

# Искусственные нейронные сети: градиентные методы оптимизации

Воронцов Константин Вячеславович

vokov@forecsys.ru

<http://www.MachineLearning.ru/wiki?title=User:Vokov>

Этот курс доступен на странице вики-ресурса

<http://www.MachineLearning.ru/wiki>

«Машинное обучение (курс лекций, К.В.Воронцов)»

МФТИ • 15 сентября 2023

- 1 Многослойные нейронные сети**
  - Аппроксимационные возможности линейных моделей
  - Аппроксимационные возможности нейронных сетей
  - Многослойная нейронная сеть
- 2 Метод обратного распространения ошибок**
  - Метод стохастического градиента
  - Алгоритм BackProp
  - BackProp: преимущества и недостатки
- 3 Эвристики**
  - Эвристики для улучшения сходимости
  - Методы регуляризации
  - Функции активации и другие эвристики

## Напоминание: линейные модели классификации и регрессии

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

**Задача регрессии:**  $Y = \mathbb{R}$

$a(x, w) = \langle w, x_i \rangle$  — линейная модель регрессии

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

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

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

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

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

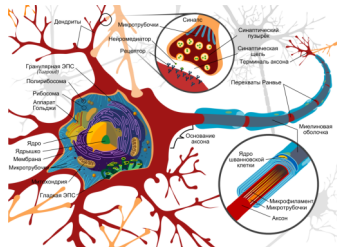
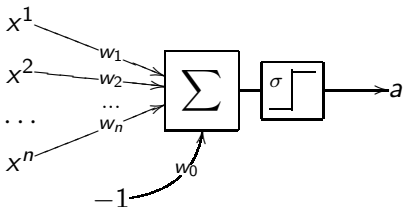
$$Q(w; X^\ell) = \sum_{i=1}^{\ell} \mathcal{L}(\underbrace{\langle w, x_i \rangle y_i}_{M_i(w)}) \rightarrow \min_w;$$

## Напоминание: линейная модель нейрона МакКаллока-Питтса

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

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

$w_j$  — веса признаков,  $\sigma(z)$  — функция активации,  $x^j \equiv f_j(x)$



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

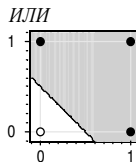
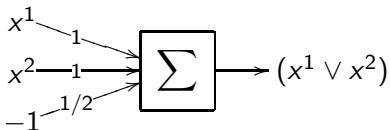
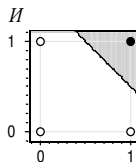
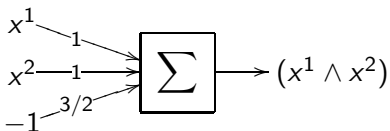
## Нейронная реализация логических функций

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

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

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

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



## Логическая функция XOR (исключающее ИЛИ)

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

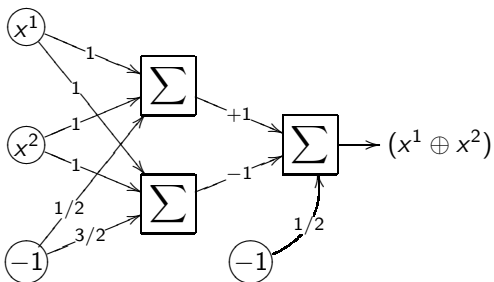
Два способа реализации:

- Добавлением нелинейного признака:

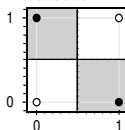
$$x^1 \oplus x^2 = [x^1 + x^2 - 2x^1x^2 - \frac{1}{2} > 0];$$

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

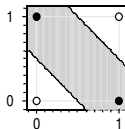
$$x^1 \oplus x^2 = [(x^1 \vee x^2) - (x^1 \wedge x^2) - \frac{1}{2} > 0].$$



1-й способ



2-й способ



## Любую ли функцию можно представить нейросетью?

- Двухслойная сеть в  $\{0, 1\}^n$  позволяет реализовать произвольную булеву функцию (ДНФ).
- Линейный нейрон в  $\mathbb{R}^n$  позволяет отделить полупространство гиперплоскостью.  
Двухслойная сеть в  $\mathbb{R}^n$  позволяет отделить многогранную область, не обязательно выпуклую и связную.
- С помощью линейных операций и одной нелинейной функции активации  $\sigma$  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.

