Машинное обучение (Machine Learning)

Нейронные сети (Neural networks)

Уткин Л.В.

ПОЛИТЕХ
Санкт-Петербургский
политехнический университет
Петра Великого

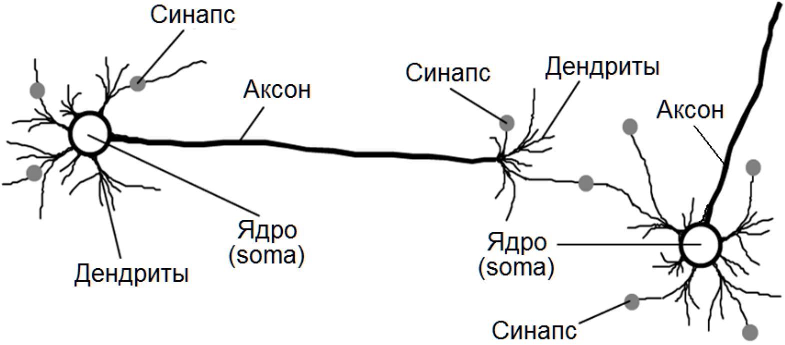
Содержание

Презентация является компиляцией и заимствованием материалов из замечательных курсов и презентаций по машинному обучению:

К.В. Воронцова, А.Г. Дьяконова, Н.Ю. Золотых, С.И. Николенко, Andrew Moore, Lior Rokach, Rong Jin, Luis F. Teixeira, Alexander Statnikov и других.

Понятие персептрона

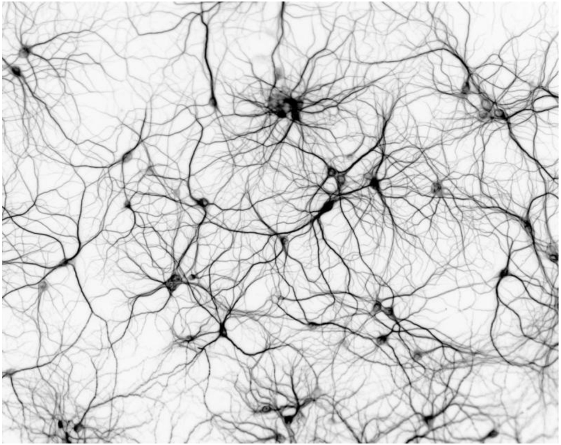
Биологический прототип



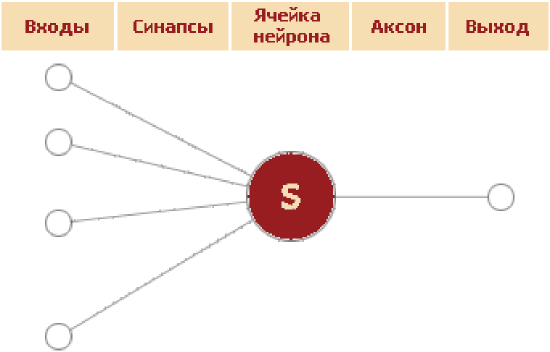
Особенности биологии

- **Нервная клетка** - это черный ящик, у которого есть **дендриты**, входы, по которым поступают сигналы (отрицательные ионы) в клетку или ее ядро.
- Если внутри накапливается большой отрицательный заряд, клетка возбуждается и генерирует импульс, который по **аксону** к дендритам следующих клеток.
- Место соединения аксона нейрона с дендритом называется **синапсом**.
- Заряды аддитивны, т.е. они накапливаются в клетках. Отсюда клетка - это элементарный **классификатор**, принимающий решение, возбудиться или нет.

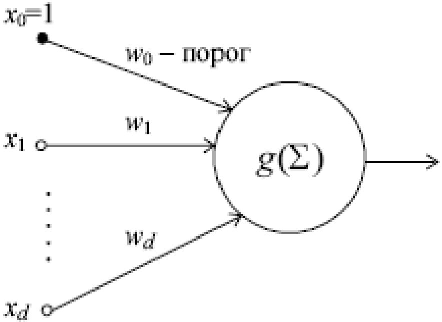
Нейронная сеть (биология)



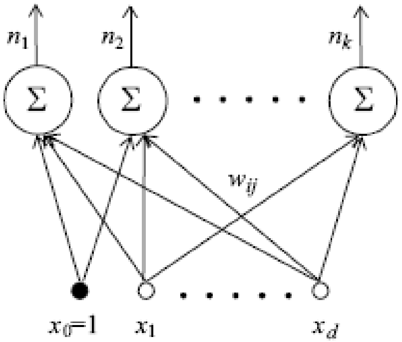
Структура нейрона



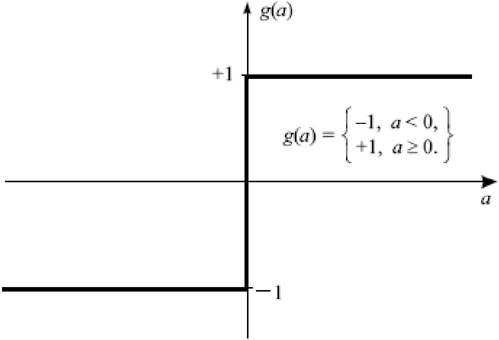
Структура нейрона



Персептрон Розенблатта



Пороговая функция



Формальный нейрон

Нейрон вычисляет взвешенную сумму своих входов:

$$y(\mathbf{x}) = \sum_{i=1}^{d} x_i w_i + w_0$$

Для удобства входной вектор расширяется до $x = (1, x_1, ..., x_d)$ и порог w_0 вносится под знак суммы:

$$y(\mathbf{x}) = \sum_{i=0}^{d} x_i w_i$$

w_0 - лимит активации;

Выход персептрона

Выход y^* вычисляется как

$$y^* = \begin{cases} 1, & \text{если } w_0 + x_1 w_1 + \ldots + x_d w_d > 0 \\ -1, & \text{иначе} \end{cases}$$

Веса изначально $w_0, w_1, \ldots, w_d$ неизвестны.

Обучить нейрон - означает найти такие веса по обучающей выборке, чтобы нейрон с максимальной точностью классифицировал данные

Вопрос: как обучить нейрон или вычислить веса?

