САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-Механический факультет

Кафедра Системного Программирования

Корыстов Максим Андреевич

Исследование алгоритмов обучения искусственной нейронной сети для задач классификации

Курсовая работа

Научный руководитель:
Невоструев К. Н.

Санкт-Петербург
2014 г.

Содержание

Введение **3**
1.1 Мотивация . . . 3
1.2 Постановка задачи . . . 3
1.3 Доступные программные средства . . . 3

2 Нейронные сети **4**
2.1 Задача классификации . . . 4
2.2 Нейронная сеть . . . 4
2.3 Прямонаправленная нейронная сеть . . . 4
2.4 Штрафная функция . . . 5
2.5 Проблема переобучения и недообучения . . . 6
2.6 Автоматический подбор параметров . . . 6
2.7 Инициализация параметров . . . 7

3 Алгоритмы оптимизации **7**
3.1 Производные штрафной функции . . . 7
3.2 Общие сведения об алгоритмах . . . 8
3.3 Градиентный спуск с постоянным шагом . . . 8
3.4 RPROP . . . 8
3.5 QuickProp . . . 9
3.6 L-BFGS-B . . . 9

4 Сравнения **10**
4.1 Набор данных adult . . . 10
4.2 Набор данных titanic . . . 11
4.3 Набор данных mnist . . . 11
4.4 Набор данных poker . . . 12
4.5 Сравнение результатов обучения . . . 12

Заключение **13**

Введение

1.1 Мотивация

В последнее время в области разработки программных продуктов появилось много задач требующих применение методов машинного обучения. *Машинное обучение* – комплекс методов и алгоритмов для построения моделей, способных обучаться по набору данных.

Одна из задач машинного обучение – *задача классификации*. Например бинарная классификация: есть набор данных описывающий результаты анализов m пациентов и состояние больного - болен/не болен. Задача в том, чтобы по этим данным найти зависимость между анализами и состоянием здоровья человека, и предсказать состояние больного не из начального набора по его анализам.

Нейронная сеть одна из математических моделей применяемая для решение задачи классификации. Самый трудоемкий процесс – это обучение нейронной сети. Для этой цели существуют несколько алгоритмов. Правда ли что для разных наборов данных выгоднее использовать различные алгоритмы обучения нейронной сети? И если да, то какая между ними зависимость?

1.2 Постановка задачи

Цель работы состоит в обзоре существующих подходов и алгоритмов для решения задачи классификации с помощью нейронных сетей. А так же в проведении исследования зависимостей между набором входных данных и эффективностью различных алгоритмов обучения. В этой работе планировалось решить следующие задачи:

- Подготовить наборы данных для задачи классификации
- Установить зависимость эффективности обучения нейронной сети использующую разные алгоритмы обучения от набора входных данных
- Произвести сравнения результатов обучения на разных наборах данных

1.3 Доступные программные средства

Все вычисления производились с использованием языка программирования Python 2.7 и библиотек numpy, scipy.

2 Нейронные сети

2.1 Задача классификации

Пусть есть m объектов, каждый с n вещественными параметрами. Задача k-классификации заключатся в поиске с помощью этих данных функции

$$f : \mathbb{R}^n \to \{0, \ldots, K-1\}$$

которая любому объекту (не только из заданного набора) сопоставляет класс.

2.2 Нейронная сеть

Нейронная сеть – математическая модель для решения задачи классификации и предсказания. Далее мы будем ее рассматривать только как классификатор. По своей сути - это граф с n вершинами-входами и K вершинами-выходами. Здесь K - количество классов, в задачи классификации. Вершины в этом графе называются нейронами. Нейроны связаны взвешенными ребрами. Значение нейрона определяется весами ребер входящими в него и значением нейронов на противоположных концах этих ребер. По n параметрам объекта нейронная сеть выдает K чисел в отрезке $[0, 1]$, индекс максимального числа объявляется классом объекта. Отдельно рассматривается ситуация, когда $K = 2$, в этом случае нейронная сеть выдает одно число O_0 в отрезке $[0, 1]$, и класс объекта объявляется равным 0 если $O_0 < 0.5$ и 1 иначе.

