

# DEEP LEARNING

## Введение

---

Святослав Елизаров, Борис Коваленко, Артем Грачев

7 ноября 2017

Высшая школа экономики

# ОСНОВНЫЕ ПОНЯТИЯ

---

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

Множество  $Y$  – это множество значений целевого признака.

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

Дано:

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

Задача: Предсказать значения  $\tilde{f}$  для всего множества  $\Omega$ , 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  $\Omega$  – 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  $Q(Y|X)$

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

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

## **С учителем:**

1. Классификация
2. Регрессия
3. Сегментация изображений

## **Без учителя:**

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

И многими другими...

# ЛИНЕЙНЫЕ МОДЕЛИ

---

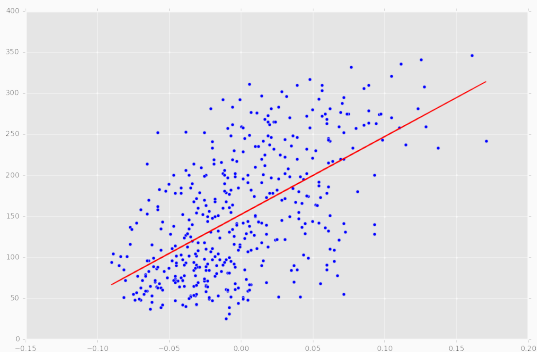


# ЛИНЕЙНАЯ РЕГРЕССИЯ

Простейшая линейная регрессия:

$$f(x) = w^T x + b$$

Здесь  $x$  — входной вектор признаков,  $w$ ,  $b$  — настраиваемые параметры



# MULTICLASS SVM (HINGE LOSS)

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



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

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

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

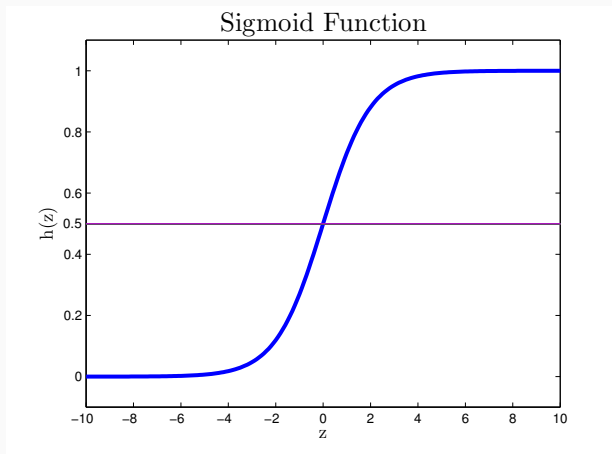
Логистическая регрессия:

$$f(x) = \sigma(w^T x + b)$$

Где  $\sigma(z) = \frac{1}{1+e^{-z}}$

# ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ

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  $x^2$ :

$$f(x) = \sigma (w_1x + w_2x^2 + b)$$

$$z_{k,i} = w_k^T x_i + b; \quad z_k = w_k^T x + b$$

Функция softmax переводит оценки в "вероятности":

$$P(y = k | x_i) = \text{softmax}(z_{k,i}) = \frac{e^{z_k}}{\sum_m e^{z_{m,i}}}$$

Лосс-функция для логистической регрессии для элемента  $x$ :

$$L(x) = - \sum_{j=1}^K \mathbb{I}(y_j = 1) \log \left( \frac{e^{z_j}}{\sum_m e^{z_m}} \right) = - \log \left( \frac{e^{z_{true}}}{\sum_m e^{z_m}} \right)$$

Количество информации:

$$I(x) = -\log_2 p(x)$$



Количество информации:

$$I(x) = -\log_2 p(x)$$

Информационная энтропия:

$$H(p) = -\sum_x p(x) \log_2 p(x) = \mathbb{E}I(x)$$

Количество информации:

$$I(x) = -\log_2 p(x)$$

Информационная энтропия:

$$H(p) = -\sum_x p(x) \log_2 p(x) = \mathbb{E}I(x)$$

Кросс-энтропия:

$$H(p, q) = -\sum_x p(x) \log_2 q(x)$$

Кросс-энтропия :

$$\begin{aligned} H(p, q) &= - \sum_x p(x) \log q(x) = - \sum_x p(x) (\log q(x) + \log p(x) - \log p(x)) = \\ &H(p) + \sum_x p(x) (\log p(x) - \log q(x)) \end{aligned}$$