Обучение персептрона

1. Веса $\mathbf{w} = (w_1, ..., w_d)$ инициализируются случайными значениями.
2. Подаем на вход персептрона вектор $\mathbf{x}_k = (x_1, ..., x_d)$ из обучающей выборки и вычисляем выход нейрона y_k^*.
3. Правило изменения весов:

$$w_i(k+1) \leftarrow w_i(k) + \eta\,(y_k - y_k^*)\,x_i(k)$$

4. Переход к Шагу 2 для выбора следующего $\mathbf{x}_{k+1}$ из обучающей выборки.

y_k^* - выход нейрона на k-ом элементе выборки
y_k - метка класса k-го элемента обучающей выборки
$\eta > 0$ - коэффициент, задающий скорость обучения
$x_i(k)$ - значение i-го признака k-го элемента обучающей выборки

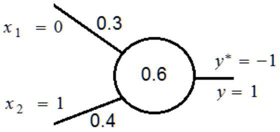
Пример дизъюнкции

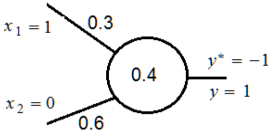
Логическая функция “или” ($y = x_1 \vee x_2$):

x_1	0	0	1	1
x_2	0	1	0	1
y	0	1	1	1

Обучающая выборка $((x_1, x_2), y)$ состоит из двух примеров: $((0, 1), 1)$ и $((1, 0), 1)$.

Пример обучения нейрона





1. Начальные веса: $w = (0.6, 0.3, 0.4)$, $\eta = 0.1$
2. Вход: $(x_1, x_2) = (0, 1)$, выход: $y = 1 \Rightarrow y^* = [0 \cdot 0.3 + 1 \cdot 0.4 < 0.6] = -1$ (выходы не совпадают)

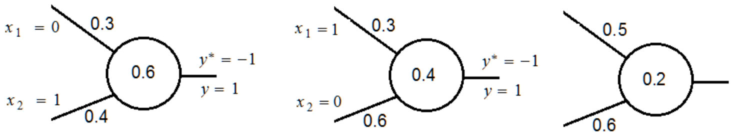
$$w_i(k+1) \leftarrow w_i(k) + \eta\,(y_k - y_k^*)\,x_i(k)$$

$$w_0 \leftarrow w_0 + \eta\,(y - y^*) \cdot x_0 = -0.6 + 0.1\,(1 - (-1)) \cdot 1 = -0.4$$

$$w_1 \leftarrow w_1 + \eta\,(y - y^*) \cdot x_1 = 0.3 + 0.1\,(1 - (-1)) \cdot 0 = 0.3$$

$$w_2 \leftarrow w_2 + \eta\,(y - y^*) \cdot x_2 = 0.4 + 0.1\,(1 - (-1)) \cdot 1 = 0.6$$

Пример обучения нейрона



3. Вход: $(x_1, x_2) = (1, 0)$, выход: $y = 1 \Rightarrow y^* = [1 \cdot 0.3 + 0 \cdot 0.6 < 0.4] = -1$ (выходы не совпадают)

$$w_i(k+1) \leftarrow w_i(k) + \eta\,(y_k - y_k^*)\,x_i(k)$$

$$w_0 \leftarrow w_0 + \eta\,(y - y^*) \cdot x_0 = -0.4 + 0.1\,(1 - (-1)) \cdot 1 = -0.2$$

$$w_1 \leftarrow w_1 + \eta\,(y - y^*) \cdot x_1 = 0.3 + 0.1\,(1 - (-1)) \cdot 1 = 0.5$$

$$w_2 \leftarrow w_2 + \eta\,(y - y^*) \cdot x_2 = 0.6 + 0.1\,(1 - (-1)) \cdot 0 = 0.6$$

4. **Нейрон обучен!**

Метод градиентного спуска

- Идея построения перцептрона - минимизация ошибки.
- Перцептронная функция $y^*(x_1, ..., x_d) = \sum_{i=0}^{d} x_i w_i$ должна быть приближена к функции, заданной примерами обучающей выборки: $y = g(x_1, ..., x_d)$.
- Мера ошибки - среднеквадратичное отклонение от целевых значений:

$$E(w_0, ..., w_d) = \frac{1}{2} \sum_{k=1}^{n} (y_k - y^*(x_1(k), ..., x_d(k)))^2$$

- Цель - минимизировать $E(w_0, ..., w_d)$ по $w_0, ..., w_d$.

Метод градиентного спуска

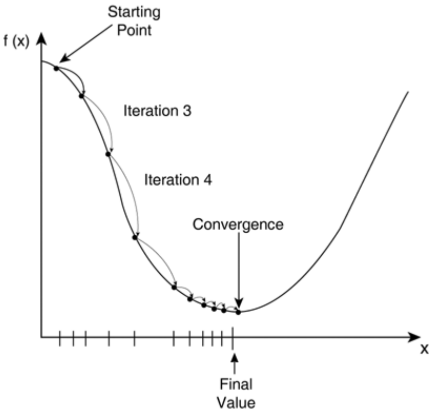
- $E(w_0, ..., w_d)$ - параболическую поверхность с единственным минимумом.
- Двигаемся в направлении, противоположном градиенту

$$-\nabla E(w_0, ..., w_d) = -\left[\frac{\partial E}{\partial w_0}, ..., \frac{\partial E}{\partial w_d}\right]$$

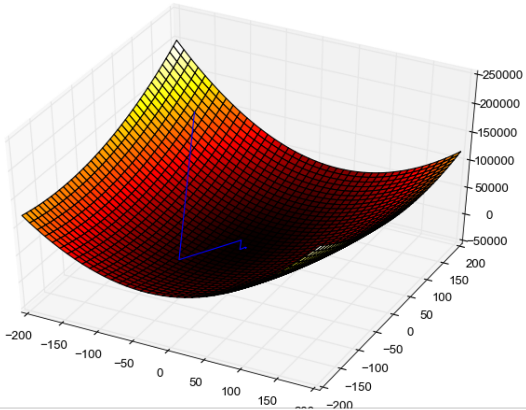
- Коррекция весов:

$$w_i(k+1) \leftarrow w_i(k) - \eta \frac{\partial E}{\partial w_i}$$

Движение к минимуму



Движение к минимуму



Метод градиентного спуска (далее)

