Практический анализ данных и машинное обучение
Искусственные нейронные сети. Начало

Кашницкий Юрий

Национальный исследовательский университет Высшая школа экономики
Факультет компьютерных наук

12 мая 2018

План лекции

1. Модель нейронной сети
   - Модель нейрона
   - Представление простых булевых блоков перцептронами
   - Решение XOR-проблемы с помощью нейронной сети
   - Модель нейронной сети
2. Аглоритм обратного распространения ошибки
   - Этапы вычисления градиента функции ошибки
   - Прямое распространение ошибки
   - Обратное распространение ошибки

План лекции

1. Модель нейронной сети
   - Модель нейрона
   - Представление простых булевых блоков перцептронами
   - Решение XOR-проблемы с помощью нейронной сети
   - Модель нейронной сети
2. Аглоритм обратного распространения ошибки
   - Этапы вычисления градиента функции ошибки
   - Прямое распространение ошибки
   - Обратное распространение ошибки

Модель нейрона

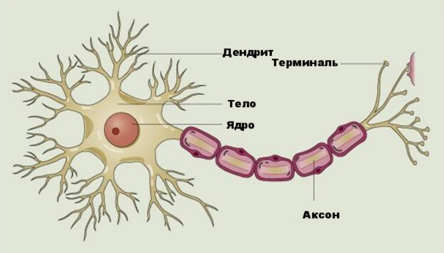


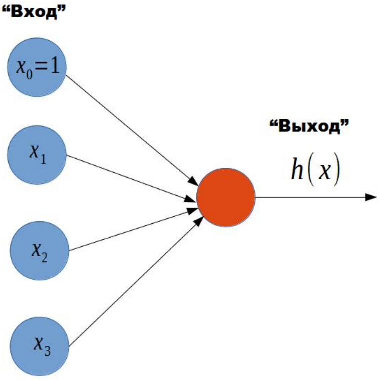
Схема нейрона

В мозгу человека примерно 100 млрд. нейронов. Каждый нейрон - передатчик нервного импульса (возбуждения).
Дендрит – отросток, передающий возбуждение к телу нейрона («вход»).
Аксон – обычно длинный отросток нейрона, приспособленный для проведения возбуждения от тела нейрона («выход»)

Модель перцептрона с линейной функцией активации

- Вход: $\vec{\mathrm{x}} = [1, \mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3]^{\mathrm{T}}$
- Параметры: $\vec{\beta} = [\beta_0, \beta_1, \beta_2, \beta_3]^{\mathrm{T}}$
- Выход: $\mathrm{h}_{\vec{\beta}}(\vec{\mathrm{x}}) = \vec{\beta}^{\mathrm{T}}\vec{\mathrm{x}}$

Линейность функции активации означает, что нейрон (перцептрон) «возбуждается» при $\vec{\beta}^{\mathrm{T}}\vec{\mathrm{x}} \geq 0$, то есть в данном случае при $\beta_1\mathrm{x}_1 + \beta_2\mathrm{x}_2 + \beta_3\mathrm{x}_3 \geq -\beta_0$.



Модель перцептрона

Модель перцептрона с логистической функцией активации

- Вход: $\vec{x} = [1, x_1, x_2, x_3]^T$
- Параметры: $\vec{\beta} = [\beta_0, \beta_1, \beta_2, \beta_3]^T$
- Выход: $h_{\vec{\beta}}(\vec{x}) = \sigma(\vec{\beta}^T\vec{x}) = \frac{1}{1+e^{-\vec{\beta}^T\vec{x}}}$

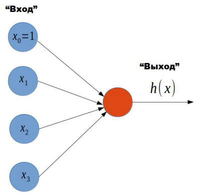
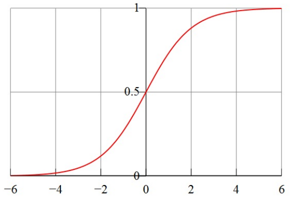
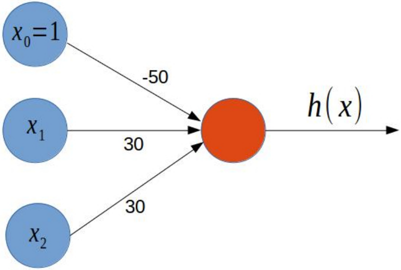


