Дисциплина

Основы машинного обучения и нейронные сети

Лекция 6

Искусственные нейронные сети

Линейные модели классификации и регрессии

Обучающая выборка: $X^\ell = (x_i, y_i)_{i=1}^\ell$, объекты $\mathbf{x}_i \in \mathbb{R}^d$, ответы y_i

Задача регрессии: $Y = \mathbb{R}$

$a(\mathbf{x}, \mathbf{w}) = \langle \mathbf{w}, \mathbf{x}_i \rangle$ — линейная модель регрессии

$$Q(\mathbf{w}, X^\ell) = \sum_{i=1}^{\ell} \big(\sigma(\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)\big)^2 \to \min_{\mathbf{w}}$$

Задача классификации с двумя классами: $Y = \{\pm 1\}$

$a(\mathbf{x}, \mathbf{w}) = \text{sign}\langle \mathbf{w}, \mathbf{x}_i \rangle$ — линейная модель классификации

$\mathcal{L}(M)$ — невозрастающая функция отступа, например,

$\mathcal{L}(M) = \ln(1+\mathrm{e}^{-M})$, $(1-\mathrm{M})_+$, e^{-M}, $\frac{1}{1+\mathrm{e}^{-M}}$ и др.

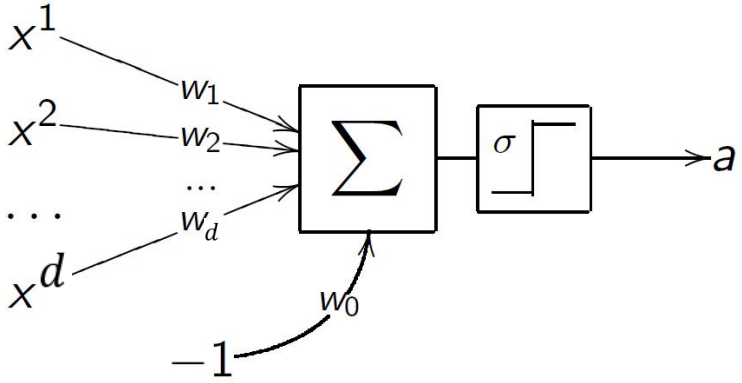
$$Q(\mathbf{w}, X^\ell) = \sum_{i=1}^{\ell} \mathcal{L}\underbrace{(\langle \mathbf{w}, \mathbf{x}_i \rangle y_i)}_{M_i(\mathbf{w})} \to \min_{\mathbf{w}}$$

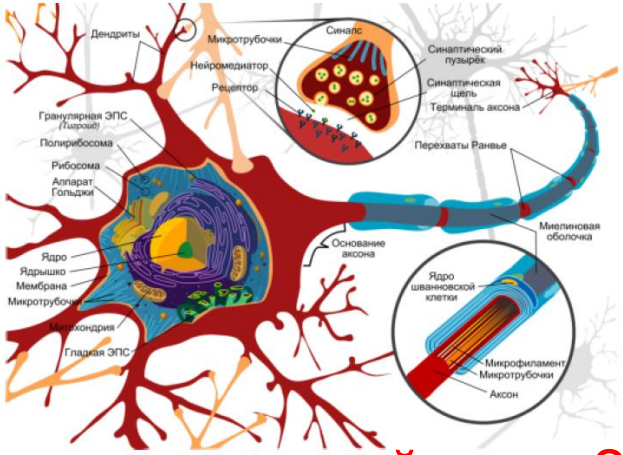
Линейная модель нейрона МакКаллока-Питтса (1943)

$f_i: X \to \mathbb{R}, j = 1, \ldots, d$ – числовые признаки;

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

w_j – синаптические веса признаков, $\sigma(z)$ - ф- активации, $x^j = f_j(\mathbf{x})$





Насколько богатый класс функций реализуется нейроном?
А сетью (суперпозицией) нейронов?

Однослойная сеть

$f_i: X \to \mathbb{R}, \quad j = 1, \ldots, d$ – числовые признаки
$x^j = f_j(\mathbf{x})$ – вектор описания объекта (признаковое описание)

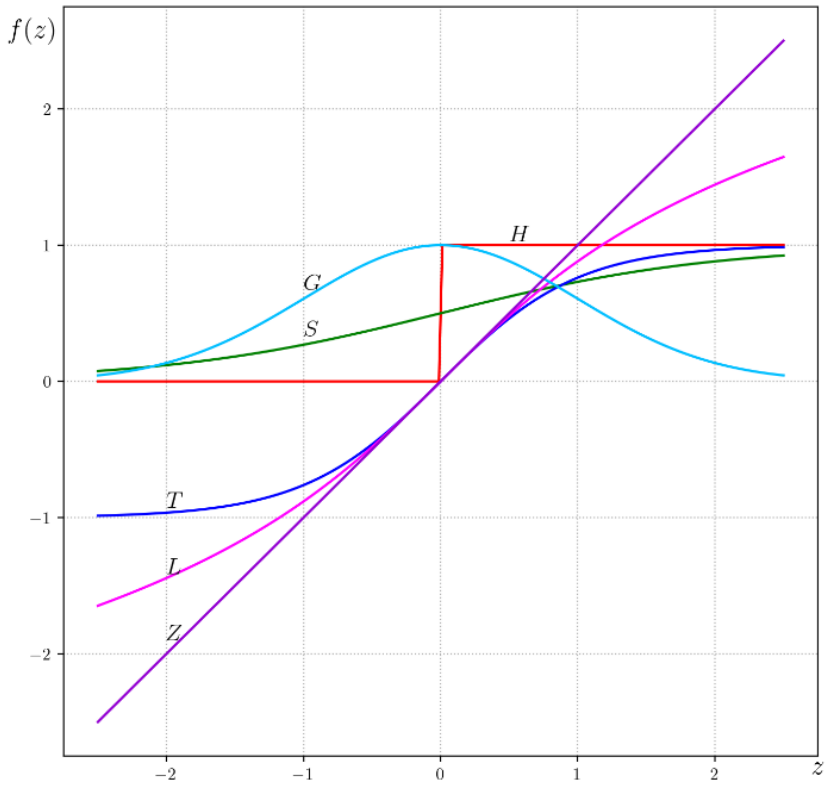
Однослойная сеть, или нейрон, определяется выражением

$$a(\mathbf{x}, \mathbf{w}) = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle) = \sigma\left(\sum_{j=1}^{d} w_j f_j(\mathbf{x}) + w_0\right),$$

$\mathbf{w}$ – синаптические веса признаков, $\sigma(z)$ - функция активации (непрерывная, монотонная и, желательно, дифференцируемая)

Функция активации

Функция активации преобразует значение суммы в выходное значение нейрона.



