

Искусственные нейронные сети

К. В. Воронцов
vokov@forecsys.ru

Этот курс доступен на странице вики-ресурса
<http://www.MachineLearning.ru/wiki>
«Машинное обучение (курс лекций, К.В.Воронцов)»

6 мая 2010

Содержание

- 1 Проблема полноты**
 - Напоминания: линейные алгоритмы
 - Проблема «исключающего ИЛИ»
 - Вычислительные возможности нейронных сетей
- 2 Метод обратного распространения ошибок**
 - Многослойная нейронная сеть
 - Быстрое вычисление градиента
 - Алгоритм BackProp
- 3 Эвристики**
 - Стандартные эвристики и ускорение сходимости
 - Диагональный метод Левенберга-Марквардта
 - Нарращивание и прореживание сети

Линейные алгоритмы классификации и регрессии

$f_j: X \rightarrow \mathbb{R}$, $j = 1, \dots, n$ — числовые признаки;

$\langle w, x_i \rangle = \sum_{j=1}^n w_j f_j(x) - w_0$ — линейная комбинация признаков;

$w_0, w_1, \dots, w_n \in \mathbb{R}$ — веса признаков;

Задача классификации: $Y = \{\pm 1\}$, $a(x, w) = \text{sign}\langle w, x_i \rangle$;

$$Q(w; X^\ell) = \sum_{i=1}^{\ell} \mathcal{L}(\underbrace{\langle w, x_i \rangle}_{M_i(w)} y_i) \rightarrow \min_w;$$

Задача регрессии: $Y = \mathbb{R}$, $a(x, w) = \langle w, x_i \rangle$;

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

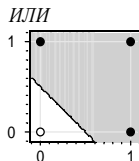
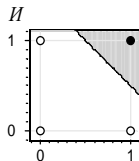
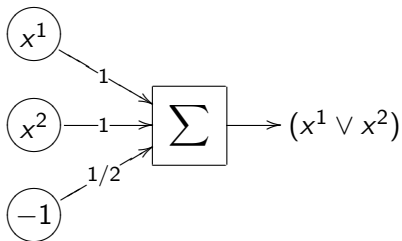
Нейронная реализация логических функций

Функции И, ИЛИ, НЕ от бинарных переменных x^1 и x^2 :

$$x^1 \wedge x^2 = [x^1 + x^2 - \frac{3}{2} > 0];$$

$$x^1 \vee x^2 = [x^1 + x^2 - \frac{1}{2} > 0];$$

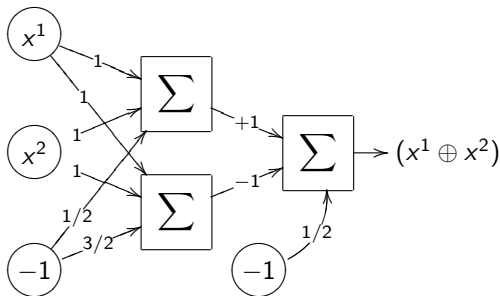
$$\neg x^1 = [-x^1 + \frac{1}{2} > 0];$$



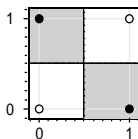
Логическая функция XOR (исключающее ИЛИ)

Функция $x^1 \oplus x^2 = [x^1 \neq x^2]$ не реализуема одним нейроном.
Два способа реализации:

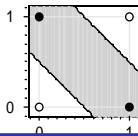
- Добавлением нелинейного признака:
$$x^1 \oplus x^2 = [x^1 + x^2 - 2x^1x^2 - \frac{1}{2} > 0];$$
- **Сетью** (суперпозицией) функций И, ИЛИ, НЕ:
$$x^1 \oplus x^2 = [(x^1 \vee x^2) - (x^1 \wedge x^2) - \frac{1}{2} > 0].$$



1-й способ



2-й способ



Любую ли функцию можно представить нейросетью?

Утверждение

Любая булева функция представима в виде ДНФ, следовательно, и в виде двухслойной сети.

Решение тринадцатой проблемы Гильберта:

Теорема (Колмогоров, 1957)

Любая непрерывная функция n аргументов на единичном кубе $[0, 1]^n$ представима в виде суперпозиции непрерывных функций одного аргумента и операции сложения:

$$f(x^1, x^2, \dots, x^n) = \sum_{k=1}^{2n+1} h_k \left(\sum_{i=1}^n \varphi_{ik}(x^i) \right),$$

где h_k, φ_{ik} — непрерывные функции, и φ_{ik} не зависят от f .

