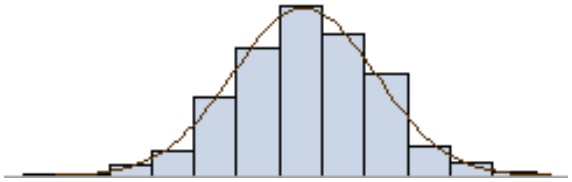
Лекция 7:
Обобщенные линейные модели, нелинейные модели, нейронные сети

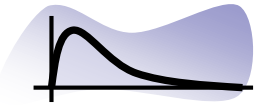
Основные предположения линейной регрессии

- Независимость наблюдений (и ошибок)
- Нормальное распределение ошибки с константной дисперсией

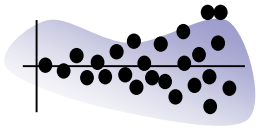
$\varepsilon \sim iid\ N(0,\sigma^2)$



- Часто возникающие «особенности»:
  - Несимметричные распределения отклика



  - Гетероскедастичность



  - Ограниченная область определения отклика

- Что делать?
  - Явно преобразовывать отклик: $E(g(Y)\mid X)$
  - Использовать функцию связи: $g(E(Y\mid X))$

Преобразование отклика и логнормальная регрессия

- Если логнормальное распределение отклика Y, тогда log(Y) – нормальное

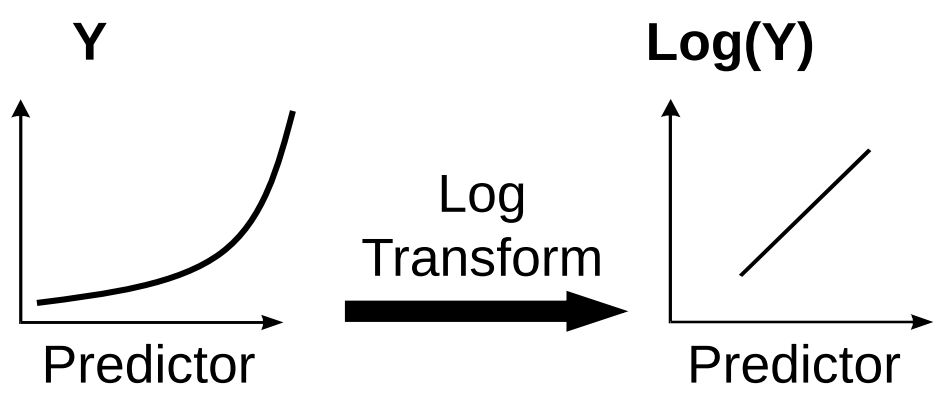
$$\log(Y) \sim N(\mu, \sigma^2)$$

$$E[Y] = \exp\left(\mu + \frac{\sigma^2}{2}\right)$$

$$Var[Y] = \left(e^{\sigma^2} - 1\right)(E[Y])^2$$

- Строим модель для преобразованного отклика:

MSE, но желательно на валидационном наборе



$$E[Log(Y)] = X\hat{\beta}$$

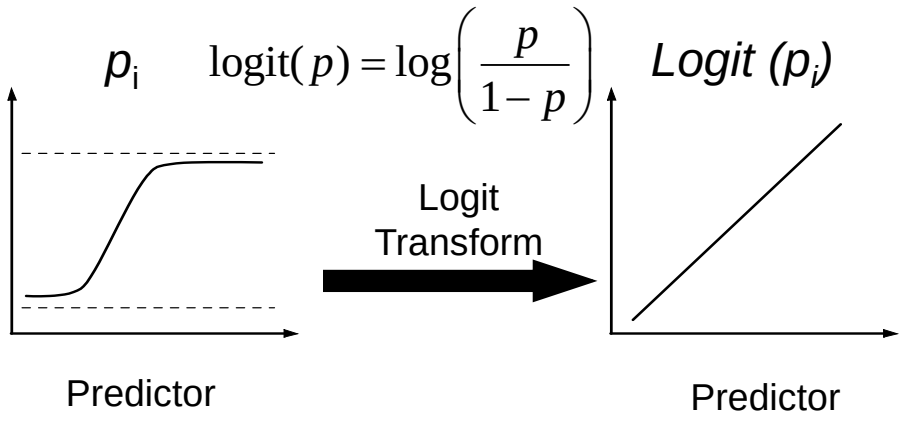
Но чему равно E[Y] ?

$$E[Y] \approx \exp\left(X\hat{\beta} + \frac{\sigma^2}{2}\right)$$

Обобщенные линейные модели

Функция связи → $g(E(y_i)) = \beta_0 + \beta_1 x_{1i} + \quad + \beta_k x_{ki} = X\beta$

- Распределение отклика наблюдений принадлежит экспоненциальному семейству. $f(y \mid \theta) = h(y)c(\theta)\exp\left(t(y)\cdot W(\theta)\right)$
- Дисперсия зависимой переменной Y – функция от среднего.
- Xβ моделирует функцию от E(y) (link function – функция связи)
- Распределение отклика наблюдений может подсказать какую функцию связи выбрать (дальше)
- Пример (лоистическая регрессия):



$$f(y \mid p) = p^y (1-p)^{1-y} =$$
$$= (1-p) I_y \exp\left(y \log(p/(1-p))\right)$$

$$I_y = \begin{cases} 1, y \in \{0,1\} \\ 0, \textit{иначе} \end{cases}$$

Типовые функции связи для обобщенных линейных моделей

Model	Response	Distribution	Mean	Variance	Canonical Link
Linear Regression	Continuous	Normal	μ	σ^2	identity μ
Logistic regression	Dichotomous	Binomial	π	$\pi(1-\pi)/n$	logit $\log[\pi/(1-\pi)]$
Poisson Regression	Count	Poisson	λ	λ	log $\log(\lambda)$
Gamma Regression	Continuous	Gamma	μ	μ^2/ν	*inverse $1/\mu$

*часто используется функция связи LOG

Параметры положения и разброса

Экспоненцияальное семество распределений: $f(y \mid \theta) = \exp\{\frac{y\theta - b(\theta)}{a(\phi)} + c(y, \phi)\}$

- Линейная регрессия $f(y \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{(y-\mu)^2}{(2\sigma^2)}}$,
- Логистическая регресси $f(y \mid p) = p^y \cdot (1-p)^{(1-y)}$
- Пуассоновская регресси $f(y \mid \lambda) = \frac{e^{-\lambda}\lambda^y}{y!}$
- Гамма регрессия $f(y \mid \nu, \mu) = \frac{1}{\Gamma(\nu) \cdot y} \cdot \left(\frac{y\nu}{\mu}\right)^{\nu} \cdot e^{-\left(\frac{y\nu}{\mu}\right)}$,

Регрессия	Параметр положения	Параметр разброса
Линейная	μ	σ
Логистическая	p	1
Пуассоновская	λ	1
Гамма	μ	ν

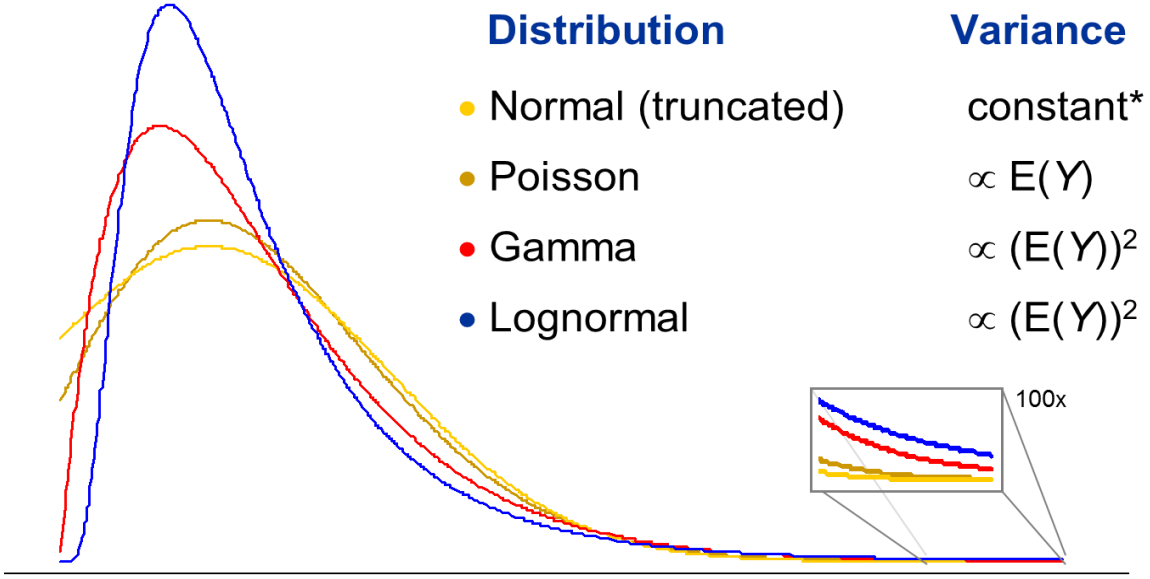
Оценка отклонения

Поиск параметров модели

- решается задача оптимизации
- max loglik с заданным распределением и функцией связи

Распределение	**Отклонение**
Normal	$Q(\mathbf{w}) = \sum (y - \mu(\mathbf{w}))^2$
Poisson	$Q(\mathbf{w}) = 2\sum \left[y \ln(y / \mu(\mathbf{w})) - (y - \mu(\mathbf{w})) \right]$
Gamma	$Q(\mathbf{w}) = 2\sum \left[-\ln(y / \mu(\mathbf{w})) + (y - \mu(\mathbf{w})) / \mu(\mathbf{w}) \right]$
Bernoulli	$Q(\mathbf{w}) = -2\sum \left[y \ln(\mu(\mathbf{w})) + (1 - y) \ln(1 - \mu(\mathbf{w})) \right]$

Выбор распределения для обощенной линейной модели



- В случае гетероскедастичности вместо линейной часто применяется гамма регрессия (с различными функциями связи)
- Гамма распределение:
  - Асиметричное распределение для **положительных** значений
  - Дисперсия пропорциональна квадрату среднего
  - Хвост «легче» чем у логнормального

Нелинейные зависимости

Истинная зависимость никогда (или почти никогда) не бывает линейной!

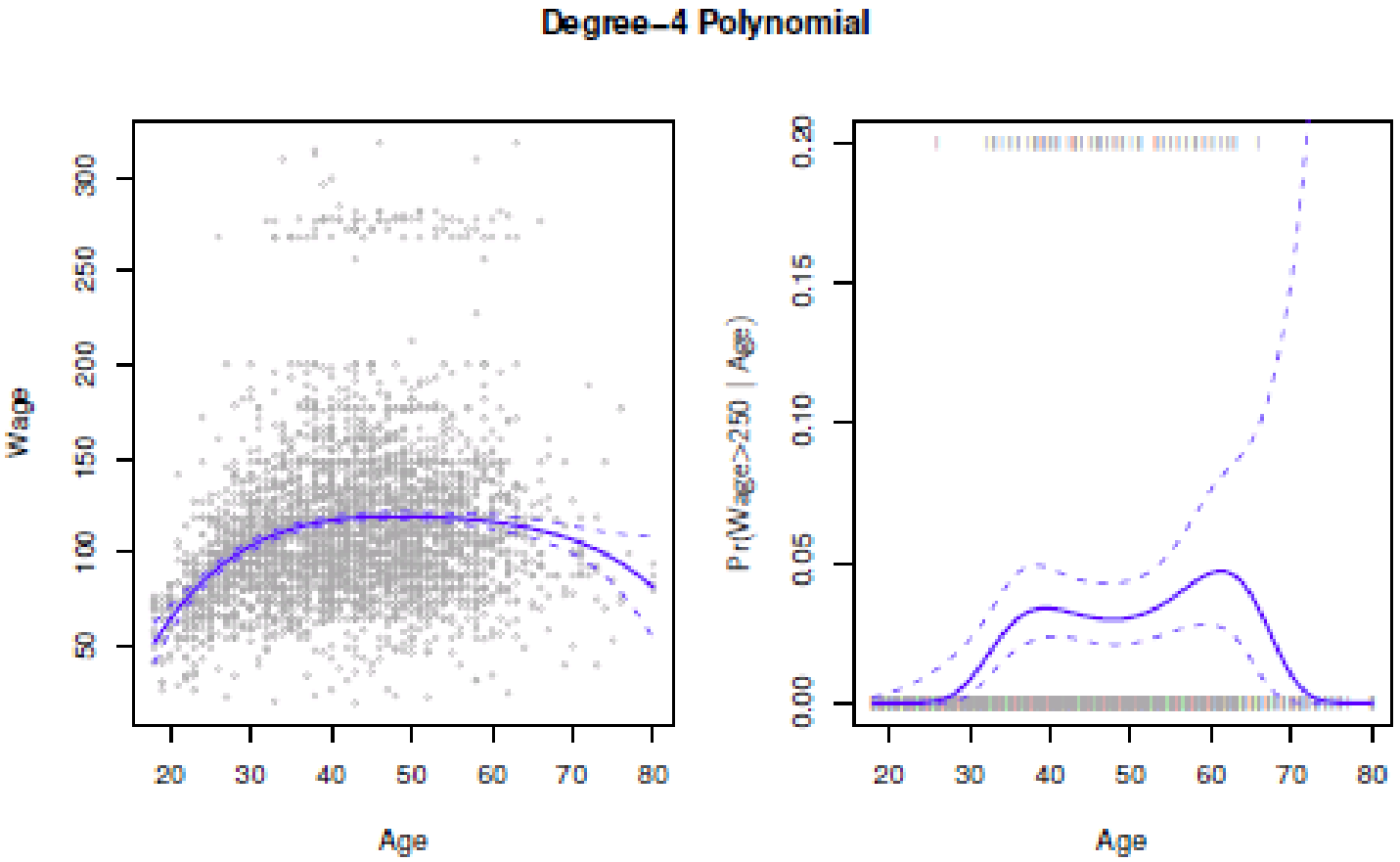
Но часто предположение о линейности достаточно хорошее.

Когда его нет, можно использовать:

- Полиномы
- Ступенчатые функции
- Сплайны
- Локальную регрессию
- Обобщенные аддитивные модели
- **Нейронные сети**
- **Деревья решений и их ансамбли**

Полиномиальная регрессия

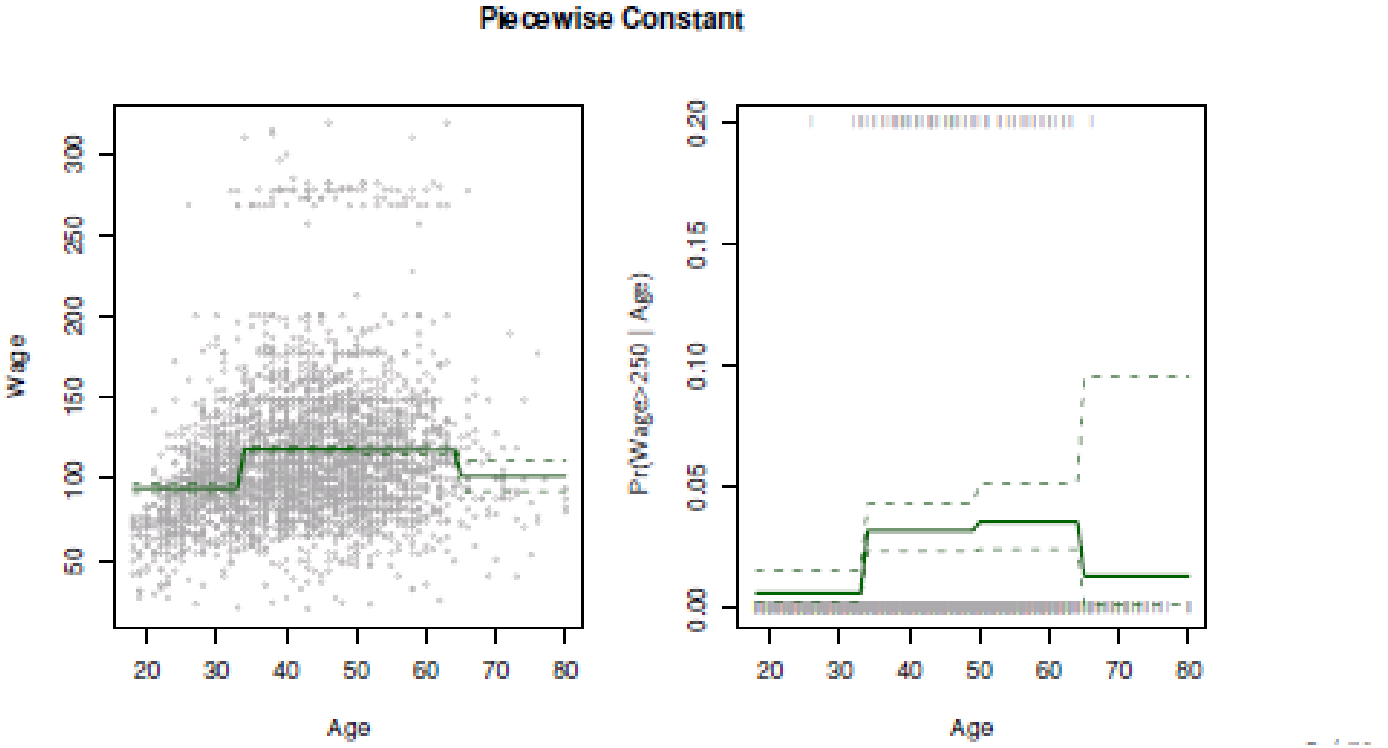
$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \ldots + \beta_d x_i^d + \epsilon_i$$



Ступенчатые функции

- Идея - обрезать переменную по отдельным областям.
- Выбор точек разрыва или *узлов* может быть проблематичным.
- Есть более «гладкие» альтернативы, такие как *сплайны*.

$$C_1(X) = I(X < 35), \quad C_2(X) = I(35 \leq X < 50), \ldots, C_3(X) = I(X \geq 65)$$

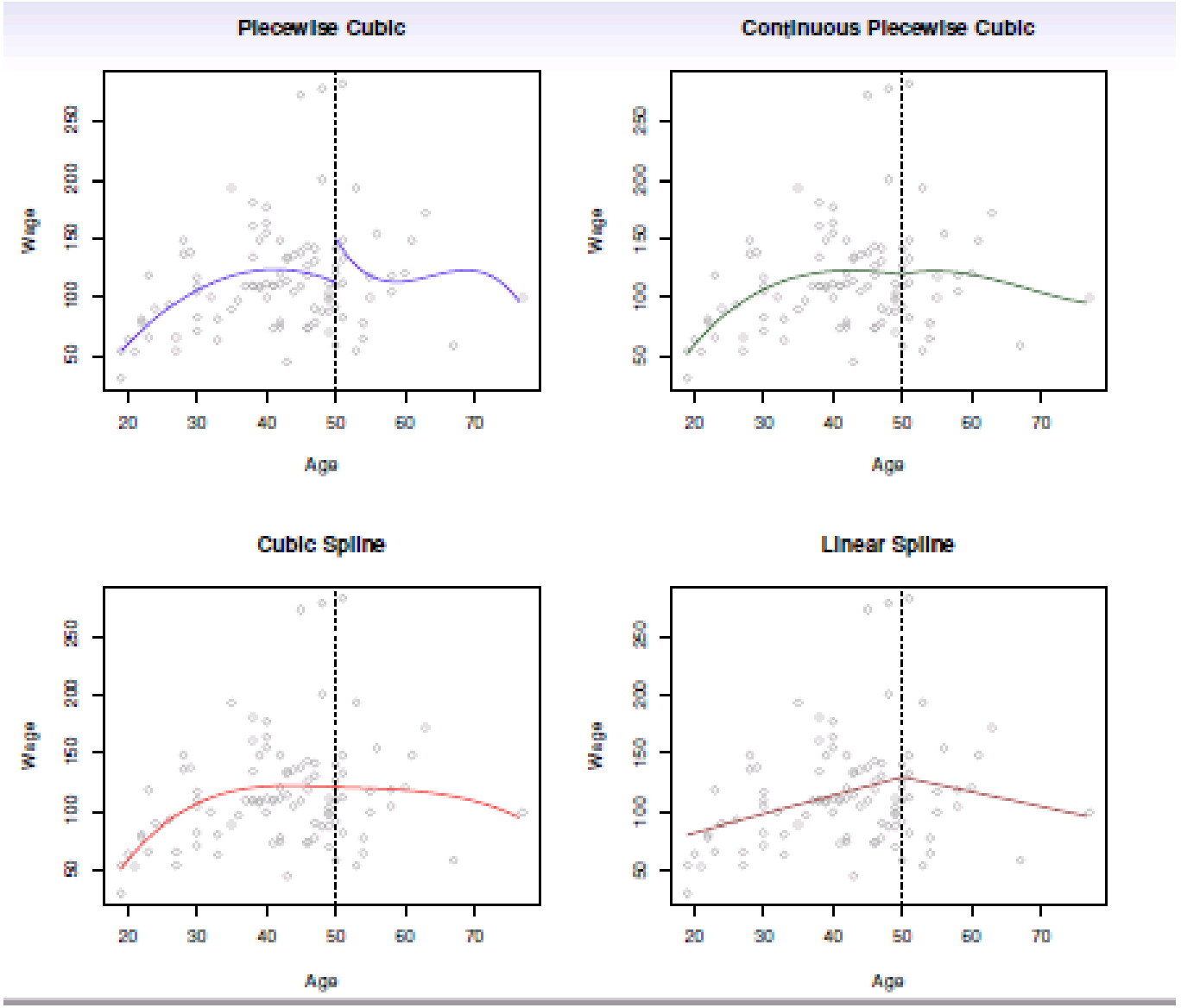


Кусочные полиномы

- Вместо одного полинома в *X* по всей его области определения мы можем использовать различные многочлены в областях, определяемых узлами.

$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c; \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } x_i \geq c. \end{cases}$$

- Лучше добавить ограничения для многочленов, например, непрерывность.
- *Сплайны* имеют «максимальную» непрерывность.


Piecewise Cubic
Continuous Piecewise Cubic
Cubic Spline
Linear Spline
Wage
Age
50
100
150
200
250
20
30
40
50
60
70

Линейные сплайны

Линейный сплайн с узлами ξ_k, k = 1,…,K является кусочно-линейным многочленом, непрерывным в каждом узле.

Мы можем интерпретировать эту модель как

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i,$$

где b_k - *базисные функции*

$$\begin{aligned} b_1(x_i) &= x_i \\ b_{k+1}(x_i) &= (x_i - \xi_k)_+, \quad k = 1, \ldots, K \end{aligned}$$

Здесь $()_+$ означает положительную часть, т.е.

$$(x_i - \xi_k)_+ = \begin{cases} x_i - \xi_k & \text{if } x_i > \xi_k \\ 0 & \text{otherwise} \end{cases}$$

Кубические сплайны

Кубические сплайны с узлами ξ_k, $k = 1,\ldots,K$ представляют собой кусочно-кубический многочлен с непрерывными производными до второго порядка в каждом узле.

