Искусственные нейронные сети

1. **Многослойные нейронные сети**
   - Проблема полноты
   - Вычислительные 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

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

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

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

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

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

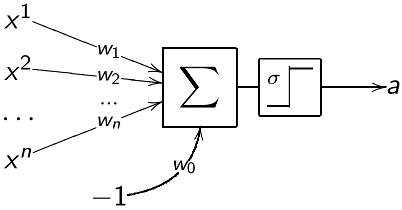
$$Q(w; X^\ell) = \sum_{i=1}^{\ell} \mathscr{L}\bigl(\underbrace{\langle w, x_i \rangle y_i}_{M_i(w)}\bigr) \to \min_w;$$

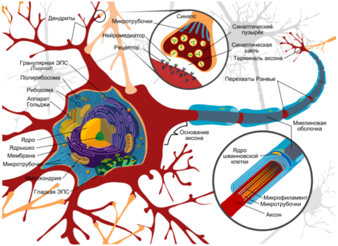
Напоминание: линейная модель нейрона МакКаллока-Питтса

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

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

w_j — веса признаков, $\sigma(z)$ — функция активации



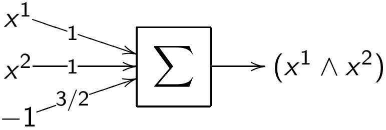


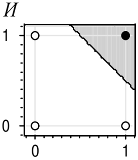
Насколько богатый класс функций реализуется нейроном?
А сетью (суперпозицией) нейронов?

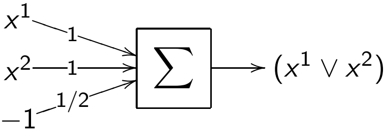
Нейронная реализация логических функций

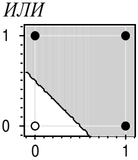
Функции И, ИЛИ, НЕ от бинарных переменных x^1 и x^2:

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





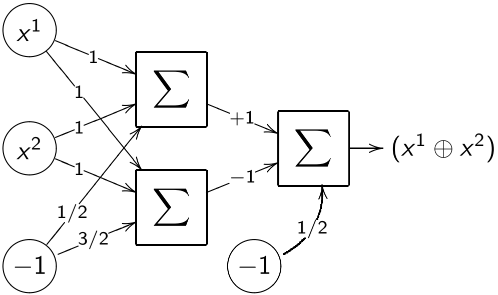


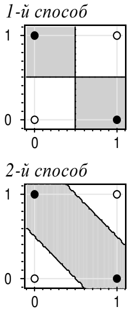


Логическая функция XOR (исключающее ИЛИ)

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

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





Любую ли функцию можно представить нейросетью?

- Двухслойная сеть в $\{0, 1\}^n$ позволяет реализовать произвольную булеву функцию (ДНФ).
- 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 теоретически достаточно.
- Глубокие сети — это встроенное обучение признаков.

Любую ли функцию можно представить нейросетью?

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 $[0,1]^n$ 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

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

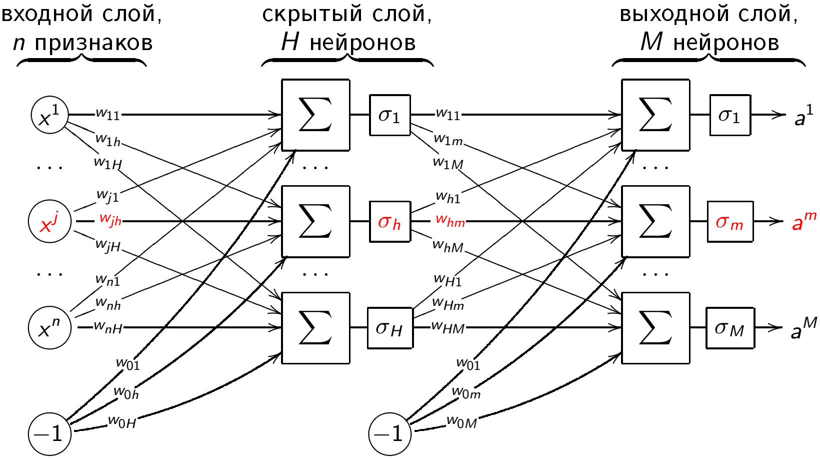
равномерно приближает $f(x)$ с любой точностью ε:

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

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

Двухслойная нейронная сеть с M-мерным выходом

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



Вектор параметров модели $w \equiv (w_{jh}, w_{hm}) \in \mathbb{R}^{Hn+H+MH+M}$.