- Вычислим $\partial E / \partial w_i$:
- Двигаемся в направлении, противоположном градиенту

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_{k=1}^{n} \frac{\partial}{\partial w_i} \left(y_k - \sum_{i=0}^{d} x_i(k) w_i \right)^2$$
$$= \sum_{k=1}^{n} \left(y_k - \sum_{i=0}^{d} x_i(k) w_i \right) (-x_i(k)) \, .$$

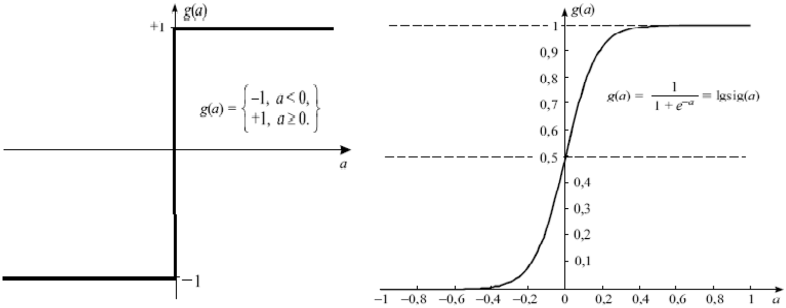
- Коррекция весов:

$$w_i(k+1) \leftarrow w_i(k) + \eta \sum_{k=1}^{n} \left(y_k - \sum_{i=0}^{d} x_i(k) w_i \right) x_i(k)$$

Пороговая функция или функция активации

Пороговая функция - **сигмоид**:

$$y(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$



Пороговая функция или функция активации

- Еще одна пороговая функция - **бисигмоид или гиперб. тангенс**:

$$y(x) = \sigma(x) = \frac{2}{1+e^{-x}} - 1$$
$$= \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- Изменяется от -1 до 1.

Пороговая функция или функция активации

- И еще одна пороговая функция - **rectified linear unit (ReLU)**:
$$y(x) = \sigma(x) = \max(0, x)$$
- Изменяется от 0 до $\sup x$.

Пороговая функция и перцептрон

- Общая формула работы перцептрона с учетом пороговой функции сигмоида

$$y^*(x_1, \dots, x_d) = \frac{1}{1 + e^{-\sum_{i=0}^{d} x_i w_i}}.$$

- Сигмоид обладает важным преимуществом: от него легко считать производную: $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
- Гипербол. тангенс еще проще: $\sigma'(x) = 1 - \sigma^2(x)$
- Коррекция весов:

$$w_i(k+1) \leftarrow w_i(k) + \eta y^*(1 - y^*)(y_k - y^*) x_i(k)$$

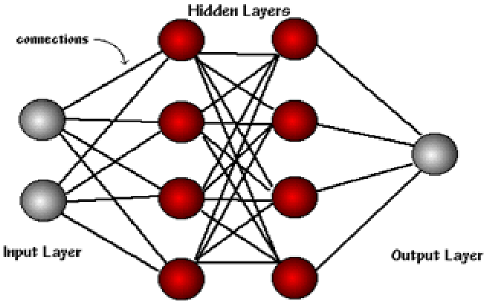
Возможности перцептрона

- В 1973 г. Дуда и Харт доказали теорему сходимости перцептрона: для любых **линейно разделяемых** входных данных правило обучения находит решение за конечное число шагов.
- Многие задачи не могут быть решены.
- Добавление нейронов в перцептрон не решает проблему.

Что делать?

Многослойная сеть

Многослойный перцептрон



Теорема Колмогорова

Теорема Колмогорова по сути: для решения любой задачи возможно построить нейронную сеть.

Формально: *Каждая непрерывная функция d переменных, заданная на единичном кубе d-мерного пространства, представима в виде*

$$f(x_1, ..., x_d) = \sum_{i=1}^{2d+1} h_i \left(\sum_{j=1}^{n} \varphi_i^j(x_j) \right),$$

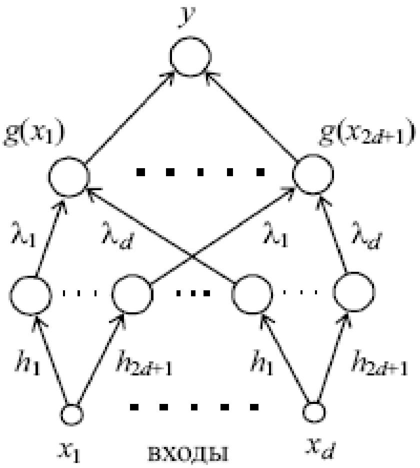
где h_i – непрерывные негладкие функции; $\varphi_i^j(x_j)$ - стандартные функции, не зависящие от вида f.

Теорема Колмогорова

Теорема Колмогорова по простому:

Любое отображение входов нейронной сети в ее выходы может быть реализовано трехслойной нейронной сетью прямого распространения с $d(2d+1)$*нейронами на первом и* $2d+1$*на втором слое.*

Теорема Колмогорова в терминах нейронных сетей



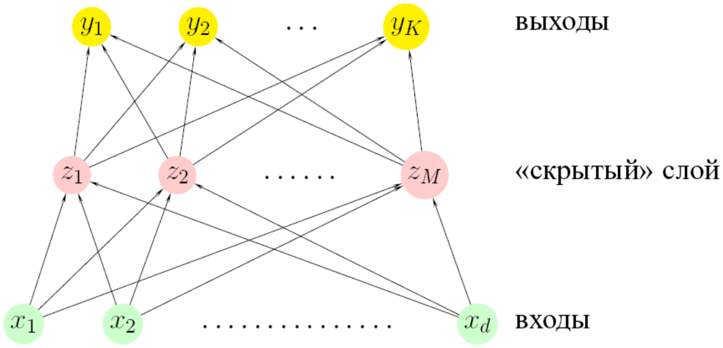
«Универсальная теорема об аппроксимации»

Пусть σ - ограниченная, не постоянная, монотонно возрастающая, непрерывная функция. Тогда для любой непрерывной функции $f^* : [0,1]^d \to \mathbb{R}$, для любого $\varepsilon > 0$ существуют M, α_m $(m = 1, 2, ..., M)$, w_{mj} $(m = 1, 2, ..., \ j = 0, 2, ..., d)$, такие, что функция

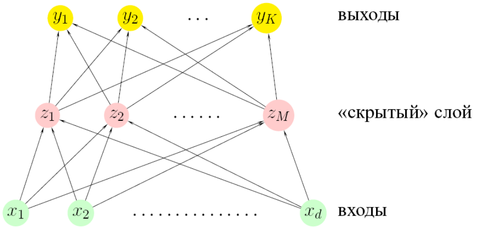
$$f(x_1, ..., x_d) = \sum_{i=1}^{M} \alpha_m \sigma \left(w_{m0} + \sum_{j=1}^{d} w_{mj} x_j \right)$$

является ε-аппроксимацией функции f, т.е. $|f(x) - f^*(x)| \leq \varepsilon$.

