Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра системного программирования

Исследование возможности применения бинарных диаграмм решений для распознавания текста

Курсовая работа студента 445 группы
Зубаревича Дмитрия Александровича

Научный руководитель …… Д.Ю. Бугайченко

Санкт-Петербург
2013

Оглавление

Введение

В современной информатике существует широкий класс задач, для решения которых необходимо оперировать большими множествами. К таким задачам можно отнести кластеризацию данных, обработку и распознавание образов и прочее. Обычные описывающие подходы к представлению данных в таких задачах ведут к непомерному расходу памяти, а конструирующиеподходы ведут к сильному усложнению алгоритма обработки данных.

Альтернативой к указанным выше подходам является представление с помощью бинарных диаграмм решений [1](далее БДР). В этом случае описываемые объекты (множества, функции, матрицы) кодируются графовой структурой в соответствии с определёнными правилами. Зачастую такое представление позволяет добиться того, чтобы размер графа рос гораздо медленнее, чем размер соответствующего перечисляющего описания. Кроме того операции над объектами, представленными БДР, выполняются за полиномиальное время от размера соответствующих графов.

Рассмотрим подробнее задачу распознавания графических образов. Её решение предполагает необходимость отнести переданное на вход изображение к одному из предопределенных классов на основании предварительно размеченного тренировочного множества.При решении этой задачи обычно выделяют три подхода:

- перебора вида объекта под различными углами, масштабами, смещениями и прочее;
- исследование топологических свойств выделенного контура объекта (связность, наличие углов и т. д.);
- использование искусственных нейронных сетей[3] (далее ИНС). В частности для распознавания оптических образов хорошо зарекомендовали себя свёрточные нейронные сети.

Идея первого подхода достаточно проста, при этом очевидно, что он достаточно трудоёмок. Второй подход, по сути, является конструирующим, а значит сложным для разработки и обобщения. Третий подход требует наличия большого обучающего множества при построении ИНС, хотя в результате получается весьма прозрачная и легкая структура для распознавания.

Отметим, что последний подход имеет неприятный недостаток: обучение ИНС по сути есть подгонка большого числа параметров (коэффициентов связей между нейронами) методом проб и ошибок, поэтому в конечном счёте знания ИНС хранятся в неявном виде. Это приводит к тому, что ИНС не может, обосновать принятие решения, что в свою очередь делает невозможным определение того, правильно ли ИНС выделила характеристические признаки образов при обучении или ошибочно.Желание избавиться от этого недостатка наталкивает на мысль о хранении тестового множества внутри ИНС. При таком подходе нейрон долженхранить множество правильных шаблонов, т.е. образов или их частей, и предоставлять метрику, позволяющую определять, насколько близко находится тестовый шаблон к обучающему множеству. Для этой цели как раз уместно применить БДР: множества для хранения шаблонов и функции для построения метрик.

Подход для такого построения нейрона был предложен в работе [4], и представлен библиотекой BddNeuron, в основе которой лежит библиотека для работы с БДР – BddFunctions.Данная работа является развитием этого подхода.

1. Постановка задачи

Задачей данной курсовой работы является:

- Реализация библиотеки для построения и тестирования ИНС, использующих БДР
- Создание схем ИНС, использующих БДР;
- Их анализ с точки зрения временных затрат на обучение и качества распознавания.

2. Краткие сведения о бинарных диаграммах решений

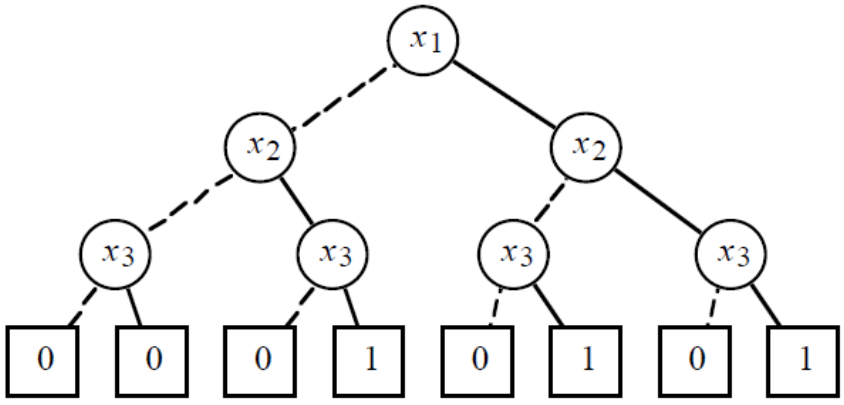
Формально, упорядоченная БДР функции вида $f : \{0,1\}^n \rightarrow S$есть ориентированный корневой ациклический граф с множеством вершин $V = T \cup N,\ T \cap N = \emptyset$. Вершины множества N называются *нетерминилами*, идля каждой такой вершины $v \in N$определены *значение порядка* $index(v) \in \{1, \dots, n\}$и ровно две дочерние вершины $low(v),\ high(v) \in V$. Индексы нетерминальных вершин iсоответствуют аргументам определяемой функции x_i. Вершины множестваTназываются *терминалами*и не имеют дочерних вершин. Для каждой терминальной вершины $v \in T$определено значение $value(v) \in S$. Кроме того выполнено *условие порядка:* для любого нетерминала $v \in N$и любой его дочерней вершины $v`$ выполнено либо $v` \in T$, либо $index(v) < index(v`)$. Теперь каждой вершине $v \in V$ можно сопоставить функцию $f_v : \{0,1\}^n \rightarrow S$:

$$f_v(x_1, \dots, x_n) = \begin{cases} value(v), & v \in T \\ f_{high(v)}(x_1, \dots, x_n), & x_{index\,(v)} = 1,\ v \in N \\ f_{low\,(v)}(x_1, \dots, x_n), & x_{index\,(v)} = 0,\ v \in N \end{cases}$$

Существует несколько разновидностей решающих диаграмм. В задачах связанных с использованием булевых функций вида $f : \{0,1\}^n \rightarrow \{0,1\}$и конечные множества широкое распространение получили бинарные решающие диаграммы (BDD) [10]. В задачах, связанных с использованием функций вида $f : \{0,1\}^n \rightarrow S$, где S есть некоторое конечное непустое множество, и нечётких множеств, часто применяются многотерминальные бинарные решающие диаграммы (MTBDD) и различные их модификации.Альтернативой MTBDD являются многокорневые бинарные решающие диаграммы (MRBDD). Они работают с конечнозначными функциями, как с векторами из булевых функций. Их основное преимущество перед MTBDD – меньший размер, достигаемый, за счёт более эффективного повторного использования фрагментов одинаковой структуры.

