Лекция 7

Автоматическое дифференцирование и «нейронные» сети

Машинное обучение

Алексей Забашта

16.10.2020

План лекции

- Сведение задачи обучения к задаче дифференцирования
- Дифференцирование составных функций по графу вычислений
- Дифференцирование сложных функций
- Soft-Max и Soft-Arg-Max
- История нейронных сетей
- Функции активации
- Дропаут
- Дополнительные темы

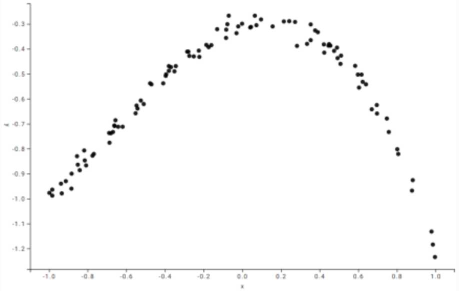
- Слайды доступны: **shorturl.at/ltVZ3**
- Видео доступны: **shorturl.at/hjyAX**

План лекции

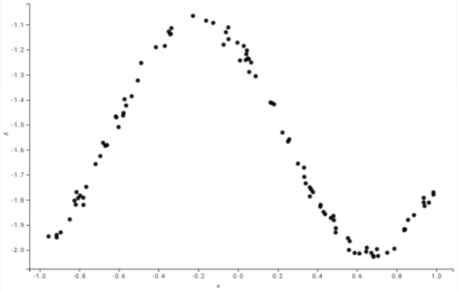
- Сведение задачи обучения к задаче дифференцирования
- Дифференцирование составных функций по графу вычислений
- Дифференцирование сложных функций
- Soft-Max и Soft-Arg-Max
- История нейронных сетей
- Функции активации
- Дропаут
- Дополнительные темы

Выбор модели для данных

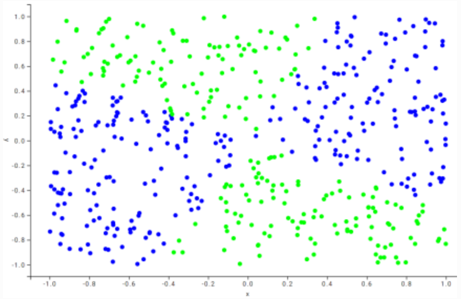
Пример моделей с обучаемыми параметрами



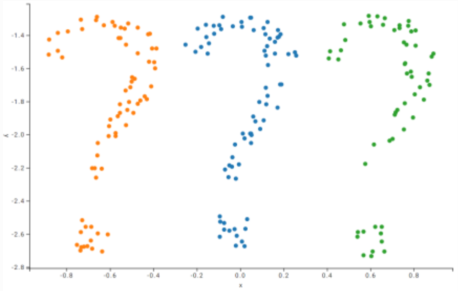
Какой-то полином.
Модель: $y(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 \ldots$
Обучаемые параметры: $a_0, a_1, a_2 \ldots$



Синусоида.
Модель: $\mathrm{y}(x) = \sin(a_0 \cdot x + a_1) \cdot a_2 + a_3$
Обучаемые параметры: a_0, a_1, a_2, a_3



Кривая второго порядка.
Модель: $\mathrm{a}_0\mathrm{y}^2 + \mathrm{a}_1\mathrm{x}^2 + a_2 yx + a_3 y + a_4 x + a_5 > 0$
Обучаемые параметры: $a_0, a_1, a_2, a_3, a_4, a_5$



Неизвестная модель — берём «мощную» модель.
Модель: $\mathrm{f}(x, y, a) = (p_1, p_2, p_3)$
Обучаемые параметры: $a_0, a_1, a_2 \ldots$

Обучение функции

- Обучаемая функция: $f(x_1, x_2, \dots, x_n, a_1, a_2, \dots, a_m): \mathbb{R}^{n+m} \to \mathbb{R}^k$, где n — число признаков, m — число обучаемых параметров, k — число предсказываемых (целевых) признаков.
- Набор данных: $D = \{(\overrightarrow{x_i}, \overrightarrow{y_i})\}$.
- Функция ошибки: $E\left(\hat{y}_1, \dots, \hat{y}_{|D|}, y_1, \dots, y_{|D|}\right)$, где $\hat{y}_i$ — предсказанный вектор для i-го объекта, а y_i — реальный. Например $\mathrm{MSE}\left(\hat{y}_1, \dots, \hat{y}_{|D|}, y_1, \dots, y_{|D|}\right) = \frac{1}{|D| \cdot k} \sum_{i=1}^{|D|} \sum_{j=1}^{k} \left(y_{i,j} - \hat{y}_{i,j}\right)^2$

- Режим обучения: $E_{train}(a_1, \dots, a_m) = E\left(f(\overrightarrow{x_1}, \vec{a}), \dots, f\left(\overrightarrow{x_{|D|}}, \vec{a}\right), y_1, \dots, y_{|D|}\right)$. Переменные x зафиксированы функцией ошибки. Переменные a изменяются в процессе минимизации.

- Режим предсказания: $f_{predict}(x_1, \dots, x_n) = f(x_1, \dots, x_n, a_1, \dots, a_m)$. Переменные x изменяются от запроса к запросу. Переменные a зафиксированы в процессе обучения.

Мягкая классификация

Обучаемая функция: $f(x_1, x_2, \dots, x_n, a_1, a_2, \dots, a_m) = \vec{y} = (p_1, p_2, \dots, p_k)$, где $(p_1, p_2, \dots, p_k)$ распределение вероятностей.

- Наивный подход: взять в качестве функции ошибки любую функцию ошибки для задачи восстановления регрессии.
- Использовать функцию ошибки для задачи классификации: Рассмотрим F_1-меру: $F_1 : CM \to \mathbb{R}$, где CM — матрица неточностей.
  - F_1-мера — дифференцируемая функция
  - При вычислении F_1-меры не требуется, чтобы CM содержала только целые числа.

  Следовательно можно напрямую «не округляя» прибавлять вектор $\vec{p}$ к соответствующей реальному классу строке матрицы CM и вычислять F_1-меру, либо другую подобную функцию.
- Использовать перекрёстную энтропию: $H(p, q) = -\sum_{i=1}^{k} q_i \cdot \log p_i$, где q — реальное распределение вероятностей, а p — предсказанное. Для классификации вырождается в метод максимального правдоподобия, так как $q = (0, \dots, 1, \dots, 0)$.

План лекции

- Сведение задачи обучения к задаче дифференцирования
- Дифференцирование составных функций по графу вычислений
- Дифференцирование сложных функций
- Soft-Max и Soft-Arg-Max
- История нейронных сетей
- Функции активации
- Дропаут
- Дополнительные темы

Словарь неправильных терминов (1)

Неправильно	Правильно
Сложные функции	Составные функции*

Для удобства будем различать эти термины:

- **Составная функция** – функция, которая описывается графом вычислений.
- **Сложная функция** – функция, тип значения которой отличен от скаляра, может быть вектором, матрицей и т.д.

* Взято из «Зубодробящие факты и смешные картинки» https://vk.com/wall-87397140_1128

Наивное вычисление функции

Если собрать из нескольких функций одну, то её размер может экспоненциально расти.

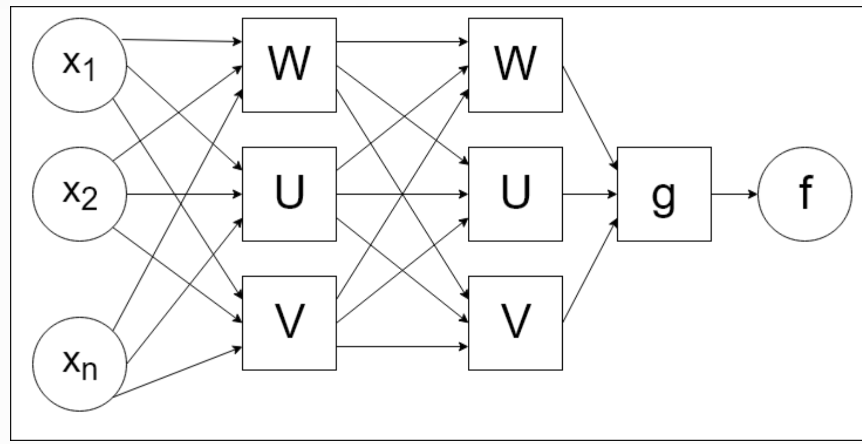
```
f(x1, x2 ... xn):
    a1 = w(x1, x2 ... xn)
    a2 = u(x1, x2 ... xn)
    a3 = v(x1, x2 ... xn)

    b1 = w(a1, a2, a3)
    b2 = u(a1, a2, a3)
    b3 = v(a1, a2, a3)

    return g(b1, b2, b3)
```

```
f=g(w(w(x1,x2 ... xn),
      u(x1,x2 ... xn),
      v(x1,x2 ... xn)),
    u(w(x1,x2 ... xn),
      u(x1,x2 ... xn),
      v(x1,x2 ... xn)),
    v(w(x1,x2 ... xn),
      u(x1,x2 ... xn),
      v(x1,x2 ... xn)))
```

Представление функции сетью (граф вычислений)

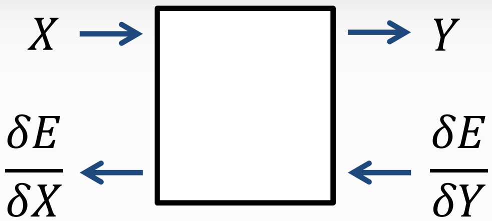
Будем хранить историю вычисления функции в виде направленного ацикличного графа.

```
f(x1, x2 ... xn):
    a1 = w(x1, x2 ... xn)
    a2 = u(x1, x2 ... xn)
    a3 = v(x1, x2 ... xn)

    b1 = w(a1, a2, a3)
    b2 = u(a1, a2, a3)
    b3 = v(a1, a2, a3)

    return g(b1, b2, b3)
```



- **Статический граф** вычислений строится до вычисления функции. Можно заранее соптимизировать.
- **Динамический граф** вычислений строится во время вычисления функции. Более гибкий.