Рис. 3: Модель перцептрона



У сигмоид-фунцкии «удобная» производная: $\frac{d}{dy}\sigma(y) = \sigma(y)(1-\sigma(y))$

Блок AND в виде перцептрона



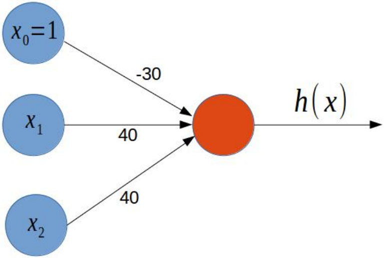
x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	$h_{\vec{\beta}}(\vec{x})$
0	0	$\sigma(-50) \approx 0$
0	1	$\sigma(-20) \approx 0$
1	0	$\sigma(-20) \approx 0$
1	1	$\sigma(10) \approx 1$

- Вход: $x_0 = 1, x_1, x_2 \in \{0, 1\}$
- Параметры: $\beta_0 = -50, \beta_1 = 30, \beta_2 = 30$
- Выход: $h_{\vec{\beta}}(\vec{x}) = \sigma(-50 + 30x_1 + 30x_2)$, где $\sigma(x) = \frac{1}{1+e^{-\vec{\beta}^T\vec{x}}}$

$$h_{\vec{\beta}}(\vec{x}) \approx x_1 \text{ AND } x_2$$

Блок OR в виде перцептрона



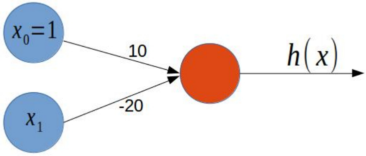
x_1	x_2	x_1 OR x_2
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	$h_{\vec{\beta}}(\vec{x})$
0	0	$\sigma(-30) \approx 0$
0	1	$\sigma(10) \approx 1$
1	0	$\sigma(10) \approx 1$
1	1	$\sigma(50) \approx 1$

- Вход: $x_0 = 1, x_1, x_2 \in \{0, 1\}$
- Параметры: $\beta_0 = -30, \beta_1 = 40, \beta_2 = 40$
- Выход: $h_{\vec{\beta}}(\vec{x}) = \sigma(-30 + 40x_1 + 40x_2)$, где $\sigma(x) = \frac{1}{1+e^{-\vec{\beta}^T \vec{x}}}$

$$h_{\vec{\beta}}(\vec{x}) \approx x_1 \text{ OR } x_2$$

Блок NOT в виде нейрона



- Выход:
  $h_{\vec{\beta}}(\vec{x}) = \sigma(10 - 20x_1)$,
  где $\sigma(\vec{x}) = \frac{1}{1+e^{-\vec{\beta}^T\vec{x}}}$

x_1	NOT x_1
0	1
1	0

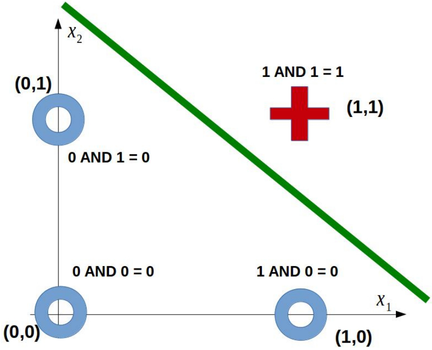
x_1	$h_{\vec{\beta}}(\vec{x})$
0	$\sigma(10) \approx 1$
1	$\sigma(-10) \approx 0$

- Вход:
  $x_0 = 1, x_1 \in \{0, 1\}$
- Параметры:
  $\beta_0 = 10, \beta_1 = -20$

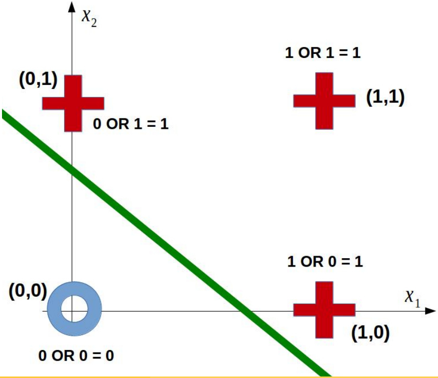
$$h_{\vec{\beta}}(\vec{x}) \approx \text{NOT } x_1$$

Линейная разделимость классов

Если посмотреть на прошлые примеры как на задачу бинарной классификации, видно, что классы линейно разделимы. В случае AND классы можно разделить прямой $-50 + 30x_1 + 30x_2 = 0$.