Дивергенция Кульбака—Лейблера:

$$D_{KL}(p||q) = \sum_x p(x)(\log p(x) - \log q(x))$$

Дивергенция Кульбака—Лейблера:

$$D_{KL}(p||q) = \sum_x p(x)(\log p(x) - \log q(x))$$

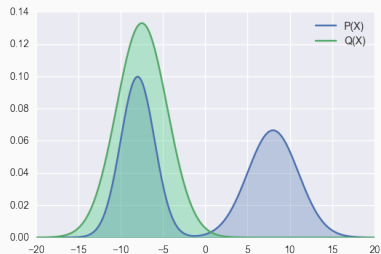
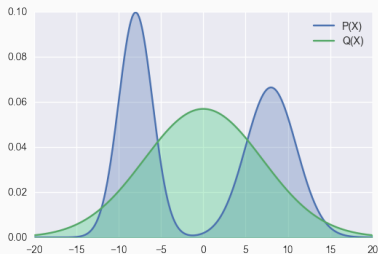
- $\forall p, q, D_{KL}(p||q) \geq 0$
- $D_{KL}(p||q) \neq D_{KL}(q||p)$

Дивергенция Кульбака—Лейблера:

Прямая  $D_{KL}(p||q)$  или обратная  $D_{KL}(q||p)$ ?

Дивергенция Кульбака—Лейблера:

Прямая  $D_{KL}(p||q)$  или обратная  $D_{KL}(q||p)$ ?



Важной величиной является **взимная информация** (mutual information).

Рассмотрим две случайные величины  $x$  и  $y$ , а так же их совместное распределение  $p(x, y)$



Важной величиной является **взимная информация** (mutual information).

Рассмотрим две случайные величины  $x$  и  $y$ , а так же их совместное распределение  $p(x, y)$

Если случайные величины независимы, то

$$p(x, y) = p(x)p(y)$$

$$p(x) = p(x|y)$$

$$p(y) = p(y|x)$$

$$D_{KL}(p(x, y) || p(x)p(y))$$

Называется взаимной информацией и является мерой зависимости двух случайных величин.

Взаимная информация обозначается  $I(x; y)$ .

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 -  $[0, \dots, 1, \dots, 0]$

Предсказанное распределение -  $[0.05, \dots, 0.6, \dots, 0.1]$

Пример:

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

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

В чем отличия hinge loss и cross entropy loss?

Пример:

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

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

В чем отличия hinge loss и cross entropy loss?

На практике получаемые результаты очень похожи

Пример:

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

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

В чем отличия hinge loss и cross entropy loss?

На практике получаемые результаты очень похожи

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

Опишем в общем виде модель линейной регрессии:

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

где

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

Теперь введём следующую модель:

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

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

Где  $\phi_i$  – 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  $x$ .

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



Искусственные нейронные сети прямого распространения (feed-forward artificial neural networks):.

- В качестве базисных функций  $\phi_i(x)$  возьмём эту же 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  $x \in R^3$ :

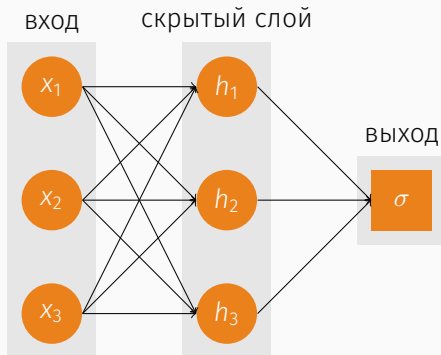
$$f(x) = \sigma(w_1^2 \sigma(w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3) + \dots + w_3^2 \sigma(w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3))$$

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

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

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  $h_i = \sigma(w_{i1}^1 x_1 + w_{i2}^1 x_2 + w_{i3}^1 x_3)$

В качестве функций активации обычно используют:

- Уже хорошо знакомый нам  $\sigma(x)$
- Гиперболический тангенс  $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} = 2\sigma(2x) - 1$
- Линейный выпрямитель  $ReLU(x) = \max(0, x)$
- $LReLU(x) = \max(x, \alpha x)$ , где  $\alpha \leq 1$

# ОБУЧЕНИЕ НЕЙРОННЫХ СЕТЕЙ

---

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.