Элементарные дифференцируемые блоки



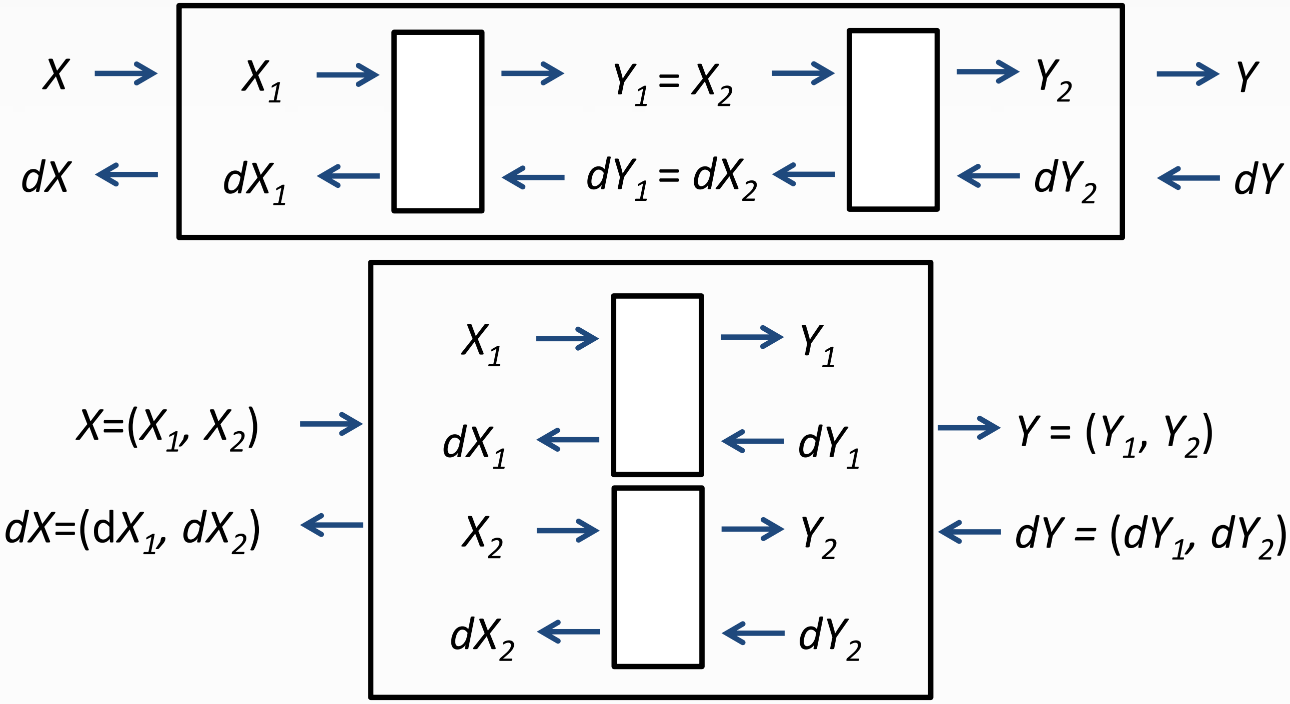
Пример для умножения *a* на *b*:

```
def mul(a,b):
    def d_mul(dc):
        return (b * dc, a * dc)
    return a * b, d_mul
```

- Тип и размер X совпадает с $\frac{\delta E}{\delta X}$, а Y с $\frac{\delta E}{\delta Y}$.
- Блок «по желанию» может запоминать X, Y и любые другие промежуточные значения, необходимые для пересчёта производной.
- Блоку для пересчёта производной целевой функции не нужно знать всю функцию целиком.

Композиции блоков

Блоки могут состоять из других блоков, например:



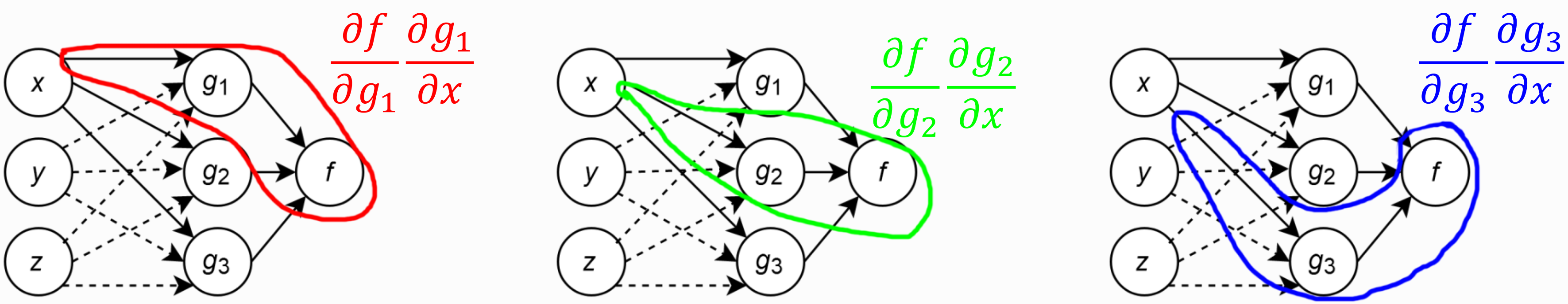
dX – это **сокращение** для $\frac{\delta E}{\delta X}$

$dX = df_{X \to Y}(dY)$ – пересчёт производной из dY в dX

Цепное правило и обобщение

$$(f(g(x)))' = g'(x) \cdot f'(g(x))$$

$$\frac{\partial f}{\partial x} = \frac{\partial g}{\partial x}\frac{\partial f}{\partial g} \qquad \frac{\partial f(g_1(x), g_2(x), \ldots, g_n(x))}{\partial x} = \sum_i \frac{\partial f}{\partial g_i}\frac{\partial g_i}{\partial x}$$



- Если вершина это простая функция, то производная через неё пересчитывается по стандартному цепному правилу.
- Если воспользовались вершиной несколько раз, то производную нужно просуммировать по всем использованиям. Это следует из обобщённого цепного правила. Для сложной функции суммироваться будут не скаляры, а векторы, матрицы и т.д.

Алгоритм дифференцирования по графу вычислений

- Вычисляем составную функцию Q сохраняя граф вычислений $G = (V, E)$: $(u, v) \in E$, если вершина(блок) v напрямую зависит от u
- Для всех вершин v устанавливаем:

$$\frac{\partial Q}{\partial v} = I[v = Q]$$

- Для каждого ребра $(u, v) \in E$ в обратном порядке вычисления целевой функции:

$$\frac{\partial Q}{\partial u} \mathrel{+}= \frac{\partial Q}{\partial v} \cdot \frac{\partial u}{\partial v} \quad \text{или} \quad \frac{\partial Q}{\partial u} \mathrel{+}= df_{u \to v}\left(\frac{\partial Q}{\partial v}\right)$$

если v простая функция

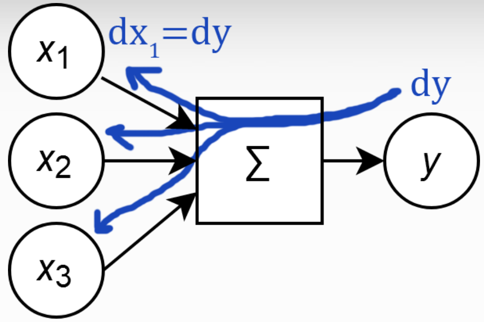
если v сложная функция (вычисляется абстрактным блоком)

Пример дифференцирования для простых функций

Сумма:

$$y = \sum x_i$$

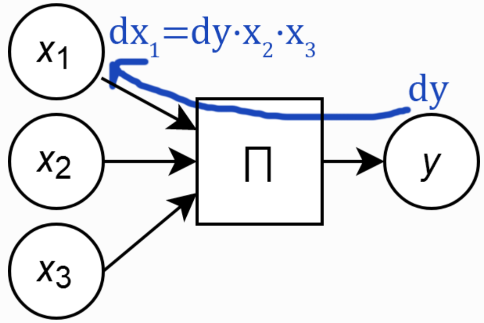
$$dx_i = dy$$



Произведение:

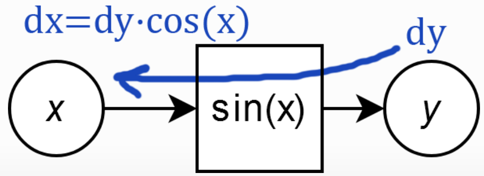
$$y = \prod x_i$$

$$dx_i = dy \cdot \prod_{j \neq i} x_j$$



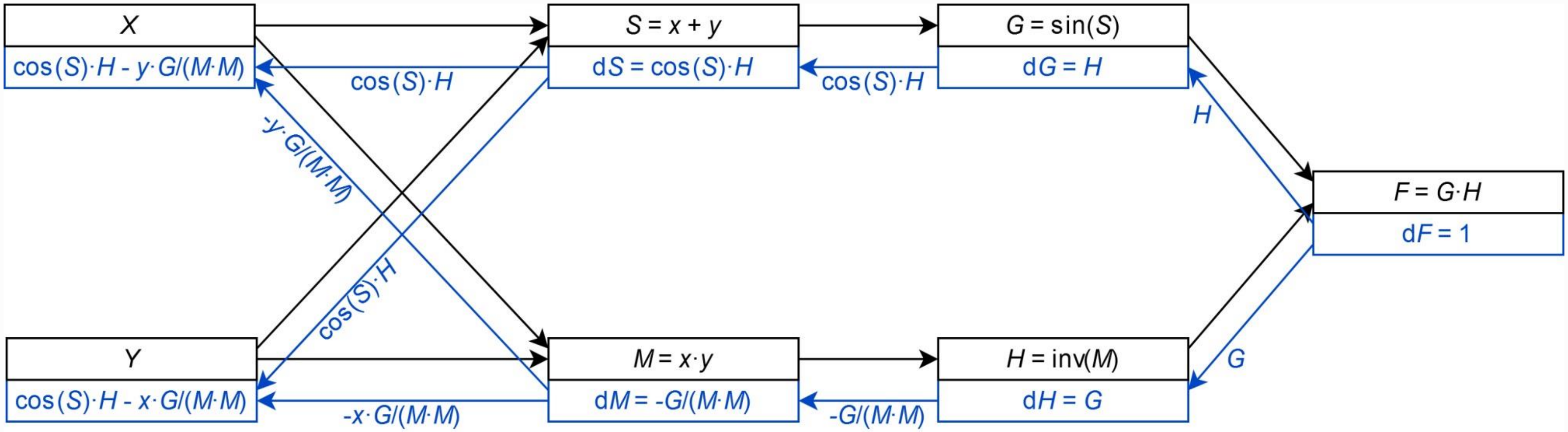
Применение функции:

$$y = f(x)$$
$$dx = dy \cdot f'(x)$$



Пример дифференцирования

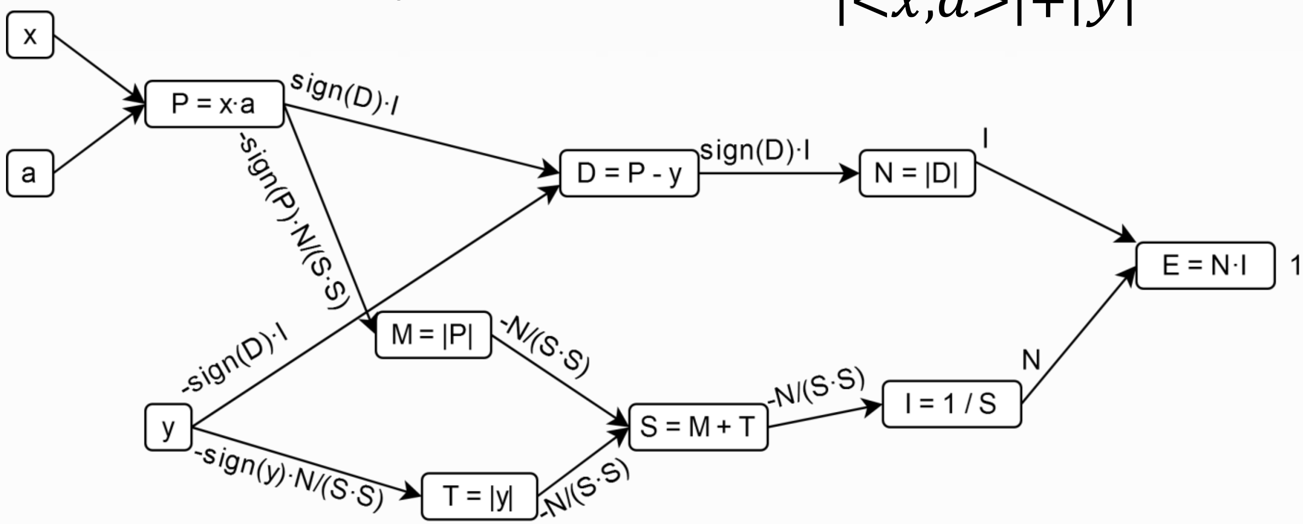
- Рассмотрим функцию $F = \frac{\sin(x+y)}{x \cdot y}$