Мы можем снова представить эту модель со степенными базисными функциями

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i,$$

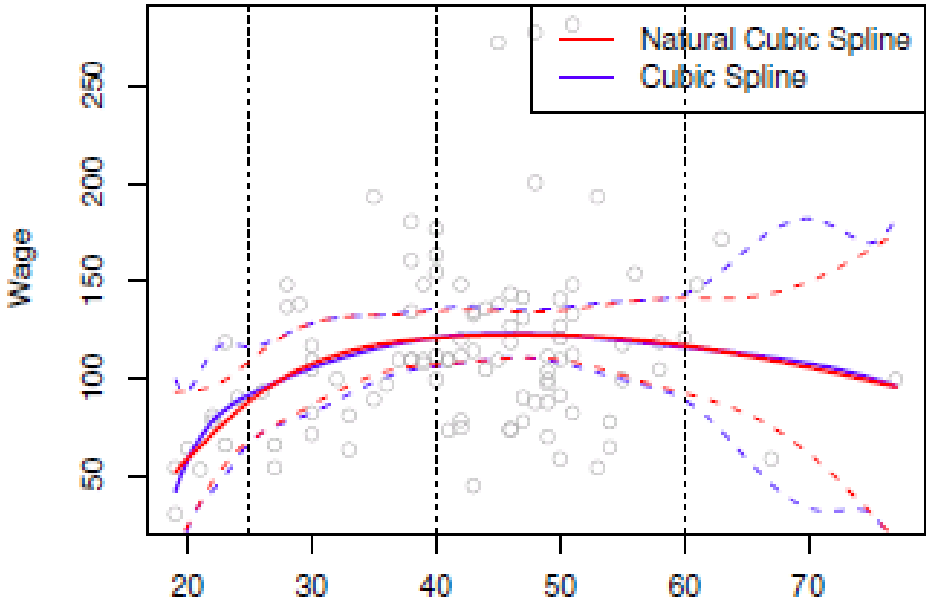
$$\begin{aligned} b_1(x_i) &= x_i \\ b_2(x_i) &= x_i^2 \\ b_3(x_i) &= x_i^3 \\ b_{k+3}(x_i) &= (x_i - \xi_k)_+^3, \quad k = 1, \ldots, K \end{aligned}$$

где

$$(x_i - \xi_k)_+^3 = \begin{cases} (x_i - \xi_k)^3 & \text{if } x_i > \xi_k \\ 0 & \text{otherwise} \end{cases}$$

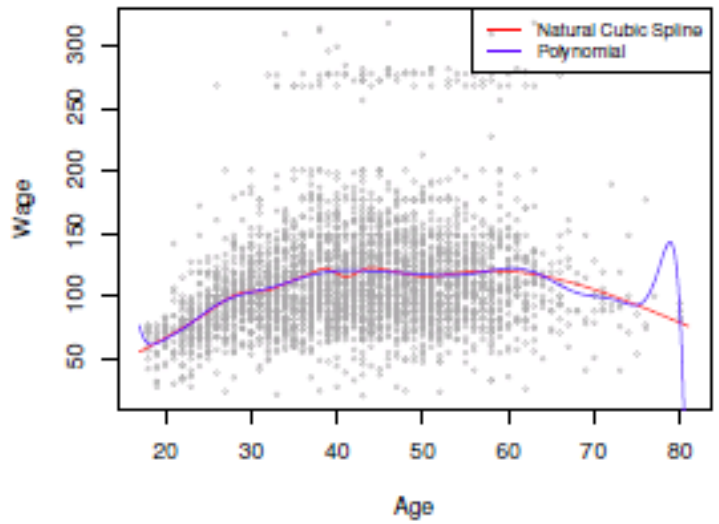
Естественные кубические сплайны

Естественный кубический сплайн осуществляет линейную экстраполяцию за граничные узлы. Это добавляет 4 = 2 * 2 дополнительных ограничения и позволяет нам делать больше внутренних узлов для тех же степеней свободы, по сравнению с обычным кубическим сплайном.



Размещение узлов

- Одна из стратегий состоит в том, чтобы определить значение *K* (количество узлов), а затем поместить их в соответствующие квантили наблюдаемого *X*.
- Кубический сплайн с *K* узлами имеет *K + 4* параметров или степеней свободы.
- Естественный сплайн с K узлами имеет K степеней свободы.



Сравнение полинома степени 14 и естественного кубического сплайна, каждый с 15df.

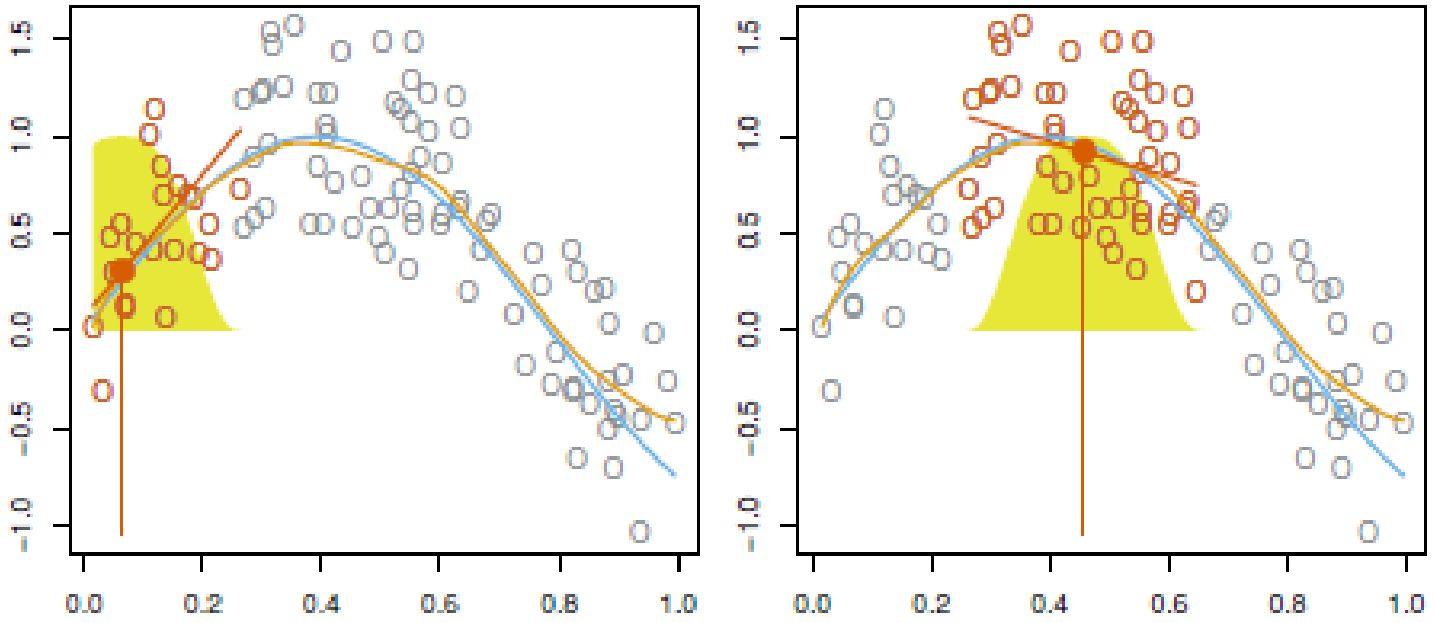
Сглаживание сплайнов

Рассмотрим критерий для подгонки гладкой функции *g(x)* к некоторым данным:

$$\underset{g \in \mathcal{S}}{\text{minimize}} \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$$

- Первый терм - RSS и он нацелен на то, чтобы *g(x)* соответствовала данным в каждом x_i.
- Второй терм - это штраф *за грубое приближение* и он управляет тем, насколько *g(x)* «извилистая». Он варьируется *параметром настройки* $\lambda \geq 0$.
  - Чем меньше, тем более извилистая функция, в конечном счете интерполирующая y_i когда $\lambda = 0$.
  - Когда $\lambda \to \infty$, функция *g(x)* становится линейной.

Локальная регрессия

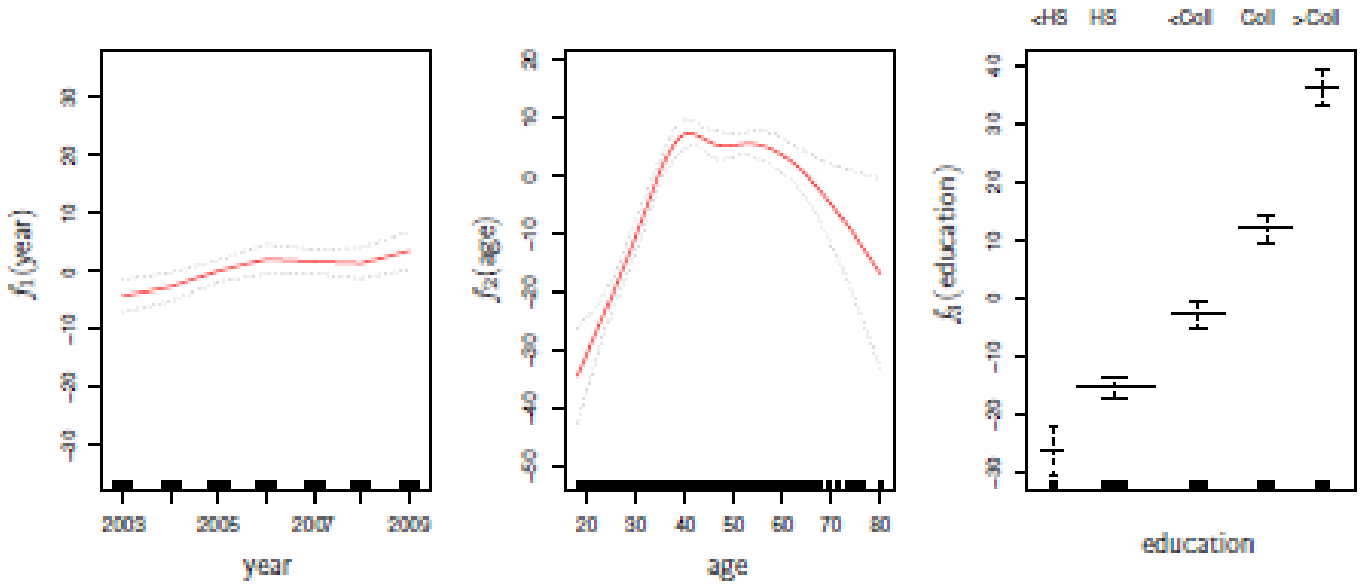


С помощью скользящей весовой функции мы отдельно подгоняем линейные участки по диапазону X с помощью взвешенных наименьших квадратов.

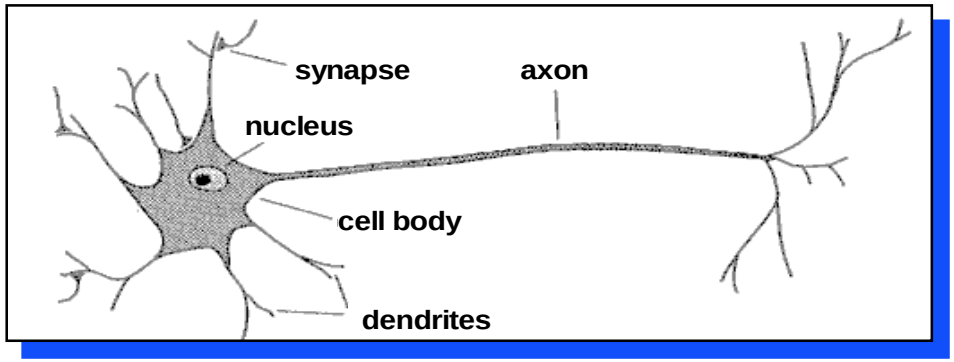
Обобщенные аддитивные модели

Рассмотрим гибкие нелинейные модели с несколькими переменными, но сохраним аддитивную структуру линейных моделей.

$$y_i = \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}) + \epsilon_i.$$



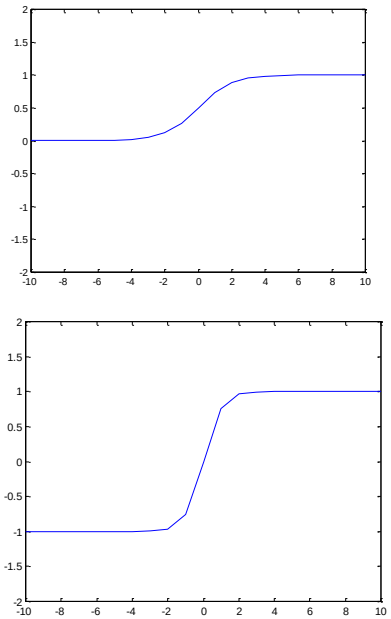
Нейронные сети - биологическая мотивация



- Человеческий мозг
  - Более 10^6 клеток (нейронов)
  - Каждый нейрон соединен через 10^6 синапсов с другими нейронами
  - Мозг может: обучаться, адаптироваться, распознавать образы, осознавать «себя», устойчив к шуму, травмам и ошибкам
- Нейрон
  - «Входные» отростки (дендриты)
  - «Выходные» отростки (аксоны)
- Информация (сигнал, «нервный импульс»):
  - идет от дендритов к аксону через тело (ядро) клетки
- Аксоны соединяются с дендритами (других клеток) через синапсы
  - Синапсы разные по силе могут быть возбуждены или подавлены

Искусственный нейрон

- Определение:
  - Нелинейная, параметризованная функция с ограниченным диапазоном значений
- Функции активации:



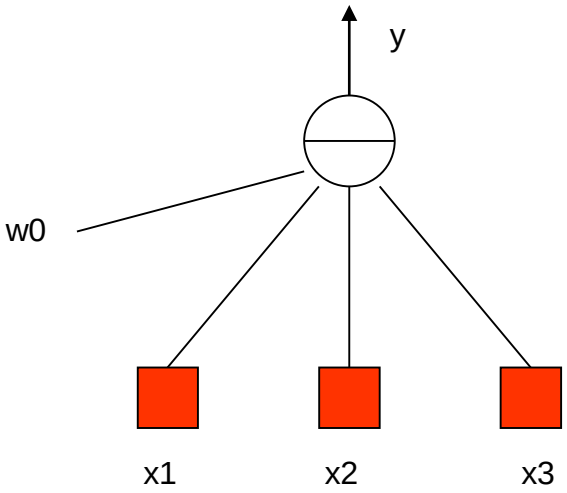
логистическая

$$y = \frac{1}{1+\exp(-x)}$$

Гиперболический тангенс

$$y = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$

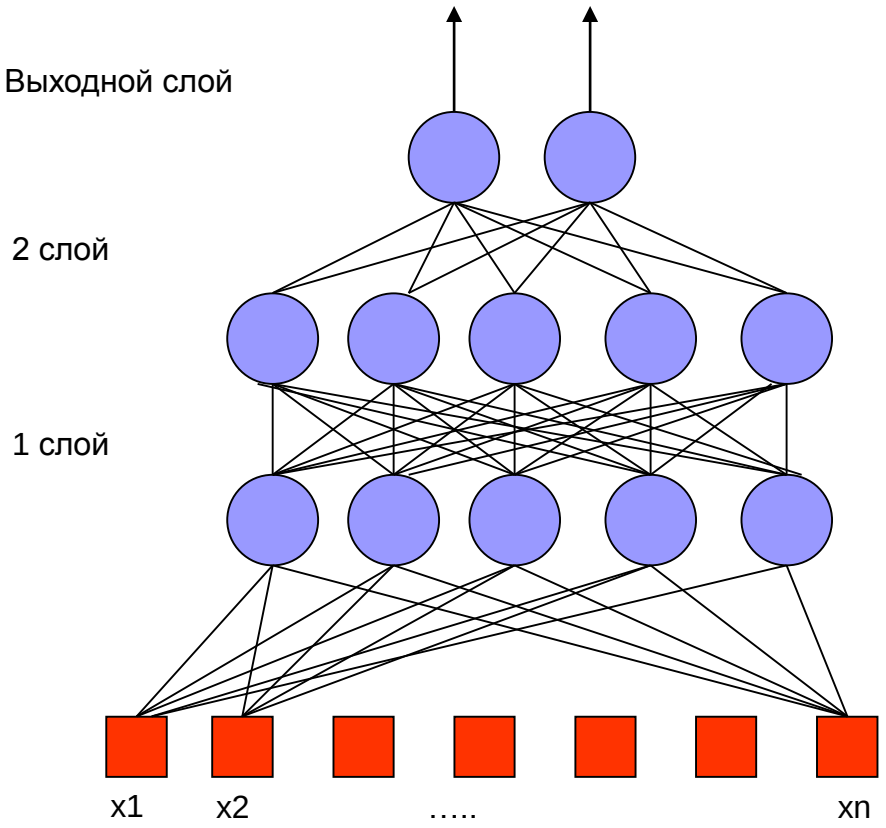
$$y = f\left(w_0 + \sum_{i=1}^{n-1} w_i x_i \right)$$



Нейронная сеть (искусственная)

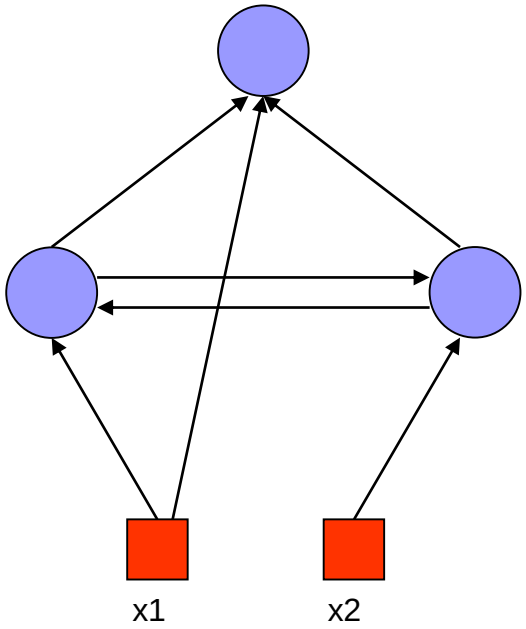
- Математическая модель для решения задач машинного обучения
  - Реализуется группой соединенных нейронов для моделирования нелинейных зависимостей
- Задачи:
  - Классификации, дискриминации, оценки плотности, регрессии, группировки и кластеризации, выявления зависимостей, главных и независимых компонент
- Два типа нейронных сетей:
  - Сети прямого распространения (Feed forward Neural Networks)
  - Рекуррентные нейронные сети (Recurrent Neural Networks)

Сети прямого распространения



- Сигнал передается от входного уровня нейронов к выходному по «слоям»
- Расчет нелинейных выходных функций, от входных переменных каждая, как композиции алгебраических функций активации
- Нет задержек, времени, т.к. нет циклов

Рекуррентные сети



- Произвольные топологии с циклами
- Моделирует системы с состояниями (динамические системы)
- Есть понятие «задержки» у некоторых весов
- Процесс обучения - тяжелый
- Результат не всегда предсказуемый
  - Нестабильный (неустойчивый) сигнал на выходе
  - Неожиданное поведение (осцилляции, хаос, …)

Обучение нейронных сетей (с учителем)

- Цель –найти параметры нейронов (веса)
- Процедура:
  - Дан тренировочный набор – множество пар (объект, отклик)
  - Оценить, насколько хорошо сеть аппроксимирует этот набор
  - Модифицировать параметры для улучшения аппроксимации
- Нейросети (для обучения с учителем)
  - универсальные аппроксиматоры (для нерекуррентных сетей)
- Достоинства:
  - Адаптивность
  - Обобщающая способность (сложность определяется в том числе архитектурой сети)
  - Устойчивость к ошибкам – не катастрофическая потеря точности при «порче» отдельных нейронов и весов, так как информация «распределена» по сети

Правила обучения

- *Правило Хэбба*: сила связи (вес связи) между нейронами i и j должна модифицироваться согласно формуле::

$$\Delta w_{ij} = \eta \, \hat{y}_i x_j$$

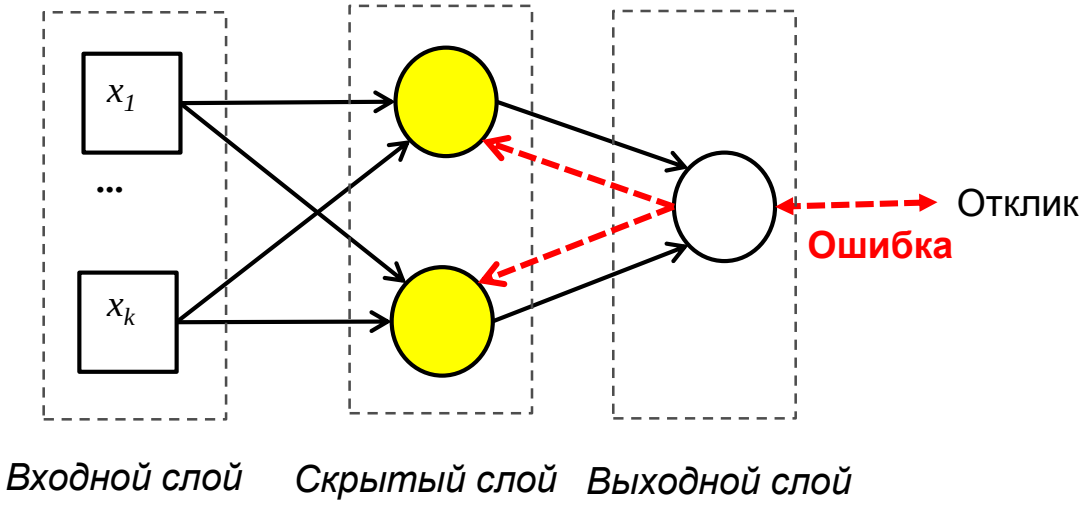
- Параметр скорости обучения, η, контролирует размер шага изменения.
- Чем меньше скорость обучения тем медленней процесс сходится.
- Большой размер шага обучения может привести к расходимости.
- Правило Хэбба не стабильно.
- Более стабильный вариант:

$$\Delta w_{ij} = \eta \, (y_i - \hat{y}_i) \, x_j$$

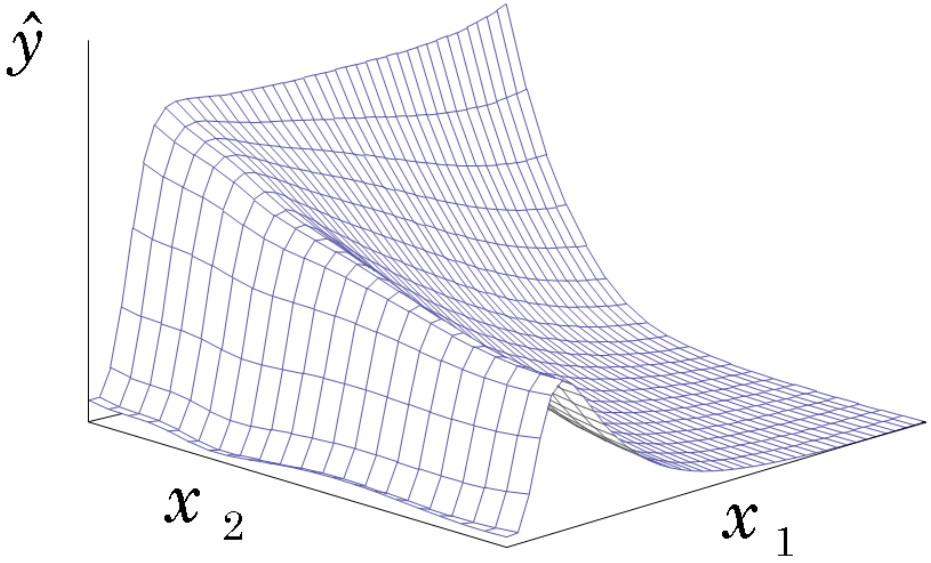
- Называется *дельта правио.*
- Иногда правило наименьших квадратов, т.к. минимизирует квадратичную ошибку.