2.3 Прямонаправленная нейронная сеть

Прямонаправленная нейронная сеть (feedforward neural network) – это нейронная сеть разбитая на слои, где все ребра направлены в одну сторону. У такой нейронной сети всегда есть входной слой и выходной слой и возможно несколько скрытых слоев. Рассмотрим пример прямонаправленной нейронной сети с одним слоем (Рис. 1). Здесь

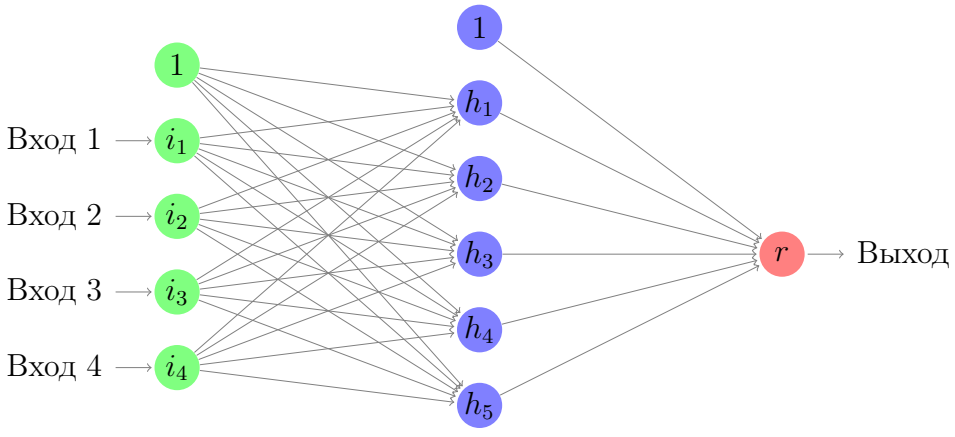


Рис. 1: Прямонаправленная нейронная сеть с одним скрытым слоем.

нейроны из входного слоя обозначаются i_s, нейроны из скрытого слоя h_s и r – нейрон выходного слоя. В каждом слое, кроме последнего добавляется фиктивный нейрон со значением 1. Веса ребер между слоем с n нейронами и с m нейронами представляется матрицей Θ размера $m \times (n+1)$, то есть θ_{ji} – вес ребра между i-ым и j-ым нейроном в разных слоях, причем θ_{j0} – веса ребер из фиктивного нейрона. Каждый слой представим как вектор v из значений нейронов, обозначим за $\tilde{v}$ вектор v расширенный единицей.

Обозначим соответствующие слоям вектора как i, h, r, а матрицы соответствующие ребрами между ними, как $\Theta^{i,h}$, $\Theta^{h,r}$. Тогда вычисление значения в слоях выполняется следующим образом:

$$h = \sigma(\Theta^{i,h}\,\tilde{i}),\ r = \sigma(\Theta^{h,r}\,\tilde{h})$$

Где σ действует по координатно и имеет вид $\sigma(x) = \frac{1}{1+e^{-x}}$. График функции $\sigma(x)$ представлен на Рис. 2. Последовательное вычисление значения нейронов во всех слоях назы-

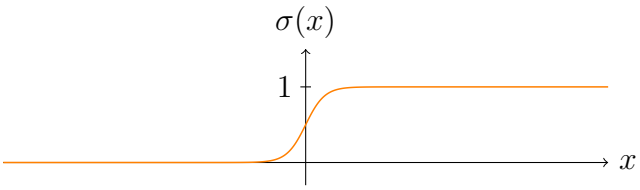


Рис. 2: График функции $\sigma(x)$

вают *прямым проходом*. Аналогичные действия предпринимаются для нейронной сети с большим количеством слоев, однако, согласно *Обобщенной аппроксимационной теореме*, утверждающей что, прямонаправленная нейронная сеть с одним скрытым слоем и с конечным числом нейронов позволяет вычислять значение произвольной непрерывной функции, с любой точностью. [1] [2]. Только нет гарантий, что можно легко найти соответствующие такой нейронной сети веса ребер и количество вершин в скрытом слое. Поиск матриц Θ соответствующих весам ребер между слоями называется *процессом обучения нейронной сети*. Далее в работе прямонаправленные нейронные сети будут называться просто нейронными сетями.

2.4 Штрафная функция

Рассмотрим выборку из m объектов (*тренировочная выборка*), про каждый из которых известен класс, к которому он принадлежит. Назовем Θ вектор всех весов ребер нейронной сети. Предположим, что мы нашли какую-то Θ, теперь надо проверить насколько точные предсказания осуществляет нейронная сеть с этими весами, для этого введем штрафную функцию $J_i(\Theta)$, которая для i-го объекта из выборки считает насколько совпадает предсказанный результат с реальным. Пусть i-ый объект принадлежит классу $c_i \in \{0, \ldots, K-1\}$, тогда введем бинарный вектор $y^{(i)}$, такой что его c_i-ая компонента равна единице, а остальные равны нулю. Так же введем функцию $h_\Theta(x)$ выдающую результат работы нейронной сети (вектор из K чисел в интервале $[0, 1]$) для n-мерного вектора (объекта) x. Функция $J_i(\Theta)$ выглядит так:

$$J_i(\Theta) = \sum_{k=0}^{K} y^{(i)} - \log(h_\Theta(x^{(i)})_k) - (1 - y^{(i)})log(1 - h_\Theta(x^{(i)})_k)$$