Напоминание: алгоритм SG (Stochastic Gradient)

Минимизация средних потерь на обучающей выборке:

$$Q(w) := \frac{1}{\ell} \sum_{i=1}^{\ell} \mathscr{L}_i(w) \to \min_w .$$

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

Выход: вектор весов $w \equiv \bigl(w_{jh}, w_{hm}\bigr)$;

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

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

Без ограничения общности (только для примера) будем рассматривать среднеквадратичную функцию потерь:

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

Промежуточная задача: найти частные производные

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

Быстрое вычисление градиента

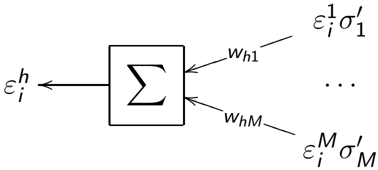
Промежуточная задача: частные производные

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

— это ошибка на выходном слое (для квадратичных потерь);

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

— назовём это *ошибкой на скрытом слое*. Похоже, что ε_i^h вычисляется по ε_i^m, если запустить сеть «задом наперёд»:



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

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

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

инициализировать веса w_{jh}, w_{hm};

повторять

  выбрать объект x_i из X^ℓ (например, случайно);

  прямой ход:

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

  $a_i^m := \sigma_m\bigl(\sum_{h=0}^{H} w_{hm} u_i^h\bigr)$, $m = 1..M$;

  $\varepsilon_i^m := \frac{\partial \mathscr{L}_i(w)}{\partial a^m}$, $m = 1..M$;

  $Q := (1-\lambda)Q + \lambda \mathscr{L}_i(w)$;

  обратный ход:

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

  градиентный шаг:

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

  $w_{jh} := w_{jh} - \eta \varepsilon_i^h \sigma'_h x_i^j$, $j = 0..n$, $h = 1..H$;

пока Q не стабилизируется;

Алгоритм BackProp: преимущества и недостатки

Преимущества:

- быстрое вычисление градиента
- обобщение на любые σ, $\mathscr{L}$ и любое число слоёв
- возможность динамического (потокового) обучения
- сублинейное обучение на сверхбольших выборках (когда части объектов x_i уже достаточно для обучения)
- возможно распараллеливание

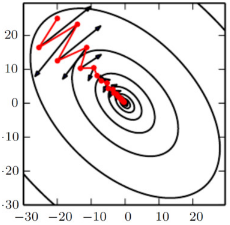
Недостатки — все те же, свойственные SG:

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

Напоминание: метод накопления инерции (momentum)

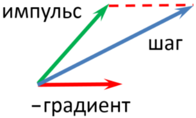
Momentum — экспоненциальное скользящее среднее градиента по $\approx \frac{1}{1-\gamma}$ последним итерациям [Б.Т.Поляк, 1964]:

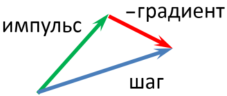
$$v := \gamma v + (1-\gamma)\mathscr{L}_i'(w)$$
$$w := w - \eta v$$



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

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





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:

$$G := \alpha G + (1-\alpha)\,\mathscr{L}_i'(w) \odot \mathscr{L}_i'(w)$$
$$w := w - \eta \mathscr{L}_i'(w) \oslash \left(\sqrt{G} + \varepsilon\right)$$

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

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

$$G := \alpha G + (1-\alpha)\,\mathscr{L}_i'(w) \odot \mathscr{L}_i'(w)$$
$$\delta := \mathscr{L}_i'(w) \odot \frac{\sqrt{\Delta}+\varepsilon}{\sqrt{G}+\varepsilon}$$
$$\Delta := \alpha \Delta + (1-\alpha)\,\delta \odot \delta$$
$$w := w - \eta\delta$$