Как работает сеть?



Как работает сеть?



$$z_m = \sigma\left(w_{0m} + w_{1m}x_1 + \ldots + w_{dm}x_d\right),\ m = 1, \ldots, M$$
$$t_k = v_{0k} + v_{1k}z_1 + \ldots + v_{Mk}z_M,\ k = 1, \ldots, K$$
$$y_k = f_k(x_1, \ldots, x_d) = g_k(t_1, \ldots, t_K),\ k = 1, \ldots, K$$

Это - прямое распространение (forward propagation)

Сеть для классификации с K классами

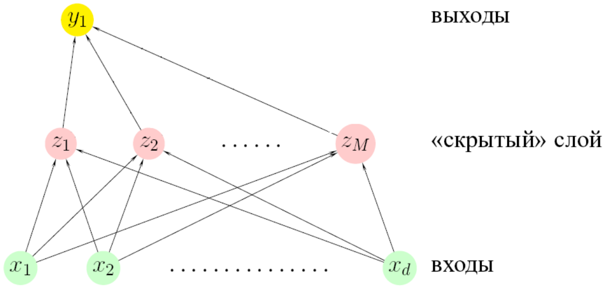
- K выходов: каждый выход моделирует вероятность данного класса
- $z_m = \sigma\left(w_{0m} + w_{1m}x_1 + \ldots + w_{dm}x_d\right),\ m = 1, \ldots, M$
- $t_k = v_{0k} + v_{1k}z_1 + \ldots + v_{Mk}z_M,\ k = 1, \ldots, K$
- g_k тождественная функция или *softmax*-функция (как в логистической регрессии)

$$
\begin{aligned}
g_k(t_1, \ldots, t_K) &= \frac{\exp\left(t_k\right)}{\sum_{l=1}^{K} \exp\left(t_l\right)} \\
&= \frac{\exp\left(v_{0k} + v_{1k}z_1 + \ldots + v_{Mk}z_M\right)}{\sum_{l=1}^{K} \exp\left(v_{0l} + v_{1l}z_1 + \ldots + v_{Ml}z_M\right)}
\end{aligned}
$$

- $f(x) = \arg\max_k g_k(x)$

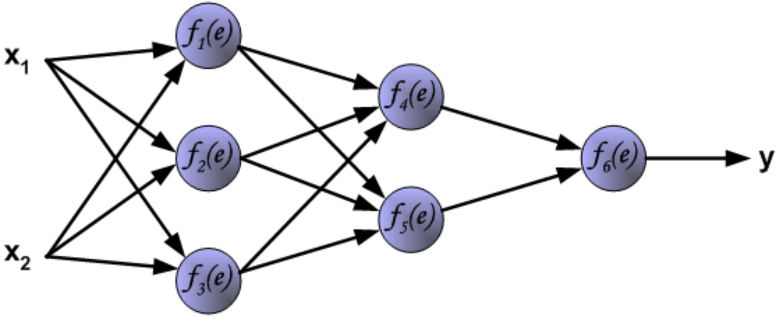
Сеть для регрессии

Один выход



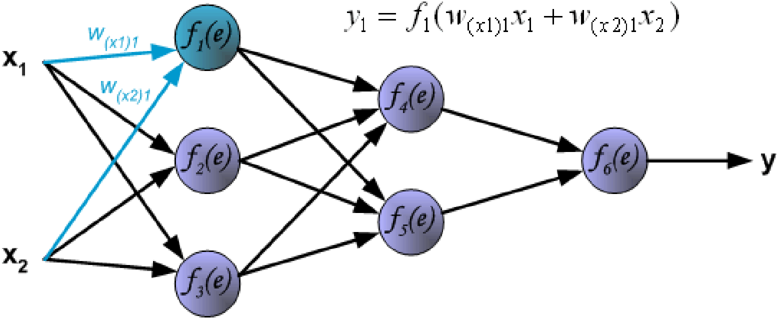
$z_m = \sigma\left(w_{0m} + w_{1m}x_1 + \ldots + w_{dm}x_d\right),\ m = 1, \ldots, M$
$t = v_{0k} + v_{1k}z_1 + \ldots + v_{Mk}z_M,\ k = 1, \ldots, K$
$y = f(x_1, \ldots, x_d) = g(t) = g(v_{0k} + v_{1k}z_1 + \ldots + v_{Mk}z_M)$
Функция g на выходе - тождественная функция