– пороговая ф-я Хевисайда (H),
– сигмоидная функция (S)
$a(\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})}$,
– гиперболический тангенс (T)
$th(x) = \frac{\exp(2x)-1}{\exp(2x)+1}$,
– логарифмическая ф-я (L),
– гауссовская функция (G),
– линейная функция (Z), в .т.ч. тожденственная
$a(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x}$.

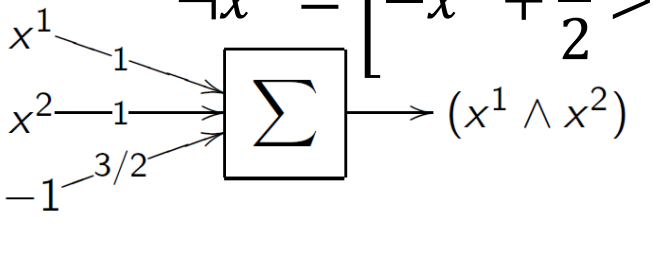
Нейронная реализация логических функций

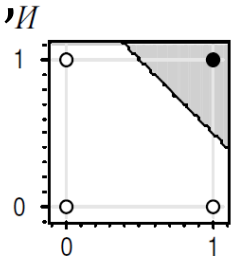
Функции И, ИЛИ, НЕ бинарных переменных (признаков) x^1, x^2:

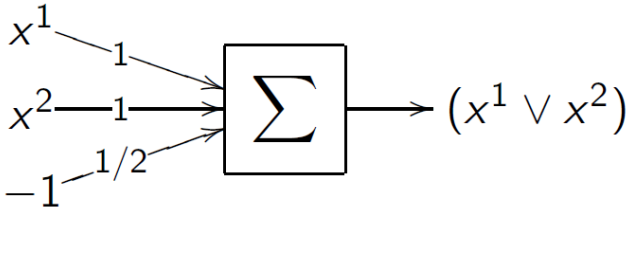
$$x^1 \wedge x^2 = \left[x^1 + x^2 - \frac{3}{2} > 0 \right];$$

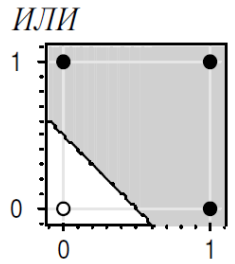
$$x^1 \vee x^2 = \left[x^1 + x^2 - \frac{1}{2} > 0 \right];$$

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









Логическая функция XOR (исключающее ИЛИ)

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

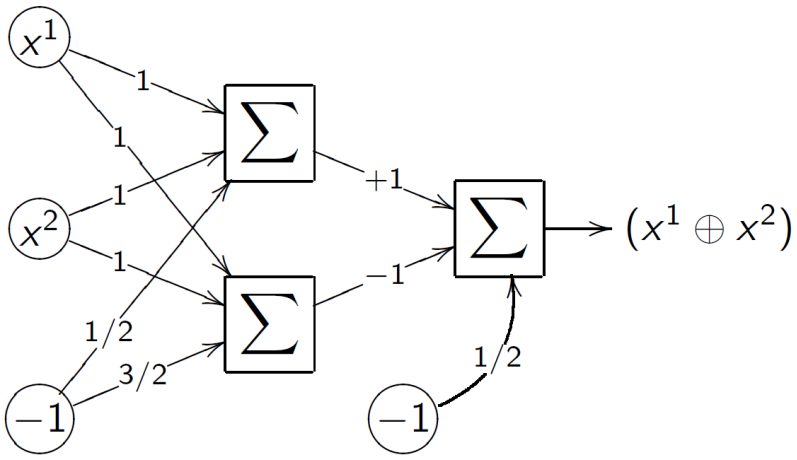
Два способа реализации:

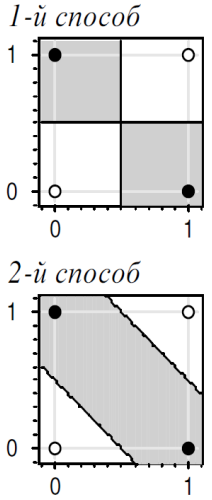
- Добавлением нелинейного признака:

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

- Сетью (двухслойной суперпозицией) функций И, ИЛИ, НЕ:

$$x^1 \oplus x^2 = [(x^1 \vee x^2) - (x^1 \wedge x^2) > 0].$$





Любую ли функцию можно представить нейросетью?

- Двухслойная сеть в $\{0,1\}^d$ позволяет реализовать произвольную булеву функцию (ДНФ).
- Линейный нейрон в $\mathbb{R}^d$ позволяет отделить полупространство гиперплоскостью. Двухслойная сеть в $\mathbb{R}^d$ позволяет отделить многогранную область, не обязательно выпуклую и связную.
- С помощью линейных операций и одной нелинейной *функции активации* σ можно приблизить любую непрерывную функцию с любой желаемой точностью (теорема Горбаня, 1998).

Практические выводы:

- Двух слоёв достаточно для аппроксимации функций.
- Глубокая сеть – это обучаемое преобразование признаков.

Нейронная сеть – универсальная модель

Способна аппроксимировать любые поверхности

Теорема Колмогорова (1957)

Каждая непрерывная функция $a(\mathbf{x})$, заданная на единичном кубе d -мерного пространства, представима в виде

$$a(\mathbf{x}) = \sum_{i=1}^{2d+1} \sigma_i \left(\sum_{j=1}^{d} f_{ij}(x_j) \right),$$

где $\mathbf{x} = (x_1, \dots, x_d)^T$ — вектор описания объекта,
функции $\sigma_i(\cdot)$ и $f_{ij}(\cdot)$ являются непрерывными,
а f_{ij} j не зависят от выбора a.

В теореме не указан конкретный вид функций функции $\sigma_i(\cdot)$ и $f_{ij}(\cdot)$. Проблема конструирования нейронной сети остается сложной задачей до сих пор.

Двухслойные сети – аппроксиматоры непрерывных функций

Функция $\sigma(z)$ — *сигмоида*, если $\lim\limits_{z\to-\infty}\sigma(z)=0$ и $\lim\limits_{z\to+\infty}\sigma(z)=1$.

