



Лекция 8: нелинейные модели: нейронные сети

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**

Подготовка данных для примера

```
df = pd.read_csv(file_path, index_col=0)
df.head()
```

year	age	maritl	race	education	region	jobclass	health	health_ins	logwage	wage
2006	18	1. Never Married	1. White	1. < HS Grad	2. Middle Atlantic	1. Industrial	1. <=Good	2. No	4.318063	75.043154
2004	24	1. Never Married	1. White	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	2. No	4.255273	70.476020
2003	45	2. Married	1. White	3. Some College	2. Middle Atlantic	1. Industrial	1. <=Good	1. Yes	4.875061	130.982177
2003	43	2. Married	3. Asian	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	1. Yes	5.041393	154.685293
2005	50	4. Divorced	1. White	2. HS Grad	2. Middle Atlantic	2. Information	1. <=Good	1. Yes	4.318063	75.043154

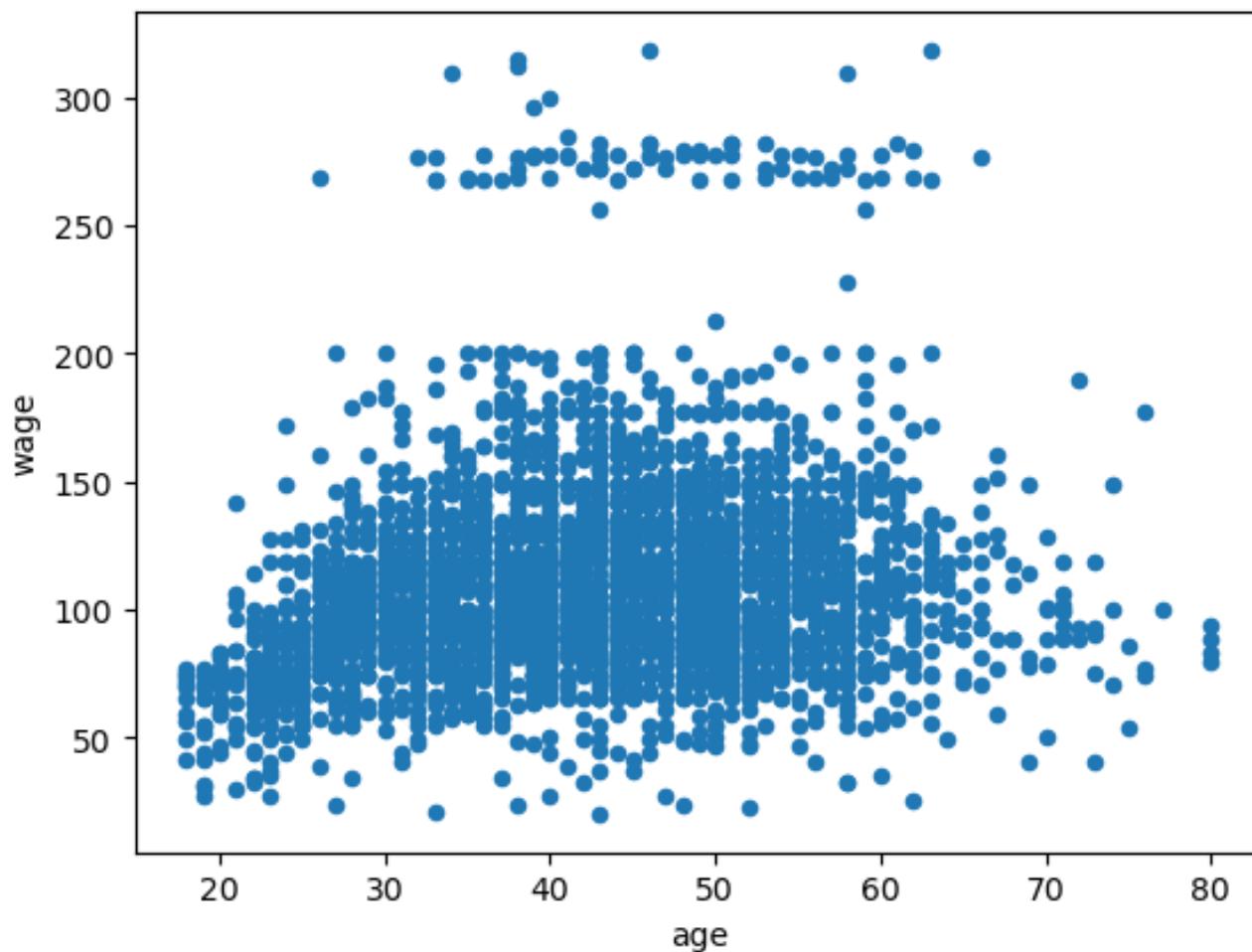
```
str_feats = ['maritl', 'race', 'education',
             'region', 'jobclass',
             'health', 'health_ins']

map_dict = {} 
for s_f in str_feats:
    l = df[s_f].unique()
    map_dict[s_f] = {e: int(e.split('.')[0])
                    for e in l}
    df[s_f] = df[s_f].map(map_dict[s_f])
```

```
{'maritl': {'1. Never Married': 1,
            '2. Married': 2,
            '4. Divorced': 4,
            '3. Widowed': 3,
            '5. Separated': 5},
 'race': {'1. White': 1, '3. Asian': 3, '4. Other': 4, '2. Black': 2},
 'education': {'1. < HS Grad': 1,
               '4. College Grad': 4,
               '3. Some College': 3,
               '2. HS Grad': 2,
               '5. Advanced Degree': 5},
 'region': {'2. Middle Atlantic': 2},
 'jobclass': {'1. Industrial': 1, '2. Information': 2},
 'health': {'1. <=Good': 1, '2. >=Very Good': 2},
 'health_ins': {'2. No': 2, '1. Yes': 1}}
```

Подготовка данных для примера

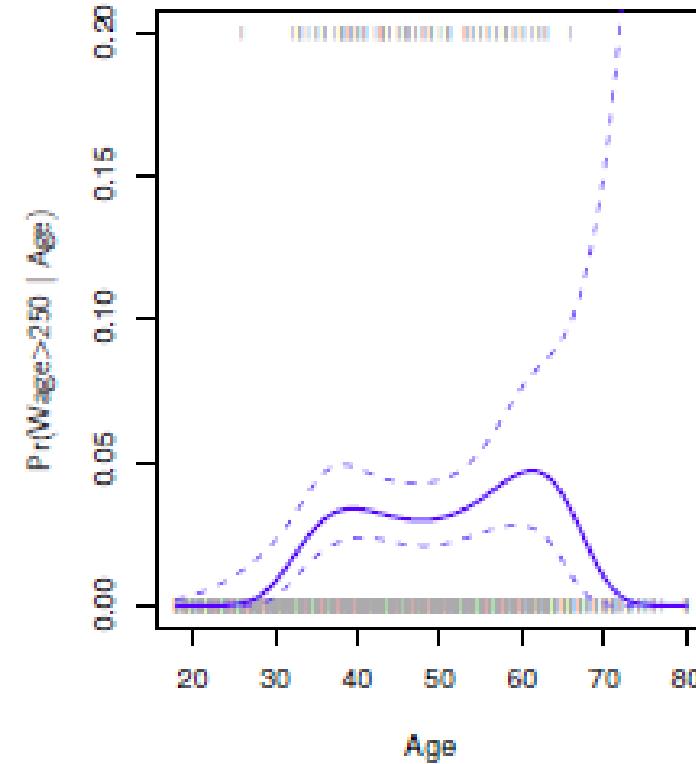
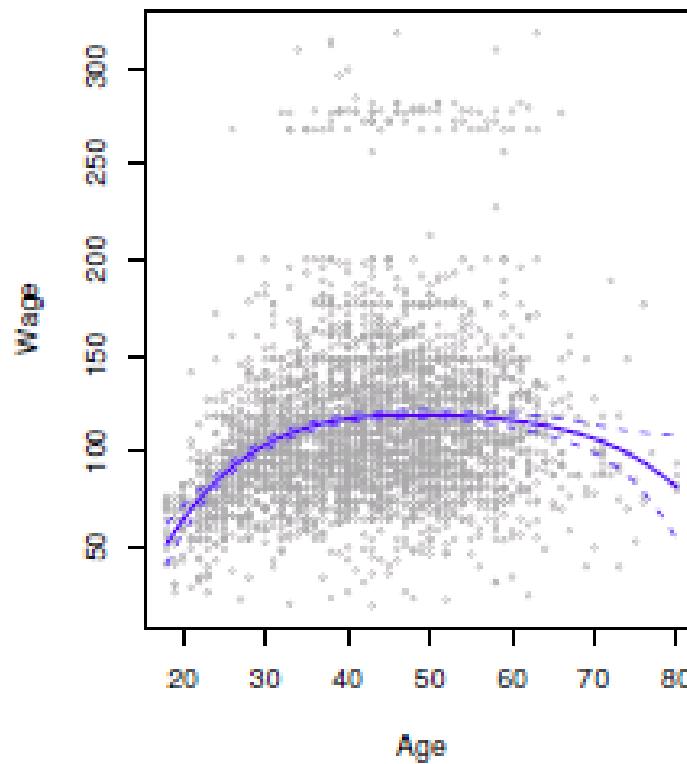
```
df.plot.scatter(x="age", y="wage")
```



Полиномиальная регрессия

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

Degree-4 Polynomial

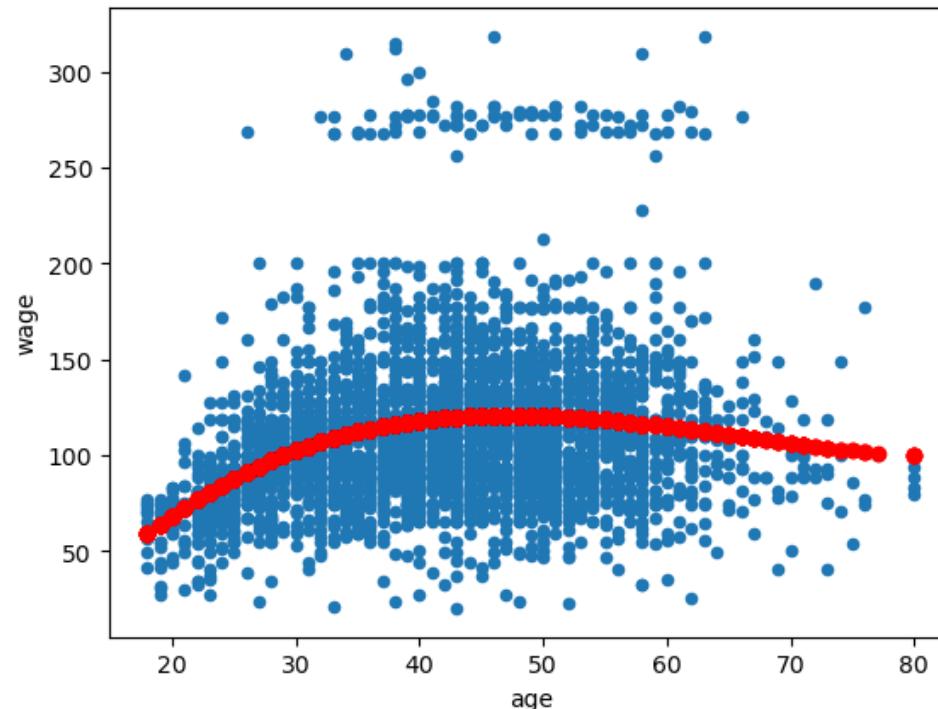


Пример использования

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline

model = Pipeline([('poly', PolynomialFeatures(degree=3)),
                  ('linear', LinearRegression(fit_intercept=False))])

model.fit(X, y)
```



Пример использования

```
!pip install mlinsights
```

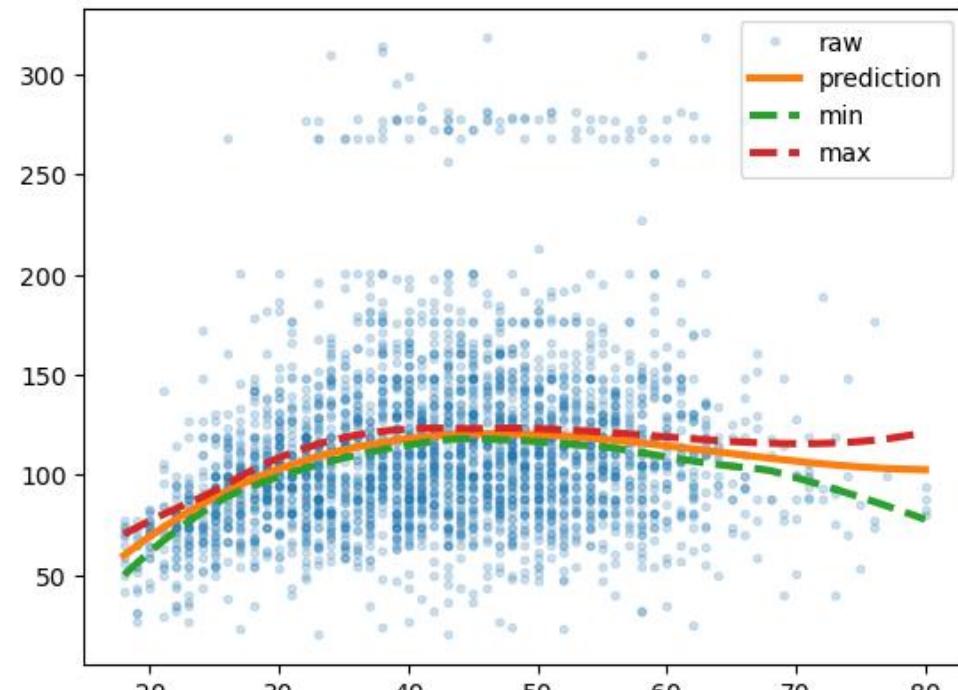
```
from mlinsights.mlmodel import IntervalRegressor
from sklearn.linear_model import LinearRegression

poly_model = PolynomialFeatures(degree=3)
X_p = poly_model.fit_transform(X)
model = LinearRegression(fit_intercept=False)

lin = IntervalRegressor(model,
                       alpha=0.2)
lin.fit(X_p, y.to_numpy())

sorted_X = X.sort_values(by="age").to_numpy()
sorted_X_p = poly_model.fit_transform(sorted_X)

pred = lin.predict(sorted_X_p)
pred_sort = lin.predict_sorted(sorted_X_p)
min_pred = pred_sort[:, 0]
max_pred = pred_sort[:, pred_sort.shape[1]-1]
```



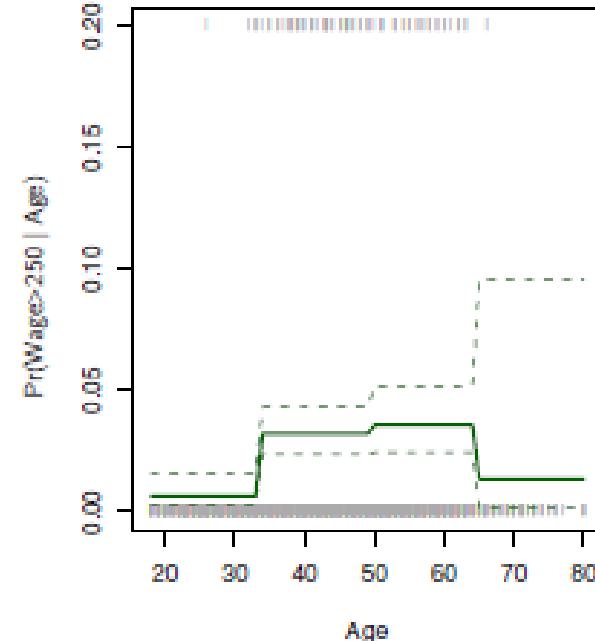
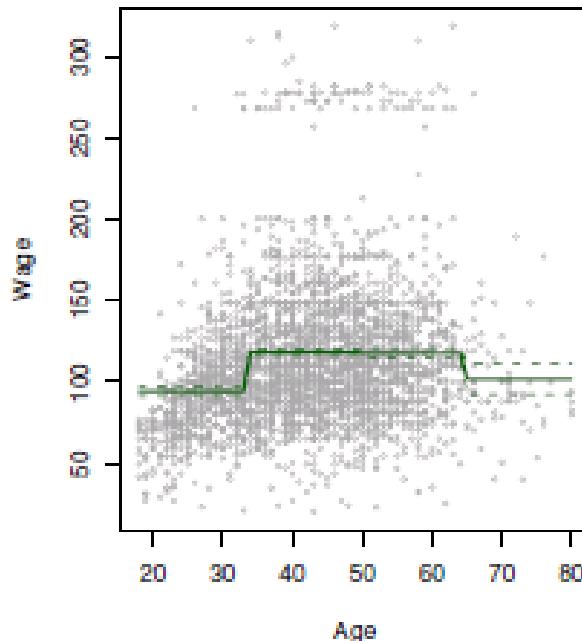
```
ax.plot(X, y, '.', label="raw", alpha=0.2)
ax.plot(sorted_X, pred, label="prediction", linewidth=3)
ax.plot(sorted_X, min_pred, '--', label="min", linewidth=3)
ax.plot(sorted_X, max_pred, '--', label="max", linewidth=3)
```

Ступенчатые функции

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

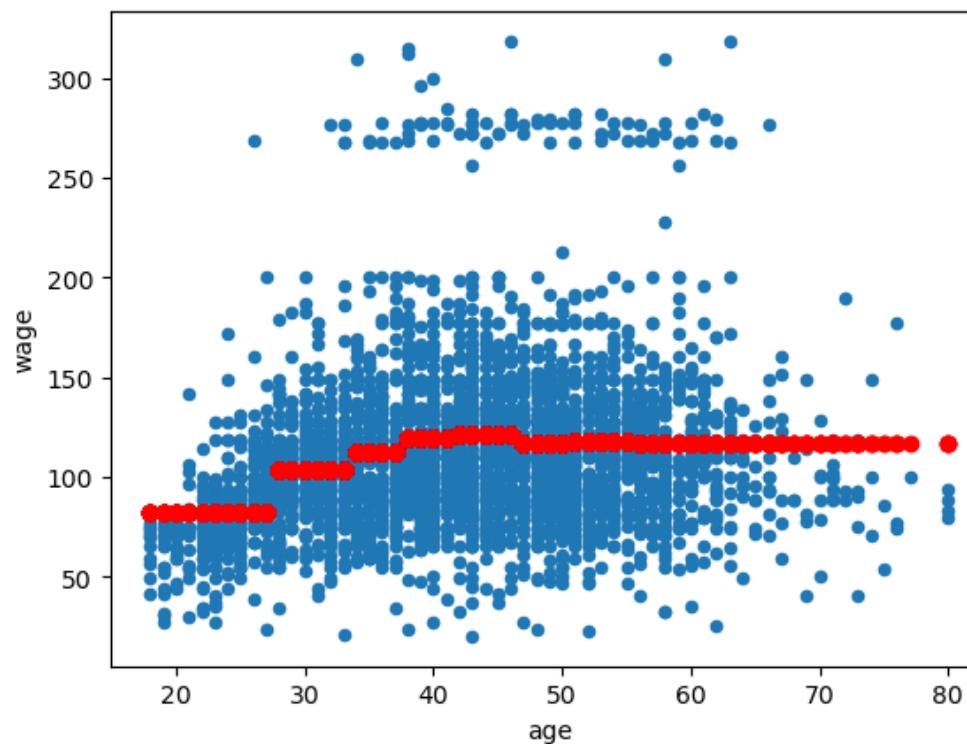
$$C_1(X) = I(X < 35), \quad C_2(X) = I(35 \leq X < 50), \dots, C_3(X) = I(X \geq 65)$$

Piecewise Constant



Пример использования

```
from mlinsights.mlmodel import PiecewiseRegressor  
from sklearn.dummy import DummyRegressor  
from sklearn.preprocessing import KBinsDiscretizer  
  
model = PiecewiseRegressor(estimator=DummyRegressor(),  
                           binner=KBinsDiscretizer(n_bins=8))  
model.fit(X, y)
```



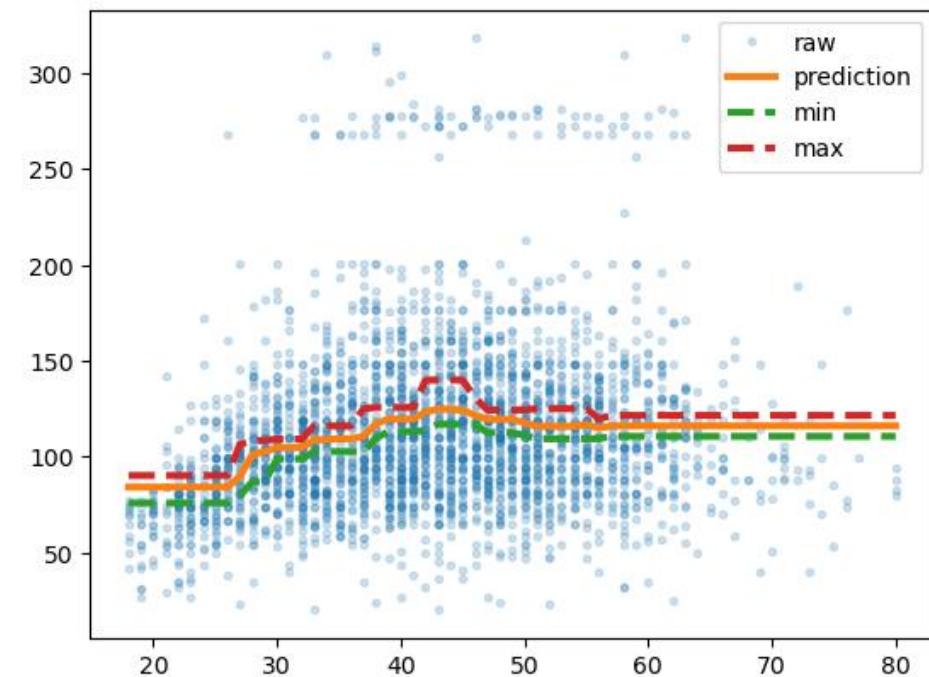
Пример использования

```
from mlinsights.mlmodel import IntervalRegressor

model = PiecewiseRegressor(
    estimator=DummyRegressor(),
    binner=KBinsDiscretizer(n_bins=8)
)

lin = IntervalRegressor(model, alpha=0.2)
lin.fit(X.to_numpy(), y.to_numpy())

sorted_X = X.sort_values(by="age").to_numpy()
pred = lin.predict(sorted_X)
pred_sort = lin.predict_sorted(sorted_X)
min_pred = pred_sort[:, 0]
max_pred = pred_sort[:, pred_sort.shape[1]-1]
```



