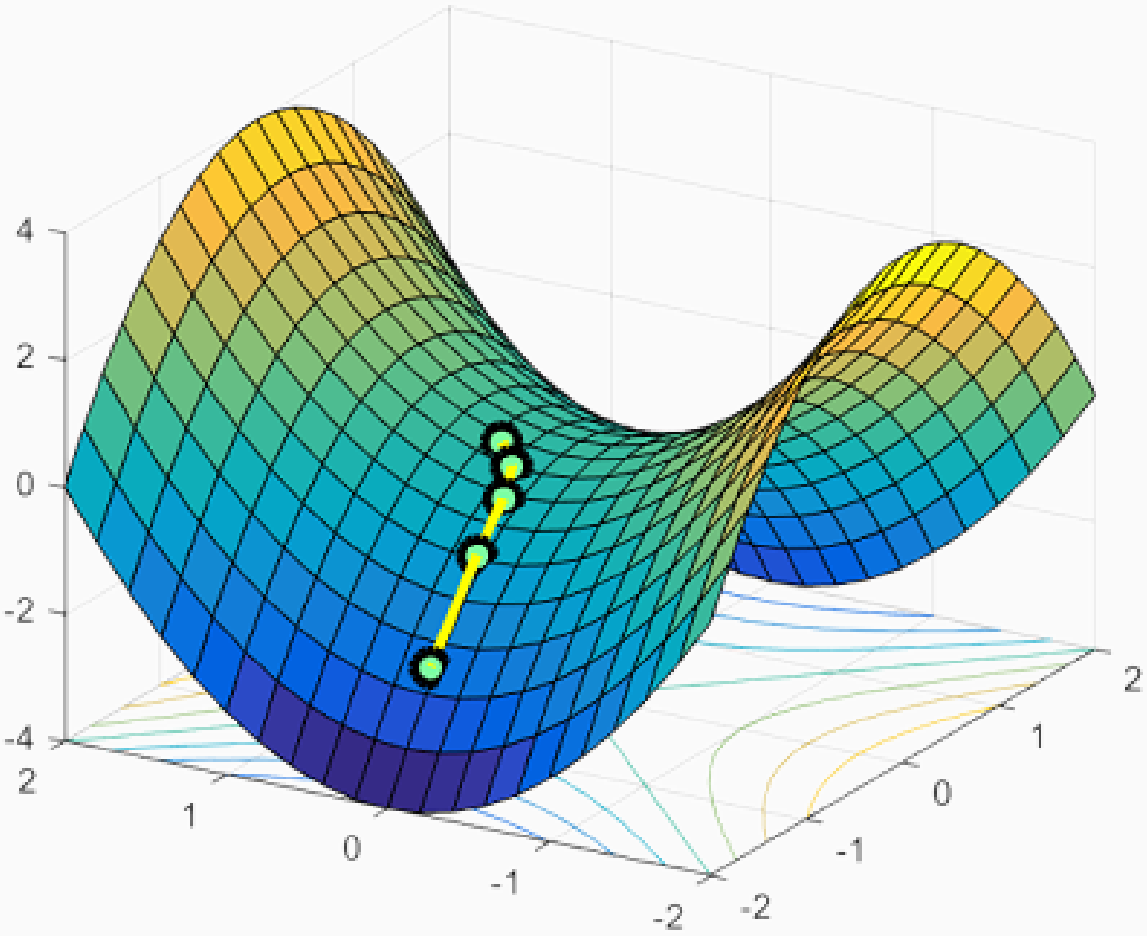
Добавим шум к градиенту. Если представить шарик, который катится по поверхности, то велика вероятность, что он скатится с седловой точки, если будет двигаться с небольшими случайными флуктуациями.

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

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).

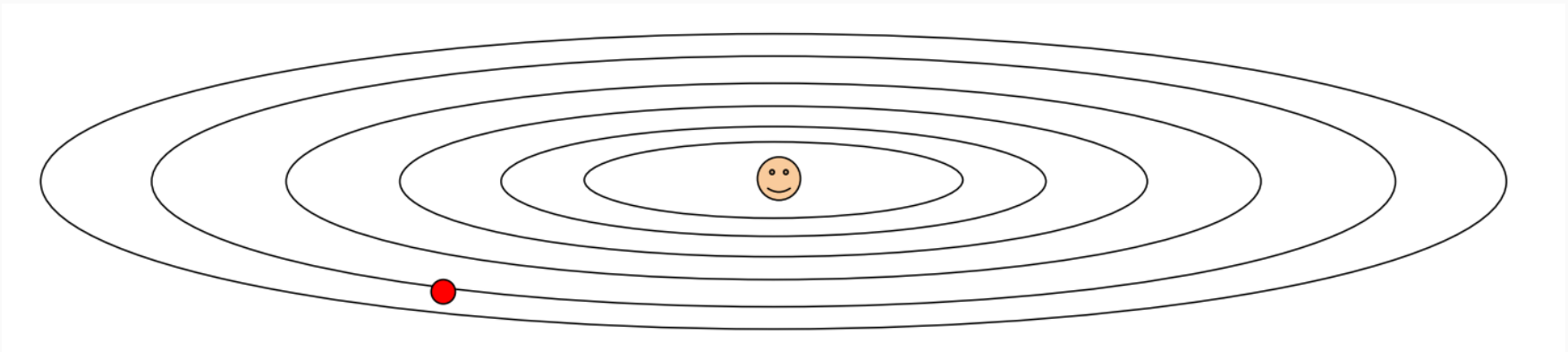
Количество объектов для вычисления градиента выбирается исходя из объема памяти который имеется (максимально заполняем память) или выбирается с помощью кросс-валидации.