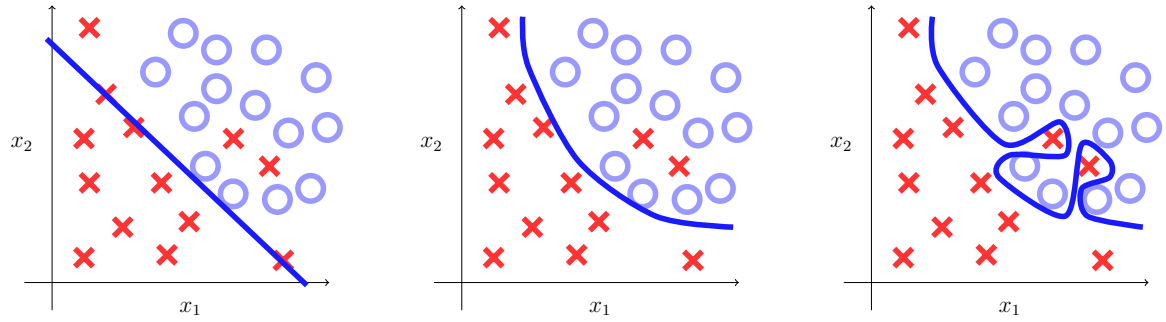
Видно, что для абсолютно верных предсказаний (когда $h_\Theta(x^{(i)}) = y^{(i)}$) значение этой функции будет равно нулю, и значение функции увеличивается при удалении предсказания от верного. Теперь рассмотрим штрафную функцию $J(\Theta)$, описывающую качество предсказания сразу для всех объектов из выборки:

$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m} J_i(\Theta)$$

Теперь чтобы обучить нейронную сеть достаточно найти *минимум функции* $J(\Theta)$.

2.5 Проблема переобучения и недообучения

Если с определенной Θ-ой нейронная сеть будет слишком точно предсказывать результат для тренировочной выборки, а в этой выборке есть шумы или отклонения, то это негативно повлияет на качество предсказаний для объектов не состоящих в тренировочной выборке, эта проблема называется *проблемой переобучения*. Противоположная ситуация случается, когда для тренировочной выборки нейронная сеть делает не точные предсказания и значение функции $J(\Theta)$ велико, такое положение дел называется *проблемой недообучения*. В качестве примера рассмотрим задачу бинарной классификации на плоскости (Рис. 3). На рисунке Рис. 3 (a) объекты разделяются классификатором



(a) Недообучение (b) Удов. обучение (c) Переобучение

Рис. 3

слишком грубо – недообучение. На рисунке Рис. 3 (c) объекты из тестовой выборки разделяются слишком точно, что может привести к ошибкам предсказаний для новых объектов.

В нейронных эти проблемы связанны с количеством нейронов в скрытом слое. Если их достаточно много случается переобучение, если мало – недообучение. Для решения этих проблем в нейронной сети функция $J(\Theta)$ модифицируется следующим образом:

$$J_{reg}(\Theta) = J(\Theta) + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} {\theta_{ji}^{(l)}}^2$$

Где s_i – количество нейронов в слое номер i, а $\theta_{ji}^{(l)}$ – вес ребра между i-ым нейроном l-го слоя и j-ым нейроном слоя номер $l+1$. $\lambda \geqslant 0$ называется параметром регуляризации: когда он слишком большой нейронная сеть называется *предубежденной* и происходит недообучение, когда же он не достаточно велик может произойти переобучение.

2.6 Автоматический подбор параметров

При создании выше описанной модели возникает вопрос как оптимально выбрать количество нейронов в скрытом слое n и значение параметра регуляризации λ. При различных n и λ нейронная сеть может получится склонная к переобучению или к недообучению. Для поиска параметров можно применить *алгоритм поиска в сетке (Grid search)*. Для параметров определим интервалы $[n_{min}, n_{max}]$ и $[\lambda_{min}, \lambda_{max}]$ в которых находятся все допустимые значения. Представим пару параметров (n, λ), как точку в прямоугольнике $P^{(0)} = [n_{min}, n_{max}] \times [\lambda_{min}, \lambda_{max}]$. Идея состоит в том что бы разбить этот прямоугольник на прямоугольники меньшего размера, в каждом из них выбрать центр (получить параметры n и λ) и при этих параметрах обучить нейронную сеть, потом

проверить на сколько точные предсказания данная нейронная сеть делает на тестовой выборке, сравнить точность полученную во всех центрах прямоугольников и взять лучший, после чего повторить алгоритм для него. Повторять до тех пора, пока не будет достигнута приемлемая точность параметров. Иллюстрация алгоритма на Рис. 4. λ мо-

