Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский физико-технический институт
(государственный университет)»

Факультет управления и прикладной математики

Кафедра проблем передачи информации и анализа данных

# ОБУЧЕНИЕ ГЛУБИННЫХ НЕЙРОННЫХ СЕТЕЙ НА ОСНОВЕ МЕТОДОВ БАЙЕСОВСКОЙ ФИЛЬТРАЦИИ

Выпускная квалификационная работа
(бакалаврская работа)

Направление подготовки: 03.03.01 Прикладные математика и физика

Выполнил:
студент 373 группы _____ Павлов Сергей Владиславович

Научный руководитель:
к.ф.-м.н. _____ Бурнаев Евгений Владимирович

Москва 2017

# Содержание

# 1 Введение

В настоящее время существуют и активно применяются эффективные методы обучения искусственных нейронных сетей(например, метод обратного распространения ошибки). Однако в ряде задач данные методы оставляют открытым вопрос о времени обучения нейронной сети. Причиной такого поведения зачастую являются сугубо вычислительные аспекты: наличие локальных экстремумов, "паралич"сети. Всё это может привести к неограниченному возрастанию времени обучения сети. Поэтому имеет смысл рассматривать принципиально иные подходы к обучению искусственных нейронных сетей. В данной работе речь пойдёт о принципиальной возможности применения методов байесовской фильтрации для обучения нейронных сетей.

Цель работы – разработка алгоритма обучения, основанного на идее байесовской фильтрации и исследование поведения такого алгоритма на конкретных примерах.

Методы исследования:

1. построение алгоритма обучения нейронной сети;
2. написание тестовых программ, реализующих построенный алгоритм;
3. тестирование данных программ на различных выборках;

Объект исследования — методы байесовской фильтрации.
Предмет исследования — искусственные нейронные сети.

# 2 Математический аппарат нейронных сетей

## 2.1 Задача обучения по прецедентам

Для начала рассмотрим общую задачу, которую решает нейронная сеть. Пусть $X$ - множество объектов, $Y$ - множество допустимых ответов, $y^* : X \to Y$ - целевая функция, $X^l = (x_i, y_i)_{i=1}^l$ - обучающая выборка, состоящая из прецедентов $(x_i, y_i)$.

Задача обучения по прецедентам заключается в восстановлении зависимости $y^*$ по выборке $X^l$. [3] Для этого строится решающая функция $a : X \to Y$, которая приближала бы $y^*$ на всём множестве $X$. Эту функцию также называют алгоритмом, т.к. она должна допускать эффективную компьютерную реализацию.

Процесс построения алгоритма называется обучением. Методом обучения называется отображение $\mu : (X, Y)^l \to A$, которое сопоставляет алгоритм произвольной конечной выборке. В задачах обучения по прецедентам всегда есть два этапа:

- этап обучения (метод $\mu$ по выборке $X^l$ строит алгоритм $a = \mu(X^l)$);
- этап применения (алгоритм $a$ для новых объектов $x$ выдаёт ответы $a(x)$).

Одним из классических методов обучения является минимизация эмпирического риска (empirical risk minimization). Для него вводится функция потерь $\mathscr{L}(a, x)$, показывающая величину ошибки алгоритма $a$ на объекте $x$. Ответ $a(x)$ называется корректным, если $\mathscr{L}(a, x) = 0$.

Также вводится функционал качества (или эмпирический риск) алгоритма $a$ на выборке $X^l$:

$$Q(a, X^l) = \frac{1}{l} \sum_{i=1}^{l} \mathscr{L}(a, x_i). \tag{1}$$

Метод заключается в минимизации эмпирического риска на заданной выборке $X^l$ по множеству $A$ алгоритмов заданной модели:

$$\mu(X^l) = arg \min_{a \in A} Q(a, X^l). \tag{2}$$

## 2.2 Определение и устройство нейронной сети

Перейдём непосредственно к нейронным сетям. Процедура, которая используется для процесса обучения, называется алгоритмом обучения.

Единицей обработки информации в нейронной сети является нейрон. Его модель показана на рисунке 1.

Основными элементами нейрона являются:

1. Набор синапсов, характеризующихся весами. Если синапсу $i$, связанному с нейроном $k$ на вход подаётся сигнал $x_i$, то сигнал умножается на вес $w_{kj}$.

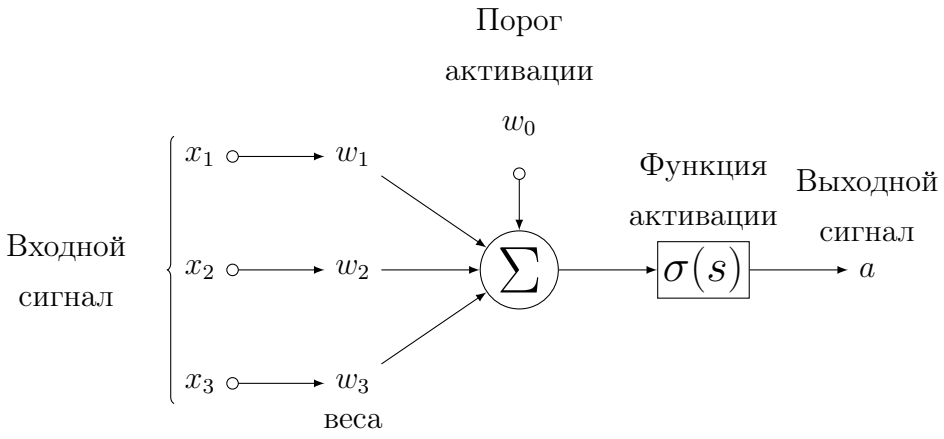



Рис. 1: Структура нейрона

2. Сумматор, который вычисляет линейную комбинацию сигналов путём сложения значений этих сигналов, умноженных на соответствующий вес синапса нейрона.

3. Функция активации, которая позволяет ограничить амплитуду выходного сигнала нейрона. Как правило, после нормировки диапазон амплитуд выхода нейрона находится в интервале [0,1] или [-1,1].

Также в модели нейрона присутствует пороговый элемент (или порог активации), который обозначается $w_0$. Эта величина отражает увеличение или уменьшение входного сигнала, подаваемого на функцию активации. Использование порога обеспечивает эффект афинного преобразования выхода линейного сумматора.

## 2.3 Модель МакКаллока-Питтса

Рассмотрим математическую модель нейрона, предложенную МакКаллоком и Питтсом. [3] Это алгоритм, который принимает на вход вектор описаний признаков $x = (x^1, x^2, ..., x^n)$. Для простоты считаем, что признаки бинарные. Значения признаков - импульсы, поступающие на вход нейрона через синапсы. Эти импульсы складываются с весами $w_1, ..., w_n$, причём синапс называется возбуждающим, если соответствующий ему вес $w_i > 0$, и тормозящим, если $w_i < 0$. Если суммарный импульс превышает заданный порог активации $w_0$, то нейрон подаёт на выход 1, иначе он подаёт 0. Таким образом, нейрон занимается вычислением n-арной булевой функции вида

$$a(x) = \varphi(\sum_{j=1}^{n} w_j x_j - w_0), \tag{3}$$

где $\varphi(z) = [z \geq 0]$ - функция Хэвисайда. Функцию $\varphi$ ещё называют функцией активации.

Также модель можно обобщить на случай произвольных входов и выходов и произвольной функции активации. Наиболее часто используются следующие функции $\varphi(z)$:

- Функция единичного скачка

$$\varphi(z) = \begin{cases} 1, & \text{если } z \geq 0 \\ 0, & \text{если } z < 0 \end{cases} \tag{4}$$

- Кусочно-линейная функция

$$\varphi(z) = \begin{cases} 1, & z \geq +\frac{1}{2} \\ |z|, & -\frac{1}{2} < z < +\frac{1}{2} \\ 0, & z \leq -\frac{1}{2}, \end{cases} \tag{5}$$

- Сигмоидальная функция

$$\varphi(z) = \frac{1}{1 + exp(-\alpha z)}, \tag{6}$$

где $\alpha$ - параметр наклона.

## 2.4 Стохастический градиентный спуск

Исходя из принципа минимизации эмпирического риска, задачу о настройке весов можно свести к задаче минимизации функционала качества:

$$Q(w) = \sum_{i=1}^{l} \mathscr{L}(a(x_i), y_i) \to \min_w, \tag{7}$$

где $\mathscr{L}(a, y)$ - заданная функция потерь, которая характеризует величину ошибки ответа $a$ при правильном ответе $y$. Воспользуемся методом градиентного спуска. Изменение весов на каждой итерации можно записать следующим образом:

$$w := w - \eta \frac{\partial Q}{\partial w}, \tag{8}$$

где $\eta > 0$ - величина шага в направлении антиградиента. В предположении, что $\mathscr{L}$ и $\varphi$ дифференцируемы, распишем градиент:

$$w := w - \eta \sum_{i=1}^{l} \mathscr{L}'_a(a(x_i), y_i)\varphi'(\langle w, x_i \rangle)x_i, \tag{9}$$

Вектор весов $w$ можно изменять после предъявления всех $l$ объектов, каждый из которых вносит свой вклад в $w$, а можно обновлять вектор для каждого объекта. Второй способ называется методом стохастического градиента; при этом объекты перебираются в случайном порядке. Таким образом,

$$w := w - \eta \mathscr{L}'_a(a(x_i), y_i)\varphi'(\langle w, x_i \rangle)x_i, \tag{10}$$

Приведём алгоритм обучения методом стохастического градиента из [3].