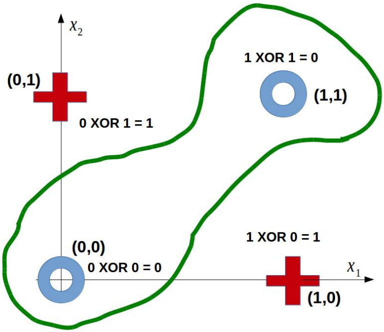


В случае OR классы можно разделить прямой $-30 + 40x_1 + 40x_2 = 0$.



Проблема нелинейного разделения

Однако уже в простейшем случае известной проблемы «исключающего ИЛИ» (the XOR problem) классы нельзя разделить одной прямой. Разделяющая граница будет нелинейной, и она уже не может быть представлена одним перцептроном.



x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

Представление функции XOR

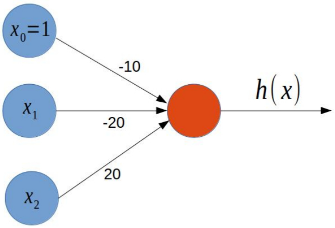
Для решения XOR-проблемы приходится уже комбинировать простые логические блоки, представленные перцептронами.
x_1 XOR x_2 = ((NOT x_1) AND x_2) OR (x_1 AND (NOT x_2)).

x_1	x_2	$a_2 = x_1$ AND (NOT x_2)
0	0	0
0	1	0
1	0	1
1	1	0

x_1	x_2	$a_1 =$ (NOT x_1) AND x_2
0	0	0
0	1	1
1	0	0
1	1	0

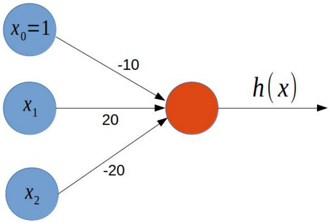
x_1	x_2	x_1 XOR x_2	a_1 OR a_2
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Составные блоки сети, решающей XOR-проблему



$$h_{\vec{\beta}_1}(\vec{x}) \approx (\text{NOT } x_1) \text{ AND } x_2$$

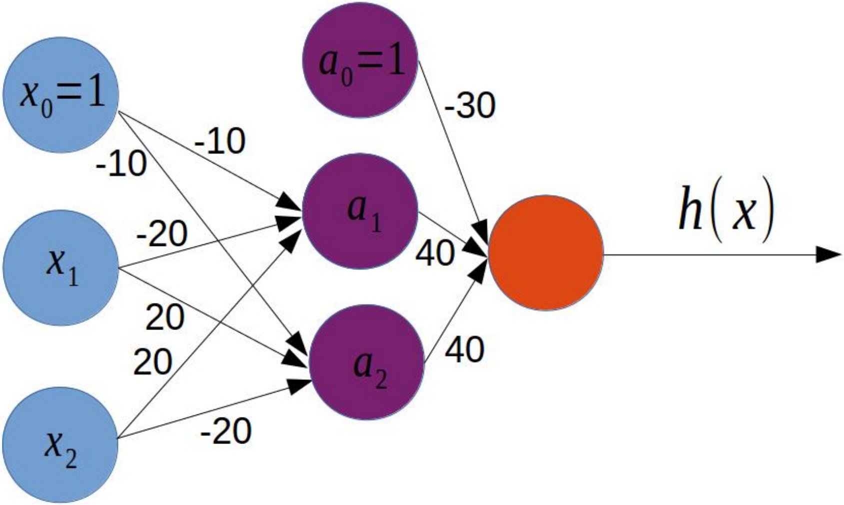
x_1	x_2	$h_{\vec{\beta}}(\vec{x})$
0	0	$\sigma(-10) \approx 0$
0	1	$\sigma(10) \approx 1$
1	0	$\sigma(-30) \approx 0$
1	1	$\sigma(-10) \approx 0$