Рассмотрим пример. На рисунке слева изображена таблица истинности функции f, а справа её представление в виде бинарного дерева решений:

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



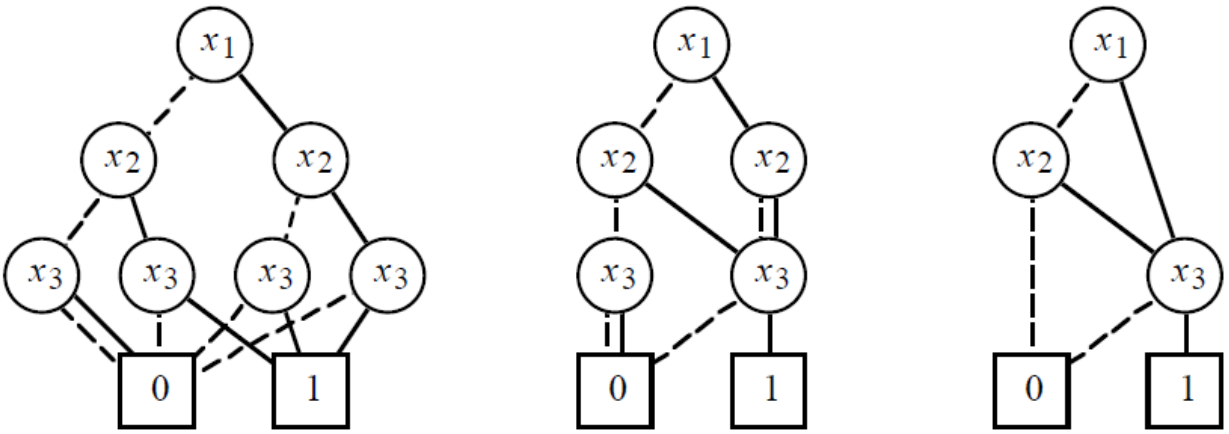
Табличное задание функции (слева) и бинарное дерево решений (справа).

Далее производится редукция графа в соответствии с тремя правилами:

- Слияние дубликатов терминалов с соответствующим перенаправлением дуг;

- Слияние дубликатов нетерминалов, т.е. если нетерминальные вершины $u, v \in N$такие, что $index(u) = index(v)$, $low(u) = low(v)$, $high(u) = high(v)$, то вершины $uиv$ совмещаются с соответствующим перенаправлением дуг;
- Удаление нетерминалов с одной дочерней вершиной с перенаправлением в неё входящих дуг.

Возвращаясь к нашему примеру и последовательно применяя к нему перечисленные правила редукции, получим следующий результат:



БДР после применения первого (слева), второго (в центре) и третьего (справа) правил редукции.

На картинке справа изображён конечный вариант диаграммы для функции f. Как видим такое представление гораздо более компактно, чем таблица истинности.

Для представления множеств с помощи БДР, обычно используют характеристические функции. Это позволяет легко выразить операции над множествами через операции над характеристическими функциями:

- $\chi_{S \cup T} = \chi_S + \chi_T$
- $\chi_{S \cap T} = \chi_S \cdot \chi_T$
- $\chi_{S \setminus T} = \chi_S \cdot \overline{\chi_T}$

Заметим, что такое представление позволяет строить множество без явного перечисления всех его элементов.

Чтобы получить представление конечнозначных матриц $A \in S^{n \times n}$с помощью БДР, заметим, что матрицу можно рассматривать как функцию$f_A : \{1, \dots, n\}^2 \to S$, определяемую следующим образом: $f_A(i, j) = A_{ij}$. Тогда сложение матриц реализуется тривиальным образом через сложение соответствующих функций, а вот алгоритм вычисления произведения матриц выглядит несколько сложнее и подробно описывается в статье [1]. Заметим, что матрица, в широком смысле, при таком подходе, является расширением функции, основным отличием которого является то, что множество аргументов разбито на два непересекающихся подмножества: строки и столбцы.

3. Алгоритм определения расстояния от точки до множества.

3.1 Определения

Предположим, у нас есть некоторое множество $S = \{s_1, \dots, s_{|S|}\}$ и введена метрика $\rho : T \times T \to R$, где $T \supset S$, для определения расстояния между двумя точками. Нам необходимо научиться определять расстояние от точки, лежащей в множестве T, до множества S.

Дадим формальное определение. Расстоянием от точки $y \in T$ до множества S, называется функционал $\delta_{\rho,S} : T \to R$такой, что:

$$\delta_{\rho,S}(y) = \begin{cases} 0, & \text{если } y \in S, \\ \dfrac{\sigma_{\rho,S}(y)}{|S|}, & \text{если } y \notin S, \end{cases}$$

где $\sigma_{\rho,S}(y) = \sum_{i=1}^{|S|} \rho(s_i, y)$.

По сути это означает, что расстоянием от точки до множества, мы будем считать ноль, если точка принадлежит множеству, и среднее арифметическое расстояний от точки до всех точек множества, в противном случае.

Заметим, что определение расстояния от точки до множества, в зависимости от необходимости применения в той или иной задаче, может быть дано по-разному. Расстояние $\delta_{\rho,S}$ подходит для класса задач, в которых важна статистика расстояний от точки до точек множества.

3.2 Статический подход

Заметим, что с множеством S, можно работать, оперируя его характеристической функцией $\chi_S : T \to \{0, 1\} \subset R$, выдающей единицу для точки из множества и ноль в противном случае. Тогда функцию $\sigma_{\rho,S}$ можно расписать так:

$$\sigma_{\rho,S}(y) = \sum_{x \in T} \chi_S(x) * \rho(x, y).$$

Теперь обратим внимание на то, что функции χ_S и ρ задают некоторые матрицы. В частности, исходя из областей определения функций,делаем вывод, что χ_S задаёт вектор-строку$A \in \{0, 1\}^{1\times|T|} \subset R^{1\times|T|}$, если аргумент xиндексирует столбцы, а ρ – матрицу $B \in R^{|T|\times|T|}$, если аргументы x и y индексируют строки и столбцы соответственно. При этом элементы матриц выглядят следующим образом: $A_{1x} = \chi_S(x)$, $B_{xy} = \rho(x, y)$.Заметим, что корректно определена операция умножения вектораA на матрицу B, так как количество строк матрицы B равно количеству столбцов вектораA. Проводя умножение, получим $(\chi_S \times \rho) = A * B = C \in R^{1\times|T|}$ – матрица, C_{1y} элемент которойбудет выглядеть так:

$$C_{1y} = \sum_{x \in T} A_{1x} * B_{xy} = \sum_{x \in T} \chi_S(x) * \rho(x, y) = \sigma_{\rho,S}(y).$$

Таким образом, получили что функция $\sigma_{\rho,S}$ задаётся матрицей$C = (\chi_S \times \rho)(y)$.