Обобщенное дельта правило

- Два этапа (для каждого примера):
  1. Прямой ход: прогон примера через сеть и расчет ошибки (отклонения отклика от прогноза).
  2. Обратный ход: прогон ошибки обратно – модификация весов по дельта правилу
  3. Пока не сойдется (веса перестанут существенно меняться).



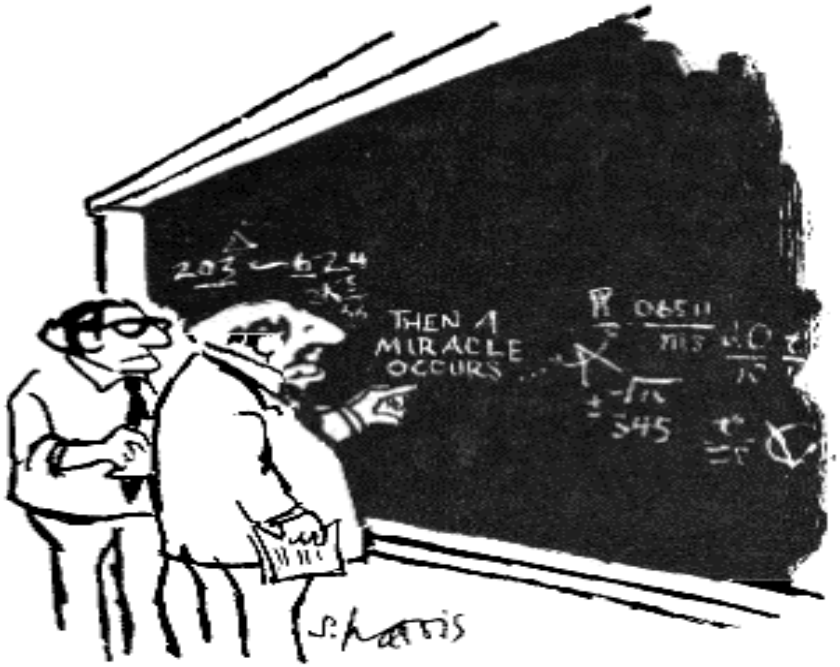
Универсальный апроксиматор



- Любая ограниченная функция может быть сколь угодно точно приближена некоторой нейронной сетью с конечным числом нейронов

Не нужна явная формулировка искомой зависимости

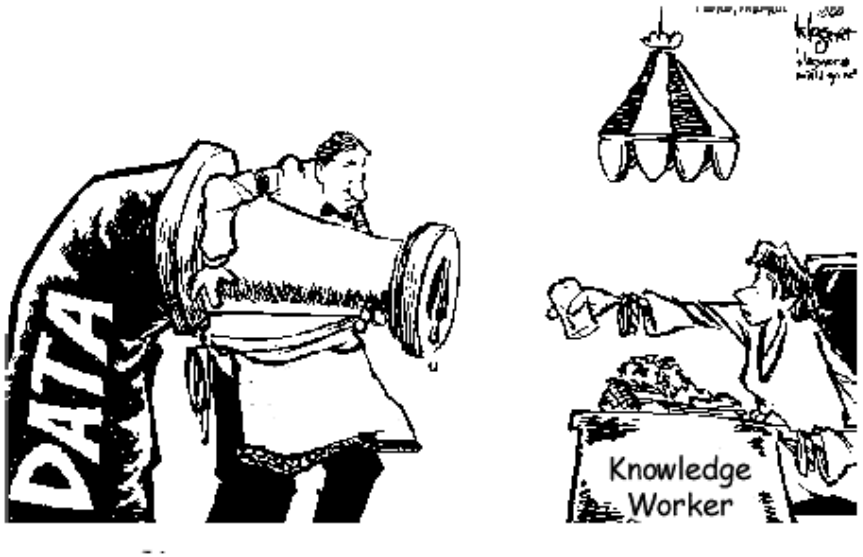
- Не нужно задавать форму зависимости априори (как в регрессиях и опоных векторах), даже приблизительно «понимать» ее не нужно
- сложнее сеть => сложнее зависимость, быстрее переобучение



"I think you should be more explicit here in step two."

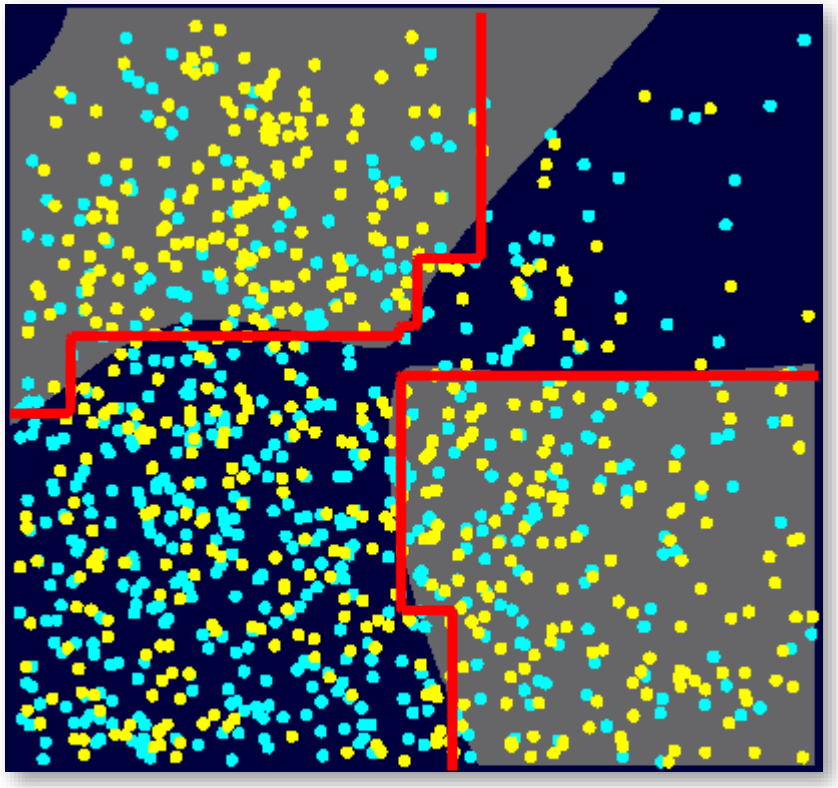
Скорость применения

- Нейронные сети - один из самых «быстрых» моделей на этапе прогнозирования.
- Могут применяться для Больших данных (но мало кто этим пока пользуется).

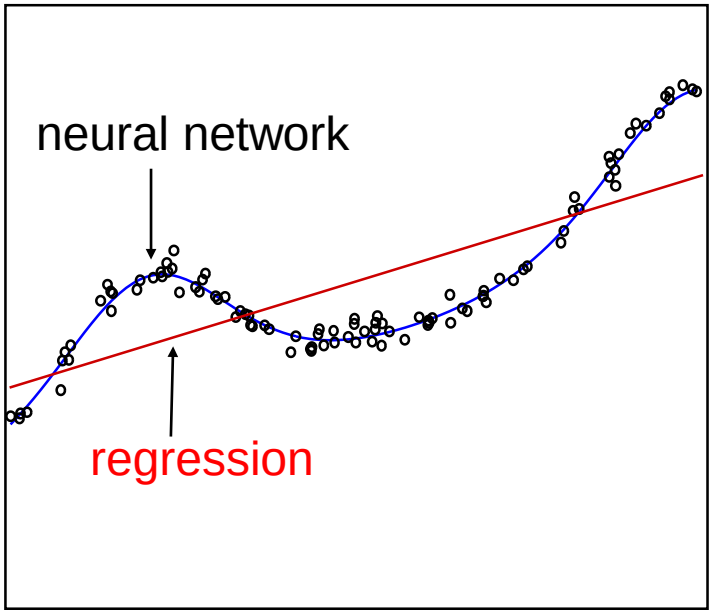


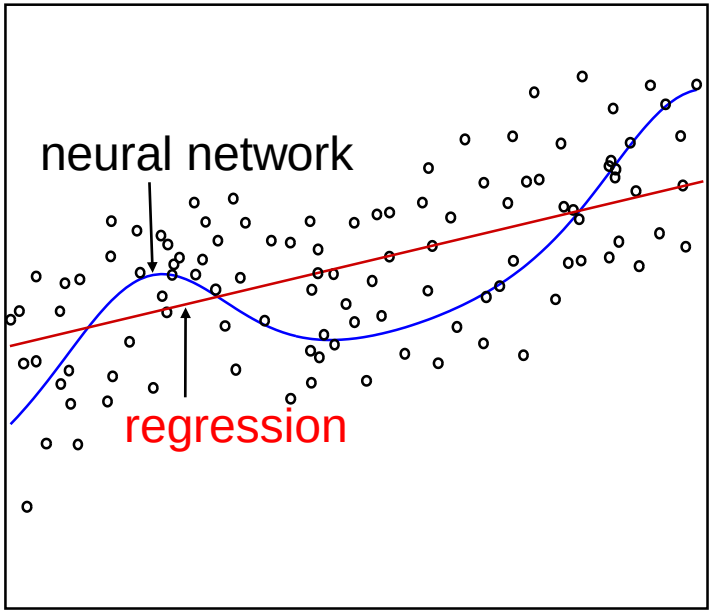
Недостаточная итерпретируемость

- Известная проблема *черного ящик*.
- Вариант решения - Суррогатные модели
  - интерпретируемые модели типа деревьев решений для «приближения» результата нейросети.



Влияние шума





$$\frac{\text{signal}}{\text{noise}} = \text{high}$$

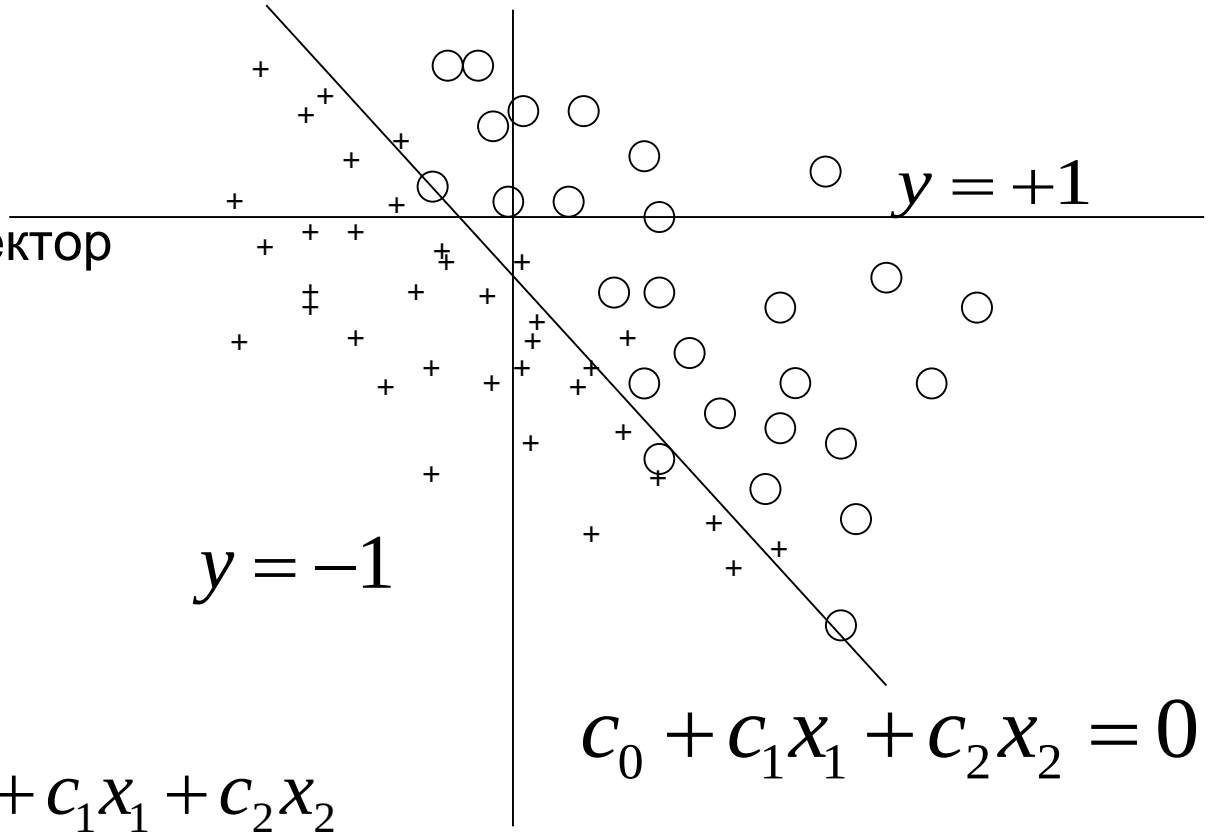
$$\frac{\text{signal}}{\text{noise}} = \text{low}$$

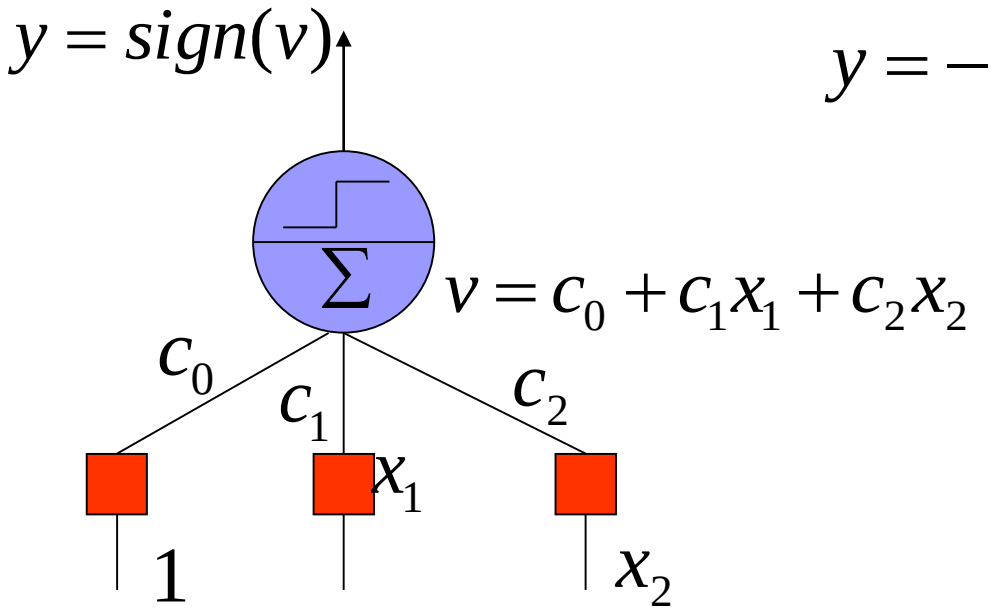
критика

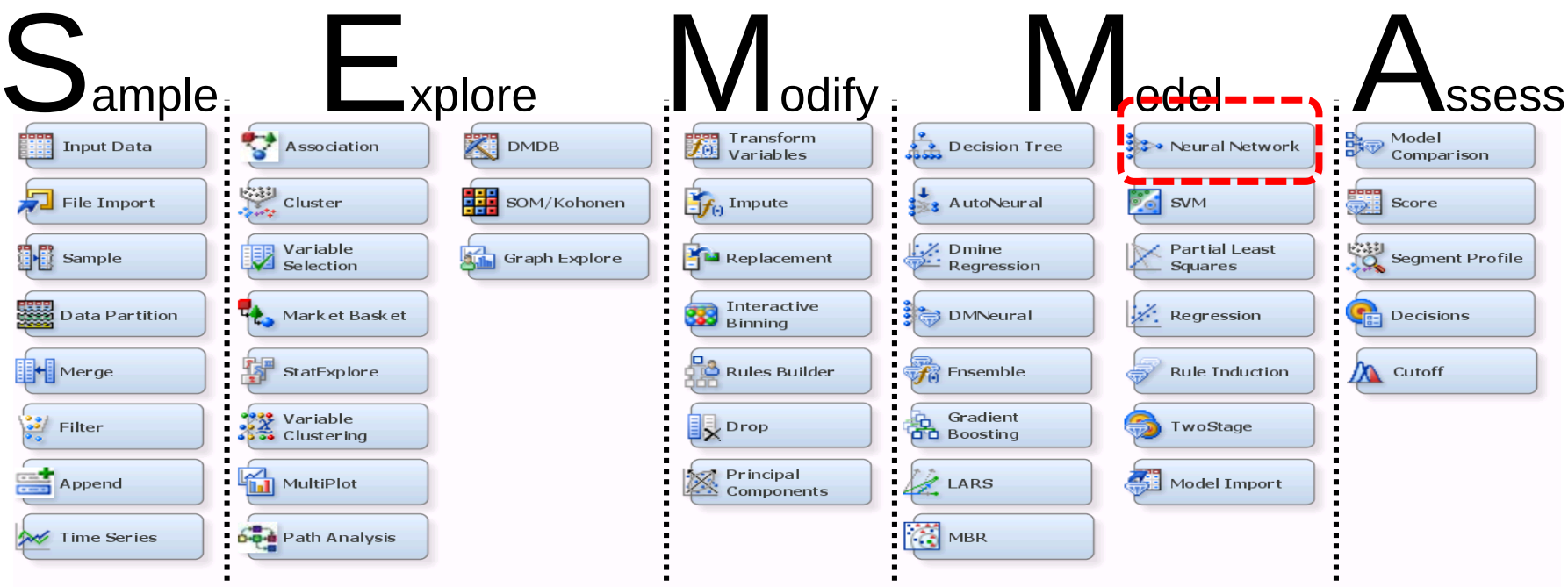
- “It is shown that, at least for the data used in this study, the fit achieved [by regression] is approximately the same, but the process of configuring and setting up a neural network for a database marketing application
is not straightforward, and may require extensive experimentation and computer resources.”
  - Zahavi and Levin. 1997. “Applying Neural Computing to Target
  - Marketing.” *Journal of Direct Marketing*.
- А по сути – для задачи, в которой нейронная сеть дает хороший результат, почти всегда можно найти достаточно точное решение на основе более простых регрессионных моделей.