- $\frac{\partial F}{\partial x} = \cos(S) \cdot H - y \cdot \frac{G}{M^2} = \frac{\cos(x+y)}{x \cdot y} - y \cdot \frac{\sin(x+y)}{(x \cdot y)^2}$

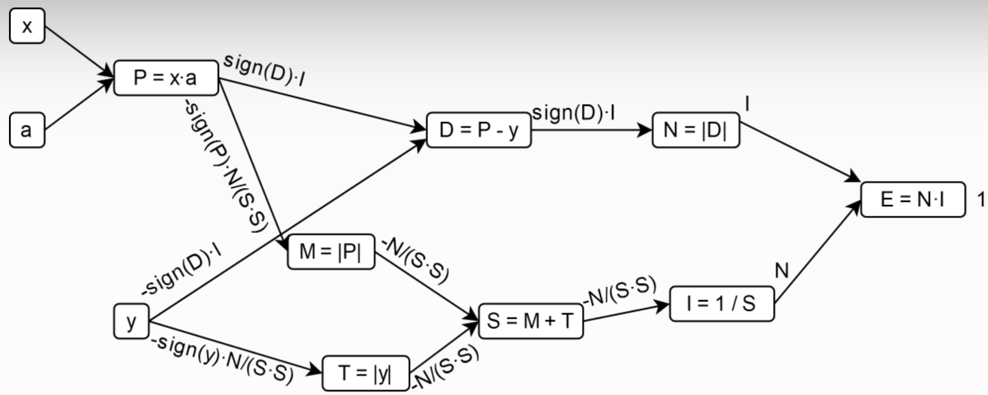
Пример для линейной регрессии со SMAPE

- Рассмотрим функцию $E = \frac{|<x,a>-y|}{|<x,a>|+|y|}$



- $\frac{\partial E}{\partial P} = sign(D) \cdot I - sign(P) \cdot \frac{N}{S^2} = \frac{sign(<x,a>-y)}{|<x,a>|+|y|} - \frac{sign(<x,a>) \cdot |<x,a>-y|}{(|<x,a>|+|y|)^2}$
- $\frac{\partial E}{\partial a_i} = x_i \cdot \frac{\partial E}{\partial P}$

Линеаризованная версия графа SMAPE



x	$dx_i = da_i \cdot dP$
a	$da_i = dx_i \cdot dP$
y	$dy = -sign(D) \cdot I - sign(y) \cdot N/(S \cdot S)$
$P = \langle x, a \rangle$	$dP = sign(D) \cdot I - sign(P) \cdot N/(S \cdot S)$
$M = \lvert P \rvert$	$dM = -N/(S \cdot S)$
$T = \lvert y \rvert$	$dT = -N/(S \cdot S)$
$D = P - y$	$dD = sign(D) \cdot I$
$S = M + T$	$dS = -N/(S \cdot S)$
$N = \lvert D \rvert$	$dN = I$
$I = 1/S$	$dI = N$
$E = N \cdot I$	$dE = 1$

План лекции

- Сведение задачи обучения к задаче дифференцирования
- Дифференцирование составных функций по графу вычислений
- Дифференцирование сложных функций
- Soft-Max и Soft-Arg-Max
- История нейронных сетей
- Функции активации
- Дропаут
- Дополнительные темы

Сложное скалярное произведение

```
for i ...
   for j ...
      for k ...
         if (...)
            x = f(i,j,k)
            y = g(i,j,k)
            z = h(i,j,k)

            c[z] += a[x] * b[y]
```

```
for i ...
   for j ...
      for k ...
         if (...)
            x = f(i,j,k)
            y = g(i,j,k)
            z = h(i,j,k)

            da[x] += dc[z] * b[y]
            db[y] += dc[z] * a[x]
```

Само скалярное произведение может содержаться внутри любого «сложного» кода. Для пересчёта производной код копируется напрямую и в нём заменяется одно скалярное произведение на два. Важно, чтобы поток выполнения **не зависел** от a[x], b[y], c[z]!

Пример сложного скалярного произведения — операция свёртки в свёрточных сетях.

Произведение матриц

```
for (x = 1 ... n):
    for (y = 1 ... m):
        for (z = 1 ... k):
            c[x][z] += a[x][y] * b[y][z]

for (x = 1 ... n):
    for (y = 1 ... m):
        for (z = 1 ... k):
            da[x][y] += dc[x][z] * b[y][z]

for (y = 1 ... m):
    for (z = 1 ... k):
        for (x = 1 ... n):
            db[y][z] += a[x][y] * dc[x][z]
```

$$C_{[n,k]} = A_{[n,m]} \cdot B_{[m,k]}$$
$$dA_{[n,m]} = dC_{[n,k]} \cdot B^{\top}_{[k,m]}$$
$$dB_{[m,k]} = A^{\top}_{[m,n]} \cdot dC_{[n,k]}$$

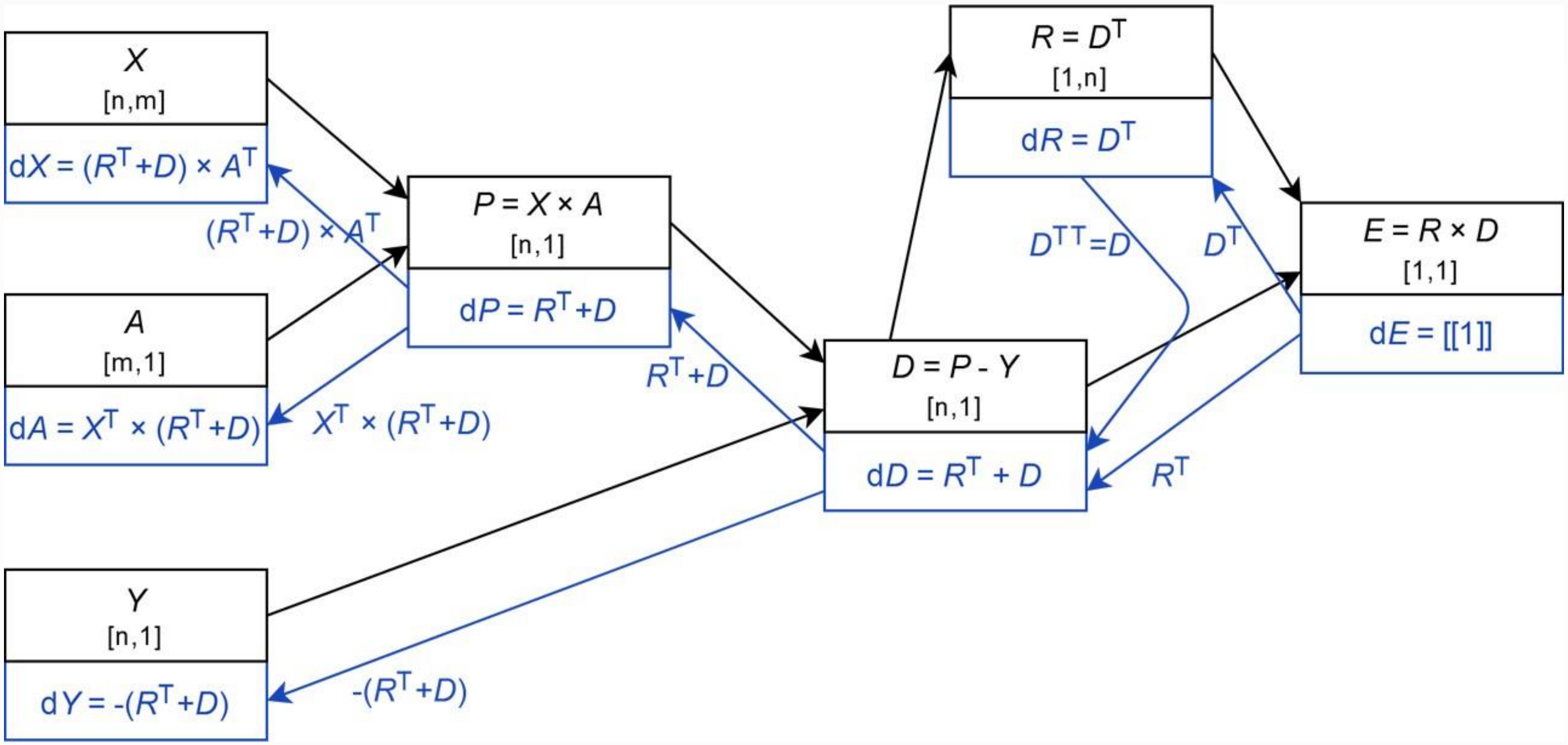
- Произведение матриц можно представить, как сложное скалярное произведение.
- Пересчёт производной можно представить как произведение матриц.

Другие матричные функции

- Сумма:
$$Y = \sum_i X_i, dX_i = dY$$
- Произведение Адамара (покомпонентное):
$$Y = X_1 \circ X_2 \circ \cdots \circ X_n$$
$$dX_i = X_1 \circ \cdots \circ X_{i-1} \circ dY \circ X_{i+1} \circ \cdots \circ X_n$$
- Функция (покомпонентное применение):
$$Y = f(X), Y_{i,j} = f\left(X_{i,j}\right)$$
$$dX = f'(X) \circ dY$$
- Обратная матрица:
$$Y \ = X^{-1}$$
$$dX = -X^{-T} \times dY \times X^{-T} = -Y^T \times dY \times Y^T$$
- Разложение Холецкого (корень из матрицы):
$$Y = X^T \times X, Z(Y) = X$$
$$dY = (X^T + X)^{-1} \times dZ$$

Пример для линейной регрессии с MSE в матричном виде

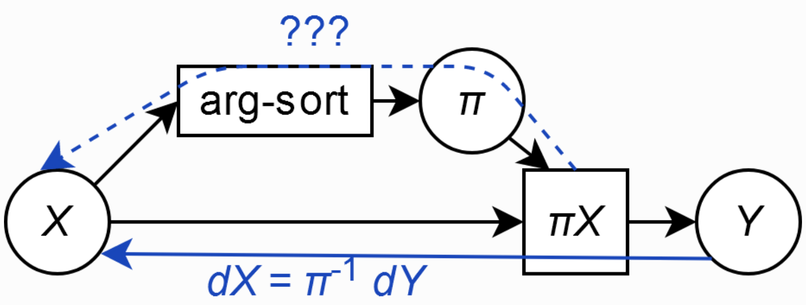
- Рассмотрим функцию $E = (X \times A - Y)^T \times (X \times A - Y)$



- $\frac{\partial E}{\partial A} = X^T \times (R^T + D) = X^T \times (2D) = X^T \times 2(X \times A - Y)$

Перестановки

- Транспонирование матрицы:
$$Y = X^T \Longrightarrow dX = (dY)^T$$
- Перестановка π элементов массива X:
$$Y = \pi X \ \Longrightarrow dX = \pi^{-1} dY,$$
где π^{-1} — это обратная перестановка.
- Сортировка массива: *arg-sort* зависит от массива X, но производная через *arg-sort* не пересчитывается. Поэтому **производная «нечестная»**.