Заметим, что описанный подход легко обобщается на многомерный случай. Проводя аналогичные рассуждения можно построить функцию:

$$\delta_{\rho,S}(y_1, \dots, y_n) = \begin{cases} 0, & \text{если } (y_1, \dots, y_n) \in S, \\ \dfrac{\sigma_{\rho,S}(y_1, \dots, y_n)}{|S|}, & \text{если } (y_1, \dots, y_n) \notin S, \end{cases}$$

где $\sigma_{\rho,S}(y_1, \dots, y_n) = (\chi_S \times \rho)(y),\ S \subset T^n$.

Чтобы построить БДР для $\delta_{\rho,S}$, необходимо задать бинарную кодировку для множеств T и R. Это даст возможность построить БДР для функций χ_S и ρ, что в свою очередь, позволит, применяя операцию матричного умножения, построить БДР и для функции $\delta_{\rho,S}$.

3.3 Динамический подход

Статический подход вполне очевиден, однако имеет узкое место с точки зрения производительности. Дело в том, что матричное умножение– операция очень дорогая, что ставит под сомнение жизнеспособность, описанного выше алгоритма. Однако, не меняя сути алгоритма, от матричного умножения легко избавиться. Достаточно заметить, что мы владеем процессом построения множества S, то есть мы знаем, какую точку мы добавляем, а какую удаляем. Откуда очевидным образом вытекает, что функцию $\sigma_{\rho,S}$ мы можем строить динамически, изменяя её в момент изменения S.

Предположим, у нас есть промежуточное множество S', которому соответствует функция $\sigma_{\rho,S'}$. Пусть мы хотим изменить S', добавив в него точку $x \in T$. Тогда получим: $S = S' \cup \{y\}$, - новое множество, $\sigma_{\rho,S}(y) = \sigma_{\rho,S'} + \rho(x, y)$ – соответствующая ему функция. Для удаления точки из множества, рассуждения аналогичны.

Заметим, что при таком подходе, от построения S, можно было бы отказаться вообще, однако на практике время построения этого множества ощутимо меньше, чем построение функции нахождения расстояния до него и, кроме того, его можно использовать для быстрого определения точного присутствия точки во множестве, а также для контроля уникальности точек.

Описанный подход на практике работает за гораздо более обозримое время, нежели статический, благодаря своей простоте.

3.4 Метрики для нахождения расстояния от точки до множества

В зависимости от задачи, в которой необходимо находить расстояние от точки до множества, следует выбирать наиболее подходящую функцию ρ. При этом выборе важно опираться также на следующие критерии:

- Функция ρ должна по возможности выражаться через наиболее простые операции. Наиболее простой операцией является сложение. При

необходимости также можно использовать умножение, композицию функцийи прочее;

- Операции, через которые выражается функция ρ, должны быть наименее трудоемкими. Сложение – наименее трудоемкая операция.Возведение в квадрат, умножение, композиция функций и т.д. – гораздо более трудоемкие операции.

Рассмотрим некоторые метрики, для которых несложно построить БДР, оперируя API библиотеки *BDDFunctions*. Рассмотрим точки$x = (x_1, \dots, x_n)$,$y = (y_1, \dots, y_n) \in T^n$, и метрику $\rho : T^n \times T^n \to R$.

Одной из простейших метрик, удовлетворяющей указанным выше критериям, является L_1-метрика ρ_{L_1}:

$$\rho_{L_1}(x, y) = \sum_{i=1}^{n} |x_i - y_i| = \sum_{i=1}^{n} f(x_i, y_i), \text{ где } f(x_i, y_i) = \begin{cases} x_i - y_i, x_i > y_i, \\ y_i - x_i, \ x_i < y_i. \end{cases}$$

При построении ρ_{L_1} применяется всего две операции: сравнение и вычитание. Данная метрика является весьма универсальной и может иметь достаточно широкое применение при хорошем подборе бинарной кодировки элементов множества T.

Другой несложной метрикой является расстояние Хэммингаρ_H. Чтобы определить его, введём функцию $bin: T \to \{0,1\}^m$, которая переводит точку из множества Тв её бинарную кодировку, и функцию $get: \{0, \dots, m-1\} \times \{0,1\}^m \to \{0,1\}$, которая возвращает значение указанного бита в данной кодировке.Положим $\alpha(j, i, x) = get(j, bin(x_i))$ – значение j-ого бита -ой компоненты вектора x. Тогда ρ_H определяется следующим образом:

$$\rho_H(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{m} |\alpha(j, i, x) - \alpha(j, i, y)| = \sum_{i=1}^{n} \sum_{j=1}^{m} \beta(j, i, x, y),$$

где$\beta(j, i, x, y) = \begin{cases} 0, \alpha(j, i, x) = \alpha(j, i, y) \\ 1, \alpha(j, i, x) \neq \alpha(j, i, y) \end{cases} = XOR(\alpha(j, i, x), \ \alpha(j, i, y))$.

При построении этой метрики применяются операции: сравнение, взятие бита с указанным номером в данной кодировке и суммирование по количеству бит в кодировке. Несмотря на количество операций, все они не очень требовательны к ресурсам, что свидетельствует о том, что ρ_H вполне можно применять для построения $\delta_{\rho,S}$.

Сама метрика ρ_H применяется, например, при сравнении хэшей, полученных с помощью локально-чувствительного хэширования. Последнее широко применяется в задачах распознавания образов[2],для снижения размерности множества, представляющего образы, что является весьма важным аспектом при применении БДР, так как БДР является очень требовательными к затратам по памяти.В дальнейшем под метрикой ρбудем подразумевать метрику ρ_H.