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

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

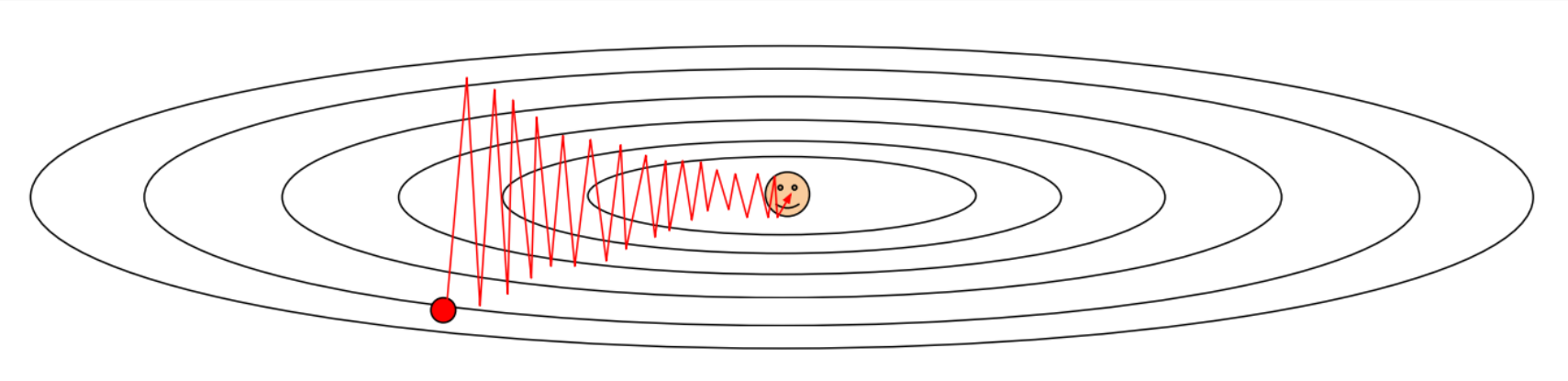
VANILLA SGD

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



VANILLA SGD

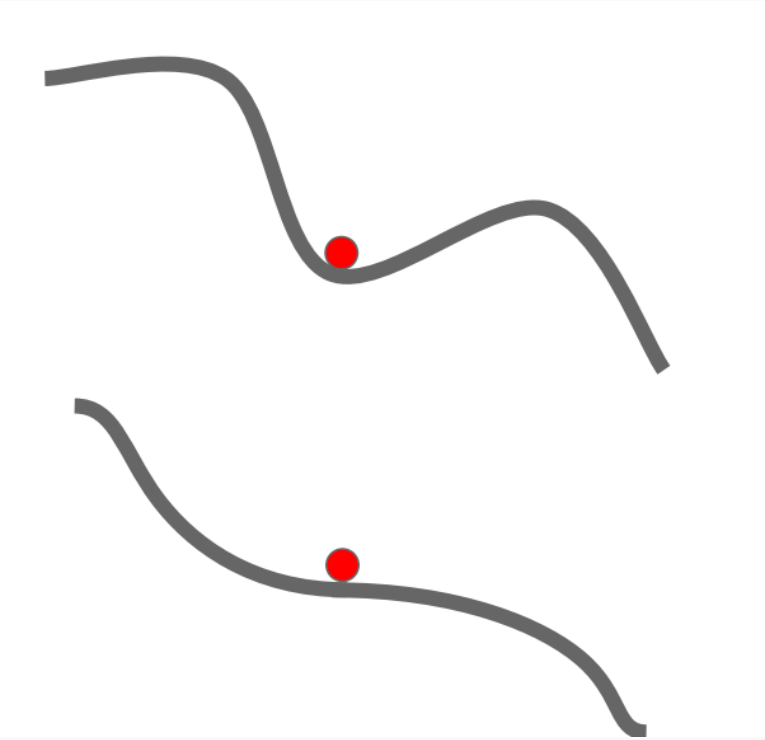
Что будет с Vanilla SGD если линии уровня функции потерь выглядят так:



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:

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

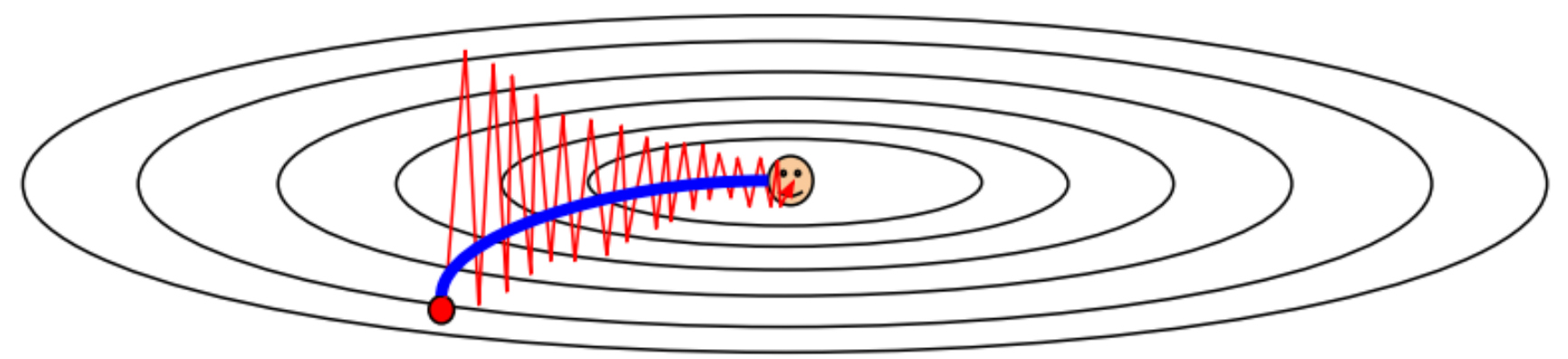
Импульсный метод

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

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

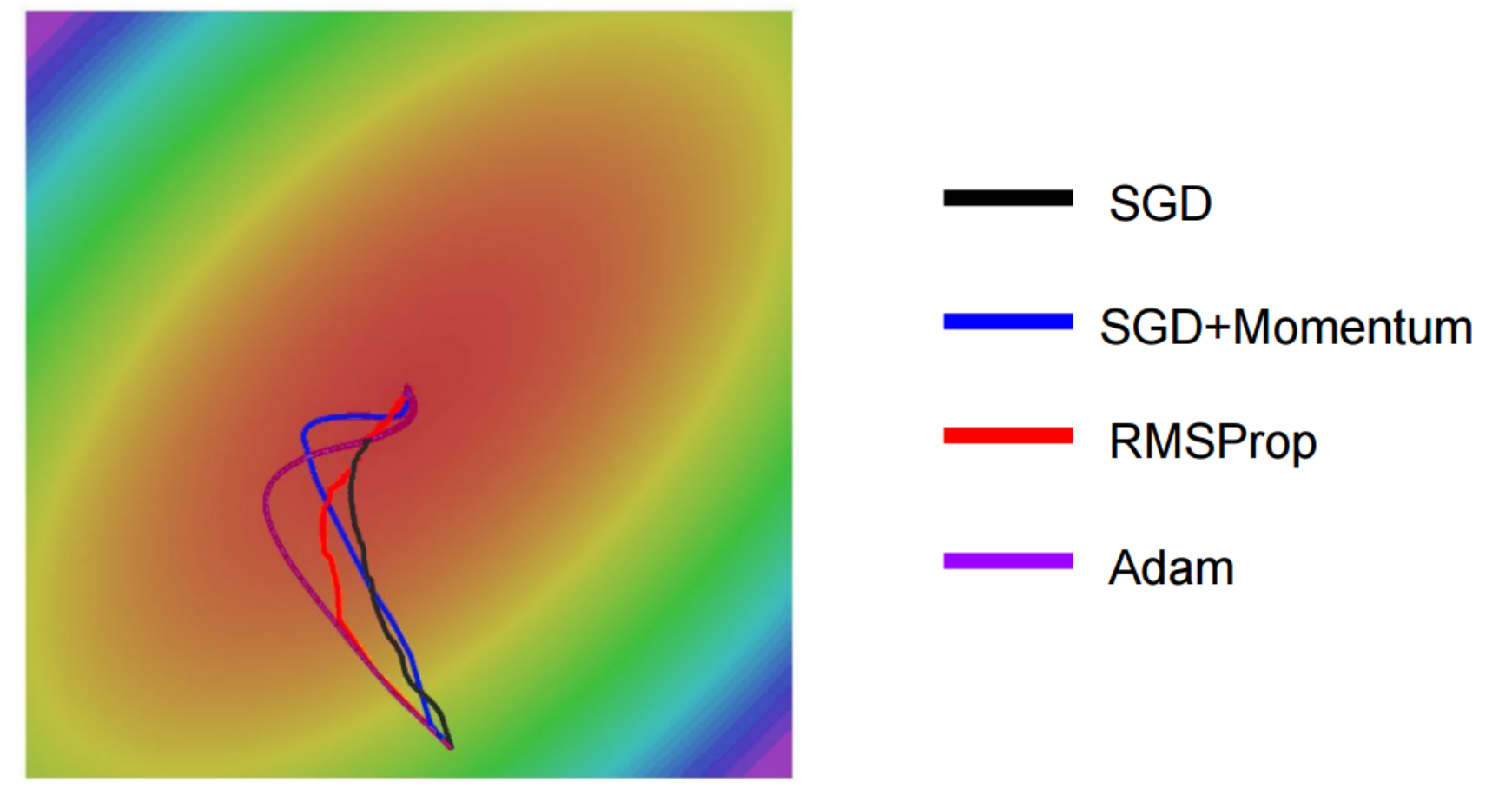
$$Cache_{t+1} = \gamma Cache_t + (1 - \gamma)\nabla L(W_t, ...)$$

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

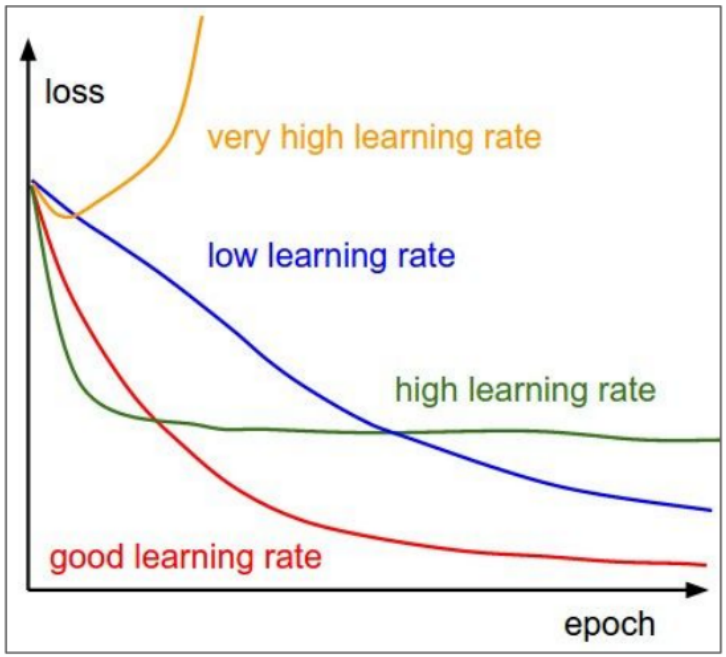
γ - Параметр затухания для истории градиентов, учитываем только окно недавних градиентов

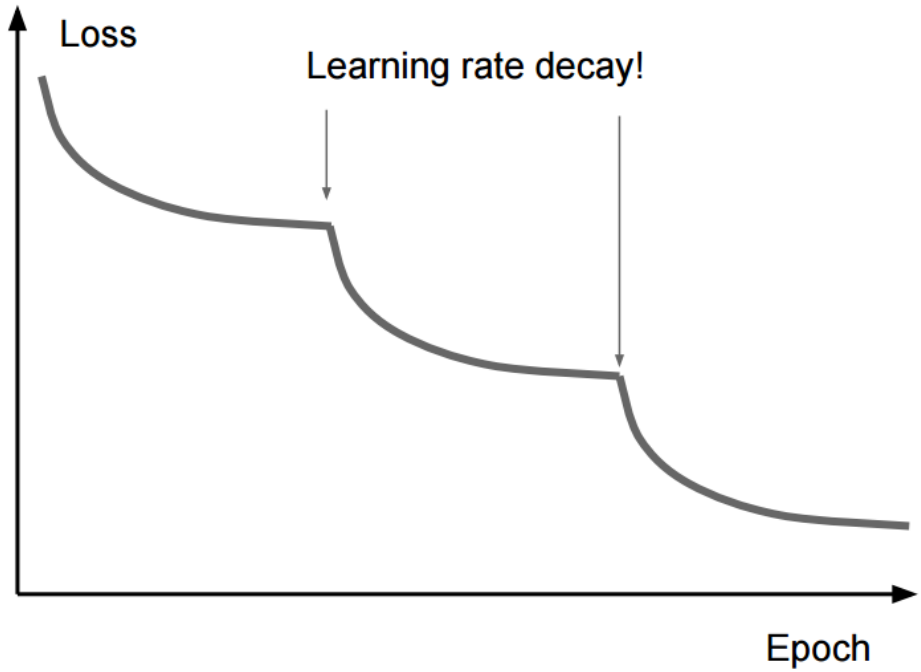
$$Adam = RMSProp + Momentum$$

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.