План лекции

- Сведение задачи обучения к задаче дифференцирования
- Дифференцирование составных функций по графу вычислений
- Дифференцирование сложных функций
- Soft-Max и Soft-Arg-Max
- История нейронных сетей
- Функции активации
- Дропаут
- Дополнительные темы

Пересчёт производной для максимума

$$Y = \max(X)$$

- Используя дискретный *arg-max*: $dX_j = I[j = i] \cdot dY$, где $i = argmax(X)$. Предвзят к порядку элементов.
- Используя численный *arg-max*: $Y = \langle argmax(X), X \rangle$ и $dX = argmax(X) \cdot dY$, где $argmax(X) = \left(0, \dots, \frac{1}{m}, \dots, \frac{1}{m}, \dots, 0\right)$, а m — число элементов, на которых достигается максимум. Требует больше времени и памяти.
- Смесь подходов: $dX_j = I\left[X_j = Y\right] \cdot dY$. Если максимум достигается на нескольких элементах, производная будет завышена.

В любом случае **производная будет «нечестная»**, так как *arg-max* зависит от X, но производная через *arg-max* не пересчитывается.

Soft-arg-max

$$y = \text{soft-arg-max}(x), \text{ где } y_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

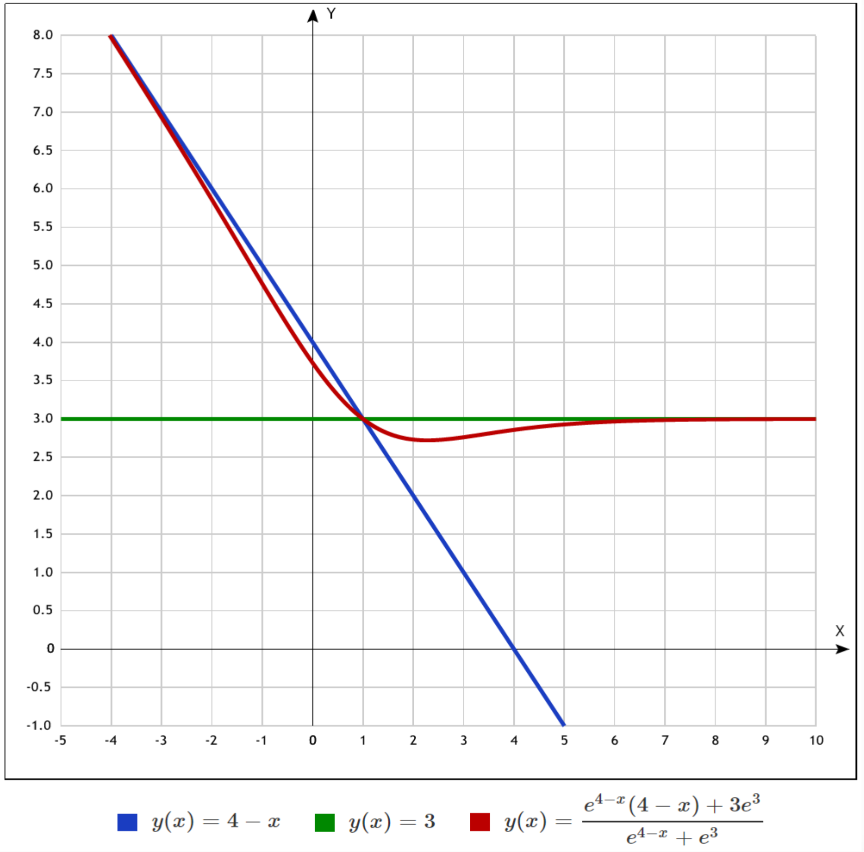
$$\frac{\partial y_i}{\partial x_j} = \begin{cases} y_i(1 - y_j) & i = j \\ -y_i \cdot y_j & i \neq j \end{cases} = y_i\big(I[i = j] - y_j\big)$$

- soft-arg-max вычисляет по вектору чисел вектор с распределением вероятностей.
- Можно интерпретировать как вероятность нахождения максимума в i-й координате.
- soft-arg-max(x - c, y - c, z - c) = soft-arg-max(x, y, z)
- Предыдущее свойство используют для устойчивости вычислений. При $c = \max(x, y, z)$

Плохой Soft-max

$$soft\_max(x_1, \ldots, x_n) = \frac{x_i \cdot e^{x_i}}{\sum_j e^{x_j}} = \langle x, soft\_arg\_max(x_1, \ldots, x_n) \rangle$$

- Гладкая аппроксимация максимума. Мат.ожидание или средневзвешенное, где веса – экспоненты значений соответствующих элементов. Сохраняет некоторые свойства максимума:
- SoftMax(a, a, a) = a
- SoftMax(x + a, y + a, z + a) = SoftMax(x, y, z) + a

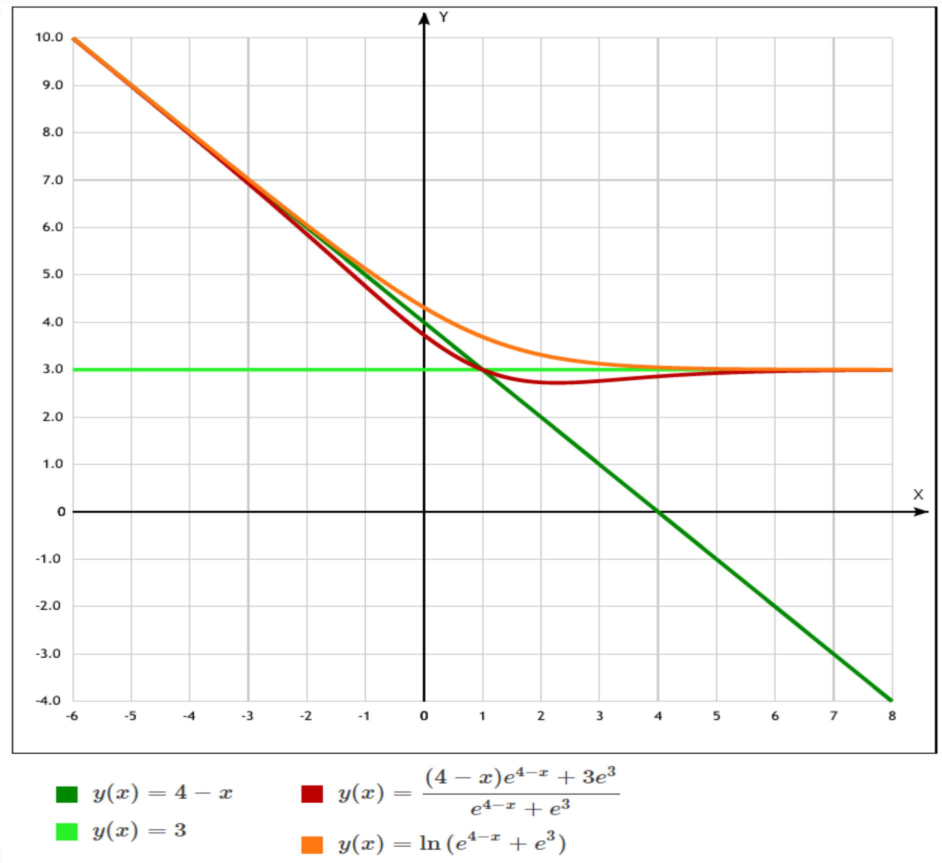


Хороший Soft-max

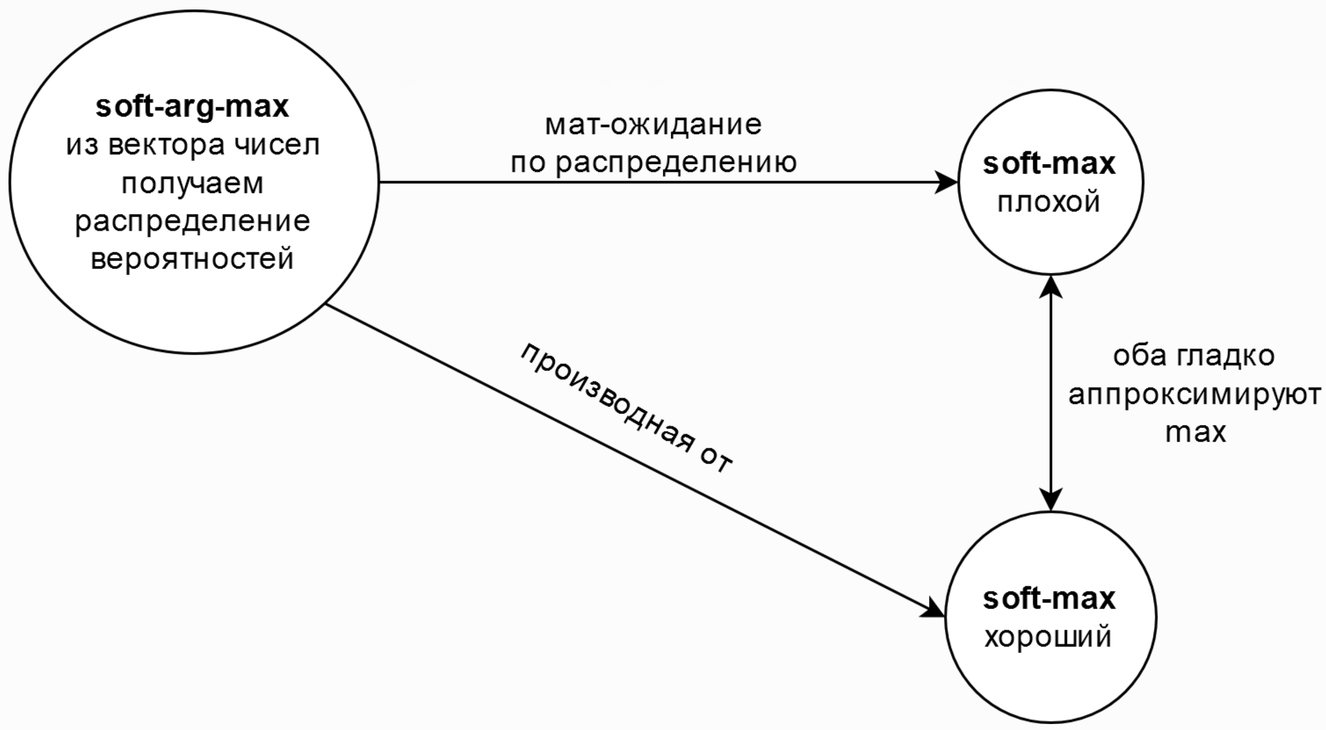
$$soft\_max(x_1, \ldots, x_n) = \log\left(\sum_i e^{x_i}\right)$$

Обычный мир	Логарифмированный
1	0
a^b	a*b
a*b	a+b
a+b	≈ max(a,b)

- Не сохраняет свойство SoftMax(a, a, a) = a
- Производная = Soft-arg-max



Связь Soft-max-в



Словарь неправильных терминов (2)

