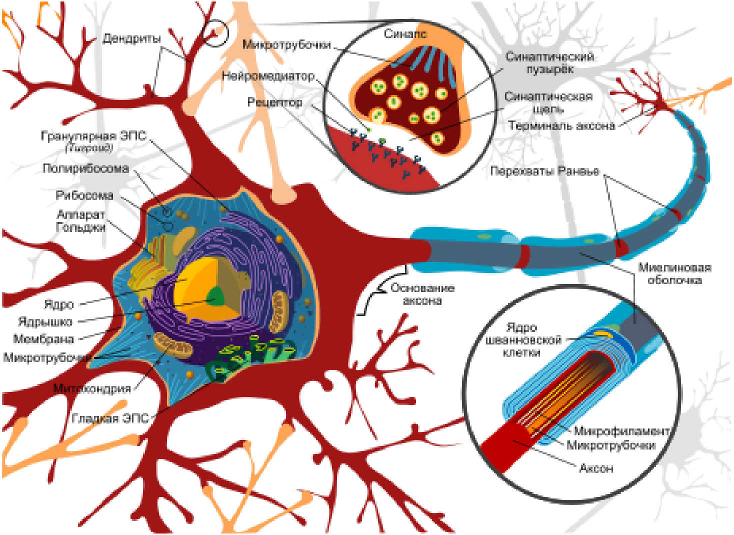
Машинное обучение

Лекция 12. Нейронные сети.

Катя Тузова

Разбор летучки

Нейрон


Дендриты
Микротрубочки
Нейромедиатор
Рецептор
Синапс
Синаптический пузырёк
Синаптическая щель
Терминаль аксона
Гранулярная ЭПС
(Тигроид)
Полирибосома
Рибосома
Аппарат Гольджи
Перехваты Ранвье
Миелиновая оболочка
Основание аксона
Ядро
Ядрышко
Мембрана
Микротрубочки
Митохондрия
Гладкая ЭПС
Ядро шванновской клетки
Микрофиламент
Микротрубочки
Аксон

Линейная модель нейрона

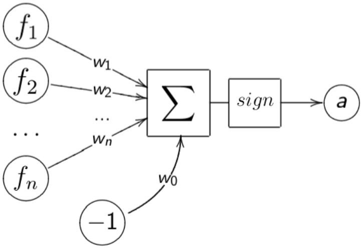
Модель МакКаллока-Питтса:

$f_j : X \to R, \qquad j = 1, \dots, n$ — числовые признаки

$$a(x, w) = \sigma(\langle w, x_i \rangle) = \sigma\left(\sum_{j=1}^{n} w_j f_j(x) - w_0\right)$$

где $w_0, w_1, \dots, w_n \in R$ – веса признаков

$\sigma(s)$ – функция активации (например, sign)



Линейные алгоритмы классификации и регрессии

Задача классификации:

$$Y = \{\pm 1\}, \qquad a(x, w) = \operatorname{sign}\langle w, x_i \rangle$$

$$Q(w; X^l) = \sum_{i=1}^{l} \mathcal{L}(\langle w, x_i \rangle y_i) \to \min_w$$

Задача регрессии:

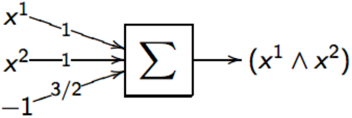
$$Y = \mathbb{R}, \qquad a(x, w) = \sigma(\langle w, x_i \rangle)$$

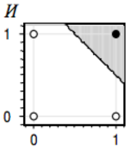
$$Q(w; X^l) = \sum_{i=1}^{l} (\sigma(\langle w, x_i \rangle) - y_i)^2 \to \min_w$$

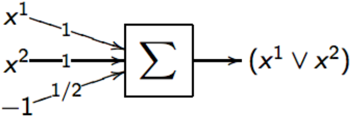
Вопрос

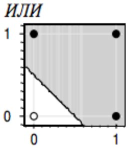
Какой класс функций можно реализовать нейроном?