**Алгоритм 1:** Обучение персептрона методом стохастического градиента

**Вход :** $X^l$ - обучающая выборка;

$\eta$ - темп обучения;

**Выход:** Синаптические веса $w_0, w_1, ..., w_n$;

инициализируем веса:

$w_j := random(-\frac{1}{2n}, \frac{1}{2n})$;

инициализируем текущую оценку функционала:

$Q := \sum_{i=1}^{l} \mathscr{L}(a(x_i), y_i)$;

**повторять**

выбрать объект $x_i$ из $X^l$ случайным образом

вычислить выходное значение алгоритма $a(x_i)$ и ошибку:

$\varepsilon_i := \mathscr{L}(a(x_i), y_i)$;

сделать шаг градиентного спуска:

$w := w - \eta \mathscr{L}'_a(a(x_i), y_i)\varphi'(\langle w, x_i \rangle)x_i$;

оценить значение функционала:

$Q := \frac{l-1}{l}Q + \frac{1}{l}\varepsilon_i^2$;

**до тех пор, пока** *значение $Q$ не стабилизируется*;

## 2.5 Метод обратного распространения ошибок

На практике часто рассматриваются многослойные нейронные сети, которые, как понятно из названия, состоят из нескольких слоёв нейронов: одного входного, одного выходного и нескольких скрытых. Такие сети применяются для большого числа задач. При этом обучение происходит с помощью метода обратного распространения ошибки (error backpropagation algorithm). [1]

Приведём описание алгоритма из [3]. Рассмотрим полную многослойную сеть (т.е. такую сеть, в которой присутствуют все синаптические связи между нейронами предыдущего слоя и следующего слоя) и положим $X = R^n, Y = R^m$.

Пусть выходной слой состоит из $M$ нейронов с функциями активации $\sigma_m$ и выходами $a^m, m = 1, ..., M$. Перед ним находится скрытый слой из $H$ нейронов с функциями активации $\sigma_h$ и выходами $u^h, h = 1, ..., H$. Веса связей между двумя слоями будем обозначать $w_{hm}$. Перед скрытым слоем находится ещё один слой (распределительный или скрытый) с выходами $v^j, j = 1, ..., J$ и весами $w_{jh}$.

Выходные значения сети на объекте $x_i$ вычисляются как суперпозиция:

$$a^m(x_i) = \sigma_m(\sum_{h=0}^{H} w_{hm}u^h(x_i)); \quad u^h(x_i) = \sigma_h(\sum_{j=0}^{J} w_{jh}v^j(x_i)). \tag{11}$$

Запишем функционал среднеквадратичной ошибки для отдельного объекта $x_i$:

$$Q(w) = \frac{1}{2}\sum_{m=1}^{M}(a^m(x_i) - y_i^m)^2. \tag{12}$$

Выпишем частные производные Q по выходам нейронов для выходного слоя:

$$\frac{\partial Q(w)}{\partial a^m} = a^m(x_i) - y_i^m = \varepsilon_i^m. \tag{13}$$

Теперь продифференцируем $Q$ по выходам скрытого слоя:

$$\frac{\partial Q(w)}{\partial u^h} = \sum_{m=1}^{M}(a^m(x_i) - y_i^m)\sigma'_m w_{hm} = \sum \varepsilon_i^m \sigma'_m w_{hm} = \varepsilon_i^h. \tag{14}$$

Назовём эту величину ошибкой сети на скрытом слое.

Теперь можно выписать градиент $Q$ по весам:

$$\frac{\partial Q(w)}{\partial w_{hm}} = \frac{\partial Q(w)}{\partial a^m}\frac{\partial a^m}{\partial w_{hm}} = \varepsilon_i^m \sigma'_m u^h, \quad m = 1, ..., M, \quad h = 0, ..., H; \tag{15}$$

$$\frac{\partial Q(w)}{\partial u^h} = \frac{\partial Q(w)}{\partial u^h}\frac{\partial u^h}{\partial w_{jh}} = \varepsilon_i^h \sigma'_h v^j, \quad h = 1, ..., H, \quad j = 0, ..., J; \tag{16}$$

и так далее для каждого слоя. Выпишем алгоритм полностью.

**Алгоритм 2:** Обучение двухслойной сети методом обратного распространения ошибки

**Вход :** $X^l = (x_i, y_i)_{i=1}^l$ - обучающая выборка, $x_i \in R^n, y_i \in R^M$;

$H$ - число нейронов в скрытом слое;

$\eta$ - темп обучения;

**Выход:** Синаптические веса $w_{jh}, w_{hm}$;

инициализировать веса небольшими случайными значениями:

$w_{jh} := random(-\frac{1}{2n}, \frac{1}{2n})$;

$w_{hm} := random(-\frac{1}{2H}, \frac{1}{2H})$;

**повторять**

выбрать объект $x_i$ случайным образом

прямой ход:

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

$a_i^m := \sigma_m(\sum_{h=0}^{H} w_{hm} u^h(x_i))$, для всех $m = 1, ..., M$;

$\varepsilon_i^m := a_i^m - y_i^m$, для всех $m = 1, ..., M$;

$Q_i := \sum_{m=1}^{M} (\varepsilon_i^m)^2$

обратный ход:

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

Градиентный шаг:

$w_{hm} := w_{hm} - \eta \epsilon_i^m \sigma'_m u^h$, для всех $h = 0, ..., H, m = 1, ..., M$;

$w_{jh} := w_{jh} - \eta \epsilon_i^h \sigma'_h x^j$, для всех $j = 0, ..., n, h = 1, ..., H$;

$Q := \frac{l-1}{l} Q + \frac{1}{l} Q_i$;

**до тех пор, пока** $Q$ *не стабилизируется*;

## 2.6 Обобщение и проблема переобучения

Одним из недостатков метода обратного распространения ошибки является то, что сеть способна переобучаться, если чрезмерно увеличивать веса. При таком методе обучения в сеть подаётся обучающая выборка, и с помощью алгоритма вычисляются синаптические веса многослойного персептрона. При этом мы надеемся, что полученная сеть способна к обобщению, т.е. для данных из контрольной выборки, которых она никогда ранее не видела, отображение входа на выход будет корректным. Предполагается, что контрольная выборка взята из той же популяции, что и обучающая.

Однако, если сеть обучается на слишком большом количестве примеров, она может "запомнить"примеры обучения. Например, из-за шума она может найти признаки, свойственные обучающей выборке, но не моделируемой функции. В этом случае говорят о переобучении сети. Такие сети теряют способность к обобщению на схожих входных сигналах. [1]

Для улучшения качества и сходимости градиентного обучения, а также для борьбы с переобучением пользуются сокращением весов. [3] Идея заключается в добавлении к функционалу $Q(w)$ штрафного слагаемого с целью ограничения роста абсолютных значений весов:

$$Q_\tau(w) = Q(w) + \frac{\tau}{2}\|w\|^2. \tag{17}$$

Пересчитаем градиент:

$$\frac{\partial Q_\tau(w)}{\partial w} = \frac{\partial Q(w)}{\partial w} + \tau w. \tag{18}$$

Веса будут обновляться по формуле

$$w := w(1 - \eta\tau) - \eta\frac{\partial Q(w)}{\partial w} \tag{19}$$

# 3 Постановка задачи в терминах байесовской фильтрации

## 3.1 Задача байесовской фильтрации

Пусть имеется марковская цепь $\mathbf{x_k}, k = 0, 1, \ldots$, состояния которой $\mathbf{x_k}$ назовём скрытыми переменными. Определим также переменные $\mathbf{z_k}$, которые назовём наблюдаемыми. Пусть при этом выполнены равенства:

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{x}_{k-2}, \ldots, \mathbf{x}_0) = p(\mathbf{x}_k|\mathbf{x}_{k-1})$$

и

$$p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{x}_{k-1}, \ldots, \mathbf{x}_0) = p(\mathbf{z}_k|\mathbf{x}_k)$$

Задача фильтрациии может иметь несколько формулировок.

- Зная текущее наблюдение $\mathbf{z_k}$ и предыдущие, оценить текущее состояние системы $\mathbf{x_k}$;
- Оценка прошлых состояний системы по текущему и предыдущим измерениям;
- Предсказание будущего состояния системы по текущему и предыдущим измерениям;

## 3.2 Общая оптимизационная задача

В данном разделе приведём общую оптимизационную задачу [6], которую затем применим к задаче обучения нейронной сети.

Введём измеримое пространство $(\Omega, \mathcal{F})$ с фильтрацией $\mathcal{F} := (\mathcal{F}_r)_{r=0,1,2,\dots T}, T \in N_+$. Рассмотрим также случайный процесс(управляющий процесс) $\mathbf{a} : \Omega \times \{0, \dots, T-1\} \to A$, где $(A, \mathcal{B})$ - измеримое пространство. Полагаем, что множество допустимых управлений задаётся $\mathcal{A}$. Задавшись управлениями $\mathbf{a} = (\mathbf{a_0}, \mathbf{a_1}, \dots, \mathbf{a_{T-1}}) \in \mathcal{A}$, рассмотрим Марковский процесс X со значениями в некотором измеримом пространстве $(S, \mathcal{S})$ и определённый в вероятностном пространстве $(\Omega, \mathcal{F}, P^{\mathbf{a}})$ с $X_0 = x_0$ п.н. и с заданной вероятностью перехода:

$$\mathbf{P^a}(X_{r+1} \in dy | X_r = x) = P^{a_r}(x, dy), \quad 0 \le r < T.$$

Таким образом, предполагается, что условное распределение $X_{r+1}$ по $\mathcal{F}_r$ управляется вероятностью $P^{a_r}(X_r, dy)$, которая, в свою очередь, управляется $a_r$. В данных терминах сформулируем основную оптимизационную задачу:

$$Y_0^* := \sup_{\mathbf{a} \in \mathcal{A}} E^{\mathbf{a}} \left[ \sum_{r=0}^{T-1} f_r(X_r, a_r) \right],$$

для заданных функций $f_r, \quad r = 0, \dots, T-1$. Можно ещё в большей степени обобщить данную задачу, записав для измеримых функций $f_r : S \times A \to \mathbb{R}, g_r : S \to \mathbb{R}$ и для множества $\mathcal{T} \subset \{0, \dots, T\}$:

$$Y_0^* := \sup_{\mathbf{a} \in \mathcal{A}, \tau \in \mathcal{T}} E^{\mathbf{a}} \left[ \sum_{r=0}^{\tau-1} f_r(X_r, a_r) + g_\tau(X_\tau) \right]. \tag{20}$$

## 3.3 Интерпретация и применение к задаче обучения нейронных сетей

В терминах предыдущего раздела можно интерпретировать задачу обучения следующим образом. Отметим сразу, что "обучение"будет применяться в смысле отличном от привычного.