Любую ли функцию можно представить нейросетью?

Теорема (Вейерштрасс)

Любую непрерывную функцию n переменных можно равномерно приблизить полиномом с любой точностью.

Определение

Набор функций F называется *разделяющим точки* множества X , если для любых различных $x, x' \in X$ существует функция $f \in F$ такая, что $f(x) \neq f(x')$.

Теорема (Стоун, 1948)

Пусть X — компактное пространство, $C(X)$ — алгебра непрерывных на X вещественных функций, F — кольцо в $C(X)$, содержащее константу ($1 \in F$) и разделяющее точки множества X . Тогда F плотно в $C(X)$.

Любую ли функцию можно представить нейросетью?

Определение

Набор функций $F \subseteq C(X)$ называется *замкнутым относительно функции* $\varphi : \mathbb{R} \rightarrow \mathbb{R}$, если для любого $f \in F$ выполнено $\varphi(f) \in F$.

Теорема (Горбань, 1998)

Пусть X — компактное пространство, $C(X)$ — алгебра непрерывных на X вещественных функций, F — **линейное подпространство** в $C(X)$, замкнутое относительно нелинейной непрерывной функции φ , содержащее константу ($1 \in F$) и разделяющее точки множества X . Тогда F плотно в $C(X)$.

Любую ли функцию можно представить нейросетью?

Выводы:

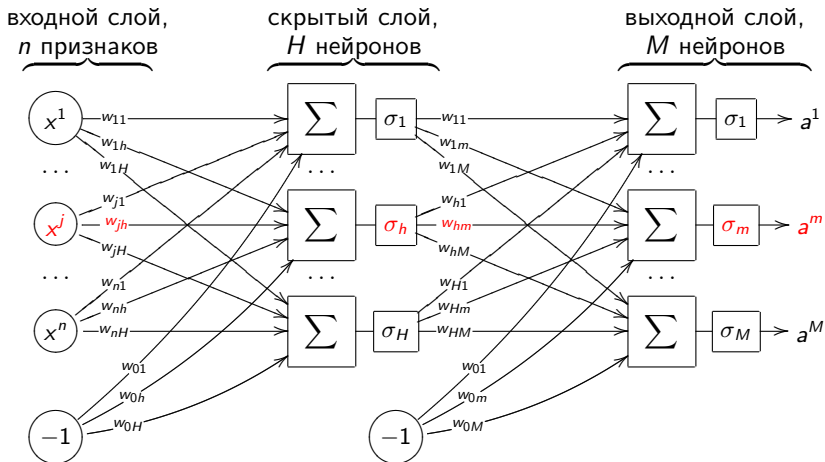
- С помощью линейных операций и одной нелинейной функции активации φ можно вычислять любую непрерывную функцию с любой желаемой точностью.
- Однако эти теоремы ничего не говорят о числе слоёв нейронной сети и о числе нейронов в каждом слое.

Практические рекомендации:

- Двух-трёх слоёв обычно достаточно.
- Можно достраивать нейроны в произвольных местах сети по необходимости, вообще не заботясь о числе слоёв.

Многослойная нейронная сеть

Пусть для общности $Y = \mathbb{R}^M$, для простоты слоёв только два.



Напоминание: Алгоритм SG (Stochastic Gradient)

Задача минимизации суммарных потерь:

$$Q(w) := \sum_{i=1}^{\ell} \mathcal{L}(w, x_i, y_i) \rightarrow \min_w.$$

Вход: выборка X^ℓ ; темп обучения η ; параметр λ ;

Выход: веса $w \equiv (w_{jh}, w_{hm}) \in \mathbb{R}^{H(n+M+1)+M}$;

- 1: инициализировать веса w и текущую оценку $Q(w)$;
- 2: **повторять**
- 3: выбрать объект x_i из X^ℓ (например, случайно);
- 4: вычислить потерю $\mathcal{L}_i := \mathcal{L}(w, x_i, y_i)$;
- 5: градиентный шаг: $w := w - \eta \nabla \mathcal{L}(w, x_i, y_i)$;
- 6: оценить значение функционала: $Q := (1 - \lambda)Q + \lambda \mathcal{L}_i$;
- 7: **пока** значение Q и/или веса w не стабилизируются;