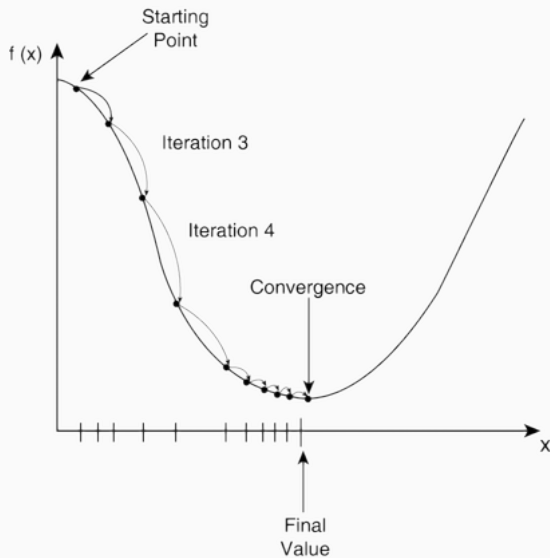
Таким образом мы можем применять основанные на вычислении градиента методы, не боясь застрять в локальном минимуме. Так же важным является то, что мы можем вычислить Гессиан, если необходимо.

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



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

Где  $\theta_n$  – вектор параметров функции  $f$  на итерации  $n$ .

$\lambda$  – learning rate, может быть как константой, так и функцией от номера итерации.

Для искусственных нейронных сетей в общем виде требование выпуклости функции потерь не соблюдается (почему?).

## Проблемы:

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

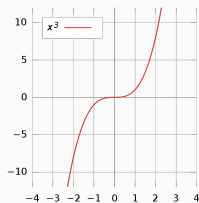
1. Все локальные минимумы примерно одинаковы (в некоторых случаях являются глобальными минимумами)
2. Чем глубже модель, тем меньше вероятность встретить плохой локальный минимум
3. Любая критическая точка, не являющаяся минимумом, является седловой

Статьи:

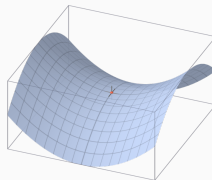
1. LeCun et al., 2014. The Loss Surfaces of Multilayer Networks
2. Kenji Kawaguchi, 2016. Deep Learning without Poor Local Minima

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.

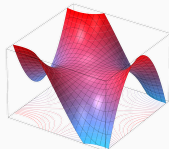
# СЕДЛОВЫЕ ТОЧКИ



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



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



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

Идеи?



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.

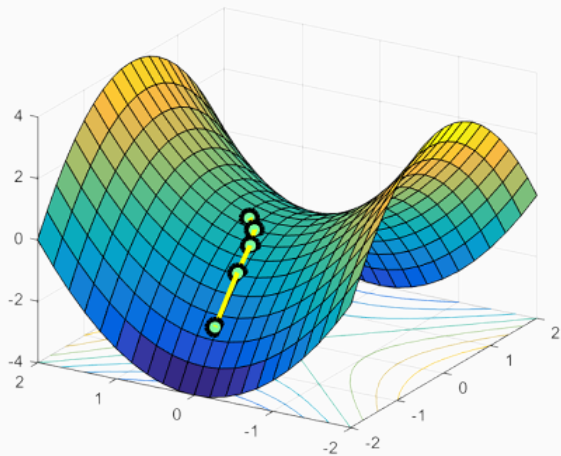
$$\theta_{n+1} = \theta_n - \lambda \nabla f(\theta) + \epsilon$$

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

---

- Модели на основе искусственных нейронных сетей могут иметь сотни 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.

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.

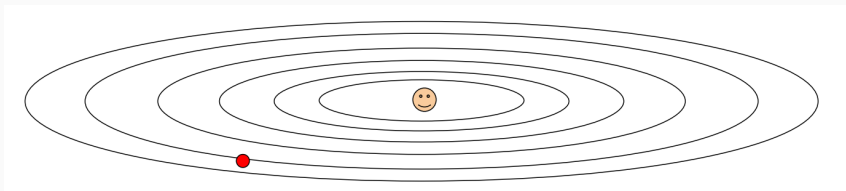
На практике производят корректировку коэффициентов сети с использованием градиента, который аппроксимируется градиентом функции потерь, вычисленной только на случайном подмножестве обучающей выборки (mini batch).