1. Зафиксируем выход сети Y, для которого будем производить обучение;

2. Пусть в исследуемой сети имеется T слоёв;

3. Будем интерпретировать $X_r$ как вектор значений нейронов на слое r.

4. Будем интерпретировать управляющие параметры переходов $a_r$ как соответствующую пару $(\mathbf{w^r}, \mathbf{b^r})$ - матрицы весов перехода к соответствующему слою и пороги активации на данном слое; Это также случайные величины, которые мы охарактеризуем ниже;

5. Обозначим функцию потерь сети $L(X_{T-1}, Y)$, которая вычисляется для выхода последнего слоя сети;

6. Все функции $f_r$ из формулы (20) равны тождественно нулю;

7. $\mathcal{T}$ состоит из одного элемента $T-1$ и, таким образом в формуле (20) супремум по $\mathcal{T}$ не берётся: $\tau = T-1$. Функция $g_\tau(X_\tau) = L(X_{T-1}, Y)$.

Таким образом, наша задача принимает вид:

$$Y_0^* := \sup_{\mathbf{a}\in\mathcal{A}} E^{\mathbf{a}}\left[L(X_{T-1}, Y)\right]. \tag{21}$$

Обозначим $A_r$ допустимое множество управлений $\mathbf{a}$: $\Omega \times \{r, \dots, T-1\} \to A$. Введём также случайные величины

$$Y_r^* := \sup_{\mathbf{a}\in\mathcal{A}_r} E^{\mathbf{a}}\left[L(X_{T-1}, Y)|X_r\right], \quad 0 \le r \le T. \tag{21}$$

и зададим функции

$$h_r^*(x) = \sup_{\mathbf{a}\in\mathcal{A}_r} E^{\mathbf{a}}\left[L(X_{T-1}, Y)|X_r = x\right], \quad 0 \le r \le T$$

В соответствии с принципом Беллмана(см. [6]), мы можем записать:

$$Y_r^* = h_r^*(X_r).$$

Это означает следующее: если мы знаем зависимость $h_{r+1}^*(x')$ для всех $x'$, то задача поиска $h_r^*(x)$ сводится к оптимизации по единственному параметру $a_r$:

$$h_r^*(x) = \sup_{a_r} \int P^{a_r}(x, dy) h_{r+1}^*(y)$$

.

Таким образом, установлено следующее.
Для каждого выхода Y нейронной сети мы можем последовательно, начиная с последнего слоя и заканчивая первым, настроить параметры перехода так, что для каждого входа сети они будут максимизировать искомое математическое ожидание.

## 3.4 Построение алгоритма обучения

Адаптируем обозначения для большей наглядности их использования для нейронной сети.
Зададимся измеримым пространством $(\Omega, \mathcal{A} = 2^{\Omega})$.

Если на этом пространстве задана случайная величина $Y : \Omega \to (E, \mathcal{B})$, то будем обозначать $\sigma(Y)$ минимальную $\sigma-$алгебру, содержащую семейство множеств $\{\{Y \in B\}, B \in \mathcal{B}\}$. Аналогичное определение имеет место для конечного набора случайных величин. Будем называть её $\sigma-$**алгеброй, порождённой случайной величиной Y**.

Введём также следующие определения:

- Определим последовательности $\mathbf{w}$ и $\mathbf{b}$ векторов: $\mathbf{w} = (\mathbf{w^1}, \mathbf{w^2} \dots)$ и $\mathbf{b} = (\mathbf{b^1}, \mathbf{b^2} \dots)$, каждый из которых имеет свою фиксированную длину;
- Определим последовательность $\mathbf{W}$ независимых случайных матриц $\mathbf{W} = (\mathbf{W^1}, \mathbf{W^2} \dots)$, элементы каждой из которых, выписанные по строкам(сверху вниз) одна за другой в одну строку, образуют многомерную нормально распределённую случайную величину с распределением: $\mathbf{W^i} \sim \mathcal{N}(\mathbf{w^i}, \mathbb{I})$;
- Определим последовательность $\mathbf{B}$ независимых случайных векторов $\mathbf{B} = (\mathbf{B^1}, \mathbf{B^2} \dots)$, таких, что $\mathbf{B^i} \sim \mathcal{N}(\mathbf{b^i}, \mathbb{I})$;
- Определим функцию $\Phi((M, Z), X) = sigmoid(MX + Z) : \mathbb{R}^{lenght(X)} \to \mathbb{R}^{length(Z)}$; Размерности элементов входа должны быть согласованы, но не фиксированны; $sigmoid()$ есть функция – сигмоида.

Пусть на этом пространстве $(\Omega, \mathcal{A})$ задана марковская цепь $(X_0, X_1, X_2, \dots)$, которая определяется следующим образом:

$$X_{i+1} = \Phi((W^{i+1}, B^{i+1}), X_i), i \geq 0$$

$$X_0 = const, p.s.$$

Такое определение, безусловно, накладывает дополонительные требования согласования размерностей величин, определённых выше. Размерности векторов $X_i$ могут также меняться в зависимости от i. Такое соотношение действительно определяет марковскую цепь, хотя и не стационарную.

Для определённой марковской цепи можно ввести понятие фильтрации: последовательность вложенных $\sigma$ - алгебр, определённых следующим образом:

$$\mathcal{F}_i = \sigma(X_0, X_1, \dots X_i).$$

Таким образом, получаем вложенную последовательность $\sigma$-алгебр:

$$\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \dots.$$

Отметим важное свойство фильтрации, которое будет полезно при дальнейшем изложении. Пусть Y есть случайная величина на $(\Omega, \mathcal{A})$ с конечным мат.ожиданием.Тогда верно:

$$\mathbb{E}\left[\mathbb{E}(Y|\mathcal{F}_{i+1})|\mathcal{F}_i\right] = \mathbb{E}\left[Y|\mathcal{F}_i\right]$$

Зададимся нейронной сетью, содержащей T слоёв(занумеруем их $0, 1, 2, 3, \dots T-1$), в каждом по $dim_i$ ($i \in \{0, 1, \dots T-1\}$) нейрона, каждый слой полностью связан со своими соседями. Функцию потерь определим на выходном слое и будем считать квадратичной. Назовём её $L'(X_{T-1}, Y)$. Функцию, обратную по знаку функции потерь, назовём $L = -L'$.

Тогда, зафиксировав $X_0$, можно интерпретировать $X_i$ как состояние слоя сети с номером i. С учётом определений, данных выше, становится понятно, что работа сети управляется марковским процессом, который в свою очередь управляется наборами случайных величин $\mathbf{W}$ и $\mathbf{B}$ , законы распределения которых в свою очередь зависят от $\mathbf{w}$ и $\mathbf{b}$.

Это позволяет нам ставить вопрос обучения в следующем ключе: **выбрать значения параметров w и b таким образом, чтобы условное математическое ожидание $\mathbb{E}\left[L'(X_{T-1}, Y(X_0))|X_0\right]$ было минимальным при каждом значении входа** $X_0$. Формально мы можем написать так:

$$\inf_{\mathbf{w},\mathbf{b}} \mathbb{E}\left[L'(X_{T-1}, Y(X_0))|X_0\right] = -\sup_{\mathbf{w},\mathbf{b}} \mathbb{E}\left[L(X_{T-1}, Y(X_0))|X_0\right]$$

С учётом свойства, указанного в конце предыдущей части, мы можем переписать данную задачу в следующем виде:

$$Z_0(X_0) = \sup_{\mathbf{w},\mathbf{b}} \mathbb{E}\left[L(X_{T-1}, Y(X_0))|X_0\right] = \sup_{\mathbf{w},\mathbf{b}} \mathbb{E}\left[L(X_{T-1}, Y(X_0))|\mathcal{F}_0\right] =$$
$$= \sup_{\mathbf{w},\mathbf{b}} \mathbb{E}\left[\mathbb{E}\left[L(X_{T-1}, Y(X_0))|\mathcal{F}_1\right]|\mathcal{F}_0\right]$$

Введём обозначения:

$$Z_r(X_r) = \sup_{\mathbf{w},\mathbf{b}} \mathbb{E}\left[L(X_{T-1}, Y(X_0))|\mathcal{F}_r\right],$$

$$\mathbf{w_r} = (\mathbf{w^{r+1}}, \dots \mathbf{w^{T-1}}),$$

$$\mathbf{b_r} = (\mathbf{b^{r+1}}, \dots \mathbf{b^{T-1}}),$$

$$r \in \{0 \dots T-1\}$$

При этом становится понятно, что

$$Z_r(X_r) = \sup_{\mathbf{w},\mathbf{b}} \mathbb{E}\left[L(X_{T-1}, Y(X_0))|\mathcal{F}_r\right] = \sup_{\mathbf{w_r},\mathbf{b_r}} \mathbb{E}\left[L(X_{T-1}, Y(X_0))|\mathcal{F}_r\right].$$

При данных обозначениях, продолжим цепочку равенств, которою начали выше:

$$Z_0(X_0) = \sup_{\mathbf{w},\mathbf{b}} \mathbb{E}\left[\mathbb{E}\left[L(X_{T-1}, Y(X_0))|\mathcal{F}_1\right]\mathcal{F}_0\right] = \sup_{\mathbf{w^1},\mathbf{b^1}} \mathbb{E}\left[Z_1(X_1)|X_0\right]$$

Итак, мы получили следующее:

$$Z_0(X_0) = \sup_{\mathbf{w^1},\mathbf{b^1}} \mathbb{E}\left[Z_1(X_1)|X_0\right],$$

где

$$Z_r(X_r) = \sup_{\mathbf{w_r},\mathbf{b_r}} \mathbb{E}\left[L(X_{T-1}, Y(X_0))|\mathcal{F}_r\right]$$

.