Прямое распространение

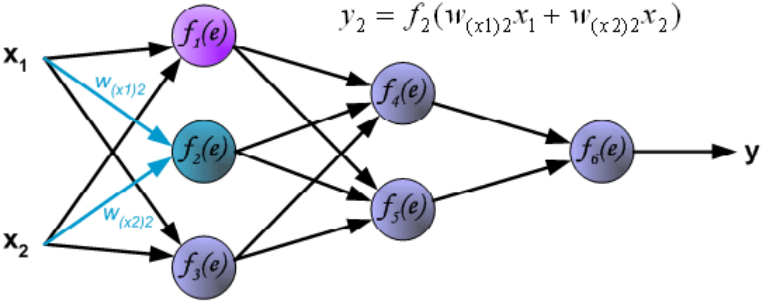


http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Прямое распространение

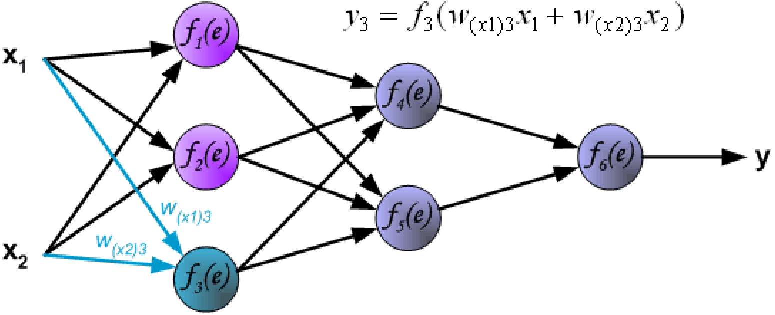


Прямое распространение

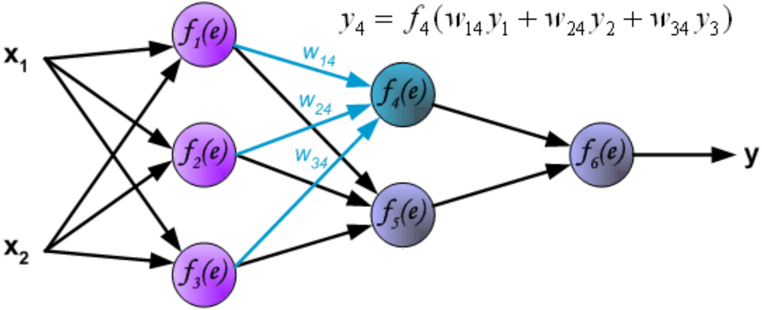


Прямое распространение

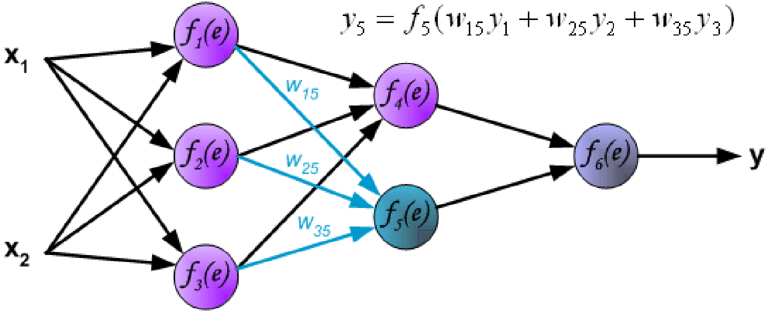
$$y_3 = f_3(w_{(x1)3}x_1 + w_{(x2)3}x_2)$$



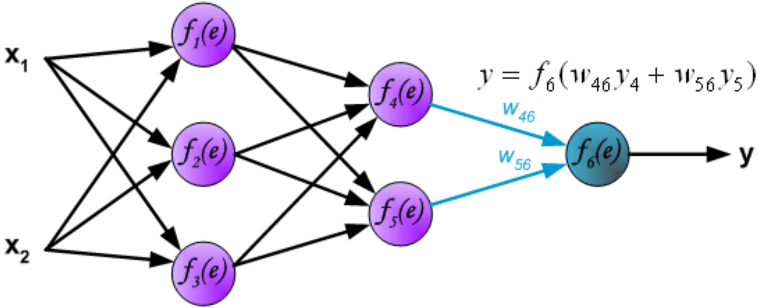
Прямое распространение



Прямое распространение



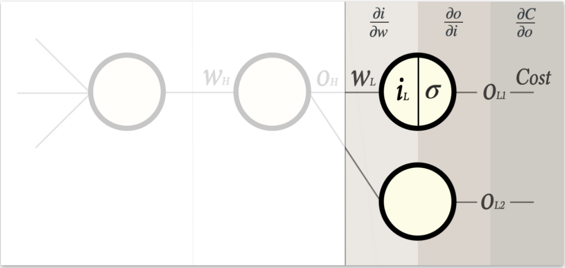
Прямое распространение



$$y = f_6(w_{46}y_4 + w_{56}y_5)$$

Обратное распространение ошибки (1)

Последний слой



Как найти $\partial C / \partial w_L$?

Обратное распространение ошибки (2)

Последний слой

$$\frac{\partial C}{\partial w_L} = \frac{\partial i_L}{\partial w_L} \frac{\partial o_{L1}}{\partial i_L} \frac{\partial C}{\partial o_{L1}}$$

Эти три частные производные вычислить несложно:

Обратное распространение ошибки (3)

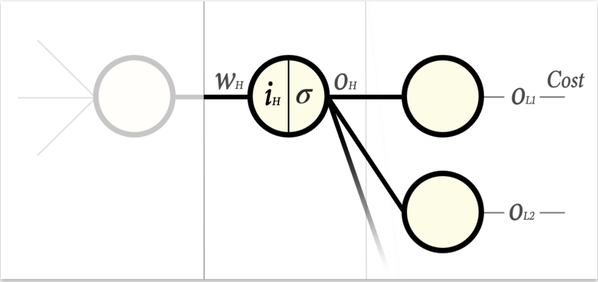
$$\frac{\partial C}{\partial w_L} = \frac{\partial i_L}{\partial w_L}\frac{\partial o_{L1}}{\partial i_L}\frac{\partial C}{\partial o_{L1}}$$

- $\frac{\partial i_L}{\partial w_L}$ - *как сильно вход нейрона изменяется, когда* w_L *измен*. Вход нейрона $o_{L1}w_L + b$
- $\frac{\partial o_{L1}}{\partial i_L}$ - *как сильно выход нейрона изм-ся, когда вход изменяется* - это $\sigma^{'}(i) = \sigma(i)(1 - \sigma(i))$
- $\frac{\partial C}{\partial o_{L1}}$ - *как меняются потери при изменении выхода нейрона*

$$\frac{\partial}{\partial o_{L1}}\sum_j (o_{Lj} - e_j)^2 = 2\,(o_{L1} - e_1)$$

Обратное распространение ошибки (4)

Скрытый слой



$$\frac{\partial C}{\partial w_H} = \frac{\partial i_H}{\partial w_H} \frac{\partial o_H}{\partial i_H} \frac{\partial C}{\partial o_H}$$

Обратное распространение ошибки (5)

Скрытый слой

$$\frac{\partial C}{\partial w_H} = \frac{\partial i_H}{\partial w_H} \frac{\partial o_H}{\partial i_H} \frac{\partial C}{\partial o_H}$$

Два первых множителя такие же, как и раньше, и сводятся к следующему:

- выход предыдущего слоя
- производная от функции активации соответственно
- но $\partial C / \partial o_H$ сложнее, т.к. изменение o_H меняет входные данные для **всех** нейронов в последнем слое и, как следствие, изменяет функцию потерь в более широком смысле.