Персептрон Розенблатта

- Rosenblatt (1962)
- Линейное разделение:
- вход :вещественный вектор
- выход :1 или -1
- Решающее правило:

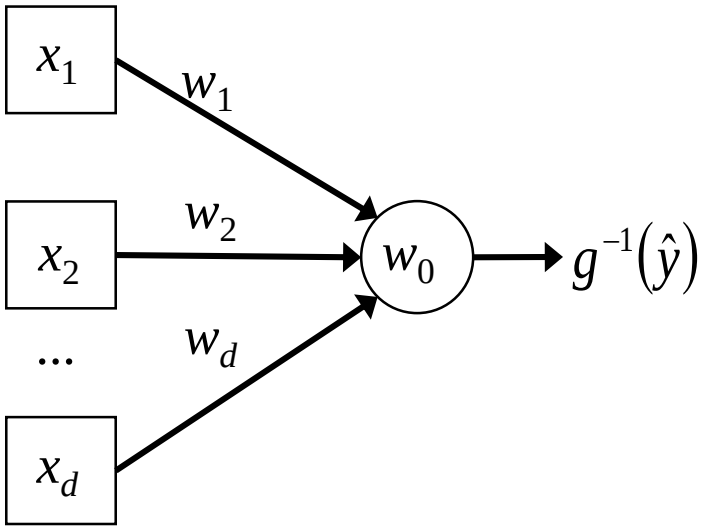





Sample
Explore
Modify
Model
Assess
Input Data
File Import
Sample
Data Partition
Merge
Filter
Append
Time Series
Association
Cluster
Variable Selection
Market Basket
StatExplore
Variable Clustering
MultiPlot
Path Analysis
DMDB
SOM/Kohonen
Graph Explore
Transform Variables
Impute
Replacement
Interactive Binning
Rules Builder
Drop
Principal Components
Decision Tree
AutoNeural
Dmine Regression
DMNeural
Ensemble
Gradient Boosting
LARS
MBR
Neural Network
SVM
Partial Least Squares
Regression
Rule Induction
TwoStage
Model Import
Model Comparison
Score
Segment Profile
Decisions
Cutoff

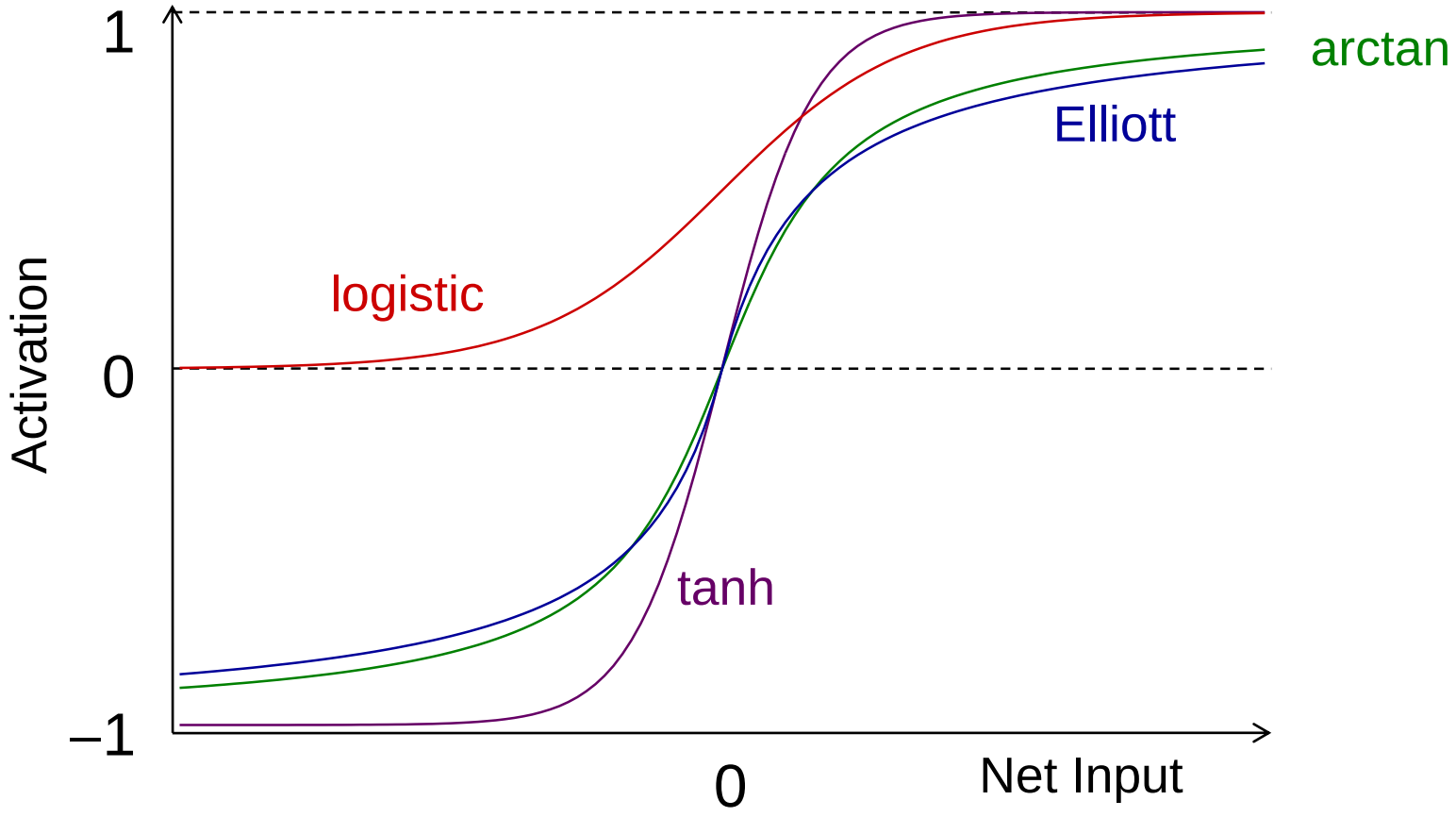
Линейный персептрон (он же GLM)

$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{d} w_i x_i$$

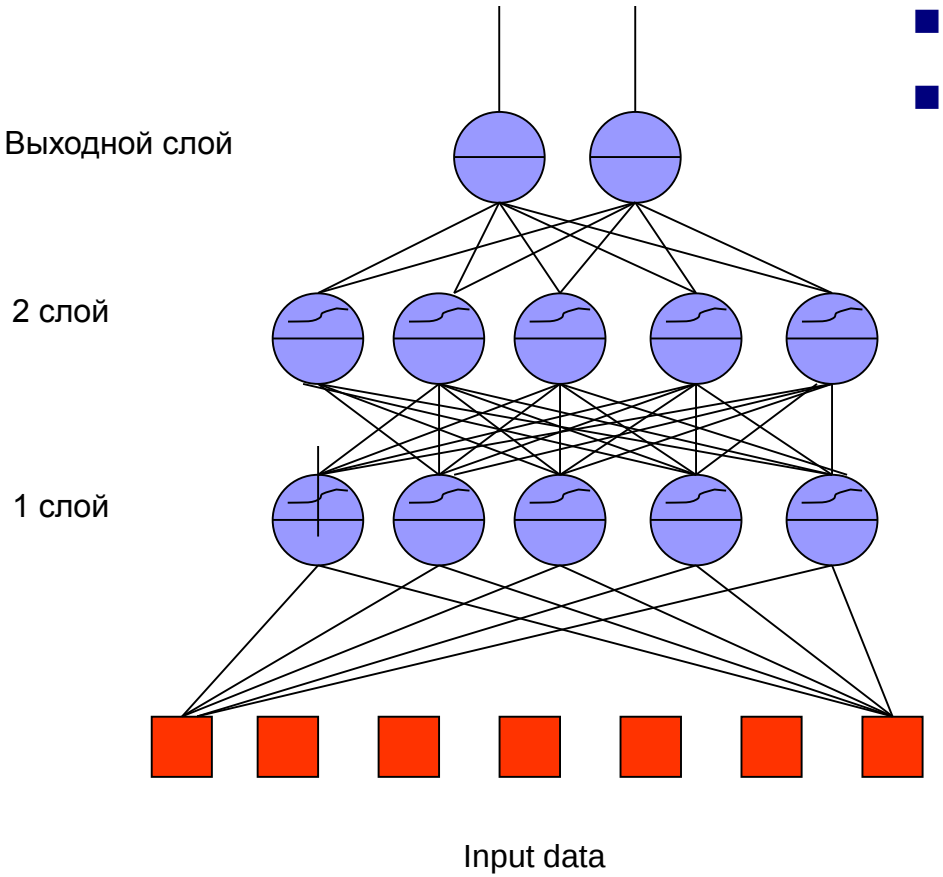


- Доступные функции комбинации:
  - Linear взвешенная сумма(default).
  - Additive не взвешенная сумма
  - Equal Slopes сумма с одинаковыми весами (но сдвиг разный)

Функции активации (она же обратная к функции связи)



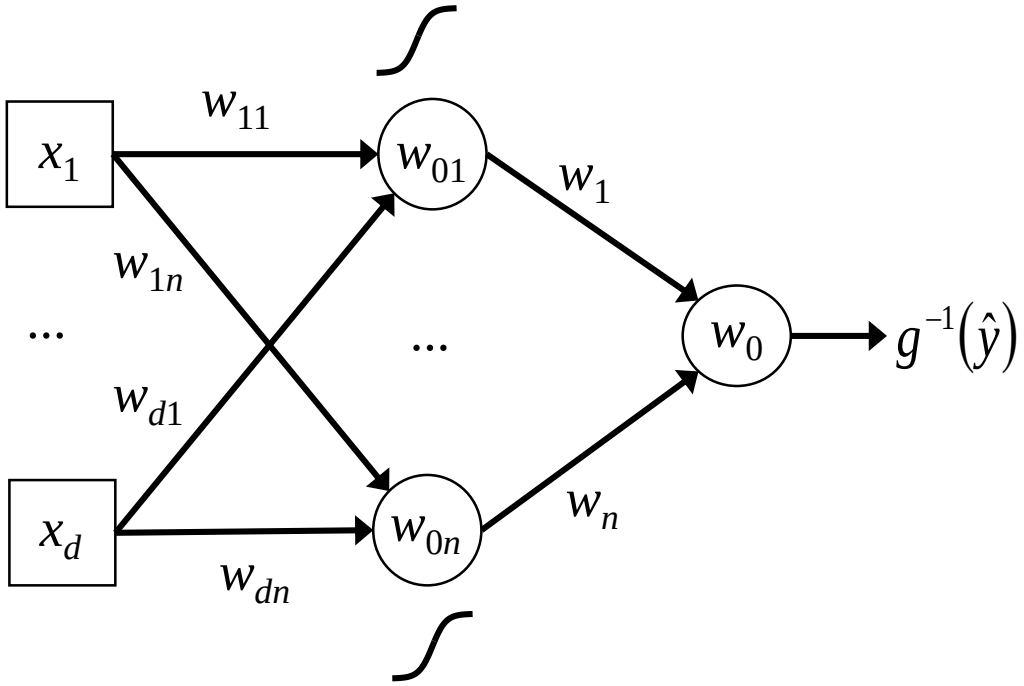
Многослойный персептрон



- Один или более скрытых уровней
- Функции активации сигмоидального типа

Многослойный персептрон

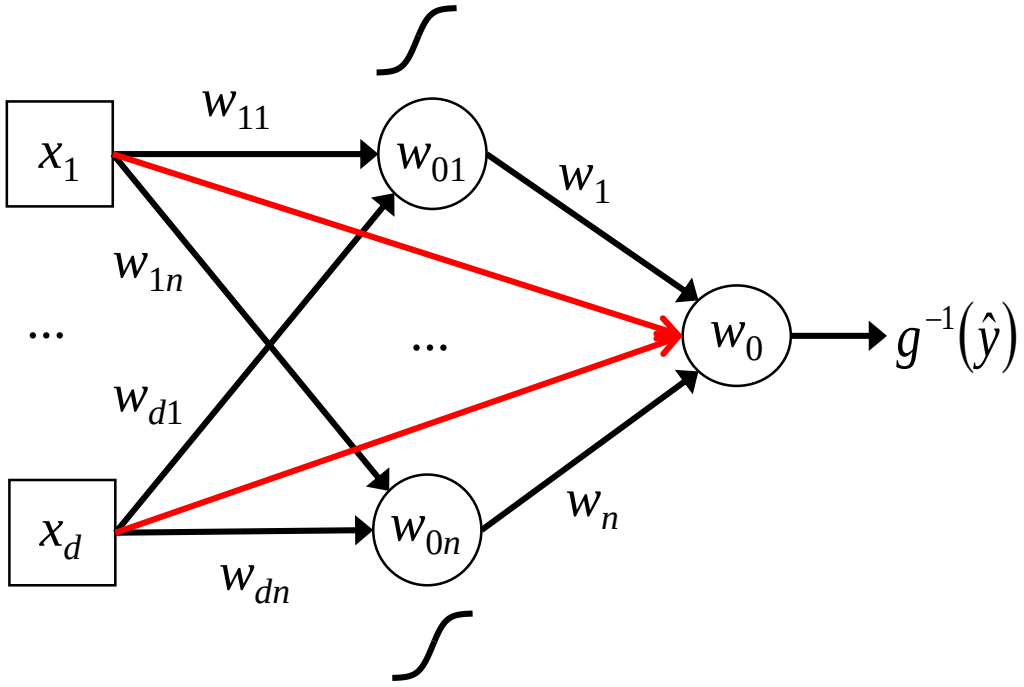
$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{h} w_i g_i \underbrace{\left(w_{0i} + \sum_{j=1}^{d} w_{ij} x_j \right)}_{\text{Скрытый слой}}$$



Персептрон с прямыми соединениями

$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{h} w_i g_i\left(w_{0i} + \sum_{j=1}^{d} w_{ij} x_j\right) + \sum_{k=1}^{d} w_{11k} x_k$$

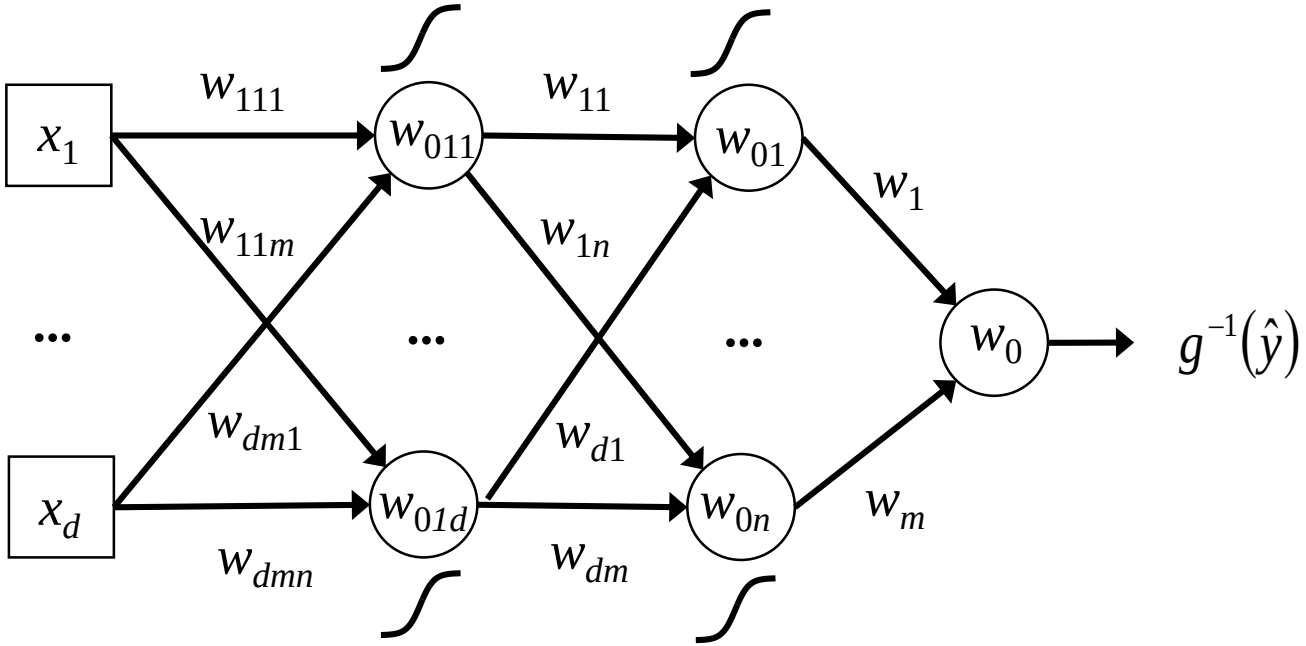
Скрытый слой Прямые соединения



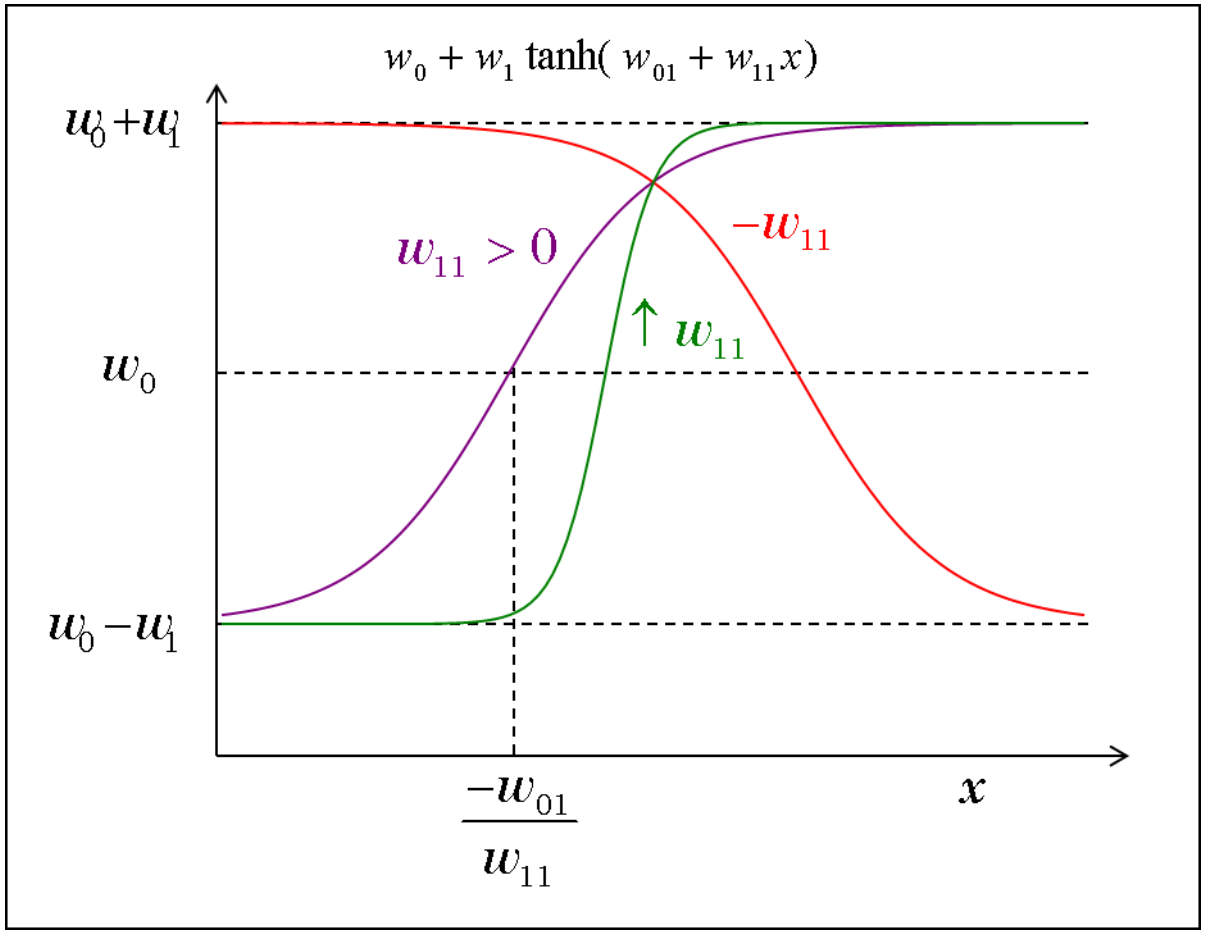
Два и более скрытых слоя

$$g^{-1}(\hat{y}) = w_0 + \sum_{k=1}^{m} w_k g_k \left(w_{0k} + \sum_{j=1}^{n} w_{jk}\, g_j \left(w_{0\,jk} + \sum_{i=1}^{d} w_{ijk} x_i \right) \right)$$

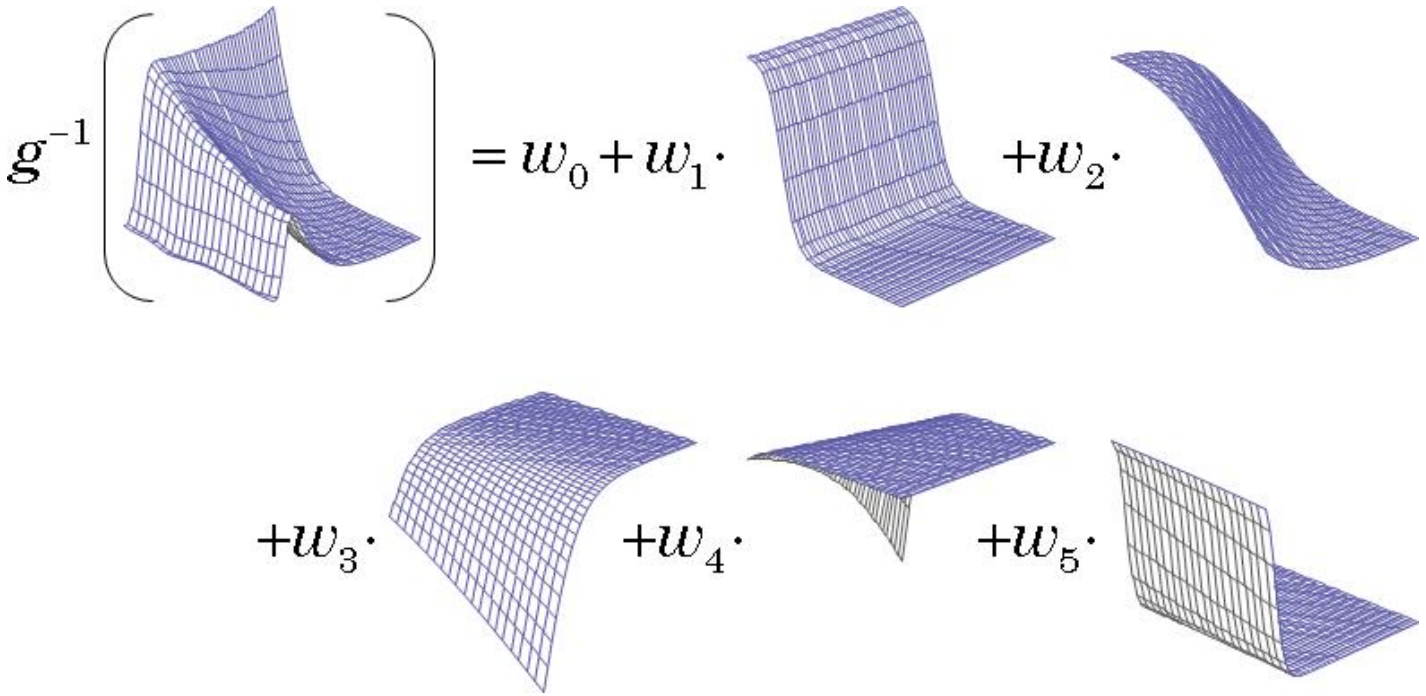
Вложенные скрытые слои



Форма сигмоида

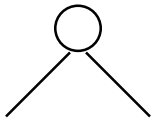
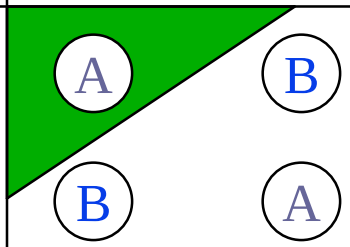
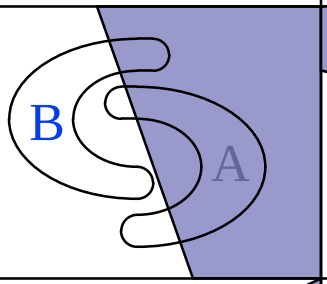
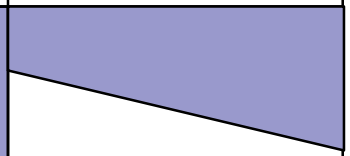
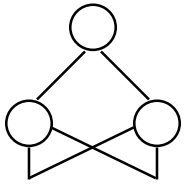
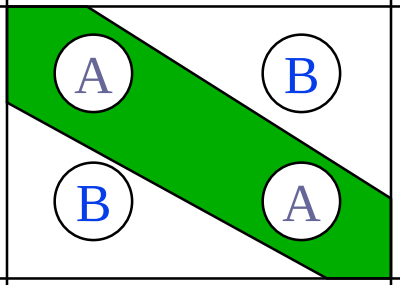
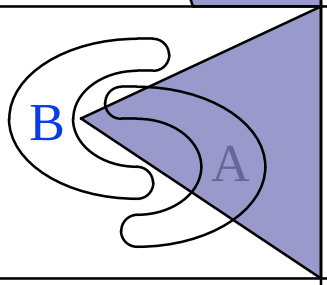
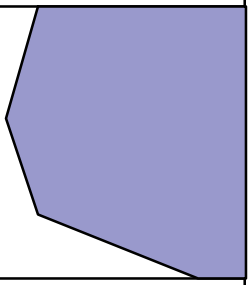
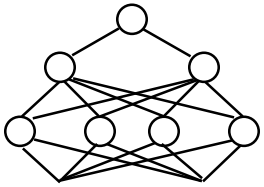
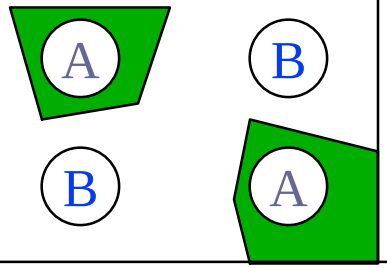
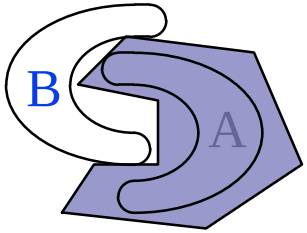
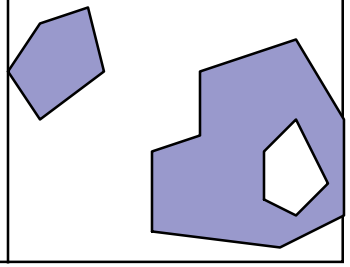