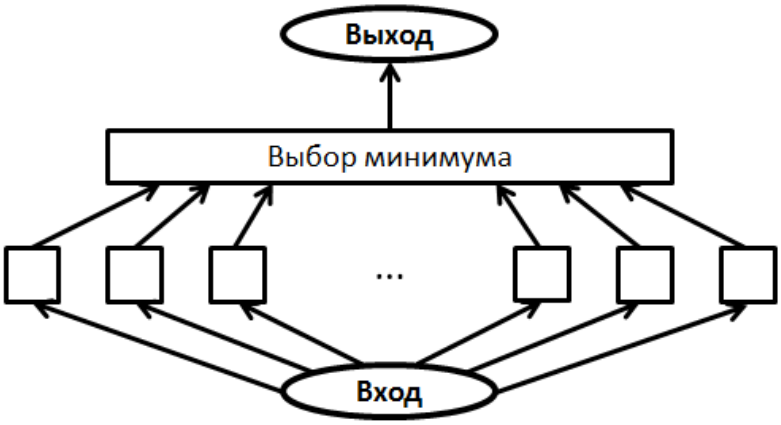
4. Структуры и алгоритмы работы ИНС на основе БДР

Вернёмся к вопросу распознавания графических образов, описанному в ведении. Там была сформулирована идея использования БДР для создания искусственного нейрона с явным представлением данных. Такой нейронформально является парой $(S, \delta_{\rho,S})$, где S – множество, хранящее шаблоны хороших точек, а$\delta_{\rho,S}$ – метрика, определяющая расстояние от точки до S.

Мы хотим построить ИНС с такими нейронами для распознавания графических образов – рукописных символов. Очевидно, что тестовый шаблон изображения одного символа должен быть более похожим на шаблоны нейрона, распознающего данный символ, чем на шаблоны нейрона, распознающего другой символ, в смысле расстояния ρ.Это оправдывает возможность использования описанной, выше функции $\delta_{\rho,S}$, в качестве метрики для определения расстояния от тестового шаблона, до множества шаблонов, содержащихся в нейроне.

4.1 Простейшая схема

Самая простая схема которая приходит в голову, – завести по одному нейрону на каждый символ. Тогда процесс обучения одного нейрона будет заключаться в том, чтобы добавить во множество шаблонов все хорошие точки и построить соответствующую функцию вычисления расстояния. Для распознавания необходимо взять интересующую точку и найти расстояние до каждого из нейронов. Символ, соответствующий нейрону, расстояние до которого получилось наименьшим, будем считать результатом распознавания. Ниже изображена схема такой ИНС.



Простейшая схема нейронной сети на основе
бинарных диаграмм решений.

4.2 Нейрон, разрешающий конфликты

Любая схема распознавания может ошибаться. Причем процент “перепутывания” пары символов может быть достаточно высок. Это приводит к необходимости создания способа разрешения конфликтов между парой символов.

Пусть у нас есть пара символов, которые часто конфликтуют, то есть вместо первого символа ИНС выдает второй, а вместо второго – первый. Такое может происходить потому, что многие бинарные кодировки, находящиеся в обучающем множестве первого символа, имеют похожие, с точки зрения расстояния Хэмминга, бинарные кодировки в обучающем множестве второго символа. Иными словами некоторые бинарные кодировки из разных обучающих множеств имеют много похожих битов.

Первое что напрашивается, – исключать из рассмотрения те биты, которые часто совпадают у кодировок из разных множеств. Для этого можно ввести понятие взвешенного расстояния Хэмминга. В обозначениях раздела 3.4 получим:

$$\rho_w(x,y,w) = \sum_{i=1}^{n}\sum_{j=1}^{m}|w(i,j) * (\alpha(j,i,x) - \alpha(j,i,y))| = \sum_{i=1}^{n}\sum_{j=1}^{m}\beta_w(j,i,x,y),$$

$$\beta_w(j,i,x,y) = \begin{cases} 0, & \alpha(j,i,x) = \alpha(j,i,y) \\ w(j,i), & \alpha(j,i,x) \neq \alpha(j,i,y) \end{cases} = XOR\big(\alpha(j,i,x),\ \alpha(j,i,y)\big) * w(j,i),$$

где w – вес j-ого бита -ой компоненты вектора x.

Возможность исключать из рассмотрения биты, часто встречающиеся в бинарных кодировках пары символов, можно применить для создания нейрона, разрешающего конфликты. Формально этот нейрон представляет пару $((S_1, \delta_{\rho_{w(2)},S_1}), (S_2, \delta_{\rho_{w(1)},S_2}))$, где S_1 – множество, хранящее шаблоны первого символа, S_2 – второго, а $\delta_{\rho_{w(2)},S_1}$ – функция нахождения расстояния от точки до множества S_1, взвешенная относительно точек множества S_2 и, аналогично для $\delta_{\rho_{w(1)},S_2}$.

После соответствующего обучения нейрон, разрешающий конфликты, получив на вход точку x, вернёт первый символ, если $\delta_{\rho_{w(2)},S_1}(x) < \delta_{\rho_{w(1)},S_2}(x)$и второй символ в противном случае.

Полное исключение часто встречающихся битов в обоих множествах – возможность не достаточно гибкая. Однако можно расширить область значений функцииw с $\{0,1\}$ до множества $\{1,\dots,2^n\}$.Это позволит более точно учитывать статистику встречаемости битов в бинарных кодировках символов.

Из определения расстояния Хэмминга, а также нашего определения расстояния от точки до множества, очевидно, что точкам, хорошо характеризующим символ нужно придавать вес меньше, чем остальным точкам, так как тогда расстояние от похожей бинарной кодировки до множества будет меньше, чем расстояние от сильно отличающейся кодировки.

Подведём итог: нейрон, разрешающий конфликты, более гибкая конструкция, чем обычный нейрон, которая требует предварительного анализа битовых кодировок для вычисления значений весов тех или иных битов. Чем лучше бит характеризует символ, тем меньше его вес. Бит тем лучше характеризует символ, чем чаще он встречается в битовых кодировках точек обучающего множества для данного символа и тем хуже, – чем чаще он встречается в кодировках обучающего множества для второго символа.

4.3 Матричная схема

Имея в распоряжении нейрон, разрешающий конфликты, можно предложить более надежную схему распознавания символов. Пусть мы хотим распознавать символы, пронумерованные от 1 до n. Рассмотрим матрицу из нейронов размерности n×n:

$$N = \left(S_i, \delta_{\rho_{w(j)}, S_i}\right)_{1 \le i,j \le n,\ i \ne j}.$$

Заметим, что элементы $\left(S_i, \delta_{\rho_{w(j)}, S_i}\right)$ и $\left(S_j, \delta_{\rho_{w(i)}, S_j}\right)$ матрицы Nобразуют пару нейронов, являющуюся нейроном, разрешающим конфликты между символами с номерами i, j при $1 \le i, j \le n,\ i \ne j$. Обучение сети, устроенной таким образом, нужно проводить посредствам обучения всех этих пар в соответствии с рассуждениями, приведенными в предыдущем разделе.