## Двухслойные сети — аппроксиматоры непрерывных функций

Функция  $\sigma(z)$  — сигмоида, если  $\lim_{z \rightarrow -\infty} \sigma(z) = 0$  и  $\lim_{z \rightarrow +\infty} \sigma(z) = 1$ .

### Теорема Цыбенко (1989)

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

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

равномерно приближает  $f(x)$  с любой точностью  $\varepsilon$ :

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

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



Обобщение: полносвязная нейронная сеть с  $L$  слоями

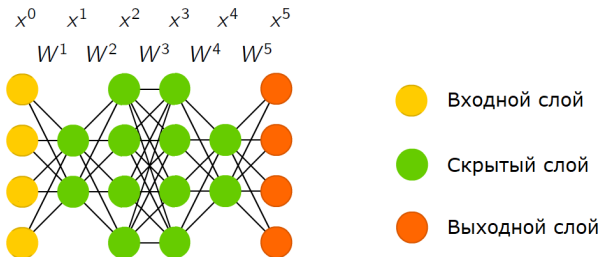
Архитектура сети:  $H_l$  — число нейронов в  $l$ -м слое,  $l = 1, \dots, L$

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

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

$x^L = a(x) = (a_m(x))_{m=1}^M$  — выходной вектор сети,  $H_L = M$

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



## Напоминание: алгоритм SG (Stochastic Gradient)

Минимизация средних потерь на обучающей выборке:

$$Q(w) = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}_i(w) \rightarrow \min_w.$$

**Вход:** выборка  $(x_i, y_i)_{i=1}^{\ell}$ ; темп обучения  $\eta$ ; параметр  $\lambda$ ;

**Выход:** вектор весов всех слоёв  $w = (W^1, \dots, W^L)$ ;

инициализировать веса  $w$  и текущую оценку  $Q(w)$ ;

**повторять**

выбрать объект  $x_i$  из  $X^{\ell}$  (например, случайно);

вычислить потерю  $\mathcal{L}_i := \mathcal{L}_i(w)$ ;

градиентный шаг:  $w := w - \eta \nabla \mathcal{L}_i(w)$ ;

оценить значение функционала:  $Q := (1 - \lambda)Q + \lambda \mathcal{L}_i$ ;

**пока** значение  $Q$  и/или веса  $w$  не стабилизируются;

## Задача дифференцирования суперпозиции функций

Вычисление сети по входному вектору  $x$ , рекуррентно по слоям:

$$x_h^l = \sigma_h^l(S_h^l), \quad S_h^l = \sum_{k=0}^{H_{l-1}} w_{kh}^l x_k^{l-1}, \quad h = 1, \dots, H_l, \quad l = 1, \dots, L,$$

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

Функция потерь на объекте  $x_i$  (в общем виде и квадратичная):

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

По формуле дифференцирования суперпозиции функций:

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

## Рекуррентное вычисление частных производных

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

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

для квадратичной функции потерь это *ошибка выходного слоя*.

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

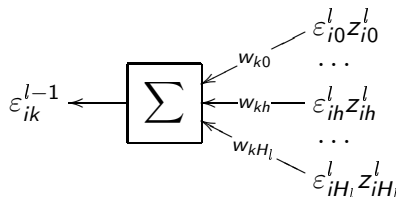
$$\frac{\partial \mathcal{L}_i(w)}{\partial x_k^{l-1}} = \sum_{h=0}^{H_l} \frac{\partial \mathcal{L}_i(w)}{\partial x_h^l} \underbrace{(\sigma_h^l)'(S_{ih}^l)}_{z_{ih}^l} w_{kh}^l = \sum_{h=0}^{H_l} \varepsilon_{ih}^l z_{ih}^l w_{kh}^l = \varepsilon_{ik}^{l-1}$$

— формально назовём это *ошибкой скрытого слоя*.