Продолжая рекуррентно цепочку равенств, получаем:

$$Z_0(X_0) = \sup_{\mathbf{w^1},\mathbf{b^1}} \mathbb{E}\left[\sup_{\mathbf{w^2},\mathbf{b^2}} \mathbb{E}\left[\sup_{\mathbf{w^3},\mathbf{b^3}} \mathbb{E}[\ldots \sup_{\mathbf{w_{T-1}},\mathbf{b_{T-1}}} \mathbb{E}\left[Z_{T-1}(X_{T-1})|X_{T-1})\right]\ldots|X_1\right]|X_0\right] =$$

$$= \sup_{\mathbf{w^1},\mathbf{b^1}} \mathbb{E}\left[\sup_{\mathbf{w^2},\mathbf{b^2}} \mathbb{E}\left[\sup_{\mathbf{w^3},\mathbf{b^3}} \mathbb{E}\left[\ldots \sup_{\mathbf{w^{T-1}},\mathbf{b^{T-1}}} \mathbb{E}\left[L(X_{T-1}, Y(X_0))|X_{T-2}\right]\ldots|X_3\right]|X_1\right]|X_0\right]$$

Как видно из последнего равенства, мы можем обучать нашу сеть(в терминах поставленной задачи) поэтапно, однако начиная с конца. Последовательность действий будет следующей.

1 Решаем задачу $\sup_{\mathbf{w^{T-1}},\mathbf{b^{T-1}}} \mathbb{E}\left[L(X_{T-1}, Y(X_0))|X_{T-2}\right]$:

- Находим зависимости минимизирующих параметров от значения $X_{T-2}$: $\mathbf{w^{T-1}_{opt}}(\mathbf{X_{T-2}}), \mathbf{b^{T-1}_{opt}}(\mathbf{X_{T-2}})$
- Объявляем $P_{T-2}(X_{T-2}) := \sup_{\mathbf{w^{T-1}},\mathbf{b^{T-1}}} \mathbb{E}\left[L(X_{T-1}, Y(X_0))|X_{T-2}\right]$

2 Решаем задачу $\sup_{\mathbf{w^{T-2}},\mathbf{b^{T-2}}} \mathbb{E}\left[P_{T-2}(X_{T-2})|X_{T-3}\right]$:

- Находим зависимости минимизирующих параметров от значения $X_{T-3}$: $\mathbf{w^{T-2}_{opt}}(\mathbf{X_{T-3}}), \mathbf{b^{T-2}_{opt}}(\mathbf{X_{T-3}})$
- Объявляем $P_{T-3}(X_{T-3}) := \sup_{\mathbf{w^{T-2}},\mathbf{b^{T-2}}} \mathbb{E}\left[P_{T-2}(X_{T-2})|X_{T-3}\right]$

$\ldots$ .....

$(T-1)$ Решаем задачу $\sup_{\mathbf{w^1},\mathbf{b^1}} \mathbb{E}\left[P_1(X_1)|X_0\right]$:

- Находим зависимости минимизирующих параметров от значения $X_0$: $\mathbf{w^1_{opt}}(\mathbf{X_0}), \mathbf{b^1_{opt}}(\mathbf{X_0})$
- Объявляем $Z_0(X_0) = P_0(X_0) := \sup_{\mathbf{w^1},\mathbf{b^1}} \mathbb{E}\left[P_1(X_1)|X_0\right]$

## 3.5 Практическая формулировка и реализация алгоритма

- Значения $X_0$ и $Y_0(X_0)$ берутся из тренировочного набора данных;
- На шаге $i \in [1, \dots, T-1]$ оптимизационная задача решается перебором по следующей схеме:
  - Сгенерировать набор $\{X^j_{T-i-1}\}_{j=0}^N$ из гиперкуба $[0,1]^{dim(X_{T-i-1})}$;
  - Для каждого из $X^j_{T-i-1}$ генерируем по M значений параметров $\mathbf{w_{jk}^{T-i-1}}, \mathbf{b_{jk}^{T-i-1}}$, $k \in \{1, \dots, M\}$ из ограниченного гиперкуба(с базой [-2,2]);
  - Для каждой пары параметров $\mathbf{w_{jk}^{T-i-1}}, \mathbf{b_{jk}^{T-i-1}}$ получаем K реализаций случайных величин: $B^{T-i-1}_{jkl}, W^{T-i-1}_{jkl}, l = 1, \dots, K$
  - Вычисляем эмперическое мат.ожидание по этим K значениям:

$$h^{*,emp}_{T-i-1,jk}(X^j_{T-i-1}) := \frac{1}{K}\sum_{l=1}^{K} h^*_{T-i}(\sigma(W^{T-i-1}_{jkl})X^j_{T-i-1} + B^{T-i-1}_{jkl})$$

  - Выбираем пару параметров $w^{T-i-1}_{jk}, b^{T-i-1}_{jk}$, которая даёт максимум мат.ожидания в предыдущем пункте; обозначим их соответственно $w^{*,T-i-1}_j, b^{*,T-i-1}_j$;
  - Строим регрессионную модель для зависимости оптимальных параметров $w^{*,T-i-1}_j, b^{*,T-i-1}_j$ от значения $X^j_{T-i-1}$; данную модель стараемся строить линейной(линейная регрессия);
- Результатом обучения является последовательность регрессионных моделей, которые отображают текущее состояние очередного слоя $T-i-1$ сети в оптимальные параметры переходов $w^{*,T-i-1}, b^{*,T-i-1}$
- При вычислении прогноза будем полагать случайные величины $\mathbf{W^i}, \mathbf{B^i}$ равными своим мат.ожиданиям $\mathbf{w^i(X_{i-1})}, \mathbf{b^i(X_{i-1})}$, полученным последовательно по данному входу $X_0$;
- Чтобы сделать прогноз по заданному входу $X_0$ необходимо последовательно вычислить: $\mathbf{w^1(X_0)}, \mathbf{b^1(X_0)}, \mathbf{X_1}, \mathbf{w^2(X_1)}, \mathbf{b^2(X_1)}, \mathbf{X_2} \dots \mathbf{X_{T-1}}$

## 3.6 Согласование обучений для различных выходов сети

Напомним, что до сих пор описанный алгоритм позволяет производить обучение сети для заданного фиксированного выхода Y. Однако, поскольку интерес состоит как раз в том, чтобы строить прогнозы, необходимо описать процедуру, позволяющую соединять воедино результаты обучения сети для различных выходов. Далее, в разделе описания экспериментов мы предложим несколько таких процедур объединения.

# 4 Эксперименты

## 4.1 Задача регрессии

Отметим сразу три главные отличия архитектуры нейронной сети для задачи регрессии и задачи классификации.

1. В задаче регрессии выход последнего слоя сети берётся без функции-активатора. Это позволяет сети приближать широкий спектр значений, а не только значения из ограниченного интервала(например, $[0, 1]$).

2. В задаче классификации выходной слой $T - 1$ всегда имеет единичную размерность;

3. Функция потерь для задачи регрессии берётся квадратичная:

$$L(X_{T-1}, Y) = ||X_{T-1} - Y||^2,$$

а в задаче классификации берётся логарифмическая функция потерь:

$$L(X_{T-1}, Y) = -(1 - Y)log(1 - X_{T-1}) - Ylog(X_{T-1}).$$

При этом в силу сказанного выше для классификации имеет место:

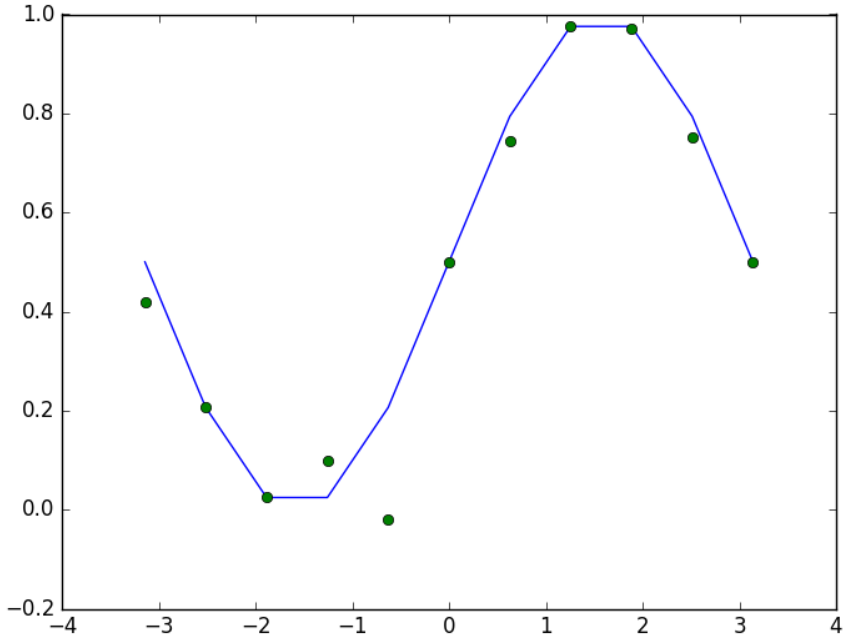
$$Y \in \{0, 1\}, X_{T-1} \in [0, 1].$$

Зададимся функцией $(1 + \sin x)/2$ на отрезке $[-\pi, \pi]$.

Поймём, насколько качественно наш алгоритм способен строить приближения. Для этого применим обучения для различных Y к соответствующим им входам $X_0$. То есть каждому Y соответствует свой $X_0$. Обучив сеть для Y, применим его к $X_0$. Получим одну зелёную точку на графике. Сплошная линия — целевая функция. Пример для N = 15 , M = 15, K = 5, регрессионные модели в обучениях переходов сети применяются линейные. Архитектура сети: [1,2,2,2,2,2,2,1].