Теорема Цыбенко (1989)

Если $\sigma(z)$ непрерывная сигмоида, то для любой непрерывной на $[0,1]^d$ функции $f(\mathbf{x})$ существуют такие значения параметров H, $\alpha_h \in \mathbb{R}, \mathbf{w}_h \in \mathbb{R}^d, w_0 \in \mathbb{R}$, что двухслойная сеть

$$a(\mathbf{x}) = \sum_{h=1}^{H} \alpha_h \sigma(\langle \mathbf{x}, \mathbf{w}_h \rangle - w_0)$$

равномерно приближает $f(\mathbf{x})$ с любой точностью ε:

$$|a(\mathbf{x}) - f(\mathbf{x})| < \varepsilon, \text{ для всех } \mathbf{x} \in [0,1]^d.$$

George Cybenko. Approximation by Superpositions of a Sigmoidal function. Mathematics of Control, Signals, and Systems. 1989.

Обобщение: полносвязная нейронная сеть с L слоями

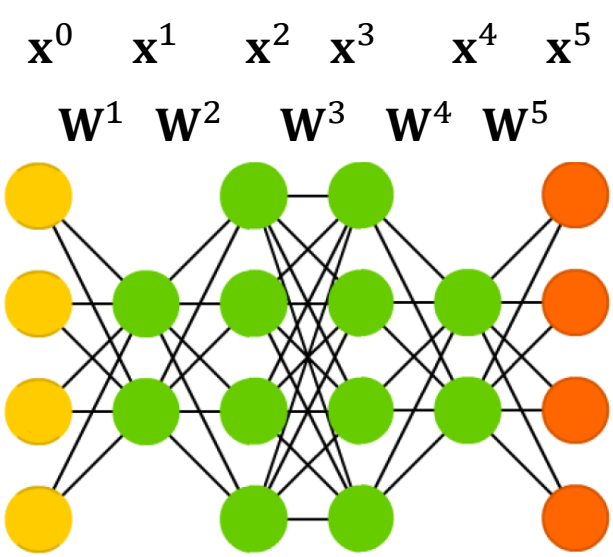
Архитектура сети: H_l — число нейронов в l-м слое, $l = 1, \dots, L$

$\mathbf{x}^0 = \mathbf{x} = (f_j(\mathbf{x}))_{j=0}^{d}$ — вектор признаков на входе сети, $H_0 = n$

$\mathbf{x}^l = (x_h^l)_{h=0}^{H_l}$ — вектор признаков на выходе l-го слоя, $x_0^l = -1$

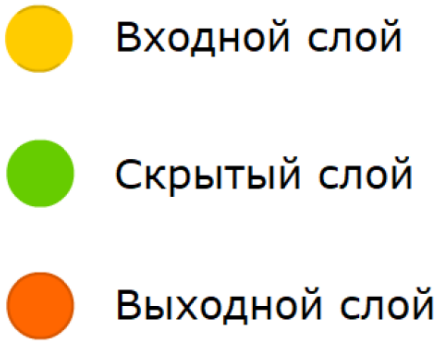
$\mathbf{x}^L = a(\mathbf{x}) = (a_m(\mathbf{x}))_{m=0}^{M}$ — выходной вектор сети, $H_L = M$

$\mathbf{W}^l = (w_{kh}^l)$ — матрица весов l-го слоя, размера $(H_{l-1} + 1) \times H_l$

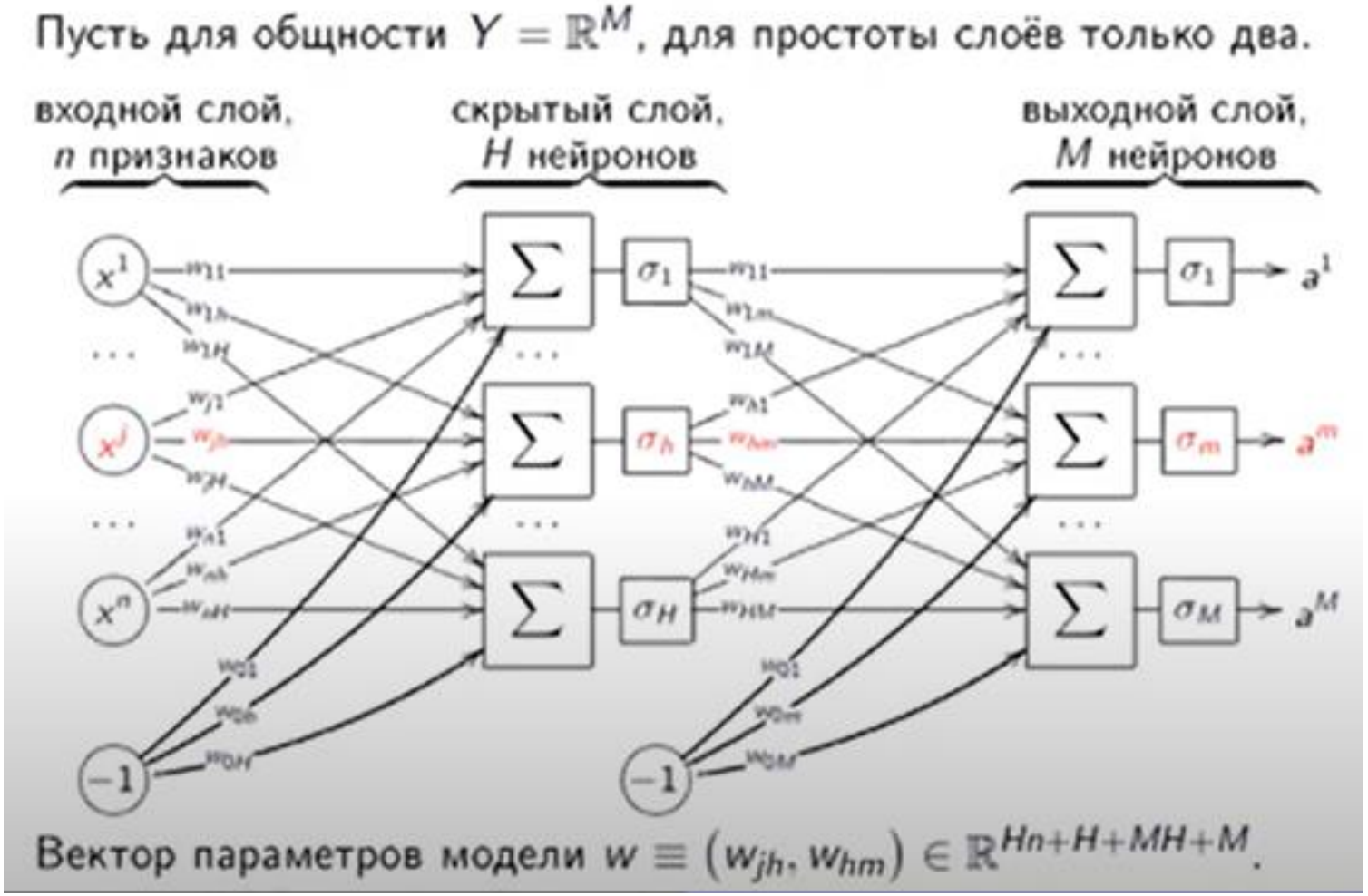


$L = 5, \quad d = 4, \quad \mathbf{W} = (\mathbf{W}^1, \dots, \mathbf{W}^5)$

$\mathbf{W}^1 \in \mathbb{R}^{H_1 d + H_1}, \dots, \mathbf{W}^5 \in \mathbb{R}^{H_4 H_5 + H_5}$