```
ax.plot(X, y, '.', label="raw", alpha=0.2)
ax.plot(sorted_X, pred, label="prediction", linewidth=3)
ax.plot(sorted_X, min_pred, '--', label="min", linewidth=3)
ax.plot(sorted_X, max_pred, '--', label="max", linewidth=3)
```

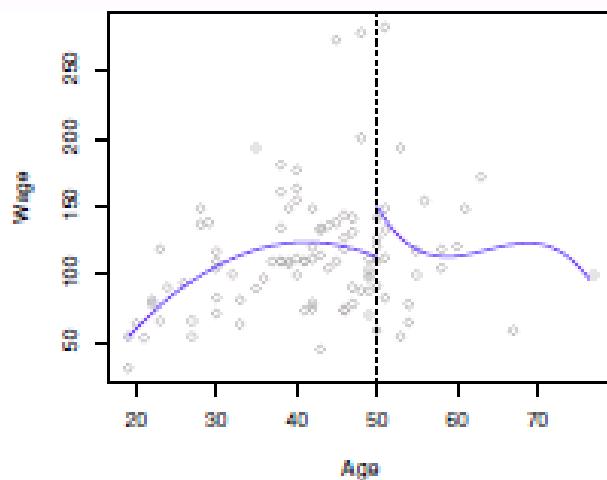
Кусочные полиномы

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

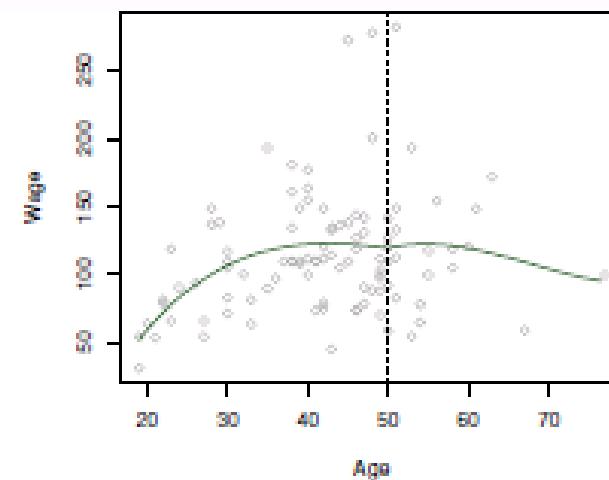
$$y_i = \begin{cases} \beta_{01} + \beta_{11}x_i + \beta_{21}x_i^2 + \beta_{31}x_i^3 + \epsilon_i & \text{if } x_i < c; \\ \beta_{02} + \beta_{12}x_i + \beta_{22}x_i^2 + \beta_{32}x_i^3 + \epsilon_i & \text{if } x_i \geq c. \end{cases}$$

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

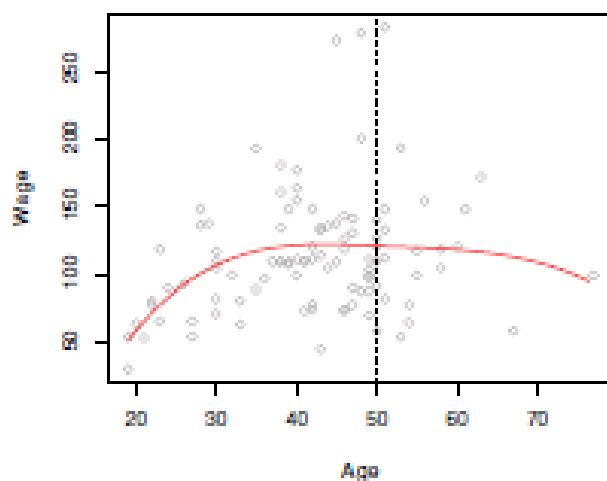
Piecewise Cubic



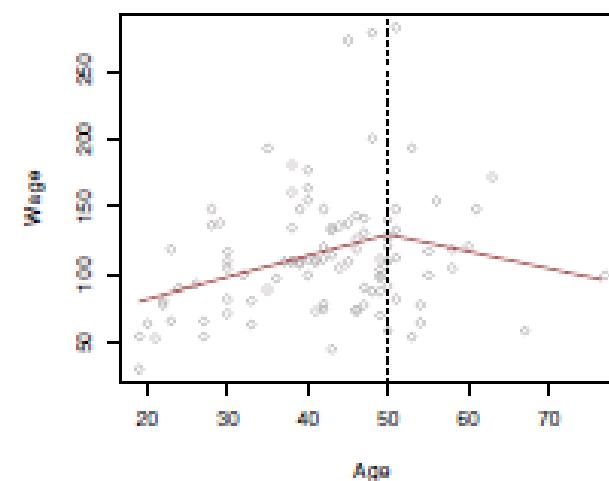
Continuous Piecewise Cubic



Cubic Spline



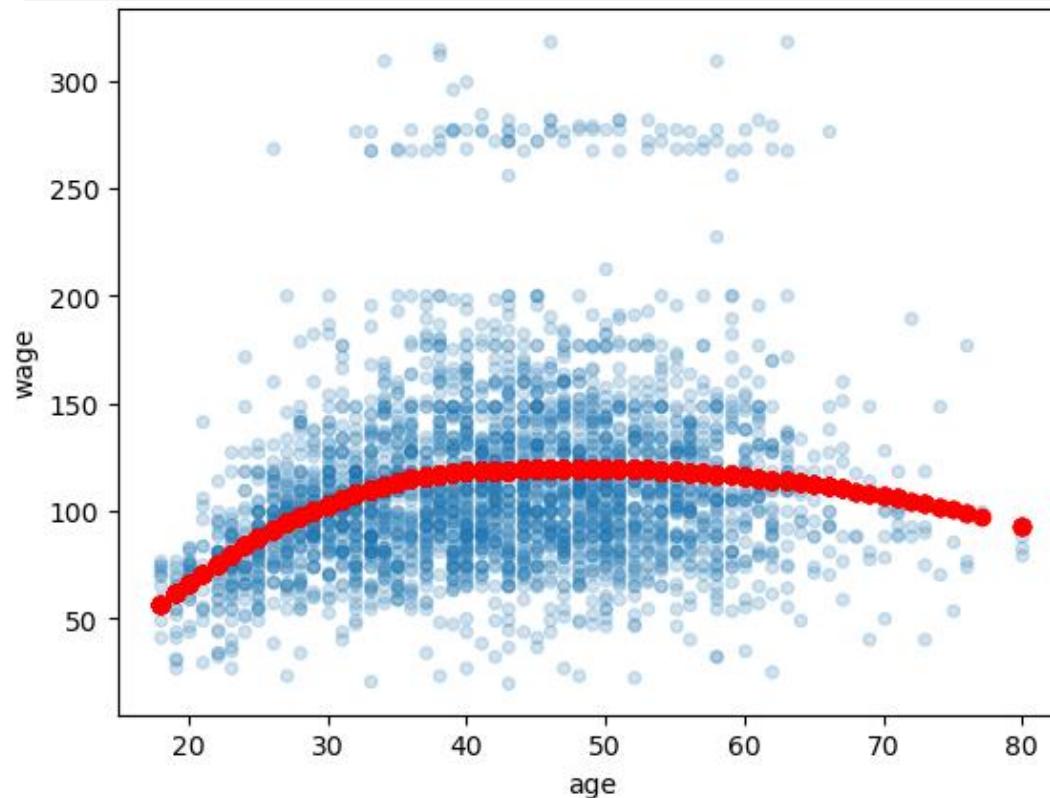
Linear Spline



Пример использования

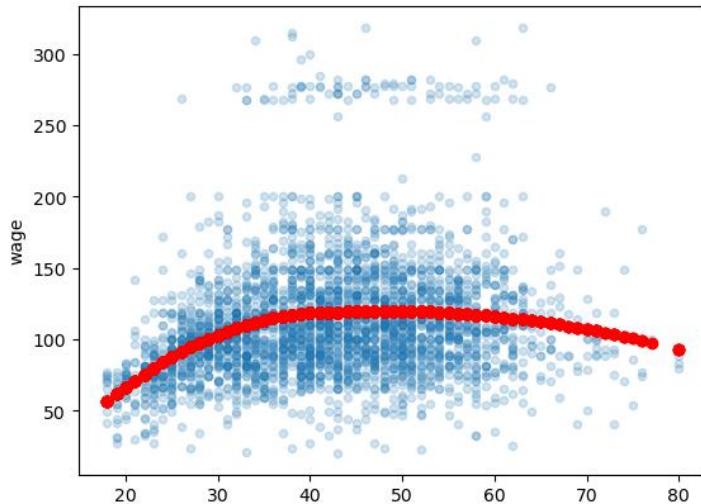
```
poly_model = PolynomialFeatures(degree=2)
X_p = poly_model.fit_transform(X)

model = PiecewiseRegressor(
    estimator=LinearRegression(),
    binner=KBinsDiscretizer(n_bins=2))
model.fit(X_p, y)
```

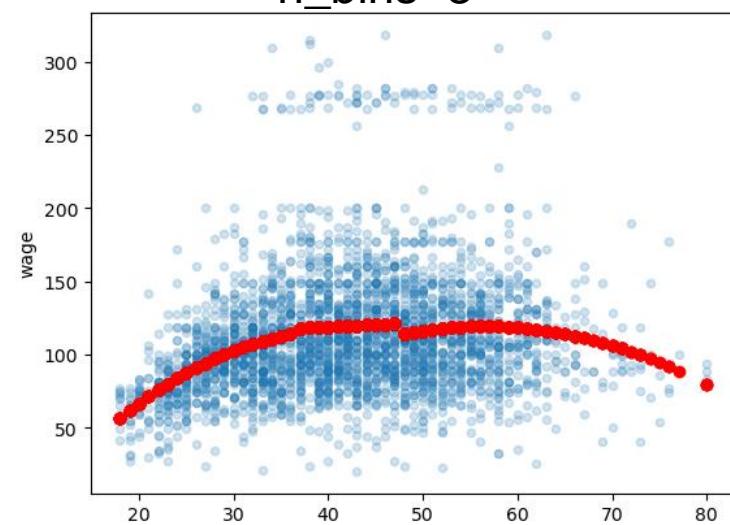


Пример использования

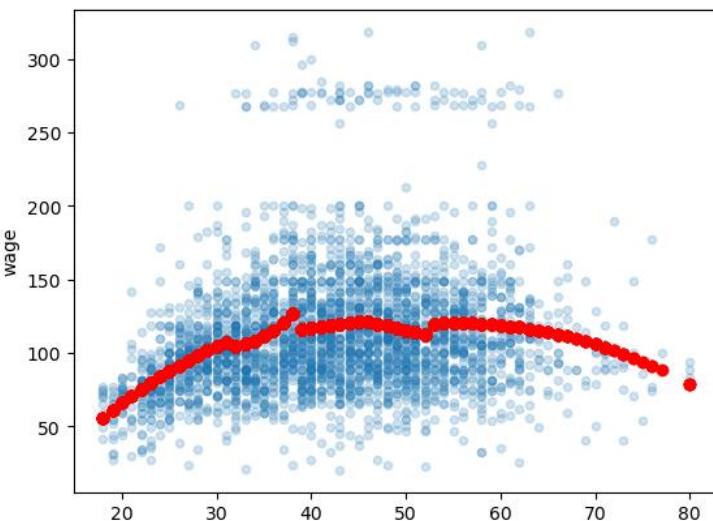
$n_bins=2$



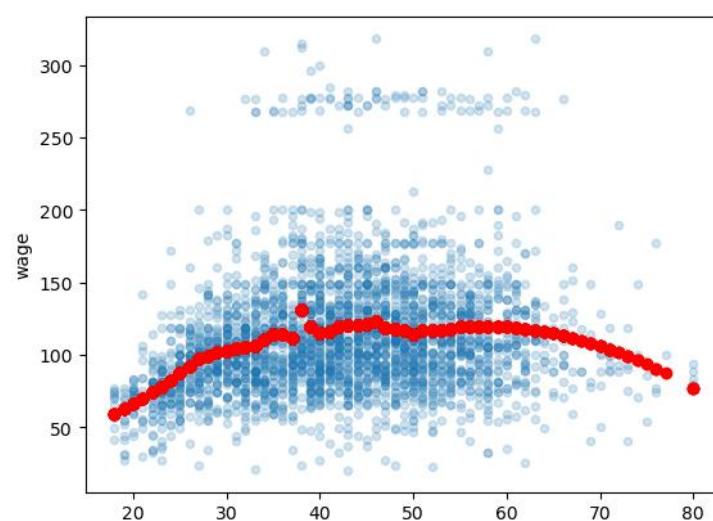
$n_bins=3$



$n_bins=5$



$n_bins=8$



Линейные сплайны

Линейный сплайн с узлами ξ_k , $k = 1, \dots, K$ 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

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

Здесь $(\cdot)_+$ 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 , $k = 1, \dots, K$ представляют собой кусочно-кубический многочлен с непрерывными производными до второго порядка в каждом узле.

Мы можем снова представить эту модель со степенными базисными функциями

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

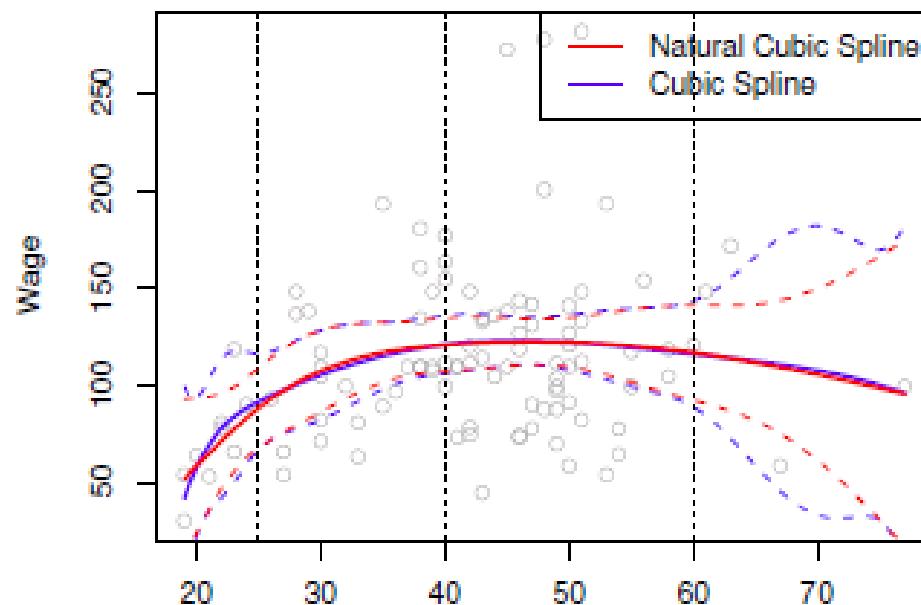
$$\begin{aligned} b_1(x_i) &= x_i \\ b_2(x_i) &= x_i^2 \\ b_3(x_i) &= x_i^3 \\ b_{k+3}(x_i) &= (x_i - \xi_k)_+^3, \quad k = 1, \dots, K \end{aligned}$$

где

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

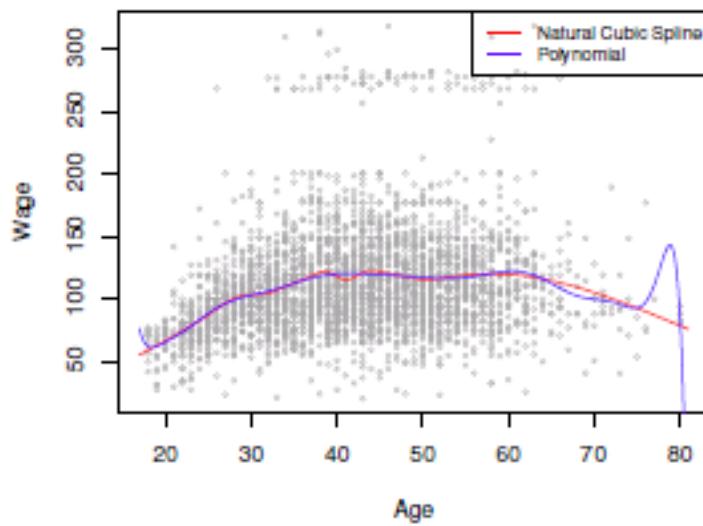
Естественные кубические сплайны

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 сплайнами

Подгонка гладкой функцией $g(x)$:

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

- Первое слагаемое - RSS и он нацелен на то, чтобы $g(x)$ соответствовала данным в каждом x_i .
- Второе слагаемое - 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 $\lambda \rightarrow \infty$, функция $g(x)$ становится линейной.

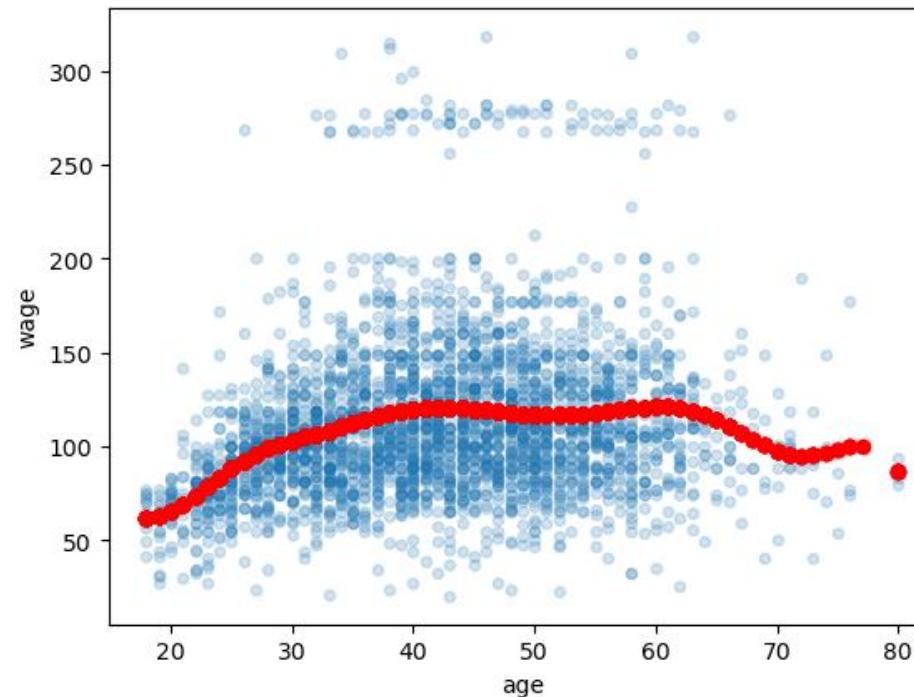
Приимер использования

```
from sklearn.preprocessing import SplineTransformer
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline

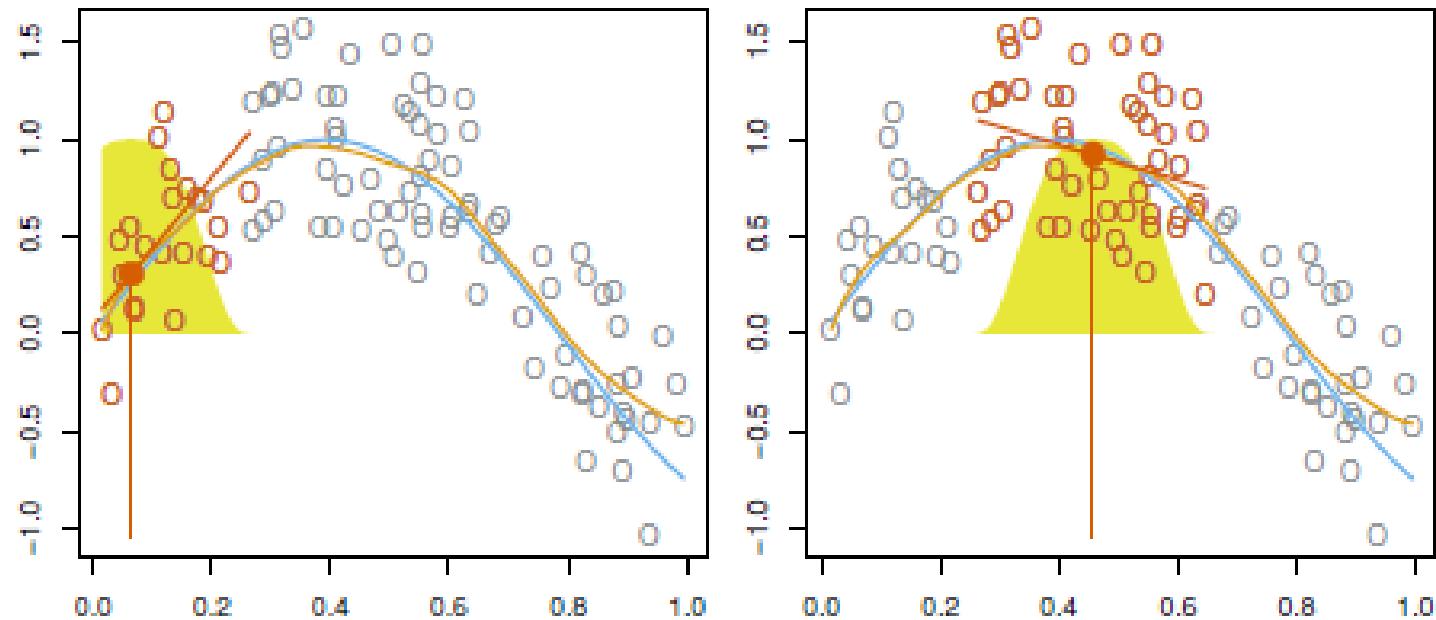
model = Pipeline([('poly', SplineTransformer(n_knots=10, degree=5)),
                  ('linear', LinearRegression(fit_intercept=False))])

model.fit(X, y)

df.plot.scatter(x="age", y="wage", alpha=0.2)
plt.scatter(X, model.predict(X), color="r")
```



Локальная регрессия



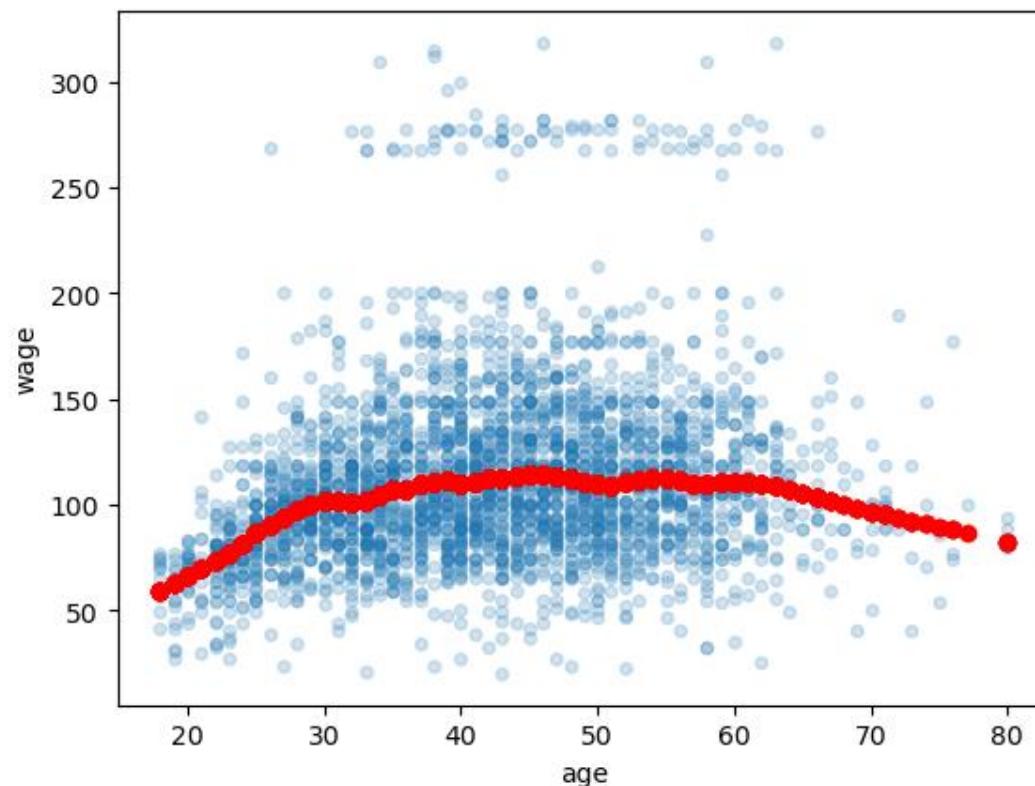
С помощью скользящей весовой функции мы отдельно подгоняем линейные участки по диапазону X с помощью взвешенных наименьших квадратов.

Пример использования