Мы видим, что в основном обучение для каждой отдельно взятой точки довольно точно запоминает целевую функцию. Безусловно, поскольку процесс обучения рандомизирован, результаты при перезапуске программы получаются немного различные. Но в целом данный график правильно отражает основную тенденцию.

Теперь было бы интересно получить процедуру, которая позволила бы объединять результаты нескольких обучений воедино и получать прогнозы для входов вне обучающей выборки.




### 4.1.1 Простое усреднение и линейная регрессия

Предлагается сделать следующее.

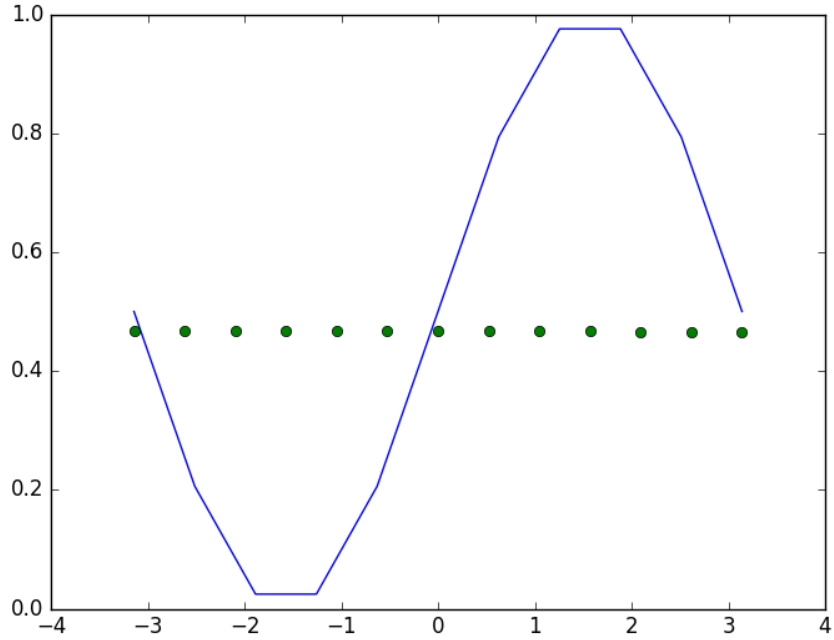
- Получить обучения(последовательности линейных моделей на слоях) для различных выходов сети: $\{M^{Y(X)} = (M_0^{Y(X)}, \ldots, M_{T-2}^{Y(X)})\}_{(X,Y) \in TrainSet}$;
- С помощью этих моделей, последовательно из применяя начиная с X, получить для каждой пары $(X, Y)$ последовательность $(X_0^{(X,Y)} = X, X_1^{(X,Y)}, \ldots, X_{T-1}^{(X,Y)})$
- Для каждого отдельно взятого слоя i произвести обучение новой линейной регрессионной модели для зависимости коэффиентов перехода от входа данного слоя; иными словами, настраиваем регрессию на парах:

$$\{(X_i^{(X,Y)}, (w^{i,(X,Y)}, b^{i,(X,Y)}))\}_{(X,Y) \in TrainSet}$$

  .

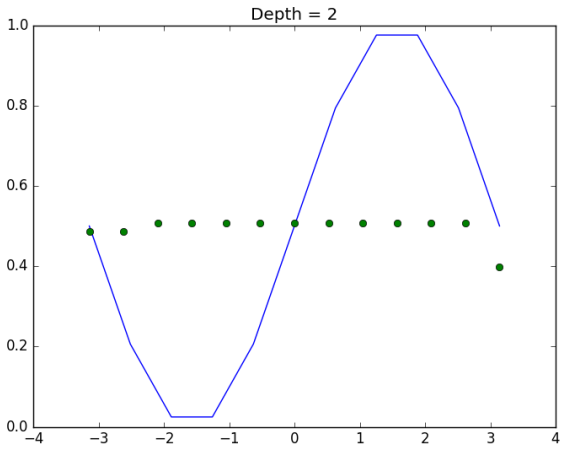
- Полученная регрессионная модель считается финальной для данного слоя;

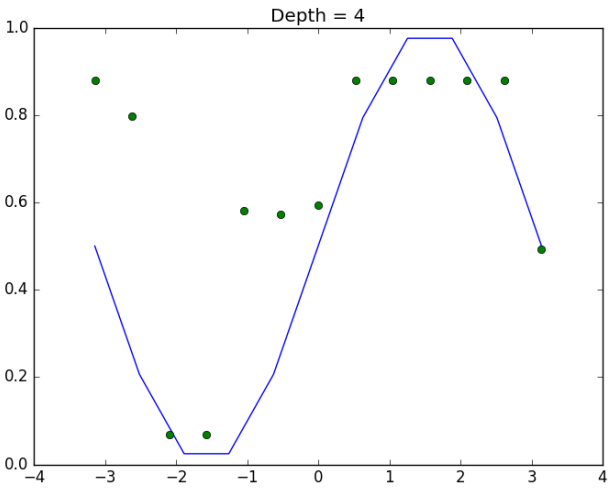
Было установлено, что если в качестве регрессионной модели выбирать линейную(или обычное усреднений, например), то полученное обучение уничтожает всякую информацию об обучающей выборке.



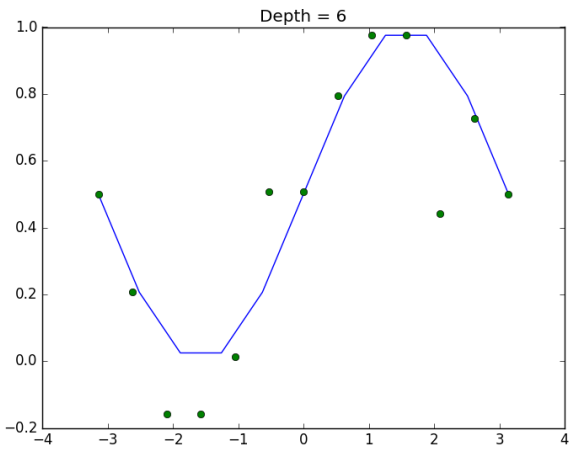

### 4.1.2 Нелинейная регрессия

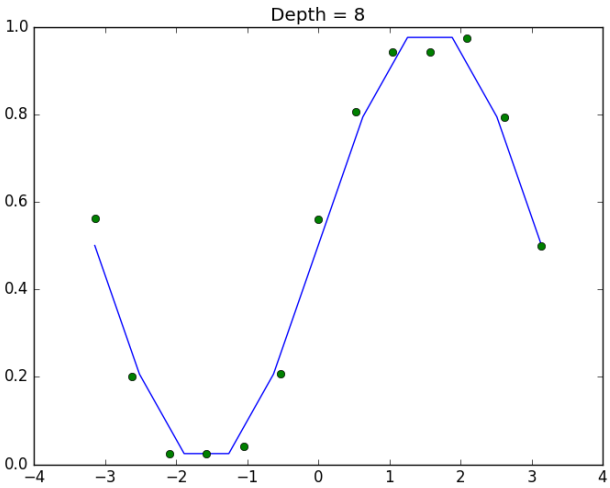
Однако ситуацию можно исправить, использовав в качестве регрессионной модели более сложные. Мы будем использовать решающие деревья различной глубины. Из графиков видно, что с ростом глубины решающих деревьев качество прогнозов растёт.




Стоит также отметить, что, поскольку в задаче регрессии существует множество возможных выходов Y, проведение процесса обучения для каждого из них оказывается затратным по времени. В то же время для задачи классификации выходов всего два: 0 и 1. Далее мы займёмся исследованием задачи классификации и разработкой процедуры объединения результатов для неё.