Разложение по базису сигмоидальных функций



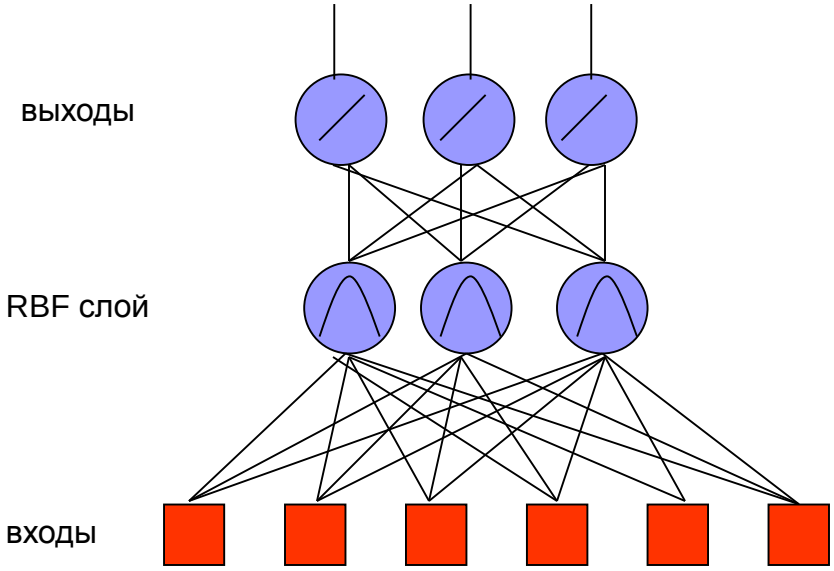
- Сумма правильного числа правильно вложенных взвешенных сигмоидов с подобранными коэфициентами может приблизить любую зависисимость
- Оптимальная архитектура для каждой задачи своя, подбирается эмпирически

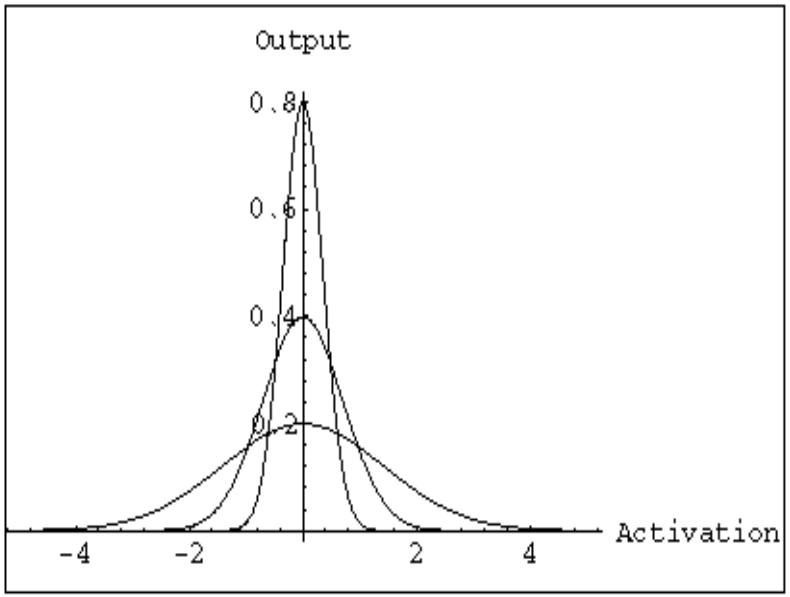
Типы решаемых задач

Архитектура сети	*Тип разделяющего правила*	*XOR задача*	*Получаемые области*	*Самый общий возможный вид*
Только выход 	*Линейная гиперплоскость*			
однослойный 	*Выпуклые открытые области*			
двухслойный 	*Произвольные области (сложность ограничена числом нейронов)*			

Радиально-базисные сети

- Свойства:
  - Один скрытый слой нейронов
  - Функция активации типа потенциальной (ядерной)
  - Зависит от расстояния между входным сигналом и прототипом





- Скрытый слой:
  - Каждый нейрон связан с прототипом – центр «зоны влияния»
  - Обычно гауссова ядерная функция, значение зависит от расстояния, но не от конкретных значений:

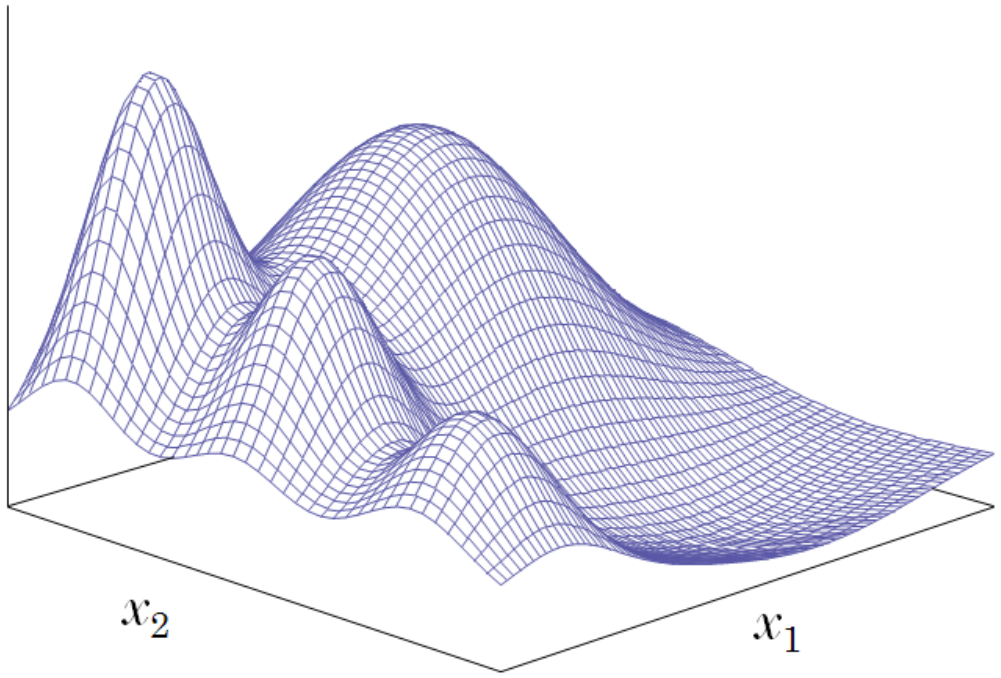
$$\Phi\left(\left\|x-c_j\right\|\right)=\exp-\left(\left\|x-c_j\right\|/\sigma_j\right)^2$$

- Выходной слой линейный, реализуемая функция:

$$s(x)=\sum_{j=1}^{K} W_j \Phi\left(\left\|x-c_j\right\|\right)$$

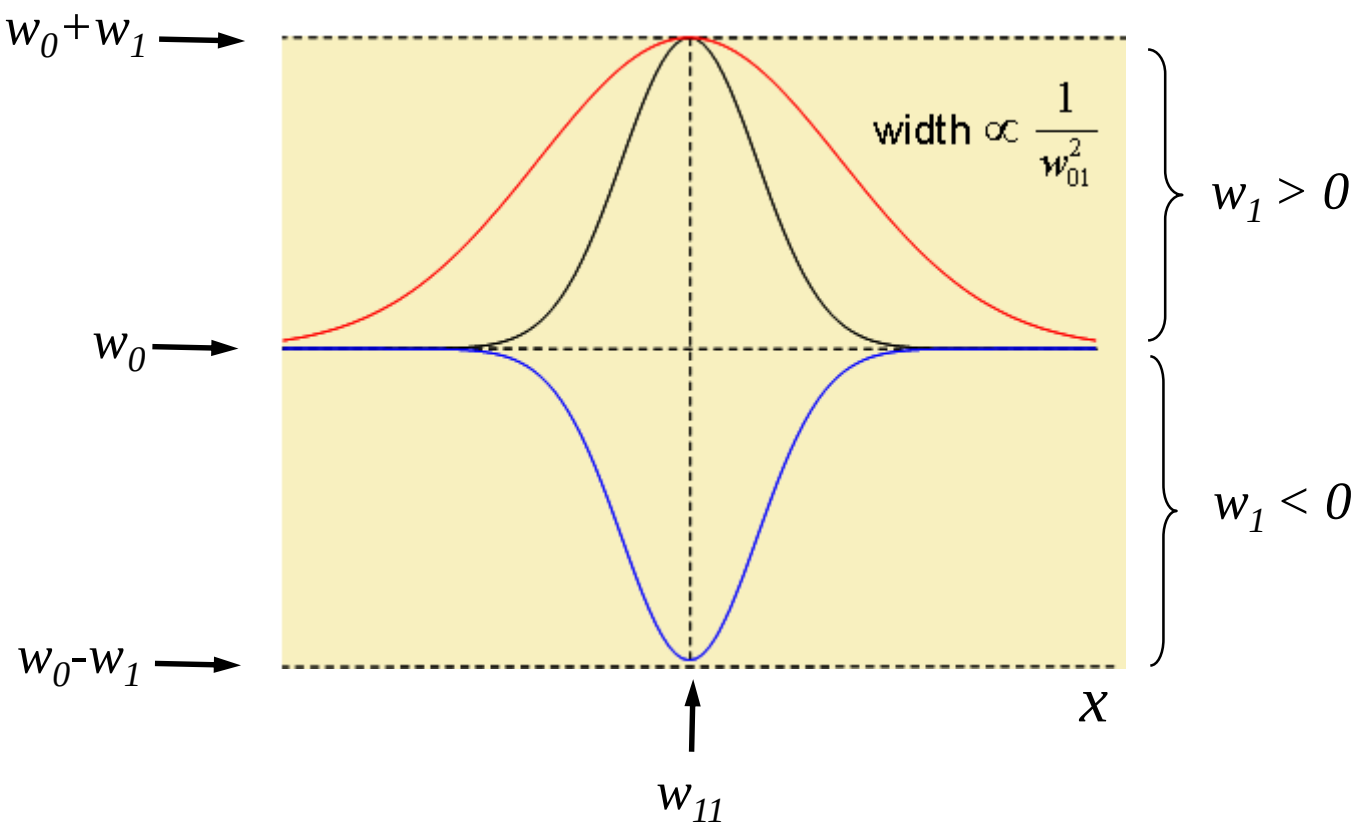
- Прототипы - центры регионов классов с высокой плотностью

- Ordinary Radial Basis Functions (ORBFs)
- Normalized Radial Basis Functions (NRBFs)



Форма функции гаусса

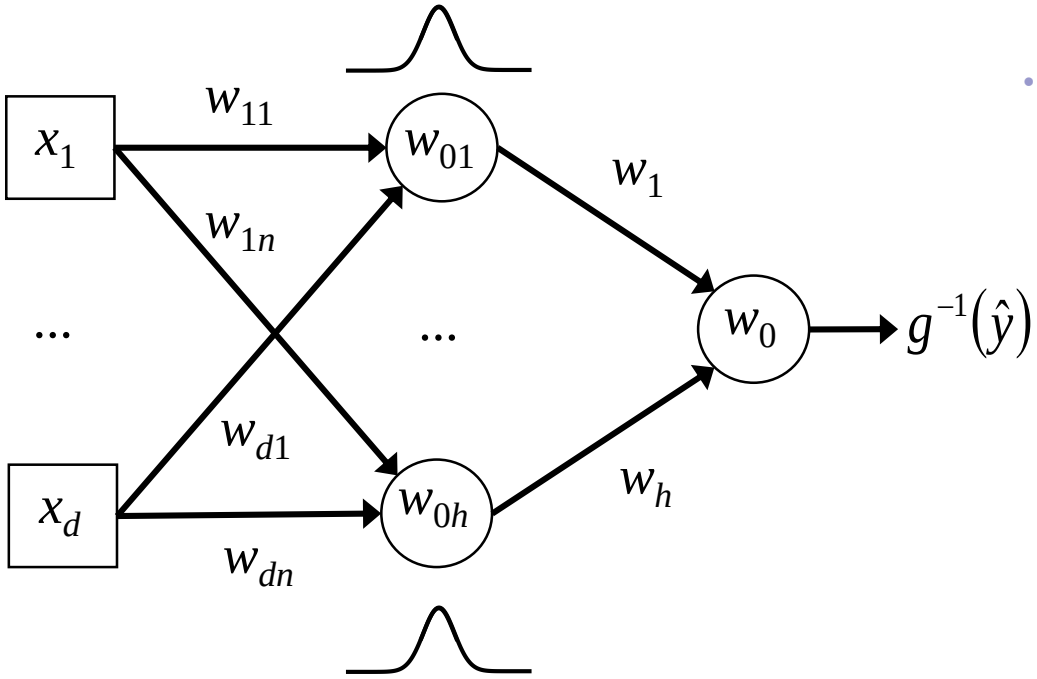
$$w_0 + w_1 \exp\left(-w_{01}^2 \left(x - w_{11}\right)^2\right)$$



RBF нейронная сеть

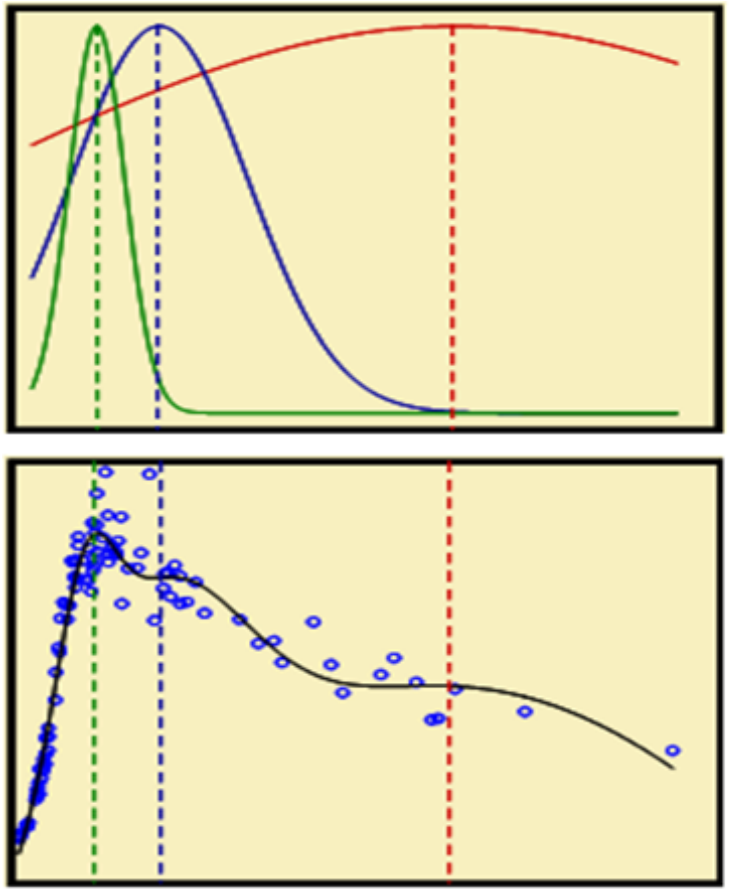
$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{h} w_i \exp\left[-w_{0i} \left(\sum_j (w_{ij} - x_j)^2 \right) \right]$$

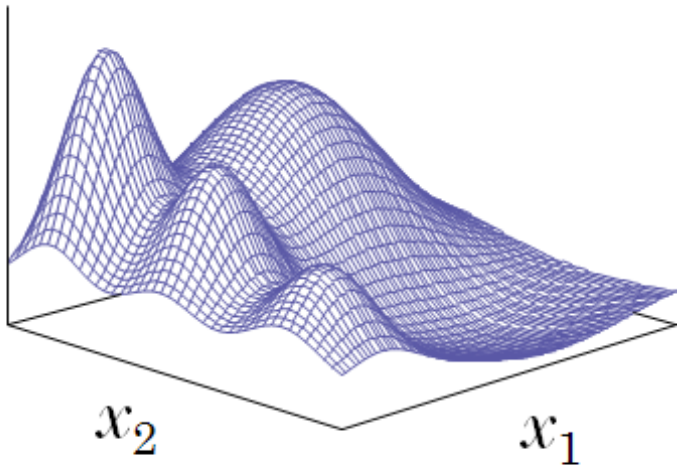
Скрытый слой



- Типы параметров обычной RBF сети:
  - **XRADIAL -** высота и ширина ядра различные у всех нейронов
  - **EQRADIAL -** высота и ширина ядра одинаковые
  - **EWRADIAL -** одинаковая ширина
  - **EHRADIAL -** одинаковая высота

Проблема локального эффекта



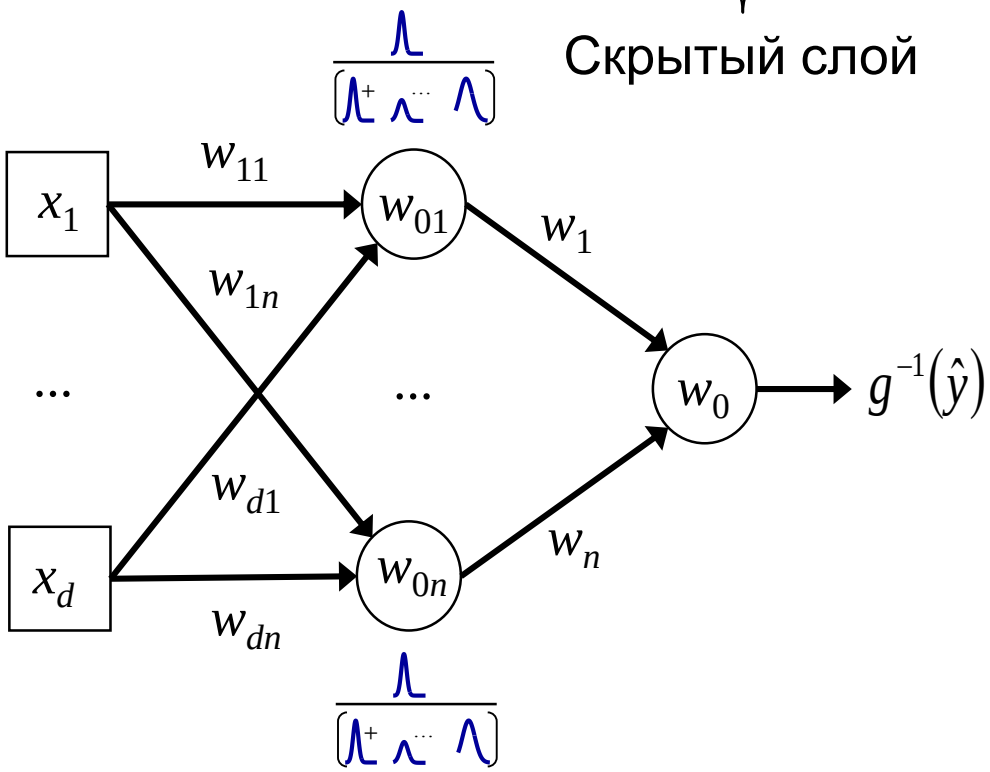


- Локальный эффект:
  - сложнее функция – больше прототипов
  - Проклятие размерности

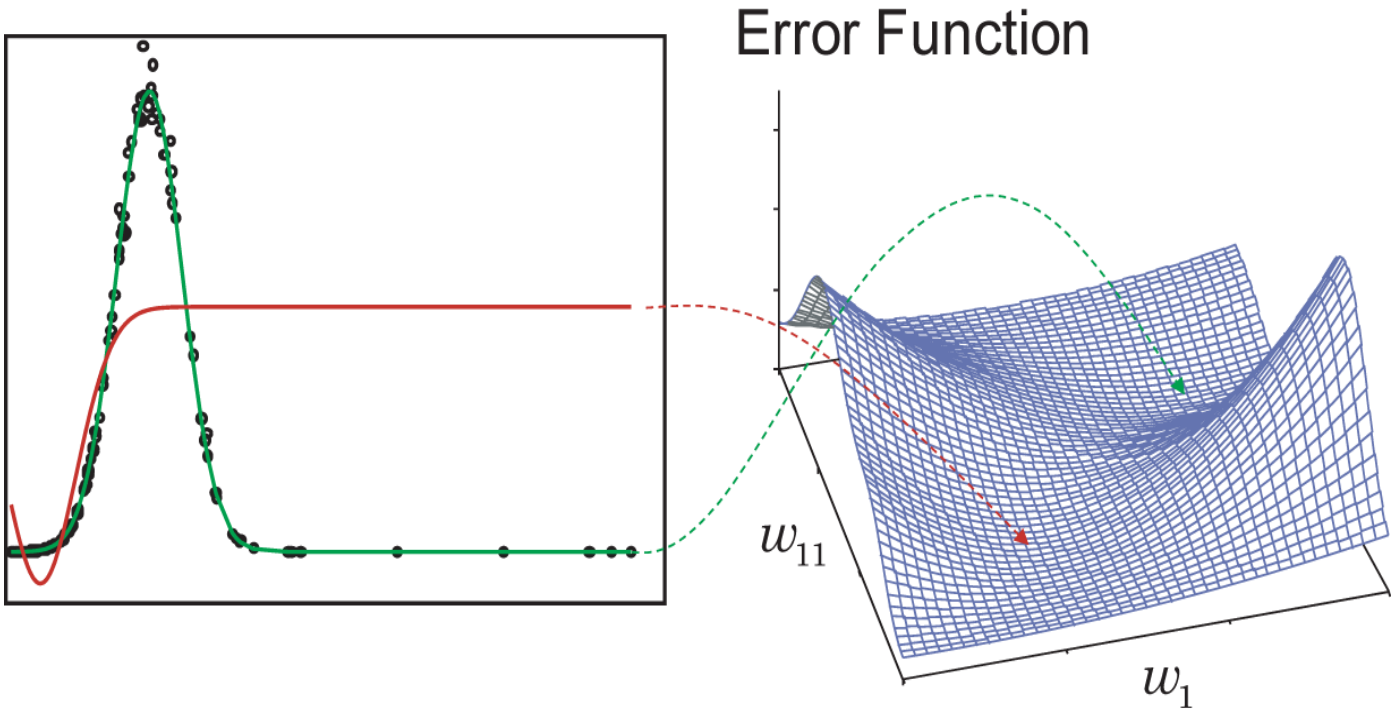
Нормализованные радиально-базисные сети

$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{h} w_i \; \text{softmax}\left\{ f \,.\ln(a_i) - w_{0i}^2 \left(\sum_j (w_{ij} - x_j)^2 \right) \right\}$$