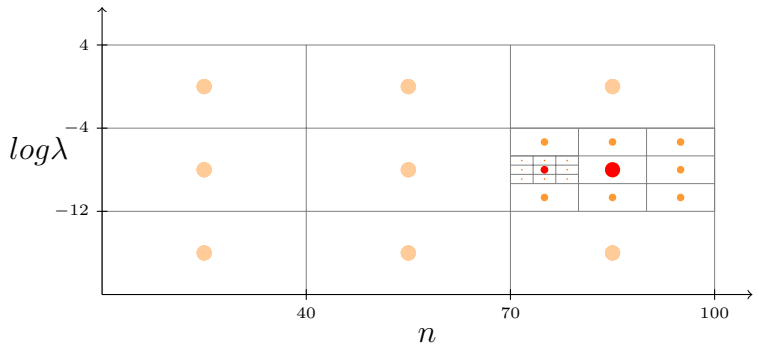


Рис. 4: Алгоритм поиска в сетке

жет достигать больших значений, поэтому в алгоритме ищется $\log \lambda$. Этот алгоритм не всегда находит оптимальные параметры, в некоторых случаях он может найти лишь локально оптимальные параметры. Чтобы этого избежать прямоугольник разбивают на большее количество частей.

2.7 Инициализация параметров

Для начала алгоритма обучения надо выбрать стартовые значения параметров в $J_{reg}(\Theta)$. Это делается по следующему правилу: если $\Theta^{(l)}$ имеет размеры $m \times (n+1)$, то все θ_{ij} выбираются как случайные величины из равномерного распределения в отрезке $\left[-\frac{3}{\sqrt{(n+m+1)}}, \frac{3}{\sqrt{(n+m+1)}}\right]$. Для многих задач инициализация начальных параметров таким способом дает хорошие результаты при обучении нейронной сети [3].

3 Алгоритмы оптимизации

3.1 Производные штрафной функции

Как было выяснено ранее для обучения нейронной сети надо минимизировать функцию $J_{reg}(\Theta)$. Далее в этом разделе описываются алгоритмы поиска минимума функции используемые в сравнении. Все они используют частные производные функции $J_{reg}(\Theta)$, можно вычислить эти производные с помощью *алгоритма обратного распространения ошибки (backpropagation)* [4].

Начнем с вычисления производных функции $J(\Theta)$. Рассмотрим объект $x^{(p)}$, где $1 \leqslant p \leqslant m$ – номер объекта. Введем $a^{(l)}$ – вектор значений нейронов на слое номер l вместе с фиктивным нейроном равным единице (входной слой имеет номер 1, выходной L). Также обозначим за $z^{(l)} = \sigma^{-1}(a^{(l)})$ без фиктивного нейрона. И пусть $y^{(i)}$ – бинарный вектор описывающий класс этого объекта (одна единица в компоненте с индексом k). Тогда назовем следующую величину ошибкой нейронов на слое L:

$$\delta^{(L)} = a^{(L)} - y^{(i)}$$

А на слое с номерами $1 < l < L$:

$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} * \sigma'(z^{(l)})$$

Утверждается, что если сложить величины $a_j^{(l)}\delta_i^{(l+1)}$ для всех m объектов из выборки то в результате получится производная $J(\Theta)$ [4], иначе говоря:

$$\frac{\partial J(\Theta)}{\partial \theta_{ij}^{(l)}} = \frac{1}{m}\sum_{p=1}^{m} a_j^{(l)}\delta_i^{(l+1)}$$

Тогда производная $J_{reg}(\Theta)$ имеет вид:

$$\frac{\partial J_{reg}(\Theta)}{\partial \theta_{ij}^{(l)}} = \begin{cases} \frac{\partial J(\Theta)}{\partial \theta_{ij}^{(l)}} + \lambda\theta_{ij}^{(l)}, & j \neq 0 \\ \frac{\partial J(\Theta)}{\partial \theta_{ij}^{(l)}}, & j = 0 \end{cases}$$

3.2 Общие сведения об алгоритмах

Все алгоритмы которые буду рассмотрены ниже являются *релаксационными*, иначе говоря для поиска минимума функции $f(x)$ алгоритм строит последовательность $x^{(0)}, x^{(1)}, \ldots, x^{(k)}, \ldots$ удовлетворяющую условию:

$$f(x^{(0)}) \geqslant f(x^{(1)}) \geqslant \ldots \geqslant f(x^{(k)}) \geqslant \ldots$$

3.3 Градиентный спуск с постоянным шагом

Идея алгоритма заключается в том, чтобы осуществлять шаг в направлении наискорейшего убывания функции, то есть в направлении вектора $-\nabla f(x)$ [5]. Выбирая в точке $x^{(k)}$ в качестве направления спуска антиградиент функции $(-\nabla f(x^{(k)}))$ приходим к итерационному процессу вида:

$$x^{(k+1)} = x^{(k)} - \alpha_k \nabla f(x^{(k)})$$

Существует много различных способов выбора параметра α_k – длинны шага, но наиболее распространенные это метод с постоянным шагом $\alpha_k = \alpha$ и с дроблением шага.