```
import statsmodels.api as sm

yest_sm = sm.nonparametric.lowess(y, X["age"].values,
                                 frac=1/10, return_sorted = False)

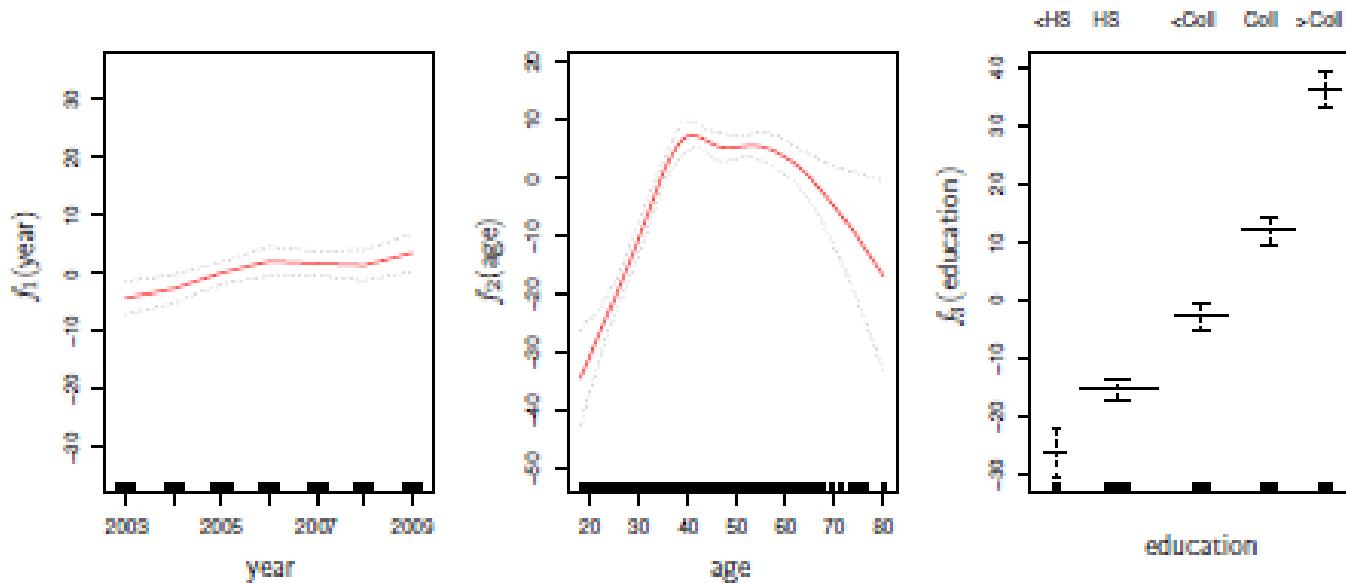
df.plot.scatter(x="age", y="wage", alpha=0.2)
plt.scatter(X, yest_sm, color="r")
```



Обобщенные аддитивные модели

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

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



Обобщенные аддитивные модели

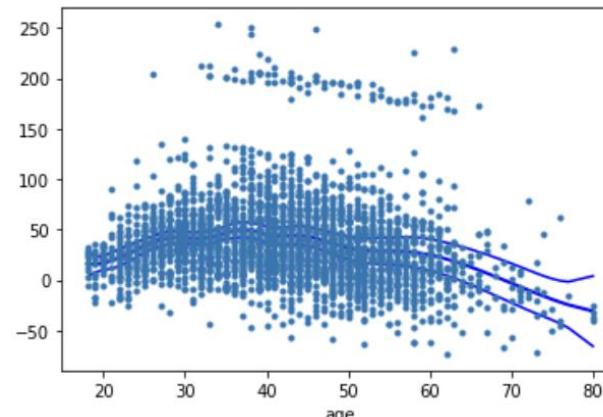
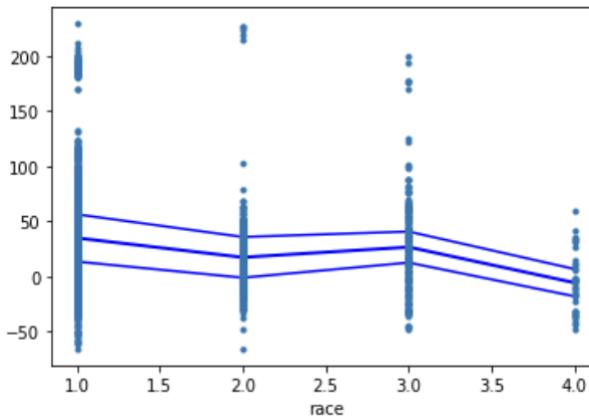
```
from statsmodels.gam.api import GLMGam, BSplines

x_spline = df[["age", "race"]]

bs = BSplines(x_spline, df=[12, 10], degree=[3, 3])

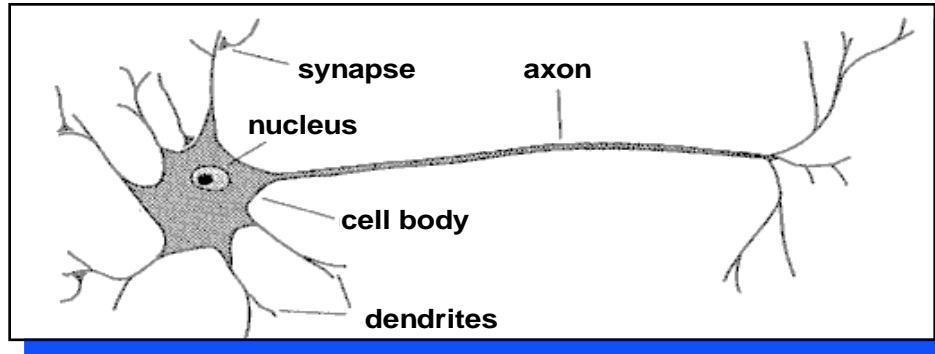
gam_bs = GLMGam.from_formula('wage ~ age + race',
                             data=df,
                             smoother=bs)