Adam (adaptive momentum) = инерция + RMSProp:

$$v := \gamma v + (1-\gamma)\,\mathscr{L}_i'(w) \qquad \hat{v} := v(1-\gamma^k)^{-1}$$

$$G := \alpha G + (1-\alpha)\,\mathscr{L}_i'(w) \odot \mathscr{L}_i'(w) \qquad \hat{G} := G(1-\alpha^k)^{-1}$$

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

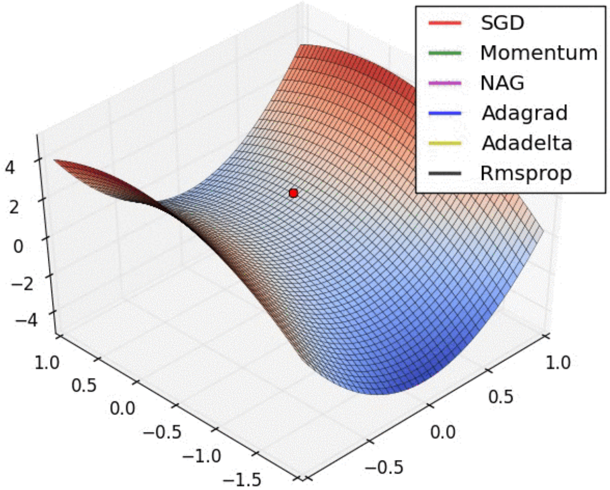
Калибровка $\hat{v}$, $\hat{G}$ увеличивает v, G на первых итерациях, где k — номер итерации; $\gamma = 0.9$, $\alpha = 0.999$, $\varepsilon = 10^{-8}$

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

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

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

Сравнение сходимости методов



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

Напоминание: диагональный метод Левенберга-Марквардта

Метод Ньютона-Рафсона (второго порядка):

$$w := w - \eta\bigl(\mathscr{L}_i''(w)\bigr)^{-1}\mathscr{L}_i'(w),$$

где $\bigl(\mathscr{L}_i''(w)\bigr) = \bigl(\frac{\partial^2 \mathscr{L}_i(w)}{\partial w_{jh}\partial w_{j'h'}}\bigr)$ — гессиан, размера $\bigl(H(n{+}M{+}1){+}M\bigr)^2$.

Эвристика. Считаем, что гессиан диагонален:

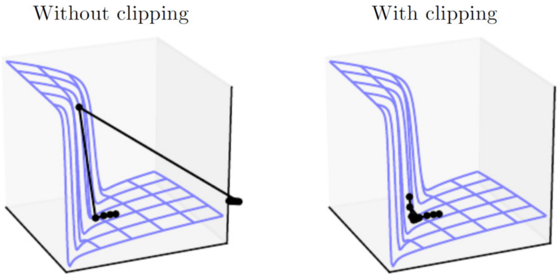
$$w_{jh} := w_{jh} - \eta\left(\frac{\partial^2 \mathscr{L}_i(w)}{\partial w_{jh}^2} + \mu\right)^{-1}\frac{\partial \mathscr{L}_i(w)}{\partial w_{jh}},$$

η — темп обучения (можно брать $\eta = 1$),
μ — параметр, предотвращающий обнуление знаменателя.

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

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.

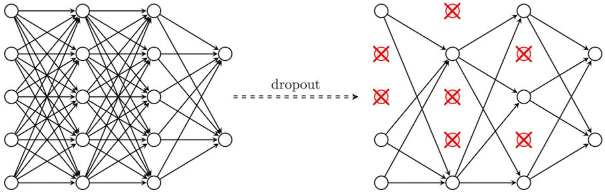
Метод случайных отключений нейронов (Dropout)

Этап обучения: делая градиентный шаг $\mathscr{L}_i(w) \to \min\limits_w$, отключаем h-ый нейрон ℓ-го слоя с вероятностью p_ℓ:

$$x_{ih}^{\ell+1} = \xi_h^\ell \sigma_h\Bigl(\sum_j w_{jh} x_{ij}^\ell\Bigr), \qquad \mathsf{P}(\xi_h^\ell = 0) = p_\ell$$