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

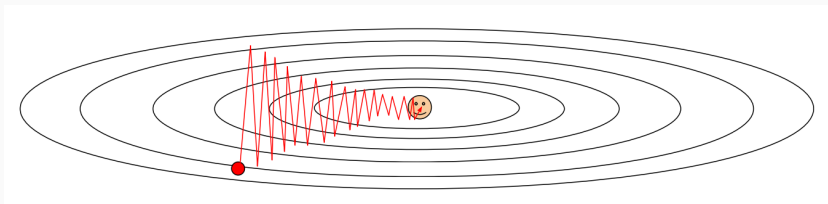
$$L(W) = \frac{1}{N} \sum_i^N L_i(W, x_i, y_i)$$

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

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

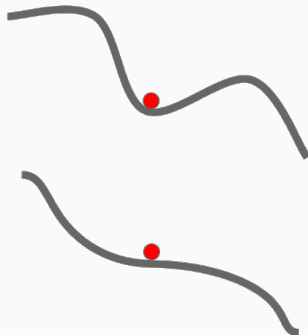


Что будет с Vanilla SGD если линии уровня функции потерь выглядят так:



Как можно улучшить алгоритм?

Не забываем о локальных минимумах и седловых точках!



Как можно улучшить алгоритм?



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:

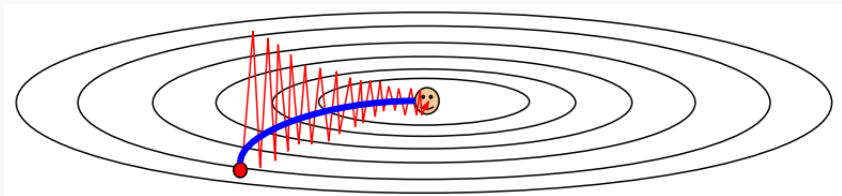
$$W_{t+1} = W_t - \lambda \nabla L(W_t, \dots)$$

Импульсный метод

$$v_{t+1} = \rho v_t + \nabla L(W_t, \dots)$$

$$W_{t+1} = W_t - \alpha v_{t+1}$$

Метод даёт существенный прирост в скорости сходимости. Более подробно про метод можно прочесть в онлайн-журнале [distill.pub](https://distill.pub/):  
Gabriel Goh 2017. Why Momentum Really Works



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

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

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

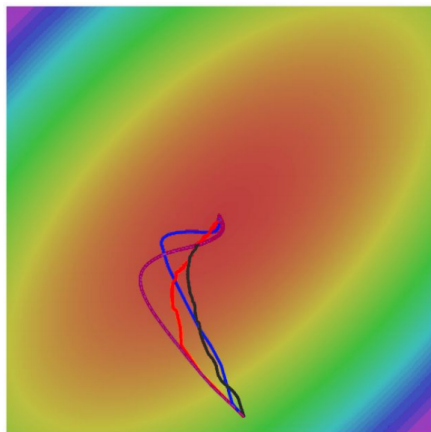
Какие могут быть проблемы? Как их решить?

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

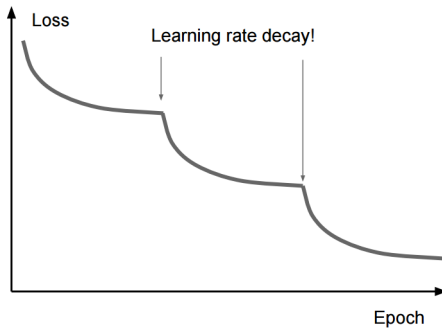
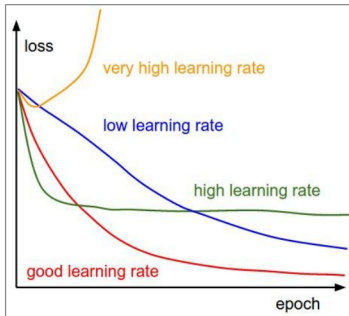
$$W_{t+1} = W_t - \frac{\lambda \nabla L(W_t, \dots)}{\sqrt{Cache_{t+1} + 1e^{-10}}}$$

$\gamma$  - Параметр затухания для истории градиентов, учитываем только окно недавних градиентов

$$Adam = RMSProp + Momentum$$



- SGD
- SGD+Momentum
- RMSProp
- Adam



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.