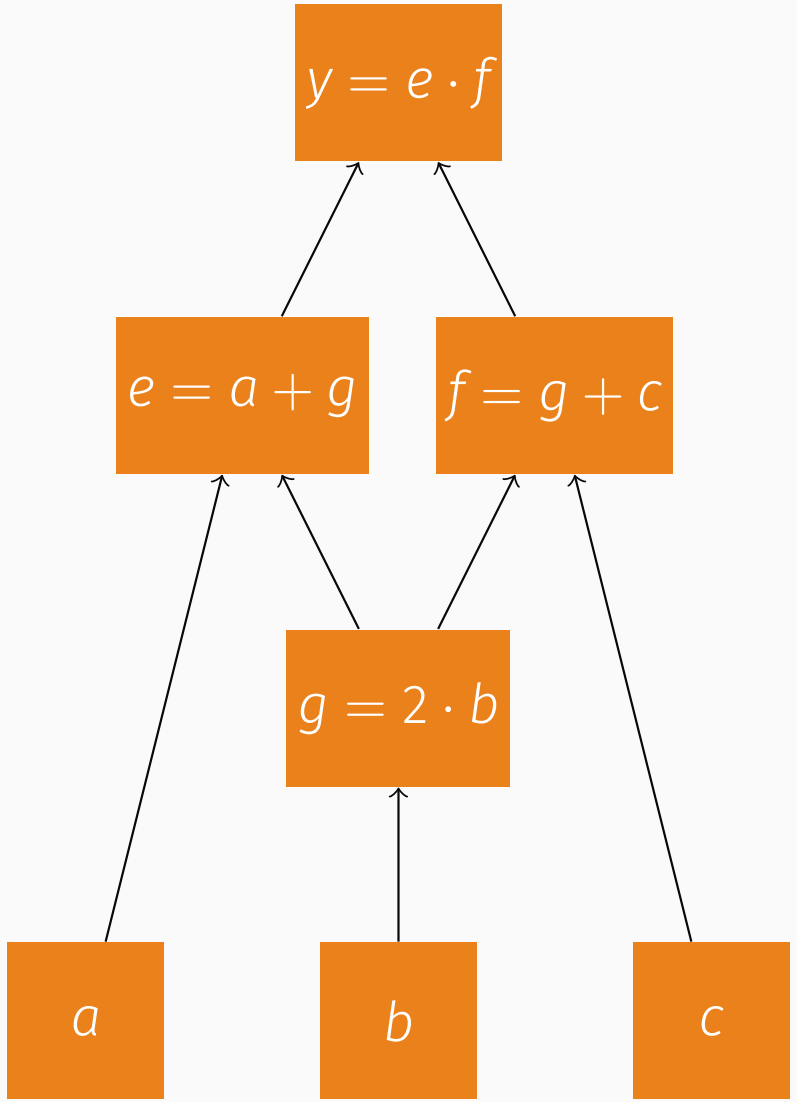
ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

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

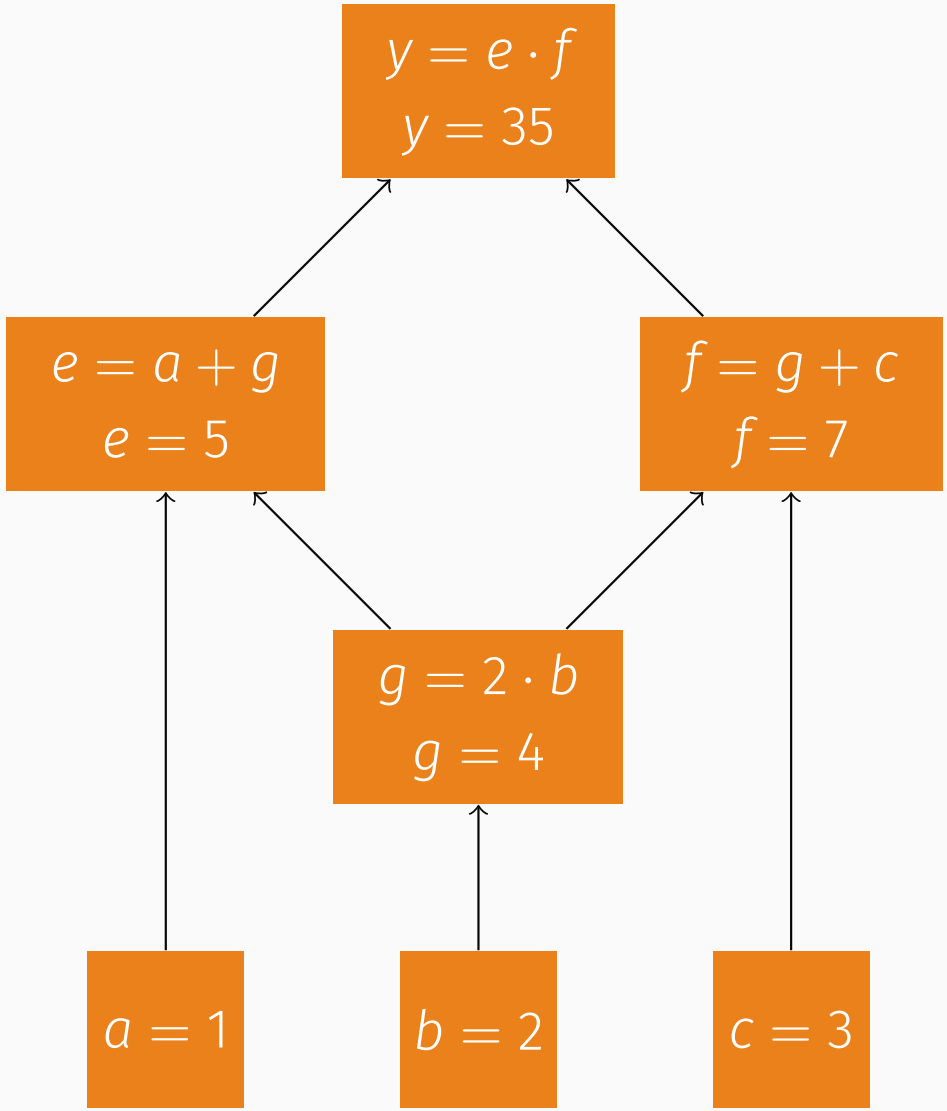
Она состоит из четырёх операций, следовательно в графе будет четыре вершины (и три входа). Выпишем их:

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

ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

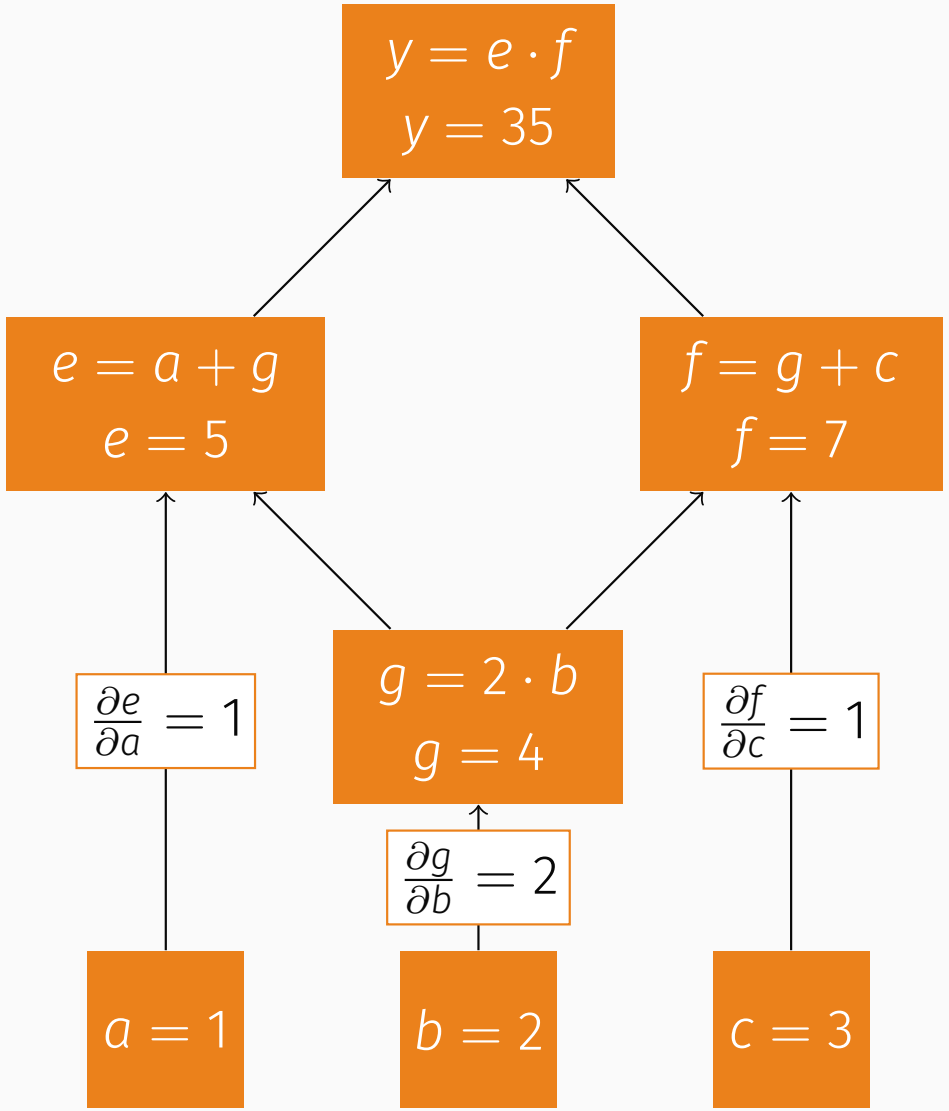


ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

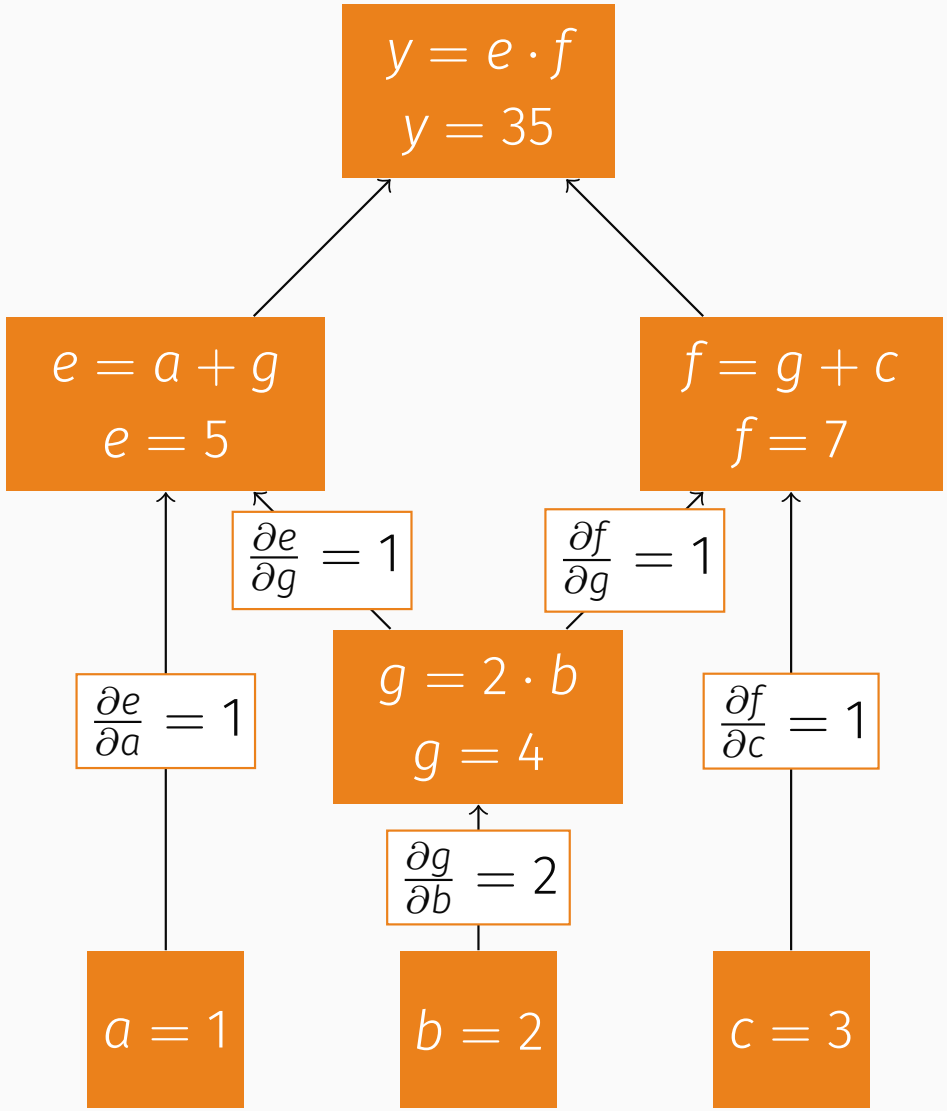
Как видно из примера, значения узла g было рассчитано один раз, но использовалось дважды.

Теперь вычислим градиент функции y.