Нейронная реализация логических функций









Нейронная реализация логических функций

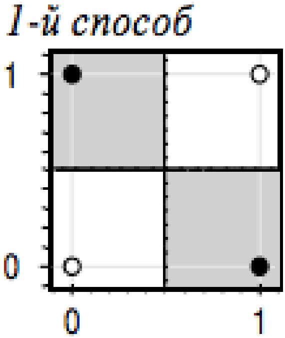
Функции И, ИЛИ, НЕ от бинарных переменных x_1 и x_2:

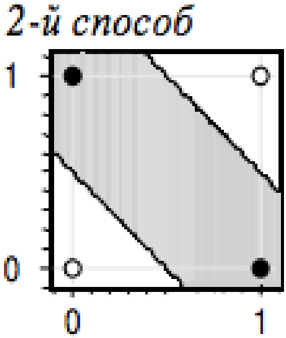
$$x_1 \wedge x_2 = x_1 + x_2 - \frac{3}{2} > 0$$

$$x_1 \vee x_2 = x_1 + x_2 - \frac{1}{2} > 0$$

$$\neg x_1 = -x_1 + \frac{1}{2} > 0$$

Логическая функция XOR



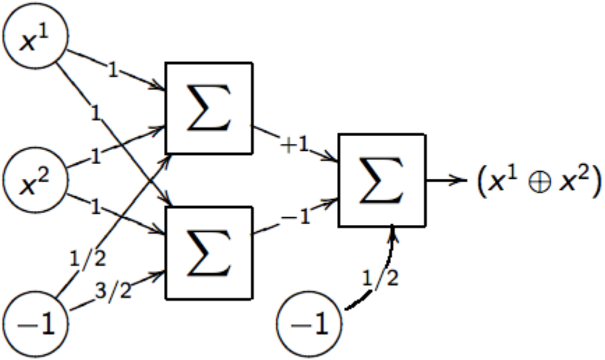


Логическая функция XOR

Функция $x_1 \oplus x_2 = [x_1 \neq x_2]$ не реализуема одним нейроном.
Два способа реализации:

- Добавлением нелинейного признака:
  $x_1 \oplus x_2 = \left[x_1 + x_2 + 2x_1 x_2 - \tfrac{1}{2} > 0\right]$
- Сетью (двухслойной суперпозицией) функций И, ИЛИ, НЕ:
  $x_1 \oplus x_2 = \left[(x_1 \vee x_2) - (x_1 \wedge x_2) - \tfrac{1}{2} > 0\right]$

Логическая функция XOR



Любую ли функцию можно представить нейросетью?

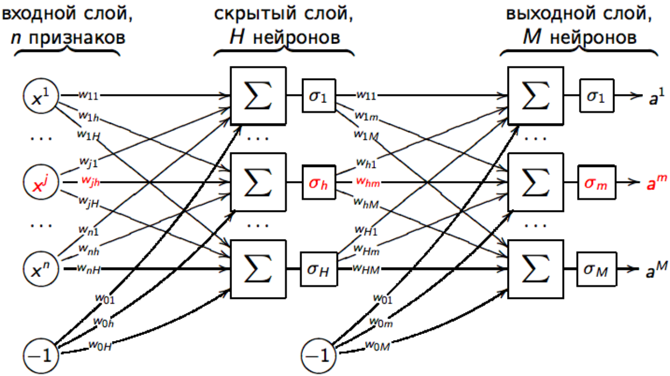
- Двухслойная сеть в $\{0,1\}^n$ позволяет реализовать произвольную булеву функцию.
- Двухслойная сеть в $\mathbb{R}^n$ позволяет отделить произвольный выпуклый многогранник.
- Трёхслойная сеть $\mathbb{R}^n$ позволяет отделить произвольную многогранную область, не обязательно выпуклую, и даже не обязательно связную.
- С помощью линейных операций и одной нелинейной функции активации σ можно приблизить любую непрерывную функцию с любой желаемой точностью

Практические рекомендации

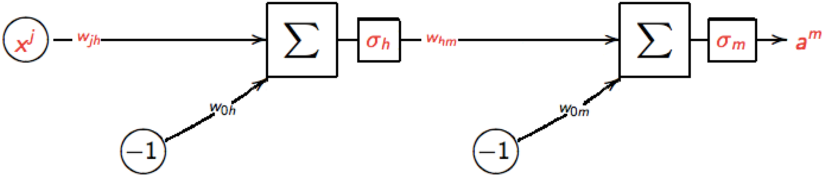
- Двух-трёх слоёв обычно достаточно.
- Можно достраивать нейроны в произвольных местах сети по необходимости.

Многослойная нейронная сеть

Пусть $Y = \mathbb{R}^M$, два слоя в сети.



Многослойная нейронная сеть



Стохастический градиент

Задача минимизации суммарных потерь:

$$Q(\mathbf{w}) = \sum_{i=1}^{l} \mathcal{L}(w, x_i, y_i) \to \min_w$$

Input: X^l, α, η

Output: $w = (w_{jh}, w_{hm}) \in \mathbb{R}^{H(n+M+1)+M}$

Перемешать данные в X^l

Инициализировать: w, $Q(w)$

Повторить пока Q и/или w не стабилизируются:

  Взять x_i из X^l

  Потеря: $\varepsilon_i = \mathcal{L}(w, x_i, y_i)$

  Градиентный шаг: $w = w - \alpha \nabla \mathcal{L}(w, x_i, y_i)$

  Оценить $Q = (1 - \eta)Q + \eta\varepsilon_i$

Вопрос

Сколько операций потребуется для вычисления градиента в точке?