Двухслойная нейронная сеть с n-мерным входом и M-мерным выходом



Алгоритм SG (Stochastic Gradient)

Минимизация средних потерь на обучающей выборке:

$$Q(w) = \frac{1}{\ell}\sum_{i=1}^{\ell} \mathcal{L}_i(\mathbf{W}) \to \min_{\mathbf{W}}$$

Вход: выборка $(\mathbf{x}_i, y_i)_{i=1}^{\ell}$; темп обучения η; параметр λ;
Выход: вектор весов всех слоёв $\mathbf{W} = (\mathbf{W}^1, \ldots, \mathbf{W}^L)$;
инициализировать веса $\mathbf{W}$ и текущую оценку $Q(\mathbf{W})$;
повторять
    выбрать объект $\mathbf{x}_i$; из X^ℓ (например, случайно);
    вычислить функцию потерь $\mathcal{L}_i := \mathcal{L}_i(\mathbf{W})$;
    градиентный шаг: $\mathbf{W} := \mathbf{W} - \eta\nabla\mathcal{L}_i(\mathbf{W})$;
    оценить значение функционала: $Q := (1-\lambda)Q + \lambda\mathcal{L}_i$;
пока значение Q и/или веса и не стабилизируются;

Задача дифференцирования суперпозиции функций

l - номер слоя, h - номер нейрона в l-м слое, S_h^l - рез-т работы сумматора

Вычисление сети по входному вектору $\mathbf{x}$, рекуррентно по слоям:

$$x_h^l = \sigma_h^l\left(S_h^l\right), \qquad S_h^l = \sum_{k=0}^{H_{l-1}} w_{kh}^l \, x_k^{l-1}, \qquad h = 1, \ldots . H_l, \qquad l = 1, \ldots, L,$$

то же самое в матричной записи: $\mathbf{x}^l = \sigma^l\left(\mathbf{W}^l \mathbf{x}^{l-1}\right)$.

Функция потерь на объекте $\mathbf{x}_i$ (в общем виде и квадратичная):

$$\mathcal{L}_i(\mathbf{w}) = \sum_{m=1}^{M} \mathcal{L}(a_m(\mathbf{x}_i, \mathbf{w}), y_{im}) = \sum_{m=1}^{M} \frac{1}{2}(a_m(\mathbf{x}_i, \mathbf{w}), y_{im})^2$$

По формуле дифференцирования суперпозиции функций:

$$\frac{\partial \mathcal{L}_i(\mathbf{w})}{\partial w_{kh}^l} = \frac{\partial \mathcal{L}_i(\mathbf{w})}{\partial x_h^l} \frac{\partial x_h^l}{\partial w_{kh}^l}, \qquad k = 0, \ldots, H_{l-1}, \qquad h = 1, \ldots, H_l$$

Рекуррентное вычисление частных производных

Найдём сначала частные производные $\mathcal{L}_i(\mathbf{w})$ по $x_h^l \equiv a_h(\mathbf{x}_i, \mathbf{w})$:

$$\frac{\partial \mathcal{L}_i(\mathbf{w})}{\partial x_h^l} = \frac{\partial \mathcal{L}(x_h^l, y_{ih})}{\partial x_h^L} = a_h(\mathbf{x}_i, \mathbf{w}) - y_{ih} \equiv \varepsilon_{ih}^L;$$

для квадратичной функции потерь это ошибка выходного слоя.

Частные производные по x_h^l будем вычислять рекуррентно, по уровням справа налево, $l = L, \ldots, 2$:

$$\frac{\partial \mathcal{L}_i(\mathbf{w})}{\partial x_k^{l-1}} = \sum_{h=0}^{H_l} \frac{\partial \mathcal{L}_i(\mathbf{w})}{\partial x_h^l} \underbrace{\left(\sigma_h^l\right)'\left(S_h^l\right) w_{kh}^l}_{z_{ih}^l} = \sum_{h=0}^{H_l} \varepsilon_{ih}^l z_{ih}^l w_{kh}^l = \varepsilon_{ik}^{l-1}$$

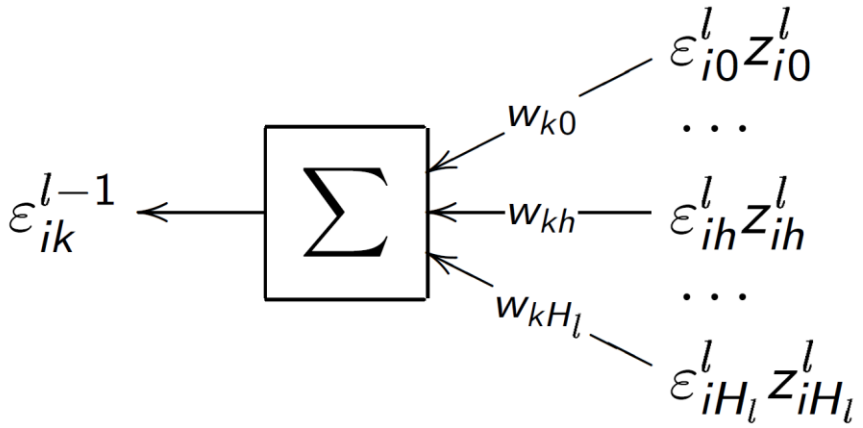
— формально назовём это ошибкой скрытого слоя.

$$\sum_{i=1}^{\ell} \mathcal{L}\underbrace{\left(\langle \mathbf{w}, \mathbf{x}_i \rangle y_i\right)}_{M_i(\mathbf{w})} \to \min_{\mathbf{w}}$$

Замечание: функция активации σ_h^l и её производная $\left(\sigma_h^l\right)'$ вычисляются в одной и той же точке $S_{ih}^l = \sum_{k=0}^{H_{l-1}} w_{kh}^l x_{ik}^{l-1}$

Быстрое вычисление градиента

Рекуррентная формула записана так, будто сеть запускается «задом наперёд», чтобы вычислять ε_{ik}^{l-1} по ε_{ih}^{l}:



Теперь, имея частные производные $\mathcal{L}_i(\mathbf{W})$ по всем x_h^l, легко найти градиент $\mathcal{L}_i(\mathbf{W})$ по вектору весов $\mathbf{W}$:

$$\frac{\partial \mathcal{L}_i(\mathbf{W})}{\partial w_{kh}^l} = \frac{\partial \mathcal{L}_i(\mathbf{W})}{\partial x_h^l} \frac{\partial x_h^l}{\partial w_{kh}^l} = \varepsilon_{ih}^l z_{ih}^l x_{ik}^{l-1}$$

Алгоритм обратного распространения ошибки
BackProp (1974, А.И.Галушкин, Пол Дж Вербос)

Вход: выборка $(\mathbf{x}_i, y_i)_{i=1}^{\ell}$, архитектура $(H_l)_{l=1}^{L}$, параметры η, λ;
Выход: матричный вектор весов всех слоёв $\mathbf{W} = (W^1, \ldots, W^L)$;
инициализировать веса $\mathbf{W}$;
повторять
выбрать объект $\mathbf{x}_i$ из X^{ℓ} (например, случайно);
прямой ход: для всех $l = 1, \ldots, L,\ h = 1, \ldots, H_l$

$$S_{ih}^l := \sum_{k=0}^{H_{l-1}} w_{kh}^l x_{ik}^{l-1}; \quad x_{ih}^l := \sigma_h^l\left(S_{ih}^l\right); \quad z_{ih}^l := \left(\sigma_h^l\right)'\left(S_{ih}^l\right);$$

$$\varepsilon_{hi}^L := \frac{\partial \mathcal{L}_i(\mathbf{W})}{\partial x_h^L}, \quad h = 1, \ldots, H_L; \quad Q := (1-\lambda)Q + \lambda \mathcal{L}_i(\mathbf{W});$$

обратный ход: для всех $l = L, \ldots, 2,\ k = 0, \ldots, H_{l-1}$

$$\varepsilon_{ik}^{l-1} := \sum_{h=0}^{H_l} \varepsilon_{ih}^l z_{ih}^l w_{kh}^l;$$

градиентный шаг: для всех $l = 1, \ldots, L,\ k = 0, \ldots, H_{l-1},\ h = 1, \ldots H_l$

$$w_{kh}^l := w_{kh}^l - \eta \varepsilon_{ih}^l z_{ih}^l x_{ik}^{l-1};$$

пока значения Q и/или веса $\mathbf{W}$ не стабилизируются;

Алгоритм BackProp: преимущества и недостатки

Преимущества:

- время вычисления градиента $O(\dim \mathbf{W})$ вместо $O(\dim^2 \mathbf{W})$
- обобщение на любые σ, $\mathcal{L}$ и любое число слоёв
- возможность динамического (потокового) обучения
- сублинейное обучение на сверхбольших выборках (когда части объектов $\mathbf{x}_i$ уже достаточно для обучения)
- возможно распараллеливание

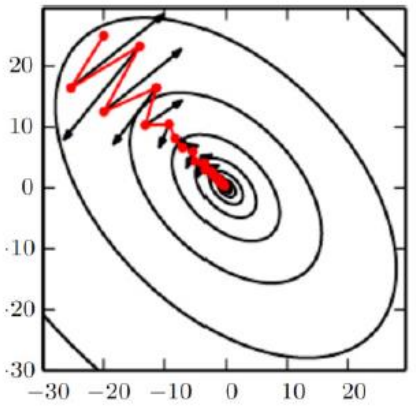
Недостатки — все те же, свойственные SG:

- медленная сходимость
- застревание в локальных экстремумах
- «паралич сети» из-за горизонтальных асимптот σ
- проблема переобучения
- подбор комплекса эвристик является искусством

Метод накопления инерции (momentum)

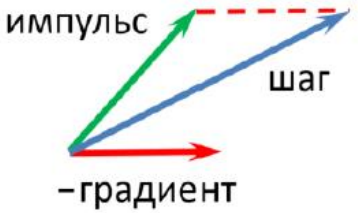
Momentum — экспоненциальное скользящее среднее градиента по $\approx \frac{1}{1-\gamma}$ последним итерациям [Б.Т.Поляк, 1964]:

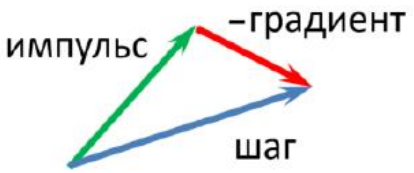
$$v := \gamma v + (1-\gamma)\mathcal{L}_i'(w)$$
$$w := w - \eta v$$



NAG (Nesterov's accelerated gradient) — стохастический градиент с инерцией [Ю.Е.Нестеров, 1983]:

$$v := \gamma v + (1-\gamma)\mathcal{L}_i'(w - \eta\gamma v)$$
$$w := w - \eta v$$





Адаптивные градиенты

RMSProp (running mean square) — выравнивание скоростей изменения весов скользящим средним по $\approx \frac{1}{1-\alpha}$ итерациям, ускоряет обучение по весам, которые пока мало изменялись:

$$G := \alpha G + (1-\alpha)\mathcal{L}_i'(\mathbf{W}) \odot \mathcal{L}_i'(\mathbf{W})$$

$$\mathbf{W} := \mathbf{W} - \eta \mathcal{L}_i'(\mathbf{W}) \oslash (\sqrt{G} + \varepsilon)$$

где $\odot$ и $\oslash$ — покоординатное умножение и деление векторов.

AdaDelta (adaptive learning rate) — двойная нормировка приращений весов, после которой можно брать $\eta = 1$:

$$G := \alpha G + (1-\alpha)\mathcal{L}_i'(\mathbf{W}) \odot \mathcal{L}_i'(\mathbf{W})$$

$$\delta := \mathcal{L}_i'(\mathbf{W}) \odot \frac{\sqrt{\Delta} + \varepsilon}{\sqrt{G} + \varepsilon}$$

$$\Delta := \alpha\Delta + (1-\alpha)\delta \odot \delta$$

$$\mathbf{W} := \mathbf{W} - \eta\delta$$

Комбинированные градиентные методы

Adam (adaptive momentum) = инерция + RMSProp:

$$v := \gamma v + (1-\gamma)\mathcal{L}_i'(\mathbf{W}) \qquad \hat{v} := v\left(1-\gamma^k\right)^{-1}$$

$$G := \alpha G + (1-\alpha)\mathcal{L}_i'(\mathbf{W}) \odot \mathcal{L}_i'(\mathbf{W}) \qquad \hat{G} := G\left(1-\alpha^k\right)^{-1}$$

$$\mathbf{W} := \mathbf{W} - \eta\hat{v} \oslash \left(\sqrt{\hat{G}} + \varepsilon\right)$$