Обратное распространение ошибки (6)

Скрытый слой

$$\frac{\partial C}{\partial w_H} = \frac{\partial i_H}{\partial w_H}\frac{\partial o_H}{\partial i_H}\left(\frac{\partial C}{\partial o_H}\right)$$

$$\frac{\partial C}{\partial o_H} = \frac{\partial C_{oL1}}{\partial o_H} + \frac{\partial C_{oL2}}{\partial o_H} + \ldots + \frac{\partial C_{oLn}}{\partial o_H}$$

каждое слагаемое член описывает, насколько изменяется функция потерь при изменении o_H, но только для той части сигнала, которая направляется через этот конкретный выходной нейрон.

Обратное распространение ошибки (7_1)

- Рассмотрим первое слагаемое для первого нейрона слоя:
$$\frac{\partial C_{oL1}}{\partial o_H} = \frac{\partial i_{L1}}{\partial o_H} \frac{\partial C_{oL1}}{\partial i_{L1}}$$
- $\partial i_{L1}/\partial o_H$ - как изменение выхода нейрона в текущем скрытом слое изменяет вход последнего слоя?
Вход нейрона последнего слоя $i_{L1} = o_H w_L + b$, тогда $\partial i_{L1}/\partial o_H = w_L$!

Обратное распространение ошибки (7_2)

- Рассмотрим первое слагаемое для первого нейрона слоя:
$$\frac{\partial C_{oL1}}{\partial o_H} = \frac{\partial i_{L1}}{\partial o_H}\frac{\partial C_{oL1}}{\partial i_{L1}}$$
- $\partial C_{oL1}/\partial i_{L1}$ - как изменение входа нейрона в последнем слое изменяет функцию потерь? Разве это не знакомо? Это то, что было получено для нейрона, когда корректировали веса последнего слоя. Это всего лишь два последних множителя в цепи $\frac{\partial C}{\partial w_L} = \frac{\partial i_L}{\partial w_L}\frac{\partial o_{L1}}{\partial i_L}\frac{\partial C}{\partial o_{L1}}$. Это и есть “magic” часть обратного распространиения.

Обратное распространение ошибки (8)

Итог:

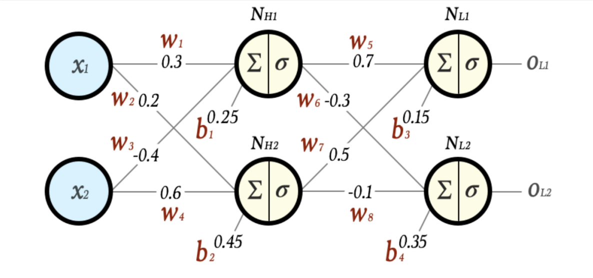
- Чтобы узнать, как обновить вес в скрытом слое, берем частную производную потерь в терминах этого веса.
- Применяя правило разложения, получаем три множителя, два из которых уже умеем вычислять.
- Третий множитель в этой цепочке - это взвешенная сумма произведения двух множителей, которые уже вычислены на последнем слое.
- Это означает, что можем рассчитать все веса в этом скрытом слое так же, как и в последнем слое, с той лишь разницей, что используем уже рассчитанные данные из предыдущего слоя вместо производной функции стоимости.

Обратное распространение ошибки (9)

Смещение b!

- Схема та же. Отличие - первый множитель цепочки производной как последнего слоя, так и скрытого слоя будет иметь вид: $\partial i / \partial b$ вместо $\partial i / \partial w$.
- Так как вход нейрона равен $o_H w_L + b$, частная производная по b просто равна 1!
- Если для весов умножали цепочку производных на выход последнего слоя, то здесь просто игнорируем выход в случае смещения и умножаем на единицу
- Изменение смещения не зависит от выхода предыдущего нейрона

Обратное распространение ошибки (самостоятельно)



$$w_6^+ = -0.306 \quad w_2^+ = 0.203$$
$$w_7^+ = 0.5118 \quad w_3^+ = -0.390$$
$$w_8^+ = -0.114 \quad w_4^+ = 0.6048$$

$$b_3^+ = 0.1627 \quad b_1^+ = 0.253$$
$$b_4^+ = 0.334 \quad b_2^+ = 0.4516$$

Алгоритм обратного распространения ошибки

- **Входы сети**: $x_1, \ldots, x_m$
- **Вес**, стоящий на ребре, соединяющем i-ый и j-ый узлы: w_{ij}
- **Выход** i-го узла сети: y_i^*
- **Целевые значения в обучающей выборке**: $y_1, \ldots, y_n$
- **Функция ошибки**:

$$E(\{w_{ij}\}) = \frac{1}{2} \sum_{i=1}^{n} \sum_{k \in \text{Выходы}} \left(y_k^{(i)} - y_k^*(x_1(i), \ldots, x_m(i)) \right)^2$$

Алгоритм обратного распространения ошибки (модификация весов)

Основная идея: **градиентный метод:**

$$\triangle w_{ij} = -\eta \frac{\partial E_s(\{w_{ij}\})}{dw_{ij}},$$

где

$$E_s(\{w_{ij}\}) = \frac{1}{2} \sum_{k \in \text{Выходы}} \left(y_k^{(d)} - y_k^*(d) \right)^2$$

Как подсчитать эту производную?

Как подсчитать эту производную?

1. Сначала $j \in$Выходы: вес w_{ji} входит в перцептрон последнего (выходного) слоя.
2. Каждый нейрон расчитывает взвешенную сумму своих входов:
   $$a_j = \sum_i w_{ji} z_i,$$
   здесь z_i - входы нейрона и соответственно выходы предыдущего слоя нейронов.
3. Выход нейрона i – это преобразование суммы a_i пороговой функцией g: $z_i = g(a_i)$
4. $$\frac{\partial E_s}{\partial w_{ji}} = \frac{\partial E_s}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \frac{\partial E_s}{\partial a_j} \frac{\partial \sum_i w_{ji} z_i}{\partial w_{ji}} = \frac{\partial E_s}{\partial a_j} z_i = \delta_j z_i.$$
   δ_j - значение ошибки.