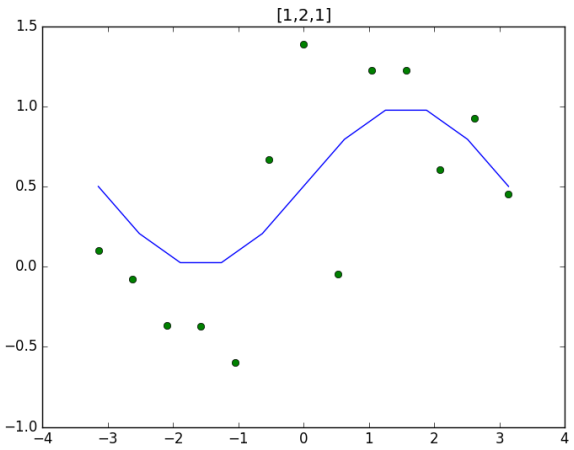


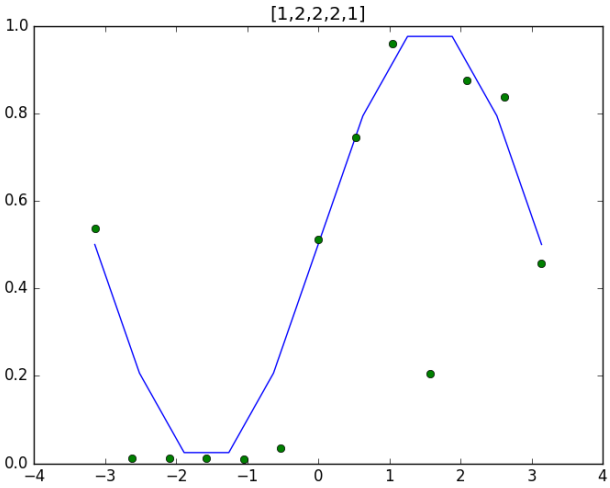



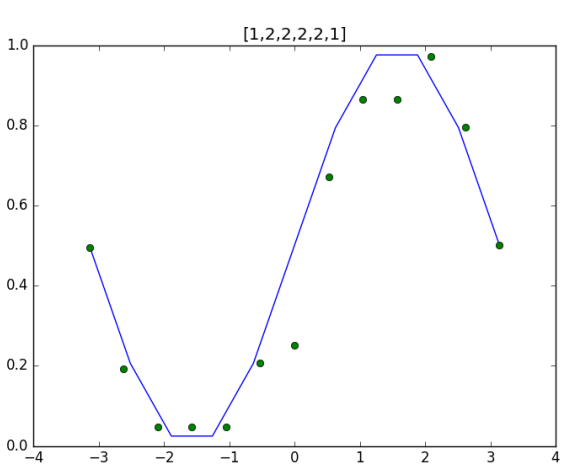

### 4.1.3 Зависимость результата от архитектуры сети

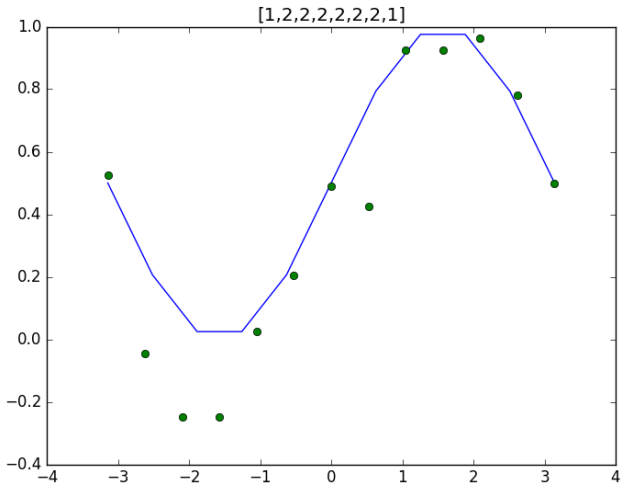
В ходе экспериментов была замечена зависимость качества прогнозирующей способности от количества слоёв нейронной сети. Зафиксируем параметры N = 15 , M = 15, K = 5). Исследуем, как количество слоёв влияет на качество обучения. Глубину решающего возьмём равной 5.

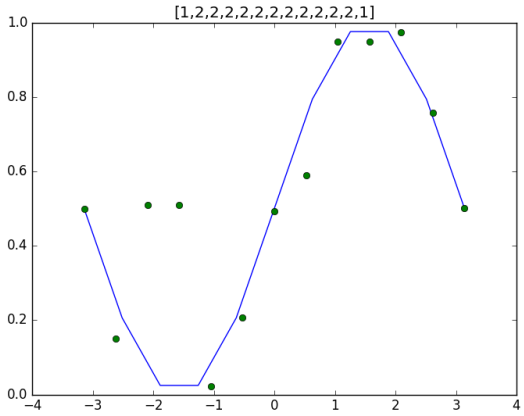
Можно заметить, что с ростом глубины сети качество прогноза сначала улучшается, а затем начинает деградировать.






## 4.2 Классификация

Перейдём к задаче классификации. Существует ряд причин, по которым задача классификации может быть более перспективной и интересной в рамках нашего исследования, чем задача регрессии.

- Процесс обучения повтояется только дважды: для 0 и для 1;
- Биполярность обучения может иметь понятную интерпретацию и быть использована для построения метода объединения обучений для 0 и для 1;

### 4.2.1 Подход 1

Одной из наших задач будет создание метода обобщения, который по возможности не использовал бы нелинейных регрессионных моделей.
Применим сперва подход, который применялся для задачи регрессии. За тем исключением, что процесс обучения производится только дважды: для 0 и для 1.

Сформулируем **алгоритм**.

**Дано:** $\{(X_r, Y(X_r))\}_{r=1}^{Q}$ - обучающая выборка;

**Выход:** $M^* = (M_1^*, \ldots, M_{T-2}^*)$;

1: Получить обучение для 0: $M^- = (M_1^-, \ldots, M_{T-2}^-)$
2: Получить обучение для 1: $M^+ = (M_1^+, \ldots, M_{T-2}^+)$
3: $S_i := \emptyset, i \in \{0, 1, \ldots, T-2\}$
4: **для** r=1,. . . ,Q
5: **если** $Y(X_r) = 1$ **то**
6: $M = (M_1, \ldots, M_{T-2}) := M^+$
7: **иначе**
8: $M = (M_1, \ldots, M_{T-2}) := M^-$
9: $\widetilde{X_0} := X_r$
10: **для** i = 1, . . . , T-1
11: $(w^{i-1}, b^{i-1}) := M_{i-1}(\widetilde{X_{i-1}})$
12: $\widetilde{X_i} := \sigma\left(w^{i-1}\widetilde{X_{i-1}} + b^{i-1}\right)$
13: Добавить в $S_{i-1}$ пару $\left(\widetilde{X_{i-1}}, (w^{i-1}, b^{i-1})\right)$
14: **для всех** $i = 0, \ldots, T-2$
15: Обучить классификатор $M_i^*$ на парах из множества $S_i$

### 4.2.2 Подход 2

В данном подходе применим идею биполярности входных данных. Этот метод может быть также интересен в качестве метода сведения многомерной классификации к двумерной.

Сформулируем **алгоритм**.

**Дано:** $\mathcal{X} = \{(X_r, Y(X_r))\}_{r=1}^{Q}$ - обучающая выборка;

**Выход:** Классификатор C;

1: $X_{zeros} := \{X_r | Y(X_r) = 0, r \in \{0, \ldots, Q\}\}$
2: $X_{ones} := \{X_r | Y(X_r) = 1, r \in \{0, \ldots, Q\}\}$
3: Для $X_{ones}$ применить Подход 1 и получить обобщающую последовательность моделей $\mathbf{M}^{*+}$;
4: Для $X_{ones}$ применить Подход 1 и получить обобщающую последовательность моделей $\mathbf{M}^{*+}$;
5: $S := \emptyset$
6: **для** $r \in \{1, \ldots, Q\}$
7: $\widetilde{X_0} := X_r$
8: **для** $i \in \{1, \ldots, T-1\}$
9: $(w^{i-1}, b^{i-1}) := M_{i-1}^{*+}(\widetilde{X_{i-1}})$
10: $\widetilde{X_i} := \sigma\left(w^{i-1}\widetilde{X_{i-1}} + b^{i-1}\right)$
11: $\overline{X_0} := X_r$

12: **для** $i \in \{1, \ldots, T-1\}$

13: $(w^{i-1}, b^{i-1}) := M^{*-}_{i-1}(\overline{X_{i-1}})$

14: $\overline{X_i} := \sigma\left(w^{i-1}\overline{X_{i-1}} + b^{i-1}\right)$

15: Поместить пару $\left((\overline{X_{T-1}}, \widetilde{X_{i-1}}), Y(X_r)\right)$ в множество S;

16: Обучить классификатор C на множестве двумерных векторов из S;

В качестве нелинейного двумерного классификатора будем брать полиномиальный SVM со степенью полинома не ниже трёх.

Основным наблюдением является то, что для каждой полученной пары $\mathbf{M}^{*-}$ и $\mathbf{M}^{*+}$ получается своя уникальная геометрия распределения классов в думерном пространстве(см. рисунки в разделе "Эксперименты"). Недостатком является то, что часто такая геометрия получается слишком сложной, а поэтому очень сложно настроить двумерный классификатор. Для каждого набора данных геометрия получается своя. Поэтому степень полиномов для SVM берётся каждый раз разная.

## 4.3 Тестирование алгоритмов

Для тестирования описанных подходов возьмём 6 наборов данных(для бинарной классификации)из общедоступного реестра:
http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html

1. "diabetes"
2. "breast-cancer"
3. "fourclass"
4. "german.numer"
5. "heart"
6. "liver-disorders"

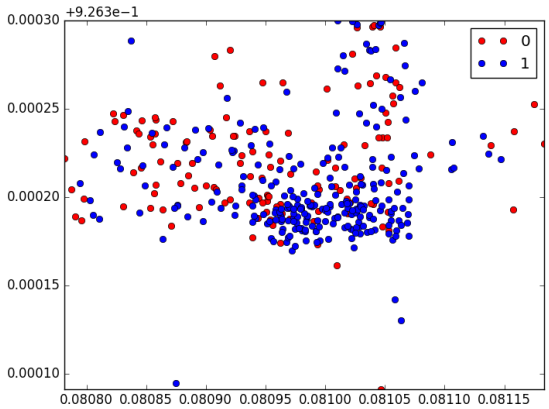
Зафиксируем архитектуру сети: $[1, 5, 5, 5, 5, 5, 1]$.

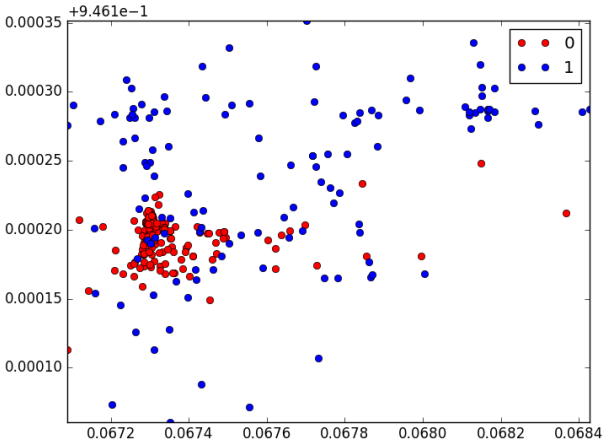
Данные берутся нормированные на $[-1, 1]$ для каждой компоненты входа.

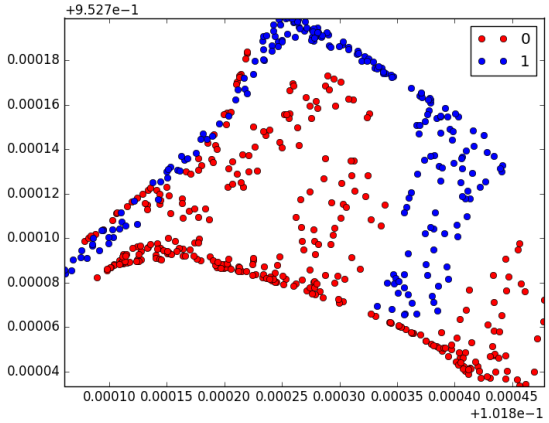
Для каждого из 100 запусков программы данные делятся случайным образом на тренировочную и тестовую подвыборки(для каждого набора своего фиксированного размера - см.таблицу). Строятся обучения при помощи описанных методов, а также берётся нейронная сеть в традиционном смысле такой же архитектуры. Результаты сравниваются.