Задача дифференцирования суперпозиции функций

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

$$a^m(x_i) = \sigma_m \left(\sum_{h=0}^H w_{hm} u^h(x_i) \right); \quad u^h(x_i) = \sigma_h \left(\sum_{j=0}^J w_{jh} f_j(x_i) \right).$$

Пусть для конкретности $\mathcal{L}_i(w)$ — 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

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

Быстрое вычисление градиента

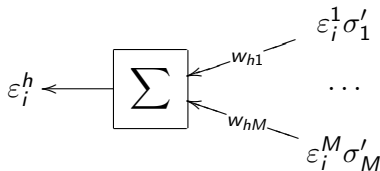
Промежуточная задача: частные производные

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

— это ошибка на выходном слое;

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

— назовём это *ошибкой на выходном слое*. Похоже, что 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 :

$$\frac{\partial \mathcal{L}_i(w)}{\partial w_{hm}} = \frac{\partial \mathcal{L}_i(w)}{\partial a^m} \frac{\partial a^m}{\partial w_{hm}} = \varepsilon_i^m \sigma'_m u^h(x_i), \quad m = 1..M, \quad h = 0..H;$$

$$\frac{\partial \mathcal{L}_i(w)}{\partial w_{jh}} = \frac{\partial \mathcal{L}_i(w)}{\partial u^h} \frac{\partial u^h}{\partial w_{jh}} = \varepsilon_i^h \sigma'_h f_j(x_i), \quad h = 1..H, \quad j = 0..n;$$

Алгоритм обратного распространения ошибки BackProp:

Вход: $X^\ell = (x_i, y_i)_{i=1}^\ell \subset \mathbb{R}^n \times \mathbb{R}^M$; параметры H, λ, η ;

Выход: синаптические веса w_{jh}, w_{hm} ;

1: ...

Алгоритм BackProp

- 1: инициализировать веса w_{jh} , w_{hm} ;
- 2: **повторять**
- 3: выбрать объект x_i из X^ℓ (например, случайно);
- 4: прямой ход:

$$u_i^h := \sigma_h \left(\sum_{j=0}^J w_{jh} x_i^j \right), \quad h = 1..H;$$

$$a_i^m := \sigma_m \left(\sum_{h=0}^H w_{hm} u_i^h \right), \quad \varepsilon_i^m := a_i^m - y_i^m, \quad m = 1..M;$$

$$\mathcal{L}_i := \sum_{m=1}^M (\varepsilon_i^m)^2;$$
- 5: обратный ход:

$$\varepsilon_i^h := \sum_{m=1}^M \varepsilon_i^m \sigma'_m w_{hm}, \quad h = 1..H;$$
- 6: градиентный шаг:

$$w_{hm} := w_{hm} - \eta \varepsilon_i^m \sigma'_m u_i^h, \quad h = 0..H, \quad m = 1..M;$$

$$w_{jh} := w_{jh} - \eta \varepsilon_i^h \sigma'_h x_i^j, \quad j = 0..n, \quad h = 1..H;$$
- 7: $Q := (1 - \lambda)Q + \lambda \mathcal{L}_i;$
- 8: **пока** Q не стабилизируется;

Алгоритм BackProp: преимущества и недостатки

Преимущества:

- эффективность: градиент вычисляется за время, сравнимое со временем вычисления самой сети;
- метод легко обобщается на любые σ , \mathcal{L} ;
- возможно динамическое (потокковое) обучение;
- на сверхбольших выборках не обязательно брать все x ;
- возможность распараллеливания;

Недостатки — все те же, свойственные SG:

- возможна медленная сходимость;
- застревание в локальных минимумах;
- проблема переобучения;
- подбор комплекса эвристик является искусством;

Стандартные эвристики для метода SG

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

- инициализация весов;
- порядок предъявления объектов;
- оптимизация величины градиентного шага;
- регуляризация (сокращение весов);

Кроме того, появляются новые проблемы:

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

Ускорение сходимости

1. Более тщательный подбор начального приближения.
Нейроны настраиваются как отдельные линейные алгоритмы
 - либо по случайной подвыборке $X' \subseteq X^\ell$;
 - либо по случайному подмножеству входов;
 - либо из различных случайных начальных приближений;тем самым обеспечивается *различность* нейронов.
2. Выбивание из локальных минимумов (jogging of weights).
3. Адаптивный градиентный шаг (метод скорейшего спуска).
4. Метод сопряжённых градиентов и chunking — разбиение суммы $Q(w) = \sum_{i=1}^{\ell} \mathcal{L}_i(w)$ на блоки (chunks).