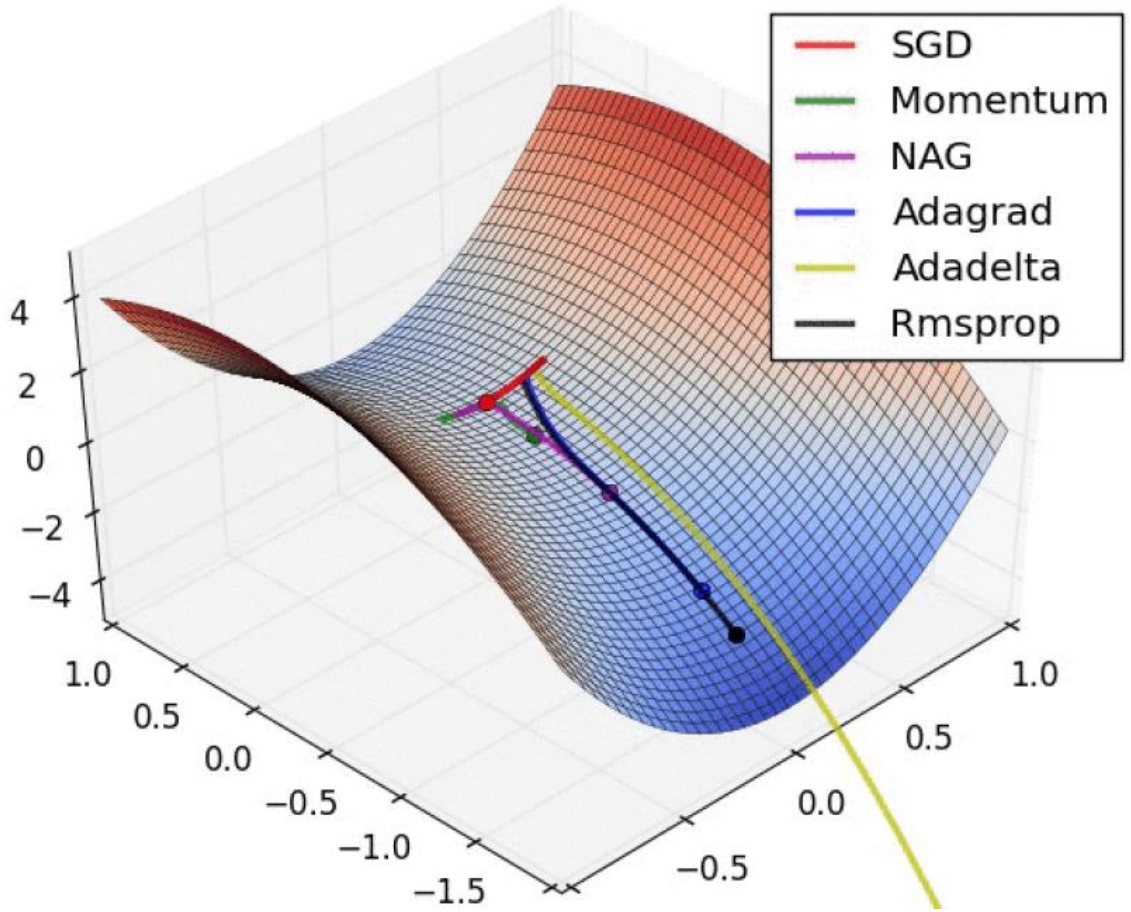
Калибровка $\hat{v}$, $\hat{G}$ увеличивает v, G на первых итерациях, где k — номер итерации; $\gamma = 0.9$, $\alpha = 0.999$, $\varepsilon = 10^{-8}$

Nadam (Nesterov-accelerated adaptive momentum):
те же формулы для v, $\hat{v}$, G, $\hat{G}$,

$$\mathbf{W} := \mathbf{W} - \eta\left(\gamma\hat{v} + \frac{1-\gamma}{1-\gamma^k}\mathcal{L}_i'(\mathbf{W})\right) \oslash \left(\sqrt{\hat{G}} + \varepsilon\right)$$

Timothy Dozat. Incorporating Nesterov Momentum into Adam. ICLR-2016.

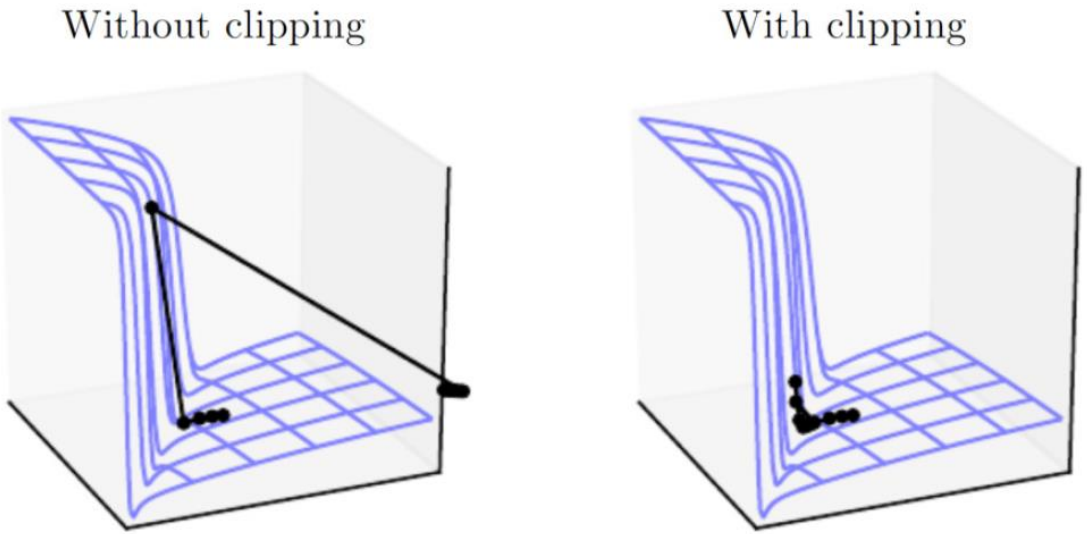
Сравнение сходимости методов



Alec Radford's animation:
https://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Проблема взрыва градиента и эвристика gradient clipping

Проблема взрыва градиента (gradient exploding)



Эвристика Gradient Clipping:
если $||g|| > \theta$ **то** $g := g\theta/||g||$

При грамотном подборе γ проблема взрыва градиента не возникает, и эвристика Gradient Clipping не нужна.