Рассмотрим метод с постоянным шагом. При достаточно больших α алгоритм может расходится. Если же α достаточно маленькие то алгоритму потребуется много шагов. Часто α выбирают экспериментальным путем пробуя значения $10^{-3}, 10^{-2}, 10^{-1}, 1, 10$.

3.4 RPROP

Схожим с градиентным спуском является алгоритм *RPROP (Resilient propagation)*. Однако, этот алгоритм использует только знаки производных. Пусть требуется найти минимум функции $f : \mathbb{R}^n \to \mathbb{R}$. Величина шага Δ определятся следующим образом:

$$\Delta_i^{(k)} = \begin{cases} \eta^+\Delta_i^{(k)}, & \frac{\partial f(x^{(k)})}{\partial x_i}\frac{\partial f(x^{(k-1)})}{\partial x_i} > 0 \\ \eta^-\Delta_i^{(k)}, & \frac{\partial f(x^{(k)})}{\partial x_i}\frac{\partial f(x^{(k-1)})}{\partial x_i} < 0 \end{cases}$$

Где величины η^-, η^+ являются параметрами алгоритма и удовлетворяют условиям $0 < \eta^- < 1 < \eta^+$. Если на текущем шаге алгоритма производная по соответствующей переменной x_i поменяла свой знак, то это свидетельствует о слишком большом последнем изменении и о пропущенном локальном минимуме, следовательно надо уменьшить Δ_i на η^-. Если же знак частной производной не изменился то нужно увеличить размер

шага Δ_i на η^+. Для не допущения слишком больших и слишком маленьких размеров шага величину Δ_i ограничивают сверху Δ_{max}, а снизу Δ_{min}. В итоге чтобы получить $x_i^{(k+1)}$ по $x_i^{(k)}$ используется формула:

$$x_i^{(k+1)} = x_i^{(k)} - sign\left(\frac{\partial f(x^{(k)})}{\partial x_i}\right)\Delta_i$$

Во многих задачах алгоритм показывает себя наилучшим образом при $\eta^- = 0.5$, $\eta^+ = 1.2$, $\Delta_{max} = 50$, $\Delta_{min} = 10^{-6}$, и $\Delta_i^{(0)} = 0.1$ [**?**].

3.5 QuickProp

Этот алгоритм также аналогичен градиентному спуску. Приращение по аргументу x_i на шаге k вычисляется по формуле:

$$\Delta x_i^{(k)} = \begin{cases} \Delta x_i^{(k-1)} G_0, & \Delta x_i^{(k-1)} \neq 0 \\ \eta \frac{\partial f(x^{(k)})}{\partial x_i}, & \Delta x_i^{(k-1)} = 0 \end{cases}$$

где

$$G_0 = \max\left(\min\left(\frac{\frac{\partial f(x^{(k)})}{\partial x_i}}{\frac{\partial f(x^{(k-1)})}{\partial x_i} - \frac{\partial f(x^{(k)})}{\partial x_i}}, \mu\right), -\mu\right)$$

Тогда чтобы получить значение $x_i^{(k+1)}$ вычисляют:

$$x_i^{(k+1)} = x_i^{(k)} - \Delta x_i^{(k)}$$

Значение коэффициентов η и μ устанавливают экспериментально. Часто хорошим выбором бывает $\mu = 1.75$, $\eta = 0.1$ [3].

3.6 L-BFGS-B

Алгоритм Бройдена — Флетчера — Гольдфарба — Шанно с ограниченным использованием памяти (Limited-memory BFGS). Основан на алгоритме BFGS, который более требователен к памяти, порядка $O(n^2)$, где n – количество параметров функции f. Идея BFGS заключается в разложении f в полином второй степени:

$$f(p + x_k) \approx f(x_k) + \nabla f^T(x_k)p + \frac{1}{2}p^T B_k p$$

в текущей точке x_k, где B_k симметричная положительно определенная матрица (приближение гессиана – матрицы вторых производных) обновляемая на каждой итерации. После чего находится минимум данной квадратичной задачи:

$$p_k = -B_k^{-1}\nabla f(x_k)$$

И находится следующий шаг:

$$x_{k+1} = x_k + \alpha_k p_k$$

Где α_k удовлетворяет условиям Вольфе:

$$\begin{aligned} f(x_k + \alpha_k p_k) &\leqslant f(x_k) + c_1 \alpha p_k^T \nabla f(x_k) \quad c_1 \in (0,1) \\ p_k^T \nabla f(x_k + \alpha_k p_k) &\geqslant c_2 p_k^T \nabla f(x_k) \qquad\qquad c_2 \in (c_1, 1) \end{aligned}$$

Вместо того чтобы заново считать B_k на каждой итерации, к B_k добавляется матрица изменений на каждом шаге [6]. В отличии от BFGS алгоритм L-BFGS-B менее требовательный к памяти, потому что хранит матрицу B_k неявно. Размер потребляемой памяти часто бывает критичен, так как функция $J_{reg}(\Theta)$ может иметь тысячи параметров.