res_bs = gam_bs.fit()
print(res_bs.summary())
res_bs.plot_partial(0, cpr=True)
res_bs.plot_partial(1, cpr=True)
```



Generalized Linear Model Regression Results						
Dep. Variable:	wage	No. Observations:	3000			
Model:	GLMGam	Df Residuals:	2985.00			
Model Family:	Gaussian	Df Model:	14.00			
Link Function:	identity	Scale:	1576.2			
Method:	PIRLS	Log-Likelihood:	-15293.			
Date:	Tue, 21 Mar 2023	Deviance:	4.7049e+06			
Time:	16:32:38	Pearson chi2:	4.70e+06			
No. Iterations:	3	Pseudo R-squ. (CS):	0.1036			
Covariance Type:	nonrobust					
coef	std err	z	P> z	[0.025	0.975]	
Intercept	16.1033	6.733	2.392	0.017	2.907	29.300
age	1.1633	0.228	5.099	0.000	0.716	1.610
race	7.7810	2.440	3.188	0.001	2.998	12.564
age_s0	-3.8239	15.797	-0.242	0.809	-34.785	27.137
age_s1	34.9049	8.833	3.952	0.000	17.592	52.218
age_s2	20.7848	8.235	2.524	0.012	4.644	36.925
age_s3	39.5867	6.703	5.906	0.000	26.450	52.724
age_s4	24.9658	6.410	3.895	0.000	12.402	37.530
age_s5	31.0542	5.743	5.407	0.000	19.797	42.311
age_s6	19.4717	5.395	3.609	0.000	8.898	30.045
age_s7	11.2821	4.921	2.293	0.022	1.638	20.926
age_s8	17.7031	9.969	1.776	0.076	-1.835	37.241
age_sq	-21.1752	17.611	-1.958	0.050	-68.993	0.042
	.926	-3.136	0.002	-76.069	-17.559	
z=16	2.220	0.026	1.05e-16	1.69e-15		
0	nan	nan	0	0		
0	nan	nan	0	0		
0	nan	nan	0	0		
.491	4.124	0.000	9.720	27.325		
.664	-1.777	0.076	-17.431	0.852		
.989	1.505	0.132	-1.360	10.359		
.563	4.225	0.000	12.602	34.408		
.894	-4.523	0.000	-31.727	-12.542		

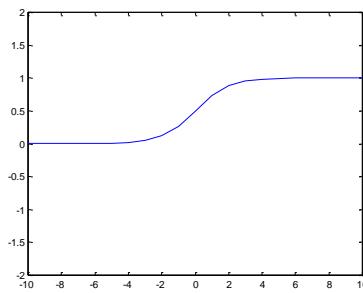
Нейронные сети - биологическая мотивация



- 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 (дendриты)
 - «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

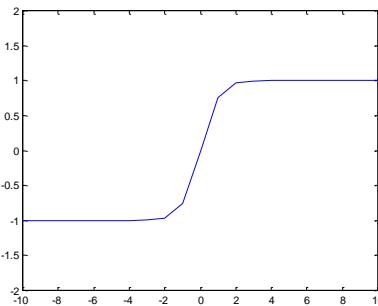
- Определение:
 - Нелинейная, параметризованная функция с ограниченным диапазоном значений
- Функции активации:



логистическая

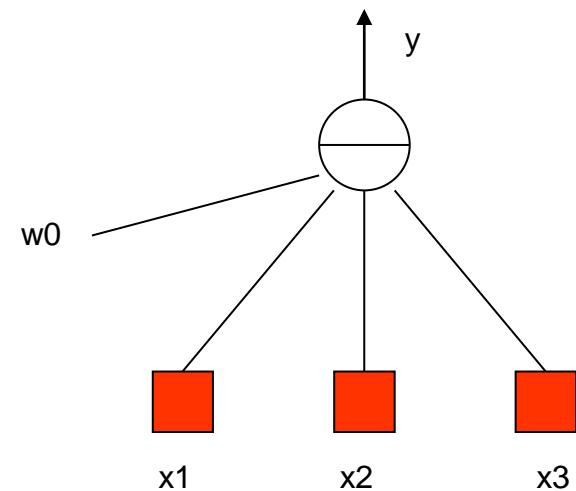
$$y = f\left(w_0 + \sum_{i=1}^{n-1} w_i x_i\right)$$

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



Гиперболический тангенс

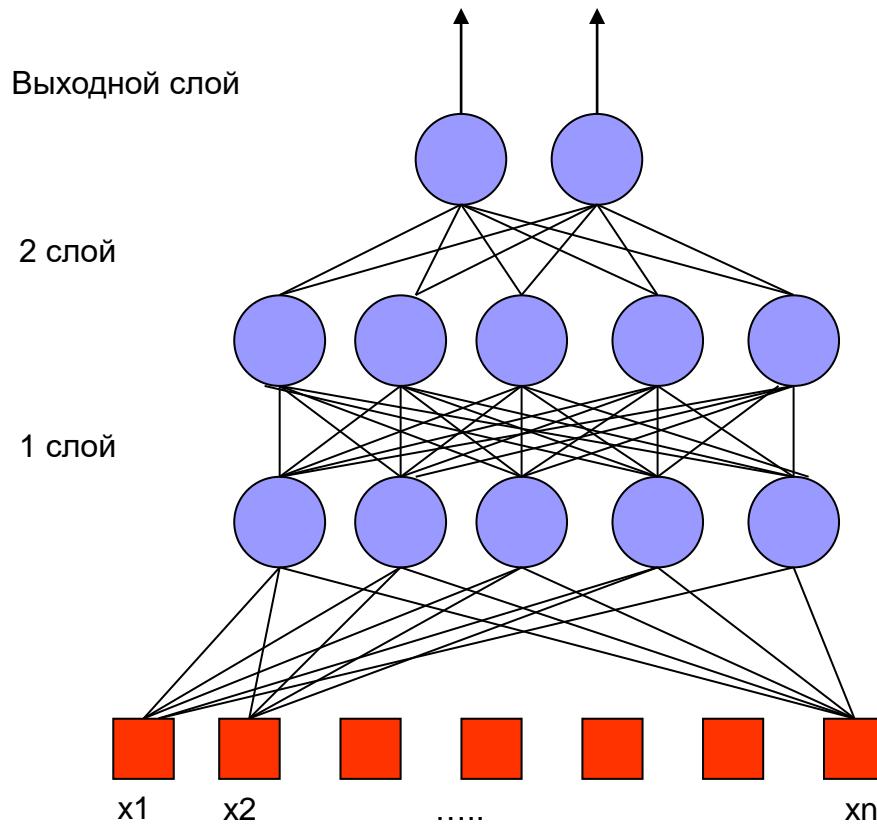
$$y = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



Нейронная сеть (искусственная)

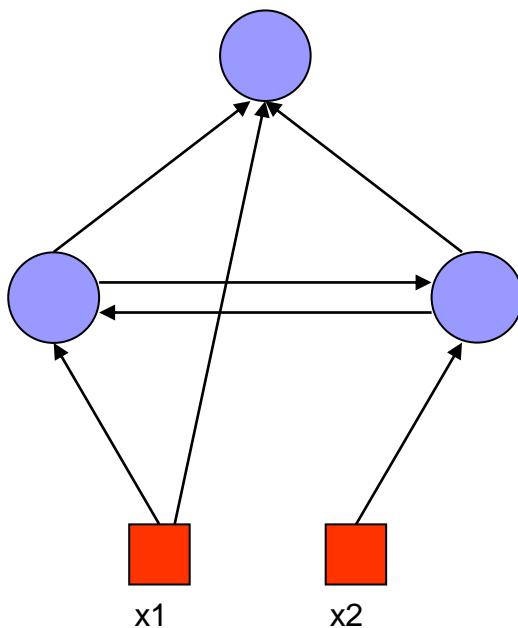
- Математическая модель для решения задач машинного обучения
 - Реализуется группой соединенных нейронов для моделирования нелинейных зависимостей
- Задачи:
 - Классификации, дискриминации, оценки плотности, регрессии, группировки и кластеризации, выявления зависимостей, главных и независимых компонент
- 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:

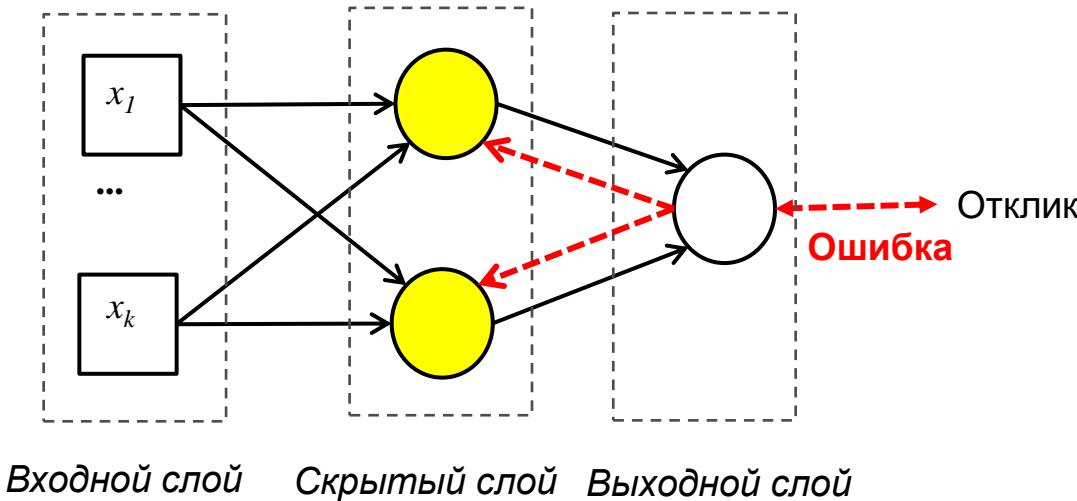
$$\Delta w_{ij} = \eta (y_i - \hat{y}_i) x_j$$

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

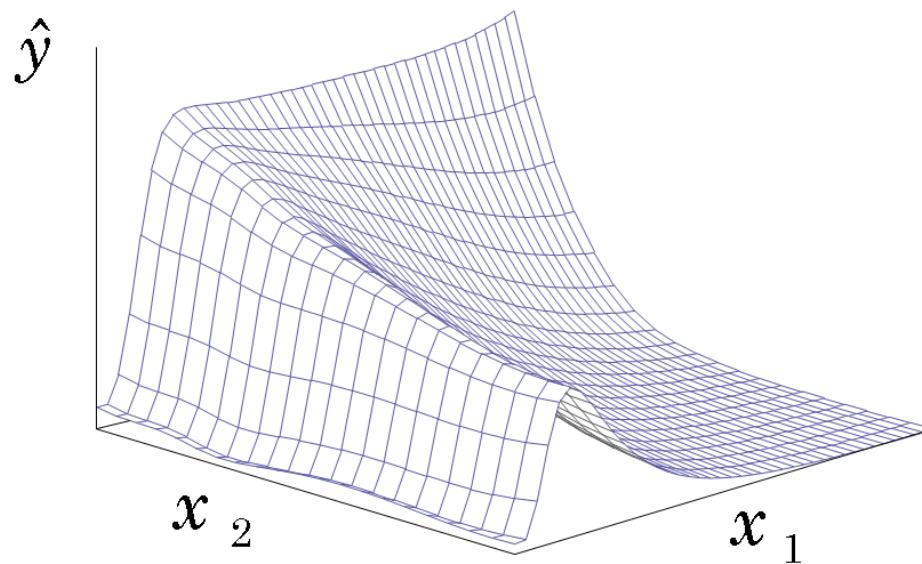
Обобщенное дельта правило

- 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).



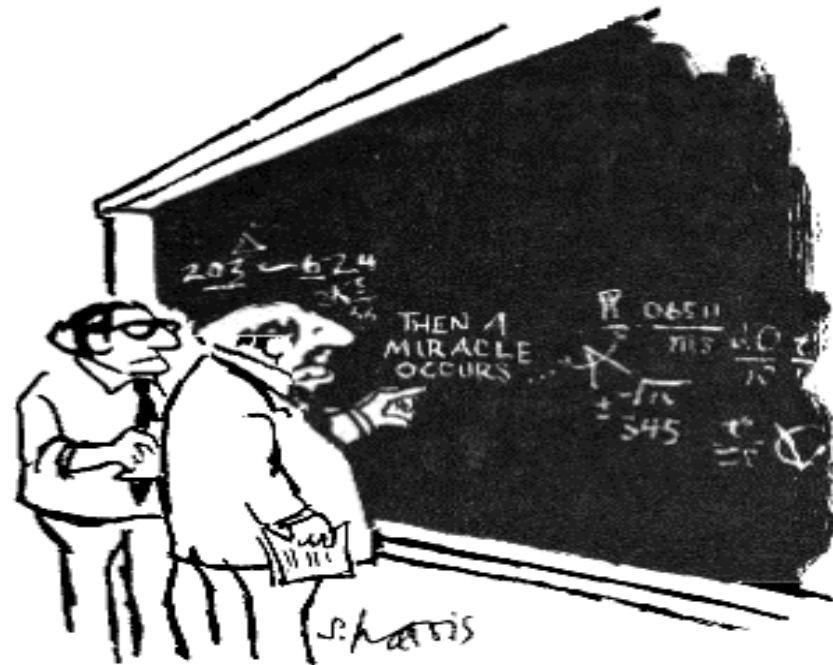
Универсальный аппроксиматор



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

Не нужна явная формулировка искомой зависимости

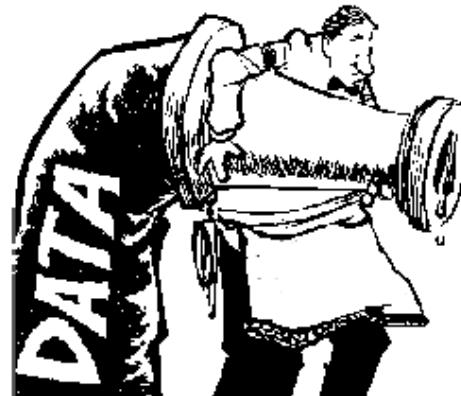
- Не нужно задавать форму зависимости априори (как в регрессиях и опоных векторах), даже приблизительно «понимать» ее не нужно