Скрытый слой

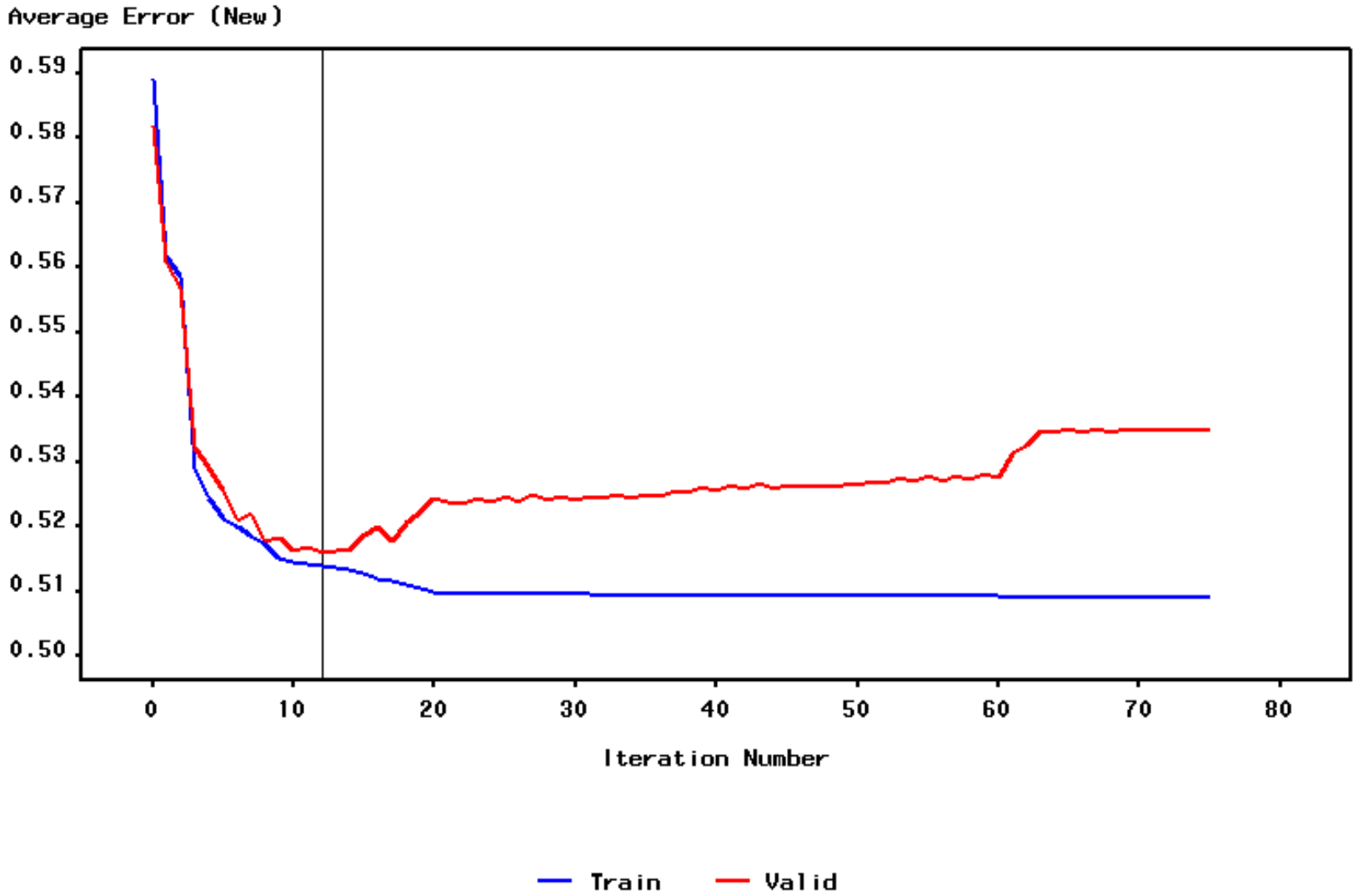


Проблема локальных минимумов

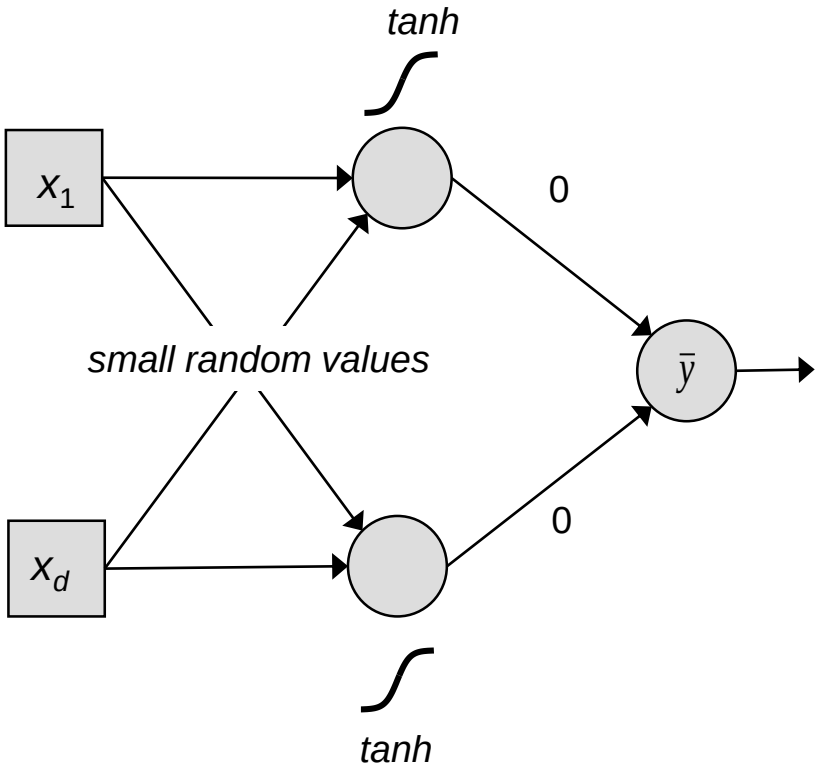


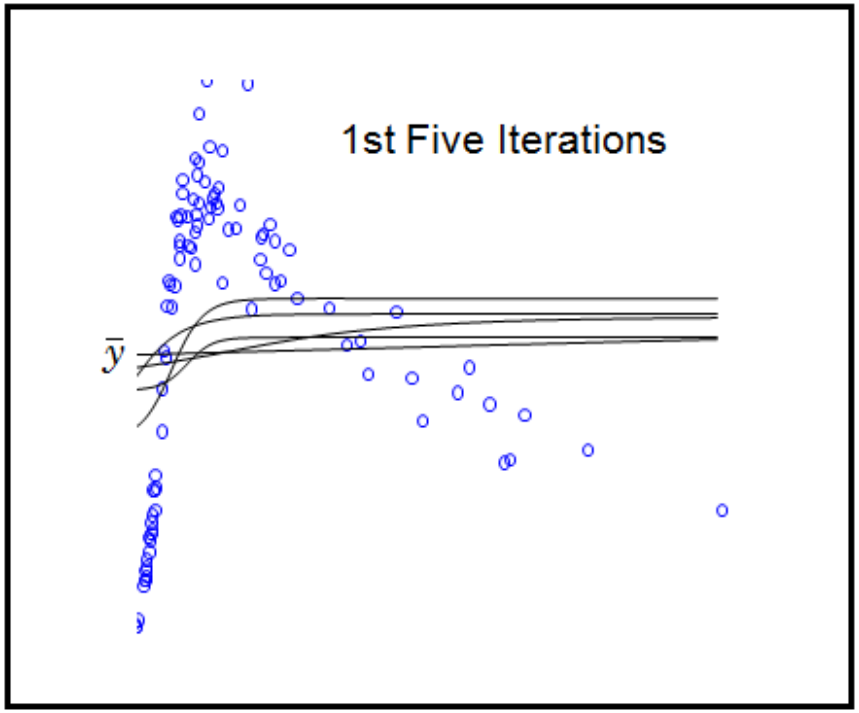
$$w_0 + w_1 \exp(-w_{01}^2 (x - w_{11})^2)$$

Ранняя остановка – борьба с переобучением

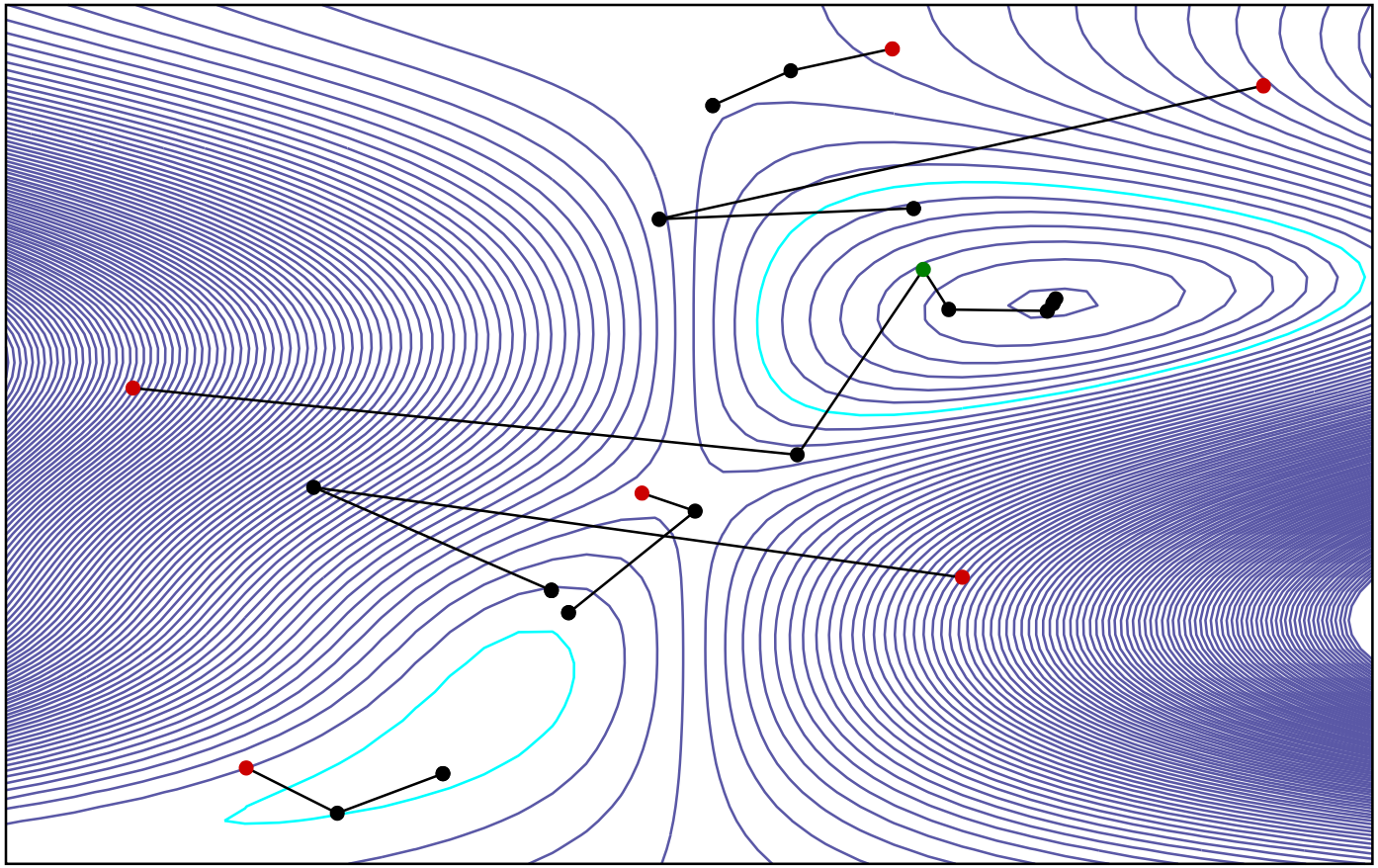


Инициализация

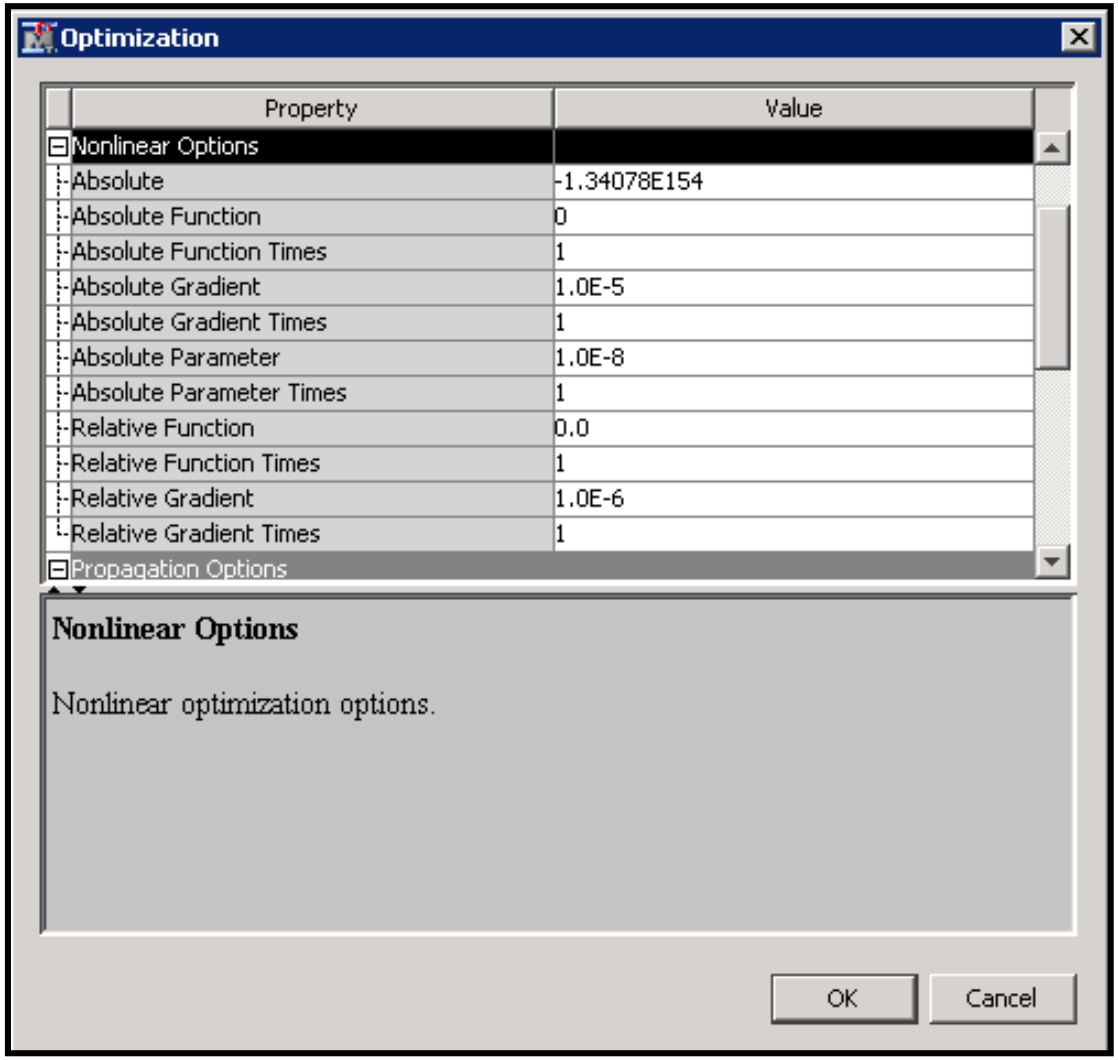




Предварительное обучение

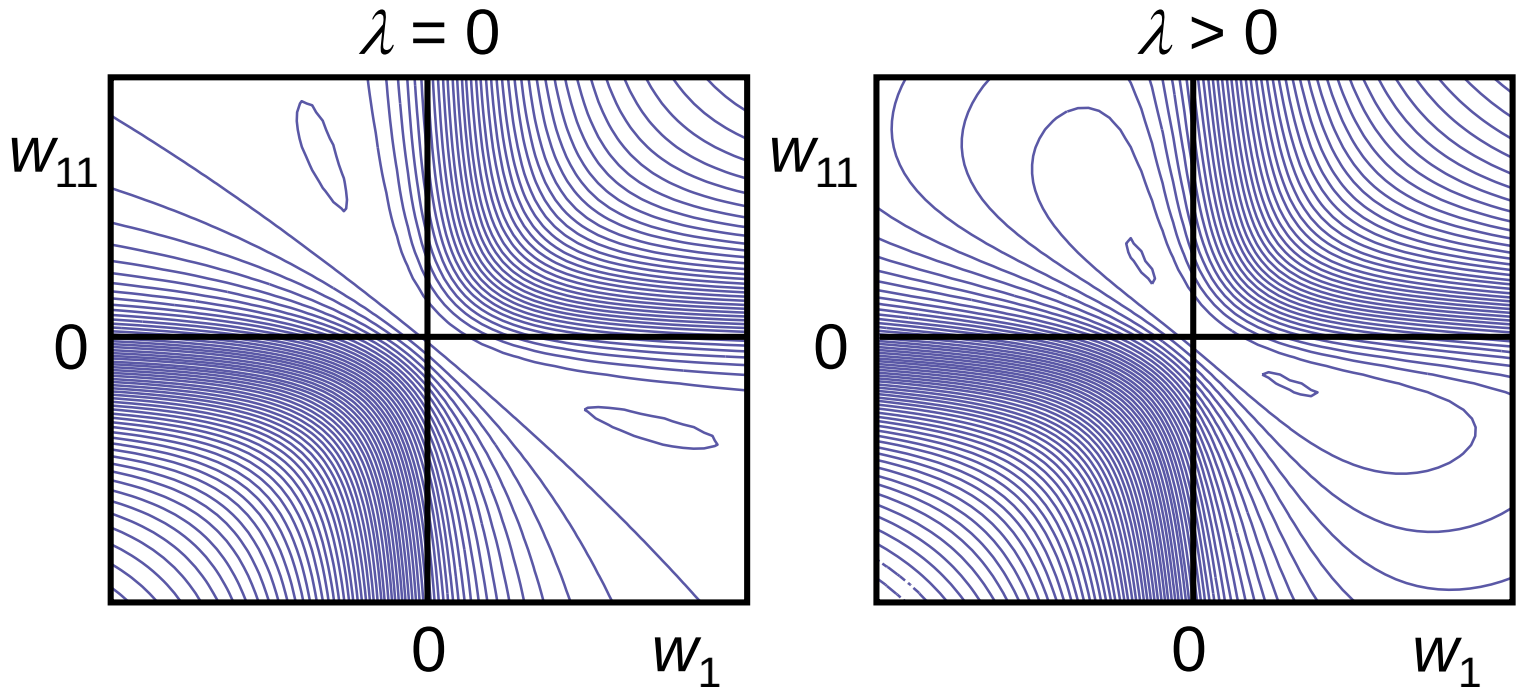


Критерии схоимости



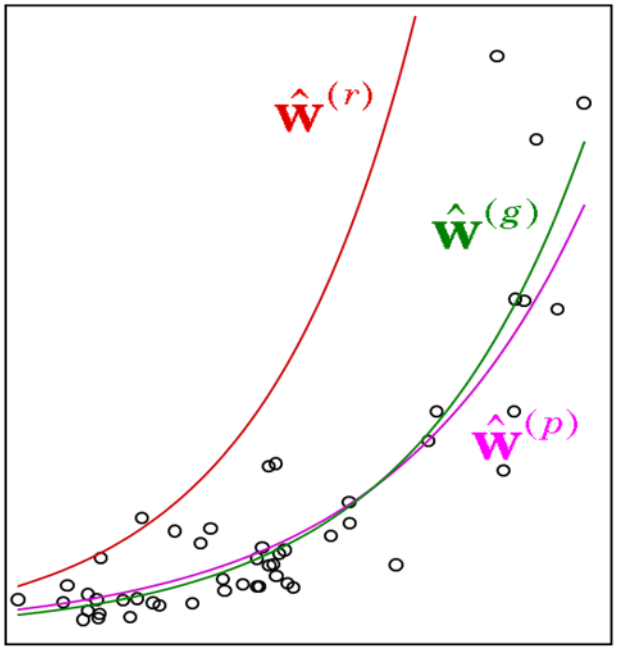
Регуляризация

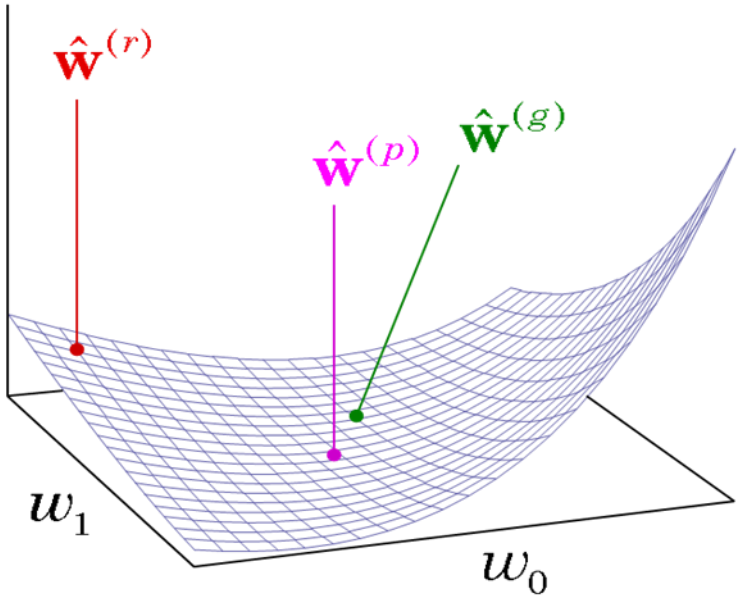
$$\text{Objective Function} = \text{Error Function} + \lambda \|\mathbf{w}\|^2$$



Оценки максимального правдоподобия

$$Q(\mathbf{w}) = 0.5\left[\left(\frac{y-\mu(\mathbf{w})}{\vartheta}\right)^2 + \ln(2\pi)\right] + \ln(\vartheta)$$





Оценка отклонения

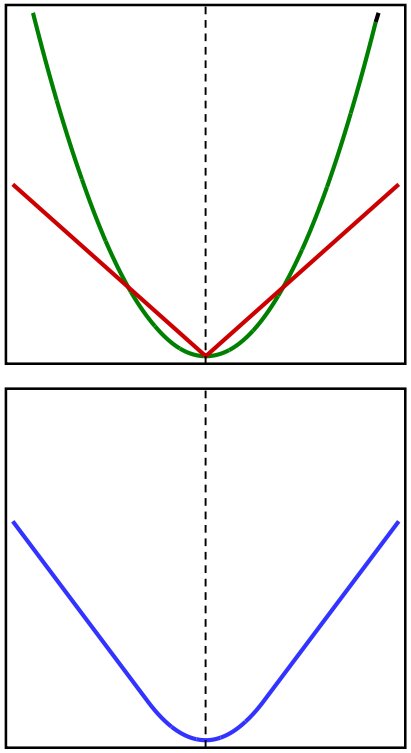
Поиск параметров модели

- решается задача оптимизации
- max loglik с заданным распределением и функцией связи

Распределение	**Отклонение**
Normal	$Q(\mathbf{w}) = \sum (y - \mu(\mathbf{w}))^2$
Poisson	$Q(\mathbf{w}) = 2\sum \left[y \ln(y / \mu(\mathbf{w})) - (y - \mu(\mathbf{w})) \right]$
Gamma	$Q(\mathbf{w}) = 2\sum \left[-\ln(y / \mu(\mathbf{w})) + (y - \mu(\mathbf{w})) / \mu(\mathbf{w}) \right]$
Bernoulli	$Q(\mathbf{w}) = -2\sum \left[y \ln(\mu(\mathbf{w})) + (1 - y) \ln(1 - \mu(\mathbf{w})) \right]$

Робастные оценки

$$Q(\mathbf{w}) = \sum_{i=1}^{n} \psi\left(\frac{y_i - \mu_i(\mathbf{w})}{\sigma_y}\right) = \sum_{i=1}^{n} \psi(z_i)$$



$\psi(z) = 0.5z^2$ Normal

$\psi(z) = |z|$ Laplace

$\psi(z) = 0.5z^2$ if $|z| < 1$

$\psi(z) = |z| - 0.5$ if $|z| \geq 1$

Huber's

Комбинации функций активации и распределения ошибок

Отклик	Функция связи	Функуция активации	Распределение ошибок
Числа	Identity	Identity	Normal
	Identity	Identity	Huber
	Log	Exponential	Poisson
	Log	Exponential	Gamma
Категории и порядки	Logit	Logistic	Bernoulli
	Generalized Logit	Softmax	MBernoulli
	Cumulative Logit	Logistic (See note.)	MBernoulli
Пропорции	Logit	Logistic	Entropy
	Generalized Logit	Softmax	MEntropy

- Обратная кумулятивная logit называется *Logistic*.