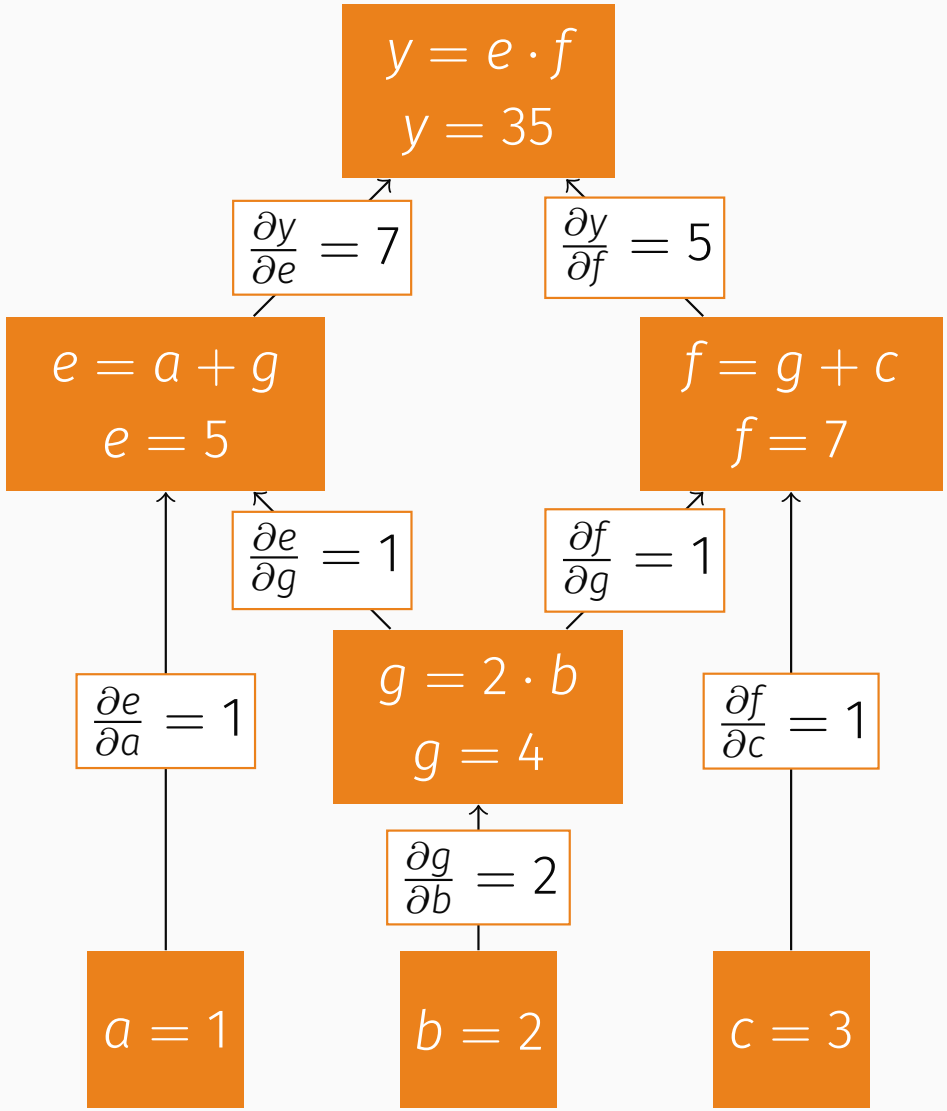
ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



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

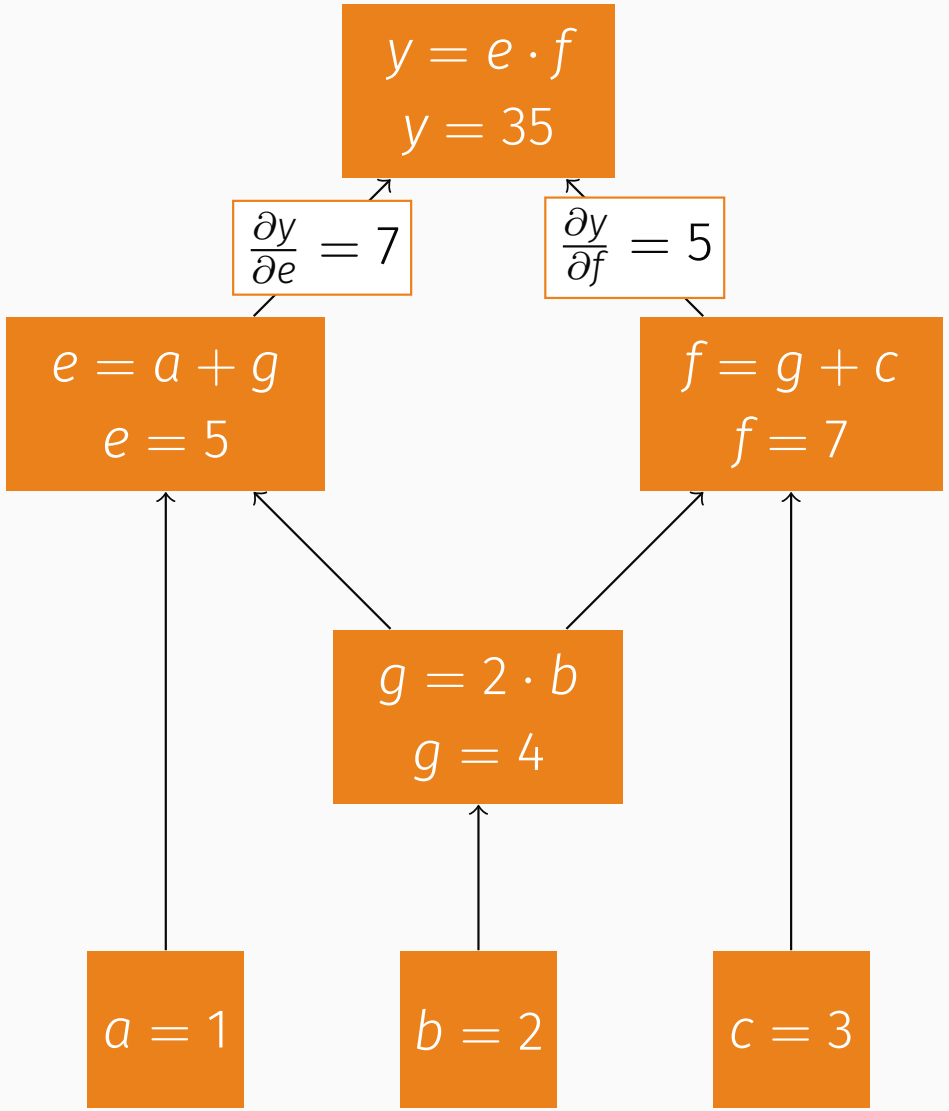
Что с этим делать?

Что с этим делать? Применим динамическое программирование!