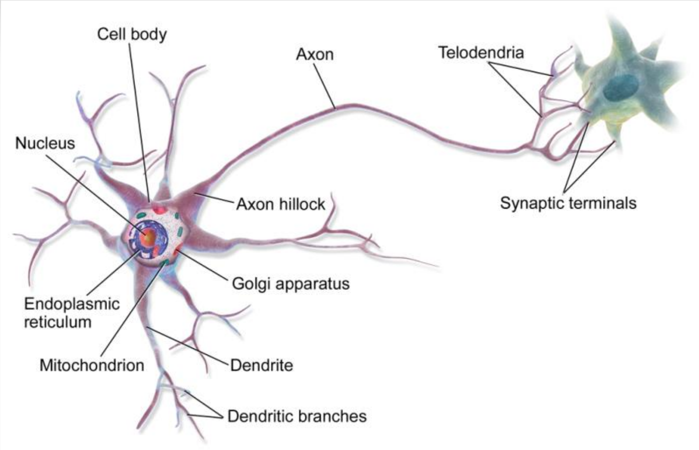
<table>
<tr><th>Неправильно</th><th>Правильно</th></tr>
<tr><td rowspan="3">SoftMax*</td><td>SoftArgMax</td></tr>
<tr><td>Обобщённая (многомерная) сигмоида</td></tr>
<tr><td>Алгоритм подсчёта весов для SoftMax-а</td></tr>
</table>

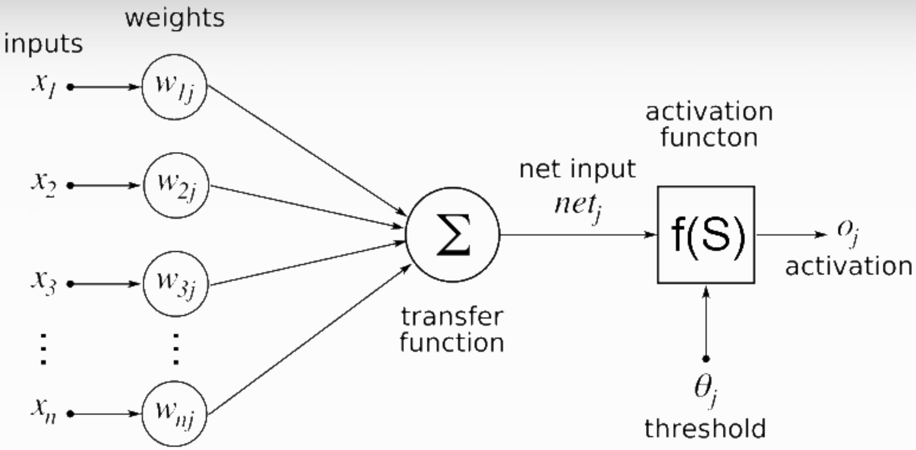
* во многих статьях SoftMax-м почему-то называют SoftArgMax.

План лекции

- Сведение задачи обучения к задаче дифференцирования
- Дифференцирование составных функций по графу вычислений
- Дифференцирование сложных функций
- Soft-Max и Soft-Arg-Max
- История нейронных сетей
- Функции активации
- Дропаут
- Дополнительные темы

Искусственный нейрон (перцептрон)



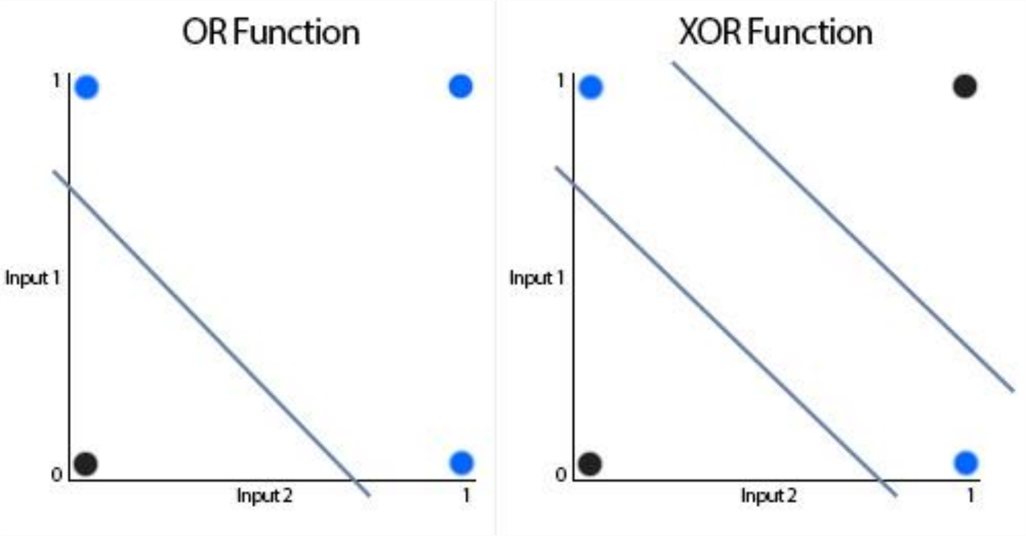


Реальный нейрон	**Искусственный нейрон**
Сигналы от дендритов накапливаются.	Входные значения суммируются.
Не все дендриты одинокого важны.	Значения суммируются с некоторым весом, возможно отрицательным.
Нейрон либо активирован, либо нет.	Применяется функция активации.
Активация происходит после некоторого порога.	Результат суммы сдвигается перед функцией активации.

Разделяющая способность

$$sign(\langle x, w \rangle + w_0)$$

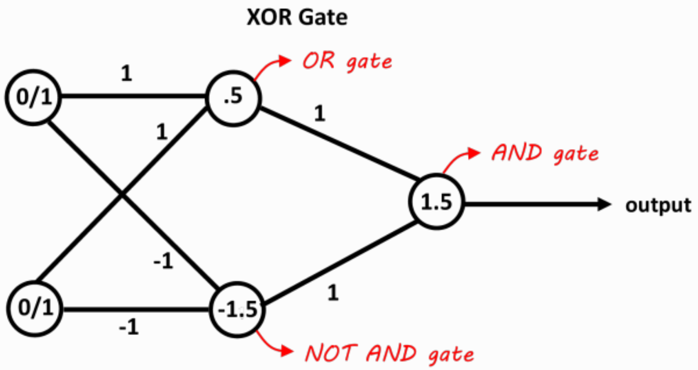
Сумма произведений – это скалярное произведение. Смотрим на знак скалярного произведения, следовательно вычислительная мощность «нейрона», как у разделяющей плоскости.



Логические функции

AND	$x^1 \wedge x^2 = [x^1 + x^2 - 3/2 > 0]$
OR	$x^1 \vee x^2 = [x^1 + x^2 - 1/2 > 0]$
NOT	$\neg x = [-x + 1/2 > 0]$
XOR	$x^1 \oplus x^2 = [(x^1 \vee x^2) - (x^1 \wedge x^2) - 1/2 > 0]$
	$x^1 \oplus x^2 = [x^1 + x^2 - 2x^1 x^2 - 1/2 > 0]$

Пример для XOR:



Обучение перцептрона «до изобретения» производной

Правило Хебба для $\{1; -1\}$ классификации:

$$\text{Если } \langle w^{[k]} x_{(k)} \rangle y_{(k)} < 0 \text{ то}$$
$$w^{[k+1]} := w^{[k]} + \eta x_{(k)} y_{(k)}.$$

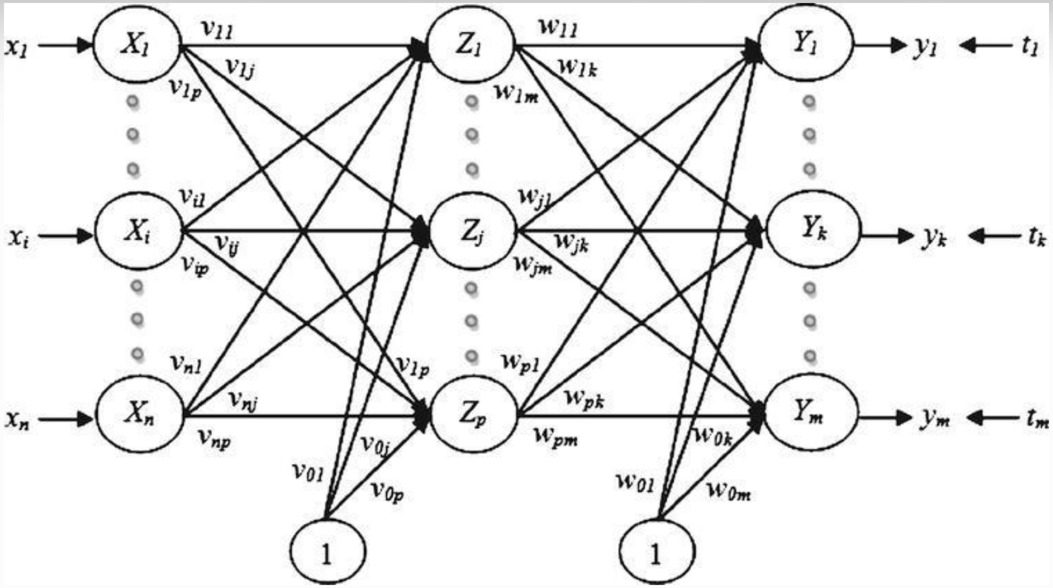
Правило Розенблатта для $\{1; 0\}$ классификации:

$$w^{[k+1]} := w^{[k]} - \eta\left(a_w\left(x_{(k)}\right) - y_{(k)}\right).$$

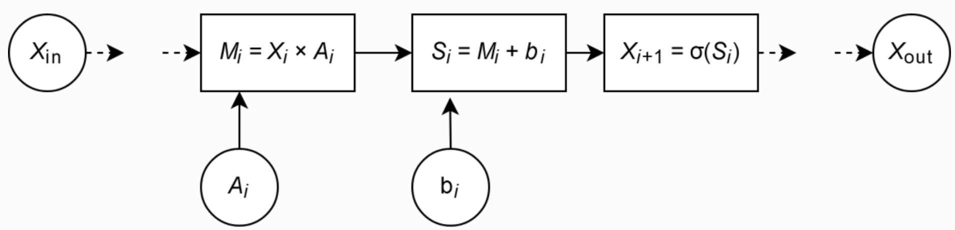
Дельта-правило для $L(a_w, x) = (\langle w, x \rangle - 1)^2$:

$$w^{[k+1]} := w^{[k]} - \eta\left(\langle w, x_{(k)} \rangle - y_{(k)}\right).$$

Многослойные сети (DNN, ANN, MLP, FNN)



В матричном виде: каждый слой — умножение на матрицу A, сдвиг вектором b и применение функции активации σ.



В таком виде «нейронная сеть» не только не «нейронная», но и не «сеть».

Пересчёт производной для одного слоя сети

Пусть i-й слой кодирует преобразование вектора длины n в вектор длины m.

- X_i — вход i-го слоя (вектор длины n)
- Y_i — выход i-го слоя (вектор длины m)
- Если слой не первый, то $X_i = Y_{i-1}$
- A_i, b_i — параметры слоя: матрица $n \times m$ и вектор длины m
- f — функция активации (для вектора вычисляется покомпонентно)

Тогда $Y_i = f(X_i \times A_i + b_i)$

Прямой проход (вычисление слоя)	**Обратный проход (вычисление производной)**
$M_i = X_i \times A_i$	$dX_i = dM_i \times A_i^T$ $dA_i = X_i^T \times dM_i$
$S_i = M_i + b_i$	$dM_i = dS_i$ $db_i = dS_i$
$Y_i = f(S_i)$	$dS_i = f'(S_i) \circ dY_i$

$\circ$ — произведение Адамара (покомпонентное произведение)

Словарь неправильных терминов (3)

Неправильно	Правильно
Метод обратного распространения ошибки	Вычисление (пересчёт) производной

- Формально ошибка отвечает на вопрос: «Что нужно прибавить к полученному ответу, чтобы получить верный ответ?»
- Если функция потери равна сумме квадратов разностей, то производная функции потери будет совпадать с ошибкой, но это будет верно только для значений последнего слоя.
- На самом деле, несмотря на название, используется производная!