Идея

Сохранять некоторые промежуточные результаты в узлах сети.

Задача дифференцирования суперпозиции функций

Выходные значения $a^m(x_i)$, $m = 1, \dots, M$ на объекте x_i:

$$a^m(x_i) = \sigma_m\Bigl(\sum_{h=0}^{H} w_{hm} u^h(x_i)\Bigr), \qquad u^h(x_i) = \sigma_h\Bigl(\sum_{j=0}^{n} w_{jh} f_j(x_i)\Bigr)$$

Возьмем $\mathcal{L}_i(w)$ – средний квадрат ошибки:

$$\mathcal{L}_i(w) = \frac{1}{2} \sum_{m=1}^{M} (a^m(x_i) - y_i^m)^2$$

Промежуточная задача: найти частные производные

$$\frac{\partial \mathcal{L}_i(w)}{\partial a^m} \qquad \frac{\partial \mathcal{L}_i(w)}{\partial u^h}$$

Быстрое вычисление градиента

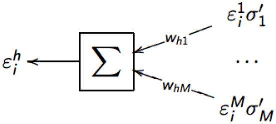
Ошибка на выходном слое:

$$\frac{\partial \mathcal{L}_i(w)}{\partial a^m} = a^m(x_i) - y_i^m = \varepsilon_i^m$$

Ошибка на скрытом слое:

$$\frac{\partial \mathcal{L}_i(w)}{\partial u^h} = \sum_{m=1}^{M} (a^m(x_i) - y_i^m)\sigma'_m w_{hm} = \sum_{m=1}^{M} \varepsilon_i^m \sigma'_m w_{hm} = \varepsilon_i^h$$

ε_i^h вычисляется по ε_i^m, если запустить сеть «задом наперёд»:



Быстрое вычисление градиента

Теперь, имея частные производные $\mathcal{L}_i(w)$ по a^m и u^h, легко выписать градиент $\mathcal{L}_i(w)$ по весам w.

Вопрос: как?

Быстрое вычисление градиента

$$\frac{\partial \mathcal{L}_i(w)}{\partial w_{hm}} = \frac{\partial \mathcal{L}_i(w)}{\partial a^m} \frac{\partial a^m}{\partial w_{hm}} = \varepsilon_i^m \sigma'_m u^h(x_i),$$

$$m = 1, \dots, M, \;\; h = 0, \dots, H$$

$$\frac{\partial \mathcal{L}_i(w)}{\partial w_{jh}} = \frac{\partial \mathcal{L}_i(w)}{\partial u^h} \frac{\partial u^h}{\partial w_{jh}} = \varepsilon_i^h \sigma'_h f_j(x_i),$$

$$h = 1, \dots, H, \;\; j = 0, \dots, n$$

Алгоритм обратного распространения ошибки

Input: $X^l = (x_i, y_i)_{i=1}^l \subset \mathbb{R}^n \times \mathbb{R}^M$, H, α, η

Output: синаптические веса w_{jh}, w_{hm}

Повторять пока Q не стабилизируется:

выбрать объект x_i из X^l

$$\begin{cases} u_i^h = \sigma_h\Big(\sum\limits_{j=0}^{J} w_{jh} x_i^j\Big), \quad h = 1, \dots, H \\ a_i^m = \sigma_m\Big(\sum\limits_{h=0}^{H} w_{hm} u_i^h\Big), \quad \varepsilon_i^m = a_i^m - y_i^m, \quad m = 1, \dots, M \\ \mathcal{L}_i = \sum\limits_{m=1}^{M} (\varepsilon_i^m)^2 \end{cases}$$

$$\begin{cases} \varepsilon_i^h = \sum\limits_{m=1}^{M} \varepsilon_i^m \sigma'_m w_{hm}, \quad h = 1, \dots, H \end{cases}$$

$$\begin{cases} w_{hm} = w_{hm} - \alpha \varepsilon_i^m \sigma'_m u_i^h, \quad h = 0, \dots, H, \quad m = 1, \dots, M \\ w_{jh} = w_{jh} - \alpha \varepsilon_i^h \sigma'_h x_i^j, \quad j = 0, \dots, n, \quad h = 1, \dots, H \\ Q = (1-\eta) Q + \eta \mathcal{L}_i \end{cases}$$

Особенности

+ Эффективность: градиент вычисляется за время, сравнимое со временем вычисления самой сети
+ Легко обобщается на любые σ, $\mathcal{L}$
+ Возможно динамическое (потоковое) обучение
+ На сверхбольших выборках не обязательно брать все x_i
+ Возможность распараллеливания

– Возможна медленная сходимость
– Застревание в локальных минимумах
– Проблема переобучения
– Подбор комплекса эвристик

Стандартные эвристики для метода SG

Применимы все те же эвристики, что и в обычном SG.