Значение ошибки (дельта) для каждого слоя

1. Для выходного слоя

$$\delta_k = \frac{\partial E_s}{\partial a_k} = \left| \begin{array}{c} a_k = \sum_j w_{jk} z_j \\ y_k = f(a_k) \end{array} \right| = \frac{\partial E_s}{\partial y_k} \frac{\partial y_k}{\partial a_k}.$$

2. $\partial y_k / \partial a_k$ - производная пороговой функции $f'(a) = f(a)(1 - f(a))$.
3. $\partial E_s / \partial y_k$ - производная от уже известной функции

$$E_s(\{w_{ij}\}) = \frac{1}{2} \sum_{k \in \text{Выходы}} \left(y_k^{(d)} - y_k^*(d) \right)^2,$$

   которая равна $y_k^{(d)} - y_k^*(d)$.
4. Итог

$$\delta_k = f'(y_k^{(d)} - y_k^*(d))$$

Ошибка для промежуточного слоя

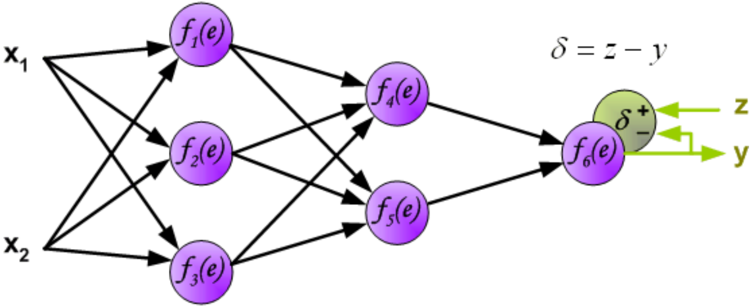
Аналогично:

$$\delta_j = f'(a_j) \sum_k \delta_k w_{jk}$$

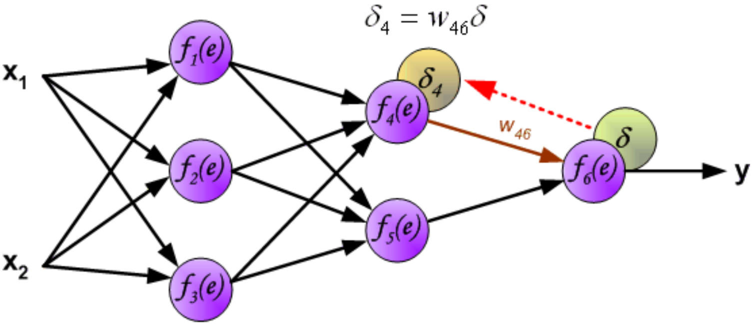
Здесь суммирование происходит по всем k, к которым нейрон j посылает сигнал

Ошибка на каждом слое вычисляется рекурсивно через значения ошибки на предыдущих слоях: коррекция ошибки распространяется обратно по нейронной сети

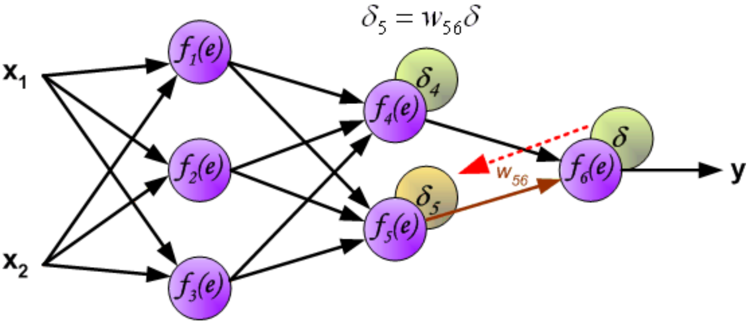
Алгоритм обратного распространения ошибки



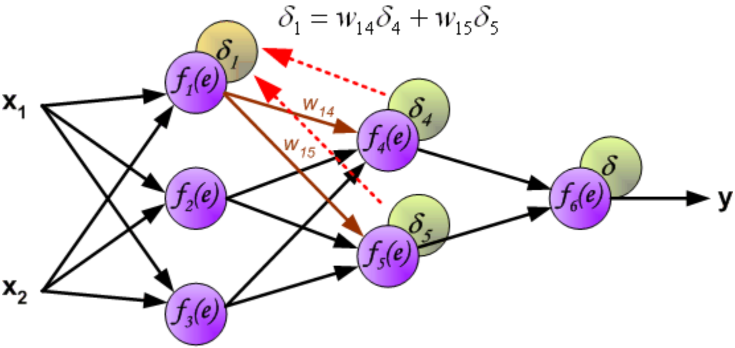
Алгоритм обратного распространения ошибки



Алгоритм обратного распространения ошибки

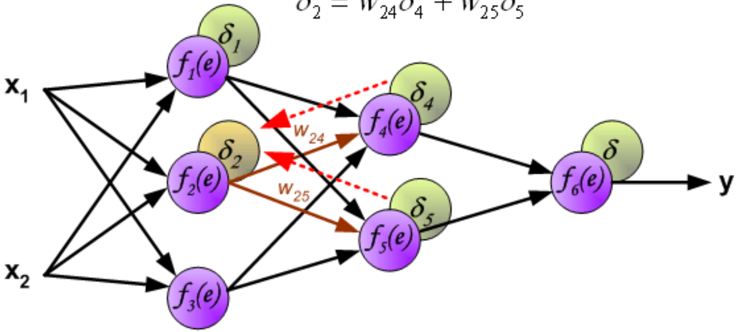


Алгоритм обратного распространения ошибки



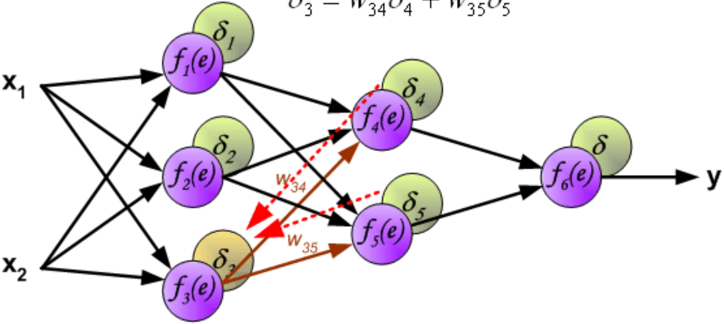
Алгоритм обратного распространения ошибки