Несколько слоёв – это хорошо

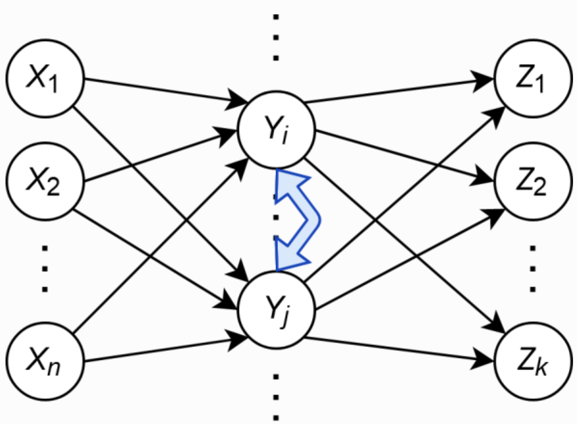
- Для **логических** функций: любую логическую функцию можно представить в ДНФ или КНФ, следовательно требуется не более двух слоёв, но с возможно экспоненциальным числом «нейронов» на промежуточном слое.
- Для **числовых** функций: Теорема Цыбенко или Универсальная теорема аппроксимации (1989). Искусственная нейронная сеть с одним скрытым слоем может аппроксимировать любую непрерывную функцию многих переменных с любой точностью при условии достаточного числа нейронов на скрытом слое.

В любом случае процесс больше похож на интерполяцию, чем на аппроксимацию. В реальности используют больше двух слоёв!

Несколько слоёв – это плохо (1)

- Проблема симметрии архитектуры многослойной нейронной сети: если изменить местами нейроны на промежуточном слое, то функция не изменится.
- Перестановка может быть непрерывной, следовательно уже для двух слоёв функция ошибки содержит множество минимумов.

Дискретная перестановка:



Непрерывная перестановка:

- $w(u, v)$ – связь между нейронами u и v.
- $\alpha \in [0; 1]$: при $0 \xrightarrow{\alpha} 1$ перестановка $Y_i \to Y_j$.
- $\forall\ 1 \leq u \leq n$ и $1 \leq v \leq k$:

$$w_\alpha(X_u, Y_i) = \alpha \cdot w(X_u, Y_i) + (1 - \alpha) \cdot w\left(X_u, Y_j\right)$$
$$w_\alpha\left(X_u, Y_j\right) = \alpha \cdot w\left(X_u, Y_j\right) + (1 - \alpha) \cdot w(X_u, Y_i)$$
$$w_\alpha(Y_i, Z_v) = \alpha \cdot w(Y_i, Z_v) + (1 - \alpha) \cdot w\left(Y_j, Z_v\right)$$
$$w_\alpha\left(Y_j, Z_v\right) = \alpha \cdot w\left(Y_j, Z_v\right) + (1 - \alpha) \cdot w(Y_i, Z_v)$$

Решение: инициализация параметров случайными значениями. Например из нормального или равномерного распределения.

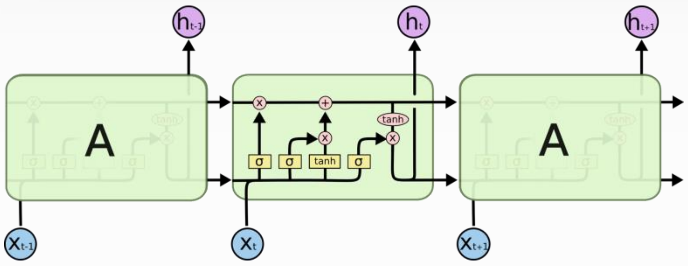
Несколько слоёв – это плохо (2)

- Проблема затухания градиента (vanishing gradient problem): с каждым слоем градиент затухает. Чем далее блок от выхода, тем хуже он обучается.
- Проблема «взрыва» (разрастания) градиента (exploding gradient problem): с каждым слоем градиент может неограниченно расти. Иногда возникает, если в данных присутствует выброс. При обновлении вектор параметров может быть испорчен.

Решение:
- Использование специальных преобразований (например LSTM) или функции активации (ReLU).
- Использование предобработки данных или специальной инициализации параметров (например Xavier или He).
- Подрезка градиента (gradient clipping):
    - Глобальная: $\|g\| > c \Rightarrow g_{new} = c \cdot \frac{g_{old}}{\|g\|}$
    - Локальная: $|g_i| > c \Rightarrow g_i = sign(c) \cdot c$

А нужны ли слои?

LSTM[1] (1997)



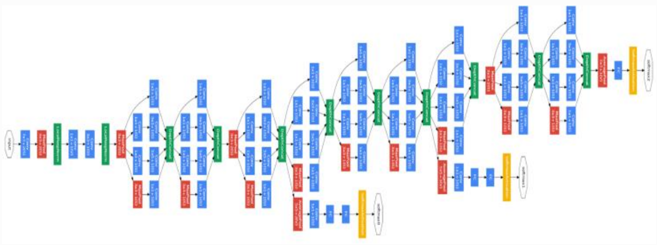
Статья Лекуна[2] (1998)

Sometimes it is helpful to add a small linear term, e.g.

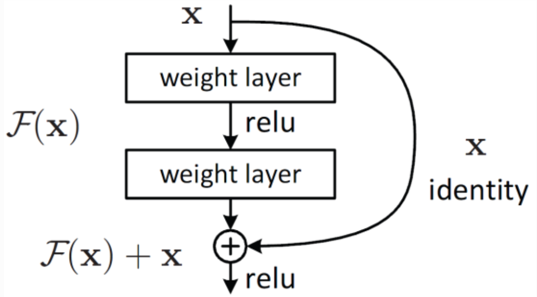
$$f(x) = \tanh(x) + ax$$

so as to avoid flat spots.

GoogLeNet[3] (2015)



ResNet [4] (2016)



[1] Hochreiter S., Schmidhuber J. Long short-term memory //Neural computation. 1997. Vol. 9. no. 8. P. 1735-1780.
[2] LeCun Y. A. et al. Efficient backprop // Neural networks: Tricks of the trade. Springer, Berlin, Heidelberg, 1998. P. 9-48.
[3] Szegedy C. et al. Going deeper with convolutions // Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. P. 1-9.
[4] He K. et al. Deep residual learning for image recognition // Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. P. 770-778.

Словарь неправильных терминов (4)

Неправильно	Правильно
Нейронные сети	Вычисление и оптимизация дифференцируемых функций
Глубокое (глубинное) обучение, Deep learning	
Нейронные сети (Искусственный интеллект)	Матричные вычисления
Слои	Преобразования
Веса	Настраиваемые параметры
Функция активации	Функция

Немного истории

1943 Искусственный нейрон (МакКаллок и Питтс)

1949 Правило обучения нейронов (Hebb)

1957 Персептрон (Rosenblatt)

1960 Правило обучения персептрона (Уидроу и Хофф)

1969 «Персептроны» (Мински и Паперт)

1974 Алгоритм обратного распространения ошибок (Уэброс и, независимо, Галушкин)

1998 Обобщенная аппроксимационная теорема (Вунш и, независимо, Горбань)

Эпоха создания моделей

1980 Сверточные сети (Фукусима)

1982 Рекуррентные сети (Хопфилд)

1991 Проблема затухания градиента указана Хохрайтером

1997 LSTM-модуль (Хохрайтер и Шмидхубер)

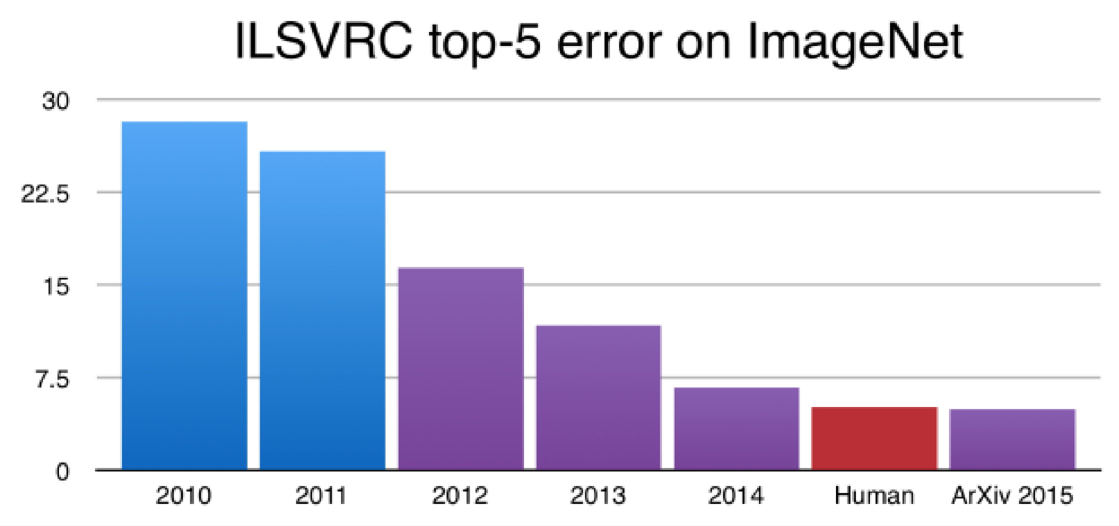
1998 Градиентный спуск для сверточных сетей (ЛеКун и др.)

2006 Глубокая нейронная сеть (Хинтон и др.)