4 Сравнения

Все описанные выше алгоритмы были реализованы на языке Python 2.7 с использованием библиотек numpy, scipy. Для каждого набора данных были определенны наилучшие параметры λ (параметр регуляризации) и n (количество нейронов в скрытом слое) с помощью *алгоритма поиска в сетке*, после чего запущен процесс обучения с использованием четырех алгоритмов: Градиентный спуск, RPROP, QuickProp, L-BFGS-B. Алгоритмы сравниваются по значению штрафной функции $J_{reg}(\Theta)$ в зависимости от количества шагов. На каждом наборе данных стартовые параметры инициализировались 10 раз и запускался процесс обучения четырьмя алгоритмами. В графиках ниже показано медианное значение функции $J_{reg}(\Theta)$ на каждом из шагов.

4.1 Набор данных adult

Состоит из 32 561 объектов тренировочной выборки и из 16 281 объектов тестовой выборки. Каждый объект описывается 123 атрибутами. Бинарная задача классификации [7].

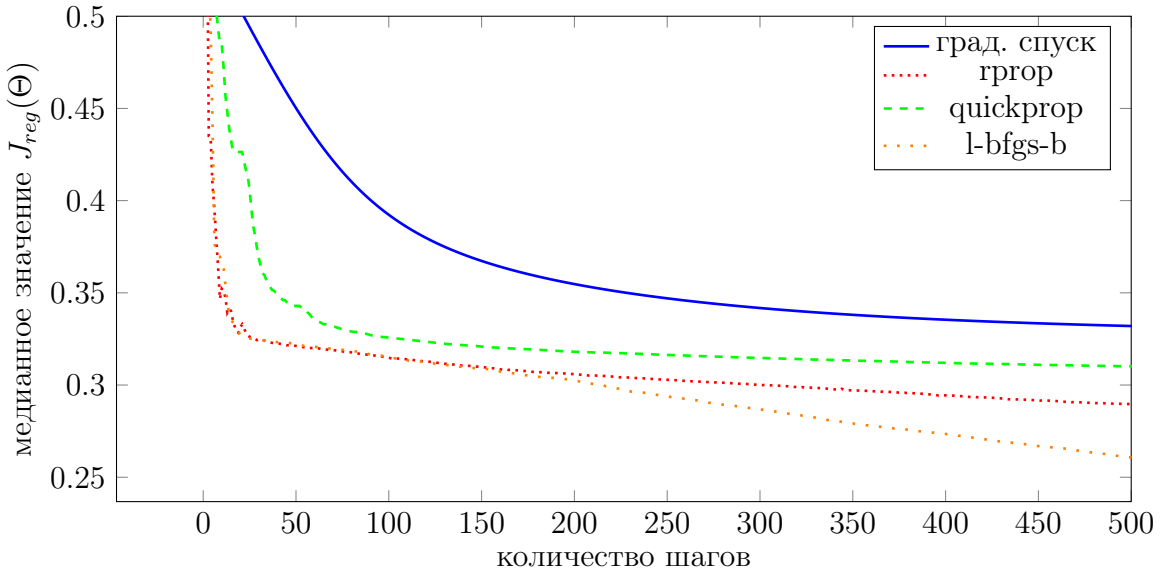


Рис. 5: Набор данных: adult

4.2 Набор данных titanic

Состоит из 1526 объектов тренировочной выборки и из 673 объектов тестовой выборки. Каждый объект описывается 6 атрибутами. Бинарная задача классификации [8].