Вычислительные графы позволяют:

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

Вычислительные графы используются в большинстве современных библиотек для deep learning.

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

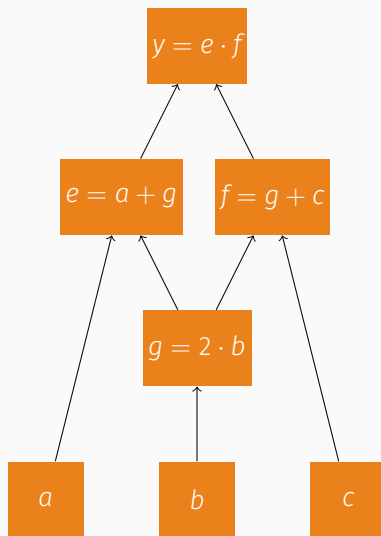
Она состоит из четырёх операций, следовательно в графе будет четыре вершины (и три входа). Выпишем их:

$$g = 2 \cdot b$$

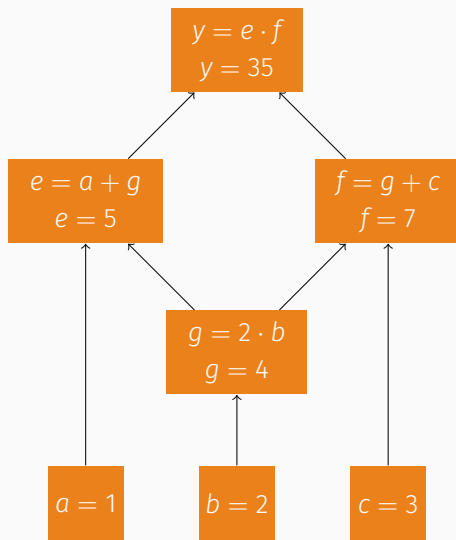
$$e = a + g$$

$$f = g + c$$

$$y = e \cdot f$$



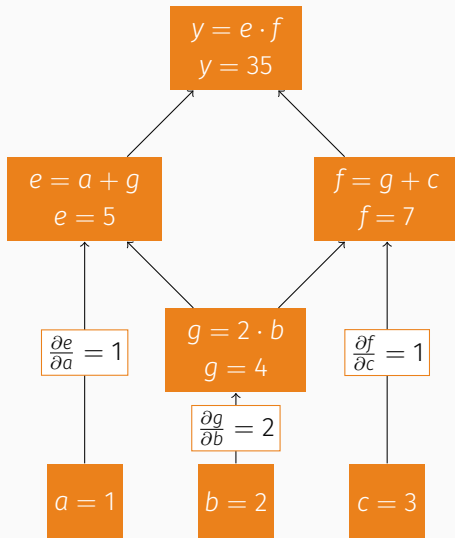
# ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



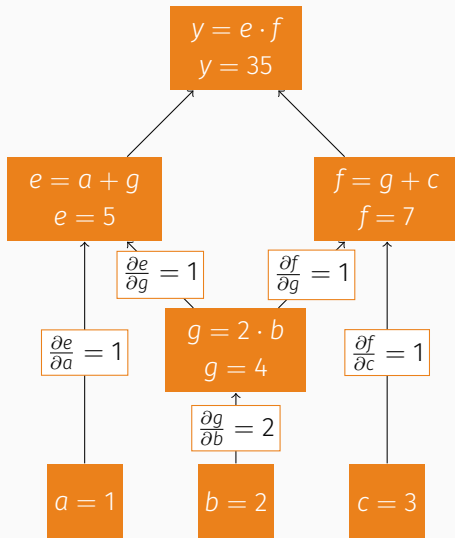
Как видно из примера, значения узла  $g$  было рассчитано один раз, но использовалось дважды.

Теперь вычислим градиент функции  $u$ .