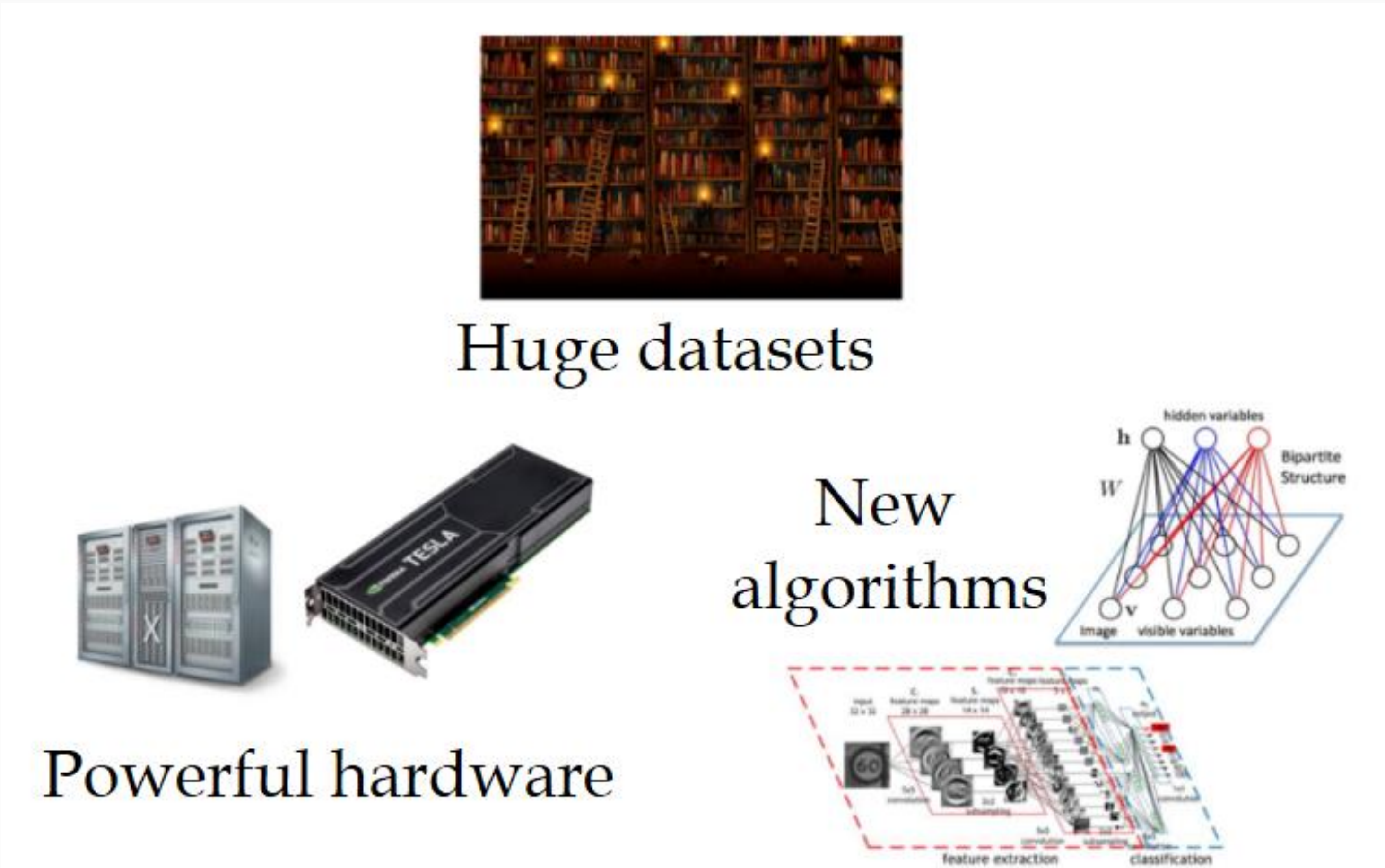
Начало современности

2012 Хинтон, Крижевский и Сутскевер предложили Dropout

2012 Они выиграли ImageNet (и два менее известных соревнования). Началась эпоха глубокого обучения.



Почему сейчас?



Проблемы

- У глубоких нейронных сеток много параметров, что является предпосылкой для переобучения
- Сложные зависимости можно находить только на большом наборе примеров
- Для увеличения размеров данных используется **аугментация данных**

Аугментация данных

Простейший и наиболее часто используемый метод уменьшить переобучение состоит в увеличении набора данных изменением исходных объектов преобразованиями, сохраняющими класс объектов.

Словарь неправильных терминов (5)

Неправильно	Правильно
Большие данные (Big data)	Данные не умещаются в один компьютер — **распределённые вычисления***.
	Данных достаточно для выявления закономерностей — можно использовать **машинное обучение**.
	Данных слишком много для обработки «руками» — вынуждены использовать **машинное обучение**.

* В этом значении используется крайне редко.

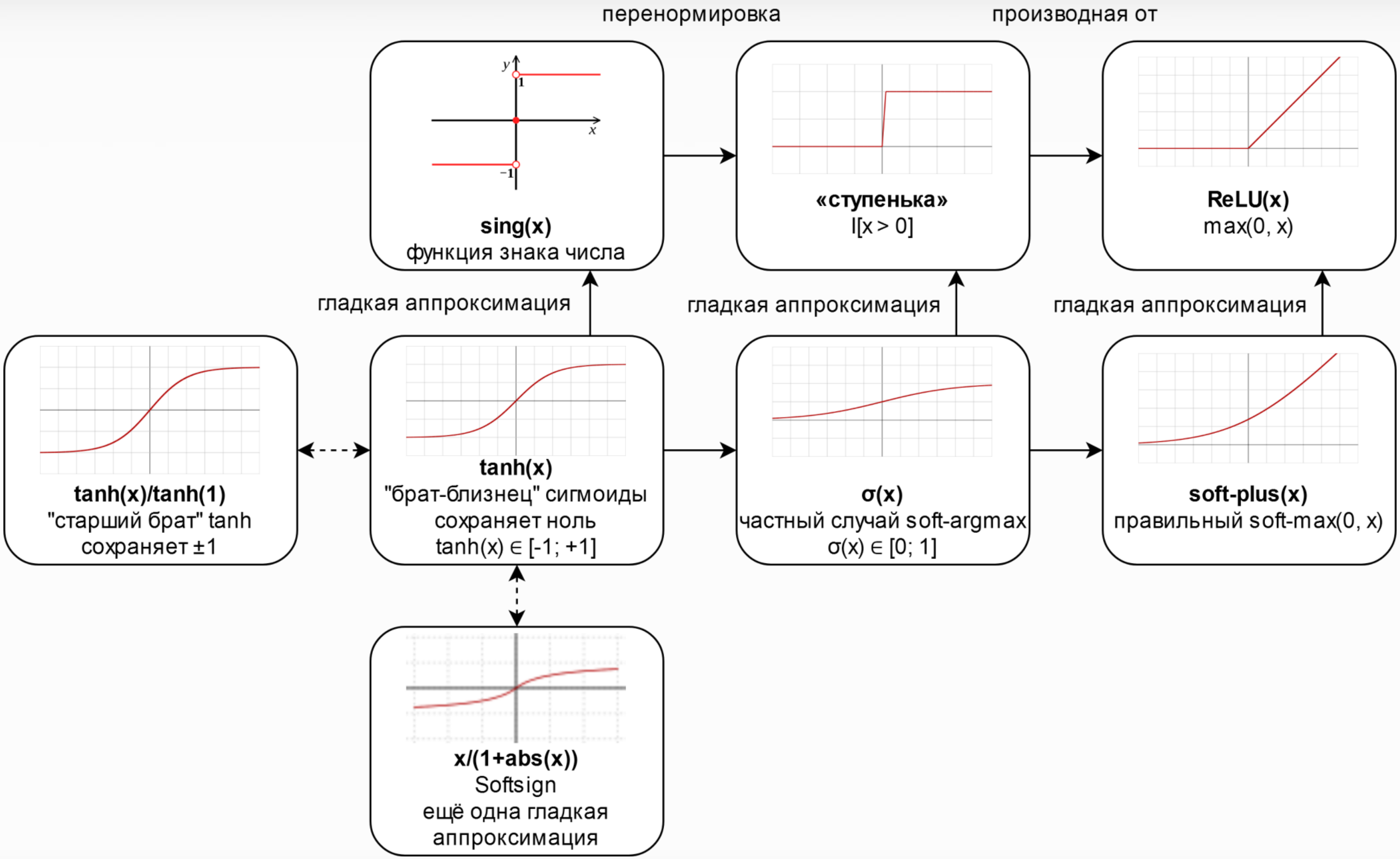
План лекции

- Сведение задачи обучения к задаче дифференцирования
- Дифференцирование составных функций по графу вычислений
- Дифференцирование сложных функций
- Soft-Max и Soft-Arg-Max
- История нейронных сетей
- Функции активации
- Дропаут
- Дополнительные темы

Некоторые функции активации

Название	Функция	Производная
Логистическая (сигмоида)	$f(x) = \sigma(x) = \dfrac{1}{1+e^{-x}}$	$f'(x) = f(x)\big(1 - f(x)\big)$
Гиперболический тангенс	$f(x) = \tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$
ReLU (Leaky ReLU) $0 \le \alpha \le 1$	$f(x) = \max(\alpha x, x) = \begin{cases} \alpha x, x < 0 \\ x, x \ge 0 \end{cases}$	$f'(x) = \begin{cases} \alpha, & x < 0 \\ 1, & x \ge 0 \end{cases}$
SoftPlus	$f(x) = \ln(1 + \mathrm{e}^{\mathrm{x}})$	$f'(x) = \dfrac{1}{1+e^{-x}}$
Arctg	$f(x) = tg^{-1}(x)$	$f'(x) = \dfrac{1}{1+x^2}$
SoftSign	$f(x) = \dfrac{x}{1+\lvert x\rvert}$	$f(x) = \dfrac{1}{(1+\lvert x\rvert)^2}$

Связь функций активации



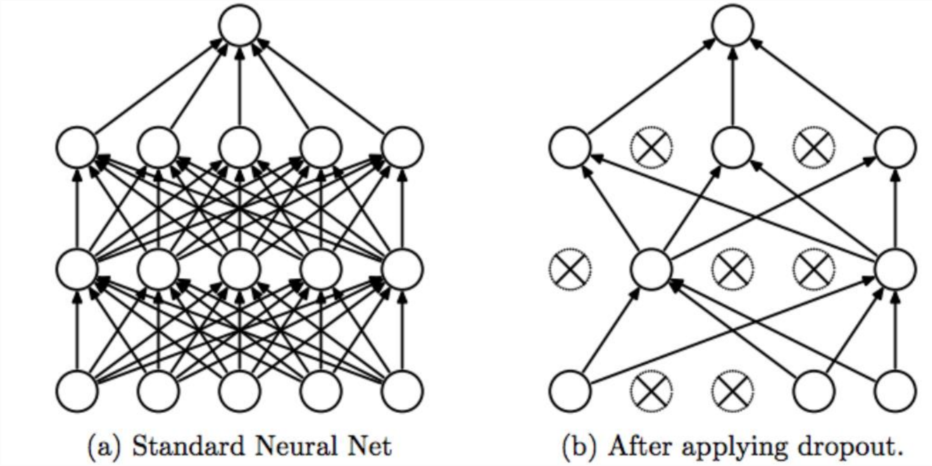
План лекции

- Сведение задачи обучения к задаче дифференцирования
- Дифференцирование составных функций по графу вычислений
- Дифференцирование сложных функций
- Soft-Max и Soft-Arg-Max
- История нейронных сетей
- Функции активации
- Дропаут
- Дополнительные темы

Дропаут (1/2)

Дропаут: при обучении на итерации обнуляем выходы нейронов с некоторой вероятностью (часто 0,5)