**Замечание:** функция активации  $\sigma_h^l$  и её производная  $(\sigma_h^l)'$  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  $\varepsilon_{ik}^{l-1}$  по  $\varepsilon_{ih}^l$ :



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

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

## Алгоритм обратного распространения ошибки BackProp

**Вход:** выборка  $(x_i, y_i)_{i=1}^{\ell}$ , архитектура  $(H_l)_{l=1}^L$ , параметры  $\eta, \lambda$ ;

**Выход:** вектор весов всех слоёв  $w = (W^1, \dots, W^L)$ ;

инициализировать веса  $w$ ;

**повторять**

выбрать объект  $x_i$  из  $X^{\ell}$  (например, случайно);

прямой ход: для всех  $l = 1..L, h = 1..H_l$

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

$$\varepsilon_{hi}^L := \frac{\partial \mathcal{L}_i(w)}{\partial x_h^L}, \quad h = 1..H_L;$$

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

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

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

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

**пока** значения  $Q$  и/или веса  $w$  не стабилизируются;

## Алгоритм BackProp: преимущества и недостатки

### Преимущества:

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

### Недостатки — все те же, свойственные SG:

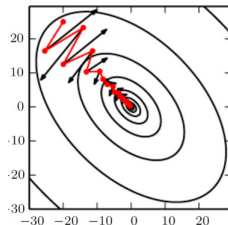
- медленная сходимость
- застревание в локальных экстремумах
- «паралич сети» из-за горизонтальных асимптот  $\sigma$
- проблема переобучения
- подбор комплекса эвристик является искусством

## Напоминание: метод накопления инерции (momentum)

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

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

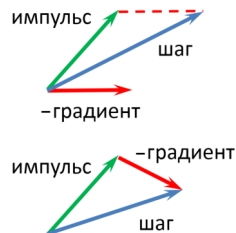
$$w := w - \eta v$$



**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:

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

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

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

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

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

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

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

$$w := w - \eta \delta$$

## Комбинированные градиентные методы

**Adam** (adaptive momentum) = инерция + RMSProp:

$$\begin{aligned}
 v &:= \gamma v + (1 - \gamma) \mathcal{L}'_i(w) & \hat{v} &:= v(1 - \gamma^k)^{-1} \\
 G &:= \alpha G + (1 - \alpha) \mathcal{L}'_i(w) \odot \mathcal{L}'_i(w) & \hat{G} &:= G(1 - \alpha^k)^{-1} \\
 w &:= w - \eta \hat{v} \odot (\sqrt{\hat{G}} + \varepsilon)
 \end{aligned}$$

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

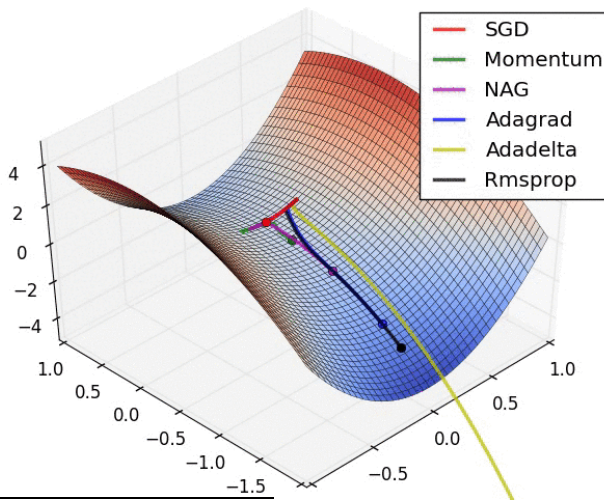
**Nadam** (Nesterov-accelerated adaptive momentum):  
 те же формулы для  $v$ ,  $\hat{v}$ ,  $G$ ,  $\hat{G}$ ,