Распознавание символов при описанном выше устройстве сети будет проходить путем опроса нейронов, разрешающих конфликты. Каждый символ при этом, можетполучить от 0 до $n - 1$ положительных срабатываний нейронов. Сеть будет считать ответом тот символ, который наберёт больше всего срабатываний нейронов, разрешающих конфликты.

4.4 Динамическая схема

Главным недостатком матричной схемы является количество нейронов, нужное для её построения. Если для простейшей схемы нужно n нейронов, то для матричной схемы нужно $n * (n - 1)$нейронов. Поэтому в сфере требовательности БДР к вычислительным ресурсам, матричная схема является не пригодной для использования.

Обобщая простейшую и матричную схему, можно получить подход к построению сети более рациональный, с точки зрения вычислительных ресурсов, и более надежный, с точки зрения качества распознавания. Идея такого подхода заимствована из распространенной гипотезы о том, что ошибки при решении задач, сделанные биологическими нейронными сетями, приводят к образованию новых нейронов, а также новых связей между нейронами.

Будем проводить построение и обучение сети динамически. На первом шаге возьмем простейшую схему и обучим её.Затем возьмем всё обучающее множество и попросим сеть распознать это множество, игнорируя то, что обучающие шаблоны известны сети. Получив результаты, выявим пару символов, которые сеть путает между собой чаще всего, и для этой пары создадим и обучим нейрон, разрешающий конфликты.Теперь сеть, в случае принятия решения о том, что ответ – один символ из ранее выявленной пары, не будет выдавать этот символ на выход, а будет просить новый нейрон решить, какой же символ (предполагаемый нижним слоем, или чаще всего с ним конфликтующий) на самом деле считать ответом.

Все последующие шаги, проводятся аналогично. Берем текущую сеть, считая, её простейшей, анализируем результаты распознавания на обучающей выборке, добавляем новый слой из одного нейрона, разрешающего конфликты между самыми схожими, по

мнению текущей сети, символами. Повторяем процесс итеративно. Сеть в результате будет состоять из меньшего количества нейронов, чем матричная, потому что многие символы между собой совершенно не похожи, а результативность останется той же, что и у матричной схемы.

5. Программная реализация

5.1 Текущая реализация библиотеки BddNeuron

Текущая реализация библиотеки *BddNeuron* изменилась, в сравнении с описанием в работе [4]. В её основе лежит класс *Neuron*, описывающий искусственный нейрон. Он содержит множество “хороших” шаблонов (*обучающее множеств*о в терминах ИНС), представленное классом *Pattern*.Класс *Neuron* разделился на два класса, за счет добавления динамического способа построения функции, вычисляющей расстояние от точки до множества. Этот класс теперь имеет два потомка *StaticNeuron*и *DynamicNeuron*, определяющие способ работы с функцией нахождения расстояния.*StaticNeuron*работает с функцией, представленной классом *FunctionOfDistance*, которая строится в соответствии с подходом, описанным в разделе 3.2. *DynamicNeuron* работает с функцией, представленной классом *DynamicFunctionOfDistance*, которая строится в соответствии с подходом, описанным в разделе 3.3.Ниже представлена диаграмма классов библиотеки:

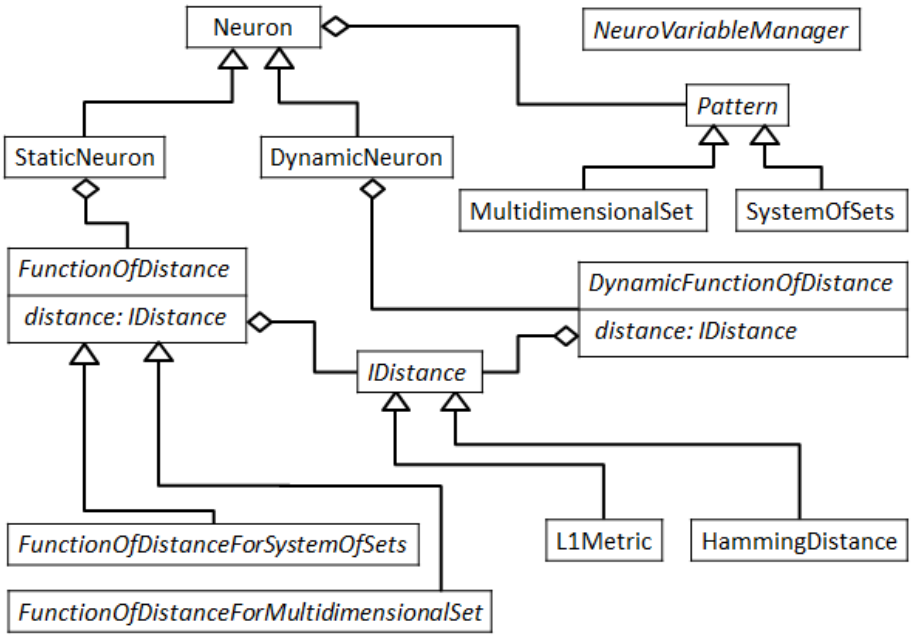


Диаграмма классов библиотеки BddNeuron.

Абстрактный класс *Pattern* предоставляет общий интерфейс для работы с многомерным множеством. Он имеет два класса-наследника, каждый из которых опирается в своей реализации на класс *BDDFunctions::Set*:

- *SystemOfSets* – хранит множество в виде nодномерных независимых множествSet, где n – размерность исходного множества;
- *MultidimensionalSet* – хранит множество ввиде многомерного множества Set.

Расстояние высчитывается одинаково независимо от представления множества. При первом подходе учитываются лишь те закономерности в данных, которые есть в сечениях множества, так как при этом теряются связи между переменными, представляющими данные, и множество вырождается в гиперкуб. Это положительно сказывается как на скорости построения БДР функции, вычисляющей расстояние, так и на размере БДР.

Однако, при втором подходе учитываются закономерности в данных в совокупности, что позволяет строить функцию, вычисляющую расстояние более эффективно, однако на её построение уходит больше времени и памяти, чем при первом подходе.