$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5$$



Алгоритм обратного распространения ошибки

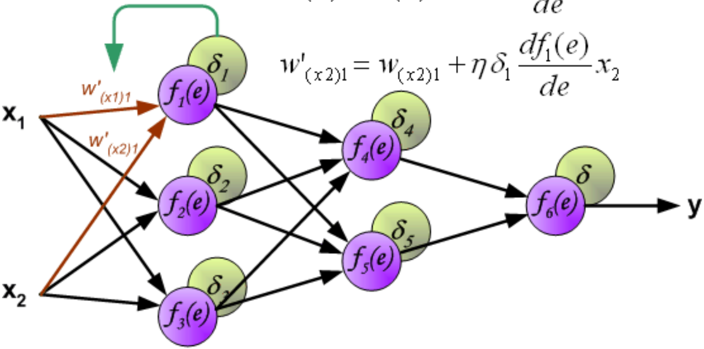
$$\delta_3 = w_{34}\delta_4 + w_{35}\delta_5$$



Алгоритм обратного распространения ошибки

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

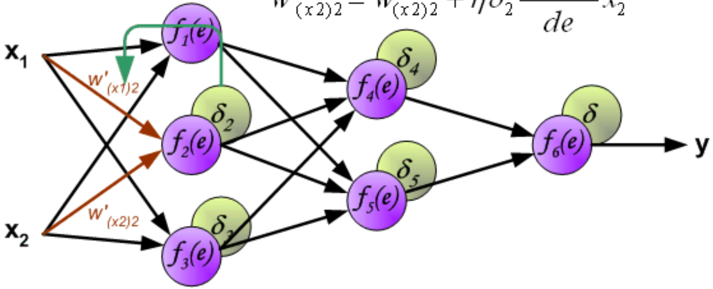
$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



Алгоритм обратного распространения ошибки

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

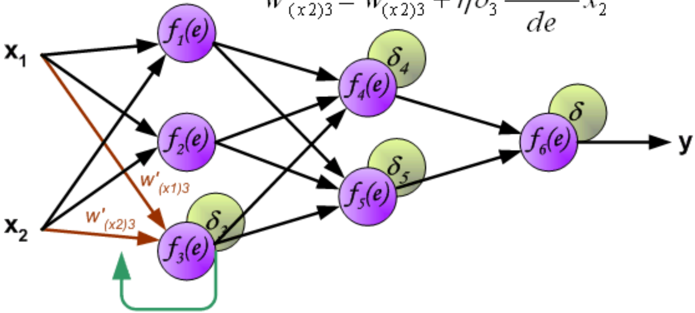
$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



Алгоритм обратного распространения ошибки

$$w'_{(x1)3} = w_{(x1)3} + \eta\delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta\delta_3 \frac{df_3(e)}{de} x_2$$

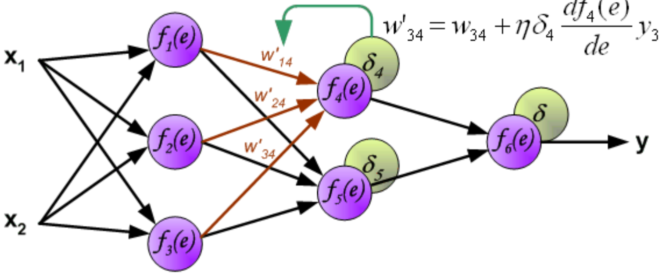


Алгоритм обратного распространения ошибки

$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$

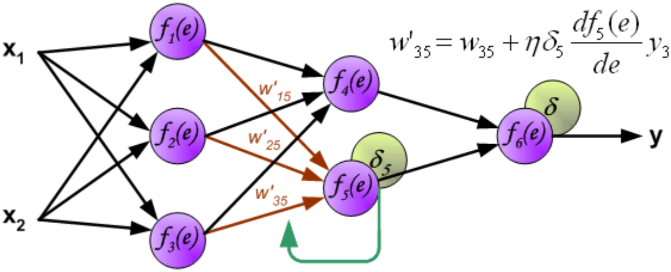


Алгоритм обратного распространения ошибки

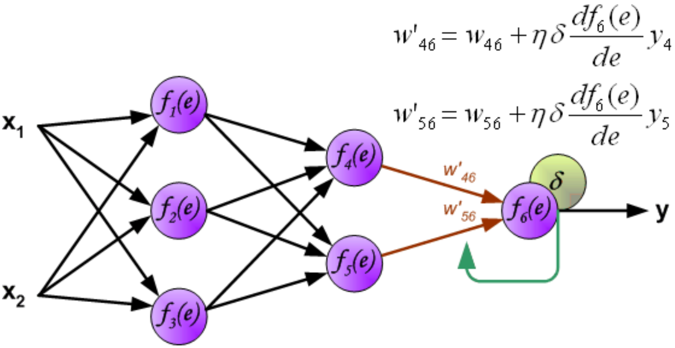
$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$



Алгоритм обратного распространения ошибки



Нейросетевое программное обеспечение

сайт: http://www.i-intellect.ru/neural-networks/programs.html

- EasyNN - Нейросетевое программное обеспечение для Windows с числовыми, текстовыми и графичискими функциями.
- NeuralWorks - Professional II/PLUS - среда для разработки нейронных сетей для Windows и Unix. Predict - нейросетевой инструмент как надстройка Excel для Windows
- Neuro Office - ориентирован на проектирование нейронных сетей с ядерной организацией
- Deductor — платформа для создания законченных аналитических решений.
- **neuralnet** - пакет в R

Предсказания размера пенсии в зависимости средней зарплаты на R

```
# средняя зарплата за каждый год
traininginput <- c(0.225, 690, 2313, 2931, 4061, 4937, 5809, 7096,
8803, 10095, 12229, 13572)
# средняя пенсия за каждый год
trainingoutput <- c(0.118, 274, 949, 1270, 1668, 2001, 2434, 3028,
3393, 4519, 5594, 7610)
# # данные для обучения
trainingdata <- cbind(traininginput,trainingoutput)
colnames(trainingdata) <- c("Input "Output")
# # обучение
net.pension <- neuralnet(Output~Input,trainingdata, hidden=10,
threshold=0.01)
print(net.pension)
# # тестирование
```

Вопросы

?