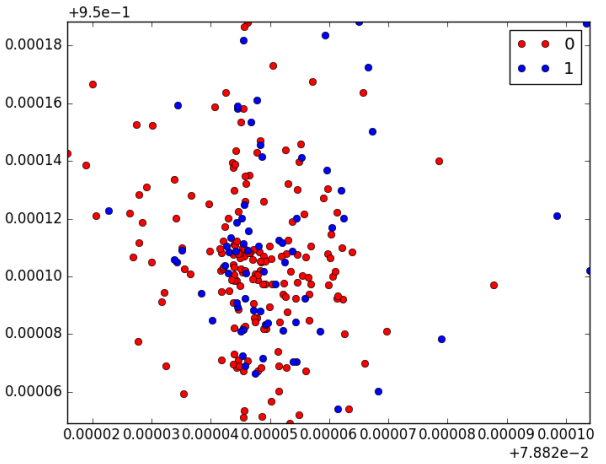
В качестве критерия оценки эффективности подхода берётся точность предсказания на тестовой выборке.

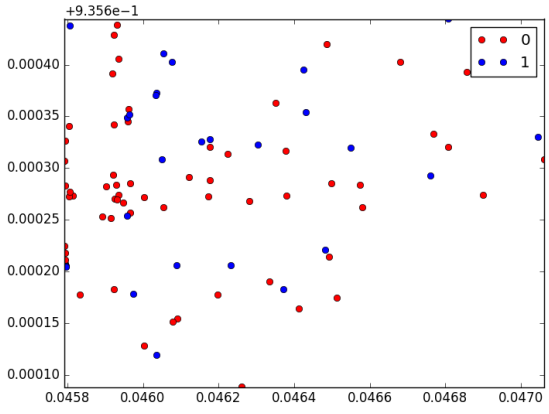
Определимся, какие степени полиномов будем брать для реализации второго подхода. Для этого посмотрим на типичную структуру распределения точек в двумерном пространстве.

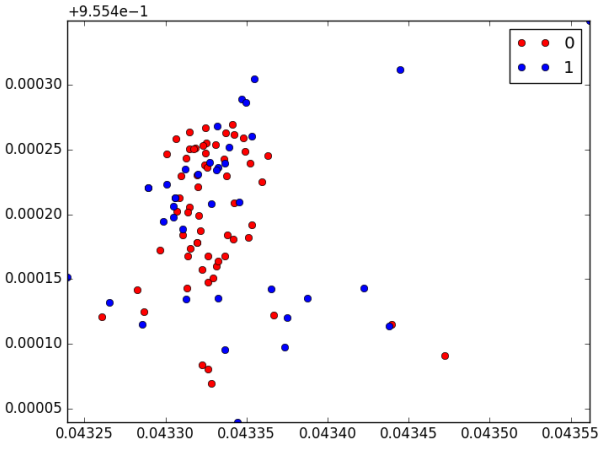



Логика такова: если прослеживается структура, то степень полинома берётся меньшей, поскольку "запоминать"частных случаев нужно меньше. Если же структура

отсутствует, то следует экспериментировать. В таких случаях степень полинома выбирается эмпирически. Для последующих экспериментов мы выбрали следующие степени: 4, 5, 3, 5, 7, 5 (в порядке нумерации наборов данных).

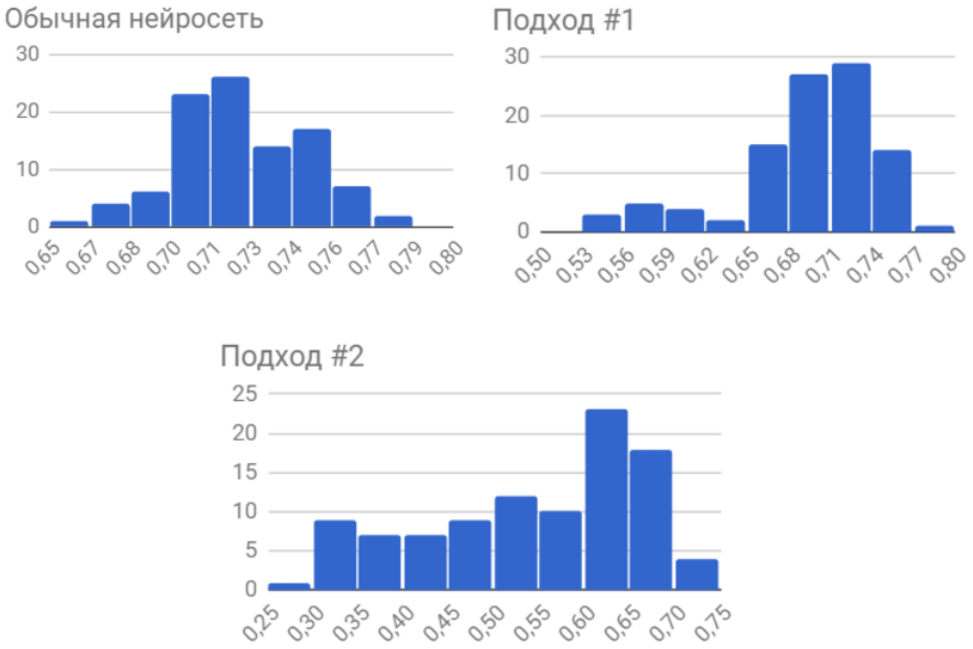
Приведём сводную таблицу для выборочных средних значений и выборочных дисперсий, а также гистограммы распределения значений для обычной нейросети, и для подходов 1 и 2 для различных наборах данных.

Сводная таблица:

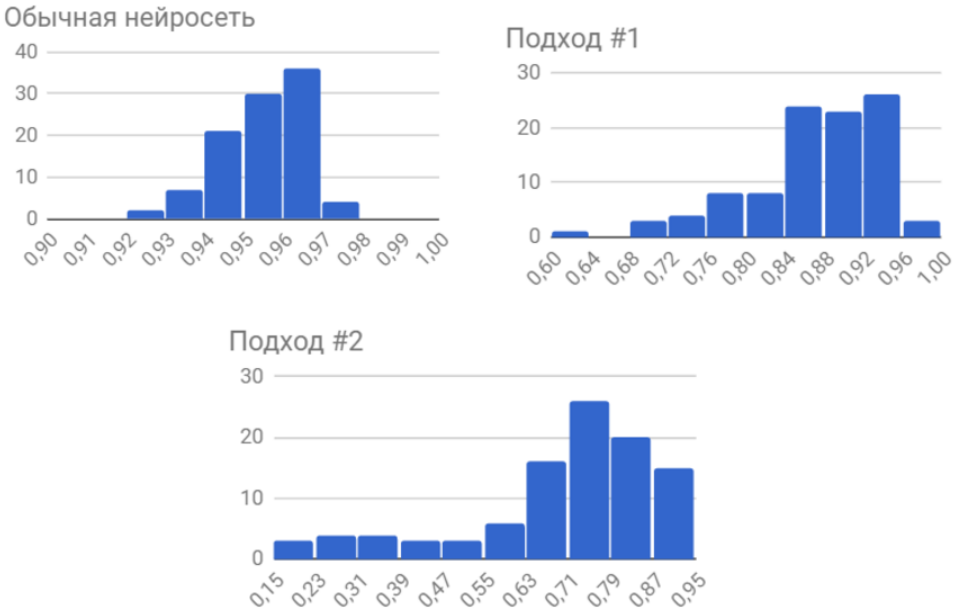
| Набор данных | Среднее(о/с) | Дисперсия(о/с) | Среднее(#1) | Дисперсия(#1) | Среднее(#2) | Дисперсия(#2) | Размер обуч. выб | Размер тест. выб |
|---|---|---|---|---|---|---|---|---|
| 1 | 0,72 | 0,0006 | 0,69 | 0,003 | 0,54 | 0,014 | 400 | 368 |
| 2 | 0,96 | 0,0001 | 0,87 | 0,004 | 0,7 | 0,04 | 350 | 333 |
| 3 | 0,93 | 0,005 | 0,69 | 0,004 | 0,58 | 0,02 | 550 | 312 |
| 4 | 0,73 | 0,006 | 0,68 | 0,009 | 0,51 | 0,02 | 900 | 100 |
| 5 | 0,78 | 0,003 | 0,67 | 0,01 | 0,52 | 0,02 | 170 | 100 |
| 6 | 0,68 | 0,004 | 0,63 | 0,01 | 0,58 | 0,02 | 100 | 45 |

Гистограммы для различных наборов:

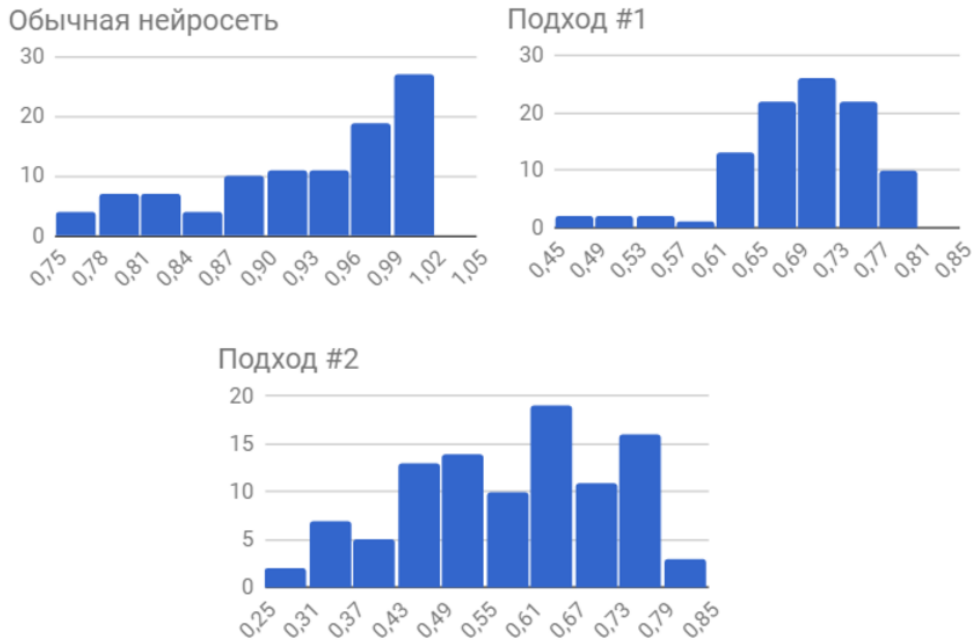
### 4.3.1 "diabetes"