Абстрактный класс *FunctionOfDistance*предоставляет общий интерфейс для построения и использования функции, определяющей расстояние до множества шаблонов. Ввиду наличия двух реализаций представления множеств, этот класс имеет два абстрактных класса-наследника:

- *FunctionOfDistanceForSystemOfSets* – базовый класс для построения функции, вычисляющей расстояние от точки до множества, представленного классом *SystemOfSets*;
- *FunctionofDistanceForMultidimensionalSet* – базовый класс для построения функции, вычисляющей расстояние от точки до множества, представленного классом *MultidimensionalSet*.

Класс *DynamicFunctionOfDistance* отличается от класса *FunctionOfDistance* способом построения функции вычисляющей, расстояние от точки до множества и независимостью от представления множества.

Кроме того работа с конкретными метриками стала более гибкой, за счет добавления интерфейса *IDistance*, позволяющего абстрагироваться от того, как мы считаем функцию нахождения расстояния и от представления множества, которое мы хотим использовать. Теперь для добавления новой метрики достаточно отнаследоваться от этого класса и реализовать логику расчета метрики между двумя точками. Таким способом были реализованы классы, позволяющие работать сL_1- метрикой и расстоянием Хэмминга. Это легко можно видеть на диаграмме классов библиотеки.

Для уменьшения затрат по памяти, был также реализован менеджер переменных *NeuroVariableManager*. Он позволяет переиспользоватьодни и те же переменные (в терминах библиотеки *BddFunctions*) в разных нейронах. Соответственно теперь перед началом работы с библиотекой производится анализ количества переменных, необходимых для построения нужного пользователю количества нейронов. Однако если пользователь захочет динамически завести нейрон, которому нужно больше переменных, чем есть в пуле, у него этого сделать не получится. Поэтому при работе с библиотекой в случае необходимости динамического создания нейронов имеет смысл заводить столько нейронов, сколько максимально может потребоваться сразу. Сам нейрон занимает немного памяти и создается достаточно быстро, поэтому это ограничение нет смысла считать серьезным.

В работе [4] подробно описаны этапы работы с библиотекой в случае статического построения функции вычисления расстояния от точки до множества. Здесь описание этих этапов приводиться не будет, так как статический подход оказался не рентабельным. Организация же работы с нейроном в случае динамического построения функции расстояния гораздо проще. Опишем в общем случае этапы работы с библиотекой для динамических нейронов:

Создание нейронов – создавать нужно сразу все нейроны, которые могут понадобиться при работе. При этом не создается никаких низкоуровневых объектов библиотеки *BddFunctions*. Каждый нейрон регистрируется в менеджере переменных.

Инициализация низкоуровневых переменных – менеджер переменных пробегается по всем созданным нейронам, узнает максимальное количество переменных каждого типа, которое могут понадобится одному нейрону, заводит все нужные переменные, инициализирует библиотеку *BddFunctions* и отдает каждому нейрону ссылки на переменные, которые ему нужны. Затем производится ленивая инициализация нейронов, в которую входят создание множества, функции расстояния, возвращающей ноль, а так же некоторых постоянных функций.

Обучение и использование – включает в себя добавление, удаление точек в обучающее множество, с сопутствующей перестройкой функции вычисления расстояния. С увеличением размера множества время, необходимое для перестройки функции расстояния, увеличивается, но в целом определяется по большей части длинной битовых кодировок и сложностью используемой метрики. В любой момент работы с нейроном, можно попросить рассчитать расстояние до актуального на данный момент множества шаблонов.

Помимо прочего, для удобства работы с библиотекой была создана Java-обертка *BddNeuronJ*.

5.2 Библиотека NeuroLab для построения ИНС на основе БДР

Библиотека *NeuroLab* построена на основе Java-обертки *BddNeuronJ*, позволяющей работать с нейроном на основе БДР. Библиотека создана с целью упрощения создания схем ИНС на основе БДР, а также сбора статистики распознавания символов созданными схемами. Здесь мы не будем приводить полную диаграмму классов библиотеки, ограничимся словесным описанием основных классов библиотеки.

ИНС в терминах библиотеки *NeuroLab* состоит из блоков, представленных классом *Block*. Для каждого доступного типа блоков создается класс – наследник класса *Block*. На данный момент в библиотеки доступны следующие блоки: входной, выходной, агрегирующий, представляющий нейрон, представляющий нейрон, разрешающий конфликты.

Основной класс библиотеки – *NeuroNet* – хранит все блоки, которые есть в данной схеме сети, отвечает за связи между блоками. Объект этого класса можно получить из XML описания, посредствам класса *NeuroNetXmlModel*. Такая возможность удобна для статических схем ИНС. В случае динамической схемы, статическую часть можно получить также из XML описания, а затем достраивать сеть, в соответствии с заложенным алгоритмом обучения. Перед обучением с помощью класса *DataManager*в библиотеку загружаются предварительно подготовленные шаблоны хороших точек для каждого нейрона. Обмен сигналами между блоками сети осуществляется с помощью класса *SignalManager*. Для алгоритмов, требующих динамического изменения схемы сети в

процессе обучения, есть возможность резервирования заданного количества нейронов перед обучением сети с помощью класса *NativeNeuronPool*.

Обучение сети может состоять из двух этапов. Первый этап – обучение статических нейронов. В случае динамического построения схемы есть также второй этап обучения, на котором производится тестирование сети, добавление нейронов, требуемых алгоритмом, и их соответствующее обучение.

После обучения сети, можно её тестировать. Тестирование состоит из чтения файла, хранящего тестовые точки, и оценки сетью каждой из тестовых точек. Ответы сети заносятся в модуль статистики *Statistic*. После тестирования на всех точках, модуль статистики формирует отчет о результатах распознавания.

Заметим, что для работы с нативной реализацией нейрона на основе БДР, не обязательно строить полноценную сеть, с помощью класса *NeuroNet*. Иногда удобней для проверки тех или иных гипотез, воспользовавшись классом *BddNeuron*, завести необходимое количество нейронов и напрямую работать с ними, самостоятельно организуя обучение и тестирование.

6. Тестирование

При тестировании использовалось представление множества шаблонов в виде системы множеств и функция нахождения расстояния от точки до множества, построенная на основе расстояния Хэмминга. Проводилось тестирование на процессоре с тактовой частотой 3,4 ГГц.

6.1 Распознавание печатных цифр и букв

На данный момент большая часть экспериментов проводилась на распознавании печатных цифр и букв. Для этого были сгенерированы описания шаблонных и тестовых точек в форме понятной библиотеке *NeuroLab*.