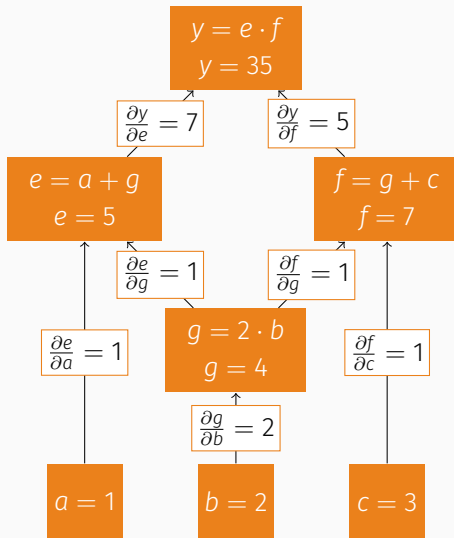
# ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



# 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?

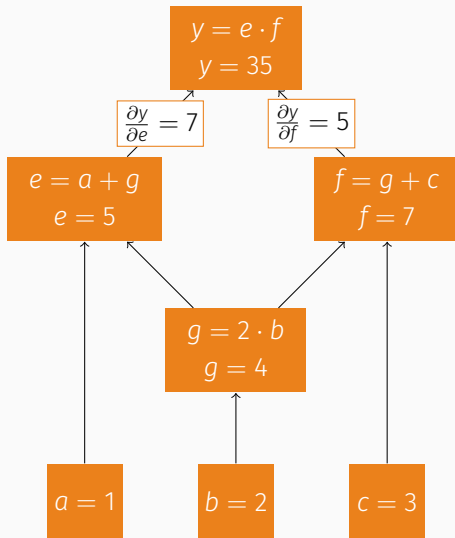
Что с этим делать?

**Что с этим делать?** Применим динамическое программирование!

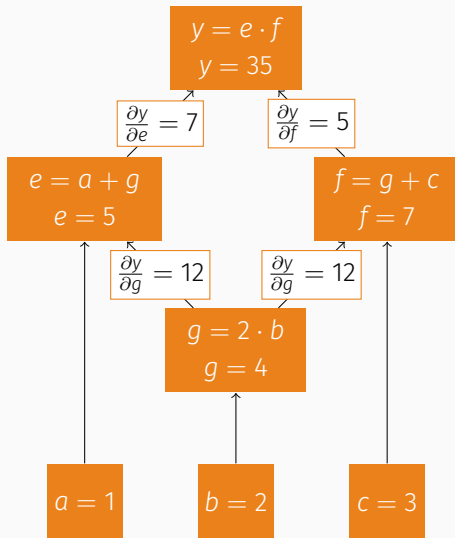
Будем считать частные производные с конца, используя полученную на предбудущих шагах информацию для вычисления значений.

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

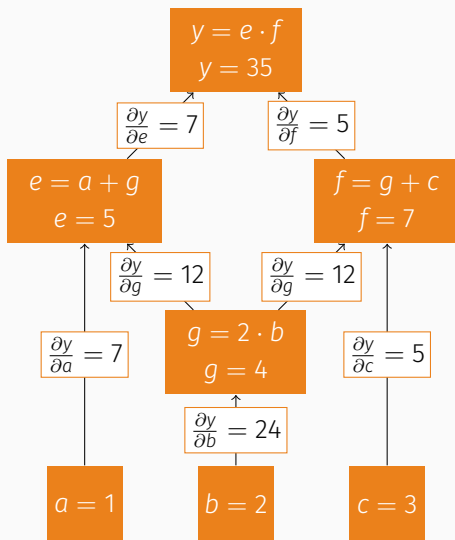
# ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



# ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



# ВЫЧИСЛИТЕЛЬНЫЙ ГРАФ



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

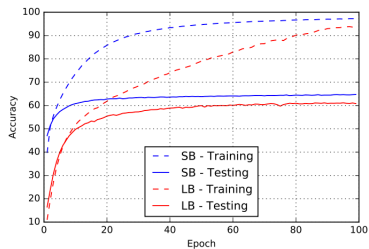
`peter.tang@intel.com`



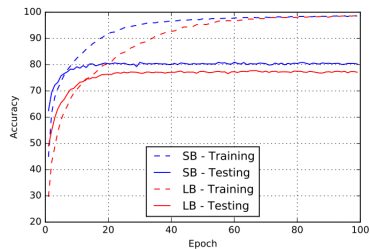
Table 1: Network Configurations

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.