- сложнее сеть => сложнее зависимость, быстрее переобучение



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

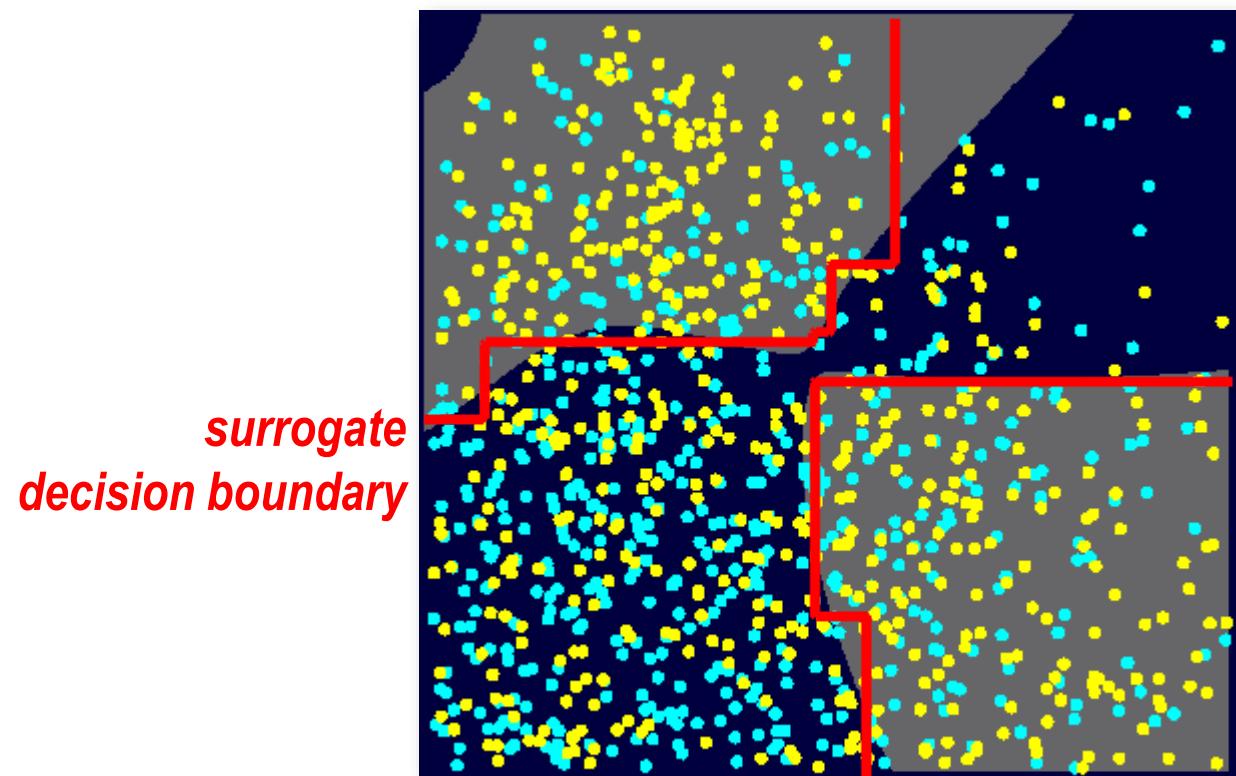
Скорость применения

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



Недостаточная интерпретируемость

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

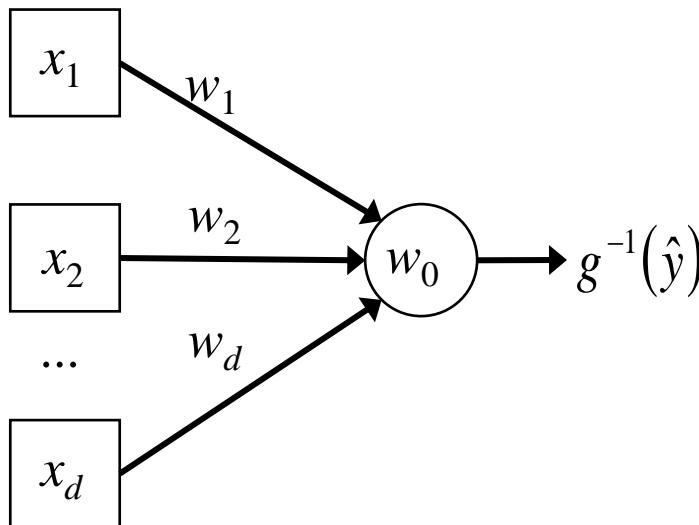


kritika

- “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.

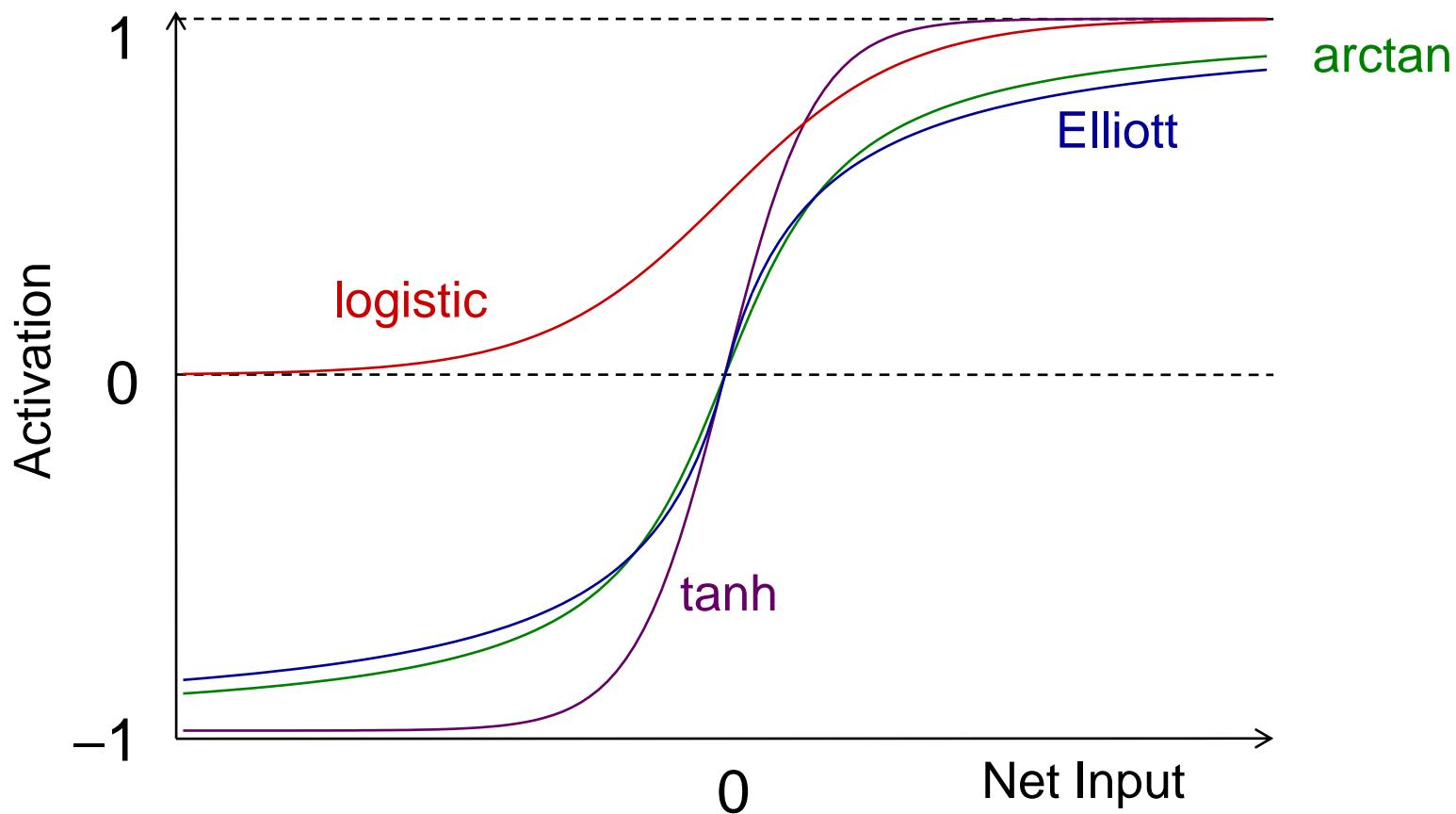
Линейный персепtron (он же GLM)

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



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

Функции активации (она же обратная к функции связи)

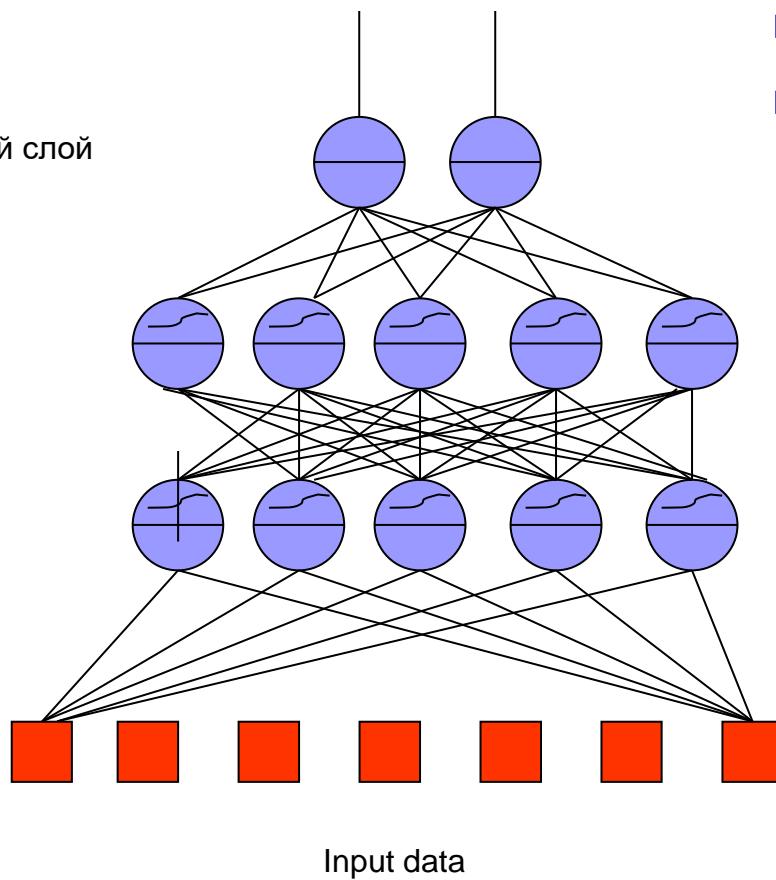


Многослойный персепtron

Выходной слой

2 слой

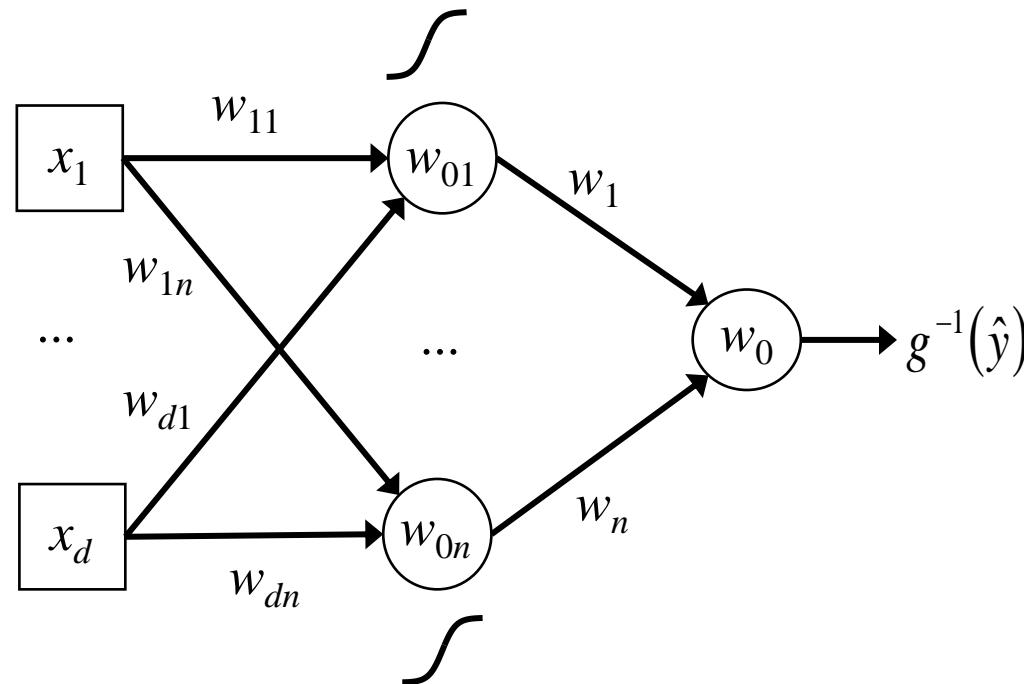
1 слой



- Один или более скрытых уровней
- Функции активации
сигмоидального типа

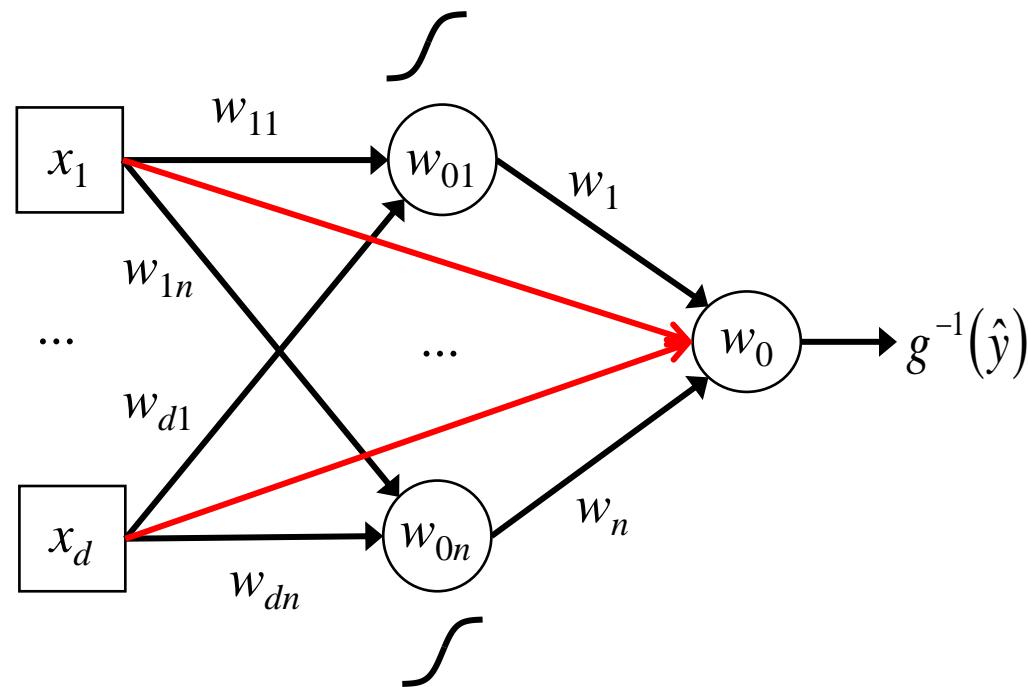
Многослойный персепtron

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



Персептрон с прямыми соединениями

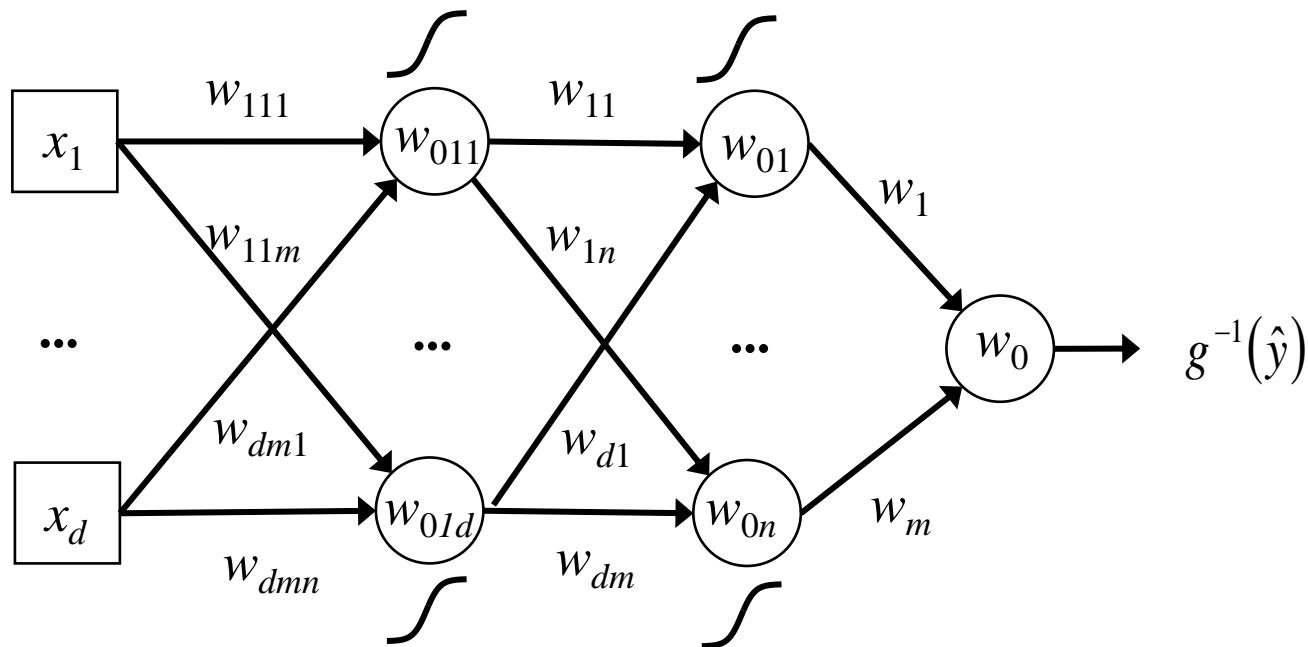
$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^h w_i g_i \left(w_{0i} + \underbrace{\sum_{j=1}^d w_{ij} x_j}_{\text{Скрытый слой}} \right) + \underbrace{\sum_{k=1}^d w_{11k} x_k}_{\text{Прямые соединения}}$$



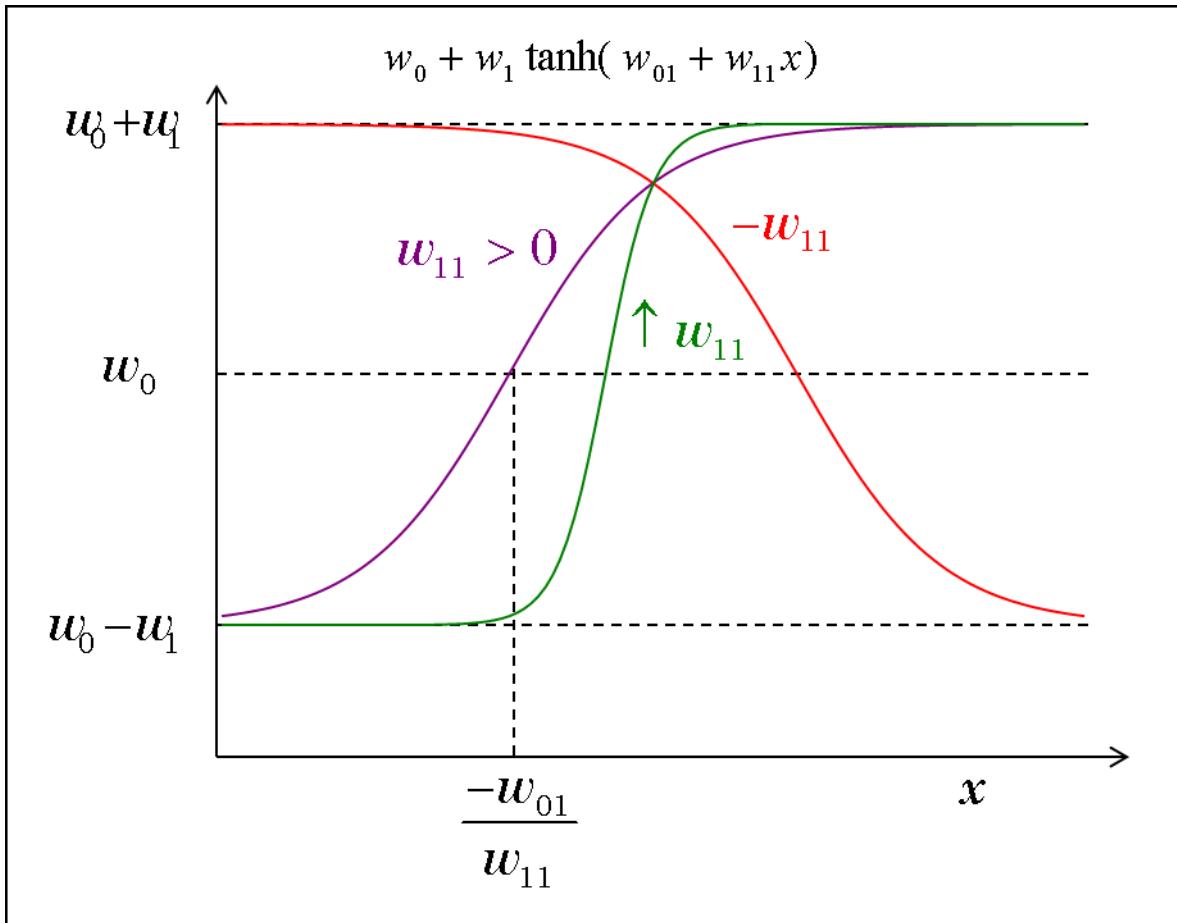
Два и более скрытых слоя

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

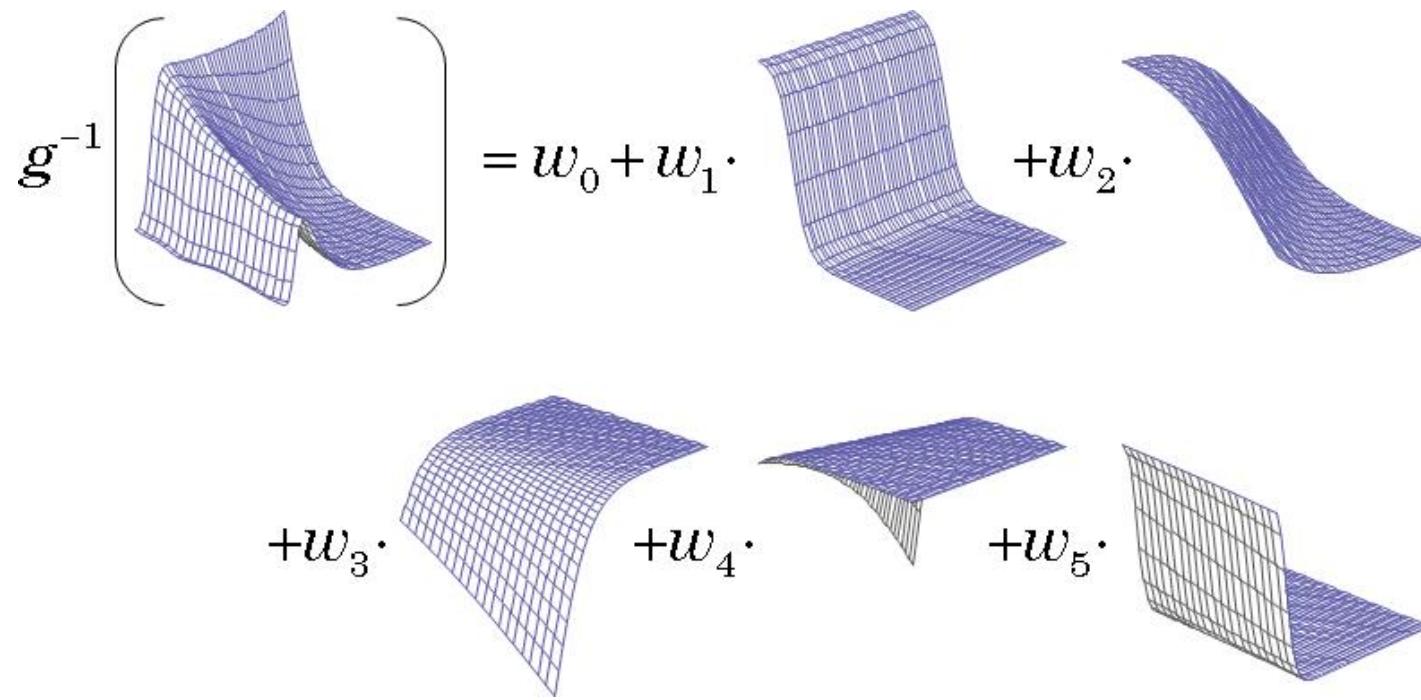
Вложенные скрытые слои



Форма сигмоида

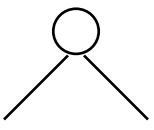
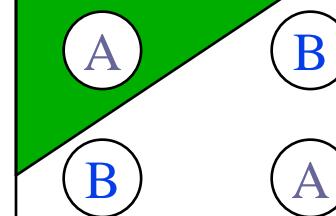
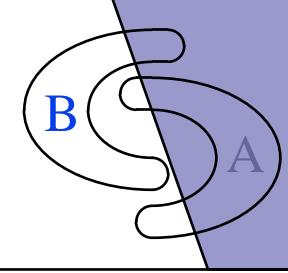
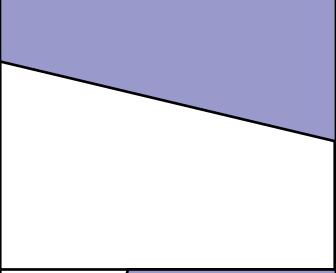
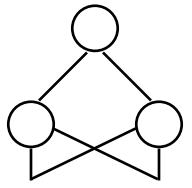
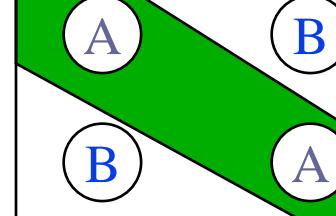
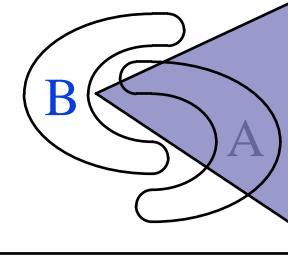
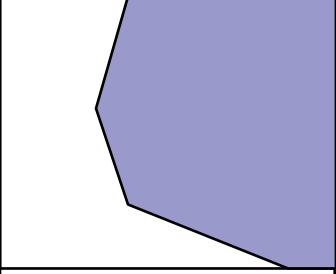
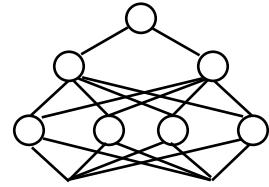
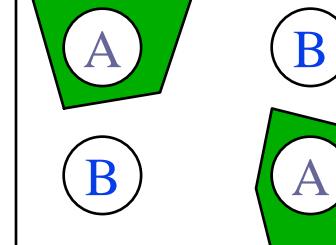
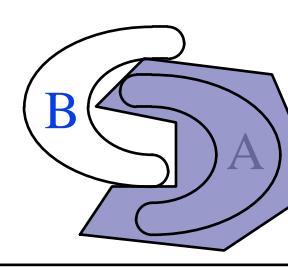
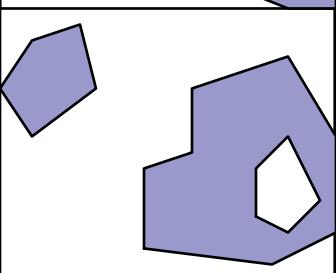


Разложение по базису сигмоидальных функций



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

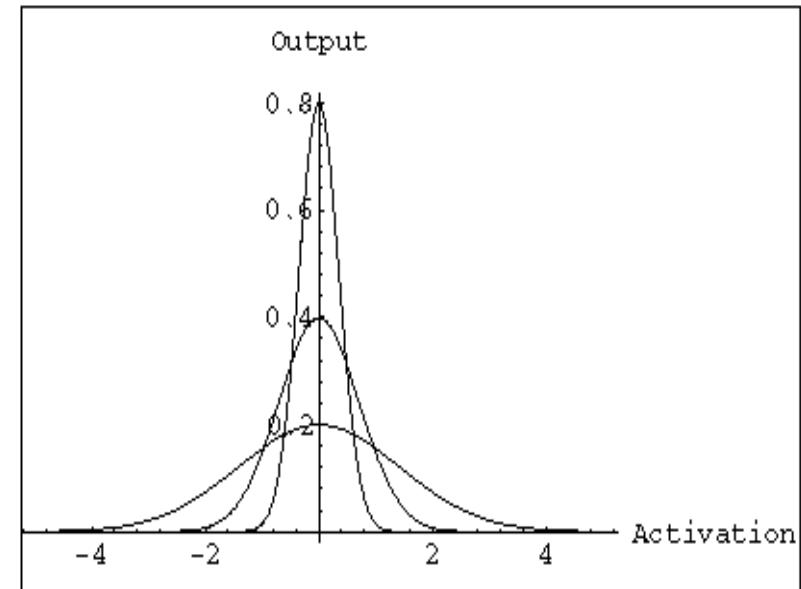
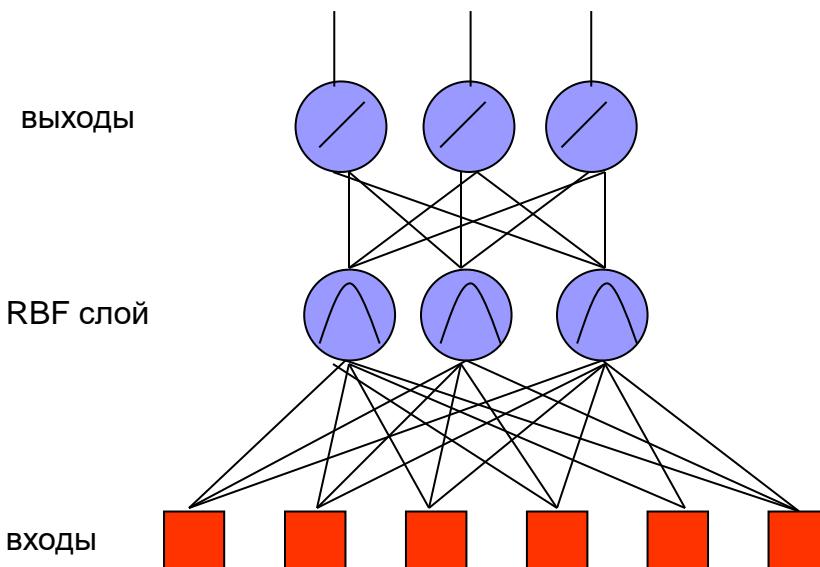
Типы решаемых задач

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

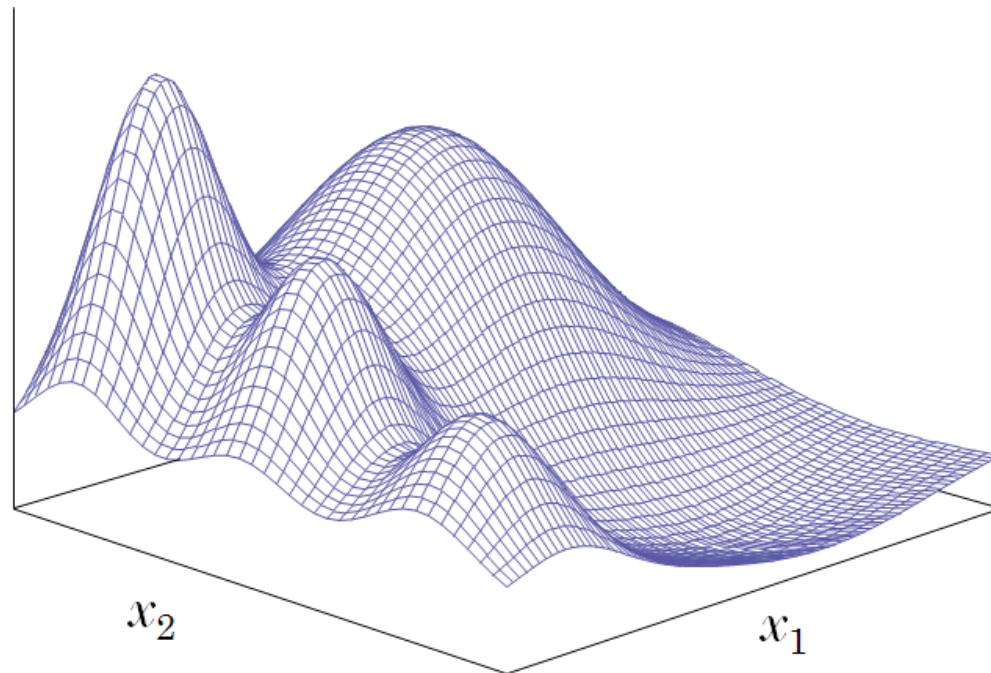
Радиально-базисные сети

■ Свойства:

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

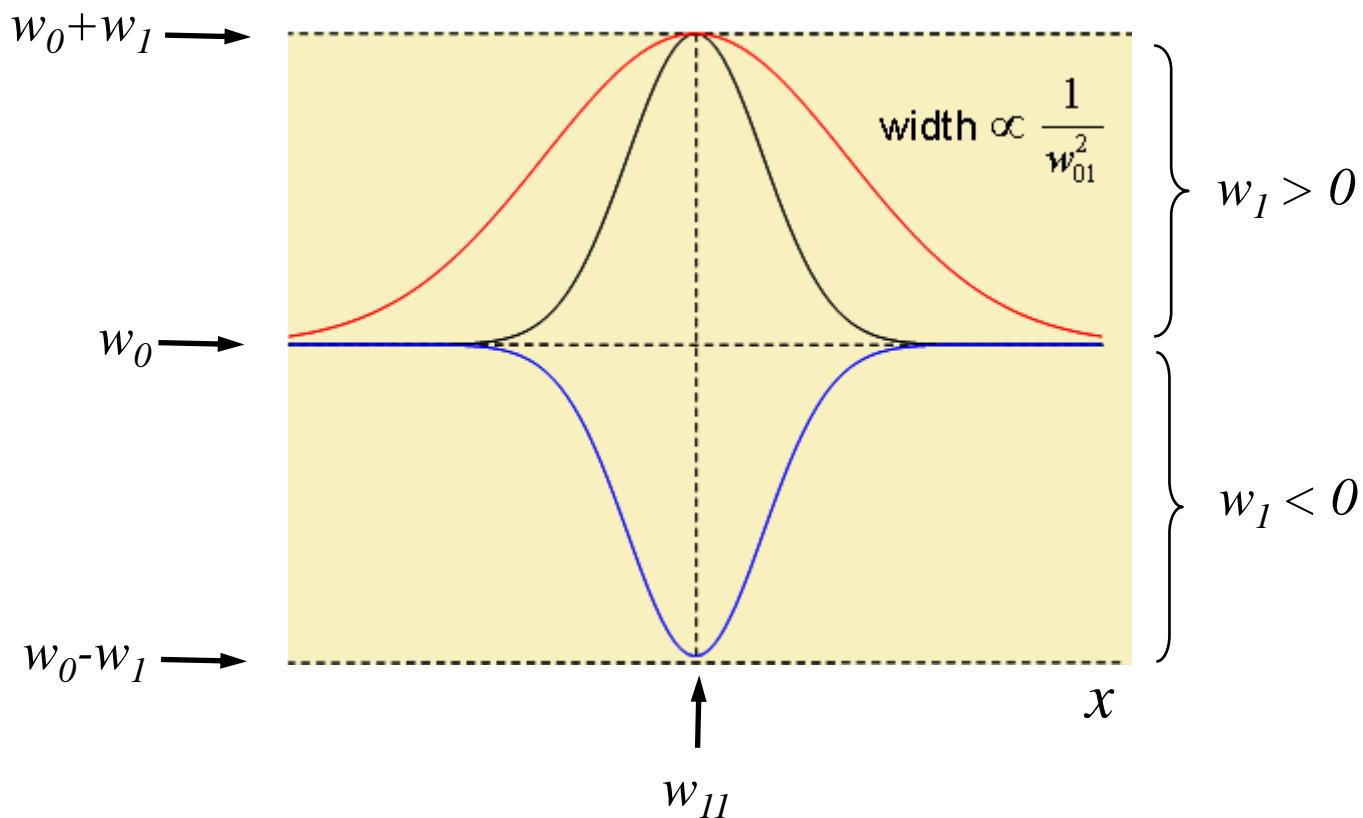


- Ordinary Radial Basis Functions (ORBFs)
- Normalized Radial Basis Functions (NRBFs)



Форма функции гаусса

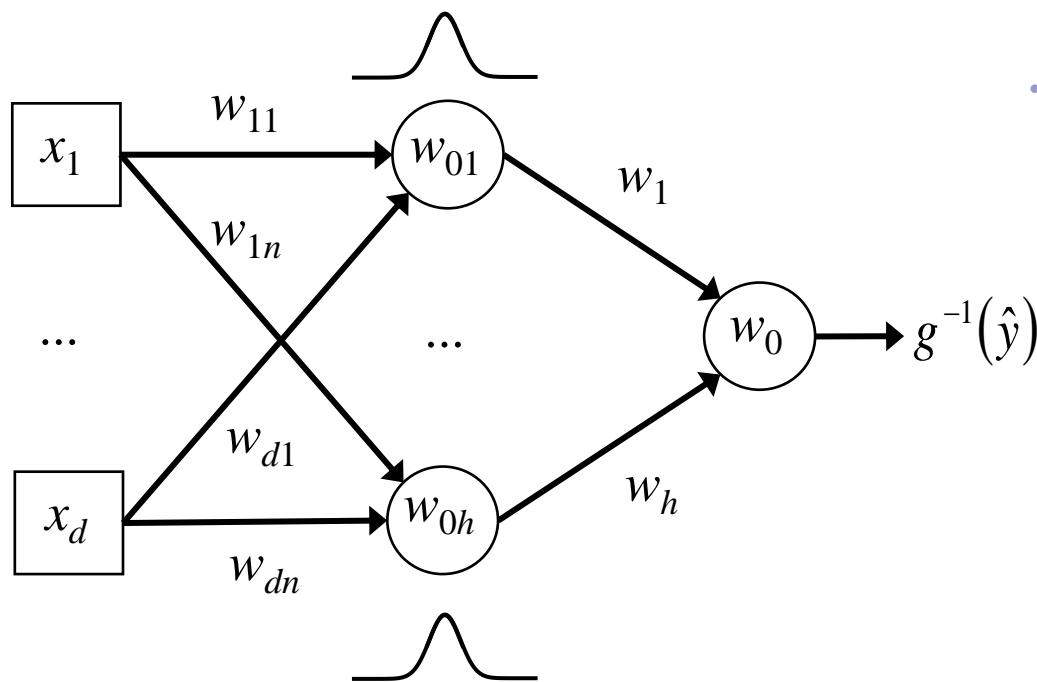
$$w_0 + w_1 \exp\left(-w_{01}^2(x - w_{11})^2\right)$$



RBF нейронная сеть

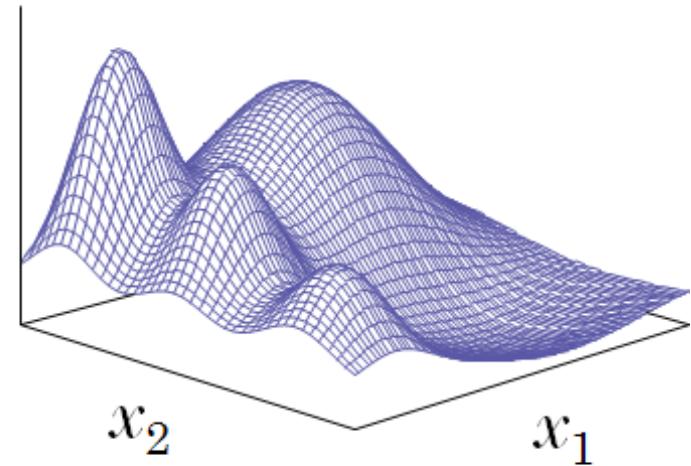
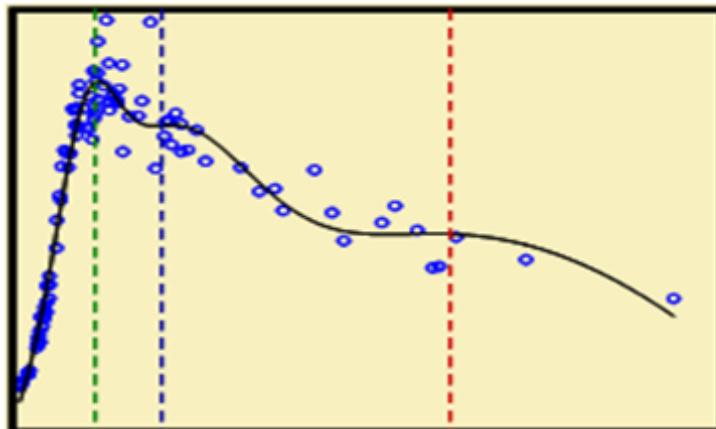
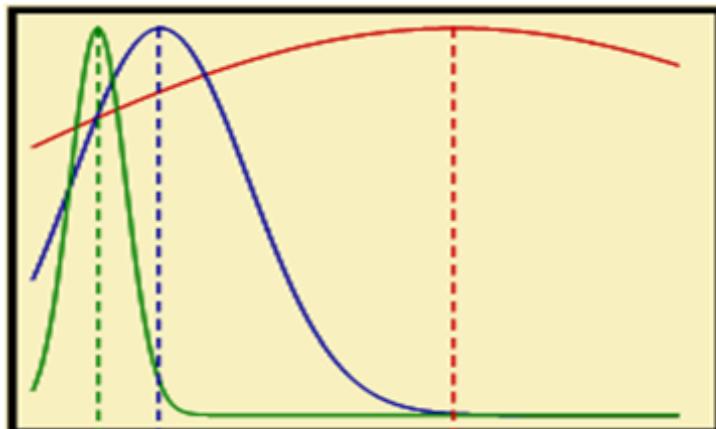
$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^h w_i \exp \left[-w_{0i} \left(\sum_j (w_{ij} - x_j)^2 \right) \right]$$

Скрытый слой



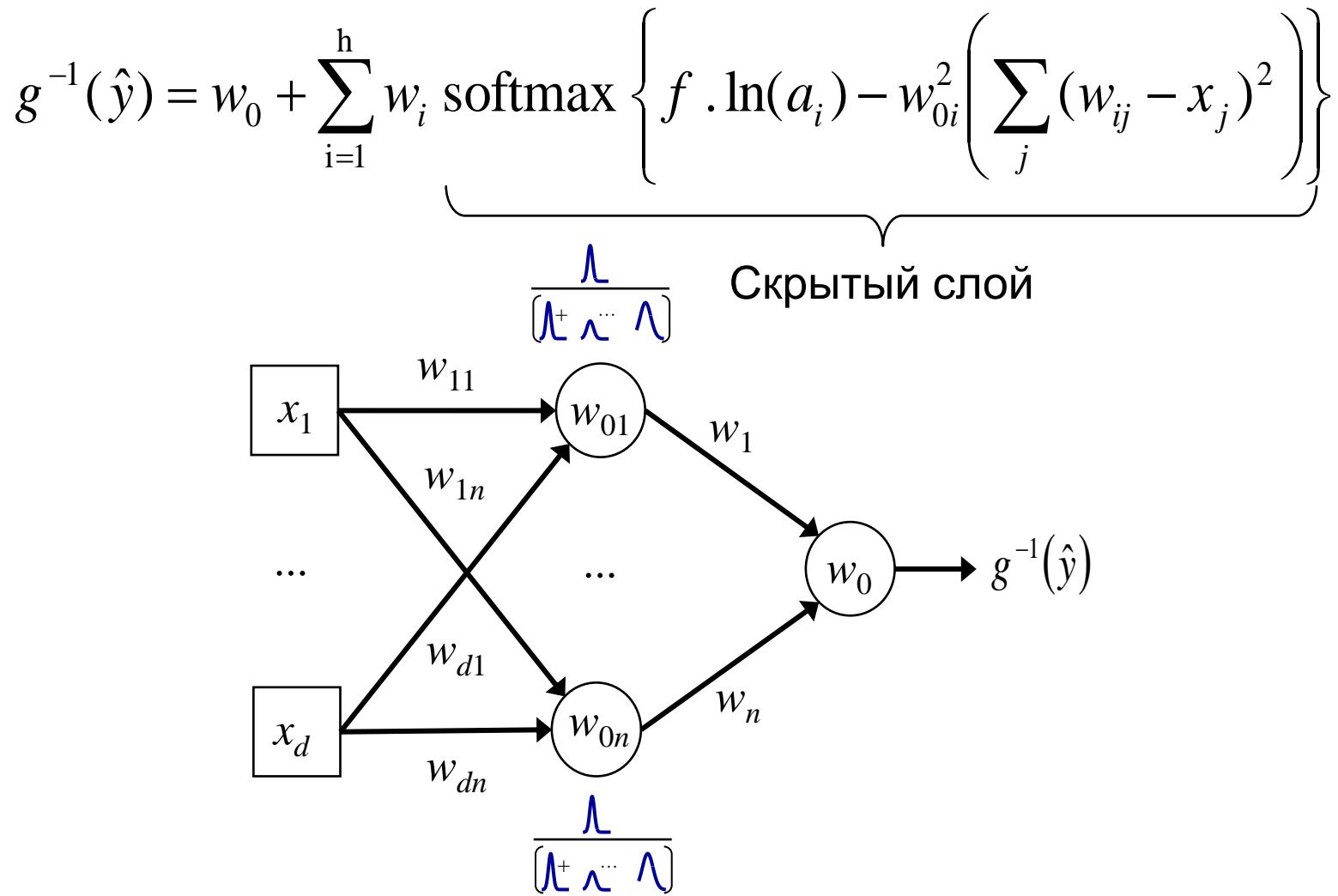
- Типы параметров обычной RBF сети:
 - высота и ширина ядра различные у всех нейронов
 - высота и ширина ядра одинаковые