Напомните.

Стандартные эвристики для метода SG

- Инициализация весов
- Порядок предъявления объектов
- Оптимизация величины градиентного шага
- Регуляризация (сокращение весов)

Новые проблемы

- Выбор функций активации в каждом нейроне
- Выбор числа слоёв и числа нейронов;
- Выбор значимых связей;

Ускорение сходимости

- Тщательный подбор начального приближения
- Адаптивный градиентный шаг
- Метод сопряжённых градиентов и chunking – разбиение суммы $Q(w) = \sum\limits_{i=1}^{l} \mathcal{L}_i(w)$ на блоки

Начальное приближение

Нейроны настраиваются как отдельные линейные алгоритмы:

- либо по случайной подвыборке $X' \subseteq X^l$
- либо по случайному подмножеству входов
- либо из различных случайных начальных приближений

Тем самым обеспечивается различность нейронов.

Оптимизация структуры сети

Динамическое наращивание сети.

- Обучение при заведомо недостаточном числе нейронов H
- После стабилизации $Q(w)$ – добавление нового нейрона и его инициализация путём обучения
- Снова итерации BackProp

Противоположная идея – прореживание сети.

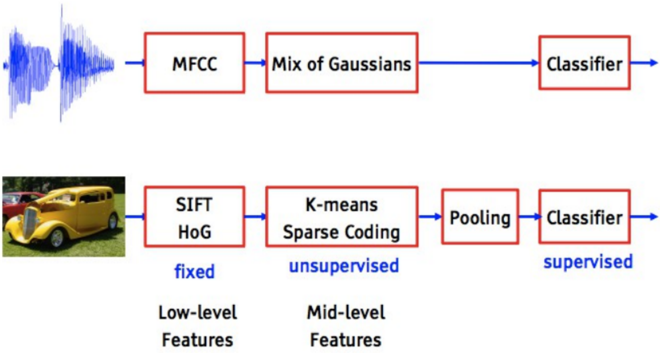
Специальные нейронные сети

Большинство используемых на сегодняшний день методов придуманы в 80-х.
Почему их снова стали использовать?

- Большие вычислительные возможности (в том числе распределенные вычисления)
- Большие объемы данных
- Новые методы предобучения, новые архитектуры

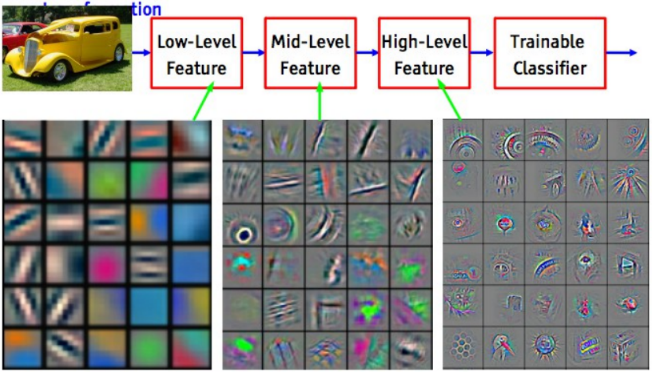
Как было раньше?

Ученые придумывали признаки. На них обучали классификатор.



Deep learning

Извлечение признаков во время обучения.



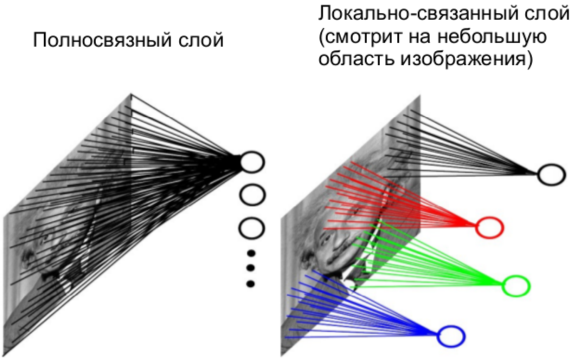
Сверточные нейросети

Архитектура сети выбирается таким образом, чтобы заложить в нее априорные знания из предметной области:

- Пиксель изображения сильнее связан с соседними пикселями (локальная корреляция)
- Объект может встретиться в любой части изображения (инвариантность к перемещению)

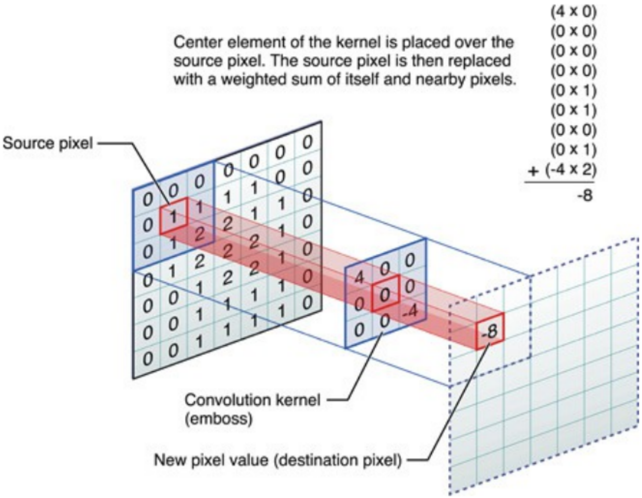
Локально-связанный слой

Закладываем в архитектуру сети априорное знание о том, что соседние пиксели изображения сильнее связаны между собой.

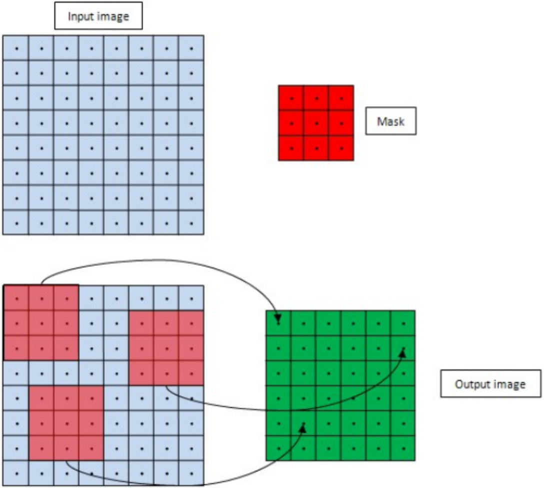


Инвариантность к перемещению

Операция свертки.



Инвариантность к перемещению



Простейшая свертка

Original

$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$$

?

Простейшая свертка

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Простейшая свертка

Original

0	0	0
0	0	1
0	0	0

?

Простейшая свертка

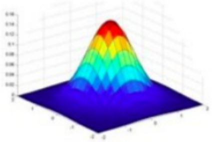
Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Shifted left
By 1 pixel

Фильтр Гаусса

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



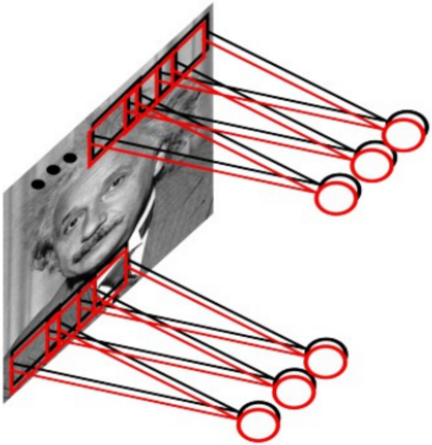
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, σ = 1

Подавление шумов

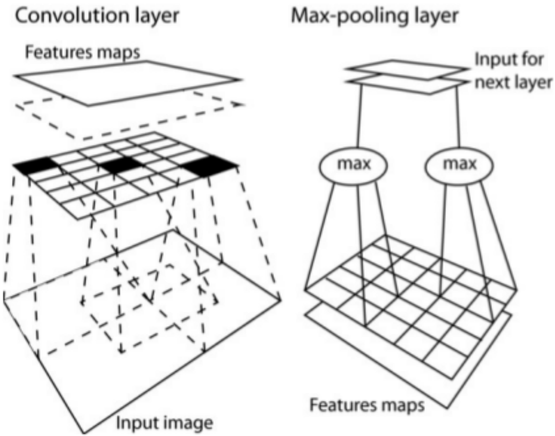
Сверточный слой

Локально-связанный слой с одинаковыми весами в разных частях изображения.
Закладывает в сеть априорное знание о том, что объект может встретиться в любой части изображения.



Subsampling слой

Добавляет устойчивости к небольшим деформациям.
Max-subsampling аналогичен сверточному слою, в котором операция «+» заменена на «max»



Сверточная нейросеть