$$h_{\vec{\beta}_2}(\vec{x}) \approx x_1 \text{ AND } (\text{NOT } x_2)$$

x_1	x_2	$h_{\vec{\beta}}(\vec{x})$
0	0	$\sigma(-10) \approx 0$
0	1	$\sigma(-30) \approx 0$
1	0	$\sigma(10) \approx 1$
1	1	$\sigma(-10) \approx 0$

Нейронная сеть для решения XOR-проблемы



Возможности нейронных сетей по представлению функций

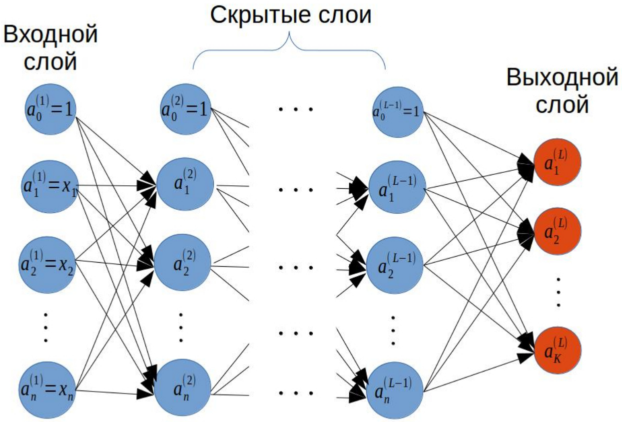
Утверждения[a]

[a] Machine Learning, Глава 4, стр. 105. T. Mitchell, McGraw Hill, 1997

- Любая булева функция представима в виде нейронной сети с одним скрытым слоем с нелинейной функцией активации нейрона (но может потребоваться экспоненциально много нейронов в скрытом слое).
- Любая непрерывная и ограниченная функция может быть сколь угодно точно аппроксимирована нейронной сетью с одним скрытым слоем с нелинейной функцией активации нейрона.
- Любая функция может быть сколь угодно точно аппроксимирована нейронной сетью с двумя скрытыми слоями с нелинейной функцией активации нейрона.

Модель нейронной сети

Классификация на K классов. Вход: ℓ примеров $\{(\mathrm{x_i}, \mathrm{y_i})\}, \mathrm{x_i} \in \mathrm{R^d}, \mathrm{y_i} \in \mathrm{R^K}$



L – число слоев, d – число признаков примера.

Функция ошибки для нейронной сети

Логистическая регрессия (с регуляризацией):

$$\mathrm{Cost}(\beta) = -\frac{1}{\ell}\sum_{i=1}^{\ell}[y^{(i)}\log(h_\beta(x^{(i)})) + (1-y^{(i)})\log(1-h_\beta(x^{(i)}))] + \frac{\lambda}{2\ell}\sum_{j=1}^{d}\beta_j^2.$$

Именно такая функция ошибки выпуклая и может быть минимизирована алгоритмами типа градиентного спуска.

(А квадратичная функция ошибки $\mathrm{Cost}(\beta) = \frac{1}{\ell}\sum_{i=1}^{\ell}(h_\beta(x^{(i)}) - y^{(i)})^2$ не будет выпуклой при нелинейной $h_{\vec\beta}(\vec x) = \sigma(\vec\beta^T\vec x)$).

Нейронная сеть:

$$\mathrm{Cost}(\beta) = -\frac{1}{\ell}\sum_{i=1}^{\ell}\sum_{k=1}^{K}[y_k^{(i)}\log(h_\beta(x^{(i)})_k) + (1-y_k^{(i)})\log(1-h_\beta(x^{(i)})_k)] +$$

$$\frac{\lambda}{2\ell}\sum_{m=1}^{L-1}\sum_{i=1}^{s_m}\sum_{j=1}^{s_{l+1}}(\beta_{ij}^{(m)})^2.$$

Здесь s_m – число нейронов в слое m, $h_\beta(x^{(i)})_k$ – выход нейрона k, $\beta_{ij}^{(m)}$ – параметр (вес) на входе нейрона i в слое m, “пришедший” от нейрона j предшествующего слоя.