 - одинаковая ширина
 - одинаковая высота

Проблема локального эффекта

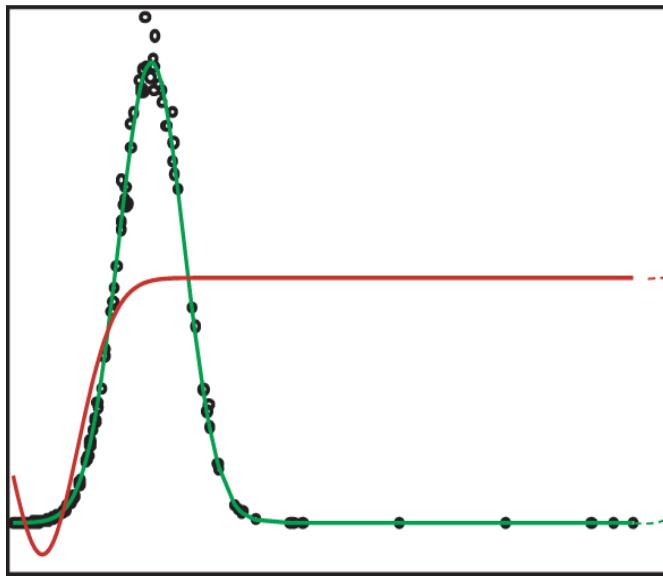


- Локальный эффект:
 - сложнее функция – больше прототипов
 - Проклятие размерности

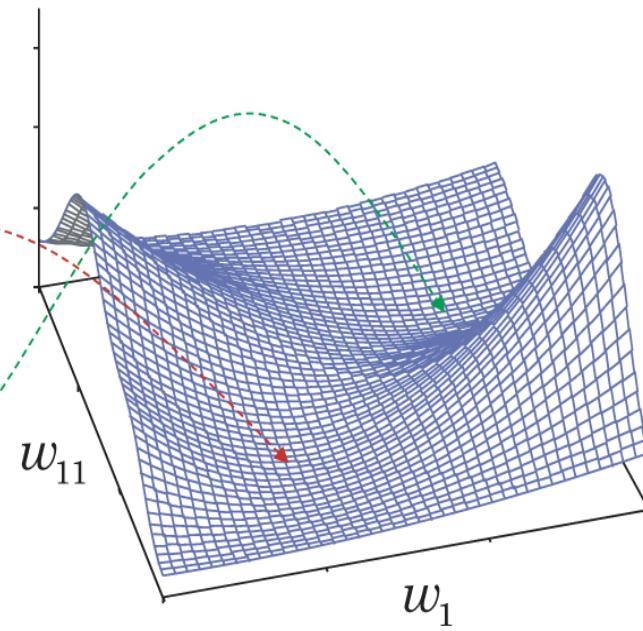
Нормализованные радиально-базисные сети



Проблема локальных минимумов

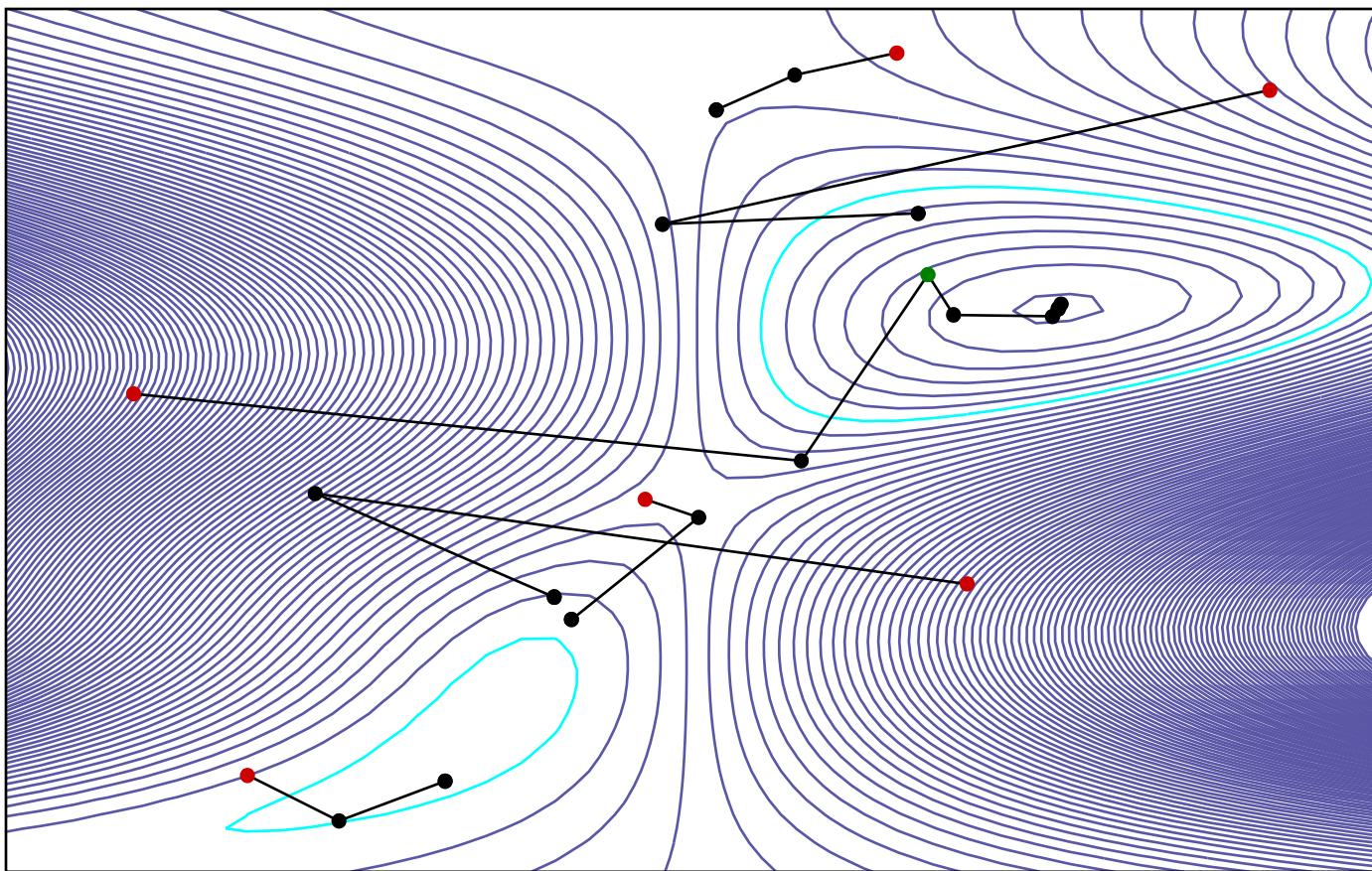


Error Function



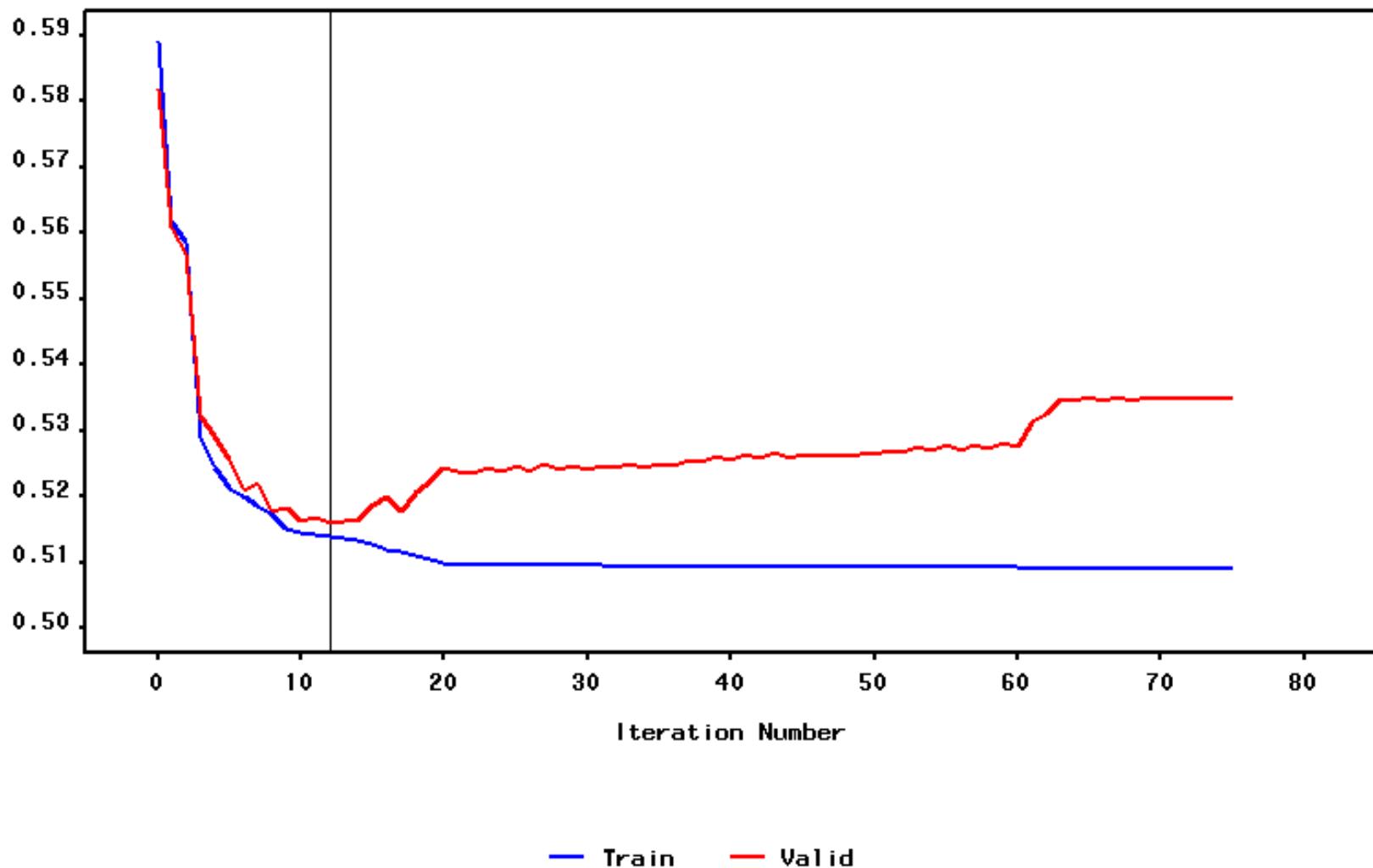
$$w_0 + w_1 \exp(-w_{01}^2(x - w_{11})^2)$$

Предварительное обучение

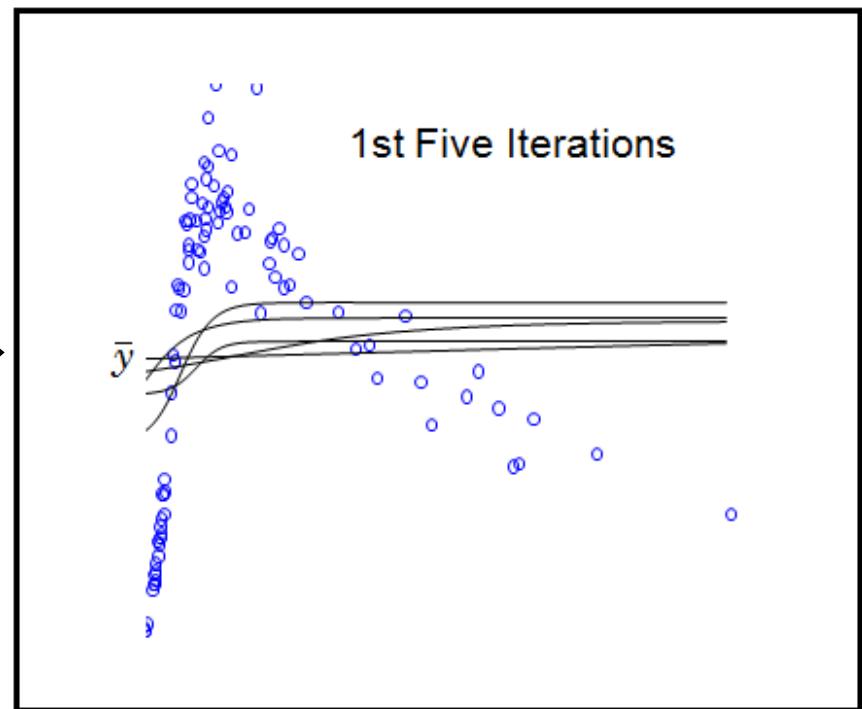
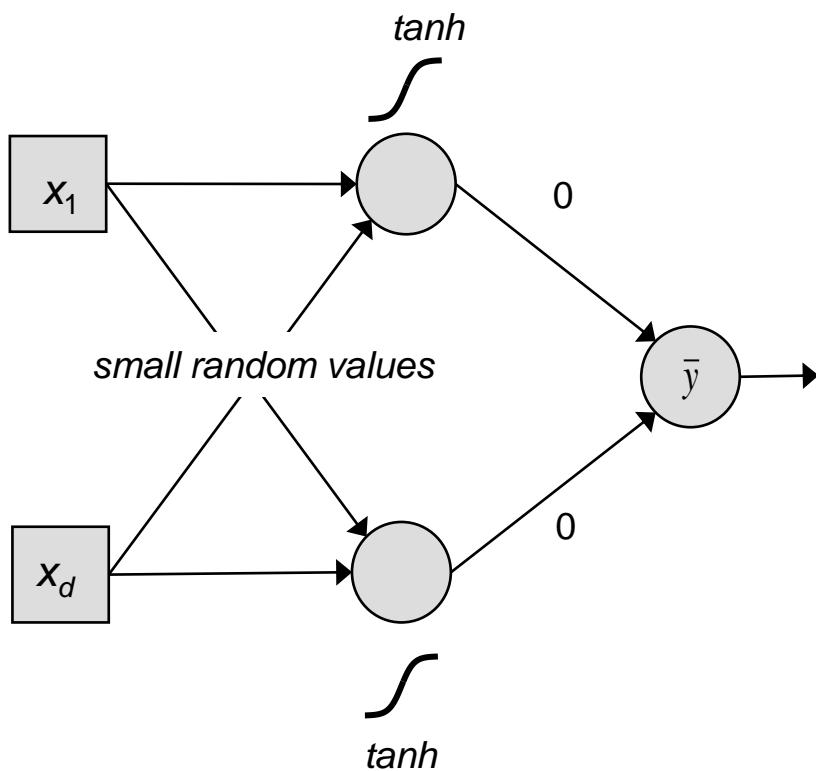


Ранняя остановка – борьба с переобучением

Average Error (New)

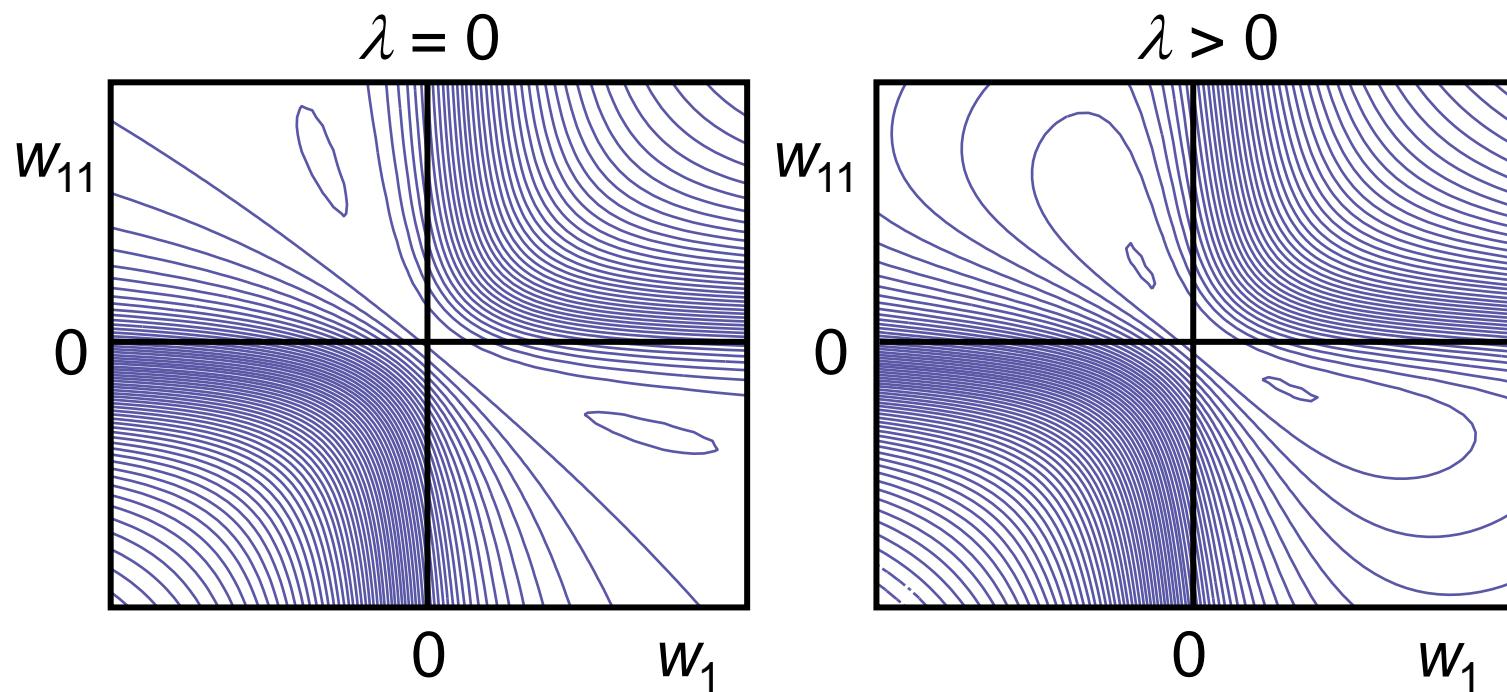


Инициализация



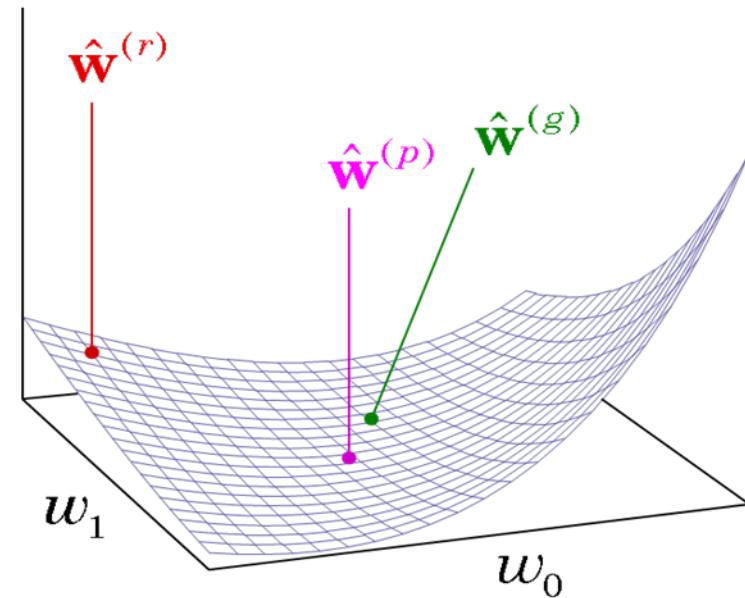
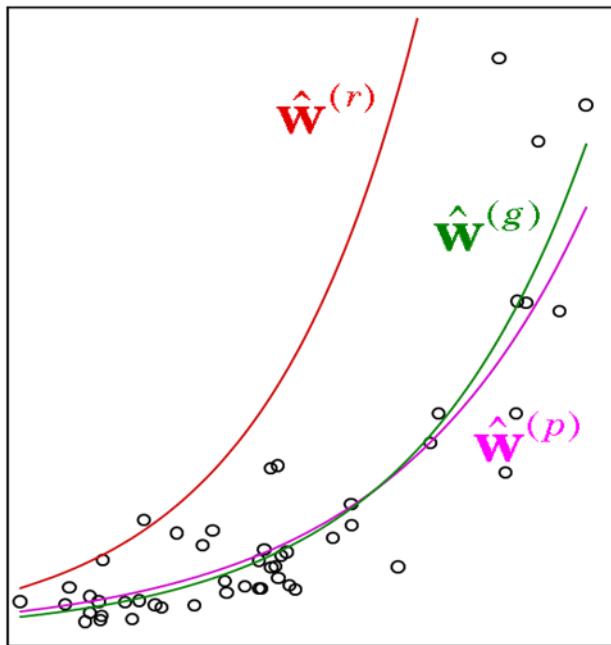
Регуляризация

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



Оценки максимального правдоподобия

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



Оценка отклонения

Поиск параметров модели

- решается задача оптимизации
- $\max \text{loglik}$ с заданным распределением и функцией связи

Распределение Отклонение

Normal $Q(\mathbf{w}) = \sum (y - \mu(\mathbf{w}))^2$

Poisson $Q(\mathbf{w}) = 2 \sum [y \ln(y / \mu(\mathbf{w})) - (y - \mu(\mathbf{w}))]$

Gamma $Q(\mathbf{w}) = 2 \sum [-\ln(y / \mu(\mathbf{w})) + (y - \mu(\mathbf{w})) / \mu(\mathbf{w})]$

Bernoulli $Q(\mathbf{w}) = -2 \sum [y \ln(\mu(\mathbf{w})) + (1 - y) \ln(1 - \mu(\mathbf{w}))]$

Типовые функции связи

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

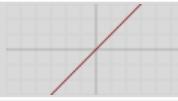
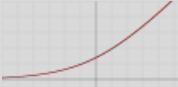
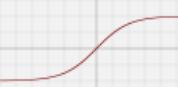
Комбинации функций активации и распределения ошибок

Отклик	Функция связи	Функция 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*.

Функции активации

Function	Plot	Equation	Range
Exponential		$f(x) = e^x$	$[0, \infty)$
Identity		$f(x) = x$	$(-\infty, \infty)$
Logistic		$f(x) = \frac{1}{1 + e^{-x}}$	$(0,1)$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Sine		$f(x) = \sin(x)$	$[-1,1]$
Softplus		$f(x) = \ln(1 + e^x)$	$[0, \infty)$
Hyperbolic Tangent (Tanh)		$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$(-1,1)$

Задача оптимизации

$$\underset{x}{\text{minimize}} \quad f(x)$$

$$\text{subject to} \quad g_i(x) \leq 0, \quad i = 1, \dots, m$$

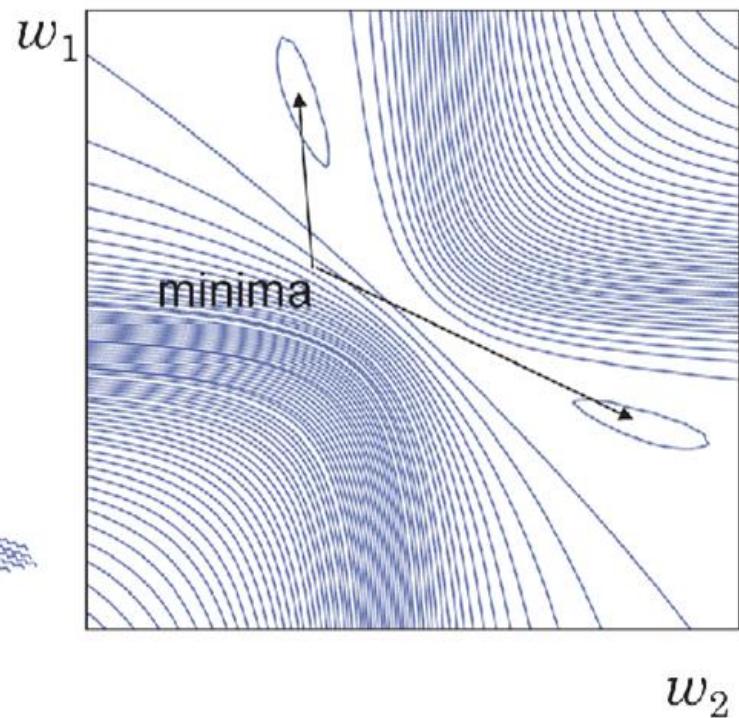
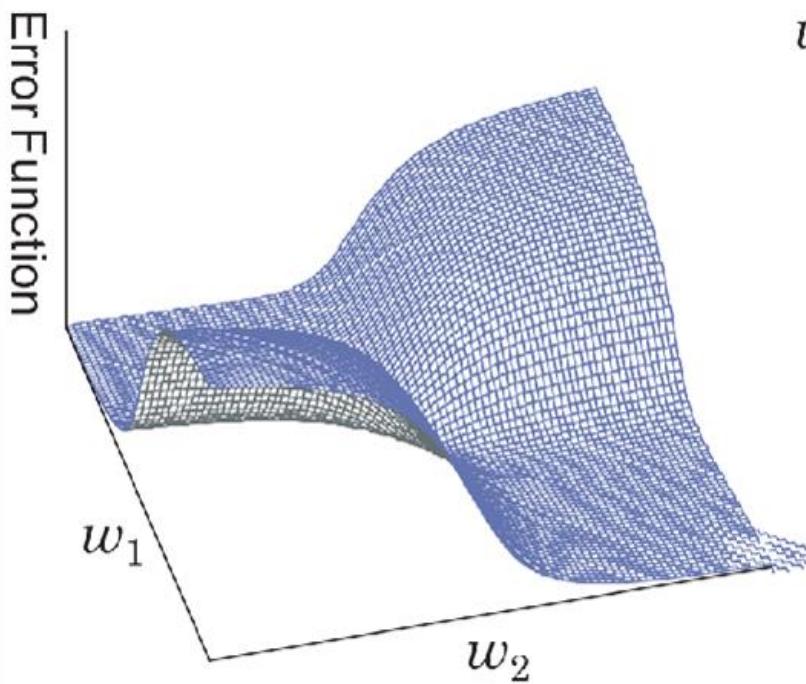
$$h_i(x) = 0, \quad i = 1, \dots, p$$

- Методы первого порядка – градиентные (используют шаг «вдоль» направления градиента – вектора первых производных)
 - выбор шага (константа, дробный выбор, адаптивный, наискорейший)
 - выбор направления (с учетом предыдущих шагов, например сопряженные градиенты)
- Методы второго порядка – ньютоновские (используют матрицу вторых производных Гессе для «выбора шага»)
 - проблема – вычисление обратной матрицы Гессе на каждом шаге

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

Итерационные методы

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



Градиентный:

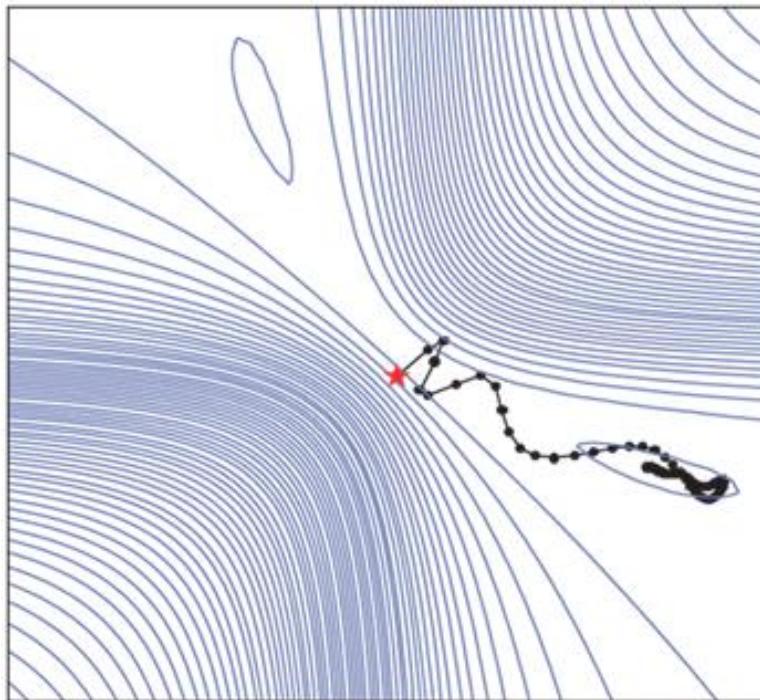
$$\boldsymbol{\delta}^{(t)} = -\eta \nabla g^{(t)}$$

Ньютона:

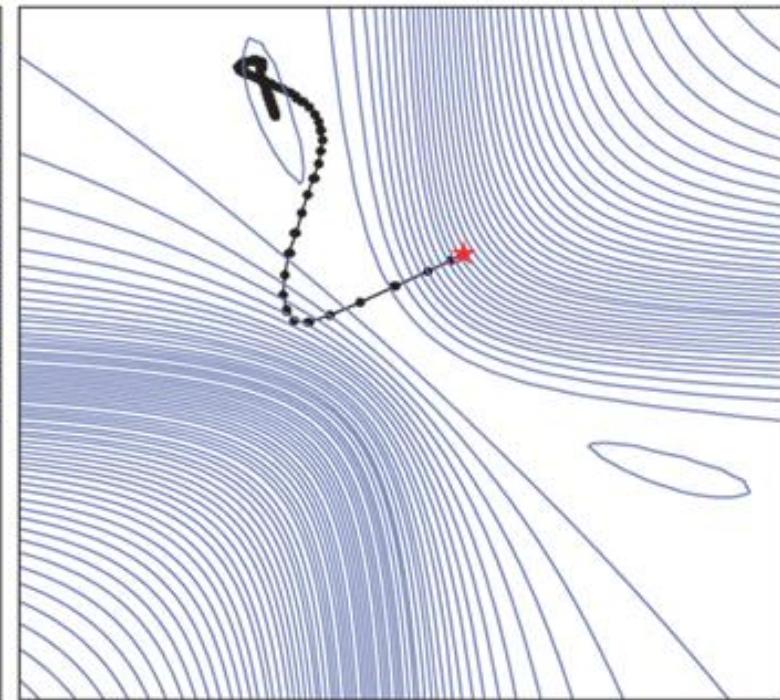
$$\boldsymbol{\delta}^{(t)} = -[\mathbf{H}^{(t)}]^{-1} \nabla g^{(t)}$$

Обратное распространение ошибки (градиентный метод)

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



87 iterations
($\eta = 0.5$, $\alpha = 0.9$)

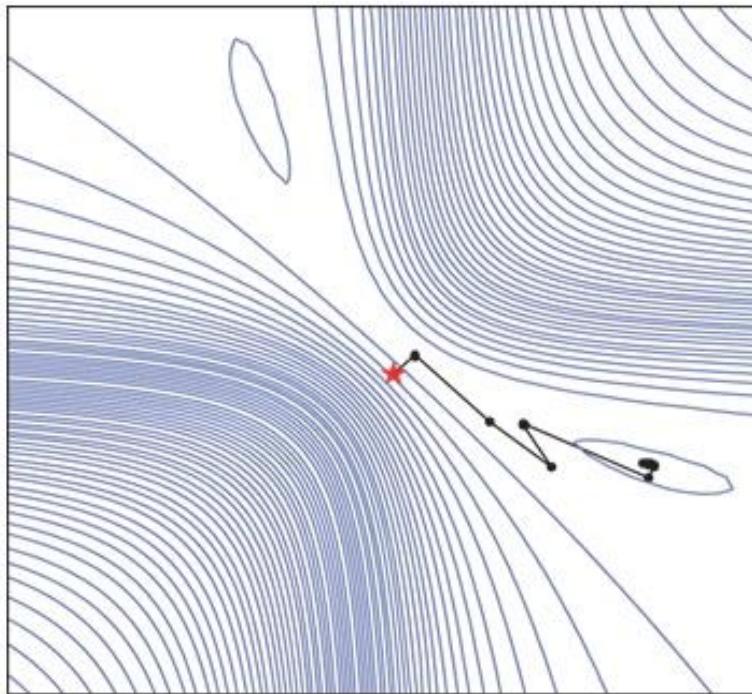


285 iterations
($\eta = 0.1$, $\alpha = 0.9$)

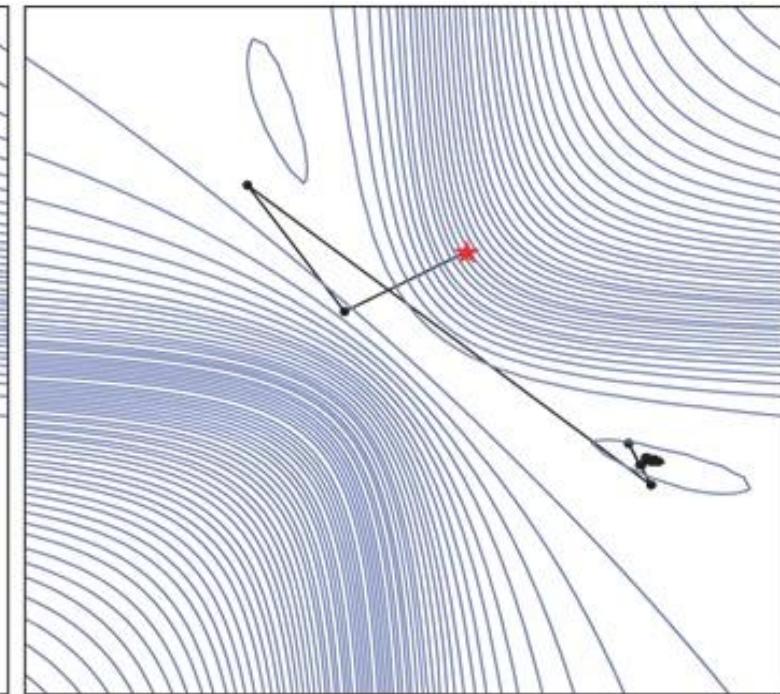
Недостатки: долго, тяжело «угадать» параметры

Быстрое обратное распространение ошибки

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



38 iterations

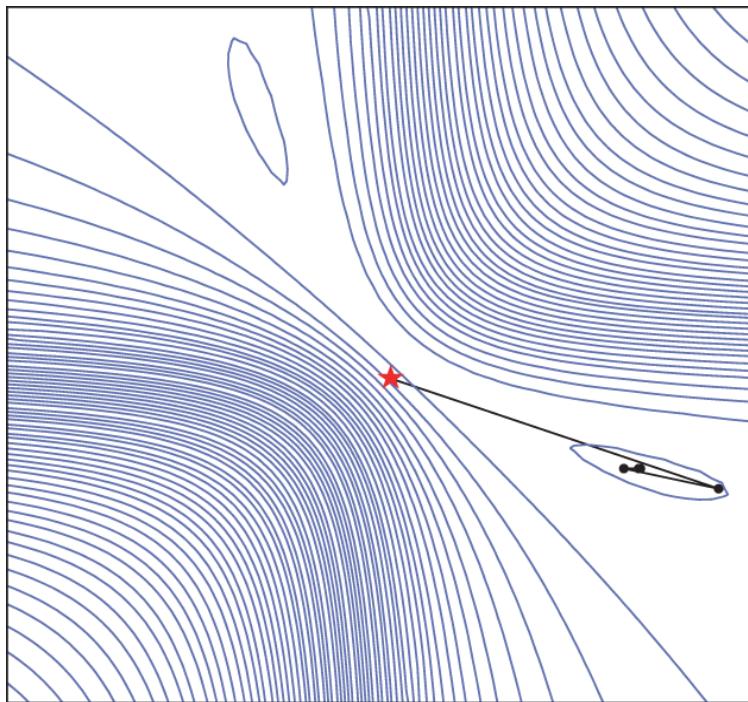


57 iterations

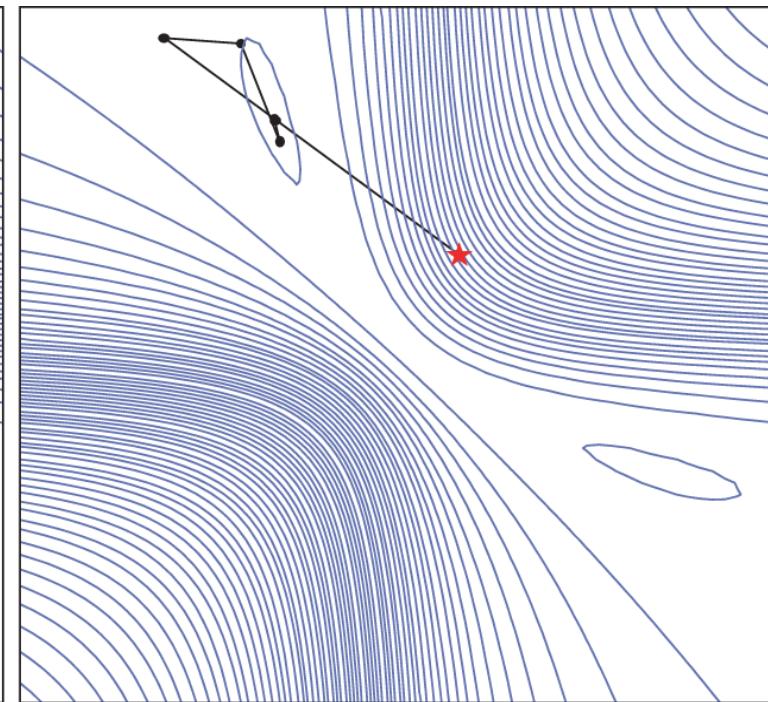
Приближаем функцию ошибки «параболой», вычисляем
диагональ Гессиана «приближенной» функции

Левенберга — Марквардта

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



8 iterations

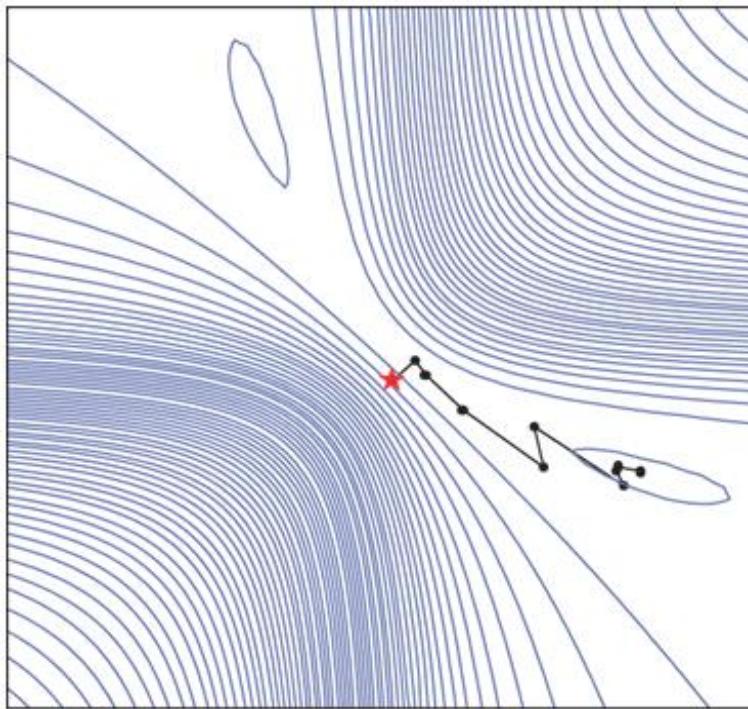


6 iterations

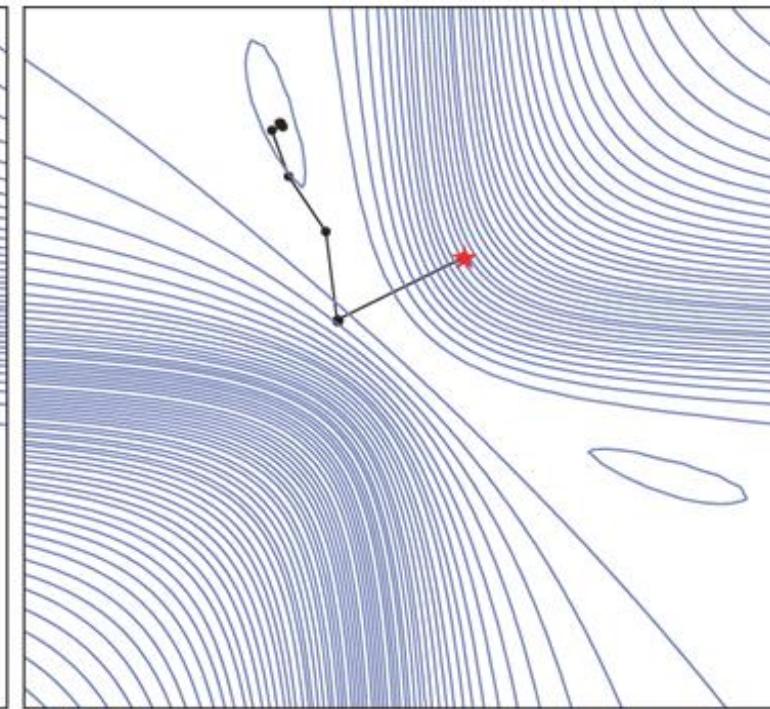
Комбинация градиентного (лямбда велико) и Ньютона (лямбда=0),
Применим для небольшого количества переменных <100

Квазиньютоновские методы

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



11 iterations

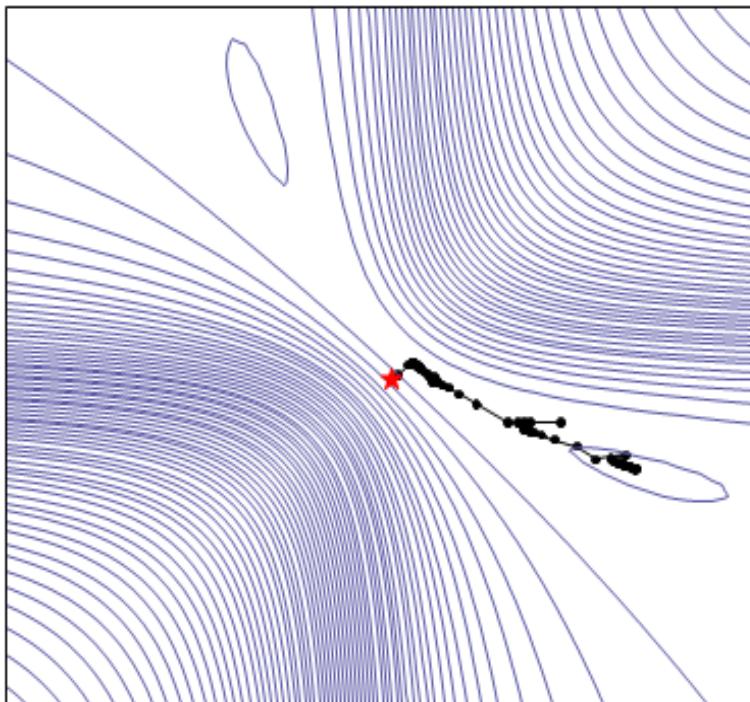


8 iterations

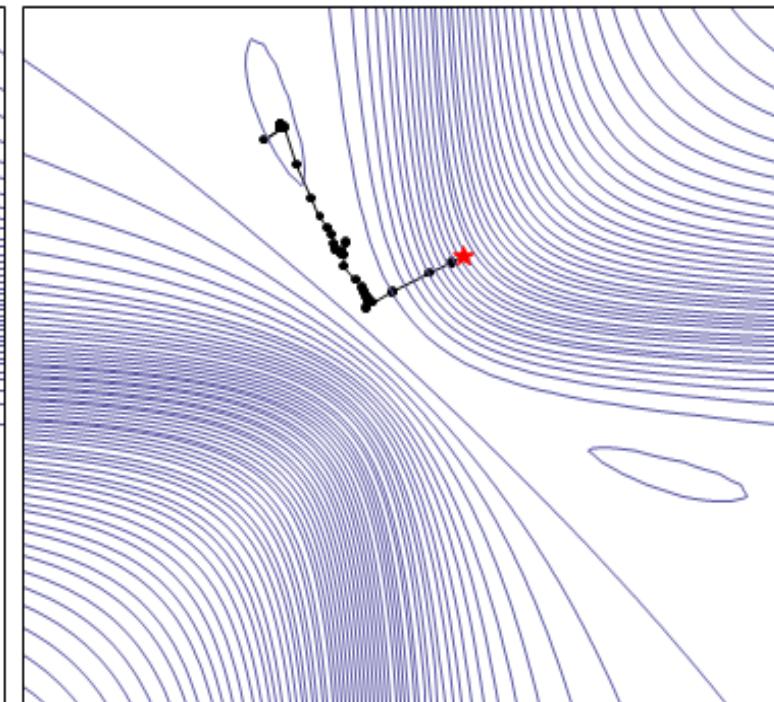
Приближаем H как сумму B и E , обычно E – единичная
Применим для среднего размера задач <500 переменных

Метод Сопряженных градиентов

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



66 iterations

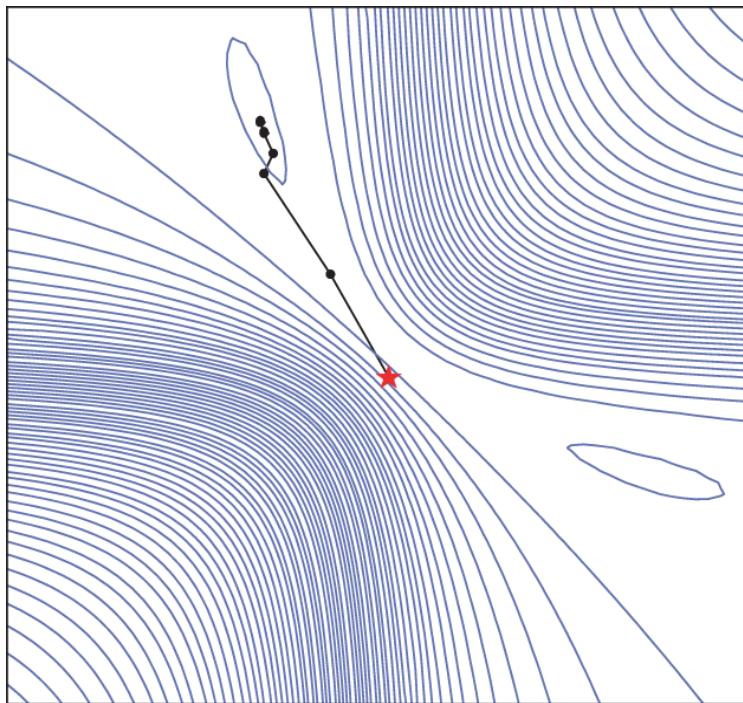


45 iterations

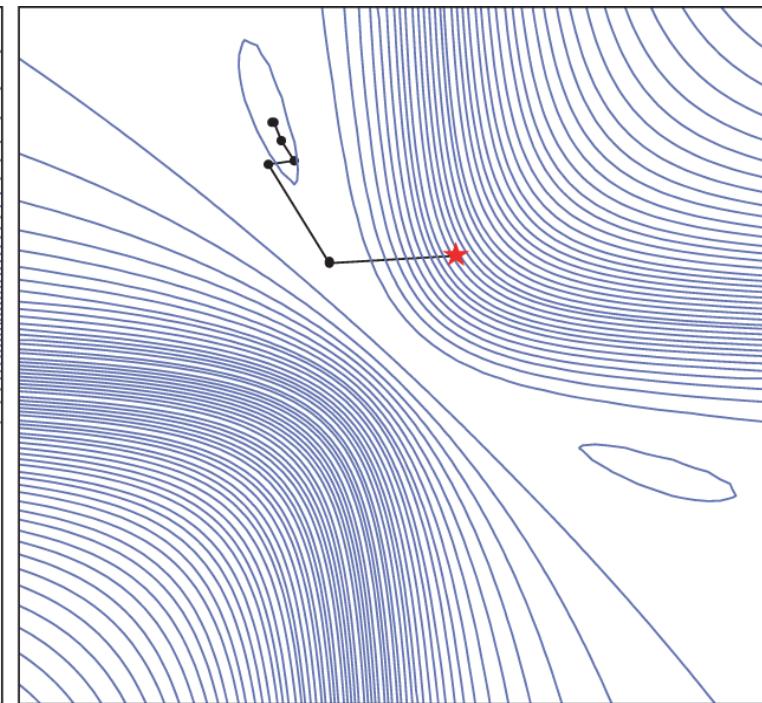
Выбор следующего направления как сопряженного (относительно матрицы Гессе) к предыдущим направлениям шага. Позволяет не рассчитывать H на каждом шаге и работает с большими задачами.

Метод доверительных областей (trusted regions)

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



6 iterations

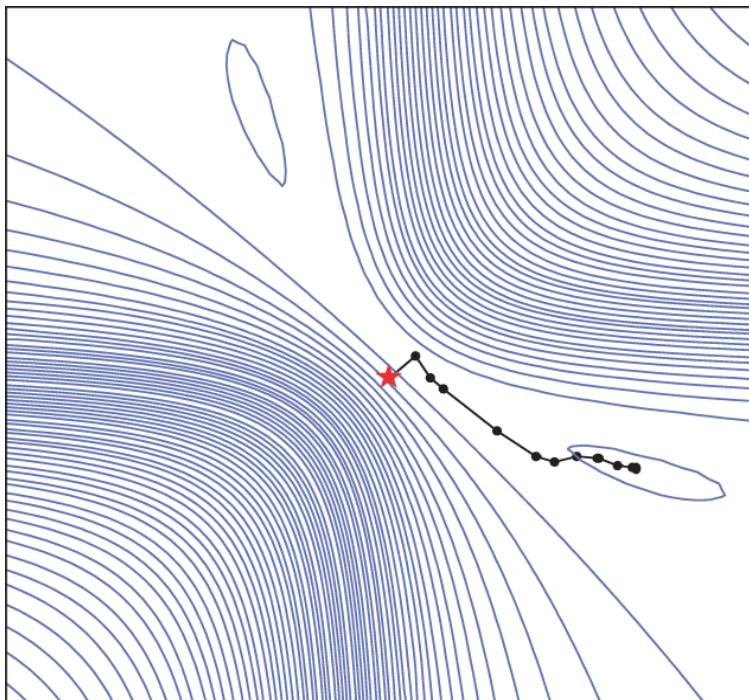


6 iterations

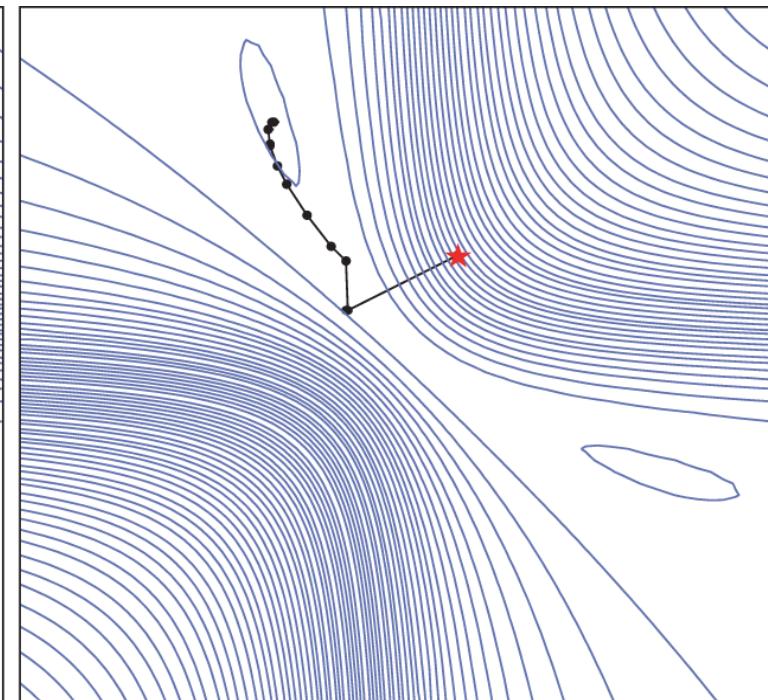
Работает для небольших задач <40, но зато с сильно «не квадратичными» целевыми функциями

Комбинированный (градиент+ニュートン) Double-Dogleg

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

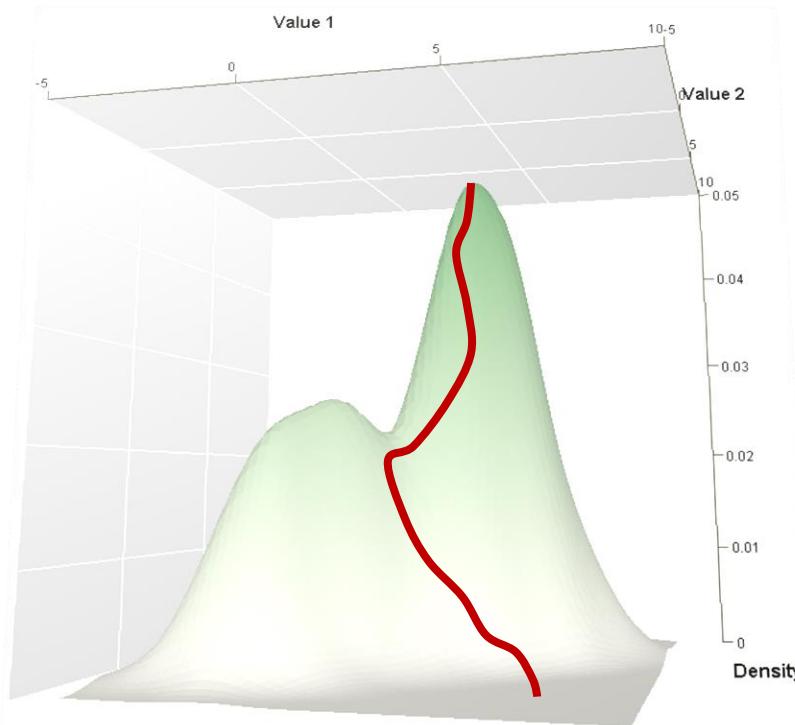


12 iterations



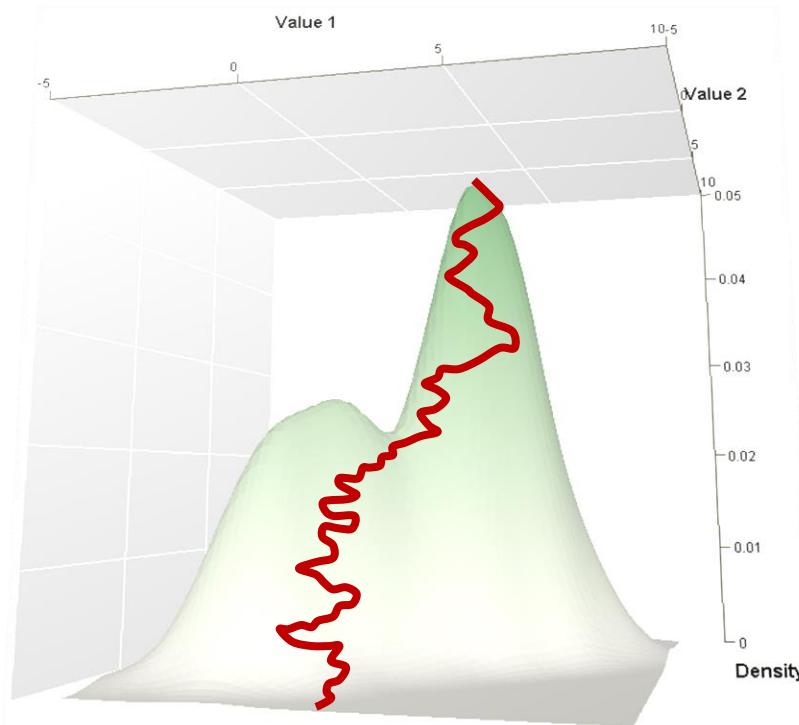
11 iterations

Обычные методы обучения нейросетей



- Используют все наблюдения выборки для расчета градиента и приближения Гессиана
- Результат – гладкая траектория оптимизации
- Но долго по времени и много по памяти

Стохастические и пакетные методы обучения нейросетей



- Стохастические используют только одно наблюдение
- Пакетные – часть наблюдений
- Результат – «хаотическая» траектория (чем больше пакет тем меньше «хаоса»)
- Зато быстро и не нужно все брать из памяти – МРР!!!

Наиболее популярные методы обучения для МРР



1. SDG с моментами и имитацией отжига для скорости обучения:

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

$$\eta' = \frac{\eta}{1 + \beta t}$$

2. Limited-Memory Broyden-Fletcher-Goldfarb-Shanno – пакетный квази-ニュтоновский метод

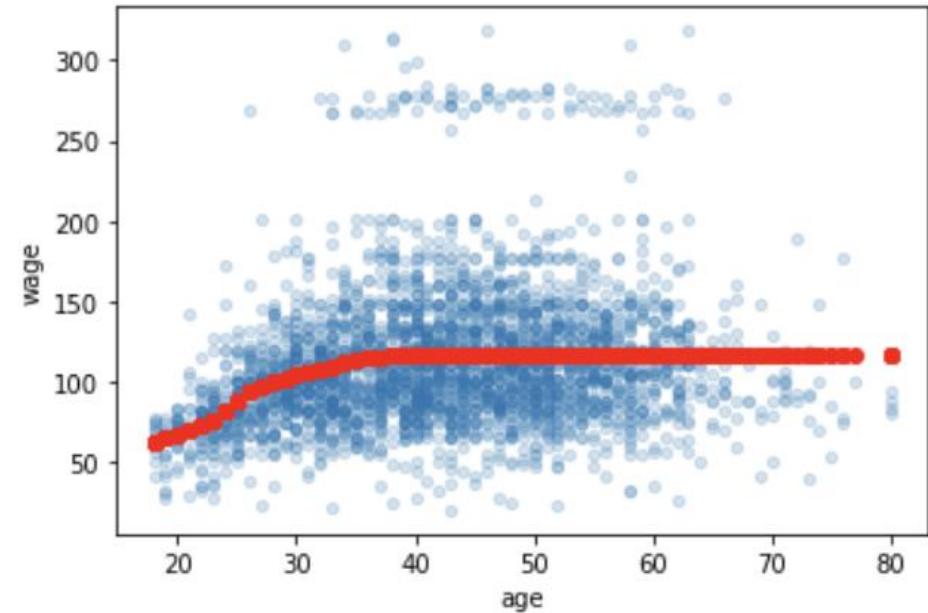
3. Использование L1 и L2 регуляризации

Пример использования MLP