Метод случайных отключений нейронов (Dropout)

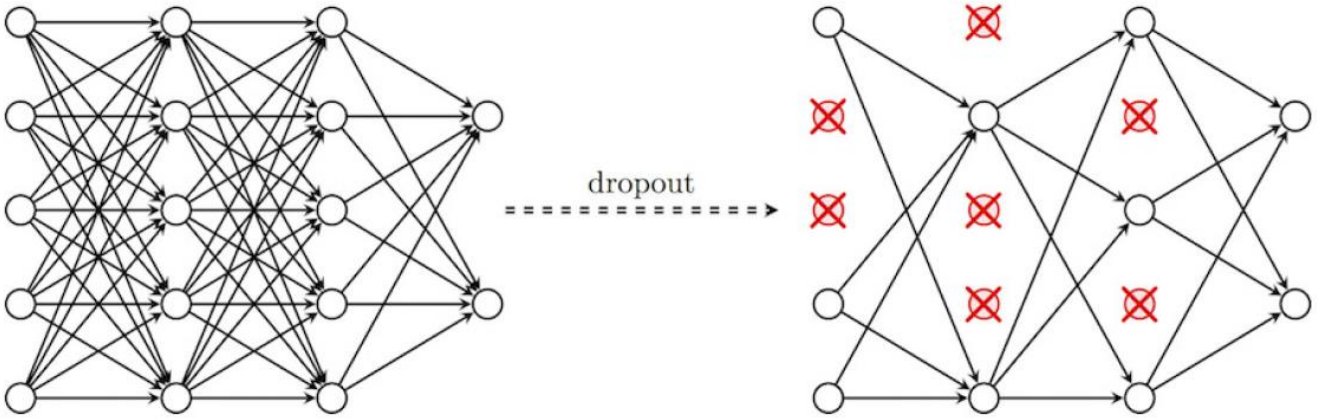
Этап обучения: делая градиентный шаг $\mathcal{L}_i(\mathbf{W}) \to \min\limits_{\mathbf{W}}$,

отключаем h-й нейрон l-го слоя с вероятностью p_l:

$$x_{hi}^l = \xi_h^l \sigma_h^l\left(\sum_k w_{kh}^l x_{ki}^{l-1}\right), \qquad \mathrm{P}\left(\xi_h^l = 0\right) = p_l$$

Этап применения: включаем все нейроны, но с поправкой:

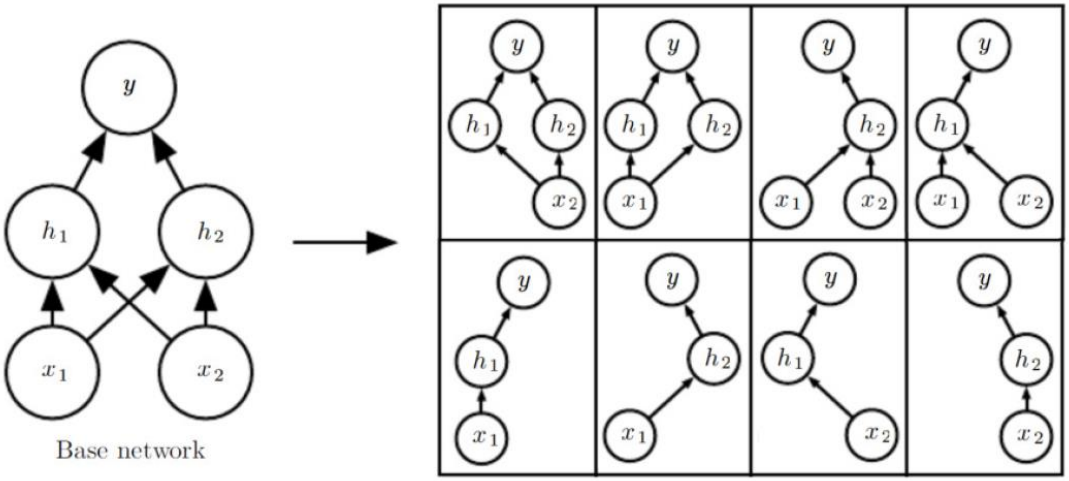
$$x_{hi}^l = (1 - p_l)\sigma_h^l\left(\sum_k w_{kh}^l x_{ki}^{l-1}\right)$$



N.Srivastava, G.Hinton, A.Krizhevsky, I.Sutskever, R.Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. 2014.

Интерпретации Dropout

1. Аппроксимируем простое голосование по 2^N сетям с общим набором из N весов, но с различной архитектурой связей
2. Регуляризация: из всех сетей выбираем более устойчивую к утрате $\mathrm{p}N$ нейронов, моделируя надёжность мозга
3. Сокращаем переобучение, заставляя разные части сети решать одну и ту же исходную задачу вместо того, чтобы подстраивать их под компенсацию ошибок друг друга



Обратный Dropout и L_2-регуляризация

На практике чаще используют не Dropout, а *Inverted Dropout*.

Этап обучения:

$$x_{hi}^l = \frac{1}{1-p_l}\xi_h^l \sigma_h^l\left(\sum_k w_{kh}^l x_{ki}^{l-1}\right), \qquad \mathrm{P}\left(\xi_h^l = 0\right) = p_\ell$$

Этап применения не требует ни модификаций, ни знания p_ℓ:

$$x_{hi}^l = \sigma_h^l\left(\sum_k w_{kh}^l x_{ki}^{l-1}\right)$$

L_2-регуляризация предотвращает рост параметров на обучении:

$$\mathcal{L}_i(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2 \to \min_{\mathbf{w}}$$

Градиентный шаг с Dropout и L_2-регуляризацией:

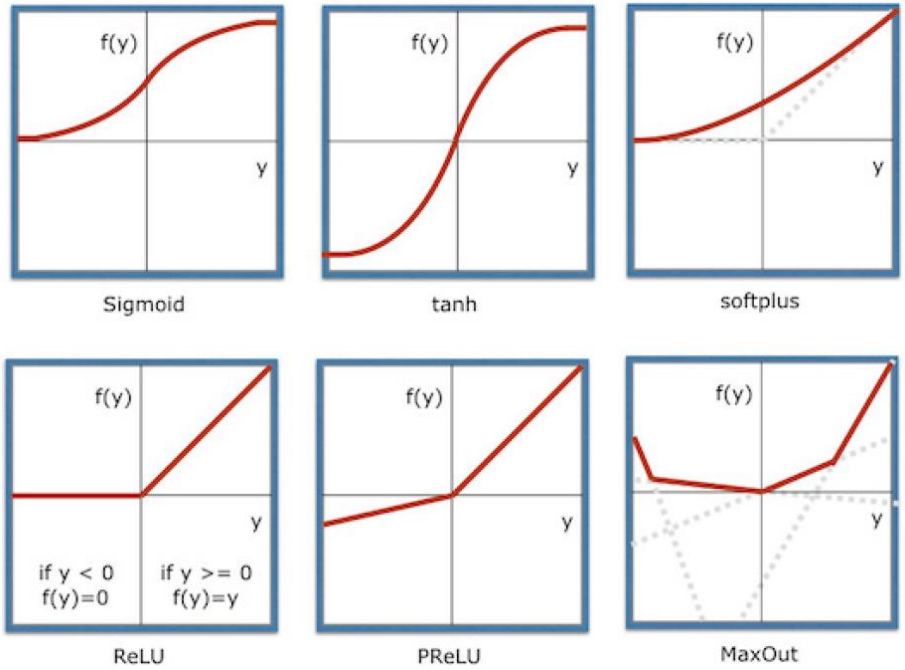
$$\mathbf{w} := \mathbf{w}(1-\eta\lambda) - \eta\frac{1}{1-p_l}\xi_h^l \mathcal{L}_i'(\mathbf{W})$$

Функции активации ReLU и PReLU (LeakyReLU)

Функции $\sigma(y) = \frac{1}{1+e^{-y}}$ и $th(y) = \frac{e^{y}-e^{-y}}{e^{y}+e^{-y}}$ могут приводить к затуханию градиентов или «параличу сети»

Функция положительной срезки (rectified linear unit)

$$ReLU(y) = \max\{0, y\}; \quad PReLU(y) = \max\{0, y\} + \alpha \min\{0, y\}$$



Пакетная нормализация данных (Batch Normalization)

$B = \{x_i\}$ пакеты (mini-batch) данных.

Усреднение градиентов $\mathcal{L}_i(\mathbf{W})$ по пакету ускоряет сходимость.

$B^l = \{x_i^l\}$ — векторы объектов x_i на выходе l-го слоя.

Batch Normalization:

1. Нормировать каждую h-ю компоненту вектора x_i^l по пакету:

$$\hat{x}_{hi}^l = \frac{x_{hi}^l - \mu_h}{\sqrt{\sigma_h^2 + \varepsilon}}; \quad \mu_h = \frac{1}{|B|}\sum_{x_i \in B} x_{hi}^l \; ; \; \sigma_h^2 = \frac{1}{|B|}\sum_{x_i \in B}(x_{hi}^l - \mu_h)^2.$$

2. Добавить линейный слой с настраиваемыми весами:

$$\tilde{x}_{hi}^l = \gamma_h^l \hat{x}_{hi}^l + \beta_h^l$$

3. Параметры γ_h^l и β_h^l настраиваются BackProp.

S.Ioffe, C.Szegedy (Google) Batch normalization: accelerating deep network training by reducing internal covariate shift. 2015.

Эвристики для начального приближения

1. Выравнивание дисперсий выходов в разных слоях:

$$w_{kh} := \text{uniform}\left(-\frac{1}{\sqrt{H_l}}, \frac{1}{\sqrt{H_l}}\right)$$

2. Выравнивание дисперсий градиентов в разных слоях:

$$w_{kh} := \text{uniform}\left(-\frac{6}{\sqrt{H_{l-1}+H_l}}, \frac{6}{\sqrt{H_{l-1}+H_l}}\right),$$

где H_{l-1}, H_l — число нейронов в предыдущем и текущем слое

3. Послойное обучение нейронов как линейных моделей:
   - либо по случайной подвыборке $X' \subseteq X^{\ell}$;
   - либо по случайному подмножеству входов;
   - либо из различных случайных начальных приближений;

тем самым обеспечивается различность нейронов.

4. Инициализация весами предобученной модели

5. Инициализация случайным ортогональным базисом

Прореживание сети (OBD - Optimal Brain Damage) (1/2)

Пусть $\mathbf{W}$ — локальный минимум $Q(\mathbf{W})$, тогда $Q(\mathbf{W})$ можно аппроксимировать квадратичной формой:

$$Q(\mathbf{W}+\delta) = Q(\mathbf{W}) + \frac{1}{2}\delta^T Q''(\mathbf{W})\delta + o(\|\delta\|^2),$$

где $Q''(\mathbf{W}) = \left(\frac{\partial^2 Q(\mathbf{W})}{\partial w_{kh}\partial w_{k'h'}}\right)$ — гессиан, размера $\dim^2(\mathbf{W})$.

Эвристика. Пусть гессиан $Q''(\mathbf{W})$ диагонален, тогда

$$\delta^T Q''(\mathbf{W})\delta = \sum_{l=1}^{L}\sum_{k=0}^{H_{l-1}}\sum_{h=1}^{H_l}\delta_{kh}^2\frac{\partial^2 Q(\mathbf{W})}{\partial w_{kh}^2}$$

Хотим обнулить вес: $\mathrm{w}_{kh} + \delta_{kh} = 0$. Как изменится $Q(\mathbf{W})$?

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

Yann LeCun, John Denker, Sara Solla. Optimal Brain Damage. 1989

Прореживание сети (OBD — Optimal Brain Damage) (2/2)

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

Отбор признаков с помощью OBD — аналогично.

Суммарная значимость признака: $S_j = \sum_{h=1}^{H_1} S_{jh}$.

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

Резюме

- Нейрон = линейная классификация или регрессия.
- Нейронная сеть = суперпозиция нейронов с нелинейной функцией активации. Теоретически двух-трёх слоёв достаточно для решения очень широкого класса задач.
- Глубокие нейросети автоматизируют выделение признаков из сложно структурированных данных (feature extraction)
- BackProp = быстрое дифференцирование суперпозиций. Позволяет обучать сети практически любой архитектуры.
- Некоторые меры по улучшению сходимости и качества:
  - адаптивный градиентный шаг
  - функции активации типа ReLU
  - регуляризация и DropOut
  - пакетная нормализация (batch normalization)
  - инициализация нейронов как отдельных алгоритмов

Соколов Е.А.
доц., руководитель департамента больших данных и информационного поиска ВШЭ

http://wiki.cs.hse.ru/Основы_машинного_обучения/2023
Материалы курса «Основы машинного обучения», ВШЭ, майнор ИАД (доп. профиль «Интеллектуальный анализ данных)

Воронцов К.В.
д.ф.-м.н., проф.,
ВМиК МГУ,
МФТИ, ВЦ РАН им. Дородницына, Яндекс, ВШЭ

http://www.MahineLearning.ru/wiki
«Машинное обучение (курс лекций, К.В.Воронцов)»