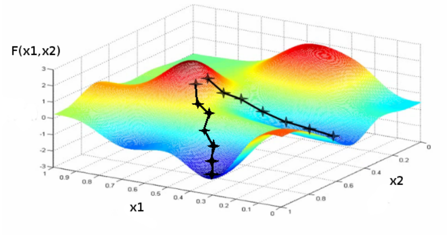
Градиентный спуск

На примере задачи минимизации функции $F(x_1, x_2)$

η – темп обучения, случайно выбираются начальные значения $x_1^{(0)}$ и $x_2^{(0)}$.
Повторяется до сходимости:

- $x_1^{(t+1)} = x_1^{(t)} - \eta \frac{\partial}{\partial x_1} F(x_1^{(t)}, x_2^{(t)})$
- $x_2^{(t)} = x_2^{(t+1)} - \eta \frac{\partial}{\partial x_2} F(x_1^{(t)}, x_2^{(t)})$

Чтобы найти минимум функции ошибки $\min(Cost(\beta))$ методом градиентного спуска, надо найти ее производные по параметрам: $\frac{\partial}{\partial \beta_{ij}^{(m)}} Cost(\beta)$.



План лекции

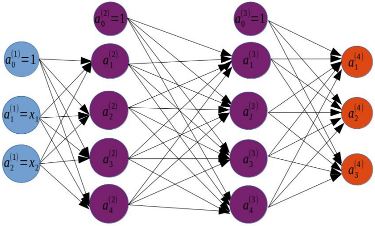
Этапы вычисления градиента функции ошибки

В случае итерационного обучения сети (входные объекты обрабатываются один за одним) градиенты функции ошибки для каждого нейрона вычисляются в два этапа:

- Прямое распространение ошибки (forward propagation)
- Обратное распространение ошибки (back propagation)

Прямое распространение ошибки

Рассмотрим пример нейронной сети из 4 слоев ($L = 4$) для задачи классификации с 3 классами ($K = 3$). В скрытых слоях по 4 нейрона ($s_2 = s_3 = 4$).



$$B^{(m)} = \begin{pmatrix} \beta_{11}^{(m)} & \dots & \beta_{1(s_\ell+1)}^{(\ell)} \\ . & . & . \\ \beta_{s_{m+1}1}^{(\ell)} & \dots & \beta_{s_{m+1}(s_m+1)}^{(m)} \end{pmatrix}$$

Для одного входного объекта (x, y):

$\vec{a}^{(1)} = \vec{x} = [1;\ x_1;\ x_2]$

$\vec{a}^{(2)} = [1;\ \sigma(B^{(1)}\vec{a}^{(1)})]$ (доб-ся $a_0^{(2)}$)

$\vec{a}^{(3)} = [1;\ \sigma(B^{(2)}\vec{a}^{(2)})]$ (доб-ся $a_0^{(3)}$)

$\vec{a}^{(4)} = \sigma(B^{(3)}\vec{a}^{(3)}) = h_{\vec{\beta}}(\vec{x})$

Накопилась «ошибка предсказания» $\vec{\delta}^{(4)} = \vec{y} - \vec{a}^{(4)}$.

Будем обозначать $\delta_j^{(m)}$ – эту “накопленную ошибку” нейрона j в слое m.

Обратное распространение ошибки

Обучающая выборка $\{(\vec{x}^{(1)}, \vec{y}^{(1)}), ..., (\vec{x}^{(\ell)}, \vec{y}^{(\ell)})\}$.

Коэффициенты $\beta_{ij}^{(m)}$ инициализируются малыми случайными числами.

Коэфф-ты ошибок в каждом слое инициалицируются нулями: $\Delta_{ij}^{(m)} = 0$.

