

Искусственные нейронные сети: градиентные методы оптимизации

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)»

МФТИ • 18 сентября 2021

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

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

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.

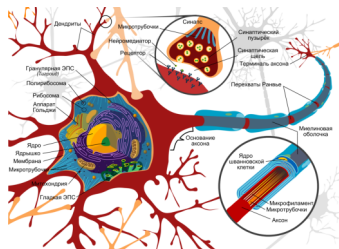
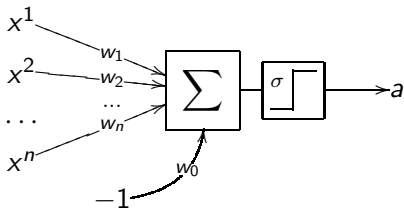
$$Q(w; X^\ell) = \sum_{i=1}^{\ell} \mathcal{L}(\underbrace{\langle w, x_i \rangle y_i}_{M_i(w)}) \rightarrow \min_w;$$

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



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

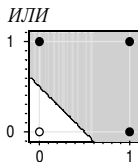
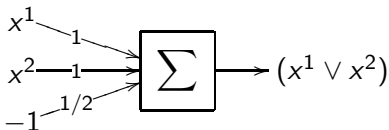
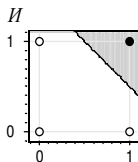
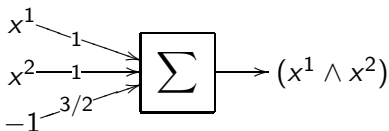
Нейронная реализация логических функций

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

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

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

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



Логическая функция XOR (исключающее ИЛИ)

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

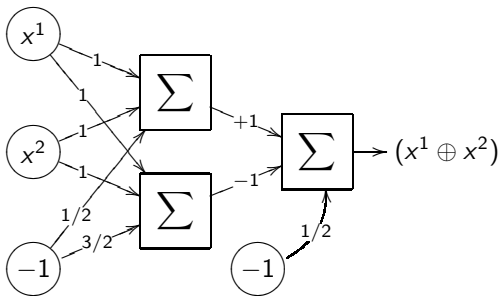
Два способа реализации:

- Добавлением нелинейного признака:

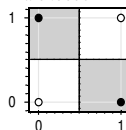
$$x^1 \oplus x^2 = [x^1 + x^2 - 2x^1x^2 - \frac{1}{2} > 0];$$

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

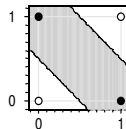
$$x^1 \oplus x^2 = [(x^1 \vee x^2) - (x^1 \wedge x^2) - \frac{1}{2} > 0].$$



1-й способ



2-й способ



Любую ли функцию можно представить нейросетью?

- Двухслойная сеть в $\{0, 1\}^n$ позволяет реализовать произвольную булеву функцию (ДНФ).
- 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 $\lim_{z \rightarrow -\infty} \sigma(z) = 0$ 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 ε :

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

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

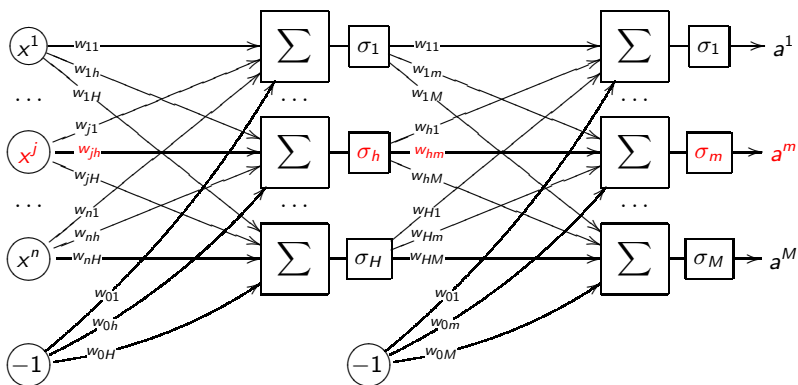
Двухслойная нейронная сеть с M -мерным выходом

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

входной слой,
 n признаков

скрытый слой,
 H нейронов

выходной слой,
 M нейронов



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:

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

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

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

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

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

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

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

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

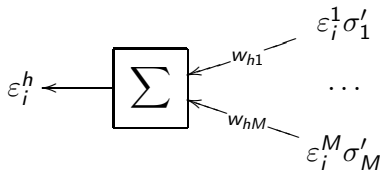
Промежуточная задача: частные производные

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

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

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

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



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:
$$u_i^h := \sigma_h\left(\sum_{j=0}^J w_{jh}x_i^j\right), \quad h = 1..H;$$
$$a_i^m := \sigma_m\left(\sum_{h=0}^H w_{hm}u_i^h\right), \quad m = 1..M;$$
$$\varepsilon_i^m := \frac{\partial \mathcal{L}_i(w)}{\partial a_i^m}, \quad m = 1..M;$$
- 5: $Q := (1 - \lambda)Q + \lambda \mathcal{L}_i(w)$;
- 6: обратный ход:
$$\varepsilon_i^h := \sum_{m=1}^M \varepsilon_i^m \sigma'_m w_{hm}, \quad h = 1..H;$$
- 7: градиентный шаг:
$$w_{hm} := w_{hm} - \eta \varepsilon_i^m \sigma'_m u_i^h, \quad h = 0..H, \quad m = 1..M;$$
$$w_{jh} := w_{jh} - \eta \varepsilon_i^h \sigma'_h x_i^j, \quad j = 0..n, \quad h = 1..H;$$
- 8: **пока** Q не стабилизируется;