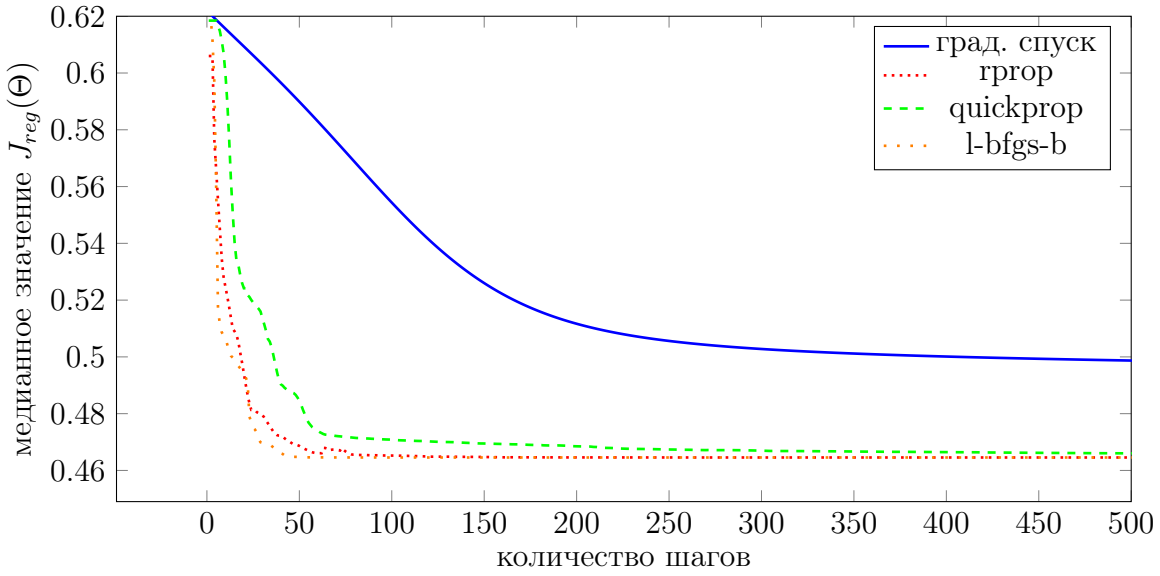


Рис. 6: Набор данных: titanic

4.3 Набор данных mnist

Состоит из 60 000 объектов тренировочной выборки и из 10 000 объектов тестовой выборки. Каждый объект описывается 784 атрибутами. Задача классификации с 10 классами [9].

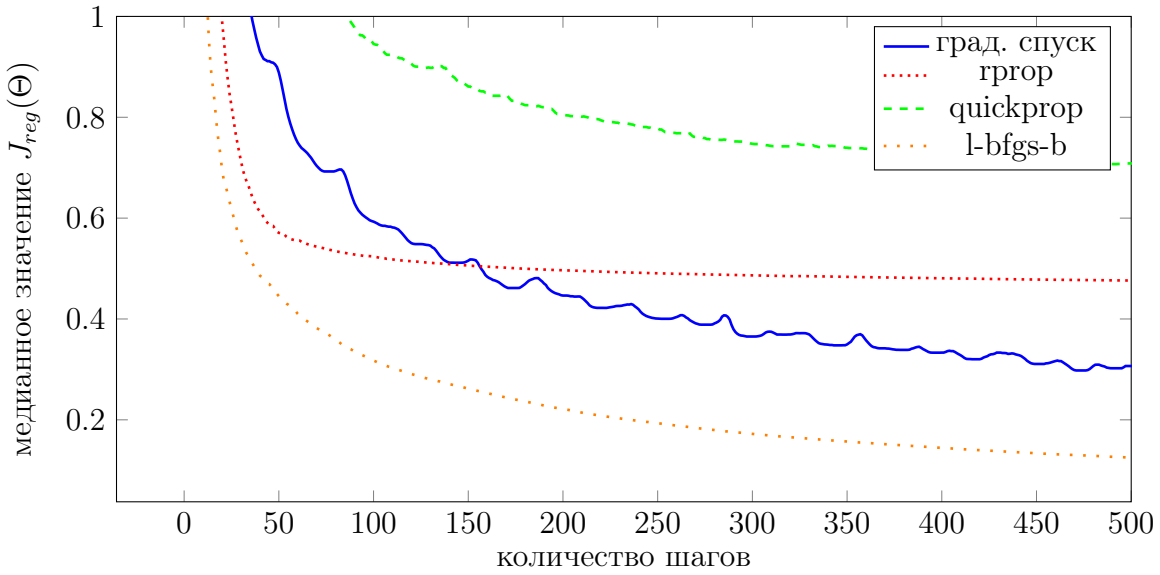


Рис. 7: Набор данных: mnist

4.4 Набор данных poker

Состоит из 25 010 объектов тренировочной выборки и из 1 000 000 объектов тестовой выборки. Каждый объект описывается 17 атрибутами. Задача классификации с 10 классами [10].