Для каждого печатного символа генерировалось черно-белое изображение размера 16×16 пикселей. Сопоставляя белым пикселям ноль, а черным единицу, получаем, что каждое такое изображение можно закодировать восемью 32-битными числами. То есть точка (шаблонная или тестовая) в данном случае – это вектор из восьми 32-битных чисел.

Из стандартных печатных шрифтов, установленных на машине (их получилось 210, за исключением слишком специфических), было сгенерировано тестовое множество. Обучающее множество выбиралось из тех же 210 шрифтов. Тестирование качества распознавания проводилось так: указываем размерность обучающего множества n, для каждого из распознаваемых символов формируем обучающее множество, выбирая n точек из соответствующего тестового множества, обучаем сеть, прогоняем всё тестовое множество через сеть. Ниже представлены результаты тестирования простейшей схемы (см. часть 4.1) при описанном подходе:

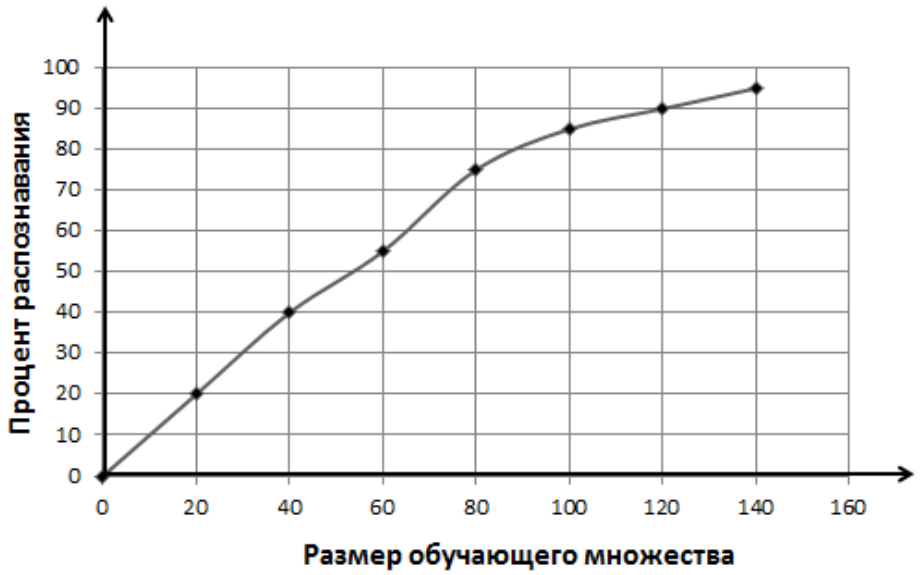


График зависимости процента успешных тестов
от размера обучающего множества.

Как видно, при увеличении количества точек для обучения, результаты распознавания улучшаются.

Заметим, что на обучение 36 нейронов (10 цифр и 26 букв) при 60 обучающих точках на один нейрон уходит порядка 20 минут, а при 140 точках – уже порядка 9 часов. При этом, несмотря на неутешительную статистику по росту времени обучения, при росте количества обучающих точек, стоит отметить, что среднее время на распознавание одного символа не меняется и составляет порядка 10-15мс.

Согласно принципу работы простейшей схемы, список результирующих расстояний от тестовой точки до нейронов, представляющих каждый из возможных символов, упорядочивается по возрастанию, и ответом считается тот символ, расстояние до нейрона которого, оказалось минимальным, т. е. попало в начало списка. Отметим, что независимо от размера обучающего множества в случае, когда сеть ошибается, считая, что расстояние до ошибочного символа меньше, чем расстояние до верного символа, расстояние до верного символа всё равно оказывается близко к началу списка, зачастую оказываясь вторым.

Последнее наблюдение как раз привело к идее создания и применения нейрона, разрешающего конфликты, описанного в части 4.2.

6.2 Распознавание рукописных цифр

Длятестирования распознавания рукописных символов была использована база MNIST [11]. Изображения символов в этой базе достаточно велики: 28×28 пикселей. При таких размерах пришлось бы работать с представлением точки во множестве, как с вектором из 32-битных чисел. Для использования БДР размерность данных имеет очень весомое значение, поэтому был применен метод перцептивногохеширования[2], для уменьшения размерности множеств шаблонов и, соответственно, ресурсоёмкости ИНС.

Алгоритм метода таков:

- Берём чёрно-белое изображение цифры и масштабируем его до размеров 14× 14 пикселей;
- Если пиксель белый, сопоставляем ему ноль, иначе единицу, тем самым получаем 196 битов;
- Получаем всего семь 32-битных чисел, необходимых для представления одной точки.

При таком представлении элемент множества (хеш) будет состоять из 7 байт вместо 25,что гораздо более приемлемо для БДР.

В отличие от обучающих множеств, построенных для тестирования распознавания на печатных символах, обучающие множества для рукописных символов не имеют пересечений с тестовыми множествами благодаря объемности базы MNIST.В этой базе для каждого символа присутствует порядка 6000 изображений для обучения и порядка 1000 изображений для распознавания. Соответственно тестовое множество рассматривалось полностью, обучение же проводилось пока только на множествах размерностью от 200 до 1000 точек для каждого символа. Результаты распознавания для простейшей схемы при этом получились от, примерно, 50% до 63,86% на больших

изображениях символов. Лучший результат для маленьких (хешированных) изображений – 61,43% на 1000 точках для обучения.

Полноценные тесты динамической схемы распознавания на данный момент пока не проводились, проверялась лишь правильность логики работы при добавлении нейронов разрешающих конфликты. Кроме того проводились отдельные тесты нейрона разрешающего конфликты. Для этого была выбрана пара цифр, которые путались чаще всего. Определение весов битов при обучении нейрона проводилось следующим образом:

- Обучаем на выбранных 2nточках пару нейронов (nизображений каждого символа)
- Прогоняем все 2nточек через нейроны, определяя на каких точках нейроны ошибаются
- Обучая нейрон, разрешающий конфликты, при добавлении точки, смотрим на все точки сопряженного множества и для каждого бита сравниваем его с соответствующим битом, добавляемой точки. Если биты нулевые (фоновые) – к весу прибавляется 4, если бит, добавляемой точки 1, а анализируемой 0, добавляем 3, если наоборот – 2, если оба бита единицы (значимые), добавляем 1 – самый низкий вес.
- Добавляем точку с накопленным весом, переходим к обработке следующей точки

Тестирование при такой настройке на 10, 20, 50 точках для каждого из символов, сократило количество ошибок с 3 до 0, с 7 до 0, с 10 до 3 соответственно. На большем количестве точек тестирование пока не проводилось.