```
from sklearn.neural_network import MLPRegressor

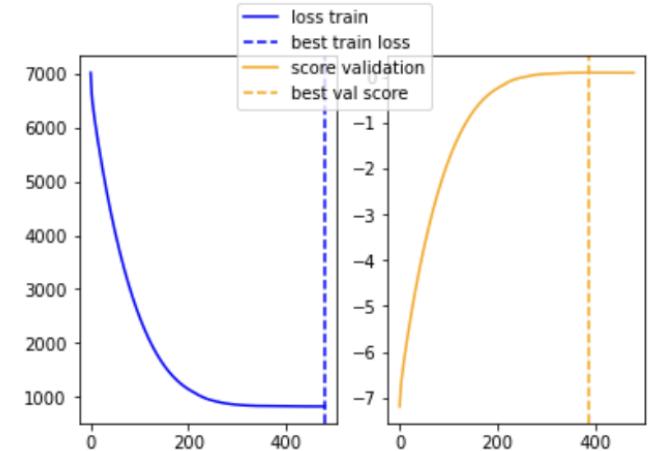
regr = MLPRegressor(
    hidden_layer_sizes=(25, 25),
    activation="tanh",
    solver='adam',
    alpha=1e-3,
    batch_size='auto',
    learning_rate='constant',
    learning_rate_init=0.002,
    tol=1e-4,
    max_iter=3000,
    validation_fraction=0.3,
    early_stopping=True,
    n_iter_no_change=100)
regr.fit(X, y)

df.plot.scatter(x="age", y="wage", alpha=0.2)
plt.scatter(X, regr.predict(X), color="r")
```



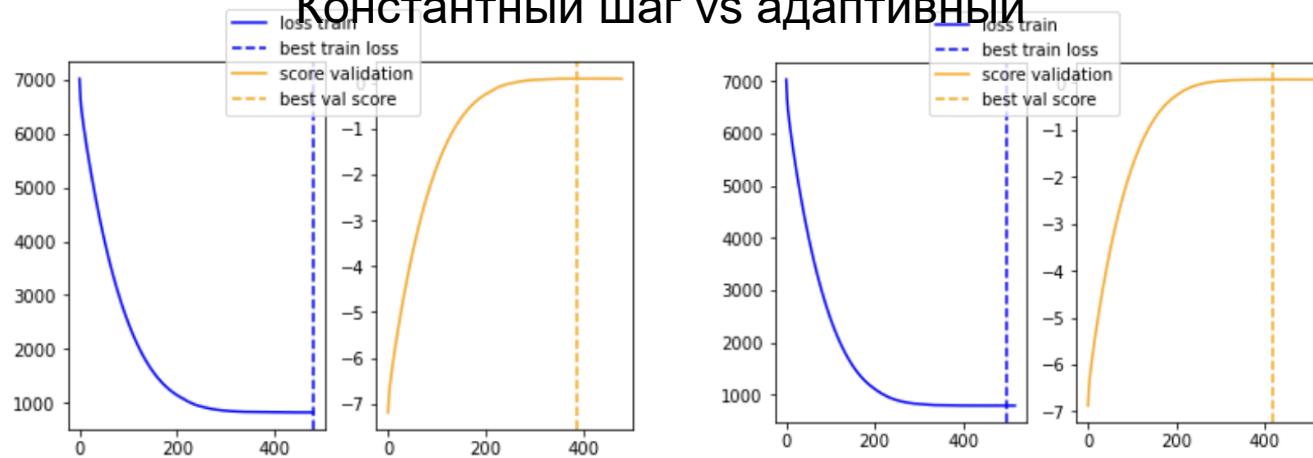
```
fig, axes = plt.subplots(ncols=2)
axes[0].plot(range(regr.n_iter_), np.array(regr.loss_curve_),
             label="loss train", color="blue")
best_tr_iter = np.argmin(regr.loss_curve_)
axes[0].axvline(best_tr_iter, color="blue",
                linestyle="--", label="best train loss")

axes[1].plot(range(regr.n_iter_), regr.validation_scores_,
             label="score validation", color="orange")
best_val_iter = np.argmax(regr.validation_scores_)
axes[1].axvline(best_val_iter, color="orange",
                linestyle="--", label="best val score")
fig.legend(loc="upper center")
```

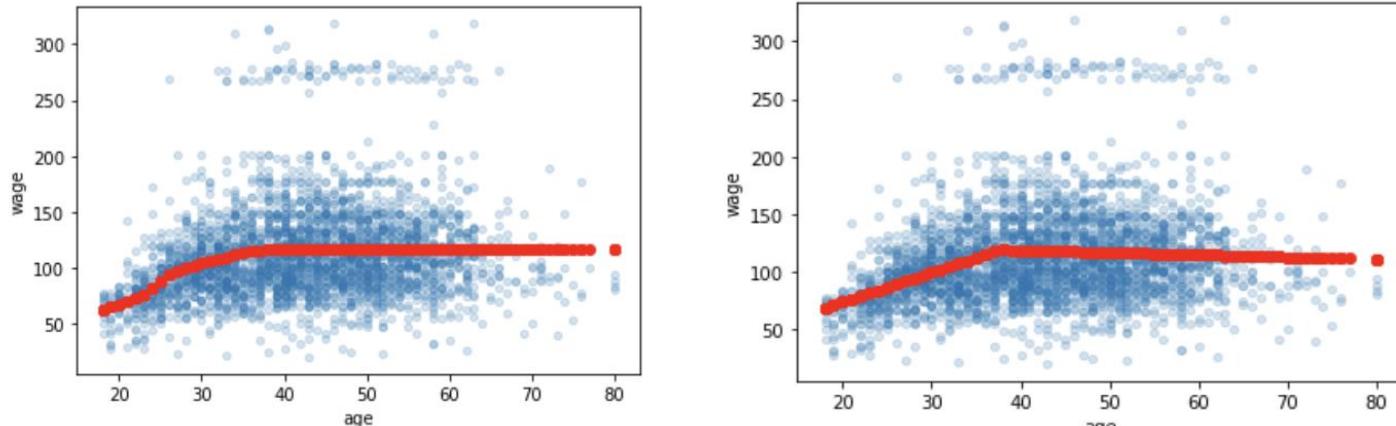


Примеры использования MLP с разными настройками

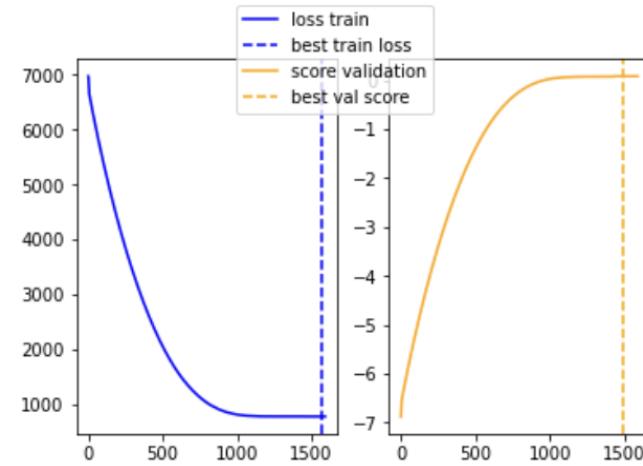
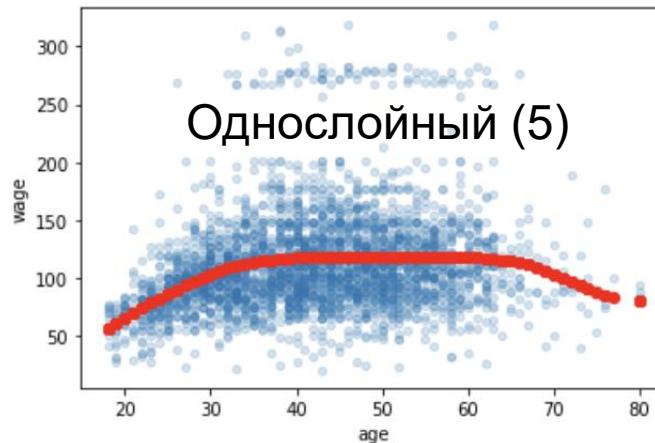
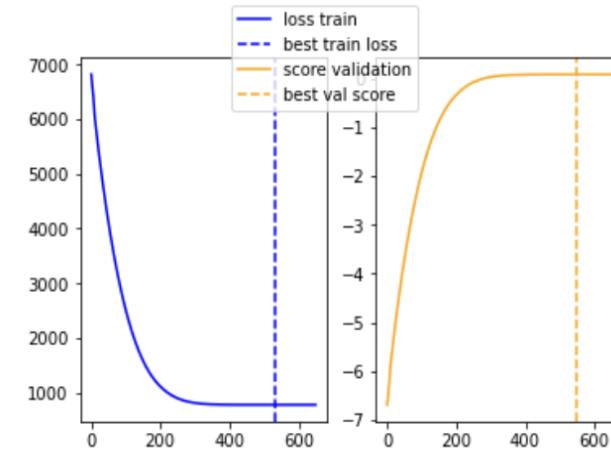
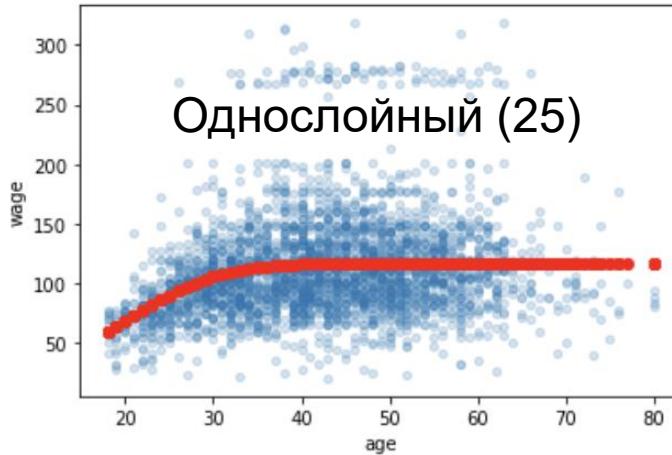
Константный шаг vs адаптивный



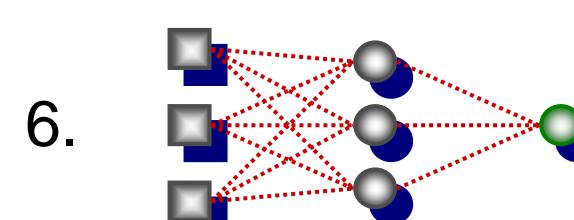
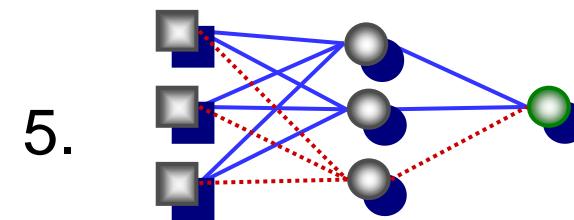
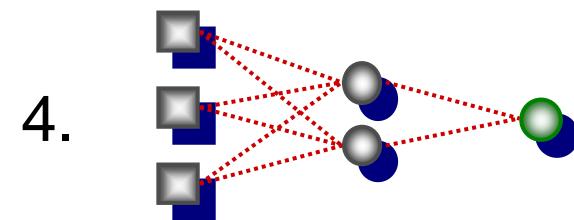
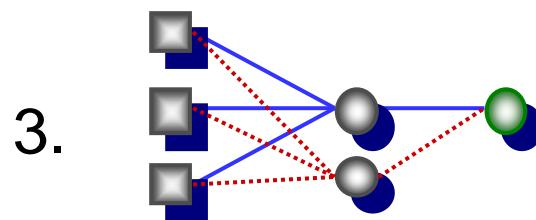
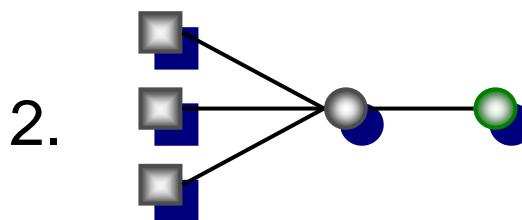
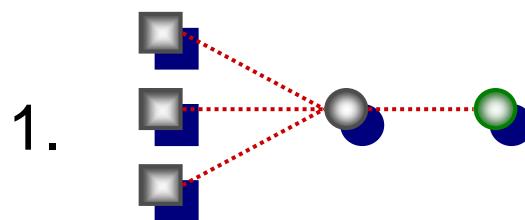
Активация tanh vs relu



Примеры использования MLP с разными настройками



Последовательное построение нейронных сетей



Train:

Freeze:

Глубокое обучение (в общем смысле)

Глубокое 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 **многослойные 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)

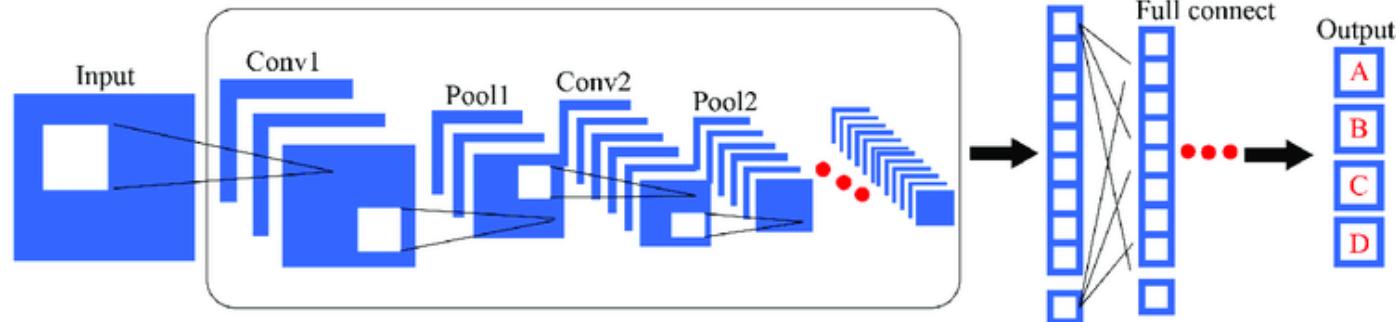
Одна из ключевых особенностей – требуется поэтапное обучение в несколько «проходов» с «заморозкой» коэффициентов части слоев или структур.

Глубокое обучение (в узком «нейросетевом» смысле)

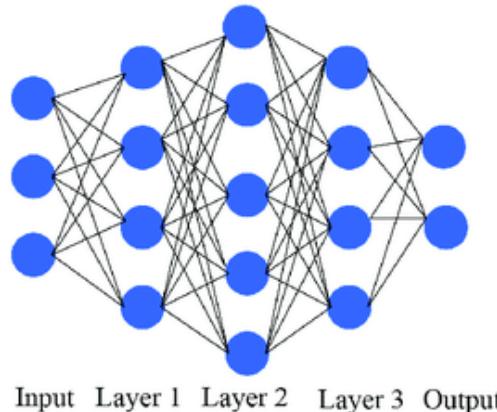
- «Новые» архитектуры (им 30+ лет):
 - Полносвязные многослойные сети (DNN)
 - Сверточные сети (CNN)
 - Рекурентные сети (RNN)
 - Автоэнкодеры и факторизация
- Глубокая кастомизация:
 - «инженерный» подход – много разнотипных уровней в сети
 - считывание сигнала с любого уровня (в задачах выявления признаков бывает важнее чем сигнал на выходе)
 - пользовательские функции активации
- Регуляризация для борьбы с переобучением:
 - Dropout, L1, L2
- Основной тип алгоритмов обучения:
 - стохастический градиентный спуск (SGD) и LBGS, учитывающие массовый параллелизм и ограниченность локальных вычислительных ресурсов
- Расчет на GPU
- Автоматическая подстройка гиперпараметров

Архитектуры нейросетей глубокого обучения в Viya (DLPy)

CNN



DNN



RNN