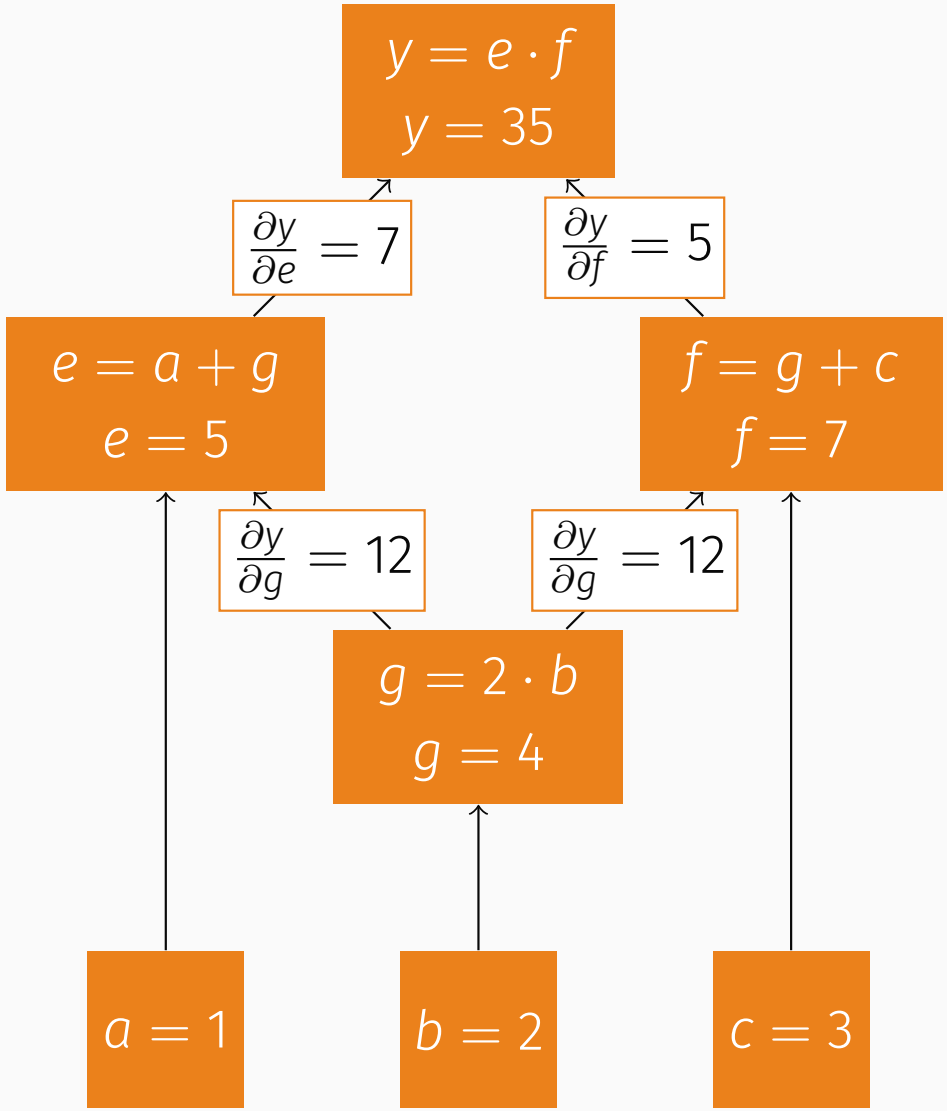
Будем считать частные производные с конца, используя полученную на предбудущих шагах информацию для вычисления значений.

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

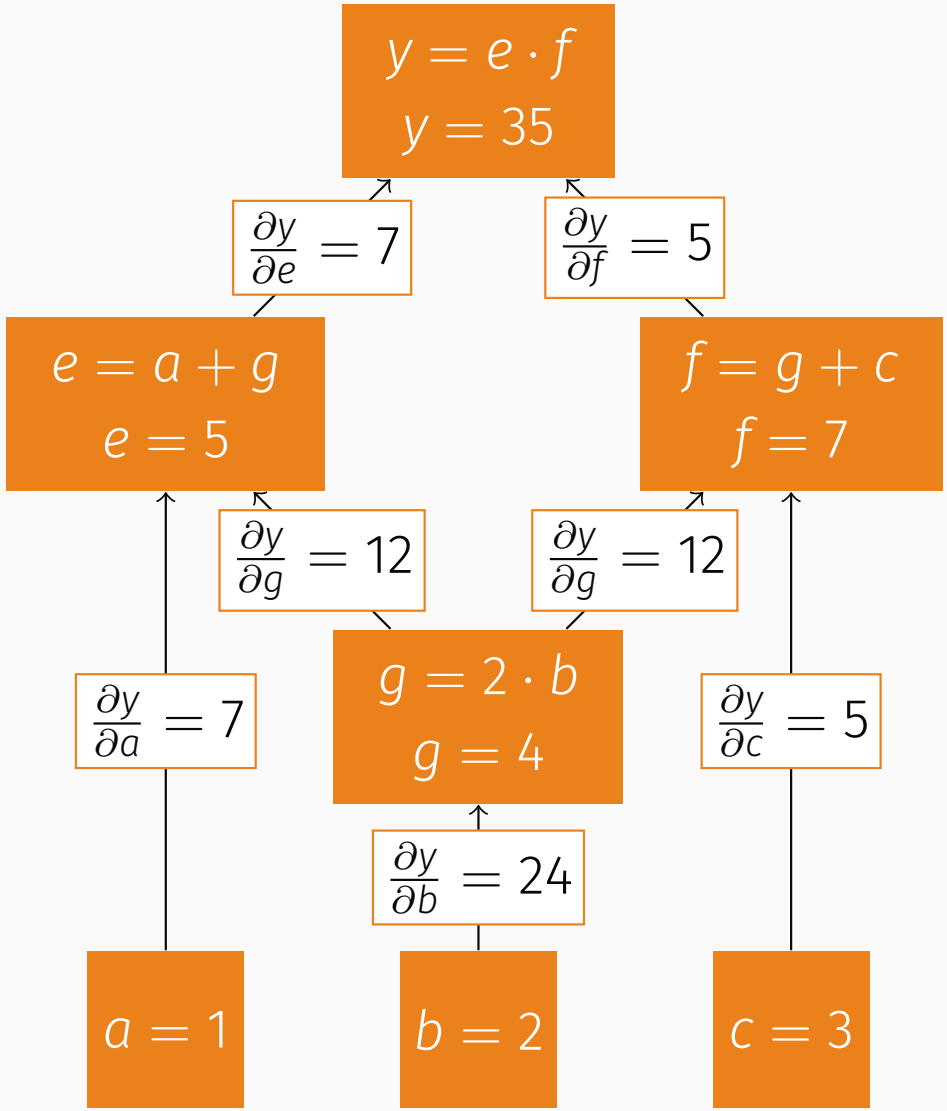
ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ

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

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

ON LARGE-BATCH TRAINING FOR DEEP LEARNING: GENERALIZATION GAP AND SHARP MINIMA

Nitish Shirish Keskar*
Northwestern University
Evanston, IL 60208
keskar.nitish@u.northwestern.edu

Dheevatsa Mudigere
Intel Corporation
Bangalore, India
dheevatsa.mudigere@intel.com

Jorge Nocedal
Northwestern University
Evanston, IL 60208
j-nocedal@northwestern.edu