Алгоритм BackProp: преимущества и недостатки

Преимущества:

- быстрое вычисление градиента
- обобщение на любые σ , \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:

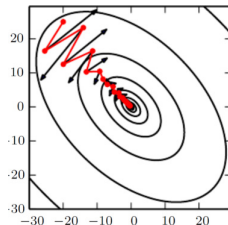
- медленная сходимость
- застревание в локальных экстремумах
- «паралич сети» из-за горизонтальных асимптот σ
- проблема переобучения
- подбор комплекса эвристик является искусством

Напоминание: метод накопления инерции (momentum)

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

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

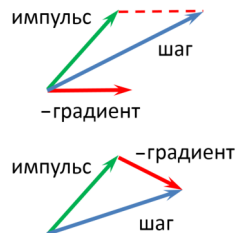
$$w := w - \eta v$$



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:

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

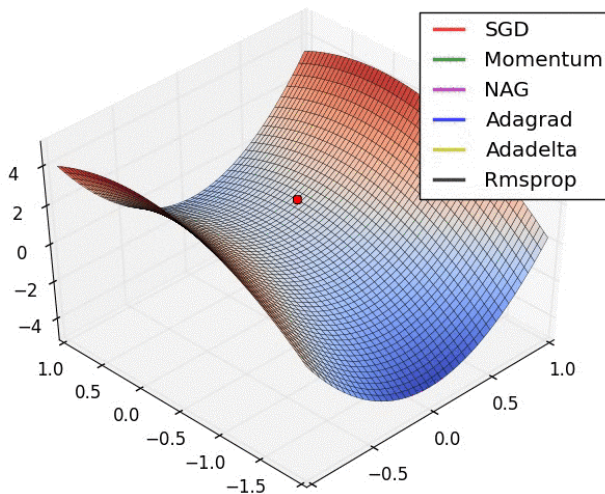
Калибровка \hat{v} , \hat{G} увеличивает v , G на первых итерациях, где k — номер итерации; $\gamma = 0.9$, $\alpha = 0.999$, $\varepsilon = 10^{-8}$

Nadam (Nesterov-accelerated adaptive momentum):

те же формулы для v , \hat{v} , G , \hat{G} ,

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

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):

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

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

Эвристика. Считаем, что гессиан диагонален:

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

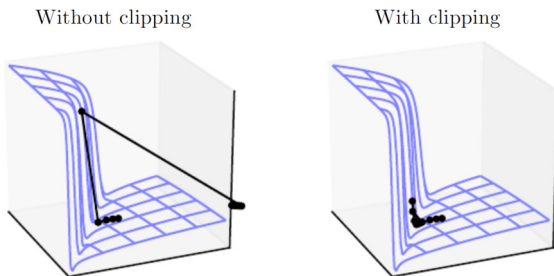
η — темп обучения (можно брать $\eta = 1$),

μ — параметр, предотвращающий обнуление знаменателя.

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

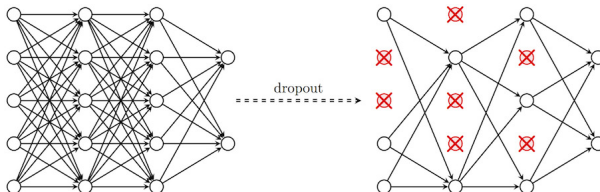
Метод случайных отключений нейронов (Dropout)

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

$$x_{ih}^{\ell+1} = \xi_h^\ell \sigma_h \left(\sum_j w_{jh} x_{ij}^\ell \right), \quad P(\xi_h^\ell = 0) = p_\ell$$

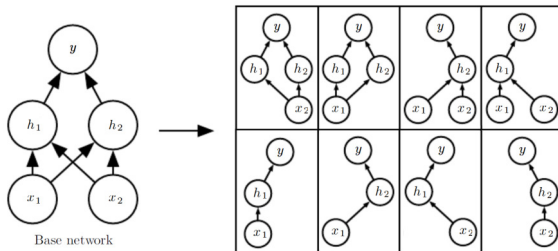
Этап применения: включаем все нейроны, но с поправкой:

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



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:

$$x_{ih}^{\ell+1} = \frac{1}{1-p_\ell} \xi_h^\ell \sigma_h \left(\sum_j w_{jh} x_{ij}^\ell \right), \quad P(\xi_h^\ell = 0) = p_\ell$$

Этап применения не требует ни модификаций, ни знания p_ℓ :

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

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:

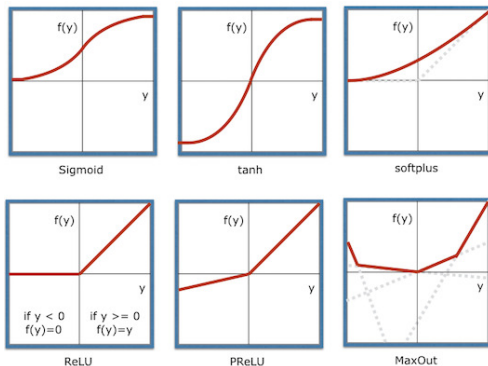
$$w := w(1 - \eta\lambda) - \eta \frac{1}{1-p_\ell} \xi_h^\ell \mathcal{L}'_i(w)$$

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.

$B^\ell = \{u_i^\ell\}$ — 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:

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

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

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

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

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 $\frac{\partial^2 Q}{\partial w_{jh}^2}$, $\frac{\partial^2 Q}{\partial w_{nm}^2}$.
- 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.

- 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