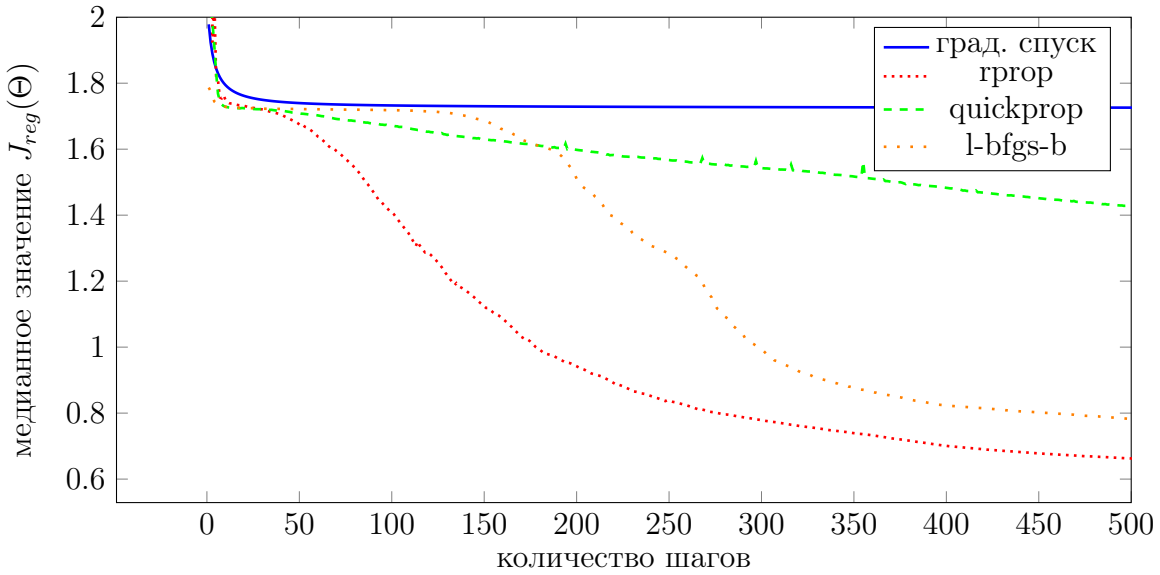


Рис. 8: Набор данных: poker

4.5 Сравнение результатов обучения

	Наборы данных							
	adult		**titanic**		**mnist**		**poker**	
Алгоритм	%	F1	%	F1	%	F1	%	F1
Градиентный спуск	82.85	63.89	77.22	50.91	94.07	93.99	50.04	6.75
RPROP	83.44	64.30	77.31	52.11	90.64	90.51	92.43	19.91
QuickProp	83.06	64.01	77.31	52.11	88.85	88.67	69.05	14.14
L-BFGS-B	84.05	64.58	77.31	52.11	95.97	95.93	91.02	18.88

В таблице приведены медианные значения точности в процентах и $F1 * 100$ для 10 запусков четырех алгоритмов на каждом наборе данных. Точность – сколько процентов предсказанных классов совпало с настоящими значениями классов. $F1$ – мера качества обучения, число в отрезке $[0, 1]$, больше – лучше [11]. Совпадение отношений точности обучения алгоритмов с соответствующими графиками штрафной функции говорит о устойчивости модели к переобучению, иначе говоря *алгоритм поиска в сетке* нашел подходящие параметры λ, n.

Заключение

Как видно из сравнений, лидирующие результаты показывают алгоритмы RPROP, L-BFGS-B. Однако, на небольшом по количеству объектов наборе данных (titanic) схожие результаты с RPROP, L-BFGS-B показывает QuickProp и к ним приближается градиентный спуск, то есть на небольших наборах данных выгодно использовать простые алгоритмы: QuickProp, градиентный спуск, так как они требуют меньшее количество операций для получения следующего шага. Когда важна скорость обучения (непозволительно делать много шагов) хорошим выбором будет RPROP, схема алгоритма проста и не требует больших ресурсов для вычисления следующего шага по сравнению с L-BFGS-B. В случае когда скорость обучения не важна, при большом количестве шагов, чаще всего наименьшее значение штрафной функции достигается алгоритмом L-BFGS-B.

Сравнения показали, что нельзя выбрать один алгоритм и получить лучшие результаты на всех типах задач. Решением будет проанализировать графики ошибок для всех алгоритмов и выбрать лучший.

Список литературы

[1] Csanád Csáji Balázs. Approximation with artificial neural networks. Master's thesis, Faculty of Sciences; Eötvös Loránd University, Hungar, 2001.

[2] А. Н. Горбань. Обобщенная аппроксимационная теорема и вычислительные возможности нейронных сетей. *Сибирский журнал вычислительной математики*, 1998.

[3] K.-L Du and M.N.S. Swamy. *Neural Networks in a Softcomputing Framework*. Springer-Verlag London Limited, 2006.

[4] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature 323, 533-536*, 1986.

[5] Н. И. Глебов, Ю. А. Кочетов, and А. В. Плясунов. *Методы оптимизации*. Новосибирский государственный университет, Новосибирск, 2000.

[6] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer, New York, 2006.

[7] Набор данных adult. `http://archive.ics.uci.edu/ml/datasets/Adult` (Дата обращения 20 мая 2014).

[8] Набор данных titanic. `http://www.kaggle.com/c/titanic-gettingStarted` (Дата обращения 20 мая 2014).

[9] Набор данных mnist. `http://yann.lecun.com/exdb/mnist/` (Дата обращения 20 мая 2014).

[10] Набор данных poker. `http://archive.ics.uci.edu/ml/datasets/Poker+Hand` (Дата обращения 20 мая 2014).

[11] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.