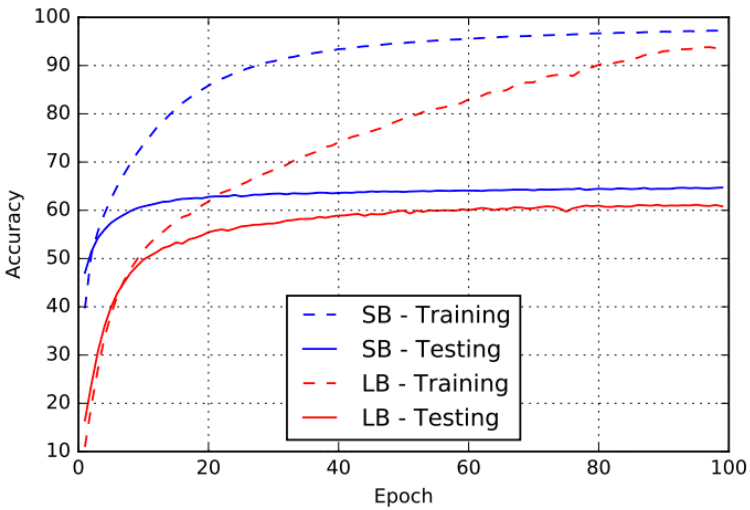
Mikhail Smelyanskiy
Intel Corporation
Santa Clara, CA 95054
mikhail.smelyanskiy@intel.com

Ping Tak Peter Tang
Intel Corporation
Santa Clara, CA 95054
peter.tang@intel.com

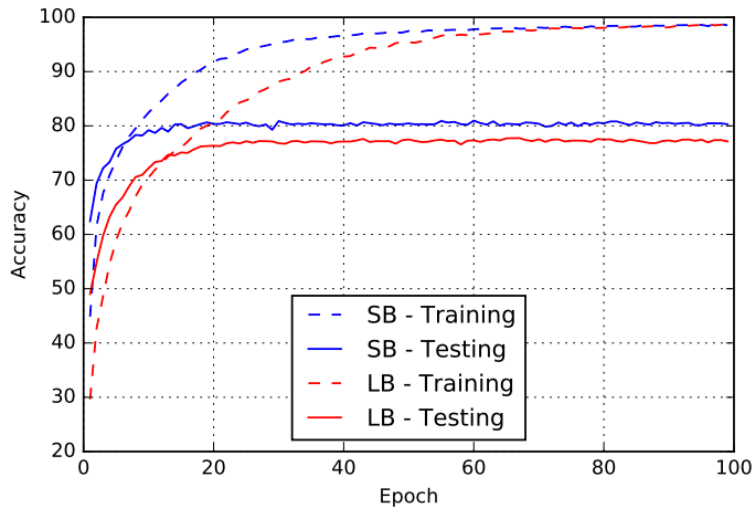
Table 1: Network Configurations

Name	Network Type	Architecture	Data set
F_1	Fully Connected	Section B.1	MNIST (LeCun et al., 1998a)
F_2	Fully Connected	Section B.2	TIMIT (Garofolo et al., 1993)
C_1	(Shallow) Convolutional	Section B.3	CIFAR-10 (Krizhevsky & Hinton, 2009)
C_2	(Deep) Convolutional	Section B.4	CIFAR-10
C_3	(Shallow) Convolutional	Section B.3	CIFAR-100 (Krizhevsky & Hinton, 2009)
C_4	(Deep) Convolutional	Section B.4	CIFAR-100

Name	Training Accuracy		Testing Accuracy	
	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	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$

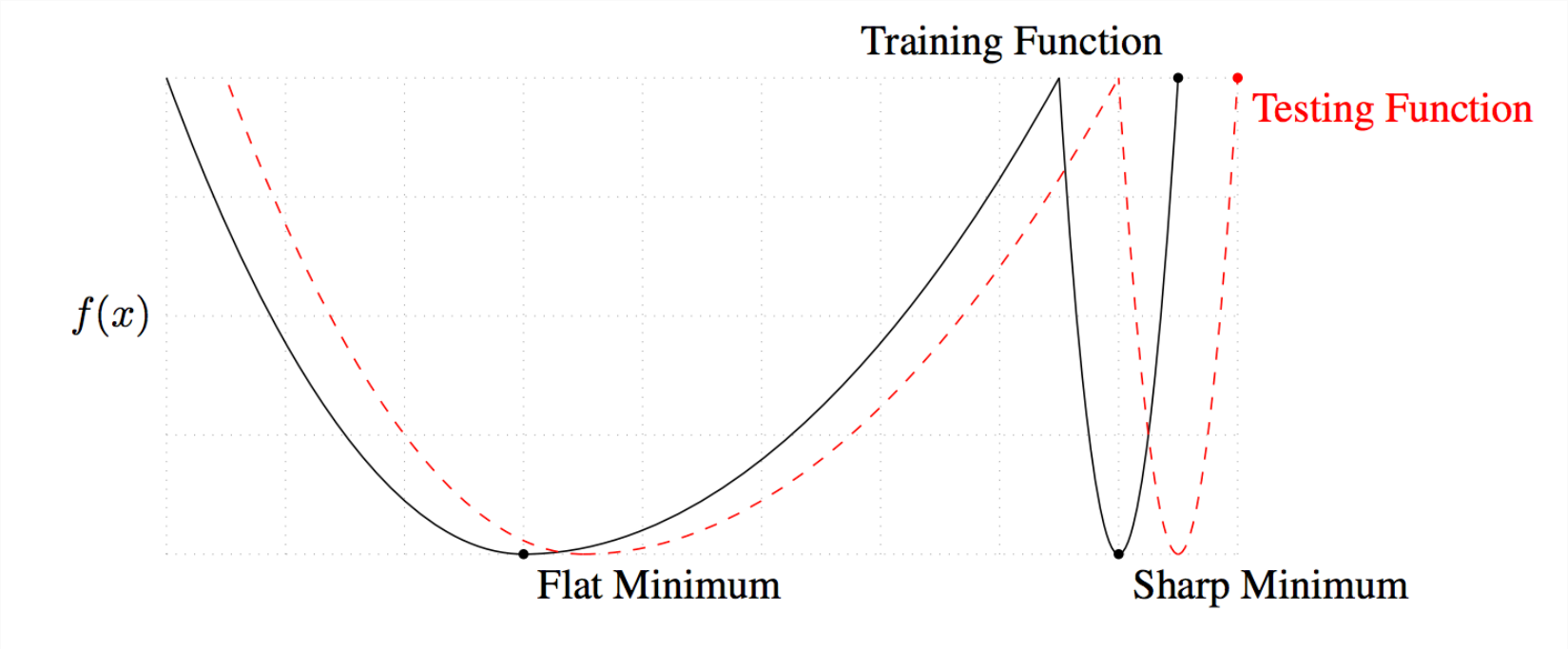


(a) Network F_2



(b) Network C_1

Figure 2: Training and testing accuracy for SB and LB methods as a function of epochs.


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