Постановка задачи оптимизации

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{aligned}$$

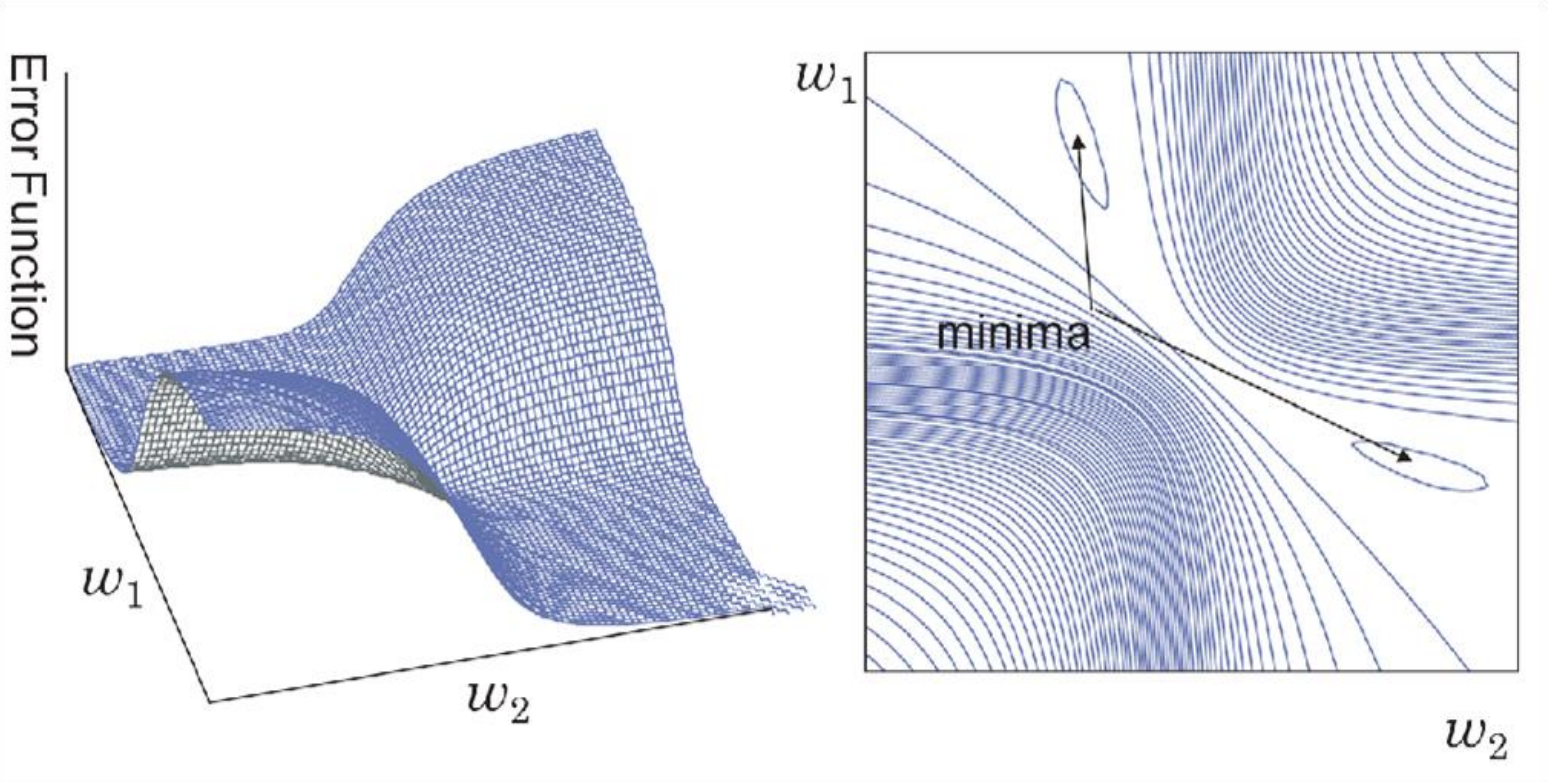
- Методы первого порядка – градиентные (используют шаг «вдоль» направления градиента – вектора первых производных)
  - выбор шага (константа, дробный выбор, адаптивный, наискорейший)
  - выбор напраления (с учетом предыдущих шагов, например сопряженные градиенты)

$$y = f(\mathbf{x} + \Delta\mathbf{x}) \approx f(\mathbf{x}) + J(\mathbf{x})\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^{\mathrm{T}} H(\mathbf{x})\Delta\mathbf{x}$$

- Методы второго порядка – ньютоновские (используют матрицу вторых производных Гессе для «выбора шага»)
  - проблема – вычисление обратной матрицы Гессе на каждом шаге

Итерационные методы

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \delta^{(t)}$$



Градиентный:

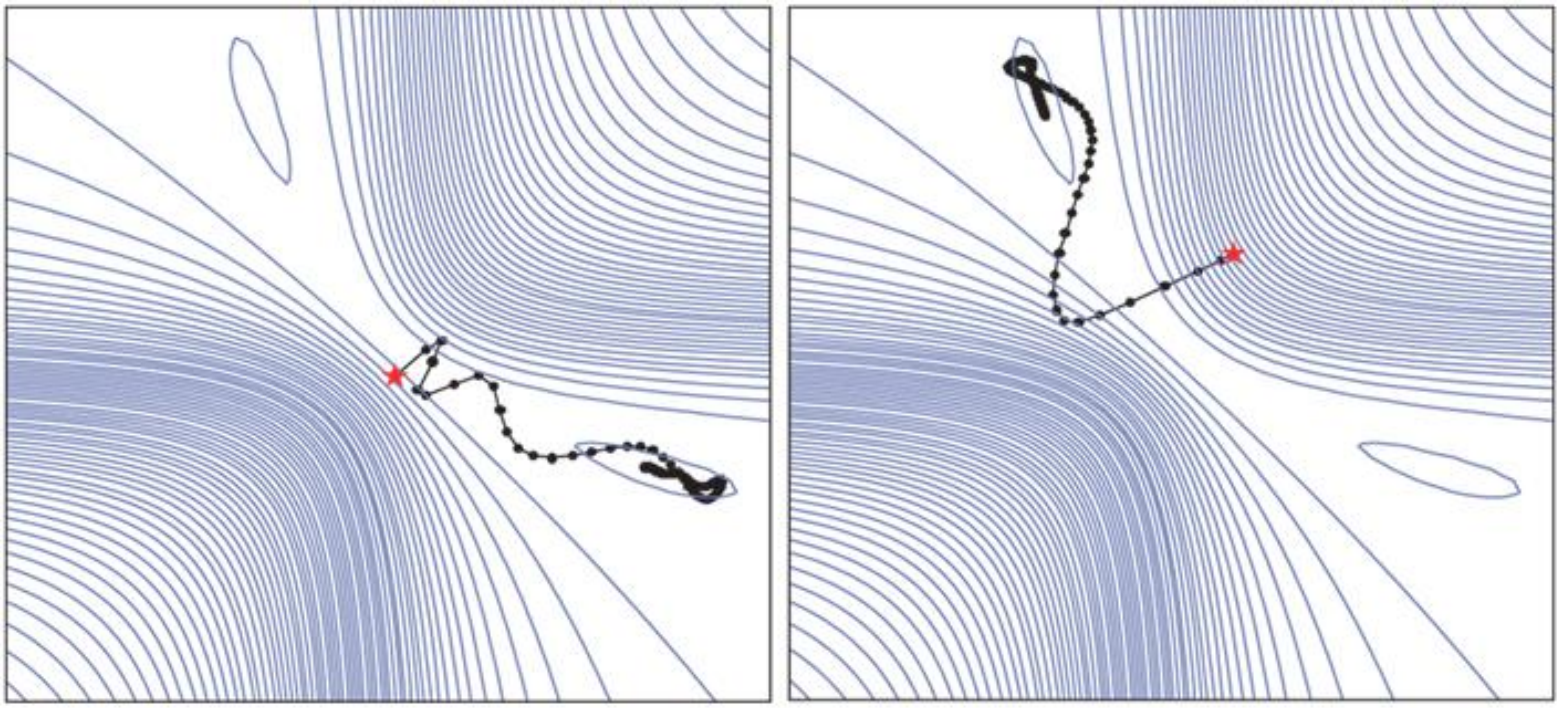
$$\delta^{(t)} = -\eta \nabla \mathrm{g}^{(t)}$$

Ньютона:

$$\delta^{(t)} = -[\mathrm{H}^{(t)}]^{-1} \nabla \mathrm{g}^{(t)}$$

Обратное распространение ошибки (градиентный метод)

$$\delta^{(t)} = -\eta \nabla g^{(t)} + \alpha \delta^{(t-1)}$$



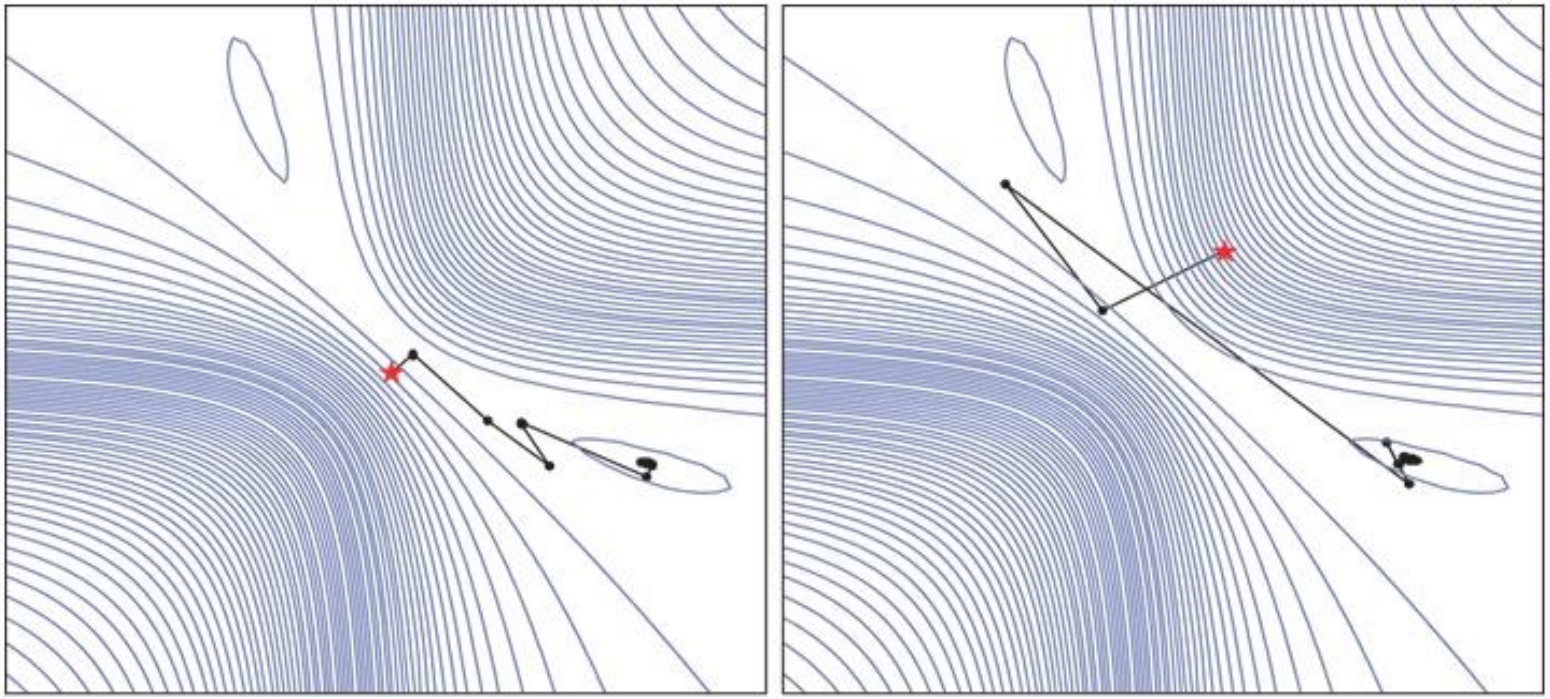
87 iterations
(η = 0.5, α = 0.9)

285 iterations
(η = 0.1, α = 0.9)

Недостатки: долго, тяжело «угадать» параметры

Быстрое обратное распространение ошибки

$$\delta^{(t)} = -[diag(\tilde{\mathrm{H}}^{(t-1)})]^{-1}\nabla \mathrm{g}^{(t)}$$



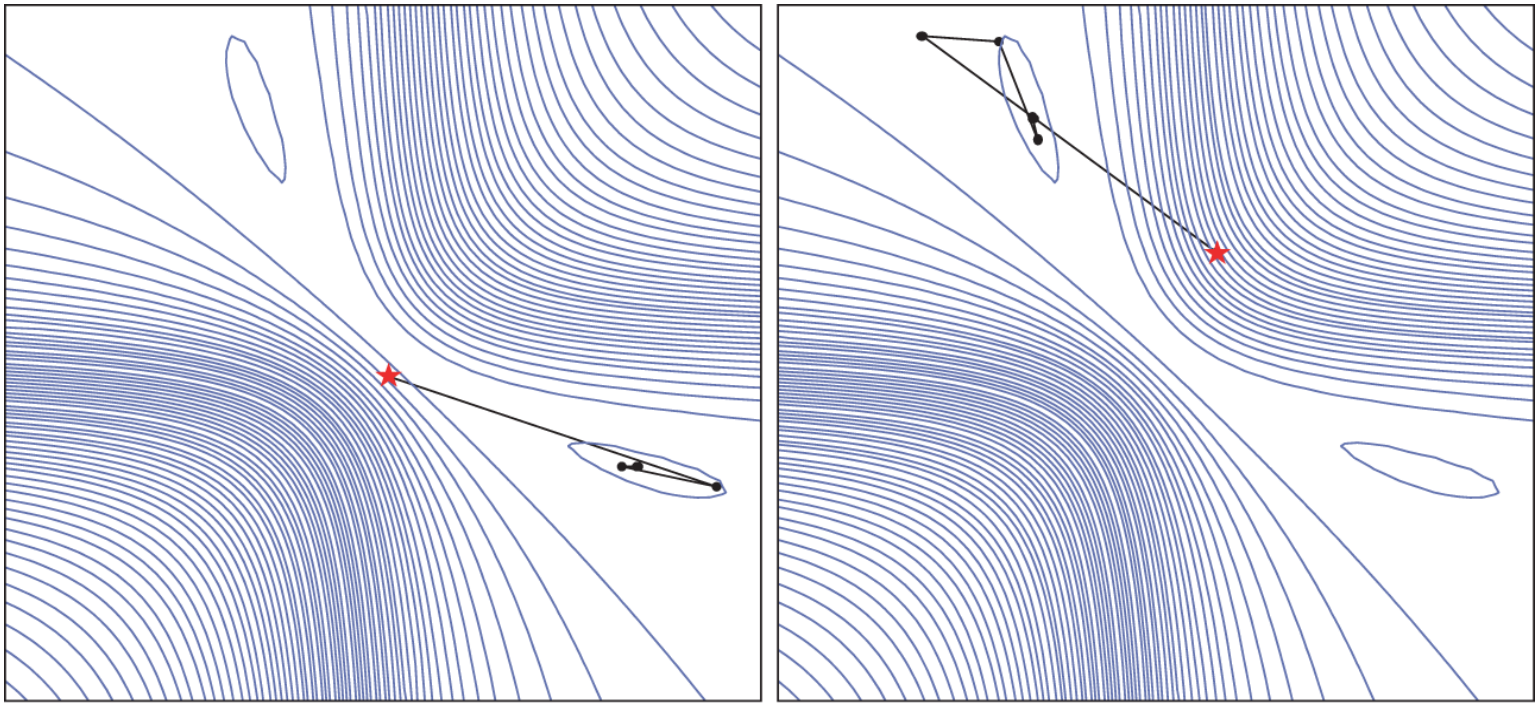
38 iterations

57 iterations

Приближаем функцию ошибки «параболой», вычисляем диагональ Гессиана «приближенной» функции

Левенберга — Марквардта

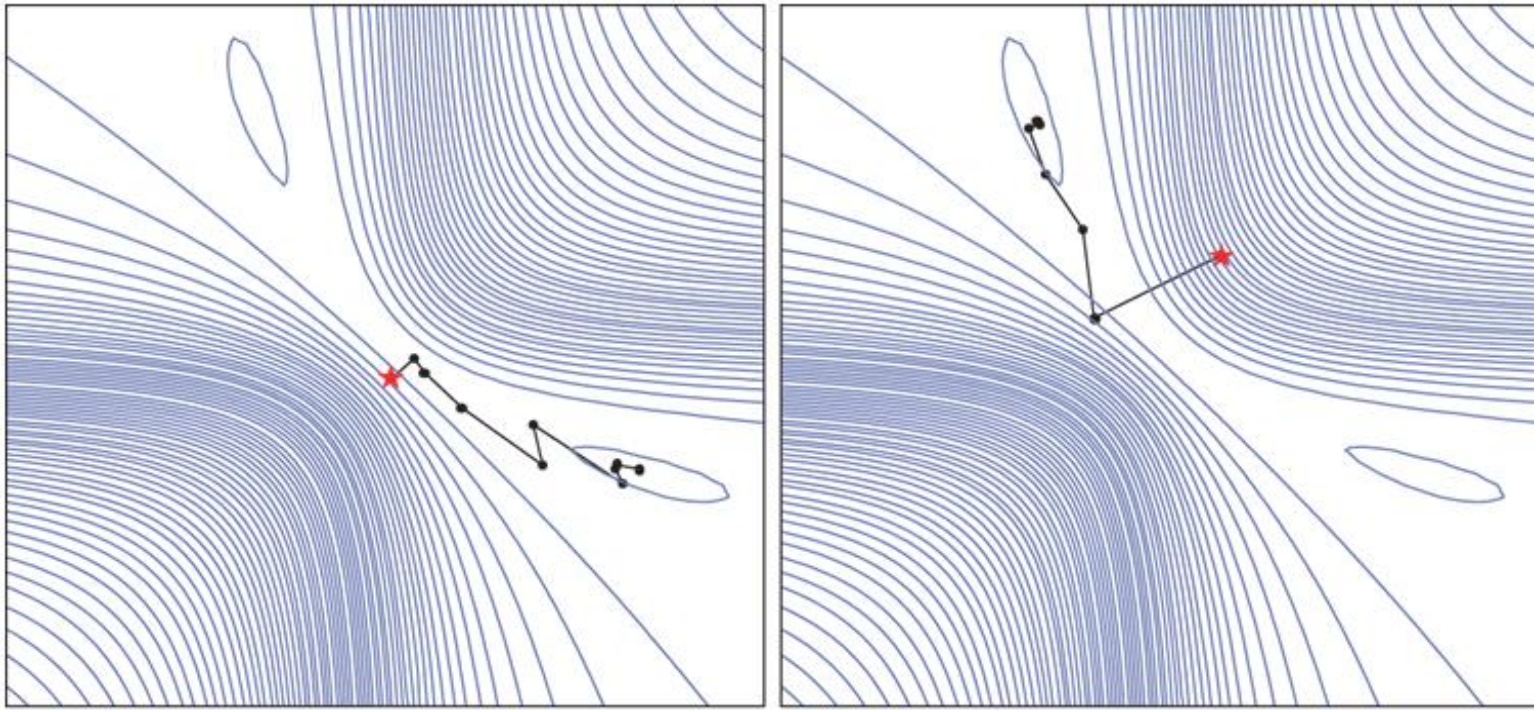
$$\delta^{(t)} = -(\mathrm{J}^{(t)\prime}\mathrm{J}^{(t)} + \lambda^{(t)}\mathrm{I})^{-1}\mathrm{J}^{(t)\prime}\mathrm{r}^{(t)}$$



Комбинация градиентного (лямбда велико) и Ньютона (лямбда=0),
Применим для небольшого количества переменных <100