Диагональный метод Левенберга-Марквардта

Метод Ньютона-Рафсона (второго порядка):

$$w := w - \eta \nabla \mathcal{L}(w) H(w),$$

где $H(w) = \left(\frac{\partial^2 Q(w)}{\partial w_{jh} \partial w_{j'h'}} \right)$ — гессиан, размера $(H(n+M+1)+M)^2$.

Эвристика. Считаем, что гессиан диагонален:

$$\eta_{jh} = \frac{\eta}{\frac{\partial^2 \mathcal{L}_i(w)}{\partial w_{jh}^2} + \mu},$$

η — темп обучения,

μ — параметр, предотвращающий обнуление знаменателя.

Шаг η вычисляется индивидуально для каждого веса.

Отношение η/μ есть темп обучения на ровных участках функционала $\mathcal{L}_i(w)$, где вторая производная обнуляется.

Динамическое наращивание сети

- 1 обучение при заведомо недостаточном числе нейронов N ;
- 2 после стабилизации $Q(w)$ — добавление нового нейрона и его инициализация путём обучения
 - либо по случайной подвыборке $X' \subseteq X^{\ell}$;
 - либо по объектам с наибольшими значениями потерь;
 - либо по случайному подмножеству входов;
 - либо из различных случайных начальных приближений;
- 3 снова итерации BackProp;

Эмпирический опыт: Общее время обучения обычно лишь в 1.5–2 раза больше, чем если бы в сети сразу было нужное количество нейронов. Полезная информация, накопленная сетью, не теряется при добавлении новых нейронов.

Прореживание сети (OBD — Optimal Brain Damage)

Пусть w — локальный минимум $Q(w)$, тогда $Q(w)$ можно аппроксимировать квадратичной формой:

$$Q(w + \delta) = Q(w) + \frac{1}{2} \delta^T H(w) \delta + o(\|\delta\|^2),$$

где $H(w) = \left(\frac{\partial^2 Q(w)}{\partial w_{jh} \partial w_{j'h'}} \right)$ — гессиан, размера $(H(n+M+1)+M)^2$.

Эвристика. Пусть гессиан $H(w)$ диагонален, тогда

$$\delta^T H(w) \delta = \sum_{j=0}^n \sum_{h=1}^H \delta_{jh}^2 \frac{\partial^2 Q(w)}{\partial w_{jh}^2} + \sum_{h=0}^H \sum_{m=0}^M \delta_{hm}^2 \frac{\partial^2 Q(w)}{\partial w_{hm}^2}.$$

Хотим обнулить вес: $w_{jh} + \delta_{jh} = 0$. Как изменится $Q(w)$?

Определение. *Значимость* (salience) веса w_{jh} — это изменение функционала $Q(w)$ при его обнулении: $S_{jh} = w_{jh}^2 \frac{\partial^2 Q(w)}{\partial w_{jh}^2}$.

Прореживание сети (OBD — Optimal Brain Damage)

- 1 В BackProp вычислять вторые производные $\frac{\partial^2 Q}{\partial w_{jh}^2}$, $\frac{\partial^2 Q}{\partial w_{hm}^2}$.
- 2 Если процесс минимизации $Q(w)$ пришёл в минимум, то
 - упорядочить все веса по убыванию S_{jh} ;
 - удалить N связей с наименьшей значимостью;
 - снова запустить BackProp.
- 3 Если $Q(w, X^\ell)$ или $Q(w, X^k)$ существенно ухудшился, то вернуть последние удалённые связи и выйти.

Отбор признаков с помощью OBD — аналогично.

Суммарная значимость признака: $S_j = \sum_{h=1}^H S_{jh}$.

Эмпирический опыт: результат постепенного прореживания обычно лучше, чем BackProp изначально прореженной сети.