В цикле по всем объектам $i = 1 \dots \ell$:

- На вход подаются признаки $\vec{x}^{(i)} : \vec{a}^{(1)} = \vec{x}^{(i)}$. С помощью прямого распространения ошибки считаются активации $\vec{a}^{(m)}$ в каждом слое.
- Ошибка в последнем слое: $\vec{\delta}^{(L)} = \vec{y}^{(i)} - \vec{a}^{(L)}$
- Последовательно от предпоследнего слоя ко второму вычисляются ошибки $\vec{\delta}^{(m)}$ в каждом слое:[a]
  $\vec{\delta}^{(L-1)} = \left(B^{(L-1)}\right)^T \vec{\delta}^{(L)} \vec{a}^{(L-1)}(\vec{1} - \vec{a}^{(L-1)})$, ..., $\vec{\delta}^{(2)} = \left(B^{(2)}\right)^T \vec{\delta}^{(3)} \vec{a}^{(2)}(\vec{1} - \vec{a}^{(2)})$
- Ошибки в каждом слое обновляются: $\Delta^{(m)} = \Delta^{(m)} + \vec{\delta}^{(m+1)}\left(\vec{a}^{(m)}\right)^T$

$$\frac{\partial}{\partial \beta_{ij}^{(m)}} \mathrm{Cost}(\vec{\beta}) = \frac{1}{m} \Delta_{ij}^{(m)} + \begin{cases} \lambda \beta_{ij} & \text{если } j \neq 0 \\ 0 & \text{если } j = 0 \end{cases}$$

[a] Вывод для сети с одним скрытым слоем: http://goo.gl/dTJr17,
Вывод в общем случае, но для квадратичной функции ошибки:
http://inst.eecs.berkeley.edu/~cs182/sp06/notes/backprop.pdf

Алгоритм обучения

Обучающая выборка $\{(\mathrm{x}^{(1)}, \mathrm{y}^{(1)}), \ldots, (\mathrm{x}^{(\ell)}, \mathrm{y}^{(\ell)})\}$.

- Коэффициенты $\beta_{\mathrm{ij}}^{(\mathrm{m})}$ инициализируются малыми случайными числами
- С помощью прямого распространения ошибки считаются выходы $\mathrm{h}_{\vec{\beta}}(\vec{\mathrm{x}}^{(\mathrm{i})})$ для каждого примера $\vec{\mathrm{x}}^{(\mathrm{i})}$
- Вычисляется функция ошибки $\mathrm{Cost}(\vec{\beta})$
- С помощью обратного распространения ошибки вычисляются производные $\frac{\partial}{\partial \beta_{\mathrm{ij}}^{(\mathrm{m})}} \mathrm{Cost}(\vec{\beta})$
- Функция ошибки $\mathrm{Cost}(\vec{\beta})$ минимизируется с помощью алгоритма градиентного спуска, его стохастической версии других более совершенных алгоритмов оптимизации. Для этого нужны вычисленные ранее производные функции ошибки по параметрам $\vec{\beta}$.

Замечания

- Из-за нелинейности функции ошибки возможно, что результатом оптимизации будет локальный минимум, а не глобальный (но на практике результаты хороши, больше проблем вызывают седловые точки). Можно осуществлять оптимизацию несколько раз с разными начальными параметрами
- Для увеличения вероятности нахождения глобального минимума функции ошибки можно регулировать скорость обучения (сначала быстро, потом медленней)
- У нейронной сети обычно много параметров, поэтому роль регуляризации в борьбе с переобучением очень высока (сейчас на практике больше пользуются дропаутом и нормализацией батча)
- Обучение сети может быть долгим, зато потом использование обученной сети быстрое

Что обсудим дальше

- Эффективные векторизованные вычисления (NumPy)
- Простой линейный подход к классификации изображений, Softmax-классификатор
- Как эффективно реализовать backpropagation
- Что такое граф вычислений и автоматическое дифференцирование
- Какие методы оптимизации сейчас используются
- Какая используется регуляризация (batchnorm, dropout)
- Какие еще трюки позволяют эффективно обучать нейронные сети (активации, инициализация весов, настройка гиперпараметров)
- Как учитывать двухмерную структуру объекта-изображения (сверточные нейронные сети)
- Поработаем с библиотекой PyTorch