Квазиньютоновские методы

$$\delta^{(t)} = -\eta^{(t)}[\mathrm{B}^{(t-1)} + \mathrm{E}^{(t-1)}]^{-1}\nabla g^{(t)}$$



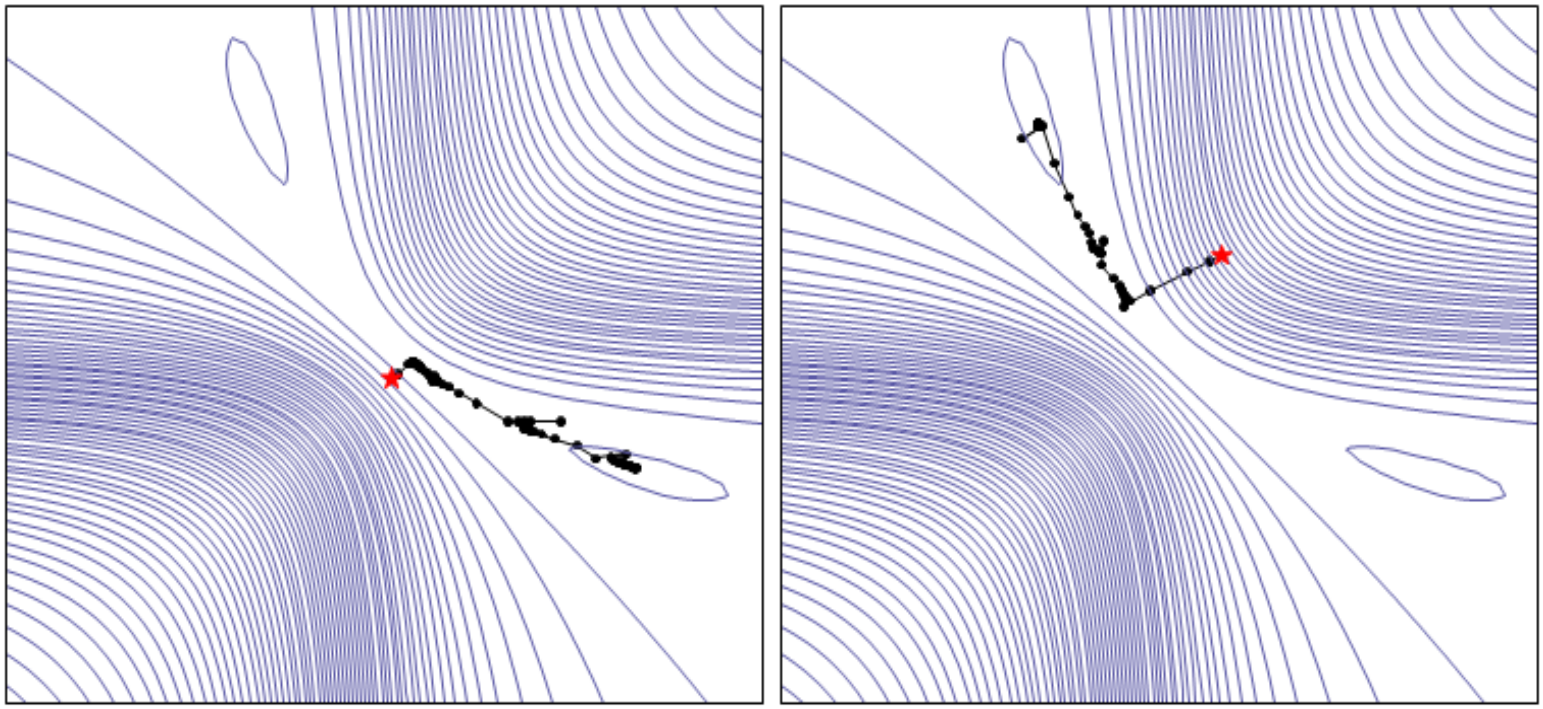
11 iterations

8 iterations

Приближаем H как сумму B и E, обычно E – единичная
Применим для среднего размера задач <500 переменных

Метод Сопряженных градиентов

$$\delta^{(t)} = -\eta^{(t)}[-\nabla g^{(t)} + \beta^{(t-1)}\delta^{(t-1)}]$$



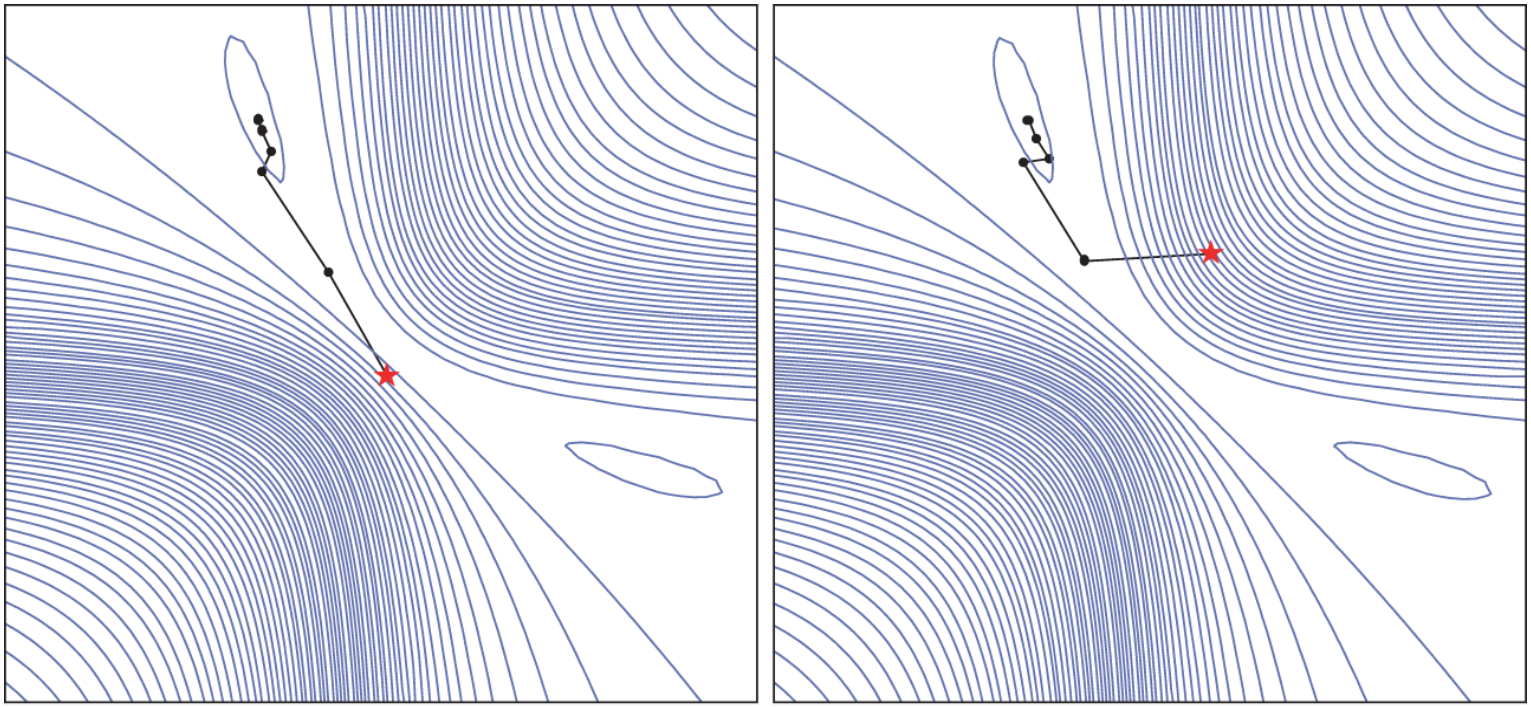
66 iterations

45 iterations

Выбор следующего направления как сопряженного (относительно матрицы Гессе) к предыдущим направлениям шага. Позволяет не рассчитывать H на каждом шаге и работает с большими задачами.

Метод доверительных областей (trusted regions)

$$\delta^{(t)} = -(\mathrm{H}^{(t)} + \lambda^{(t)}\mathrm{I})^{-1}\nabla \mathrm{g}^{(t)}$$



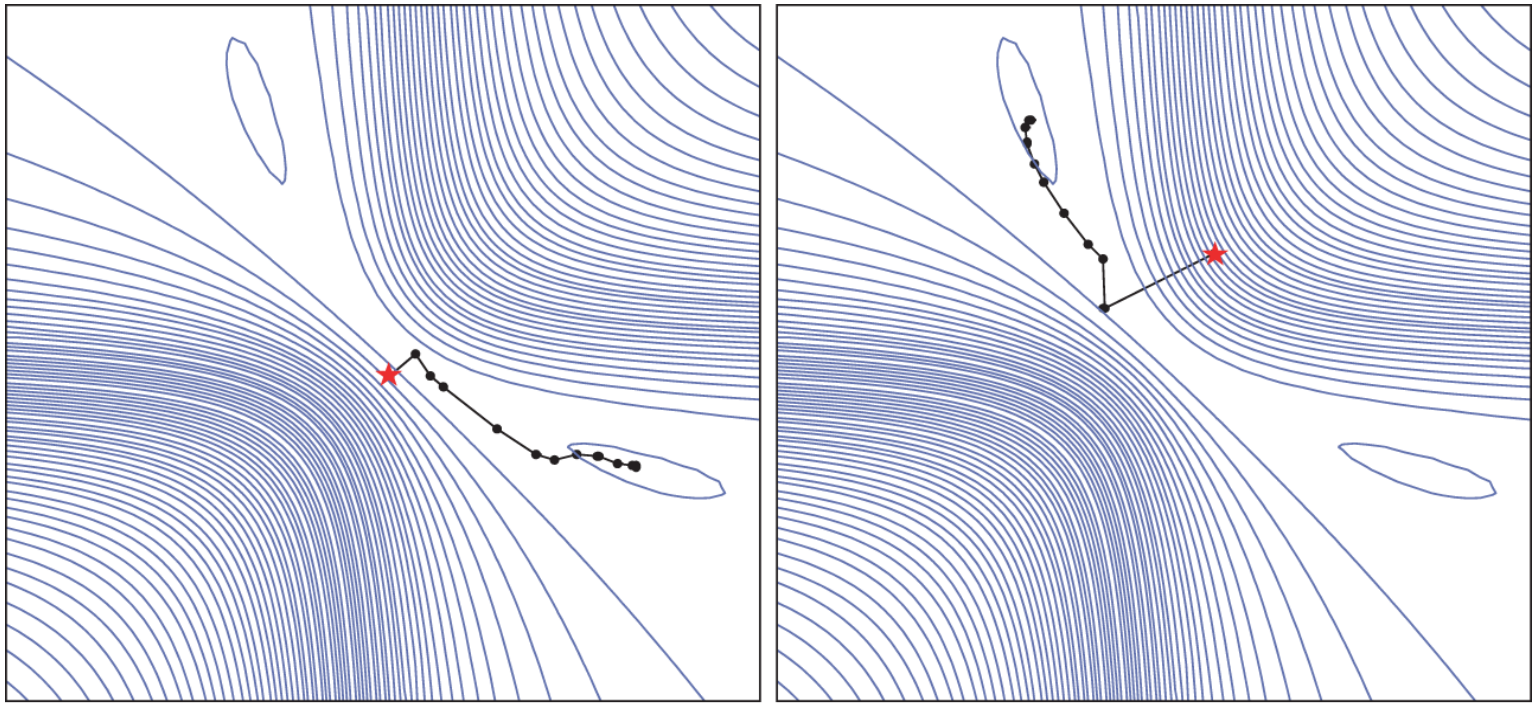
6 iterations

6 iterations

Работает для небольших задач <40, но зато с сильно «не квадратичными» целевыми функциями

Комбинированный (градиент+ньютон) Double-Dogleg

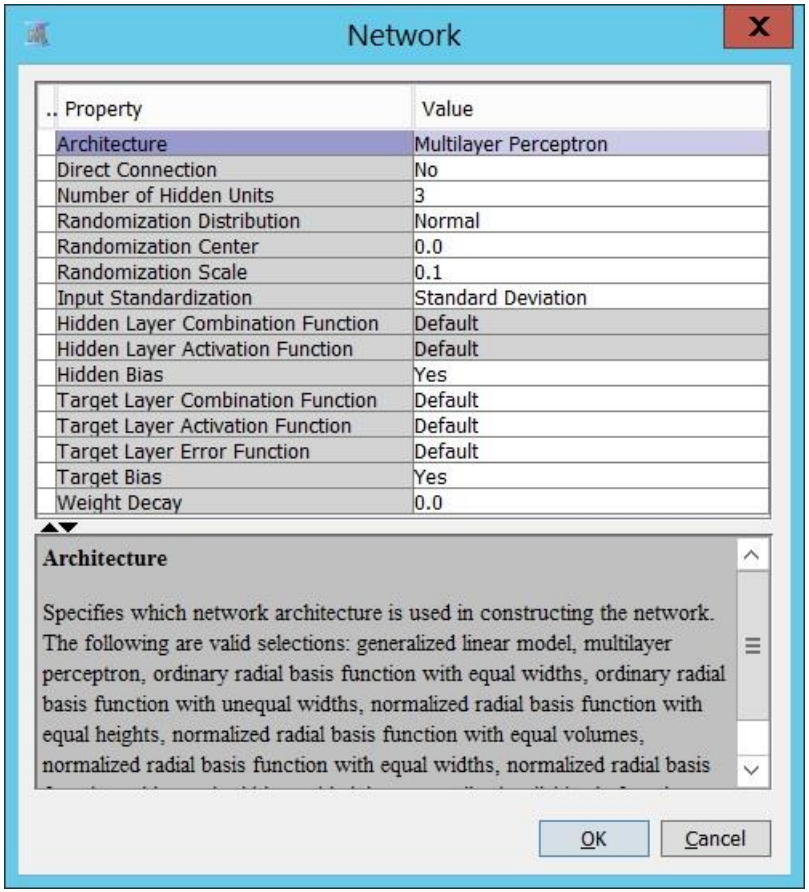
$$\delta^{(t)} = -\alpha_1 s^{(t)}_{Steepest\ Descent} + \alpha_2 s^{(t)}_{Quasi\ Newton}$$

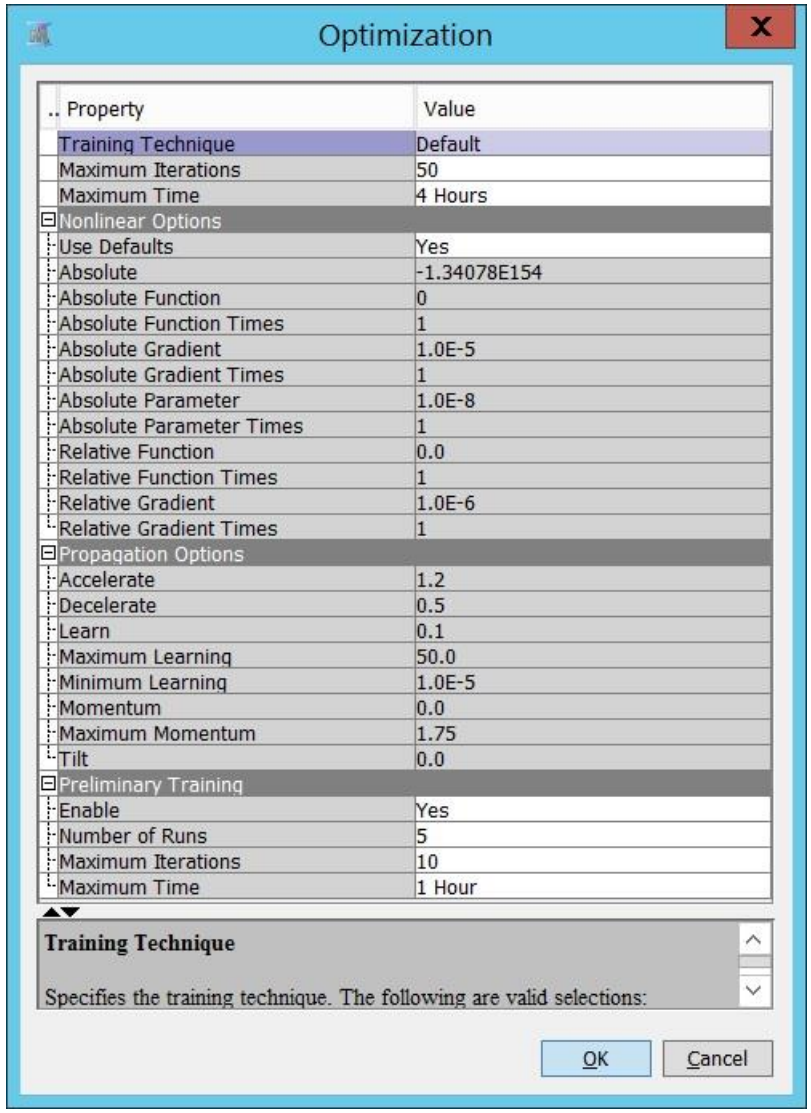


12 iterations

11 iterations

Основные настройки нейронной сети





Процедура NEURAL

```
PROC NEURAL DATA=<SAS-data-set> DMDBCAT=<catalog>;
      INPUT <variable(s)> / LEVEL=<level> ID=<name>;
      HIDDEN <integer> / ID=<name>;
      TARGET <targets> / LEVEL=<level> ID=<name>;
      CONNECT <name> … <name>;
      PRELIM <integer> MAXITER=<integer>;
      TRAIN;
RUN;
```

- PROC NEURAL позволяет строить сети прямого распространения пользовательской архитектуры

Задание архитектуры

```
PROC NEURAL DATA=<SAS data set> DMDBCAT=<catalog>;
      INPUT <inputs> / LEVEL=<level>;
      TARGET <targets> / LEVEL=<level>;
      ARCHI <architecture-name> <HIDDEN=integer> <DIRECT>;
      PRELIM <integer> MAXITER=<integer>;
      TRAIN;
RUN;
```

```
PROC NEURAL DATA=<SAS data set> DMDBCAT=<catalog>;
      INPUT <inputs> / LEVEL=<level> ID=<name>;
      HIDDEN <integer> / ID=<name>;
      TARGET <targets> / LEVEL=<level> ID=<name>;
      CONNECT <ID-list>;
      PRELIM <integer> MAXITER=<integer>;
      TRAIN;
RUN;
```

Глубинное обучение

Глубинное обучение (англ. Deep learning) — набор алгоритмов машинного обучения, которые пытаются моделировать высокоуровневые абстракции в данных, используя архитектуры, состоящие из множества нелинейных трансформаций //Wikipedia

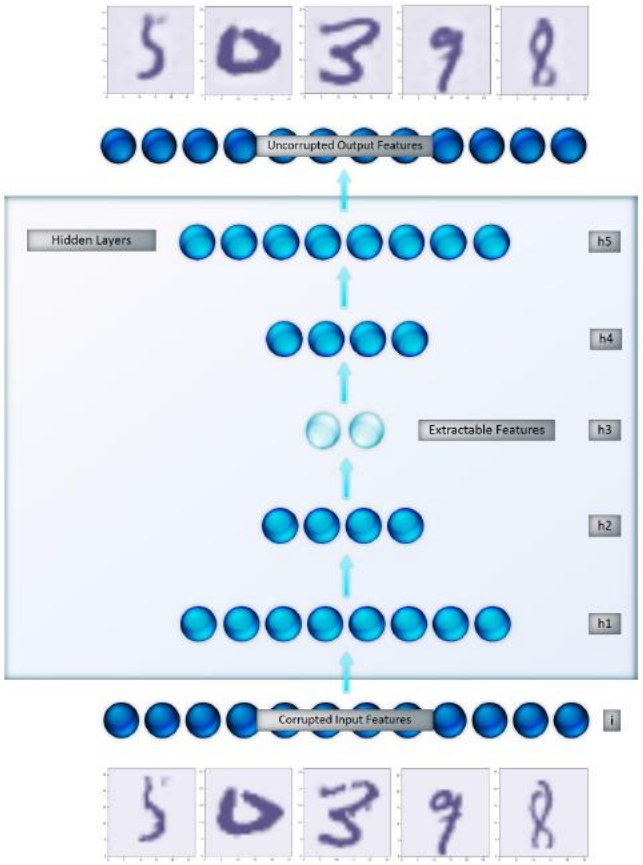
Обычно:

- либо **многослойниые нейронные сети**, где часть уровней отвечает за выявление признаков (*unsupervised режим*), часть за прогнозирование (*supervised режим*)
- либо **kernel методы** (будут далее), также включают как структуры выявления признаков (например, kernel PCA, *unsupervised* режим) так и структуры для прогнозирования (SVM на основе найденных нелинейных главных компонент, *supervised режим*)

Одна из ключевых особенностей – требуется поэтапное обучение в несколько «проходов» с «заморозкой» коэфициентов части слоев или структур.

В процедуре neural есть операторы **freeze** и **thaw**!!!

Пример: репликаторные нейронные сети




```
proc neural
       data= autoencoderTraining
       dmdbcat= work.autoencoderTrainingCat;
       performance compile details cpucount= 12 threads= yes;

       /* DEFAULTS: ACT= TANH COMBINE= LINEAR */
       /* IDS ARE USED AS LAYER INDICATORS ╤ SEE FIGURE 6 */
       /* INPUTS AND TARGETS SHOULD BE STANDARDIZED */
       archi MLP hidden= 5;
       hidden 300 / id= h1;
       hidden 100 / id= h2;
       hidden 2 / id= h3 act= linear;
       hidden 100 / id= h4;
       hidden 300 / id= h5;
       input corruptedPixel1-corruptedPixel400 / id= i level= int std= std;
       target pixel1-pixel400 / act= identity id= t level= int std= std;

       /* BEFORE PRELIMINARY TRAINING WEIGHTS WILL BE RANDOM */
       initial random= 123;
       prelim 10 preiter= 10;

       /* TRAIN LAYERS SEPARATELY */
       freeze h1->h2;
       freeze h2->h3;
       freeze h3->h4;
       freeze h4->h5;
       train technique= congra maxtime= 129600 maxiter= 1000;

       freeze i->h1;
       thaw h1->h2;
       train technique= congra maxtime= 129600 maxiter= 1000;

       freeze h1->h2;
       thaw h2->h3;
       train technique= congra maxtime= 129600 maxiter= 1000;

       freeze h2->h3;
       thaw h3->h4;
       train technique= congra maxtime= 129600 maxiter= 1000;

       freeze h3->h4;
       thaw h4->h5;
       train technique= congra maxtime= 129600 maxiter= 1000;

       /* RETRAIN ALL LAYERS SIMULTANEOUSLY */
       thaw i->h1;
       thaw h1->h2;
       thaw h2->h3;
       thaw h3->h4;
       train technique= congra maxtime= 129600 maxiter= 1000;

       code file= 'C:\Path\to\code.sas';
run;
```