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

---

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

## Сравнение сходимости методов

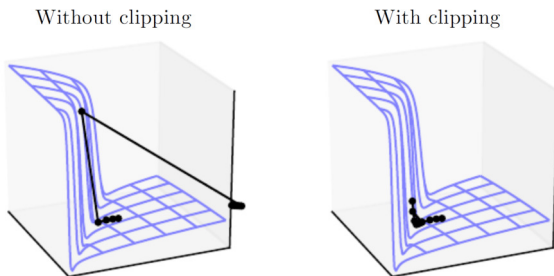


Alec Radford's animation:

<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  $\gamma$  проблема взрыва градиента не возникает, и эвристика Gradient Clipping не нужна.

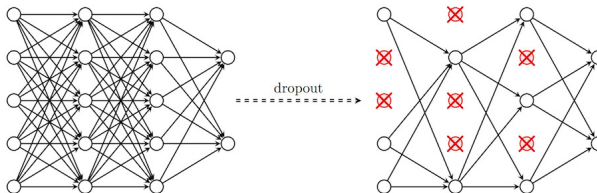
## Метод случайных отключений нейронов (Dropout)

Этап обучения: делая градиентный шаг  $\mathcal{L}_i(w) \rightarrow \min_w$ , отключаем  $h$ -ый нейрон  $l$ -го слоя с вероятностью  $p_l$ :

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

Этап применения: включаем все нейроны, но с поправкой:

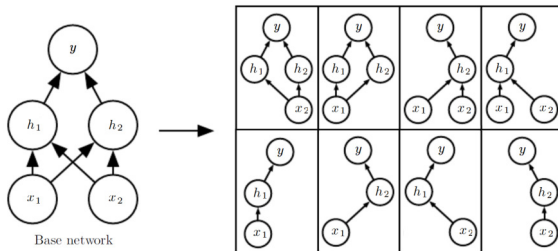
$$x_{hi}^l = (1 - p_l) \sigma_h^l \left( \sum_k w_{kh}^l x_{ki}^{l-1} \right)$$



*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  $pN$  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:

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

Этап применения не требует ни модификаций, ни знания  $p_\ell$ :

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

$L_2$ -регуляризация предотвращает рост параметров на обучении:

$$\mathcal{L}_i(w) + \frac{\lambda}{2} \|w\|^2 \rightarrow \min_w$$

Градиентный шаг с Dropout и  $L_2$ -регуляризацией:

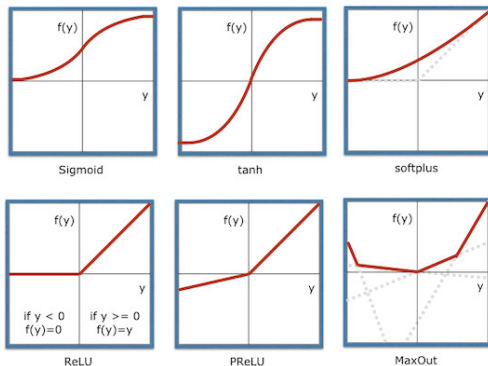
$$w := w(1 - \eta\lambda) - \eta \frac{1}{1-p_\ell} \xi_h^l \mathcal{L}'_i(w)$$

## Функции активации ReLU и PReLU (LeakyReLU)

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

Функция положительной срезки (rectified linear unit)

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





## Пакетная нормализация данных (Batch Normalization)

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

Усреднение градиентов  $\mathcal{L}_i(w)$  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:

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

2. Добавить линейный слой с настраиваемыми весами:

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

3. Параметры  $\gamma_h^l$  и  $\beta_h^l$  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)

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

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

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

**Эвристика.** Пусть гессиан  $Q''(w)$  диагонален, тогда

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

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

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

## Прореживание сети (OBD — Optimal Brain Damage)

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

**Отбор признаков** с помощью OBD — аналогично.

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

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

- 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