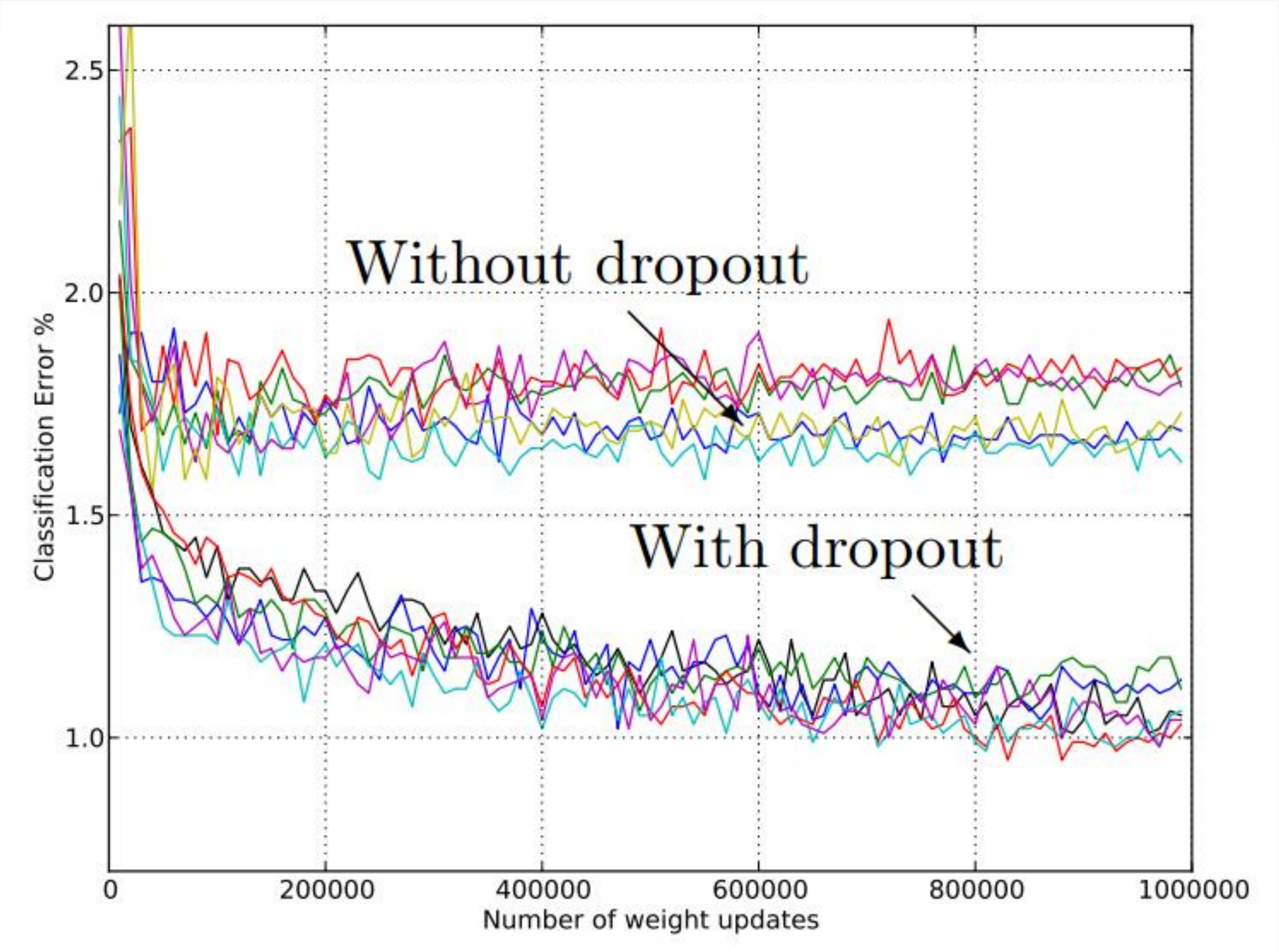
- Обнуленные нейроны не участвуют в обучении, не внося вклад в ошибку
- Для каждого выхода мы фактически строим новую сеть, но у всех этих сетей общие веса



Дропаут (2/2)

- Уменьшается соадаптация нейронов, потому что они не могут рассчитывать на соседей
- Обучается более робастное представление
- Без дропаута сети значительно переобучаются
- Дроупаут приблизительно вдвое увеличивает число операций до сходимости

Эффект дропаута



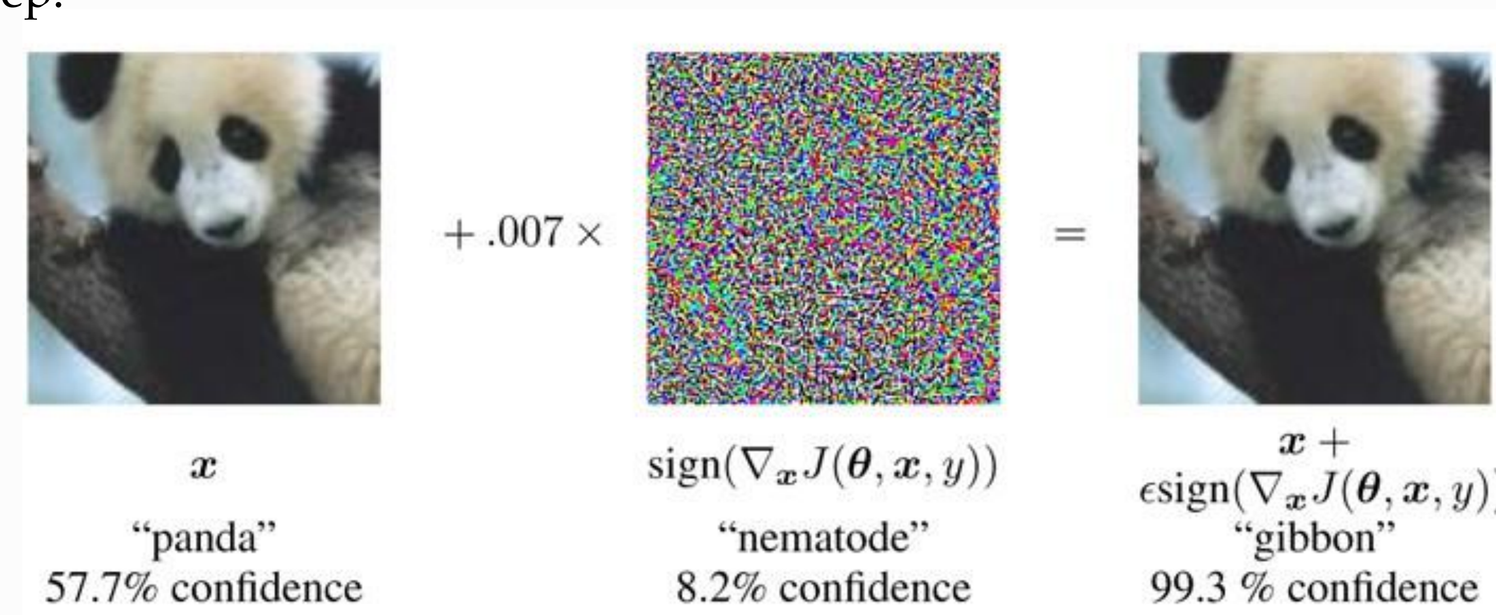
План лекции

- Сведение задачи обучения к задаче дифференцирования
- Дифференцирование составных функций по графу вычислений
- Дифференцирование сложных функций
- Soft-Max и Soft-Arg-Max
- История нейронных сетей
- Функции активации
- Дропаут
- Дополнительные темы

Производную по входу тоже можно использовать

Состязательные атаки (adversarial attack): объект можно «незаметно» модифицировать при помощи производной. При этом предсказанный класс объекта изменится.

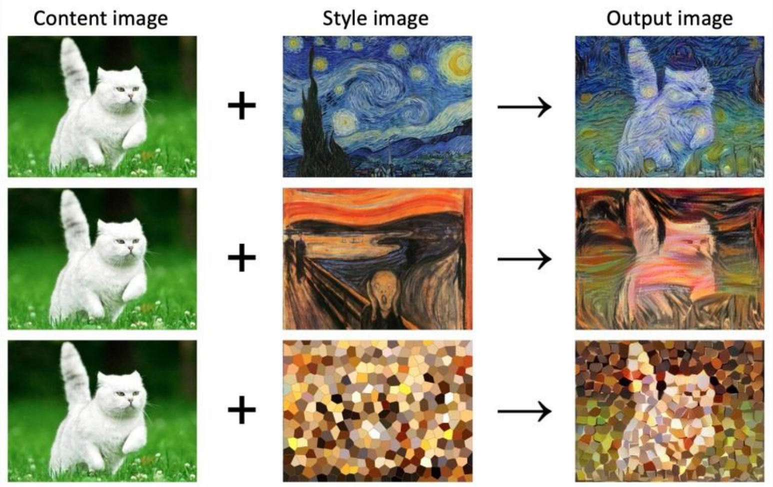
Пример:



Изображение взято из: Goodfellow I. J., Shlens J., Szegedy C. Explaining and harnessing adversarial examples //arXiv preprint arXiv:1412.6572. 2014.

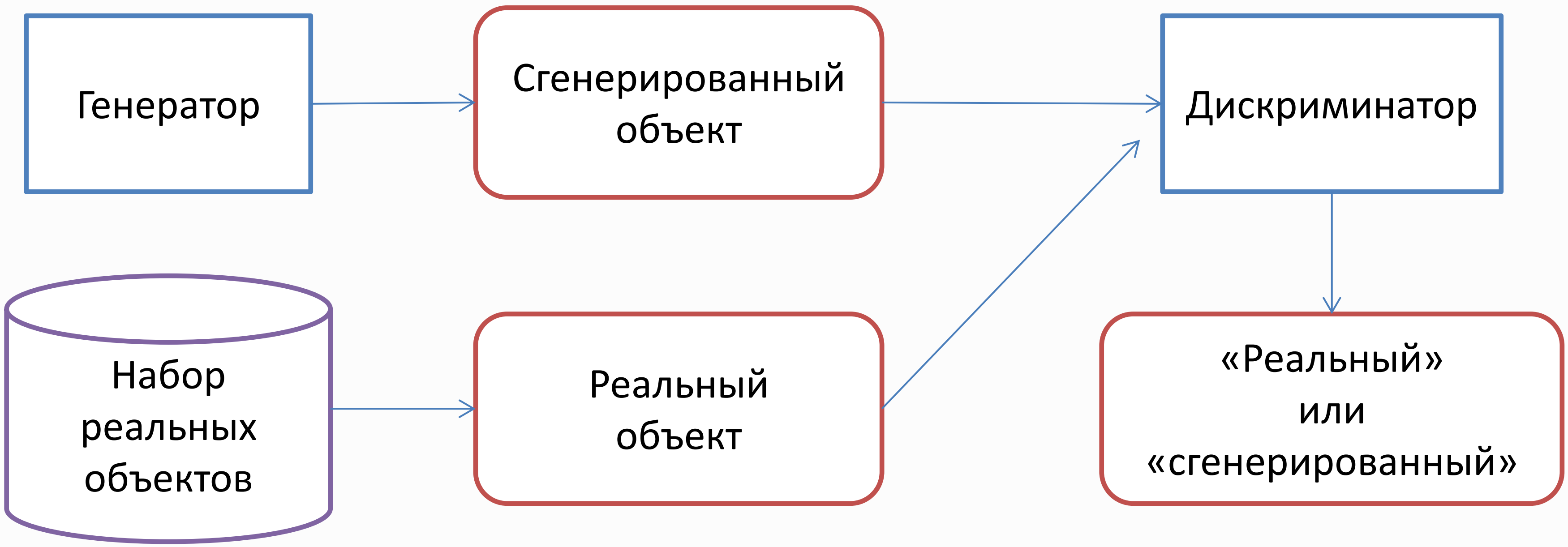
Другое применение производной по объекту

- Проверка того, чему обучилась / переобучилась сеть на промежуточных слоях или выходе. Идея применяется в DeepDream стилизациях (рисунок слева).
- Перенос стиля (рисунок справа).



Генеративно состязательные сети

- Генеративно-состязательная сеть (GAN – Generative Adversarial Nets): Производная по объекту используется как производная по выходу генератора, который эти объекты генерирует.



Словарь неправильных терминов (6) или что ещё можно глянуть по теме

Неправильно	Правильно
«Введение в коммуникационную сложность»*	«Нейронные сети»

* Секретный курс от Сергея Николенко на портале Лекториум:

- https://www.lektorium.tv/node/32968
- https://www.youtube.com/playlist?list=PL-_cKNuVAYAUhvlUfW7P2cdhWCRDWs0pG

Не обращайте внимание на название! На самом деле это годный курс по Нейронным сетям! Но из-за неправильного названия он «секретный».