7. Использованные технологии

Библиотека *BddNeuron* реализована на основе библиотеки *BddFunctions*, предоставляющей гибкий объектно-ориентированный C++ интерфейс для работы с функциями, множествами и матрицами, представленными БДР. Всвоюочередьбиблиотека*BDDFunctions*основана на *пакете решающих диаграмм– ColoradoUniversityDecisionDiagramPackage* (CUDD)[6], реализованном на языке C.

CUDD реализует основные алгоритмы БДР, однако предоставляет весьма низкоуровневый интерфейс, которым сложно пользоваться. Библиотека*BDDFunctions*, в свою очередь, решает большинство инфраструктурных проблем, с которыми приходится сталкиваться разработчику при использовании БДР, позволяя работать на более высоком уровне абстракции.

Ключевые понятиями программного интерфейса библиотеки *BDDFunctions*являются *тип данных, переменная, функция, множество, матрица.* Основные алгоритмы при этом, так же как и в CUDD, реализованы на языке C, с целью обеспечения максимальной производительности. С более подробным описанием библиотеки *BDDFunctions* можно ознакомиться в [1].

Библиотека *BddNeuron* разработана на языке GNUC++ с использованием компилятора MinGWи интегрированной среды разработки Code::Blocks[5].

Для покрытия исходного кода тестами использована библиотека с открытым исходным кодом *GoogleC++ TestingFramework* (GTest)[7].GTest позволяет легко проводить модульное тестирование и обладает дружественным интерфейсом.

Библиотека *NeuroLab*реализована на языке Java, с использованием среды разработки NetBeans 6.9.1 [9].

Генерация тестового и обучающего множеств,в случае печатных символов, а также конвертация файлов базы MNIST в понятный библиотеке *NeuroLab*формат, в случае рукописных символов,реализованы посредством языка C#,с использованием среды разработки MicrosoftVisualStudio 2010[8]. Для прозрачности процесса, генератор разработан с использованием технологии WindowsForms.

8. Итоги

В ходе проделанной работы достигнуты следующие результаты:

- Разработан более быстрый подход к построению функции вычисляющей расстояние от точки до множества.
- Повышена гибкость архитектуры библиотеки *BddNeuron*.
- Оптимизированы затраты по памяти при использовании большого количества нейронов
- СозданаJava-обертка над библиотекой *BddNeuron*.
- Разработана библиотека *NeuroLab*, позволяющая строить схемы сетей для распознавания и проводить их тестирование.
- Рассмотрены три схемы сетей на основе БДР.
- Проведены тесты по распознаванию печатных букв и цифр и рукописных цифр.

9. Дальнейшие планы

В ходе анализа полученных результатов была обнаружена необходимость дальнейшей оптимизации скорости обучения нейронов, а также необходимость развития методов обучения нейронов.

С точки зрения оптимизации скорости обучения, может принести плоды эксперимент с порядком низкоуровневых переменных, используемых для построения БДР. Известно, что порядок переменных в значительной степени может повлиять на размер диаграммы [10], что очевидным образом скажется на скорости её перестройки при добавлении точек. В данный момент используется классический “чередующийся” порядок, позволяющий достичь оптимального размера для большинства диаграмм. В нашем случае диаграммы носят весьма специфичный характер за счет стремления свести нахождение расстояния к операциям сложения.

С точки зрения качества распознавания, планируется продолжить исследование в направлении поиска хорошего алгоритма обучения нейронов разрешающих конфликты. Это направление, очевидным образом, зависит от способа нахождения расстояния. Приведем несколько идей, которые потенциально могут привести к успеху:

- Обучающее множество можно кластеризовать, поделив один нейрон на несколько. Действительно, один и тот же символ может иметь несколько способов написания, которые, с точки зрения взаимного расположения битов, могут достаточно сильно различаться, что будет сильно портить расстояние до множества.
- Даже поделив обучающее множество на кластеры, в каждом кластере символы все равно будут в разной мере типичны для данного кластера. Это в свою очередь можно использовать несколькими способами:
  - Можно задавать вес не отдельным битам, а расстоянию до данной точки в целом. Соответственно точки, слишком отличающиеся от остальных, с точки зрения среднего расстояния до прочих точек множества, можно учитывать с сильно понижающим весом или вообще отбрасывать, чтобы не портить общее расстояние.
  - Порог для срабатывания нейрона можно определить как некоторую функцию от максимума среднего расстояния между точками самого обучающего множества.
- Расстояние Хэмминга при всей его простоте и удобстве, достаточно плохо учитывает топологию изображения. Например, сдвинув изображение единички, изображенной одной линией, в сторону, получим достаточно сильное увеличение расстояние Хэмминга. Это приводит к мысли попытаться использовать какую-нибудь более сложную, но более надежную метрику, для определения расстояния.

10. Список использованной литературы

[1] Бугайченко Д.Ю., Соловьев И.П. Библиотека многокорневых решающих диаграмм BddFunctions и её применение. //Системное программирование. Вып. 5: Сб. статей / Под ред. А.Н.Терехова, Д.Ю.Булычева. - СПб.: Изд-во СПбГУ, 2010 г. С. 190-213.

[2] Р. 293–318.АлизарА.“Выглядит похоже”. Как работает перцептивный хэш. URL: http://habrahabr.ru/post/120562/

[3] Искусственные нейронные сети. URL: http://ru.wikipedia.org/wiki/Искусственная_нейронная_сеть.

[4] Зубаревич Д.А. Нахождение расстояния от точки до множества, представленного бинарной диаграммой решений. URL: http://se.math.spbu.ru/SE/YearlyProjects/2012/YearlyProjects/2012/345/345_ZubareZuba_report.pdf

[5] Code::BlocksC++ IDE. URL: http://www.codeblocks.org/.

[6] SomenziF. CUDD: ColoradoUniversityDecisionDiagramPackage. URL: http://vlsi.colorado.edu/ fabio/CUDD.

[7] Google C++ Testing Framework. URL: http://code.google.com/p/googletest/.

[8] MicrosoftVisualStudio 2010. URL: http://www.microsoft.com/visualstudio/ru-ru.

[9] NetBeans IDE URL: https://netbeans.org//.

[10] Bryant R. E. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams // ACM Computing Surveys. 1992. Vol. 24, No. 3.

[11] The MNIST database of handwriting digits URL: http://yann.lecun.com/exdb/mnist/