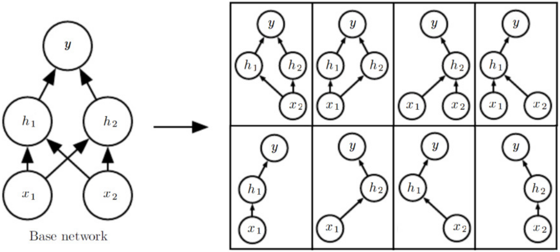
Этап применения: включаем все нейроны, но с поправкой:

$$x_{ih}^{\ell+1} = (1 - p_\ell)\sigma_h\Bigl(\sum_j w_{jh} x_{ij}^\ell\Bigr)$$



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:

$$x_{ih}^{\ell+1} = \frac{1}{1-p_\ell}\xi_h^\ell\,\sigma_h\Bigl(\sum_j w_{jh}x_{ij}^\ell\Bigr), \qquad \mathsf{P}(\xi_h^\ell = 0) = p_\ell$$

Этап применения не требует ни модификаций, ни знания p_ℓ:

$$x_{ih}^{\ell+1} = \sigma_h\Bigl(\sum_j w_{jh}x_{ij}^\ell\Bigr)$$

L_2-регуляризация предотвращает рост параметров на обучении:

$$\mathscr{L}_i(w) + \tfrac{\lambda}{2}\|w\|^2 \to \min_w$$

Градиентный шаг с Dropout и L_2-регуляризацией:

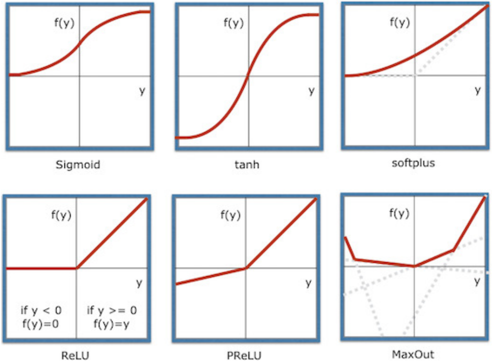
$$w := w(1-\eta\lambda) - \eta\,\frac{1}{1-p_\ell}\xi_h^\ell\,\mathscr{L}_i'(w)$$

Функции активации ReLU и PReLU (LeakyReLU)

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

Функция положительной срезки (rectified linear unit)

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



Пакетная нормализация данных (Batch Normalization)

$B = \{x_i\}$ — пакеты (mini-batch) данных.
Усреднение градиентов $\mathscr{L}_i(w)$ по пакету ускоряет сходимость.

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

Batch Normalization:

1. Нормировать каждую j-ю компоненту вектора u_i^ℓ по пакету:

$$\hat{u}_{ij}^\ell = \frac{u_{ij}^\ell - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}; \quad \mu_j = \frac{1}{|B|} \sum_{x_i \in B} u_{ij}^\ell; \quad \sigma_j^2 = \frac{1}{|B|} \sum_{x_i \in B} (u_{ij}^\ell - \mu_j)^2.$$

2. Добавить линейный слой с настраиваемыми весами:

$$\tilde{u}_{ij}^\ell = \gamma_j^\ell \hat{u}_{ij}^\ell + \beta_j^\ell$$

3. Параметры γ_j^ℓ и β_j^ℓ настраиваются BackProp.

Эвристики для начального приближения

1. Выравнивание дисперсий выходов в разных слоях:
$$w_j := \text{uniform}\left(-\frac{1}{\sqrt{h}}, \frac{1}{\sqrt{h}}\right)$$
2. Выравнивание дисперсий градиентов в разных слоях:
$$w_j := \text{uniform}\left(-\frac{6}{\sqrt{h+m}}, \frac{6}{\sqrt{h+m}}\right),$$
где h, m — 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:

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

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

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

$$\delta^{\mathsf{T}} Q''(w)\delta = \sum_{j=0}^{n}\sum_{h=1}^{H} \delta_{jh}^2 \frac{\partial^2 Q(w)}{\partial w_{jh}^2} + \sum_{h=0}^{H}\sum_{m=0}^{M} \delta_{hm}^2 \frac{\partial^2 Q(w)}{\partial w_{hm}^2}.$$

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: $S_j = \sum\limits_{h=1}^{H} S_{jh}$.

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