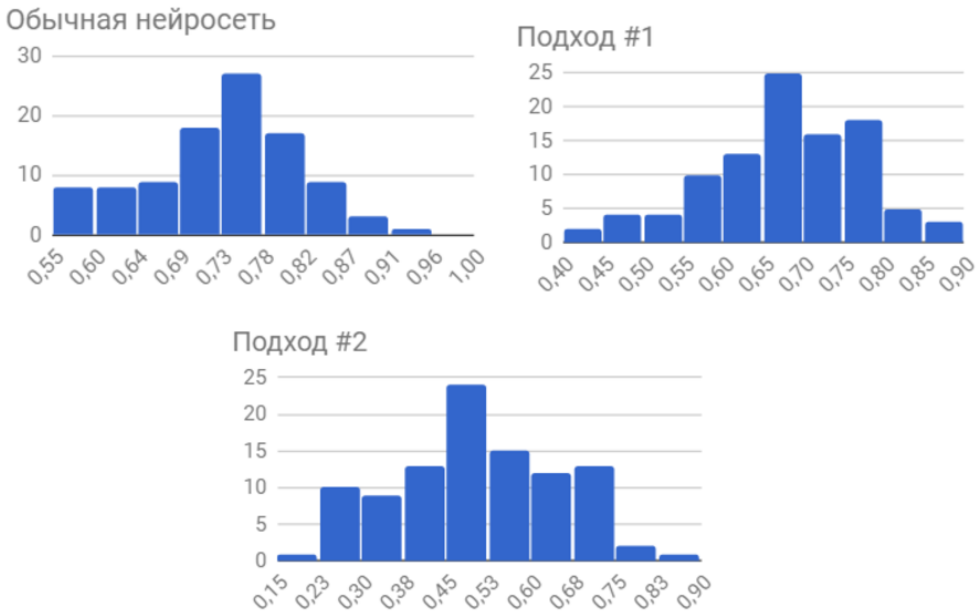
### 4.3.2 "breast-cancer"



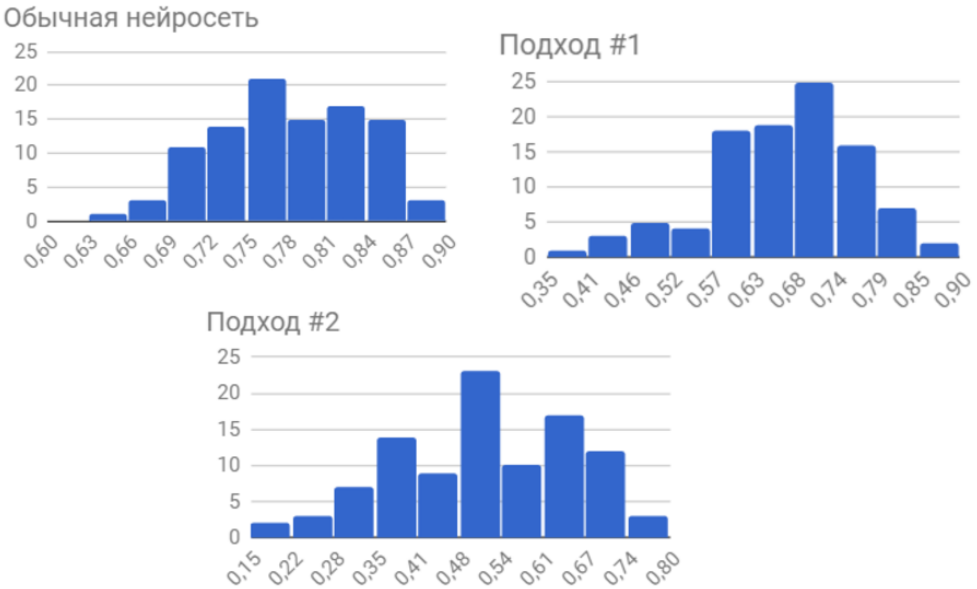

### 4.3.3 "fourclass"



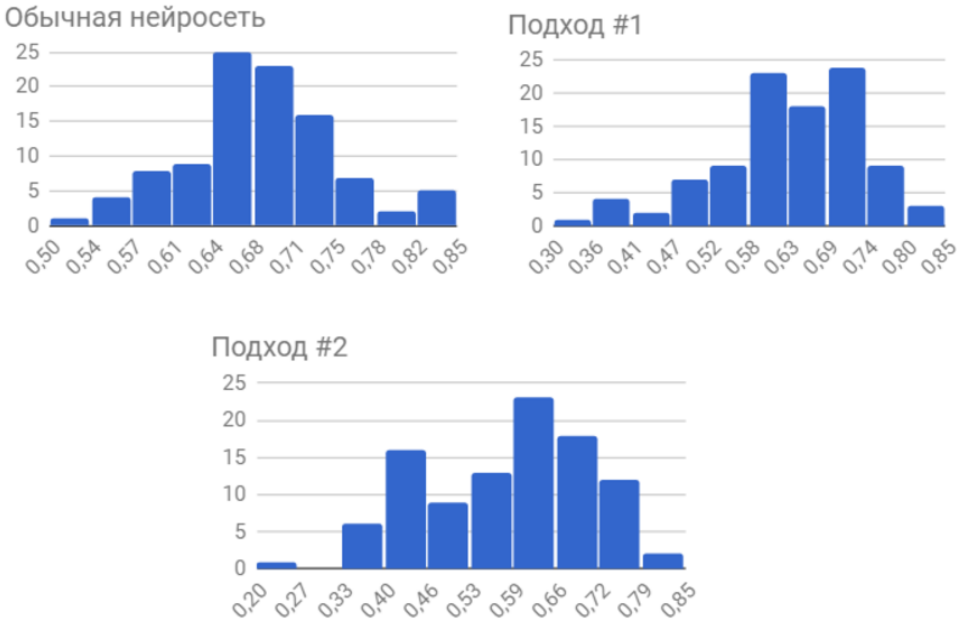
### 4.3.4 "german.numer"




### 4.3.5 "heart"



### 4.3.6 "liver-disorders"




### 4.3.7 Выводы

Мы видим, что первый подход выдаёт результаты, близкие к результатам обычной нейросети такой же архитектуры. Второй подход ему уступает в производительности.

Примечательно, что все вычисления сделаны для крайне небольших значениях параметров N, M, K: $N = 15, M = 15, K = 5$.

## 4.4 Эффективность вычислений

При указанных выше параметрах($N = 15, M = 15, K = 5$) и архитектуре сети $(X, 5, 5, 5, 5, 5, 1)$ время работы обоих подходов составляет 15с. Это существенно больше времени обучения традиционной нейронной сети. Обычный алгоритм обратного распространения ошибки работает порядка 0.5с.

Детальный анализ кода показал, что большая часть времени тратится на выполнение операций умножения строки на матрицу и преобразование(reshape) вектора в матрицу.

Было установлено, что:

- Время, уходящее на вычисление значения состояния следующего слоя по значению предыдущего равно $10^{-4}$;
- Для вычисления потери одного входа на предпоследнем слое нужна 1 такая операция, для слоя с номером 4 - 2 операции. Каждый раз количество операций увеличивается на 1 с уменьшением номера слоя;

- Поэтому, зная параметры M,N,K, можно оценить время работы алгоритма как $(1+2+3+4+5+6)*M*N*K*10^{-4} = 21*15*15*5*10^{-4} = 2.3c$
- На практике время работы равно 15с; но при оценке мы многое не учли. Например, время на рекуррентные вызовы функций(а обучение основано на рекурсии);

Предложим пути решения этой проблемы:

- Переписать код на языке C++ с целью оптимизации времени вычислений;
- Использование параллельных вычислений: зная результаты обучения на последующих слоях, потери можно вычислять параллельным образом для нескольких входов текущего слоя;

# 5 Заключение

В ходе работы мы:

- Построили алгоритм обучения нейронных сетей на основе метода байесовской фильтрации;
- Реализовали алгоритм на языке python;
- Проделали ряд экспериментов на задачах регрессии и классификации и произвели сравнительный анализ предложенных методов.

Таким образом, цель работы достигнута.
В дальнейшем планируется усовершенствование подходов для задачи классификации, тестирование его работы с применением более значительных вычислительных ресурсов, а также оптимизация эффективности его реализации.

# Список литературы


[1] Хайкин, Саймон. Нейронные сети: полный курс, 2-е изд., испр. : Пер. с англ. — М. : 000 "И.Д. Вильяме 2006. — 1104 с. : ил. — Парал. тит. англ.

[2] Bishop C. M. Pattern Recognition and Machine Learning: Springer, 2006. - 749 с.

[3] Воронцов К. В. Лекции по искусственным нейронным сетям http://www.ccas.ru/voron/download/NeuralNets.pdf

[4] Nielsen M. Neural Networks and Deep Learning http://neuralnetworksanddeeplearning.com/

[5] Nielsen M. Neural Networks and Deep Learning http://neuralnetworksanddeeplearning.com/

[6] Denis Belomestny, Anastasia Kolodko, and John Schoenmakers Regression Methods for Stochastic Control Problems and Their Convergence Analysis http://epubs.siam.org/doi/abs/10.1137/090752651